Concurrent GCs
ZGC & Shenandoah

 @simonebordet

sbordet@webtide.com

Simone Bordet

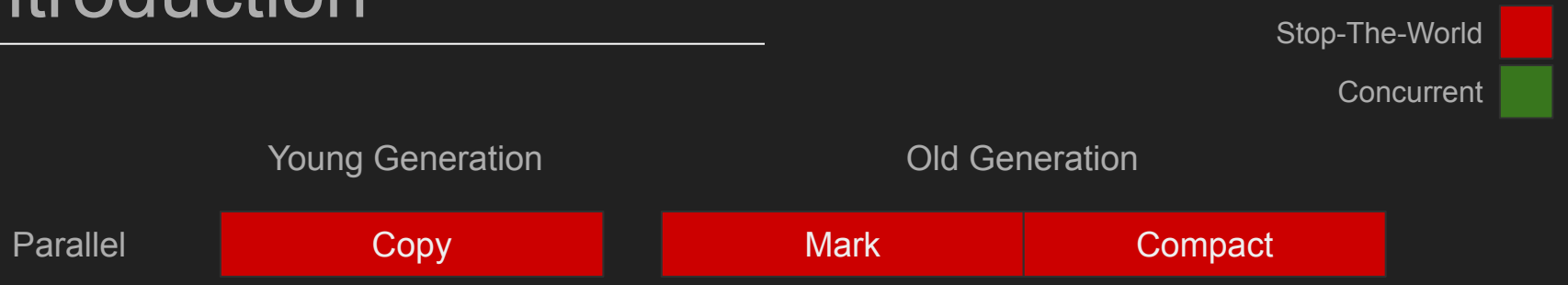
- @simonebordet
- sbordet@webtide.com

- Works @ Webtide
  - The company behind Jetty and CometD

- JVM Tuning Expert

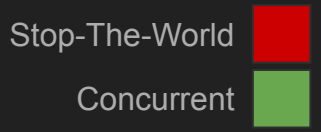
Introduction

Introduction



simone.bordet@gmail.com

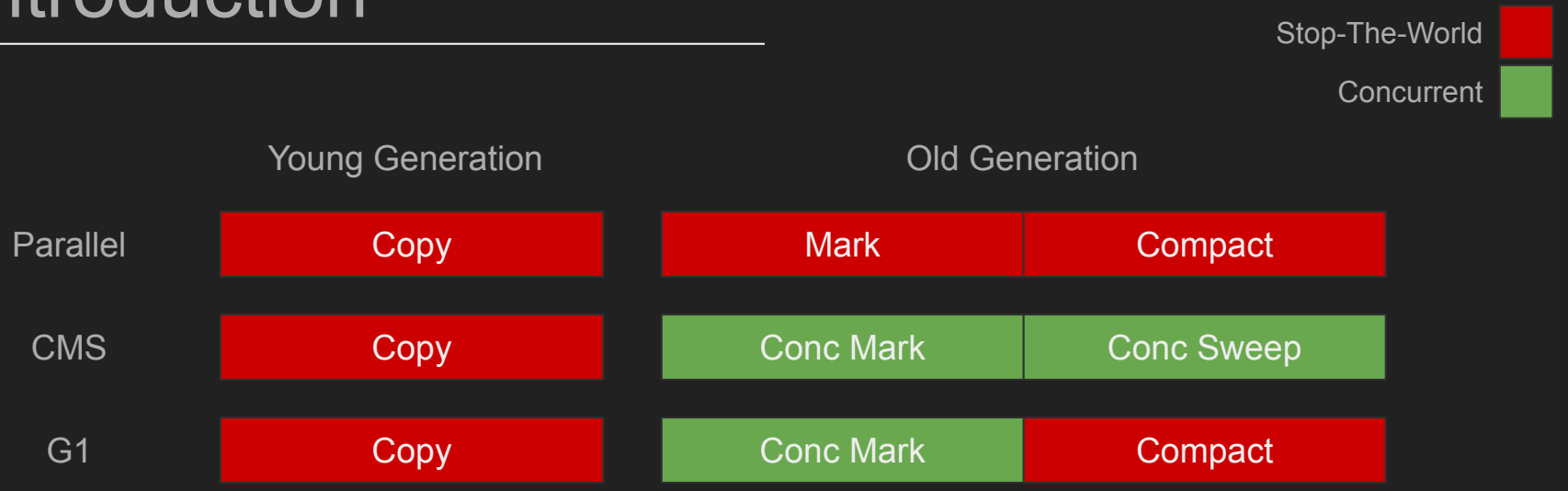
Introduction



	Young Generation	Old Generation	
Parallel	Copy	Mark	Compact
CMS	Copy	Conc Mark	Conc Sweep

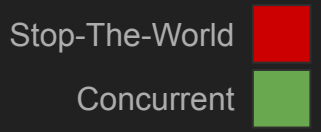
simone.bordet@gmail.com

Introduction



simone.bordet@gmail.com

Introduction



	Young Generation	Old Generation	
Parallel	Copy	Mark	Compact
CMS	Copy	Conc Mark	Conc Sweep
G1	Copy	Conc Mark	Compact
ZGC		Conc Mark	Conc Compact
Shenandoah		Conc Mark	Conc Compact

simone.bordet@gmail.com

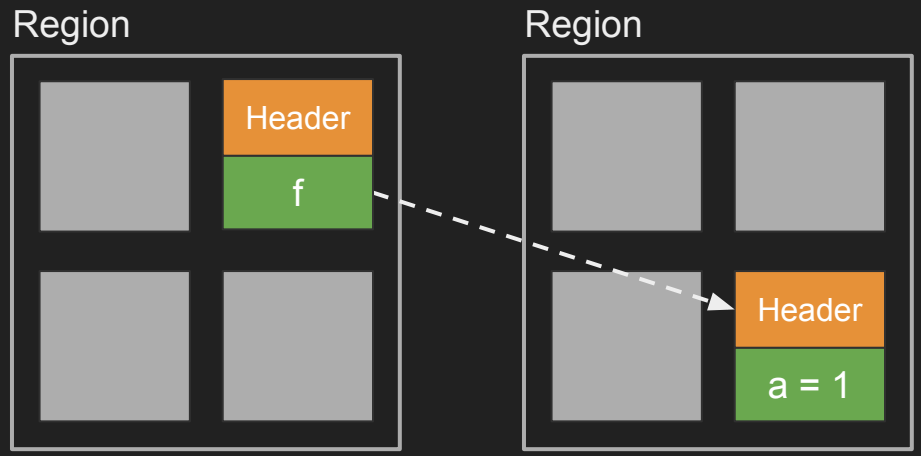
Introduction

- Concurrent GCs
  - GC runs concurrently with the application
- GC races with Application
  - Marking an object “alive”
  - Compacting / Moving objects

simone.bordet@gmail.com

Introduction

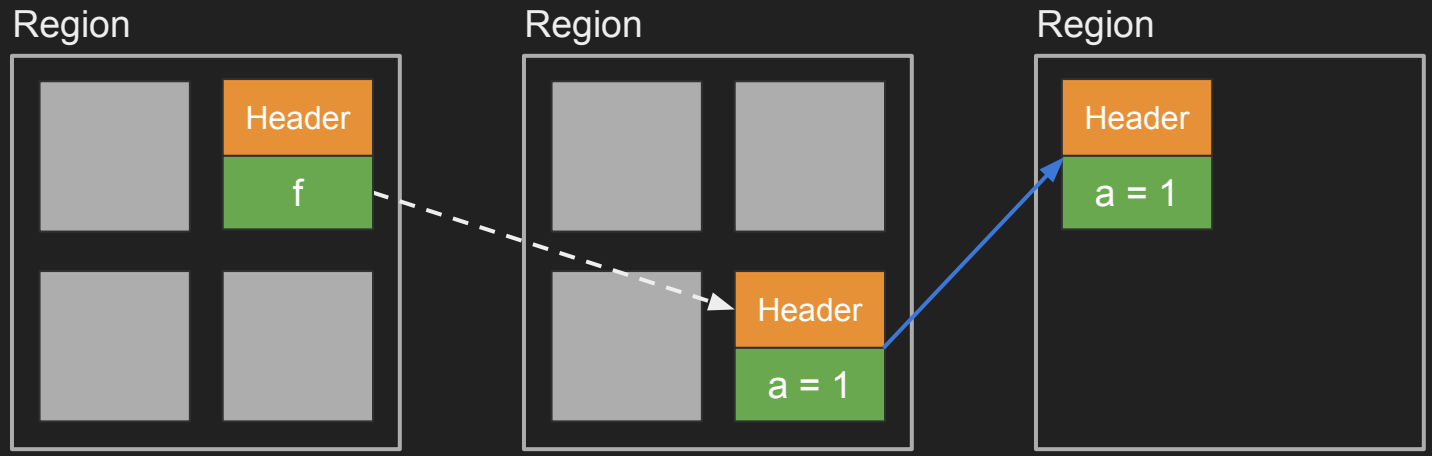
- Concurrent copying example



simone.bordet@gmail.com

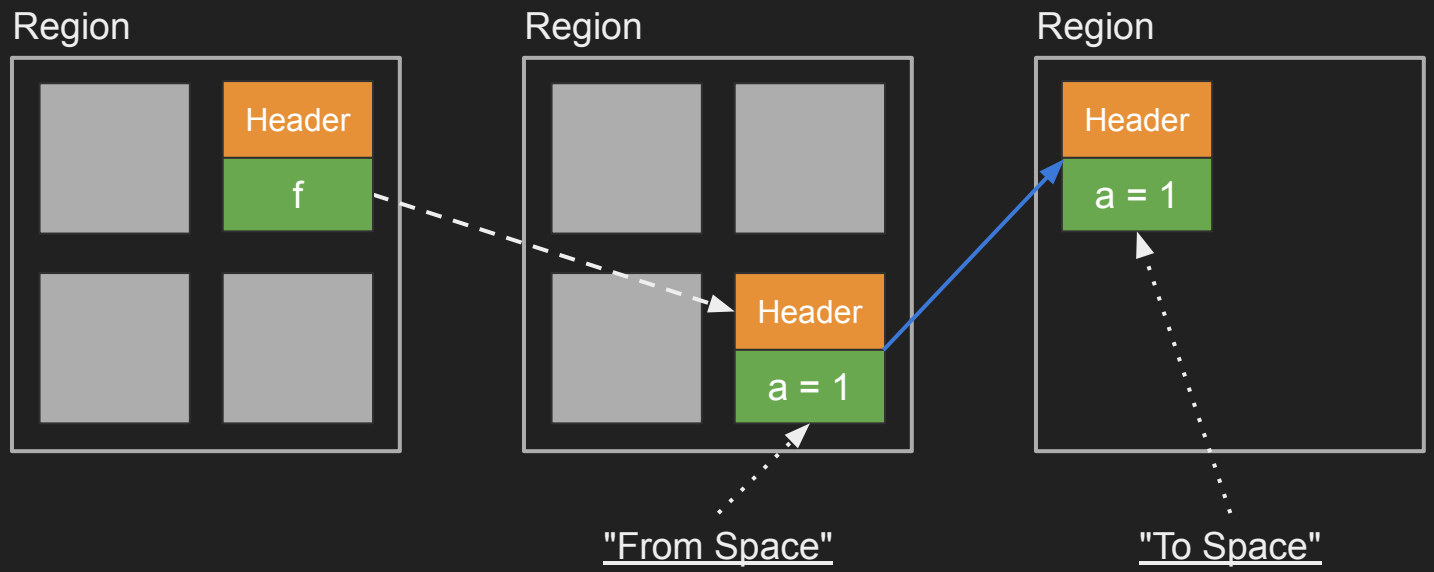
Introduction

- Garbage Collector compacts object



simone.bordet@gmail.com

Introduction



simone.bordet@gmail.com

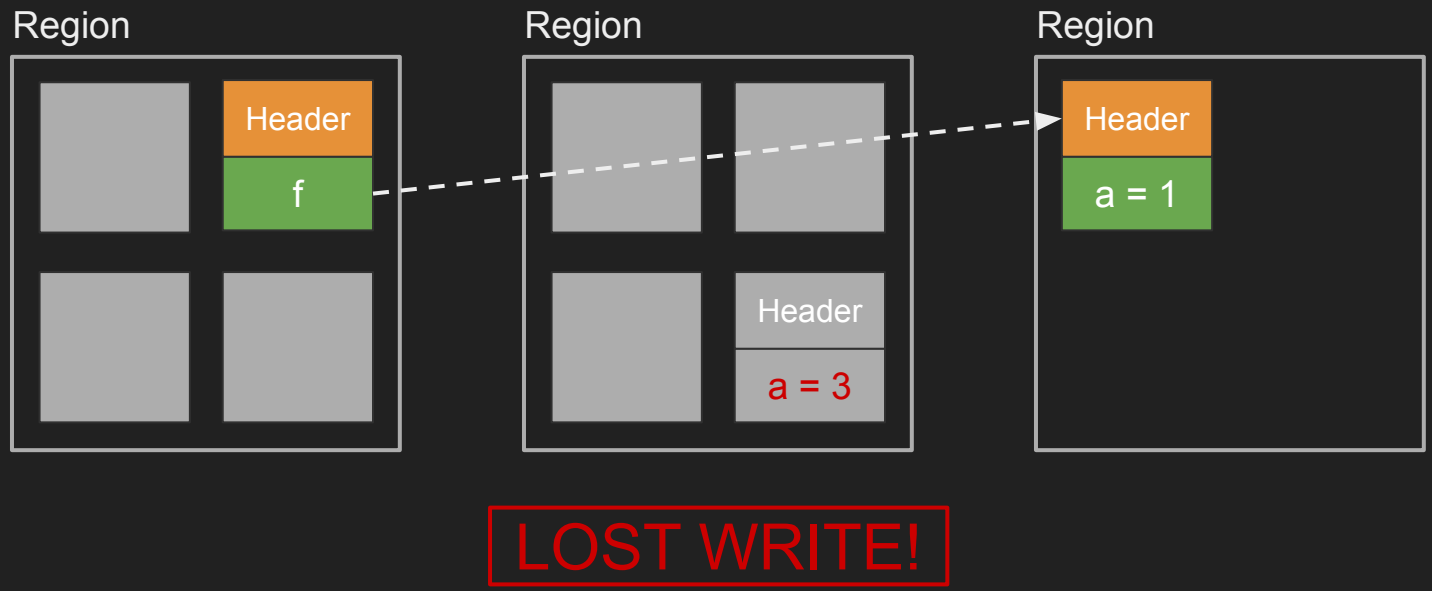
Introduction

- Application writes concurrently

simone.bordet@gmail.com

Introduction

- Garbage Collector updates references



Introduction

- Concurrent GCs store object metadata
  - E.g. whether an object has been marked
- Concurrent GCs require JIT support
- JIT injects code that helps the GC
  - GC barriers

ZGC

ZGC

- Present in OpenJDK 11+
  - Available in AdoptOpenJDK builds
  - Available in Oracle OpenJDK builds
    - From https://jdk.java.net
- Scalable Low Latency GC
- Concurrent Compaction, Single Generation

ZGC

- Only for Linux x86 64-bit
  - No compressed pointers
  - ARM port underway

- Region Based
  - ZPages - similar to G1 regions
    - Small (2 MiB - object size up to 256 KiB)
    - Medium (32 MiB - object size up to 4 MiB)
    - Large (4+ MiB - object size > 4 MiB)

ZGC

GC Thread
App Thread

STW
Mark Start

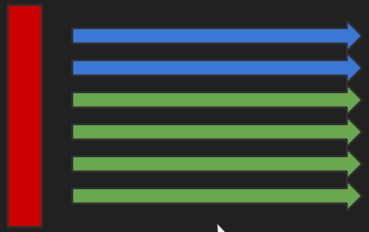
Scan threads stacks
Mark object roots

simone.bordet@gmail.com

ZGC

GC Thread
App Thread

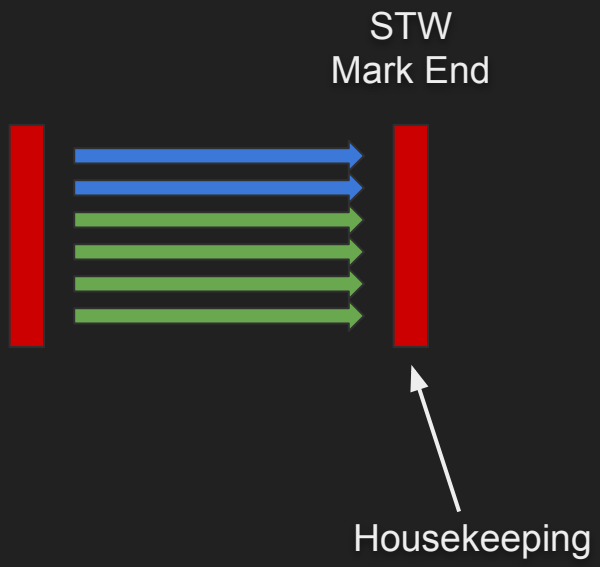
Concurrent
Marking



Mark object graph
Gather liveness information

simone.bordet@gmail.com

ZGC

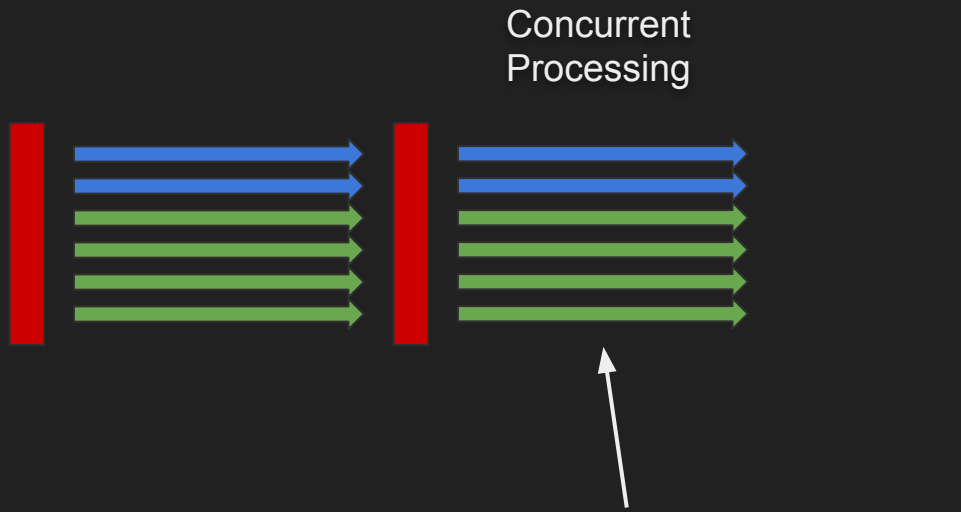


GC Thread
App Thread

simone.bordet@gmail.com

ZGC

GC Thread
App Thread

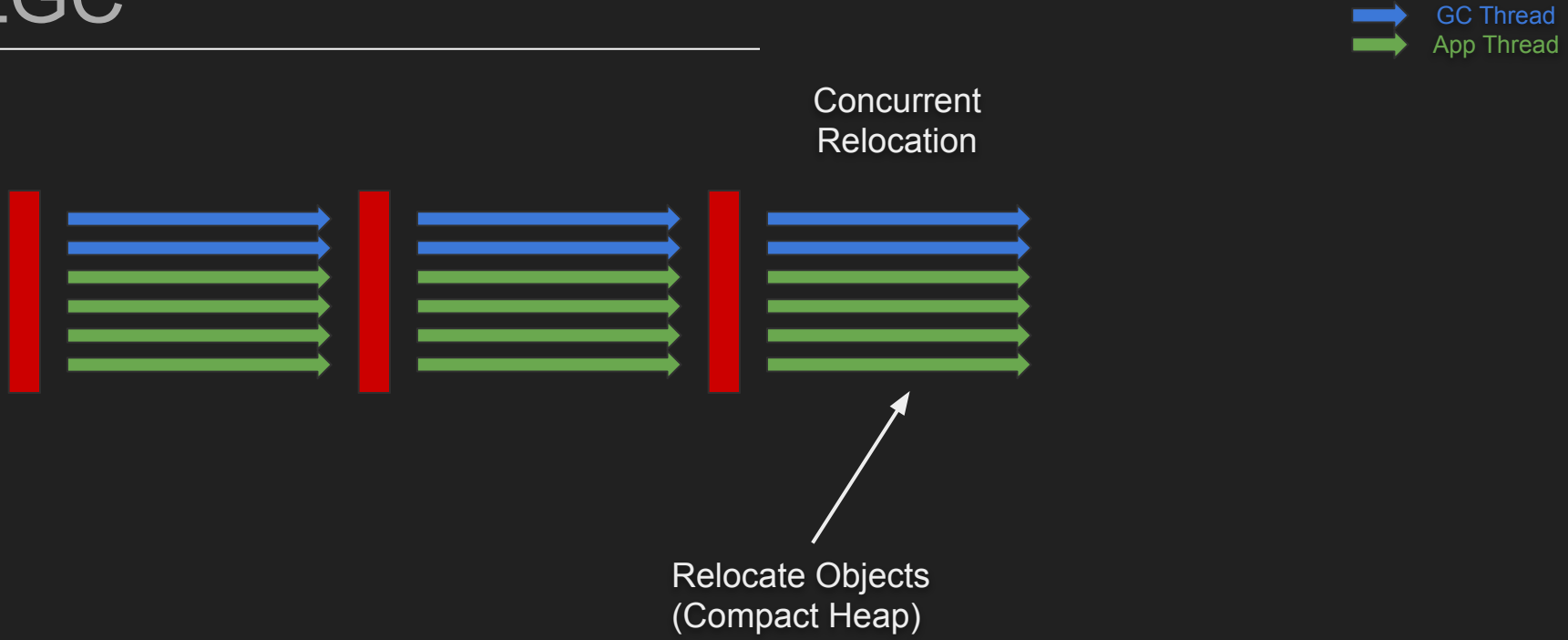


simone.bordet@gmail.com

ZGC

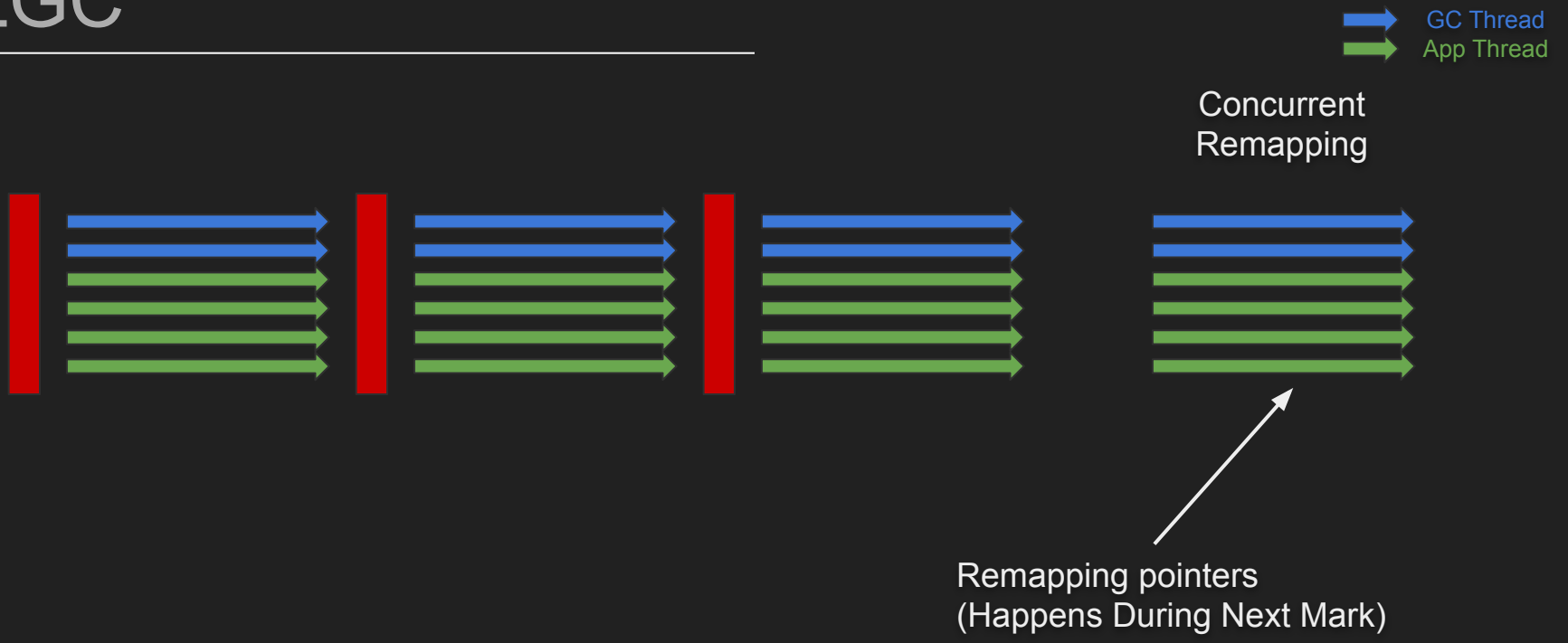
simone.bordet@gmail.com

ZGC



simone.bordet@gmail.com

ZGC



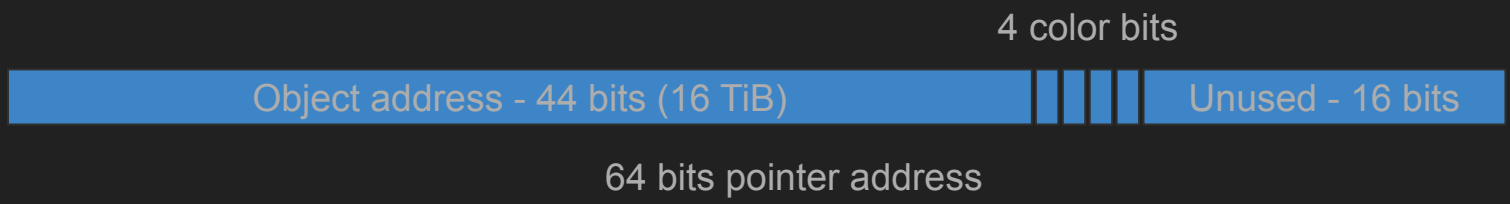
simone.bordet@gmail.com

ZGC

- ZGC solution
  - To avoid races with the application
- Colored Pointers
- Load GC barrier

ZGC

- ZGC colored pointers
  - Color indicates GC metadata



- Check pointer color against GC phase
  - Wrong color => take action

ZGC

- ZGC load GC barrier

```
Object f = obj.field;
<load_barrier>
```

simone.bordet@gmail.com

ZGC

- ZGC load GC barrier

```
Object f = obj.field;
if (addr_of(f) & wrong_gc_color) {
  slow_path()
}
```

simone.bordet@gmail.com

ZGC

- ZGC load GC barrier

```
Object f = obj.field;
<load_barrier>

mov 0x10(%rdi), %rsi
test %rsi, 0x20(%r15)
jne slow_path
```

simone.bordet@gmail.com

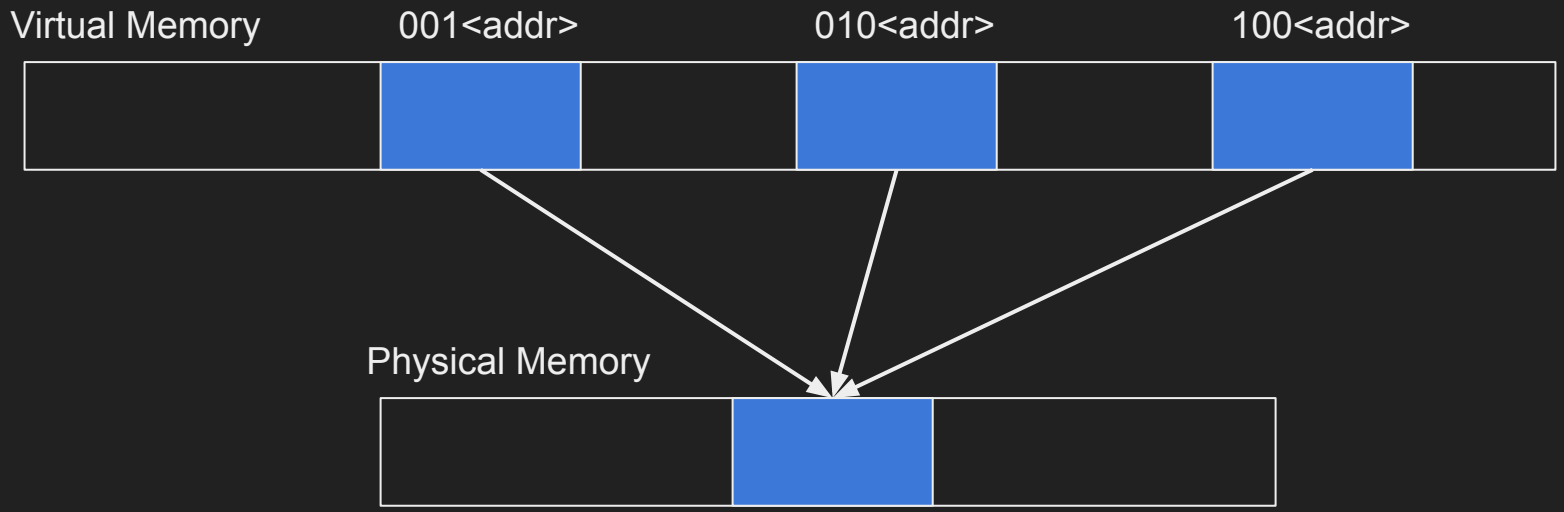
ZGC

- ZGC load GC barrier tests for the right color
- Wrong color -> take `slow_path`
  - Fix color & Run some action - atomically
- The action depends on the GC phase

simone.bordet@gmail.com

ZGC

- ZGC Multi-Mapping

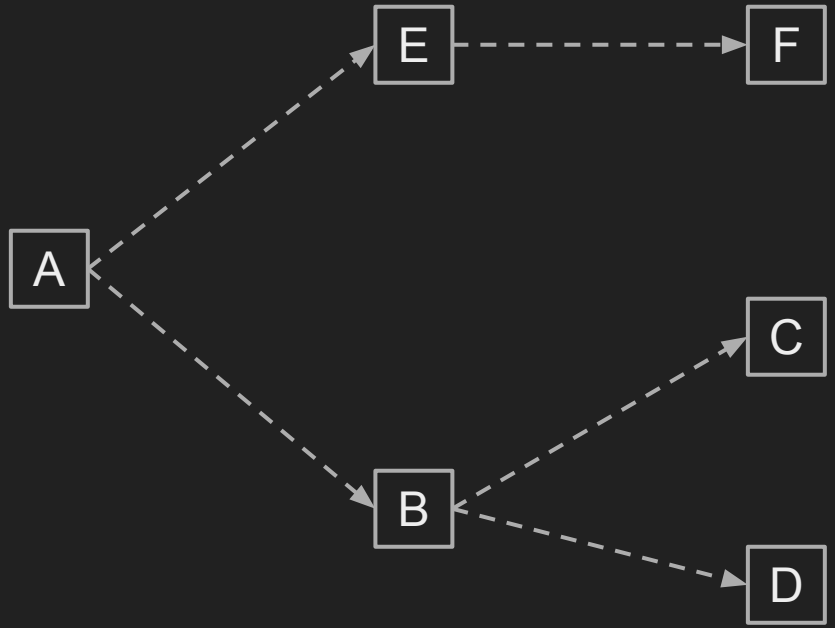


ZGC

- ZGC Multi-Mapping
- RSS shows 3x size
  - E.g. -Xmx=16G -> RSS~=50G
  - On Linux, use `smem` and track PSS
- Watch out for OOM Killers in clouds
  - E.g. AWS

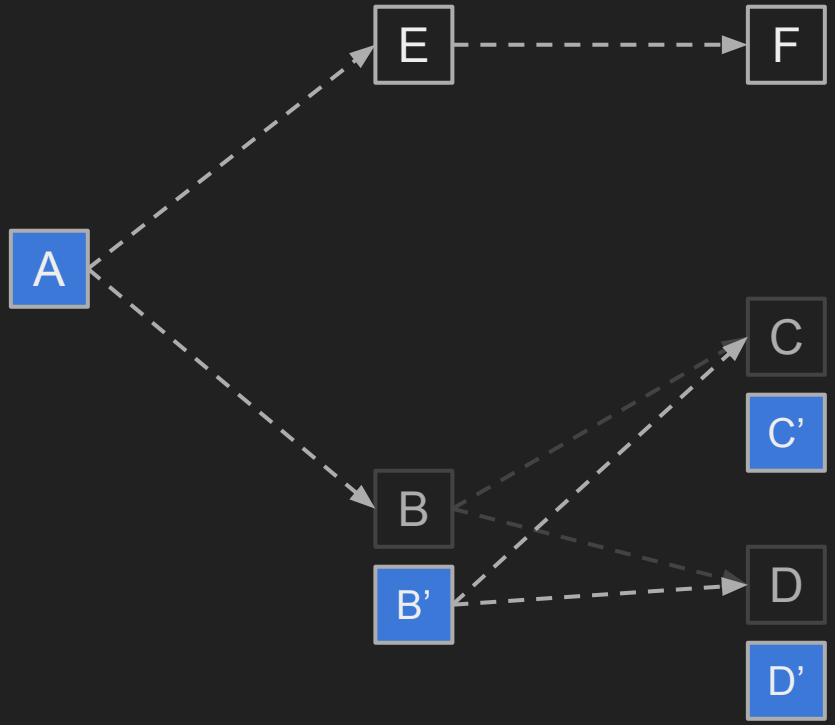
ZGC

- Relocation



ZGC

- Relocation



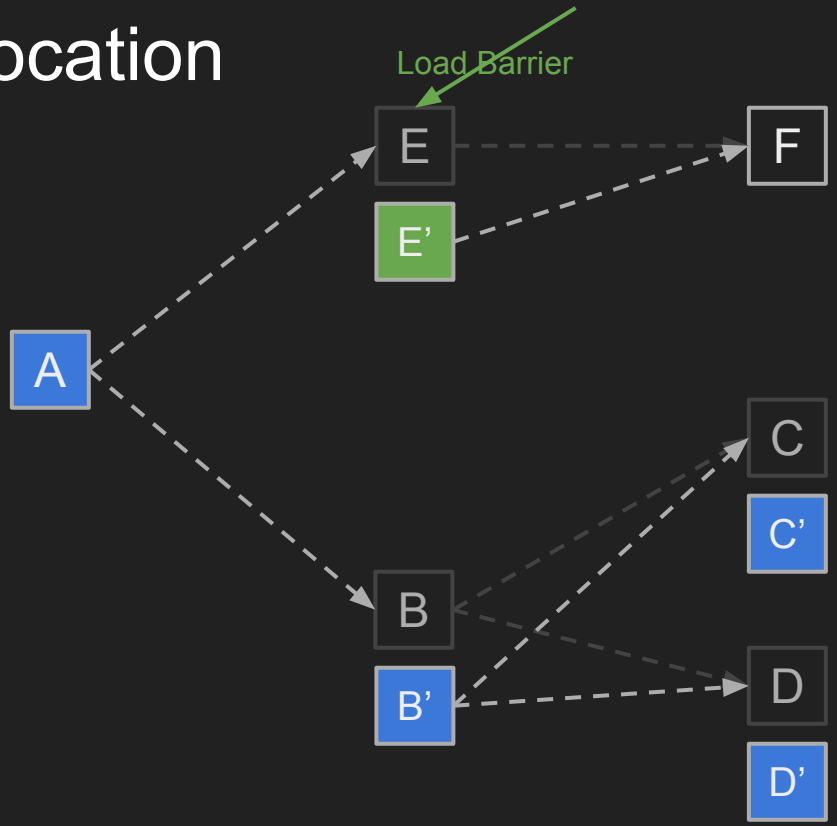
MetaData

B -> B'
C -> C'
D -> D'

simone.bordet@gmail.com

ZGC

- Relocation

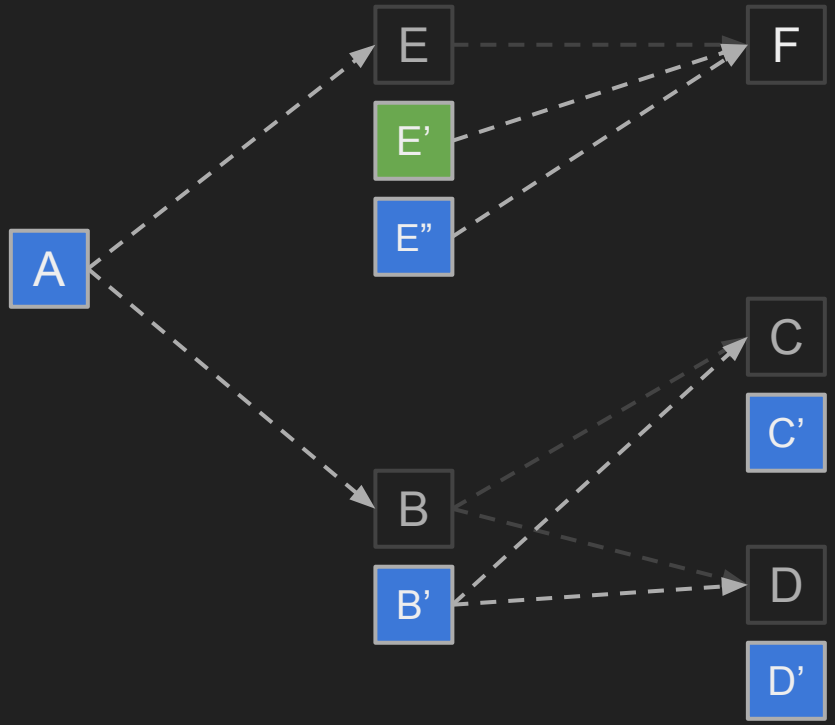


MetaData

B -> B'
C -> C'
D -> D'
E -> E'

ZGC

- Relocation



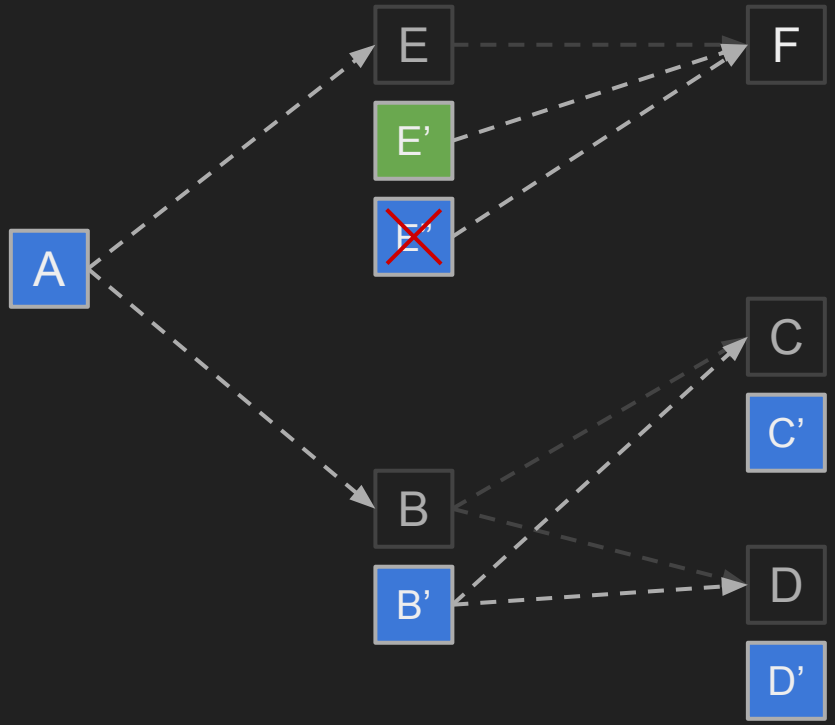
MetaData

B -> B'
C -> C'
D -> D'
E -> E'

simone.bordet@gmail.com

ZGC

- Relocation



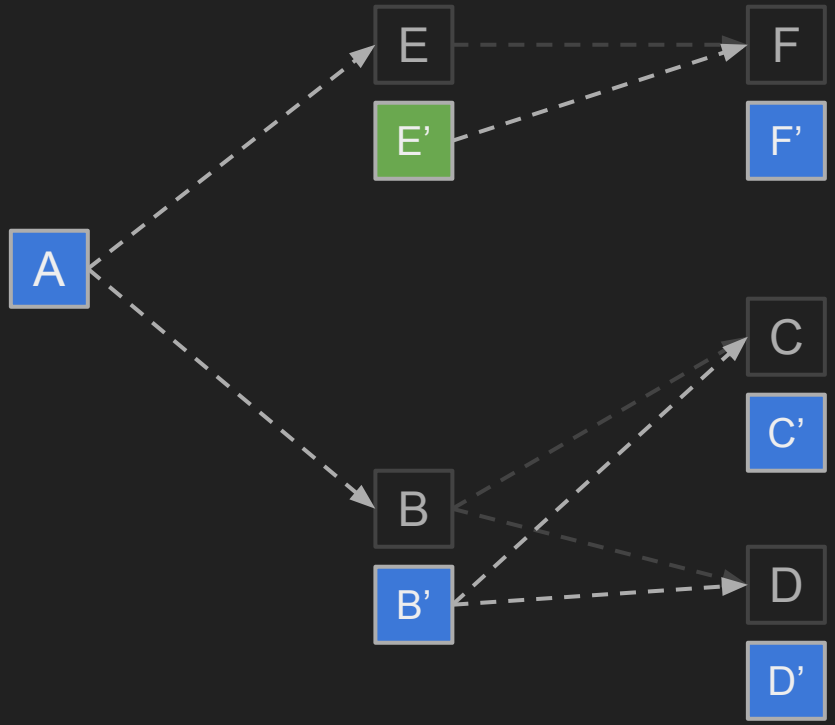
MetaData

B -> B'
C -> C'
D -> D'
E -> E'
~~E -> E''~~

simone.bordet@gmail.com

ZGC

- Relocation



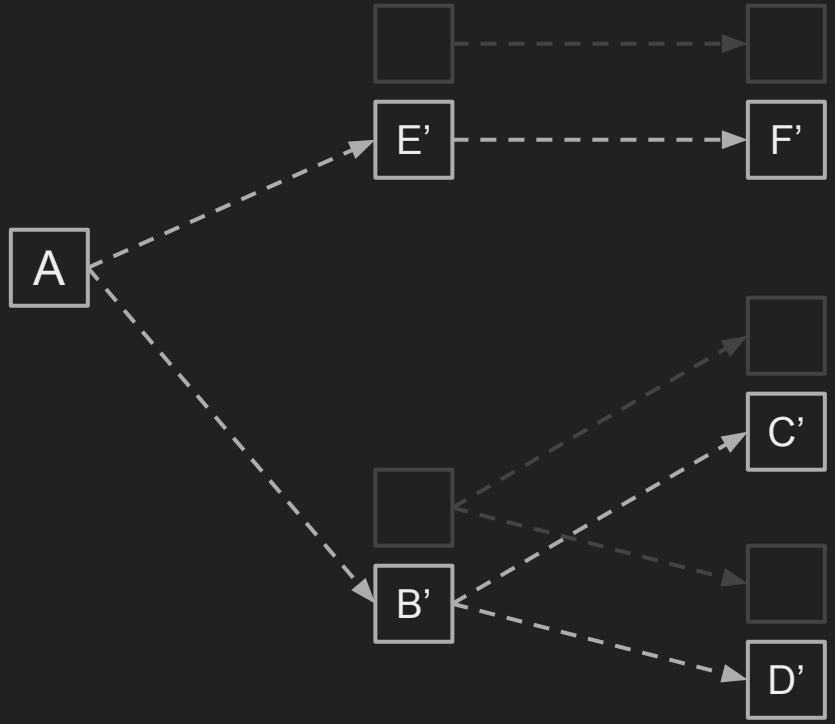
MetaData

B -> B'
C -> C'
D -> D'
E -> E'
F -> F'

simone.bordet@gmail.com

ZGC

- Remapping



simone.bordet@gmail.com

ZGC

- References
  - Per Lidén, Erik Österlund et al.
    - https://www.youtube.com/watch?v=7cWiwu7kYkE
    - https://www.youtube.com/watch?v=kF_r3GE3zOo
  - Project Wiki
    - https://wiki.openjdk.java.net/display/zgc/Main
  - Mailing List
    - zgc-dev@openjdk.java.net

ShenandoahGC

ShenandoahGC

- Present in OpenJDK 12+ repository
  - Available in AdoptOpenJDK builds
  - NOT available in Oracle's OpenJDK builds
  - Present in RedHat OpenJDK 8, 11+
- Scalable low latency GC
- Concurrent Compaction, Single Generation

simone.bordet@gmail.com

ShenandoahGC

- Available for x86 32-bit and 64-bit
  - ARM ports available
- Linux, MacOS, Windows
  - JDK 8, 11 & Latest
- Region Based
  - Derived from G1

simone.bordet@gmail.com

ShenandoahGC

GC Thread
App Thread

STW
Mark Start

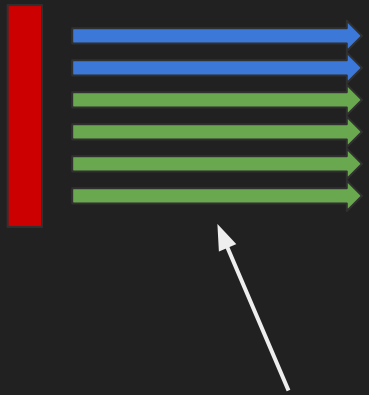
Scan threads stacks
Mark object roots

simone.bordet@gmail.com

ShenandoahGC

GC Thread
App Thread

Concurrent
Marking



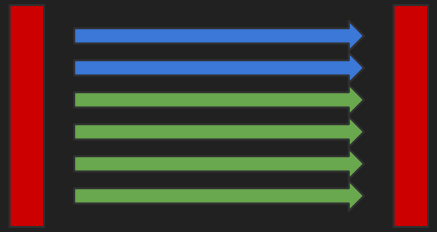
Mark object graph
Gather liveness information

simone.bordet@gmail.com

ShenandoahGC

GC Thread
App Thread

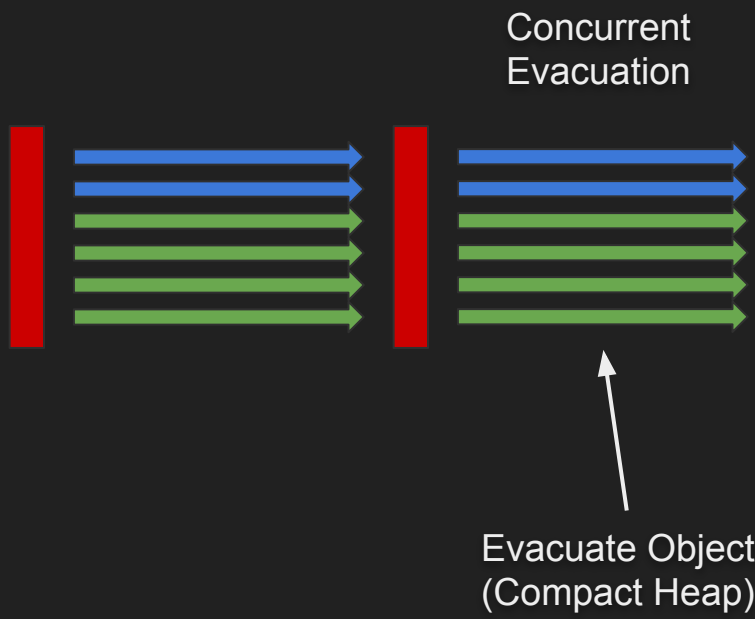
STW
Mark End



Weak Reference Processing
Unload Classes
Evacuate Roots
Housekeeping

simone.bordet@gmail.com

ShenandoahGC

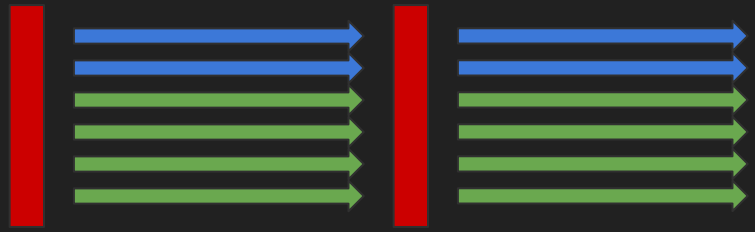


GC Thread
App Thread

simone.bordet@gmail.com

ShenandoahGC

GC Thread
App Thread



Concurrent
Update References

Remapping pointers
(Happens During Next Mark)

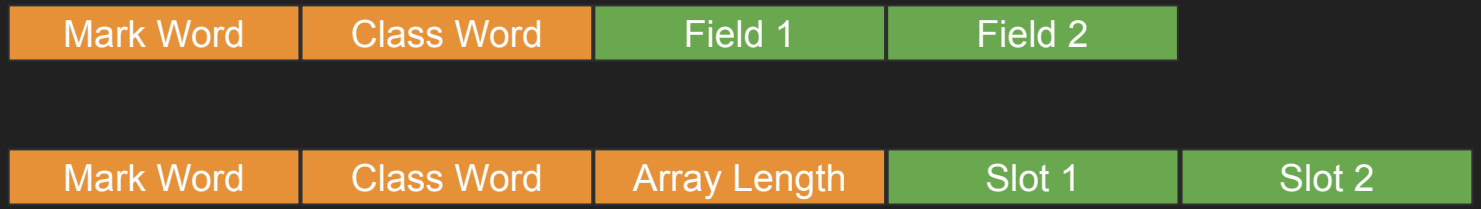
simone.bordet@gmail.com

ShenandoahGC

- ShenandoahGC solution
  - To avoid races with the application
- Brooks (Forward) Pointers
- Load & Store GC Barriers

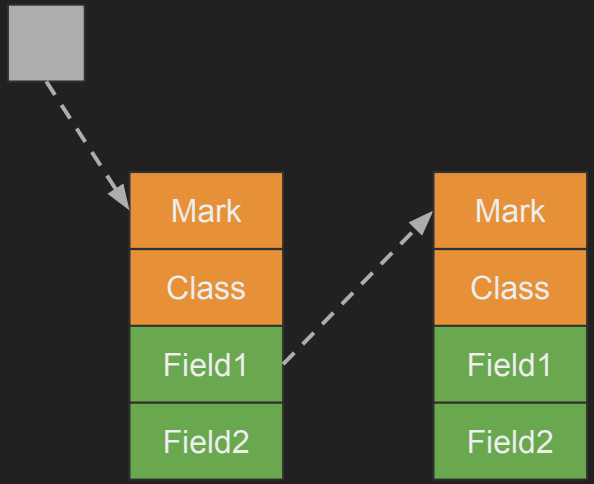
ShenandoahGC

- HotSpot Object Header
  - "Word" == 32 bits on i386; 64 bits on x86-64



ShenandoahGC 1.0

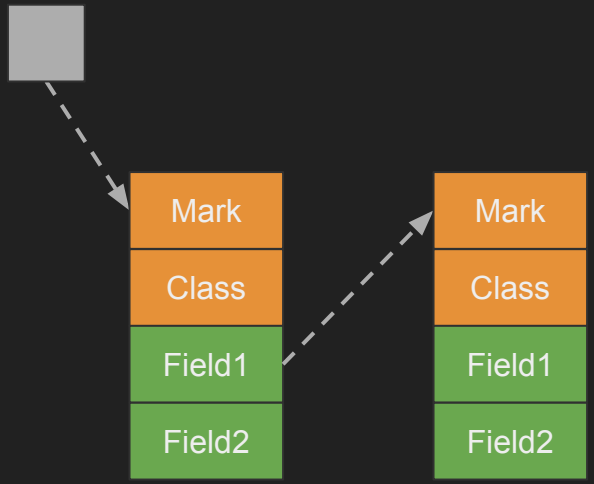
- ShenandoahGC 1.0

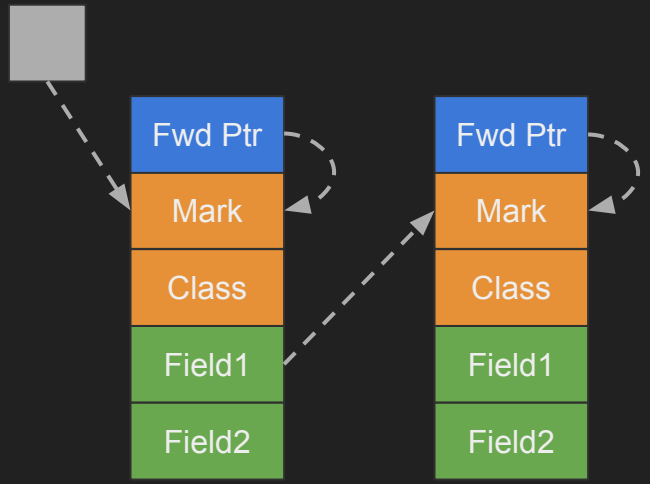


simone.bordet@gmail.com

ShenandoahGC 1.0

- ShenandoahGC 1.0





ShenandoahGC 1.0

- ShenandoahGC 1.0

- Maintains weak to-space invariant
  - Reads from both from-space to-space copies
  - Writes only to to-space copy

simone.bordet@gmail.com

ShenandoahGC 1.0

- Load GC barrier: dereference forward pointer

```
Object f = obj.field;
<load_barrier>
```

simone.bordet@gmail.com

ShenandoahGC

- Load GC barrier: dereference forward pointer

```
Object f = obj.field;
f = deref(addr_of(f) - 8);
```

simone.bordet@gmail.com

ShenandoahGC 1.0

- Load GC barrier: dereference forward pointer

```
Object f = obj.field;
<load_barrier>

mov 0x10(%rdi), %rsi
mov -0x08(%rsi), %rsi
```

simone.bordet@gmail.com

ShenandoahGC

- Store GC barrier
  - Runs some action depending on GC phase

```
obj.f = o;
<store_barrier>
```

simone.bordet@gmail.com

ShenandoahGC

- Store GC barrier
  - Runs some action depending on GC phase

```
obj.f = o;
if (in_evac_phase &&
    in_collection_set(obj) &&
    !is_forwarded(obj))
  slow_path();
```

simone.bordet@gmail.com

ShenandoahGC

- Store GC barrier
  - Runs some action depending on GC phase

```
obj.f = o;
mov 0x3d8(%r15), %r11
test %r11, %r11
...
jne <slow_path>
```

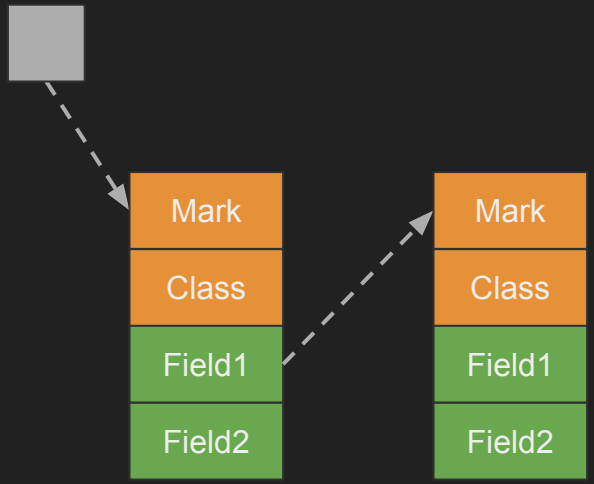
simone.bordet@gmail.com

ShenandoahGC

- ShenandoahGC 1.0 Problems
- More memory needed (due to the forward pointer)
  - Worst case 50%, common case 5-10%
- Complicated (exotic) GC barriers
  - Load & Store GC barriers to maintain invariants

ShenandoahGC

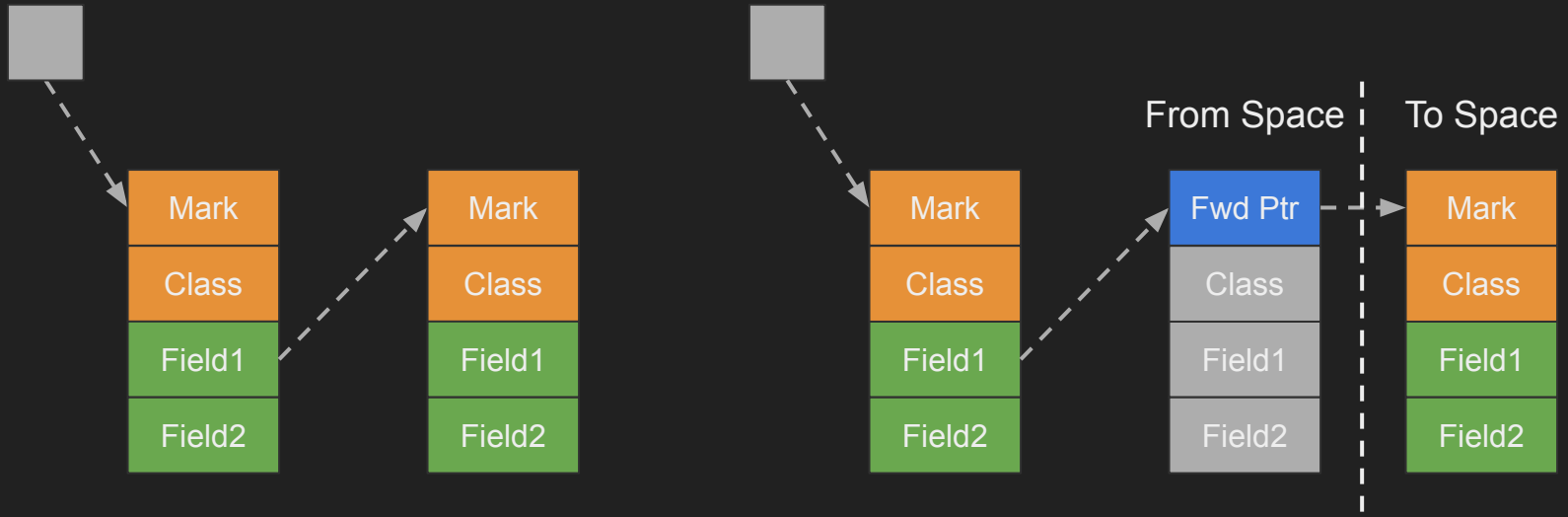
- ShenandoahGC 2.0



simone.bordet@gmail.com

ShenandoahGC

- ShenandoahGC 2.0



simone.bordet@gmail.com

ShenandoahGC

- ShenandoahGC 2.0

- Maintains a strong to-space invariant
  - Reads only possible from to-space copy
  - Writes only possible from to-space copy

simone.bordet@gmail.com

ShenandoahGC 2.0

- Load Barrier (LRB - Load Reference Barrier)
  - Checks whether there may be forwarded objects
    - And whether in c-set and whether is forwarded

```
Object f = obj.f;
<load_barrier>
```

simone.bordet@gmail.com

ShenandoahGC 2.0

- Load Barrier (LRB - Load Reference Barrier)

```
Object f = obj.f;
if (in_evac_phase &&
    in_collection_set(obj) &&
    !is_forwarded(obj))
  slow_path();
```

simone.bordet@gmail.com

ShenandoahGC 2.0

- Load Barrier (LRB - Load Reference Barrier)

```
Object f = obj.f;
test  $0x1,0x20(%r15)
...
jne <slow_path>
```

simone.bordet@gmail.com

ShenandoahGC 2.0

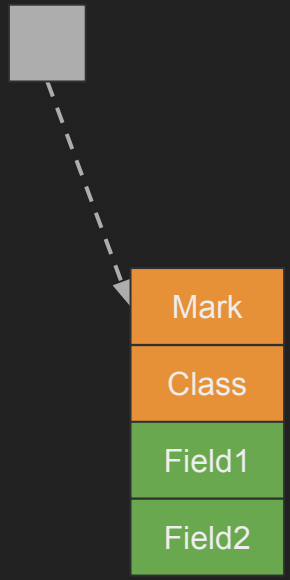
- Shenandoah 2.0

- Load and Store GC barrier are the same!
  - Big codebase simplification
  - Improved performance

simone.bordet@gmail.com

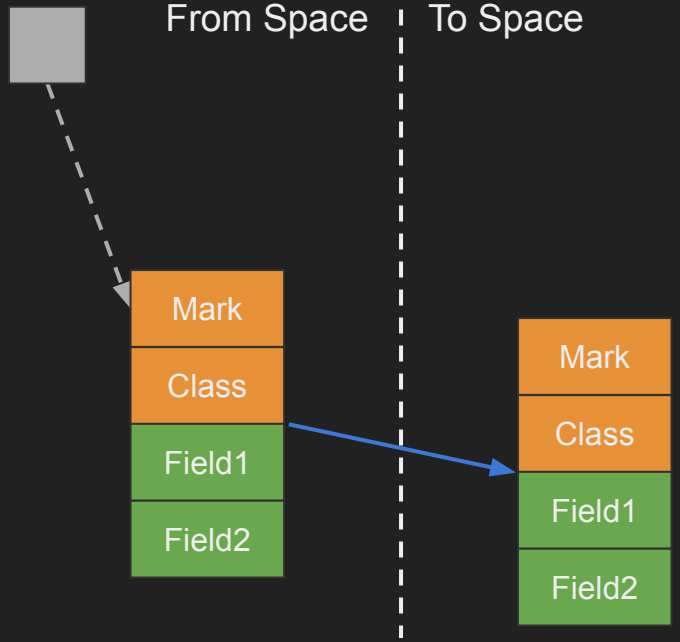
ShenandoahGC

- Evacuation



ShenandoahGC

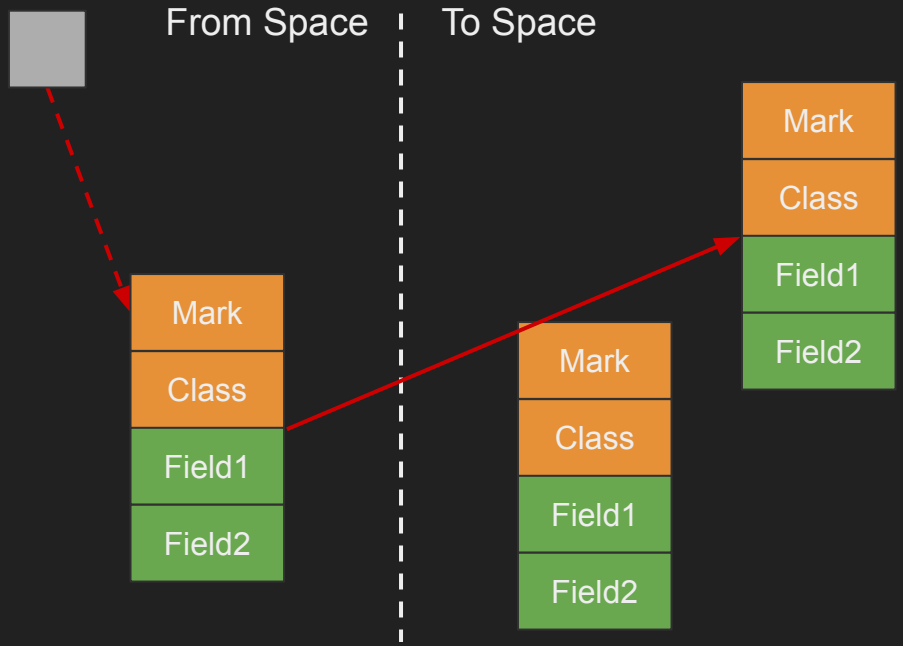
- Evacuation (GC Copy)



simone.bordet@gmail.com

ShenandoahGC

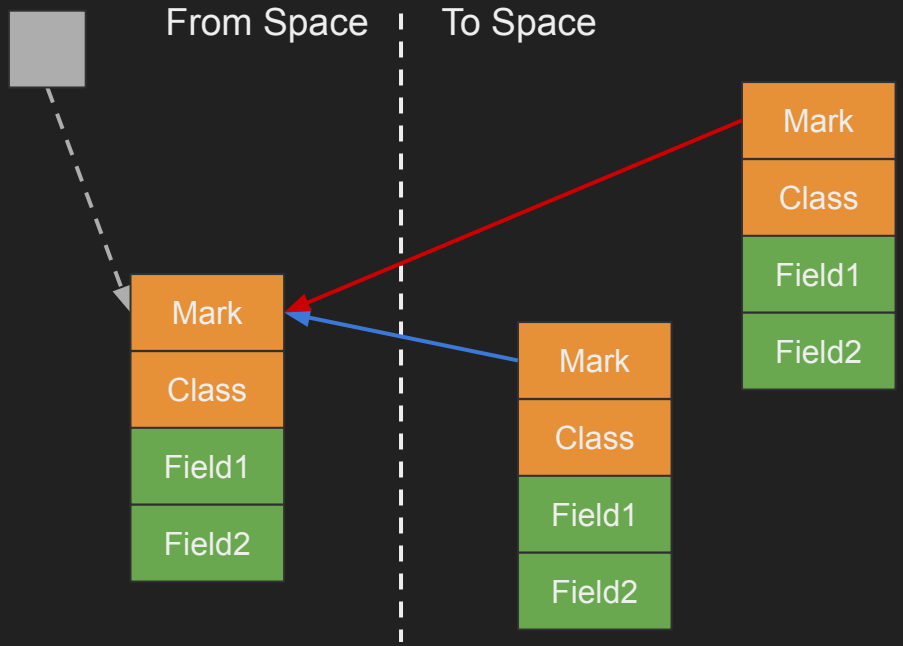
- Evacuation (Application triggers LRB)



simone.bordet@gmail.com

ShenandoahGC

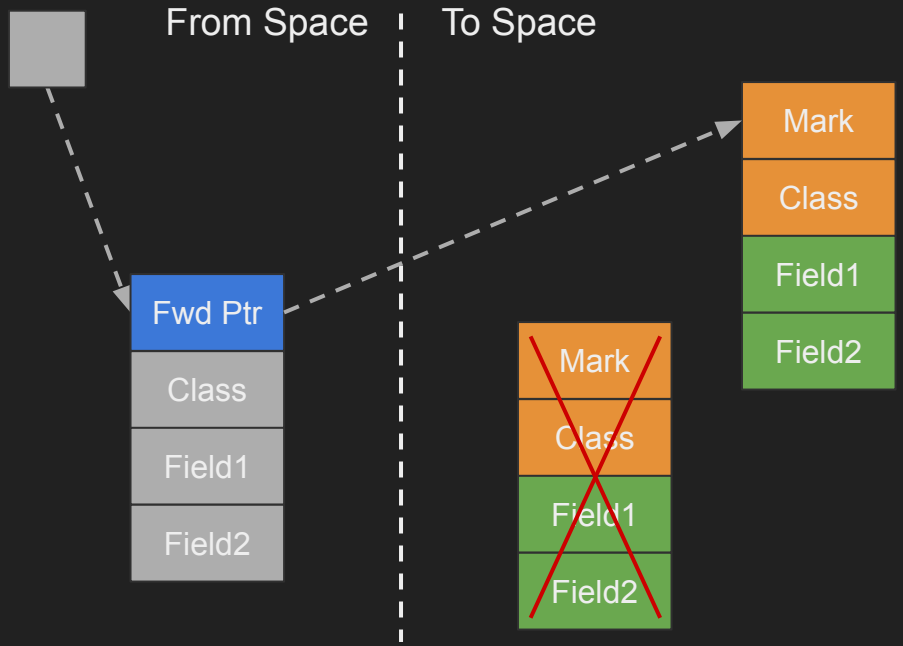
- Evacuation (Atomic Fwd Ptr Update)



simone.bordet@gmail.com

ShenandoahGC

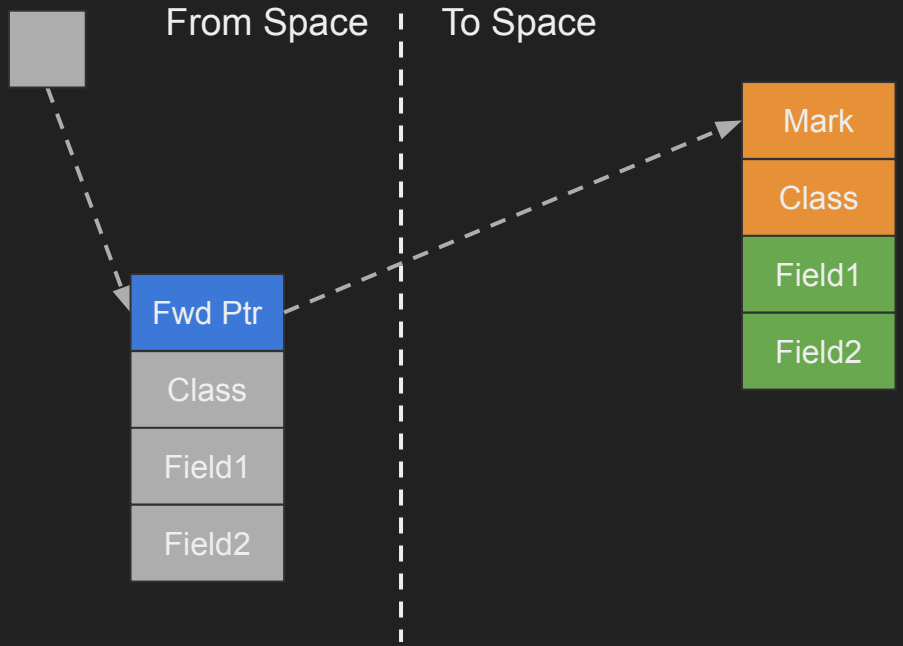
- Evacuation (Application Wins)



simone.bordet@gmail.com

ShenandoahGC

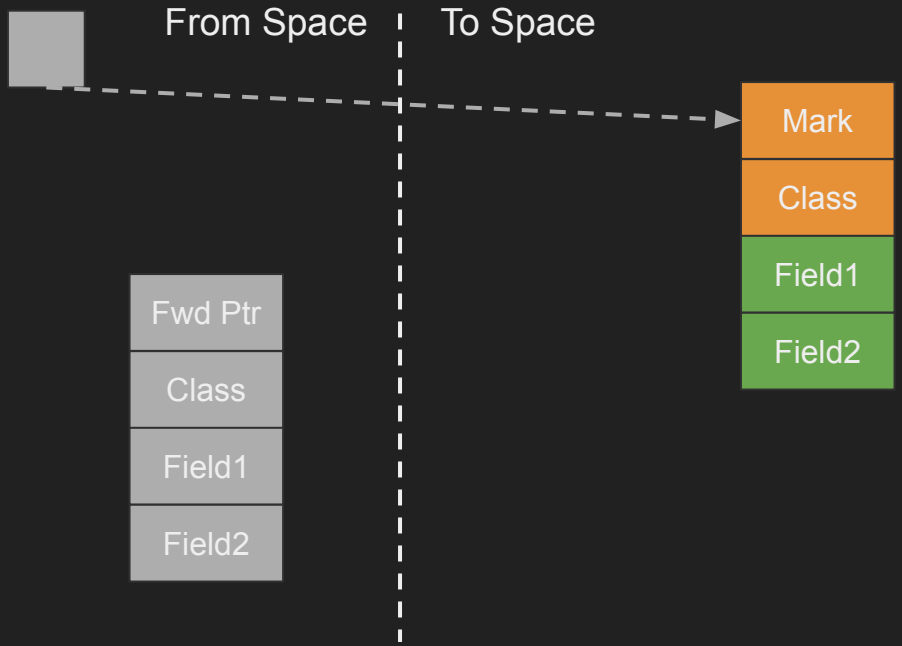
- Evacuation (Application Read/Write)



simone.bordet@gmail.com

ShenandoahGC

- Update References



simone.bordet@gmail.com

ShenandoahGC

- Shenandoah Heuristics
  - Adaptive (default)
    - Maintains free heap amount
  - Static
    - Starts GC when tripping thresholds
  - Compact
    - Small footprint
  - Aggressive
    - Back to back GCs
  - Passive
    - Non concurrent, only Full GCs
  - Traversal

ShenandoahGC

- References
  - Project Wiki
    - https://wiki.openjdk.java.net/display/shenandoah/Main
  - Aleksey Shipilëv, Roman Kennke et al.
    - https://www.youtube.com/watch?v=VCeHkcwfF9Q
    - https://www.youtube.com/watch?v=E1M3hNlhQCg
  - Mailing List
    - shenandoah-dev@openjdk.java.net

Shenandoah Questions?

Conclusions

Conclusions

- Give Concurrent GCs a go

- Stuck with JDK 8?
  - Use Shenandoah

Already using JDK 11+?
  - ZGC or Shenandoah

- Report your feedback!

simone.bordet@gmail.com