

Best Practices for Migrating Spring to

JAVA EE 6



WTF !?





Bert Ertman

Fellow at Luminis in the Netherlands
JUG Leader for NLJUG and a Java Champion



@bertertman



Paul Bakker

Senior developer at Luminis Technologies



@pbakker





Who is this talk for?

- You are using old school Spring and wonder how to move forward
- Java EE seems to be hot again, should you jump on this train?
- You love the J2EE Design and Development book; but is it still actual?

Why listen to us?

We've had endless
Java EE vs.
Spring
discussions...

..not employed by
an application
server vendor

..worked
extensively with
J2EE, Spring and
modern Java EE

..and we're no
Rod Johnson
groupies either

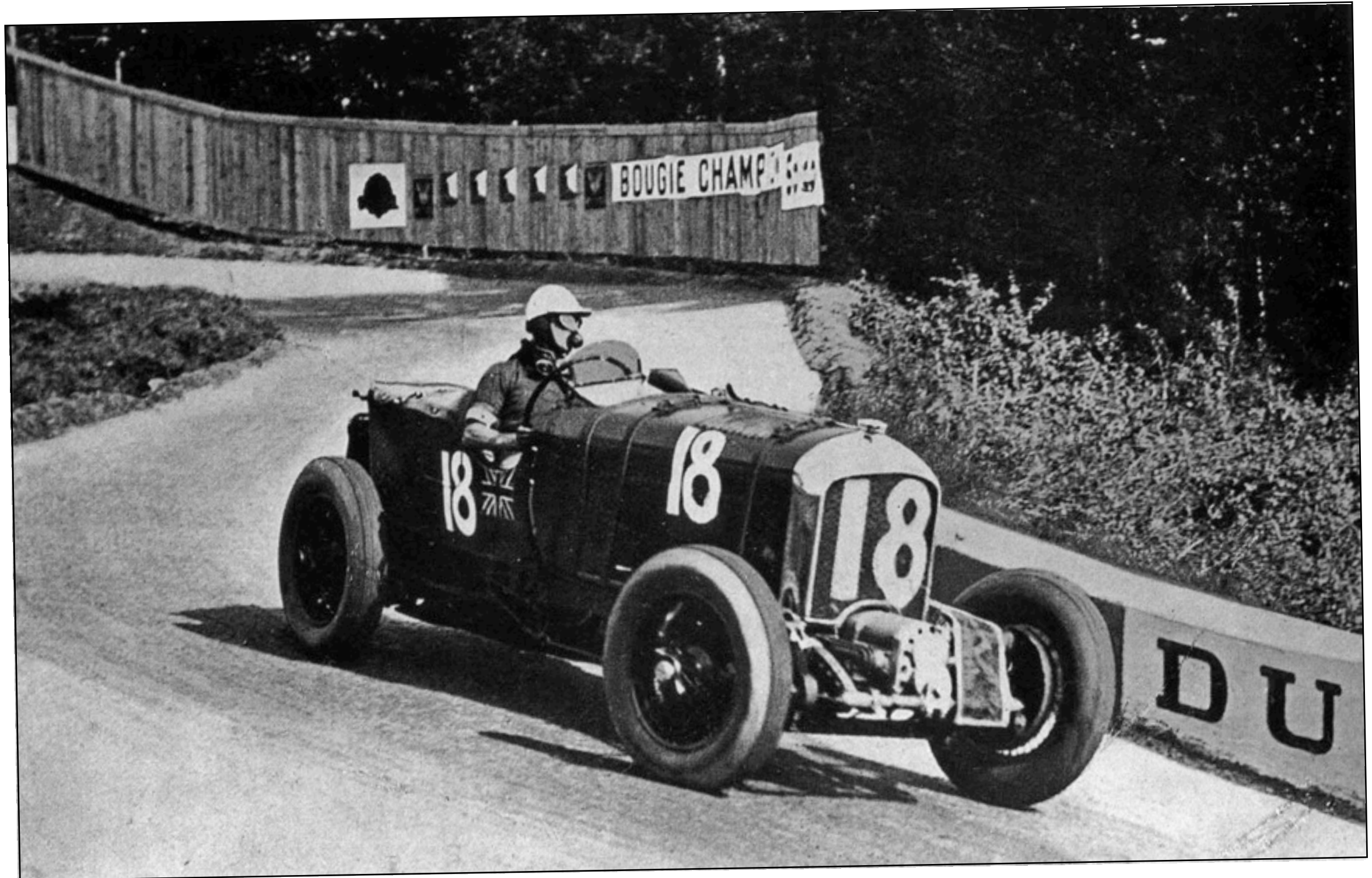
Why migrate?

upgrading from
old school Spring
requires a lot of
work anyway

Spring is
proprietary
technology

why not take it to
the standard?

Your app might have been competitive in the past...





But now we want THIS!

Let's get

a couple of misunderstandings

out of the way

first...



Isn't Java EE too fat?

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

Isn't Java EE too fat?

- Startup times with application deployed
 - JBoss AS 7 ~2 seconds
 - Glassfish V3 ~4 seconds
 - Tomcat 6 + Spring ~10 seconds
- Java EE 6 WAR file < 100MB

Our favorite quote:

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

Andrew L. Rubinger

But I need dependency injection?

- Java EE 6 introduced CDI
 - More powerful, contextual DI model
 - Makes the platform extensible in a standard way



But I must have AOP!

Really?

You love getting
your code all
asymmetric and
unreadable?

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





do I need heavy tooling?



do I need heavy tooling?

Capabilities comparison

Capability	Spring	JAVAEE
Dependency Injection	Spring Container	CDI
Transactions	AOP / annotations	EJB
Web framework	Spring Web MVC	JSF
AOP	AspectJ (limited to Spring beans)	Interceptors
Messaging	JMS	JMS / CDI
Data Access	JDBC templates / other ORM / JPA	JPA
RESTful Web Services	Spring Web MVC (3.0)	JAX-RS
Integration testing	Spring Test framework	Arquillian *

* Not part of the Java EE specification

Apparently,

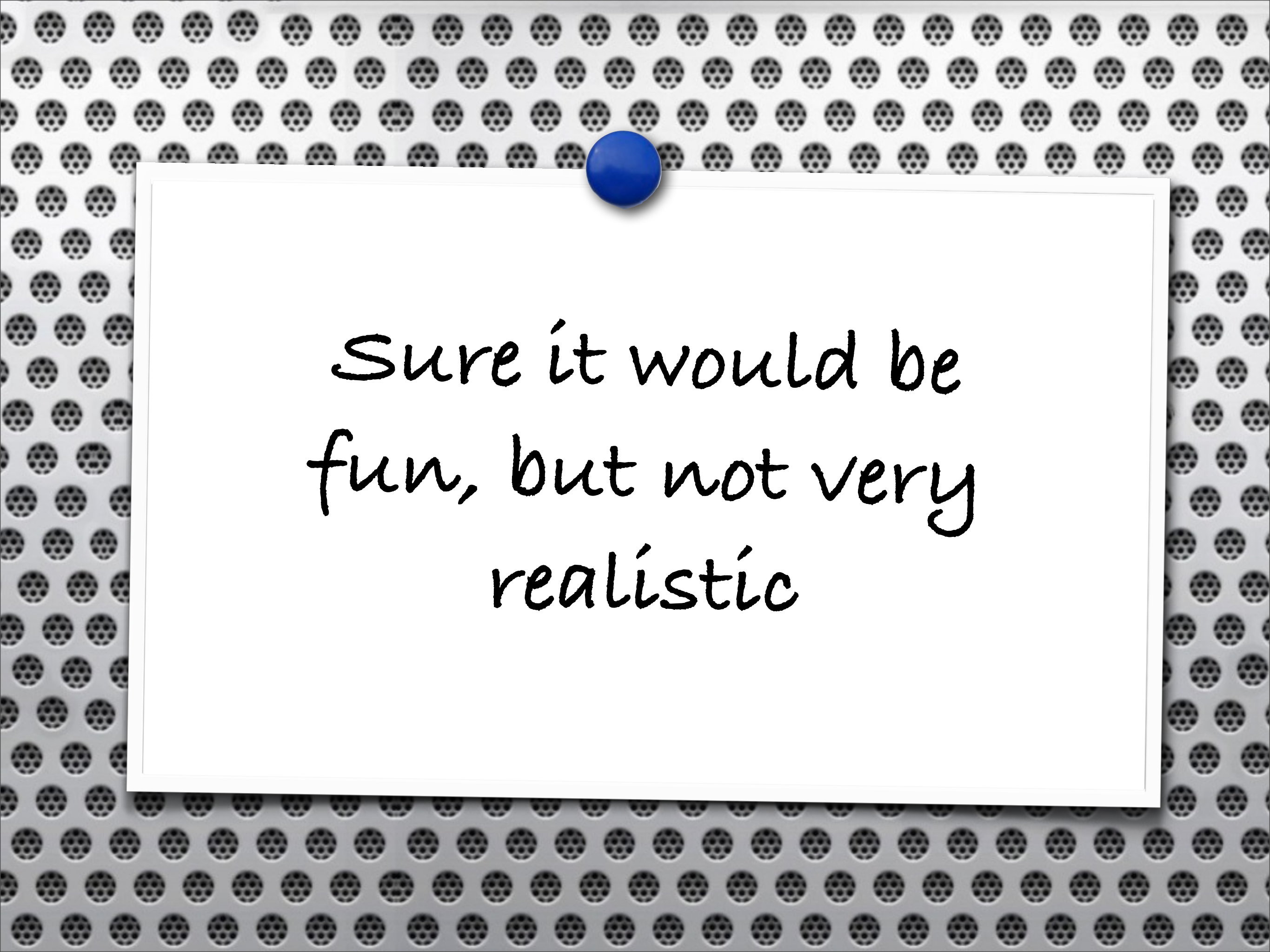
it can all be done using

plain vanilla light weight

Java EE



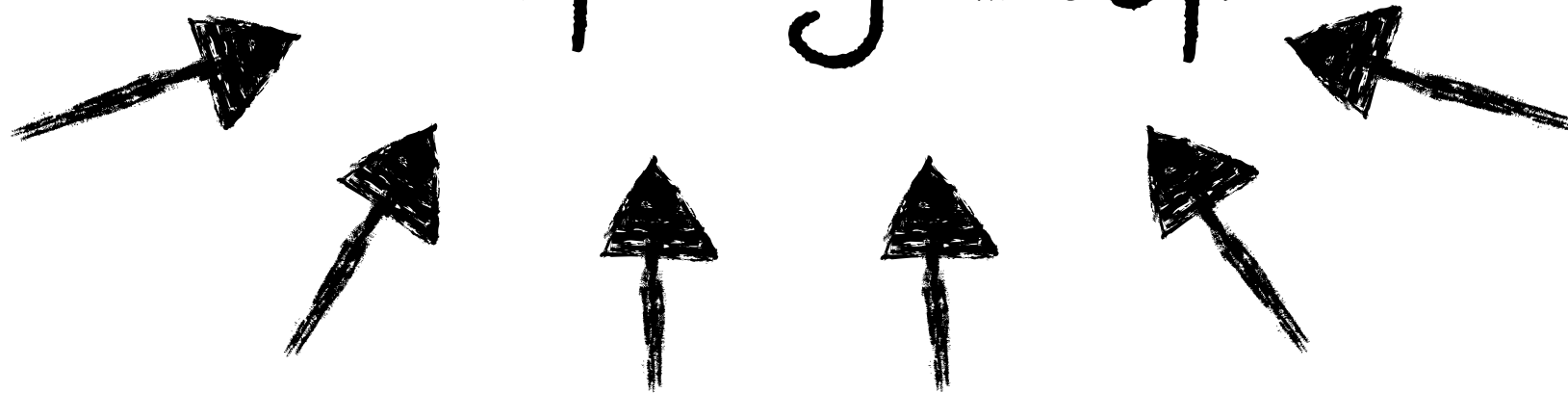
rm -Rf spring*



Sure it would be
fun, but not very
realistic



We need a recipe that
moves us forward
step by step





Typical old school Spring app

- ❑ lots of complex XML, no annotations
- ❑ old / outdated ORM solution
 - JDBC Templates, Kodo, Toplink, etc.
- ❑ deprecated extension based web MVC (SimpleFormController, etc.)



Migration path

1. Upgrade Spring version
2. Replace old frameworks
(ORM, web framework)
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 version
2. Replace old frameworks
(ORM, web framework)
within Spring
3. Run Spring and Java EE
container side by side
4. Replace Spring entirely
5. Remove Spring container



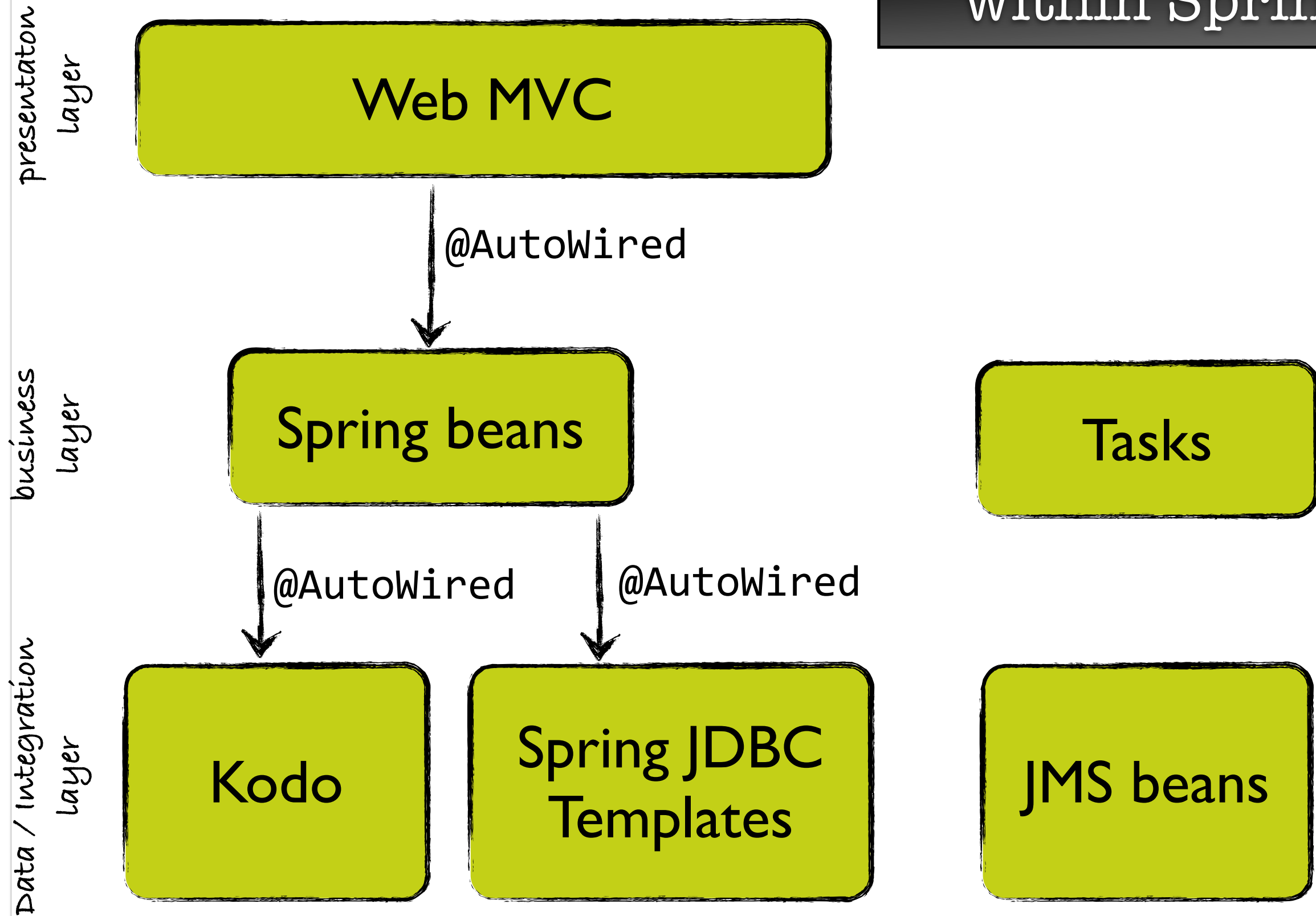
Upgrade Spring version

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

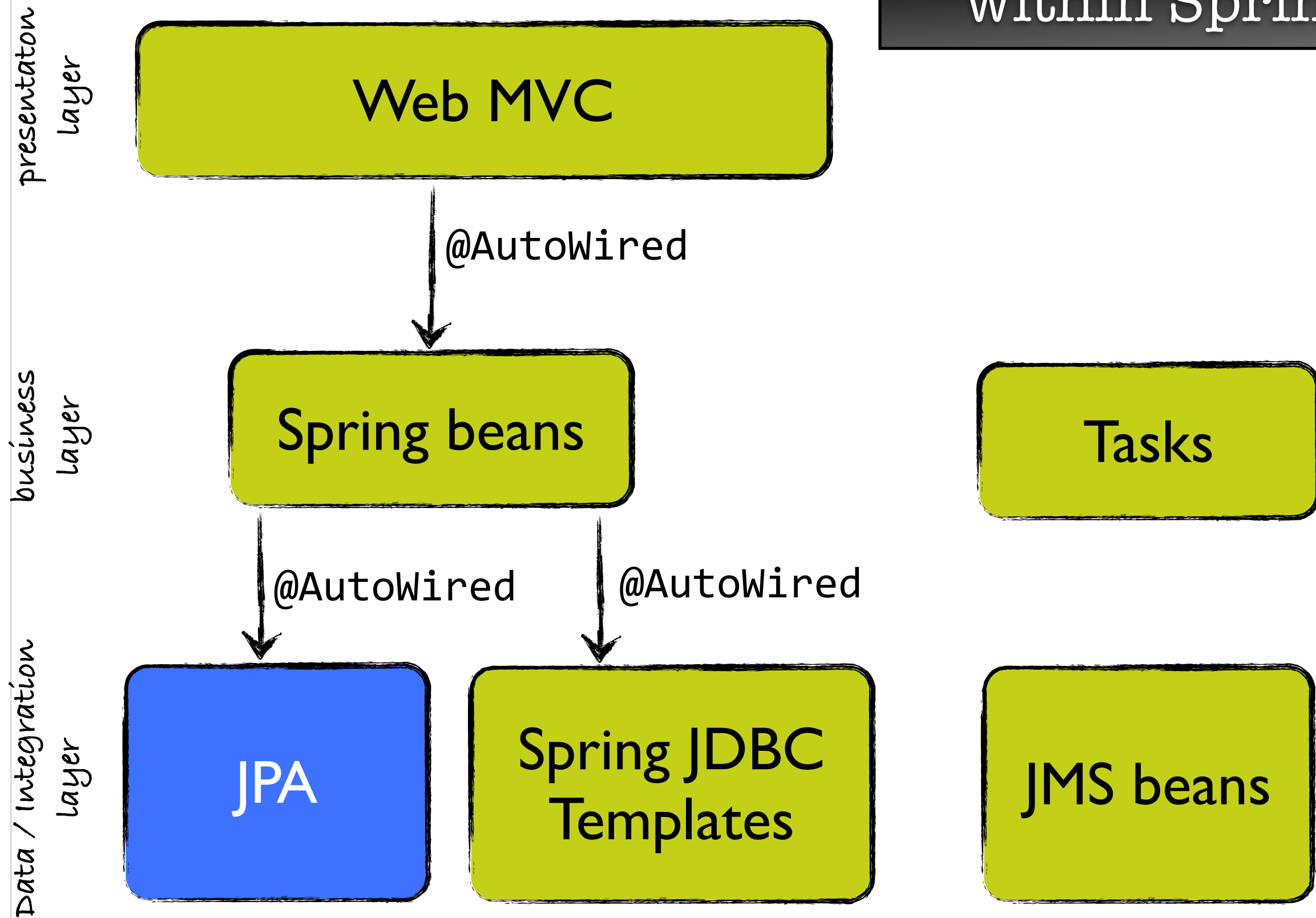
Migration path

1. Upgrade Spring version
2. Replace old frameworks
(ORM, web framework)
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



replace old frameworks
within Spring



replace old frameworks
within Spring

presentation
layer

JSF

@Autowired

business
layer

Spring beans

Tasks

Data / Integration
layer

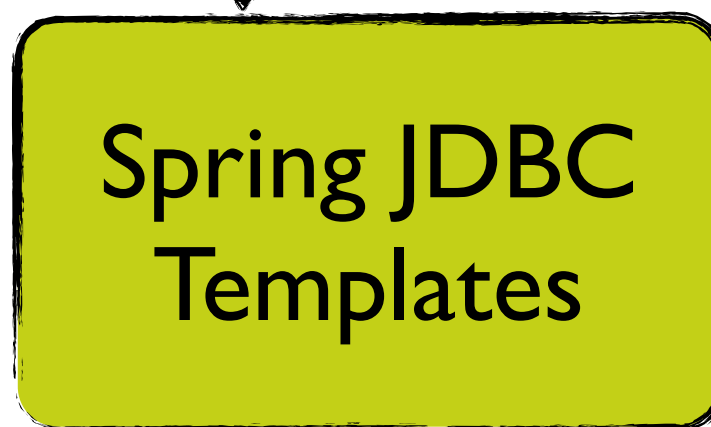
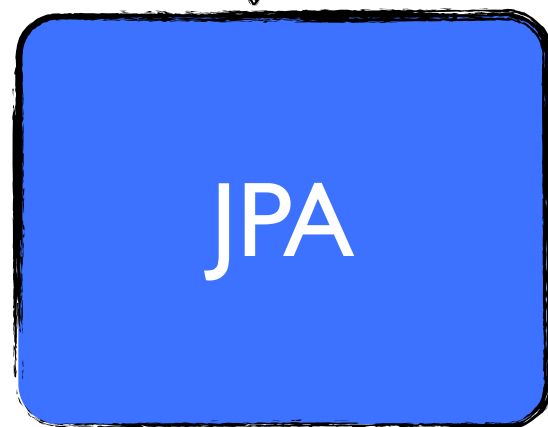
JPA

@Autowired

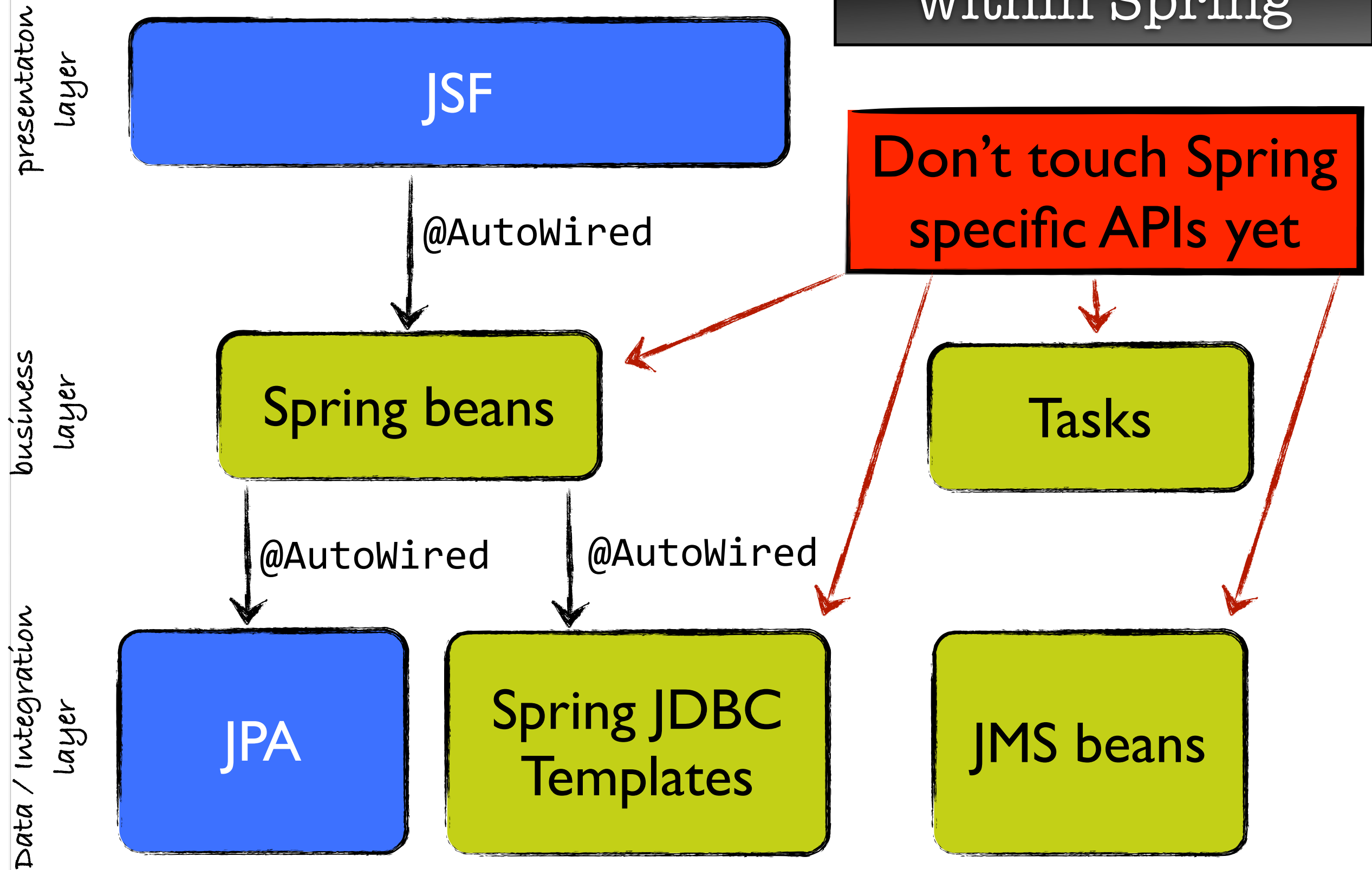
@Autowired

Spring JDBC
Templates

JMS beans



replace old frameworks
within Spring



Migration path

1. Upgrade Spring version
2. Replace old frameworks
(ORM, web framework)
within Spring

3. Run Spring and Java EE
container side by side

4. Replace Spring entirely
5. Remove Spring container

using Spring and Java EE side-by-side

Disclaimer 1: this is just to give an idea, this is not a production ready framework!

Disclaimer 2: this is a migration path. Not the way you should build your next application...

Spring application

Servlet Container

myapp.war

Spring container

TX manager

AOP

ORM

Spring
beans

Spring
beans

Java EE application

Java EE 6 application server

CDI / EJB container

TX manager

Security

Interceptors

JPA

myapp.war

CDI beans

Session beans

Mixed

Java EE 6 application server

CDI / EJB container



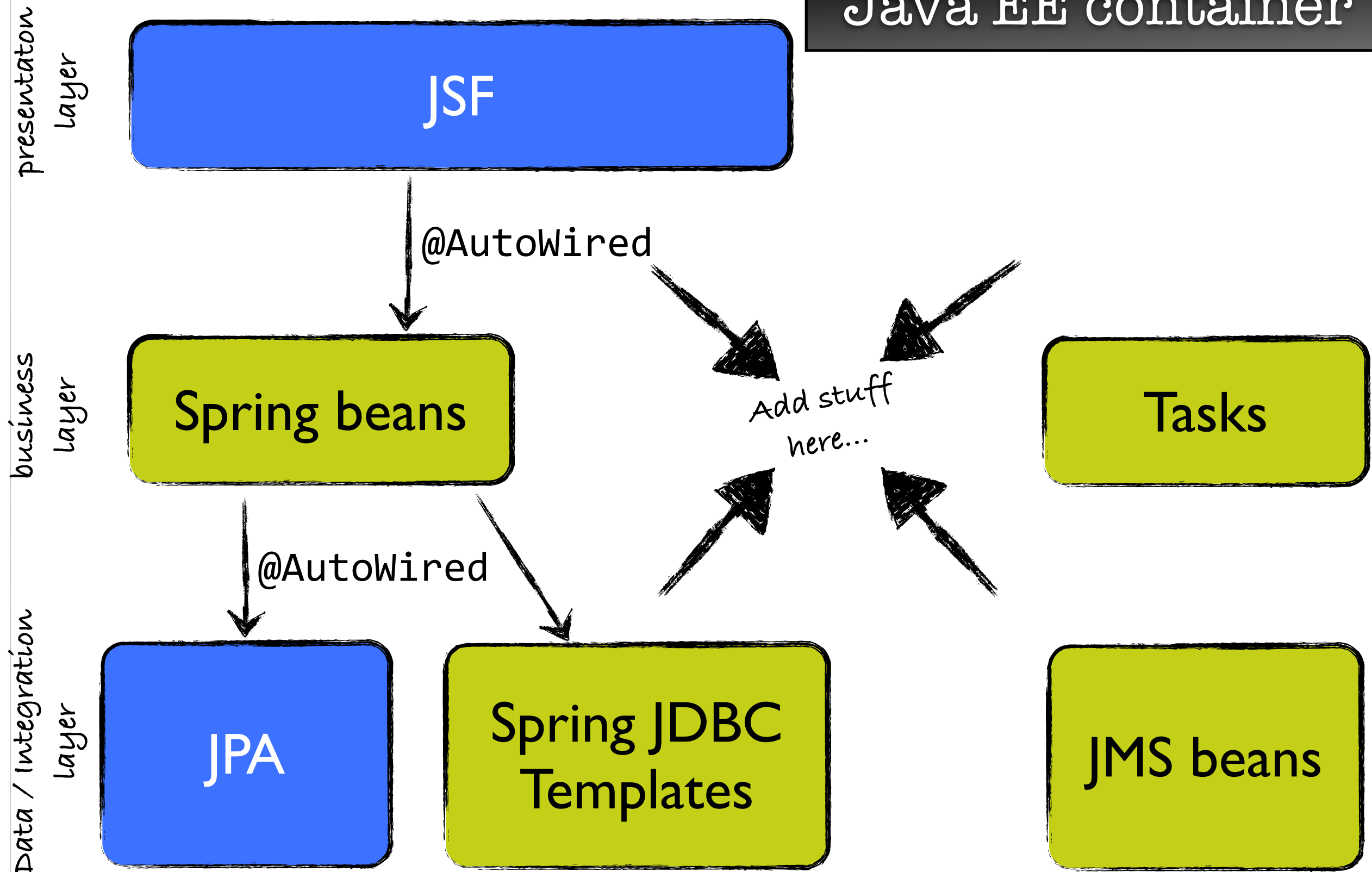
myapp.war

CDI beans
Session beans

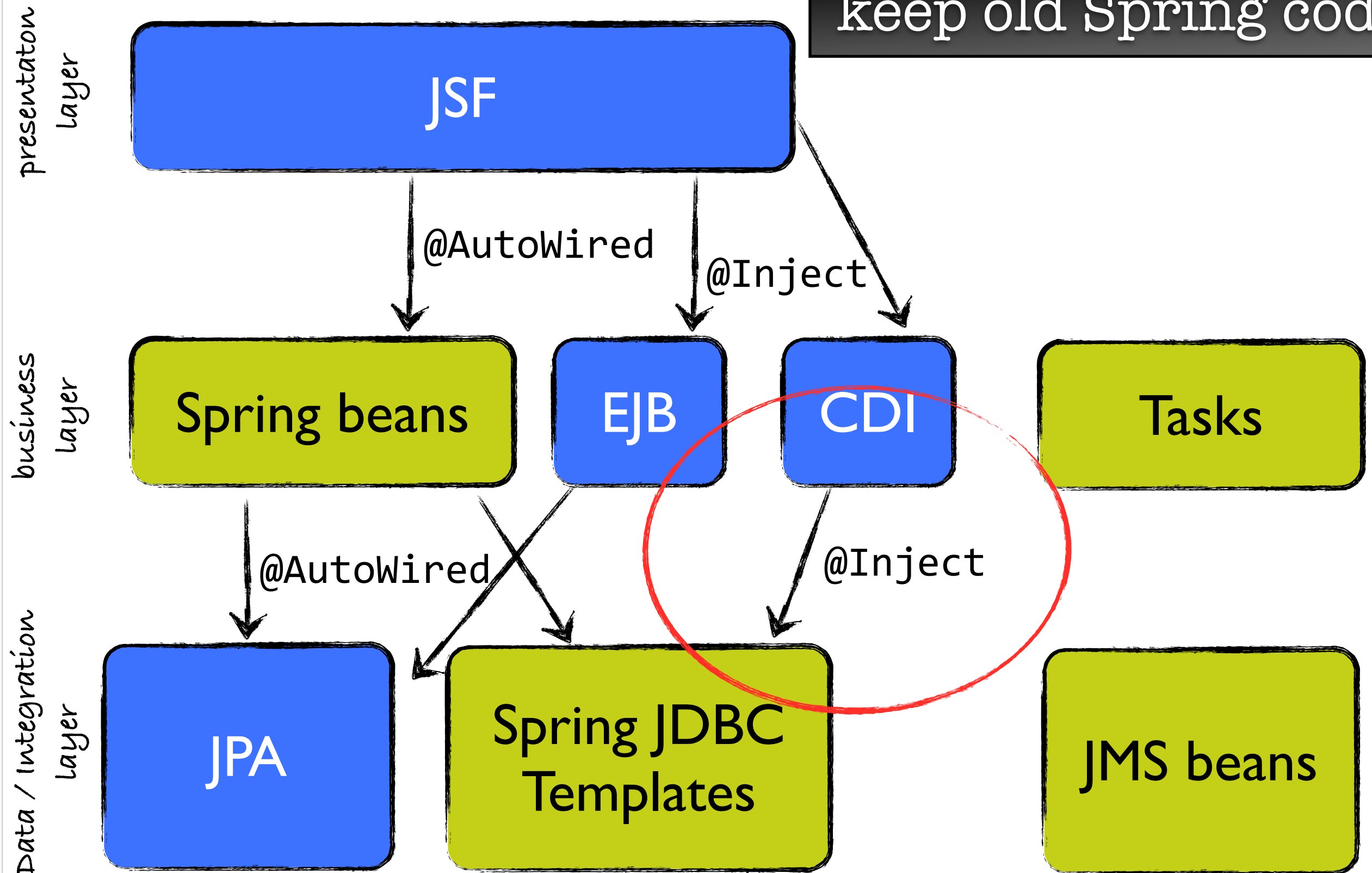
Spring



run Spring within
Java EE container



add Java EE code
keep old Spring code





Encapsulate Spring with CDI

- Write a CDI extension that
 - bootstraps the Spring container
 - looks up Spring Beans in the Spring container and publish in CDI context

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:annotation-config/>
```


JSF / CDI bean



```
@Named
@RequestScoped
public class BooksBean {
    @Inject BooksJdbcDao booksDao;

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

Here we don't
want to know
about Spring

CDI extension example

```
public class StartupListener implements Extension {
    ApplicationContext applicationContext;

    void beforeBeanDiscovery(@Observes BeforeBeanDiscovery bbd, BeanManager bm) {
        applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");
    }

    void afterBeanDiscovery(@Observes AfterBeanDiscovery abd, BeanManager bm) {
        AnnotatedType<BooksJdbcDao> at = bm.createAnnotatedType(BooksJdbcDao.class);

        final InjectionTarget<BooksJdbcDao> it = bm.createInjectionTarget(at);
        abd.addBean(new Bean<BooksJdbcDao>() {
            @Override public BooksJdbcDao create(CreationalContext<BooksJdbcDao> ctx) {

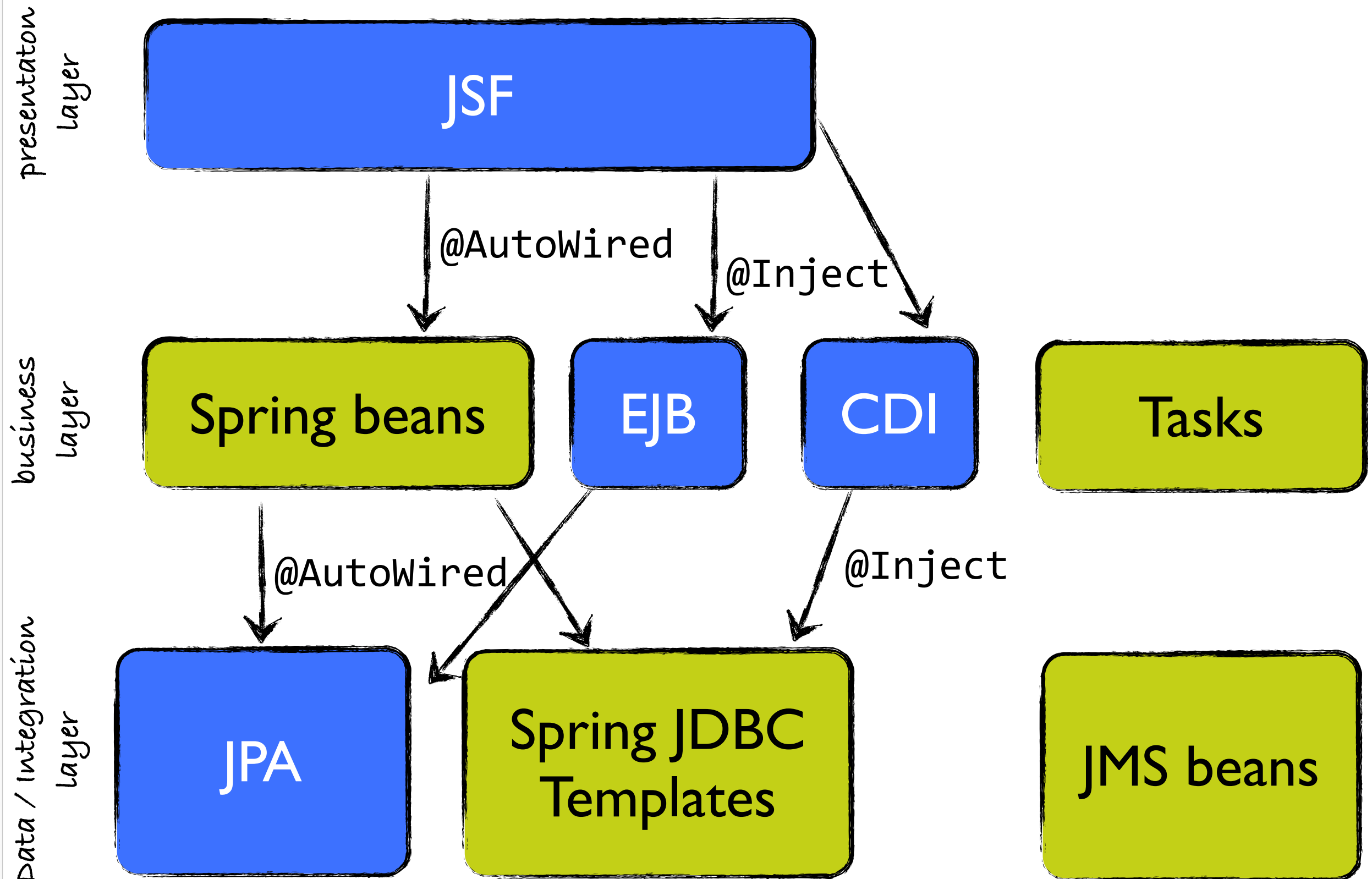
                BooksJdbcDao instance = applicationContext.getBean(BooksJdbcDao.class);
                it.inject(instance, ctx);
                it.postConstruct(instance);
                return instance;
            }
        });
    }
}
```

<https://github.com/paulbakker/migrating-spring-to-javaee>

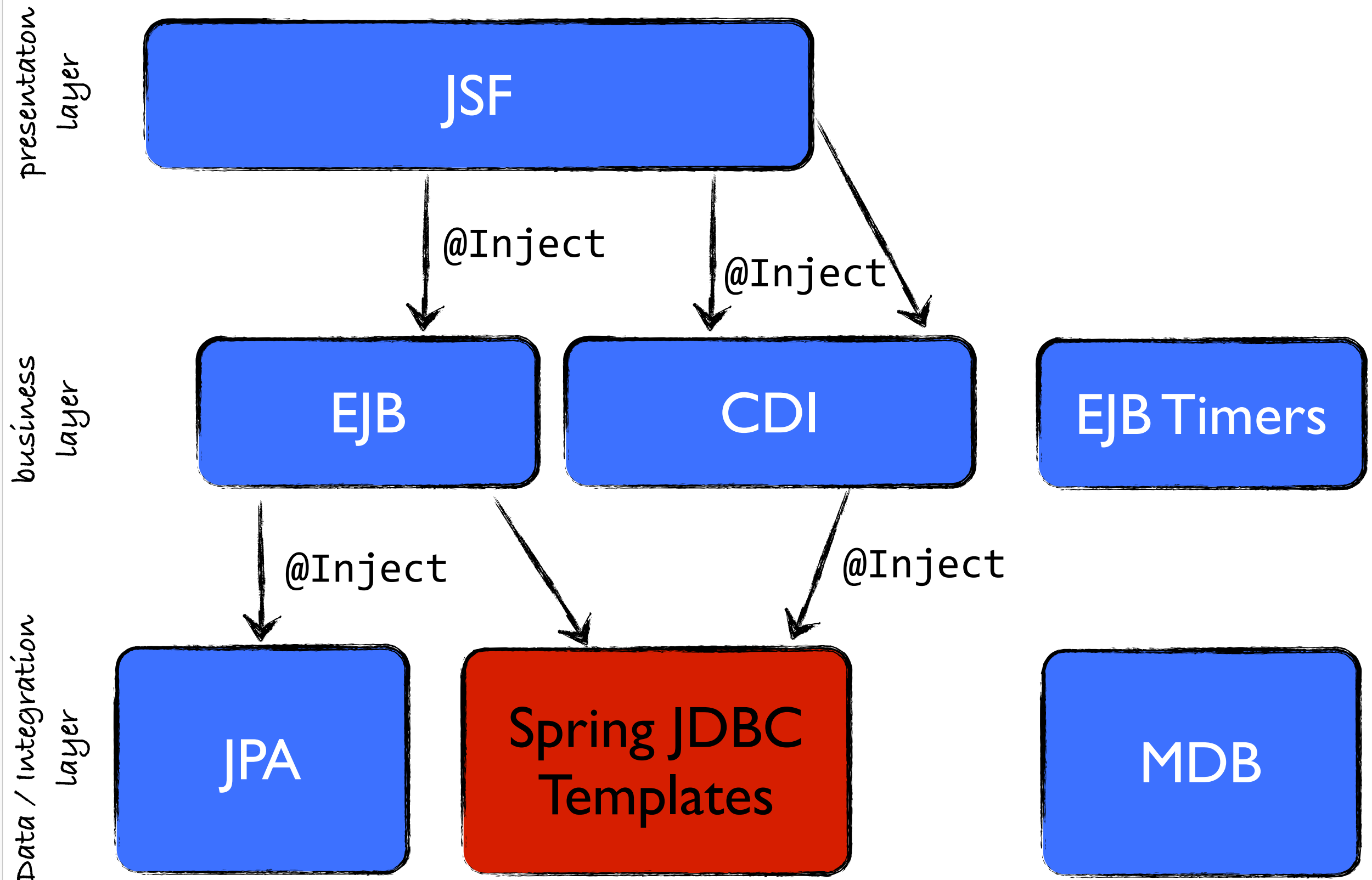
Migration path

1. Upgrade Spring version
2. Replace old frameworks
(ORM, web framework)
within Spring
3. Run Spring and Java EE
container side by side
4. Replace Spring entirely
5. Remove Spring container

entirely replace Spring



entirely replace Spring





The TX Layer

- Migrate Spring TX and DAOs to EJB
- The TX manager is in the app server
- An EJB is transactional by default
- EJB has JPA integration

Come on, 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

The Spring DAO

```
@Repository
@Transactional
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" ref="dataSource"/>
    <property name="jpaDialect" ref="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();
    }
}
```

btw -
this is an
EJB! :)

Dealing with lazy loading

- Many Spring apps use the Open-EntityManager-In-View pattern
- EJB has the Extended Persistence Context



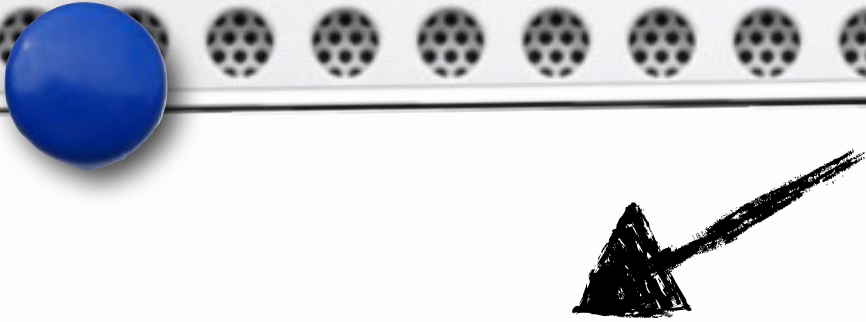
LazyInitializationException

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

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

```
<c:repeat value="#{book.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 (all singletons)
- ❑ Load > edit > save needs a merge
- ❑ With an Extended PU we don't



```
@ConversationScoped
@Stateful
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class UpgradeAccountWizard implements Serializable{
    @PersistenceContext(type = PersistenceContextType.EXTENDED)
    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();
        //entity merges automatically
    }
}
```


Template addict?

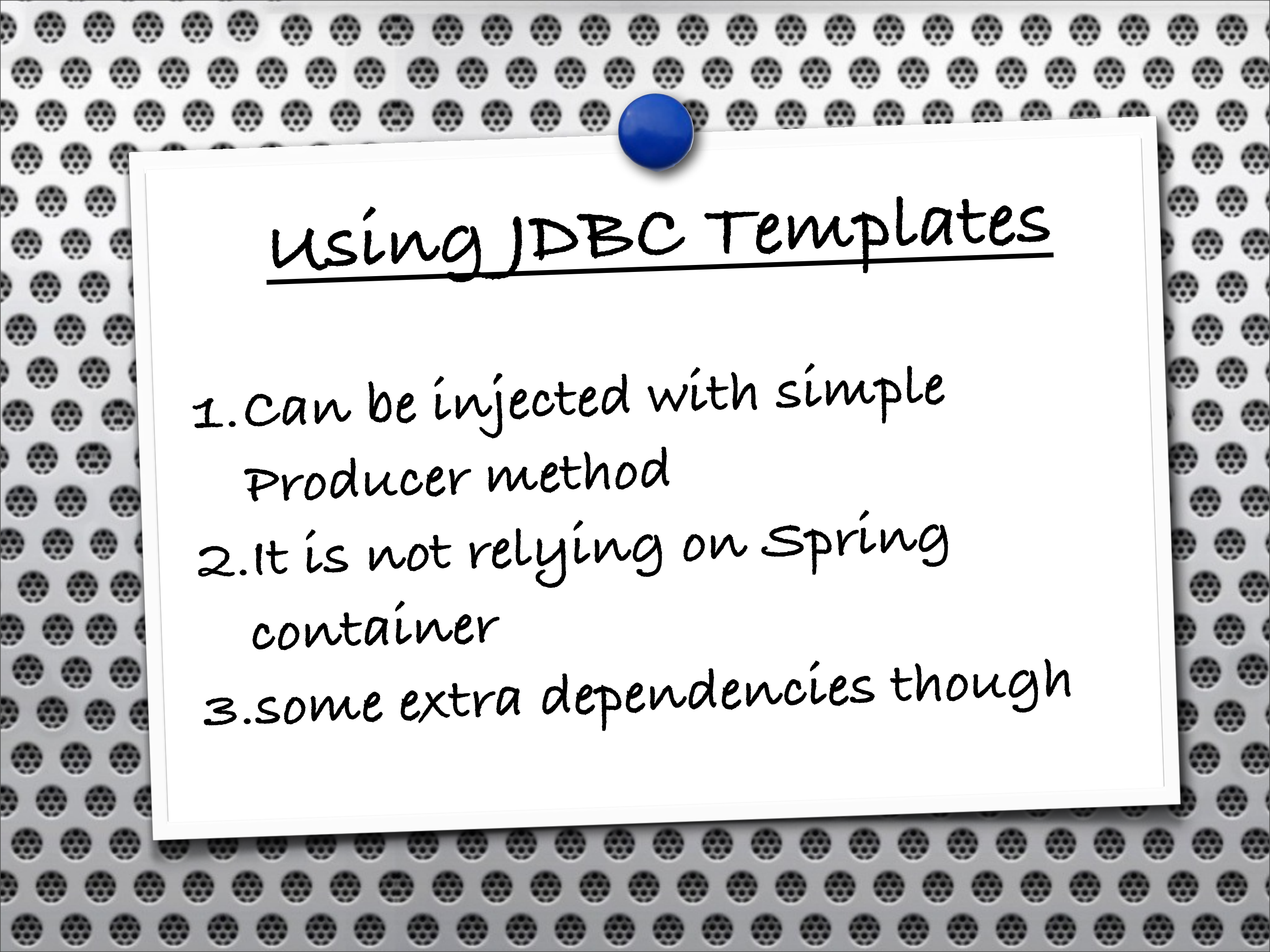


Hooked to JDBC Templates!?

Hmm, let's start
the old discussion
whether or not to
use ORM...

Phase 1: denial

Phase 2: migrate ;-)



Using JDBC Templates

1. Can be injected with simple
Producer method
2. It is not relying on Spring
container
3. some extra dependencies though

Template producer example

```
public class JdbcTemplateProducer {  
    @Resource(mappedName = "java:demo") DataSource ds;  
  
    @Produces  
    public SimpleJdbcTemplate jdbcTemplate() {  
        return new SimpleJdbcTemplate(ds);  
    }  
}
```

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


Migration path

1. Upgrade Spring version
2. Replace old frameworks
(ORM, web framework)
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

How to test this?

```
@Model
public class ContactsBean {
    @Inject ContactsDao contactsDao;

    public List<String> getContactList() {
        return contactsDao.listNames();
    }
}
```

```
public class ContactsDaoImpl implements ContactsDao{
    @Override public List<String> listNames() {
        throw new RuntimeException("I need a database");
    }
}
```


CDI alternatives

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

```
<alternatives>
  <stereotype>demo.framework.Mock</stereotype>
</alternatives>
```

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




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

    public List<Contact> listContacts() {
        return em.createQuery("select c From Contact c").getResultList();
    }
}
```

What about
JPA code?

Arquillian

Create micro deployments
using an API

Run tests in real containers



Arquillian example

```
@RunWith(Arquillian.class)
public class ContactDaoTest {
    @Inject private ContactDao contactdao;

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

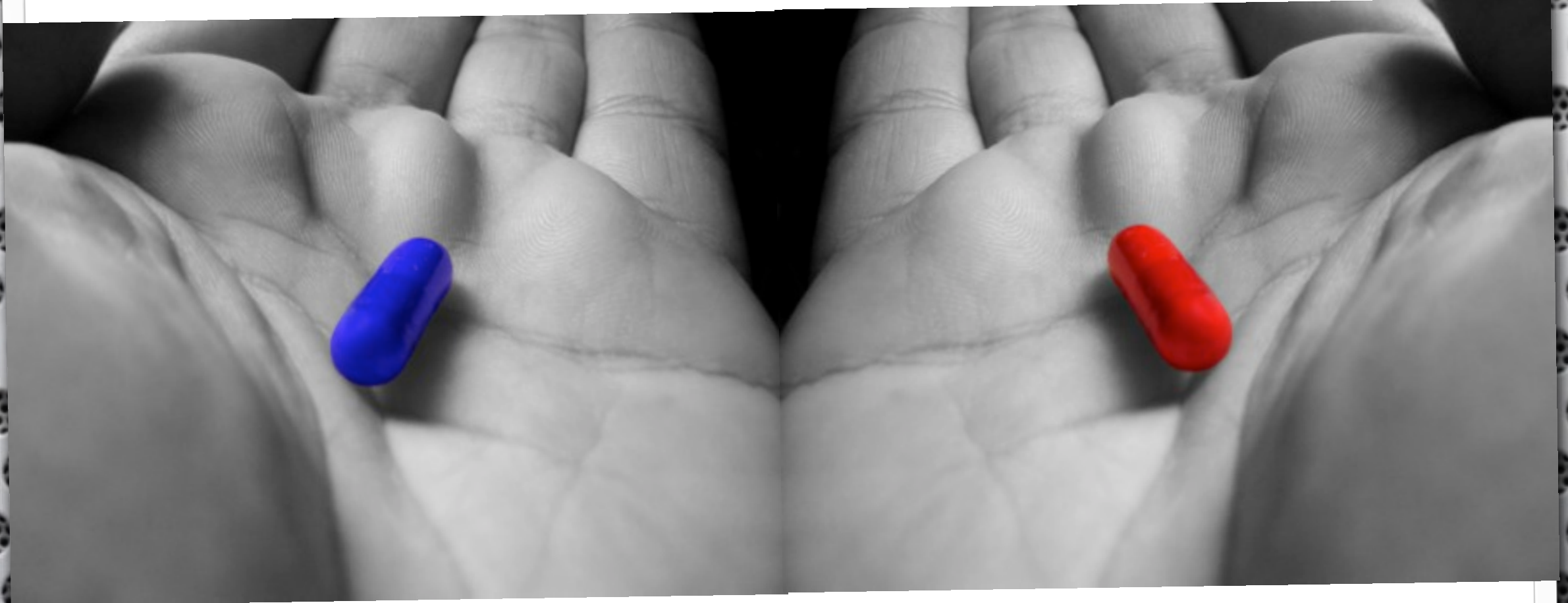
    @Test public void testIsDeployed() {
        Assert.assertNotNull(contactdao);
    }

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


wrap up



Is it all worth it?



blissful ignorance or painful reality ?!



Is there life after Spring?

- We believe that there is always room for innovation
 - Ideally through Open Source
- When it flies, bring it back to the spec!



There is more

Article series on jboss web site
with example Github projects:

- ❑ Encapsulating the Bean Container
- ❑ Step-by-step migration of the Spring Pet Clinic

COMING SOON

contact us:



Bert Ertman

bert.ertman@luminis.eu

Paul Bakker

paul.bakker@luminis.eu