# Garbage collection considered harmful?

John Fremlin

http://john.freml.in

2009 May 28

# Contents

This talk is about garbage collection (GC), and why it might be overused.

# Outline

# In practice

- In practice, often the particular algorithm (usually generational mark-sweep compacting garbage collection) or implementation of it is not so important.
- Just tune the limits so that it doesn't run too much and doesn't tenure many temporary objects.
- But garbage collection imposes overheads even when a collection is not taking place.

# In practice

- In practice, often the particular algorithm (usually generational mark-sweep compacting garbage collection) or implementation of it is not so important.
- Just tune the limits so that it doesn't run too much and doesn't tenure many temporary objects.
- But garbage collection imposes overheads even when a collection is not taking place.

# Memory layout

- Memory is divided into two areas, old and new,
- Allocations are made by incrementing a pointer in a block of new space (the nursery). When the block is full, the garbage collector runs a local collection.
- Old space is protected (pages marked read-only and moved into new space if modified).

# Allocations

- Allocations have a type and length header, so that pointers inside them can be discovered.
- Function entry checks for nursery overflow.
- Closures/thunks are invisible allocations.

# Common implementation issues

- Terrible handling of out of memory conditions (crash, lockup).
- Collections require all threads to stop.
- Weak pointers are inefficient.
- Buffers keep moving around.

# Outline

# Without GC

- malloc/free - this is the favourite straw man of garbage lovers
- The stack!
- Pools for particular object types.

# Without GC

- malloc/free - this is the favourite straw man of garbage lovers
- The stack!
- Pools for particular object types.

# More common patterns

- Reference counting.
- Clear responsibility for deallocation passed with pointers.
- Never freeing (fork/exit).

# More common patterns

- Reference counting.
- Clear responsibility for deallocation passed with pointers.
- Never freeing (fork/exit).

# Issues

- Wrongly freeing or forgetting to free.
- Fragmentation.
- Programming effort and discipline.
- Wasted copying or extra work for the allocator.

# Outline

# Uncertain complexity

- Repeating something $n$ times takes more than $n$ times as long.
- Often observed 20% efficiency (where a badly tuned GC runs 80% of the time).
- How much memory or time is needed?

# The distributed future

- Concurrent GC is hard and inherently difficult to scale.
- Versus object pools or the stack, GC does more memory accesses.
- Requires more memory barriers.
- Erlang has some good ideas here.

# Outline

# GC has theoretical as well as practical issues

- Garbage collection was invented for Lisp. It was widely derided.
- It is now everywhere.
- Time for the pendulum to swing back.