# Exploring Terracotta

Alex Miller

http://terracotta.org

http://tech.puredanger.com

Network Attached Memory

Open Source

Virtual Heap

Clustering

Persistent Heap

Grid

Scalability

Java Memory Model

# Agenda

- What is it?
- How do I use it?
- What's it good for?
- Q&A

# Clustered Heap

- Virtual
- Shared
- Persistent
- Large (1 TB)

# Clustered Synchronization

- synchronized
- wait/notify
- java.util.concurrent
- Java Memory Model

# Transparency

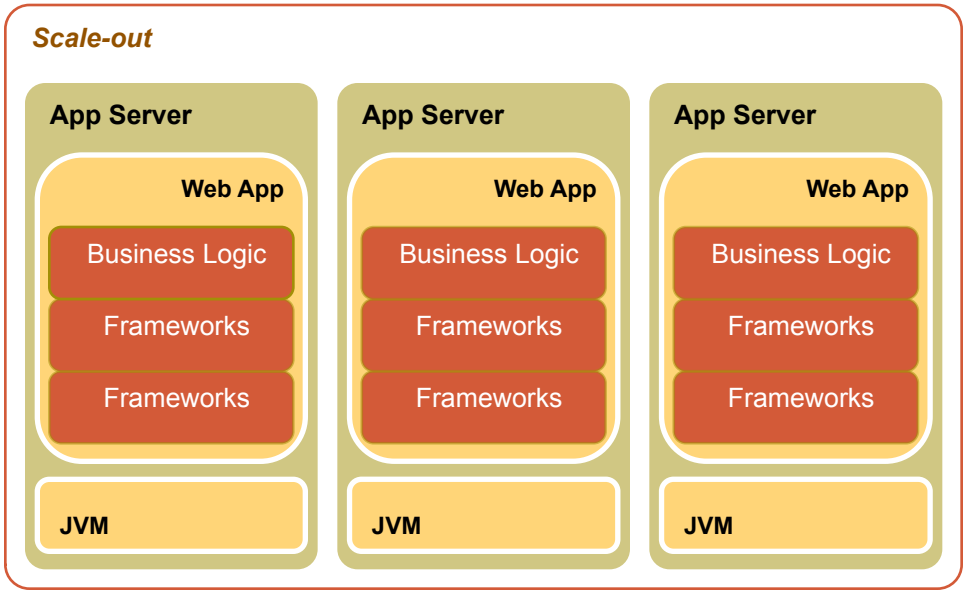
- No API, no JAR, just Java
- No serialization
- External declarative configuration

# Enterprise-ready

- Scalability
- High-availability
- Monitoring
- Management

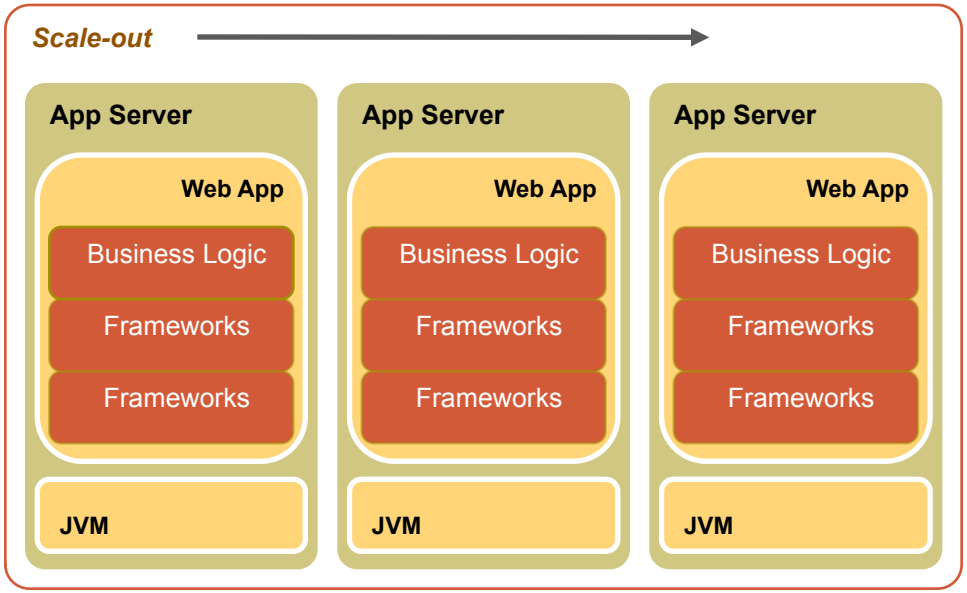
# Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code



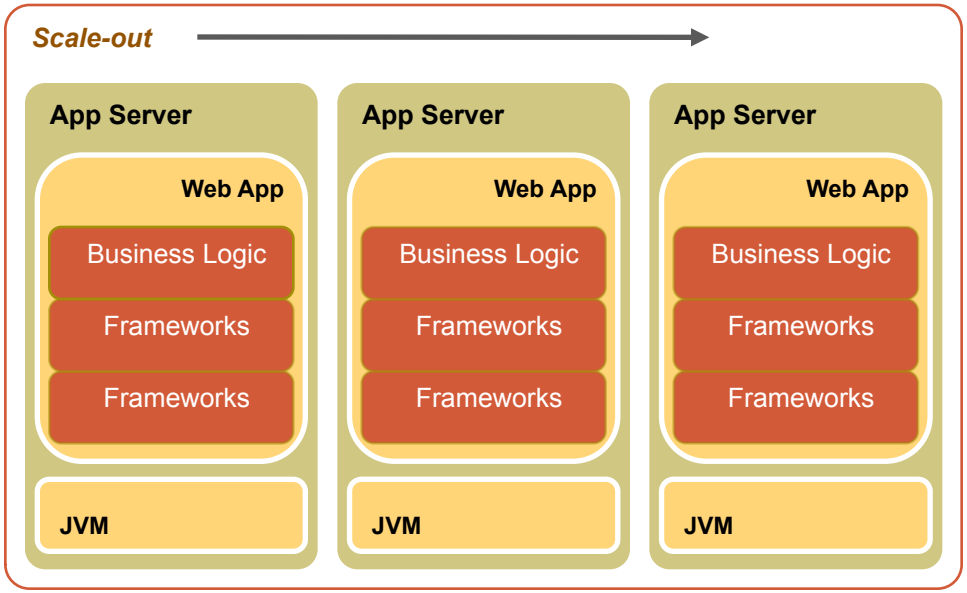
# Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code



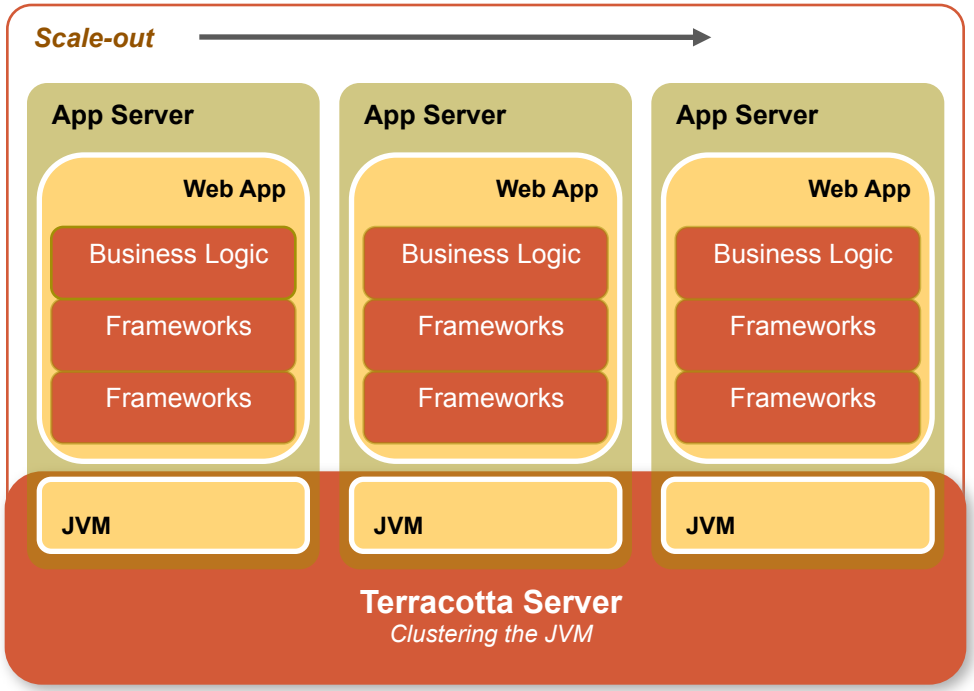
# Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code



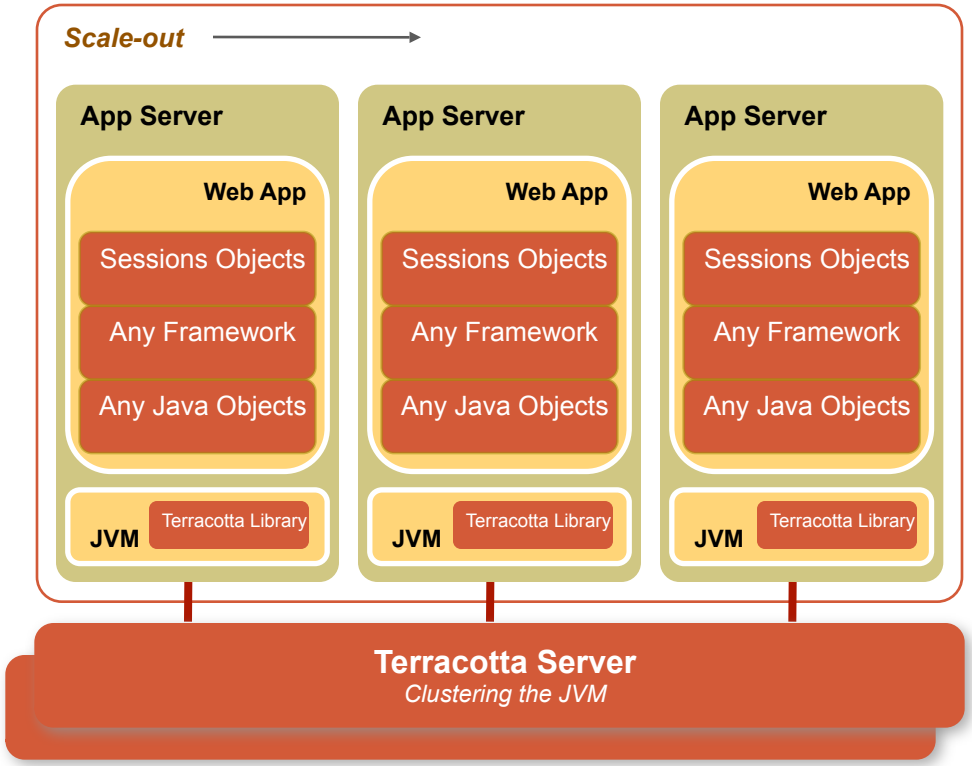
# Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code
- Our different approach
  - Cluster the JVM
  - Eliminate need for custom



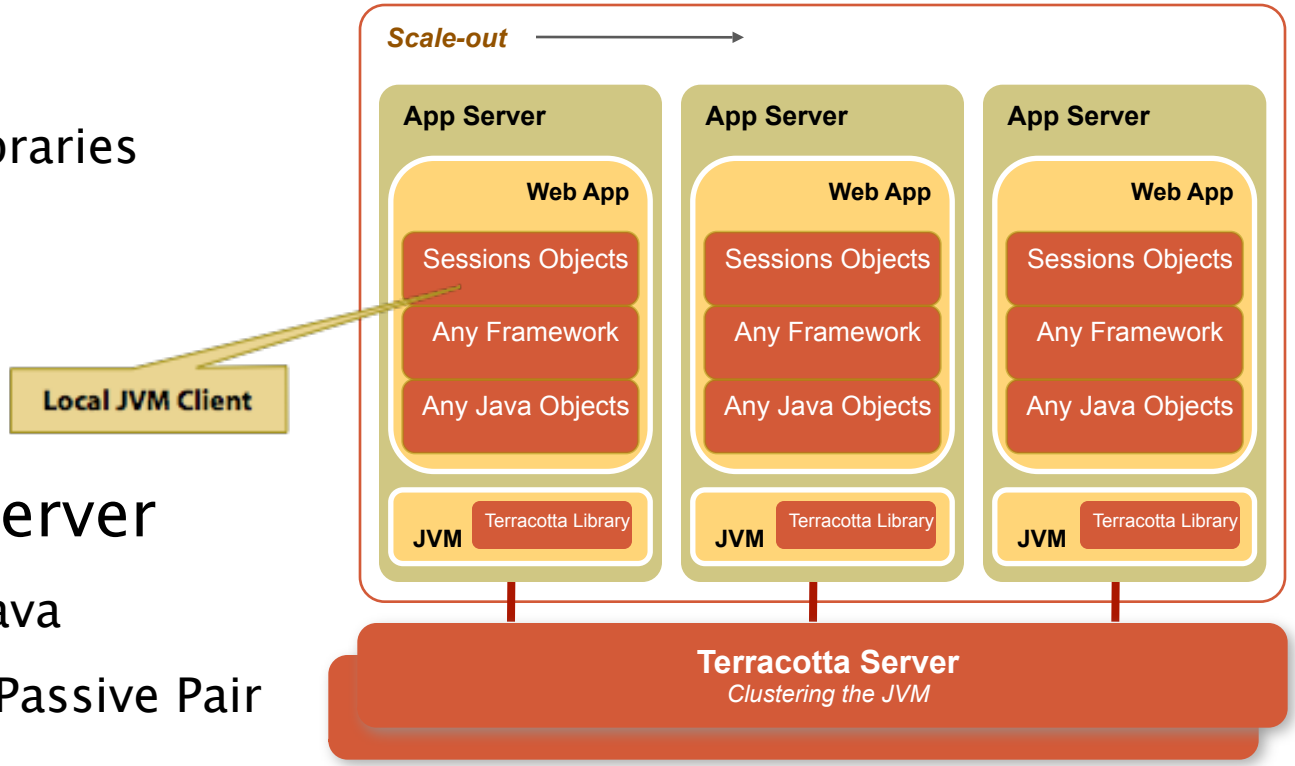
# The architecture of Terracotta

- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair



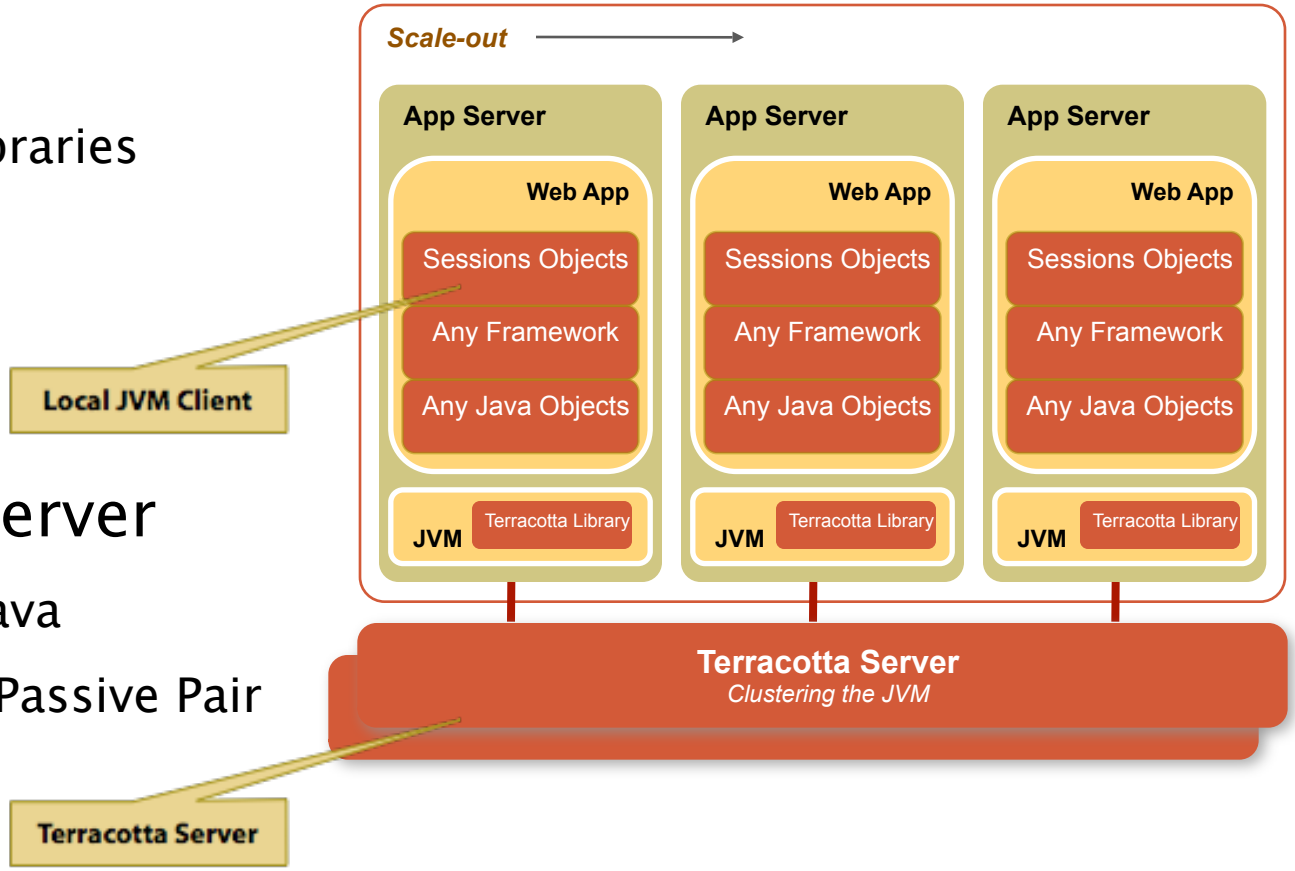
# The architecture of Terracotta

- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair



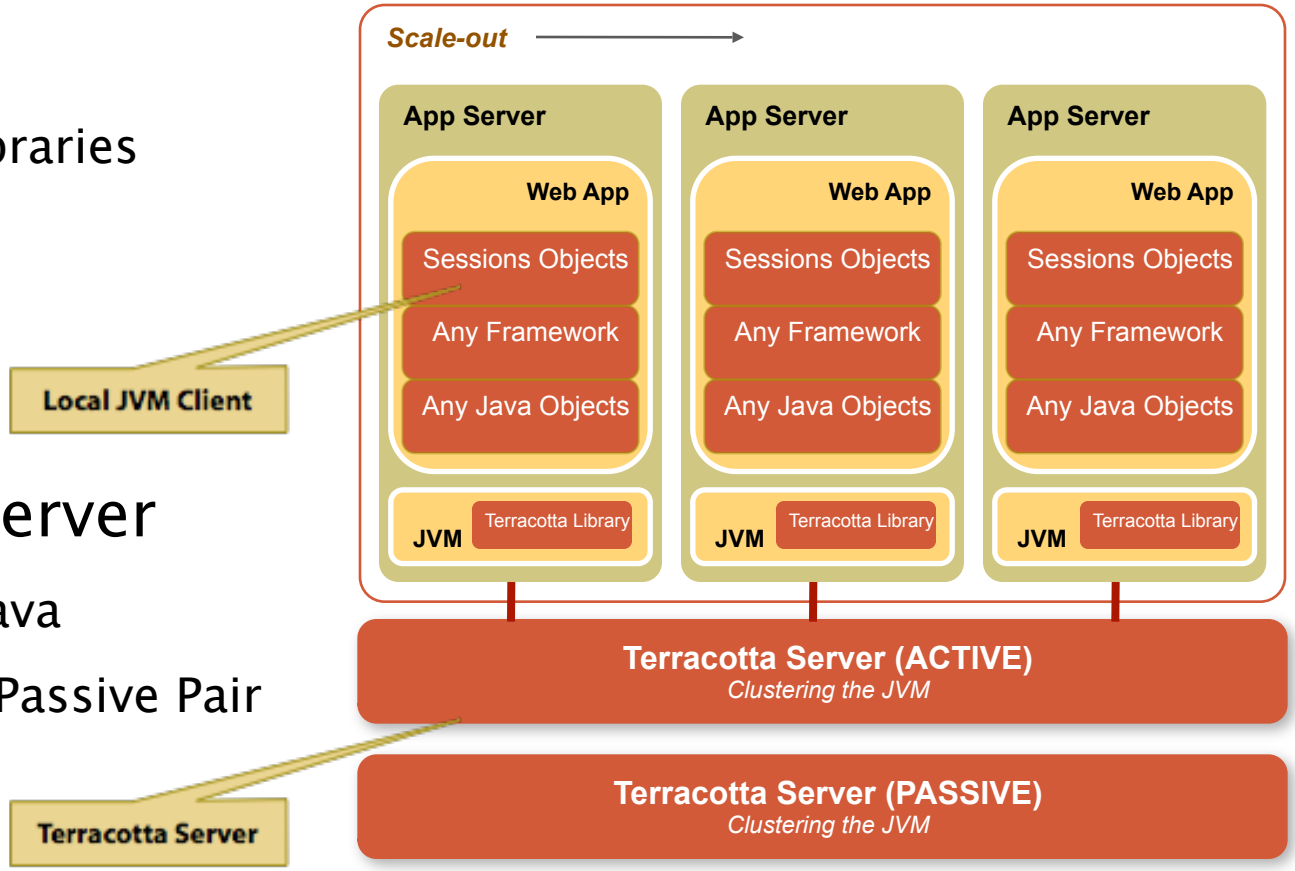
# The architecture of Terracotta

- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair



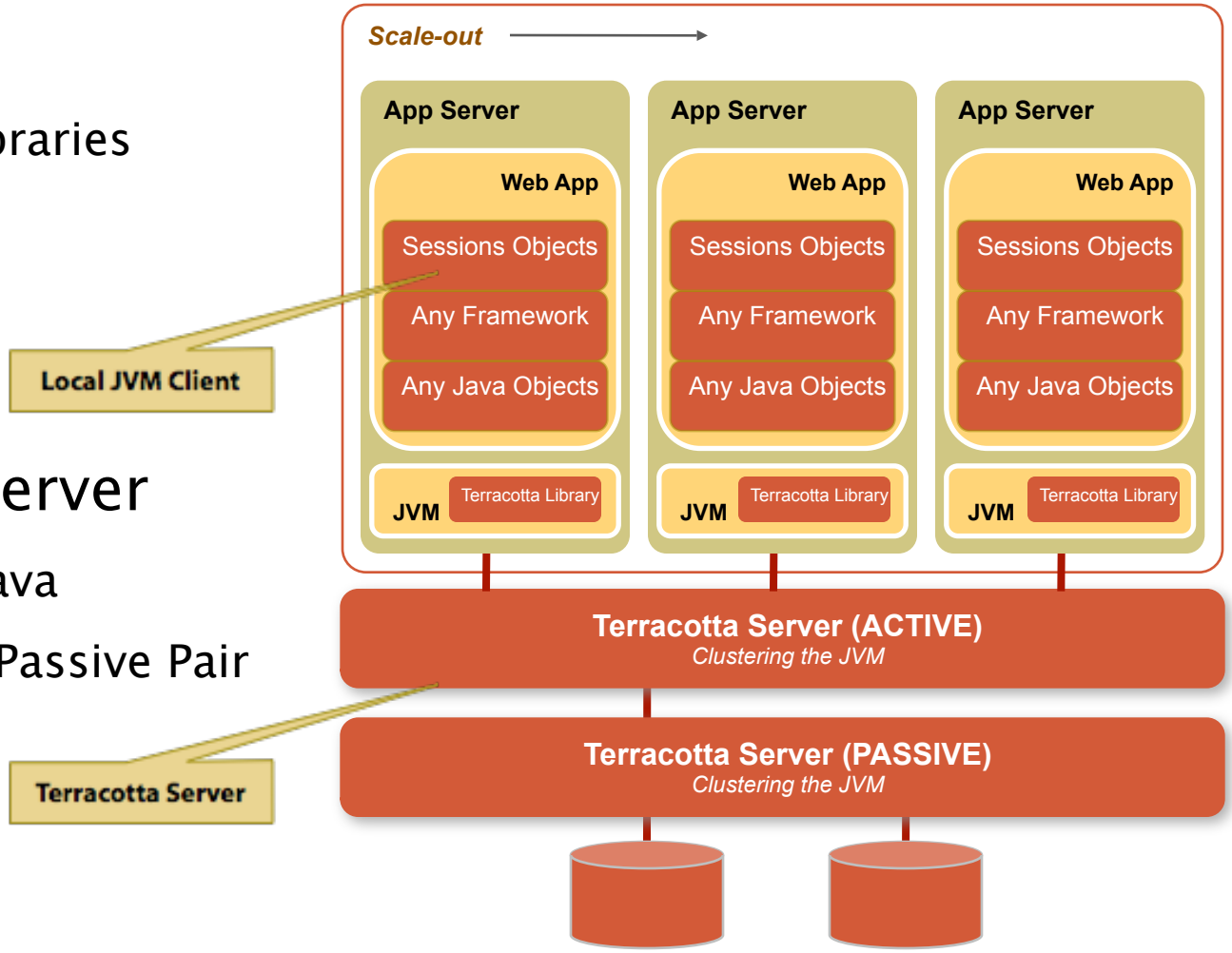
# The architecture of Terracotta

- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair

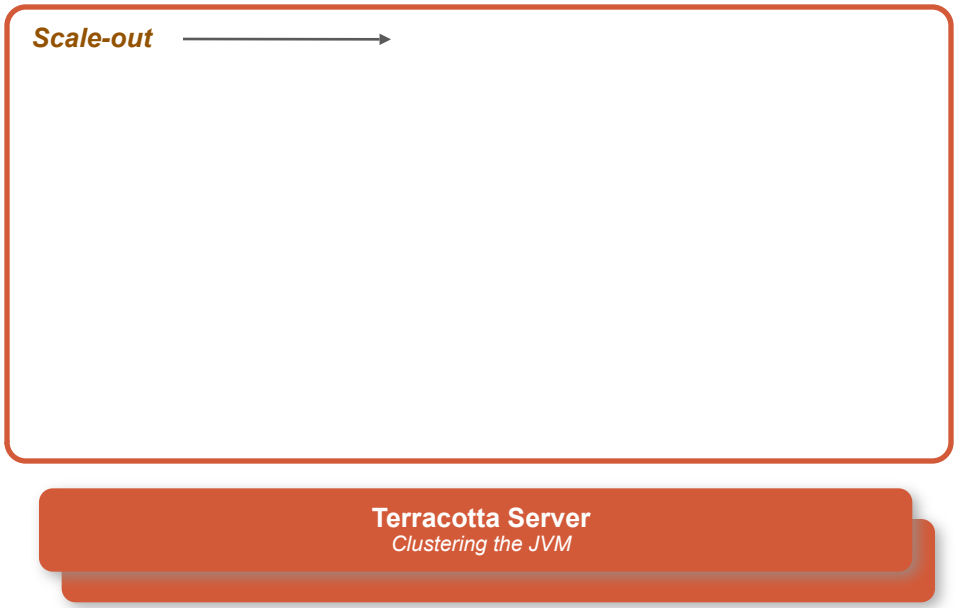


# The architecture of Terracotta

- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair

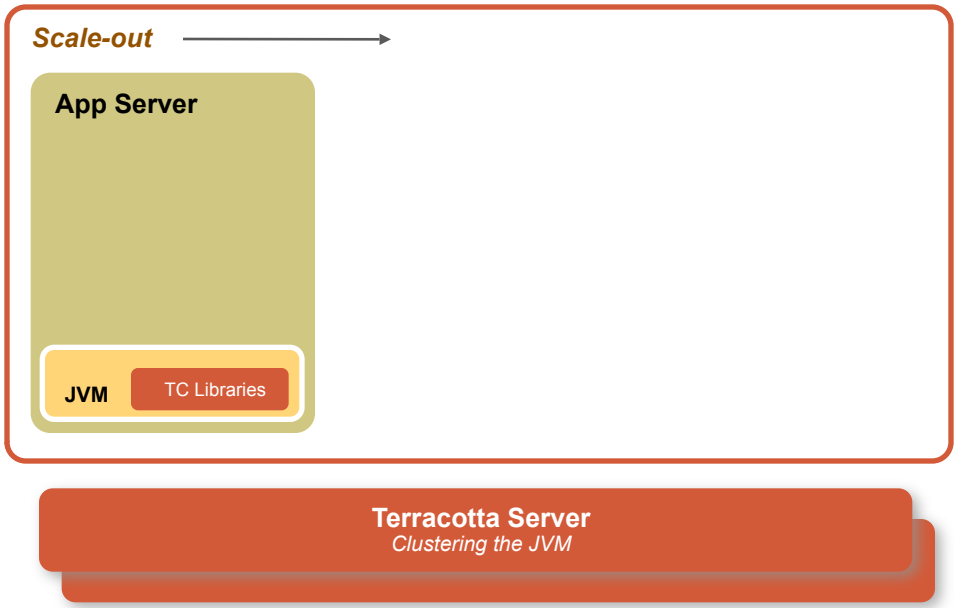


# Network-Attached Memory



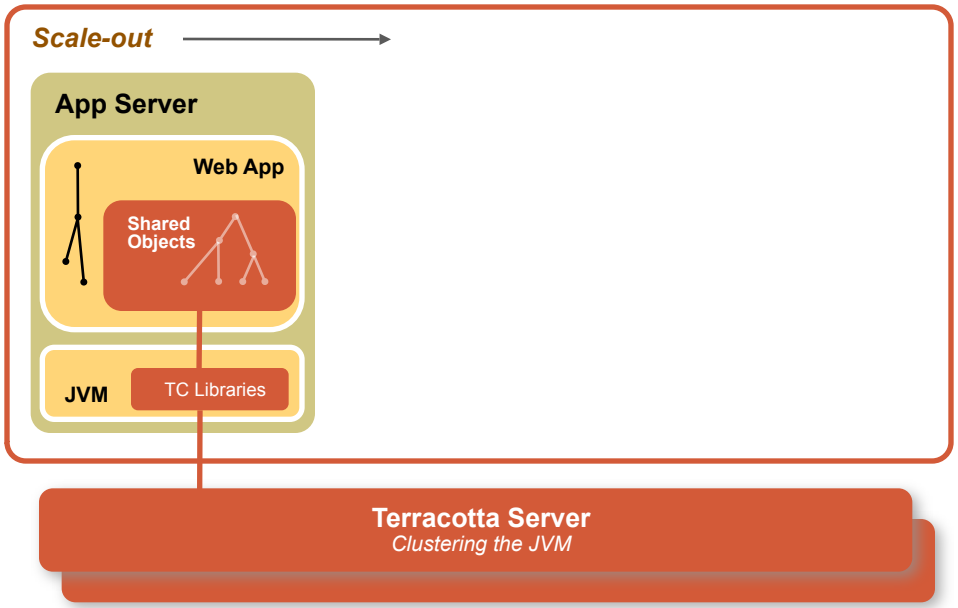

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



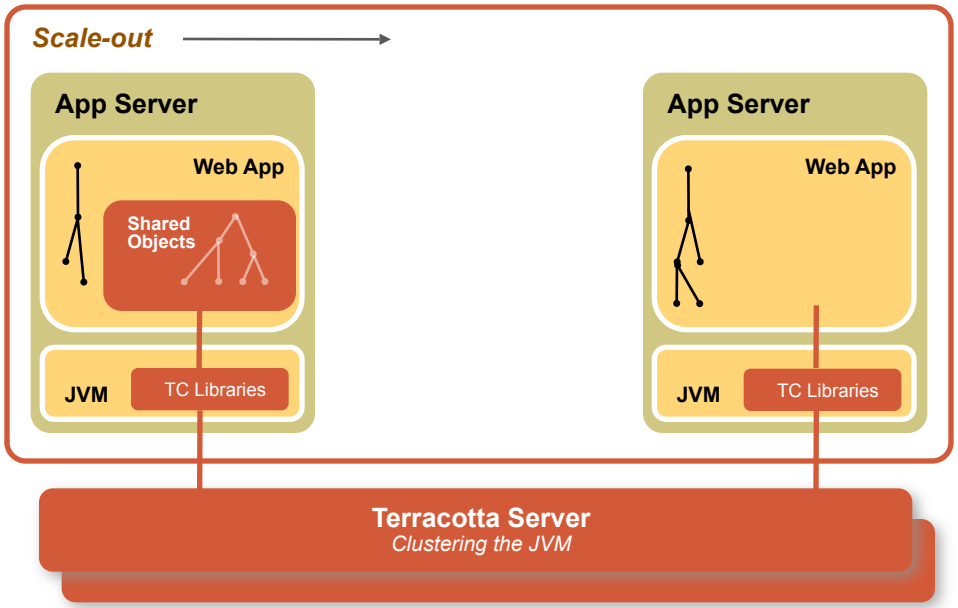

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD – PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY – MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



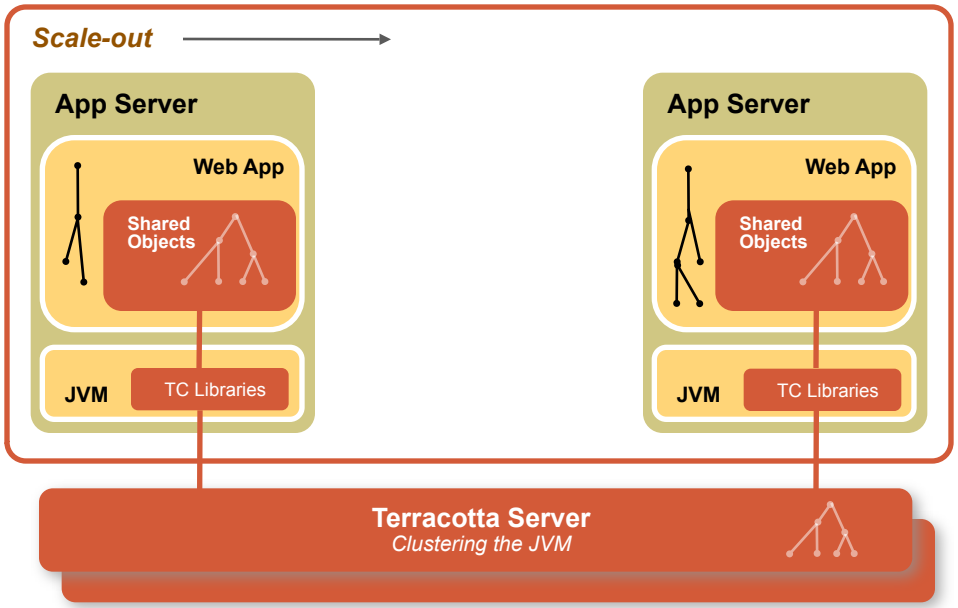

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



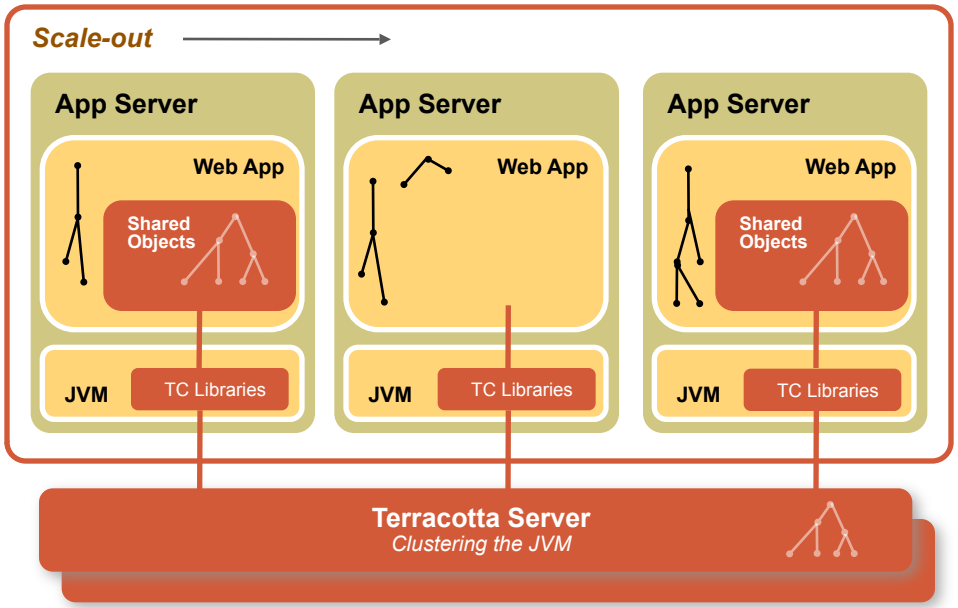

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



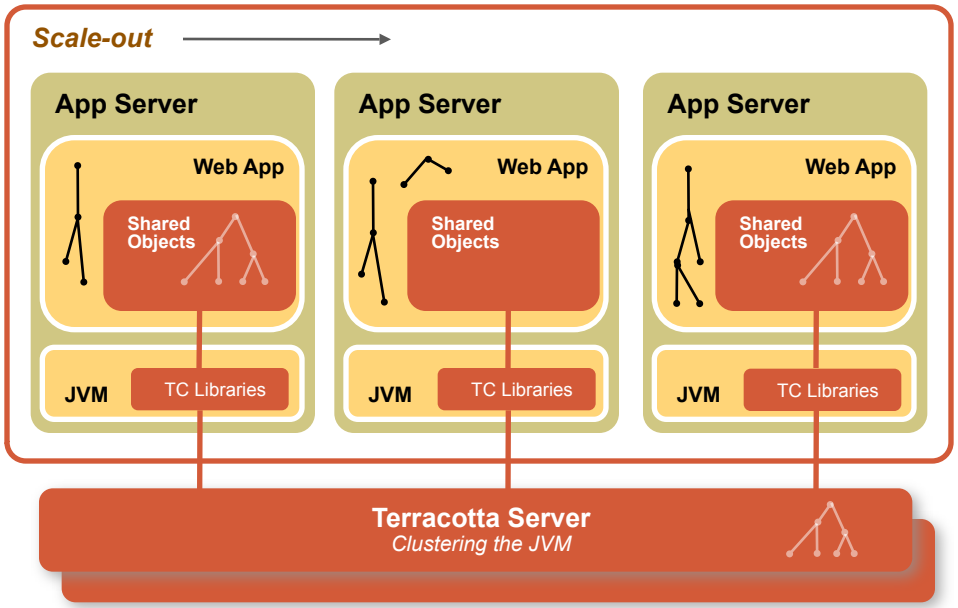

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



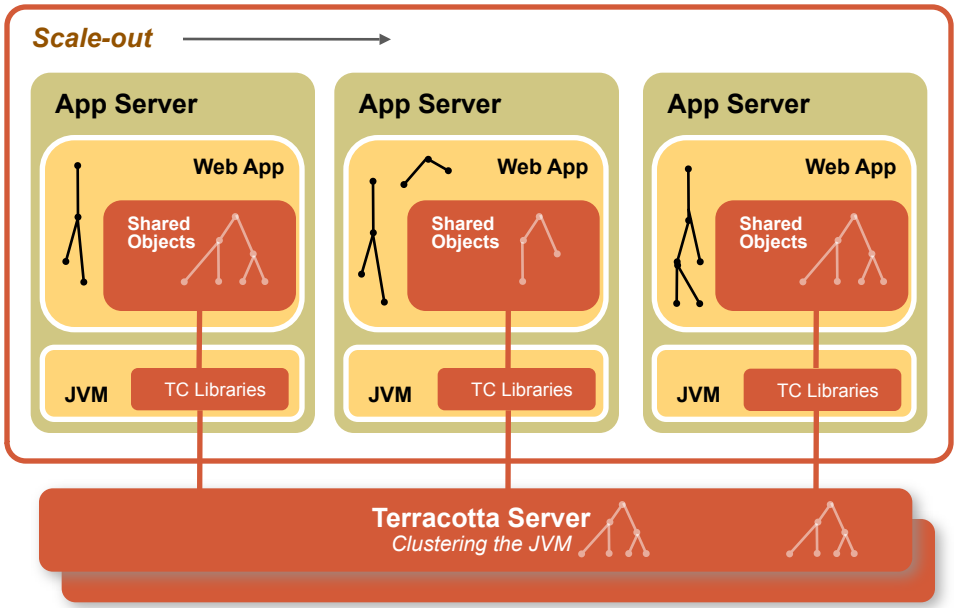

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



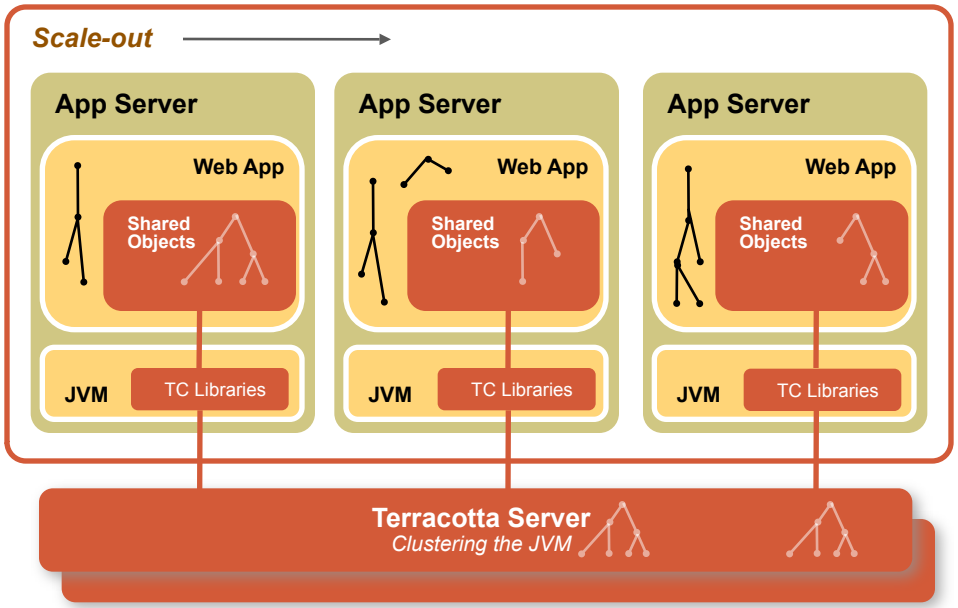

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory



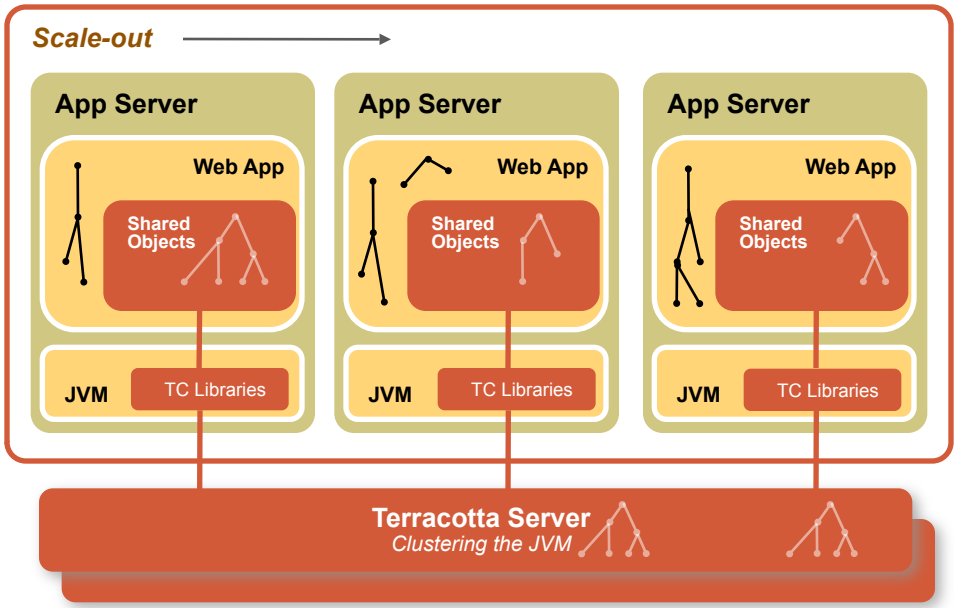

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory




- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD - PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY - MONITOR_EXIT
- Large Virtual Heaps

# Network-Attached Memory




- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - GET_FIELD – PUT_FIELD
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed wait()/notify()
  - Fine Grained Locking MONITOR_ENTRY – MONITOR_EXIT
- Large Virtual Heaps

TC Management Console

- Management Console
  - Runtime visibility
  - Data introspection
  - Cluster monitoring

# Agenda

- What is it?
- How do I use it?
- What's it good for?
- Q&A

# A simple counter...

```java
public class Main {
  // Create single static Main instance
  public static Main instance = new Main();

  // Main - just count the instance twice
  public static void main(String[] args) throws InterruptedException {
    for (int i = 0; i < 100; i++) {
      instance.count();
      Thread.sleep(1000);
    }
  }

  // Main state - a simple counter
  private int counter = 0;

  // Synchronize access to make Main thread-safe
  public synchronized void count() {
    counter++;
    System.out.println("Counter is: " + counter);
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<con:tc-config xmlns:con="http://www.terracotta.org/config">
  <servers>
    <server host="%i" name="localhost">
      <dso-port>9510</dso-port>
      <jmx-port>9520</jmx-port>
      <data>terracotta/server-data</data>
      <logs>terracotta/server-logs</logs>
    </server>
  </servers>
  <clients>
    <logs>terracotta/client-logs</logs>
  </clients>
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>Main</class-expression>
        </include>
      </instrumented-classes>
      <roots>
        <root>
          <field-name>Main.instance</field-name>
        </root>
      </roots>
      <locks>
        <autolock>
          <method-expression>void Main.count()</method-expression>
          <lock-level>write</lock-level>
        </autolock>
      </locks>
    </dso>
  </application>
</con:tc-config>
```

# Coordination

```
public class Coordination {
  public static final Coordination instance = new Coordination();

  public static void main(String[] args) throws Exception {
    instance.run();
  }

  public AtomicInteger     counter  = new AtomicInteger(0);
  public String            msg;

  public void run() throws Exception {
    // ...choose a role and call a method below forever
  }

  private synchronized void getInput() throws IOException {
    System.out.print("Enter a message> ");
    System.out.flush();
    msg = new BufferedReader(new InputStreamReader(System.in)).readLine();
    notify();
  }

  private synchronized void printInput() throws InterruptedException {
    while (msg == null) {
      wait();
    }
    System.out.println(msg);
    msg = null;
  }
}
```

# Shared Map

```
public class SharedData {

  private final Map<String, Date> data = new HashMap<String, Date>();

  public synchronized void addName(String name) {
    data.put(name, new Date());
  }

  public synchronized void removeName(String name) {
    data.remove(name);
  }

  public synchronized Date getJoined(String name) {
    return data.get(name);
  }

  public synchronized Set<String> getNames() {
    return data.keySet();
  }
}
```

```
public class Main {
  public static void main(String arg[]) throws IOException {
    new Main().run();
  }

  private final SharedData data = new SharedData();

  public void run() throws IOException {
    while(true) {
      String[] command = readCommand();
      if(command[0].equals("add")) {
        data.addName(command[1]);
      } else if(command[0].equals("remove")) {
        data.removeName(command[1]);
      } else if(command[0].equals("list")) {
        for(String name : data.getNames()) {
          System.out.println("  " + name + " : " + data.getJoined(name));
        }
      } else if(command[0].equals("quit")) {
        return;
      }
    }
  }

  private String[] readCommand() throws IOException { ... }
}
```

# CyclicBarrier

```
public class Node {
  public static void main(String[] args) throws Exception {
    int parties = Integer.parseInt(args[0]);
    Node node = new Node(parties);
    node.run();
  }

  private final CyclicBarrier barrier;

  public Node(int parties) {
    barrier = new CyclicBarrier(parties);
  }

  public void run() throws Exception {
    System.out.println("Waiting for other node to join...");
    int joined = barrier.await();
    System.out.println("Started: " + joined);

    barrier.await();
    System.out.println("Ended: " + joined);
  }
}
```

# Agenda

- What is it?
- How do I use it?
- What's it good for?
- Q&A

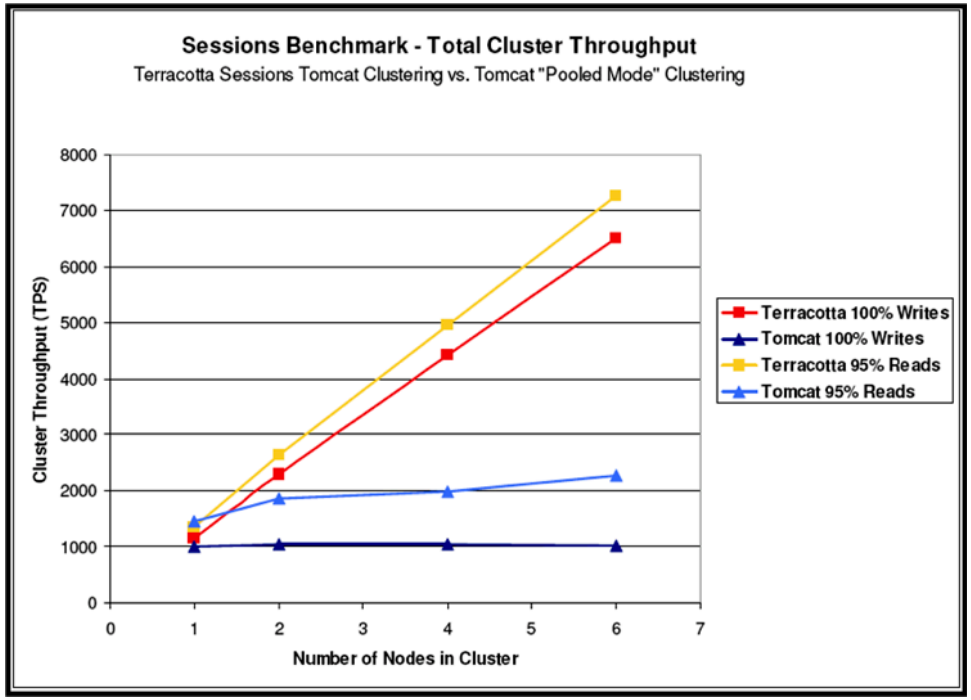
# Use-cases

- **Relieving Database Overload**
  - Distributed Caching
  - Hibernate Clustering
  - HTTP Session Clustering
- **Simplifying Application Architecture and Development**
  - Virtual Heap for Large Datasets
  - Clustering OSS Frameworks (Spring, Struts, Lucene, Wicket, EHCache etc.)
  - Master/Worker – Managing Large Workloads
  - POJO Clustering
  - Messaging, Event-based Systems and Coordination-related Tasks

# HTTP Session Clustering without Java Serialization

# HTTP Session Clustering – Benefits

- Terracotta Sessions gives you:
  - Near-Linear Scale
  - No java.io.Serializable
  - Large Sessions – MBs
  - Higher Throughput
- Supported Platforms:
  - Jetty,
  - JBoss 4.x,
  - Tomcat 5.0 & 5.5,
  - WebLogic 8.1, WebLogic 9.2,
  - WebSphere CE, Geronimo Alpha, WebSphere 6.1



# HTTP Session Clustering - DummyCart.java

```
package demo.cart;

import java.util.*;

public class DummyCart {
    private List items = new ArrayList();

    public List getItems() {
        return Collections.unmodifiableList(items);
    }

    public void addItem(String name) {
        items.add(name);
    }

    public void removeItem(String name) {
        items.remove(name);
    }
}
```

# HTTP Session Clustering – carts.jsp

```
<%@ page import="java.util.Iterator" %>
<html>
  <jsp:useBean id="cart" scope="session" class="demo.cart.DummyCart" />
  <%  String submit = request.getParameter("submit");
      String item = request.getParameter("item");
      if (submit != null && item != null) {
        if (submit.equals("add")) {
          cart.addItem(item);
        } else if (submit.equals("remove")) {
          cart.removeItem(item);
        }
      }%>
  <body>
    <p>You have the following items in your cart:</p>
    <ol><%
      Iterator it = cart.getItems().iterator();
      while (it.hasNext()) {
        %><li><% out.print(it.next()); %></li> <%
      } %>
    </ol>

    <form method="get" action="<%=request.getContextPath()%>/cart.jsp">
      <p>Item to add or remove: <input type="text" name="item" /></p>
      <input type="submit" name="submit" value="add" />
      <input type="submit" name="submit" value="remove" />
    </form>
  </body>
</html>
```

# HTTP Session Clustering - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <web-applications>
        <web-application>cart</web-application>
      </web-applications>
      <instrumented-classes>
        <include>
          <class-expression>demo.*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

# Demo

# Cluster Spring and other OSS frameworks

# Glimpse of supported integrations

- 'Supported' by Terracotta means:
  - Just include a ‘Configuration Module’ (OSGi bundle) in your config to cluster your application (a one liner)
  - Plugs in underneath without any setup
  - Technically, Terracotta supports all integrations as long as it runs



# Example Configuration Module - EHCache clustering

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <clients>
    <modules>
      <module name="clustered-ehcache-1.3" version="1.0.0"/>
    </modules>
  </clients>
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>tutorial.*..*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

# Example – Spring clustering

## Terracotta config

```
<spring>
 <application name="tc-jmx">
  <application-contexts>
   <application-context>
    <paths>
     <path>*/applicationContext.xml</path>
    </paths>
    <beans>
     <bean name="clusteredCounter"/>
     <bean name="clusteredHistory"/>
    </beans>
   </application-context>
  </application-contexts>
 </application>
</spring>
```

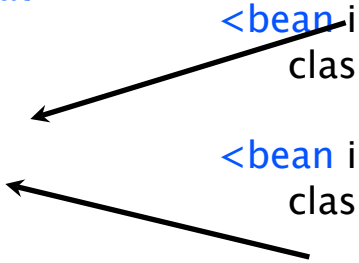
## Spring config

```
<bean id="localCounter"
    class="demo.jmx.Counter"/>

<bean id="clusteredCounter"
    class="demo.jmx.Counter"/>

<bean id="localHistory"
    class="demo.jmx.HistoryQueue"/>

<bean id="clusteredHistory"
    class="demo.jmx.HistoryQueue"/>
```



- Terracotta can declaratively cluster Spring beans (Singleton + Session and Custom scoped) with zero code changes
- Can also cluster Spring ApplicationContext events, JMX State and Spring Web Flow

# Distributed client-side events and data

# Distributed client-side - overview

- Terracotta works for all Java apps, not just server-side
- Method invocations can be distributed : DMI

```
<distributed-methods>
  <method-expression>
    void com.MyClass.somethingHappened(String, int)
  </method-expression>
</distributed-methods>
```

- Works out-of-the-box for Swing events and listeners

# Distributed client-side – TableDemo.java

```
class TableDemo extends JFrame {
  private DefaultTableModel model;

  private static Object[][] tableData = {
    { " 9:00", "", "", ""}, { "10:00", "", "", ""}, { "11:00", "", "", ""},
    { "12:00", "", "", ""}, { " 1:00", "", "", ""}, { " 2:00", "", "", ""},
    { " 3:00", "", "", ""}, { " 4:00", "", "", ""}, { " 5:00", "", "", ""}
  };

  TableDemo()  {
    super("Table Demo");
    setSize(350, 220);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Object[] header = {"Time", "Room A", "Room B", "Room C"};
    model = new DefaultTableModel(tableData, header);
    JTable schedule = new JTable(model);
    getContentPane().add(new JScrollPane(schedule), java.awt.BorderLayout.CENTER);
  }

  public static void main(String[] args) {
    new TableDemo().setVisible(true);
  }
}
```

# Distributed client-side - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>demo.*..*</class-expression>
        </include>
      </instrumented-classes>
      <roots>
        <root>
          <field-name>demo.TableDemo.model</field-name>
        </root>
      </roots>
    </dso>
  </application>
</tc:tc-config>
```

# Demo

# Fine-grained Distributed Caches

# Distributed Caches - Benefits

- Simple : pick the cache and just declare the module
  - EHCache, JBoss TreeCache,  OSCache
  - Java Collection (e.g. java.util.Hashmap, java.util.HashSet, java.util.TreeMap, ...)
- Fast:
  - Field Level Delta changes
  - Replicate only where needed
  - Appliance-like design that can optimize itself
- Big:
  - Virtual heap that exceeds limitations of single JVM
  - Lazy-loading with on-demand faulting and flushing of data
  - Coherent

# Distributed EHCache - EHCache.java

```
package tutorial;
import net.sf.ehcache.*;

public class EHCache {
  private CacheManager cacheManager = CacheManager.create("ehcache.xml");
  private Cache cache;

  public EHCache() {
    this.cache = cacheManager.getCache("TestCache");
    if (null == this.cache) {
      Cache cache = new Cache("TestCache", 1000, false, false, 120, 120);
      cacheManager.addCache(cache);
      this.cache = cache;
    }
  }

  public void doCache() {
    cache.put(new Element(new java.util.Date(), System.currentTimeMillis()));
    for (Object key : cache.getKeys()) {
      System.out.println(cache.get(key));
    }
  }

  public static void main(String[] args) {
    new EHCache().doCache();
  }
}
```

# Distributed Ehcache - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <clients>
    <modules>
      <module name="clustered-ehcache-1.3" version="1.0.0"/>
    </modules>
  </clients>
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>tutorial.*..*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

# Demo

# Agenda

- What is it?
- How do I use it?
- What's it good for?
- Q&A

# Wrapping up

# Wrapping up

- Terracotta is Network-Attached Memory for the JVM
- Turns Scalability and High-Availability into a **deployment artifact**
- Keep the simplicity of POJO-based development – get **Scale-Out with Simplicity**
- Makes mission-critical applications **simpler to**:
  - **Write**
  - **Understand**
  - **Test**
  - **Maintain**
- Endless possibilities for clustering and distributed programming – these were just a few
- Be creative, use your imagination and have fun…

# Questions?


http://terracotta.org
http://tech.puredanger.com