Clustered Architecture Patterns: Examinator

Ari Zilka – Terracotta

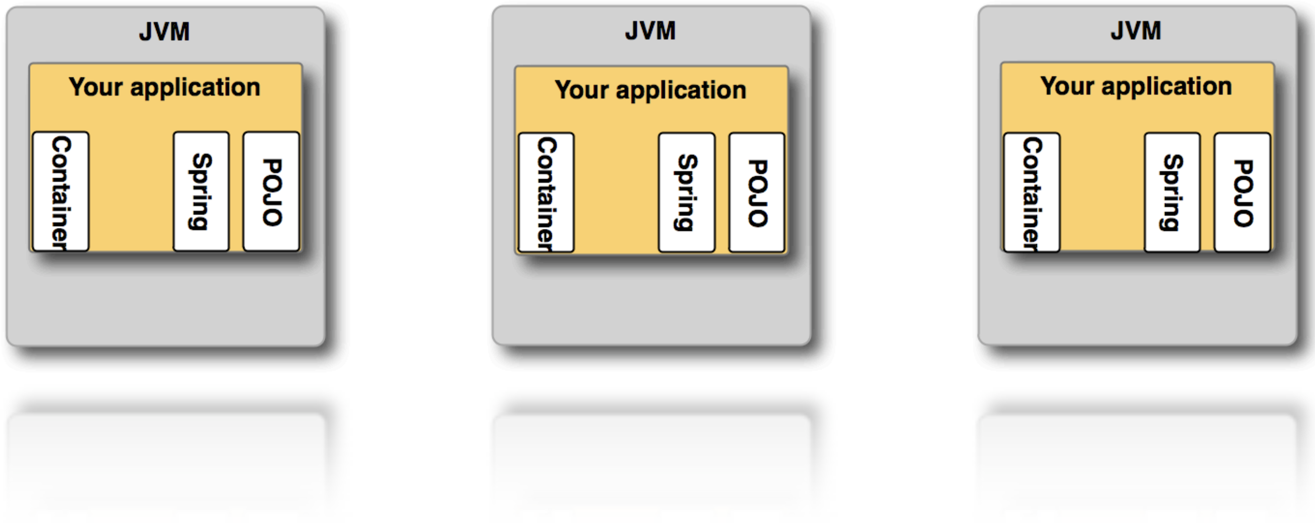
A bit of housekeeping...

- How many have heard of terracotta?
- Terracotta is open source
- It is designed to make Java apps less expensive
  - smaller database servers
  - no expensive J2EE app servers
  - and no custom code
  - all at the same time
- Terracotta servers manage your application data
  - cluster in arrays that stripe and mirror
- Terracotta does not require (or support) SAN
- We are not asking you to program to threads

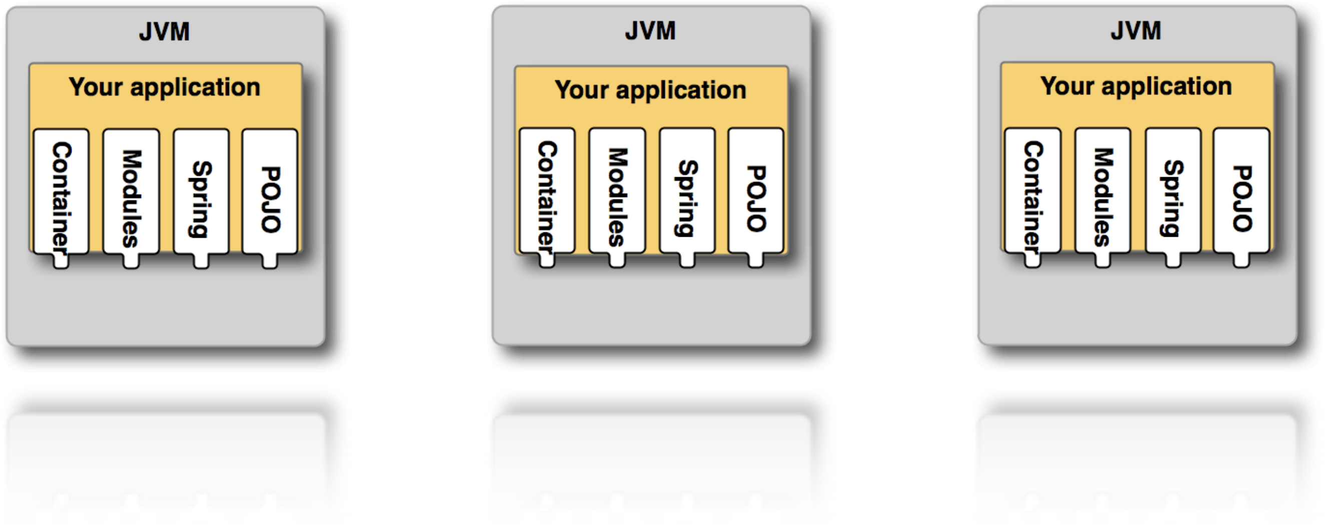
How our community uses Terracotta

1. Remove ORM/DB for a certain class of data
2. Cache DB to avoid expensive scale-up
3. Stop using JMS for replication
4. Simple/flexible messaging
5. Replace multiple caching and messaging tools
6. HTTP Sessions clustering that works
7. Scale a one node app to more without code rewrite
8. Get HA, low-latency apps on commodity hardware

Starting with your existing Java app

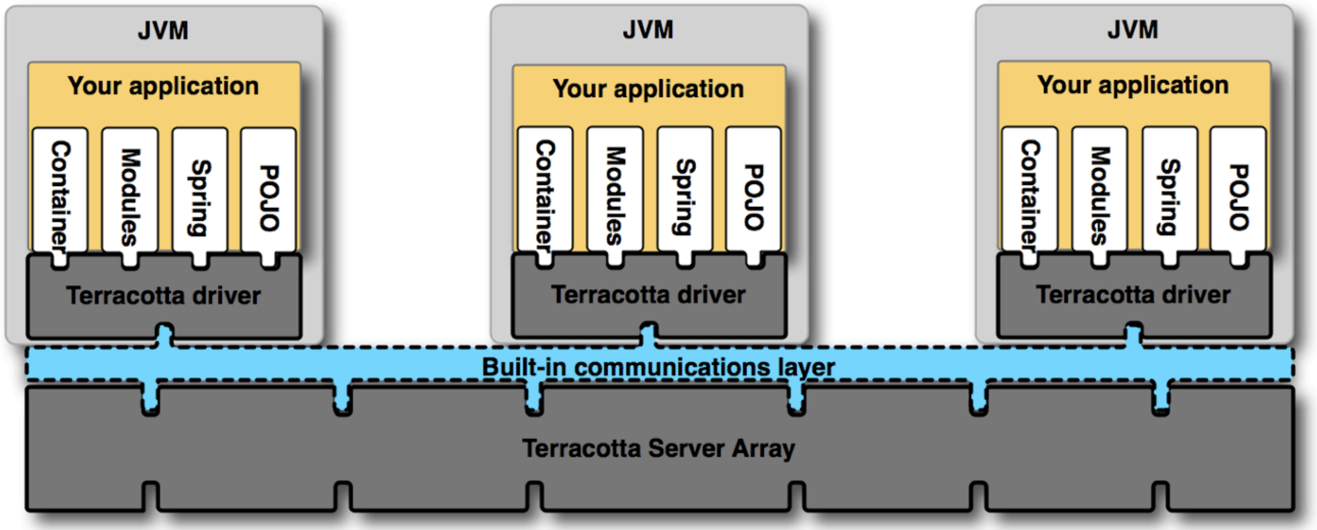


Easily integrate by using Terracotta Integration Modules (TIMs)



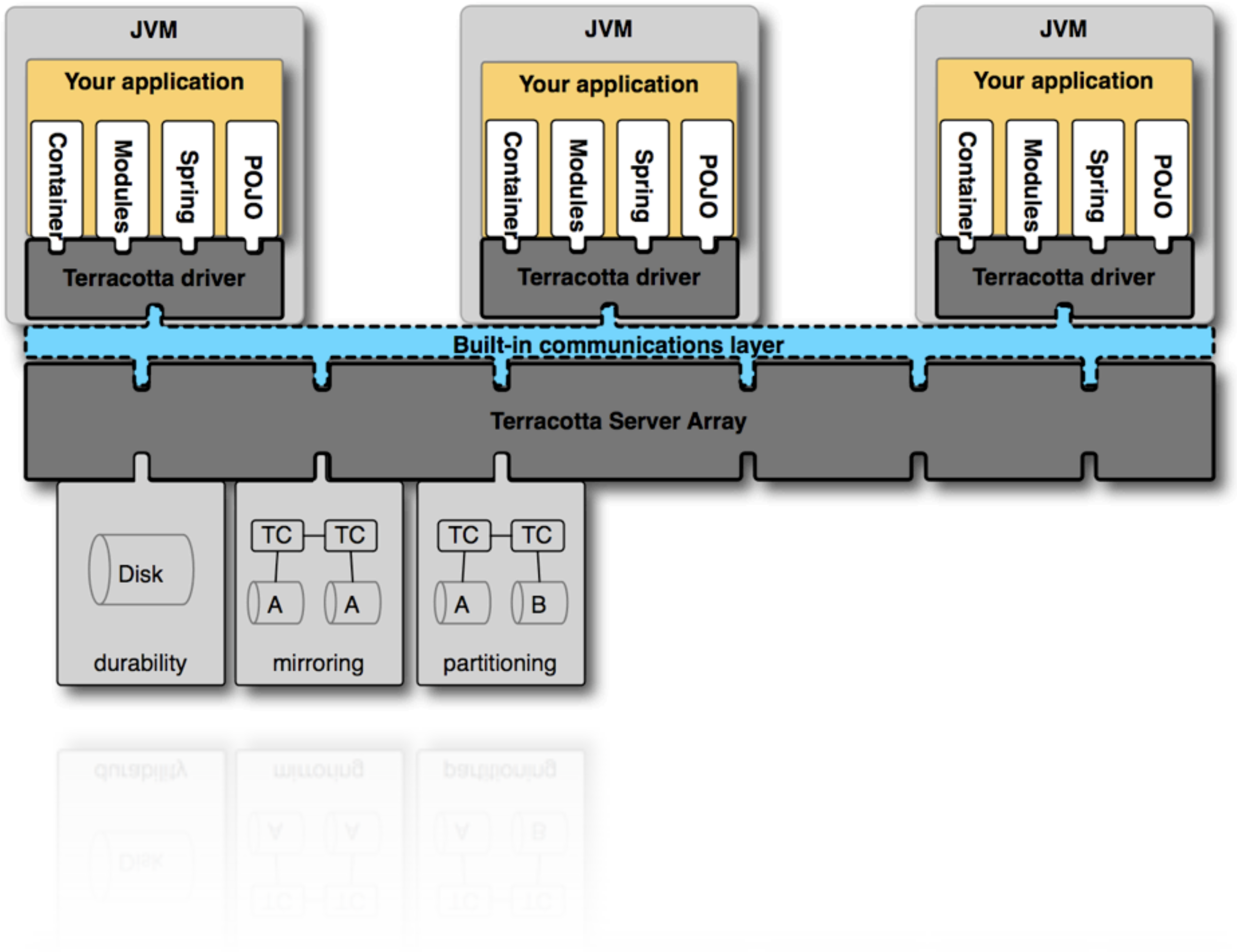
- Container of your choice
- Spring in the box
- Popular Frameworks
- Pluggable solutions
  - WAN, Write-behind, M/W, etc.
- Write your own

Very high performance, low latency scale out

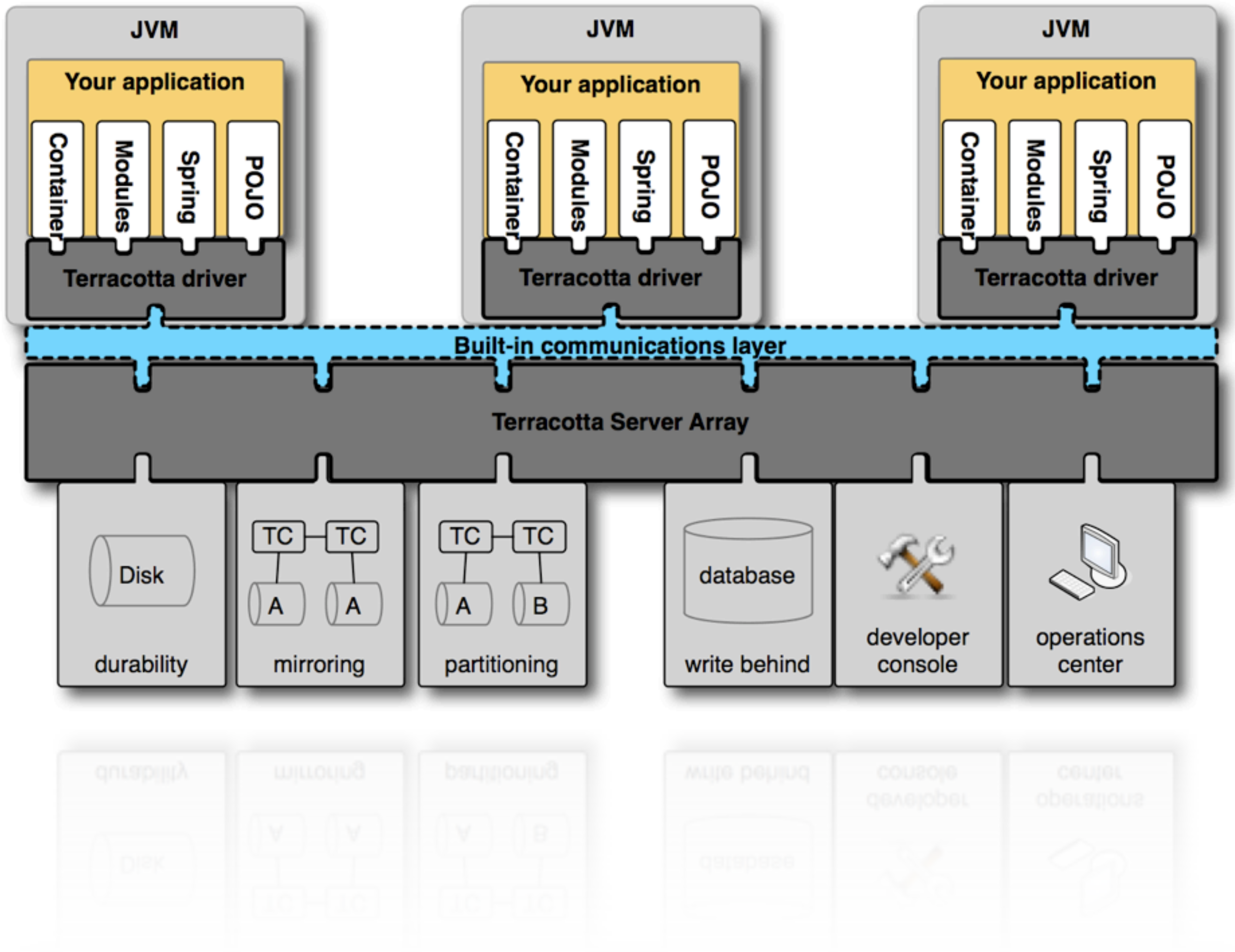


- Data locality optimized
- Efficient cluster-wide coordination

Built-in scalability and availability



Tools help avoid tuning, debugging, downtime



Tools in more detail

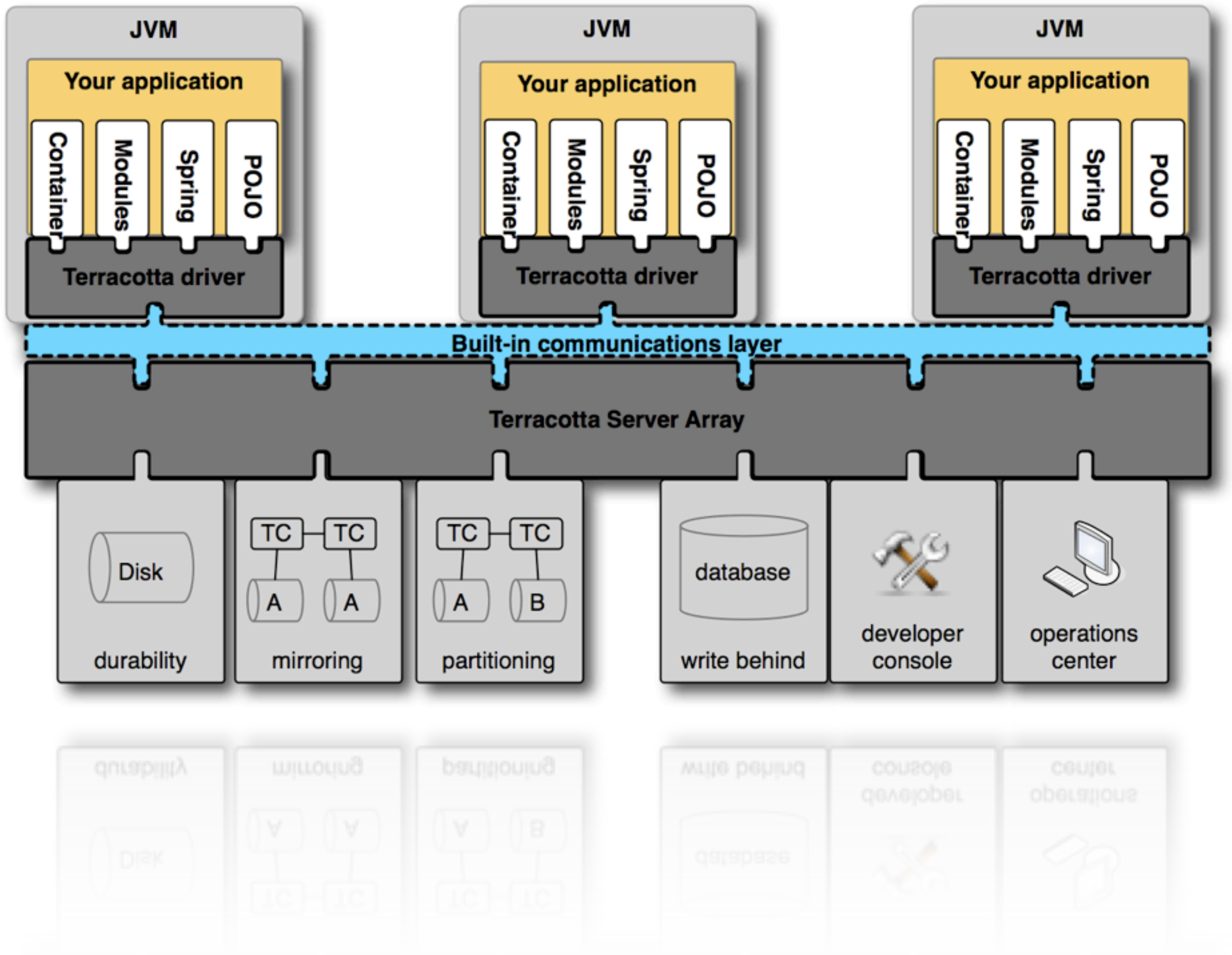
For the developer...

- Developer console
  - Visualization
  - Lock profiling
  - Tuning & debugging
- Eclipse plug-in
- Maven integration

For production...

- Push button app management
- Runtime statistics monitoring
- Rolling upgrades
- Backup / Restore
- Root cause analysis

Comprehensive Data Mgmt Solution



High performance + high scale + POJO = simple

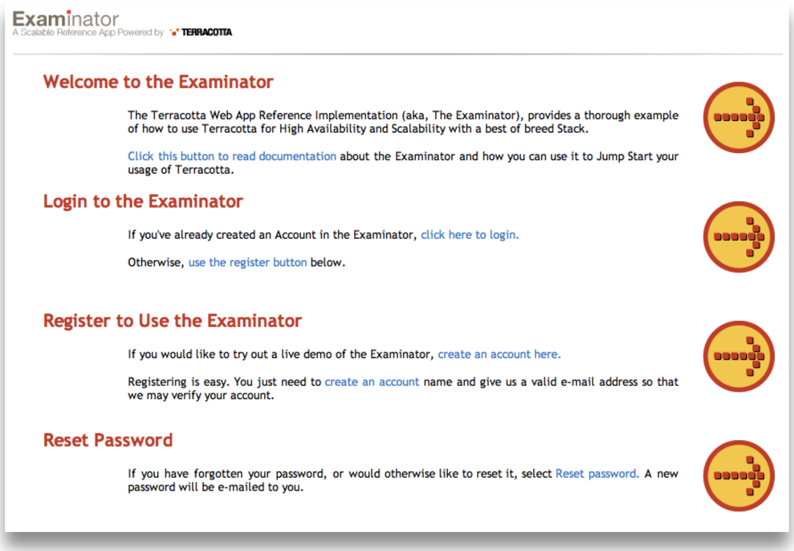
- 10X throughput over conventional APIs
  - All Reads from memory (implicit locality)
  - All Writes are deltas-only
  - Statistics and heuristics (greedy locks)
- Terracotta Server Array scales to 50K+ tps (w/o partitioning)
- Looks like Java to me (code like your mom used to make)
  - Normal access patterns: no check-out before view and check-in on commit
  - Code and test at a unit level without infrastructure intruding on app logic
  - Threads on multiple JVMs look like threads on the same JVM

TERRACOTTA

But how do I get these benefits?

- This is where Examinator comes in

Best Practices – Examinator


Examinator
A Scalable Reference App Powered by TERRACOTTA

Welcome to the Examinator

The Terracotta Web App Reference Implementation (aka, The Examinator), provides a thorough example of how to use Terracotta for High Availability and Scalability with a best of breed Stack.

Click this button to read documentation about the Examinator and how you can use it to Jump Start your usage of Terracotta.

Login to the Examinator

If you've already created an Account in the Examinator, click here to login.

Otherwise, use the register button below.

Register to Use the Examinator

If you would like to try out a live demo of the Examinator, create an account here.

Registering is easy. You just need to create an account name and give us a valid e-mail address so that we may verify your account.

Reset Password

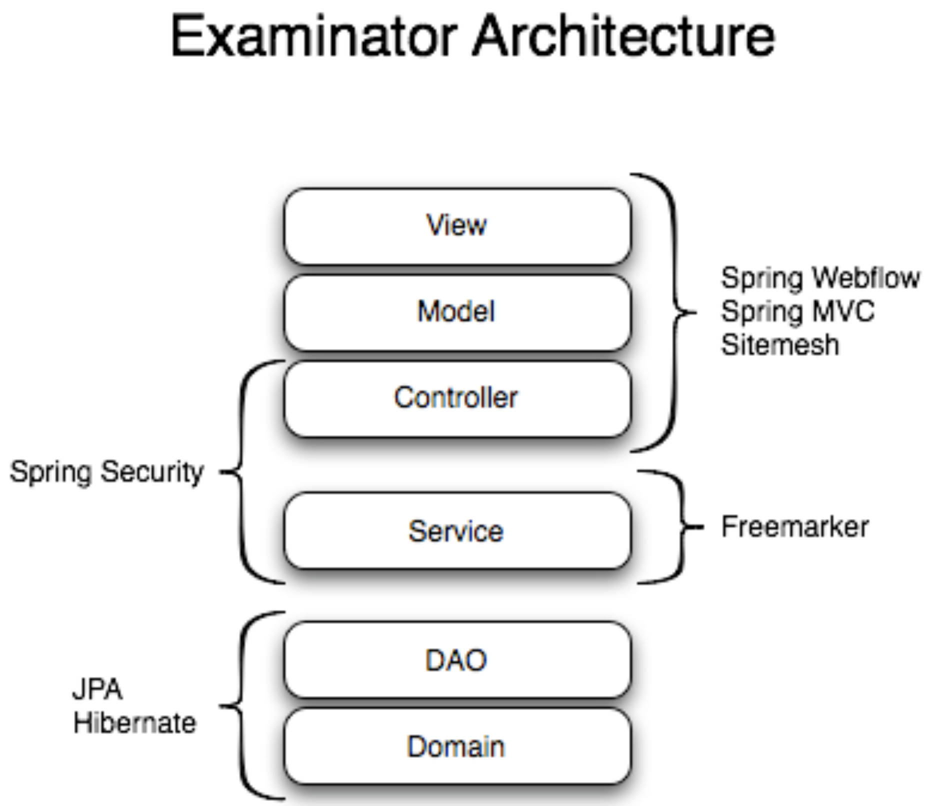
If you have forgotten your password, or would otherwise like to reset it, select Reset password. A new password will be e-mailed to you.

- Best of breed OSS stack
  - Tomcat, Spring, Hibernate, EHCache, ServiceMix
- 16 nodes
- 20k concurrent users
- 5ms average response time

Examinator Use Cases

Business Function	Terracotta Usage	Alternative	Benefit
Account creation	Incomplete user validations held in-memory	State machine in db	Db scales with actual users, not potential ones
User authc	Spring Security state clustered inside HTTP Session	No session clustering	Failover between app servers does not impact user experience
In progress Exams	Only ADMINs can see all exams	Store roles in DB	DB offload
Flush to DB	Asynch write-behind to DB	Custom code	DB offload / smaller DB

Pattern: Conversation Clustering



Set a conversation key (cookie)

Value should be object graph

- not primitives like string or byte array

Load Balancer required

- Sticky load balancing (layer 7 preferred, or Apache mod_jk)
- Ensures locality of reference

No DB required

Ref. impl. code available now:
http://svn.terracotta.org/svn/forge/projects/exam/

TERRACOTTA

AuthC/AuthZ

- Spring Security TIM
  - Authentication (AbstractAuthenticationToken) and authorization (GrantedAuthorityImpl) tokens held in HTTP Session via Spring Security

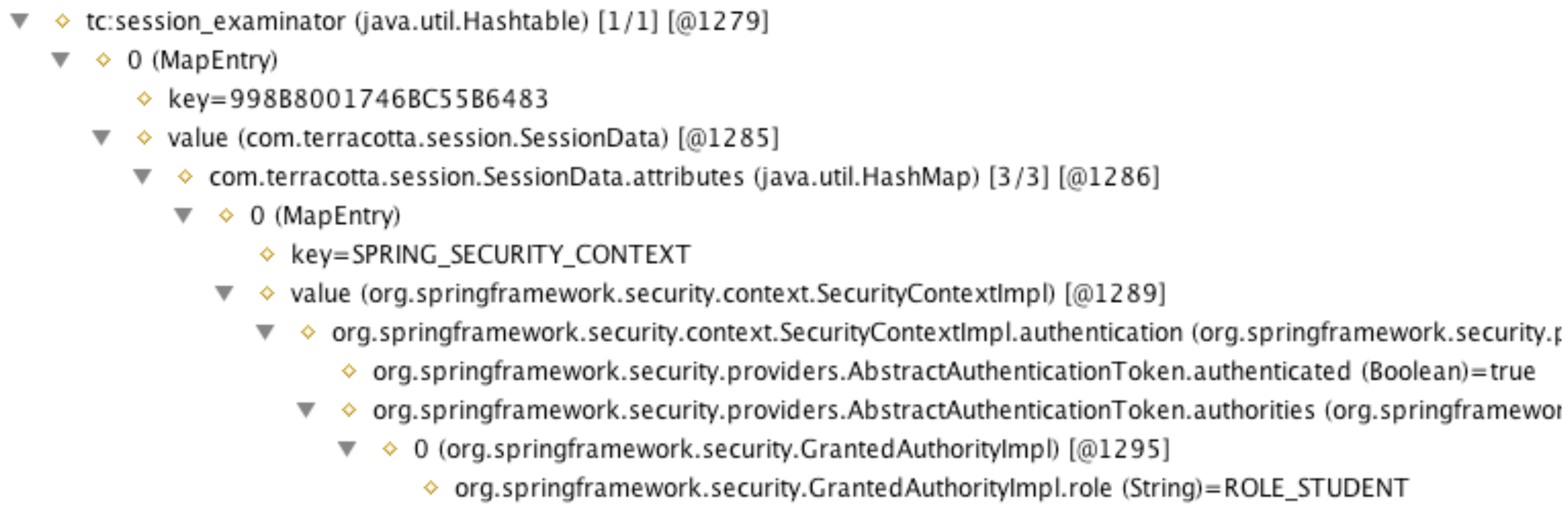
```
<filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

  - Clustered via Terracotta session clustering module
  - Spring Security TIM handles all instrumentation configuration for objects in HTTP session

- LoginController

TERRACOTTA

AuthC/AuthZ: Spring Security



```
▼ ◇ tc:session_examinator (java.util.Hashtable) [1/1] [@1279]
  ▼ ◇ 0 (MapEntry)
      ◇ key=998B8001746BC55B6483
    ▼ ◇ value (com.terracotta.session.SessionData) [@1285]
      ▼ ◇ com.terracotta.session.SessionData.attributes (java.util.HashMap) [3/3] [@1286]
        ▼ ◇ 0 (MapEntry)
            ◇ key=SPRING_SECURITY_CONTEXT
          ▼ ◇ value (org.springframework.security.context.SecurityContextImpl) [@1289]
            ▼ ◇ org.springframework.security.context.SecurityContextImpl.authentication (org.springframework.security.p
                ◇ org.springframework.security.providers.AbstractAuthenticationToken.authenticated (Boolean)=true
              ▼ ◇ org.springframework.security.providers.AbstractAuthenticationToken.authorities (org.springframewor
                ▼ ◇ 0 (org.springframework.security.GrantedAuthorityImpl) [@1295]
                    ◇ org.springframework.security.GrantedAuthorityImpl.role (String)=ROLE_STUDENT
```

TERRACOTTA

Conversation State: Take The Exam

- Spring WebFlow
  - Workflow through examination process
- ExamService (CachingWrapperExamService)
  - Caches exam meta-data (questions, sections, etc.)
- ExamSessionService
  - Manages the state of in-progress examinations using clustered:
  - ConcurrentHashMap

TERRACOTTA

Conversational State: List Available Exams

```
<view-state id="chooseExam" view="exam/list">
  <on-render>
    <evaluate expression="examService.getAllExams()" result="viewScope.exams" />
  </on-render>
  <transition on="selectExam" to="examSelected">
    <set name="flowScope.examId" value="requestParameters.examId"></set>
  </transition>
  <transition on="return" to="return" />
</view-state>
```

TERRACOTTA

Conversational State: View Exam Details

```
<view-state id="examSelected" view="exam/details">
  <on-render>
    <evaluate expression="examService.findById(flowScope.examId)" result="viewScope.exam"/>
  </on-render>
  <transition on="back" to="chooseExam" />
  <transition on="return" to="return" />
  <transition on="startExam" to="startExam" />
</view-state>
```

```
public class CachingWrapperExamService implements ExamService {
  /* as defined in ehcache.xml */
  private static final String EXAM_CACHE_NAME = "examCache";

  private final CacheManager cacheManager;
  private final Ehcache cache;
  private final ExamService examService;

  public CachingWrapperExamService(ExamService examService){
    this.examService = examService;
    this.cacheManager = CacheManager.getInstance();
    this.cache = this.cacheManager.getEhcache(EXAM_CACHE_NAME);
  }
  // ...
 public Exam findById(Long id) {
    Exam exam = getCached(id);
    if (exam == null){
      exam = cache(this.examService.findById(id));
    }
    return exam;
  }
  // ...
}
```

TERRACOTTA

Conversation State: Start Exam

```
public class ExamSessionServiceImpl implements ExamSessionService {

  // a map storing userName -> examSession mapping of currently active exams
  @Root
  private final ConcurrentHashMap<String, ExamSession> ongoingExams          = new ConcurrentHashMap<String, ExamSess


  private final ScheduledExecutorService                  examTimeoutExecutor   = Executors.newScheduledThreadPool(1);
  private final Map<String, Future>                       scheduledTimeOutTasks = new HashMap<String, Future>();
  private final ExamService                               examService;
  private final UserService                               userService;
  private final QuestionComparator                        questionComparator    = new QuestionComparator();

  @Autowired
  public ExamSessionServiceImpl(final ExamService examService, final UserService userService) {
    this.examService = examService;
    this.userService = userService;
  }

  // ...
}
```

TERRACOTTA

Conversation State: Start Exam (cont.)

```
public class ExamSessionServiceImpl implements ExamSessionService {
  // ...
public ExamSession startExam(final String userName, final Long examId) throws ExamException {
    final User user = userService.findByUserName(userName);
    final Exam exam = examService.findById(examId);
    if (null == exam) { return null; }
    // initialise the exam facade
    exam.getExamFacade();
    final ExamSession session = new ExamSession(exam.getTimeLimitInMinutes());
    session.setUserId(user.getId());
    session.setExamId(exam.getId());
    final ExamSession prev = ongoingExams.putIfAbsent(userName, session);
    if (prev != null && prev.getExamId().longValue() != examId.longValue()) {
      throw new ExamAlreadyInProgressException(userName,
                                        examService.findById(prev.getExamId()),
                                        "Exam already in progress for user: " + userName);
    }
    // return previously ongoing exam if attempted to start same exam again
    if (prev != null) return prev;

    // run the timeout after 2 secs from the actual time so that exam time out happens on the client first
    // and give the user a chance to get ExamTimedOutException when exam times out on client
    final Future timeoutTask = examTimeoutExecutor.schedule(new ExamTimeoutTask(this, userName),
                                                 session.getRemainingTimeInSeconds() + 2, TimeUnit.SECONDS
    scheduledTimeOutTasks.put(userName, timeoutTask);
    return session;
  }

  // ...
}
```

Conversation State: Take Exam

- Spring Web Flow state clustered via HTTP Session clustering
  - Spring Web Flow TIM
- ExamSession clustered via custom POJOs.

TERRACOTTA

Conversation State: Take Exam: ExamSessionService

```
public class ExamSessionServiceImpl implements ExamSessionService {

  // a map storing userName -> examSession mapping of currently active exams

  @Root

  private final ConcurrentHashMap<String, ExamSession> ongoingExams  = new ConcurrentHashMap<String,
     ExamSession>(20000, 0.75f, 512);

  // …

  public void evaluateExamQuestionForm(final String userName, final ExamQuestionForm examQuestionForm)

      throws ExamException {

    final ExamSession examSession = getExamSession(userName);

    examSession.addUserQuestionChoiceId(examQuestionForm.getQuestion().getId(),

                                                            examQuestionForm.getUserChoiceId());

    examSession.markQuestionForReview(examQuestionForm.getQuestion(),

                                                            examQuestionForm.isMarkQuestionForReview());

  }
```

Continued next slide…

Conversation State: Take Exam: ExamSessionService

```
public ExamSession getExamSession(final String userName) throws ExamException {

   ExamSession examSession = null;

   examSession = ongoingExams.get(userName);

   if (examSession == null) throw new ExamNotInProgressException(userName, "No Exam In Progress for
    user:" + userName);

   if (examSession.getRemainingTimeInSeconds() <= 0) {

     ongoingExams.remove(userName);

     final ExamResult result = getExamResult(examSession);

     examService.saveExamResult(result);

     throw new ExamTimedOutException("Exam timed out", examService.findById(examSession.getExamId()),
    result);

   }

   return examSession;

 }


 // ...

}
```

TERRACOTTA

ConversationState: Take Exam: ExamSession

```
@InstrumentedClass
public class ExamSession implements Serializable {
  private Long                        userId;
  private final Date                  startTime;
  private Long                        examId;
  private final int                   examTimeLimitInMinutes;

  // mapping of questionId -> choiceId, solutions submitted by user
  private final Map<Long, Long>       userQuestionChoiceMapping = new HashMap<Long, Long>();
  // questions marked for review, keeping id instead of actual question to save some bits in serialization
  private final Set<Long>             questionsMarkedForReview  = new HashSet<Long>();
  // ordered choices for questions mapping; keeps the ordering with choices id, this field will be serialized
  // key is questionId, value is a List of choiceIds
  private final Map<Long, List<Long>> questionChoicesOrder      = new HashMap<Long, List<Long>>();

  //this field is used in ongoing.jsp to display the name of the exams
  private transient String            examName;
  //this field is used in ongoing.jsp to display the name of the user taking the exam
  private transient String            userName;
```

ConversationState: Take Exam: ExamSession

```
@InstrumentedClass
public class ExamSession implements Serializable {
  // ...
  public void addUserQuestionChoiceId(final Long questionId, final Long choiceId) {
    if (null == questionId || null == choiceId) return;
    userQuestionChoiceMapping.put(questionId, choiceId);
  }
  public void markQuestionForReview(final Question question, final boolean mark) {
    if (question == null) return;
    if (mark) questionsMarkedForReview.add(question.getId());
    else questionsMarkedForReview.remove(question.getId());
  }

}
```

TERRACOTTA

Conversation State: Take Exam: ExamSession

- ExamSession objects are normal objects on the local heap
- Object state is persistent
- ExamSession objects are only on heap where needed
- Fine-grained changes sent only where needed
- Simple locking: HTTP Session lock is all you need

Coherent View Of Global Data: View Ongoing Exams

Administrators have a global view of all ongoing exams

```
@Controller
@RequestMapping("/exam/ongoing.do")
@RolesAllowed( { StandardAuthoritiesService.ADMINISTRATOR })
public class OngoingExamsController {

  private final ExamSessionService service;

  @Autowired
  public OngoingExamsController(final ExamSessionService examSessionService) {
    this.service = examSessionService;
  }

  @RequestMapping(method = RequestMethod.GET)
  public ModelAndView listOngoingExams() {
    final ModelAndView result = new ModelAndView("exam/ongoing");
    result.addObject("examSessions", service.getOngoingExams());
    return result;
  }

  protected OngoingExamsController() {
    // protected default constructor is needed for CGLib AOPx
    service = null;
  }
}
```

TERRACOTTA

Coherent View Of Global Data: View Ongoing Exams

```
public class ExamSessionServiceImpl implements ExamSessionService {

  // a map storing userName -> examSession mapping of currently active exams
  @Root
  private final ConcurrentHashMap<String, ExamSession> ongoingExams         = new ConcurrentHashMap<String, ExamSess



  private final ScheduledExecutorService                 examTimeoutExecutor   = Executors.newScheduledThreadPool(1);
  private final Map<String, Future>                      scheduledTimeOutTasks = new HashMap<String, Future>();
  private final ExamService                              examService;
  private final UserService                              userService;
  private final QuestionComparator                       questionComparator    = new QuestionComparator();  // ...

  public Collection<ExamSession> getOngoingExams() {
    Collection<ExamSession> values = null;
    values = new ArrayList(ongoingExams.values());
    return values;
  }
  // ...

}
```

TERRACOTTA

Coherent View Of Global Data: View Ongoing Exams

```
public class ExamSessionServiceImpl implements ExamSessionService {

 public PageData<ExamSession> getOngoingExamsByPage(final PageRequest pageRequest) {
    List<ExamSession> values;
    PageRequest newPageRequest;
    final List<String> examSessionKeys = new ArrayList(ongoingExams.keySet());
    Collections.sort(examSessionKeys);
    int total = examSessionKeys.size();
    newPageRequest = PageRequest.adjustPageRequest(pageRequest, total);
    values = new ArrayList<ExamSession>(newPageRequest.getPageSize());

    int i = 1;
    boolean dirty = false;
    for (final Iterator<String> iter = examSessionKeys.iterator(); iter.hasNext(); i++) {
      String nextUser = iter.next();
      ExamSession nextSession = ongoingExams.get(nextUser);
      if (nextSession == null) {
        //continue if the session is no longer present in the map
        dirty = true;
        total--;
        continue;
      }
      if (i < newPageRequest.getStart()) {
        // move upto start point
        continue;
      }
      // break when reached the pageSize
      if (i >= newPageRequest.getStart() + newPageRequest.getPageSize()) break;
      values.add(nextSession);
    }
    if (dirty) newPageRequest = PageRequest.adjustPageRequest(pageRequest, total);
```

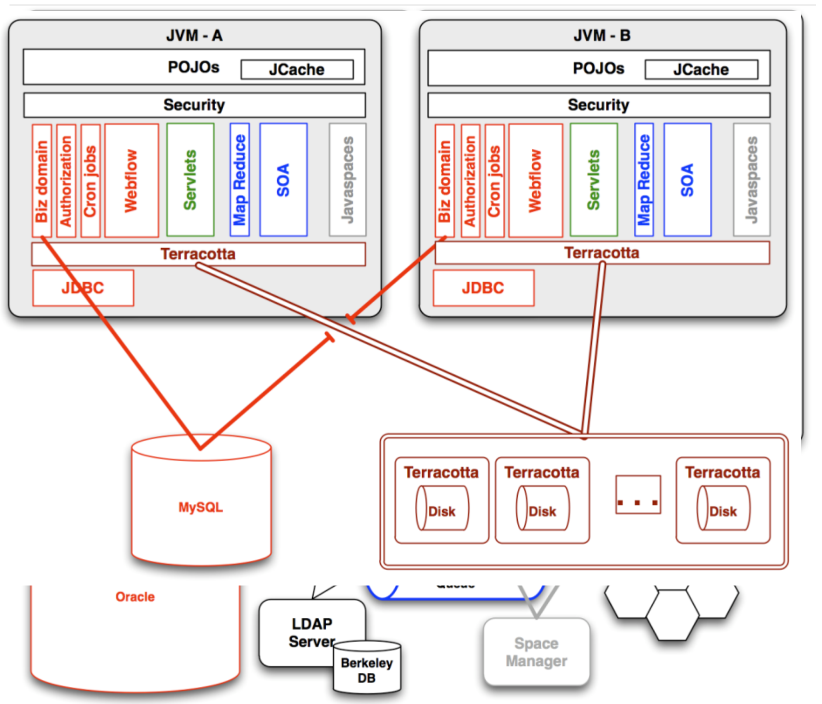
Conversation State And Coherent View Of Global Data

- Conversation State
  - Memory-speed access to flow data
  - Coherent cluster-wide
  - Database-like durability for failover…
  - Yet, no database abuse: less database use == less $$$
  - Handles failure of any/every node
  - Completely transparent: no put-back on change
- Coherent View Of Global Data
  - Coherent cluster-wide
  - No round-trip to database
    - Ergo, no database abuse (< $$$)
    - Ergo, no caching
    - Ergo, no cache freshness probs
  - Memory-speed read locks
  - Simple data model

Terracotta Gives You Your Brain Back

JEE: Complex code and infrastructure, high reliance on DBs

Terracotta: Lightweight code and infrastructure, less reliance on DBs

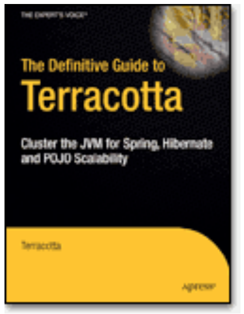


Summary

- Simplicity, scalability, and availability can be friends
  - Write normal Java code that works across JVMs
  - Use clustered architecture patterns and TIMs for popular frameworks
  - Simplicity saves $$$
- Don’t abuse the database
  - Leave business data in the database, use durable NAM for application state data
  - Less database utilization saves $$$
- Scale the JVM ⇒ Use Less Infrastructure
  - Terracotta lets you use best of breed components in a scalable and HA way
  - Throw out the maze of JMS, EJB, RMI, etc. and per-component scale
  - Reduce codebase by 30% ⇒ fewer bugs, < $$$
- Centralized operational control: manage your application cluster like you do your database
- Terracotta is open source
  - Free to use through production
  - Commercial versions, training, and services available

Resources

- Open Source (MPL-based) JVM-level clustering: http://www.terracotta.org
- Apress / Amazon.com: “Definitive Guide to Terracotta”
  - By Alex Miller, Ari Zilka, Geert Bevin, Jonas Bonér, Orion Letizi, Taylor Gautier



- Forums: http://forums.terracotta.org/
- Enterprise Offerings: http://www.terracottatech.com/