

# Android Memory Usage - revealed with the Eclipse Memory Analyzer

Markus Kohler // 23th February 2010 // [kohlerm.blogspot.com](http://kohlerm.blogspot.com)

# 1.Introduction

## 2.Fundamentals

## 3.Advanced

## 4.Summary

Why should you  
optimize memory  
usage?



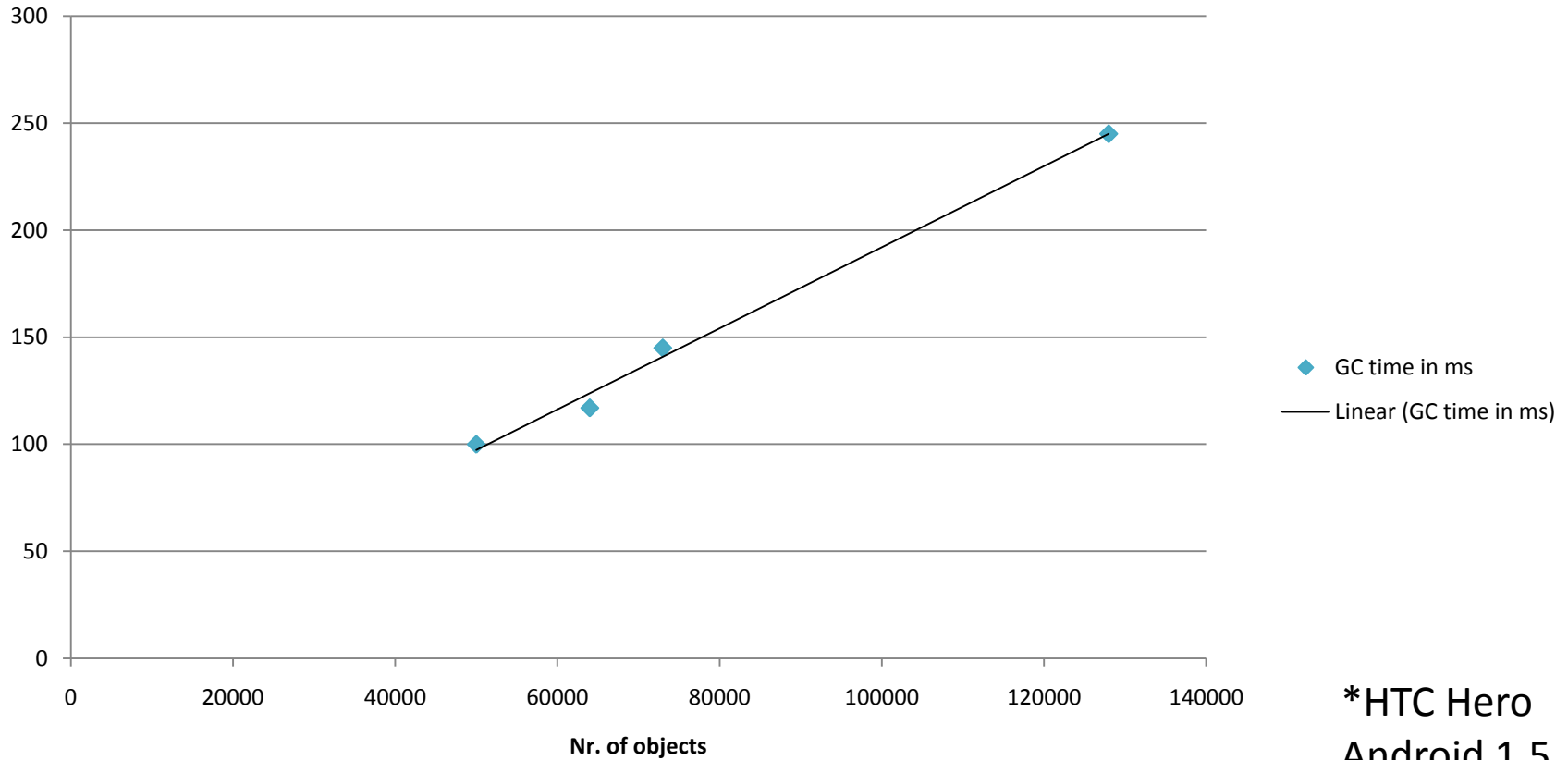
Bernat Casero - [www.bernatcasero.com](http://www.bernatcasero.com)



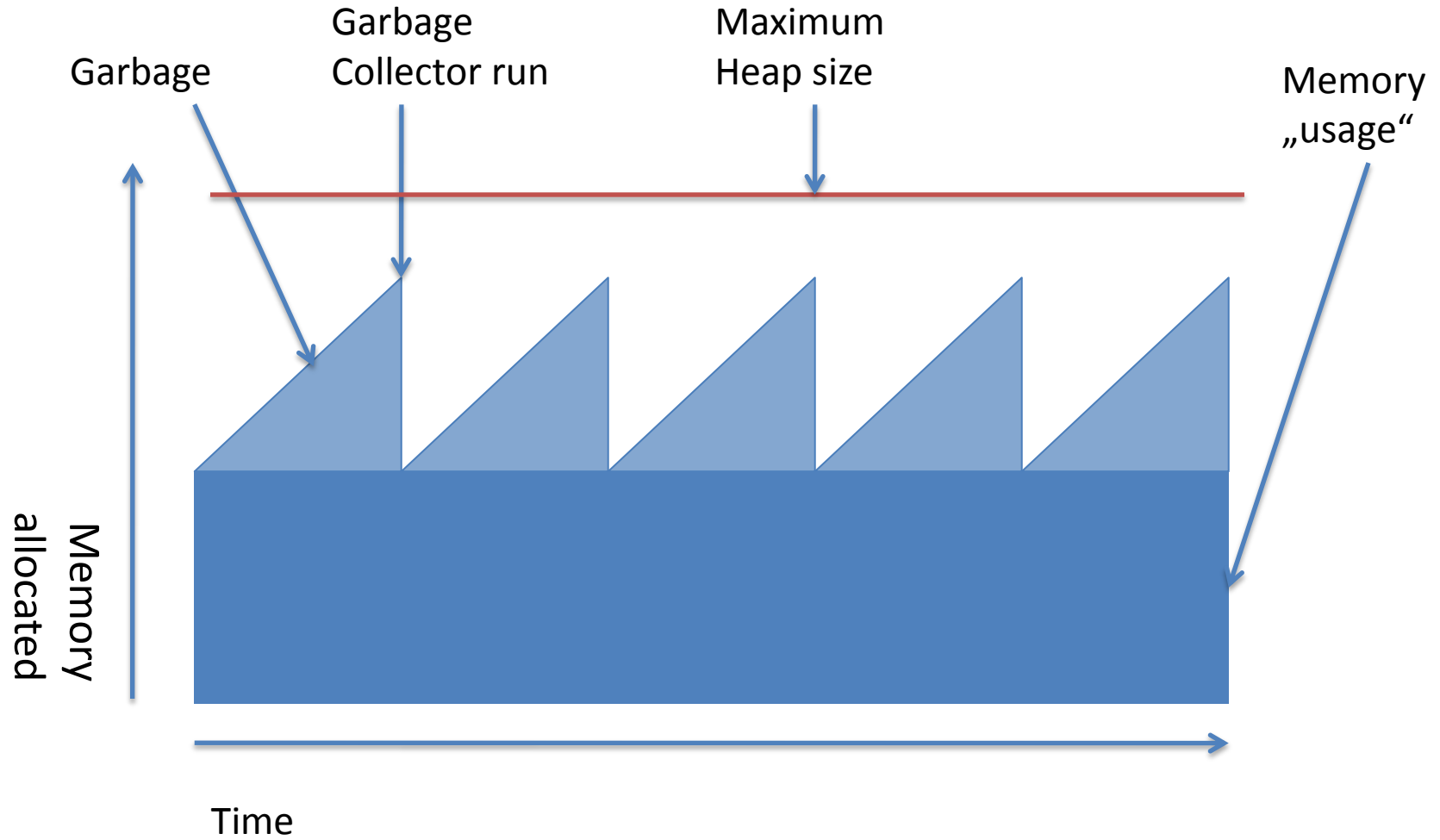
<http://www.flickr.com/photos/ytwhitelight/504212595/>

# Performance of the Garbage collector

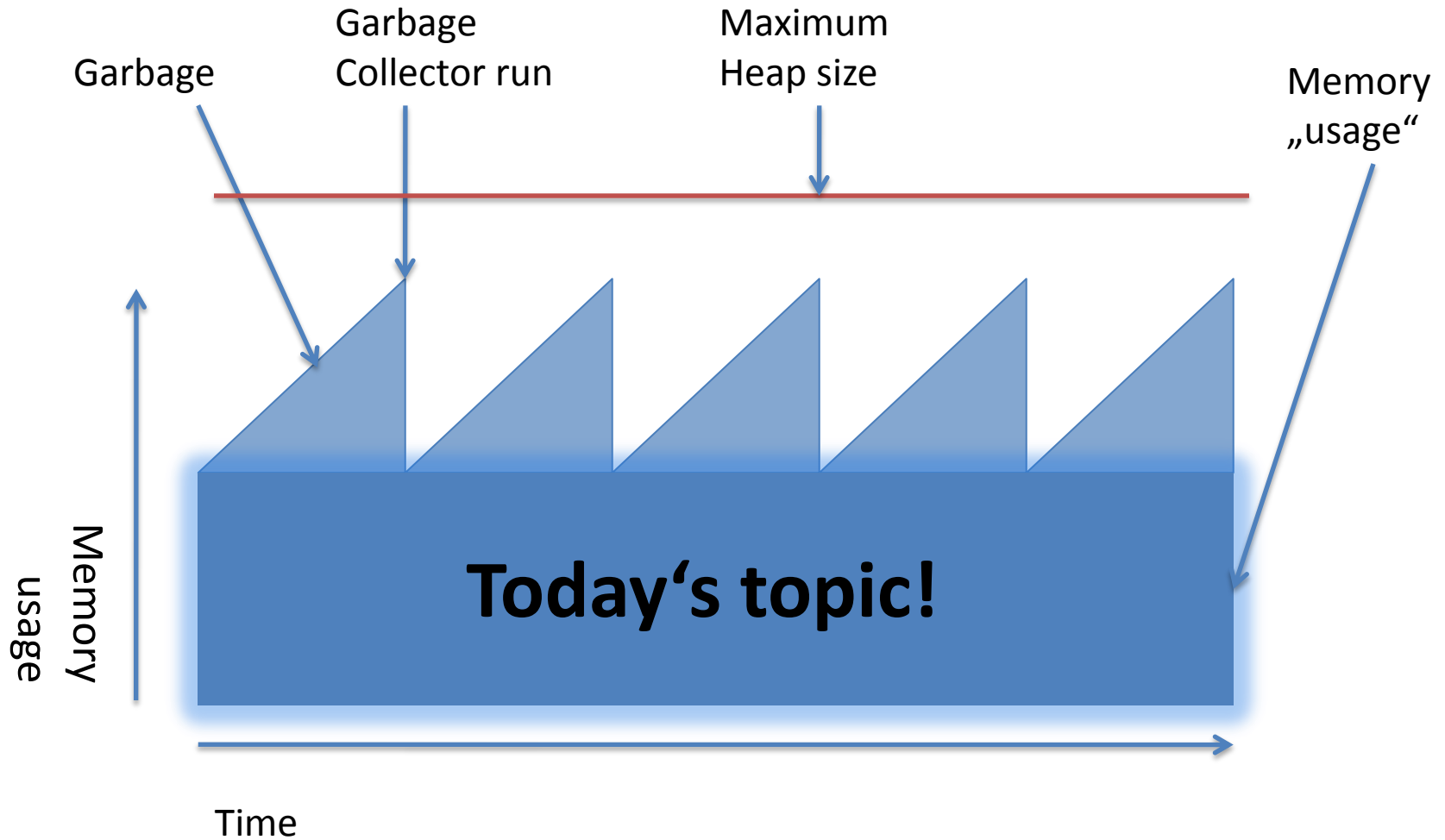
Garbage collection time in ms\*



# Typical memory usage pattern



# Typical memory usage pattern



# Heap Dump

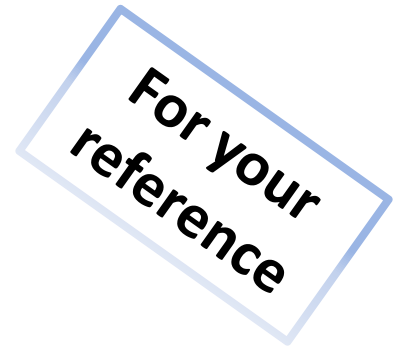
A heap dump contains

- \* A snapshot of all “living” objects at a given point in time
- \* Information about which objects are Garbage Collector roots
  - \_ GC roots will not be reclaimed even when there’s no reference to them

A heap Dump does **not** contain

- \* Information, where in the code objects have been created or which objects were reclaimed by the GC
- \* Information about “native” objects such as Bitmaps, but those are still counted for the 16 Mbyte limit on android

# Dalvik Heap dumps



On Android <2.0

\* Make sure /data/misc is writable:

\_ adb shell

\_ chmod 777 /data/misc

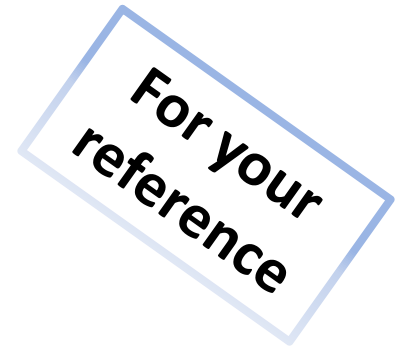
\* Create a heap dump by sending the process a SIGUSR1 (-10) signal

\_ adb shell ps

\* Look for the pid and create the heap dump

\_ adb shell kill -10 <pid>

# Dalvik Heap dumps



On Android <2.0

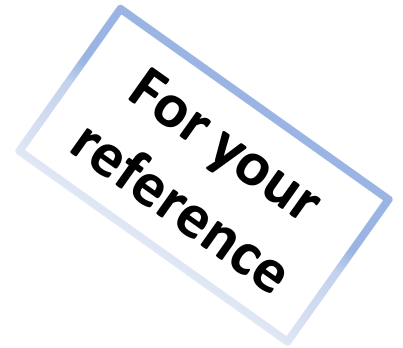
\* Check with whether it worked

\_ adb shell logcat

\* You should find something like

\_ dalvikvm( ): threadid=7: reacting to signal 10 I/dalvikvm( ): SIGUSR1  
forcing GC and HPROF dump I/dalvikvm( ): hprof: dumping VM heap to  
"/data/misc/ ...

# Dalvik Heap dumps



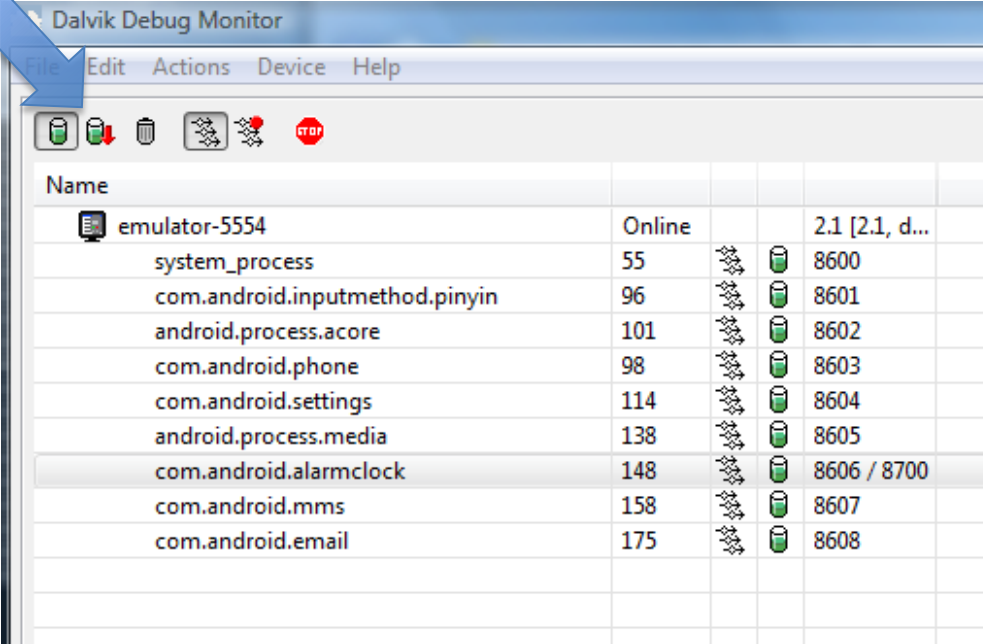
On Android <2.0

- \* copy the heap dump from the phone to the Desktop machine
  - \_ `adb pull /data/misc/heap-dump-tm-pid.hprof mydump.hprof`
- \* Convert from the Dalvik heap dumpformat to hprof:
  - \_ `hprof-conv heap-dump-tm-pid.hprof 4mat.hprof`

# Dalvik Heap dumps

For your  
reference

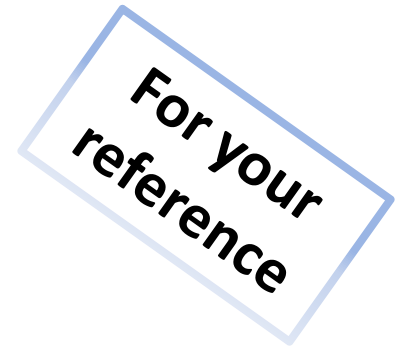
On Android  $\geq 2.0$



The screenshot shows the Dalvik Debug Monitor interface. A blue arrow points to the 'File' menu. The interface displays a list of processes with columns for Name, PID, and memory usage. The 'com.android.alarmclock' process is highlighted.

Name	PID	Memory Usage
emulator-5554	Online	2.1 [2.1, d...]
system_process	55	8600
com.android.inputmethod.pinyin	96	8601
android.process.acore	101	8602
com.android.phone	98	8603
com.android.settings	114	8604
android.process.media	138	8605
com.android.alarmclock	148	8606 / 8700
com.android.mms	158	8607
com.android.email	175	8608

# Dalvik Heap dumps



- \* On Android  $\geq 2.0$ 
  - \_ Create heap dumps from Dalvik debug monitor (you still need to call the converter tool)
  - \_ Lightweight integration of the Android SDK and the Eclipse Memory Analyzer
    - < Heap dumps can be opened directly in the Eclipse based Android SDK on the condition that the Memory Analyzer plugings are installed
    - < Conversion is done automatically



# Tools for analysing heap dumps

**Yourkit**

**Eclipse Memory Analyzer**

**JHat**

**Netbeans**

**Jprofiler**

**Hat**

# Eclipse Memory Analyzer

# Eclipse Memory Analyzer?

# SAP NETWEAVER

# SAP NETWEAVER

- \* ~2.750 employees
- \* ~1.500 Java developers
- \* ~24 Million Lines of Code
- \* ~255k Classes
- \* Applications using Gigabytes of memory
- \* [http://jax.de/konferenzen/jax08/sessionupdate/harald\\_mueller-keynote.pdf](http://jax.de/konferenzen/jax08/sessionupdate/harald_mueller-keynote.pdf)

# SAP NETWEAVER



## Performance

**Free**

# 2006

# SAP Memory Analyzer

Open Source

# 2007

## Eclipse Memory Analyzer

<http://www.eclipse.org/mat/>



# 1. Introduction

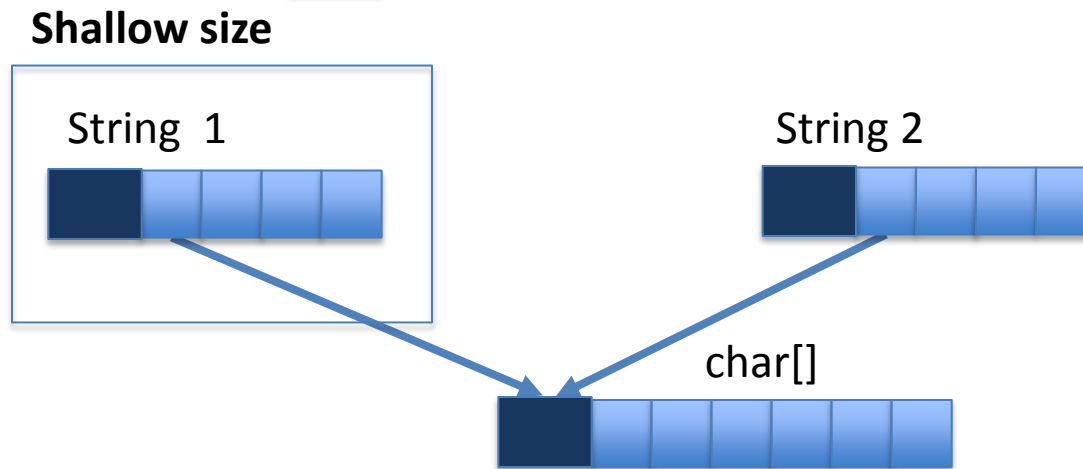
## 2. Fundamentals

### 3. Advanced

### 4. Summary

# Memory Usage

- \* **Shallow size** of an object is the amount of memory allocated to store the object itself, not taking into account the referenced objects == Sum of the size of the “object header”  and the fields  of an object



# Memory usage

Shallow Size of a String object

JDK source code **String** class:

```
public final class String {8 Bytes header  
private char value[]; 4 Bytes  
private int offset; 4 Bytes  
private int count; 4 Bytes  
private int hash = 0; 4 Bytes  
...}
```

**„Shallow size“ of a String ==24 Bytes  
(32 bit Sun JDK)**

# Memory usage, shallow Size

Class name	Nr. of Objects	Shallow size in bytes
char[]	1.696.360	171.641.928
com.erp.Order	1.723	110.273
java.lang.String	1.652.780	66.111.200

How many String instances are caused by Order instances?

How many char[] are caused by String instances?

Shallow size often does not help you to answer these questions, and is in practice often useless for memory usage analysis

Can we do  
better?



<http://www.flickr.com/photos/joecrimmings/2238699461/sizes/l/>

# Memory usage, retained set

How can we really measure memory usage?

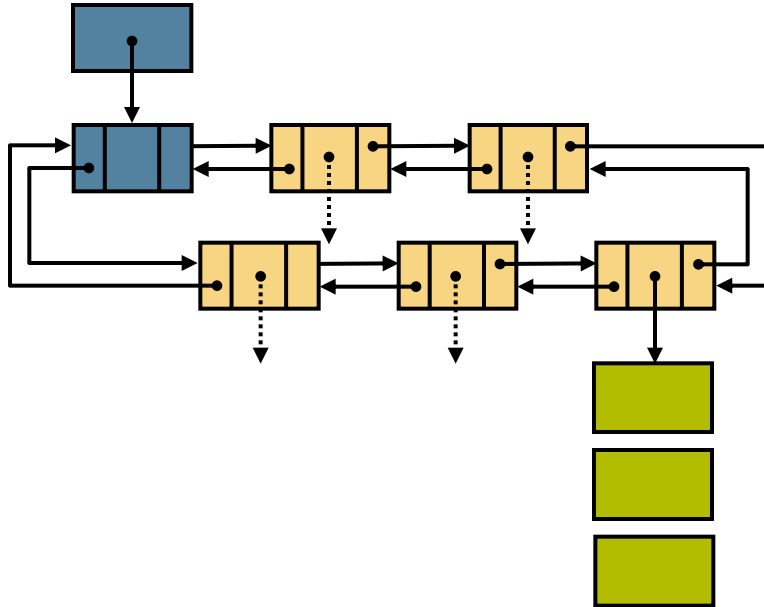
- \* Retained set of X

- \_ The set of objects that would be reclaimed, if we could delete Object X

- \* Retained size of X

- \_ The Retained size of an object X is equal to the shallow size of the „Retained sets“ of X

# Retained set



`LinkedList`

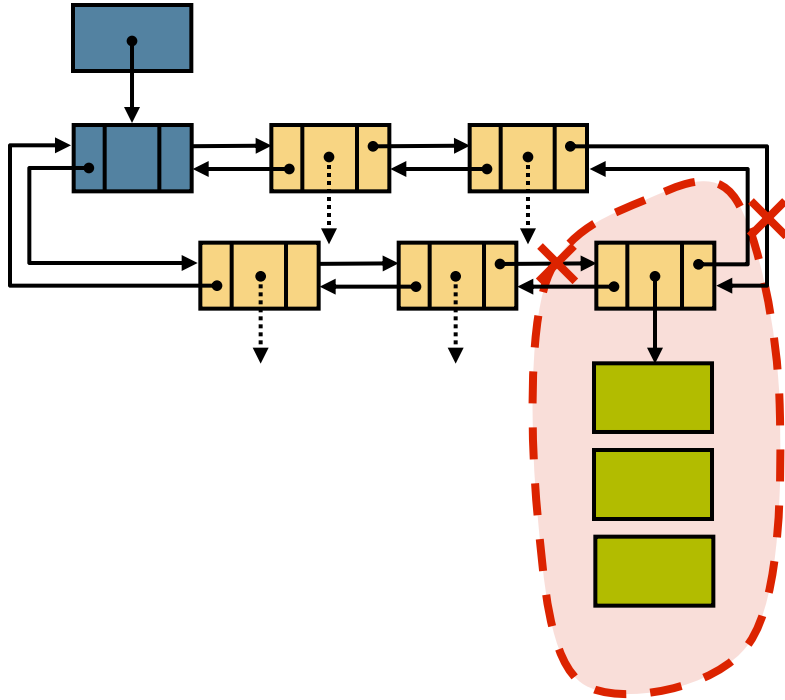
`LinkedList$Entry`

`Entries`

`String`

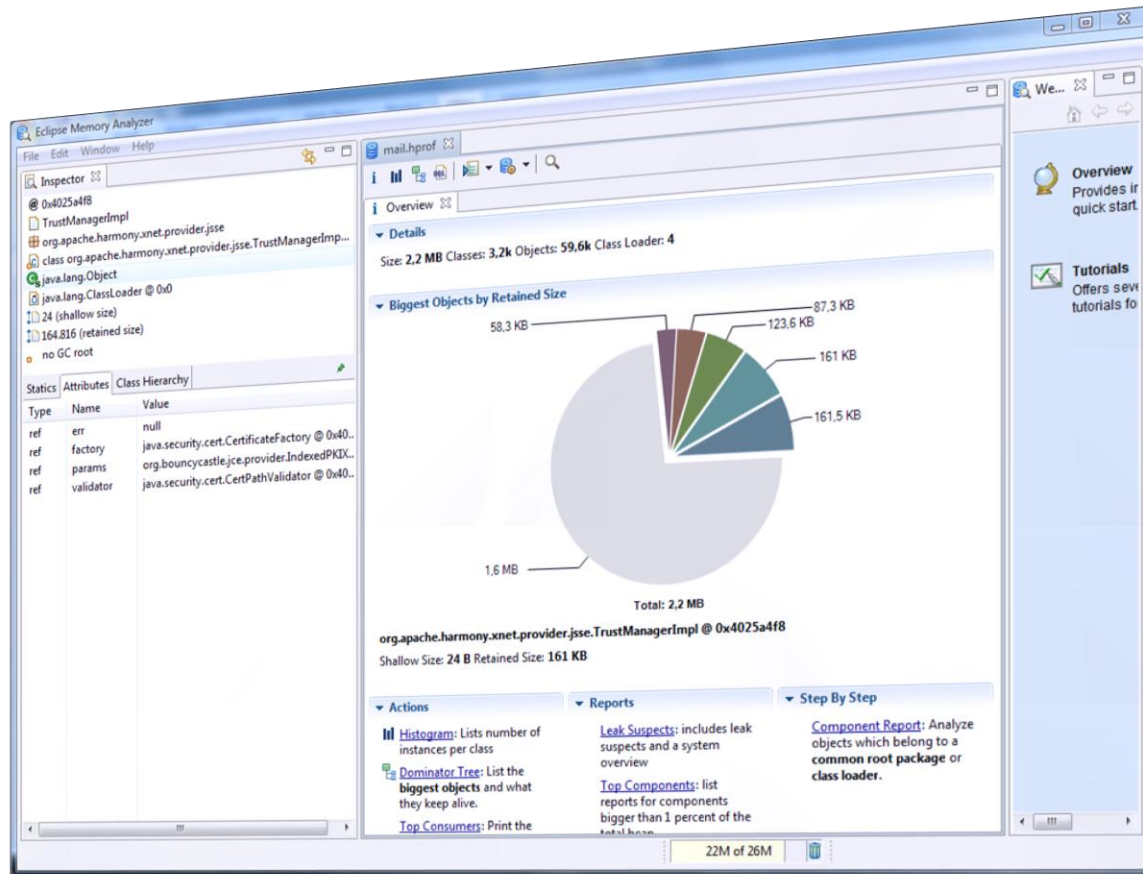
`char[]`

# Retained Size via GC Simulation



1. Delete X from memory
2. Mark all objects reachable from GC Roots
3. The “retained Set” of X is the set of objects not yet marked

# DEMO Memory Analyzer



1.Introduction

2.Fundamentals

3.Advanced

4.Summary

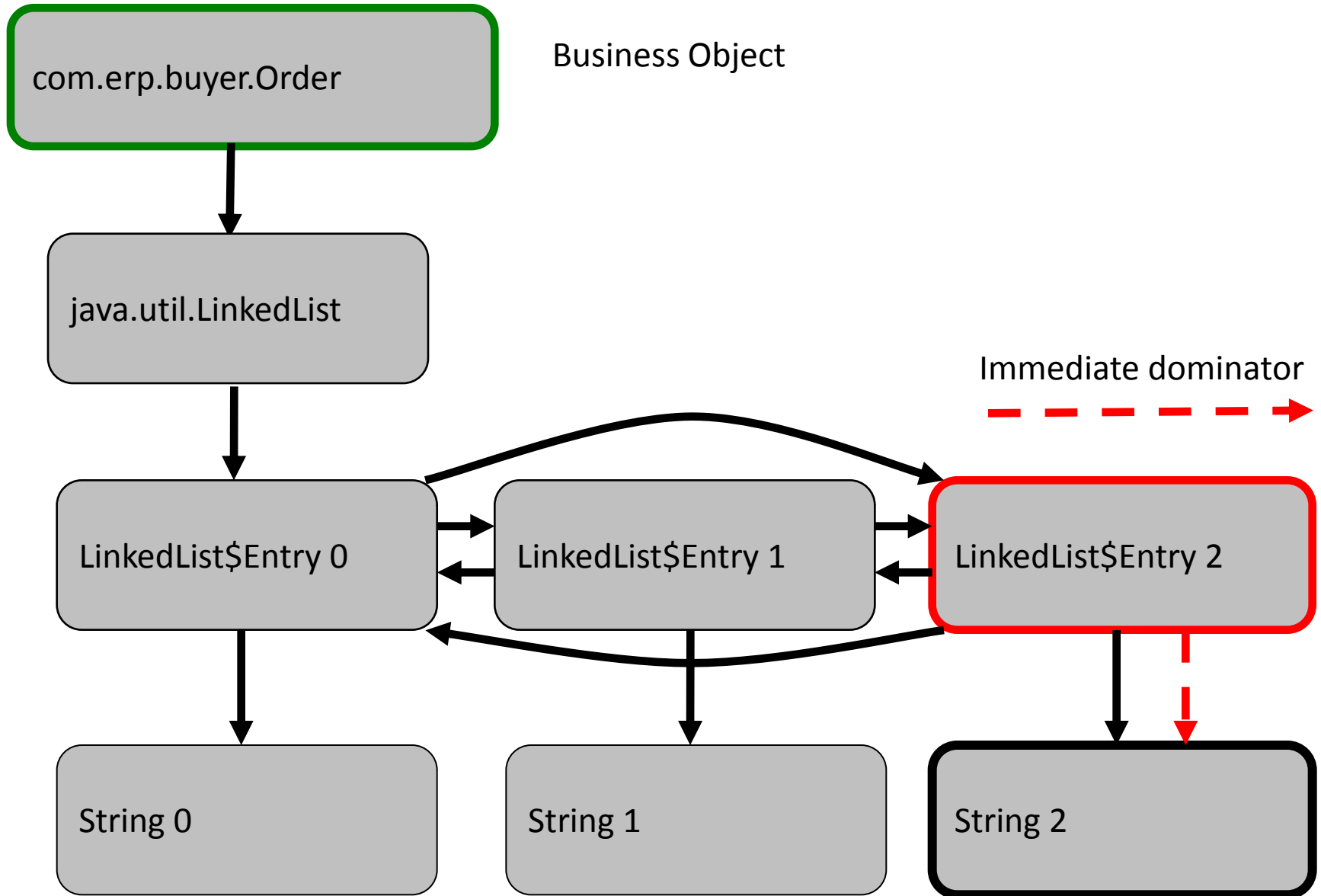
# Eclipse Memory Analyzer, advanced features

## \* Dominator Tree

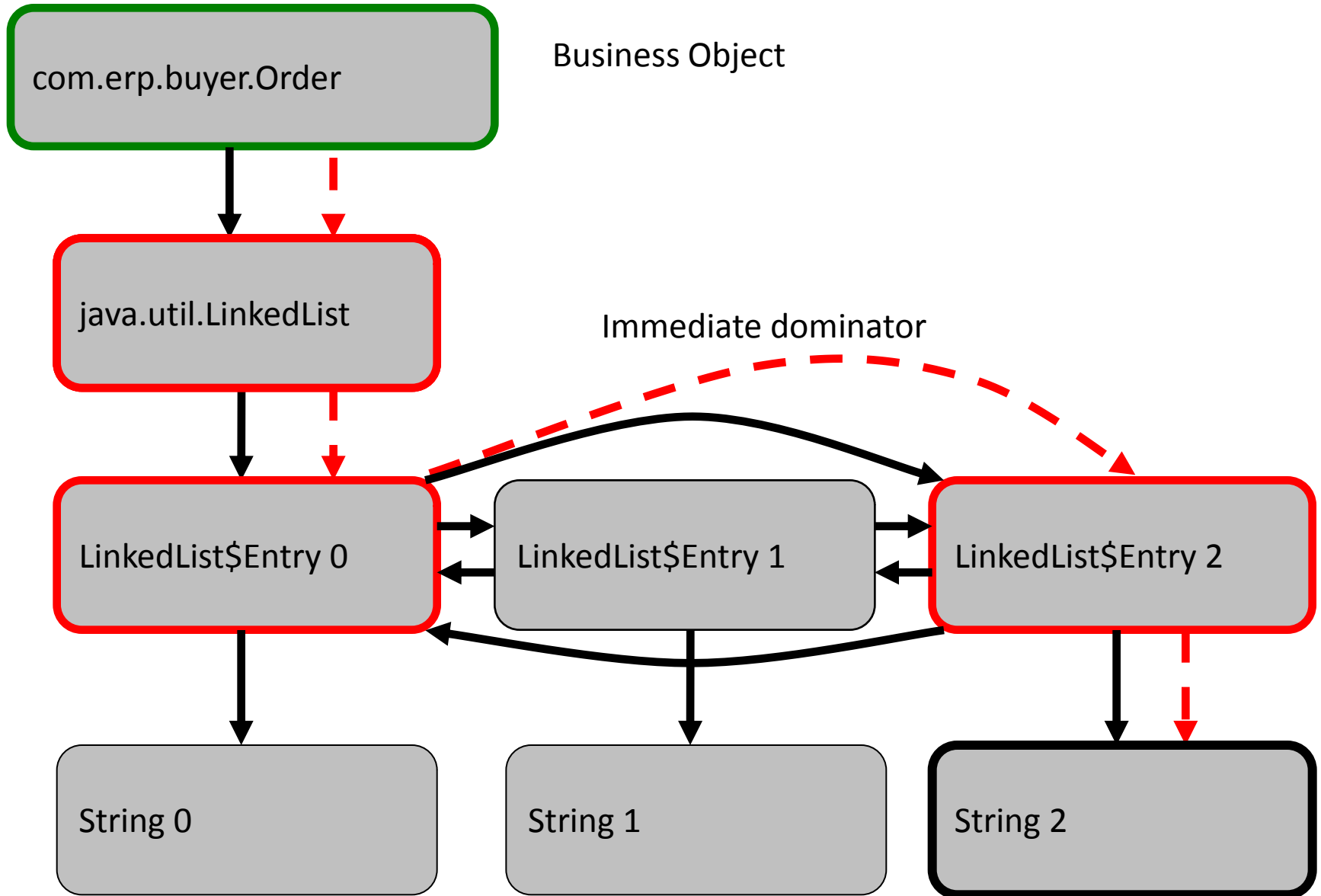
([http://en.wikipedia.org/wiki/Dominator\\_\(node\)](http://en.wikipedia.org/wiki/Dominator_(node)))

- \_ Unique feature of the Memory Analyzer
  - < find the (single) Objects, with the biggest retained size
  - < Can be used to quickly compute a good lower bound for the “retained Size” of objects
- \_ X dominates Y, if all paths from a GC Root to Y go via X. “X is a Dominator of Y”
- \_ The closest Dominator to Y is called “immediate Dominator”

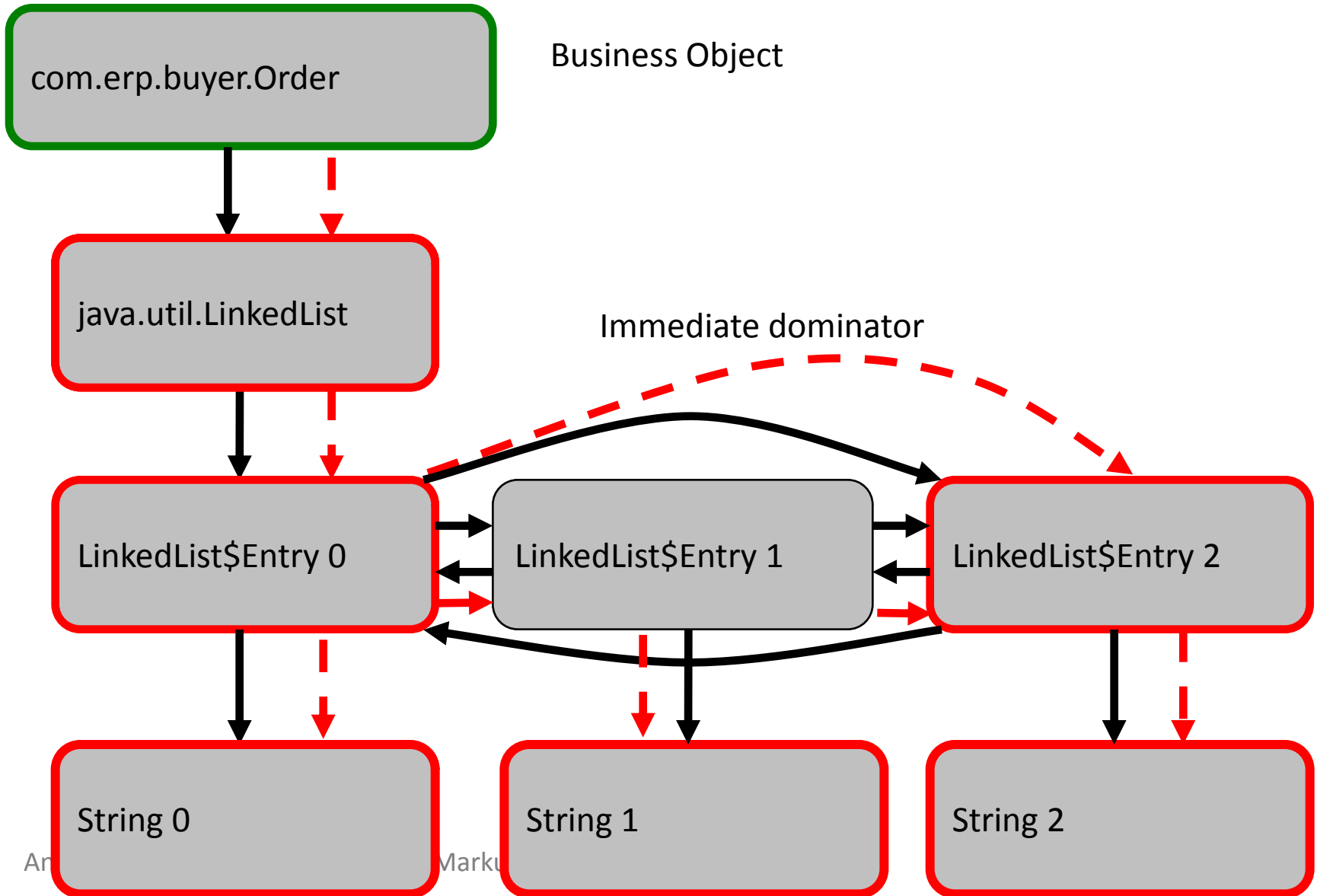
# Immediate Dominator



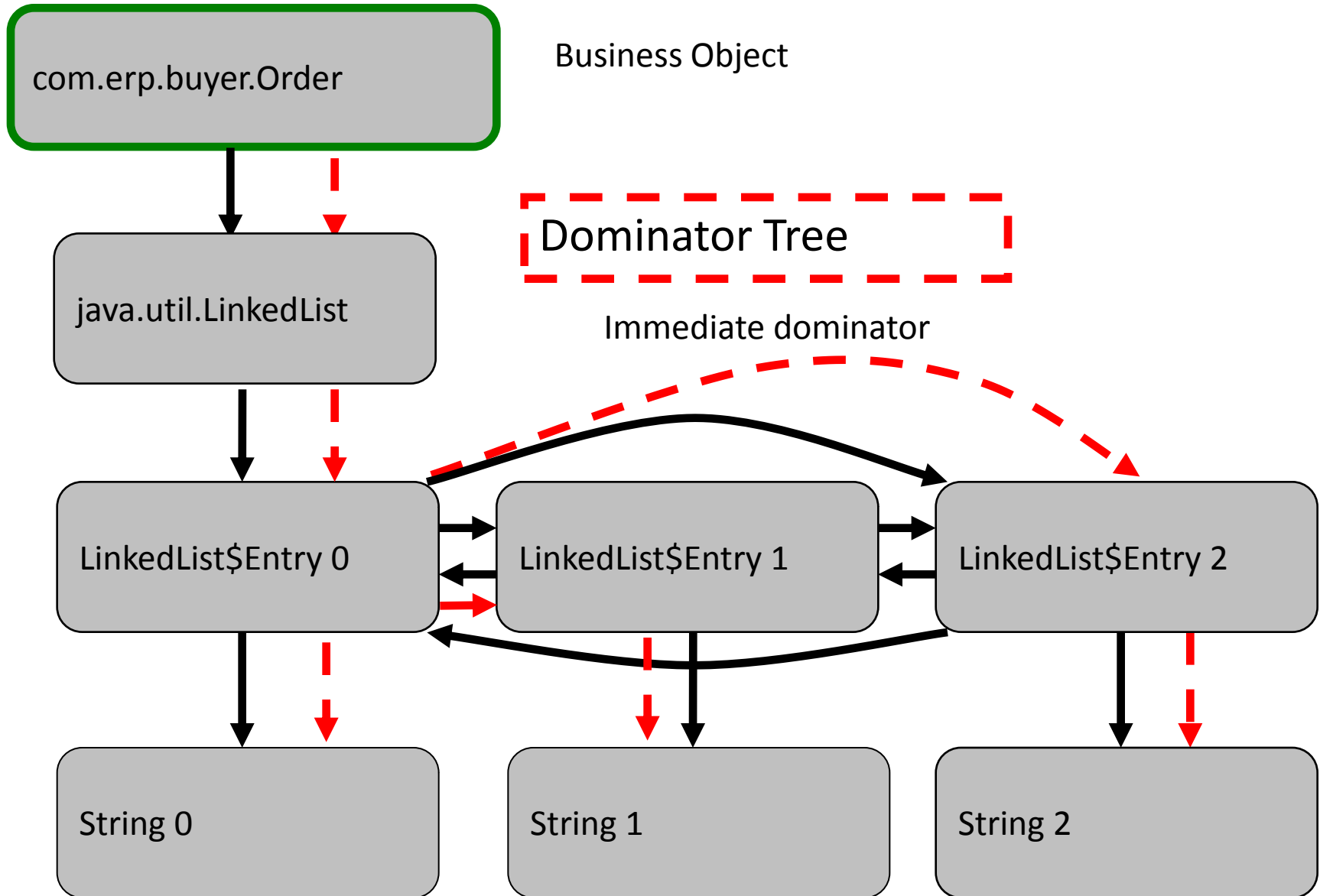
# Immediate Dominator



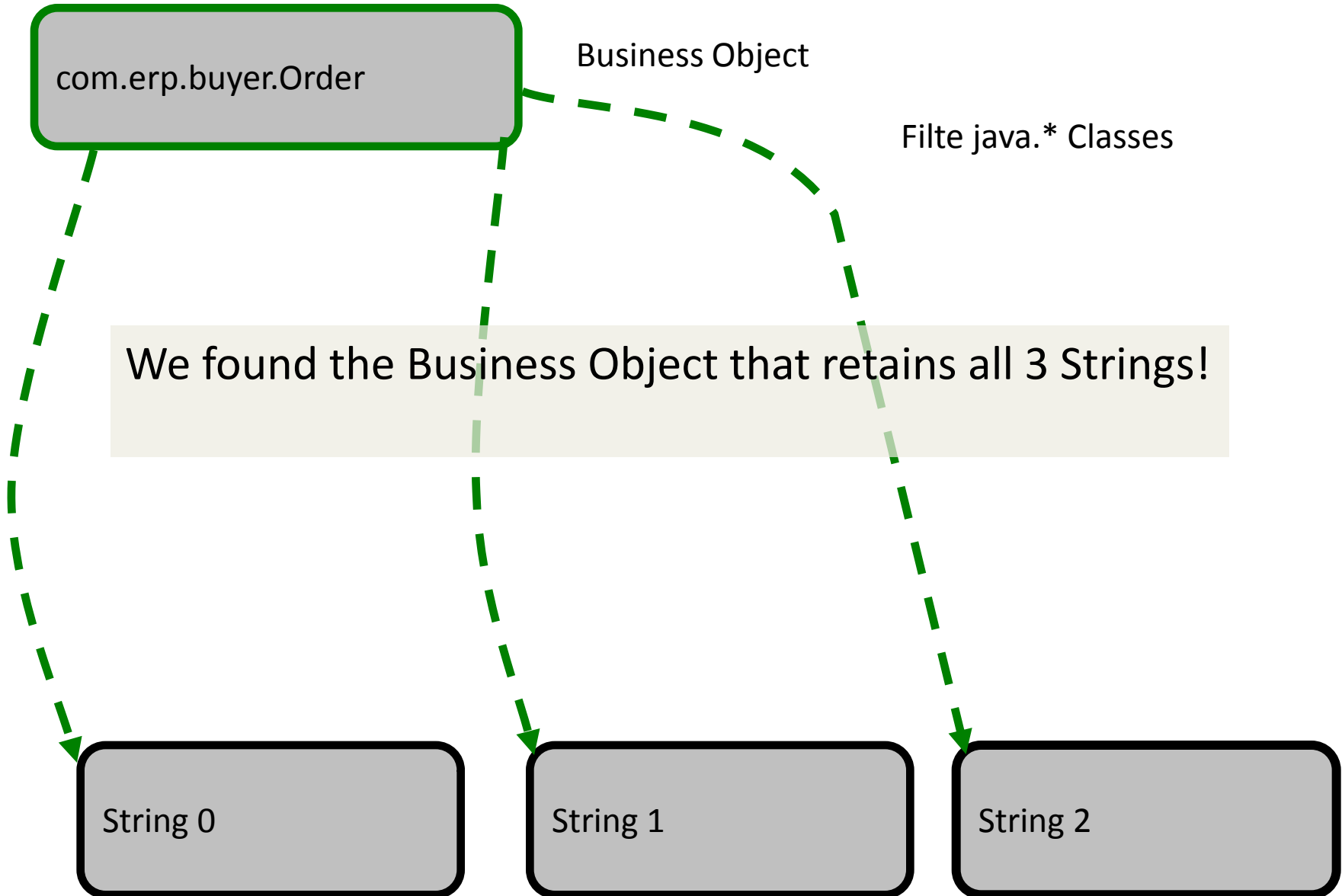
# Immediate Dominator



# Immediate Dominator



# Immediate Dominator filtered





1.Introduction

2.Fundamentals

3.Advanced

4.Summary

# Summary

- \* Memory usage analysis is a very important topic for high performance android applications!
- \* Using the Eclipse memory analyzer it's relatively easy to identify issues
- \* Currently there are certain limitations on the android platform
  - \_ Shared data cannot be identified
  - \_ Native objects are not shown
  - \_ Special commands for Android are missing

**Romain Guy (Google):** “I would definitely love to see better **#mat** support, it would make my life much much easier.  
Stupid **#hat**”

THE  
END.