Spring Batch Technology Preview + Q&A

Kansas City Spring User Group
August 5, 2015

David Pitt, Managing Partner
dpitt@keyholesoftware.com

Presentation & Demo © Keyhole Software 2015

Batch Processing

- Emphasis seems to be on SOA and real time integration
- Still lots of mission critical apps processing billions of transactions per day in "batch mode"
- Many interfaces are still "file-based" batch processes
- Lack of architecture and framework specs have lead to expensive one-off and in-house custom batch implementations

How many times have you created a Java main(..) method, kicked off from a chron job to process an import file....?

Spring Batch – What Is It?

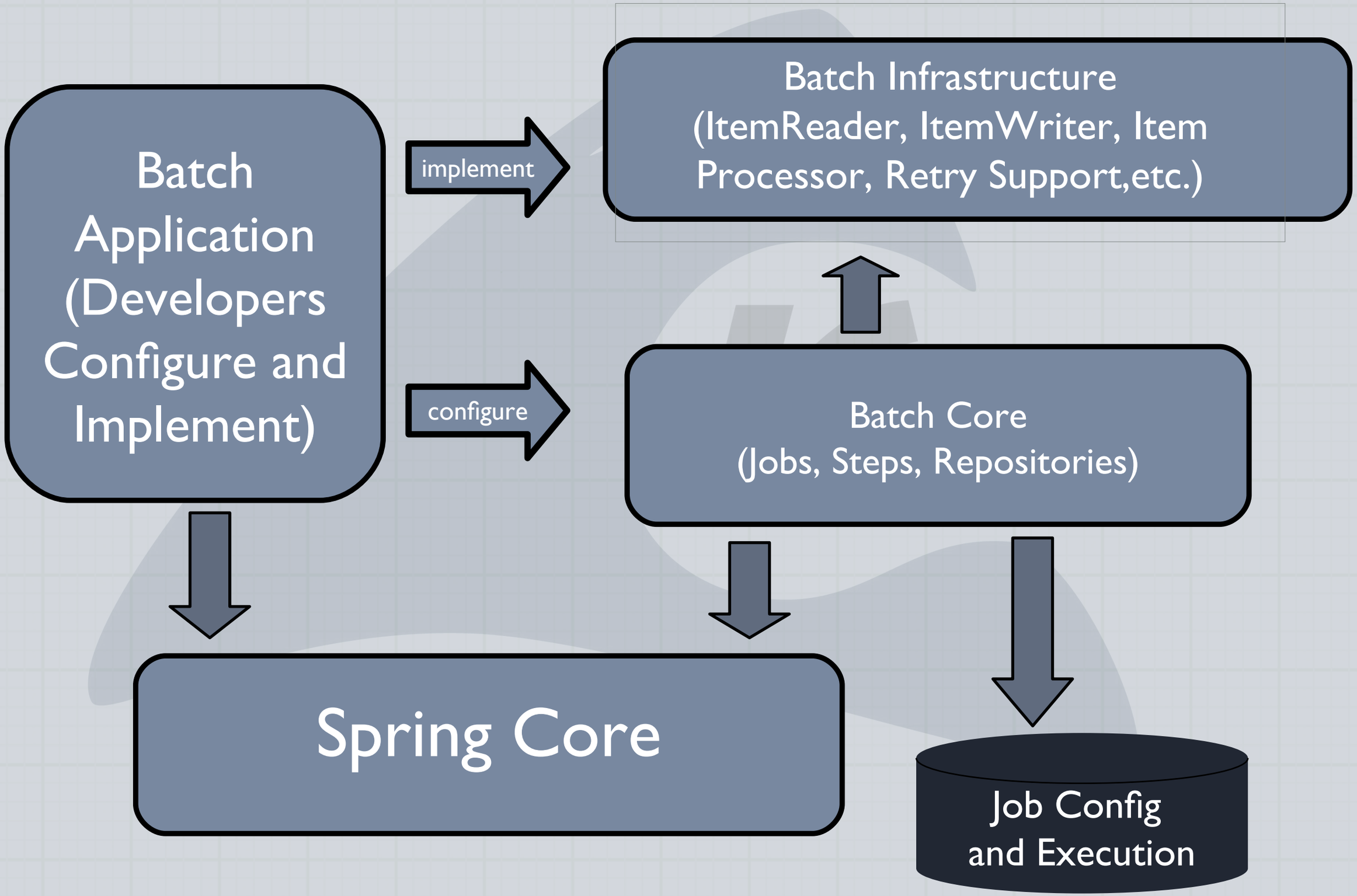
- ETL Batch processing framework for Java
- Jobs built using POJOs and Spring Dependency Injection(DI) framework
- Provides Execution Runtimes for Jobs
- Ability to Process Large Amounts of Data
- Transaction Integrity and Restartable

Where It Came From

- Collaboration between Accenture and Spring (Vmware), now Pivotal
- Accenture had many years experience with numerous enterprise batch processing environments and solutions
- Spring simplifies and modularizes JEE application architecture frameworks via dependency injection

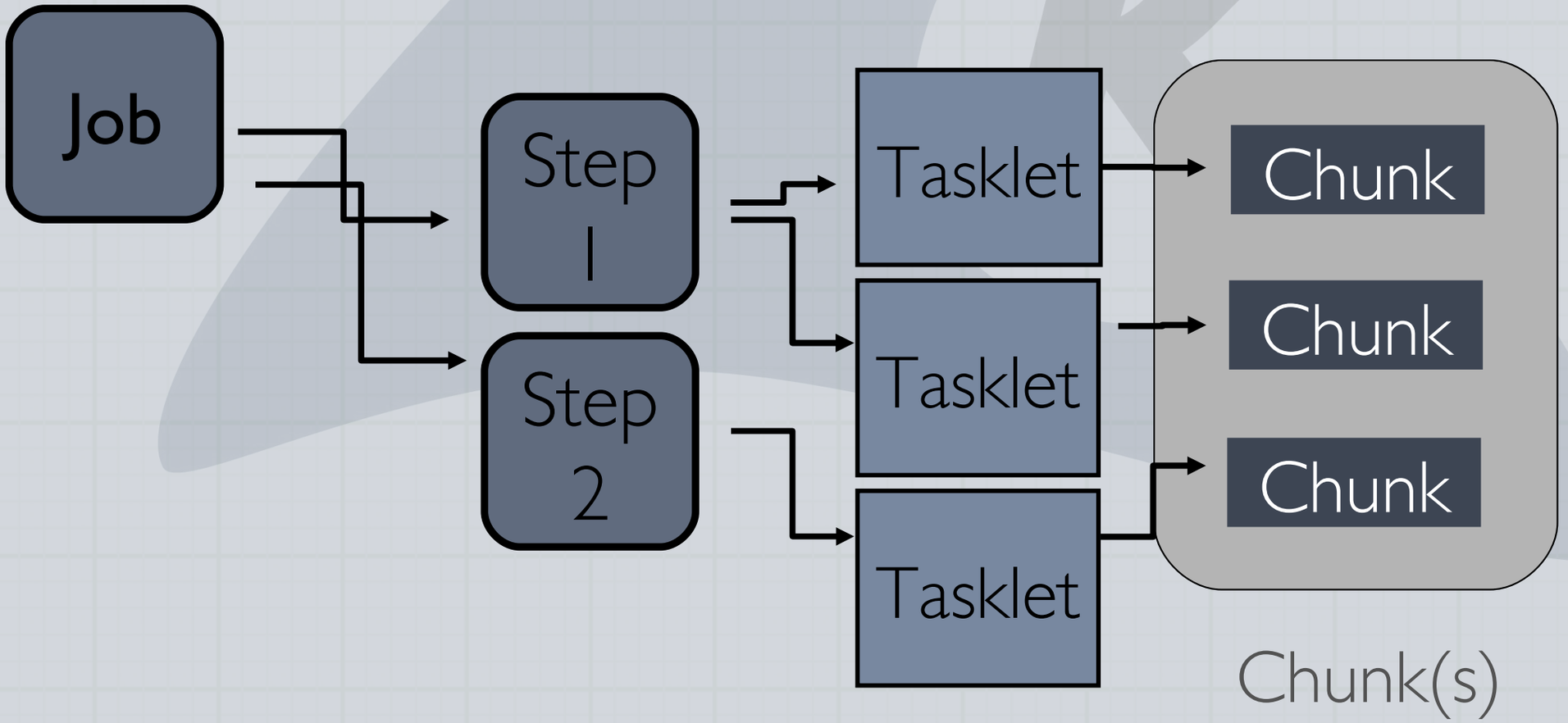
Pivotal™

Architecture



Concepts and Features

- Borrows from Traditional Batch Processing Concepts
  - Jobs are made up from steps that perform tasks
  - Run in a single VM?



Job Definition

- Spring DI used to configure (WIRE UP) classes configuration
- Elements to "Wire UP" in application Context
  - JobOperator
  - JobRepository
  - JobLauncher
  - Job
  - Steps
  - Tasklet
  - Chunks (Reader/Processor/Writer)
  - Standard Stuff (Properties, DBconnections, tx manager, etc)

Spring Core Provides Impl

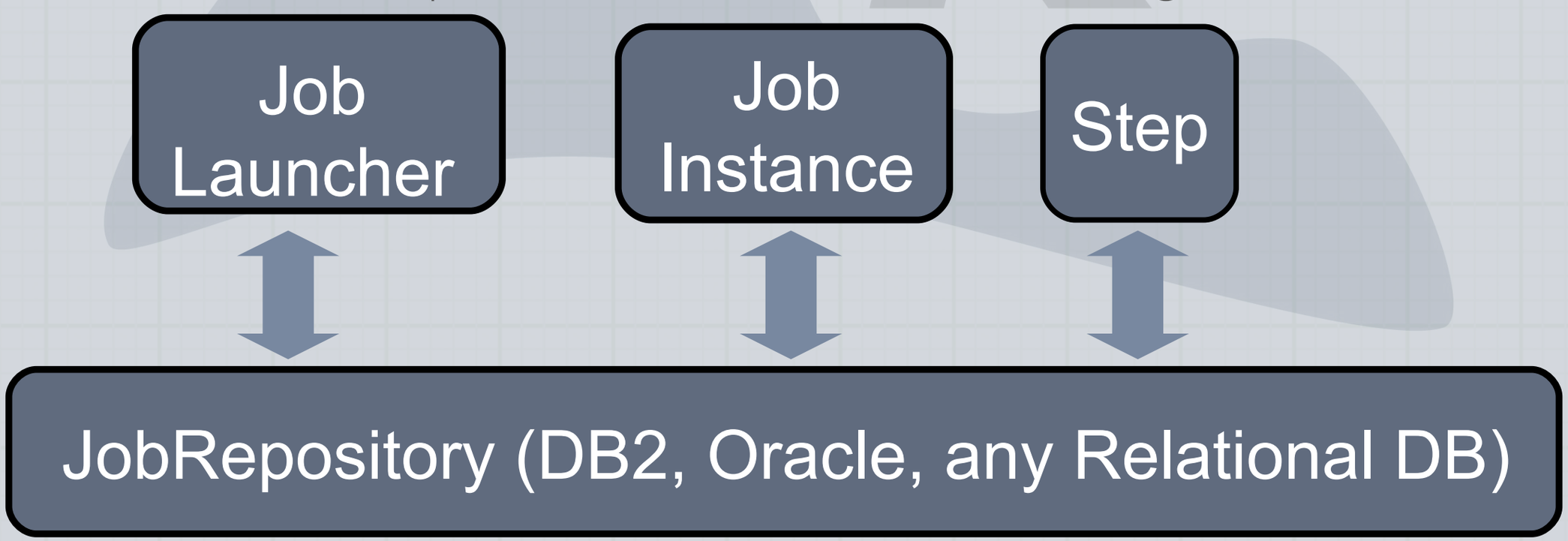
Developer Configures

Implement

Job Repository

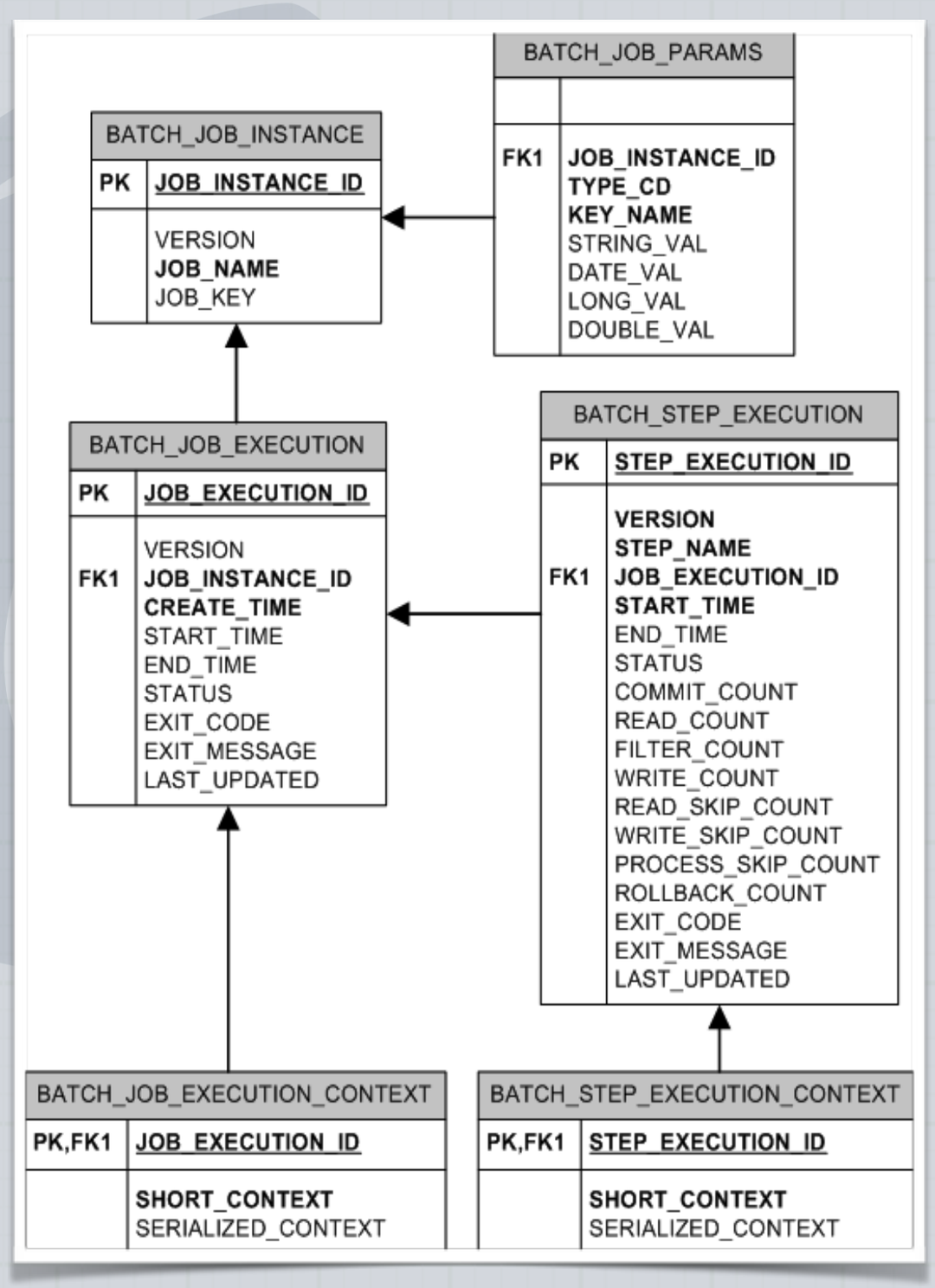
- Governs CRUD for domain objects during job and step execution
- Repositories can be a database or in memory (default)

```
<job-repository id="jobRepository"
        data-source="dataSource"
        transaction-manager="transactionManager"
        isolation-level-for-create="SERIALIZABLE“
    table-prefix="BATCH_" max-varchar-length="1000" />
```



Batch META-DATA

Relational Meta Data
Supporting Spring Batch



Job

- Encapsulates entire Job Process steps
- Configured in Spring job context(s) files
- Or, Java Config

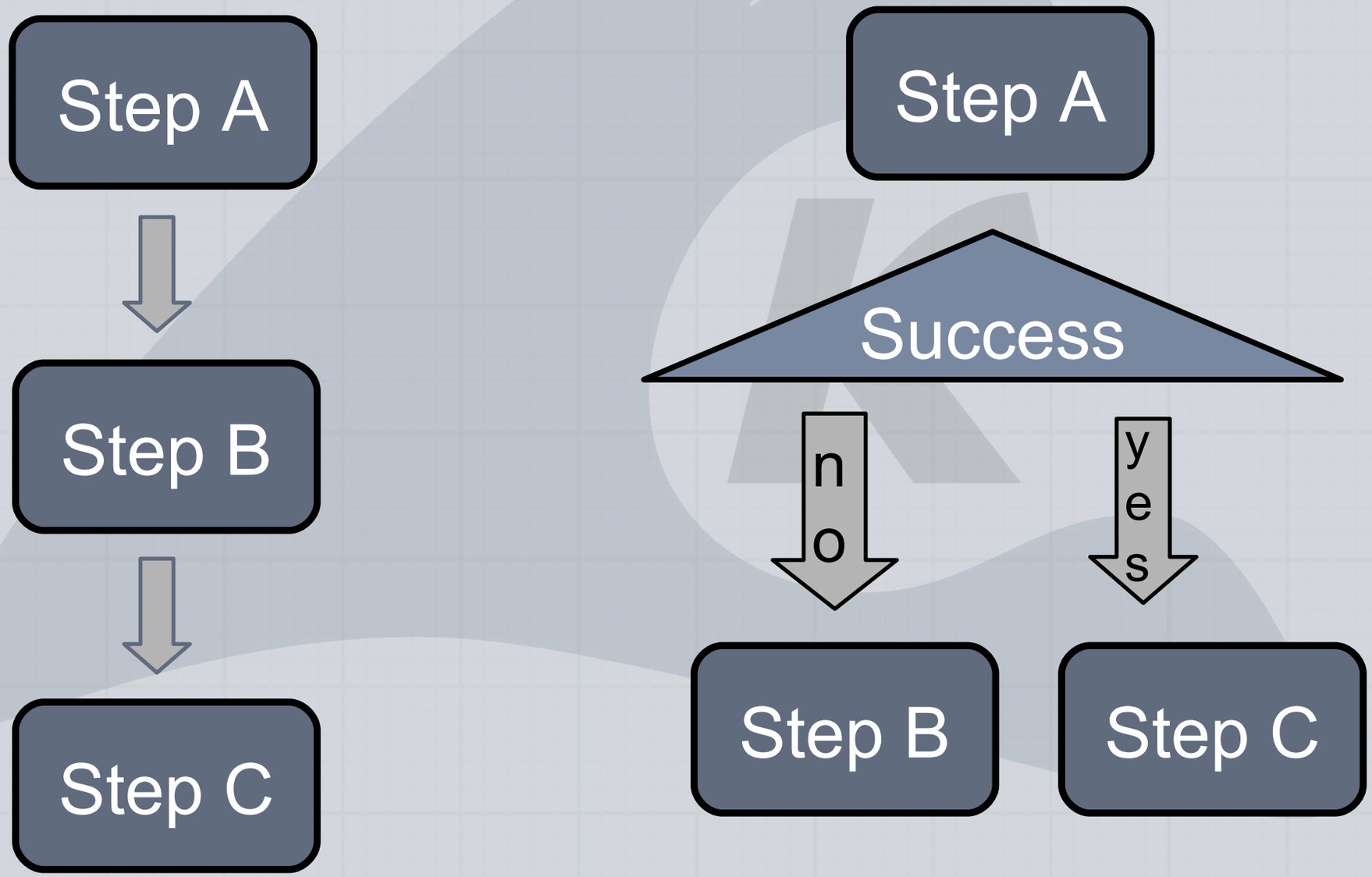
```
<job id="TimesheetInvoicing" restartable="false">
<step id="importTimesheets" next="invoiceDetail">
<tasklet transaction-manager="txManager" >
<chunk reader="timesheetFileItemReader"
writer="timesheetWriter"
commit-interval="3">
</chunk>
</tasklet>
</step>
<step id="invoiceDetail" next="outputPdfReport">
<tasklet transaction-manager="txManager" >
<chunk reader="timesheetReader"
processor="createInvoice"
writer="writeInvoice"
commit-interval="3">
</chunk>
</tasklet>
</step>
<step id="outputPdfReport">
<tasklet transaction-manager="txManager" >
<chunk reader="reportReader"
writer="reportWriter"
commit-interval="3">
</chunk>
</tasklet>
</step>
</job>
```

Job Java Config

```java
@Configuration
@EnableBatchProcessing
public class TickerPriceConversionConfig {

    @Autowired
    private JobBuilderFactory jobs;

    @Autowired
    private StepBuilderFactory steps;

    @Bean
    public ItemReader<TickerData> reader() throws MalformedURLException {
        FlatFileItemReader<TickerData> reader = new FlatFileItemReader<TickerData>();
        reader.setResource(new UrlResource("http://finance.yahoo.com/d/quotes.csv?s=XOM+IBM+JNJ+MSFT&amp;f=snd1ol1p2"));
        reader.setLineMapper(new DefaultLineMapper<TickerData>() {{
            setLineTokenizer(new DelimitedLineTokenizer());
            setFieldSetMapper(new TickerFieldSetMapper());
        }});
        return reader;
    }

    @Bean
    public ItemProcessor<TickerData, TickerData> processor() {
        return new TickerPriceProcessor();
    }

    @Bean
    public ItemWriter<TickerData> writer() {
        return new LogItemWriter();
    }

    @Bean
    public Job TickerPriceConversion() throws MalformedURLException {
        return jobs.get("TickerPriceConversion").start(convertPrice()).build();
    }

    @Bean
    public Step convertPrice() throws MalformedURLException {
        return steps.get("convertPrice")
                .<TickerData, TickerData> chunk(5)
                .reader(reader())
                .processor(processor())
                .writer(writer())
                .build();
    }
}
```

Steps

- Can be sequential or conditional



Step Types

Spring Core Provides:

- Simple Step
- Fault Tolerant Step
  - Skip and Retry Behavior
- Or, Implement Custom

Step Definition

```
<job id="TimesheetInvoicing" restartable="false">

<step id="importTimesheets" next="invoiceDetail">
    <tasklet transaction-manager="txManager" >
<chunk reader="timesheetFileItemReader"

                writer="timesheetWriter"
                commit-interval="3">
                </chunk>
                </tasklet>
                </step>

<step id="invoiceDetail" next="outputPdfReport">
    <tasklet transaction-manager="txManager" >
        <chunk reader="timesheetReader"
        processor="createInvoice"
                writer="writeInvoice"
                commit-interval="3">
                </chunk>
                </tasklet>
                </step>

        <step id="outputPdfReport">
  <tasklet transaction-manager="txManager" >
<chunk reader="reportReader"
                writer="reportWriter"
                commit-interval="3">
                </chunk>
                </tasklet>
                </step>
                </job>
```

Steps

Chunks - Where The Work Happens

- Tasklets
  - Contain Chunks
  - Chunks define Readers, Processors, Writer and Commit interval
  - Unit of Work
  - Allows HUGE amounts of data to be processed
- Developer implements Reader, Writer, Processor (optional)
  - Implements Interface in Spring Core classes

Chunk Elements

Chunk Definition

```
<step id="invoiceDetail" next="step2">
<tasklet transaction-manager="txManager"
>
<chunk reader="timesheetReader"
processor="createInvoice"
writer="writeInvoice"
commit-interval="3">
</chunk>
</tasklet>
</step>
```

Read Timesheets

Process, create invoice objects

Write (persist) Invoice Objects

Records read/write before tx is committed

Supplied Readers/Writer

- Flat Files
- JMS
- Cursor and Paging (Hibernate, Spring, IBATIS, raw JDBC)
- XML
- HTTP/FTP

- No coding required, just wire them up...
- Many apply convention over configuration...

Supplied JPA and FlatFile Reader

Read timesheet records from timesheet database...

```
<beans:bean id="timesheetReader"
class="org.springframework.batch.item.database.JpaPagingItemReader">
<beans:property name="entityManagerFactory" ref="dataFactory"/>
 <beans:property name="queryString" value="select t from Timesheet
t"/>
<beans:property name="pageSize" value="1000"/>
</beans:bean>
```

Supplied JPA and FlatFile Reader

Read timesheet records from comma delimited flat file...

```
<beans:bean id="timesheetFileItemReader"
class="org.springframework.batch.item.file.FlatFileItemReader">
<beans:property name="resource" value="classpath:data/input/timesheets.sdf" />
<beans:property name="lineMapper">
<beans:bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
<beans:property name="lineTokenizer">
<beans:bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
<beans:property name="names"
value="ID,week,company,monday,tuesday,wednesday,thursdsay,friday,saturday,sunday" />
</beans:bean>
</beans:property>
<beans:property name="fieldSetMapper">
<beans:bean class="com.khs.batch.timesheet.TimesheetFieldSetMapper" />
</beans:property>
</beans:bean>
</beans:property>
</beans:bean>
```

FlatFileItemReader Java Config

```
@Bean
    public ItemReader<TickerData> reader() throws
MalformedURLException {
        FlatFileItemReader<TickerData> reader = new
FlatFileItemReader<TickerData>();
        reader.setResource(new
UrlResource("http://finance.yahoo.com/d/quotes.csv?
s=XOM+IBM+JNJ+MSFT&amp;f=snd1ol1p2"));
        reader.setLineMapper(new DefaultLineMapper<TickerData>() {{
            setLineTokenizer(new DelimitedLineTokenizer());
            setFieldSetMapper(new TickerFieldSetMapper());
        }});
        return reader;
    }
```

Example - Reader Implementation

```
public class TimesheetReader implements ItemReader<Timesheet> {

List<Timesheet> timesheets;
int current = 0;

@Autowired
TimesheetJPADao dao;

@BeforeStep
public void init() {
    timesheets = dao.findAll();
}

@AfterStep
public void reset() {
    timesheets = null;
    current = 0;
}

@Override
public Timesheet read() throws Exception, UnexpectedInputException,
ParseException, NonTransientResourceException {

    Timesheet ts = null;
    if (current < timesheets.size()) {
        ts = timesheets.get(current);
        current++;
    }
    return ts;
}
}
```

Initializing and Housekeeping Before/After Step Execution...

Read a Timesheet; Null Indicates Done...

Example – Processor Implementation

Timesheet object
from reader....

```
public class InvoiceProcessor implements ItemProcessor<Timesheet,
InvoiceItem> {

@Override
public InvoiceItem process(Timesheet ts) throws Exception {

    InvoiceItem item = new InvoiceItem();
    item.setCompany(ts.getCompany());
    item.setAmount(ts.total() * 100.00);

    return item;
}
}
```

InvoiceItem
Object Goes
to Writer...

Example – Writer Implementation

```
public class TimesheetWriter implements ItemWriter<Timesheet> {

@Autowired
TimesheetJPADao dao;

@Override
public void write(List<? extends Timesheet> items) throws
Exception {

    for (Timesheet ts : items) {
        dao.persist(ts);
    }

    }
}
```

Intercepting Processing

- Listeners/Event Handler
  - Before Step/Job
  - After Step/Job
  - Job Completion

Scalability Strategies

- Vertical
  - Faster machine, more RAM, CPU, GHz
- Horizontal
  - Add machines

Spring Batch Scalability Support

- Remote Chunking
  - Chunks can be partitioned across process spaces
  - Using Messaging data read from chunk sent to a remote processor
- Step Partitioning
  - Step processing is partitioned to separate process or thread
  - Interacts with a partition event listener. JMS, Remote EJB, Web Service, Grid and more handlers are provided.

Threads or Processes

- Multi-threaded Step (single process) – Vertical scaling Reader/Processor/Writer threaded
- Parallel Steps (single process) – Vertical scaling steps execute in parallel
- Remote Chunking of Step (multi process) – Horizontal scaling Processing/Writing on remote servers
- Partitioning a Step (single or multi process) – Horizontal scaling, Partitions reader data to multiple threads or remote nodes

Example Parallel Step Execution

```
<job id="job1">
  <split id="split1" task-executor="taskExecutor" next="step4">
    <flow>
      <step id="step1" parent="s1" next="step2"/>
      <step id="step2" parent="s2"/>
    </flow>
    <flow>
      <step id="step3" parent="s3"/>
    </flow>
  </split>
  <step id="step4" parent="s4"/>
</job>

<beans:bean id="taskExecutor"
class="org.spr...SimpleAsyncTaskExecutor"/>
```

Execution and Monitoring Environment

- Spring Batch Admin (Web App)
- Application Launchers (Command Line, JMS)
- OSGI Spring DM Server
- JMX (Used to interface with 3rd Party Scheduling and monitoring Software)
- Spring Integration (i.e. message bean or service to invoke job launcher(s))

Scheduling

- Primitive support, does not really have full function scheduling features
- Out-of-the-Box
  - Quartz
  - Chron
- Third Party (JMX or custom API)

Batch Admin Console

Spring Batch Admin: Jobs

http://localhost:8080/khs-batch-admin/jobs

Spring Batch Admin

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Job Names Registered

Name	Description	Execution Count	Launchable	Incrementable
InvoiceReport	No description	0	true	false
TimesheetInvoicing	No description	0	true	false

Rows: 1-2 of 2 Page Size: 20

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Execution Statistics

Job Launch
Starting/Stopping

Spring Batch Admin: Job Summary

http://localhost:8080/khs-batch-admin/jobs/TimesheetInvoicing

Spring Batch Admin

Home | Jobs | Executions | Files | SpringSource | Spring Batch

Job name=TimesheetInvoicing: Launch

Job Parameters (key=value pairs): (Not incrementable)

If the parameters are marked as "Not incrementable" then the launch button launches an instance of the job with the parameters shown (which might be a restart if the last execution failed). You can always add new parameters if you want to (but not if you want to restart).

© Copyright 2009-2010 SpringSource. All Rights Reserved.

Contact SpringSource

Now, A Demo!

For more, check out our 6-part tutorial series on how to get started with Spring Batch on https://keyholesoftware.com/blog.

Short Link: bit.ly/springbatchkhs

Presentation & Demo © Keyhole Software 2015