Shenandoah: An ultra-low pause time garbage collector for OpenJDK

Christine Flood
Principal Software Engineer
Red Hat

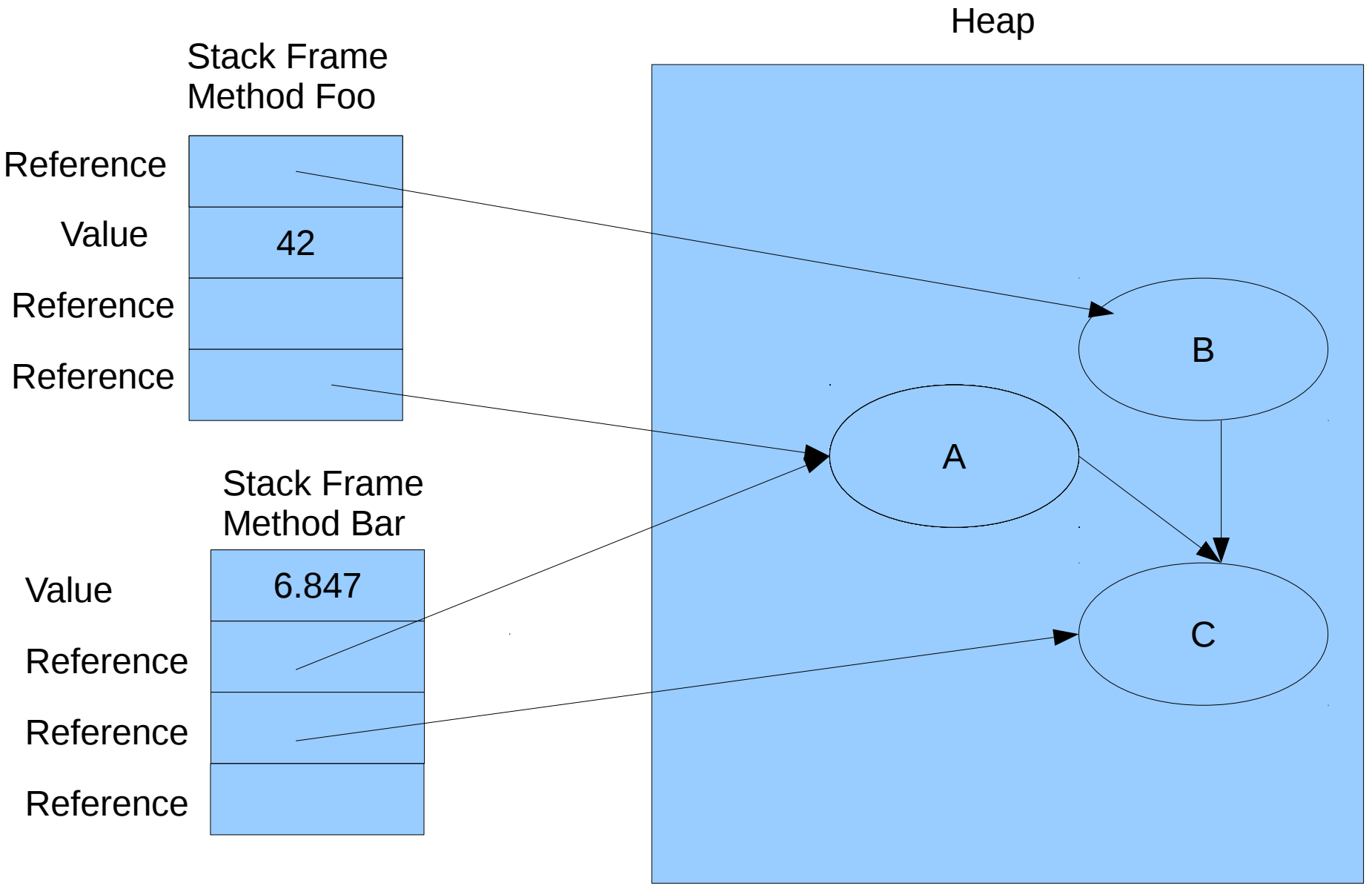
Why do we need another Garbage Collector?

- OpenJDK currently has:
  - SerialGC
  - ParallelGC
  - ParNew/Concurrent Mark Sweep(CMS)
  - Garbage First (G1)

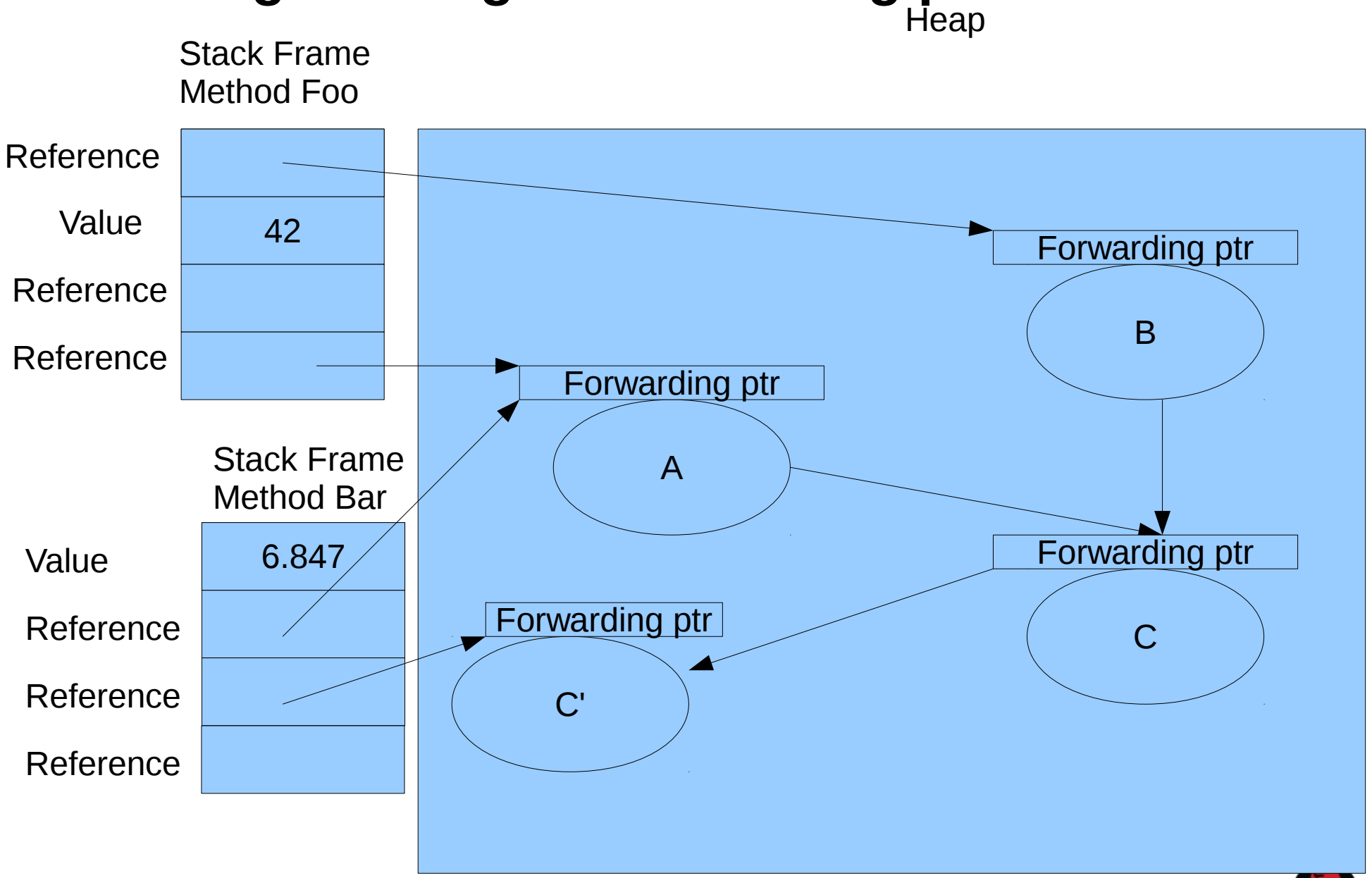
Why do we need another Garbage Collector?

- OpenJDK currently has:
  - SerialGC
  - ParallelGC
  - ParNew/Concurrent Mark Sweep(CMS)
  - Garbage First (G1)
- Shenandoah
  - Pause times similar to CMS.
  - Region based like G1.
  - Concurrent Compaction

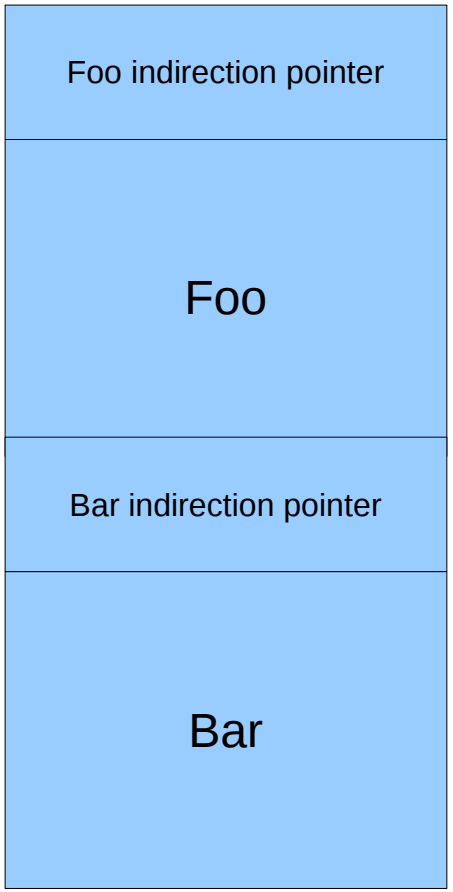
Why is concurrent compaction difficult?



Pause to update the roots, and have heap object accesses go through a forwarding pointer.

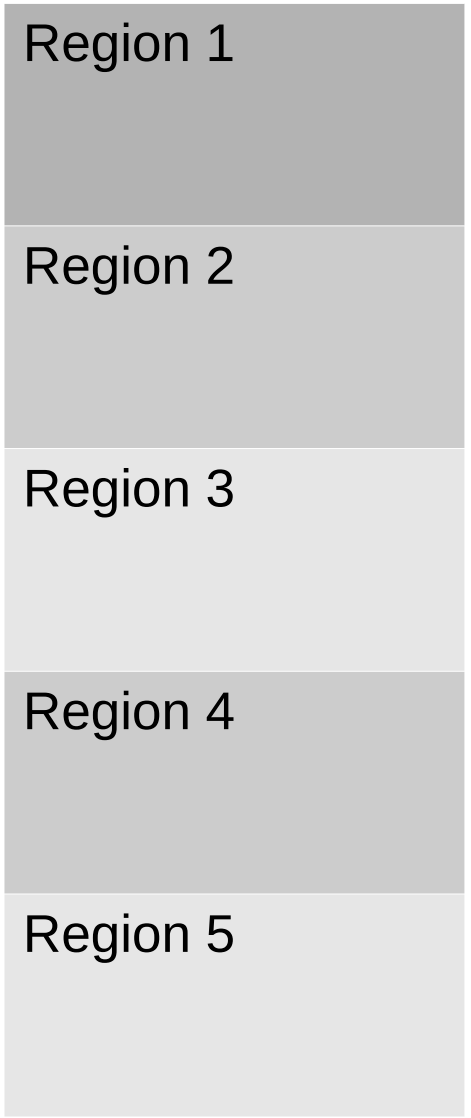


Forwarding Pointers



- You can still walk the heap.
- You can still choose your GC at runtime.
- Software only solution.

Shenandoah divides the heap into regions.



We use concurrent marking to keep track of the live data in each region.

Regions	Live Data
Region 1	20k
Region 2	100k
Region 3	500k
Region 4	10k
Region 5	70k

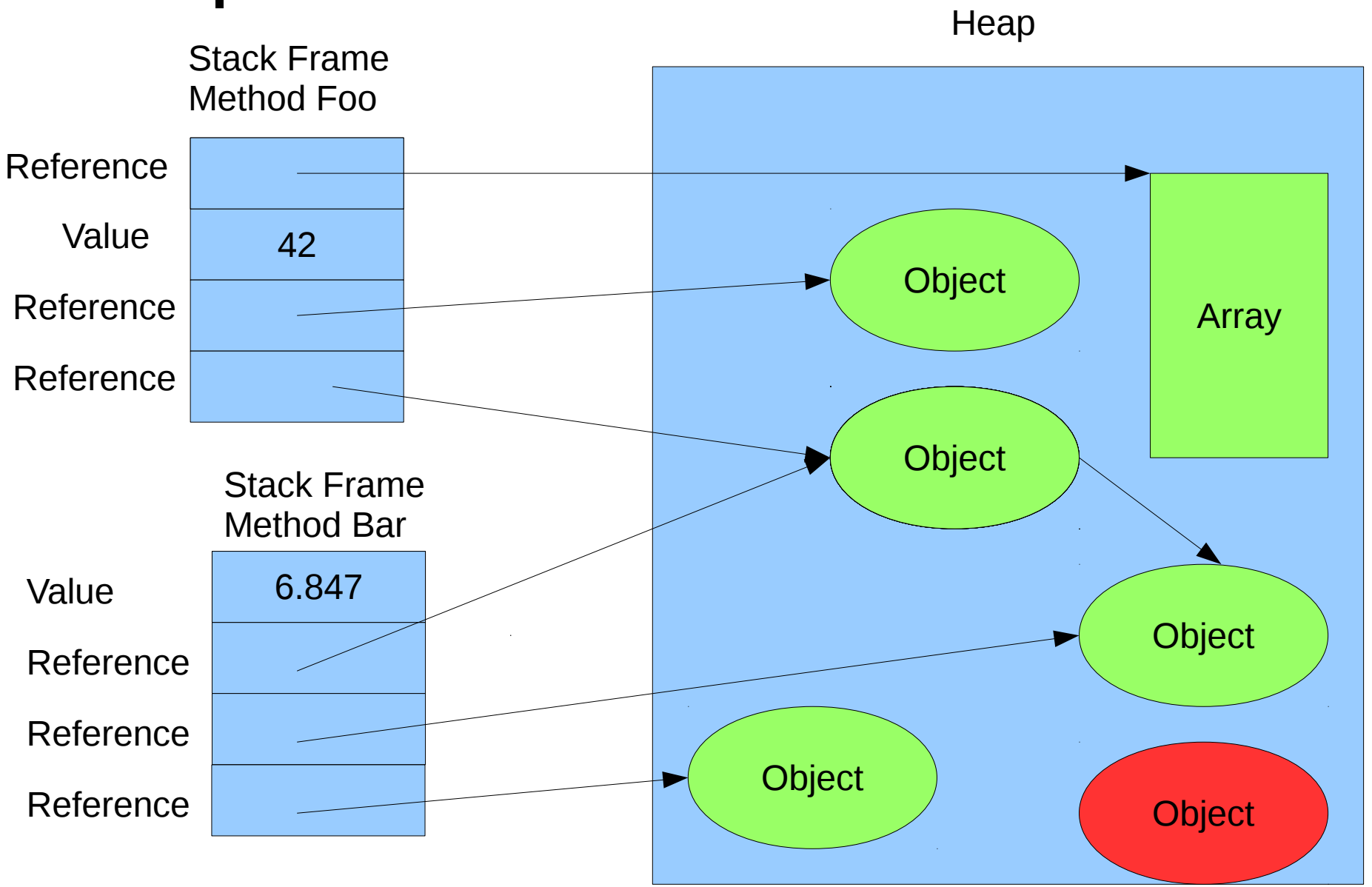
Regions	Live Data
Region 6	200k
Region 7	100k
Region 8	empty
Region 9	empty
Region 10	empty

We pick the most garbage-y regions for evacuation.

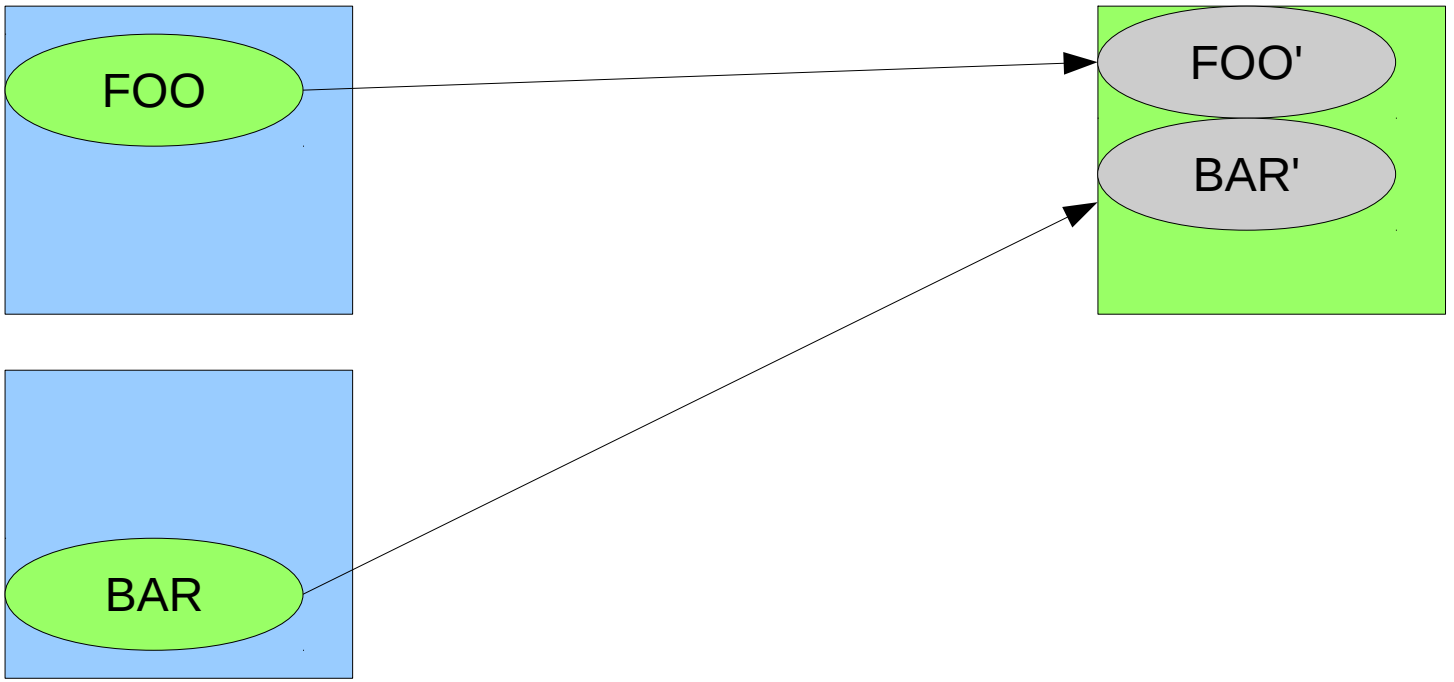
Regions	Live Data
Region 1	20k From-region
Region 2	100k
Region 3	500k
Region 4	10k From-region
Region 5	70k

Regions	Live Data
Region 6	20k
Region 7	100k
Region 8	to-region
Region 9	empty
Region 10	empty

Concurrent Marking tells us which objects need to be copied.



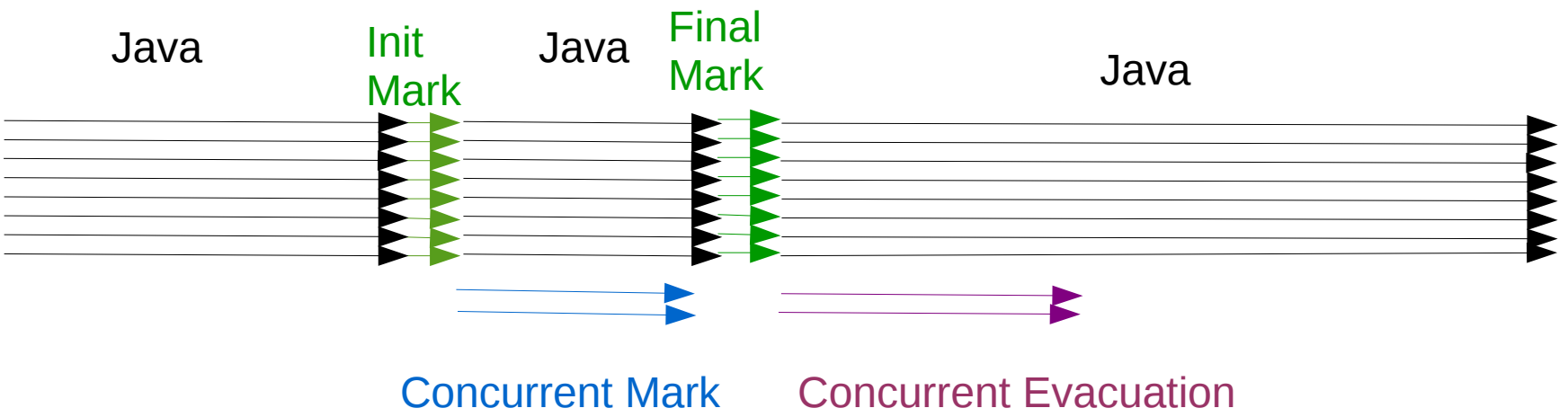
We evacuate the live objects while the Java threads are running.



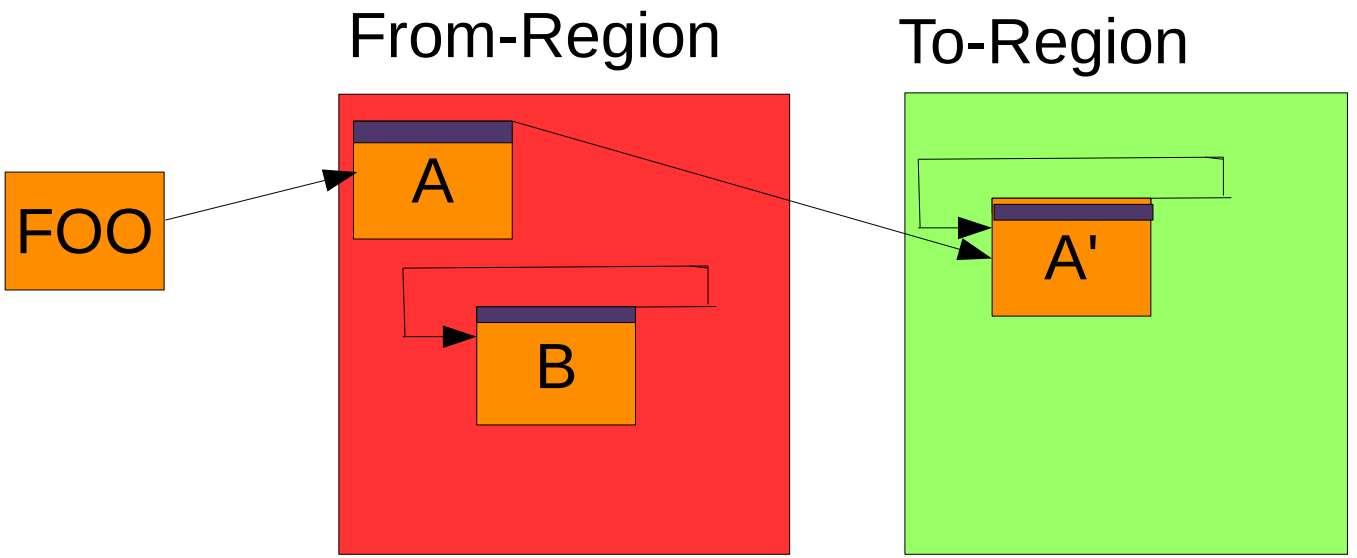
Reclaiming free space

- Eagerly
  - Run another pass over the data to update references.
- Lazily
  - Wait for the next concurrent mark to update references.
- Once the references have been updated we can free the now empty regions.

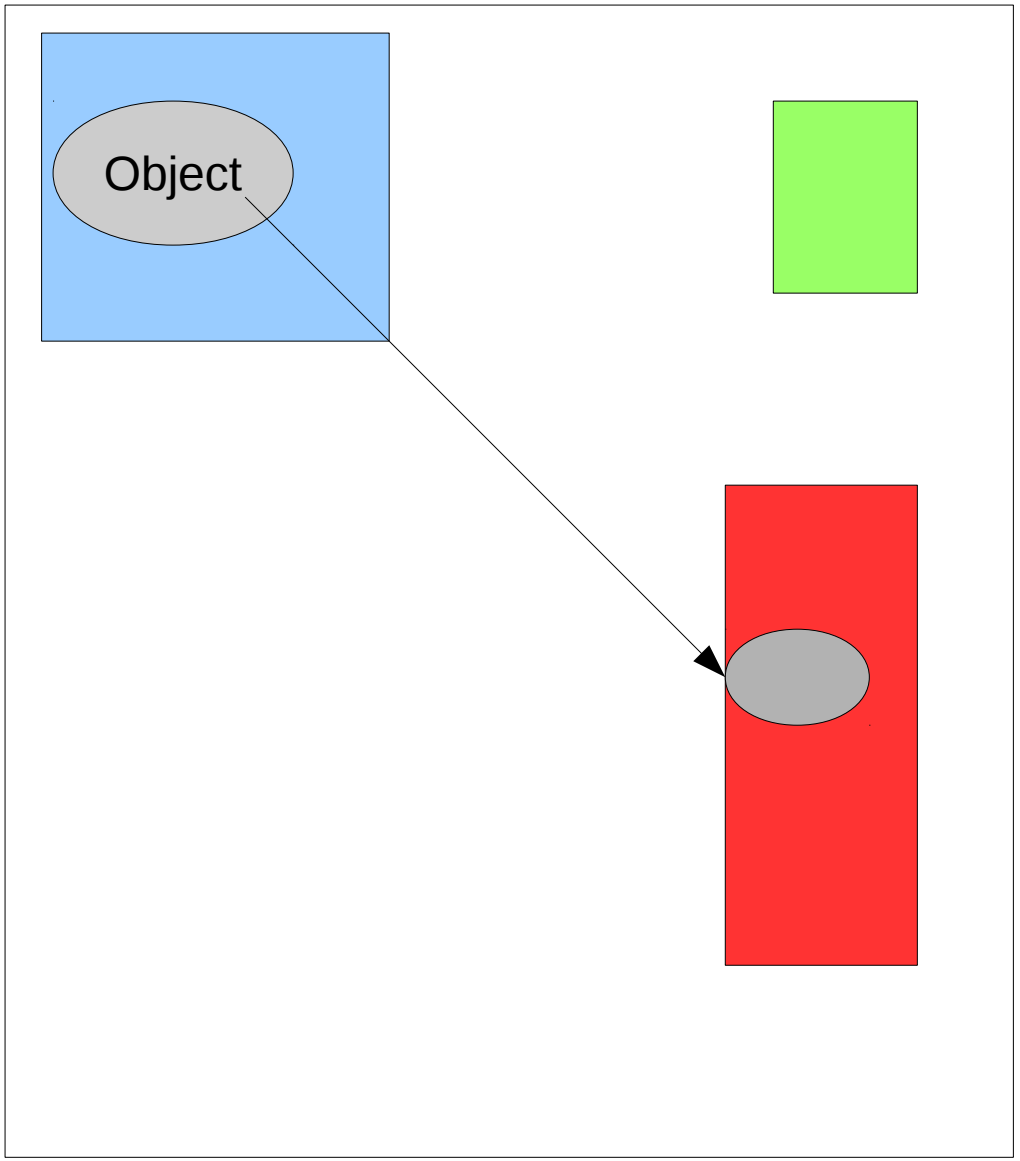
Shenandoah



We use as many threads as are available to do concurrent phases.


From-Region
To-Region
FOO
A
B
A'

Added benefit to forwarding pointers



We can be lazy about updating references to objects in from regions.

We no longer need remembered sets.

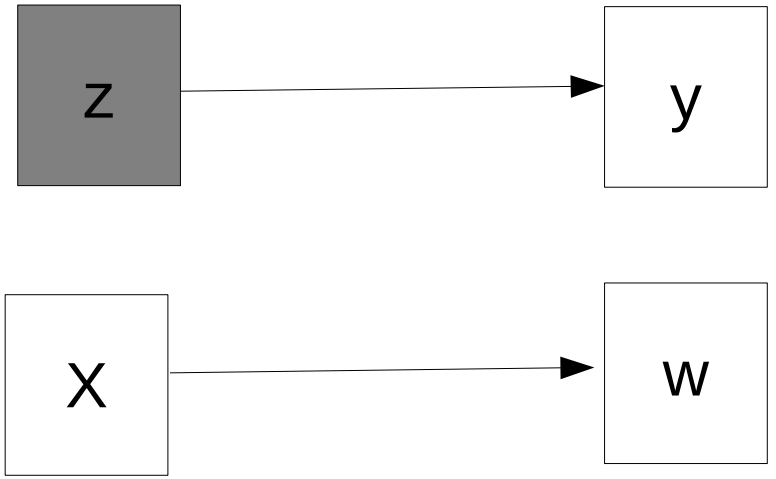
Concurrent Marking

- SATB – Snapshot At The Beginning
  - Anything live at Initial Marking is considered live.
  - Anything allocated since Initial Marking is considered live.
- Used to update references, keep track of amount of live data for each region, and tell us which objects are live and need to be evacuated.

What's tricky about SATB?

Start of Concurrent Marking

Sometime during marking

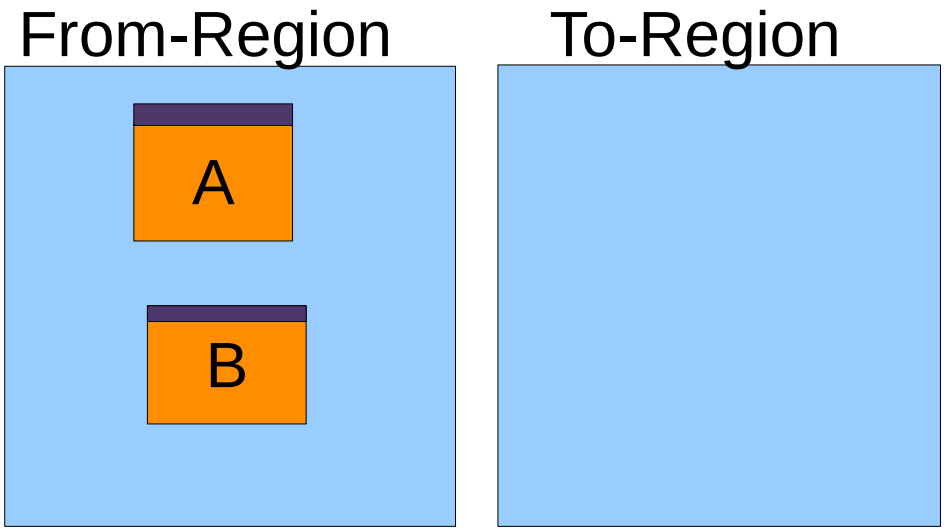


Requires a write barrier to ensure overwritten values get marked.

How to move an object while the program is running.

- Read the forwarding pointer to the from-region.
- Allocate a temporary copy of the object in a to-region.
- Speculatively copy the data.
- CAS the forwarding pointer to point to the new copy.
  - If the CAS fails, another thread already copied the object and you can roll back your speculative copy.

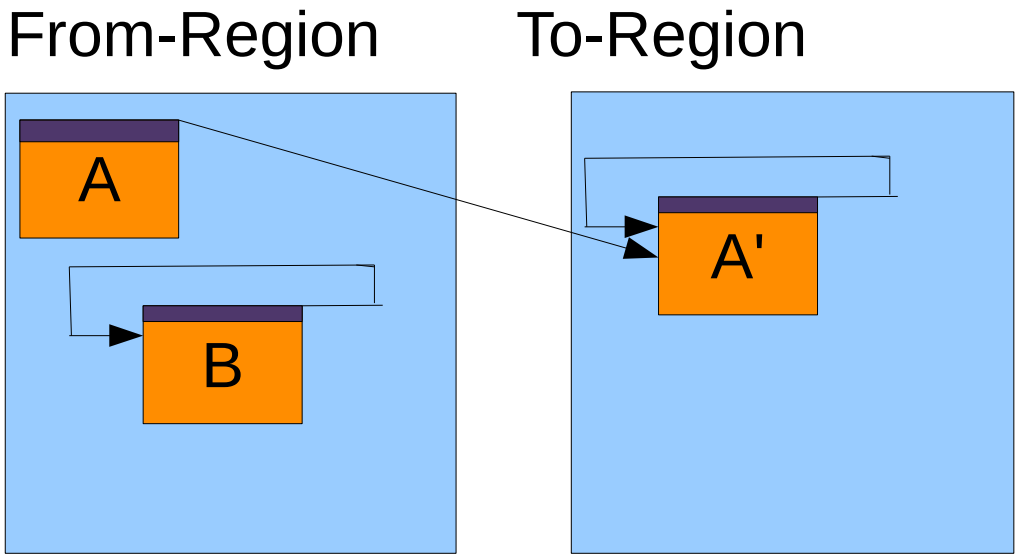
Forwarding Pointers - reads



Reading an object in a From-region doesn't trigger an evacuation.

Note: If reads were to cause copying we might have a “read storm” where every operation required copying an object. Our intention is that since we are only copying on writes we will have less bursty behavior.

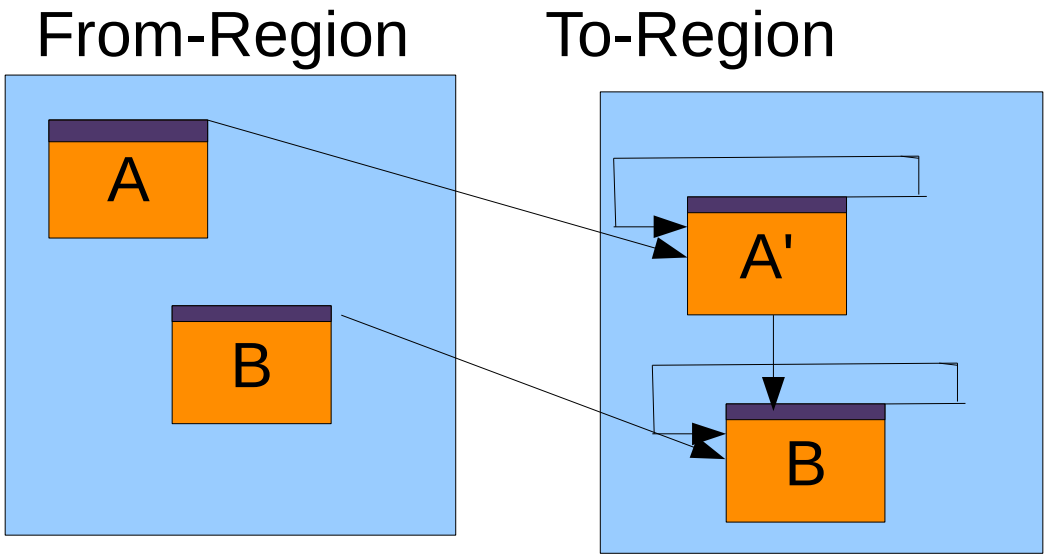
Forwarding Pointers - writes



Writing an object in a From-Region will trigger an evacuation of that object to a To-Region and the write will occur in there.

Invariant: Writes never occur in from-regions.

Forwarding Pointers – writes of references

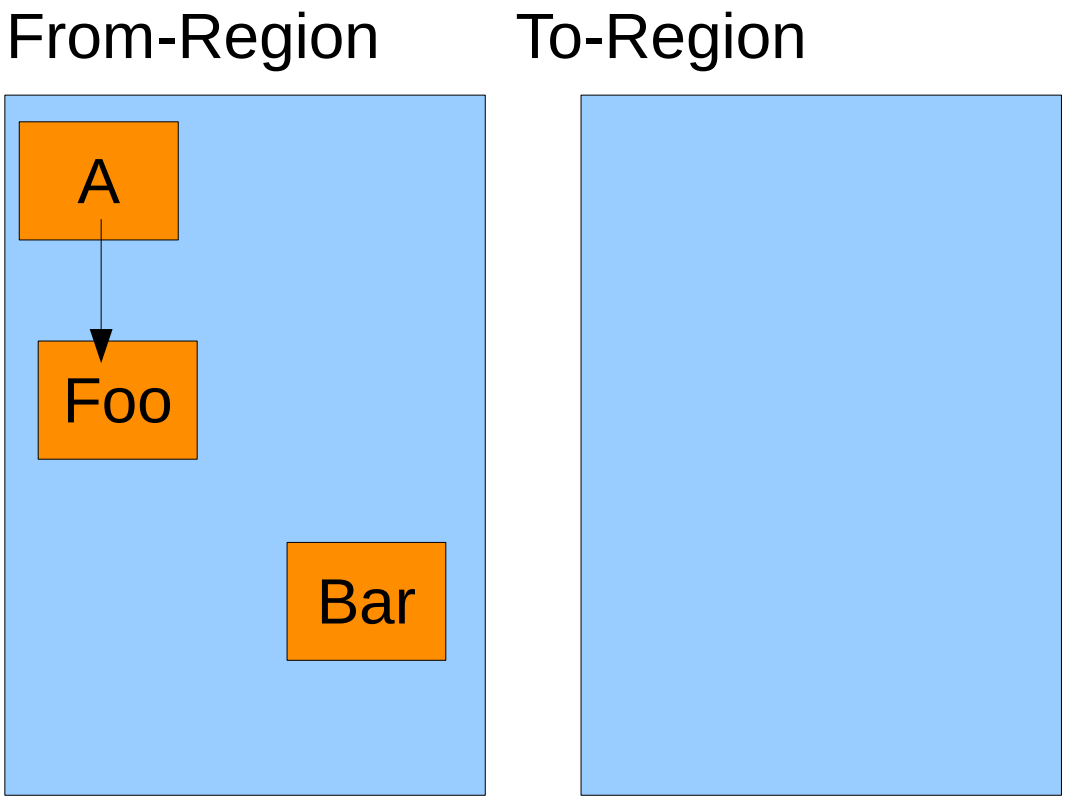


We resolve all references before we write them.

Invariant: Never write a reference to a from-region object into a to-region object.

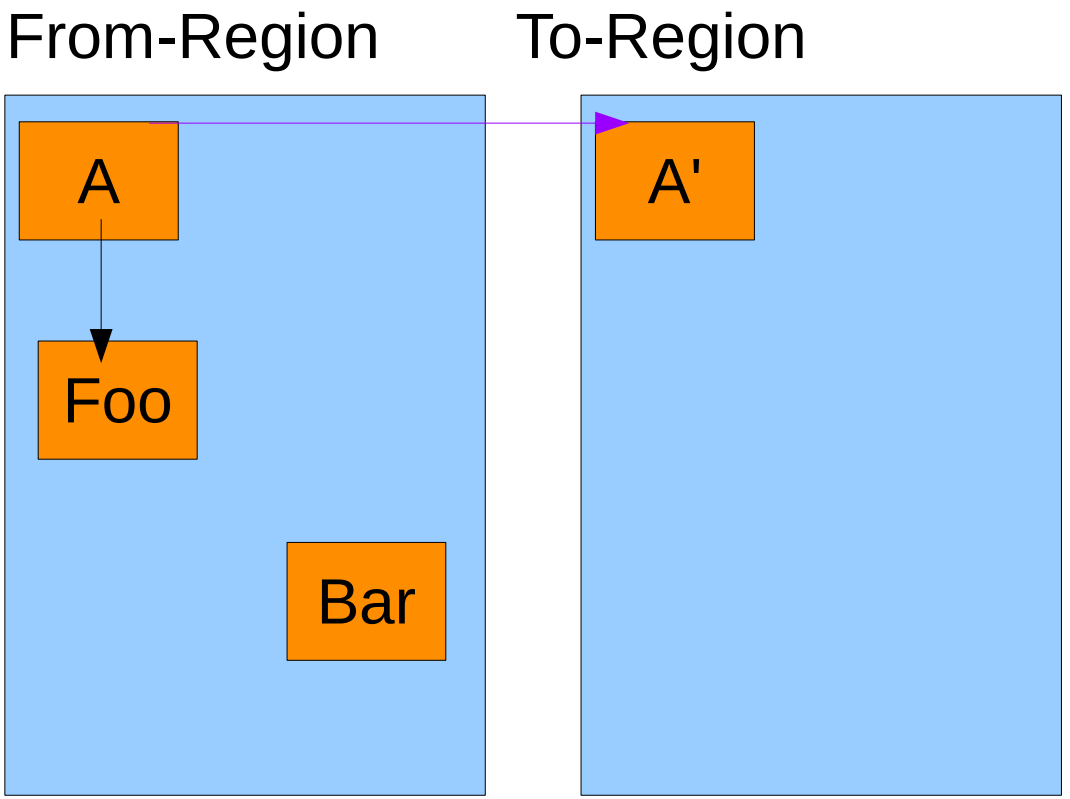
Forwarding Pointers - CAS

- CompareAndSwap(A.X, Foo, Bar)



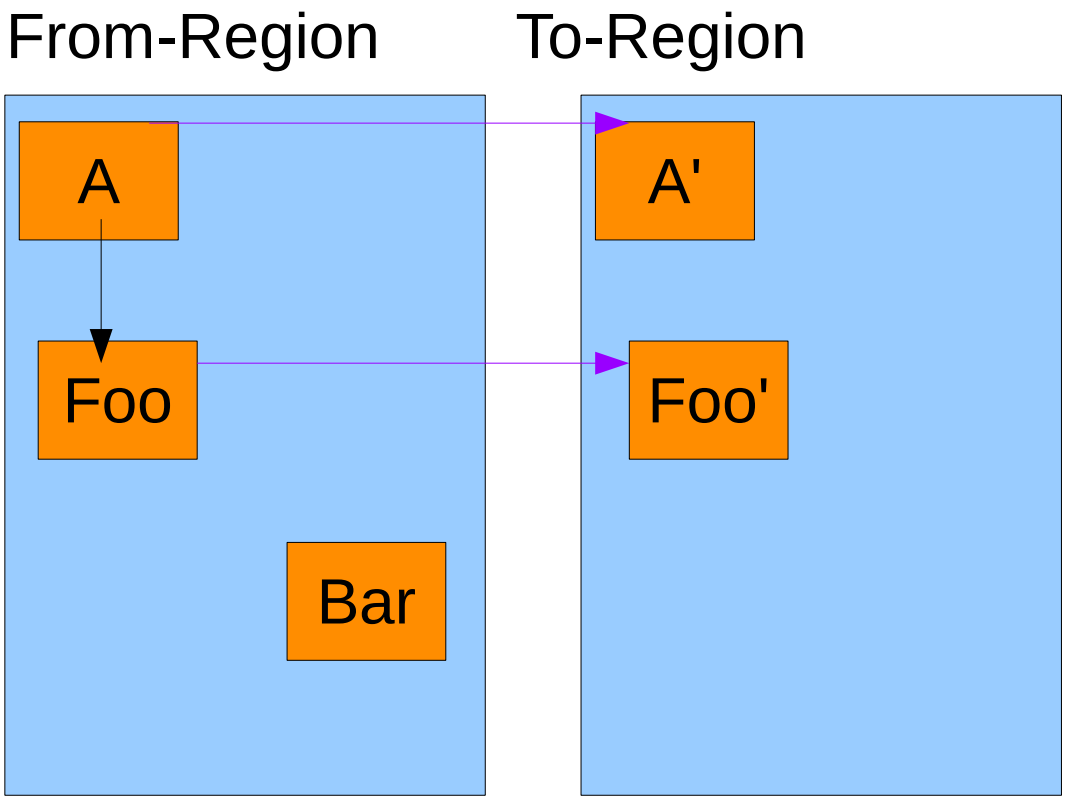
Forwarding Pointers - CAS

- CompareAndSwap(A.X, Foo, Bar)



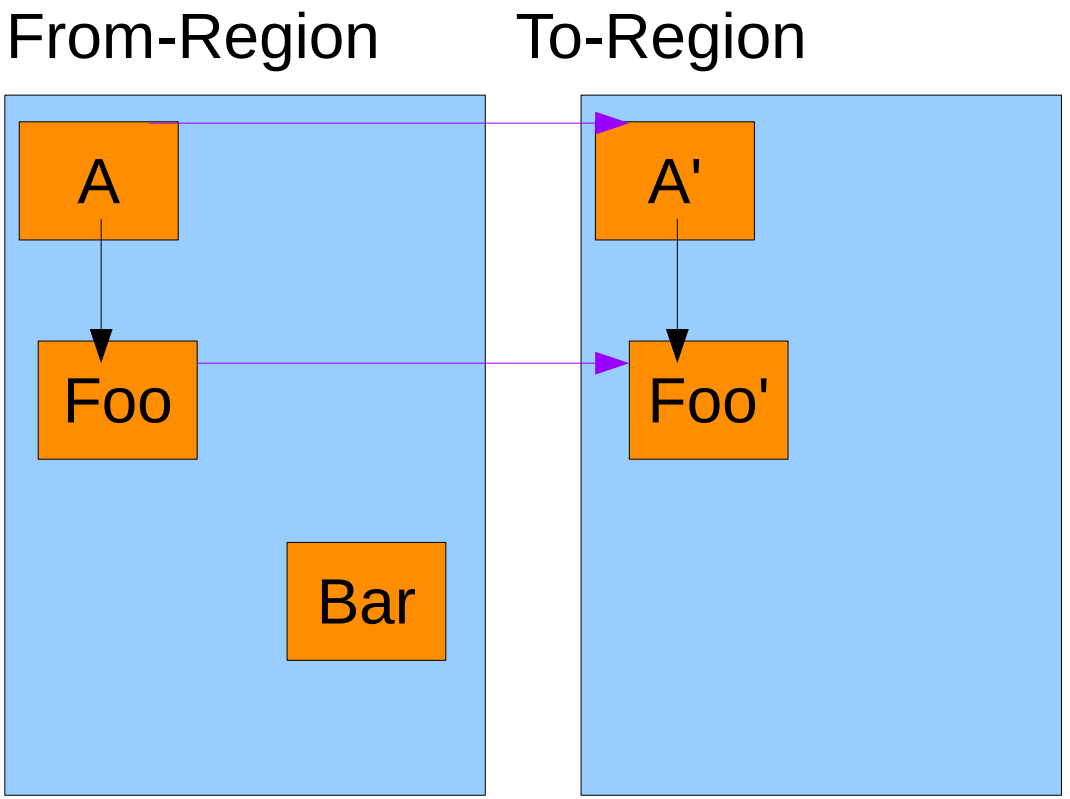
Forwarding Pointers - CAS

- CompareAndSwap(A.X, Foo, Bar)



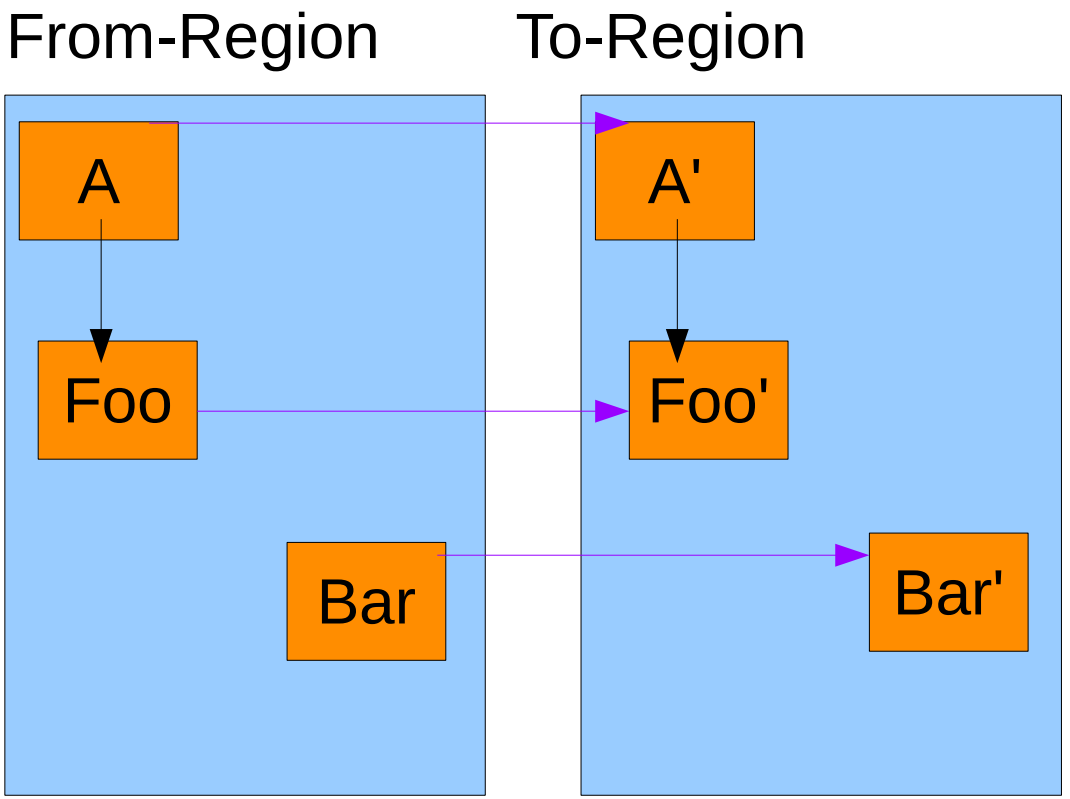
Forwarding Pointers - CAS

- CompareAndSwap(A.X, Foo, Bar)



Forwarding Pointers - CAS

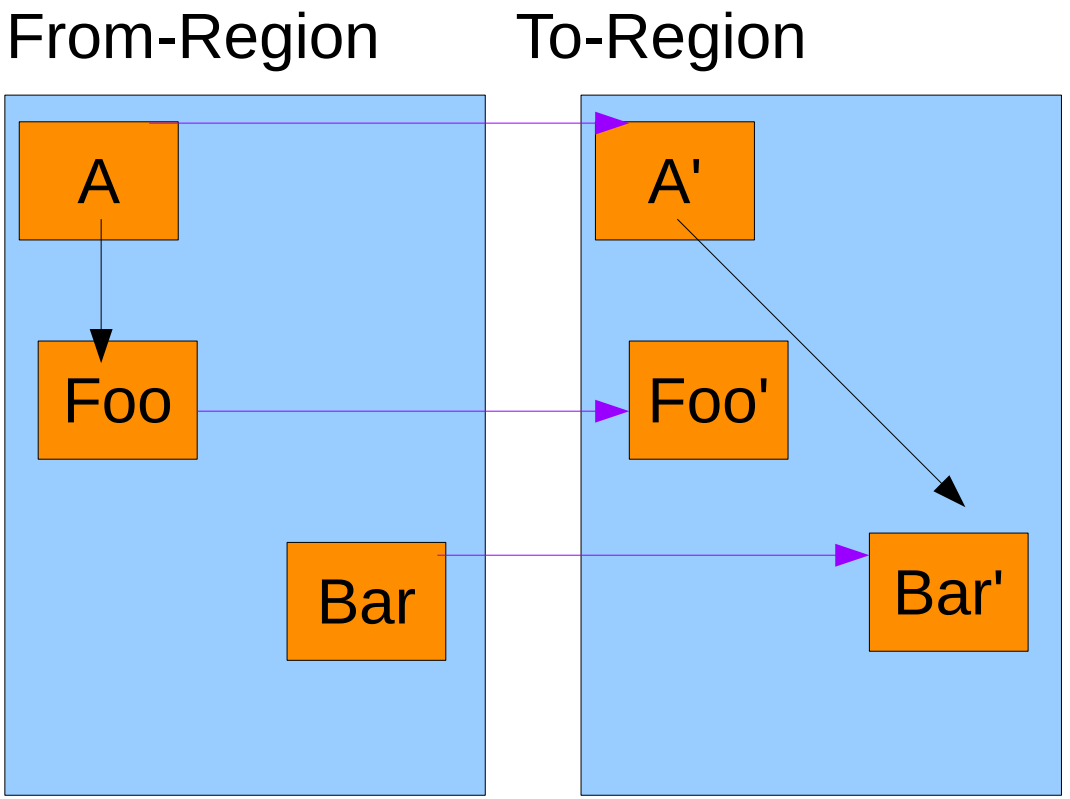
- CompareAndSwap(A.X, Foo, Bar)



Now we can CAS.

Forwarding Pointers - CAS

- CompareAndSwap(A.X, Foo, Bar)



Does that mean a write could keep a Java thread from making progress for an unbounded amount of time?

- No
- Copies are bounded by region size.
- Objects that are larger than a region are treated specially.

Read Barriers?

Unconditional Read Barrier

```
movq    R10, [R10 + #-8 (8-bit)] # ptr
movq    RBX, [R10 + #16 (8-bit)]    # ptr ! Field: java/lang/ClassLoader.parent
```


Write Barriers?

- We need the SATB write barrier which adds previous reference values onto a queue to be scanned.
- We also need write barriers on all writes (even base types) to ensure we copy objects in targeted regions before we write to them.

Costs and Benefits

- Costs
- Space
  - An extra word / object can be expensive if you have a lot of small objects.
- Time
  - Reads and Writes require barriers

- Benefits
- Ultra-low pause times which can be important to interactive and SLA applications.

Current status

- We have something working.
- We can pass small tests specjvm, specjbb
- We have passed the smoke test for
  - Eclipse, Thermostat
- We are working on performance tuning
  - Radargun, Elastic Search/Lucene

Performance Tuning

- One very important area for application specific performance tuning will be Shenandoah heuristics.
  - Lazy Heuristics
    - GC as little as possible
  - Aggressive Heuristics
    - GC as frequently as possible to maintain minimum heap size.
  - ...

Encouraging preliminary results SpecJVM2008 compiler

- Not our target application.
- We are just starting performance tuning.
- Shenandoah
  - Initial Mark (avg=5.65ms,max=9.53ms, total=1.40s)
  - Final Mark (avg=8.74ms,max=15.43ms,total=2.17s)
- As compared to G1
  - (avg=31.38ms, max=75.48ms, total=6.84s)

References

- “Trading Data Space for Reduced Time and Code Space in Real-Time Garbage Collection on Stock Hardware” - Brooks
- “Garbage-first garbage collection” - Detlefs, Flood, Heller, Printezis.

Future Work

- Finish big application testing.
- Move the barriers to right before code generation.
- Heuristics tuning.
- Round Robbin Thread Stopping?
- NUMA Aware?

More information

- Download the code and try it.
  - http://icedtea.classpath.org/wiki/Shenandoah
- Blogs
  - http://christineflood.wordpress.com/
  - http://rkennke.wordpress.com/
- Email
  - chf@redhat.com
  - rkennke@redhat.com