Interactive Portion
Unlock your phones
Selfie Sticks
Two Purposes
Help you identify people who like fun stuff and to have a fun time
Let you know who doesn’t like to have fun or like fun things
Have you seen this?
It’s me!
Hey Siri what's the temperature in Cincy right now

Sorry, I don’t know where that is.
Kinsey Ann Durham @KinseyAnnDurham
Who keeps airdropping me random pics of dogs and cats @rubyconf? ha ha lovin it #rubyconf
pic.twitter.com/oLx7QvHUA8

3 Likes 1 Retweet
Nov 12, 2016 at 10:03 AM via Twitter Web Client
Methods of Memory Management in MRI
Aaron Patterson
@tenderlove

PGP Fingerprint: 4CE9 1B75 A798 28E8 6B1A  A8BB 9531 70BC B4FF AFC6
I am from Seattle
Which is not Ohio
"Ohio"
LeGit
GitHub Certified®
Engineer
Bear Metal
tenderlove
Please call me "Aaron"
("tenderlove" is fine too)
I love cats!
Cats are the best!
I have stickers of my cats
I also have GitHub stickers!
I was in the news recently
Keyboards
New Ruby Features
Soft Typing
Dynamic Typing
Static Typing
GC
Garbage Collector
Memory Terms
GC in MRI
Apps in Production
Scaling Issues
Tuning Issues
What I want you to learn
If you don’t know much about GC
If you know about GC terminology
If you already know the algorithms
If you already know the new stuff
GC Algorithms
(in MRI)
Two sides of a GC
Collection Algorithm
Allocation Algorithm
Introspection API
Tuning Variables
Collection
Algorithm
What type is the MRI collector?
Mark & Sweep

Generational

Incremental
High level:
What is a GC?
Ruby

```ruby
a = [
  { c: 'd' }
]
```

**Tree of Objects**

- **Root**
  - **A**
    - **Array**
  - **B**
    - **Hash**
  - **C**
    - **Symbol**
  - **D**
    - **String**
Tree of Objects

Ruby

```
a = [
    { c: 'd' }
]a = nil
```

- **Root**
  - **A**
    - **Array**
    - **B**
      - **Hash**
      - **C**
        - **Symbol**
      - **D**
        - **String**
Important Words!
Root Set
Garbage
Live Data
GC’s job:
Find unlinked nodes,
then free them
How to find unlinked nodes
Mark & Sweep
2 distinct phases
Mark Phase

- Root
- A
- D
- B
- C
- E
- F
- G
- H
Mark Phase

Diagram showing a tree structure with nodes labeled A, D, B, C, E, F, G, H connected by arrows indicating the flow or relationship between them. The root node is labeled "Root."
Mark Phase
Mark Phase

Diagram:
- **Root**
- **A**
  - **D**
  - **B**
  - **C**
- **E**
- **F**
- **G**
- **H**
Sweep Phase

Root

A

D

B

C

E

F

G

H
Sweep Phase
Mark & Sweep

Very Easy!

Too Slow
"Stop the world"
Visits every object, every time
Walk every object
every time
Generational
Objects die young
Divide objects into "old" and "new"
Generational

Root

A

B

C

D

Gen 0

Gen 1
Generational

Root

Gen 0

Gen 1
Generational

Root

Gen 0

Gen 1
Generational
Generational

Gen 0

Root

Gen 1

B

E

F

G

D
Generational

Gen 0

Gen 1

Root

B

D

F

G
Generational

Root

B
D
F
G

Gen 0

Gen 1
We didn’t have to touch "B"
One *Slight* Problem
Generational

Root

Gen 0

Gen 1

B

D
Remembered Set

Root

Gen 0

E

Gen 1

B

D
Remembered Set

Root → B → D → E

Gen 0
Gen 1

Remember!
Remembered Set

Root

Gen 0

E

Gen 1

B

D
Remembered Set

Gen 0

Root

Gen 1

B

D

E
Remembered Set

Root

Gen 0

Gen 1

B

D

E
Important Words
Write Barrier
Remembered Set
Generational

Fast(er)!

Not so easy
"Stop the world"
Incremental GC
Tri-Color Marking
Object Colors

- **White**: will be collected
- **Black**: No reference to white objects, but reachable from the root
- **Gray**: Reachable from root, but not yet scanned
Algorithm

1. Pick an object from the gray set and move it to the black set
2. For each object it references, move it to the gray set
3. Repeat 1 and 2 until the gray set is empty
Tri-Color Marking

Root

A

D

F

G

B

C

E

H
Tri-Color Marking

![Diagram of Tri-Color Marking]
Tri-Color Marking

Root

A

D

B

C

F

E

G

H
Tri-Color Marking

Diagram:

- Root
- A
- D
- F
- B
- C
- E
- G
- H
Tri-Color Marking

Root

A

D

F

B

C

E

G

H
What is the benefit?
We can interrupt steps!
Each step can be performed incrementally
Halting time is reduced
One *Slight*
Problem
Tri-Color Marking

Root

A

D

F

B

C
Tri-Color Marking

Root

A

B

c

d

C

F

G
Tri-Color Marking
Tri-Color Marking
Important Words
Incremental
Write Barrier
Remembered Set
Minimize tracing
Decrease halting
Things our GC is not
Parallel
Real-Time
Compacting
Allocation Algorithm
Heap layout
malloc isn’t free

GET IT?????
Large Chunk: Page (or Slab)
Page memory is contiguous
Each page holds a linked list
Nodes are called "slots"
Each slot is a Ruby object
Page Layout

Ruby Object → Ruby Object → Ruby Object → Ruby Object → Ruby Object
"Free List"
Find the first open slot!
Move to the next space in the free list
"Bump Pointer" allocation
"Eden" pages are searched
GC Time

calendar

Object

Object

Object

Object
Important Words
Slot
Eden
Tomb
Not every object requires allocation
1 Object: 40 bytes
Pages are "aligned"
Not `malloc` but "aligned malloc"

`posix_memalign`
Choose "40" as a multiple
Start = 40

>> start = 40
=> 40
>> (start * 1).to_s(2).rjust(10, '0')
=> "0000101000"
>> (start * 2).to_s(2).rjust(10, '0')
=> "0001010000"
>> (start * 3).to_s(2).rjust(10, '0')
=> "0001111000"
>> (start * 4).to_s(2).rjust(10, '0')
=> "0010100000"
>> (start * 5).to_s(2).rjust(10, '0')
=> "0011001000"
>> (start * 6).to_s(2).rjust(10, '0')
=> "0011110000"
>> (start * 7).to_s(2).rjust(10, '0')
=> "0100011000"
Use these bits to add meaning
Represent Integers Without Allocation
Encode Number 2

```
>>> INT_FLAG = 0x1
=> 1

>>> 2.to_s(2)
=> "10"

>>> ((2 << 1) | INT_FLAG).to_s(2)
=> "101"
```
Decode Number 2

>> 0b101
=> 5
>> 0b101 >> 1
=> 2
Biggest Fixnum

```ruby
>> ((2 ** 64) - 1).to_s(2)
=>
"11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111"

>> ((2 ** 64) - 1).class
=> Bignum

>> ((2 ** 63) - 1).class
=> Bignum

>> ((2 ** 62) - 1).class
=> Fixnum

>> ((2 ** 62)).class
=> Bignum
```
Biggest Before Heap Allocation

```ruby
>> ((2 ** 64) - 1).to_s(2)
=>
"1111111111111111111111111111111111111111111111111111111111111111"
>> ((2 ** 64) - 1).class
=> Integer
>> ((2 ** 63) - 1).class
=> Integer
>> ((2 ** 62) - 1).class
=> Integer
>> ((2 ** 62)).class
=> Integer
```
Fixnums are singletons

```ruby
>> ((2 ** 62) - 1).object_id
=> 9223372036854775807
>> ((2 ** 62) - 1).object_id
=> 9223372036854775807
>> ((2 ** 62)).object_id
=> 70213216135940
>> ((2 ** 62)).object_id
=> 70213216117840
```
Tagged Pointer
Tagged Pointers

- Fixnum
- Floats
- True / False / Nil
- Symbols
Allocation Problems
Poor Reclamation
Poor Reclamation
Poor Reclamation
CoW problems
1 Ruby Memory Page ≠ 1 OS Memory Page
1 Ruby Page: 16k
1 OS Page: 4k
OS Copies
1 OS Page
We wrote 40 bytes, but 4kb got copied.
Solution:
Group Old Objects
Two Page Types

Probably Old Page

Page

Object
What is going to be old?
Probably Old

class Foo
end

module Bar
  CONSTANT = Object.new

  def foo
    "frozen string".freeze
  end
end
Statistically Determined
Efficiently use space
Reduce GC time
CoW Friendly
GC Work

GitHub
Key

- Red: Objects
- Green: Class / Module
- White: Empty
Not Very Good MRI Heap Visualizer

Heap Dump JSON:
Choose File  out_regular.txt

Submit
Not Very Good MRI Heap Visualizer

Heap Dump JSON:
Choose File: out_regular_split.txt
Not Very Good MRI Heap Visualizer

Heap Dump JSON:

Choose File: out_regular.txt

Submit
Not Very Good MRI Heap Visualizer

Heap Dump JSON:

Choose File: `out-split.txt`
Not Very Good MRI Heap Visualizer

Heap Dump JSON:

Choose File

out.txt

SUBMIT
~17%
Smaller Heap
github.com/
github/ruby
Future Work
Moving Objects
Poor Reclamation

Object

Object

Object

Object

Object
Stack Scanning,
Forward Pointers
GC
Introspection
Seeing GC info
GC.stat

{:
:count=>21,
:heap_allocated_pages=>87,
:heap_sorted_length=>87,
:heap_allocatable_pages=>0,
:heap_available_slots=>35460,
:heap_live_slots=>35342,
:heap_free_slots=>118,
:heap_final_slots=>0,
:heap_marked_slots=>22207,
:heap_eden_pages=>87,
:heap_tomb_pages=>0,
:total_allocated_pages=>87,
:total_freed_pages=>0,
:total_allocated_objects=>208577,
:total_freed_objects=>173235,
:malloc_increase_bytes=>5152,
:malloc_increase_bytes_limit=>16777216,
:minor_gc_count=>19,
:minor_gc_count=>2,
:malloc_increase_bytes_limit=>16777216}
GC Performance
GC::Profiler

>> GC::Profiler.enable
=> nil
>> GC::Profiler.report
=> nil
>> GC.start
=> nil
>> GC::Profiler.report
GC 22 invokes.

<table>
<thead>
<tr>
<th>Index</th>
<th>Invoke Time(sec)</th>
<th>Use Size(byte)</th>
<th>Total Size(byte)</th>
<th>Total Object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>906920</td>
<td>1419840</td>
<td>35496</td>
</tr>
<tr>
<td>1</td>
<td>0.143</td>
<td>906920</td>
<td>1419840</td>
<td>35496</td>
</tr>
</tbody>
</table>

3.7250000000000586198
=> nil
>> GC.start
=> nil
>> GC::Profiler.report
GC 23 invokes.

<table>
<thead>
<tr>
<th>Index</th>
<th>Invoke Time(sec)</th>
<th>Use Size(byte)</th>
<th>Total Size(byte)</th>
<th>Total Object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>906920</td>
<td>1419840</td>
<td>35496</td>
</tr>
<tr>
<td>1</td>
<td>0.143</td>
<td>906920</td>
<td>1419840</td>
<td>35496</td>
</tr>
</tbody>
</table>

3.7250000000000586198
2       | 0.148            | 906920         | 1419840          | 35496        |

3.4780000000000864020
GC::Profiler.enable
Heap Inspection
ObjectSpace.dump_all
>> require 'objspace'
=> false
>> x = Object.new
=> #<Object:0x007fbcd09334a8>
>> ObjectSpace.dump x
=> "{"address":"0x007fbcd09334a8","type":"OBJECT","class":"0x007fbcd08dd878","ivars":0,"memsize":40,"flags":{"wb_protected":true}}\n"
ObjectSpace.dump

```ruby
>> x = Object.new
=> #<Object:0x007fbcd0959248>
>> JSON.parse(ObjectSpace.dump(x))['flags']
=> {"wb_protected"=>true}
>> GC.start
=> nil
>> JSON.parse(ObjectSpace.dump(x))['flags']
=> {"wb_protected"=>true}
>> GC.start
=> nil
>> JSON.parse(ObjectSpace.dump(x))['flags']
=> {"wb_protected"=>true}
>> GC.start
=> nil
>> JSON.parse(ObjectSpace.dump(x))['flags']
=> {"wb_protected"=>true, "old"=>true, "uncollectible"=>true, "marked"=>true}
```
Object have 3 generations
ObjectSpace
trace_object_allocations
GC Tuning
RUBY_GC_HEAP_FREE_SLOTS

Number of free slots available after a GC
RUBY_GC_HEAP_INIT_SLOTS

Number of free slots to initialize the GC with
RUBY_GC_HEAP_GROWTH_MAX_SLOTS

Never grow more than this many objects
THANKS!