Optimizing Ruby's Memory Layout

Peter Zhu

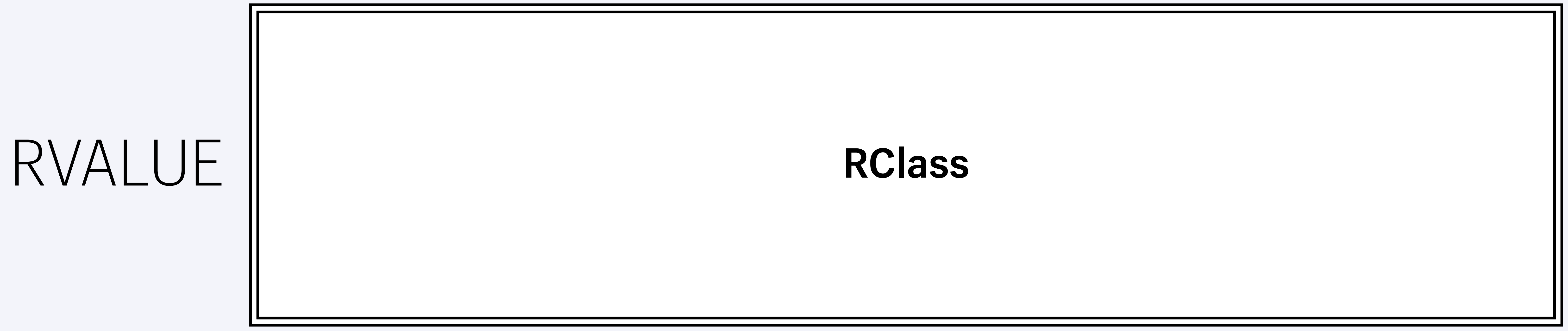
Ruby Core Committer
Production Engineer, Shopify

Matt Valentine-House

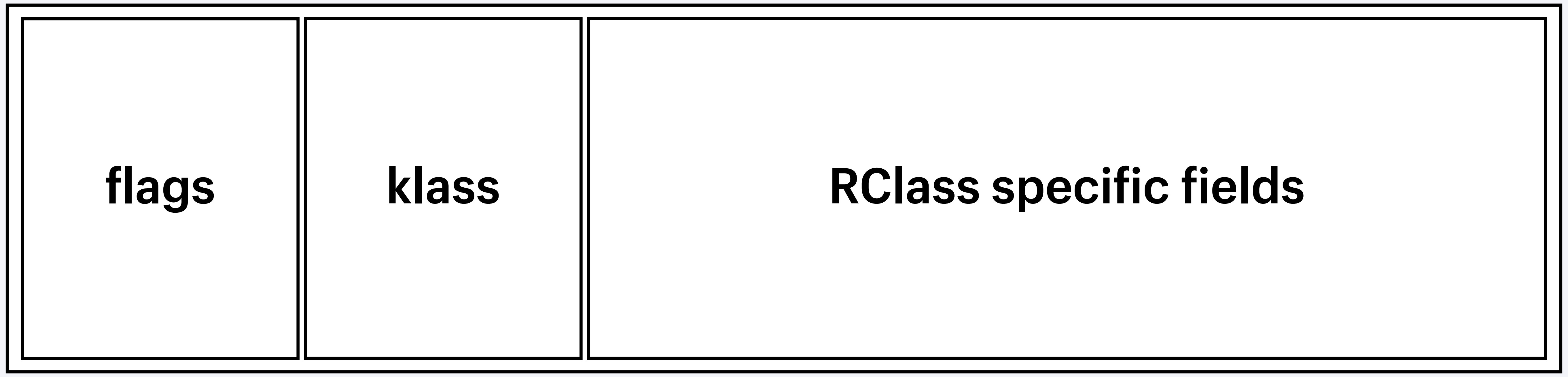
Senior Developer, Shopify

How does Ruby manage memory?

RVALUE structure



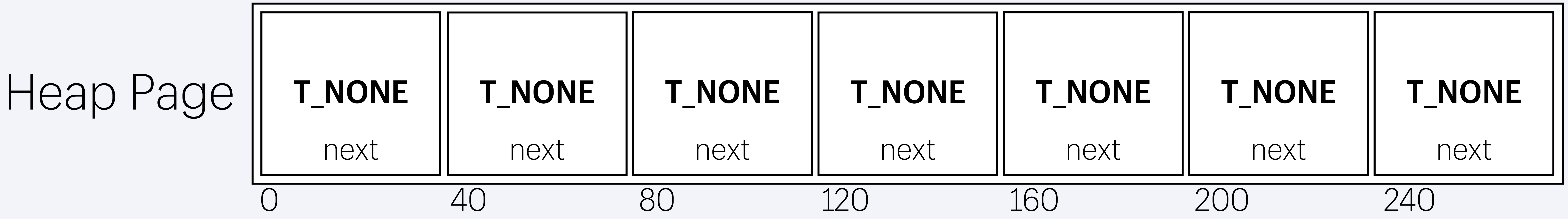
Ruby Object Structure



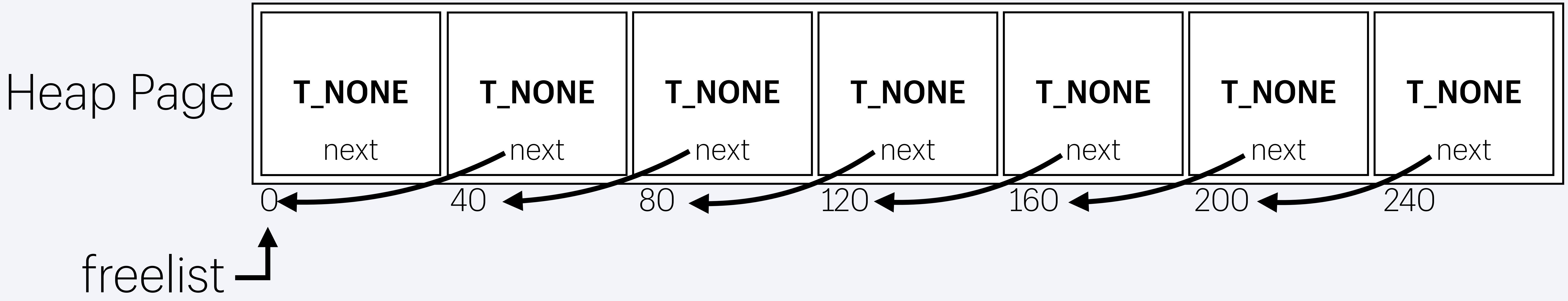
Heap page structure

- Heap pages are a container for a 16Kb memory region
- 409 slots per page
- All slots on the same page are contiguous. No gaps in addresses

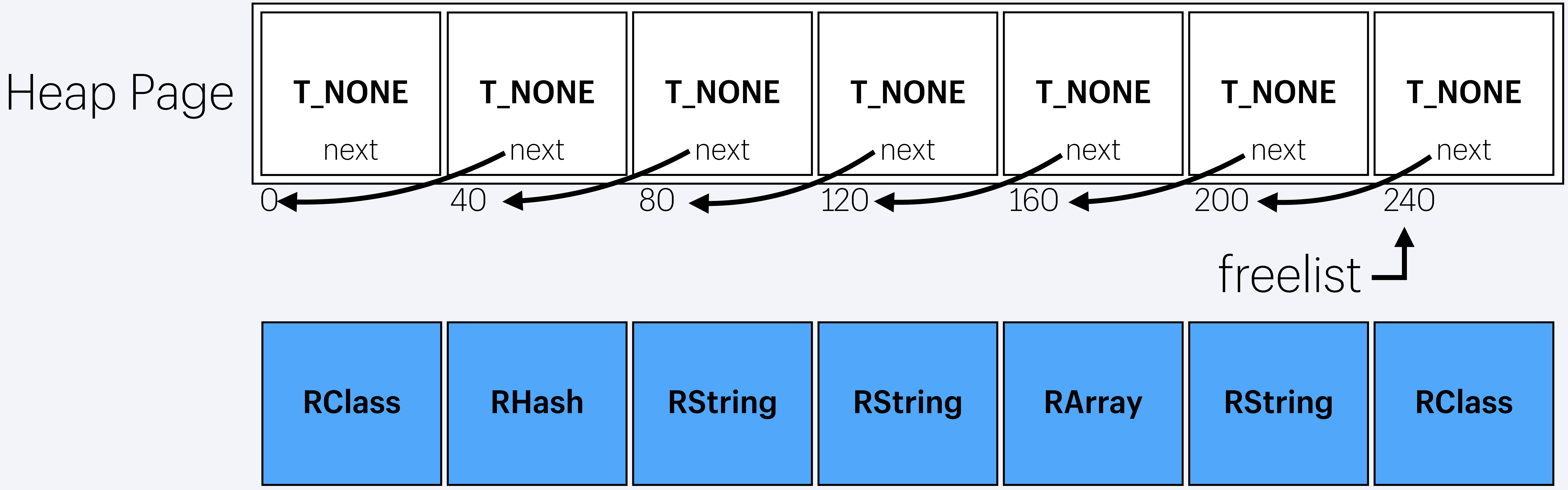
Heap page structure



Building the freelist



Allocating Ruby objects



How does Ruby’s Garbage Collector work?

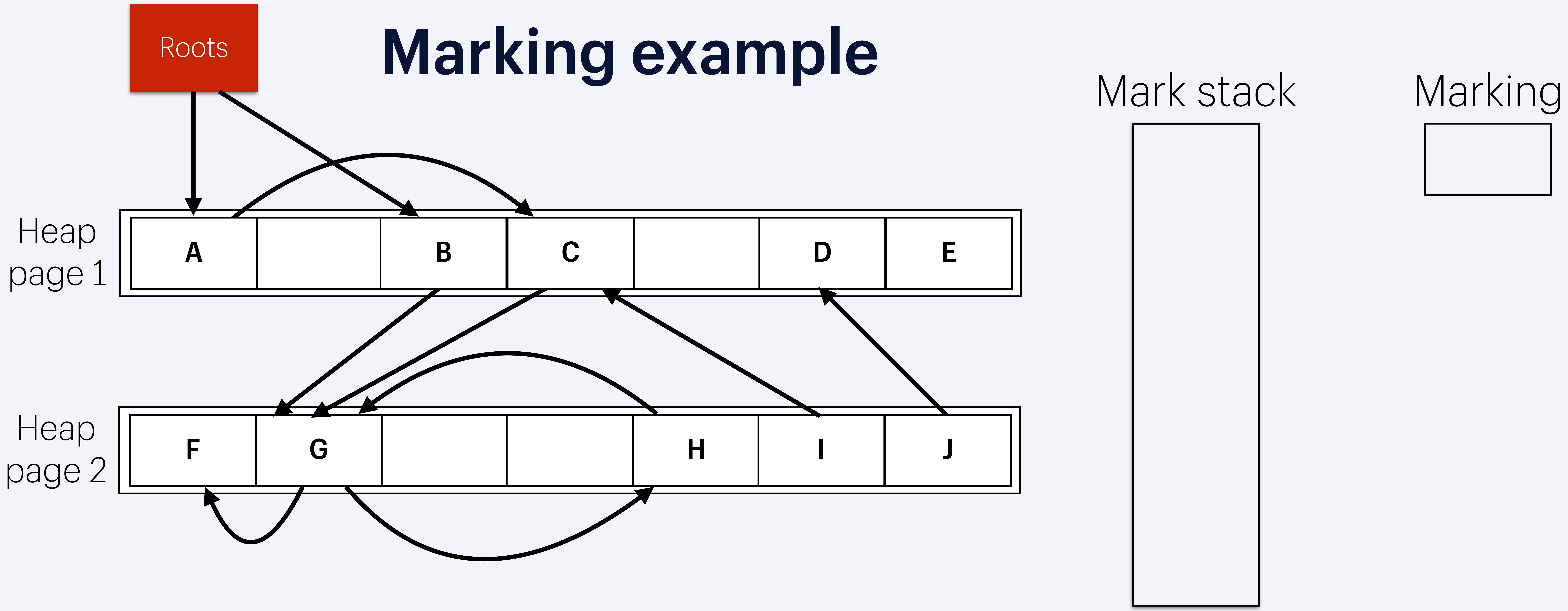
Ruby's garbage collector

- Three phases:
  - Marking
  - Sweeping
  - Compaction (optional)
- Stop-the-world garbage collection
- Disclaimer: algorithms are simplified and some details are skipped

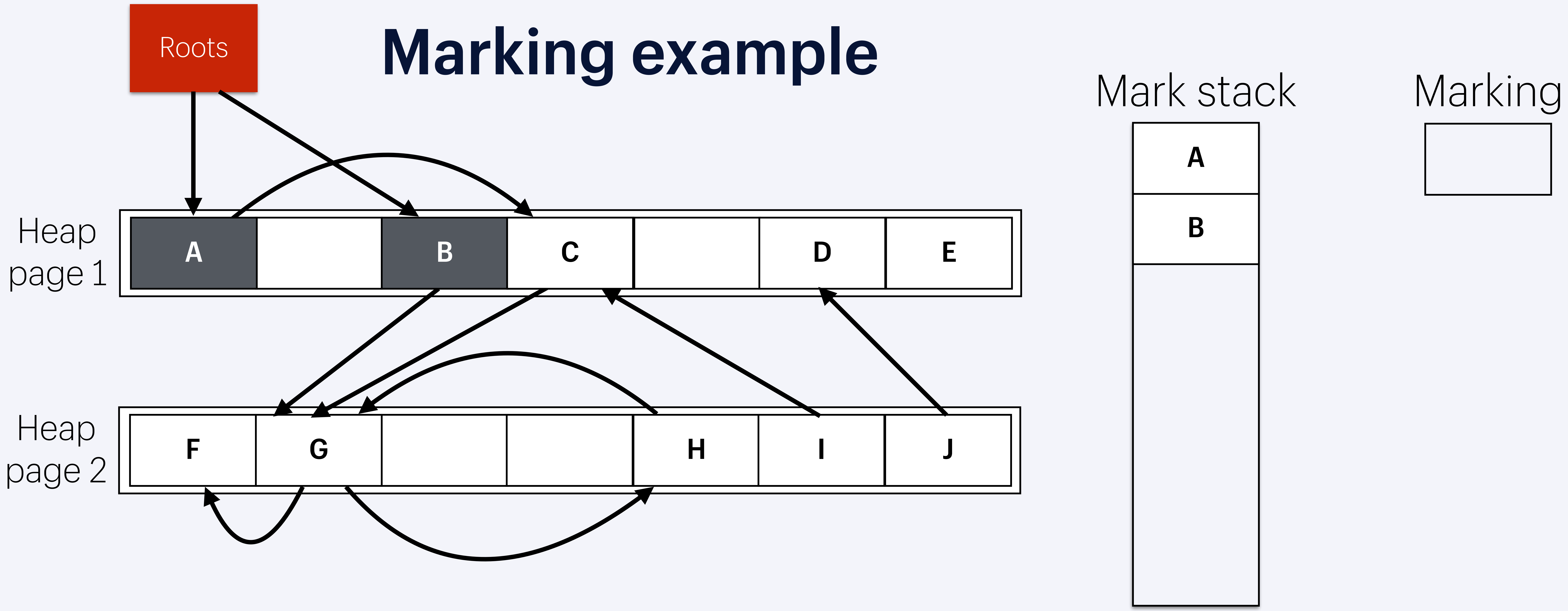
Marking phase

- Determines which Ruby objects are alive
- Push the object onto the mark stack when marked
- Recursively mark unmarked children of marked objects until empty

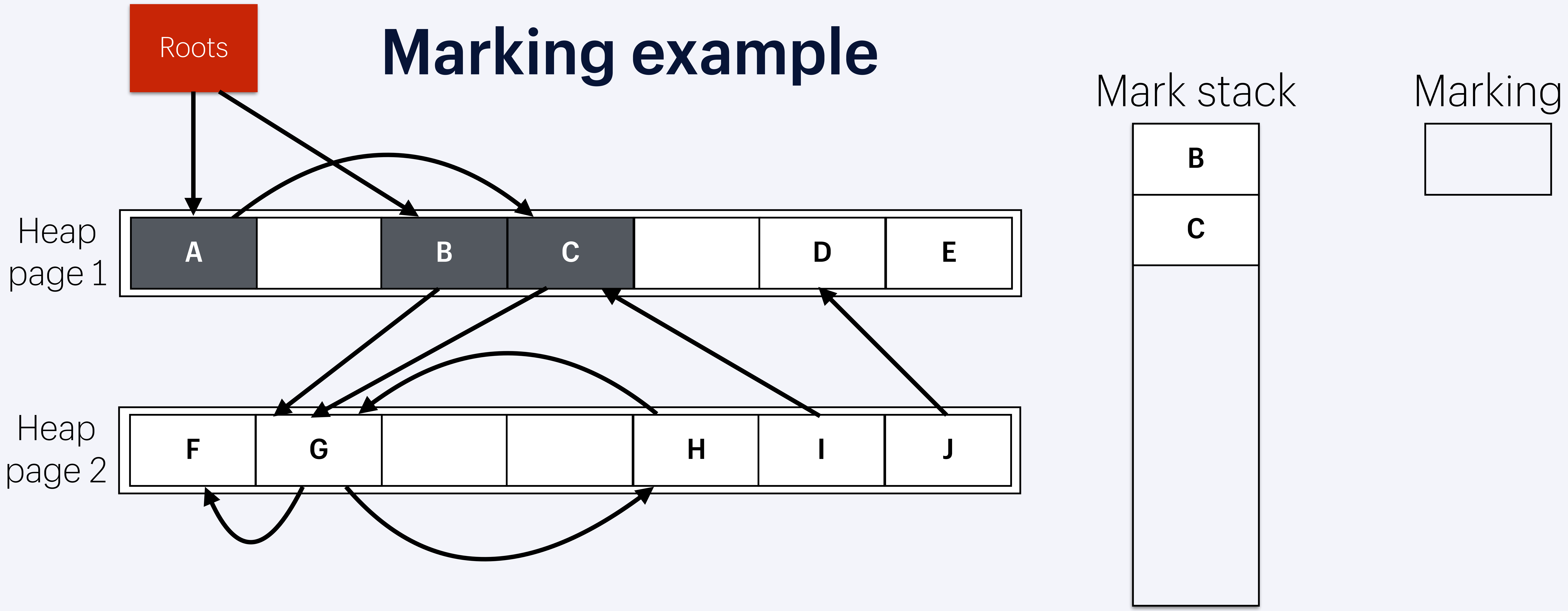
Marking example



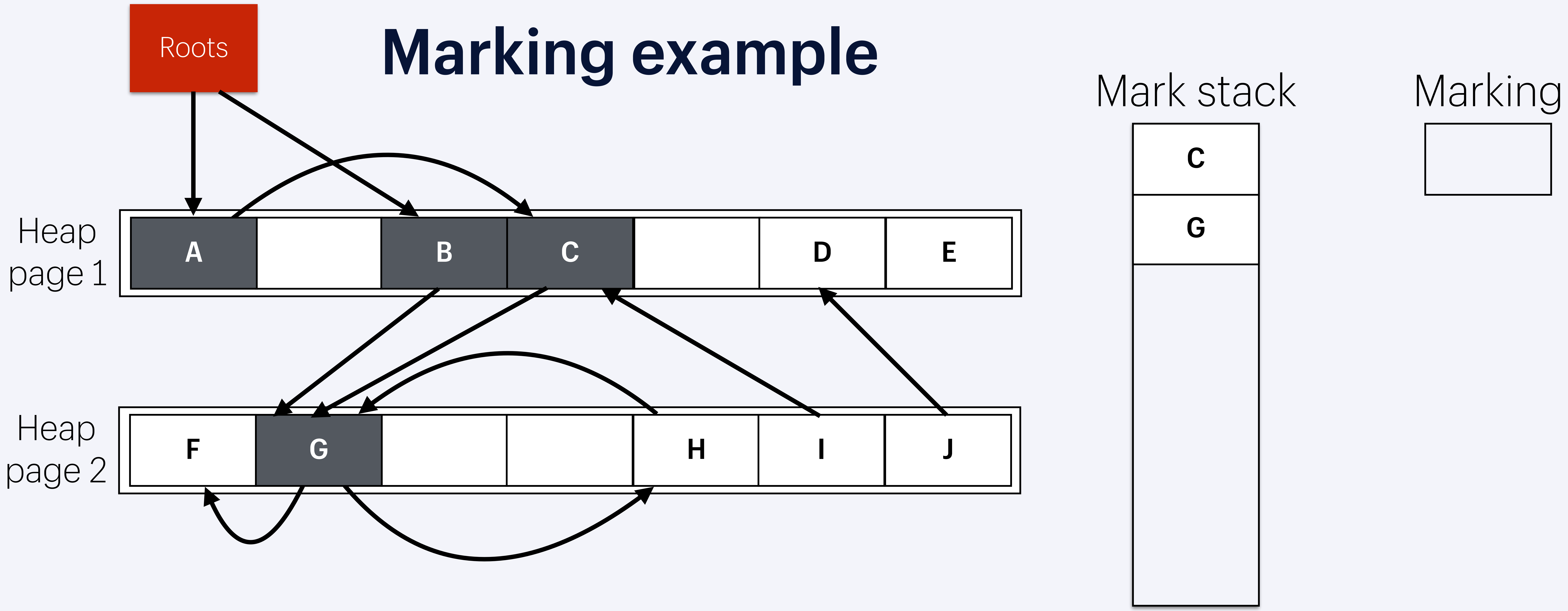
Marking example



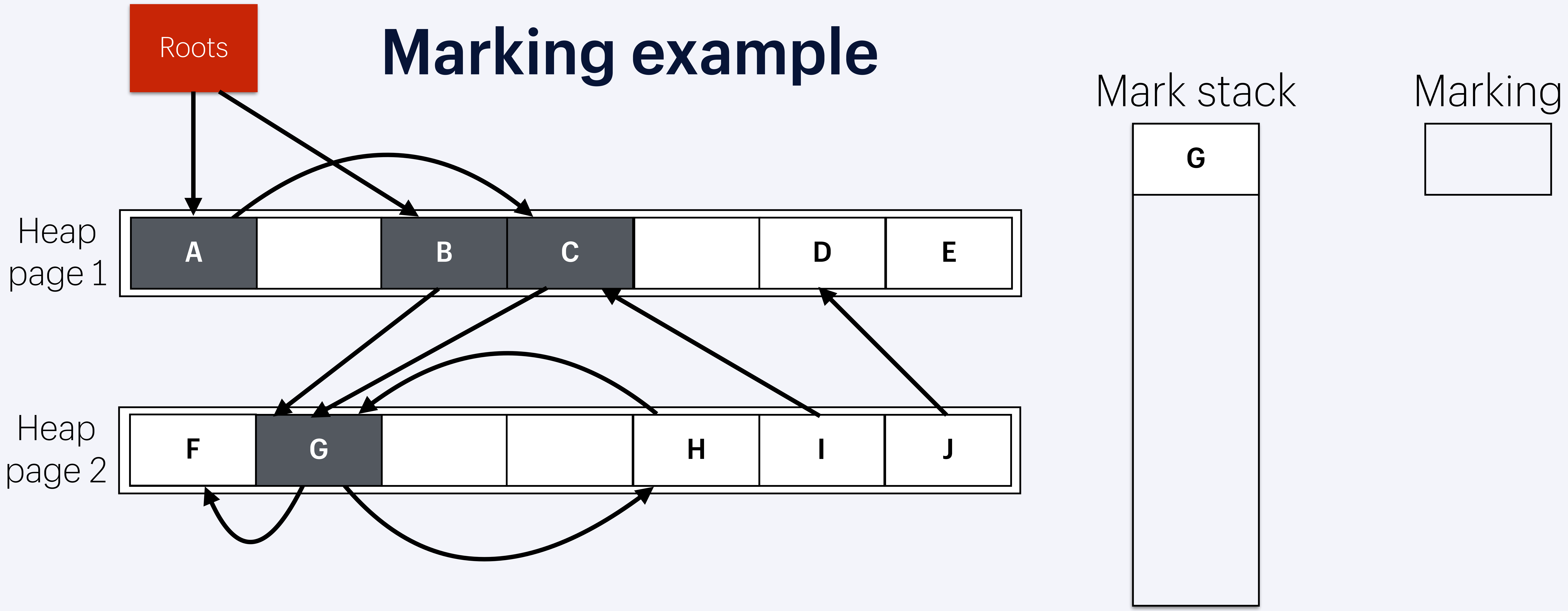
Marking example



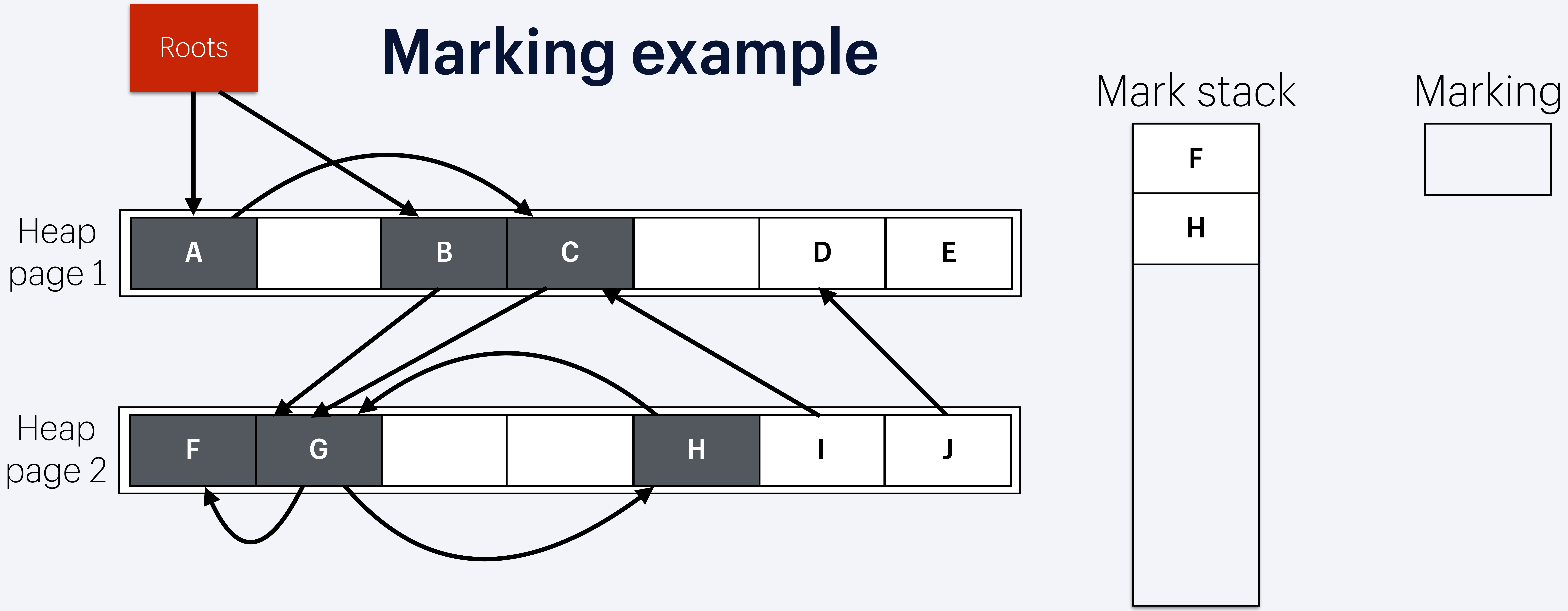
Marking example



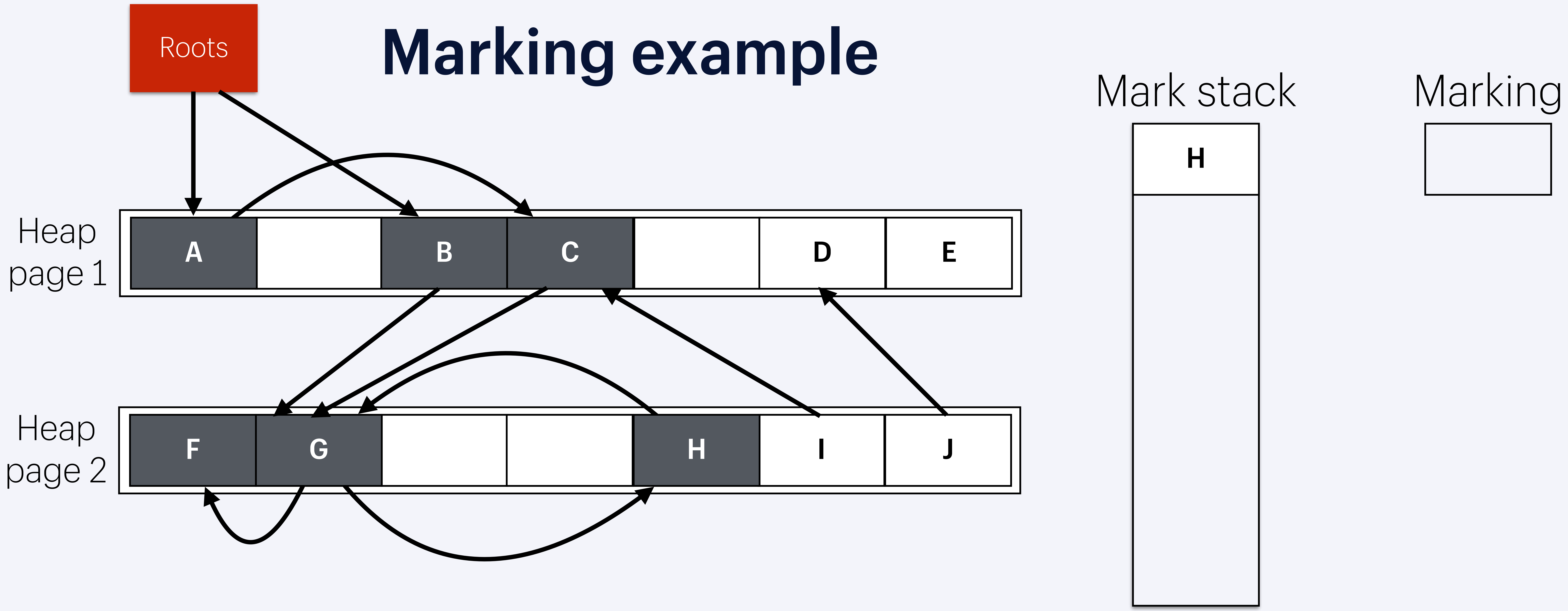
Marking example



Marking example



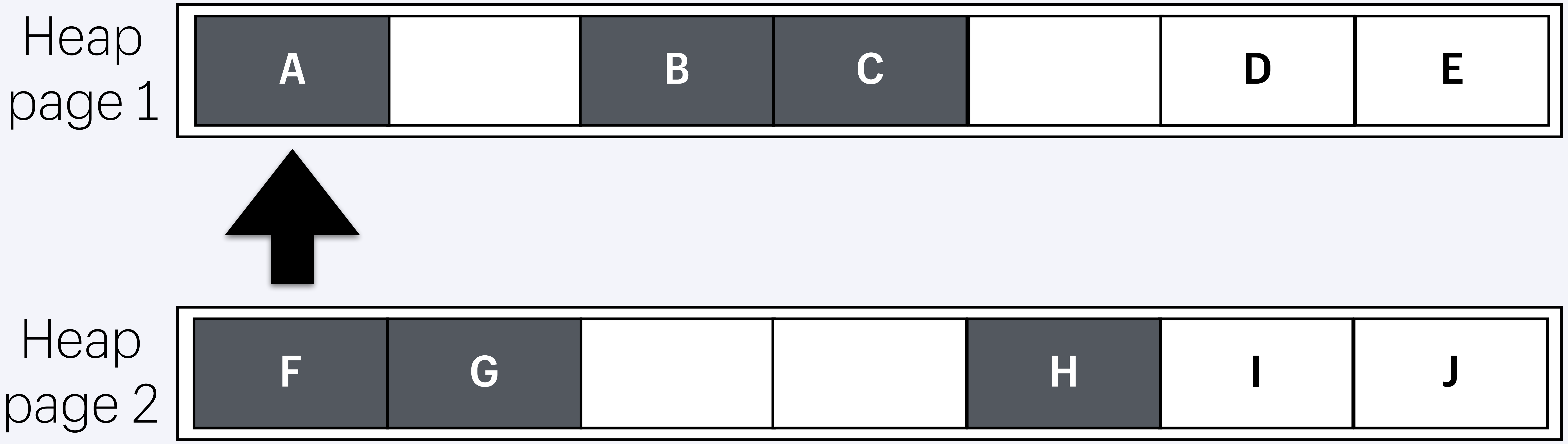
Marking example



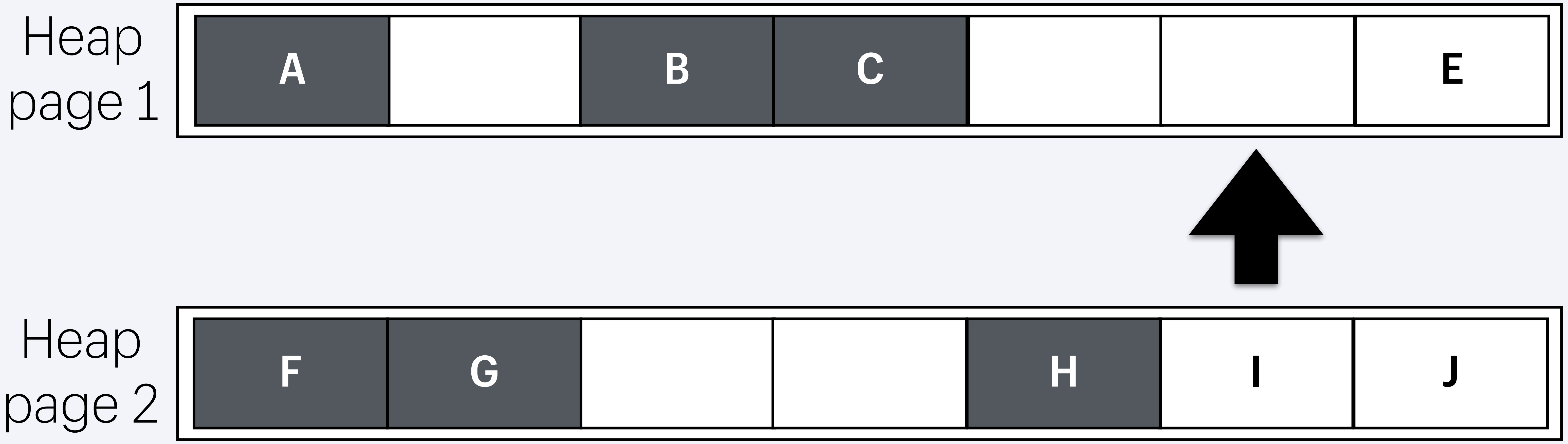
Sweeping phase

- Marked objects = live objects
- Unmarked objects = dead objects
- Scan pages and free objects that are not marked

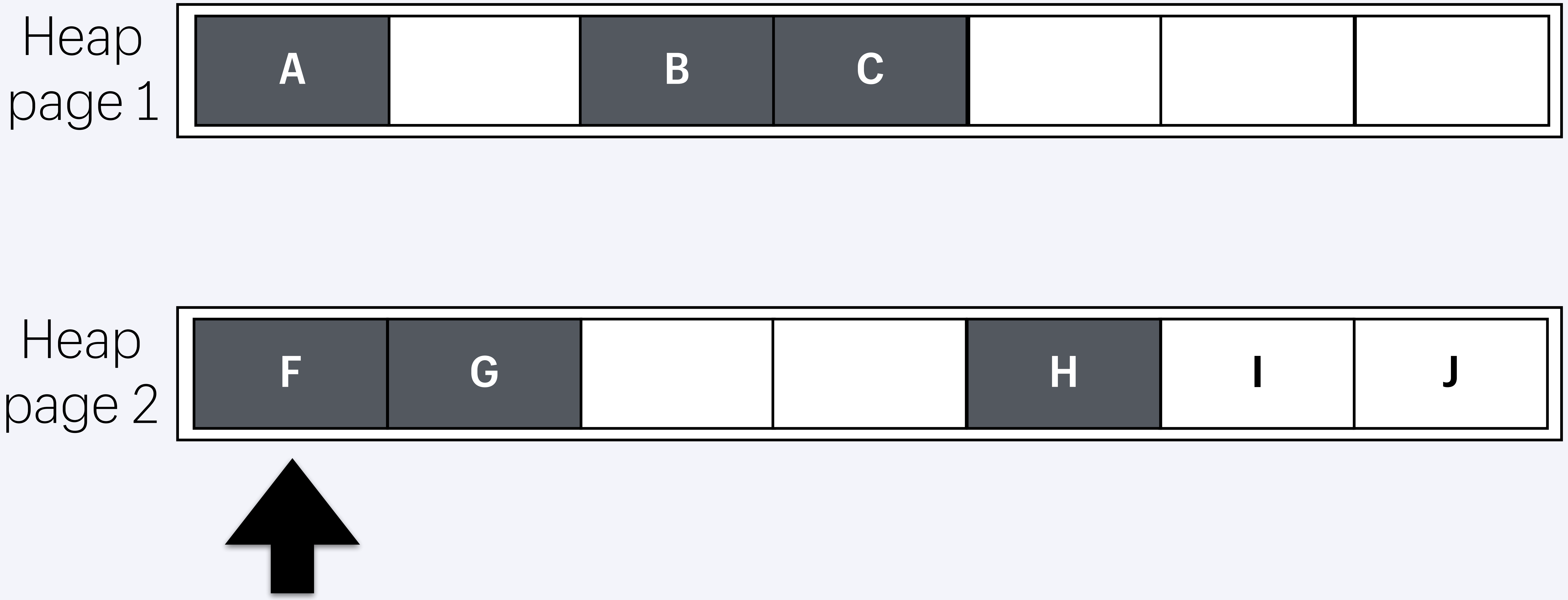
Sweeping example



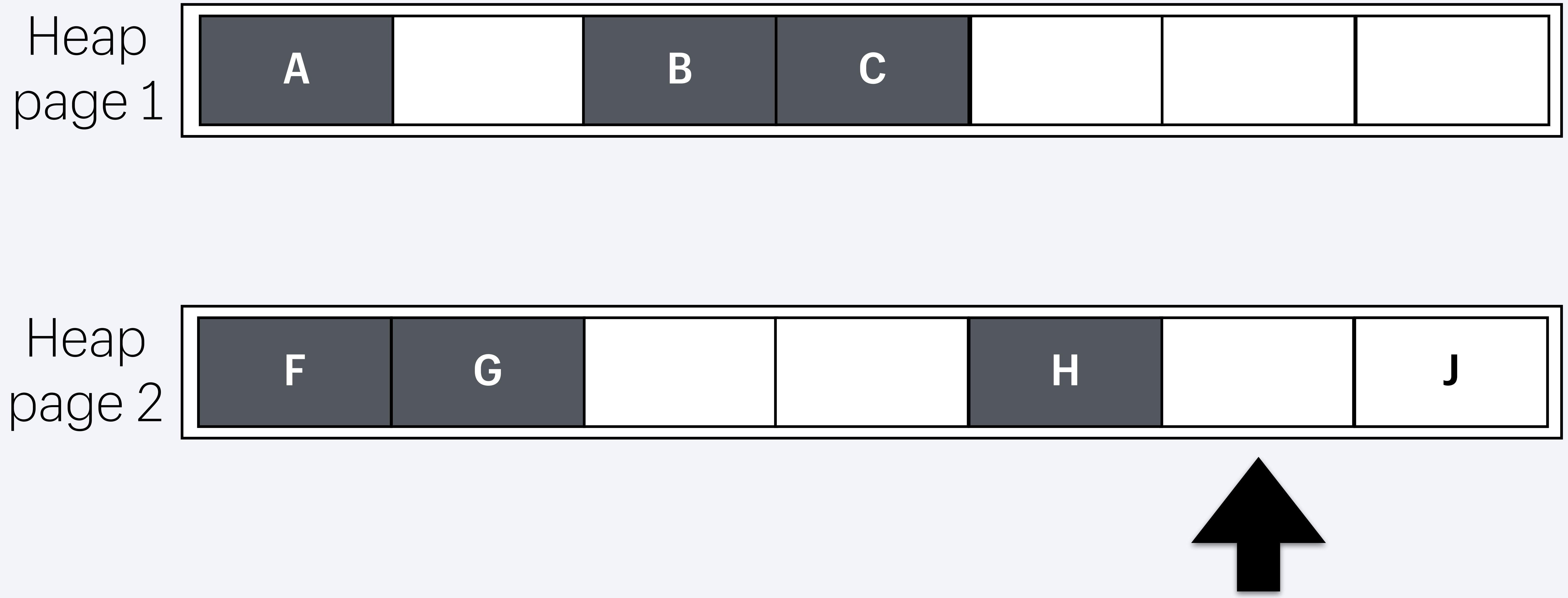
Sweeping example



Sweeping example



Sweeping example



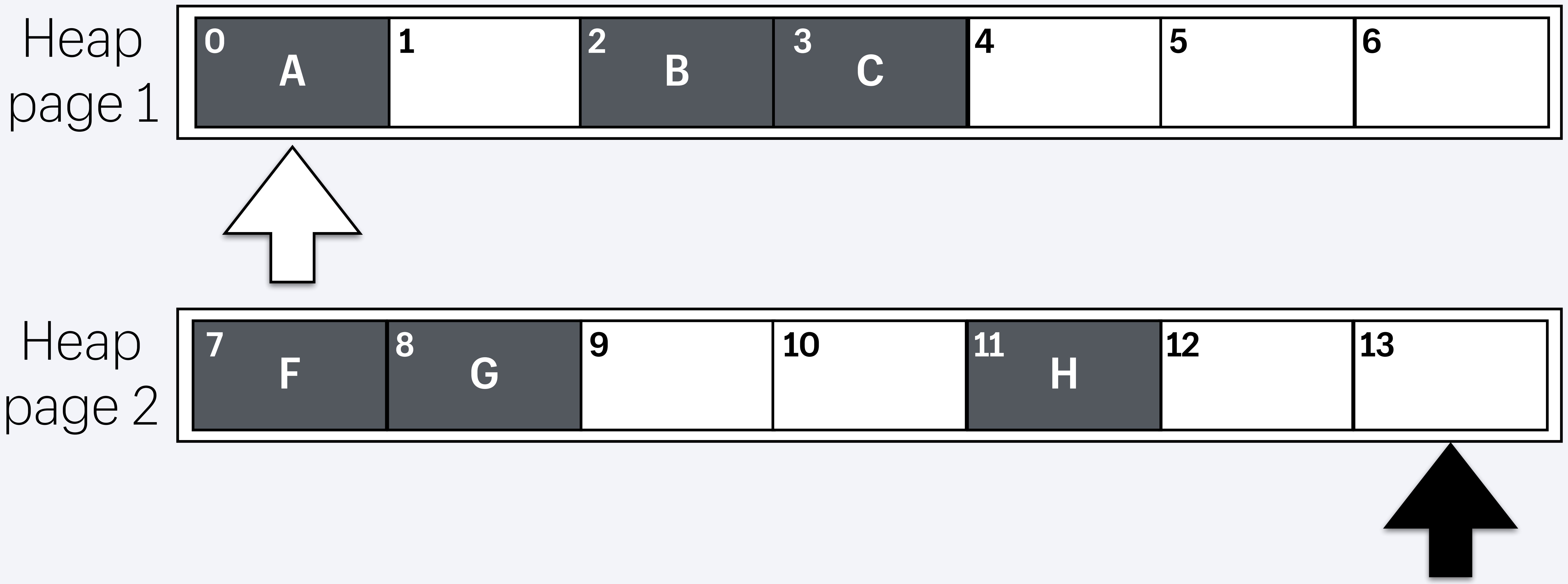
Compact phase

- Optional phase & turned off by default
- Move objects to compact the heap
- Can reduce memory usage
- Ruby uses a Two-Finger compaction algorithm

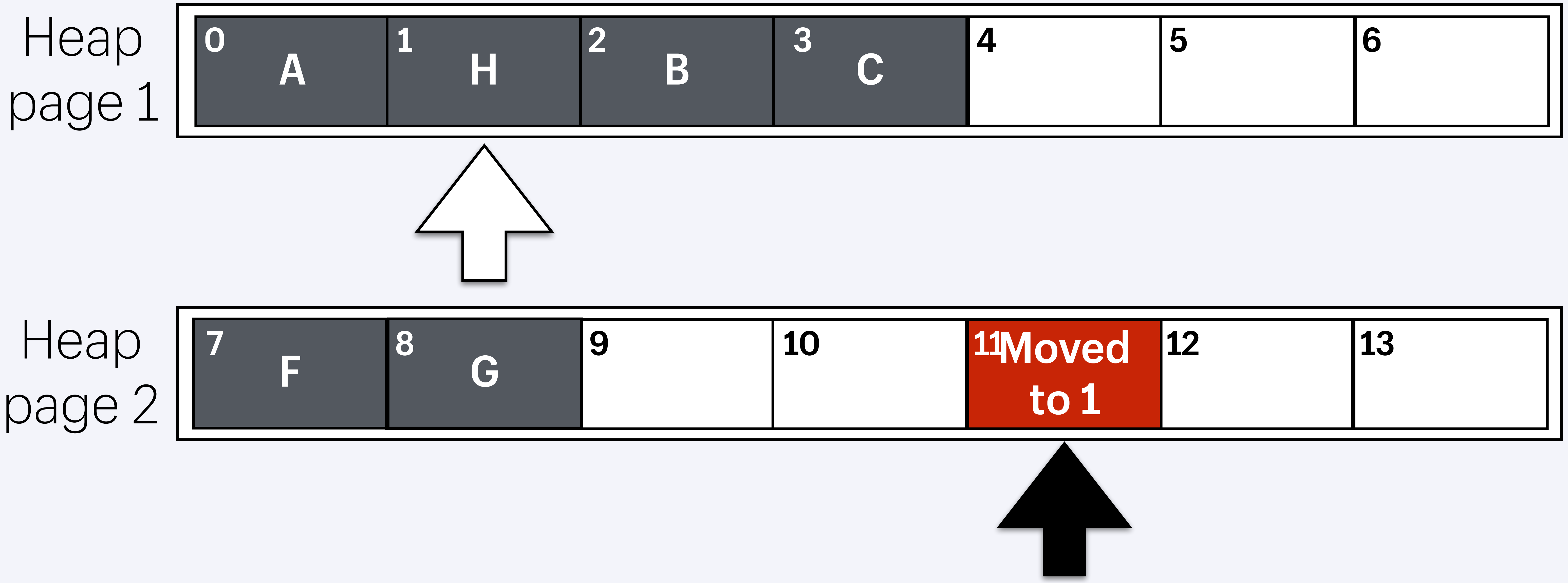
Compaction algorithm

- Compact step:
  - Two cursors: compact and free
  - Free cursor moves forward and compact cursor moves backward
- Update reference step: update pointers for all objects

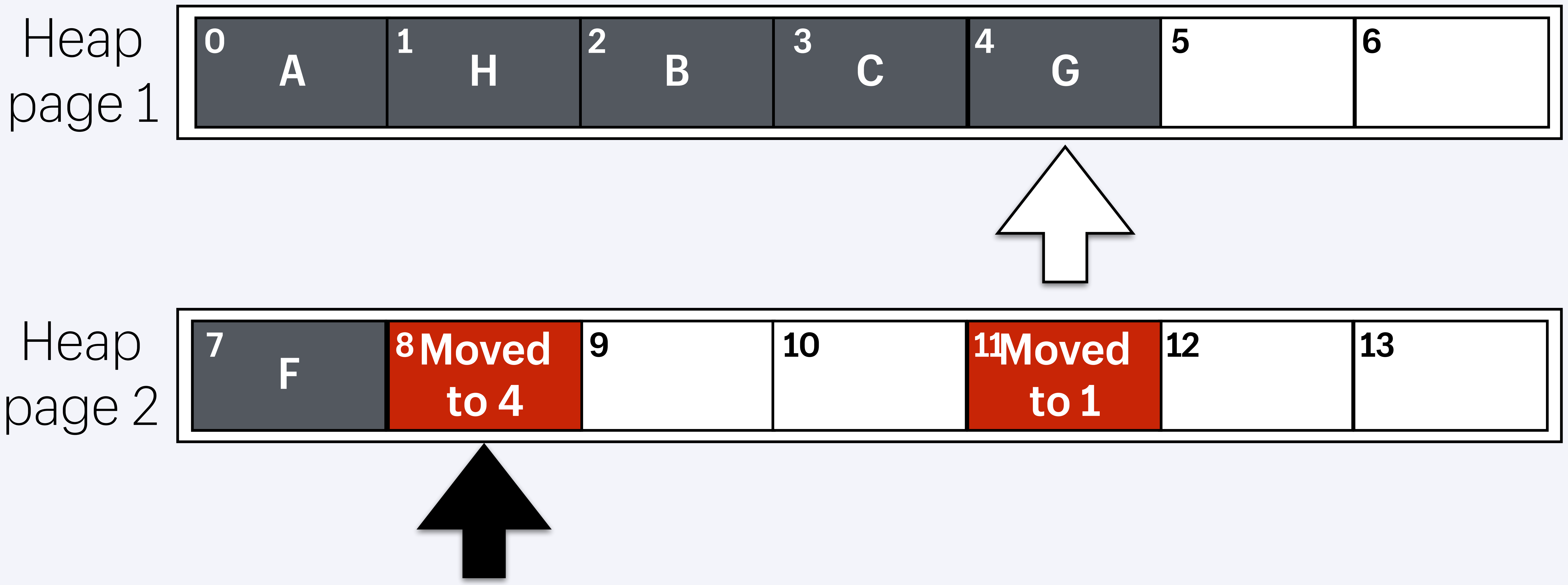
Compaction example



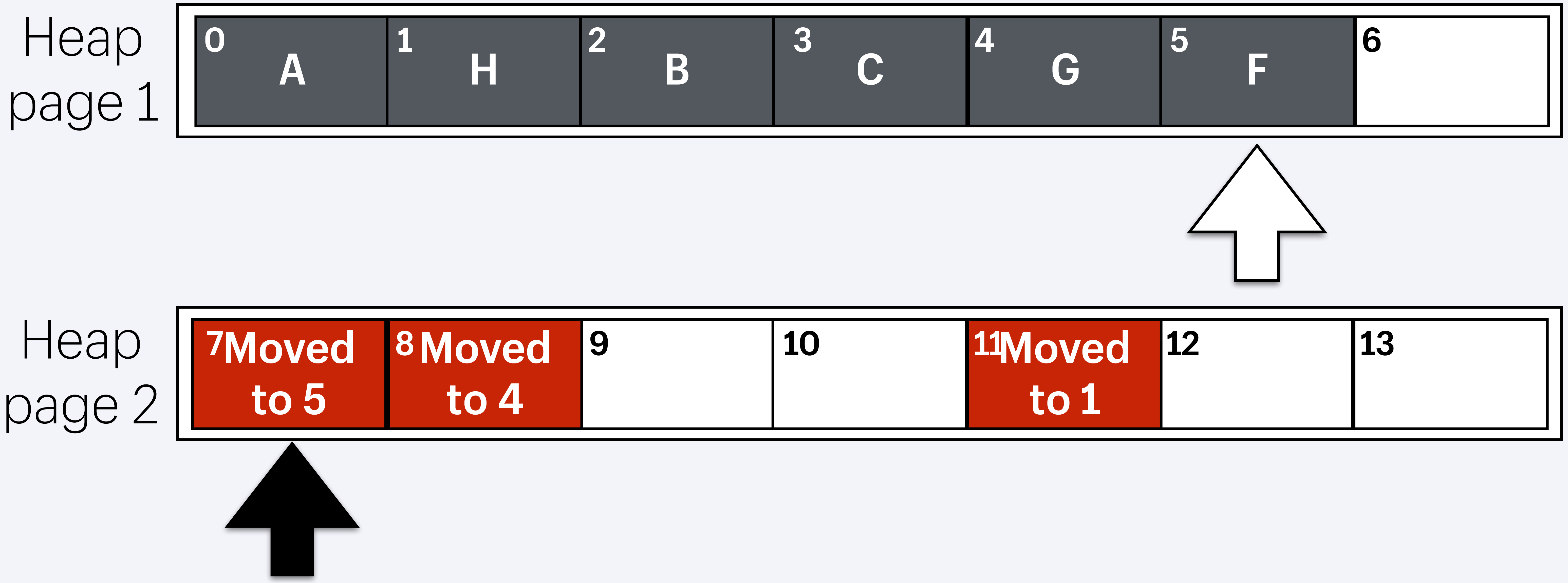
Compaction example



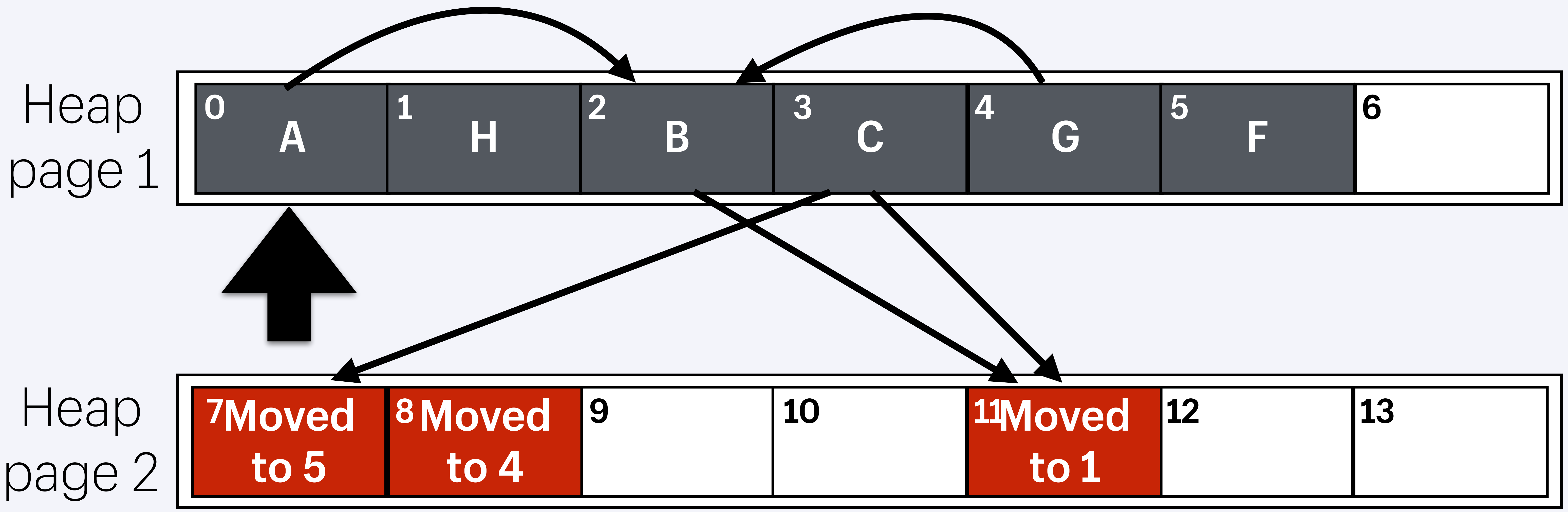
Compaction example



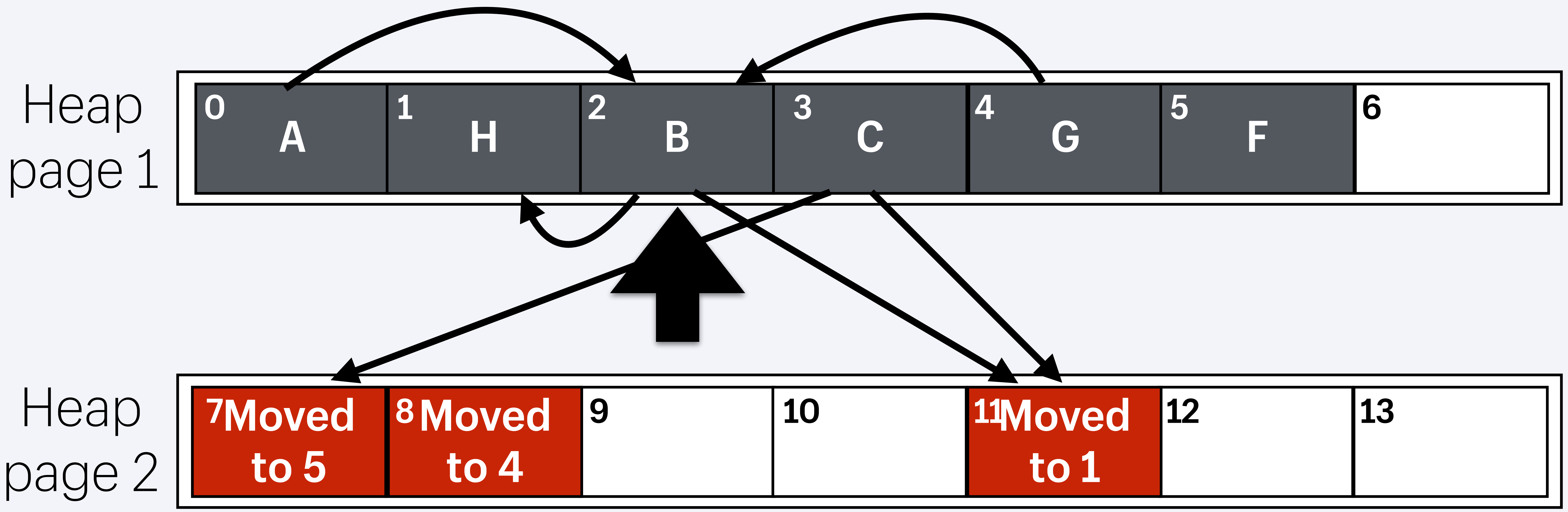
Compaction example



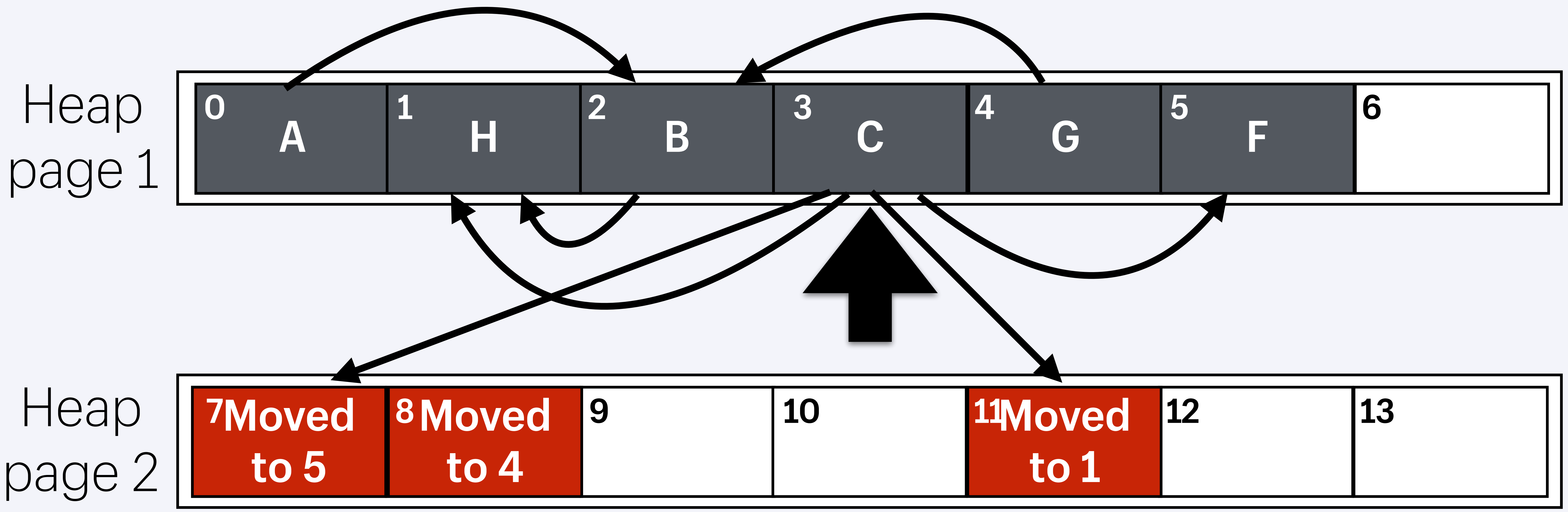
Compaction example



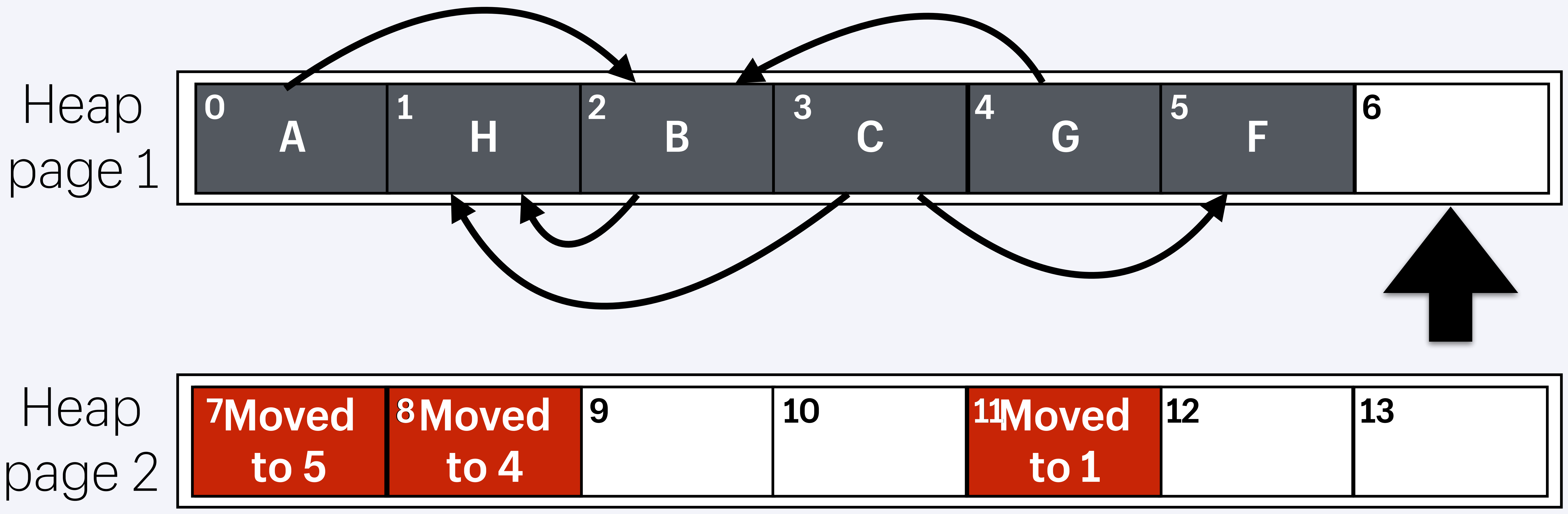
Compaction example



Compaction example



Compaction example



Large objects on the heap

Two different categories of Strings

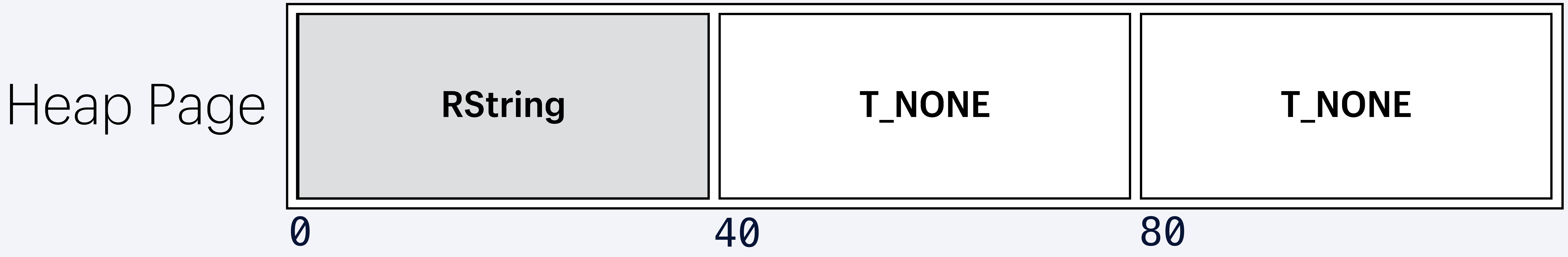
- < 24 bytes

  `“Hello, World”` 12 bytes

- > 24 bytes

  `“Hello RubyKaigi, thanks for having us”` 37 bytes

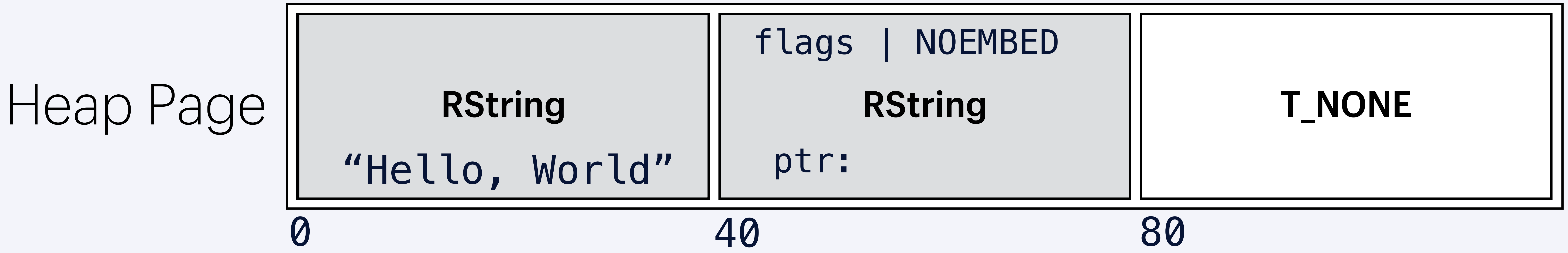
Allocating an embedded string



“Hello, World” < RSTRING_EMBED_LEN_MAX == TRUE 12 bytes

“Hello RubyKaigi, thanks for having us” 37 bytes

Allocating a heap allocated string



“Hello RubyKaigi, thanks for having us”

37 bytes

800:

malloc(< RSTRING_EMBED_LEN_MAX == FALSE)

Summary: What we’ve learned

- How Ruby lays out it’s memory.
- How large and small objects are allocated
- What garbage collection is for, and how it works

What problems are we trying to solve?

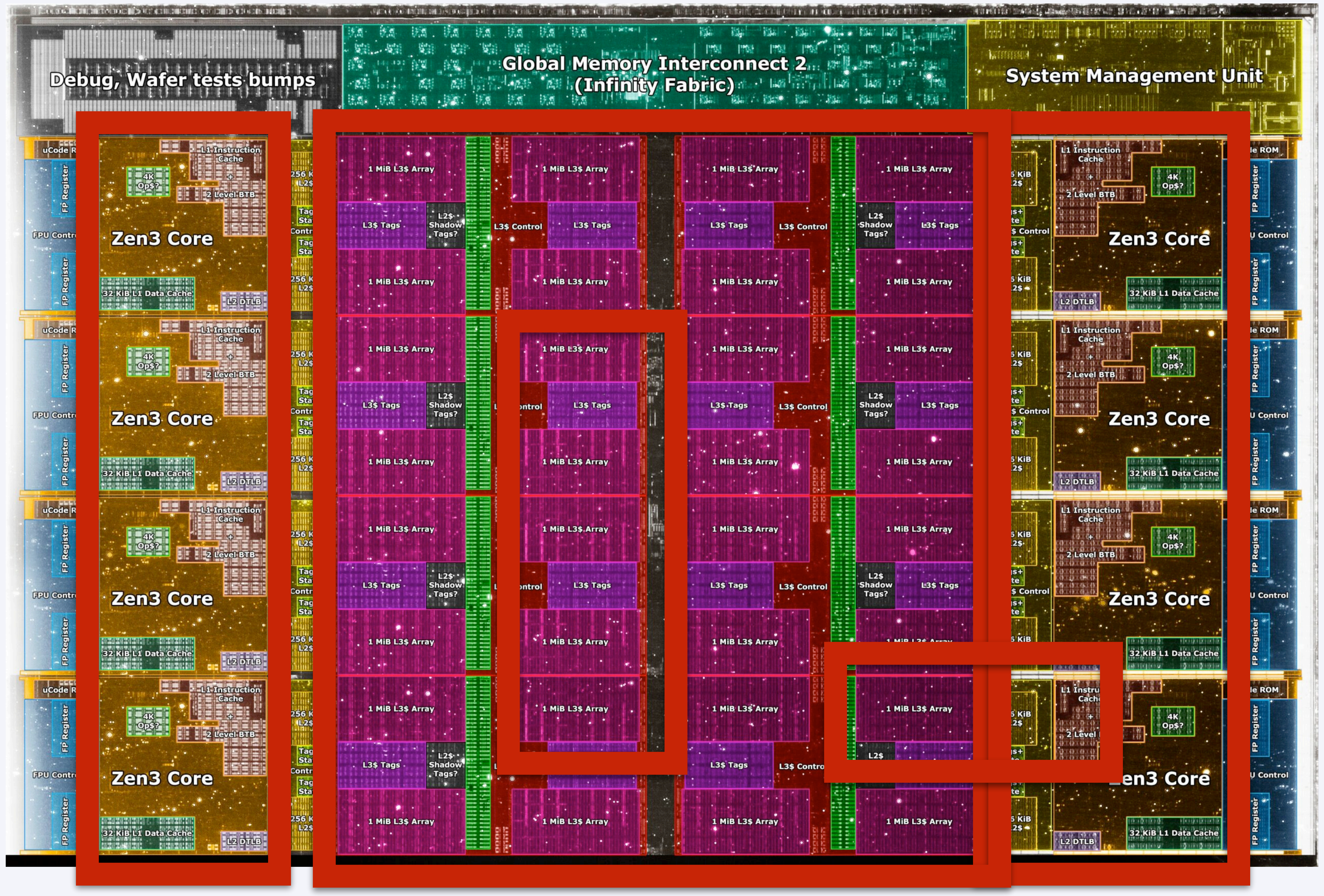
Bottlenecks in the heap

- Pointer indirection causing poor cache locality
- Performance and memory overhead caused by malloc

CPU caches

- Memory in the system lives in many levels:
  - Level 1 cache: on CPU core, very fast
  - Level 2 cache: beside CPU core, slightly slower
  - Level 3 cache: shared between CPU cores, slower
  - Main memory: off CPU, very slow

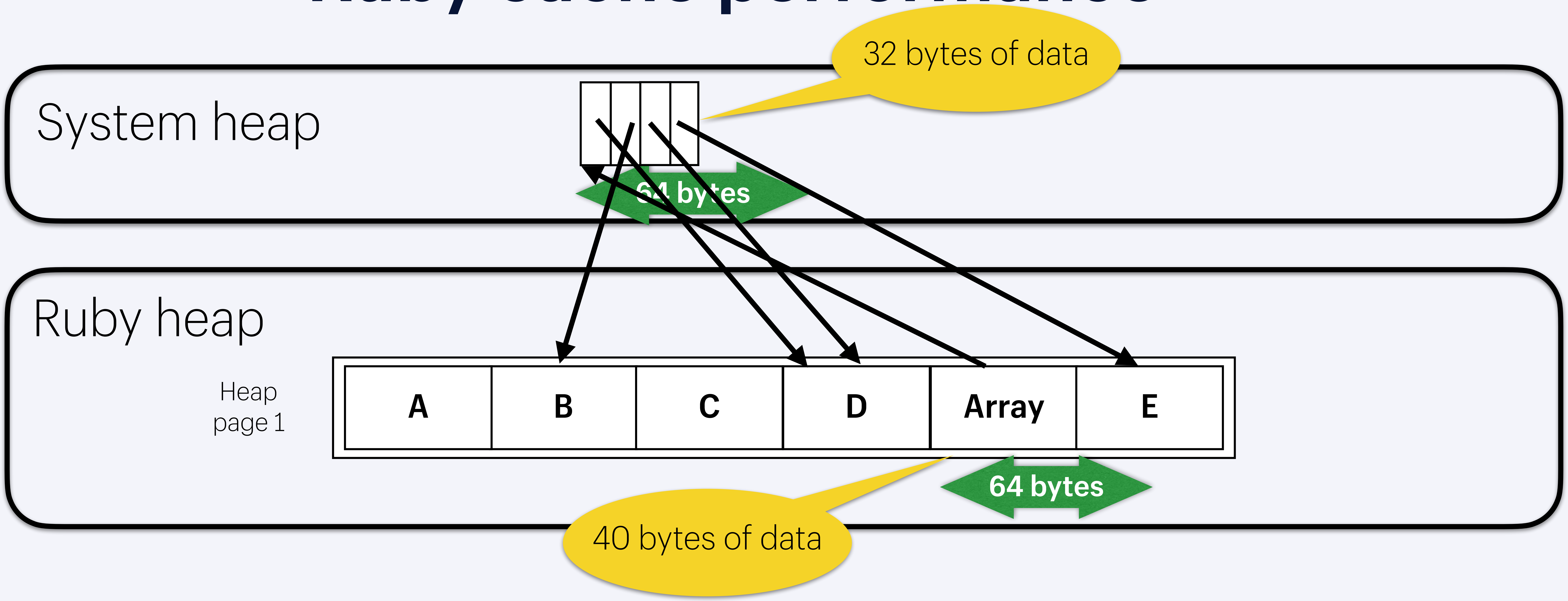
CPU caches


Debug, Wafer tests bumps
Global Memory Interconnect 2
(Infinity Fabric)
System Management Unit
Zen3 Core
Zen3 Core
Zen3 Core
Zen3 Core
Zen3 Core
Zen3 Core
Zen3 Core

CPU cache properties

- CPU caches stores data fetched from main memory
  - CPU caches store a cache line at a time (64 B on x86)
  - Old cache entries are evicted to make space for new entries
- Cache hit: data exists in cache, no fetch from main memory required
- Cache miss: data does not exist in cache, need to fetch from main memory

Ruby cache performance



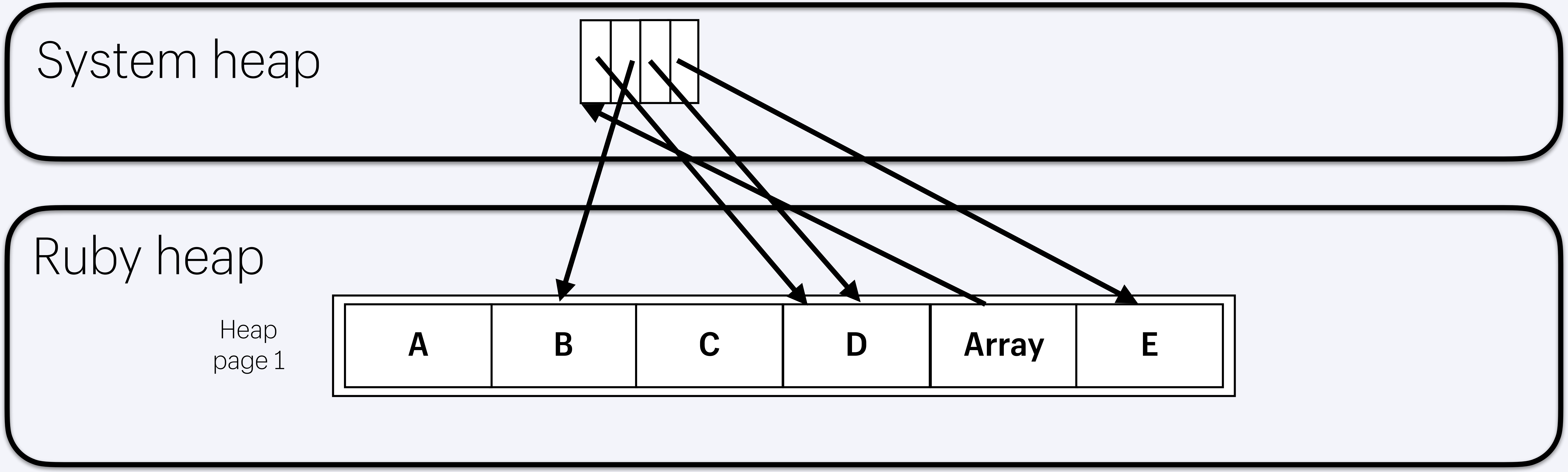
Overhead of malloc

- malloc has performance overhead
- malloc stores additional metadata, increasing memory consumption
- Ruby 2.6+ introduced a second heap called the “transient heap” used to reduce the number of malloc calls
  - Increased performance in some benchmarks by 50%

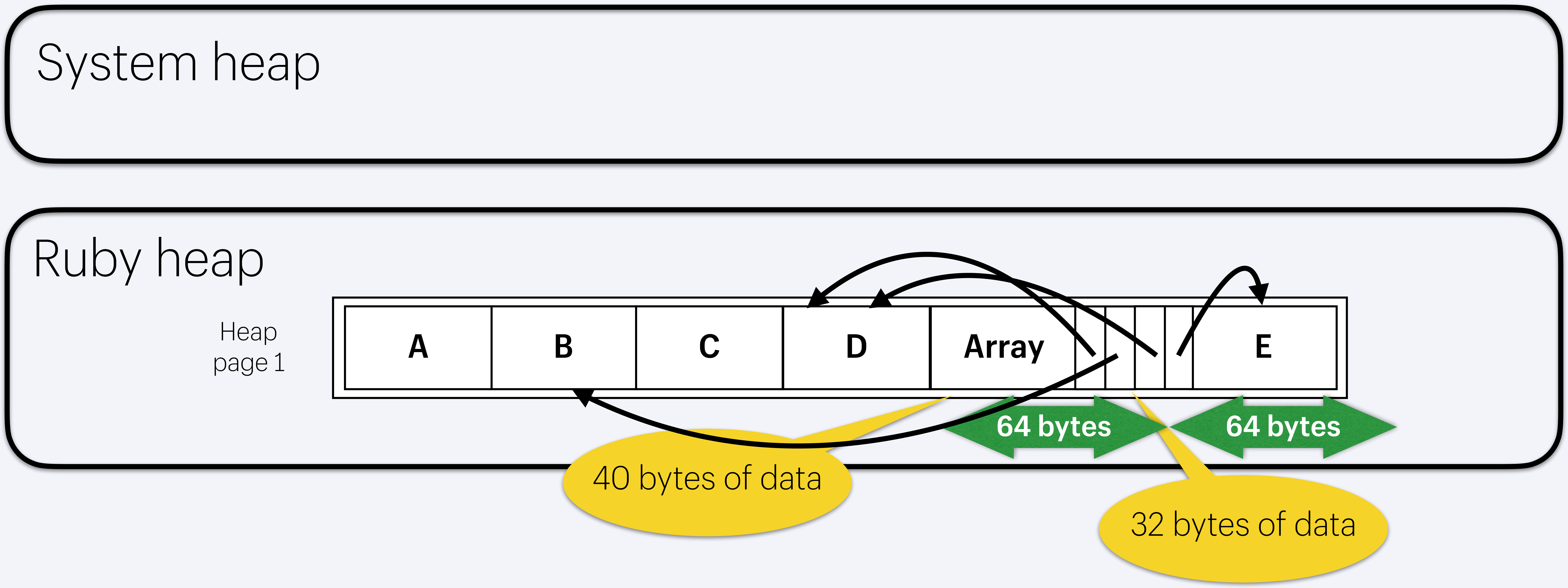
The Variable Width Allocation project

- Extend Ruby's garbage collector to allow dynamic sized allocation
- Data will be allocated following the object RVALUE
- Reduce the number of malloc calls

Variable Width cache performance



Variable Width cache performance



Where are we today?

ruby/ruby

#4391 Move C heap allocations for RVALUE object data into GC heap

0 comments 1 review 7 files +452 −57

eightbitraptor • April 20, 2021 3 commits

Move C heap allocations for RVALUE object data into GC heap by eightbitraptor · Pull Request #4391 · ruby/ruby

GITHUB.COM

ruby/ruby

#4680 Variable Width Allocation Phase II

1 comment · 11 reviews · 7 files · +760 −692

peterzhu2118 • July 26, 2021 · 2 commits

Variable Width Allocation Phase II by peterzhu2118 · Pull R...

Ticket: https://bugs.ruby-lang.org/issues/18045 Feature description Since merging the initial implementation in #1757...

github.com

🚨🚨DON’T USE THIS* 🚨🚨

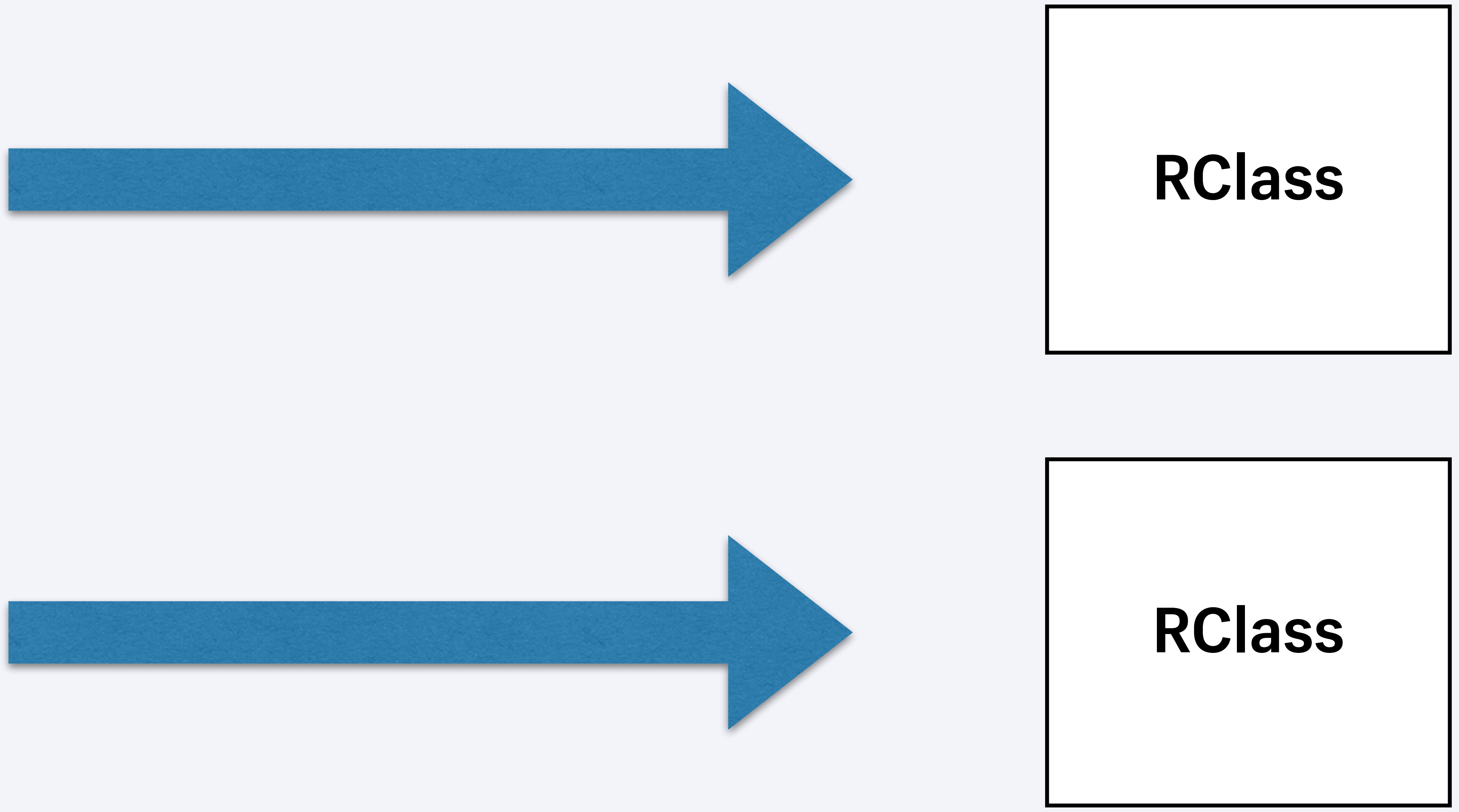
* on production workloads

```
export cflags="-DUSE_RVARGC=1"
./configure
make
make install
```

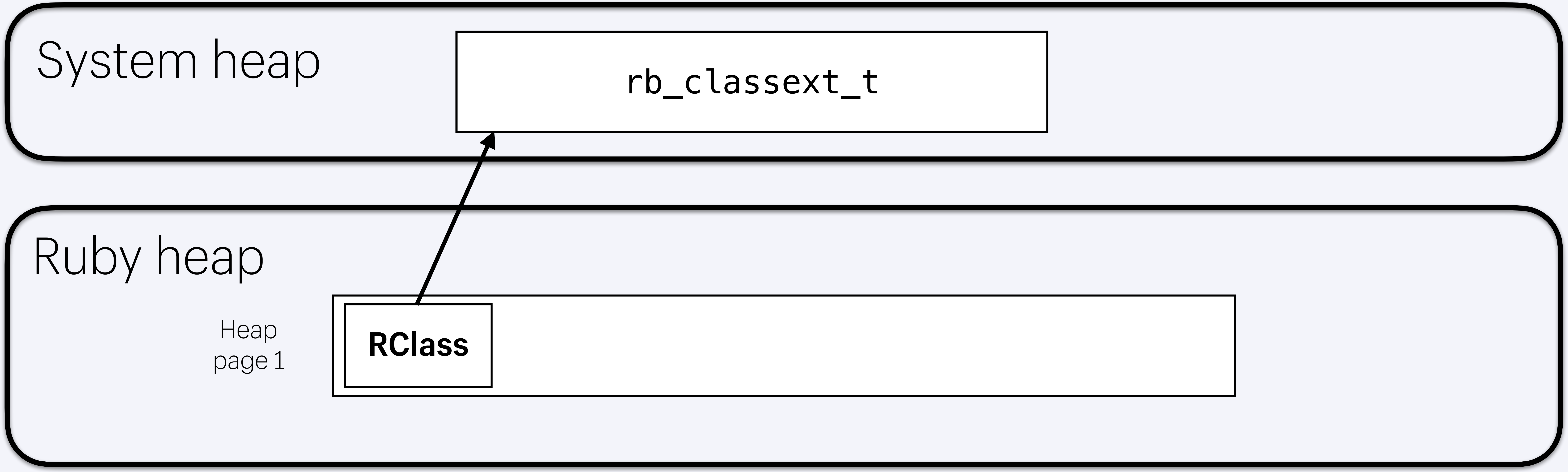
RClass allocation

```
class MyNewClass
end
```

```
Class.new(Object)
```



RClass Allocation



```
static VALUE
class_alloc(VALUE flags, VALUE klass)
{
    NEWOBJ_OF(obj, struct RClass, klass, (flags & T_MASK) | FL_PROMOTED1 | (RGENGC_WB_PROTECTED_CLASS ?
FL_WB_PROTECTED : 0));

    obj->ptr = ZALLOC(rb_classext_t);

   /* snipped unrelated code */

    return (VALUE)obj;
}
```

```c
static VALUE
class_alloc(VALUE flags, VALUE klass)
{
    payload_size = sizeof(rb_classext_t);

    RVARGC_NEWOBJ_OF(obj, struct RClass, klass, (flags & T_MASK) | FL_PROMOTED1 |
(RGENGC_WB_PROTECTED_CLASS ? FL_WB_PROTECTED : 0), payload_size);

    obj->ptr = (rb_classext_t *)rb_rvargc_payload_data_ptr((VALUE)obj + rb_slot_size());

    return (VALUE)obj;
}
```

Variable Width Allocation

- RVARGC_NEWOBJ_OF called with a desired payload size
- Object is allocated in the appropriate size pool
  - Pages in size pools have different slot sizes
  - Slots of size pools have powers of 2 multiples of RVALUE size

Size pools

Size pool 0
(Slot size: 40B)

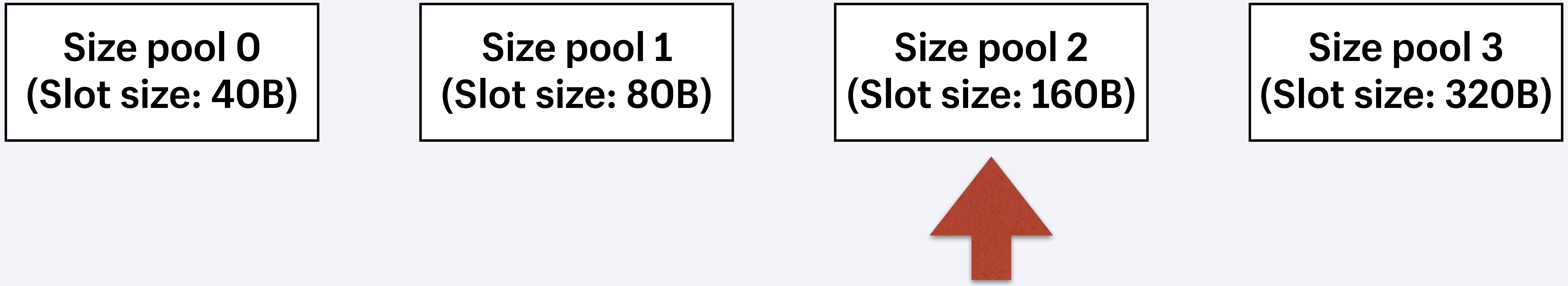
Size pool 1
(Slot size: 80B)

Size pool 2
(Slot size: 160B)

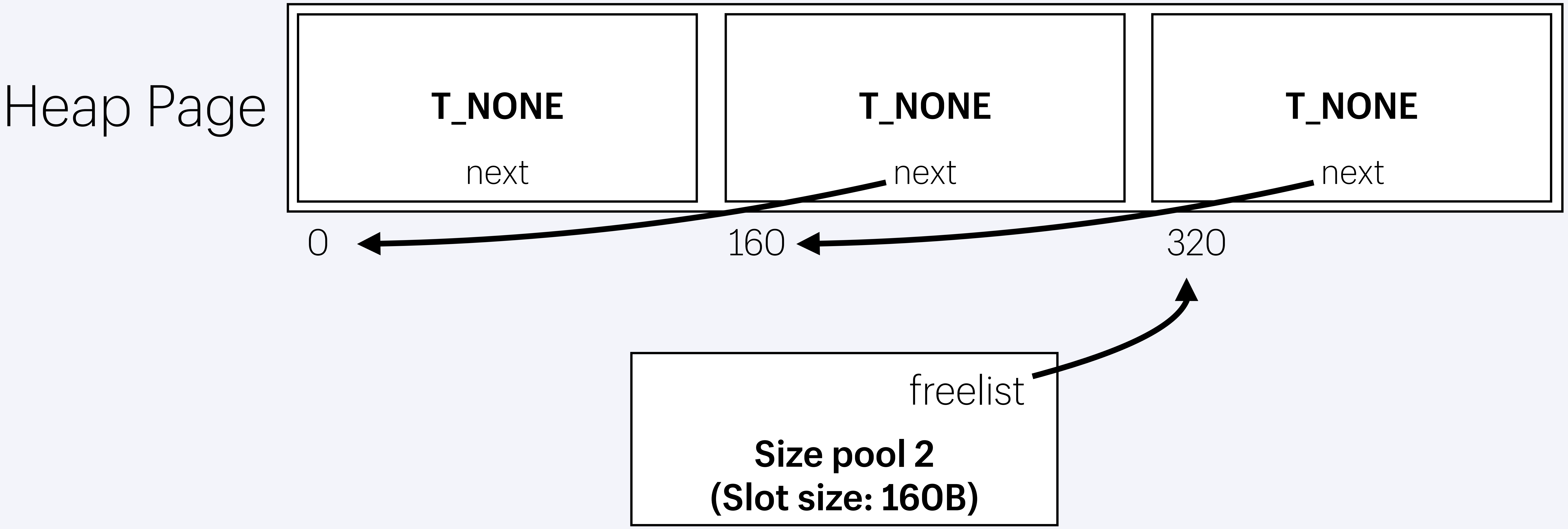
Size pool 3
(Slot size: 320B)

Size pools

- Allocating a class requires 40B + 104B = 144B
  - 144B = 3.6 x RVALUE

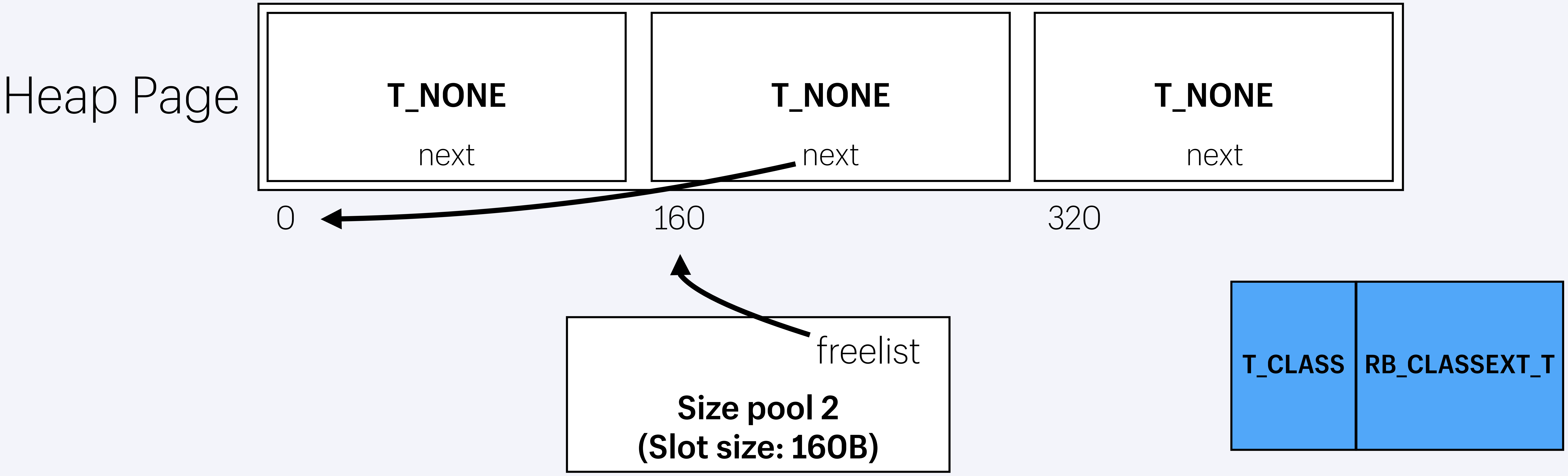


Allocation



Heap Page
T_NONE
next
T_NONE
next
T_NONE
next
0
160
320
freelist
Size pool 2
(Slot size: 160B)

Allocation



Heap Page
T_NONE
next
T_NONE
next
T_NONE
next
0
160
320
freelist
Size pool 2
(Slot size: 160B)
T_CLASS
RB_CLASSEXT_T

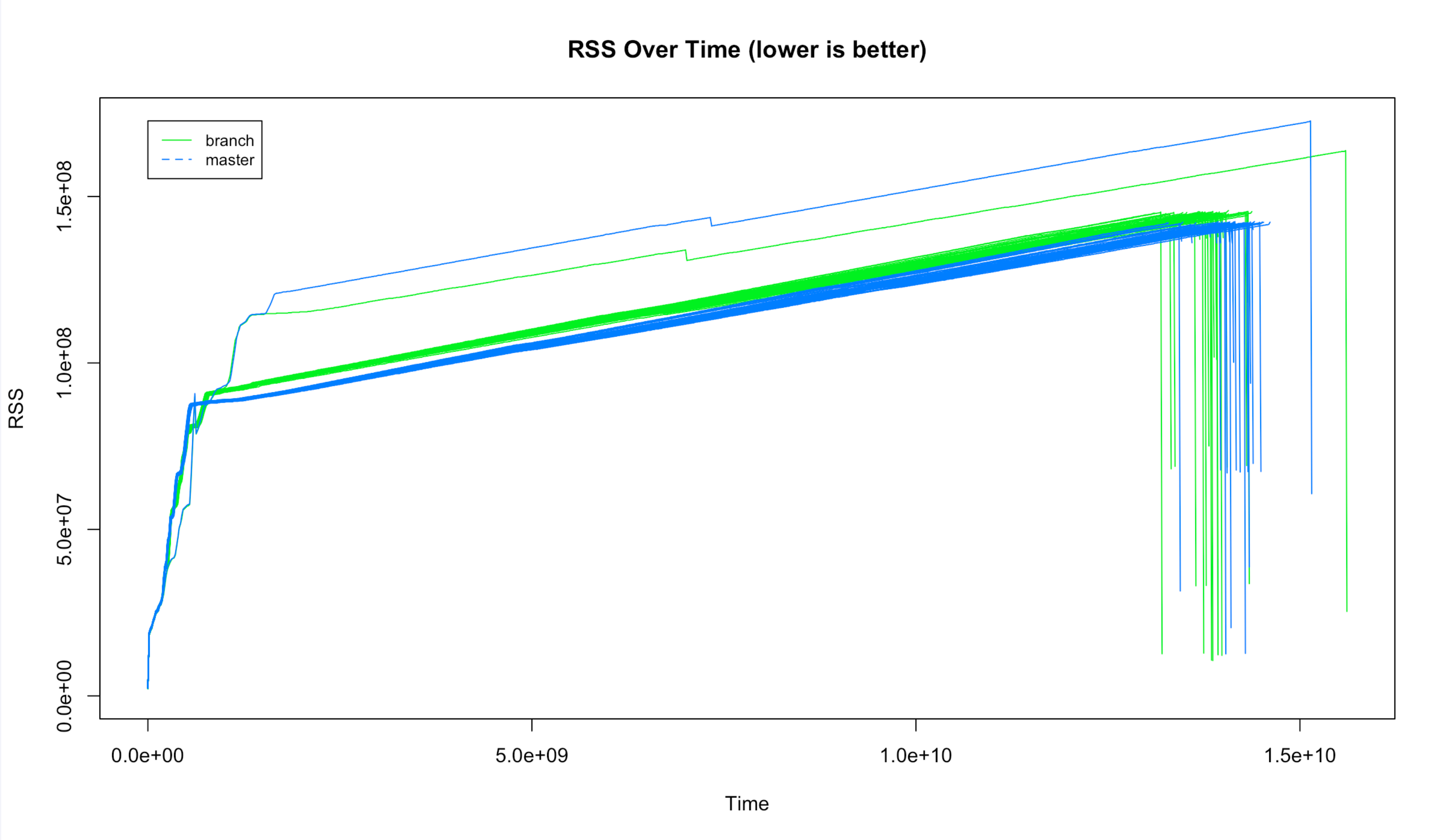
Benchmarks

Methodology

- Benchmarked on bare-metal Ubuntu machine on AWS
- railsbench and rdoc generation was benchmarked using the glibc and jemalloc allocators
- See ticket for more detailed results and analysis: https://bugs.ruby-lang.org/issues/18045

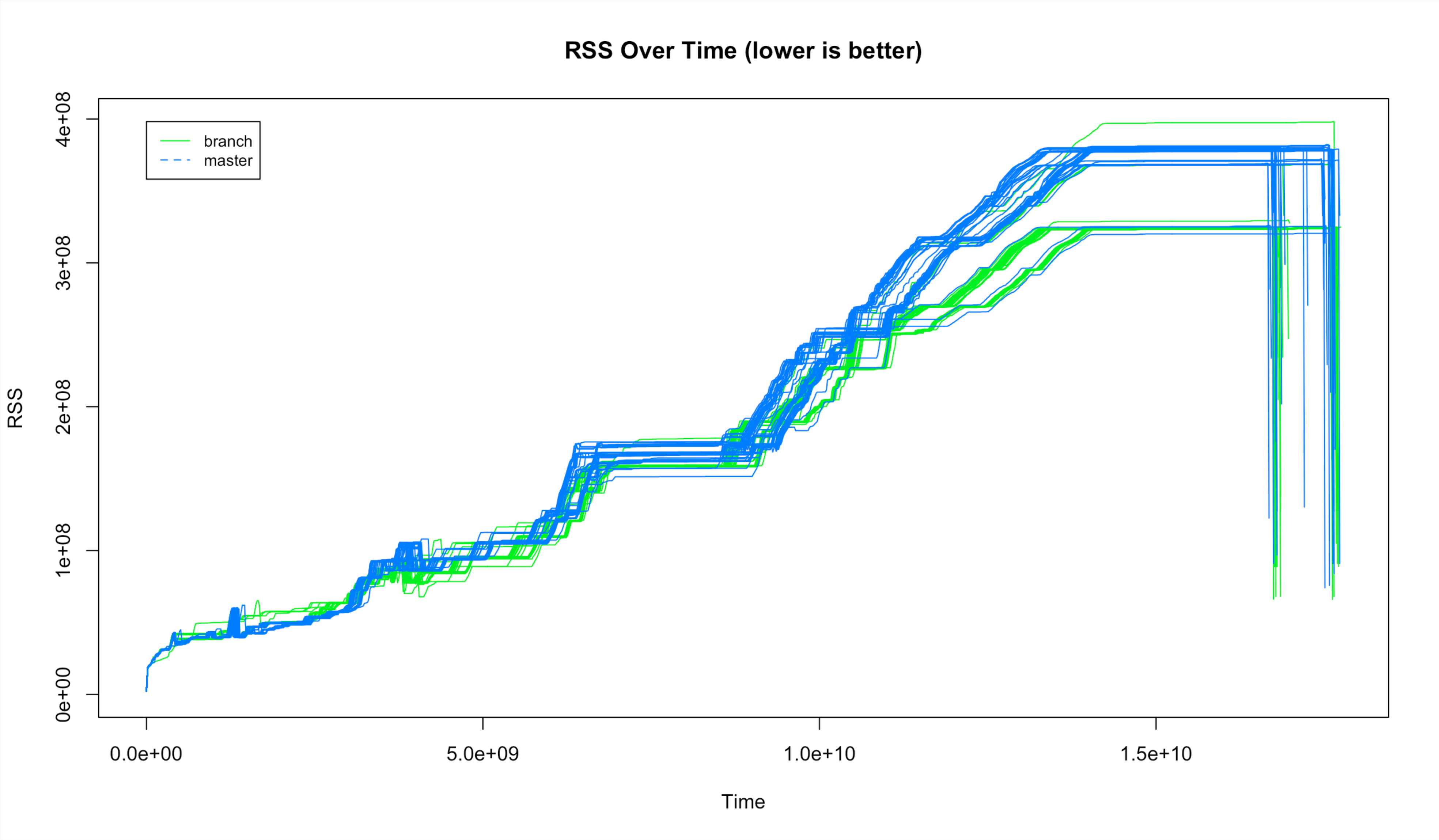
railsbench

- 2% higher max memory usage when using glibc and jemalloc
- No significant performance change when using glibc
- 2.7% faster when using jemalloc



rdoc generation

- 13% lower memory usage than master when using glibc and jemalloc



Liquid & optcarrot benchmarks

- No significant performance difference beyond margin of error

Limitations and future plans

VWA everywhere

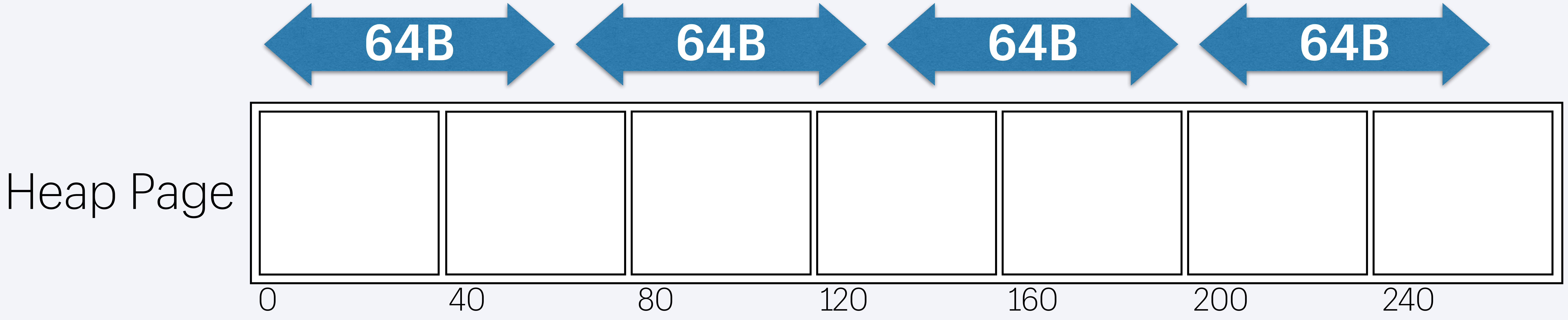
- Currently only classes are using Variable Width Allocation
- Add support for arrays and strings

Resizing objects

- Arrays and strings can resize upwards
- Difficult problem to tackle
- One idea: allocate extra space in a larger size pool and take advantage of compaction to move resized object

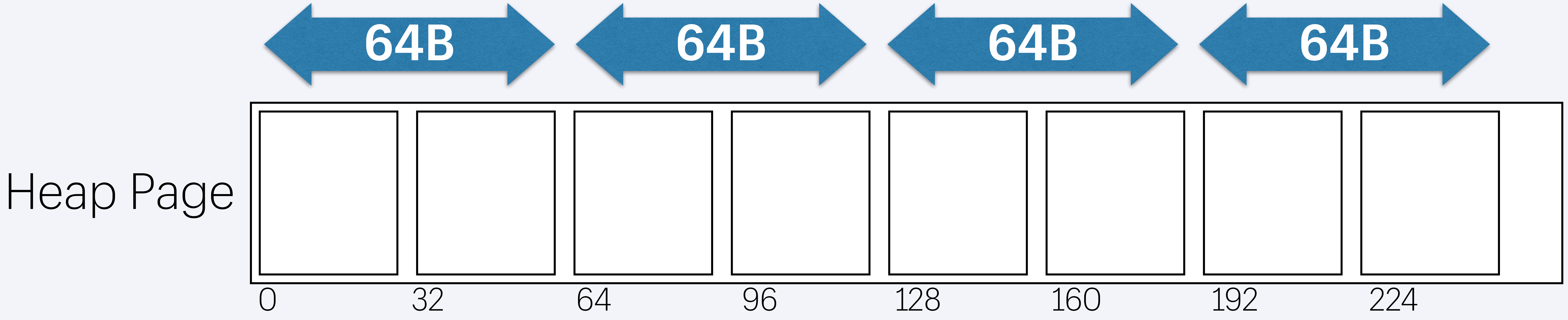
Shrinking RVALUE

- We'd like to shrink RVALUE from 40B to 32B
- Align on 64B cache line boundaries



Shrinking RVALUE

- We'd like to shrink RVALUE from 40B to 32B
- Align on 64B cache line boundaries



Thanks!

shopify