Welcome to Enterprise Spring

A 4-day, hands-on course that teaches you how to use Spring to integrate business applications

© Copyright 2014 Pivotal. All rights reserved.

Enterprise Integration, defined

"The goal of Enterprise Integration is to connect multiple enterprise systems with heterogenous data formats to support business use cases"

Examples:

- Connect a new order processing system to a legacy billing system
- Consolidate two inventory systems after a company merger
- Import nightly batch of transaction data from credit card processor

© Copyright 2014 Pivotal. All rights reserved.

The Goal of Spring in Enterprise Integration

- Spring's goal is to provide comprehensive infrastructural support for enterprise integration
  - Spring handles the “plumbing” of integration
  - So you can focus on domain logic / business use cases
  -
- This course will cover multiple Spring projects
  - Spring Framework (Spring Core)
  - Spring Web Services
  - Spring Integration
  - Spring Batch

© Copyright 2014 Pivotal. All rights reserved.

Format and Materials

- Course is 50% theory, 50% lab work
  - Theory covers integration concepts and how to use Spring
  - Labs provide hands-on experience
- USB keys are yours to keep and contain
  - State-of-the-art lab environment based on
    - Spring Tool Suite (STS)
    - Apache Tomcat
    - Maven
  - Lab sources, documentation and dependencies

© Copyright 2014 Pivotal. All rights reserved.

Approach and Philosophy

- Focus on real-world problems and solutions
  - All labs are based on realistic integration use cases
- Testing matters
  - All labs are test-driven using JUnit and Mockito
- Keep it simple
  - Integration is inherently complex; this course will show you how to keep it as simple as possible

© Copyright 2014 Pivotal. All rights reserved.

Agenda: Day 1

- Introductions and Getting Started
- Integration Styles (presentation only)
- Tasks and Scheduling
- Remoting
- SOAP web services

© Copyright 2014 Pivotal. All rights reserved.

Agenda: Day 2

- Advanced Spring Web Services
- RESTful web services
- Introduction to Messaging (presentation only)
- Working with JMS
- Transactional JMS

© Copyright 2014 Pivotal. All rights reserved.

Agenda: Day 3

- Global transaction management (XA and JTA)
- Introduction to Spring Integration
- Configuring Spring Integration
- Spring Integration Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Agenda: Day 4

- Introduction to Spring Batch
- Restart and Recovery With Spring Batch
- Spring Batch Admin and Scaling Batch Jobs

© Copyright 2014 Pivotal. All rights reserved.

Prerequisites: Java

- Strong ability with language
  - Advanced syntax such as anonymous classes
- Understanding of Collections API
  - List, Set, Map, etc
- Annotations
  - Basic familiarity is OK
  - Helps to understand how to author your own
- Generics
  - Generics syntax will be used throughout the course

© Copyright 2014 Pivotal. All rights reserved.

Prerequisites: Concepts

- TDD (“Test-driven development”)
  - Understanding the value of developer testing
  - Experience with JUnit 4
- Dependency Injection / Inversion of Control
- POJO Programming

© Copyright 2014 Pivotal. All rights reserved.

Prerequisites: Spring

- Experience with <beans/> XML
- Familiarity with Spring XML namespaces
  - context:, tx:, aop:, etc.
- Familiarity with “annotation-driven” injection
  - @Autowired
  - <context:component-scan/>
- Familiarity with Spring's transaction management
  - @Transactional
  - <tx:annotation-driven/>

© Copyright 2014 Pivotal. All rights reserved.

Prerequisites: Tools

- Basic experience with Eclipse
  - Navigating views and perspectives
  - Using content-assist (ctrl+space)
  - Using quick-fix (ctrl+1)
  - Running code
    - on Server (via WTP)
    - as JUnit Test

© Copyright 2014 Pivotal. All rights reserved.

Prerequisites: Java EE

- Basic experience with Servlet container
  - Working with Tomcat
  - Configuring web.xml
    - <servlet/>, <listener/>, <filter/>, <context-param/>

© Copyright 2014 Pivotal. All rights reserved.

How to get the most value

- Ask questions!
  - During presentations
  - Especially during labs
- Team up
  - Consider working the labs with a partner

© Copyright 2014 Pivotal. All rights reserved.

Logistics

- Hours
- Lunch and breaks
- Restrooms, parking, etc.
- Other questions?

© Copyright 2014 Pivotal. All rights reserved.

Feedback!

- We value your input
  - Evaluation instructions provided at end of course
- Send any additional feedback or questions to education@pivotal.io

© Copyright 2014 Pivotal. All rights reserved.

Lab: ei-course-intro

Getting to know the reference domain and courseware environment

© Copyright 2014 Pivotal. All rights reserved.

Styles of Enterprise Integration

© Copyright 2014 Pivotal. All rights reserved.

Topics in this session

- Introduction
- Integration Styles
  - File Transfer
  - Shared Database
  - Remoting
  - Messaging
- For Each Style:
  - Pros/Cons
  - Spring Support

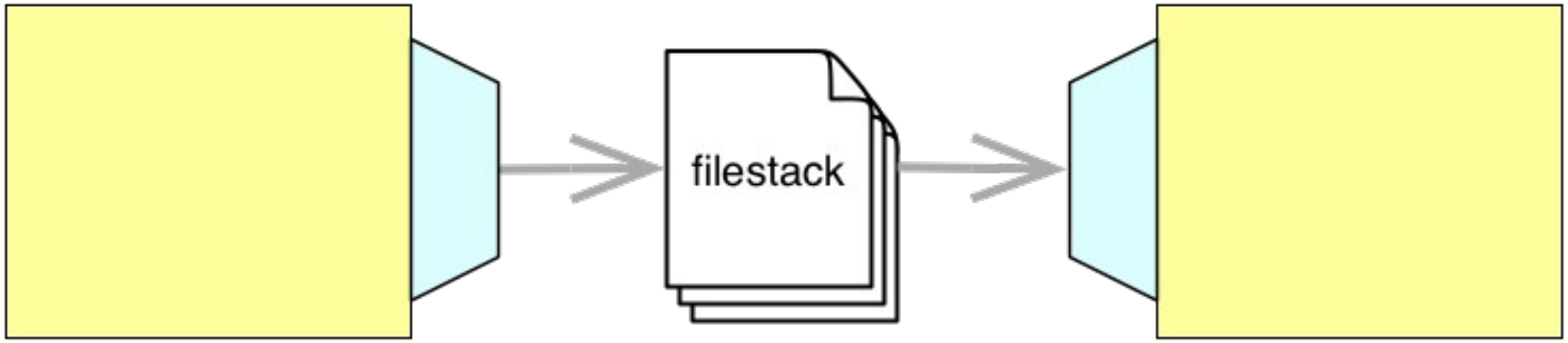
© Copyright 2014 Pivotal. All rights reserved.

Introduction

- Integrating Enterprise Applications can be done in many ways
- Each way has its own pros and cons
- Best solution depends on requirements
- Things to consider:
  - Coupling (logical, temporal)
  - Synchronous or asynchronous
  - Overhead
  - Data formats
  - Reliability

© Copyright 2014 Pivotal. All rights reserved.

File Transfer: *Overview*



© Copyright 2014 Pivotal. All rights reserved.

File Transfer: *Tradeoffs*

- Pros
  - Simple
  - Interoperable
  - Fast
- Cons
  - Unsafe
    - Non transactional
    - Concurrency issues
    - Security
  - Platform dependent
  - Not event-driven

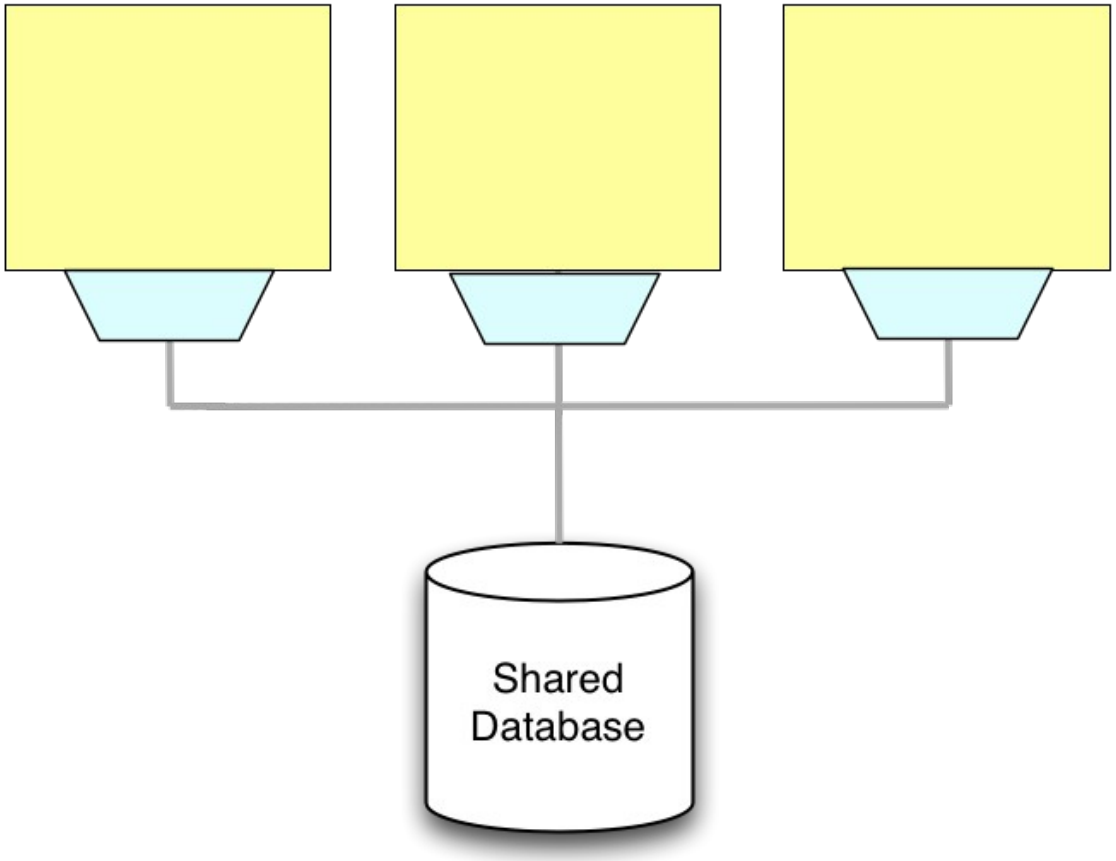
© Copyright 2014 Pivotal. All rights reserved.

File Transfer: *Spring Support*

- Resource API
  - Helps to simplify common I/O operations
- Spring Batch
  - Parsing of CSV- and XML-files
  - Stateful streaming of large files in and out of the app
- Spring Integration
  - Monitoring directories for new files
  - Working with (S)FTP and FTPS

© Copyright 2014 Pivotal. All rights reserved.

Shared Database: *Overview*



© Copyright 2014 Pivotal. All rights reserved.

Shared Database: *Tradeoffs*

- Pros
  - Simple
  - Transactional
  - Triggers
    - But non-portable
- Cons
  - Slower
  - Impedes schema evolution
    - Less so with NoSQL DBs

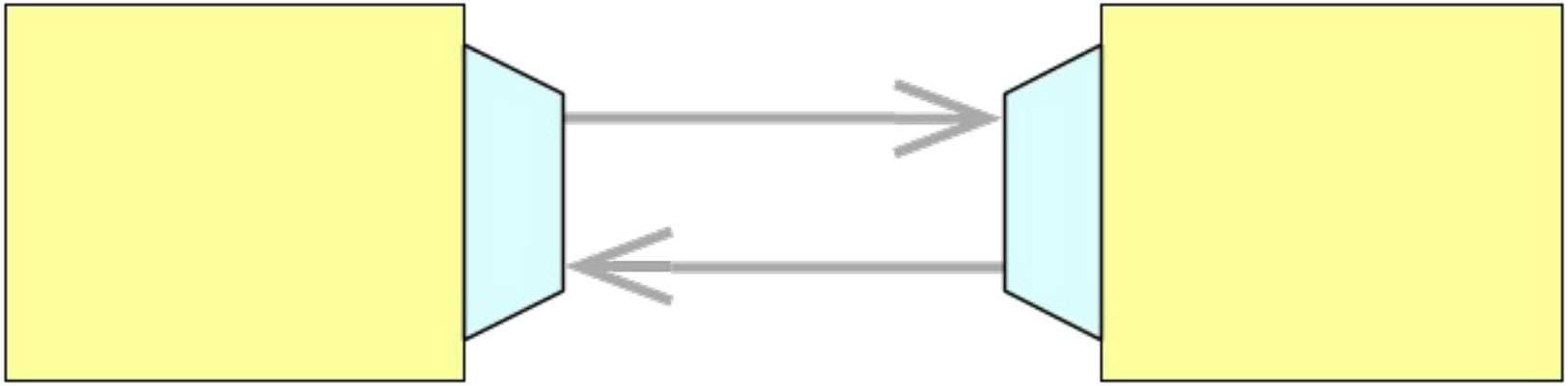
© Copyright 2014 Pivotal. All rights reserved.

Shared Database: *Spring Support*

- DataAccessException Hierarchy
- Transaction Management
- Spring JDBC
- Spring-ORM integration
  - Hibernate, JPA, iBatis
- Spring-Data umbrella project
  - Dynamic JPA repository generation
  - Support for various NoSQL solutions
    - MongoDB, Redis, Riak, Neo4J, etc.

© Copyright 2014 Pivotal. All rights reserved.

Remoting: *Overview*



© Copyright 2014 Pivotal. All rights reserved.

Remoting: *Tradeoffs*

- Pros
  - Convenient
  - Stick with OO paradigm
  - Speed
- Cons
  - Not interoperable
  - Hard to version
  - Not scalable
  - Hidden complexity

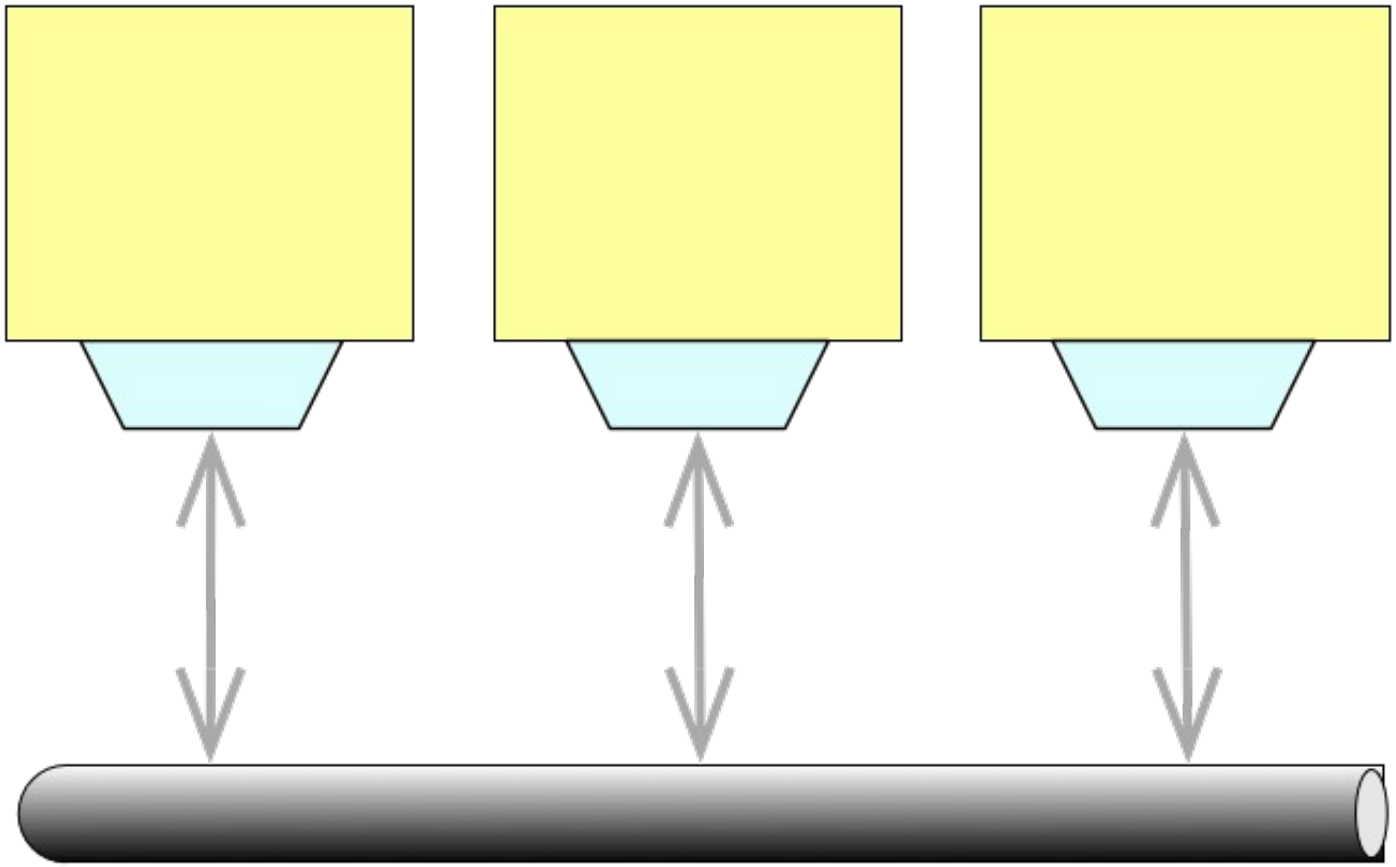
© Copyright 2014 Pivotal. All rights reserved.

Remoting: *Spring Support*

- *ProxyFactoryBean* and *Exporter* implementations for
  - RMI
  - HttpInvoker
  - Hessian / Burlap
  - JAX-RPC
  - JAX-WS
  - JMS

© Copyright 2014 Pivotal. All rights reserved.

Messaging: *Overview*



© Copyright 2014 Pivotal. All rights reserved.

Messaging: *Tradeoffs*

- Pros
  - Asynchronous
  - Efficient
  - Scalable
  - Extensible
- Cons
  - Complexity
  - Longer response times
  - Loss of Transaction context
  - Loss of Security context

© Copyright 2014 Pivotal. All rights reserved.

Messaging: *Spring Support*

- Spring JMS
  - Sending and receiving JMS messages
  - Transaction support
- Spring AMQP
  - AMQP messaging support, similar to JMS support
- Spring Web Services
  - Contract first web services
- Spring Integration
  - Supports Hohpe's *Enterprise Integration Patterns*
  - Builds on top of other Spring support for integration

© Copyright 2014 Pivotal. All rights reserved.

What about Web Services?

- Web services can be designed to exhibit characteristics of Remoting *or* Messaging
  - **Remoting**
    - RPC/literal SOAP messages
    - XML-RPC
    - Some of the WS-* standards
  - **Messaging**
    - Document/literal SOAP messages
    - POX, REST
    - “Contract-first” web services
  - This training focuses on loosely-coupled, contract-first, messaging-style web services

© Copyright 2014 Pivotal. All rights reserved.

Tasks and Scheduling

© Copyright 2014 Pivotal. All rights reserved.

Topics in this presentation

- **Introduction to concurrency**
- Java Concurrency support
- Spring's Task Scheduling support

© Copyright 2014 Pivotal. All rights reserved.

Concurrency

- Also known as parallel processing or Multithreading / multitasking
- Allows tasks to be run simultaneously
  - Useful when applications are I/O-bound
  - incl. servers serving multiple clients
  - Maximum advantage of multicore/-processor systems
- A built-in feature of the Java language and platform
  - Unlike, for example, C/C++

© Copyright 2014 Pivotal. All rights reserved.

Example: Simple Task

Download 100 photos from flickr.com

photos.txt

```
http://farm4.static.flickr.com/3267/2803733448_ea757aab49.jpg
http://farm3.static.flickr.com/2160/2616535176_ac7db42a78.jpg
http://farm4.static.flickr.com/3068/2545650273_6c1b07d4c3.jpg
http://farm4.static.flickr.com/3024/2530880977_ff6cfc5012.jpg
 ...
```


© Copyright 2014 Pivotal. All rights reserved.

Serial processing

```
public static void main(String[] args) throws IOException {
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(new FileInputStream("c:\\photos.txt")));

    String tmpDir = "c:\\temp";
    String photoUrl = null;
    while ((photoUrl = reader.readLine()) != null) {
        DownloadUtils.download(photoUrl, tmpDir);
    }

}
```

Too slow? Might be better to download all 100 photos *simultaneously*

© Copyright 2014 Pivotal. All rights reserved.

Task Scheduling

- Another form of concurrency
- Schedule piece of work to run once or repeatedly
- Examples: poll for new email every second, start nightly batch job every working day, etc.

© Copyright 2014 Pivotal. All rights reserved.

Topics in this presentation

- Introduction to concurrency
- **Java Concurrency support**
- Spring's Task Scheduling support

© Copyright 2014 Pivotal. All rights reserved.

Java Concurrency Support

- Thread is basic concurrency building block Java
- Executes instance of Runnable interface
- Multiple threads can run in parallel
- Using private data or accessing shared memory
- Sharing requires synchronization when data is changed

© Copyright 2014 Pivotal. All rights reserved.

Implement *java.lang.Runnable*

```
public class DownloadTask implements Runnable {
    private final String url;
    private final String tmpDir;
    public DownloadTask(String url, String tmpDir) {
        this.url = url;
        this.tmpDir = tmpDir;
    }
    @Override
    public void run() {
        try {
            DownloadUtils.download(url, tmpDir);
        } catch (IOException ex) { ex.printStackTrace(System.err); }
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Java Concurrency APIs

- Shouldn't typically use Threads directly
  - Thread management is low-level cross-cutting concern
- Java 5 and later provide higher-level language support
  - Transparently use thread pools, task queues, etc.
- All support is under *java.util.concurrent.**
  - Concurrent collections, atomic wrapper types, thread pools, task types, executor framework, etc.

© Copyright 2014 Pivotal. All rights reserved.

java.util.concurrent.Executor

- Simple abstraction for executing tasks
  - Hides the details of low-level thread management
  - Some implementations execute tasks in a separate thread
  - Other implementations execute serially

```
public interface Executor {
  void execute(Runnable task);
}
```

- Spring has similar *TaskExecutor* interface
  - Extends *java.util.concurrent.Executor*

© Copyright 2014 Pivotal. All rights reserved.

java.util.concurrent.ExecutorService

- Extends *Executor*, adding lifecycle methods
  - **shutdown()**: accept no more new tasks
  - **awaitTermination()**: block until all tasks are complete

```
public interface ExecutorService extends Executor {

    void shutdown();

    boolean awaitTermination(long amount, TimeUnit unit)
                                        throws InterruptedException;
    // … any other methods
}
```


© Copyright 2014 Pivotal. All rights reserved.

java.util.concurrent.Executors

- Provides factory methods for commonly used *ExecutorService* implementations

```
public class Executors {
    public static ExecutorService newFixedThreadPool(int size);
    public static ExecutorService newSingleThreadExecutor();
    public static ExecutorService newCachedThreadPool();
    // ...
}
```


© Copyright 2014 Pivotal. All rights reserved.

Putting it all Together

```
public class PhotoDownloader {
    public PhotoDownloader(List<String> urls, String tmpDir) { … }
    public void download() throws InterruptedException {
        ExecutorService service = getExecutorService();
        for (String photoUrl : urls) {
            service.execute(new DownloadTask(photoUrl, tmpDir));
        }
        service.shutdown();
        service.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
    }
    private ExecutorService getExecutorService() {
        return Executors.newCachedThreadPool();
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Callable

- For tasks that return a result and/or throw exceptions, Java 5 introduced *Callable*

```
public class NextPrimeNumberFinder implements Callable<Integer> {
    private int number;

    public NextPrimeNumberFinder(int number) { this.number = number; }

    @Override
    public Integer call() throws Exception {
        for (;;)
            if (isPrime(++number)) return number;
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Future

- *ExecutorService* can schedule *Callable*, return a ***Future***
  - Placeholder that contains result once it's available
  - Allows to check if result is there, cancel the task, or block and wait (with optional timeout) for the result

```
Future<Integer> futurePrime =
    executorService.submit(new NextPrimeNumberFinder(10000));
// do some other work until we need the result...
// now block and wait for the result
try {
    Integer nextPrime = futurePrime.get();
} catch (ExecutionException e) {
    System.err.println("NextPrimeNumberFinder threw exception: " + e.getCause());
}
```


© Copyright 2014 Pivotal. All rights reserved.

ScheduledExecutorService / ScheduledFuture

- *ScheduledExecutorService* provides delay & repetition.
  - Use ScheduledFuture to obtain results.

```
ScheduledExecutorService scheduledExecutorService =
    new ScheduledThreadPoolExecutor(10);

// start the finding of the next prime in 10 seconds
ScheduledFuture<Integer> scheduledPrime =
    scheduledExecutorService.schedule(
        new NextPrimeNumberFinder(1000), 10, TimeUnit.SECONDS);

// run the download task repeatedly every second with initial delay of 0
scheduledExecutorService.scheduleWithFixedDelay(
    new DownloadTask(photoUrl, tmpDir), 0, 1, TimeUnit.SECONDS);
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this presentation

- Introduction to concurrency
- Java Concurrency support
- **Spring's Task Scheduling support**

© Copyright 2014 Pivotal. All rights reserved.

<task:> namespace

- A simple way to define TaskExecutors & Schedulers

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://www.springframework.org/schema/beans"
   xmlns:task="http://www.springframework.org/schema/task"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/task
      http://www.springframework.org/schema/task/spring-task-4.0.xsd">

   <!--  add <task:> elements -->
</beans>
```

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/scheduling.html

© Copyright 2014 Pivotal. All rights reserved.

<task:executor />

- Defines an *Executor* instance
  - Specifically, a Spring *ThreadPoolTaskExecutor*.

```
<task:executor id="exec" … />
```

  - Dependency inject wherever an *Executor* is needed:

```
public class PhotoDownloader {
    @Autowired Executor executor;
    …
    ...
    for (String photoUrl : urls) {
        executor.execute(new DownloadTask(photoUrl, tmpDir));
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

<task:executor/> examples

```
<task:executor id="exec" pool-size="5"/>
```

- Defines a thread pool, up to 5 threads

```
<task:executor id="exec"  pool-size="5-10"  queue-capacity="50" />
```

- Create up to 5 threads for new tasks
- Queue tasks when all threads used
- When queue full, allocate up to 10 threads max
- Reject tasks when capacity reached

```
<task:executor id="exec" pool-size="5-10" queue-capacity="50"
                    rejection-policy="CALLER_RUNS" />
```

- Run tasks in caller's thread instead of rejecting.

© Copyright 2014 Pivotal. All rights reserved.

<task:scheduled-tasks/>

- Allows scheduled method execution:

```
<task:scheduled-tasks>
   <task:scheduled ref="photoDownloader" method="download"
                   fixed-delay="1000"/>
   <task:scheduled ref="photoDownloader" method="download"
                   cron="*/5 * 9-17 * * MON-FRI"/>
</task:scheduled-tasks>

<bean id="photoDownloader">
   <!-- constructor args: urls, tmpDir -->
</bean>
```

Calls the `download()` method every second

Every 5 seconds, 9 to 5, weekdays

Warning: *scheduled-tasks* uses single-threaded Executor by default...

© Copyright 2014 Pivotal. All rights reserved.

<task:scheduler/>

- Defines an *Executor* instance
  - Specifically, a Spring *ThreadPoolTaskScheduler*.

```
<task:scheduler id="scheduler" pool-size="5"/>
```

  - Dependency inject wherever *TaskScheduler* is needed.

```
<task:scheduled-tasks scheduler="scheduler">
   <task:scheduled …  />
   <task:scheduled …  />
</task:scheduled-tasks>
```

Provides a source for threads to run scheduled tasks.

© Copyright 2014 Pivotal. All rights reserved.

Annotation-based Configuration

- Can also annotate methods instead of using XML

```
<task:scheduler id="scheduler" pool-size="5"/>
<task:annotation-driven scheduler="scheduler"/>
...
```

```
public class PhotoDownloader {
    public PhotoDownloader(List<String> urls, String tmpDir) { … }

    @Scheduled(cron="*/5 * 9-17 * * MON-FRI")
    public void download() {
        ...
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

@Async

- *@Async* annotated methods will run asynchronously (in a separate thread)
  - So they return immediately!
- Have to return *void* or a *Future*
  - In *Future* case, just wrap result in *AsyncResult* to keep compiler happy

```
@Async
public Future<File> download(String url, File directory) {
    File file = DownloadUtils.downloadTo(url, directory);
    return new AsyncResult(file);
}
```


© Copyright 2014 Pivotal. All rights reserved.

Running @Async Methods

- By default *SimpleAsyncTaskExecutor* is used
  - Creates new Thread for every task
- Override using the executor attribute

```
<task:annotation-driven executor="executor"/>

<task:executor id="executor" ... />
```

© Copyright 2014 Pivotal. All rights reserved.

Application Server Integration

- Java EE apps should not create their own threads
- Integrate with app-server managed threads through CommonJ (WebSphere & WebLogic)
  - *org.sfw.scheduling.commonj.WorkManagerTaskExecutor*
- Or through JCA
  - *org.sfw.jca.work.glassfish.GlassFishWorkManagerTaskExecutor*
  - *org.sfw.jca.work.jboss.JBossWorkManagerTaskExecutor*
  - *org.sfw.jca.work.WorkManagerTaskExecutor* for other app servers
- Configuration done in app server, not in Spring

© Copyright 2014 Pivotal. All rights reserved.

Lab

tasks-1

© Copyright 2014 Pivotal. All rights reserved.

Introduction to Spring Remoting

Simplifying Distributed Applications

© Copyright 2014 Pivotal. All rights reserved.

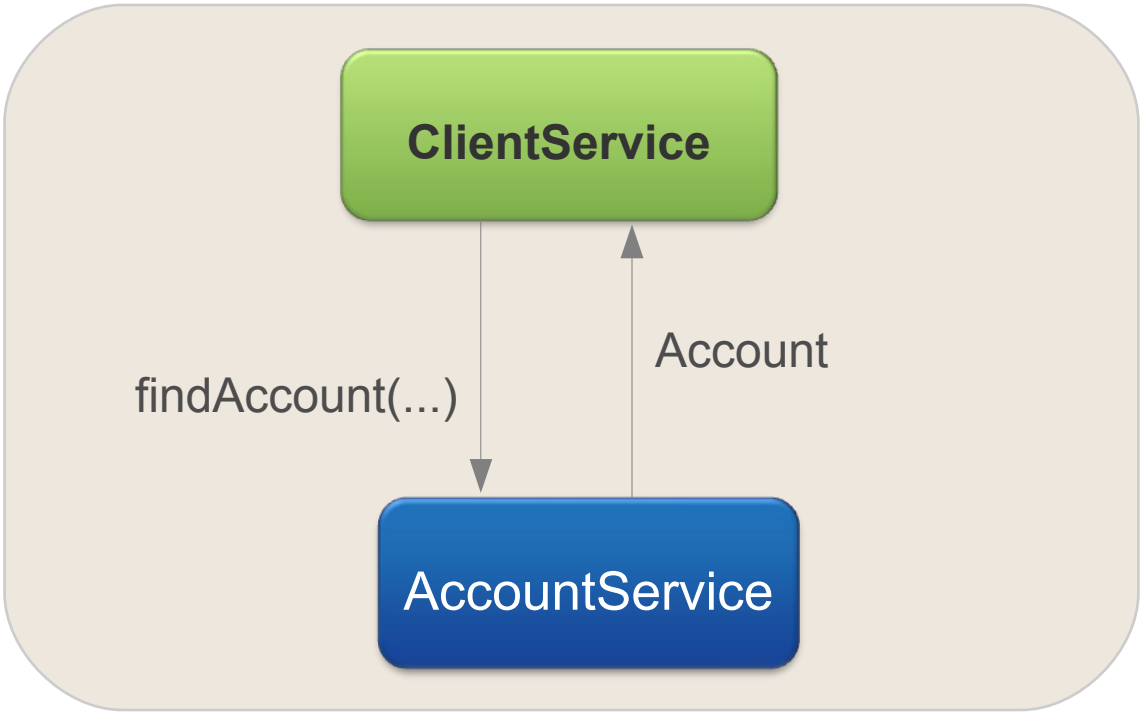
Topics in this Session

- **Introduction to Remoting**
- Spring Remoting Overview
- Spring Remoting and RMI
- HttpInvoker
- Additional supported Protocols

© Copyright 2014 Pivotal. All rights reserved.

Local access

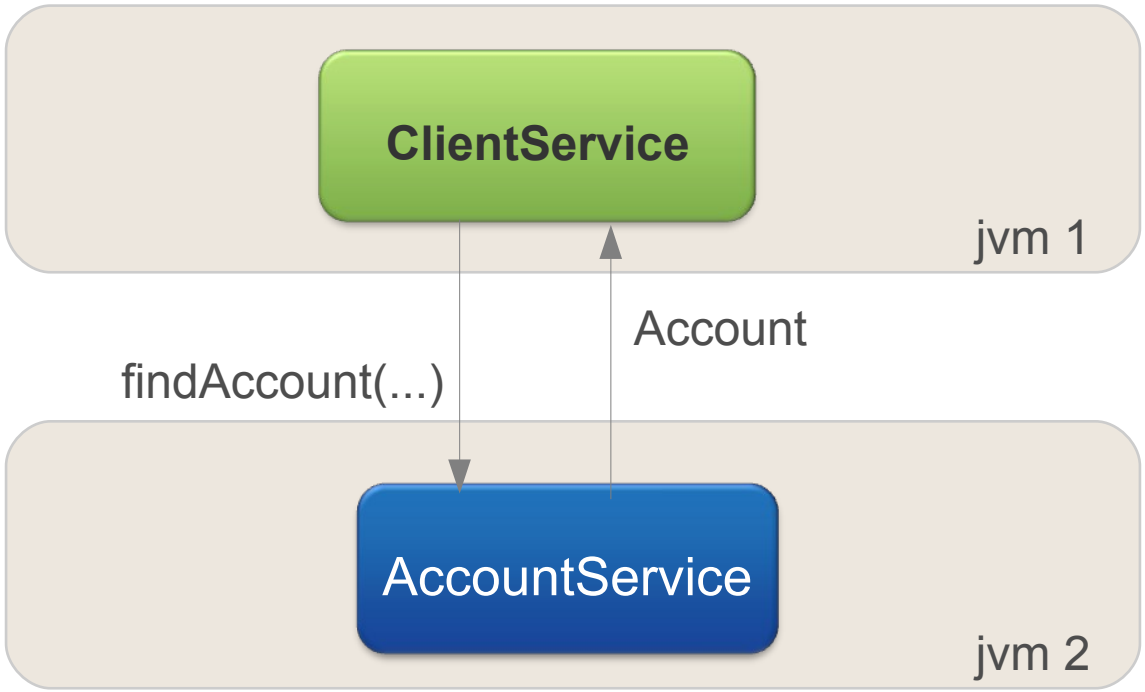
- So far, you have seen how to access objects locally



© Copyright 2014 Pivotal. All rights reserved.

Remote access

- What if those 2 objects run in some separate JVMs?



In this module, only Java-to-Java communication is addressed (as opposed to remote access using Web Services or JMS)

© Copyright 2014 Pivotal. All rights reserved.

The RMI Protocol

- Standard Java remoting protocol
  - Remote Method Invocation
  - Java's version of RPC
- Server-side exposes a *skeleton*
- Client-side invokes methods on a *stub* (proxy)
- Java serialization is used for marshalling

© Copyright 2014 Pivotal. All rights reserved.

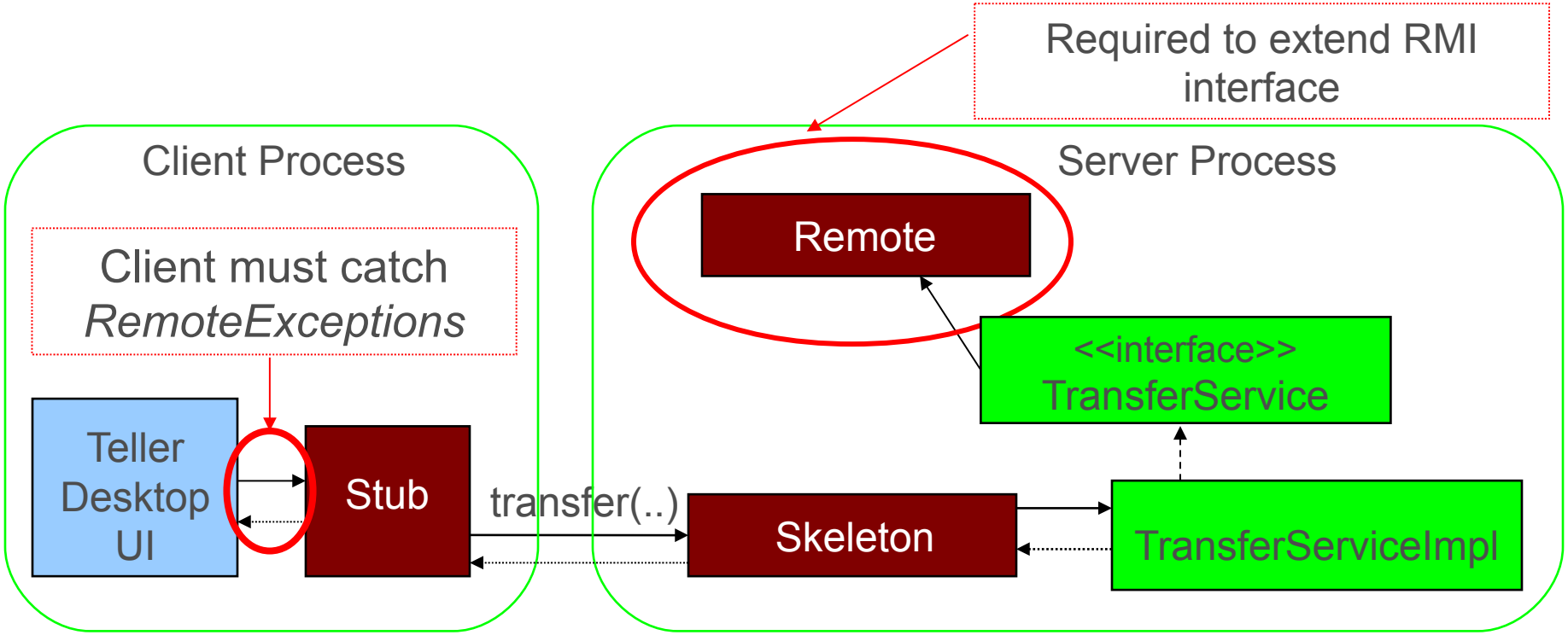
Working with plain RMI

- RMI violates separation of concerns
  - Couples business logic to remoting infrastructure
- For example, with RMI:
  - Service interface extends **Remote**
  - Client must catch **RemoteExceptions**
- Technical Java code needed for binding and retrieving objects on the RMI server

© Copyright 2014 Pivotal. All rights reserved.

Traditional RMI

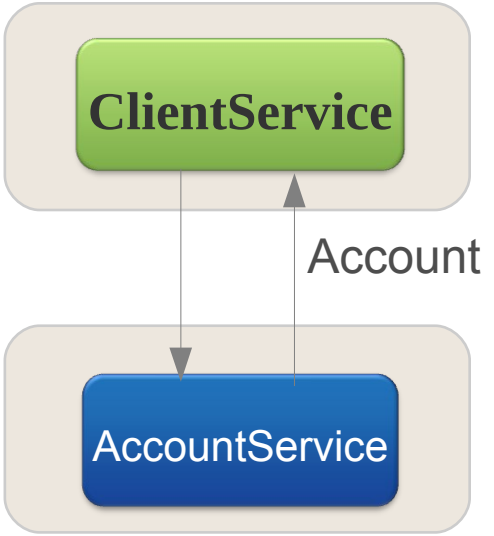
- The RMI model is invasive - server *and* client code is coupled to the framework



© Copyright 2014 Pivotal. All rights reserved.

RMI and Serialization

- RMI relies on Object Serialization
  - Objects transferred using RMI should implement interface *Serializable*
  - Marker interface, no method to be implemented




```
import java.io.Serializable;

public class Account implements Serializable {
   private static final long
         serialVersionUID = 9123849272938193534L;
   ...
}
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Remoting
- **Spring Remoting Overview**
- Spring Remoting and RMI
- HttpInvoker
- Additional supported Protocols

© Copyright 2014 Pivotal. All rights reserved.

Goals of Spring Remoting

- Hide “plumbing” code
- Configure and expose services declaratively
- Support multiple protocols in a consistent way

© Copyright 2014 Pivotal. All rights reserved.

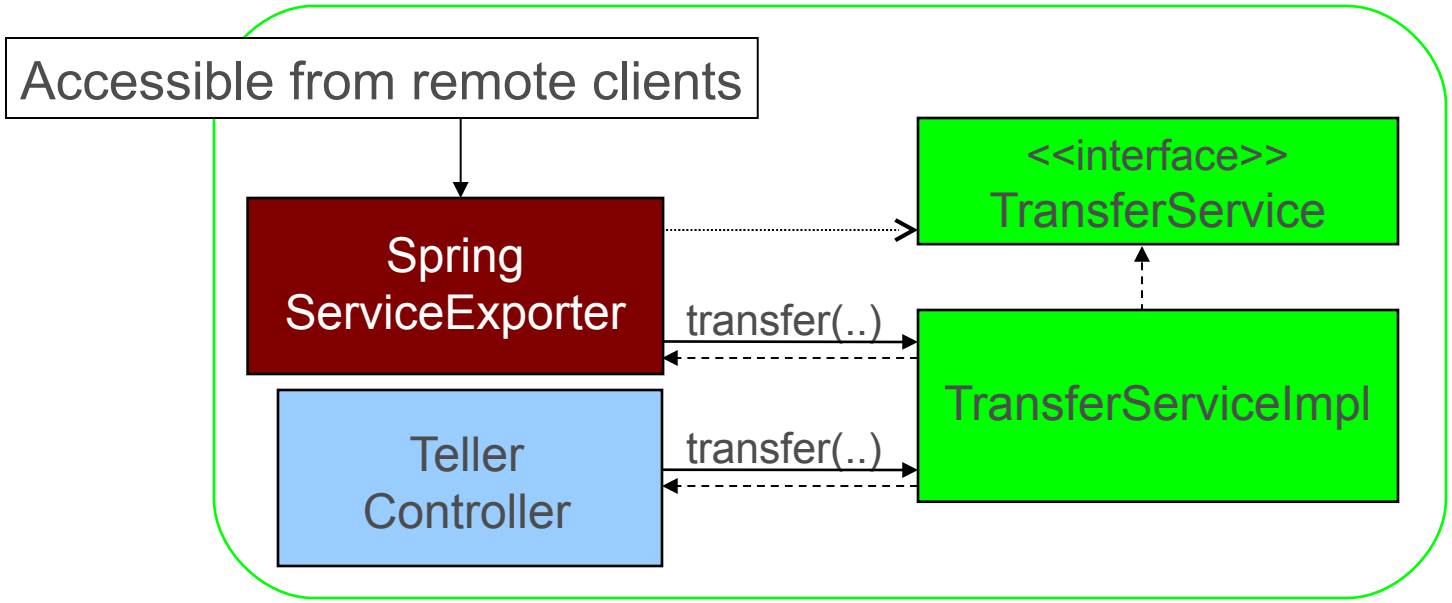
Hide the Plumbing

- Spring provides **exporters** to handle server-side requirements
  - Binding to registry or exposing an endpoint
  - Conforming to a programming model if necessary
- Spring provides FactoryBeans that generate **proxies** to handle client-side requirements
  - Communicate with the server-side endpoint
  - Convert remote exceptions to a runtime hierarchy

© Copyright 2014 Pivotal. All rights reserved.

Service Exporters

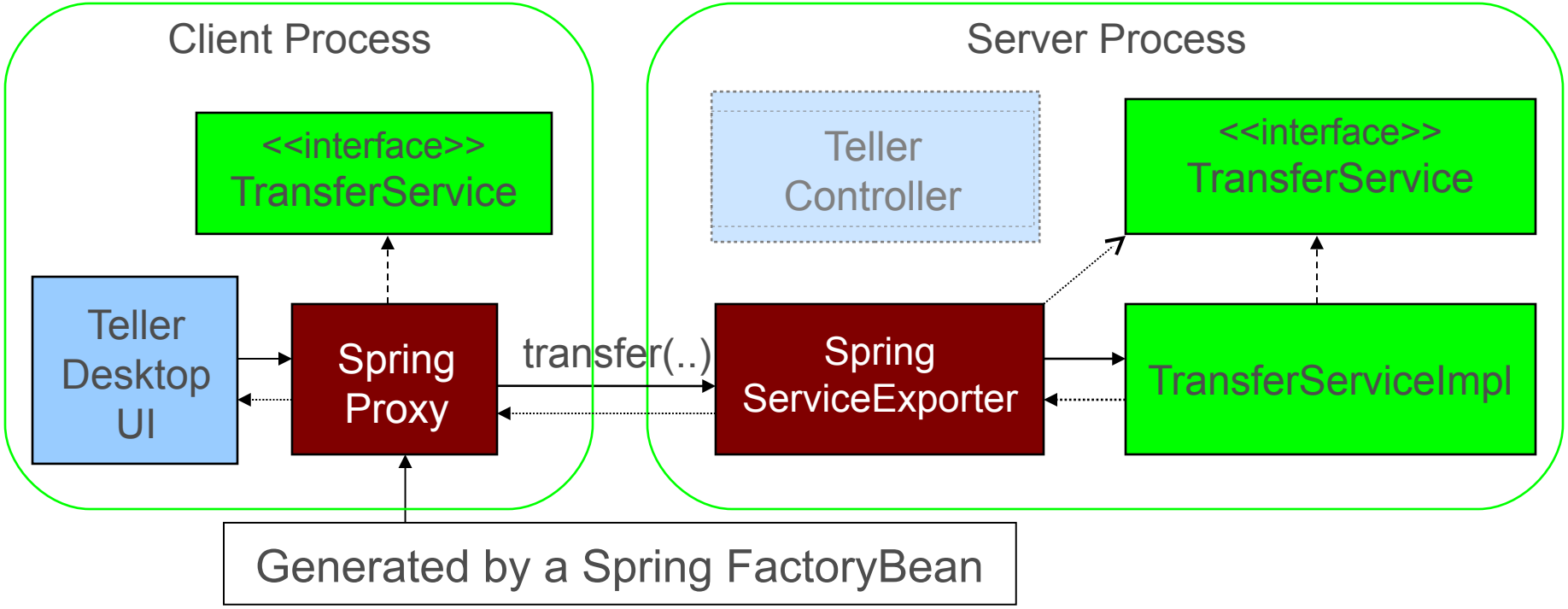
- Spring provides service exporters to enable declarative exposing of existing services



© Copyright 2014 Pivotal. All rights reserved.

Client Proxies

- Dynamic proxies generated by Spring communicate with the service exporter



© Copyright 2014 Pivotal. All rights reserved.

A Declarative Approach

- Uses a configuration-based approach
  - No code to write
- On the server side
  - Expose existing services with NO code changes
- On the client side
  - Invoke remote methods from existing code
  - Take advantage of polymorphism by using dependency injection
  - Migrate between remote vs. local deployments

© Copyright 2014 Pivotal. All rights reserved.

Consistency across Protocols

- Spring's exporters and proxy FactoryBeans bring the same approach to *multiple* protocols
  - Provides flexibility
  - Promotes ease of adoption
- On the server side
  - Expose a single service over multiple protocols
- On the client side
  - Switch easily between protocols

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Remoting
- Spring Remoting Overview
- **Spring Remoting and RMI**
- HttpInvoker
- Additional supported Protocols

© Copyright 2014 Pivotal. All rights reserved.

Spring's RMI Service Exporter

- Transparently expose an existing POJO service to the RMI registry
  - No need to write the binding code
- Avoid traditional RMI requirements
  - Service interface does not extend **Remote**
  - Service class is a POJO

Transferred objects still need to implement the interface *java.io.Serializable*

© Copyright 2014 Pivotal. All rights reserved.

Configuring the RMI Service Exporter

Server

- Start with an existing POJO service

```
<bean id="transferService" class="app.impl.TransferServiceImpl">
   <property name="accountRepository" ref="accountRepository"/>
</bean>
```

- Define a bean to export it

```
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
   <property name="serviceName" value="transferService"/>
   <property name="serviceInterface" value="app.TransferService"/>
   <property name="service" ref="transferService"/>
   <property name="registryPort" value="1096"/>
</bean>
```

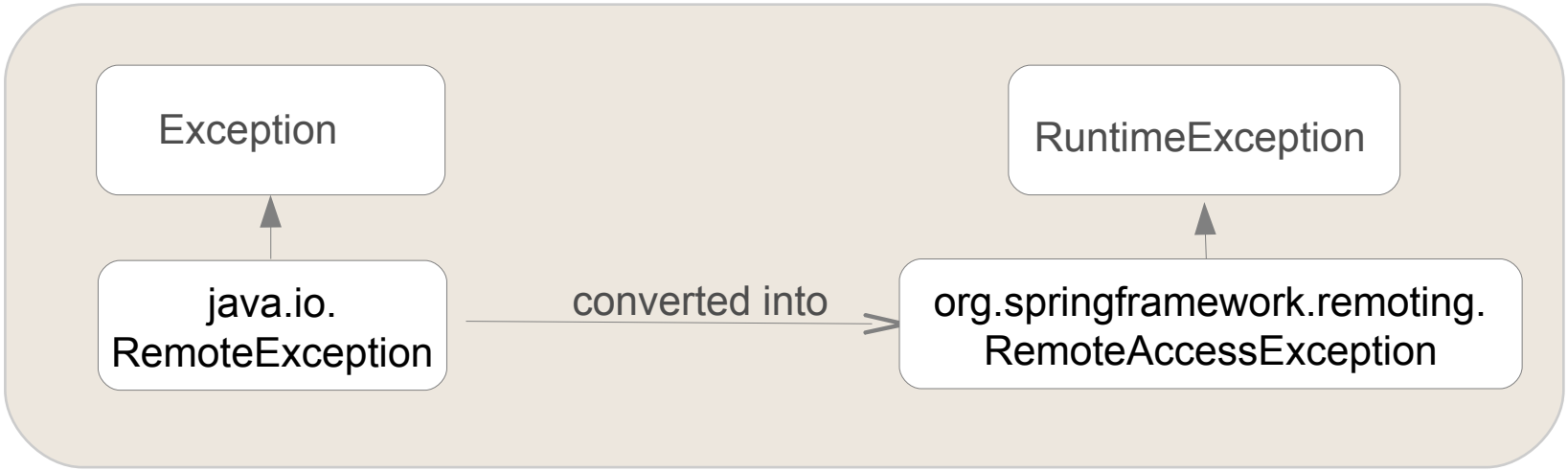
Binds to rmiRegistry as "transferService"

"*registryPort*" defaults to "1099"

© Copyright 2014 Pivotal. All rights reserved.

Spring's RMI Proxy Generator

- Spring provides FactoryBean implementation that generates RMI client-side proxy
- Simpler to use than traditional RMI stub
  - Converts checked RemoteExceptions into Spring's *runtime* hierarchy of RemoteAccessExceptions
  - Dynamically implements the business interface



© Copyright 2014 Pivotal. All rights reserved.

Configuring the RMI Proxy

Client

- Define a factory bean to generate the proxy

```
<bean id="transferService"
    class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
  <property name="serviceInterface" value="app.TransferService"/>
  <property name="serviceUrl" value="rmi://foo:1099/transferService"/>
</bean>
```

- Inject it into the client

```
<bean id="tellerDesktopUI" class="app.TellerDesktopUI">
  <property name="transferService" ref="transferService"/>
</bean>
```

TellerDesktopUI only depends on the TransferService interface

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Remoting
- Spring Remoting Overview
- Spring Remoting and RMI
- **HttpInvoker**
- Additional supported Protocols

© Copyright 2014 Pivotal. All rights reserved.

Spring's HttpInvoker

- Lightweight HTTP-based remoting protocol
  - Method invocation converted to HTTP POST
  - Method result returned as an HTTP response
  - Method parameters and return values marshalled with standard Java serialization

HttpInvoker is using serialization → transferred objects need to implement the interface *java.io.Serializable*

© Copyright 2014 Pivotal. All rights reserved.

Configuring the HttpInvoker Service Exporter (1)

Server

- Start with an existing POJO service

```
<bean id="transferService" class="app.impl.TransferServiceImpl">
   <property name="accountRepository" ref="accountRepository"/>
</bean>
```

- Define a bean to export it

```
<bean  id="/transfer"
       class="org.springframework.remoting.httpinvoker.
              HttpInvokerServiceExporter">
   <property name="serviceInterface" value="app.TransferService"/>
   <property name="service" ref="transferService"/>
</bean>
```

endpoint for HTTP request handling

Pre Spring 3.1: *Spring* did not allow characters such as '/' in an *id* attribute. If special characters are needed, the *name* attribute could be used instead.

© Copyright 2014 Pivotal. All rights reserved.

Configuring the HttpInvoker Service Exporter (2a)

- Expose the HTTP service via DispatcherServlet
  - Uses BeanNameUrlHandlerMapping to map requests to service
  - Can expose *multiple* exporters

```
<servlet>
  <servlet-name>transfer</servlet-name>
  <servlet-class>org.sfw...DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>transfer</servlet-name>
  <url-pattern>/services/</url-pattern>
</servlet-mapping>
```

http://foo:8080/services/transfer

service exporter bean name

servlet context

web.xml

© Copyright 2014 Pivotal. All rights reserved.

Configuring the HttpInvoker Service Exporter (2b)

- Alternatively define HttpRequestHandlerServlet
  - Doesn't require Spring-MVC
  - Service exporter bean is defined in root context
  - No handler mapping – servlet can only be used by *one* HttpInvoker exporter
  - Servlet name *must* match bean name

http://foo:8080/services/transfer

```
<servlet>
  <servlet-name>transfer</servlet-name>
  <servlet-class>org.sfw...HttpRequestHandlerServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>transfer</servlet-name>
  <url-pattern>/services/transfer</url-pattern>
</servlet-mapping>
```

service exporter bean name

web.xml

© Copyright 2014 Pivotal. All rights reserved.

Configuring the HttpInvoker Proxy

Client

- Define a factory bean to generate the proxy

```
<bean id="transferService"
        class="org.springframework.remoting.httpinvoker.
                HttpInvokerProxyFactoryBean">
  <property name="serviceInterface" value="app.TransferService"/>
  <property name="serviceUrl" value="http://foo:8080/services/transfer"/>
</bean>
```

HTTP POST requests will be sent to this URL

- Inject it into the client

```
<bean id="tellerDesktopUI" class="app.TellerDesktopUI">
  <property name="transferService" ref="transferService"/>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Remoting: Pros and Cons

- Advantages
  - (Too) Simple to setup and use
  - Abstracts away all messaging concerns
- Disadvantages
  - Client-server tightly coupled by interface
    - hard to maintain, especially with lots of clients
  - Can't control underlying messaging (hidden)
  - Difficult to scale
    - Make your requests *high-level*
  - Not interoperable, Java only

© Copyright 2014 Pivotal. All rights reserved.

Additional supported Protocols

- Hessian/Burlap (Cross Platform support)
- JAX-RPC (J2EE 1.4 standard for Web Services)
- JAX-WS (Java EE 5 standard for Web Services)
- JMS (basic JMS support)

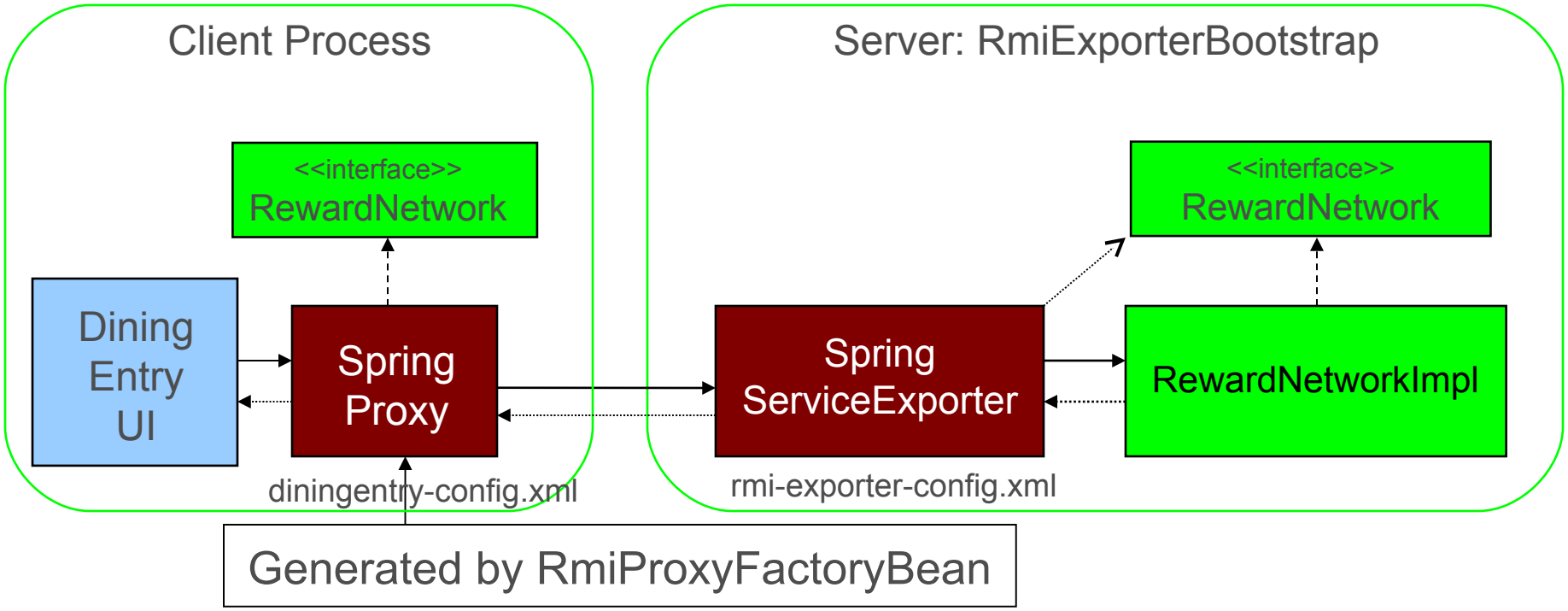
© Copyright 2014 Pivotal. All rights reserved.

Lab

Simplifying Distributed Applications with Spring Remoting

© Copyright 2014 Pivotal. All rights reserved.

Remoting Lab



© Copyright 2014 Pivotal. All rights reserved.

Spring Web Services

Implementing Loosely Coupled Communication with Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

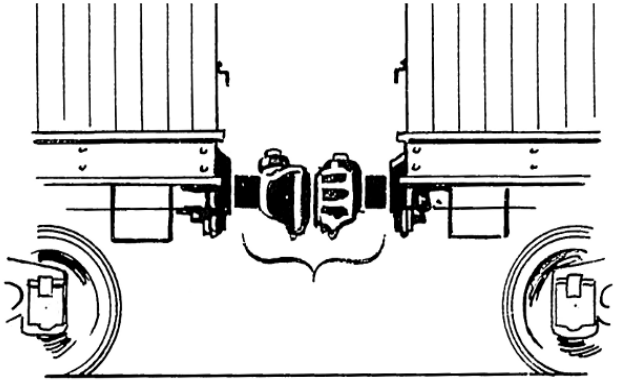
- **Introduction to Web Services**
  - **Why use or build a web service?**
  - **Best practices for implementing a web service**
- Spring Web Services
- Client access

© Copyright 2014 Pivotal. All rights reserved.

Web Services enable *Loose Coupling*

*"Loosely coupled systems are considered useful when either the **source** or the **destination** computer systems are subject to frequent **changes**"*

Karl Weikek (attrib)



Loose coupling
increases tolerance …
Changes should not
cause incompatibility

© Copyright 2014 Pivotal. All rights reserved.

Web Services enable *Interoperability*

- XML is the lingua franca in the world of interoperability
- XML is understood by all major platforms
  - *SAX*, *StAX* or *DOM* in Java
  - *System.XML* or *.NET XML Parser* in .NET
  - *REXML* or *XmlSimple* in Ruby
  - *Perl-XML* or *XML::Simple* in Perl

© Copyright 2014 Pivotal. All rights reserved.

Best practices for implementing web services

- Remember:
  - web services are not only about SOAP (e.g. REST)
  - web services != RPC
- Design contract independently from service interface
  - Generating contract from code results in tight coupling and awkward contracts
- Refrain from using stubs and skeletons
  - Code and contract should be able to evolve independently

© Copyright 2014 Pivotal. All rights reserved.

Loose Coupling & Validation

- Consider skipping validation for incoming requests...
  … and using Xpath to only extract what you need
- Easier to successfully process different requests
  - Looser coupling
  - Fewer versioning issues
  - Increases flexibility for clients
    - e.g. add additional contents, so request can be passed to other services as well
- Not necessarily a best practice, depends on the service

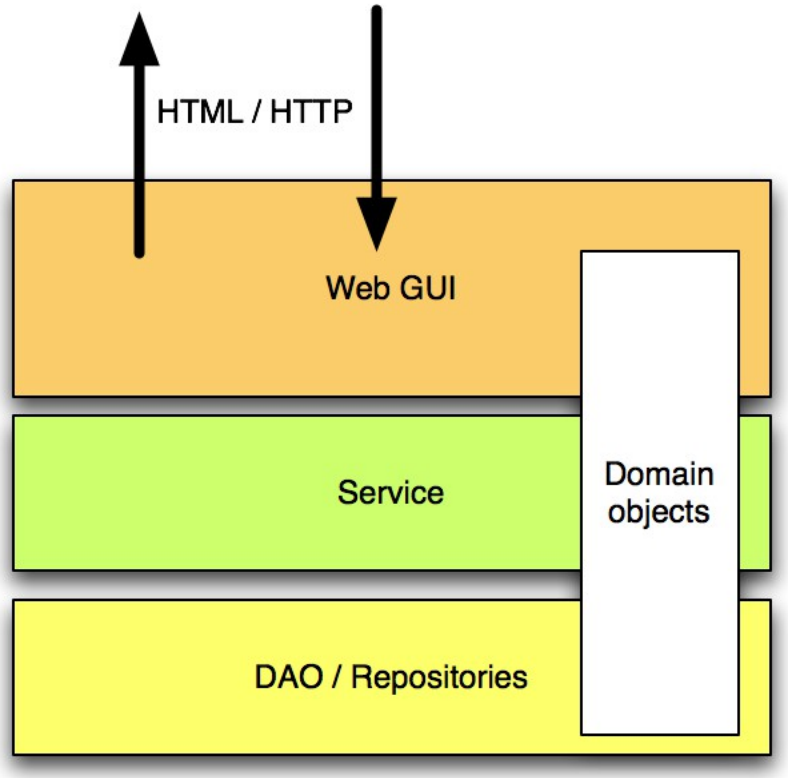
Postel's Law

"Be conservative in what you do;
be liberal in what you accept from others"

© Copyright 2014 Pivotal. All rights reserved.

Web GUI on top of your services

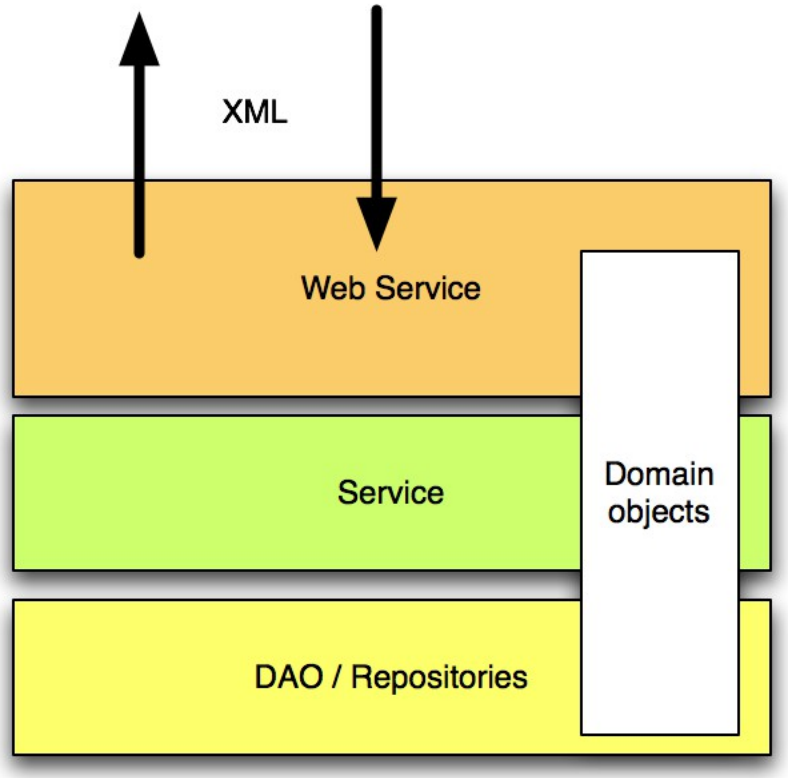
The **Web GUI layer** provides **compatibility** between **HTML-based** world of the user (the browser) and the **OO-based** world of your service



© Copyright 2014 Pivotal. All rights reserved.

Web Service on top of your services

Web Service layer provides **compatibility** between **XML-based** world of the user and the **OO-based** world of your service



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Web Services
  - Why use or build a web service?
  - Best practices for implementing a web service
- **Spring Web Services**
- Client access

© Copyright 2014 Pivotal. All rights reserved.

Define the contract

- Spring-WS uses Contract-first
  - Start with XSD/WSDL
- Widely considered a Best Practice
  - Solves many interoperability issues
- Also considered difficult
  - But isn't

© Copyright 2014 Pivotal. All rights reserved.

Contract-first in 3 simple steps

- Create sample messages
- Infer a contract
  - Trang
  - Microsoft XML to Schema
  - XML Spy
- Tweak resulting contract

© Copyright 2014 Pivotal. All rights reserved.

Sample Message

Namespace for this message

```
<transferRequest xmlns="http://mybank.com/schemas/tr"
    amount="1205.15">
    <credit>S123</credit>
    <debit>C456</debit>
</transferRequest>
```

© Copyright 2014 Pivotal. All rights reserved.

Define a schema for the web service message

```
<transferRequest
   amount="1205.15">
   <credit>S123</credit>
   <debit>C456</debit>
</transferRequest>
```

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tr="http://mybank.com/schemas/tr"
    elementFormDefault="qualified"
    targetNamespace="http://mybank.com/schemas/tr">
  <xs:element name="transferRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="credit" type="xs:string"/>
        <xs:element name="debit" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="amount" type="xs:decimal"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


© Copyright 2014 Pivotal. All rights reserved.

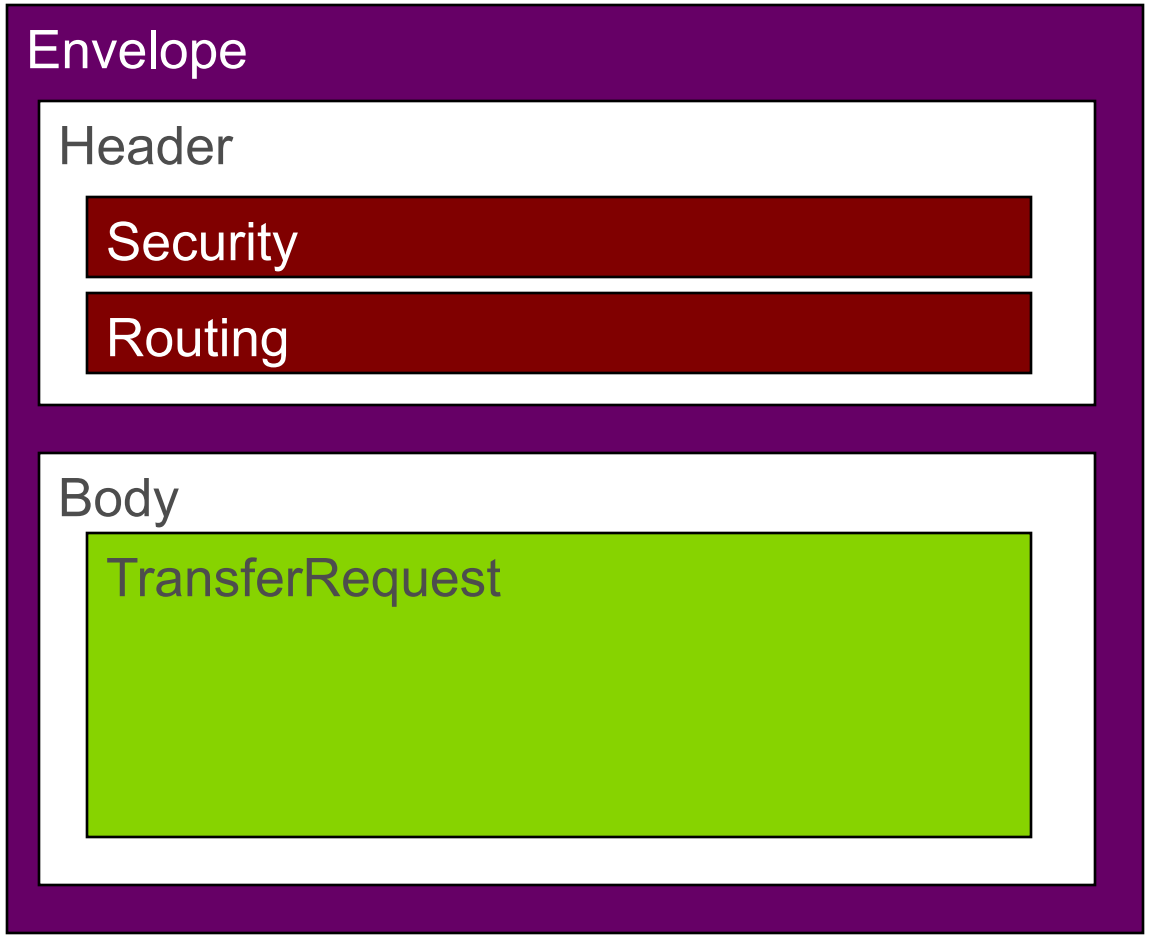
Type constraints

```
<xs:element name="credit" >
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="\w\d{3}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

1 Character + 3 Digits

© Copyright 2014 Pivotal. All rights reserved.

SOAP Message



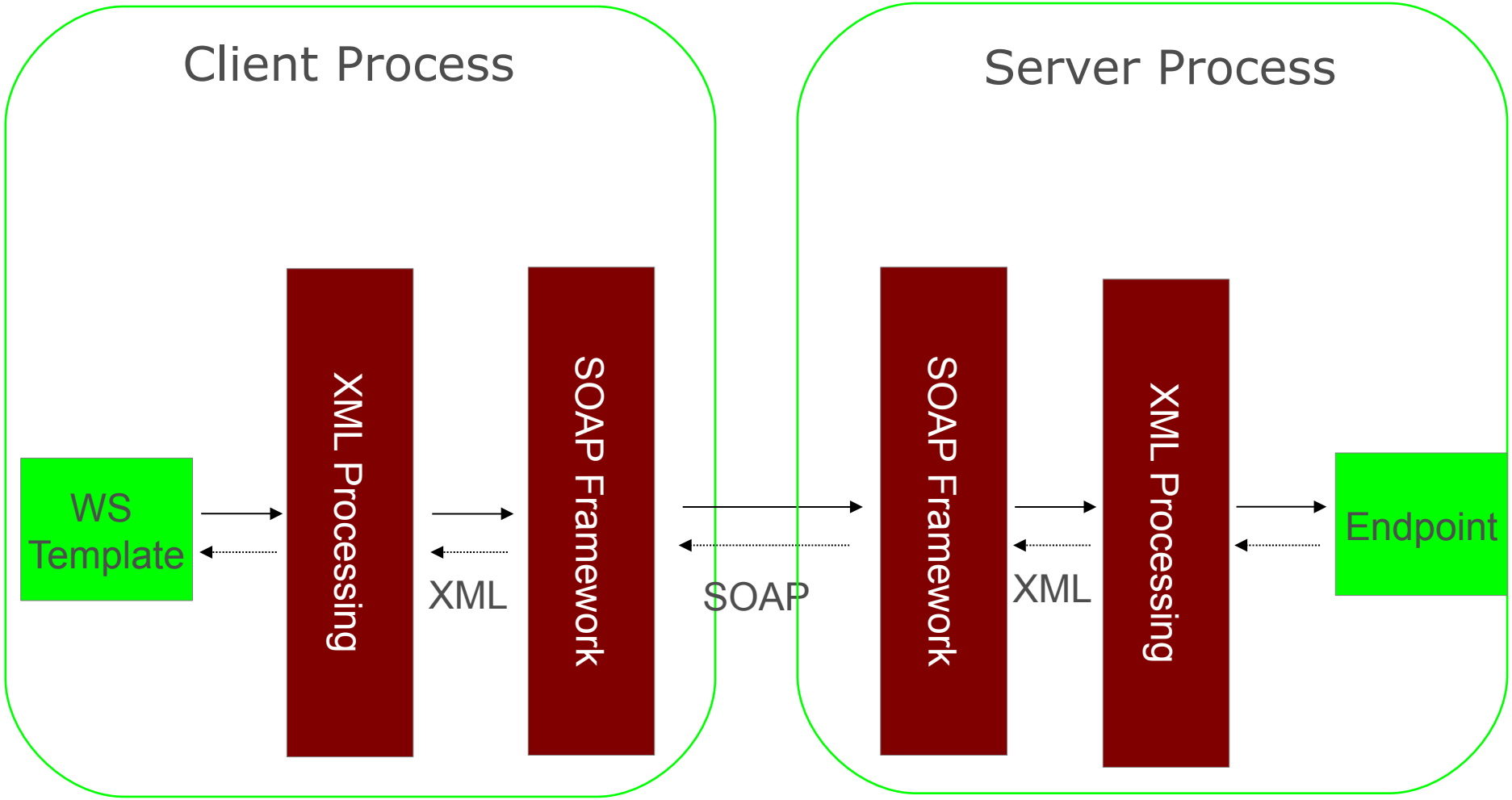
© Copyright 2014 Pivotal. All rights reserved.

Simple SOAP 1.1 Message Example

```
<SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <tr:transferRequest xmlns:tr="http://mybank.com/schemas/tr"
           amount=”1205.15”>
         <tr:credit>S123</tr:credit>
         <tr:debit>C456</tr:debit>
      </tr:transferRequest>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

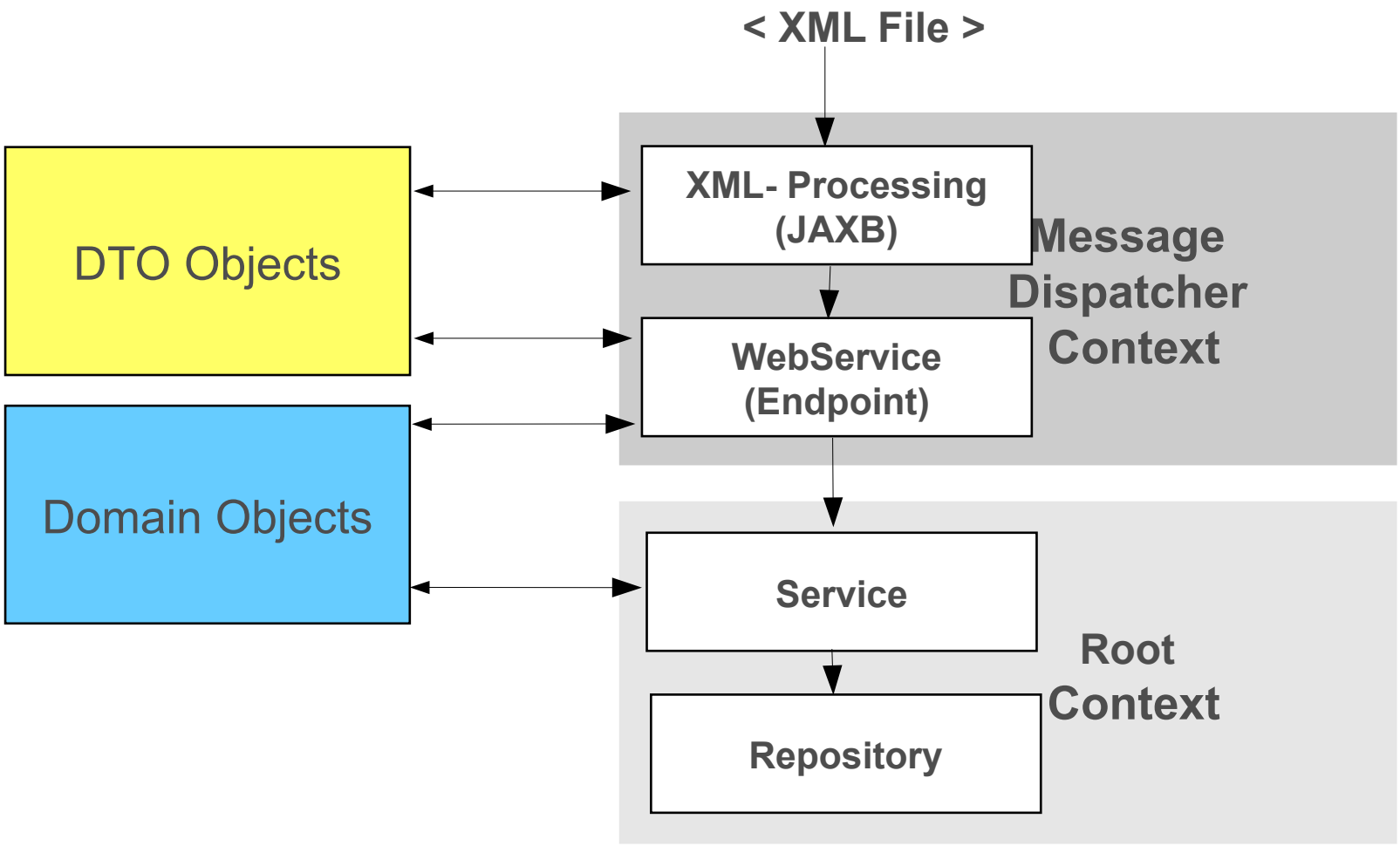
© Copyright 2014 Pivotal. All rights reserved.

Spring Web Services



© Copyright 2014 Pivotal. All rights reserved.

Spring Web Services Configuration



© Copyright 2014 Pivotal. All rights reserved.

Bootstrap the application tier

- Inside <webapp/> within web.xml

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
      /WEB-INF/transfer-app-cfg.xml
   </param-value>
</context-param>

<listener>
   <listener-class>
      org.springframework.web.context.ContextLoaderListener
   </listener-class>
</listener>
```

The application context's configuration file(s)

Loads the ApplicationContext into the ServletContext *before* any Servlets are initialized

© Copyright 2014 Pivotal. All rights reserved.

Wire up the Front Controller (MessageDispatcher)

- Inside <webapp/> within web.xml

```
<servlet>
   <servlet-name>transfer-ws</servlet-name>
   <servlet-class>..ws..MessageDispatcherServlet</servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/transfer-ws-cfg.xml</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
```

The application context's configuration file(s) containing the web service infrastructure beans

© Copyright 2014 Pivotal. All rights reserved.

Map the Message Dispatcher Servlet

- Inside <webapp/> within web.xml

```
<servlet-mapping>
   <servlet-name>transfer-ws</servlet-name>
   <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

There might also be a web interface (GUI) that is mapped to another path

© Copyright 2014 Pivotal. All rights reserved.

Endpoint

- Endpoints handle SOAP messages
- Similar to MVC Controllers
  - Handle input message
  - Call method on business service
  - Create response message
- With Spring-WS you can focus on the Payload
- Switching from SOAP to POX without code change

© Copyright 2014 Pivotal. All rights reserved.

XML Handling techniques

- Low-level techniques
  - DOM (JDOM, dom4j, XOM)
  - SAX
  - StAX
- Marshalling
  - JAXB (1 and 2)
  - Castor
  - XMLBeans
- XPath argument binding

© Copyright 2014 Pivotal. All rights reserved.

JAXB 2

- JAXB 2 is part of Java EE 5 and JDK 6
- Uses annotations for mapping metadata
- Generates classes from a schema and vice-versa
- Supported as part of Spring-OXM

```
<oxm:jaxb2-marshaller id="marshaller" contextPath="rewards.ws.types"/>
```

- No explicit marshaller bean definition necessary to be used in Spring-WS 2.0 endpoints

```
<!-- registers all infrastructure beans needed for annotation-based
     endpoints, incl. JABX2 (un)marshalling: -->
<ws:annotation-driven/>
```


© Copyright 2014 Pivotal. All rights reserved.

Implement the Endpoint

```
@Endpoint
public class TransferServiceEndpoint {
    private TransferService transferService;
    @Autowired
    public TransferServiceEndpoint(TransferService transferService) {
        this.transferService = transferService;
    }

    @PayloadRoot(localPart="transferRequest",
                 namespace="http://mybank.com/schemas/tr")
    public @ResponsePayload TransferResponse newTransfer(
                 @RequestPayload TransferRequest request) {
        // extract necessary info from request and invoke service
    }
}
```

Spring WS Endpoint

Mapping

Converted with JAXB2

© Copyright 2014 Pivotal. All rights reserved.

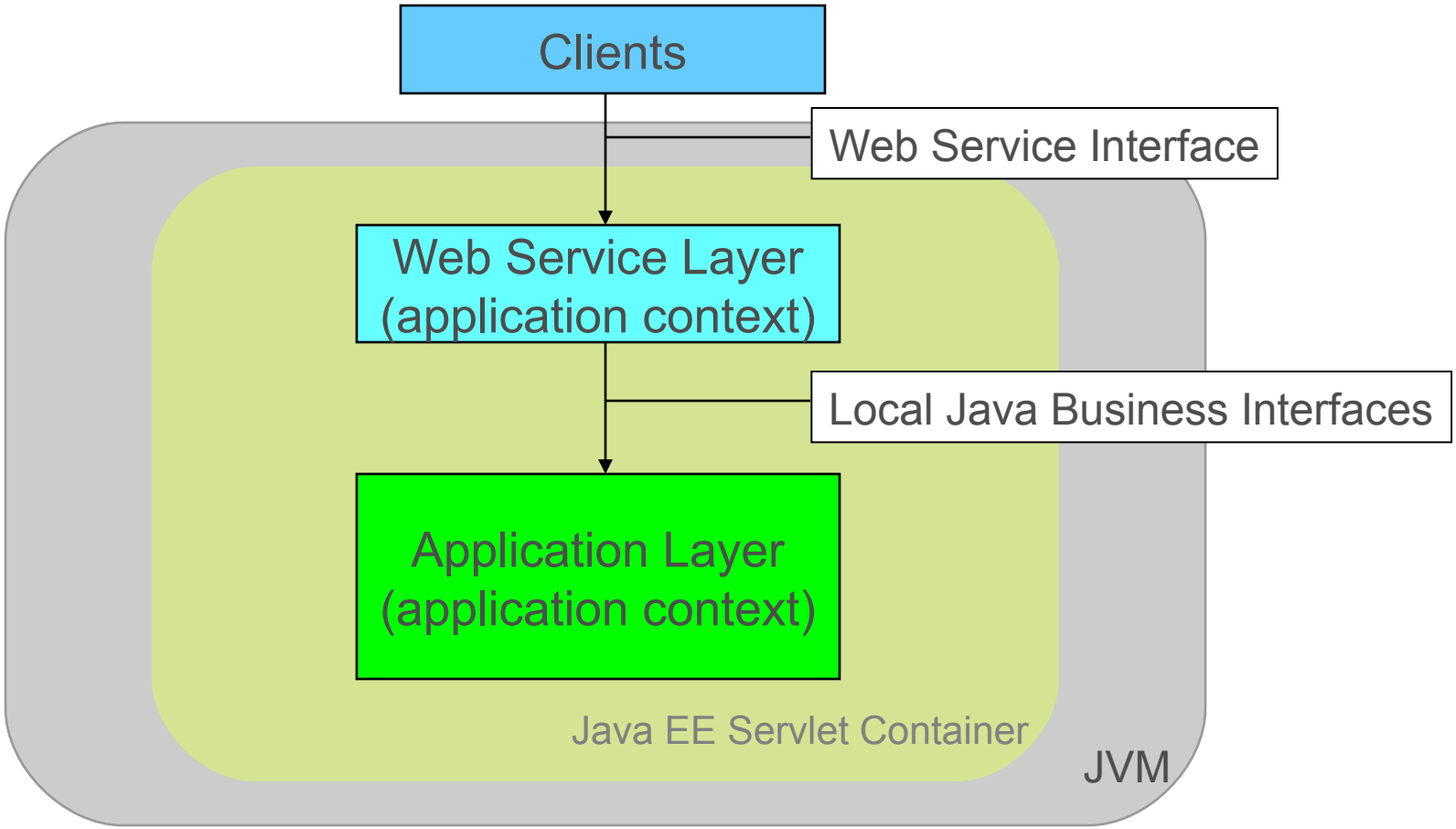
Configure Spring-WS and Endpoint beans

- @Endpoint annotated classes can be component-scanned
  - Saves writing lots of trivial XML bean definitions
  - Just make sure not to scan classes that belong in the root context instead

```
<!-- instead of defining explicit endpoints beans
     we rely on component scanning: -->
<context:component-scan base-package="transfers.ws"/>

<!-- registers all infrastructure beans needed for
     annotation-based endpoints, incl. JAXB2 (un)marshalling:
-->
<ws:annotation-driven/>
```


© Copyright 2014 Pivotal. All rights reserved.

Architecture of our application exposed using a a web service



© Copyright 2014 Pivotal. All rights reserved.

Further Mappings

- You can also map your Endpoints in XML by
  - Message Payload
  - SOAP Action Header
  - WS-Adressing
  - XPath

See: Spring Web Services Reference, 5.5 Endpoint Mappings

© Copyright 2014 Pivotal. All rights reserved.

Publishing the WSDL

http://somehost:8080/transferService/transferDefinition.wsdl

```
<ws:dynamic-wsdl id="transferDefinition"
                 portTypeName="Transfers"
        locationUri="http://somehost:8080/transferService/">
  <ws:xsd location="/WEB-INF/transfer.xsd"/>
</ws:dynamic-wsdl>
```

- Generates WSDL based on given XSD
  - id becomes part of WSDL URL
- Most useful during development
- Just use static WSDL in production
  - Contract shouldn't change unless YOU change it

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Web Services
  - Why use or build a web service?
  - Best practices for implementing a web service
- Spring Web Services
- **Client access**

© Copyright 2014 Pivotal. All rights reserved.

Spring Web Services on the Client

- WebServiceTemplate
  - Simplifies web service access
  - Works directly with the XML payload
    - Extracts *body* of a SOAP message
    - Also works with POX (Plain Old XML)
  - Can use marshallers/unmarshallers
  - Provides convenience methods for sending and receiving web service messages
  - Provides callbacks for more sophisticated usage

© Copyright 2014 Pivotal. All rights reserved.

Marshalling with WebServiceTemplate

```
<bean id="webServiceTemplate"
        class="org.springframework.ws.client.core.WebServiceTemplate">
   <property name="defaultUri" value="http://mybank.com/transfer"/>
   <property name="marshaller" ref="marshaller"/>
   <property name="unmarshaller" ref="marshaller"/>
</bean>

<bean id="marshaller" class="org.springframework.oxm.castor.CastorMarshaller">
   <property name="mappingLocation" value="classpath:castor-mapping.xml"/>
</bean>
```

```
WebServiceTemplate ws = context.getBean(WebServiceTemplate.class);
TransferRequest request = new TransferRequest("S123", "C456", "85.00");
Receipt receipt = (Receipt) ws.marshalSendAndReceive(request);
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to Web Services
  - Why use or build a web service?
  - Best practices for implementing a web service
- Spring Web Services
- Client access

© Copyright 2014 Pivotal. All rights reserved.

Lab

Exposing SOAP Endpoints using Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Advanced Spring Web Services

Interceptors, Error Handling and Testing

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Interceptors**
- Error handling
- Out-of-container testing

© Copyright 2014 Pivotal. All rights reserved.

Interceptors

- EndpointInterceptors receive callbacks during message handling flow
  - handleRequest: invoked before endpoint is called
  - handleResponse: invoked after endpoint (no fault)
  - handleFault: invoked after endpoint (fault)
- All methods can manipulate MessageContext (request/response) and abort further execution
- Built-in implementations for logging, validation, XSLT, WS-Security and WS-Addressing
- Add your own to customize the framework

© Copyright 2014 Pivotal. All rights reserved.

Interceptor Registration

- Register through namespace: apply to all endpoints

```
<ws:interceptors>
  <bean class="org.sfw.ws...SoapEnvelopeLoggingInterceptor" />
</ws:interceptors>
```

- Optionally restrict to requests with certain payload roots or SOAP actions

```
<ws:interceptors>
  <ws:payloadRoot localPart="MyRequest"
                  namespaceUri="http://some.domain.com/namespace">
    <ref bean="xwsSecurityInterceptor"/>
  </ws:payloadRoot>
</ws:interceptors>
```


© Copyright 2014 Pivotal. All rights reserved.

Built-in Interceptors (1)

- Logging interceptors:
  - SoapEnvelopeLoggingInterceptor
    - Logs full request & response envelope
  - PayloadLoggingInterceptor
    - Logs request & response payload only
- PayloadValidatingInterceptor
  - Validates request and/or response payload against schema
  - Results in SOAP fault for invalid requests
- PayloadTransformingInterceptor
  - Transforms request and/or response payload using XSLT stylesheet (e.g. for different version formats)

© Copyright 2014 Pivotal. All rights reserved.

Built-in Interceptors (2)

- AddressingEndpointInterceptor
  - Provides WS-Addressing support
- WS-Security interceptors
  - XwsSecurityInterceptor for Sun's XWSS
    - Requires Sun JVM and SAAJ
  - Wss4jSecurityInterceptor for WSS4J integration
    - Also supports non-Sun JVMs and Axiom
  - Support for Spring Security, JAAS and keystores

© Copyright 2014 Pivotal. All rights reserved.

Client-side Interceptors

- WebServiceTemplate also supports interceptors
  - ClientInterceptors, interface comparable to server
- Built-in implementations for validation and WS-Security

```
<bean id="webServiceTemplate" class="org.sfw.ws.client.core.WebServiceTemplate">
  <property name="interceptors">
    <bean class="org.sfw.ws.client.support.interceptor.PayloadValidatingInterceptor">
      <property name="schema" value="/WEB-INF/schemas/request-schema.xsd"/>
    </bean>
  </property>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Interceptors
- **Error handling**
- Out-of-container testing

© Copyright 2014 Pivotal. All rights reserved.

SOAP Faults

- SOAP uses faults to signal errors to a client
- Replace regular payload of response message
- Contain code, reason string and optional detail XML and actor URI

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <SOAP:Fault>
      <faultcode>SOAP:Server</faultcode>
      <faultstring>Oops!</faultstring>
    </SOAP:Fault>
  </SOAP:Body>
</SOAP:Envelope>
```

'Client', 'Server', 'MustUnderstand' and 'VersionMismatch' are predefined for SOAP 1.1

© Copyright 2014 Pivotal. All rights reserved.

Error Handling

- EndpointExceptionResolver maps exceptions from endpoint methods to SOAP Messages

Request/response wrapper

```
public interface EndpointExceptionResolver {
    boolean resolveException(MessageContext messageContext,
                             Object endpoint, Exception ex);
}
```

- Default is SimpleSoapExceptionResolver
  - Creates SOAP Fault with 'Server' code and exception message as fault string
- Define EndpointExceptionResolver to override

© Copyright 2014 Pivotal. All rights reserved.

SoapFaultMapping-ExceptionResolver

- Maps exceptions to faults by type name

```
<bean class="org.sfw.ws...SoapFaultMappingExceptionResolver">
   <property name="exceptionMappings">
      <value>
      org.springframework.oxm.ValidationFailureException=CLIENT,Invalid request
      </value>
   </property>
   <property name="defaultFault" value="SERVER"/>
</bean>
```

- Keys: exception types
- Values: code (required), fault string (optional), locale (defaults to English)

© Copyright 2014 Pivotal. All rights reserved.

SoapFaultAnnotation-ExceptionResolver

- Allows annotating exceptions to map to SOAP faults
  - So limited to custom application exceptions

```
<bean class="org.sfw.ws...SoapFaultAnnotationExceptionResolver"/>
```

```
@SoapFault(faultCode=FaultCode.CLIENT)
public class InvalidInputException extends Exception {
    public InvalidInputException(String message) {
        super(message);
    }
}
```

- Fault string defaults to exception message
  - Override using faultStringOrReason attribute

© Copyright 2014 Pivotal. All rights reserved.

WebServiceTemplate

- SoapFaultMessageResolver used by default for Fault response handling by WebServiceTemplate
  - Throws SoapFaultClientExceptions which wrap the SOAP message and expose the SoapFault
- Can provide custom implementation

```
public interface FaultMessageResolver {
   void resolveFault(WebServiceMessage message) throws IOException;
}
```

```
<bean class="org.springframework.ws.client.core.WebServiceTemplate">
  <property name="faultMessageResolver">
    <bean class="com.example.CustomFaultMessageResolver"/>
  </property>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Interceptors
- Error handling
- **Out-of-container testing**

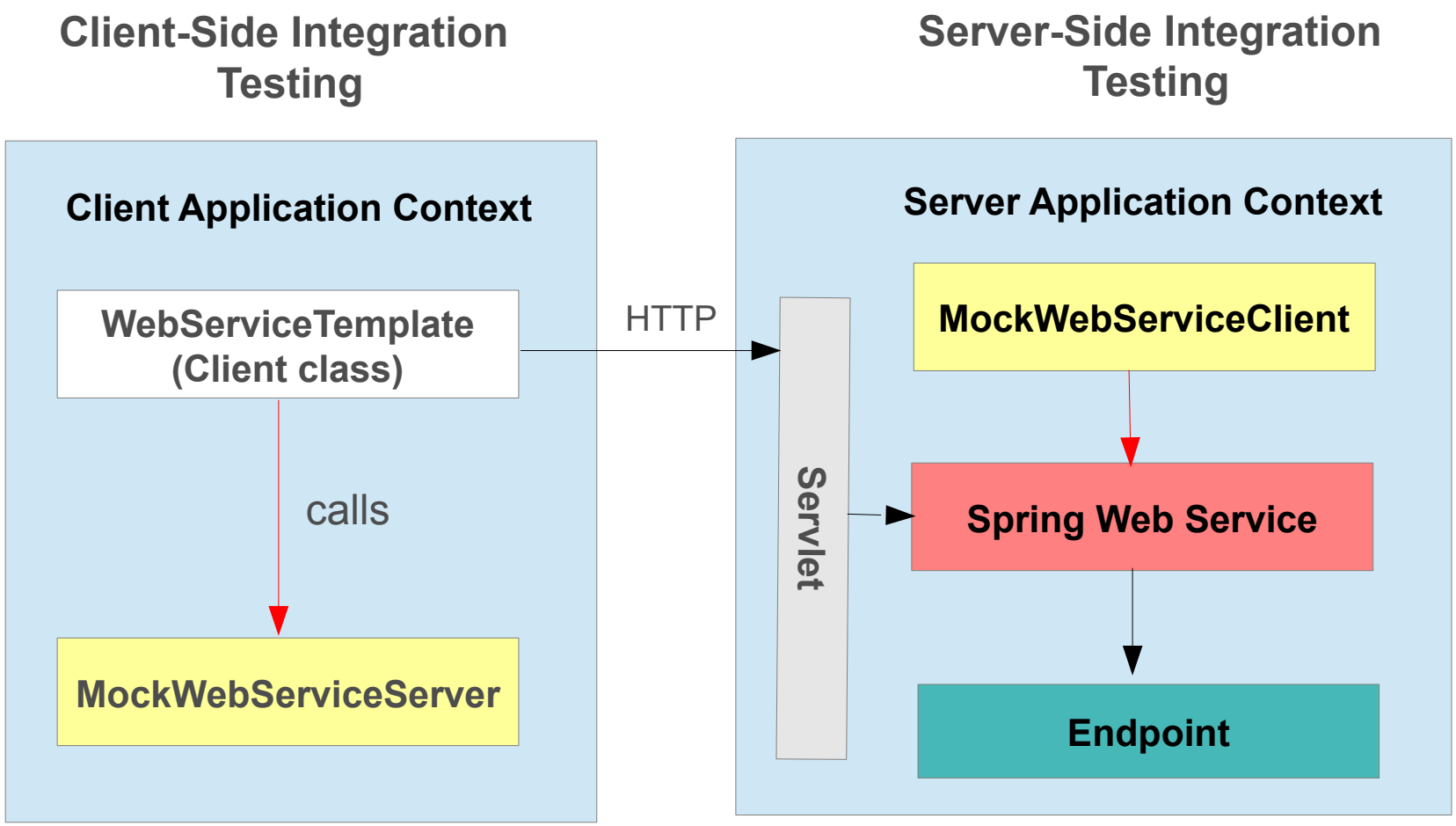
© Copyright 2014 Pivotal. All rights reserved.

Out-of-container Testing

- Spring-WS 2.0 adds support for out-of-container integration testing
  - No stubbing/mocking of endpoint parameters
  - No deployments to local server
- Test endpoints with MockWebServiceClient
  - Fake client that sends request and validates response
  - Tests full MessageDispatcher configuration
- Test clients with MockWebServiceServer
  - Fake server that validates request and sends response
  - Test full WebServiceTemplate configuration

© Copyright 2014 Pivotal. All rights reserved.

Integration Testing



© Copyright 2014 Pivotal. All rights reserved.

Using MockWebServiceClient

Steps to use the MockWebServiceClient:

1. Create an instance using one of its factory methods
2. Call the sendRequest method with a RequestCreator callback
3. Set up response expectations by calling andExpect method with a ResponseMatcher callback
4. Optionally repeat step 3 for multiple assertions

© Copyright 2014 Pivotal. All rights reserved.

Creating The Mock Client

- Pass in ApplicationContext to createClient
  - Same defaults apply as with MessageDispatcher

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("spring-ws-servlet.xml")
public class ServerTest {

    @Autowired ApplicationContext applicationContext;

    MockWebServiceClient mockClient;

    @Before
    public void createClient() {
        mockClient = MockWebServiceClient.createClient(applicationContext);
    }
```


© Copyright 2014 Pivotal. All rights reserved.

Send The Request

- sendRequest takes RequestCreator callback
  - Easiest to use RequestCreators.withPayload static factory methods
  - Provide Source or Resource for XML payload

```
import static org.springframework.ws.test.server.RequestCreators.withPayload;
… // code from previous slide omitted
@Test
public void customerEndpoint() throws Exception {
   Source requestPayload = new StringSource(
     "<customerCountRequest xmlns='http://springframework.org/spring-ws'>" +
      "<customerName>John Doe</customerName>" +
     "</customerCountRequest>");
   …
   mockClient.sendRequest(withPayload(requestPayload))
        .andExpect([someResponseMatcher]);
}
```

See next slide

© Copyright 2014 Pivotal. All rights reserved.

Check The Response

- andExpect takes ResponseMatcher callback
  - Easiest to use ResponseMatchers static factory methods for payload, header and/or fault checks
  - payload() takes Source or Resource, like withPayload()

```
import static org.springframework.ws.test.server.ResponseMatchers.*;

  … // code from previous slide omitted
  Source responsePayload = new StringSource(
    "<customerCountResponse xmlns='http://springframework.org/spring-ws'>" +
      "<customerCount>10</customerCount>" +
    "</customerCountResponse>");

  mockClient.sendRequest(withPayload(requestPayload))
      .andExpect(payload(responsePayload));
}
```

extra andExpect() calls can be chained

© Copyright 2014 Pivotal. All rights reserved.

Resource Access In Tests

- Spring-WS applications use WebApplicationContext
  - Resources loaded from root of war file by default
  - e.g. "/WEB-INF/schemas/reward-network.xsd"
- Server tests use ClassPathXmlApplicationContext
  - Resources loaded from classpath by default
  - Won't work with web-relative resources!
- Advice: use classpath resources for things you also need in integration tests
  - e.g. "classpath:schemas/reward-network.xsd"
  - Works with every ApplicationContext type
  - Same tip applies to other Spring web applications

© Copyright 2014 Pivotal. All rights reserved.

Using MockWebServiceServer

Steps to use the MockWebServiceServer:

1. Create an instance using one of its factory methods
2. Set up request expectations by calling andExpect method with a RequestMatcher callback
3. Optionally repeat step 2 for multiple assertions
4. Create a response using andRespond() with a ResponseCreator callback
5. Use the WebServiceTemplate
6. Verify expectations using verify()

© Copyright 2014 Pivotal. All rights reserved.

Creating The Mock Server

- Call createServer with WebServiceTemplate or ApplicationContext
  - Latter uses template defined in context
  - Template is configured to use fake MessageSender

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("integration-test.xml")
public class ClientTest {

    @Autowired WebServiceTemplate template;

    MockWebServiceServer mockServer;

    @Before
    public void createServer() throws Exception {
        mockServer = MockWebServiceServer.createServer(template);
    }
```


© Copyright 2014 Pivotal. All rights reserved.

Set Request Expectations

- expect takes RequestMatcher callback
  - Easiest to use RequestMatchers static factory methods for payload, XPath, header or URI checks

```
import static org.springframework.ws.test.client.RequestMatchers.*;
… // code from previous slide omitted

@Test
public void customerClient() throws Exception {
   Source requestPayload = new StringSource(
      "<customerCountRequest xmlns='http://springframework.org/spring-ws'>" +
       "<customerName>John Doe</customerName>" +
      "</customerCountRequest>");
   …
   mockServer.expect(payload(requestPayload))
        .andRespond([someResponseCreator]);
   …
}
```

extra expect() calls can be chained

© Copyright 2014 Pivotal. All rights reserved.

Create A Response

- andRespond takes ResponseCreator callback
  - Easiest to use ResponseCreators static factory methods for payload, connection error, fault or

```
import static org.springframework.ws.test.client.ResponseCreators.*;

   … // code from previous slide omitted
   Source responsePayload = new StringSource(
     "<customerCountResponse xmlns='http://springframework.org/spring-ws'>" +
       "<customerCount>10</customerCount>" +
     "</customerCountResponse>");

   mockServer.expect(payload(requestPayload))
        .andRespond(withPayload(responsePayload));
   …
}
```


© Copyright 2014 Pivotal. All rights reserved.

Call The Template

- After setting up expectations, use the template and verify the expectations
  - Use directly or through some client wrapper class

```
… // code from previous slide omitted
CustomerCountRequest request = new CustomerCountRequest("John Doe");
CustomerCountResponse response =
  (CustomerCountResponse) template.marshalSendAndReceive(request);

assertEquals(10, response.getCustomerCount);

mockServer.verify();
}
```

does not create real HTTP connection

© Copyright 2014 Pivotal. All rights reserved.

Lab

© Copyright 2014 Pivotal. All rights reserved.

REST

Representational State Transfer

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **What is REST?**
- Core REST Concepts
- RESTful architecture & design
- Advantages
- RestTemplate
- Java Frameworks
- Spring-MVC
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

What is REST?

- Representational State Transfer
- Term coined up by Roy Fielding
  - Author of HTTP spec
- Architectural style
- Architectural basis for HTTP
  - Defined a posteriori
- Embraces HTTP as an *application* protocol, not just as a *transport* protocol

© Copyright 2014 Pivotal. All rights reserved.

REST is NOT

- An API or framework
  - It is only an architectural style
- Equal to HTTP
  - REST principles can be followed using other protocols
- The opposite of SOAP
  - REST vs. SOAP is a false dichotomy

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- **Core REST Concepts**
- RESTful architecture & design
- Advantages
- RestTemplate
- Java Frameworks
- Spring-MVC
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

Core REST Concepts

- Identifiable Resources
- Uniform interface
- Stateless conversation
- Resource representations
- Hypermedia

© Copyright 2014 Pivotal. All rights reserved.

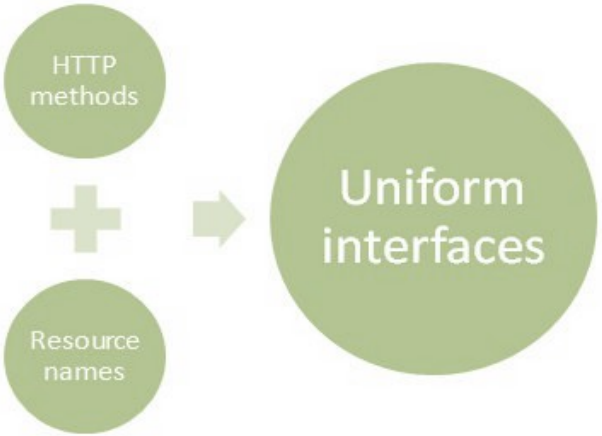
Identifiable Resources

- Everything is a resource
  - Customer
  - Car
  - Shopping cart
- Resources are expressed by URIs
  - Meaning URLs: REST community prefers the term URI
- Each URI adds value to the client
  - Don't just expose your entire domain

© Copyright 2014 Pivotal. All rights reserved.

Uniform Interface

- Interact with resources using a constrained set of operations
  - Many nouns (resources)
  - Few verbs (operations)
- Use HTTP Methods
  - GET
  - POST
  - PUT, PATCH
  - DELETE
  - HEAD and OPTIONS (for meta-data)



See *Appendix* at end of section for a overview of these commands

© Copyright 2014 Pivotal. All rights reserved.

Resource representations

- Resources are abstract
  - Only accessed through a particular *representation*
- Multiple representations are possible
  - `text/html`, `image/gif`, `application/pdf`
- Request specifies the desired representation using the `Accept` HTTP header
  - Or sometimes interpreted through file extension in URI like `.xml`, `.pdf`, ...
- Response reports the actual representation using the `Content-Type` HTTP header
- Best practice: use well-known media types

© Copyright 2014 Pivotal. All rights reserved.

Stateless Protocol

- Server does not maintain state, client does
  - *Don't use the HTTP Session!*
- Very scalable
  - Any server instance can serve a request
  - Just put a load balancer in front
- Client maintains state through links
  - Just like a "real" application
  - Next few slides
- Enforces loose coupling
  - no shared session knowledge

© Copyright 2014 Pivotal. All rights reserved.

Hypermedia

- Resources contain links
  - Client state transitions are made through these links
  - Links are provided by server
- Seamless evolution
  - Don't have to update clients when changing the server, just send new links
- Not always easy
  - Client needs some knowledge of server semantics
    - But REST has no WSDL
  - Should abstract fully: HATEOAS
    - See advanced section

© Copyright 2014 Pivotal. All rights reserved.

REST Example: **GET**

```
GET /transfers/121
Host: www.mybank.com
Accept: application/xml
…
```

```
HTTP/1.1 200 OK
Date: …
Content-Length: 1456
Content-Type: application/xml

<transfer id="121"
          amount="300.00">
   <credit>S123</credit>
   <debit>C456</debit>
</transfer>
```

Accept header defines representation

```
GET /transfers/121
Host: www.mybank.com
Accept: application/json
…
```

```
HTTP/1.1 200 OK
Date: …
Content-Length: 83
Content-Type: application/json

{ id: 121, amount: 300.00,
  credit: S123, debit:  C456 }
```

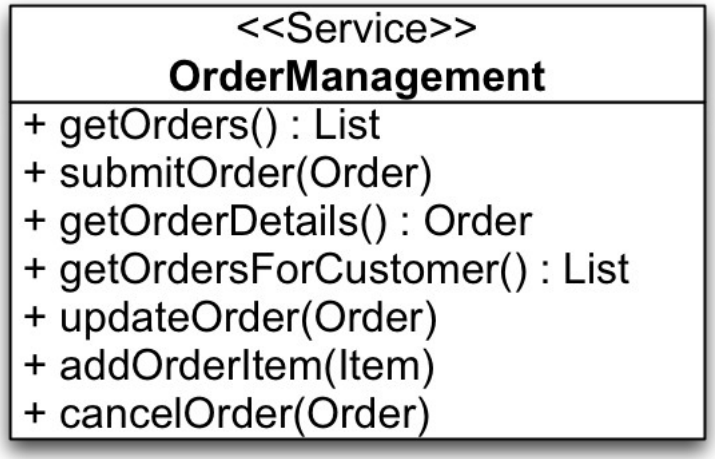

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- Core REST Concepts
- **RESTful architecture & design**
- Advantages
- RestTemplate
- Java Frameworks
- Spring-MVC
- Advanced Topics

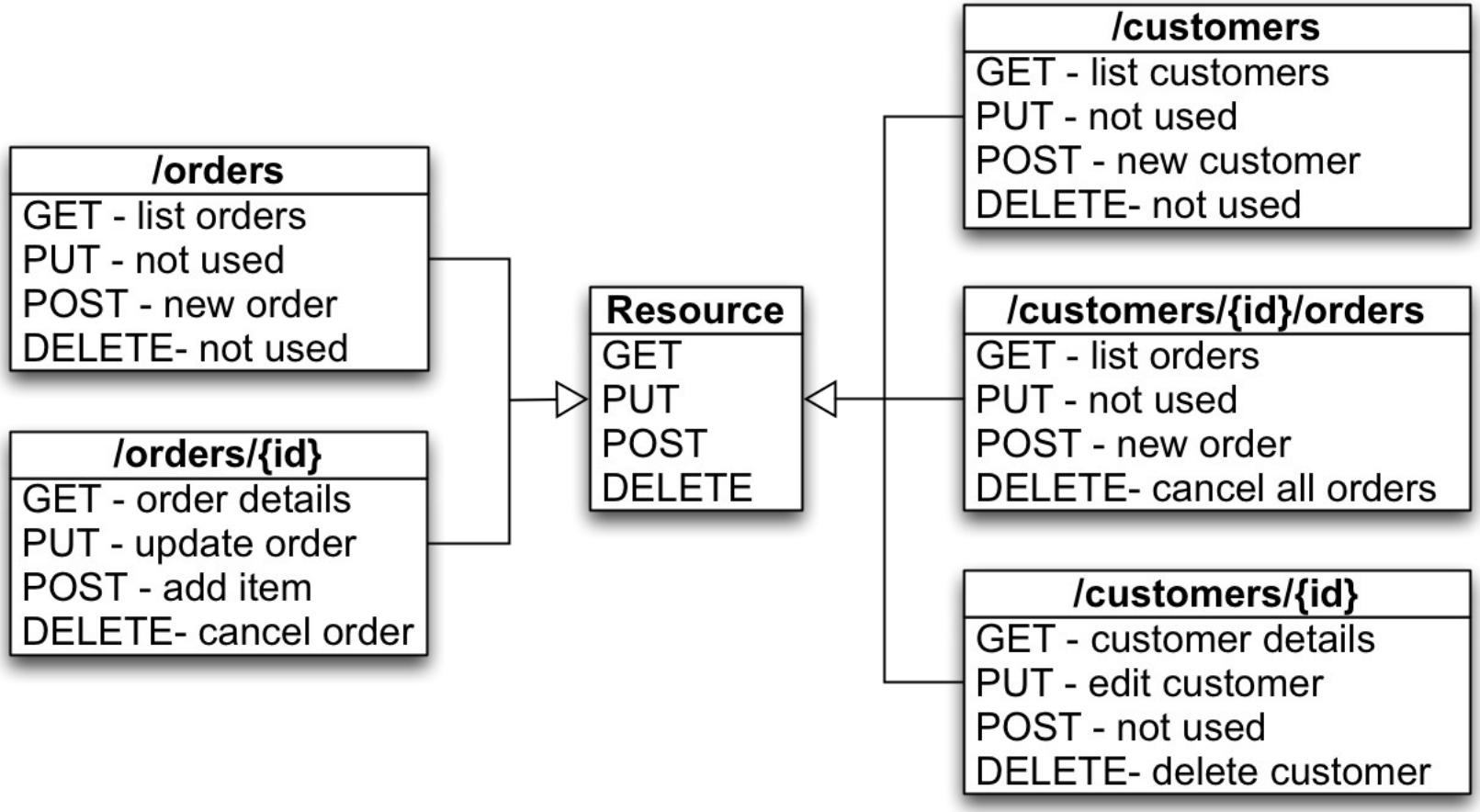
© Copyright 2014 Pivotal. All rights reserved.

RESTful Architecture



© Copyright 2014 Pivotal. All rights reserved.

RESTful Architecture



© Copyright 2014 Pivotal. All rights reserved.

RESTful application design

- Identify resources
  - Design URIs
- Select representations
  - Or create new ones
- Identify method semantics
- Select response codes

© Copyright 2014 Pivotal. All rights reserved.

HTTP Status Codes

- Web apps just use a handful of status codes
  - Success: 200 OK
  - Redirect: 302/303
  - Client Error: 404 Not Found
  - Server Error: 500 (such as unhandled Exceptions)
- RESTful applications use many additional codes to communicate with their clients

© Copyright 2014 Pivotal. All rights reserved.

Common Response Codes

200	• After a successful GET where content is returned
201	• When new resource was created on POST or PUT • Location header should contain URI of new resource
204	• When the response is empty • e.g. after successful update with PUT or DELETE
404	• When requested resource was not found
405	• When HTTP method is not supported by resource
409	• When a conflict occurs while making changes • e.g. when POSTing unique data that already exists
415	• When request body type not supported

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- Core REST Concepts
- RESTful architecture & design
- **Advantages**
- RestTemplate
- Java Frameworks
- Spring-MVC
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

Advantages

- Widely supported
  - Languages
  - Scripts
  - Browsers
    - only GET and POST through HTML
- Scalability
- Support for redirect, caching, different representations, resource identification, ...
- Support for XML, but also other formats
  - JSON and Atom are popular choices

© Copyright 2014 Pivotal. All rights reserved.

Security

- REST uses HTTP Basic or Digest
- Sent on every request
  - REST is stateless, remember ☺
- Requires SSL (transport-level security)
- Use XML-DSIG or XML-Encryption for message-level security
  - Like WS-Security for SOAP messages

© Copyright 2014 Pivotal. All rights reserved.

Transactions

- HTTP is not designed for long-running transactions
- Use *compensating transactions* instead
- This is also a WS-* best practice
  - WS-Business Activity

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- Core REST Concepts
- RESTful architecture & design
- Advantages
- **RestTemplate**
- Java Frameworks
- Spring-MVC
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

RestTemplate Introduction

- Provides access to RESTful services
  - Encapsulates all HTTP Methods
  - Pass variables as Map or String...
  - Specify url as java.net.URI or String
- Configuration Options
  - Support for URI templates
  - Uses *HttpMessageConverters* (later slide)
  - Custom execute() with callbacks
  - Support for asynchronous interactions

© Copyright 2014 Pivotal. All rights reserved.

RestTemplate API

HTTP	RestTemplate Method
DELETE	`delete(String url, String...urlVariables)`
GET	`getForObject(String url, Class<T> responseType, String...urlVariables)`
HEAD	`headForHeaders(String url, String...urlVariables)`
OPTIONS	`optionsForAllow(String url, String...urlVariables)`
POST	`postForLocation(String url, Object data, String...urlVariables)` `postForObject(String url, Object data, Class<T> responseType, String...uriVariables)`
PUT	`put(String url, Object data, String...urlVariables)`
PATCH	Since Spring 3.2: using `exchange()` and `execute()`

© Copyright 2014 Pivotal. All rights reserved.

Message Convertors – 1

- RestTemplate only deals in objects
  - Automated Object to/from payload conversion
  - Many defaults available “out-of-the-box”
    - Strings, XML, JSON, …
  - Automatically used and configured if found on classpath
- Spring Version Changes
  - Jackson JSON V1 convertor deprecated by Spring 4.0, removed by Spring 4.1
  - Spring 4.1 added convertors for: Jackson data-format XML, Google's GSON and Google Prototype buffers

© Copyright 2014 Pivotal. All rights reserved.

Message Convertors – 2

* Since Spring 4.1

Convertor	Data-type Handled	Media-Type
`StringHttpMessage Convertor`	A string from a request or response	**In:** `text/*` **Out:** `text/plain`
`MarshallingHttp MessageConvertor`	XML Payload using Spring OXM	`application/xml`
`MappingJackson2Xml HttpMessageConverter`	XML Payload using Spring OXM *	`application/xml`
`MappingJackson2Http MessageConverter`	JSON Payload using Jackson V2	`application/json`
`GsonHttpMessage Converter`	JSON payload using Google's GSON library *	`application/json`
`AtomFeedHttpMessage Converter`	ATOM feed using Rome's Feed API	`application/atom +xml`
`RssChannelHttp MessageConverter`	ATOM feed using Rome's Feed API	`application/rss +xml`

© Copyright 2014 Pivotal. All rights reserved.

Defining a RestTemplate

- Just call constructor in your code

```
RestTemplate template = new RestTemplate();
```

- Has default `HttpMessageConverters`
  - Same as on the server (depending on classpath)
- Or use external configuration
  - To use Apache Commons HTTP Client, for example

```
<bean id="restTemplate" class="org.sfw.web.client.RestTemplate">
   <property name="requestFactory">
      <bean class=
        "org.sfw.http.client.CommonsClientHttpRequestFactory"/>
   </property>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

RestTemplate Usage Examples

```
RestTemplate template = new RestTemplate();
String uri = "http://example.com/store/orders/{id}/items";

// GET all order items for an existing order with ID 1:
OrderItem[] items =
    template.getForObject(uri, OrderItem[].class, "1");

// POST to create a new item
OrderItem item = // create item object
URI itemLocation =
    template.postForLocation(uri, item, "1");

// PUT to update the item
item.setAmount(2);
template.put(itemLocation, item);

// DELETE to remove that item again
template.delete(itemLocation);
```

{id} = 1

{id} = 1

© Copyright 2014 Pivotal. All rights reserved.

HttpEntity

Spring 3.0.2

- To handle request/response explicitly
  - Set headers, control content sent/received
  - Use `HttpEntity` and `HttpHeaders`

```
// Add our own message to the Hello website
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.TEXT_PLAIN);
HttpEntity<String> entity =
            new HttpEntity<String>("Hello Spring", headers);
URI location = template.postForLocation("http://hello.org", entity);

// Get the message back again
HttpEntity<String> entity2 =
            template.getForEntity(location, String.class);
String message = entity2.getBody();
MediaType contentType = entity2.getHeaders().getContentType();
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- Core REST Concepts
- RESTful architecture & design
- Advantages
- RestTemplate
- **Java Frameworks**
- Spring-MVC
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

REST Frameworks

- Multiple Java frameworks for REST exist
- For Spring users, two options are particularly interesting:
  1) JAX-RS
  2) Spring-MVC with updated REST-support
- Both are valid choices depending on requirements and developer experience

© Copyright 2014 Pivotal. All rights reserved.

JAX-RS

- Java API for RESTful Web Services (JSR-311)
- Part of Java EE since Java EE 6
- Standard for writing RESTful applications
- Focuses mostly on application-to-application communication
  - Less focus on browsers as RESTful clients
- Very complete
- Jersey is the reference implementation
  - Ships with Spring integration out of the box
  - RESTEasy, Restlet and CXF support Spring as well

© Copyright 2014 Pivotal. All rights reserved.

JAX-RS Example

```
import javax.ws.rs.*;
import org.springframework.stereotype.Component;
import org.springframework.context.annotation.Scope;

@Path("/reward/{number}")
@Component
@Scope("request")
public class RewardResource {

    @GET
    @Produces("application/xml")
    public Reward getReward(@PathParam("number") String number) {
        return findReward(number);
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Spring MVC

- Spring MVC includes REST support (since 3.0)
  - URI templates
  - Message converters
  - Declaring response status codes
  - Content negotiation
  - `RestTemplate` for clients
  - And more
- Easier for existing MVC users than JAX-RS
  - As JAX-RS requires different programming model
- Also supports browsers as REST clients
  - Through HTTP Method conversion

© Copyright 2014 Pivotal. All rights reserved.

Spring MVC Example

```
@Controller
@RequestMapping("/reward/{number}")
public class RewardController {

    @RequestMapping(method=GET)
    public @ResponseBody Reward getReward(
                                @PathVariable String number) {
        return findRewardByNumber(number);
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- What is REST?
- Core REST Concepts
- RESTful architecture & design
- Advantages
- RestTemplate
- Java Frameworks
- **Spring-MVC**
- Advanced Topics

© Copyright 2014 Pivotal. All rights reserved.

@ResponseStatus

- Overrides default Http response (200 - OK)
- When using `@ResponseStatus`, *void* methods no longer imply a default view name!
  - *There will be no View at all*
  - Code below returns a response with an *empty* body

```
@RequestMapping(value="/orders", method=RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)  // 201
public void create(HttpServletRequest req, HttpServletResponse resp) {
  Order order = createOrder(req);
  resp.addHeader("Location", getLocationForChildResource(req, order.getId()));
}
```

Custom helper method

© Copyright 2014 Pivotal. All rights reserved.

Determining Location Header

- Location header value must be full URL
  - Determine based on request URL
    - Controller shouldn't know host name or servlet path
- URL of created child resource usually a sub-path
  - POST to http://www.myshop.com/store/orders
    gives http://www.myshop.com/store/orders/**123**
  - Can use Spring's UriTemplate for encoding where needed

```
private String getLocationForChildResource(HttpServletRequest request,
                                           Object childIdentifier) {
  StringBuffer url = request.getRequestURL();
  UriTemplate template = new UriTemplate(url.append("/{childId}").toString());
  return template.expand(childIdentifier).toASCIIString();
}
```


© Copyright 2014 Pivotal. All rights reserved.

@ResponseStatus & Exceptions

- Can also annotate exception classes with this
  - Given status code used when exception is thrown from controller method

```
@ResponseStatus(HttpStatus.NOT_FOUND)  // 404
public class OrderNotFoundException extends RuntimeException {
   …
}
```

```
@RequestMapping(value="/orders/{id}", method=GET)
public String showOrder(@PathVariable("id") long id, Model model) {
   Order order = orderRepository.findOrderById(id);
   if (order == null) throw new OrderNotFoundException(id);
   model.addAttribute(order);
   return "orderDetail";
}
```


© Copyright 2014 Pivotal. All rights reserved.

@ExceptionHandler

- For existing exceptions you cannot annotate, use @ExceptionHandler method on controller
  - Method signature similar to request handling method
  - Also supports @ResponseStatus

```
@ResponseStatus(HttpStatus.CONFLICT)  // 409
@ExceptionHandler({DataIntegrityViolationException.class})
public void conflict() {
    // could add the exception, response, etc. as method params
}
```


© Copyright 2014 Pivotal. All rights reserved.

Accessing Request/Response Data

- Can explicitly inject `HttpServletRequest` (and `HttpServletResponse`)
  - But makes Controller methods hard to test
  - Consider Spring's `MockHttp`... classes
- Spring can *automatically* inject part of the request
  - `@RequestParam`, `@PathVariable`, `Principle`, `Locale`, `@Value`, `@RequestHeader`
- To perform REST we also need:
  - `@RequestBody`, `@ResponseBody`

© Copyright 2014 Pivotal. All rights reserved.

@RequestBody

- For POSTs and PUTs, HTTP request body can be converted to method parameter
  - Converter chosen based on Content-Type of request

```
@RequestMapping(value="/orders/{id}", method=RequestMethod.PUT)
@ResponseStatus(HttpStatus.NO_CONTENT)  // 204
public void updateOrder(@RequestBody Order updatedOrder,
                        @PathVariable("id") long id) {
  // process updated order data and return empty response
  orderManager.updateOrder(id, updatedOrder);
}
```

```
PUT /store/orders/123
Host: www.myshop.com
Content-Type: application/xml
<order>
…
</order>
```

```
HTTP/1.1 204 No Content
Date: …
```


© Copyright 2014 Pivotal. All rights reserved.

@ResponseBody

- Object returned by controller method converted directly into HTTP response body
  - Instead of Model rendered by View
  - Converter chosen based on Accept header
    - XML using JAXB2, JSON using Jackson, etc.

```
@RequestMapping(value="/orders/{id}", method=RequestMethod.GET)
@ResponseStatus(HttpStatus.OK)  // 200; optional, this is the default
public @ResponseBody Order getOrder(@PathVariable("id") long id) {
  return orderRepository.findOrderById(id);
}
```

```
GET /store/orders/123

Host: www.myshop.com

Accept: application/xml
```

```
HTTP/1.1 200 OK
Date: …
Content-Length: 1456
Content-Type: application/xml

<order id="123">
…
</order>
```


© Copyright 2014 Pivotal. All rights reserved.

Content Negotiation - Accept

```
@RequestMapping(value="/orders/{id}", method=RequestMethod.GET)
@ResponseStatus(HttpStatus.OK)  // 200
public @ResponseBody Order getOrder(@PathVariable("id") long id) {
   return orderRepository.findOrderById(id);
}
```

GET /store/orders/123
Host: www.myshop.com
Accept: application/xml, ...
…

HTTP/1.1 200 OK
Date: …
Content-Length: 1456
Content-Type: application/xml

<order id="123">
…
</order>

GET /store/orders/123
Host: www.myshop.com
Accept: application/json, ...
…

HTTP/1.1 200 OK
Date: …
Content-Length: 756
Content-Type: application/json

{
"order": {"id": 123, "items": […], … }
}

© Copyright 2014 Pivotal. All rights reserved.

Spring 4.0

@RestController Simplification

```
@Controller
public class OrderController {
  @RequestMapping(value="/orders/{id}", method=RequestMethod.GET)
  public @ResponseBody Order getOrder(@PathVariable("id") long id) {
    return orderRepository.findOrderById(id);
  }

  …
}
```

```
@RestController
public class OrderController {
  @RequestMapping(value="/orders/{id}", method=RequestMethod.GET)
  public Order getOrder(@PathVariable("id") long id) {
    return orderRepository.findOrderById(id);
  }

  …
}
```

No need for @ResponseBody on GET methods

© Copyright 2014 Pivotal. All rights reserved.

Summary

You should have learned that:

- Java provides JAX-RS as standard specification
- Spring-MVC adds REST support using a familiar programming model
  - Extended by `@RequestBody`, `@ResponseBody`, `@ResponseStatus`, `@PathVariable`
  - Message convertors replace views
  - Automatic content-negotiation

© Copyright 2014 Pivotal. All rights reserved.

Lab

Building a RESTful web service

Coming Up: Testing, HATEOAS, Request/ResponseEntity, HTTP Methods

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Advanced Topics**
  - **Out-of-container Testing**
  - HATEOAS
  - RequestEntity / ResponseEntity
- Appendix
  - HTTP Methods

© Copyright 2014 Pivotal. All rights reserved.

Spring 3.2

MVC Test Framework Overview

- Began as a separate project on GitHub
  - Incorporated into Spring Framework since *Spring 3.2*
  - Found in spring-test.jar
- **Goal:** Provide first class *JUnit* support for testing Spring MVC code
  - Fluent API
  - Process requests *through* DispatcherServlet
  - Does ***not*** require Web container to test

SPRING TEST FRAMEWORK

© Copyright 2014 Pivotal. All rights reserved.

Spring 3.2

MVC Test Framework Example

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration("classpath:test-servlet-context.xml")
public final class AccountControllerTests {
    @Autowired
    private WebApplicationContext wac;
    private MockMvc mockMvc;

    @Before
    public void setup() {
        mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
    }

    @Test
    public void testBasicGet() {
        mockMvc.perform(get("/accounts")).andExpect(status().isOk());
    }
}
```

Tells test runner to instantiate a WebApplicationContext

Inject WebApplicationContext

Define MockMvc field and initialize it

Perform tests on mockMvc instance

© Copyright 2014 Pivotal. All rights reserved.

Unit Testing vs using MVC Test Framework

- Unit Testing
  - Use to test *logic* of controllers
  - Create & populate model, test for proper views, model data, etc
- MVC Test Framework
  - Use to test MVC annotations (like `@RequestMapping`, `@PathVariable`, etc)
  - Also can perform integration tests with other Spring MVC beans (such as *View Resolvers*, *Filters*, etc)

© Copyright 2014 Pivotal. All rights reserved.

Setting Up The Test Harness

```
import static o.sfw.test.web.servlet.request.MockMvcRequestBuilders.*;
import static o.sfw.test.web.servlet.result.MockMvcResultMatcher.*;

...
public final class AccountControllerTests {
    @Autowired
    private WebApplicationContext wac;
    private MockMvc mockMvc;

    @Before
    public void setup() {
        // Initialize mockMvc using WebApplicationContext
        mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
    }

    // Now perform tests using the mockMvc object.

}
```

Static imports make it easier to invoke Builder & Matcher static methods

© Copyright 2014 Pivotal. All rights reserved.

Setting Up Static Imports

- Static imports are key to fluid builders
  - `MockMvcRequestBuilders.*` and `MockMvcResultMatchers.*`
- You can add to Eclipse '*favorite static members*' in preferences
  - *Java → Editor → Content Assist → Favorites*
  - Add to favorite static members
    - `org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get`
    - `org.springframework.test.web.servlet.result.MockMvcResultMatchers.status`

© Copyright 2014 Pivotal. All rights reserved.

Perform and Expect

- Argument to 'perform' dictates the action
  - 'perform' returns `ResultActions` object
  - Can chain `expects` together – fluid syntax

```
@Test
public void basicAccountDetailsRequest() {
  mockMvc.perform(get("/accounts/{acctId}", "123456001"))
    .andExpect(status().isOk());                    // Expect status code 200

}
```

`MockMvcRequestBuilders` methods go in perform:
Options include *get*, *put*, *post*, *delete*, *fileUpload*

`MockMvcResultMatcher` methods in *italics*

Matcher specific assertion methods go here

© Copyright 2014 Pivotal. All rights reserved.

RequestBuilderMethods

- Standard HTTP *get*, *put*, *post*, *delete* operations
  - *fileUpload* also supported
  - Argument usually a URI template string
  - Returns a `MockHttpServletRequestBuilder` instance

```
// Perform a get using URI template style
mockMvc.perform(get("/accounts/{acctId}", "123456001"))

// Perform a get using request parameter style
mockMvc.perform(get("/accounts?myParam={acctId}", "123456001"))
```


© Copyright 2014 Pivotal. All rights reserved.

MockHttpServletRequetBuilder Examples

- Adding Request Parameters

```
mockMvc.perform(post("/accounts/{acctId}","123456001")
    .param("name", “Some Name”)                // Simulate form submission
    .param("dateOfBirth",  “2012-05-10”)
    ...            // More parameters and/or other MockHttpServletResponseBuilder
                   // methods can be added
```

- Setting Accept Header

```
mockMvc.perform(get("/accounts/{acctId}","123456001")
    .accept("application/json;charset=UTF-8")    // Request JSON Response
    …
```


© Copyright 2014 Pivotal. All rights reserved.

MockMvcResultMatcher methods

- Returns Matchers providing specific assertions
  - Makes use of Hamcrest
  - Consult JavaDoc for details on various assertions

Method	Matcher Returned	Description
content	ContentResultMatchers	Assertions relating to the HTTP response body
model	ModelResultMatchers	Assertions relating to the returned Model
header	HeaderResultMatchers	Assertions on HTTP headers
view	ViewResultMatchers	Assertions on returned view
status	StatusResultMatchers	Assertions on the HTTP status
forwardedUrl	ResultMatcher	Assertions on forwarded URL

© Copyright 2014 Pivotal. All rights reserved.

Testing RESTful Controllers I

- Can test RESTful interactions
  - Need to specify expected contentType and accept values for representation

```
@RequestMapping(method = RequestMethod.GET,
                produces="application/json")
public @RequestBody Account get(@PathVariable String number) {
  return accountManager.findAccount(number);
}
```

JSON representation

```
{
  "entityId": 1,
  "number": "123456001",
  "creditCardNumber": "1234123412340001",
  ...
}
```


© Copyright 2014 Pivotal. All rights reserved.

Testing RESTful Controllers II

- Note use of accept() and jsonPath()
  - Assertions use JsonPath – like XPath for JSON
    - See https://code.google.com/p/json-path/

```
@Test
public void testRestfulGet() throws Exception {
    mockMvc.perform(get("/accounts/{acctId}","123456001")
                    .accept(MediaType.APPLICATION_JSON))
                    .andExpect(content().contentType("application/json"))
                    .andExpect(jsonPath("$.number").value("123456001"));
}
```


© Copyright 2014 Pivotal. All rights reserved.

Printing Debug Information

- Sometimes you want to know what happened
  - andDo() performs action on MvcResult
  - print() sends the MvcResult to output stream
  - Could also use andReturn() to get the MvcResult object

```
// Use this to access the print() method
import static org.springframework.test.web.servlet.result.MockMvcResult.*;
// Other static imports as well

// Use print() method in test to get debug information
mockMvc.perform(get("/accounts/{acctId}","123456001")
                .andDo(print())  // Add this line to print debug information to the console
                …
```


© Copyright 2014 Pivotal. All rights reserved.

Performing Security Testing

```
public class AccountControllerIntegrationTests {
  @Autowired private WebApplicationContext context;
  @Autowired private FilterChainProxy springSecurityFilterChain;
  private MockMvc mockMvc;

  @Before public void setup() {
    mockMvc = webAppContextSetup(context).
                addFilter(springSecurityFilterChain).build();
  }
  @Test public void requiresAuthentication() throws Exception {
     mockMvc.perform(get("/"))
         .andExpect(redirectedUrl("http://localhost/login"));
   }
}
```

Inject springSecurityFilter-Chain and add as a filter

Assert unauthenticated access redirects to the configured login page

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Advanced Topics**
  - Out-of-container Testing
  - **HATEOAS**
  - RequestEntity / ResponseEntity
- Appendix
  - HTTP Methods

© Copyright 2014 Pivotal. All rights reserved.

HATEOAS

- Hypermedia As The Engine of Application State
  - Probably the world's worst acronym!
  - RESTful responses contain the links you need
  - Just like HTML pages do
  - Warning: no standard for this yet
  - Least understood part of Roy Fielding's dissertation

http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

- For example: GET /orders/123
  - Response should include possible URLs from here
    - Such as: how to get order item details, add new item ..

© Copyright 2014 Pivotal. All rights reserved.

HATEOAS - Concepts

- REST clients need *no* prior knowledge about how to interact with a particular application/server
  - SOAP web-services need a WSDL
  - SOA processes require a fixed interface defined using interface description language (IDL).
- Clients interact entirely through hypermedia
  - Provided dynamically by application servers
- Serves to *decouple* client and server
  - Allows the server to evolve functionality independently
  - Unique compared to other architectures

© Copyright 2014 Pivotal. All rights reserved.

HATEOAS Account Example

Spring Hateaos provides an API for generating these links in MVC Controller responses

```
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
  <link rel="withdraw" href="/account/12345/withdraw" />
  <link rel="transfer" href="/account/12345/transfer" />
  <link rel="close" href="/account/12345/close" />
</account>
```

Note: links *change* as *state* changes

```
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
</account>
```

Just a suggestion – there is no standard for links yet

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Advanced Topics
  - Out-of-container Testing
  - HATEOAS
  - RequestEntity / ResponseEntity
- **Appendix**
  - **HTTP Methods**

© Copyright 2014 Pivotal. All rights reserved.

RequestEntity, ResponseEntity

Spring 4.1

- Specific subclasses of `HttpEntity`
  - Fluent API since Spring 4.1

```
// Add your own message to the Hello website
ResponseEntity<Void> response = template.exchange(
    RequestEntity.post(new URI("http://hello.org"))
                 .contentType(MediaType.TEXT_PLAIN).
                 .body("Hello Spring"),
    null);
URI location = response.getHeaders().getLocation();

// Get the message back again
ResponseEntity<String> resp2 = template.exchange(
    RequestEntity.get(location)
                 .accept(MediaType.TEXT_PLAIN).build(),
    String.class);
String text = resp2.getBody();  // Should be 'Hello Spring'
```


© Copyright 2014 Pivotal. All rights reserved.

Appendix: REST Methods

- Brief overview of the HTTP methods used by REST
  - GET
  - OPTIONS
  - POST
  - PUT
  - PATCH
  - DELETE

© Copyright 2014 Pivotal. All rights reserved.

GET

- `GET` retrieves a Representation of a Resource
- `GET` is a *safe* operation
  - Has **no** side effects
- `GET` is *cacheable*
  - Servers may return ETag header when accessed
  - Clients send this header on subsequent retrieval
  - If the resource has not changed, 304 (Not Modified) is returned, with empty body
  - Similar solution exists for Last-Modified header

© Copyright 2014 Pivotal. All rights reserved.

GET Examples

```
GET /transfers/121

Host: www.mybank.com

Accept: application/xml
```

…

```
HTTP/1.1 200 OK
Date: …
Content-Length: 1456
Content-Type: application/xml

<transfer id="121"
          amount="300.00">
   <credit>S123</credit>
   <debit>C456</debit>
</transfer>
```

Accept header defines representation

```
GET /transfers/121

Host: www.mybank.com

Accept: application/json
```

```
HTTP/1.1 200 OK
Date: …
Content-Length: 83
Content-Type: application/json

{ id: 121, amount: 300.00,
  credit: S123, debit:  C456 }
```


© Copyright 2014 Pivotal. All rights reserved.

ETags - Caching

```
GET /transfers/121

Host: www.mybank.com

...
```

```
HTTP/1.1 200 OK
Date: …
ETag: "b4bdb3-5b0-
  43ad74ee73ec0"
Content-Length: 1456
Content-Type: text/html
...
```

```
GET /transfers/121

If-None-Match:"b4bdb3-
5b0-43ad74ee73ec0"

Host: www.mybank.com

...
```

```
HTTP/1.1 304 Not Modified
Date: …
ETag: "b4bdb3-5b0-
  43ad74ee73ec0"
Content-Length: 0
```


© Copyright 2014 Pivotal. All rights reserved.

HEAD

- Get meta-information about a resource
  - Returns the same HTTP response headers as GET
  - But no content

```
GET /transfers/121
Host: www.mybank.com
Accept: application/xml
…
```

```
HTTP/1.1 200 OK
Date: …
Content-Length: 1456
Content-Type: application/xml
```


© Copyright 2014 Pivotal. All rights reserved.

OPTIONS

- What HTTP methods does the server support?
  - Not a widely supported HTTP method
  - May get a *501 Not Implemented* or *405 Method Not Allowed* response

```
OPTIONS /transfers

Host: www.mybank.com
```

```
HTTP/1.1 200 OK
Allow: GET, PUT, POST, OPTIONS, HEAD, DELETE, PATCH
Accept-Patch: application/example, text/example
```


© Copyright 2014 Pivotal. All rights reserved.

POST

- POST creates a new Resource
  - Usually as child of existing Resource
  - URI of child in Location response header
- POST is powerful, don't overuse it!
  - Not safe, not idempotent
  - Cannot just resend

Idempotent

Multiple invocations have same end-result

```
POST /transfers
Host: www.mybank.com
Content-Type: application/xml

<transfer>
  …
</transfer>
```

```
HTTP/1.1 201 Created
Date: …
Content-Length: 0
Location: http://mybank.com/
  transfers/123
…
```


© Copyright 2014 Pivotal. All rights reserved.

PUT

- PUT updates a resource or creates it with a known destination URI
- Idempotent operation
  - Same request yields same result
- Not safe! (has side-effects)

```
PUT /transfers/123
Host: www.mybank.com
Content-Type: application/xml

<transfer>
  …
</transfer>
```

```
HTTP/1.1 204 No Content
Date: …
Content-Length: 0
...
```

Successful update – nothing to return

© Copyright 2014 Pivotal. All rights reserved.

Results from a **PUT**

```
PUT /transfers/123
Host: www.mybank.com
Content-Type: ...

<transfer>
...
</transfer>
```

```
HTTP/1.1 204 No Content
Date: …
Content-Length: 0
...
```

Updated existing resource

```
HTTP/1.1 201 Created
Date: …
Content-Length: 0
Location: http://...
...
```

Created new resource

© Copyright 2014 Pivotal. All rights reserved.

PATCH

- Just update *part* of a resource
  - A partial PUT
  - Not safe! (has side-effects), not idempotent
- Example below: just update amount field

```
PATCH /transfers/123
Host: www.mybank.com
Content-Type: application/xml

<transfer>
  <amount>200.00</amount>
</transfer>
```

```
HTTP/1.1 204 No Content
Date: …
Content-Length: 0
...
```


© Copyright 2014 Pivotal. All rights reserved.

DELETE

- Deletes a resource
  - Idempotent
  - Post condition is always the same
  - Not safe!

```
DELETE /transfers/123

Host: www.mybank.com

...
```

```
HTTP/1.1 204 No Content
Date: …
Content-Length: 0
...
```

Successful delete – nothing to return

© Copyright 2014 Pivotal. All rights reserved.

HTTP Methods

Method	Safe	Idempotent	Cacheable
GET	✓	✓	✓
HEAD	✓	✓	✗
OPTIONS	✓	✓	✓
POST	✗	✗	✗
PUT	✗	✓	✗
PATCH	✗	✗	✗
DELETE	✗	✓	✗

© Copyright 2014 Pivotal. All rights reserved.

Introduction To Messaging

Characteristics and Benefits of messaging-based communication

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Introduction**
- Decoupling
- Use Cases
- Enterprise Integration Patterns

© Copyright 2014 Pivotal. All rights reserved.

Introduction

- Messaging is another integration style
- Systems exchange messages via *broker*, which:
- can provide guarantees & services
  - Durability (persistent messages)
  - Atomicity (sending/receiving in single transaction)
  - Priority (some messages more important)
- can offer multiple interfaces
  - JMS driver, Stomp, AMQP, etc.
- Often called Message Oriented Middleware (MoM)
  - Available as commercial & open-source products

© Copyright 2014 Pivotal. All rights reserved.

Messages

- Messages consist of headers and payload
- Headers are metadata
  - Used for identification, routing, etc.
  - Typically key-value pairs
  - Both pre-defined by broker or standard and custom
- Payload is actual content to exchange
  - Sometimes just string or byte array
  - Some systems support typed messages
  - Often uses common data exchange format (XML, JSON, EDI, …)

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- **Decoupling**
- Use Cases
- Enterprise Integration Patterns

© Copyright 2014 Pivotal. All rights reserved.

Messaging & Decoupling

Broker decouples sender and receiver:

- Spatial
  - Don't have to be co-located to communicate
  - Other styles offer this too
- Temporal
  - No need for other system(s) to respond immediately
  - Asynchronous interaction
- Logical
  - Sender(s) don't need to know about receiver(s)

© Copyright 2014 Pivotal. All rights reserved.

Temporal Decoupling

- Asynchronous communication doesn't hold up sender
  - Just give message to the broker and resume
    - No need to block sender waiting for receiver
  - Broker buffers messages until receiver is ready
    - Receiver doesn't even need to be running!
- Allows for sending-only or request-reply
  - Replies delivered back as separate messages
  - Receiver can correlate with original request
  - a.k.a. out of band communication
    - vs. in-band like with RPC, REST or SOAP over HTTP
  - cf. email vs. phone conversation

© Copyright 2014 Pivotal. All rights reserved.

Logical Decoupling

- Senders don't target receivers directly
  - And receivers don't know about senders
- Use broker-defined destinations instead
- Allows extra logic between sender and receiver
  - Routing: determining receiver(s) for a message
  - Filtering: dropping messages
  - Transforming: changing messages
- Topologies can change without affecting existing senders/receivers

© Copyright 2014 Pivotal. All rights reserved.

Asynchronous Characteristics

- Asynchronous messaging has pros and cons
- Pros:
  - Better scalability
  - Looser coupling (temporal & logical)
- Cons:
  - Overhead caused by broker
  - Extra complexity
- For many integration use cases pros outweigh cons
  - But always consider on case-by-case basis

© Copyright 2014 Pivotal. All rights reserved.

Intra-application Integration

- Some pros apply to intra-application integration as well
  - Logically decoupled components exchanging data through in-memory messages
  - Event-driven system, no hard-coded control flow
  - Allows same patterns as with MoM within applications
- Frameworks like Spring Integration and Apache Camel enable this
  - Combined with adapters to connect to external systems

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- Decoupling
- **Use Cases**
- Enterprise Integration Patterns

© Copyright 2014 Pivotal. All rights reserved.

Use Cases

Simple producer – consumer

- Unidirectional: send message off for further processing
  - Web application places order with back-end system
  - Trigger messages, e.g. to set off a batch process
- Bi-directional: request – reply
  - Retrieve data from system with limited availability
  - Throttle messages so receiver isn't overloaded

© Copyright 2014 Pivotal. All rights reserved.

Use Cases

Pipelining

- Sending message through multiple systems
  - Order chain and other process flows
  - Content enrichment
- Often called pipes and filters architecture
  - Pipes contain the messages,
    filters are the components sending and receiving
  - cf. Unix pipes

© Copyright 2014 Pivotal. All rights reserved.

Use Cases

Message Distribution

- Same message received by multiple receivers
  - Publish-subscribe
  - Event broadcasting
- Competing receivers
  - Scalable workload distribution
  - Optimizing resource usage

© Copyright 2014 Pivotal. All rights reserved.

Use Cases

Message Aggregation

- Handling similar requests from many clients
  - Each client usually has dedicated reply destination
- Gathering events
  - Complex Event Processing (CEP)
  - Centralized logging

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- Decoupling
- Use Cases
- **Enterprise Integration Patterns**

© Copyright 2014 Pivotal. All rights reserved.

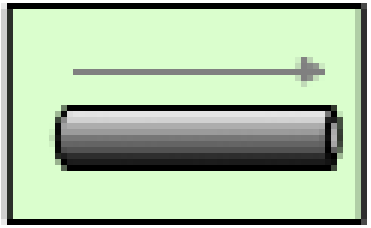
Enterprise Integration Patterns [EIP]

- Book that catalogs messaging patterns for enterprise integration architecture
  - Written by Gregor Hohpe & Bobby Woolf
  - http://eaipatterns.com/
- Standard names for common MoM concepts
  - Channel, Endpoint, etc.
- Design Patterns for architecting solutions based on these building blocks
  - Filter, Router, Splitter/aggregator, Resequencer, etc.
- Includes graphical icons for every pattern

© Copyright 2014 Pivotal. All rights reserved.

Message Channel

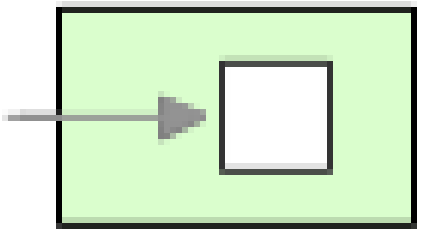
- Allows two applications to communicate
  - One application writes to the channel
  - Another reads from the channel
    - Point-to-point
  - Or multiple applications receive a copy
    - Publish-subscribe
- A channel can buffer messages



© Copyright 2014 Pivotal. All rights reserved.

Message Endpoint

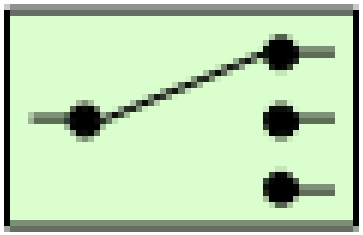
- Connects an application to a Message Channel
  - For sending or receiving
- Can take care of *polling* if necessary
  - i.e. checking for new messages
- Isolates application from the messaging system
- May wrap and unwrap messages
  - Application may only need to deal with the payload



© Copyright 2014 Pivotal. All rights reserved.

Message Router

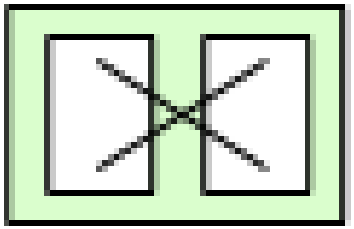
- Dynamically forwards messages to specific channel(s) based on a set of conditions
- *Content-Based* Routers use information within the Message (payload or headers) to determine channel(s) to forward to



© Copyright 2014 Pivotal. All rights reserved.

Message Translator

- Translates messages from one data format to another
- May also *enrich* content in a payload or add headers



© Copyright 2014 Pivotal. All rights reserved.

Conclusion

- Asynchronous messaging brings many benefits
  - Flexibility
  - Resilience
  - Scale
  - Performance
    - compared to e.g. file- or DB-based integration
- Integration style that applies to many scenarios
  - Even to integrating components within an application
- Well established with many mature solutions
  - But innovation still happening (e.g. AMQP / RabbitMQ)

© Copyright 2014 Pivotal. All rights reserved.

Spring JMS

Simplifying Messaging Applications

Introducing JmsTemplate and Spring's Listener Container

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Introduction to JMS**
- Apache ActiveMQ
- Configuring JMS Resources with Spring
- Spring's JmsTemplate
- Sending Messages
- Receiving Messages
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Java Message Service (JMS)

- The JMS API provides an abstraction for accessing Message Oriented Middleware
  - Avoid vendor lock-in
  - Increase portability

- JMS does *not* enable different MOM vendors to communicate
  - Need a bridge (expensive)
  - Or use AMQP (standard msg protocol, like SMTP)
    - See RabbitMQ

© Copyright 2014 Pivotal. All rights reserved.

JMS Core Components

- Message
- Destination
- Connection
- Session
- MessageProducer
- MessageConsumer

© Copyright 2014 Pivotal. All rights reserved.

JMS Message Types

- Implementations of the Message interface
  - TextMessage
  - ObjectMessage
  - MapMessage
  - BytesMessage
  - StreamMessage

© Copyright 2014 Pivotal. All rights reserved.

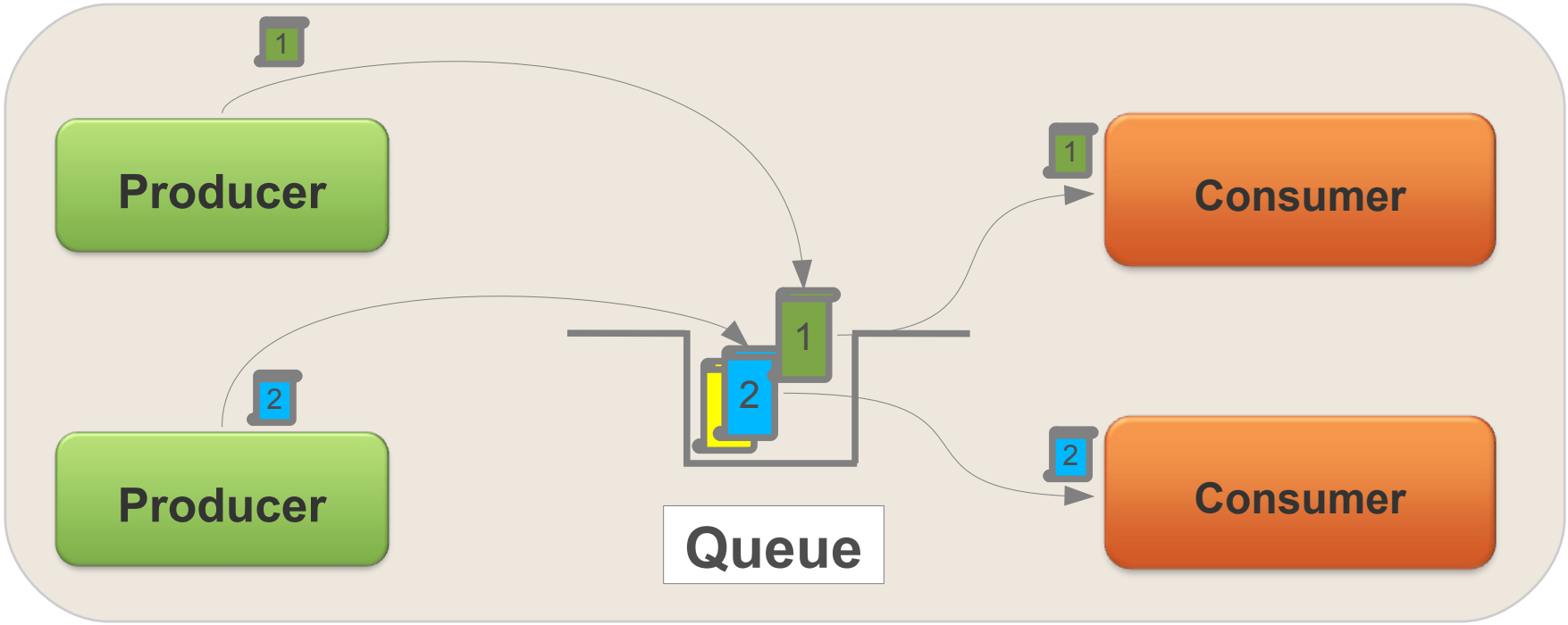
JMS Destination Types

- Implementations of the Destination interface
  - Queue
    - Point-to-point messaging
  - Topic
    - Publish/subscribe messaging
- Both support *multiple* producers and consumers
  - Messages are different
  - Let's take a closer look ...

© Copyright 2014 Pivotal. All rights reserved.

JMS Queues: Point-to-point

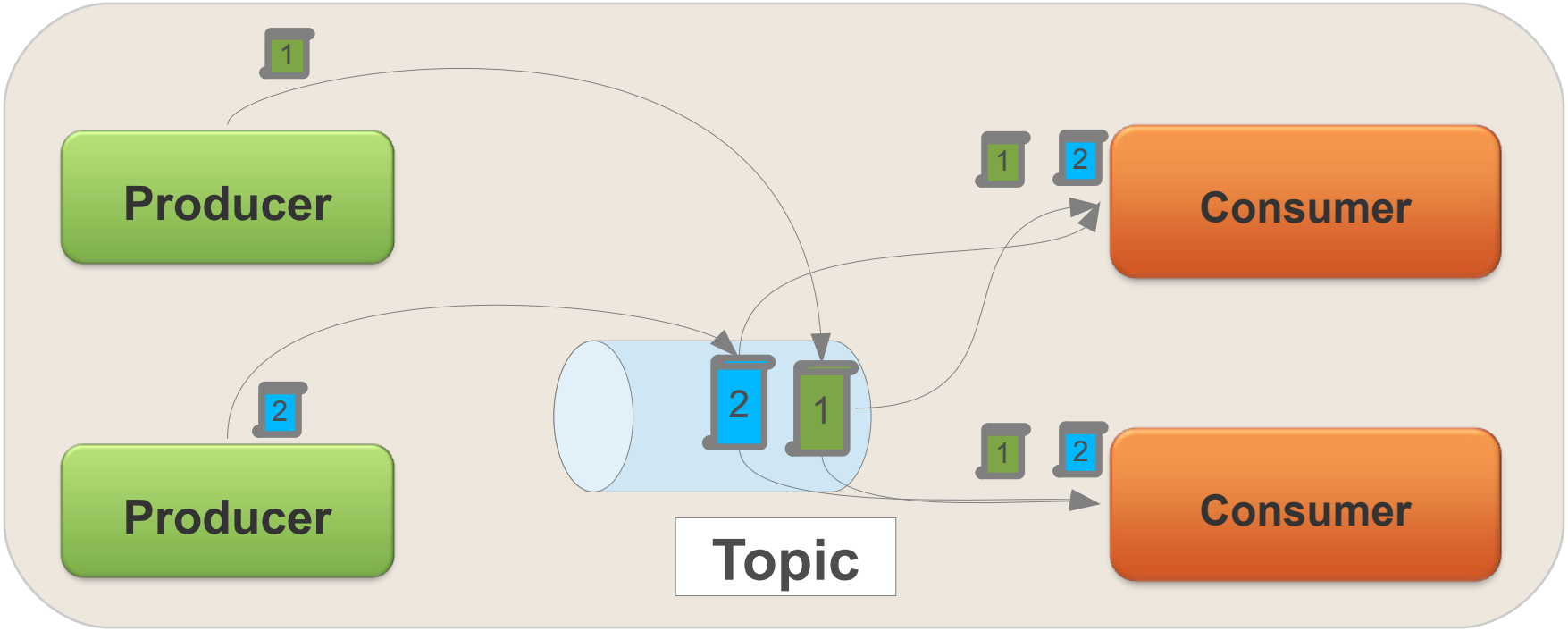
1. Message sent to queue
2. Message queued
3. Message consumed by *single* consumer



© Copyright 2014 Pivotal. All rights reserved.

JMS Topics: Publish-subscribe

1. Message sent to topic
2. Message optionally stored
3. Message distributed to *all* subscribers



© Copyright 2014 Pivotal. All rights reserved.

The JMS Connection

- A JMS Connection is obtained from a factory

```
Connection conn = connectionFactory.createConnection();
```

- Typical enterprise application:
  - ConnectionFactory is a managed resource bound to JNDI

```
Properties env = new Properties();
// provide JNDI environment properties
Context ctx = new InitialContext(env);
ConnectionFactory connectionFactory =
      (ConnectionFactory) ctx.lookup("connFactory");
```


© Copyright 2014 Pivotal. All rights reserved.

The JMS Session

- A Session is created from the Connection
  - Represents a unit-of-work
  - Provides transactional capability

```
Session session = conn.createSession(
            boolean transacted, int acknowledgeMode);

// use session
if (everythingOkay) {
    session.commit();
} else {
    session.rollback();
}
```


© Copyright 2014 Pivotal. All rights reserved.

Creating Messages

- The Session is responsible for the creation of various JMS Message types

```
session.createTextMessage("Some Message Content");

session.createObjectMessage(someSerializableObject);

MapMessage message = session.createMapMessage();
message.setInt("someKey", 123);

BytesMessage message = session.createBytesMessage();
message.writeBytes(someByteArray);
```


© Copyright 2014 Pivotal. All rights reserved.

Producers and Consumers

- The Session is also responsible for creating instances of MessageProducer and MessageConsumer

```
producer = session.createProducer(someDestination);

consumer = session.createConsumer(someDestination);
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- **Apache ActiveMQ**
- Configuring JMS Resources with Spring
- Spring's JmsTemplate
- Sending Messages
- Receiving Messages
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

JMS Providers

- Most providers of Message Oriented Middleware (MoM) support JMS
  - WebSphere MQ, Tibco EMS, Oracle EMS, JBoss AP, SwiftMQ, etc.
  - Some are Open Source, some commercial
  - Some are implemented in Java themselves
- The lab for this module uses Apache ActiveMQ

© Copyright 2014 Pivotal. All rights reserved.

Apache ActiveMQ

- Open source message broker written in Java
- Supports JMS and many other APIs
  - Including non-Java clients!
- Can be used stand-alone in production environment
  - 'activemq' script in download starts with default config
- Can also be used embedded in an application
  - Configured through ActiveMQ or Spring xml files
  - What we use in the labs

© Copyright 2014 Pivotal. All rights reserved.

Apache ActiveMQ Features

Support for:

- Many cross language clients & transport protocols
  - Incl. excellent Spring integration
- Flexible & powerful deployment configuration
  - Clustering incl. load-balancing & failover, ...
- Advanced messaging features
  - Message groups, virtual & composite destinations, wildcards, etc.
- Enterprise Integration Patterns when combined with Spring Integration or Apache Camel
  - from the book by Gregor Hohpe & Bobby Woolf

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- Apache ActiveMQ
- **Configuring JMS Resources with Spring**
- Spring's JmsTemplate
- Sending Messages
- Receiving Messages
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Configuring JMS Resources with Spring

- Spring enables decoupling of your application code from the underlying infrastructure
  - Container provides the resources
  - Application is simply coded against the API
- Provides deployment flexibility
  - use a standalone JMS provider
  - use an application server to manage JMS resources

© Copyright 2014 Pivotal. All rights reserved.

Configuring a ConnectionFactory

- ConnectionFactory may be standalone

```
<bean id="connectionFactory"
        class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>
```

- Or retrieved from JNDI

```
<jee:jndi-lookup id="connectionFactory"
    jndi-name="jms/ConnectionFactory"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Configuring Destinations

- Destinations may be standalone

```
<bean id="orderQueue"
        class="org.apache.activemq.command.ActiveMQQueue">
   <constructor-arg value="queue.order"/>
</bean>
```

- Or retrieved from JNDI

```
<jee:jndi-lookup id="orderQueue"
                 jndi-name="jms/OrderQueue"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- Apache ActiveMQ
- Configuring JMS Resources with Spring
- **Spring's JmsTemplate**
- Sending Messages
- Receiving Messages
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Spring's JmsTemplate

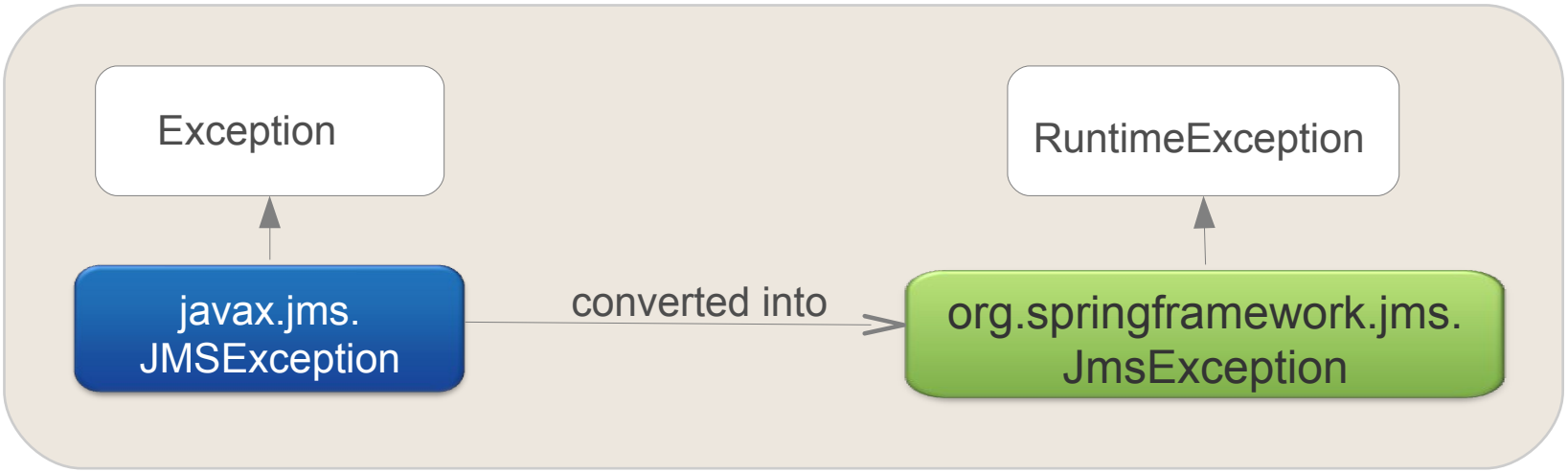
- The template simplifies usage of the API
  - Reduces boilerplate code
  - Manages resources transparently
  - Converts checked exceptions to runtime equivalents
  - Provides convenience methods and callbacks

NOTE: The *AmqpTemplate* (used with RabbitMQ) has an almost identical API to the *JmsTemplate* – they offer similar abstractions over very different products

© Copyright 2014 Pivotal. All rights reserved.

Exception Handling

- Exceptions in JMS are checked by default
- JmsTemplate converts checked exceptions to runtime equivalents



© Copyright 2014 Pivotal. All rights reserved.

JmsTemplate configuration

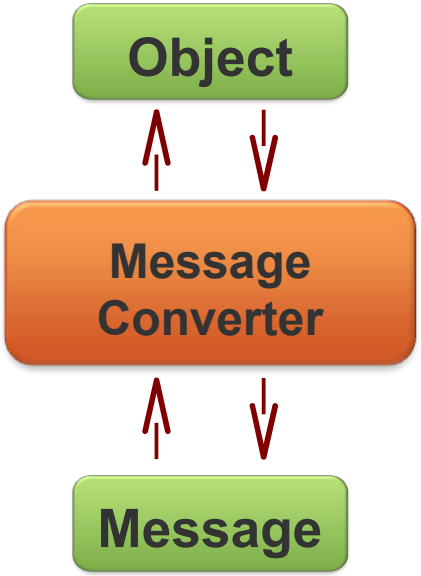
- *Must* provide reference to ConnectionFactory
  - via either constructor or setter injection
- Optionally provide delegates to handle some of the work
  - Message Converter
  - Destination Resolver

```
<bean id="jmsTemplate"
        class="org.springframework.jms.core.JmsTemplate">
   <constructor-arg ref="connectionFactory"/>
   <property name="messageConverter"  ref="...."/>
   <property name="destinationResolver"  ref="...."/>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

MessageConverter

- The JmsTemplate uses a MessageConverter to convert between objects and messages
  - You only send and receive objects
- The default SimpleMessageConverter handles basic types
  - String to TextMessage
  - Map to MapMessage
  - byte[] to BytesMessage
  - Serializable to ObjectMessage

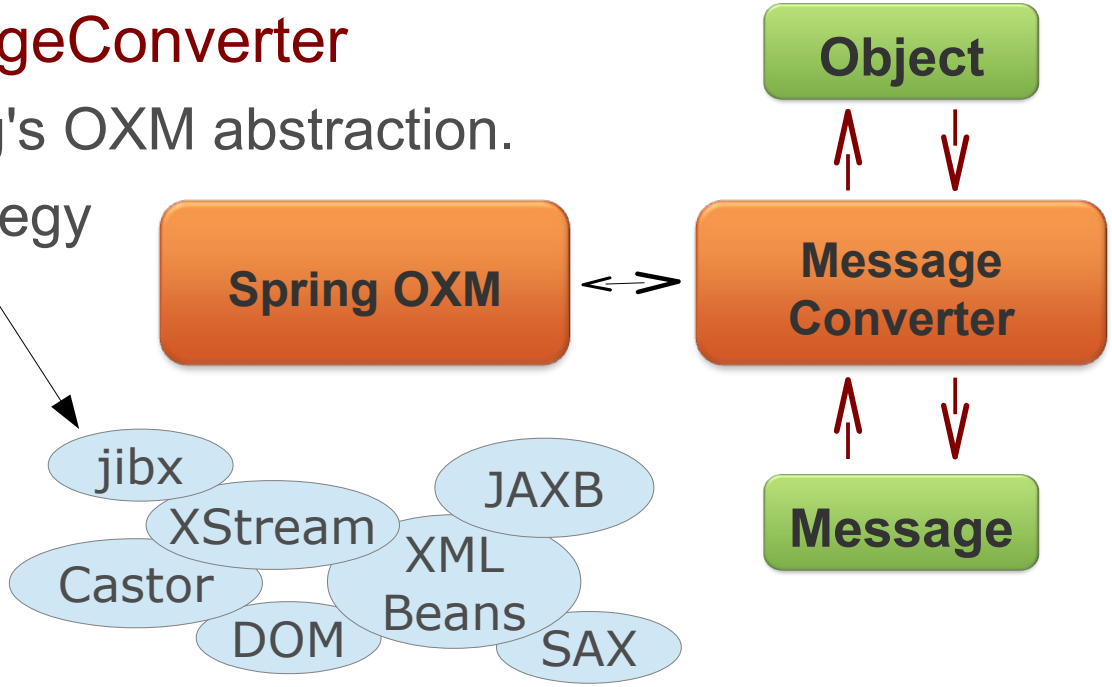


NOTE: It is possible to implement custom converters by implementing the *MessageConverter* interface

© Copyright 2014 Pivotal. All rights reserved.

XML MessageConverter

- XML is a common message payload
  - ...but there is no “XmlMessage” in JMS
  - Use TextMessage instead.
- MarshallingMessageConverter
  - Plugs into Spring's OXM abstraction.
  - You choose strategy



© Copyright 2014 Pivotal. All rights reserved.

MarshallingMessageConverter Example

```
<bean id="jmsTemplate" class="o.s.jms.core.JmsTemplate">
   <property name="connectionFactory" ref="connectionFactory"/>
   <property name="messageConverter" ref="msgConverter"/>
</bean>

<bean id="msgConverter"
         class="o.s.jms.support.converter.MarshallingMessageConverter">
   <property name="marshaller" ref="marshaller"/>
</bean>

<oxm:jaxb2-marshaller
   id="marshaller"
   contextPath="example.app.schema">
```

JAXB2 Illustrated here,
other strategies available.

© Copyright 2014 Pivotal. All rights reserved.

DestinationResolver

- Convenient to use destination names at runtime
- DynamicDestinationResolver used by default
  - Resolves topic and queue names
  - *Not* their Spring bean names
- JndiDestinationResolver also available

```
Destination resolveDestinationName(Session session,
        String destinationName,
        boolean pubSubDomain)
throws JMSException;
```

publish-subscribe?
true → Topic
false → Queue

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- Apache ActiveMQ
- Configuring JMS Resources with Spring
- Spring's JmsTemplate
- **Sending Messages**
- Receiving Messages
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Sending Messages

- The template provides options
  - Simple methods to send a JMS message
  - One line methods that leverage the template's MessageConverter
  - Callback-accepting methods that reveal more of the JMS API
- Use the simplest option for the task at hand

© Copyright 2014 Pivotal. All rights reserved.

Sending POJO

- A message can be sent in one single line

```
public class JmsOrderManager implements OrderManager {
  @Autowired private JmsTemplate jmsTemplate;
  @Autowired private Destination orderQueue;

  public void placeOrder(Order order) {
    String stringMessage = "New order " + order.getNumber();
    jmsTemplate.convertAndSend("messageQueue", stringMessage );
                                    // use destination resolver and message converter

    jmsTemplate.convertAndSend(orderQueue, order);   // use message converter

    jmsTemplate.convertAndSend(order);   // use converter and default destination
  }
}
```

No @Qualifier so Destination is wired by *name*

© Copyright 2014 Pivotal. All rights reserved.

Sending JMS Messages

- Useful when you need to access JMS API
  - eg. set expiration, redelivery mode, reply-to ...

```
public void sendMessage(final String msg) {
   MessageCreator messageCreator = new MessageCreator() {

     public Message createMessage(Session session) throws JMSException {
       TextMessage message = session.createTextMessage(msg);
       message.setJMSExpiration(2000);    // 2 seconds
       return message;
     }
   };

   this.jmsTemplate.send(messageCreator);
}
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- Apache ActiveMQ
- Configuring JMS Resources with Spring
- Spring's JmsTemplate
- Sending Messages
- **Receiving Messages**
- Advanced Features

© Copyright 2014 Pivotal. All rights reserved.

Synchronous Message Reception

- JmsTemplate can also receive messages, but methods are blocking (with optional timeout)
  - receive()
  - receive(Destination destination)
  - receive(String destinationName)
- The MessageConverter can be leveraged for message reception as well

```
Object someSerializable =
            jmsTemplate.receiveAndConvert(someDestination);
```


© Copyright 2014 Pivotal. All rights reserved.

The JMS MessageListener

- The JMS API defines this interface for asynchronous reception of messages

```
import javax.jms.Message;
import javax.jms.MessageListener;

public class OrderListener implements MessageListener {
  public void onMessage(Message message) { // ...      }
}
```

- Traditionally, using a MessageListener required an EJB container
  - Message Driven Beans

© Copyright 2014 Pivotal. All rights reserved.

Spring's MessageListener Containers

- Spring provides lightweight alternatives
  - SimpleMessageListenerContainer
    - Uses plain JMS client API
    - Creates a fixed number of Sessions
  - DefaultMessageListenerContainer
    - Adds transactional capability
- Advanced scheduling and endpoint management options available for each container option

© Copyright 2014 Pivotal. All rights reserved.

Defining a plain JMS Message Listener

- Define listeners using jms:listener elements

```
<jms:listener-container connection-factory="myConnectionFactory">
    <jms:listener destination="queue.order" ref="myOrderListener"/>
    <jms:listener destination="queue.conf" ref="myConfListener"/>
</jms:listener-container>
```

- Listener needs to implement MessageListener
  - or SessionAwareMessageListener
- jms:listener-container is configurable
  - task execution strategy, concurrency, container type, transaction manager and more

© Copyright 2014 Pivotal. All rights reserved.

Spring's Message-Driven POJO

- Spring also allows you to specify a plain Java object that can serve as a listener
  - MessageConverter provides parameter
  - Any return value sent to response-destination after conversion

```
public class OrderService {    1
        public OrderConfirmation order(Order o) {}
}                              3                2
```

```
<jms:listener
        ref="orderService"      1
        method="order"  2
        destination="queue.orders"                    3
        response-destination="queue.confirmation"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Messaging: Pros and Cons

- Advantages
  - Resilience, guaranteed delivery
  - Asynchronous support
  - Application freed from messaging concerns
  - Interoperable – languages, environments
- Disadvantages
  - Requires additional third-party software
    - Can be expensive to install and maintain
  - More complex to use – *but not with JmsTemplate!*

© Copyright 2014 Pivotal. All rights reserved.

Lab

Sending and Receiving Messages in a Spring Application

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to JMS
- Apache ActiveMQ
- Configuring JMS Resources with Spring
- Spring's JmsTemplate
- Sending Messages
- Receiving Messages
- **Advanced Features**

© Copyright 2014 Pivotal. All rights reserved.

SessionCallback Example: Synchronous Request-Reply

```
RewardConfirmation confirmation =
    jmsTemplate.execute(new SessionCallback() {

  public Object doInJms(Session session) throws JMSException {
      TemporaryQueue replyQueue = session.createTemporaryQueue();
      Message request = session.createObjectMessage(dining);
      request.setJMSReplyTo(replyQueue);
      MessageProducer producer = session.createProducer(diningQueue);
      producer.send(request);

      MessageConsumer consumer = session.createConsumer(replyQueue);
      ObjectMessage reply = (ObjectMessage) consumer.receive(60000);
      replyQueue.delete();  // delete here, since Connection might be cached
      return reply != null ? (RewardConfirmation) reply.getObject : null;
  }
});
```


© Copyright 2014 Pivotal. All rights reserved.

Advanced Option: CachingConnectionFactory

- JmsTemplate aggressively closes and reopens resources like Sessions and Connections
  - Normally these are cached by connection factory
  - Without caching can cause lots of overhead
    - Resulting in poor performance
- Use our CachingConnectionFactory to add caching within the application if needed

```
<bean id="connectionFactory"
        class="org.springframework.jms.connection.CachingConnectionFactory">
   <property name="targetConnectionFactory">
      <bean class="org.apache.activemq.ActiveMQConnectionFactory">
         <property name="brokerURL" value="vm://embedded?broker.persistent=false"/>
      </bean>
   </property>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

JMS Transactions

Transactional Messaging Applications with Spring

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Why use JMS transactions**
- JMS Session as Unit of Work
- Overview of transactional options
- Transactional JMS Resources with Spring
- Duplicate Message Handling

© Copyright 2014 Pivotal. All rights reserved.

Why use JMS Transactions

- Interaction with JMS is by definition stateful
- Adds or removes messages from a destination
- In composite stateful interactions we need to account for ACID principles
  - Prevent Partial failures or Duplication
- Transactions is one way to do it

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Why use JMS transactions
- **JMS Session as Unit of Work**
- Overview of transactional options
- Transactional JMS Resources with Spring
- Duplicate Message Handling

© Copyright 2014 Pivotal. All rights reserved.

The JMS Session

- JMS Session acts as unit of work
  - Keeps track of the performed operations
- Created through the JMS Connection

```
public interface Connection {
  Session createSession(boolean transacted, int acknowledgeMode)
    throws JMSException;
  // …
```

- Pass in true for transacted Session
  - Call commit() or rollback() to end transaction
  - All JMS operations must use the same Session!
- Acknowledge mode ignored for transacted Session

© Copyright 2014 Pivotal. All rights reserved.

Acknowledge Mode

- Indicates when message is considered successfully delivered outside a transaction
- One of the following:
  - AUTO_ACKNOWLEDGE
    after every successful **`receive()`** or **`onMessage()`**
  - CLIENT_ACKNOWLEDGE
    client has to call **`Message.acknowledge()`** itself
  - DUPS_OK_ACKNOWLEDGE
    lazily acknowledge delivery
- Defined as **`int`** constants on **`Session`**

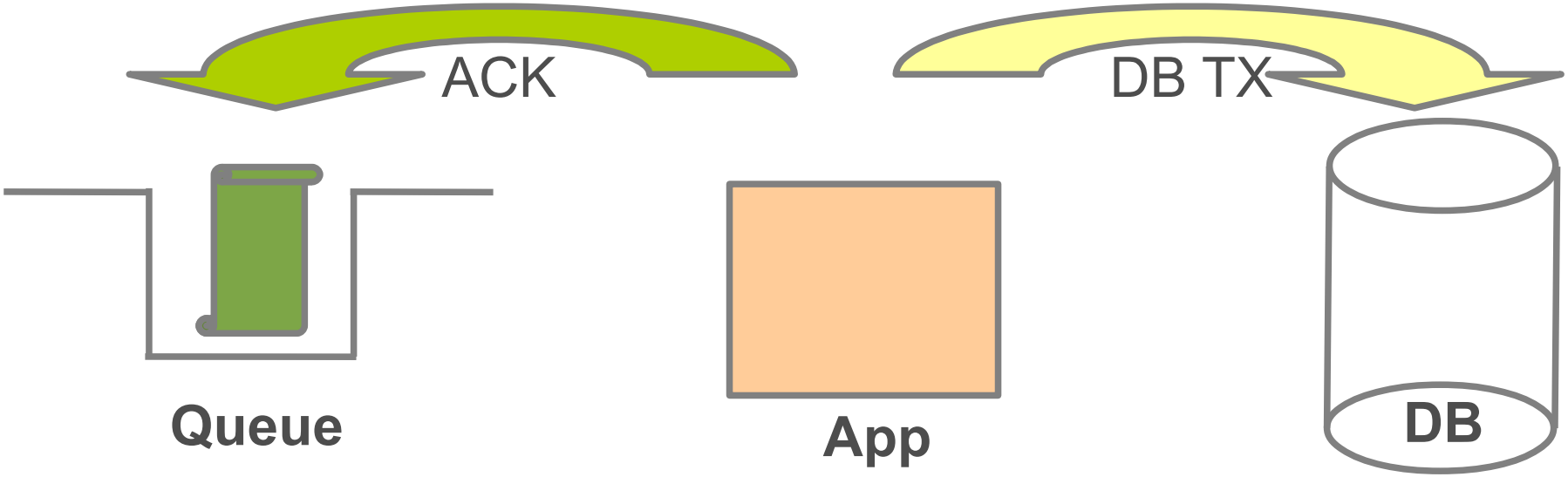
© Copyright 2014 Pivotal. All rights reserved.

AUTO_ACKNOWLEDGE

- Acknowledge after every message delivery
  - Used as default in Spring without further configuration
- Means successful reception will always mark a message as delivered
  - So it is removed from the queue
  - Message will not be redelivered, so no duplicates
- Can be slower than other acknowledge modes
  - Acknowledgement sent after every delivered message!
- Results in message loss if processing after reception fails!

© Copyright 2014 Pivotal. All rights reserved.

AUTO_ACKNOWLEDGE
Happy Case Scenario

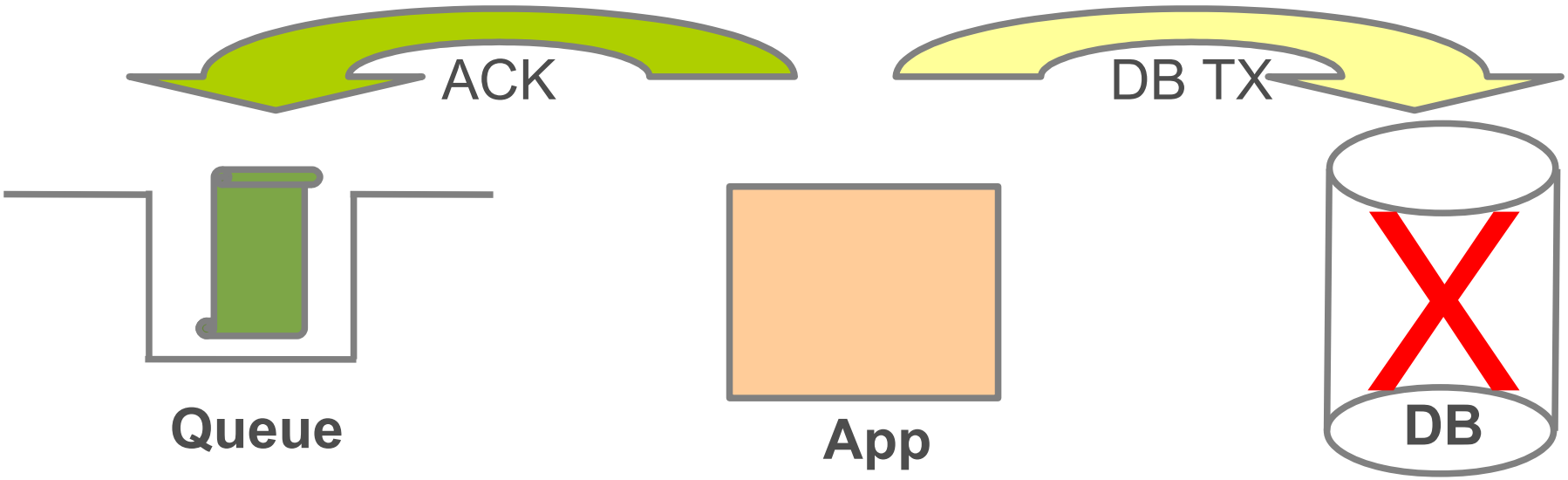


1. Receive & ack message
2. DB TX starts
3. Store message data
4. DB commits

Message in DB, no longer in Queue

© Copyright 2014 Pivotal. All rights reserved.

AUTO_ACKNOWLEDGE
Message Loss On Error



1. Receive & ack message
2. DB TX starts
3. Store message data: error!
4. DB rolls back

Message not in DB and no longer in Queue: lost!

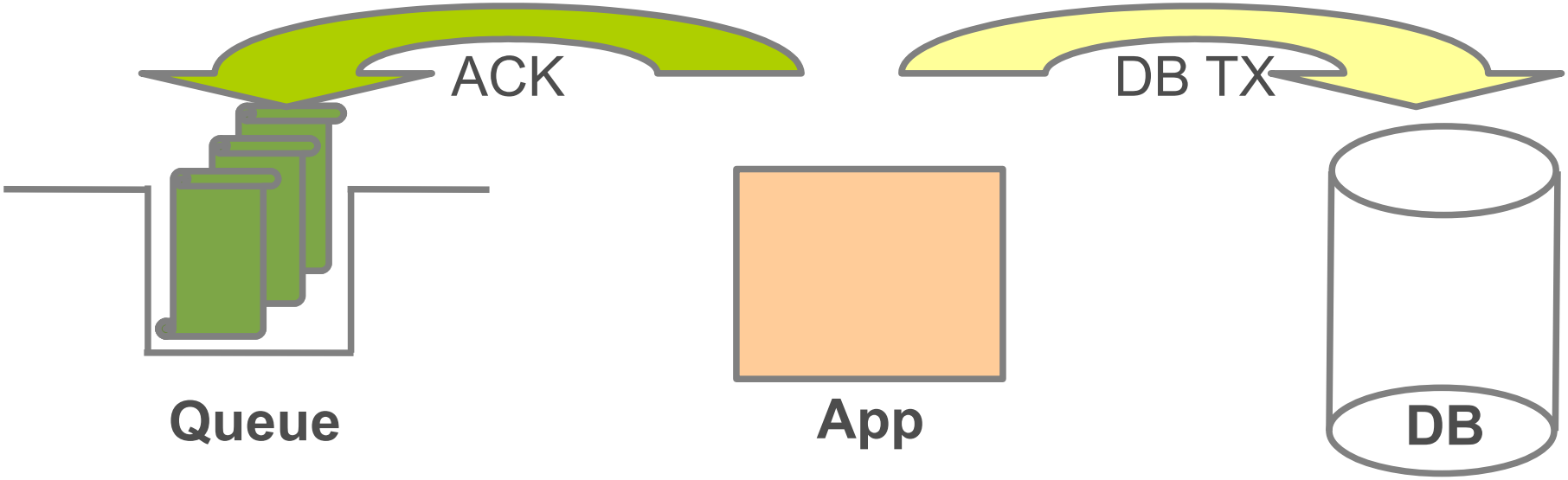
© Copyright 2014 Pivotal. All rights reserved.

CLIENT_ACKNOWLEDGE

- Client takes responsibility for acknowledging
  - Requires manually calling **`Message.acknowledge()`**
- Acknowledge multiple messages after processing
  - **`Message.acknowledge()`** acts on *all* consumed messages of current Session!
  - Optimize nr. of calls MessageConsumer makes to broker
- If client fails before acknowledging, messages will be redelivered
  - Requires call to **`Session.recover()`** for same Session
  - Results in *duplicates* if processing succeeded already!

© Copyright 2014 Pivotal. All rights reserved.

CLIENT_ACKNOWLEDGE
Happy Case Scenario

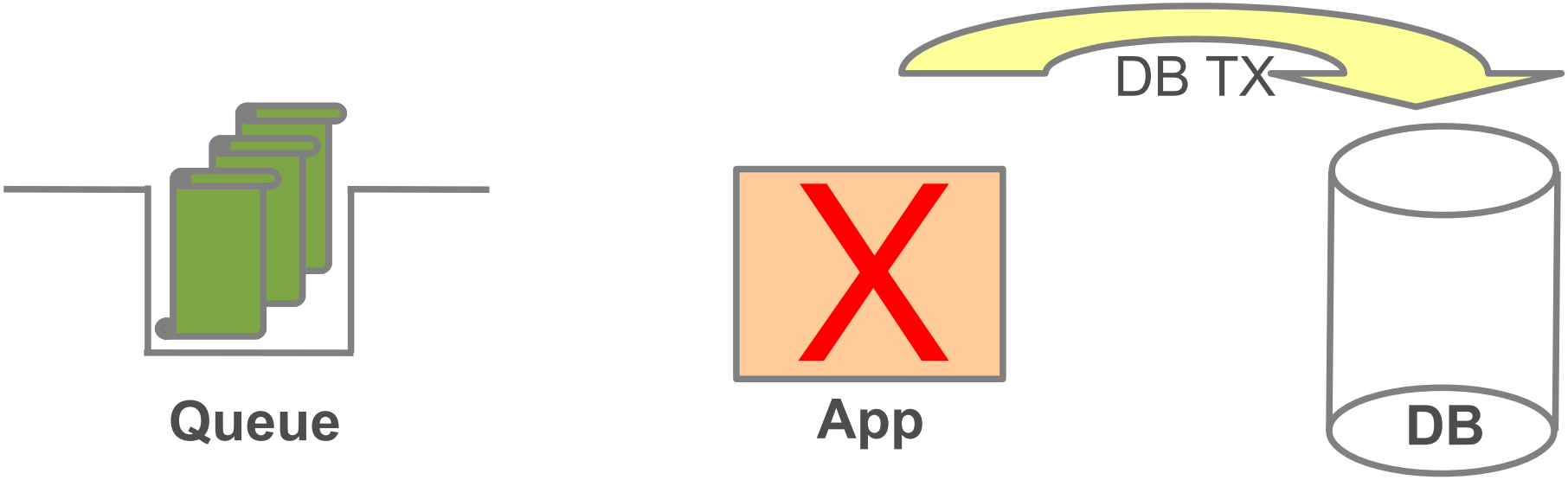


1. Receives
2. DB TX starts
3. Store message data
4. DB commits
5. Messages acknowledged

Messages in DB, no longer in Queue

© Copyright 2014 Pivotal. All rights reserved.

CLIENT_ACKNOWLEDGE
Duplicates On Error



1. Receives
2. DB TX starts
3. Store message data
4. DB commits
5. Error in app before ack

Messages in DB and still in Queue: will result in duplicate messages

© Copyright 2014 Pivotal. All rights reserved.

DUPS_OK_ACKNOWLEDGE

- Message delivery is lazily acknowledged
  - Not directly after successful delivery!
  - But still automatically, i.e. not manually by client code
- Can result in duplicates
  - If JMS provider fails before acknowledgement
  - Same as with CLIENT_ACKNOWLEDGE really
  - Means application needs to be able to handle this
  - Good for idempotent consumers
- May result in better performance than auto-ack
  - Lowers overhead for Session to prevent duplicates and for MessageConsumer to communicate with broker

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Why use JMS transactions
- JMS Session as Unit of Work
- **Overview of transactional options**
- Transactional JMS Resources with Spring
- Duplicate Message Handling

© Copyright 2014 Pivotal. All rights reserved.

Flavors of JMS Transactions

1) No transaction
   - Acknowledge mode applies

2) Local transaction
   - Pass **`true`** to **`Connection.createSession(boolean, int)`** to start new transaction
   - Messages acknowledged on TX commit

3) Global transaction (next chapter)
   - Multiple transactional resources managed together.
   - Values passed to **`createSession`** are ignored

© Copyright 2014 Pivotal. All rights reserved.

Local JMS Transactions

- Only apply to JMS
  - Ignores other transactional resources
  - (Transactions spanning multiple resources requires global)

- Several 'best effort' strategies will work without and could be sufficient:
  - Commit database before JMS
    - Never loses messages
    - Causes duplicate messages if JMS commit fails, like with client-ack/dups_ok
  - Put commits close together to reduce failure opportunity
    - no *application* errors possible after 1st commit, only external errors

© Copyright 2014 Pivotal. All rights reserved.

Spring Namespace Conventions

- Transacted Session and Acknowledge Mode are mutually exclusive in JMS...
- ...So Spring Namespace specifies them in a single attribute called **`acknowledge`**:
  - Can be **`auto`**, **`client`** or **`dups_ok`** for acknowledge mode
  - Or **`transacted`** for transacted session
- **`transacted`** starts local JMS Transaction
  - Assuming no global transaction is in progress
  - Will be synchronized with existing transaction

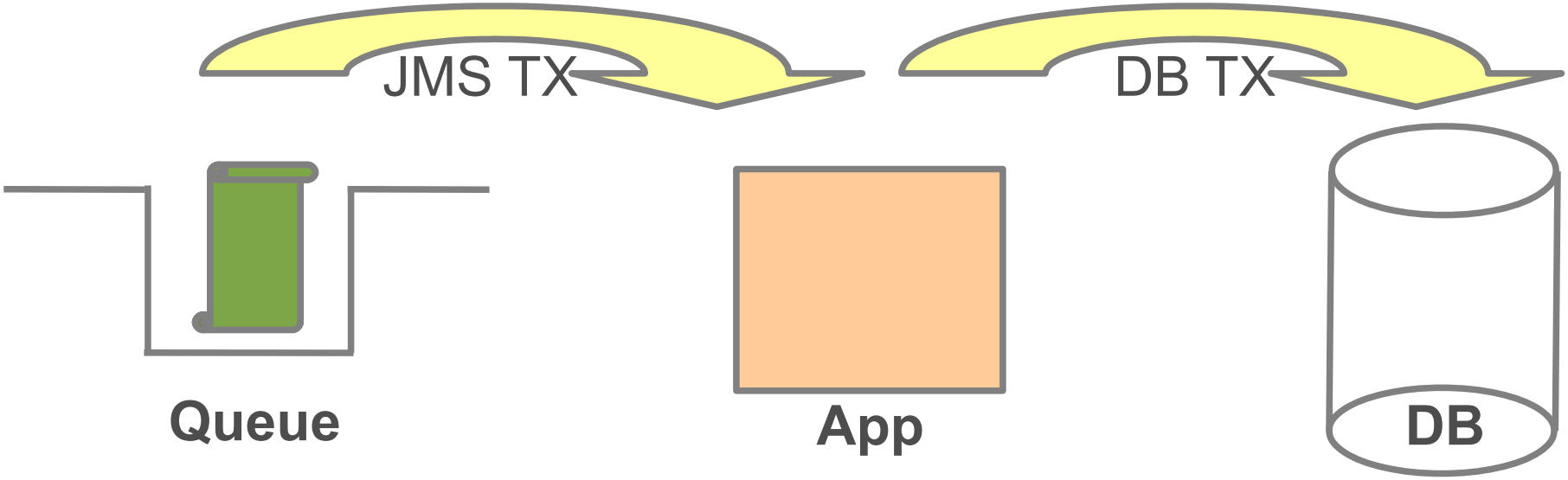
© Copyright 2014 Pivotal. All rights reserved.

Participating in a Transaction

- **`acknowledge="transacted"`** is often enough
  - Assuming listener container started the TX
  - JmsTemplate will automatically use the same Session
  - Synchronizes with TX from other local TX manager (e.g. DataSourceTransactionManager)
    - Commit Session only *after* successful database commit
- Spring also provides **`JmsTransactionManager`** for when **`acknowledge="transacted"`** cannot be used or isn't sufficient with local JMX TXs

© Copyright 2014 Pivotal. All rights reserved.

JMS+DB TX Synchronization: Happy Case Scenario

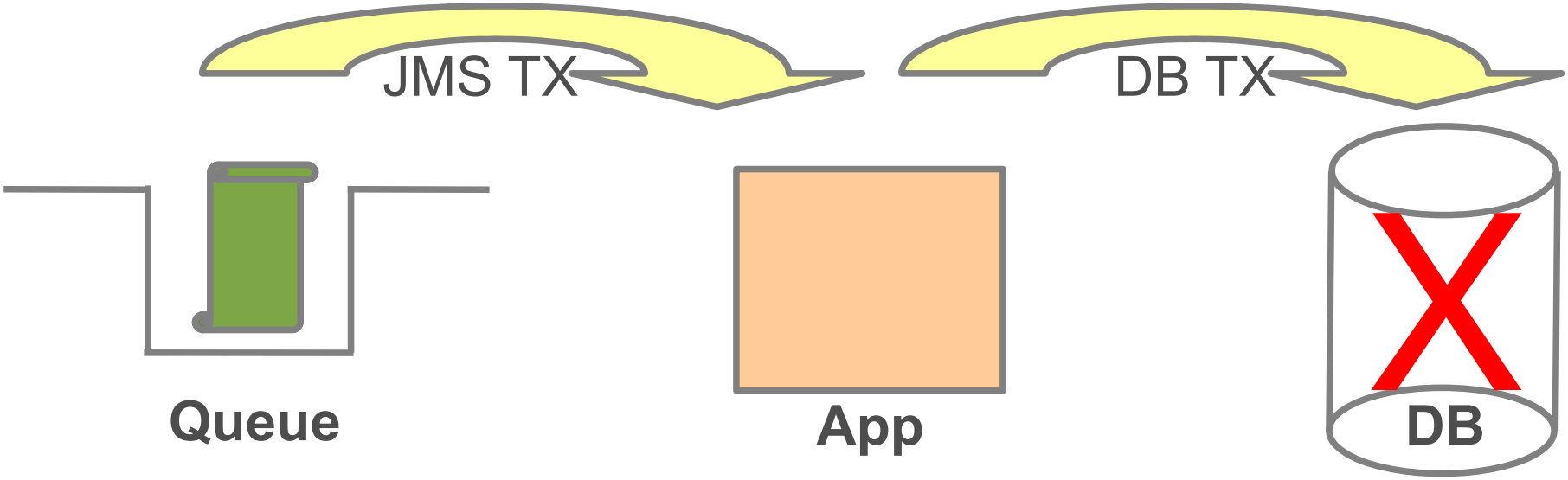


1. JMS TX starts
2. Receive message
3. DB TX starts
4. Store message data
5. DB TX commits
6. JMS TX commits

Message in DB, no longer in Queue

© Copyright 2014 Pivotal. All rights reserved.

JMS+DB TX Synchronization: DB Rollback

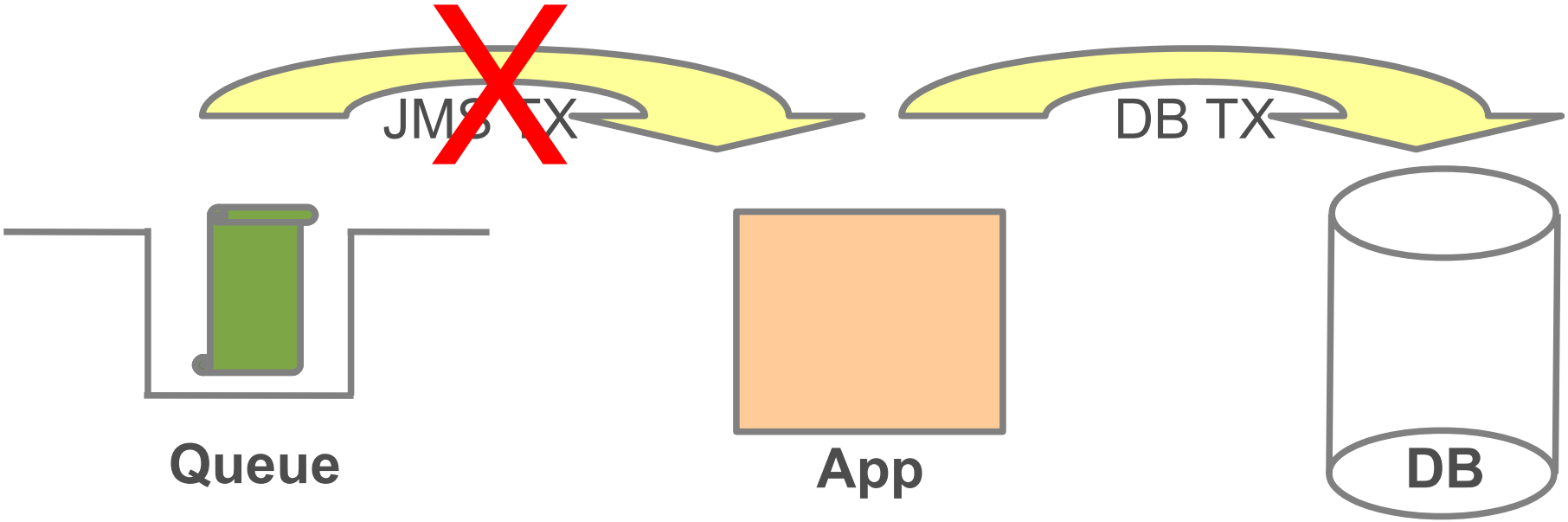


1. JMS TX starts
2. Receive message
3. DB TX starts
4. Store message data: error!
5. DB TX rolls back
6. JMS TX rolls back

Message not in DB, still in Queue

© Copyright 2014 Pivotal. All rights reserved.

JMS+DB TX Synchronization: JMS Commit Failure



1. JMS TX starts
2. Receive message
3. DB TX starts
4. Store message data
5. DB TX commits
6. JMS TX fails to commit
   → rollback

Message in DB and still in Queue: will result in duplicate message

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Why use JMS transactions
- JMS Session as Unit of Work
- Overview of transactional options
- **Transactional JMS Resources with Spring**
- Duplicate Message Handling

© Copyright 2014 Pivotal. All rights reserved.

Transacted Listener Container

- Specify acknowledge attribute
  - **`transacted`** gives you local JMS transaction

```
<jms:listener-container acknowledge="transacted">
    <jms:listener … />
</jms:listener-container>
```

- Or specify a transaction manager
- Delegates to specified transaction manager
  - Typically global, could also be JMS

```
<jms:listener-container transaction-manager="transactionManager">
    <jms:listener … />
</jms:listener-container>
```


© Copyright 2014 Pivotal. All rights reserved.

Listener Container Transactions

- Transaction starts when message is received
- Setting the listener container's transaction manager to a global transaction manager enables global transactions.
  - Always consider if you really need global
  - Necessary if once-and-once-only delivery must be guaranteed at all times
  - If application can handle duplicates, two local TXs are much faster and easier to set up

© Copyright 2014 Pivotal. All rights reserved.

Transacted JmsTemplate

- Specify default modes for native Sessions
- Will be ignored if the Session was created within an active transaction already

```
<bean class="...JmsTemplate">
    <property name="sessionTransacted" .../>
    <property name="sessionAcknowledgeMode" .../>
</bean>
```

- Don't specify acknowledge mode for transacted sessions (will be ignored)

© Copyright 2014 Pivotal. All rights reserved.

JmsTemplate Usage

- Once set up nothing changes in the usage
- Read and write operations use the same Session when transaction is in progress
  - Obtained through ConnectionFactoryUtils.doGetTransactionalSession
- Transactional behavior is defined in configuration

```
// participates in the Session and therefore the transaction
jmsTemplate.convertAndSend(order);
```

```
// participates in the Session and therefore the transaction
jmsTemplate.receiveAndConvert();
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Why use JMS transactions
- JMS Session as Unit of Work
- Overview of transactional options
- Transactional JMS Resources with Spring
- **Duplicate Message Handling**

© Copyright 2014 Pivotal. All rights reserved.

Dealing With Duplicates

- Coordinating local transactions can cause duplicate messages in many scenarios
  - Only global guarantees once-and-once-only delivery
- Not always a problem
  - If processing is idempotent, just process it again
  - Otherwise, recognize that you already processed the message and ignore it
  - This is recommended if this suffices: faster & easier
- But how do you recognize a duplicate?

© Copyright 2014 Pivotal. All rights reserved.

Detecting Duplicates

- First check if message is a redelivery

```
if (message.getJMSRedelivered()) { … }
```

- If not, just process it
- If so, check if you processed it already
  - Query for a result based on the message payload

```
String orderNumber = extractOrderNumber(message);
if (existingOrder(orderNumber)) { ... }
```

  - You cannot tell this by looking at the message!
    - Could have been rolled back before *or* after processing
- Redeliveries are rare, so overhead of extra check is not incurred often

© Copyright 2014 Pivotal. All rights reserved.

Summary

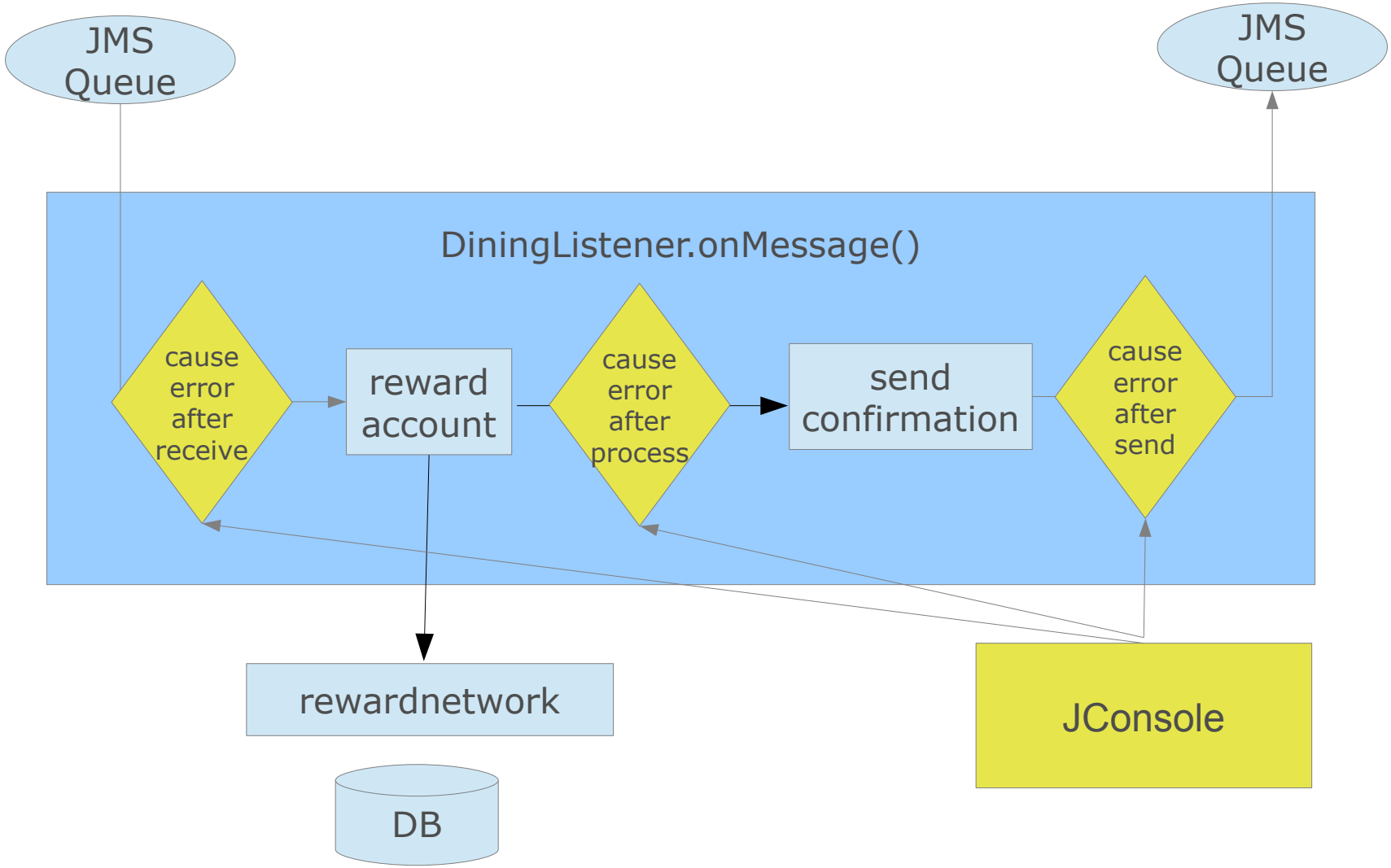
- To avoid duplicating or losing messages transactions can be used
- JMS and database transactions need to be synchronized
  - Prevents losing messages
  - Only global transaction management completely avoids duplicates
- This is never foolproof
- Think about failure scenarios carefully

© Copyright 2014 Pivotal. All rights reserved.

Lab

Resolving transactional issues with Spring JMS

© Copyright 2014 Pivotal. All rights reserved.



© Copyright 2014 Pivotal. All rights reserved.

Global transactions using XA, JTA and Spring

Enabling ACID transactions across multiple transaction resources

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Introduction**
- Two Phase Commit and XA
- JTA and Spring
- Transaction Demarcation

© Copyright 2014 Pivotal. All rights reserved.

Introduction

- Transactions manage changes made to stateful resources
  - Databases, message queues, etc.
- Provide ACID guarantees for all operations performed in a single transaction
  - Atomic
  - Consistent
  - Isolated
  - Durable
- Two ways to manage: *local* or *global*

© Copyright 2014 Pivotal. All rights reserved.

Local Transactions

- Manage TXs on the resource itself
  - JDBC Connection, JMS Session, …
  - AKA *Native* transaction
- Best solution if TX involves a single resource
  - Works in all environments
  - Very fast
- Can cause issues with multiple resources
  - ACID properties no longer guaranteed
  - Something needs to coordinate the TX across resources

© Copyright 2014 Pivotal. All rights reserved.

Global Transactions

- Use a dedicated **transaction manager**
  - Allows for ACID guarantees with multiple transactional resources in a single TX
  - Coordinates TXs across all participating resources
- Sometimes called *distributed* transaction
  - Same term is also used for propagating TX contexts to other transaction managers
  - We'll use *global* in this presentation to avoid confusion

© Copyright 2014 Pivotal. All rights reserved.

Committing a Global TX

- With multiple resources, TXs can't simply commit in one step
  - Subsequent resources might fail to commit after the first resource successfully commits
  - Prevents ACID guarantees for entire TX
- This means global transaction committing needs multiple, coordinated steps

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- **Two Phase Commit and XA**
- JTA and Spring
- Transaction Demarcation

© Copyright 2014 Pivotal. All rights reserved.

Two Phase Commit

- Global TXs use **Two Phase Commit** (2PC)
- Driven by the transaction manager
- Phase 1: First check with all participating resources if they're ready to commit (*prepare*)
- Phase 2: Only if they all are, do the commit
- Otherwise perform a rollback

© Copyright 2014 Pivotal. All rights reserved.

XA

- 2PC requires a common interface between transaction manager and the resources
  - How else can they communicate?
- The common standard to do this is **XA**
- Specification from the X/Open group
- Very complicated
  - You don't need to read the spec to use global TXs
  - You do need to understand the basics, though!

© Copyright 2014 Pivotal. All rights reserved.

XA Commit Process

- TX manager performs the following operations on participating resources:
  1) start(Xid): join the TX
  2) end(Xid): no more TX work for this resource
  3) prepare(Xid): is resource ready to commit?
  4) commit(Xid): perform the actual commit
- where Xid is transaction id for a resource
  - Contains global TX id, same for all resources in TX
  - Also has *branch qualifier*: part of the TX that the resource needs to care about

© Copyright 2014 Pivotal. All rights reserved.

XA Aware Resources

- This means you need two things:
  - An XA-aware transaction manager
  - XA-aware transactional resources
- First is provided by JTA transaction manager
  - We'll discuss JTA in a moment
- Second needs to be provided by resource's vendor
  - Often in different class than used with non-XA TXs

© Copyright 2014 Pivotal. All rights reserved.

Two Phase Commit Failures

- 2PC presents some hard cases to deal with:
  - Resource becomes unavailable after 1st phase
  - Transaction manager itself fails during execution
- Transaction manager is ultimately responsible
- Keeps detailed log on disk while TX in progress
  - As do the XA resources themselves!
- Allows TX to be *recovered* later
  - Guarantees consistent state between all resources
  - At the cost of a *lot* of additional overhead
  - XA Specification covers this in detail

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- Two Phase Commit and XA
- **JTA and Spring**
- Transaction Demarcation

© Copyright 2014 Pivotal. All rights reserved.

JTA

- Java Transaction API
- Allows global transactions using XA in Java
  - JTA can also be used without XA
  - But XA requires JTA!
- JTA support built-in with J2EE / Java EE servers
  - Stand-alone implementations exist as well
    - e.g.: Atomikos, Bitronix, JOTM, Narayana (JBoss Transactions)
  - Prefer built-in if available
- Typically not used by application code directly

© Copyright 2014 Pivotal. All rights reserved.

Using JTA

- Required if you use EJBs for TX management
  - Even if you don't need global TXs
  - Means Container-Managed Transactions always incur JTA overhead
- Optional if you use Spring
  - Just use when you need global TXs
  - Switching from local to global is easy

© Copyright 2014 Pivotal. All rights reserved.

Using JTA with Spring

- JtaTransactionManager is part of Spring's transaction management support
- Integrates with external JTA TX manager
  - Does NOT provide JTA support by itself!
- Used instead of local implementation of PlatformTransactionManager
- Doesn't require any code changes and very little configuration changes

© Copyright 2014 Pivotal. All rights reserved.

Configuring JTA with Spring

- Within a Java EE server, just ask Spring to look up the JTA TX manager:

```
<tx:jta-transaction-manager />
```

- Creates bean called transactionManager
- Uses server-specific subclass if present
  - Allows things not in JTA spec, like TX suspension
- Combined with JNDI lookups for TX resources

```
<jee:jndi-lookup id="dataSource"
            jndi-name="java:comp/env/jdbc/myDS" />
<jee:jndi-lookup id="connectionFactory"
            jndi-name="java:comp/env/jms/myConnFact" />
```


© Copyright 2014 Pivotal. All rights reserved.

Configuring JTA with Spring

- For stand-alone applications, define plain bean:

```
<bean id="transactionManager"
      class="org.springframework.tx.jta.JtaTransactionManager">
   <property name="transactionManager" ref="jtaTxMgr"/>
   <property name="userTransaction" ref="userTx"/>
</bean>
```

- Note: 'transactionManager' as id is a best practice
  - allows many Spring-components to find the TX manager automatically without wiring instructions, esp. when using namespace support

© Copyright 2014 Pivotal. All rights reserved.

Configuring TX Resources

- For JDBC or JMS, using XA-aware types is typically enough
  - Not configured in application with real app server
- Other code may need instructions to find JTA TX manager (note: this is *not* Spring-specific!)

```
<bean id="sessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
 <property name="hibernateProperties">
  <value>
   hibernate.transaction.factory_class = \
      org.hibernate.transaction.JTATransactionFactory
   hibernate.transaction.manager_lookup_class = \
      org.hibernate.transaction.WeblogicTransactionManagerLookup
   hibernate.current_session_context_class = jta
  </value>
 ...
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- Two Phase Commit and XA
- JTA and Spring
- **Transaction Demarcation**

© Copyright 2014 Pivotal. All rights reserved.

Transaction Demarcation

- Indicating *where* TXs should start / stop
- Typically done declaratively with Spring
  - using @Transactional annotations or AOP pointcuts
- Not directly related to *how* TXs need to be managed! (local or global)
  - Orthogonal concerns
- Means that with Spring, global TXs don't require code changes

© Copyright 2014 Pivotal. All rights reserved.

TX Demarcation in Spring for JTA

- For databases or synchronous JMS, just use @Transactional with <tx:annotation-driven /> or use <tx:advice>
- For asynchronous JMS, pass JTA transaction manager to listener container:

```
<jms:listener-container transaction-manager="transactionManager">
   <jms:listener ref="jmsListener" destination="queue.name"/>
</jms:listener-container>
```

- Note: for local TXs you would use acknowledge="transacted" instead

© Copyright 2014 Pivotal. All rights reserved.

Summary

- Global TXs enable multiple resources to participate in a single transaction
- Requires JTA transaction manager and XA-aware resources
- Spring supports JTA, but doesn't require it
- Just a matter of changing configuration
  - use Spring's JtaTransactionManager
  - may also need to reconfigure resources like Hibernate
  - no code changes necessary

© Copyright 2014 Pivotal. All rights reserved.

Lab

tx-xa-1: Combining JMS and JDBC operations in a global transaction

© Copyright 2014 Pivotal. All rights reserved.

Introduction to Spring Integration

Enterprise Integration Patterns with Spring Integration

© Copyright 2014 Pivotal. All rights reserved.

Topics in this session

- **Spring Integration**
  - **Goals and Concepts**
- Spring Integration
  - Basics
  - External integration
  - Visual Editor (STS)
- Summary

© Copyright 2014 Pivotal. All rights reserved.

Spring Integration

© Copyright 2014 Pivotal. All rights reserved.

Spring Integration

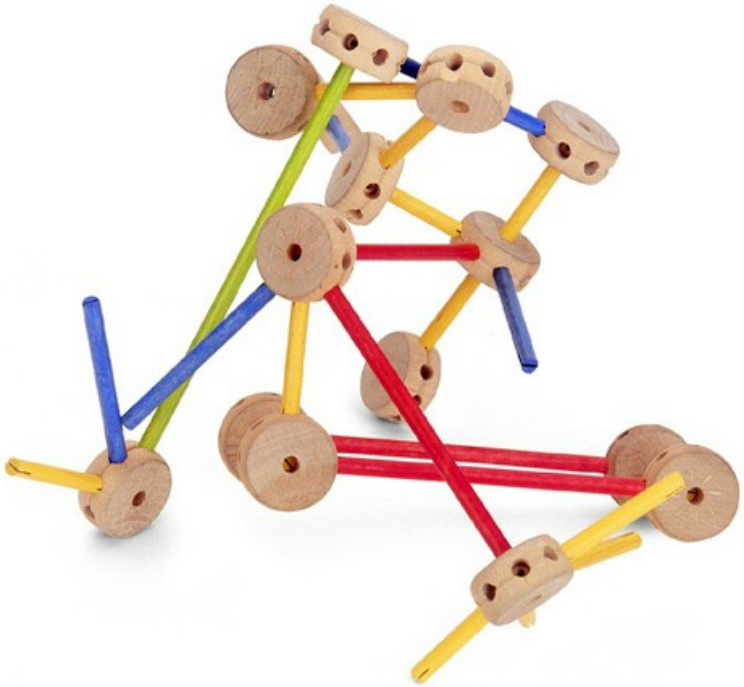
SPRING INTEGRATION

- Spring Integration allows you to
  - Let application components exchange data through in-memory messaging
  - Integrate with external applications in a variety of ways through adapters
- Origins
  - Builds on Enterprise Integration Patterns for both
  - Builds on the Spring portfolio & programming model
  - See http://spring.io/spring-integration

© Copyright 2014 Pivotal. All rights reserved.

Tinker Toy™ Analogy

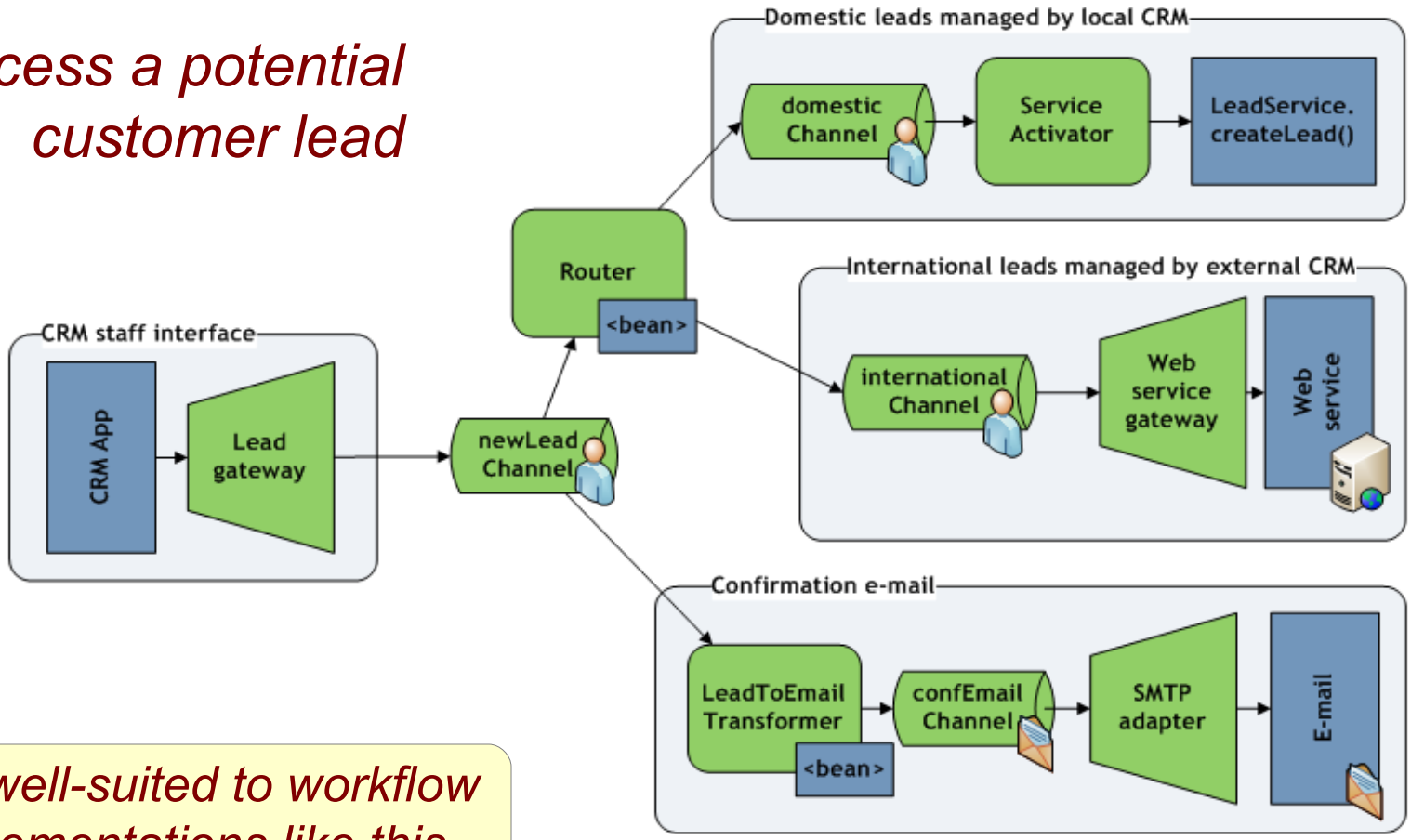
- Hub and spoke models
  - The spokes are communication *channels*
  - The hubs are your Java *components* or *endpoints*
- Use Spring Integration to build applications the same way
  - Architectural decision up-front



© Copyright 2014 Pivotal. All rights reserved.

Example System using S.I.

Process a potential customer lead



SI is well-suited to workflow implementations like this

© Copyright 2014 Pivotal. All rights reserved.

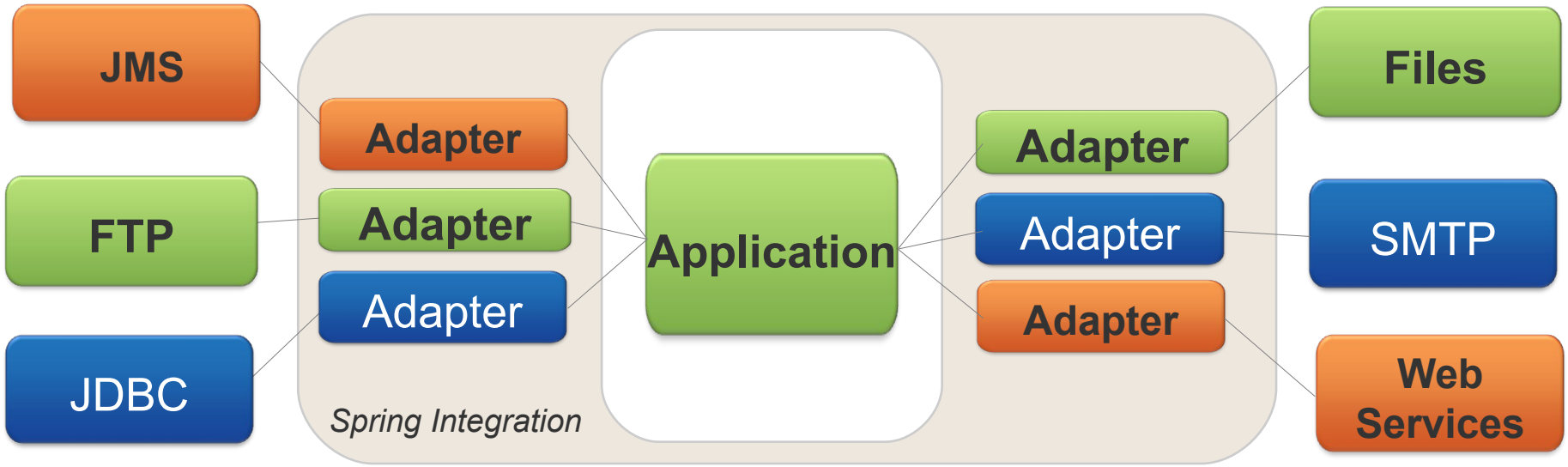
Spring Integration - Benefits

- Loose coupling between components
  - Small, focused components
  - Eases testing, reuse, etc.
- Event-driven architecture
  - No hard-coded process flow
  - Easy to change or expand
- Separates integration and processing logic
  - Framework handles routing, transformation, etc.
  - Easily switch between sync & async processing

© Copyright 2014 Pivotal. All rights reserved.

Spring Integration - Adapters

- Connect your application to the outside world
  - Remoting, REST, WS, File & FTP, SMTP, Twitter, …
  - For accepting input and producing output



© Copyright 2014 Pivotal. All rights reserved.

Spring Integration - Adapters

- Adapters shield application components from integration details
  - External events produce incoming message
    - Incoming email, new file, SOAP request, ...
  - Components just deal with (payload of) messages
    - Don't care about message origin nor destination
  - Internal message can trigger external event
    - Calling a service, sending JMS message or email, ...

© Copyright 2014 Pivotal. All rights reserved.

Topics in this session

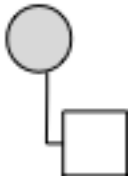
- Spring Integration – Goals
- Event driven architecture
- **Spring Integration**
  - **Basics**
  - External integration
  - Visual Editor (STS)
- Summary

© Copyright 2014 Pivotal. All rights reserved.

Ground rules

- Simple core API:
- A **Message** is sent by an **endpoint**
- **Endpoints** are connected to each other using **MessageChannels**
- An **endpoint** can receive **Messages** from a **MessageChannel**
  - By subscribing (passive) or polling (active)

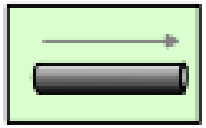
© Copyright 2014 Pivotal. All rights reserved.



Message

- A **Message** consists of **MessageHeaders** and a **payload**
  - Some headers are pre-defined
  - Payload is just a Java object
- A Message is immutable
  - Let framework wrap payload for you or use a **MessageBuilder** to create it
- Each Message is created with unique ID

© Copyright 2014 Pivotal. All rights reserved.



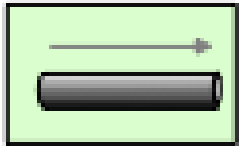
MessageChannels

- Central to loose coupling
  - Runtime IoC
- Connect message endpoints
- Optional buffering, interception
- Just Spring Beans
  - No broker needed
  - No persistence by default
    - May be backed by JMS or JDBC Message Store

© Copyright 2014 Pivotal. All rights reserved.

MessageChannel Types

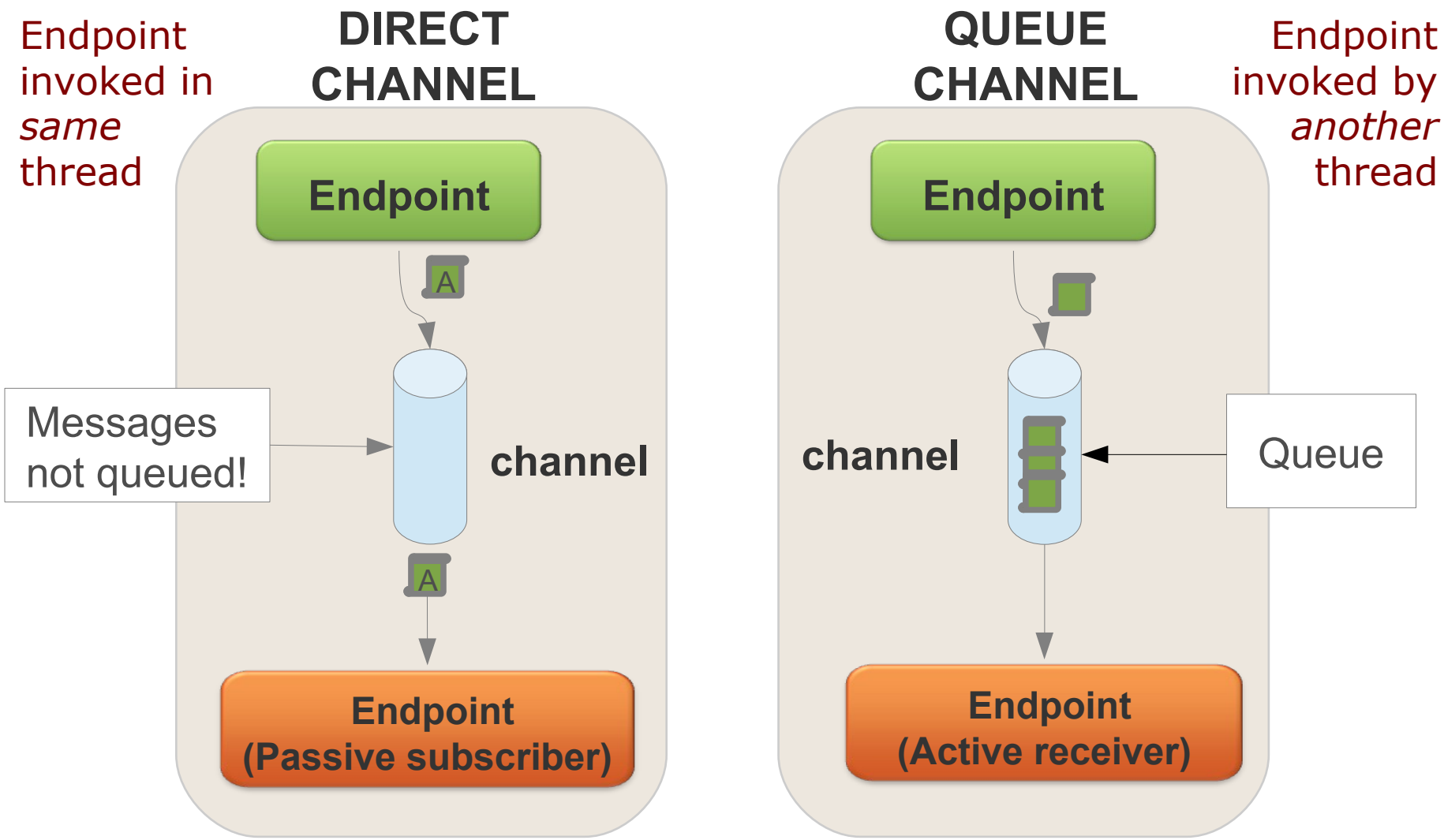
Point-to-point



- Only one receiver per message
- DirectChannel
  - Message passed to receiver in sender's thread
  - Sending blocks, not asynchronous!
- QueueChannel
  - Message is queued, sending doesn't block
  - Receiver polls from a different thread

© Copyright 2014 Pivotal. All rights reserved.

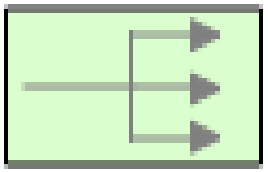
Direct vs Queue Channel



© Copyright 2014 Pivotal. All rights reserved.

MessageChannel Types

Publish-subscribe



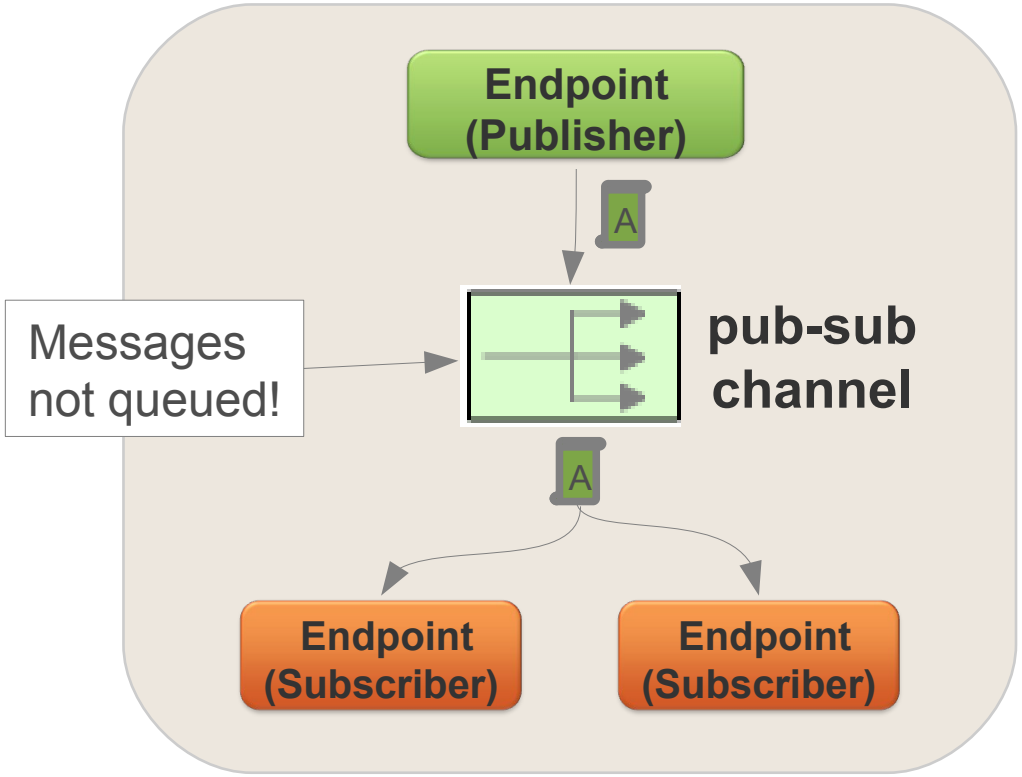
- Multiple receivers per message
- PublishSubscribeChannel
  - Receivers invoked consecutively in sender's thread
  - Or invoked in parallel on different threads using TaskExecutor

© Copyright 2014 Pivotal. All rights reserved.

MessageChannel Types

Publish-subscribe - 1

Endpoints invoked *serially* in *same* thread

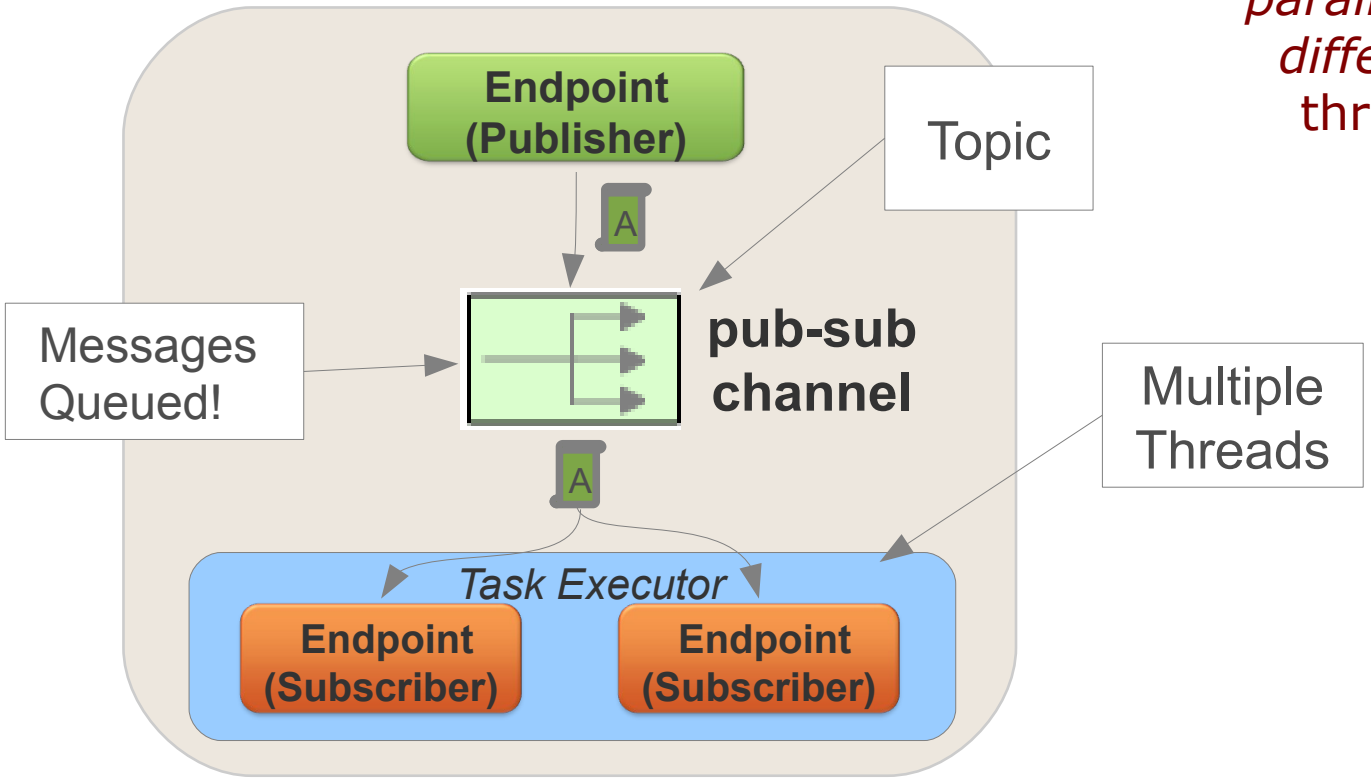


© Copyright 2014 Pivotal. All rights reserved.

MessageChannel Types

Publish-subscribe - 2

Endpoints invoked *in parallel* in *different* threads



© Copyright 2014 Pivotal. All rights reserved.

Defining Channels

DirectChannel (sync)

```
<channel id="incoming"/>
```

QueueChannel (async)

```
<channel id="orderedNotifications">
    <queue capacity="10"/>
</channel>
```

PublishSubscribeChannel (sync)

```
<publish-subscribe-channel id="statistics"/>
```

PublishSubscribeChannel (async)

```
<publish-subscribe-channel id="appEvents"
                           task-
    executor="pubSubExecutor"/>
<task:executor id="pubSubExecutor" pool-size="10"/>
```

Samples assume spring-integration namespace is the default

© Copyright 2014 Pivotal. All rights reserved.

Channel Interceptors

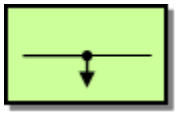
- Can operate on Messages
  - pre-/postSend, pre-/postReceive

```
<channel id="intercepted">
   <interceptors>
      <ref bean="someInterceptorImplementation"/>
   <interceptors>
</channel>
```

- Selective global interceptor

```
<channel-interceptor ref="interceptorForXChannels"
     pattern="x*" />
```


© Copyright 2014 Pivotal. All rights reserved.

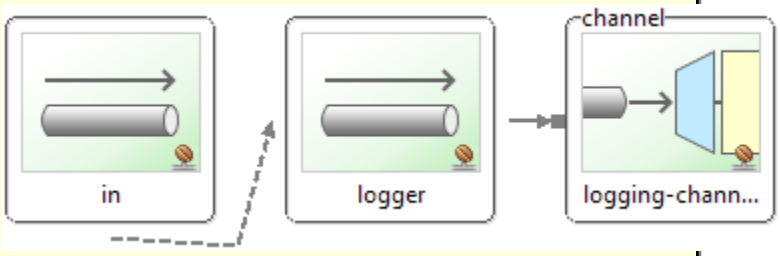


Wire Tap

- Standard pattern implemented as interceptor
- Copies incoming messages to given channel
  - 'Spy' on channel, good for debugging and monitoring
  - Often used with logging channel adapter

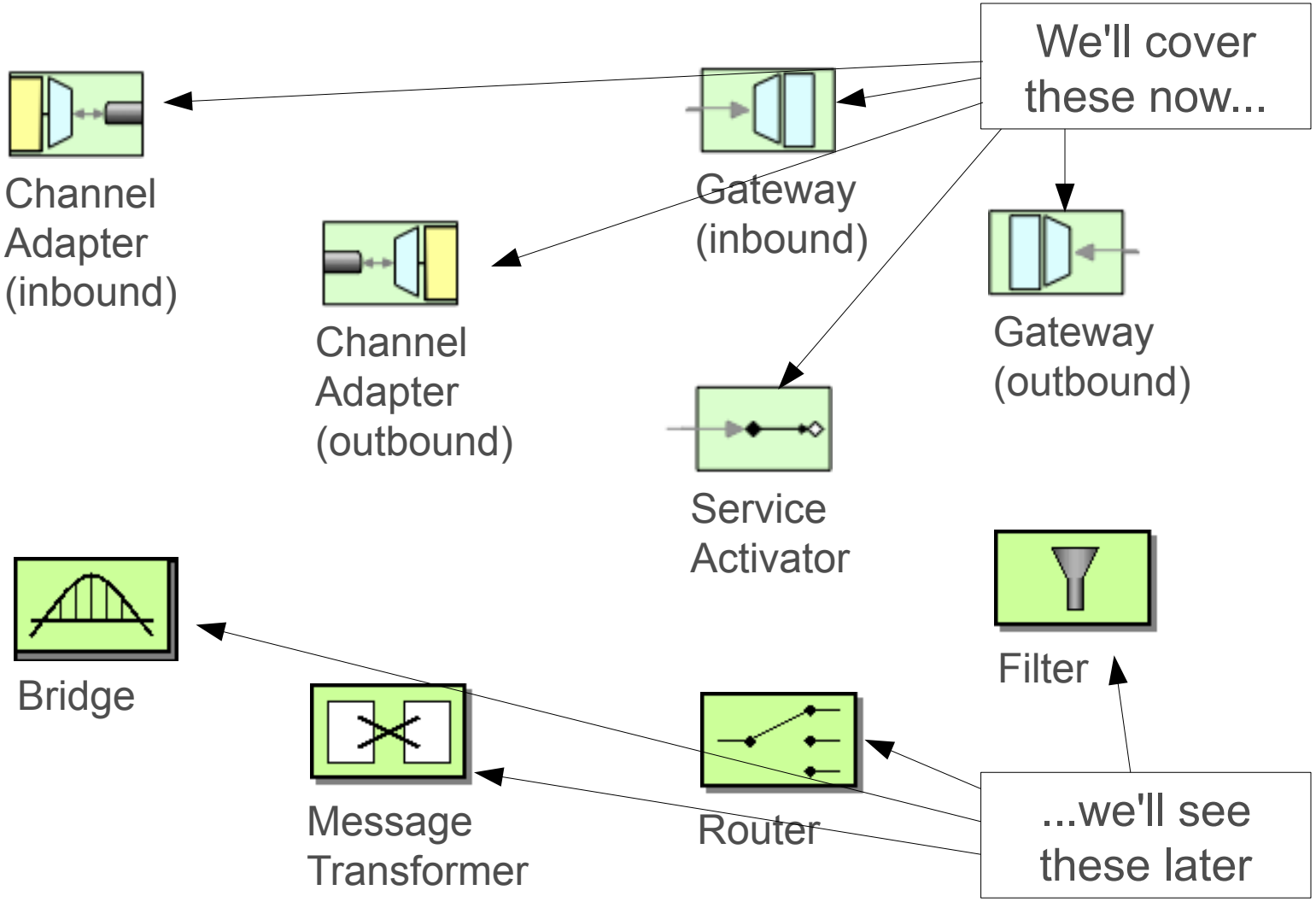
```
<channel id="in">
  <interceptors>
    <wire-tap channel="logger"/>
  </interceptors>
</channel>

<channel id="logger"/>

<logging-channel-adapter channel="logger" level="DEBUG"
                         log-full-message="true"/>
```

not just payload



© Copyright 2014 Pivotal. All rights reserved.

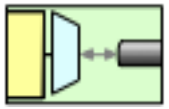
Message Endpoints



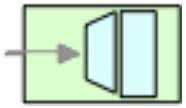
© Copyright 2014 Pivotal. All rights reserved.

Message Endpoints

- Channel Adapter: One way integration



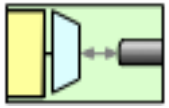
  - message enters or leaves application
  - Called 'inbound' or 'outbound'
- Gateway: Two way integration



  - Bring message into application and wait for response (inbound), or invoke external system and feed response back into application (outbound)
- Service Activator

  - Call method and wrap result in response message
  - Basically outbound gateway for invoking bean method

© Copyright 2014 Pivotal. All rights reserved.



Channel Adapters

- Used for generating new messages
  - Needs a poller to *activate* it on a schedule

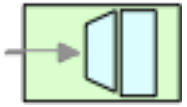
```
<int:inbound-channel-adapter id="generator"
    channel="events" ref="eventGenerator" method="generate" >
   <poller …/>
</int:inbound-channel-adapter>
```

- More typically used to accept external input
  - Such as an HTTP request or from a JMS queue

```
<int-http:inbound-channel-adapter id="httpAdapter"
        channel="updates"  supported-methods="PUT,DELETE" />
```

Outbound too

© Copyright 2014 Pivotal. All rights reserved.



Messaging Gateway

```
<gateway id="orderService"
    service-interface="com.example.OrderService"
    default-request-channel="orders" />
```

```
public interface OrderService {
    public OrderConfirmation submitOrder(Order order);
}
```

Java → Channel

- Proxy for sending new messages
  - Code doesn't depend on SI API
- Temporary reply channel created automatically
  - Can also specify default-response-channel

© Copyright 2014 Pivotal. All rights reserved.

Gateway Method Signatures

- Gateway interface methods may return Future
  - Becomes async, non-blocking gateway then
- 'void' methods also supported
  - Still called messaging *gateway*, even though that's technically a passive inbound *adapter* (as it's one-way)
- Use annotations for per-method configuration

```
public interface OrderService {
    @Gateway(requestChannel="orderChannel")
    public Future<Confirmation> submitOrder(Order order);

    @Gateway(requestChannel="cancellationChannel")
    public void cancelOrder(Order order);
}
```


© Copyright 2014 Pivotal. All rights reserved.

Service Activator

```
<service-activator ref="orderProcessor"
    input-channel="orders" output-channel="confirmations" />

<beans:bean id="orderProcessor" class="broker.OrderProcessor" />
```

```
public class OrderProcessor {
    public OrderConfirmation processOrder(Order order) {
        ...
    }
}
```

Channel → Java

- Invoke any bean method for incoming message
  - Specify method attribute or @ServiceActivator if there are multiple methods

© Copyright 2014 Pivotal. All rights reserved.

Service Activator Methods

- 'void' and null-returning methods also supported
  - No response message then (one-way)
  - <outbound-channel-adaptor> can be used as alternative for void methods
- Can cause problems when inbound gateway expects reply message
  - Set **requires-reply** to **true** to throw an exception on null

© Copyright 2014 Pivotal. All rights reserved.

Topics in this session

- Spring Integration – Goals
- **Spring Integration**
  - Basics
  - **External integration**
  - Visual Editor (STS)
- Summary

© Copyright 2014 Pivotal. All rights reserved.

Gateways and Adapters

- Remember the difference:
  - Channel Adapter is one way (in or out)
  - Inbound Gateway awaits internal reply and returns it in-band
  - Outbound Gateway awaits external response and puts it on a channel in invoking Thread
- Often use or add message headers
  - inbound HTTP: copies request header to SI headers
  - outbound JMS: copies SI headers to JMS headers

© Copyright 2014 Pivotal. All rights reserved.

Temporary Reply Channels (1)

- Temporary reply channels created automatically for inbound gateways if not explicitly defined
  - Anonymous point-to-point channel
  - Set as 'replyChannel' message header
- Used by components that produce output when no explicit output channel is provided
  - e.g. outbound gateways, service activators
  - Output becomes reply message
- Reply channel removed automatically after receiving reply message

© Copyright 2014 Pivotal. All rights reserved.

Temporary Reply Channels (2)

- Advice: only define explicit reply channel when you need a channel definition
  - i.e. to refer to the channel by name and/or to change its type from point-to-point to publish-subscribe
- Rely on default temporary reply channel otherwise

© Copyright 2014 Pivotal. All rights reserved.

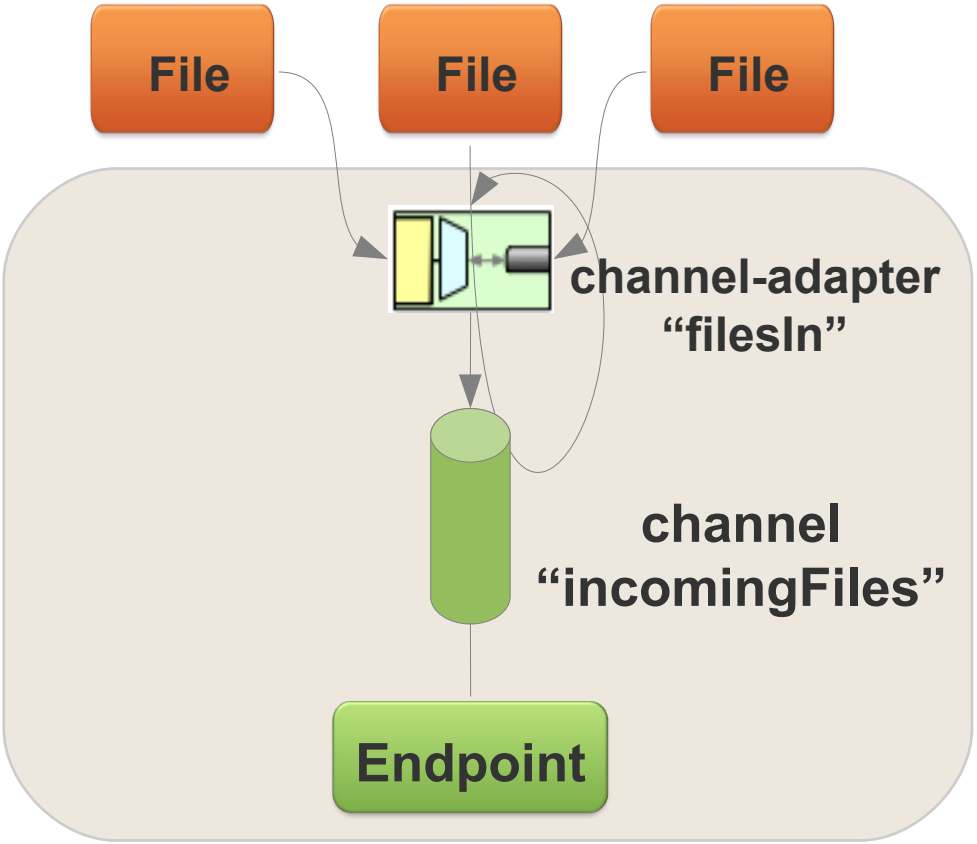
Integration Namespaces

- Spring Integration has dedicated namespaces for different integration types
  - http://..sfw../schema/integration/file
  - http://..sfw../schema/integration/http
  - http://..sfw../schema/integration/xml
  - http://..sfw../schema/integration/jms
  - http://..sfw../schema/integration/ip
  - http://..sfw../schema/integration/xmpp
  - http://..sfw../schema/integration/twitter
  - ...

© Copyright 2014 Pivotal. All rights reserved.

Sample: Inbound File Adapter

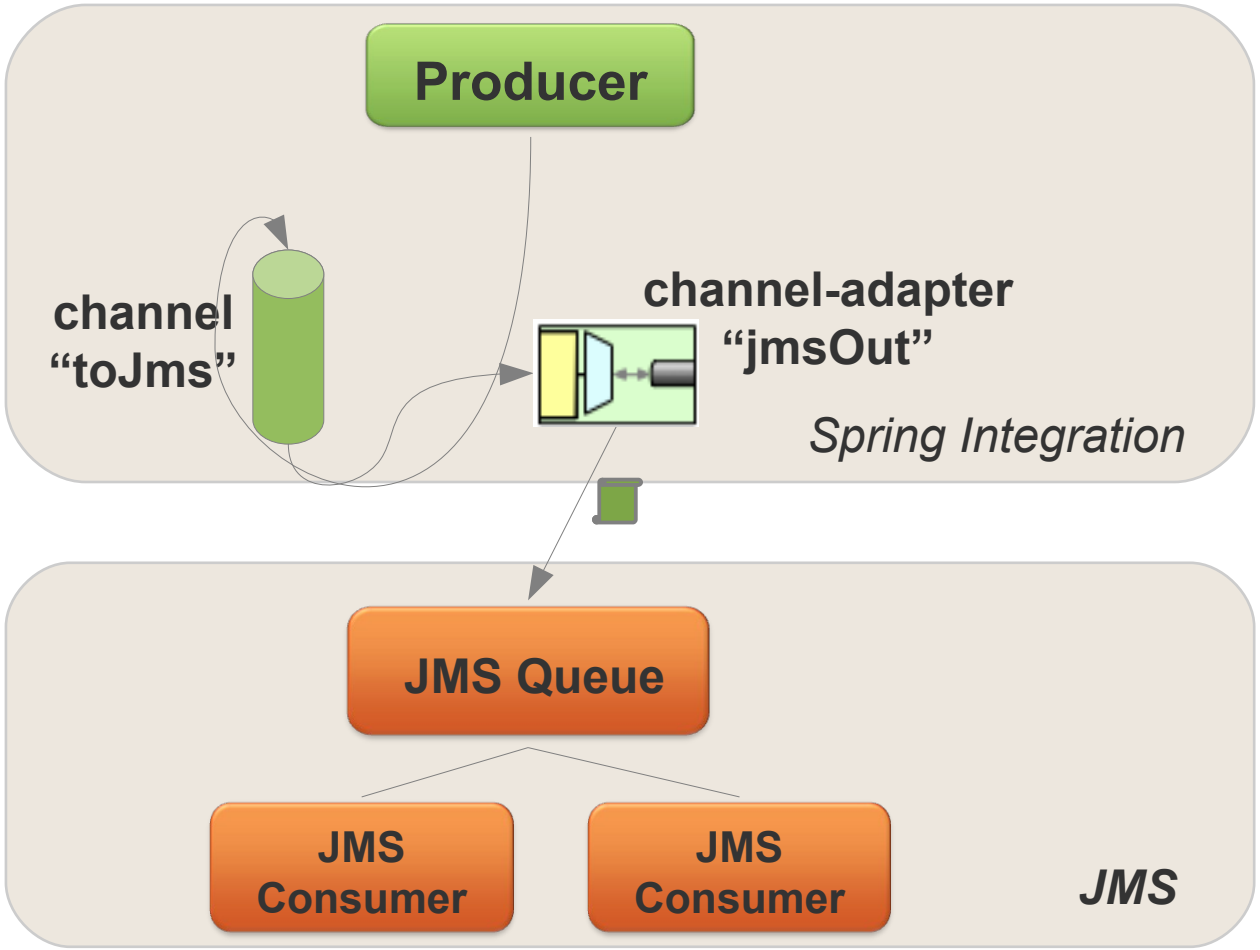
```
<int-file:inbound-channel-adapter id="filesIn"
    channel="incomingFiles"
    directory="file:C:/inputResource/" />
```



© Copyright 2014 Pivotal. All rights reserved.

Sample: Outbound JMS Adapter

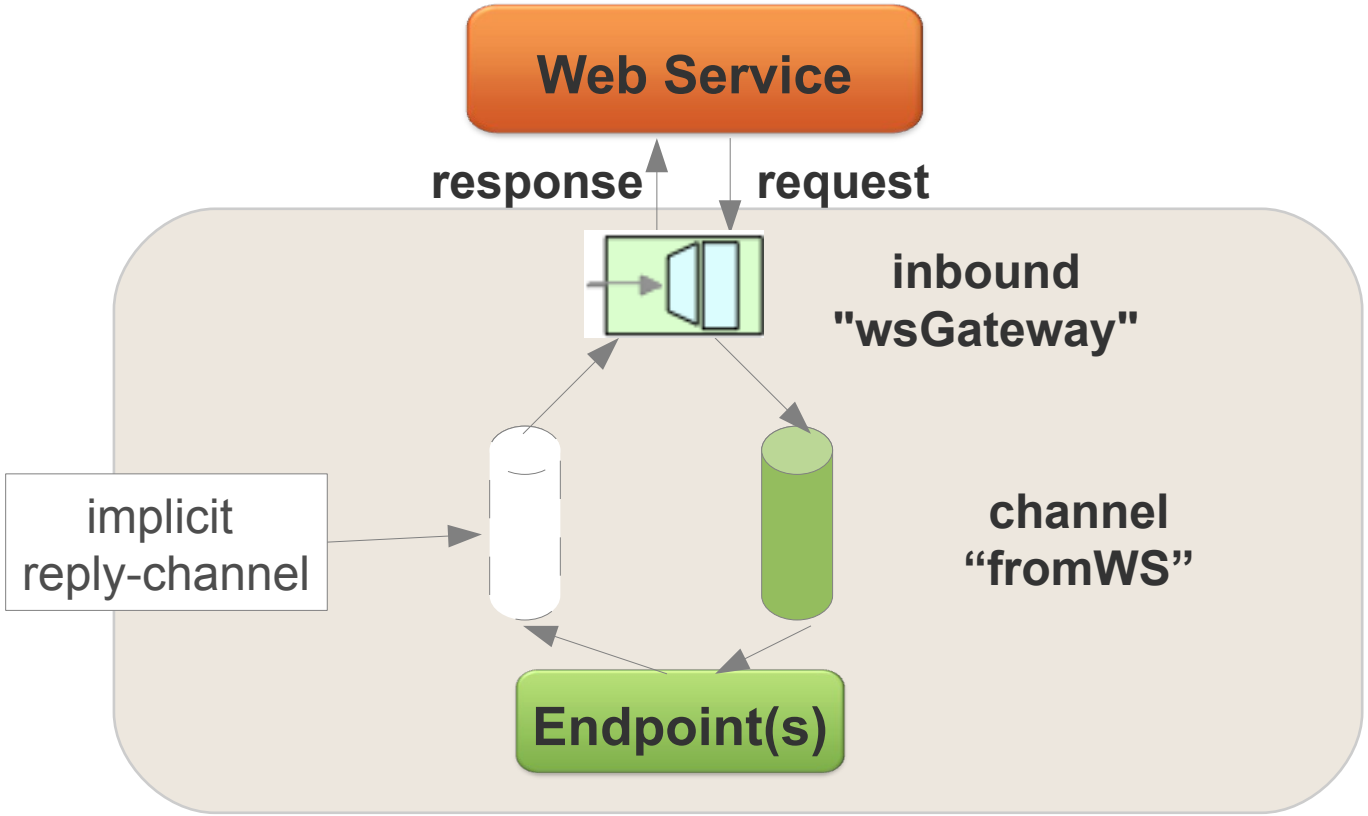
```
<int-jms:outbound-channel-adapter id="jmsOut"
    channel="toJms" destination="jmsQueue" />
```



© Copyright 2014 Pivotal. All rights reserved.

Sample: Inbound Web Service Gateway

```
<int-ws:inbound-gateway id="wsGateway" channel="fromWS"
                        marshaller="jaxb2" unmarshaller="jaxb2" />

<oxm:jaxb2-marshaller id="jaxb2" contextPath="com.example.xml"/>
```



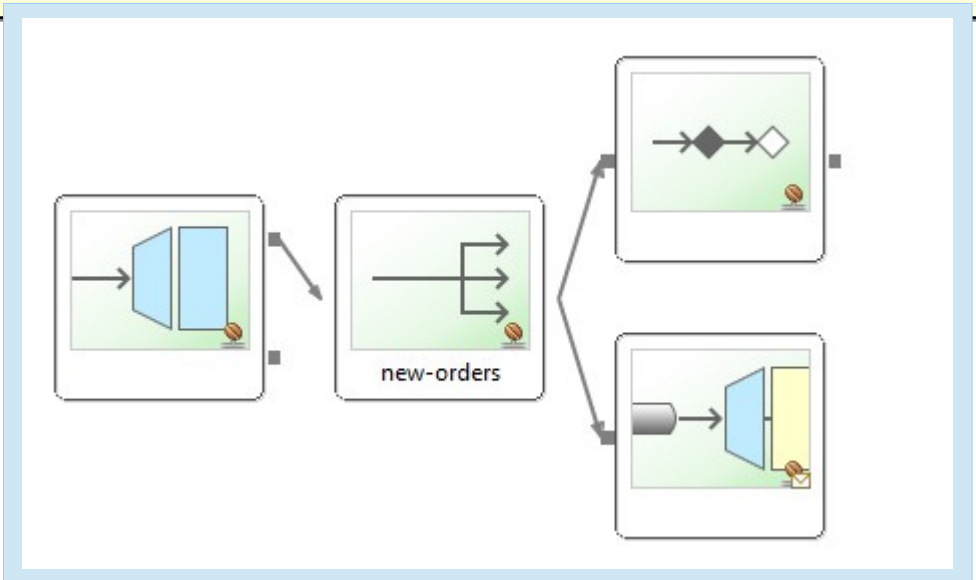
© Copyright 2014 Pivotal. All rights reserved.

Sample: Combining Components

```
<gateway default-request-channel="new-orders"
         service-interface="com.example.OrderService"/>
<publish-subscribe-channel id="new-orders"/>
<service-activator input-channel="new-orders" requires-reply="true"
                   ref="orderProcessor" method="processOrder" />
<int-jms:outbound-channel-adapter channel="new-orders"
                                  destination-name="queue.orders"/>
```

Send confirmation to implicit reply-channel

Send additional JMS message



© Copyright 2014 Pivotal. All rights reserved.

Sample: Combining Components

```
@Controller
public class OrderController {
  @Autowired OrderService orderService;

  @RequestMapping(value="/orders", method=POST)
  @ResponseStatus(CREATED)
  public void placeOrder(@RequestBody Order order,
          HttpServletRequest req, HttpServletResponse resp)  {
   Confirmation conf = orderService.submitOrder(order);
   …
  }
```

Use gateway to interact with SI

```
<gateway default-request-channel="new-orders"
          service-interface="com.example.OrderService"/>

   ...
```


© Copyright 2014 Pivotal. All rights reserved.

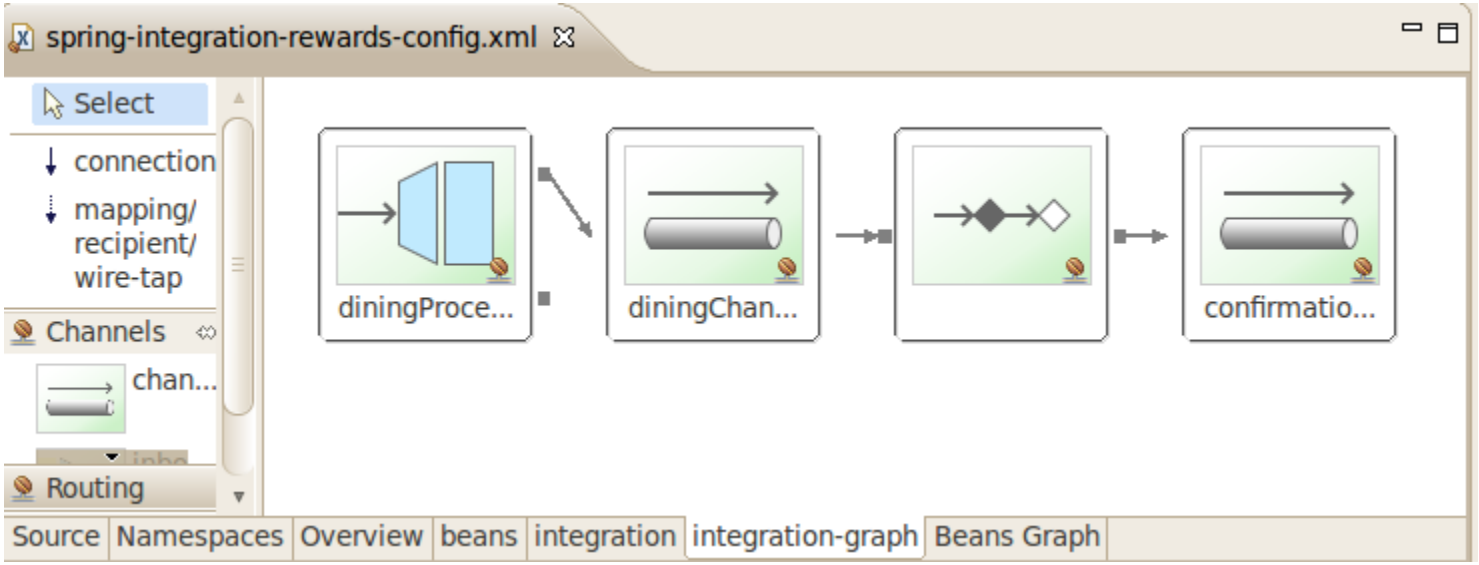
Topics in this session

- Spring Integration – Goals
- **Spring Integration**
  - Basics
  - External integration
  - **Visual Editor (STS)**
- Summary

© Copyright 2014 Pivotal. All rights reserved.

STS Visual Editor

- SpringSource Tool Suite includes a visual editor for Spring Integration flows
- Select 'integration-graph' tab on open Spring Integration configuration file



© Copyright 2014 Pivotal. All rights reserved.

Summary

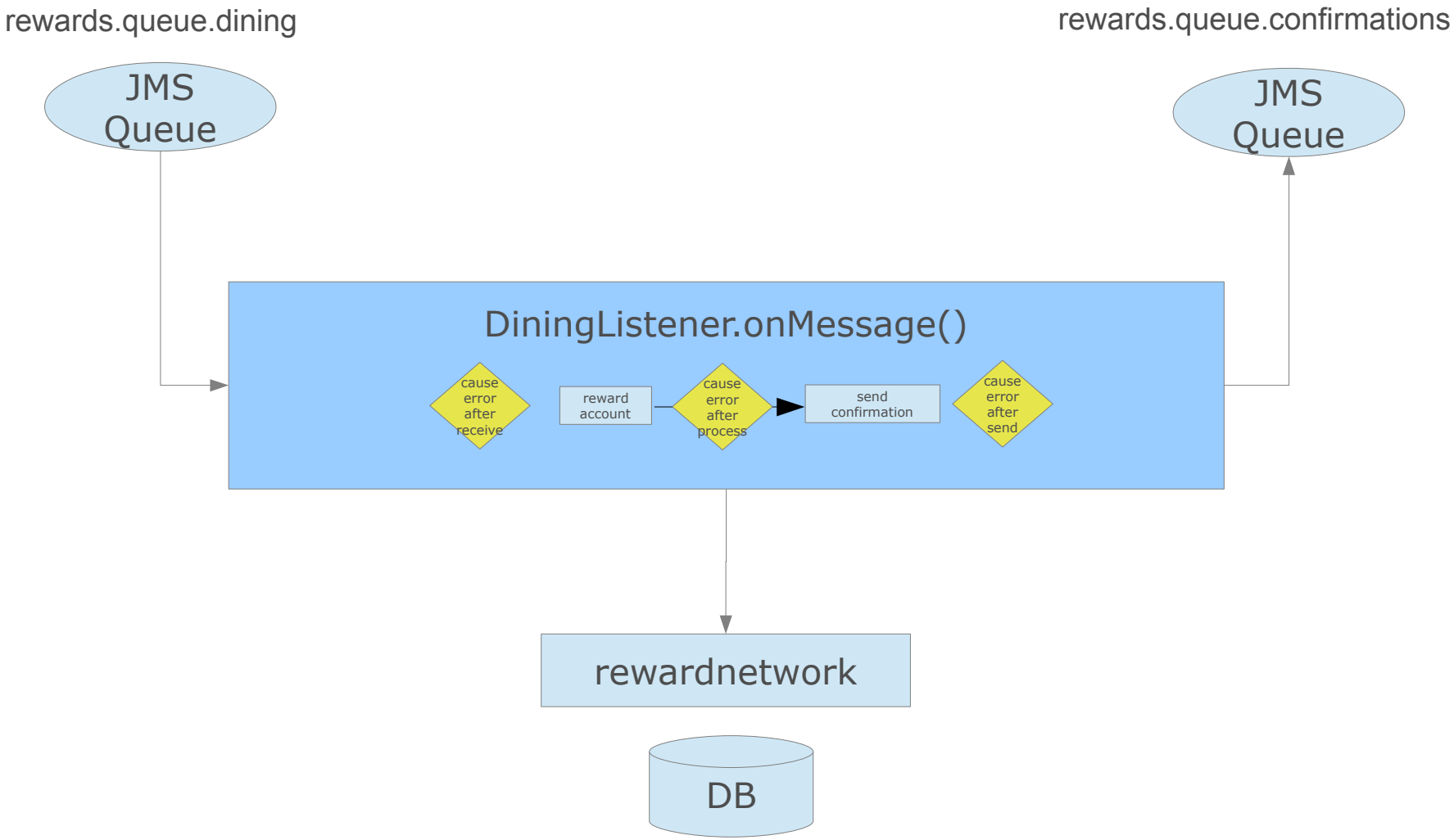
- Spring Integration provides the base components to implement EIP
  - To integrate application components
  - To integrate with external systems
- Allows for loosely coupled, event-driven architecture and separation of integration and processing logic

© Copyright 2014 Pivotal. All rights reserved.

Lab

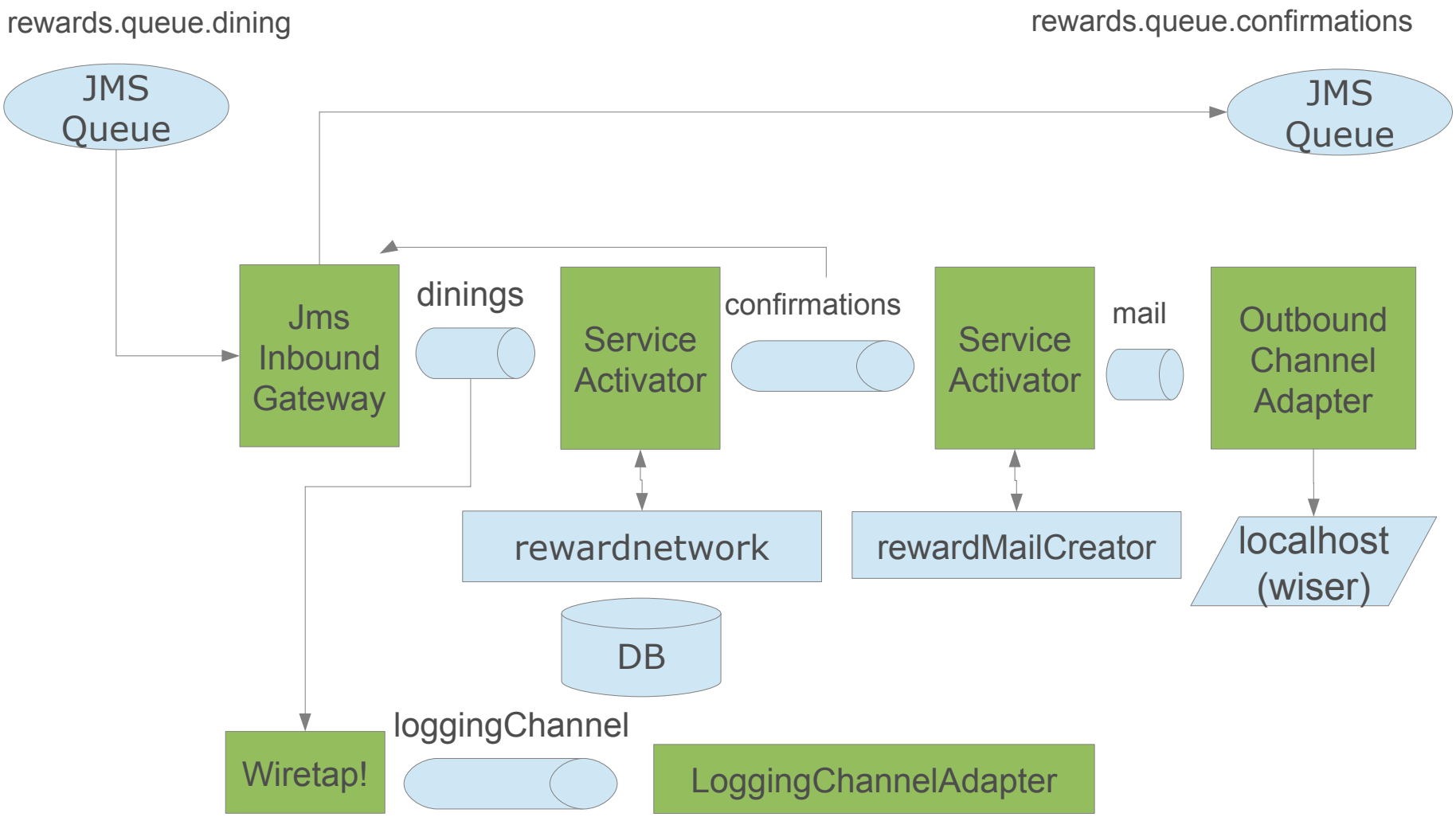
© Copyright 2014 Pivotal. All rights reserved.

Previous Exercise



© Copyright 2014 Pivotal. All rights reserved.

Converted to Spring Integration



© Copyright 2014 Pivotal. All rights reserved.

Spring Integration Configuration Overview

Configuring a working Pipes and Filters architecture

© Copyright 2014 Pivotal. All rights reserved.

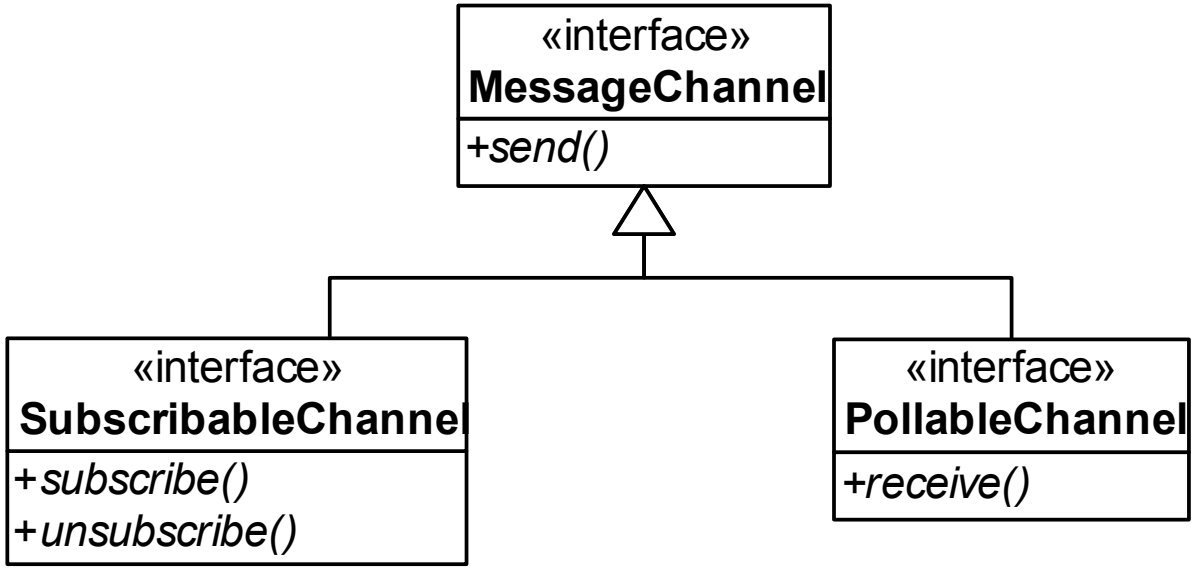
Topics

- **Channel Types and Polling**
- Synchronous vs. asynchronous handoff
- Error handling
- More Endpoint types
- Simplifying configuration

© Copyright 2014 Pivotal. All rights reserved.

Types of MessageChannels

- *SubscribableChannels* invoke subscribers on *send()*
- *PollableChannels* wait for *receive()* to be called
  - typically from another thread



© Copyright 2014 Pivotal. All rights reserved.

MessageChannel Implementations

- SubscribableChannel
  - DirectChannel — direct point-to-point
  - PublishSubscribeChannel — pub-sub, optional TaskExecutor
  - ExecutorChannel — point-to-point via TaskExecutor
- PollableChannel (always point-to-point)
  - Queue Channel — FIFO buffering queue
    - Priority Channel — priority-ordered
    - Rendezvous Channel — zero-capacity, ensures handoff
  - NullChannel — /dev/null

See Spring Integration Reference – Message Channel Implementations for other kinds of channels

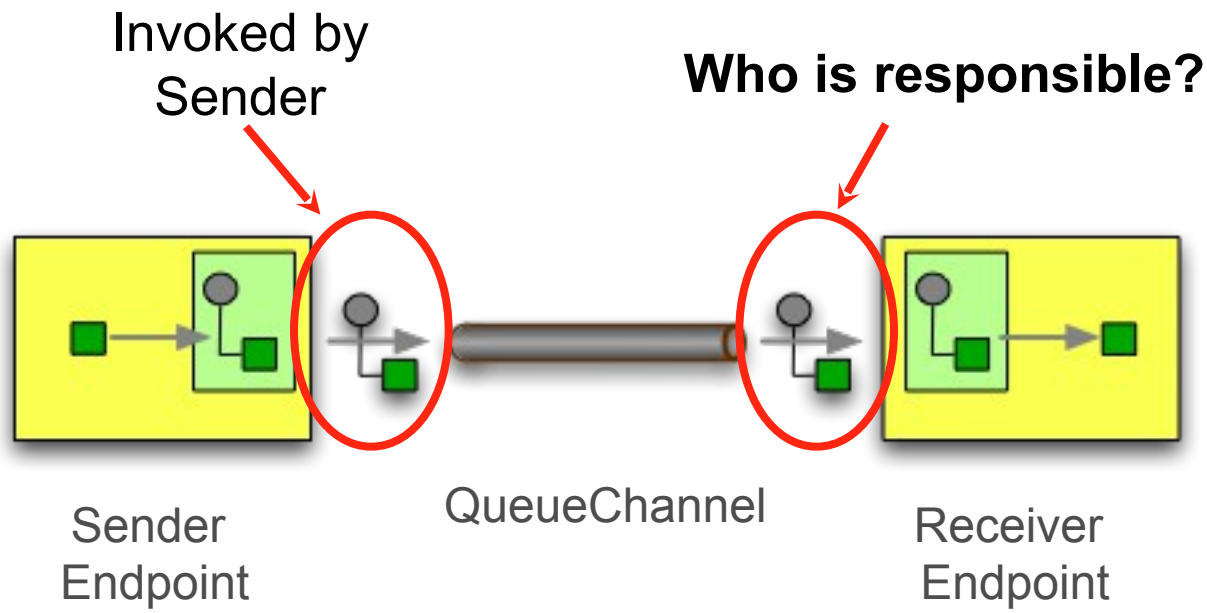
© Copyright 2014 Pivotal. All rights reserved.

Passive Components

- Most application components are passive
  - Controllers, services, repositories etc.
  - Just wait to be invoked
  - Container takes care of threading
- Channels are also passive!
  - SubscribableChannels call passive subscriber(s) directly when send(Message) is called
  - But when sending to a PollableChannel, something must actively receive the message

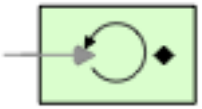
© Copyright 2014 Pivotal. All rights reserved.

Missing Link: Who Picks Up New Messages?



© Copyright 2014 Pivotal. All rights reserved.

Active Components



- Endpoints can be made active using a poller
- By default single thread does the polling
  - Optionally use a thread pool

```
<poller default="true" task-executor="pool"
        fixed-delay="500"/>

<task:executor id="pool" pool-size="9" />
```

milliseconds by default

© Copyright 2014 Pivotal. All rights reserved.

Endpoints Wired with Poller

- Endpoints use default poller when needed
  - i.e. for PollableChannels as input
- Overridable per endpoint

```
<service-activator … >
   <poller task-executor="smallPool" fixed-delay="500"/>
</service-activator>

<task:executor id="smallPool" pool-size="3" />
```


© Copyright 2014 Pivotal. All rights reserved.

Transactional Pollers

- Transactional pollers make message handling flow atomic
  - Assuming flow is single threaded
  - TX spans receive() call and handler invocation
- Configurable on the poller
  - Same configuration options and defaults as Spring

```
<service-activator … >
  <poller fixed-rate="1000">
    <transactional/>
  </poller>
</service-activator>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics

- Channel Types and Polling
- **Synchronous vs. asynchronous handoff**
- Error handling
- More Endpoint types
- Simplifying configuration

© Copyright 2014 Pivotal. All rights reserved.

Synchronous Handoff

- DirectChannels and PublishSubscribeChannels without Executors perform synchronous handoff
- Sending thread invokes receiver(s) directly
- Just like a regular method call:
  - Transactions work
  - Security context available
  - Exceptions are simply propagated back to caller
  - Low overhead
  - But may not scale as sending thread is held up

```
<channel id="direct"/>
```

```
<publish-subscribe-channel id="pubsub"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Asynchronous Handoff

- With other channel types receiver runs on different thread than sender
- When breaking thread boundary:
  - Transaction and security context are lost
  - Exceptions can typically not be propagated to caller
- Make no assumptions about temporal (happens-before) relations
  - Avoid relying on them
  - Can still wait for reply message of course

© Copyright 2014 Pivotal. All rights reserved.

From Sync to Async

- To switch from sync to async simply add a queue or task-executor
- Small change has profound effects
  - ensure you understand them well

```
<channel id="queueChannel">
   <queue capacity="10"/>
</channel>
```

```
<publish-subscribe-channel id="pubsub"
          task-executor="taskExecutor"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics

- Channel Types and Polling
- Synchronous vs. asynchronous handoff
- **Error handling**
- More Endpoint types
- Simplifying configuration

© Copyright 2014 Pivotal. All rights reserved.

Synchronous Error Handling

- Sender receives a MessageHandlingException when receiving endpoint throws exception in sender's thread
  - Wraps original exception and failed Message

```
<gateway default-request-channel="in" service-interface="foo.SimpleGateway" />
<service-activator input-channel="in" expression="1/0" />
```

```
SimpleGateway gateway = context.getBean(SimpleGateway.class);
gateway.send("test");
```

```
Exception in thread "main" org.springframework.integration.MessageHandlingException:
Expression evaluation failed: 1/0
        ...
Caused by: java.lang.ArithmeticException: / by zero
```


© Copyright 2014 Pivotal. All rights reserved.

Asynchronous Error Handling

- What if async receiver throws Exception?
  - Sender has already moved on
- MessageHandlingException is sent to error channel
  - Use errorChannel header from request message, set by some endpoint
  - If absent, use global channel called "errorChannel"

```
<gateway default-request-channel="gatewayChannel"
         service-interface="foo.SimpleGateway"
         error-channel="exceptionTransformationChannel"/>
```


© Copyright 2014 Pivotal. All rights reserved.

errorChannel

- Created as publish-subscribe-channel by default
  - Define your own "errorChannel" to override, e.g. for asynchronous handling
- Register handler for global error handling
  - Could be simple logger or complete custom flow
  - Built-in router based on exception type:

```
<exception-type-router input-channel="errorChannel"
                       default-output-channel="genericErrors">
  <mapping exception-type="example.OutOfStockException"
           channel="resupplyChannel"/>
  <mapping exception-type="example.InsufficientFundsException"
           channel="loanProposalChannel"/>
</exception-type-router>
```

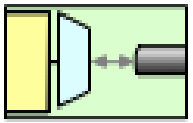

© Copyright 2014 Pivotal. All rights reserved.

Topics

- Channel Types and Polling
- Synchronous vs. asynchronous handoff
- Error handling
- **More Endpoint types**
- Simplifying configuration

© Copyright 2014 Pivotal. All rights reserved.

Inbound Channel Adapter

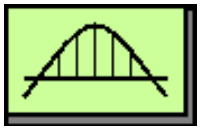


- Method invoking adapter, or transport related implementation
- Often requires polling
  - Unless transport calls subscribed adapter

```
<inbound-channel-adapter channel="inputChannel"
        ref="newWorkSource"
        method="getJob"/>
   <poller cron="0 0-5 14 * * *"/>
</inbound-channel-adapter>
```


© Copyright 2014 Pivotal. All rights reserved.

Bridge

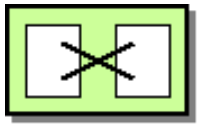


- Simply connects two channels or channel adapters
  - e.g. to connect pollable to subscribable channel
- Can have a <poller /> - used to throttle msgs

```
<bridge input-channel="input" output-channel="output" />

<bridge input-channel="input" output-channel="output" >
   <poller .../>
</bridge>

<stream:stdin-channel-adapter id="stdin" />
<stream:stdout-channel-adapter id="stdout" />
<bridge id="echo" input-channel="stdin" output-channel="stdout" />
```


© Copyright 2014 Pivotal. All rights reserved.

Message Transformer



- Basically service activator with specific intent

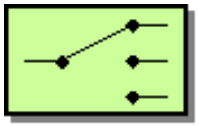
```
<transformer input-channel="inputChannel"
             output-channel="outputChannel"
             ref="anyPOJO" method="doTransform"/>
```

- Common use cases:
  - Payload format *conversion*, incl. (un)marshalling
  - Payload or header *enriching*

Message is immutable, so transformer always creates new Message!

© Copyright 2014 Pivotal. All rights reserved.

Router



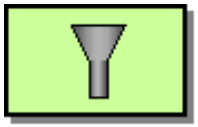
- Decides next channel(s) to send message to
- Typically based on payload or headers
  - Several default implementations available

```
public class OrderRouter {
  public String routeOrder(Order order) {
    return stockChecker.inStock(order) ? “warehouse” : “resupply”;
  }
}
```

```
<router input-channel="orders" ref="orderRouter" method="routeOrder"/>
<beans:bean id="orderRouter" class="example.OrderRouter" />
```


© Copyright 2014 Pivotal. All rights reserved.

Filter



- Decides whether to pass or drop a message
- Usually implemented as boolean method
  - True means pass, false means drop

```
public boolean filter(Order order) {
    return order.isValid();
}
```

```
<filter ref="orderFilter" method="filter"
        input-channel="orders" output-channel="validOrders"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Filter Options

- Default is to silently drop message
- Alternatives:
  - Throw exception on rejection
  - Send message to discard channel
    - 2-way router, or *switch*

```
<filter ref="orderFilter" method="filter" input-channel="orders"
output-channel="validOrders" throw-exception-on-rejection="true"/>

<filter ref="orderFilter" method="filter" input-channel="orders"
output-channel="validOrders" discard-channel="invalidOrders"/>
```


© Copyright 2014 Pivotal. All rights reserved.

MessagingTemplate

- Using Spring Integration API often not necessary
  - Code shielded from framework through gateways, service activators, etc.
- Sometimes direct Message and/or Message-Channel usage is more convenient
  - Test code, custom Message creation, etc.
- Framework offers MessagingTemplate for this
  - Sending & receiving
  - Similar to JmsTemplate
    - incl. MessageConverter and channel name resolving
    - Uses payload as-is and resolves bean names by default

© Copyright 2014 Pivotal. All rights reserved.

MessagingTemplate Usage

```
<bean class="org.springframework.integration.core.MessagingTemplate">
  <property name="receiveTimeout" value="1000"/>
</bean>
```

```
@Autowired MessagingTemplate template;
@Autowired MessageChannel someChannel;

Message<String> msg = MessageBuilder.withPayload("Hello")
  .setHeader("customHeader", someObject)
  .setErrorChannelName("someErrorChannel").build();
template.send(someChannel, msg);

RewardConfirmation confirmation =
  (RewardConfirmation) template.receiveAndConvert("confirmations");

// synchronous request-response:
Message responseMsg = template.sendAndReceive("requests", requestMsg);
```

Dependency injection by name – bean with id="someChannel"

message conversion

named channels

© Copyright 2014 Pivotal. All rights reserved.

Topics

- Channel Types and Polling
- Synchronous vs. asynchronous handoff
- Error handling
- More Endpoint types
- **Simplifying configuration**

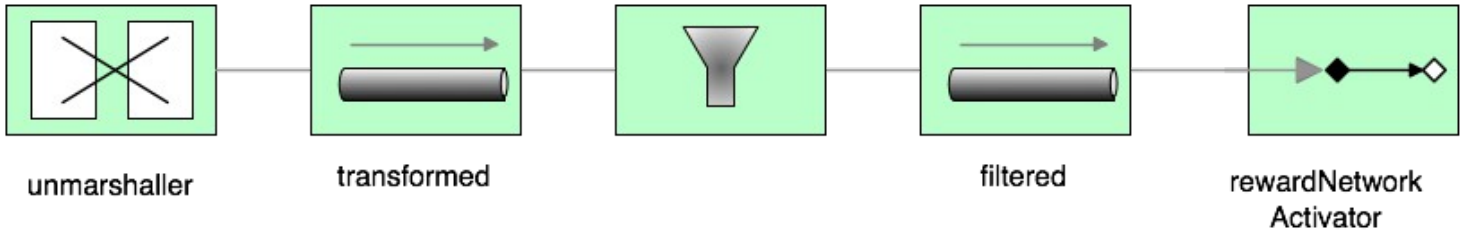
© Copyright 2014 Pivotal. All rights reserved.

Chaining Endpoints

- Many endpoints work together in a chain
  - Unmarshal
  - Filter out redundant messages
  - Invoke service
- Requires multiple channels to tie them together
- Reduce boilerplate with chains

© Copyright 2014 Pivotal. All rights reserved.

Consider This Setup



© Copyright 2014 Pivotal. All rights reserved.

Without Chain

```
<transformer input-channel="input"
    output-channel="transformed"
    ref="unmarshallingTransformer"/>

<channel id="transformed"/>

<filter input-channel="transformed"
    output-channel="filtered" ref="filterBean"/>

<channel id="filtered"/>

<service-activator input-channel="filtered"
    output-channel="output"
    ref="rewardNetwork" method="rewardAccountFor" />
```


© Copyright 2014 Pivotal. All rights reserved.

With Chain

```
<chain input-channel="input" output-channel="output" >
   <transformer ref="unmarshallingTransformer"/>
   <filter ref="filterBean"/>
   <service-activator ref="rewardNetwork"
                      method="rewardAccountFor" />
</chain>
```

© Copyright 2014 Pivotal. All rights reserved.

Chaining Contract

- All endpoints wired with implicit DirectChannels in order
- All endpoints but the last MUST return output
  - Endpoint can filter a message by returning null
- If last endpoint returns something, either output-channel or replyChannel must be set

© Copyright 2014 Pivotal. All rights reserved.

Endpoint Configuration

- So far most configuration was XML
- Annotations are also supported
  - @MessageEndpoint stereotype
    - Class can be picked up by component-scanning
  - Method annotations for various endpoint types
    - @Gateway, @ServiceActivator, @Router, @Filter, etc.
- As well as Spring Expression Language and Groovy
  - Can even monitor file for dynamic changes
  - Groovy not covered in this course

© Copyright 2014 Pivotal. All rights reserved.

Method Annotations

- Disambiguate between multiple endpoint methods
  - when xml only has ref attribute
- Specify internal configuration, like channels
- Pass certain or all headers to endpoint methods

```
@MessageEndpoint
public class QuoteService {
  @ServiceActivator(inputChannel="tickers", outputChannel="quotes")
  public Quote lookupQuote(String ticker) { ... }
}
```

```
@Router
public List<String> route(@Header("status") OrderStatus status) { … }
```


© Copyright 2014 Pivotal. All rights reserved.

SpEL Expressions

- Many endpoints support expression attribute
- Inline handler logic to avoid writing trivial Java code
- SI variables available: headers, payload

```
<router input-channel="in" expression="payload + 'Channel'"/>

<filter input-channel="in"
        expression="payload.equals('nonsense')"/>

<service-activator input-channel="in" output-channel="out"
  expression="@accountService.processAccount(payload,
                                             headers.accountId)"/>
```

named Spring bean

© Copyright 2014 Pivotal. All rights reserved.

Summary

- Spring Integration takes care of messaging/threading plumbing code
  - Transparently configures polling
  - Allows passive programming model for active endpoints
- Simple but powerful scheduling
- Declarative configuration for endpoints

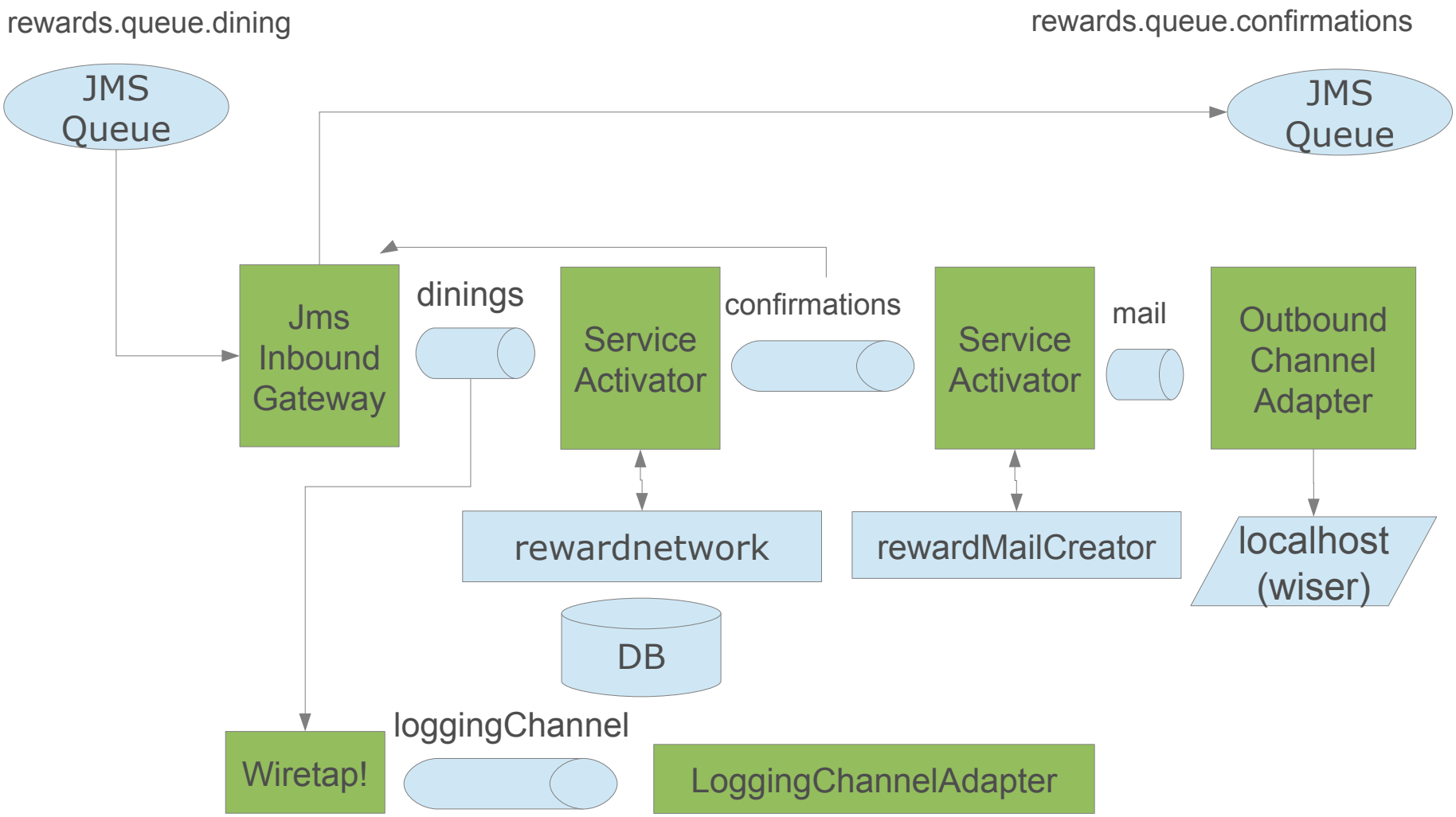
© Copyright 2014 Pivotal. All rights reserved.

Lab

Optimize the Messaging application using an Idempotent Receiver

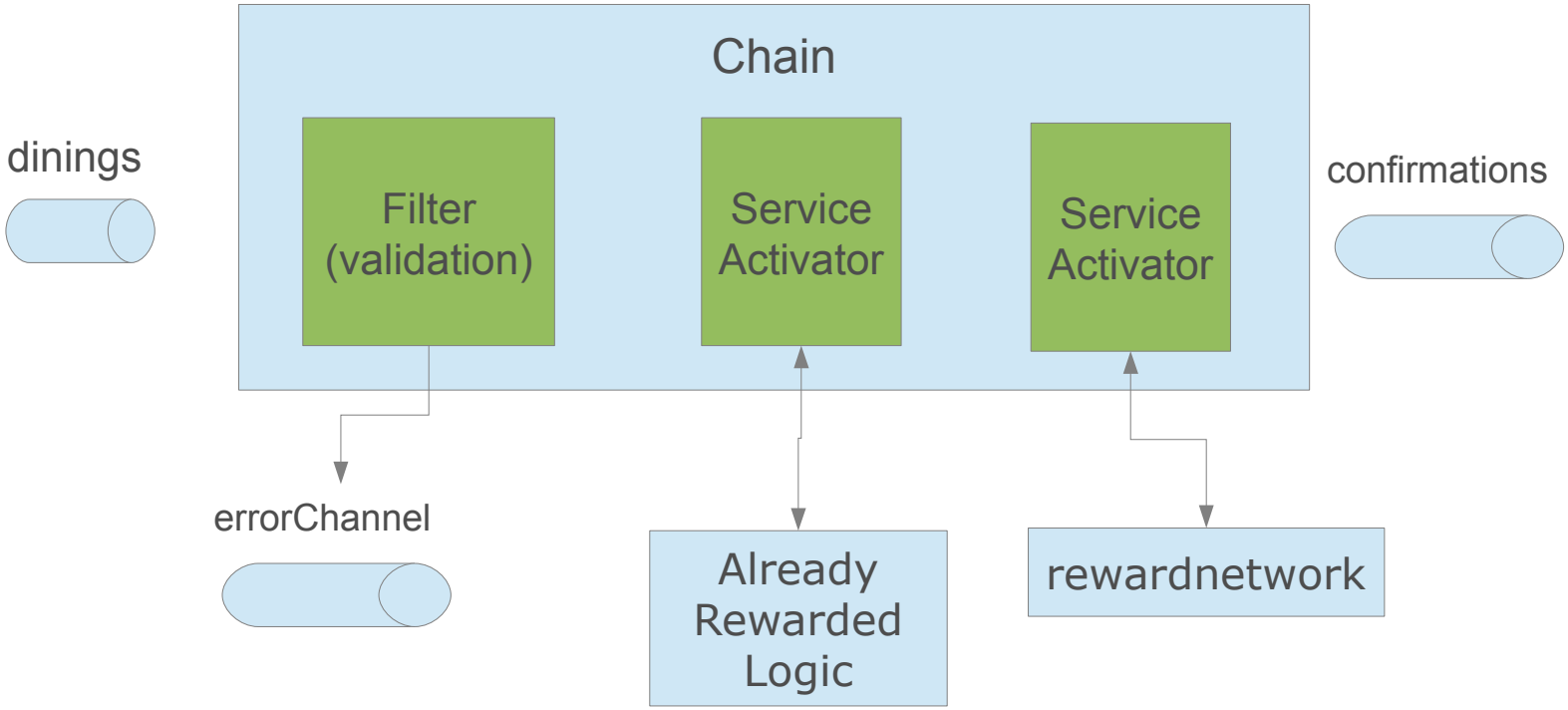
© Copyright 2014 Pivotal. All rights reserved.

Converted to Spring Integration



© Copyright 2014 Pivotal. All rights reserved.

Previous Exercise



© Copyright 2014 Pivotal. All rights reserved.

Spring Integration Advanced Features

Splitting & aggregating, dispatching and XML support

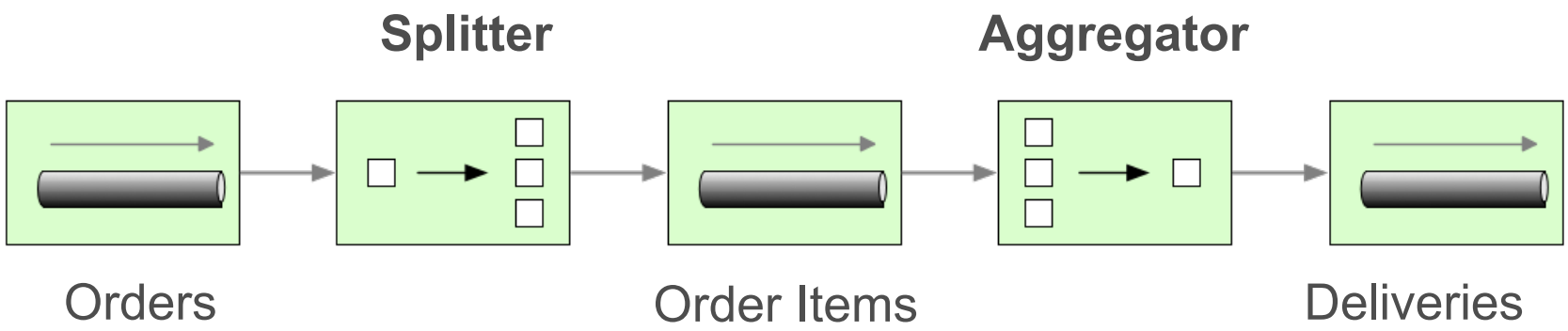
© Copyright 2014 Pivotal. All rights reserved.

Topics

- **Splitting and aggregating**
- Dispatcher configuration
- XML support

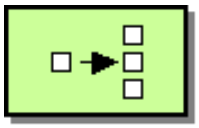
© Copyright 2014 Pivotal. All rights reserved.

Splitting and Aggregating



© Copyright 2014 Pivotal. All rights reserved.

Splitter



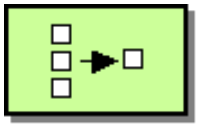
- Input: a single message
- Output: multiple messages
- Splitting strategy should be provided in a dedicated method

```
@Splitter
public List<OrderItem> split(Order order) {
    return order.getItems();
}
```

```
<splitter ref="orderSplitter" ... />
```


© Copyright 2014 Pivotal. All rights reserved.

Aggregator



- Input: multiple messages
- Output: a single message
- Aggregating strategy should be provided in a dedicated method

```
@Aggregator
public Order combine(List<OrderItem> items) {
    return Order.withItems(items);
}
```

```
<aggregator ref="itemAggregator" ... />
```

© Copyright 2014 Pivotal. All rights reserved.

Correlation and Release

- Aggregator holds messages until a set of **correlated** messages is ready to **release**

- Correlation strategy
  - What correlates this message?
- Release strategy
  - Is this list of messages complete and ready to be released?

© Copyright 2014 Pivotal. All rights reserved.

Correlation Strategy

- Default: CORRELATION_ID header
- Inject implementation of CorrelationStrategy into aggregator
  - also correlation-method
  - also correlation-expression

```
<aggregator  ref="itemAggregator"
    correlation-strategy="correlationStrategyImpl" ... />

<aggregator  ref="personIdAggregator"
    correlation-strategy-expression="payload.person.id" ... />
```


© Copyright 2014 Pivotal. All rights reserved.

Release Strategy

- Default: SequenceSizeReleaseStrategy
  - SEQUENCE_SIZE header
- Inject implementation of ReleaseStrategy into aggregator
  - also release-strategy-method
  - also release-strategy-expression

```
<aggregator ref="itemAggregator"
    release-strategy="releaseStrategyImpl" ... />

<aggregator ref="msgGroupSizeSpELAggregator"
    release-strategy-expression="payload.size() gt 5" ... />
```


© Copyright 2014 Pivotal. All rights reserved.

Splitter and Aggregator

- Splitter sets CORRELATION_ID to MESSAGE_ID of original message
- Also sets SEQUENCE_* headers
- If you don't change the headers Aggregator **just works** with the products of a Splitter
- Since 2.0, nested splitters work too

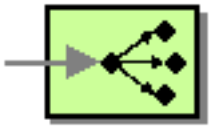
© Copyright 2014 Pivotal. All rights reserved.

Topics

- Splitting and aggregating
- **Dispatcher configuration**
- XML support

© Copyright 2014 Pivotal. All rights reserved.

Point-to-point Dispatch



- Point-to-point SubscribableChannel ensures single handler per message
- But can still have multiple subscribers

- How is handler determined?
- What happens if handler fails?

© Copyright 2014 Pivotal. All rights reserved.

Dispatching Defaults

- Default is round-robin load balancer with failover
  - Rotate over subscribers for subsequent messages
  - Call next subscriber if current one throws Exception
- Handlers can have 'order' property set for failover
  - If not, order of subscribing is used

```
<inbound-channel-adapter channel="in" expression=" 'foo' ">
   <poller fixed-rate="100"/>
</inbound-channel-adapter>

<channel id="in"/>

<logging-channel-adapter channel="in" expression=" 'first' "/>
<logging-channel-adapter channel="in" expression=" 'second' "/>
```


© Copyright 2014 Pivotal. All rights reserved.

Dispatcher Configuration

- Can disable load balancing and/or failover

```
<channel id="in">
  <dispatcher failover="false"/>
</channel>
```

```
<channel id="in">
  <dispatcher load-balancer="none"/>
</channel>
```

- Without failover exception propagates back to caller immediately
- Without load balancer, subscribers are always called in same order
  - Good for 'primary' handlers with fallback
  - Only makes sense with failover enabled

© Copyright 2014 Pivotal. All rights reserved.

Asynchronous Dispatching

- Can also dispatch using TaskExecutor
  - Defines ExecutorChannel instead of DirectChannel

```
<channel id="in">
  <dispatcher task-executor="someExecutor"/>
</channel>
```

- Sending doesn't block (assuming thread available)
- Subscribers still called from a single thread
  - Which is no longer the caller's thread!
  - failover and/or load-balancer still supported

© Copyright 2014 Pivotal. All rights reserved.

Topics

- Splitting and aggregating
- Dispatcher configuration
- **XML support**

© Copyright 2014 Pivotal. All rights reserved.

XML Support Overview

- XPath:
  - Splitting
  - Routing
  - Filtering
- XSLT
- OXM
- XML Payloads

© Copyright 2014 Pivotal. All rights reserved.

XPath Support

```
<!-- splits Node into Nodes, String and File into Strings -->
<xml:xpath-splitter input-channel="messagesContainingOrders"
                    output-channel="orders"
                    create-documents="true">
   <xml:xpath-expression expression="//order"/>
</xml:xpath-splitter>

<xml:xpath-router input-channel="orders”>
   <xml:xpath-expression expression="/order/@type"/>
</xml:xpath-router>
```

turns Nodes into Documents

© Copyright 2014 Pivotal. All rights reserved.

XPath Support

```
<filter input-channel="orders" output-channel="typedOrders"
        ref="selector"/>

<xml:xpath-selector id="selector" evaluation-result-type="boolean">
   <xml:xpath-expression expression="boolean(/order/@type)"/>
</xml:xpath-selector>
```

© Copyright 2014 Pivotal. All rights reserved.

XSLT Support

```
<xml:xslt-transformer id="usingResource"
    input-channel="orders10"
    output-channel="orders"
    xsl-resource="file:${xsl-dir}/order1.0to1.1translation.xsl"/>

<xml:xslt-transformer id="usingTemplates"
    input-channel="orders10"
    output-channel="orders"
    xsl-templates="templatesBean"
    result-transformer="resultTransformerBean"/>
```

javax.xml.transform.Templates

© Copyright 2014 Pivotal. All rights reserved.

OXM Support

- Spring Integration builds on top of Spring OXM
- Spring's object-to-XML abstraction
  - Marshaller and Unmarshaller interfaces
  - Generic XmlMappingException hierarchy
  - Implementations for JAXB 1 & 2, JiBX, XMLBeans, XStream and Castor
    - With namespace support for some

© Copyright 2014 Pivotal. All rights reserved.

Spring OXM Interfaces

```
public interface Marshaller {
    void marshal(Object graph, Result result)
            throws XmlMappingException, IOException;
}
```

javax.xml.transform.Result

```
public interface Unmarshaller {
    Object unmarshal(Source source)
            throws XmlMappingException, IOException;
}
```

javax.xml.transform.Source

Result and Source may be DOM, SAX, or Stream

© Copyright 2014 Pivotal. All rights reserved.

ResultTransformer

- Marshallers write XML to Result instance
  - DOMResult by default, can be changed
- 'Result' typically bad XML message payload type
  - Needs instanceof checks and downcasting to process
- ResultTransformer converts to suitable type
  - ResultToStringTransformer
  - ResultToDocumentTransformer

© Copyright 2014 Pivotal. All rights reserved.

Marshalling and Unmarshalling

```
<si-xml:unmarshalling-transformer
        input-channel="xmlInputChannel"
        output-channel="objectOutputChannel"
        unmarshaller="unmarshaller"/>

<si-xml:marshalling-transformer
        input-channel="objectInputChannel"
        output-channel="xmlOutputChannel"
        marshaller="marshaller"
        result-transformer="resultToStringTransformer"/>

<bean id="resultToStringTransformer"
  class="org.sfw...xml.transformer.ResultToStringTransformer" />
```

© Copyright 2014 Pivotal. All rights reserved.

XML Payloads

- XML Message payloads can have various types
- XML endpoints convert where needed
  - All Support Node and String
  - XPath Splitter: plus File
  - XPath Router, Selector, Transformer and Header Enricher: plus File and DOMSource
  - XML Validating Selector: plus Source
  - XML Unmarshaller: plus File and Source
- For other payloads use explicit transformer

© Copyright 2014 Pivotal. All rights reserved.

Summary

- Splitting and Aggregating
  - symmetric, customizable
- Configurable dispatching for point-to-point subscribable channels
- XPath, XSLT and OXM support for dealing with XML

© Copyright 2014 Pivotal. All rights reserved.

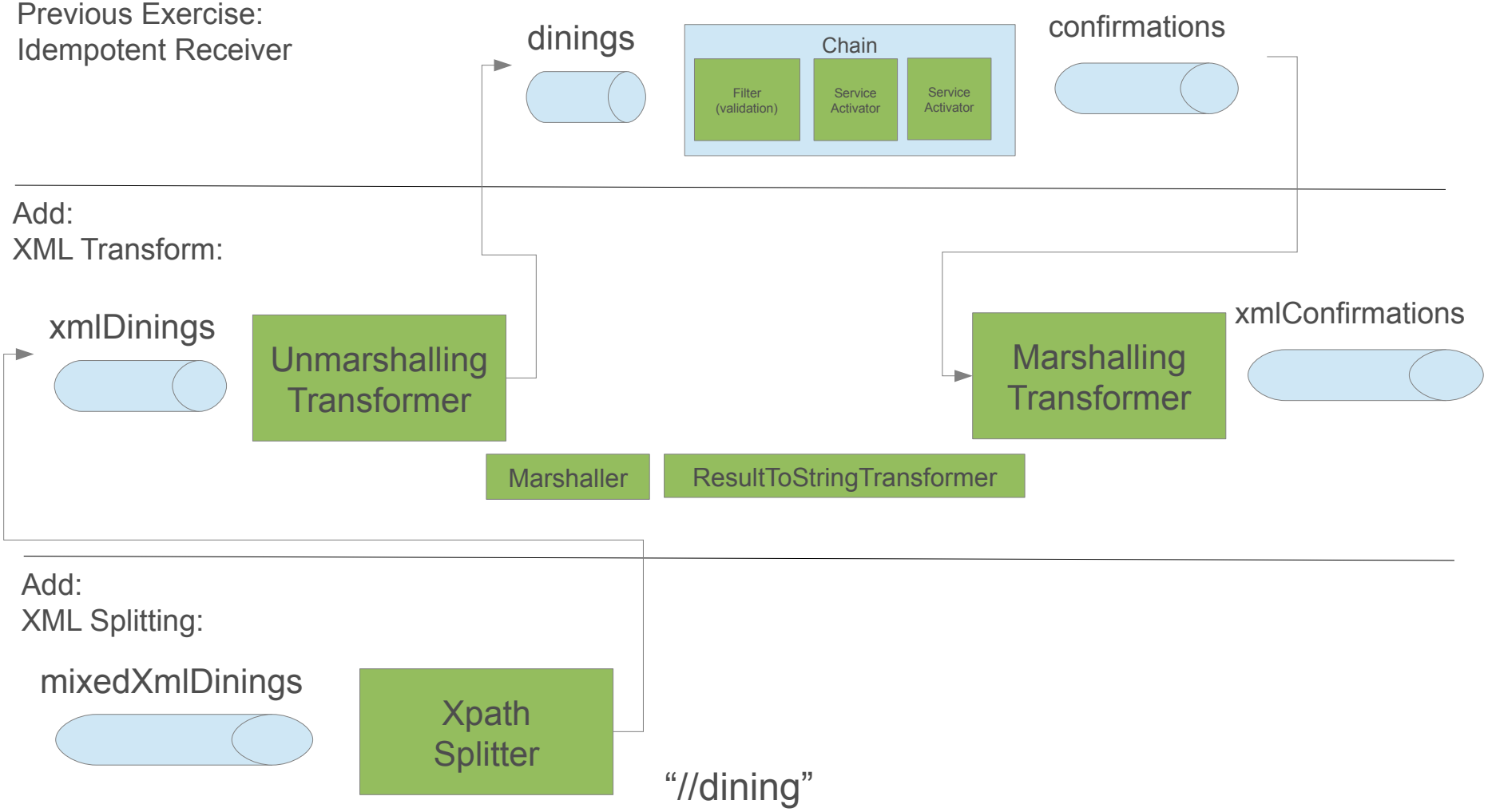
Lab

XML Transformation and Splitting

© Copyright 2014 Pivotal. All rights reserved.

Exercise

Previous Exercise:
Idempotent Receiver



© Copyright 2014 Pivotal. All rights reserved.

Spring Integration 2.0 New Advanced Features

Datatype Channels,
Persisting Messages,
Control Bus and Message History

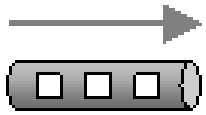
© Copyright 2014 Pivotal. All rights reserved.

Topics

- **Datatype Channels**
- Persisting Messages
- Control Bus
- Message History

© Copyright 2014 Pivotal. All rights reserved.

Datatype Channels



- By default channels can hold any type of Message
- Can optionally be typed
  - Restricts acceptable messages to those with payload of the given type(s)
  - Attempts conversion for messages of different type

```
<channel id="dinings" datatype="rewards.Dining"/>
```

```
<channel id="wholeNumberChannel"
  datatype="java.lang.Integer,
            java.math.BigInteger,
            java.util.concurrent.atomic.AtomicInteger"/>
```


© Copyright 2014 Pivotal. All rights reserved.

Message Conversion

- SI uses **integrationConversionService** bean (named ConversionService) when defined
  - Must have suitable Converter

must have this name

```
<bean id="integrationConversionService"
   class="org.springframework.context.support.ConversionServiceFactoryBean">
   <property name="converters">
      <bean class="com.example.StringToFooConverter"/>
   </property>
</bean>
```

Allows sending Message<String> to channel of type Foo

```
public interface Converter<S, T> {
   T convert(S source);
}
```

Spring 3.0 introduced Converter interface

© Copyright 2014 Pivotal. All rights reserved.

Integration ConversionService

- ConversionServiceFactoryBean registers many default Converters
  - See *ConversionServiceFactory.addDefaultConverters()*
- Shortcut to ensure integrationConversionService exists and add a custom converter to it
  - Multiple converter elements can be used

'ref' attribute also supported

```
<int:converter>
  <bean class="com.example.StringToFooConverter"/>
</int:converter>
```

Rely on ObjectToObjectConverter conventions to prevent having to code dedicated Converters for your classes

© Copyright 2014 Pivotal. All rights reserved.

Topics

- Datatype Channels
- **Persisting Messages**
- Control Bus
- Message History

© Copyright 2014 Pivotal. All rights reserved.

Persistent Messages

- QueueChannels keep messages in memory by default
- As do stateful endpoints like aggregators

- Custom Message Store or channel types allow persisting messages in a database or message broker

© Copyright 2014 Pivotal. All rights reserved.

Persistent Messages Use Cases

- Preventing message loss on application shutdown
- Transactional message processing
- Dealing with large message payloads
- Using pub-sub facilities of message broker
  - e.g. to address all nodes in a cluster

- All internal to application
  - Entire Message stored, not just payload
  - Use adapters and gateways for integration with other applications through databases and brokers

© Copyright 2014 Pivotal. All rights reserved.

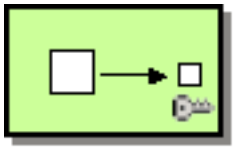
JDBC Message Store

- JdbcMessageStore stores messages in relational DB
  - DB schema ships with spring-integration-jdbc jar
  - (De)serializes messages to single column
    - Uses Java Serialization by default, so ensure headers and payload implement Serializable!

```
<int-jdbc:message-store id="messageStore" data-source="dataSource" />

<channel id="persistentChannel">
   <queue message-store="messageStore"/>
</channel>

<aggregator id="longLivedAggregator" message-store="messageStore" … />
```


© Copyright 2014 Pivotal. All rights reserved.

Claim Check Pattern



- Deals with large payloads using Message Store
  - Payload stored using identifier (UUID)
  - SI payload set to identifier
  - Identifier acts as 'claim check' to query Store for real payload when needed
- Supported using pair of transformers

```
<claim-check-in id="claim-in" message-store="jdbcMessageStore"
                input-channel="checkin" output-channel="checkedin"/>

<claim-check-out id="claim-out" message-store="jdbcMessageStore"
                 input-channel="checkout" output-channel="checkedout"
                 remove-message="true"/>
```

Claim only once: defaults to false

© Copyright 2014 Pivotal. All rights reserved.

JMS Channels

- Channels can be backed by JMS Queue or Topic
  - For Queues, channel can be pollable or subscribable
  - Topics are always subscribable
- Uses regular Spring JMS support
  - Same configuration options, but defaults to local TXs
  - Defaults to ObjectMessages (SimpleMessageConverter)
    - So implement Serializable for headers and payload!

```
<int-jms:channel id="pollableQueueChannel" queue-name="si-queue"
                 message-driven="false"/>
```

pollable (default is true subscribable)

```
<int-jms:publish-subscribe-channel id="pubsubChannel"
                                   topic-name="si-topic"/>
```

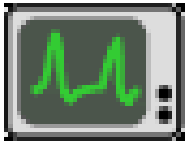

© Copyright 2014 Pivotal. All rights reserved.

Topics

- Datatype Channels
- Persisting Messages
- **Control Bus**
- Message History

© Copyright 2014 Pivotal. All rights reserved.

Control Bus



- Manage and monitor channels and endpoints by sending messages to a special component
  - Payloads are SpEL expressions
  - Access to any Spring bean in ApplicationContext
  - But only @ManagedOperation/-Attribute annotated methods
    - As well as Spring's Lifecycle start() & stop() and CustomizableThreadCreator subtype methods
  - Acts as dynamic service activator with built-in restrictions

```
<control-bus input-channel="operationChannel" />
```

optional output-channel for results

```
messagingTemplate.convertAndSend("operationChannel",
                                 "@jmsListenerContainer.stop()");
```


© Copyright 2014 Pivotal. All rights reserved.

Groovy Control Bus

- Groovy instead of SpEL supported as well
  - Auto-exposes beans annotated with @ManagedResource and instances of LifeCycle or CustomizableThreadCreator

```
<int-groovy:control-bus input-channel="operationChannel" />
```

```
messagingTemplate.convertAndSend("operationChannel",
                                 "jmsListenerContainer.stop()");
```


© Copyright 2014 Pivotal. All rights reserved.

Topics

- Datatype Channels
- Persisting Messages
- Control Bus
- **Message History**

© Copyright 2014 Pivotal. All rights reserved.

Message History

- Track components a message passes through
  - For auditing, debugging, profiling, etc.
- Enabled using single tag: <message-history />
- Adds List<Properties> 'history' header
  - With 'name', 'type' and 'timestamp' Properties keys for each channel or endpoint
    - Defined as String constants of MessageHistory class
- Optionally limit tracked components by name

```
<message-history tracked-components="*Gateway, sample*, foo" />
```

© Copyright 2014 Pivotal. All rights reserved.

Message History Sample

- Log history of every message for given channel:

```
<channel id="someChannel"/>

<message-history />

<channel-interceptor pattern="someChannel">
   <wire-tap channel="historyLogChannel"/>
</channel-interceptor>

<logging-channel-adapter id="historyLogChannel"
  expression="payload + ': ' + headers.history.![name + ' - ' + timestamp]"/>
```

Collection projection

```
messagingTemplate.convertAndSend("someChannel", "hello");
```

```
INFO : org.springframework.integration.handler.LoggingHandler -
hello: [someChannel - 1312462840910, historyLogChannel - 1312462840915,
historyLogChannel.adapter - 1312462840916]
```


© Copyright 2014 Pivotal. All rights reserved.

Summary

- Spring Integration 2 supports many advanced Enterprise Integration Patterns out of the box
  - Datatype Channels, Message Store, Claim Check, Control Bus, Message History
- Adding support requires no code changes
  - But keep serialization requirements in mind

© Copyright 2014 Pivotal. All rights reserved.

Introducing Spring Batch

Quickstart guide to offline processing with Spring Batch

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Batch and offline processing**
- Spring Batch high-level overview
- Job parameters and job identity
- Quick start using Spring Batch
- Readers, Writers & Processors
- JDBC Item Readers

© Copyright 2014 Pivotal. All rights reserved.

Batch Jobs

Differ from online/real-time processing applications:

- Long-running
  - Often outside office hours
- Non-interactive
  - Often include logic for handling errors or restarts
- Process large volumes of data
  - More than fits in memory or a single transaction

© Copyright 2014 Pivotal. All rights reserved.

Batch and offline processing

- Close of business processing
  - Order processing
  - Business reporting
  - Account reconciliation
- Import/export handling
  - a.k.a. ETL jobs (Extract-Transform-Load)
  - Instrument/position import
  - Data warehouse synchronization
- Large-scale output jobs
  - Loyalty scheme emails
  - Bank statements

© Copyright 2014 Pivotal. All rights reserved.

Batch Domain

- The batch domain adds some value to a plain business process by introducing new concepts:
  - A job describes a series of steps to be done.
  - A step is an independent unit of processing
  - A job instance can be restarted after a failure – a new job execution
  - Each job execution has a start time, stop time, status
  - The job instance's status = the status of its latest job execution
  - Each execution can tell us how many items were processed, how many commits, rollbacks, skips

© Copyright 2014 Pivotal. All rights reserved.

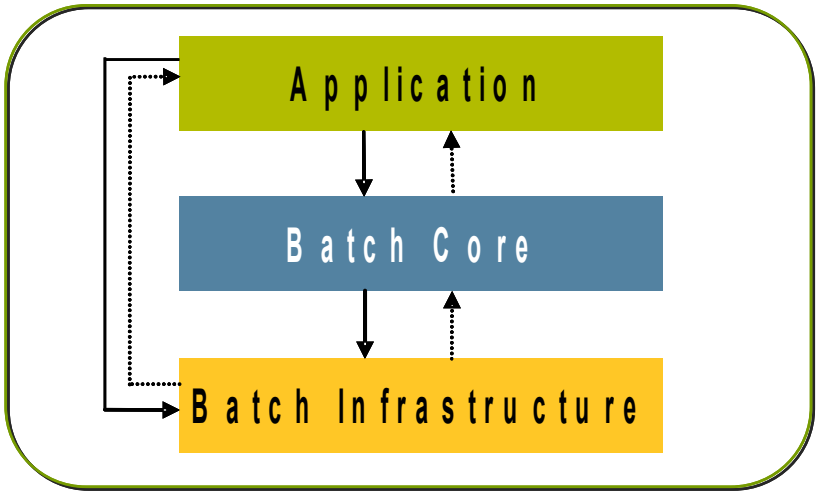
Topics in this Session

- Batch and offline processing
- **Spring Batch high-level overview**
- Job parameters and job identity
- Quick start using Spring Batch
- Readers, Writers & Processors
- JDBC Item Readers

© Copyright 2014 Pivotal. All rights reserved.

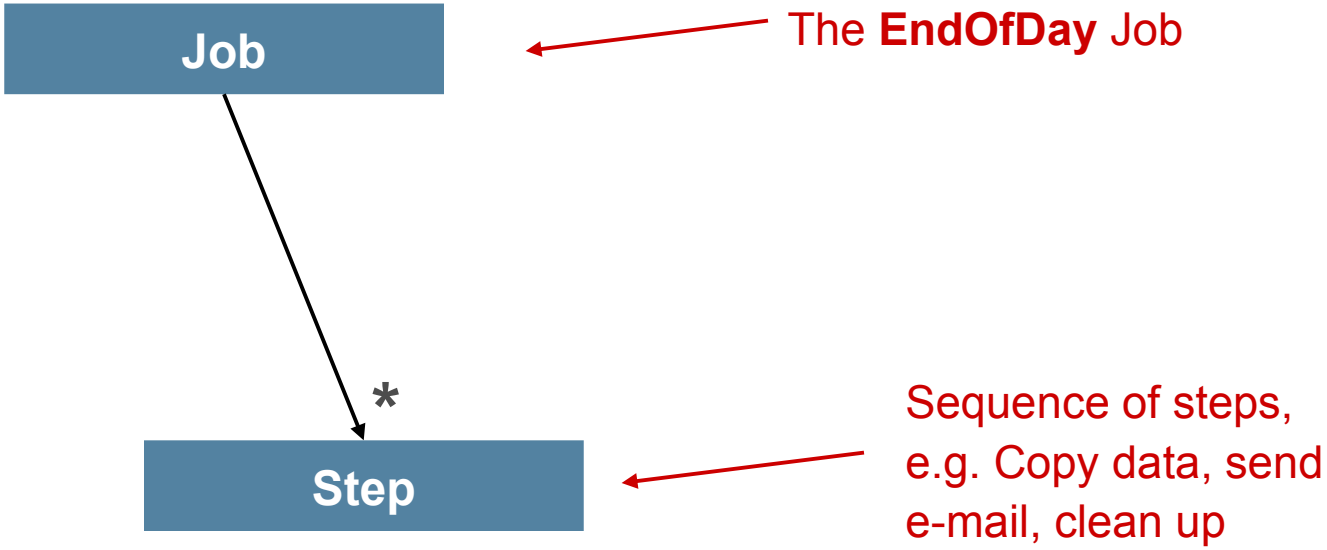
Spring Batch Overview

- The Spring programming model applied to batch processing
  - Don’t write code that doesn’t make you money
- Project page: http://projects.spring.io/spring-batch/
- Implementation of JSR-352



© Copyright 2014 Pivotal. All rights reserved.

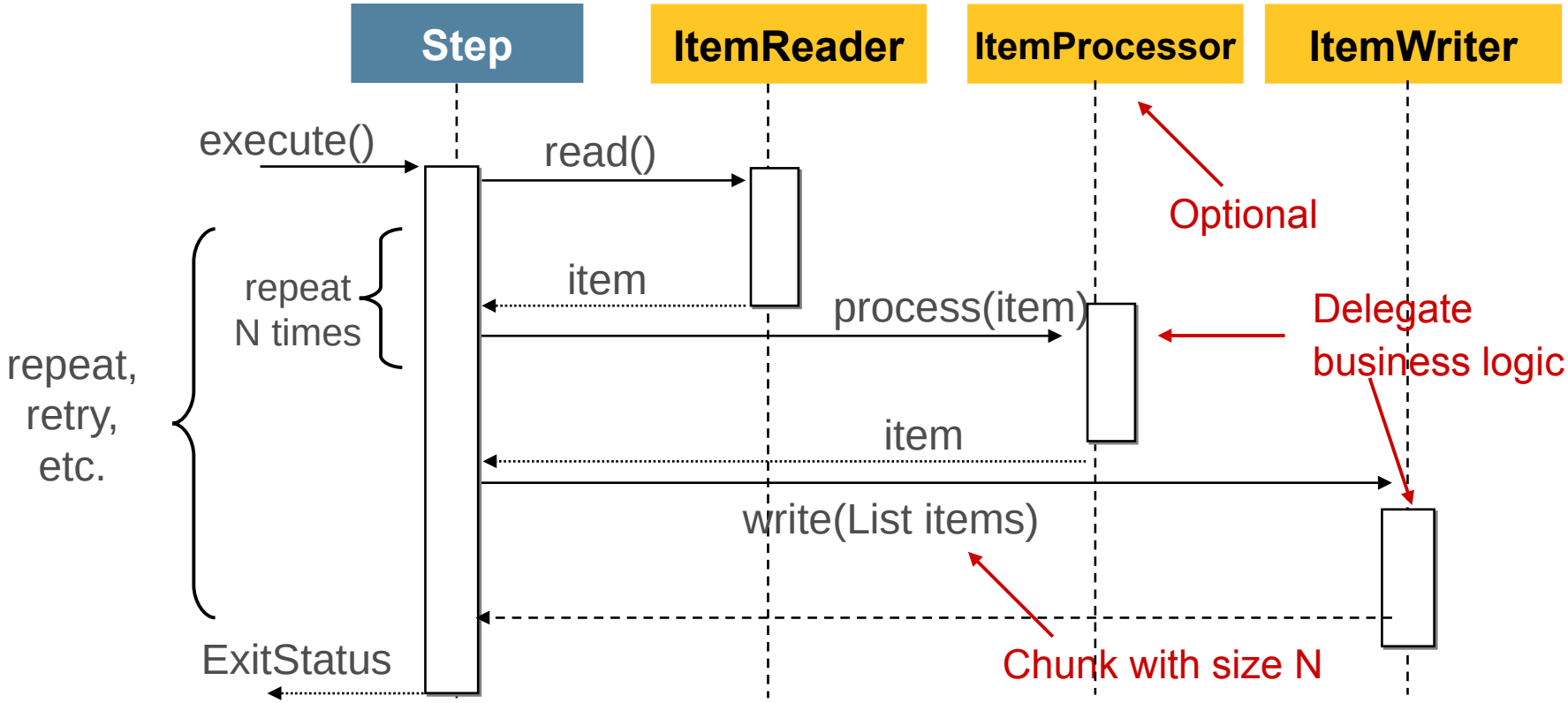
Job and Step



© Copyright 2014 Pivotal. All rights reserved.

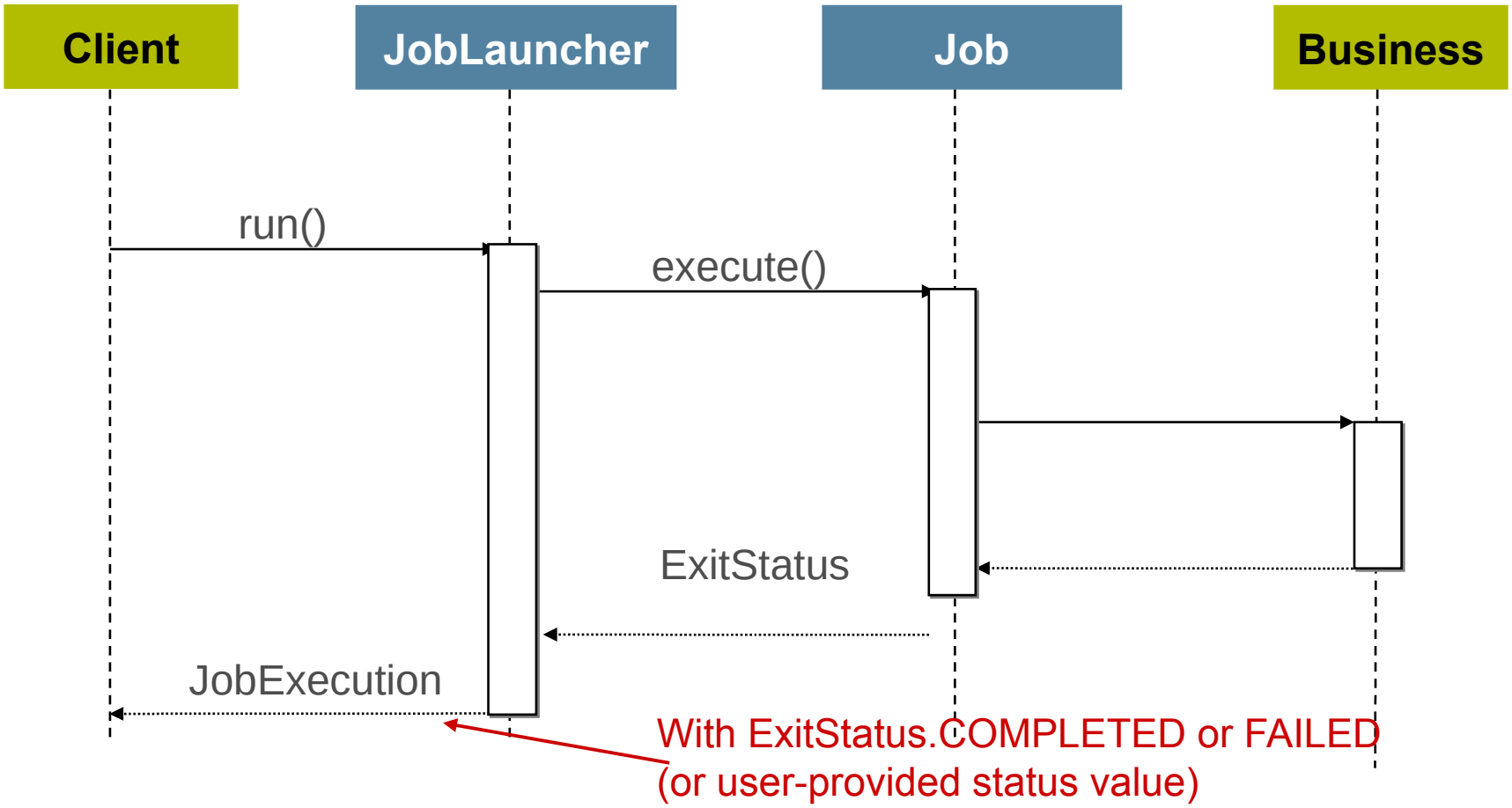
Chunk-Oriented Processing

- Input-output can be grouped together
- Input collects Items before outputting: *Chunk-Oriented Processing*
- Optional ItemProcessor



© Copyright 2014 Pivotal. All rights reserved.

JobLauncher



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Batch and offline processing
- Spring Batch high-level overview
- **Readers, Writers & Processors**
- Job parameters and job identity
- Quick start using Spring Batch
- JDBC Item Readers

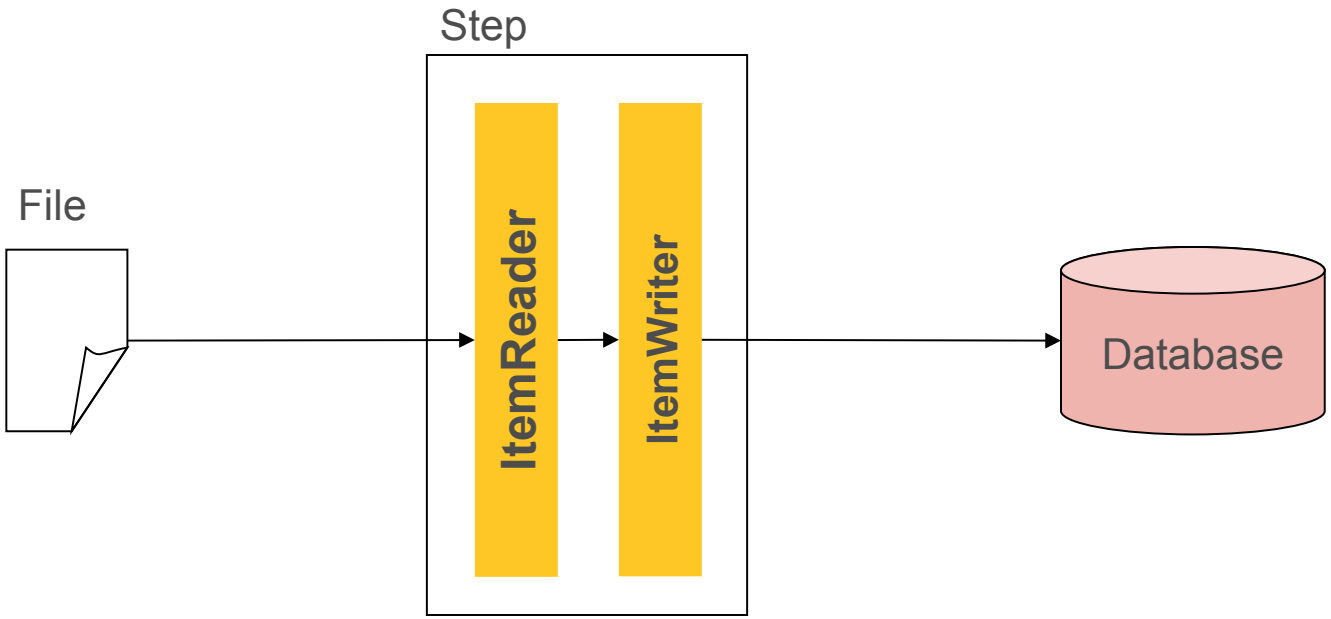
© Copyright 2014 Pivotal. All rights reserved.

Readers and Writers

- Spring Batch provides many implementations of ItemReader and ItemWriter, e.g.
  - Flat files
  - XML
  - JDBC: cursor & driving query
  - Hibernate
  - JMS
- Some simple jobs can be implemented with off-the-shelf components

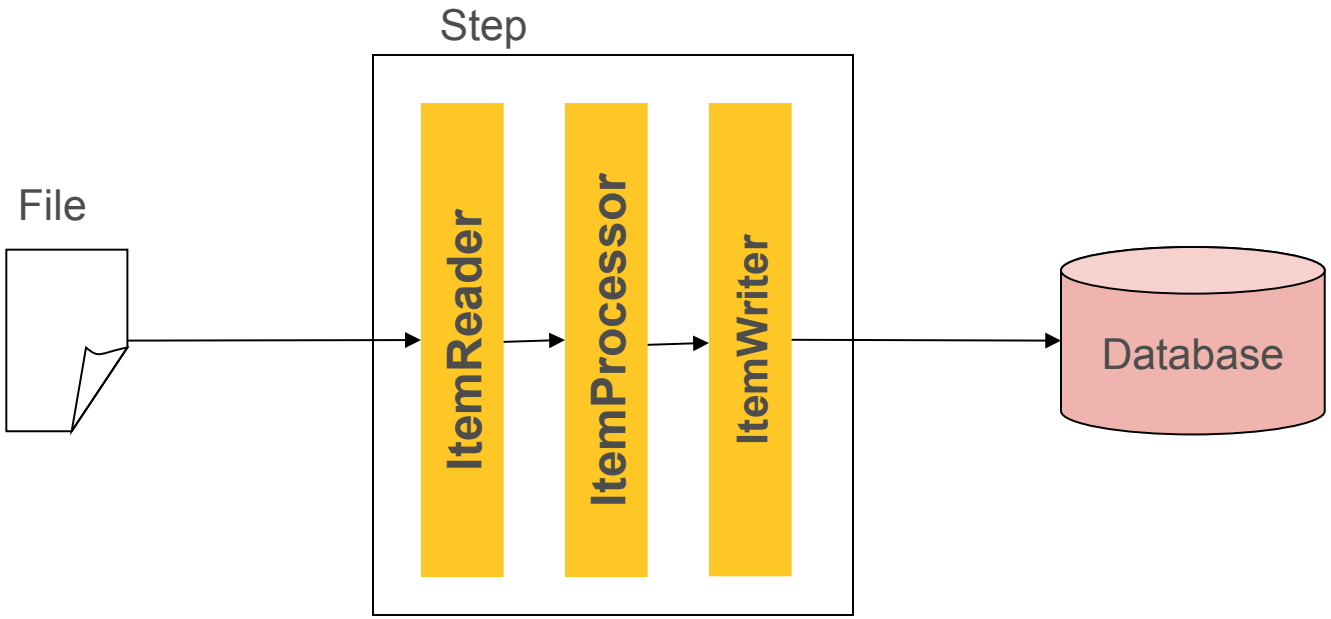
© Copyright 2014 Pivotal. All rights reserved.

Simple Copy Job



© Copyright 2014 Pivotal. All rights reserved.

Copy Job with Processing



© Copyright 2014 Pivotal. All rights reserved.

More Complex Use Cases

- It's very common to use an off-the-shelf reader and writer
- More complex jobs often require custom readers or writers
- ItemProcessor is often used if there's a need to delegate to existing business logic
- Use a writer if it's more efficient to process a complete chunk

© Copyright 2014 Pivotal. All rights reserved.

ItemReader

- The interface is parameterized, so e.g.

```
public class DiningItemReader implements ItemReader<Dining> {

    public Dining read() {
        // Read a Dining record from somewhere...
        return dining;
    }
    …

}
```


© Copyright 2014 Pivotal. All rights reserved.

ItemProcessor

- The interface is parameterized
  - perhaps the item that was read is transformed, or other business logic delegation occurred, resulting in an object of a different type

```
public class DiningItemProcessor implements
                ItemProcessor<XMLDining, Dining> {

   public Dining process(XMLDining xmlDining) {
      // transform the data...
      return dining;
   }
   …
}
```


© Copyright 2014 Pivotal. All rights reserved.

ItemWriter

- Also parameterized; notice the chunk (i.e. a List of items) as the parameter!

```
public class DiningItemWriter implements ItemWriter<Dining> {

    public void write(List<? extends Dining> dinings) {
        // Business logic here and write out to e.g. database
    }
    …

}
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Batch and offline processing
- Spring Batch high-level overview
- Readers, Writers & Processors
- **Job parameters and job identity**
- Quick start using Spring Batch
- JDBC Item Readers

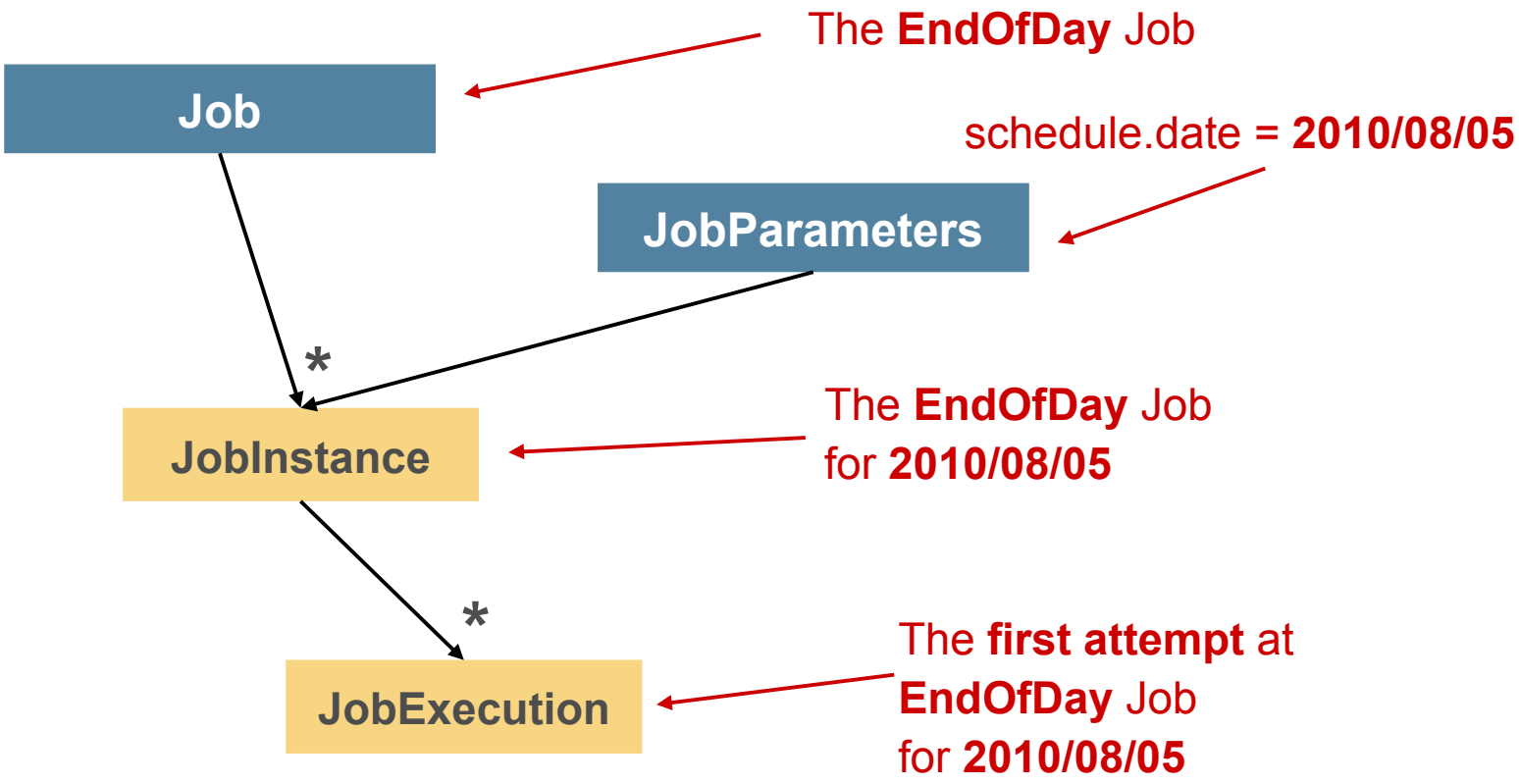
© Copyright 2014 Pivotal. All rights reserved.

Job Identity

- When you launch a job, what is it that you launch?
- If it fails how would you identify it?
- How can we track the eventual successful execution?

© Copyright 2014 Pivotal. All rights reserved.

Job Identity and Parameters



© Copyright 2014 Pivotal. All rights reserved.

Job Parameters

- Provide uniqueness for each Job Instance
  - JobInstanceAlreadyCompletedException
  - Often a natural unique parameter such as run Date/Time
  - Can use a JobParametersIncrementer to create uniqueness
- Framework allows for non-identifying parameters
  - ...Such as folder to read files from

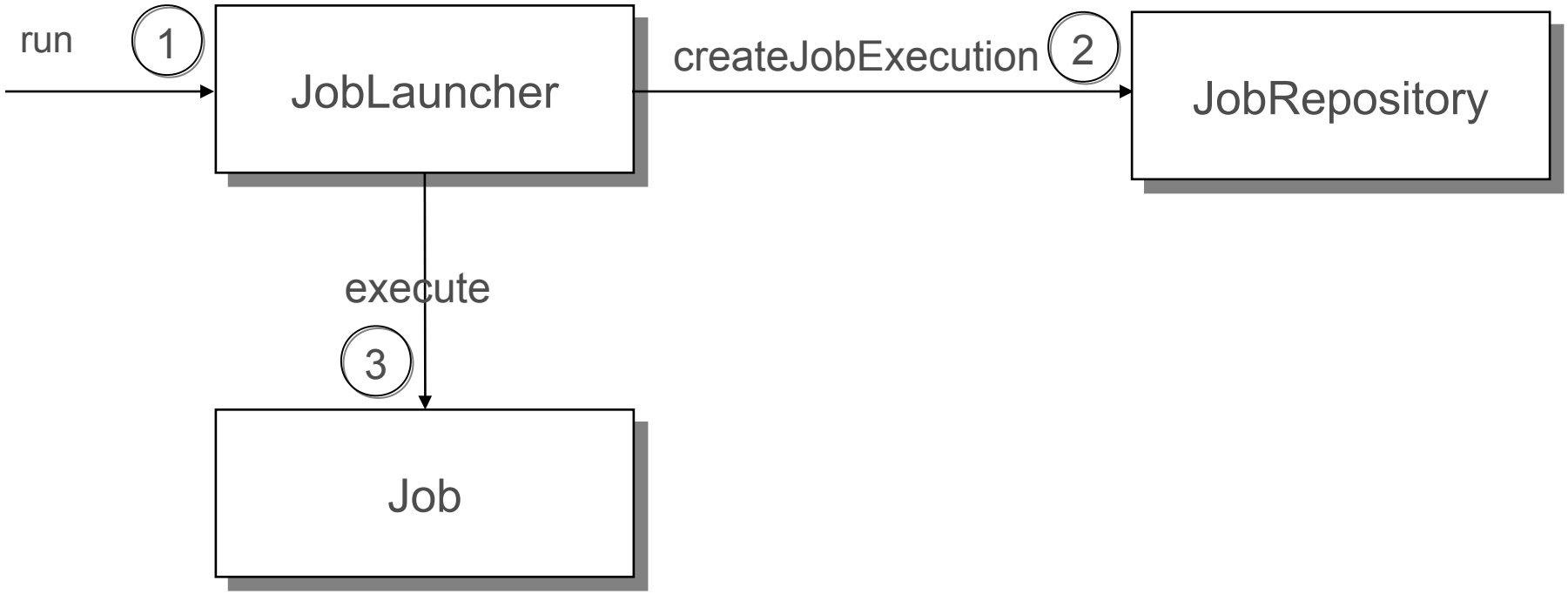
© Copyright 2014 Pivotal. All rights reserved.

Job Persistence

- Need to track status of job executions
- Readers and writers are often stateful, so need to persist e.g. intermediate state
- Batch meta-data usually stored in Database
  - Use in-memory DB if no need for permanent storage.
- Core interface = JobRepository

© Copyright 2014 Pivotal. All rights reserved.

JobLauncher and JobRepository



© Copyright 2014 Pivotal. All rights reserved.

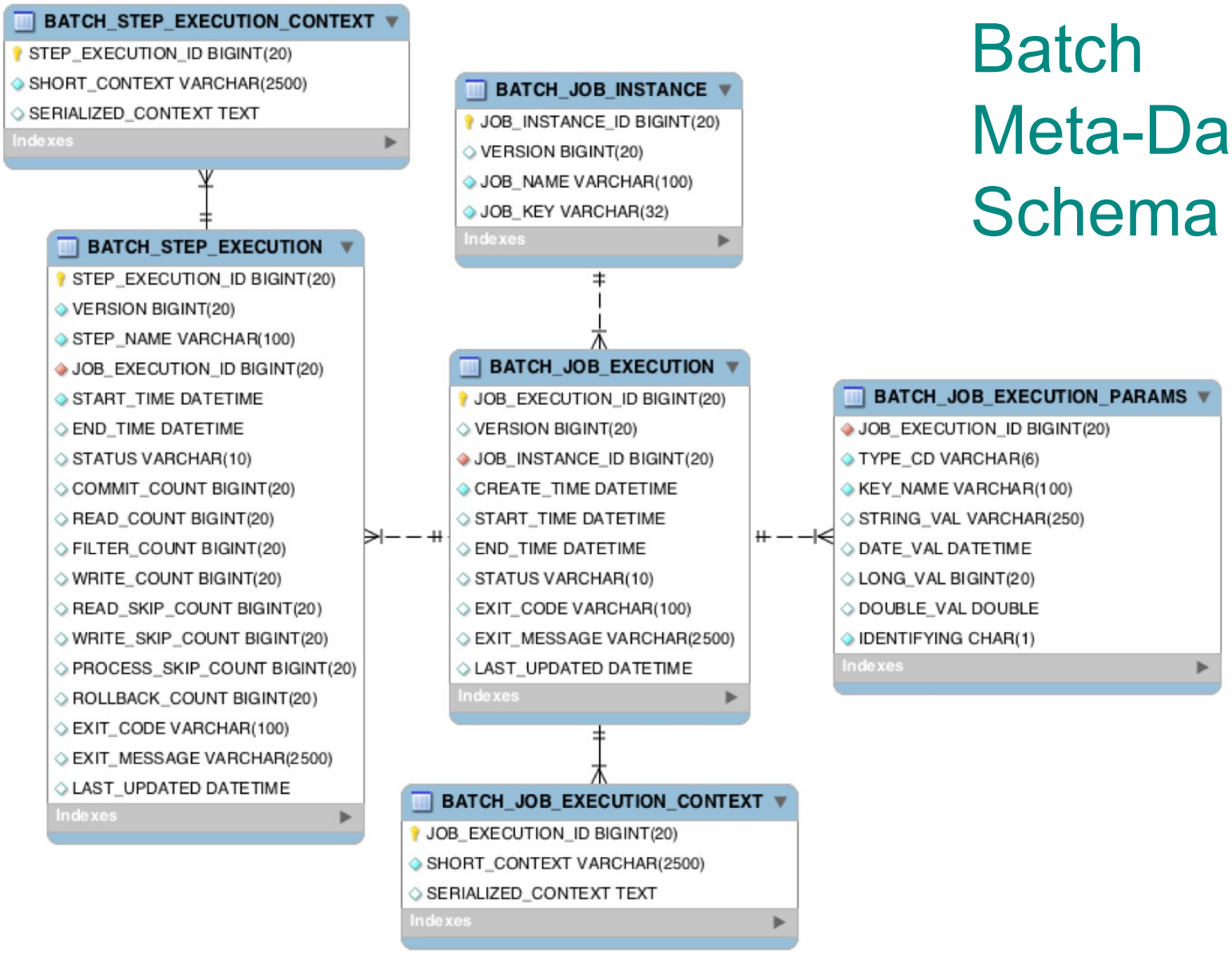
JobRepository Practicalities

- JobRepository is really an **internal** interface
  - no need for users to know details
- Still need to create an instance of JobRepository
- Spring Batch provides namespace support:

```
<!-- defaults to 'transactionManager' for transaction-manager,
     'dataSource' for data-source and 'BATCH_' for table-prefix -->
<batch:job-repository id="jobRepository" />
```

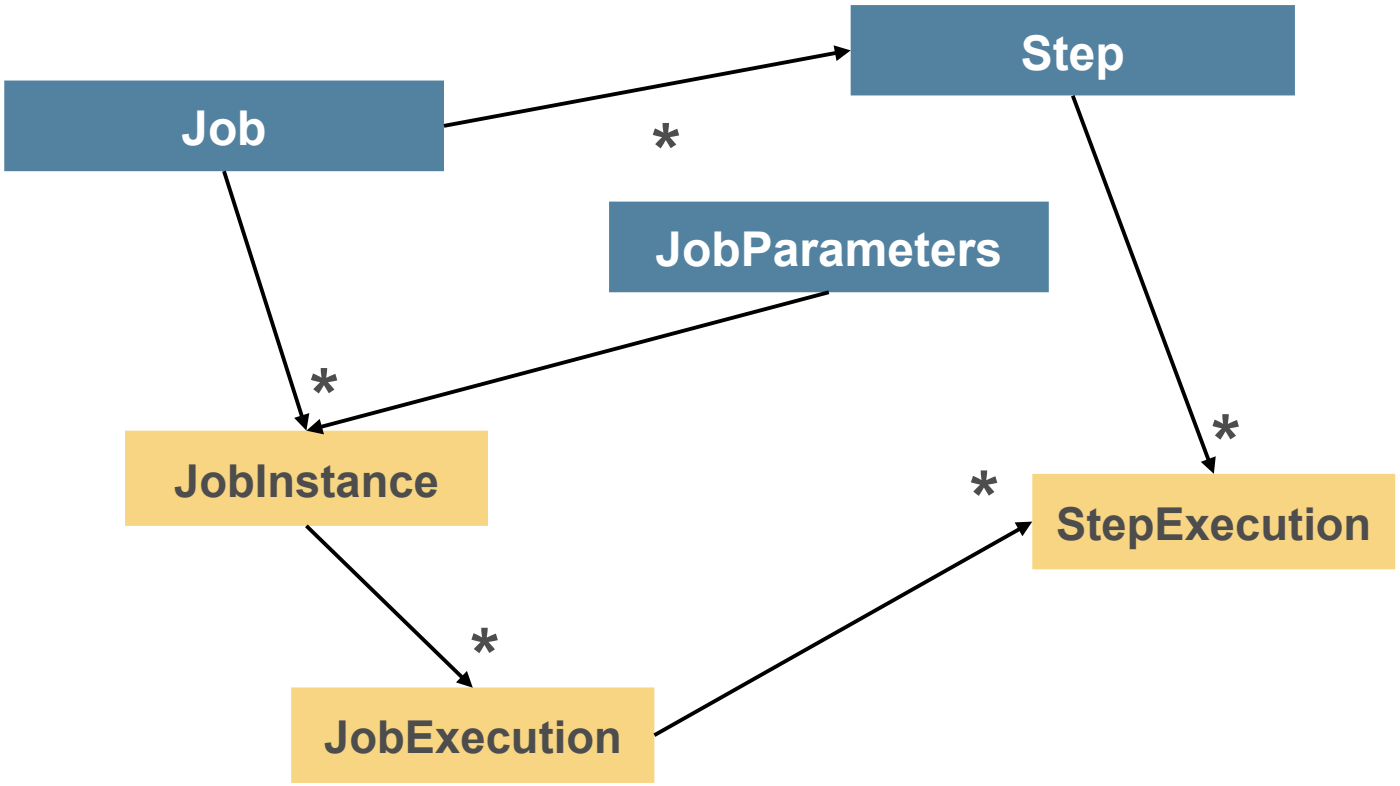
© Copyright 2014 Pivotal. All rights reserved.

Batch Meta-Data Schema



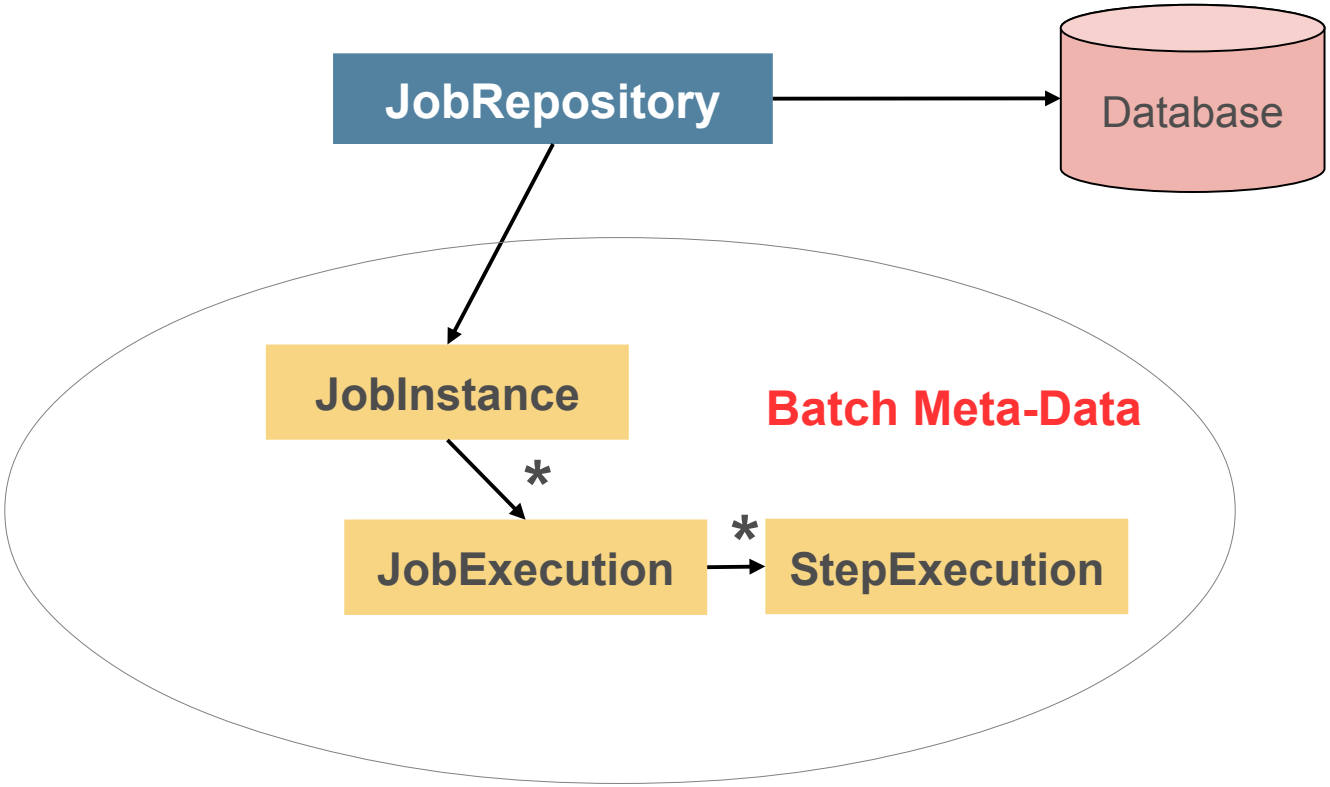
© Copyright 2014 Pivotal. All rights reserved.

Extended Picture: Job and Step



© Copyright 2014 Pivotal. All rights reserved.

JobRepository and Batch Meta-Data



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Batch and offline processing
- Spring Batch high-level overview
- Readers, Writers & Processors
- Job parameters and job identity
- **Quick start using Spring Batch**
- JDBC Item Readers

© Copyright 2014 Pivotal. All rights reserved.

Spring Batch Quickstart

- Configure the readers, processors and writers
- Configure a Job with Step(s)
- Configure a JobLauncher
- Load and run the job
  - Write a wrapper for the JobLauncher
  - Use off-the-shelf wrapper (CommandLineJobRunner, ...)
- Result is in return value from the JobLauncher

© Copyright 2014 Pivotal. All rights reserved.

Configure JobLauncher

```
<beans>

   <bean id="jobLauncher" class="org.sfw..SimpleJobLauncher">
      <property name="jobRepository" ref="jobRepository" />
   </bean>

   <batch:job-repository id="jobRepository" />

</beans>
```

© Copyright 2014 Pivotal. All rights reserved.

Configure Reader and Writer

```
<beans>

  <bean id="itemReader" class="org.sfw..FlatFileItemReader">
    <property name="fieldSetMapper"ref="customMapper" />
    <property name="resource" value="file://home/jobs/data/input.csv"/>
    ...
  </bean>

  <bean id="itemWriter" class="org.sfw..FlatFileItemWriter">
    <property name="fieldSetCreator"ref="customCreator" />
    ...
  </bean>

</beans>
```

© Copyright 2014 Pivotal. All rights reserved.

Configure a Job with Step(s)

- Using the Spring Batch namespace support :

```
<beans:beans>

   <job id="endOfDayJob" >
      <step id="copyStep">
         <tasklet>
            <chunk reader="itemReader" writer="itemWriter"
                   commit-interval="${chunk.size}"/>
         </tasklet>
      </step>
      ...
   </job>

</beans:beans>
```

© Copyright 2014 Pivotal. All rights reserved.

Launch the Job

```
// Create the application from the configuration
ApplicationContext context =
    new ClassPathXmlApplicationContext("application-config.xml");
```

```
// Look up the job launcher and job
@Autowired JobLauncher jobLauncher;
@Autowired @Qualifier("endOfDayJob") Job job;

// …

// Launch the job
JobExecution execution =
        jobLauncher.run(job, new JobParameters());
```

Use this to see the exit status

© Copyright 2014 Pivotal. All rights reserved.

CommandLineJobRunner

- Framework-provided command line utility
- 2 mandatory parameters: XML config location and job id

```
# Command line prompt:

$ java -classpath ... org.sfw...CommandLineJobRunner
    application-config.xml endOfDayJob
```

(All on the same line)

© Copyright 2014 Pivotal. All rights reserved.

Adding Parameters

- With JobLauncher

```
JobParametersBuilder builder  = new JobParametersBuilder();
builder.addString("file.locator", "2008-05-05");
...
JobExecution execution =
        jobLauncher.run(job, builder.toJobParameters());
```


© Copyright 2014 Pivotal. All rights reserved.

Adding Parameters

- With CommandLineJobRunner

```
$ java -classpath ... org.sfw...CommandLineJobRunner
   application-config.xml endOfDayJob
      run.id=1003AHQ7
      schedule.date(date)=2008/05/05
```

key(*type*)=value pairs at end of line
where *type* = string | date | long

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Batch and offline processing
- Spring Batch high-level overview
- Readers, Writers & Processors
- Job parameters and job identity
- Quick start using Spring Batch
- **JDBC Item Readers**

© Copyright 2014 Pivotal. All rights reserved.

DB Item Readers & Writers

- Useful when DB is primary input/output or to stage file-based I/O in temporary DB table
  - Flat file I/O in batch processes can cause difficulties
  - Read / discard much data during restart for files not formatted with fixed record lengths
  - Difficult to synchronize on output (non-transactional)

- Spring Batch provides DB ItemReaders and -Writers to help with this

We'll just talk about readers here

© Copyright 2014 Pivotal. All rights reserved.

DB Item Readers

- Cursor based
  - JdbcCursorItemReader
  - HibernateCursorItemReader
  - StoredProcedureItemReader
- Paging
  - JdbcPagingItemReader
  - JpaPagingItemReader
  - IbatisPagingItemReader

We'll just talk about the JDBC Implementations here;
refer to the Spring Batch Documentation for more information

© Copyright 2014 Pivotal. All rights reserved.

Cursor Based Item Readers

- JdbcTemplate + RowMapper callback load all data into memory
  - Not practical for large batch data sets
- Cursor-based Item Reader maps row at a time, using same RowMapper interface
- Reader's read() method returns mapped domain object
  - Data is 'streamed' - written and discarded according to the step's commit-interval
- Check JDBC driver for server-side cursor support

© Copyright 2014 Pivotal. All rights reserved.

JdbcCursorItemReader

```
<bean id="itemReader" class="org.spr...JdbcCursorItemReader">
  <property name="dataSource" ref="dataSource"/>
  <property name="sql"
            value="select ID, NAME, CREDIT from CUSTOMER"/>
  <property name="rowMapper">
    <bean class="com.acme.domain.CustomerCreditRowMapper"/>
  </property>
</bean>
```

Additional properties, including:
fetchSize, maxRows, driverSupportsAbsolute, …
Refer to documentation

© Copyright 2014 Pivotal. All rights reserved.

Paging Item Readers

- Alternative technique to cursor – uses distinct queries for each set (or page) of data
  - Start and end included in query
  - Syntax depends on DBMS
- JdbcPagingItemReader delegates to a PagingQueryProvider implementation
  - SqlPagingQueryProviderFactoryBean detects DBMS to create the appropriate provider
- More control over memory utilization than cursor based readers

© Copyright 2014 Pivotal. All rights reserved.

JdbcPagingItemReader

- You must provide
  - SELECT clause
  - FROM clause
  - Optional WHERE clause
  - Sort Key
- The sort key is used as the start/end
  - Reader keeps track of the last read sort key and adds it to the where clause for the next query
  - MUST BE UNIQUE!

© Copyright 2014 Pivotal. All rights reserved.

JdbcPagingItemReader

```
<bean id="itemReader" class="org.spr...JdbcPagingItemReader">
   <property name="dataSource" ref="dataSource"/>
   <property name="queryProvider">
      <bean class="org.spr...SqlPagingQueryProviderFactoryBean">
         <property name="dataSource"        ref="dataSource"/>
         <property name="selectClause"      value="select id, name, credit"/>
         <property name="fromClause"        value="from customer"/>
         <property name="whereClause"       value="where status='NEW'"/>
         <property name="sortKey"           value="id"/>
      </bean>
   </property>
   <property name="pageSize" value="1000"/>
   <property name="rowMapper">
      <bean class="com.example.CustomerMapper"/>
   </property>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Lab

Creating a simple 2-step job using Spring Batch

© Copyright 2014 Pivotal. All rights reserved.

Spring Batch Restart and Recovery

State & execution management for restarting and recovering from failures with Spring Batch

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **ExecutionContext**
- Stateful ItemReaders/Writers
- Reading Flat Files
- Sharing State Between Steps
- Intro to Skip, Retry, Repeat, Restart
- Other Listeners
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

ExecutionContext

- We need to know where a failure occurred to restart a batch process
- Job Repository meta data is used to determine the step at which the failure occurred
- Application Code (in reader/writer) needs to maintain state within a step (e.g. current chunk)
- Spring Batch can supply that data during restart

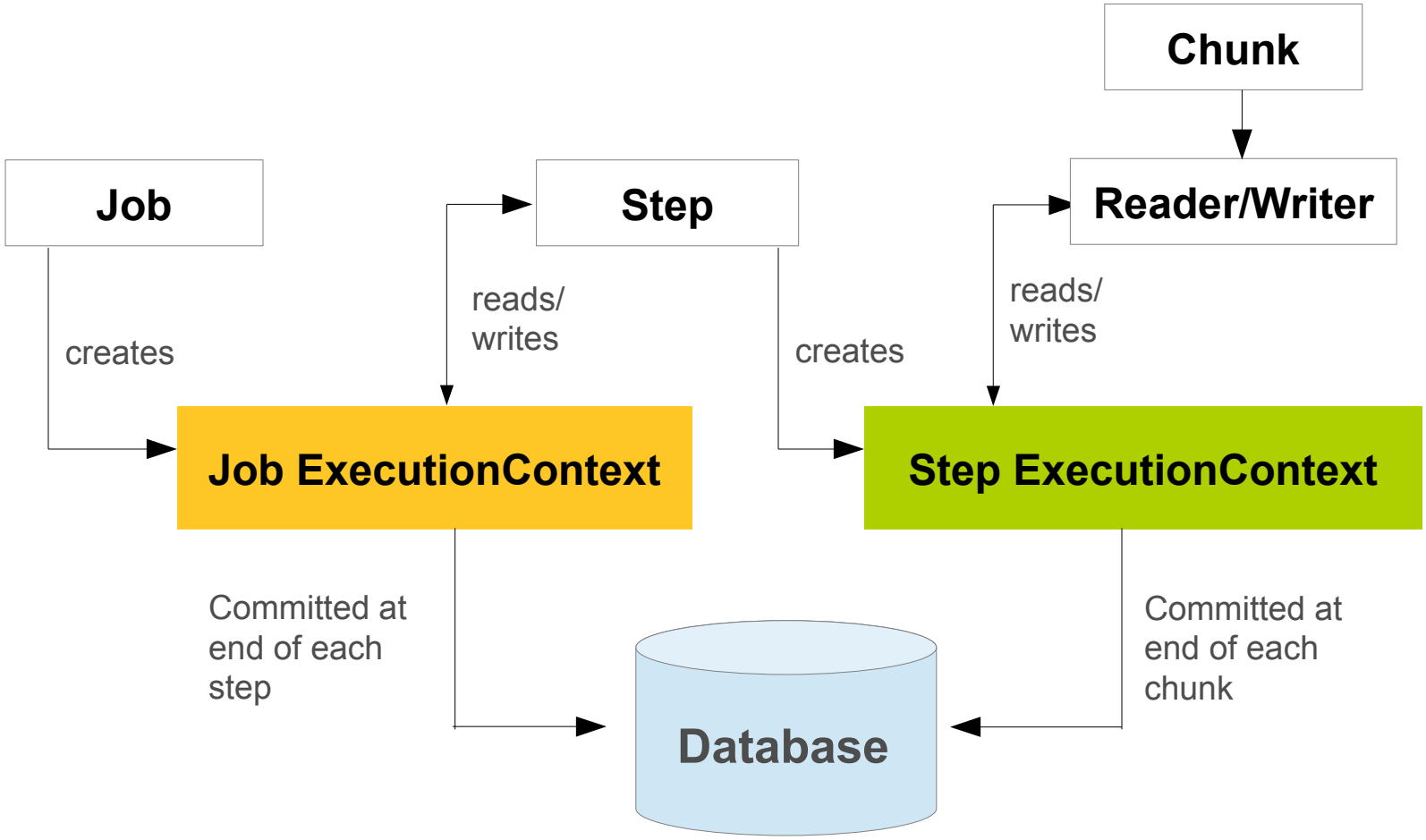
© Copyright 2014 Pivotal. All rights reserved.

ExecutionContext

- Key/Value pairs persisted by the framework
- During restart, provides initial state
- Job ExecutionContext
  - Committed at end of step
  - Can also be used to pass state between steps
- Step ExecutionContext
  - Committed at end of each chunk

© Copyright 2014 Pivotal. All rights reserved.

ExecutionContext



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- ExecutionContext
- **Stateful ItemReaders/Writers**
- Reading Flat Files
- Sharing State Between Steps
- Intro to Skip, Retry, Repeat, Restart
- Other Listeners
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

Stateful ItemReaders/-Writers
ItemStream

- If ItemReader implements ItemStream interface
  - open() - called before any read() calls
  - update() - called at the end of each chunk, before commit
  - close() - called at the end of the step
- Implemented by many built-in ItemReaders and -Writers

© Copyright 2014 Pivotal. All rights reserved.

ItemStream

```
public class DiningReader implements ItemReader<Dining>, ItemStream {
    private int filePosition;
    public Dining read() throws Exception {
        ...
        filePosition++;
        return dining;
    }
    public void open(ExecutionContext executionContext)
        throws ItemStreamException {
            filePosition = executionContext.getInt("position", 0);
    }
    public void update(ExecutionContext executionContext)
        throws ItemStreamException {
            executionContext.putInt("position", filePosition);
    }
    public void close() throws ItemStreamException { }
}
```

Initialize File Position from Last Run

Assumes single-threaded step

© Copyright 2014 Pivotal. All rights reserved.

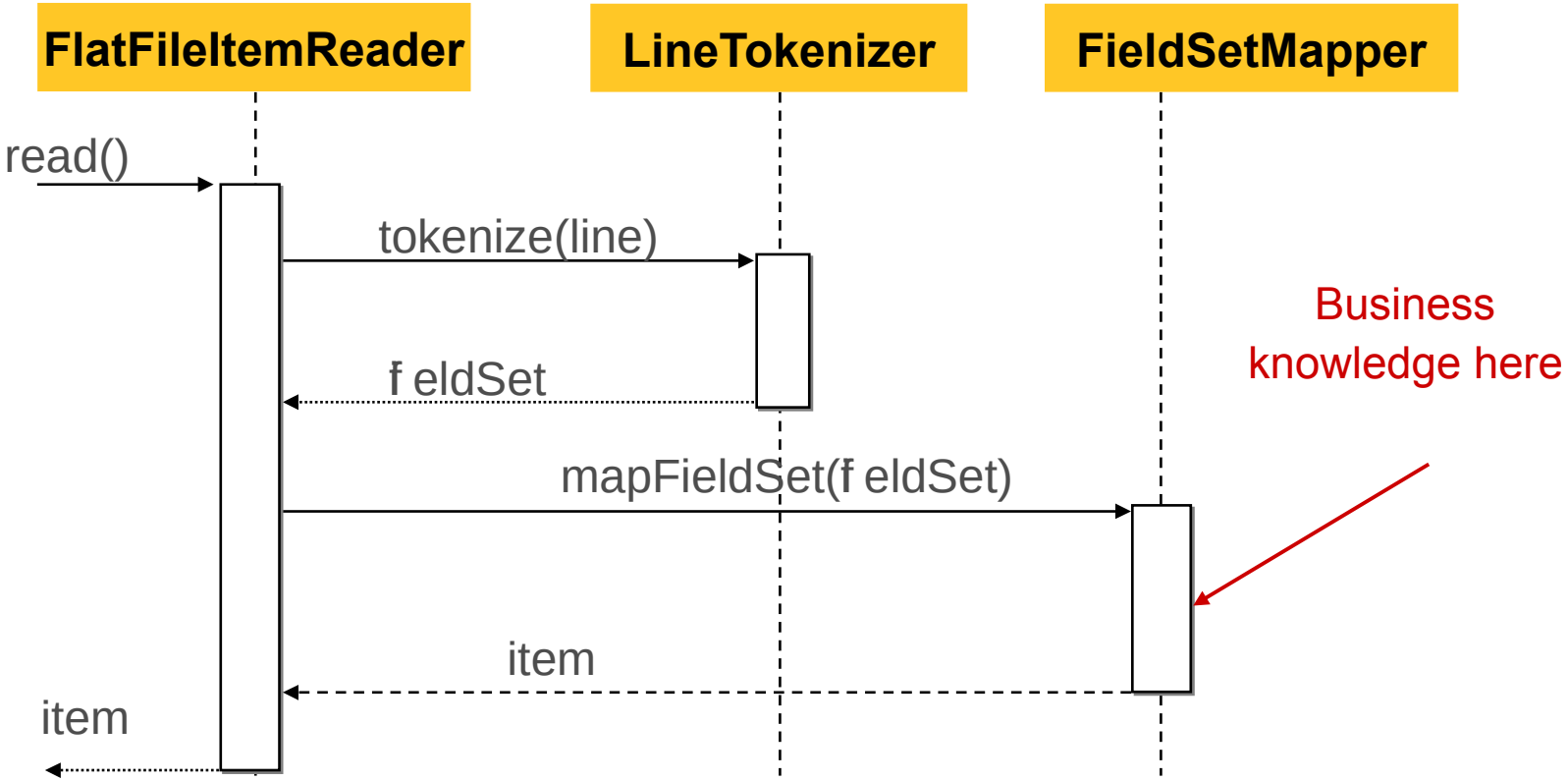
Topics in this Session

- ExecutionContext
- Stateful ItemReaders/Writers
- **Reading Flat Files**
- Sharing State Between Steps
- Intro to Skip, Retry, Repeat, Restart
- Other Listeners
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

FlatFileItemReader

- Stateful, remembers number of lines read
- Delegates to LineTokenizer and FieldSetMapper for parsing when used with DefaultLineMapper



© Copyright 2014 Pivotal. All rights reserved.

FieldSet

- Intermediate between line (record) in file and domain object
- Immutable wrapper for array of Strings
- Access fields by name or index, and by type

```
FieldSet fs = ... ;

Date date = fs.readDate(0, “dd/MM/yyyy”); // first field
Long number = fs.readLong(1);             // second field
String value = fs.readString(“city”);     // named field
String[] values = fs.getValues();         // all fields
```


© Copyright 2014 Pivotal. All rights reserved.

FieldSetMapper

- Maps a FieldSet to a domain Object

```
public class DiningFieldSetMapper
    implements FieldSetMapper<Dining> {

    public Dining mapFieldSet(FieldSet fs) {
        // Construct and and return a Dining object
    }

}
```


© Copyright 2014 Pivotal. All rights reserved.

FlatFileItemReader Configuration

```
<bean id="diningRequestsReader" class="org.sfw.batch.item.file.FlatFileItemReader">
  <property name="resource" value="file:batch-input.csv"/>
  <property name="lineMapper">
    <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
      <property name="lineTokenizer">
        <bean class="org.sfw.batch.item.file.transform.DelimitedLineTokenizer">
          <property name="names">  <list>
              <value>creditCardNumber</value>
              <value>merchantNumber</value>
              <value>amount</value>
              <value>date</value>
            </list>  </property>
        </bean>
      </property>
      <property name="fieldSetMapper">
        <bean class="rewards.batch.DiningFieldSetMapper"/>
      </property>
    </bean>
  </property>
</bean>
```

File to read

default impl

For use with CSV-like files

for use in FieldSet

© Copyright 2014 Pivotal. All rights reserved.

FieldSetMappers in Spring Batch

- Spring Batch provides mappers for common use cases:

- PassthroughFieldSetMapper
  - just returns the FieldSet as-is
  - simple use cases with no domain model, e.g. copy straight from file to database

- BeanWrapperFieldSetMapper
  - uses field names to bind to a prototype object
  - Binds using Spring TypeConverter (a.k.a. PropertyEditor) support

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- ExecutionContext
- Stateful ItemReaders/Writers
- Reading Flat Files
- **Sharing State Between Steps**
- Intro to Skip, Retry, Repeat, Restart
- Other Listeners
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

Sharing State Between Steps

- Data in step execution context can be “promoted” to job execution context
- Downstream steps can then retrieve those data
- Prevents state-storing code from having to interact with Job ExecutionContext directly
- ExecutionContextPromotionListener does this for specified key names

© Copyright 2014 Pivotal. All rights reserved.

Sharing State Between Steps

```
<job id="job1">
  <step id="step1" next="step2">
    <tasklet>
      <chunk reader="reader" writer="savingWriter" commit-interval="10"/>
      <listeners>
        <listener ref="promotionListener"/>
      </listeners>
    </tasklet>
  </step>
  <step id="step2">
    …
  </step>
</job>

<beans:bean id="promotionListener"
    class="org.spr....ExecutionContextPromotionListener">
  <beans:property name="keys" value="someKey"/>
</beans:bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Sharing State Between Steps

```
// Used in step 2:
public class RetrievingItemWriter implements ItemWriter<Object> {
    private Object someObject;
    public void write(List<? extends Object> items) throws Exception {
        // ...
    }

    @BeforeStep
    public void retrieveInterstepData(StepExecution stepExecution) {
        JobExecution jobExecution = stepExecution.getJobExecution();
        ExecutionContext jobContext = jobExecution.getExecutionContext();
        this.someObject = jobContext.get("someKey");
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Step Scope

- Spring Batch defines custom 'step' scope
- Useful for just-in-time initialization
  - To pass parameters from JobParameters or Job's ExecutionContext to step using SpEL expressions
  - jobParameters variable available automatically

```
<bean id="diningRequestsReader" scope="step"
            class="org.springframework.batch.item.file.FlatFileItemReader">
  <property name="resource"
                  value="#{jobParameters['input.resource.path']}"/>
  <property name="lineMapper">
    <bean class="org.sfw.batch.item.file.mapping.DefaultLineMapper">
      ...
    </bean>
  </property>
</bean>
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- ExecutionContext
- Stateful ItemReaders/Writers
- Reading Flat Files
- Sharing State Between Steps
- **Intro to Skip, Retry, Repeat, Restart**
- Other Listeners
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

Common Batch Idioms

- Batch jobs typically process large amounts of homogeneous input
- Makes iteration a common concern: **Repeat**
- Transient errors during processing may require a **Retry** of an input item
- Some input may not be valid, may want to **Skip** it without failing
- Some errors should fail the job execution, allowing to fix the problem and **Restart** the job instance where it left off

© Copyright 2014 Pivotal. All rights reserved.

Spring Batch Support

- Spring Batch supports these common concerns
- Abstracts them in the framework
  - Job business logic doesn't need to care about details
- Allows for simple configuration with pluggable strategies

© Copyright 2014 Pivotal. All rights reserved.

Repeat

- Most batch steps read items to process from some input source
- Framework can perform iteration over the input
  - Done using RepeatTemplate with RepeatCallback
  - Chunked step invokes ItemReader.read() in callback impl
  - Means no direct interaction with template is needed
- When there's no more input the step is complete
  - For chunked steps, ItemReader.read() simply returns null

© Copyright 2014 Pivotal. All rights reserved.

Retry

- A processing error could be *transient*
  - intermittent failure caused by external circumstances
- Simply retrying the operation might succeed
  - No need to fail the entire execution immediately
- Can mark certain exceptions as retryable
  - With max. # of retries per <u>item</u>

© Copyright 2014 Pivotal. All rights reserved.

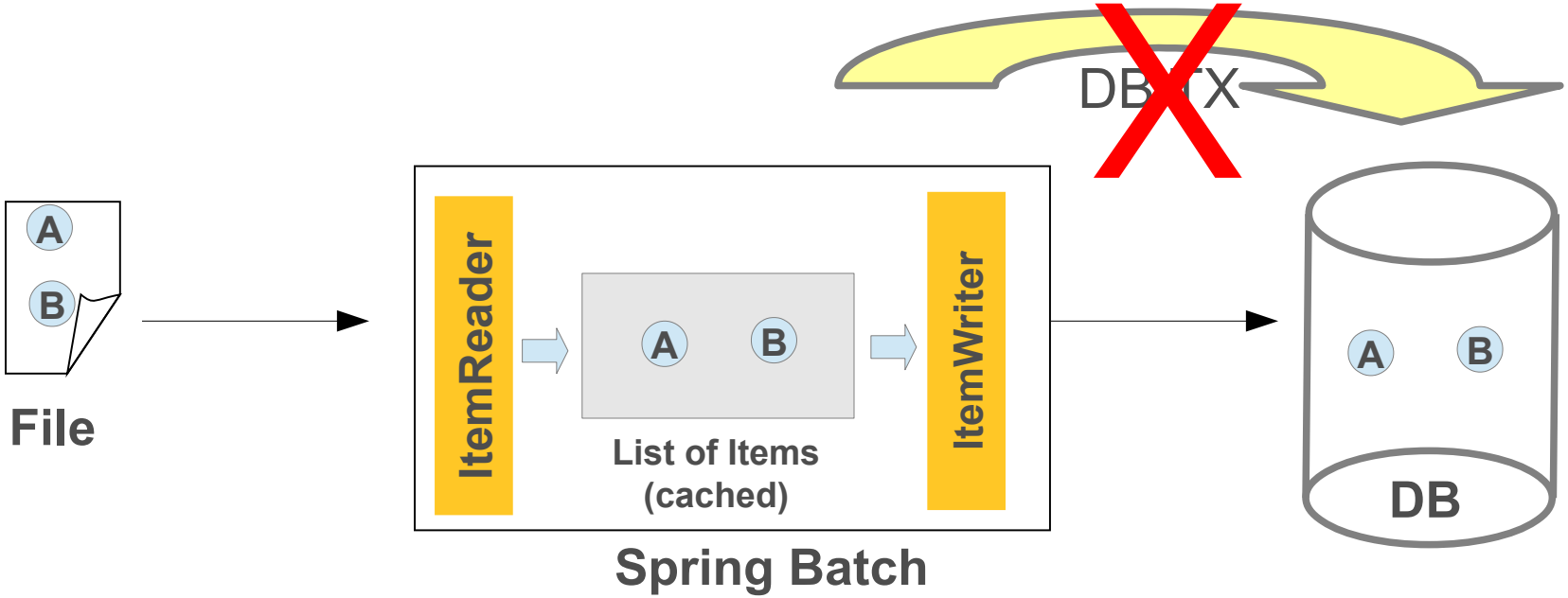
Retry Configuration

```
<step id="step1">
  <tasklet>
    <chunk reader="itemReader" writer="itemWriter" commit-interval="20"
              retry-limit="3">
      <retryable-exception-classes>
        <include class="org.sfw.dao.DeadlockLoserDataAccessException"/>
      </retryable-exception-classes>
    </chunk>
  </tasklet>
</step>
```

- Can also use excludes
  - For example, to include java.lang.Exception and then exclude specific exceptions that are not retryable

© Copyright 2014 Pivotal. All rights reserved.

Transactional ItemReaders

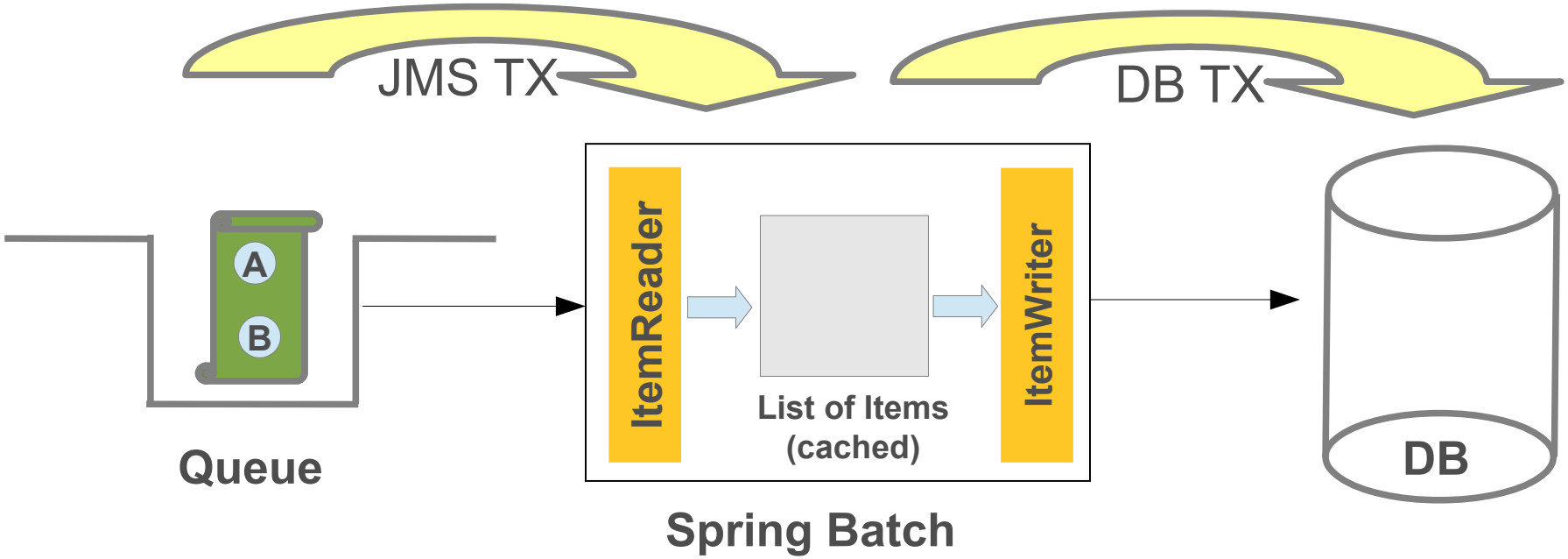


1. Reads from file
2. Receive message
3. DB TX starts
4. DB rollbacks
5. Message cached
6. Retries to commit

Message in cache, not required to read from file again

© Copyright 2014 Pivotal. All rights reserved.

Transactional ItemReaders: JMS–DB Best Case Scenario

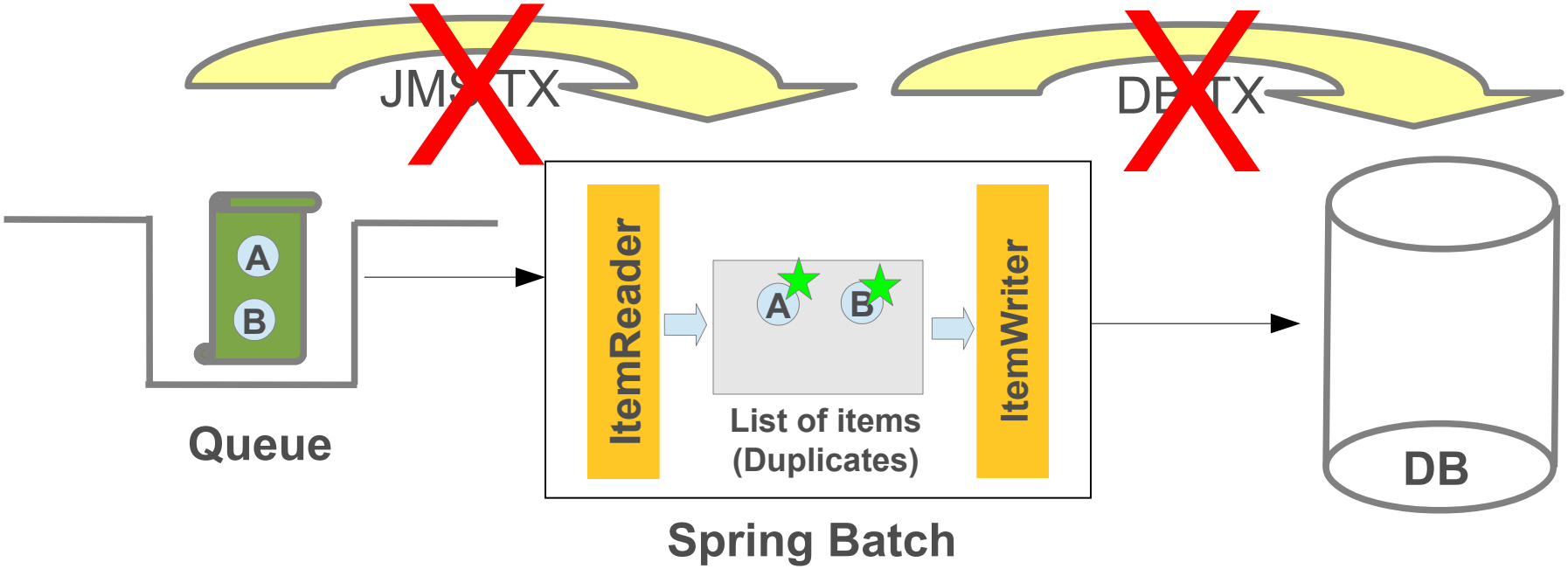


1. JMS TX starts
2. Receive message
3. DB TX starts
4. Store message data
5. DB TX commits
6. JMS TX commits

Message in DB, no longer in Queue

© Copyright 2014 Pivotal. All rights reserved.

Transactional ItemReaders: JMS-DB Rollback

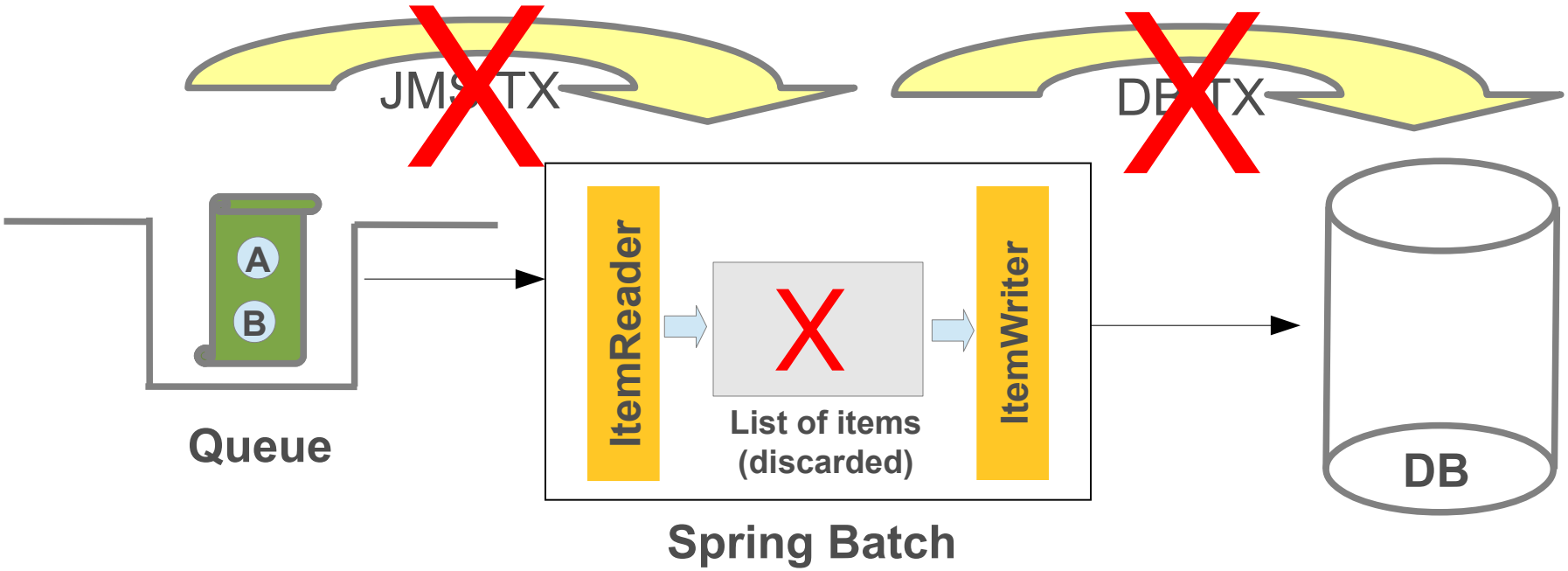


1. JMS TX starts
2. Receive message
3. DB TX starts
4. DB TX rollbacks
5. Items in cache
6. JMS TX rollbacks

Message in queue as well as in cache, resulting in duplicates

© Copyright 2014 Pivotal. All rights reserved.

Transactional ItemReaders: JMS-DB Rollback



1. JMS TX starts
2. Receive message
3. DB TX starts
4. DB TX rollbacks
5. Items in cache discarded
6. JMS TX rollbacks

Message in queue, ItemReader is invoked again

© Copyright 2014 Pivotal. All rights reserved.

Transactional ItemReaders and -Processors

- Items read and processed are cached by default
  - Prevents losing data in case of error
- Not needed for transactional reading/processing
  - e.g. when reading from JMS Queue
  - Rollback will restore original state automatically
- Configure this on the chunk

```
<step id="step1">
  <tasklet>
    <chunk reader="itemReader" writer="itemWriter" commit-interval="10"
            reader-transactional-queue="true" processor-transactional="true">
    </chunk>
  </tasklet>
</step>
```


© Copyright 2014 Pivotal. All rights reserved.

Skip

- Not all processing errors should cause a failure
  - Input often contains small percentage of invalid items
  - Just log the error and continue
- Can mark certain exceptions as skippable
  - With max. # of items to skip per step execution

© Copyright 2014 Pivotal. All rights reserved.

Skip Configuration

```
<step id="step1">
  <tasklet>
    <chunk reader="flatFileItemReader" writer="itemWriter"
              commit-interval="10" skip-limit="10">
      <skippable-exception-classes>
        <include class="org.sfw.batch.item.file.FlatFileParseException"/>
      </skippable-exception-classes>
    </chunk>
  </tasklet>
</step>
```

- Can also use excludes, like with retry
  - For example, to include java.lang.Exception and then exclude specific exceptions that should cause failure

© Copyright 2014 Pivotal. All rights reserved.

Restart

- Jobs defined via batch namespace considered restartable after failure by default
  - Disable with restartable="false"
- Requires execution context state to be persisted
  - For both job & step executions
  - Or executions would always start from the beginning
  - Various timestamps, status, etc. also persisted
  - See section on stateful readers/writers earlier
- Spring Batch checks for existing job execution
  - If found, it's a restart of an existing job instance
  - If not, it's a regular first execution

© Copyright 2014 Pivotal. All rights reserved.

Restarting a Job Instance

- Step execution context persisted at commit
  - incl. # of items read, written, skipped
  - incl. # of TXs committed and rolled back
- On restart ItemReader/-Writer can access that state
- Uses it do determine if and where to resume
  - Built in for many supplied ItemReaders and -Writers
  - Implement ItemStream for your own implementations

© Copyright 2014 Pivotal. All rights reserved.

Restart Configuration

- Allow completed steps to run again on restart
  - not allowed by default
- Limit the number of step restarts allowed
  - default is unlimited

```
<step id="gameLoad" next="playerSummarization">
   <tasklet allow-start-if-complete="true">
      <chunk reader="gameFileItemReader" writer="gameWriter"
         commit-interval="10" />
   </tasklet>
</step>
<step id="playerSummarization">
   <tasklet start-limit="3">
      <chunk reader="playerSummarizationSource"
               writer="summaryWriter" commit-interval="10" />
   </tasklet>
</step>
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- ExecutionContext
- Stateful ItemReaders/Writers
- Reading Flat Files
- Sharing State Between Steps
- Intro to Skip, Retry, Repeat, Restart
- **Other Listeners**
- Business Logic Delegation

© Copyright 2014 Pivotal. All rights reserved.

Listeners

- Register to get callbacks during execution
  - For logging/auditing, state & error handling, ...
  - JobExecutionListener
  - StepListener (marker interface)
    - StepExecution-, Chunk-, Item(Read|Process|Write)- & SkipListener
  - Annotation-based approach supported as alternative
- Implemented by ItemReader/-Processor/-Writer or in dedicated class
  - Explicit registration often not needed without dedicated class

© Copyright 2014 Pivotal. All rights reserved.

SkipListener

```
public interface SkipListener<T,S> extends StepListener {

    void onSkipInRead(Throwable t);

    void onSkipInProcess(T item, Throwable t);

    void onSkipInWrite(S item, Throwable t);
}
```


© Copyright 2014 Pivotal. All rights reserved.

SkipListener

- Skipped item is available unless skip occurred on read
- Appropriate skip method only called once per item
  - depending on when the error occurred
- Methods are NOT called when actual error occurred – called immediately before tx commit, after all chunk processing complete
- @OnSkipInRead, @OnSkipInWrite, @OnSkipInProcess annotations also available

Note: The item responsible for exceeding the skip limit is not received by the SkipListener

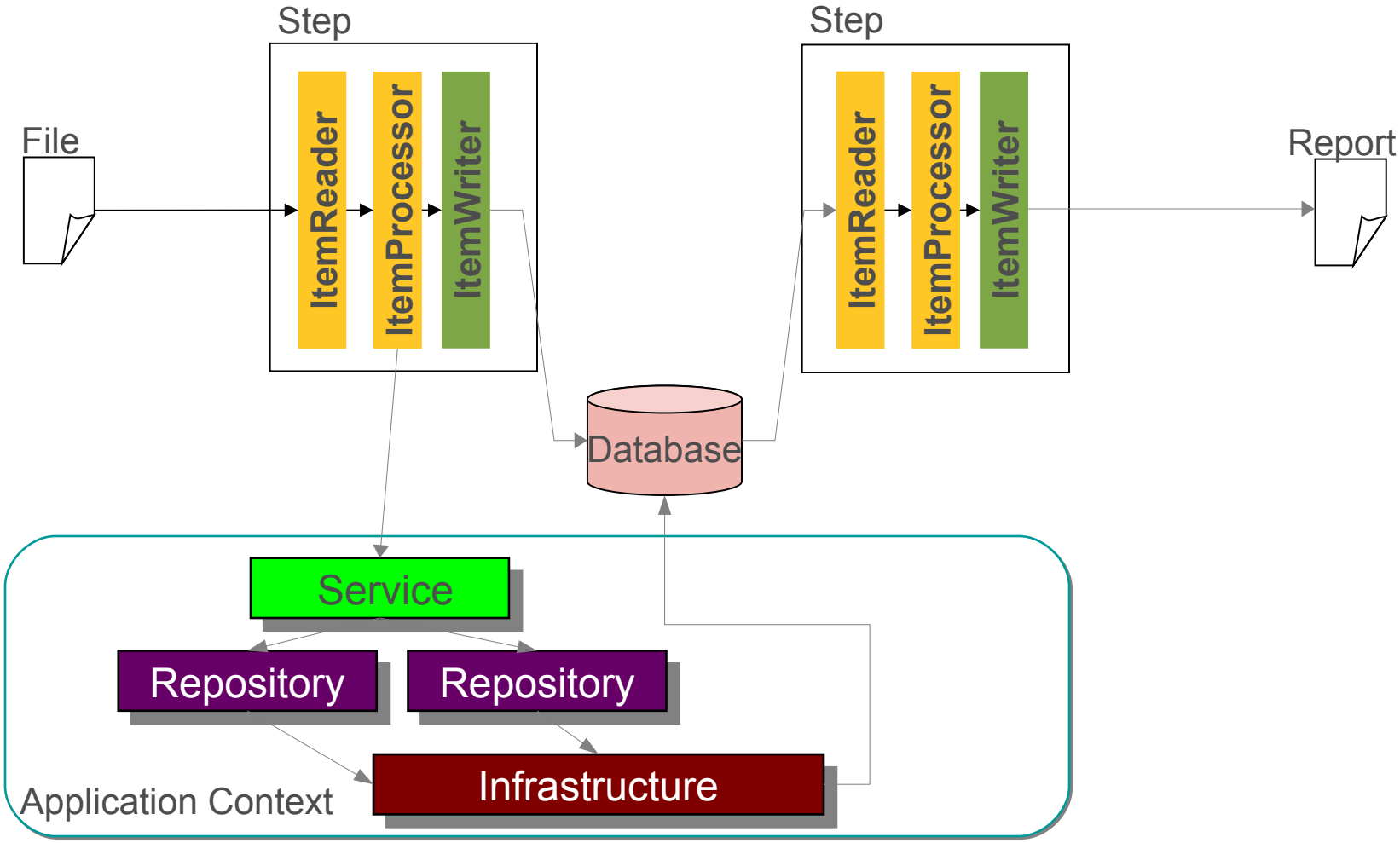
© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- ExecutionContext
- Stateful ItemReaders/Writers
- Reading Flat Files
- Sharing State Between Steps
- Intro to Skip, Retry, Repeat, Restart
- Other Listeners
- **Business Logic Delegation**

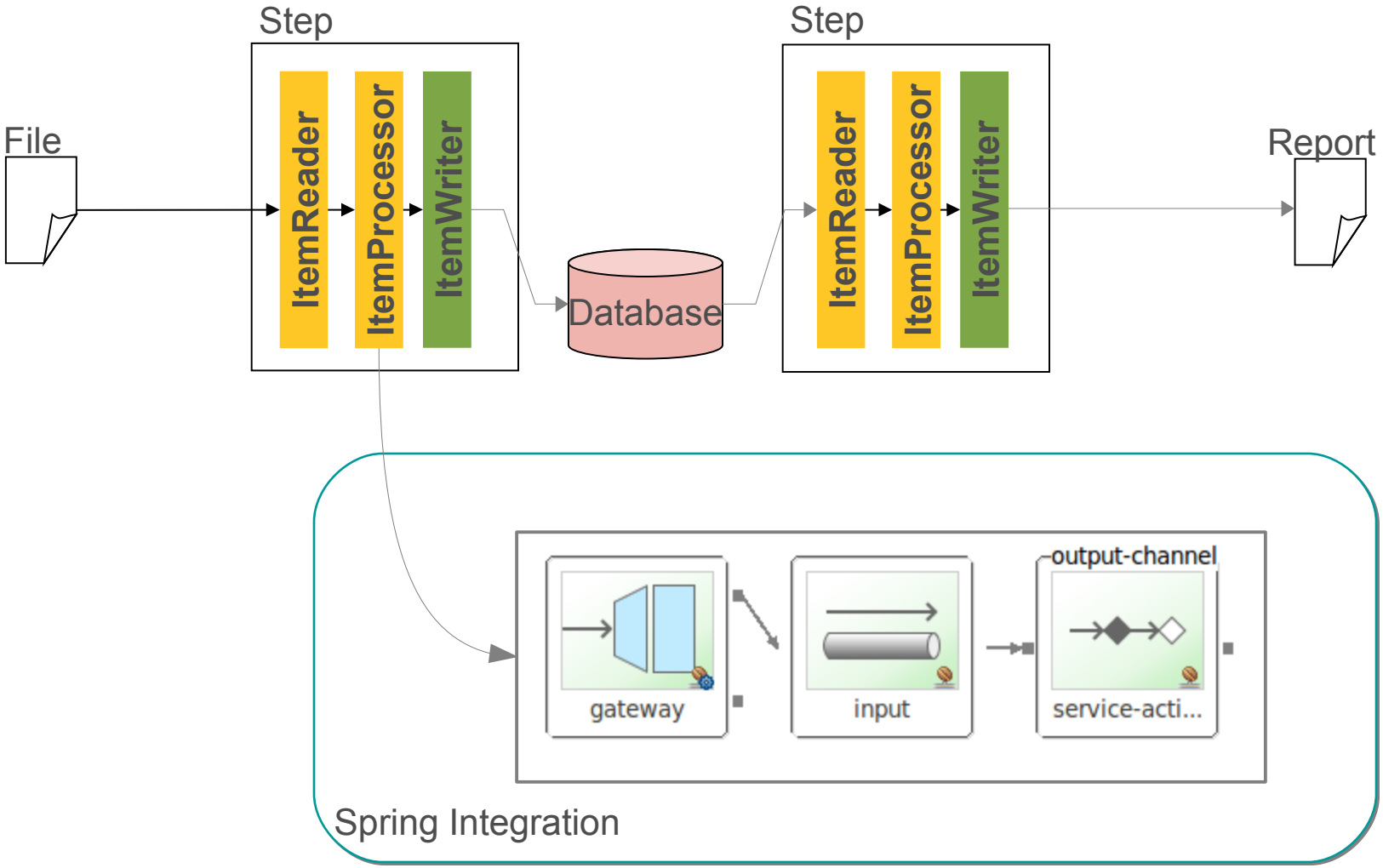
© Copyright 2014 Pivotal. All rights reserved.

Business Logic Delegation – Spring Application



© Copyright 2014 Pivotal. All rights reserved.

Business Logic Delegation – Spring Integration



© Copyright 2014 Pivotal. All rights reserved.

Lab

Restarting and Recovering a Failed Batch Job

© Copyright 2014 Pivotal. All rights reserved.

Spring Batch Admin and Parallel processing

A guide to Spring Batch Admin

and

Parallel processing in Batch jobs

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Batch Admin**
- Scaling and Parallel Processing

© Copyright 2014 Pivotal. All rights reserved.

Spring Batch Admin

- Sub project of Spring Batch
- Provides Web UI and ReSTFul interface to manage batch processes

  http://docs.spring.io/spring-batch-admin/

- Manager, Resources, Sample WAR
  - Deployed with batch job(s) as single app to be able to control & monitor jobs
  - Or monitors external jobs only via shared database

© Copyright 2014 Pivotal. All rights reserved.

Home Page

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/

Most Visited

Spring Batch Admin

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Resource	Method	Description
/configuration	GET	
/files	GET	
/files/**	GET	
/files/{path}	POST	
/home	GET	
/job-configuration-files	GET	
/job-configuration-multipart	GET	
/job-requests	GET	
/job-restarts	GET	
/jobs	GET	
/jobs/executions	GET	
/jobs/executions	DELETE	
/jobs/executions/{jobExecutionId}	GET	
/jobs/executions/{jobExecutionId}	DELETE	
/jobs/executions/{jobExecutionId}/steps	GET	
/jobs/executions/{jobExecutionId}/steps/{stepExecutionId}	GET	
/jobs/executions/{jobExecutionId}/steps/{stepExecutionId}/progress	GET	
/jobs/{jobName}	GET	
/jobs/{jobName}	POST	
/jobs/{jobName}/executions	GET	
/jobs/{jobName}/{jobInstanceId}/executions	GET	
/jobs/{jobName}/{jobInstanceId}/executions	POST	
/steps/step1	GET	

Done

© Copyright 2014 Pivotal. All rights reserved.

Registered Jobs

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/batch/job

Most Visited

Spring Batch Admin: Jobs

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Job Names Registered

Name	Description	Execution Count	Launchable	Incrementable
infinite	No description	0	true	true
job1	No description	0	true	true
job2	No description	0	true	false

Rows: 1-3 of 3 Page Size: 20

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Done

© Copyright 2014 Pivotal. All rights reserved.

Launch

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/batch/job

Google

Most Visited

Spring Batch Admin: Job Summ...

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Job name=job1: Launch

Job Parameters (key=value pairs): fail=false (Incrementable)

If the parameters are marked as "Not incrementable" then the launch button launches an instance of the job with the parameters shown. You can always add new parameters if you want to.

There are no job instances for this job.

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Done

S3Fox

© Copyright 2014 Pivotal. All rights reserved.

Launched

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/batch/job

Most Visited

Spring Batch Admin: Job Summ...

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Job name=job1: Launch

Job Parameters (key=value pairs): run.count(long)=0,fail=fal (Incrementable)

If the parameters are marked as "Not incrementable" then the launch button launches an instance of the job with the parameters shown. You can always add new parameters if you want to.

Job Instances for Job (job1)

ID		JobExecution Count	Last JobExecution	Parameters
0	executions	1	STARTED	{fail=false, run.count=0}

Rows: 1-1 of 1 Page Size: 20

The table above shows instances of this job with an indication of the status of the last execution. If you want to look at all executions for see here.

Done

© Copyright 2014 Pivotal. All rights reserved.

Completed

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/batch/j

Most Visited

Spring Batch Admin: Job Execut...

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Recent and Current Job Executions for Job = job1, instanceId = 0

Stop All

ID	Instance	Name	Date	Start	Duration	Status	ExitCode
0	0	job1	2010-09-01	14:05:58	00:00:00	COMPLETED	COMPLETED

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Done

S3Fox

© Copyright 2014 Pivotal. All rights reserved.

Job Execution Details

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Details for Job Execution

Stop

Property	Value
ID	0
Job Name	job1
Job Instance	0
Job Parameters	run.count(long)=0,fail=false
Start Date	2010-09-01
Start Time	14:05:58
Duration	00:00:00
Status	COMPLETED
Exit Code	COMPLETED
Step Executions Count	1
Step Executions	[j1.s1]

© Copyright 2014 Pivotal. All rights reserved.

Step Execution

File Edit View History Bookmarks Tools Help

http://localhost:8080/batch-2-solution/batch/jobs/exec

Most Visited

Spring Batch Admin: Step Exec...

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Step Executions for Job = job1, JobExecution = 0

ID		Job Execution	Name	Date	Start	Duration	Status	Reads	Writes	Skips	ExitCode
0	detail	0	j1.s1	2010-09-01	18:05:58	00:00:00	COMPLETED	5	5	0	COMPLETED

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Done

© Copyright 2014 Pivotal. All rights reserved.

Step Execution Details

Spring Batch Admin

Home | Jobs | Executions | Files

Step Execution Progress

This execution is estimated to be 100% complete after 164 ms based on end time (alre

History of Step Execution for Step=j1.s1

Summary after total of 1 executions:

Property	Min	Max	Mean	Sigma
Duration per Read	32	32	32	0
Duration	164	164	164	0
Commits	6	6	6	0
Rollbacks	0	0	0	0
Reads	5	5	5	0
Writes	5	5	5	0
Filters	0	0	0	0
Read Skips	0	0	0	0
Write Skips	0	0	0	0
Process Skips	0	0	0	0

Details for Step Execution

Property	Value

Done

Details for Step Execution

Property	Value
ID	0
Job Execution	0
Job Name	job1
Step Name	j1.s1
Start Date	2010-09-01
Start Time	18:05:58
Duration	00:00:00
Status	COMPLETED
Reads	5
Writes	5
Filters	0
Read Skips	0
Write Skips	0
Process Skips	0
Commits	6
Rollbacks	0
Exit Code	COMPLETED
Exit Message	

Done

© Copyright 2014 Pivotal. All rights reserved.

Failures

Step Name	j1.s1
Start Date	2010-09-01
Start Time	18:19:16
Duration	00:00:00
Status	FAILED
Reads	1
Writes	0
Filters	0
Read Skips	0
Write Skips	0
Process Skips	0
Commits	0
Rollbacks	1
Exit Code	FAILED
Exit Message	java.lang.RuntimeException: Planned failure at org.springframework.batch.admin.sample.ExampleItemWriter.write(ExampleItemWriter.java:42) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at

Done

© Copyright 2014 Pivotal. All rights reserved.

Restart or Abandon

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Details for Job Execution

Abandon

Restart

Property	Value
ID	1
Job Name	job1
Job Instance	1
Job Parameters	run.count(long)=1,fail=true
Start Date	2010-09-01
Start Time	14:19:16
Duration	00:00:00
Status	FAILED
Exit Code	FAILED

Done

S3Fox

© Copyright 2014 Pivotal. All rights reserved.

Stop a Running Job

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Details for Job Execution

Stop

Property	Value
ID	2
Job Name	job1
Job Instance	2
Job Parameters	run.count(long)=2,fail=false
Start Date	2010-09-01
Start Time	14:33:24
Duration	00:00:19
Status	STARTED
Exit Code	UNKNOWN
Step Executions Count	1
Step Executions	[j1.s1]

© Copyright 2014 Pivotal. All rights reserved.

Stopped

Spring Batch Admin

Home | Jobs | Executions | Files

Job name=job1: Launch

Job Parameters (key=value pairs): run.count(long)=2,fail=fal

If the parameters are marked as "Not incrementable" the
shown. You can always add new parameters if you want

Job Instances for Job (job1)

ID		JobExecution Count	Last JobExe
2	executions	1	STOPPED
1	executions	1	FAILED
0	executions	1	COMPLETED

Rows: 1-3 of 3 Page Size: 20

The table above shows instances of this job with an indi
see here.

Done

Details for Step Execution

Property	Value
ID	2
Job Execution	2
Job Name	job1
Step Name	j1.s1
Start Date	2010-09-01
Start Time	18:33:24
Duration	00:01:35
Status	STOPPED
Reads	1
Writes	1
Filters	0
Read Skips	0
Write Skips	0
Process Skips	0
Commits	1
Rollbacks	0
Exit Code	STOPPED
Exit Message	org.springframework.batch.core.JobInterruptedException

Done

© Copyright 2014 Pivotal. All rights reserved.

Restart or Abandon

Spring Batch Admin

spring source

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Details for Job Execution

Abandon

Restart

Property	Value
ID	2
Job Name	job1
Job Instance	2
Job Parameters	run.count(long)=2,fail=false
Start Date	2010-09-01
Start Time	14:33:24
Duration	00:01:35
Status	STOPPED
Exit Code	STOPPED
Step Executions Count	1
Step Executions	[j1.s1]

Done

S3Fox

© Copyright 2014 Pivotal. All rights reserved.

Restarting a Stopped/Failed Job

Spring Batch Admin

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Restart is a new *Execution* of the same *Instance*

Recent and Current Job Executions

Stop All

ID	Instance	Name	Date	Start	Duration	Status	ExitCode
3	2	job1	2010-09-01	14:39:00	00:00:13	COMPLETED	COMPLETED
2	2	job1	2010-09-01	14:33:24	00:01:35	STOPPED	STOPPED
1	1	job1	2010-09-01	14:32:36	00:00:00	FAILED	FAILED
0	0	job1	2010-09-01	14:32:19	00:00:00	COMPLETED	COMPLETED

Rows: 1-4 of 4 Page Size: 20

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Done

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Batch Admin
- **Scaling and Parallel Processing**

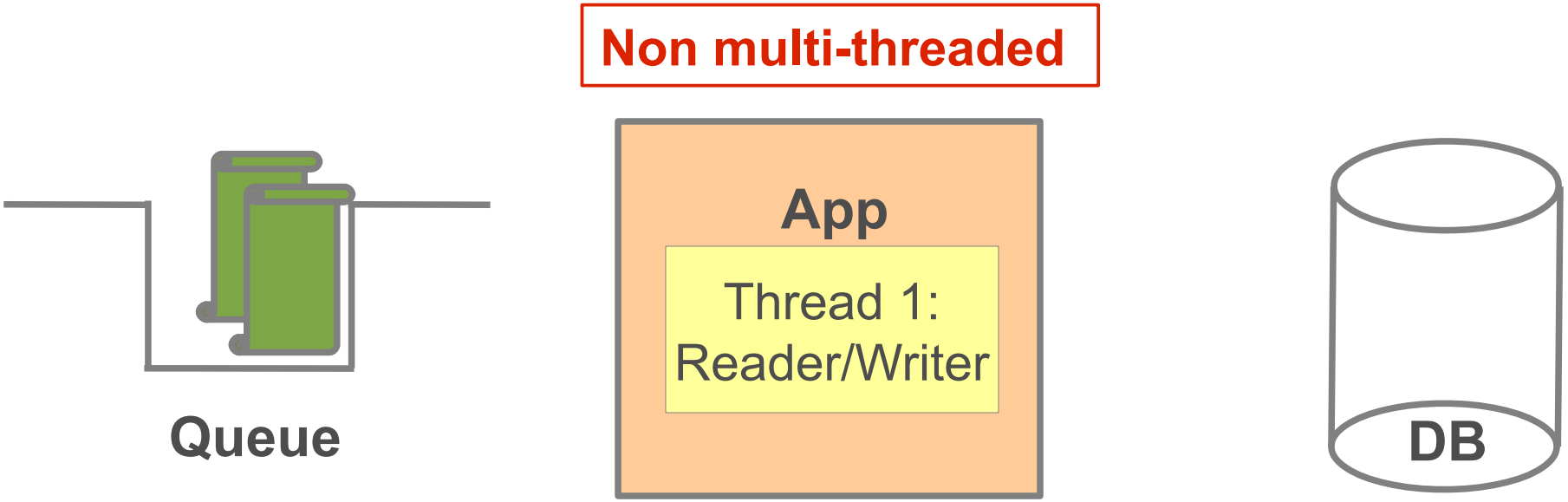
© Copyright 2014 Pivotal. All rights reserved.

Scaling and Parallel Processing

- First Rule:
  - Use the simplest technique to get the job done in the required time
  - Do not optimize/parallelize unnecessarily
- Different options available:
  - Where is the bottleneck? Reader, Writer, both?
  - What type of Input/Output? JMS, DB, File?

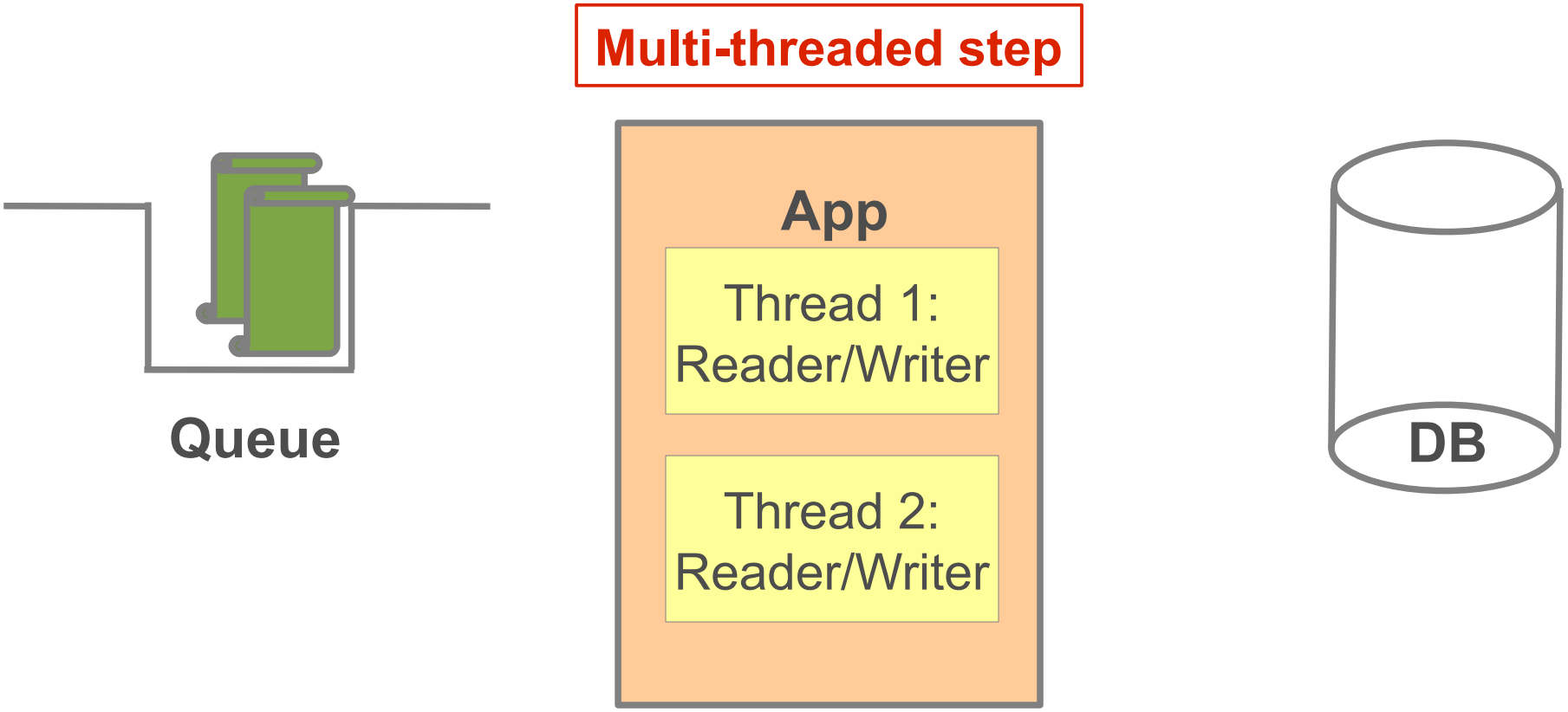
© Copyright 2014 Pivotal. All rights reserved.

Use case: JMS Input



© Copyright 2014 Pivotal. All rights reserved.

Use case: JMS Input



© Copyright 2014 Pivotal. All rights reserved.

Multi-threaded Step

- Just add a TaskExecutor to the tasklet

```
<step id="loading">
   <tasklet task-executor="taskExecutor" throttle-limit=”20” … />
</step>
```

- Number of threads limited by task executor and throttle-limit (which defaults to 4)
  - Throughput may be further limited by other components; e.g. pooled DataSources

© Copyright 2014 Pivotal. All rights reserved.

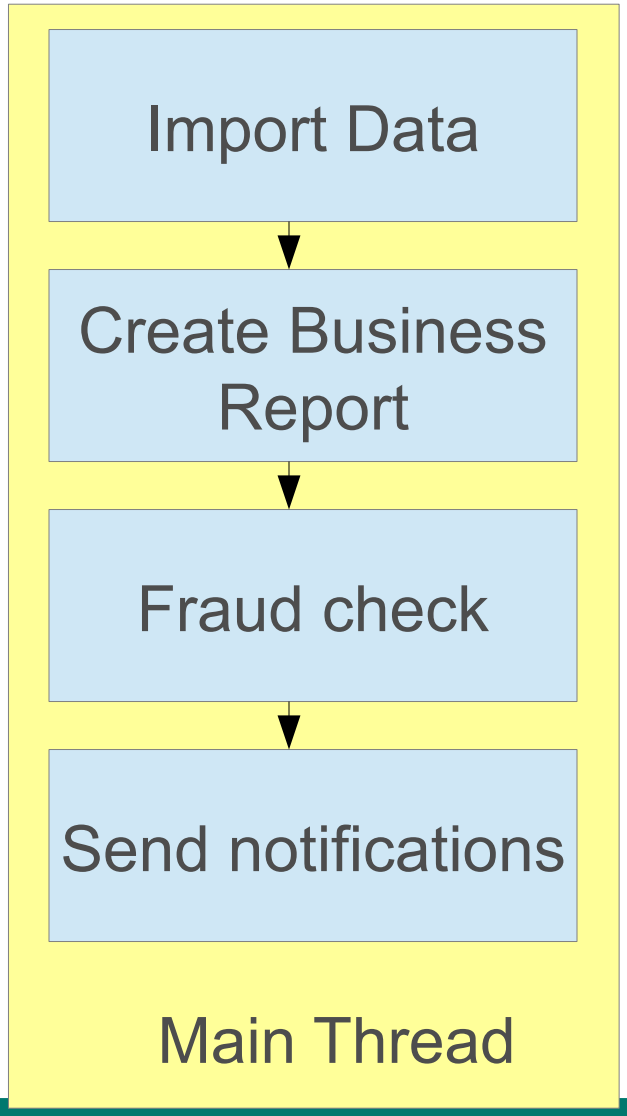
Multi-threaded Step

- When to use:
  - Source can handle the load balancing (like JMS, AMQP)
  - Own stateful readers taking care of partitioning the data
  - To speed up the step execution

- When to use NOT:
  - With off-the-shelf readers for DB, File, etc.
    - Every Thread would read the same data
  - If the reader/write is not thread safe.
    - Not all off-the-shelf readers/writers are thread safe, check the JavaDocs

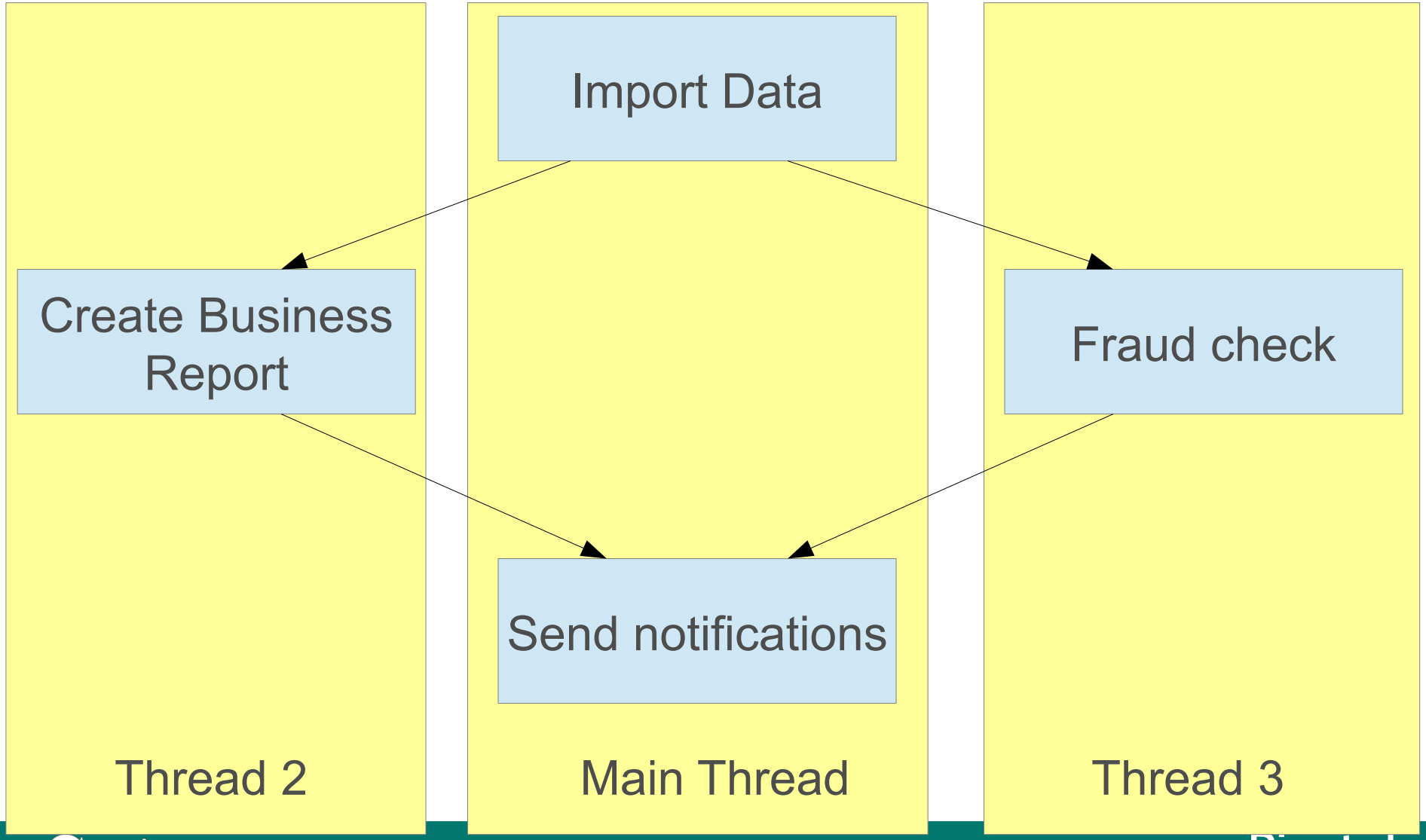
© Copyright 2014 Pivotal. All rights reserved.

Use case: Independent Steps



© Copyright 2014 Pivotal. All rights reserved.

Use case: Independent Steps



© Copyright 2014 Pivotal. All rights reserved.

Parallel Steps

```
<job id="job1">
    <step id="importData" />
    <split id="split1" task-executor="taskExecutor" next="sendNotifications">
        <flow>
            <step id="createBusinessReport" />
        </flow>
        <flow>
            <step id="fraudCheck" />
        </flow>
    </split>
    <step id="sendNotifications" />
</job>
```


© Copyright 2014 Pivotal. All rights reserved.

Parallel Steps

- When to use:
  - For independent steps.
  - To speed up the whole JOB execution.

- When to use NOT:
  - For dependent steps.
  - To speed up the STEP execution itself.

© Copyright 2014 Pivotal. All rights reserved.

Use case: DB Input

ID	NAME
1	Noelle
2	Paul
3	Mike
4	Michael
5	Mark
6	Ken
7	Kevin
8	Wes
9	Bijoy
10	Clarence

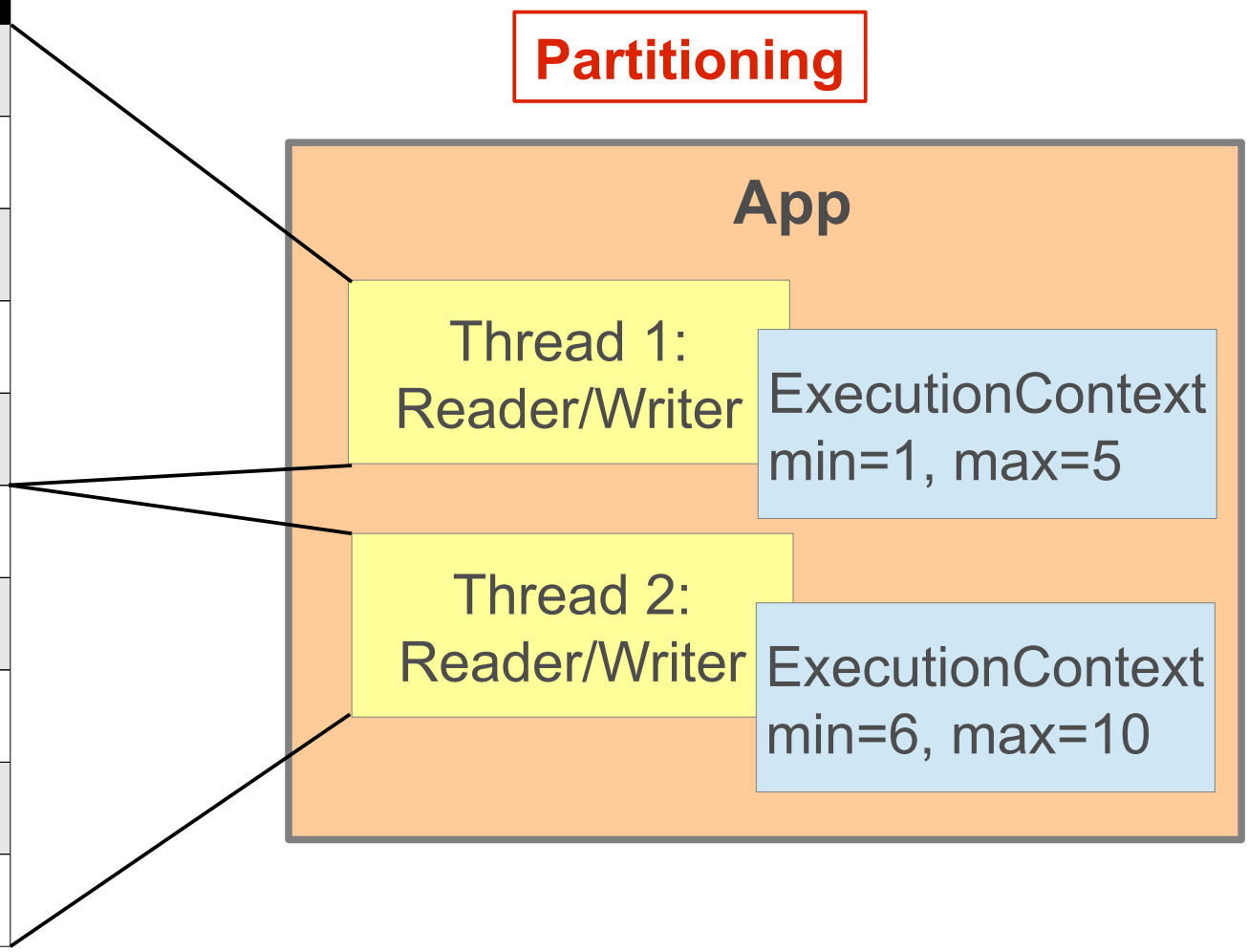
No optimization

Cannot use multi-threaded step,
as both steps would access same data.

© Copyright 2014 Pivotal. All rights reserved.

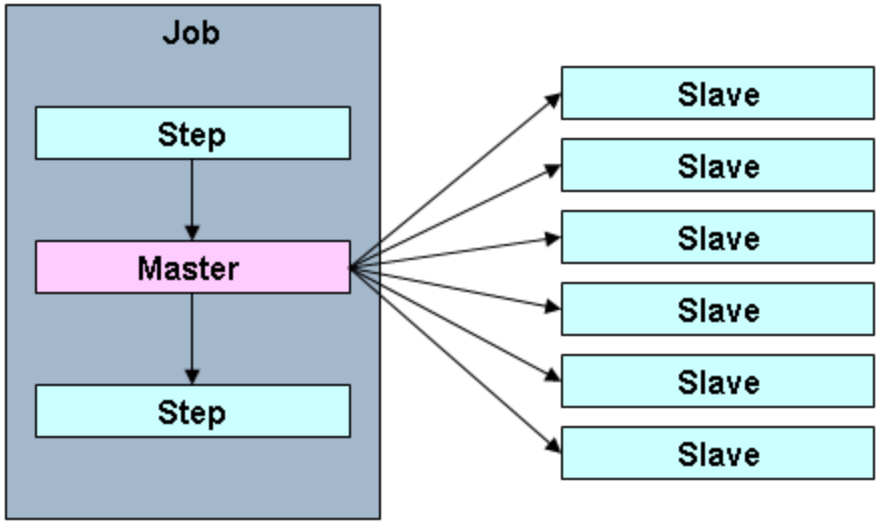
Use case: DB Input

ID	NAME
1	Noelle
2	Paul
3	Mike
4	Michael
5	Mark
6	Ken
7	Kevin
8	Wes
9	Bijoy
10	Clarence



© Copyright 2014 Pivotal. All rights reserved.

Partitioning



- Similar to multi-threaded step, but with a “Partitioner” who splits up the input data.
- Master step is just the container.
- Slaves are all the same steps, but with different Execution Contexts, running in different threads.

© Copyright 2014 Pivotal. All rights reserved.

Partitioning

```
<job id="job1">
   <step id="master">
      <partition step="processData" partitioner="partitioner">
         <handler grid-size="2" task-executor="taskExecutor" />
         <!-- Creates processData.partition0 and processData.partition1 -->
      </partition>
   </step>
</job>
<step id="processData">
   <tasklet>
      <chunk reader="jdbcReader" writer="..."/>
   </tasklet>
</step>
```

- See batch-admin-solution for a complete example.

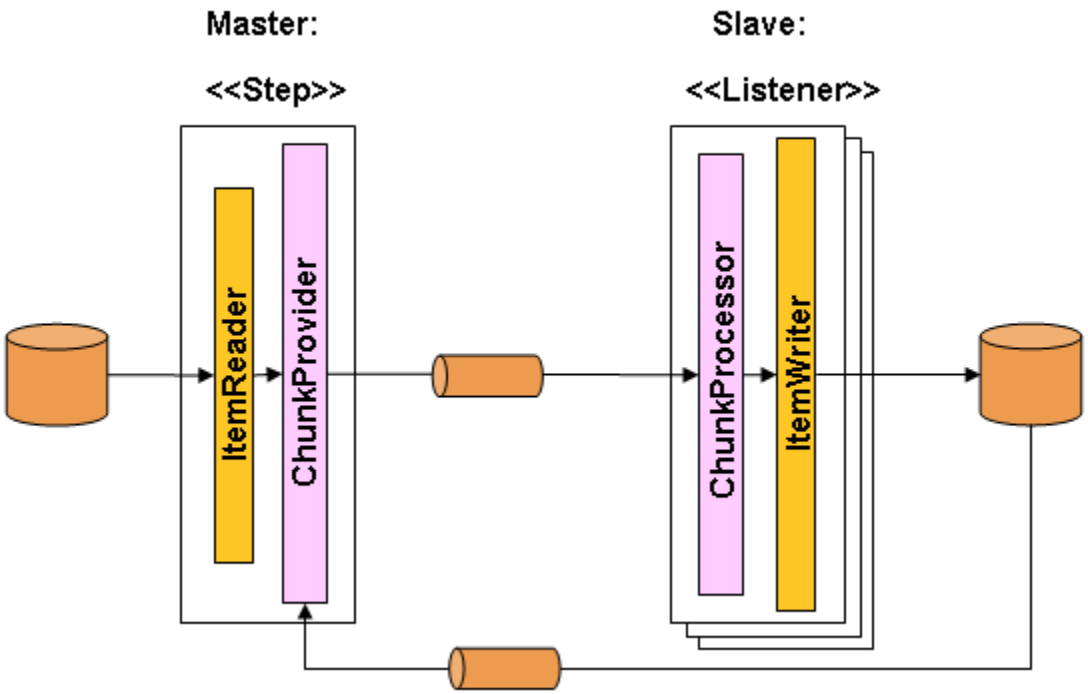
© Copyright 2014 Pivotal. All rights reserved.

Partitioning

- When to use:
  - For non-messaging sources, like DBs
  - To speed up the step execution
- When to use NOT:
  - For single threaded sources, like a File
  - For data, which cannot be partitioned (very rarely)

© Copyright 2014 Pivotal. All rights reserved.

Remote Chunking



- Reader reads chunks; they are sent for remote processing
- Requires durable middleware (e.g. JMS) for communication between master and slaves

© Copyright 2014 Pivotal. All rights reserved.

Remote Chunking

- When to use:
  - To speed up processing and/or writing
  - Can use off-the-shelf readers, writers and processors
  - Slaves are multiple processes

- When to use NOT:
  - To speed up reading (still single threaded)

© Copyright 2014 Pivotal. All rights reserved.

Lab

Using Spring Batch Admin to Manage Jobs

© Copyright 2014 Pivotal. All rights reserved.

Finishing Up

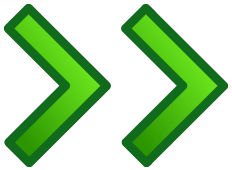
Course Completed

What's Next?

© Copyright 2014 Pivotal. All rights reserved.

What's Next

- Congratulations, we've finished the course
- What to do next?
  - Certification
  - Other courses
  - Resources
  - Evaluation



- Check-out optional sections on XML, SOAP WS Security, REST and AMQP

© Copyright 2014 Pivotal. All rights reserved.

Certification

Pivotal CERTIFIED
Spring Enterprise Integration Specialist

- Computer-based exam
  - 50 multiple-choice questions
  - 90 minutes
  - Passing score: 76% (38 questions answered successfully)
- Preparation
  - Review all the slides
  - Redo the labs

© Copyright 2014 Pivotal. All rights reserved.

Certification: Questions

Pivotal CERTIFIED
Spring Enterprise Integration Specialist

Typical question:

- Which statement best describes REST features in Spring? (select one)
  - Spring MVC provides its own JAX-RS implementation
  - Spring MVC's REST features may rely on any JAX-RS implementation
  - Spring MVC does not provide REST features
  - Spring MVC's REST features do not rely on the JAX-RS standard

© Copyright 2014 Pivotal. All rights reserved.

Certification: Logistics

- Where?
  - At any Pearson VUE Test Center
  - Most large or medium-sized cities
    - See http://www.pearsonvue.com/vtclocator
- How?
  - At the end of the class, you will receive a certification voucher by email
  - Make an appointment
  - Give them the voucher when you take the test
- For any further inquiry, you can write to
  - education@pivotal.io

© Copyright 2014 Pivotal. All rights reserved.

Other courses

- Many courses available
  - Processing Big Data with Hadoop and Pivotal HD
  - What's New in Spring
  - Spring Web
  - JPA with Spring
  - Core Spring
  - Groovy and Grails
  - Cloud Foundry
  - Gemfire ...
- See http://www.pivotal.io/training

© Copyright 2014 Pivotal. All rights reserved.

What's New in Spring

- Three day course covering new features of Spring 3.x and 4.x
  - Annotation and Java based configuration
  - Servlet 3.x
  - Spring Security
  - Spring Data
  - Spring 4 and Java 8
  - Spring Boot

© Copyright 2014 Pivotal. All rights reserved.

Processing Big Data with Hadoop and Pivotal HD

- Four day course
  - Big/Fast data, NoSQL, and their role in modern Business applications
  - Background on Hadoop and the significance of the Pivotal HD distribution
  - Introduction to GemFire and SQLFire, distributed cache technologies

© Copyright 2014 Pivotal. All rights reserved.

Spring Web

Pivotal CERTIFIED
Spring Web Application Developer

- Four-day workshop
- Making the most of Spring in the web layer
  - Spring MVC
  - Spring Web Flow
  - REST using MVC and AJAX
  - Security of Web applications
  - Mock MVC testing framework
- Spring Web Application Developer certification

© Copyright 2014 Pivotal. All rights reserved.

JPA with Spring

- Three day course covering
  - Using JPA with Spring and Spring Transactions
  - Implement inheritance and relationships with JPA
  - Discover how JPA manages objects
  - Go more in depth on locking with JPA
  - Advanced features such as interceptors, caching and batch updates

© Copyright 2014 Pivotal. All rights reserved.

Developing Applications with Cloud Foundry

- Three day course covering
  - Application deployment to Cloud Foundry
  - Cloud Foundry Concepts
  - Deployment using cf tool or an IDE
  - Accessing and defining Services
  - Using and customizing Buildpacks
  - Design considerations: “12 Factor”
  - JVM Application specifics, using Spring Cloud

© Copyright 2014 Pivotal. All rights reserved.

Pivotal Support Offerings

Pivotal Support

- Global organization provides 24x7 support
  - How to Register: http://tinyurl.com/piv-support

- Premium and Developer support offerings:
  - http://www.pivotal.io/support/offerings
  - http://www.pivotal.io/support/oss
  - Both Pivotal App Suite *and* Open Source products

- Support Portal: https://support.pivotal.io
  - Community forums, Knowledge Base, Product documents

© Copyright 2014 Pivotal. All rights reserved.

Pivotal Consulting

- Custom consulting engagement?
  - Contact us to arrange it
    - http://www.pivotal.io/contact/spring-support
    - Even if you don't have a support contract!

- Pivotal Labs
  - Agile development experts
  - Assist with design, development and product management
    - http://www.pivotal.io/agile
    - http://pivotallabs.com

© Copyright 2014 Pivotal. All rights reserved.

Resources

- The Spring reference documentation
  - http://spring.io/docs
  - Already 800+ pages!
- The official technical blog
  - http://spring.io/blog
- Stack Overflow – Active Spring Forums
  - http://stackoverflow.com

© Copyright 2014 Pivotal. All rights reserved.

Resources (2)

- You can register issues on our Jira repository
  - https://jira.spring.io
- The source code is available here
  - https://github.com/spring-projects/spring-framework

© Copyright 2014 Pivotal. All rights reserved.

Thank You!

We hope you enjoyed the course

Please fill out the evaluation form

http://tinyurl.com/mylearneval

© Copyright 2014 Pivotal. All rights reserved.

Advanced XML

Using XML for Integration Purposes

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Introduction**
- XML Namespaces
- XML Schema
- Java APIs
- Object-XML Mapping
- JAXB2
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

Introduction

- Integration by definition involves multiple systems and/or applications
- Often requires shared data format
  - For messages or RPC parameters
- Also important if systems themselves don't use the format directly
  - Think about transformations, content-based routing and filtering, validation, etc.

© Copyright 2014 Pivotal. All rights reserved.

Shared Data Format

- Integration requirements for data format:
  - Easy to author and to read
  - Open standard that's platform-independent
  - Extensible, with formal way to specify valid content
  - Well supported, including tool support
- XML fits those requirements nicely

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- **XML Namespaces**
- XML Schema
- Java APIs
- Object-XML Mapping
- JAXB2
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

Why Namespaces?

- Regular XML uses simple identifiers as element names:

```
<transfer>
    <credit>S123</credit>
    <debit>C456</debit>
    <amount>1205.15</amount>
</transfer>
```

- This only works for restricted domains
- It can lead to *naming conflicts*

© Copyright 2014 Pivotal. All rights reserved.

XML Naming Conflict (1)

- What if we have both in one document?

```
<transfer>
   <credit>S123</credit>
   <debit>C456</debit>
   <amount>1205.15</amount>
</transfer>
```

```
<transfer>
   <player name="Stephen Hughes" />
   <from team="Glasgow Rangers" />
   <to team="Leicester City" />
</transfer>
```


© Copyright 2014 Pivotal. All rights reserved.

XML Naming Conflict (2)

- <transfer> elements are not the same
- But we cannot easily distinguish
- Let alone mix-and match

- **We need an extra unique discriminator**
- URI = Uniform Resource Identifier:
  perfect for the job

© Copyright 2014 Pivotal. All rights reserved.

URIs

- URL-like strings whose precise syntax and semantics depend upon their scheme
- Either a *URL* or a *URN*
- UR*L* specifies a location, UR*N* just a name
- URL Location doesn't have to exist, just be unique
- Examples:
  - urn:issn:1535-3613
  - http://www.w3.org/2001/XMLSchema
  - http://www.springframework.org/schema/beans

© Copyright 2014 Pivotal. All rights reserved.

Namespaces

- Namespace name represented by URI
- Elements and attributes can be assigned to namespace
- Only elements/attrs with the same local name *and* the same namespace are the same
  - Combination is known as *qualified name*
- Allows unique identification
- Allows grouping of related elements/attrs

© Copyright 2014 Pivotal. All rights reserved.

Namespace Examples (1)

```
<transfer xmlns="http://mybank.com/schema/moneytransfer" >
   <credit>S123</credit>
   <debit>C456</debit>
   <amount>1205.15</amount>
</transfer>
```

Note: URI is just an identifier!

```
<transfer xmlns="http://soccer.com/schema/playertransfer" >
   <player name="Stephen Hughes" />
   <from team="Glasgow Rangers" />
   <to team="Leicester City" />
</transfer>
```


© Copyright 2014 Pivotal. All rights reserved.

Namespace Examples (2)

- Namespaces can be mixed:

```
<transfer xmlns="http://soccer.com/schema/playertransfer" >
   <player name="Stephen Hughes" />
   <from team="Glasgow Rangers" />
   <to team="Leicester City" />
   <transfer xmlns="http://mybank.com/schema/moneytransfer" >
      <credit>S123</credit>
      <debit>C456</debit>
      <amount>15000000</amount>
   </transfer>
</transfer>
```


© Copyright 2014 Pivotal. All rights reserved.

Namespace Prefixes

- Repeating the URI is not practical
- We can bind namespace URI to short prefix
- Syntax: xmlns:*namespace-prefix*="*namespaceURI*"

```
<mt:transfer xmlns:mt="http://mybank.com/schema/moneytransfer" >
   <mt:credit>S123</mt:credit>
   <mt:debit>C456</mt:debit>
   <mt:amount>1205.15</mt:amount>
</mt:transfer>
```

- Plain xmlns specifies a *default* namespace

© Copyright 2014 Pivotal. All rights reserved.

Binding Prefixes

- Typically all namespaces are bound in the root element

```
<pt:transfer xmlns:pt="http://soccer.com/schema/playertransfer"
             xmlns:mt="http://mybank.com/schema/moneytransfer">
  <pt:player name="Stephen Hughes" />
  <pt:from team="Glasgow Rangers" />
  <pt:to team="Leicester City" />
  <mt:transfer>
    <mt:credit>S123</mt:credit>
    <mt:debit>C456</mt:debit>
    <mt:amount>15000000</mt:amount>
  </mt:transfer>
</pt:transfer>
```


© Copyright 2014 Pivotal. All rights reserved.

Namespace Gotchas

- Prefix is meaningless, URI is what matters
- Still, some prefixes are very common:
  - xs for XML Schema
  - xsi for XML Schema instances
  - xsl for XSLT
  - fo for XSL-FO
- xml and xmlns prefix are reserved for XML
- Namespaces introduced after XML 1.0: older tooling may ignore namespaces
- DTDs are not namespace-aware

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- XML Namespaces
- **XML Schema**
- Java APIs
- Object-XML Mapping
- JAXB2
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

XML Schema introduction

- XML is derived from SGML
- SGML uses DTDs to define document structure
- DTD is quite limited
- So other schema languages were created:
  - W3C XML Schema
  - RELAX NG
  - Schematron
- XML Schema is most commonly used and supported

© Copyright 2014 Pivotal. All rights reserved.

XML Schema

- W3C specification
- Uses XML to define the schema
- More powerful than DTD
  - Better validation support
- Also more complex

© Copyright 2014 Pivotal. All rights reserved.

XML Schema

- Allows defining constraints on data
  - min-/maxOccurs on subelements
  - length, regular expression, etc. on text
- Fully namespace aware
- Allows annotating elements with meta-data
- Works with *types*

© Copyright 2014 Pivotal. All rights reserved.

Types

- Simple type
  - Attributes and elements that can only contain text
  - Several built-in types defined
- Complex type
  - Elements that can contain attributes or other elements
- Let’s look at an example

© Copyright 2014 Pivotal. All rights reserved.

XML Schema example (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.springsource.com/reward-network"
        xmlns:rewards="http://www.springsource.com/reward-network"
        elementFormDefault="qualified">

    <element name="rewardAccountForDiningResponse">
        <complexType>
            <sequence>
                <element name="rewardConfirmation"
                         type="rewards:RewardConfirmationType"/>
            </sequence>
        </complexType>
    </element>
…
```

default namespace

complex type with element sequence

custom data type

© Copyright 2014 Pivotal. All rights reserved.

XML Schema example (2)

```
…
    <complexType name="RewardConfirmationType">
        <sequence>
            <element name="distribution" type="rewards:DistributionType"
                    maxOccurs="unbounded"/>
        </sequence>
        <attribute name="confirmationNumber" type="string"/>
        <attribute name="accountNumber" type="string"/>
        <attribute name="amount" type="decimal"/>
    </complexType>

    <complexType name="DistributionType">
        <attribute name="beneficiary" type="string"/>
        <attribute name="amount" type="decimal"/>
        <attribute name="percentage" type="decimal"/>
        <attribute name="totalSavings" type="decimal"/>
    </complexType>
</schema>
```

constraint

built-in data types

complex type with attributes

© Copyright 2014 Pivotal. All rights reserved.

Referring to a schema

- Namespace URI is not a real location
- So how to get the corresponding schema?
- Typically through schemaLocation attribute:

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

   <!-- bean definitions go here -->
<beans>
```

© Copyright 2014 Pivotal. All rights reserved.

schemaLocation Attribute

- Couples namespace URI to real URL
- Just a hint to the parser
  - parser not obliged to fetch location URL
  - In previous sample, Spring will *not* try to read the XSDs from the given URLs!
- In Eclipse, both namespace URI and schemaLocation URL can be mapped to local resource
  - Search for XML Catalog under Preferences

© Copyright 2014 Pivotal. All rights reserved.

Creating an XML Schema

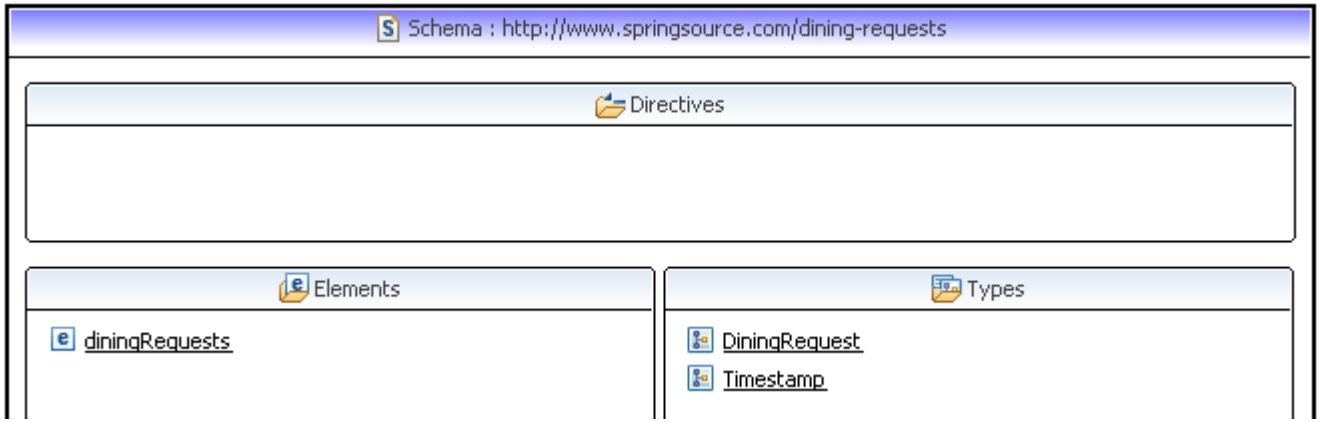
Common approaches:

- Write schema yourself
- Generate from existing XML examples
- Generate from non-XML source, like Java classes

© Copyright 2014 Pivotal. All rights reserved.

Writing a schema

- Writing an XML Schema definition (XSD) by hand is hard for large schemas
- Several authoring tools exist
  - XML Spy very popular example
- Eclipse's WTP also offers some support:



© Copyright 2014 Pivotal. All rights reserved.

Generate schema from XML samples

- Nice approach if samples have good coverage
- Typically needs some manual tweaking afterwards
- Several tools available
  - Trang
  - MS schema inference tool
  - XML Spy

© Copyright 2014 Pivotal. All rights reserved.

Generate schema from other source

- For automated conversion to and from xml
  - objects <-> XML, DB result sets <-> XML
- Convenient for *producing* side
- Might result in awkward schema
  - Esp. if source is non-hierarchical
  - So not always convenient for *consuming* side
  - True for code-centric solutions in general

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- XML Namespaces
- XML Schema
- **Java APIs**
- Object-XML Mapping
- JAXB2
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

Java APIs for XML

- Multiple libraries and APIs exist for XML
- Since Java 1.4, a standard API was included
  - *JAXP*: the Java API for XML Processing
- Approaches are tree- or stream-based
- Tree: work with the entire document contents
  - Through a DOM (Document Object Model)
  - org.w3c.dom (in JAXP), JDOM, DOM4J, XOM
- Stream: event-based procesing, more efficient
  - Push-based: SAX (Simple API for XML)
  - Pull-based: StAX (Streaming API for XML)

© Copyright 2014 Pivotal. All rights reserved.

JAXP Sample: parsing using DOM

```
import javax.xml.parsers.*;
…
// Step 1: create a DocumentBuilderFactory
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
// the following options default to ‘false’:
dbf.setNamespaceAware(true);
dbf.setValidating(true);
// Step 2: create a DocumentBuilder
DocumentBuilder db = dbf.newDocumentBuilder();
// Step 3: parse XML file to get a Document object
org.w3c.dom.Document doc = db.parse(new File(file));
```

- Dynamically chooses implementation to use
- Parses entire XML file into memory!

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- XML Namespaces
- XML Schema
- Java APIs
- **Object-XML Mapping**
- JAXB2
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

The Object/XML Mismatch

- When programming Java, we work with objects
- Many integration solutions use XML
- So unless we use XML directly, we need to convert between objects and XML
- At first sight, this is straightforward
- But in reality, this is not a 1-to-1 mapping

© Copyright 2014 Pivotal. All rights reserved.

The Object/XML Mismatch

- Object relations are graph-based
- XML is hierarchical
- A mismatch exists between objects and XML!
- Similar to Object/relational mismatch, but with different issues
- OXM Tools help to overcome the differences

© Copyright 2014 Pivotal. All rights reserved.

OXM Solutions

- Tools exist to help mapping XML to Java
- OXM: Object to XML mapping
- AKA XML Data Binding
- Commonly used in integration solutions
- Several popular implementations:
  - JAXB / JAXB2
  - JiBX
  - Castor
  - XMLBeans
  - XStream

© Copyright 2014 Pivotal. All rights reserved.

OXM Concepts

- Marshalling:
  - Create XML from Objects
- Unmarshalling:
  - Create Objects from XML
- A Mapping or Binding:
  - Description of how (un)marshalling should occur for a given schema or class
- Document-centric:
  - Focuses on XML-first approach, binding schema
- Object-centric:
  - Focuses on object-first approach, binding classes

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- XML Namespaces
- XML Schema
- Java APIs
- Object-XML Mapping
- **JAXB2**
- Spring-OXM

© Copyright 2014 Pivotal. All rights reserved.

JAXB / JAXB2

- Java API for XML Binding
- Specification with Sun-provided RI
- Binds a schema to classes (*document-centric*)
- XML can then be (un)marshalled to/from objects of these types
- JAXB2 can also generate schema from annotations (*object-centric*)

© Copyright 2014 Pivotal. All rights reserved.

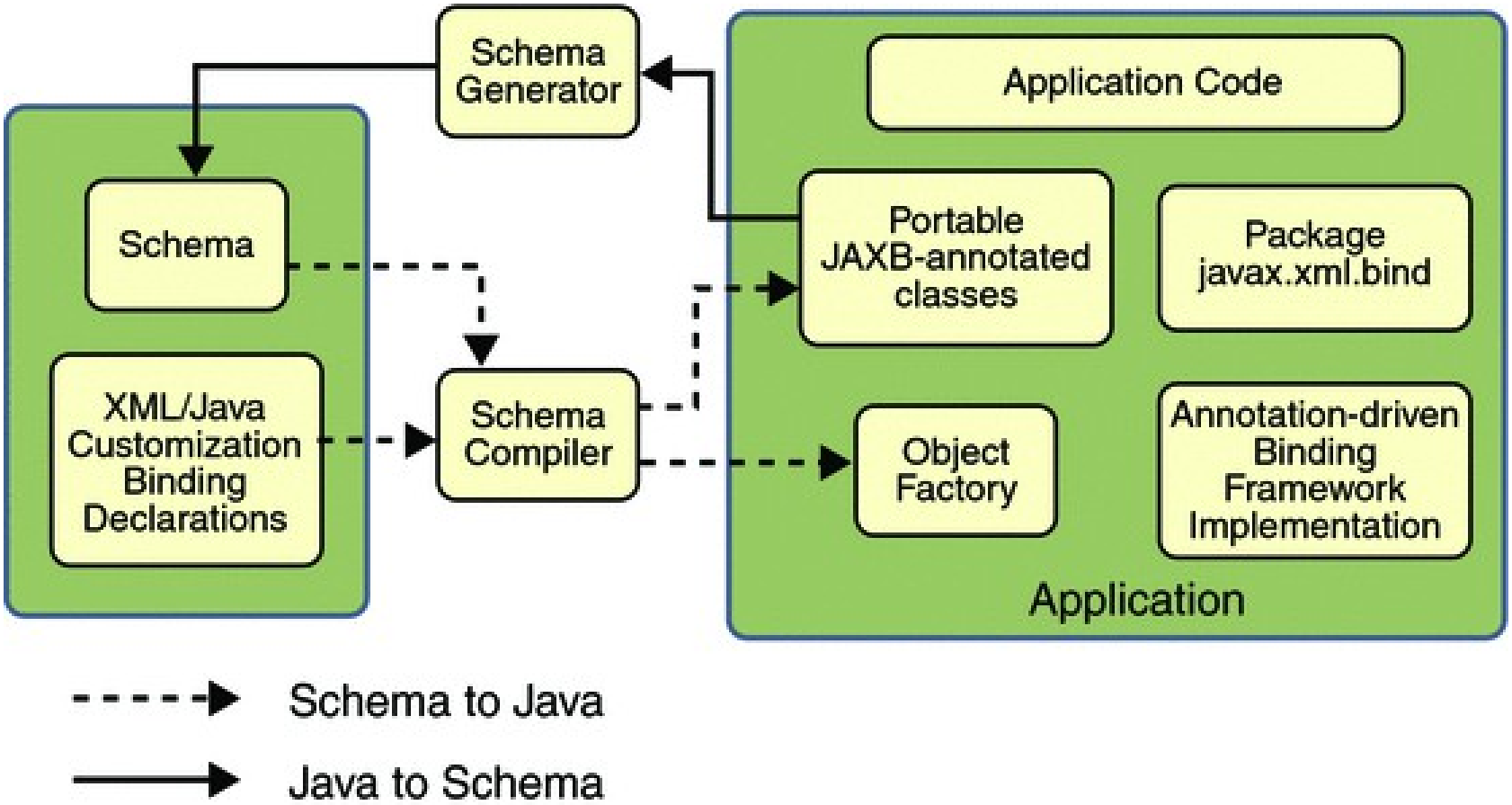
JAXB2

- Requires Java 5 or higher
- Very popular OXM solution
- Enhances JAXB1 in several areas:

- Can use annotations for binding metadata
  - Thus supporting an object-centric approach
- Improved support for XML Schema features
- Improved validation support
  - Can also validate during marshalling

© Copyright 2014 Pivotal. All rights reserved.

JAXB2 Architecture



© Copyright 2014 Pivotal. All rights reserved.

JAXB2 Code Generation

- Uses Binding Compiler (`xjc`)
- Run as script
  - Using Ant or Maven, for example
- Takes list of schemas as input
- Produces packages with Java classes
  - One package per schema
  - This package is called the *context path*
- Customizable via binding files

© Copyright 2014 Pivotal. All rights reserved.

JAXB2 API

```
import java.io.*;
import javax.xml.bind.*;
…
// Step 1: create JAXBContext based on a package name/context path
JAXBContext context = JAXBContext.newInstance("rewards.domain");
// Step 2: create Marshaller or Unmarshaller
Unmarshaller unmarshaller = context.createUnmarshaller();
Marshaller marshaller = context.createMarshaller();
// Step 3: unmarshal…
InputStream is = // some input stream; or Node, Source, …
RewardRequest req = (RewardRequest) unmarshaller.unmarshal(is);
// … or marshal
marshaller.setProperty("jaxb.formatted.output", Boolean.TRUE);
OutputStream os = // some output stream; or Node, Result, …
marshaller.marshal(req, os);
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction
- XML Namespaces
- XML Schema
- Java APIs
- Object-XML Mapping
- JAXB2
- **Spring-OXM**

© Copyright 2014 Pivotal. All rights reserved.

Spring OXM

- Abstraction layer for OXM libraries
  - Provides generic interfaces and exception hierarchy
- Originated as part of Spring Web Services
  - Can be used as a stand-alone module
- Integrated into Spring 3.0 as core module
- Has implementations for all libraries mentioned
- Decouples your code from a specific OXM solution

© Copyright 2014 Pivotal. All rights reserved.

Spring OXM Interfaces

```
public interface Marshaller {
    void marshal(Object graph, Result result)
            throws XmlMappingException, IOException;
}
```

javax.xml.transform.Result

```
public interface Unmarshaller {
    Object unmarshal(Source source)
            throws XmlMappingException, IOException;
}
```

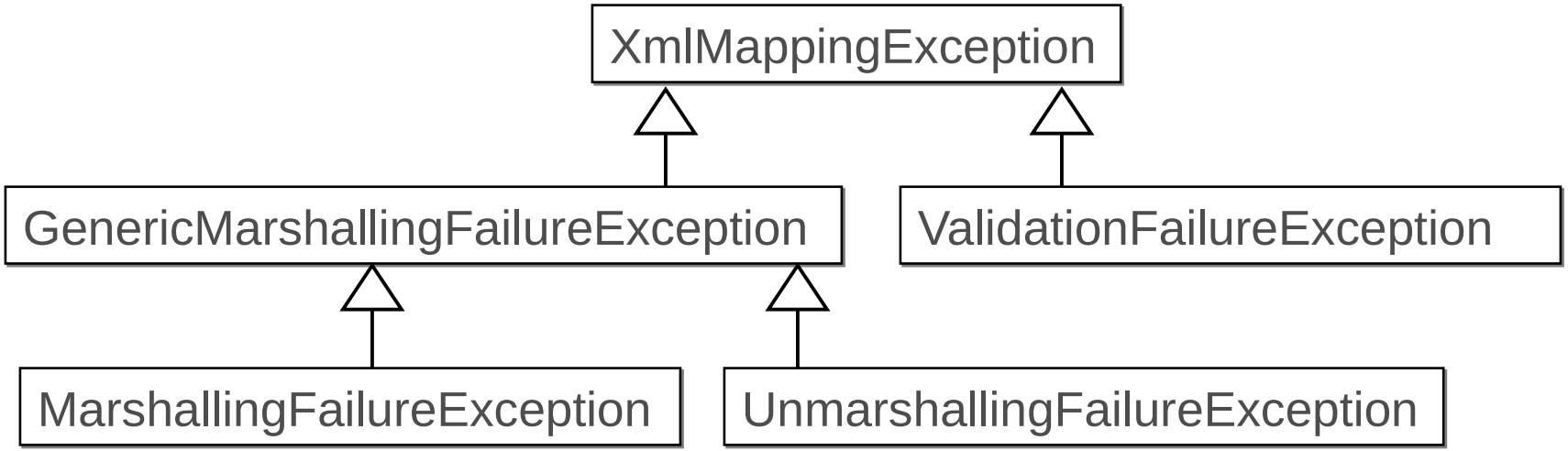
javax.xml.transform.Source

Result and Source may be DOM, SAX, or Stream

© Copyright 2014 Pivotal. All rights reserved.

XmlMappingException Hierarchy

- Exceptions thrown from the underlying marshalling technology are translated into a common hierarchy



© Copyright 2014 Pivotal. All rights reserved.

Spring-OXM Config Example

- To configure with JAXB2, use something like this:

```
<bean id="oxmMarshaller"
            class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
  <property name="contextPath" value="rewards.request"/>
  <property name="marshallerProperties">
    <map>
      <entry key="jaxb.formatted.output">
        <util:constant static-field="java.lang.Boolean.TRUE"/>
      </entry>
    </map>
  </property>
</bean>
```

- Use through Marshaller and Unmarshaller interfaces
- Framework-provided classes implement both

© Copyright 2014 Pivotal. All rights reserved.

Spring-OXM Config <oxm/>

- Namespace support to simplify configuration
  - JAXB2
  - JIBX
  - XMLBeans

```
<oxm:jaxb2-marshaller id="oxmMarshaller"
                      contextPath="rewards.ws.types"/>

<oxm:jibx-marshaller id="oxmMarshaller"  .../>

<oxm:xmlbeans-marshaller id="oxmMarshaller"  .../>
```

- Inject wherever a marshaller or unmarshaller needed

© Copyright 2014 Pivotal. All rights reserved.

Summary

- XML often used as shared data integration format
- Namespaces prevent name clashes
- XML Schema describes valid content for a given namespace
- OXM is a commonly used technique to work with XML
- Several mature solutions exist to help you
  - JAXB2 is particularly relevant
- Spring-OXM provides a common abstraction

© Copyright 2014 Pivotal. All rights reserved.

Lab

Advanced-XML

© Copyright 2014 Pivotal. All rights reserved.

WS-Security

- Authentication
- Signatures
- Encryption

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Core Security Concepts**
- Authentication
- Signatures
- Encryption and Decryption
- WS-Security in Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Core Security Concepts

- Authentication
  - Determining whether a principal is who it claims to be
    - Username/password, or X.509 certificate
- Digital signatures
  - Verifying the origin of a document
    - Uses a private key
  - Checking that the document hasn't been tampered with
- Encryption
  - Making data impossible to read without the proper key
- Decryption
  - Transforming encrypted data back into readable form

© Copyright 2014 Pivotal. All rights reserved.

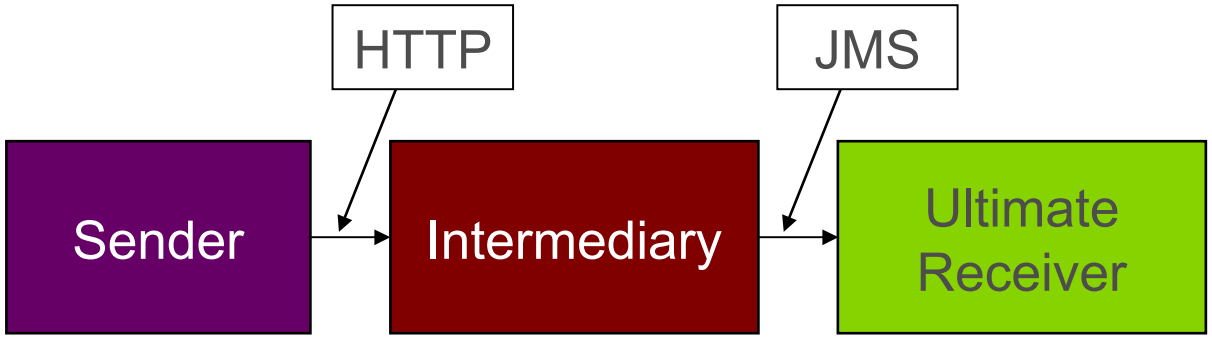
Problems that WS-Security solves

- How to authenticate
  - We can’t use HTTP Basic or Digest
    - Transport-specific
- How to sign messages
- How to encrypt messages

© Copyright 2014 Pivotal. All rights reserved.

Why is HTTP Basic/Digest not enough?

- Security information should not be transport-specific
- The HTTP entry point might not be the ultimate destination
  - HTTP authentication info cannot be re-sent



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Core Security Concepts
- **Authentication**
- Signatures
- Encryption and Decryption
- WS-Security in Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

WS-Security SOAP Header

- Contains:
  - Security Token
    - UsernameToken
    - BinarySecurityToken
  - Signature
  - EncryptedKey

© Copyright 2014 Pivotal. All rights reserved.

Security Tokens

- UsernameToken
  - Username
  - Password
    - Plain text
    - Digest
- BinarySecurityToken
  - X.509 certificate

© Copyright 2014 Pivotal. All rights reserved.

UsernameToken – Plain Text

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security
   xmlns:wsse="http://docs.oasis-open.org/...-wss-wssecurity-secext-1.0.xsd">
   <wsse:UsernameToken>
    <wsse:Username>Bert</wsse:Username>
      <wsse:Password
     Type="http://docs.oasis-open.org/...-wss-username-token-profile-1.0#PasswordText">
       Ernie
     </wsse:Password>
   </wsse:UsernameToken>
  </wsse:Security>
 </SOAP-ENV:Header>

 <SOAP-ENV:Body>
  <tr:transfer xmlns:tr="http://mybank.com/schema/moneytransfer">
   <tr:credit>S123</tr:credit>
   <tr:debit>C456</tr:debit>
   <tr:amount>1205.15</tr:amount>
  </tr:transfer>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Username

Password

© Copyright 2014 Pivotal. All rights reserved.

Digest, Nonce & Timestamp

- Digest
  - “Fingerprint” of the password
  - Includes nonce and timestamp if present
- Nonce
  - Pseudo-random number
  - Prevents replay-attacks
  - Must be cached by the server
- Timestamp
  - When nonce was created
  - Useful for determining nonce expiration to limit server's used nonce cache

© Copyright 2014 Pivotal. All rights reserved.

UsernameToken – Digest

```
<SOAP-ENV:Header>
  <wsse:Security
    xmlns:wsse="http://docs.oasis-open.org/...-wss-wssecurity-secext-1.0.xsd">
    <wsse:UsernameToken
      xmlns:wsse="http://docs.oasis-open.org/...-wss-wssecurity-secext-1.0.xsd">
      <wsse:Username>Bert</wsse:Username>

      <wsse:Password Type="http://docs.oasis-open.org/wss/...1.0#PasswordDigest">
        kwNstEaiFOrI7B31j7GuETYvdgk=
      </wsse:Password>

      <wsse:Nonce  EncodingType="http://docs.oasis-open.org/...1.0#Base64Binary">
        9mdsYDCrjjYRur0rxzYt2oD7
      </wsse:Nonce>

      <wsu:Created xmlns:wsu="http://docs.oasis-open.org/...-wss-wssecurity-utility-1.0.xsd">
        2006-06-01T23:48:42Z
      </wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>

</SOAP-ENV:Header>
…
```

Username

Digest

Nonce

Timestamp

© Copyright 2014 Pivotal. All rights reserved.

BinarySecurityToken

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security
   xmlns:wsse="http://docs.oasis-open.org/...-wss-wssecurity-secext-1.0.xsd">
   <wsse:BinarySecurityToken
    EncodingType="http://docs.oasis-open.org/...-soap-message-security-1.0#Base64Binary"
    ValueType="http://docs.oasis-open.org/wss/...-x509-token-profile-1.0#X509v3">
    MIICXDCC…v5CtcKv4QduvrHPrR8=
   </wsse:BinarySecurityToken>
  </wsse:Security>
 </SOAP-ENV:Header>

 <SOAP-ENV:Body>
  <tr:transfer xmlns:tr="http://mybank.com/schema/moneytransfer">
   <tr:credit>S123</tr:credit>
   <tr:debit>C456</tr:debit>
   <tr:amount>1205.15</tr:amount>
  </tr:transfer>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Base64-encoded X.509 certificate

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Core Security Concepts
- Authentication
- **Signatures**
- Encryption and Decryption
- WS-Security in Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Signatures

- WS-Security uses the XML signature (XMLDsig) spec
- Signature method
  - Sha1, md5, etc.
- Reference
  - What needs to be signed?
- Canonicalization
  - How to deal with whitespace, ordering of attributes, etc.
- Signature!

© Copyright 2014 Pivotal. All rights reserved.

Signature

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/...-wss-wssecurity-secext-1.0.xsd"
   SOAP-ENV:mustUnderstand="1">
   <wsse:BinarySecurityToken>… </wsse:BinarySecurityToken>

   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
     <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
     <ds:Reference URI="#XWSSGID-1149203417054-305519969">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>/ewlu0ecAQ2DsEJqDcM4AT6oWPE=</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>

    <ds:SignatureValue>hZA0zib…m9I=</ds:SignatureValue>
    <ds:KeyInfo>…</ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/…wssecurity-utility-1.0.xsd"
   wsu:Id="XWSSGID-1149203417054-305519969">
…
</SOAP-ENV:Envelope>
```

How to deal with whitespace

Algorithm

Signature

© Copyright 2014 Pivotal. All rights reserved.

Signed content

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>

      <ds:SignedInfo>
        ...
        <ds:Reference URI=                                    >
        ...
        </ds:Reference>
      </ds:SignedInfo>
    </ds:Signature>
  </wsse:Security>
</SOAP-ENV:Header>

<SOAP-ENV:Body                                    
          wsu:Id= "XWSSGID-1149203417054-305519969"
          xmlns:wsu="http://docs.oasis-open.org/…wssecurity-utility-1.0.xsd" >
…

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Core Security Concepts
- Authentication
- Signatures
- **Encryption and Decryption**
- WS-Security in Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Encryption

- WS-Security uses the XML encryption (XMLEnc) spec
- Encryption method
  - Rsa1, etc.
- Reference
  - What needs to be encrypted?
- Encrypted data

© Copyright 2014 Pivotal. All rights reserved.

Encryption

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/...-wssecurity-secext-1.0.xsd">
   <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>

   <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
    <xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
     <xenc:DataReference URI="#XWSSGID-1149204675657-83290958" />
    </xenc:ReferenceList>
   </xenc:EncryptedKey>
  </wsse:Security>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
                      Id="XWSSGID-1149204675657-83290958">
   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
   <xenc:CipherData>
    <xenc:CipherValue>
      pOP…PZCR
    </xenc:CipherValue>
   </xenc:CipherData>
  </xenc:EncryptedData>
```

Algorithm

Encrypted Data

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Core Security Concepts
- Authentication
- Signatures
- Encryption and Decryption
- **WS-Security in Spring Web Services**

© Copyright 2014 Pivotal. All rights reserved.

WS-Security Support in Spring-WS

- XwsSecurityInterceptor supports WS-Security
- Based on SUN's XML and Web Services Security package (XWSS)
  - Configured through policy files
  - Uses callbacks to perform actions
  - Requires Sun JVM and SAAJ
- Integrates with Spring Security
- WSS4J also supported as XWSS alternative
  - Through Wss4jSecurityInterceptor
  - For use with non-Sun JVMs or Axiom instead of SAAJ

© Copyright 2014 Pivotal. All rights reserved.

Example config XwsSecurityInterceptor

```
<ws:interceptors>
  <bean class="org.sfw.ws.soap.security.xwss.XwsSecurityInterceptor">
    <property name="policyConfiguration"
            value="classpath:securityPolicy.xml" />

    <property name="callbackHandlers">
      <list>
        <ref bean="certificateHandler" />
        <ref bean="authenticationHandler" />
      </list>
    </property>
  </bean>
</ws:interceptors>
```

XWSS policy file

Callback handlers

© Copyright 2014 Pivotal. All rights reserved.

XWSS configuration

```
<xwss:SecurityConfiguration dumpMessages="true"
    xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
   <xwss:RequireUsernameToken passwordDigestRequired="false"
    nonceRequired="false"/>
</xwss:SecurityConfiguration>
```

Plain text

```
<xwss:SecurityConfiguration xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
   <xwss:UsernameToken digestPassword="true" useNonce="true"/>
</xwss:SecurityConfiguration>
```

Digest

```
<xwss:SecurityConfiguration xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
  <xwss:RequireSignature requireTimestamp="false" />
  <xwss:Encrypt />
</xwss:SecurityConfiguration>
```

Signature & Encrypt

© Copyright 2014 Pivotal. All rights reserved.

Authentication handlers

- Username
  - SimplePasswordValidationCallbackHandler
    - Uses properties for username
  - SpringPlainTextPasswordValidationCallbackHandler
  - SpringDigestPasswordValidationCallbackHandler
  - JaasPlainTextPasswordValidationCallbackHandler
- X.509
  - KeyStoreCallbackHandler
  - SpringCertificateValidationCallbackHandler
  - JaasCertificateValidationCallbackHandler

© Copyright 2014 Pivotal. All rights reserved.

Example config to authenticate

```
<ws:interceptors>
  <bean class="org.springframework.ws.soap.security.xwss.XwsSecurityInterceptor">
    <property name="secureResponse" value="false" />
    <property name="policyConfiguration" value="classpath:securityPolicy.xml" />
    <property name="callbackHandler">
      <bean  class="org.swrk.ws...SpringDigestPasswordValidationCallbackHandler">
        <property name="userDetailsService" ref="securityService" />
      </bean>
    </property>
  </bean>
</ws:interceptors>
```

Spring Security Service

© Copyright 2014 Pivotal. All rights reserved.

Java Keystores

- Part of J2SE
- “keytool” program
- Storage for cryptographic keys and certificates
- Private keys
  - Signing and decryption
- Symmetric keys
  - Signing and decryption
  - Sender and recipient share the same key
- Trusted certificates
  - Certificate validation, signature verification, and encryption
- Spring-WS has KeyStoreFactoryBean

© Copyright 2014 Pivotal. All rights reserved.

keytool

Private Key password

Distinguished name

```
#keytool -genkey -dname "cn=John Doe, ou=Transfers,
          o=MyBank, c=US" -alias business
  -keypass secret -keystore keystore.jks
  -storepass changeit -validity 180
```

Filename

Keystore password

Validity (in days)

```
#keytool -list -keystore keystore.jks -storepass
   changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry
business, Nov 17, 2007, keyEntry,
Certificate fingerprint (MD5):
   30:5A:D8:2B:03:B6:DB:51:04:08:66:8C:E2:A9:15:4F
```

© Copyright 2014 Pivotal. All rights reserved.

KeyStoreFactoryBean

```
<bean id="keyStore"
    class="org.springframework.ws.soap.security.support.KeyStoreFactoryBean">
   <property name="password" value="”changeit"/>
   <property name="location" value="/WEB-INF/keystore.jks"/>
</bean>
```

Keystore password

© Copyright 2014 Pivotal. All rights reserved.

KeyStoreCallbackHandler

- Uses keystores to give XWSS security info

Cryptographic operation	**Keystore used**
Certificate validation	keyStore, then trustStore
Decryption based on private key	keyStore
Decryption based on symmetric key	symmetricStore
Encryption based on public key certificate	trustStore
Encryption based on symmetric key	symmetricStore
Signing	keyStore
Signature verification	trustStore

© Copyright 2014 Pivotal. All rights reserved.

Example config to decrypt

```
<beans>

  <bean id="keyStoreHandler"
     class="org.swrk.ws.soap.security.xwss.callback.KeyStoreCallbackHandler">
    <property name="keyStore" ref="keyStore" />
    <property name="privateKeyPassword" value="“secret" />
  </bean>

  <bean id="keyStore"
    class="org.springframework.ws.soap.security.support.KeyStoreFactoryBean">
    <property name="location" value="classpath:keystore.jks" />
    <property name="password" value="changeit" />
  </bean>

</beans>
```

Private Key password

Filename

Keystore password

© Copyright 2014 Pivotal. All rights reserved.

Summary

- WS-Security gives Message-level security
- WS-Security supports
  - Authentication
  - Signatures
  - Encryption and Decryption
- Spring-WS has the XwsSecurityInterceptor and Wss4jSecurityInterceptor and integrates with keystores, Spring Security or JAAS

© Copyright 2014 Pivotal. All rights reserved.

Lab

Implementing WS-Security
with
Spring Web Services

© Copyright 2014 Pivotal. All rights reserved.

Mature REST services

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Richardson Maturity Model**
- HATEOAS
- Spring HATEOAS
- Spring Data REST
- Out-of-container Testing

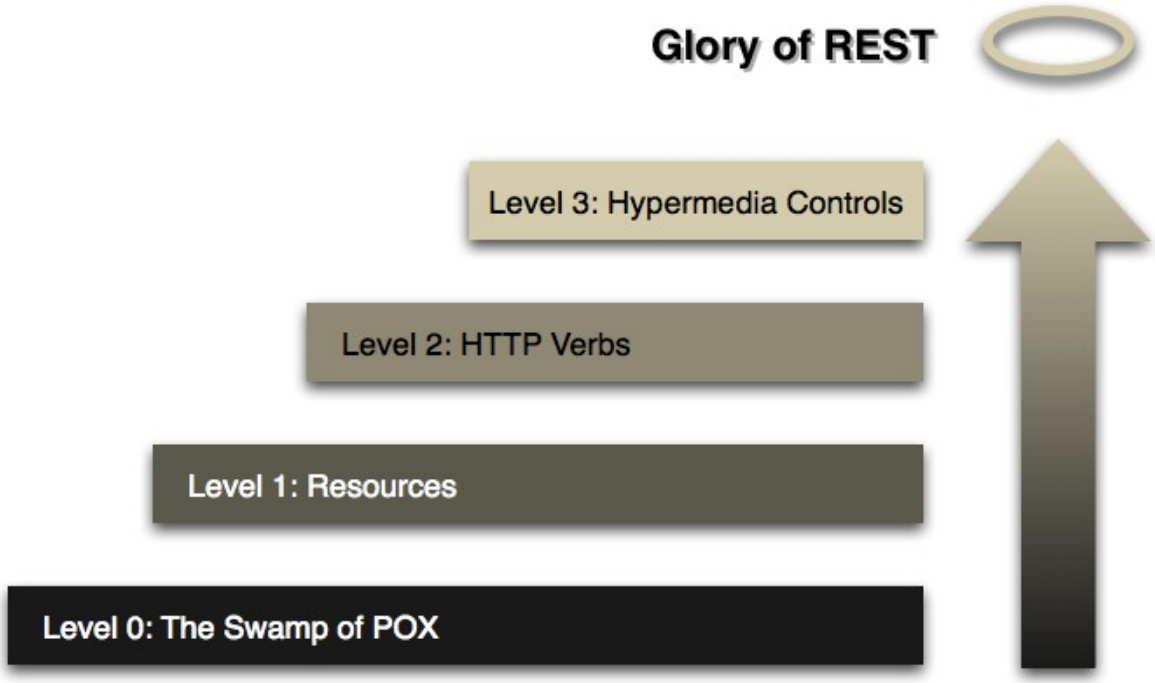
© Copyright 2014 Pivotal. All rights reserved.

Richardson Maturity Model

- REST is more than GETting objects converted to JSON/XML
- There are several levels of REST adoption
- Expressed by Richardson Maturity Model

© Copyright 2014 Pivotal. All rights reserved.

Richardson Maturity Model



See http://martinfowler.com/articles/richardsonMaturityModel.html

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Richardson Maturity Model
- **HATEOAS**
- Spring HATEOAS
- Spring Data REST
- Out-of-container Testing

© Copyright 2014 Pivotal. All rights reserved.

HATEOAS

- **H**ypermedia **A**s **T**he **E**ngine of **A**pplication **S**tate
- Resources should contain links to other resources.
- Clients decide whether to follow the links

© Copyright 2014 Pivotal. All rights reserved.

Non HATEOAS Representation


```
{"id":2,
"firstName":"Lois",
"profilePhotoMediaType":null,
"lastName":"Lane",
"username":"loislane",
"password":null,
"profilePhotoImported":false,
"enabled":true,
"signupDate":1370201631000,
"customers":[
{"id":11,"signupDate":1371091042000,"firstName":"David","lastName":"Syer","da
tabaseId":11},
{"id":17,"signupDate":1372461079000,"firstName":"Scott","lastName":"Andrews",
"databaseId":17},
{"id":19,"signupDate":13724636540
databaseId":19}]
}
```


User has collection of Customers
What if Customer references User?

© Copyright 2014 Pivotal. All rights reserved.

HATEOAS Representation


```
{"links":[
  {"rel":"self","href":"http://localhost:8080/users/2"},
  {"rel":"customers","href":"http://localhost:8080/users/2/customers"},
  {"rel":"photo","href":"http://localhost:8080/users/2/photo"}],
"id":2,
"firstName":"Lois",
"profilePhotoMediaType":null,
"lastName":"Lane",
"username":"loislane",
"password":null,
"profilePhotoImported":false,
"enabled":true,
"signupDate":1370201631000}
```


Return links to related resources
Client can follow links (if desired)

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Richardson Maturity Model
- HATEOAS
- **Spring HATEOAS**
- Spring Data REST
- Out-of-container Testing

© Copyright 2014 Pivotal. All rights reserved.

Spring HATEOAS

- Subproject within the Spring Family
  - http://projects.spring.io/spring-hateoas/
- Applies HATEOAS principle to REST resources
  - Slight modification to Controllers
- Key Concepts:
  - Special “Resource” class.
  - ResourceAssembler interface.
  - HttpEntity<T>

© Copyright 2014 Pivotal. All rights reserved.

Resource & ResourceAssembler

- Spring HATEOAS Resource class:
  - Wraps a domain object
  - Adds Links to it

```
public class Resource<T> extends ResourceSupport {
    …
}
```

- Spring HATEOAS ResourceAssembler interface:
  - Describes domain to Resource conversion logic.

```
public interface ResourceAssembler<T, D extends ResourceSupport> {
    /**
     * Converts the given entity into an {@link ResourceSupport}.
     */
    D toResource(T entity);
}
```


© Copyright 2014 Pivotal. All rights reserved.

HttpEntity<T>

- Spring Web HttpEntity<T>
  - General purpose object in Spring Web
  - Refines definition of HttpRequest / Response
    - Consists of headers and body

```
public class HttpEntity<T> {
    public HttpEntity(T body) {
        this(body, null);
    }
    ...
    public HttpHeaders getHeaders() { return headers; }
    public T getBody() { return body; }
}
```

- Used in Spring MVC as return value from @Controller method.

© Copyright 2014 Pivotal. All rights reserved.

Spring HATEOAS Applied

- Before:

```
@RequestMapping (method=RequestMethod.GET, value="/users/{user}")
@ResponseBody
User loadUser(@PathVariable Long user) {
    return crmService.findById(user);
}
```

- After:

```
@RequestMapping (method=RequestMethod.GET, value="/users/{user}")
HttpEntity<Resource<User>> loadUser(@PathVariable Long user) {
    User u = service.findById(user)
    Resource<User> resource = userResourceAssembler.toResource(u);
    return new ResponseEntity<Resource<User>>(resource, HttpStatus.OK);
}
```


© Copyright 2014 Pivotal. All rights reserved.

ResourceAssembler

```
class UserResourceAssembler implements ResourceAssembler<User, Resource<User>> {

public Resource<User> toResource(User user) {
  ...
    Resource<User> userResource = new Resource<User>(user);
    Collection<Link> links = new ArrayList<Link>();

    links.add(
      linkTo(methodOn(UserController.class).loadUser(userId))
      .withSelfRel());
    links.add(
      linkTo(methodOn(UserController.class).loadUserCustomers(userId))
      .withRel("customers"));

    for (Link l : links) {   userResource.add(l);   }
    return userResource;
   ...
}
```

Create Resource and Links collection

Create "self" link

Create "customers" link

Add links to Resource

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Richardson Maturity Model
- HATEOAS
- Spring HATEOAS
- **Spring Data REST**
- Out-of-container Testing

© Copyright 2014 Pivotal. All rights reserved.

Spring Data REST

- Why write Controllers and ResourceAssemblers at all?
- Spring Data REST defines endpoints based on conventions
- A subproject within Spring Data umbrella
  - http://projects.spring.io/spring-data-rest/

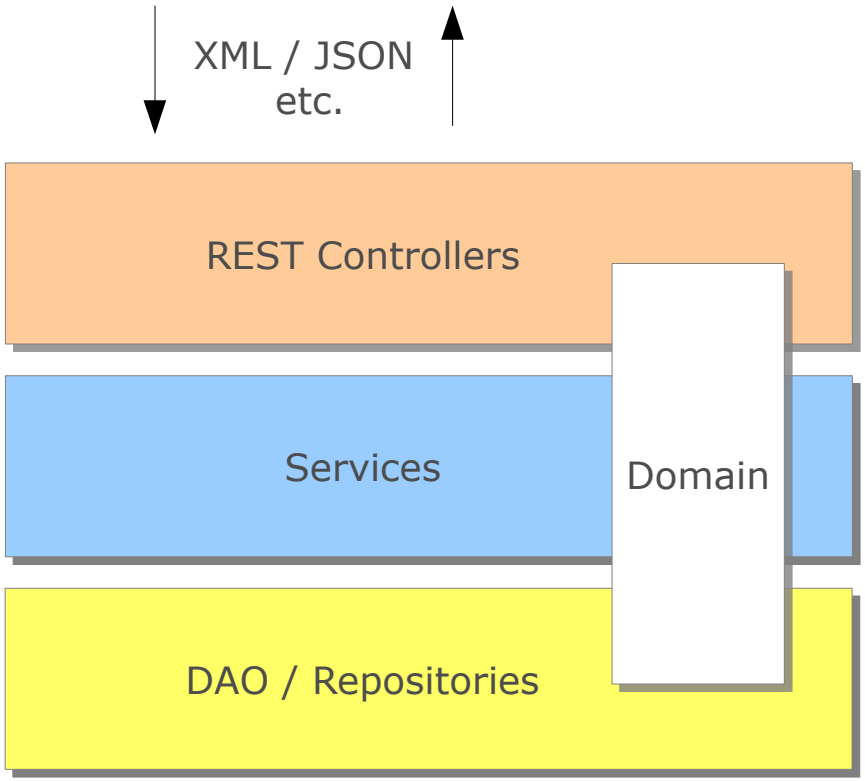
© Copyright 2014 Pivotal. All rights reserved.

Typical REST Application

Typical web application architecture

REST Controllers provide CRUD interface to clients

DAO provide CRUD interface to DB



© Copyright 2014 Pivotal. All rights reserved.

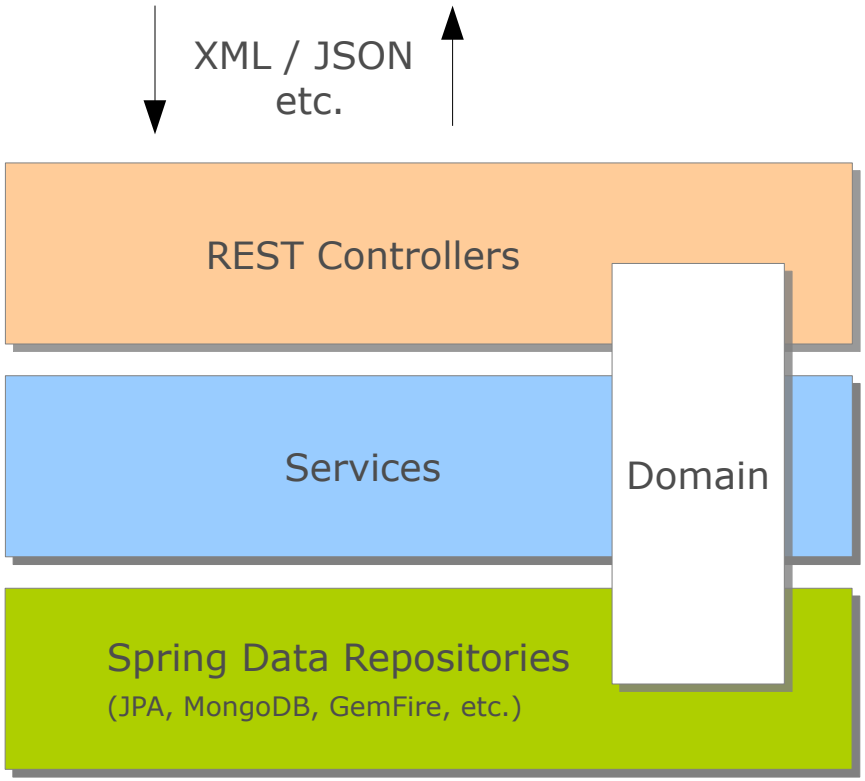
Add Spring Data

Spring Data provides dynamic Repositories

You provide the interface, Spring Data dynamically implements.

- JPA, MongoDB, GemFire, etc.

Sevice Layer / Controllers have almost no logic.



© Copyright 2014 Pivotal. All rights reserved.

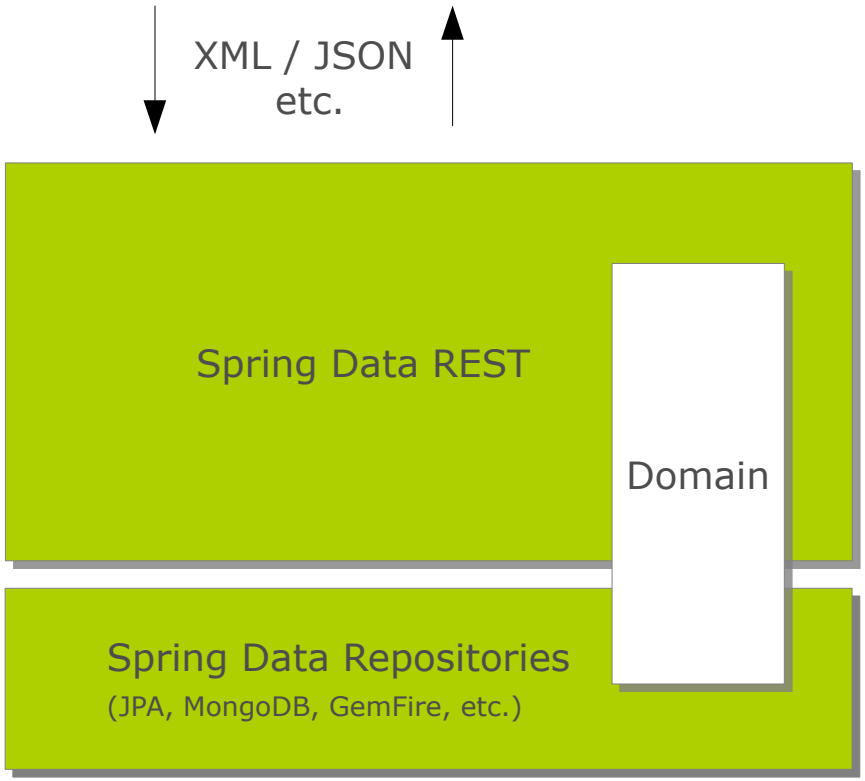
Spring Data REST

Plugs into dynamic repositories

Generates RESTful interface

- GET, PUT, POST, DELETE

Code needed only to override defaults.



© Copyright 2014 Pivotal. All rights reserved.

Quick Start

- Annotate Domain objects
  - i.e. use JPA annotations if using JPA
- Define interface for dynamic repository
  - Implement CrudRepository at minimum
  - Add @RestResource to describe how to expose via REST

```
@RestResource(path="users", rel="users")
public interface UserRepository extends CrudRepository<User, Long> {}
```

- Configure Spring Data REST
  - `@Import(RepositoryRestMvcConfiguration.class)`
  - Or, extend this class to override defaults.

© Copyright 2014 Pivotal. All rights reserved.

HTTP Verb to Method Mapping

Verb	Repository Method
GET	`CrudRepository<ID,T>.findOne(ID id)`
POST	`CrudRepository<ID,T>.save(T entity)`
PUT	`CrudRepository<ID,T>.save(T entity)`
DELETE	`CrudRepository<ID,T>.delete(ID id)`

- All methods available by default
  - Disable via override / configuration

```
//  Restrict DELETE:
@RestResource(exported=false)
void delete(Long id);
```


© Copyright 2014 Pivotal. All rights reserved.

Links

- Links are inferred based on relationships
  - JPA - @ManyToOne, @OneToOne, etc.
  - Mongo - @DBRef
- Specific links can be added manually

```
@Component
public class ExtraLinks {
  @Inject
  EntityLinks(EntityLinks entityLinks) {
    this.entityLinks = entityLinks;
  }

  Link getSelfLink(User user) {
    return entityLinks.linkForSingleResource(User.class, user.getId()).withSelfRel();
  }
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Richardson Maturity Model
- HATEOAS
- Spring HATEOAS
- Spring Data REST
- **Out-Of-Container Testing**

© Copyright 2014 Pivotal. All rights reserved.

The Need...

- Spring can eliminate a lot of the code
  - ...but the need for functional testing remains
- Very important to test the entire application from REST interface to DB
  - This usually involves deploying to an app server
- Spring's MVC Test Framework
  - Test entire application stack without app server

© Copyright 2014 Pivotal. All rights reserved.

MVC Test Framework Overview

- Part of Spring Framework since 3.2
  - Previously separate GitHub project
  - Found in spring-test.jar
- Goal: Provide first class JUnit support for testing Spring MVC code
  - Fluent API
  - Process requests through DispatcherServlet
  - Does NOT require Web container to test

© Copyright 2014 Pivotal. All rights reserved.

Setting Up The Test Harness

```
import static o.sfw.test.web.servlet.request.MockMvcRequestBuilders.*;
import static o.sfw.test.web.servlet.result.MockMvcResultMatcher.*;
import static o.sfw.test.web.servlet.result.MockMvcResultHandlers.*;

@ContextConfiguration("classpath:test-servlet-context.xml")
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
public final class AccountControllerTests {
    @Autowired
    private WebApplicationContext wac;
    private MockMvc mockMvc;

    @Before
    public void setup() {
        // Initialize mockMvc using WebApplicationContext
        mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
    }

    // Now perform tests using the mockMvc object...
}
```

Static imports: easier to invoke static methods

Instantiate WebApplicationContext

Inject WebApplicationContext

Define MockMvc field and initialize it

© Copyright 2014 Pivotal. All rights reserved.

Setting Up Static Imports

- Static imports are key to fluid builders
  - `MockMvcRequestBuilders.*` and `MockMvcResultMatchers.*`
- You can add to Eclipse '*favorite static members*' in preferences
  - *Java → Editor → Content Assist → Favorites*
  - Add to favorite static members
    - `org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get`
    - `org.springframework.test.web.servlet.result.MockMvcResultMatchers.status`

© Copyright 2014 Pivotal. All rights reserved.

Running a Test: Perform and Expect

```
@Test
public void testPut() {
  mockMvc.perform(
    put("/accounts/{acctId}", "123456001")            // PUT request
    .content( “{\"accountName\":\"json is nice\"}” )          // with JSON
    .contentType(MediaType.APPLICATION_JSON )       // and specific type
    // … many other static methods available ...
  )
  .andExpect(status().isOk())                                 // Expect status code 200
  .andExpect(content().contentType("application/json...")       // ...with a content type
  .andExpect(jsonPath("$username").value("loislane"))  // ...with specific JSON
  .andDo(print())                                 // ...and log to sysout too
    // … many other static methods available ...
}
```

Perform...

...Expect

© Copyright 2014 Pivotal. All rights reserved.

Perform() method

- Takes a `RequestBuilder` argument
  - Usually a `MockMvcRequestBuilders` static method
  - *get*(), *put*(), *post*(), etc.
- Returns `ResultsActions` object
  - Allows expects to be chained – fluid syntax

```
mockMvc.perform(
  get("/accounts/{acctId}", "123456001"))
  .andExpect(...)
  .andExpect(...)
  .andExpect(...)
  .andExpect(...);
```

`MockMvcRequestBuilders` Static methods: what to do.

Perform() returns `ResultsAction` object

Each andExpect() method returns `ResultsAction`

© Copyright 2014 Pivotal. All rights reserved.

RequestBuilderMethods

- Standard HTTP *get*, *put*, *post*, *delete* operations
  - Argument usually a URI template string
  - Embellish with *content, contentType, headers*, etc.
  - Returns a `MockHttpServletRequestBuilder` instance

```
// Perform a get using URI template style
mockMvc.perform(get("/accounts/{acctId}", "123456001"))

// Perform a put
mockMvc.perform(
    put("/accounts/{acctId}", "123456001"")          // PUT request
    .content( “{\"accountName\":\"json is nice\"}” )  // with JSON
    .contentType(MediaType.APPLICATION_JSON )    // and specific type
    …
)
```


© Copyright 2014 Pivotal. All rights reserved.

MockHttpServletRequestBuilder static methods

Method	Description
header	Add a header variable to the request.
headers	Adds multiple headers
contentType	Set content type (Mime type) for body of the request
content	Actual request body to be sent
accept	Set requested type (Mime type) of expected response
locale	Set the local for making requests

- Many others available, see JavaDoc

© Copyright 2014 Pivotal. All rights reserved.

Checking Expectations

- Perform() returns `ResultsActions` object
  - Allows expects to be chained – fluid syntax

```
mockMvc.perform(
  get("/accounts/{acctId}", "123456001"))
  .andExpect(status().isOk())                                  // Expect status code 200
  .andExpect(content().contentType("application/json...")      // ...with a content type
  .andExpect(jsonPath("$username").value("loislane"))  // ...with specific JSON
  .andDo(print())                                  // ...and log to sysout too
   // … many other static methods available ...
}
```

© Copyright 2014 Pivotal. All rights reserved.

MockMvcResultMatcher methods

- `MockMvcResultMatcher` methods allow assertions on results

Method	Description	Examples
status()	HTTP status	*status*().isOk(), *status*().isNotFound()
content()	HTTP response body	*content*() .contentType("application/json")
header()	HTTP headers	*header*() .string("content-length",0)
xPath()	Perform Xpath expressions on content.	TODO – ADD XPATH
jsonPath()	Perform JsonPath expressions on content	*jsonPath*("$username") .value("loislane")

© Copyright 2014 Pivotal. All rights reserved.

Printing Debug Information

- Sometimes you want to know what happened
  - andDo() performs action on MvcResult
  - print() sends the MvcResult to output stream
  - Could also use andReturn() to get the MvcResult object

```
// Use this to access the print() method
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.*;
// Other static imports as well

// Use print() method in test to get debug information
mockMvc.perform(get("/accounts/{acctId}","123456001")
            .andDo(print())  // Add this line to print debug information to the console
            …
```


© Copyright 2014 Pivotal. All rights reserved.

Summary

- RESTful applications utilize hypermedia/links
- Spring HATEOAS adds links to existing Spring MVC RESTful resources
- Spring Data REST provides automatic REST implementations for CRUD repositories
- Spring's MVC Test Framework can create full integration tests without a container

© Copyright 2014 Pivotal. All rights reserved.

Spring AMQP

Simplifying Messaging Applications

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- **Introduction to AMQP**
- RabbitMQ
- Configuring AMQP Resources with Spring
- Spring's AmqpTemplate
- Sending Messages
- Receiving Messages

© Copyright 2014 Pivotal. All rights reserved.

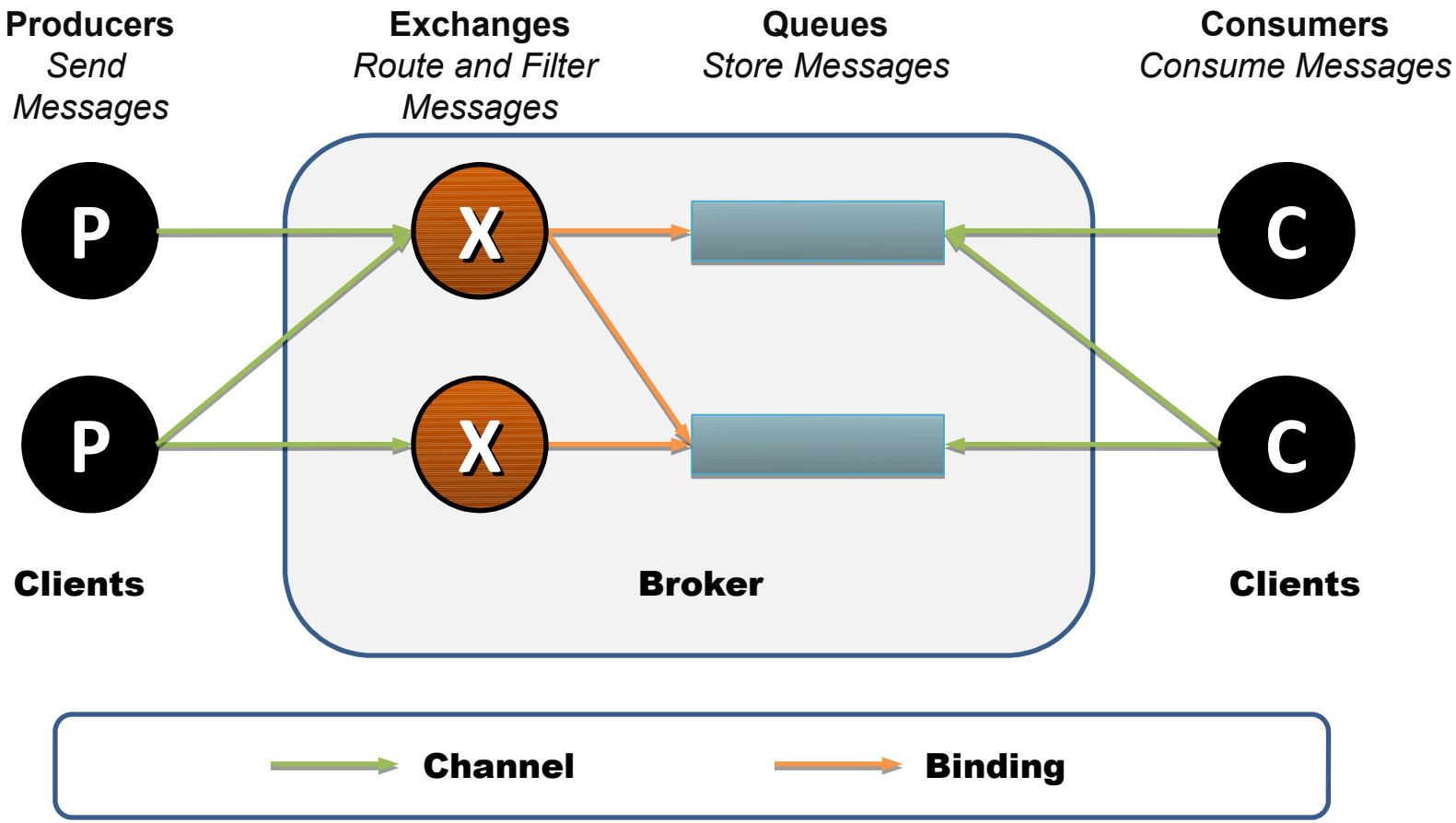
Advanced Message Queuing Protocol
AMQP

- AMQP is a messaging protocol that is compatible at the binary (or wire) level
  - Avoid vendor lock-in
  - Increase portability

- AMQP enables different Message-Oriented Middleware (MOM) vendors to communicate
  - Bridge is not required

© Copyright 2014 Pivotal. All rights reserved.

The AMQP Model



© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

- **Queue**
  - A container for holding messages
  - The first message placed into the container is the first message that can be removed (FIFO).

- **Exchange**
  - Exchanges are responsible for applying routing rules to messages in order to deliver them to the correct queue(s) for consumption by consumers.
    - 4 types: direct, fanout, topic, and headers
    - The type of exchange, in combination with message headers, determine routing

© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

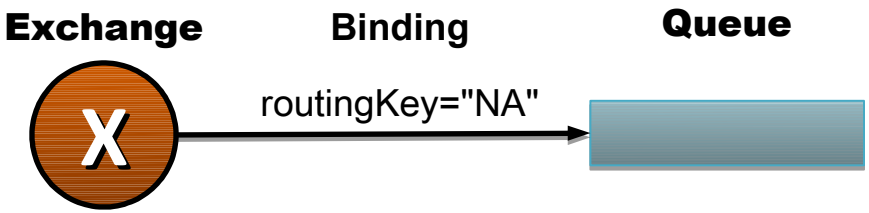
- **Producer (or Publisher)**
  - Client that creates and sends messages. Producers always publish messages to exchanges.

- **Consumer (or Listener, Subscriber)**
  - Client that consumes and processes messages
  - Consumers always consume messages from queues.
    - Can poll queue synchronously (blocking), or
    - Can register an asynchronous callback (non-blocking, preferred)

© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

- **Binding**
  - Bindings are rules that exchanges use to route messages to queues, or to other exchanges
    - The type of exchange will dictate which message header properties are considered for the binding
    - For example, direct exchanges have bindings that route messages based on the *routingKey* message header property



© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

- **Broker**
  - Brokers are AMQP servers that manage all AMQP resources
    - *Exchanges*, *Queues*, *Bindings*, etc.
  - *RabbitMQ* is a an example of a Broker.

© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

- **Connection**
  - A connection is a TCP socket between a client and a broker.
  - An AMQP Connection is obtained from a factory

```
ConnectionFactory factory = new ConnectionFactory();
factory.setUsername("guest");
factory.setPassword("guest");
factory.setVirtualHost("/");
factory.setHost("localhost");

int serverPort = 5672;
factory.setPort(serverPort);
Connection connection = factory.newConnection();
```


© Copyright 2014 Pivotal. All rights reserved.

AMQP Terminology

- **Channel**
  - The means through which multiple parallel operations can be performed over a single connection.
  - Multiple channels are multiplexed over a single connection.

- Channels can be shared by multiple threads
  - Care must be taken that only a *single* thread executes a command at a given point in time

```
Channel channel = connection.createChannel();
```


© Copyright 2014 Pivotal. All rights reserved.

AMQP Messages

- **AMQP Messages** consist of:
  - Header properties (i.e. key-value pairs)
  - Body

- Some pre-defined header properties provide instructions to the broker on how to handle the message.
  - For example, *"routingKey"* is typically used for routing.
  - *"deliveryMode"* tells the broker whether to persist the message to disk.

© Copyright 2014 Pivotal. All rights reserved.

AMQP Messages

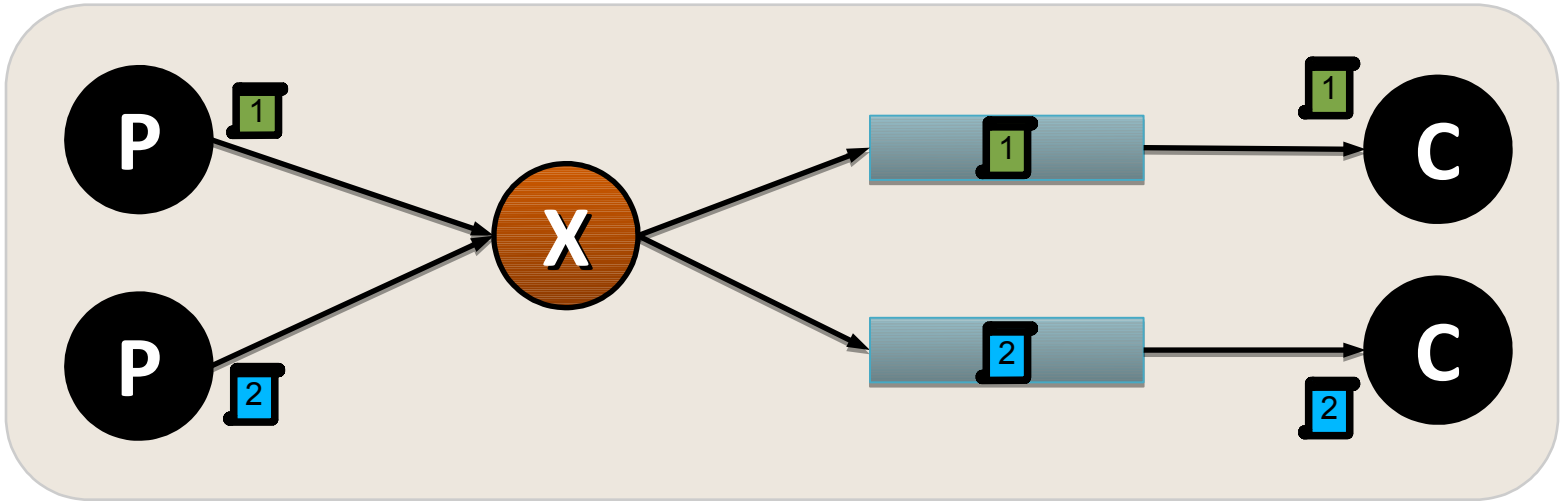
- Message body is binary – a simple byte array
- Separate parameters for the header properties and the message payload are provided in the Java client API

```
channel.basicPublish(exchange,
   routingKey,
   new AMQP.BasicProperties.Builder()
           .deliveryMode(2).build()),
   msgBodyBytes);
```


© Copyright 2014 Pivotal. All rights reserved.

Point-to-Point Pattern

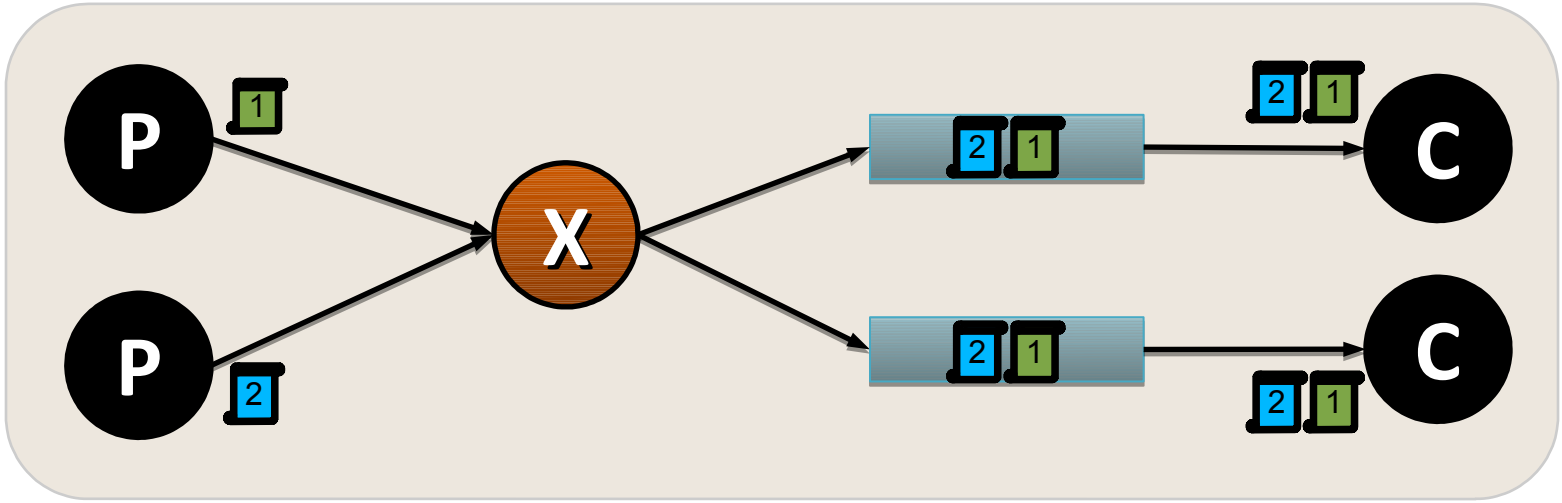
1. Message sent to exchange (eg. "direct", not a fanout exchange)
2. Routed to queue based on binding
3. Message queued
4. Message consumed by one and only one consumer



© Copyright 2014 Pivotal. All rights reserved.

Publish-Subscribe Pattern

1. Message sent to fanout exchange
2. Routed to all queues bound to the exchange
3. Message queued
4. Message consumed by multiple consumers – each gets a copy of the message.



© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to AMQP
- **RabbitMQ**
- Configuring AMQP Resources with Spring
- Spring's AmqpTemplate
- Sending Messages
- Receiving Messages

© Copyright 2014 Pivotal. All rights reserved.

AMQP Compliant Brokers

- Most common ones are:
  - Pivotal's RabbitMQ
  - Apache Qpid
- We will use RabbitMQ!

© Copyright 2014 Pivotal. All rights reserved.

RabbitMQ

- Open source AMQP compliant message broker written in Erlang
- Officially supported clients:
  - Java, .NET, Erlang, JMS (commercial only)
- Many third party OSS clients:
  - C, Python, Ruby, PHP, Perl, etc.

© Copyright 2014 Pivotal. All rights reserved.

RabbitMQ Features

- Clustering for High Availability
- Mirrored queues for guaranteed messaging
- Shovel and Federation plugins for WAN replication
- LDAP Authentication plugin
- Web-based management console plugin
- Management REST API for monitoring
- Extensible through custom plugins
- Comprehensive list at http://www.rabbitmq.com/features.html

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to AMQP
- RabbitMQ
- **Configuring AMQP Resources with Spring**
- Spring's AmqpTemplate
- Sending Messages
- Receiving Messages

© Copyright 2014 Pivotal. All rights reserved.

Configuring AMQP Resources with Spring

- Spring enables decoupling of your application code from the underlying infrastructure
  - Container provides the resources
  - Application is simply coded using POJO's

- Provides and abstracts from the developer much of the tedious boilerplate code for things such as connection retries, transactions, etc.

© Copyright 2014 Pivotal. All rights reserved.

Configuring a ConnectionFactory

- Typically use the *CachingConnectionFactory*
  - Caches channels and connections for reuse
  - Use carefully, see documentation:
    - http://docs.spring.io/spring-amqp/reference/html/amqp.html#connections

```
<bean id="connectionFactory"
        class="org.springframework.amqp.
                    rabbit.connection.CachingConnectionFactory">
   <constructor-arg value="somehost"/>
   <property name="username" value="guest"/>
   <property name="password" value="guest"/>
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to AMQP
- RabbitMQ
- Configuring AMQP Resources with Spring
- **Spring's AmqpTemplate**
- Sending Messages
- Receiving Messages

© Copyright 2014 Pivotal. All rights reserved.

Spring's AmqpTemplate

- *AmqpTemplate*
  - interface that defines the main operations, such as sending and receiving messages.
- *RabbitTemplate*
  - currently the only implementation of the *AmqpTemplate*
  - Specifically depends on the RabbitMQ Java-client.

© Copyright 2014 Pivotal. All rights reserved.

Spring's AmqpTemplate

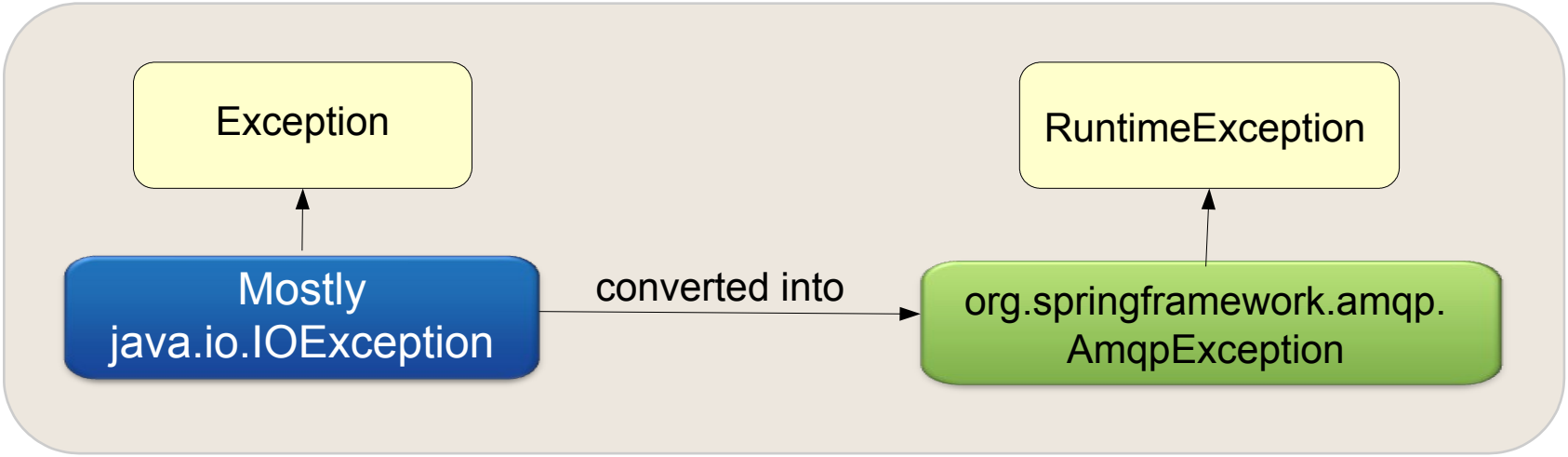
- The template simplifies usage of the API
  - Reduces boilerplate code
  - Manages resources transparently
  - Converts checked exceptions to runtime equivalents
  - Provides convenience methods and callbacks

NOTE: The *JmsTemplate* (used with JMS) has an almost identical API to the *AmqpTemplate* – they offer similar abstractions over different products

© Copyright 2014 Pivotal. All rights reserved.

Exception Handling

- Many exceptions in AMQP are checked by default
- *AmqpTemplate* converts checked exceptions to runtime equivalents



© Copyright 2014 Pivotal. All rights reserved.

RabbitTemplate Configuration

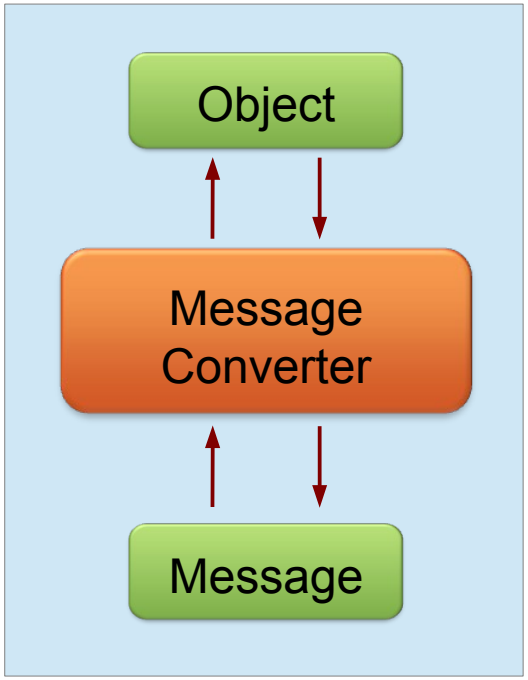
- Must provide reference to *ConnectionFactory*
  - via either constructor or setter injection
- Optionally provide delegates to handle some of the work
  - *MessageConverter* (next slide)

```
<bean id="rabbitTemplate"
        class="org.springframework.amqp.rabbit.core.RabbitTemplate">
  <property name="connectionFactory" ref="rabbitConnectionFactory"/>
  <property name="messageConverter" ref="jsonConverter" />
</bean>
```


© Copyright 2014 Pivotal. All rights reserved.

MessageConverter

- The *AmqpTemplate* uses a *MessageConverter* to convert between objects and messages
  - You only send and receive objects
- The default *SimpleMessageConverter* handles basic types
  - *String* to *TextMessage*
  - *byte[]* to *BytesMessage*
  - *Serializable* to *ObjectMessage*



NOTE: It is possible to implement custom converters by implementing the *MessageConverter* interface

© Copyright 2014 Pivotal. All rights reserved.

JSON MessageConverter

- JSON is a common message payload
- *JsonMessageConverter* is one implementation that is available

```
<bean class="org.springframework.amqp.rabbit.core.RabbitTemplate">
   <property name="connectionFactory" ref="rabbitConnectionFactory"/>
   <property name="messageConverter" ref="jsonConverter" />
</bean>

<bean id="jsonConverter"
        class="org.springframework.
                        amqp.support.converter.JsonMessageConverter">
</bean>
```

© Copyright 2014 Pivotal. All rights reserved.

Marshalling MessageConverter

Send and receive XML messages

```
<bean id="rabbitTemplate"
        class="org.springframework.amqp.rabbit.core.RabbitTemplate">
   <property name="connectionFactory" ref="rabbitConnectionFactory"/>
   <property name="messageConverter" ref="messageConverter"/>
</bean>

<oxm:jaxb2-marshaller id="marshaller" >
   <oxm:class-to-be-bound name="com.mypackage.Order"/>
</oxm:jaxb2-marshaller>

<bean id="messageConverter"
        class="org.springframework.amqp
                    support.converter.MarshallingMessageConverter" >
  <property name="marshaller" ref="marshaller" />
  <property name="unmarshaller" ref="marshaller" />
</bean>
```

Class annotated with *@XmlRootElement*

© Copyright 2014 Pivotal. All rights reserved.

CachingConnectionFactory

- For convenience, a *CachingConnectionFactory* can also be created using the rabbit namespace
  - Note that the default cache size is 1, but can easily be overridden using the *channel-cache-size* attribute

```
<rabbit:connection-factory id="rabbitConnectionFactory"
                           channel-cache-size="10"
                           addresses="localhost:5672"
                            virtual-host="/"
                           username="guest"
                           password="guest" />
```

© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to AMQP
- RabbitMQ
- Configuring AMQP Resources with Spring
- Spring's AmqpTemplate
- **Sending Messages**
- Receiving Messages

© Copyright 2014 Pivotal. All rights reserved.

Sending Messages

- The template provides options
  - One line methods that leverage the template's MessageConverter
  - Callback-accepting methods that reveal more of the AMQP Java-client API
- Use the simplest option for the task at hand

© Copyright 2014 Pivotal. All rights reserved.

Sending POJO

- A message can be sent in one single line

```
public class OrderManagerImpl implements OrderManager {
    @Autowired
    private RabbitTemplate rabbitTemplate;
    private final String ORDERS_EXCHANGE = "Orders";

    public void placeOrder(Order order) {
        String routingKey = order.getSourceRegion();

        // Use message converter
        rabbitTemplate.convertAndSend
                (ORDERS_EXCHANGE, routingKey, order);
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Synchronous Request-Reply

- RPC pattern using messaging
  - Convenience *convertSendAndReceive()* method deserializes reply message payload into object

```
…
String orderId = "12345";
OrderDetailsRequest req = new OrderDetailsRequest(orderId);

String exchange = "order-services";
String routingKey = "get-order-details-req.1_0";
Order order = (Order)template.convertSendAndReceive
                              (exchange, routingKey, req);
…
```


© Copyright 2014 Pivotal. All rights reserved.

Topics in this Session

- Introduction to AMQP
- RabbitMQ
- Configuring AMQP Resources with Spring
- Spring's AmqpTemplate
- Sending Messages
- **Receiving Messages**

© Copyright 2014 Pivotal. All rights reserved.

Synchronous Message Reception

- *AmqpTemplate* can also receive messages, but methods are blocking (with optional timeout)
  - *receive()*
  - *receive(String queueName)*
- The *MessageConverter* can be leveraged for message reception as well

```
Object someSerializable1 =
                    template.receiveAndConvert();

Object someSerializable2 =
          template.receiveAndConvert(someQueue);
```


© Copyright 2014 Pivotal. All rights reserved.

The AMQP MessageListener

- AMQP Java Client API defines this interface for *asynchronous* reception of messages
  - Or use *ChannelAwareMessageListener*

```
import org.springframework.amqp.core.Message;
import org.springframework.amqp.
            rabbit.core.ChannelAwareMessageListener;
import com.rabbitmq.client.Channel;

public class GetOrderDetails
             implements ChannelAwareMessageListener {

    public void onMessage(Message msg, Channel channel) {
        // Process message…
    }
}
```


© Copyright 2014 Pivotal. All rights reserved.

Spring's MessageListener Containers

- JEE Containers support *message-driven* beans
  - Implement an identical JMS *MessageListener* interface

- Spring provides a lightweight listener container
  - *SimpleMessageListenerContainer*
    - Uses AMQP Java client API
    - Defaults to one concurrent consumer
    - Supports transactions if input channel is transactional
- Advanced options to control consumers, scheduling, timeouts ...

© Copyright 2014 Pivotal. All rights reserved.

Defining a plain AMQP Message Listener

- Define listeners using *rabbit:listener* elements

```
<rabbit:listener-container
   connection-factory="rabbitConnectionFactory" />
  <rabbit:listener queue-names="quote"
                   ref="quoteRequestAmqpEndpoint" />
</rabbit:listener-container>
```

- Listener needs to implement *MessageListener*
  - or *ChannelAwareMessageListener*
- *rabbit:listener-container* is configurable
  - concurrency, acknowledgement mode, transaction manager and more

© Copyright 2014 Pivotal. All rights reserved.

Spring's Message-Driven POJO

- Spring also allows you to specify a plain Java object that can serve as a listener
  - *MessageConverter* provides parameter
  - Any return value sent to response-exchange after conversion

```
public class OrderService {
    public OrderConfirmation order(Order o) {}
}
```

1 2 3

```
<rabbit:listener
        ref="orderService"
        method="order"
        queue-names="queue.orders"
        response-exchange="queue.confirmation" />
```

1 2 3

© Copyright 2014 Pivotal. All rights reserved.

Messaging: Pros and Cons

- Advantages
  - Resilience, guaranteed delivery
  - Asynchronous support
  - Application freed from messaging concerns
  - Interoperable – languages, environments
- Disadvantages
  - Requires additional third-party software
  - More complex to use – but not with AmqpTemplate!

© Copyright 2014 Pivotal. All rights reserved.

Lab

Sending and Receiving Messages in a Spring Environment

© Copyright 2014 Pivotal. All rights reserved.