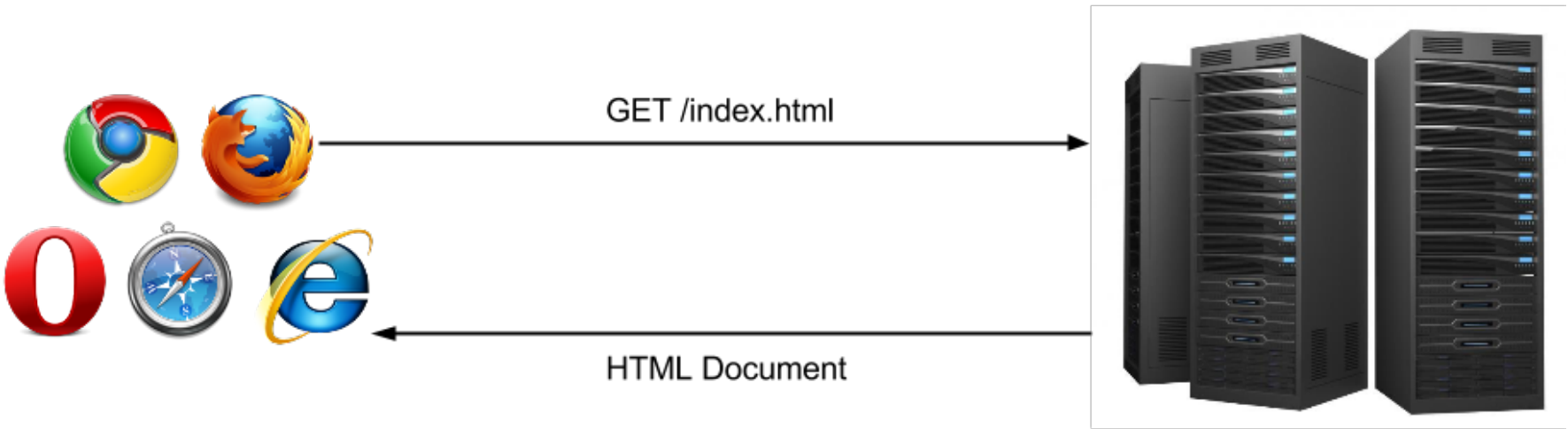
Prioritized Garbage Collection
Using the Garbage Collector to Support Caching

Diogenes Nunez, Samuel Z. Guyer, Emery D. Berger

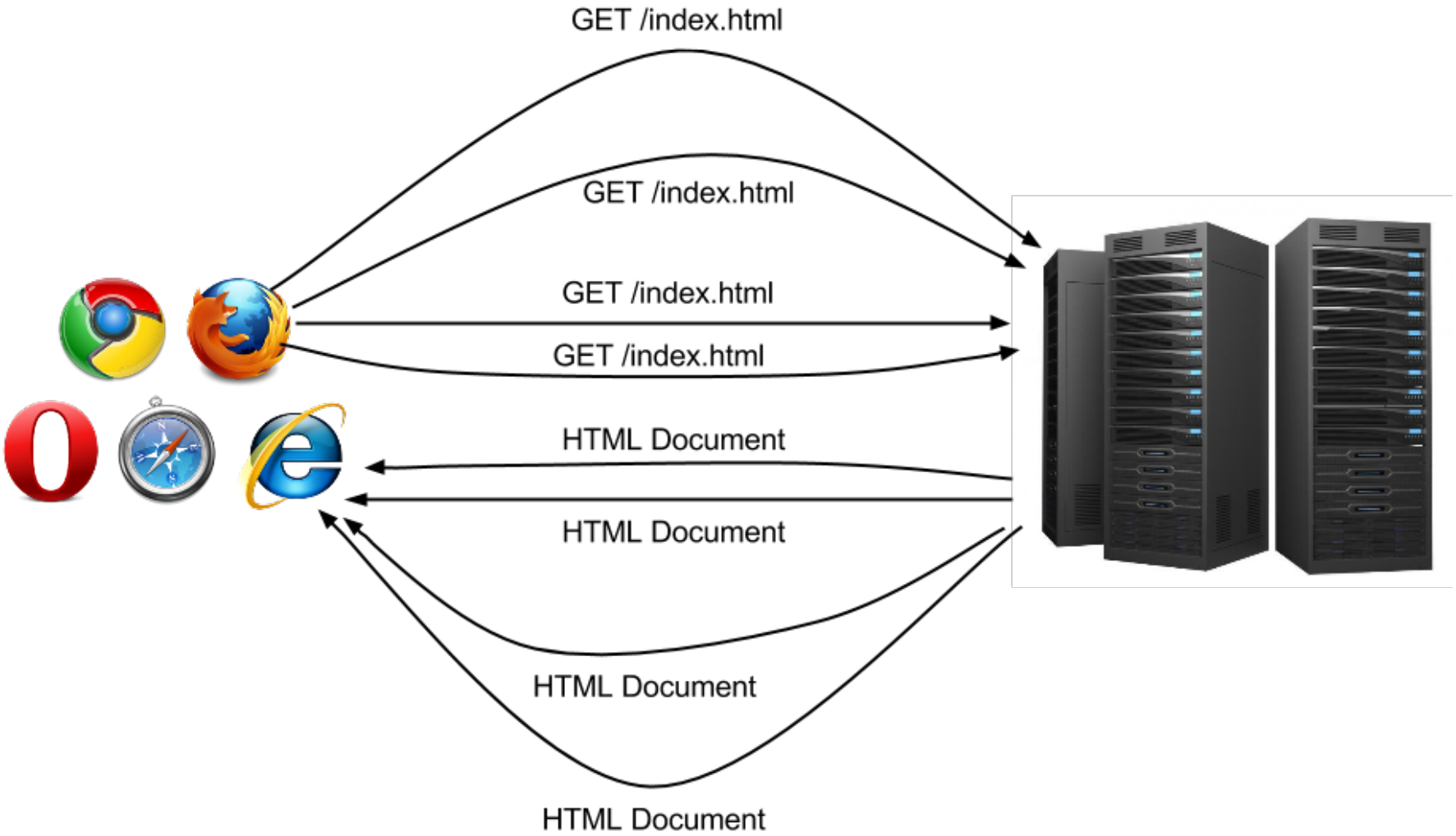
Tufts University, University of Massachusetts Amherst

November 2, 2016

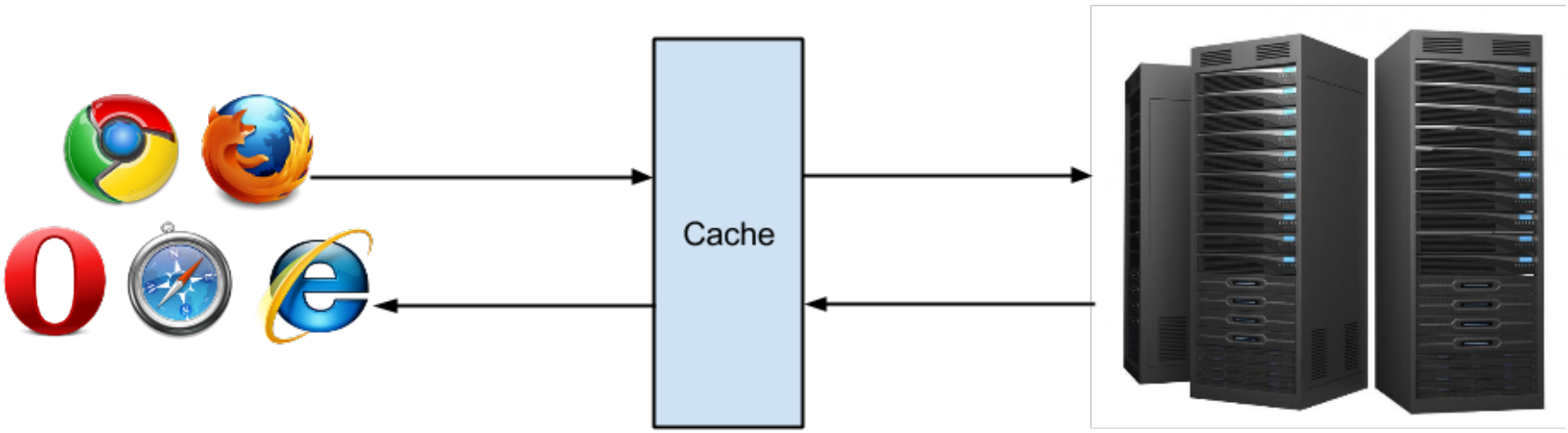
Caches are everywhere



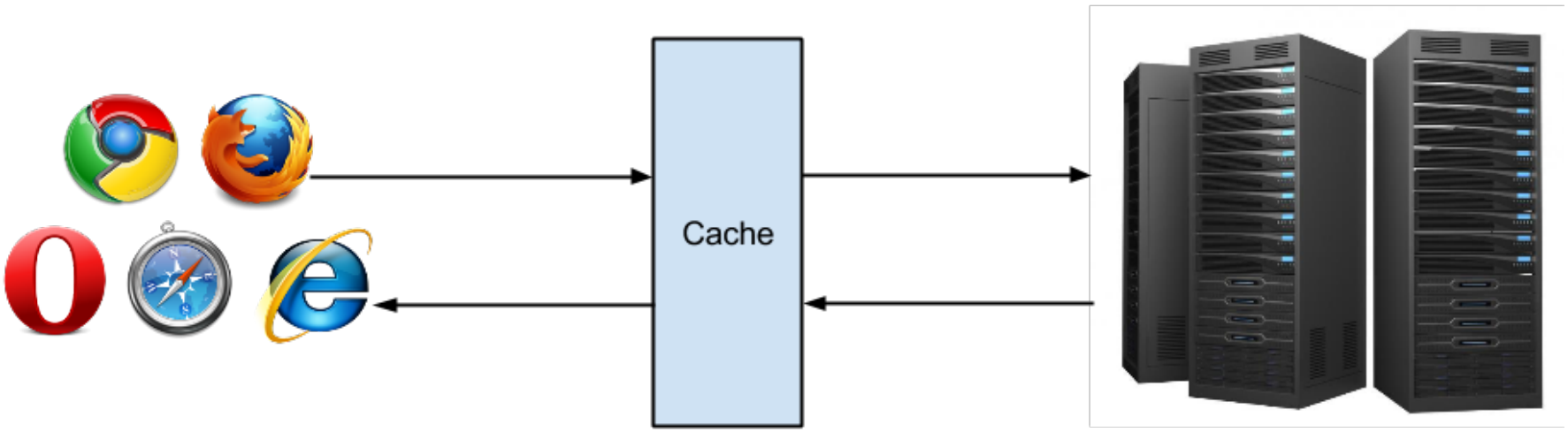
Caches are everywhere



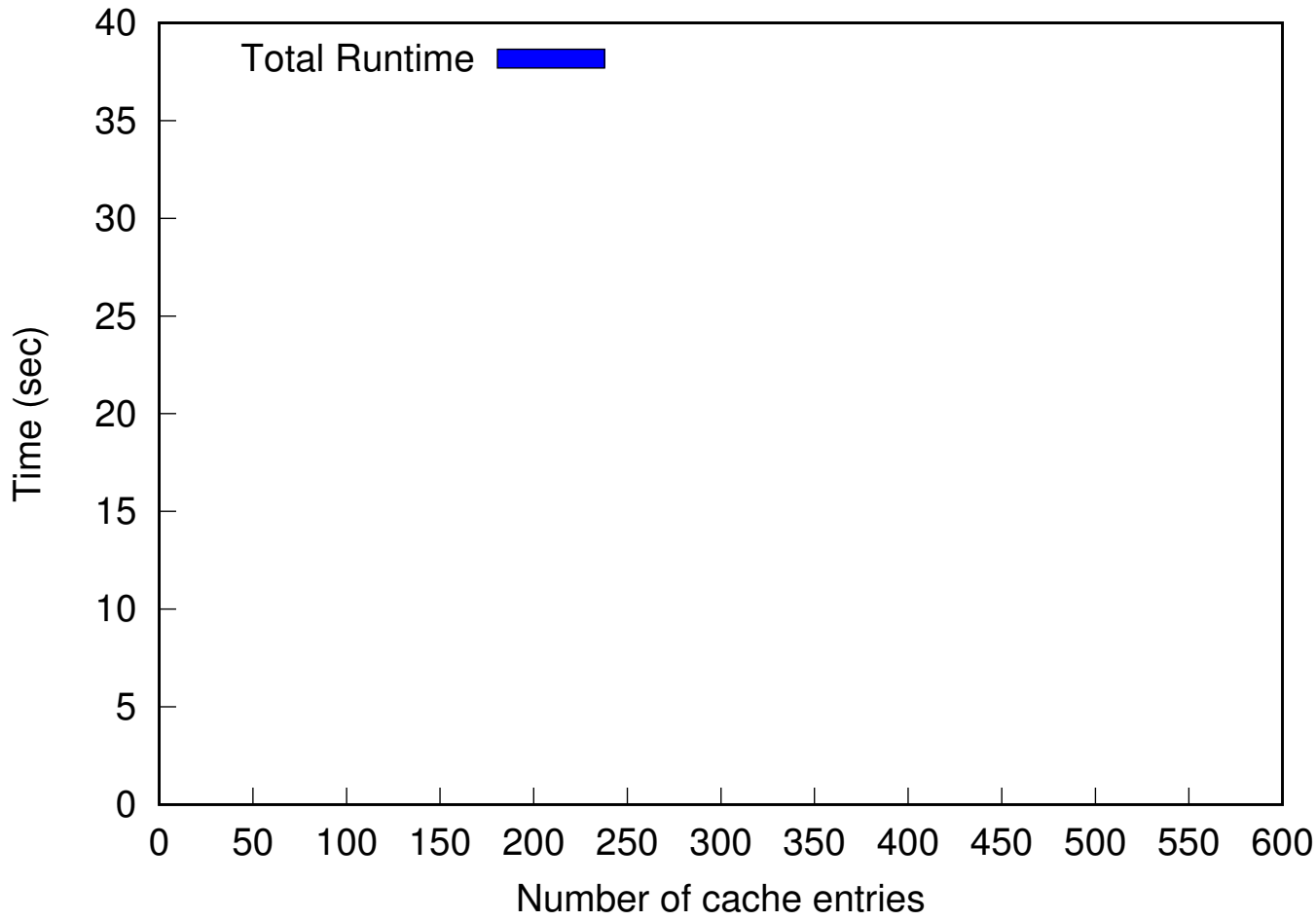
Caches are everywhere



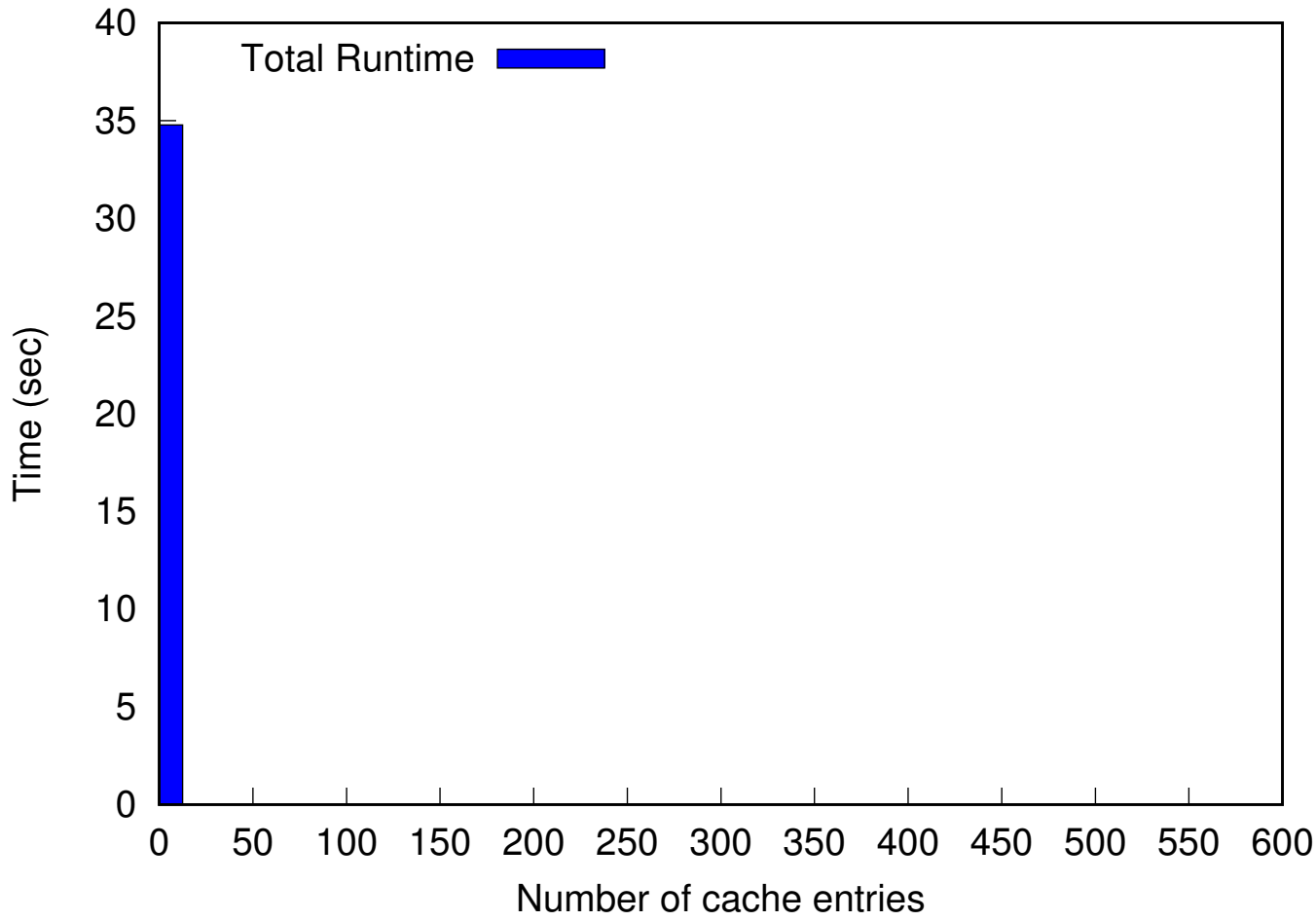
Caches are everywhere



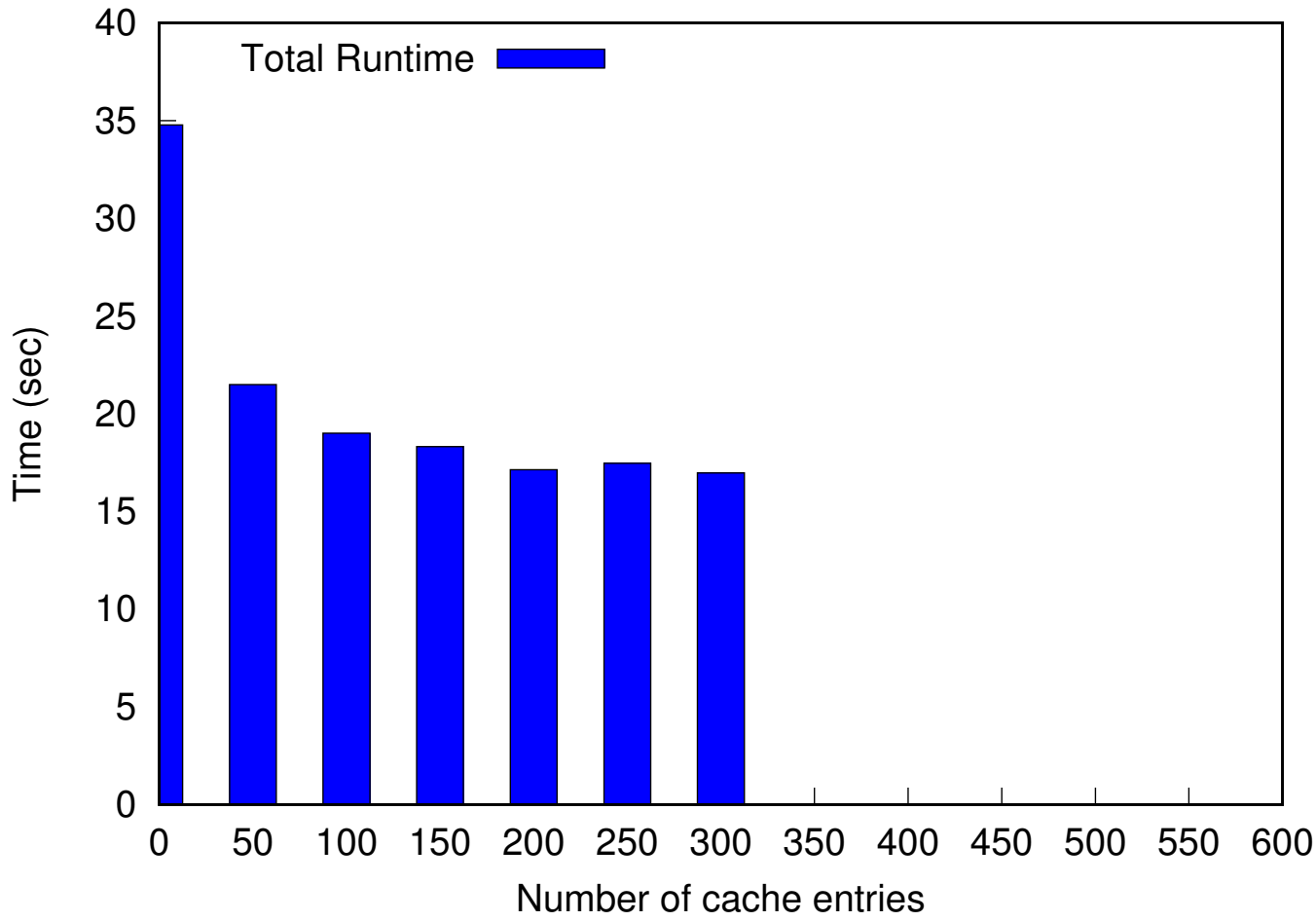
Caches are a space-time tradeoff.



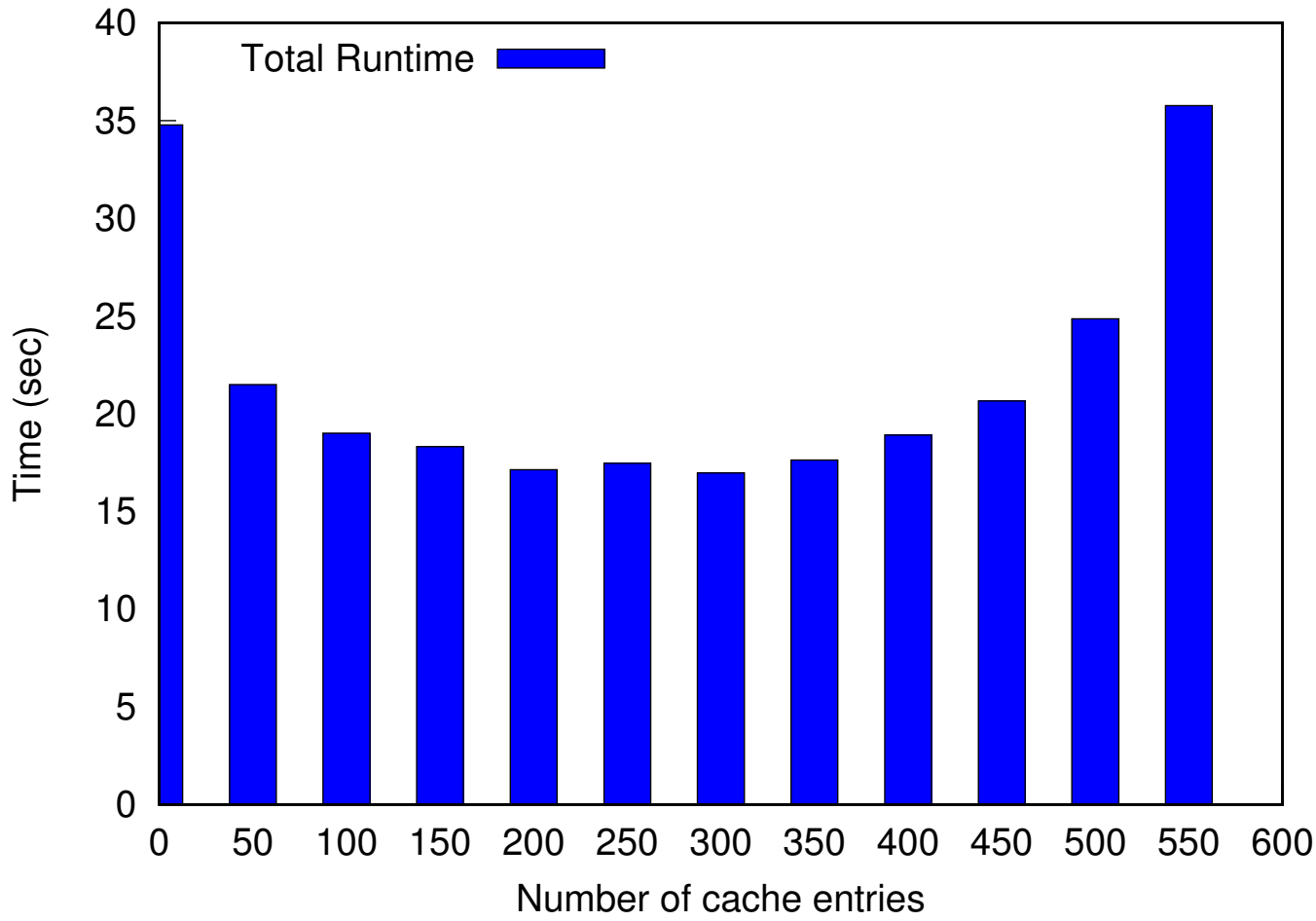
Caches are a space-time tradeoff.



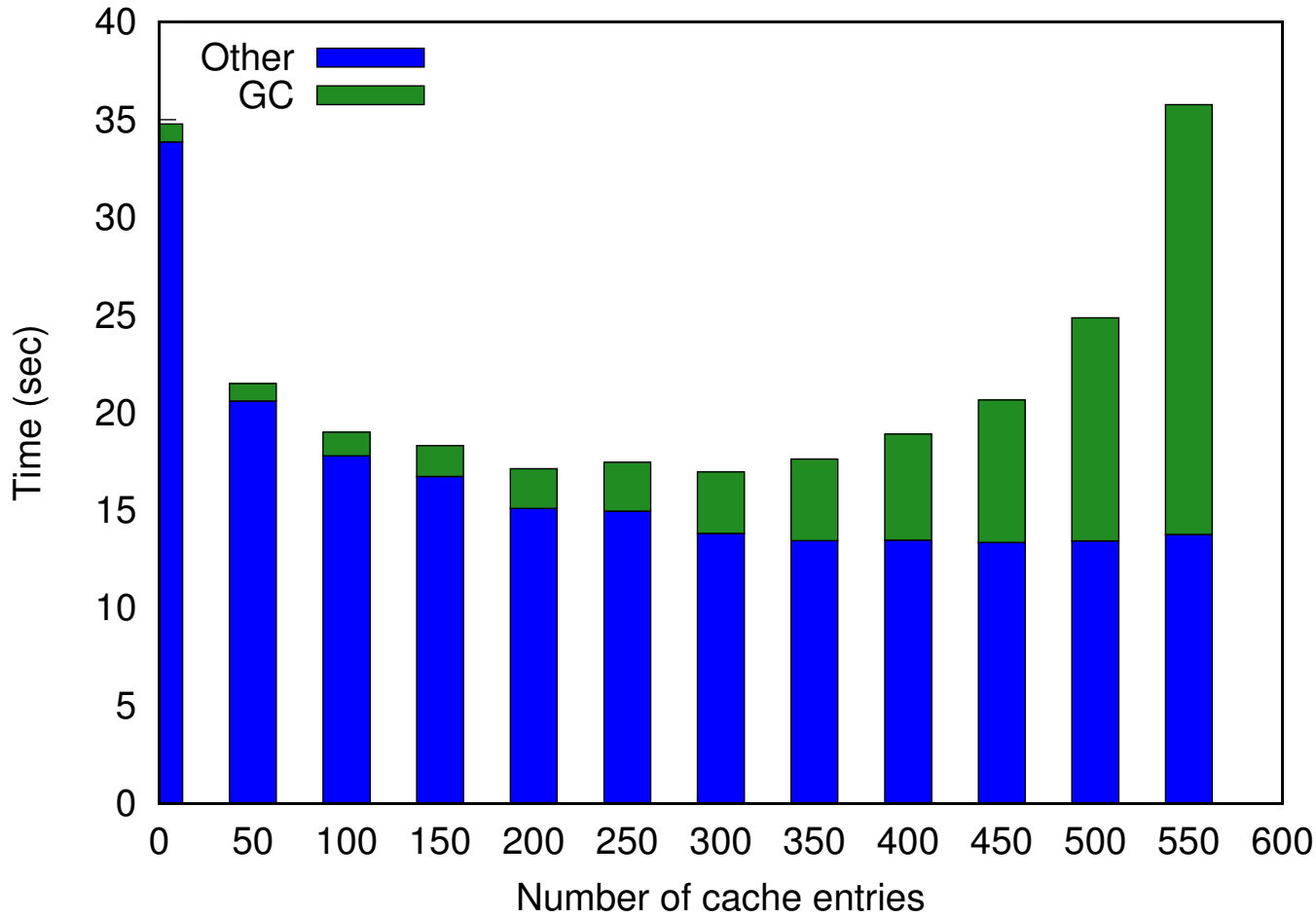
Caches are a space-time tradeoff.



Caches are a space-time tradeoff?



Garbage collection dominates runtime.

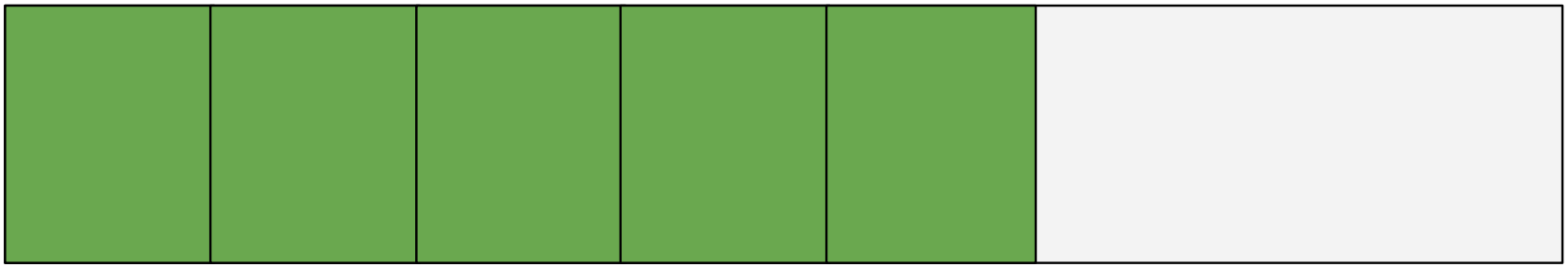


Perfect Cache Size

Elements can be too small.

- The cache can use more space.

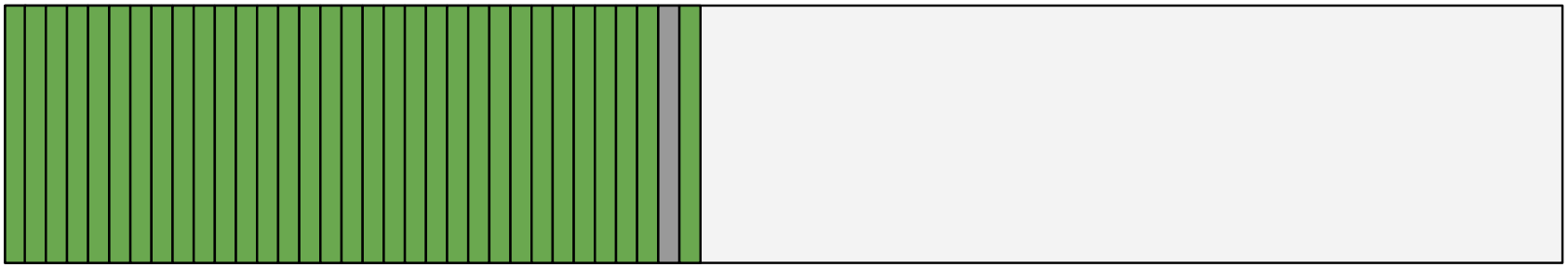
Elements can be too big.



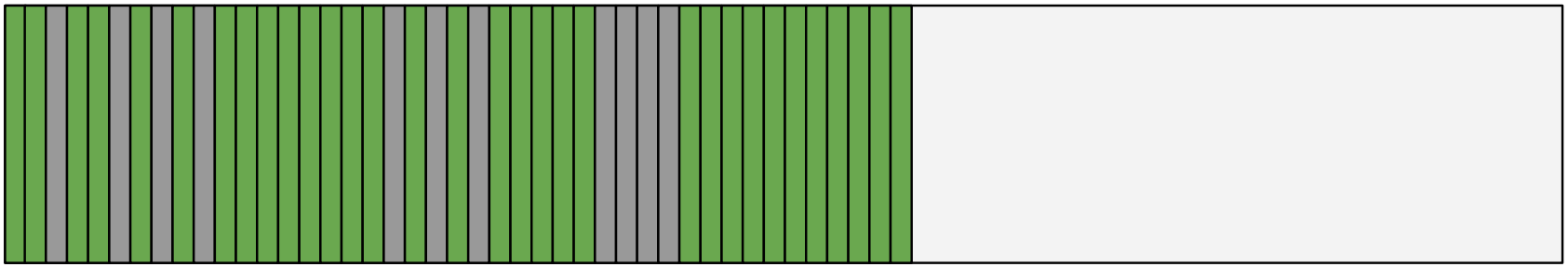
- The cache needs to use less space.

Evicted elements remain in memory until next GC.

Evicted elements remain in memory until next GC.



Evicted elements remain in memory until next GC.



Problem

Can we . . .

- Keep the garbage collector from dominating program run time
- Handle entries of various sizes

Soft References

- Removed prior to an `OutOfMemory` exception
- JVMs can remove them during normal operation
  - Hotspot removes them over time in a Least-Recently-Used (LRU) order

All soft references are equal.

- All Soft References placed in a global queue
- Queue is cleared in an LRU policy over time
- Results in removal of cache values from cold code

Our Solution

Make the Garbage Collector aware of the cache and its contents

Our Solution

- Prioritized Garbage Collector
  - Separate the references into logical spaces with unique eviction policies
  - Compute how much memory each reference is responsible for
- Sache
  - Use the Prioritized Garbage Collector to limit the memory used by stored values

Prioritized Garbage Collection: Using PrioSpaces

```
//Specify memory limit in number of bytes...
PrioSpace<Tree> space = VM.getSpace(500);

//.. or percentage of heap
PrioSpace<Tree> space = VM.getSpace(0.6);
```

Register objects into a PrioSpace

```
Tree t;
PrioReference<T> ref = space.newReference(t);
```

Updates the reference's priority every use.

```
PrioReference<Tree> ref;
Tree t = ref.get();
/* Results in a call to
 * space.updateReference(t);
 */
```

Compare two PrioReferences

```
int compare(PrioReference<T> a,
            PrioReference<T> b);
```

- Determines the eviction policy of the space given by programmer

Compare two PrioReferences

```
int compare(PrioReference<T> a,
            PrioReference<T> b);
```

- Determines the eviction policy of the space given by programmer
- Determines the priority of a reference in the space
  - Higher priority $\rightarrow$ More likely to *not* be evicted

Prioritized Garbage Collection: Cleaning PrioSpaces

- For each `PrioSpace`, garbage collector calculates and stores the amount of memory used by strucures
- Garbage collector frees structures according to priority of the references

Sache (**S**pace **A**ware Ca**che**)

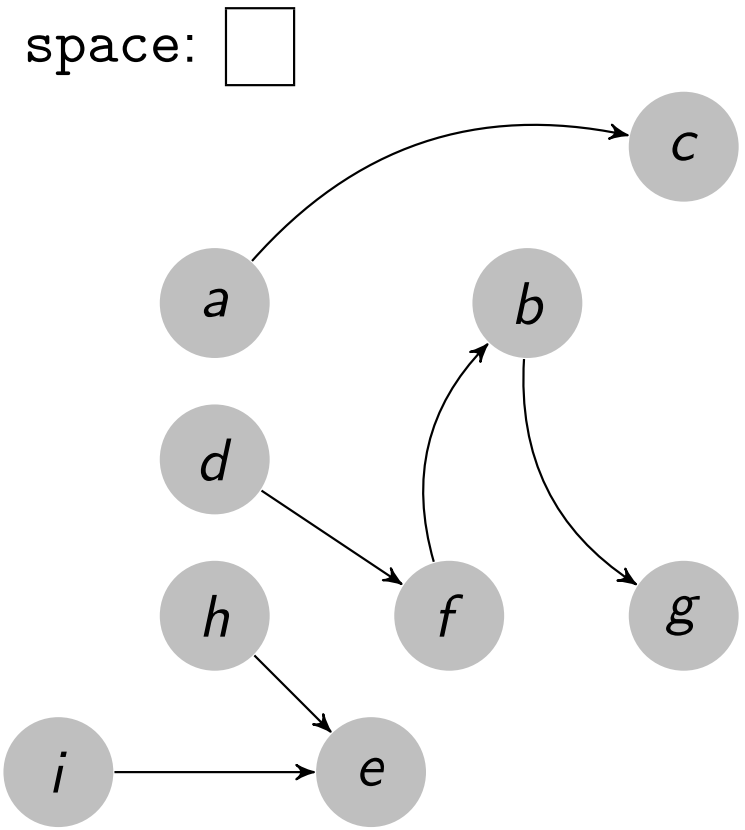
- User puts a limit on size of Sache in bytes
- Tell the prioritized garbage collector what values are in the cache and the order of eviction
  - Current eviction policy is LRU
- Measure the amount of memory used by these values during garbage collection
- If keeping a value means the Sache exceeds the size limit, evict that value

Sache: Interface

```
class Sache <K,V>
    extends HashMap<K, PrioReference<V>>
{
  protected int priority;
  protected PrioSpace<V> priospace;

  public Sache(long maxSize);

  public boolean put(K key, V value);
  public V get(K key);
  public V remove(K key);

  private void update();
}
```

Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

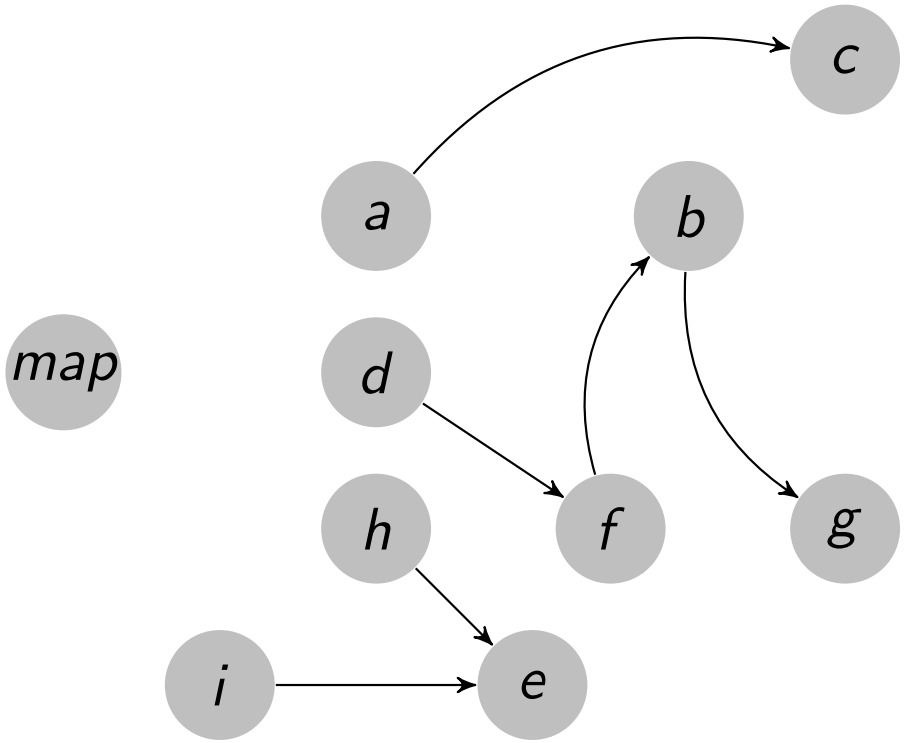


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: 

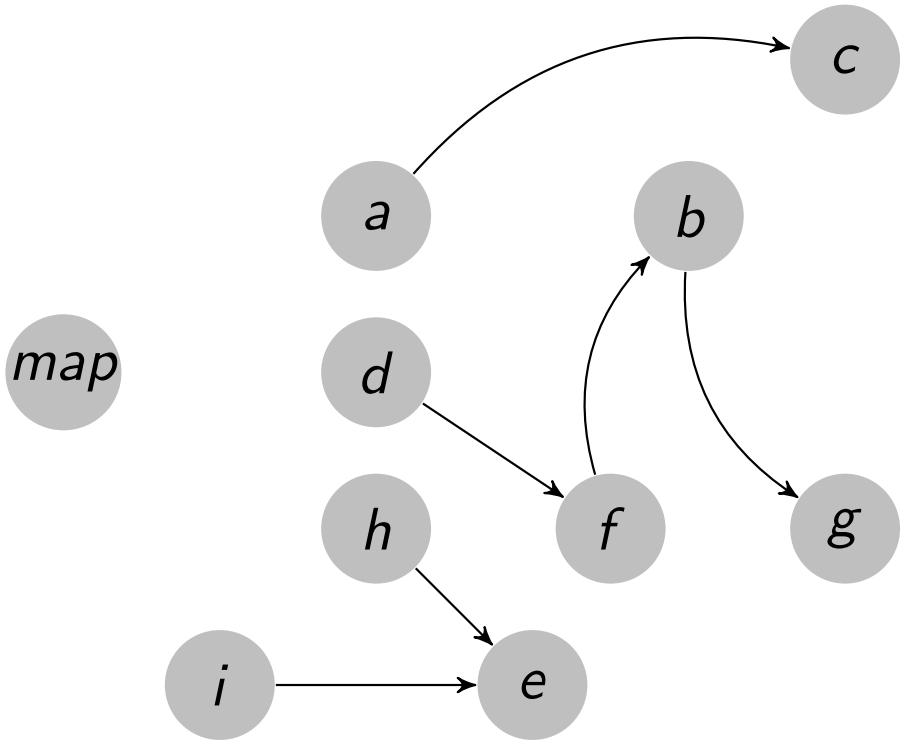


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: 

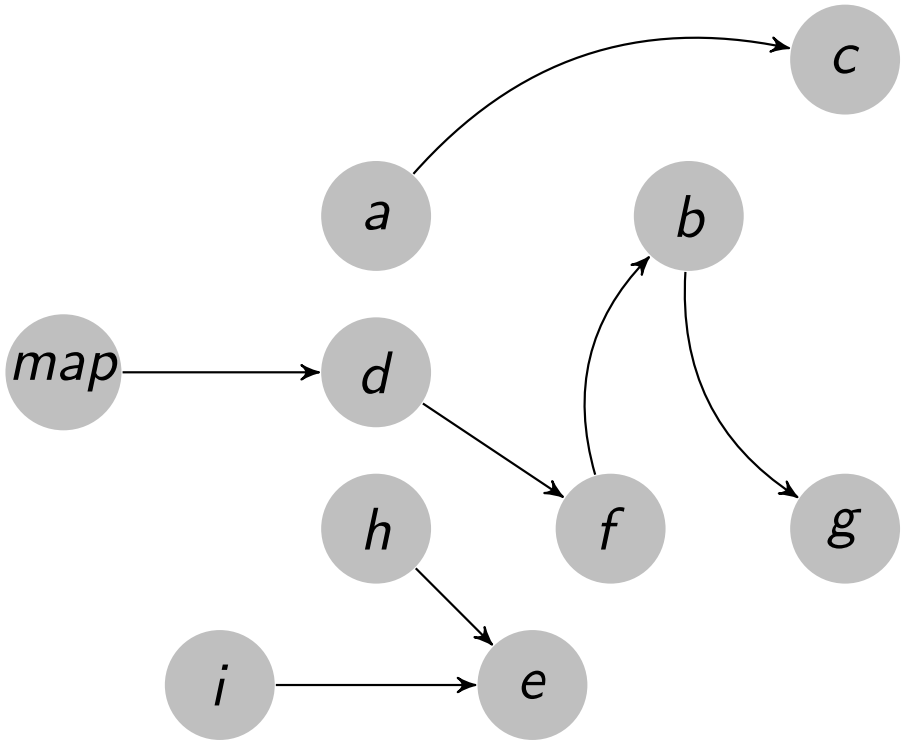


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: d

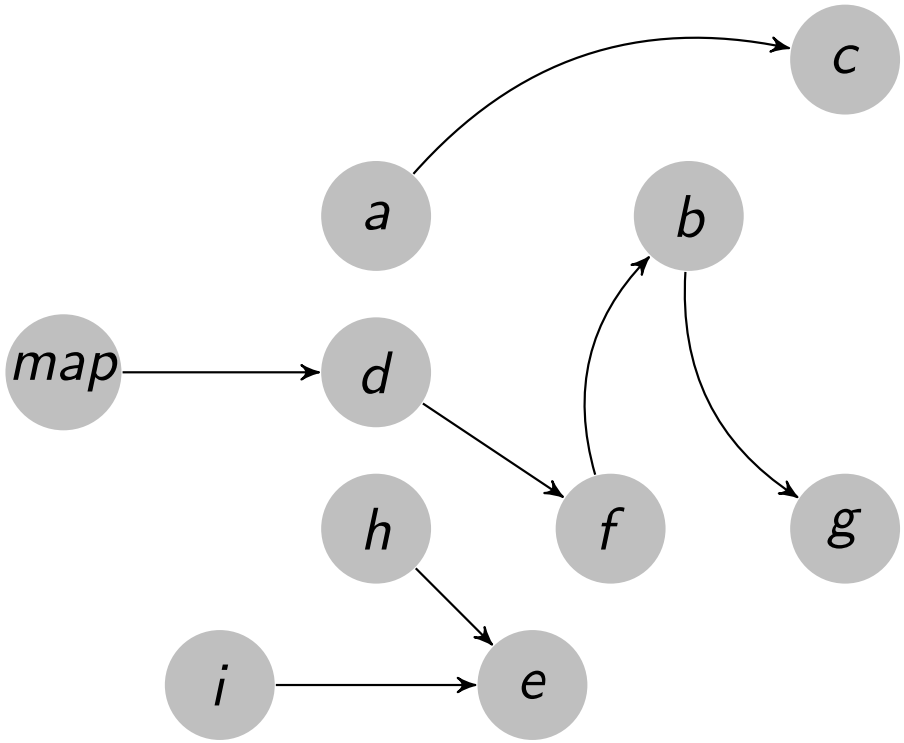


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: d

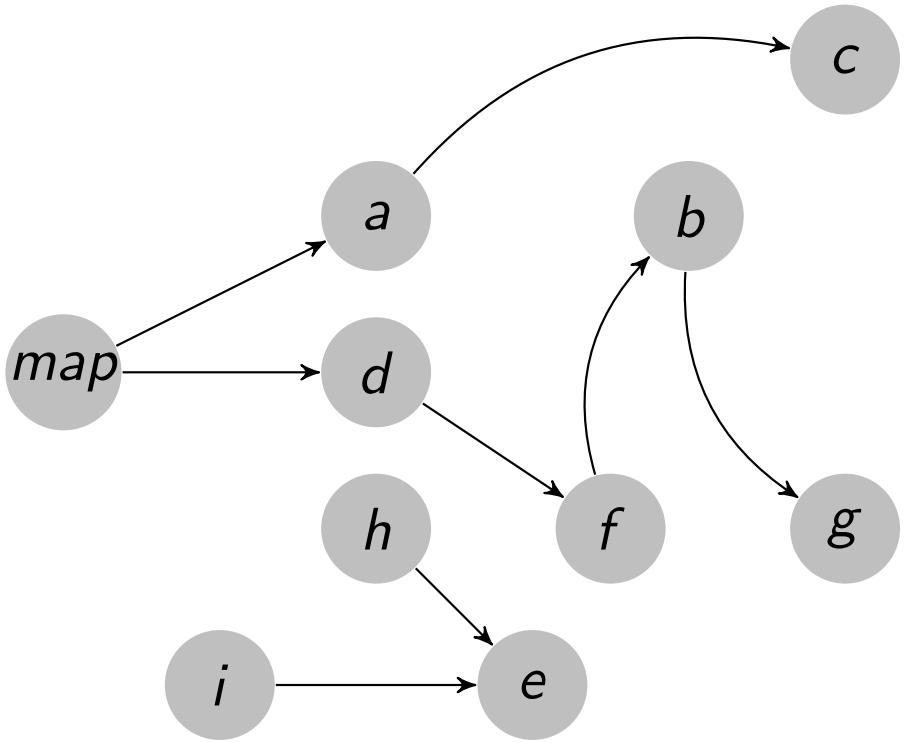


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: | a | d |

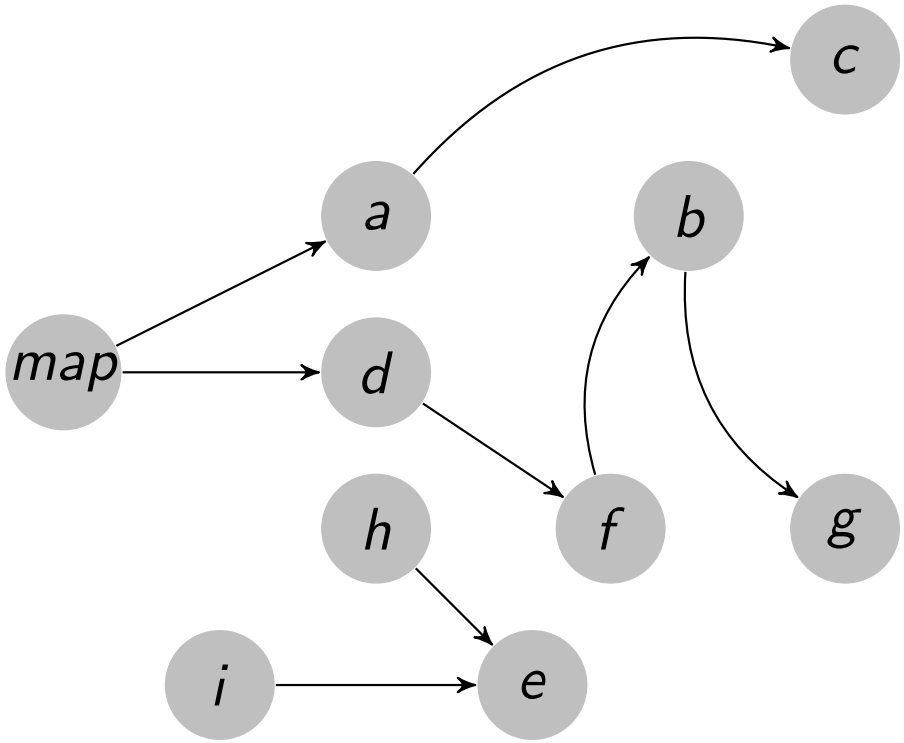


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: | a | d |

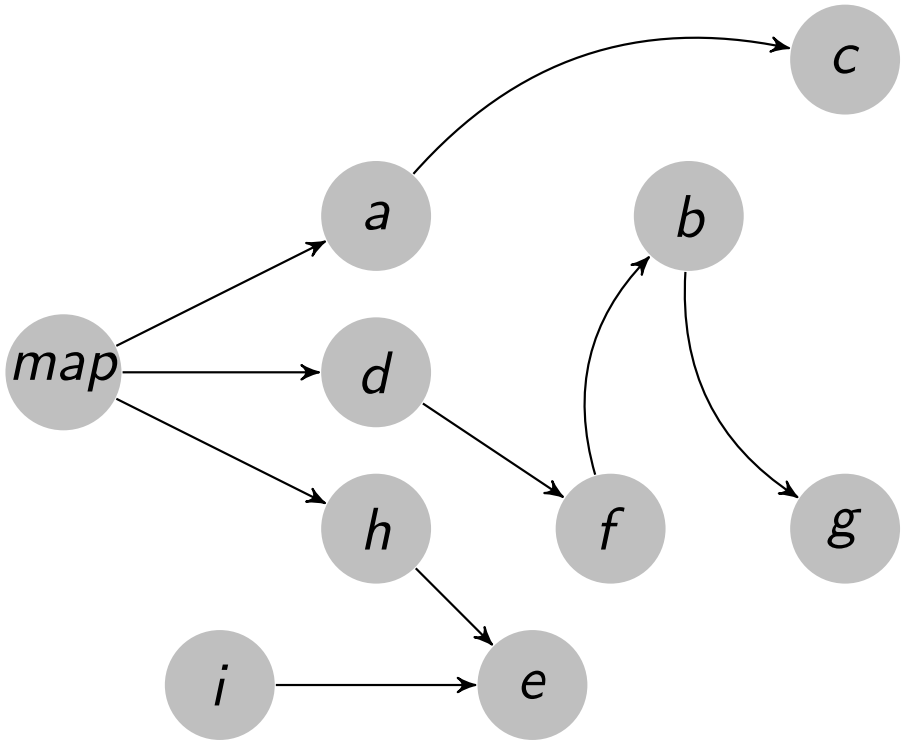


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: | h | a | d |

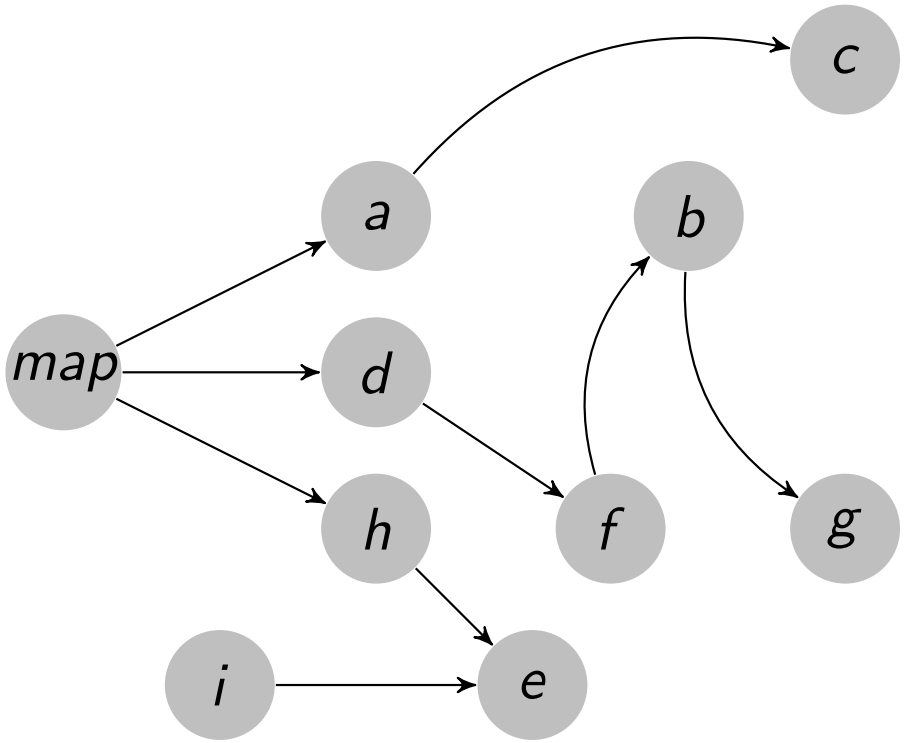


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: | h | a | d |

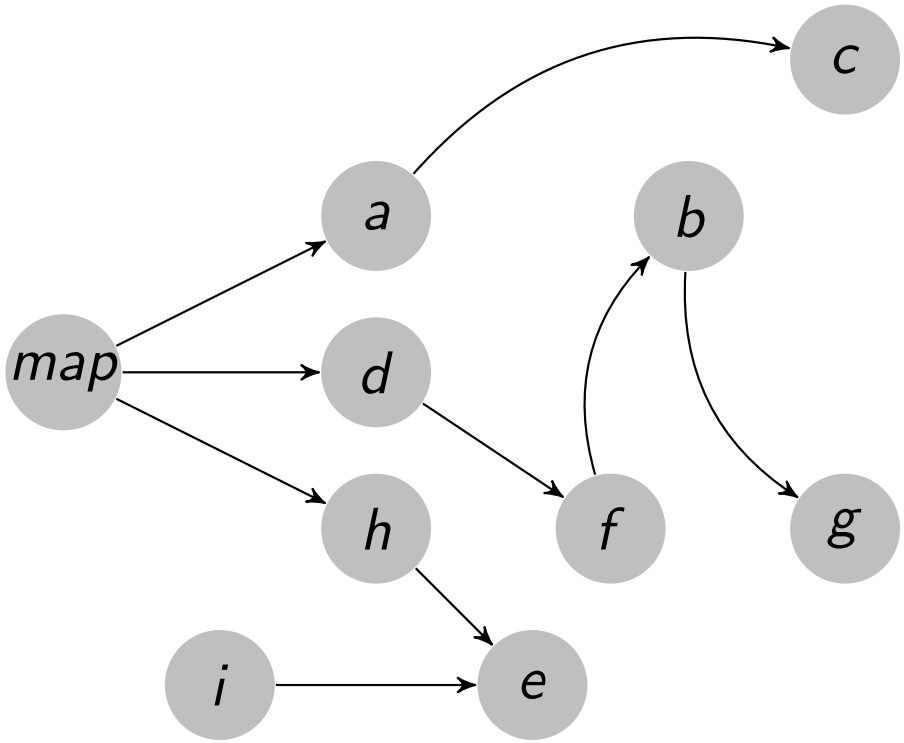


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

space: | d | h | a |

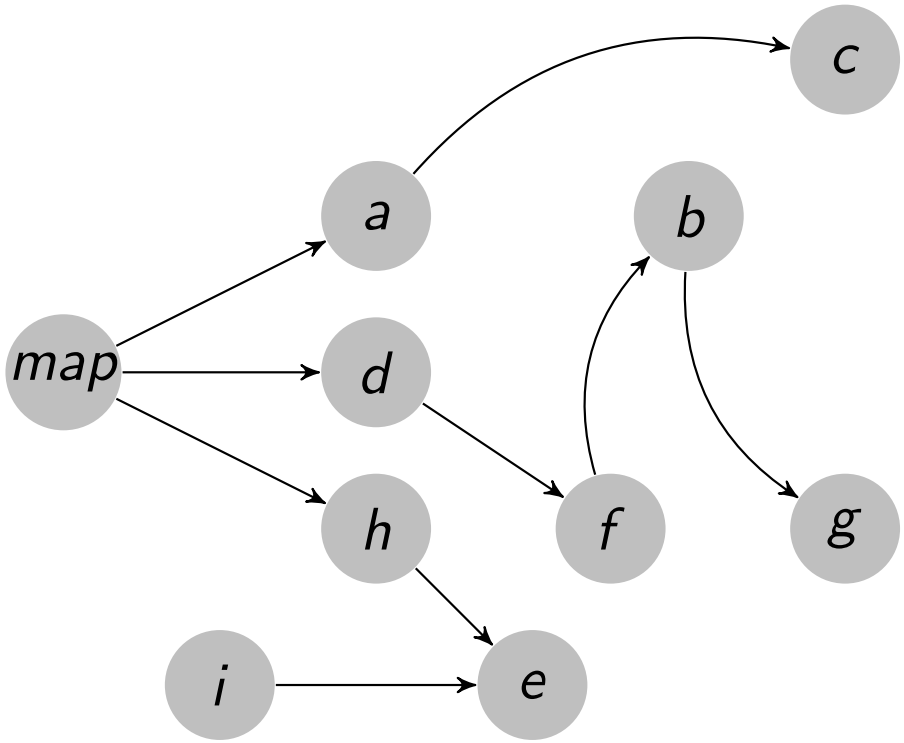


Sache: Using Sache

```
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes

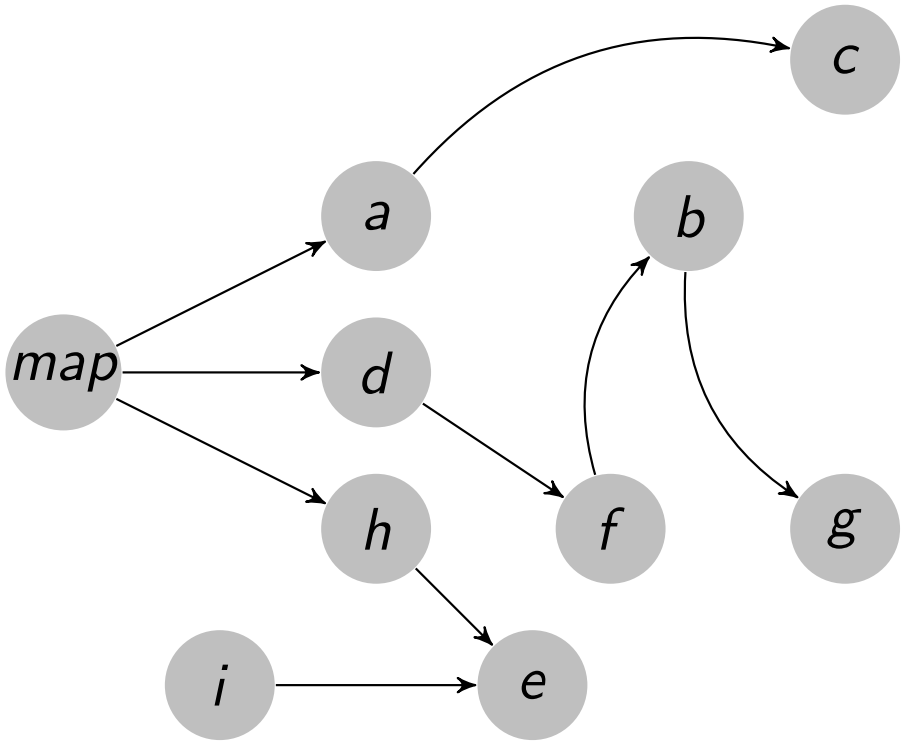
space: | d | h | a |



Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

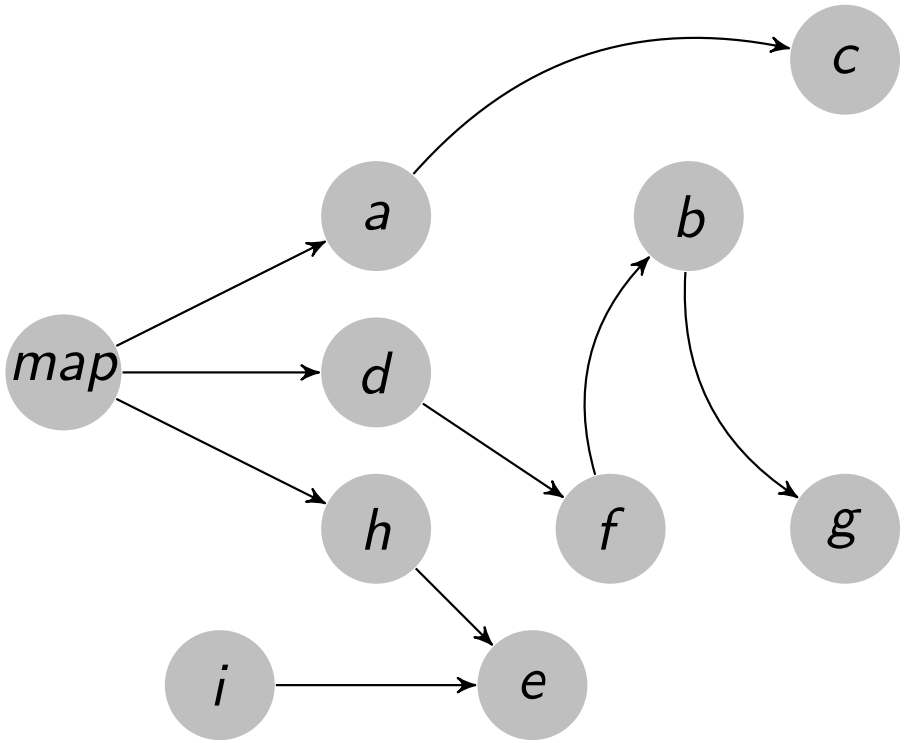

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```



Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

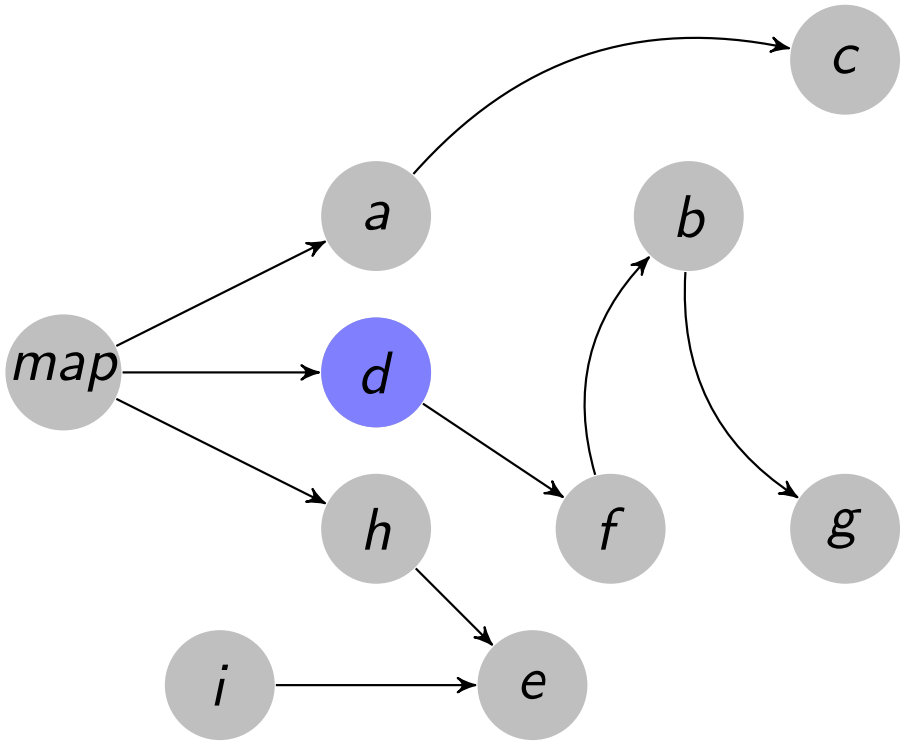
```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```



Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```

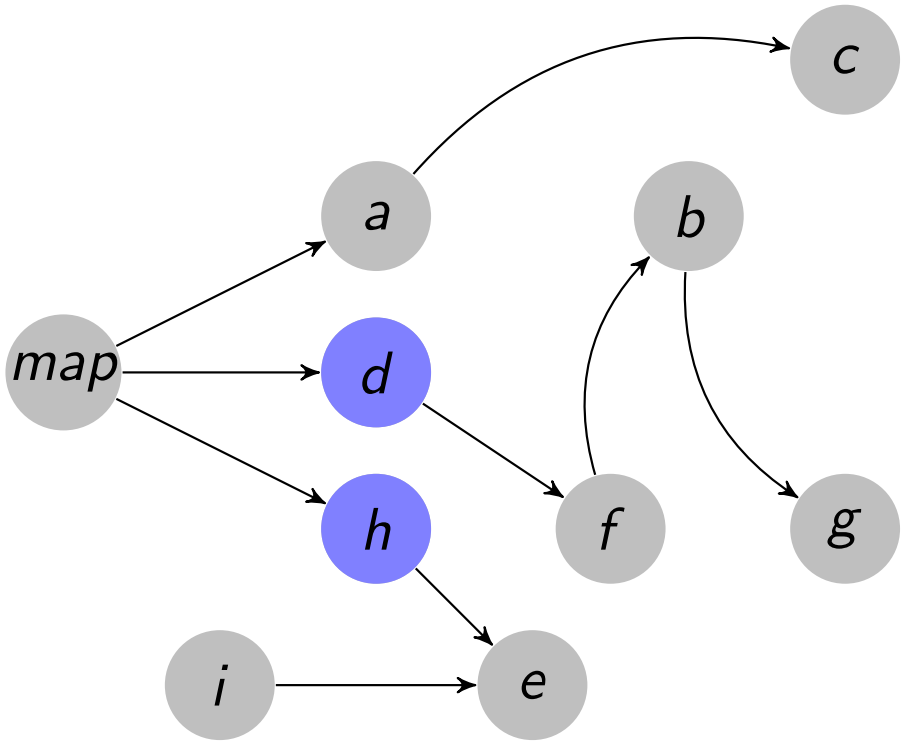


Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects



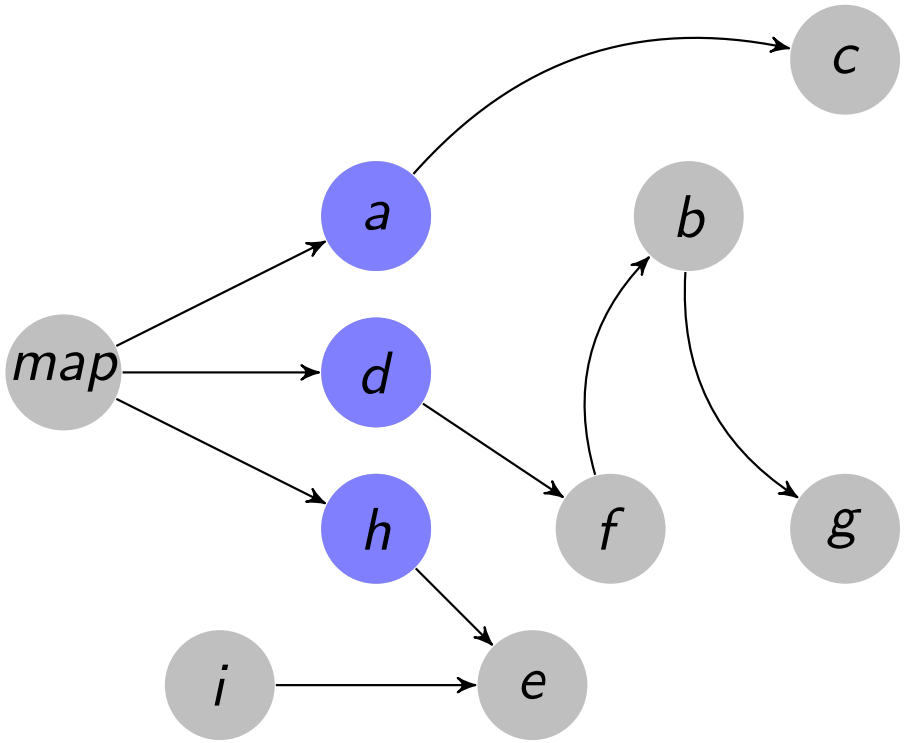
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```

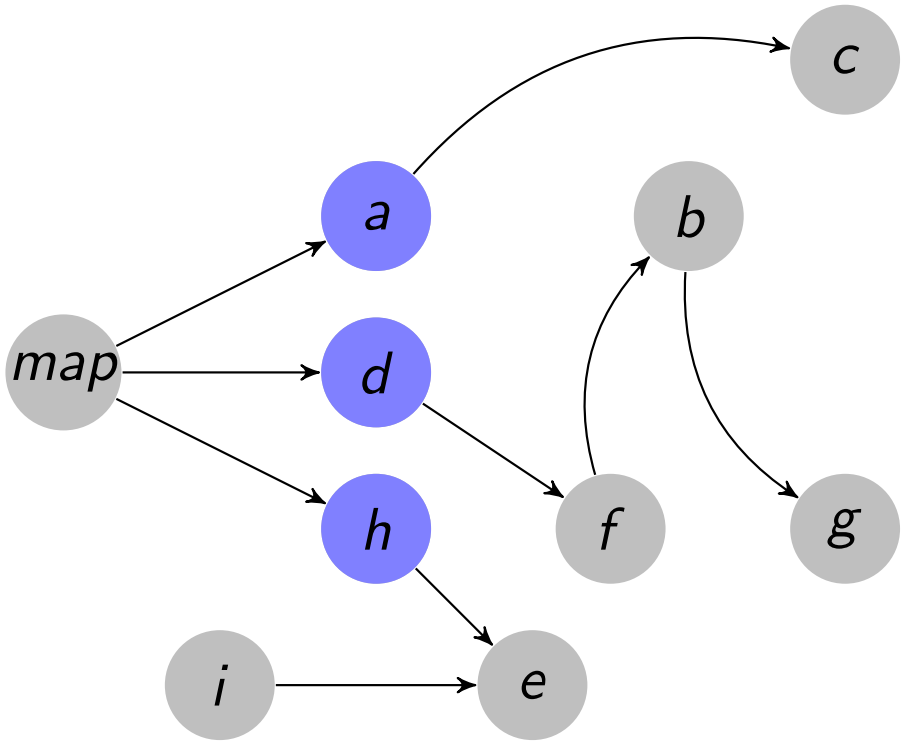


Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

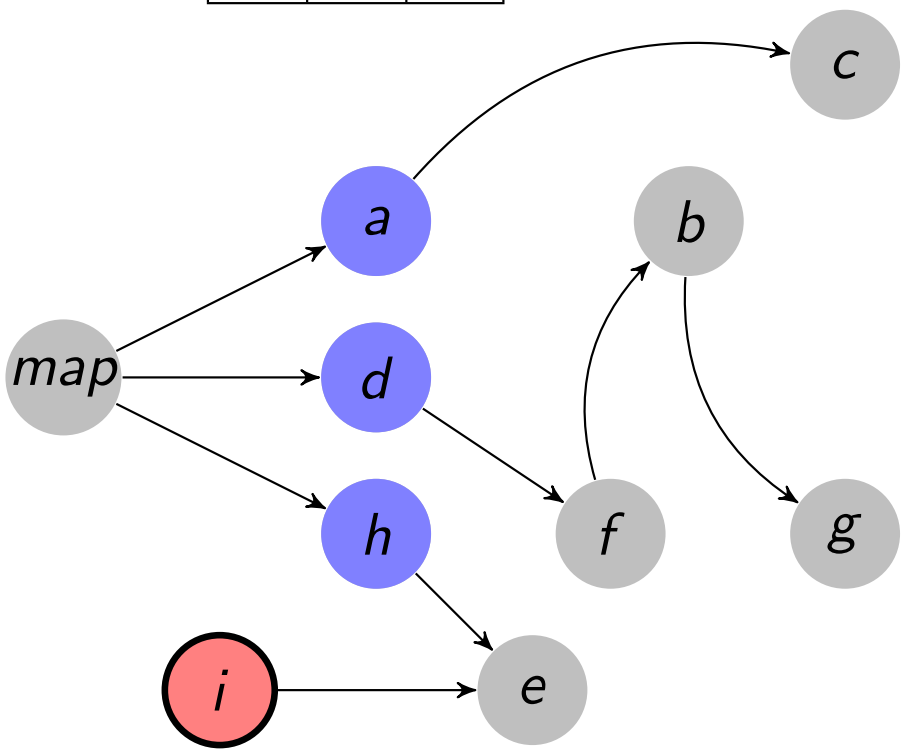


Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

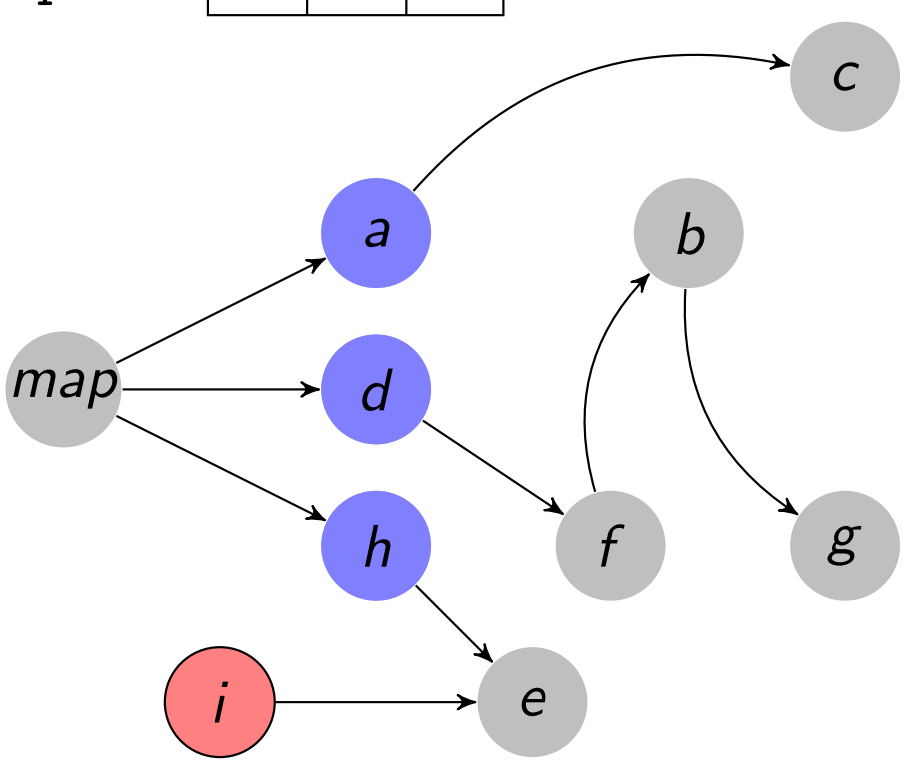


Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects



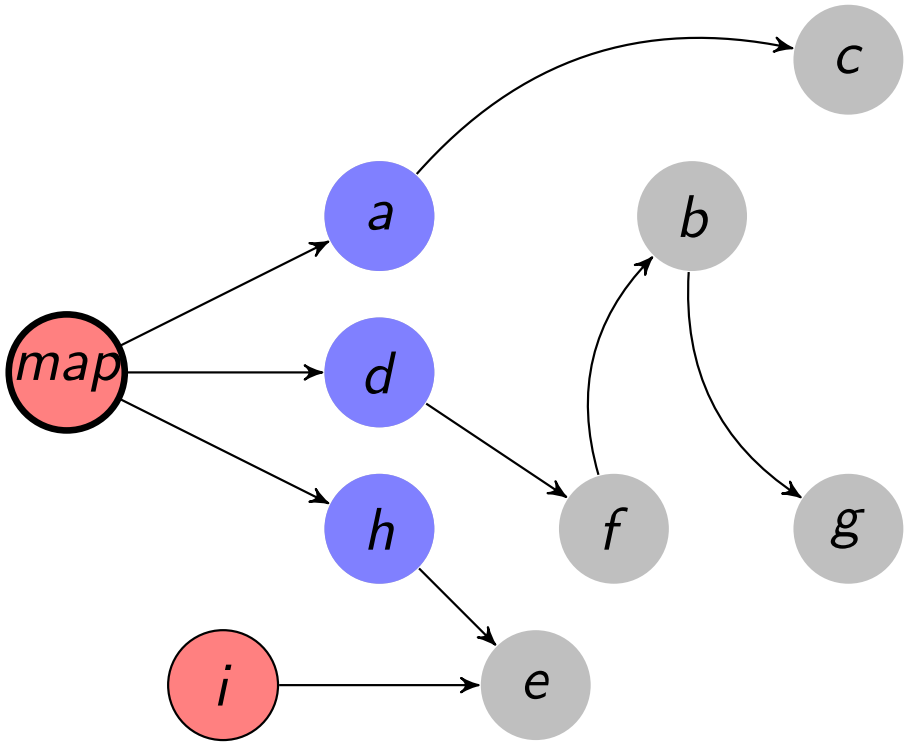
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```



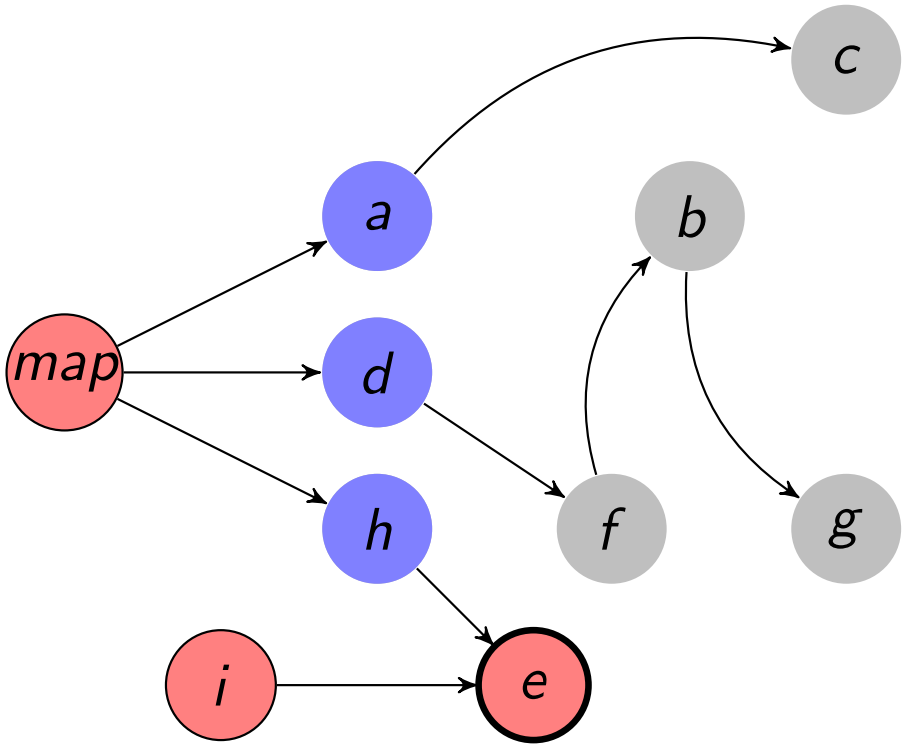
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```



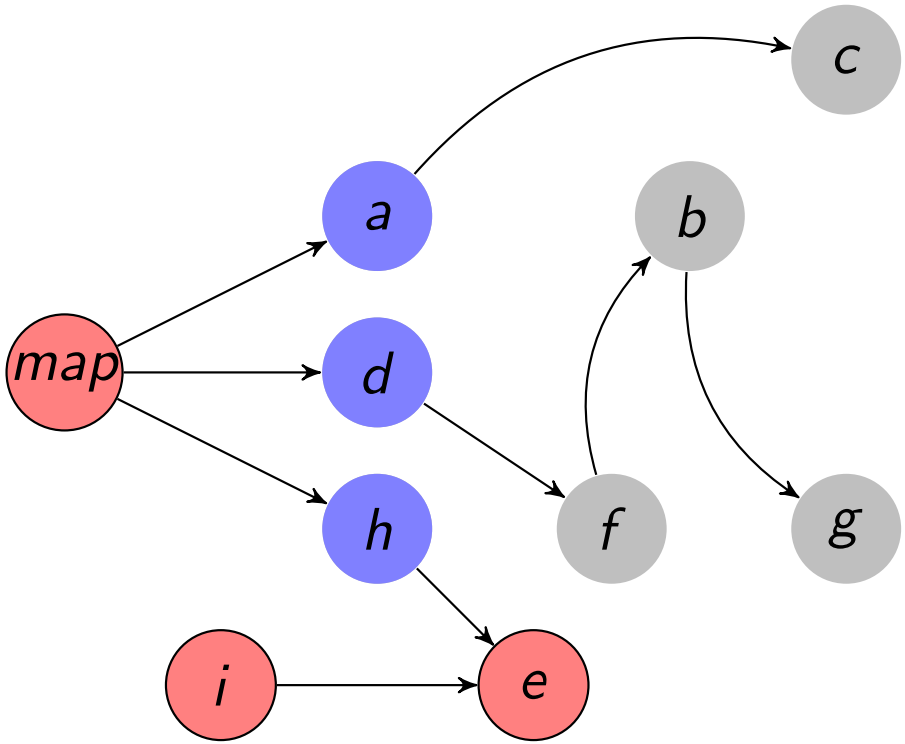
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 0 bytes
space: | d | h | a |
```



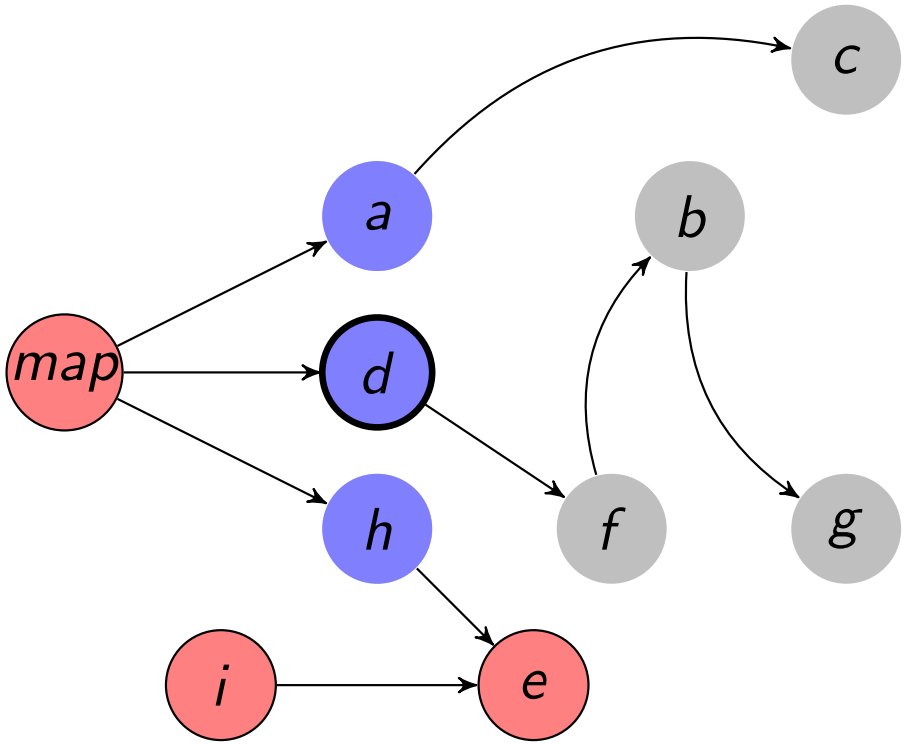
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 12 bytes
space: | d | h | a |
```



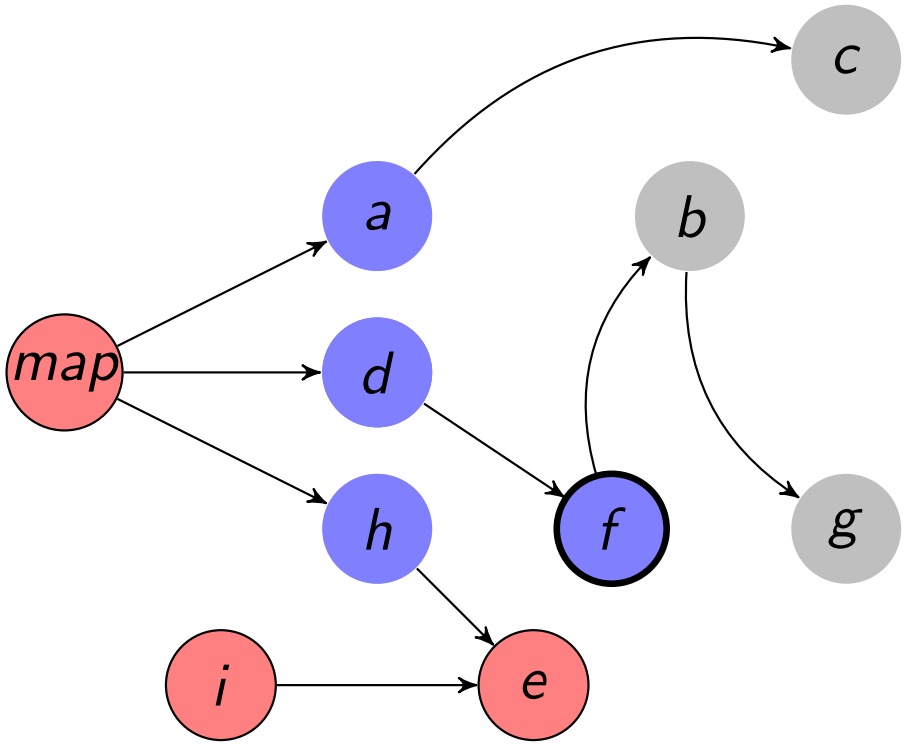
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 24 bytes
space: | d | h | a |
```



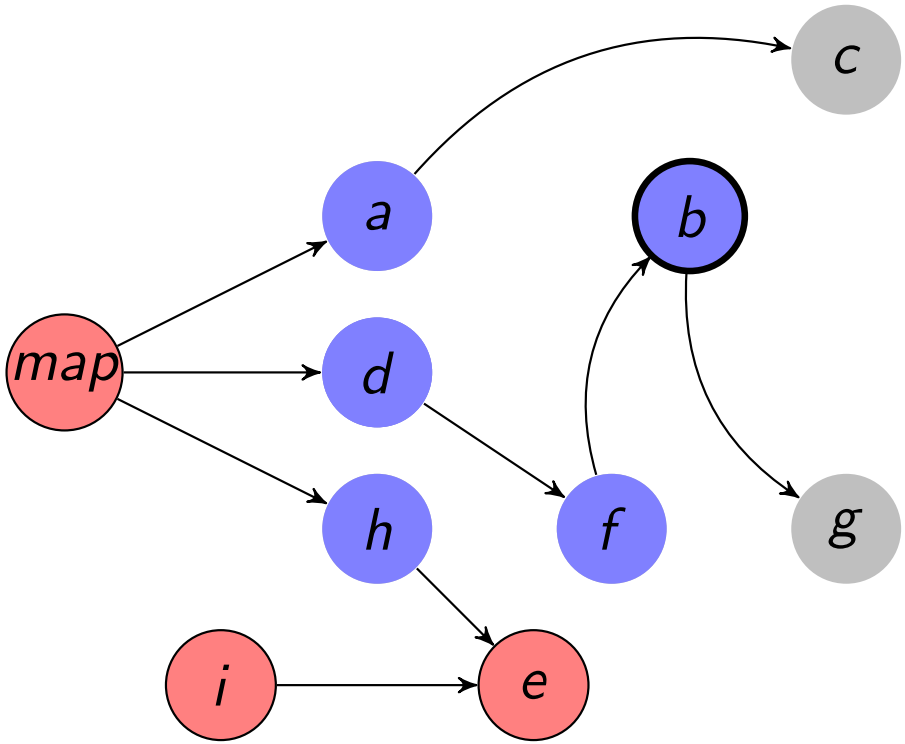
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 36 bytes
space: | d | h | a |
```



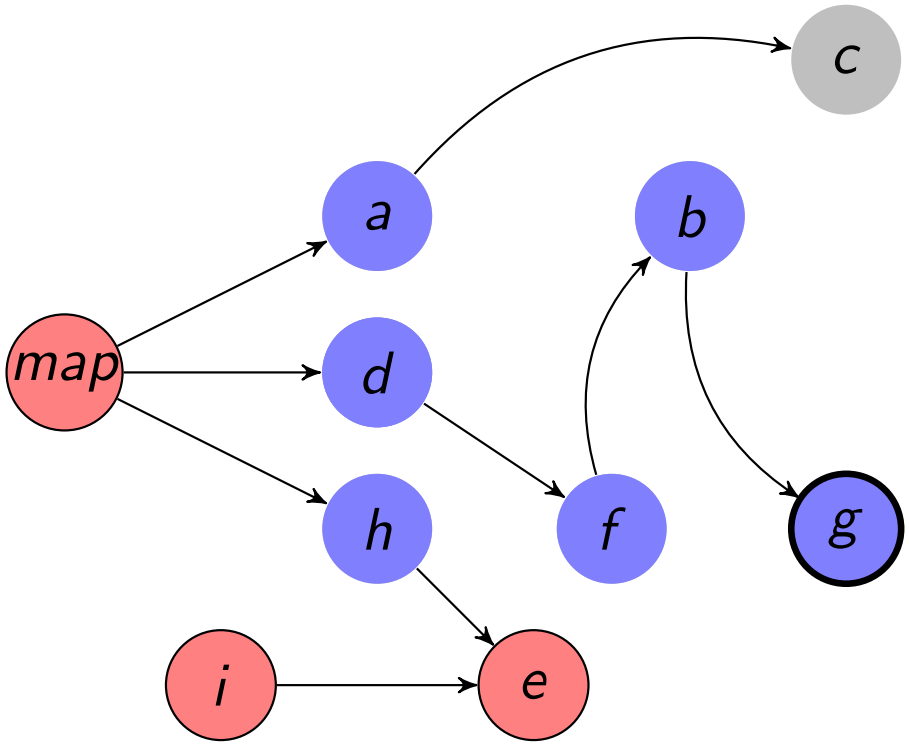
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 48 bytes
space: | d | h | a |
```



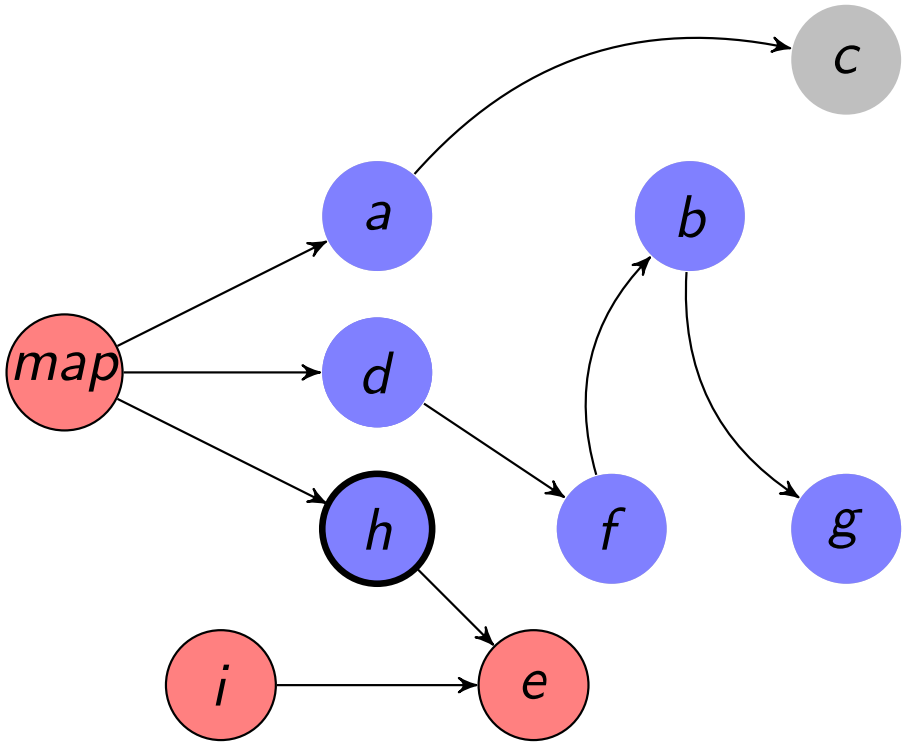
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 60 bytes
space: | d | h | a |
```



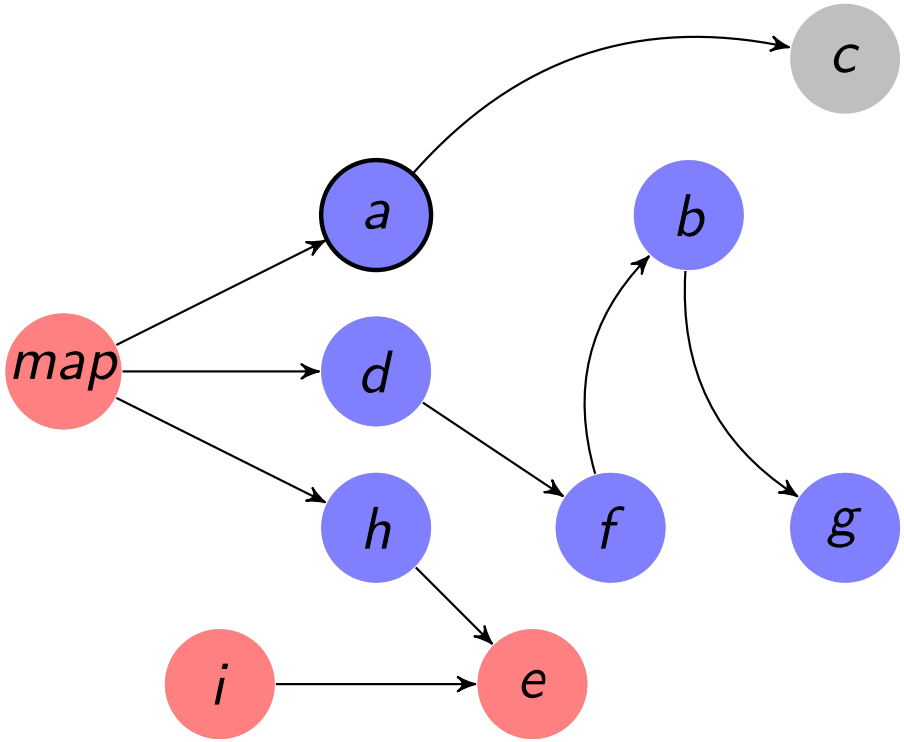
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 72 bytes
space: | d | h | a |
```



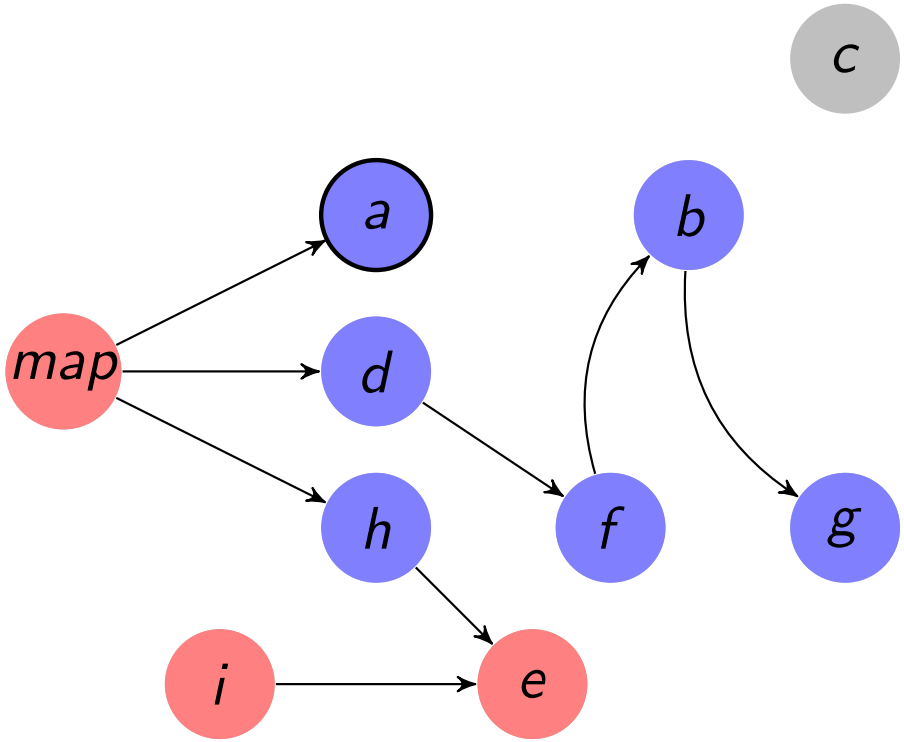
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 60 bytes
space: | d | h | a |
```



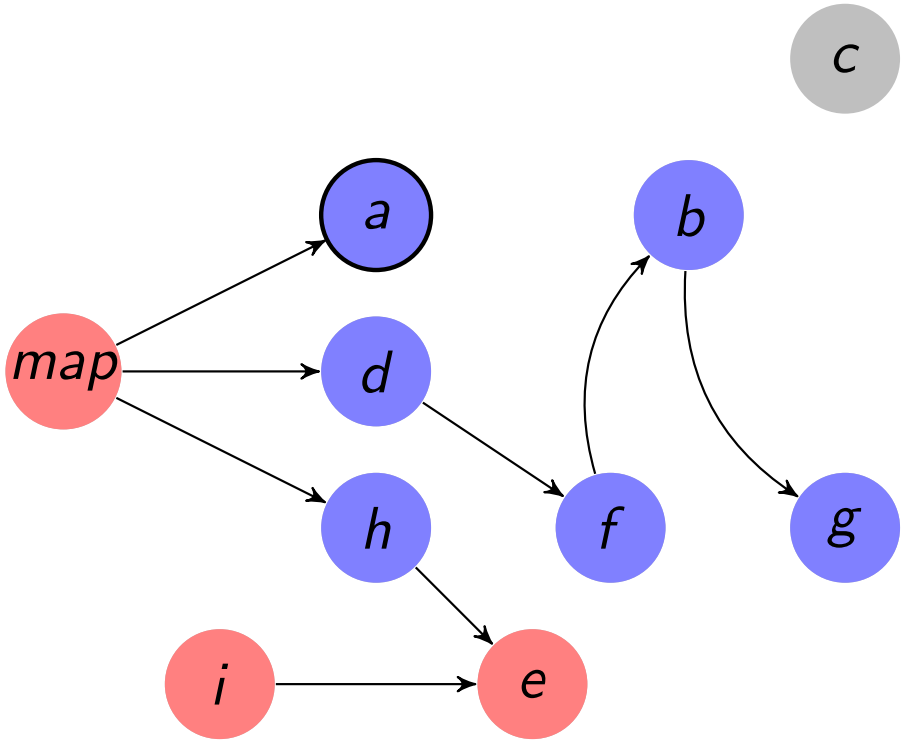
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 60 bytes
space: | d | h |
```



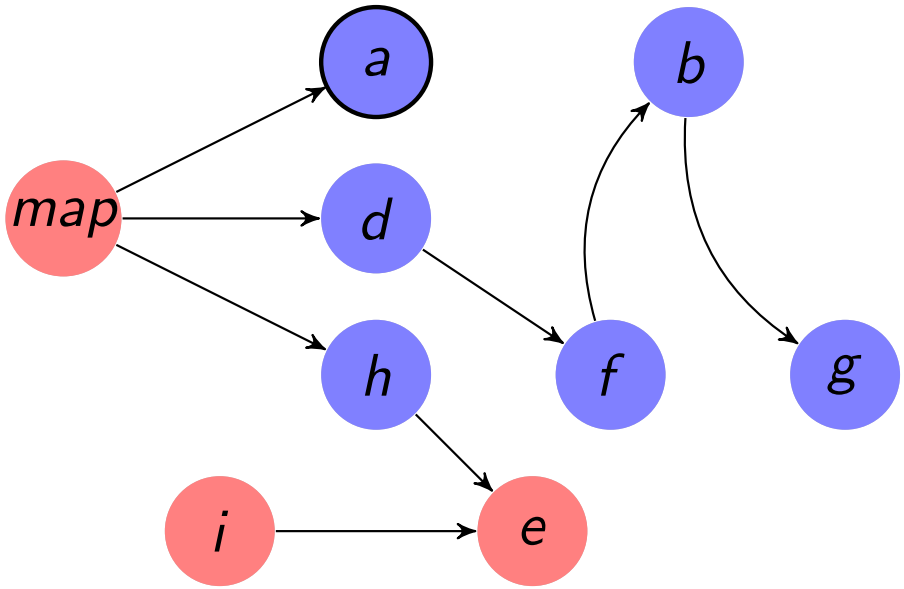
Marked by Mark Sweep

Marked by Prioritized GC

Prioritized GC: Sweep

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 60 bytes
space: | d | h |
```



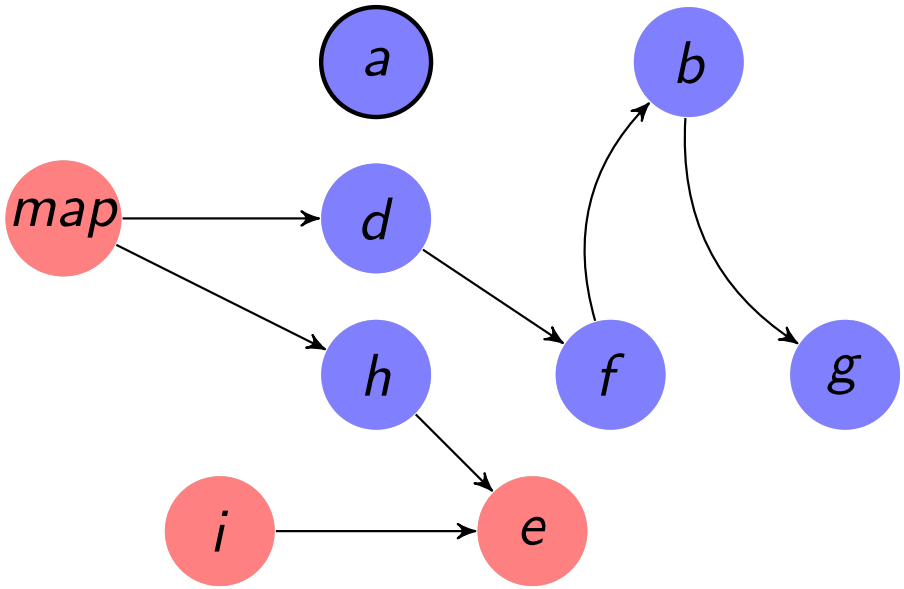
Marked by Mark Sweep

Marked by Prioritized GC

Sache evicts upon next access

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

```
limit = 60 bytes
current-total = 60 bytes
space: | d | h |
```



Marked by Mark Sweep

Marked by Prioritized GC

Experiments: Traces Used

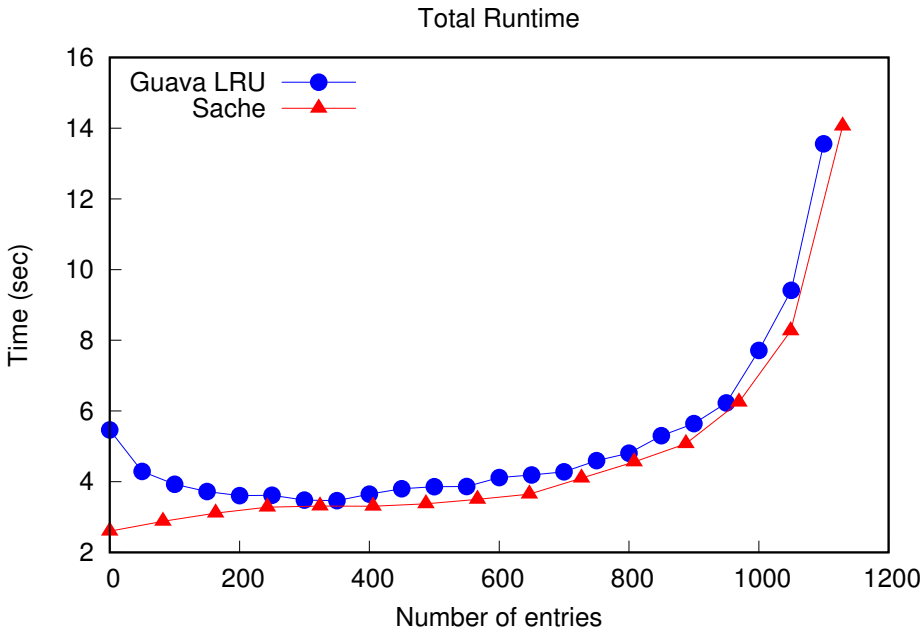
- Traces are list of (string, int) pairs
- string: unique string identifier
- int: size of structure to create
  - drawn from a predetermined range
- Request sequence follows Pareto distribution[1]

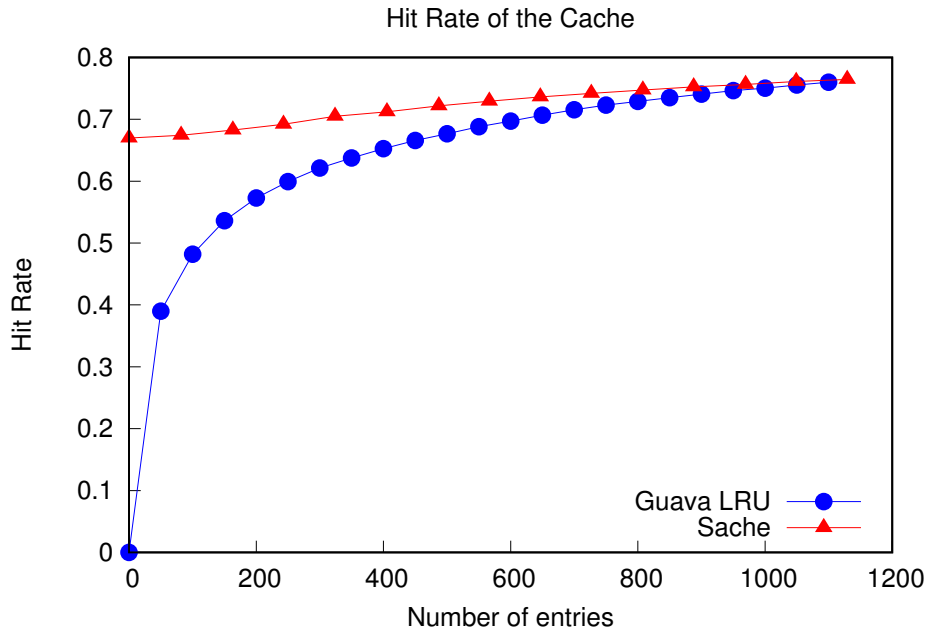
[1] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. Technical report, Boston, MA, USA, 1995.

Experiments

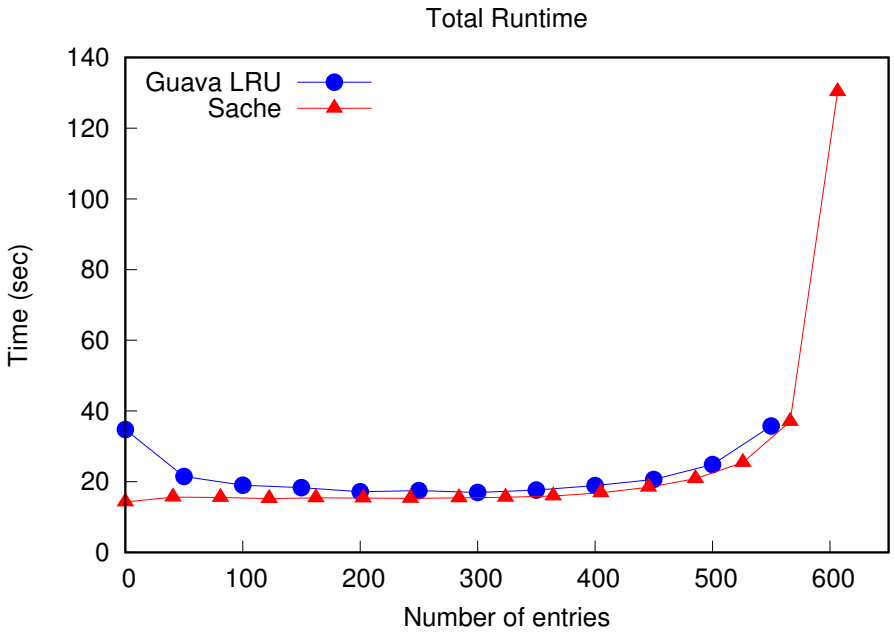
- Trace driven
  - Program reads requests: (string, int) pairs
  - Program looks the string up in the cache
  - If the read fails, create and store a tree of the requested size in the cache
- Change the cache used
  - Google Guava with LRU policy at fixed sizes
  - Sache at fixed sizes
- Implemented Prioritized GC in Jikes Research Virtual Machine

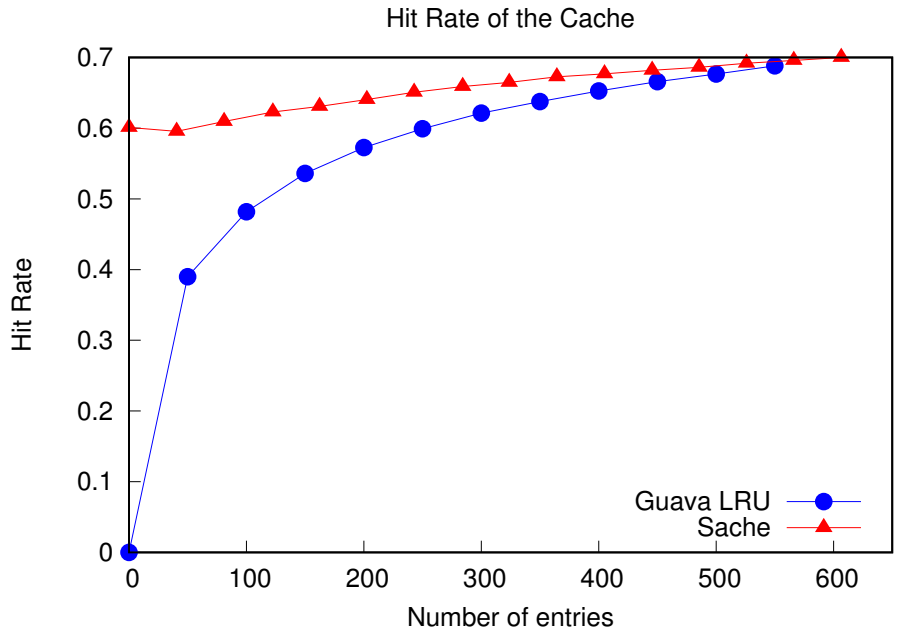
Results: Workloads (50KB – 100KB)





Results: Workloads (100KB – 200KB)





Our Solution: Sache (**S**pace **A**ware Ca**che**)

- User puts a limit on size of Sache in bytes
- Tell the garbage collector what values are in the cache
- Measure the size of these values during garbage collection
- If keeping a value means the Sache exceeds its size limit, evict that value

Our Solution: Adaptive Sache (**S**pace **A**ware Ca**che**)

- ~~User puts a limit on size of Sache in bytes~~
- Tell the garbage collector what values are in the cache
- Each collection, calculate limit on size of Sache in bytes
- Measure the size of these values during garbage collection
- If keeping a value means the Sache exceeds its size limit, evict that value

Adaptive Sache: Calculating Size Limits

Adaptive Sache: Calculating Size Limits

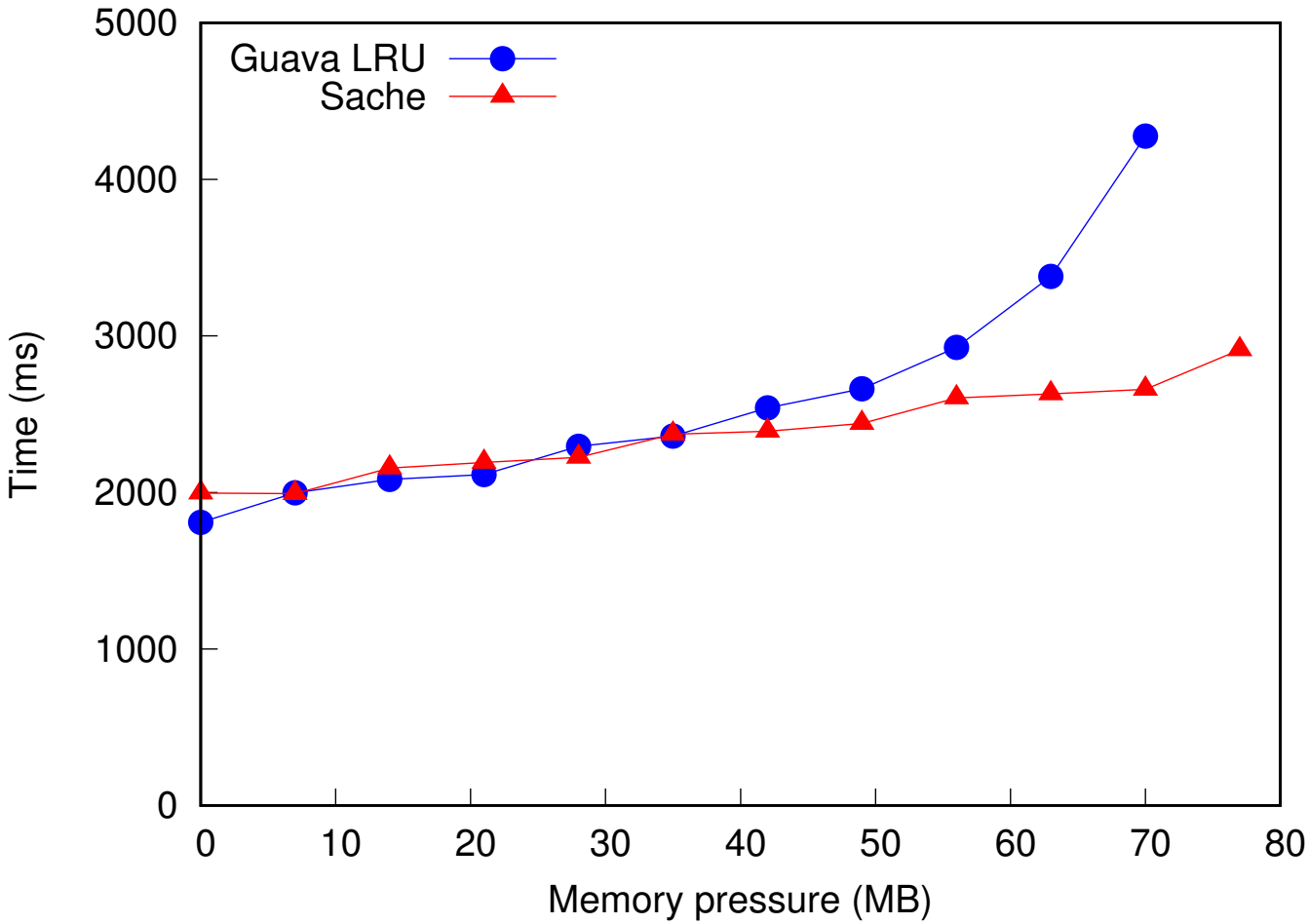
Adaptive Sache: Calculating Size Limits

Adaptive Sache: Calculating Size Limits

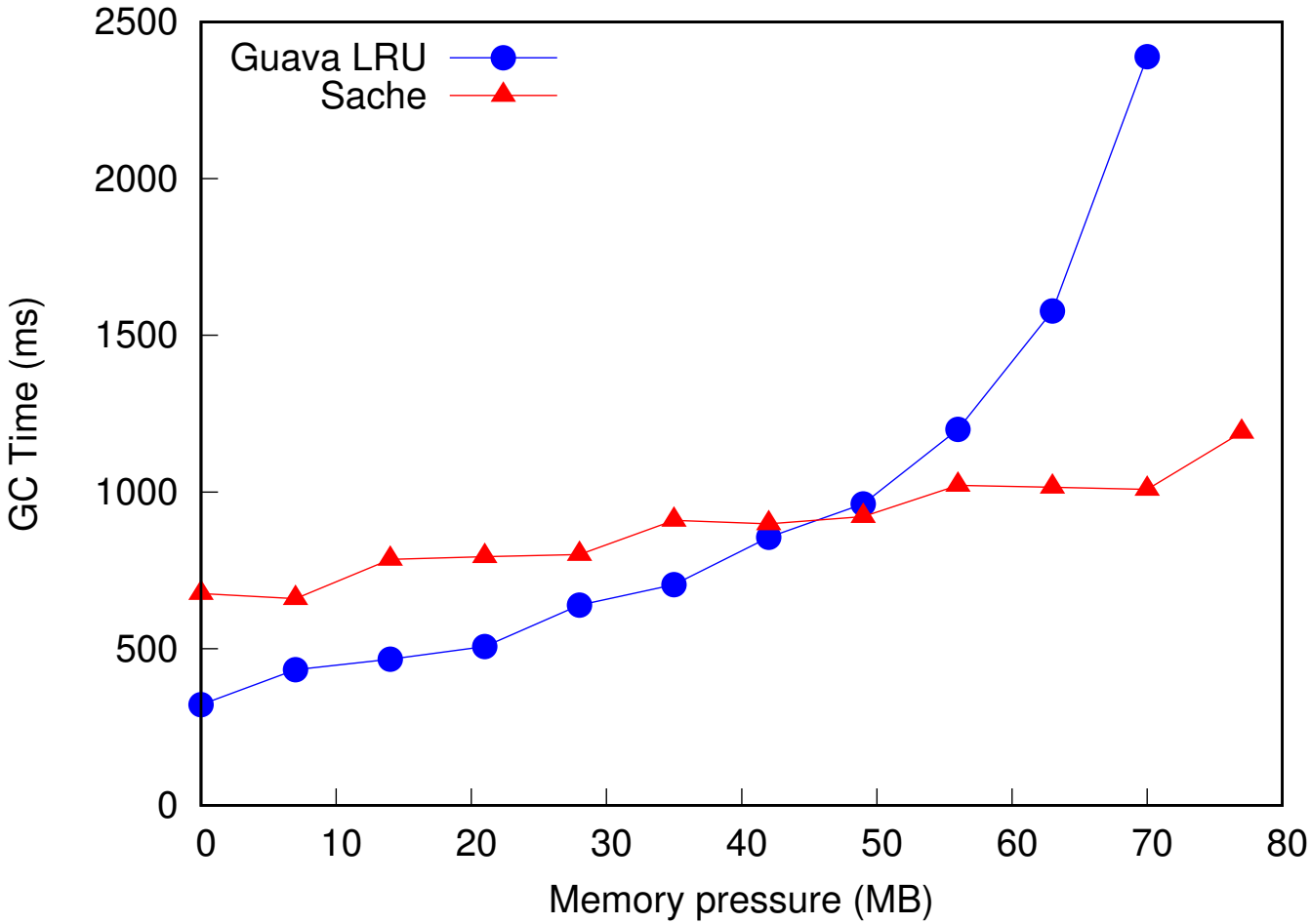
Experiments: Memory Pressure

- Trace driven
  - Use the workload with sizes in 50KB – 100KB range
  - After a third of requests read, increase memory pressure
  - After another third, decrease memory pressure
- Change the cache used
  - Google Guava with LRU policy fixed at 350 entries
  - Adaptive Sache with reserve of 50%

Results: Pressured Workload Trace: Total Run Time

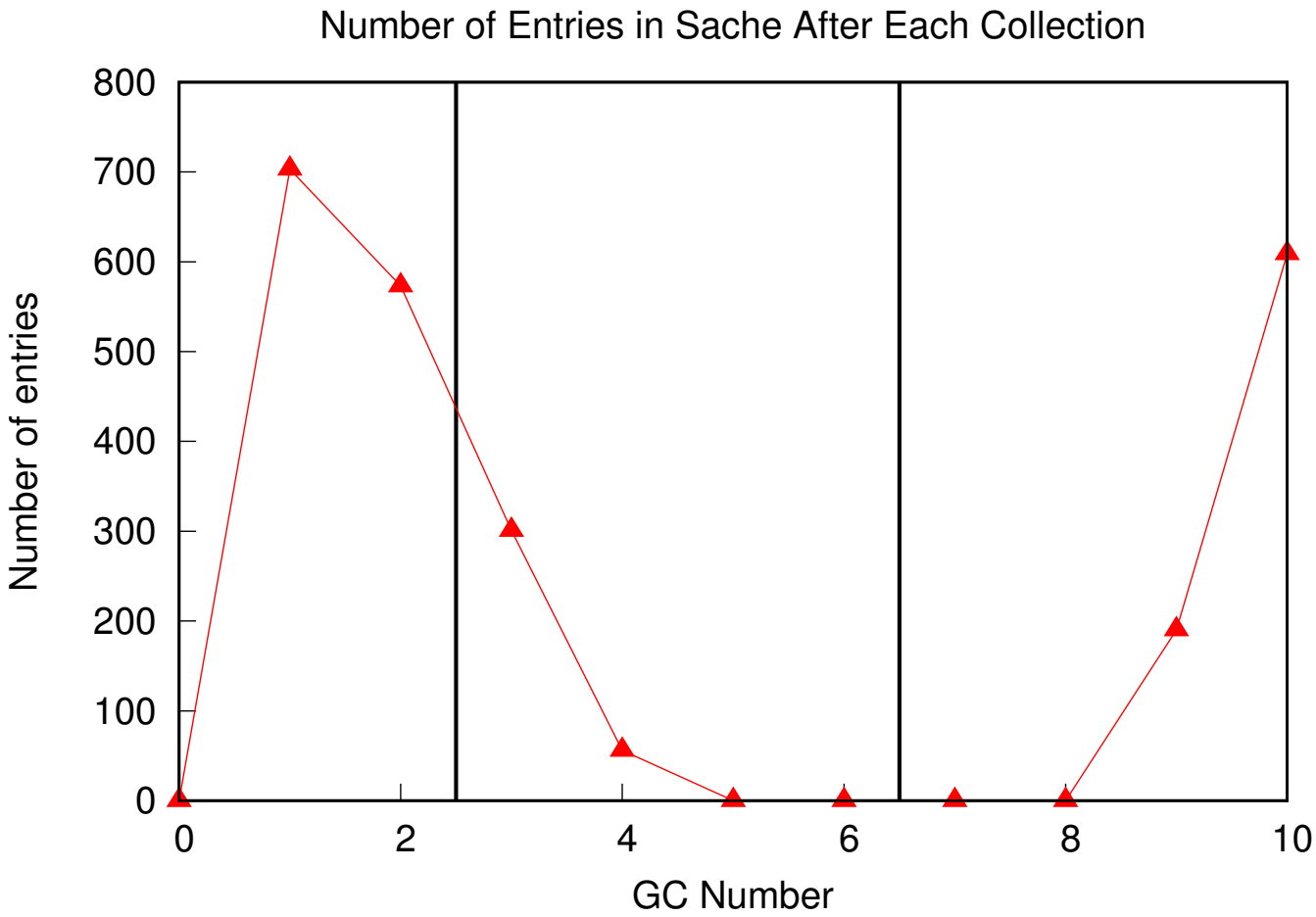


Results: Pressured Workload Trace: Total GC Time



Results: Adaptive Sache on Pressured Workload Trace

Add lines to note when memory pressure starts and ends



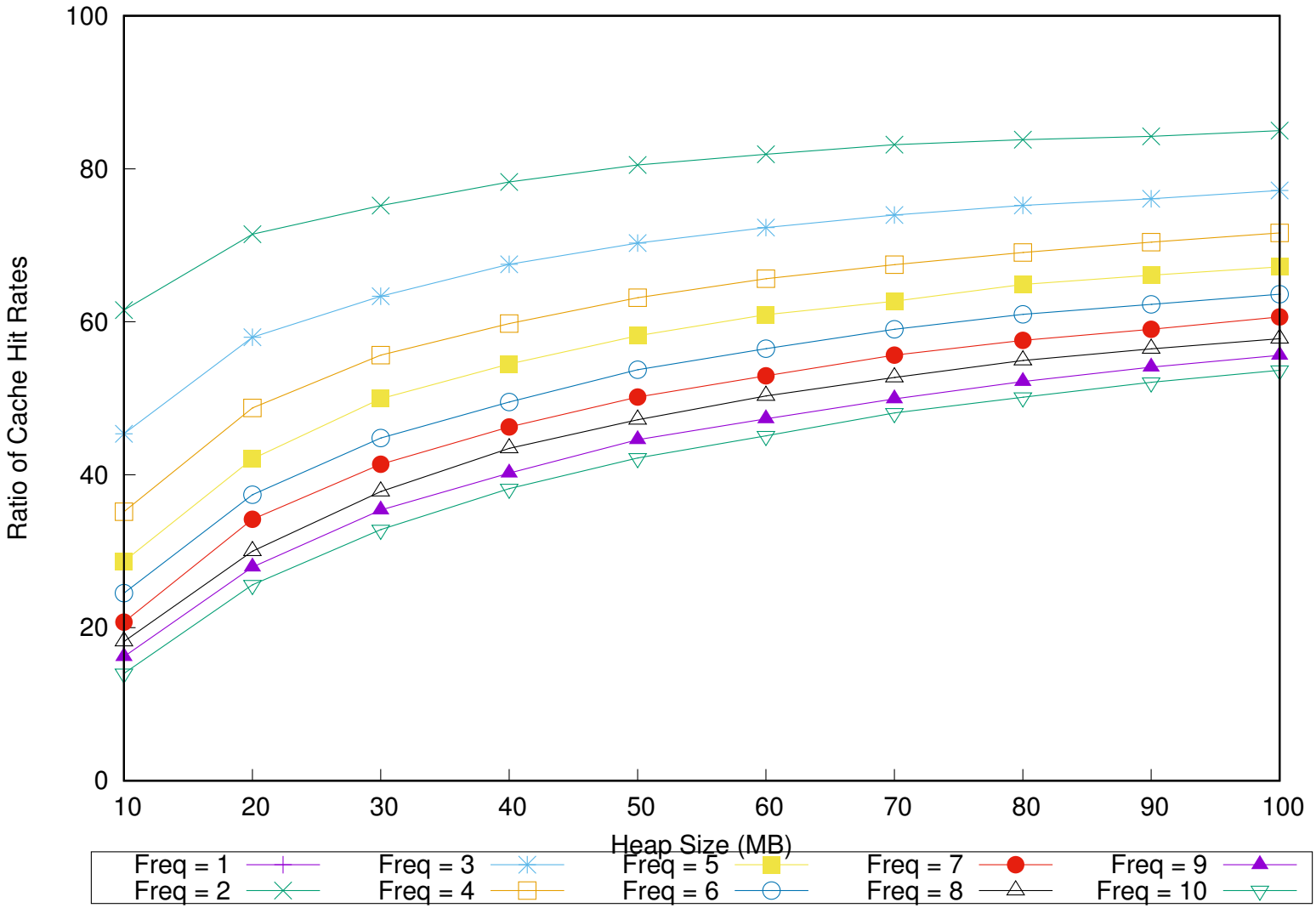
Conclusion

- Prioritized Garbage Collector allows programmers to handle references in application-specific ways
- Sache uses the Prioritized Garbage Collector to enforce memory limits
- Sache can adapt its size to memory pressure, improving robustness

Questions?

Backups

Saches and SoftReference Caches: Hotspot



Saches and SoftReference Caches: Sache