We make the figures from *The Garbage Collection Handbook: The Art of Automatic Memory Management*, Richard Jones, Antony Hosking, Eliot Moss (Chapman and Hall, 2011) available for fair use by educators and students We ask that the following credit be given:

## Chapter 1

# Introduction

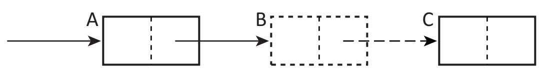



**Figure 1.1:** Premature deletion of an object may lead to errors. Here B has been freed. The live object A now contains a dangling pointer. The space occupied by C has leaked: C is not reachable but it cannot be freed.

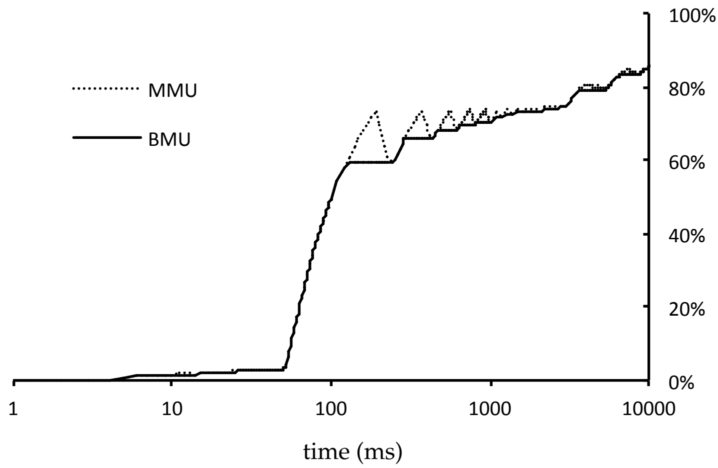



**Figure 1.2:** Minimum mutator utilisation and bounded mutator utilisation curves display concisely the (minimum) fraction of time spent in the mutator, for any given time window. MMU is the *minimum* mutator utilisation ($y$) in any time window ($x$) whereas BMU is minimum mutator utilisation in that time window or *any larger* one. In both cases, the $x$-intercept gives the maximum pause time and the $y$-intercept is the overall fraction of processor time used by the mutator.

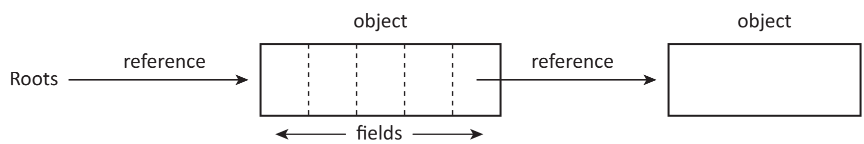



**Figure 1.3:** Roots, heap cells and references. Objects, denoted by rectangles, may be divided into a number of fields, delineated by dashed lines. References are shown as solid arrows.

# Chapter 2

# Mark-sweep garbage collection

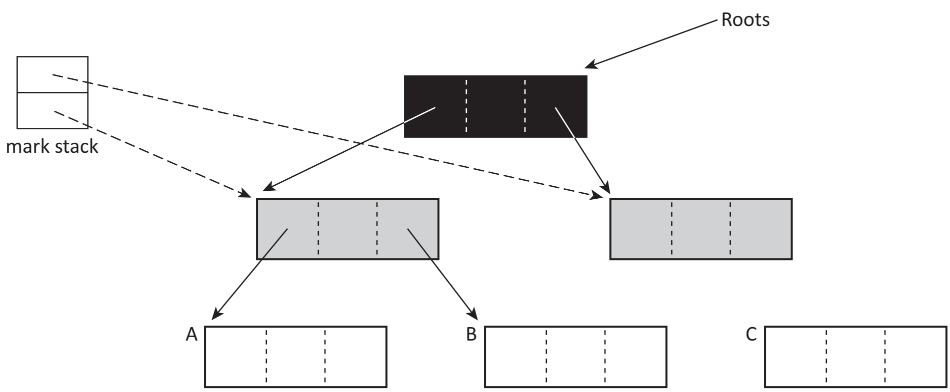



**Figure 2.1:** Marking with the tricolour abstraction. Black objects and their children have been processed by the collector. The collector knows of grey objects but has not finished processing them. White objects have not yet been visited by the collector (and some will never be).

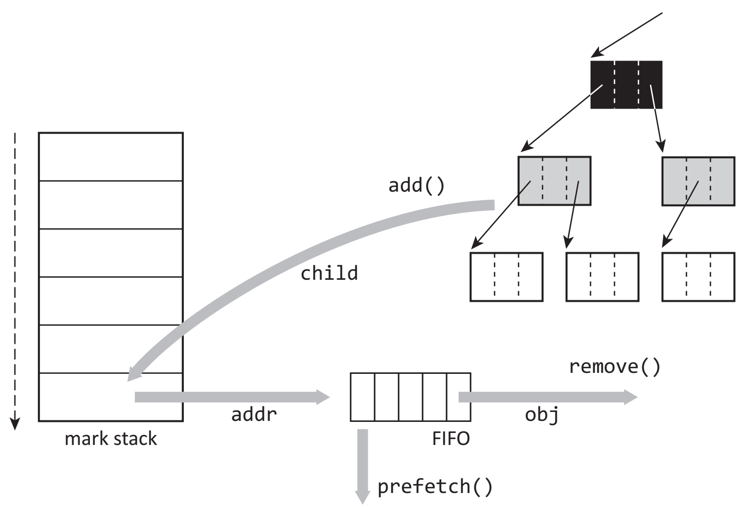



**Figure 2.2:** Marking with a FIFO prefetch buffer. As usual, references are added to the work list by being pushed onto the mark stack. However, to remove an item from the work list, the oldest item is removed from the FIFO buffer and the entry at the top of the stack is inserted into it. The object to which this entry refers is prefetched so that it should be in the cache by the time this entry leaves the buffer.

# Chapter 3

# Mark-compact garbage collection

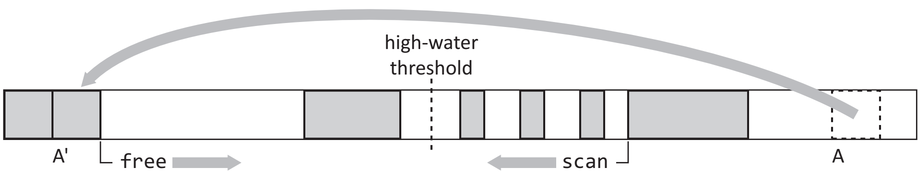
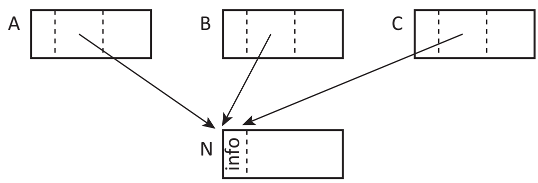
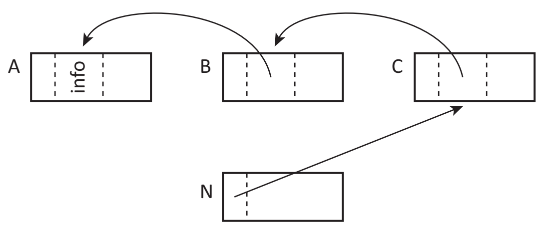



**Figure 3.1:** Edwards's Two-Finger algorithm. Live objects at the top of the heap are moved into free gaps at the bottom of the heap. Here, the object at A has been moved to A′. The algorithm terminates when the `free` and `scan` pointers meet.




(a) Before threading: three objects refer to N




(b) After threading: all pointers to N have been 'threaded' so that the objects that previously referred to N can now be found from N. The value previously stored in the header word of N, which is now used to store the threading pointer, has been (temporarily) moved to the first field (in A) that referred to N.

**Figure 3.2:** Threading pointers

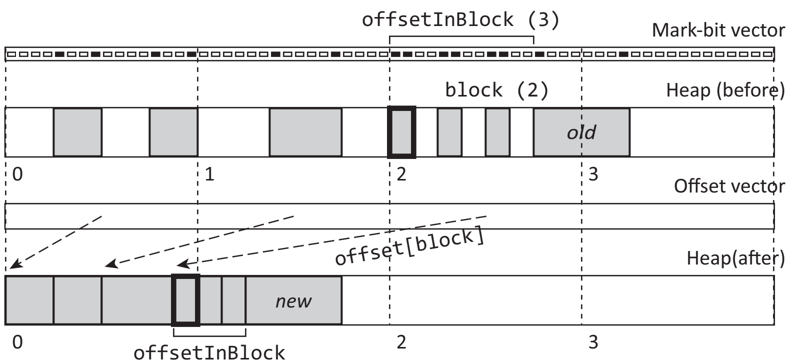
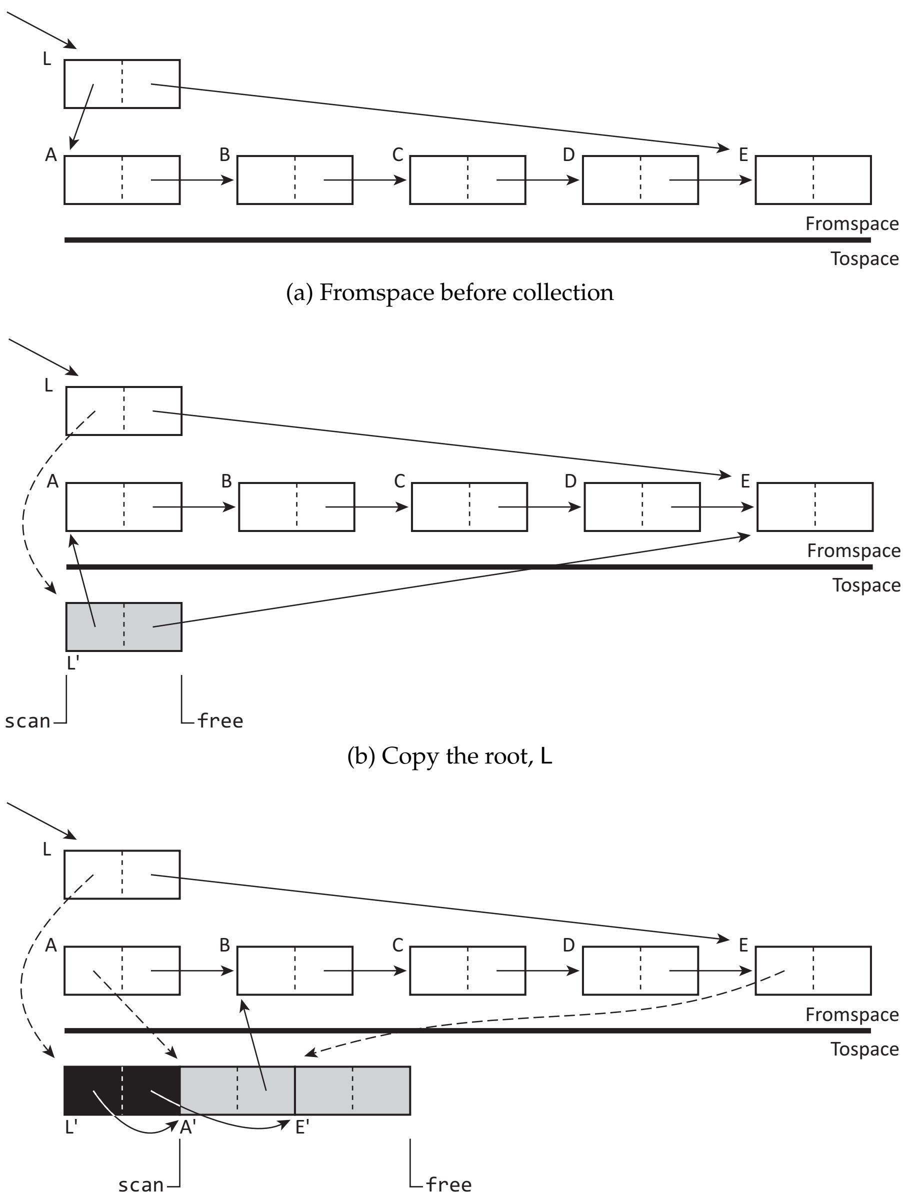



**Figure 3.3:** The heap (before and after compaction) and metadata used by Compressor [Kermany and Petrank, 2006]. Bits in the mark-bit vector indicate the start and end of each live object. Words in the offset vector hold the address to which the first live object in their corresponding block will be moved. Forwarding addresses are not stored but are calculated when needed from the offset and mark-bit vectors.

## Chapter 4

# Copying garbage collection

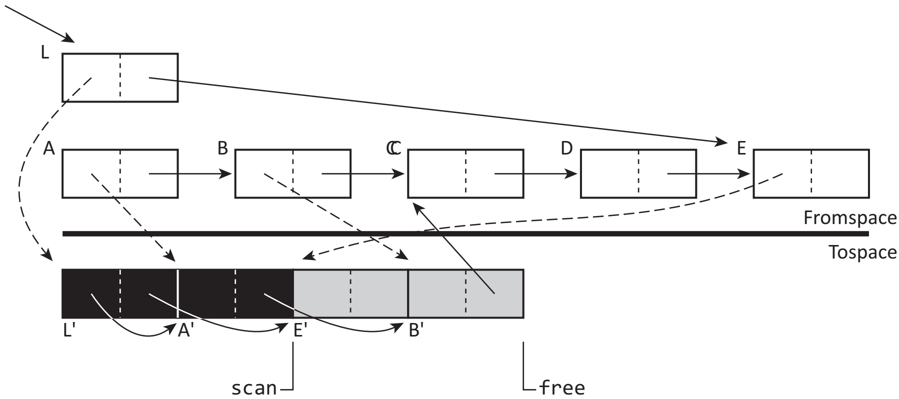



(a) Fromspace before collection

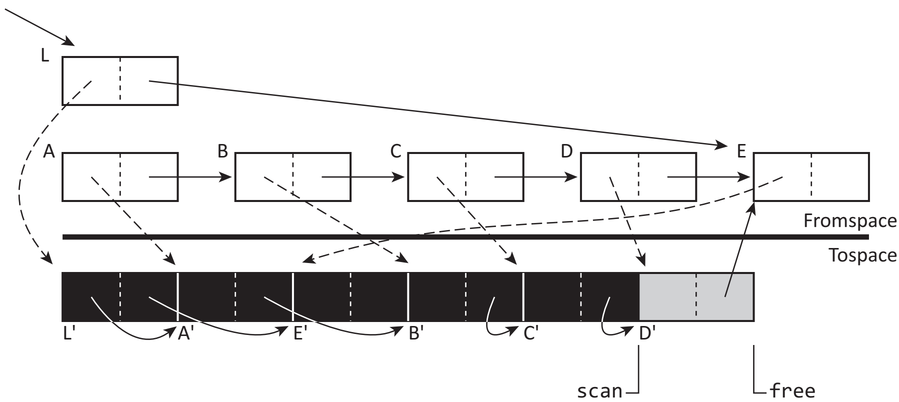
(b) Copy the root, L

(c) Scan L's replica

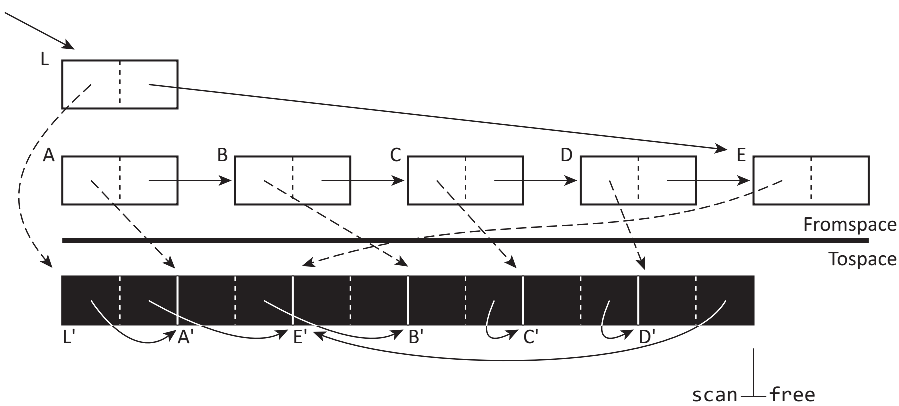
**Figure 4.1:** Copying garbage collection: an example




(d) Scan A's replica, and so on. . .

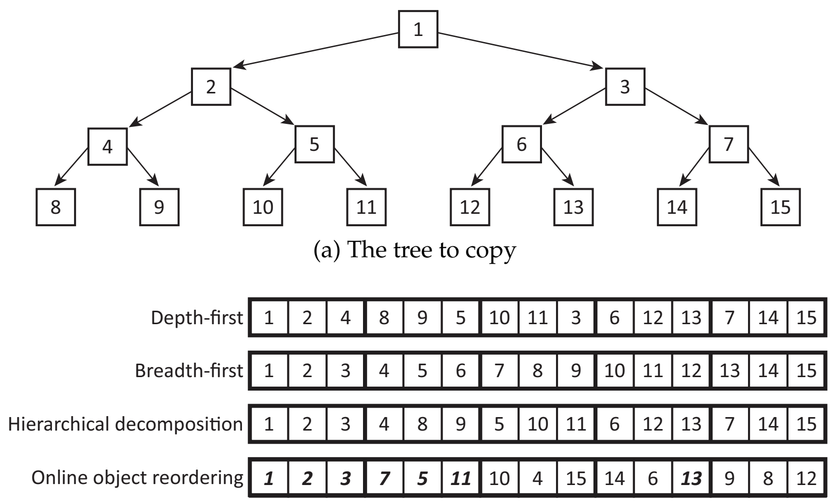



(e) Scan C's replica.




(f) Scan D's replica. `scan=free` so collection is complete.

**Figure 4.1 (continued):** Copying garbage collection: an example




(a) The tree to copy

(b) Placement of objects in the heap after copying

**Figure 4.2:** Copying a tree with different traversal orders. Each row shows how a traversal order lays out objects in tospace, assuming that three objects can be placed on a page (indicated by the thick borders). For *online object reordering*, prime numbered (bold italic) fields are considered to be hot.

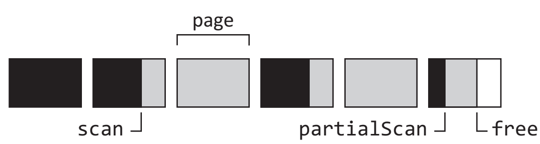



**Figure 4.3:** Moon's approximately depth-first copying. Each block represents a page. As usual, scanned fields are black, and copied but not yet scanned ones are grey. Free space is shown in white.

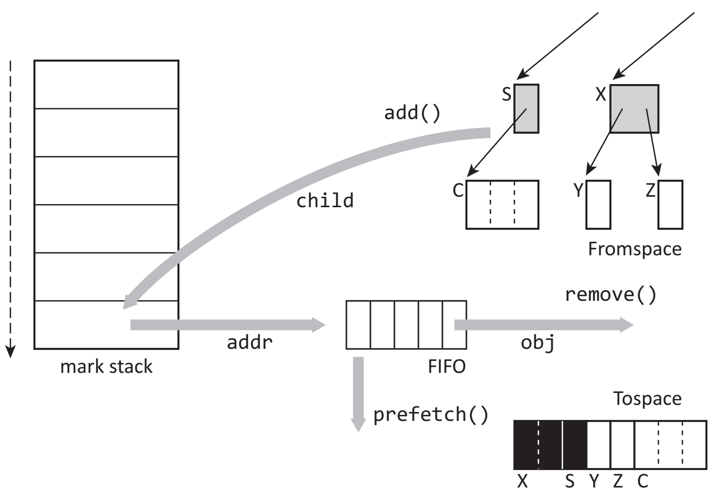



**Figure 4.4:** A FIFO prefetch buffer (discussed in Chapter 2) does not improve locality with copying as distant cousins (C, Y, Z), rather than parents and children, tend to be placed together.

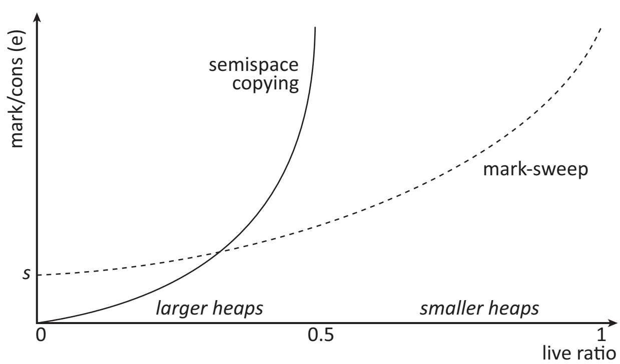



**Figure 4.5:** Mark/cons ratios for mark-sweep and copying collection (lower is better).

Chapter 5

# Reference counting

Figure withheld

**Figure 5.1:** Deferred reference counting schematic, showing whether reference counting operations on pointer loads or stores should be deferred or be performed eagerly. The arrows indicate the source and target of pointers loaded or stored.

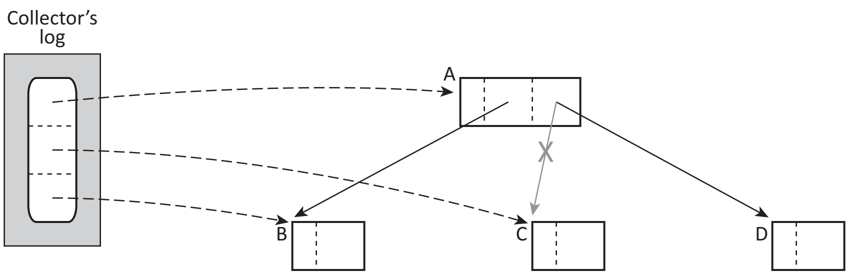
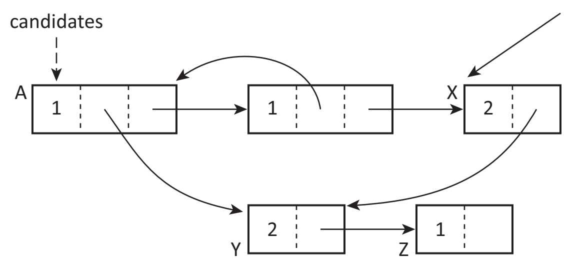
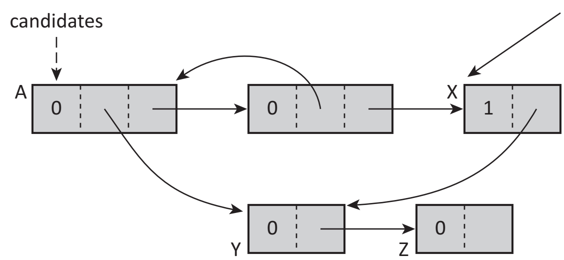


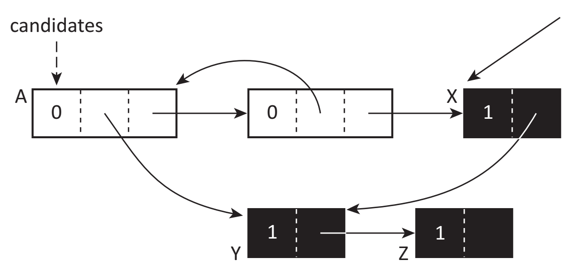

**Figure 5.2:** Coalesced reference counting: if A was modified in the previous epoch, for example by overwriting the reference to C with a reference to D, A's reference fields will have been copied to the log. The old referent C can be found in the collector's log and the most recent new referent D can be found directly from A.




(a) Before `markGrey`.




(b) After `markGrey`, all objects reachable from a candidate object have been marked grey and the effect of references internal to this grey subgraph have been removed. Note that X, which is still reachable, has a non-zero reference count.




(c) After `scan`, all reachable objects are black and their reference counts have been corrected to reflect live references.

**Figure 5.3:** Cyclic reference counting. The first field of each object is its reference count.

Figure withheld

**Figure 5.4:** The synchronous Recycler state transition diagram, showing mutator and collector operations and the colours of objects.

With kind permission from Springer Science+Business Media: Bacon and Rajan [2001], figure 3, page 214.

## Chapter 6

# Comparing garbage collectors

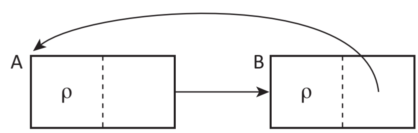
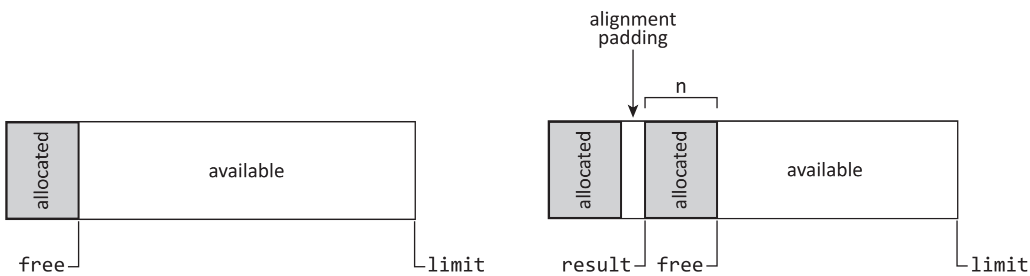



**Figure 6.1:** A simple cycle

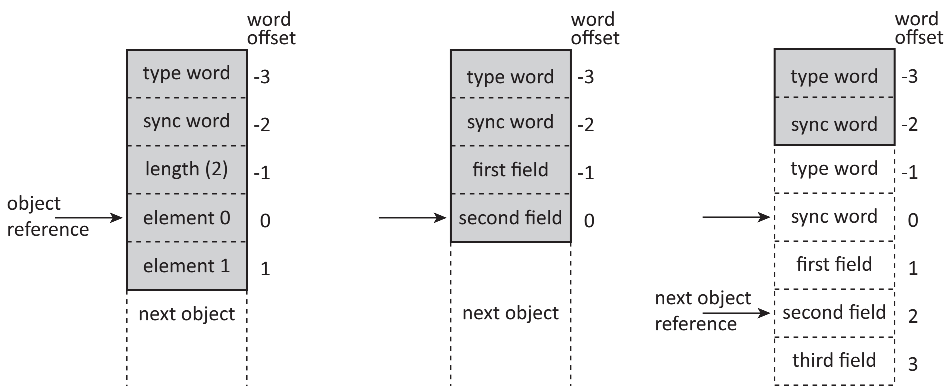
# Chapter 7

# Allocation




(a) Before (b) After

**Figure 7.1:** Sequential allocation: a call to `sequentialAllocate(n)` which advances the `free` pointer by the size of the allocation request, `n`, plus any padding necessary for proper alignment.



(a) Array (b) Scalar (non-Array) (c) No-Fields

**Figure 7.2:** A Java object header design for heap parsability. Grey indicates the words forming the referent object. Neighbouring objects are shown with dashed lines.

# Chapter 8

# Partitioning the heap

# Chapter 9

# Generational garbage collection

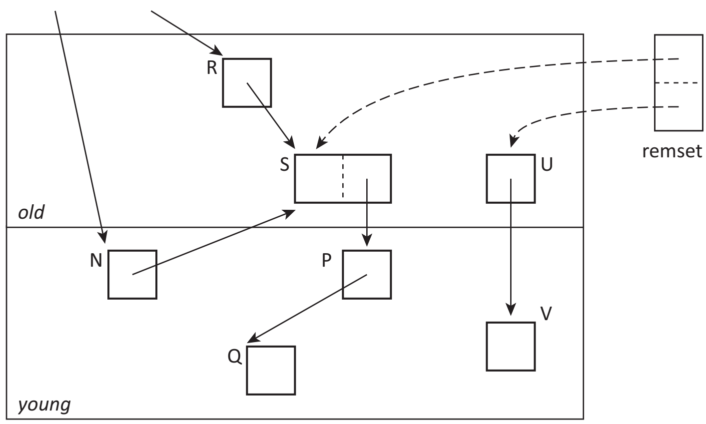



**Figure 9.1:** Intergenerational pointers. If live objects in the young generation are to be preserved without tracing the whole heap, a mechanism and a data structure are needed to remember objects S and U in the old generation that hold references to objects in the young generation.

Figure withheld

**Figure 9.2:** Survival rates with a copy count of 1 or 2. The curves show the fraction of objects that will survive a future collection if they were born at time $x$. Curve (b) shows the proportion that will survive one collection and curve (c) the proportion that will survive two. The coloured areas show the proportions of objects that will be not be copied or will be promoted (copied) under different copy count regimes.

Wilson and Moher [1989b], doi: `10.1145/74877.74882`.

Figure withheld

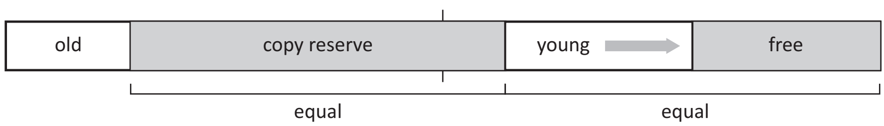
**Figure 9.3:** Shaw's bucket brigade system. Objects are copied within the young generation from a creation space to an aging semispace. By placing the aging semispace adjacent to the old generation at even numbered collections, objects can be promoted to the old generation simply by moving the boundary between generations.

Jones [1996]. Reprinted by permission.

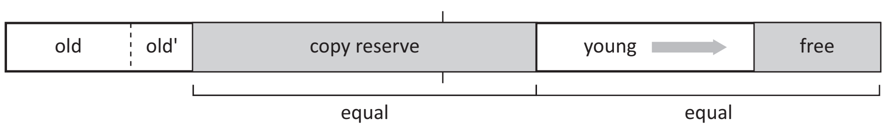
Figure withheld

**Figure 9.4:** High water marks. Objects are copied from a fixed creation space to an aging semispace within a younger generation and then promoted to an older generation. Although all survivors in an aging semispace are promoted, by adjusting a 'high water mark', we can choose to copy or promote an object in the creation space simply through an address comparison.

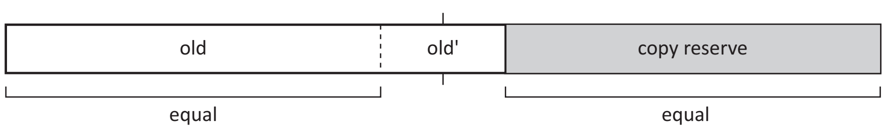
Wilson and Moher [1989b], doi: 10.1145/74877.74882.




(a) Before a minor collection, the copy reserve must be at least as large as the young generation.

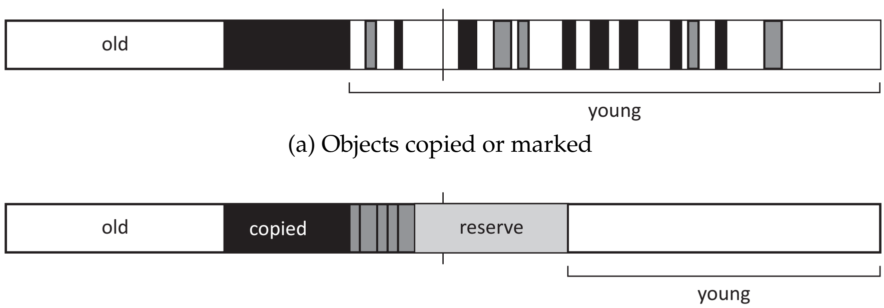



(b) At a minor collection, survivors are copied into the copy reserve, extending the old generation. The copy reserve and young generation are reduced but still of equal size.




(c) After a minor collection and before a major collection. Only objects in the oldest region, old, will be evacuated into the copy reserve. After the evacuation, all live old objects can be moved to the beginning of the heap.

**Figure 9.5:** Appel's simple generational collector. Grey areas are empty.




(a) Objects copied or marked

(b) Marked objects compacted

**Figure 9.6:** Switching between copying and marking the young generation. (a) The copy reserve is full. Black objects from the young generation have been copied into the old generation. Grey objects have been marked but not copied. All other new objects are dead. (b) The compaction pass has slid the grey objects into the old generation.

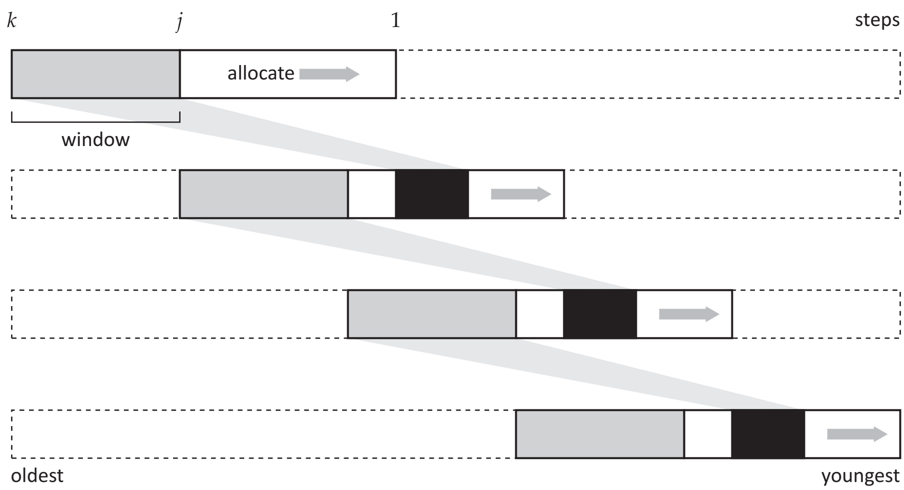



**Figure 9.7:** Renewal Older First garbage collection. At each collection, the objects least recently collected are scavenged and survivors are placed after the youngest objects.

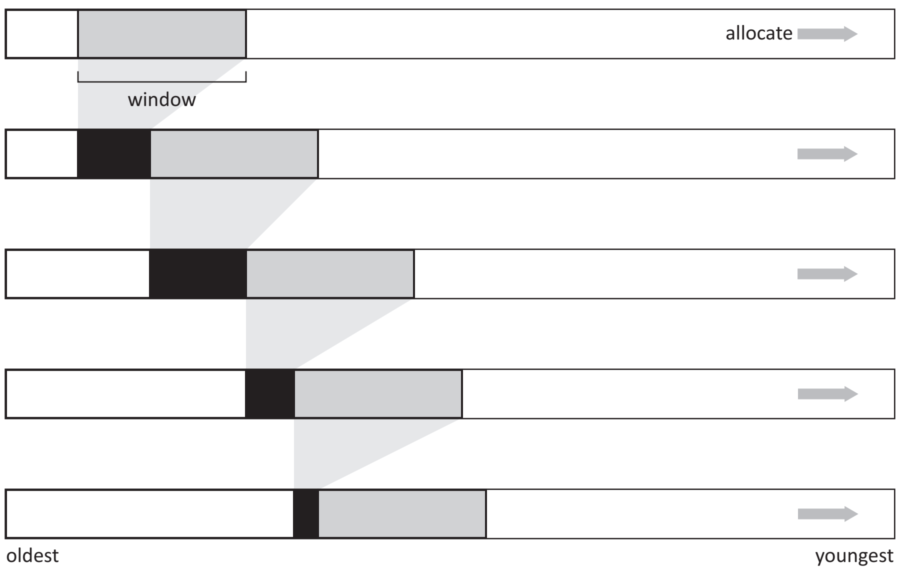



**Figure 9.8:** Deferred Older First garbage collection. A middle-aged window of the heap is selected for collection. Survivors are placed after the survivors of the previous collection. The goal is that the collector will discover a sweet spot, where the survival rate is very low and the window advances very slowly.

Figure withheld

(a) Semispaces

Figure withheld

(b) Fixed nursery generational

Figure withheld

(c) Appel-style generational

Figure withheld

(d) Aging semispaces

Figure withheld

(e) Renewal Older-First

Figure withheld

(f) Deferred Older-First

Figure withheld

(g) Beltway.$X$.$X$

Figure withheld

(h) Beltway.$X$.$X$.100

**Figure 9.9:** Beltway can be configured as any copying collector. Each figure shows the increment used for allocation, the increment to be collected and the increment to which survivors will be copied for each configuration.

Blackburn *et al* [2002], doi: `10.1145/512529.512548`.

## Chapter 10

# Other partitioned schemes

Figure withheld

**Figure 10.1:** The Treadmill collector: objects are held on a double-linked list. Each of the four segments hold objects of a different colour, so that the colour of an object can be changed by 'unsnapping' it from one segment and 'snapping' it into another. The pointers controlling the Treadmill are the same as for other incremental copying collectors [Baker, 1978]: scanning is complete when `scan` meets `T`, and memory is exhausted when `free` meets `B`.

Jones [1996]. Reprinted by permission.

Figure withheld

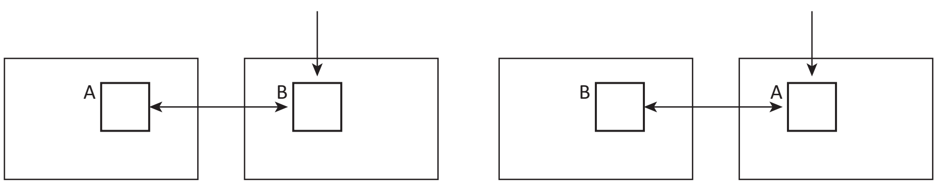
(a) Before collecting car 1, train 1 (T1C1).

Figure withheld

(b) After collecting car 1, train 1. X moved to the same car as its referent Y, A and B to a fresh train T3. The next collection cycle will isolate T2 and reclaim it wholesale. Numbered labels show the copies made in each algorithm step.

**Figure 10.2:** The Train copying collector.

Jones [1996]. Reprinted by permission.




(a) Before collecting the first car (b) Before collecting the next car

**Figure 10.3:** A 'futile' collection. After a collection which moves A to a fresh car, the external reference is updated to refer to A rather than B. This presents the same situation to the collector as before, so no progress can be made.

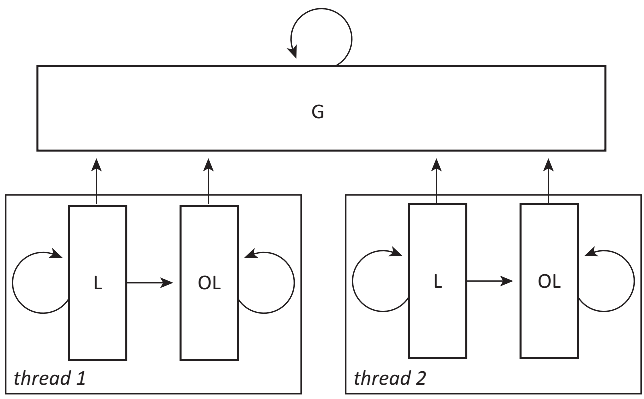



**Figure 10.4:** Thread-local heaplet organisation, indicating permitted pointer directions between purely local (L), optimistically-local (OL) and shared heaplets (G) [Jones and King, 2005].

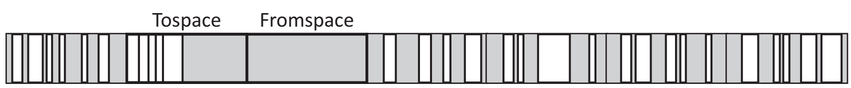
Figure withheld

**Figure 10.5:** A continuum of tracing collectors. Spoonhower *et al* contrast an *evacuation threshold* — sufficient live data to make a block a candidate for evacuation — with an *allocation threshold* — the fraction of a block's free space reused for allocation.

Spoonhower *et al* [2005], doi: `10.1145/1064979.1064989`.

Figure withheld

(a) Before collection




(b) After collection

**Figure 10.6:** Incremental incrementally compacting garbage collection. One space (fromspace) is chosen for evacuation to an empty space (tospace), shown as grey; the other spaces are collected in place. By advancing the two spaces, the whole heap is eventually collected.

Jones [1996]. Reprinted by permission.

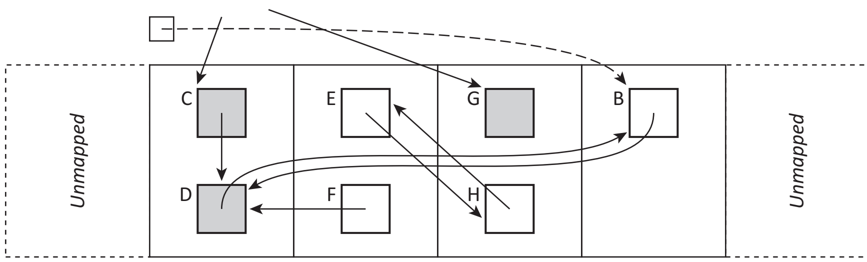
Figure withheld

**Figure 10.7:** Allocation in immix, showing *blocks* of *lines*. Immix uses bump pointer allocation within a partially empty block of small objects, advancing `lineCursor` to `lineLimit`, before moving onto the next group of unmarked lines. It acquires wholly empty blocks in which to bump-allocate medium-sized objects. Immix marks both objects and lines. Because a small object may span two lines (but no more), immix treats the line after any sequence of (explicitly) marked line as implicitly marked: the allocator will not use it.

Blackburn and McKinley [2008], doi: `10.1145/1375581.1375586`.

Figure withheld

(a) After marking (live objects are shown grey).




(b) After the first copying pass. B has been evacuated and the first block has been unmapped.

Figure withheld

(c) After the second copying pass. Note that there was sufficient room to evacuate three blocks.

**Figure 10.8:** Mark-Copy divides the space to be collected into blocks. After the mark phase has constructed a remembered set of objects containing pointers that span blocks, the blocks are evacuated and unmapped, one at a time.

Sachindran and Moss [2003], doi: `10.1145/949305.949335`.

Figure withheld

**Figure 10.9:** Ulterior reference counting schematic: the heap is divided into a space that is managed by reference counting and one that is not. The schematic shows whether reference counting operations on pointer loads or stores should be performed eagerly, deferred or ignored.

Blackburn and McKinley [2003], doi: `10.1145/949305.949336`.

# Chapter 11

# Run-time interface

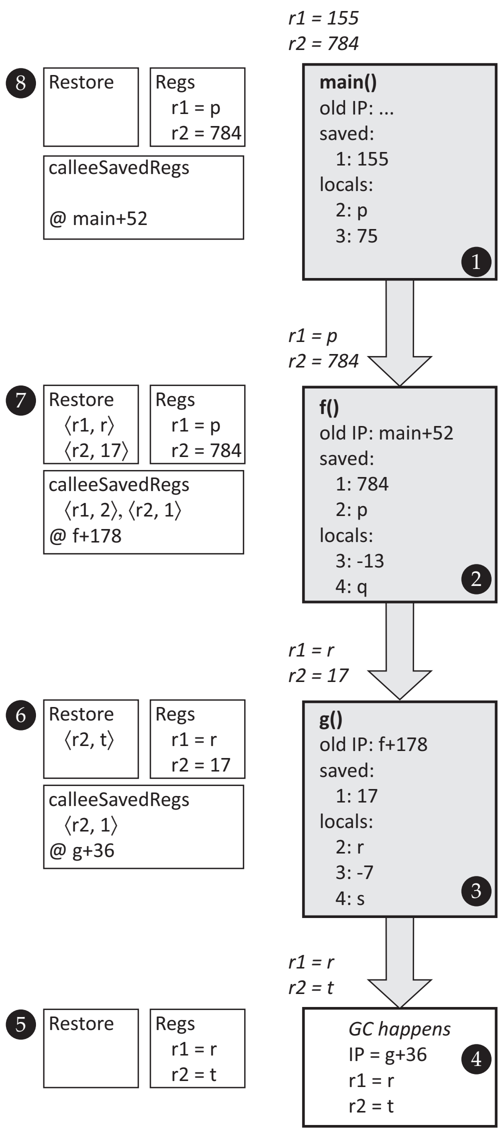
Figure withheld

To determine whether a value `p` is a pointer to an allocated object:

1. Does `p` point between the lowest and highest plausible heap addresses?
2. Use high order bits of `p` as an index into the first-level table to obtain the second-level table. In a 64-bit address space, the top-level table is a chained hash table rather than an array.
3. Use middle order bits of `p` as an index into the second-level table to get the block header.
4. Is the offset of the supposed object a multiple of `hb_size` from the start of the block?
5. Consult the object map for blocks of this size; has the slot corresponding to this object in this block been allocated?

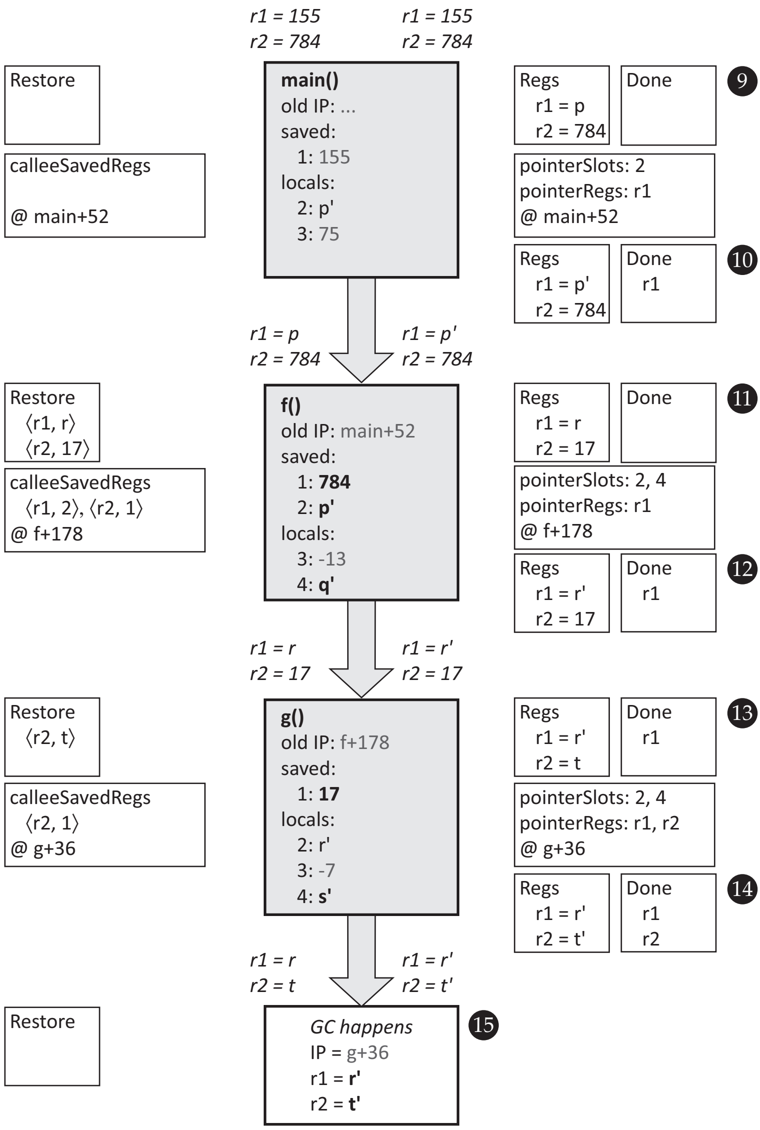
**Figure 11.1:** Conservative pointer finding. The two-level search tree, block header and map of allocated blocks in the Boehm-Demers-Weiser conservative collector.

Jones [1996]. Reprinted by permission.




(a) Stack scanning: walking from the top

**Figure 11.2:** Stack scanning




(b) Stack scanning: walking back to the top

**Figure 11.2 (continued):** Stack scanning

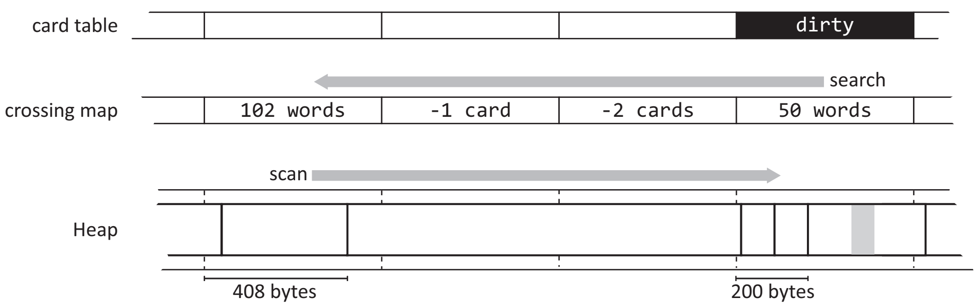



**Figure 11.3:** Crossing map with slot-remembering card table. One card has been dirtied (shown in black). The updated field is shown in grey. The crossing map shows offsets (in words) to the last object in a card.

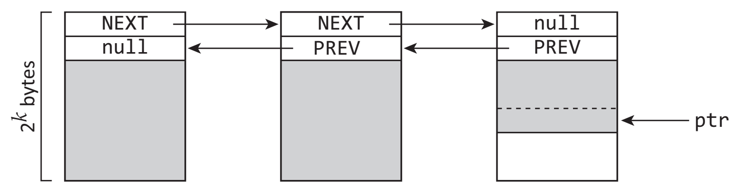



**Figure 11.4:** A stack implemented as a chunked list. Shaded slots contain data. Each chunk is aligned on a $2^k$ byte boundary.

## Chapter 12

# Language-specific concerns

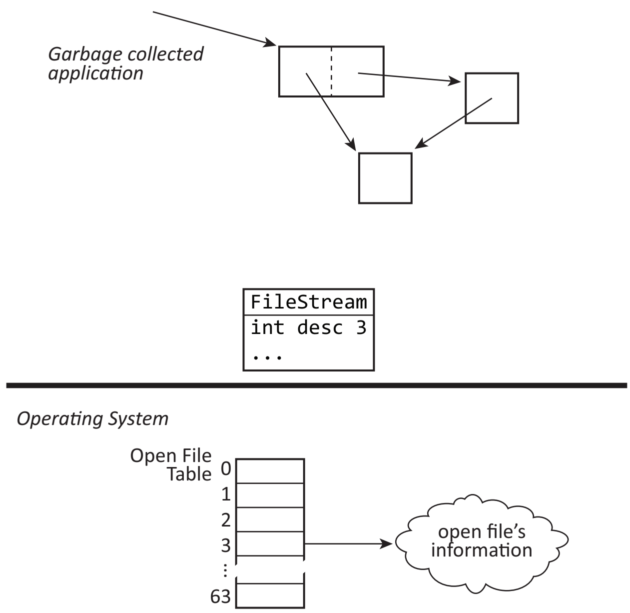



**Figure 12.1:** Failure to release a resource: a `FileStream` object has become unreachable, but its file descriptor has not been closed.

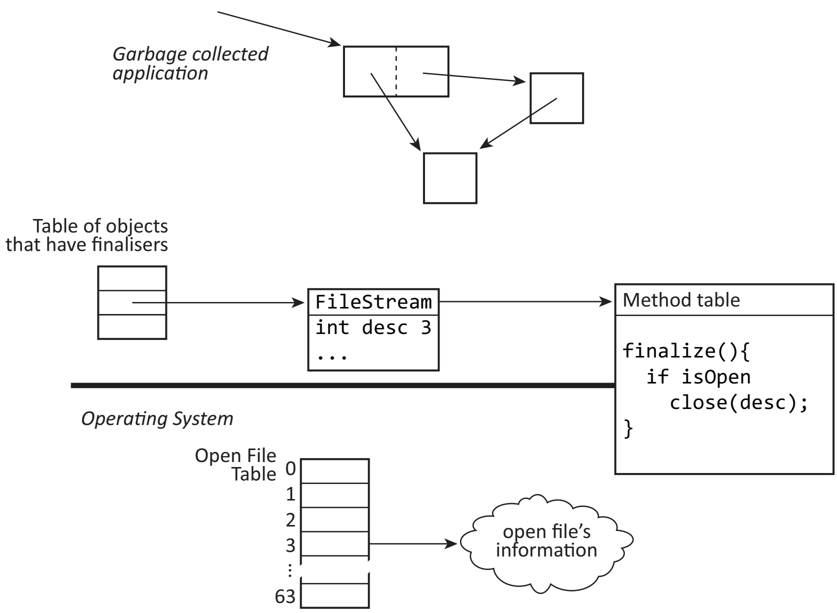



**Figure 12.2:** Using a finaliser to release a resource: here, an unreachable `FileStream` object has a finaliser to close the descriptor.

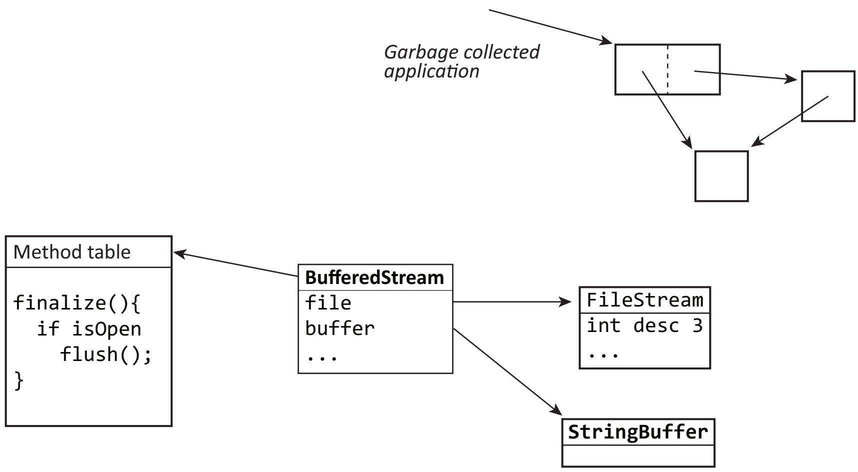
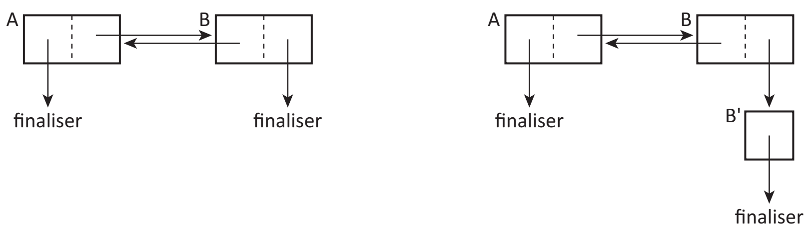



**Figure 12.3:** Object finalisation order. Unreachable `BufferedStream` and `FileStream` objects, which must be finalised in that order.




(a) Original order (b) Refactored to handle cycles

**Figure 12.4:** Restructuring to force finalisation order in cyclic object graphs

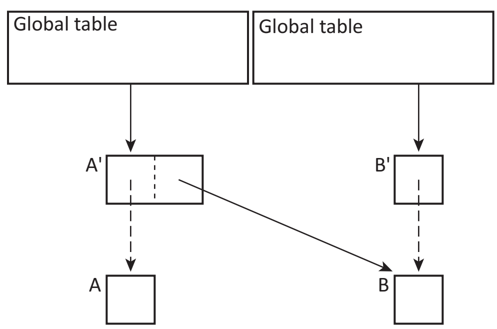


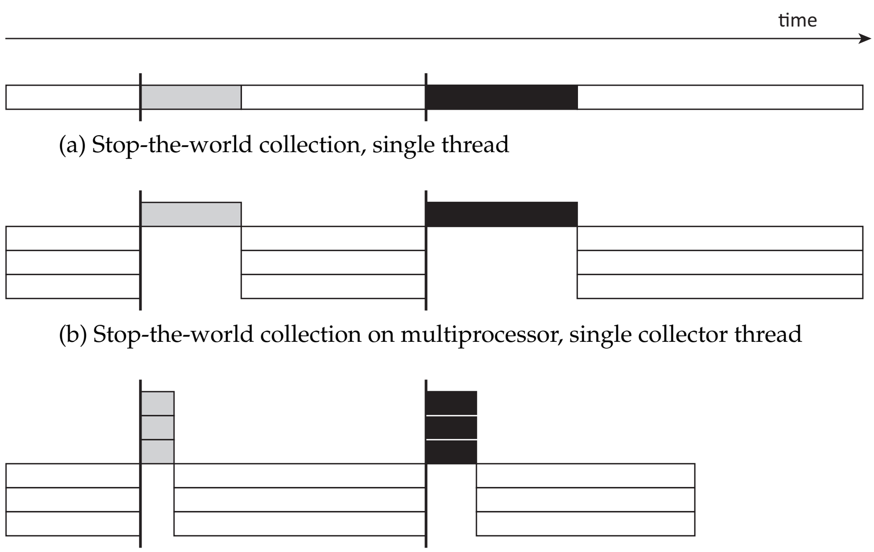

**Figure 12.5:** Finalising in order. Application objects A and B are unreachable from the application and we want to finalise them in that order. Phantom A′ has a phantom reference to A and a strong reference to B.

## Chapter 13

# Concurrency preliminaries

# Chapter 14

# Parallel garbage collection




(a) Stop-the-world collection, single thread

(b) Stop-the-world collection on multiprocessor, single collector thread

(c) Stop-the-world parallel collection

**Figure 14.1:** Stop-the-world garbage collection: each bar represents an execution on a single processor. The coloured regions represent different garbage collection cycles.

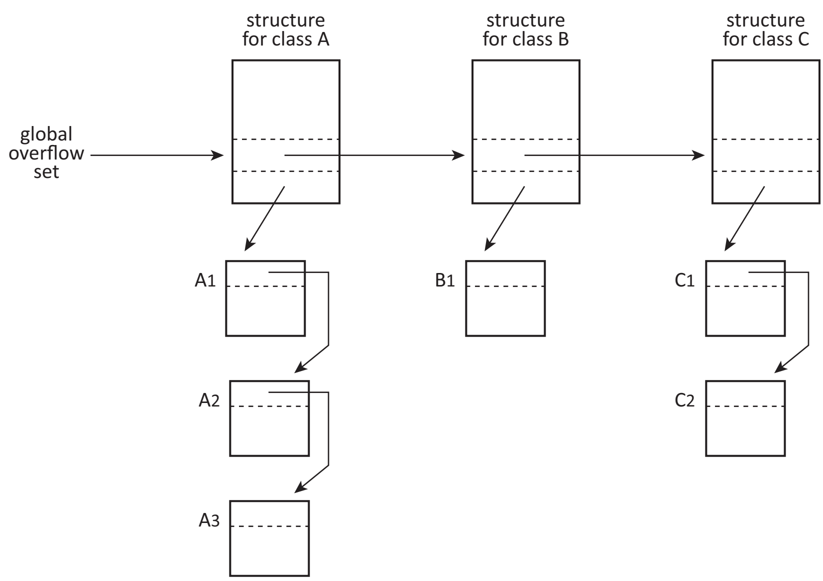



**Figure 14.2:** Global overflow set implemented as a list of lists [Flood *et al*, 2001]. The class structure for each Java class holds the head of a list of overflow objects of that type, linked through the class pointer field in their header.

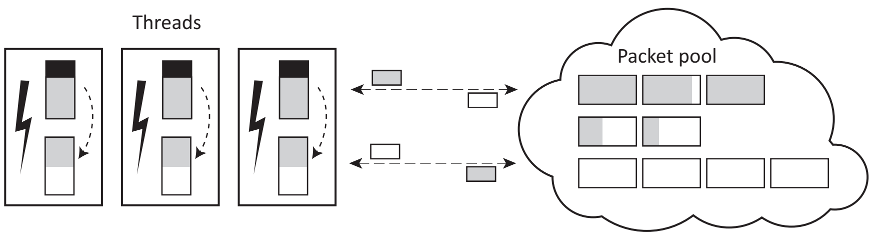



**Figure 14.3:** Grey packets. Each thread exchanges an empty packet for a packet of references to trace. Marking fills an empty packet with new references to trace; when it is full, the thread exchanges it with the global pool for another empty packet.

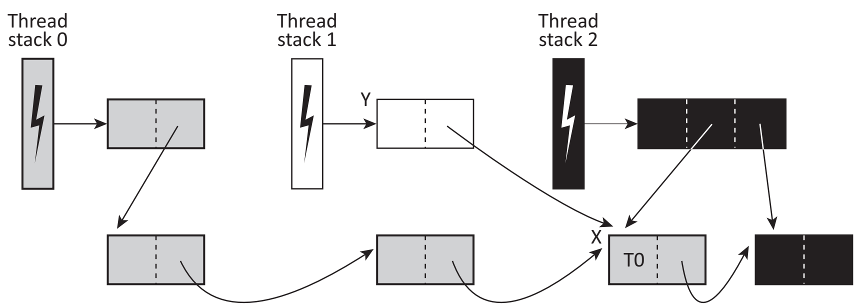
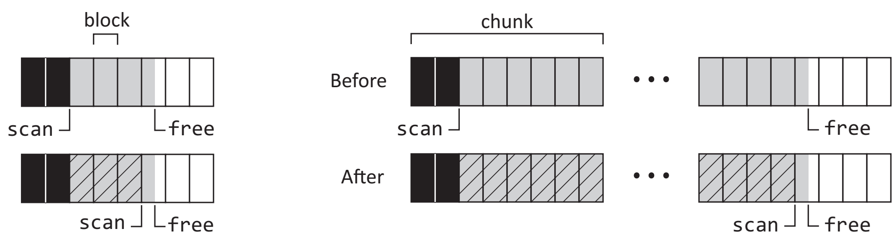



**Figure 14.4:** Dominant-thread tracing. Threads 1 to 3, coloured black, grey and white respectively, have traced a graph of objects. Each object is coloured to indicate the processor to which it will be copied. The first field of each object is its header. Thread T0 was the last to lock object X.




(a) Scan pointer and free pointer in the same chunk

(b) Scan pointer and free pointer in different chunks

**Figure 14.5:** Chunk management in the Imai and Tick [1993] parallel copying collector, showing selection of a scan block before (above) and after (below) overflow. Hatching denotes blocks that have been added to the global pool.

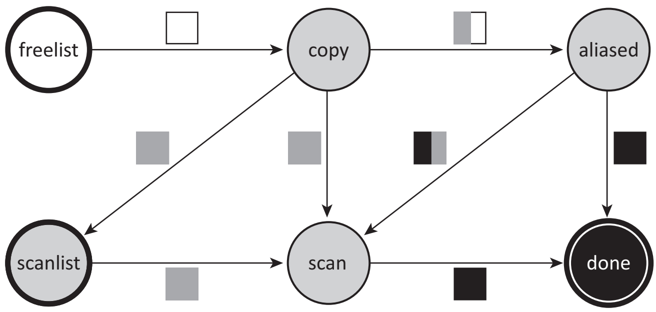



**Figure 14.6:** Block states and transitions in the Imai and Tick [1993] collector. Blocks in states with thick borders are part of the global pool, those with thin borders are owned by a thread.

| copy | scan | | |
|---|---|---|---|
| | aliased | ▒ or ■▒ | ■ |
| ▒□ | (continue scanning) | (continue scanning) | scan → done<br>copy → aliased |
| □ | aliased → copy<br>scanlist → scan | (continue scanning) | scan → done<br>scanlist → scan |
| ■□ | aliased → copy<br>scanlist → scan | (cannot happen) | (cannot happen) |
| ▒ | aliased → scan<br>freelist → copy | copy → scanlist<br>freelist → copy | scan → done<br>copy → scan<br>freelist → copy |
| ■▒ | aliased → scan<br>freelist → copy | (cannot happen) | (cannot happen) |
| ■ | aliased → done<br>freelist → copy<br>scanlist → scan | (cannot happen) | (cannot happen) |

**Table 14.1:** State transition logic for the Imai and Tick collector

Figure withheld

**Figure 14.7:** Block states and transitions in the Siegwart and Hirzel collector. Blocks in states with thick borders are part of the global pool, those with thin borders are local to a thread. A thread may retain one block of the *scanlist* in its local cache.

Siegwart and Hirzel [2006], doi: 10.1145/1133956.1133964.


Figure withheld

**Table 14.2:** State transition logic for the Siegwart and Hirzel collector.

Siegwart and Hirzel [2006], doi: 10.1145/1133956.1133964.

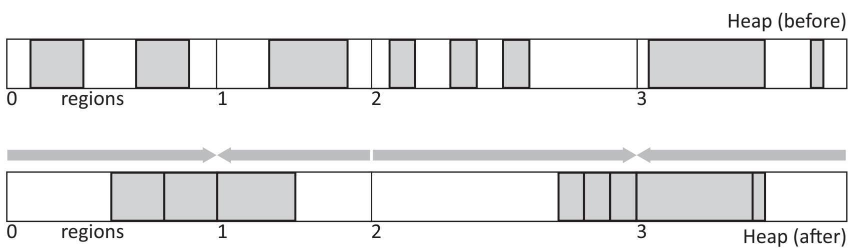



**Figure 14.8:** Flood *et al* [2001] divide the heap into one region per thread and alternate the direction in which compacting threads slide live objects (shown in grey).

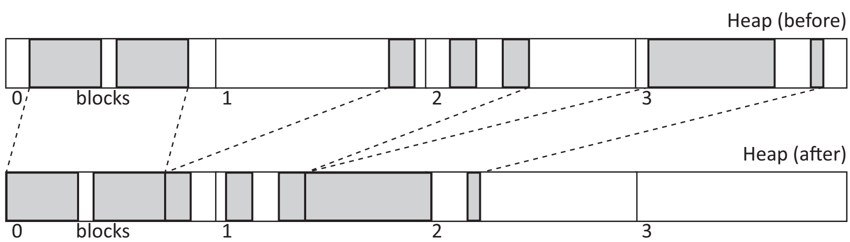
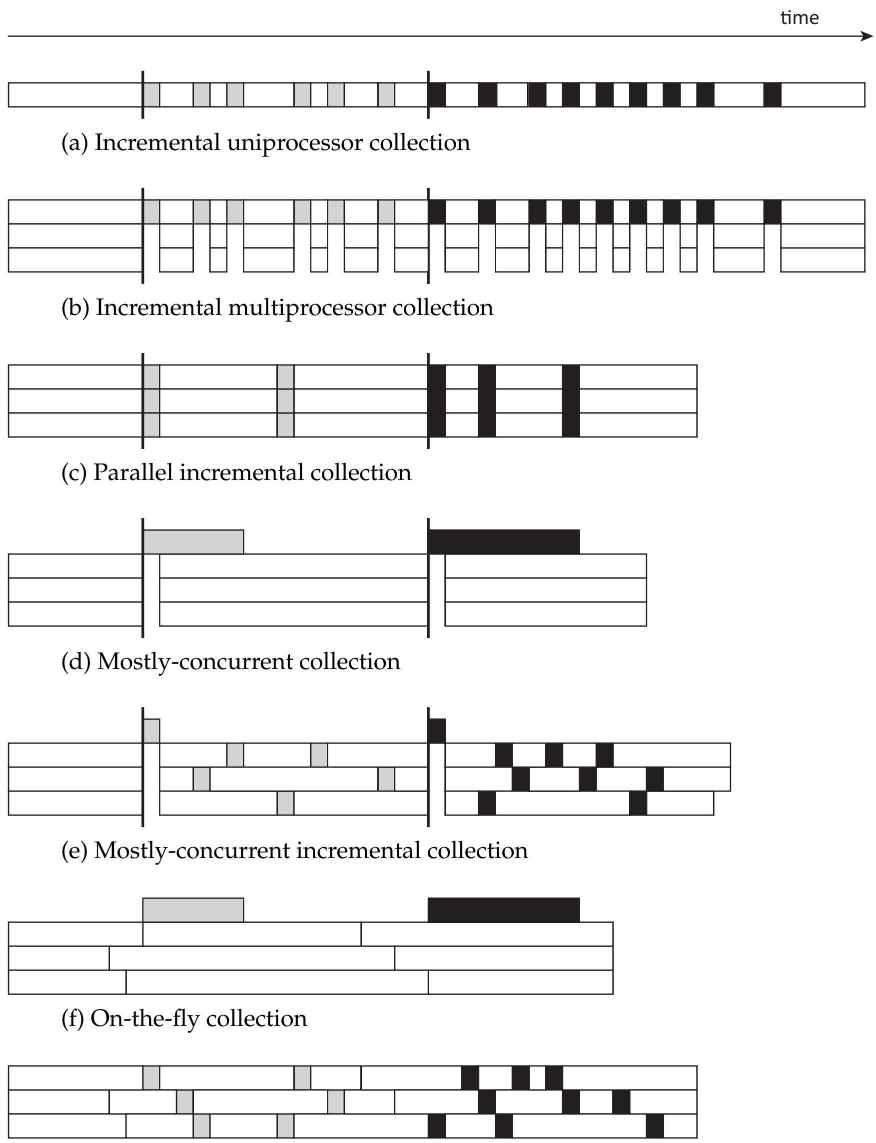



**Figure 14.9:** Inter-block compaction. Rather than sliding object by object, Abuaiadh *et al* [2004] slide only complete blocks: free space within each block is not squeezed out.

Chapter 15

# Concurrent garbage collection




(a) Incremental uniprocessor collection

(b) Incremental multiprocessor collection

(c) Parallel incremental collection

(d) Mostly-concurrent collection

(e) Mostly-concurrent incremental collection

(f) On-the-fly collection

(g) On-the-fly incremental collection

**Figure 15.1:** Incremental and concurrent garbage collection. Each bar represents an execution on a single processor. The coloured regions represent different garbage collection cycles.

Figure withheld

D1: `Write`(X, b, `Read`(Y, a))

Figure withheld

D2: `Write`(Y, a, **null**)

Figure withheld

D3: `scan`(Y)

Figure withheld

(a) Direct: hiding a reachable white object by dropping a direct link from grey.

Figure withheld

T1: `Write`(P, e, `Read`(R, d))

Figure withheld

T2: `Write`(Q, c, **null**)

Figure withheld

T3: `scan`(Q)

Figure withheld

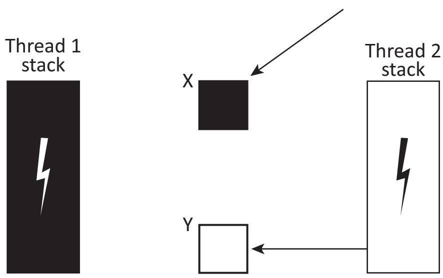
(b) Transitive: hiding a transitively reachable white object by breaking an indirect chain from grey.

**Figure 15.2:** The lost object problem: a reachable white object is hidden from the collector by making it unreachable from any grey object.

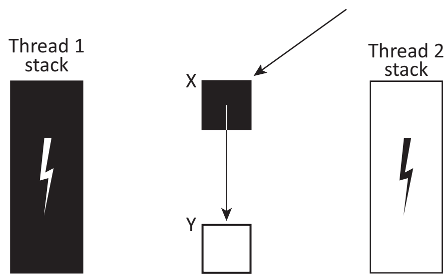
With kind permission from Springer Science+Business Media: Vechev *et al* [2005], figures 3–4, pages 584–5.

# Chapter 16

# Concurrent mark-sweep




(a) The deletion barrier is 'on'. Thread 1 has been scanned, but thread 2 has not. X has been newly allocated black.

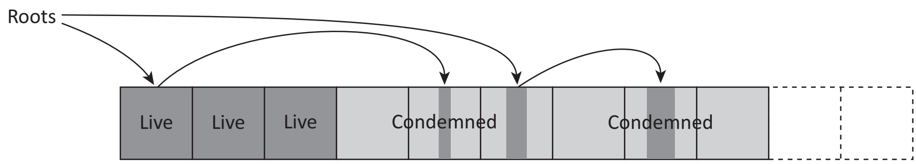



(b) X is updated to point to Y; thread 2's reference to Y is removed. Neither action triggers a deletion barrier.

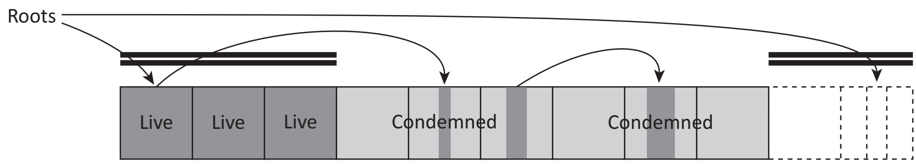
**Figure 16.1:** On-the-fly collectors that allocate black need more than a deletion barrier to prevent the scenario of a white object reachable only from a black object

## Chapter 17

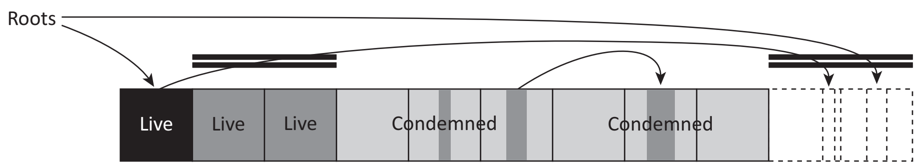
# Concurrent copying & compaction



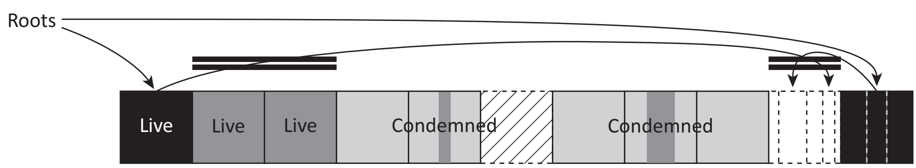

(a) Initial Compressor configuration. All pages are in fromspace.



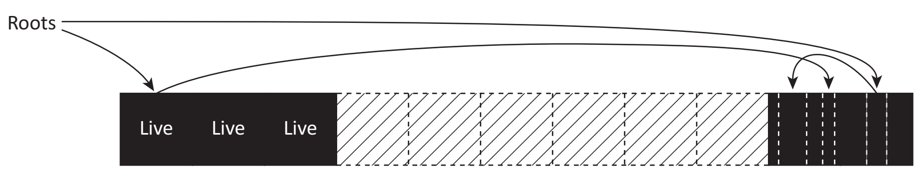

(b) Compute forwarding information, protect all tospace pages (illustrated by the double horizontal bars). These include those reserved to hold evacuated objects and those Live pages not condemned for evacuation. Then flip mutator roots to tospace. Mutators accessing a protected tospace page will now trap.




(c) Trapping on a Live page forwards pointers contained in that page to refer to their tospace targets. Unprotect the Live page once all its stale fromspace references have been replaced with tospace references.

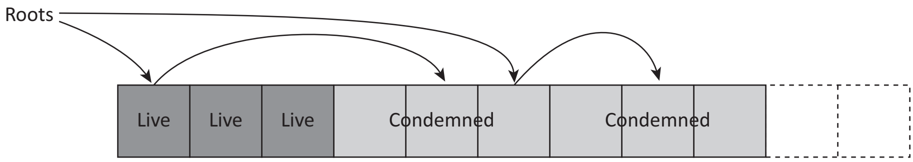



(d) Trapping on a reserved tospace page evacuates objects from fromspace pages to fill the page. The fields of these objects are updated to point to tospace. Unprotect the tospace page and unmap fully-evacuated fromspace pages (releasing their physical pages, shown as hatched).

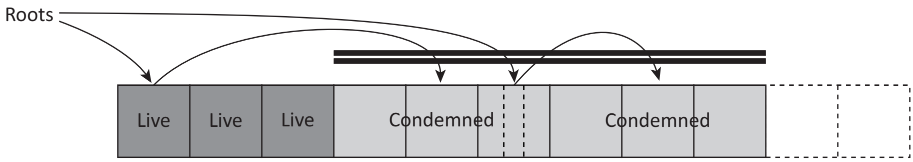



(e) Compaction is finished when all Live pages have been scanned to forward references they contain, and all live objects in condemned pages have been copied into tospace and the references they contain have been forwarded.

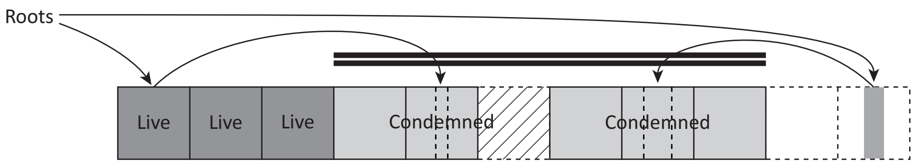
**Figure 17.1:** Compressor



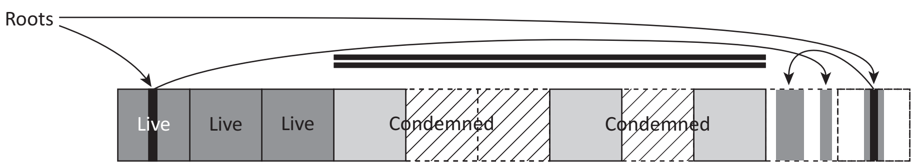

(a) Initial Pauseless configuration. All pages are in fromspace.



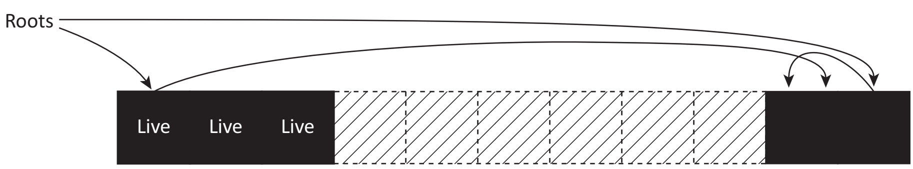

(b) Compute forwarding information, protect all condemned fromspace pages (illustrated by the double horizontal bars), but leave tospace pages unprotected. These include those reserved to hold evacuated objects and those Live pages not condemned for evacuation.




(c) Flip mutator roots to tospace, copying their targets, but leaving the references they contain pointing to fromspace. Mutators accessing an object on a protected fromspace page will trap and wait until the object is copied.




(d) Mutators loading a reference to a protected page will now trigger a GC-trap via the read barrier, copying their targets.




(e) Compaction is finished when all live objects in condemned pages have been copied into tospace, and all tospace pages have been scanned to forward references they contain.

**Figure 17.2:** Pauseless

# Chapter 18

# Concurrent reference counting

| **Thread 1** `Write(o,i,x)` | **Thread 2** `Write(o,i,y)` |
|---|---|
| `addReference(x)` | `addReference(y)` |
| `old ←o[i]` | `old ←o[i]` |
| `deleteReference(old)` | `deleteReference(old)` |
| `o[i]←x` | `o[i]←y` |

**Figure 18.1:** Reference counting must synchronise the manipulation of counts with pointer updates. Here, two threads race to update an object field. Note that `old` is a local variable of each thread's `Write` method.

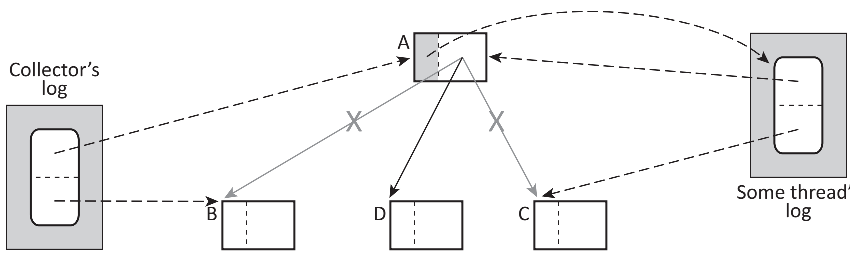



**Figure 18.2:** Concurrent coalesced reference counting: in the previous epoch A was modified to point to C and the values of its reference fields logged. However, A has been modified again in this epoch (to point to D), and so marked dirty and logged again. The original referent B can be found in the collector's global log, just as in Figure 5.2. The reference to C that was added in the previous epoch will be in some thread's current log: this log can be found from A's `getLogPointer` field.

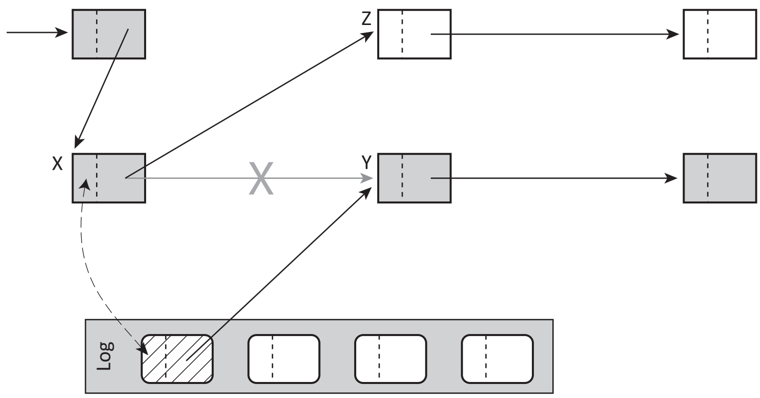



**Figure 18.3:** Sliding views allow a fixed *snapshot* of the graph to be traced by using values stored in the log. Here, the shaded objects indicate the state of the graph at the time that the pointer from X to Y was overwritten to refer to Z. The old version of the graph can be traced by using the value of X's field stored in the log.

# Chapter 19

# Real-time garbage collection

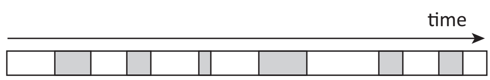



**Figure 19.1:** Unpredictable frequency and duration of conventional collectors. Collector pauses in grey.

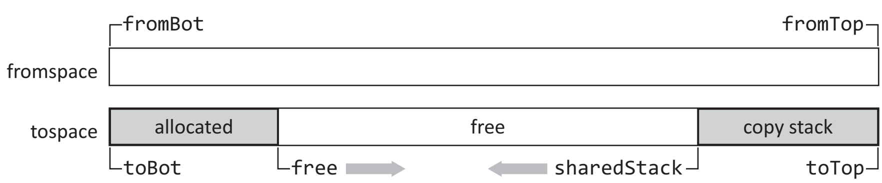



**Figure 19.2:** Heap structure in the Blelloch and Cheng work-based collector

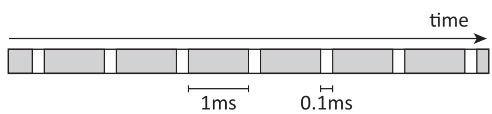



**Figure 19.3:** Low mutator utilisation even with short collector pauses. The mutator (white) runs infrequently while the collector (grey) dominates.

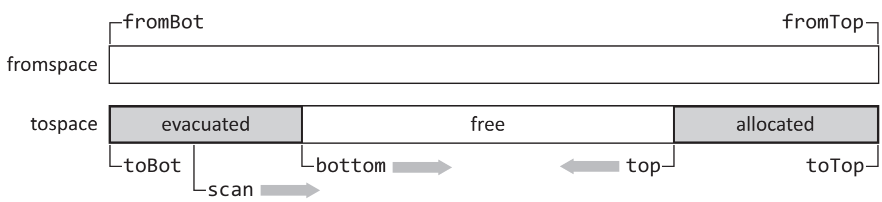



**Figure 19.4:** Heap structure in the Henriksson slack-based collector

Figure withheld

(a) Before a high-priority task performs B.y←A.x. The write barrier catches the assignment since the fromspace C object is not previously evacuated or scheduled for evacuation.

Figure withheld

(b) After having reserved a tospace location for C. A temporary `toAddress` pointer (dashed) to the reserved area prevents multiple tospace reservations for C. Forwarding pointers prevent access to the uninitialised reserved space.

Figure withheld

(c) When the high-priority task pauses, the collector finishes evacuating C to its reserved tospace location, and sets the forwarding pointers to refer to the tospace copy. A.x will be forwarded later when the A object is scanned by the collector.

**Figure 19.5:** Lazy evacuation in the Henriksson slack-based collector.

Henriksson [1998]. Reprinted by permission.

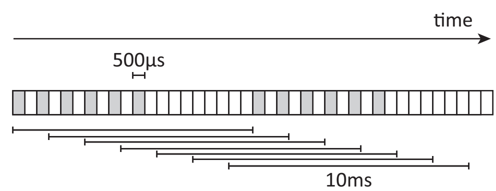



**Figure 19.6:** Metronome utilisation. Collector quanta are shown in grey and mutator quanta in white.

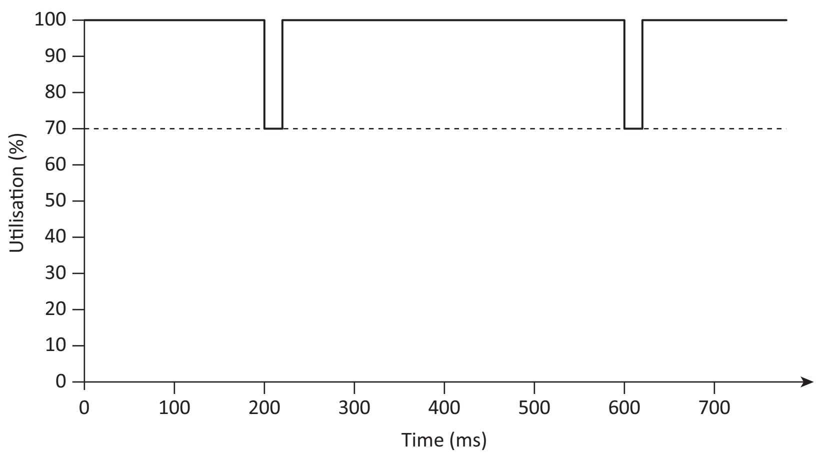



**Figure 19.7:** Overall mutator utilisation in Metronome

Figure withheld

**Figure 19.8:** Mutator utilisation in Metronome during a collection cycle.

First published on IBM developerWorks: `http://www.ibm.com/developerworks`.

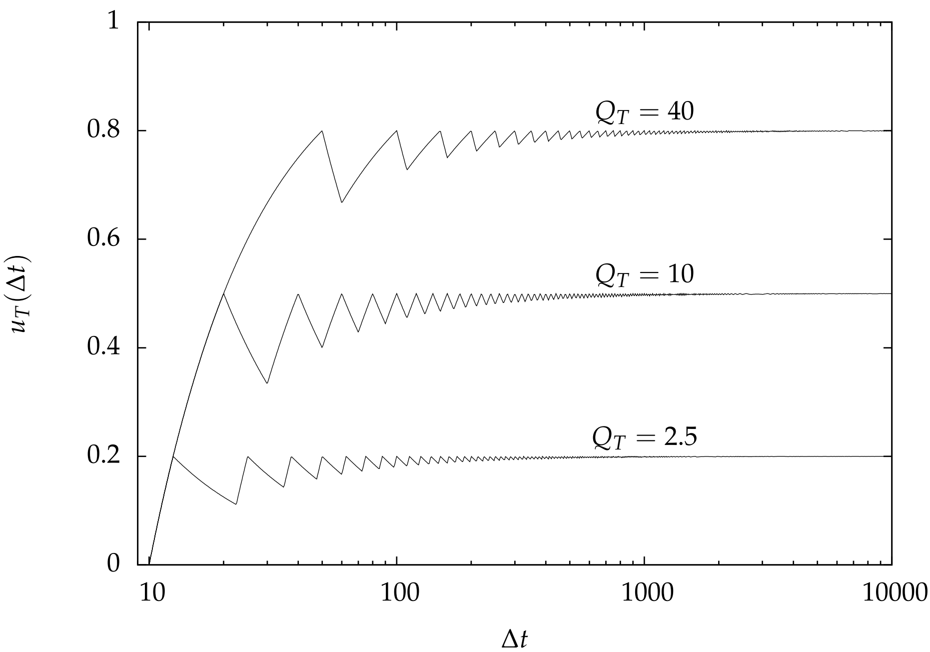



**Figure 19.9:** Minimum mutator utilisation $u_T(\Delta t)$ for a perfectly scheduled time-based collector. $C_T = 10$. Utilisation converges to $\frac{Q_T}{Q_T+C_T}$. Increasing the frequency of the collector (reducing the mutator quantum) produces faster convergence.

Figure withheld

(a) A two-fragment object with a payload of from six to twelve words. The sentinel fragment has three header words: a fragmentation pointer to the next object fragment, a garbage collection header and a type header. Each fragment has a header pointing to the next.

Figure withheld

(b) A single-fragment array with a payload of up to four words. The sentinel fragment has four header words: a **`null`** fragmentation pointer, a garbage collection header, a type header and an actual length $n \leq 4$ words, followed by the inlined array fields.

Figure withheld

(c) A multi-fragment array with a payload of up to three fragments (up to 24 words). The sentinel fragment has five header words: a non-**`null`** fragmentation pointer to the inlined array spine, a garbage collection header, a type header, a pseudo-length `0` indicating fragmentation and an actual length $4 < n \leq 24$ words (at the same negative offeset from the spine as in (b)), followed by the inlined spine. Payload fragments have no headers.

Figure withheld

(d) An array with a payload of four or more fragments (more than 24 words). The sentinel fragment has four header words: a non-**`null`** fragmentation pointer to the separately allocated array spine, a garbage collection header, a type header and a pseudo-length `0` indicating fragmentation, followed by the rest of the sentinel which is unused. The spine has a two-word header: the actual length and a forwarding pointer, at negative offsets. Payload fragments have no headers.

**Figure 19.10:** Fragmented allocation in Schism.

Pizlo *et al* [2010b], doi: `10.1145/1806596.1806615`.

# Bibliography

This bibliography contains over 400 references. However, our comprehensive database at http://www.cs.kent.ac.uk/~rej/gcbib/ contains over 2500 garbage collection related publications. This database can be searched online or downloaded as BibTeX, PostScript or PDF. As well as details of the article, papers, books, theses and so on, the bibliography also contains abstracts for some entries and URLs or DOIs for most of the electronically available ones. We continually strive to keep this bibliography up to date as a service to the community. Here you can help: Richard (R.E.Jones@kent.ac.uk) would be very grateful to receive further entries (or corrections).

Santosh Abraham and J. Patel. Parallel garbage collection on a virtual memory system. In *International Conference on Parallel Processing*, University Park, PA, August 1987, pages 243–246. Pennsylvania State University Press. Also technical report CSRD 620, University of Illinois at Urbana-Champaign, Center for Supercomputing Research and Development.

Diab Abuaiadh, Yoav Ossia, Erez Petrank, and Uri Silbershtein. An efficient parallel heap compaction algorithm. In OOPSLA 2004, pages 224–236. doi: 10.1145/1028976.1028995.

Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. WRL Research Report 95/7, Digital Western Research Laboratory, September 1995.

Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, December 1996. doi: 10.1109/2.546611.

Ole Agesen. GC points in a threaded environment. Technical Report SMLI TR-98-70, Sun Microsystems Laboratories, Palo Alto, CA, 1998.

Rafael Alonso and Andrew W. Appel. An advisor for flexible working sets. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Boulder, CO, May 1990, pages 153–162. ACM Press. doi: 10.1145/98457.98753.

Andrew W. Appel. Garbage collection can be faster than stack allocation. *Information Processing Letters*, 25(4):275–279, 1987. doi: 10.1016/0020-0190(87)90175-X.

Andrew W. Appel. Simple generational garbage collection and fast allocation. *Software: Practice and Experience*, 19(2):171–183, 1989a. doi: 10.1002/spe.4380190206.

Andrew W. Appel. Runtime tags aren't necessary. *Lisp and Symbolic Computation*, 2: 153–162, 1989b. doi: 10.1007/BF01811537.

Andrew W. Appel. Tutorial: Compilers and runtime systems for languages with garbage collection. In PLDI 1992. doi: 10.1145/143095.

Andrew W. Appel. Emulating write-allocate on a no-write-allocate cache. Technical Report TR-459-94, Department of Computer Science, Princeton University, June 1994.

Andrew W. Appel and Zhong Shao. An empirical and analytic study of stack vs. heap cost for languages with closures. Technical Report CS–TR–450–94, Department of Computer Science, Princeton University, March 1994.

Andrew W. Appel and Zhong Shao. Empirical and analytic study of stack versus heap cost for languages with closures. *Journal of Functional Programming*, 6(1):47–74, January 1996. doi: 10.1017/S095679680000157X.

Andrew W. Appel, John R. Ellis, and Kai Li. Real-time concurrent collection on stock multiprocessors. In PLDI 1988, pages 11–20. doi: 10.1145/53990.53992.

J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in Erlang*. Prentice-Hall, second edition, 1996.

Matthew Arnold and Barbara G. Ryder. A framework for reducing the cost of instrumented code. In PLDI 2001, pages 168–179. doi: 10.1145/378795.378832.

Nimar S. Arora, Robert D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. In *10th ACM Symposium on Parallel Algorithms and Architectures*, Puerto Vallarta, Mexico, June 1998, pages 119–129. ACM Press. doi: 10.1145/277651.277678.

Joshua Auerbach, David F. Bacon, Perry Cheng, David Grove, Ben Biron, Charlie Gracie, Bill McCloskey, Aleksandar Micic, and Ryan Sciampacone. Tax-and-spend: Democratic scheduling for real-time garbage collection. In *8th ACM International Conference on Embedded Software*, Atlanta, GA, 2008, pages 245–254. ACM Press. doi: 10.1145/1450058.1450092.

Thomas H. Axford. Reference counting of cyclic graphs for functional programs. *Computer Journal*, 33(5):466–470, 1990. doi: 10.1093/comjnl/33.5.466.

Alain Azagury, Elliot K. Kolodner, and Erez Petrank. A note on the implementation of replication-based garbage collection for multithreaded applications and multiprocessor environments. *Parallel Processing Letters*, 9(3):391–399, 1999. doi: 10.1142/S0129626499000360.

Hezi Azatchi and Erez Petrank. Integrating generations with advanced reference counting garbage collectors. In *12th International Conference on Compiler Construction*, Warsaw, Poland, May 2003, pages 185–199. Volume 2622 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-36579-6_14.

Hezi Azatchi, Yossi Levanoni, Harel Paz, and Erez Petrank. An on-the-fly mark and sweep garbage collector based on sliding views. In OOPSLA 2003, pages 269–281. doi: 10.1145/949305.949329.

Azul. Pauseless garbage collection. White paper AWP–005–020, Azul Systems Inc., July 2008.

Azul. Comparison of virtual memory manipulation metrics. White paper, Azul Systems Inc., 2010.

David F. Bacon and V.T. Rajan. Concurrent cycle collection in reference counted systems. In Jørgen Lindskov Knudsen, editor, *15th European Conference on Object-Oriented Programming*, Budapest, Hungary, June 2001, pages 207–235. Volume 2072 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-45337-7_12.

David F. Bacon, Clement R. Attanasio, Han Bok Lee, V. T. Rajan, and Stephen E. Smith. Java without the coffee breaks: A nonintrusive multiprocessor garbage collector. In PLDI 2001, pages 92–103. doi: 10.1145/378795.378819.

David F. Bacon, Perry Cheng, and V.T. Rajan. A real-time garbage collector with low overhead and consistent utilization. In POPL 2003, pages 285–298. doi: 10.1145/604131.604155.

David F. Bacon, Perry Cheng, and V.T. Rajan. Controlling fragmentation and space consumption in the Metronome, a real-time garbage collector for Java. In LCTES 2003, pages 81–92. doi: 10.1145/780732.780744.

David F. Bacon, Perry Cheng, and V. T. Rajan. A unified theory of garbage collection. In OOPSLA 2004, pages 50–68. doi: 10.1145/1035292.1028982.

David F. Bacon, Perry Cheng, David Grove, and Martin T. Vechev. Syncopation: Generational real-time garbage collection in the Metronome. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, Chicago, IL, June 2005, pages 183–192. ACM SIGPLAN Notices 40(7), ACM Press. doi: 10.1145/1065910.1065937a.

Scott B. Baden. Low-overhead storage reclamation in the Smalltalk-80 virtual machine. In Glenn Krasner, editor, *Smalltalk-80: Bits of History, Words of Advice*, pages 331–342. Addison-Wesley, 1983.

Brenda Baker, E. G. Coffman, and D. E. Willard. Algorithms for resolving conflicts in dynamic storage allocation. *Journal of the ACM*, 32(2):327–343, April 1985. doi: 10.1145/3149.335126.

Henry G. Baker. List processing in real-time on a serial computer. *Communications of the ACM*, 21(4):280–294, 1978. doi: 10.1145/359460.359470. Also AI Laboratory Working Paper 139, 1977.

Henry G. Baker. The Treadmill, real-time garbage collection without motion sickness. *ACM SIGPLAN Notices*, 27(3):66–70, March 1992a. doi: 10.1145/130854.130862.

Henry G. Baker. CONS should not CONS its arguments, or a lazy alloc is a smart alloc. *ACM SIGPLAN Notices*, 27(3), March 1992b. doi: 10.1145/130854.130858.

Henry G. Baker. 'Infant mortality' and generational garbage collection. *ACM SIGPLAN Notices*, 28(4):55–57, April 1993. doi: 10.1145/152739.152747.

Jason Baker, Antonio Cunei, Filip Pizlo, and Jan Vitek. Accurate garbage collection in uncooperative environments with lazy pointer stacks. In *International Conference on Compiler Construction*, Braga, Portugal, March 2007. Volume 4420 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/978-3-540-71229-9_5.

Jason Baker, Antonio Cunei, Tomas Kalibera, Filip Pizlo, and Jan Vitek. Accurate garbage collection in uncooperative environments revisited. *Concurrency and Computation: Practice and Experience*, 21(12):1572–1606, 2009. doi: 10.1002/cpe.1391. Supersedes Baker *et al* [2007].

Katherine Barabash, Yoav Ossia, and Erez Petrank. Mostly concurrent garbage collection revisited. In OOPSLA 2003, pages 255–268. doi: 10.1145/949305.949328.

Katherine Barabash, Ori Ben-Yitzhak, Irit Goft, Elliot K. Kolodner, Victor Leikehman, Yoav Ossia, Avi Owshanko, and Erez Petrank. A parallel, incremental, mostly concurrent garbage collector for servers. *ACM Transactions on Programming Languages and Systems*, 27(6):1097–1146, November 2005. doi: 10.1145/1108970.1108972.

David A. Barrett and Benjamin Zorn. Garbage collection using a dynamic threatening boundary. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, La Jolla, CA, June 1995, pages 301–314. ACM SIGPLAN Notices 30(6), ACM Press. doi: 10.1145/207110.207164.

David A. Barrett and Benjamin G. Zorn. Using lifetime predictors to improve memory allocation performance. In PLDI 1993, pages 187–196. doi: 10.1145/155090.155108.

Joel F. Bartlett. Compacting garbage collection with ambiguous roots. WRL Research Report 88/2, DEC Western Research Laboratory, Palo Alto, CA, February 1988a. Also appears as Bartlett [1988b].

Joel F. Bartlett. Compacting garbage collection with ambiguous roots. *Lisp Pointers*, 1(6): 3–12, April 1988b. doi: 10.1145/1317224.1317225.

Joel F. Bartlett. Mostly-Copying garbage collection picks up generations and C++. Technical Note TN–12, DEC Western Research Laboratory, Palo Alto, CA, October 1989a.

Joel F. Bartlett. SCHEME->C: a portable Scheme-to-C compiler. WRL Research Report 89/1, DEC Western Research Laboratory, Palo Alto, CA, January 1989b.

George Belotsky. C++ memory management: From fear to triumph. O'Reilly linuxdevcenter.com, July 2003.

Mordechai Ben-Ari. Algorithms for on-the-fly garbage collection. *ACM Transactions on Programming Languages and Systems*, 6(3):333–344, July 1984. doi: 10.1145/579.587.

Emery Berger, Kathryn McKinley, Robert Blumofe, and Paul Wilson. Hoard: A scalable memory allocator for multithreaded applications. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000, pages 117–128. ACM SIGPLAN Notices 35(11), ACM Press. doi: 10.1145/356989.357000.

Peter B. Bishop. *Computer Systems with a Very Large Address Space and Garbage Collection*. PhD thesis, MIT Laboratory for Computer Science, May 1977. doi: 1721.1/16428. Technical report MIT/LCS/TR–178.

Stephen Blackburn and Kathryn S. McKinley. Immix garbage collection: Mutator locality, fast collection, and space efficiency. In PLDI 2008, pages 22–32. doi: 10.1145/1375581.1375586.

Stephen Blackburn, Robin Garner, Chris Hoffman, Asjad M. Khan, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiederman. The DaCapo benchmarks: Java benchmarking development and analysis (extended version). Technical report, The DaCapo Group, 2006a.

Stephen Blackburn, Robin Garner, Kathryn S. McKinley, Amer Diwan, Samuel Z. Guyer, Antony Hosking, J. Eliot B. Moss, Darko Stefanović, *et al*. The DaCapo benchmarks: Java benchmarking development and analysis. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, October 2006b, pages 169–190. ACM SIGPLAN Notices 41(10), ACM Press. doi: 10.1145/1167473.1167488.

Stephen M. Blackburn and Antony L. Hosking. Barriers: Friend or foe? In ISMM 2004, pages 143–151. doi: 10.1145/1029873.1029891.

Stephen M. Blackburn and Kathryn S. McKinley. In or out? putting write barriers in their place. In ISMM 2002, pages 175–184. doi: 10.1145/512429.512452.

Stephen M. Blackburn and Kathryn S. McKinley. Ulterior reference counting: Fast garbage collection without a long wait. In OOPSLA 2003, pages 344–458. doi: 10.1145/949305.949336.

Stephen M. Blackburn, Sharad Singhai, Matthew Hertz, Kathryn S. McKinley, and J. Eliot B. Moss. Pretenuring for Java. In OOPSLA 2001, pages 342–352. doi: 10.1145/504282.504307.

Stephen M. Blackburn, Richard E. Jones, Kathryn S. McKinley, and J. Eliot B. Moss. Beltway: Getting around garbage collection gridlock. In PLDI 2002, pages 153–164. doi: 10.1145/512529.512548.

Stephen M. Blackburn, Perry Cheng, and Kathryn S. McKinley. Myths and realities: The performance impact of garbage collection. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 2004a, pages 25–36. ACM SIGMETRICS Performance Evaluation Review 32(1), ACM Press. doi: 10.1145/1005686.1005693.

Stephen M. Blackburn, Perry Cheng, and Kathryn S. McKinley. Oil and water? High performance garbage collection in Java with MMTk. In *26th International Conference on Software Engineering*, Edinburgh, May 2004b, pages 137–146. IEEE Computer Society Press. doi: 10.1109/ICSE.2004.1317436.

Stephen M. Blackburn, Matthew Hertz, Kathryn S. Mckinley, J. Eliot B. Moss, and Ting Yang. Profile-based pretenuring. *ACM Transactions on Programming Languages and Systems*, 29(1):1–57, 2007. doi: 10.1145/1180475.1180477.

Bruno Blanchet. Escape analysis for object oriented languages: Application to Java. In OOPSLA 1999, pages 20–34. doi: 10.1145/320384.320387.

Ricki Blau. Paging on an object-oriented personal computer for Smalltalk. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Minneapolis, MN, August 1983, pages 44–54. ACM Press. doi: 10.1145/800040.801394. Also appears as Technical Report UCB/CSD 83/125, University of California, Berkeley, Computer Science Division (EECS).

Guy E. Blelloch and Perry Cheng. On bounding time and space for multiprocessor garbage collection. In PLDI 1999, pages 104–117. doi: 10.1145/301618.301648.

Daniel G. Bobrow. Managing re-entrant structures using reference counts. *ACM Transactions on Programming Languages and Systems*, 2(3):269–273, July 1980. doi: 10.1145/357103.357104.

Hans-Juergen Boehm. Mark-sweep vs. copying collection and asymptotic complexity. http://www.hpl.hp.com/personal/Hans_Boehm/gc/complexity.html, September 1995.

Hans-Juergen Boehm. Reducing garbage collector cache misses. In ISMM 2000, pages 59–64. doi: 10.1145/362422.362438.

Hans-Juergen Boehm. Destructors, finalizers, and synchronization. In POPL 2003, pages 262–272. doi: 10.1145/604131.604153.

Hans-Juergen Boehm. The space cost of lazy reference counting. In *31st Annual ACM Symposium on Principles of Programming Languages*, Venice, Italy, January 2004, pages 210–219. ACM SIGPLAN Notices 39(1), ACM Press. doi: 10.1145/604131.604153.

Hans-Juergen Boehm. Space efficient conservative garbage collection. In PLDI 1993, pages 197–206. doi: 10.1145/155090.155109.

Hans-Juergen Boehm and Mike Spertus. Garbage collection in the next C++ standard. In ISMM 2009, pages 30–38. doi: 10.1145/1542431.1542437.

Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Software: Practice and Experience*, 18(9):807–820, 1988. doi: 10.1002/spe.4380180902.

Hans-Juergen Boehm, Alan J. Demers, and Scott Shenker. Mostly parallel garbage collection. In PLDI 1991 [PLDI 1991], pages 157–164. doi: 10.1145/113445.113459.

Michael Bond and Kathryn McKinley. Tolerating memory leaks. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Nashville, TN, October 2008, pages 109–126. ACM SIGPLAN Notices 43(10), ACM Press. doi: 10.1145/1449764.1449774.

Tim Brecht, Eshrat Arjomandi, Chang Li, and Hang Pham. Controlling garbage collection and heap growth to reduce the execution time of Java applications. In OOPSLA 2001, pages 353–366. doi: 10.1145/504282.504308.

Tim Brecht, Eshrat Arjomandi, Chang Li, and Hang Pham. Controlling garbage collection and heap growth to reduce the execution time of Java applications. *ACM Transactions on Programming Languages and Systems*, 28(5):908–941, September 2006. doi: 10.1145/1152649.1152652.

R. P. Brent. Efficient implementation of the first-fit strategy for dynamic storage allocation. *ACM Transactions on Programming Languages and Systems*, 11(3):388–403, July 1989. doi: 10.1145/65979.65981.

Rodney A. Brooks. Trading data space for reduced time and code space in real-time garbage collection on stock hardware. In LFP 1984, pages 256–262. doi: 10.1145/800055.802042.

David R. Brownbridge. Cyclic reference counting for combinator machines. In Jean-Pierre Jouannaud, editor, *Conference on Functional Programming and Computer Architecture*, Nancy, France, September 1985, pages 273–288. Volume 201 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-15975-4_42.

F. Warren Burton. A buddy system variation for disk storage allocation. *Communications of the ACM*, 19(7):416–417, July 1976. doi: 10.1145/360248.360259.

Albin M. Butters. Total cost of ownership: A comparison of C/C++ and Java. Technical report, Evans Data Corporation, June 2007.

Brad Calder, Chandra Krintz, S. John, and T. Austin. Cache-conscious data placement. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 1998, pages 139–149. ACM SIGPLAN Notices 33(11), ACM Press. doi: 10.1145/291069.291036.

D. C. Cann and Rod R. Oldehoeft. Reference count and copy elimination for parallel applicative computing. Technical Report CS–88–129, Department of Computer Science, Colorado State University, Fort Collins, CO, 1988.

Dante Cannarozzi, Michael P. Plezbert, and Ron Cytron. Contaminated garbage collection. In PLDI 2000, pages 264–273. doi: 10.1145/349299.349334.

Luca Cardelli, James Donahue, Lucille Glassman, Mick Jordan, Bill Kalsow, and Greg Nelson. Modula-3 language definition. *ACM SIGPLAN Notices*, 27(8):15–42, August 1992. doi: 10.1145/142137.142141.

Patrick J. Caudill and Allen Wirfs-Brock. A third-generation Smalltalk-80 implementation. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, November 1986, pages 119–130. ACM SIGPLAN Notices 21(11), ACM Press. doi: 10.1145/28697.28709.

CC 2005. *14th International Conference on Compiler Construction*, Edinburgh, April 2005. Volume 3443 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/b107108.

Pedro Celis, Per-Åke Larson, and J. Ian Munro. Robin Hood hashing. In *26th Annual Symposium on Foundations of Computer Science*, Portland, OR, October 1985, pages 261–288. IEEE Computer Society Press. doi: 10.1109/SFCS.1985.48.

Yang Chang. *Garbage Collection for Flexible Hard Real-time Systems*. PhD thesis, University of York, 2007.

Yang Chang and Andy Wellings. Integrating hybrid garbage collection with dual priority scheduling. In *11th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2005, pages 185–188. IEEE Press, IEEE Computer Society Press. doi: 10.1109/RTCSA.2005.56.

Yang Chang and Andy Wellings. Low memory overhead real-time garbage collection for Java. In *4th International Workshop on Java Technologies for Real-time and Embedded Systems*, Paris, France, October 2006a. doi: 10.1145/1167999.1168014.

Yang Chang and Andy J. Wellings. Hard real-time hybrid garbage collection with low memory requirements. In *27th IEEE Real-Time Systems Symposium*, December 2006b, pages 77–86. doi: 10.1109/RTSS.2006.25.

David R. Chase. *Garbage Collection and Other Optimizations*. PhD thesis, Rice University, August 1987. doi: 1911/16127.

David R. Chase. Safety considerations for storage allocation optimizations. In PLDI 1988, pages 1–10. doi: 10.1145/53990.53991.

A. M. Cheadle, A. J. Field, and J. Nyström-Persson. A method specialisation and virtualised execution environment for Java. In David Gregg, Vikram Adve, and Brian Bershad, editors, *4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Seattle, WA, March 2008, pages 51–60. ACM Press. doi: 10.1145/1346256.1346264.

Andrew M. Cheadle, Anthony J. Field, Simon Marlow, Simon L. Peyton Jones, and R.L While. Non-stop Haskell. In *5th ACM SIGPLAN International Conference on Functional Programming*, Montreal, September 2000, pages 257–267. ACM Press. doi: 10.1145/351240.351265.

Andrew M. Cheadle, Anthony J. Field, Simon Marlow, Simon L. Peyton Jones, and Lyndon While. Exploring the barrier to entry — incremental generational garbage collection for Haskell. In ISMM 2004, pages 163–174. doi: 10.1145/1029873.1029893.

Wen-Ke Chen, Sanjay Bhansali, Trishul M. Chilimbi, Xiaofeng Gao, and Weihaw Chuang. Profile-guided proactive garbage collection for locality optimization. In PLDI 2006, pages 332–340. doi: 10.1145/1133981.1134021.

C. J. Cheney. A non-recursive list compacting algorithm. *Communications of the ACM*, 13 (11):677–8, November 1970. doi: 10.1145/362790.362798.

Perry Cheng and Guy Blelloch. A parallel, real-time garbage collector. In PLDI 2001, pages 125–136. doi: 10.1145/378795.378823.

Perry Cheng, Robert Harper, and Peter Lee. Generational stack collection and profile-driven pretenuring. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Montreal, Canada, June 1998, pages 162–173. ACM SIGPLAN Notices 33(5), ACM Press. doi: 10.1145/277650.277718.

Perry Sze-Din Cheng. *Scalable Real-Time Parallel Garbage Collection for Symmetric Multiprocessors*. PhD thesis, Carnegie Mellon University, September 2001. SCS Technical Report CMU-CS-01-174.

Chen-Yong Cher, Antony L. Hosking, and T.N. Vijaykumar. Software prefetching for mark-sweep garbage collection: Hardware analysis and software redesign. In Shubu Mukherjee and Kathryn S. McKinley, editors, *11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, October 2004, pages 199–210. ACM SIGPLAN Notices 39(11), ACM Press. doi: 10.1145/1024393.1024417.

Trishul M. Chilimbi and James R. Larus. Using generational garbage collection to implement cache-conscious data placement. In ISMM 1998, pages 37–48. doi: 10.1145/301589.286865.

Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. Cache-conscious structure layout. In PLDI 1999, pages 1–12. doi: 10.1145/301618.301633.

Hyeonjoong Cho, Chewoo Na, Binoy Ravindran, and E. Douglas Jensen. On scheduling garbage collector in dynamic real-time systems with statistical timing assurances. *Real-Time Systems*, 36(1–2):23–46, 2007. doi: 10.1007/s11241-006-9011-0.

Hyeonjoong Cho, Binoy Ravindran, and Chewoo Na. Garbage collector scheduling in dynamic, multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):845–856, June 2009. doi: 10.1109/TPDS.2009.20.

Dave Christie, Jae-Woong Chung, Stephan Diestelhorst, Michael Hohmuth, Martin Pohlack, Christof Fetzer, Martin Nowack, Torvald Riegel, Pascal Felber, Patrick Marlier, and Etienne Riviere. Evaluation of AMD's advanced synchronization facility within a complete transactional memory stack. In *European Conference on Computer Systems (EuroSys)*, 2010, pages 27–40. ACM Press. doi: 10.1145/1755913.1755918.

Douglas W. Clark and C. Cordell Green. An empirical study of list structure in Lisp. *Communications of the ACM*, 20(2):78–86, February 1977. doi: 10.1145/359423.359427.

Cliff Click, Gil Tene, and Michael Wolf. The Pauseless GC algorithm. In VEE 2005, pages 46–56. doi: 10.1145/1064979.1064988.

Marshall P. Cline and Greg A. Lomow. *C++ FAQs: Frequently Asked Questions*. Addison-Wesley, 1995.

William D. Clinger and Lars T. Hansen. Generational garbage collection and the radioactive decay model. In PLDI 1997, pages 97–108. doi: 10.1145/258915.258925.

Jacques Cohen and Alexandru Nicolau. Comparison of compacting algorithms for garbage collection. *ACM Transactions on Programming Languages and Systems*, 5(4): 532–553, 1983. doi: 10.1145/69575.357226.

George E. Collins. A method for overlapping and erasure of lists. *Communications of the ACM*, 3(12):655–657, December 1960. doi: 10.1145/367487.367501.

W. T. Comfort. Multiword list items. *Communications of the ACM*, 7(6):357–362, June 1964. doi: 10.1145/512274.512288.

Eric Cooper, Scott Nettles, and Indira Subramanian. Improving the performance of SML garbage collection using application-specific virtual memory management. In LFP 1992, pages 43–52. doi: 10.1145/141471.141501.

Erik Corry. Optimistic stack allocation for Java-like languages. In ISMM 2006, pages 162–173. doi: 10.1145/1133956.1133978.

Jim Crammond. A garbage collection algorithm for shared memory parallel processors. *International Journal Of Parallel Programming*, 17(6):497–522, 1988. doi: 10.1007/BF01407816.

David Detlefs. Automatic inference of reference-count invariants. In *2nd Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE)*, Venice, Italy, January 2004a.

David Detlefs. A hard look at hard real-time garbage collection. In *7th International Symposium on Object-Oriented Real-Time Distributed Computing*, Vienna, May 2004b, pages 23–32. IEEE Press. doi: 10.1109/ISORC.2004.1300325. Invited paper.

David Detlefs, William D. Clinger, Matthias Jacob, and Ross Knippel. Concurrent remembered set refinement in generational garbage collection. In *2nd Java Virtual Machine Research and Technology Symposium*, San Francisco, CA, August 2002a. USENIX.

David Detlefs, Christine Flood, Steven Heller, and Tony Printezis. Garbage-first garbage collection. In ISMM 2004, pages 37–48. doi: 10.1145/1029873.1029879.

David L. Detlefs. Concurrent garbage collection for C++. Technical Report CMU–CS–90–119, Carnegie Mellon University, Pittsburgh, PA, May 1990.

David L. Detlefs, Paul A. Martin, Mark Moir, and Guy L. Steele. Lock-free reference counting. In *20th ACM Symposium on Distributed Computing*, Newport, Rhode Island, August 2001, pages 190–199. ACM Press. doi: 10.1145/383962.384016.

David L. Detlefs, Paul A. Martin, Mark Moir, and Guy L. Steele. Lock-free reference counting. *Distributed Computing*, 15:255–271, 2002b. doi: 10.1007/s00446-002-0079-z.

John DeTreville. Experience with concurrent garbage collectors for Modula-2+. Technical Report 64, DEC Systems Research Center, Palo Alto, CA, August 1990.

L. Peter Deutsch and Daniel G. Bobrow. An efficient incremental automatic garbage collector. *Communications of the ACM*, 19(9):522–526, September 1976. doi: 10.1145/360336.360345.

Sylvia Dieckmann and Urs Hölzle. The allocation behaviour of the SPECjvm98 Java benchmarks. In Rudolf Eigenman, editor, *Performance Evaluation and Benchmarking with Realistic Applications*, chapter 3, pages 77–108. MIT Press, 2001.

Sylvia Dieckmann and Urs Hölzle. A study of the allocation behaviour of the SPECjvm98 Java benchmarks. In Rachid Guerraoui, editor, *13th European Conference on Object-Oriented Programming*, Lisbon, Portugal, July 1999, pages 92–115. Volume 1628 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-48743-3_5.

Edsgar W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On-the-fly garbage collection: An exercise in cooperation. In *Language Hierarchies and Interfaces: International Summer School*, volume 46 of *Lecture Notes in Computer Science*, pages 43–56. Springer-Verlag, Marktoberdorf, Germany, 1976. doi: 10.1007/3-540-07994-7_48.

Edsgar W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On-the-fly garbage collection: An exercise in cooperation. *Communications of the ACM*, 21(11):965–975, November 1978. doi: 10.1145/359642.359655.

Robert Dimpsey, Rajiv Arora, and Kean Kuiper. Java server performance: A case study of building efficient, scalable JVMs. *IBM Systems Journal*, 39(1):151–174, 2000. doi: 10.1147/sj.391.0151.

Amer Diwan, J. Eliot B. Moss, and Richard L. Hudson. Compiler support for garbage collection in a statically typed language. In PLDI 1992, pages 273–282. doi: 10.1145/143095.143140.

Amer Diwan, David Tarditi, and J. Eliot B. Moss. Memory subsystem performance of programs using copying garbage collection. In POPL 1994, pages 1–14. doi: 10.1145/174675.174710.

Julian Dolby. Automatic inline allocation of objects. In PLDI 1997, pages 7–17. doi: 10.1145/258915.258918.

Julian Dolby and Andrew A. Chien. An automatic object inlining optimization and its evaluation. In PLDI 2000, pages 345–357. doi: 10.1145/349299.349344.

Julian Dolby and Andrew A. Chien. An evaluation of automatic object inline allocation techniques. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, Canada, October 1998, pages 1–20. ACM SIGPLAN Notices 33(10), ACM Press. doi: `10.1145/286936.286943`.

Damien Doligez and Georges Gonthier. Portable, unobtrusive garbage collection for multiprocessor systems. In POPL 1994, pages 70–83. doi: `10.1145/174675.174673`.

Damien Doligez and Xavier Leroy. A concurrent generational garbage collector for a multi-threaded implementation of ML. In *20th Annual ACM Symposium on Principles of Programming Languages*, Charleston, SC, January 1993, pages 113–123. ACM Press. doi: `10.1145/158511.158611`.

Tamar Domani, Elliot K. Kolodner, and Erez Petrank. A generational on-the-fly garbage collector for Java. In PLDI 2000, pages 274–284. doi: `10.1145/349299.349336`.

Tamar Domani, Elliot K. Kolodner, Ethan Lewis, Erez Petrank, and Dafna Sheinwald. Thread-local heaps for Java. In ISMM 2002, pages 76–87. doi: `10.1145/512429.512439`.

Kevin Donnelly, Joe Hallett, and Assaf Kfoury. Formal semantics of weak references. In ISMM 2006, pages 126–137. doi: `10.1145/1133956.1133974`.

R. Kent Dybvig, Carl Bruggeman, and David Eby. Guardians in a generation-based garbage collector. In PLDI 1993, pages 207–216. doi: `10.1145/155090.155110`.

ECOOP 2007, Erik Ernst, editor. *21st European Conference on Object-Oriented Programming*, Berlin, Germany, July 2007. Volume 4609 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: `10.1007/978-3-540-73589-2`.

Daniel R. Edelson. Smart pointers: They're smart, but they're not pointers. In *USENIX C++ Conference*, Portland, OR, August 1992. USENIX.

Toshio Endo, Kenjiro Taura, and Akinori Yonezawa. A scalable mark-sweep garbage collector on large-scale shared-memory machines. In *ACM/IEEE Conference on Supercomputing*, San Jose, CA, November 1997. doi: `10.1109/SC.1997.10059`.

A. P. Ershov. On programming of arithmetic operations. *Communications of the ACM*, 1(8): 3–6, August 1958. doi: `10.1145/368892.368907`.

Shahrooz Feizabadi and Godmar Back. Java garbage collection scheduling in utility accrual scheduling environments. In *3rd International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, San Diego, CA, 2005.

Shahrooz Feizabadi and Godmar Back. Garbage collection-aware sheduling utility accrual scheduling environments. *Real-Time Systems*, 36(1–2), July 2007. doi: `10.1007/s11241-007-9020-7`.

Robert R. Fenichel and Jerome C. Yochelson. A Lisp garbage collector for virtual memory computer systems. *Communications of the ACM*, 12(11):611–612, November 1969. doi: `10.1145/363269.363280`.

Stephen J. Fink and Feng Qian. Design, implementation and evaluation of adaptive recompilation with on-stack replacement. In *1st International Symposium on Code Generation and Optimization (CGO)*, San Francisco, CA, March 2003, pages 241–252. IEEE Computer Society Press. doi: `10.1109/CGO.2003.1191549`.

David A. Fisher. Bounded workspace garbage collection in an address order preserving list processing environment. *Information Processing Letters*, 3(1):29–32, July 1974. doi: 10.1016/0020-0190(74)90044-1.

Robert Fitzgerald and David Tarditi. The case for profile-directed selection of garbage collectors. In ISMM 2000, pages 111–120. doi: 10.1145/362422.362472.

Christine Flood, Dave Detlefs, Nir Shavit, and Catherine Zhang. Parallel garbage collection for shared memory multiprocessors. In JVM 2001.

John K. Foderaro and Richard J. Fateman. Characterization of VAX Macsyma. In *1981 ACM Symposium on Symbolic and Algebraic Computation*, Berkeley, CA, 1981, pages 14–19. ACM Press. doi: 10.1145/800206.806364.

John K. Foderaro, Keith Sklower, Kevin Layer, *et al*. *Franz Lisp Reference Manual*. Franz Inc., 1985.

Daniel Frampton, David F. Bacon, Perry Cheng, and David Grove. Generational real-time garbage collection: A three-part invention for young objects. In ECOOP 2007, pages 101–125. doi: 10.1007/978-3-540-73589-2_6.

Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, September 1960. doi: 10.1145/367390.367400.

Daniel P. Friedman and David S. Wise. Reference counting can manage the circular environments of mutual recursion. *Information Processing Letters*, 8(1):41–45, January 1979. doi: 10.1016/0020-0190(79)90091-7.

Robin Garner, Stephen M. Blackburn, and Daniel Frampton. Effective prefetch for mark-sweep garbage collection. In ISMM 2007, pages 43–54. doi: 10.1145/1296907.1296915.

Alex Garthwaite. *Making the Trains Run On Time*. PhD thesis, University of Pennsylvania, 2005.

Alex Garthwaite, Dave Dice, and Derek White. Supporting per-processor local-allocation buffers using lightweight user-level preemption notification. In VEE 2005, pages 24–34. doi: 10.1145/1064979.1064985.

Alexander T. Garthwaite, David L. Detlefs, Antonios Printezis, and Y. Srinivas Ramakrishna. Method and mechanism for finding references in a card in time linear in the size of the card in a garbage-collected heap. United States Patent 7,136,887 B2, Sun Microsystems, November 2006.

David Gay and Bjarne Steensgaard. Fast escape analysis and stack allocation for object-based programs. In *9th International Conference on Compiler Construction*, Berlin, April 2000, pages 82–93. Volume 2027 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-46423-9_6.

GC 1990, Eric Jul and Niels-Christian Juul, editors. *OOPSLA/ECOOP Workshop on Garbage Collection in Object-Oriented Systems*, Ottawa, Canada, October 1990.

GC 1991, Paul R. Wilson and Barry Hayes, editors. *OOPSLA Workshop on Garbage Collection in Object-Oriented Systems*, October 1991.

GC 1993, Eliot Moss, Paul R. Wilson, and Benjamin Zorn, editors. *OOPSLA Workshop on Garbage Collection in Object-Oriented Systems*, October 1993.

Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous Java performance evaluation. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Montréal, Canada, October 2007, pages 57–76. ACM SIGPLAN Notices 42(10), ACM Press. doi: 10.1145/1297027.1297033.

Joseph (Yossi) Gil and Itay Maman. Micro patterns in Java code. In OOPSLA 2005, pages 97–116. doi: 10.1145/1094811.1094819.

O. Goh, Yann-Hang Lee, Z. Kaakani, and E. Rachlin. Integrated scheduling with garbage collection for real-time embedded applications in CLI. In *9th International Symposium on Object-Oriented Real-Time Distributed Computing*, Gyeongju, Korea, April 2006. IEEE Press. doi: 10.1109/ISORC.2006.41.

Benjamin Goldberg. Tag-free garbage collection for strongly typed programming languages. In PLDI 1991 [PLDI 1991], pages 165–176. doi: 10.1145/113445.113460.

Benjamin Goldberg. Incremental garbage collection without tags. In *European Symposium on Programming*, Rennes, France, February 1992, pages 200–218. Volume 582 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-55253-7_12.

Benjamin Goldberg and Michael Gloger. Polymorphic type reconstruction for garbage collection without tags. In LFP 1992, pages 53–65. doi: 10.1145/141471.141504.

Marcelo J. R. Gonçalves and Andrew W. Appel. Cache performance of fast-allocating programs. In *Conference on Functional Programming and Computer Architecture*, La Jolla, CA, June 1995, pages 293–305. ACM Press. doi: 10.1145/224164.224219.

James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, third edition edition, May 2005.

Eiichi Goto. Monocopy and associative algorithms in an extended LISP. Technical Report 74-03, Information Science Laboratories, Faculty of Science, University of Tokyo, 1974.

David Gries. An exercise in proving parallel programs correct. *Communications of the ACM*, 20(12):921–930, December 1977. doi: 10.1145/359897.359903.

Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, and James Cheney. Region-based memory management in Cyclone. In PLDI 2002, pages 282–293. doi: 10.1145/512529.512563.

Chris Grzegorczyk, Sunil Soman, Chandra Krintz, and Rich Wolski. Isla Vista heap sizing: Using feedback to avoid paging. In *5th International Symposium on Code Generation and Optimization (CGO)*, San Jose, CA, March 2007, pages 325–340. IEEE Computer Society Press. doi: 10.1109/CGO.2007.20.

Samuel Guyer and Kathryn McKinley. Finding your cronies: Static analysis for dynamic object colocation. In OOPSLA 2004, pages 237–250. doi: 10.1145/1028976.1028996.

Robert H. Halstead. Implementation of Multilisp: Lisp on a multiprocessor. In LFP 1984, pages 9–17. doi: 10.1145/800055.802017.

Robert H. Halstead. Multilisp: A language for concurrent symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–538, October 1985. doi: 10.1145/4472.4478.

Lars Thomas Hansen. *Older-first Garbage Collection in Practice*. PhD thesis, Northeastern University, November 2000.

Lars Thomas Hansen and William D. Clinger. An experimental study of renewal-older-first garbage collection. In *7th ACM SIGPLAN International Conference on Functional Programming*, Pittsburgh, PA, September 2002, pages 247–258. ACM SIGPLAN Notices 37(9), ACM Press. doi: `10.1145/581478.581502`.

David R. Hanson. Storage management for an implementation of SNOBOL4. *Software: Practice and Experience*, 7(2):179–192, 1977. doi: `10.1002/spe.4380070206`.

Tim Harris and Keir Fraser. Language support for lightweight transactions. In OOPSLA 2003, pages 388–402. doi: `10.1145/949305.949340`.

Timothy Harris. Dynamic adaptive pre-tenuring. In ISMM 2000, pages 127–136. doi: `10.1145/362422.362476`.

Timothy L. Harris, Keir Fraser, and Ian A. Pratt. A practical multi-word compare-and-swap operation. In Dahlia Malkhi, editor, *International Conference on Distributed Computing*, Toulouse, France, October 2002, pages 265–279. Volume 2508 of *Lecture Notes in Computer Science*. doi: `10.1007/3-540-36108-1_18`.

Pieter H. Hartel. *Performance Analysis of Storage Management in Combinator Graph Reduction*. PhD thesis, Department of Computer Systems, University of Amsterdam, Amsterdam, 1988.

Barry Hayes. Using key object opportunism to collect old objects. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Phoenix, AZ, November 1991, pages 33–46. ACM SIGPLAN Notices 26(11), ACM Press. doi: `10.1145/117954.117957`.

Barry Hayes. Finalization in the collector interface. In IWMM 1992, pages 277–298. doi: `10.1007/BFb0017196`.

Barry Hayes. Ephemerons: A new finalization mechanism. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Atlanta, GA, October 1997, pages 176–183. ACM SIGPLAN Notices 32(10), ACM Press. doi: `10.1145/263698.263733`.

Laurence Hellyer, Richard Jones, and Antony L. Hosking. The locality of concurrent write barriers. In ISMM 2010, pages 83–92. doi: `10.1145/1806651.1806666`.

Fergus Henderson. Accurate garbage collection in an uncooperative environment. In ISMM 2002, pages 150–156. doi: `10.1145/512429.512449`.

Roger Henriksson. *Scheduling Garbage Collection in Embedded Systems*. PhD thesis, Lund Institute of Technology, July 1998.

Maurice Herlihy and J. Eliot B Moss. Lock-free garbage collection for multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 3(3):304–311, May 1992. doi: `10.1109/71.139204`.

Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufman, April 2008.

Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3): 463–492, 1990. doi: 10.1145/78969.78972.

Maurice P. Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *20th Annual International Symposium on Computer Architecture*, San Diego, CA, May 1993, pages 289–300. IEEE Press. doi: 10.1145/165123.165164.

Maurice P. Herlihy, Victor Luchangco, and Mark Moir. The repeat offender problem: A mechanism for supporting dynamic-sized lock-free data structures. In *16th International Symposium on Distributed Computing*, Toulouse, France, October 2002, pages 339–353. Volume 2508 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-36108-1_23.

Matthew Hertz. *Quantifying and Improving the Performance of Garbage Collection*. PhD thesis, University of Massachusetts, September 2006.

Matthew Hertz and Emery Berger. Quantifying the performance of garbage collection vs. explicit memory management. In OOPSLA 2005, pages 313–326. doi: 10.1145/1094811.1094836.

Matthew Hertz, Yi Feng, and Emery D. Berger. Garbage collection without paging. In Vivek Sarkar and Mary W. Hall, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Chicago, IL, June 2005, pages 143–153. ACM SIGPLAN Notices 40(6), ACM Press. doi: 10.1145/1064978.1065028.

Matthew Hertz, Jonathan Bard, Stephen Kane, Elizabeth Keudel, Tongxin Bai, Kirk Kelsey, and Chen Ding. Waste not, want not — resource-based garbage collection in a shared environment. Technical Report TR–951, The University of Rochester, December 2009. doi: 1802/8838.

D. S. Hirschberg. A class of dynamic memory allocation algorithms. *Communications of the ACM*, 16(10):615–618, October 1973. doi: 10.1145/362375.362392.

Martin Hirzel, Amer Diwan, and Matthew Hertz. Connectivity-based garbage collection. In OOPSLA 2003, pages 359–373. doi: 10.1145/949305.949337.

Urs Hölzle. A fast write barrier for generational garbage collectors. In GC 1993.

Antony L Hosking. Portable, mostly-concurrent, mostly-copying garbage collection for multi-processors. In ISMM 2006, pages 40–51. doi: 10.1145/1133956.1133963.

Antony L. Hosking and Richard L. Hudson. Remembered sets can also play cards. In GC 1993.

Antony L. Hosking and J. Eliot B. Moss. Protection traps and alternatives for memory management of an object-oriented language. In *14th ACM Symposium on Operating Systems Principles*, Asheville, NC, December 1993, pages 106–119. ACM SIGOPS Operating Systems Review 27(5), ACM Press. doi: 10.1145/168619.168628.

Antony L. Hosking, J. Eliot B. Moss, and Darko Stefanović. A comparative performance evaluation of write barrier implementations. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, Canada, October 1992, pages 92–109. ACM SIGPLAN Notices 27(10), ACM Press. doi: 10.1145/141936.141946.

Antony L. Hosking, Nathaniel Nystrom, Quintin Cutts, and Kumar Brahnmath. Optimizing the read and write barrier for orthogonal persistence. In Ronald Morrison, Mick J. Jordan, and Malcolm P. Atkinson, editors, *8th International Workshop on Persistent Object Systems (August, 1998)*, Tiburon, CA, 1999, pages 149–159. Advances in Persistent Object Systems, Morgan Kaufmann.

Xianlong Huang, Stephen M. Blackburn, Kathryn S. McKinley, J. Eliot B. Moss, Z. Wang, and Perry Cheng. The garbage collection advantage: Improving program locality. In OOPSLA 2004, pages 69–80. doi: `10.1145/1028976.1028983`.

Richard L. Hudson. Finalization in a garbage collected world. In GC 1991.

Richard L. Hudson and Amer Diwan. Adaptive garbage collection for Modula-3 and Smalltalk. In GC 1990.

Richard L. Hudson and J. Eliot B. Moss. Sapphire: Copying GC without stopping the world. In *Joint ACM-ISCOPE Conference on Java Grande*, Palo Alto, CA, June 2001, pages 48–57. ACM Press. doi: `10.1145/376656.376810`.

Richard L. Hudson and J. Eliot B. Moss. Sapphire: Copying garbage collection without stopping the world. *Concurrency and Computation: Practice and Experience*, 15(3–5): 223–261, 2003. doi: `10.1002/cpe.712`.

Richard L. Hudson and J. Eliot B. Moss. Incremental collection of mature objects. In IWMM 1992, pages 388–403. doi: `10.1007/BFb0017203`.

Richard L. Hudson, J. Eliot B. Moss, Amer Diwan, and Christopher F. Weight. A language-independent garbage collector toolkit. Technical Report COINS 91-47, University of Massachusetts, September 1991.

R. John M. Hughes. A semi-incremental garbage collection algorithm. *Software: Practice and Experience*, 12(11):1081–1082, November 1982. doi: `10.1002/spe.4380121108`.

Akira Imai and Evan Tick. Evaluation of parallel copying garbage collection on a shared-memory multiprocessor. *Transactions on Parallel and Distributed Systems*, 4(9): 1030–1040, 1993. doi: `10.1109/71.243529`.

ISMM 1998, Simon L. Peyton Jones and Richard Jones, editors. *1st International Symposium on Memory Management*, Vancouver, Canada, October 1998. ACM SIGPLAN Notices 34(3), ACM Press. doi: `10.1145/286860`.

ISMM 2000, Craig Chambers and Antony L. Hosking, editors. *2nd International Symposium on Memory Management*, Minneapolis, MN, October 2000. ACM SIGPLAN Notices 36(1), ACM Press. doi: `10.1145/362422`.

ISMM 2002, Hans-J. Boehm and David Detlefs, editors. *3rd International Symposium on Memory Management*, Berlin, Germany, June 2002. ACM SIGPLAN Notices 38(2 supplement), ACM Press. doi: `10.1145/773146`.

ISMM 2004, David F. Bacon and Amer Diwan, editors. *4th International Symposium on Memory Management*, Vancouver, Canada, October 2004. ACM Press. doi: `10.1145/1029873`.

ISMM 2006, Erez Petrank and J. Eliot B. Moss, editors. *5th International Symposium on Memory Management*, Ottawa, Canada, June 2006. ACM Press. doi: `10.1145/1133956`.

ISMM 2007, Greg Morrisett and Mooly Sagiv, editors. *6th International Symposium on Memory Management*, Montréal, Canada, October 2007. ACM Press. doi: 10.1145/1296907.

ISMM 2008, Richard Jones and Steve Blackburn, editors. *7th International Symposium on Memory Management*, Tucson, AZ, June 2008. ACM Press. doi: 10.1145/1375634.

ISMM 2009, Hillel Kolodner and Guy Steele, editors. *8th International Symposium on Memory Management*, Dublin, Ireland, June 2009. ACM Press. doi: 10.1145/1542431.

ISMM 2010, Jan Vitek and Doug Lea, editors. *9th International Symposium on Memory Management*, Toronto, Canada, June 2010. ACM Press. doi: 10.1145/1806651.

ISMM 2011, Hans Boehm and David Bacon, editors. *10th International Symposium on Memory Management*, San Jose, CA, June 2011. ACM Press. doi: 10.1145/1993478.

IWMM 1992, Yves Bekkers and Jacques Cohen, editors. *International Workshop on Memory Management*, St Malo, France, 17–19 September 1992. Volume 637 of *Lecture Notes in Computer Science*, Springer. doi: 10.1007/BFb0017181.

IWMM 1995, Henry G. Baker, editor. *International Workshop on Memory Management*, Kinross, Scotland, 27–29September 1995. Volume 986 of *Lecture Notes in Computer Science*, Springer. doi: 10.1007/3-540-60368-9.

Erik Johansson, Konstantinos Sagonas, and Jesper Wilhelmsson. Heap architectures for concurrent lanugages using message passing. In ISMM 2002, pages 88–99. doi: 10.1145/512429.512440.

Mark S. Johnstone. *Non-Compacting Memory Allocation and Real-Time Garbage Collection*. PhD thesis, University of Texas at Austin, December 1997.

Richard Jones and Chris Ryder. A study of Java object demographics. In ISMM 2008, pages 121–130. doi: 10.1145/1375634.1375652.

Richard E. Jones. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. Wiley, Chichester, July 1996. With a chapter on Distributed Garbage Collection by R. Lins.

Richard E. Jones and Andy C. King. Collecting the garbage without blocking the traffic. Technical Report 18–04, Computing Laboratory, University of Kent, September 2004. This report summarises King [2004].

Richard E. Jones and Andy C. King. A fast analysis for thread-local garbage collection with dynamic class loading. In *5th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, Budapest, September 2005, pages 129–138. IEEE Computer Society Press. doi: 10.1109/SCAM.2005.1. This is a shorter version of Jones and King [2004].

H. B. M. Jonkers. A fast garbage compaction algorithm. *Information Processing Letters*, 9(1): 26–30, July 1979. doi: 10.1016/0020-0190(79)90103-0.

Maria Jump, Stephen M. Blackburn, and Kathryn S. McKinley. Dynamic object sampling for pretenuring. In ISMM 2004, pages 152–162. doi: 10.1145/1029873.1029892.

JVM 2001. *1st Java Virtual Machine Research and Technology Symposium*, Monterey, CA, April 2001. USENIX.

Tomas Kalibera. Replicating real-time garbage collector for Java. In *7th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, Madrid, Spain, September 2009, pages 100–109. ACM Press. doi: 10.1145/1620405.1620420.

Tomas Kalibera, Filip Pizlo, Antony L. Hosking, and Jan Vitek. Scheduling hard real-time garbage collection. In *30th IEEE Real-Time Systems Symposium*, Washington, DC, December 2009, pages 81–92. IEEE Computer Society Press. doi: 10.1109/RTSS.2009.40.

Haim Kermany and Erez Petrank. The Compressor: Concurrent, incremental and parallel compaction. In PLDI 2006, pages 354–363. doi: 10.1145/1133981.1134023.

Taehyoun Kim and Heonshik Shin. Scheduling-aware real-time garbage collection using dual aperiodic servers. In *Real-Time and Embedded Computing Systems and Applications*, 2004, pages 1–17. Volume 2968 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/978-3-540-24686-2_1.

Taehyoun Kim, Naehyuck Chang, Namyun Kim, and Heonshik Shin. Scheduling garbage collector for embedded real-time systems. In *ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Atlanta, GA, May 1999, pages 55–64. ACM SIGPLAN Notices 34(7), ACM Press. doi: 10.1145/314403.314444.

Taehyoun Kim, Naehyuck Chang, and Heonshik Shin. Bounding worst case garbage collection time for embedded real-time systems. In *6th IEEE Real-Time Technology and Applications Symposium (RTAS)*, Washington, DC, May/June 2000, pages 46–55. doi: 10.1109/RTTAS.2000.852450.

Taehyoun Kim, Naehyuck Chang, and Heonshik Shin. Joint scheduling of garbage collector and hard real-time tasks for embedded applications. *Journal of Systems and Software*, 58(3):247–260, September 2001. doi: 10.1016/S0164-1212(01)00042-5.

Andy C. King. *Removing Garbage Collector Synchronisation*. PhD thesis, Computing Laboratory, The University of Kent at Canterbury, 2004.

Kenneth C. Knowlton. A fast storage allocator. *Communications of the ACM*, 8(10):623–625, October 1965. doi: 10.1145/365628.365655.

Donald E. Knuth. *The Art of Computer Programming*, volume I: Fundamental Algorithms. Addison-Wesley, second edition, 1973.

Elliot K. Kolodner and Erez Petrank. Parallel copying garbage collection using delayed allocation. Technical Report 88.384, IBM Haifa Research Lab., November 1999.

David G. Korn and Kiem-Phong Vo. In search of a better malloc. In *USENIX Summer Conference*, Portland, OR, 1985, pages 489–506. USENIX Association.

H. T. Kung and S. W. Song. An efficient parallel garbage collection system and its correctness proof. In *IEEE Symposium on Foundations of Computer Science*, 1977, pages 120–131. IEEE Press. doi: 10.1109/SFCS.1977.5.

Michael S. Lam, Paul R. Wilson, and Thomas G. Moher. Object type directed garbage collection to improve locality. In IWMM 1992, pages 404–425. doi: 10.1007/BFb0017204.

Leslie Lamport. Garbage collection with multiple processes: an exercise in parallelism. In *International Conference on Parallel Processing (ICPP)*, 1976, pages 50–54.

Bernard Lang and Francis Dupont. Incremental incrementally compacting garbage collection. In *Symposium on Interpreters and Interpretive Techniques*, St Paul, MN, June 1987, pages 253–263. ACM SIGPLAN Notices 22(7), ACM Press. doi: 10.1145/29650.29677.

LCTES 2003. *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, San Diego, CA, June 2003. ACM SIGPLAN Notices 38(7), ACM Press. doi: 10.1145/780732.

Ho-Fung Leung and Hing-Fung Ting. An optimal algorithm for global termination detection in shared-memory asynchronous multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):538–543, May 1997. doi: 10.1109/71.598280.

Yossi Levanoni and Erez Petrank. An on-the-fly reference counting garbage collector for Java. In OOPSLA 2001, pages 367–380. doi: 10.1145/504282.504309.

Yossi Levanoni and Erez Petrank. An on-the-fly reference counting garbage collector for Java. *ACM Transactions on Programming Languages and Systems*, 28(1):1–69, January 2006. doi: 10.1145/1111596.1111597.

Yossi Levanoni and Erez Petrank. A scalable reference counting garbage collector. Technical Report CS–0967, Technion — Israel Institute of Technology, Haifa, Israel, November 1999.

LFP 1984, Guy L. Steele, editor. *ACM Conference on LISP and Functional Programming*, Austin, TX, August 1984. ACM Press. doi: 10.1145/800055.

LFP 1992. *ACM Conference on LISP and Functional Programming*, San Francisco, CA, June 1992. ACM Press. doi: 10.1145/141471.

Henry Lieberman and Carl E. Hewitt. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, 26(6):419–429, 1983. doi: 10.1145/358141.358147. Also report TM–184, Laboratory for Computer Science, MIT, Cambridge, MA, July 1980 and AI Lab Memo 569, 1981.

Rafael D. Lins. Cyclic reference counting with lazy mark-scan. *Information Processing Letters*, 44(4):215–220, 1992. doi: 10.1016/0020-0190(92)90088-D. Also Computing Laboratory Technical Report 75, University of Kent, July 1990.

Boris Magnusson and Roger Henriksson. Garbage collection for control systems. In IWMM 1995, pages 323–342. doi: 10.1007/3-540-60368-9_32.

Jeremy Manson, William Pugh, and Sarita V. Adve. The Java memory model. In *32nd Annual ACM Symposium on Principles of Programming Languages*, Long Beach, CA, January 2005, pages 378–391. ACM SIGPLAN Notices 40(1), ACM Press. doi: 10.1145/1040305.1040336.

Sebastien Marion, Richard Jones, and Chris Ryder. Decrypting the Java gene pool: Predicting objects' lifetimes with micro-patterns. In ISMM 2007, pages 67–78. doi: 10.1145/1296907.1296918.

Simon Marlow, Tim Harris, Roshan James, and Simon Peyton Jones. Parallel generational-copying garbage collection with a block-structured heap. In ISMM 2008, pages 11–20. doi: 10.1145/1375634.1375637.

Johannes J. Martin. An efficient garbage compaction algorithm. *Communications of the ACM*, 25(8):571–581, August 1982. doi: 10.1145/358589.358625.

A. D. Martinez, R. Wachenchauzer, and Rafael D. Lins. Cyclic reference counting with local mark-scan. *Information Processing Letters*, 34:31–35, 1990. doi: 10.1016/0020-0190(90)90226-N.

John McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, April 1960. doi: 10.1145/367177.367199.

John McCarthy. History of LISP. In Richard L. Wexelblat, editor, *History of Programming Languages I*, pages 173–185. ACM Press, 1978. doi: 10.1145/800025.1198360.

Bill McCloskey, David F. Bacon, Perry Cheng, and David Grove. Staccato: A parallel and concurrent real-time compacting garbage collector for multiprocessors. IBM Research Report RC24505, IBM Research, 2008.

Phil McGachey and Antony L Hosking. Reducing generational copy reserve overhead with fallback compaction. In ISMM 2006, pages 17–28. doi: 10.1145/1133956.1133960.

Phil McGachey, Ali-Reza Adl-Tabatabi, Richard L. Hudson, Vijay Menon, Bratin Saha, and Tatiana Shpeisman. Concurrent GC leveraging transactional memory. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Salt Lake City, UT, February 2008, pages 217–226. ACM Press. doi: 10.1145/1345206.1345238.

Paul E. McKenney and Jack Slingwine. Read-copy update: Using execution history to solve concurrency problems. In *10th IASTED International Conference on Parallel and Distributed Computing and Systems*, October 1998.

Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):491–504, June 2004. doi: 10.1109/TPDS.2004.8.

Maged M. Michael and M.L. Scott. Correction of a memory management method for lock-free data structures. Technical Report UR CSD / TR59, University of Rochester, December 1995. doi: 1802/503.

James S. Miller and Guillermo J. Rozas. Garbage collection is fast, but a stack is faster. Technical Report AIM-1462, MIT AI Laboratory, March 1994. doi: 1721.1/6622.

David A. Moon. Garbage collection in a large LISP system. In LFP 1984, pages 235–245. doi: 10.1145/800055.802040.

F. Lockwood Morris. A time- and space-efficient garbage compaction algorithm. *Communications of the ACM*, 21(8):662–5, 1978. doi: 10.1145/359576.359583.

F. Lockwood Morris. On a comparison of garbage collection techniques. *Communications of the ACM*, 22(10):571, October 1979.

F. Lockwood Morris. Another compacting garbage collector. *Information Processing Letters*, 15(4):139–142, October 1982. doi: 10.1016/0020-0190(82)90094-1.

J. Eliot B. Moss. Working with persistent objects: To swizzle or not to swizzle? *IEEE Transactions on Software Engineering*, 18(8):657–673, August 1992. doi: 10.1109/32.153378.

Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! In Mary Lou Soffa, editor, *14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Seattle, WA, March 2008, pages 265–276. ACM SIGPLAN Notices 43(3), ACM Press. doi: 10.1145/1508244.1508275.

John Nagle. Re: Real-time GC (was Re: Widespread C++ competency gap). USENET comp.lang.c++, January 1995.

Scott Nettles and James O'Toole. Real-time replication-based garbage collection. In PLDI 1993, pages 217–226. doi: 10.1145/155090.155111.

Scott M. Nettles, James W. O'Toole, David Pierce, and Nicholas Haines. Replication-based incremental copying collection. In IWMM 1992, pages 357–364. doi: 10.1007/BFb0017201.

Yang Ni, Vijay Menon, Ali-Reza Adl-Tabatabai, Antony L. Hosking, Richard L. Hudson, J. Eliot B. Moss, Bratin Saha, and Tatiana Shpeisman. Open nesting in software transactional memory. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Jose, CA, March 2007, pages 68–78. ACM Press. doi: 10.1145/1229428.1229442.

Gene Novark, Trevor Strohman, and Emery D. Berger. Custom object layout for garbage-collected languages. Technical report, University of Massachusetts, 2006. New England Programming Languages and Systems Symposium, March, 2006.

Cosmin E. Oancea, Alan Mycroft, and Stephen M. Watt. A new approach to parallelising tracing algorithms. In ISMM 2009, pages 10–19. doi: 10.1145/1542431.1542434.

Takeshi Ogasawara. NUMA-aware memory manager with dominant-thread-based copying GC. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Orlando, FL, October 2009, pages 377–390. ACM SIGPLAN Notices 44(10), ACM Press. doi: 10.1145/1640089.1640117.

OOPSLA 1999. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Denver, CO, October 1999. ACM SIGPLAN Notices 34(10), ACM Press. doi: 10.1145/320384.

OOPSLA 2001. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Tampa, FL, November 2001. ACM SIGPLAN Notices 36(11), ACM Press. doi: 10.1145/504282.

OOPSLA 2002. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Seattle, WA, November 2002. ACM SIGPLAN Notices 37(11), ACM Press. doi: 10.1145/582419.

OOPSLA 2003. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Anaheim, CA, November 2003. ACM SIGPLAN Notices 38(11), ACM Press. doi: 10.1145/949305.

OOPSLA 2004. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, Canada, October 2004. ACM SIGPLAN Notices 39(10), ACM Press. doi: 10.1145/1028976.

OOPSLA 2005. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, San Diego, CA, October 2005. ACM SIGPLAN Notices 40(10), ACM Press. doi: 10.1145/1094811.

Yoav Ossia, Ori Ben-Yitzhak, Irit Goft, Elliot K. Kolodner, Victor Leikehman, and Avi Owshanko. A parallel, incremental and concurrent GC for servers. In PLDI 2002, pages 129–140. doi: 10.1145/512529.512546.

Yoav Ossia, Ori Ben-Yitzhak, and Marc Segal. Mostly concurrent compaction for mark-sweep GC. In ISMM 2004, pages 25–36. doi: 10.1145/1029873.1029877.

Ivor P. Page and Jeff Hagins. Improving the performance of buddy systems. *IEEE Transactions on Computers*, C-35(5):441–447, May 1986. doi: 10.1109/TC.1986.1676786.

Krzysztof Palacz, Jan Vitek, Grzegorz Czajkowski, and Laurent Daynès. Incommunicado: efficient communication for isolates. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, October 1994, pages 262–274. ACM SIGPLAN Notices 29(10), ACM Press. doi: 10.1145/582419.582444.

Stephen K. Park and Keith W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, October 1988. doi: 10.1145/63039.6304.

Harel Paz and Erez Petrank. Using prefetching to improve reference-counting garbage collectors. In *16th International Conference on Compiler Construction*, Braga, Portugal, March 2007, pages 48–63. Volume 4420 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/978-3-540-71229-9_4.

Harel Paz, David F. Bacon, Elliot K. Kolodner, Erez Petrank, and V.T. Rajan. Efficient on-the-fly cycle collection. Technical Report CS–2003–10, Technion University, 2003.

Harel Paz, Erez Petrank, David F. Bacon, Elliot K. Kolodner, and V.T. Rajan. An efficient on-the-fly cycle collection. In CC 2005, pages 156–171. doi: 10.1007/978-3-540-31985-6_11.

Harel Paz, Erez Petrank, and Stephen M. Blackburn. Age-oriented concurrent garbage collection. In CC 2005, pages 121–136. doi: 10.1007/978-3-540-31985-6_9.

Harel Paz, David F. Bacon, Elliot K. Kolodner, Erez Petrank, and V. T. Rajan. An efficient on-the-fly cycle collection. *ACM Transactions on Programming Languages and Systems*, 29 (4):1–43, August 2007. doi: 10.1145/1255450.1255453.

E. J. H. Pepels, M. C. J. D. van Eekelen, and M. J. Plasmeijer. A cyclic reference counting algorithm and its proof. Technical Report 88–10, Computing Science Department, University of Nijmegen, 1988.

James L. Peterson and Theodore A. Norman. Buddy systems. *Communications of the ACM*, 20(6):421–431, 1977. doi: 10.1145/359605.359626.

Erez Petrank and Elliot K. Kolodner. Parallel copying garbage collection using delayed allocation. *Parallel Processing Letters*, 14(2):271–286, June 2004. doi: 10.1142/S0129626404001878.

Erez Petrank and Dror Rawitz. The hardness of cache conscious data placement. In *Twenty-ninth Annual ACM Symposium on Principles of Programming Languages*, Portland, OR, January 2002, pages 101–112. ACM SIGPLAN Notices 37(1), ACM Press. doi: 10.1145/503272.503283.

Pekka P. Pirinen. Barrier techniques for incremental tracing. In ISMM 1998, pages 20–25. doi: 10.1145/286860.286863.

Filip Pizlo and Jan Vitek. Memory management for real-time Java: State of the art. In *11th International Symposium on Object-Oriented Real-Time Distributed Computing*, Orlando, FL, 2008, pages 248–254. IEEE Press. doi: 10.1109/ISORC.2008.40.

Filip Pizlo, Daniel Frampton, Erez Petrank, and Bjarne Steensgard. Stopless: A real-time garbage collector for multiprocessors. In ISMM 2007, pages 159–172. doi: 10.1145/1296907.1296927.

Filip Pizlo, Erez Petrank, and Bjarne Steensgaard. A study of concurrent real-time garbage collectors. In PLDI 2008, pages 33–44. doi: 10.1145/1379022.1375587.

Filip Pizlo, Lukasz Ziarek, Ethan Blanton, Petr Maj, and Jan Vitek. High-level programming of embedded hard real-time devices. In *5th European Conference on Computer Systems (EuroSys)*, Paris, France, April 2010a, pages 69–82. ACM Press. doi: 10.1145/1755913.1755922.

Filip Pizlo, Lukasz Ziarek, Petr Maj, Antony L. Hosking, Ethan Blanton, and Jan Vitek. Schism: Fragmentation-tolerant real-time garbage collection. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Toronto, Canada, June 2010b, pages 146–159. ACM SIGPLAN Notices 45(6), ACM Press. doi: 10.1145/1806596.1806615.

PLDI 1988. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, June 1988. ACM SIGPLAN Notices 23(7), ACM Press. doi: 10.1145/53990.

PLDI 1991. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Toronto, Canada, June 1991. ACM SIGPLAN Notices 26(6), ACM Press. doi: 10.1145/113445.

PLDI 1992. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Francisco, CA, June 1992. ACM SIGPLAN Notices 27(7), ACM Press. doi: 10.1145/143095.

PLDI 1993. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Albuquerque, NM, June 1993. ACM SIGPLAN Notices 28(6), ACM Press. doi: 10.1145/155090.

PLDI 1997. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Las Vegas, NV, June 1997. ACM SIGPLAN Notices 32(5), ACM Press. doi: 10.1145/258915.

PLDI 1999. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999. ACM SIGPLAN Notices 34(5), ACM Press. doi: 10.1145/301618.

PLDI 2000. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Vancouver, Canada, June 2000. ACM SIGPLAN Notices 35(5), ACM Press. doi: 10.1145/349299.

PLDI 2001. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Snowbird, UT, June 2001. ACM SIGPLAN Notices 36(5), ACM Press. doi: 10.1145/378795.

PLDI 2002. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Berlin, Germany, June 2002. ACM SIGPLAN Notices 37(5), ACM Press. doi: 10.1145/512529.

PLDI 2006, Michael I. Schwartzbach and Thomas Ball, editors. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Ottawa, Canada, June 2006. ACM SIGPLAN Notices 41(6), ACM Press. doi: 10.1145/1133981.

PLDI 2008, Rajiv Gupta and Saman P. Amarasinghe, editors. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Tucson, AZ, June 2008. ACM SIGPLAN Notices 43(6), ACM Press. doi: 10.1145/1375581.

POPL 1994. *21st Annual ACM Symposium on Principles of Programming Languages*, Portland, OR, January 1994. ACM Press. doi: 10.1145/174675.

POPL 2003. *30th Annual ACM Symposium on Principles of Programming Languages*, New Orleans, LA, January 2003. ACM SIGPLAN Notices 38(1), ACM Press. doi: 10.1145/604131.

POS 1992, Antonio Albano and Ronald Morrison, editors. *5th International Workshop on Persistent Object Systems (September, 1992)*, San Miniato, Italy, 1992. Workshops in Computing, Springer.

Tony Printezis. Hot-Swapping between a Mark&Sweep and a Mark&Compact Garbage Collector in a Generational Environment. In JVM 2001.

Tony Printezis. On measuring garbage collection responsiveness. *Science of Computer Programming*, 62(2):164–183, October 2006. doi: 10.1016/j.scico.2006.02.004.

Tony Printezis and David Detlefs. A generational mostly-concurrent garbage collector. In ISMM 2000, pages 143–154. doi: 10.1145/362422.362480.

Tony Printezis and Alex Garthwaite. Visualising the Train garbage collector. In ISMM 2002, pages 100–105. doi: 10.1145/512429.512436.

Feng Qian and Laurie Hendren. An adaptive, region-based allocator for Java. In ISMM 2002, pages 127–138. doi: 10.1145/512429.512446. Sable Technical Report 2002–1 provides a longer version.

Christian Queinnec, Barbara Beaudoing, and Jean-Pierre Queille. Mark DURING Sweep rather than Mark THEN Sweep. In Eddy Odijk, Martin Rem, and Jean-Claude Syre, editors, *Parallel Architectures and Languages Europe (PARLE)*, Eindhoven, The Netherlands, June 1989, pages 224–237. Volume 365 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3540512845_42.

John H. Reppy. A high-performance garbage collector for Standard ML. Technical memorandum, AT&T Bell Laboratories, Murray Hill, NJ, December 1993.

Sven Gestegøard Robertz and Roger Henriksson. Time-triggered garbage collection: Robust and adaptive real-time GC scheduling for embedded systems. In LCTES 2003, pages 93–102. doi: 10.1145/780732.780745.

J. M. Robson. An estimate of the store size necessary for dynamic storage allocation. *Journal of the ACM*, 18(3):416–423, July 1971. doi: 10.1145/321650.321658.

J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *Journal of the ACM*, 21(3):419–499, July 1974. doi: 10.1145/321832.321846.

J. M. Robson. A bounded storage algorithm for copying cyclic structures. *Communications of the ACM*, 20(6):431–433, June 1977. doi: 10.1145/359605.359628.

J. M. Robson. Storage allocation is NP-hard. *Information Processing Letters*, 11(3):119–125, November 1980. doi: 10.1016/0020-0190(80)90124-6.

Helena C. C. D. Rodrigues and Richard E. Jones. Cyclic distributed garbage collection with group merger. In Eric Jul, editor, *12th European Conference on Object-Oriented Programming*, Brussels, Belgium, July 1998, pages 249–273. Volume 1445 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/BFb0054095. Also UKC Technical report 17–97, December 1997.

Paul Rovner. On adding garbage collection and runtime types to a strongly-typed, statically-checked, concurrent language. Technical Report CSL–84–7, Xerox PARC, Palo Alto, CA, July 1985.

Erik Ruf. Effective synchronization removal for Java. In PLDI 2000, pages 208–218. doi: 10.1145/349299.349327.

Narendran Sachindran and Eliot Moss. MarkCopy: Fast copying GC with less space overhead. In OOPSLA 2003, pages 326–343. doi: 10.1145/949305.949335.

Narendran Sachindran, J. Eliot B. Moss, and Emery D. Berger. MC$^2$: High-performance garbage collection for memory-constrained environments. In OOPSLA 2004, pages 81–98. doi: 10.1145/1028976.1028984.

Konstantinos Sagonas and Jesper Wilhelmsson. Message analysis-guided allocation and low-pause incremental garbage collection in a concurrent language. In ISMM 2004, pages 1–12. doi: 10.1145/1029873.1029875.

Konstantinos Sagonas and Jesper Wilhelmsson. Efficient memory management for concurrent programs that use message passing. *Science of Computer Programming*, 62(2): 98–121, October 2006. doi: 10.1016/j.scico.2006.02.006.

Jon D. Salkild. Implementation and analysis of two reference counting algorithms. Master's thesis, University College, London, 1987.

Patrick M. Sansom and Simon L. Peyton Jones. Generational garbage collection for Haskell. In John Hughes, editor, *Conference on Functional Programming and Computer Architecture*, Copenhagen, Denmark, June 1993, pages 106–116. ACM Press. doi: 10.1145/165180.165195.

Robert A. Saunders. The LISP system for the Q–32 computer. In E. C. Berkeley and Daniel G. Bobrow, editors, *The Programming Language LISP: Its Operation and Applications*, Cambridge, MA, 1974, pages 220–231. Information International, Inc.

Martin Schoeberl. Scheduling of hard real-time garbage collection. *Real-Time Systems*, 45 (3):176–213, 2010. doi: 10.1007/s11241-010-9095-4.

Jacob Seligmann and Steffen Grarup. Incremental mature garbage collection using the train algorithm. In Oscar Nierstrasz, editor, *9th European Conference on Object-Oriented Programming*, øAarhus, Denmark, August 1995, pages 235–252. Volume 952 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/3-540-49538-X_12.

Robert A. Shaw. *Empirical Analysis of a Lisp System*. PhD thesis, Stanford University, 1988. Technical Report CSL-TR-88-351.

Yefim Shuf, Manish Gupta, Hubertus Franke, Andrew Appel, and Jaswinder Pal Singh. Creating and preserving locality of Java applications at allocation and garbage collection times. In OOPSLA 2002, pages 13–25. doi: 10.1145/582419.582422.

Fridtjof Siebert. Eliminating external fragmentation in a non-moving garbage collector for Java. In *Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, San Jose, CA, November 2000, pages 9–17. ACM Press. doi: 10.1145/354880.354883.

Fridtjof Siebert. Limits of parallel marking collection. In ISMM 2008, pages 21–29. doi: 10.1145/1375634.1375638.

Fridtjof Siebert. Concurrent, parallel, real-time garbage-collection. In ISMM 2010, pages 11–20. doi: 10.1145/1806651.1806654.

Fridtjof Siebert. Guaranteeing non-disruptiveness and real-time deadlines in an incremental garbage collector. In ISMM 1998, pages 130–137. doi: 10.1145/286860.286874.

Fridtjof Siebert. Hard real-time garbage collection in the Jamaica Virtual Machine. In *6th International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, Hong Kong, 1999, pages 96–102. IEEE Press, IEEE Computer Society Press. doi: 10.1109/RTCSA.1999.811198.

David Siegwart and Martin Hirzel. Improving locality with parallel hierarchical copying GC. In ISMM 2006, pages 52–63. doi: 10.1145/1133956.1133964.

Jeremy Singer, Gavin Brown, Mikel Lujan, and Ian Watson. Towards intelligent analysis techniques for object pretenuring. In *ACM International Symposium on Principles and Practice of Programming in Java*, Lisbon, Portugal, September 2007a, pages 203–208. Volume 272 of *ACM International Conference Proceeding Series*. doi: 10.1145/1294325.1294353.

Jeremy Singer, Gavin Brown, Ian Watson, and John Cavazos. Intelligent selection of application-specific garbage collectors. In ISMM 2007, pages 91–102. doi: 10.1145/1296907.1296920.

Vivek Singhal, Sheetal V. Kakkad, and Paul R. Wilson. Texas: an efficient, portable persistent store. In POS 1992, pages 11–33.

Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):562–686, July 1985. doi: 10.1145/3828.3835.

Patrick Sobalvarro. A lifetime-based garbage collector for Lisp systems on general-purpose computers. Bachelor of Science thesis AITR-1417, MIT AI Lab, February 1988. doi: 1721.1/6795.

Sunil Soman and Chandra Krintz. Efficient and general on-stack replacement for aggressive program specialization. In *International Conference on Software Engineering Research and Practice (SERP) & Conference on Programming Languages and Compilers, Volume 2*, Las Vegas, NV, June 2006, pages 925–932. CSREA Press.

Sunil Soman, Chandra Krintz, and David Bacon. Dynamic selection of application-specific garbage collectors. In ISMM 2004, pages 49–60. doi: 10.1145/1029873.1029880.

Sunil Soman, Laurent Daynès, , and Chandra Krintz. Task-aware garbage collection in a multi-tasking virtual machine. In ISMM 2006, pages 64–73. doi: 10.1145/1133956.1133965.

Sunil Soman, Chandra Krintz, and Laurent Daynès. $MTM^2$: Scalable memory management for multi-tasking managed runtime environments. In Jan Vitek, editor, *22nd European Conference on Object-Oriented Programming*, Paphos, Cyprus, July 2008, pages 335–361. Volume 5142 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/978-3-540-70592-5_15.

Daniel Spoonhower, Guy Blelloch, and Robert Harper. Using page residency to balance tradeoffs in tracing garbage collection. In VEE 2005, pages 57–67. doi: 10.1145/1064979.1064989.

James W. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Transactions on Computer Systems*, 2(3):155–180, May 1984. doi: 10.1145/190.194.

James William Stamos. A large object-oriented virtual memory: Grouping strategies, measurements, and performance. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, April 1982. doi: 1721.1/15807.

Thomas A. Standish. *Data Structure Techniques*. Addison-Wesley, 1980.

Guy L. Steele. Multiprocessing compactifying garbage collection. *Communications of the ACM*, 18(9):495–508, September 1975. doi: 10.1145/361002.361005.

Guy L. Steele. Corrigendum: Multiprocessing compactifying garbage collection. *Communications of the ACM*, 19(6):354, June 1976. doi: 10.1145/360238.360247.

Peter Steenkiste. *Lisp on a Reduced-Instruction-Set Processor: Characterization and Optimization*. PhD thesis, Stanford University, March 1987. Available as Technical Report CSL-TR-87-324.

Peter Steenkiste. The impact of code density on instruction cache performance. In *16th Annual International Symposium on Computer Architecture*, Jerusalem, Israel, May 1989, pages 252–259. IEEE Press. doi: 10.1145/74925.74954.

Bjarne Steensgaard. Thread-specific heaps for multi-threaded programs. In ISMM 2000, pages 18–24. doi: 10.1145/362422.362432.

Darko Stefanović. *Properties of Age-Based Automatic Memory Reclamation Algorithms*. PhD thesis, University of Massachusetts, 1999.

Darko Stefanović and J. Eliot B. Moss. Characterisation of object behaviour in Standard ML of New Jersey. In *ACM Conference on LISP and Functional Programming*, Orlando, FL, June 1994, pages 43–54. ACM Press. doi: 10.1145/182409.182428.

Darko Stefanović, Kathryn S. McKinley, and J. Eliot B. Moss. Age-based garbage collection. In OOPSLA 1999, pages 370–381. doi: 10.1145/320384.320425.

Darko Stefanović, Matthew Hertz, Stephen Blackburn, Kathryn McKinley, and J. Eliot Moss. Older-first garbage collection in practice: Evaluation in a Java virtual machine. In *Workshop on Memory System Performance*, Berlin, Germany, June 2002, pages 25–36. ACM SIGPLAN Notices 38(2 supplement), ACM Press. doi: 10.1145/773146.773042.

V. Stenning. On-the-fly garbage collection. Unpublished notes, cited by Gries [1977], 1976.

C. J. Stephenson. New methods of dynamic storage allocation (fast fits). In *9th ACM Symposium on Operating Systems Principles*, Bretton Woods, NH, October 1983, pages 30–32. ACM SIGOPS Operating Systems Review 17(5), ACM Press. doi: 10.1145/800217.806613.

James M. Stichnoth, Guei-Yuan Lueh, and Michal Cierniak. Support for garbage collection at every instruction in a Java compiler. In PLDI 1999, pages 118–127. doi: 10.1145/301618.301652.

Will R. Stoye, T. J. W. Clarke, and Arthur C. Norman. Some practical methods for rapid combinator reduction. In LFP 1984, pages 159–166. doi: 10.1145/800055.802032.

Sun Microsystems. Memory management in the Java HotSpot Virtual Machine, April 2006. Technical White Paper.

H. Sundell. Wait-free reference counting and memory management. In *19th International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, April 2005. IEEE Computer Society Press. doi: 10.1109/IPDPS.2005.451.

Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), March 2005.

M. Swanson. An improved portable copying garbage collector. OPnote 86–03, University of Utah, February 1986.

M. Tadman. Fast-fit: A new hierarchical dynamic storage allocation technique. Master's thesis, University of California, Irvine, 1978.

David Tarditi. Compact garbage collection tables. In ISMM 2000, pages 50–58. doi: 10.1145/362422.362437.

Gil Tene, Balaji Iyengar, and Michael Wolf. C4: The continuously concurrent compacting collector. In ISMM 2011, pages 79–88. doi: 10.1145/1993478.1993491.

S. Thomas, W. Charnell, S. Darnell, B. A. A. Dias, J. G. P. Kramskoy, J. Sextonand, J. Wynn, K. Rautenbach, and W. Plummer. Low-contention grey object sets for concurrent, marking garbage collection. United States Patent Application, 20020042807, 1998.

Stephen P. Thomas. *The Pragmatics of Closure Reduction*. PhD thesis, The Computing Laboratory, University of Kent at Canterbury, October 1993.

Stephen P. Thomas. Having your cake and eating it: Recursive depth-first copying garbage collection with no extra stack. Personal communication, May 1995a.

Stephen P. Thomas. Garbage collection in shared-environment closure reducers: Space-efficient depth first copying using a tailored approach. *Information Processing Letters*, 56(1):1–7, October 1995b. doi: 10.1016/0020-0190(95)00131-U.

Stephen P. Thomas and Richard E. Jones. Garbage collection for shared environment closure reducers. Technical Report 31–94, University of Kent and University of Nottingham, December 1994.

Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value $\lambda$-calculus using a stack of regions. In POPL 1994, pages 188–201. doi: 10.1145/174675.177855.

Mads Tofte, Lars Birkedal, Martin Elsman, and Niels Hallenberg. A retrospective on region-based memory management. *Higher-Order and Symbolic Computation*, 17(3): 245–265, September 2004. doi: 10.1023/B:LISP.0000029446.78563.a4.

David A. Turner. A new implementation technique for applicative languages. *Software: Practice and Experience*, 9:31–49, January 1979. doi: 10.1002/spe.4380090105.

David M. Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. In *ACM/SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Pittsburgh, PA, April 1984, pages 157–167. ACM SIGPLAN Notices 19(5), ACM Press. doi: 10.1145/800020.808261.

David M. Ungar. *The Design and Evaluation of a High Performance Smalltalk System*. ACM distinguished dissertation 1986. MIT Press, 1986.

David M. Ungar and Frank Jackson. Tenuring policies for generation-based storage reclamation. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, San Diego, CA, November 1988, pages 1–17. ACM SIGPLAN Notices 23(11), ACM Press. doi: 10.1145/62083.62085.

David M. Ungar and Frank Jackson. An adaptive tenuring policy for generation scavengers. *ACM Transactions on Programming Languages and Systems*, 14(1):1–27, 1992. doi: 10.1145/111186.116734.

Maxime van Assche, Joël Goossens, and Raymond R. Devillers. Joint garbage collection and hard real-time scheduling. *Journal of Embedded Computing*, 2(3–4):313–326, 2006. Also published in RTS'05 International Conference on Real-Time Systems, 2005.

Martin Vechev. *Derivation and Evaluation of Concurrent Collectors*. PhD thesis, University of Cambridge, 2007.

Martin Vechev, David F. Bacon, Perry Cheng, and David Grove. Derivation and evaluation of concurrent collectors. In Andrew P. Black, editor, *19th European Conference on Object-Oriented Programming*, Glasgow, Scotland, July 2005, pages 577–601. Volume 3586 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/11531142_25.

Martin T. Vechev, Eran Yahav, and David F. Bacon. Correctness-preserving derivation of concurrent garbage collection algorithms. In PLDI 2006, pages 341–353. doi: 10.1145/1133981.1134022.

Martin T. Vechev, Eran Yahav, David F. Bacon, and Noam Rinetzky. CGCExplorer: A semi-automated search procedure for provably correct concurrent collectors. In Jeanne Ferrante and Kathryn S. McKinley, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2007, pages 456–467. ACM SIGPLAN Notices 42(6), ACM Press. doi: 10.1145/1250734.1250787.

VEE 2005, Michael Hind and Jan Vitek, editors. *1st ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Chicago, IL, June 2005. ACM Press. doi: 10.1145/1064979.

David Vengerov. Modeling, analysis and throughput optimization of a generational garbage collector. In ISMM 2009, pages 1–9. doi: 10.1145/1542431.1542433.

Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 29(4): 229–239, April 1980. doi: 10.1145/358841.358852.

Michal Wegiel and Chandra Krintz. The mapping collector: Virtual memory support for generational, parallel, and concurrent compaction. In Susan J. Eggers and James R. Larus, editors, *13th International Conference on Architectural Support for Programming Languages and Operating Systems*, Seattle, WA, March 2008, pages 91–102. ACM SIGPLAN Notices 43(3), ACM Press. doi: 10.1145/1346281.1346294.

J. Weizenbaum. Recovery of reentrant list structures in SLIP. *Communications of the ACM*, 12(7):370–372, July 1969. doi: 10.1145/363156.363159.

Adam Welc, Suresh Jagannathan, and Antony L. Hosking. Transactional monitors for concurrent objects. In Martin Odersky, editor, *18th European Conference on Object-Oriented Programming*, Oslo, Norway, June 2004, pages 519–542. Volume 3086 of *Lecture Notes in Computer Science*, Springer-Verlag. doi: 10.1007/978-3-540-24851-4_24.

Adam Welc, Suresh Jagannathan, and Antony L. Hosking. Safe futures for Java. In OOPSLA 2005, pages 439–453. doi: 10.1145/1094811.1094845.

Jon L. White. Address/memory management for a gigantic Lisp environment, or, GC Considered Harmful. In *LISP Conference*, Stanford University, CA, August 1980, pages 119–127. ACM Press. doi: 10.1145/800087.802797.

Jon L. White. Three issues in objected-oriented garbage collection. In GC 1990.

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenstrøm. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computer Systems*, 7(3), April 2008. doi: 10.1145/1347375.1347389.

Jesper Wilhelmsson. *Efficient Memory Management for Message-Passing Concurrency — part I: Single-threaded execution*. Licentiate thesis, Uppsala University, May 2005.

Paul R. Wilson. A simple bucket-brigade advancement mechanism for generation-based garbage collection. *ACM SIGPLAN Notices*, 24(5):38–46, May 1989. doi: 10.1145/66068.66070.

Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, January 1994. Expanded version of the IWMM92 paper.

Paul R. Wilson and Mark S. Johnstone. Truly real-time non-copying garbage collection. In GC 1993.

Paul R. Wilson and Thomas G. Moher. A card-marking scheme for controlling intergenerational references in generation-based garbage collection on stock hardware. *ACM SIGPLAN Notices*, 24(5):87–92, 1989a. doi: 10.1145/66068.66077.

Paul R. Wilson and Thomas G. Moher. Design of the opportunistic garbage collector. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, New Orleans, LA, October 1989b, pages 23–35. ACM SIGPLAN Notices 24(10), ACM Press. doi: 10.1145/74877.74882.

Paul R. Wilson, Michael S. Lam, and Thomas G. Moher. Effective "static-graph" reorganization to improve locality in garbage-collected systems. In PLDI 1991, pages 177–191. doi: 10.1145/113445.113461.

Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. Dynamic storage allocation: A survey and critical review. In IWMM 1995, pages 1–116. doi: 10.1007/3-540-60368-9_19.

Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. Memory allocation policies reconsidered. Unpublished manuscript, 1995b.

David S. Wise. The double buddy-system. Computer Science Technical Report TR79, Indiana University, Bloomington, IN, December 1978.

David S. Wise. Stop-and-copy and one-bit reference counting. Computer Science Technical Report 360, Indiana University, March 1993a. See also Wise [1993b].

David S. Wise. Stop-and-copy and one-bit reference counting. *Information Processing Letters*, 46(5):243–249, July 1993b. doi: 10.1016/0020-0190(93)90103-G.

David S. Wise and Daniel P. Friedman. The one-bit reference count. *BIT*, 17(3):351–359, 1977. doi: 10.1007/BF01932156.

P. Tucker Withington. How real is "real time" garbage collection? In GC 1991.

Mario I. Wolczko and Ifor Williams. Multi-level GC in a high-performance persistent object system. In POS 1992, pages 396–418.

Ming Wu and Xiao-Feng Li. Task-pushing: a scalable parallel GC marking algorithm without synchronization operations. In *IEEE International Parallel and Distribution Processing Symposium (IPDPS)*, Long Beach, CA, March 2007, pages 1–10. doi: 10.1109/IPDPS.2007.370317.

Feng Xian, Witawas Srisa-an, C. Jia, and Hong Jiang. AS-GC: An efficient generational garbage collector for Java application servers. In ECOOP 2007, pages 126–150. doi: 10.1007/978-3-540-73589-2_7.

Ting Yang, Emery D. Berger, Matthew Hertz, Scott F. Kaplan, and J. Eliot B. Moss. Autonomic heap sizing: Taking real memory into account. In ISMM 2004, pages 61–72. doi: 10.1145/1029873.1029881.

Taiichi Yuasa. Real-time garbage collection on general-purpose machines. *Journal of Systems and Software*, 11(3):181–198, March 1990. doi: 10.1016/0164-1212(90)90084-Y.

Karen Zee and Martin Rinard. Write barrier removal by static analysis. In OOPSLA 2002, pages 191–210. doi: 10.1145/582419.582439.

Chengliang Zhang, Kirk Kelsey, Xipeng Shen, Chen Ding, Matthew Hertz, and Mitsunori Ogihara. Program-level adaptive memory management. In ISMM 2006, pages 174–183. doi: 10.1145/1133956.1133979.

W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, SE-13 (5):564–577, May 1987. doi: 10.1109/TSE.1987.233201.

Benjamin Zorn. Barrier methods for garbage collection. Technical Report CU-CS-494-90, University of Colorado, Boulder, November 1990.

Benjamin Zorn. The measured cost of conservative garbage collection. *Software: Practice and Experience*, 23:733–756, 1993. doi: 10.1002/spe.4380230704.

Benjamin G. Zorn. *Comparative Performance Evaluation of Garbage Collection Algorithms*. PhD thesis, University of California, Berkeley, March 1989. Technical Report UCB/CSD 89/544.

## Colophon

This book was set in Palatino (algorithms in Courier) with pdftex (from the TeX Live 2010 distribution). The Illustrations were drawn with Adobe Illustrator CS3. We found the following packages to be useful: `comment` (comments), `paralist` (in-paragraph lists), `geometry` (page size), `crop` (crop marks), `xspace` (space suppression), `setspace` (line spacing), `fnbreak` (detect footnotes spread over more than one page), `afterpage` (page break control), `multicol` (multiple columns), `tabularx` (tabular material), `multirow` (multiple rows and columns in tables), `dcolumn` ("decimal points" in tables), `graphicx` and `epstopdf` (graphics), `subfig` (subfigures), `rotating` (rotate objects), `listings` (algorithm listings), `caption` (captions), `mathpazo` (typesetting mathematics to match Palatino body text), `amssymb` and `amsmath` (mathematics), `hyperref` (hyperlinks and back-references), `glossaries` (glossaries), `natbib` (bibliography), `makeidx` and `index` (index).