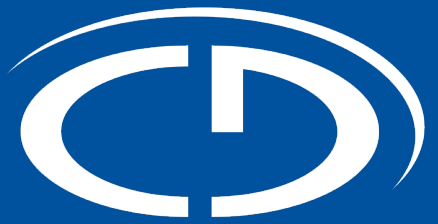


# Accurate and Efficient Monitoring

Verena Bitto
Philipp Lengauer
**Markus Weninger**

2016-03-13

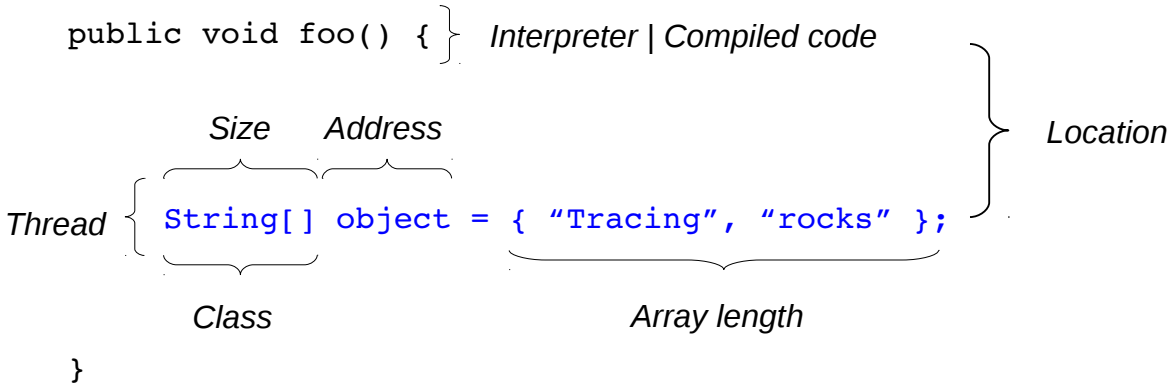
dynatrace

JOHANNES KEPLER
UNIVERSITÄT LINZ

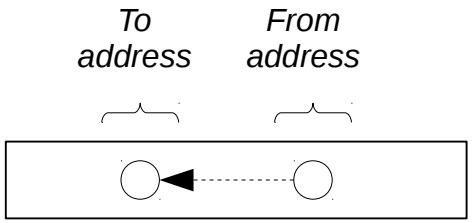
# What if ...

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

## Allocations



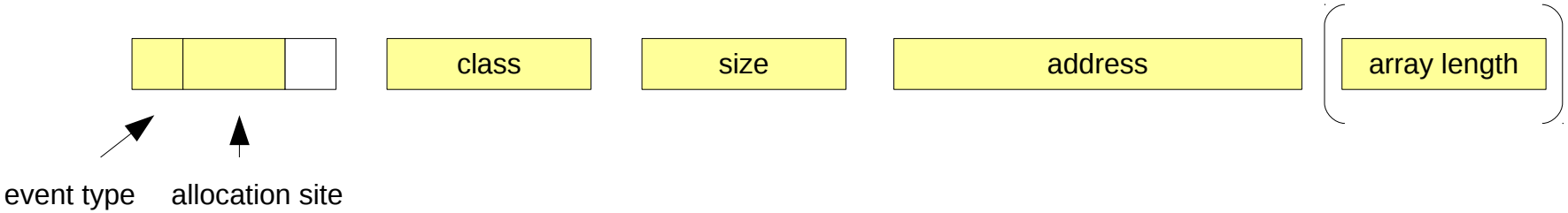

## Moves



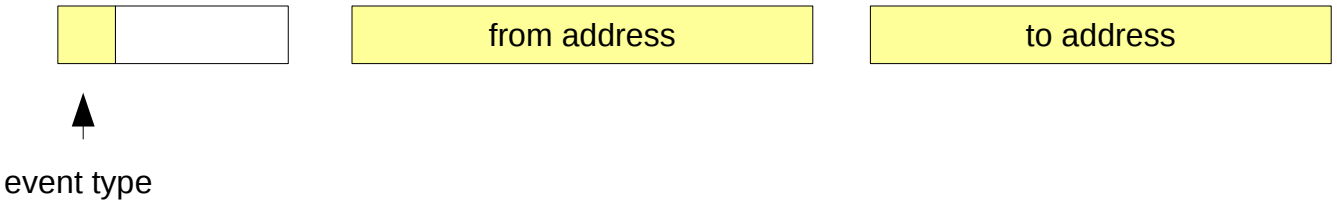

## Deallocations

# Object Events

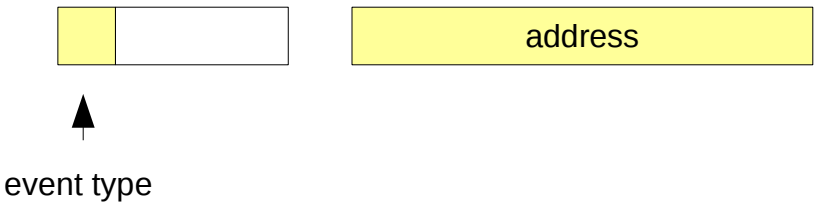
**Generic allocation event**




**Generic move event**




**Generic deallocation event**




*Incremental changes to a virtual heap*

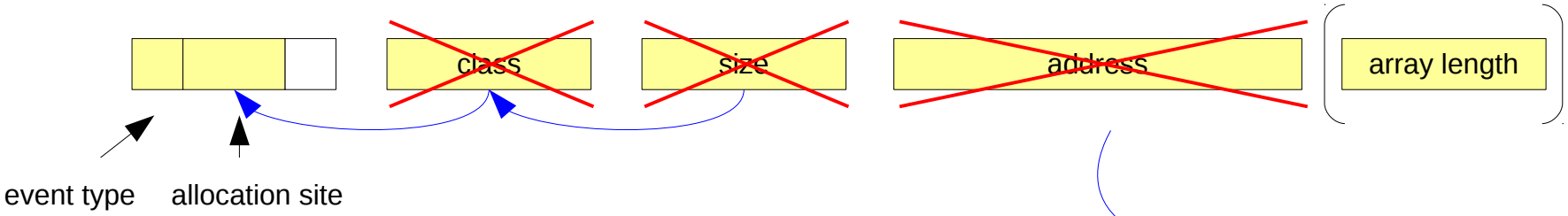
# Changing Our Thinking

*Instead of thinking about recording all the information we want explicitly ...*

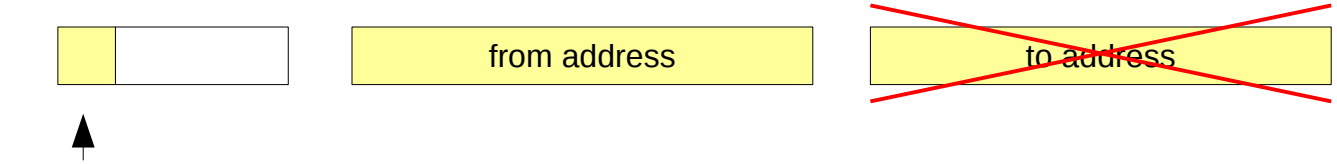
*… think about reconstructing it offline.*

# Object Events

**Generic allocation event**

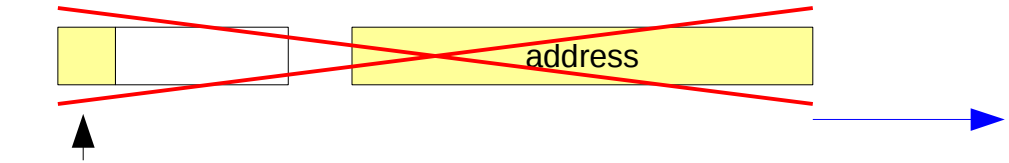



**Generic move event**



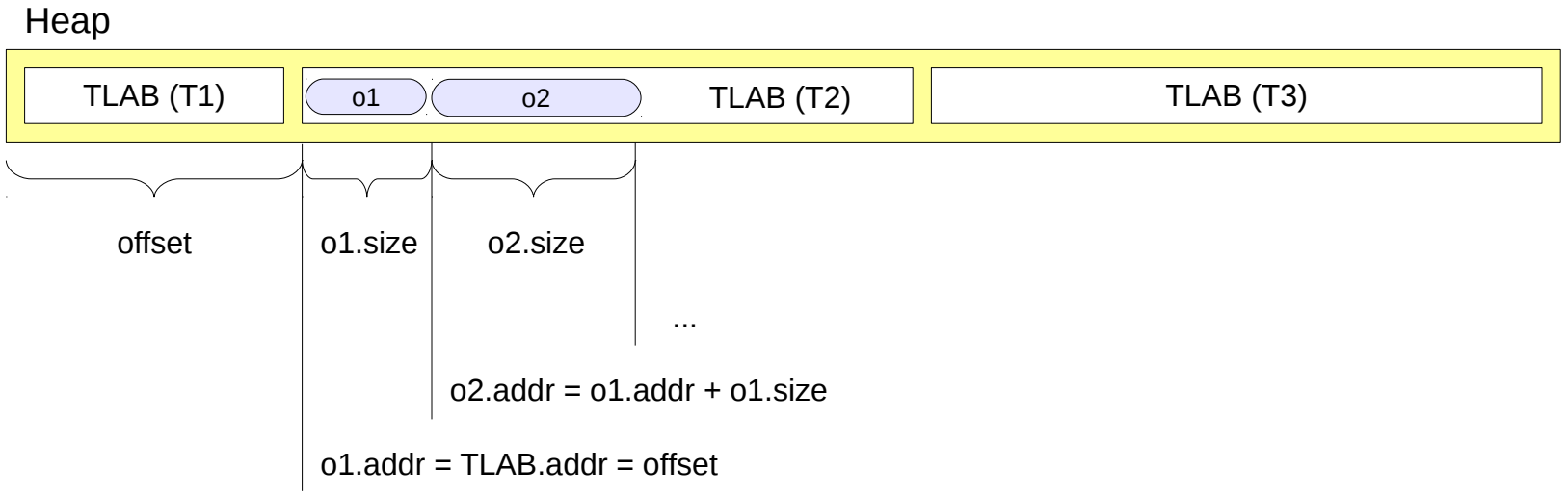

*<u>Assumption</u>: move events are sent for all live objects.*

**Generic Deallocation event**




*All objects without a move event are deallocated!*

# Allocations




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

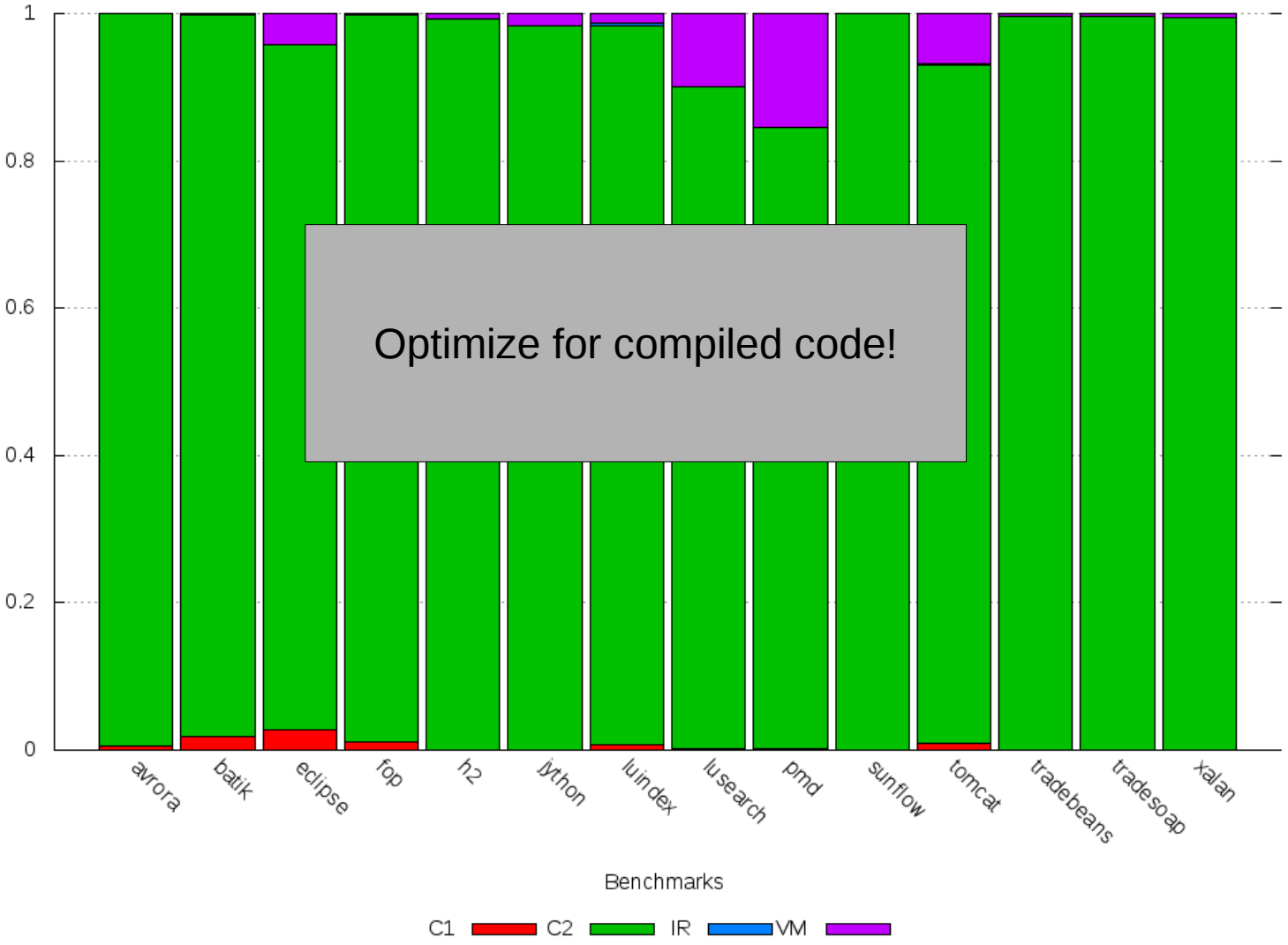
**Addresses of objects that are allocated into a TLAB are computable offline!**

# Changing Our Thinking (Again)

*Instead of thinking about algorithms that are fast for every case ...*

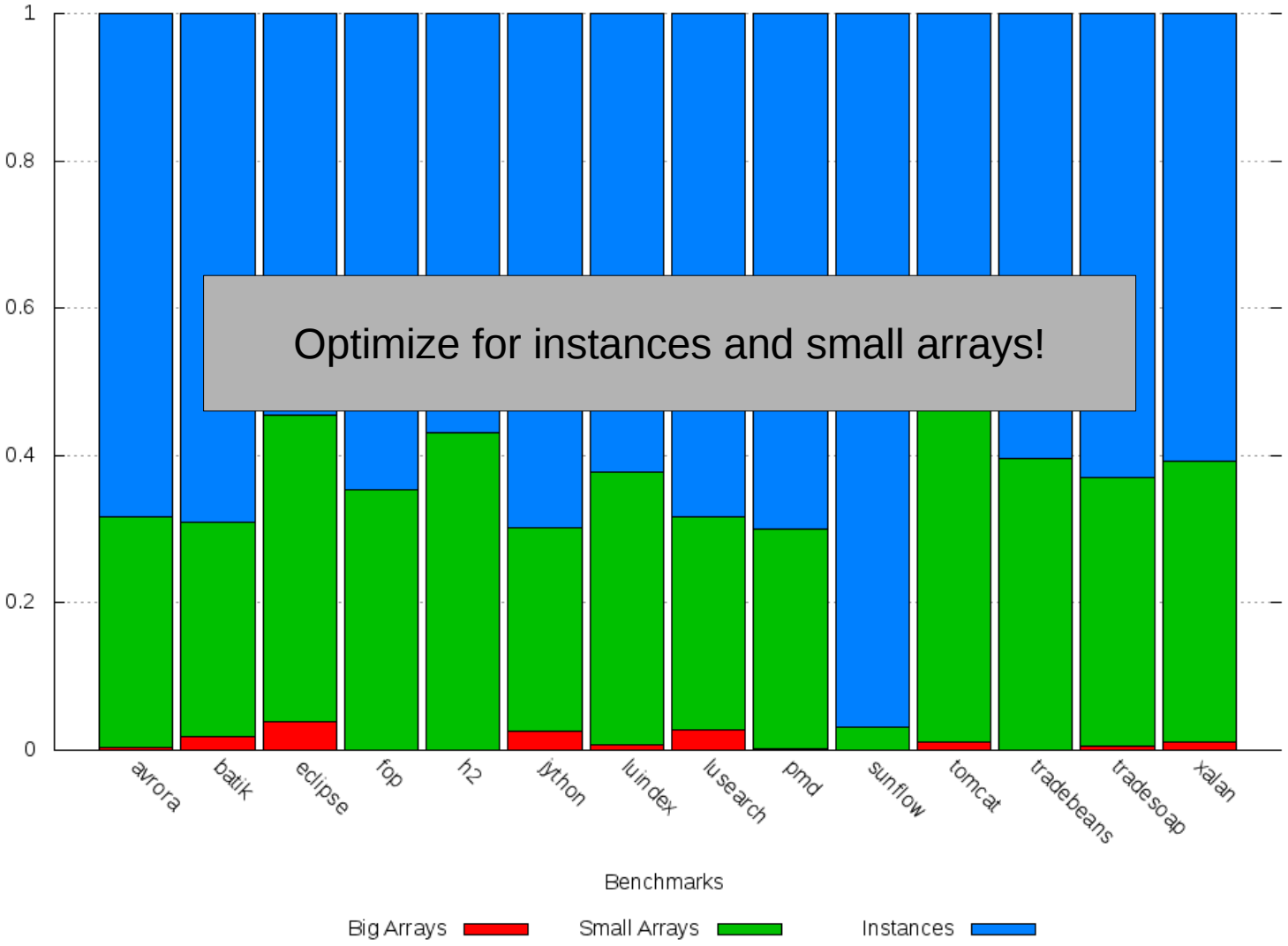
*… think about making the “common case” fast.*

# Allocating Code



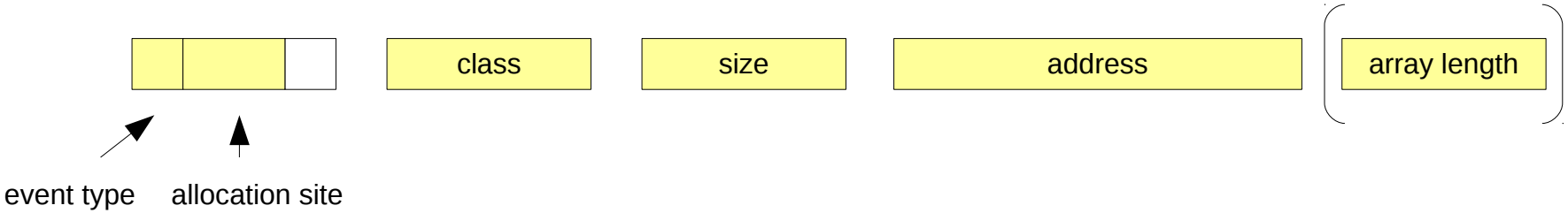

Optimize for compiled code!

# Object Kinds

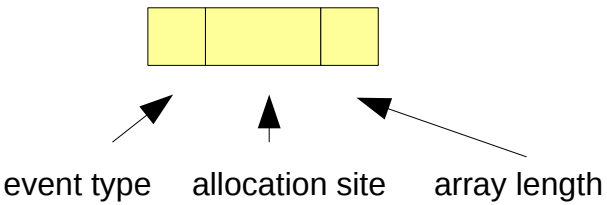


# Optimized Allocation Events

**Generic allocation event**




**Optimized allocation event**




1) only 4 bytes for **most** allocations
2) event is JIT-compile-time constant for instances

# Recording an Optimized Allocation Event

JOHANNES KEPLER UNIVERSITÄT LINZ

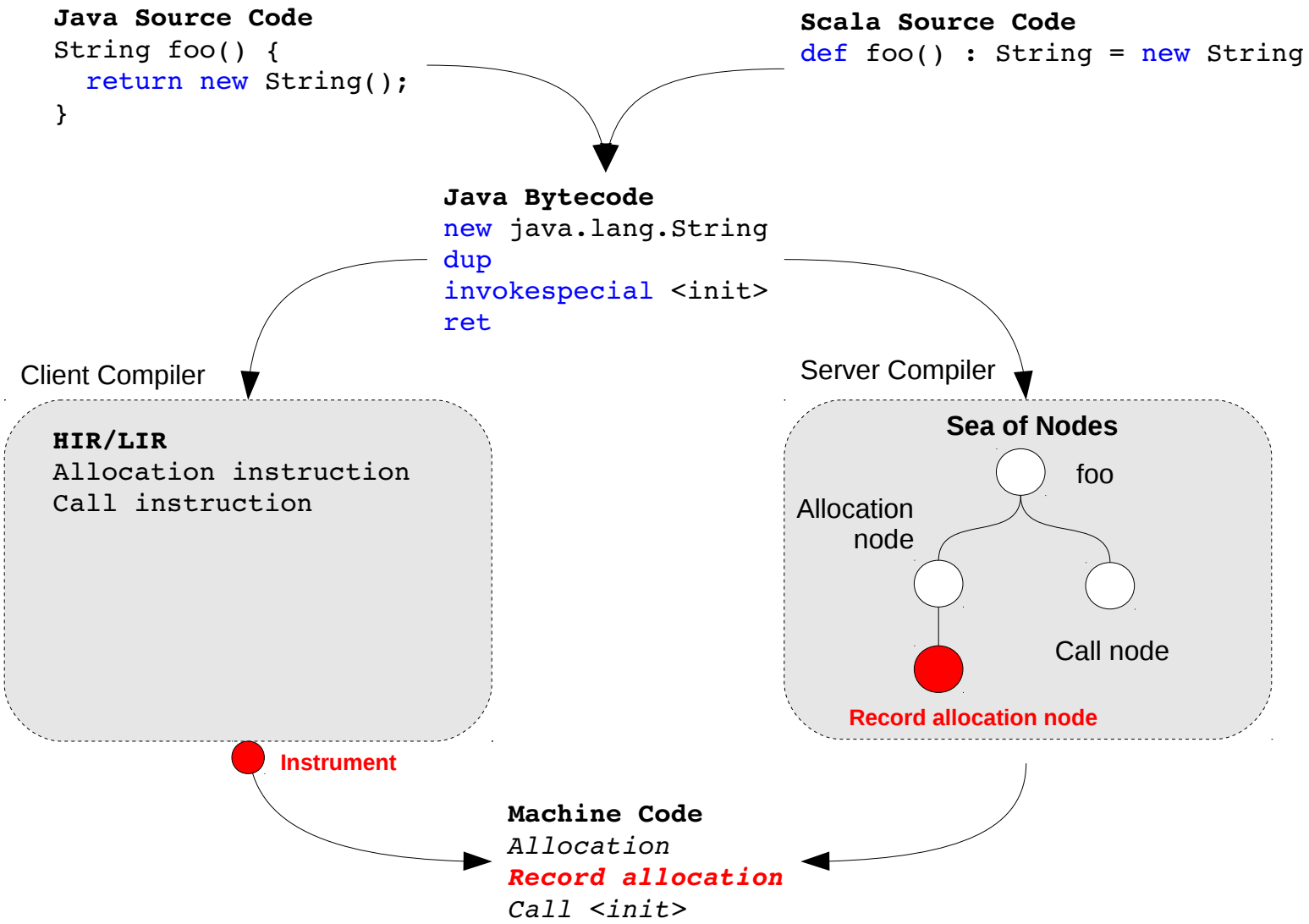
```
// optimized instance allocation
*(top++) = 0xABCDEF00;
```

1 memory write (constant)
1 pointer increment

```
// optimized array allocation
*(top++) = 0xABCDEF00 | array_length;
```

1 bitwise or
1 memory write
1 pointer increment

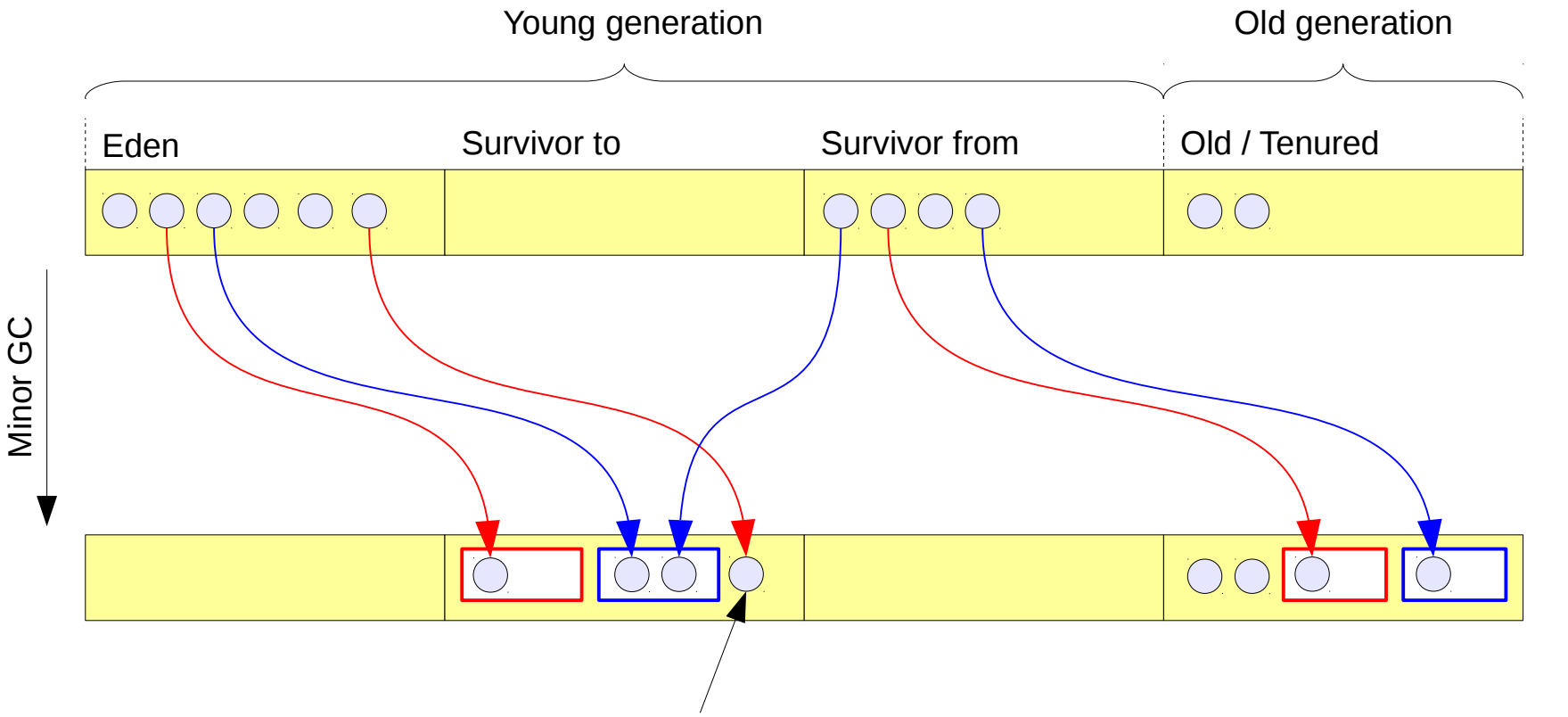
# Instrumenting Allocation Sites



# Move Events

**Generic move event**

# Minor GCs




*Use generic version of move event.*

PLAB of GC-Thread 1

PLAB of GC-Thread 2

Move by GC-Thread 1

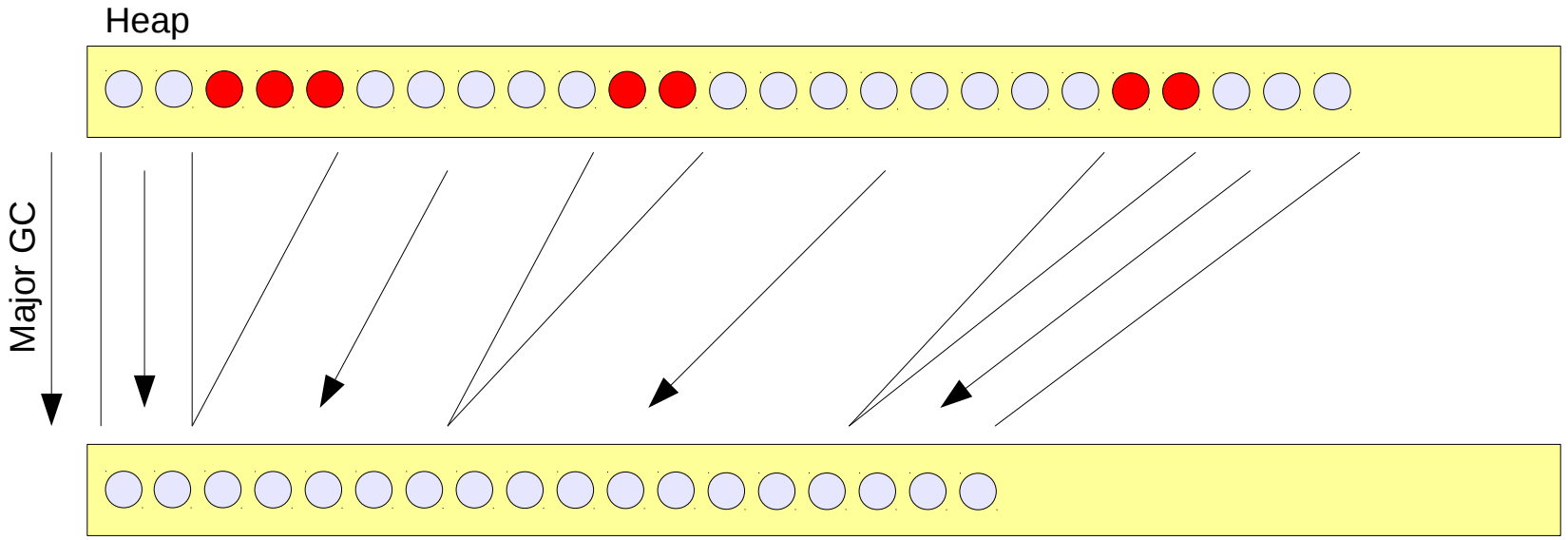
Move by GC-Thread 2

# Optimized Move Events

**Generic move event**


**Optimized move event**

# Major GCs

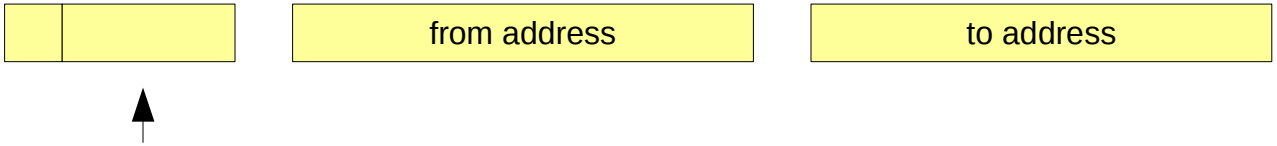



**Claim: objects live and die in groups due to their sequential allocation**

# Move Regions Events

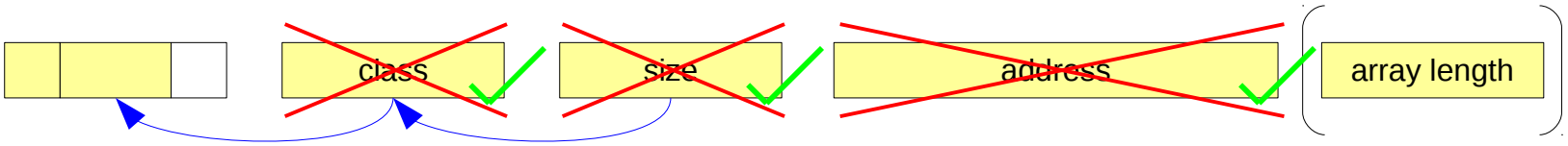
**Generic move event**


**Region move event**




**~ 312 objects per event**
**(6.24Kb -> 20b)**

# Generic Use Cases



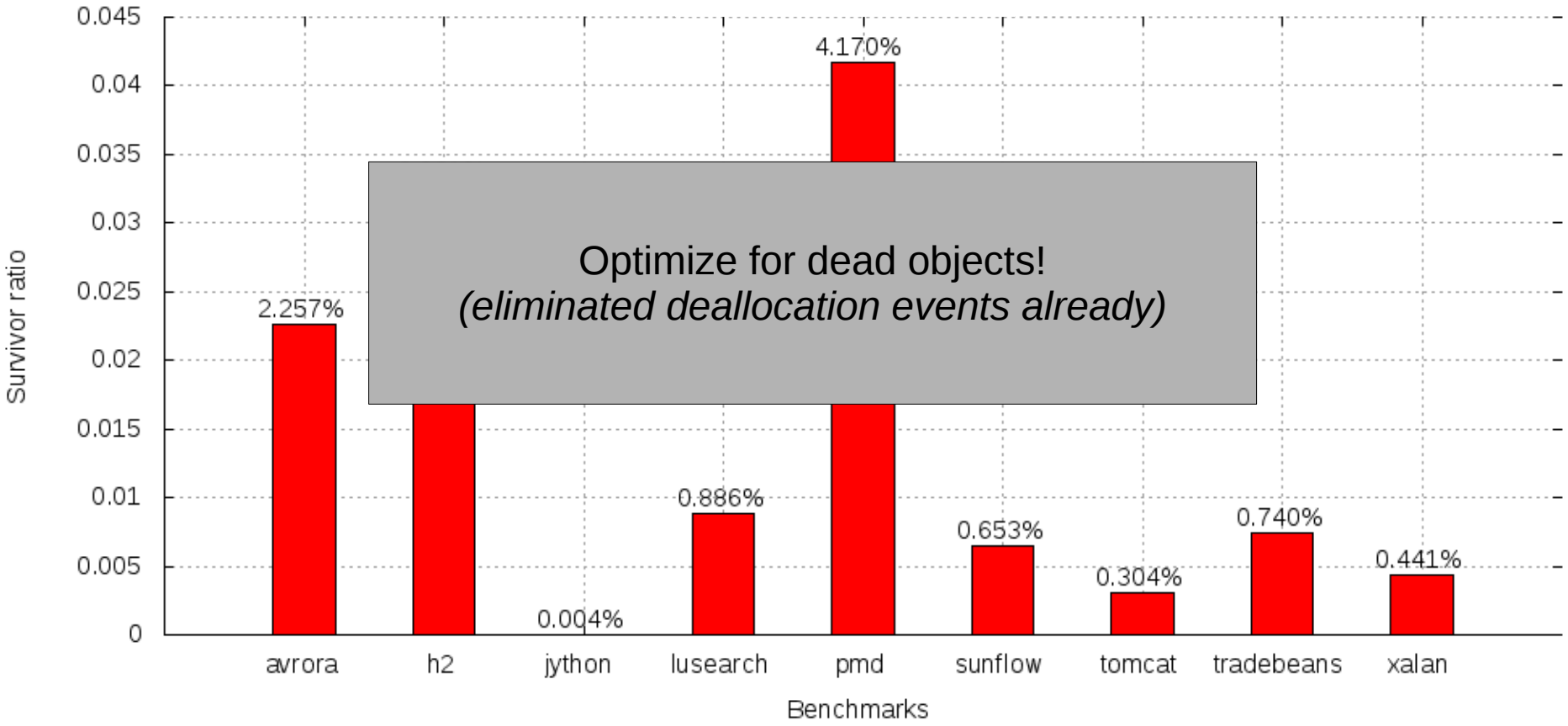

```
Object dolly = obj.clone();
// what is the class of dolly?

new int[3][3][3];
// same allocation site allocates
// objects of different classes
// 1x int[][][], 3x int[][], 9x int[]
```

*VM knows whether the allocation site does not determine the class!*

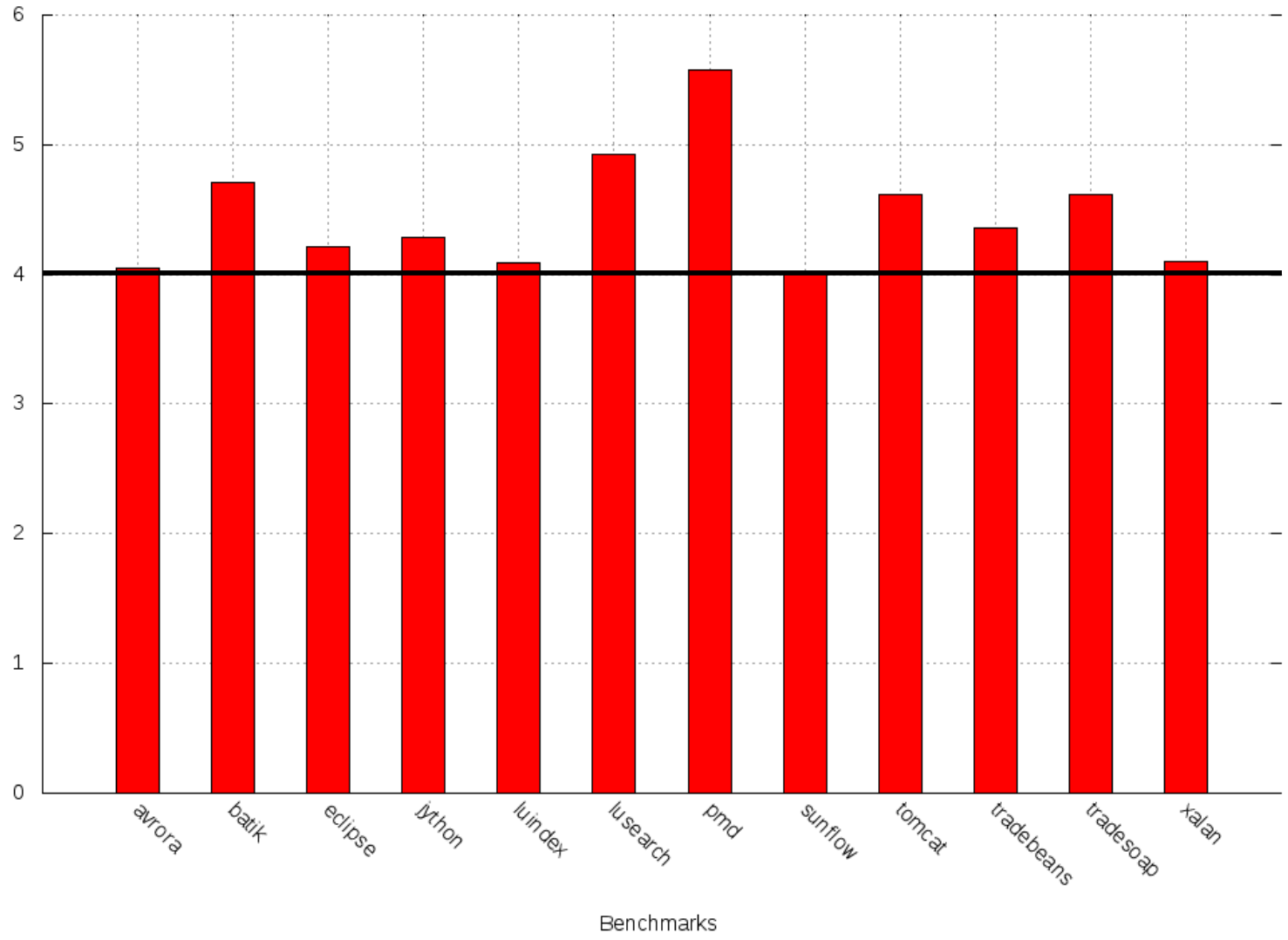
```
// pseudo code!!!
Class<?> c1 = new Class("Object");
Class<?> c2 = new Class("String");
// class objects contain static
// fields as dynamic fields
```

*VM knows whether the class determines the size!*

# Surviving Objects

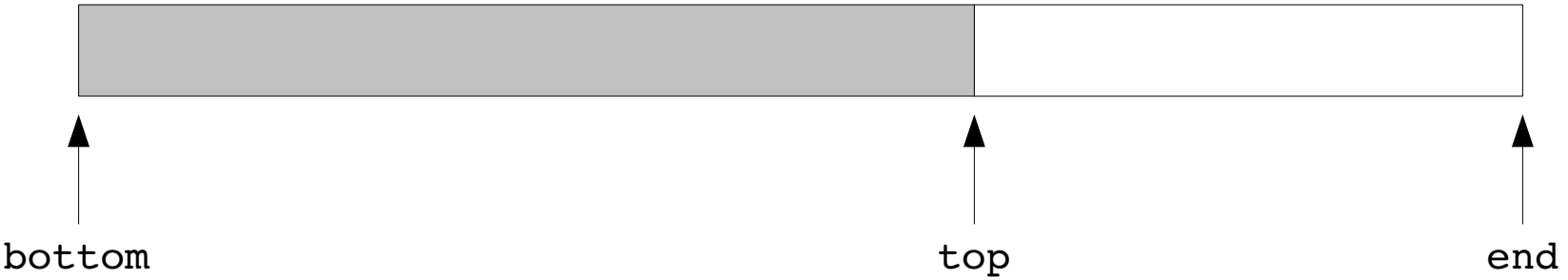


# Average Event Size [bytes]



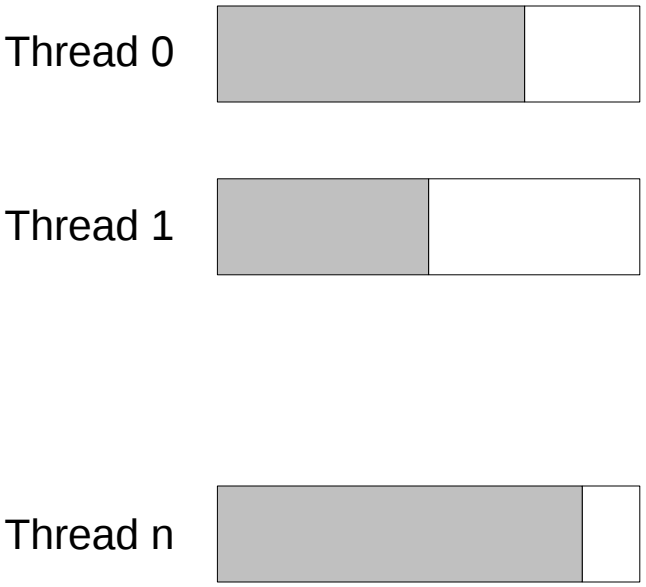
# Global Buffering

```
*(top++) = 0xABCDEF00;
```

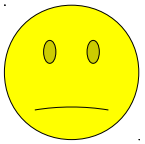



```
lock global_buffer
if(top >= end) {
  flush_buffer();
  top = bottom;
}
*(top++) = 0xABCDEF00;
unlock global_buffer
```

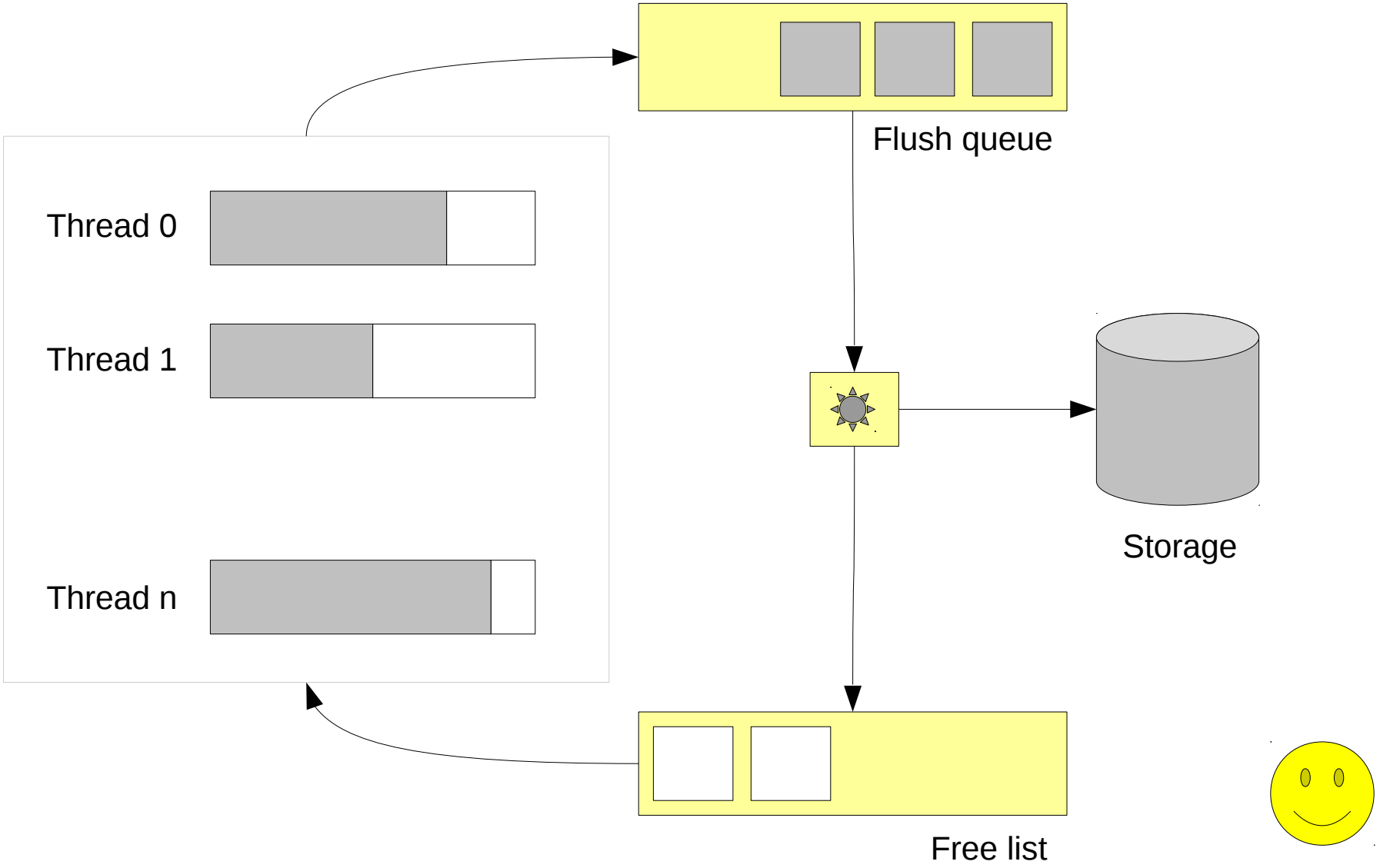
# Thread-local Buffering




```
lock buffer
if(top >= end) {
  flush_buffer(); //major pause
  top = bottom;
}
*(top++) = 0xABCDEF00;
unlock buffer
```



# Thread-local Buffering & Management



# Q&A

JOHANNES KEPLER
UNIVERSITÄT LINZ

?