Hard Real-Time Garbage Collection in Java Virtual Machines

... towards unrestricted real-time programming in Java

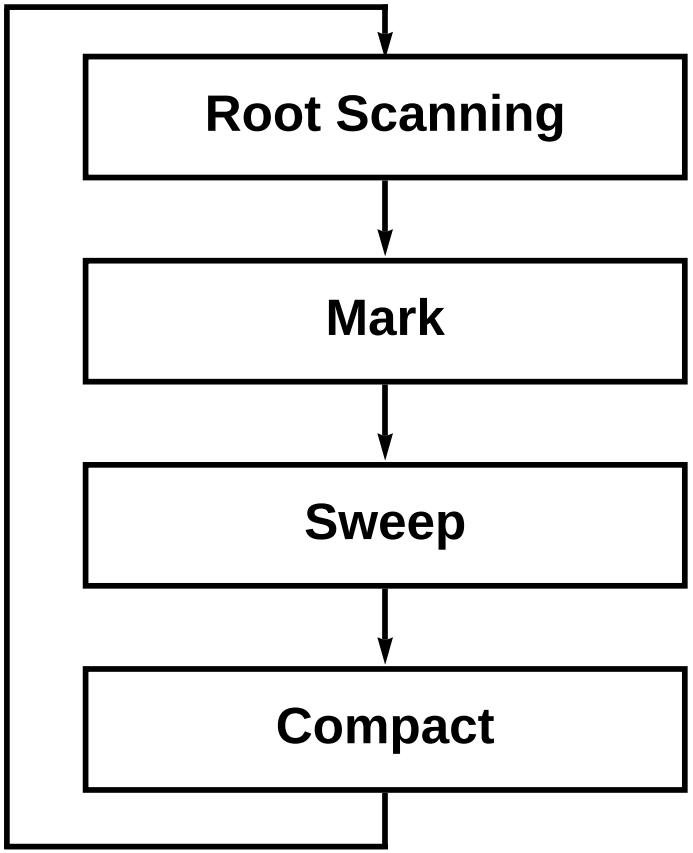
Fridtjof Siebert, IPD, University of Karlsruhe

Structure

- Exisiting GC Techniques
- Definition of Real-Time Garbage Collection
- Jamaica's Garbage Collector
- Example
- Conclusion

Garbage Collection

Automatic reclamation of unused memory



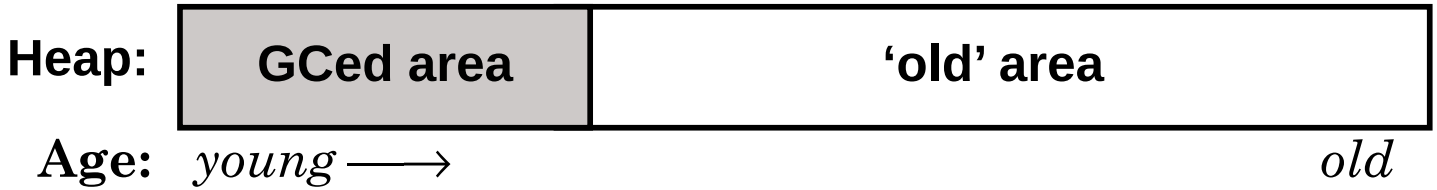
Why is Real-Time GC so difficult in Java?

- Allocation is an integral part of Java programming
- Java is Multi-Threaded
- Java Objects and Arrays may be of very different sizes

Existing Implementations

➔ **Blocking**

➔ **Generational, Age Based**



➔ **Concurrent**

GC as separate thread

➔ **'Real-Time'-Threads**

Real-Time code uses subset of Java langage

Definition of Real-Time GC

Real-Time Garbage Collection means

- **Hard upper bounds for execution time of any operation.**
- **Any operation must succeed in desired way**
- **Hard upper bound for pre-emption delay**

For the implementation to be useful

- **Upper bounds for execution time must be short (in the order of µsec/nsec)**

Definition of Real-Time GC

Operations that are affected by GC are

- **Memory reads (accesses to objects, arrays)**
  (read-barrier code?)
- **Memory writes**
  (write-barrier code?)
- **Allocation of objects**
  (recycling work!)

Real Time Garbage Collection in the

Root Scanning

Root scanning typically causes the biggest problems in real-time garbage collection:

- **Need to stop threads for a long time!**

In the Jamaica Virtual Machine:

- **All life refs are also stored on the heap**
- **There is only one single root pointer**

⇒ **Root scanning is a real-time operation**

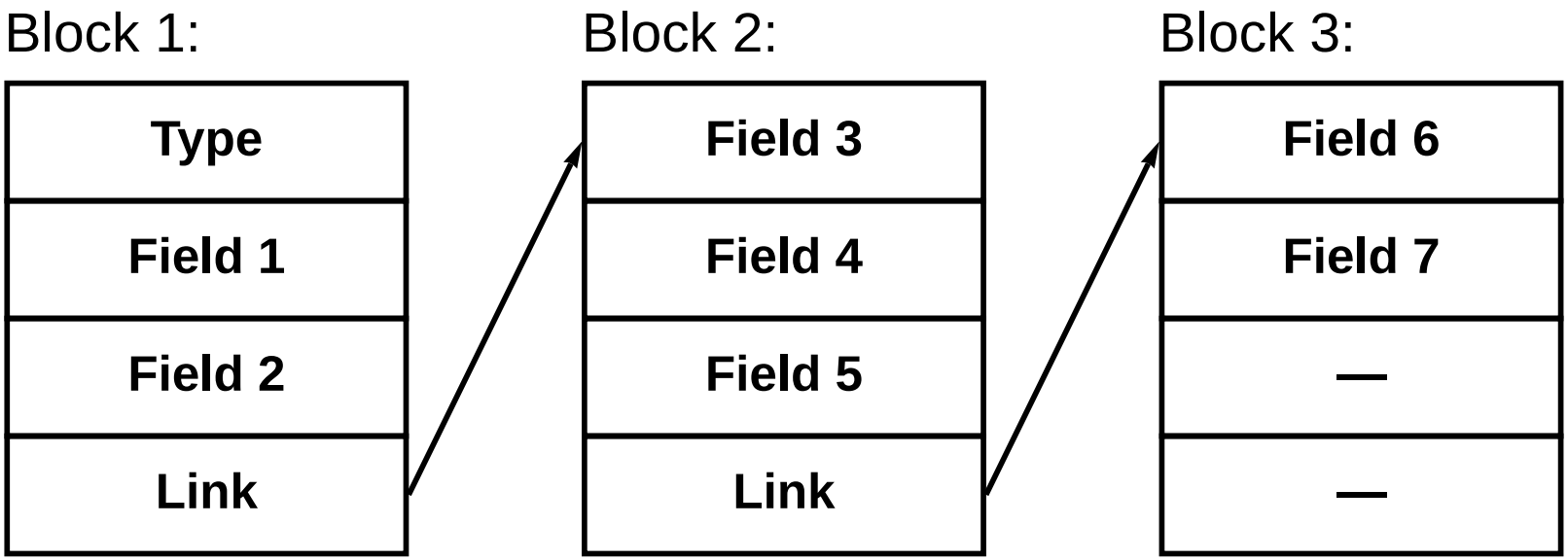
⇒ **But, additional overhead of writing 2-3 refs on every call (using write-barrier!)**

Fragmentation

Avoid fragmentation without using handles and moving objects:

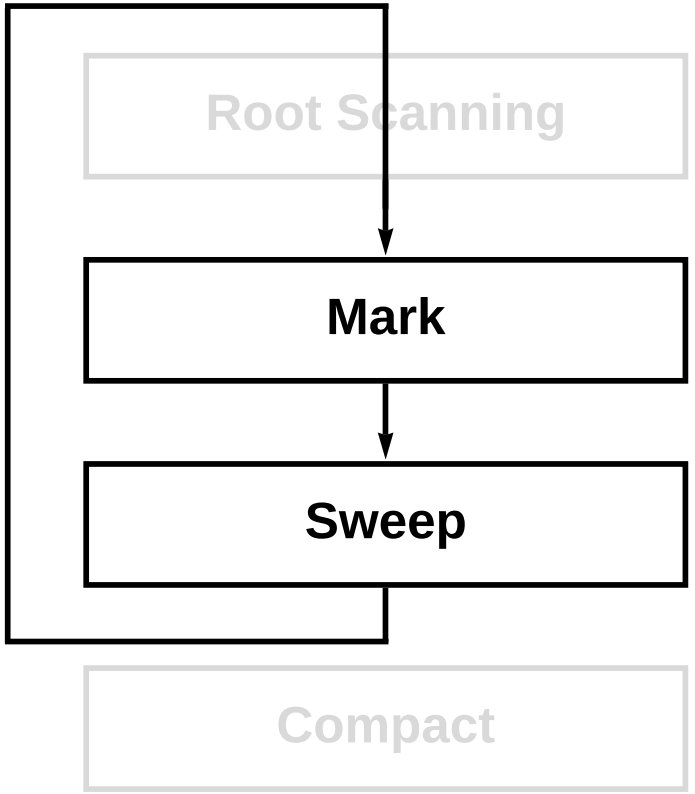
- **Heap is array of blocks of the same size**
- **At least one such block used for every Java object allocated**
- **Larger objects are represented as a graph of these blocks**

Object Layout

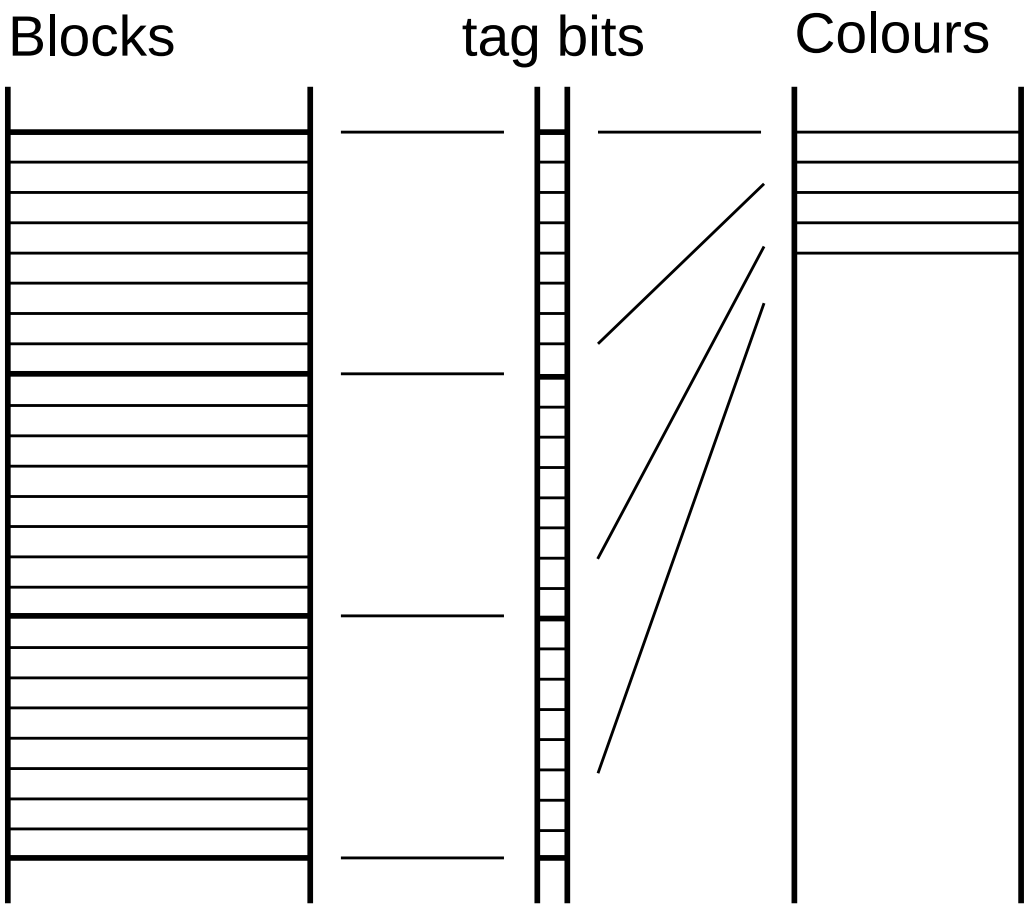


GC Algorithm

- Simple incremental mark & sweep garbage collector
- Does not know about Java objects, just works with blocks
- Tagging is used to identify references
- One word per object reserved for colour



Heap Layout



GC Activation

A GC running as a separate thread with no information on the application might not recycle enough memory or use too much CPU time.

⇒ Couple GC activity with allocation in application threads.

⇒ Amount of GC work is determined dynamically as function of amount of free memory

GC Activiation

With this

⇒ Amount of GC work is adjusted dynamically as needed by application

⇒ No allocation means no GC work at all

⇒ GC can guarantee an upper bound of allocation time for any application with limited memory demand

⇒ Trade off between this upper bound and heap size is possible

Example

```
public class HelloWorld {
  public static void main(String[] args) {
    int n,s,c;
    if (args.length > 0) {
      n = Integer.parseInt(args[0]);
    } else {
      n = 30;
    }
    s = 0;
    c = 14;
    for(int i=0; i<n; i++) {
      String s1 = "                    ".substring(s+14);
      String s2 = "                    ".substring(s/2+7);
      System.out.println(s1+"Hello "+s2+"World!");
      s = s + c / 4;
      c = c - s / 4;
    }
  }
}
```

Example

```
> jamaica -analyse 5 HelloWorld
> HelloWorld
              Hello        World!
           Hello       World!
        Hello     World!
[...]
### Application used at most 165664 bytes for Java heap
### Non-Java heap memory used: 49808 bytes.
###
###   heapSize    worst case allocation overhead:
###   373k                 7
###   319k                10
###   292k                14
###   280k                16
###   265k                21
###   251k                28
###   239k                40
###   219k               138
###   215k               286
>
```

Example

Determination of worst-case execution time of

```
new StringBuffer()
```

Determine number of blocks:

```
> numblocks java.lang.StringBuffer
1
```

Worst-case execution time:

$$wcet = numblocks \cdot max_{gc\_unit} \cdot \mathrm{wcet}_{\mathrm{gc\_unit}}$$

$$wcet_{265k} = 1 \cdot 21 \cdot 2\mu s = 42\mu s$$

$$wcet_{373k} = 1 \cdot \ 7 \cdot 2\mu s \ = 14\mu s$$

Conclusion

- **JamaicaVM allows real-time programming using the full Java language**
- **Having hard real-time guarantees on all parts of the programming language, including allocation, will ease software development for more complex real-time systems.**

3-Colour Marking

Colours:

white: not marked yet
grey: known to be reachable, but not scanned yet
black: known to be reachable and scanned

GC-Invariant during Mark-phase:

There is no reference from a black object to a white object.

No grey objects left ⇒ white objects are free

Colour Encoding

Colour values:

black: -1
white: 0
grey: any other value

Grey objects form a linked listed, the colour value is used to refer to the next element.

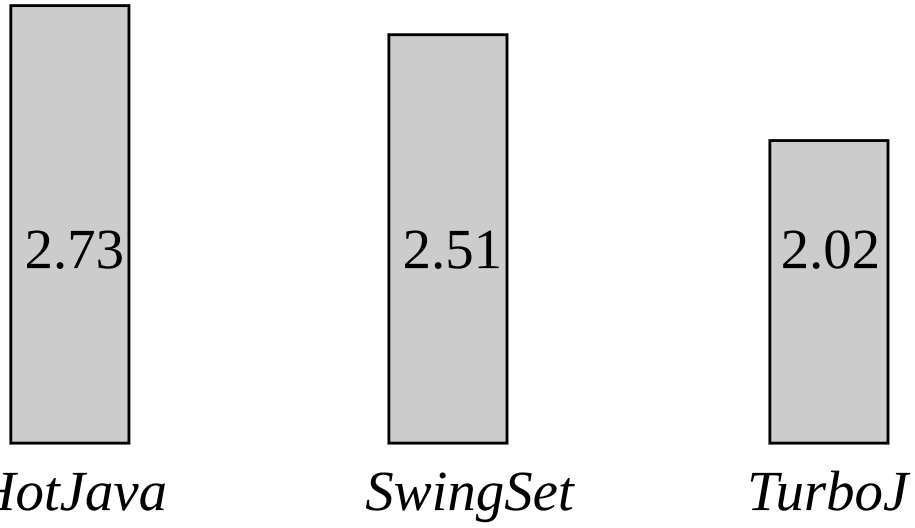
- **constant time addition of grey objects**
- **constant time retrieval of a grey object**

⇒ GC cycle completed in *O(allocated)*

Experimental Data

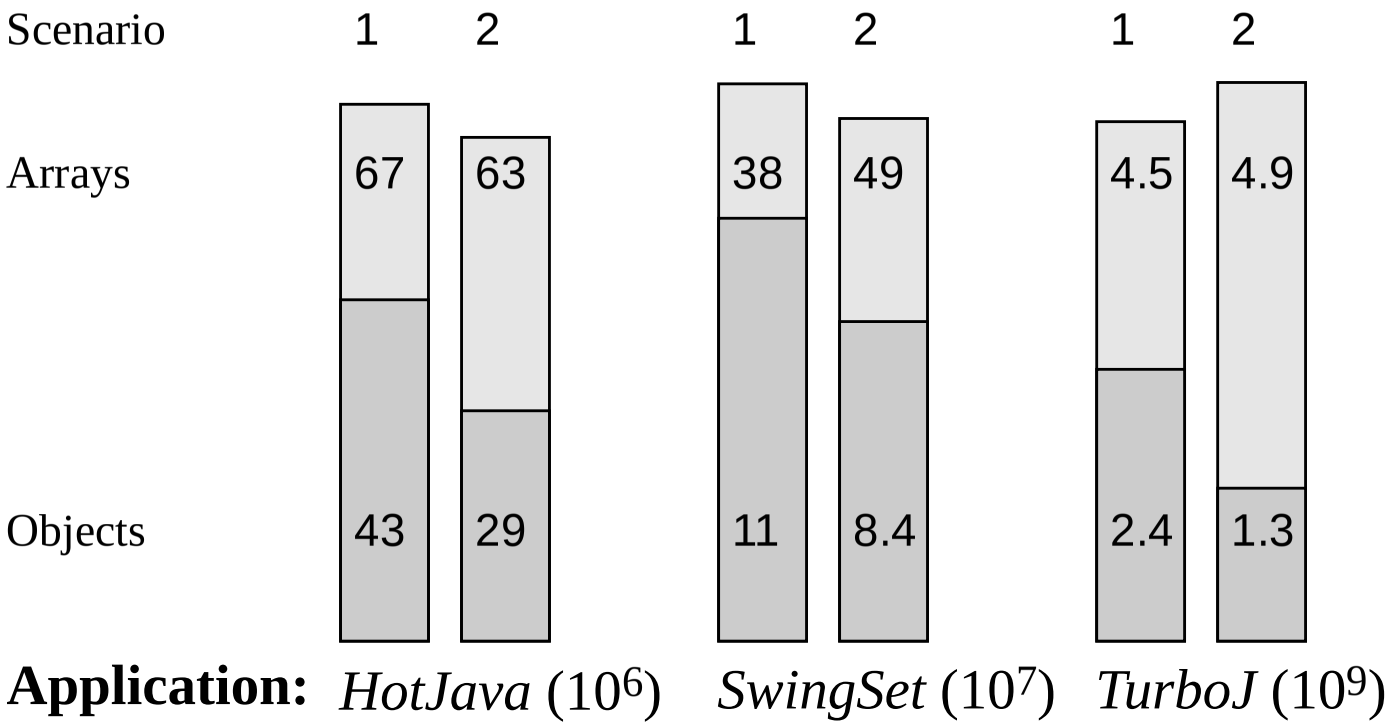
Cost of exact root scanning:

Average number of references stored on a call for three large applications:



Experimental Data

Cost for object accesses using handles (scenario 1) and blocks (scenario 2)



Array Layout