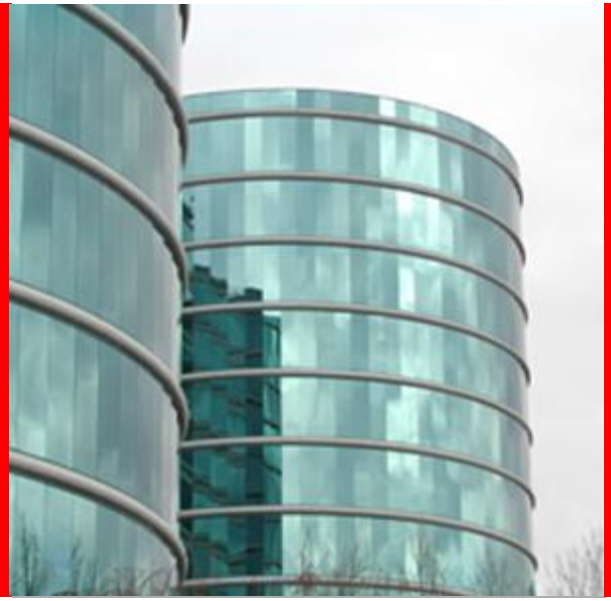

ORACLE®



ORACLE®

Tuning Oracle Fusion Middleware 11g

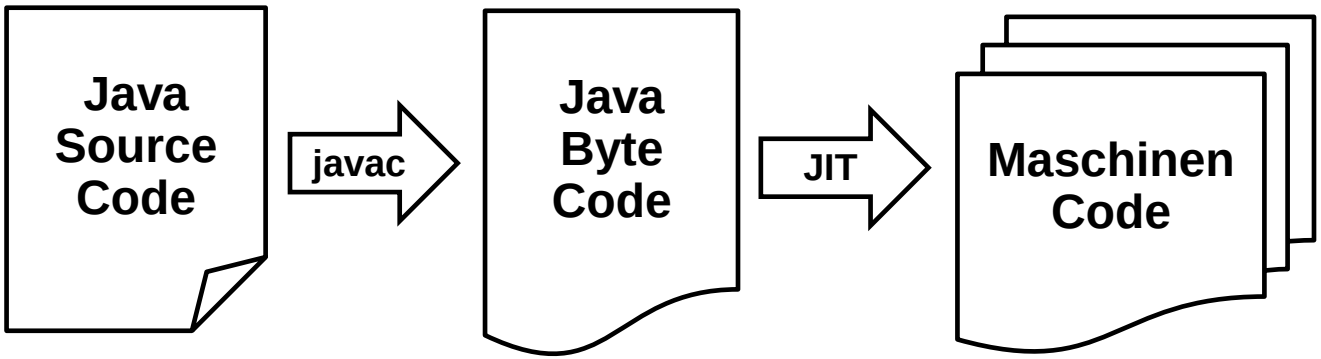
Thomas Robert
Oracle Deutschland GmbH

Jan-Peter Timmermann
Pitss GmbH

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Java VM

- OpenJDK, JRockit, HotSpot, u.a.



- HotSpot und JRockit enthalten zwei Compiler:
  - Client (Schnelle Code Kompilierung)
    - weniger Optimierung,
    - Geringerer Memoryverbrauch,
    - Dadurch schnellerer Startup
  - Server (Bessere Code Optimierung)
    - Bessere Server Performance

Garbage Collection

- Der Programmierer ist nicht (d.h. nur indirekt) für die Speicherverwaltung zuständig
  - Kein Allokieren von Speicher
  - Kein explizites Freigeben von Speicher
    - Aber „Freigeben“ von überflüssigen Objekten
- Aufgabe von Garbage Collection ist
  - Auffinden von nicht mehr referenzierten Objekten
  - Freigeben der Objekte (und deren Memory)
  - Komprimieren von Speicherbereichen

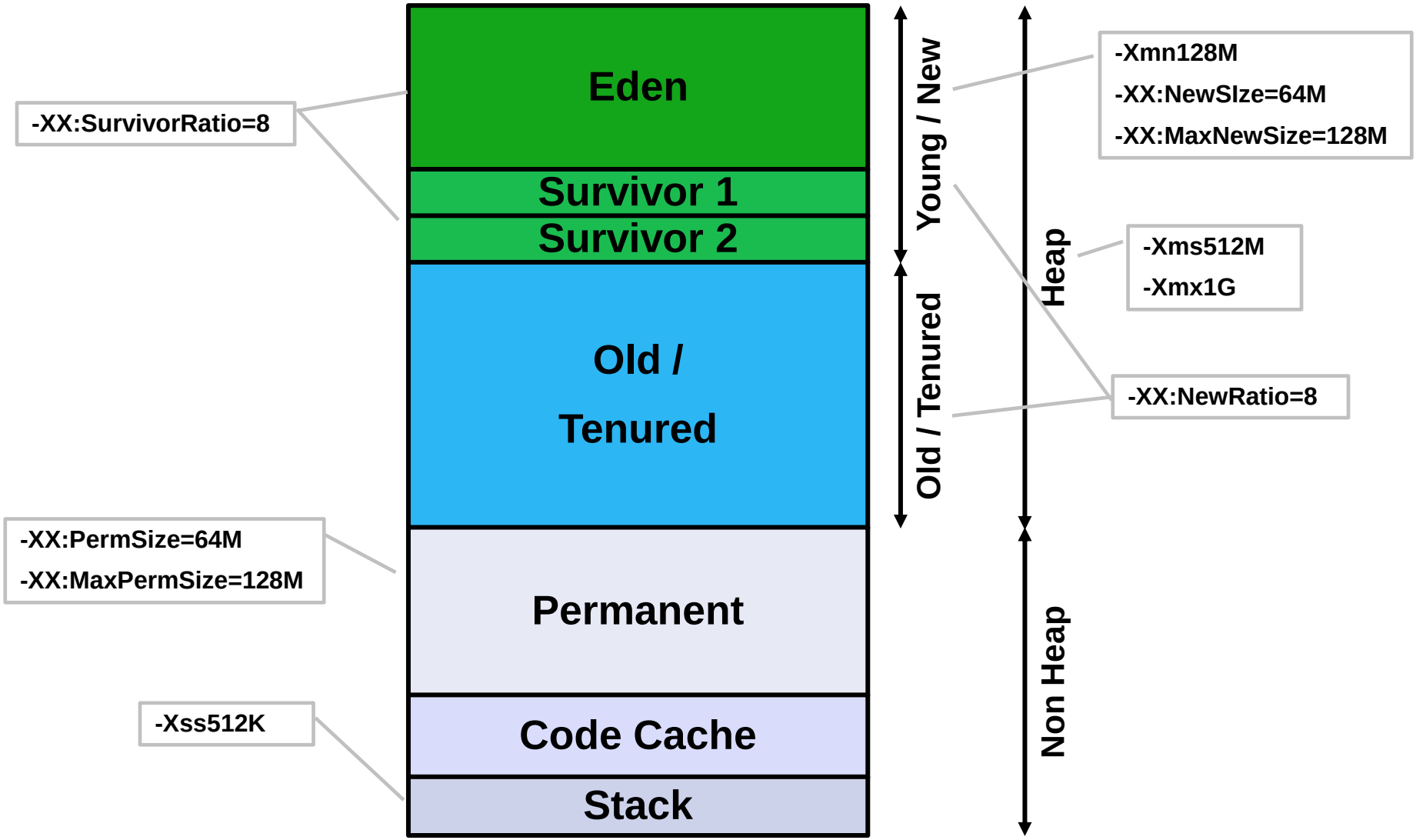
Speicherverwaltung

- Eine JVM verwaltet unterschiedliche Speicherbereiche. Warum?
  - Die meisten Objekte sind temporär, werden also nur kurze Zeit benötigt
  - Lang lebende Objekte verweisen i.d.R. auf lang lebende Objekte, nur selten auf kurz lebende Objekte
- Diese Speicherbereiche werden „Generations“ genannt
  - Young Generation für kurzlebige Objekte
  - Old Generation für langlebige Objekte
- Garbage Collector
  - Häufige Young Collections (schnell: <50ms)
  - Wenige Full Collections (langsamer: >100ms bis mehrere Sek.)
- GC Strategien: Optimierung von
  - Durchsatz
    oder
  - Antwortzeit

Speicherverwaltung

- Non Heap (PermGen)
  - Klassendefinition, statische Variablen, Stack, etc.
- Heap
  - Young (Nursery) Space
    - Eden Space
    - Survivor 0
    - Survivor 1
  - Old (Tenured) Space
- JRockit kennt keinen PermGenSpace
  - Klassendefinitionen etc. werden im Prozessmemory abgelegt
  - Keine vordefinierte maximale Größe

Heap Pools


Eden
Survivor 1
Survivor 2
Old / Tenured
Permanent
Code Cache
Stack
-XX:SurvivorRatio=8
-XX:PermSize=64M
-XX:MaxPermSize=128M
-Xss512K
Young / New
Old / Tenured
Heap
Non Heap
-Xmn128M
-XX:NewSIze=64M
-XX:MaxNewSize=128M
-Xms512M
-Xmx1G
-XX:NewRatio=8

Java Performance Tuning

1. Java Code Optimierung
   - Sun Java System Application Server Standard and Enterprise Edition 7 2004Q2 Performance and Tuning Guide
     http://docs.oracle.com/cd/E19644-01/817-5051/index.html
   - Zum Beispiel
     - Vermeidung von Synchronisierung und Serialisierung
     - Sinnvolles allokieren und Freigeben von Objekten
     - Nutzung von StringBuffern statt Strings für variable Strings
2. Garbage Collection Tuning
   - Tuning Garbage Collection with the 5.0 Java[tm] Virtual Machine
     http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html
   - Java SE 6 HotSpot[tm] Virtual Machine Garbage Collection Tuning
     http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
   - Durchsatz: Parallel GC
   - Antwortzeit: Concurrent GC

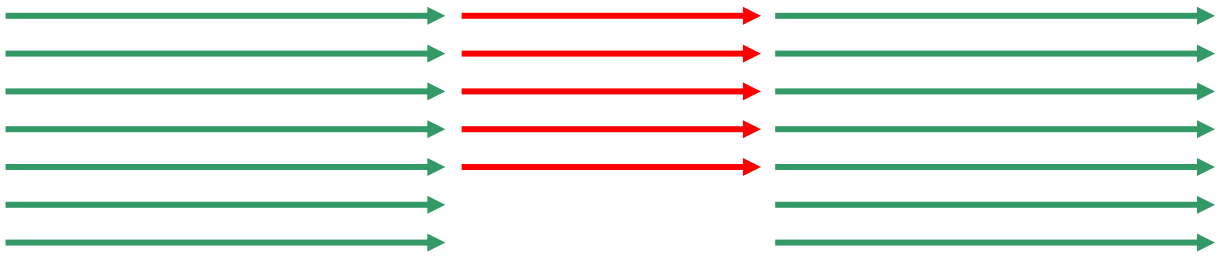
Serielles Garbage Collection

- Einschalten z.B. auf Single Processor Maschninen mit
  - -XX:+UseSerialGC

Garbage Collection
Program Code

Parallel Garbage Collection

- Einschalten für höheren Durchsatz
  - Young GC:
    - -XX:+UseParallelGC
  - Old GC
    - -XX:+UseParallelOldGC
    - Automatisch auch –XX:+UseParallelGC
  - Anzahl paralleler Threads
    - -XX:ParallelGCThreads=<#Threads>



Garbage Collection
Program Code

Concurrent Garbage Collection

- Concurrent Low Pause (Mark Sweep) Collector (CMS)
  - -XX:+UseConcMarkSweepGC
- Concurrent (Old) und parallel (young) Collector
  - -XX:+UseConcMarkSweepGC
  - -XX:+UseParNewGC

Garbage Collection
Program Code

G1 Collector

- Neu in JavaSE 6
- Einschalten mit
  - -XX:+UnlockExperimentalVMOptions
  - -XX:+UseG1GC
- Keine Trennung von Young und Old
  - Stattdessen „Regions“
- Wird CMS Collector ersetzen

Garbage Collection
Program Code

Inkrementelles Garbage Collection

- Einschalten mit
  - -Xincgc
- Oder zusammen mit Concurrent GC:
  - -XX:+UseConcMarkSweepGC
  - -XX:+CMSIncrementalMode
- Statt weniger langer GC Pausen

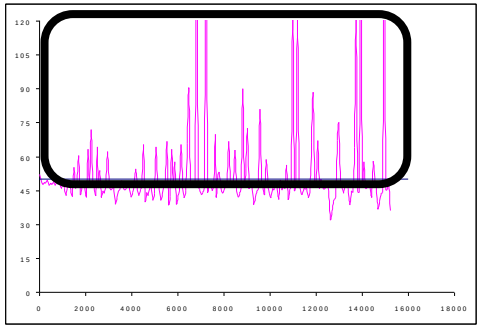
viele kurze Pausen

Garbage Collection
Program Code

JRockit RealTime

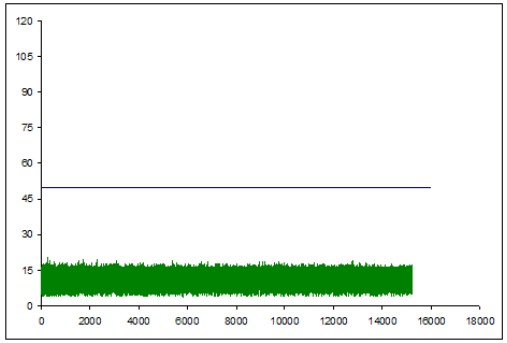
- Primäres Ziel von JRRT sind deterministische GC Pausen
- Optimimierung für deterministische Antwortzeiten
  - Kann Gesamtdurchsatz verringern
    - JRockits automatische GC-Optimierung verbessert sowohl Antwortzeit als auch Durchsatz
    - JRRT optimiert nur Antwortzeit. Einfluss auf Durchsatz ist Applikationsabhängig
  - Erfordert möglicherweise mehr CPU
  - Nicht durch GC verursachte Wartezeiten weiterhin möglich

Traditionelle JVM



Bei hoher Last: GC Pausen können zu unakzeptablen Antwortzeiten führen

JRockit Real Time



JRRT ermöglicht *deterministische* Garbage Collection. Ermöglicht SLA Garantien.

JRRT – Deterministische GC

- JRockit Erweiterung um deterministischen GC
  - -XgcPrio:[pausetime | throughput | **deterministic**]
  - -XpauseTarget kann zwischen **1ms** und 5 Sek sein
  - Pause Ziel ist “***Maximal”***-Wert
- Lizensiert mit
  - WebLogic Suite
  - Java Suite

Welcher Garbage Collector?

- Analyse des GC Verhaltens mit
  1. verboseGC

```
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps
-XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCApplicationStoppedTime
```

  2. Tools wie Mission Control oder VisualGC

verboseGC Output

- Standard GC

```
[GC [PSYoungGen: 26139K->160K(64000K)] 29642K->3679K(96768K), 0.0006630 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (System) [PSYoungGen: 160K->0K(64000K)] [PSOldGen: 3519K->2973K(32768K)] 3679K->2973K(96768K)
[PSPermGen: 9828K->9828K(21248K)], 0.0224630 secs] [Times: user=0.03 sys=0.02, real=0.02 secs]
```

- -XX:+UseParallelOldGC

```
[GC [PSYoungGen: 19012K->160K(64064K)] 22429K->3592K(96832K), 0.0004050 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[Full GC (System) [PSYoungGen: 160K->0K(64064K)] [ParOldGen: 3432K->1340K(32768K)] 3592K->1340K(96832K)
[PSPermGen: 9816K->9807K(21248K)], 0.0397240 secs] [Times: user=0.06 sys=0.01, real=0.03 secs]
```

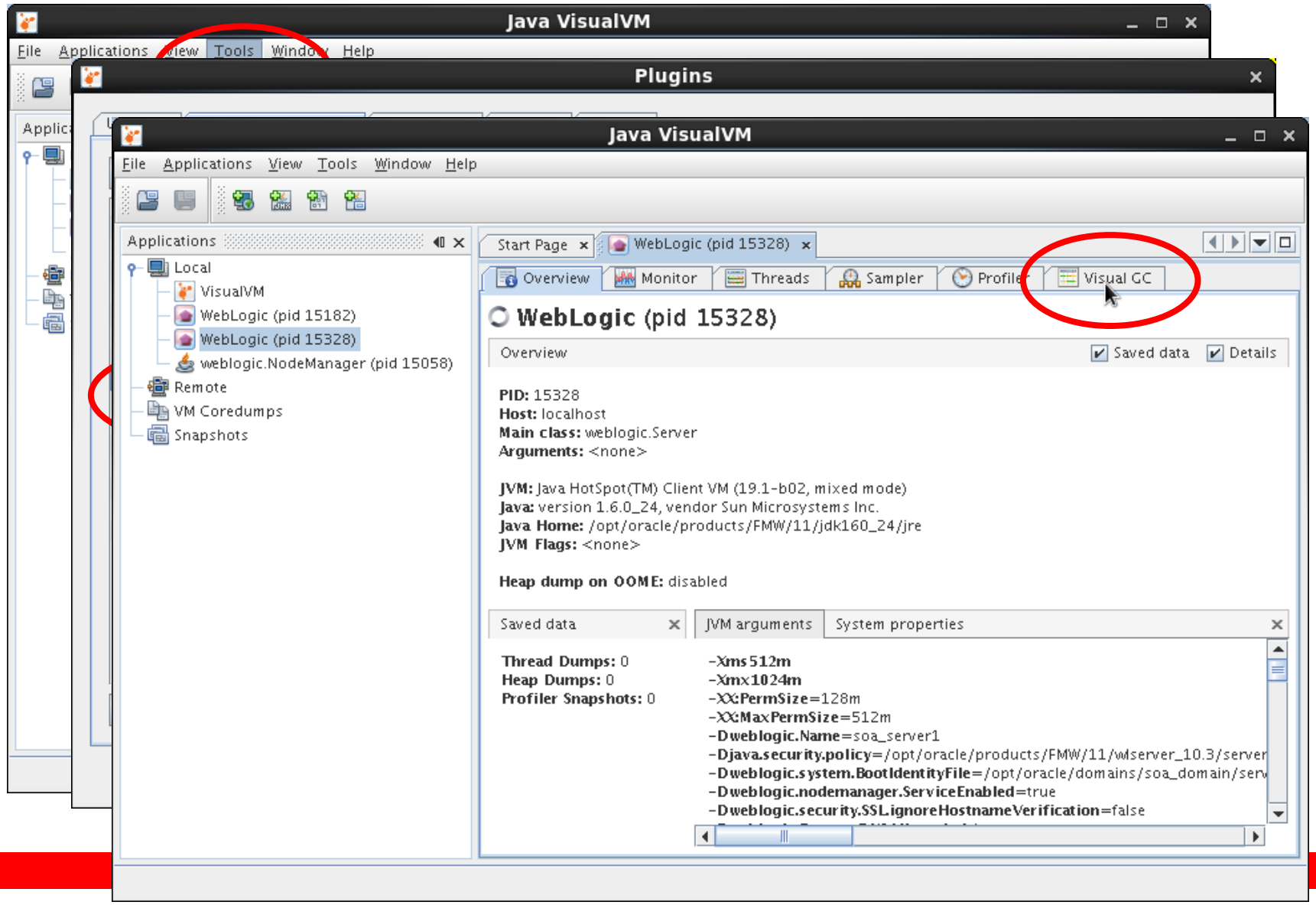
- -XX:+UseConcMarcSweepGC

```
[GC [ParNew: 54153K->1429K(59008K), 0.0007450 secs] 56087K->3363K(91776K), 0.0007720 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (System) [CMS: 1934K->1286K(32768K), 0.0278440 secs] 14746K->1286K(91776K), [CMS Perm : 9813K->9802K(21248K)], 0.0279120 secs]
[Times: user=0.03 sys=0.00, real=0.03 secs]
```

- -Xincgc

```
[CMS-concurrent-mark: 0.024/0.435 secs] [Times: user=1.49 sys=0.04, real=0.44 secs]
[CMS-concurrent-preclean: 0.013/0.013 secs] [Times: user=0.04 sys=0.00, real=0.01 secs]
[GC [ParNew: 52480K->1838K(59008K), 0.0166290 secs] 53423K->2781K(91776K), 0.0166680 secs] [Times: user=0.05 sys=0.00, real=0.02 secs]
[CMS-concurrent-abortable-preclean: 0.067/1.269 secs] [Times: user=3.65 sys=0.10, real=1.27 secs]
[GC[YG occupancy: 28258 K (59008 K)][Rescan (parallel) , 0.0299700 secs][weak refs processing, 0.0000120 secs] [1 CMS-remark: 943K(32768K)]
29201K(91776K), 0.0300470 secs] [Times: user=0.09 sys=0.00, real=0.03 secs]
[CMS-concurrent-sweep: 0.001/0.001 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[CMS-concurrent-reset: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
```

Visualisieren mit VisualGC



Demo

- VisualGC im jvisualvm
- JRockit MemoryLeak Analyse
  - Linzesiert mit
    - WebLogic
    - WebLogic
    - JavaSE A

```
public class TestMemleak{

    public static Vector staticPersons;

    public static void main(String arg[]) throws Exception{
        Vector mainPersons = new Vector();
        staticPersons = new Vector();

        while(true){
            Person p = new Person("Abe", new Contact(123, "NY"));
            mainPersons.add(p);
            staticPersons.add(p);
            Person p2 = new Person("Bob", new Contact(234, "LA"));
            p2.addFriend(p);
            mainPersons.add(p2);
            staticPersons.add(p2);
            Person p3 = new Person("Cid", new Contact(345, "Stockholm"));
            p3.addFriend(p);
            p3.addFriend(p2);
            mainPersons.add(p3);
            staticPersons.add(p3);
            Thread.sleep(10);
        }
```


ORACLE®