Efficient and Viable Handling of Large Object Traces

Philipp Lengauer
Verena Bitto
Hanspeter Mössenböck

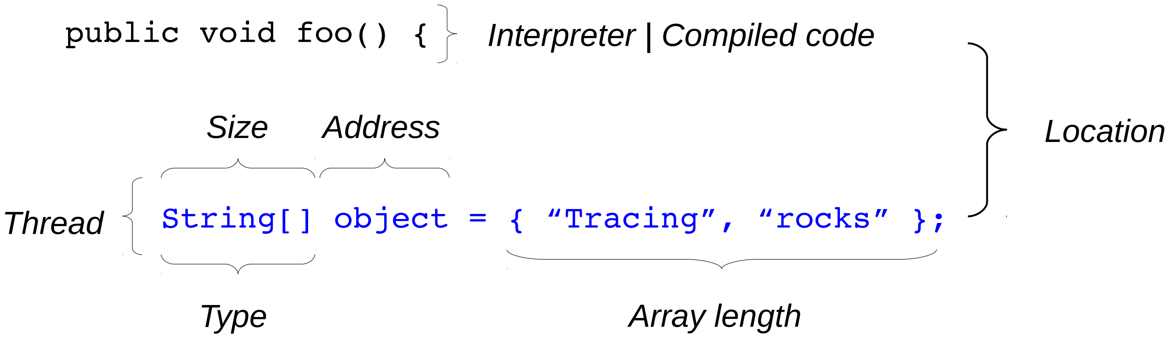
2016-03-15

dynatrace

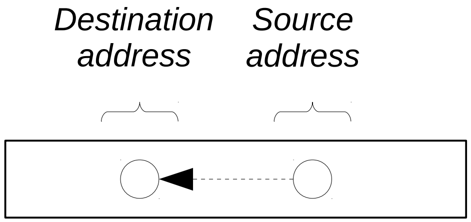
JKU
JOHANNES KEPLER
UNIVERSITÄT LINZ

Recap - AntTracks

… we would know all there is to know about every object?

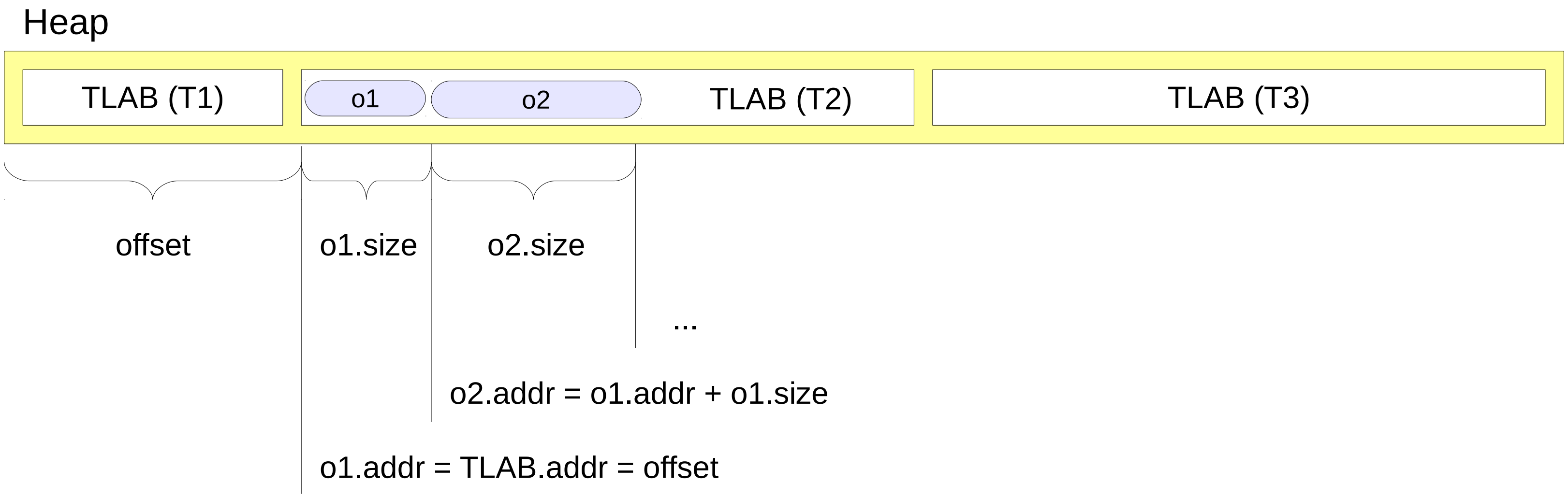


```
public void foo() {
    String[] object = { “Tracing”, “rocks” };
}
```



… then we could reproduce the entire heap for every point in time and do offline analysis!

Allocations



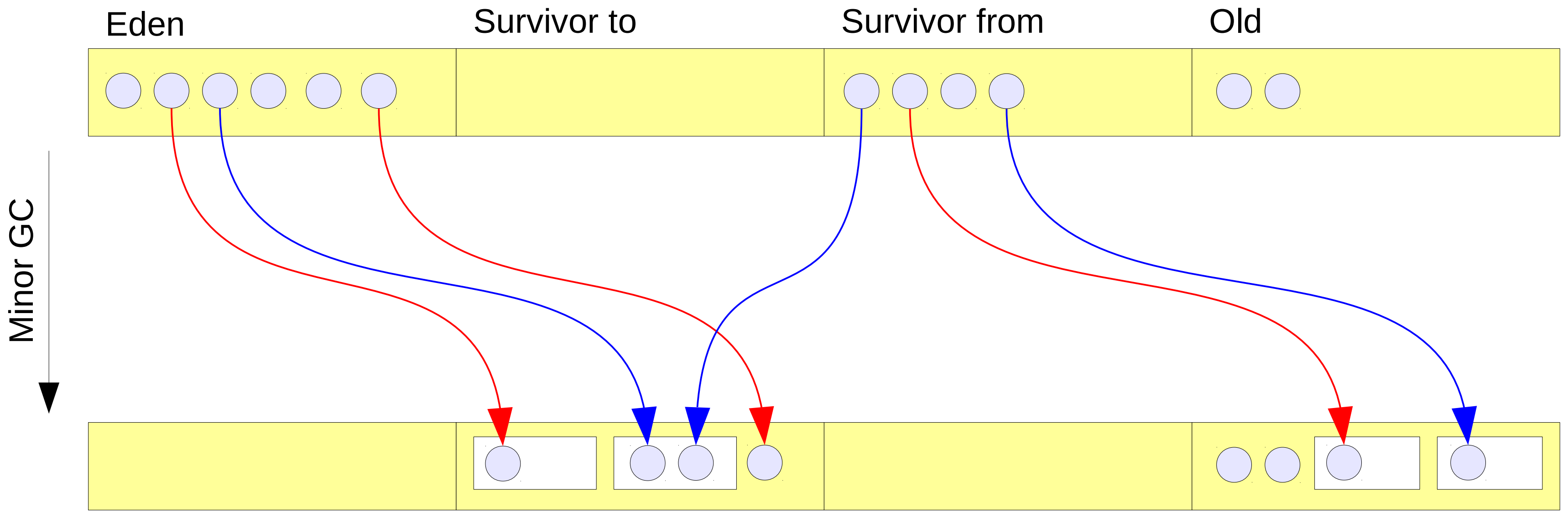
o2.addr = o1.addr + o1.size

o1.addr = TLAB.addr = offset

$$\mathit{addr}(o_n) = \begin{cases} \mathit{addr}(\mathit{TLAB}(o_n)) & \mathit{if}\ n = 1 \\ \mathit{addr}(o_{n-1}) + \mathit{size}(o_{n-1}) & \mathit{else} \end{cases}$$

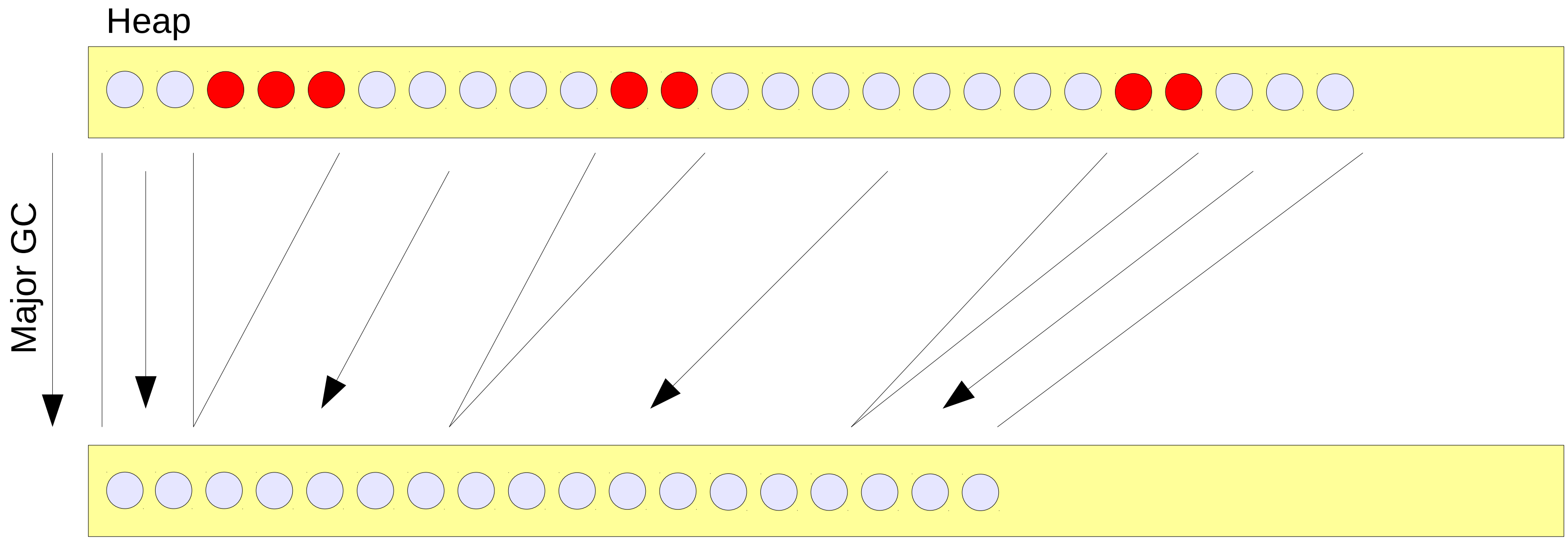
Addresses of objects that are allocated into a TLAB are computable offline!

Minor GCs



New addressed of moved objects are computable offline!

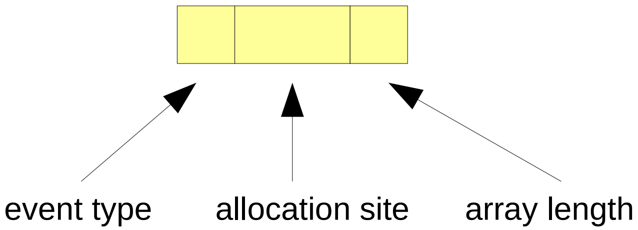
Major GCs



Claim: objects live and die in groups due to their sequential allocation

Optimized Events

Optimized allocation event



event type

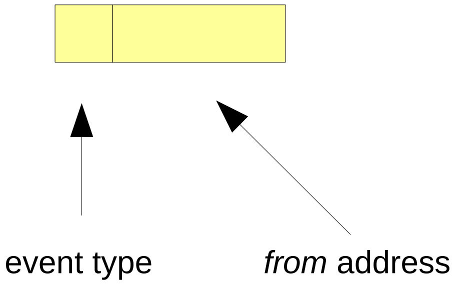
allocation site

array length

address → previous events + TLAB information

→ 4 bytes per allocation
→ **computable at compile-time (JIT)**

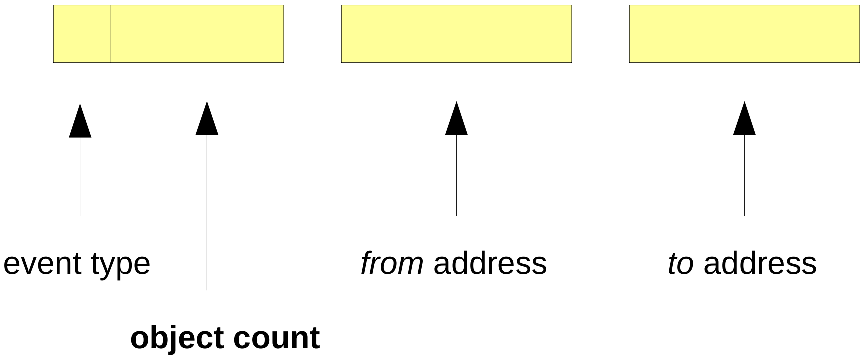
Optimized move event



event type

from address

Region move event



event type

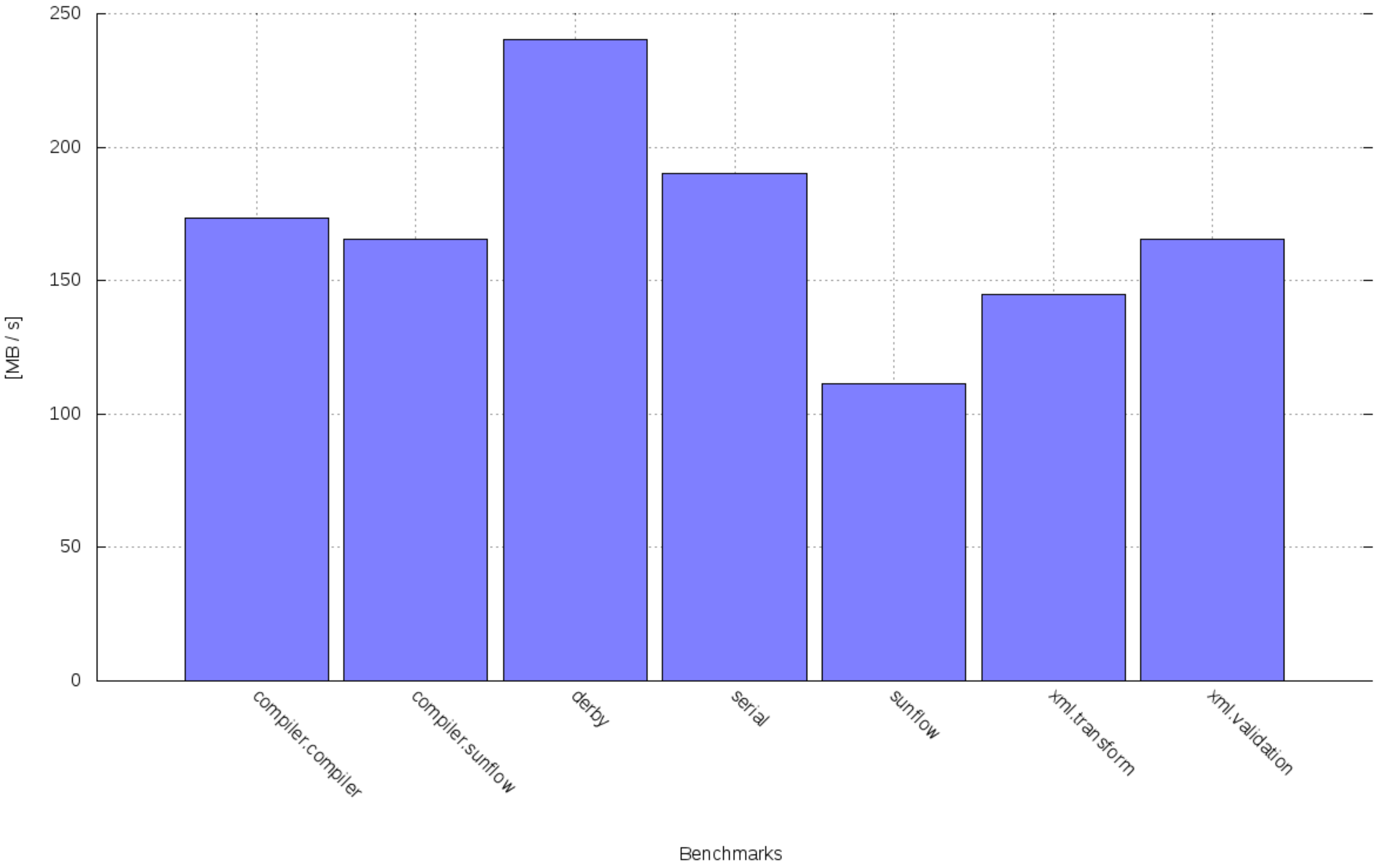
object count

from address

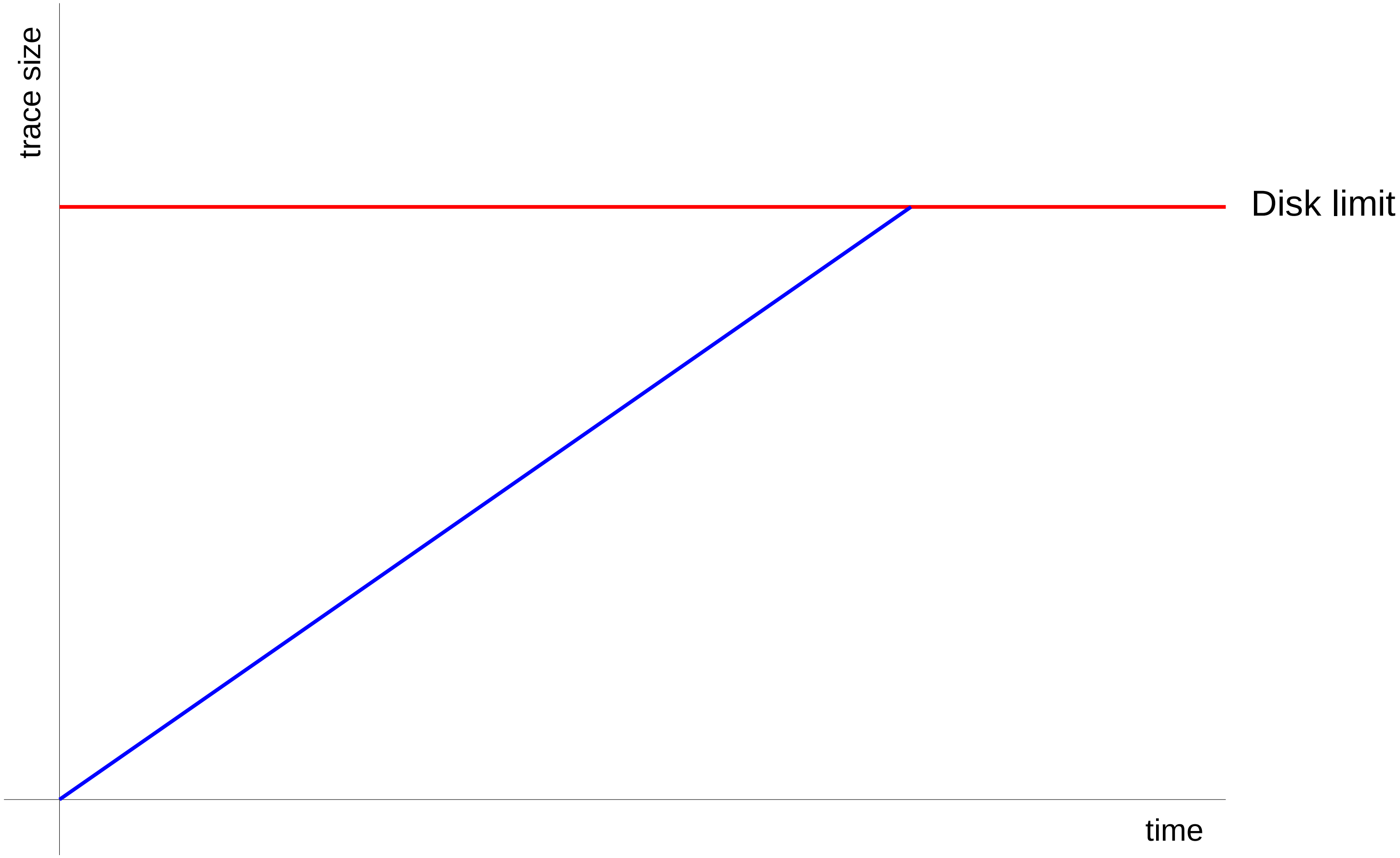
to address

~ 312 objects per event
(3.65Kb -> 12b)

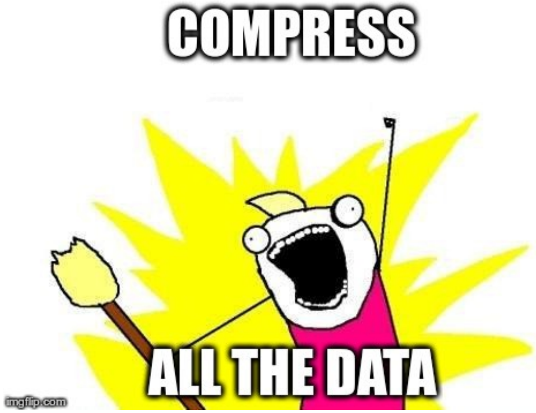
Digging Our Own Grave



Trace Size vs Disk Limit

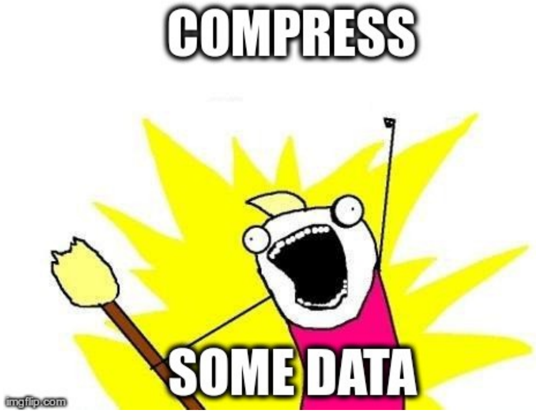


Compression



+ Trace reduced to **21.6%**

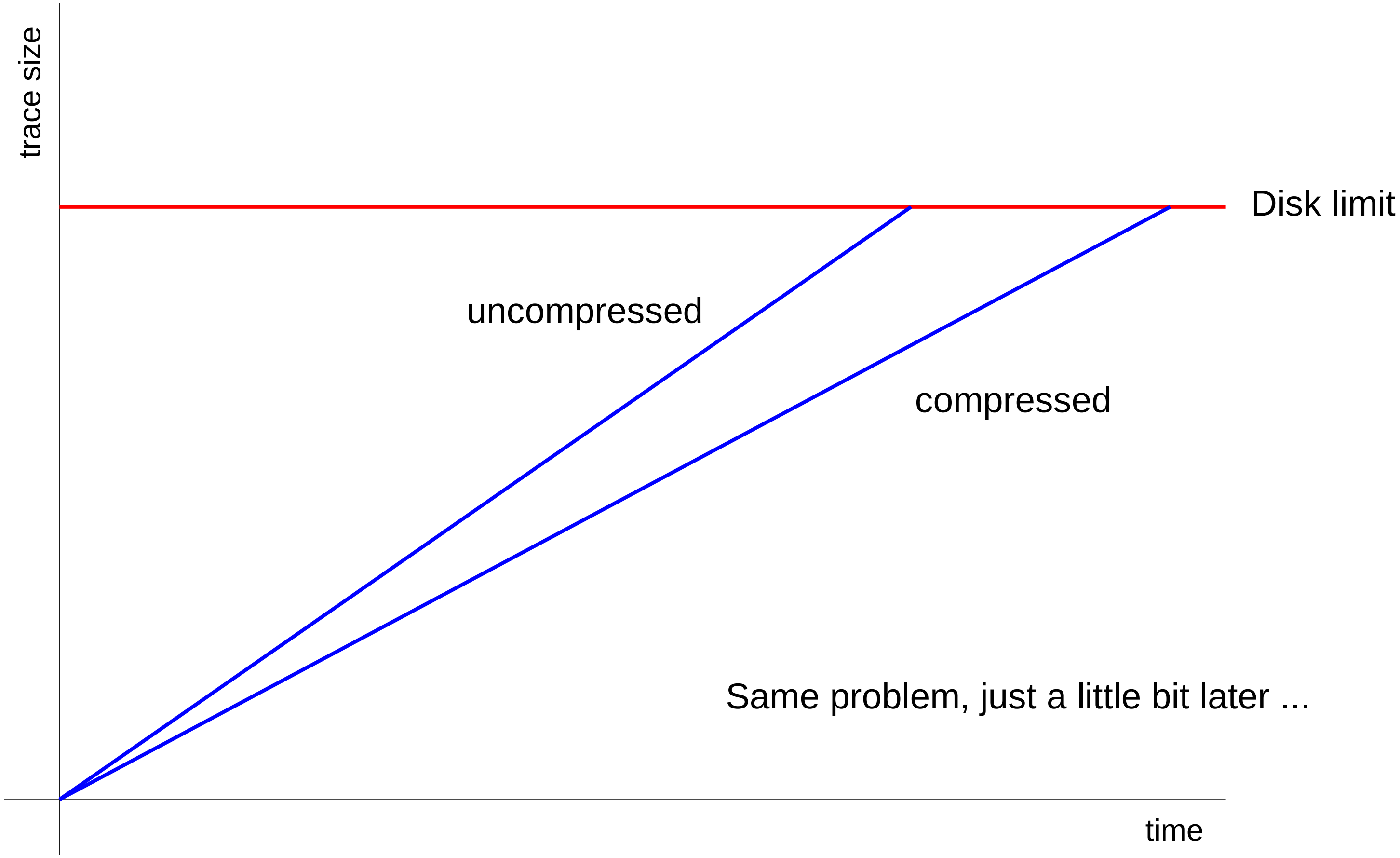
- Overhead increased by **21.9%**



~ Trace reduced to **89.7%**

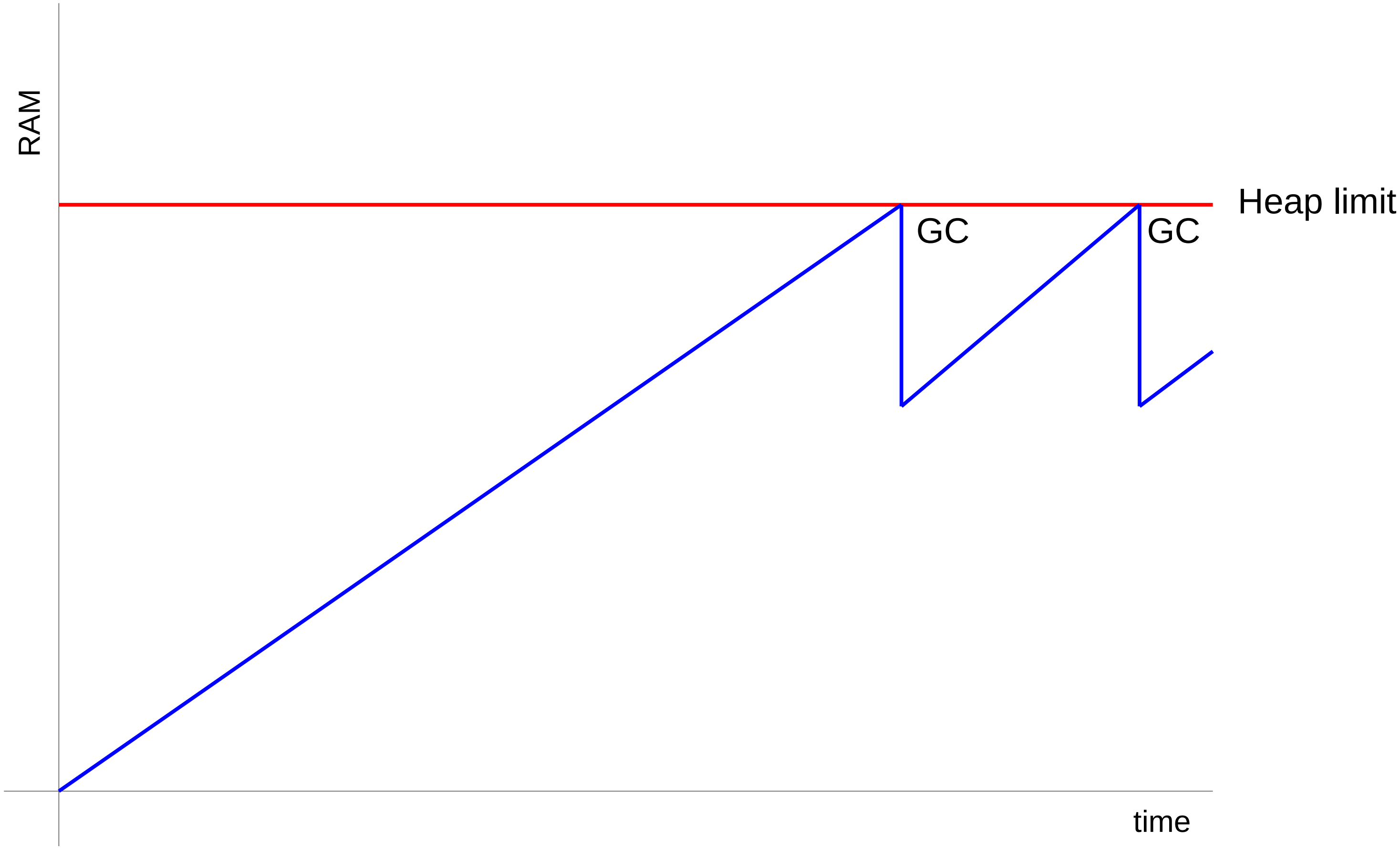
+ Overhead increased by **2.3%**

Compression

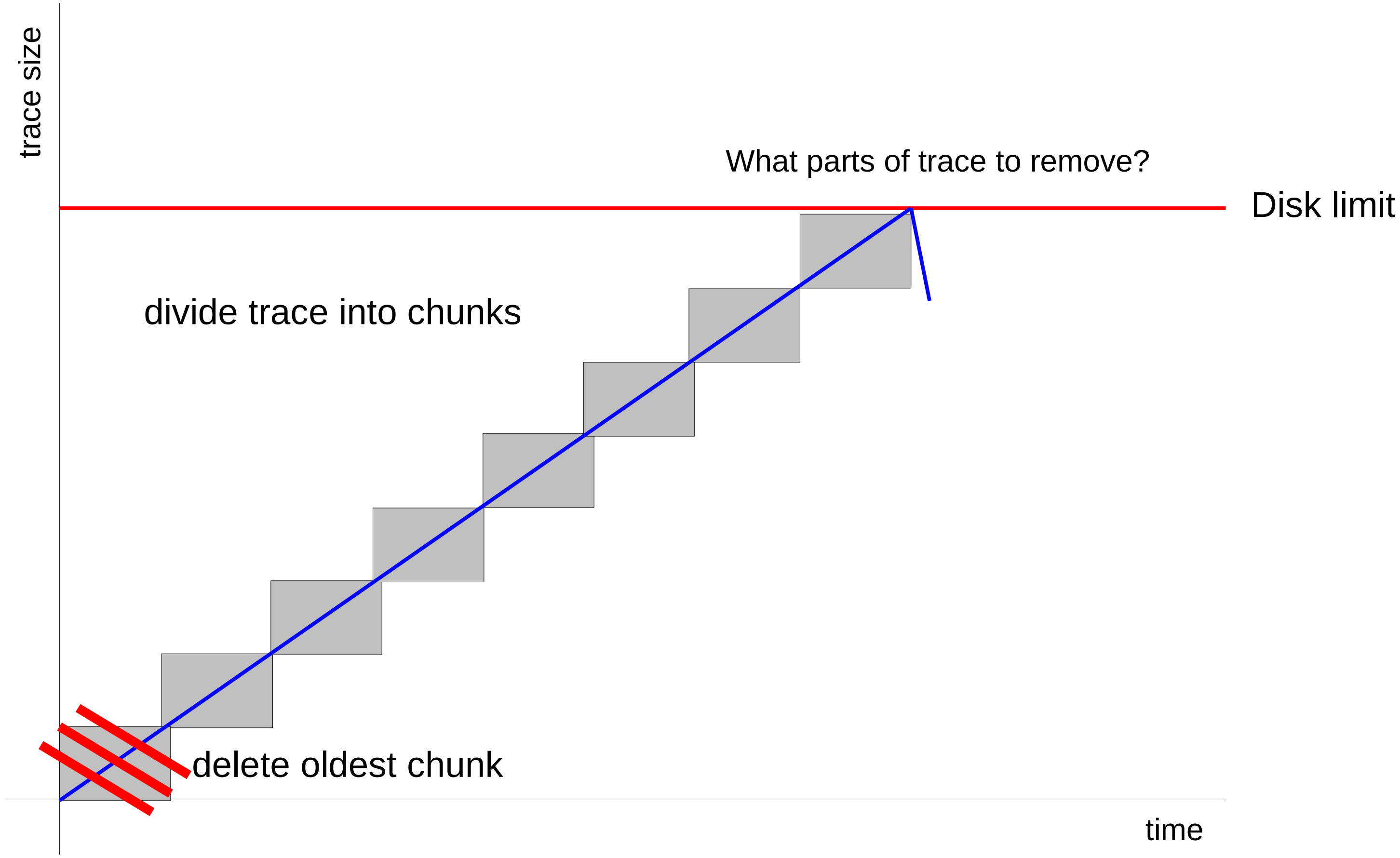


Same problem, just a little bit later ...

Similar Problem: Objects vs Heap Size



Rethink Trace Size vs Disk Limit



What parts of trace to remove?

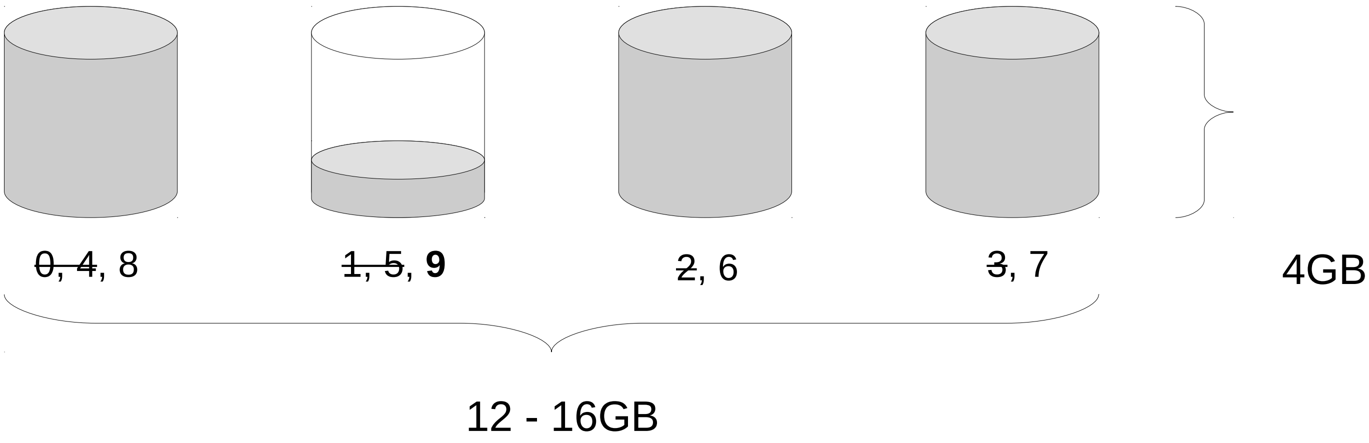
Disk limit

divide trace into chunks

delete oldest chunk

Rotation

Split trace into n files, overwrite oldest file first.



~~0, 4~~, 8

~~1, 5~~, **9**

~~2~~, 6

~~3~~, 7

4GB

12 - 16GB

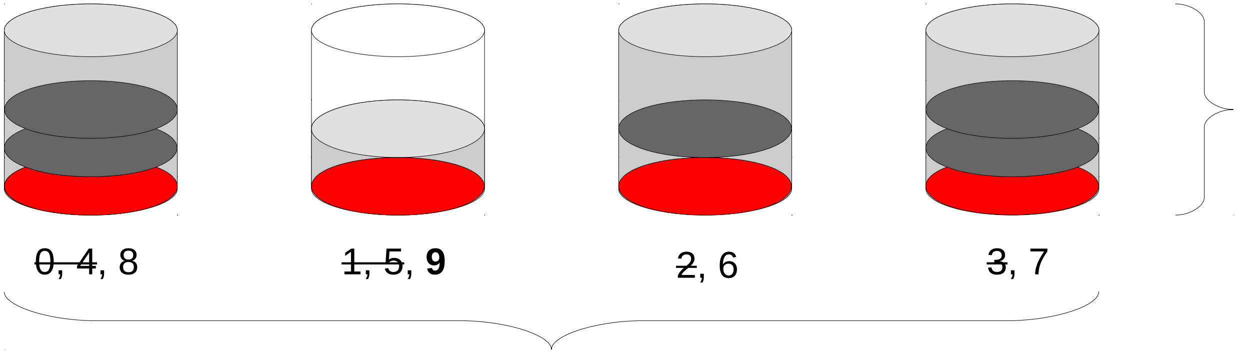
Every trace file may be eventually be the oldest.

What is the state of the heap at the beginning of the oldest file?

Need-to-know

Synchronization Points

Use GCs as synchronization points



~~0, 4~~, 8

~~1, 5~~, **9**

~~2~~, 6

~~3~~, 7

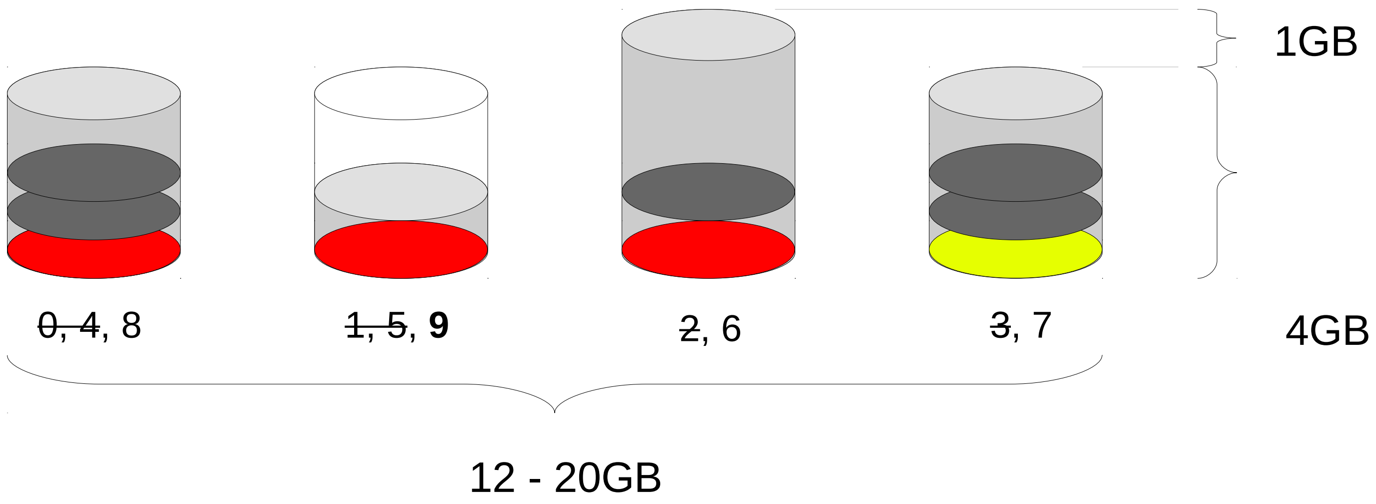
4GB

12 - 16GB

What if no GC occurs at the right point?

Trace Size Deviation

Trigger “**Emergency GCs**” after max deviation is reached.

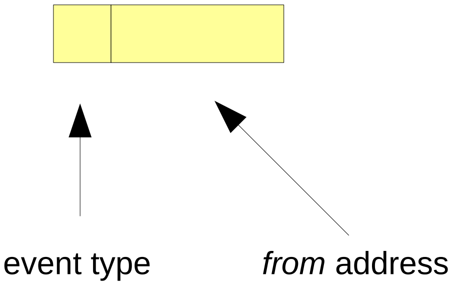


`MaxSize=16GB Deviation=25%`

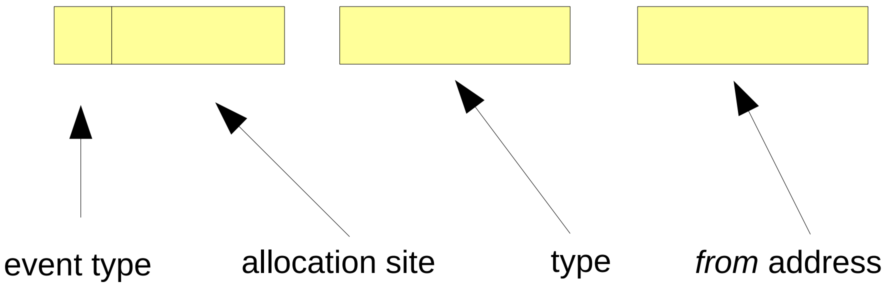
file count = 100% / Deviation
target file size = MaxSize * Deviation
max file size = file size + file size * Deviation

Synchronization GC

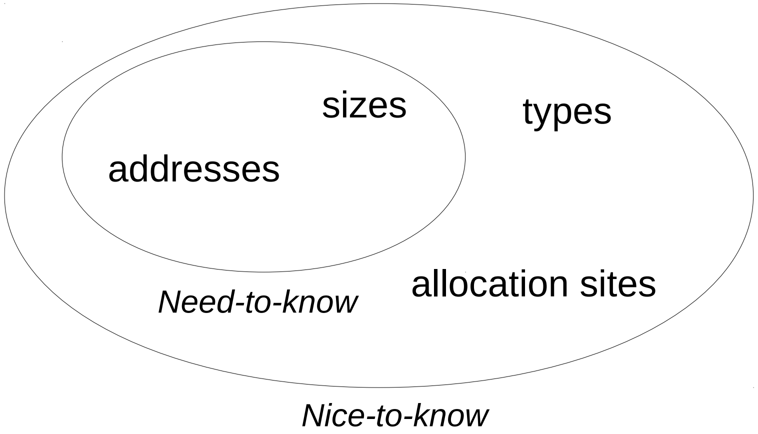
Optimized move event



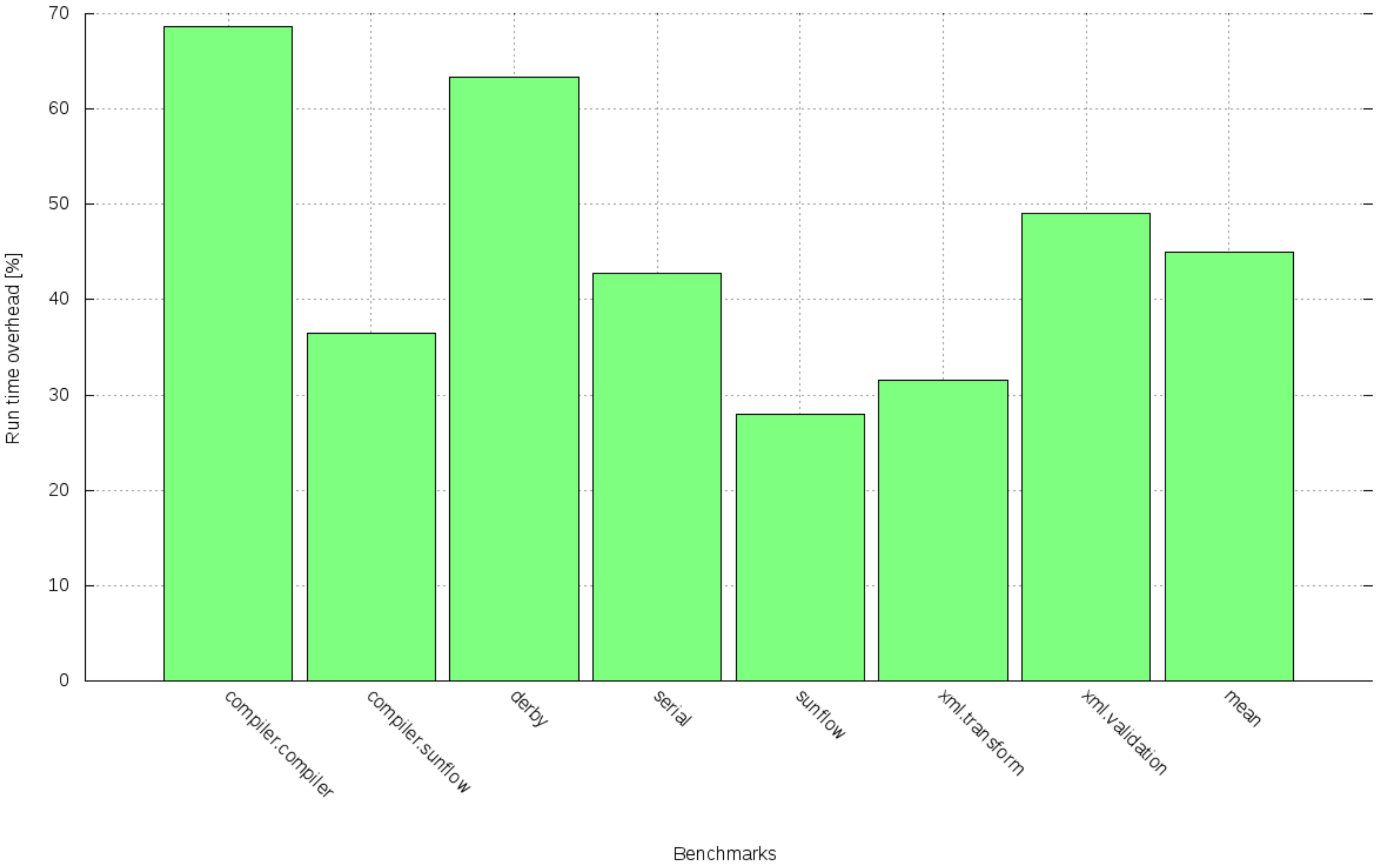
Optimized move sync event



Replace all *move* events with *move sync* events.

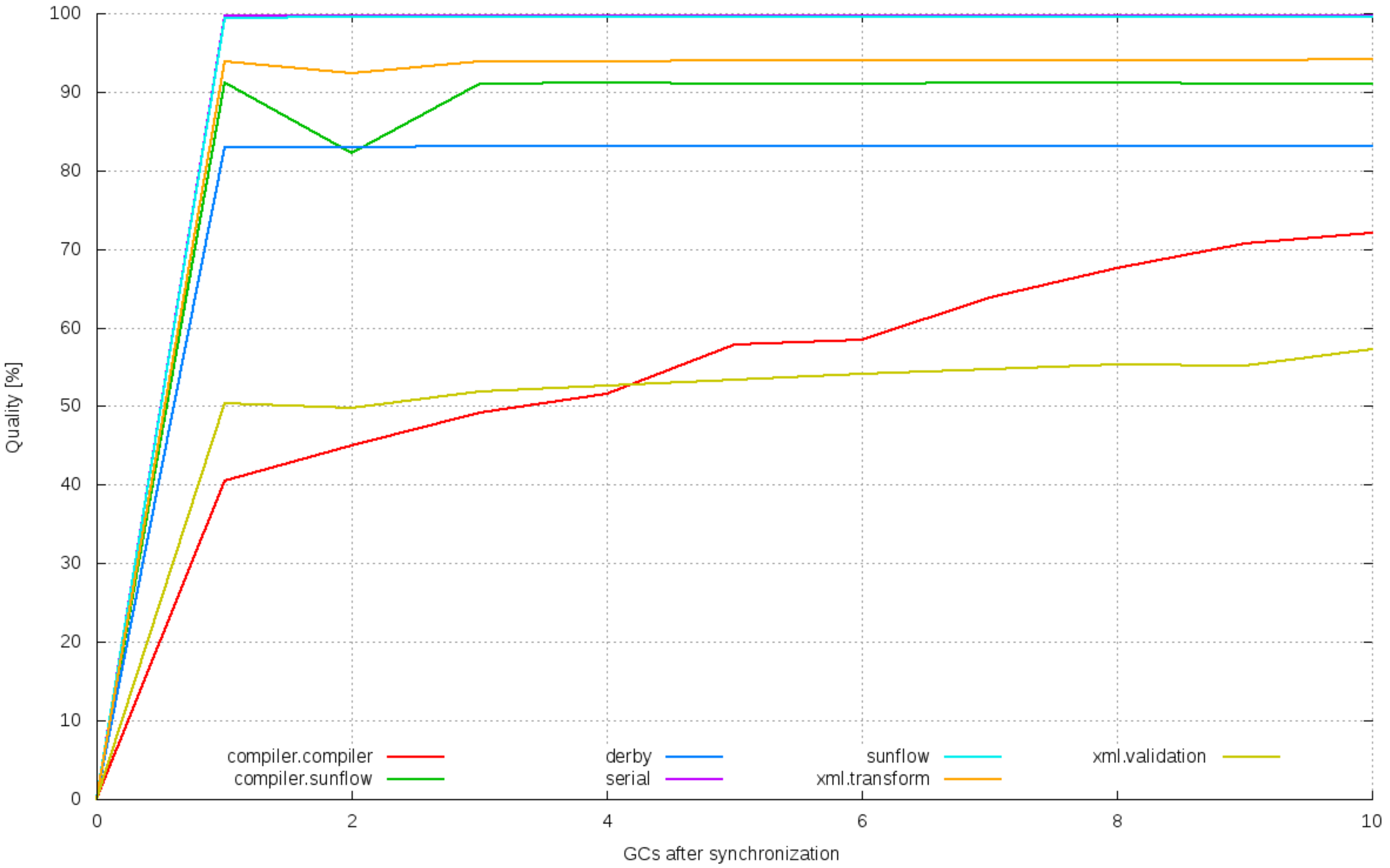


Overhead



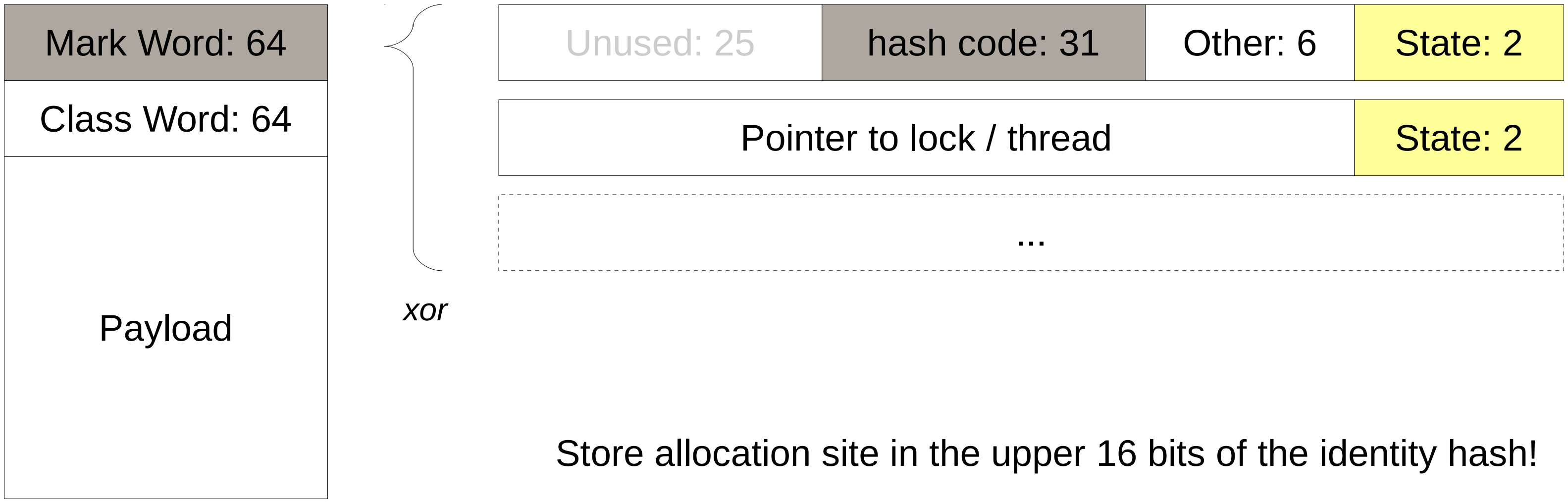
Run time overhead [%]
70
60
50
40
30
20
10
0
compiler.compiler
compiler.sunflow
derby
serial
sunflow
xml.transform
xml.validation
mean
Benchmarks

Quality



100
90
80
70
60
50
40
30
20
10
0
Quality [%]
0
2
4
6
8
10
GCs after synchronization
compiler.compiler
compiler.sunflow
derby
serial
sunflow
xml.transform
xml.validation

Restoring Allocation Sites



Mark Word: 64

Class Word: 64

Payload

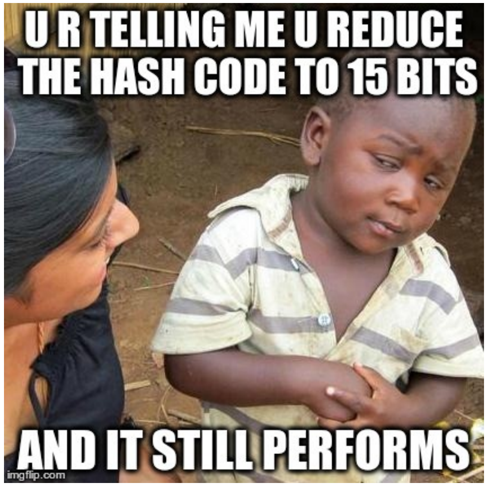
xor

Unused: 25	hash code: 31	Other: 6	State: 2

Pointer to lock / thread	State: 2

...

Store allocation site in the upper 16 bits of the identity hash!



- Must generate hash code eagerly for every (!) object.
- Reduces entropy of the hash to **0.0015%**.

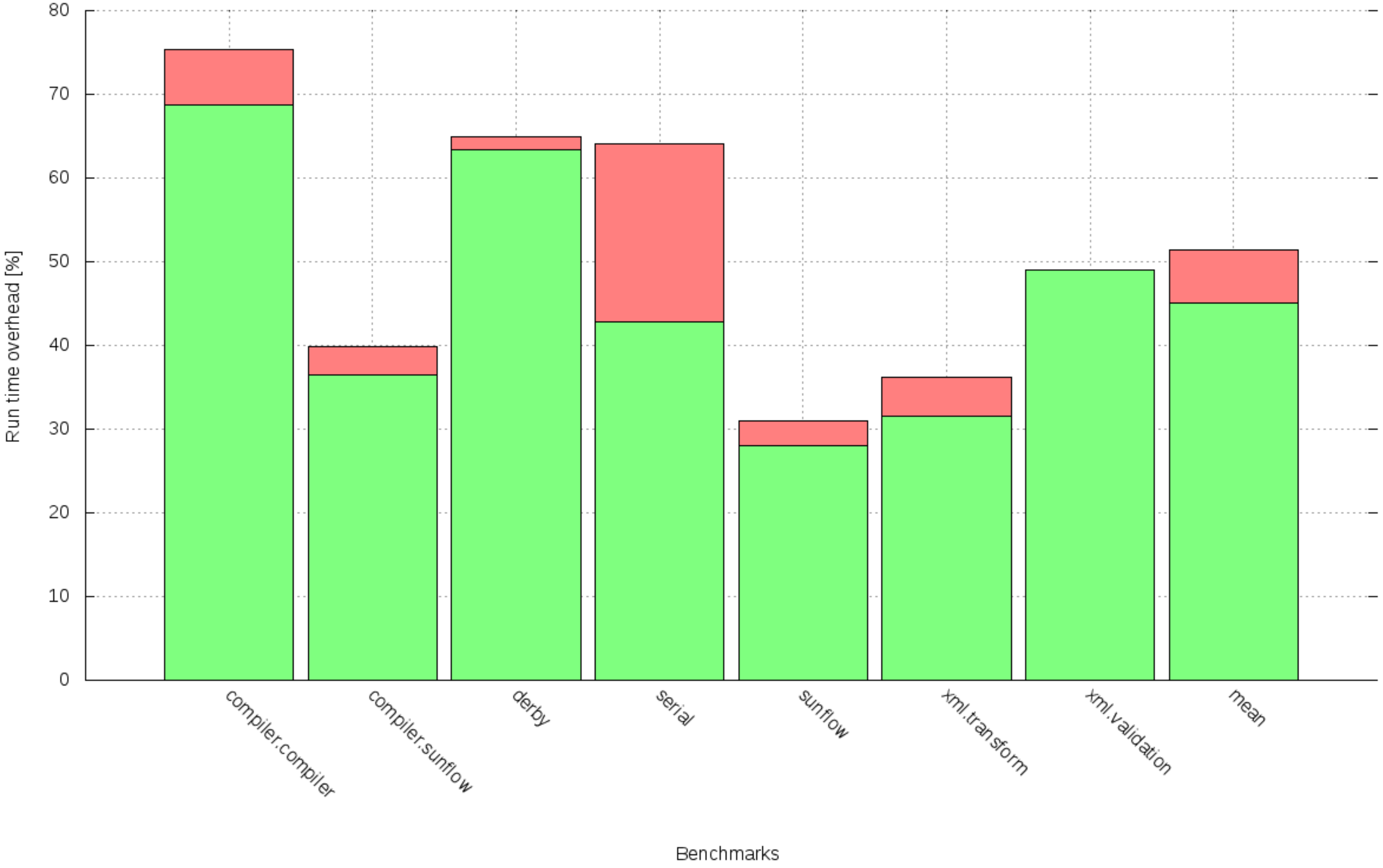
Artificial Worst Case

```
void main() {
    Set<Object> set = new HashSet<>();
    for(int i = 0; i < 1_000_000_000; i++) {
        set.put(create());
    }
    for(int i = 0; i < 1_000_000_000; i++) {
        set.contains(create());
    }
}

Object create() {
    // all objects have same allocation site
    return new Object();
}
```

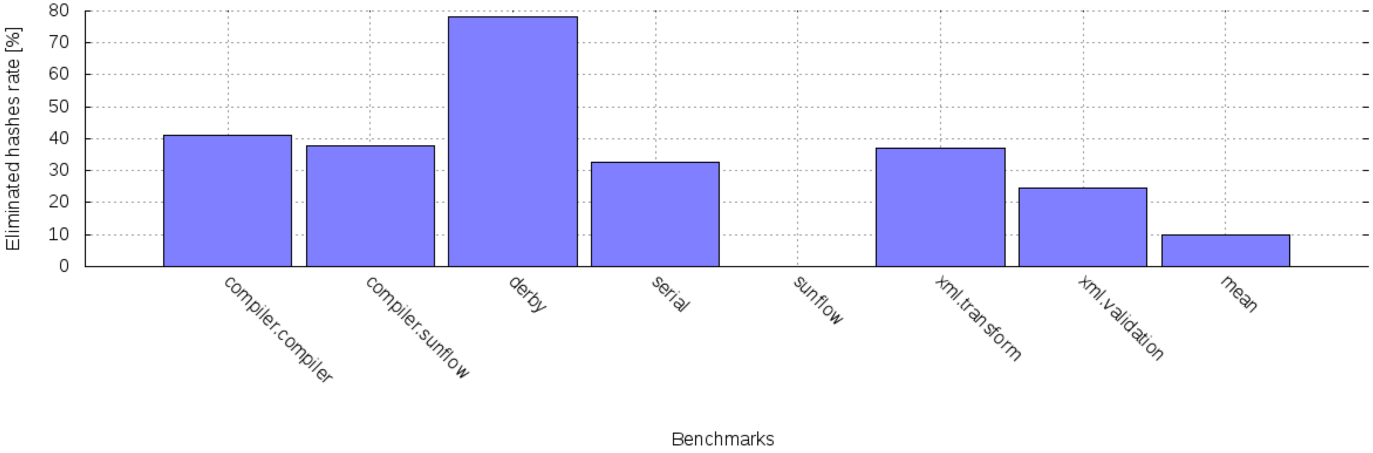
→ **run time +2191%**

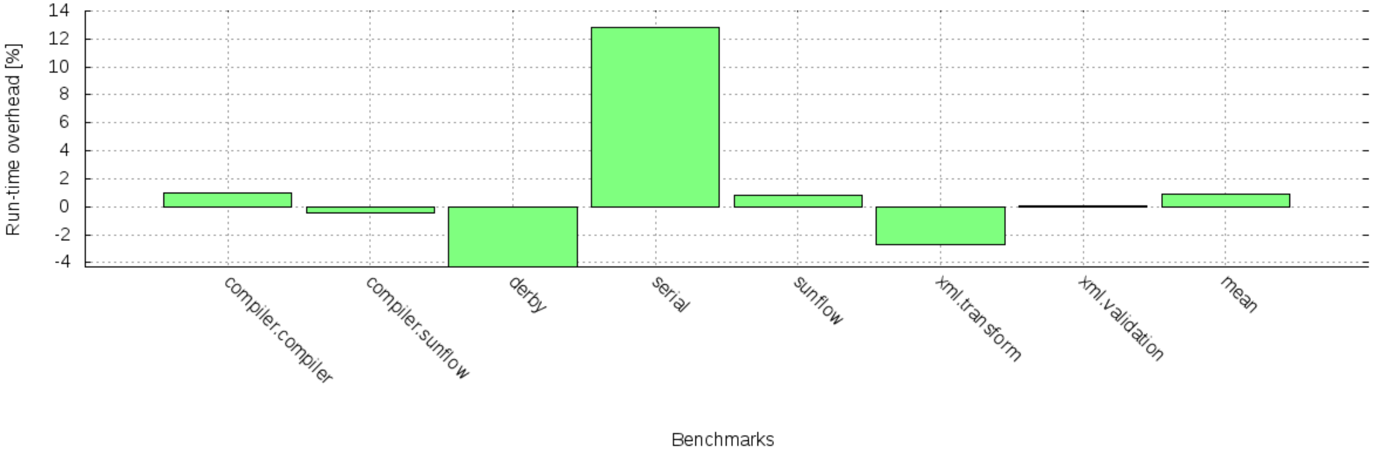
Overhead with Saving Allocation Sites



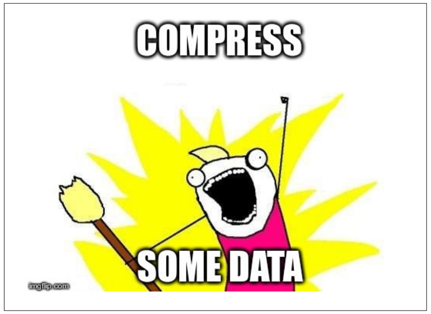
Reducing Hash Code Generation

Assumption: classes overwriting `hashcode()` will *most likely* never use the identity hash!

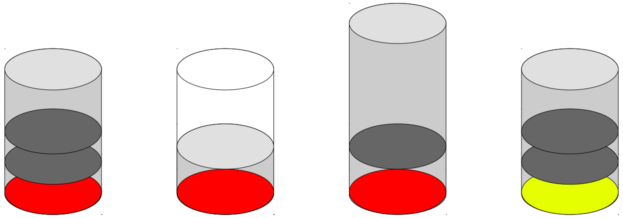




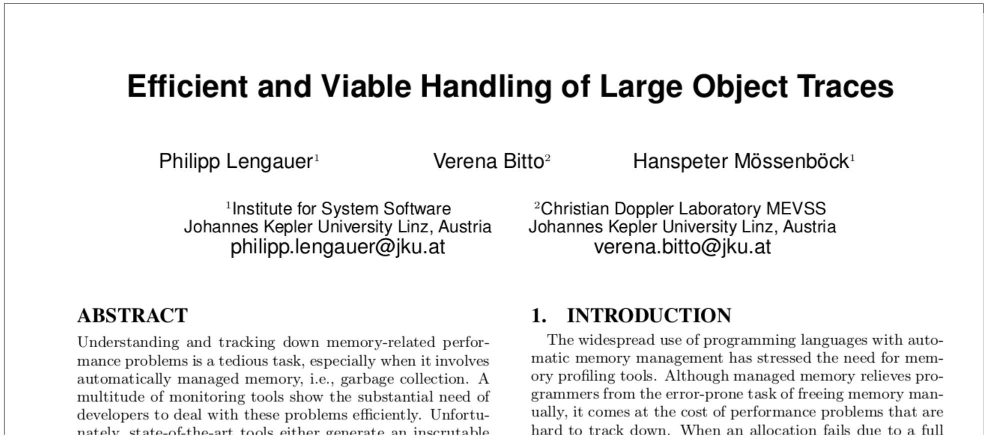
Summary



On-the-fly compression



Trace rotation



Efficient and Viable Handling of Large Object Traces

Philipp Lengauer[1] Verena Bitto[2] Hanspeter Mössenböck[1]

[1]Institute for System Software
Johannes Kepler University Linz, Austria
philipp.lengauer@jku.at

[2]Christian Doppler Laboratory MEVSS
Johannes Kepler University Linz, Austria
verena.bitto@jku.at

ABSTRACT

Understanding and tracking down memory-related performance problems is a tedious task, especially when it involves automatically managed memory, i.e., garbage collection. A multitude of monitoring tools show the substantial need of developers to deal with these problems efficiently. Unfortunately, state-of-the-art tools either generate an inscrutable

1. INTRODUCTION

The widespread use of programming languages with automatic memory management has stressed the need for memory profiling tools. Although managed memory relieves programmers from the error-prone task of freeing memory manually, it comes at the cost of performance problems that are hard to track down. When an allocation fails due to a full

ICPE'16

Q&A

?