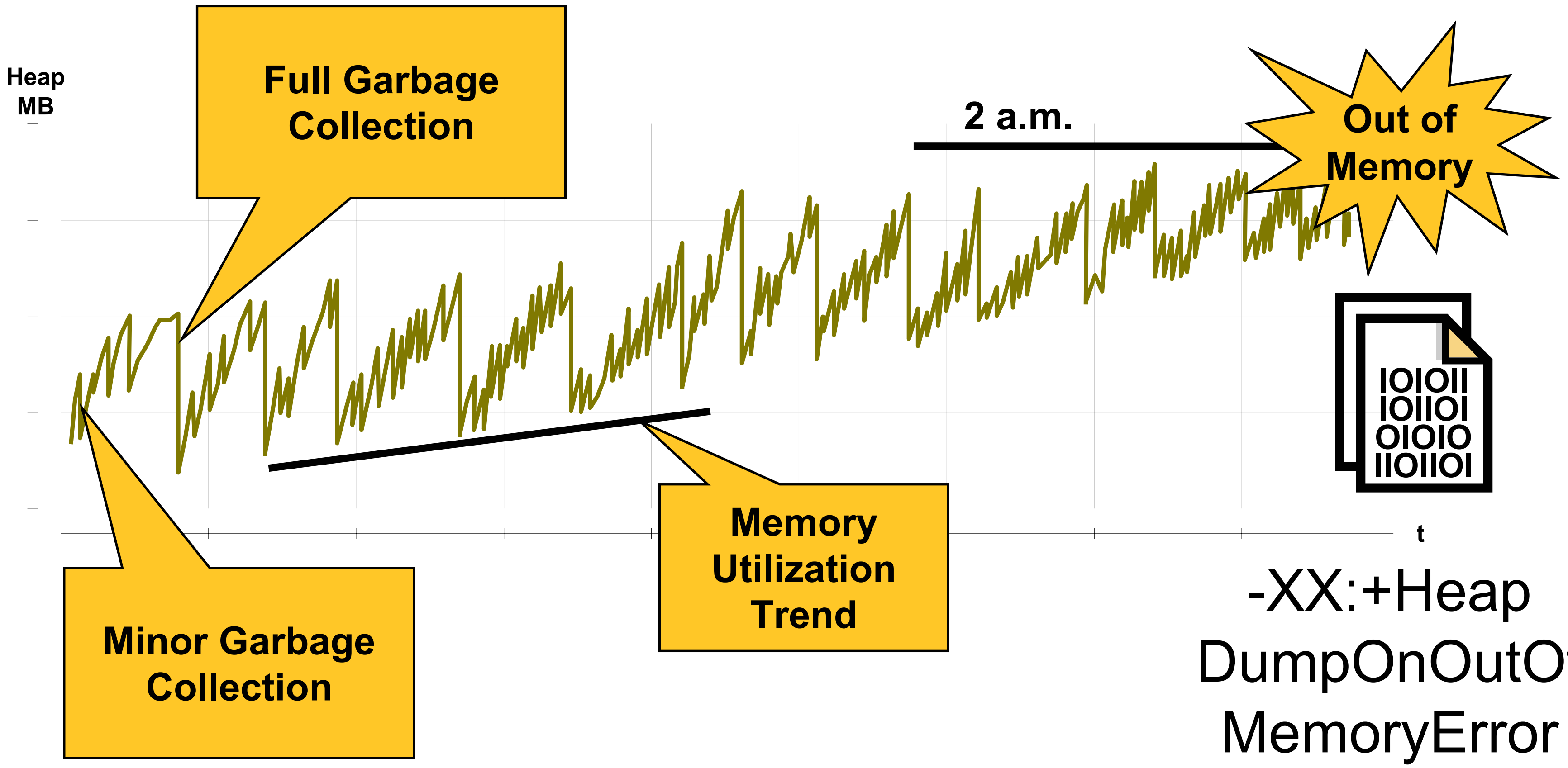
Improving the Quality of Your Enterprise Application: Innovative Ways to Spot Memory-Related Bugs and Bottlenecks

Vedran Lerenc, Andreas Buchen

Programmers
SAP AG
http://www.sap.com

TS-21935

Memory 101



-XX:+HeapDumpOnOutOfMemoryError

Goal

Learn which valuable information can be extracted from an HPROF binary heap dump.

Question

How many objects do you find in big production heap dumps?

a) ~ 1.000.000

b) ~ 10.000.000

c) ~ 100.000.000

Agenda

Memory 101

Analysis Techniques

Pathogen Memory

Lessons Learned

Q&A

Agenda

Memory 101

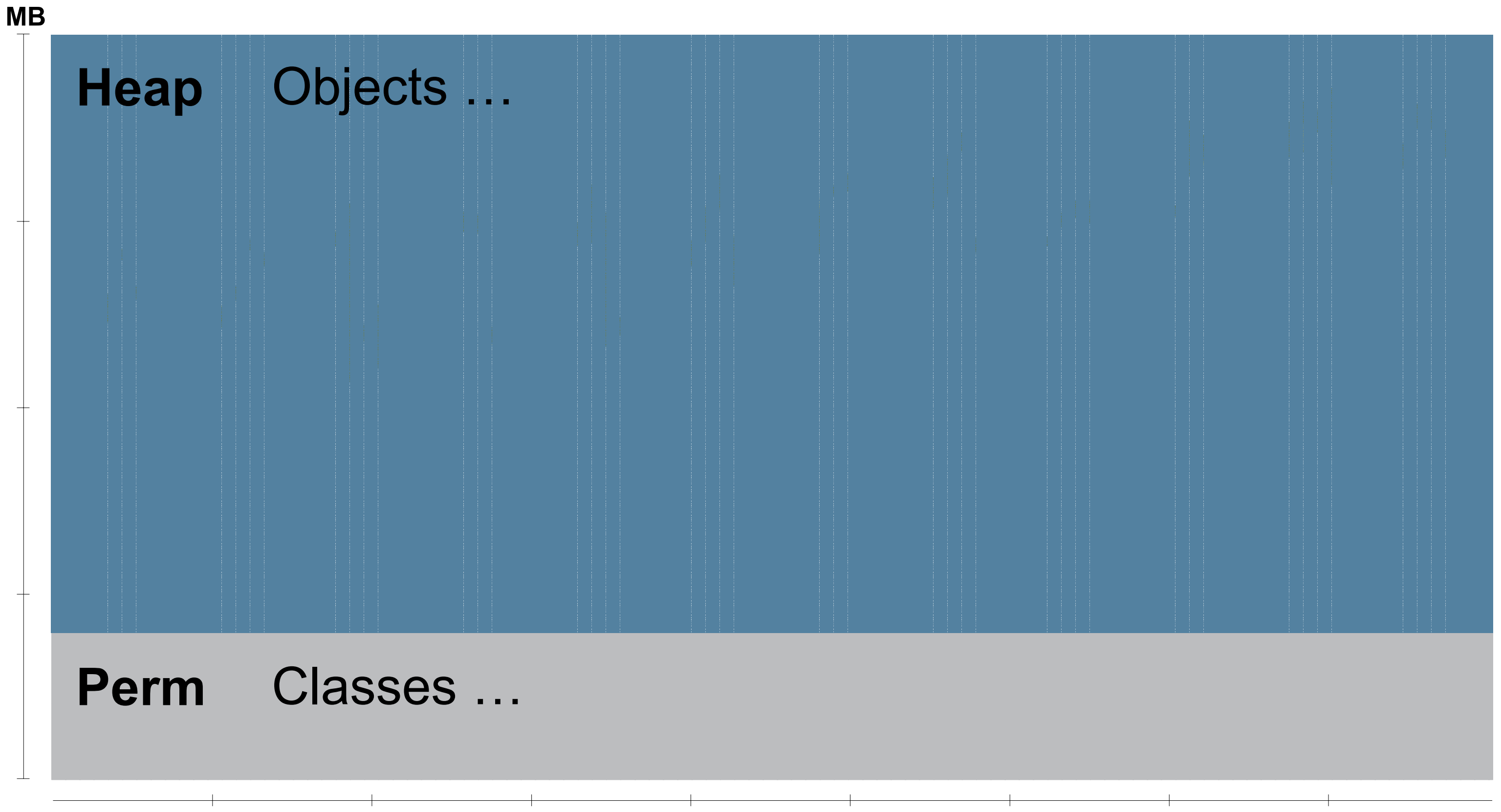
Analysis Techniques

Pathogen Memory

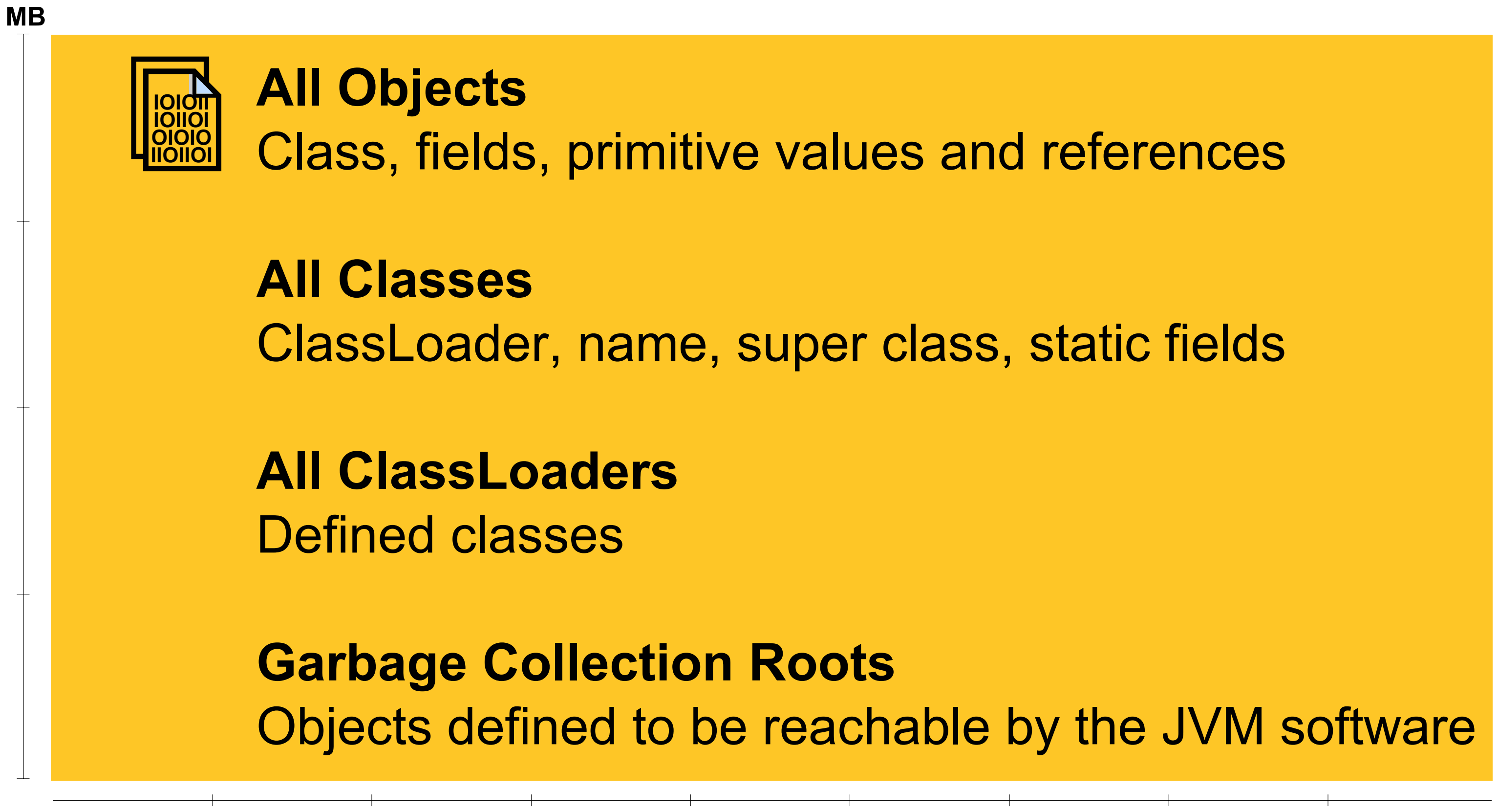
Lessons Learned

Q&A

Types of Memory



Heap Dump Content



All Objects
Class, fields, primitive values and references

All Classes
ClassLoader, name, super class, static fields

All ClassLoaders
Defined classes

Garbage Collection Roots
Objects defined to be reachable by the JVM software

JVM = Java™ Virtual Machine (JVM™)
The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java™ platform.

HPROF Binary Heap Dump

MB

A heap dump contains a **snapshot of objects that are alive** at one point in time.

A full GC is triggered before the heap dump is written.

A heap dump cannot **not** answer

- who and where objects have been created.
- which objects have been garbage collected.

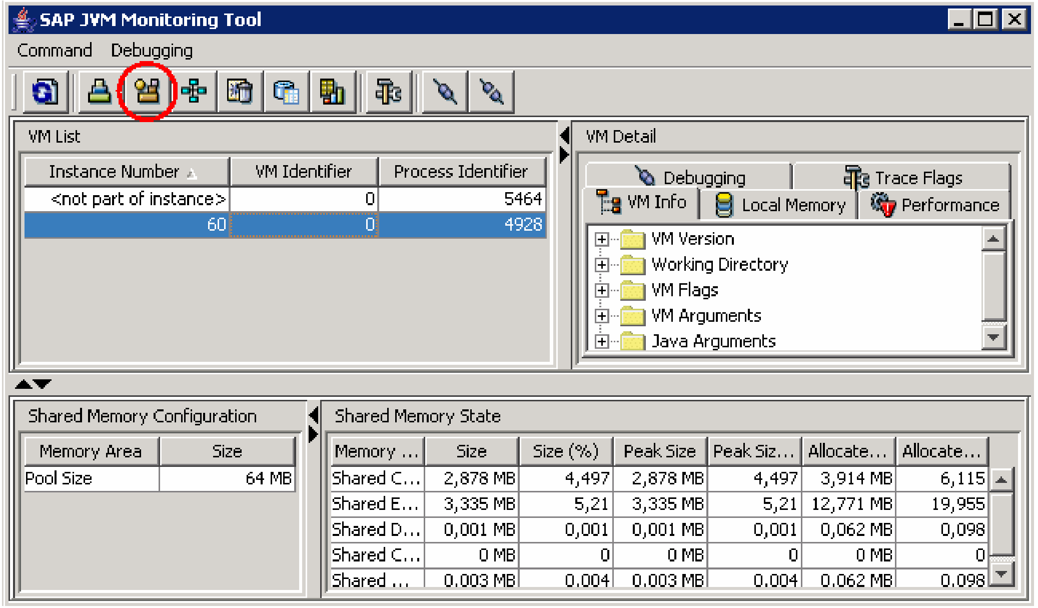
How to Acquire a Heap Dump

- Available in 1.4.2_12 and 5.0_ 7 and 6.0 upwards
  - -XX:+HeapDumpOnOutOfMemoryError
- Alternatives to get it on demand
  - -XX:+HeapDumpOnCtrlBreak
  - jmap -dump:format=b,file=<filename.hprof> <pid>
  - JConsole
  - SAP Memory Analyzer / JVMMON / (MMC)
  - …

For Your Reference

How to Get a Heap Dump via SAP JVMMON

1. **Run JVMMON from bin directory, e.g.,**
<system>/SYS/exe/run/sapjvm_5\bin\jvmmon -gui



Instance Number	VM Identifier	Process Identifier
<not part of instance>	0	5464
60	0	4928

Memory Area	Size
Pool Size	64 MB

Memory ...	Size	Size (%)	Peak Size	Peak Siz...	Allocate...	Allocate...
Shared C...	2,878 MB	4,497	2,878 MB	4,497	3,914 MB	6,115
Shared E...	3,335 MB	5,21	3,335 MB	5,21	12,771 MB	19,955
Shared D...	0,001 MB	0,001	0,001 MB	0,001	0,062 MB	0,098
Shared C...	0 MB	0	0 MB	0	0 MB	0
Shared ...	0,003 MB	0,004	0,003 MB	0,004	0,062 MB	0,098

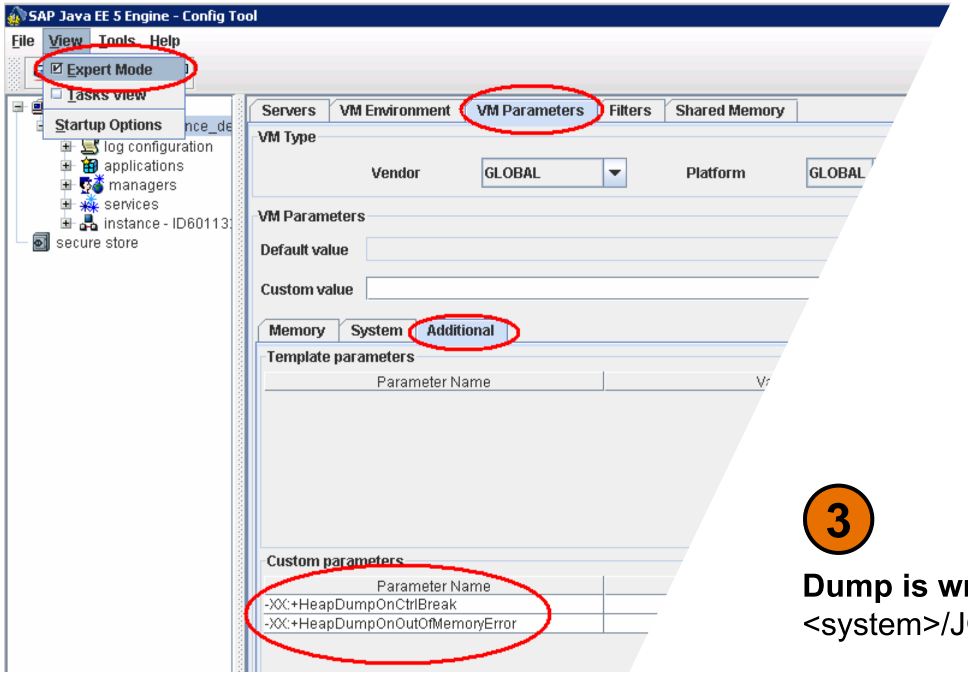
2. **Dump is written to the current working directory of the VM, e.g.,**
<system>/JC<instance>/j2ee/cluster/server<node>/java_pid<pid>.hprof

For Your Reference

How to Get a Heap Dump via SAP MMC

1 **Configure your server using the Java™ Platform, Enterprise Edition (Java™ EE platform) Config Tool:**
<system>/JC<instance>/j2ee/configtool/configtool

-XX:+HeapDumpOnOutOfMemoryError
-XX:+HeapDumpOnCtrlBreak



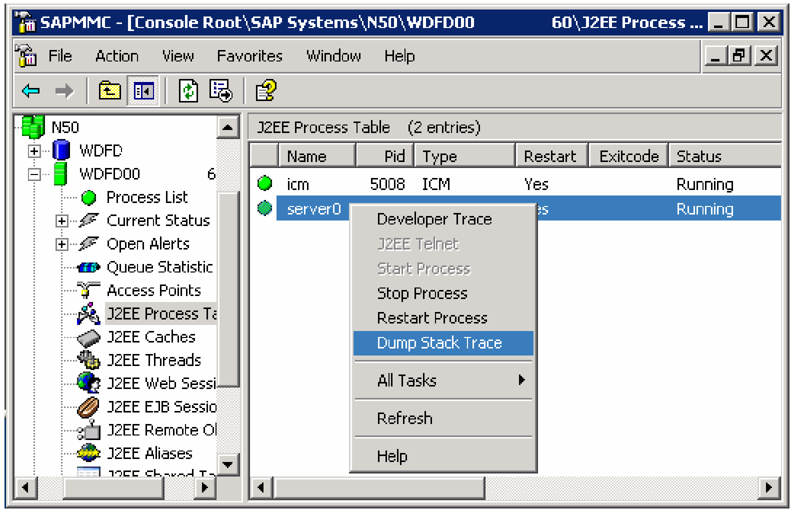
Vendor: GLOBAL
Platform: GLOBAL

VM Parameters
Default value
Custom value

Custom parameters

Parameter Name
-XX:+HeapDumpOnCtrlBreak
-XX:+HeapDumpOnOutOfMemoryError

2 **Select "Dump Stack Trace" in SAPMMC:**



J2EE Process Table (2 entries)

Name	Pid	Type	Restart	Exitcode	Status
icm	5008	ICM	Yes		Running
server0			es		Running

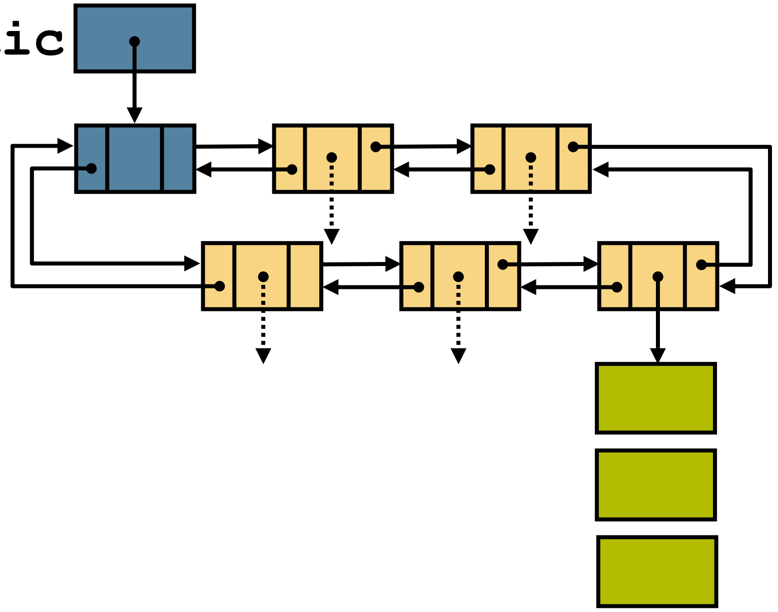
3 **Dump is written to:**
<system>/JC<instance>/j2ee/cluster/server<node>/java_pid<pid>.hprof

Agenda

- Memory 101
- **Analysis Techniques**
  - Retained Size
  - Dominator Tree
  - Grouping Anywhere
- Pathogen Memory
- Lessons Learned
- Q&A

Memory 101—Retained Size

```
class X
{
static
}
```

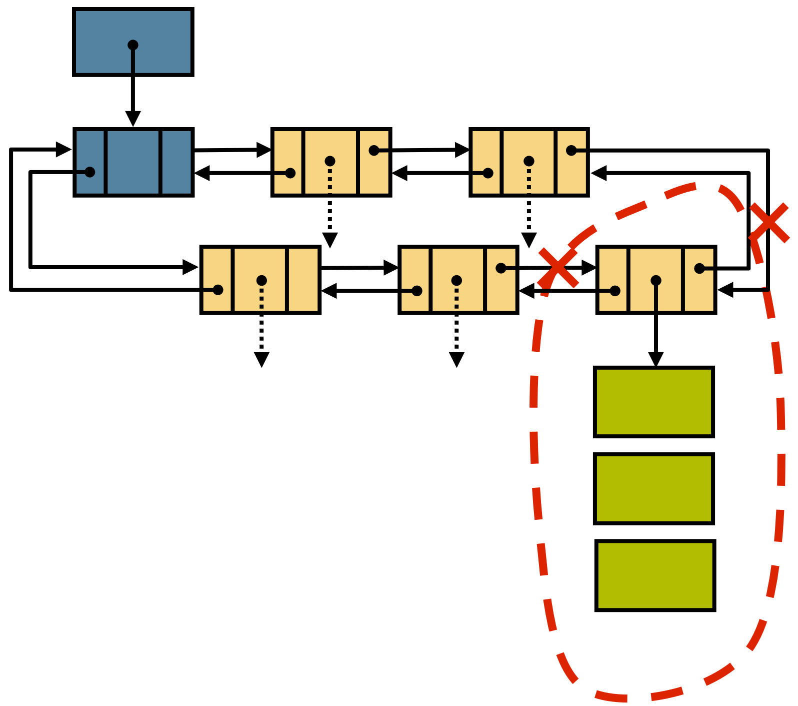


```
LinkedList

LinkedList$Entry

SomeEntry
String
char[]
```

Determine Retained Size via GC Simulation



1. Remove all references to object X.
2. Mark all objects which are still reachable from the GC Roots.
3. The unmarked objects constitute the retained set of object X.

For Your Reference

Shallow and Retained Size

- Shallow heap is the memory consumed by one object; Java HotSpot™ virtual machines (VMs) need 32 or 64 bits per object handle (depending on the machine architecture), 4 bytes per Integer, 8 bytes per Long, etc; the total is then aligned to a multiple 8 bytes
- Retained set of X is the set of objects that will be garbage collected if X is garbage collected
- Retained heap of X is the sum of shallow sizes of all objects in the retained set of X, i.e., memory kept alive by X

For Your Reference

Garbage Collection Root

…is an object which is defined to be reachable by the JVM software:

- **System Class**—Class loaded by system class loader, e.g., java.lang.String
- **Java Local**—Local variable, i.e., method input parameters or locally created objects of methods still on the stack of a thread
- **Busy Monitor**—Everything you have called wait() or notify() on or you have synchronized on
- **Thread Block**—Started but not stopped threads
- **JNI Local**—Local variable in native code
- **JNI Global**—Global variable in native code
- **Native Stack**—In or out parameters in native code; frequently seen as some methods have native parts and the objects handled as method parameters become GC roots, e.g., parameters used for file/network I/O methods or reflection

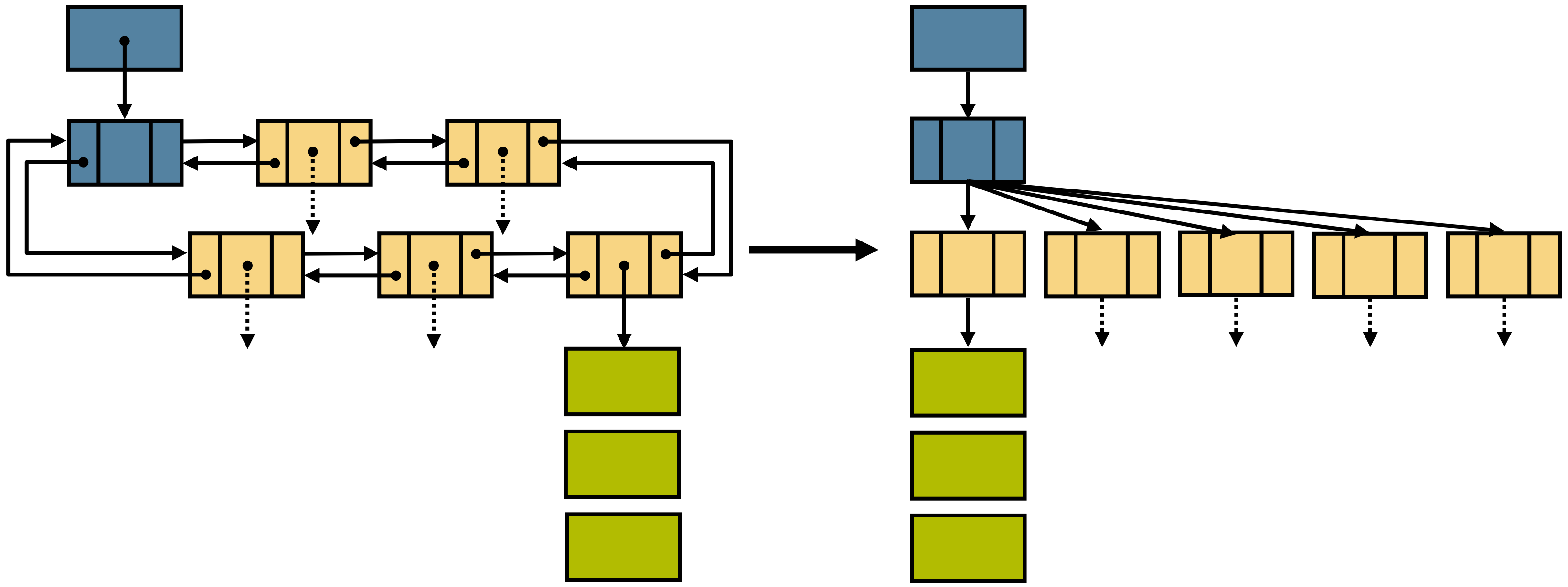
JNI = Java Native Interface (JNI™)

DEMO

SAP Memory Analyzer

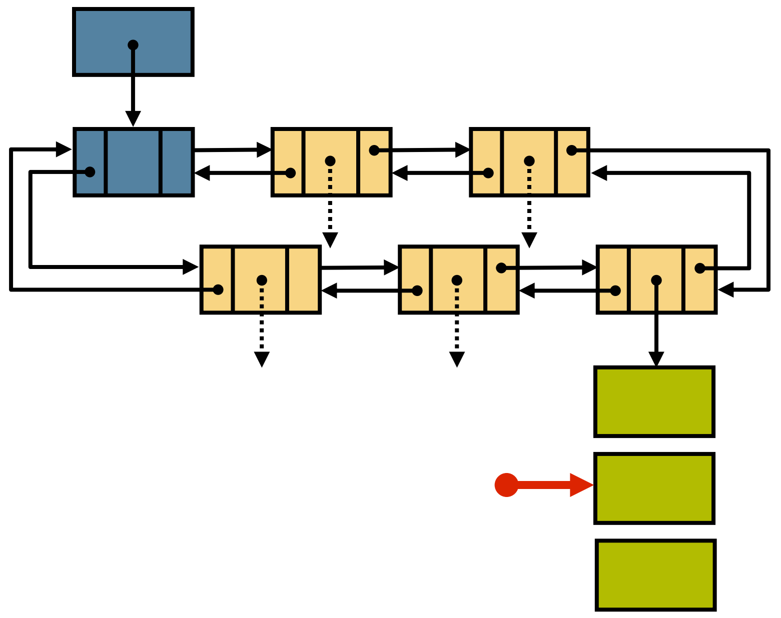
Pick it up at the SAP booth.

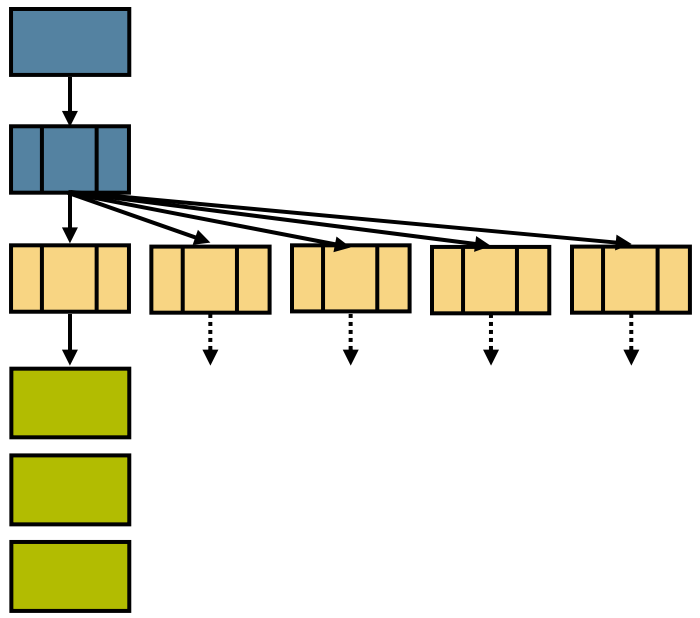
Dominator Tree



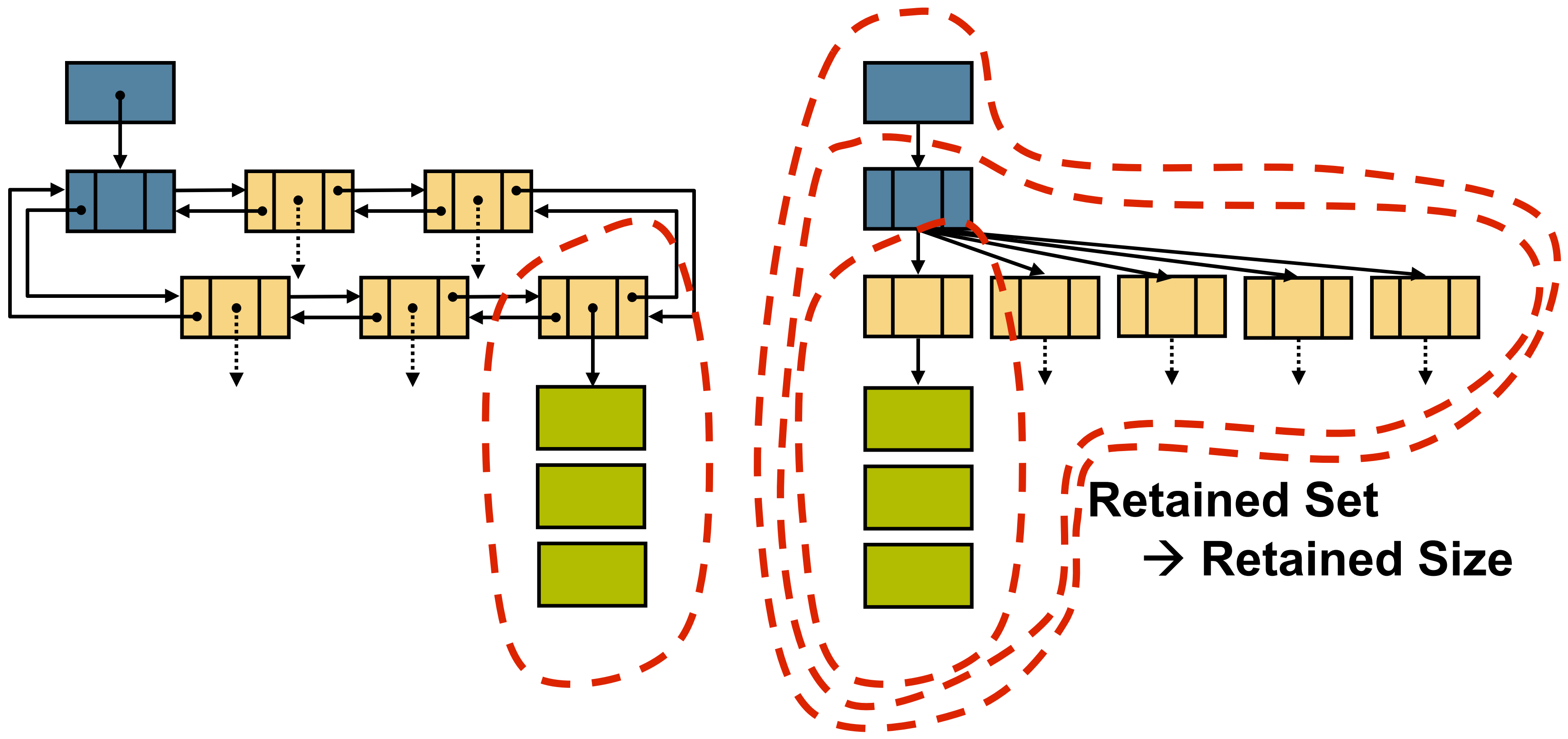
X dominates Y if all paths to Y run through X

Dominator Tree

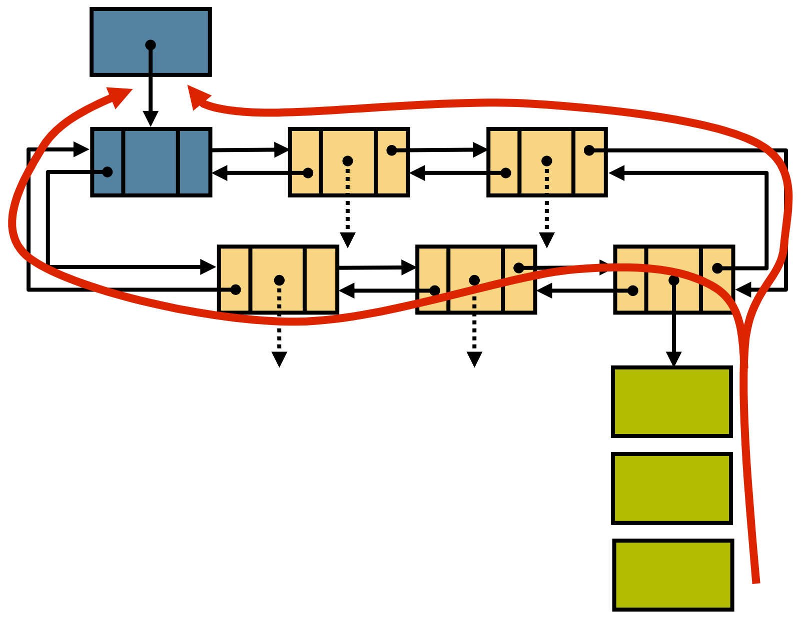


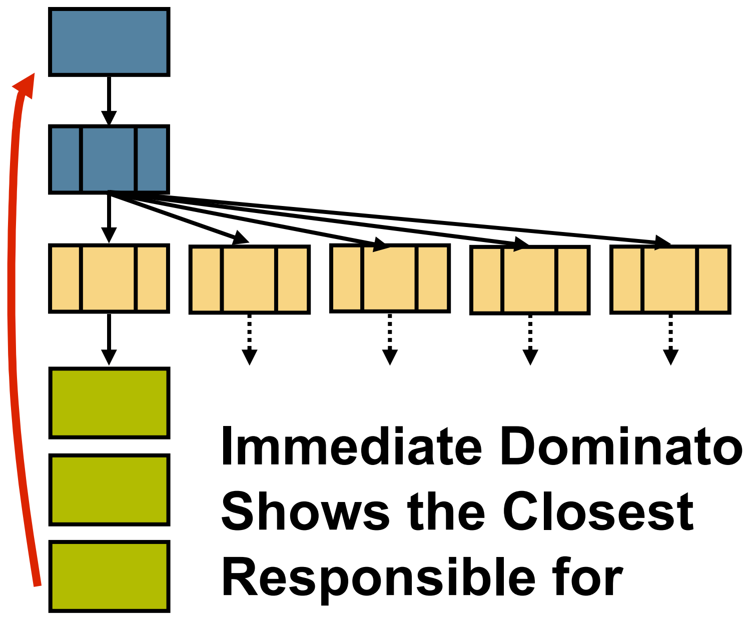


Benefit #1
Retained Set and Size Is the Subtree



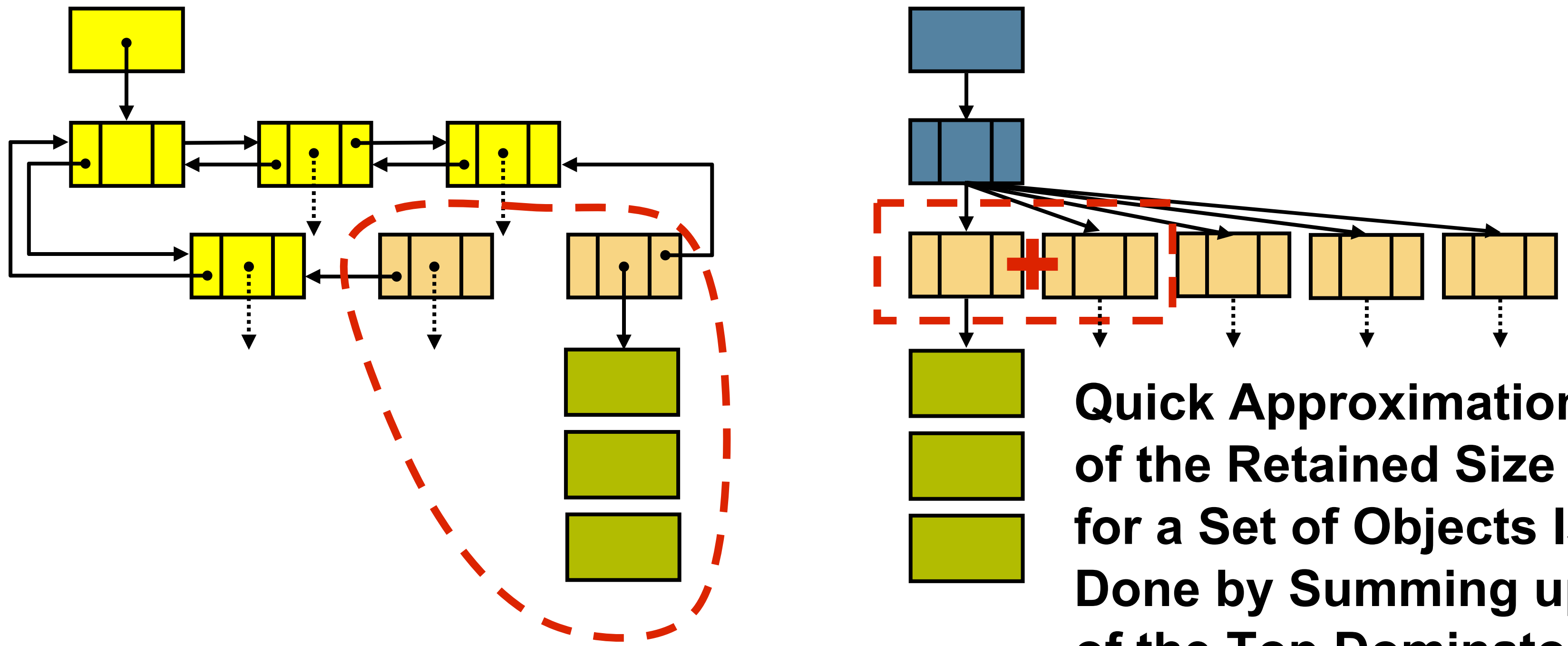
Benefit #2
Quickly Find the Greedy Memory Pigs





Immediate Dominator Shows the Closest Responsible for Keeping an Analyzed Object Alive

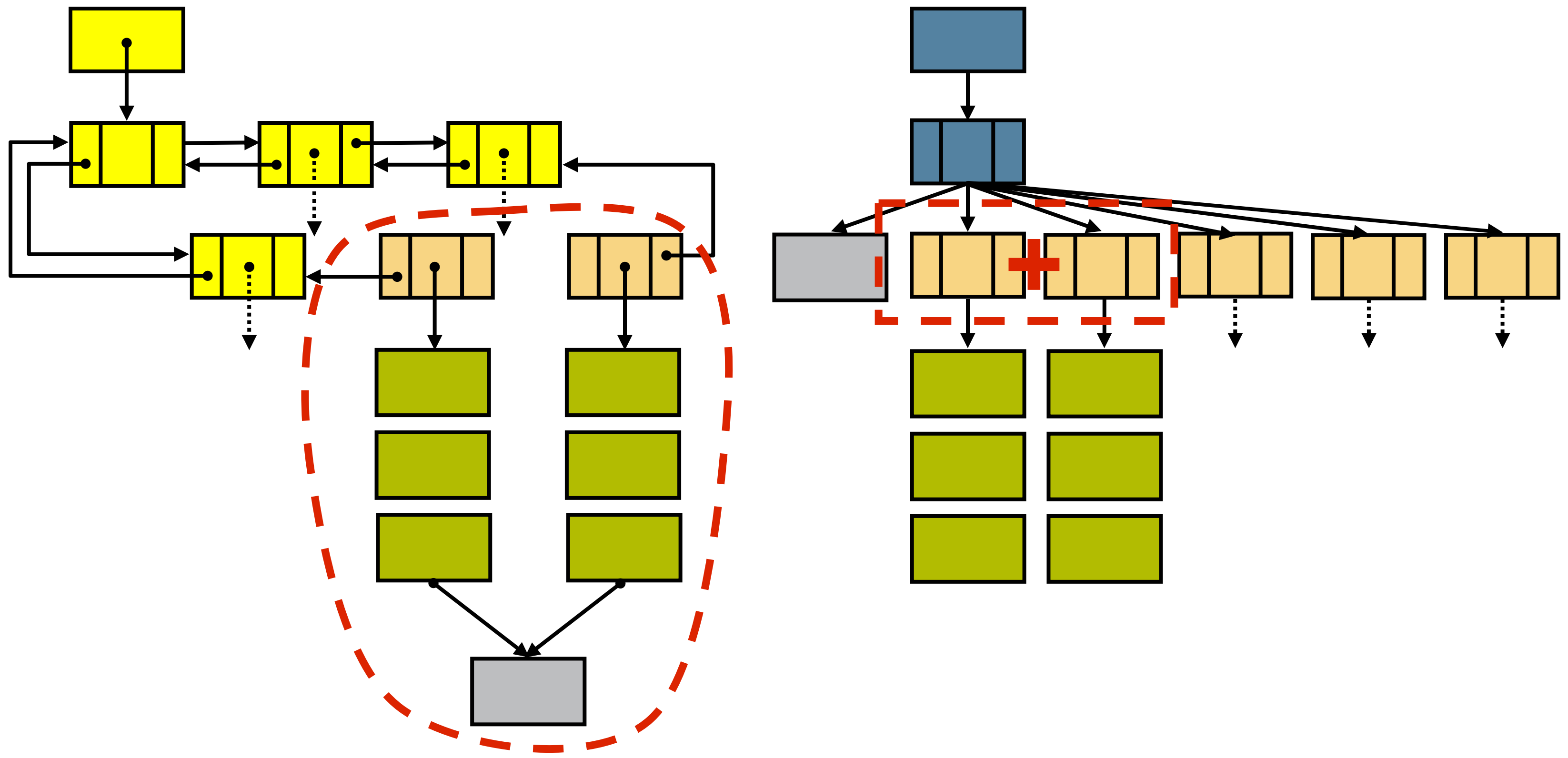
Benefit #3
Fast Retained Size Approximation



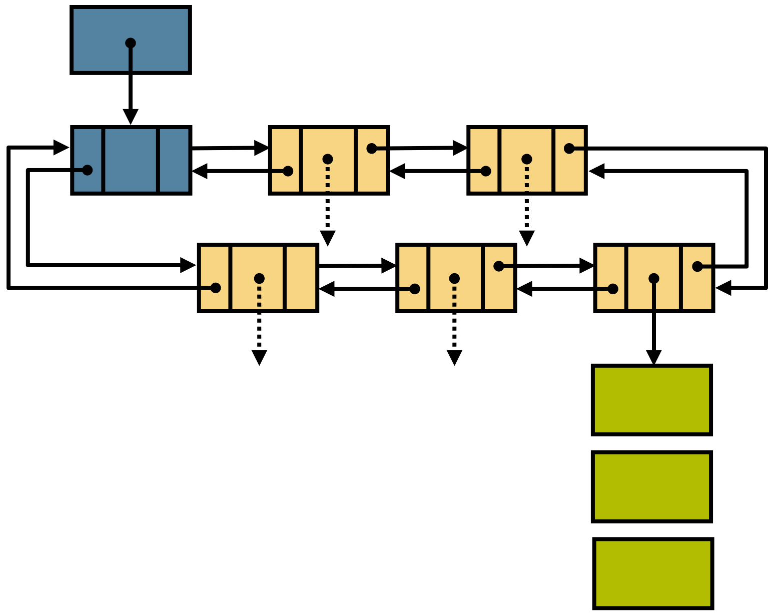
Quick Approximation of the Retained Size for a Set of Objects Is Done by Summing up of the Top Dominators in the Set

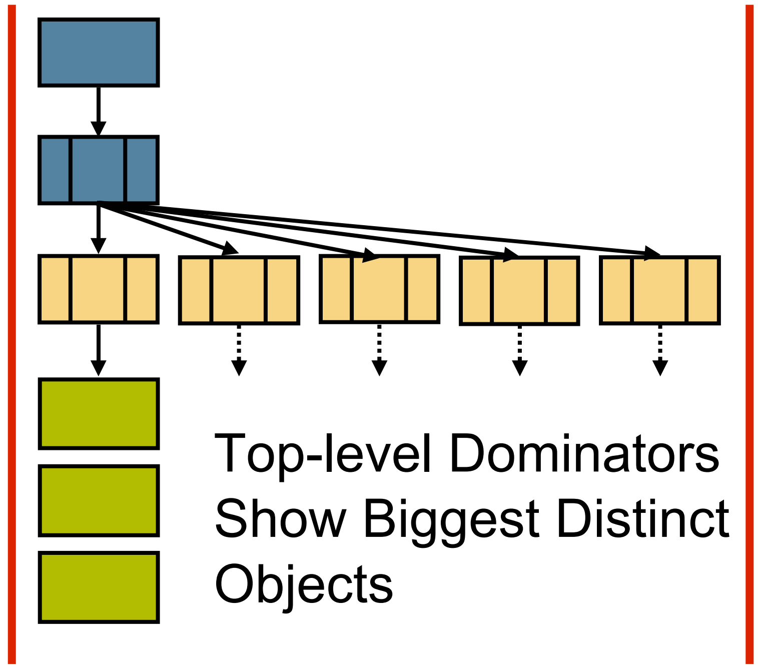
For Your Reference

Benefit #3
Fast Retained Size Approximation



Benefit #4
Biggest Distinct Object Graphs





Top-level Dominators Show Biggest Distinct Objects

→Easy Grouping by Class, Class Loader

For Your Reference

Dominators and Dominator Tree

- An object x dominates an object y if every path in the object graph from the start (or the root) node to y must go through x
- The immediate dominator x of some object y is the dominator closest to the object y
- We build a dominator tree out of the object graph; in the dominator tree each object is the immediate dominator of its children

For Your Reference

Dominator Tree Properties

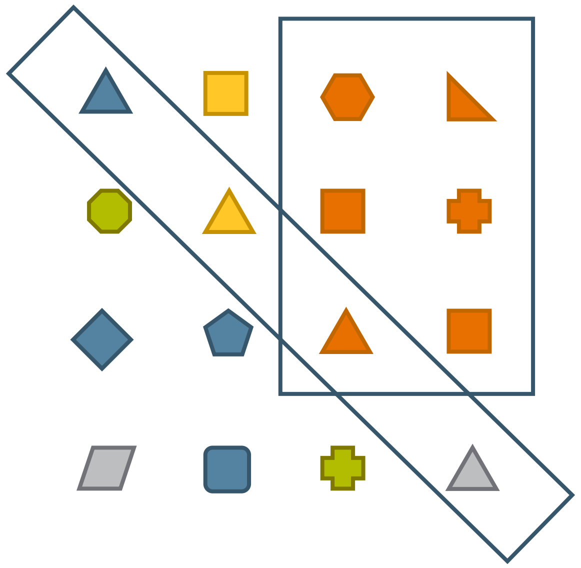
- The objects belonging to the sub-tree of x (i.e., the objects dominated by x) represent the retained set of x
- If x is the immediate dominator of y, the immediate dominator of x also dominates y
- The edges in the dominator tree do not directly correspond to object references from the object graph

DEMO

SAP Memory Analyzer

Pick it up at the SAP booth.

Grouping Anywhere



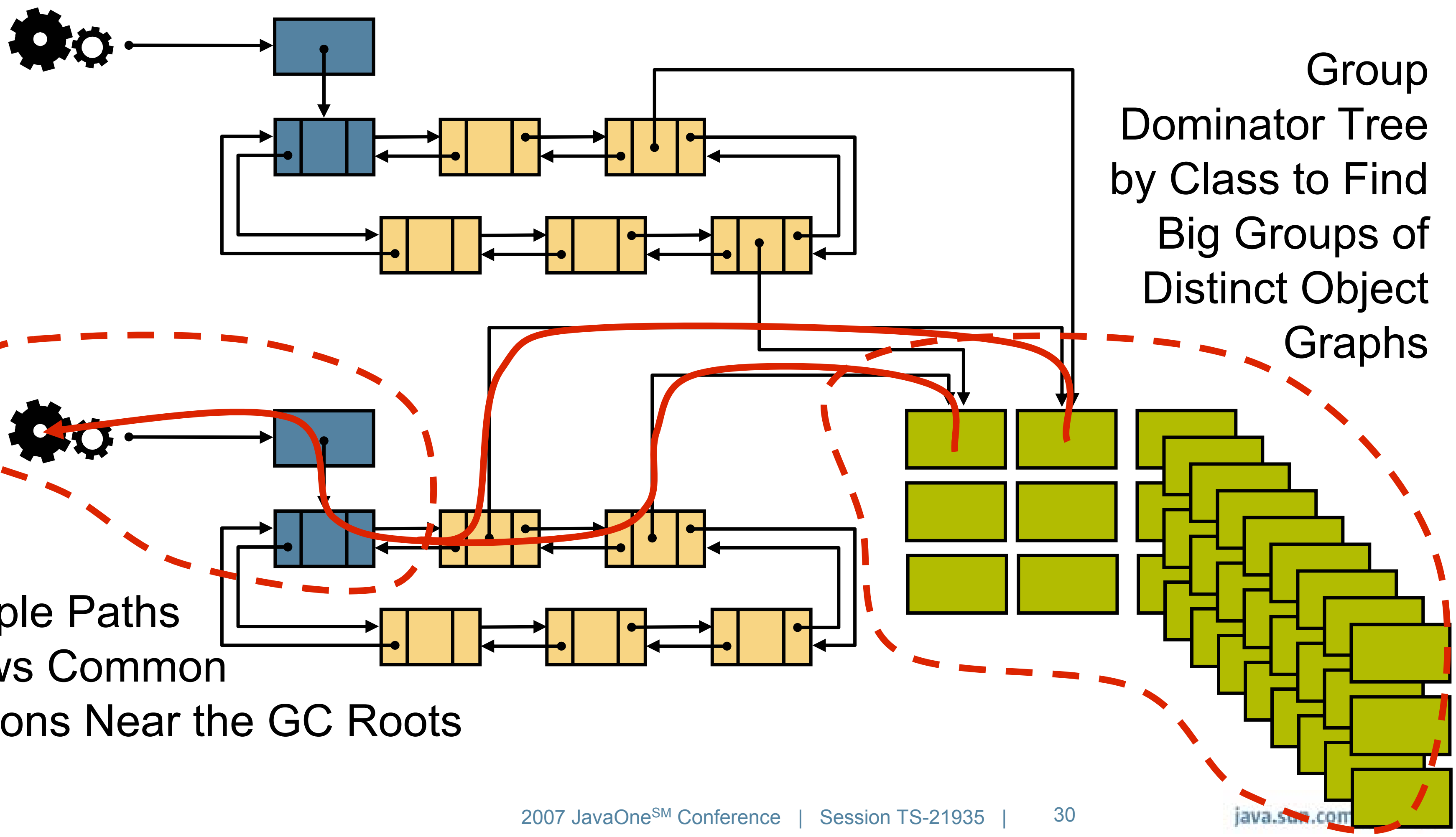
1. **Look for a property**
2. **Group objects by it**
3. **Inspect big chunks**

Examples:

- Arrays by Length
- Strings by Value

…

Group in Top Dominators Group Along Shortest Paths

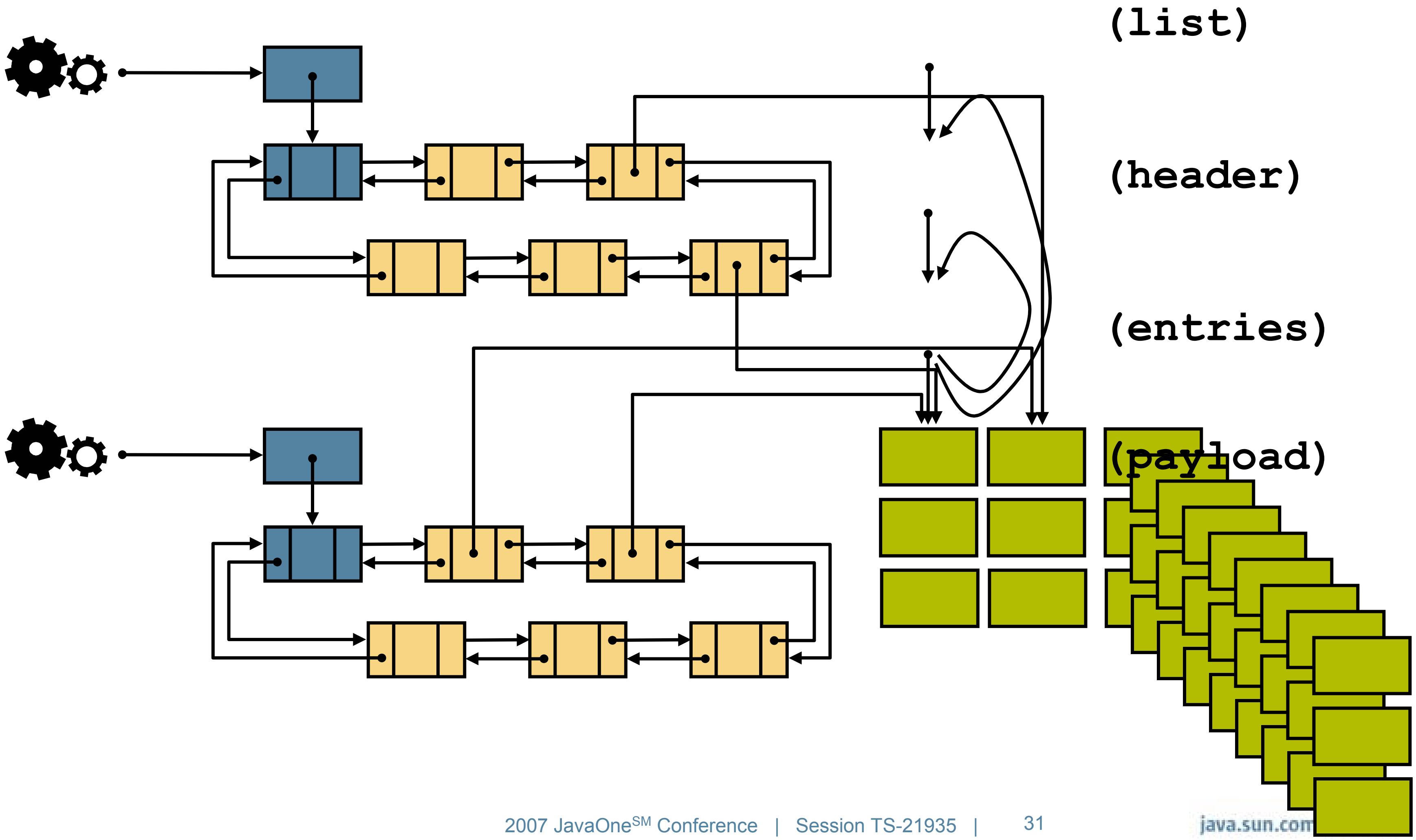


Group Dominator Tree by Class to Find Big Groups of Distinct Object Graphs

Multiple Paths Shows Common Sections Near the GC Roots

JavaOne

Group Referrers by Class

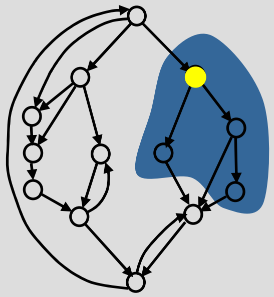
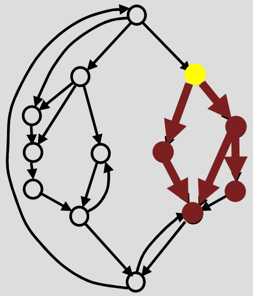
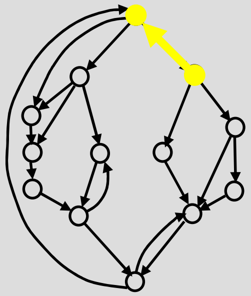


DEMO

SAP Memory Analyzer

Pick it up at the SAP booth.

A 4-Step Approach to Finding Issues

Get an Overview	Find Big Chunks	Inspect Content	Identify Holders
Total heap size **Total number of objects, classes and class loaders** **Class Histogram**			

Finding Single Objects

Get an Overview	Find Big Chunks	Inspect Content	Identify Holders
	Check Dominator Tree	Expand / explore Dominator Tree	Object inbound references
	OQL	Analyze the Retained Set	Paths from the GC roots
		Object outbound references/Object inspector	Open in Dominator Tree
		OQL	

Finding Groups of Objects

Get an Overview

Find Big Chunks

- Grouping in Dominator Tree
- OQL

Console:

```
TOP_CONSUMERS
ARR_SZ_HISTOGRAM
LOCAL_VARS
...
```

Inspect Content

- Object lists / histograms
- Analyze the Retained Set/Size
- Class-level outbound references
- OQL

Identify Holders

- Immediate dominators of
- Class-level inbound references
- Multiple paths from the GC roots

DEMO

SAP Memory Analyzer

Pick it up at the
SAP Booth.

Question

How many objects do you find in big production heap dumps?

a) ~ 1.000.000

b) ~ 10.000.000

c) ~ 100.000.000

Agenda

Memory 101

Analysis Techniques

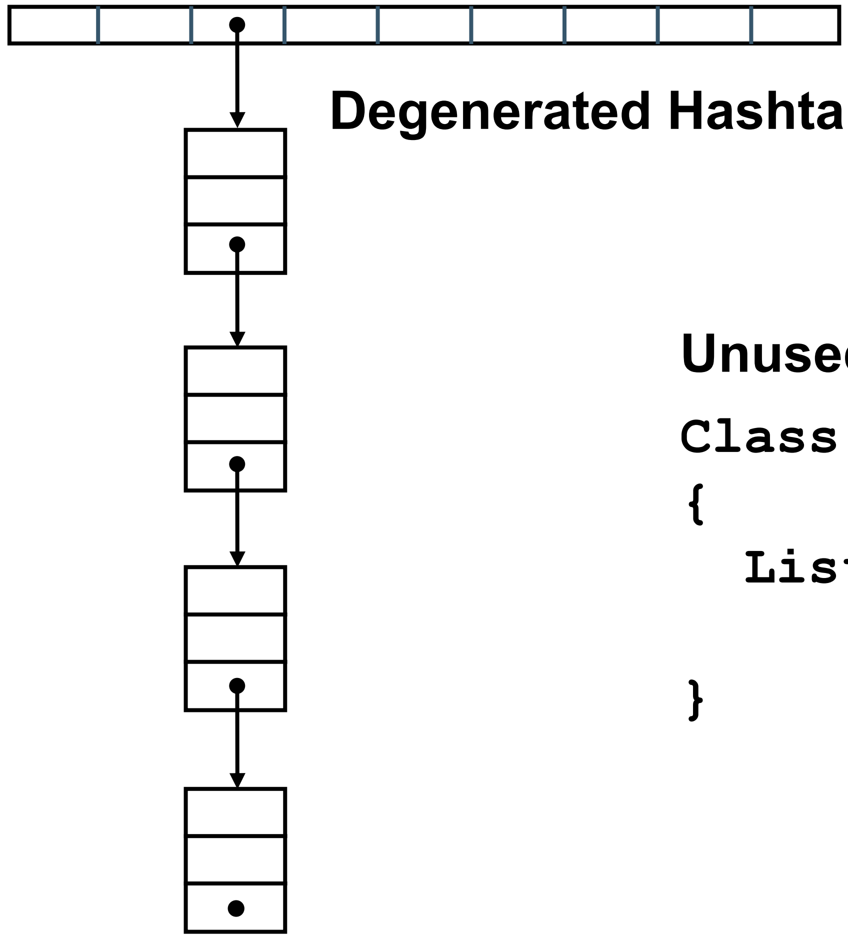
Pathogen Memory

- Inefficient Data Structures
- Duplicate Classes and Leaking Loaders

Lessons Learned

Q&A

Inefficient Data Structures

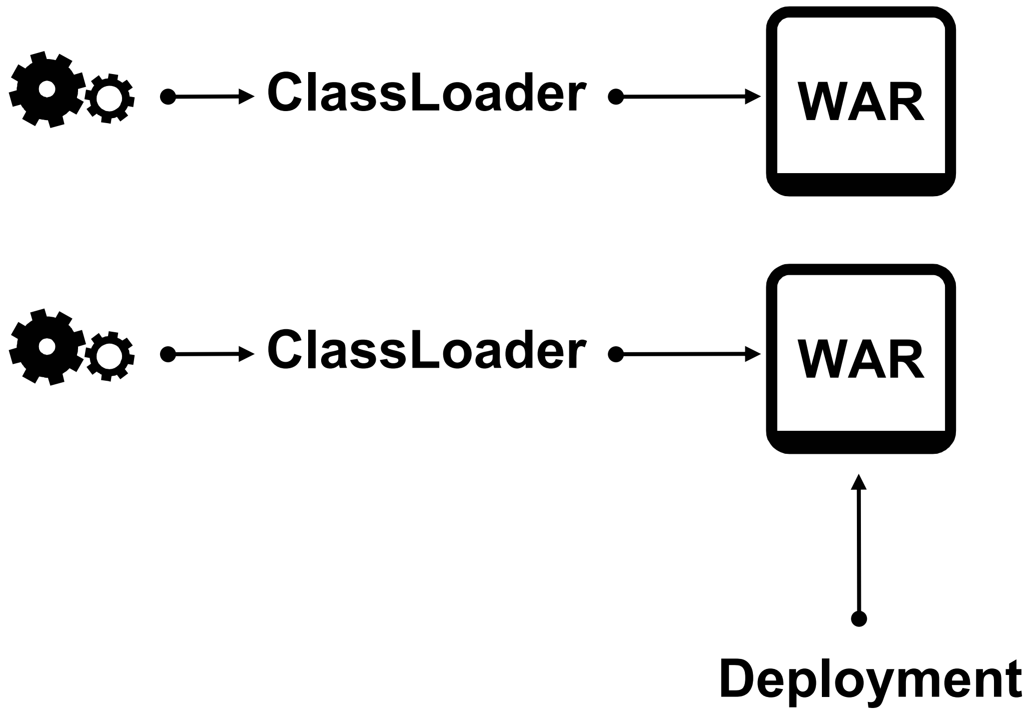


Degenerated Hashtable

Unused Collections

```
Class Node
{
  List children =
            new ArrayList();
}
```

Duplicate Classes and Leaking Loaders



Agenda

Memory 101

Analysis Techniques

Pathogen Memory

Lessons Learned

Q&A

Critical Problems

- Heap
  - Inefficient data structures (e.g., badly used collections, keeping XML DOM,...)
  - Caches (i.e., unknown entry size, different competing caches,...)
- Perm
  - Model/Proxy-driven class generation
  - “Leaking” loaders
- In General
  - Real size of objects not apparent to programmer
  - No application/user quota

Lessons Learned

- Memory is performance
- It’s not about leaks; it’s about footprint
- Developer tools do not fit enterprise demands
- Analysis can be automated (Expert System)

For Your Reference

Wish List for HPROF Binary++

- Stable object ids
- GC object survival counts
- Perm space info
- Transient field info
- Interface implementing info
- Thread dump
- No more garbage
- Class info before object data
- ...

Q&A

vedran.lerenc@sap.com

andreas.buchen@sap.com

Visit us at SAP booth

Wednesday, Thursday
11:30 am–1:30 pm

https://www.sdn.sap.com/irj/sdn/wiki?path=/display/Java/Java+Memory+Analysis

Improving the Quality of Your Enterprise Application: Innovative Ways to Spot Memory-Related Bugs and Bottlenecks

Vedran Lerenc, Andreas Buchen

Programmers
SAP AG
http://www.sap.com

TS-21935