Plug & Play Garbage Collection with MMTk

RubyKaigi 2023

Matt Valentine-House (eightbitraptor).

Senior Developer, Ruby & Rails Infrastructure Team, Shopify.

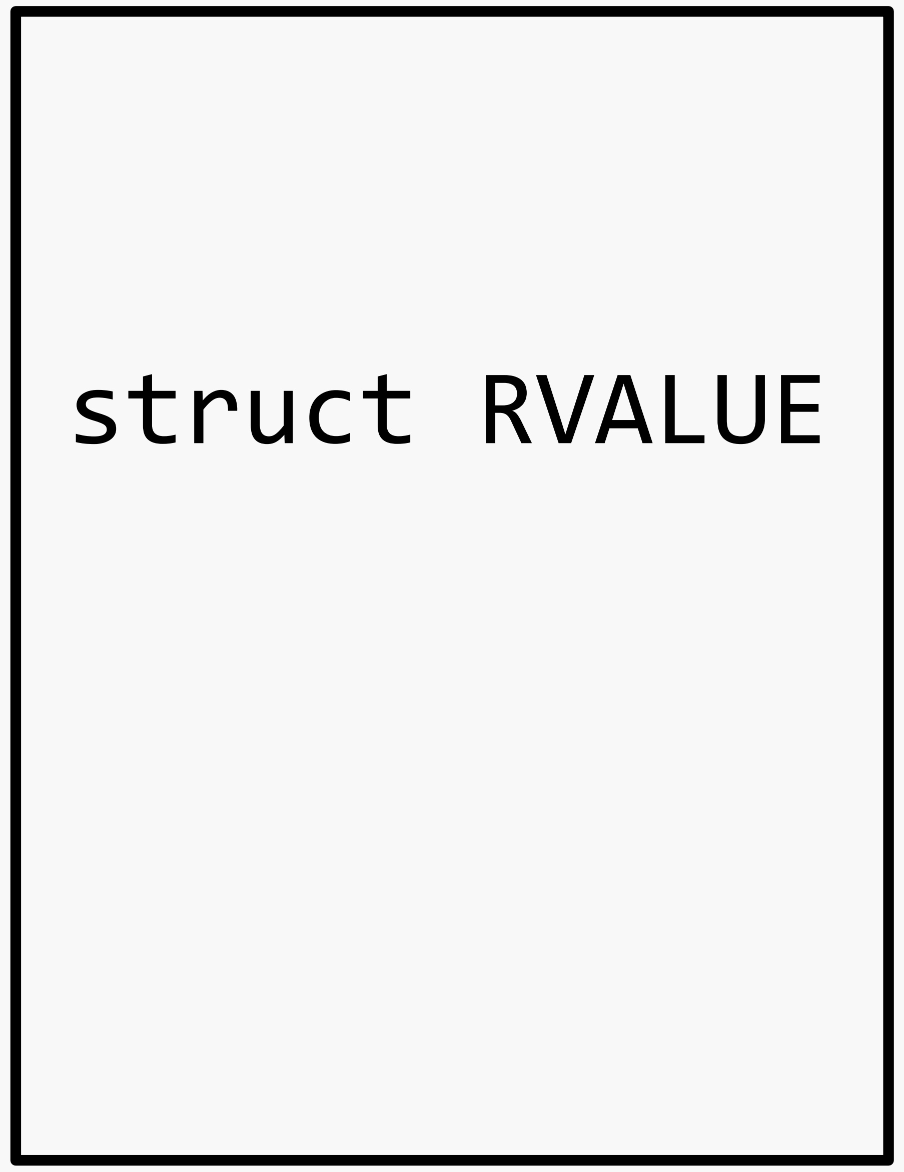
CRuby Committer since March 2023.

1995

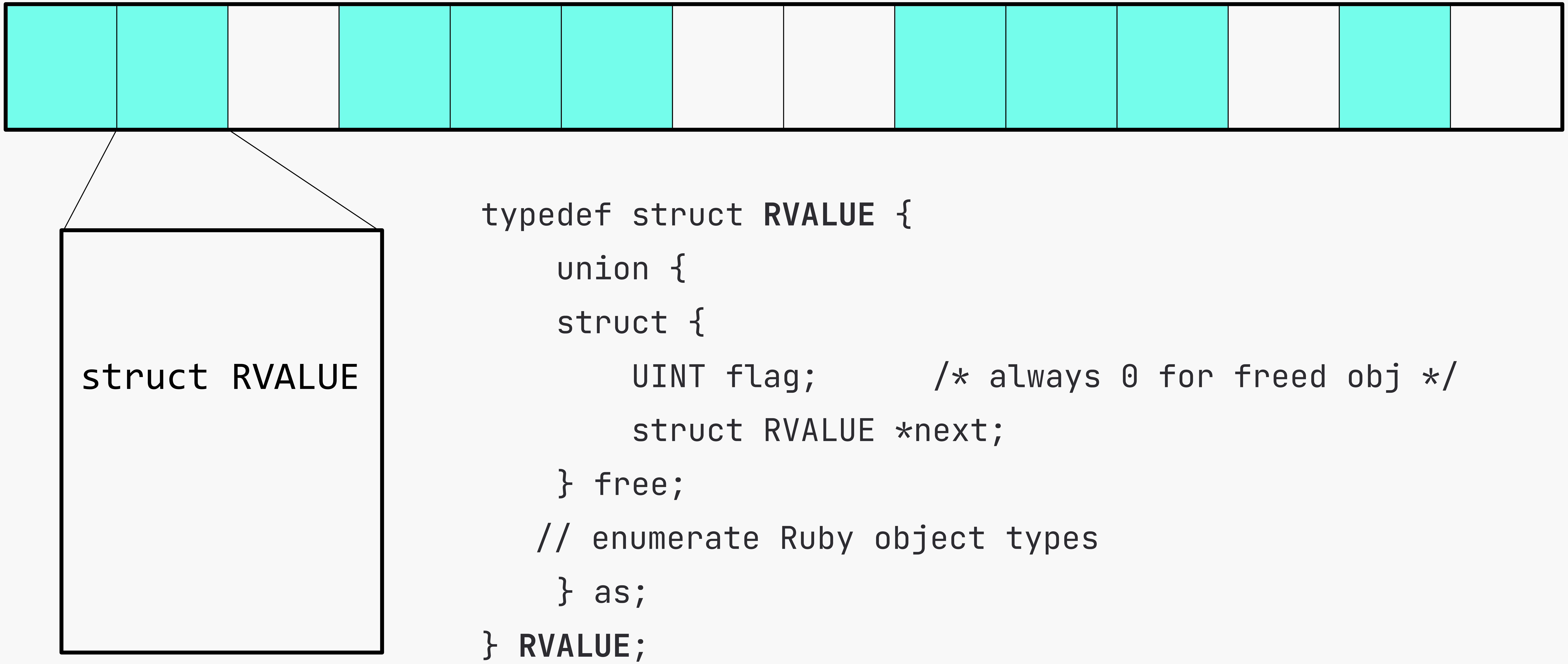
Ruby 0.95 released

562

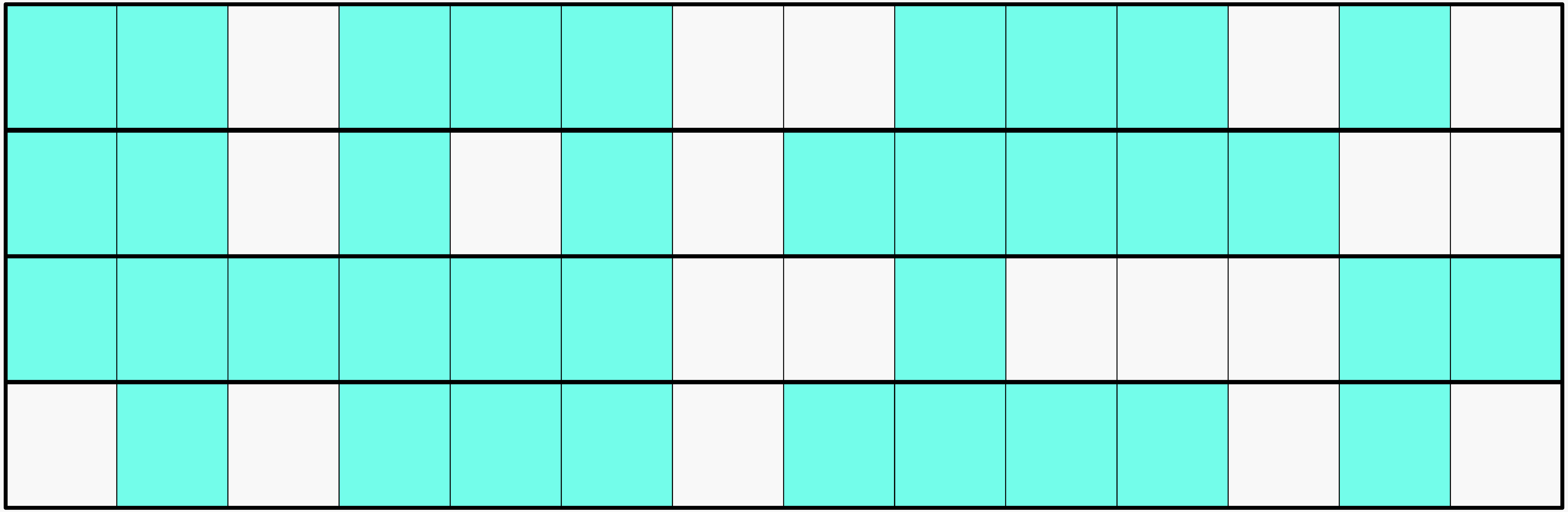
562 → 14100

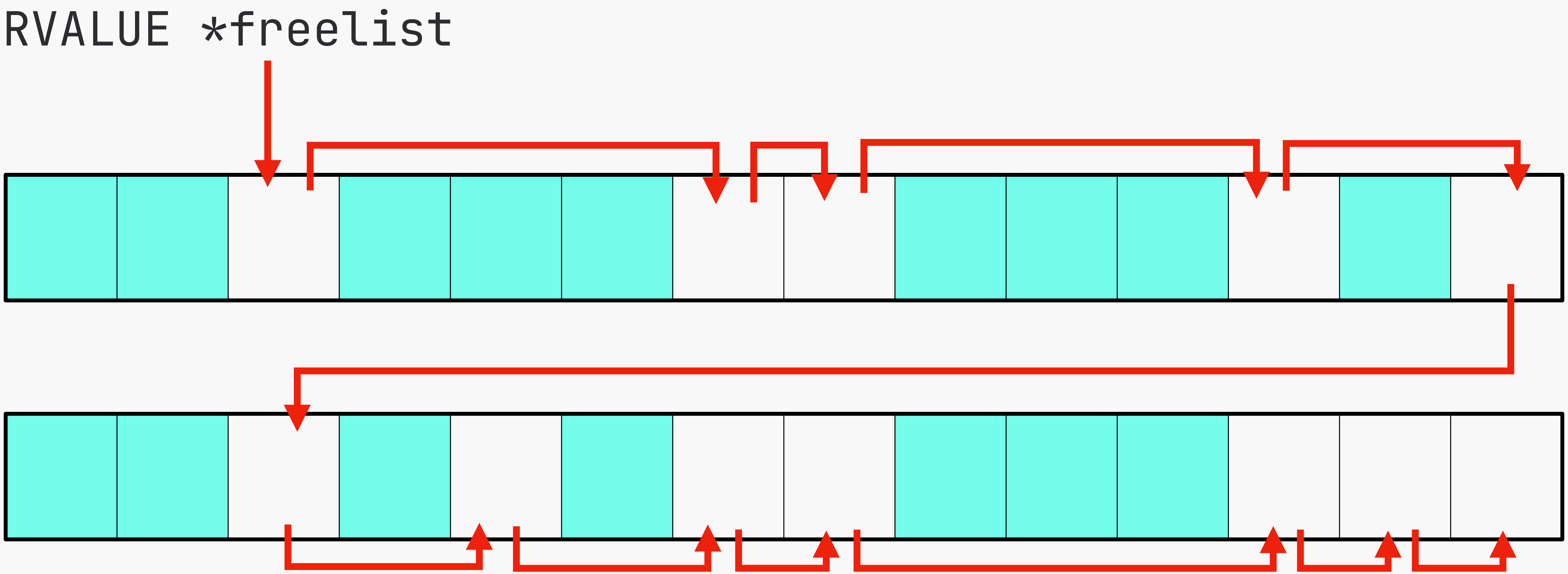


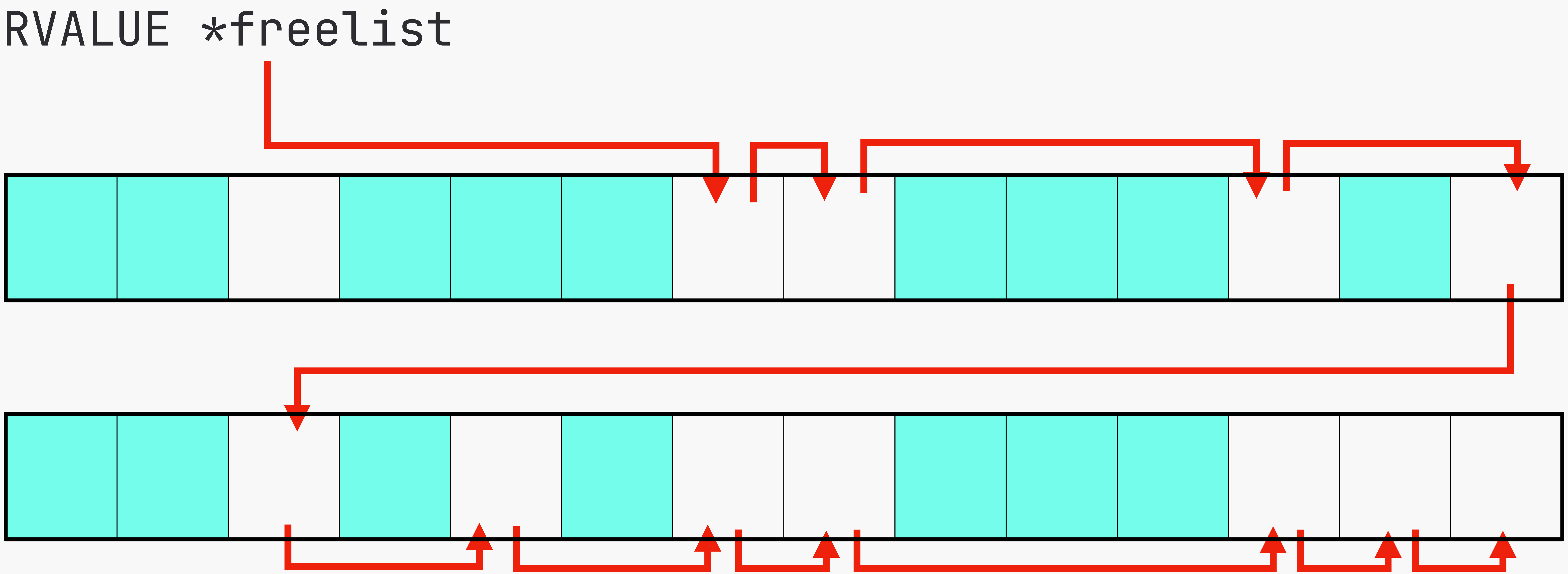

```
typedef struct RVALUE {
    union {
    struct {
        UINT flag;      /* always 0 for freed obj */
        struct RVALUE *next;
    } free;
   // enumerate Ruby object types
    } as;
} RVALUE;
```

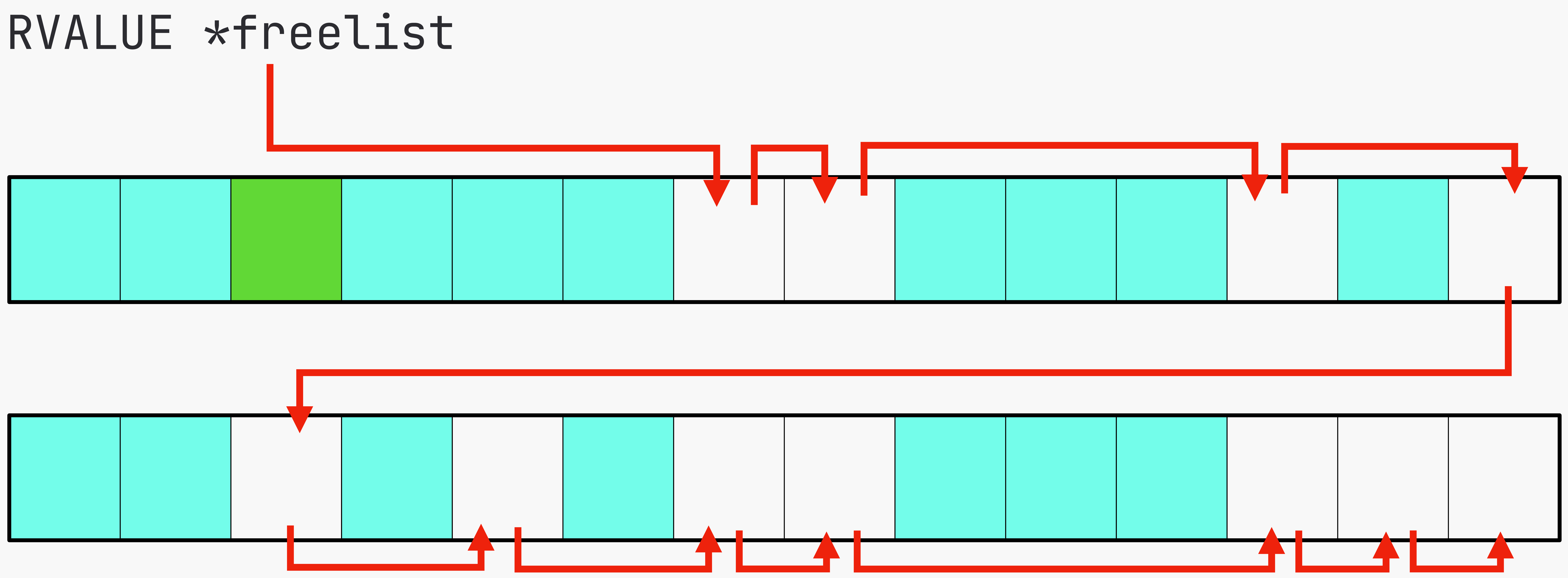



```
typedef struct RVALUE {
    union {
    struct {
        UINT flag;      /* always 0 for freed obj */
        struct RVALUE *next;
    } free;
   // enumerate Ruby object types
    } as;
} RVALUE;
```




RVALUE *freelist

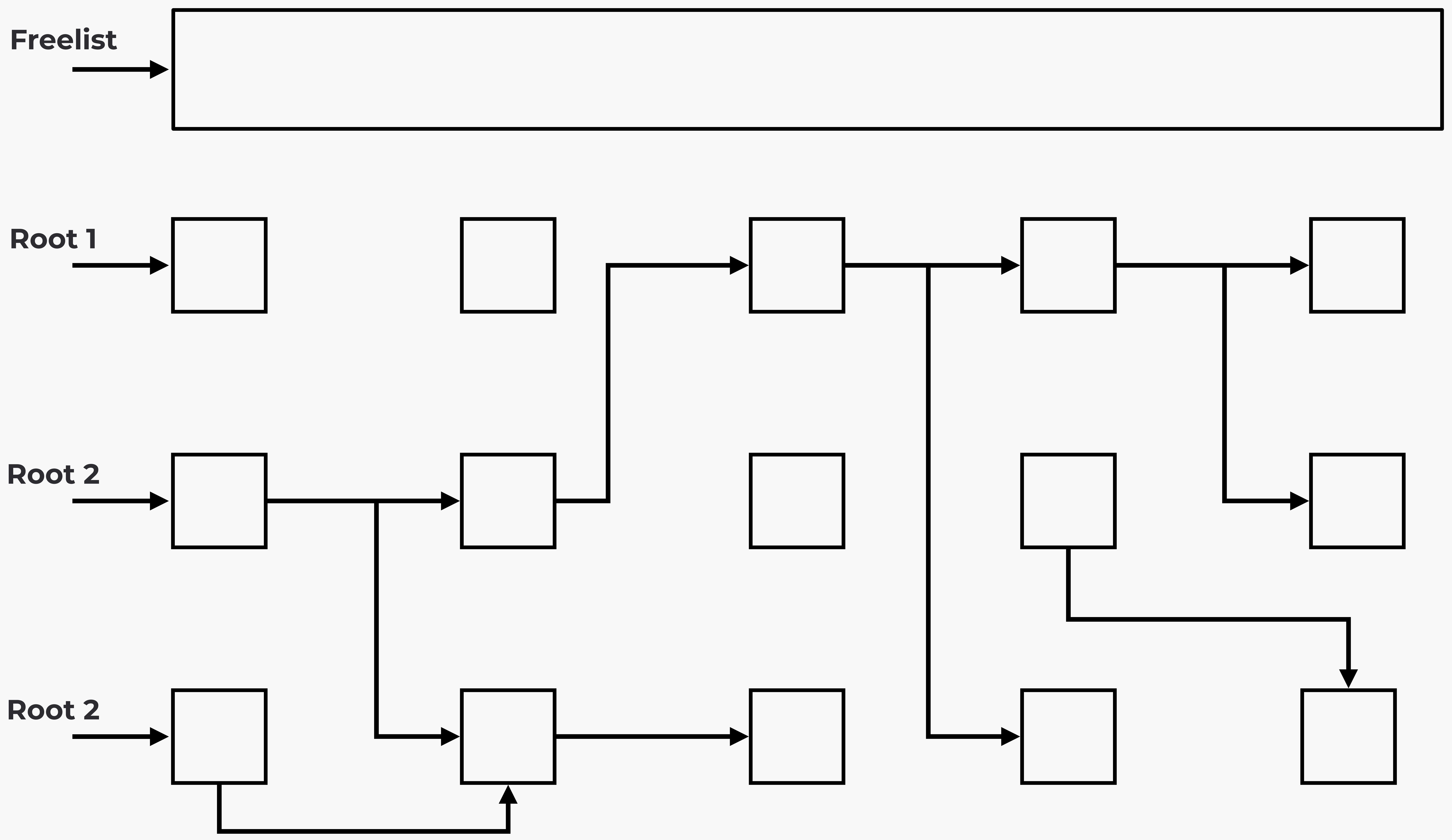

RVALUE *freelist


RVALUE *freelist

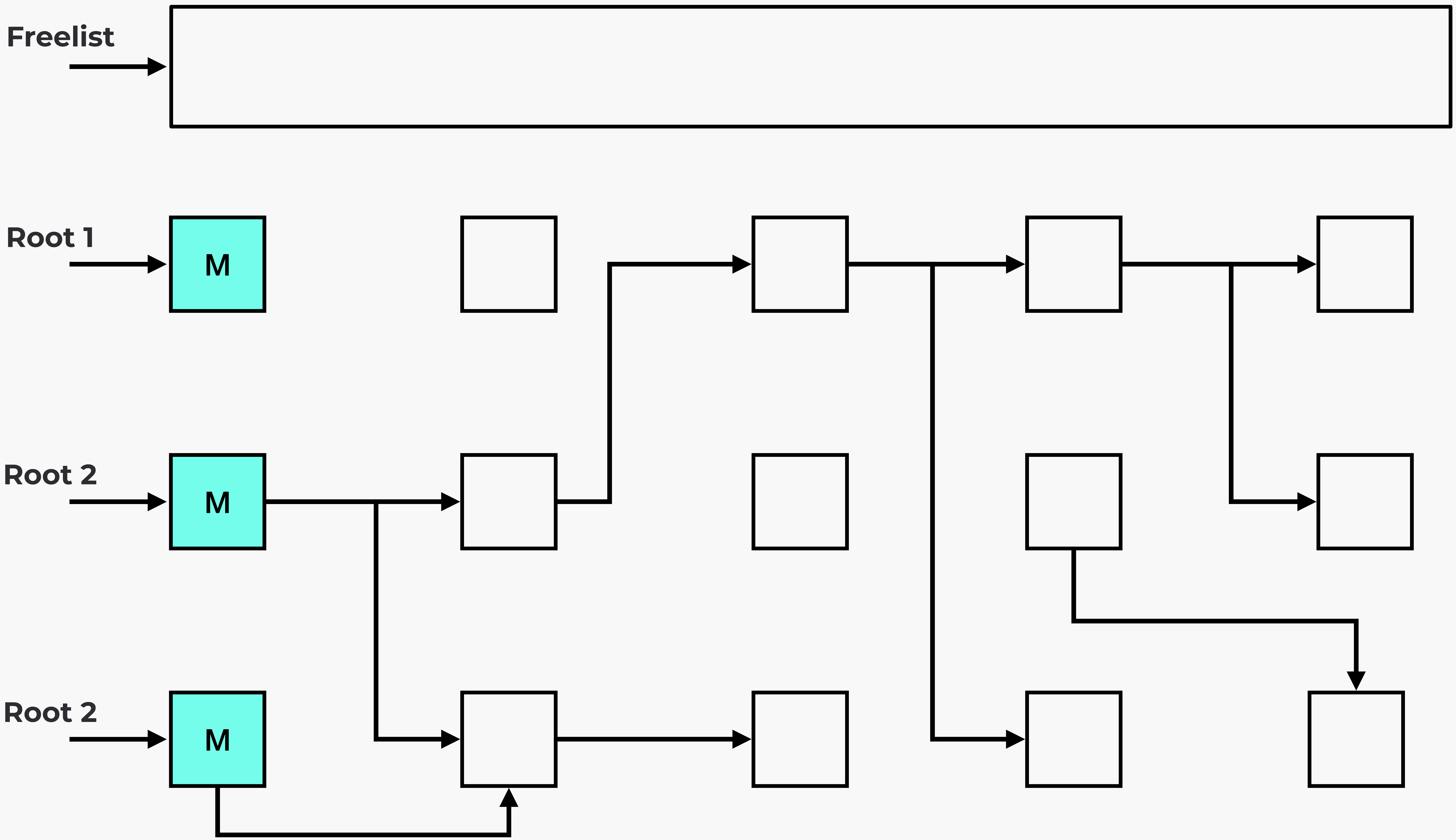
“Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1”

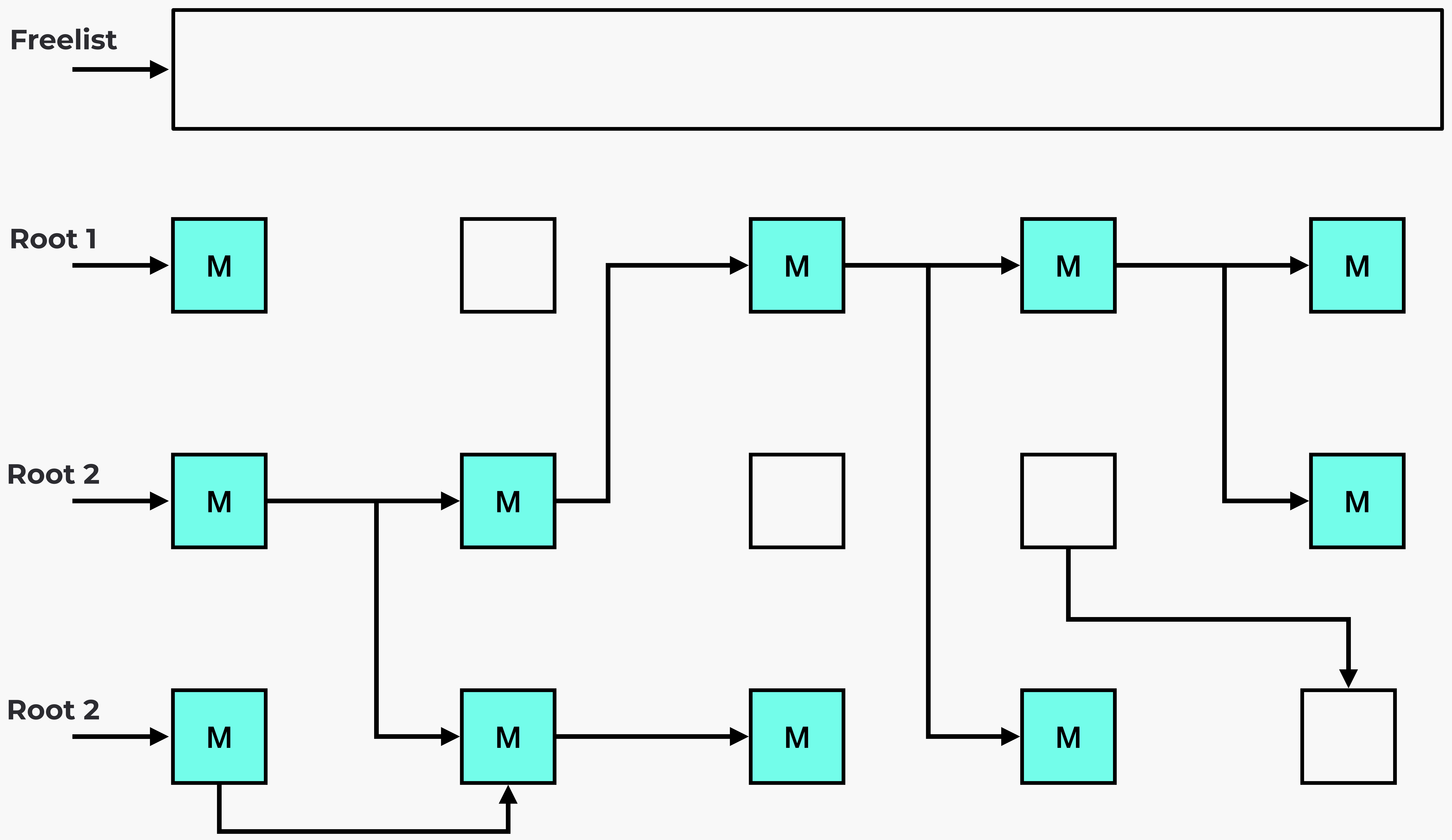
John McCarthy
Massachusetts Institute of Technology

Communications of the ACM, Volume 3, Issue 4. April 1960

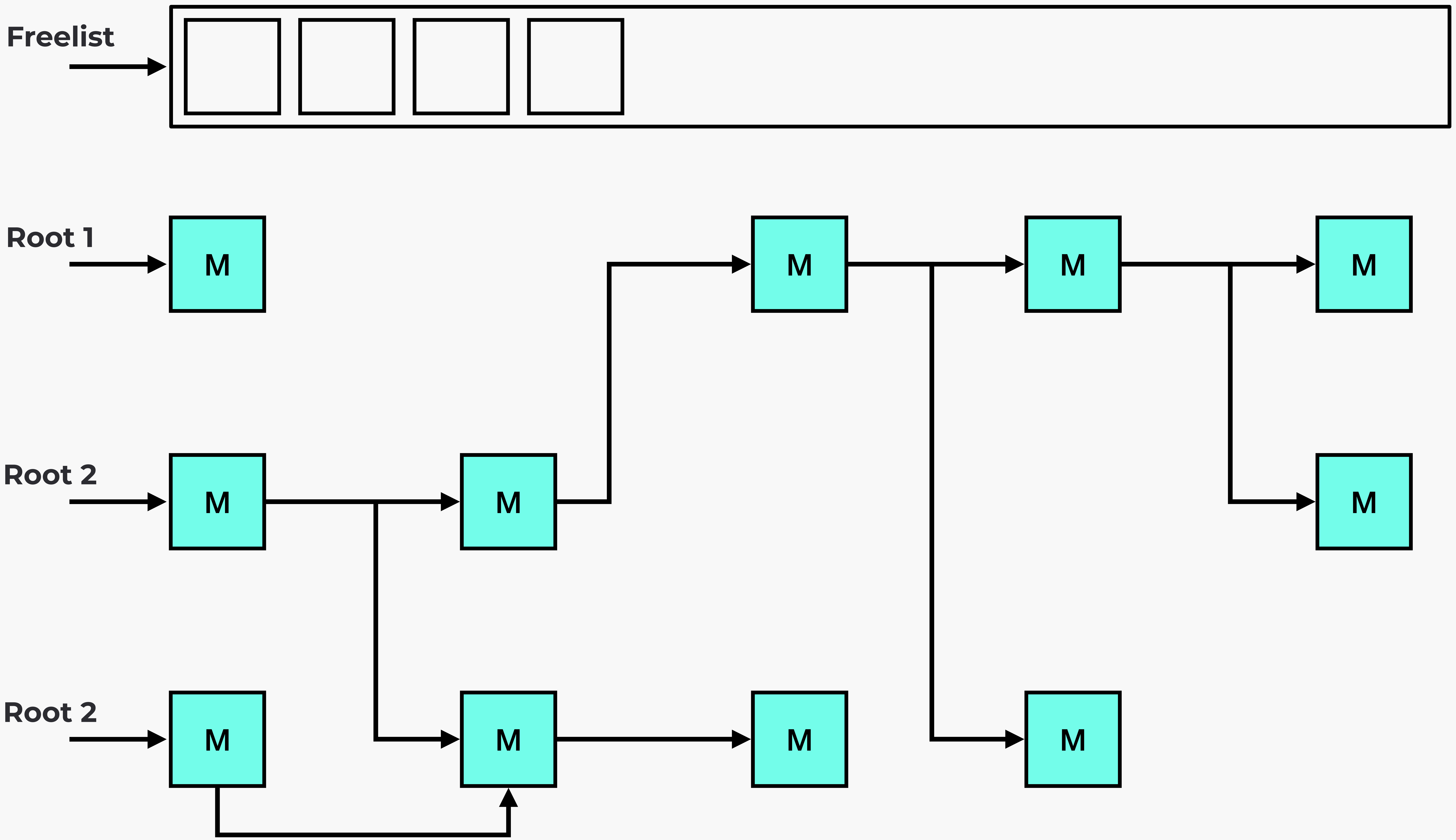


Freelist
Root 1
Root 2
Root 2


Freelist
Root 1
M
Root 2
M
Root 2
M



Freelist
Root 1
Root 2
Root 2
M
M
M
M
M
M
M
M
M
M
M


Freelist
Root 1
Root 2
Root 2
M

```
gc_mark(obj)
    register RVALUE *obj;
{
  Top:
    if (obj == Qnil) return;    /* nil not marked */
    if (FIXNUM_P(obj)) return;  /* fixnum not marked */
    if (obj->as.basic.flags == 0) return; /* free cell */
    if (obj->as.basic.flags & FL_MARK) return; /* marked */
    obj->as.basic.flags |= FL_MARK;
    switch (obj->as.basic.flags & T_MASK) {
      case T_NIL:
      case T_FIXNUM:
    Bug("gc_mark() called for broken object");
    break;
    /// ... snip...
}
```

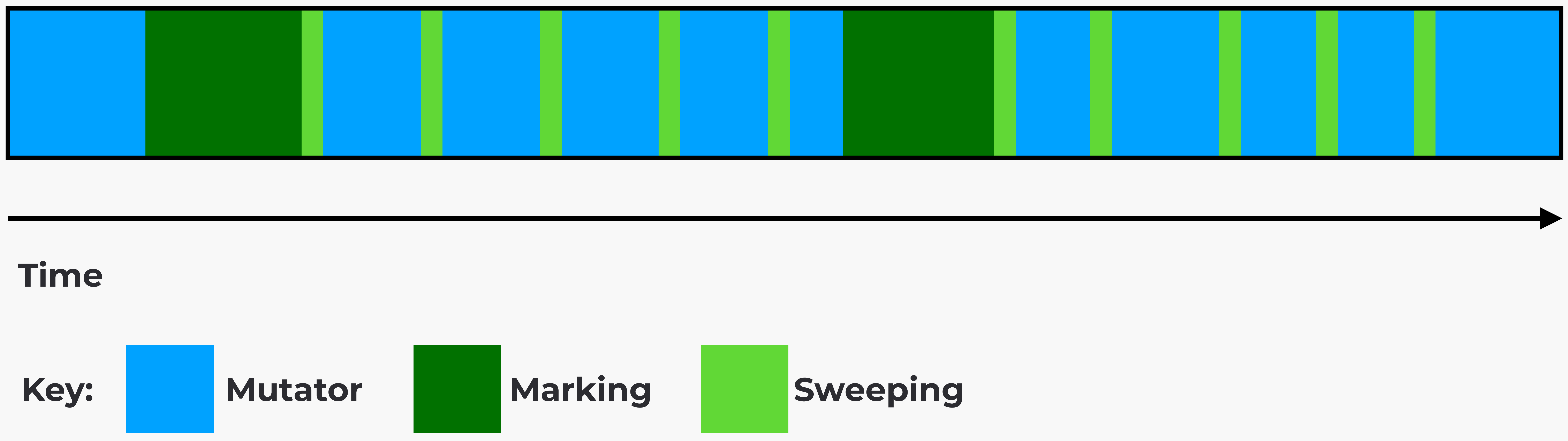
```
gc_sweep()
{
    // ... snip
    while (p < pend) {
        if (!(p->as.basic.flags & FL_MARK)) {
        if (p->as.basic.flags) obj_free(p);
        p->as.free.flag = 0;
        p->as.free.next = nfreelist;
        nfreelist = p;
        n++;
        }
        else
        RBASIC(p)->flags &= ~FL_MARK;
        p++;
    }
   // ... snip
}
```

```
gc_sweep()
{
    // ... snip
    while (p < pend) {
        if (!(p->as.basic.flags & FL_MARK)) {
        if (p->as.basic.flags) obj_free(p);
        p->as.free.flag = 0;
        p->as.free.next = nfreelist;
        nfreelist = p;
        n++;
        }
        else
        RBASIC(p)->flags &= ~FL_MARK;
        p++;
    }
   // ... snip
}
```

2011

Ruby 1.9.3 introduced Lazy Sweeping


Time
Key:
Mutator
Garbage Collection


Time
Key:
Mutator
Marking
Sweeping

Lower throughput.

Fewer total requests per second.

Lower p99 response times.

Fewer slow requests for users.

2013

Ruby 2.1 introduced
Generational GC

Measurement of Object lifetimes proved that young objects die young and old objects continue to live

– David Ungar
University of California, 1984

RGenGC - 2 generations: Young and Old

Two phase Marking:

- Minor - only young objects
- Major - all objects

Minor mark by default, Major when Old object count doubles

RGenGC - 2 generations: Young and Old

Two phase Marking:

Minor - only young objects

Major - all objects

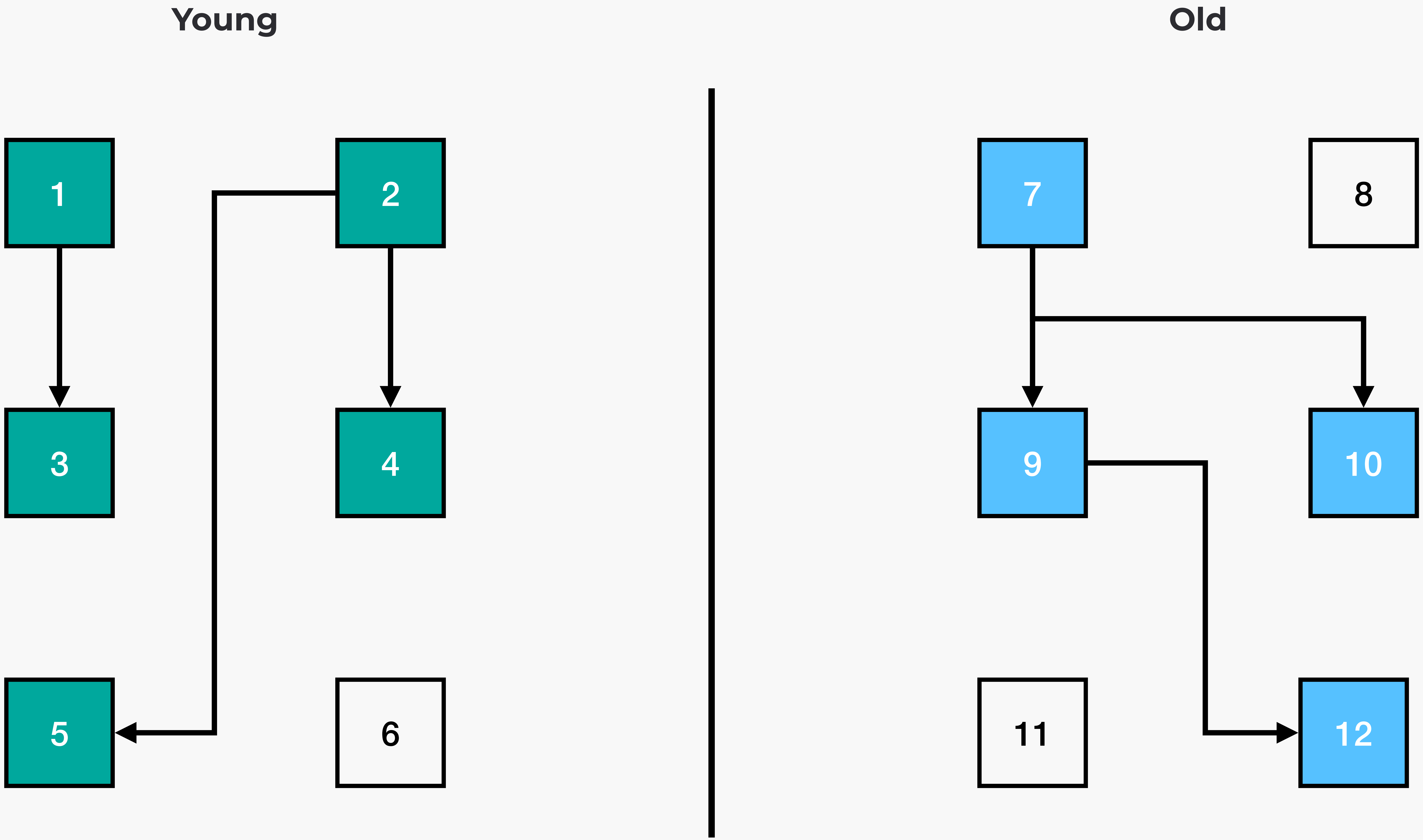
Minor mark by default, Major when Old object count doubles

RGenGC - 2 generations: Young and Old

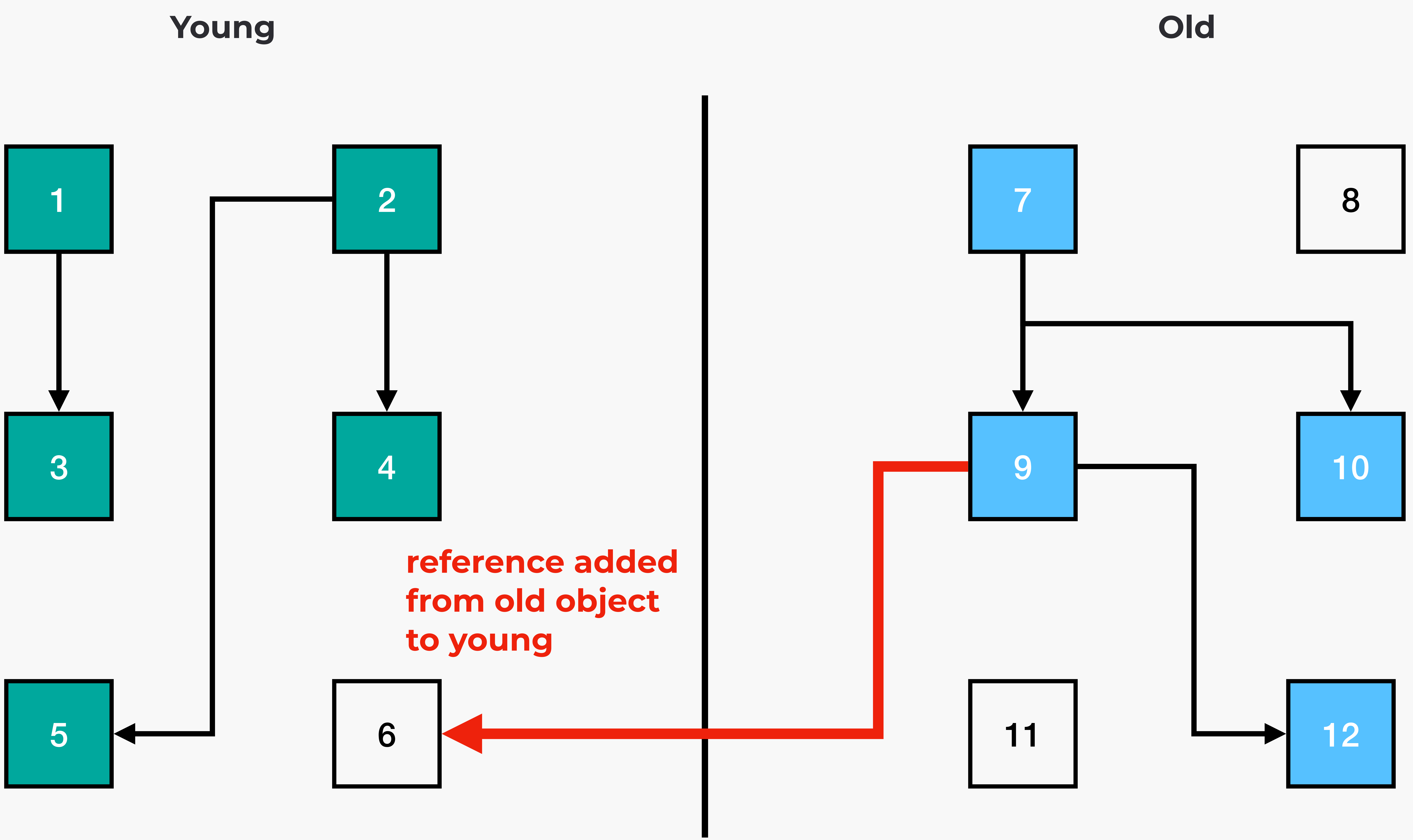
Two phase Marking:

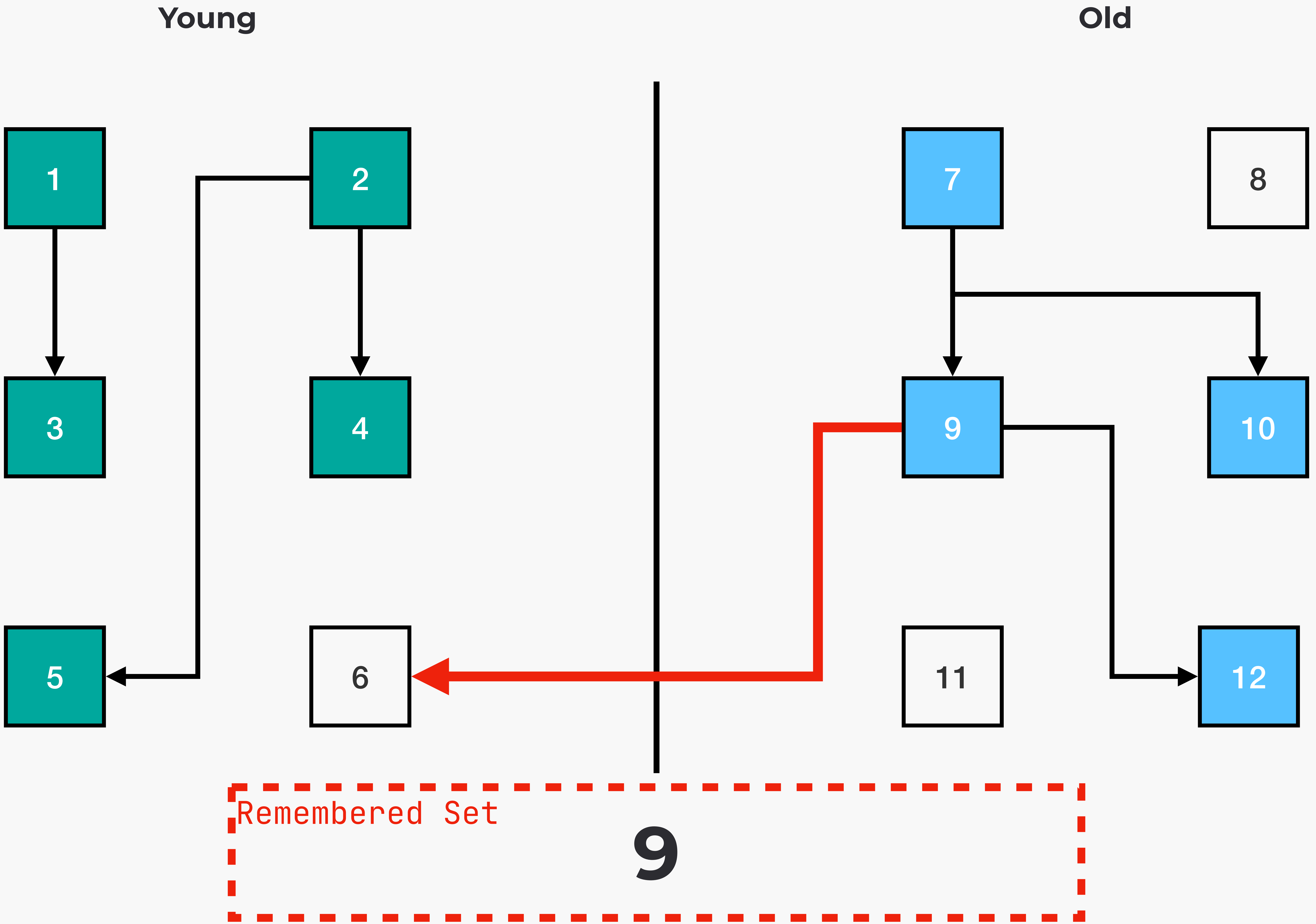
- Minor - only young objects
- Major - all objects

Minor mark by default, Major when Old object count doubles



Young
Old
1
2
3
4
5
6
7
8
9
10
11
12


Young
Old
1
2
3
4
5
6
7
8
9
10
11
12
reference added
from old object
to young


Young
Old
1
2
3
4
5
6
7
8
9
10
11
12
Remembered Set
9

RGenGC introduced Write Barriers.

C extension objects are "Write Barrier Unprotected".

Unprotected objects can never be old.

RGenGC backwards compatible, at expense of performance.

RGenGC introduced Write Barriers.

C extension objects are "Write Barrier Unprotected".

Unprotected objects can never be old.

RGenGC backwards compatible, at expense of performance.

RGenGC introduced Write Barriers.

C extension objects are "Write Barrier Unprotected".

Unprotected objects can never be old.

RGenGC backwards compatible, at expense of performance.

RGenGC introduced Write Barriers.

C extension objects are "Write Barrier Unprotected".

Unprotected objects can never be old.

RGenGC backwards compatible, at expense of performance.

RGenGC introduced Write Barriers.

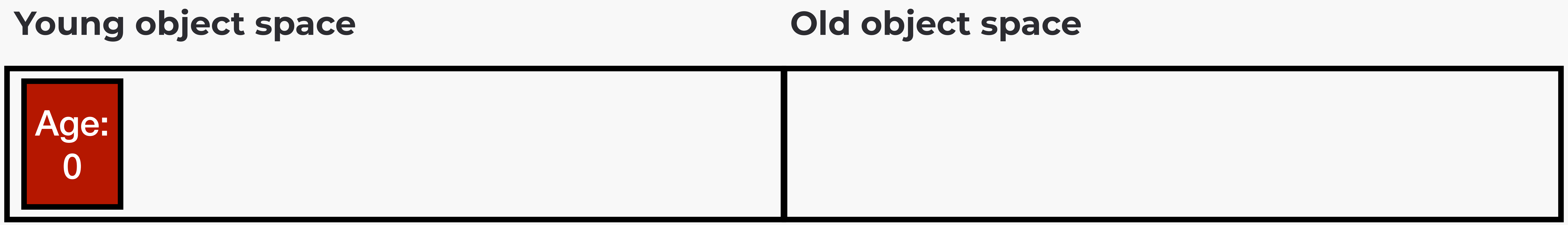
C extension objects are "Write Barrier Unprotected".

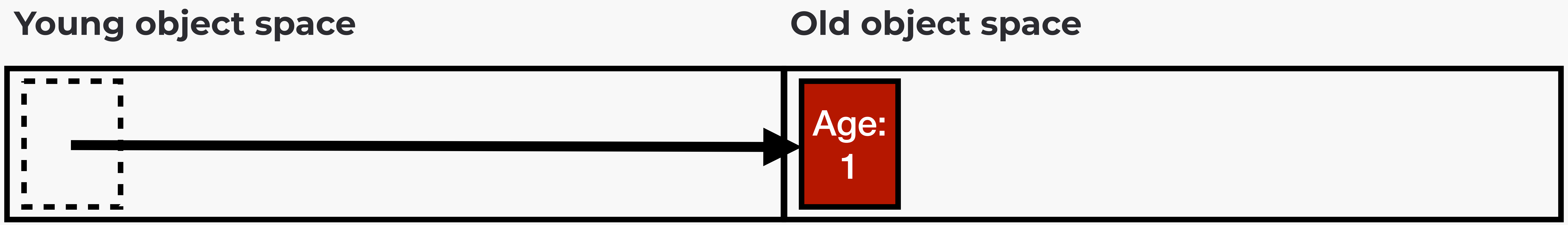
Unprotected objects can never be old.

RGenGC backwards compatible, at expense of performance.

Ruby's Generational GC is not evacuating.

Evacuation allows greater performance tuning.


Young object space
Old object space
Age:
0

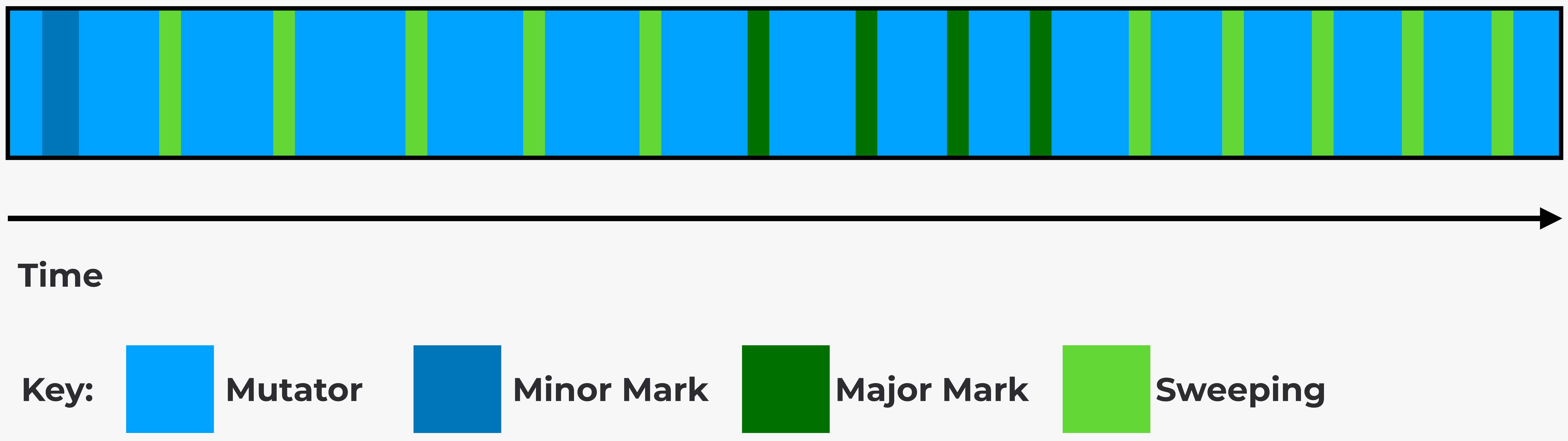

Young object space
Old object space
Age:
1

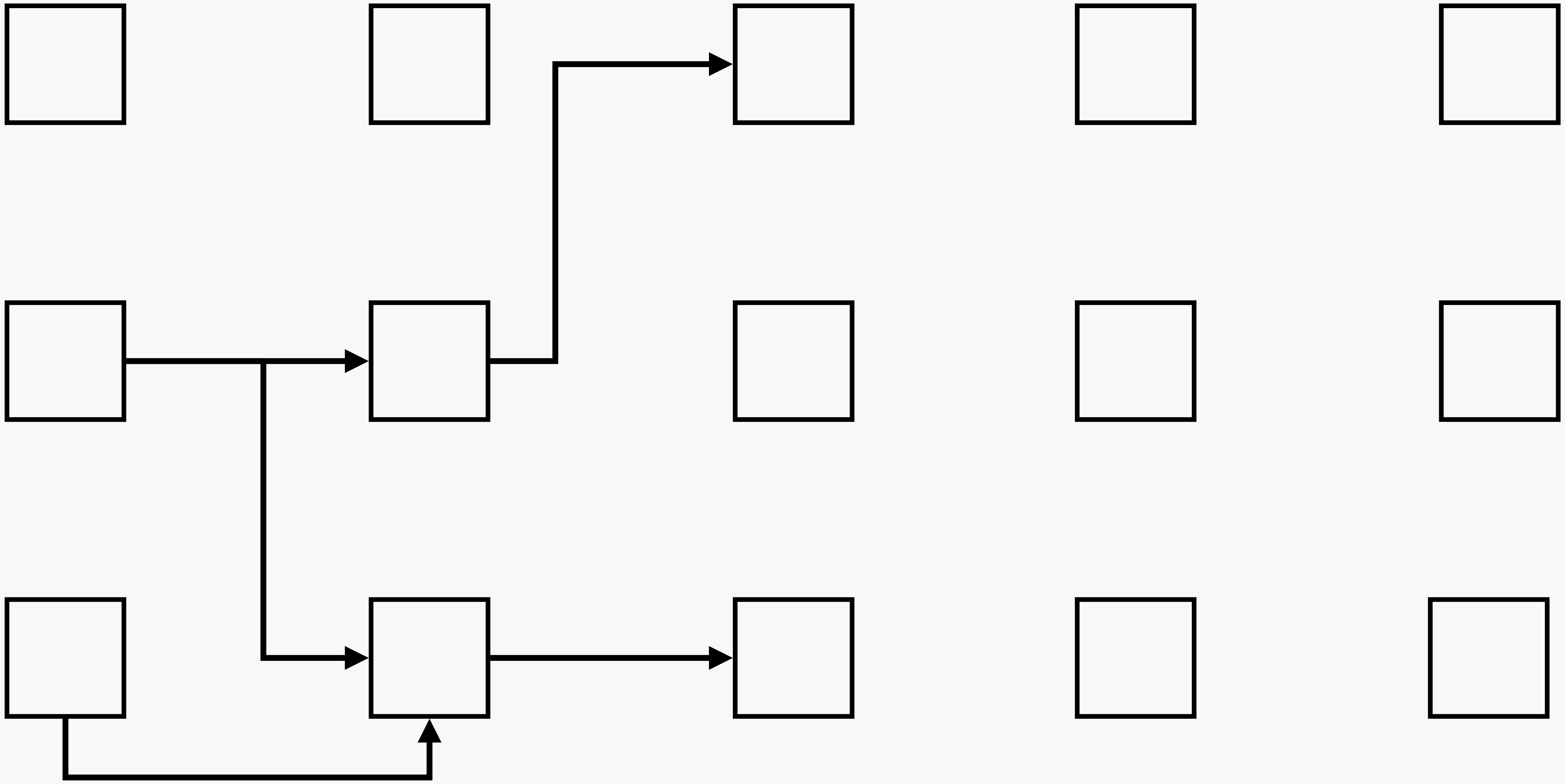
Ruby's Generational GC is not evacuating.

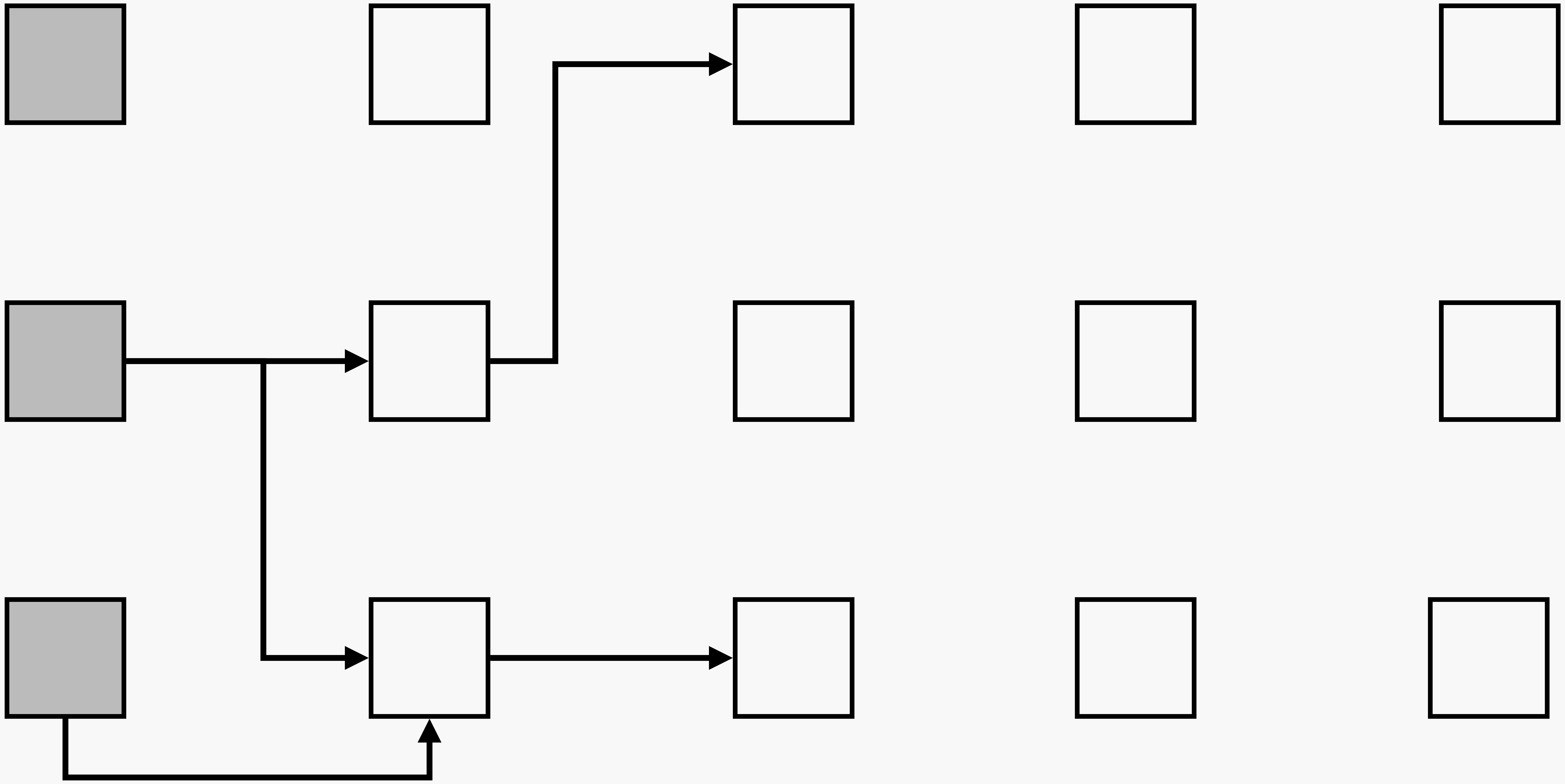
Evacuation allows greater performance tuning.

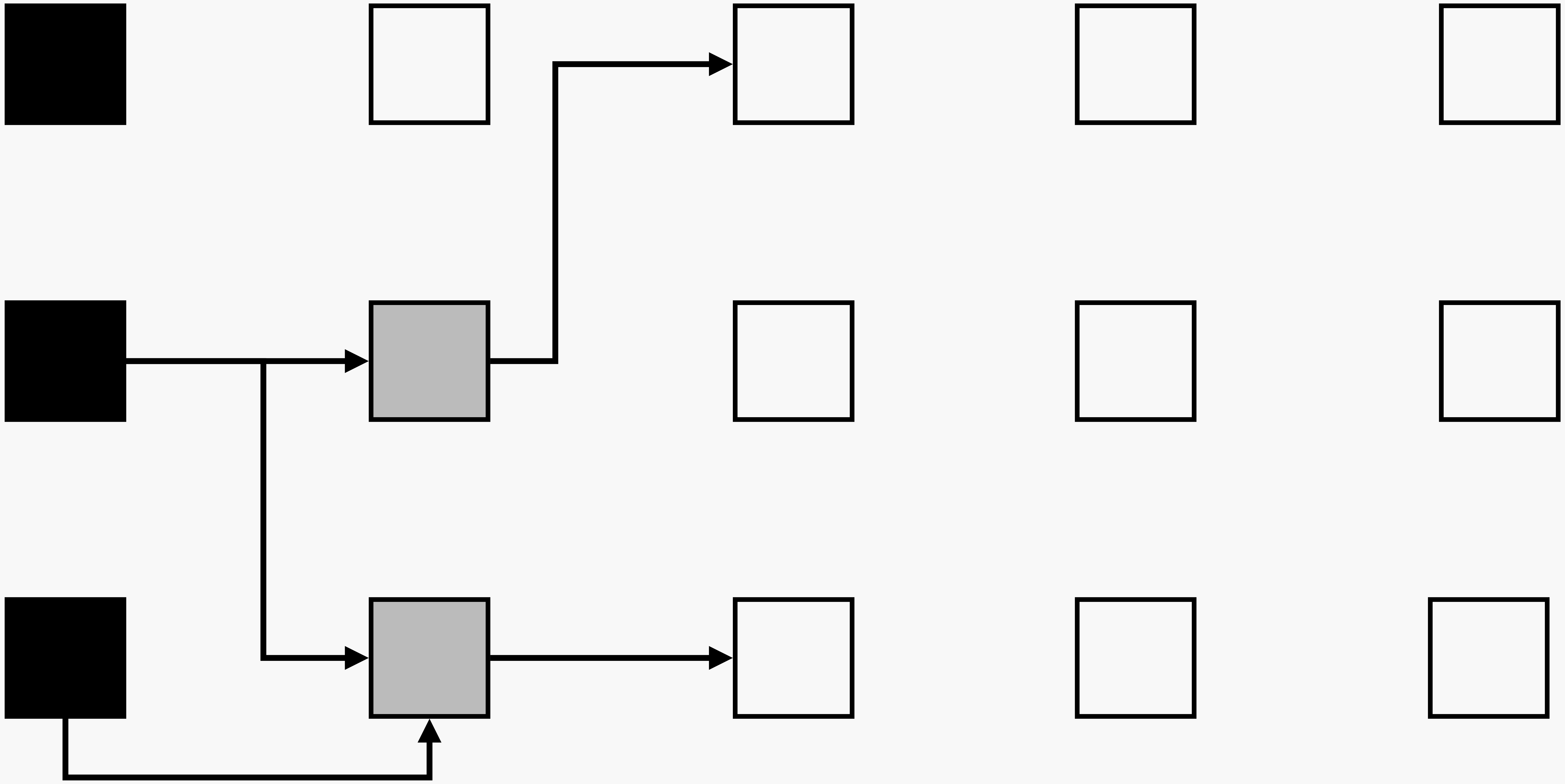
2014

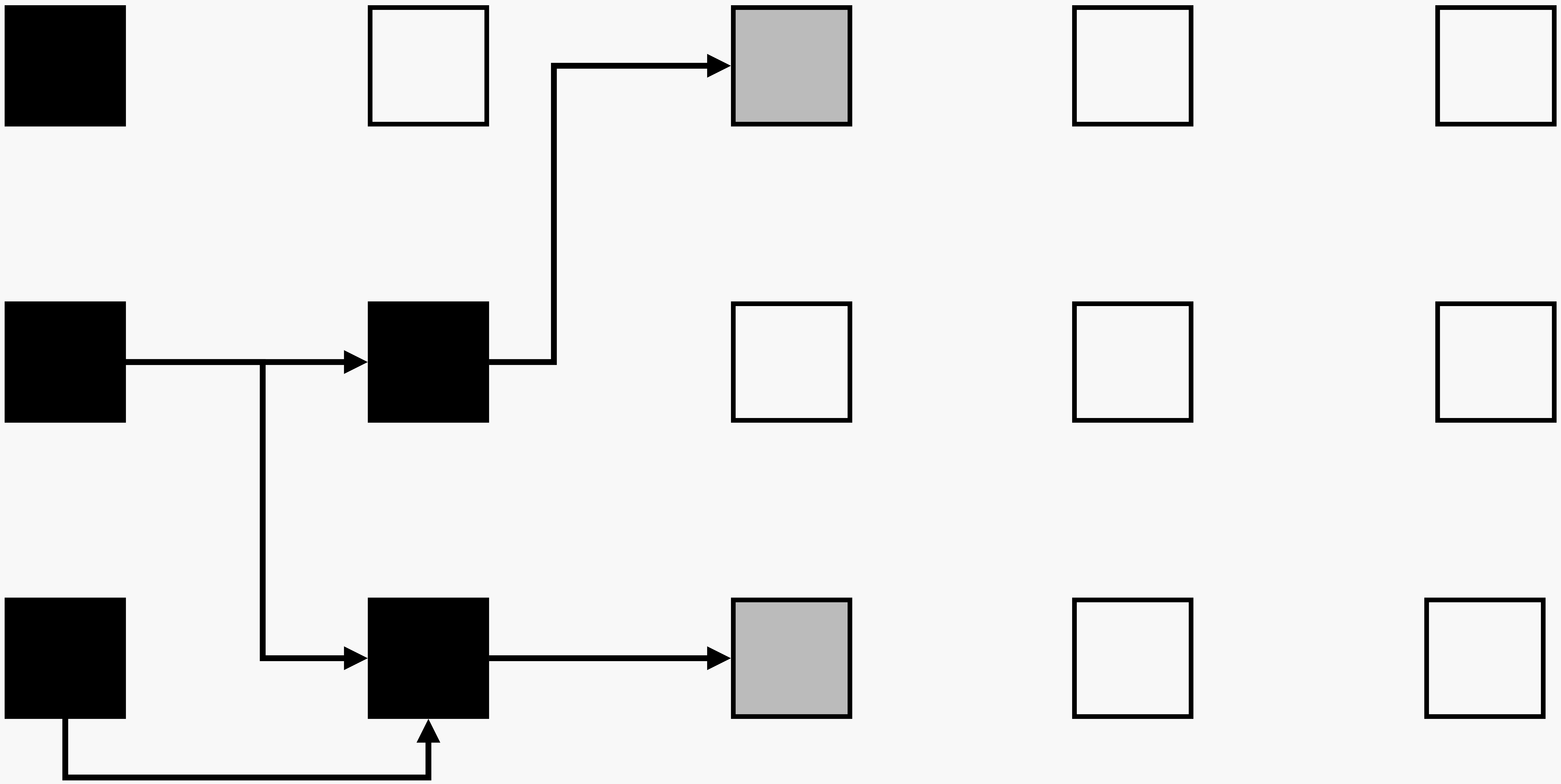
Ruby 2.2 introduced
Incremental Marking

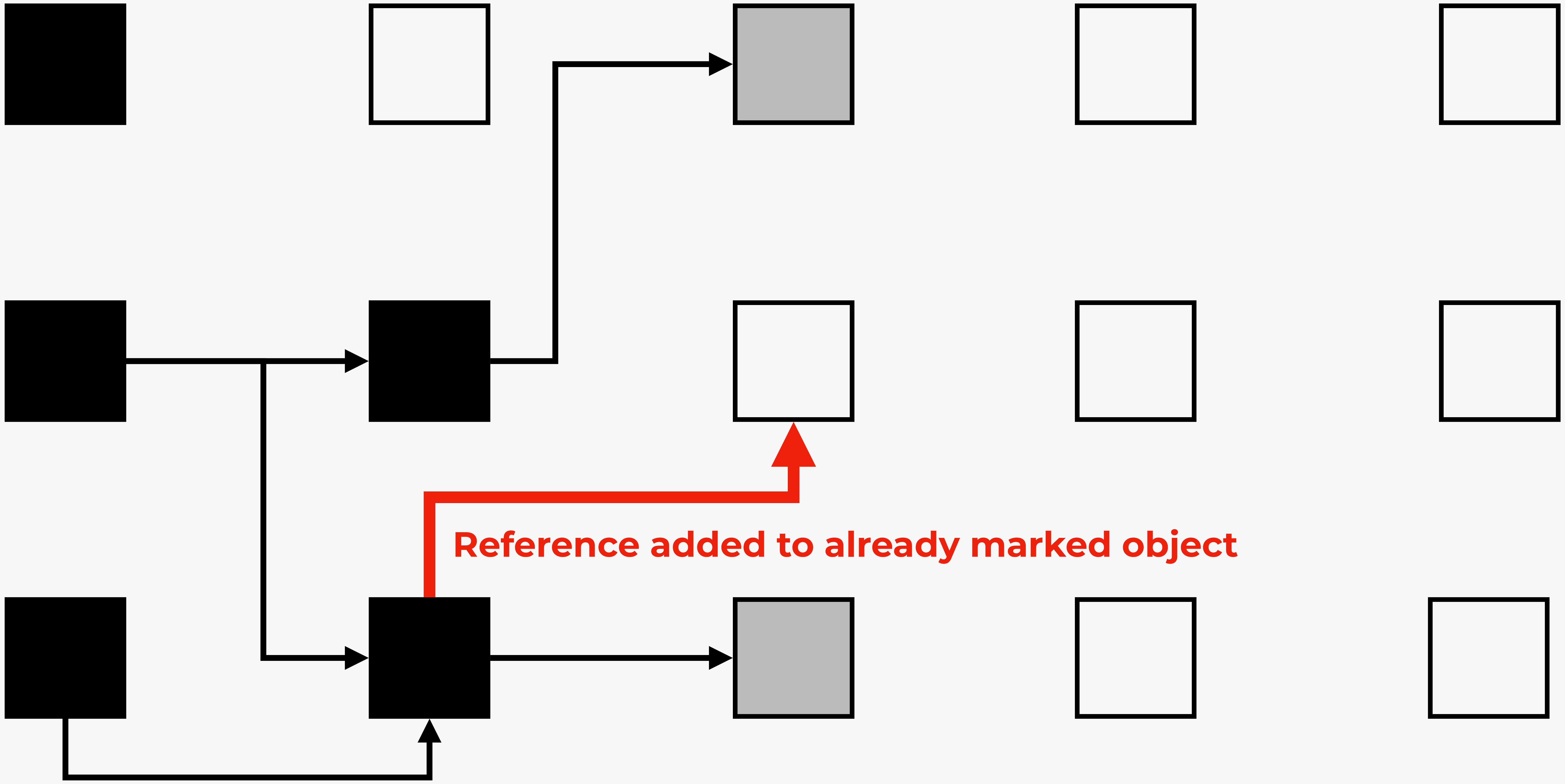

Time
Key:
Mutator
Minor Mark
Major Mark
Sweeping

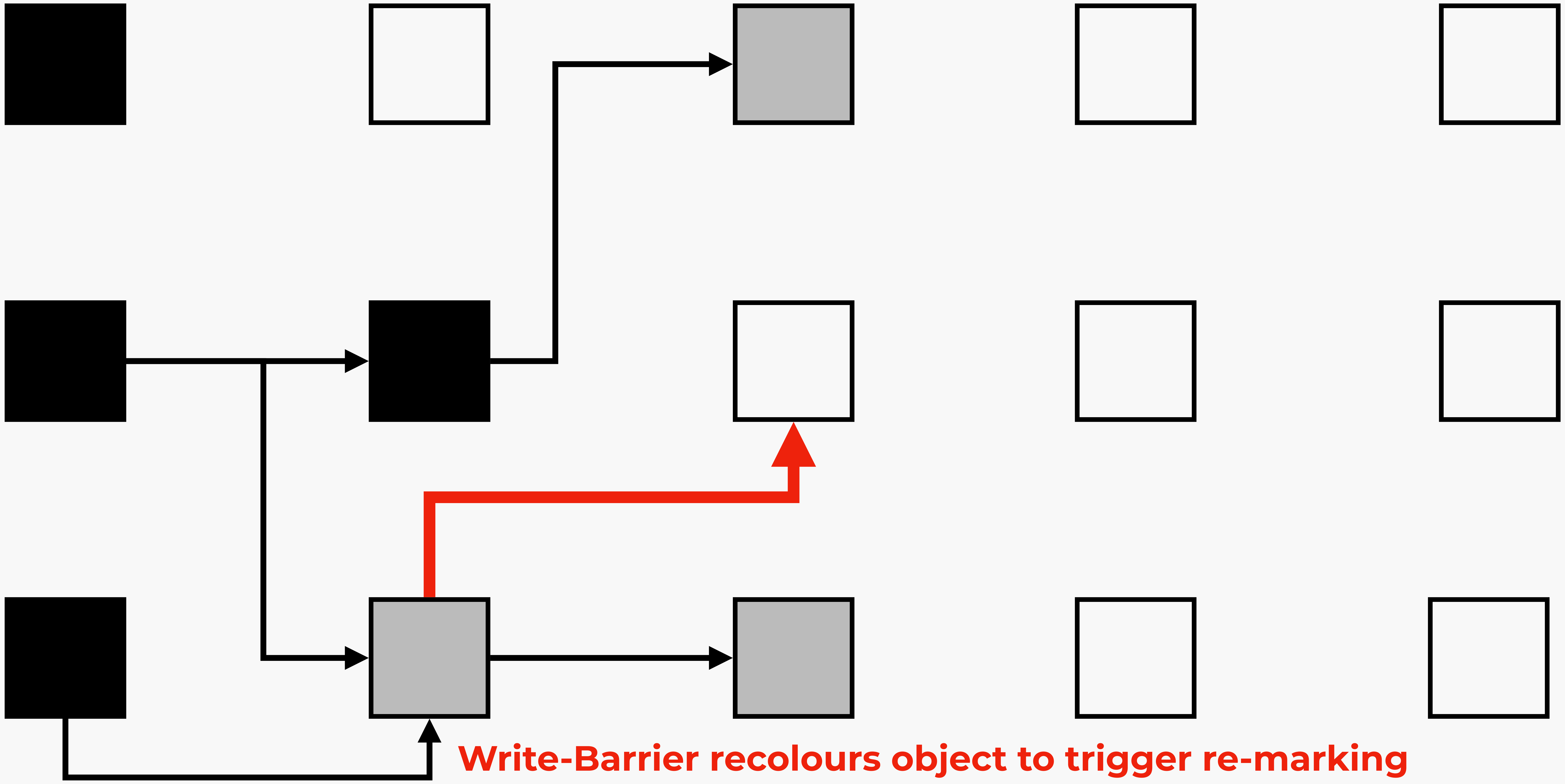









Reference added to already marked object


Write-Barrier recolours object to trigger re-marking

2019

Ruby 2.7 introduced
Compaction

First time Objects can move.

2-Finger compaction, from 1960's LISP.

Fits Ruby's memory layout.

First time Objects can move.

2-Finger compaction, from 1960's LISP.

Fits Ruby's memory layout.

First time Objects can move.

2-Finger compaction, from 1960's LISP.

Fits Ruby's memory layout.

Performance Improvements from:

- Smaller heap.
- Better locality.
- Better Copy-on-write performance.

Extension object pinned by default, explicit opt-in.

Performance Improvements from:

Smaller heap.

Better locality.

Better Copy-on-write performance.

Extension object pinned by default, explicit opt-in.

Performance Improvements from:

- Smaller heap.
- Better locality.
- Better Copy-on-write performance.

Extension object pinned by default, explicit opt-in.

```c
static void
cont_compact(void *ptr)
{
    rb_context_t *cont = ptr;

    if (cont->self) {
        cont->self = rb_gc_location(cont->self);
    }
    cont->value = rb_gc_location(cont->value);
    rb_execution_context_update(&cont->saved_ec);
}
```

2020

Ruby 3.0 introduced
Automatic Compaction

Ruby 2.7: Manual compaction, 3.0: Automatic compaction

Empty slots filled when swept.

Objects can be modified during sweeping.

Ruby 2.7: Manual compaction, 3.0: Automatic compaction

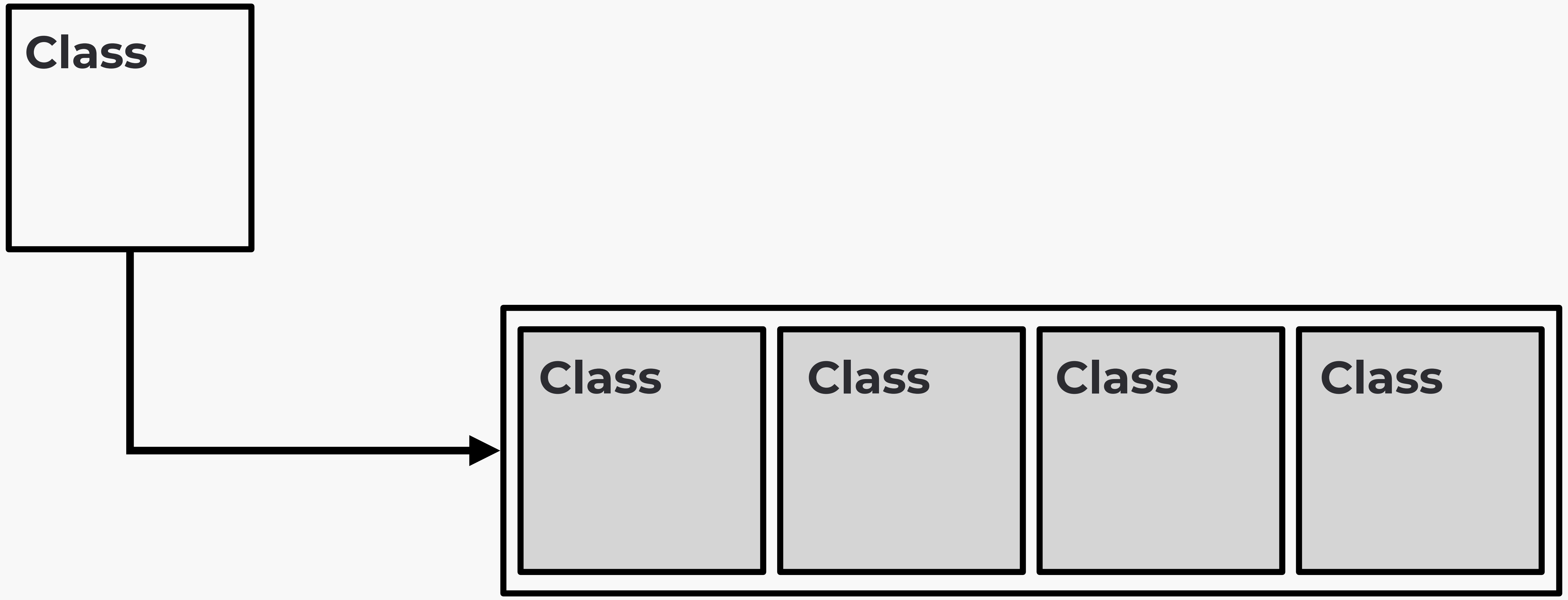
Empty slots filled when swept.

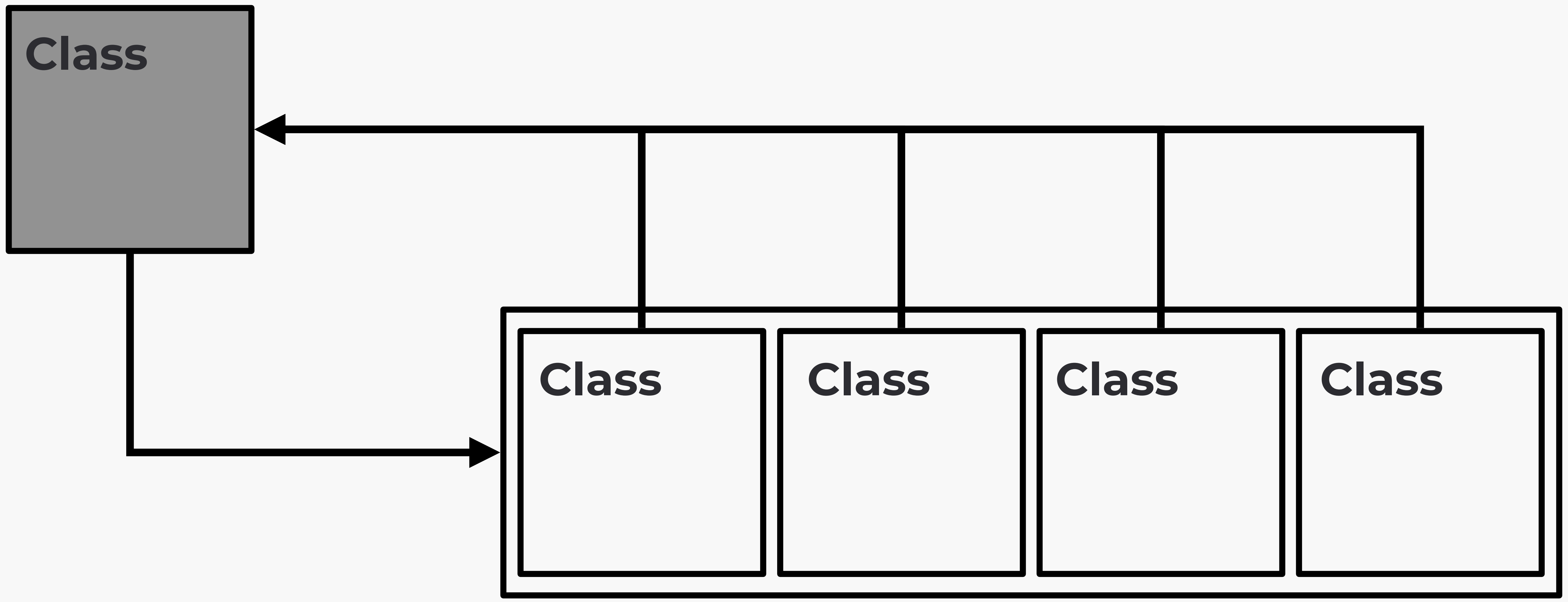
Objects can be modified during sweeping.

Ruby 2.7: Manual compaction, 3.0: Automatic compaction

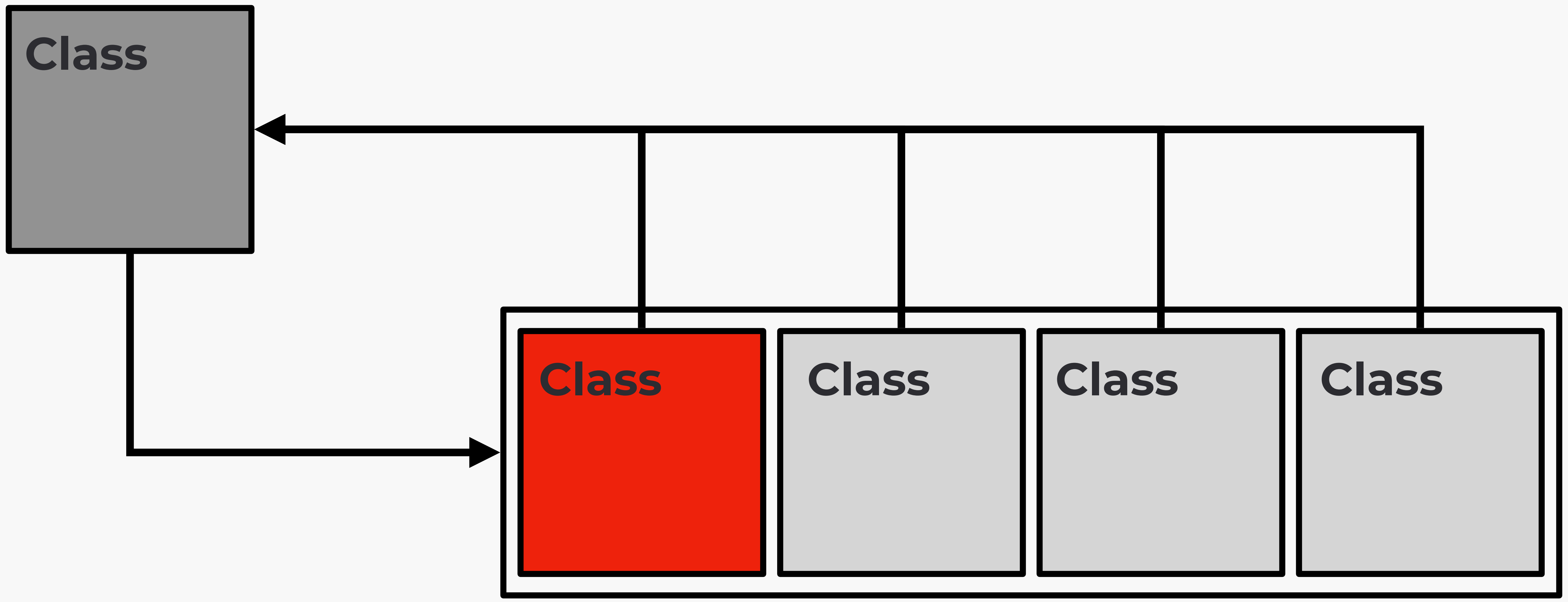
Empty slots filled when swept.

Objects can be modified during sweeping.

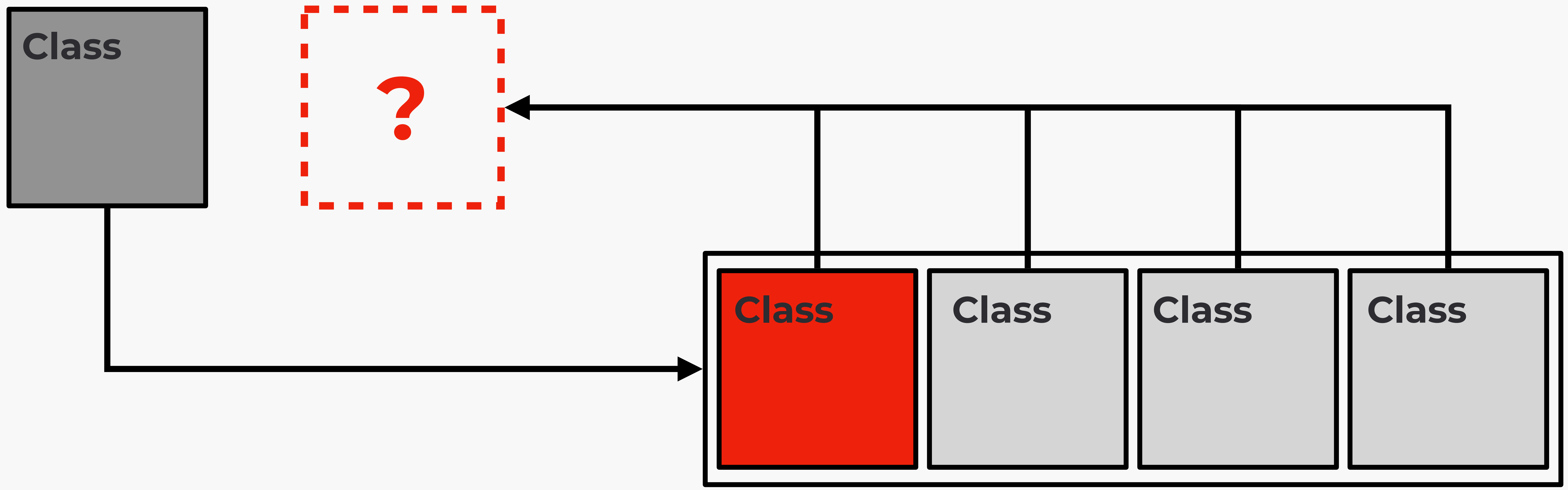

Class
Class
Class
Class
Class



Class
Class
Class
Class
Class



Class
Class
Class
Class
Class


Class
?
Class
Class
Class
Class

Assumptions made about GC that no longer held.

Solved with read barriers.

Auto-compaction reduced GC peformance.

Assumptions made about GC that no longer held.

Solved with read barriers.

Auto-compaction reduced GC peformance.

Assumptions made about GC that no longer held.

Solved with read barriers.

Auto-compaction reduced GC peformance.

2022

Ruby 3.2 introduced Variable Width Allocation

Optimizing Ruby's Memory Layout

Peter Zhu

Ruby Core Committer
Production Engineer, Shopify

Matt Valentine-House

Senior Developer, Shopify

shopify

Improved mutator performance.

Improved data locality.

fewer external allocations.

Improved mutator performance.

Improved data locality.

fewer external allocations.

Improved mutator performance.

Improved data locality.

fewer external allocations.

2023?

Ruby ships with an incremental, non-copying generation mark & sweep Garbage collector with optional compaction

–Matt Valentine-House
RubyKaigi 2023

Grown organically over 29 years.

Worked around assumptions, and made its own.

This is fine.

Grown organically over 29 years.

Worked around assumptions, and made its own.

This is fine.

Grown organically over 29 years.

Worked around assumptions, and made its own.

This is fine.

The alternative is a well worn path that starts down the easy road of reference counting or conservative GC and ends with a system that has a good compiler but is hamstrung by poor memory performance.

–Stephen Blackburn, 2011
Australian National University

Rubys GC is 30 years old.

Core algorithms are >70 years old.

~6% of the entire Ruby core codebase.

Very hard to change.

Weak abstraction boundaries.

Rubys GC is 30 years old.

Core algorithms are >70 years old.

~6% of the entire Ruby core codebase.

Very hard to change.

Weak abstraction boundaries.

Rubys GC is 30 years old.

Core algorithms are >70 years old.

~6% of the entire Ruby core codebase.

Very hard to change.

Weak abstraction boundaries.

Rubys GC is 30 years old.

Core algorithms are >70 years old.

~6% of the entire Ruby core codebase.

Very hard to change.

Weak abstraction boundaries.

Rubys GC is 30 years old.

Core algorithms are >70 years old.

~6% of the entire Ruby core codebase.

Very hard to change.

Weak abstraction boundaries.

“Immix: A Mark-Region Garbage Collector with Space Efficiency, Fast Collection, and Mutator Performance”

Stephen M. Blackburn
Australian National University

Kathryn S. McKinley
The University of Texas at Austin

presented at the ACM SIGPLAN Conference on Programming Language Design and Implementation: PLDI 2008

Allocation	Identification	Reclaimation

Allocation	Identification	Reclamation
Free-List		

Allocation	Identification	Reclaimation
Free-List		
Bump Pointer		

Allocation	Identification	Reclaimation
Free-List	**Tracing (Implicit)**	
Bump Pointer		

Allocation	**Identification**	**Reclaimation**
Free-List	Tracing (Implicit)	
Bump Pointer	**Reference Counting (Explicit)**	

Allocation	Identification	Reclaimation
Free-List	Tracing (Implicit)	**Sweep**
Bump Pointer	Reference Counting (Explicit)	

Allocation	**Identification**	**Reclaimation**
Free-List	Tracing (Implicit)	Sweep
		Evacuation
Bump Pointer	Reference Counting (Explicit)	

Allocation	Identification	Reclaimation
Free-List	Tracing (Implicit)	Sweep
		Evacuation
Bump Pointer	Reference Counting (Explicit)	**Compaction**

Allocation	Identification	Reclaimation
Free-List	Tracing (Implicit)	Sweep
		Evacuation
Bump Pointer	Reference Counting (Explicit)	Compaction

Mark & Sweep

Allocation	Identification	Reclaimation
Free-List	Tracing (Implicit)	Sweep
Bump Pointer	Reference Counting (Explicit)	Evacuation
		Compaction

Mark-Compact

Allocation	Identification	Reclamation
Free-List	Tracing (Implicit)	Sweep
		Evacuation
Bump Pointer	Reference Counting (Explicit)	Compaction

Semi-Space

Each algorithm has different performance characteristics.

Modern GC's: G1, Shenandoah, ZGC et al. are composed from canonical algorithms.

Combined with generations, concurrency and parallelism to achieve high performance.

Each algorithm has different performance characteristics.

Modern GC's: G1, Shenandoah, ZGC et al. are composed from canonical algorithms.

Combined with generations, concurrency and parallelism to achieve high performance.

Each algorithm has different performance characteristics.

Modern GC's: G1, Shenandoah, ZGC et al. are composed from canonical algorithms.

Combined with generations, concurrency and parallelism to achieve high performance.

Immix was a new canonical collector.

It formalised Mark-Region as a category of collectors.

Outperforms existing canonical collectors by 7-25% on average.

Immix was a new canonical collector.

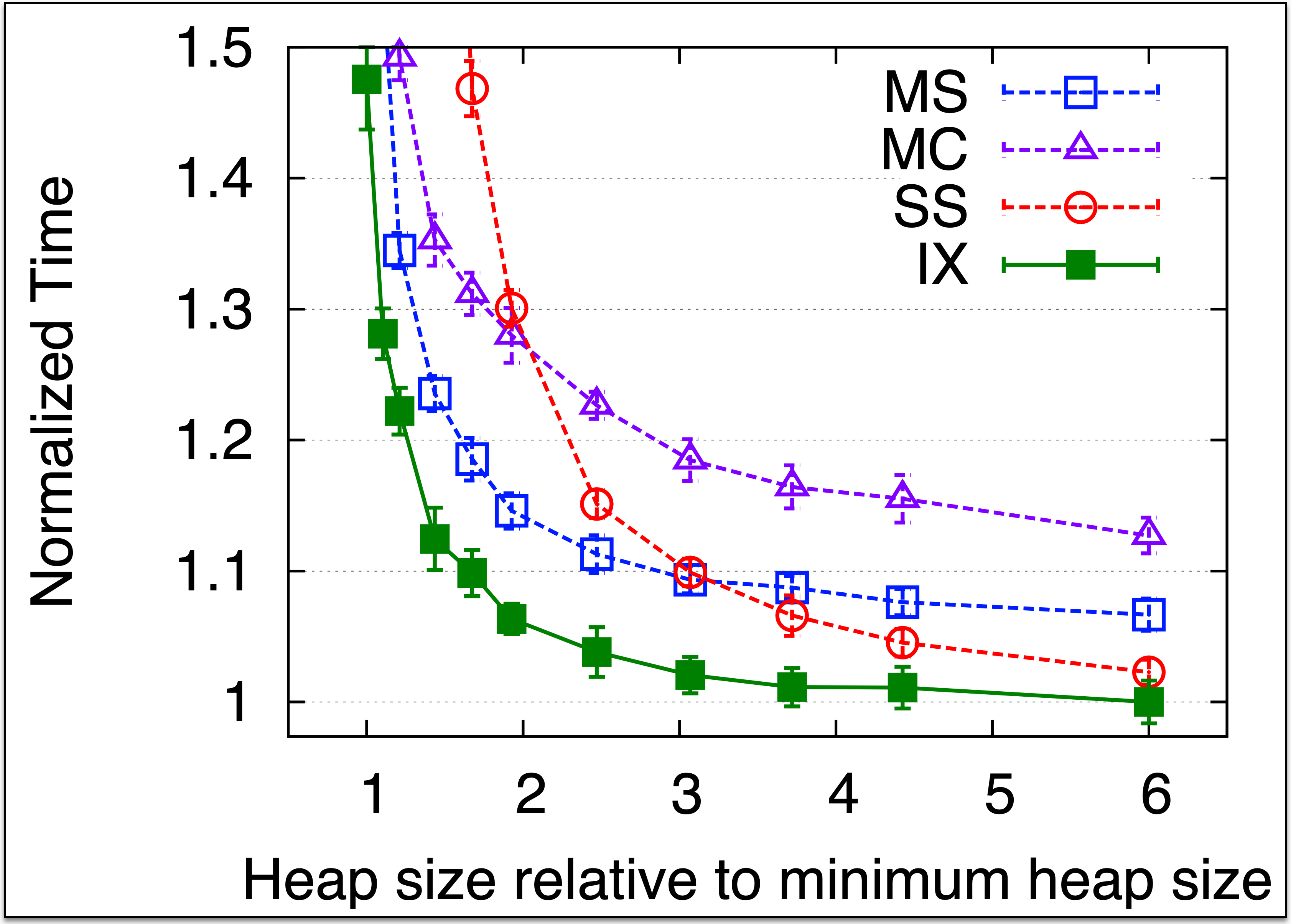
It formalised Mark-Region as a category of collectors.

Outperforms existing canonical collectors by 7-25% on average.

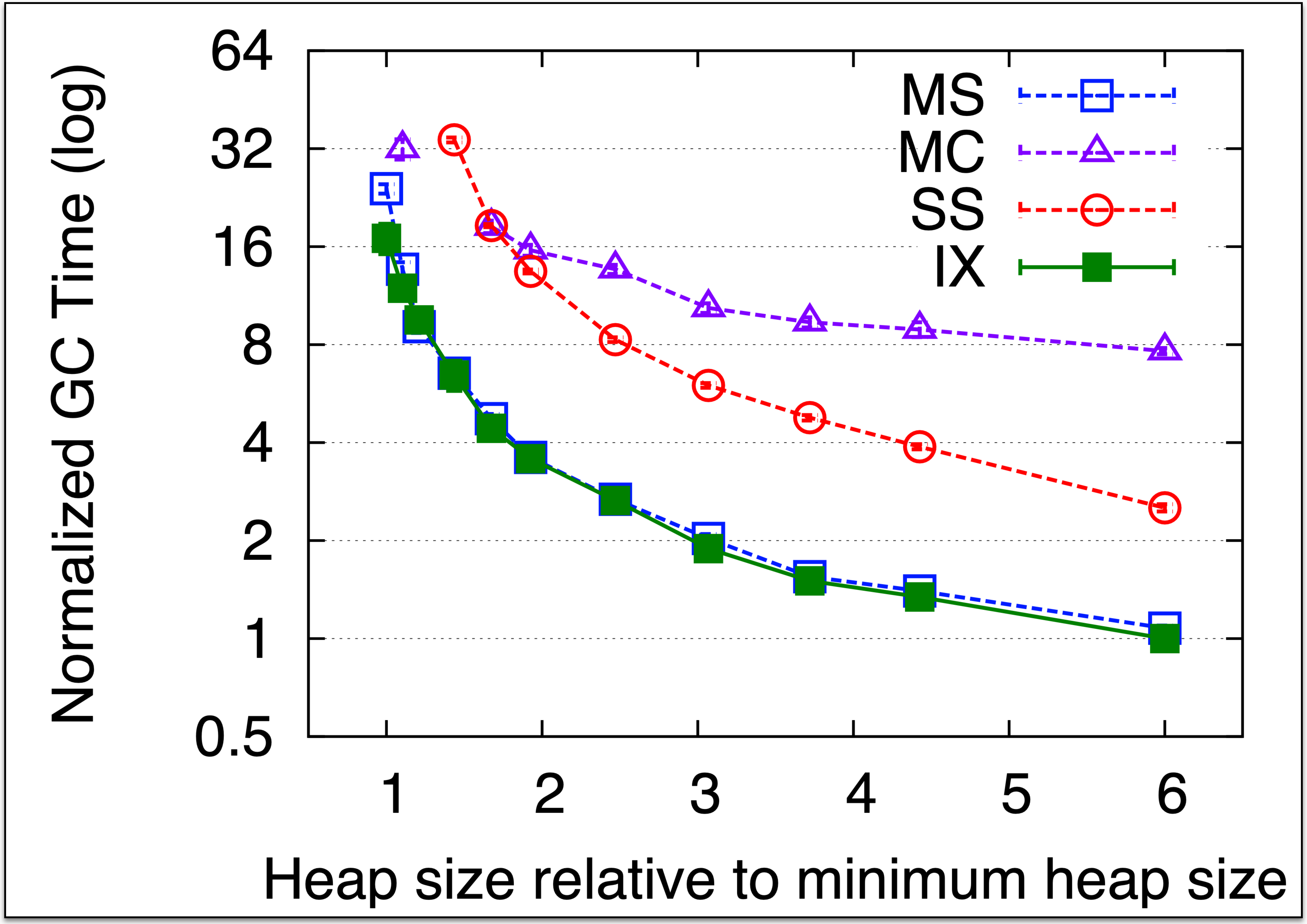
Immix was a new canonical collector.

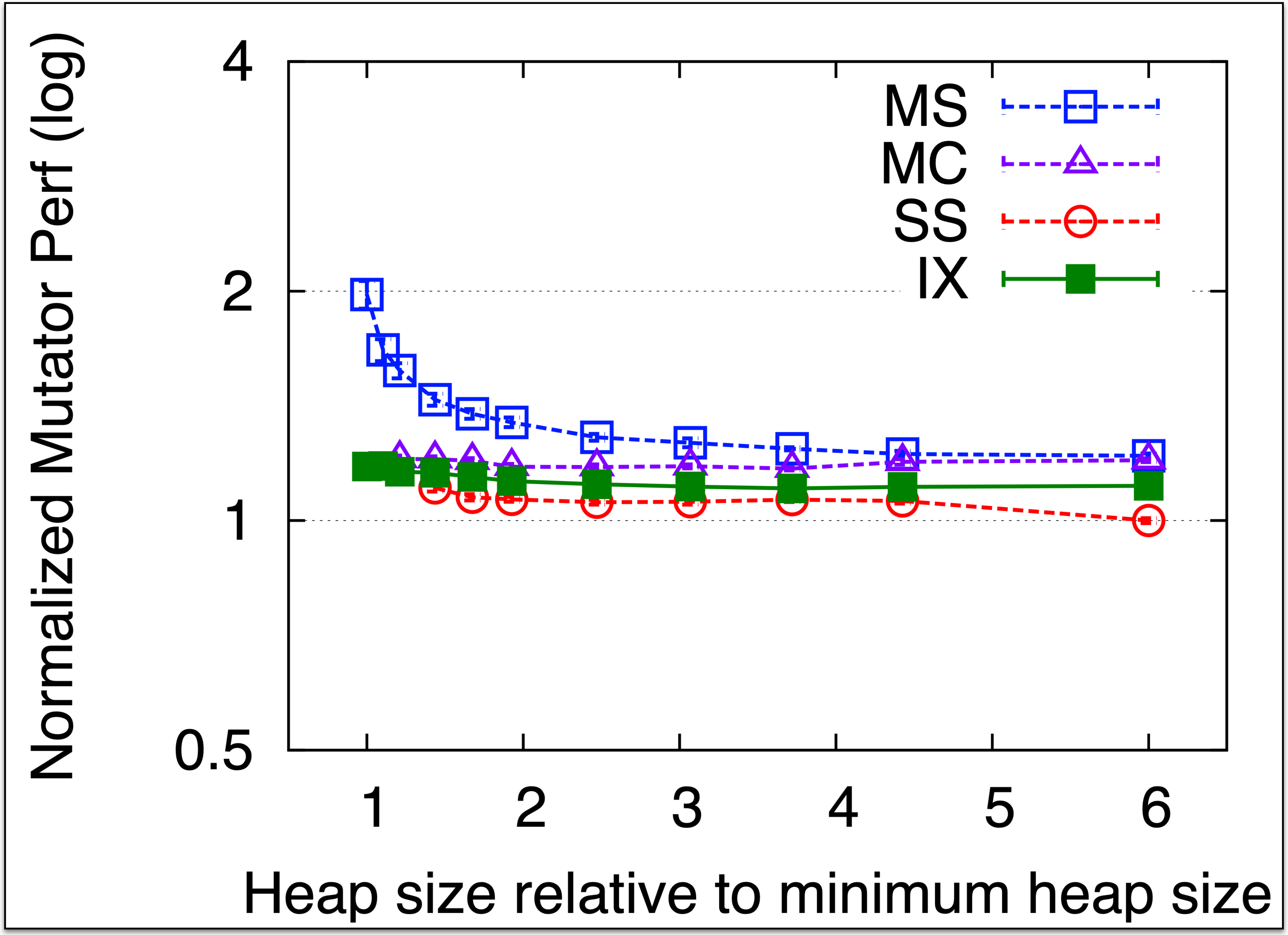
It formalised Mark-Region as a category of collectors.

Outperforms existing canonical collectors by 7-25% on average.



Normalized Time
1.5
1.4
1.3
1.2
1.1
1
1
2
3
4
5
6
Heap size relative to minimum heap size
MS
MC
SS
IX


Normalized GC Time (log)
64
32
16
8
4
2
1
0.5
MS
MC
SS
IX
1
2
3
4
5
6
Heap size relative to minimum heap size


Normalized Mutator Perf (log)
4
2
1
0.5
MS
MC
SS
IX
1
2
3
4
5
6
Heap size relative to minimum heap size

Inko, and Scala Native

Glasgow Haskell Compiler (GHC).

Rubinius

Inko, and Scala Native

Glasgow Haskell Compiler (GHC).

Rubinius

Inko, and Scala Native

Glasgow Haskell Compiler (GHC).

Rubinius

“Low-Latency, High-Throughput Garbage Collection”

Wenyu Zhao
Australian National University

Stephen M. Blackburn
Australian National University

Kathryn S. McKinley
Google

presented at the ACM SIGPLAN Conference on Programming Language Design and Implementation: PLDI 2022

LXR is a new high-level GC algorithm.

Adds reference counting, and heavy optimisations, to Immix.

Significantly outperforms high-profile production GC's

LXR is a new high-level GC algorithm.

Adds reference counting, and heavy optimisations, to Immix.

Significantly outperforms high-profile production GC's

LXR is a new high-level GC algorithm.

Adds reference counting, and heavy optimisations, to Immix.

Significantly outperforms high-profile production GC's

Geometric Mean of 99.99% Latency, and throughput relative to G1, for 4 collectors in three heap sizes

	99.99% Latency/G1				Time/G1			
Heap size	G1	LXR	Shen	ZGC	G1	LXR	Shen	ZGC
1.3x	1.00	0.72	1.51	-	1.00	0.97	1.77	-
2x	1.00	0.92	2.54	-	1.00	0.96	0.96	-
6x	1.00	0.85	1.41	1.44	1.00	1.01	1.09	1.26

Geometric Mean of 99.99% Latency, and throughput relative to G1, for 4 collectors in three heap sizes

	99.99% Latency/G1				Time/G1			
Heap size	G1	LXR	Shen	ZGC	G1	LXR	Shen	ZGC
1.3x	1.00	0.72	1.51	-	1.00	0.97	1.77	-
2x	1.00	0.92	2.54	-	1.00	0.96	0.96	-
6x	1.00	0.85	1.41	1.44	1.00	1.01	1.09	1.26

Geometric Mean of 99.99% Latency, and throughput relative to G1, for 4 collectors in three heap sizes

	99.99% Latency/G1				Time/G1			
Heap size	G1	LXR	Shen	ZGC	G1	LXR	Shen	ZGC
1.3x	1.00	**0.72**	1.51	-	1.00	**0.97**	1.77	-
2x	1.00	**0.92**	2.54	-	1.00	**0.96**	0.96	-
6x	1.00	**0.85**	1.41	1.44	**1.00**	1.01	1.09	1.26

mmtk

MEMORY MANAGEMENT TOOLKIT

MMTk - Part of JikesRVM, from 2004.

Modular design with clear abstractions.

Rust rewrite in 2017, to be runtime agnostic.

MMTk - Part of JikesRVM, from 2004.

Modular design with clear abstractions.

Rust rewrite in 2017, to be runtime agnostic.

MMTk - Part of JikesRVM, from 2004.

Modular design with clear abstractions.

Rust rewrite in 2017, to be runtime agnostic.

Current	In-Progress
OpenJDK	
V8	
JikesRVM	

Current	In-Progress
OpenJDK	
V8	
JikesRVM	

Current	In-Progress
OpenJDK	
V8	
JikesRVM	

Current	In-Progress
OpenJDK	**Julia**
V8	
JikesRVM	

Current	In-Progress
OpenJDK	Julia
V8	**GHC**
JikesRVM	

Current	In-Progress
OpenJDK	Julia
V8	GHC
JikesRVM	**Ruby**


How Can An Existing Language
(semi-)Automatically Sped
Laurence Tratt
https://tratt.net/laurie/
2022-04-21
Morning: Laurence Tratt
Stefan Marr
Mingle a bit (Hello, Gui!)
12pm: light lunch
3pm: tea

Shopify Invests in Research for Ruby at Scale

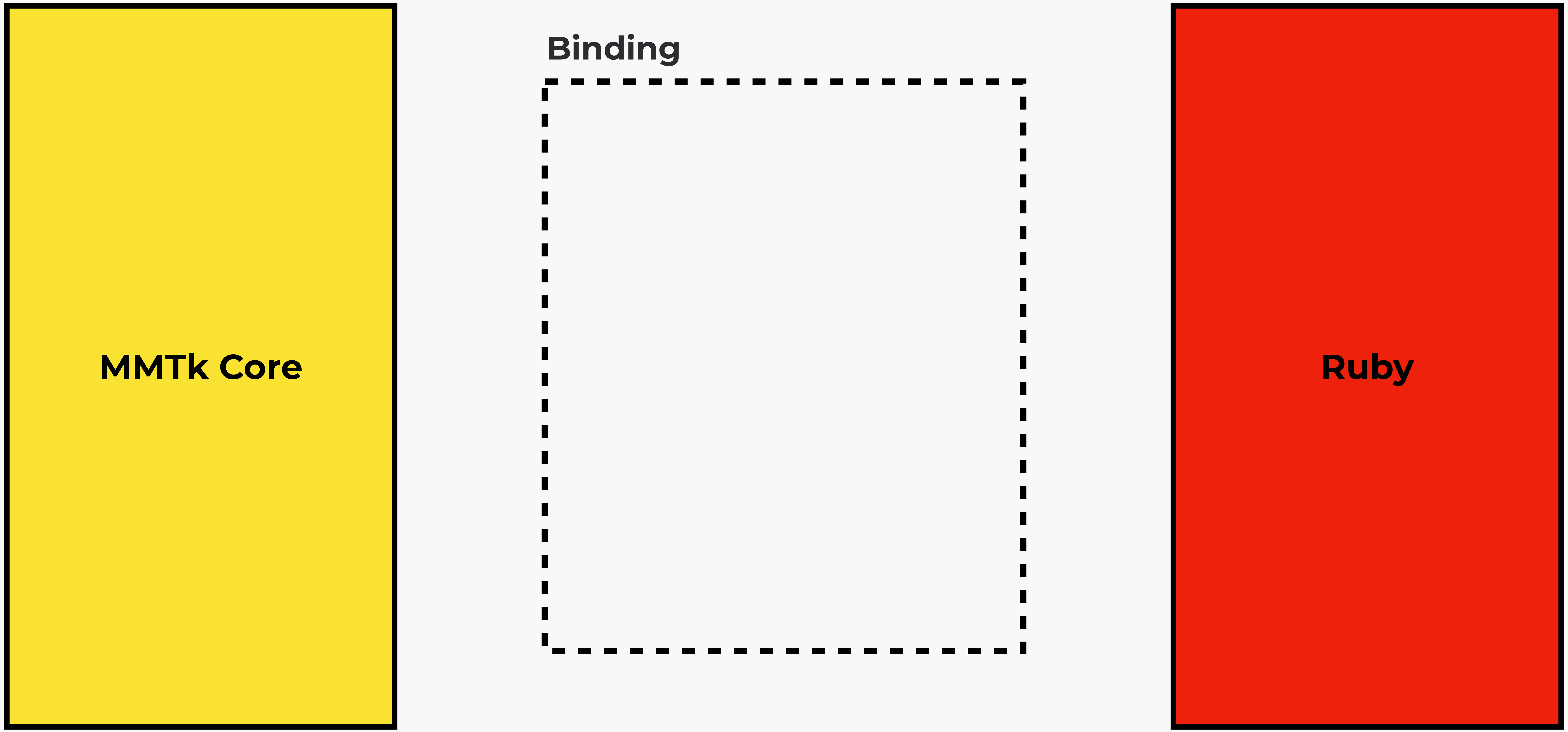
by Chris Seaton • Development
May 16, 2022 • 4 minute read

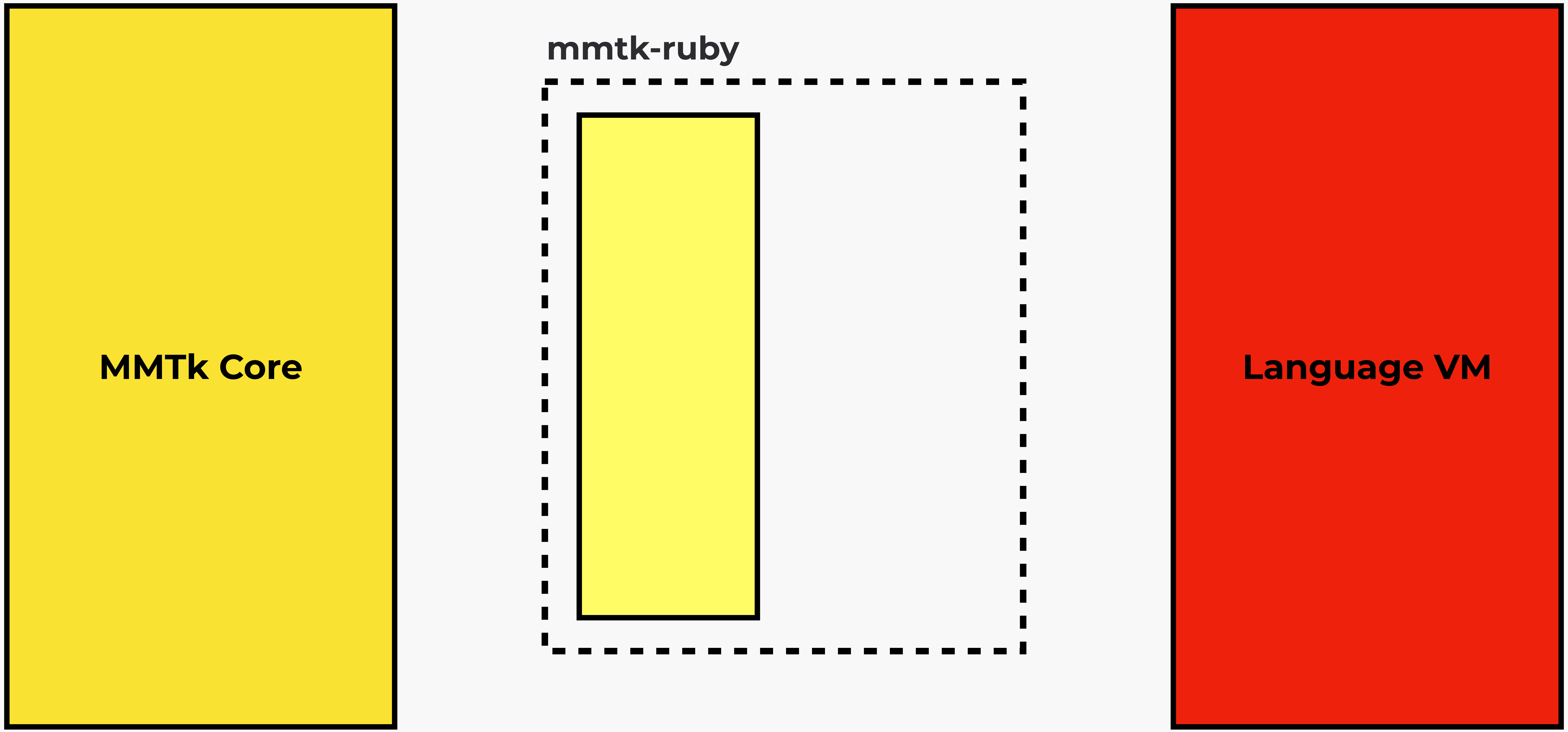
shopify

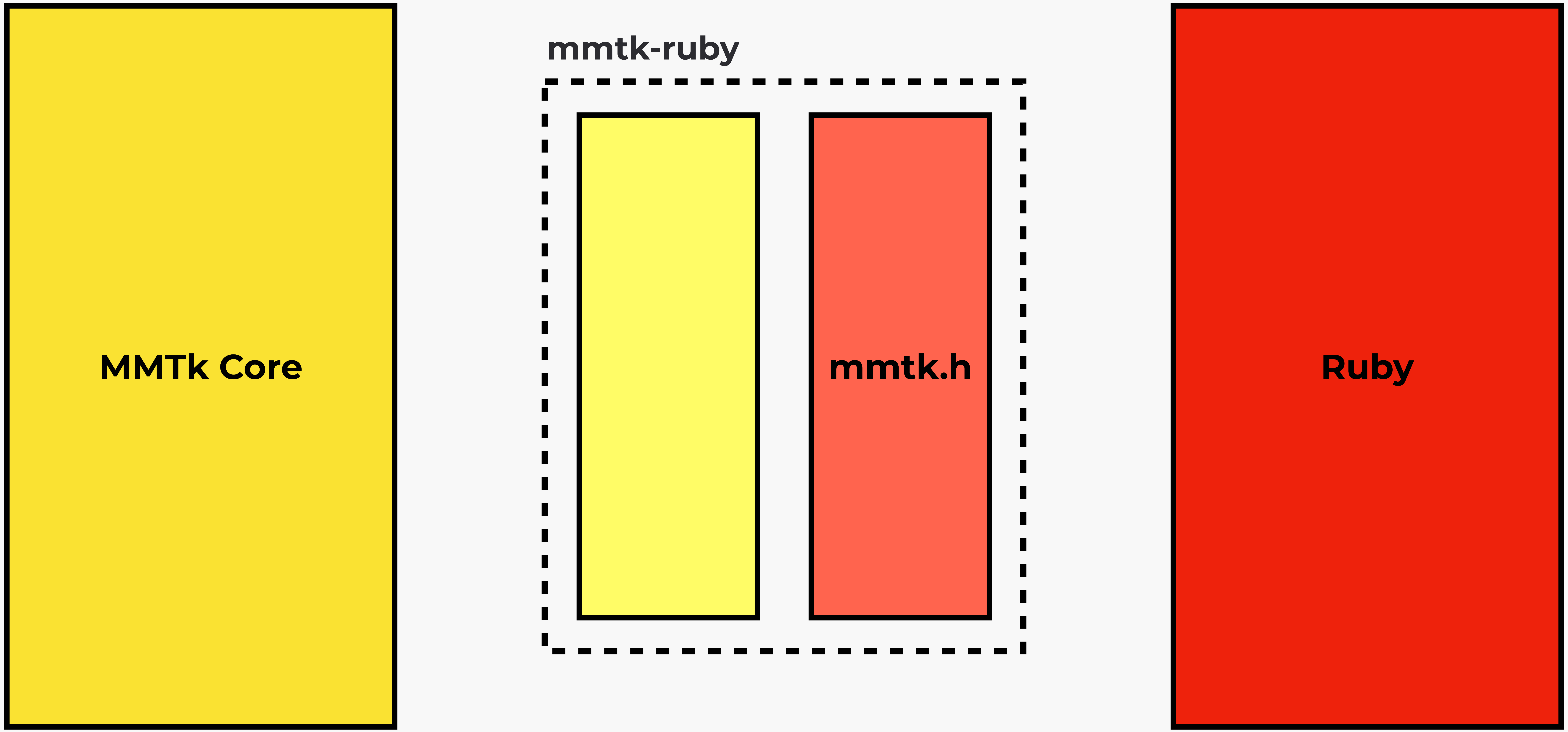
HUAWEI

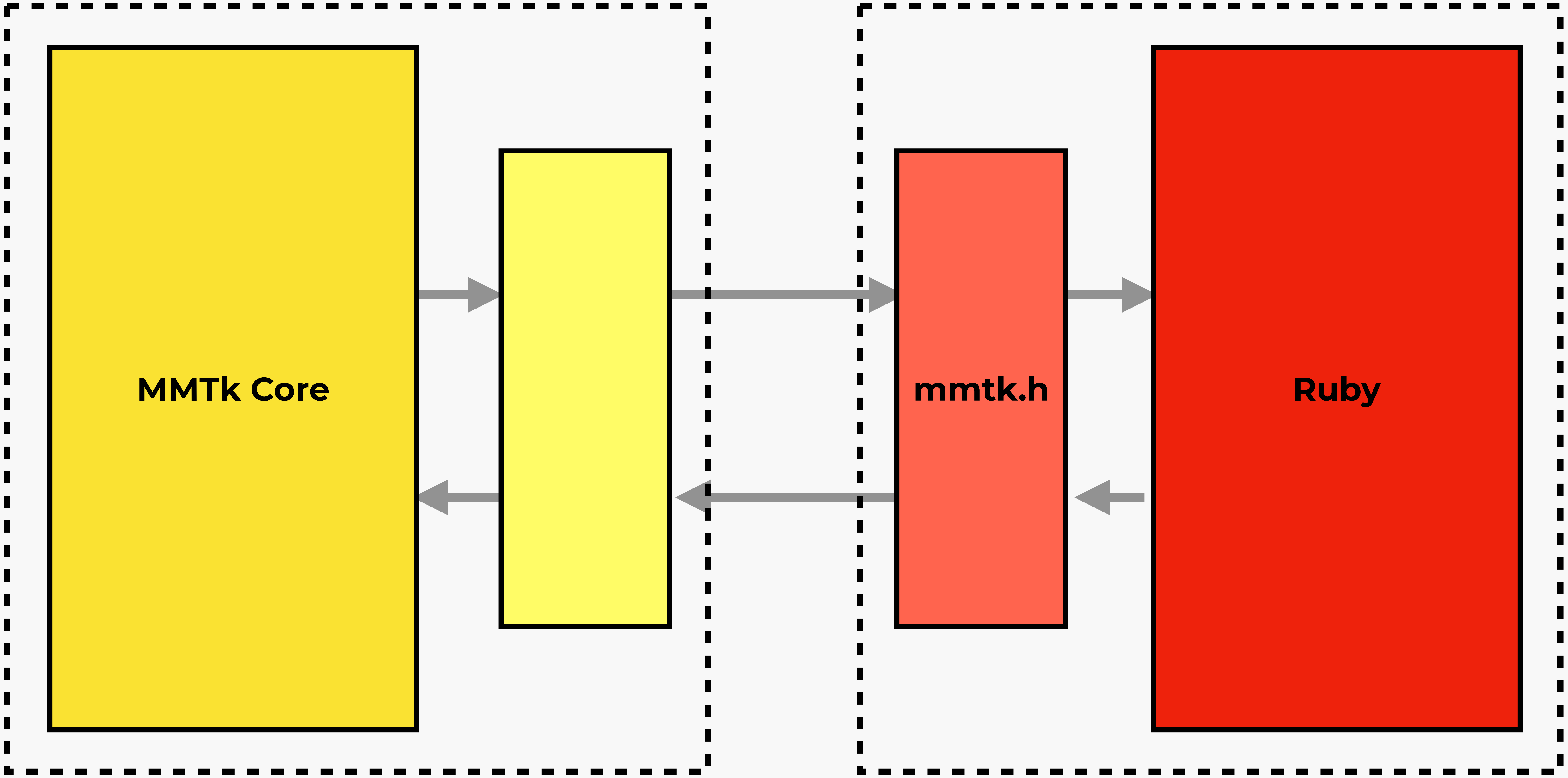
Google


MMTk Core
Ruby


MMTk Core
Binding
Ruby


MMTk Core
mmtk-ruby
Language VM


MMTk Core
mmtk-ruby
mmtk.h
Ruby


MMTk Core
mmtk.h
Ruby

Shopify/ruby-mmtk-builder

```
>
```

>

>

```
>
```

Not production ready.

Linux only, Mac soon, Windows unknown.

MarkSweep runs Rails, Immix does not.

Performance.

Not production ready.

Linux only, Mac soon, Windows unknown.

MarkSweep runs Rails, Immix does not.

Performance.

Not production ready.

Linux only, Mac soon, Windows unknown.

MarkSweep runs Rails, Immix does not.

Performance.

Not production ready.

Linux only, Mac soon, Windows unknown.

MarkSweep runs Rails, Immix does not.

Performance.

Where do we go from here?

```
#if USE_MMTK
    if (rb_mmtk_enabled_p()) {
        // When using MMTk, we pass the observed
        // ID directly as the `obj` parameter.
        saved.objid = obj;
    } else {
#endif
    saved.objid = rb_obj_id(obj);
#if USE_MMTK
    }
#endif
```

📈Positives	📉Negatives
	Complex, error-prone code

📈Positives	📉Negatives
	Complex, error-prone code
	Makes current Ruby GC harder to change.

📈 Positives	📉 Negatives
Allowed us to get up and running quickly	Complex, error-prone code
	Makes current Ruby GC harder to change.

📈Positives	📉Negatives
Allowed us to get up and running quickly	Complex, error-prone code
Helped us to discover areas that we need to change	Makes current Ruby GC harder to change.

V8 & GHC built a GC Interface.

Same interface used by internal GC & MMTk.

We are facing this choice currently in Ruby.

What if we didn't stop with MMTk?

V8 & GHC built a GC Interface.

Same interface used by internal GC & MMTk.

We are facing this choice currently in Ruby.

What if we didn't stop with MMTk?

V8 & GHC built a GC Interface.

Same interface used by internal GC & MMTk.

We are facing this choice currently in Ruby.

What if we didn't stop with MMTk?

V8 & GHC built a GC Interface.

Same interface used by internal GC & MMTk.

We are facing this choice currently in Ruby.

What if we didn't stop with MMTk?

Can we build a generic memory management interface for Ruby?

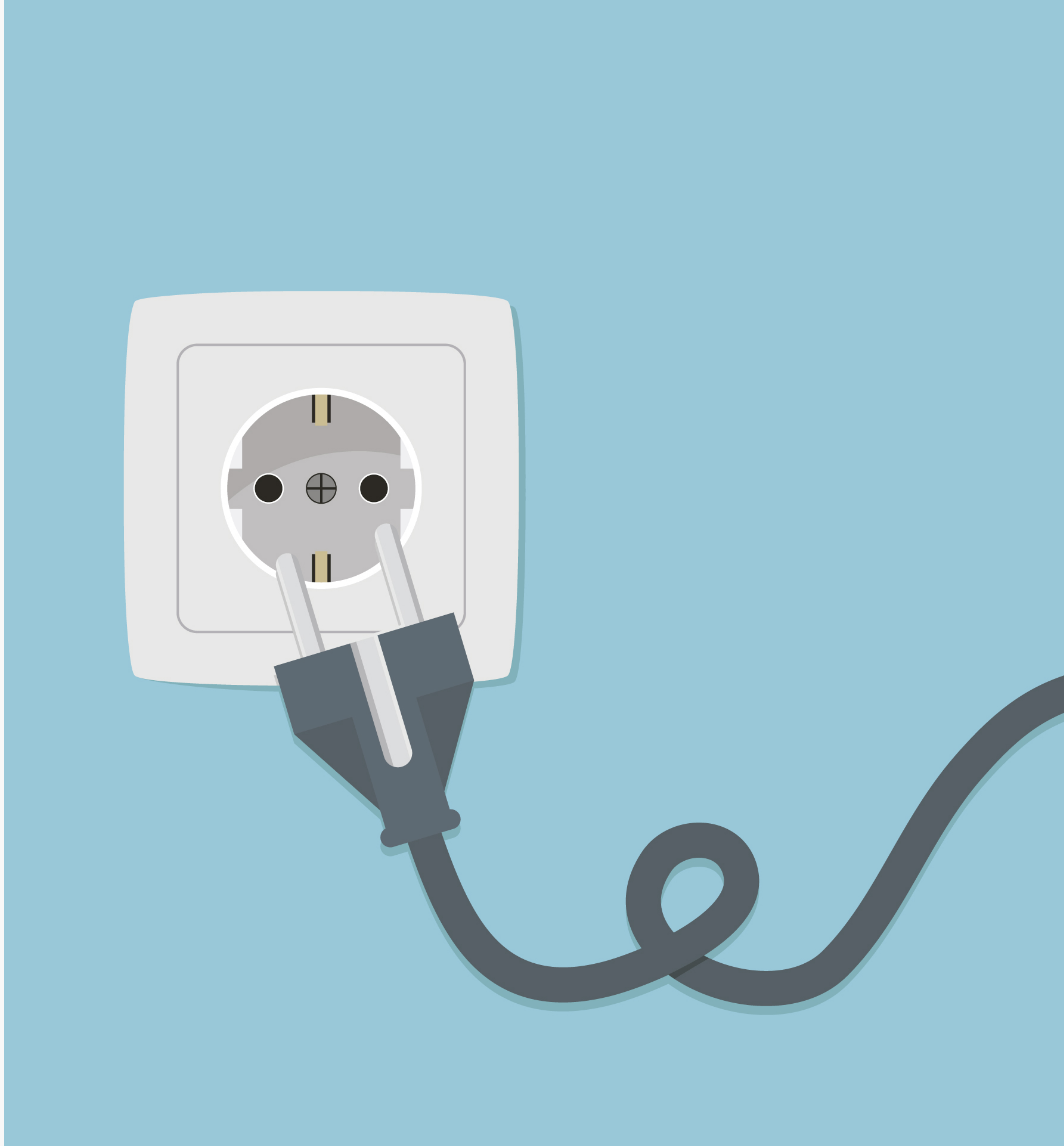
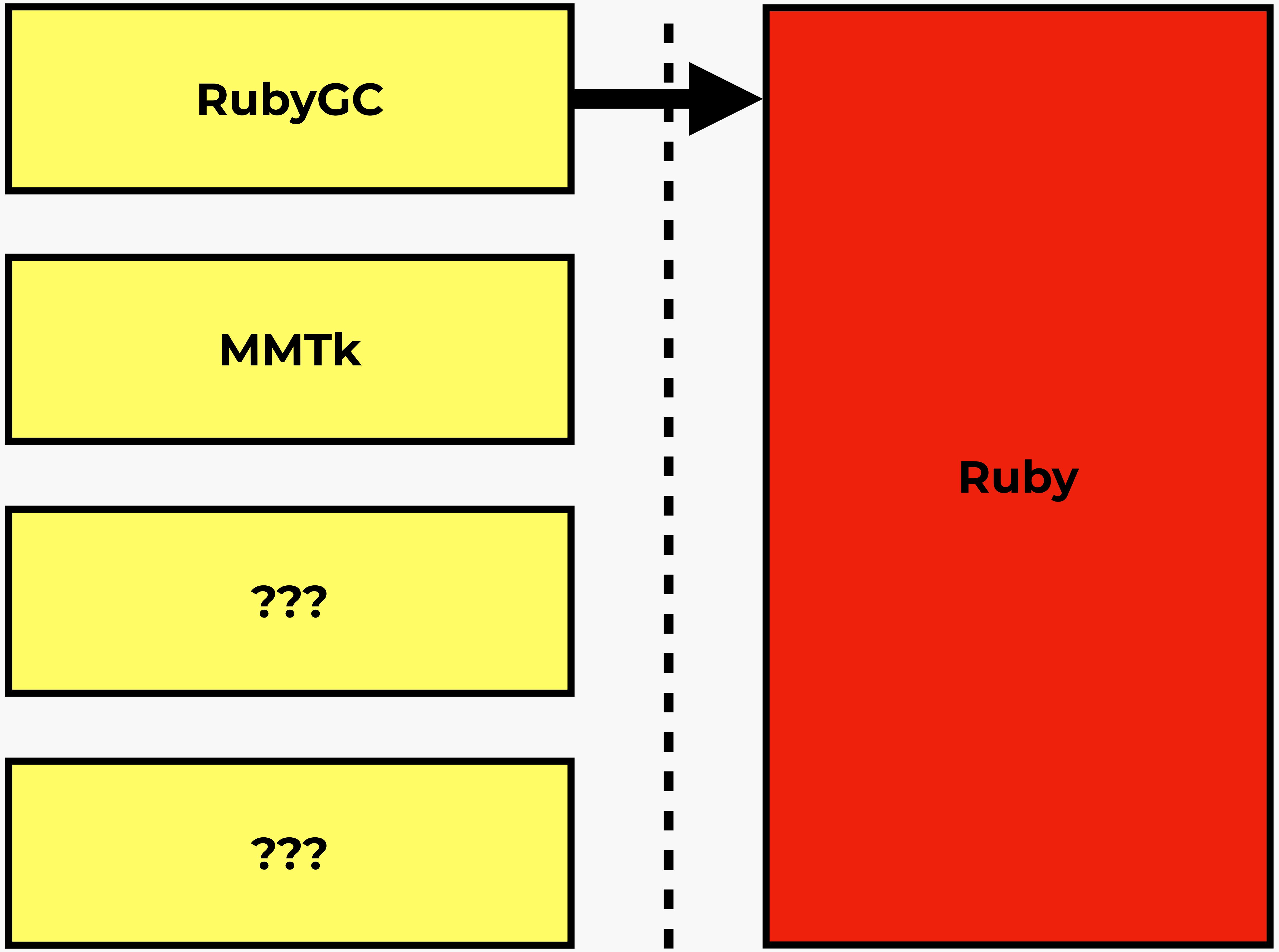
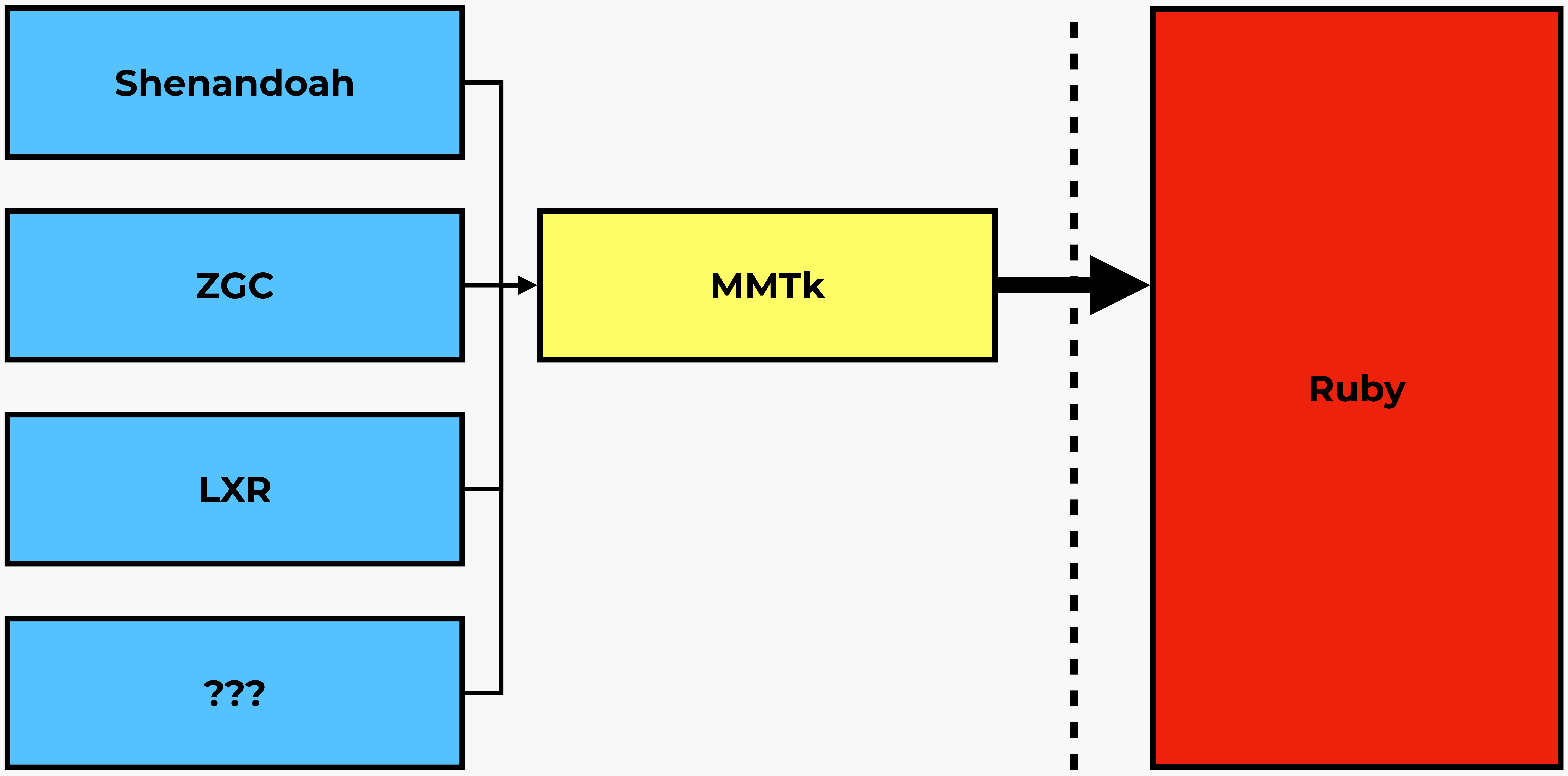


Image by photoroyalty on Freepik

📈Benefits	📉Questions
GC library	

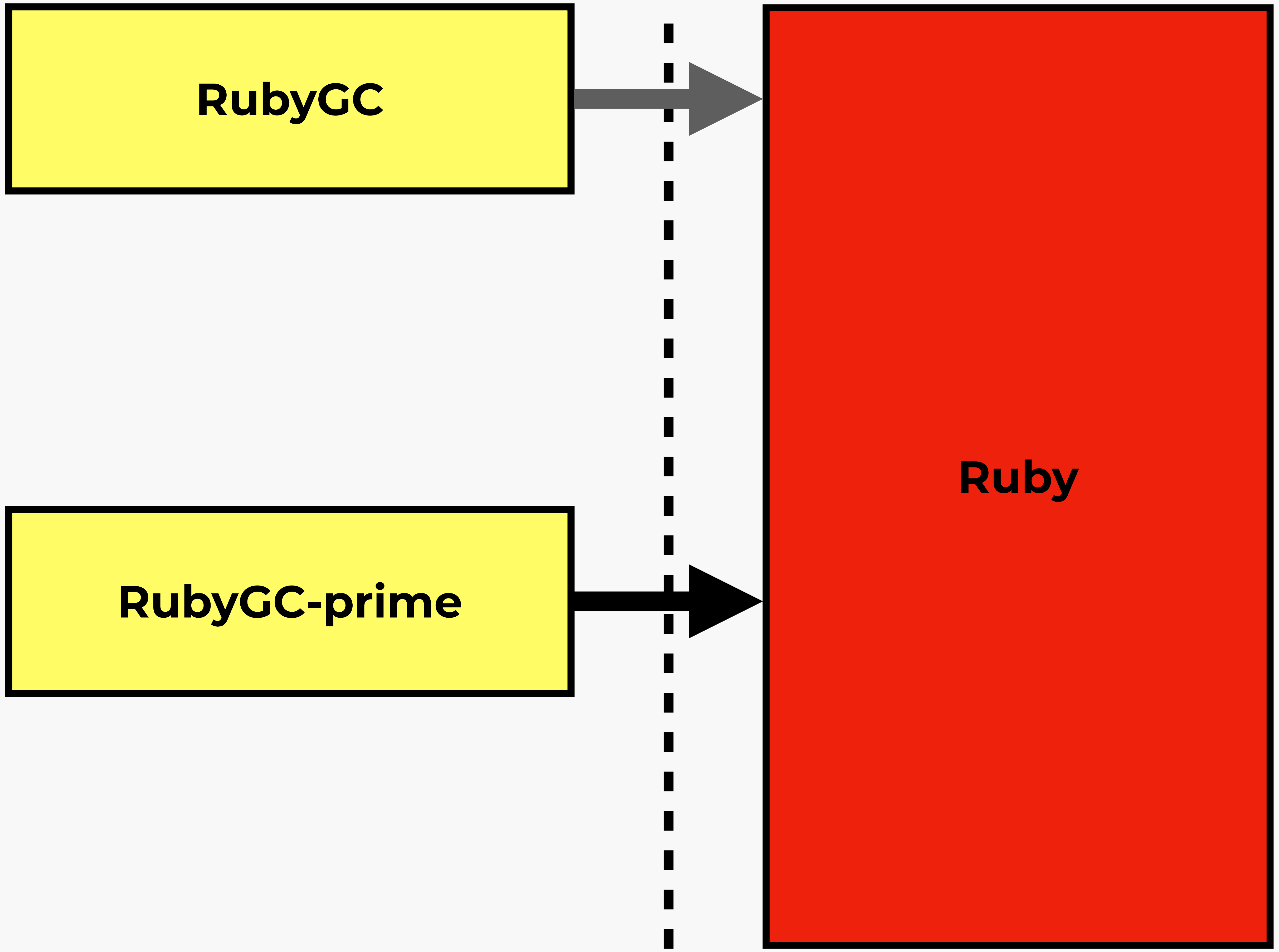

RubyGC
MMTk
???
???
Ruby

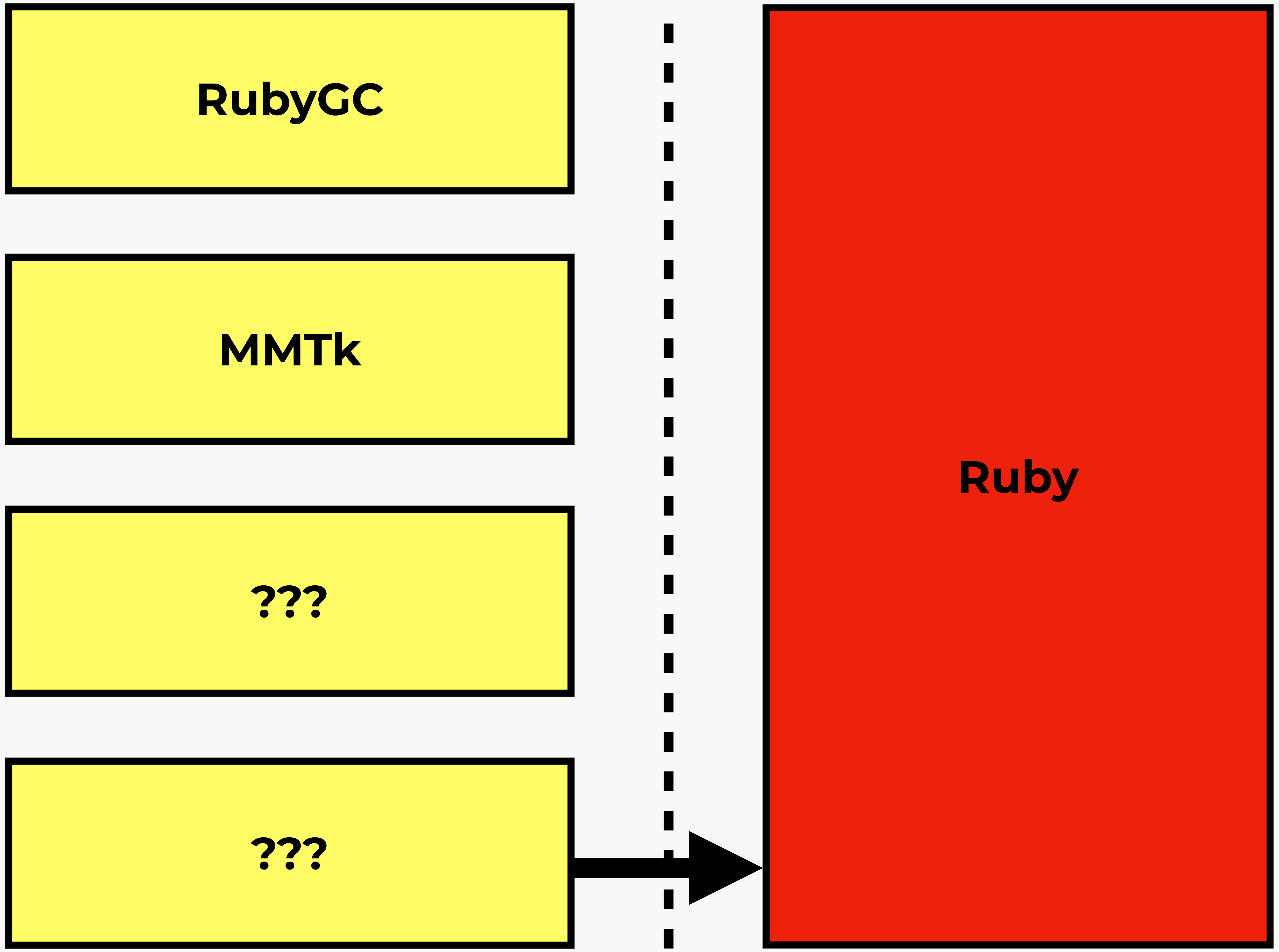
📈Benefits	📉Questions
GC library	
MMTk - Latest GC research	



Shenandoah
ZGC
LXR
???
MMTk
Ruby

📈Benefits	📉Questions
GC library	
MMTk - Latest GC research	
Easy GC split testing	


RubyGC
RubyGC-prime
Ruby


RubyGC
MMTk
???
???
Ruby

📈Benefits	📉Questions
GC library	
MMTk - Latest GC research	
Easy GC split testing	

📈 **Benefits**	📉 **Questions**
GC library	**C extensions**
MMTk - Latest GC research	
Easy GC split testing	

Add rb_gc_mark_and_move and implement on iseq #7140

Edit | <> Code

Merged **peterzhu2118** merged 2 commits into `ruby:master` from `Shopify:pz-rb-gc-mark-and-move` on Jan 19

[Feature #19406] Allow declarative definition of references for rb_typed_data_struct #7153

Edit | <> Code

Merged eightbitraptor merged 5 commits into `ruby:master` from `Shopify:mvh-declarative-marking` on Mar 17

📈Benefits	📉Questions
GC library	C extensions
MMTk - Latest GC research	**Maintenance burden**
Easy GC split testing	

Benefits	Questions
GC library	C extensions
MMTk - Latest GC research	Maintenance burden
Easy GC split testing	**Performance penalties**

Image by photoroyalty on Freepik

GC is not a solved problem. Advances in memory management research are happening all the time.

We have a real opportunity to ensure that Ruby's memory management modern and highly performant.

Thanks.

References & Acknowledgements:

bit.ly/mmtk-rubykaigi-2023