# Building the Next-Generation Garbage Collector in Ruby


## Peter Zhu

**Ruby Core Committer**
**Staff Software Engineer, Figma**

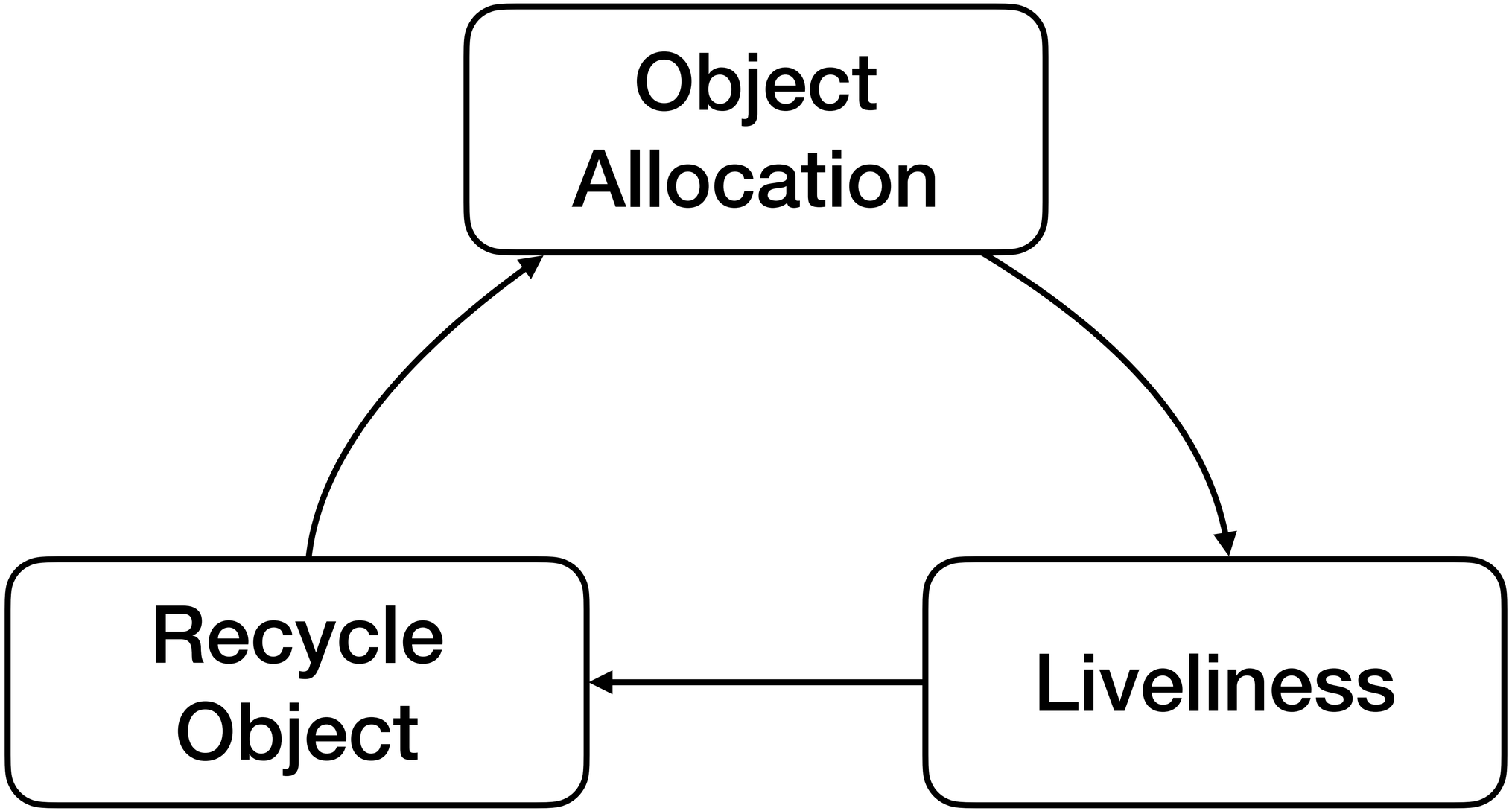
blog.peterzhu.ca/assets/rubykaigi_2026_slides.pdf

# Peter Zhu

- Based in Toronto, Canada
- Ruby Core Committer
- Staff Software Engineer at Figma
- Co-author of Variable Width Allocation, `RUBY_FREE_AT_EXIT`, and Modular Garbage Collectors in Ruby
- Author of `ruby_memcheck` and `autotuner`

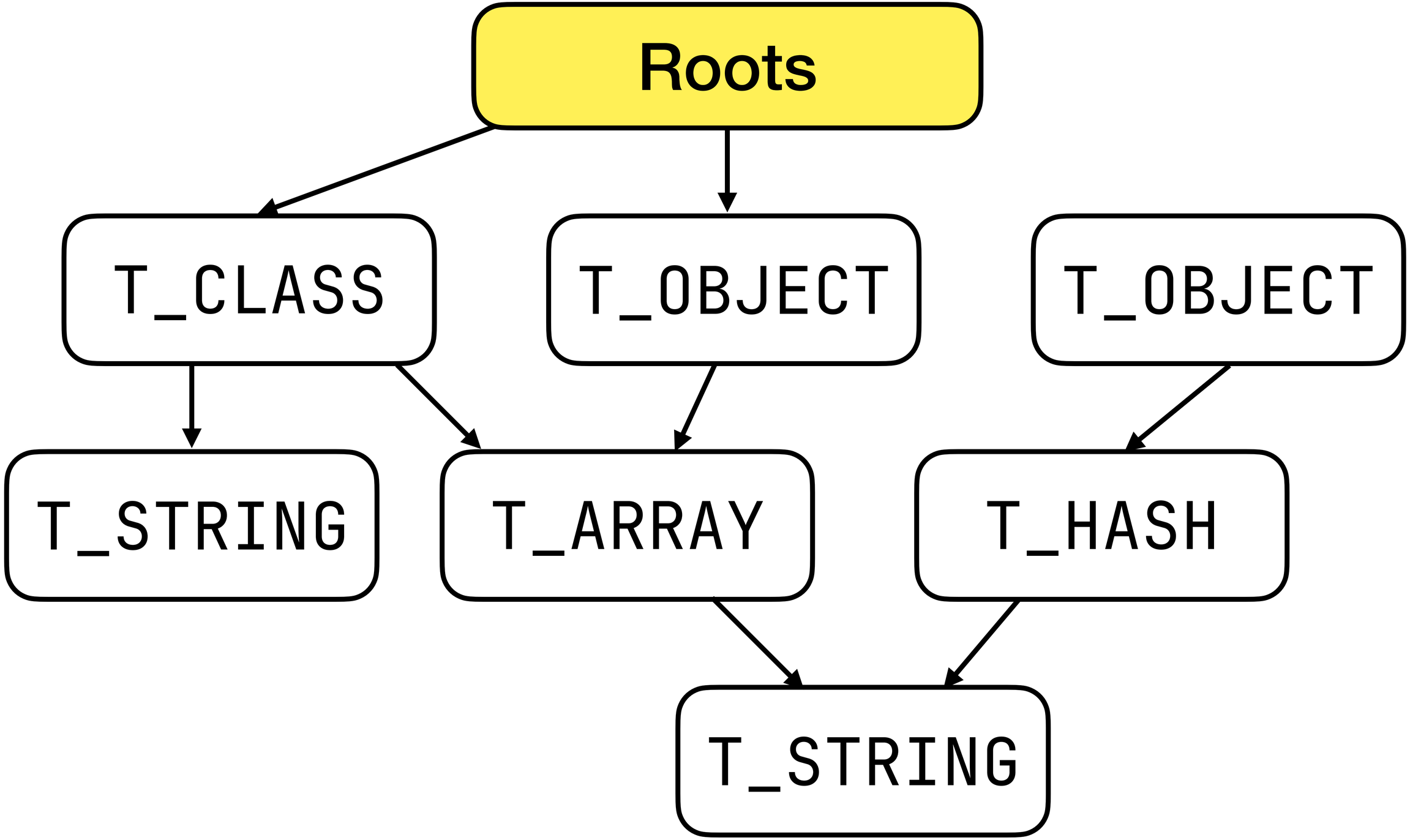
- Introduce Ruby's Garbage Collector
- Modular Garbage Collectors
- Memory Management Toolkit
  - Improvements since RubyKaigi 2025
    - Moving Immix
    - Bump Pointer Allocator Fast Path
    - Ruby Heap
  - Future Roadmap

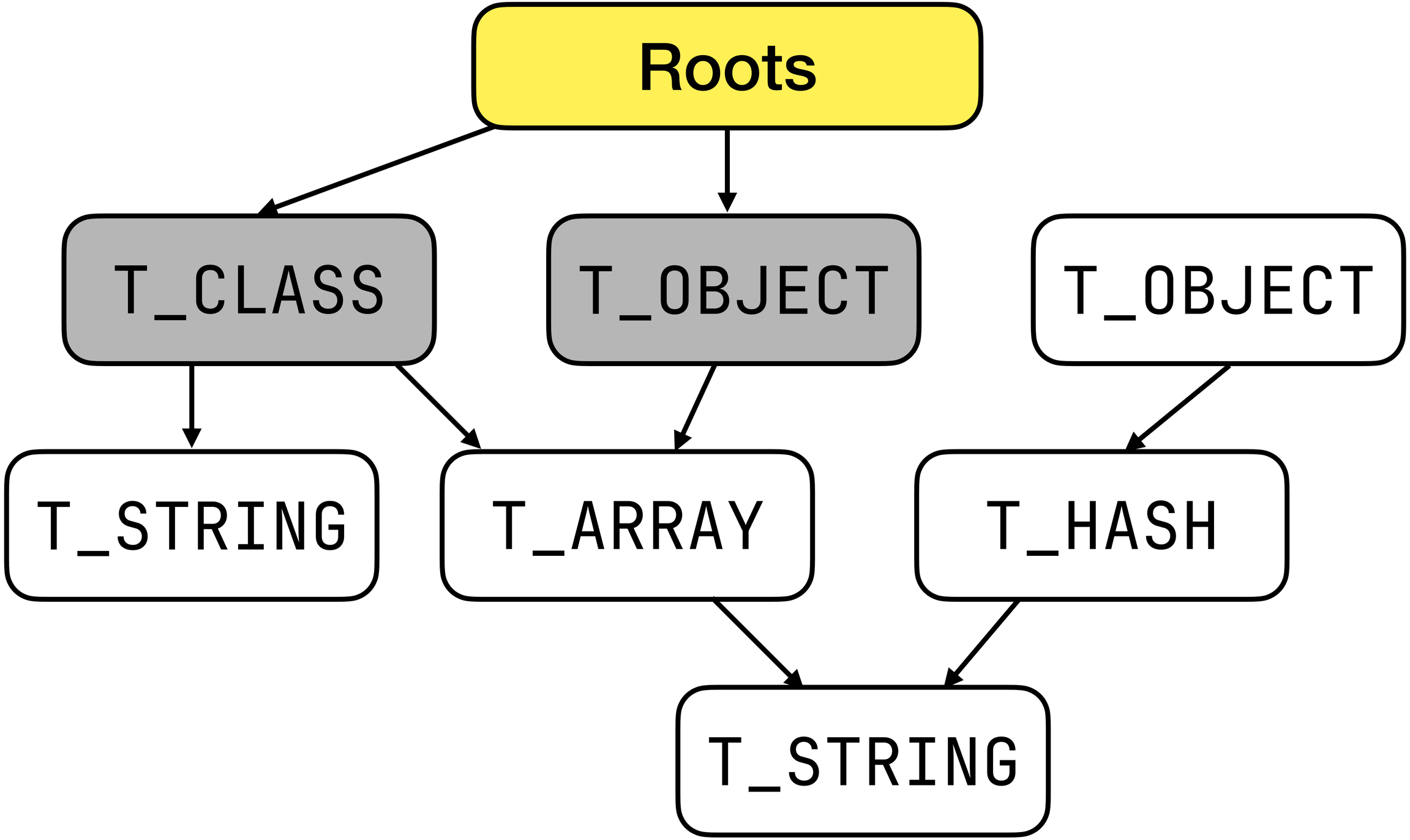
# What is a Garbage Collector?

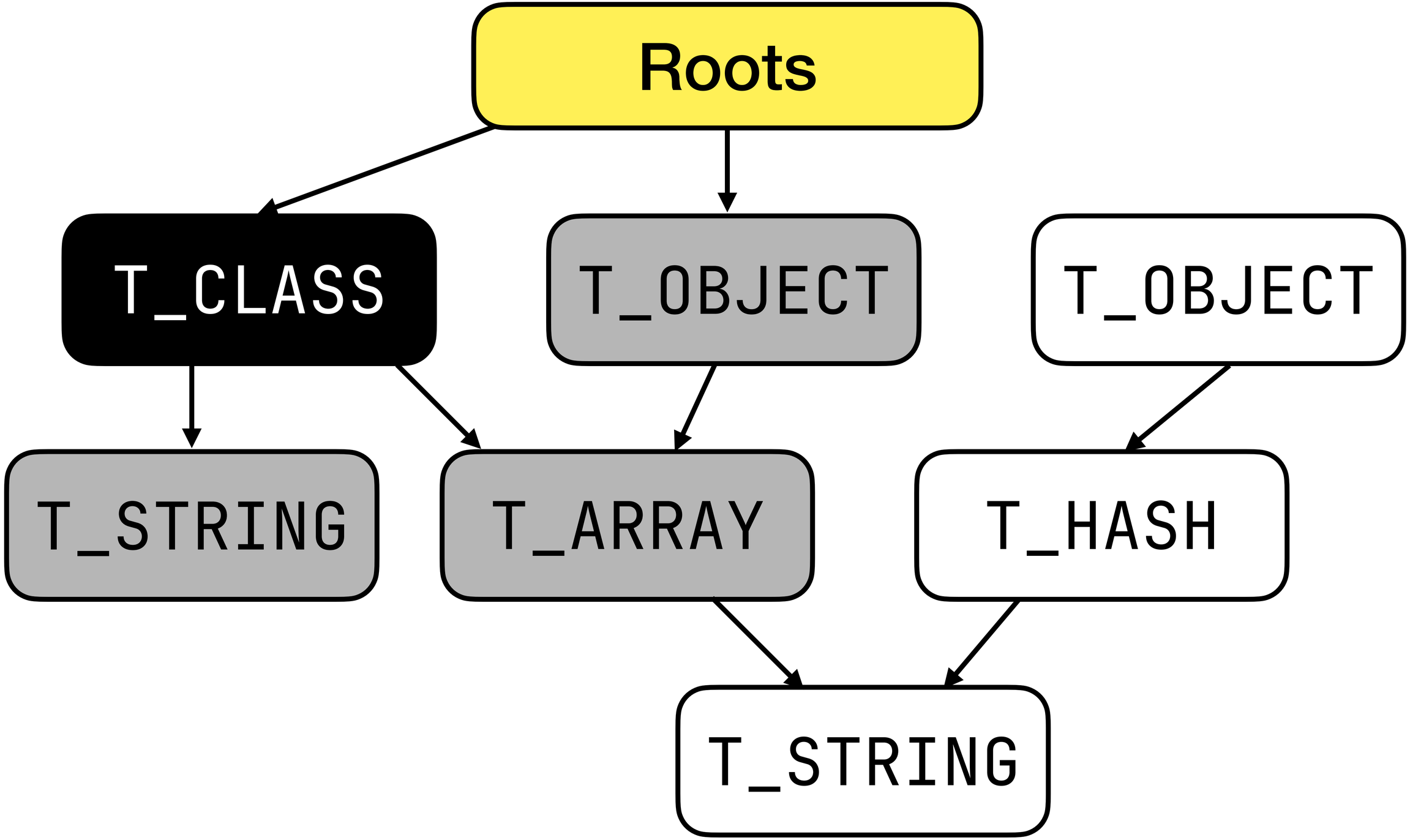

Object
Allocation
Liveliness
Recycle
Object

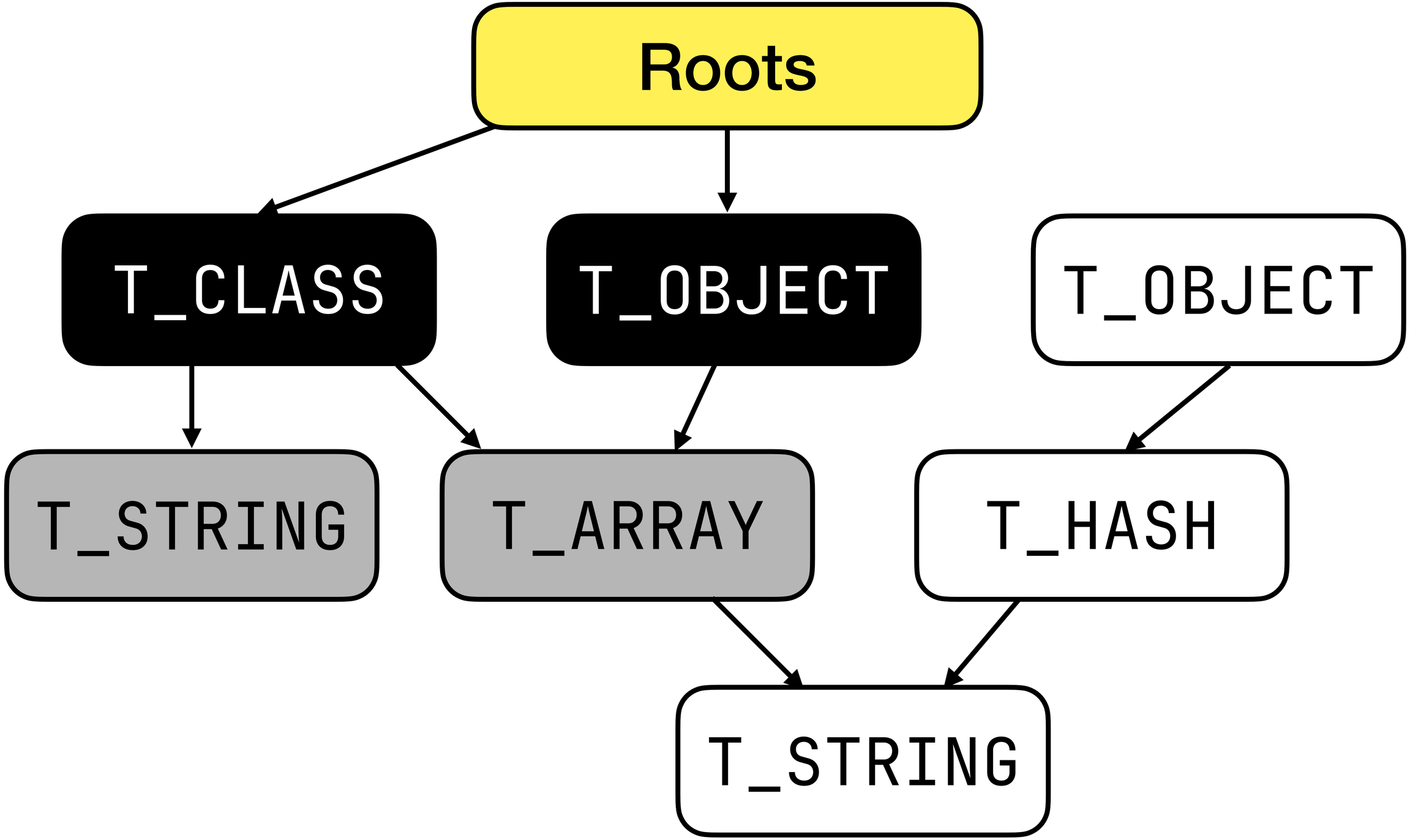
# Mark-Sweep-Compact

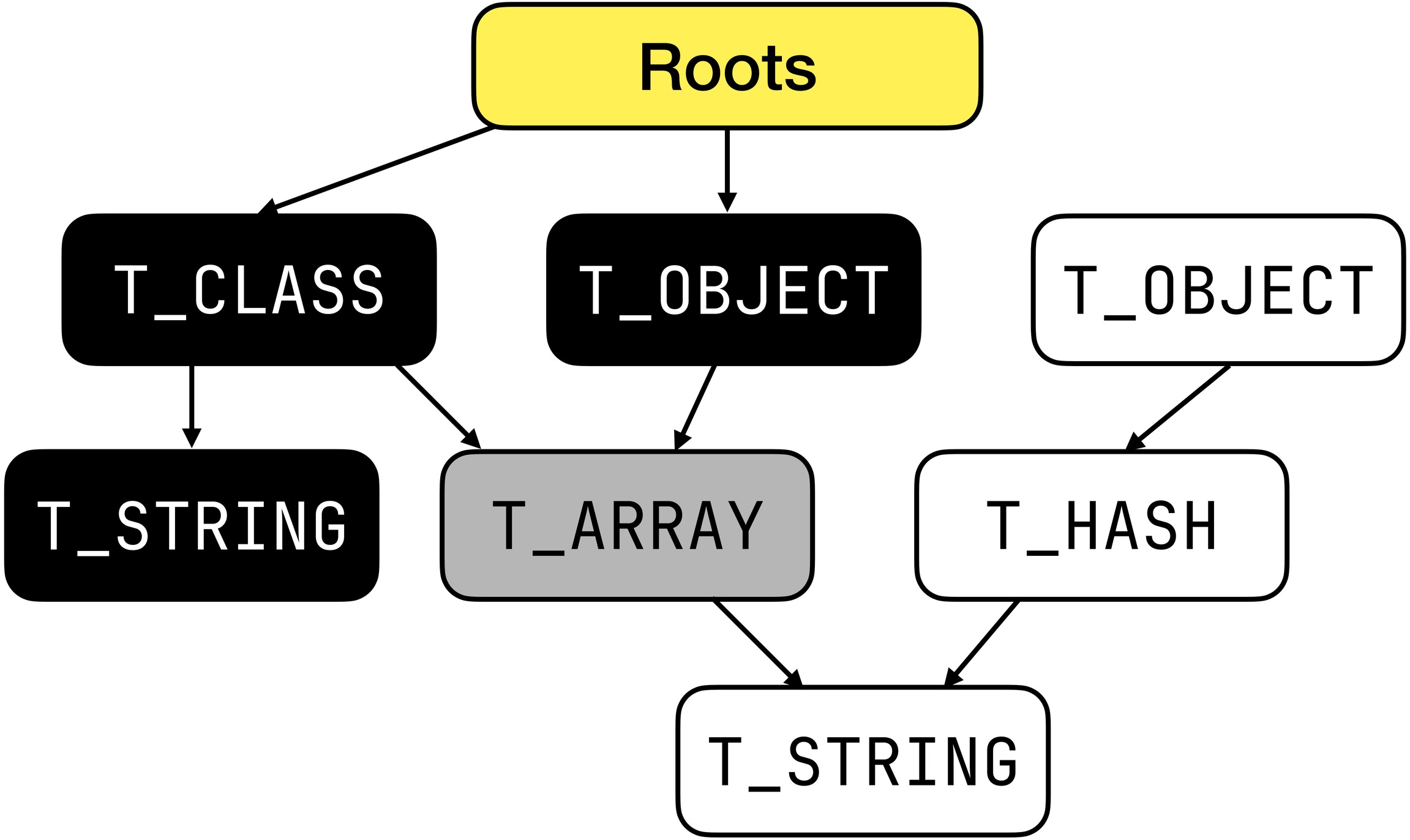
# Mark Phase

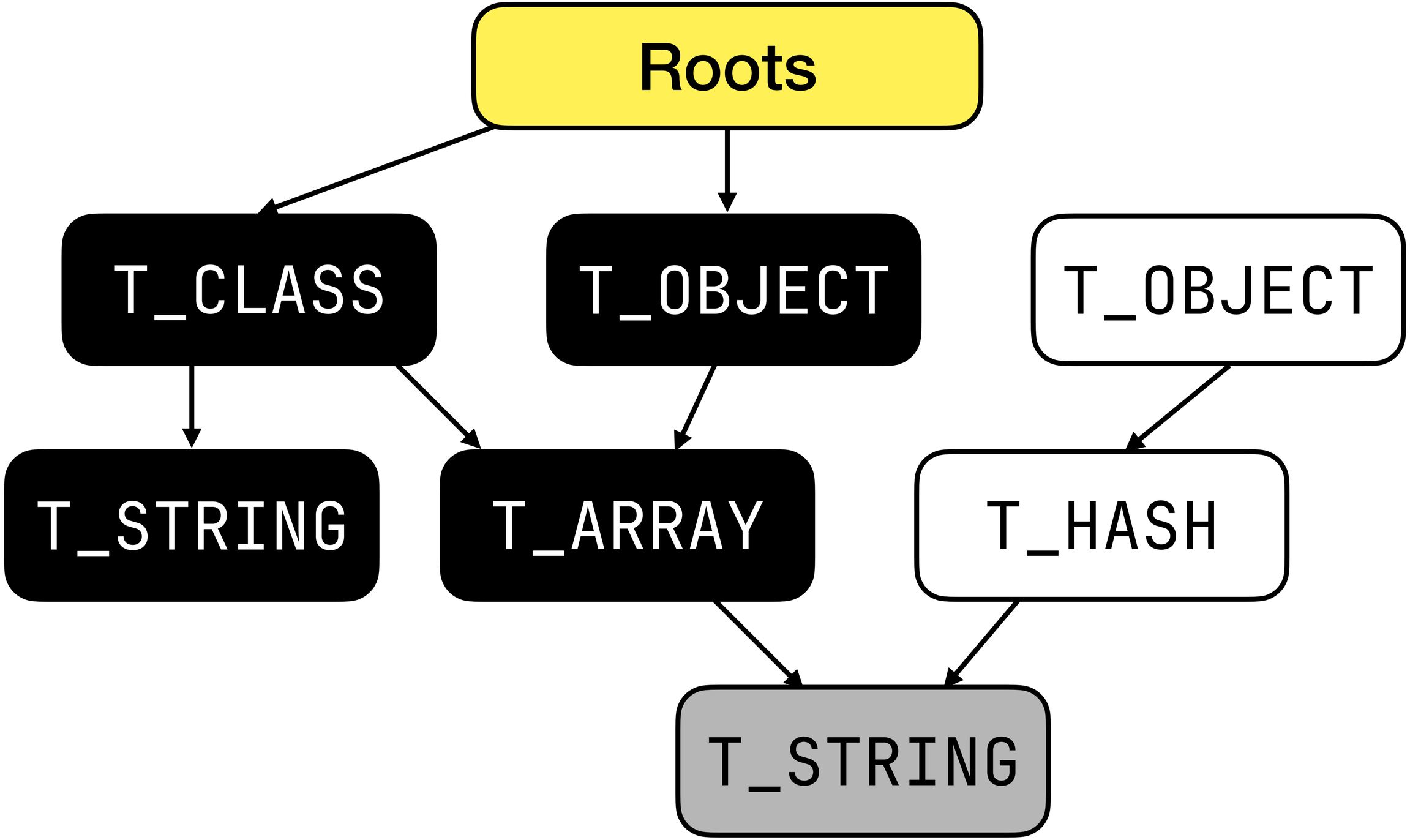

Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING

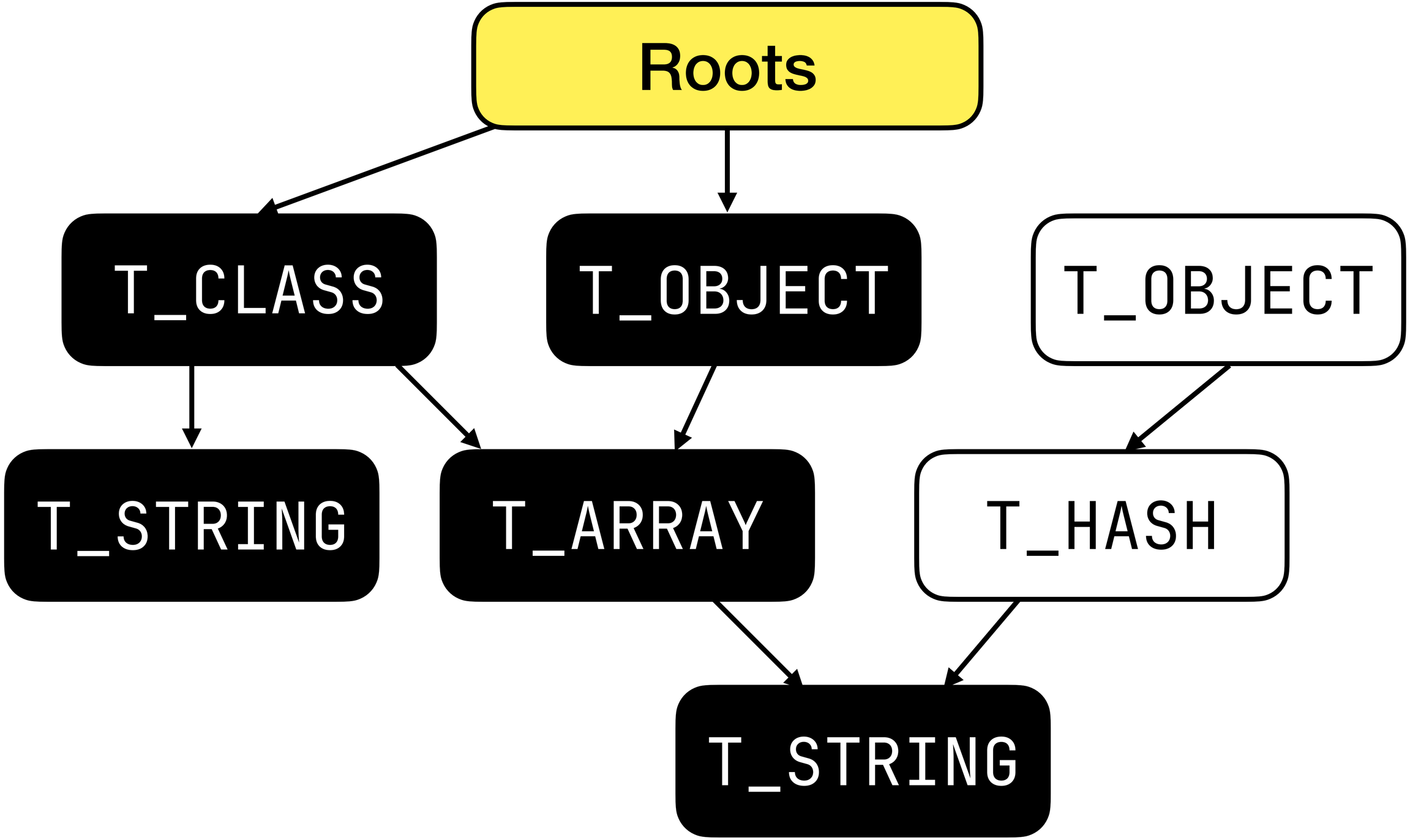

Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING


Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING

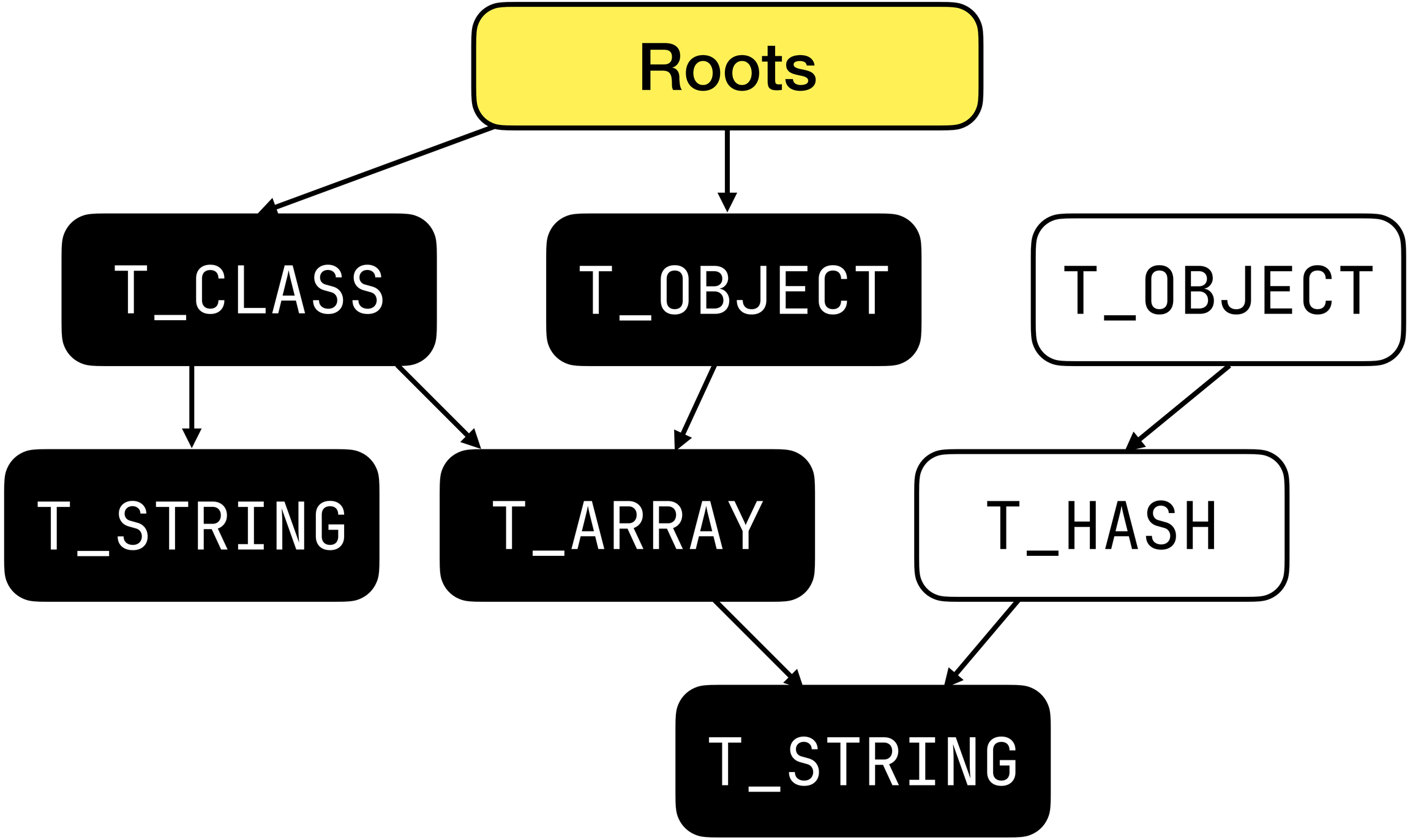

Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING

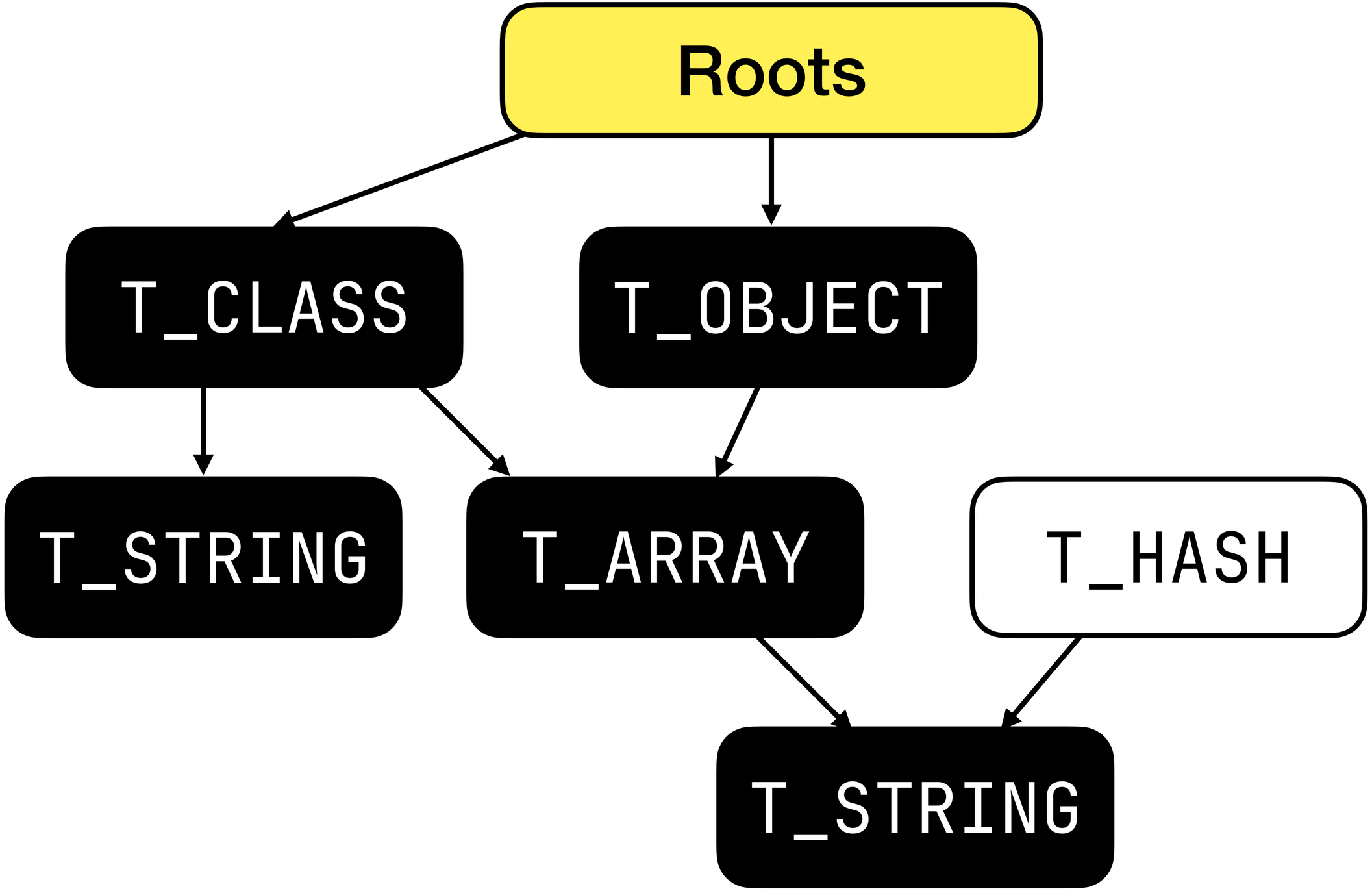

Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING

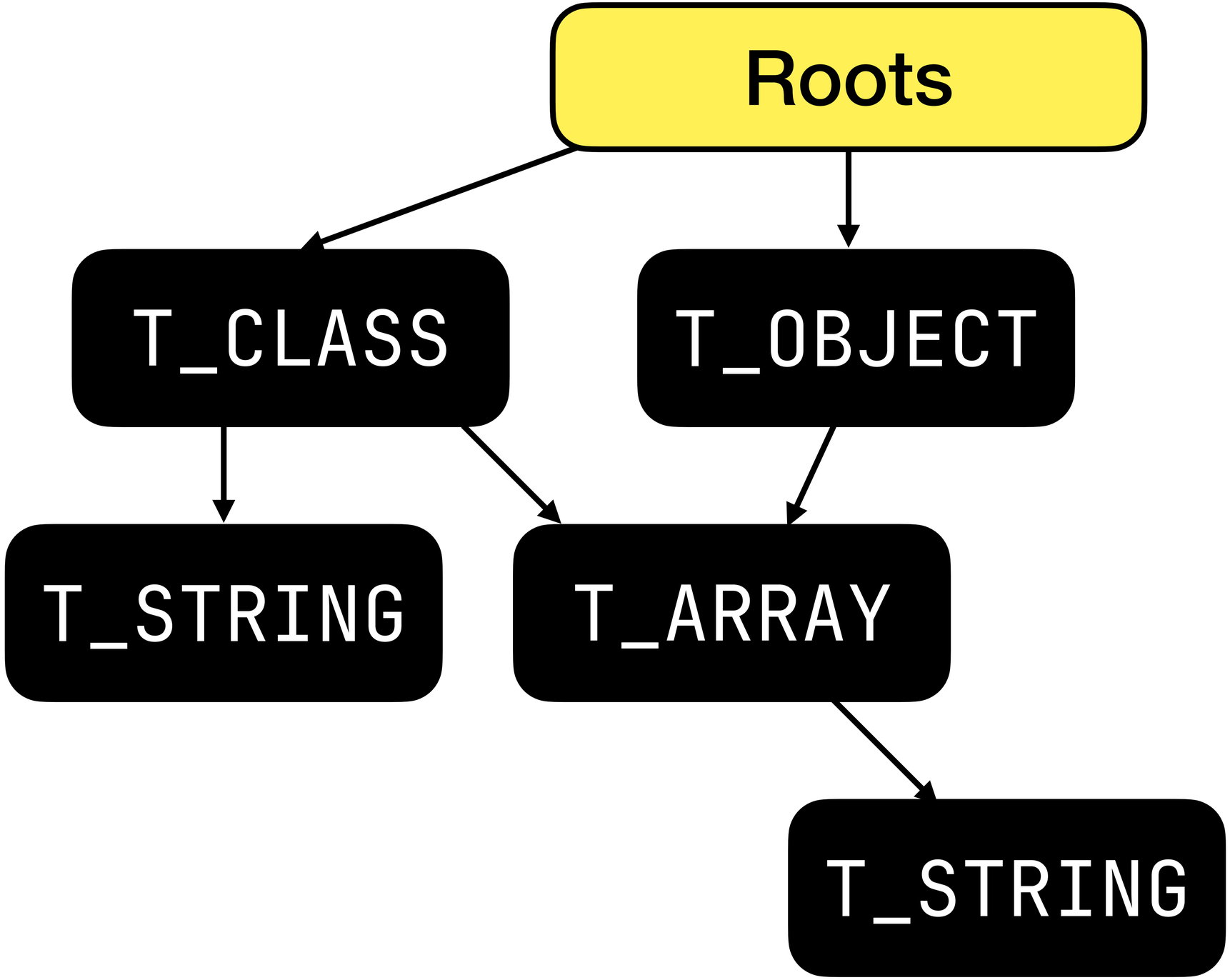

Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING


Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING

# Sweep Phase


Roots
T_CLASS
T_OBJECT
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING


Roots
T_CLASS
T_OBJECT
T_STRING
T_ARRAY
T_HASH
T_STRING


Roots
T_CLASS
T_OBJECT
T_STRING
T_ARRAY
T_STRING

# Compaction Phase


T_OBJECT
T_STRING
T_HASH


T_OBJECT
T_STRING
T_HASH


T_OBJECT
T_STRING
T_HASH

# Garbage Collection in Ruby

Jul 03, 2023 (updated at Oct 02, 2023)

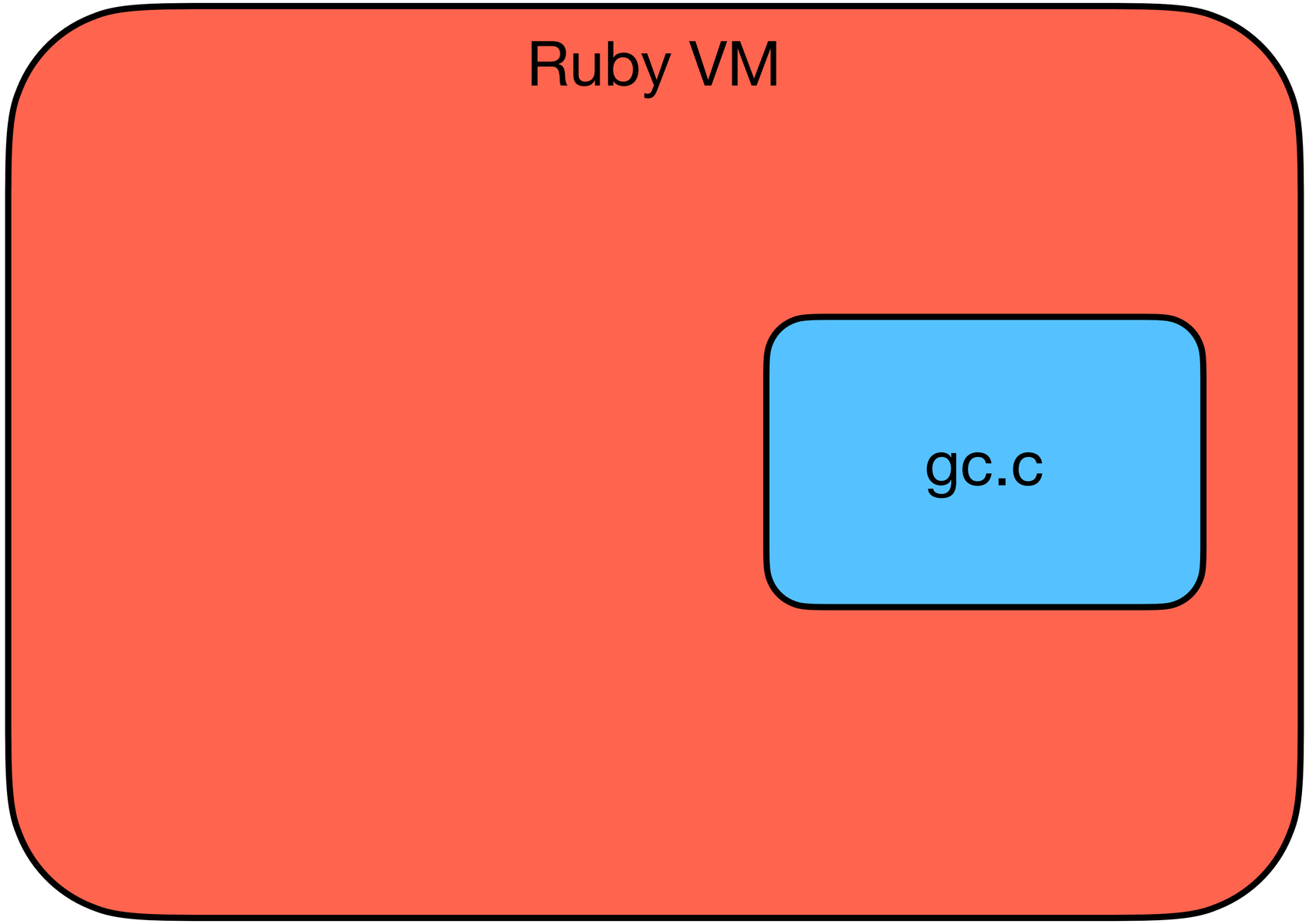
## 1. Overview

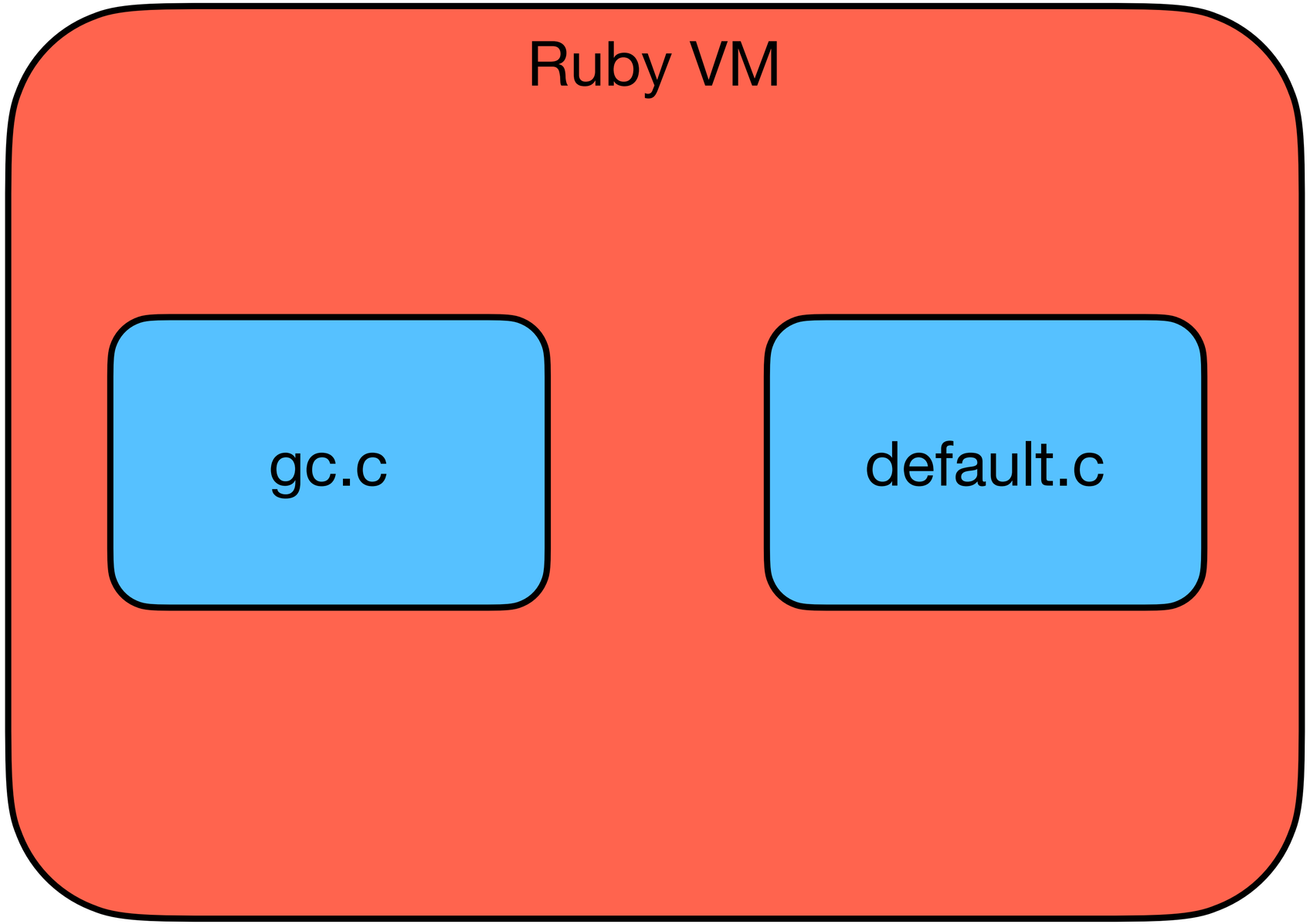
Ruby's garbage collector code lives in a single file called `gc.c`. "Garbage collector" is probably not the best term to call it because in addition to garbage collection, the code inside `gc.c` is responsible for memory allocation and management. In other words, the whole lifecycle of an object is managed by the garbage collector.

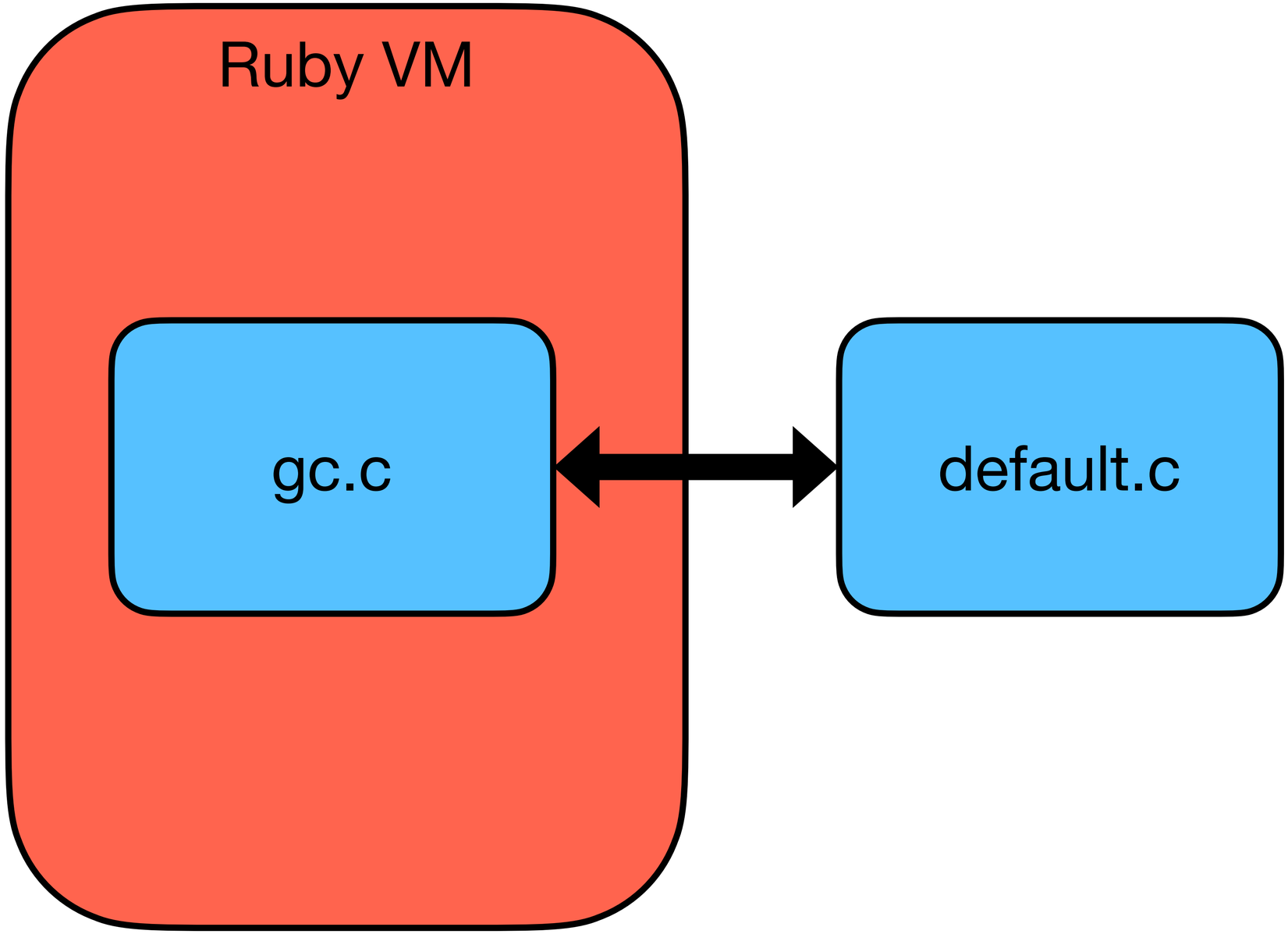
## 2. Primitive object types

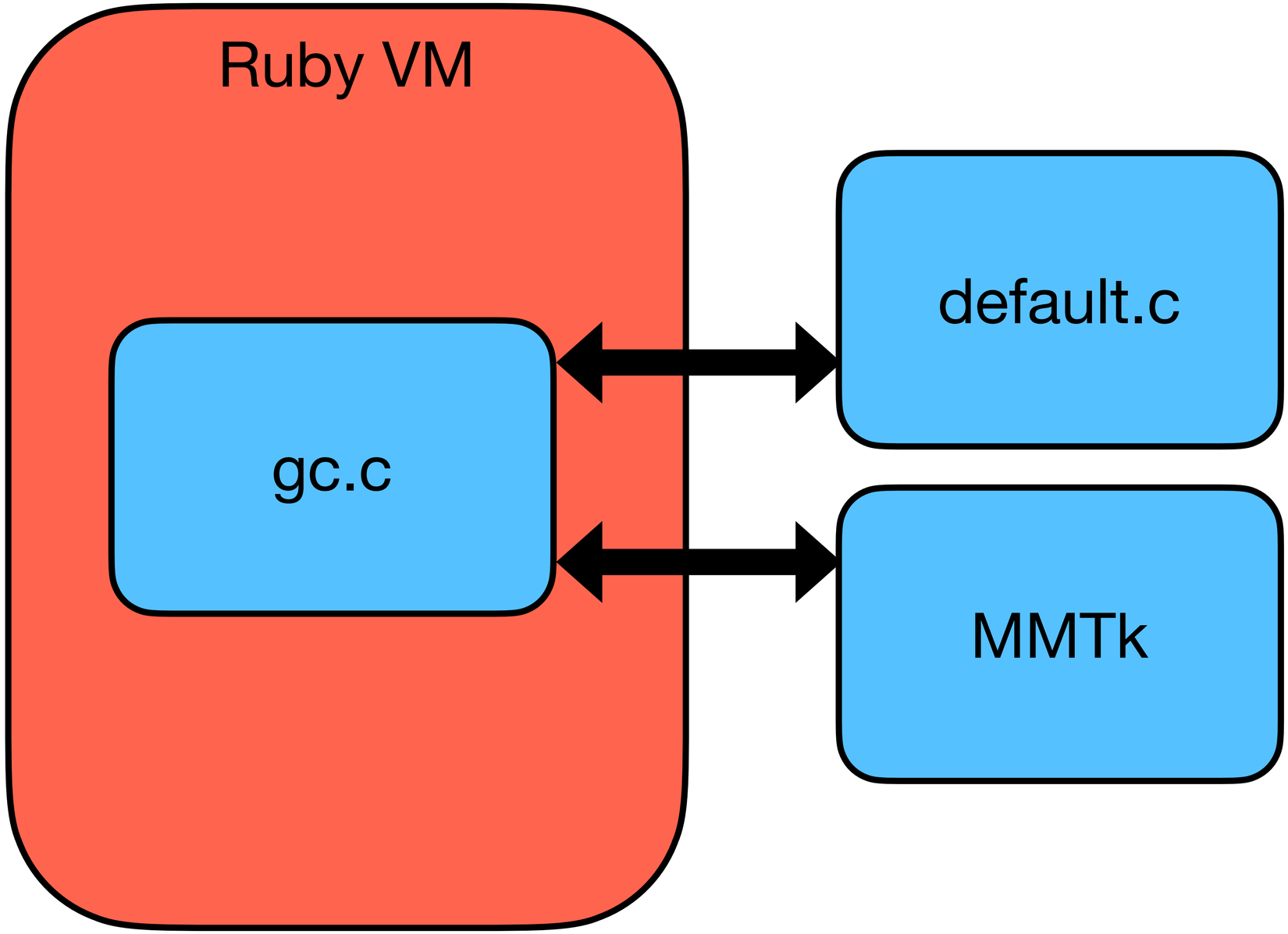
While every type of Ruby object may appear to work the same in the Ruby source code, there

# Modular Garbage Collectors


Ruby VM
gc.c


Ruby VM
gc.c
default.c


Ruby VM
gc.c
default.c


Ruby VM
gc.c
default.c
MMTk

youtu.be/04axm4JcaT4

# Modular Garbage Collectors in Ruby

Peter Zhu

Ruby Core Committer
Senior Developer, Shopify

RubyKaigi 2025

Sponsored by TWOGATE

# Memory Management Toolkit

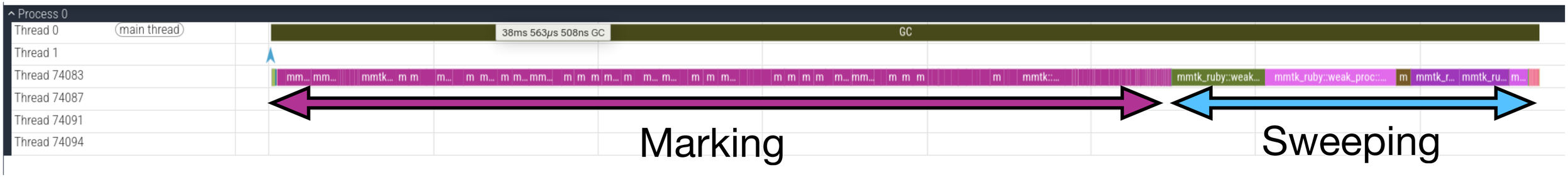
# Memory Management Toolkit

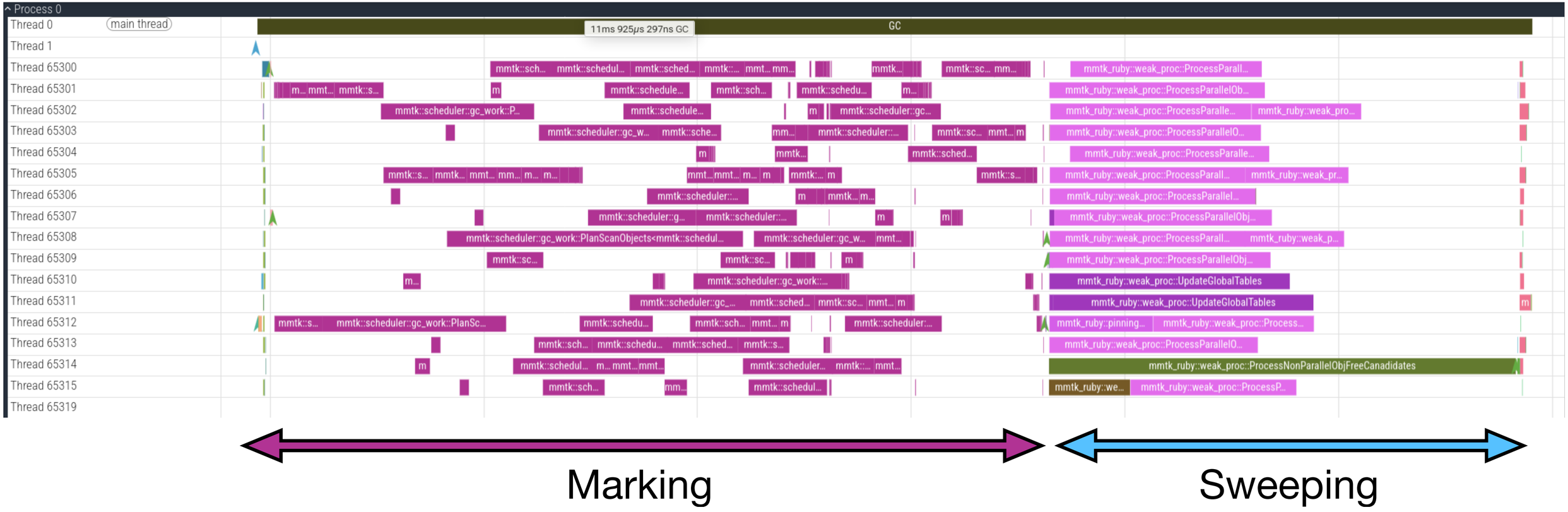
Alternative GC Algorithms

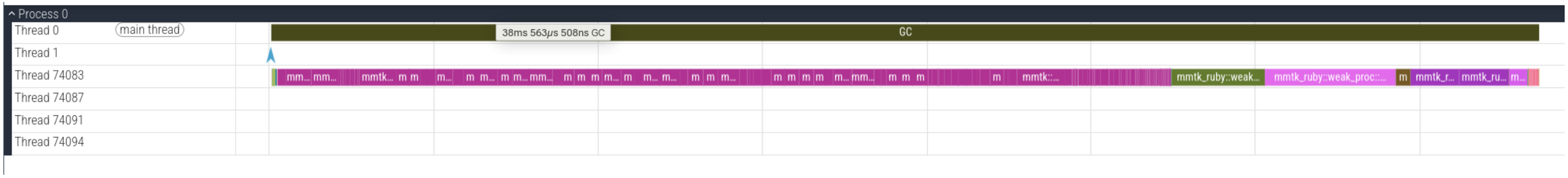
# Memory Management Toolkit

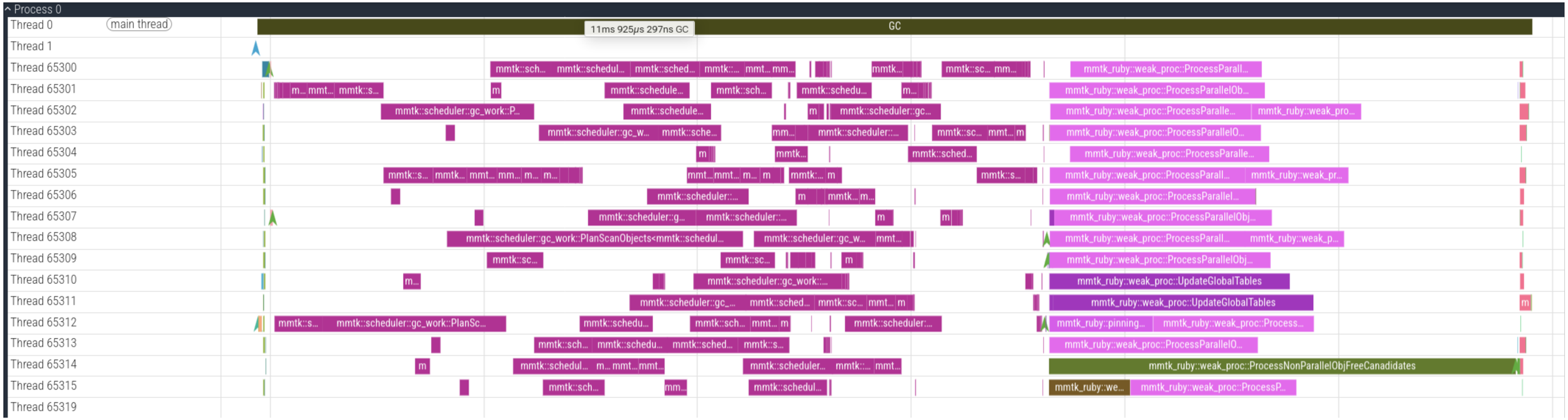
Alternative GC Algorithms

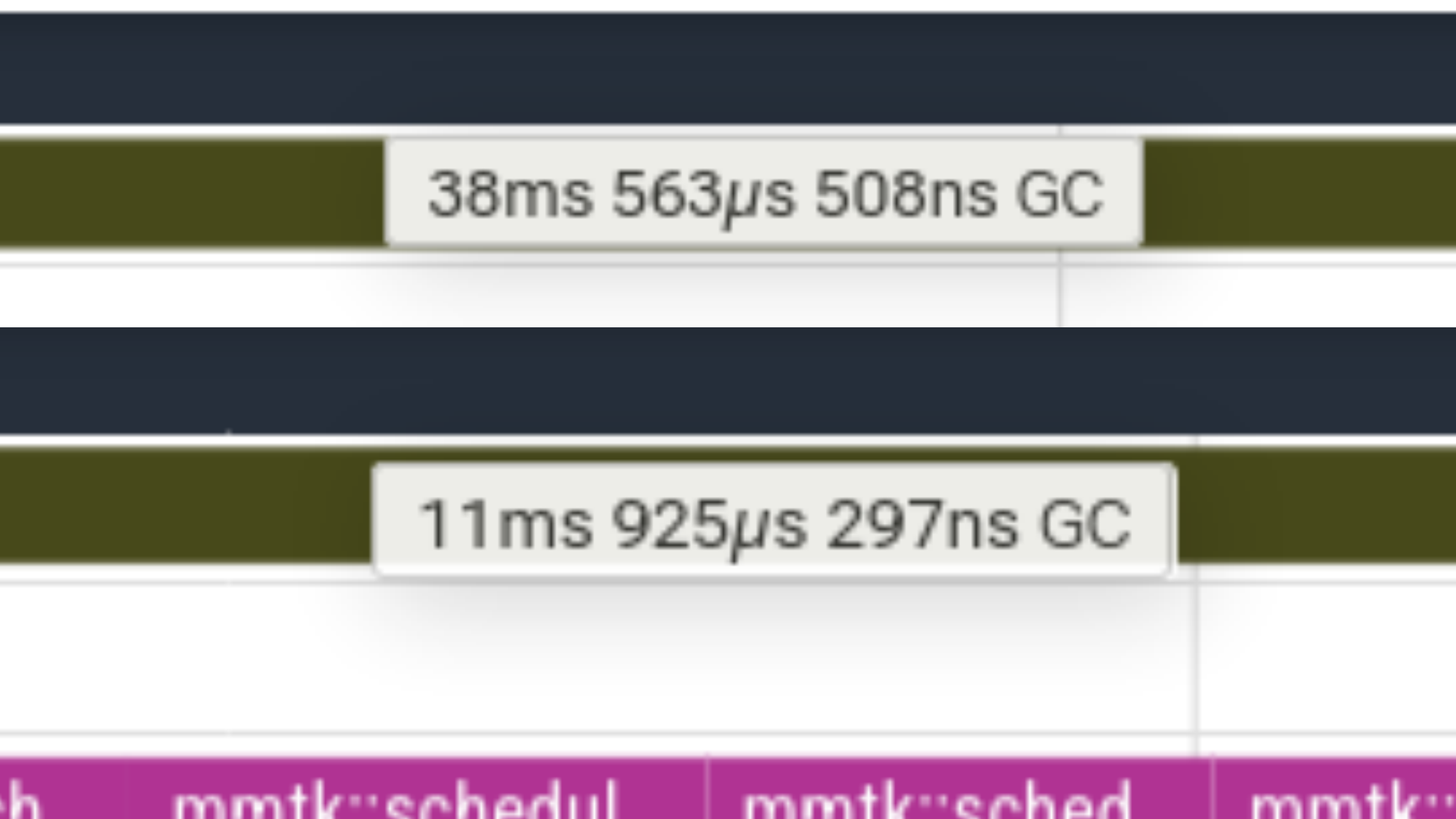
Parallelism


Process 0
Thread 0
main thread
38ms 563μs 508ns GC
GC
Thread 1
Thread 74083
Thread 74087
Thread 74091
Thread 74094
mmtk_ruby::weak...
mmtk_ruby::weak_proc::...
mmtk_r...
mmtk_ru...
mmtk::...
Marking
Sweeping


Process 0
Thread 0
main thread
11ms 925μs 297ns GC
GC
Thread 1
Thread 65300
Thread 65301
Thread 65302
Thread 65303
Thread 65304
Thread 65305
Thread 65306
Thread 65307
Thread 65308
Thread 65309
Thread 65310
Thread 65311
Thread 65312
Thread 65313
Thread 65314
Thread 65315
Thread 65319
mmtk_ruby::weak_proc::ProcessNonParallelObjFreeCanadidates
mmtk_ruby::weak_proc::UpdateGlobalTables
Marking
Sweeping


Process 0
Thread 0
main thread
38ms 563μs 508ns GC
GC
Thread 1
Thread 74083
Thread 74087
Thread 74091
Thread 74094


Process 0
Thread 0
main thread
11ms 925μs 297ns GC
GC
Thread 1
Thread 65300
Thread 65301
Thread 65302
Thread 65303
Thread 65304
Thread 65305
Thread 65306
Thread 65307
Thread 65308
Thread 65309
Thread 65310
Thread 65311
Thread 65312
Thread 65313
Thread 65314
Thread 65315
Thread 65319
mmtk_ruby::weak_proc::ProcessNonParallelObjFreeCanadidates


38ms 563µs 508ns GC
11ms 925µs 297ns GC

# Memory Management Toolkit

Alternative GC Algorithms

Parallelism

# Memory Management Toolkit

Alternative GC Algorithms

Parallelism

Dynamic Object Sizes

shopify.engineering/ruby-variable-width-allocation



BLOG | DEVELOPMENT

# Optimizing Ruby's Memory Layout: Variable Width Allocation

Shopify is improving CRuby's performance in Ruby 3.2 by optimizing the memory layout in the garbage collector through the Variable Width Allocation project.

# Memory Management Toolkit

Alternative GC Algorithms

Parallelism

Dynamic Object Sizes

# Improvements to MMTk Since RubyKaigi 2025

# Moving Immix

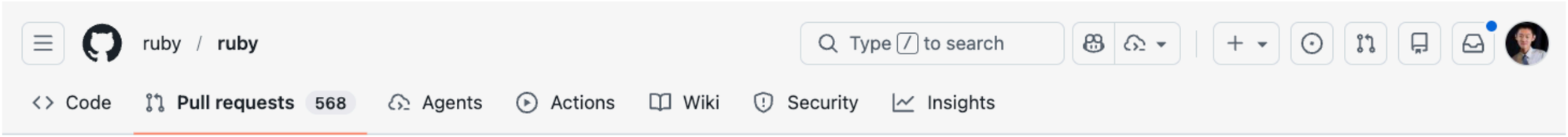
# Immix

# Immix Combines Mark and Compact

# Immix Performs Selective Compaction

# Moving Immix

# github.com/ruby/ruby/pull/15744




## Implement moving Immix in MMTk #15744

Code

Merged peterzhu2118 merged 4 commits into ruby:master from peterzhu2118:register-pinning-obj on Dec 29, 2025

Conversation 2 | Commits 4 | Checks 91 | Files changed 25 | +470 -71

**peterzhu2118** commented on Dec 26, 2025 — Member

This commit implements moving Immix in MMTk, which allows objects to move in the GC.

The performance of this implementation is not yet amazing. It is very similar to non-moving Immix in many of them and slightly slower in others. The benchmark results is shown below.

| bench | Moving Immix (ms) | stddev (%) | RSS (MiB) |
|---|---|---|---|
| activerecord | 241.9 | 0.5 | 86.6 |
| chunky-png | 447.8 | 0.8 | 74.9 |
| erubi-rails | 1183.9 | 0.8 | 136.1 |
| hexapdf | 1607.9 | 2.6 | 402.3 |
| liquid-c | 45.4 | 6.7 | 44.9 |
| liquid-compile | 44.1 | 9.3 | 53.0 |
| liquid-render | 105.4 | 4.5 | 55.9 |
| lobsters | 650.1 | 9.7 | 418.4 |
| mail | 115.4 | 2.1 | 64.4 |
| psych-load | 1656.8 | 0.8 | 43.6 |
| railsbench | 1653.5 | 1.3 | 149.8 |
| rubocop | 127.0 | 15.6 | 142.1 |

**Reviewers**
No reviews

**Assignees**
No one assigned

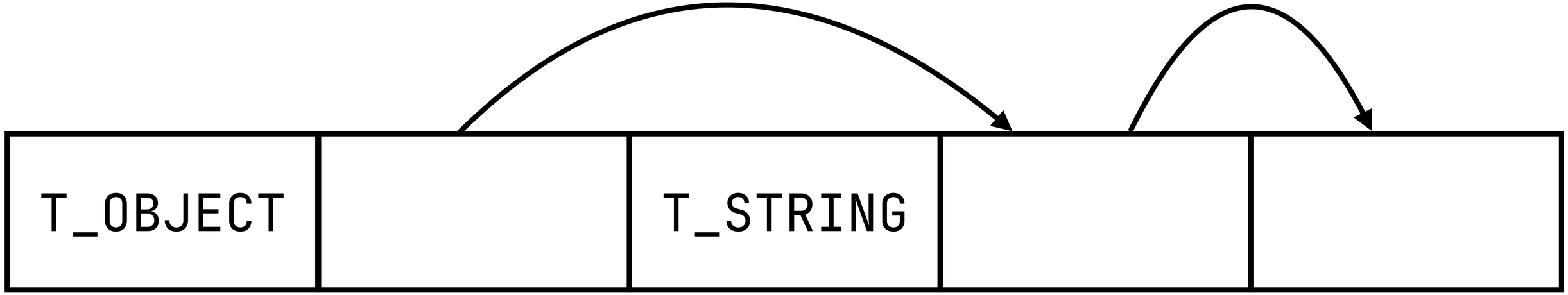
**Labels**
None yet

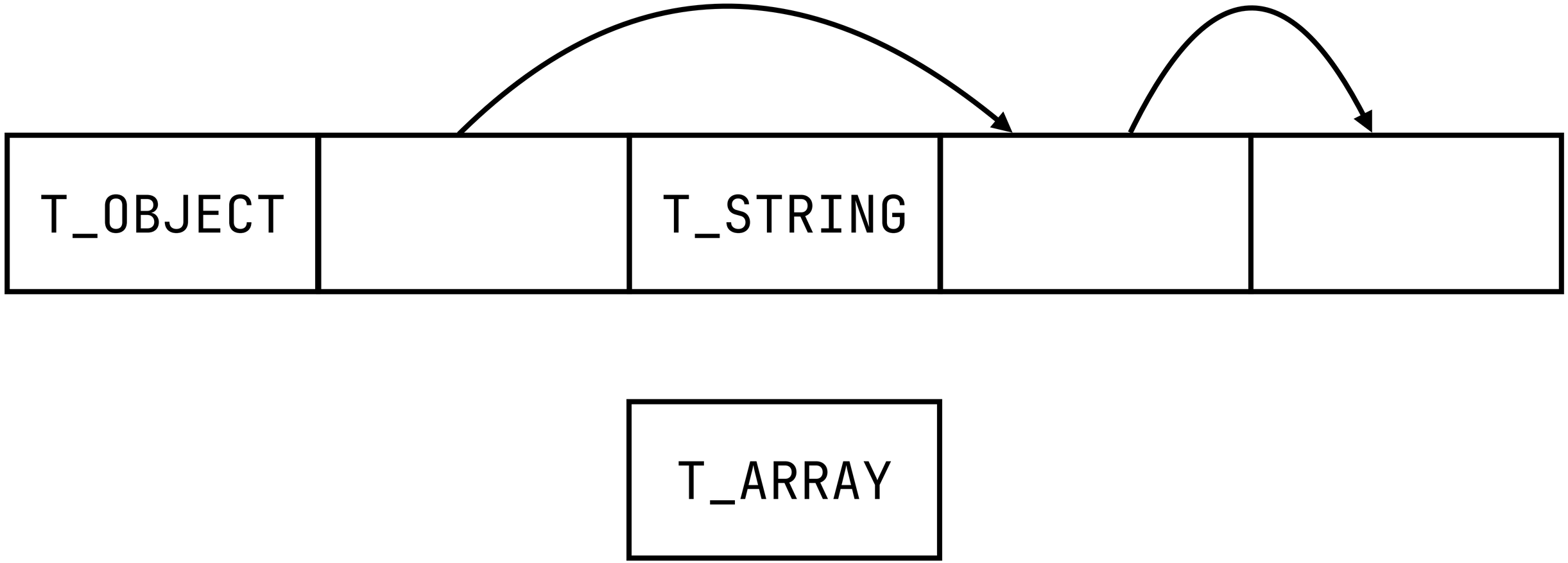
**Projects**
None yet

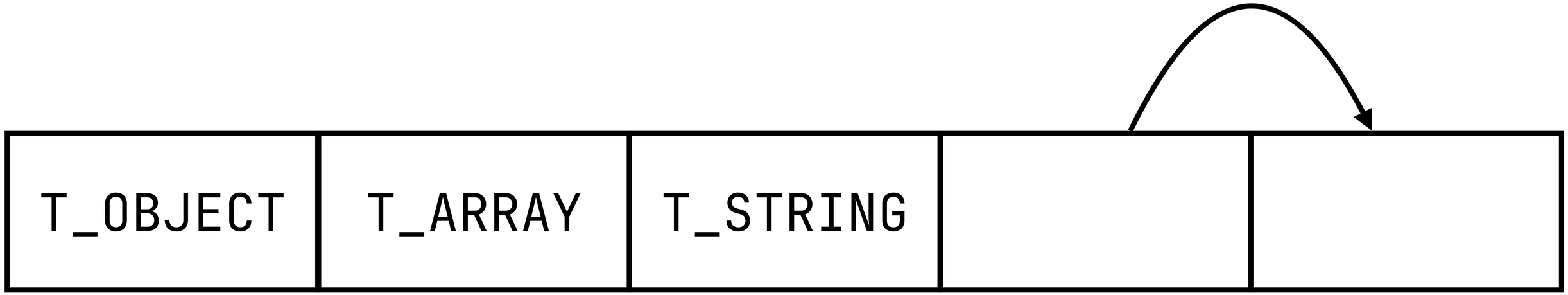
**Milestone**
No milestone

**Development**
Successfully merging this pull request may close these issues.

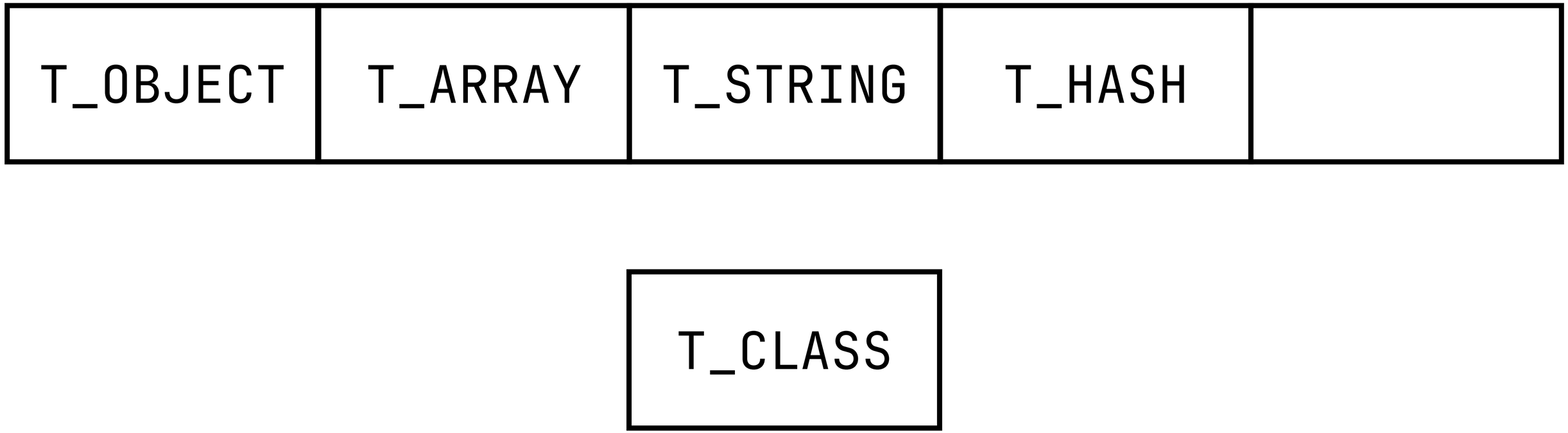
# Bump Pointer
# Fast Path Allocator

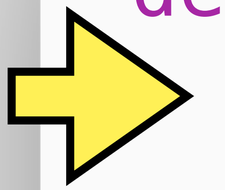

T_OBJECT
T_STRING


T_OBJECT
T_STRING
T_ARRAY

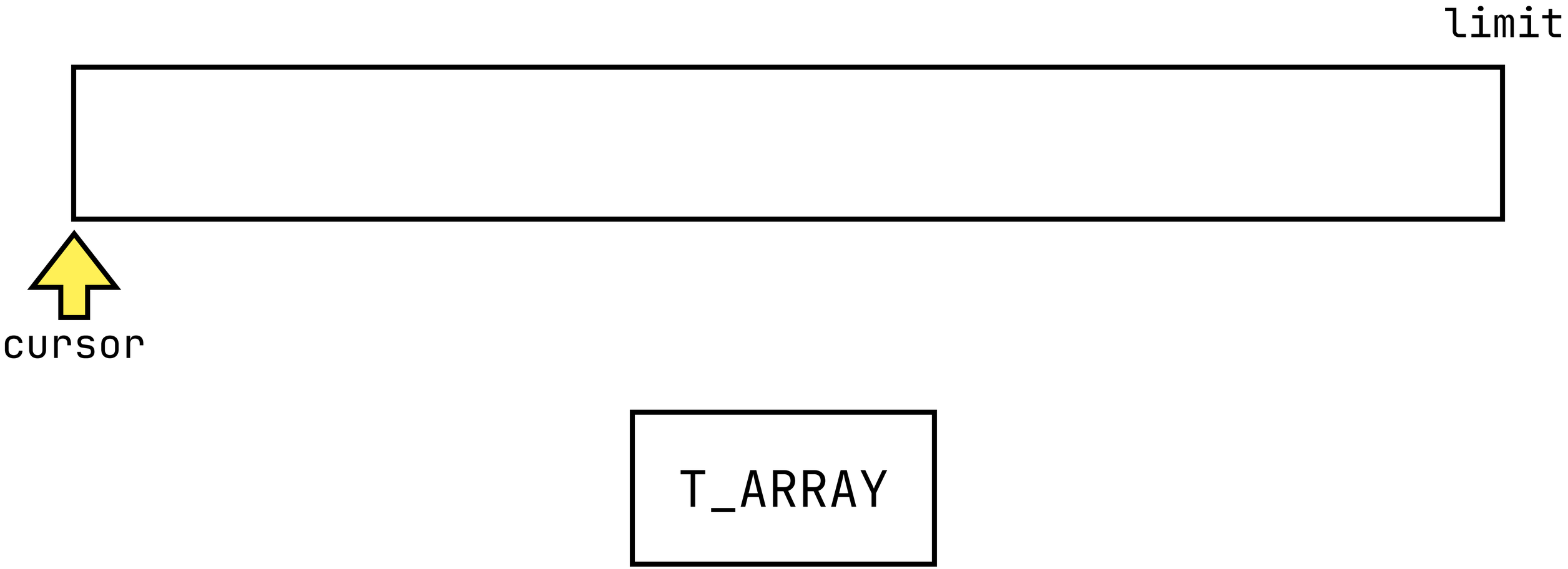


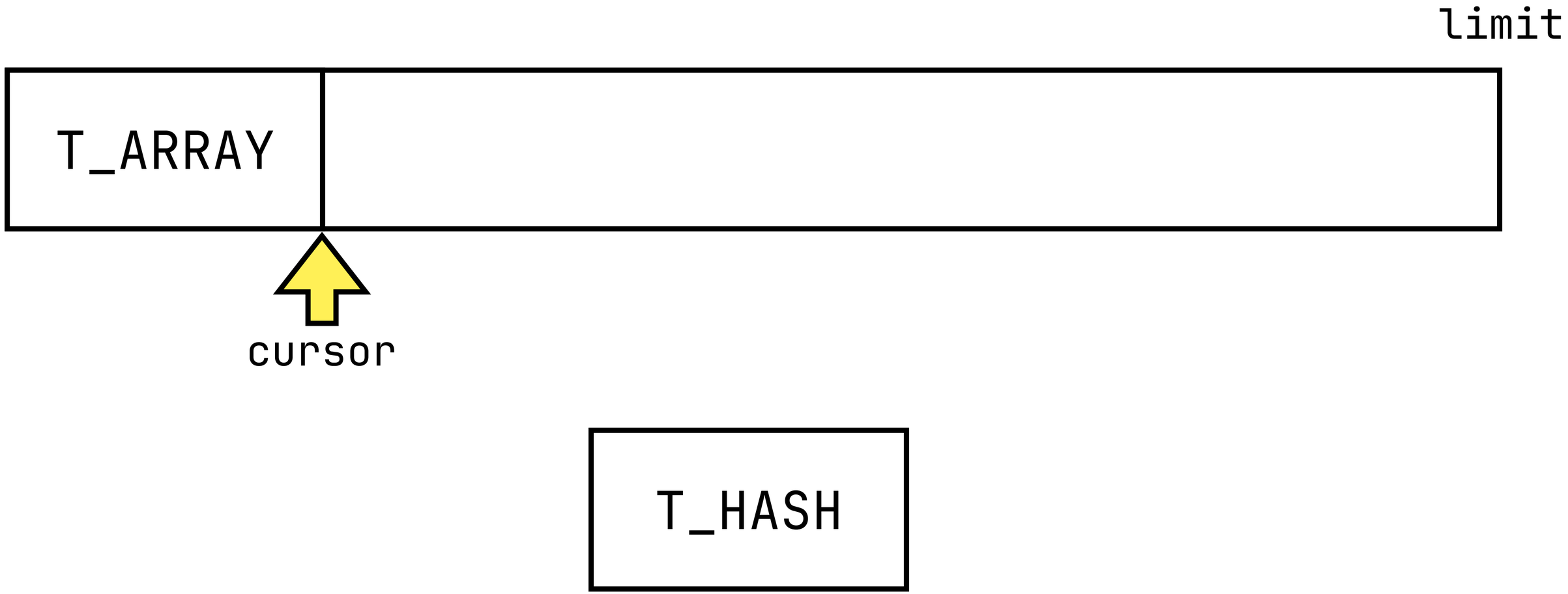

T_HASH

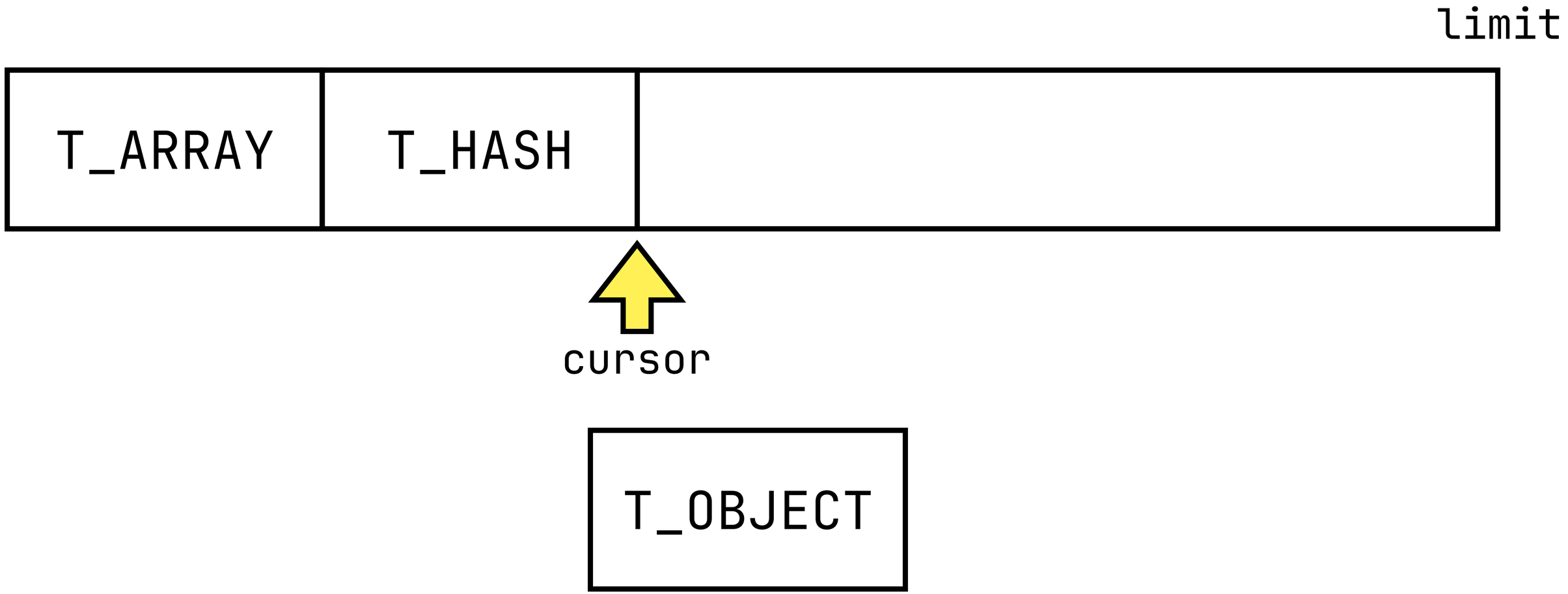

T_OBJECT
T_ARRAY
T_STRING
T_HASH
T_CLASS

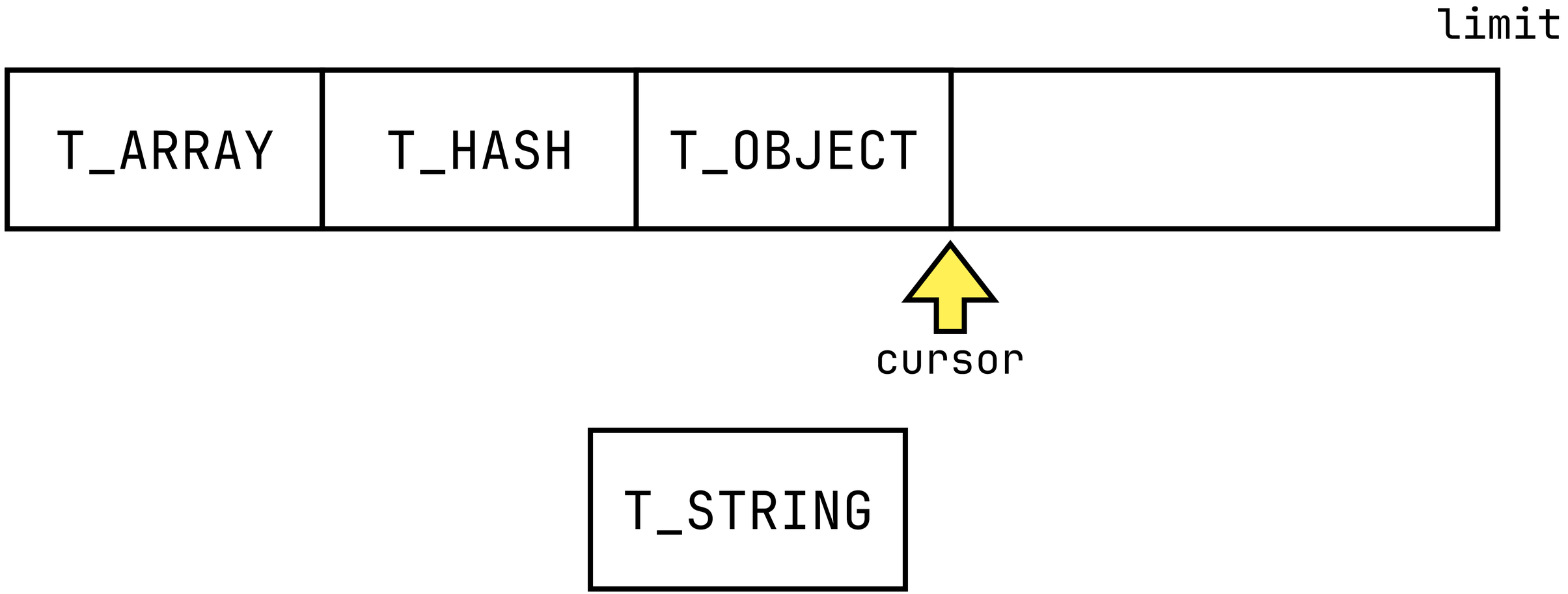


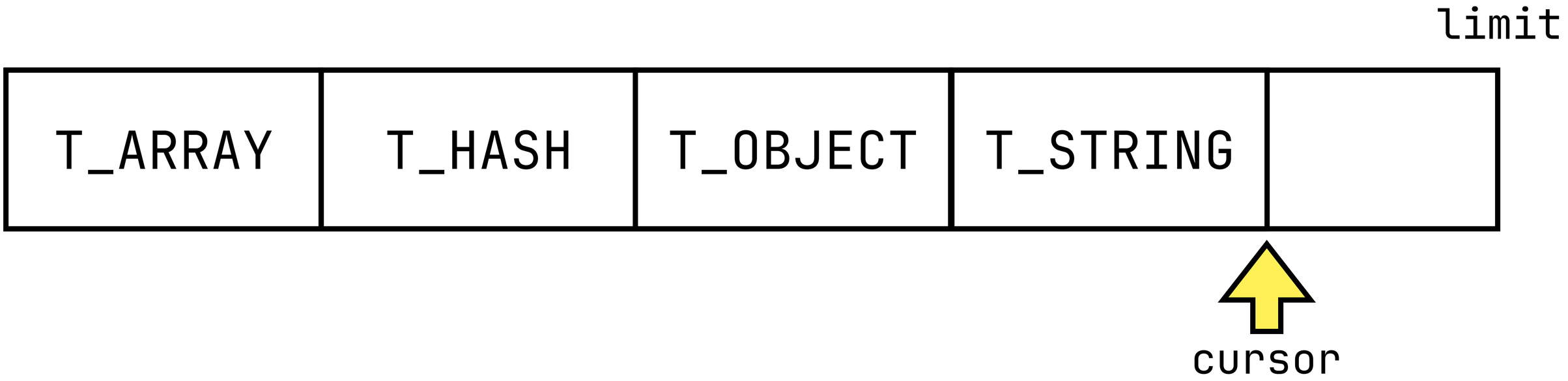
```
def allocate_from_freelist
  obj = @freelist
  if obj
    @freelist = obj.next
    obj
  else
    gc_slowpath
  end
end
```

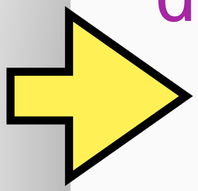

limit
cursor
T_ARRAY


limit
T_ARRAY
cursor
T_HASH


limit
T_ARRAY
T_HASH
cursor
T_OBJECT


limit
T_ARRAY
T_HASH
T_OBJECT
cursor
T_STRING


limit
T_ARRAY
T_HASH
T_OBJECT
T_STRING
cursor



```
def allocate_from_bump_pointer(size)
  obj = @cursor
  @cursor += size
  if @cursor <= @limit
    obj
  else
    gc_slowpath
  end
end
```

```
def allocate_from_freelist
  obj = @freelist
  if obj
    @freelist = obj.next
    obj
  else
    gc_slowpath
  end
end
```

```
def allocate_from_bump_pointer(size)
  obj = @cursor
  @cursor += size
  if @cursor <= @limit
    obj
  else
    gc_slowpath
  end
end
```

# Bump Pointer
# Fast Path Allocator

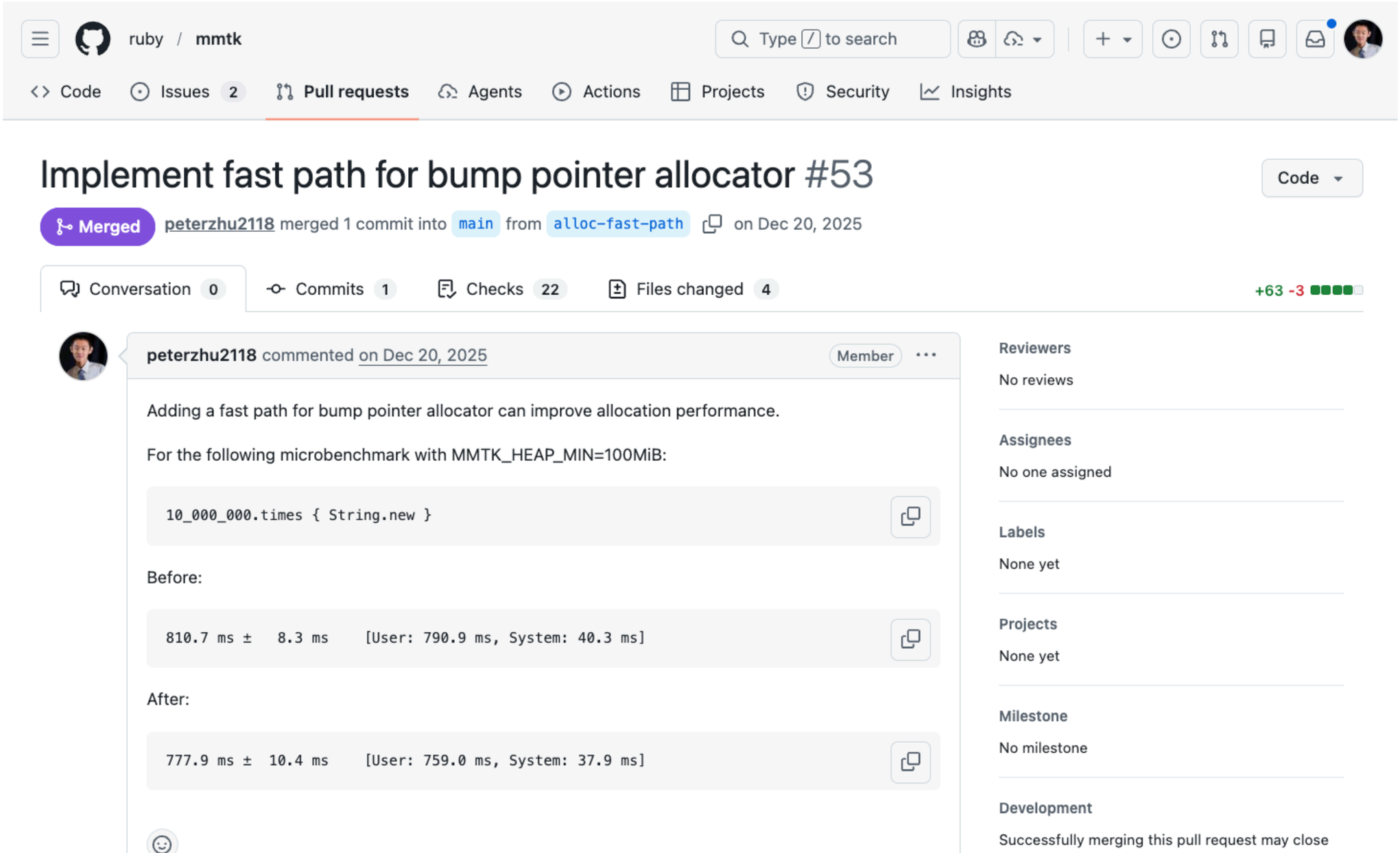
# gc/mmtk/mmtk.c

```c
static VALUE
rb_mmtk_alloc_fast_path(struct objspace *objspace, struct MMTk_ractor_cache *ractor_cache, size_t size)
{
    MMTk_BumpPointer *bump_pointer = ractor_cache->bump_pointer;
    if (bump_pointer == NULL) return 0;

    uintptr_t new_cursor = bump_pointer->cursor + size;

    if (new_cursor > bump_pointer->limit) {
        return 0;
    }
    else {
        VALUE obj = (VALUE)bump_pointer->cursor;
        bump_pointer->cursor = new_cursor;
        return obj;
    }
}
```

# github.com/ruby/mmtk/pull/53



# Ruby Heap

# Heaps in MMTk

# Heaps in MMTk

Fixed Heap

# Heaps in MMTk

Fixed Heap

Dynamic Heap

dl.acm.org/doi/10.1145/3563323

# Optimal Heap Limits for Reducing Browser Memory Use

MARISA KIRISAME, University of Utah, United States
PRANAV SHENOY, University of Utah, United States
PAVEL PANCHEKHA, University of Utah, United States

Garbage-collected language runtimes carefully tune heap limits to reduce garbage collection time and memory usage. However, there's a trade-off: a lower heap limit reduces memory use but increases garbage collection time. Classic methods for setting heap limits include manually tuned heap limits and multiple-of-live-size rules of thumb, but it is not clear when one rule is better than another or how to compare them.

We address this problem with a new framework where heap limits are set for multiple heaps at once. Our key insight is that every heap limit rule induces a particular allocation of memory across multiple processes, and this allocation can be sub-optimal. We use our framework to derive an optimal "square-root" heap limit rule, which minimizes total memory usage for any amount of total garbage collection time. Paradoxically, the square-root heap limit rule achieves this coordination without communication: it allocates memory optimally across multiple heaps without requiring any communication between those heaps.

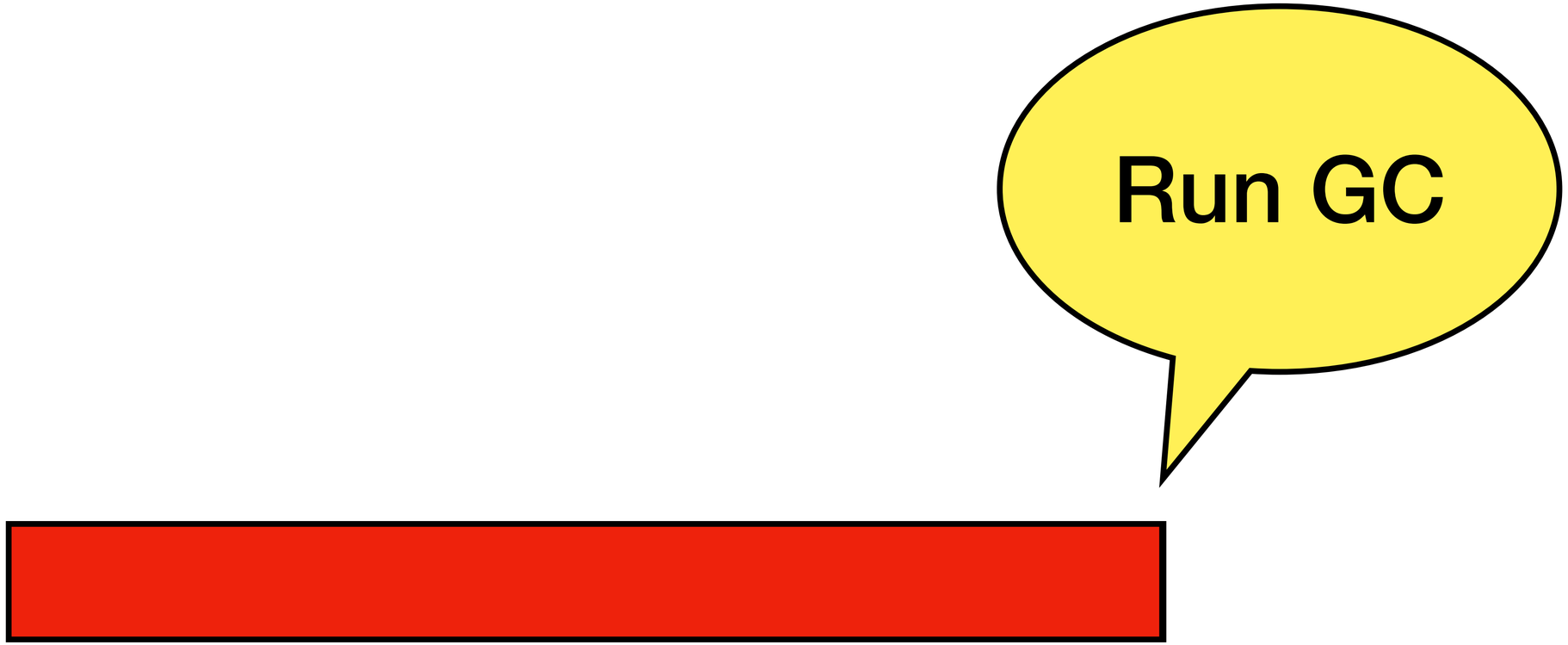
To demonstrate that this heap limit rule is effective, we prototype it for V8, the JavaScript runtime used in Google Chrome, Microsoft Edge, and other browsers, as well as in server-side frameworks like node.js and Deno. On real-world web pages, our prototype achieves reductions of approximately 16.0% of memory usage while keeping garbage collection time constant. On memory-intensive benchmarks, reductions of up to 30.0% of garbage collection time are possible with no change in total memory usage.

CCS Concepts: • **Software and its engineering → Runtime environments**; • **Information systems →**

# Ruby Heap

Used: 60%

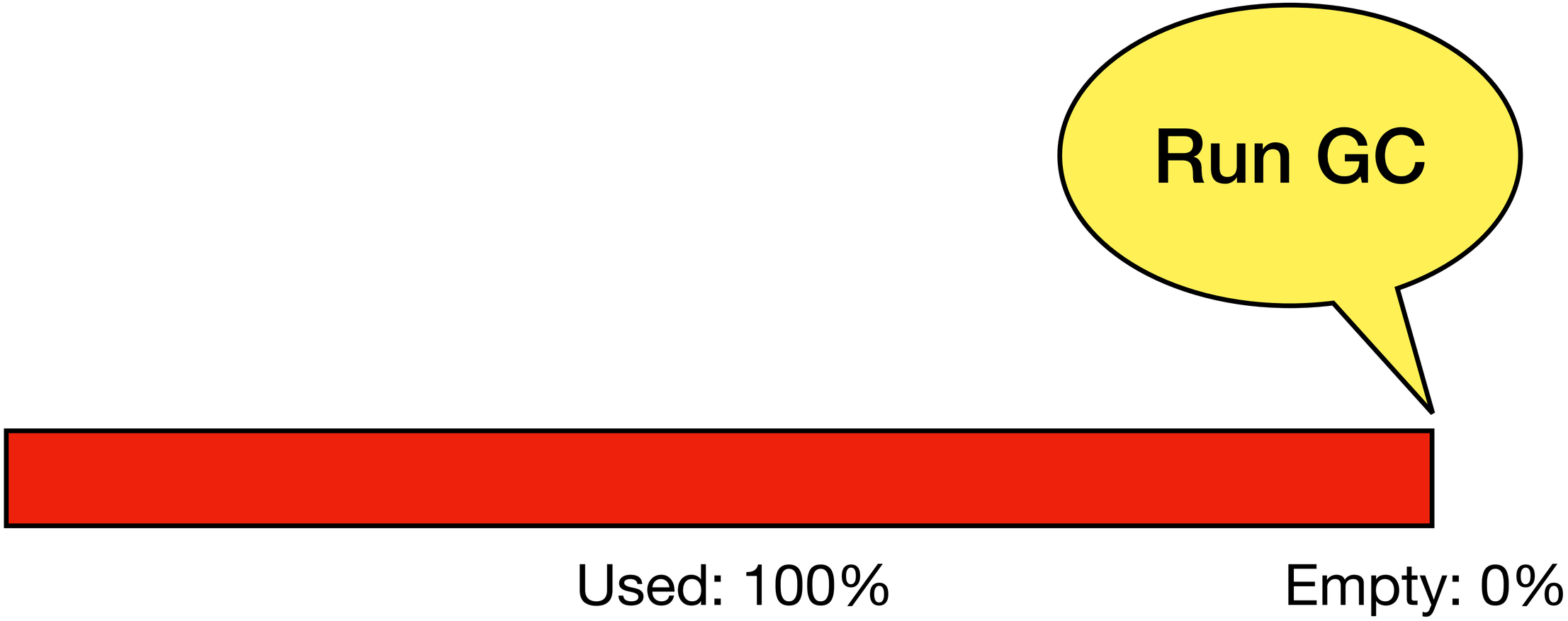
Empty: 40%


Run GC
Used: 100%
Empty: 0%

Used: 90%  Empty: 10%

Used: 60%

Empty: 40%


Run GC
Used: 100%
Empty: 0%

Used: 20%

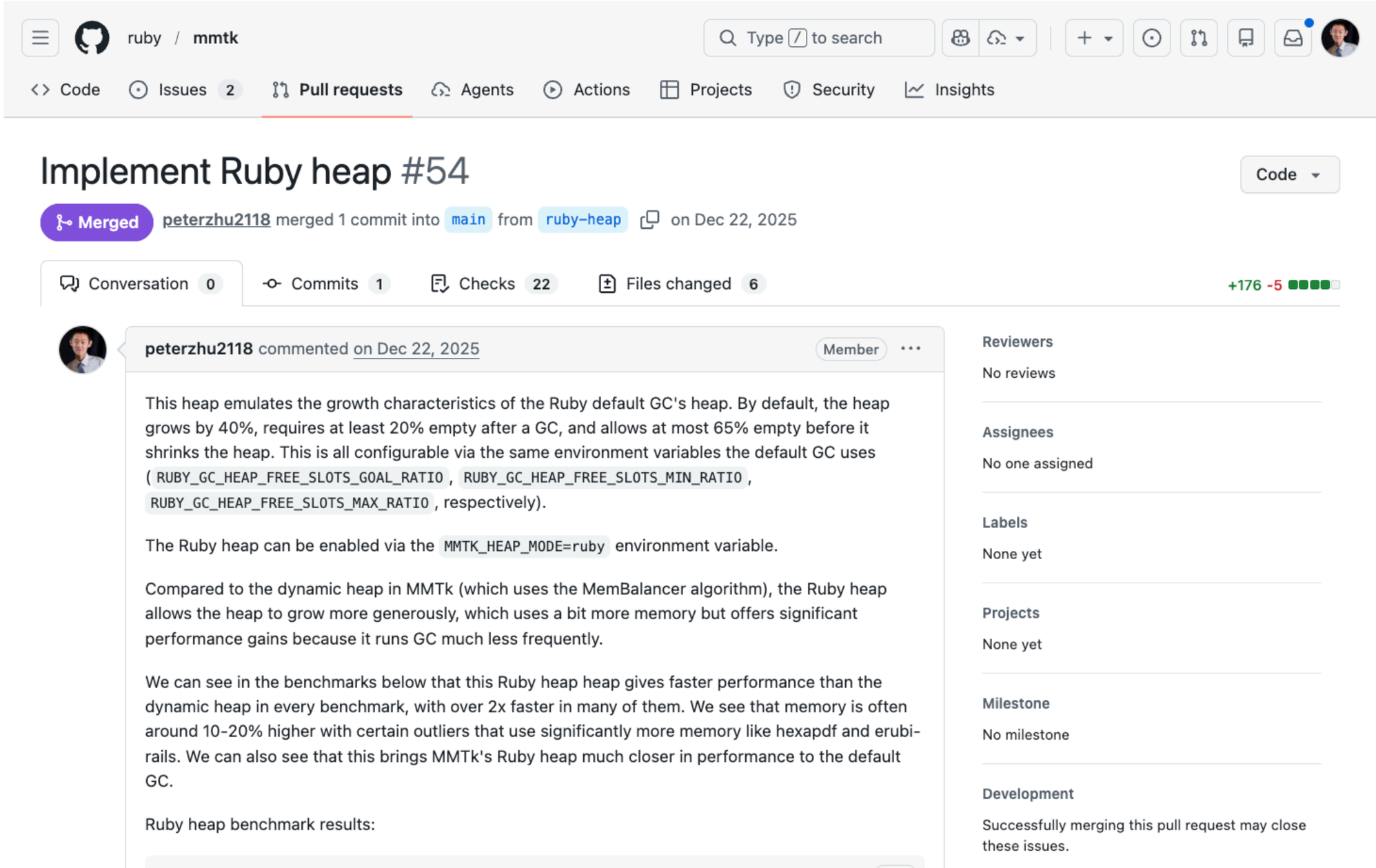
Empty: 80%

Used: 60%

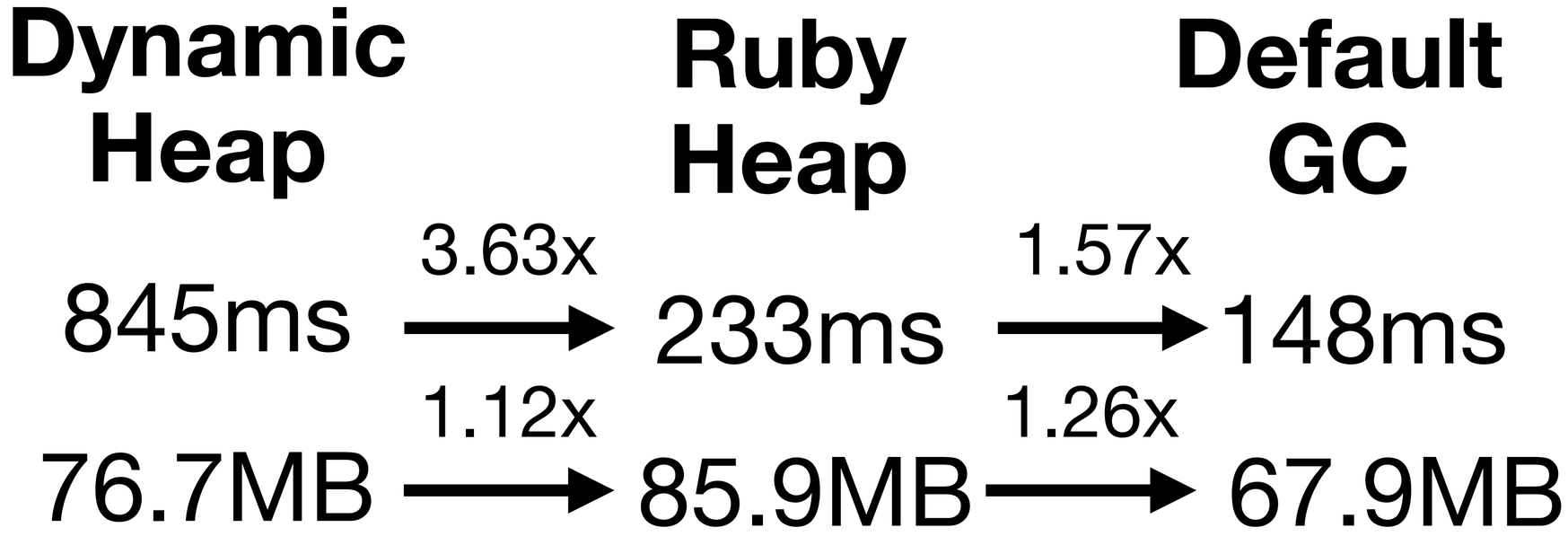
Empty: 40%

# Ruby Heap

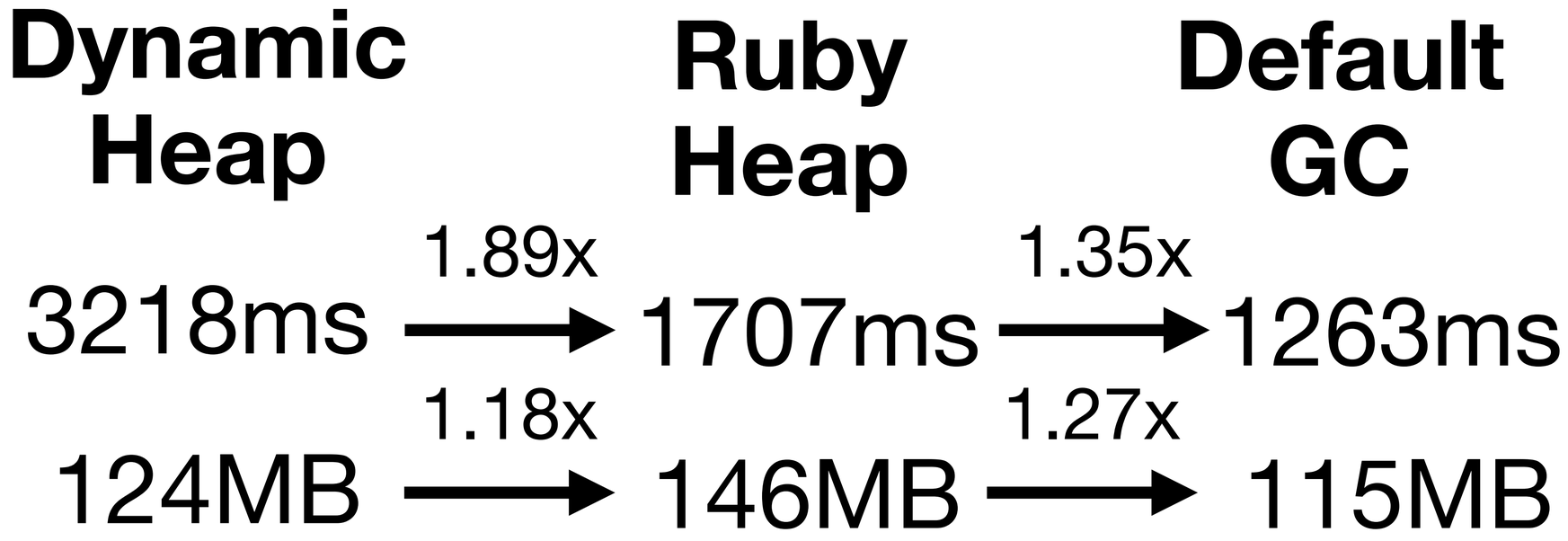
# github.com/ruby/mmtk/pull/54



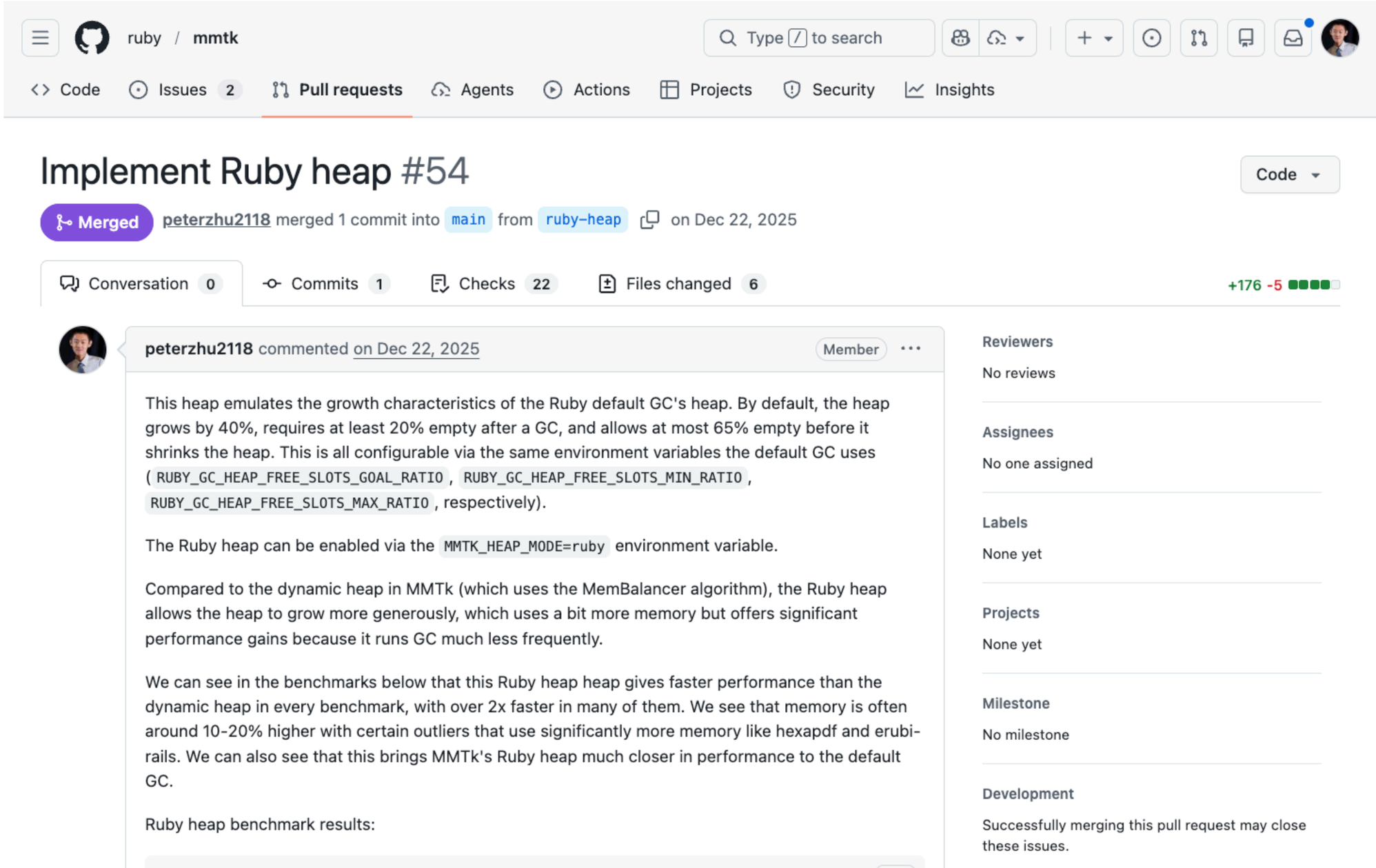
# activerecord



# railsbench


Dynamic Heap
Ruby Heap
Default GC
1.89x
1.35x
3218ms
1707ms
1263ms
1.18x
1.27x
124MB
146MB
115MB

# github.com/ruby/mmtk/pull/54



# Future Roadmap

# Sticky Immix

# Generational GC

# Weak Generational Hypothesis: *Most Objects Die Young*

# Generational GC

# Young & Old Objects

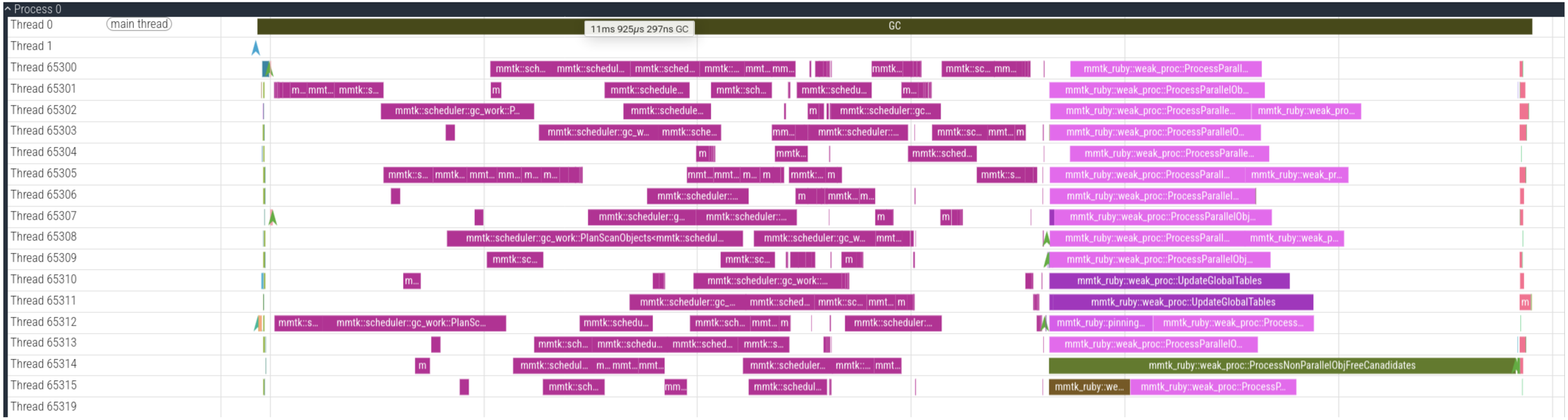
# Young & Old Objects
# Minor & Major GC

# Sticky Immix

# Sticky Mark Bit

# Sticky Immix

# Improve Parallelism


Process 0
Thread 0
main thread
11ms 925μs 297ns GC
GC
Thread 1
Thread 65300
Thread 65301
Thread 65302
Thread 65303
Thread 65304
Thread 65305
Thread 65306
Thread 65307
Thread 65308
Thread 65309
Thread 65310
Thread 65311
Thread 65312
Thread 65313
Thread 65314
Thread 65315
Thread 65319
mmtk_ruby::weak_proc::ProcessNonParallelObjFreeCanadidates
mmtk_ruby::weak_proc::UpdateGlobalTables

# Implement Buffer Objects for String & Array

# Inline GC Fast Path in JIT Compiler

- Introduce Ruby's Garbage Collector
- Modular Garbage Collectors
- Memory Management Toolkit
  - Improvements since RubyKaigi 2025
    - Moving Immix
    - Bump Pointer Allocator Fast Path
    - Ruby Heap
  - Future Roadmap

# Thank You!


blog.peterzhu.ca

peter@peterzhu.ca

@peterzhu2118

@peterzhu2118@ruby.social

@peterzhu.ca

@peterzhu.photos