Introducing Spring Batch

Dave Syer, SpringSource

Overall presentation goal

An introduction to Spring Batch and how it is being used in projects today.

Agenda

- **Introduction: batch patterns and typical use cases**
- Spring Batch overview: modeling the batch domain
- Case Studies
- Spring Batch domain detail and code samples
- Roadmap and product development process

Batch Processing Patterns

- Close of business processing
  - Order processing
  - Business reporting
  - Account reconciliation
- Import/export handling
  - Instrument/position import
  - Trade/allocation export
- Large-scale output jobs
  - Loyalty scheme emails
  - Financial statements

Batch: Business Scenarios

- Commit batch process periodically
- Concurrent batch processing: parallel processing of a jobs
- Manual or scheduled restart after failure
- Partial processing: skip records (e.g., on rollback)
- Sequential processing of dependent steps
- Staged, enterprise message-driven processing
- Whole-batch transaction for simple data models or small batch size
- Massively parallel batch processing

Java Batch – A Missing Enterprise Capability in the Market

- Batch jobs are part of most IT projects and currently no commercial or open source framework provides a robust, enterprise-scale solution/framework
- Batch processing is an Application Style for data processing pipelines (e.g. payment and settlement systems)
- The lack of a standard architecture has led many projects to create their own custom architecture at significant development and maintenance costs

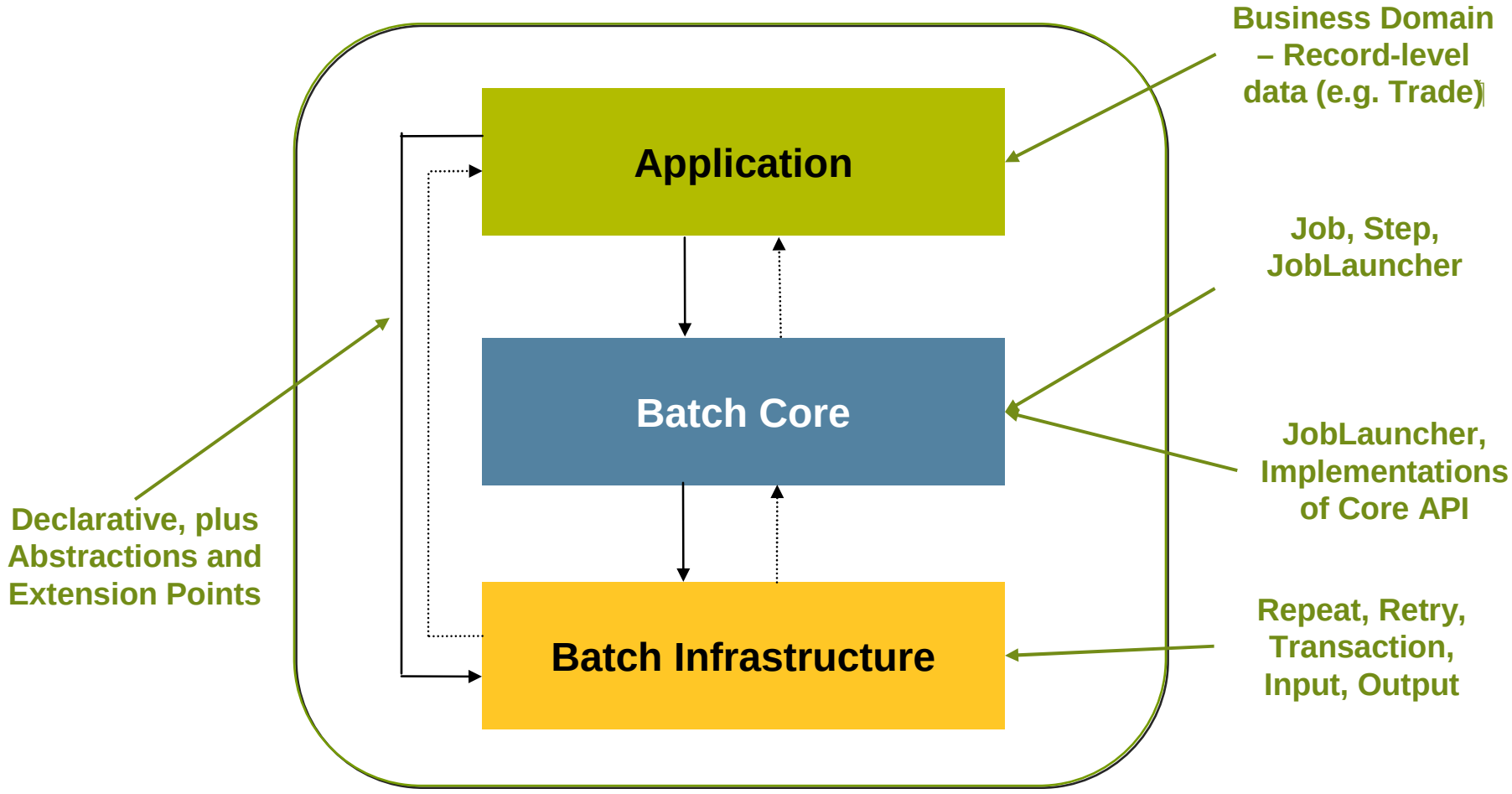
Agenda

- Introduction: batch patterns and typical use cases
- **Spring Batch overview: modeling the batch domain**
- Case Studies
- Spring Batch domain detail and code samples
- Roadmap and product development process

Spring Batch Concepts

- Infrastructure: low-level concepts
  - Input
  - Output
  - Retry
  - Repeat
- Core: boundaries and management information
  - Job
  - Step
- Execution: environment for launching and monitoring jobs

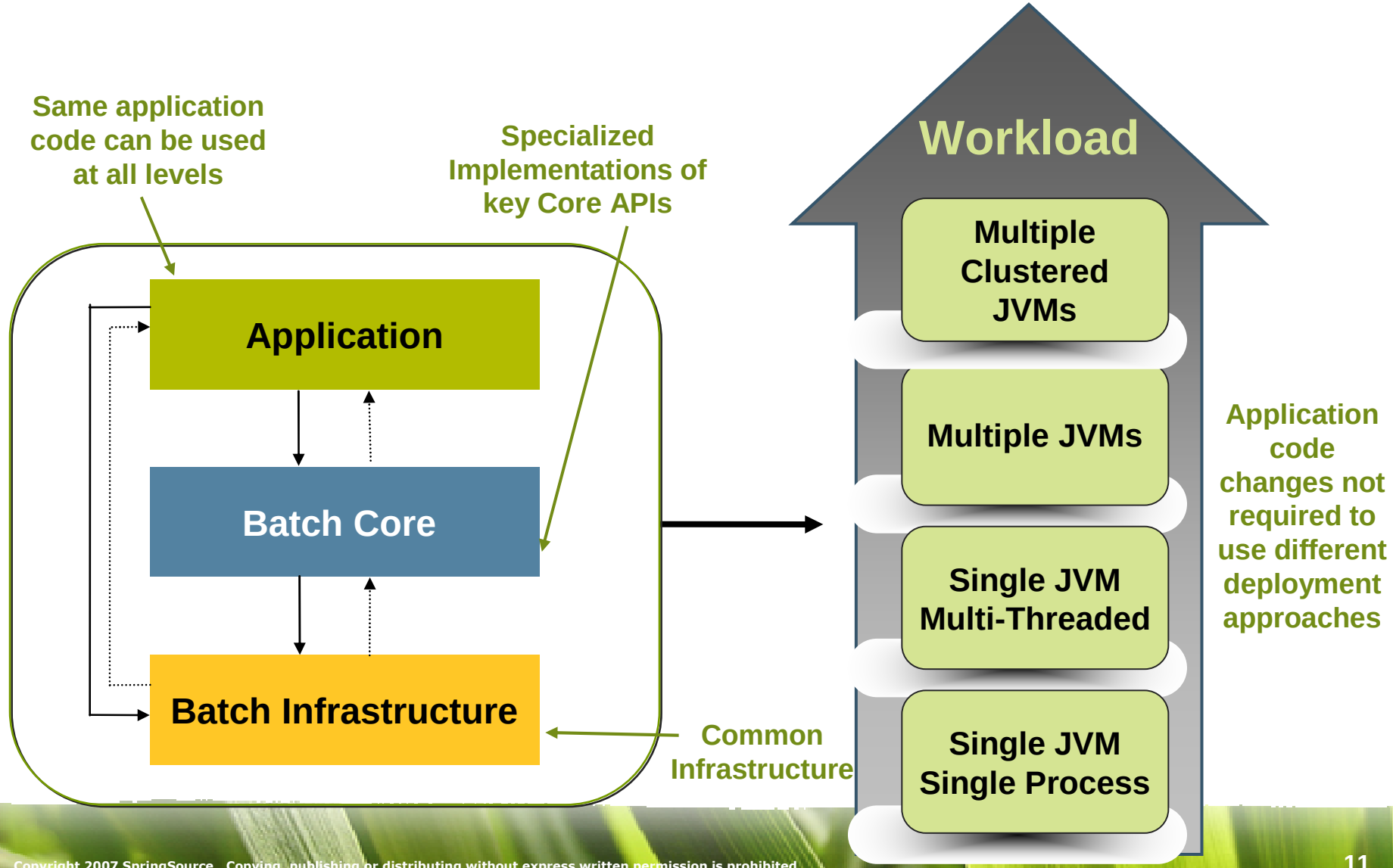
Spring Batch: Layered Architecture



Spring Batch: Features

- Support for multiple file formats
  - fixed length, delimited, XML…
- Automatic retry after failure
- Job control language for monitoring and operations
  - start, stop, suspend, cancel
- Execution status and statistics during a run and after completion
- Multiple ways to launch a batch job
  - http, Unix script, incoming message, etc.
- Ability to run concurrently with OLTP systems
- Ability to use multiple transaction resources
- Support core batch services
  - logging, resource management, restart, skip, etc.

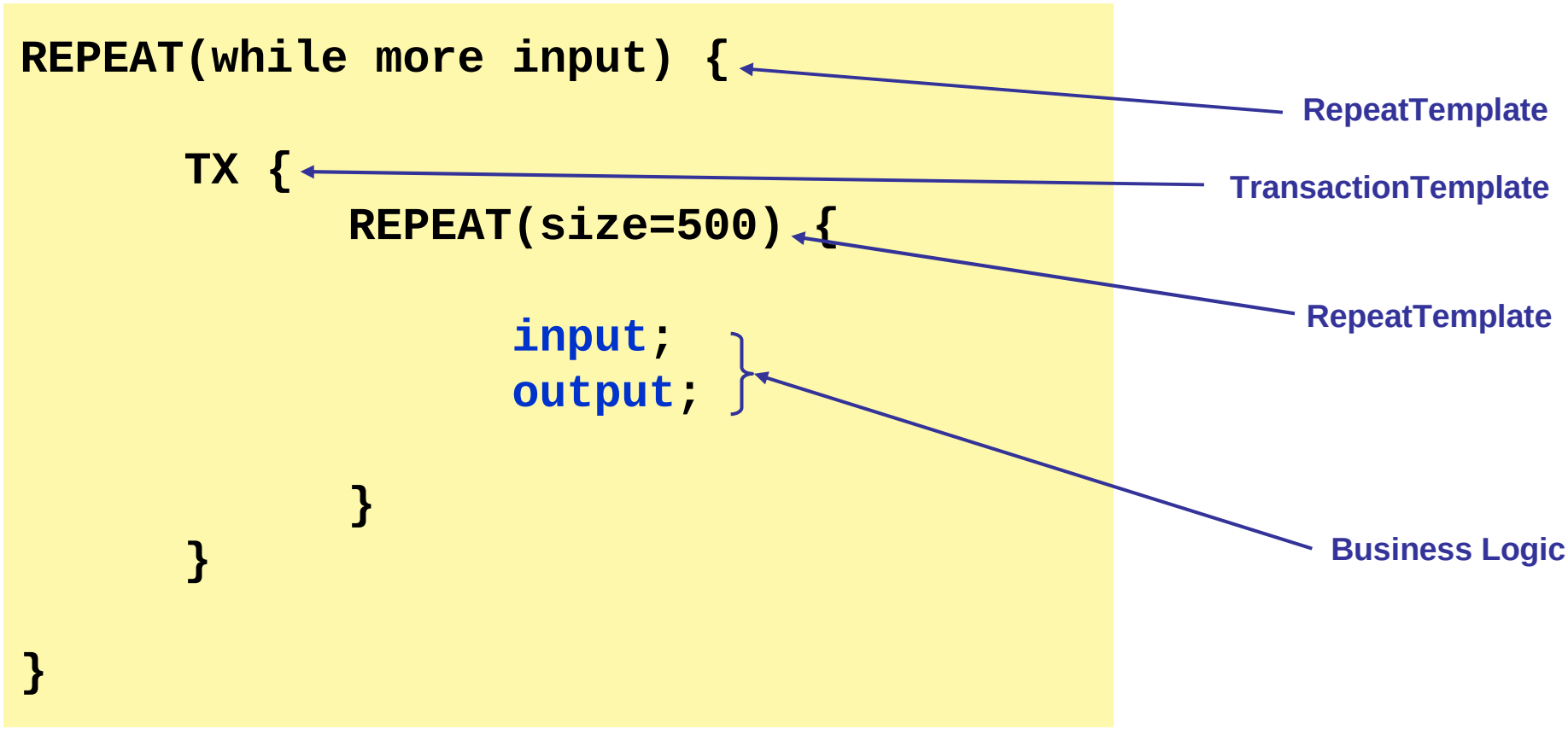
Spring Batch: Approaches to Scale



Batch Processing Style

- What is a batch?
  - Start and end
  - Read from input – write to output
  - Datasets too large to read and process all items at once
  - Optimize by committing periodically: chunk
- Pseudo code
  - Analyse behaviour and responsibilities
  - Communicate design

Item-Oriented Pseudo Code



Add Batch Domain Concepts


```
JOB {
   STEP {
        REPEAT(while more input) {
              TX {
                    REPEAT(size=500) {
                          input;
                          output;
                    }
              }
        }
   }
}
```

Agenda

- Introduction: batch patterns and typical use cases
- Spring Batch overview: modeling the batch domain
- **Case Studies**
- Spring Batch domain detail and code samples
- Roadmap and product development process

Case Studies

- Large European Healthcare
- Large Sports Organization
- State Government
- South American Healthcare Provider
- Leading Investment Bank

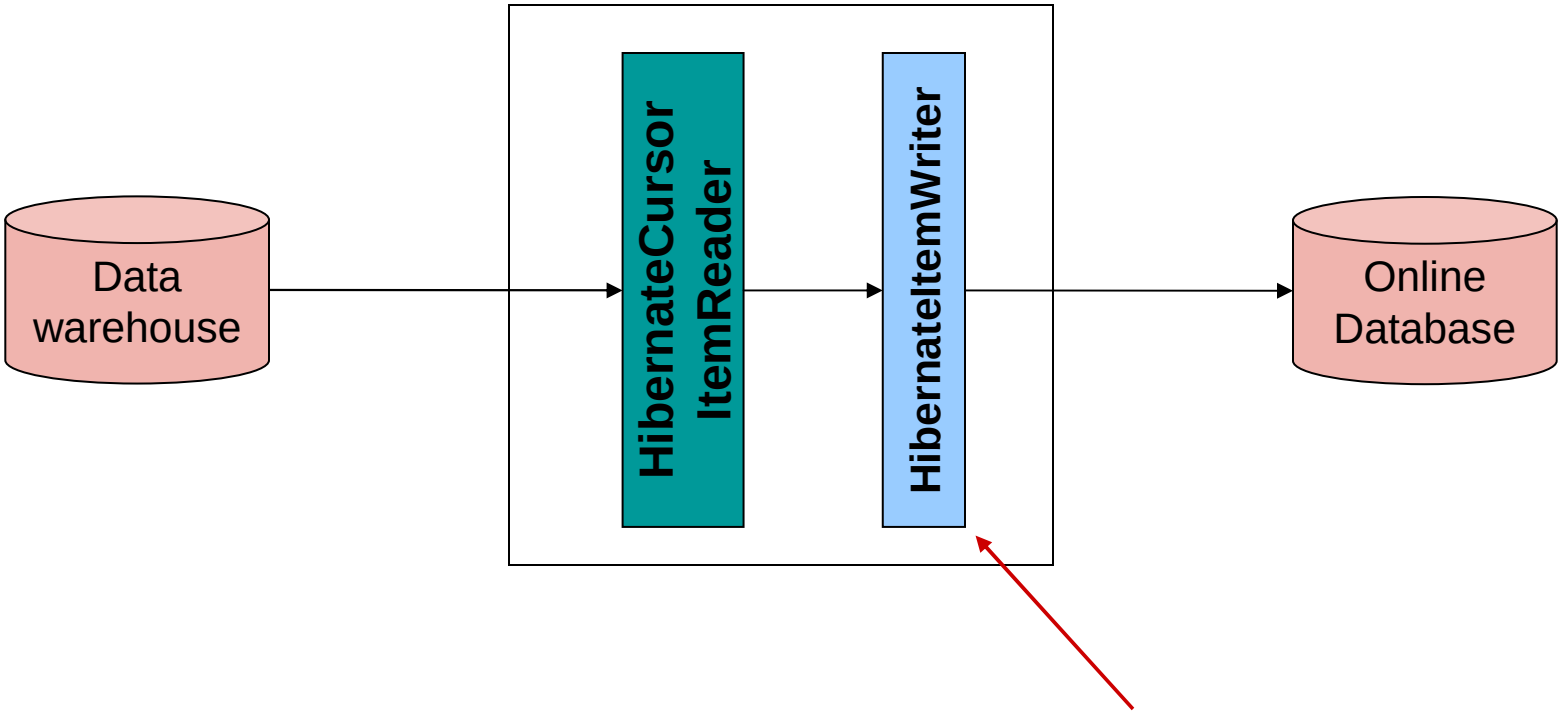
Large European Healthcare

- Background: In order to align with the corporate strategy of using Java with Spring. COBOL mainframe batch processing partially migrated to Spring Batch.
- Solution:
  - Used XML Input Sources, allowing for a batch orientated approach to XML input.
  - Both input and output from the database used Hibernate, allowing online DAOs to be reused.
- Benefit
  - Allowed for a homogeneous development environment with online developers using Spring and Web Flow.

Large European Healthcare: Example Jobs

- **Typical batch**:
  - pulling 60,000 rows out of data-warehouse and performing different operations.
  - offline, weekly
  - duration: 20 minutes
- **Integration Layer**
  - Reading in third party XML-Data, processing and inserting data into database.
  - Also vice versa, reading from database and writing to XML-Files (2-way-integration).
  - About 100 rows each execution.
  - parallel to online operations, every 15 minutes
  - duration: < 1 minute
- **Notification E-Mail**
  - Batch that sends reminder e-mails (approx. 500 / night)
  - offline, nightly
  - duration: 1 minute

Typical Job: Updates for Online Database



Re-using business logic implementation from online system

Hibernate Input

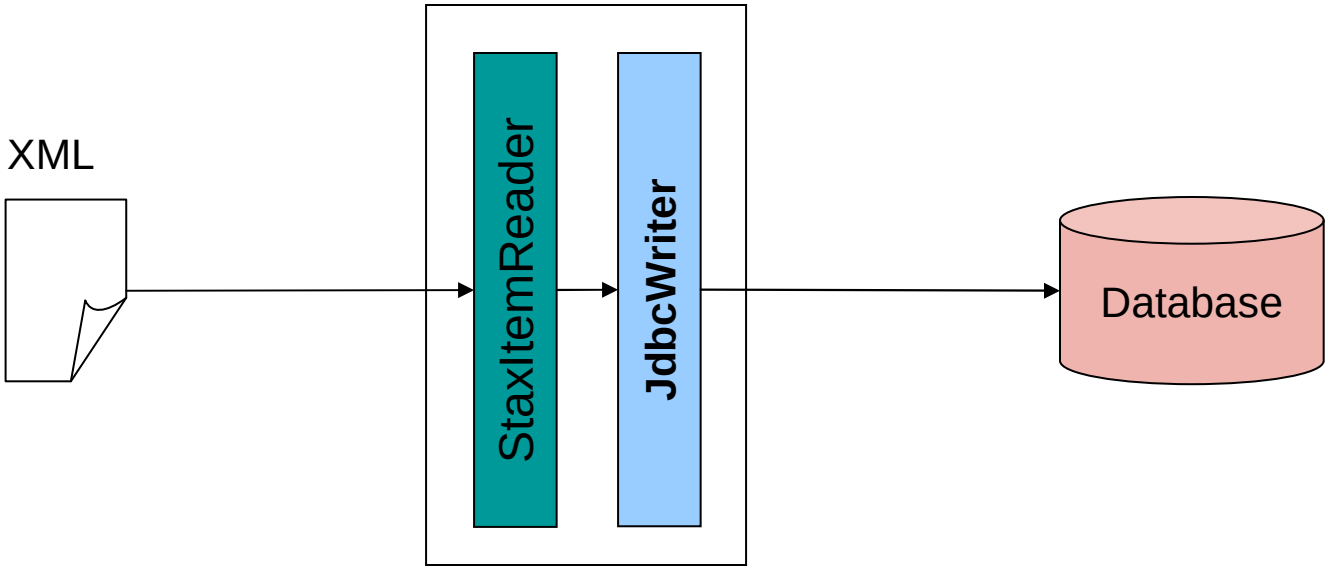
- Cursor Driven
- Stateless or standard Session.

```
<bean id="hibernateInputSource"
    class="....item.database.HibernateCursorInputSource"
    >
    <property name="queryString" value="from
    CustomerCredit" />
    <property name="sessionFactory"
    ref="sessionFactory" />
</bean>
```

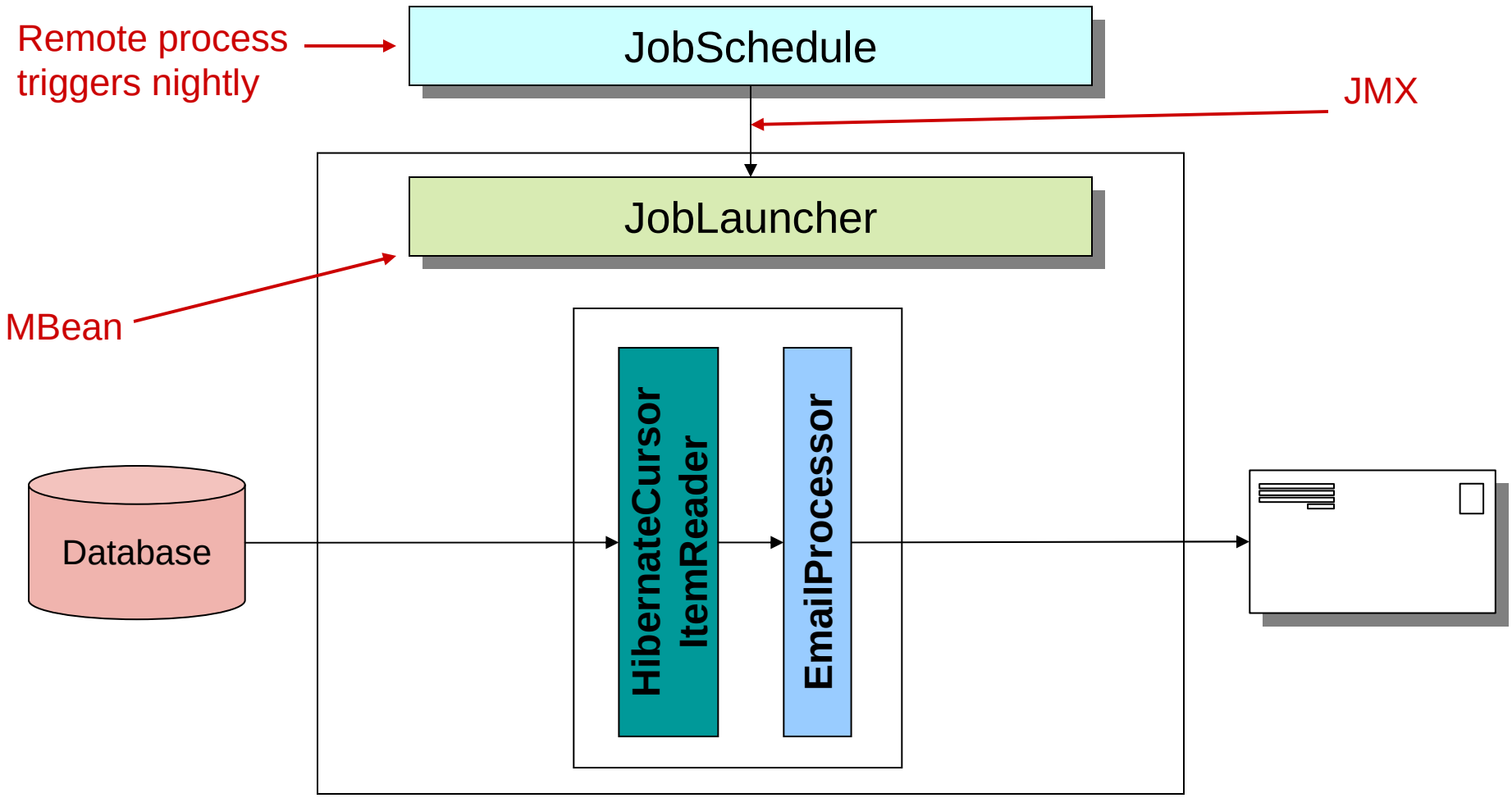
Hibernate Output

- Allows for Online DAOs to be reused by item processor
  - Flush per item
  - Inefficient
  - Allows for easy identification of failed items.
- Flush per chunk
  - Greatest efficiency
  - Failed records cannot be determined (requires search, e.g. binary)

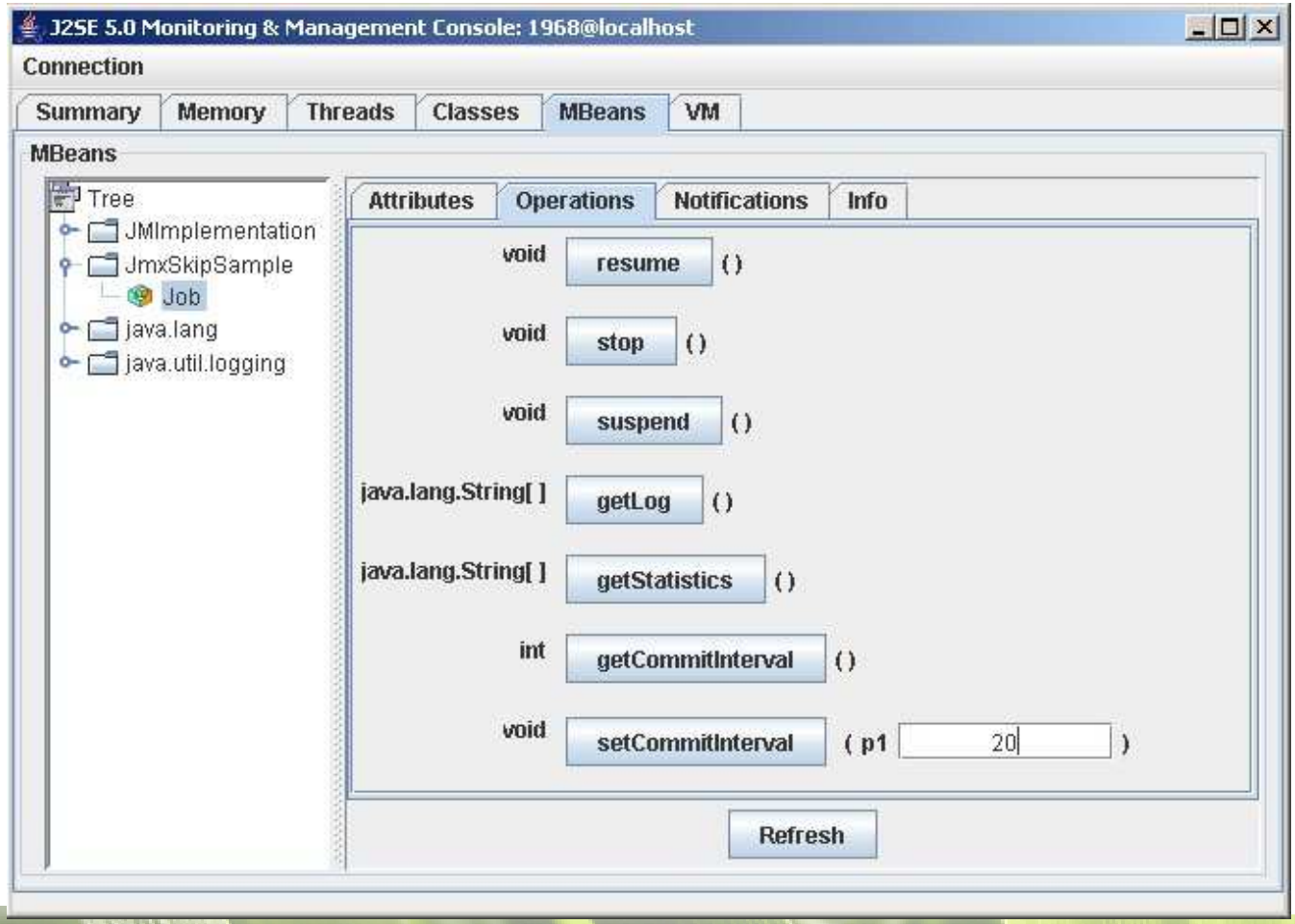
Integration Layer: Support Job



Notification E-Mails



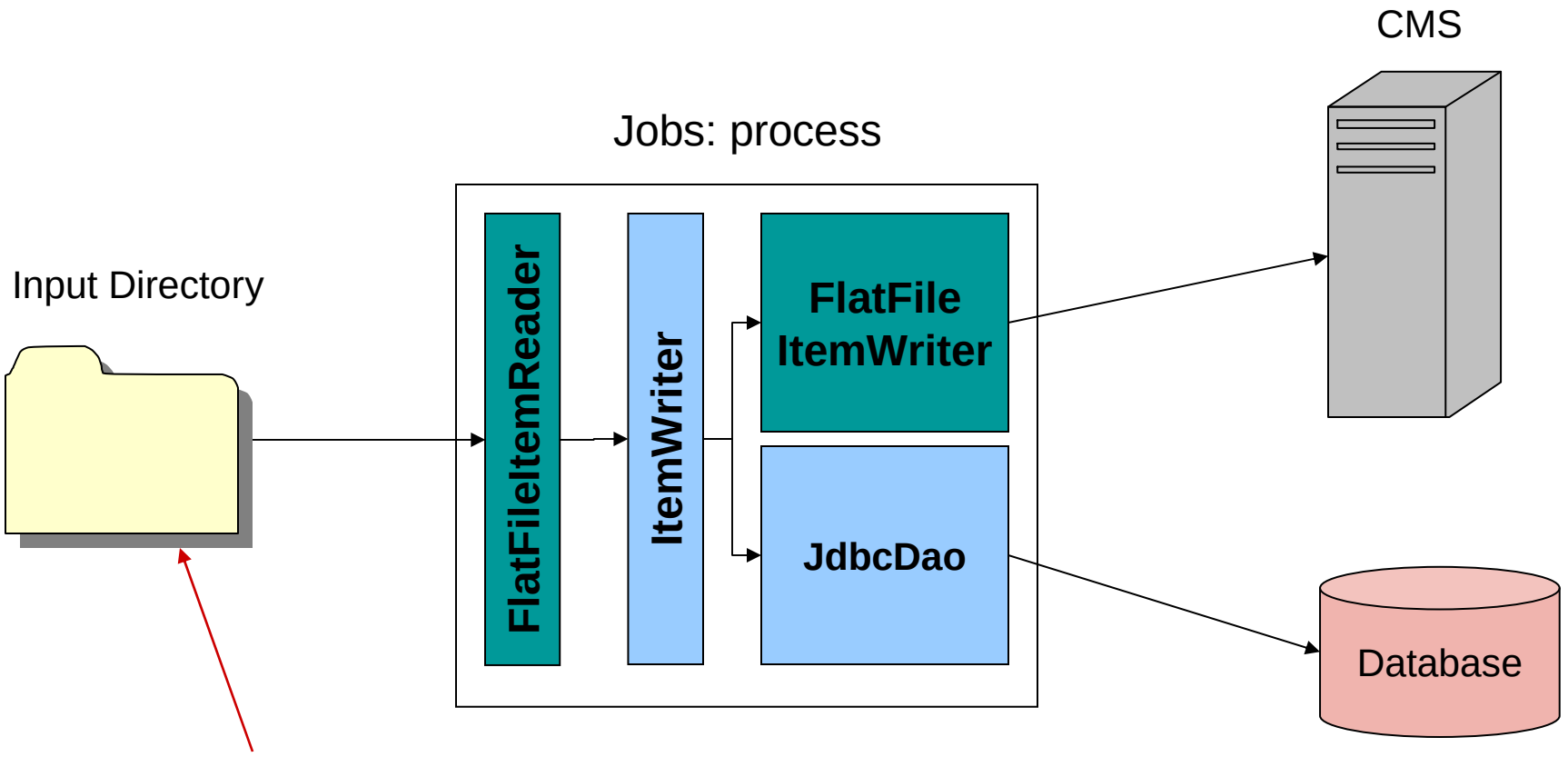
Spring Batch: JMX Management and Operations



Large Sports Organization

- Background: Scores need to be updated live as games happen for real-time tracking by users. An AJAX solution required uploaded files for major events.
- Solution:
  - File reading only required configuration, allowing for quicker development.
  - Modular approach allowed for jobs to be run faster
    - No Meta Data (status)
    - Job launched every 5 seconds.
- Benefit:
  - Allowed for simple, lightweight solution to a non-traditional batch problem.

Filesystem Directory Polling



Directory polled every 5 seconds for new files

In-Memory Repository

- No performance cost for storing meta-data
- Jobs are not identifiable but not restartable

```
<bean id="simpleJobRepository"
    class="...core.repository.support.SimpleJobRepository">
    <constructor-arg ref="jobDao" />
    <constructor-arg ref="stepDao" />
</bean>

<bean id="jobDao"
    class="...core.repository.support.dao.MapJobDao" />
<bean id="stepDao"
    class="...core.repository.support.dao.MapStepDao" />
```

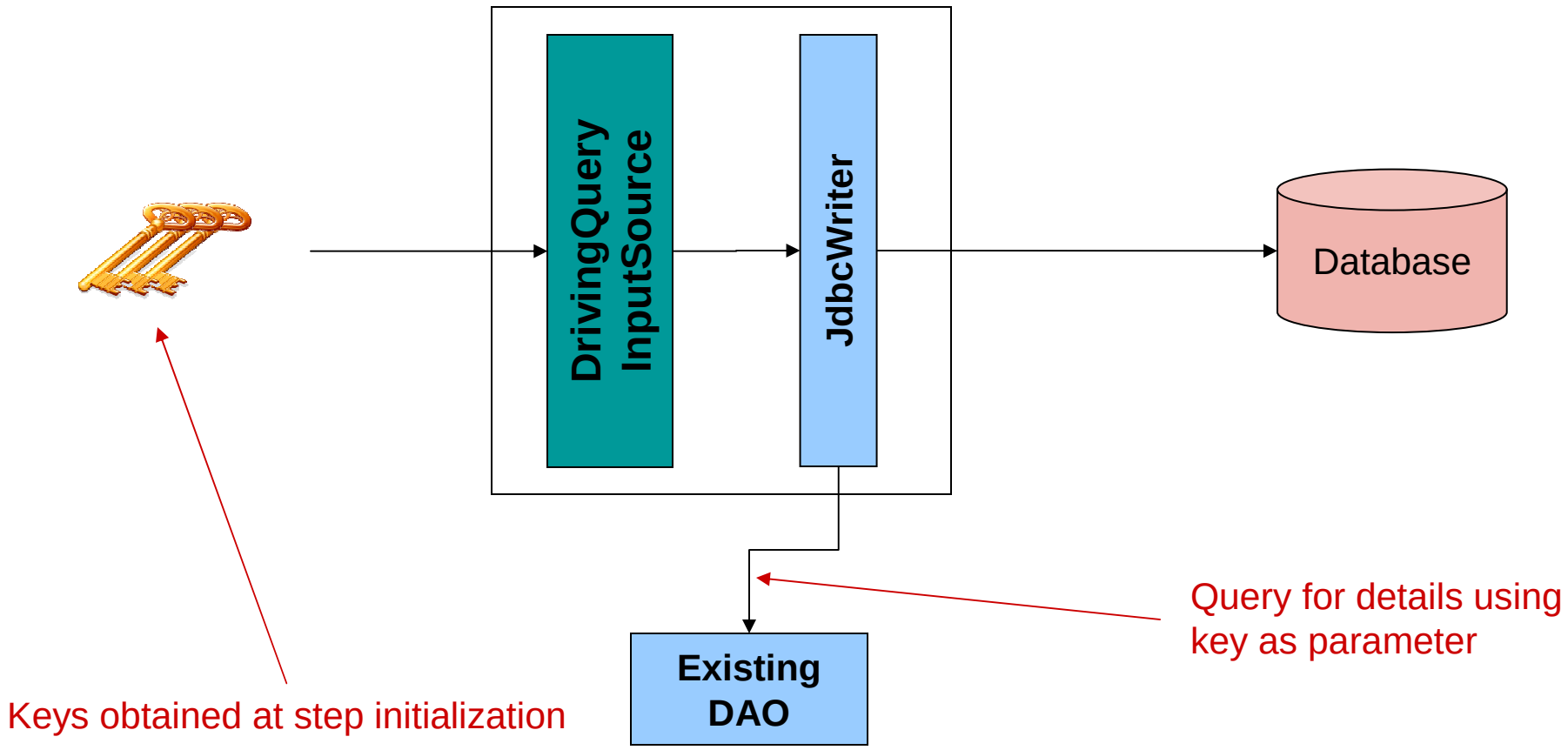
State Government

- Background: In-Progress IT Renewal project replacing mainframe batch jobs with Java in order to process unemployment claims.
- Solution:
  - Custom Item Readers for processing CopyBook and EBCDIC input
  - Custom LineTokenizer for specific Government flat file formats.
  - DrivingQuery ItemReader provided a better approach for using DB2 with online DAOs.
- Benefits
  - Batch job development is only piece ahead of schedule.

Driving Query

- Requires an initial 'driving' SQL statement that will return a list of all the unique keys to be processed.
- Each call to read() returns one key
- Allows for reuse of other DAOs to return 'details'.
- Smaller database footprint (than cursor) allowing for better concurrency with online systems, especially in database systems with pessimistic locking strategies.
- All DAOs using provided keys in a batch scenario should use PreparedStatements, to allow for caching that greatly improves performance.

Driving Query



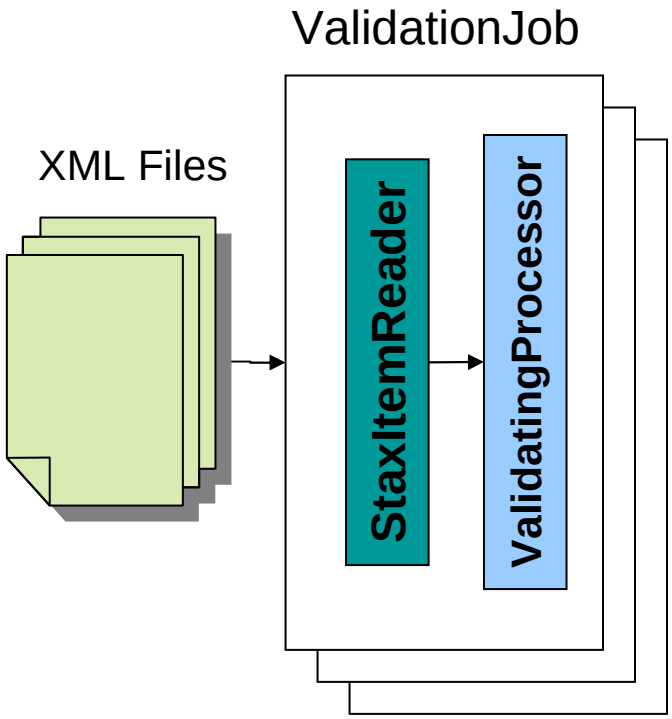
Driving Query


```
<bean id="drivingQuery"
    class="...item.database.DrivingQueryInputSource"
    <property name="keyGenerator" ref="keyGenerator" />
</bean>
<bean id="keyGenerator"
    class="...item.database.support.SingleColumnJdbcKey
    Generator">
    <constructor-arg ref="jdbcTemplate" />
    <constructor-arg ref="select ID from TRADES" />
</bean>
```

South American Healthcare Provider

- Background: Massive claims information processing from different providers. External providers send XML data and images for processing (~5K files, ~6MM claims per month). Batch process validates and processes incoming XML files into the database, however processing can only be performed at night to reduce the impact to online systems during the daytime.
- Solution:
  - Separate jobs for validation and processing. 24x7 job validates XML files as they arrive. Nightly Job inserts validated data into the database.
  - Parallel processing of files during daytime for maximum performance.
  - Used JMS Queue for sending data (domain objects) to core systems.

Xml Validation Pre-process



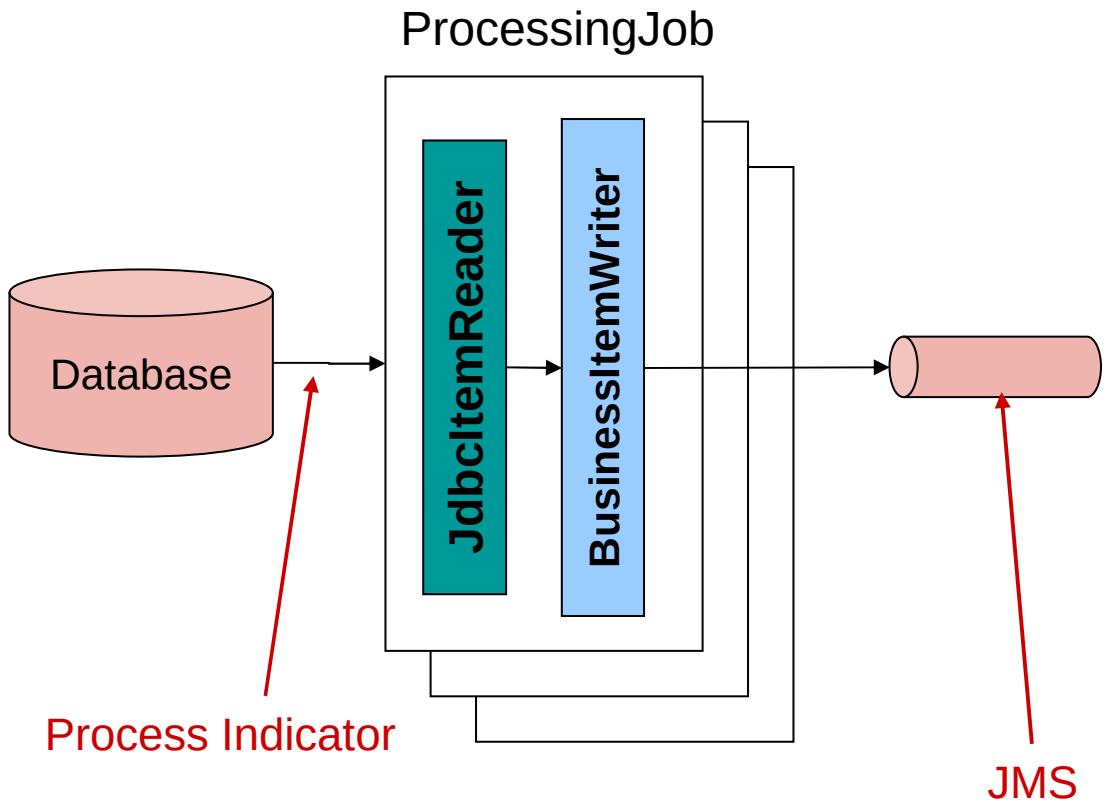
XML I/O

- Built on StAX
  - Root element representing an 'Item' must be defined.
  - Each root element is read in via a StAX parser, then deserialized to an Item
- Uses Spring-OXM to bind event fragments to 'Items'

XML I/O

```
<bean class="…item.file.support.StaxEventReaderInputSource">
   <property name="fragmentRootElementName" value="trade" />
   <property name="fragmentDeserializer">
    <bean …>
        <constructor-arg>
            <bean
   class="org.sfw.oxm.xstream.XStreamMarshaller">
                <property name="aliases" ref="aliases" />
            </bean>
        </constructor-arg>
    </bean>
   </property>
</bean>
```

Parallel Processing Case Study



Restartability and Parallel Processing

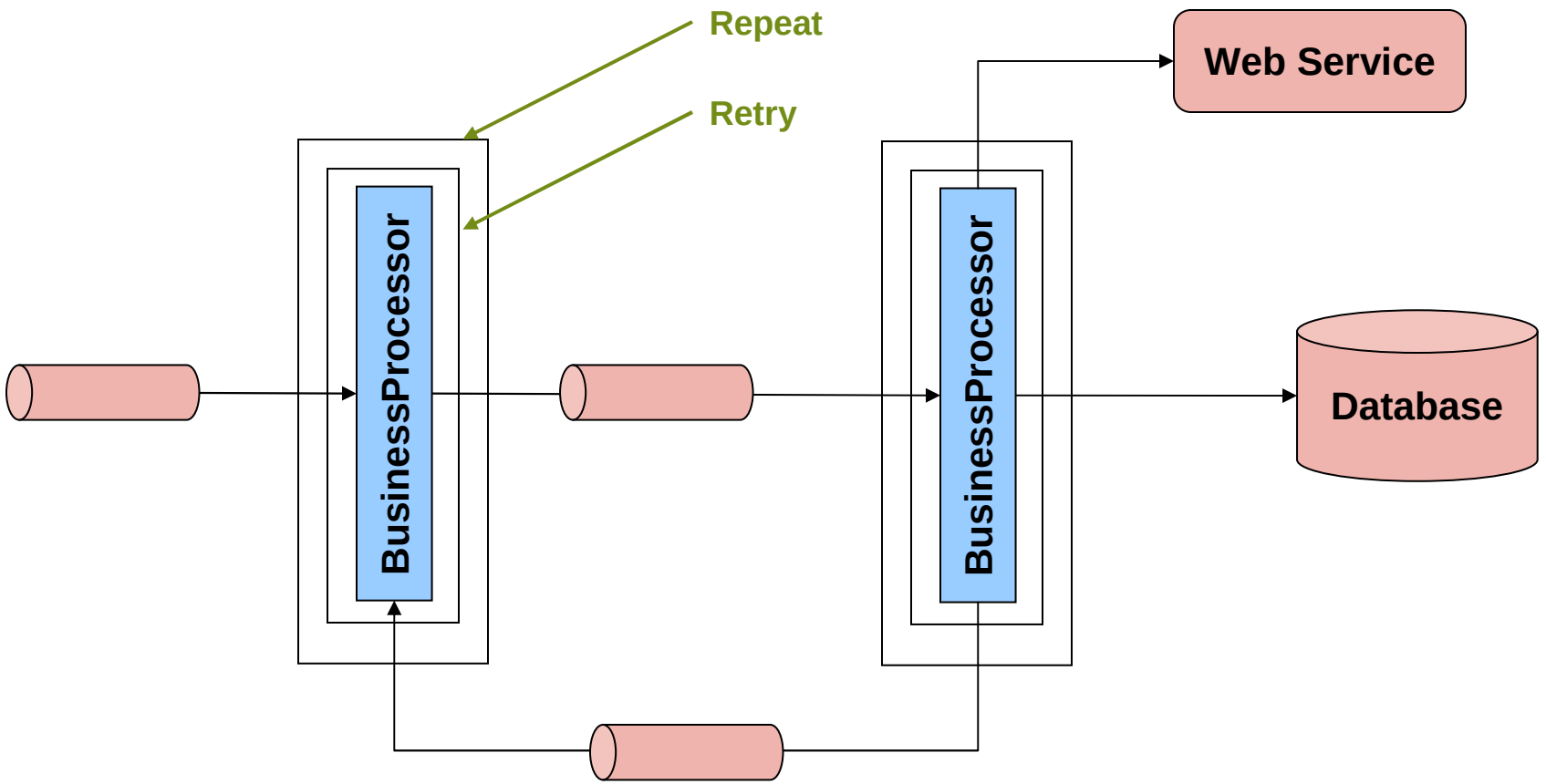
- Challenge for restartability:
  - Chunks can be processed in any order
  - Chunk 1 can fail after Chunk 100 has completed
- Single VM, multi-threaded (Spring Batch 1.0)
- Batch Pattern: **Process Indicator**
  - Add column to input records
  - Mark "finished" on chunk boundary
  - Restart: initialise to only process unfinished records

SELECT * FROM T_INPUT WHERE PROCESSED='N'

Leading Investment Bank

- Background: Large networks of message-driven pipelines processing orders, payments and other transactions.
- Solution:
  - Use RepeatTemplate to batch messages together transparently.
  - Use RetryTemplate to retry either
    - Whole transaction (batch)
    - Or an external webservice call
- Benefits:
  - Greatly increased throughput in high volume process.
  - Reduce burden on developers to understand optimisations.

Message-driven Pipeline Optimisation



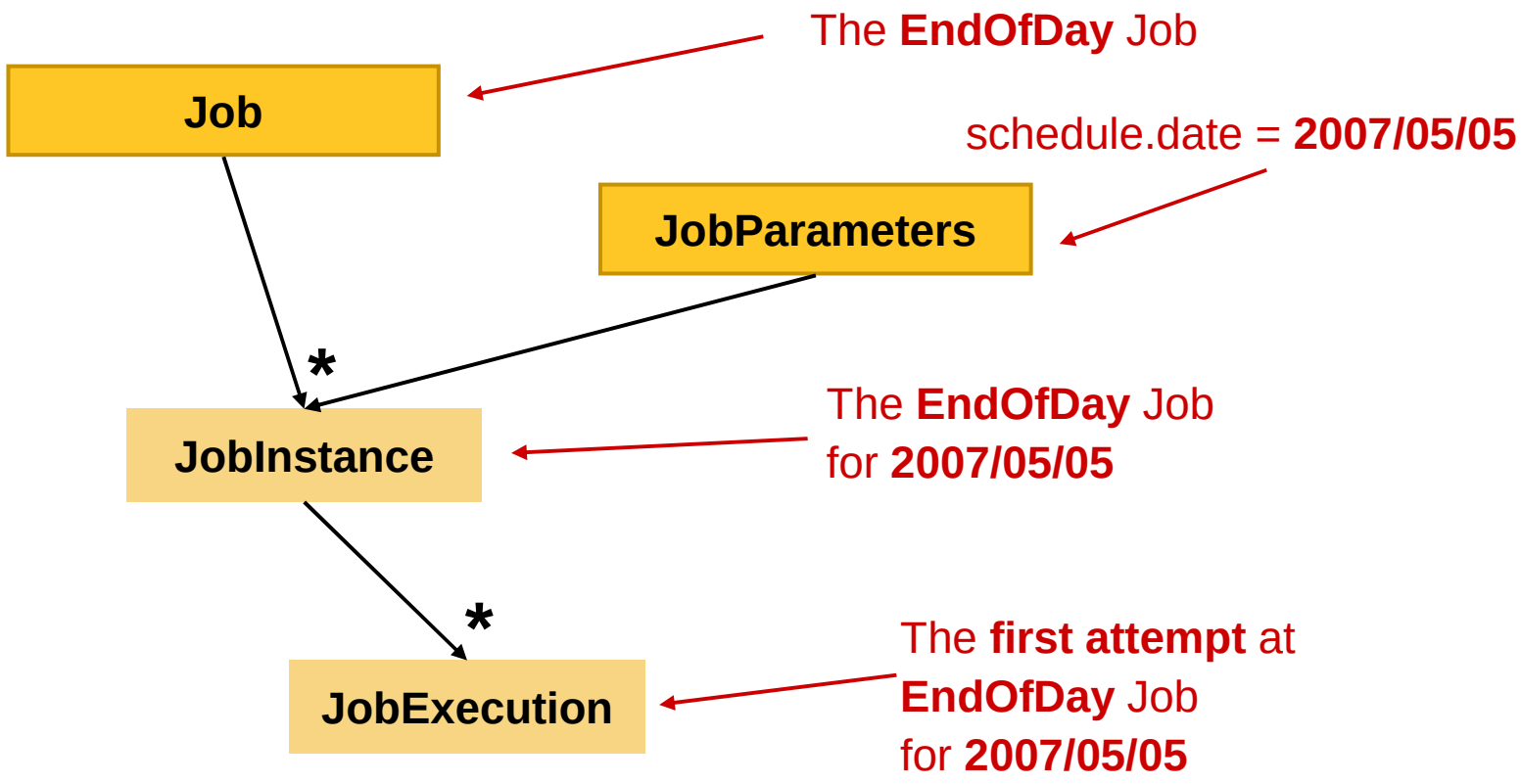
Agenda

- Introduction: batch patterns and typical use cases
- Spring Batch overview: modeling the batch domain
- Case Studies
- **Spring Batch domain detail and code samples**
- Roadmap and product development process

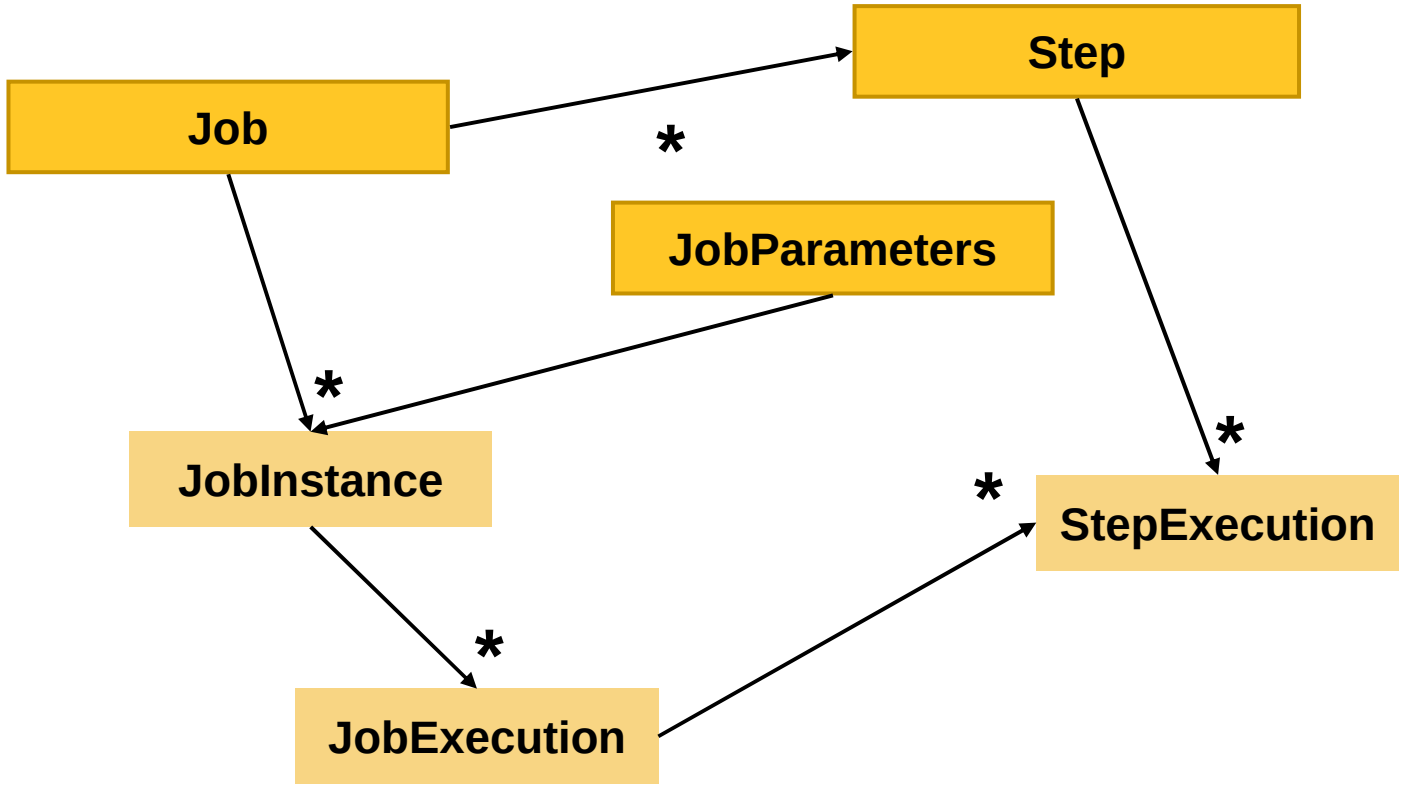
Batch Infrastructure and Batch Domain

- Batch infrastructure adds optimisation and low level abstractions
- The batch domain adds some value to a plain business process by introducing new concepts:
  - A **job** has an identity – not just a stream of bytes
  - A **job** has **steps**
  - A **job** can be restarted after a failure – a new **execution**
  - Each **execution** has a start time, stop time, status
  - The **job** has a status
  - Each **execution** can tell us how many **items** were processed, how many commits, rollbacks, skips

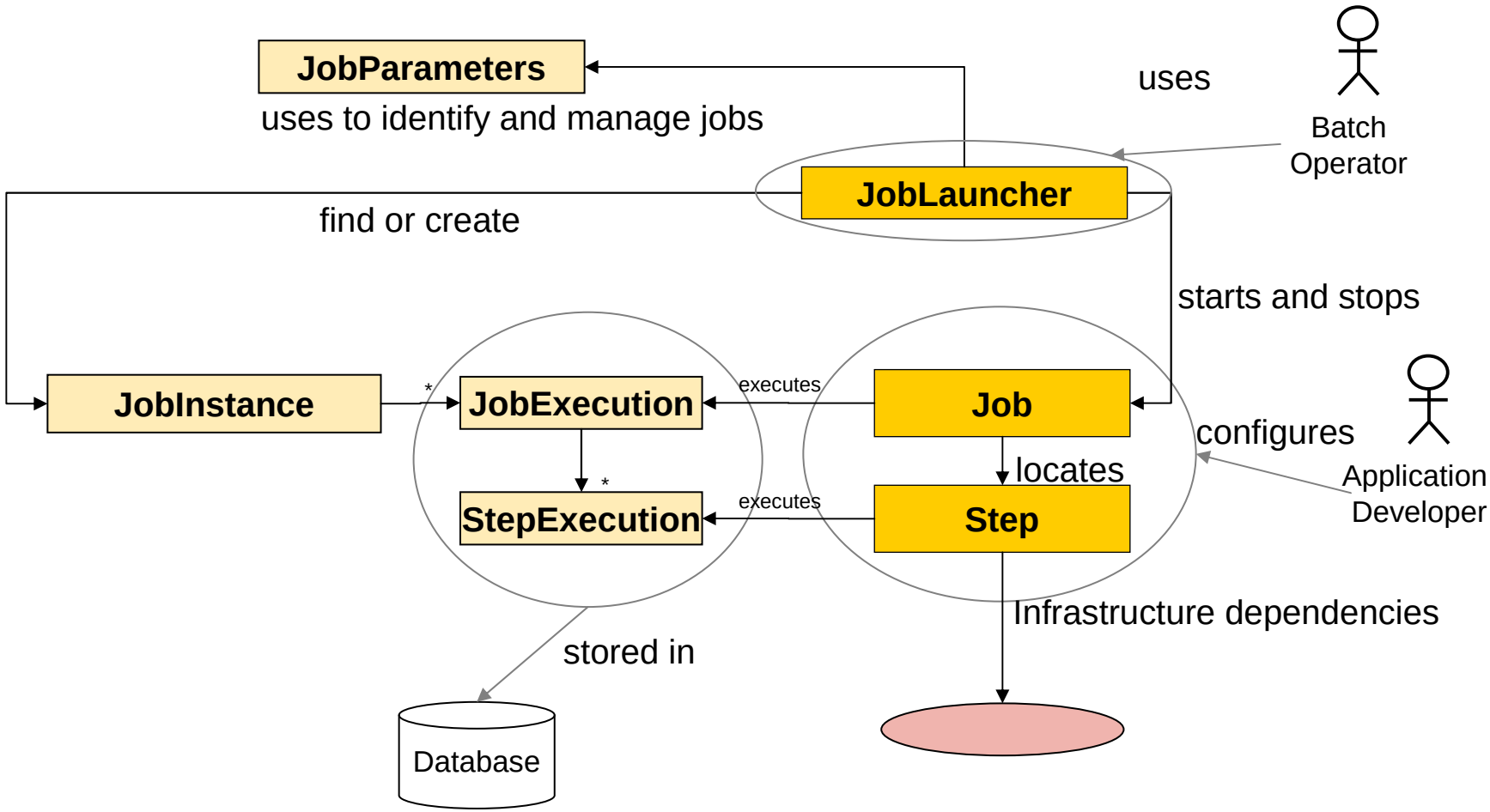
Job Configuration and Execution



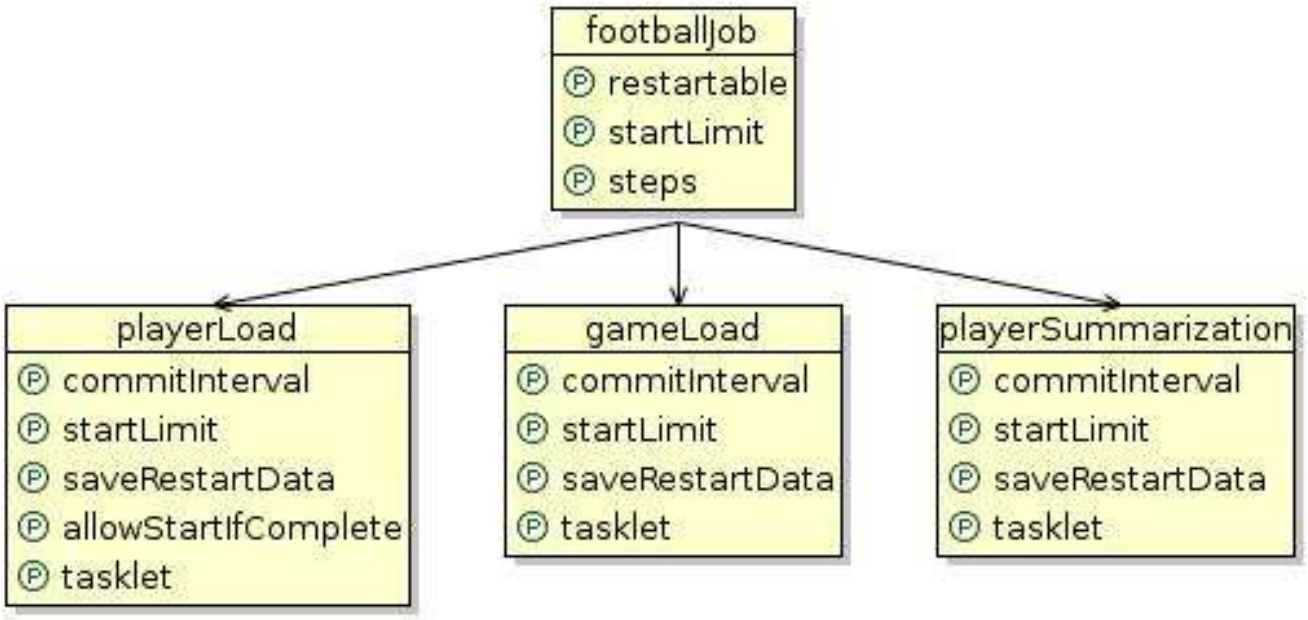
Job and Step



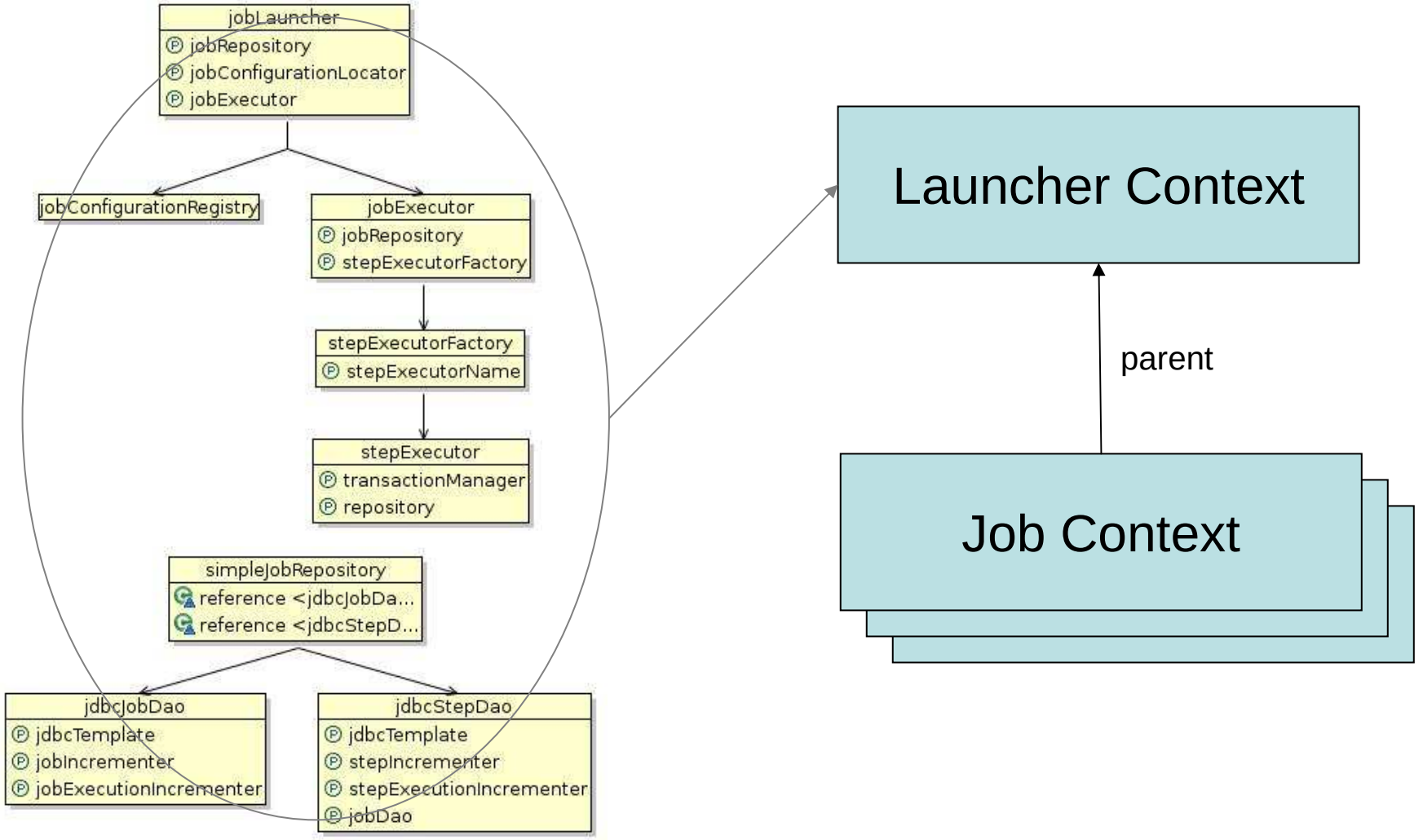
Batch Domain Diagram



Job Configuration – football job sample



Job Launcher Configuration



Demo

Sample Jobs

Job/Feature	delimited input	fixed-length input	xml input	multiline input	db driving query input	db curosr input	delimited output	fixed length output	xml output	multiline output	db output	skip	restart	quartz
fixedLengthImportJob		X									X			
multilineJob		X		X										
multi lineOrderJob	X			X				X		X				
quartzBatch														X
simpleModuleJob		X									X			
simpleSkipSample												X		
skipWithRestartSample												X	X	
sqlCursorTradeJob						X								
tradeJob	X				X		X							
xmlJob			X						X					

Agenda

- Introduction: batch patterns and typical use cases
- Spring Batch overview: modeling the batch domain
- Case Studies
- Spring Batch domain detail and code samples
- **Roadmap and product development process**

Spring Batch—Road Map

- Release 1.0
  - Infrastructure support of the development of batch execution environments
    - Repeat—batch operations together
    - Retry—retry a piece of work if there is an exception
    - File and database I/O templates
  - Simple Batch Execution implementation providing full, robust management of the batch lifecycle
    - Restart, skip, statistics, status
    - Validation/record processing
- 1.0.0.m5 released February 2008
- 1.0 final target date March 20
- Milestones already in use at roughly 40+ projects with at least 3 in production
- Future (2.0) might include…
    - Partitioned Launcher, SEDA, Grid, ESB support, and more…

Accenture and SpringSource Partnership

- Why is Accenture contributing to open source?
  - Consolidating decades worth of experience in building high-performance batch solutions
  - Driving standardization in batch processing
- Why Spring?
  - Established, active, and leading community with significant momentum
  - Logical home for a batch architecture framework as part of the Spring Portfolio
- End Goal
  - Provide a highly scalable, easy-to-use, customizable, industry-accepted Batch architecture framework

Summary

- Lack of a standard enterprise batch architecture is resulting in higher costs associated with the quality and delivery of solutions.
- Spring Batch provides a highly scalable, easy-to-use, customizable, industry-accepted batch framework collaboratively developed by Accenture and SpringSource
- Spring patterns and practices have been leveraged allowing developers to focus on business logic, while enterprise architects can customize and extend architecture concerns

Q&A