Mostly Concurrent Garbage Collection

Ausgewählte Kapitel der Systemsoftware

2021-02-02

Marco Ammon

Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme und Betriebssysteme

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

A Simple Server Program...

```c
int socket = ...;

// Accept and handle connection after connection
while (true) {
    int connection = accept(socket, NULL, NULL);
    // do stuff
    struct task *currentTask
        = malloc(sizeof(struct task));
    // do more stuff
    if (someError)
        continue;
    // do even more stuff
    // we have done ALL the stuff and can clean up
    free(currentTask);
    close(connection);
}
```

…May Not Be So Simple after All

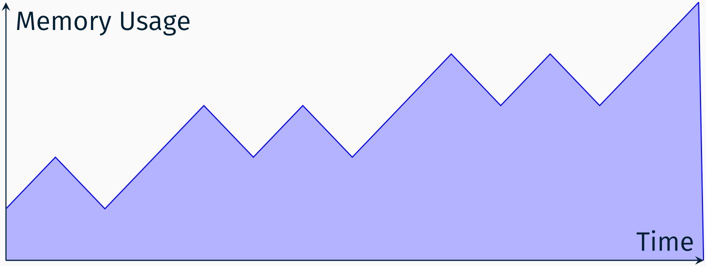


Figure 1: Memory Usage over Time

...May Not Be So Simple after All

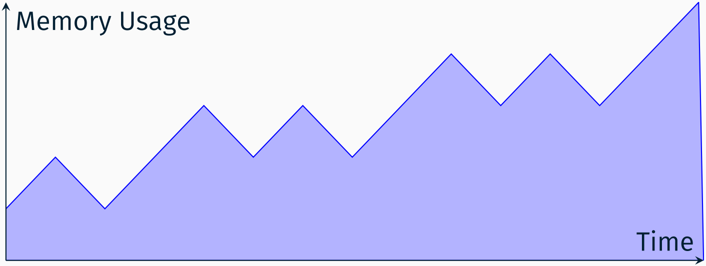


Figure 1: Memory Usage over Time

Problem: Out of Memory

forgotten `free()` call → memory leak

More Memory Management, More Problems

```c
struct list *list = ... // [1, 2, -3, 0, 1]
struct list *min = list;
for (struct list *curr = list; curr->next != NULL;
 ↪  curr = curr->next) {
    if (curr->value < min->value) min = curr;
    free(curr); // We only traverse the list once,
     ↪  so let's get rid of unused objects
}
// Add an element to another list
struct list *list2 = malloc(sizeof(struct list));
list2->value = 3;
// Everything is still alright. Right? Right?!
assert(min->value == -3);
```

More Memory Management, More Problems

```
struct list *list = ... // [1, 2, -3, 0, 1]
struct list *min = list;
for (struct list *curr = list; curr->next != NULL;
 ↪  curr = curr->next) {
    if (curr->value < min->value) min = curr;
    free(curr); // We only traverse the list once,
     ↪  so let's get rid of unused objects
}
// Add an element to another list
struct list *list2 = malloc(sizeof(struct list));
list2->value = 3;
// Everything is still alright. Right? Right?!
assert(min->value == -3);
```

Problem: *Maybe* Corrupted Data

`free()` call on in-use object → subsequent allocation may re-use heap space

Manual memory management is *hard!*

Manual memory management is *hard!*

Better automatic, but needs to be *correct* and *fast*

Table of Contents

Automatic Memory Management with Tracing Garbage Collectors

Tracing Garbage Collection

Idea

- known set of in-use objects and references (*root set*)
- recursively follow pointers and remember visited objects
- only unreachable objects remain unvisited

Tracing Garbage Collection

Idea

- known set of in-use objects and references (*root set*)
- recursively follow pointers and remember visited objects
- only unreachable objects remain unvisited

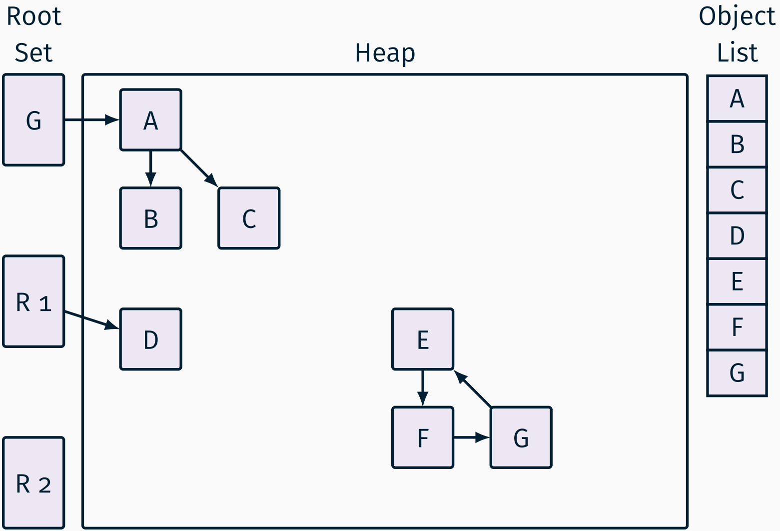
Requirements

- determination of root set
- pointer identification
- integration with memory allocator

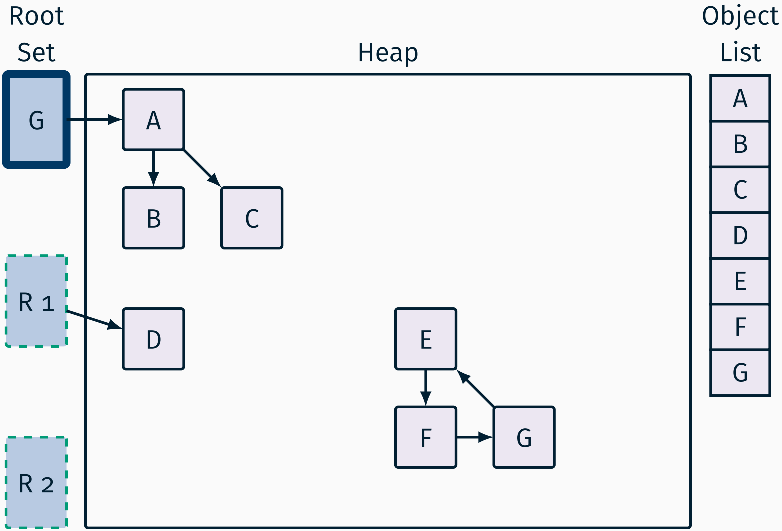
Simple Idea: Mark-and-Sweep Garbage Collection

1. Stop the application (“stop the world” [STW])
2. Trace from root set (*mark* phase)
3. Collect unmarked objects (*sweep* phase)
4. Reset all marks
5. Resume the application

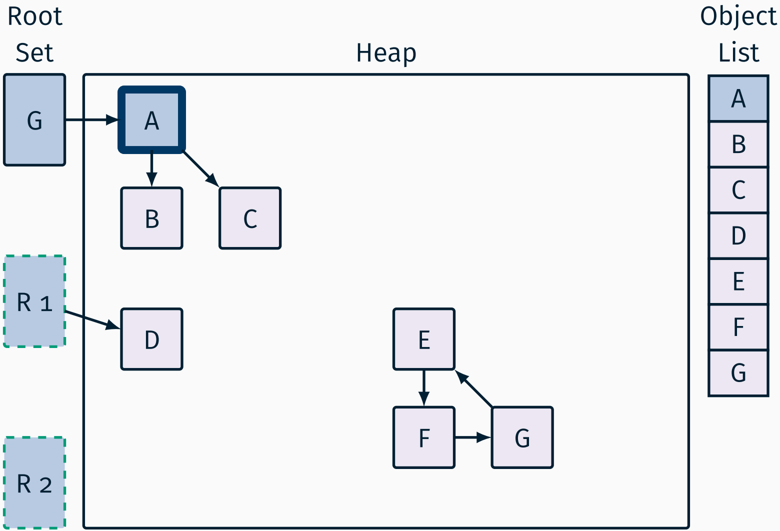
Mark-and-Sweep GC by Example: Heap State at GC Invocation



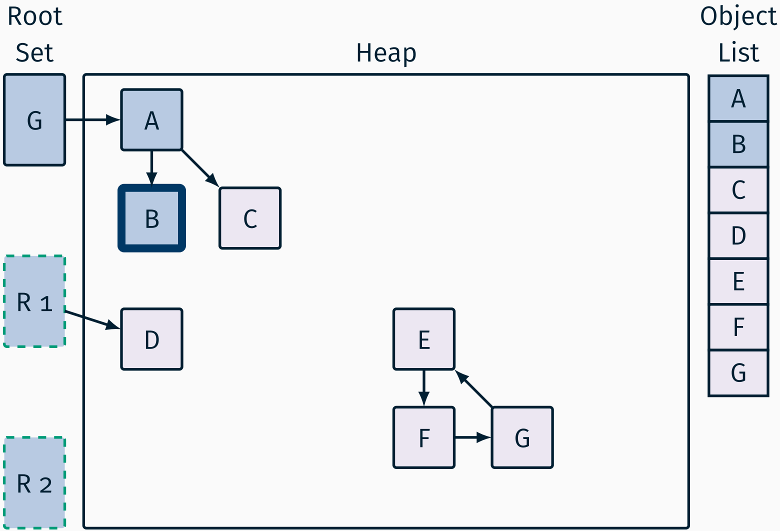
Mark-and-Sweep GC by Example: Mark Phase



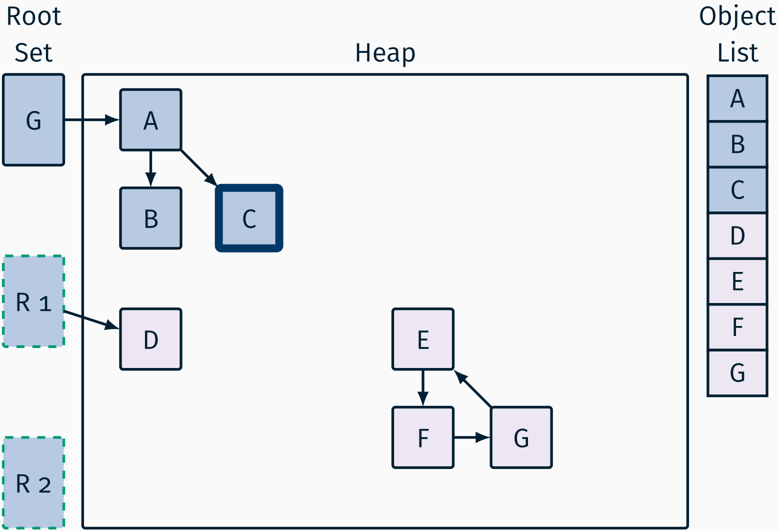
Mark-and-Sweep GC by Example: Mark Phase



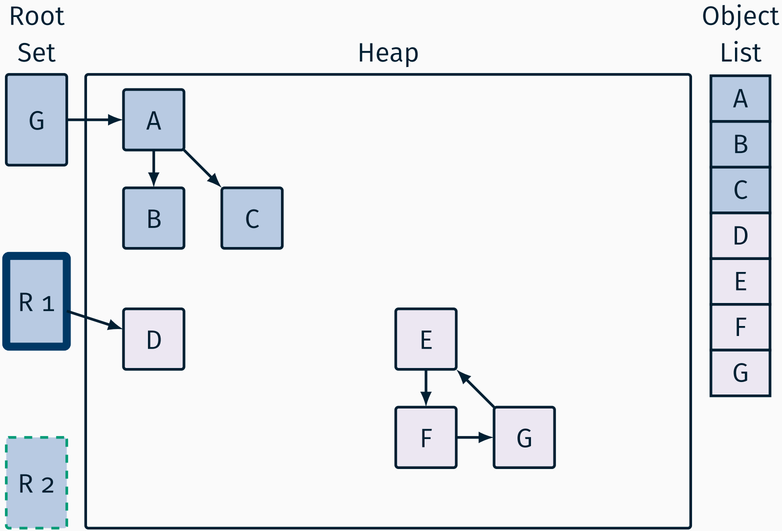
Mark-and-Sweep GC by Example: Mark Phase



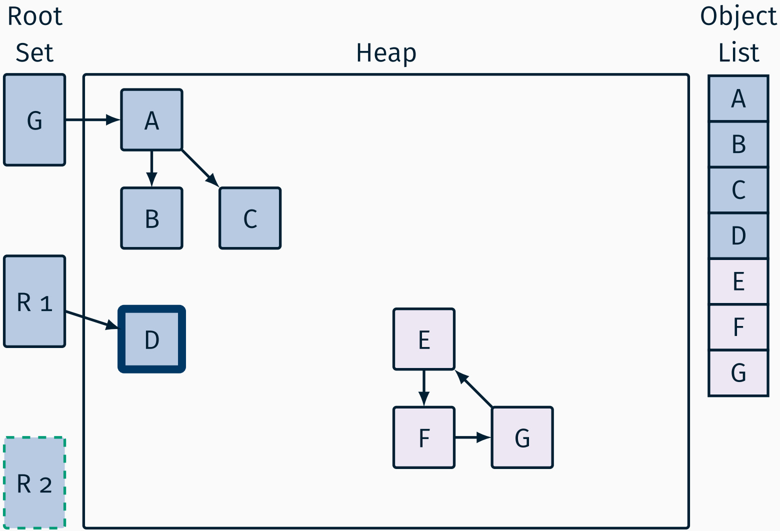
Mark-and-Sweep GC by Example: Mark Phase



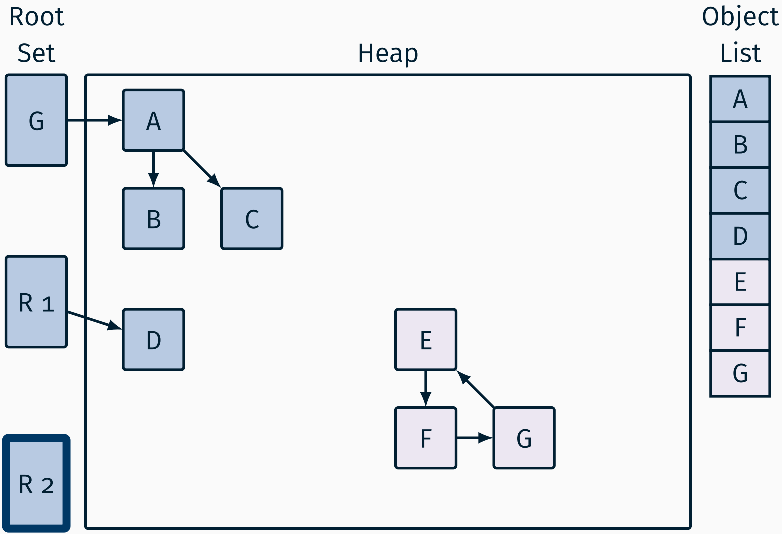
Mark-and-Sweep GC by Example: Mark Phase



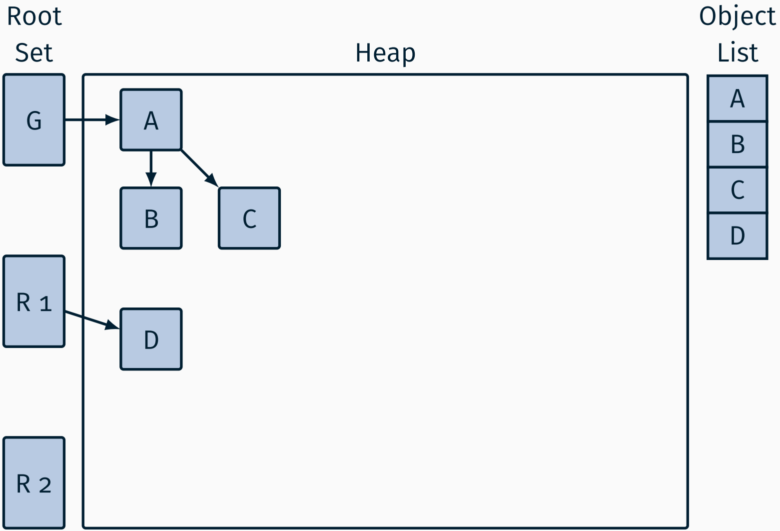
Mark-and-Sweep GC by Example: Mark Phase



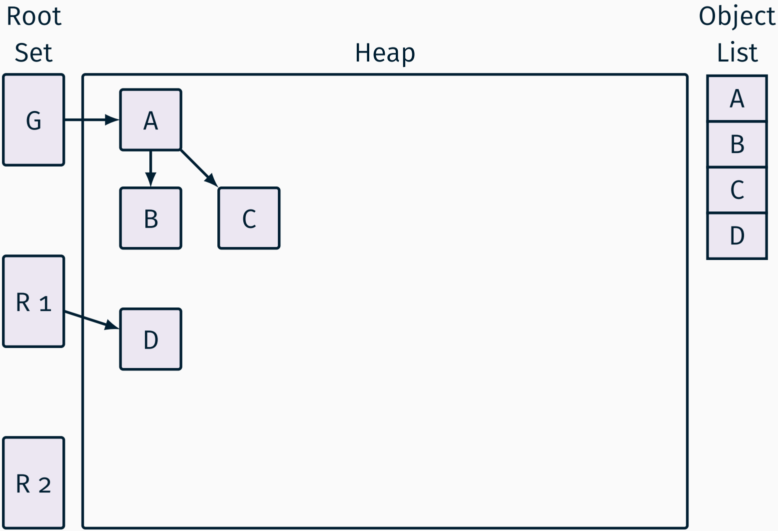
Mark-and-Sweep GC by Example: Mark Phase



Mark-and-Sweep GC by Example: Sweep Phase



Mark-and-Sweep GC by Example: Reset Mark Bits



Generational Garbage Collection

Insight: Young objects are more likely to die!

Treat young and old objects differently:

- collect young generation often
- collect old generation less frequently

Generational Garbage Collection

Insight: Young objects are more likely to die!

Treat young and old objects differently:

- collect young generation often
- collect old generation less frequently

Important

Pointers from old generation into young generation must be tracked!

There Is No Free Lunch!

There Is No Free Lunch!

Problems

- both schemes require to stop the world

  $\rightarrow$ long pauses
- intolerable for GUI applications and web services

“Mostly-Parallel Garbage Collection” by Boehm et al.

Mostly Concurrent Garbage Collection

Grand Idea

Run GC *concurrently* with application (*mutator*) for shorter pauses

Mostly Concurrent Garbage Collection

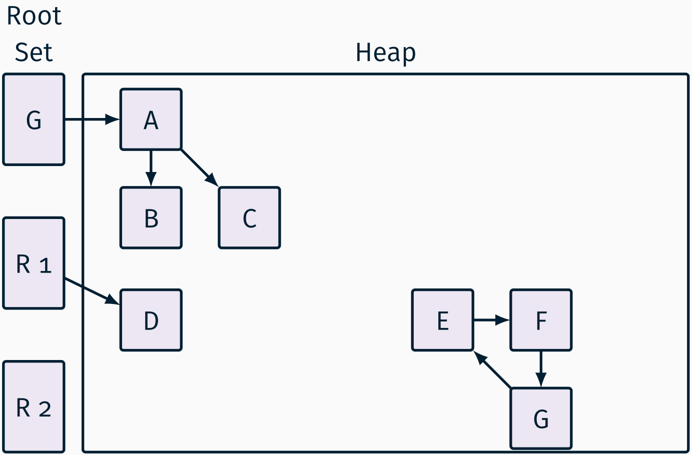
Grand Idea

Run GC *concurrently* with application (*mutator*) for shorter pauses

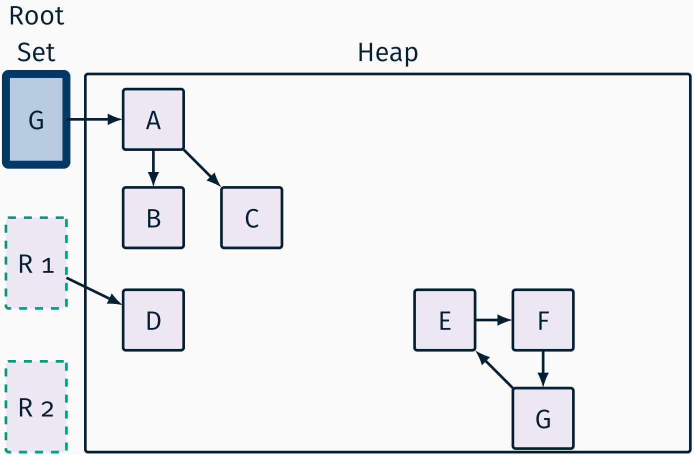
Problem

Mutator changes objects while GC traverses heap!

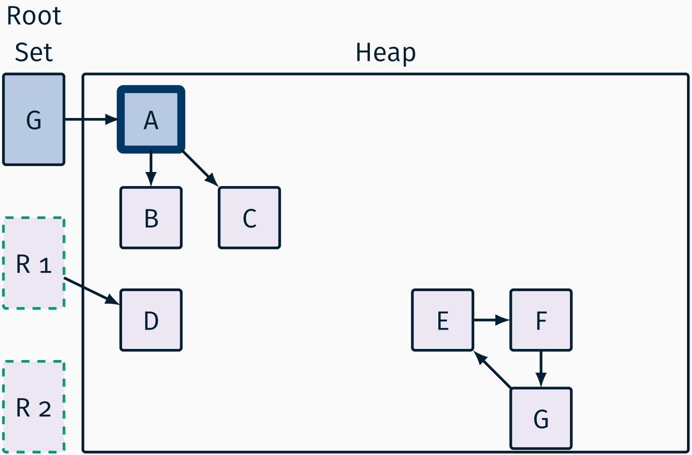
Problem of the Lost Objects: GC Invocation



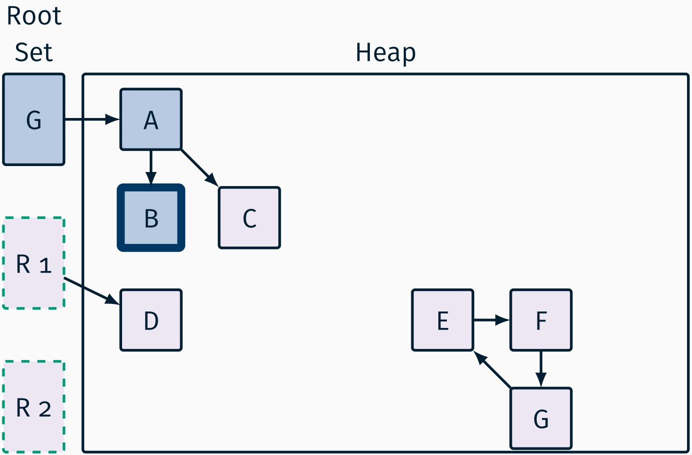
Problem of the Lost Objects: Marking



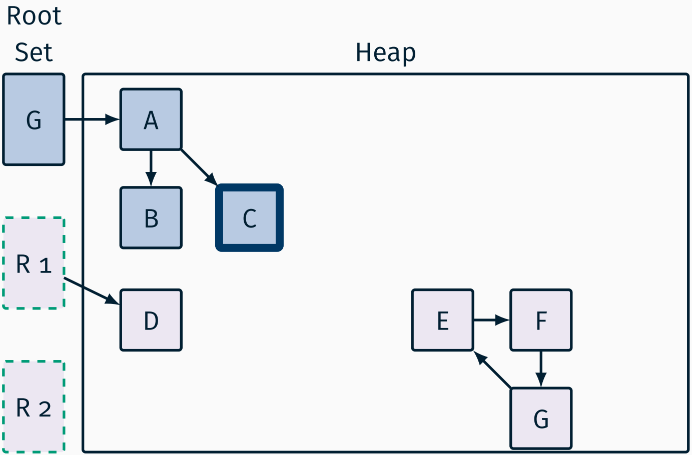
Problem of the Lost Objects: Marking



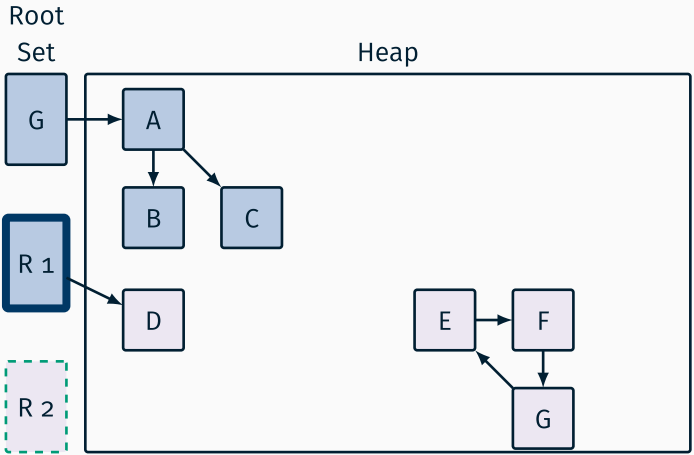
Problem of the Lost Objects: Marking



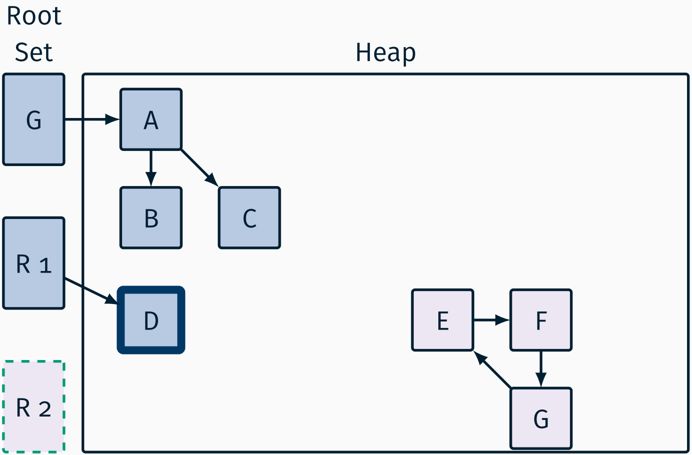
Problem of the Lost Objects: Marking



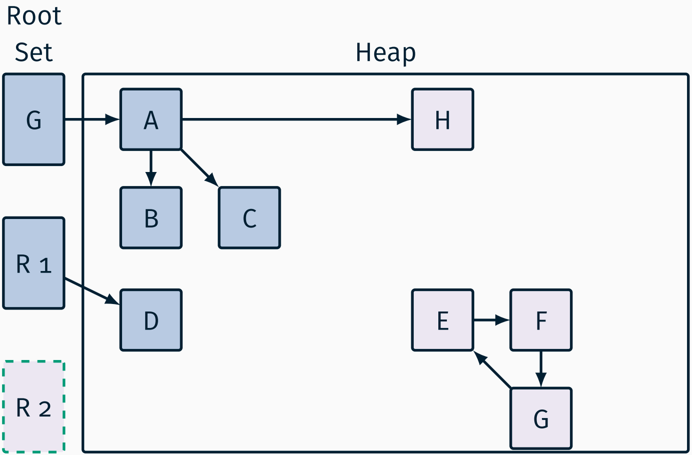
Problem of the Lost Objects: Marking



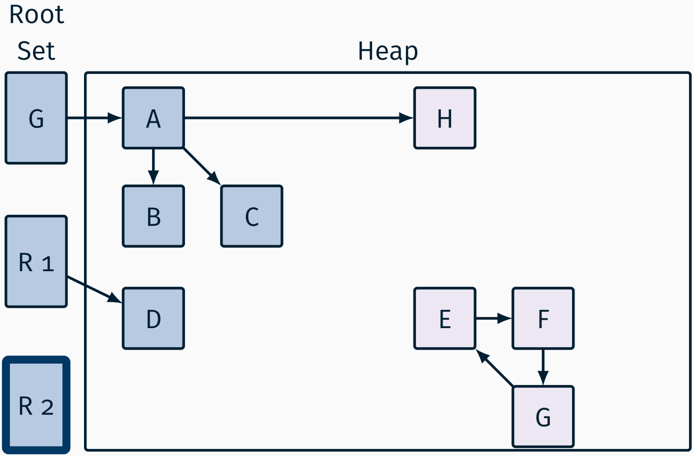
Problem of the Lost Objects: Marking



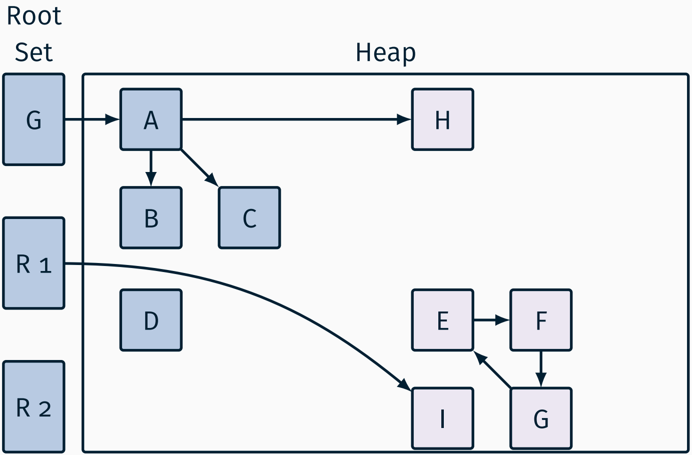
Problem of the Lost Objects: Program Allocates New Object



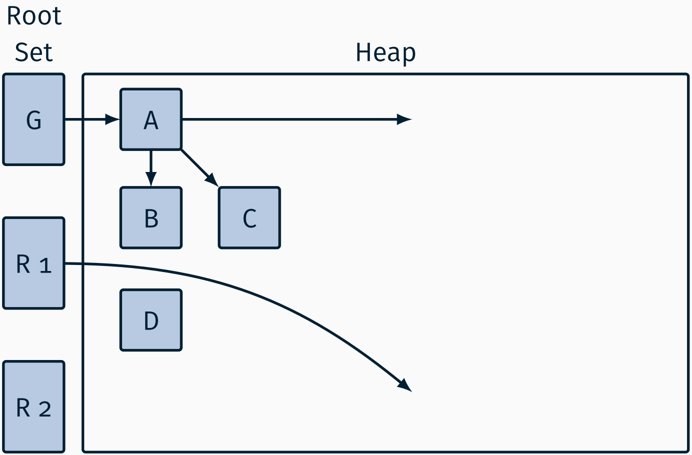
Problem of the Lost Objects: Marking Complete



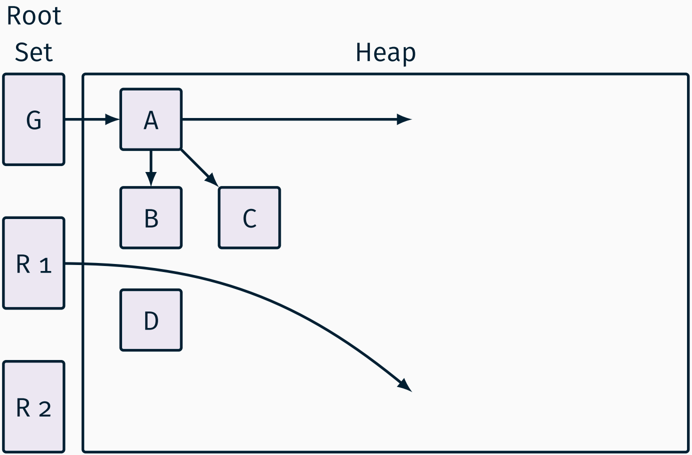
Problem of the Lost Objects: Program Allocates New Object



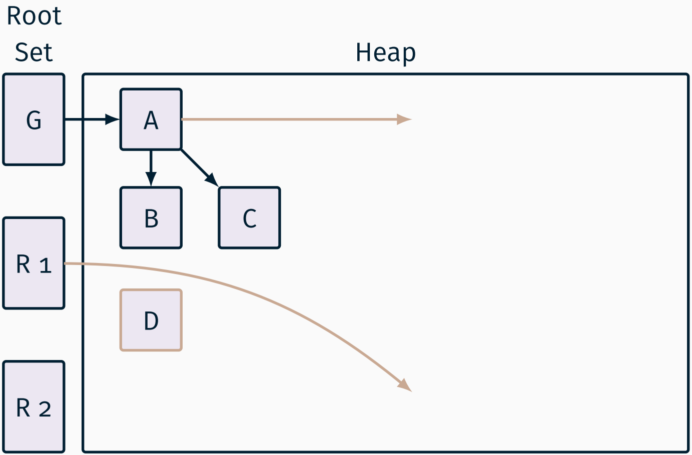
Problem of the Lost Objects: Sweep of Unmarked Objects



Problem of the Lost Objects: GC Cycle Complete



Problem of the Lost Objects: GC Cycle Complete



Problems

two still-needed objects removed, unneeded object retained

Mostly Concurrent Garbage Collection

Grand Idea

Run GC *concurrently* with application (*mutator*) for shorter pauses

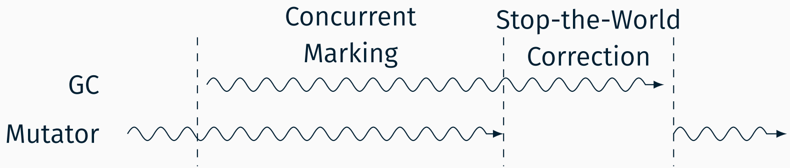
Problem

Mutator changes objects while GC traverses heap!

Solution: Synchronisation

→ mark in parallel with mutator and record all writes to objects

→ short stop-the-world correction phase



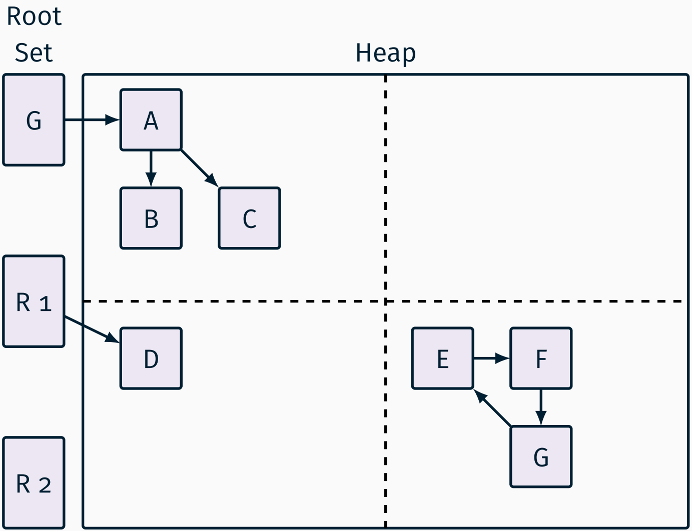
Proposed Algorithm

- Use MMU for recording writes:
  - Dirty page bits indicate writes with page-size granularity
  - if clean, traversed graph is correct: no references from marked to unmarked objects could have been added
  - if dirty, rescan the page: marking from marked objects suffices
- generational collection: marked objects are *old*
- conservative: works even without explicit compiler assistance

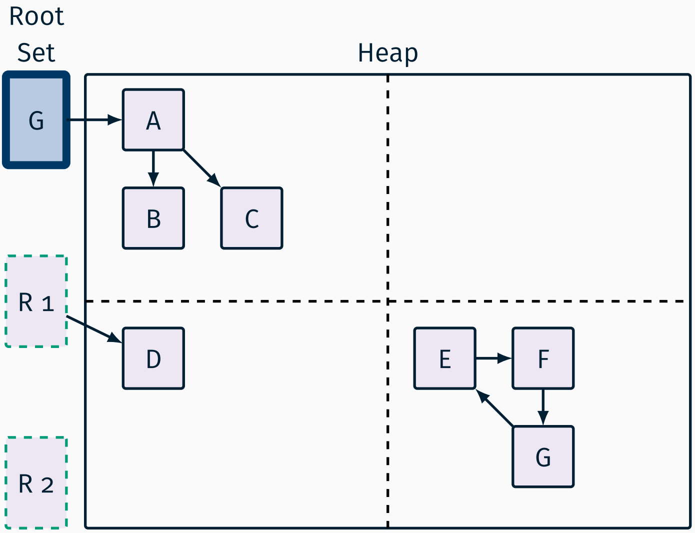
Full Collection

1. Clear all mark and dirty bits
2. Mark all objects in the root set and recursively trace from them
3. Stop the world
4. Trace from registers and all marked objects on dirty pages
5. Clear dirty bits and restart the world

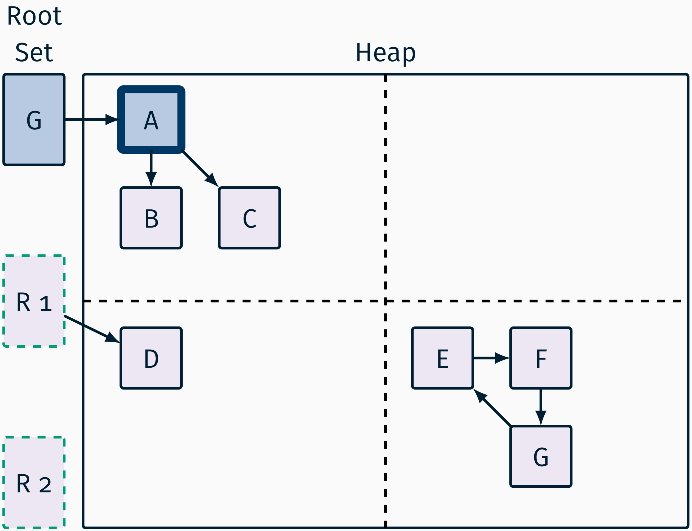
Full Collection: GC Invocation



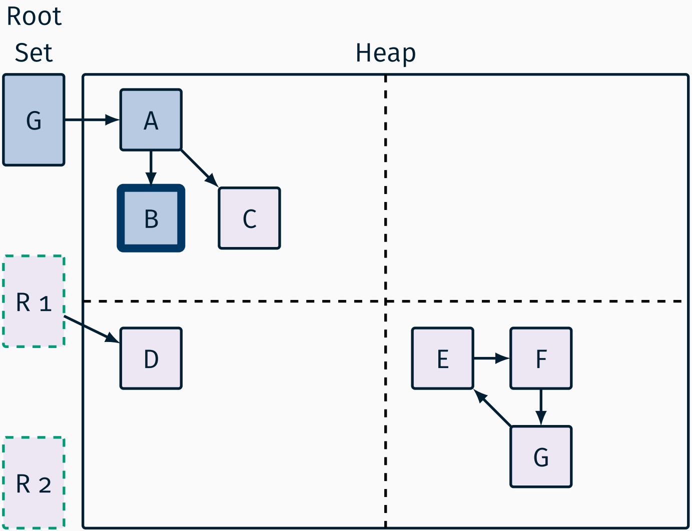
Concurrent Marking



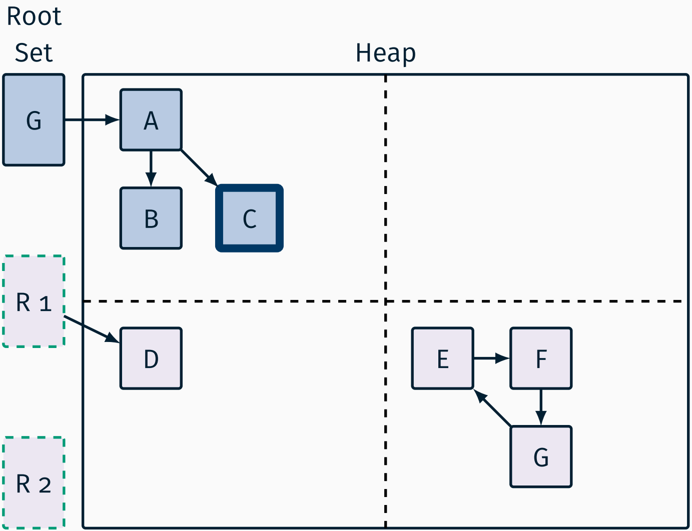
Concurrent Marking



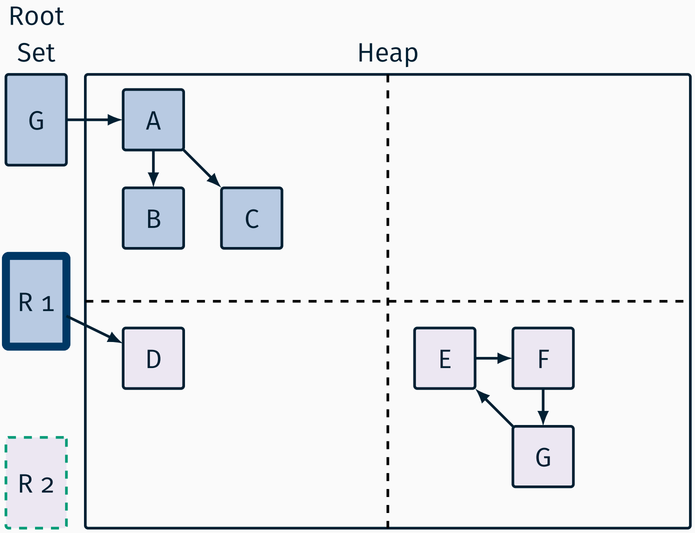
Concurrent Marking



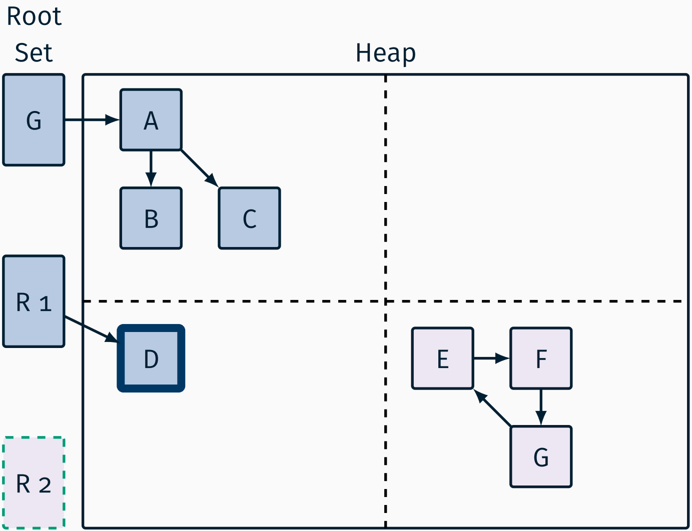
Concurrent Marking



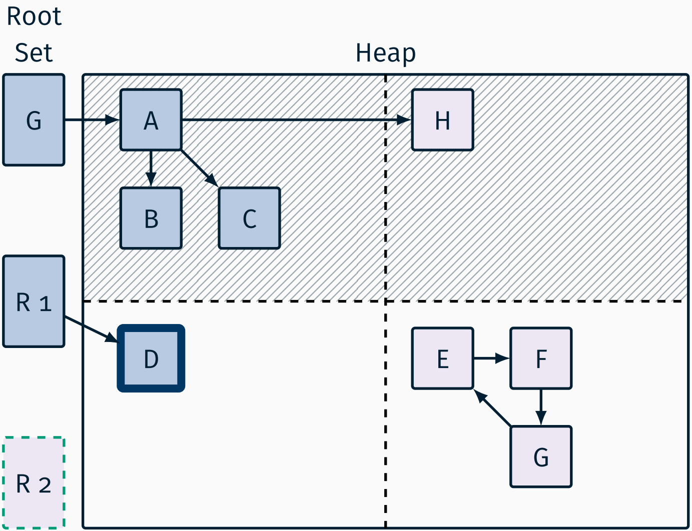
Concurrent Marking



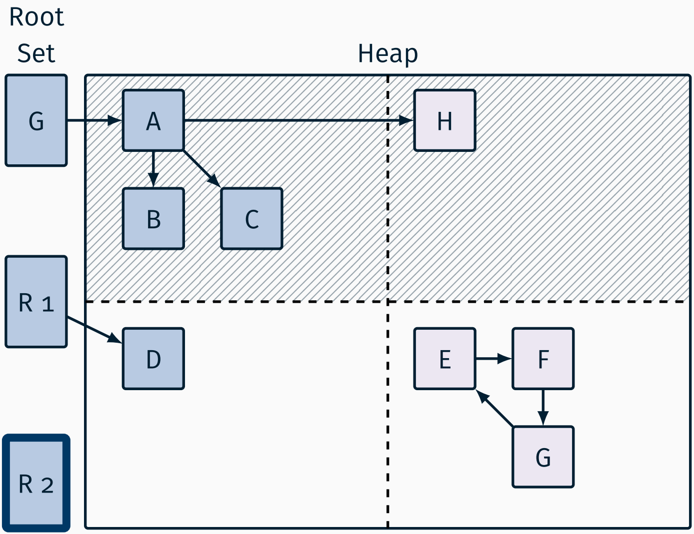
Concurrent Marking



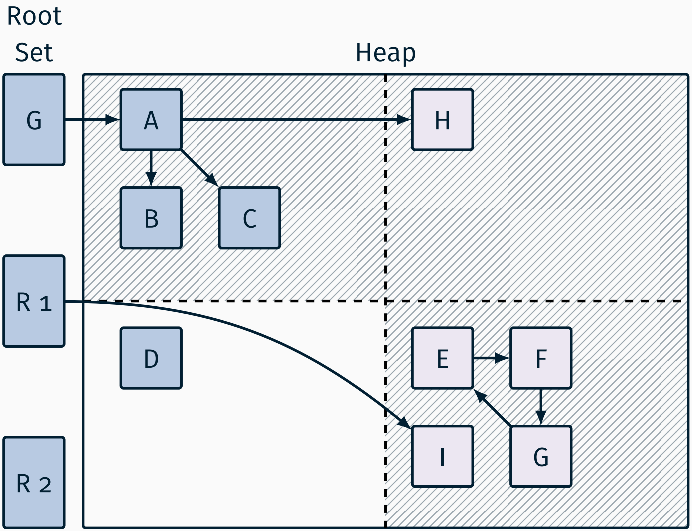
Program Allocates New Object



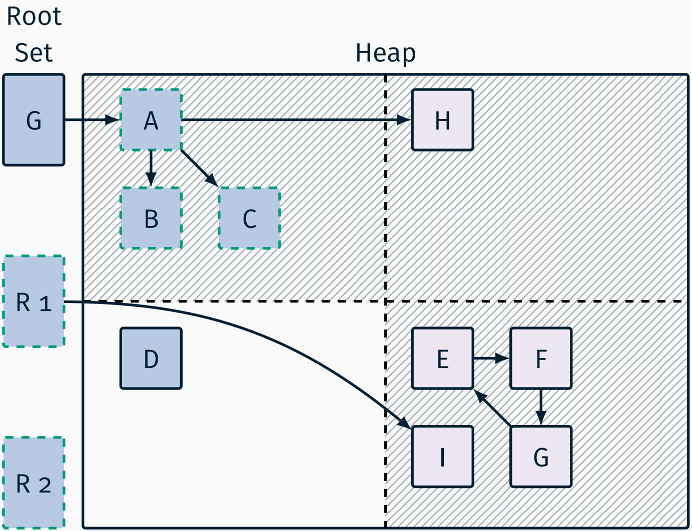
Concurrent Marking Complete



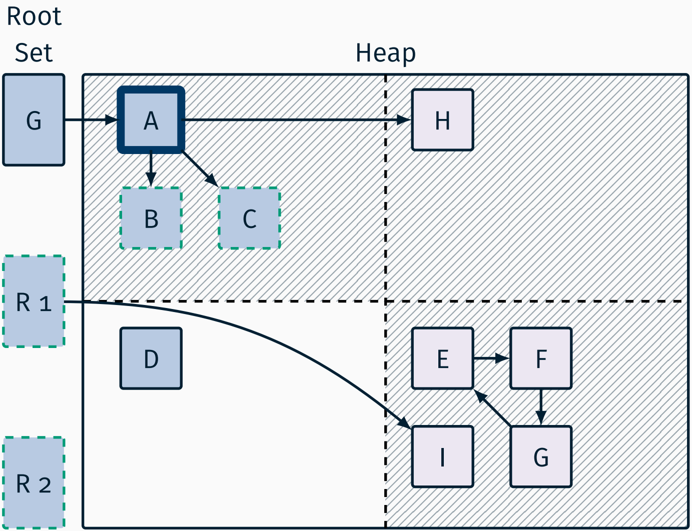
Program Allocates New Object



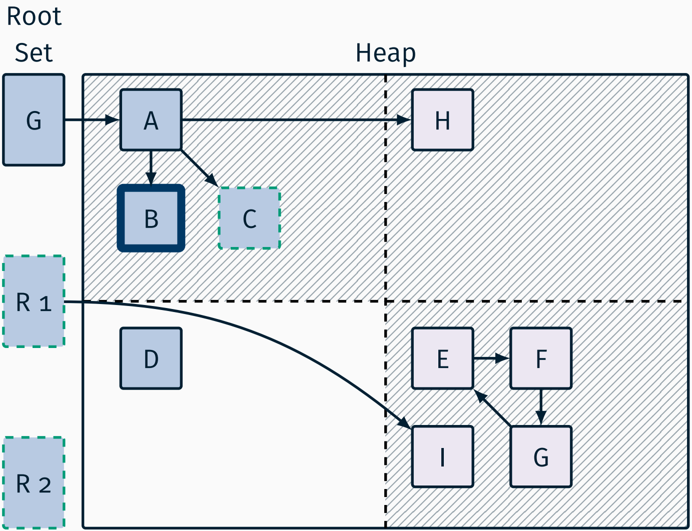
STW Correction Marking From Marked Objects on Dirty Pages



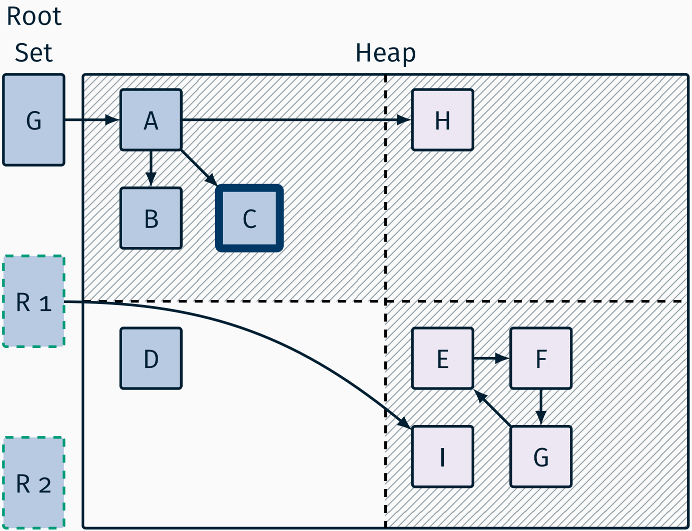
STW Correction Marking From Marked Objects on Dirty Pages



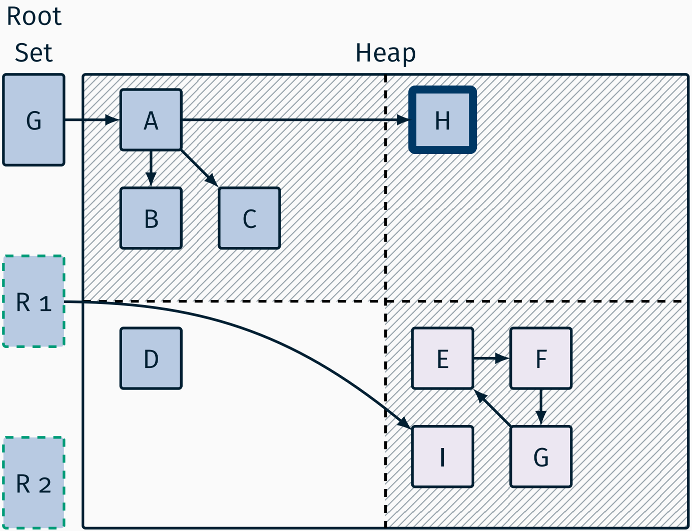
STW Correction Marking From Marked Objects on Dirty Pages



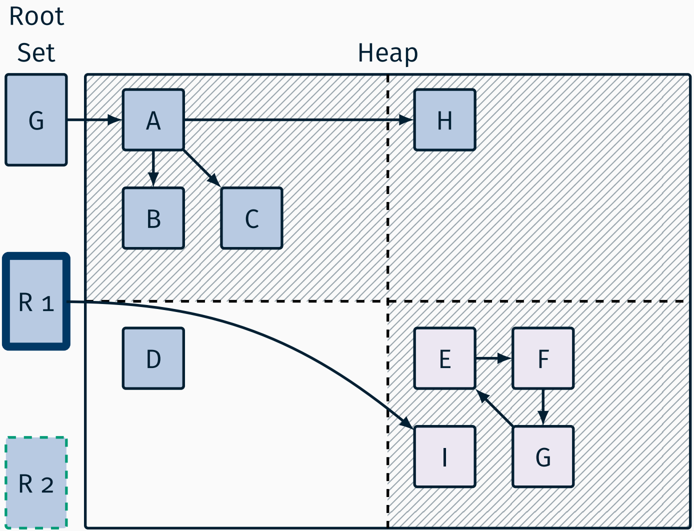
STW Correction Marking From Marked Objects on Dirty Pages



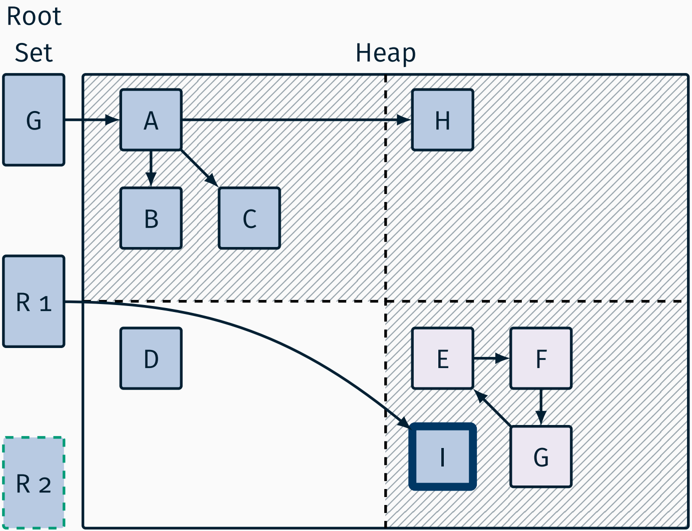
STW Correction Marking From Marked Objects on Dirty Pages



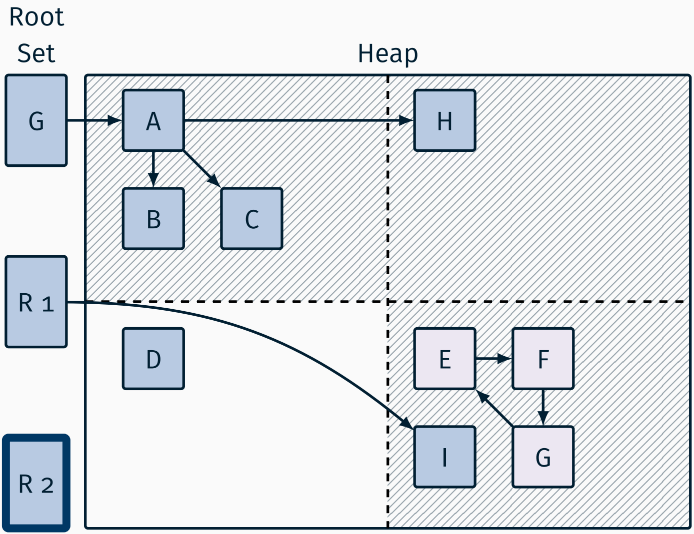
STW Correction Marking From Registers



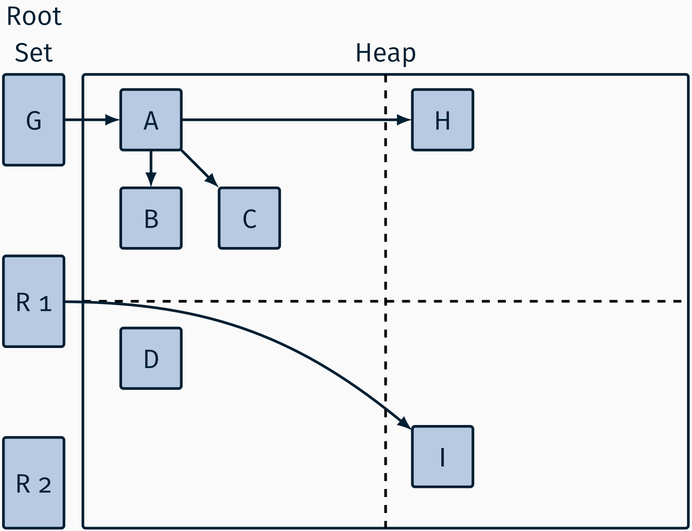
STW Correction Marking From Registers



STW Correction Marking From Registers



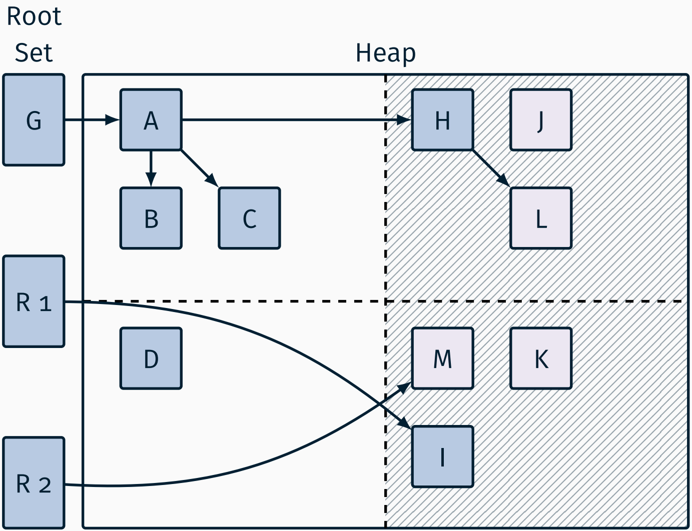
Full Collection Cycle Complete



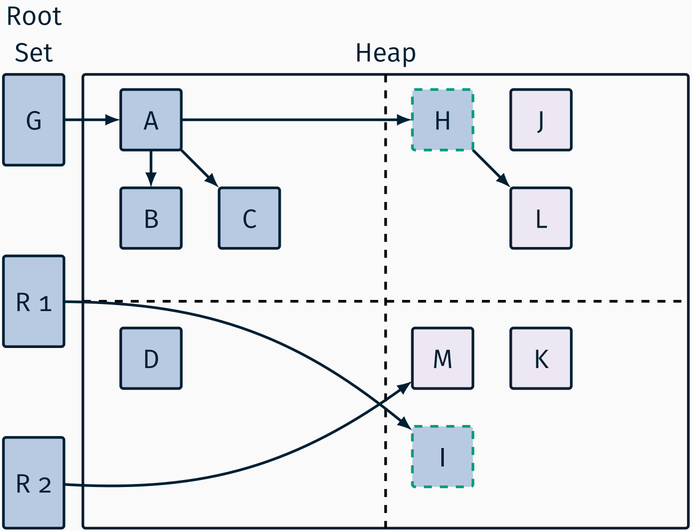
Partial (Young) Collection

1. Atomically retrieve and clear dirty bits of all pages
2. Trace from the marked objects on the retrieved dirty pages
3. Stop the world
4. Trace from registers and all marked objects on dirty pages
5. Clear dirty bits and restart the world

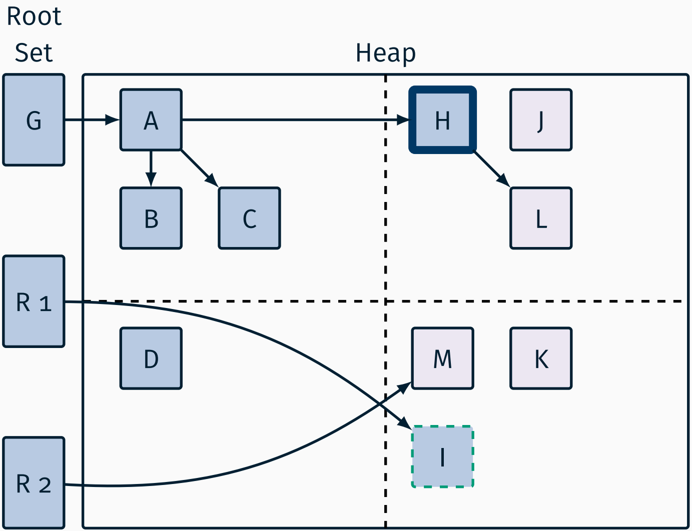
Partial Collection by Example



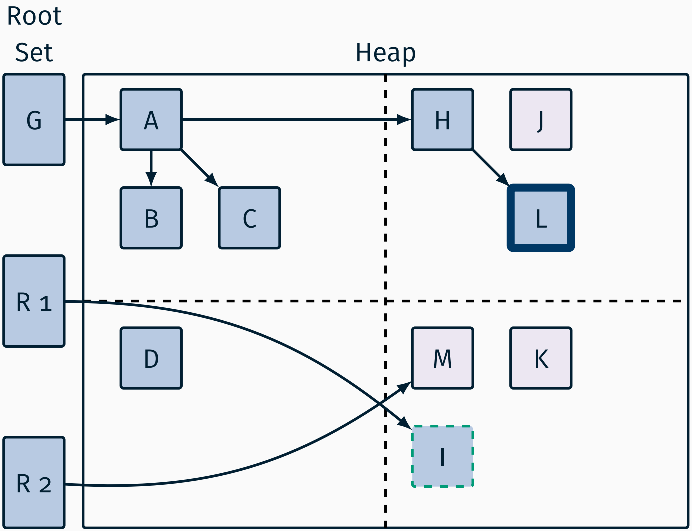
Clear Dirty Bits and Remember Pages



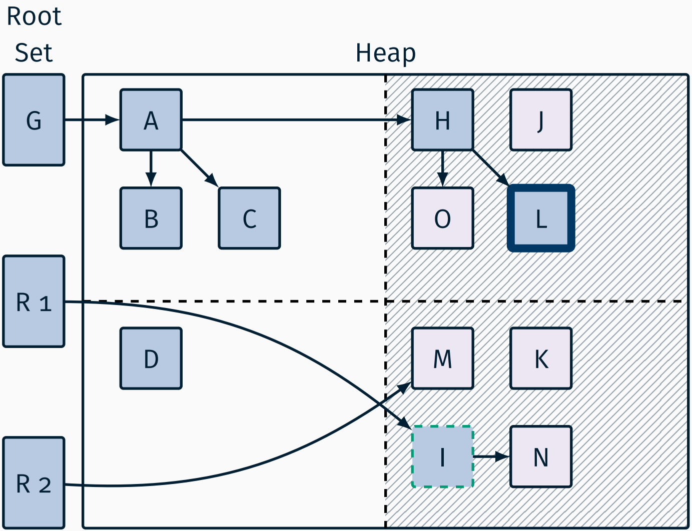
Concurrent Marking from Marked Objects on Dirty Pages



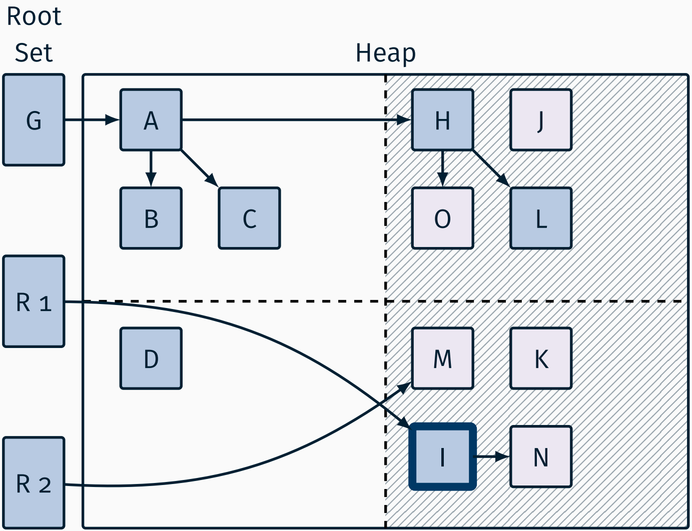
Concurrent Marking from Marked Objects on Dirty Pages



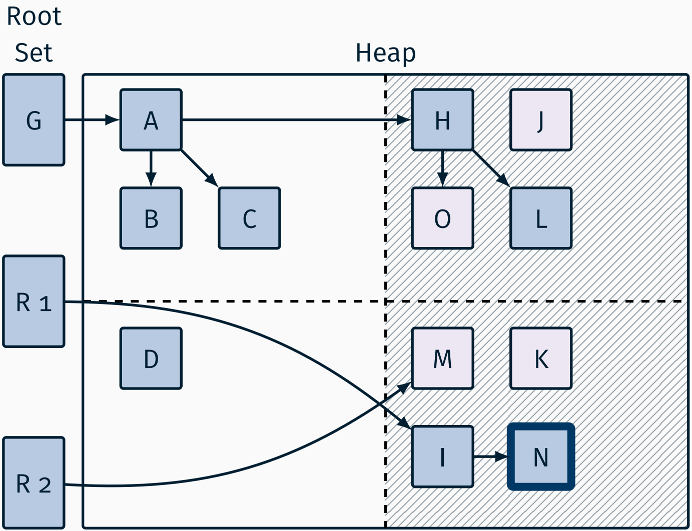
Program Allocates New Objects



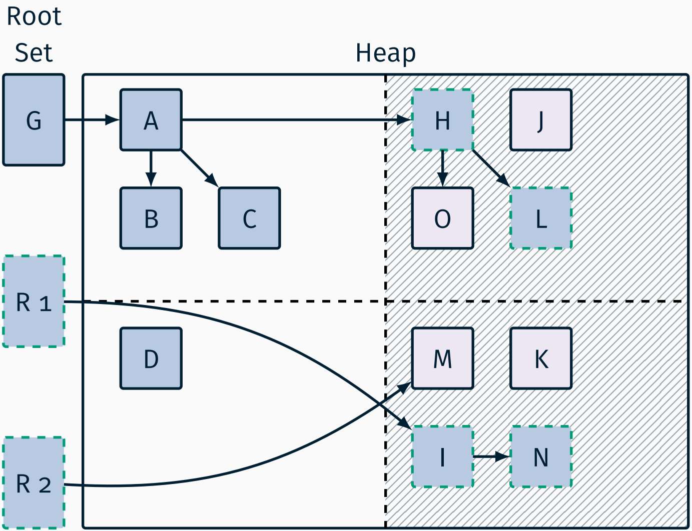
Concurrent Marking Continues



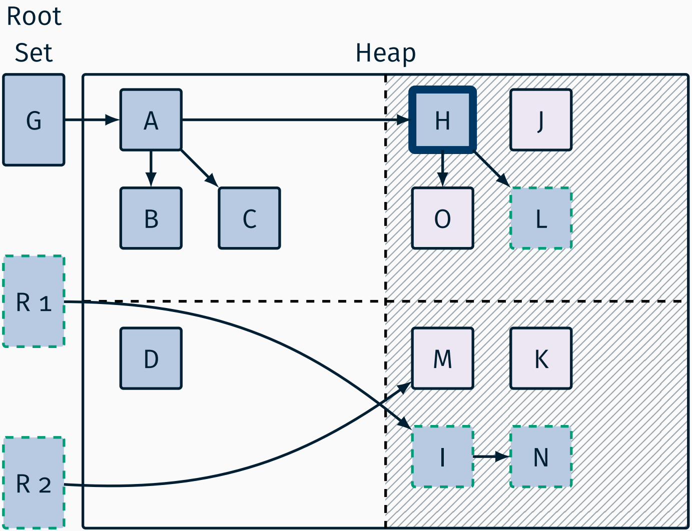
Concurrent Marking Continues



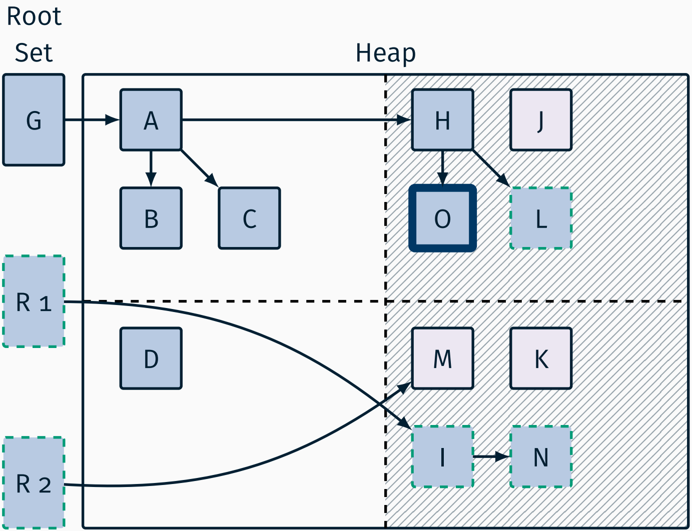
STW Marking from Marked Objects on Dirty Pages



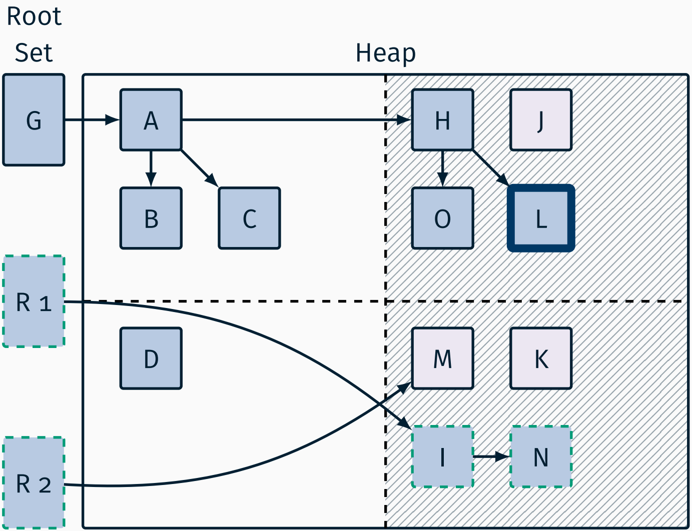
STW Marking from Marked Objects on Dirty Pages



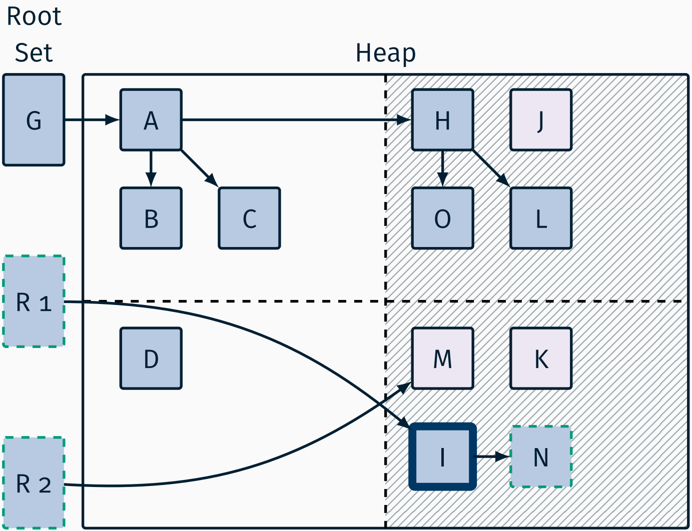
STW Marking from Marked Objects on Dirty Pages



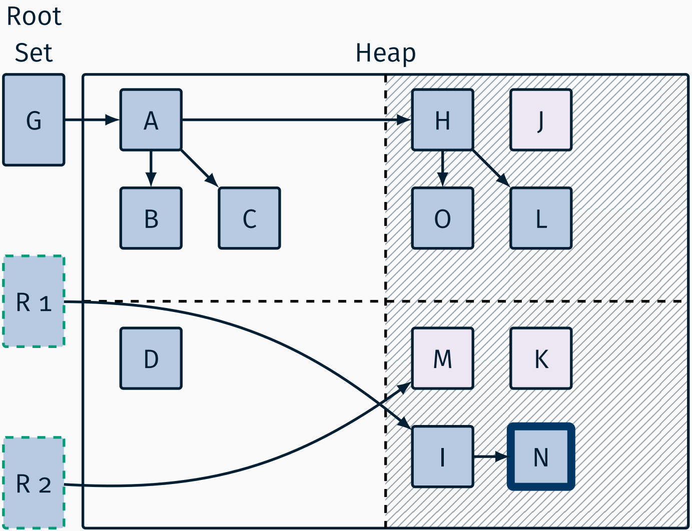
STW Marking from Marked Objects on Dirty Pages



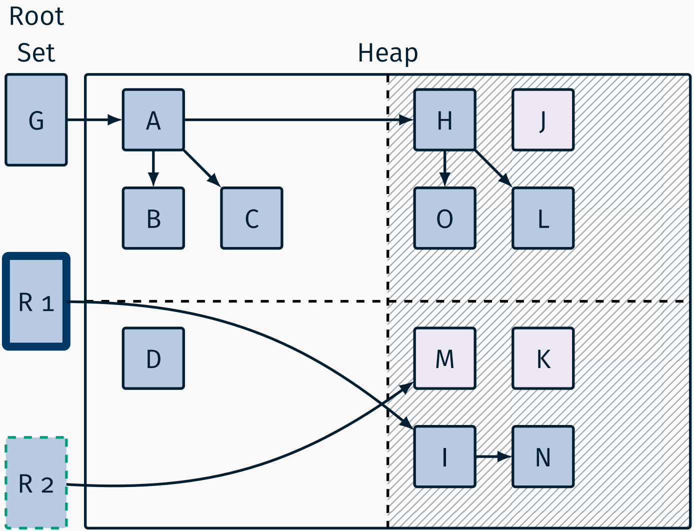
STW Marking from Marked Objects on Dirty Pages



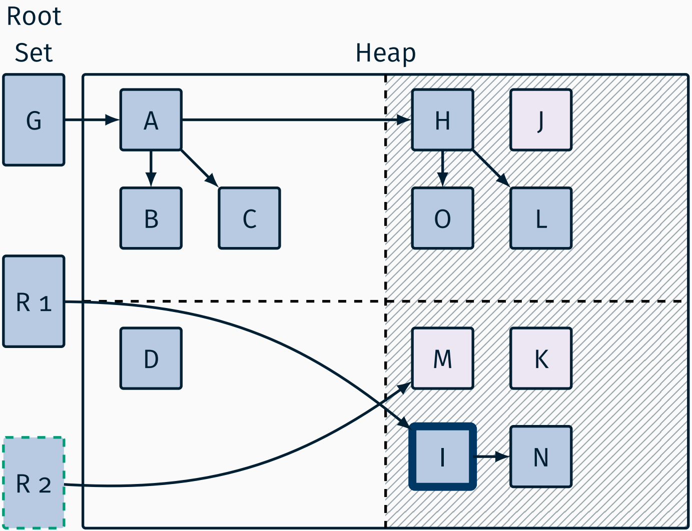
STW Marking from Marked Objects on Dirty Pages



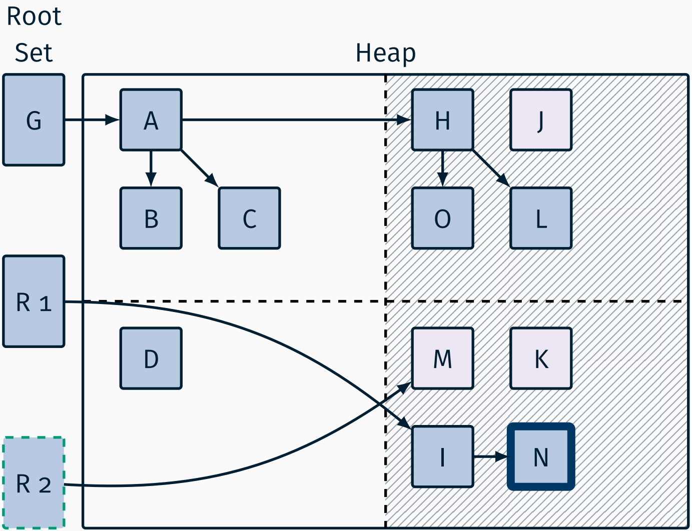
STW Marking from Registers



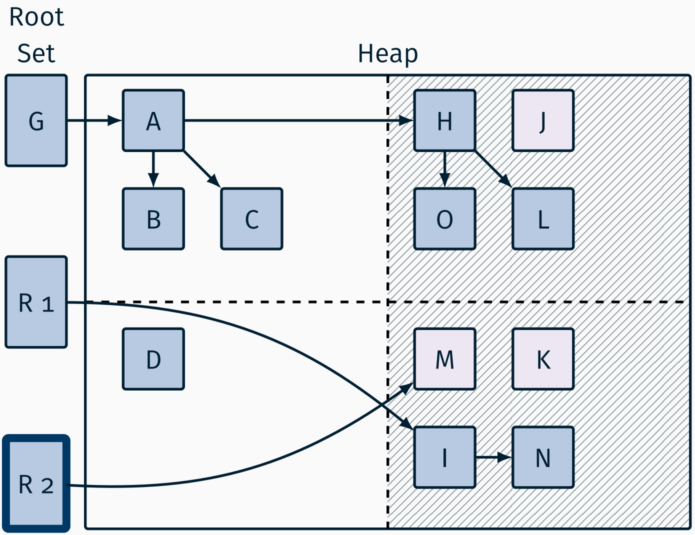
STW Marking from Registers



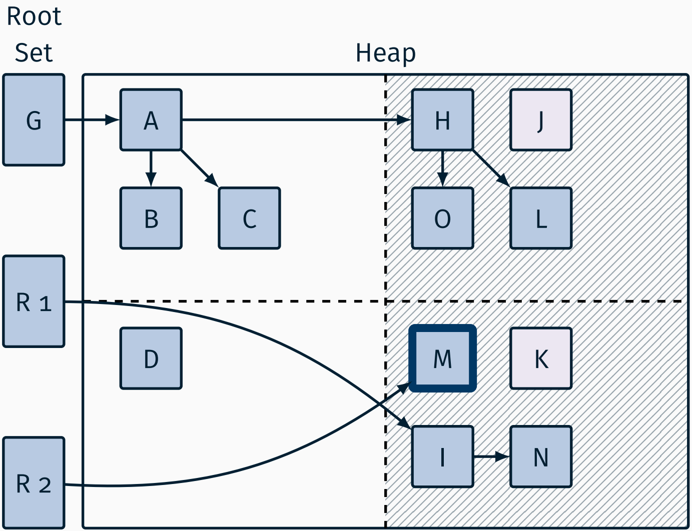
STW Marking from Registers



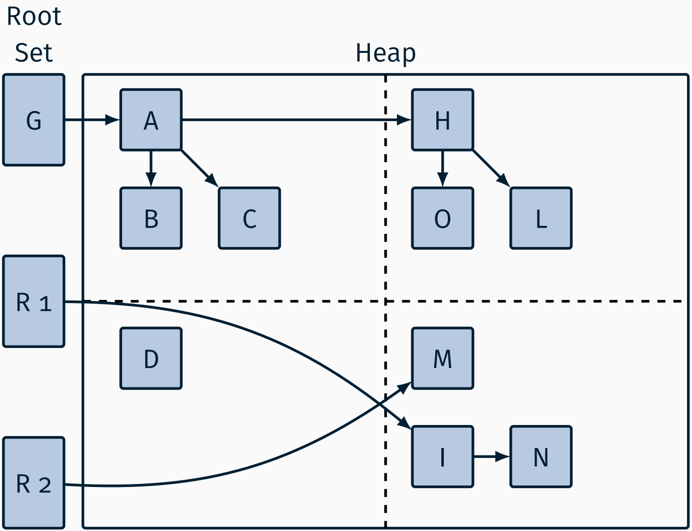
STW Marking from Registers



STW Marking from Registers



GC Cycle Complete



Discussion

Problems

- many objects are scanned unnecessarily
- some garbage remains until next full collection
- heap is potentially heavily fragmented

Discussion

Problems

- many objects are scanned unnecessarily
- some garbage remains until next full collection
- heap is potentially heavily fragmented

Advantages

+ usually registers are only a small part of the root set
+ STW correction phase still shorter than with full STW approach
+ some of the drawbacks are due to their specific use case

Discussion

Problems

- many objects are scanned unnecessarily
- some garbage remains until next full collection
- heap is potentially heavily fragmented

Advantages

+ usually registers are only a small part of the root set
+ STW correction phase still shorter than with full STW approach
+ some of the drawbacks are due to their specific use case

Main Contribution

General scheme for transforming many STW GCs to mostly concurrent versions

Modern GC Algorithms on the JVM

- compiler can assist in cooperation between mutator and GC
- large heap sizes (multiple 100 GB)
- almost all GC algorithms are generational

Modern GC Algorithms on the JVM

- compiler can assist in cooperation between mutator and GC
- large heap sizes (multiple 100 GB)
- almost all GC algorithms are generational

Concurrent Mark-Sweep

→ closely related to Boehm et al.'s for old generation

→ copying STW collector for young generation

→ often long pauses → deprecated and now removed

Modern GC Algorithms on the JVM

- compiler can assist in cooperation between mutator and GC
- large heap sizes (multiple 100 GB)
- almost all GC algorithms are generational

Concurrent Mark-Sweep

→ closely related to Boehm et al.'s for old generation

→ copying STW collector for young generation

→ often long pauses → deprecated and now removed

Garbage-First Garbage Collector (G1GC)

→ splits heap in same-sized individually collectible regions

→ *compacting*, STW young-generation collection

→ synchronisation during old collection: compiler-inserted write barrier records all pointer changes

Conclusion

Conclusion

Tracing Garbage Collection

- various different algorithms and approaches in practice and academia
- STW GC causes long pauses on large heaps

Conclusion

Tracing Garbage Collection

- various different algorithms and approaches in practice and academia
- STW GC causes long pauses on large heaps

"Mostly-Parallel Garbage Collection"

- concurrent collection can reduce pause times at the cost of more collection work
- general scheme: mark concurrently and then fix in short stop-the-world pauses

Conclusion

Tracing Garbage Collection

- various different algorithms and approaches in practice and academia
- STW GC causes long pauses on large heaps

“Mostly-Parallel Garbage Collection”

- concurrent collection can reduce pause times at the cost of more collection work
- general scheme: mark concurrently and then fix in short stop-the-world pauses

State on the JVM

- high degree of mutator cooperation
- mostly concurrent, but very different implementations

Thank you!

Thank you!
Questions?

Bibliography (1)

[BDS91] Hans-J. Boehm, Alan J. Demers, and Scott Shenker. "Mostly parallel garbage collection". In: *ACM SIGPLAN Notices* 26.6 (1991), pp. 157–164.

[Det+04] David Detlefs et al. "Garbage-First Garbage Collection". In: *Proceedings of the 4th International Symposium on Memory Management*. ISMM '04. Vancouver, BC, Canada, 2004, pp. 37–48. ISBN: 1581139454. DOI: 10.1145/1029873.1029879.

[PD00] Tony Printezis and David Detlefs. "A Generational Mostly-Concurrent Garbage Collector". In: *Proceedings of the 2nd International Symposium on Memory Management*. ISMM '00. Minneapolis, Minnesota, USA, 2000, pp. 143–154. ISBN: 1581132638. DOI: 10.1145/362422.362480.