Best Practices Migrating from Spring to Java EE 6

Ray Ploski

Director, Developer Programs & Strategy

@rayploski

Who Is This Talk For?

- You have used Spring in the past and wonder how to move forward
- Java EE seems hot again, should you jump in?

Disclaimers

- No holy war
- No shootouts
- No silver bullets

Disclaimers

- No holy war
- No shootouts
- No silver bullets

Snowdrop

Why Migrate?

- Upgrading from older versions of Spring takes a lot of work anyway
- Spring is a proprietary framework
- Why not take it to the standard

This May Have Been Competitive in The Past

Now We Want This

Addressing Misunderstandings

Isn't Java EE too Fat?

- Startup time with an application deployed
- JBoss AS 7 ~ 2 seconds
- GlassFish v3 ~ 4 seconds
- Tomcat 6 + Spring ~ 4 seconds
- Java EE 6 WAR file < 100 kb

Isn't Java EE too Fat?

- Startup time with an application deployed
- JBoss AS 7 ~ 2 seconds
- GlassFish v3 ~ 4 seconds
- Tomcat 6 + Spring ~ 4 seconds
- Java EE 6 WAR file < 100 kb

"Nowhere in the Java EE spec does it say that Java EE servers should be heavyweight and slow"

Isn't Java EE Supposed to be Evil?

- Based on 2004 rhetoric
- No longer complex
- No longer extraneous code required

Ever hear of the fable The Tortoise and the Hare?

But I Need Dependency Injection

- Java EE introduced CDI
- More powerful contextual DI model
- Makes platform extensible in a standard way

But I Must Have AOP

Really???

But I Must Have AOP

Really???

Do you love getting your code asymmetrical and unmaintainable?

But I Must Have AOP

Really???

Do you love getting your code asymmetrical and unmaintainable?

Or are you just using AOP light a.k.a. Spring AOP a.k.a. (Java EE) Interceptors?

Can I Still Do Unit Testing?

It took a while for Java EE to get there but yes, there is **robust** support for testing

Capabilities Comparison

Capability	Spring	Java EE
Dependency Injection	Spring Container	CDI
Transactions	AOP/Annotations	EJB
Web Framework	SpringMVC	JSF
AOP	AspectJ (limited to Spring Beans)	Interceptors
Messaging	JMS	JMS / CDI
Data Access	JDBCTemplate / other ORM / JPA	JPA
RESTful Web Services	SpringWeb MVC (3.0)	JAX-RS
Integration Testing	Spring Test Framework	Arquillian**

Apparently

it can all be done using

plain vanilla lightweight

Java EE

```
rm -RF Spring*
```

Sure it would be fun,
but not very realistic

We need a step-by-step approach that moves us forward gradually

Our Scenario

- Old-school Spring application
- Lots of XML configuration, no annotations
- Old / outdated ORM solution
  - JDBCTemplate, Kodo, TopLink
- Deprecated extension-based MVC (SimpleFormController etc)

Migration Path

1.) Upgrade Spring Libraries

2.) Replace old frameworks (ORM, MVC, etc) within Spring

3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely

5.) Remove Spring container

Migration Path

1.) Upgrade Spring Libraries

2.) Replace old frameworks (ORM, MVC, etc) within Spring

3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely

5.) Remove Spring container

Upgrade Spring Version

- Upgrade Spring runtime (replace jar files)
- No code / configuration changes

Migration Path

1.) Upgrade Spring Libraries

2.) Replace old frameworks (ORM, MVC, etc) within Spring

3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely

5.) Remove Spring container

Replace old frameworks within Spring

Presentation layer

WebMVC

@Autowired

Business layer

Spring Beans

Tasks

@Autowired

@Autowired

Data & Integration layer

Kodo

Spring JDBCTemplates

JMS Beans

Replace old frameworks within Spring

Presentation layer

JSF

@Autowired

Business layer

Spring Beans

Don't touch specific Spring API's

Tasks

Data & Integration layer

@Autowired

@Autowired

JPA

Spring JDBCTemplates

JMS Beans

What About The Spring API's we don't want to change yet?

- JDBCTemplates
- REST Templates
- Task Scheduling
- Etc...

Migration Path

1.) Upgrade Spring Libraries

2.) Replace old frameworks (ORM, MVC, etc) within Spring

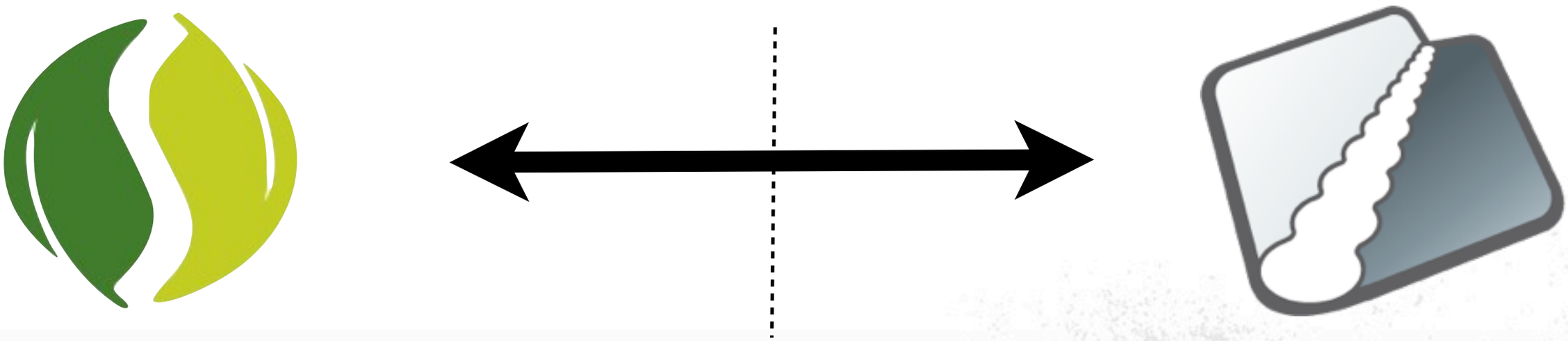
3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely

5.) Remove Spring container

Spring-CDI Bridge

- Use Java EE and Spring side by side
- Spring Beans produced as Managed CDI beans
- Seam 3 Spring Module



Spring Application

Servlet Container

MyApp.war

Spring Container

Spring Beans

Spring Beans

Spring Beans

TX Manager

AOP

ORM

Java EE Application

JBoss EAP 6

Java EE / CDI Container

MyApp.war

CDI Beans

Session Beans

CDI Beans

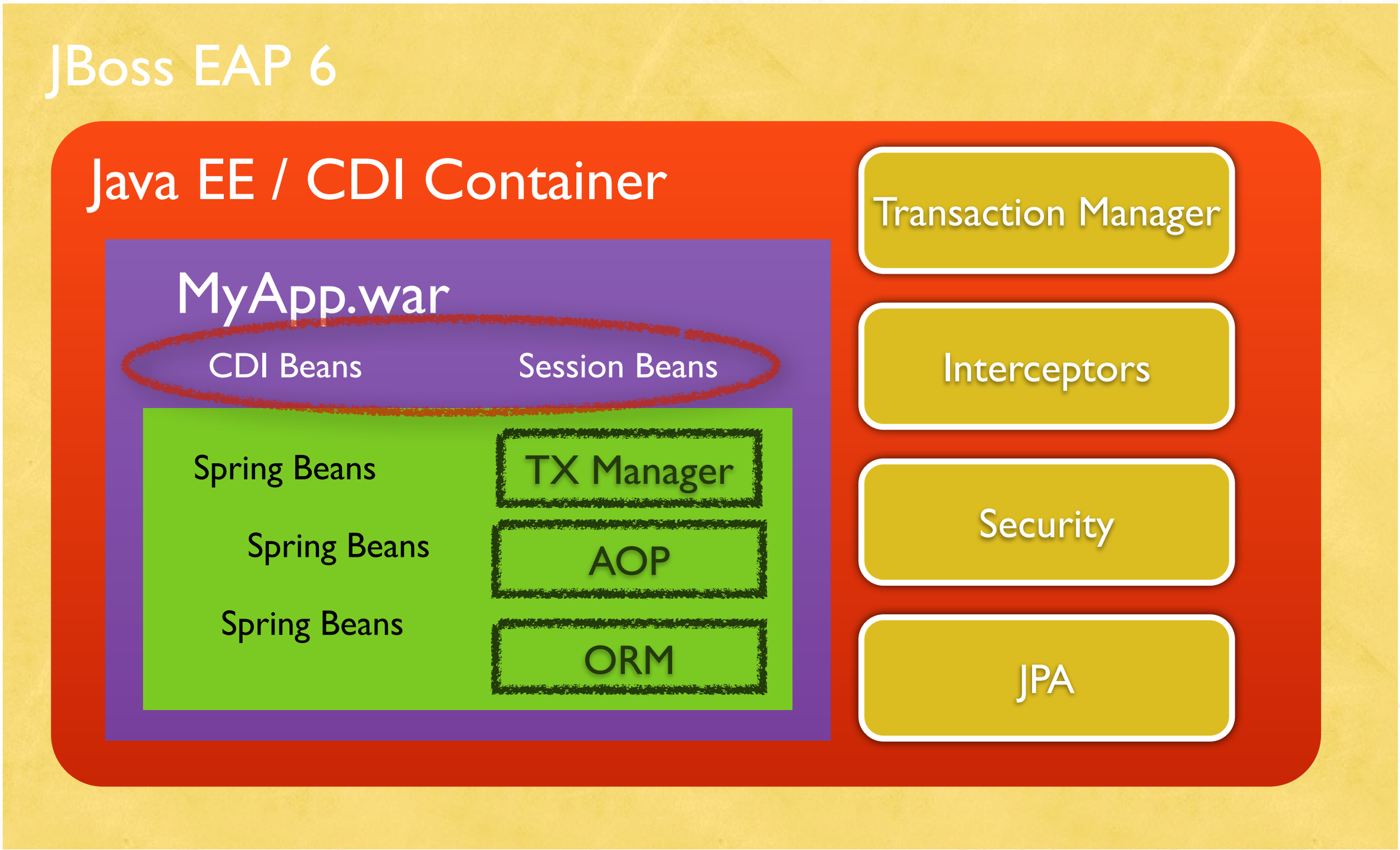
Transaction Manager

Interceptors

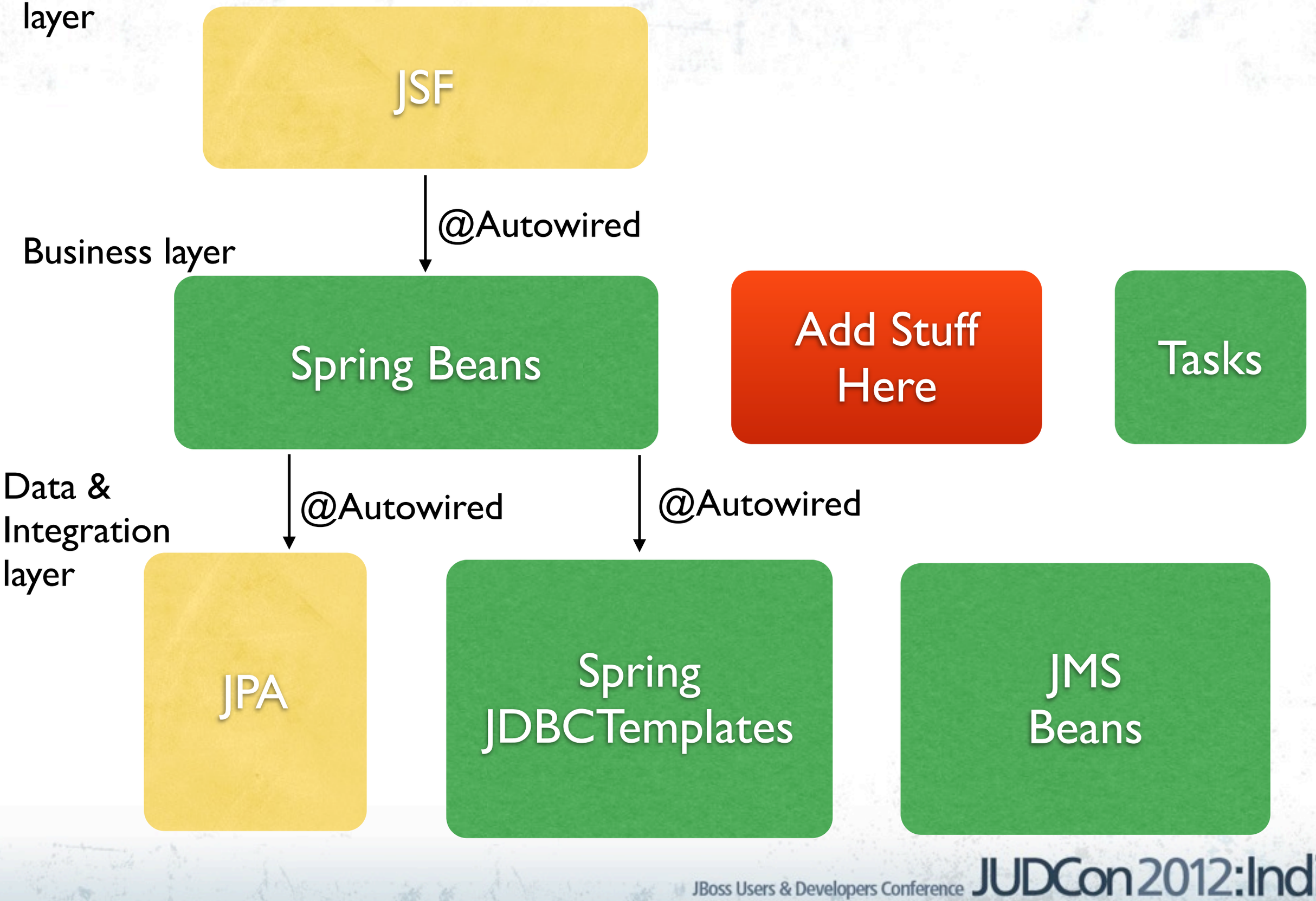
Security

JPA

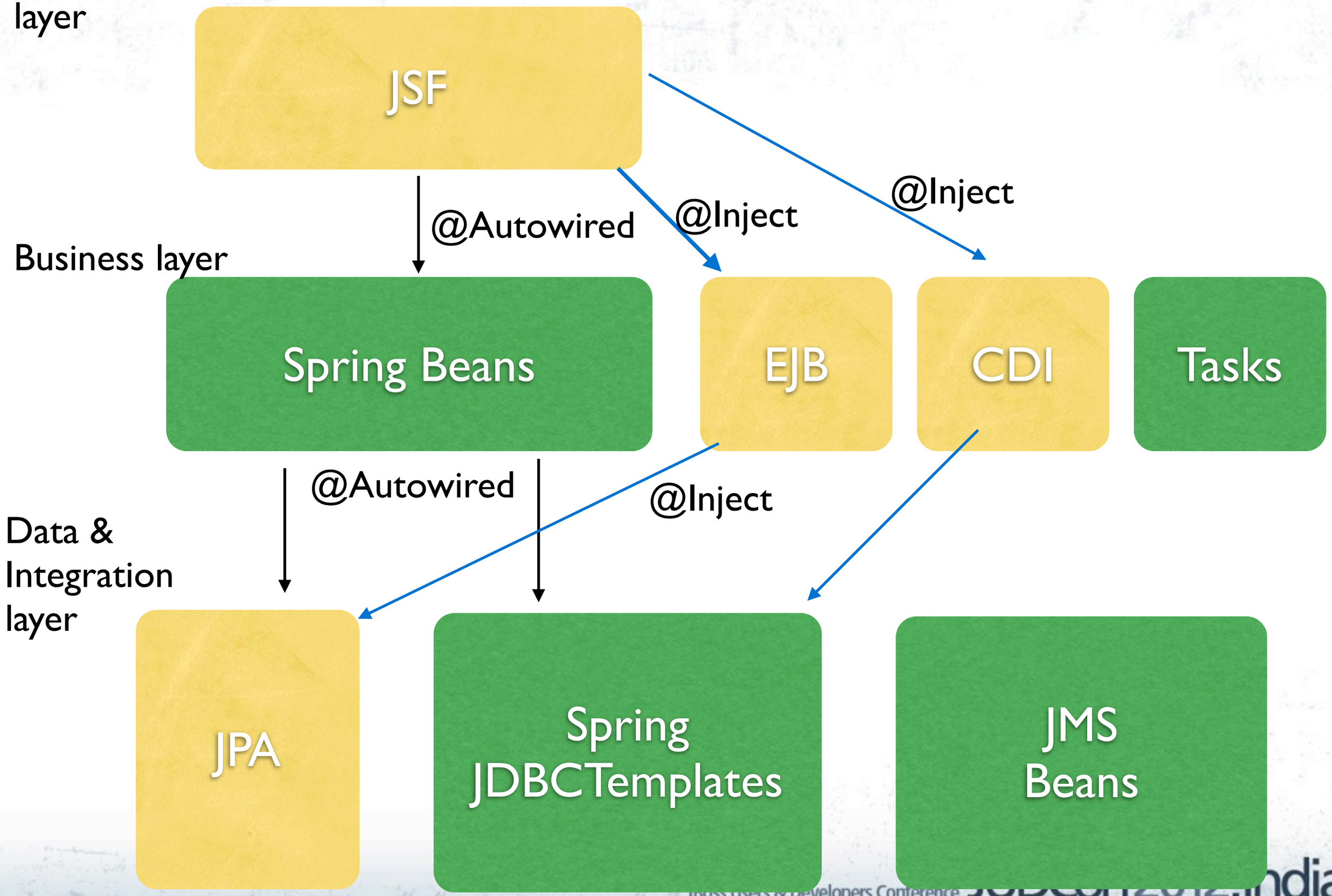
Mixed Application



Replace old frameworks within Spring



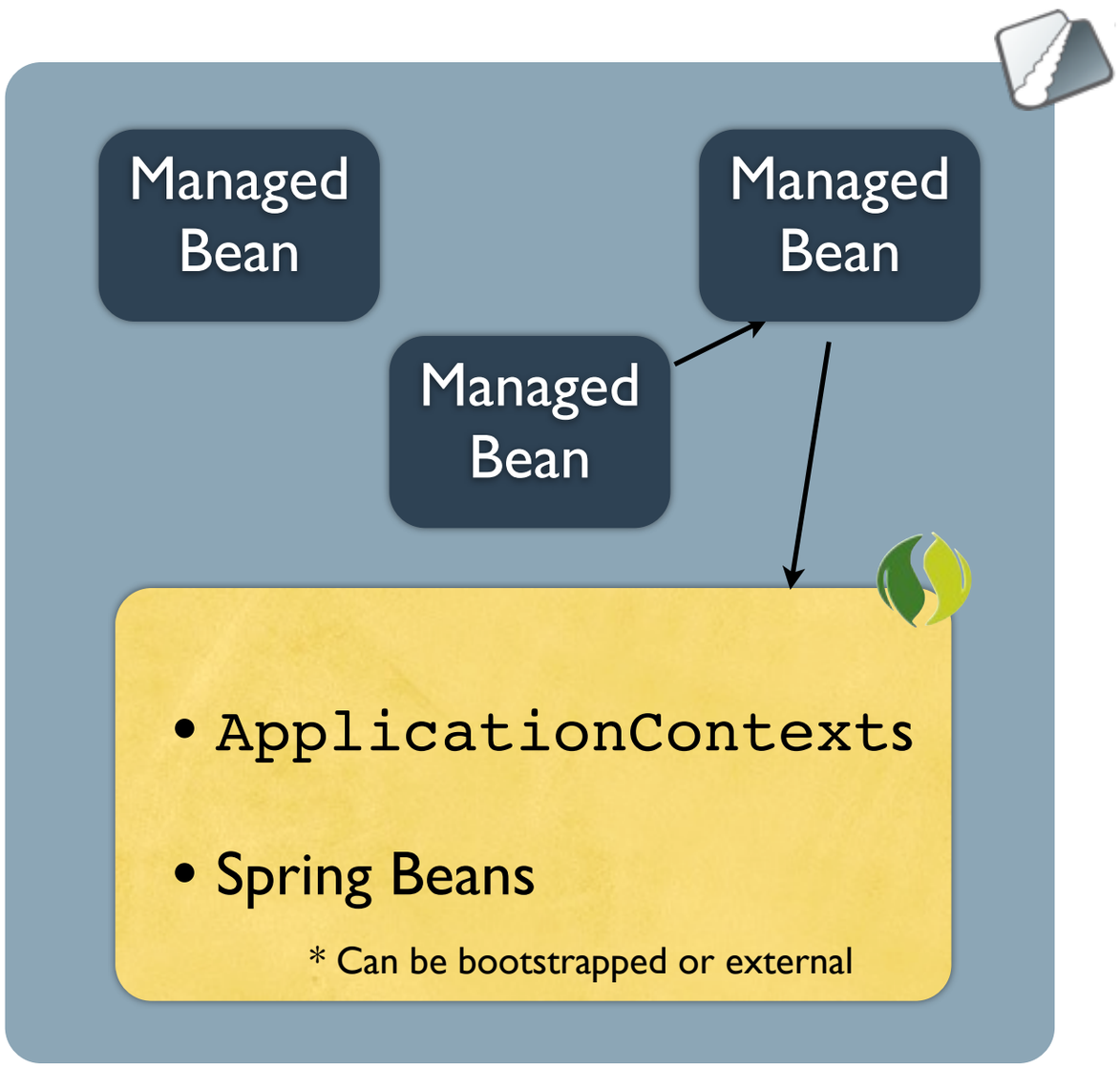
Replace old frameworks within Spring



Dependency Injection

- Spring already supports JSR-330
- Use @Inject anywhere you use @Autowired
- CDI could be almost a drop-in replacement for Spring DI
  - opens up much more powerful options

Integrating Spring components into CDI



- Implemented as a Standard CDI Extension
- Spring contexts and Spring beans are installed as CDI beans
- Implemented using the resource producer pattern

The Spring DAO

```
@Repository
public class BooksJdbcDao {
    private SimpleJdbcTemplate simpleJdbcTemplate;

    public List <String> listBookTitles() {
        return simpleJdbcTemplate.query ("select title from Book",
                new RowMapper<String>() {
            @Override public String mapRow (ResultSet resultSet, int i) throws
            SQLException {
                return resultSet.getString("title");
            }
        });
    }

    @Autowired
    public void setDataSource (DataSource ds){
        simpleJdbcTemplate = new SimpleJdbcTemplate(ds));
    }
}
```

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource”
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.DriverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<context:property-placeholder location="classpath:jdbc.properties"/>
<context:component-scan base-package="demo.spring"/>
<context:anntation-config/>
```

JSF / CDI Bean

```
  @Named
@RequestScoped
public class BooksBean {

  @Inject BooksJdbcDao booksDao;

  public List<String> getTitles() {
    return booksDao.listBookTitles();
  }
}
```

JSF / CDI Bean

```
  @Named
@RequestScoped
public class BooksBean {

  @Inject BooksJdbcDao booksDao;

  public List<String> getTitles() {
    return booksDao.listBookTitles();
  }
}
```

Here we do not want to know about Spring

`@SpringContext` Annotation

1.) Bootstrapped By CDI Extension

```
@Produces @SpringContext
@ApplicationScoped
@Configuration
 (locations={"classpath:springCtx.xml"})
ApplicationContext springContext;
```

Spring Bean Producer

Apply the *resource producer pattern* for creating Spring `ApplicationContext` CDI beans

```
@Produces @SpringBean
  BooksJdbcDao booksDao;
```

```
//Usage
@SessionScoped
public class AllocationManager {
  @Inject
  BooksJdbcDao myBooksDao;
}
```

Migration Path

1.) Upgrade Spring Libraries

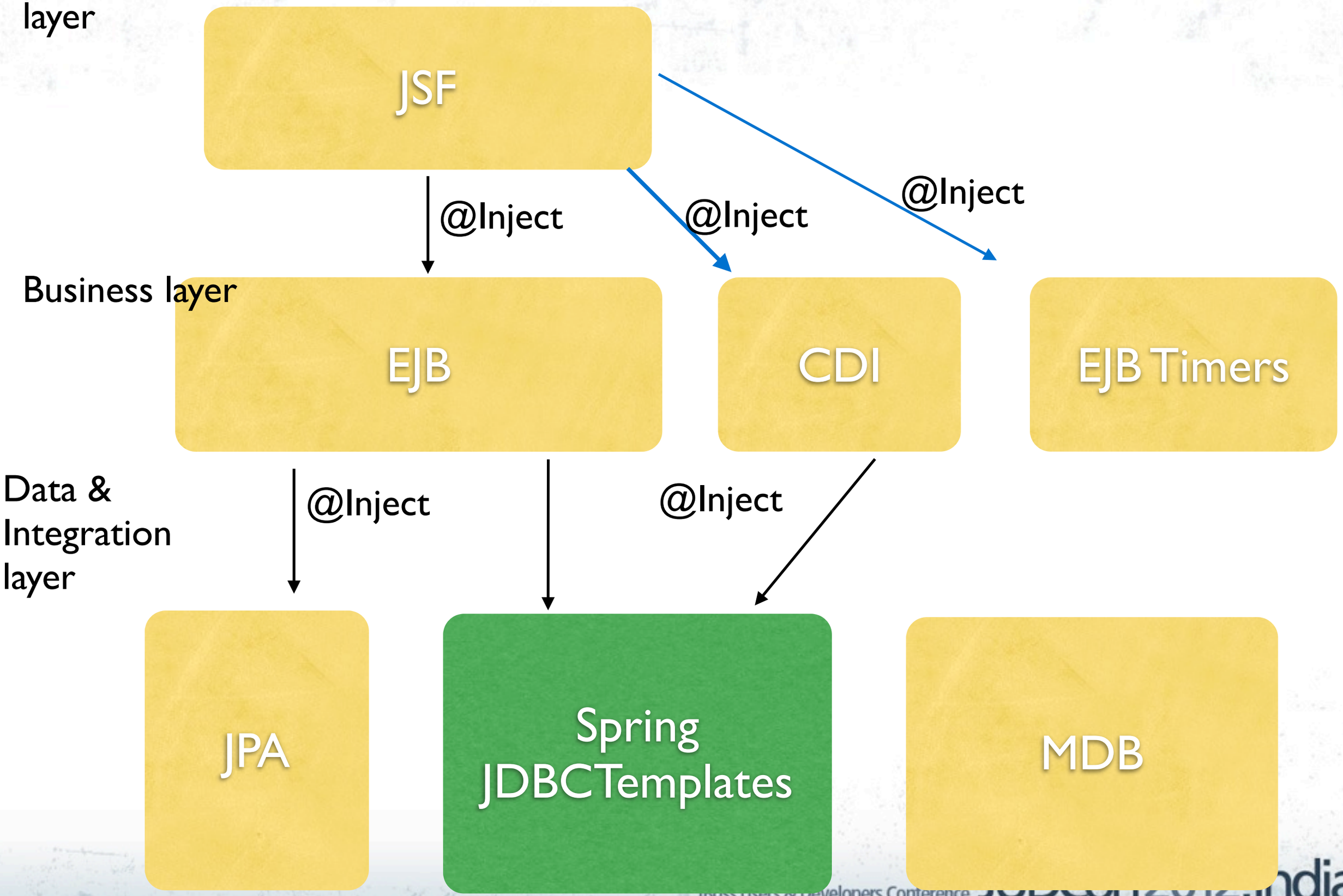
2.) Replace old frameworks (ORM, MVC, etc) within Spring

3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely - DAO layer

5.) Remove Spring container

Replace old frameworks within Spring



The Transaction Layer

- Migrate Spring TX and DAO to EJB
- The TX manager is the Application Server
- An EJB is transactional by default
- EJB has JPA integration

Are you telling me EJBs are cool now?

Are you telling me EJBs are cool now?

Are you telling me EJBs are cool now?

You Bet!

EJBs are just container managed POJOs

Just like Spring beans, but without the container configuration

Spring DAO

```
public class JpaBookDaoImpl implements BookDao {
    @PersistenceContext
    EntityManager em;

    @Override
    public List<Book> listBooks() {
        return em.createQuery ("select b from Book b", Book.class)
        .getResultList();
    }
}
```

```
    <context:component-scan base-package="webshop-dao"/>
    <context:annotation-config/>
    <tx:annotation-driven/>
    <jee:jndi-lookup jndi-name="myDS"/>

    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="myEmf"/>
    </bean>

    <bean id="jpaDialect" class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"/>

    <bean id="myEmf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" value="dataSource"/>
        <property name="dialect" value="jpaDialect"/>
    </bean>
```

The Java EE Alternative

```
@Stateless
public class BookDao {
    @PersistenceContext
    EntityManager em;

    public List <Book> listBooks() {
        return.em.createQuery("select b from Book b", Book.class).getResultList();
    }
}
```

The Java EE Alternative

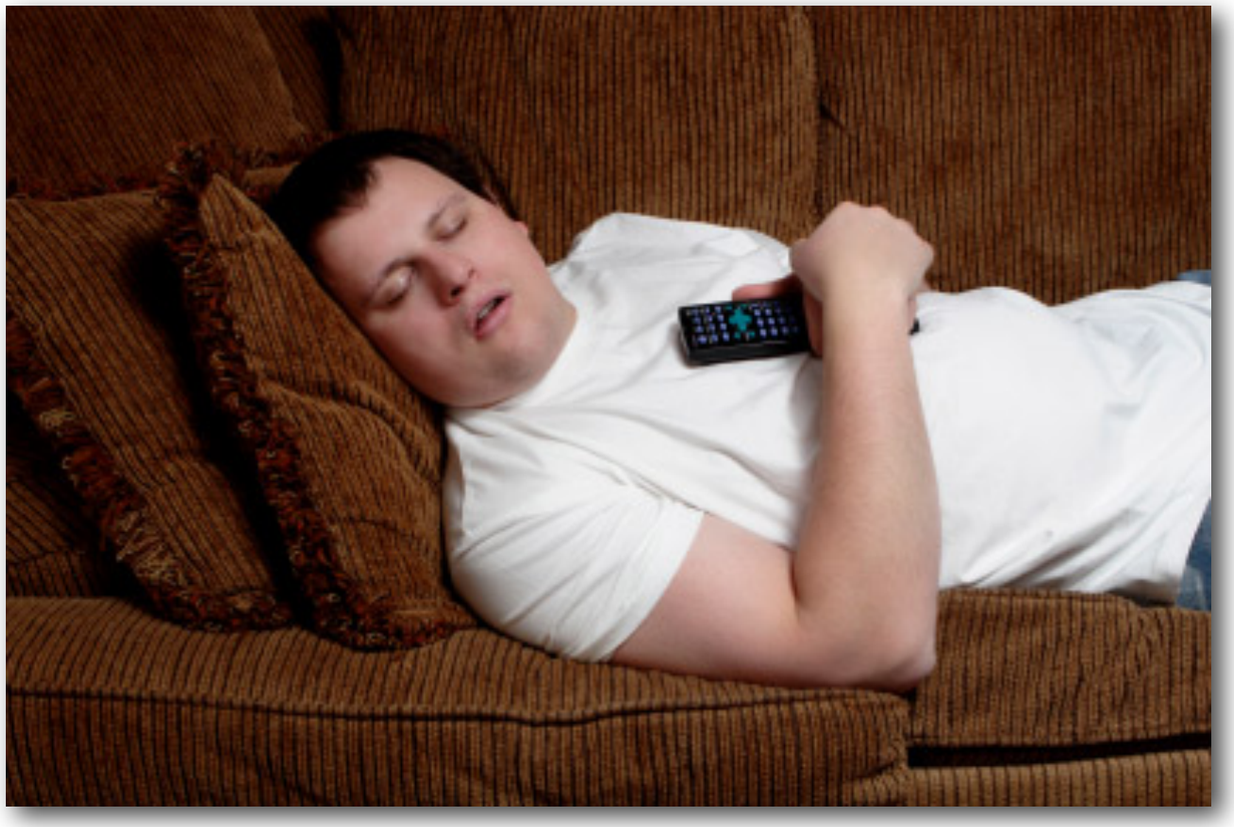
```
@Stateless
public class BookDao {
    @PersistenceContext
    EntityManager em;

    public List <Book> listBooks() {
        return.em.createQuery("select b from Book b", Book.class).getResultList();
    }
}
```

This an EJB (all of it)

Dealing with Lazy Loading

- Many Spring Apps use the pattern *open-EntityManager-in-view*

- EJB has the **Extended** Persistence Context



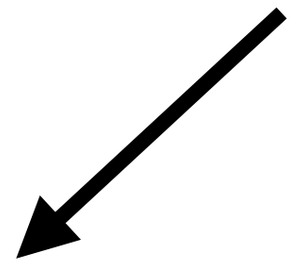
LazyInitializationException

```
@Stateless
public class BookListing {
    @PersistenceContext
    EntityManager em;

    public List <Book> listBooks() {
        return.em.createQuery("select b from Book b", Book.class).getResultList();
    }
}
```

```
<c:repeat value="#{books.authors}" var="author">
    <li>#{author.name}</li>
</c:repeat>
```

Fixing Lazy Loading

```
@Stateful
@Named
@RequestScoped
public class BookListing {
   @PersistenceContext (type=PersistenceContextType.EXTENDED)
   EntityManager em;

   public List <Book> getBooks() {
      return em.createQuery("select b from Book b", Book.class)
      .getResultList();
   }
}
```



Detached Entities

- Spring DAOs are stateless
- Load > edit > save needs a merge
- With an Extended PU we do not

```java
@ConversationScoped
@Stateful
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class UpgradeAccountWizard implements Serializable {

    @PersistenceContext
    EntityManager em;

    private Account account;

    @Inject Conversation conversation;

    public void step1() {
        account = em.find(Account.class, 1L);
    }

    public void step2() {
        //Use Lazy Loading etc.
    }

    public void step3(){
    }

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void finish() {
        conversation.end();
        //end
    }
```

Template Addict?

Hooked on JDBC Templates?

Phase 1: Denial

Phase 2: Migrate

Using JDBC Templates

- Can be injected with a simple Produce method
- Do not rely on Spring container
- (some extra dependencies required)

Template Producer Example

```
public class JdbcTemplateProducer {
    @Resource (mappedName = "java:demo") DataSource ds;

    @Produces
    public SimpleJdbcTemplate jdbcTemplate(){
        return new SimpleJdbcTemplate(ds);
    }
}
```

```
@Inject
private SimpleJdbcTemplate template;

public List<String> getBookTitles() {
    return template.query("select title from Book", new RowMapper<String>() {
        @Override public String mapRow(ResultSet resultSet, int i) throws
        SQLException {
            return resultSet.getString("title");
        }
    });
}
```

Migration Path

1.) Upgrade Spring Libraries

2.) Replace old frameworks (ORM, MVC, etc) within Spring

3.) Run Spring and Java EE container side-by-side

4.) Replace Spring entirely

5.) Remove Spring container

Removing Dependencies

- Our classpath just has to contain APIs, no framework classes
- From ~40 dependencies to just:

```
<dependency>
  <groupId>org.jboss.spec</groupId>
  <artifactId>jboss-javaee-6.0</artifactId>
  <version>1.0.0.Final</version>
  <type>pom</type>
  <scope>provided</scope>
</dependency>
```

What about testing?

Spring is pretty good at this

Java EE was not

CDI 'Alternative'

```
@Stereotype
@Alternative
@Target({ TYPE })
@Retention(RUNTIME)
public @interface Mock {
}
```

```
<alternatives>
  <stereotype>snippets.ejb.Mock</stereotype>
</alternatives>
```

```
@Mock
public class MockBookDao implements BookDao {
   @Override
   public List<String> listBooks() {
      return Arrays.asList("A","B","C");
   }
}
```

```java
@Stateful
@Named
@RequestScoped
public class BookListing {
   @PersistenceContext (type=PersistenceContextType.EXTENDED)
   EntityManager em;

   public List <Book> getBooks() {
      return em.createQuery("select b from Book b", Book.class)
      .getResultList();
   }
}
```

```java
@Stateful
@Named
@RequestScoped
public class BookListing {
   @PersistenceContext (type=PersistenceContextType.EXTENDED)
   EntityManager em;

   public List <Book> getBooks() {
      return em.createQuery("select b from Book b", Book.class)
      .getResultList();
   }
}
```

What about JPA Code?

Arquillian

Create micro-deployments
using an API

Run tests in real containers

Arquillian Example

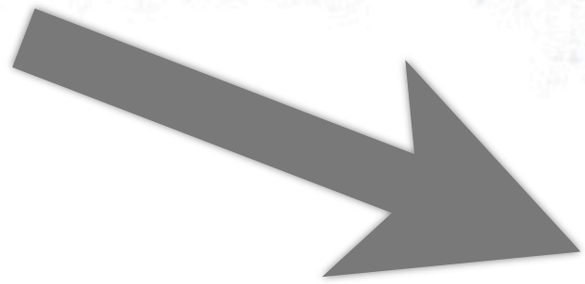
```
@RunWith(Arquillian.class)
public class BooksDaoTest {
    @Inject private BookListing bookdao;

    @Deployment public static JavaArchive createDeployment() {
        return ShrinkWrap.create(JavaArchive.class, "test.jar")
                .addClass(Book.class)
                .addClass(BookListing.class)
                .addAsManifestResource(EmptyAsset.INSTANCE, ArchivePaths
                .create("beans.xml"))
                .addAsManifestResource("META-INF/persistence.xml", "persistence.xml")
    }

    @Test public void testIsDeployed() {
        Assert.assertNotNull(bookdao);
    }

    @Test public void testList() {
        assertEquals(0, bookdao.listTitles().size());
    }
}
```

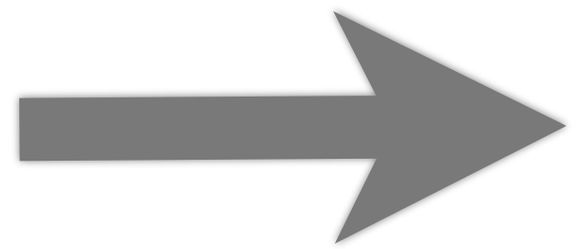
In Conclusion

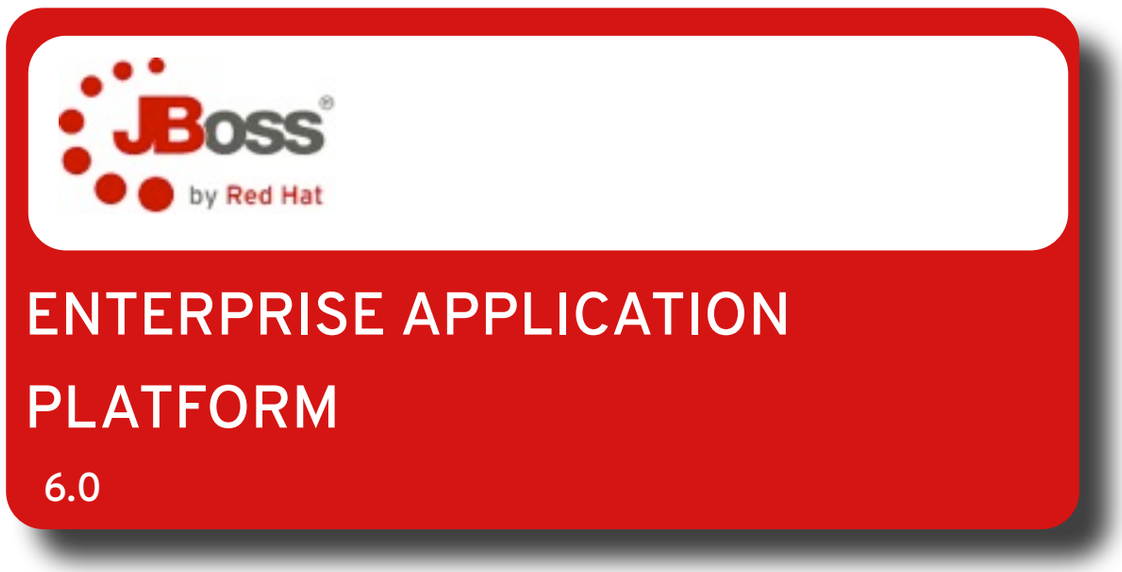
Is There Life After Spring?

- Spring is a fine framework
  - JBoss continues it's support for Spring on JBoss EAP
- Java EE isn't painful, heavy or complex
- There is always room for innovations
- When it works, bring it back to the specification

JBoss AS 7.1







SUBSCRIPTIONS = DEVELOPER PRODUCTIVITY

"Q: I'm writing a WS-Policy and..."

Frees you to focus on the new stuff

Seam 3 Spring module http://bi

KILL
DRUDGERY

ACCELERATE
YOUR PROJECT!

Fight your bugs, not others'

http://www.redhat.com/jboss/

Thank You!

Seam 3 Spring Module - http://bit.ly/Seam3Spring

Example Code - http://bit.ly/JUDConSeamSpringCode

Apache DeltaSpike - http://bit.ly/zsWz2x

Ray Ploski

@rayploski

Credits

Thanks to Paul Bakker and Bert Ertman from Luminis who originally posted much of this material at JavaOne 2011.