BED-Con 4.4.2013

Memory Management und Garbage Collectoren: TP, CMS und G1

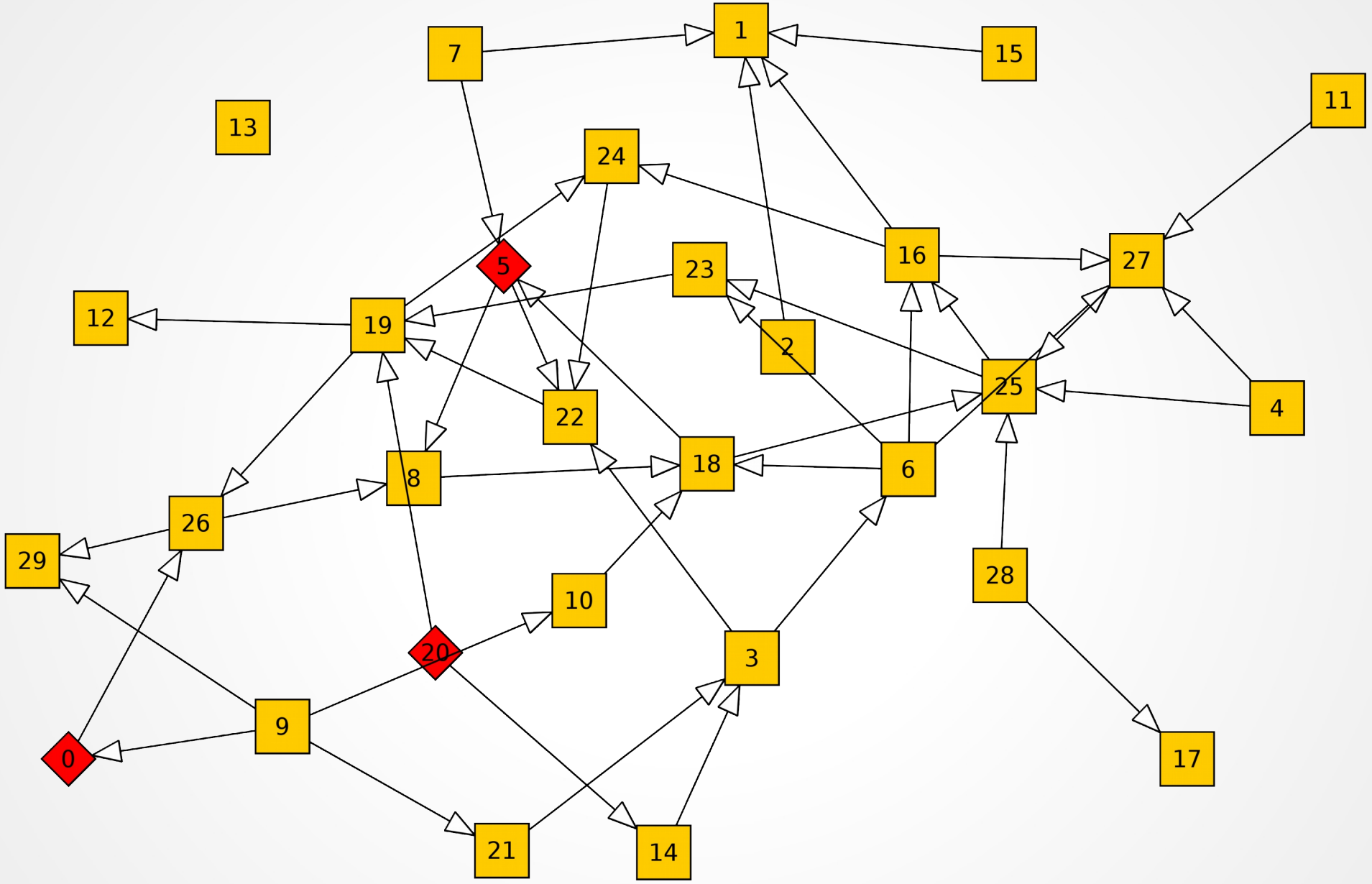
Welche GC-Strategie ist die richtige?

Tobias@Frech.info / @TobiasFrech

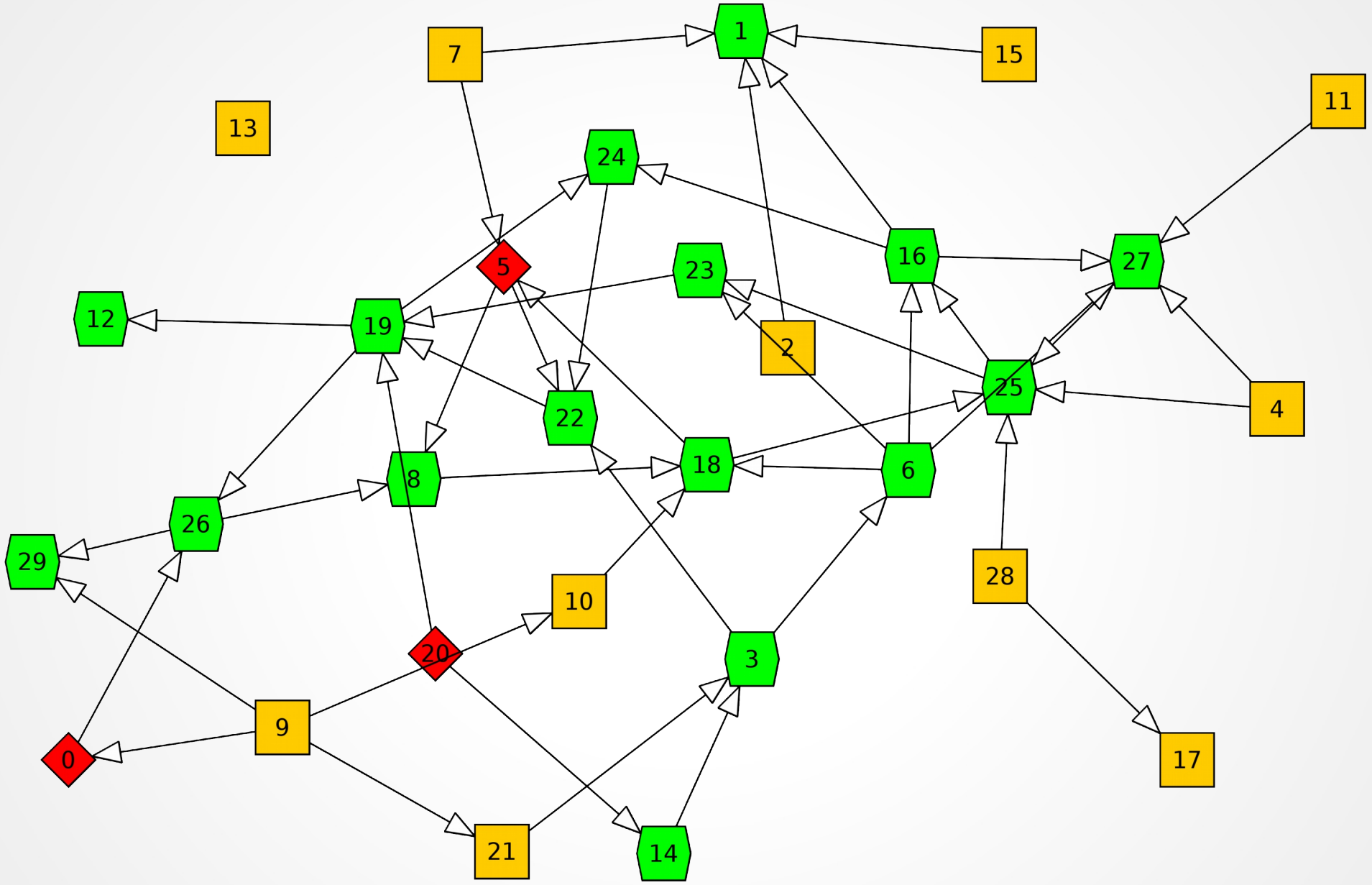
Tobias Frech

Java - Administrator

Objekte oder Garbage?



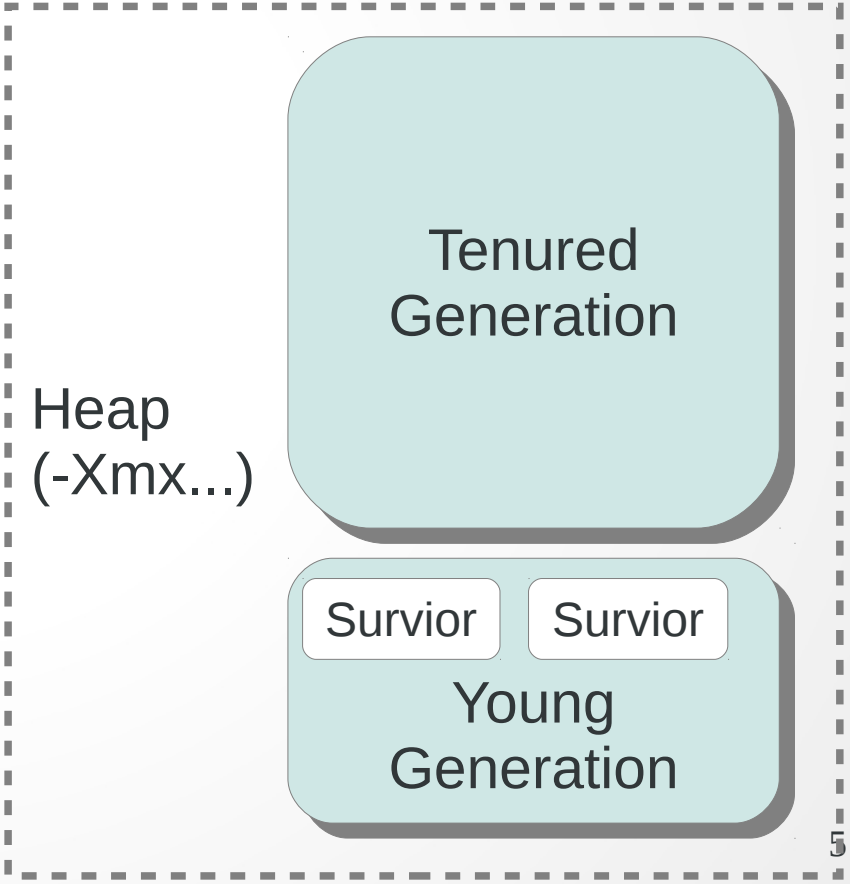
Garbage Collection



GC Optimierung

- Objekte nach der Lebensdauer sortiert:

- Einführung von Generationen:



GC Logging

- GC-Logs nicht überschreiben:
  - -Xloggc:gc.log_`date +%F_%T`
- Für die Menschen unter uns:
  - -XX:-PrintGCTimeStamps
  - -XX:+PrintGCDateStamps
- Für die Admins:
  - -XX:+UseGCLogFileRotation
  - -XX:GCLogFileSize=, -XX:NumberOfGCLogFiles=

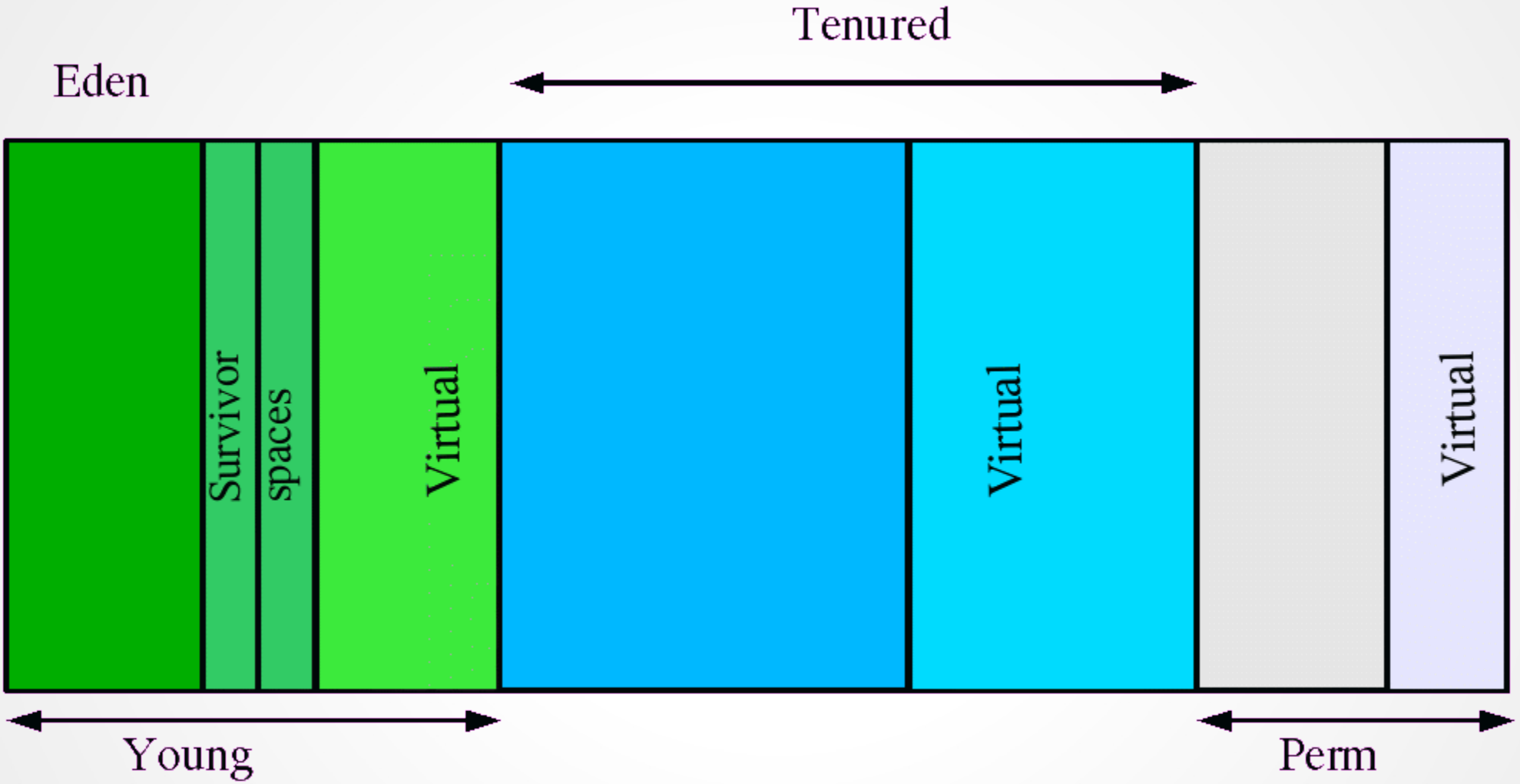
Throughput – GC - Log

(201828s = 2 Tage 8 Stunden 3 Minuten 48 Sekunden)

201828.280: [**GC**
[PSYoungGen: 1816941K->29609K(4072576K)]
3126522K->1400938K(12461184K), 0.2465840 secs]
[Times: **user**=0.44 sys=0.00, **real**=0.24 secs]

201828.527: [**Full GC** (System)
[PSYoungGen: 29609K->0K(4072576K)]
[ParOldGen: 1371329K->1154377K(8388608K)]
1400938K->1154377K(12461184K)
[PSPermGen: 181489K->181449K(182528K)],
3.3035730 secs]
[Times: **user**=5.98 sys=0.00, **real**=3.31 secs]

GC Ergonomics



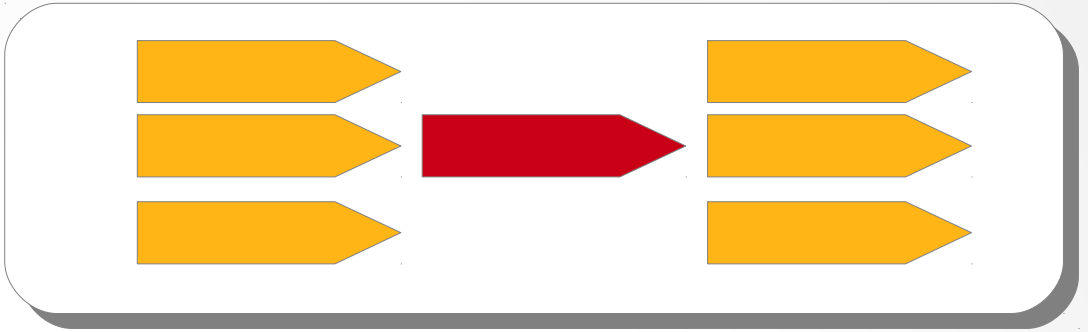
aus http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

TP - Ergonomics – GC - Log

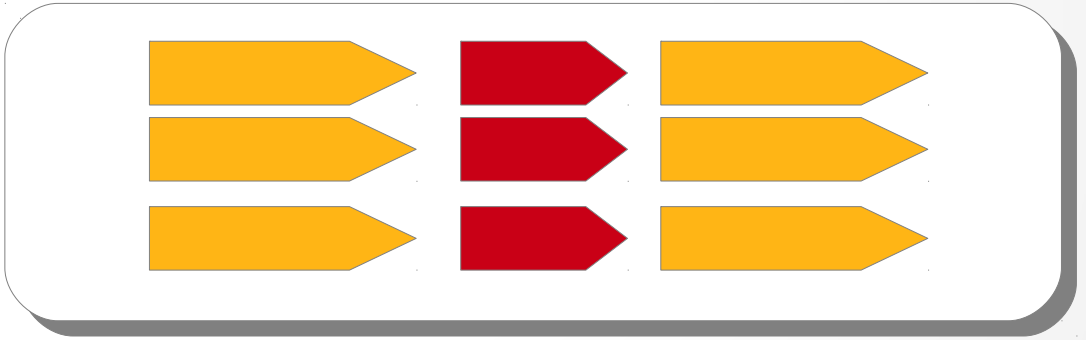
0,187: [GC [PSYoungGen: 3072K->496K(3584K)] 3072K->508K(9088K), 0,0020340 secs] [Times: user=0,00 sys=0,00, real=0,00 secs]
1,481: [GC [PSYoungGen: 9168K->400K(9856K)] 11059K->2467K(15360K), 0,0022780 secs] [Times: user=0,00 sys=0,00, real=0,00 secs]
2,747: [Full GC [PSYoungGen: 352K->0K(9472K)] [ParOldGen: 5106K->4316K(9408K)] 5458K->4316K(18880K) [PSPermGen: 1981K->1980K(16384K)], 0,0361440 secs] [Times: user=0,05 sys=0,00, real=0,03 secs]
5,839: [Full GC [PSYoungGen: 336K->0K(9984K)] [ParOldGen: 9133K->8668K(15936K)] 9469K->8668K(25920K) [PSPermGen: 1980K->1980K(16384K)], 0,0718830 secs] [Times: user=0,10 sys=0,00, real=0,07 secs]
9,457: [Full GC [PSYoungGen: 352K->0K(9152K)] [ParOldGen: 16003K->13835K(21888K)] 16355K->13835K(31040K) [PSPermGen: 1980K->1980K(16384K)], 0,0782570 secs] [Times: user=0,10 sys=0,00, real=0,07 secs]

Multicore, Parallelisierung

Serial



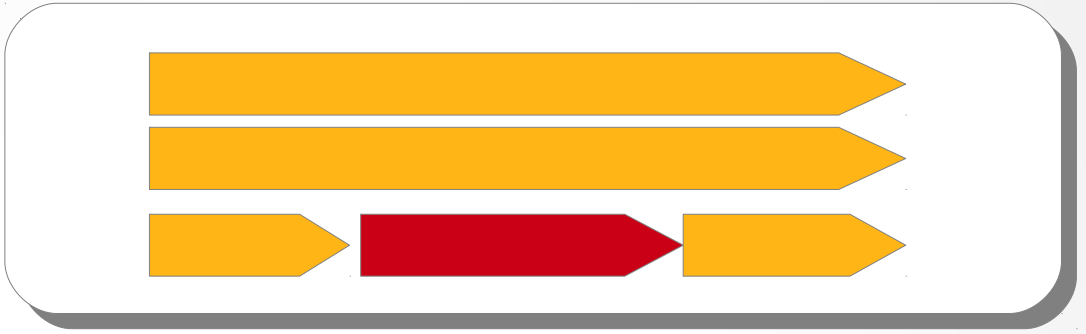
Parallel



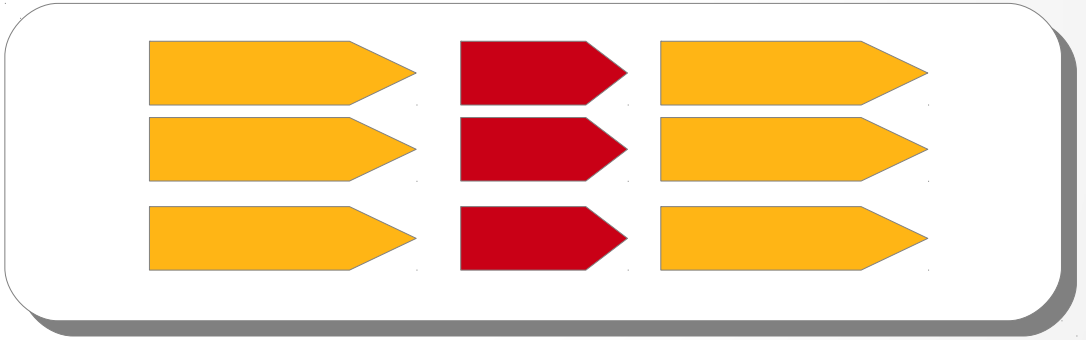
-XX:+UseParallelGC / -XX:+UseParallelOldGC

Concurrency

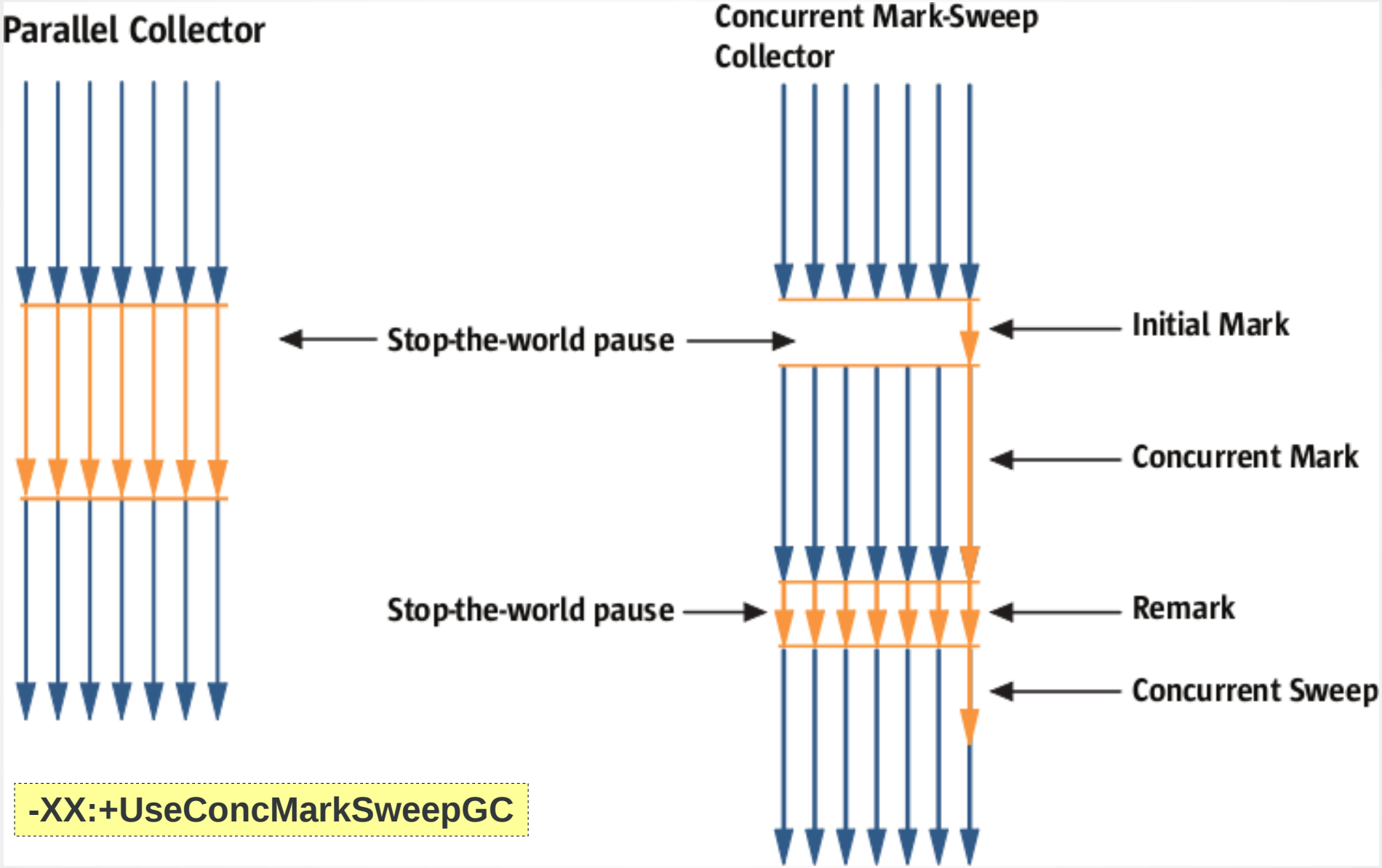
Concurrent



Parallel



Concurrent Mark and Sweep



-XX:+UseConcMarkSweepGC

CMS GC – Log I

108238.954: [GC 108238.954: [ParNew: 1561405K->38199K(2831168K), 0.1625150 secs] 2210735K->687532K(12268352K), 0.1628820 secs]
[Times: user=0.30 sys=0.00, real=0.16 secs]

108239.124: [GC [1 **CMS-initial-mark**: 649332K(9437184K)] 687558K(12268352K), 0.0279780 secs]
[Times: user=0.02 sys=0.00, real=0.03 secs]

108239.152: [**CMS-concurrent-mark-start**]
108240.555: [**CMS-concurrent-mark**: 1.403/1.403 secs]
[Times: user=1.40 sys=0.00, real=1.40 secs]

108240.555: [**CMS-concurrent-preclean-start**]
108240.600: [**CMS-concurrent-preclean**: 0.045/0.045 secs]
[Times: user=0.04 sys=0.00, real=0.04 secs]

CMS GC – Log II

108240.600: [**CMS-concurrent-abortable-preclean-start**]
CMS: abort preclean due to time
108245.863: [**CMS-concurrent-abortable-preclean**: 3.937/5.263 secs]
[Times: user=3.84 sys=0.00, real=5.26 secs]

108245.865: [GC[YG occupancy: 61713 K (2831168 K)]108245.865:
[Rescan (parallel) , 0.0785890 secs]108245.943: [weak refs processing,
0.0000770 secs]108245.944: [scrub string table, 0.0039600 secs] [1
CMS-remark: 649332K(9437184K)] 711045K(12268352K), 0.0828430
secs] [Times: user=0.07 sys=0.00, real=0.08 secs]

108245.948: [**CMS-concurrent-sweep-start**]
108246.333: [**CMS-concurrent-sweep**: 0.385/0.385 secs]
[Times: user=0.38 sys=0.00, real=0.38 secs]

108246.334: [**CMS-concurrent-reset-start**]
108246.365: [**CMS-concurrent-reset**: 0.032/0.032 secs]
[Times: user=0.04 sys=0.00, real=0.04 secs]

Situation bisher

- Throughput Collector (TP)
  - optimiert auf Durchsatz
  - selbst-tunend
  - De-Fragmentierung zuschaltbar / neuer Default
- Concurrent Mark and Sweep (CMS)
  - optimiert auf Antwortzeit
  - De-Fragmentierung nur in STW-Full-GCs
  - Heuristik spielt entscheidende Rolle
  - Tuning von Hand notwendig

Garbage First

- Wunschliste:
  - stabil
  - kurze oder keine Pausenzeiten
  - vorhersagbares Verhalten
  - überwachbar (Monitoring)
  - De-Fragmentierung
  - skalierbar (Multicore)
  - kein Tuning notwendig

neue GC-Strategie

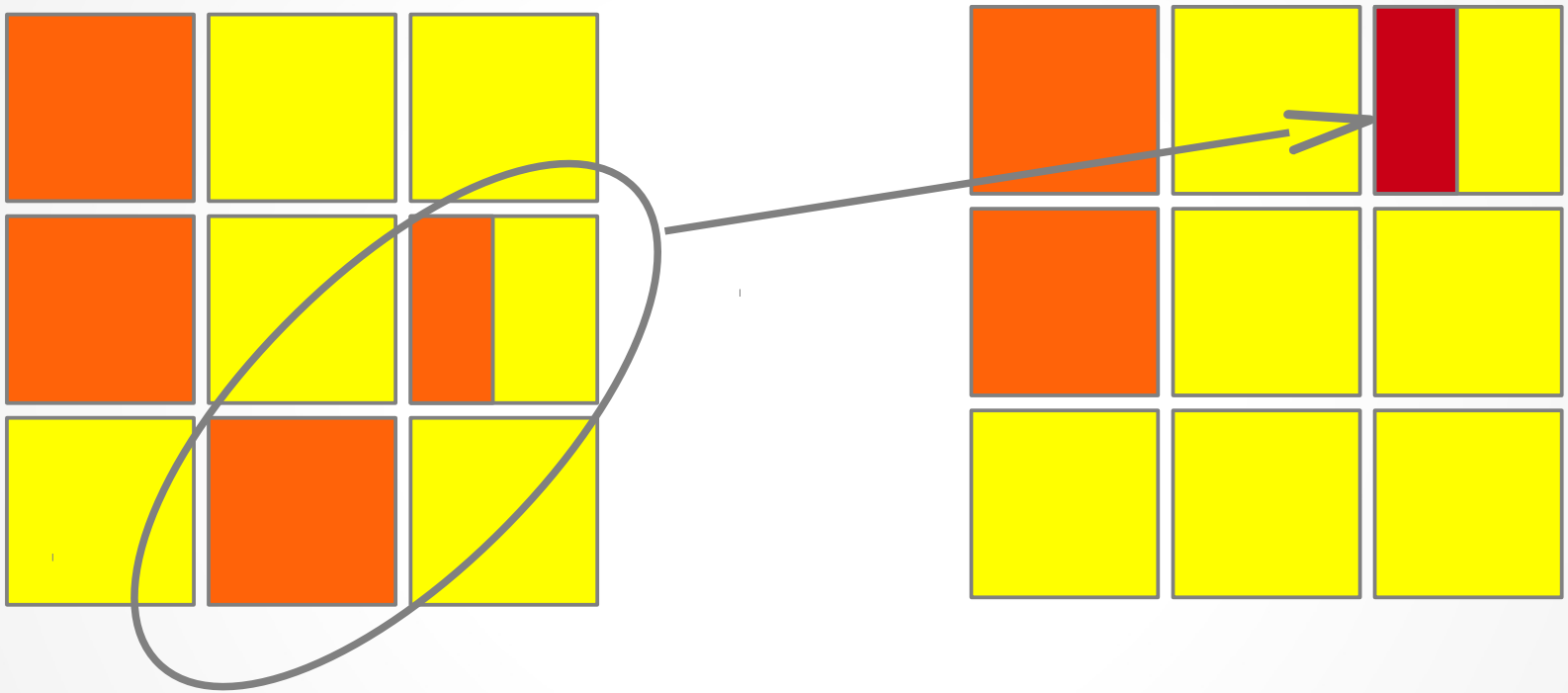
- Risiken
  - JVM-Crashes (→ hs_err<pid>.pid)
  - Performance
    - reduzierter Durchsatz?
    - Antwortzeit <-> Pausen

Garbage First G1 - Theorie

- Feste Einteilung des Speichers in Regions
  - min. 1 MB
  - Ziel ca. 2000 Regions
- Auswahl der „am meisten Erfolg versprechenden“ Regions für GC-Lauf
- Aufräumen:
  - Evakuierung der noch lebenden Objekte („Evacuation Pause“)
  - leere Region ist dann wieder verfügbar

Garbage First

- 1MB Regions



-XX:+UseG1GC

Garbage First - Praxis

- Young- und Old-Generation
- Concurrent Marking und Evacuation Pauses
- nicht mehr experimentell, bis dahin viele Crashes produzierbar
- rege Aktivität auf der Code-Basis
- Vorteil gegenüber CMS:
  - Defragmentierung
  - kein Tuning
- Full-GC wenn Speicher nicht mehr ausreicht
  - keine Parallelisierung!

G1 GC – Log: Evacuation

```
222515.738: [GC pause (young), 0.38581200 secs]
   [Parallel Time: 370.2 ms]
      [GC Worker Start (ms):  222515738.3  222515738.4
       Avg: 222515738.4, Min: 222515738.3, Max: 222515738.4, Diff:   0.0]
      [Ext Root Scanning (ms):  43.2  43.1
       Avg:  43.2, Min:  43.1, Max:  43.2, Diff:   0.1]
      [Update RS (ms):  7.2  8.7
       Avg:   8.0, Min:   7.2, Max:   8.7, Diff:   1.4]
         [Processed Buffers : 80 69
          Sum: 149, Avg: 74, Min: 69, Max: 80, Diff: 11]
      [Scan RS (ms):  17.0  17.0
       Avg:  17.0, Min:  17.0, Max:  17.0, Diff:   0.0]
      [Object Copy (ms):  297.0  295.6
       Avg: 296.3, Min: 295.6, Max: 297.0, Diff:   1.4]
      [Termination (ms):  0.0  0.0
       Avg:   0.0, Min:   0.0, Max:   0.0, Diff:   0.0]
         [Termination Attempts : 1 1
          Sum: 2, Avg: 1, Min: 1, Max: 1, Diff: 0]
...
```

G1 GC – Log: Evacuation

```
...
      [GC Worker End (ms):  222516102.8  222516102.8
       Avg: 222516102.8, Min: 222516102.8, Max: 222516102.8, Diff:   0.0]
      [GC Worker (ms):  364.5  364.5
       Avg: 364.5, Min: 364.5, Max: 364.5, Diff:   0.0]
      [GC Worker Other (ms):  5.8  5.8
       Avg:   5.8, Min:   5.8, Max:   5.8, Diff:   0.0]
   [Clear CT:   0.9 ms]
   [Other:  14.7 ms]
      [Choose CSet:   0.1 ms]
      [Ref Proc:  11.2 ms]
      [Ref Enq:   0.1 ms]
      [Free CSet:   2.6 ms]
   [Eden: 2284M(2284M)->0B(2284M) Survivors: 172M->172M
    Heap: 7461M(12288M)->5189M(12288M)]
 [Times: user=0.75 sys=0.00, real=0.38 secs]
```

G1 GC – Log: Concurrent

222879.431: [GC pause (young) (initial-mark), 0.62468900 secs]
...
[Eden: 2272M(2272M)->0B(2272M) Survivors: 184M->184M Heap: 7555M(12288M)->5306M(12288M)]
[Times: user=1.19 sys=0.01, real=0.63 secs]

222880.056: [GC **concurrent-root-region-scan-start**]
222880.560: [GC **concurrent-root-region-scan-end**, 0.5045520]

222880.560: [GC **concurrent-mark-start**]
222885.610: [GC **concurrent-mark-end**, 5.0490610 sec]

222885.611: [GC **remark** 222885.612: [GC ref-proc, 0.0143660 secs], 0.0829630 secs]
[Times: user=0.07 sys=0.00, real=0.07 secs]

222885.699: [GC **cleanup** 5375M->4924M(12288M), 0.0548790 secs]
[Times: user=0.07 sys=0.00, real=0.05 secs]

222885.754: [GC **concurrent-cleanup-start**]
222885.755: [GC **concurrent-cleanup-end**, 0.0008480]

G1 GC – Log: Evacuation

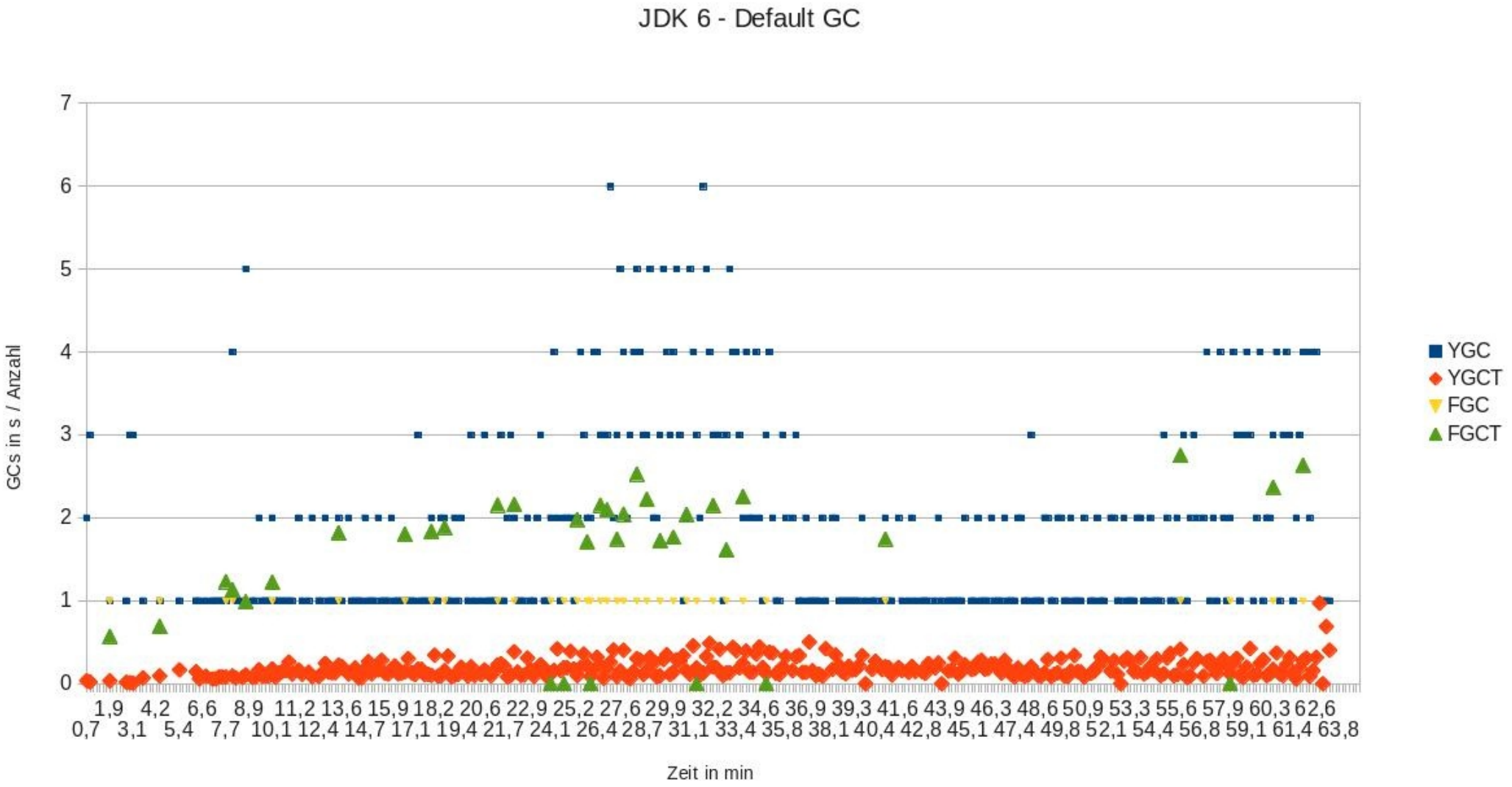
```
223007.294: [GC pause (mixed), 0.37419100 secs]
   [Parallel Time: 350.0 ms]
...
      [GC Worker End (ms):  223007638.5  223007638.5
       Avg: 223007638.5, Min: 223007638.5, Max: 223007638.5, Diff:   0.0]
      [GC Worker (ms):  342.6  342.6
       Avg: 342.6, Min: 342.6, Max: 342.6, Diff:   0.0]
      [GC Worker Other (ms):  7.4  7.5
       Avg:   7.4, Min:   7.4, Max:   7.5, Diff:   0.0]
   [Clear CT:   1.3 ms]
   [Other:  22.9 ms]
      [Choose CSet:   1.9 ms]
      [Ref Proc:  15.1 ms]
      [Ref Enq:   0.1 ms]
      [Free CSet:   5.0 ms]
   [Eden: 2316M(2316M)->0B(2300M) Survivors: 140M->156M
    Heap: 7160M(12288M)->3988M(12288M)]
 [Times: user=0.71 sys=0.00, real=0.37 secs]
```

Details unter https://blogs.oracle.com/poonam/

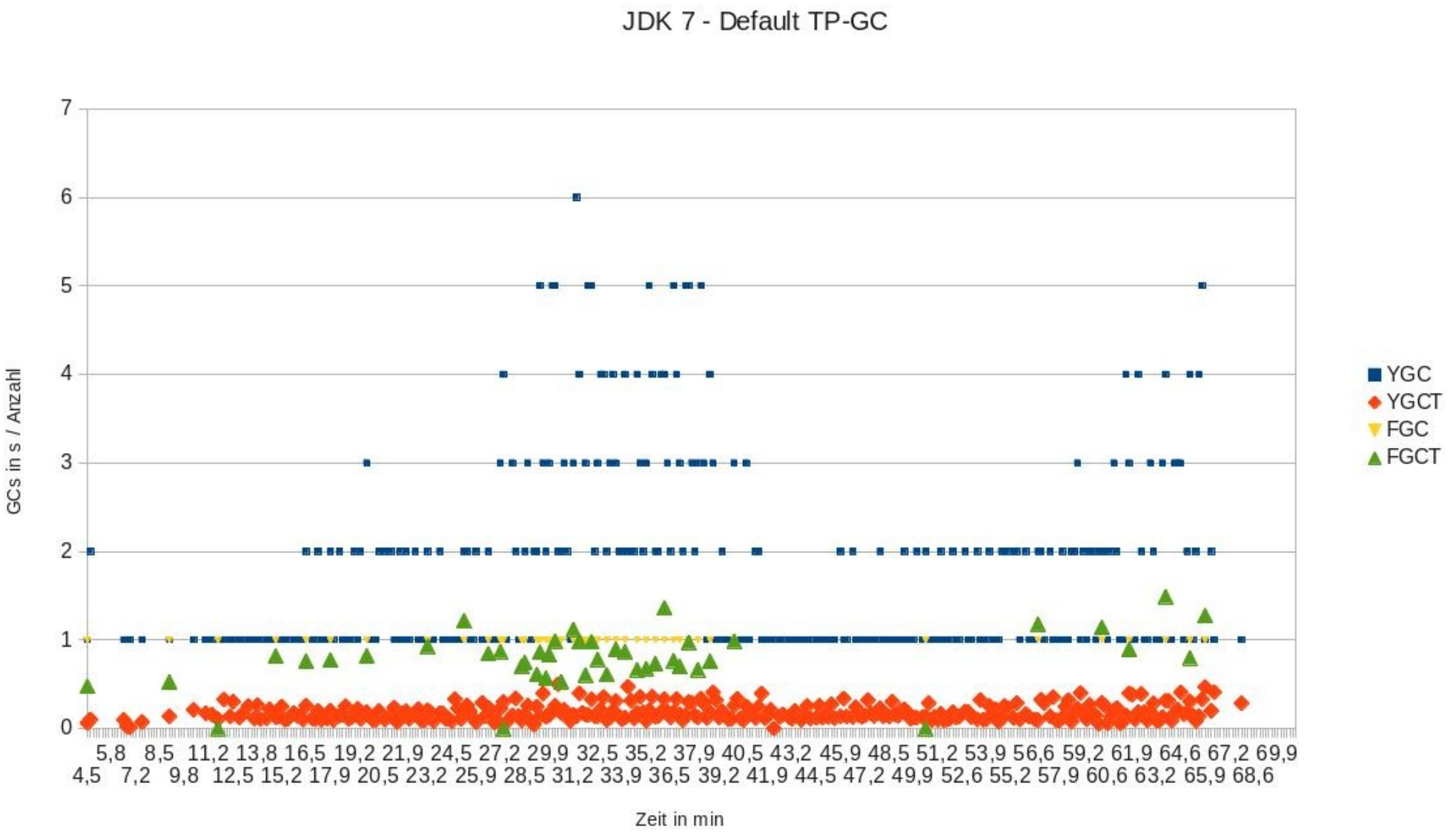
GC im Lasttest

- Lasttest mit rein interaktiver (HTTP) Last
- Linux-VM mit 12GB RAM, 6 VCPU
- JVM mit -Xmx8g
- Testläufe:
  - JDK 6 Update 31, TP-Collector
  - JDK 7 Update 7, TP-Collector
  - JDK 7 Update 7, G1-Collector (-XX:+UseG1GC)
- Aufbereitung der „jstat -gc“-Ausgabe mit 10-Sekunden-Intervall

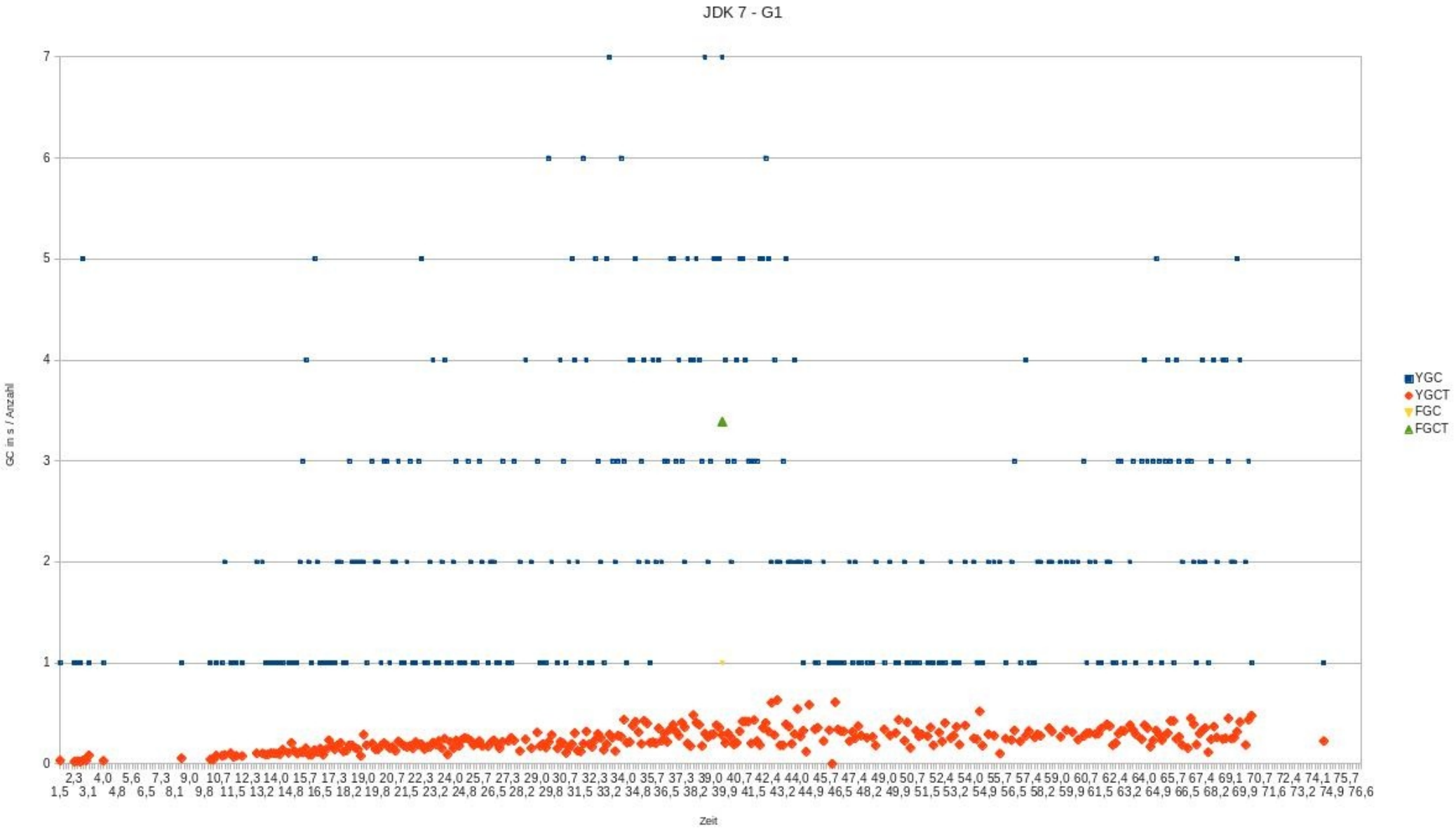
JDK 6, TP-Collector



JDK 7, TP-Collector



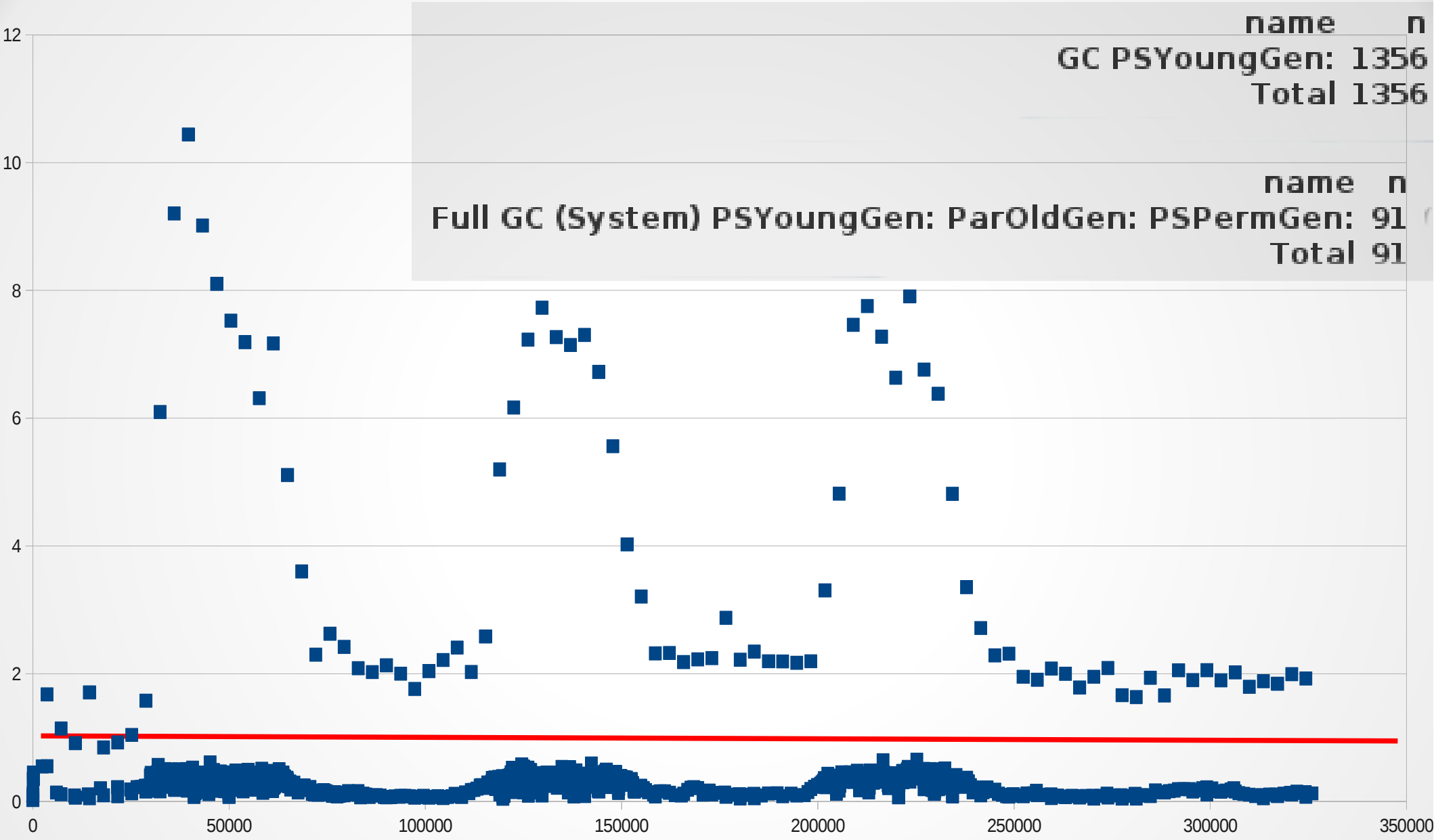
JDK 7, G1-Collector



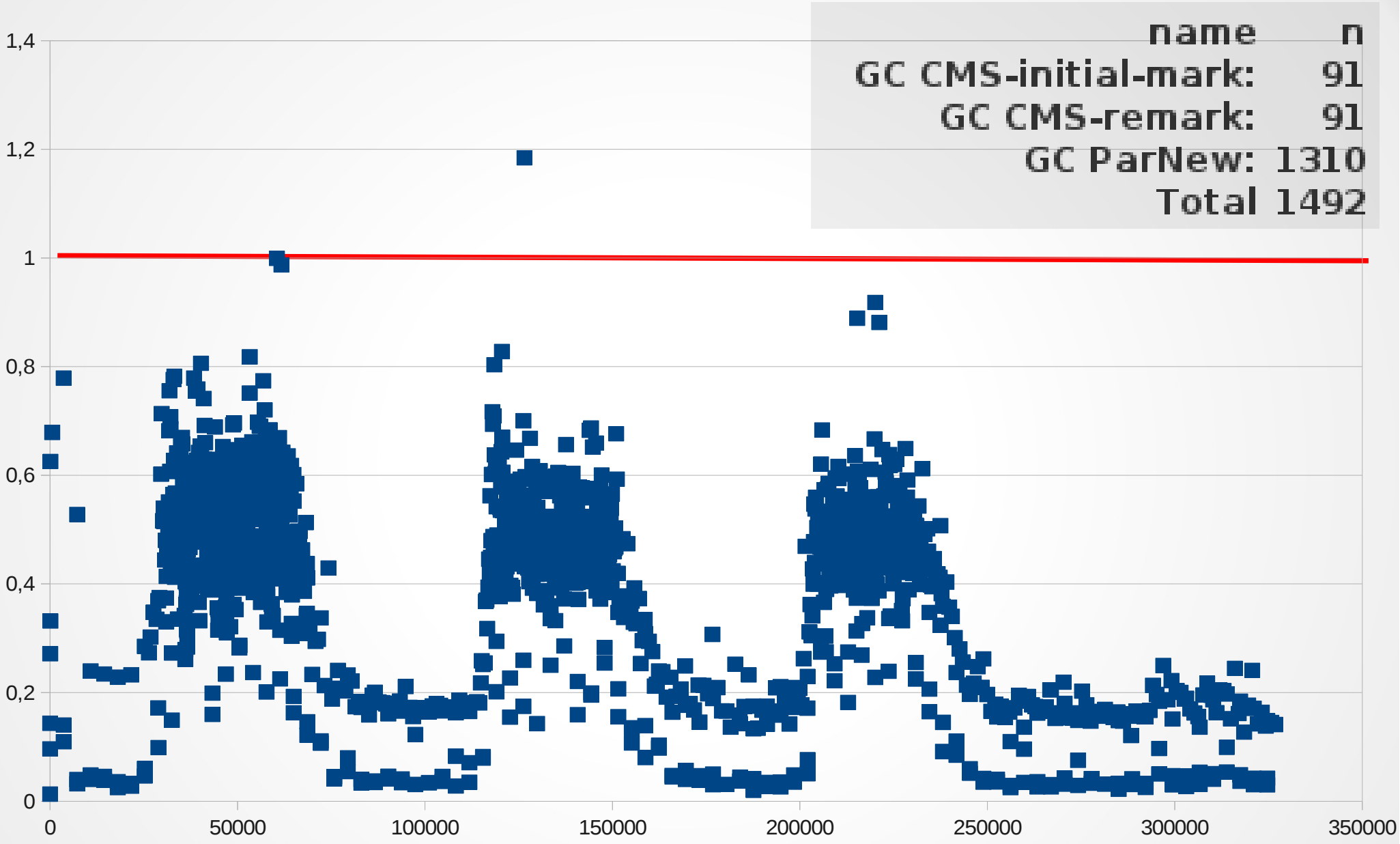
Vergleich TP / CMS / G1

- Produktivsystem
  - 4 VMs
- kein Speichermangel
- Auswertung des GC-Log (-XX:+PrintGCDetails)

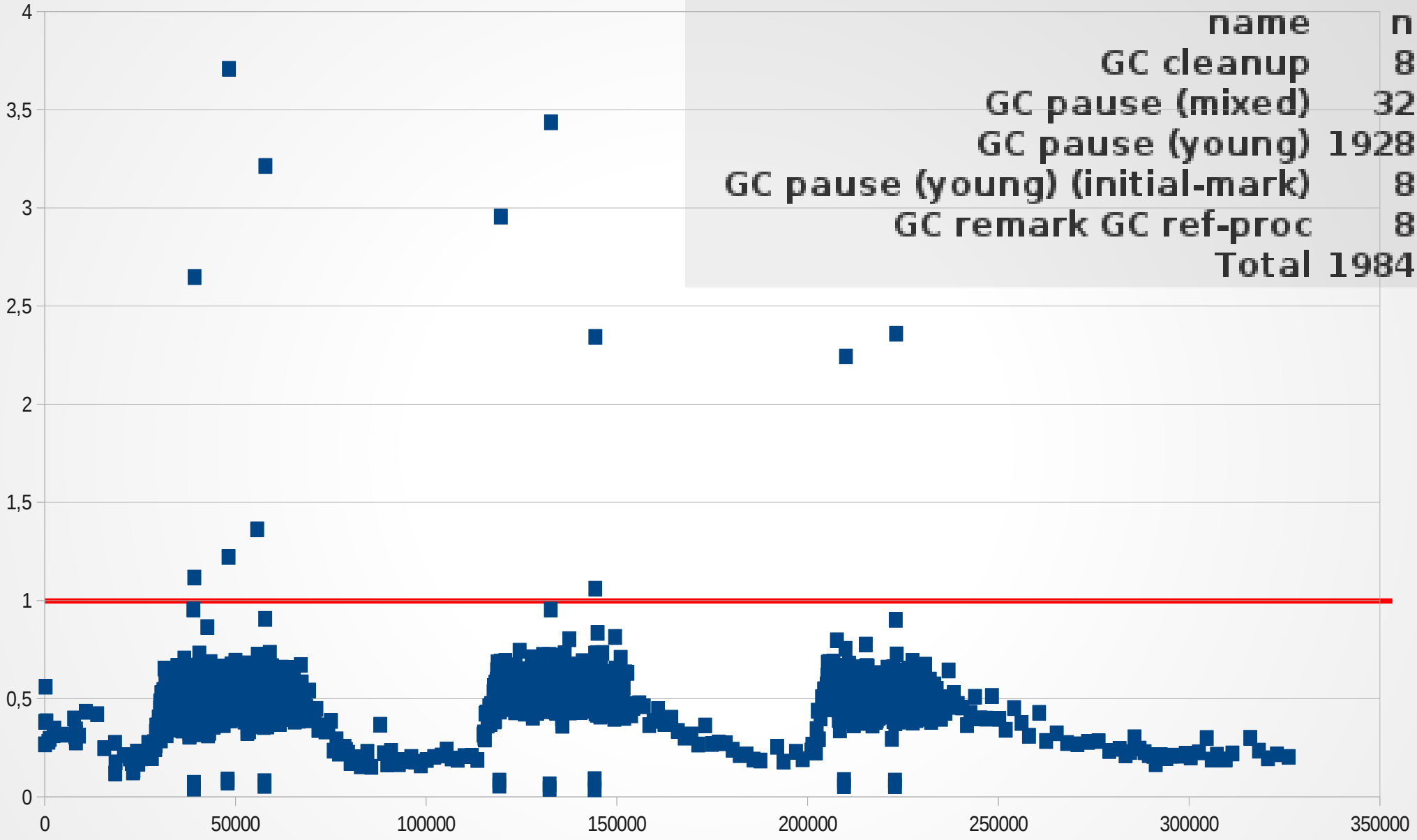
TP – GC in Produktion



CMS – GC in Produktion



G1 – GC in Produktion



name	n
GC cleanup	8
GC pause (mixed)	32
GC pause (young)	1928
GC pause (young) (initial-mark)	8
GC remark GC ref-proc	8
Total	1984

Garbage First - Fazit

- Wunschliste:
  - stabil
  - kurze oder keine Pausenzeiten
  - vorhersagbares Verhalten
  - überwachbar (Monitoring)
  - De-Fragmentierung
  - skalierbar (Multicore)
  - kein Tuning notwendig

G1 Switches

- Alle JVM-Switches ausgeben:

  java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal -version

- Zielvorgabe für Pausen:

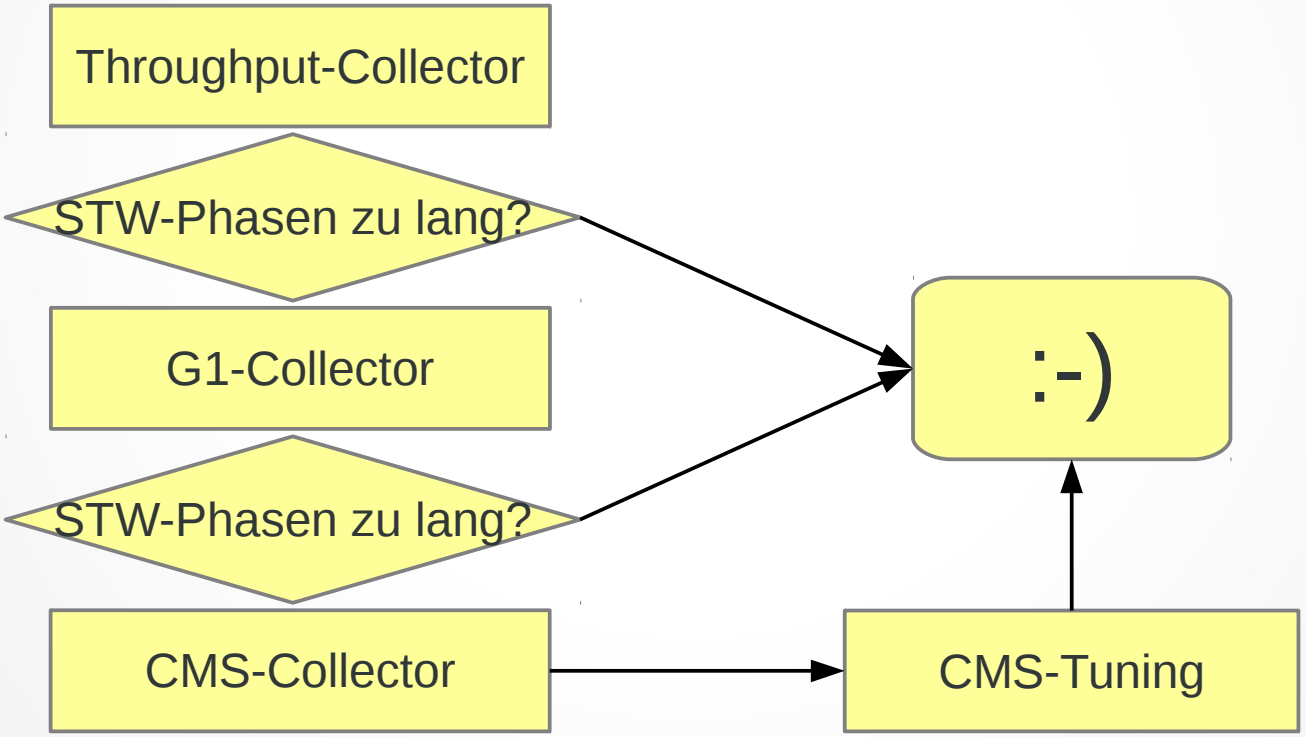
  -XX:MaxGCPauseMillis=200

- „Mixed“-Optimierung in JDK 8, zukünftig:

  -XX:G1MixedGCCountTarget=8

GC - Entscheidung

- „Don't try to fix me, I'm not broken“ (Evanescence)
- **Wirksamkeit** von Optimierung
  - Messen besser als Bauchgefühl



Resourcen

- Tools:
  - GCViewer: https://github.com/chewiebug/GCViewer
  - GCHisto: http://java.net/projects/gchisto
- Dokumentation:
  - GC Whitepaper: http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf
  - G1-Tuning: https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=6583