Extending the Java™ Runtime Plug-in Capacity and Availability for Java Technology

Ari Zilka

Terracotta, Inc.

TS-4219

Goal of This Session

- Learn how transparent clustering works at a high level
- Learn how clustering at runtime provides a simpler environment for development without hindering scale-out

LET'S START WITH A DEMO

Since a picture says more than a thousand words

Shared Figure Editor

Agenda

- Evolution of the Managed Runtime
- Why do this at runtime
- Why not at dev time
- Real-world Implementation: Terracotta Architecture
- Simple, scalable, and fault tolerant
- Use Case: Inventory Application

Java™ Specification Is Good

- Java language has a very strict and valuable set of semantics and rules that developers trust
  - Object Identity and Pass-by-reference:
    - map.put(“ID”, obj1);
    - Object obj2 = map.get(“ID”);
    - (obj1 == obj2)⇒true
  - Coordination between threads:
    - synchronized(..)
    - wait() notify()
    - (data integrity, race conditions, etc.)
- These natural rules of Java technology should not be broken
  - Breaking these rules open up many problems

But Java Technology Under a Load-Balancer Is Bad

- Replication infrastructure is not up to the task
  - The database is a single point of failure (SPoF)
  - Message queuing or JGroups bottlenecks on the network
  - Buddy systems cannot react to cascading failure
- Serializing objects is not fun
  - Breaks your object graph and domain model
  - Leads to coarse-grained replication regardless of delta
- Tuning is never-ending
- What is needed is a JRE service that handles these issues transparently… at runtime

Cluster at Runtime… It Is Not Unprecedented

- Oracle RAC
  - Vendor knows best
  - High performance
  - Easy upgrade path from small to medium to large deployments
- Cisco IOS
  - Immediate failover
  - Easier than BGP, routing table maintenance
- In all cases, it is black boxed
  - I don't use it till I need it
  - I don't write a different app in its presence
- Load balancing is good; load balancing with application-level consistency is best

And, Managed Runtimes Relieve Developers

- Example: Memory Management
  - Remember malloc() and free(), heap vs. stack
  - The JVM™ software introduced most developers to garbage collection, but compile-time used to win
  - Today, the JVM software is faster; it decides what do do at runtime instead of at compile time
    - More information available at runtime
    - "…only 12 times faster than C means you haven't started optimizing" —Doug Lea
- Other Java technology features that make Developers' lives easier:
  - Platform-independent thread coordination/locking
  - Fat/thin locks in Jrockit decided at runtime
  - Platform-specific optimizations

Impact of Development-Time Solutions

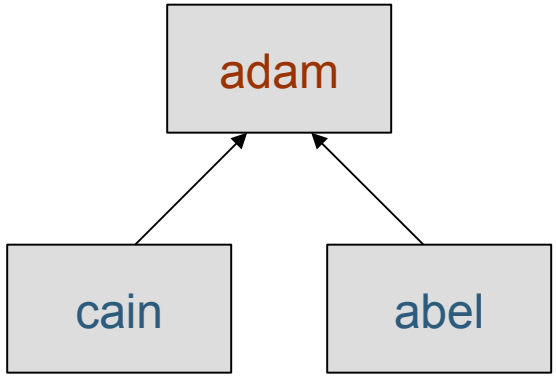
- Most existing caching/clustering solutions are API based
- Roughly cache.get() and cache.put()
  - This is a simplified view, of course, transactions, replication schemes, fault tolerance, etc. are also included
- These APIs affect simplicity
- These APIs affect scalability

Scale-Out or Simple: APIs Are Not Simple

- Scale-out solutions rely on Java language serialization
- This breaks object identity
  - Data put into the cache and then read back will fail:
  - (obj == obj) ⇒ false
- Perturbs the Domain Model
  - Management of object references using primary keys
- Adds new coding rules
  - Need to put() changes back—easy to forget
  - Can't trust callers outside the caching class to put a top-level object back in the cache if they edited it
- This is not as simple as the Java language can be

Impact of Java Language Serialization

```
// let's create one father and two sons
Person adam = new Person("Adam", null);
Person cain = new Person("Cain", adam);
Person abel = new Person("Abel", adam);
```



Object Identity **Is** preserved

API-based Clustering Is No Longer Simple

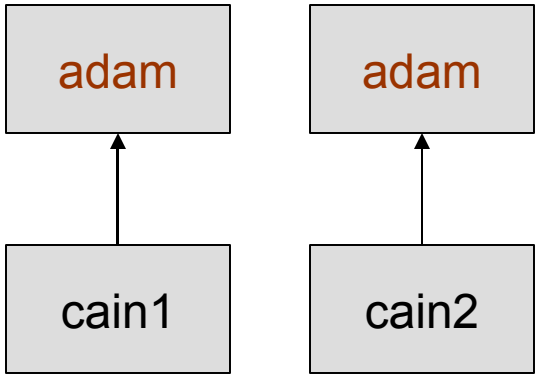
```
// create Cain and put him in the distributed map
Person cain1 = new Person("Cain", adam);
distMap.put("Cain", cain1);


...
// later in time we want to modify Cain
// so we have to get Cain out of the map
Person cain2 = (Person)distMap.get("Cain");
cain2.addBrother(abel);
// then we need to put him back into the map
distMap.put("Cain", cain2);
```

Traditional Clustering (Cont.)

- Why is it needed to get the object out of the map?
  - Don't we already have a reference to it?
- Why is it needed to put the object back into the map?
- Is it not there already, under the correct key?

Object Identity
Is NOT preserved

You end up with
two distinct
object graphs



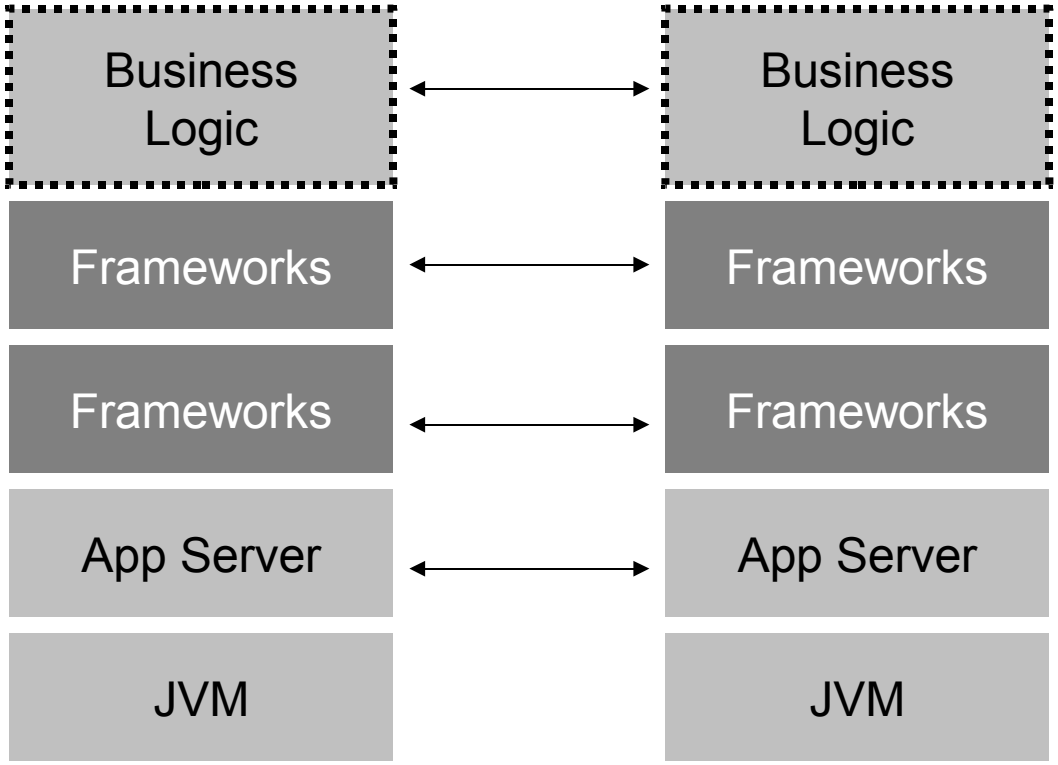
Scale-Out or Simple: APIs Are Not Scalable

- Java language serialization is not scalable
- There is a high cost to serialization
  - field updates⇒push object graph⇒too much data
  - Coarse-grained locks⇒locking a top-level object in cache, regardless of scope of change⇒premature lock contention
- There is a high cost to scale-out
  - DB sees too many clients
  - Clustering takes immeasurable JVM software resources (not “too much” just “not factored”)

Clustering at Runtime…

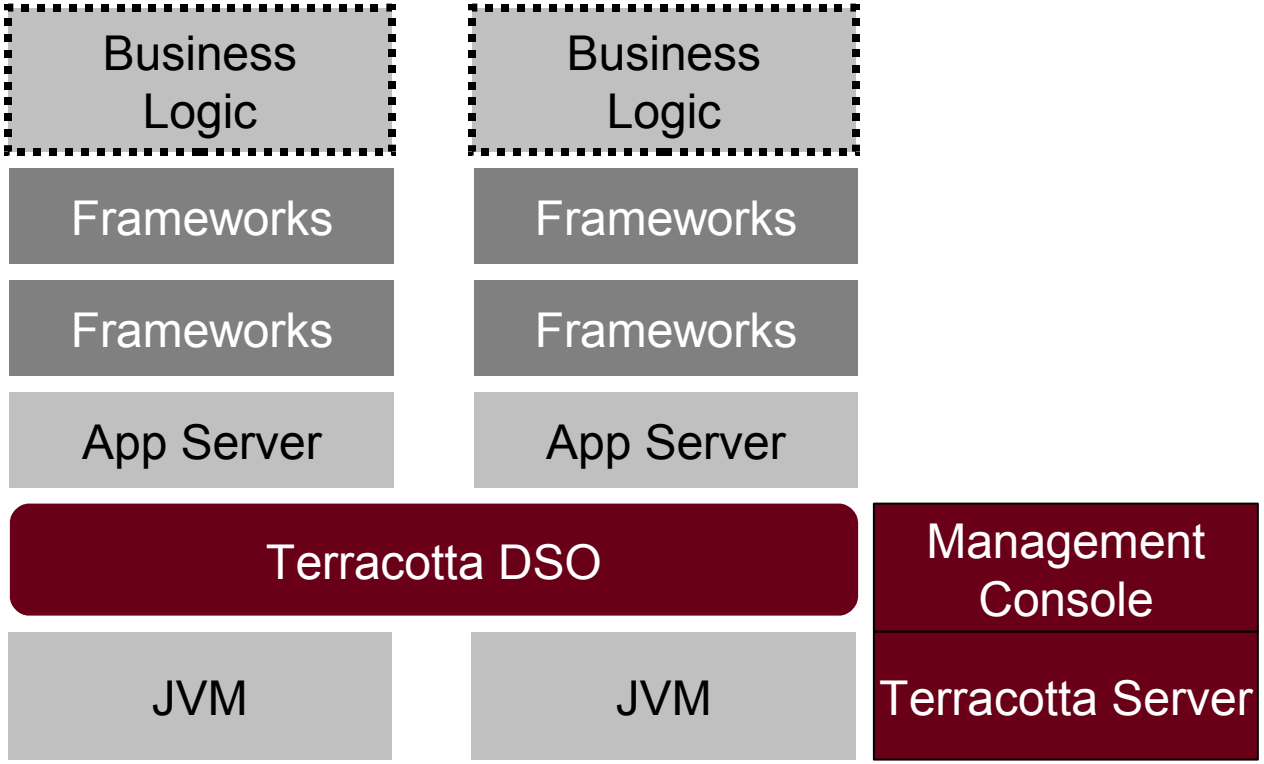
- TC let's you cluster Java technology in a natural fashion
  - No API
  - Zero code
- How is that possible?
- Terracotta is Instrumenting the appropriate level
  - Heap-level memory read/write operations
  - JDBC™ API Driver-level embedded caching
  - Network-based clustering with consistency
  - Transparent to the business logic (think: VMWare for the Java platform turned on its head)

Take Your Applications From This…

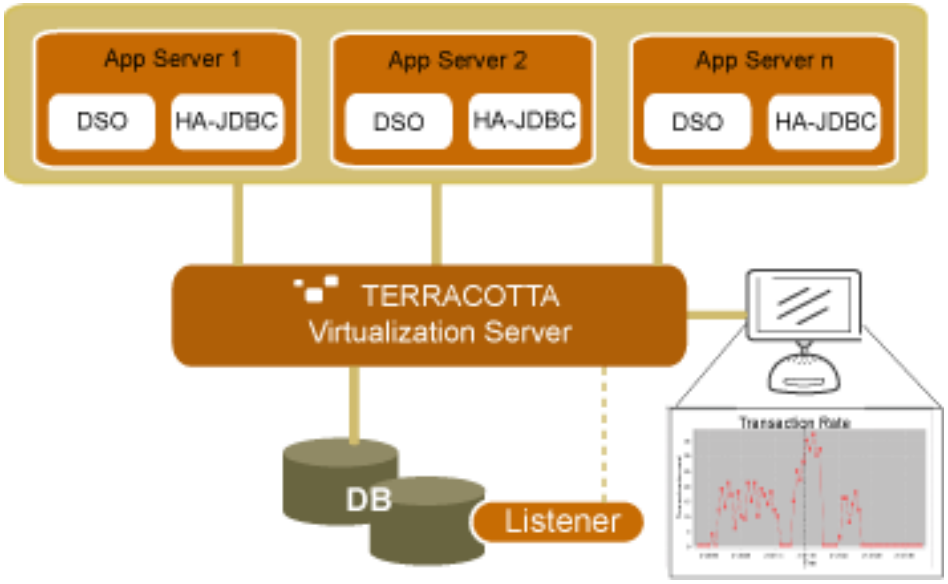


Clustered App Servers Are Expensive

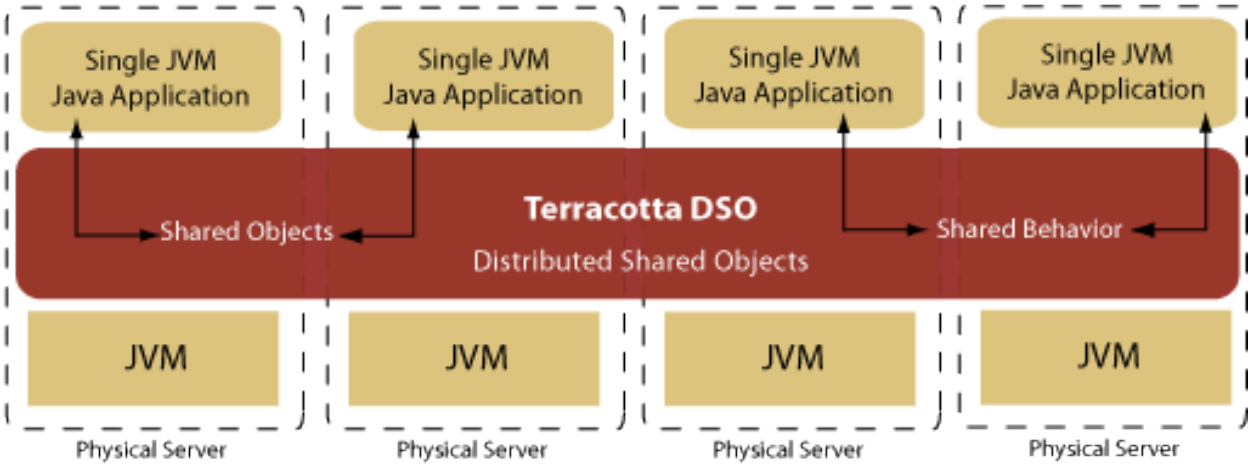
…To This


Business Logic
Business Logic
Frameworks
Frameworks
Frameworks
Frameworks
App Server
App Server
Terracotta DSO
Management Console
JVM
JVM
Terracotta Server

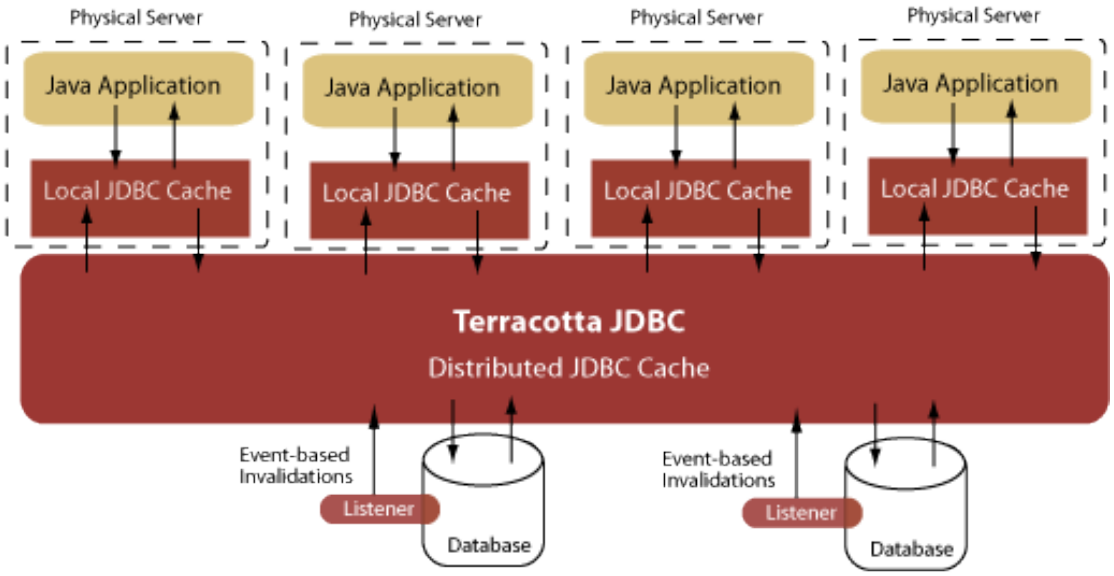
Terracotta Architecture



Distributed Shared Objects



Terracotta's Implementation of the JDBC API



DEMO 2

Shared JTable (Spreadsheet)

Entire Application

```
package demo.jtable;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

class TableDemo extends JFrame {

    // Shared object
    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);
    }
}
```

Magic Is in Config File

```
<terracotta-config>
  <dso>
    <server-host>localhost</server-host>
    <server-port>9510</server-port>
    <dso-client>

      <roots>
        <root>
          <field-name>demo.jtable.TableDemo.model</field-name>
        </root>
      </roots>

      <included-classes>
        <include><class-expression>demo..*</class-expression></include>
      </included-classes>

    </dso-client>
  </dso>

</terracotta-config>
```

Zero Impact and Still Scalable?

- Hub and Spoke as a SPOF?
- Field-level changes too chatty?
- Networking overhead to clustering?

Zero Impact With Scale: Having Your Cake…

- Hub and Spoke ⇒ scale the hub
- Field-level changes ⇒ batched
- Network overhead ⇒ runtime optimized

- So, we can have our cake and eat it too. Let's now look at natural Java technology that clusters at runtime.

Use Case: Inventory Demo

- Simple domain model
- Add nodes at runtime to scale-out
- Restart without losing state
- Runtime console provides visibility

- NOTE: There is a JDBC technology version of this demo as well that assumes a DB exists but will run w/o one for a time

Learn More…

http://www.terracottatech.com/

http://blog.terracottatech.com/

Summary

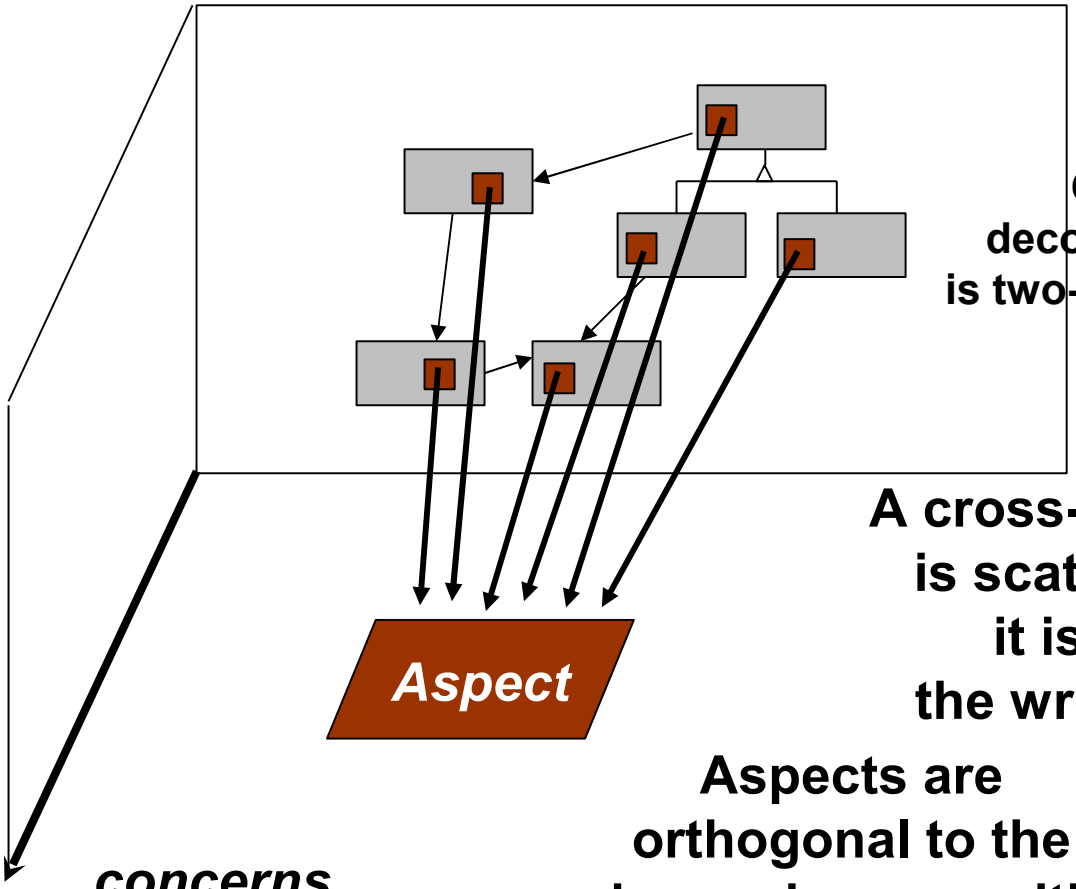
- Infrastructure services are the responsibility of the Runtime, not the developer
- New APIs are not the answer
- Technology exists to cluster and cache transparently today
- The value is in getting scale-out with simplicity

Appendix

AOP-Style Techniques Make It Possible

- Aspect-Oriented Programming is all about Separation of Concerns
  - Cross-cutting concerns: Issues in an application that cut across an application
- Some AOP frameworks allow transparent injection of these concerns at runtime and/or load time
  - AspectWerkz, AspectJ 5, Spring AOP
- Enterprise scalability (e.g., clustering, caching) services are ideal use cases for using AOP to inject transparently at runtime

AOP Adds a New Dimension (Orientation)

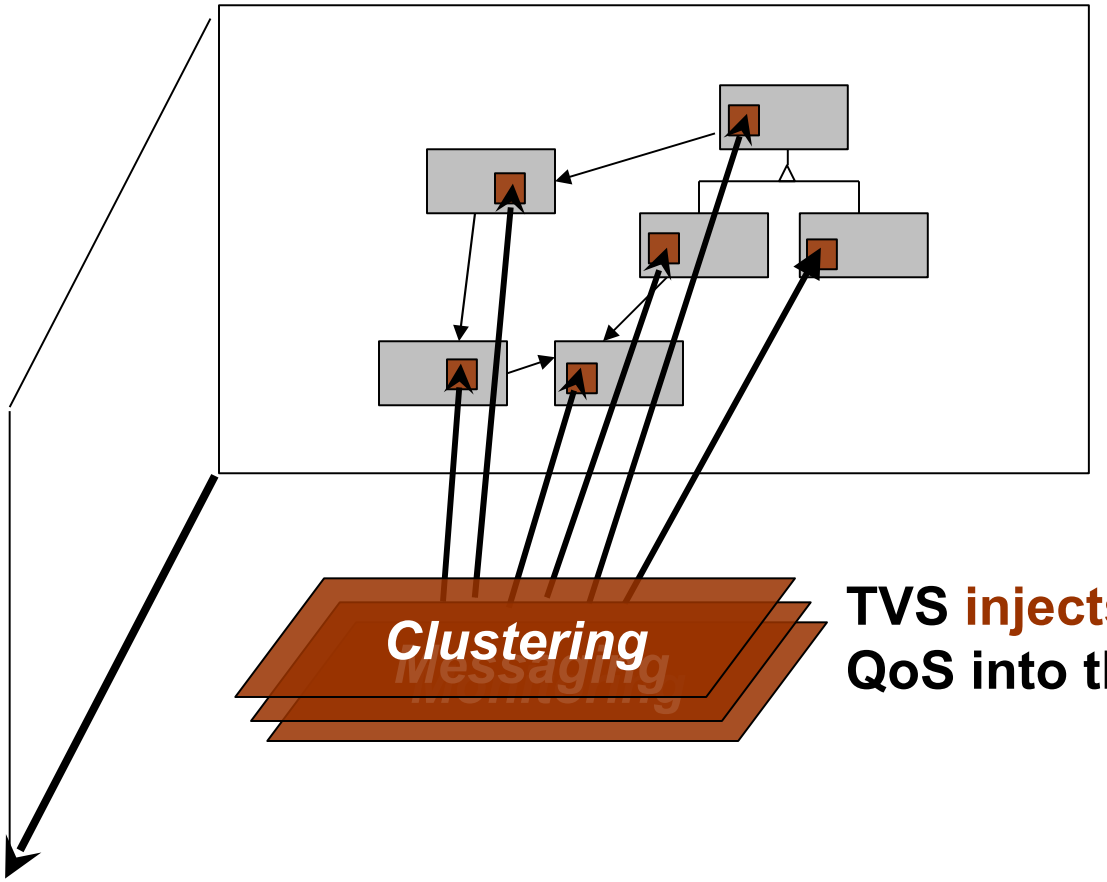


Object decomposition is two-dimensional

A cross-cutting concern is scattered because it is realized in the wrong dimension

Aspects are orthogonal to the primary decomposition

We Inject Quality of Services Transparently at Runtime



Developer **focuses solely on the business logic**, using POJOs, Spring Beans, EJB™ beans, etc.

TVS **injects pre packaged** QoS into the application

New APIs, Not the Answer

- Plenty of Java technology APIs exist for distributed computing
  - RMI, Java Message Service (JMS), JCache maps, etc.
- We can extend the semantics of plain Java language:
  - APIs: synchronized, Thread, wait(), notify(), Java collection classes etc.
  - Bytecode Instructions: INVOKEXXX, MONITOR_ENTRY, MONITOR_EXIT, PUTFIELD etc.
- More detail later…

Q&A

Extending the Java™ Runtime Plug-in Capacity and Availability for Java Technology

Ari Zilka

Terracotta, Inc.

TS-4219