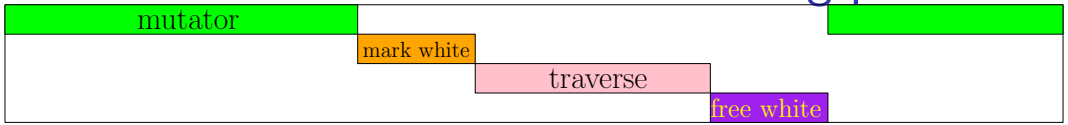
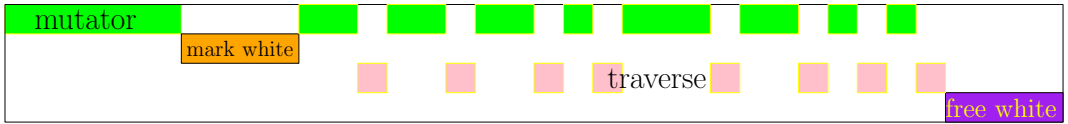
GC VI: Generational Improvements

David Bremner

April 6, 2024

Reducing pause times





- For many applications, long mutator pauses for GC cannot be tolerated

Heap layout

	0	1	2	3	4	5	6	7	8	9
0	1	'free	'free	'free	'free	'free	'free	'free	'free	'free
10	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
20	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
30	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
40	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
50	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
60	'free	'free	'free	'free	65	'free	'free	'free	'free	'free
70	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
80	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
90	'free	'free	'free	'free	'free	'free	'out	100	0	#f
100	'free-n	#f	156	'free	'free	'free	'free	'free	'free	'free
110	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
120	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
130	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
140	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
150	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
160	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
170	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
180	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
190	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
200	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
210	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
220	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
230	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
240	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
250	'free	'free	'free	'free	'free	'free				

Allocation Clock

- There are many possible strategies for interleaving GC and mutator
- Here we take a simple approach and trigger some GC after every k words of allocation.

```
(define (copy/alloc n some-roots more-roots)
  (define next (find-free-space (heap-ref
     (free-list-head)) #f n))

  ;; incremental collection
  (heap-set!
   (step-count-word)
   (+ n (heap-ref (step-count-word))))
  (when (>= (heap-ref (step-count-word))
            step-length)
    (traverse/incre-mark (next/cont)))
```

Incremental Traversal

```
(define (traverse/incre-mark loc)
  (cond
    [(not loc) (heap-set! (step-count-word) 0)]
    [else
     (case (heap-ref loc)
       ; ⋮
       [(pair grey-pair)
        (mark-black loc)
        (step/count 3) ; 3 words scanned
        (push/cont (heap-ref (+ loc 2)))
        (push/cont (heap-ref (+ loc 1)))
        (continue/incre-mark)]
       ; ⋮
       [else (error 'traverse/incre-mark
                    "wrong tag at ~s" loc)])]))
```

Saving gray objects

```
(define (push/cont ptr)
  (mark-grey ptr)
  (when (not (member (heap-ref ptr) '(pair flat proc)))
    (define loc (heap-ref (tracing-head-word)))

      ; ⋮
     (let ([top
            (if loc (sub1 loc) (- (2nd-gen-end) 1))])
       (heap-set! top ptr)
       (heap-set! (tracing-head-word) top))))
```

Marking / Freeing white nodes

- Combine `mark-white!`, `free-white!` into single pass of heap
- Typical approach, e.g Alg. 2.3 in GC handbook
- Can be made incremental, see *lazy sweeping*
- not as bad as it sounds, see Section 2.5 in GC handbook

```
(case (heap-ref loc)
  [(flat pair proc) (mark-white! loc)
   #;(⋮)]
  [(white-flat white-pair white-proc
               free free-2 free-n)
   (let* ([length (object-length loc)]
          [join (and last-start so-far)]
          [start (if join last-start loc)]
          [newlen (if join (+ so-far length) length)])
     (free/mark-white!
```

Two stage collection

```
(define (malloc n . extra-roots)
  (define addr (heap-ref (alloc-word)))
  (cond
    [(space-on-young-heap? addr n)
     (heap-set! (alloc-word) (+ addr n))
     addr]
    [else
     (collect-garbage extra-roots)
     (unless (need-forwarding-pointers? extra-roots)
       (free-1st-gen))
     (unless (space-on-young-heap? 1 n)
       (error 'alloc "object too large"))
     (heap-set! (alloc-word) (+ 1 n))
     1]))

;; collect-garbage : roots roots -> void
```

Sum Demo

sum

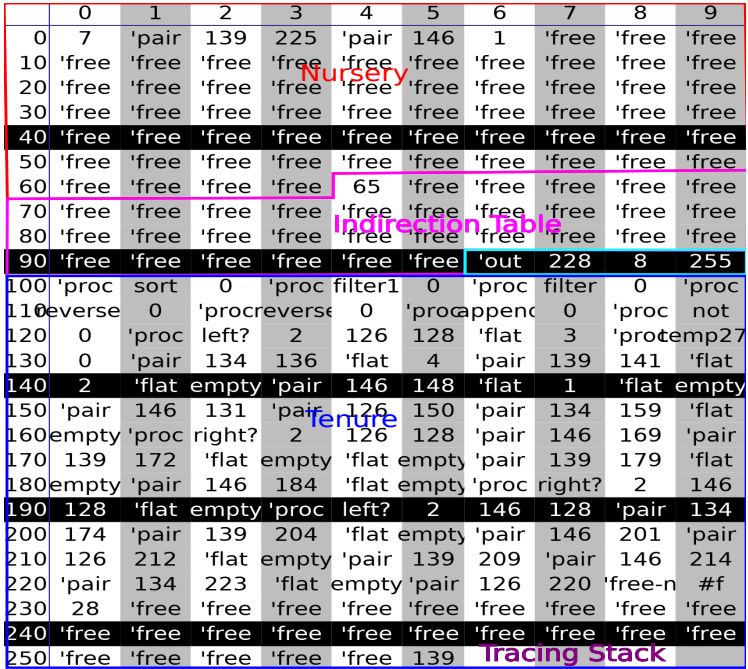
```
(allocator-setup "incremental.rkt" 256)
(define (sum lst)
  (cond
    [(empty? lst) 0]
    [else (+ (first lst) (sum (rest lst)))]))

(sum '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
   20 21 22 23))
```

Quicksort Demo

sort

```
(define (sort l less?)
  (if (empty? l) l
      (let*
          ([pivot (first l)]
           [left? (lambda (x) (less? x pivot))]
           [left (filter left? (rest l))]
           [right? (lambda (x) (not (less? x pivot)))]
           [right (filter right? (rest l))])
        (append (append (sort left less?)
                        (cons pivot empty))
                (sort right less?)))))
(sort '(3 1 4 2) (lambda (a b) (< a b)))
```

Heap State



	0	1	2	3	4	5	6	7	8	9
0	7	'pair	139	225	'pair	146	1	'free	'free	'free
10	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
20	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
30	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
40	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
50	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
60	'free	'free	'free	'free	65	'free	'free	'free	'free	'free
70	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
80	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
90	'free	'free	'free	'free	'free	'free	'out	228	8	255
100	'proc	sort	0	'proc	filter1	0	'proc	filter	0	'proc
110	reverse	0	'proc	reverse	0	'proc	append	0	'proc	not
120	0	'proc	left?	2	126	128	'flat	3	'proc	temp27
130	0	'pair	134	136	'flat	4	'pair	139	141	'flat
140	2	'flat	empty	'pair	146	148	'flat	1	'flat	empty
150	'pair	146	131	'pair	126	150	'pair	134	159	'flat
160	empty	'proc	right?	2	126	128	'pair	146	169	'pair
170	139	172	'flat	empty	'flat	empty	'pair	139	179	'flat
180	empty	'pair	146	184	'flat	empty	'proc	right?	2	146
190	128	'flat	empty	'proc	left?	2	146	128	'pair	134
200	174	'pair	139	204	'flat	empty	'pair	146	201	'pair
210	126	212	'flat	empty	'pair	139	209	'pair	146	214
220	'pair	134	223	'flat	empty	'pair	126	220	'free-n	#f
230	28	'free	'free	'free	'free	'free	'free	'free	'free	'free
240	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
250	'free	'free	'free	'free	'free	139				

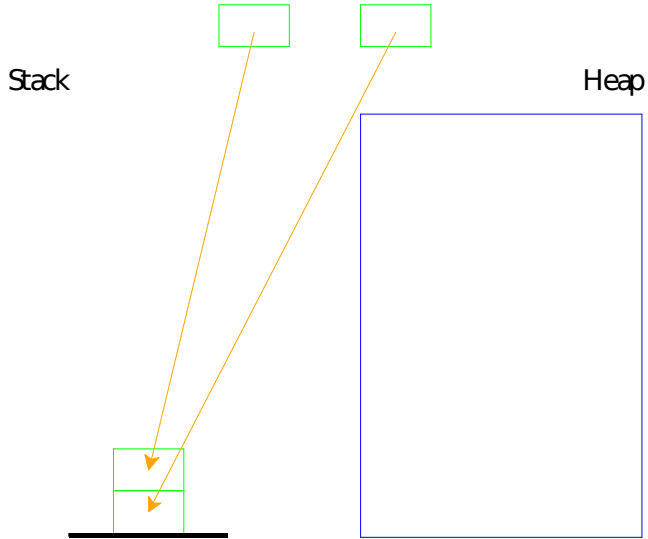
Efficient Continuation Based Interpreters

- Even with generational GC, allocating every continuation in the heap is potentially slow.
- One solution is to use the C stack for our nursery.
- Continuations are translated to C functions (with an extra parameter for the next continuation).

Cheney on the MTA

- Two-space copying collection = Cheney's algorithm
- "on the MTA". in-joke about the Boston transit system, and never returning.

Cheney on the MTA



References / Acknowledgements

- Cheney-Baker is used in Chicken Scheme
- Lecture 23 based in part on slides by Vincent St. Amour.
- CONS Should Not CONS Its Arguments, Part II: Cheney on the M.T.A., Henry Baker, 1994
  https://dl.acm.org/doi/10.1145/214448.214454
  https://www.plover.com/~mjd/misc/hbaker-archive/CheneyMTA.html