RED HAT SUMMIT

LEARN. NETWORK. EXPERIENCE OPEN SOURCE.

June 11-14, 2013
Boston, MA

redhat.

Java EE 6 & Spring: A Lover's Quarrel

Derrick Kittler
Mauricio "Maltron" Leal
Vamsi Chemitiganti

Agenda

- The goal of this talk
- Evolution of Spring
- Evolution of Java EE 6
- Side-by-Side
- Migrating
- Java EE 6 and Spring Coexistence
- Conclusion

Fair Comparison

X

Orange

Apple

Fair Comparison: **NOT**

Java **EE 6**

Specification

X

Technology Framework

Fair Comparison

Java EE 6

X

Approach to a Enterprise Solution !!

If can’t beat them, join them !!!!

Java EE 6

Sometimes, a join solution can be the best approach

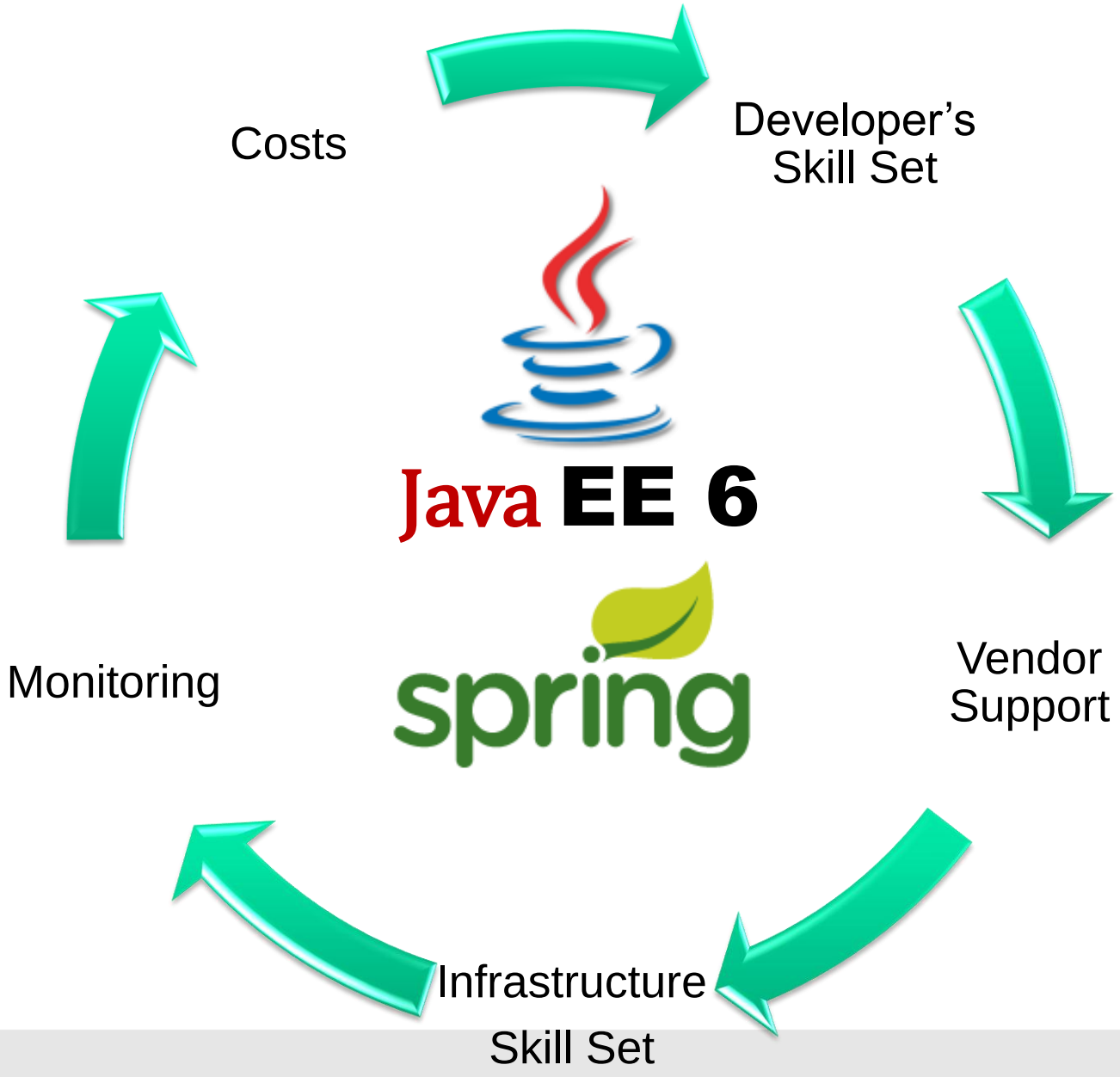
The goal of this talk is NOT…

Compare both technologies and declare a winner

The goal of this talk is…

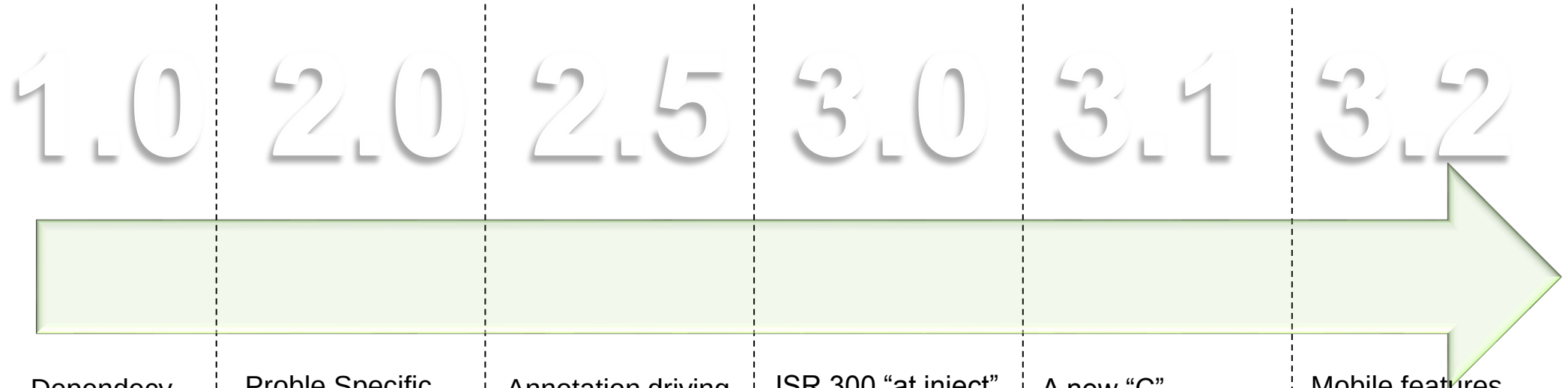
- How to deliver a good enterprise application
- Evaluate both technologies
- Show some migration path **(only when makes sense)**

Technology Choice: The big picture



EVOLUTION OF 

Evolution of Spring



Dependecy Injection
POJO oriented development
Declarative AOP & transactions
MVC Framework

Proble Specific XML
Extensible Configuration
Bean Scoping
Groovy, JRuby and BeanShell
JSP tag library
Java 5 autoboxing and generics

Annotation driving wiring
Automatic bean configuration
New annotation driven MVC framework
JUnit 4-based integration testing

JSR 300 “at inject”
New Spring Expression Language
First-class REST support
Java based configuration
Several new Spring MVC features
Support for JSR 303 declarative validation
Scheduled Jobs

A new “C” namespace
Configuration profiles
Unified property resolution
Java configuration features
Servlet 3.0 support
Declarative caching
Spring MVC enhancements

Mobile features
Modern Web
Data Access
Social
Security
Cloud Ready

What is Spring ?

- Explicit Configuration
- Aspect oriented
- Ease-of-use abstractions over lower-level, general purpose API's like JDBC, JMS and JavaMail
- More flexibility for fine grained control, but more complexity
- Deployment/Runtime complexity
- Uses a Java EE container underneath

Google App Engine

AWS Beanstalk

Heroku

Cloud Foundry

OpenShift

Tomcat 5+

GlassFish 2.1+

WebLogic 9+

WebSphere 6.1+

Spring Data

Redis | HBase | GemFire | JPA | QueryDSL

MongoDB | Neo4j | Soir | JDBC | Splunk

Spring for Apache Hadoop

HDFS | MapReduce | Hive

Pig | Cascading | SI/Batch

Spring Batch

Spring AMQP

Spring Web Flow

Spring Security

Spring Integration

Spring Roo

Spring Web Services

Spring Secure OAuth

Spring Social

Twitter | LinkedIn | FaceBook

Spring Framework

DI | AOP | TX | JMS | JDBC | ORM | OXM | Scheduling

MVC | REST | JMX | Testing | Caching | Profiles | Expression

Java EE 1.4+ / SES+

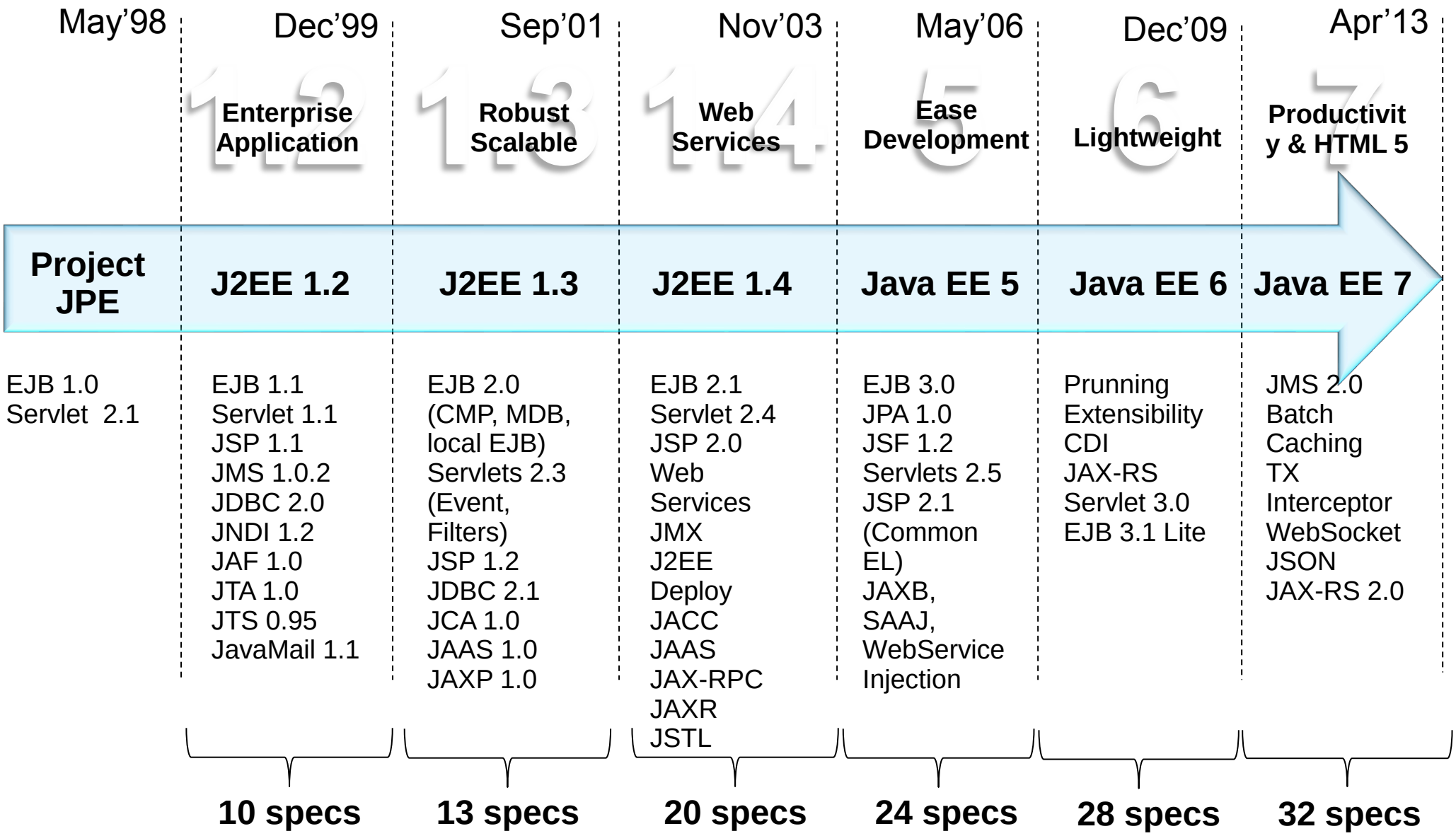
JPA 2.0 | JSF 2.0 | JSR 250 | JSR 330 | JSR 303 | JTA | JDBC 4.1 | JMX 1.0+

Spring JMS Configuration

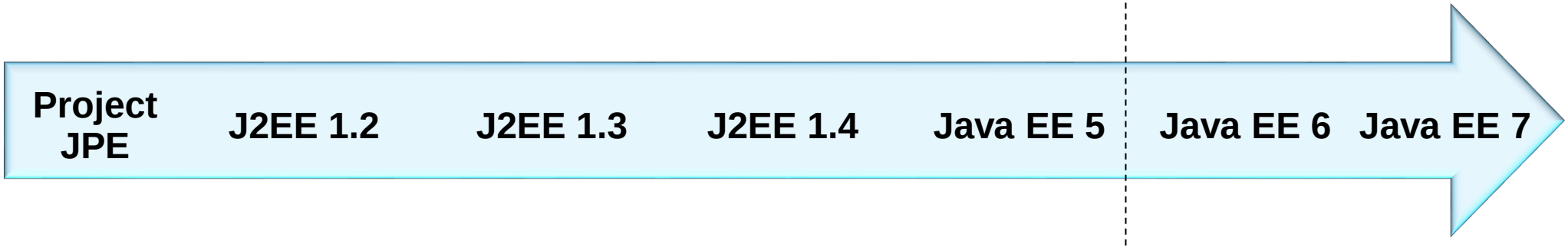
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:amq="http://activemq.apache.org/schema/core"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/jms
    http://www.springframework.org/schema/jms/spring-jms-3.1.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd">
  ...
  <amq:broker useJmx="false" persistent="false">
    <amq:transportConnectors>
      <amq:transportConnector uri="tcp://localhost:0" />
    </amq:transportConnectors>
  </amq:broker>
  <amq:queue id="jms/OrderQueue" physicalName="queue.OrderQueue"/>
  <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost"/>
  <jms:listener-container acknowledge="transacted" concurrency="3-5">
    <jms:listener destination="jms/OrderQueue" ref="orderProcessor"/>
  </jms:listener-container>
</beans>
```

EVOLUTION OF Java EE 6

Evolution of J2EE (later Java EE)



Lessons learned with Spring



- One monolitic server
- Hard to decouple technologies
- Heavyweight

- Lightweight and Heavyweight servers (separated by Profiles)
- Technologies doesn't depend solely on containers
- CDI is a gamer changer
- Arquilian is **ALSO** a gamer changer

Java EE 6 Highlights

Java EE 6

- Extensibility
- Enhanced easy of Development
- JSF 2.1
  - Built-In AJAX
  - Expanded View Model
- Profiles
- Vendor Support
- CDI (Contexts & Dependency Injection)

What is CDI anyway ?


```
MyInterface hello = new MyImplementation();
```

What I want ⟵ | ⟶ **How is going to work**

What is CDI anyway ?

CDI

```
@Inject
MyInterface hello;
```

Hollywood Principle:

“Don’t call us, we will call you”

- Less coding = More Productivity
- Server (Container) “injects” the other part **FOR YOU**
- Loose coupling
- Refactoring easy
- Easy Maintenance

Java EE 6

SIDE-BY-SIDE

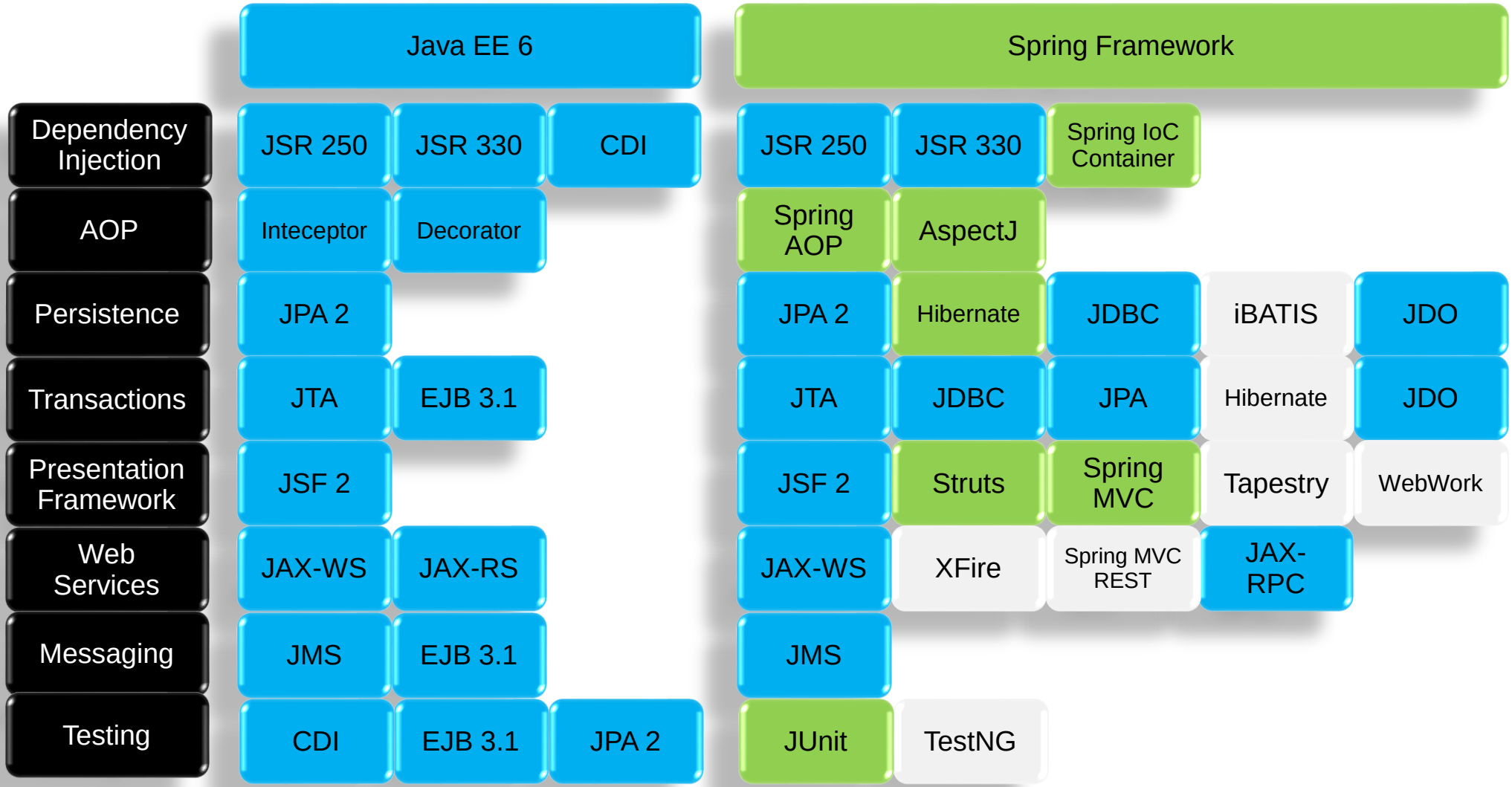
Side-by-Side

Java EE 6

spring

- Java EE 6 and Spring are both capable middleware solutions
- Functionally equivalent for the most part, with very different approaches
- For very vanilla applications, code is almost identical

Java EE vs Spring Framework Features



Spring Framework and Java EE 6 (Side-by-Side)

Java EE		Spring
JSF/CDI/JAAS/JPA/JAX-RS/JAX-WS	Components	Spring MVC/IoC/Security/JPA/JAX-WS
JBoss EAP 6	Server	Tomcat 7
11 Kb (WAR file)	File Size	32,153 Kb (WAR file)
0 / 4	JARs/ XMLs	36 / 5
37	Configuration Lines	92
33 Mb Heap / 48 Mb Perm	Pre-Deploy Memory	23 Mb Heap / 21 Mb Perm
41 Mb Heap / 84 Mb Perm	Post-Deploy Memory	107 Mb Heap/ 52 Mb Perm
71 Mb Heap / 92 Mb Perm	Memory (100 Threads)	81 Mb Heap/ 47 Mb Perm
2459 ms (average)	Response Time	1100 ms (average)

Should I migrate my current Spring application to Java EE 6 ?

Why ?

To migrate or not to migrate ?

- Application is a legacy (written in early versions of Spring)
- A real benefit will come from a migration
  - Faster
  - More secure
  - Easy to manage
- Support from multiple vendors

Java EE 6

MIGRATING

CDI and IoC


```
@Named
@RequestScoped
public  class  PersonBean {

    @Inject
    private SocialID social;
```


```
@Controller
@Scope("request")
public  class  PersonBean {

    @AutoWired
    private SocialID social;
```

Persistence


```
@Stateless
public  class  PersonDAO {

    @PersistenceContext
    private EntityManager em;
```


```
@Repository
public  class  PersonDAO {

    @PersistenceContext
    private EntityManager em;
```

Persistence


```
@Stateless
public  class  PersonDAO {

    @Resource
    private DataSource ds;

    public void save(Person p) {
        try {
        Connection c =
        ds.getConnection();
        PreparedStatement ps =
        c.prepareStatement("…
        ps.setString(1, p.name());
        ps.setInt(2, p.age());
        ps.execute();
        …
```


```
@Repository
public  class  PersonDAO {

    @Autowired
    private JdbcTemplate temp;

    public void save(Person p) {
        temp.update(
        "insert … values (?,?)",
        p.name(), p.age());
    }
}
```

Transaction and Exception


```
@TransactionAttribute(…)
@Stateless
public  class  PersonDAO {

    @PersistenceContext
    private EntityManager em;
```


```
@Transactional(…)
@Repository
public  class  PersonDAO {

    @PersistenceContext
    private EntityManager em;
```

Transaction and Exception


```
@ApplicationException(rollback=tr
ue)
public  class  DAOException
              extends Exception {
```


```
@Transactional(rollbackFor=DaoExc
eption.class)
public  class  PersonController {
```

Web


```
@Named
@RequestScoped
public  class  PersonMBean {

    private Person p;
    private List<Person> ls;

    @Inject
    private PersonDAO dao;

    public void save() {
        dao.save(p);
        resetForm();
    }
…
```


```
@Controller
@Transactional
public  class  PersonController {

    @Autowired
    private PersonDAO dao;

    @RequestMapping("/add")
public ModelAndView add(Person p)
{
    dao.save(p);
    ModelAndView mv = new
ModelAndView("ok");
    mv.addObject("person",
dao.list);
    return mv;
```

WebService (SOAP)


```
@WebService
public  class  PersonWS {

    @Inject
    private PersonDAO dao;

    public List<Person> list() {
        return dao.list();
    }
…
```


```
@WebService
public  class  PersonWS extends
SpringBeanAutowiringSupport {

    @Autowired
    private PersonDAO dao;

    public List<Person> list() {
        if(dao == null)
        injectionContext(this);

        …
…
```

WebService (REST)


```
@Path("/")
public  class  PersonResource {

    @Inject
    private PersonDAO dao;

    @GET
    @Path("/person/{id}")
    @Produces("application/json")
    public Person
load(@PathParam("id")Long ID)  {
        return dao.seek(ID);
    }
…
```


```
@Controller
public  class  PersonResource {

    @Autowired
    private PersonDAO dao;

    @RequestMapping(
    value="/person/{id}",
    produces={"application/json"}
    method=RequestMethod.GET)
    @ResponseBody
    public Person
load(@PathVariable("ID")Long ID)
{
    return dao.seek(ID);
…
```

Messaging Queue


```
InitialContext ic = …
ConnectionFactory cf = …
Connection c = …

Session s = c.createSession(…
TextMessage m = s.createText(
    "hello");
Queue q = (Queue)ic.lookup(…
QueueSender sender = …
Sender.send(m);

c.close();
```


```
@Autowired JmsTemplate t;
@Autowired Queue q;

public void sendMessage() {
    this.t.send(q, new
MessageCreator() {
    public Message
createMessage(Session s) throws
JMSException {
    return
s.createTextMessage("hello");
}});
}
```

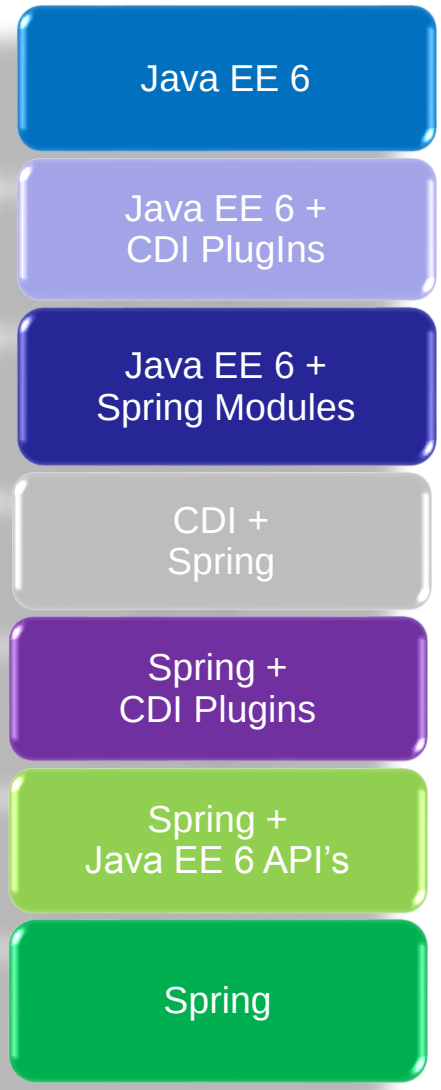
Java EE 6

COEXISTENCE

Java EE 6 and Spring coexistence

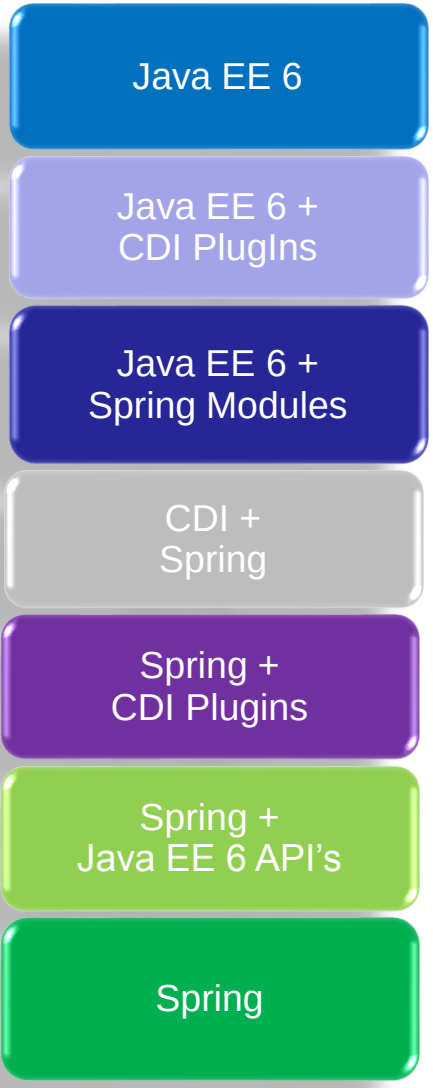
- Integration with Java EE API's
  - Spring beans can be injected into JSF Managed Beans
  - Spring beans can be referenced in EL with no JSF Backing beans
  - Spring JmsTemplate can be used on top of raw JMS API for convenience
  - Spring Listeners similar to EJB MDB especially JCA rather than JMS listeners
  - Hibernate validator standardized as Bean Validation (JSR 303)
  - Spring 3 supports excellent bi-directional integration with EJB's
  - CDI and Spring Integration through the Spring Bridge to CDI
- Native support for Java EE
  - Java EE 6 annotations supported by Spring
  - Spring can use JPA / Hibernate natively
- Application Server Integration
  - DataSources can use application Server QoS pooling, transactions, statement caching, debugging, monitoring and security

Java EE 6 and Spring coexistence



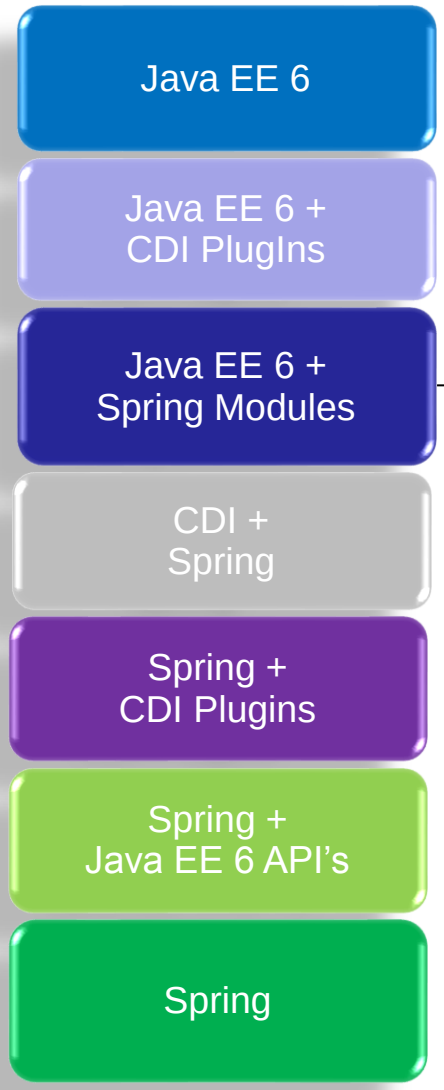
- Preference for Standardization
- Vendor Neutrality
- Simplicity
- XML Aversion
- Annotations
- Type-Safety
- Near Zero Configuration
- Streamlined Stack
- Early Adopters
- Some Risk Tolerance

Java EE 6 and Spring coexistence



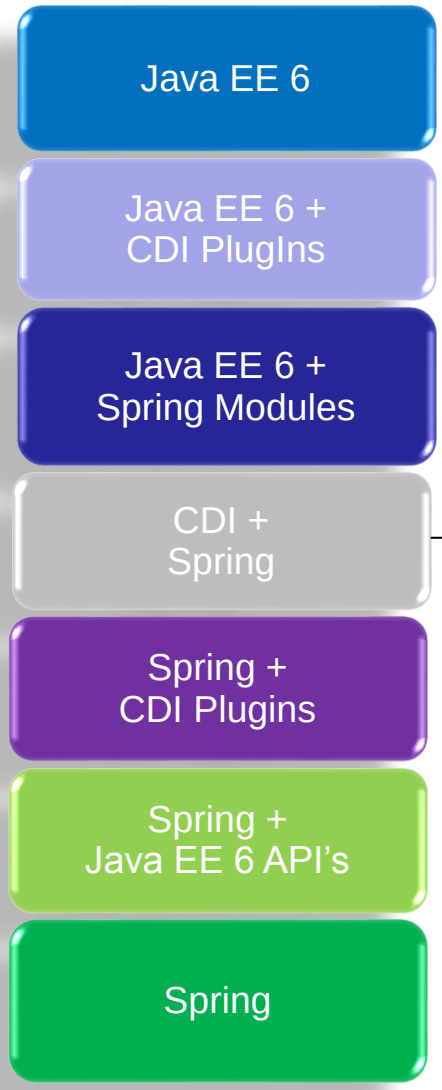
- Some specialized integration needs
- Interim technology updates
- Slight less vendor neutral, but
- Standards compatible

Java EE 6 and Spring coexistence



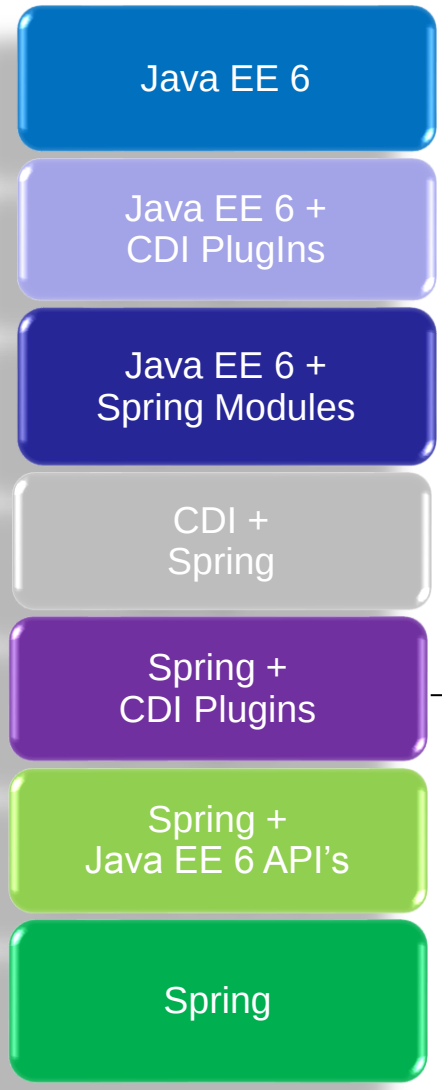
- Specialized integration needs
- Existing skill set
- Some sacrifices to vendor neutrality

Java EE 6 and Spring coexistence



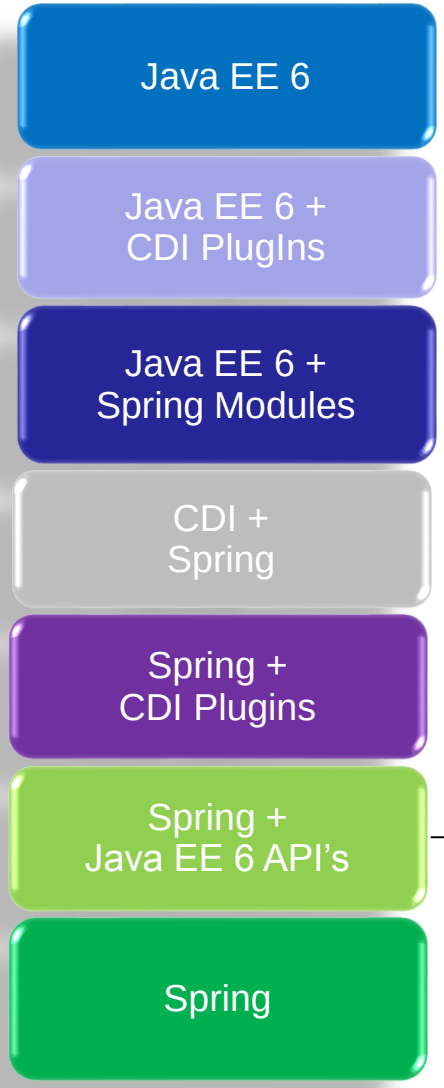
- Gradual migration
- Skill set transfer
- Technology agnosticism
- Highly advanced integration needs
- Aggressive technology adoption

Java EE 6 and Spring coexistence



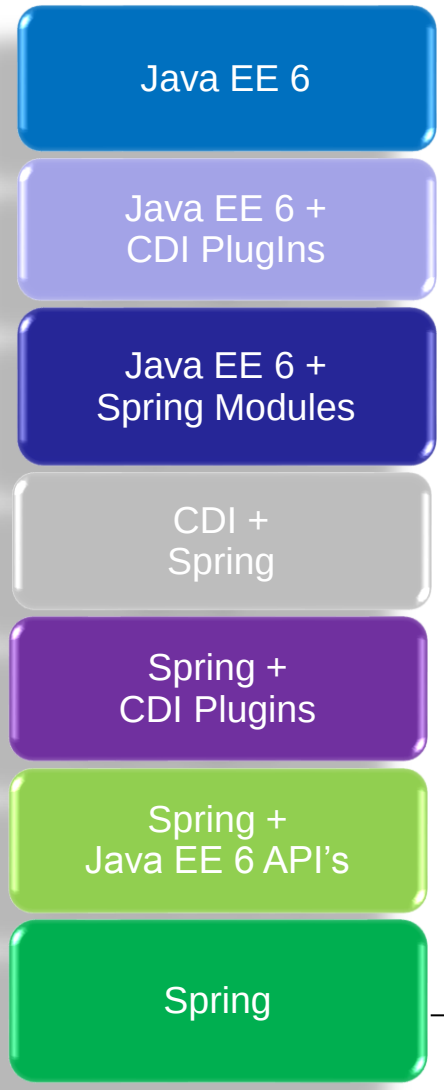
- Technology agnosticism
- Advanced integration needs
- Likely using JSF

Java EE 6 and Spring coexistence



- Existing skill set
- Low risk tolerance
- Standardization
- Technology agnosticism
- Uses Tomcat or more likely a Java EE Server
- Likely using JSF, JPA, JMS, JAX-RS maybe EJB

Java EE 6 and Spring coexistence



- Highly specialized custom system, likely on Tomcat
- Established skill set
- Wide deployment
- Open Source activism
- Some vendor lock-in risk

Java EE 6

CONCLUSION

Conclusion

Java EE 6

- There is no practical reason to choose between the two
- Balanced competition in server-side Java is good for both developers and vendors
- There is excellent integration possibilities

THANK YOU