Concurrent, Parallel Garbage Collection in Linear Time

Steven R. Brandt, Hari Krishnan, Gokarna Sharma and Costas Busch

Proceedings of the 2014 International Symposium on Memory Management

Concurrency and Memory Management Seminar 2015

Günther Eder

Computer Sciences
University of Salzburg

May 2015, Salzburg

GARBAGE COLLECTION (GC)

- Invented by John McCarthy in 1959[1].
- Two main categories:
    - *Tracing Garbage Collection*
    - *Reference Counting*

[1] McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I".

Garbage Collection (GC)

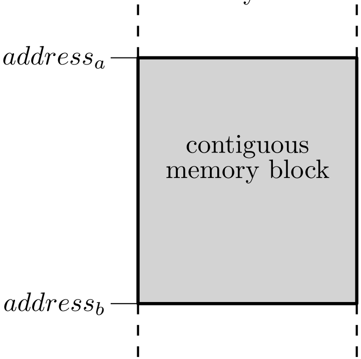
- Invented by John McCarthy in 1959[1].
- Two main categories:
    - *Tracing Garbage Collection*
    - *Reference Counting*

[1] McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I".

GARBAGE COLLECTION (GC)

- Invented by John McCarthy in 1959[1].
- Two main categories:
  - *Tracing Garbage Collection*
  - *Reference Counting*

Tracing GC
Memory Model

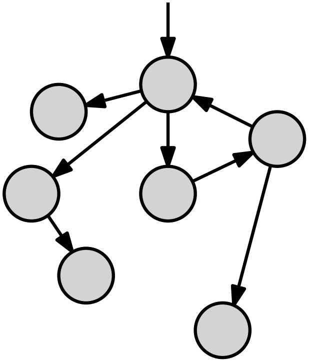


[1] McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I".

Garbage Collection (GC)

- Invented by John McCarthy in 1959[1].
- Two main categories:
  - *Tracing Garbage Collection*
  - *Reference Counting*

Reference Counting
Memory Model



[1] McCarthy, “Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I”.

Garbage Collection (GC)

- Invented by John McCarthy in 1959[1].
- Two main categories:
    - *Tracing Garbage Collection*
    - *Reference Counting*

Pros

- Solves dangling pointer problems.
- Solves double free issues.
- Prohibit certain types of memory leaks.

[1] McCarthy, “Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I”.

Garbage Collection (GC)

- Invented by John McCarthy in 1959[1].
- Two main categories:
  - *Tracing Garbage Collection*
  - *Reference Counting*

Pros

- Solves dangling pointer problems.
- Solves double free issues.
- Prohibit certain types of memory leaks.

Cons

- Consumes computing resources.
- Unpredictable.

[1] McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I".

Reverence Counting

Basics

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

Reverence Counting

Basics

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

Reverence Counting

Basics

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

Pros

- Clearly defined lifetime of every object.
- Does not “stop the world”.

Reverence Counting

Basics

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

Pros

- Clearly defined lifetime of every object.
- Does not “stop the world”.

REVERENCE COUNTING

BASICS

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

PROS

- Clearly defined lifetime of every object.
- Does not "stop the world".

CONS

- Naive approach can not handle cyclic references
- Typically less overall throughput than a tracing GC

Reverence Counting

Basics

- Every object stores a count of references to itself
- If the last reference is lost, the object gets collected

Pros

- Clearly defined lifetime of every object.
- Does not “stop the world”.

Cons

- Naive approach can not handle cyclic references
- Typically less overall throughput than a tracing GC

REVERENCE COUNTING

BROWNBRIDGE'S APPROACH[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".

Reverence Counting

Brownbridge's approach[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".

Reverence Counting

Brownbridge's approach[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".

Reverence Counting

Brownbridge's approach[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

Open Problem

Find a way to lower the time complexity for Brownbridge's cleanup process.

[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".

REVERENCE COUNTING

BROWNBRIDGE'S APPROACH[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

OPEN PROBLEM

Find a way to lower the time complexity for Brownbridge's cleanup process.

"NEW" APPROACH[3]

- strong, weak and phantom references
- at most three linear traversals

[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".
[3] Brandt et al., "Concurrent, Parallel Garbage Collection in Linear Time".

Reverence Counting

Brownbridge's approach[2]

- strong and weak references
- correct cycle detection
- exponential cleanup time

Open Problem

Find a way to lower the time complexity for Brownbridge's cleanup process.

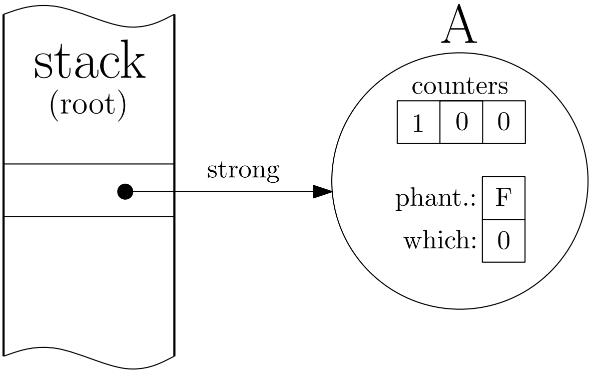
"New" approach[3]

- strong, weak and phantom references
- at most three linear traversals

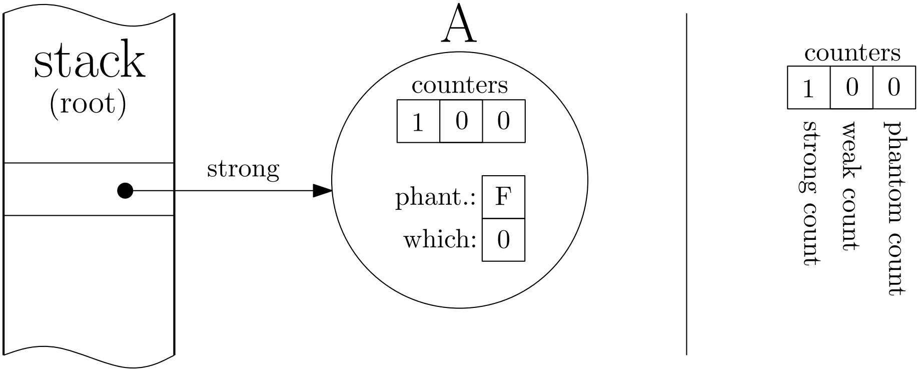
[2] Brownbridge, "Cyclic Reference Counting for Combinator Machines".

[3] Brandt et al., "Concurrent, Parallel Garbage Collection in Linear Time".

An Object and its References



An Object and its References



stack

(root)

strong

A

counters

1	0	0

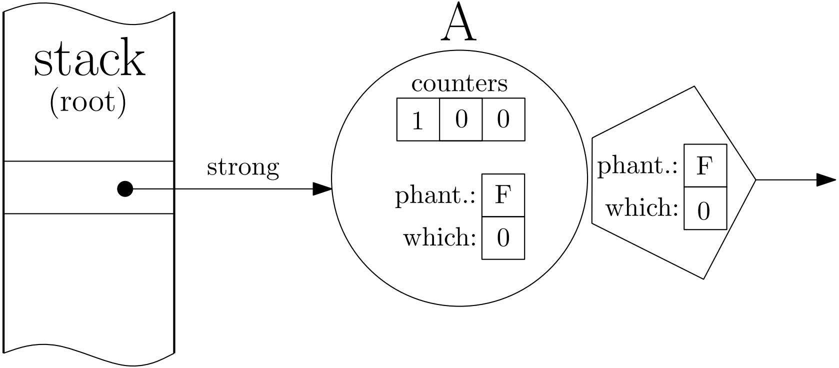
phant.: F

which: 0

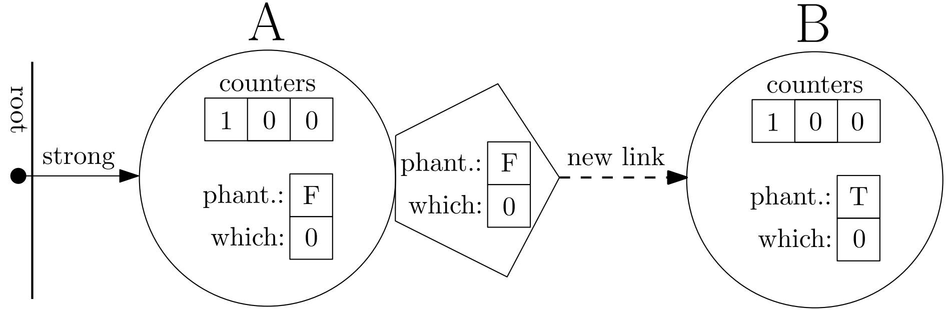
counters

1	0	0
strong count	weak count	phantom count

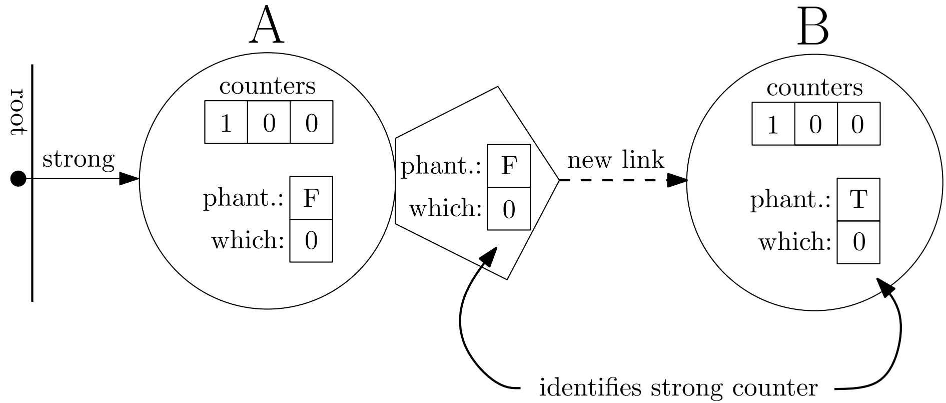
An Object and its References



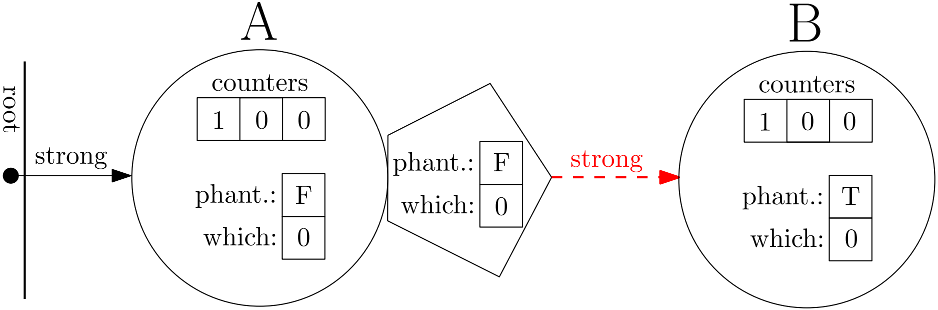
Object Creation



Object Creation



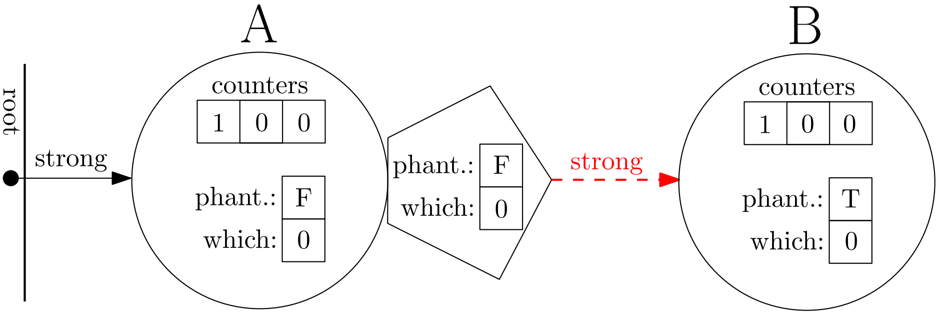
Object Creation



A strong link from A to B is created iff

- The only strong links to A are from roots.
- Object B has no outgoing links.
- Object B is phantomized, and A is not.

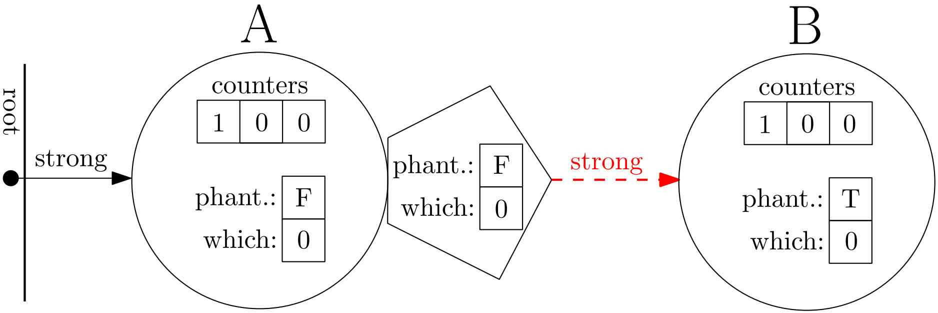
Object Creation



A strong link from A to B is created iff

- The only strong links to A are from roots.
- Object B has no outgoing links.
- Object B is phantomized, and A is not.

Object Creation



A strong link from A to B is created iff

- The only strong links to A are from roots.
- Object B has no outgoing links.
- Object B is phantomized, and A is not.

References and Collection

Reference Details

- All selfreferences are weak.
- Any other link is created phantom or weak.

References and Collection

Reference Details

- All selfreferences are weak.
- Any other link is created phantom or weak.

References and Collection

Reference Details

- All selfreferences are weak.
- Any other link is created phantom or weak.

Collecting Process

- When the strong reference count of an object reaches zero, the GC process begins.

References and Collection

Reference Details

- All selfreferences are weak.
- Any other link is created phantom or weak.

Collecting Process

- When the strong reference count of an object reaches zero, the GC process begins.
    - If the object's weak reference count is zero, the object is immediately reclaimed.
    - If the weak count is positive, then a sequence of three phases is initiated:
        - Phantomization
        - Recovery
        - CleanUp

REFERENCES AND COLLECTION

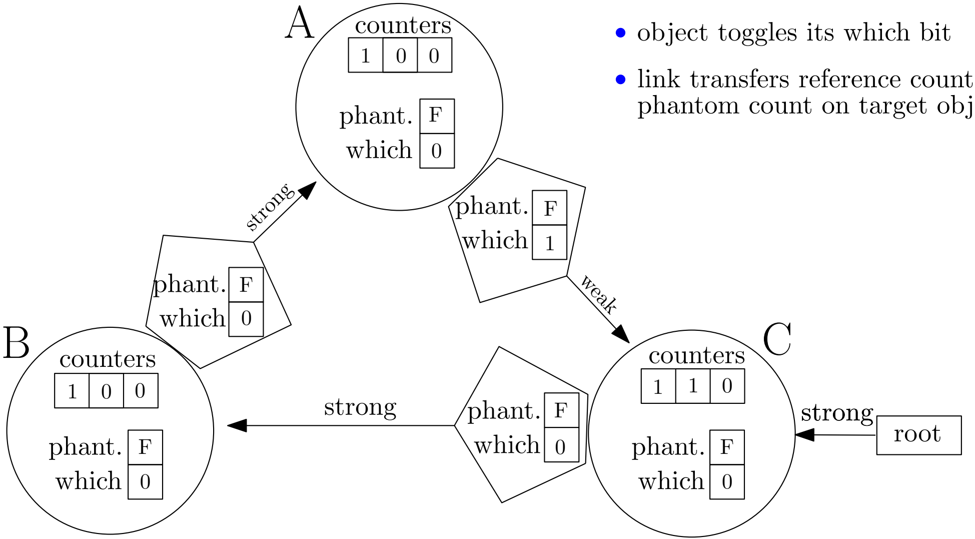
REFERENCE DETAILS

- All selfreferences are weak.
- Any other link is created phantom or weak.

COLLECTING PROCESS

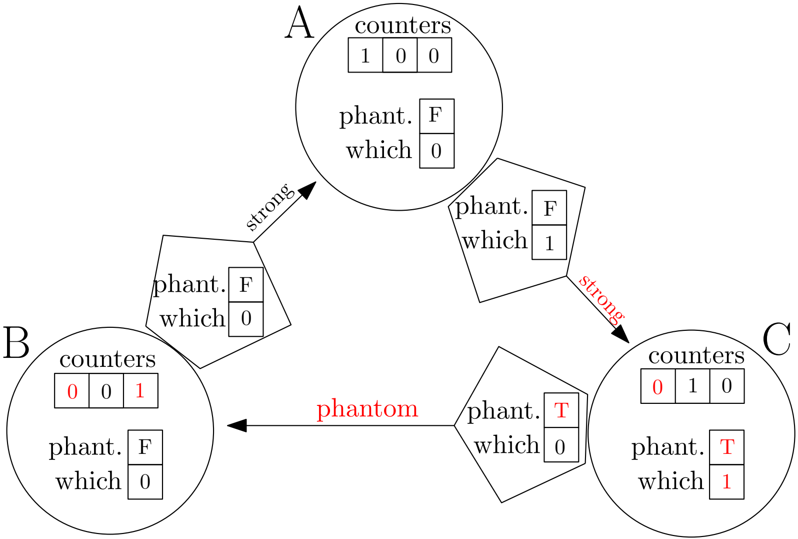
- When the strong reference count of an object reaches zero, the GC process begins.
  - If the object's weak reference count is zero, the object is immediately reclaimed.
  - If the weak count is positive, then a sequence of three phases is initiated:
    - Phantomization
    - Recovery
    - CleanUp

Phantomization

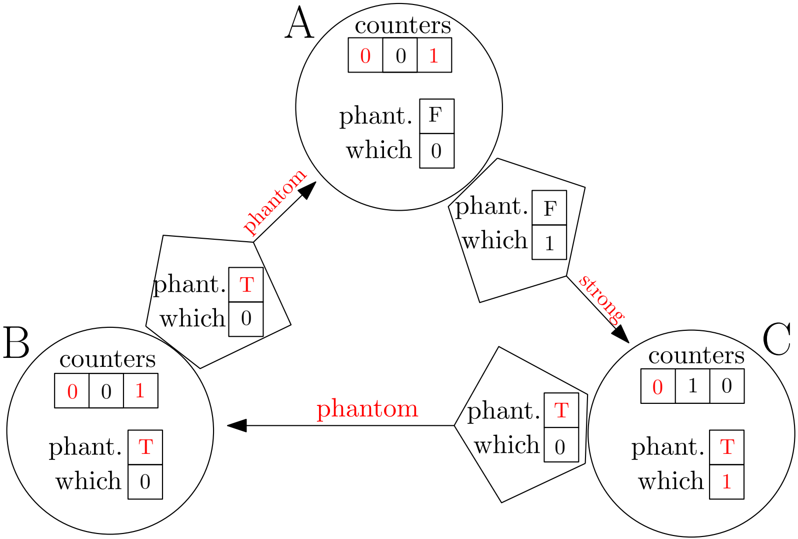


- object toggles its which bit
- link transfers reference count to phantom count on target object

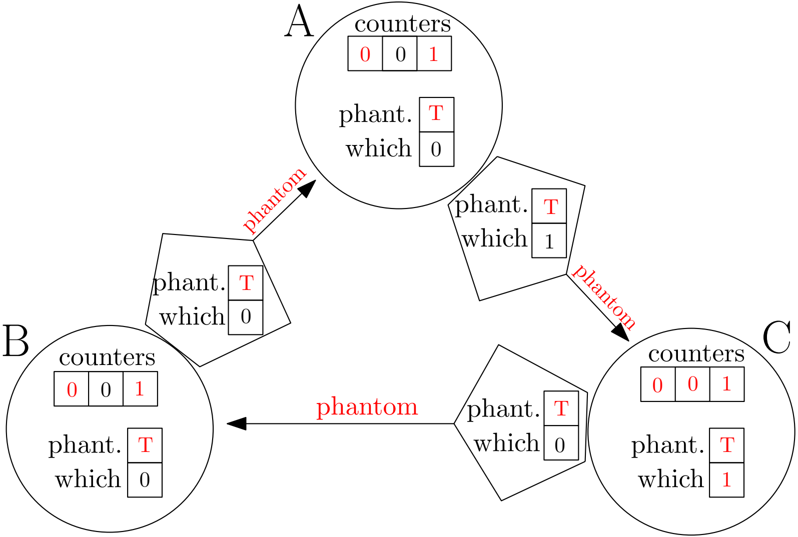
Phantomization



Phantomization



PHANTOMIZATION



A

counters

0 0 1

phant. T

which 0

phantom

phant. T

which 0

phant. T

which 1

phantom

B

counters

0 0 1

phant. T

which 0

phantom

phant. T

which 0

C

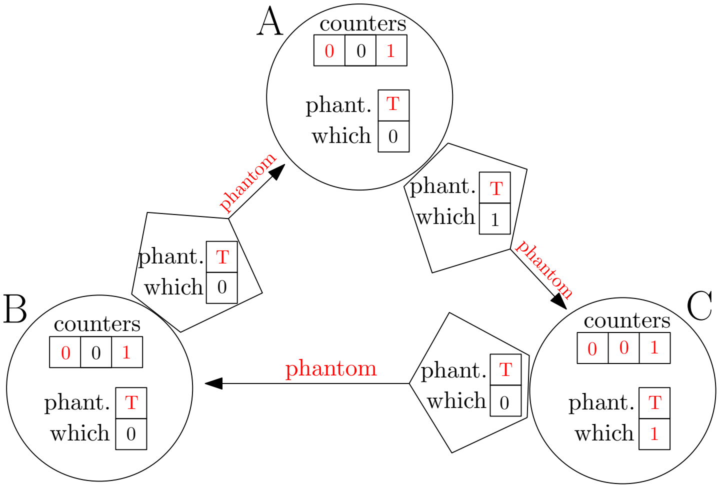
counters

0 0 1

phant. T

which 1

Phantomization



- all objects which recover a positive strong reference count are stored into a recovery list.

Recovery and Cleanup

Recovery

- For each object in the recovery list which has a positive strong reference count:
  - Set phantomized boolean to false
  - Rebuild outgoing links

Recovery and Cleanup

Recovery

- For each object in the recovery list which has a positive strong reference count:
    - Set phantomized boolean to false
    - Rebuild outgoing links

Recovery and Cleanup

Recovery

- For each object in the recovery list which has a positive strong reference count:
  - Set phantomized boolean to false
  - Rebuild outgoing links

Recovery and Cleanup

Recovery

- For each object in the recovery list which has a positive strong reference count:
  - Set phantomized boolean to false
  - Rebuild outgoing links

Cleanup

- Revisite recovery list a second time.
- Collect all objects that have no positive strong reference count.

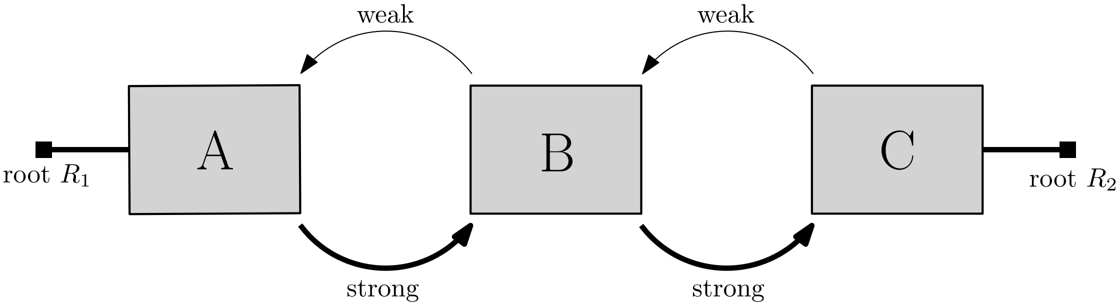
Recovery and Cleanup

Recovery

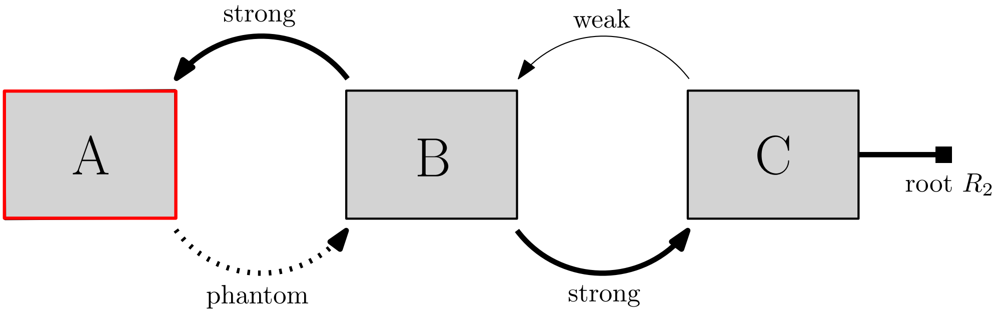
- For each object in the recovery list which has a positive strong reference count:
  - Set phantomized boolean to false
  - Rebuild outgoing links

Cleanup

- Revisite recovery list a second time.
- Collect all objects that have no positive strong reference count.

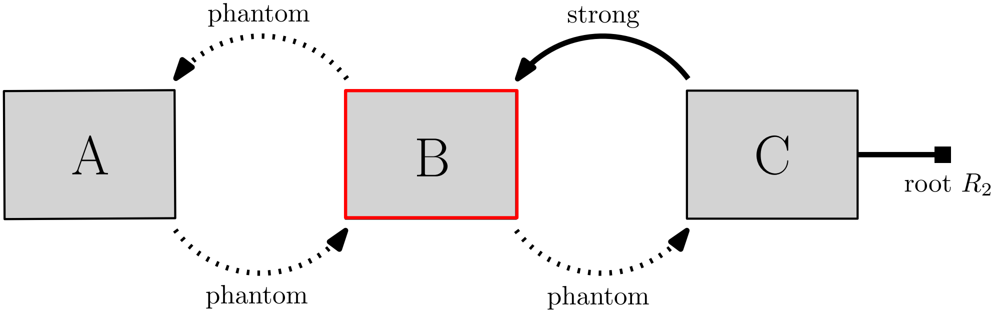


Phantomization



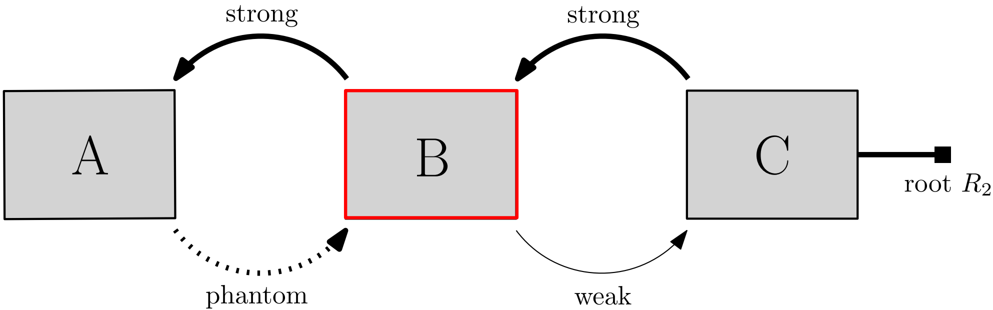
Phantomization

DOUBLY-LINKED-LIST EXAMPLE



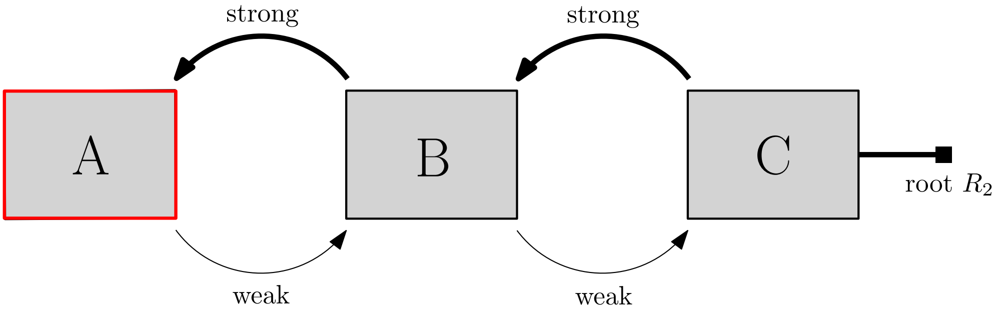
Phantomization

DOUBLY-LINKED-LIST EXAMPLE

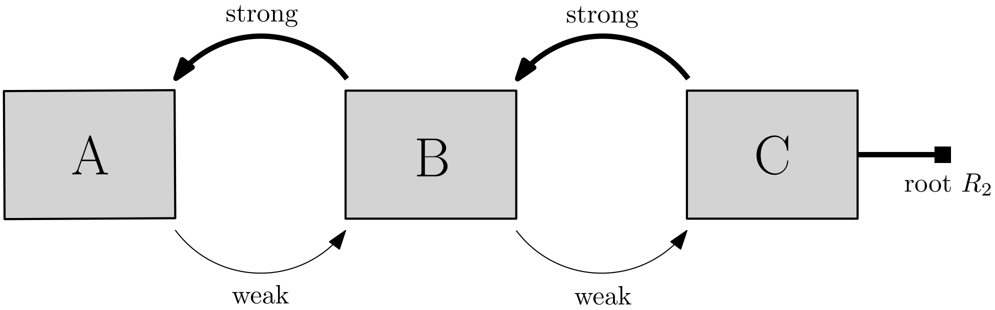


Recovery

DOUBLY-LINKED-LIST EXAMPLE



Recovery



Cleanup

Concurrent and Parallel

One Single-Threaded Collector.

- Constraints: phantomization, recovery and cleanup have to run in-order and to completion.
- If the last strong link to an object with a positive weak or phantom count is removed the live system transfers this link to the collector to run the collection at an appropriate time.

Concurrent and Parallel

One Single-Threaded Collector.

- Constraints: phantomization, recovery and cleanup have to run in-order and to completion.
- If the last strong link to an object with a positive weak or phantom count is removed the live system transfers this link to the collector to run the collection at an appropriate time.

Concurrent and Parallel

One Single-Threaded Collector.

- Constraints: phantomization, recovery and cleanup have to run in-order and to completion.
- If the last strong link to an object with a positive weak or phantom count is removed the live system transfers this link to the collector to run the collection at an appropriate time.

Multi-Threaded Collector

- Collector threads run in disjoint areas in memory.
- Each object needs to carry a reference to its collection thread.
- If collection extends to another memory area, a merge of both areas occur and one collection thread terminates.

CONCURRENT AND PARALLEL

ONE SINGLE-THREADED COLLECTOR.

- Constraints: phantomization, recovery and cleanup have to run in-order and to completion.
- If the last strong link to an object with a positive weak or phantom count is removed the live system transfers this link to the collector to run the collection at an appropriate time.

MULTI-THREADED COLLECTOR

- Collector threads run in disjoint areas in memory.
- Each object needs to carry a reference to its collection thread.
- If collection extends to another memory area, a merge of both areas occur and one collection thread terminates.

Concurrent and Parallel

One Single-Threaded Collector.

- Constraints: phantomization, recovery and cleanup have to run in-order and to completion.
- If the last strong link to an object with a positive weak or phantom count is removed the live system transfers this link to the collector to run the collection at an appropriate time.

Multi-Threaded Collector

- Collector threads run in disjoint areas in memory.
- Each object needs to carry a reference to its collection thread.
- If collection extends to another memory area, a merge of both areas occur and one collection thread terminates.

Correctness

- All three phases are linear in the number of links.
- At most one state change per phase can occur.

- **phantomization**: weak or strong → phantom
- **recovery**: phantom → weak or strong

CORRECTNESS

- All three phases are linear in the number of links.
- At most one state change per phase can occur.

- **phantomization**: weak or strong → phantom
- **recovery**: phantom → weak or strong

CORRECTNESS

- All three phases are linear in the number of links.
- At most one state change per phase can occur.

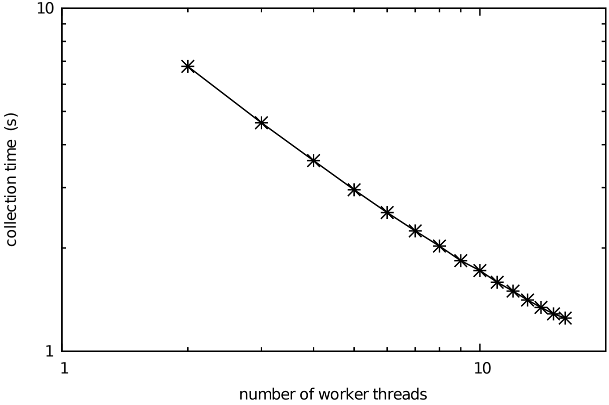
- **phantomization**: weak or strong → phantom
- **recovery**: phantom → weak or strong

CORRECTNESS

- All three phases are linear in the number of links.
- At most one state change per phase can occur.

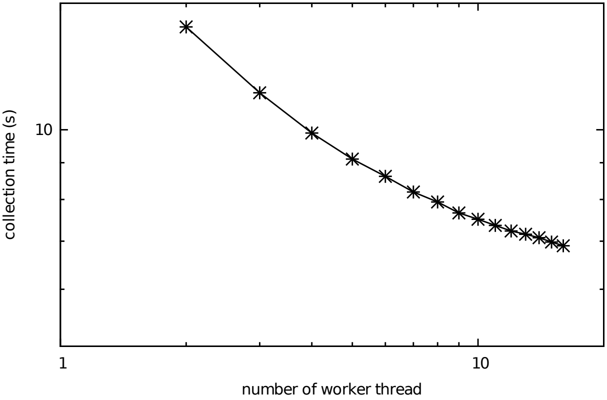
- **phantomization**: weak or strong → phantom
- **recovery**: phantom → weak or strong

RESULTS



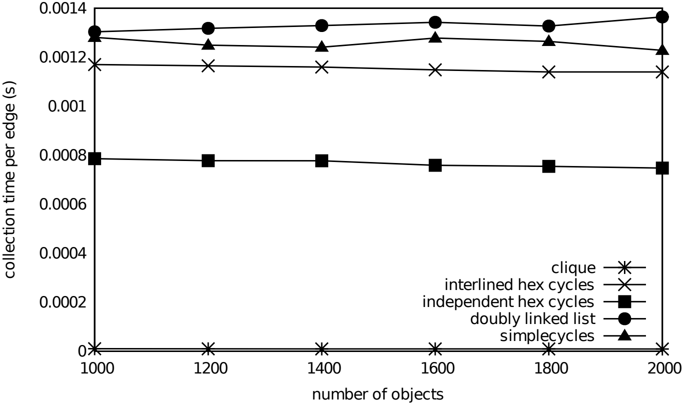
- A large number of independent rings are collected by various number of worker threads.

RESULTS



- A chain of linked cycles is created in memory. The connections are severed, then the roots are removed. Multiple collector threads are created and operations partially overlap.

RESULTS



- Graphs of different types are created at various sizes in memory, including cliques, chains of cycles, large cycles, and large doubly linked lists.

Conclusion

Advantages

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

Conclusion

Advantages

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

Conclusion

Advantages

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

CONCLUSION

ADVANTAGES

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

CONCLUSION

ADVANTAGES

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

CONCLUSION

ADVANTAGES

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

Conclusion

Advantages

- Can run at the same time as live system, using multiple threads if desired.
- No need to "stop the world".
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

Disadvantages

- Increased memory overhead.
- Additional cost of pointer creation/mutation..
- No optimized implementation yet.

CONCLUSION

ADVANTAGES

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

DISADVANTAGES

- Increased memory overhead.
- Additional cost of pointer creation/mutation..
- No optimized implementation yet.

CONCLUSION

ADVANTAGES

- Can run at the same time as live system, using multiple threads if desired.
- No need to “stop the world”.
- Performs nontrivial work only when the last strong link is removed.
- When objects do need to be collected, the collector only needs to trace the cycle twice.
- The collector does not need or use back-pointers.
- Useful for distributed systems.

DISADVANTAGES

- Increased memory overhead.
- Additional cost of pointer creation/mutation..
- No optimized implementation yet.

End

Thanks for your attention!

References I

- McCarthy, John. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". In: *Commun. ACM* 3.4 (Apr. 1960), pp. 184–195. ISSN: 0001-0782. DOI: 10.1145/367177.367199. URL: http://doi.acm.org/10.1145/367177.367199.
- Brownbridge, David R. "Cyclic Reference Counting for Combinator Machines". In: *Record of the 1985 Conference on Functional Programming and Computer Architecture*. Ed. by Jean-Pierre Jouannaud. Vol. 201. Nancy, France, Sept. 1985, pp. 273–288.
- Brandt, Steven R. et al. "Concurrent, Parallel Garbage Collection in Linear Time". In: *Proceedings of the 2014 International Symposium on Memory Management*. ISMM '14. Edinburgh, United Kingdom: ACM, 2014, pp. 47–58. ISBN: 978-1-4503-2921-7. DOI: 10.1145/2602988.2602990. URL: http://doi.acm.org/10.1145/2602988.2602990.