PRESENTED BY RED HAT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

TROUBLESHOOTING JBOSS EAP 5: PART 2

Rich Raposa
JBoss Curriculum Manager
Red Hat, Inc.
May 6, 2011

Troubleshooting JBoss EAP 5, Part 2

- In my previous talk, I covered the lifecycle of a client request and discussed various areas where a bottleneck could occur.
- In this talk, I will discuss the memory of a Java Virtual Machine and Garbage Collection.

JVM Settings in run.conf

- -Xms
- -Xmx
- -XX:MaxPermSize

Java Code consists of *.class files

- Java classes are loaded by a JVM's **ClassLoader**
- For each class that is loaded, an object (comparable to metadata) gets created of type...
  - **java.lang.Class**
- The **Class** objects are almost never unloaded, and reside in memory for the lifetime of your Java application.
- **What part of memory do these Class files reside in?**

PermGen (The Permanent Generation)

- No matter how high you set your **-Xmx**, large applications that load lots of classes get **OutOfMemoryError**'s when the **PermGen** fills up.
- The size is set by the **-XX:MaxPermSize** parameter:
- The default value in **run.conf** is:
  - **-XX:MaxPermSize=256m**
- Monitor the **PermGen** size (using **JON**, **JConsole**, etc.) after your application has been running for a while, and make sure it has plenty of room.

Java Applications contain lots of Objects

- Java objects are created using the **new** operator.
- Where do Java objects reside in memory?
  - On the **Heap**.
- How long do these objects consume memory on the heap?
  - Until the **Garbage Collector** comes by and removes them.
- An object becomes eligible for garbage collection when it is no longer reachable within the application.

The Stack

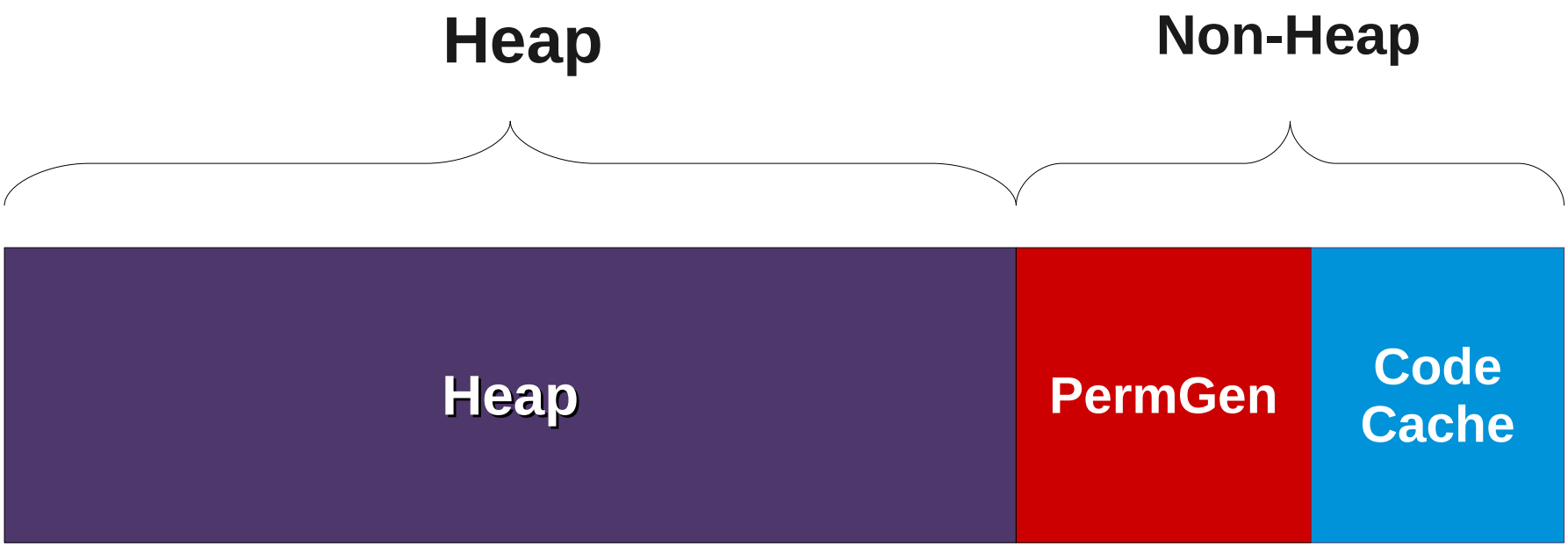
- Java applications contain many **threads**.
- Each thread can perform a specific task “simultaneously” as other threads.
- The threads need their own memory space, and all the threads together consume a portion of memory referred to as the **stack**.

The Code Cache

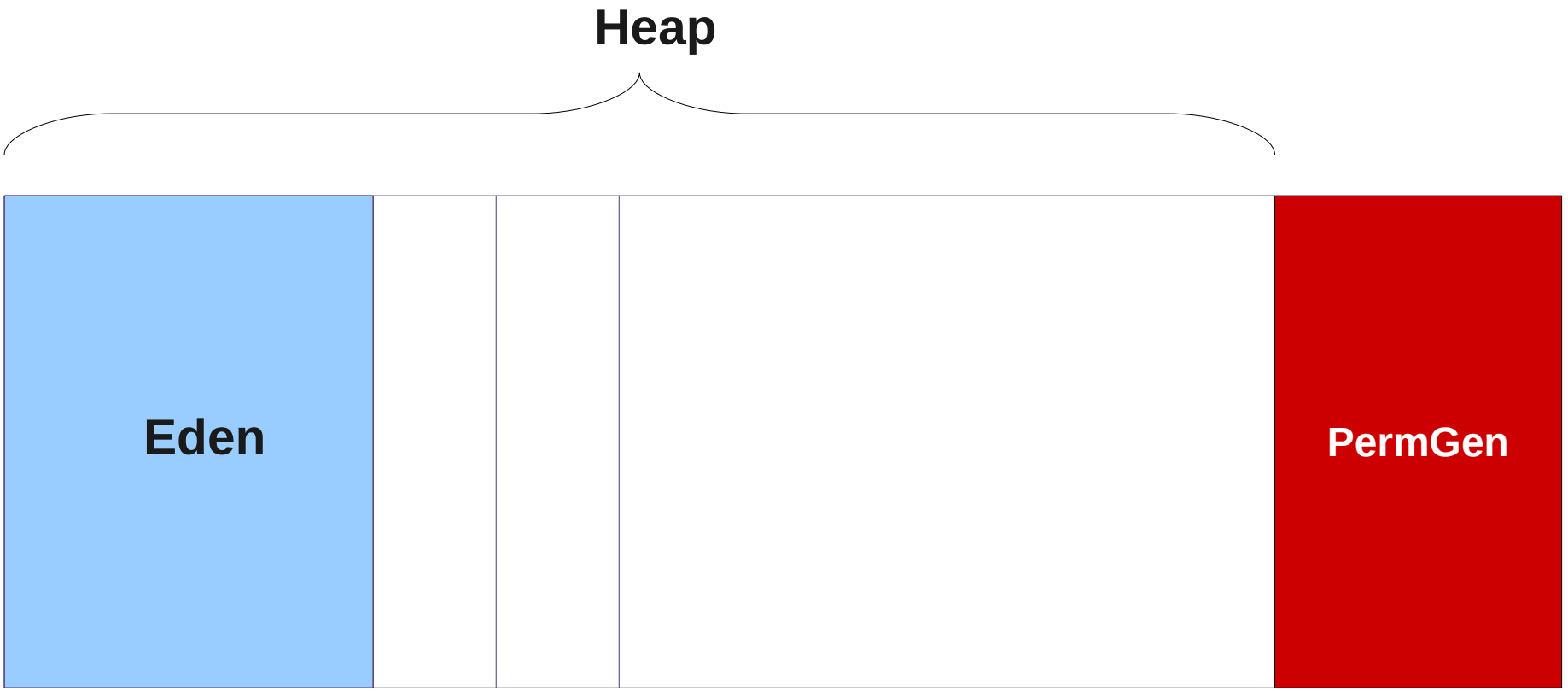
- Another area of memory in the JVM is called the Code Cache.
  - Not part of the Heap.
  - JIT compiled code is stored there.
- Configure using **-XX:ReservedCodeCacheSize**

The JVM's Memory

The total memory your JVM consumes is:
Heap + PermGen + Code Cache/Stack



Let's look at the Generations that makeup the Heap



Eden memory is where new objects reside in memory.
This is also known as the “Young Generation”.

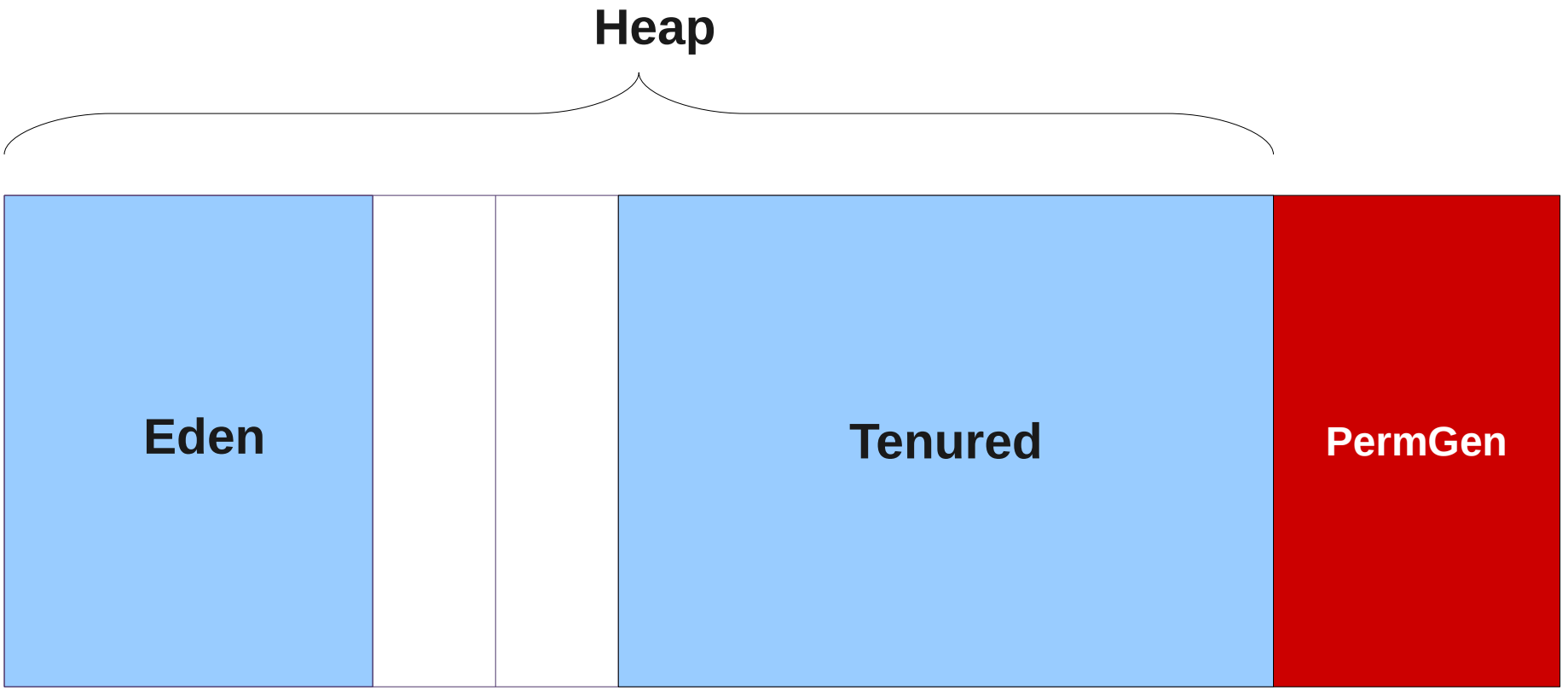
The Eden Generation

- **Minor collections** occur in the Eden space.
- The bigger the Eden space is, the less frequently minor garbage collections will occur.
- This may not be a good thing!
- You want the garbage collector to remove an object in Eden space if it can.
- Objects that survive the Eden space get moved into the **Tenured Generation**.

Configuring the Eden Generation

- **-XX:NewRatio** is the ratio of Eden space to Tenured space.
- For example:
  - **-XX:NewRatio=1** means Eden will equal Tenured
  - **-XX:NewRatio=3** means Eden will be ¼ of the Tenured
- You can also set the size of Eden specifically:
  - **NewSize** is the initial size (and lower limit) of Eden
  - **MaxNewSize** is the max size of Eden

The Tenured Generation



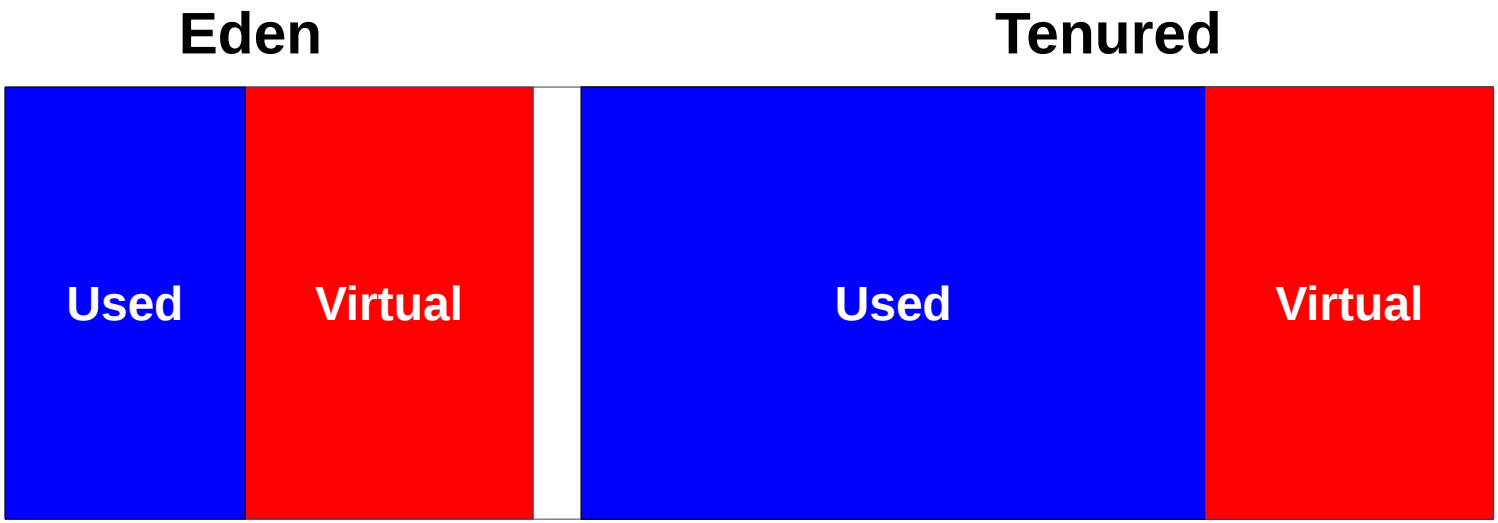
Tenured memory is for objects that survive Garbage Collection.
This is also known as the “Young Generation”.

Major Collections

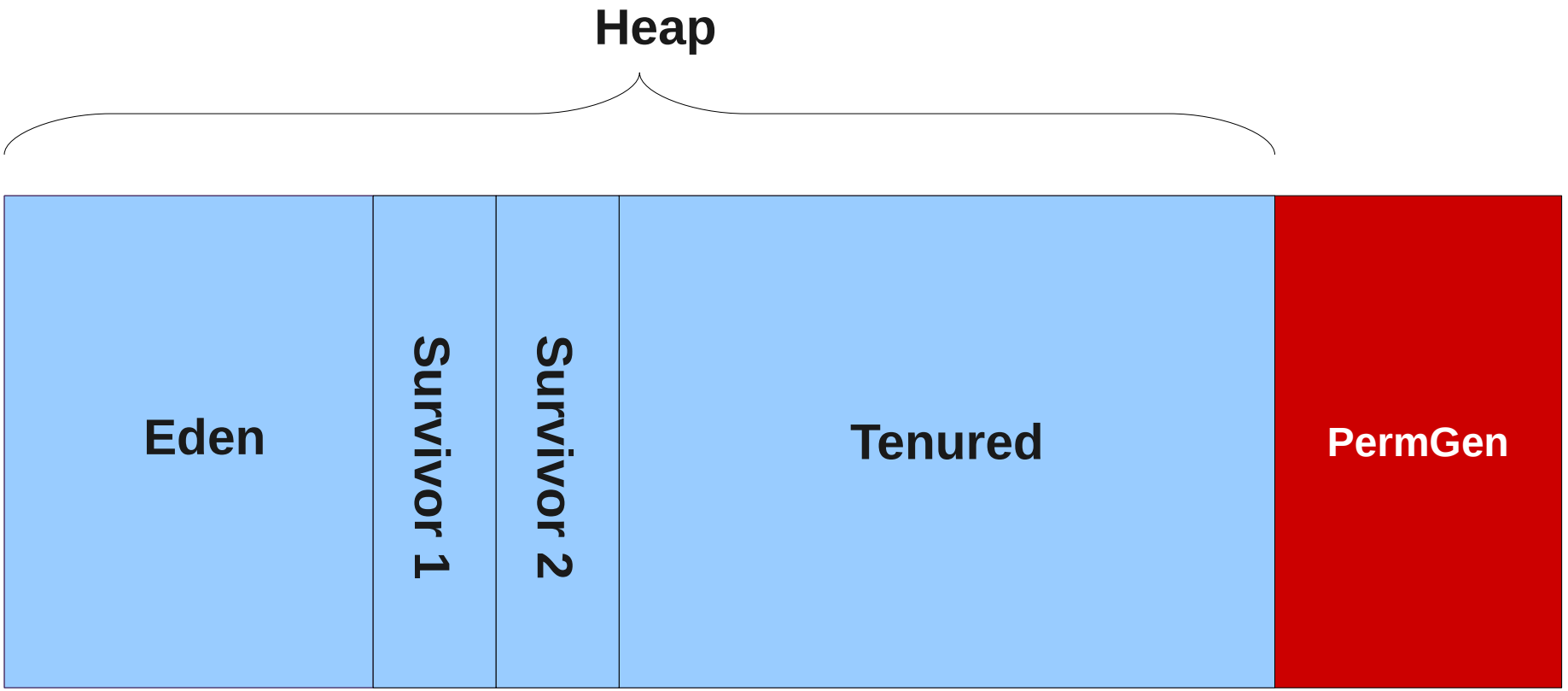
- What's important to understand about the Tenured Generation is that the **Garbage Collector** does **major collections** in the Tenured space.
- It is these major collections that cause pauses in your JBoss applications.
- Later in my talk, I will discuss the various types of **Garbage Collection algorithms** that are used to help minimize these pauses, based on the needs of your specific applications.

Configuring the Tenured Generation

- After sizing Eden, the Tenured generation gets whatever space remains based on **Xmx**.
- You can improve performance by setting **Xms** and **Xmx** to be the same value, which causes the Eden and Tenured spaces to remain a fixed size (no virtual spaces for the JVM to try to resize and manage).



The Survivor Spaces



Objects moving to the Tenured Generation pass through an area known as the Survivor space.

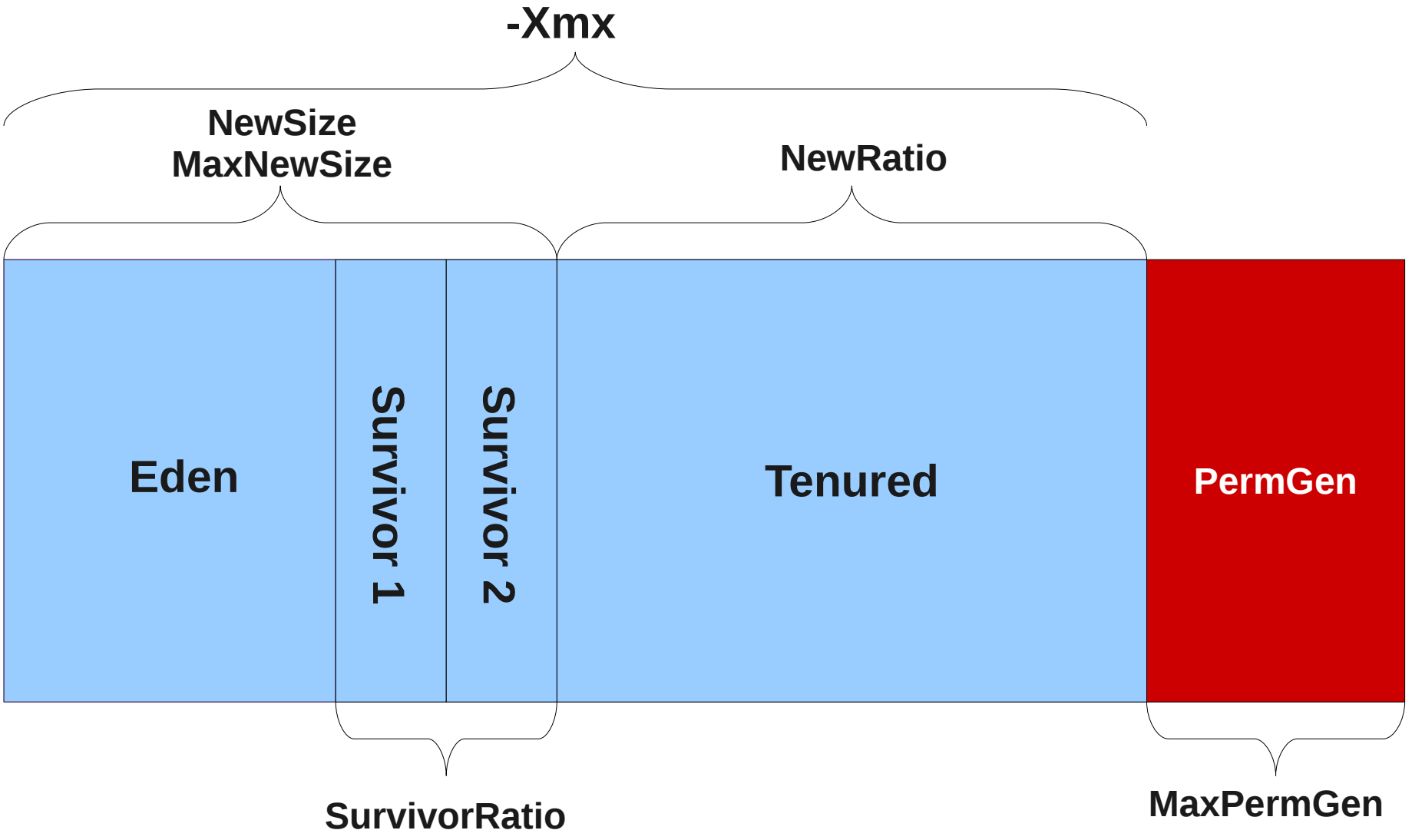
Understanding Survivor Space

- There are two Survivor spaces of equal size.
- Each space takes turns being used by the JVM to temporarily store objects that survive Eden.
- At any given time, one survivor space contains the objects moving from Eden to Tenured,
- and the other survivor space is empty.
- The Survivor space is actually a portion of the memory allocated by Eden.

Configuring Survivor Space

- You will rarely need to configure survivor space.
- The configuration flag is:
  - **-XX:SurvivorRatio**
- This the ratio of each Survivor space to the entire Young Generation space.
- For exampe:
  - **-XX:SurvivorRatio=8**
- With an 8 to 1 ratio, each Survivor space will be 1/10th and Eden will be 8/10th of the entire Young Generation.

Review of Memory Settings



Configuring the Stack Size

- Stack size is configured using **-Xss**
- The default value is typically:
  - **-Xss1024k**
- Keep in mind that each thread in your application has its own stack, so each thread can consume 1024k.
- The maximum memory consumed by the entire stack space will be:
  - **(#num of threads) x (stack size)**
- **java.lang.StackOverflowError** means your stack size is not big enough!

Garbage Collection

- Now that you have seen the terminology for a JVM's memory...
- Let's take a look at the Garbage Collector.

Configuring Garbage Collection (GC)

- GC runs in a low-priority thread on your JVM.
- There are different algorithms for determining when and how the garbage collector does its job.
    1. **Serial collector** (typically for small data sets)
    2. **Parallel collector** (for medium to large data sets)
    3. **Concurrent collector** (for medium to large data sets with minimal pauses)

The Serial Collector

- Configured using **-XX:UseSerialGC**
- Uses a single thread to perform all garbage collection work, and is efficient for single-processor machines.
- But can also be used on multi-processor machines with **small data sets**.
- Use the Serial Collector when:
  - You have small data sets
  - You have a single processor with no pause times
  - There are no pause time requirements

The Parallel Collector

- Also known as the **Throughput Collector**
- Configured using **-XX:+UseParallelGC**
- Performs **minor** collections in parallel on multi-processor **OR** multi-threaded machines.
- Use this GC algorithm with **medium to large data sets**.
- Use the Parallel Collector when:
  - Peak performance is the highest priority, and
  - Pause times of one second or longer are acceptable

Parallel Compaction

- When using the Parallel Collector, you can also specify **Parallel Compaction**:
  - **-XX:+UseParallelOldGC**
- This allows for major collections to occur in parallel.
- Without parallel compaction, major collections are performed using a single thread, which can significantly limit scalability.
- Use Parallel Compaction:
  - Whenever you use the Parallel Collector!

Specifying the Number of Parallel Threads

- Using **-XX:ParallelGCThreads**, you can limit the number of threads that the Parallel Collector uses to perform garbage collection.
- This allows you to guarantee a certain number of CPU's will be always be available for your application.

Ergonomics

- Refers to the **behavioral tuning** you can configure for the Parallel Collector, specifically:
  - Maximum pause times for GC
  - Throughput
- Use **-XX:MaxGCPauseMillis** to “hint” that pause times should not exceed a certain length of time.
- Use **-XX:GCTimeRatio** to set a “goal” ratio of time spent in GC vs. application time. For example:
  - **-XX:GCTimeRatio=99**
- 1% of time spent in GC, other 99% is application time.

The Concurrent Collector

- Configure using **-XX:+UseConcMarkSweepGC**
- Performs its work concurrently with your application.
- For applications with **medium to large data sets**.
- **Pause times** are kept to a minimum (to the detriment of application performance)
- Use the Concurrent Collector when:
  - Response time is more important than throughput
  - Pauses must be kept shorter than one second
  - You have a lot of processors

Incremental Mode for Concurrent Collection

- If low pause times are a requirement and you need to use the Concurrent Collector,
- but you **only have 1 or 2 processors** on your machine,
- then you can turn on **Incremental Mode** (used only for the Concurrent Collector):
  - **-XX:+CMSIncrementalMode**
- Divides the work done concurrently by the collector into small chunks of time which are scheduled between Young Generation collections.

Which Collector should I use?

- The only way to really determine which collector to use is to test each one individually with your application.
- Along with tuning your memory settings.
- There is an ideal configuration for your JBoss applications!
  - With proper allocation of Eden/Tenured/Perm space,
  - And proper selection of a GC algorithm,
  - You have a lot of options for fine-tuning JBoss and making it run the best on your environment!

Monitoring Memory

- Use your favorite monitoring tool:
  - JON
  - JConsole
  - JVisualVM
  - jmx-console
  - Many others...

Monitoring Garbage Collection

- You can obtain very specific details about the Garbage Collector and what it is doing.
  - **-verbose:gc**
    - [GC 325816K->83372K(776768K), 0.2454258 secs]
    - [Full GC 267628K->83769K(776768K), 1.8479984 secs]
  - **-XX:+PrintGCDetails**
    - [GC [DefNew: 64575K->959K(64576K), 0.0457646 secs] 196016K->133633K(261184K), 0.0459067 secs]
  - **-XX:+PrintGCTimeStamps**
    - 111.042: [GC 111.042: [DefNew: 8128K->8128K(8128K), 0.0000505 secs]111.042: [Tenured: 18154K->2311K(24576K), 0.1290354 secs] 26282K->2311K(32704K), 0.1293306 secs]

Thank you for coming!

- The Red Hat Curriculum Team hopes you enjoyed this year's Summit and JBossWorld!

JOIN THE CONVERSATIONS AND STAY IN TOUCH WITH JBOSS YEAR ROUND!

LIKE US ON FACEBOOK

facebook.com/jboss

FOLLOW US ON TWITTER

@JBossDeveloper
@JBossOperations
@JBossNews

READ THE BLOG

redhat.com/about/news/blog/
jboss.org/feeds/