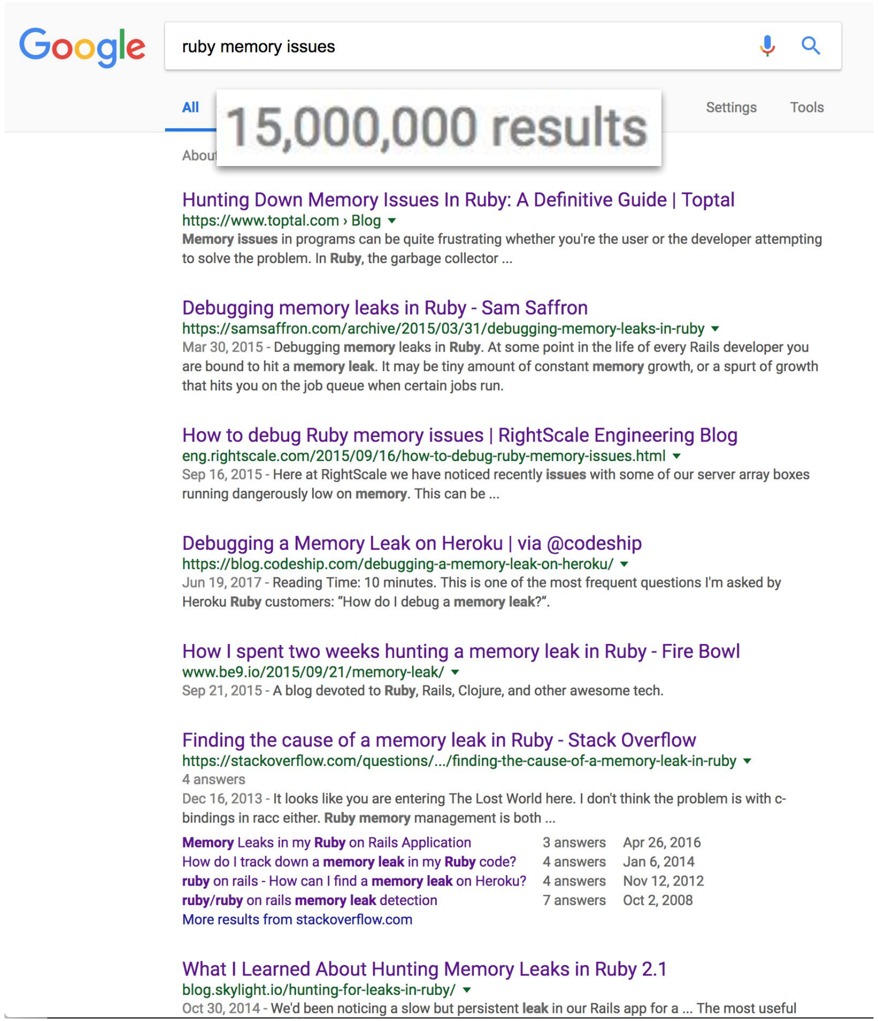
TOTAL RECALL

A Deep Dive into Ruby Memory Management


ruby memory issues

All

Settings Tools

About 15,000,000 results

Hunting Down Memory Issues In Ruby: A Definitive Guide | Toptal
https://www.toptal.com › Blog
Memory issues in programs can be quite frustrating whether you're the user or the developer attempting to solve the problem. In **Ruby**, the garbage collector ...

Debugging memory leaks in Ruby - Sam Saffron
https://samsaffron.com/archive/2015/03/31/debugging-memory-leaks-in-ruby
Mar 30, 2015 - Debugging **memory** leaks in **Ruby**. At some point in the life of every Rails developer you are bound to hit a **memory leak**. It may be tiny amount of constant **memory** growth, or a spurt of growth that hits you on the job queue when certain jobs run.

How to debug Ruby memory issues | RightScale Engineering Blog
eng.rightscale.com/2015/09/16/how-to-debug-ruby-memory-issues.html
Sep 16, 2015 - Here at RightScale we have noticed recently **issues** with some of our server array boxes running dangerously low on **memory**. This can be ...

Debugging a Memory Leak on Heroku | via @codeship
https://blog.codeship.com/debugging-a-memory-leak-on-heroku/
Jun 19, 2017 - Reading Time: 10 minutes. This is one of the most frequent questions I'm asked by Heroku **Ruby** customers: "How do I debug a **memory leak**?".

How I spent two weeks hunting a memory leak in Ruby - Fire Bowl
www.be9.io/2015/09/21/memory-leak/
Sep 21, 2015 - A blog devoted to **Ruby**, Rails, Clojure, and other awesome tech.

Finding the cause of a memory leak in Ruby - Stack Overflow
https://stackoverflow.com/questions/.../finding-the-cause-of-a-memory-leak-in-ruby
4 answers
Dec 16, 2013 - It looks like you are entering The Lost World here. I don't think the problem is with c-bindings in racc either. **Ruby memory** management is both ...

Memory Leaks in my **Ruby** on Rails Application	3 answers	Apr 26, 2016
How do I track down a **memory leak** in my **Ruby** code?	4 answers	Jan 6, 2014
ruby on rails - How can I find a **memory leak** on Heroku?	4 answers	Nov 12, 2012
ruby/ruby on rails **memory leak** detection	7 answers	Oct 2, 2008

More results from stackoverflow.com

What I Learned About Hunting Memory Leaks in Ruby 2.1
blog.skylight.io/hunting-for-leaks-in-ruby/
Oct 30, 2014 - We'd been noticing a slow but persistent **leak** in our Rails app for a ... The most useful

WE GOT BIG REPUTATIONS

A Little About Me

Matt Duszynski

@bsdzunk

github.com/dzunk

- Software engineer at weedmaps
- From Houston, TX
- Ruby developer since 2012
- Interests
    - Coffee
    - Music
    - Golf
- Docker contributor
    - OK, I updated the docs one time

What I'm Talking About

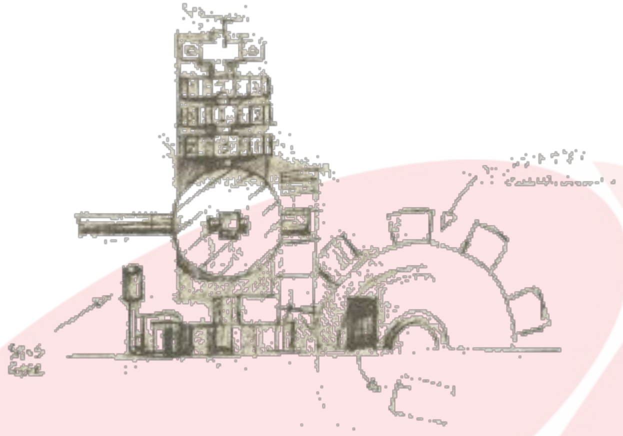
- Terms and concepts
- Under the hood
  - Page allocation
  - Object allocation
  - Garbage collection
  - Visualizing the heap
- Improvements, past and future
- Troubleshooting memory consumption
  - Leak or bloat?
  - Using memory efficiently
  - Tools for the toolbox

What I’m **NOT** Talking About

- Other interpreters have different memory models
- The concepts will overlap (JVM has garbage collection too)
- Specific terms, variables, libraries, etc. will not apply across platforms

Let’s Get On The Same Page

- Allocation
  - OS assigns memory to a Ruby process (`malloc`)
- Deallocation
  - Ruby process returns unused memory to the OS (`free`)
- Heap
  - The entire memory space allocated to a Ruby process
- Page
  - 16 kilobyte section of the heap
  - Smallest amount of memory that can be allocated
- Slot
  - 40 byte section of an individual page
  - One object per slot



How it Works: Allocation

The layer cake of memory abstraction

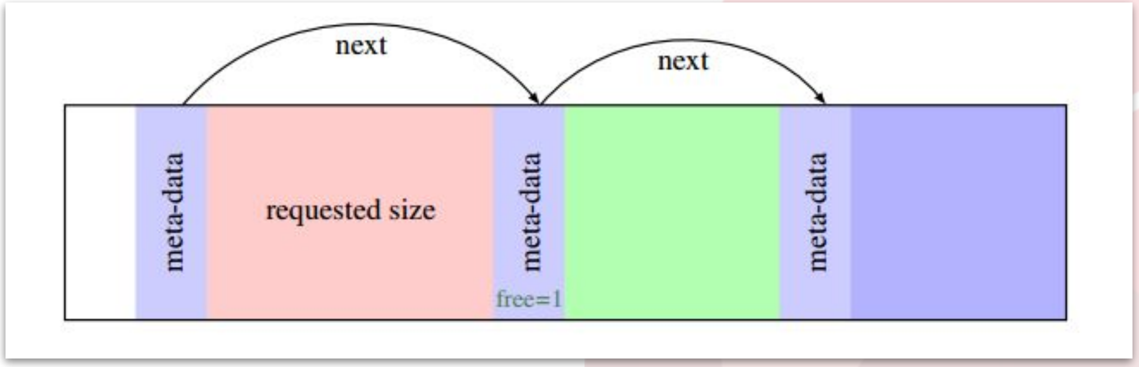
- Ruby VM - `Object.new` in `ObjectSpace`
- Allocator - `malloc`
- Operating system - virtual memory
- Hardware - memory management unit

We’re only talking about the first two

How it Works: Page Allocation

When your process needs memory for anything:

1. Is there enough contiguous free space in the heap?
2. Run garbage collection
3. Is there enough contiguous free space in the heap?
4. Allocate additional pages



How it Works: Object Allocation

`ObjectSpace` – Every living object in a Ruby process

```
[1] pry(main)> Object.new
=> #<Object:0x00007ff5eaa76010>
```

Object address to binary:

```
111111111101011110101010011101 10000000010000
Page address                   Object ID
```

`ObjectSpace` entries are 40 bytes or less. Any non-trivial amount of data lives elsewhere in memory.

How it Works: Object Allocation

`String.new "Lorem ipsum dolor sit amet..."` (445 chars)

1. Create an object in a slot in `ObjectSpace`
   - No room? Add a page with `malloc`
2. Allocate memory for the string itself
   - `malloc(445)`

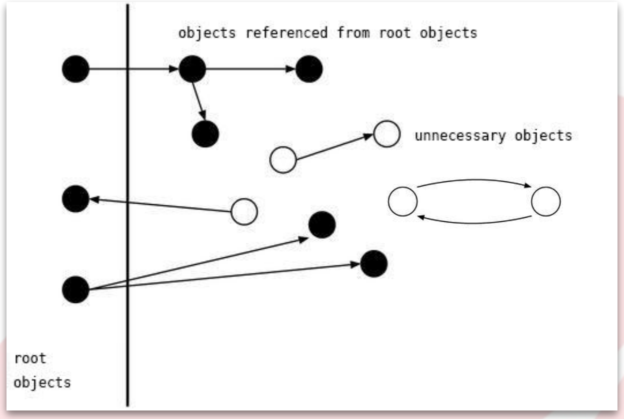
The `ObjectSpace` entry is a pointer to the memory location from step two.

It's in the heap somewhere, but locations aren't guaranteed.
Causes memory fragmentation.

`ObjectSpace` does **not** represent all of Ruby's memory!

How it Works: Garbage Collection

- Mark and sweep
  - Start from root objects
  - Mark all reachable (referenced) objects
  - Free all unmarked objects
  - Unmark all remaining objects
- Advantages
  - Transparent to running program
  - Handles cyclic references
- Disadvantages
  - Program must be suspended during GC runs
  - Leads to memory fragmentation over time without compaction



How it Works: Generational GC

Most objects are dereferenced very soon after creation.

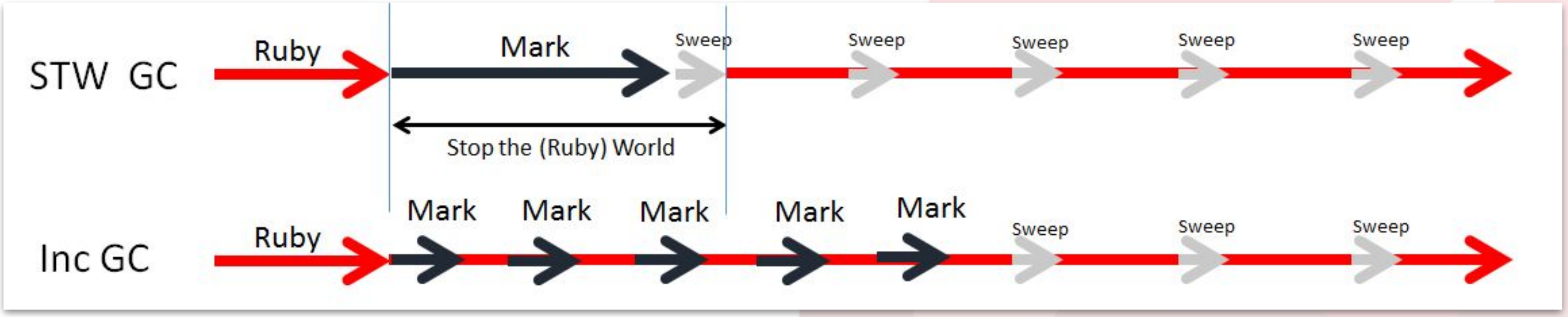
Mark and sweep only objects that have been created since the last GC run.

Faster and more efficient, but you eventually have to traverse all objects.

In Ruby, this is “minor GC” and “major GC”

How it Works: Incremental GC

- Separates “mark” phase and “sweep” phase
- Distributes GC pause time more evenly
  - Better to pause 5 times for 3ms each than one time for 15ms
- Doesn’t decrease overall pause time

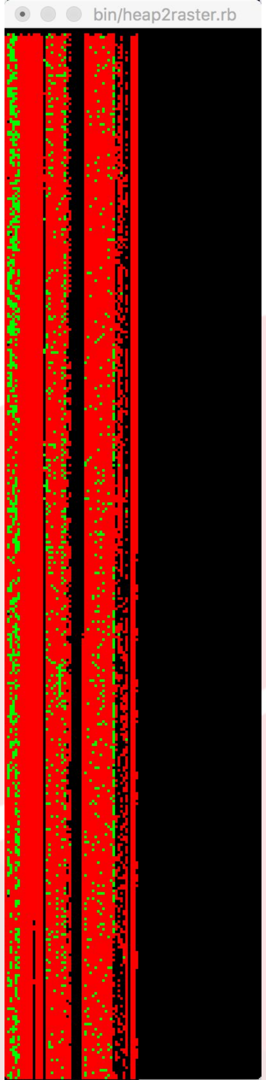


How it Works: Visualization

`heapfrag` gem, by tenderlove

Let’s see what’s going on in real-time

- Black pixel = unused or unallocated memory
- Green pixel = “young” object
- Red pixel = “old” object



Memory Usage Statistics

```
[51] pry(main)> GC.stat
=> {:count=>71,
 :heap_allocated_pages=>4129,
 :heap_sorted_length=>4129,
 :heap_allocatable_pages=>0,
 :heap_available_slots=>1682983,
 :heap_live_slots=>1682338,
 :heap_free_slots=>645,
 :heap_final_slots=>0,
 :heap_marked_slots=>1282624,
 :heap_eden_pages=>4129,
 :heap_tomb_pages=>0,
 :total_allocated_pages=>4129,
 :total_freed_pages=>0,
 :total_allocated_objects=>10316997,
 :total_freed_objects=>8634659,
 :malloc_increase_bytes=>789296,
 :malloc_increase_bytes_limit=>25288021,
 :minor_gc_count=>57,
 :major_gc_count=>14,
 :remembered_wb_unprotected_objects=>7514,
 :remembered_wb_unprotected_objects_limit=>14268,
 :old_objects=>1266437,
 :old_objects_limit=>2336758,
 :oldmalloc_increase_bytes=>12184848,
 :oldmalloc_increase_bytes_limit=>31509677}
```

```
[50] pry(main)> GC::INTERNAL_CONSTANTS

=> {:RVALUE_SIZE=>40, :HEAP_PAGE_OBJ_LIMIT=>408,
:HEAP_PAGE_BITMAP_SIZE=>56,
:HEAP_PAGE_BITMAP_PLANES=>4}
```

Better, Stronger, Faster

Ruby 2.1

- Generational garbage collection

Ruby 2.2

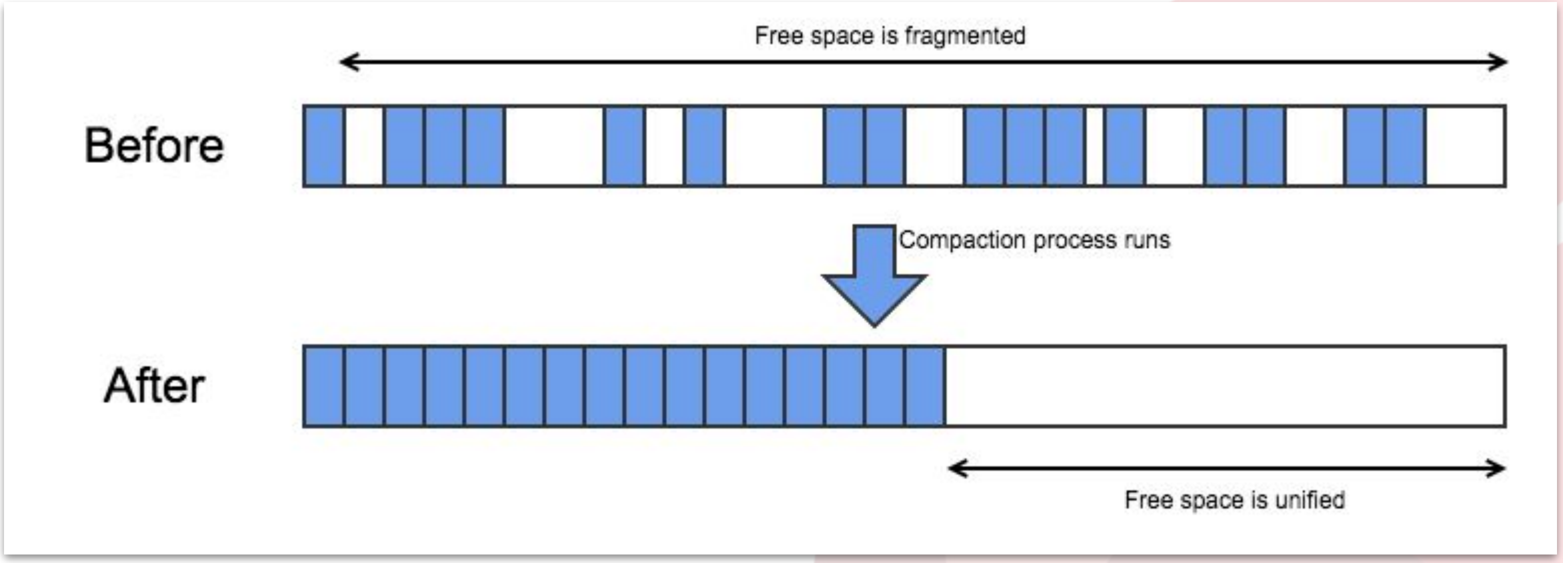
- Garbage collection of symbols
- Incremental garbage collection
- Support for `jemalloc`

Ruby 2.3

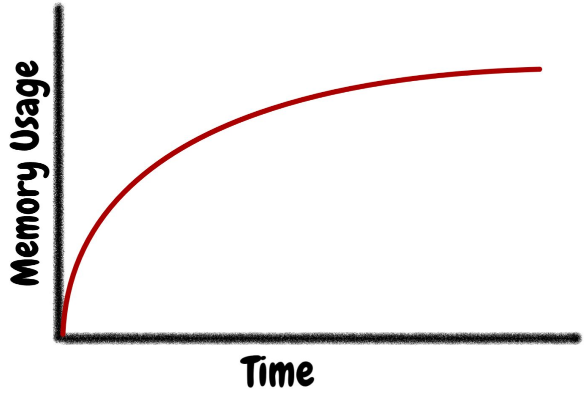
- `frozen_string_literal` pragma directive

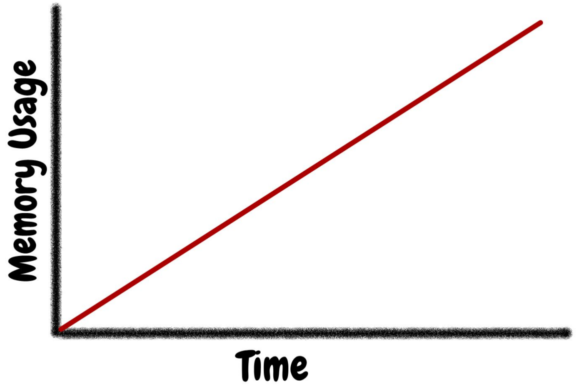
What’s Next?

- Compaction?
- Default to `jemalloc`?



Memory Leak, or Memory Bloat?





Memory Bloat

- Logarithmic
- “Natural” behavior of a Ruby process
- Just download more RAM
- Stepped/rapid growth is probably a code issue

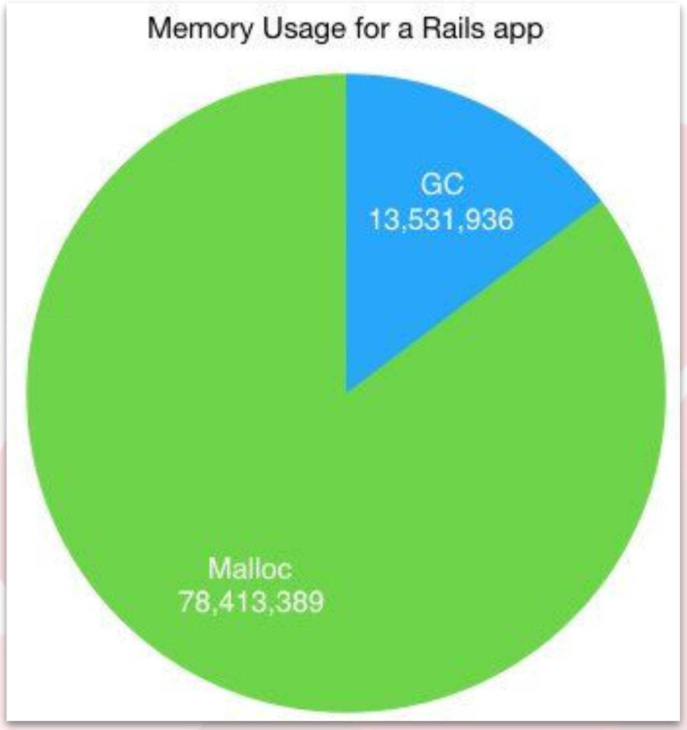
Memory Leak

- Linear
- Will crash your application
- Far less common in Rubyland
- It’s not a leak if your code is doing it intentionally

Use Memory Efficiently

Avoid unnecessary object allocations

- Everything is an object, thus everything your program does will allocate objects.
- This includes deallocating objects.
- Practically speaking, memory used for data outside of `ObjectSpace` will never be freed.
- Common problem: `Enumerator`
- Uncommon solution: `Enumerator::Lazy`



Don’t Load Data You Don’t Need

- At Weedmaps, we deal with a lot of geospatial data
- Geometry data is (relatively) huge

```
[1] pry(main)> 'Texas'.length
=> 5
[2] pry(main)> Region.find_by(name: 'Texas').geometry.to_s.length
=> 1520683
```

- Where is that geometry data in memory?
- `malloc` -ed somewhere outside of `ObjectSpace`

Don’t Load Data You Don’t Need

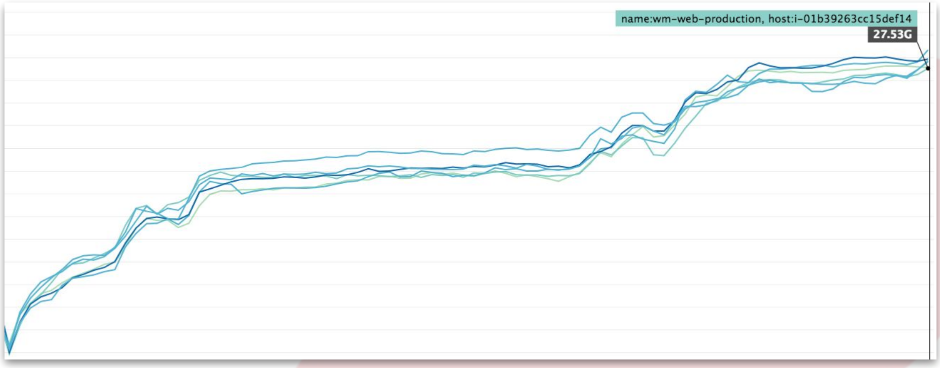
- Added `without_geometry` scope to avoid loading unnecessary data
- Excludes `geometry` column from `SELECT` statements

```
[1] pry(main)> Region.find_by(name: 'Texas').inspect.to_s.length
=> 1521852
[2] pry(main)> Region.without_geometry.find_by(name:
'Texas').inspect.to_s.length
=> 1104
```

- Turns out it wasn’t necessary for almost all requests
- Originally noticed as a query issue, not a memory one

Tools for the Toolbox

- APM
  - New Relic
  - Skylight
  - Scout
- Gems
  - derailed_benchmarks
  - rbtrace
  - memory_profiler
- You
  - It's much easier to avoid memory issues in development than to troubleshoot them in a production application
  - Be aware of how your code uses memory. If you're unsure, profile things *before* you commit them.



Recap

- Terms and concepts
- Under the hood
  - Page allocation
  - Object allocation
  - Garbage collection
  - Visualizing the heap
- Improvements, past and future
- Troubleshooting memory consumption
  - Leak or bloat?
  - Using memory efficiently
  - Tools for the toolbox

Questions?

@bsdzunk

github.com/dzunk