SUMMIT

JBoss
WORLD

PRESENTED BY RED HAT

# LEARN. NETWORK.
# EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

# TROUBLESHOOTING

# JBOSS EAP 5: PART 1

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

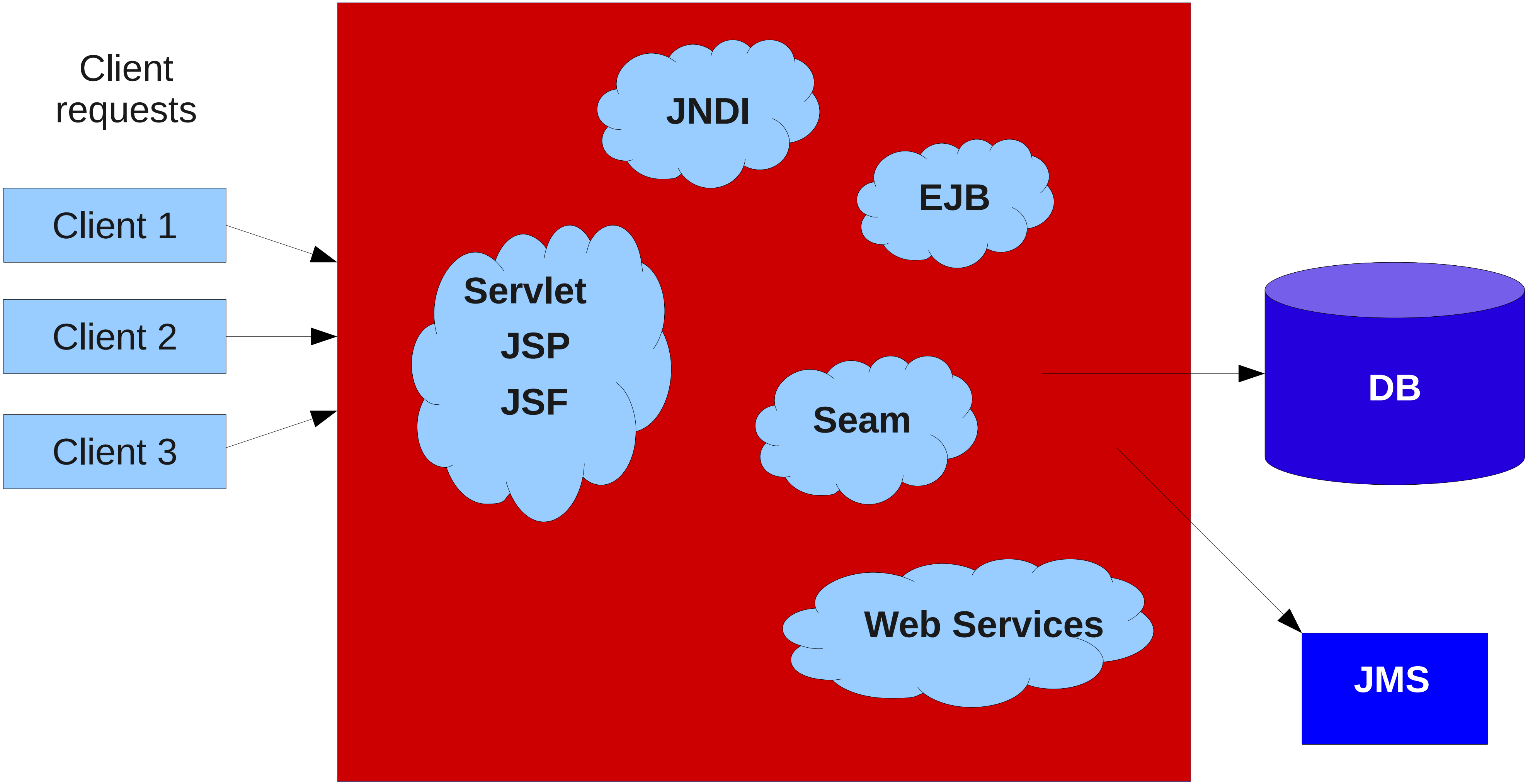
# Troubleshooting JBoss EAP 5, Part 1

- There are often hiccups on the way to production.
- Identifying potential issues and resolving them are important for server administrators and developers alike.
- My talks today will focus on two main elements:
  1) Common Bottlenecks
  2) JBoss Memory Consumption

# Common Bottlenecks

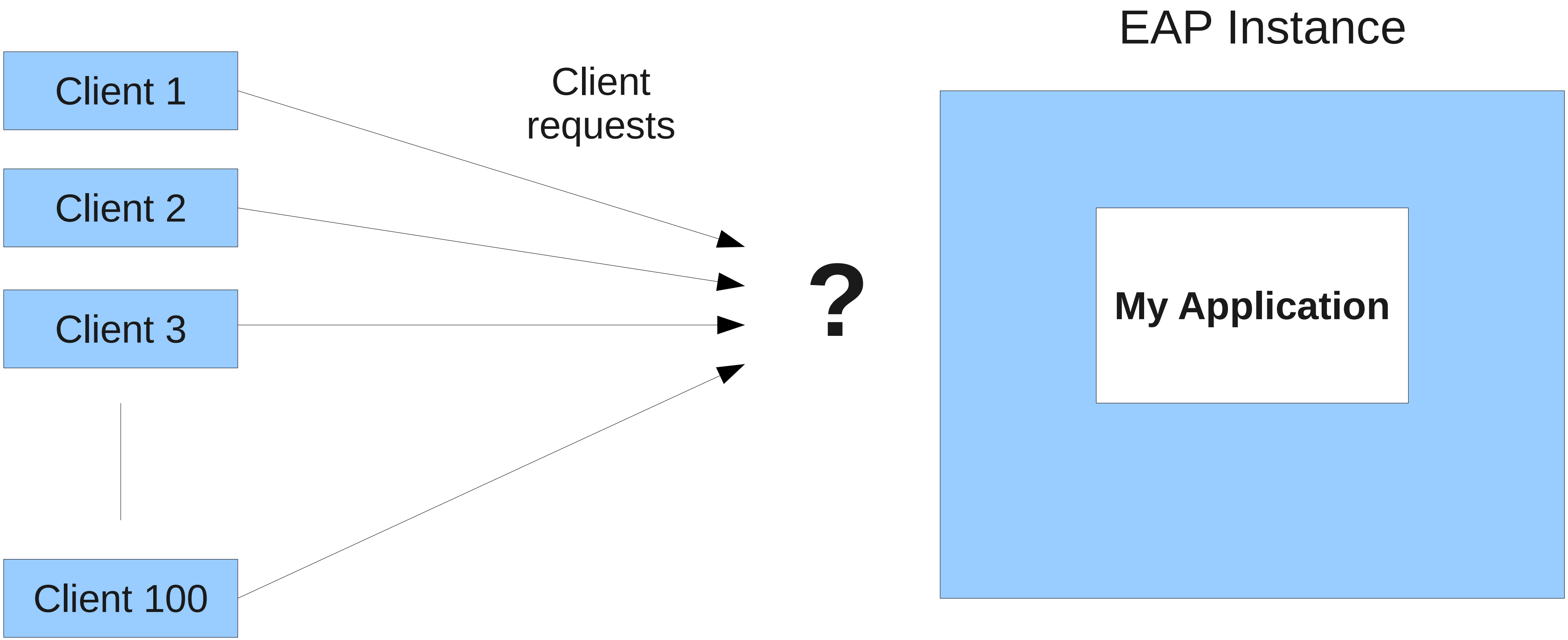
- Bottlenecks can occur in many places:
  - 75% of performance issues originate from the application
  - Database connections
  - EJBs/Servlets/Web components
  - The Java Virtual Machine (JVM)
  - Logging
  - Any areas causing bottlenecks for you?

# JBoss EAP

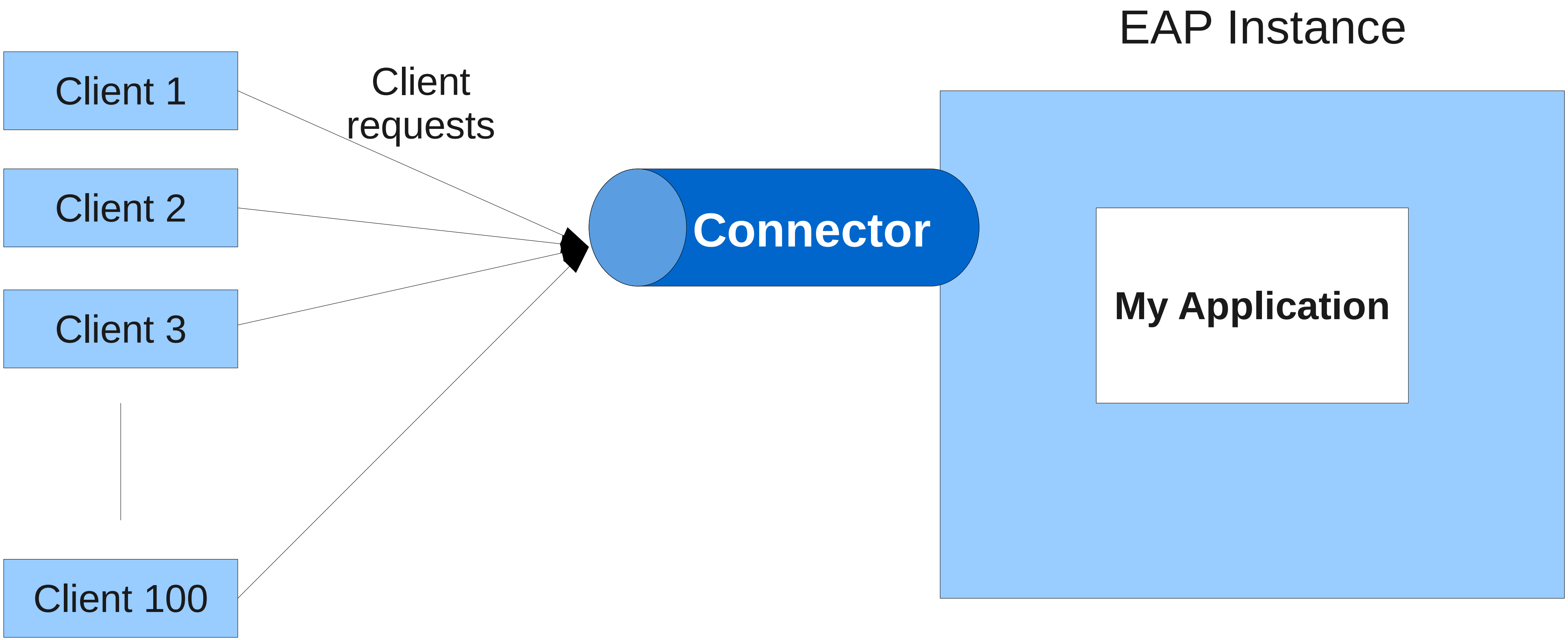


Client requests
Client 1
Client 2
Client 3
JNDI
EJB
Servlet
JSP
JSF
Seam
Web Services
DB
JMS

# Client Requests (HTTP or AJP)



# The Connector



# What is the configuration file for modifying the settings of your HTTP or AJP Connectors?

- The file name is **server.xml**
- Located in:
  - /jboss-as/server/<profile-name>/deploy/jbossweb.sar

# The <Connector> Tag

The default settings look like:

```
<Connector protocol="HTTP/1.1" port="8080"
    address="${jboss.bind.address}"
    connectionTimeout="20000" redirectPort="8443" />
```

**Let's take a look at some important Connector attributes...**

# acceptCount

- **acceptCount**
  - The maximum queue length for incoming requests.
  - Default is 100
  - Any requests received when the queue is full will be refused.
  - **Make sure this value is large enough to handle the number of requests that you expect at peak times.**
  - Careful: it's not the number of clients, but the **number of requests**, which is typically much higher than the number of clients.

# maxKeepAliveRequests

Keep the connection between client and server
open between requests

Client

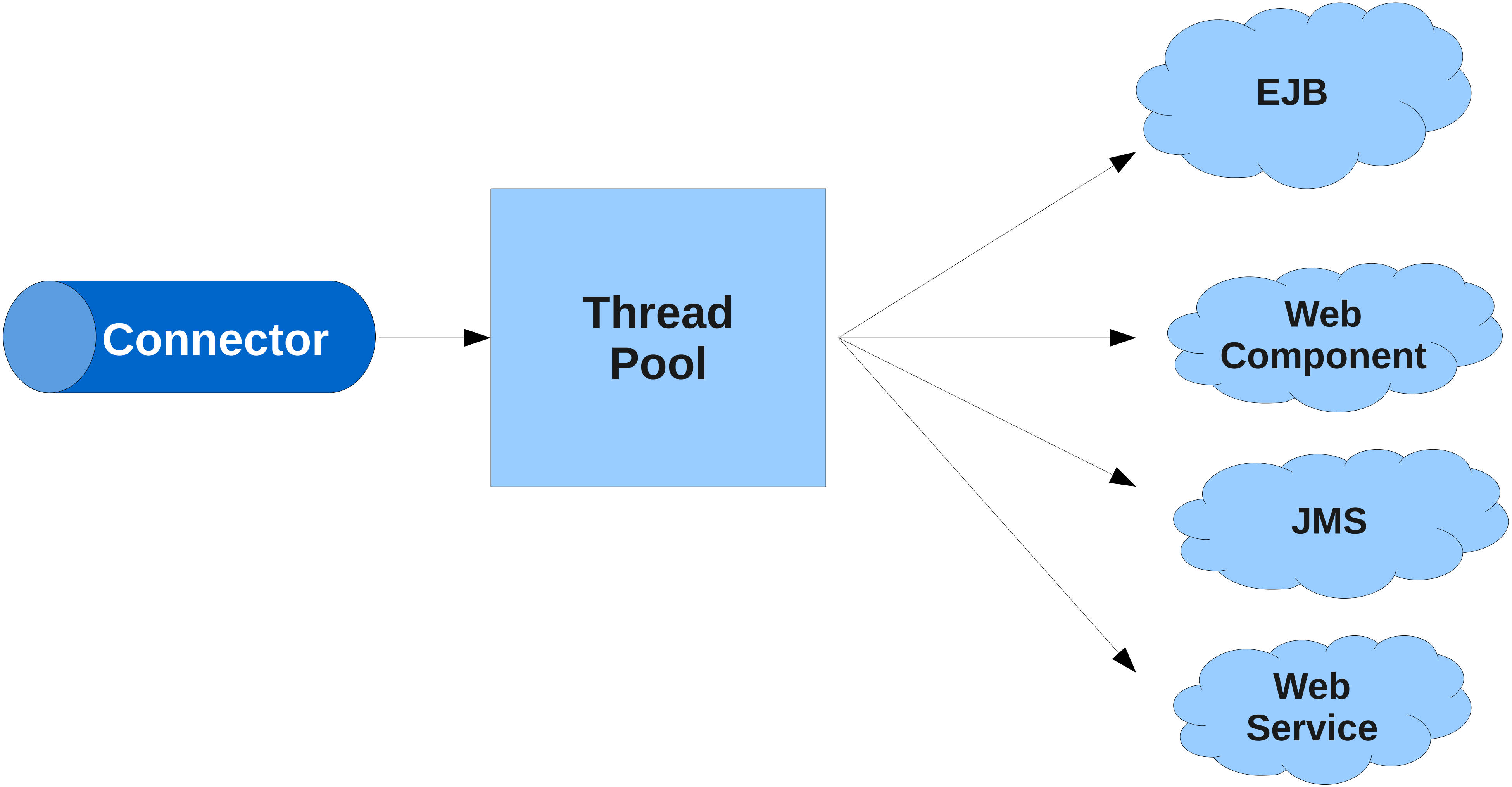
Connector

1. Request #1 opens a connection

2. Subsequent requests don't need to open a new connection

- **maxKeepAliveRequests**
  - Default is 100
  - Set to 1 to disable it (will likely lead to disaster!)
  - Set to -1 for unlimited (only limited by hardware constraints)
  - Opening and closing an HTTP connection is not a huge expense, but like any aspect of a high-volume application, every little bit helps!

# The other side of the Connector...



# maxThreads and minSpareThreads

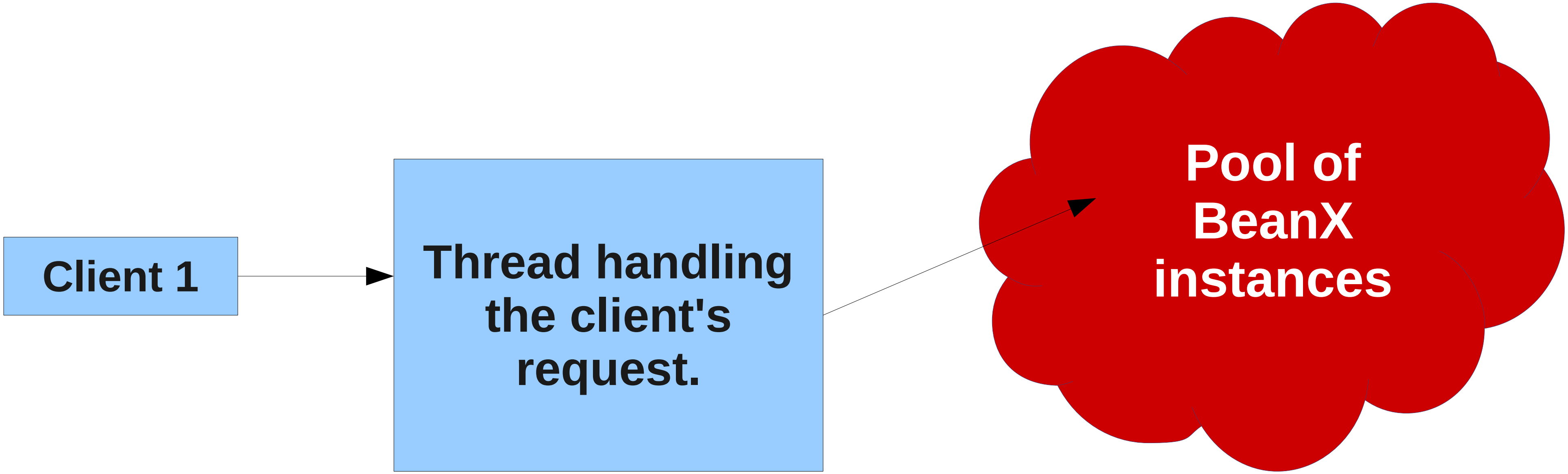
- **maxThreads**
  - The maximum number of simultaneous requests that can be handled.
  - Default is 200, which is not very high!
- **minSpareThreads**
  - The minimum number of threads always kept running.
  - Default is 10
  - Set to the # of threads needed at peak load
  - Set **maxThreads** equal to **minSpareThreads** (or a slightly larger value)

# What happens after the Connector?

- OK – so we have gotten the request through the Connector
- And it is being handled by a thread in the JVM.
- What happens next is based on the end target of the request. Some examples include:
  - An EJB method
  - A Servlet, JSP, JSF, Seam or other Web component
  - A Web Service
  - A Message Queue or Topic

# Pooling EJBs

- Whether an EJB is stateless or stateful, pooling instances greatly improves performance.



Client 1

Thread handling the client's request.

Pool of BeanX instances

# Stateless Session Bean Pools

- There are two types of pools that can be configured for use with stateless session beans:
  - 1. **ThreadlocalPool**: a pool of EJB instances local to a thread. It has no limit (so there is no need to configure it), and there is a pool for each thread.
  - 2. **StrictMaxPool**: a single pool shared by all threads that has a fixed maximum size.
- You need to monitor and test your applications using both modes to determine which is faster.

# How large is my thread pool?

- The size of the thread pool, as well as the number of threads being used, is available through JMX.
- Use any of these tools to monitor the pool size:
  - JBoss Operations Network (JON)
  - Jconsole, JVisualVM, JMX Console
  - or any other JMX-supported tool

# Configuring StrictMaxPool

- If all EJB instances in a **StrictMaxPool** are used, the waiting thread blocks and waits for one to become available.
- To configure this setting, open the file:
  - **ejb3-interceptors-aop.xml**
- in your profile's **/deploy** folder. Search for:

  @org.jboss.ejb3.annotation.Pool (**value="StrictMaxPool", maxSize=500, timeout=10000**)

# Stateful Session Beans

- Performance tuning Stateful EJBs involves configuring the cache in your cluster.
- The config file is:
  - **jboss-cache-manager-jboss-beans.xml**
- located in the following folder:
  - jboss-as/server/<profile>/deploy/cluster/jboss-cache-manager.sar/META-INF

# Stateful Session Beans

- The settings of interest are:

```
<property name="nodeLockingScheme">
    PESSIMISTIC
</property>
<property name="isolationLevel">
    REPEATABLE_READ
</property>
```

- These are typically the best values for performance and reliability.

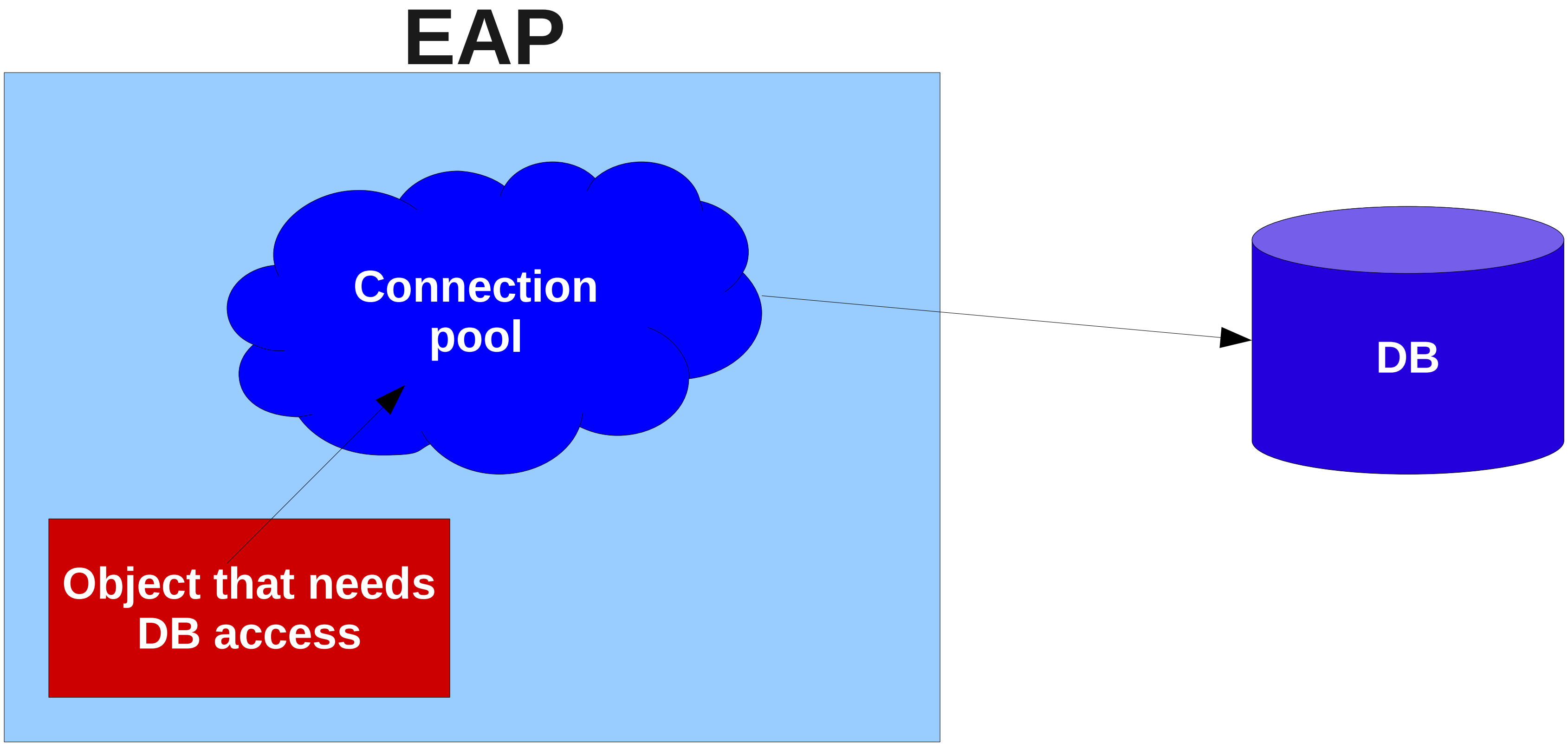
# Seam Performance Tip

- If a **Seam component** is accessed many times, you can greatly improve performance by disabling the interceptor stack.

```
@BypassInterceptors
public class MySeamComponent
{ ... }
```

- This might not be an option! But if it is, the benefit can be significant.

# Database Connection Pools



# Configuring the DB Pool Size

- Within your *-ds.xml file:

```
<min-pool-size>1000</min-pool-size>
<max-pool-size>3000</max-pool-size>
```

- The default value of **min-pool-size** is **0**, which rarely is a good option.
- Best practice: set **min-pool-size** to the minimum you need for maximum throughput.
- Set **max-pool-size** much higher than your expectations.
- Use JMX or JON to watch the pool size.

# The JCA Thread Pool

- The **JCA container** has its own thread pool (also called the **Work Manager** thread pool).
- The pool needs to be sized appropriately to handle the messaging load of your applications.
- **MessageDriven Beans** and your **JMS** code all share the same JCA Thread Pool.

# Configuring the JCA Thread Pool

- The config file for this pool is in your **/deploy** folder:
  - **jca-jboss-beans.xml**
- The bean named “**WorkManagerThreadPool**” contains the important settings:

```
<property  name="maximumQueueSize">
    1024
</property>
<property name="maximumPoolSize">
    100
</property>
```

# Monitoring the JCA Thread Pool

- Make sure **maximumPoolSize** contains enough threads to handle your peak volume.
- If all the threads are in use, the message is placed in a separate queue and waits for a thread to become available.
- Use your favorite JMX tool to monitor your JMS and make sure “**QueueSize**” is always **0**.
- If it's ever not 0, your thread pool is too small!

# A Performance Tip for JMS

- If you are using the Java Messaging Service in your EAP applications,
- then use **HornetQ** for the implementation.
- Run **./build.sh** in HornetQ config/jboss-as-5 directory
  - This creates EAP profiles for use in EAP5
- Instead of a database, HornetQ uses the Java NIO for persisting messages, using the native OS's asynchronous I/O.

# The System's Basic Thread Pool

- Configured in **jboss-service.xml** (in **/conf** folder)
- <mbean code="**org.jboss.util.threadpool.BasicThreadPool**"
- name="jboss.system:service=ThreadPool">
- <attribute name="Name">JBoss System Threads</attribute>
- <attribute name="ThreadGroupName">System Threads</attribute>
- <attribute name="KeepAliveTime">60000</attribute>
- <attribute name="**MaximumPoolSize**">10</attribute>
- <attribute name="**MaximumQueueSize**">1000</attribute>
- </mbean>

# A Peformance Tip for Logging

- Logging is configured in the file:

  **/<profile>/conf/jboss-log4j.xml**

- In production, TURN OFF **console logging**. It is the most expensive form of logging.
- Test asynchronous logging (the JMS appender), which may improve performance is you have a high frequency of logging.

# Wrap Log Statements

- If you have a lot of log statements that occur frequently, consider **wrapping** them in “**if**” statements:

```
if(debugging()) {
    log.debug(...);
}
```

- If debugging is turned off, you save the time and memory of wasted log objects that a call to “**log**” creates.

# Troubleshooting Bottlenecks

- First off, you need a good monitoring tool:
  - Java VisualVM
  - JON
  - jmx-console
- Don't forget your helpful system tools:
  - top
  - vmstat
  - iostat

# CPU Under Utilization

- Monitor trends in your CPU usage
- **Under utilization** may not be a good thing!
- Watch for high CPU idle time followed by degrading response times (once traffic picks back up)
- This may be a sign that JBoss is waiting for resources to be released by another process.
- Perform a thread dump to see which threads are waiting.

# CPU Over Utilization

- High CPU may not always be a bad thing!
- Of course, many times it can be a sign of bottlenecks.
- One useful task is to use vmstat to check the size of the run queue

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 701768  97128 771036    0    0    67    14  312  469  7  1 90  2  0
```

- If the value or **r** is bigger than the # of CPU's, than you likely have resource issues.

# High Disk I/O

- Use **iostat** to monitor possible bottlenecks caused by high usage of disk input/output.

```
iostat -xd sda
Linux 2.6.32-71.14.1.el6.x86_64 (rraposa.csb)    05/06/2011       _x86_64_  (4 CPU)
Device:        rrqm/s   wrqm/s     r/s     w/s   rsec/s   wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda              3.99     6.72    8.09    5.58   405.79    93.39    36.52     0.23   16.62   3.74   5.11
```

- If service time (svctm) and %util are high:
  - Excessive logging
  - Excessive passivating of Stateful Session Beans
  - Poorly configured database connection pool
  - Use multiple and/or faster file systems

# Thanks and Recognition to...

- The information on performance tuning can be found in the paper titled “Performance Tuning Guide for EAP” by **Andrig Miller**

# In summary...

- Thank you for attending and I hope you gained some useful information in tuning your applications.
- If you stick around, the next talk will be on understanding the various JVM memory settings.

# 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/