Terracotta

Simpler way to availability, scalability and performance

Pär Wenåker
par.wenaker@callistaenterprise.se
www.callistaenterprise.se

CALLISTA

Agenda

- Background to availability, scalability and performance
- What is Terracotta?
- How does it work?
- Use-cases for Terracotta

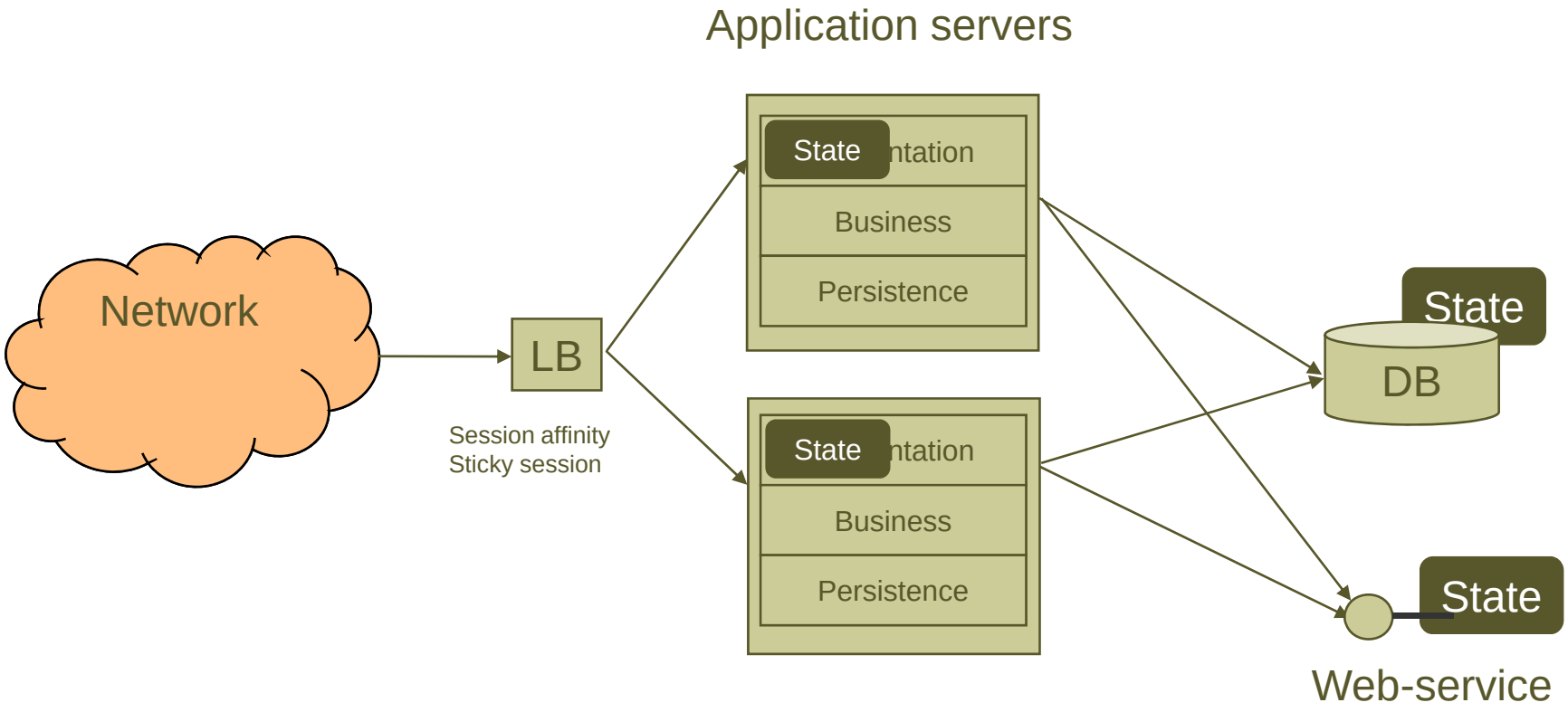
CALLISTA

Terracotta

- www.terracotta.org
- Open Source since December 2006.
- Distributed under the Terracotta Public License that is based on the Mozilla Public License 1.1
- Sponsored by Terracotta Inc.

CALLISTA

Java Enterprise System Setup Example



CALLISTA

How do we secure these properties?

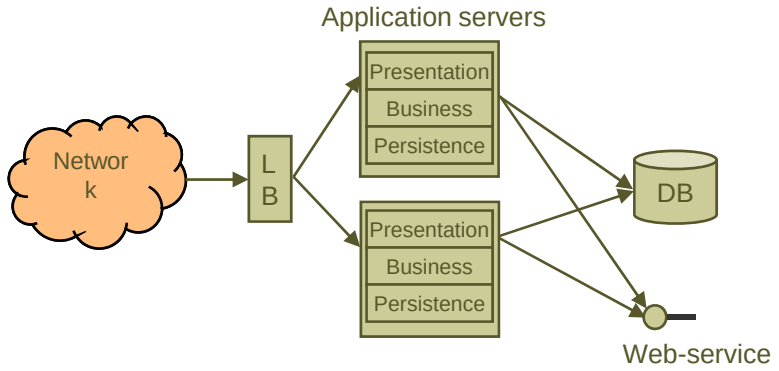
- **Availability**
  - The proportion of time that the system is in a functionall condition.
- **Scalability**
  - The ability of a system to handle a bigger workload when more resources are made available to the system
    - Horizontal vs. Vertical scalability
- **Performance**
  - How fast a system can execute a specific task it is given.

CALLISTA

Availability

We have two servers to get good availability but…

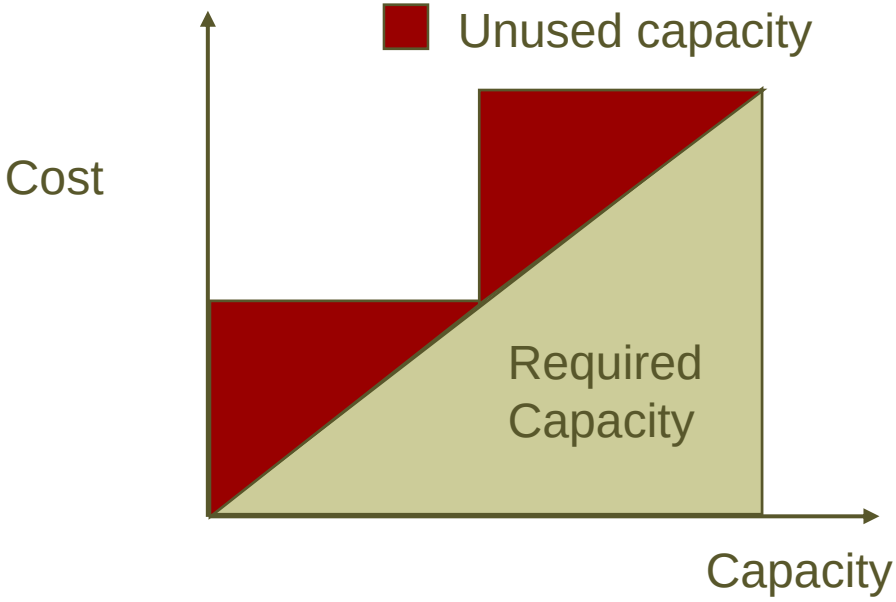
- If one of the servers goes down we will loose the HTTP-sessions in that server.
- One HTTP-session = One shopping basket = One Order = Money!
- The web service is not always available.
- The database is not always available.

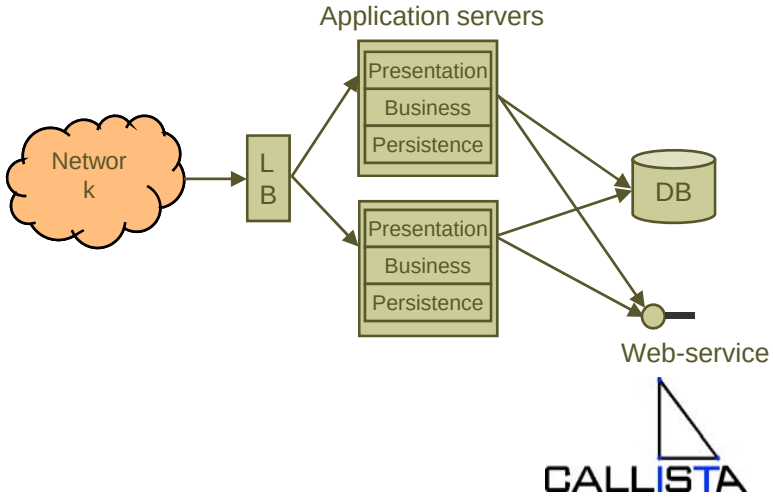


CALLISTA

Scalability

- With increased load on the application servers the load on the database and web-service will increase.
- The database and web-service will not be able to handle the increased load.
- Scaling up the database can be expensive.



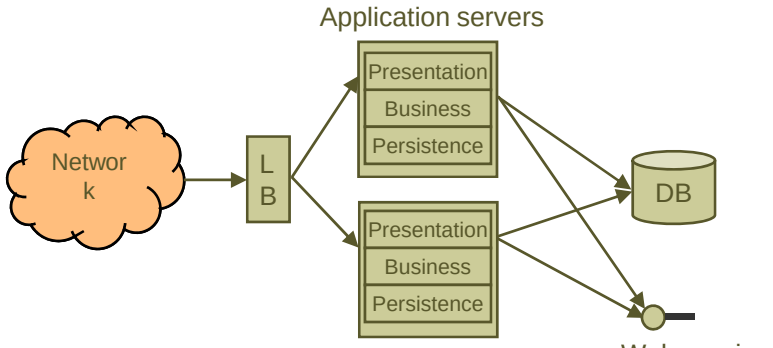


CALLISTA

Scalability (cont.)

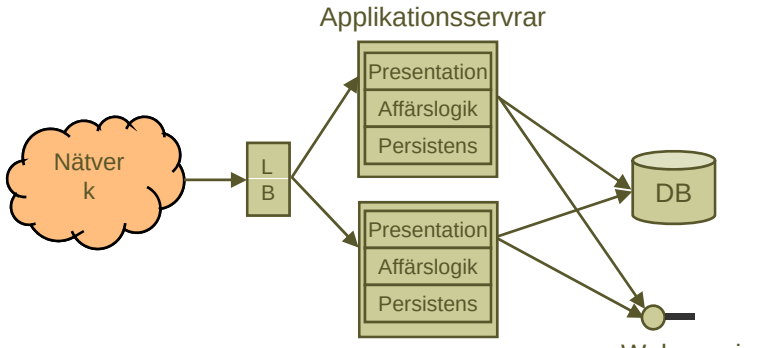
To abstract the database away using an ORM is a beautiful thought, but…

- The ORM invites to save all kind of state in the database.
  - Conversational
  - Data that is built up in pieces over time.
  - Throw-away-data
- We are tormenting our database…



Performance

- Some database interaction will take long time and the system will be perceived as slow.
- The OR-mapping might generate the 1+n roundtrip problem.
- The external web-service might slow down under heavy load.



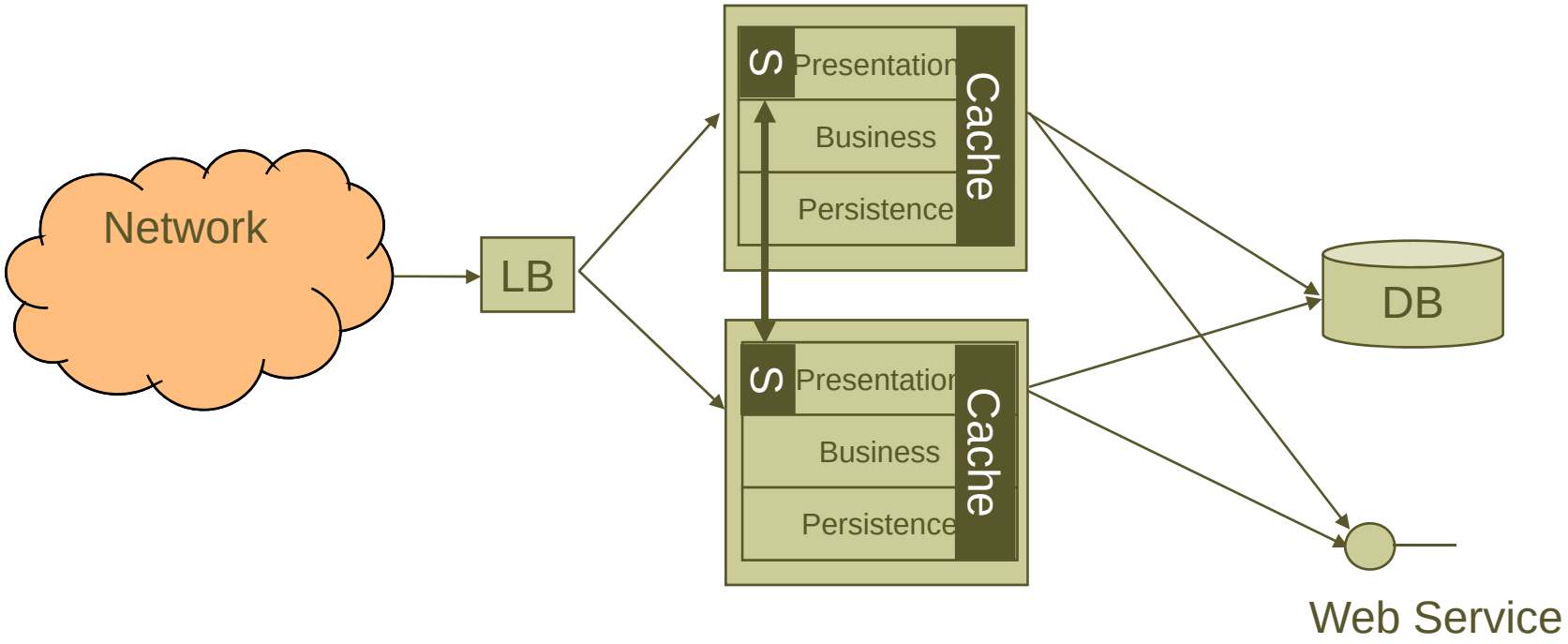
State

- All these properties relate to how we handle state!

CALLISTA

Traditional solution

- Availability -> Session replication
- Performance/Scalability -> Caching



CALLISTA

Challenges with session replication

- How do we do the replication? There is no standard way.
  - We could serialize the session to the database at each HTTP-request.
  - We could replicate over the network.
- Memory demands on the servers will increase in order to hold all the sessions.
- The web application has to be written for distribution.
  - Minimal session/setAttribute/invalidate etc.

CALLISTA

Challenges with caching

- The cache should be up-to-date.
  - Meaning is often depending on the application.
- Do we require coherence between application servers?
- If the cache is non-persistent it can only hold “mirrored” data.
- Consumes memory.
- Complicated to implement yourself.
- Might be complicated to configure and tune.

CALLISTA

What is Terracotta?

Terracotta offers:

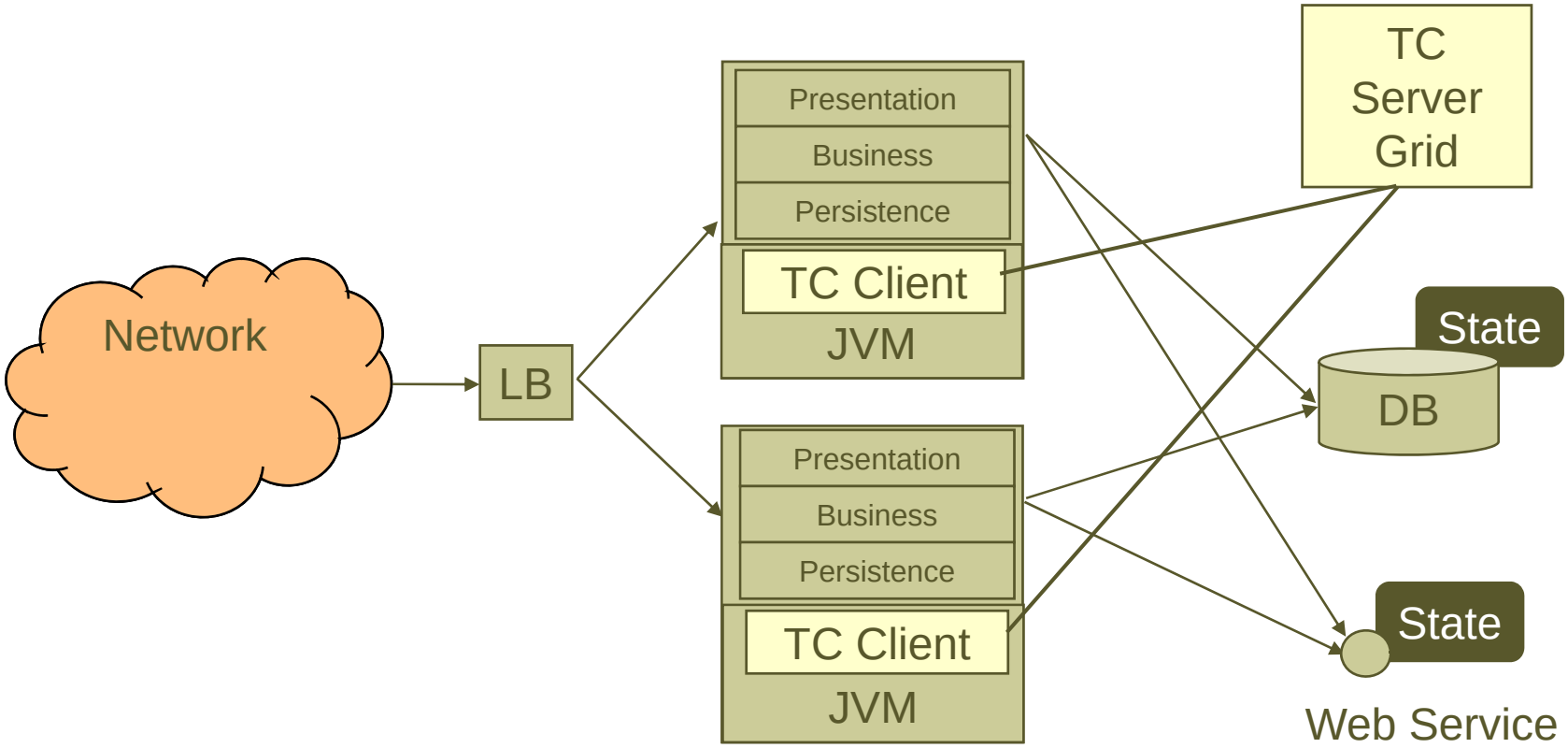
- A ***coherent***, ***distributed*** and ***persistent*** JVM heap.
  - Java objects created on the heap are available in all JVMs.
  - Objects survive a JVM restart.
  - Object identity is preserved between JVMs (no copies!)
  - The heap can spill and does not have to stay in memory, to help avoid OOME.
- The heap follows the memory and thread model of Java.
  - Java objects have coherent state between JVMs.
  - Threads in different JVMs interact just like threads in the same JVM.
- Requires no specific Java APIs.
- Integrates with other Java frameworks.

CALLISTA

Network Attached Storage (NAS)

CALLISTA

Network Attached Memory (NAM)



CALLISTA

Terracotta

- Terracotta server (100% Java)
  - Can be configured in HA mode (active-passive).
  - Can be configured to persist all state on disk.
  - Handles distributed object, memory and locks.
- Terracotta clients
  - Is loaded into the JVM at boot time.
  - Instruments specified Java classes with cluster behaviour.
  - Automatically connects to the Terracotta server at boot time.
  - Can be started with specific wrapper script (dso-java.sh).

CALLISTA

Demo: Terracotta HelloWorld

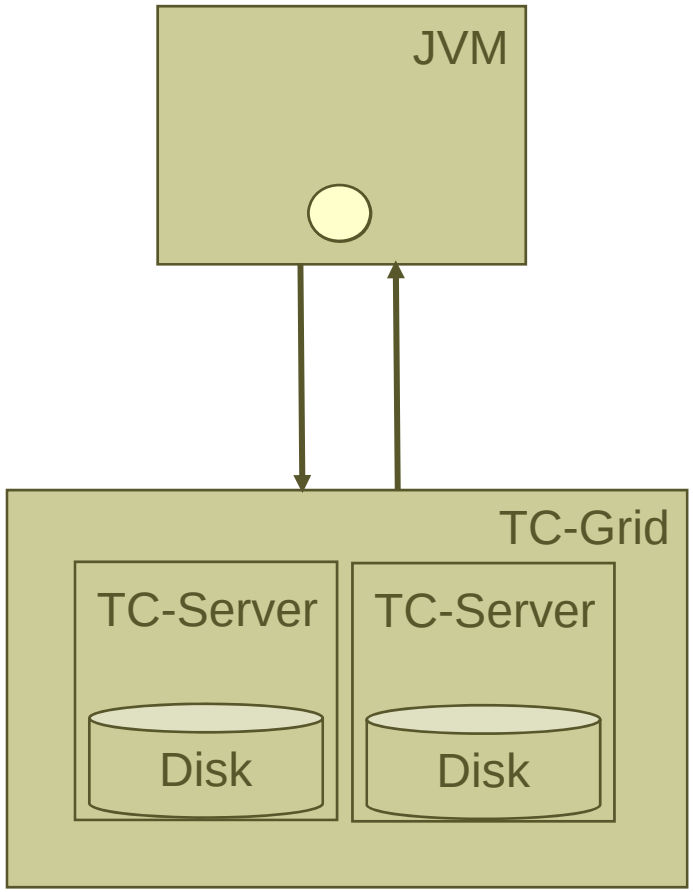
```
public class HelloWorld {
  private static int counter;

  public static void main(String [] args) {
    System.out.println("Hello world, counter=" + counter++);
  }
}
```


CALLISTA

How does it work?

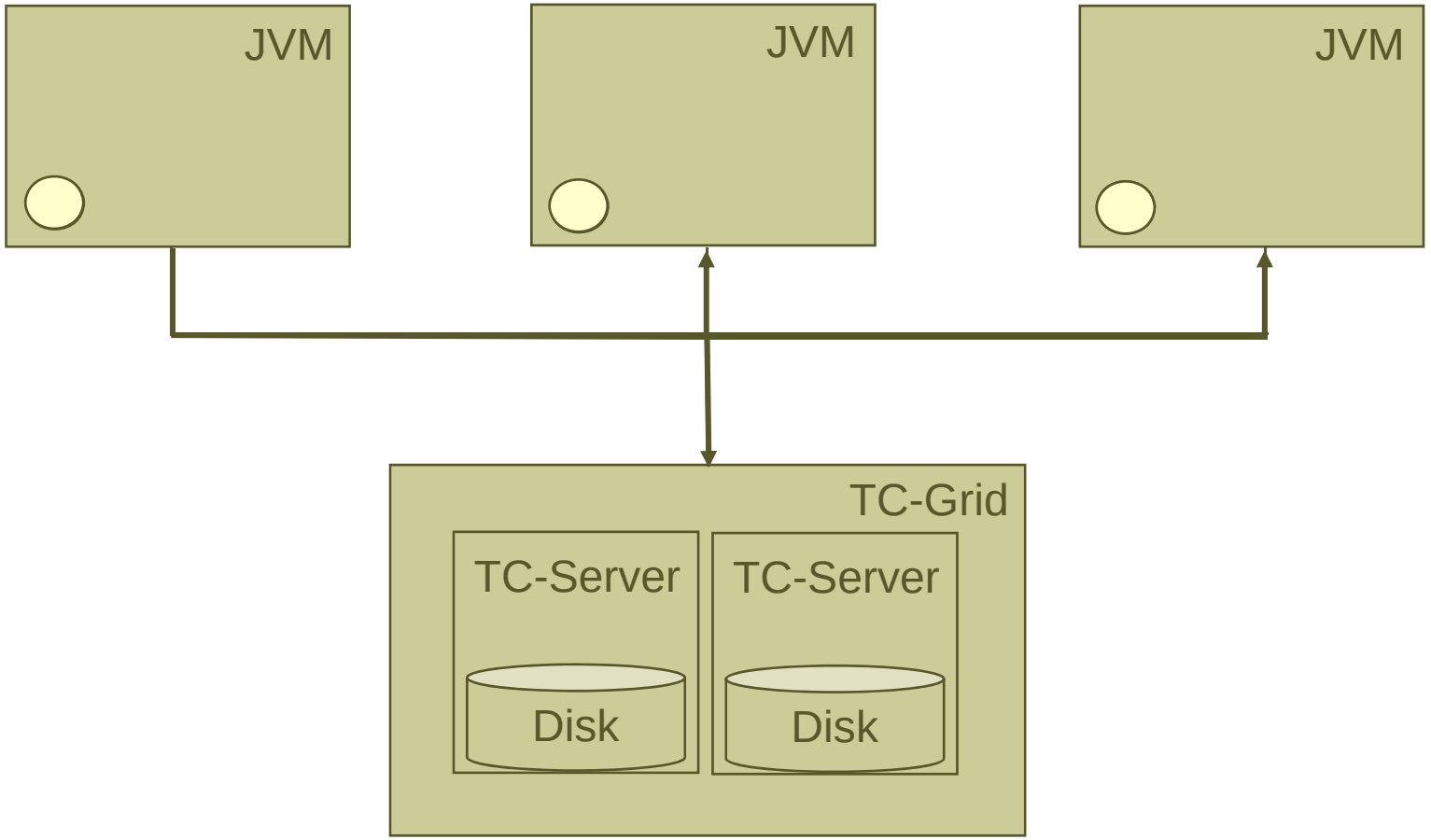
CALLISTA

How does it work?



CALLISTA

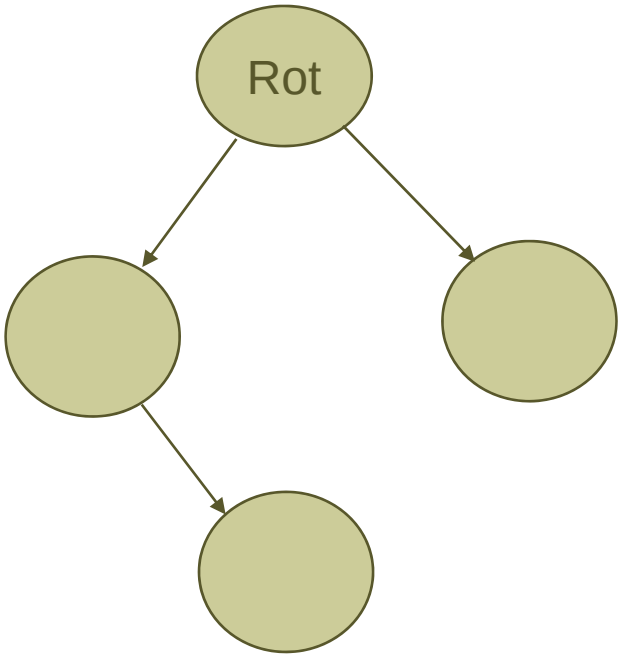
How does it work?



CALLISTA

How do you do it?

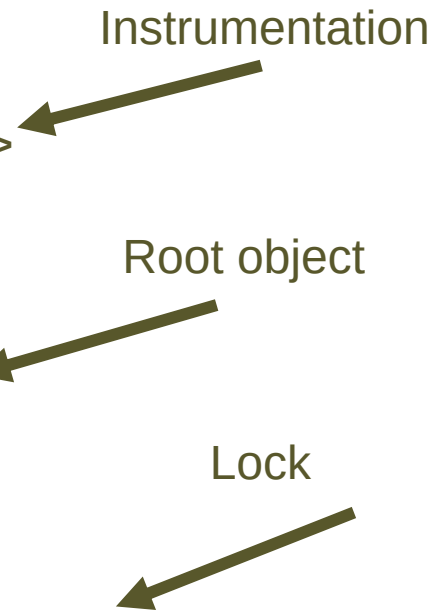
- Define root objects in your Java classes and instrument the classes that are to be clustered.
- All objects reachable from a root are clustered.



CALLISTA

Terracotta HelloWorld (configuration)

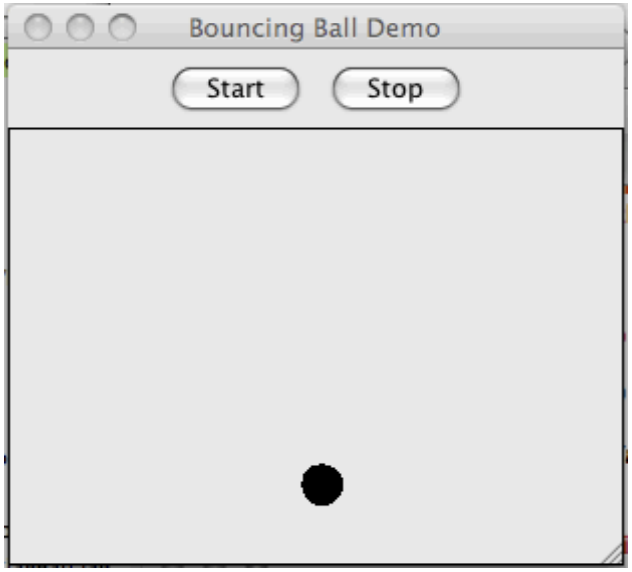
```
<dso>
   <instrumented-classes>
      <include>
         <class-expression>HelloWorld</class-expression>
      </include>
   </instrumented-classes>
   <roots>
      <root>
         <field-name>HelloWorld.counter</field-name>
         <root-name>counter</root-name>
      </root>
   </roots>
   <locks>
      <autolock auto-synchronized="false">
         <method-expression>* HelloWorld.main(..)</method-expression>
         <lock-level>write</lock-level>
      </autolock>
   </locks>
</dso>
```



CALLISTA

Demo: Bouncing Ball

- http://leepoint.net/notes-java/examples/animation/40BouncingBall/bouncingball.html



CALLISTA

Use cases for Terracotta

- Distributed caching (HashMap, EHCache)
- Session Replication (Out-of-the-box).
- Offload the database.
  - Handle objects that does not have to be stored in the database.
- Simple messaging (LinkedBlockingQueue).
- Workload partitioning.

CALLISTA

Demo: Simple Messaging (Writer)

```
import java.util.concurrent.*;

import static java.lang.System.*;

public class Writer {
   static final BlockingQueue<String> queue =
          new LinkedBlockingQueue<String>(5);

   public static void main(String [] args) throws Exception {
      out.println("Writer started...");
      for(int i = 0; i < 10 ; i++){
         queue.put("msg-"+i);
         out.println("Written msg-"+i);
      }
      queue.put("end");
      out.println("Writer done");
   }
}
```


CALLISTA

Demo: Simple Messaging (Reader)

```
import java.util.concurrent.*;

import static java.lang.System.*;

public class Reader {
   static final BlockingQueue<String> queue =
          new LinkedBlockingQueue<String>(5);

   public static void main(String [] args) throws Exception {
      out.println("Reader started...");
      boolean end = false;
      while(!end) {
         String msg = queue.take();
         out.println("Reader received:" + msg);
         if(msg.equals("end")) {end = true;}
      }
      out.println("Reader terminated!");
   }
}
```

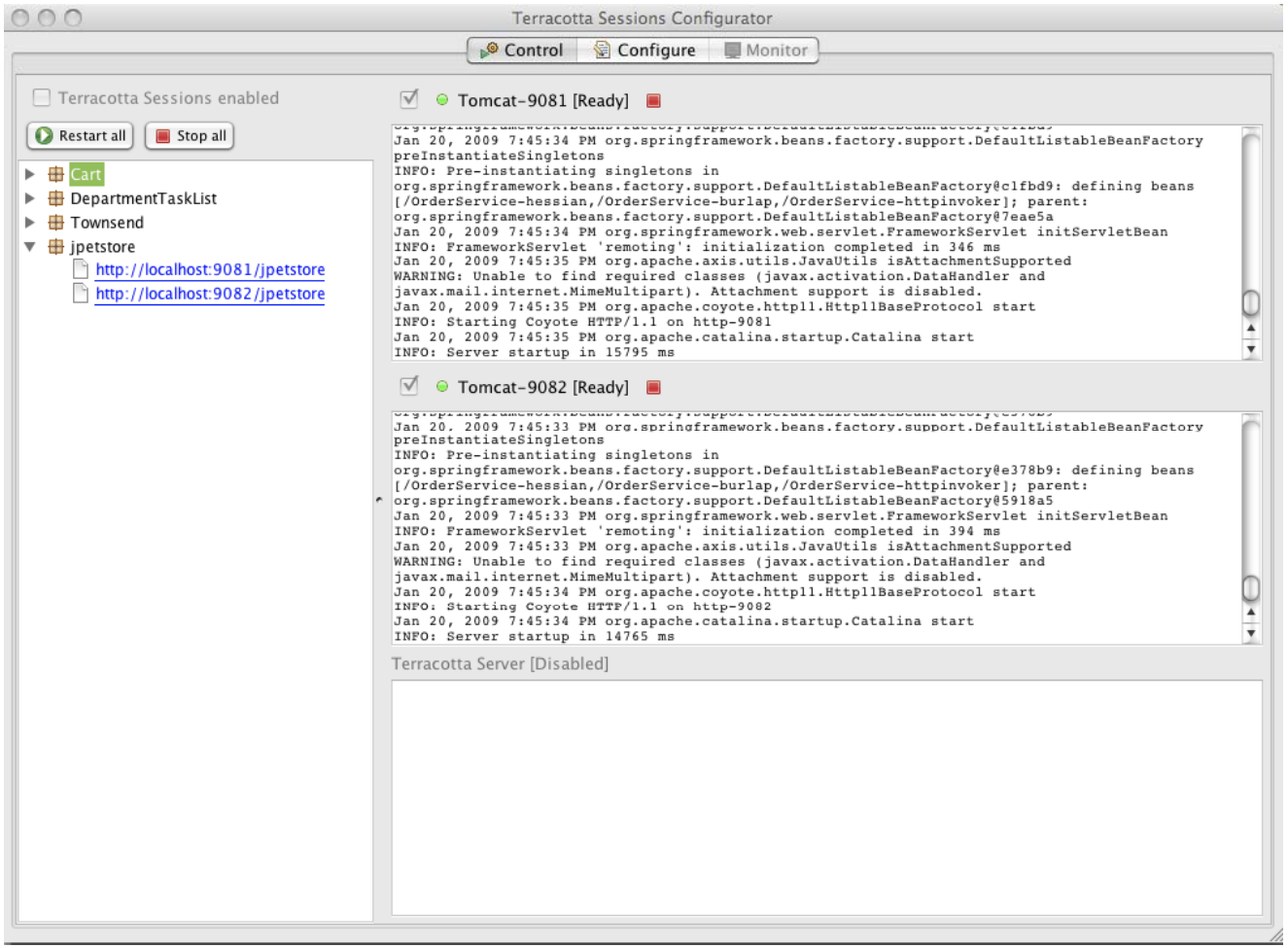

CALLISTA

Demo: Simple Messaging (Config.)

```
<application>
   <dso>
     <instrumented-classes>
       <include>
         <class-expression>Writer</class-expression>
       </include>
       <include>
         <class-expression>Reader</class-expression>
       </include>
     </instrumented-classes>
     <roots>
        <root>
          <field-name>Writer.queue</field-name>
          <root-name>queue</root-name>
       </root>
        <root>
          <field-name>Reader.queue</field-name>
          <root-name>queue</root-name>
       </root>
     </roots>
   </dso>
 </application>
```

CALLISTA

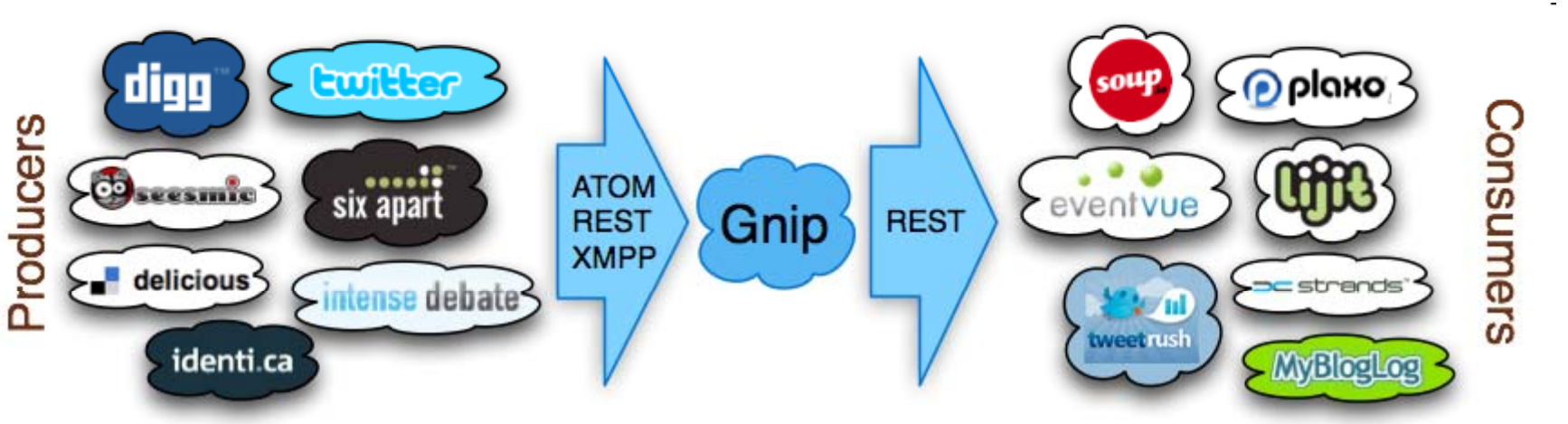
Demo: Session Replication



CALLISTA

Gnip Web 2.0 ESB

- Gnip: http://www.gnipcentral.com/
- Blog: http://blog.gnipcentral.com



CALLISTA

Gnip Web 2.0 ESB

- 99.9%: the Gnip service has 99.9% up-time.
- 10: ten Ec2 instances, of various sizes, run the core, redundant, message bus infrastructure.
- 2.5 million unique activities are HTTP POSTed (pushed) into Gnip's Publisher front door each day.
- 2.8 million activities are HTTP POSTed (pushed) out Gnip's Consumer back door each day.
- 2.4 million activities are HTTP GETed (polled) from Gnip's Consumer back door each day.
- $0: no money has been spent on framework licenses (unless you include "AWS").
- >50.000 Terracotta transactions per second.

CALLISTA

Conclusions and reflections

- Do we really need a RDBMS for this system?
- If we do, which objects needs to be stored in the database?
- We can build real object oriented domain models without constraints.

- Terracotta is no database!
- There is an integration module for Lucene/Compass.

- Terracotta is built to make the world simpler (for developer and operator).
- Terracotta has been open sourced for two years.

CALLISTA

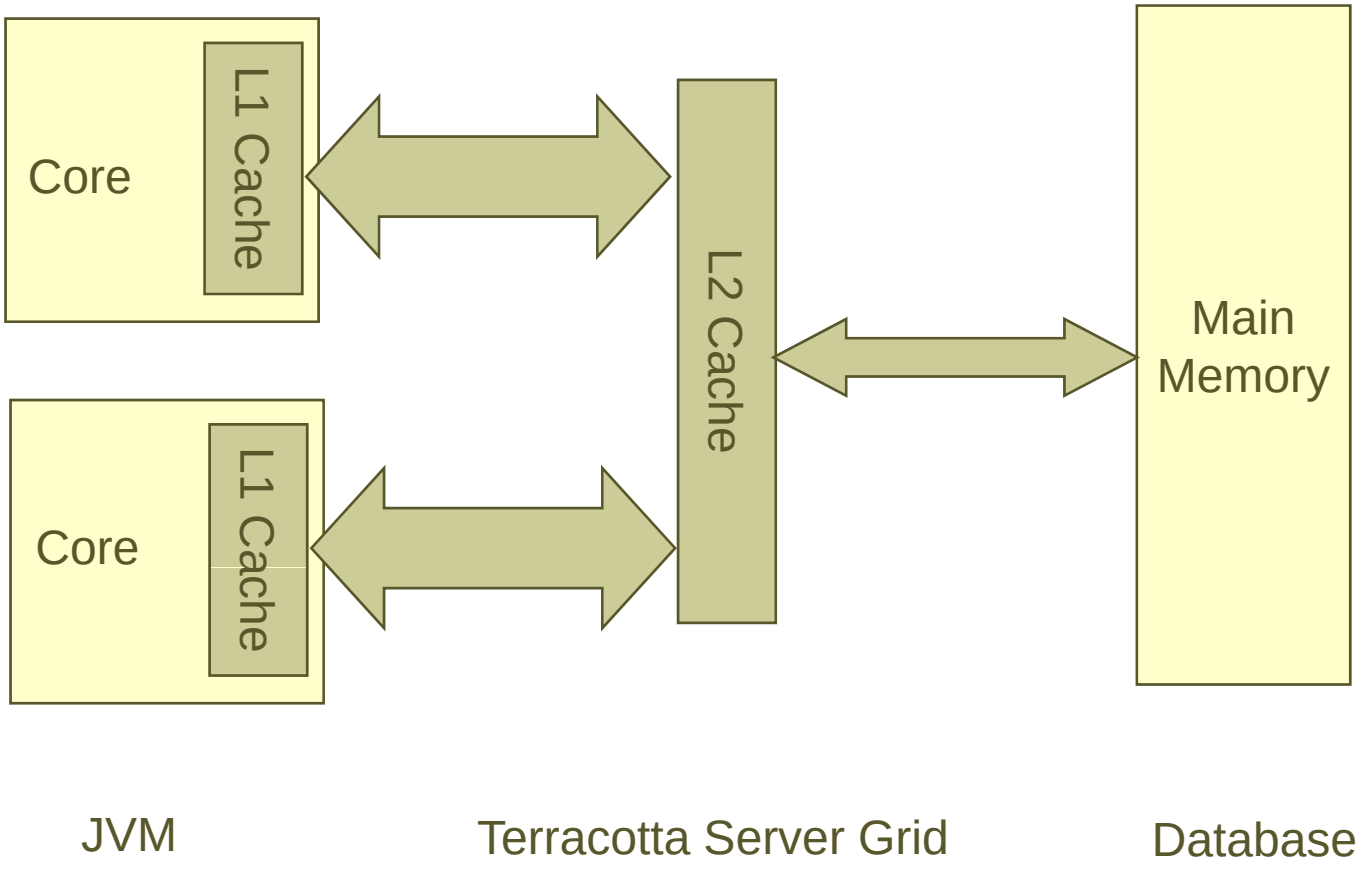
There is no silver bullet

CALLISTA

Extra slides

CALLISTA

Network Attached Memory (NAM)



CALLISTA