Shenandoah GC:
Your Next Garbage Collector

Zhengyu Gu
Principal Software Engineer
May 6, 2019

Geekbang> 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

OVERVIEW: WHAT'S A GARBAGE COLLECTOR?

Automatic Memory Management

Allocate new objects

obj = new Object()

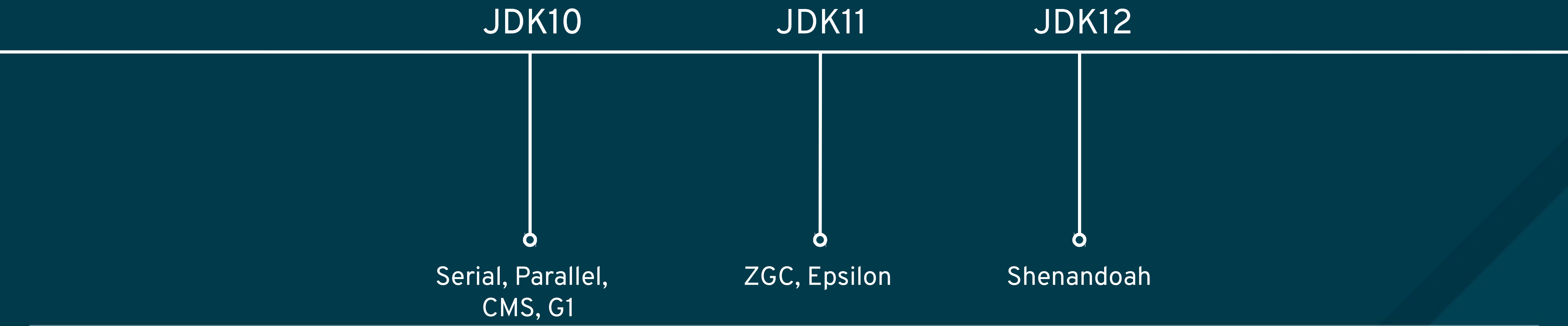
Identify live objects

Reclaim dead objects

~~*delete obj*~~

OVERVIEW: GC LANDSCAPE

Evolution of Garbage Collectors in Hotspot JVM



JDK10

Serial, Parallel, CMS, G1

JDK11

ZGC, Epsilon

JDK12

Shenandoah

SHENANDOAH GC: INTRODUCTION

JEP 189: Shenandoah: A Low-Pause-Time Garbage Collector

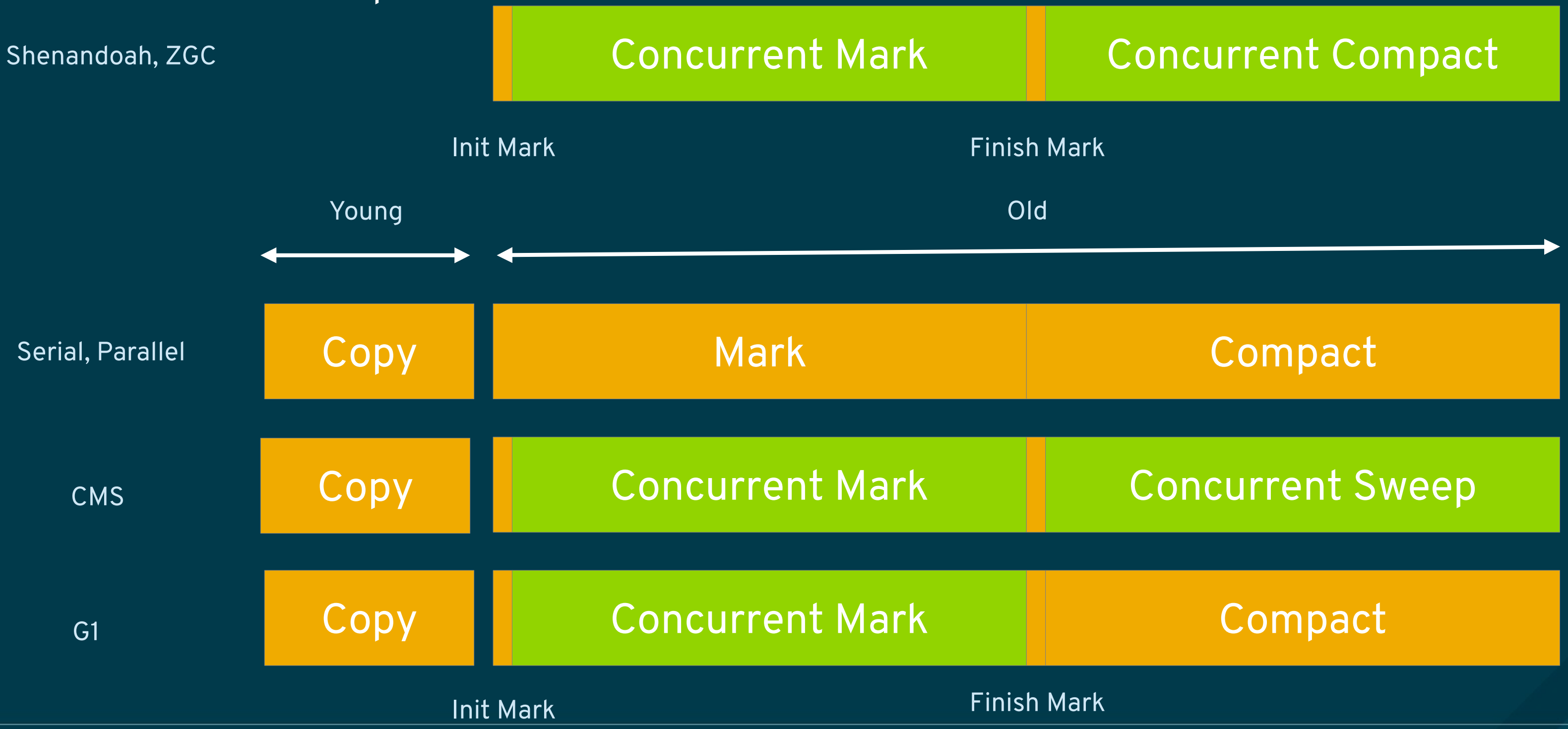
Summary

Add a new garbage collection (GC) algorithm named Shenandoah which reduces GC pause times by doing evacuation work concurrently with the running Java threads. Pause times with Shenandoah are independent of heap size, meaning you will have the same consistent pause times whether your heap is 200 MB or 200 GB.

Shenandoah 实现了拷贝 / 迁移阶段与 Java 线程的并发运行，从而降低 GC 的暂停时间，并使得 Shenandoah 的暂停时间不再与 Java 堆的大小相关。 也就是说，不管 Java 堆是 200MB 还是 200GB ，它的暂停时间基本保持一致。

SHENANDOAH GC: INTRODUCTION

A Concurrent Mark-Compact Collector

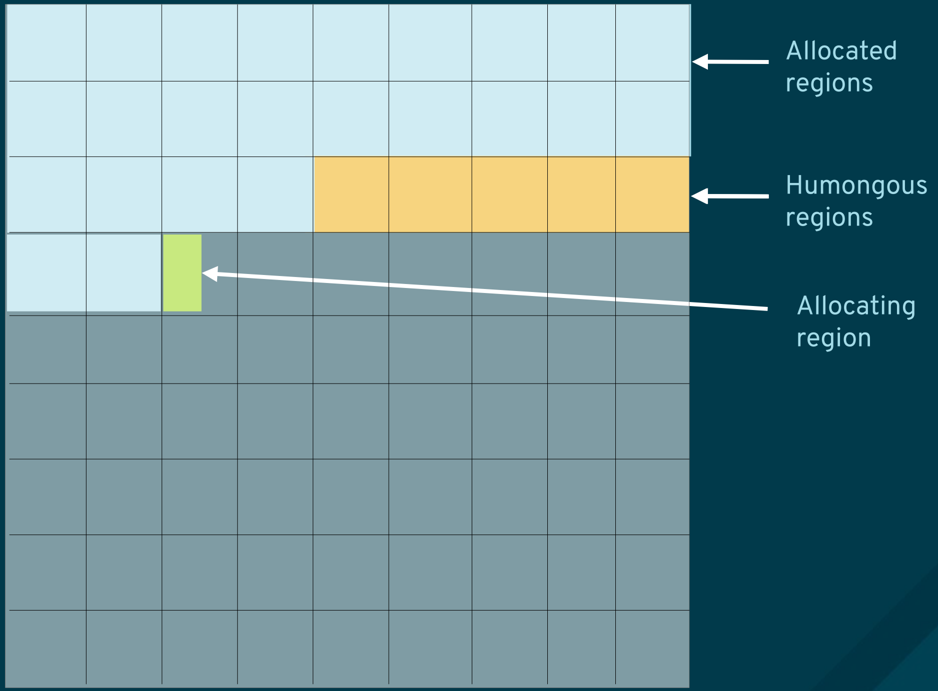


SHENANDOAH GC: INTRODUCTION

A Region Based, Single Generation, Concurrent Mark-Compact Collector

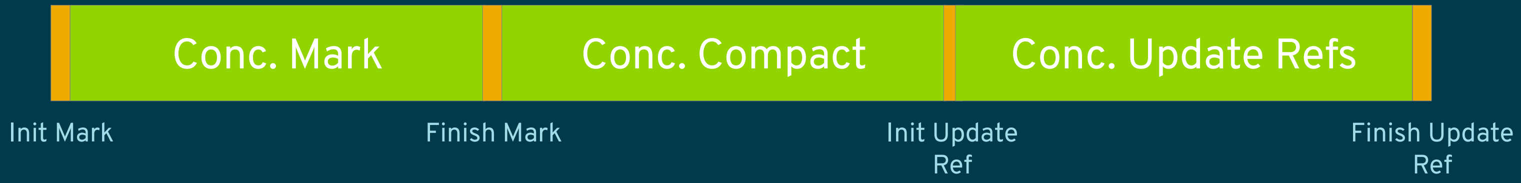
Heap Layout

- Heap is divided into equal sized regions
- Humongous object may occupy multiple regions
- Not generational, no young/old separation



SHENANDOAH GC: INTRODUCTION

A Typical Shenandoah GC Cycle



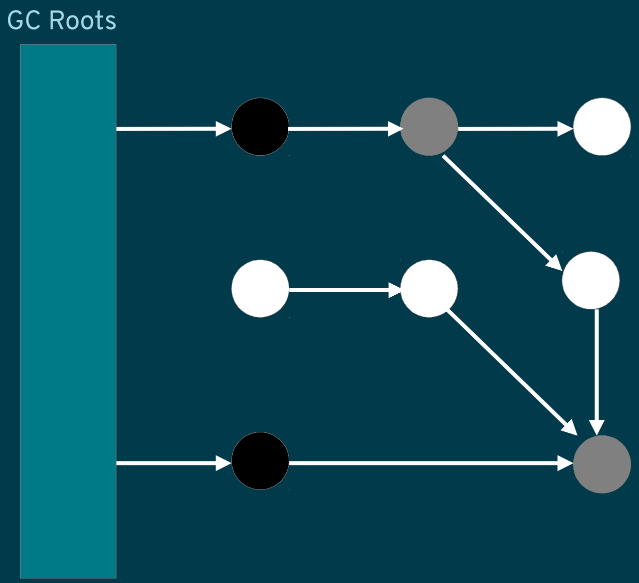
PHASE 1: MARK

Goal

- Identify live objects
- Collect region’s liveness information
- Use region’s liveness information to select collection set (aka. from-space)

STOP-THE-WORLD MARK

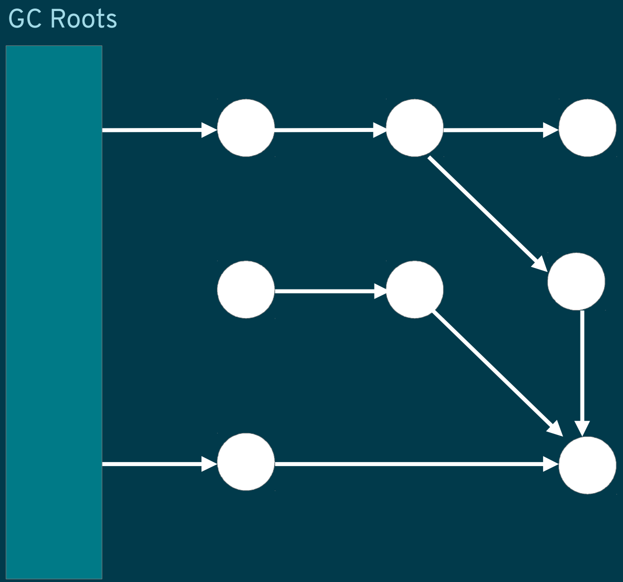
Three Color Abstraction



- White: dead, candidate for reclaiming
- Gray: intermediate state, live, but outbound references are not yet scanned
- Black: live, reachable from the roots

STOP-THE-WORLD MARK

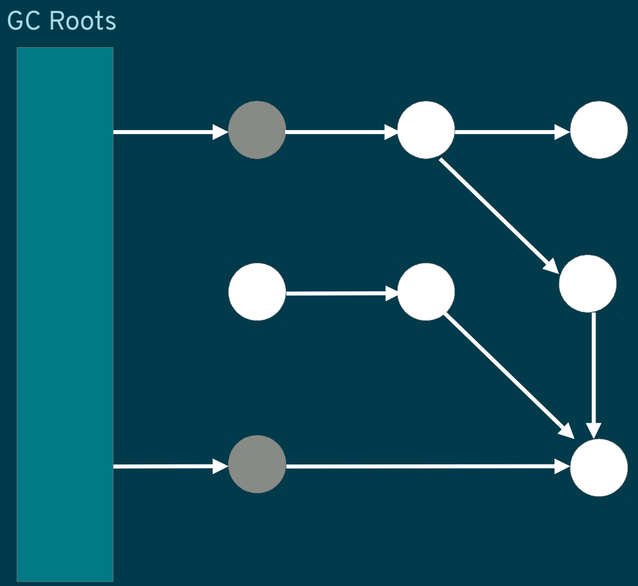
Three Color Abstraction



Step 1: All objects are colored White at mark start

STOP-THE-WORLD MARK

Three Color Abstraction



Step 2: References from GC roots are colored Gray

<<GC Roots>>

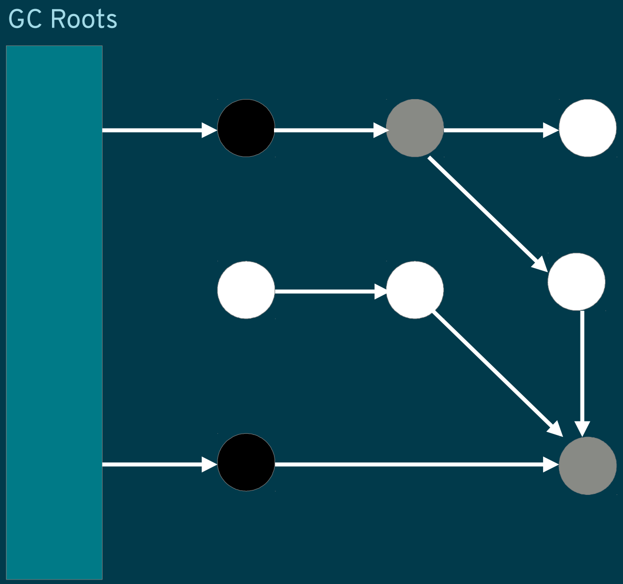
Objects that are directly accessible to the mutators without going through other objects.

<< GC 根 >>

应用程序的线程不经过第三者就可以直接读写到的引用

STOP-THE-WORLD MARK

Three Color Abstraction



Step 3: Scanning Gray references

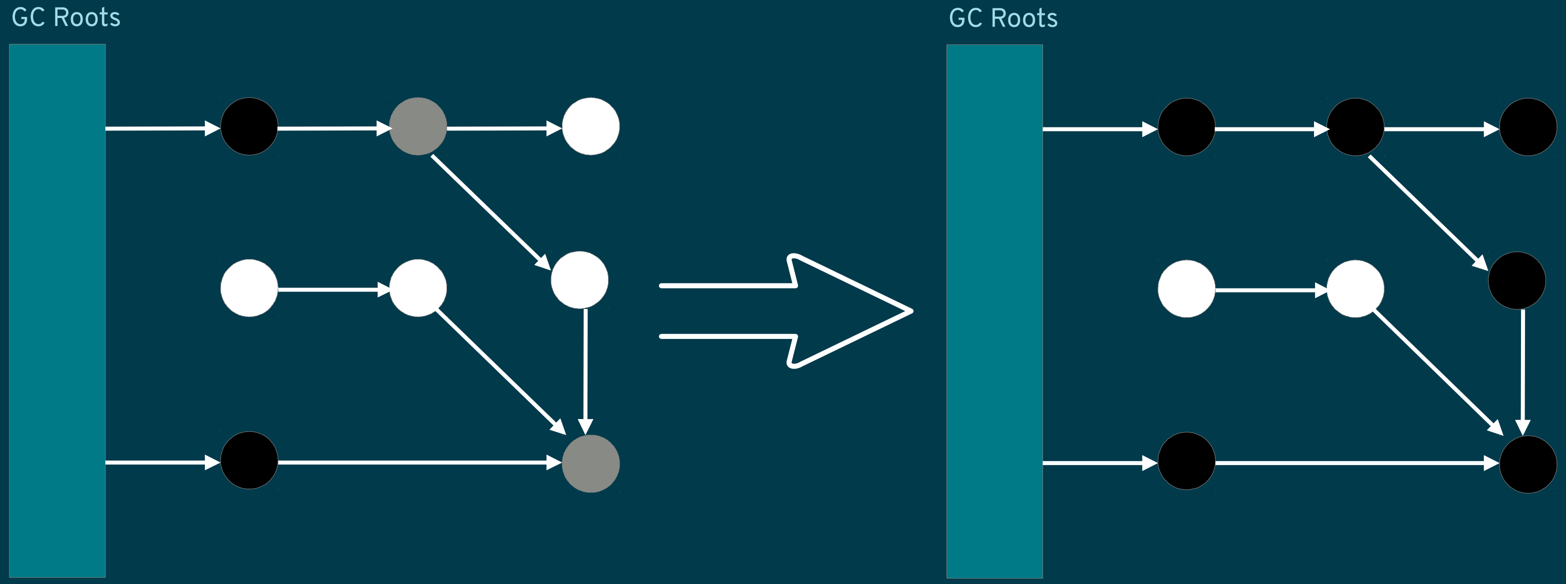
References, that are reachable from Gray, are colored Gray

Scanned Gray references turn into Black

STOP-THE-WORLD MARK

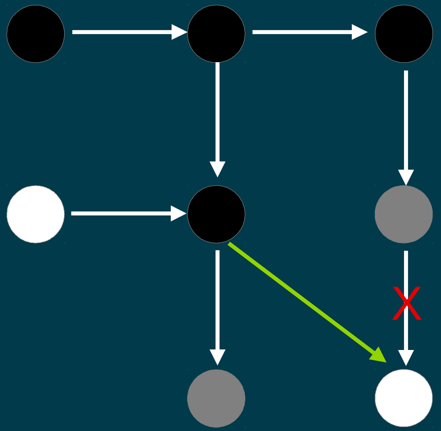
Three Color Abstraction

Step 4. Repeat Step 3, until all objects are either Black (live) or White (dead)



CONCURRENT MARK

Mutator Interference

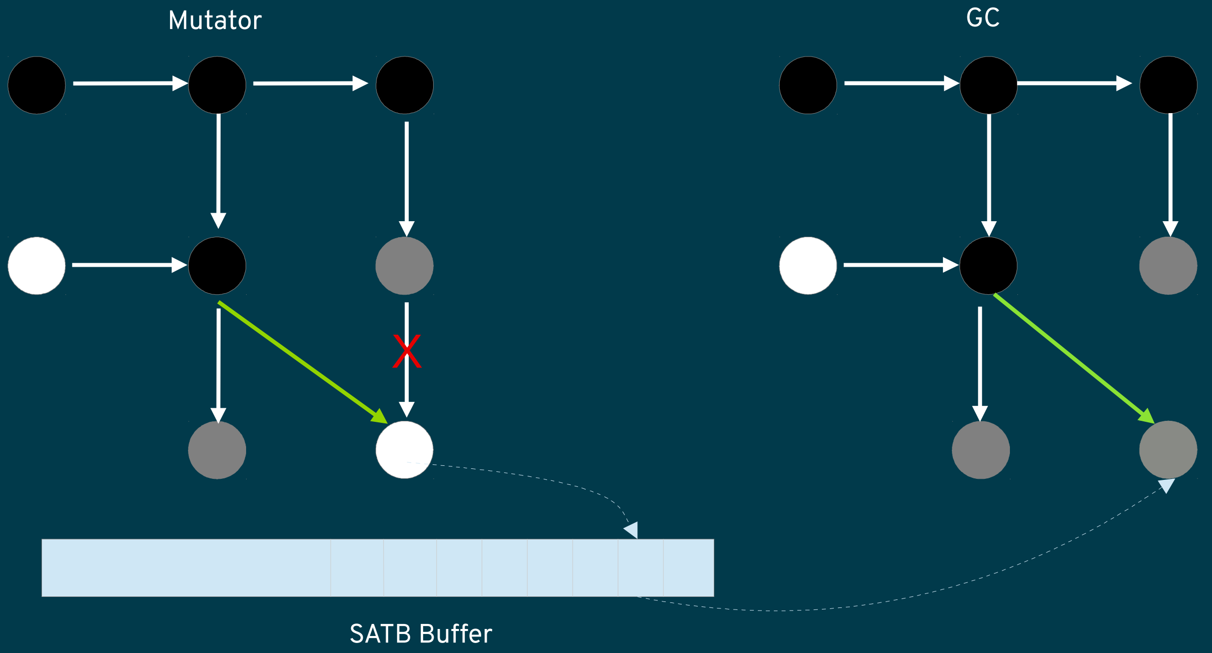


<< Problem >> The lost object

Mutator removes a White reference from a Gray object and inserts into a Black object

CONCURRENT MARK

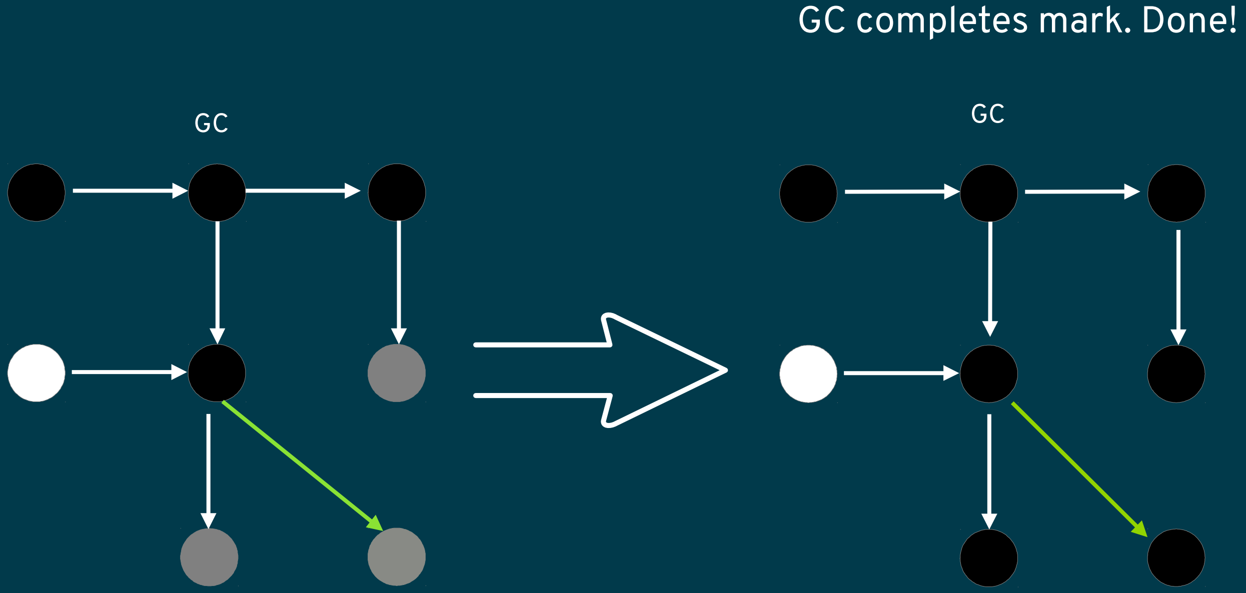
Mutator Interference



<< Solution >>
Use SATB barrier to intercept overwritten value and hand over to GC for scanning

CONCURRENT MARK

Mutator Interference



CONCURRENT MARK

Termination

<< Problem >> Mutators race against GC

- Mutators continue allocating new objects
- GC has to mark them?

CONCURRENT MARK

Termination

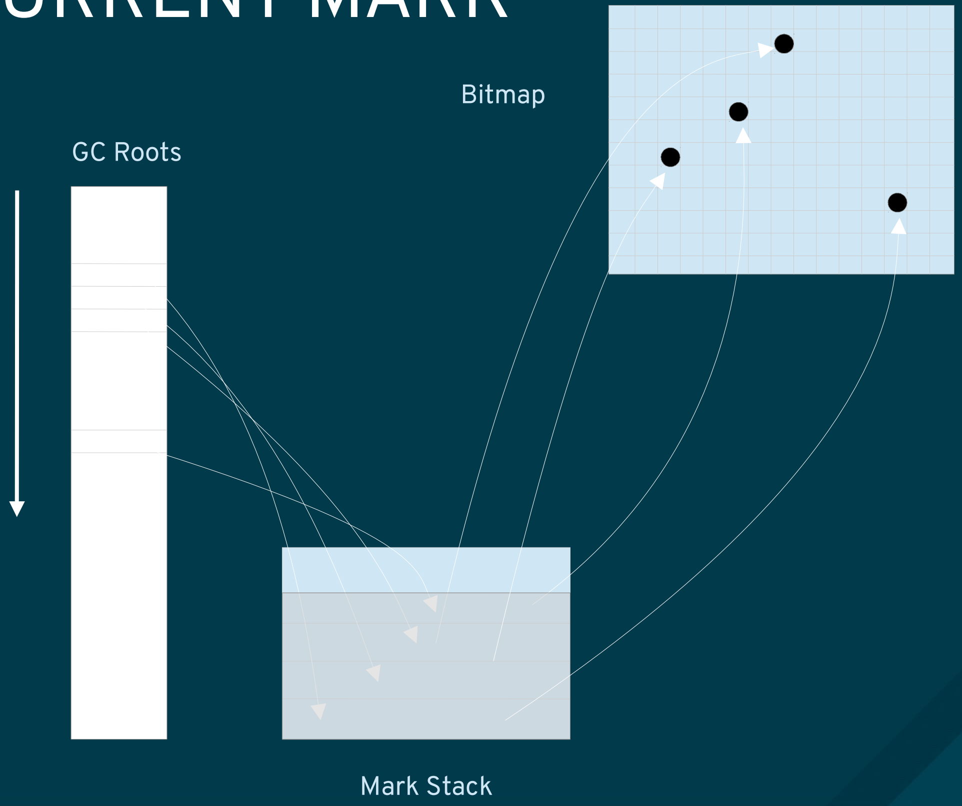
<< Solution >> Snapshot-At-The-Beginning

- Preserve objects that were live at the start of collection
- Newly allocated objects are implicitly live

SHENANDOAH CONCURRENT MARK

Step 1: Pause Init Mark

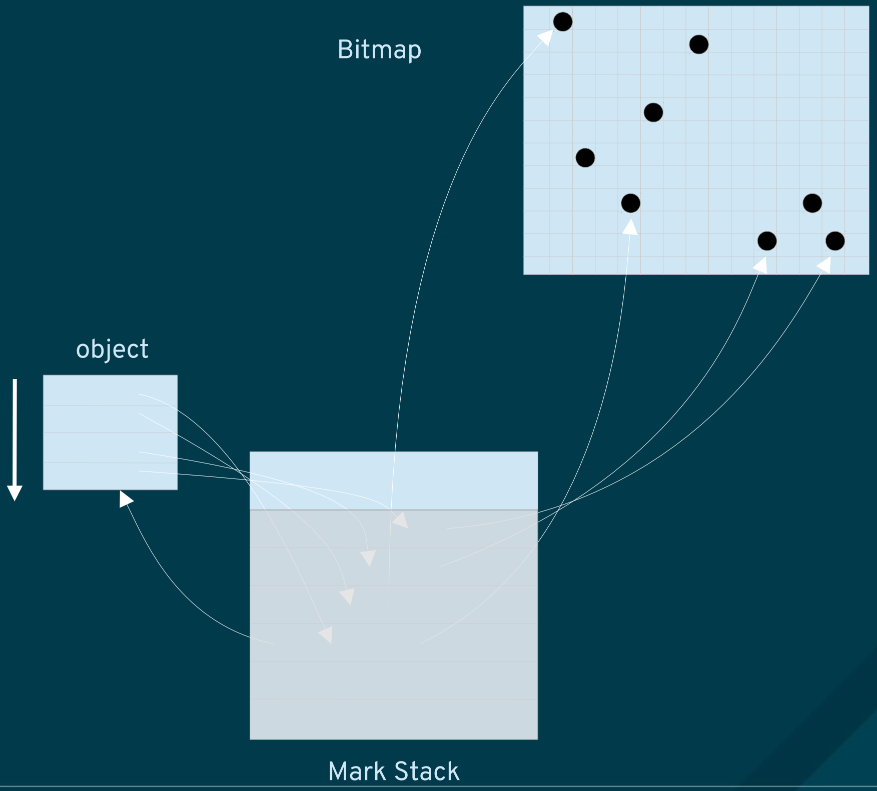
- Scan GC roots to seed mark
- Activate SATB to intercept mutator interference



SHENANDOAH CONCURRENT MARK

Step 2: Concurrent Mark

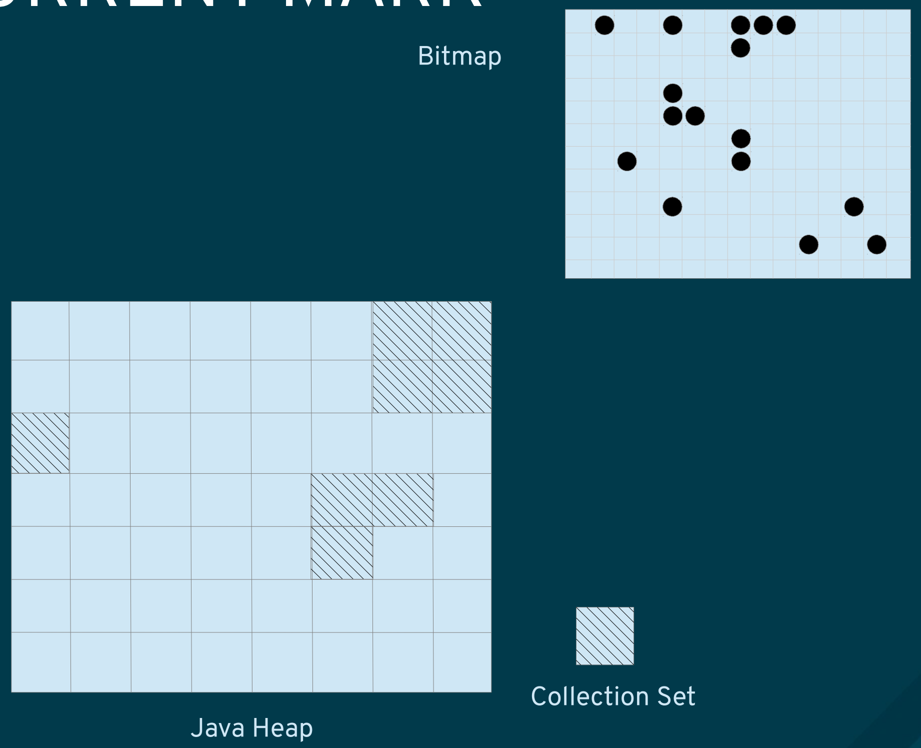
- Pop an object off mark stack
- Scan its outbound references
- Push references into mark stack if they are not yet marked
- Repeat until mark stack is empty



SHENANDOAH CONCURRENT MARK

Step 3: Pause Finish Mark

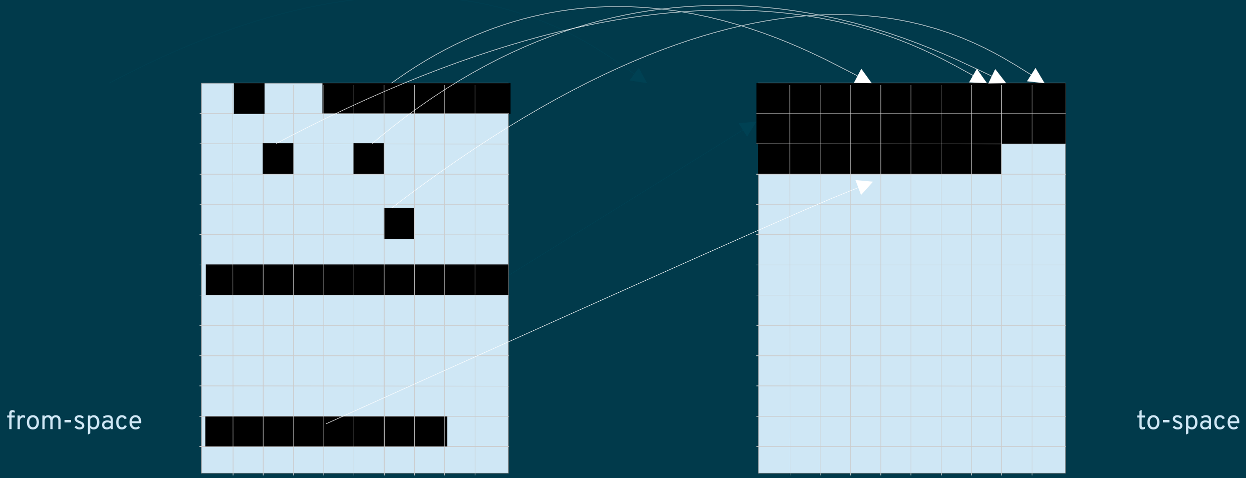
- Process intercepted objects
- Finish marking
- Deactivate SATB barrier
- Select collection set



PHASE 2: COMPACT

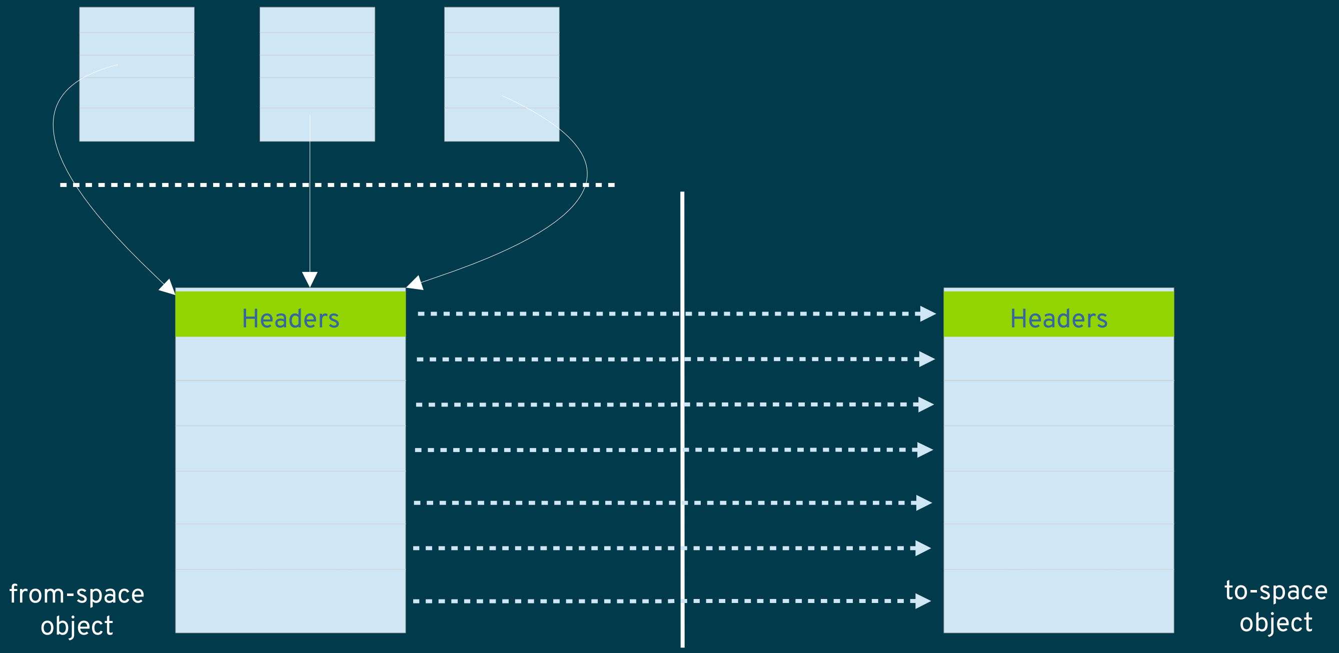
Goal

- Copy live objects out of from-space and compact them into to-space



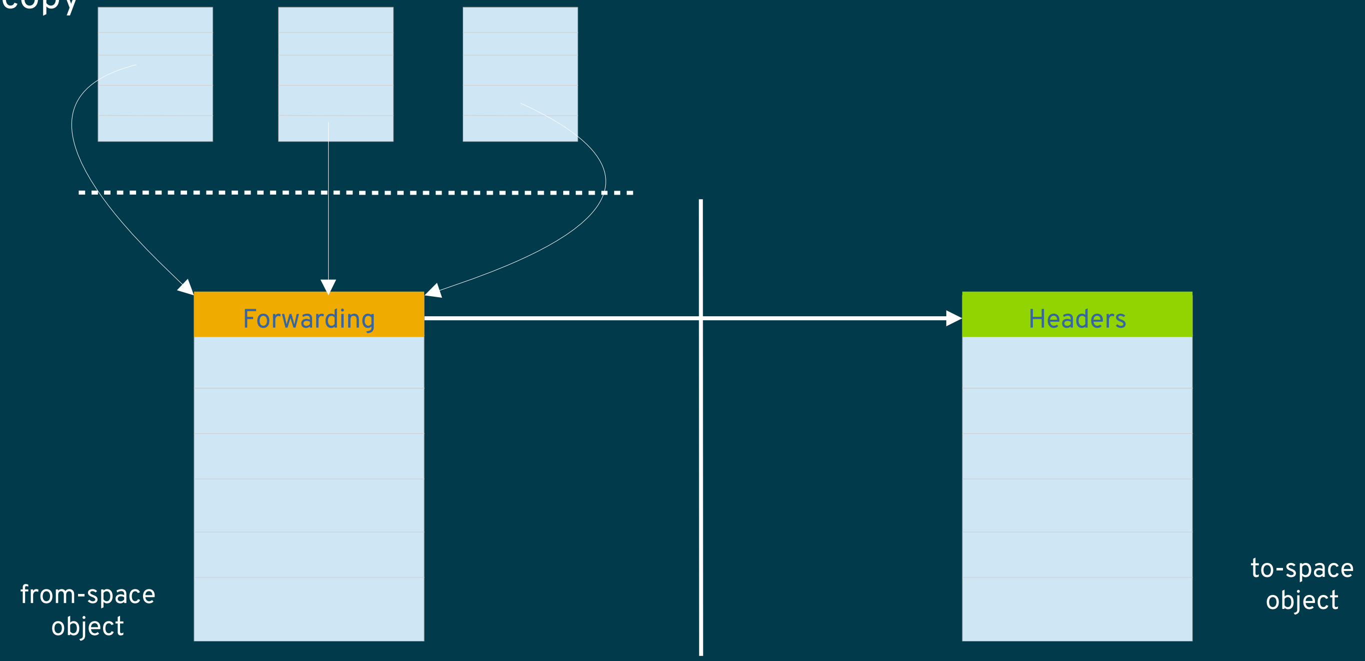
STOP-THE-WORLD COPY

Step 1: Copy a live object in from-space to to-space



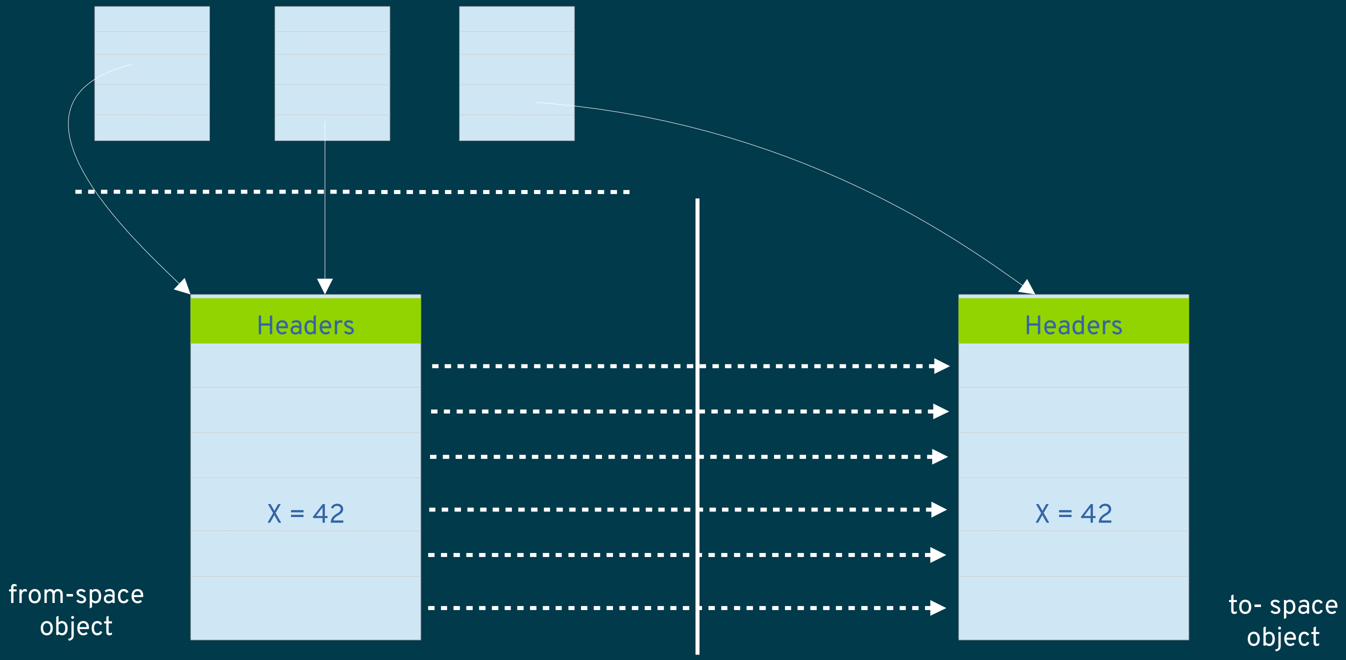
STOP-THE-WORLD COPY

Step 2: Use from-space object's header as forwarding pointer to to-space copy



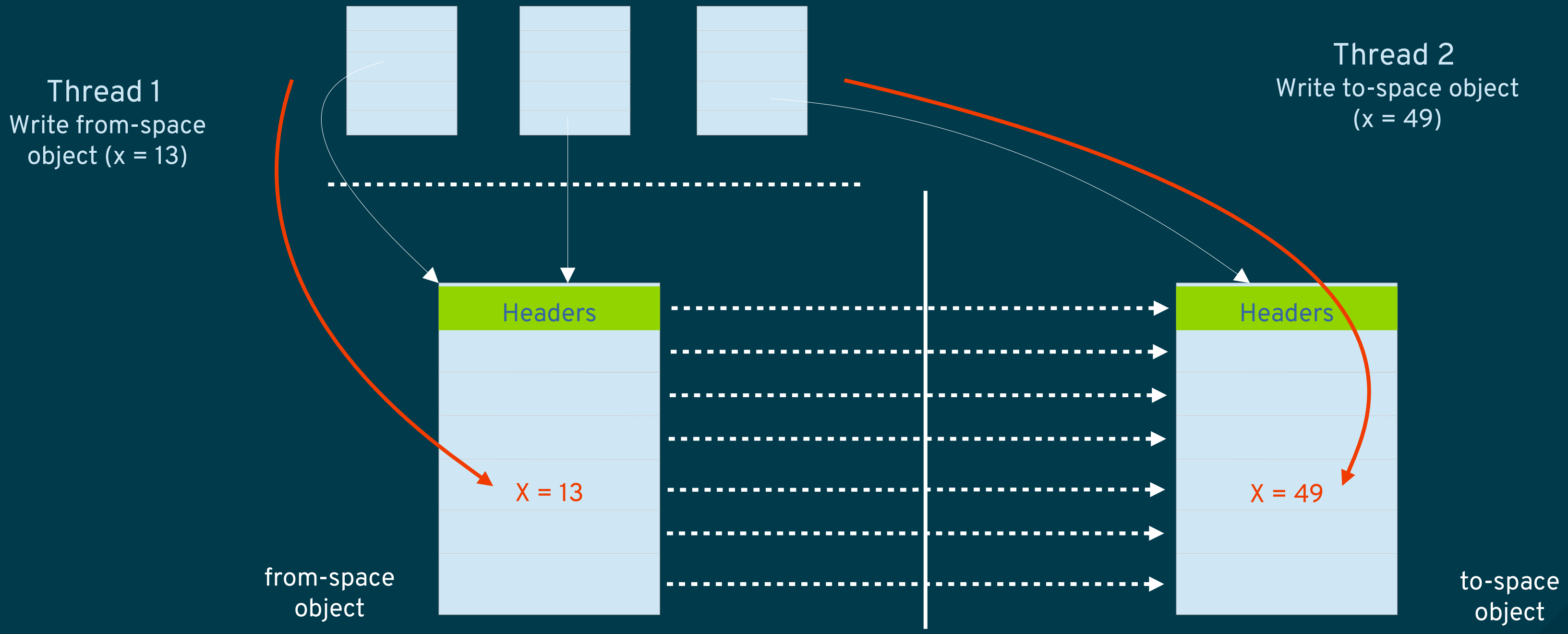
CONCURRENT COPY

Copy a live from-space object to to-space



CONCURRENT COPY

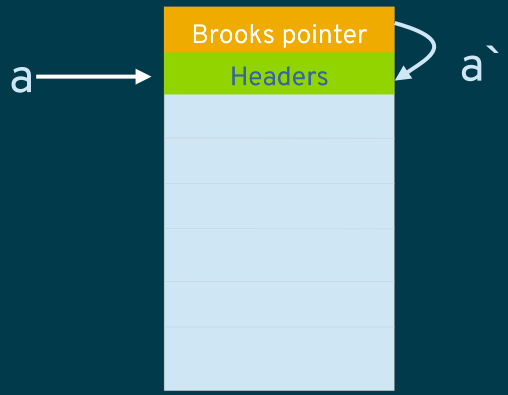
Mutator Problems



SHENANDOAH CONCURRENT COPY

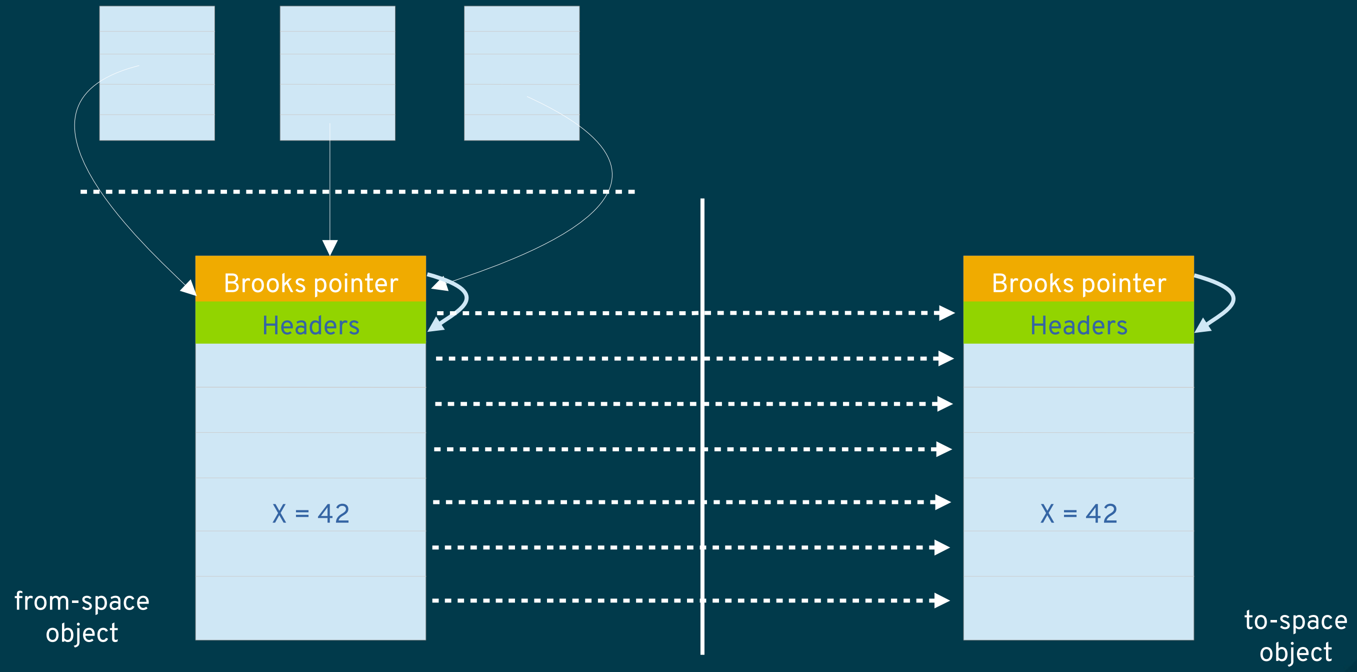
Brooks Pointers

An indirection, initially points to itself
Object access via Brooks pointer:
Load a → Load a’s Brooks pointer a` → Load a`



SHENANDOAH CONCURRENT COPY

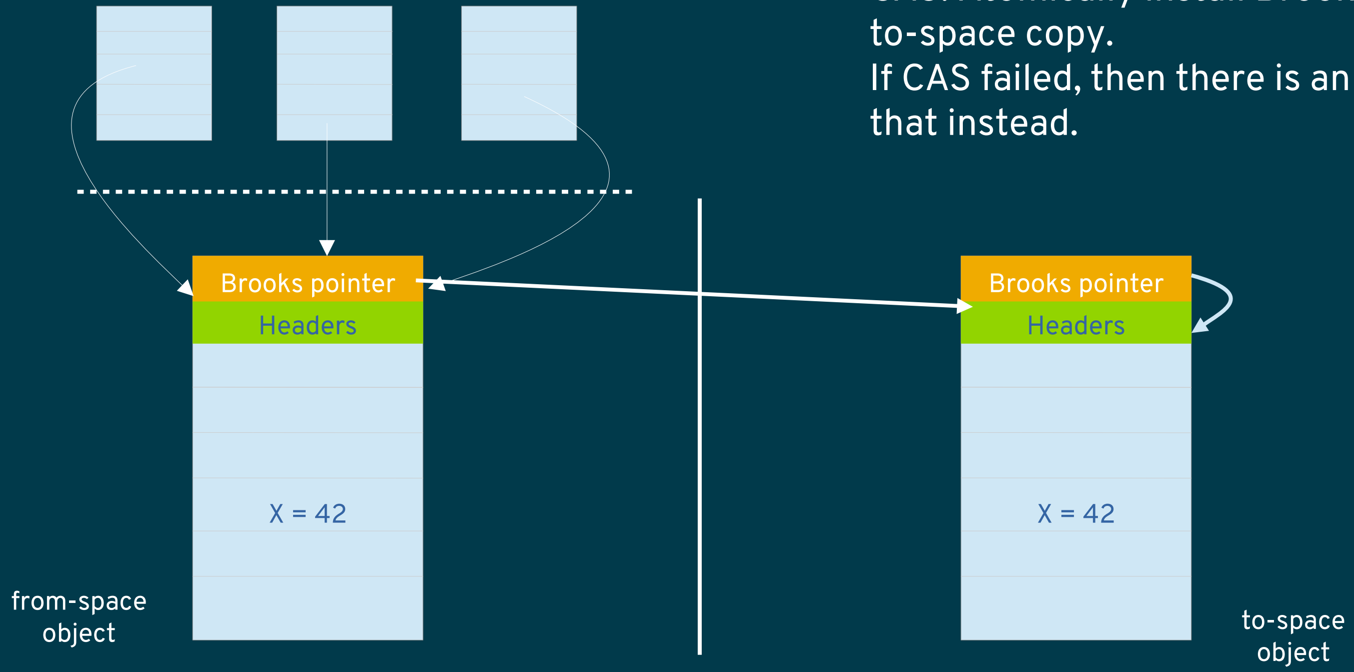
Step 1. Copy a live from-space object to to-space



SHENANDOAH CONCURRENT COPY

Step 2: Install Brooks pointer

CAS! Atomically install Brooks pointer to point to to-space copy.
If CAS failed, then there is an existing copy, use that instead.



SHENANDOAH CONCURRENT COPY

Concurrent Copy Pseudo Code

```
// Concurrent copy always returns a to-space object
func concurrent_copy (obj) {
    var copy = copy(obj);                       // make to-space copy
    if (CAS(brooks-ptr-addr(obj), obj, copy)) {
        return copy;                            // success!
    } else {
        return brooks-ptr(obj);                 //  someone just beat us to it
    }
}
```

SHENANDOAH CONCURRENT COPY

to-space Invariant

When to copy?

- Copy on Write

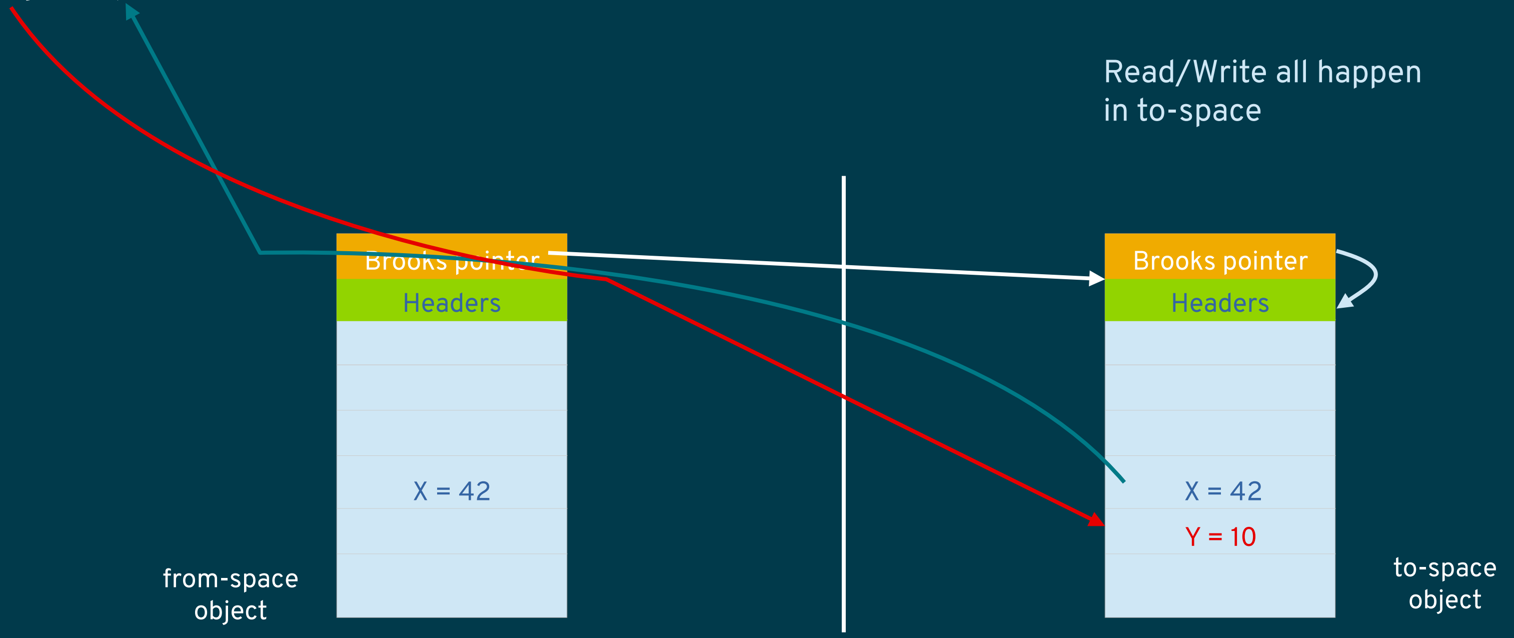
  Weak to-space Invariant (JDK12)
- Copy on Read

  Strong to-space Invariant (JDK13+)

SHENANDOAH CONCURRENT COPY

Strong to-space Invariant

Read/Write all happen in to-space



SHENANDOAH CONCURRENT COPY

Load-Reference-Barrier Pseudo Code

```
// load_reference_barrier always returns a to-space object
func load_reference_barrier (obj) {
    var fwd = brooks-ptr(obj);                 // read obj’s Brooks pointer
    if (fwd != obj) {
        return fwd;                            // fwd is obj’s to-space copy
    } else if (!in_collection_set(obj)) {
        return obj;                            // obj is in to-space
    } else {
        return concurrent_copy(obj);           //  copy obj and return to-space reference
    }
}
```

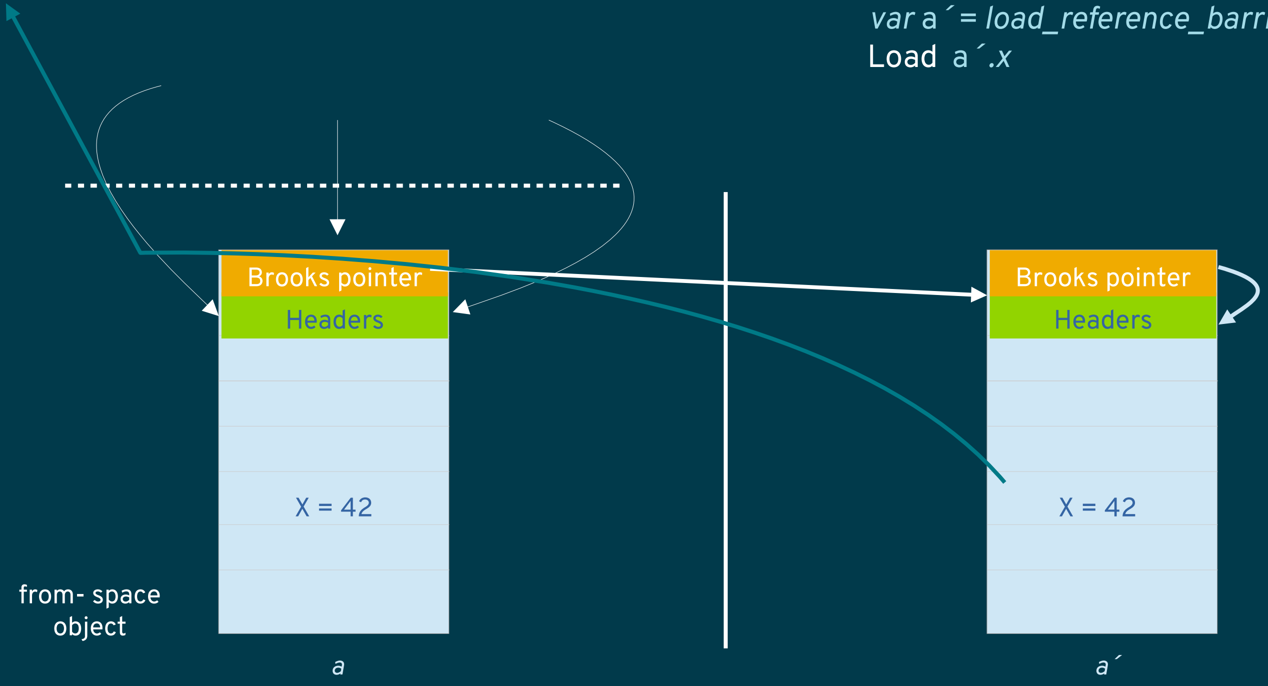
SHENANDOAH CONCURRENT COPY

Read via Load-Reference-Barrier

Load *a.x*

var a*´= load_reference_barrier(a)*

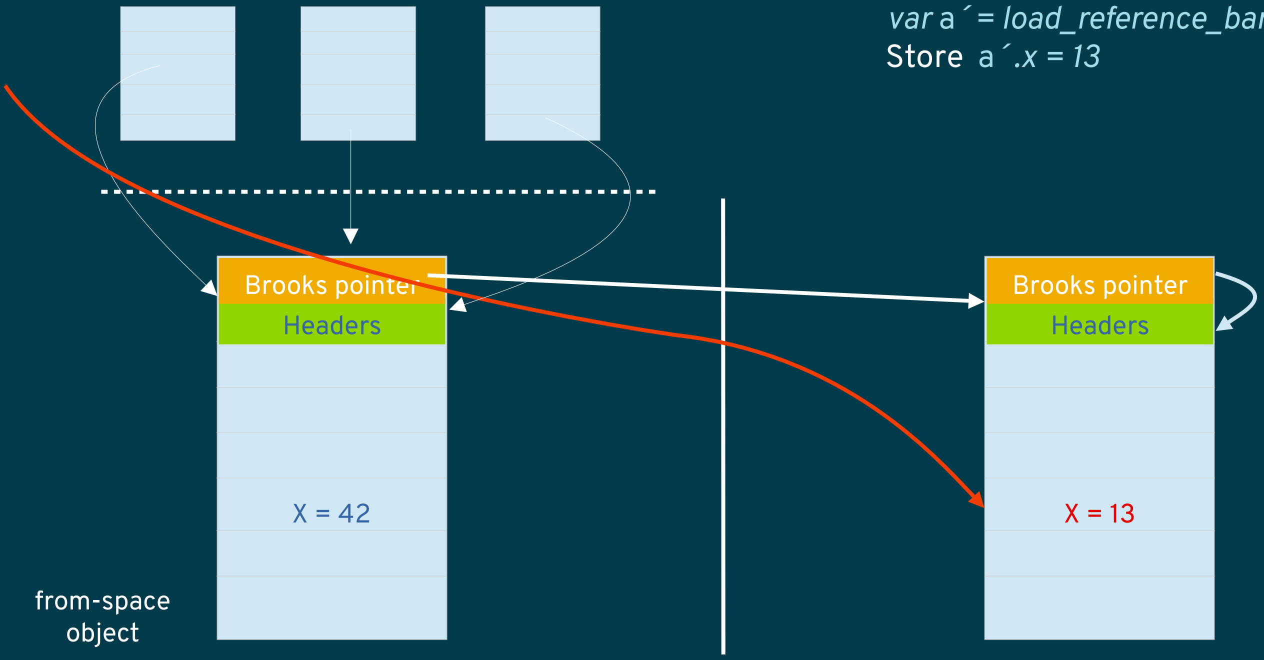
Load a*´.x*



SHENANDOAH CONCURRENT COPY

Write via Load-Reference-Barrier

Store *a.x = 13*
 var a*´ = load_reference_barrier(a)*
 Store a*´.x = 13*



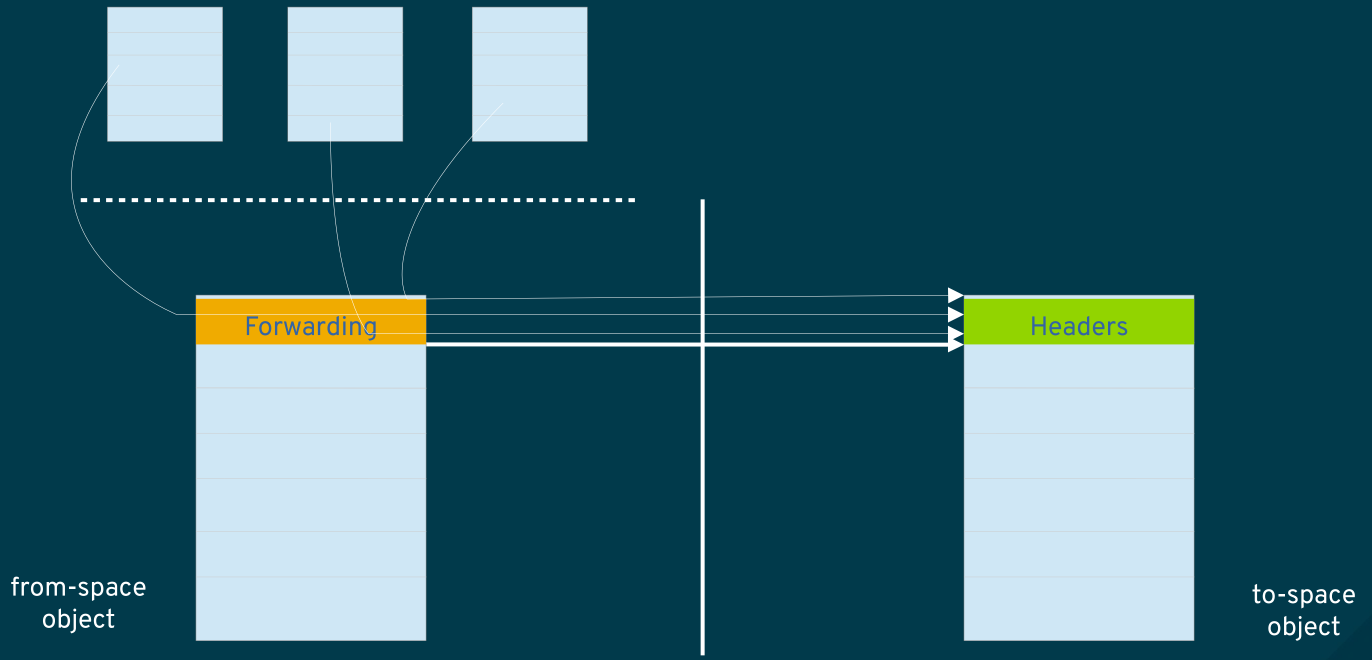
PHASE 3: UPDATE REFERENCES

Goal

- Update references that point to from-space objects with their to-space copies
- Upon completion, from-space regions are reclaimed

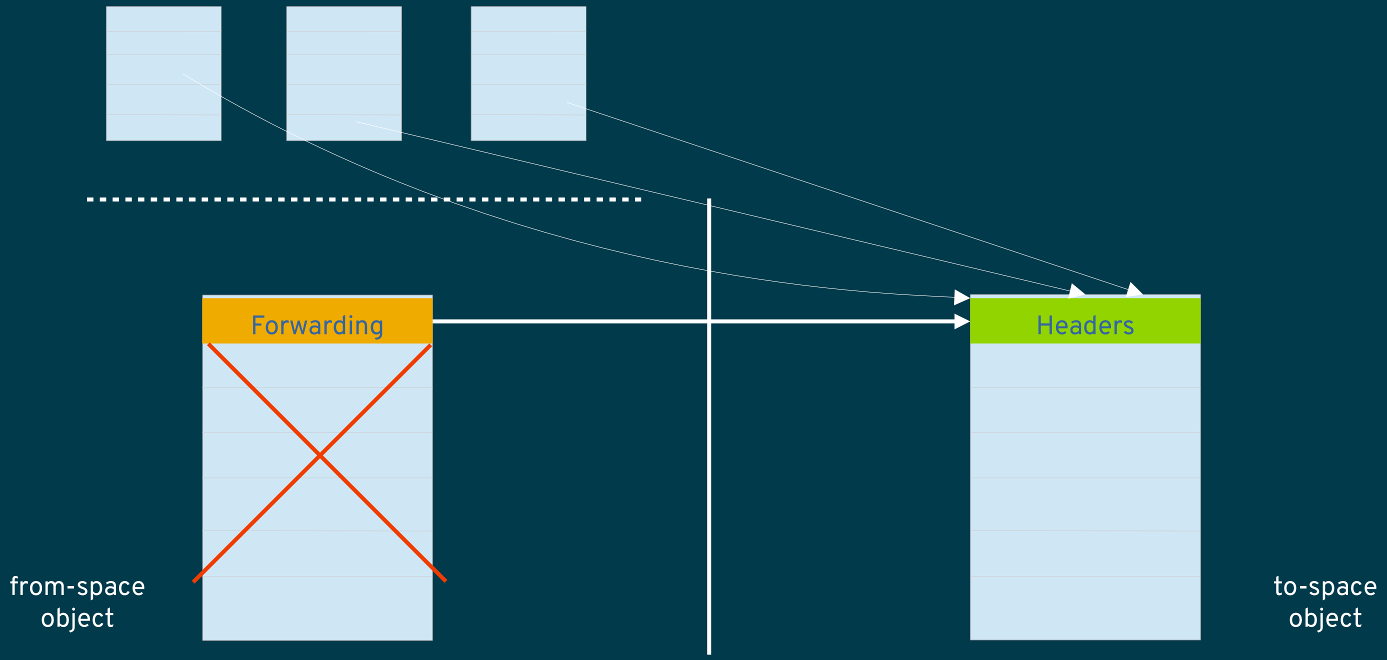
STOP-THE-WORLD UPDATE REFERENCES

Step 3: Walk the heap, update references with forwarding pointer to to-space objects



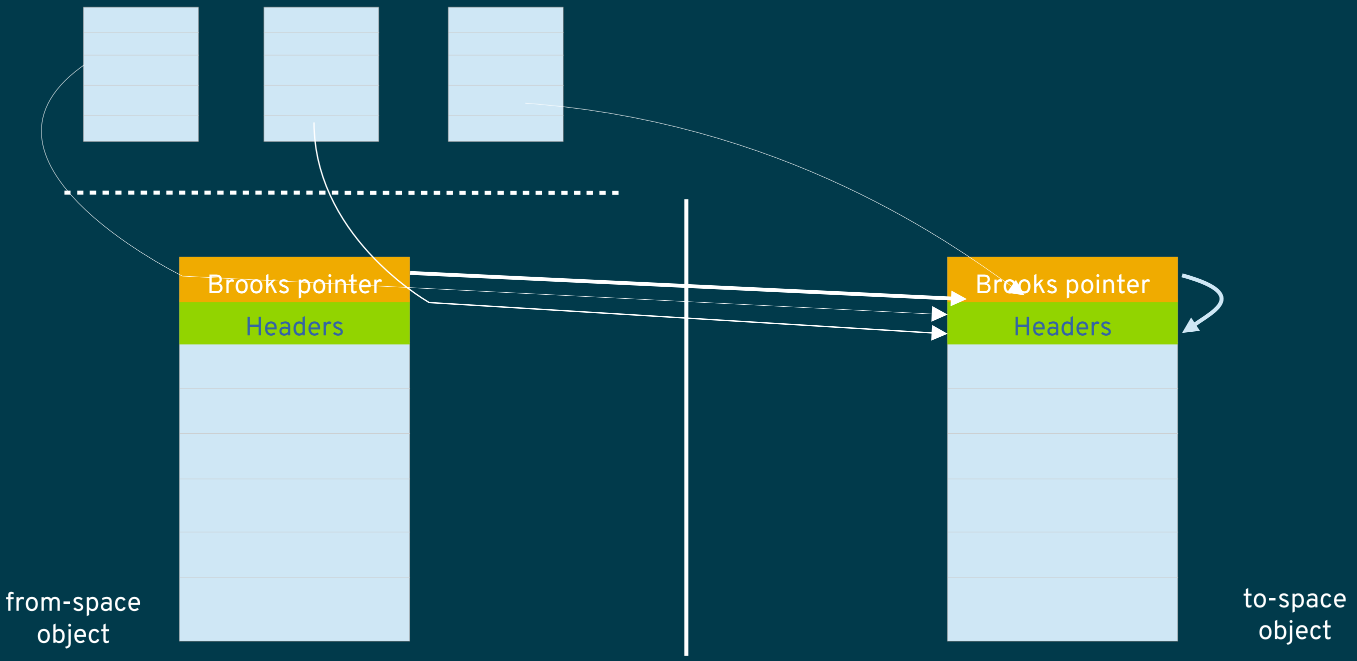
STOP-THE-WORLD UPDATE REFERENCES

Step 4: Done! from-space object is ready for reclaiming



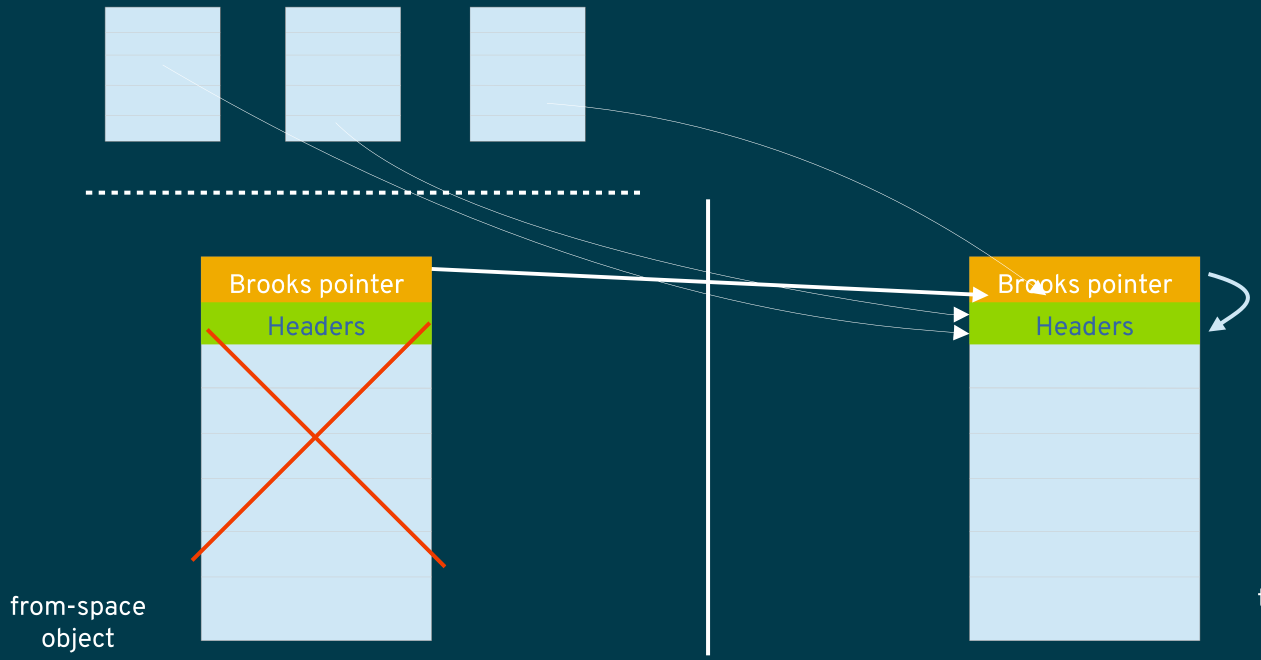
SHENANDOAH CONCURRENT UPDATE REFERENCES

Walk the heap, update references with Brooks pointer to to-space objects



SHENANDOAH CONCURRENT UPDATE REFERENCES

Done! from-space object is ready for reclaiming



RESULTS

SPECJBB 2015 with 100GB Java Heap

```
[info][gc] GC(81) Concurrent reset 87020M->87020M(102400M) 9.753ms
[info][gc] GC(81) Pause Init Mark 1.658ms
[info][gc] GC(81) Concurrent marking 87020M->88597M(102400M) 93.430ms
[info][gc] GC(81) Pause Final Mark 2.530ms
[info][gc] GC(81) Concurrent cleanup 88591M->85493M(102400M) 0.110ms
[info][gc] GC(81) Concurrent evacuation 85493M->88394M(102400M) 96.460ms
[info][gc] GC(81) Pause Init Update Refs 0.470ms
[info][gc] GC(81) Concurrent update references 88460M->89838M(102400M) 79.843ms
[info][gc] GC(81) Pause Final Update Refs 1.075ms
[info][gc] GC(81) Concurrent cleanup 89838M->7375M(102400M) 0.390ms
```

YOUR NEXT GARBAGE COLLECTOR

Try It!

- Available in all major OpenJDK releases (8, 11, 12 and 13)

  Yes! Shenandoah is available in Red Hat OpenJDK8/11 releases

  JDK8 shipped since RHEL 7.4/Fedora 24

  JDK11 shipped since Fedora 27

- Not just for large heap

  Support Compressed Oops

- Perform well in Container environment

  Automatic memory scaling

  Allocation spike mitigation

YOUR NEXT GARBAGE COLLECTOR

Join us, contribute to Shenandoah

- Shenandoah Wiki

  https://wiki.openjdk.java.net/display/shenandoah

- Mailing list

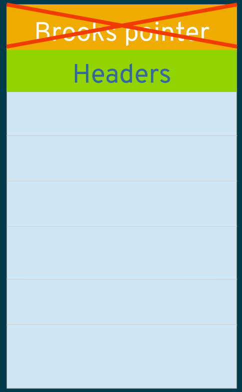
  shenandoah-dev@openjdk.java.net

WHAT'S NEW AND WHAT WE ARE WORKING ON?

- Switched to strong to-space invariant (JDK13)

  Will backport to OpenJDK8, 11, 12

- Eliminating extra space for Brooks pointer
- Concurrent class unloading
- Concurrent reference processing



LOAD-REFERENCE-BARRIER

Fastpath

```
          testb    $0x1, 0x20(%r15)                 // Is heap stable?
          jne      HAS-FORWARDING-OBJECTS
BACK:
   // ... actual load follows ...
HAS-FORWARDING-OBJECTS:
          mov      %rsi,  %r10
          shr      $0x15, %r10
          movabs $0x7fcf64099447, %r11
          cmpb     $0x0, (%r11, %r10 ,1)     // Is  object in "to" space?
          je       BACK
          mov      -0x8(%r8), %rdi           // Nope
          cmp      %r8, %rdi                 // But does it have "to" space copy?
          je       LRB-SLOWPATH              // Nope,  go to Slow-path
          mov      %rdi, %r8                 // Load "to" space reference
          jmp      BACK
```

LOAD-REFERENCE-BARRIER

Slowpath

```
stub load_reference_barrier (obj) {
  if (evacuation-in-progress &&
     in-collection-set(obj)    &&        // obj is still in "from" space
     brooks-ptr-to-self(obj) ) {         // does not have "to" space copy
    var copy = copy(obj);
    if (CAS(brooks-ptr-addr(obj), obj, copy)) {
        return copy;                      // success!
    } else {
        return brooks-ptr(obj);           //  someone just beat us to it
    }
}
```

QUESTIONS ?

redhat.

THANK YOU

- G+ plus.google.com/+RedHat
- in linkedin.com/company/red-hat
- ▶ youtube.com/user/RedHatVideos
- f facebook.com/redhatinc
- twitter.com/RedHat