

Application Performance Management

Frühling 2021

Class Loading

Zoltán Majó

Agenda

Diskussion Übung und Finalisierung Java NIO

Ergänzungen GC

Besprechung Evaluation

Class Loading

Arbeitsblatt

Neben `position`, `limit` und `capacity` gibt es noch die Variable `mark`. Wozu dient sie und wie wird sie manipuliert? (Klasse `Buffer`)

Antwort: `mark` markiert die unterste Position im Buffer die noch bearbeitet werden soll. Es gilt immer:

$$0 \leq \text{mark} \leq \text{position} \leq \text{limit} \leq \text{capacity}$$

Arbeitsblatt

Frage 2: Was versteht man unter *gathering*? (GatheringByteChannel)

Antwort: Die Operation wenn mehrere Buffer auf einen Channel geschrieben werden heisst Gathering. Dabei wird der Inhalt der Buffer in ihrer Reihenfolge auf den Kanal geschrieben. Es ist zum Beachten, dass die einzelnen Buffer unterschiedliche Grössen haben können.

Arbeitsblatt

Frage 3: Was versteht man unter *scattering*? (ScatteringByteChannel)

Antwort: Das ist das Gegenteil von gathering. Daten aus einem Kanal werden in mehrere Buffer geschrieben. Ist der erste Buffer voll, so wird der zweite gefüllt, bis dieser voll ist etc.

Arbeitsblatt

Frage 4: Was ist ein *direct buffer*? Wozu dient er? (ByteBuffer)

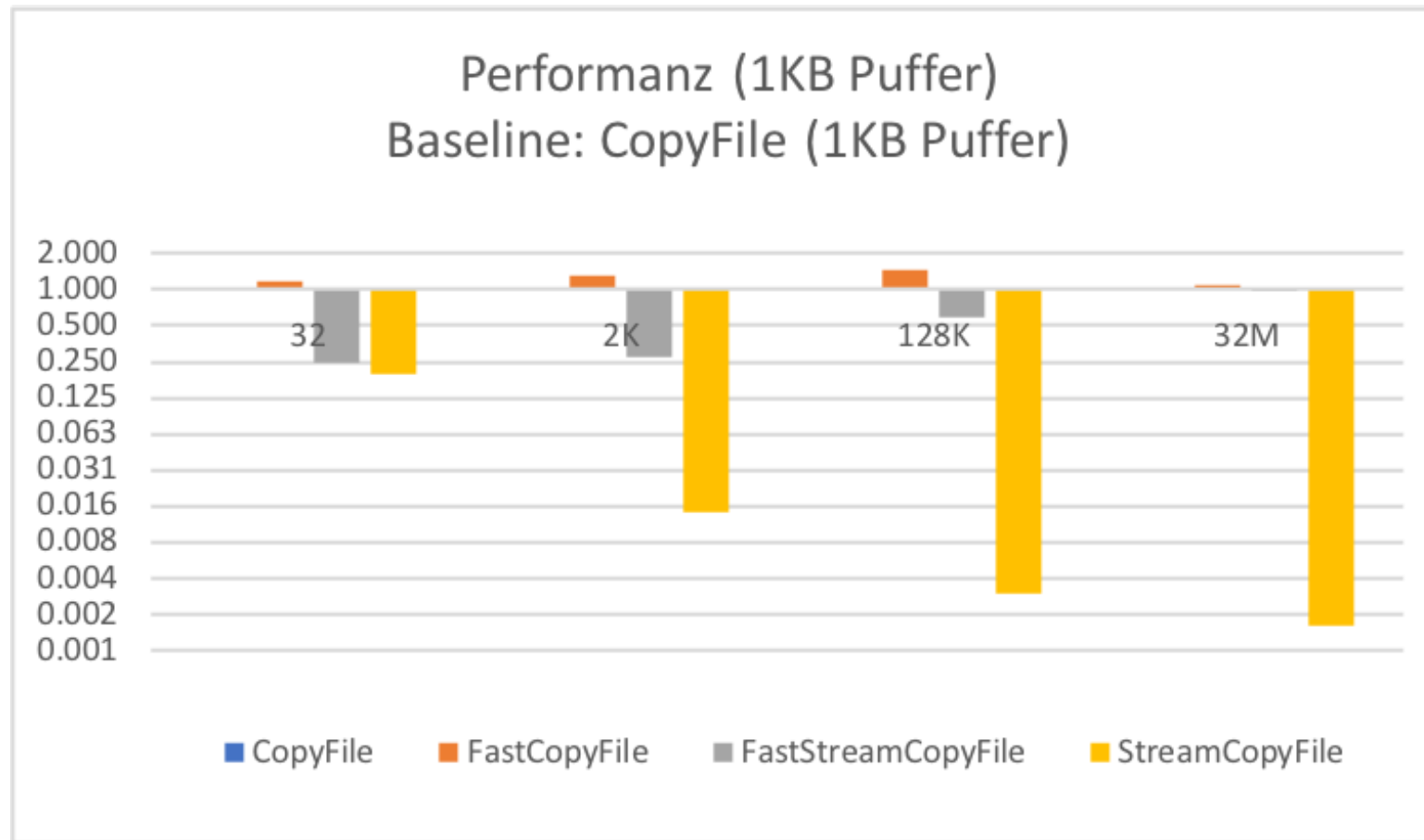
Antwort: Direct Buffers werden meist vom Betriebssystem verwaltet und deren Speicher kann sich auch ausserhalb der JVM befinden.

Arbeitsblatt

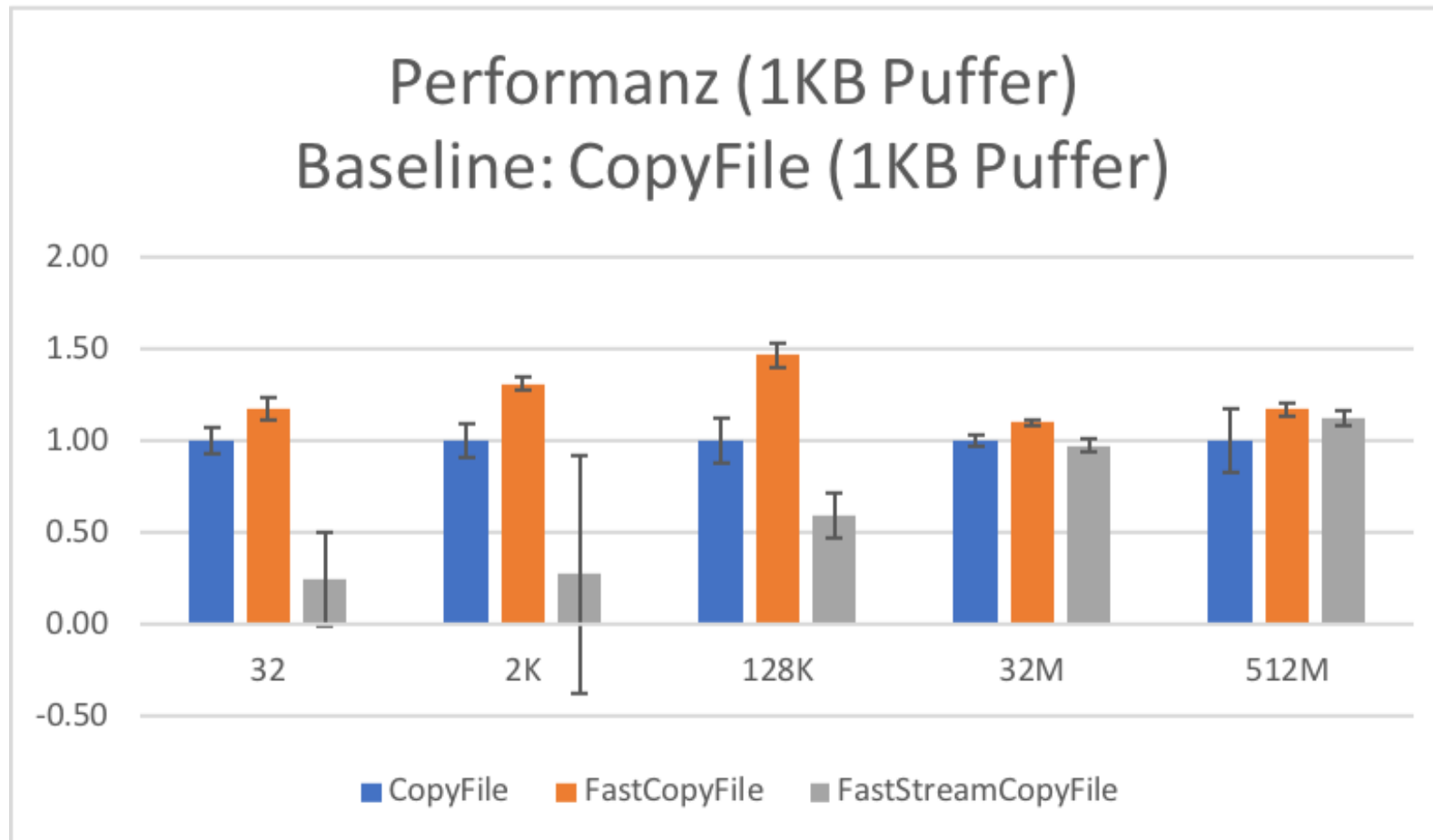
Frage 5: Wozu dient *buffer slicing*? (slice() in ByteBuffer)

Antwort: Damit kann ein Teil des Buffers wiederum als Buffer verwendet werden. Die Daten – der darunterliegende Byte-Array – werden aber von beiden Buffer geteilt.

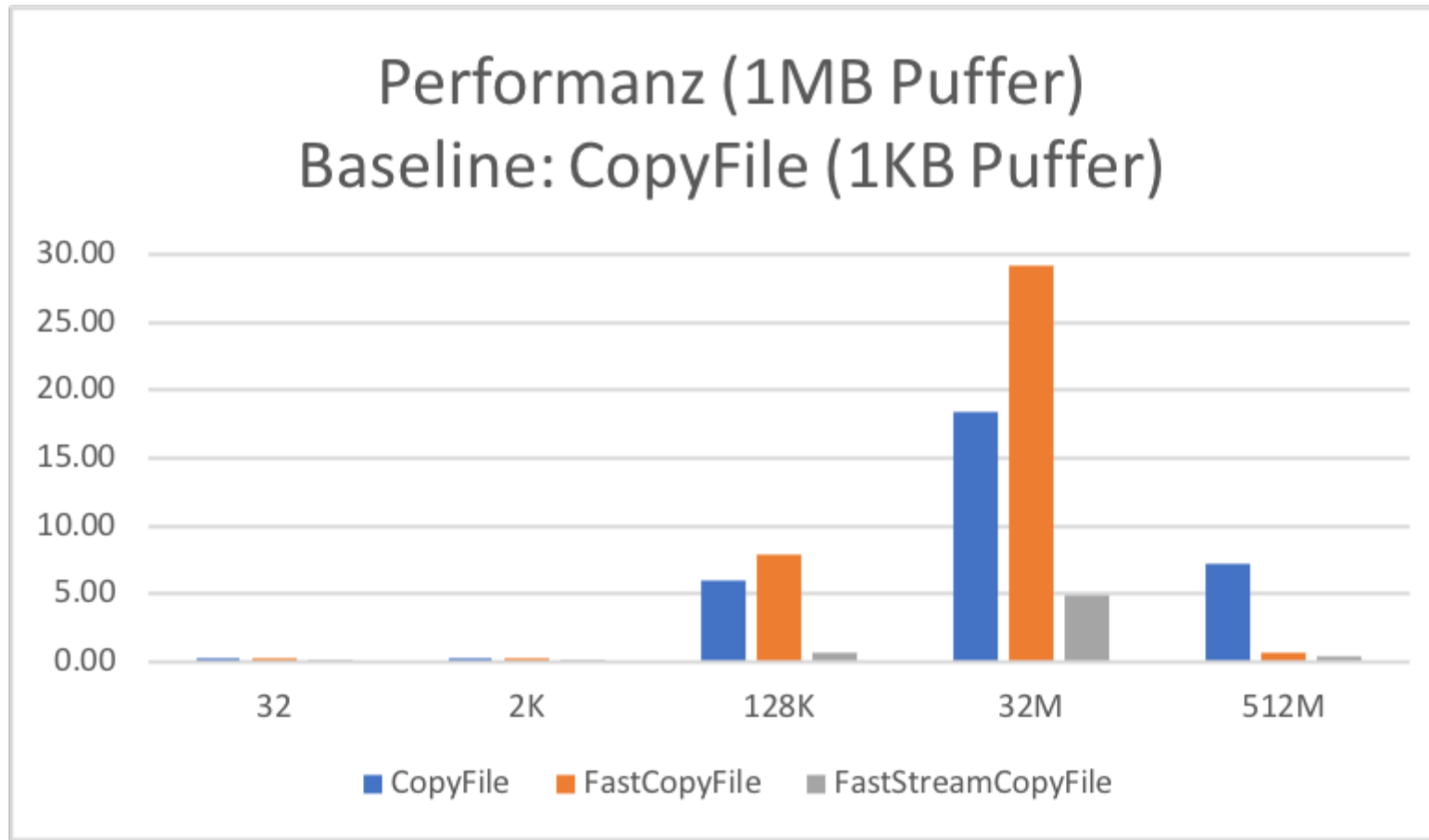
Frage 7



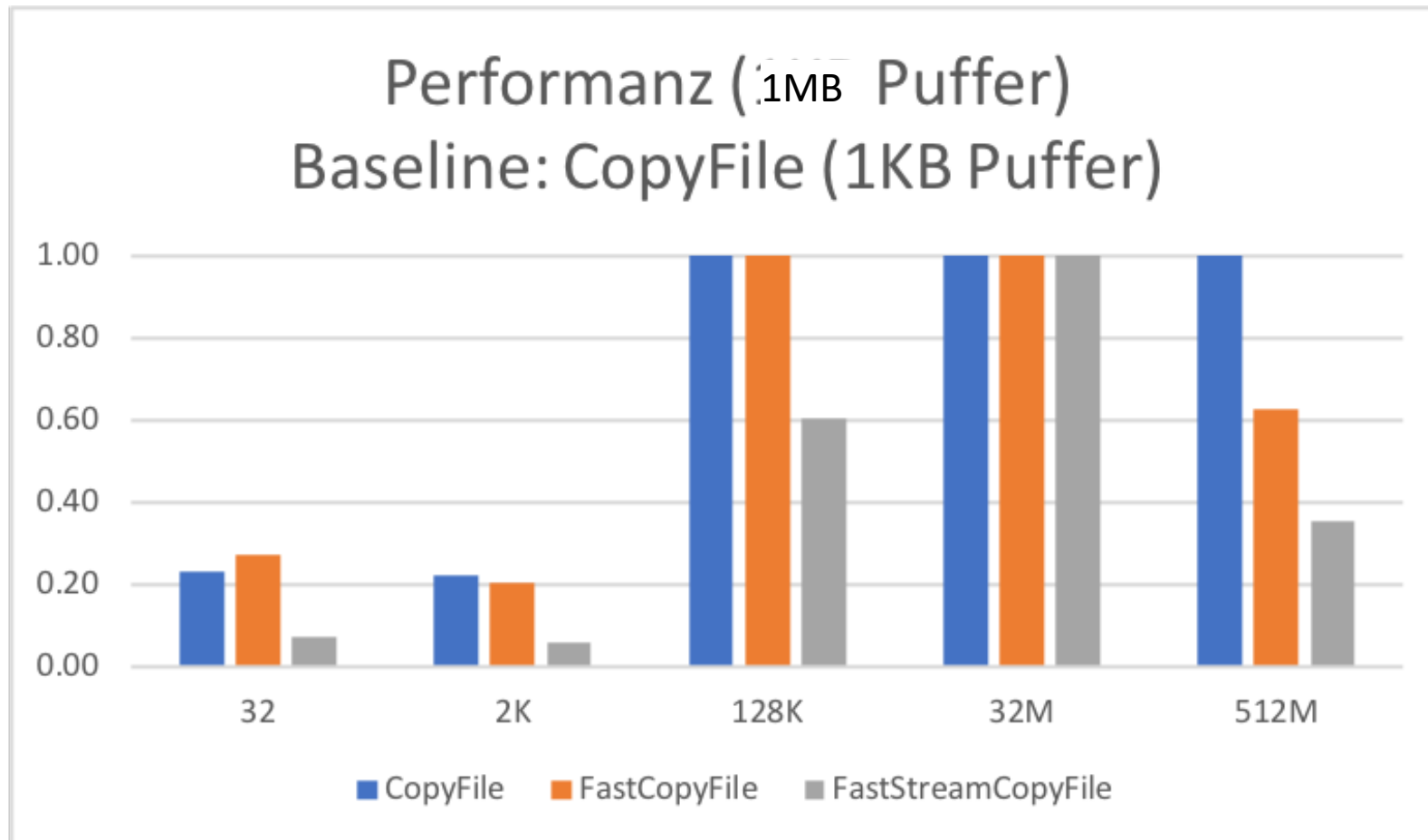
Frage 7



Frage 7



Frage 7



Ergänzung 1: Algorithmus Mark and Compact

7.6. INTRODUCTION TO TRACE-BASED COLLECTION

```
/* mark */  
1) Unscanned = set of objects referenced by the root set;  
2) while (Unscanned  $\neq \emptyset$ ) {  
3)   remove object o from Unscanned;  
4)   for (each object o' referenced in o) {  
5)     if (o' is unreached) {  
6)       mark o' as reached;  
7)       put o' on list Unscanned;  
     }  
   }  
}
```



all heap

```
/* compute new locations */  
8) free = starting location of heap storage;  
9) for (each chunk of memory o in the heap, from the low end) {  
10)   if (o is reached) {  
11)     NewLocation(o) = free;  
12)     free = free + sizeof(o);  
   }  
}
```



```
/* retarget references and move reached objects */  
13) for (each chunk of memory o in the heap, from the low end) {  
14)   if (o is reached) {  
15)     for (each reference o.r in o)  
16)       o.r = NewLocation(o.r);  
17)     copy o to NewLocation(o);  
   }  
}  
18) for (each reference r in the root set)  
19)   r = NewLocation(r);
```

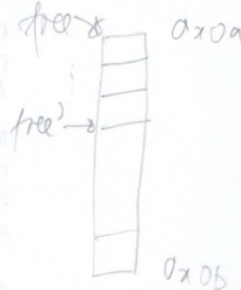
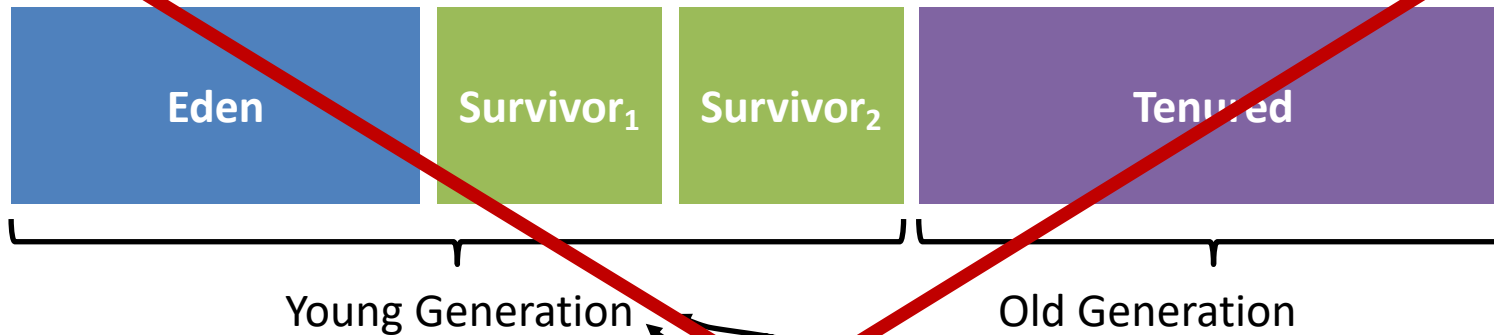


Figure 7.26: A Mark-and-Compact Collector

Ergänzung 2: Minor / Major / Full GC

Heap Layout



Minor GC

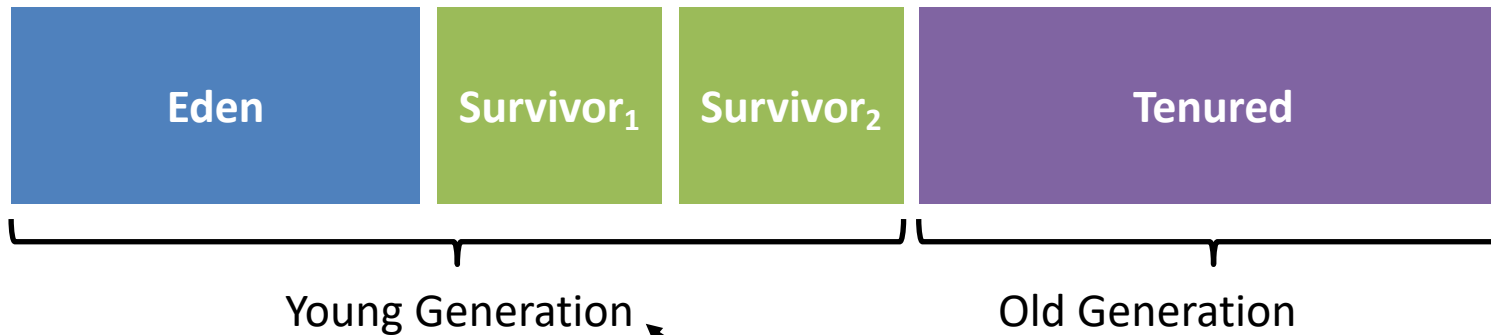
- Nur young Generation wird collected
- Mark and Copy Algorithmus

Major GC

- Young und old Generation wird collected
- Mark and Compact Algorithmus

Ergänzung 2: Minor / Major / Full GC

Heap Layout



Full GC {

Minor GC

- Nur young Generation wird collected
- Mark and Copy Algorithmus

Major GC

- Old Generation wird collected
- Mark and Compact Algorithmus

Besprechung Evaluation

Class Loading

Script