Clustered Architecture Patterns: Delivering Scalability and Availability

Qcon London, 2008

Ari Zilka – Terracotta CTO and Founder

TERRACOTTA

Confidential - for information of designated recipient only. Copyright Terracotta 2006

Agenda

- Patterns from Years of Tier-Based Computing
- Network Attached Memory / JVM-level clustering
- Applying NAM To Eliminate the DB
- Use Case #1: Hibernate
- Use Case #2: Service Orientation
- Lessons Learned

Confidential – for information of designated recipient only. Copyright Terracotta 2006

The State Monster

- At Walmart.com we started like everyone else: stateless + load-balanced + Oracle (24 cpus, 24GB)
- Grew up through distributed caching + partitioning + write behind
- We realized that "ilities" conflict
  - Scalability: avoid bottlenecks
  - Availability: write to disk (and I/O bottleneck)
  - Simplicity: No copy-on-read / copy-on-write semantics (relentless tuning, bug fixing)
- And yet we needed a stateless runtime for safe operation
  - Start / stop any node regardless of workload
  - Cluster-wide reboot needed to be quick; could not wait for caches to warm
- The "ilities" clearly get affected by architecture direction and the stateless model leads us down a precarious path

Confidential – for information of designated recipient only. Copyright Terracotta 2006

The Precarious Path: Our tools lead us astray

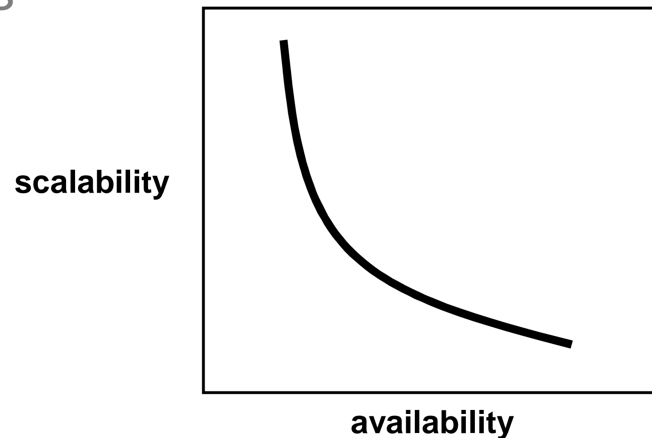
- Stateless load-balanced architecture ⇒ bottleneck on DB
- In-memory session replication ⇒ bottleneck on CPU, Memory
- Clustered DB cache ⇒ bottleneck on Memory, DB
- Memcache ⇒ bottleneck on server
- JMS-based replicated cache ⇒ bottleneck on network

- …Pushing the problem between our app tier CPU and the data tier I/O

Confidential – for information of designated recipient only. Copyright Terracotta 2006

CRUD Pules Up…

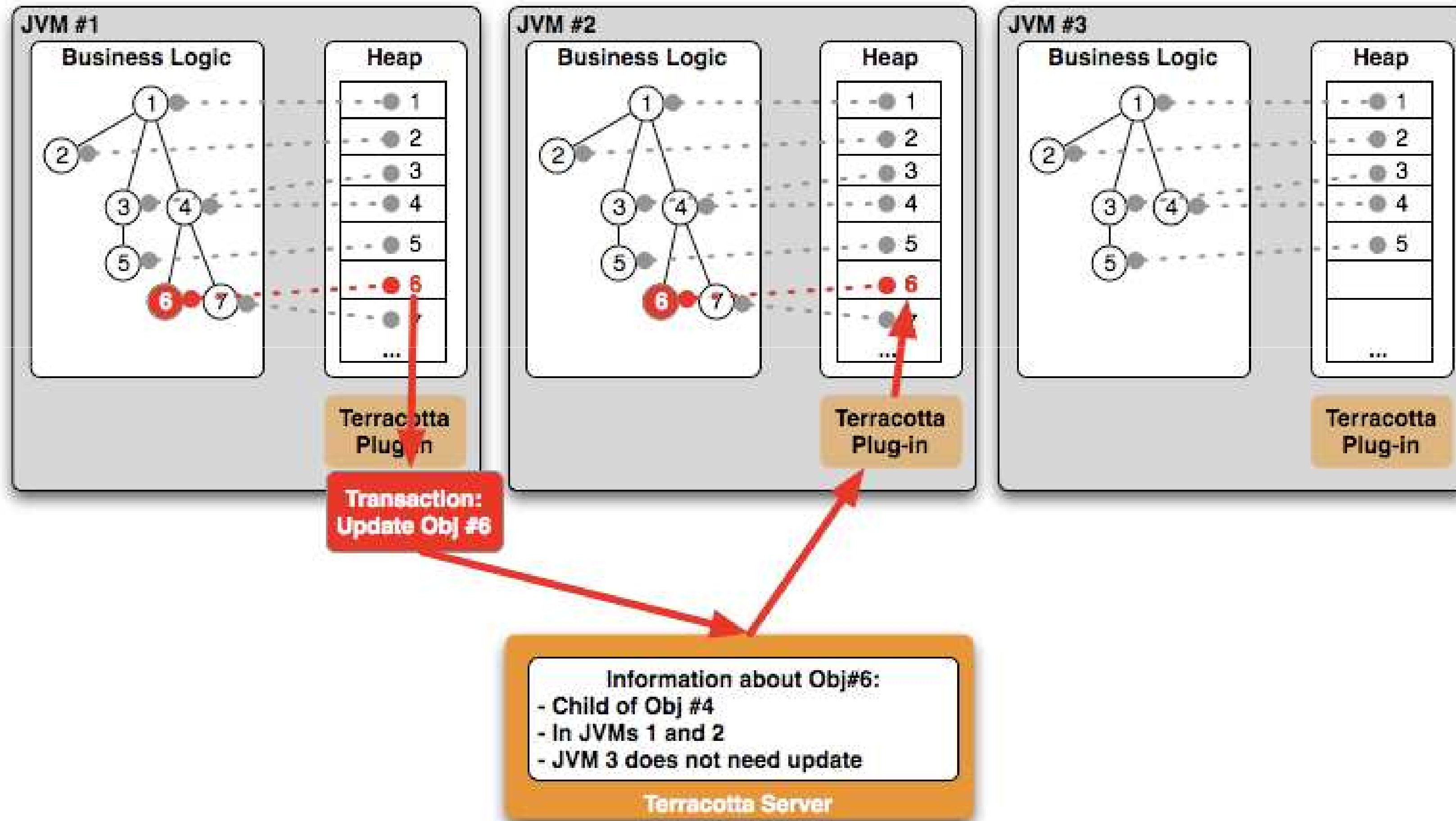
- Types of clustering:
  - Load-balanced (non-partitioned) Scale Out
  - Partitioned Scale Out
- Both Trade-off Scalability **or** availability (usually by hand) in different ways



- …and everything we do forces the trade-offs

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Changing the Assumptions: JVM-level Clustering



Confidential – for information of designated recipient only. Copyright Terracotta 2006

Performance + Reliability

- 10X throughput over conventional APIs
  - All Reads from Cache (implicit locality)
  - All Writes are Deltas-only
  - Write in log-forward fashion (no disk seek time)
  - Statistics and Heuristics (greedy locks)
- Scale out the Terracotta Server
  - Simple form of Active / active available today
  - V1.0 GA this year

Confidential – for information of designated recipient only. Copyright Terracotta 2006

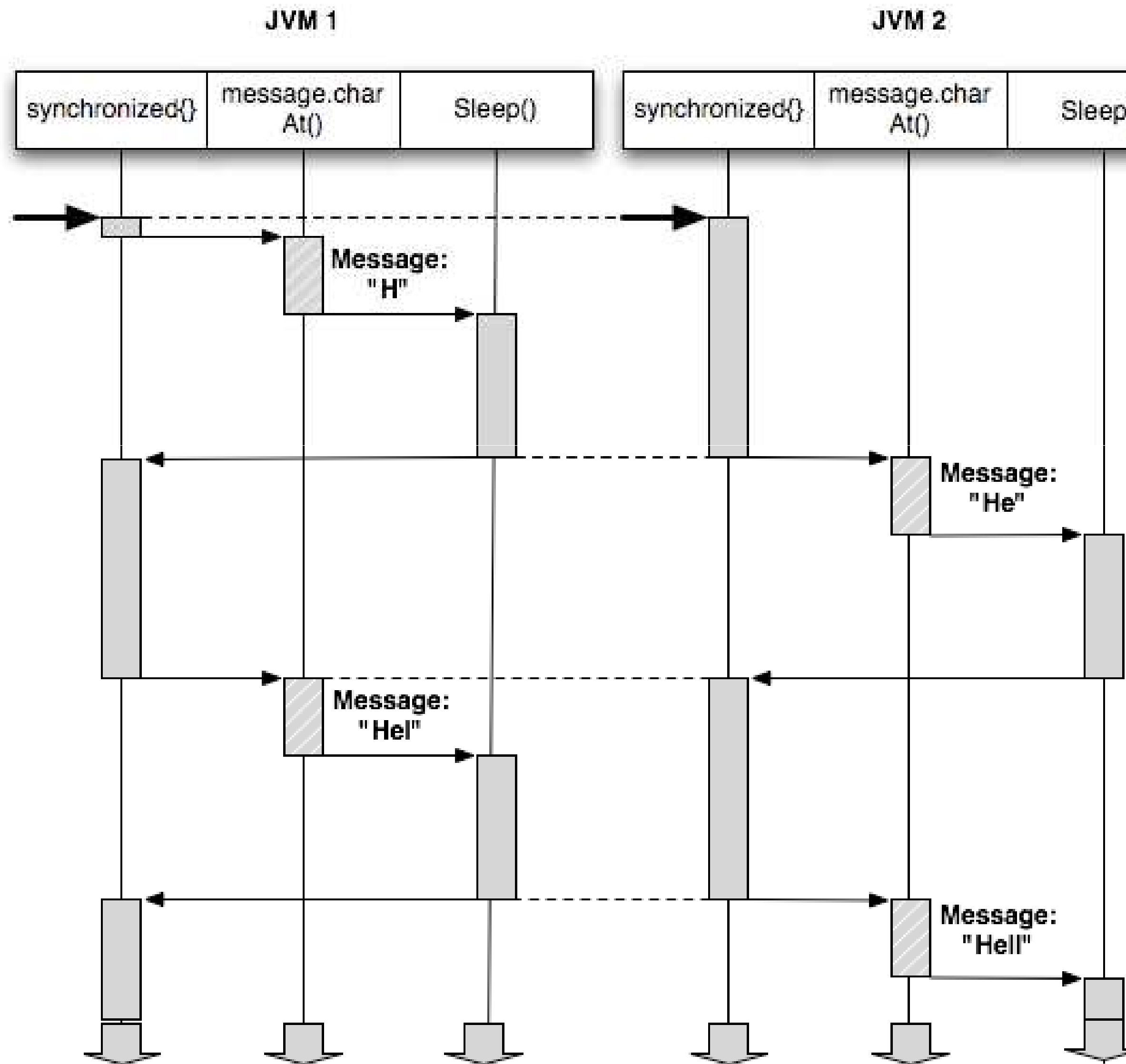
HelloClusteredWorld (from our pending Apress book)

- Chapter 3: Definitive Guide to Terracotta

```
public class HelloClusteredWorld {
  private static final String message = "Hello Clustered World!";
  private static final int length = message.length();

  private static char[] buffer = new char [length ];
  private static int loopCounter;

  public static void main( String args[] ) throws Exception {
    while( true ) {
      synchronized( buffer ) {
        int messageIndex = loopCounter++ % length;
        if(messageIndex == 0) java.util.Arrays.fill(buffer, '¥u0000');

        buffer[messageIndex] = message.charAt(messageIndex);
        System.out.println( buffer );
        Thread.sleep( 100 );
      }
    }
  }
}
```

Confidential – for information of designated recipient only. Copyright Terracotta 2006

HelloClusteredWorld Sequence Diagram



Confidential – for information of designated recipient only. Copyright Terracotta 2006

TERRACOTTA

HelloClusteredWorld Config File

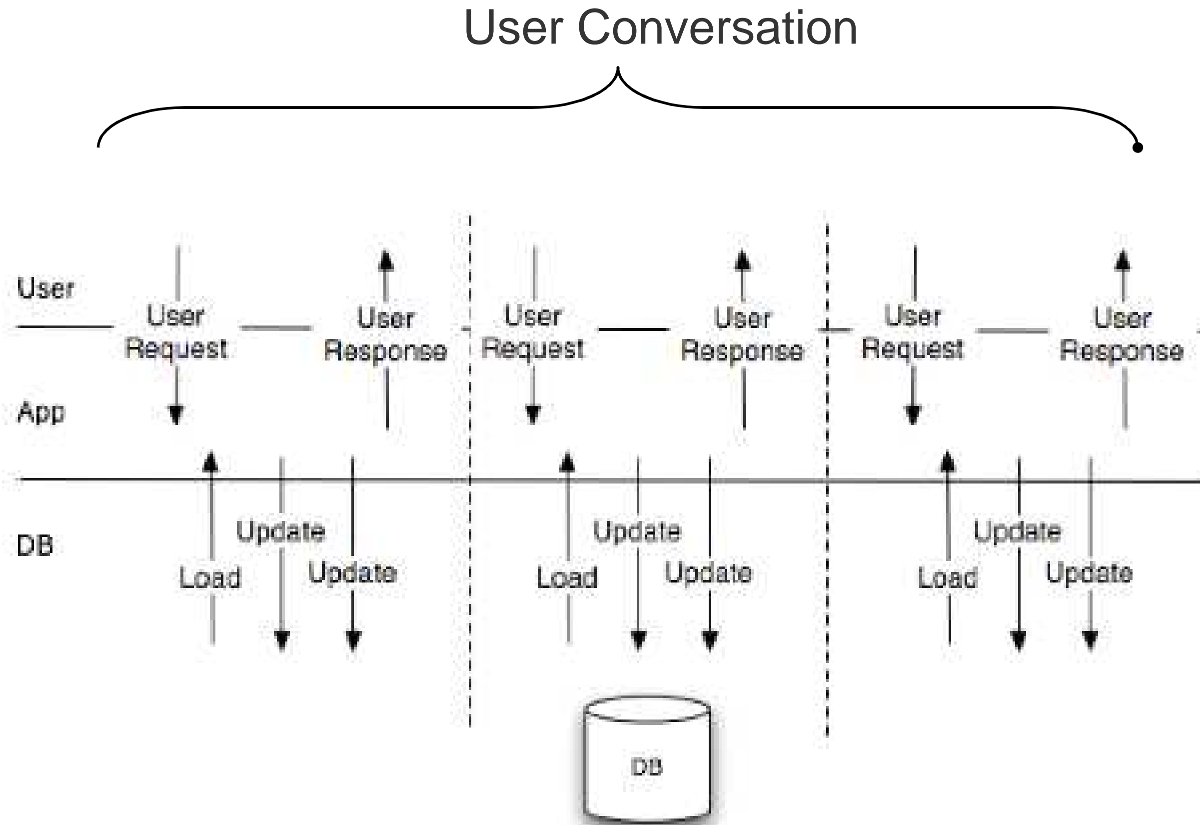
```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.terracotta.org/schema/terracotta-4.xsd">

<!-- servers and clients stanzas ommitted -->
  <application>
    <dso>
      <roots>
          <root>
            <field-name>HelloClusteredWorld.buffer</field-name>
          </root>
          <root>
            <field-name>HelloClusteredWorld.loopCounter</field-name>
          </root>
      </roots>
      <instrumented-classes>
        <include>
          <class-expression>HelloClusteredWorld</class-expression>
        </include>
      </instrumented-classes>
      <locks>
        <autolock>
          <lock-level>write</lock-level>
          <method-expression>void HelloClusteredWorld.main(..)</method-expression>
        </autolock>
      </locks>
    </dso>
  </application>
</tc:tc-config>
```

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Applying NAM To DB Offload

Confidential – for information of designated recipient only. Copyright Terracotta 2006

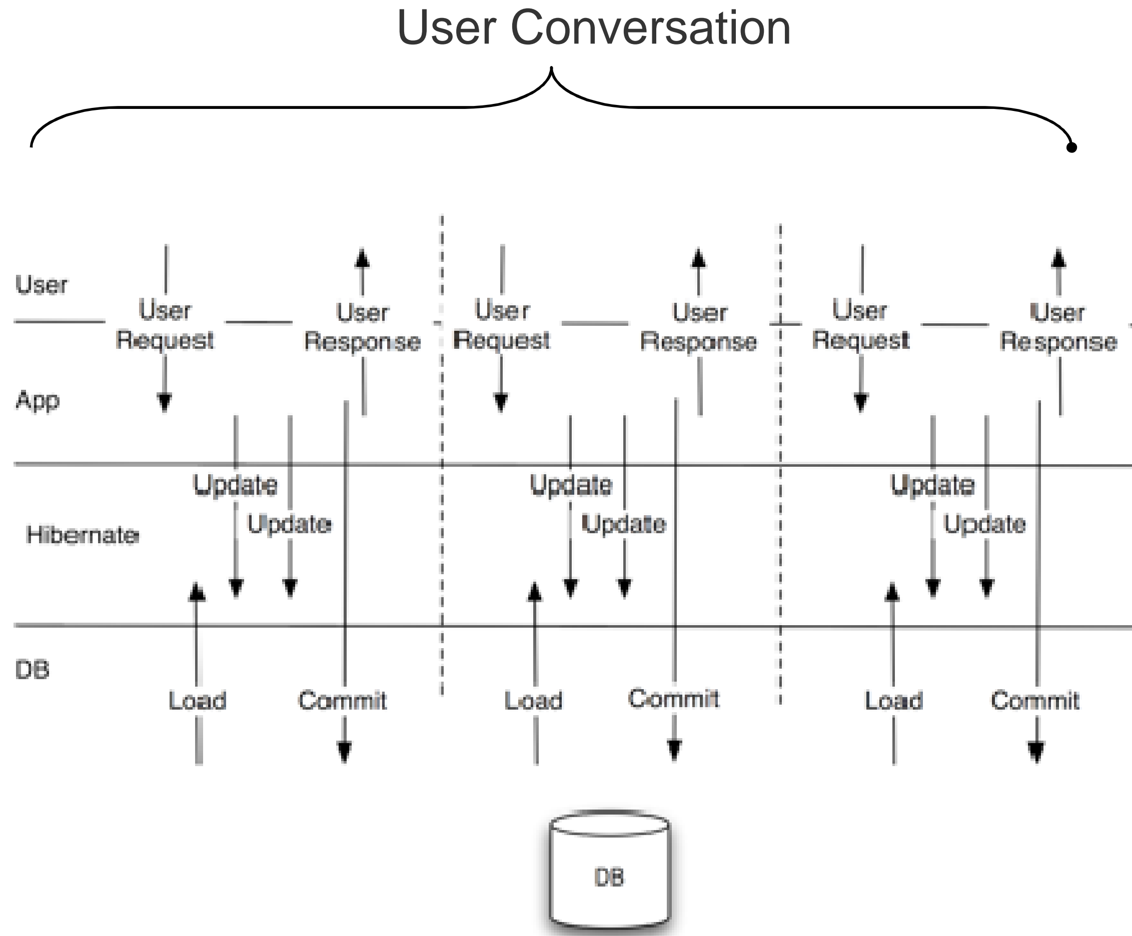
Stateless By Hand is Cumbersome and Inefficient



- Baseline Application
- 3 User Requests during one Conversation
- 2 POJO Updates per Request
- Total DB Load: 9

Confidential – for information of designated recipient only. Copyright Terracotta 2006

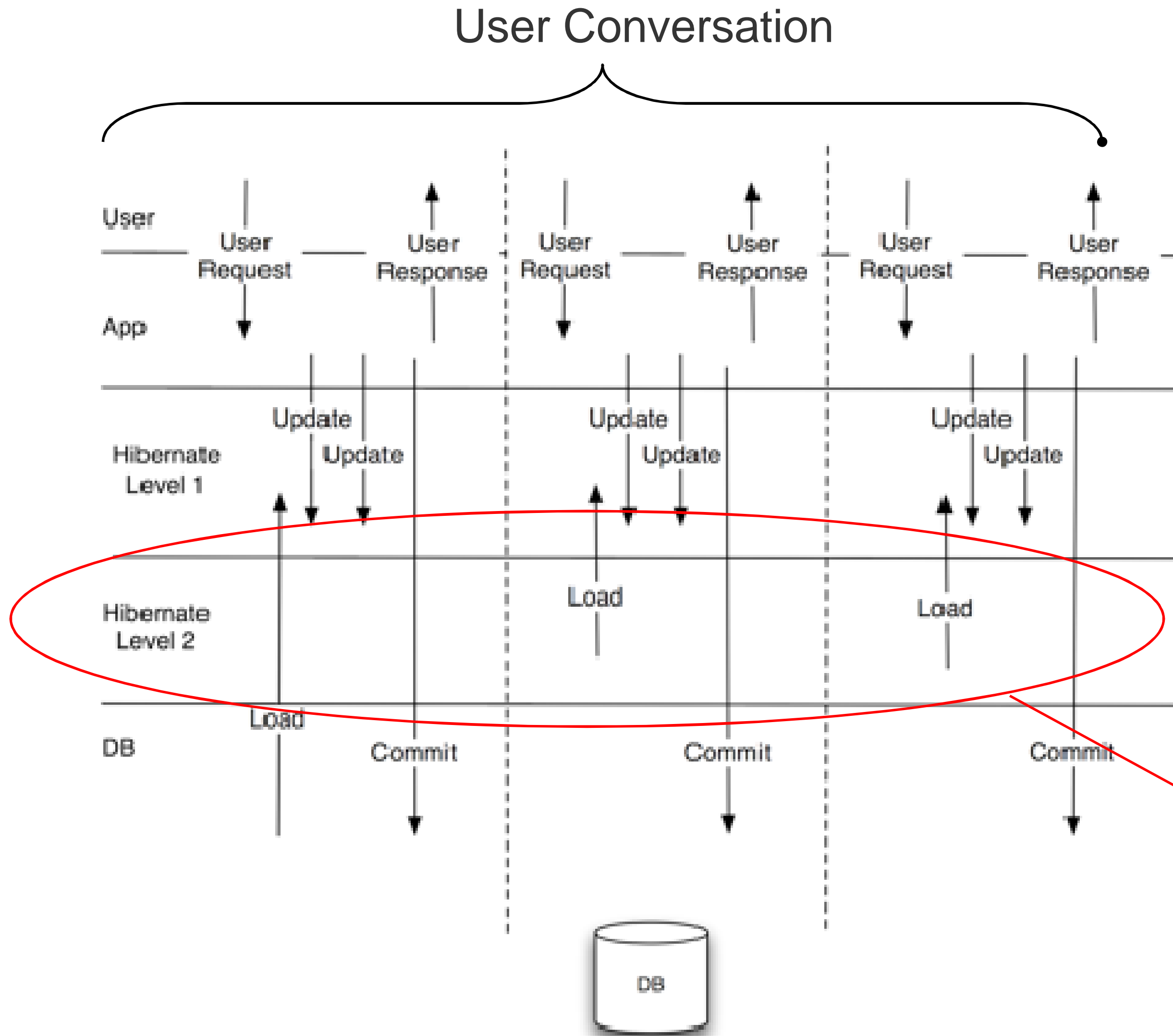
So We Add Hibernate



- Add Hibernate
- Eliminate Direct Connection to the DB via JDBC
- Eliminate Hand-Coded SQL
- Eliminate Intermediate POJO Updates
- Total DB Load: 6

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Then We Turn on Caching

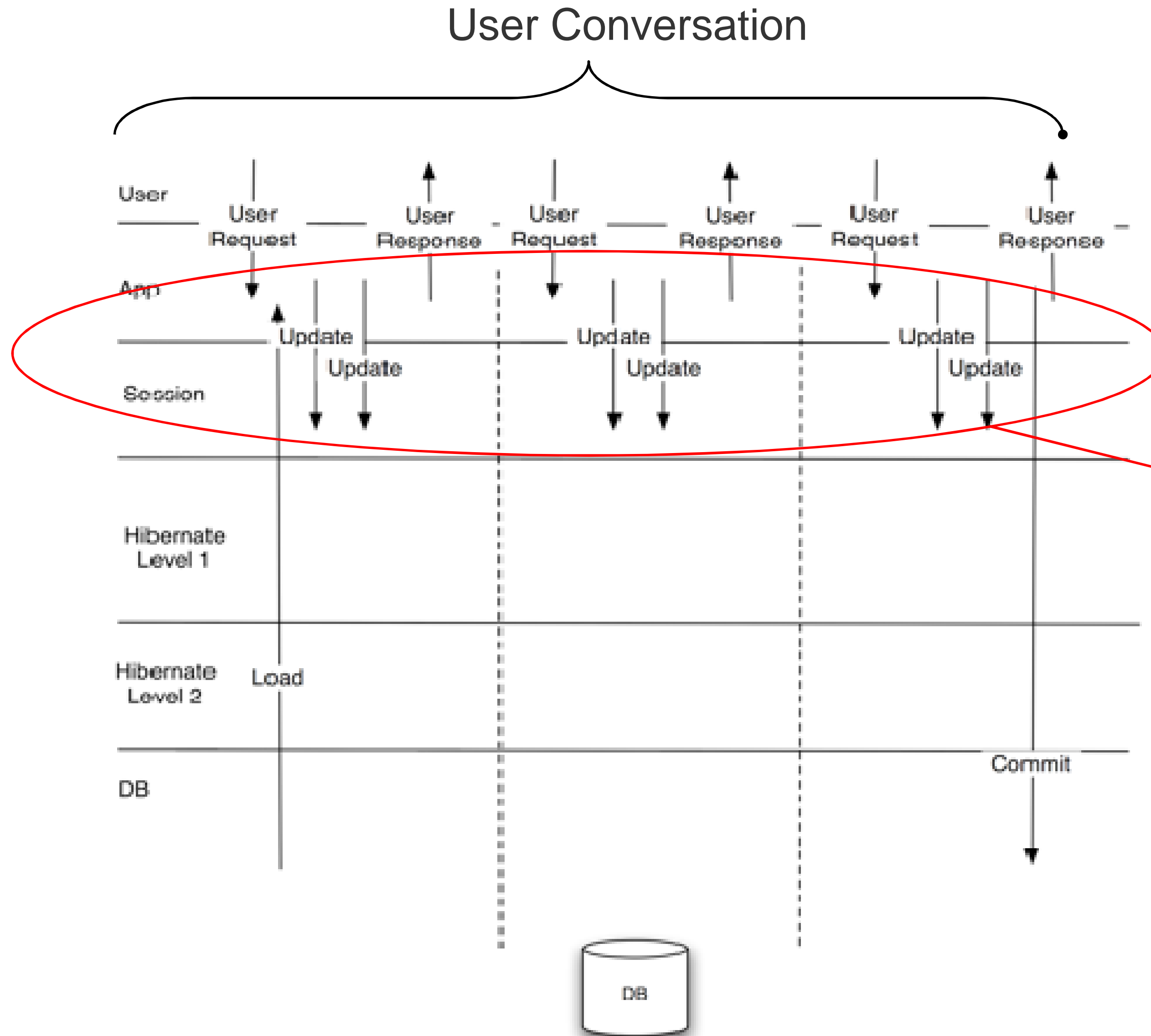


- Enable 2nd Level cache
- Eliminates Intermediate Loads
- Total DB Load: 4

Serialization is required
BLOB Replication requirements are heavy

Confidential – for information of designated recipient only. Copyright Terracotta 2006

So We Disconnect But Lose Availability



- Detached POJOs
- Eliminates Intermediate Commits
- Total DB Load: 2

Can lose state in case of failure!
Replication is expensive
Hibernate says to keep graphs small

JVM-Level Clustering + Hibernate Together

- Cluster 2nd Level Cache - Hibernate Performance Curve Level 2
  - EHCache Support Built in to the product
  - Advantages
    - Coherent Cache Across the cluster
    - Easy to integrate with existing applications
    - Performs very well
    - Eliminate the artificial cache misses in clustered environment
  - Disadvantages
    - Objects are represented as BLOBs by Hibernate
    - Doesn't take direct advantage of Terracotta Scale-Out Features
- Cluster Detached POJOs - Hibernate Performance Curve Level 3
  - Cluster Pure POJOs
  - Re-attach Session in the same JVM or a different JVM
  - Advantages
    - Scales the best
    - Take Advantage of POJOs - Fine-grained changes, replicate only where resident
  - Disadvantages
    - Some code changes required to refactor Hibernate's beginTransaction(), commit()

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Demonstration Application

- Simple CRUD application
  - Based on Hibernate Tutorial (Person, Event)
  - Already Refactored for Detached POJOs
  - Simple Session Management in Terracotta Environment - POJO wrapper
  - Detached Strategy requires a flush operation
- CREATE OPERATION
  - Creates a new Person
- UPDATE OPERATION
  - UpdateAge -> updates the age
  - UpdateEvent -> creates a new event and adds to Person
- READ OPERATION
  - Sets the current object to a random Person
- DELETE OPERATION
  - Not implemented
- FLUSH OPERATION
  - Re-attaches Session and writes modified POJO to DB

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Source Code

DETACHED MODE

```
 Person person = (Person) session.load(Person.class, (long) 1);
        person.getAge();
        session.getTransaction().commit();
        HibernateUtil.getSessionFactory().close();

for (int i = 0; i < TRANSACTIONS; i++) {
    person.setAge((int) i % 100);
}
 // Flush the changes
session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
session.saveOrUpdate(person);
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

HIBERNATE LEVEL2 CACHE MODE

```
for (int i = 0; i < TRANSACTIONS; i++) {
            session = HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            person = (Person) session.load(Person.class, (long) 1);
            // update the person's age to a "random" number between 0 and 99
            person.setAge((int) i % 100);

            session.getTransaction().commit();
            if (i % 1000 == 0) { System.out.print("."); System.out.flush(); }
}
```

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Performance Tests

- ReadAgeHibernate
  - 25k iterations
    - Reads a Person object, reads the age, commits
  - Run with and without 2nd level cache
- UpdateAgeHibernate
  - 25k iterations
    - Reads a Person object, updates the age, commits
  - Run with and without 2nd level cache
- ReadAgeTC
  - Reads a Person object
  - Sets person object into Terracotta clustered graph
  - 25k iterations
    - Reads the age
- UpdateAgeTC
  - Reads a Person object
  - Sets person object into Terracotta clustered graph
  - 25k iterations
    - Updates the age
  - Commits

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Results: Hibernate vs. Detached POJOs

Operation	Type	Results
Update	Hibernate	~ 1000 ops / sec
Update	Hibernate + 2nd Level Cache	~ 1800 ops / sec
Update	Terracotta	~ 7000 ops / sec
Operation	**Type**	**Results**
Read	Hibernate	~ 1000 ops / sec
Read	Hibernate + 2nd Level Cache	~ 1800 ops / sec
Read	Terracotta	~ 500,000 ops / sec

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Case Studies

TERRACOTTA

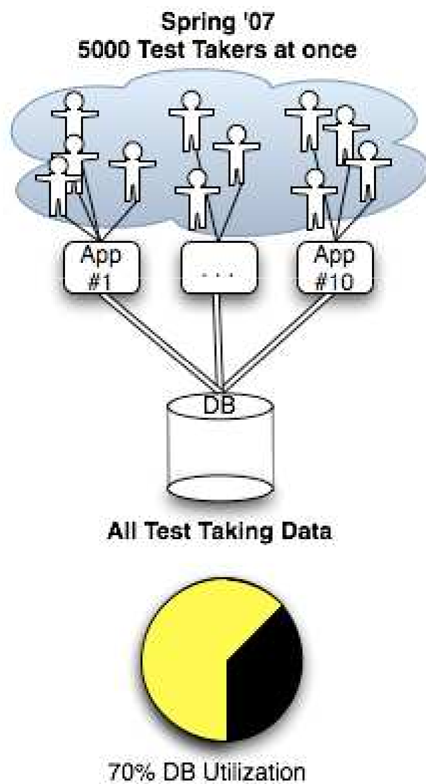
Confidential – for information of designated recipient only. Copyright Terracotta 2006

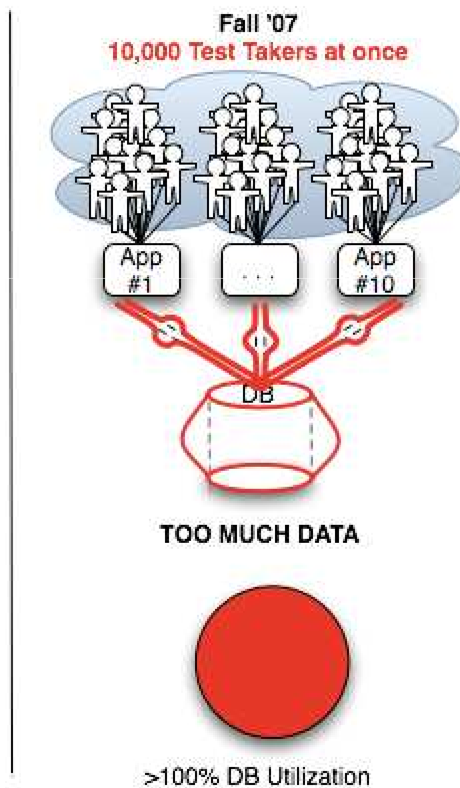
Comparison: non-partitioned vs. partitioned scale out

- Load Balanced Application
  - Publishing Company
  - Happy with availability and simplicity using Hibernate + Oracle
  - Not happy with scalability
  - SOLUTION: Hibernate disconnected mode
- Partitioned Application
  - Travel Company
  - Happy with MQ-based availability, 4 dependent apps mean no API changes allowed
  - System of Record too expensive to keep scaling
  - SOLUTION: Proxy the System or Record; Partition for scale

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Large Publisher Gets Caught Down the Path with Oracle





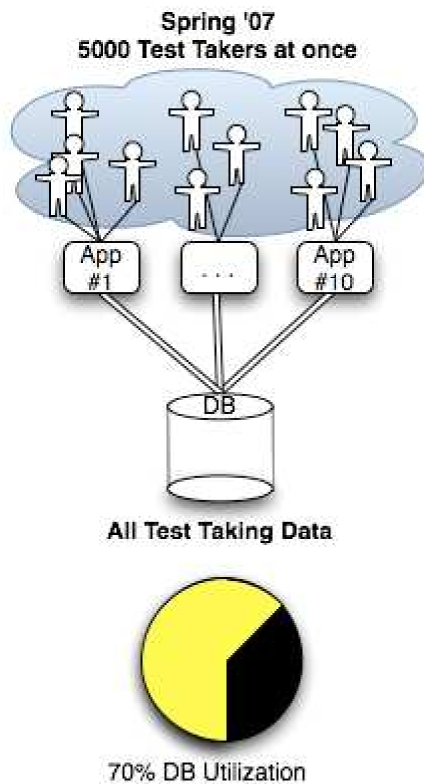
Scaling Out or Up?

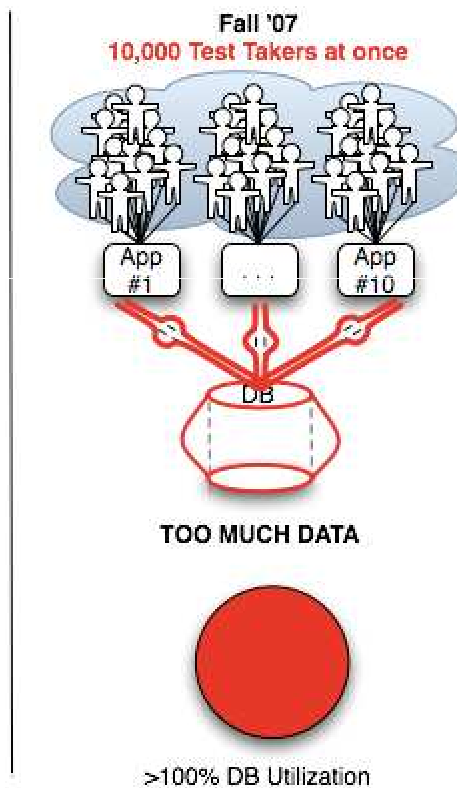
Confidential – for information of designated recipient only. Copyright Terracotta 2006

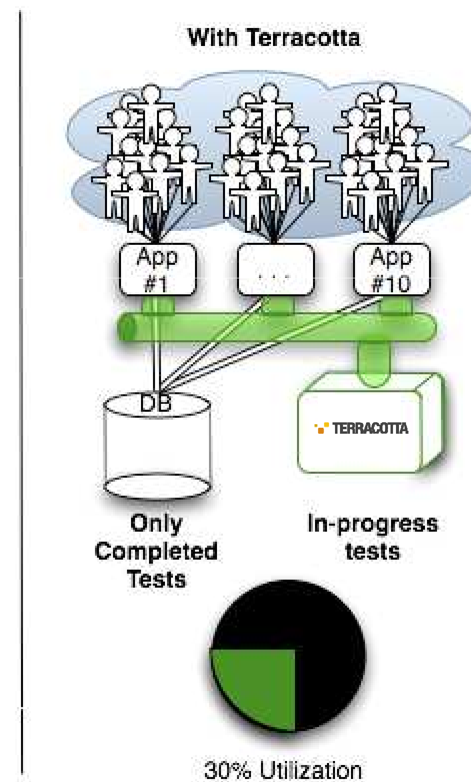
TERRACOTTA

Breaking the Pattern without Leaving "Load-Balanced" World

- $1.5 Million DB & HW savings
- Doubled business
- More than halved database load







Confidential – for information of designated recipient only. Copyright Terracotta 2006

User Was Happy

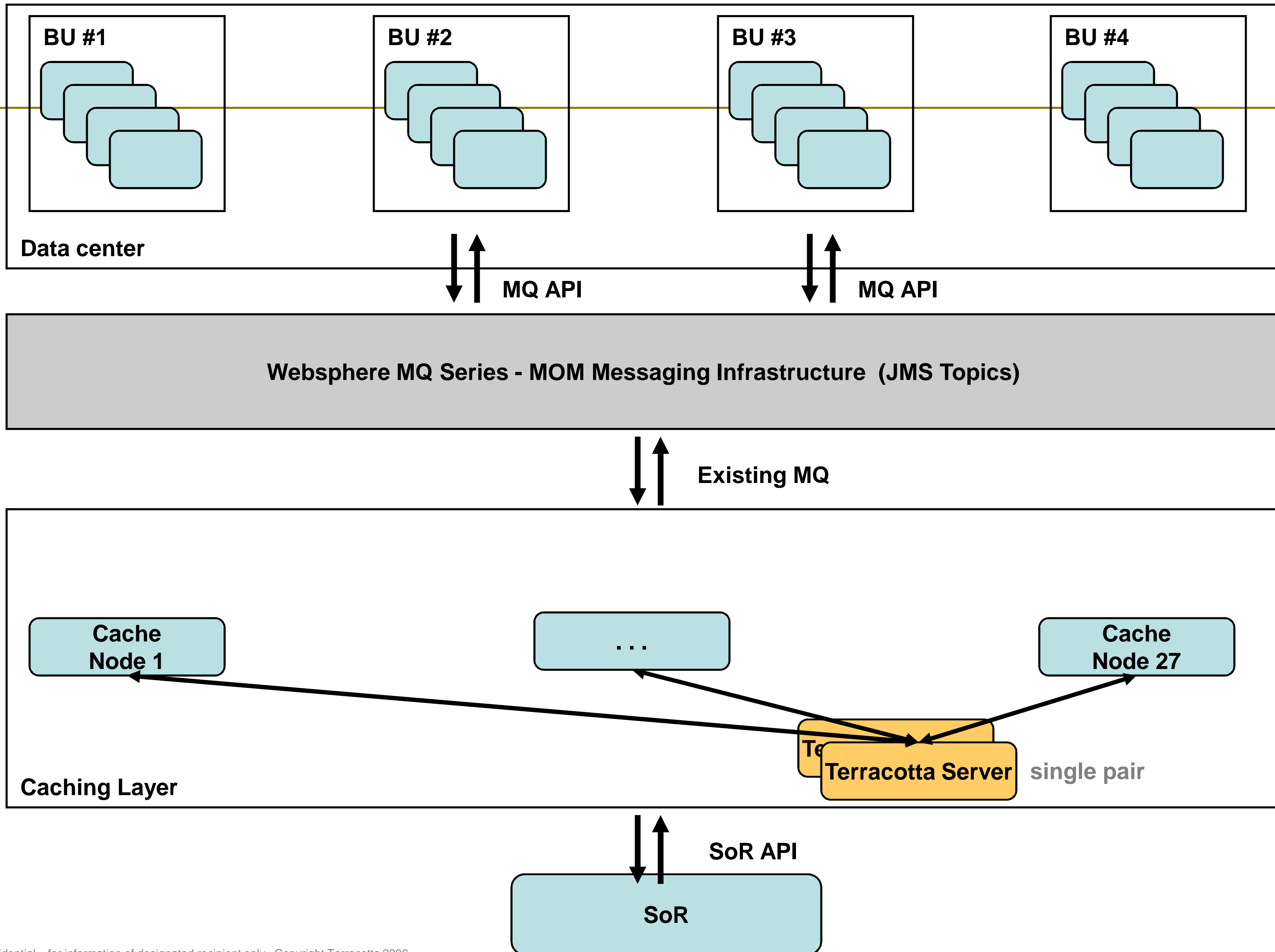
- Database was still the SoR which kept reporting and backup simple
- Scalability was increased by over 10X
- Availability was not compromised since test data was still on disk, but in memory-resident format instead of relational
- …simple scalability + availability

Confidential – for information of designated recipient only. Copyright Terracotta 2006

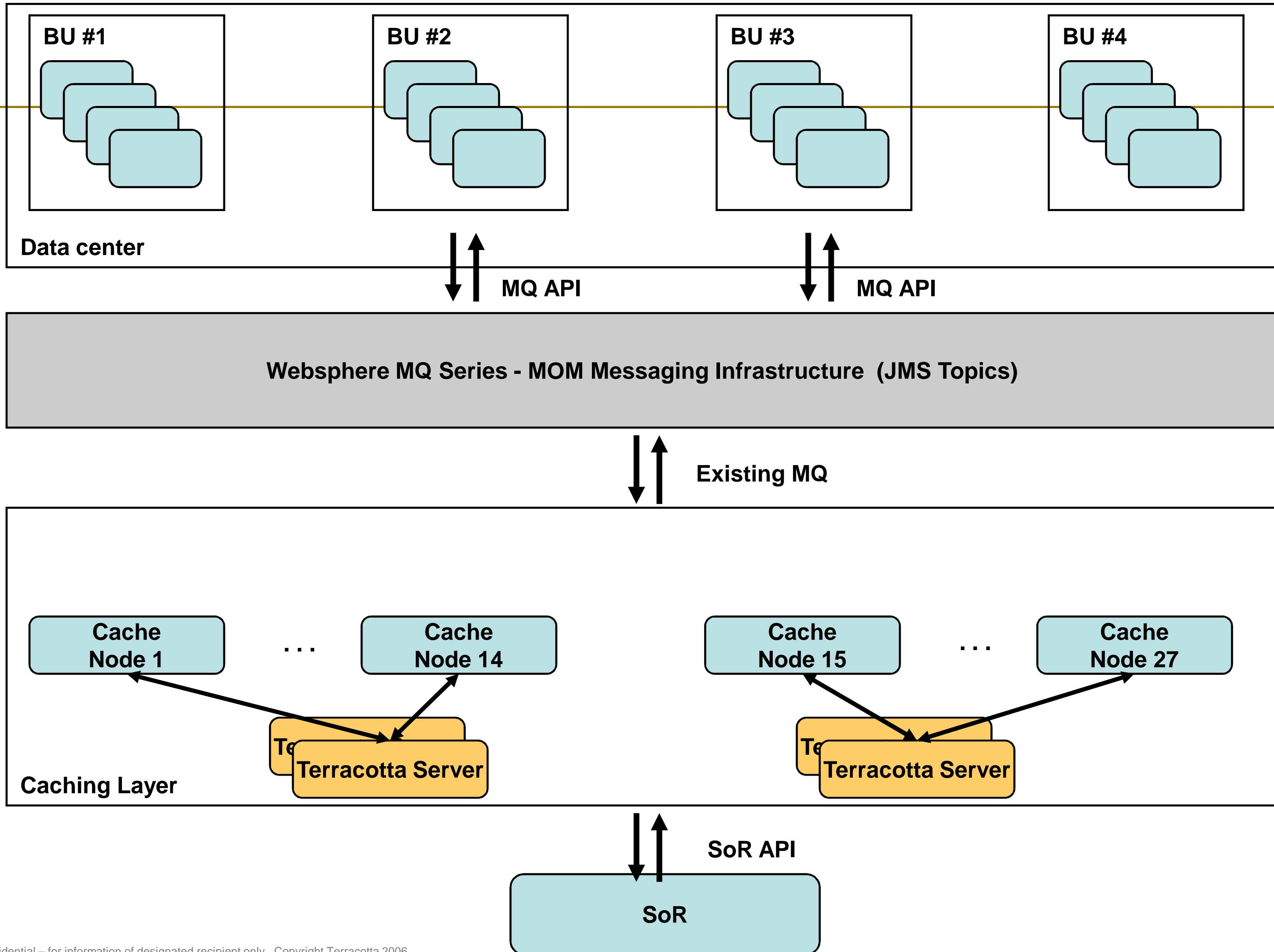
Example Caching Service

- Reduce utilization of System of Record
- Support 4 BUs
- 10K queries / second today
- Headroom for 40K queries / second
- (Note: all performance goals)

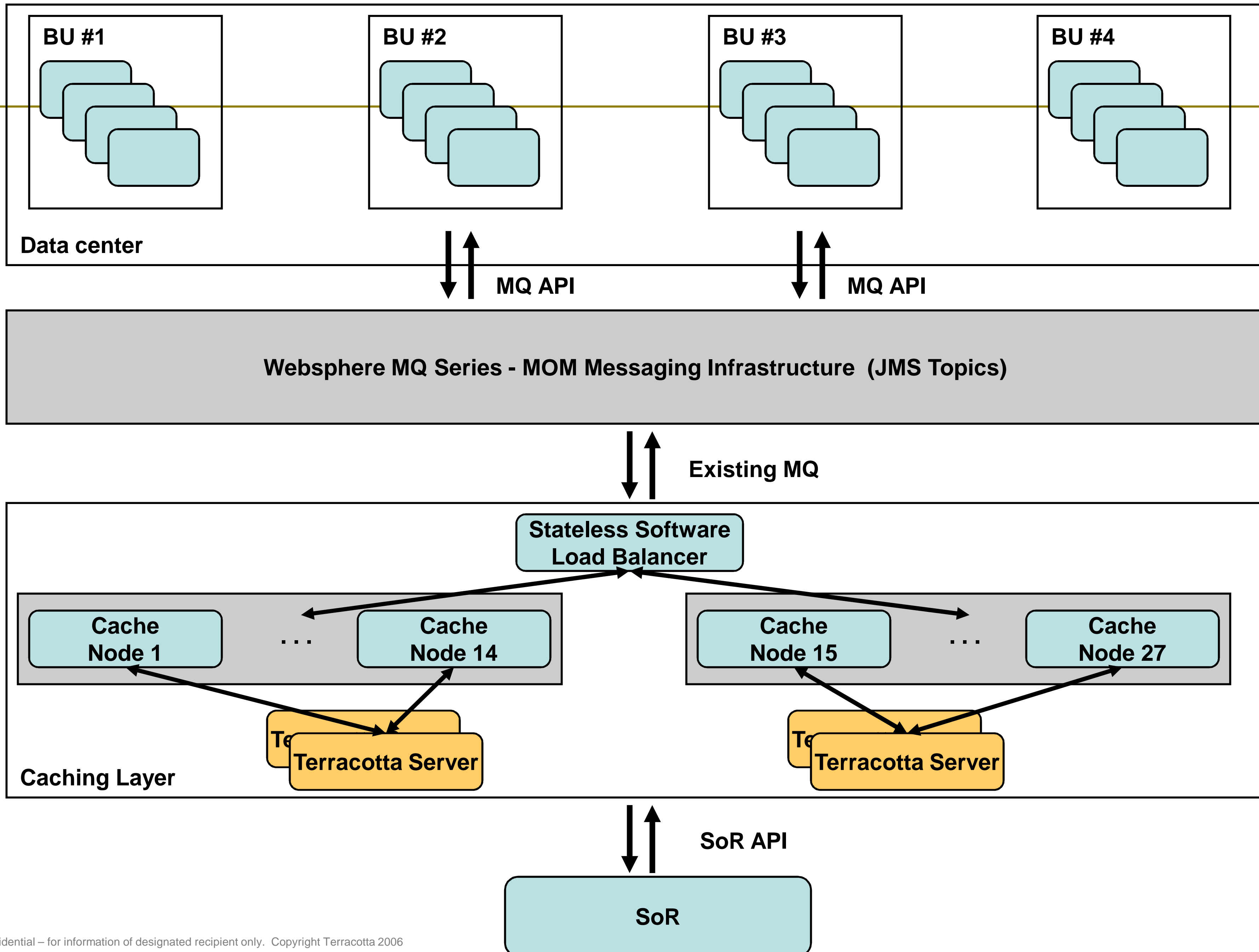
Confidential – for information of designated recipient only. Copyright Terracotta 2006



Confidential – for information of designated recipient only. Copyright Terracotta 2006



Confidential – for information of designated recipient only. Copyright Terracotta 2006



Confidential – for information of designated recipient only. Copyright Terracotta 2006

User Was Unhappy

- Simplicity was lost. The Partitioning leaked up the application stack
- Availability was no longer easy (failure had to partition as well)
- Scalability was the only “Scale Out Dimension” delivered

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Lessons Learned: Scalability + Availability + Simplicity

- Stop the Madness
  - Stop the hacking! Stop the clustering!
  - Start naïve and get more sophisticated on demand
- Balancing the 3 requires scalable, durable memory across JVM boundaries (spread out to scale out)
- **Simplicity** ⇒ Require no specific coding model and no hard-coded replication / persistence points
- **Scalability** ⇒ Read from Local Memory; write only the deltas in batches
- **Availability** ⇒ Write to external process + write to disk

Confidential – for information of designated recipient only. Copyright Terracotta 2006

JVM-level Clustering Addresses the Trade-offs

"ILITIES"

SIMPLE
- Honors Language Spec across JVM boundaries
- Transparent to source code
- Thread Coordination too

SCALABLE
- Virtual Heap
- Read from Heap
- Write deltas, only where needed

AVAILABLE
- Persist to disk at wire speed
- Active / Passive and Active / Active strategies

Scale Out Model

Models
- Load balanced stays naïve
- Partitioned stays POJO (abstractions are easy)

Models
- Load balanced scales through implicit locality
- Partitioned scales by avoiding data sharing

Models
- Load balanced apps made available by writing heap to disk
- Partitioned made available by using the cluster to store workflow

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Guidelines: NAM helps Load-Balanced Scale Out

- **Simplicity** ⇒ Ignore the impedance mismatch. Don't be afraid of the DB.
- **Scalability** ⇒ Just cache it! (EHCache, JBossCache, custom) Disconnect from the DB as often as you can
- **Availability** ⇒ Distributed caches can be made durable / reliable and shared with JVM-level clustering

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Guidelines: NAM helps Partitioned Scale Out

- **Simplicity** ⇒ Never simple…but SEDA, MapReduce, master / worker, Scala, all help
- **Scalability** ⇒ Share the data not the control flow to optimize locality
- **Availability** ⇒ Guarantee either or both the events and the data cannot be lost

- Honestly. Punt on partitioning if you can. Most people who need it will know, based on the use case outrunning disk, network, CPU, etc.
  - Example: Pushing more than 1GBit on a single node where multiple nodes could each push 1GBit

Confidential – for information of designated recipient only. Copyright Terracotta 2006

Thank You

- Learn more at http://www.terracotta.org/

Confidential – for information of designated recipient only. Copyright Terracotta 2006