Spring Batch Bootcamp

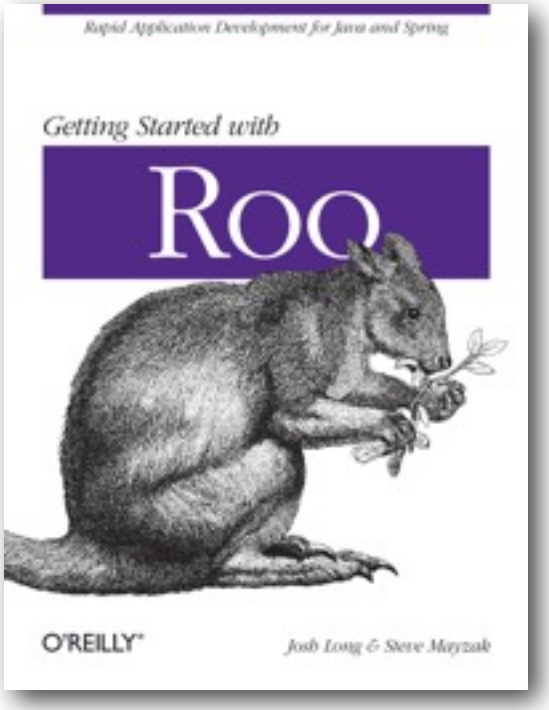
- Your host: Josh Long

  SpringSource, a division of VMware

- **Code: github.com/joshlong/spring-batch-bootcamp**
- **Deck: slideshare.net/joshlong/spring-batch-behind-the-scenes**

a *huge* amount of this came from Dr. Dave Syer (he's awesome)

About Josh Long (Spring Developer Advocate)



@starbuxman

josh.long@springsource.com

Why we're here...

ComputerWeekly.com

Article from ComputerWeekly.com 4 May 2000

The Department of Social Security has suffered from two major IT failures over the past year. What were the failures, and what action is being taken? *Tony Collins* reports.

Failure 2: Overpayment by BACS

In an unrelated incident, about 112,000 claimants of income support received double their expected payments by automated credit transfer, directly to their bank accounts, when an EDS "Autobacs" batch file was accidentally processed twice.

An irony of the problem was that, having scrapped the Debt Accounting and Management System which was designed to collect overpayment debt, the Department, a few months later, went on to accidentally overpay 112,000 Income Support claimants.

The Department has repeatedly rejected Computer Weekly's requests for a detailed explanation of what went wrong. However staff say the problem began with a bug in a DSS batch programme, which was not intercepted by an EDS "Autobacs" system that collects and reconciles the batch programmes.

Agenda

- Introduce Spring Batch
  - concepts
  - specifics
  - demos

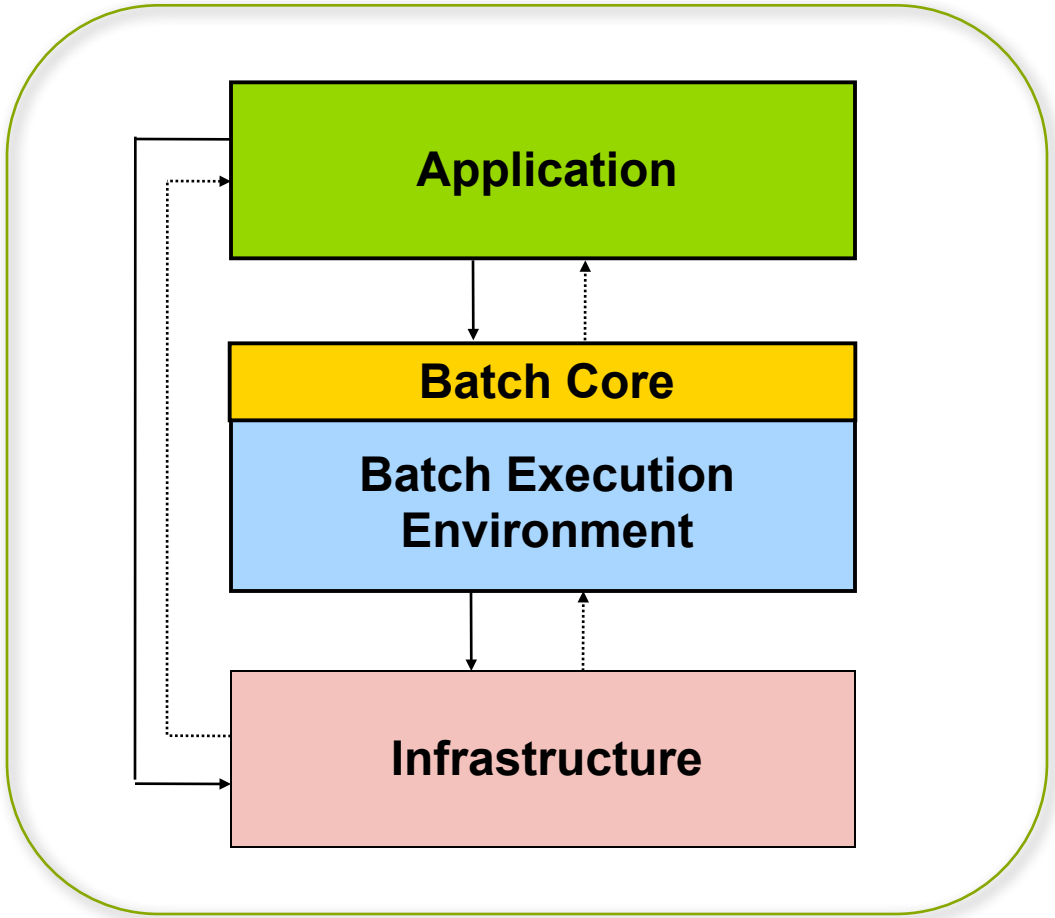
Inside Spring Batch

- Architecture and Domain Overview
- Application concerns and Getting Started
- Chunk-Oriented Processing

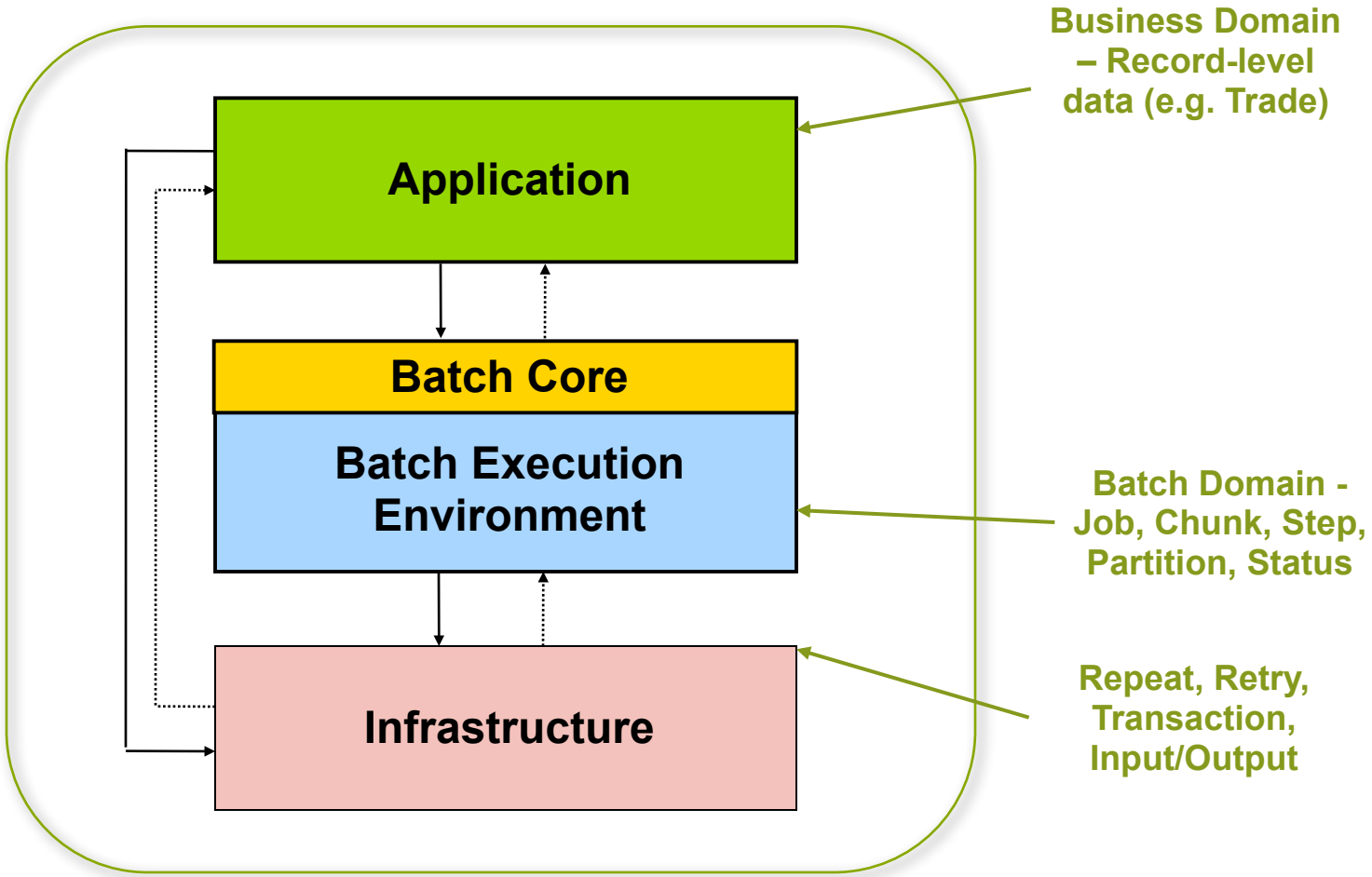
Inside Spring Batch

- Architecture and Domain Overview
- Application concerns and Getting Started
- Chunk-Oriented Processing

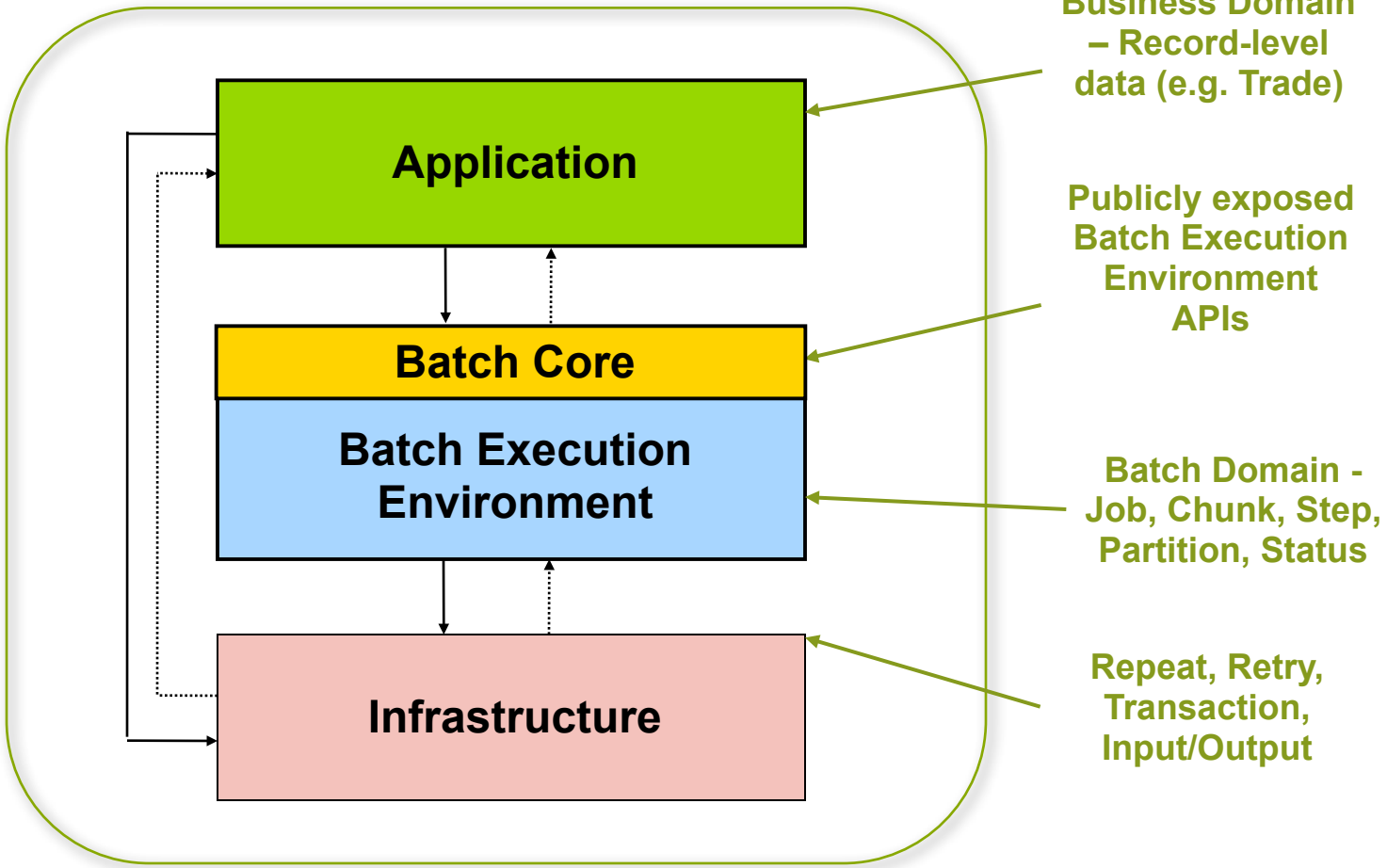
Spring Batch: Layered Architecture



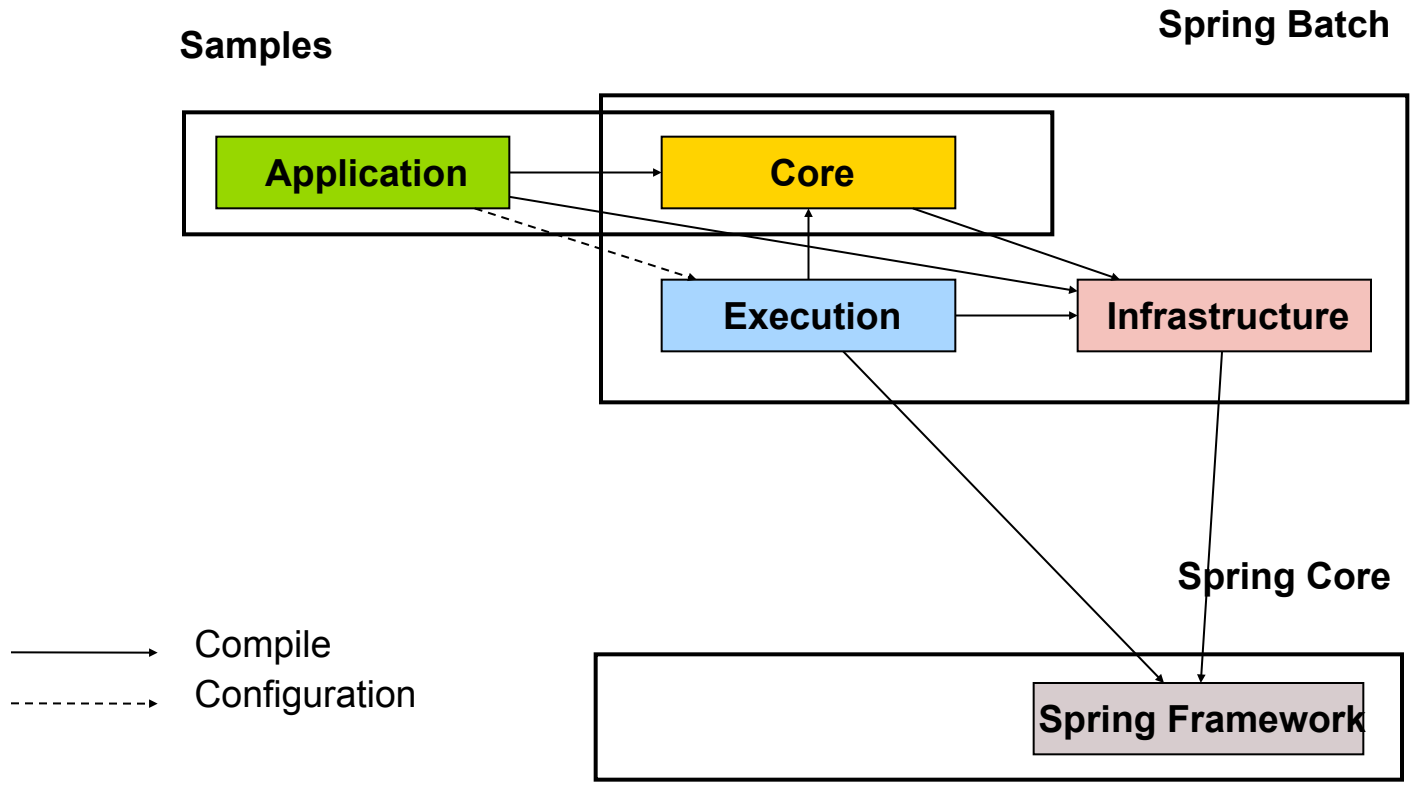
Spring Batch: Layered Architecture



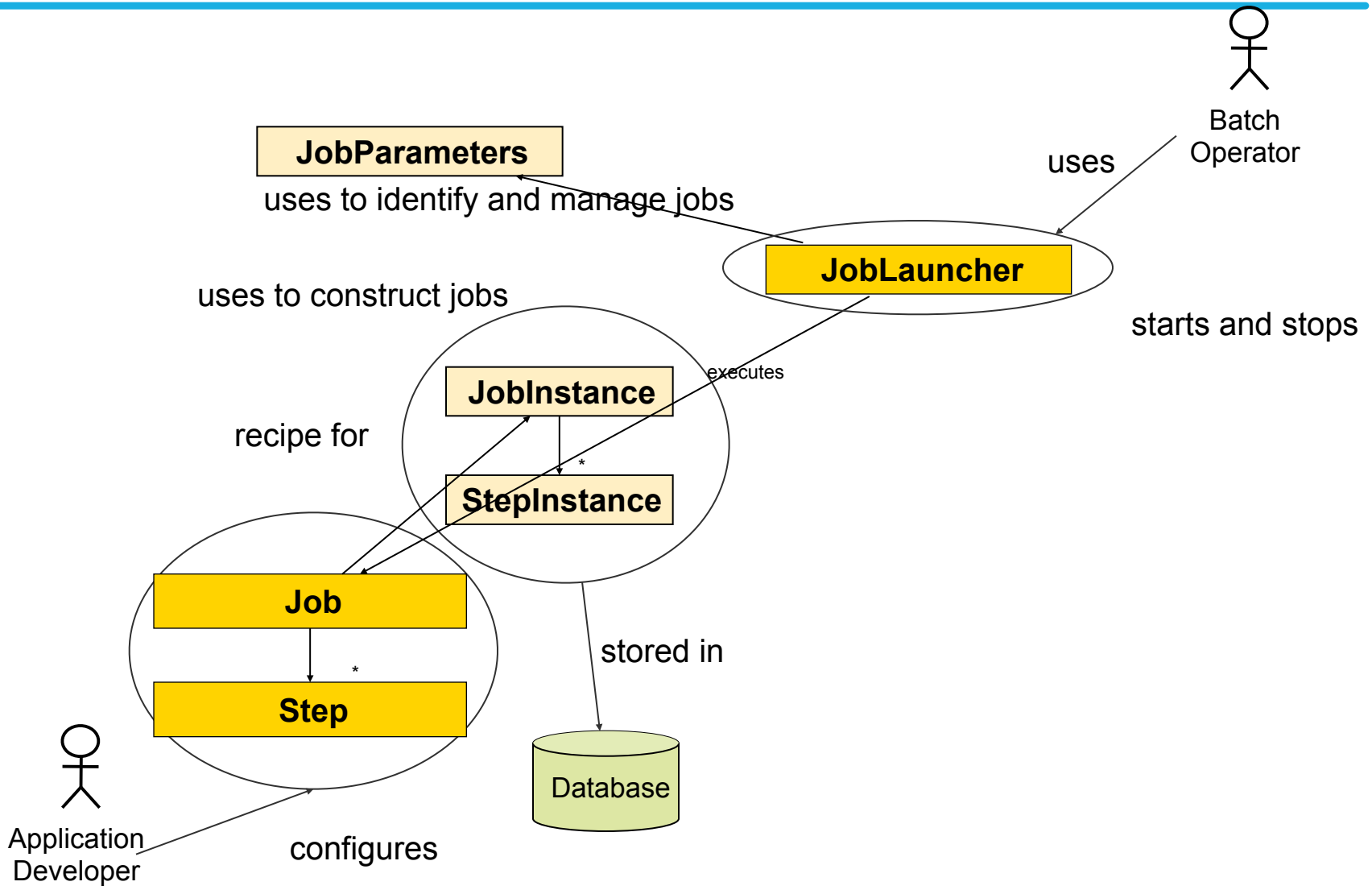
Spring Batch: Layered Architecture



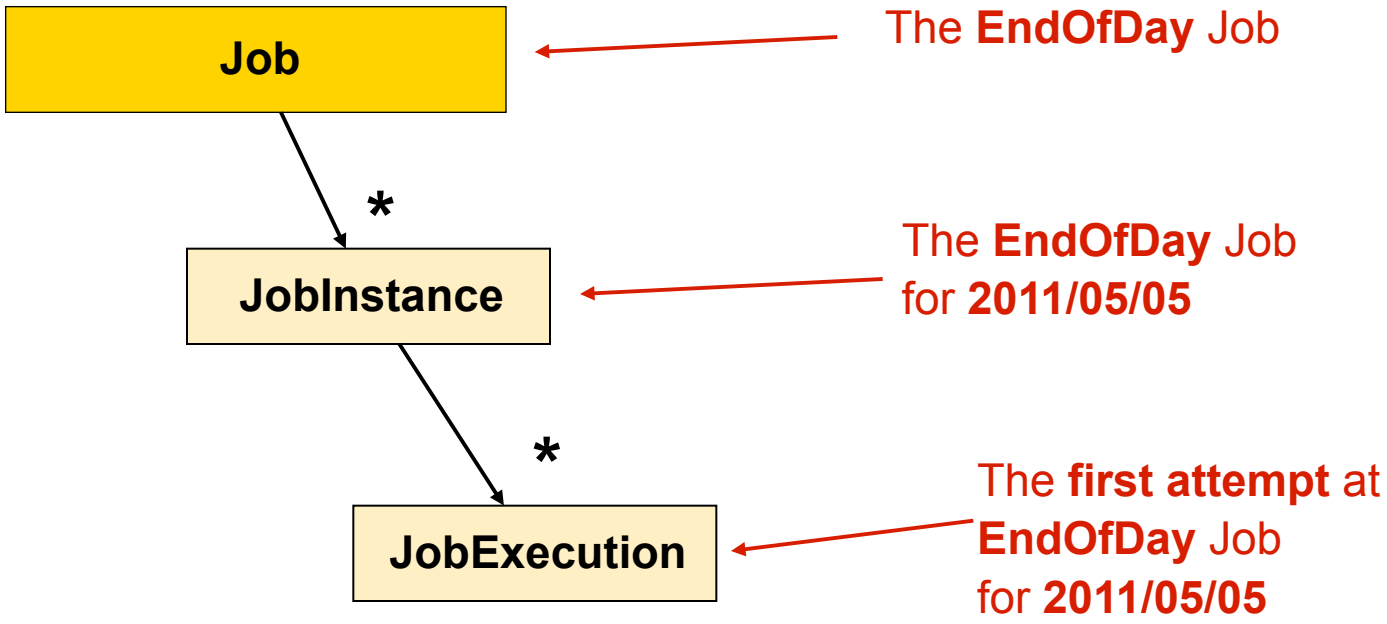
Spring Batch Dependencies



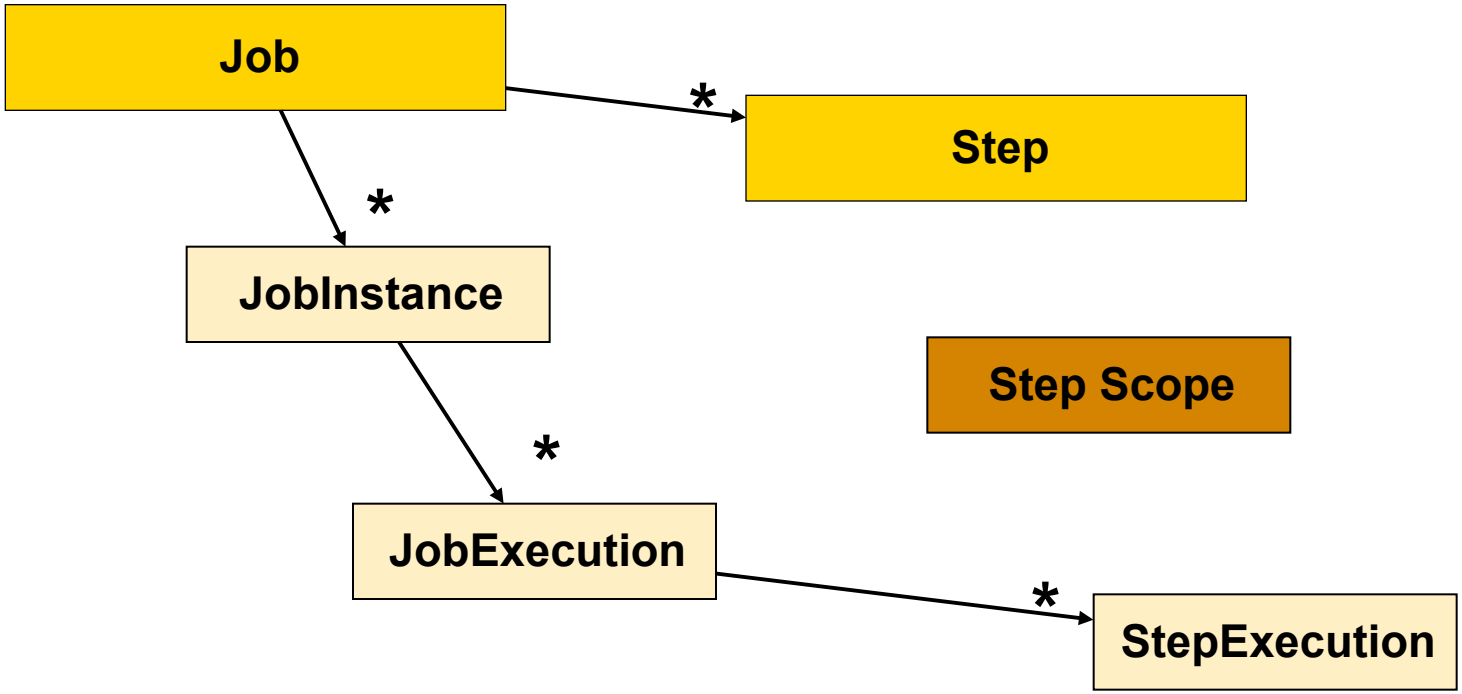
Batch Domain Diagram



Job Configuration and Execution



Job and Step



DEMO of Spring Batch Application

Inside Spring Batch

- Architecture and Domain Overview
- Application concerns and Getting Started
- Chunk-Oriented Processing

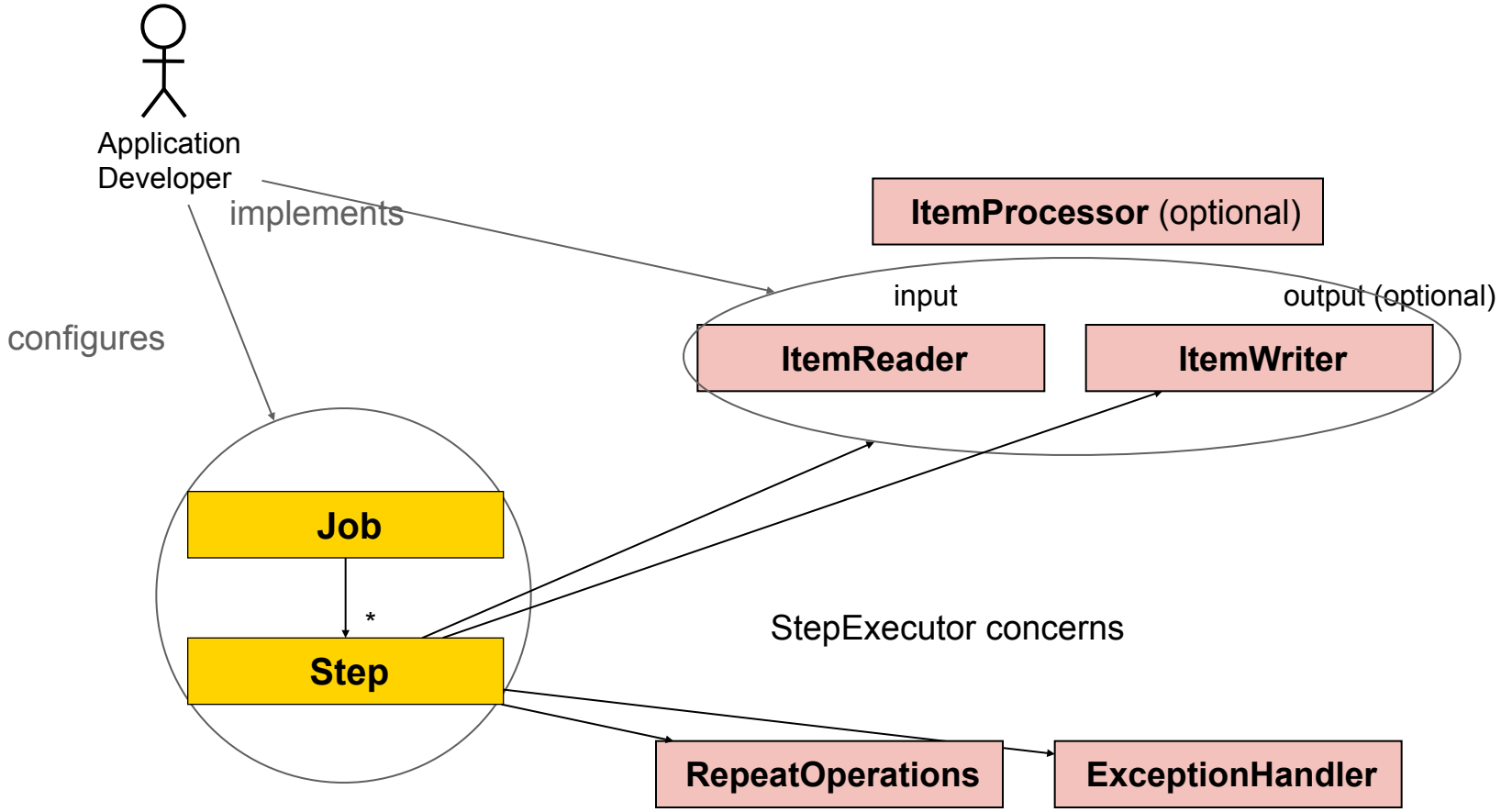
Application Concerns

- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Application Concerns

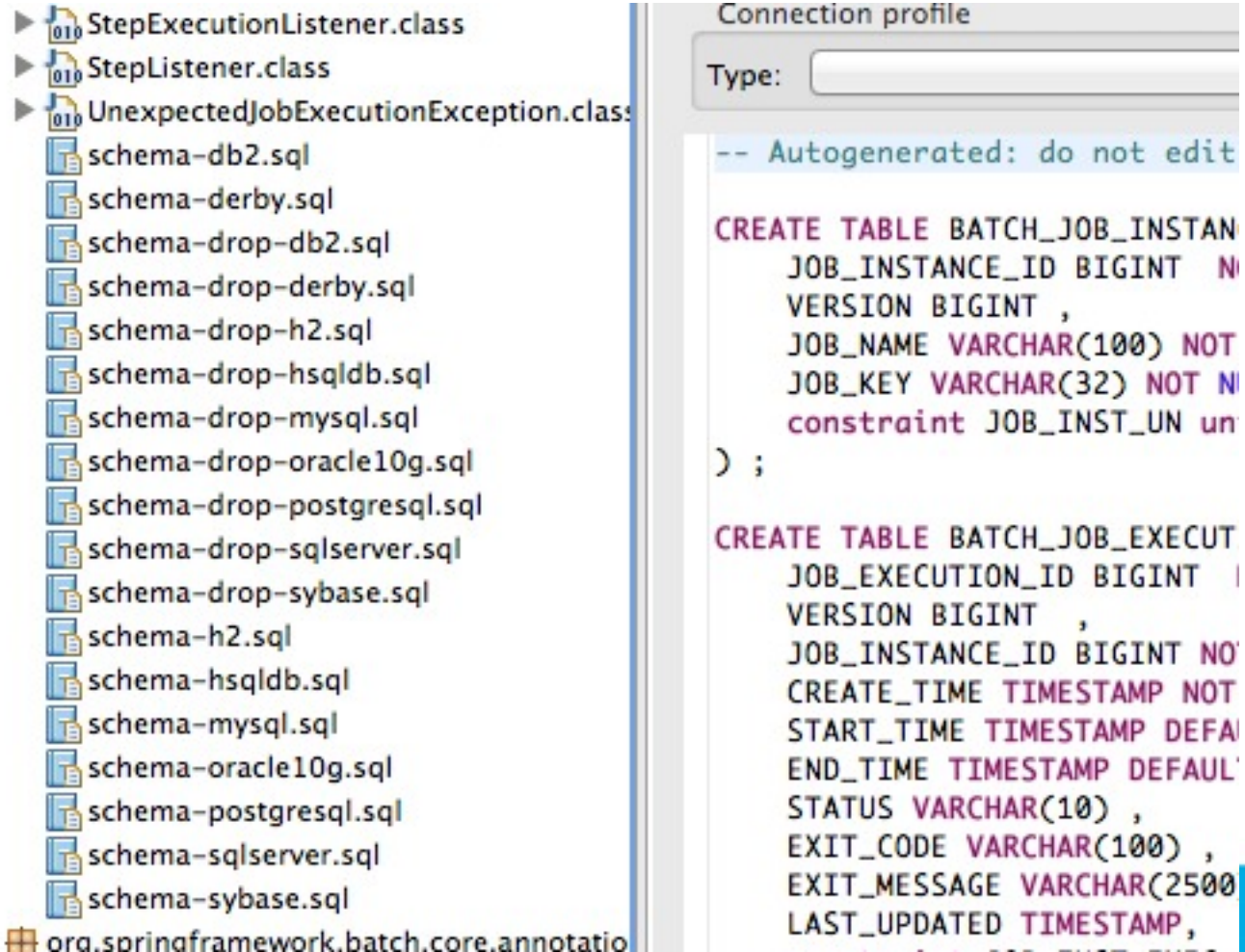
- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Getting Started



OK, So How Do I start?

- Find and install the appropriate .sql script in your database
  - they live in **org.springframework.batch.core** in **spring-batch-core.jar**



OK, So How Do I start?

```
@Inject JobLauncher launcher ;
@Inject @Qualifier("importData") Job job ;

@Schedule(cron  = "* 15 9-17 * * MON-FRI ")
public void run15MinutesPastHourDuringBusinessDays() throws Throwable {

    Resource samplesResource = new ClassPathResource("/sample/a.csv");
    String absFilePath = "file:///" + samplesResource.getFile().getAbsolutePath();

    JobParameters params = new JobParametersBuilder()
        .addString("input.file", absFilePath)
        .addDate("date", new Date())
        .toJobParameters();

    JobExecution jobExecution = jobLauncher.run(job, params);
    BatchStatus batchStatus = jobExecution.getStatus();
    while (batchStatus.isRunning()) Thread.sleep(1000);
    JobInstance jobInstance = jobExecution.getJobInstance();
}
```

OK, So How Do I start?

- Or... Deploy the Spring Batch Admin
  - good for operations types
  - good for auditing the batch jobs

DEMO of Spring Batch Admin

ItemReader

```
public interface ItemReader<T> {

        T read()  throws Exception,
                UnexpectedInputException,
                ParseException,
                NonTransientResourceException;

}
```

Returns **null** at end of dataset

delegate Exception handling to framework

Database Cursor input

- Cursor is opened over all data that will be input for a given job
- Each row in the cursor is one ‘item’
- Each call to read() will advance the ResultSet by one row, and return one item that is equivalent to one row

Database Cursor Input

ID	NAME	BAR
1	foo1	bar1
2	foo2	bar2
3	foo3	bar3
4	foo4	bar4
5	foo5	bar5
6	foo6	bar6
7	foo7	bar7
8	foo8	bar8

Database Cursor Input

FOO 2
id=2
name=foo2
bar=bar2

```
Select * from FOO
where id > 1 and id < 7
```

ID	NAME	BAR
1	foo1	bar1
2	foo2	bar2
3	foo3	bar3
4	foo4	bar4
5	foo5	bar5
6	foo6	bar6
7	foo7	bar7
8	foo8	bar8

Database Cursor Input

```
Select * from FOO
where id > 1 and id < 7
```

FOO 3
id=3
name=foo3
bar=bar3

ID	NAME	BAR
1	foo1	bar1
2	foo2	bar2
3	foo3	bar3
4	foo4	bar4
5	foo5	bar5
6	foo6	bar6
7	foo7	bar7
8	foo8	bar8

Database Cursor Input

```
Select * from FOO
where id > 1 and id < 7
```

ID	NAME	BAR
1	foo1	bar1
2	foo2	bar2
3	foo3	bar3
4	foo4	bar4
5	foo5	bar5
6	foo6	bar6
7	foo7	bar7
8	foo8	bar8

FOO 4
id=4
name=foo4
bar=bar4

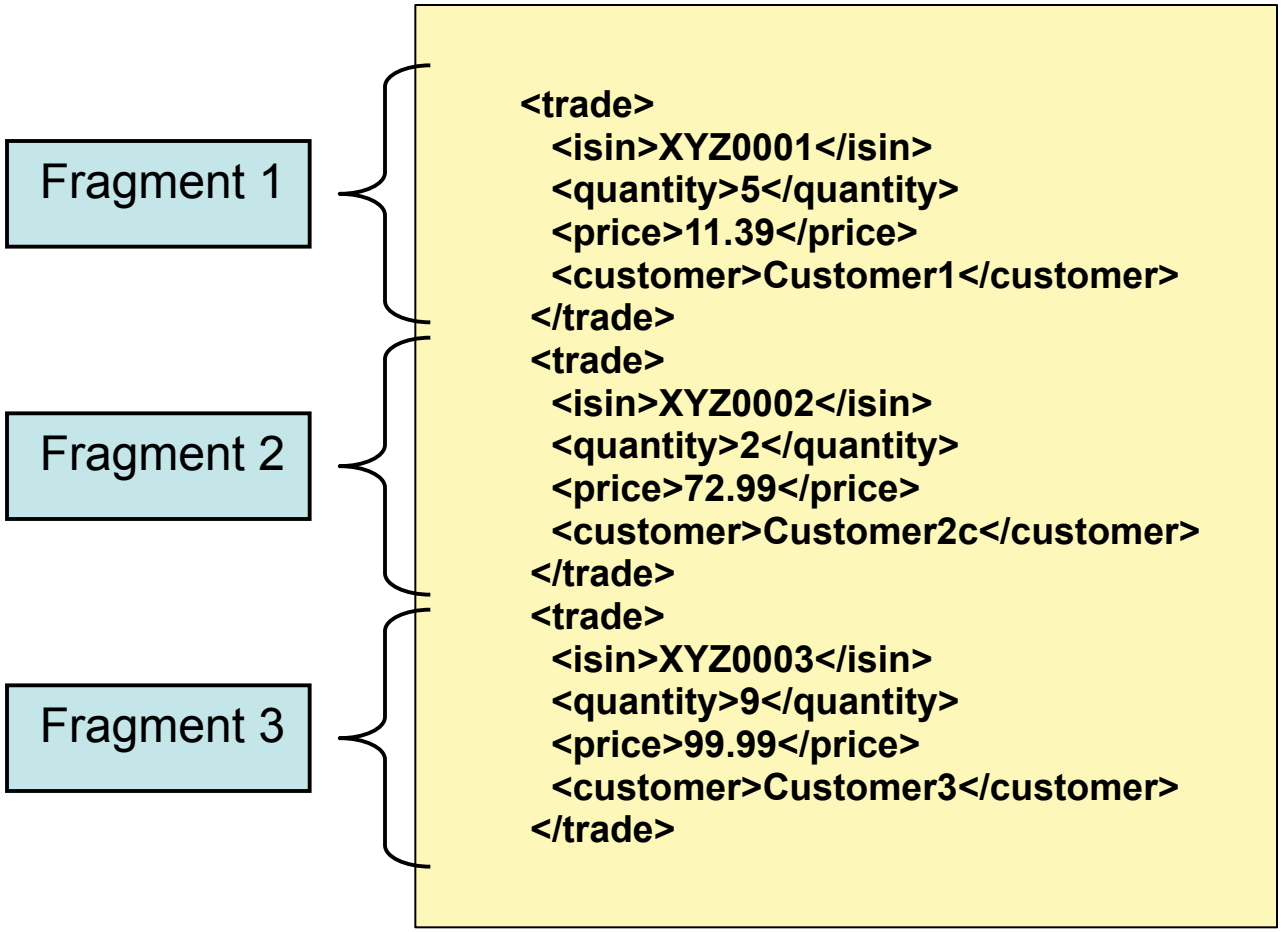
Database Cursor input

```
@Bean
public JdbcCursorItemReader reader (){
   JdbcCursorItemReader reader = new  JdbcCursorItemReader();
   reader.setDataSource(dataSource());
   reader.setVerifyCursorPosition(true);
   reader.setRowMapper( new PlayerSummaryMapper());
   reader.setSql("SELECT GAMES.player_id, GAMES.year_no, SUM(COMPLETES), "+
        "SUM(ATTEMPTS), SUM(PASSING_YARDS), SUM(PASSING_TD), "+
        "SUM(INTERCEPTIONS), SUM(RUSHES), SUM(RUSH_YARDS), "+
        "SUM(RECEPTIONS), SUM(RECEPTIONS_YARDS), SUM(TOTAL_TD) "+
        "from GAMES, PLAYERS where PLAYERS.player_id = "+
        "GAMES.player_id group by GAMES.player_id, GAMES.year_no");
   return reader;
}
```

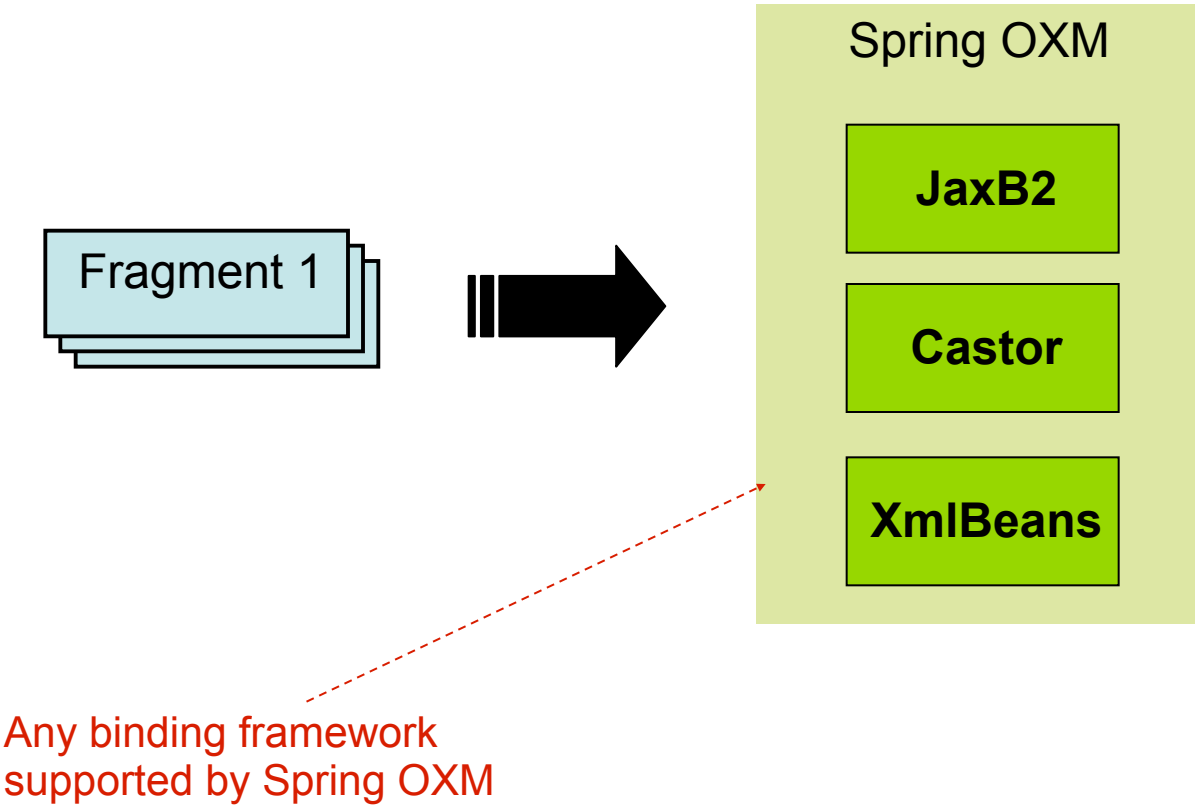
Xml input

- Xml files are separated into fragments based on a root element
- Each fragment is sent to Spring OXM for binding.
- One fragment is processed per call to read().
- Synchronized with the transaction to ensure any rollbacks won't cause duplicate records.

Xml Input


Fragment 1
Fragment 2
Fragment 3
<trade>
<isin>XYZ0001</isin>
<quantity>5</quantity>
<price>11.39</price>
<customer>Customer1</customer>
</trade>
<trade>
<isin>XYZ0002</isin>
<quantity>2</quantity>
<price>72.99</price>
<customer>Customer2c</customer>
</trade>
<trade>
<isin>XYZ0003</isin>
<quantity>9</quantity>
<price>99.99</price>
<customer>Customer3</customer>
</trade>

Xml Input

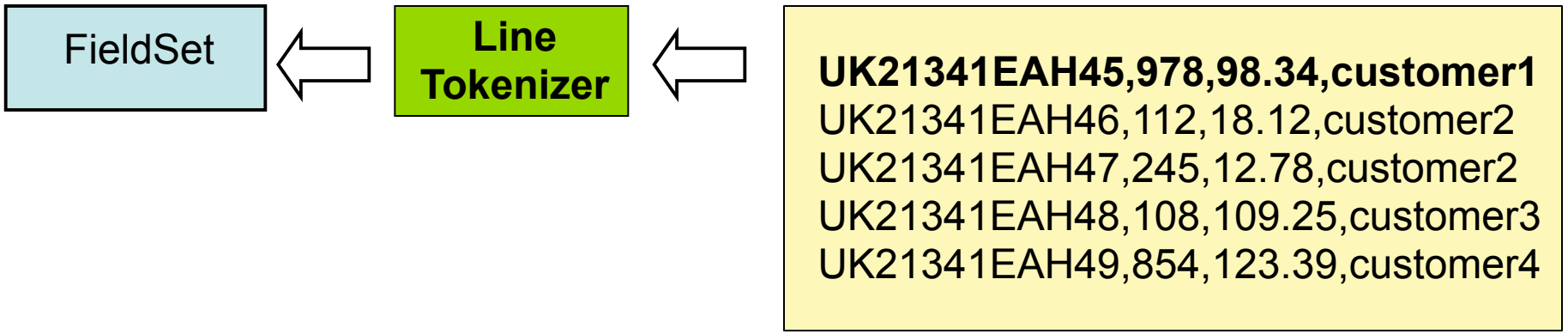

Fragment 1
Spring OXM
JaxB2
Castor
XmlBeans
Any binding framework
supported by Spring OXM

Flat File input

- How lines are read from a file is separated from how a line is parsed allowing for easy support of additional file formats.
- Parsing is abstracted away from users
- Familiar usage patterns: FieldSets can be worked with in similar ways as ResultSets
- Supports both Delimited and Fixed-Length file formats.

LineTokenizer

```
public interface LineTokenizer {
   FieldSet tokenize(String line);
}
```



FieldSetMapper

```
public class TradeFieldSetMapper
        implements FieldSetMapper<Trade> {

    public Trade mapFieldSet(FieldSet fieldSet)
        throws BindException {

      Trade trade = new Trade();
        trade.setIsin(fieldSet.readString(0));
        trade.setQuantity(fieldSet.readLong(1));
        trade.setPrice(fieldSet.readBigDecimal(2));
        trade.setCustomer(fieldSet.readString(3));
        return trade;
    }
}
```

Column-name Access to FieldSet

```
@Bean
public DelimitedLineTokenizer tradeTokenizer() throws Exception {
        DelimitedLineTokenizer dlt = new DelimitedLineTokenizer();
        dlt.setDelimiter( DelimitedLineTokenizer.DELIMITER_COMMA);
        dlt.setNames( "ISIN,Quantity,Price,Customer".split(","));
        return dlt;
}
```

```
Trade trade = new Trade();
trade.setIsin(fieldSet.readString(0));
trade.setQuantity(fieldSet.readLong(1));
trade.setPrice(fieldSet.readBigDecimal(2));
trade.setCustomer(fieldSet.readString(3));
return trade;
```

Column-name Access to FieldSet

```
@Bean
public DelimitedLineTokenizer tradeTokenizer() throws Exception {
        DelimitedLineTokenizer dlt = new DelimitedLineTokenizer();
        dlt.setDelimiter( DelimitedLineTokenizer.DELIMITER_COMMA);
        dlt.setNames( "ISIN,Quantity,Price,Customer".split(","));
        return dlt;
}
```

Column-name Access to FieldSet

```
@Bean
public DelimitedLineTokenizer tradeTokenizer() throws Exception {
        DelimitedLineTokenizer dlt = new DelimitedLineTokenizer();
        dlt.setDelimiter( DelimitedLineTokenizer.DELIMITER_COMMA);
        dlt.setNames( "ISIN,Quantity,Price,Customer".split(","));
        return dlt;
}
```

```
Trade trade = new Trade();
trade.setIsin( fieldSet.readString("ISIN"));
trade.setQuantity( fieldSet.readLong("Quantity"));
trade.setPrice( fieldSet.readBigDecimal("Price"));
trade.setCustomer( fieldSet.readString("Customer"));
return trade;
```

Flat File Parsing errors

- It is extremely likely that bad data will be read in, when this happens information about the error is crucial, such as:
  - Current line number in the file
  - Original Line
  - Original Exception (e.g. `NumberFormatException`)
- Allows for detailed logs, that can be processed on an ad-hoc basis, or using a specific job for bad input data.

ItemProcessor

```
public interface ItemProcessor<I, O> {
  O process(I item) throws Exception;
}
```

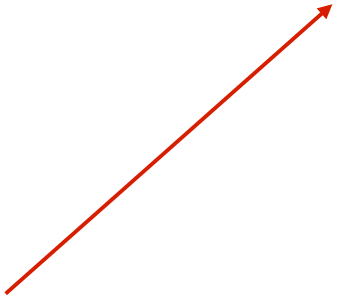
Delegate Exception handling to framework

Item processors

- *optional*
  - *simple jobs may be constructed entirely with out-of-box readers and writers*
- sit between input and output
- typical customization site for application developers
- good place to *coerce* data into the right format for output
- chain transformations using `CompositeItemProcessor`

ItemWriter

```
public interface ItemWriter<T> {

        void write(List<? extends T> items) throws Exception;

}
```



expects a “chunk”

delegate Exception handling to framework

Item Writers

- handles writing and serializing a row of data
- the input might be the output of a **reader** or a **processor**
- handles transactions if necessary and associated rollbacks

ItemWriters

```
@Value("#{systemProperties['user.home']}")
private String userHome;

@Bean @Scope("step")
public FlatFileItemWriter writer( ){
        FlatFileItemWriter w = new  FlatFileItemWriter();
        w.setName( "fw1");
        File out = new File(  this.userHome,
                                "/batches/results").getAbsolutePath();
        Resource res = new FileSystemResource(out);
        w.setResource(res);
        return w;
}
```

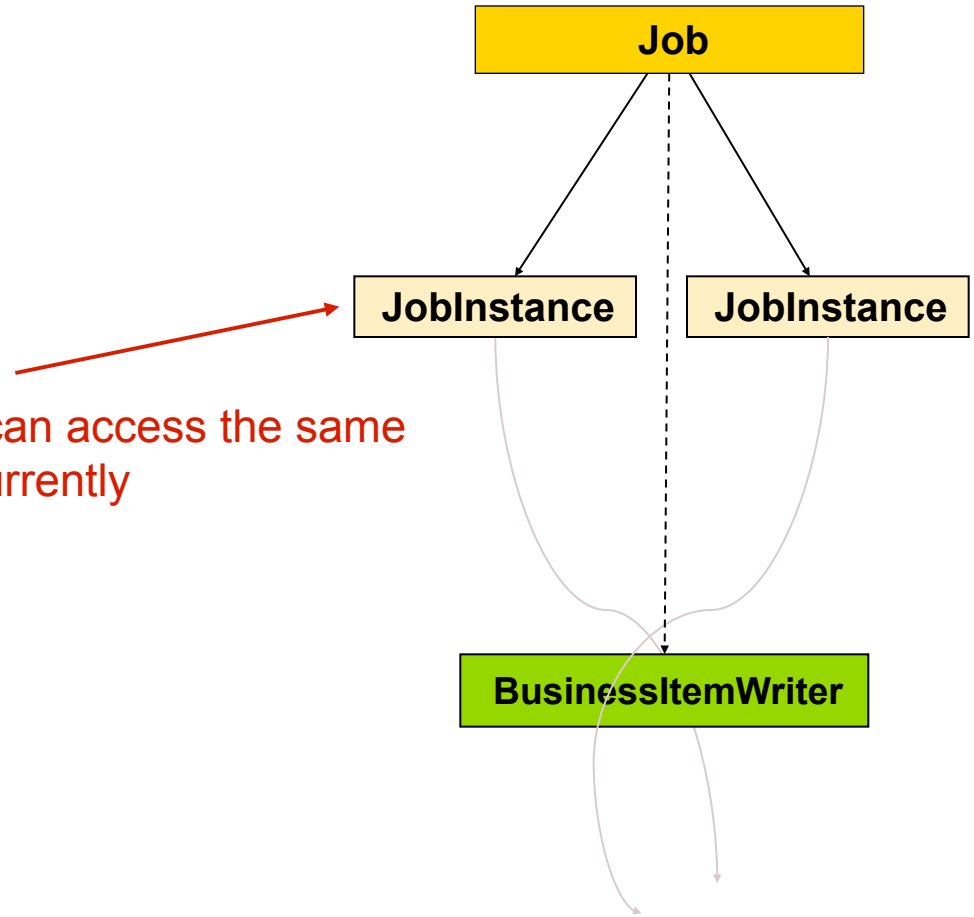
ItemWriters

```
@Bean
public JpaItemWriter jpaWriter() {
        JpaItemWriter writer = new  JpaItemWriter();
        writer.setEntityManagerFactory( entityManagerFactory() );
        return writer;
}
```

Application Concerns

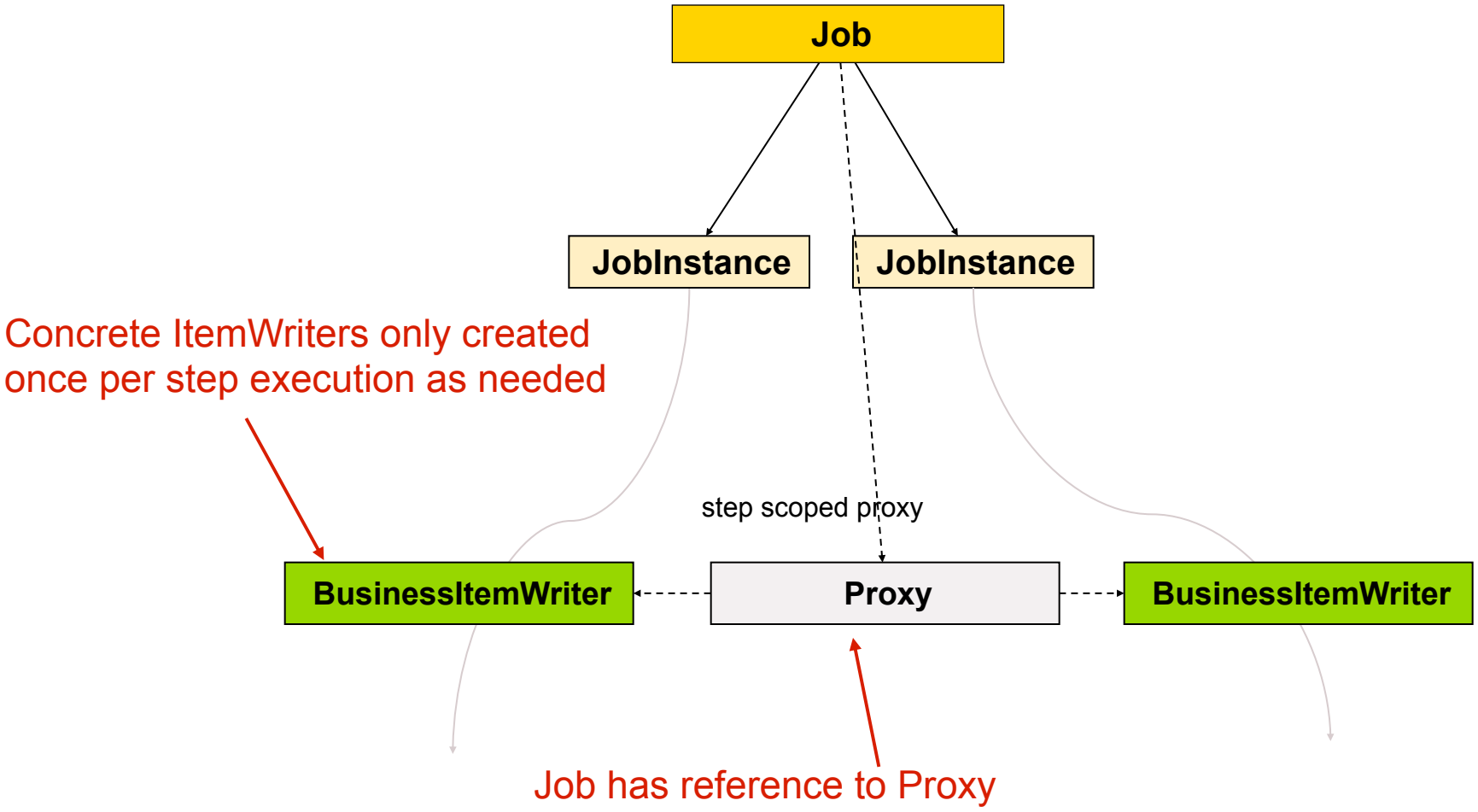
- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Stateful or Stateless?



2 JobInstances can access the same ItemWriter concurrently

Stateful or Stateless: StepContext



Introducing the Step scope

File writer needs to be step scoped so it can flush and close the output stream

```
@Scope("step")
@Bean
public FlatFileItemReader reader(
        @Value("#{jobParameters['input.file']}")
        Resource input ){
    FlatFileItemReader fr = new  FlatFileItemReader();
    fr.setResource(input);
    fr.setLineMapper( lineMapper() );
    fr.setSaveState(true);
    return fr;
}
```

Make this bean injectable

Inner beans inherit the enclosing scope by default

Because it is step scoped the bean has access to the StepContext values and can get at it through Spring EL (in Spring >3)

Step Scope Responsibilities

- Create beans for the duration of a step
- Respect Spring bean lifecycle metadata (e.g. `InitializingBean` at start of step, `DisposableBean` at end of step)
- Allows stateful components in a multithreaded environment

Application Concerns

- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Domain Specific Language

- Keeping track of all the application concerns can be a large overhead on a project
- Need a DSL to simplify configuration of jobs
- DSL can hide details and be aware of specific things like well-known infrastructure that needs to be step scoped
- The Spring way of dealing with this is to use a custom namespace

XML Namespace Example

```
<job id="skipJob" incrementer="incrementer"
    xmlns="http://www.springframework.org/schema/batch">

 <step id="step1">
  <tasklet>
   <chunk reader="fileItemReader" processor="tradeProcessor"
writer="tradeWriter" commit-interval="3" skip-limit="10">
            <skippable-exception-classes>
              <include class="....FlatFileParseException" />
               <include class="....WriteFailedException" />
             </skippable-exception-classes>
           </chunk>
         </tasklet>
         <next on="*" to="step2" />
         <next on="COMPLETED WITH SKIPS" to="errorPrint1" />
         <fail on="FAILED" exit-code="FAILED" />
 </step>

 ...

</job>
```

chunk has input, output and a processor

lots of control over errors

flow control from one step to another

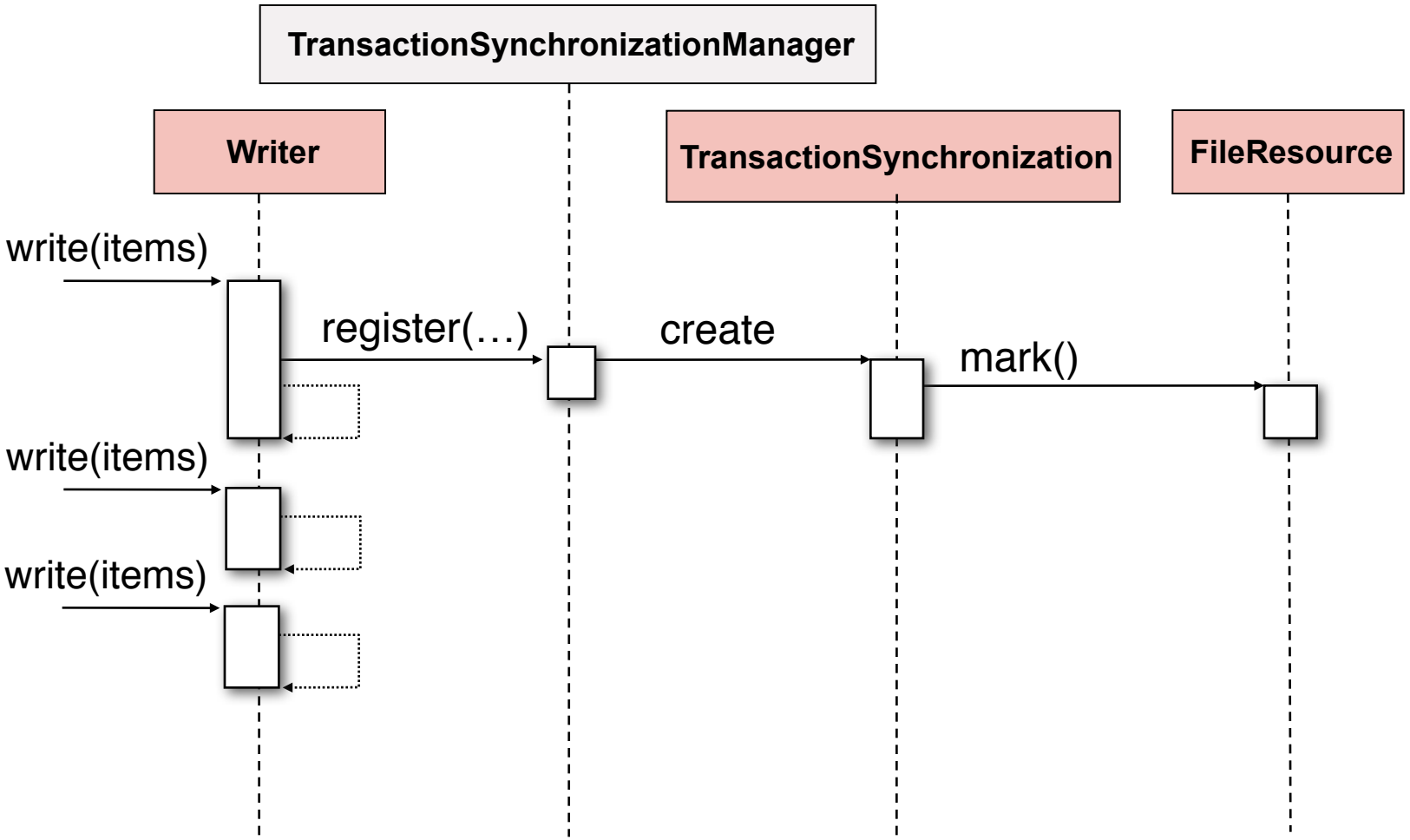
Application Concerns

- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

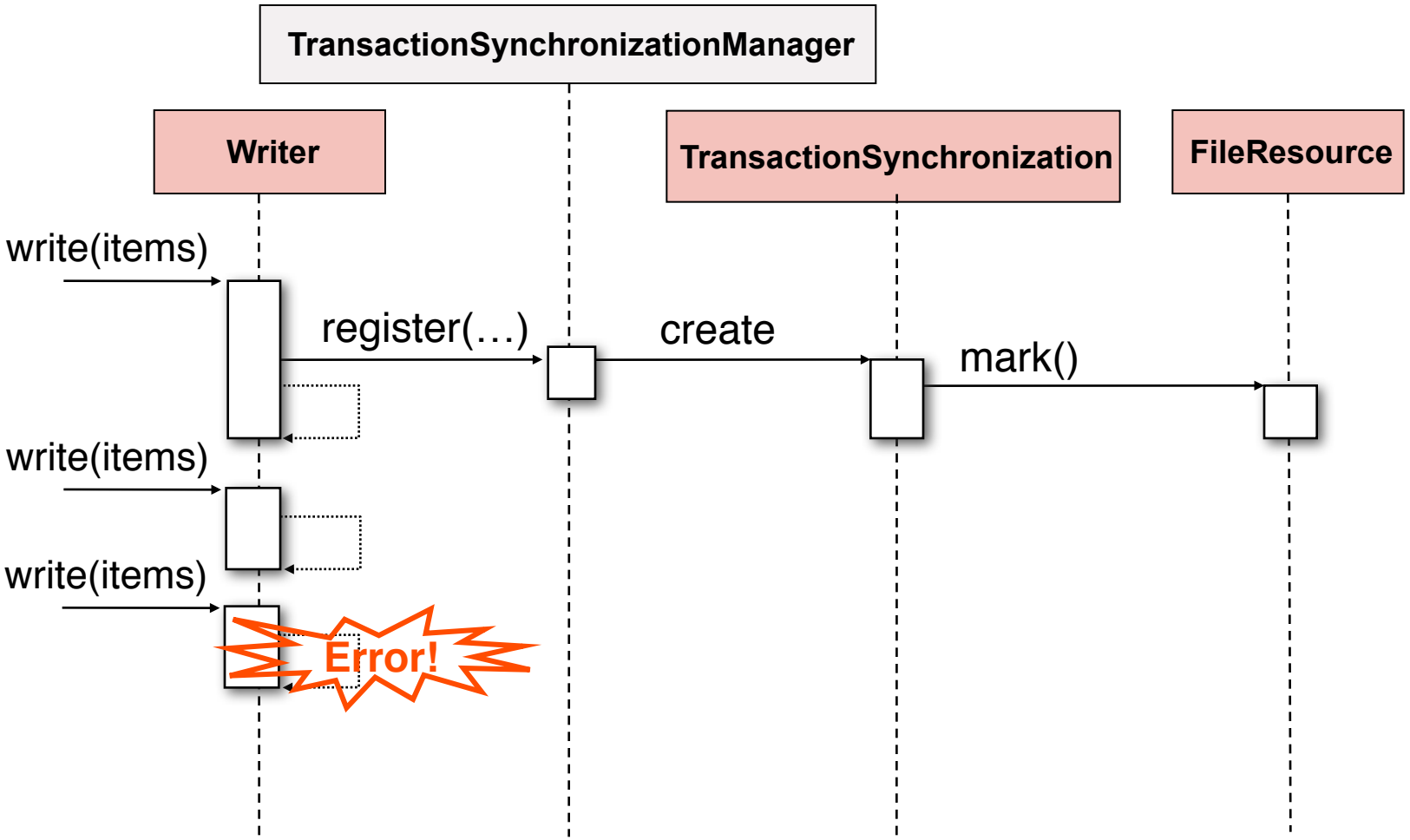
Resource Management Responsibilities

- Open resource (lazy initialisation)
- Close resource at end of step
- Close resource on failure
- Synchronize output file with transaction – rollback resets file pointer
- Synchronize cumulative state for restart
  - File pointer, cursor position, processed keys, etc.
  - Statistics: number of items processed, etc.
- Other special cases
  - Hibernate flush policy
  - JDBC batch update

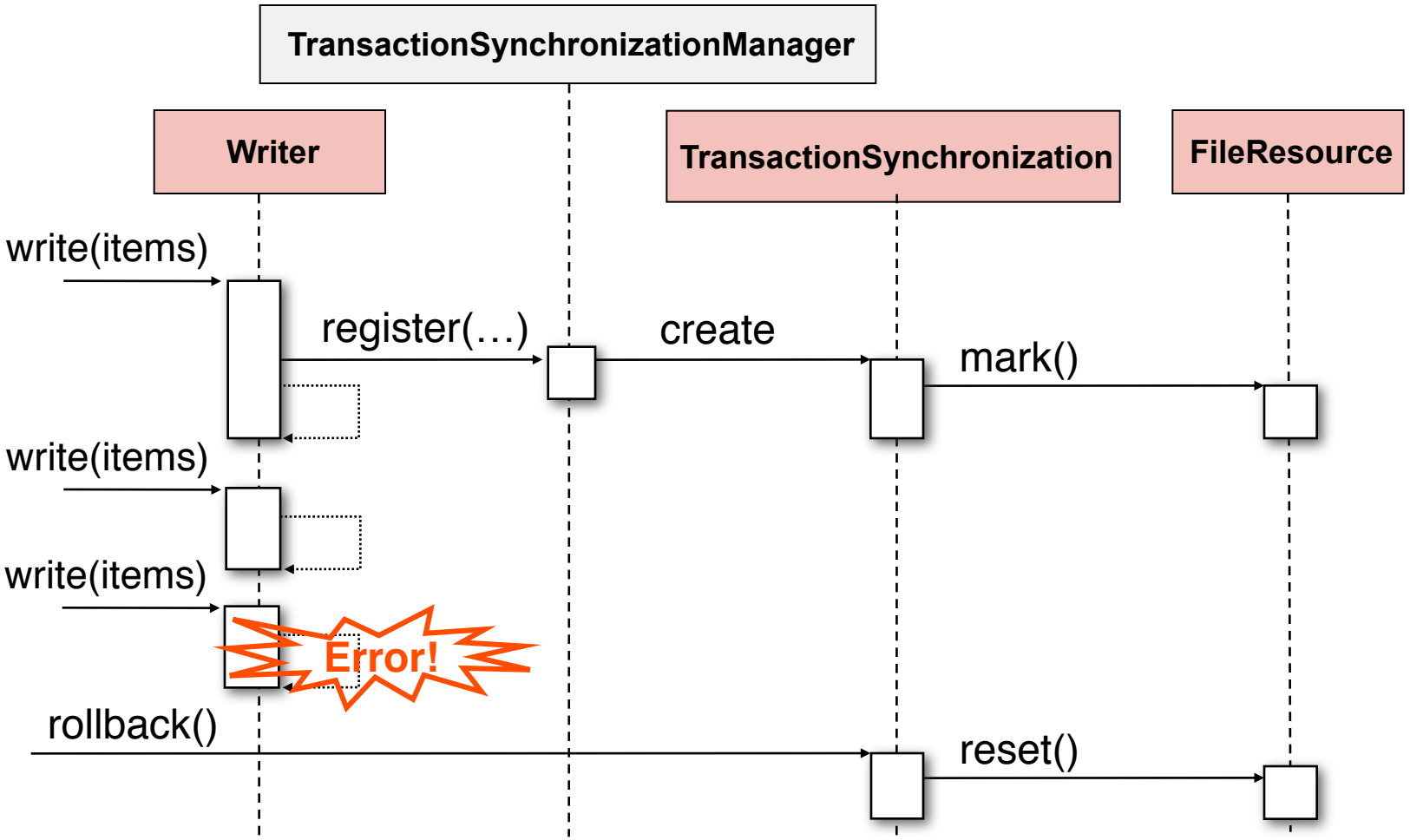
File Output Resource Synchronization



File Output Resource Synchronization



File Output Resource Synchronization



Hibernate Flush Policy

- First problem
  - If we do not flush manually, transaction manager handles automatically
  - …but exception comes from inside transaction manager, so cannot be caught and analyzed naturally by StepExecutor
  - Solution: flush manually on chunk boundaries
- Second problem
  - Errors cannot be associated with individual item
  - Two alternatives
    - Binary search through chunk looking for failure = O(logN)
    - Aggressively flush after each item = O(N)
- `HibernateItemWriter`

Application Concerns

- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Tasklet: Alternative Approaches for Application Developer

- Sometimes Input/Output is not the way that a job is structured
- E.g. Stored Procedure does everything in one go, but we want to manage it as a Step in a Job
- E.g. repetitive process where legacy code prevents ItemReader/Processor being identified
- For these cases we provide Tasklet

Tasklet

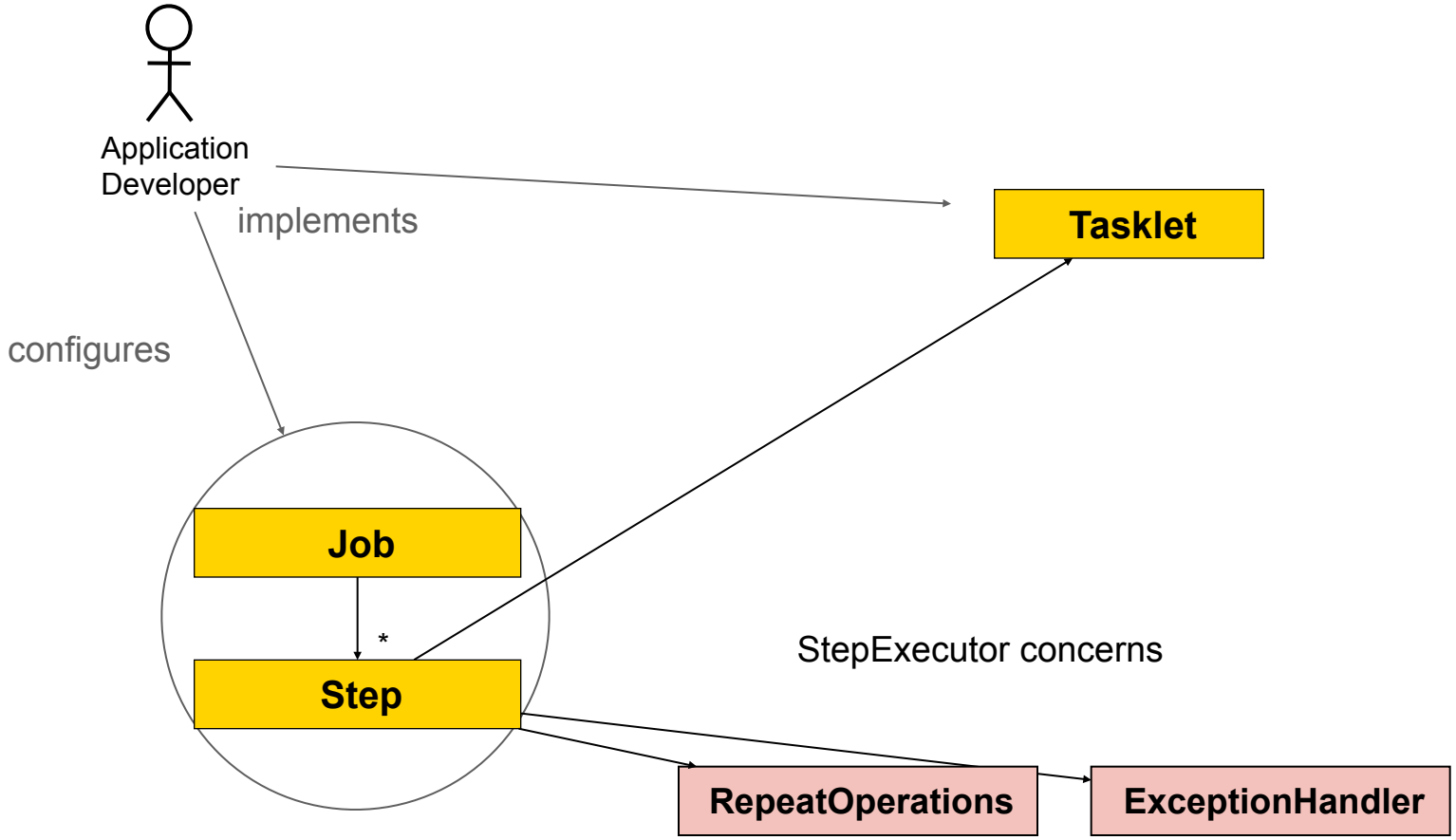
```
public interface Tasklet {

    RepeatStatus execute(StepContribution contribution,
                         ChunkContext chunkContext) throws Exception;
}
```

Signal to framework about end of business process:

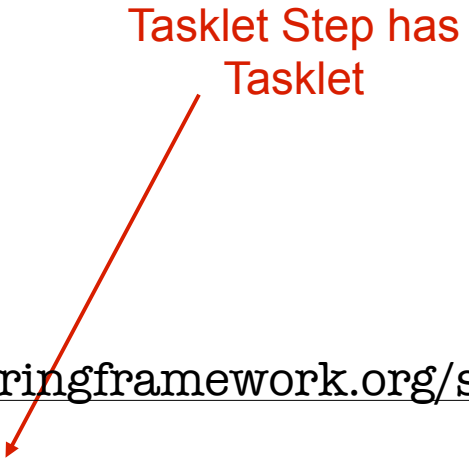
RepeatStatus.*CONTINUABLE*
RepeatStatus.*FINISHED*

delegate Exception handling to framework

Getting Started With a Tasklet



XML Namespace Example with Tasklet




```
<job id="loopJob" xmlns="http://www.springframework.org/schema/batch">
        <step id="step1">
                <tasklet ref="myCustomTasklet">
                        <transaction-attributes propagation="REQUIRED"/>
                </tasklet>
        </step>
</job>
```

Application Concerns

- Getting Started
- Stateful or Stateless
- Step Scope
- Domain Specific Language
- Resource Management
- Alternative Approaches
- Failure Modes

Failure Modes

Event	Response	Alternatives
Bad input record (e.g. parse exception)	Mark item as skipped but exclude from Chunk	Abort Job when skipLimit reached
Bad output record – business or data integrity exception	Mark item as skipped. Retry chunk excluding bad item.	Abort Job when skipLimit reached (after Chunk completes)
Bad output record – deadlock loser exception	Retry Chunk including bad item.	Abort Chunk when retry limit reached
Bad Chunk – e.g. Jdbc batch update fails	Retry Chunk but flush and commit after every item	Discard entire chunk; binary search for failed item
Output resource failure – e.g. disk full, permanent network outage	Graceful abort. Attempt to save batch data.	If database is unavailable meta data remains in "running" state!

Inside Spring Batch

- Architecture and Domain Overview
- Application concerns and Getting Started
- Chunk-Oriented Processing

Chunk-Oriented Processing

- Input-output can be grouped together = Item-Oriented Processing (e.g. `Tasklet`)
- …or input can be aggregated into chunks first = Chunk-Oriented Processing (the `chunk` element of a `tasklet`)
- Chunk processing can be encapsulated, and independent decisions can be made about (e.g.) partial failure and retry
- Step = Chunk Oriented (default, and more common)
- Tasklet = Item Oriented
- Here we compare and contrast the two approaches

Item-Oriented Pseudo Code

```
REPEAT(while more input) {

      TX {
            REPEAT(size=500) {

                  input;
                  output;

            }
      }

}
```

Item-Oriented Pseudo Code

```
REPEAT(while more input) {

      TX {
            REPEAT(size=500) {

                  input;
                  output;

            }
      }

}
```

RepeatTemplate

TransactionTemplate

RepeatTemplate

Business Logic

Item-Oriented Retry and Repeat

```
REPEAT(while more input
          AND exception[not critical]) {
     TX {
          REPEAT(size=500) {
               RETRY(exception=[deadlock loser]) {
                    input;
               } PROCESS {
                    output;
               } SKIP and RECOVER {
                    notify;
               }
          }
     }
}
```

Chunk-Oriented Pseudo Code

```
REPEAT(while more input) {
   chunk = ACCUMULATE(size=500) {
      input;
   }
   RETRY {
      TX {
         for (item : chunk) { output; }
      }
   }
}
```

Summary

- Spring Batch manages the loose ends so you can sleep
- Easy to Use Along with Other Spring Services
- Things We Didn't talk about:
  - remote chunking
  - all the ItemReaders/Writers
- Spring Batch Admin Lets Operations Types Sleep Easier