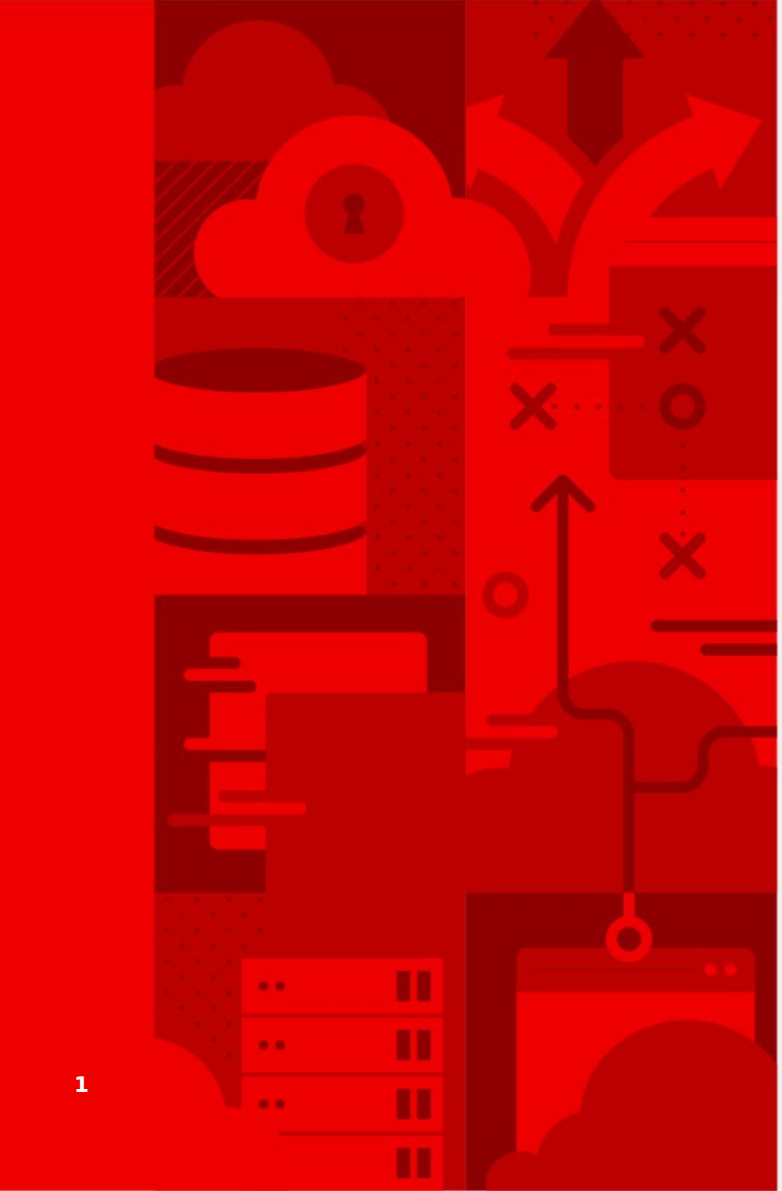


Shenandoah GC 2.0

Roman Kennke
@rkennke

Red Hat

Overview

- Shenandoah introduction
- Old vs new barrier scheme
- New platforms
- Elimination of forwarding pointer word
- Self-fixing barriers

Shenandoah Introduction

Shenandoah is:

- A garbage collector in OpenJDK
- Concurrent GC
- Universal
- Aimed to solve the GC pause problem
  - Regardless of heap size
  - Slow/tiny hardware, cloud, constrained environments, etc

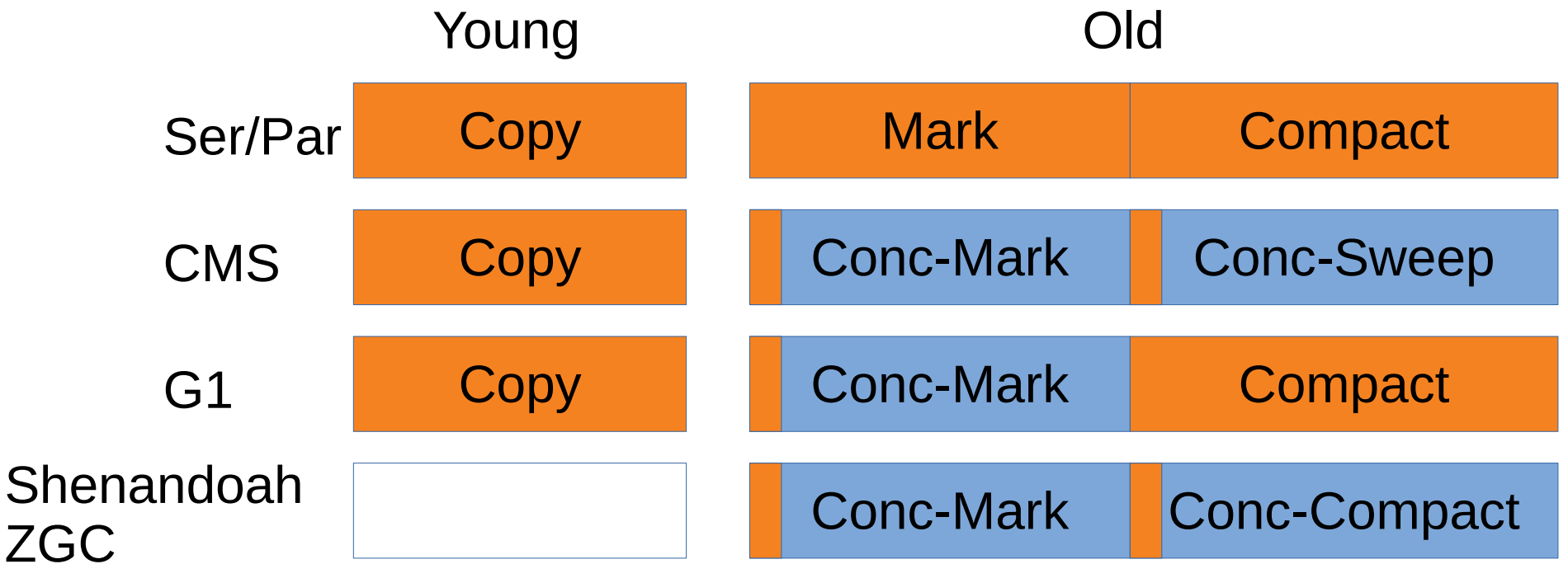
Shenandoah Introduction

You may have heard:

- That Shenandoah requires more memory
- Has complicated barriers
- Lacks some optimizations

→ No longer true

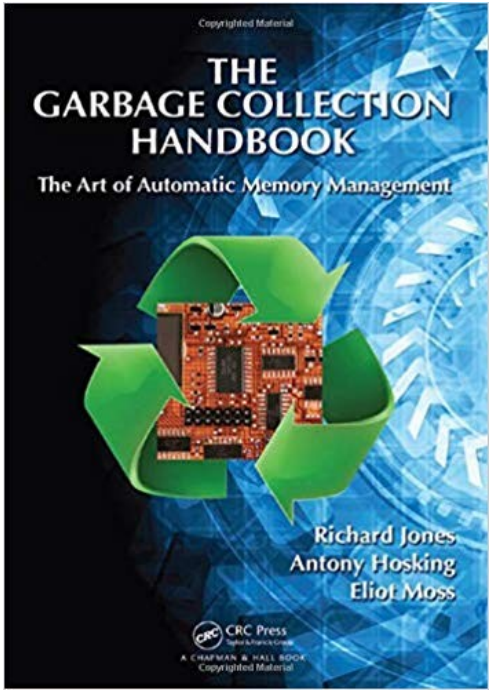
Shenandoah Introduction



Concurrent Marking

Not covered here, no time, sorry

If you’re interested, it’s using “Snapshot at the beginning” algorithm, similar to G1



Red Hat

Concurrent Compaction

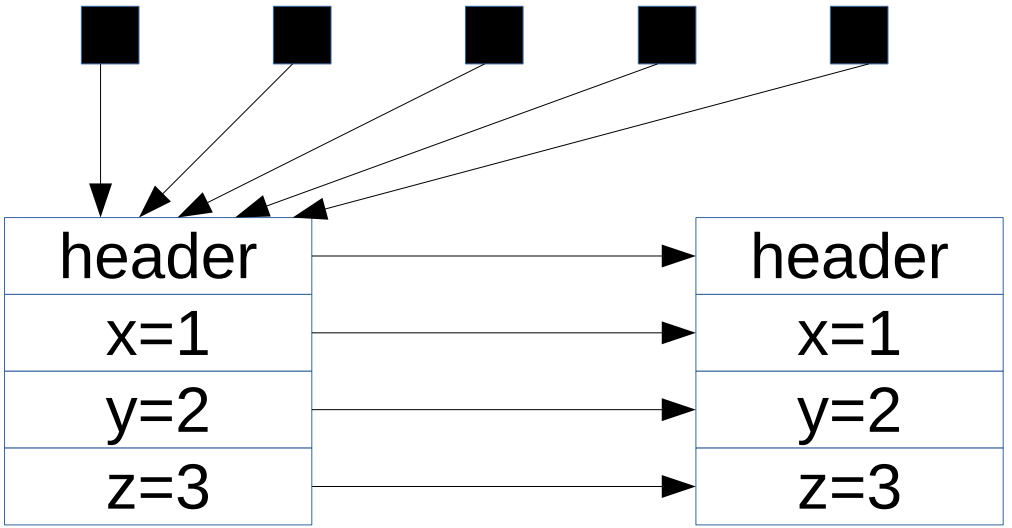
Compaction:

1. Copy all reachable objects from the collection set to empty regions
2. Adjust all references to old copies to point to new copies

→ This is easy while the program is stopped.

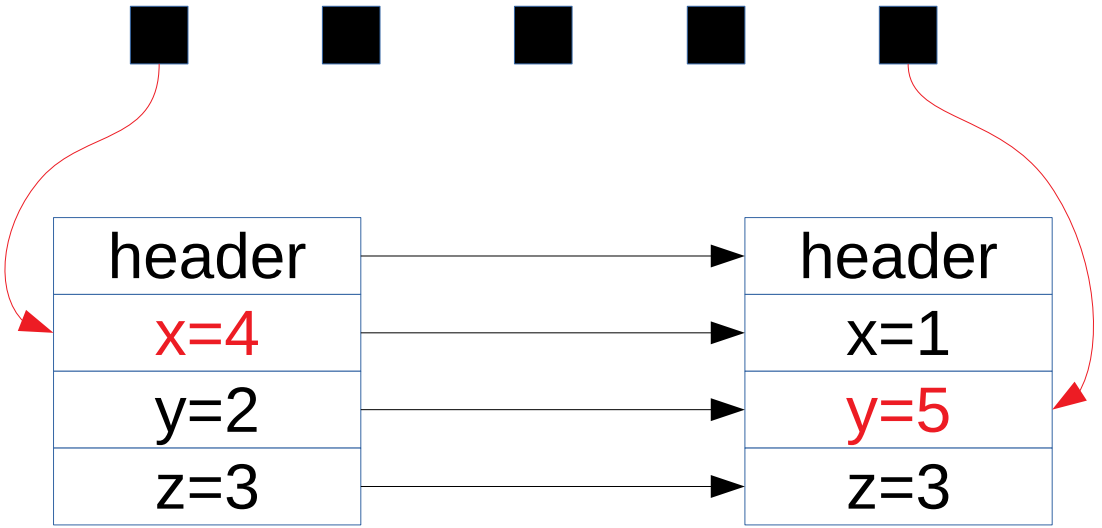
Problem: how to do this while program is running?

Concurrent Compaction



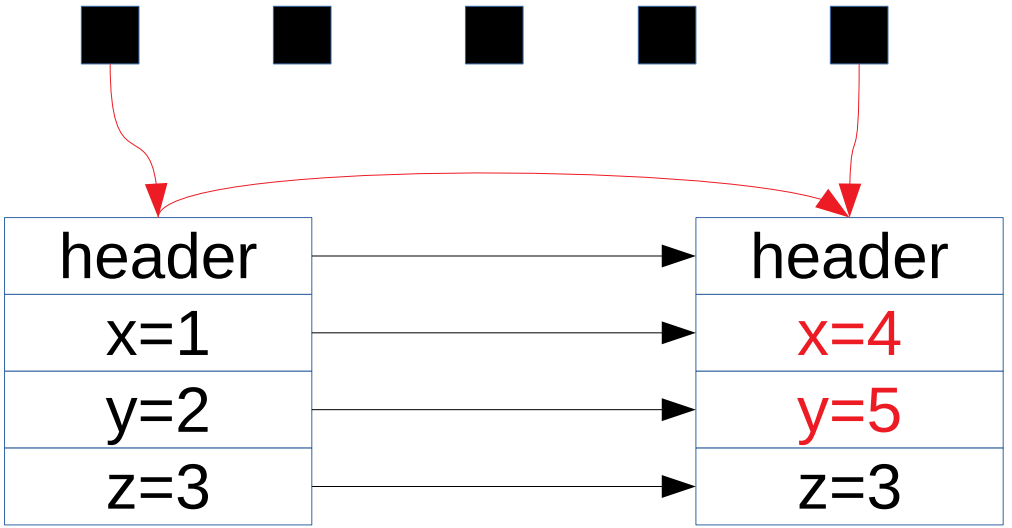
Concurrent Compaction

Problem:



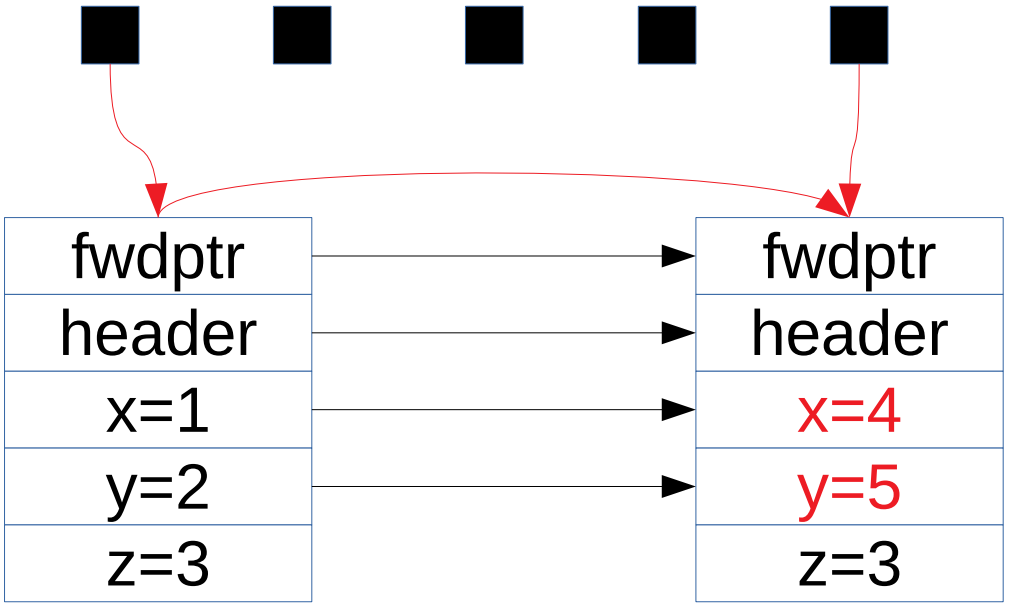
Concurrent Compaction

Solution:



Concurrent Compaction

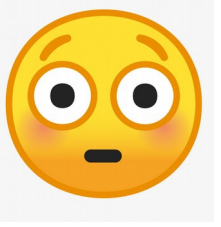
Solution (Shenandoah 1.0):



Shenandoah Barriers

Requires:

- Read barriers: Intercept reads, resolve by reading forwarding pointer.
- Write barriers: Intercept writes, establish canonical copy by CASing forwarding pointer
- Object-Equals barriers: to ensure correct result when comparing two copies of same object
- CAS barriers: combined problem of above
- Arraycopy barriers: Special case for bulk array copy



Shenandoah Barriers

Example:

```
void doSomething() {
  Foo foo = bar.foo;
  while (…) { // hot loop
    a = foo.x;
    foo.y = b;
  }
}
```

Shenandoah Barriers

Example:

```
void doSomething() {
  Foo foo = bar.foo;
  while (…) { // hot loop
    foo’= rb(foo); // Required for primitive fields too
    a = foo’.x;
    foo’’ = wb(foo); // Required for primitive fields too
    foo’’.y = b;
  }
}
```

Shenandoah Barriers

We can optimize this with some JIT heroics:

```
void doSomething() {
  Foo foo = bar.foo;
  foo’= rb(foo);
  foo’’ = wb(foo);
  while (…) { // hot loop
    a = foo’.x;
    foo’’.y = b;
  }
}
```

Shenandoah Barriers

We can optimize this with some JIT heroics:

```
void doSomething() {
  Foo foo = bar.foo;
  foo’= wb(foo);
  while (…) { // hot loop
    a = foo’.x;
    foo’.y = b;
  }
}
```

Shenandoah Barriers

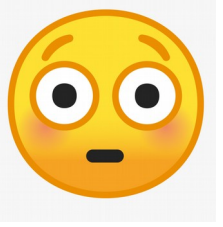
Idea: Establish canonical copy right when object is loaded

```
void doSomething() {
  Foo foo’ = bar.foo;
  foo= lrb(foo’);
  while (…) { // hot loop
    a = foo.x;
    foo.y = b;
  }
}
```

Shenandoah 2.0 Barriers: LRB

Load-reference-barriers:

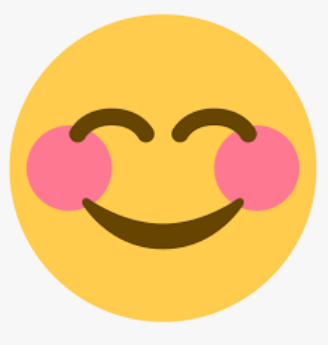
- ~~Read-barriers: Intercept reads, resolve by reading forwarding pointer.~~
- Write-barriers: Intercept ~~writes~~reference-loads, establish canonical copy by CASing forwarding pointer
- ~~Above two also required for primitive fields~~
- ~~Object-equals-barriers: to ensure correct result when comparing two copies of same object~~
- CAS-object-barriers: combined problem of above



Shenandoah 2.0 Barriers: LRB

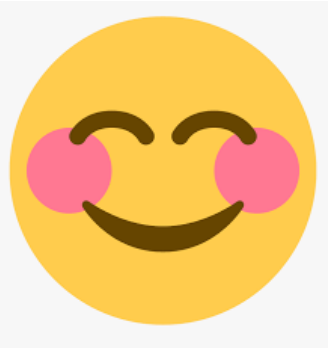
Load-reference-barriers:

- LRB: Intercept loads of objects, establish canonical copy by CASing forwarding pointer
- CAS-object-barriers: still necessary but simpler



Shenandoah 2.0 Barriers: LRB

Load-reference-barriers:



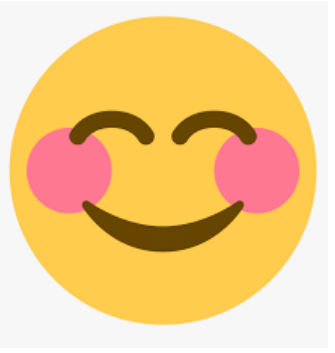
- Much simpler barrier scheme

- Strong invariant
- Much less frequent (only object-loads!)

- Much simpler optimization story

Shenandoah 2.0 Barriers: LRB

Load-reference-barriers:



- Much simpler barrier scheme
  → Makes porting much easier
- Strong invariant
- Much less frequent (only object-loads)
  → Enables elimination of forwarding pointer
- Much simpler optimization story

Shenandoah Architectures

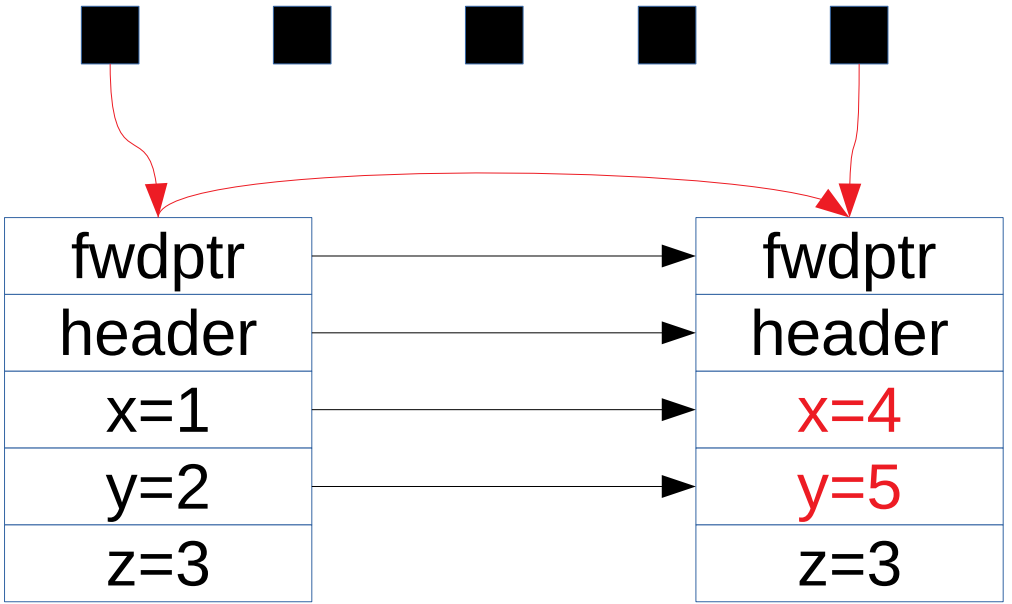
x86_64	✅	Primary target; continuously tested.
x86_32	✅	Secondary target; continuously tested.
AArch64	✅	Primary target; continuously tested.
ARM32	⭐	In development; help welcome.
PPC64	❌	Not supported; contributions welcome.
S390X	❌	Not supported; contributions welcome.
SPARC	❌	Not supported. No hardware to test on.
Others	❌	Please contact Shenandoah devs for guidance if you are willing to port Shenandoah to another platform.

Shenandoah OSes

Linux	✅	Primary target; continuously tested.
Windows	✅	Secondary target; continuously tested.
macOS	✅	Additional target; tested by community.
Solaris	✅	Additional target; tested by community.
Others	❓	The porting should be trivial, please try and contact Shenandoah devs with your success and failure reports.

New Forwarding Pointer

Concurrent Compaction (Shenandoah 1.0):



New Forwarding Pointer

Shenandoah 1.0 – weak invariant:

- All reads **may** read from old copy (if no canoncical copy has been established yet)
- All writes **must** write to new copy

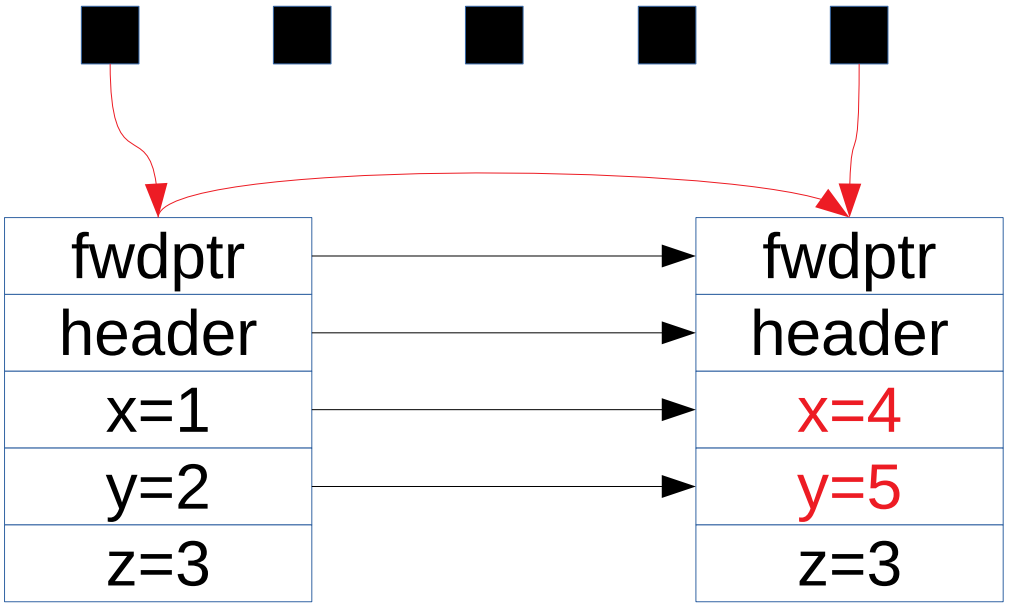
New Forwarding Pointer

Shenandoah 2.0 – strong invariant:

- All reads **must** read from new copy
- All writes **must** write to new copy
  - → old copy is 100% unused
  - → can use it to keep forwarding information

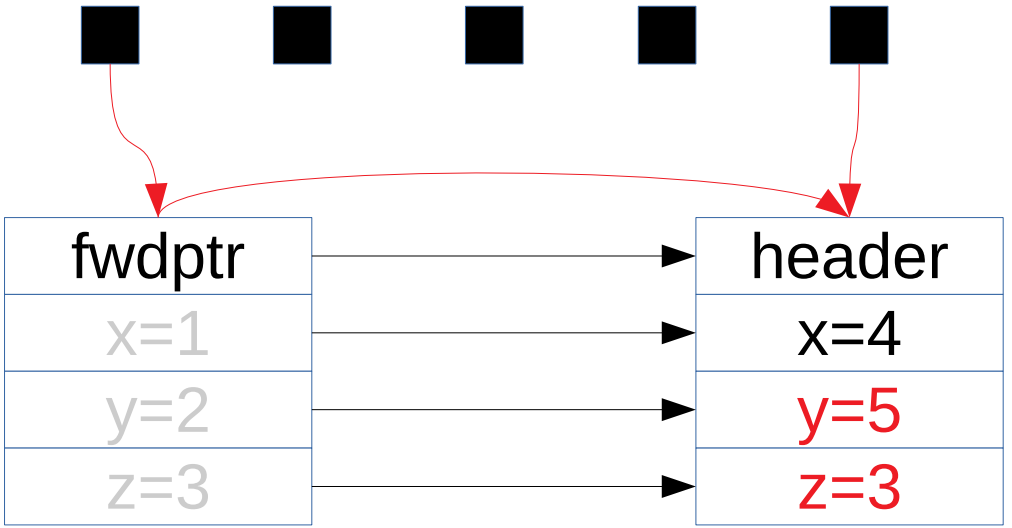
New Forwarding Pointer

Concurrent Compaction (Shenandoah 1.0):



New Forwarding Pointer

Concurrent Compaction (Shenandoah 2.0):



New Forwarding Pointer

```
T load_ref_barrier(T* addr) {
  T obj = *addr;
  if (in_cset(obj)) {
    T fwd = decode_fwdptr(obj);
    if (oop == fwd) {
      fwd = lrb_slowpath(oop);
    }
    return fwd;
  }
  return obj;
}
```

New Forwarding Pointer

```
// Shenandoah 1.0 impl
T decode_fwdptr(T obj) {
  return ((T*)obj)[fwdptr_offset];
}

// Shenandoah 2.0 impl
T decode_fwdptr(T obj) {
  intptr_t hdr = ((intptr_t*)obj)[header_offset];
  if ((hdr & 3) == 3) { // Forwarded
    return (T)(hdr & ~3);
  } else {
    Return obj;
  }
}
```

New Forwarding Pointer

- Reduces memory footprint per-object
- Down to 0.66x (best case), realistically 0.75x-0.95x
  → reduces allocation pressure
  → fewer GC cycles for same amount of allocs

Self-fixing Barriers

```
T load_ref_barrier(T* addr) {
  T obj = *addr;
  if (in_cset(obj)) {
    T fwd = decode_fwdptr(obj);
    if (oop == fwd) {
      fwd = lrb_slowpath(oop);
    }
    return fwd;
  }
  return obj;
}
```

Self-fixing Barriers

```
T load_ref_barrier(T* addr) {
  T obj = *addr;
  if (in_cset(obj)) {
    T fwd = decode_fwdptr(obj);
    if (oop == fwd) {
      fwd = lrb_slowpath(oop);
    }
    return fwd;
  }
  return obj;
}
```

Infrequent case: Object not forwarded yet

Self-fixing Barriers

```
T load_ref_barrier(T* addr) {
  T obj = *addr;
  if (in_cset(obj)) {
    T fwd = decode_fwdptr(obj);
    if (oop == fwd) {
      fwd = lrb_slowpath(oop);
    }
    return fwd;
  }
  return obj;
}
```

Frequent case: Object forwarded, but field not updated, yet

Self-fixing Barriers

- While we’re there, couldn’t we update the field?

Self-fixing Barriers

```
T load_ref_barrier(T* addr) {
  T obj = *addr;
  if (in_cset(obj)) {
    T fwd = decode_fwdptr(obj);
    if (oop == fwd) {
      fwd = lrb_slowpath(oop);
    } else {
      cas(addr, fwd, obj);
    }
    return fwd;
  }
  return obj;
}
```

Frequent case: Object forwarded, field updated already

Availability

- Shenandoah 1.0 in JDK12
- Shenandoah 2.0 in JDK13
- 2.0 in shenandoah/jdk11 (Fedora, RHEL, ...)
- 2.0 in shenandoah/jdk8 (Fedora, RHEL, ...)
- Upstream 2.0 to jdk11u proper in progress
- Upstream 2.0 to jdk8u proper soon

Contacts

- Wiki: https://wiki.openjdk.java.net/display/shenandoah/Main
- Mailing list: https://mail.openjdk.java.net/mailman/listinfo/shenandoah-dev
- Twitter: @rkennke @shipilev

Questions?