JSR 352 - The Future of Java Batch and WebSphere Compute Grid

David Follis

IBM

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence.**

WebSphere Application Server

Session	Title	Time	Room
17363	Debug 101-Using ISA Tools for Apps in WebSphere Application Server z/OS	Monday 11:15	Europe 11
17367	WebSphere Liberty on Windows and z/OS (Among Other Things) Hands-On Lab	Tuesday 10:00	Asia 5
17361	ABCs of WAS	Tuesday 1:45	Oceanic 7
17368	z/OS Connect: Opening up z/OS Assets to the Cloud and Mobile Worlds	Tuesday 3:15	Oceanic 7
17362	Configuring Timeouts for WebSphere Application Server on z/OS	Wednesday 8:30	Oceanic 7
17366	WebSphere Liberty and WebSphere Application Server Classic - What's New?	Wednesday 11:15	Oceanic 7
17364	IBM Installation Manager for z/OS System Programmers: Web-based Installs, Fix Packs, and How iFixes Really Work	Thursday 4:30	Oceanic 7
17365	JSR 352 - The Future of Java Batch and WebSphere Compute Grid	Friday 10:00	Oceanic 6

Topics to be Discussed

- **Brief Overview of Batch Processing**

  Including background on Java Batch evolution

- **Overview of JSR 352**

  A review of the key elements of the standard

- **IBM Implementation and Extensions**

  A review of how JSR 352 is implemented by IBM, including extensions to the standard that provide additional operational features and benefits

Batch Processing …

… and what led up to Java Batch

Batch Processing Has Been Around a Very Long Time

A picture from the 1960s, and batch processing pre-dated this by several decades, or even centuries, depending on what is considered a "computer"

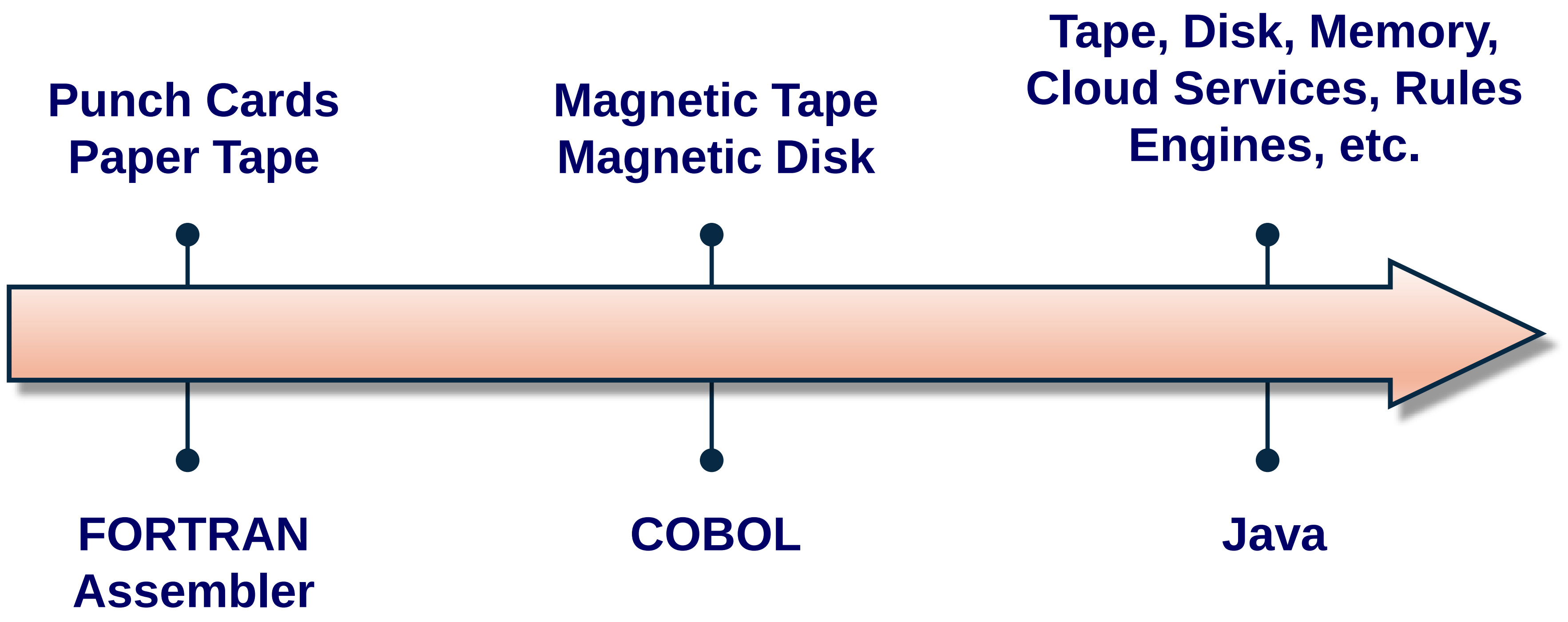
There has long been a need to process large amounts of data to arrive at results from the data

There continues to be the same need today

It is unlikely the need to do processing in batch will go away any time soon

The need persists, the approach has evolved …

Evolution: Data Storage and Programming Languages



Change is driven by need. So what is driving the trend towards Java for batch processing?

Things Creating Push to Java for Batch

Desire to Modernize Batch Processes

Motivation behind this takes many forms – new business needs; some update to an existing batch program is needed and it’s seen as a good opportunity to re-write in Java; separate business logic into rules engine for more agile processing

Availability of Java Skills

Particularly relative to other skills such as COBOL.

z/OS: Ability to Offload to Specialty Engines

Workload that runs on z/OS specialty engines (zAAP, zIIP) is not counted towards CPU-based software charges.

Can Java Run as Fast as Compiled Code?

Comparably … and *sometimes* faster*:

- **Batch processing is by its nature iterative, which means Java classes prone to being Just-in-Time (JIT) compiled at runtime**
- **Java JIT compilers are getting very good at optimizing JIT'd code**
- **z/OS: System z processor chips have instructions specifically designed to aid JIT-compiled code**
- **COBOL that has not been compiled in a long time is operating with less-optimal compiled code that does not take specific advantage of chip instructions**

*****

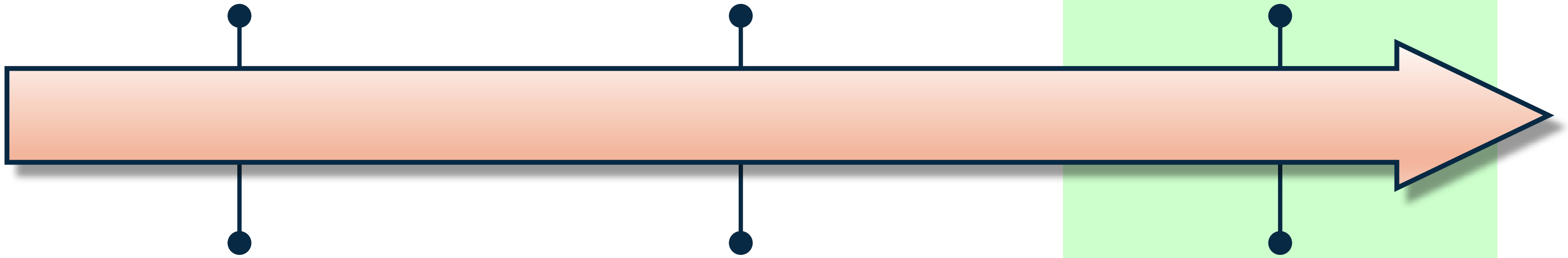
Results vary, depending on many factors. This is not a promise of performance results.

The Evolution of Java Batch …

Roll Your Own

Vendor Batch Frameworks

Open Standard



Typically built around the JVM launcher concept

Processing logic all custom-code, often specific to the batch program, with only some re-use

Examples: IBM Compute Grid, Spring Batch

Framework removed a lot of coding effort from developer and allowed focus to be on business logic

JSR 352

Open standards allow for code portability and the development of larger libraries of re-usable code

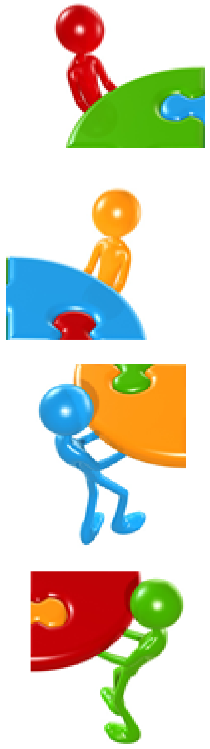
Overview of JSR 352

The Process of Creating an Open Standard

Formation of Working Group

Initial Release of Standard Specification

Release of Vendor Implementations



Individuals working on the challenges independent of one another

The group works to create a vision and a document of the proposed specification.

After review and acceptance, it becomes a published specification.

IBM led this group, with involvement from people from several other companies.

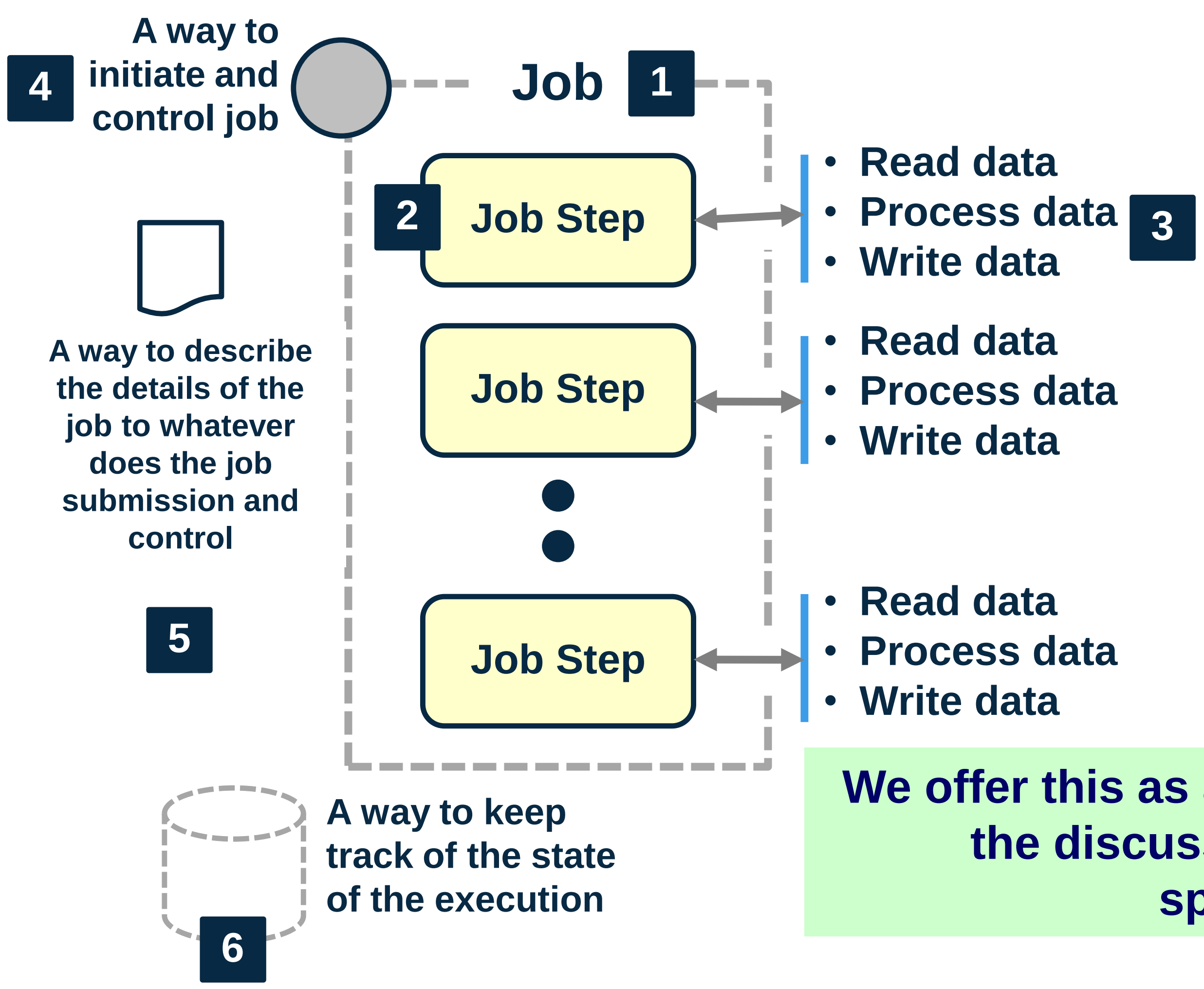
https://jcp.org/en/jsr/detail?id=352

The specification details the requirements and interfaces.

The JSR 352 specification was released in May 2013, and has been accepted as a component of the Java EE 7 specification as well.

Vendors release products and provide extensions for additional value

Very Abstract Representation of a "Batch Job"


A way to initiate and control job
4
Job
1
2
Job Step
Job Step
Job Step
3
A way to describe the details of the job to whatever does the job submission and control
5
A way to keep track of the state of the execution
6

- Read data
- Process data
- Write data

- Read data
- Process data
- Write data

- Read data
- Process data
- Write data

We offer this as a way to set the stage for the discussion of the JSR 352 specification

The JSR 352 Diagram to Describe the Architecture

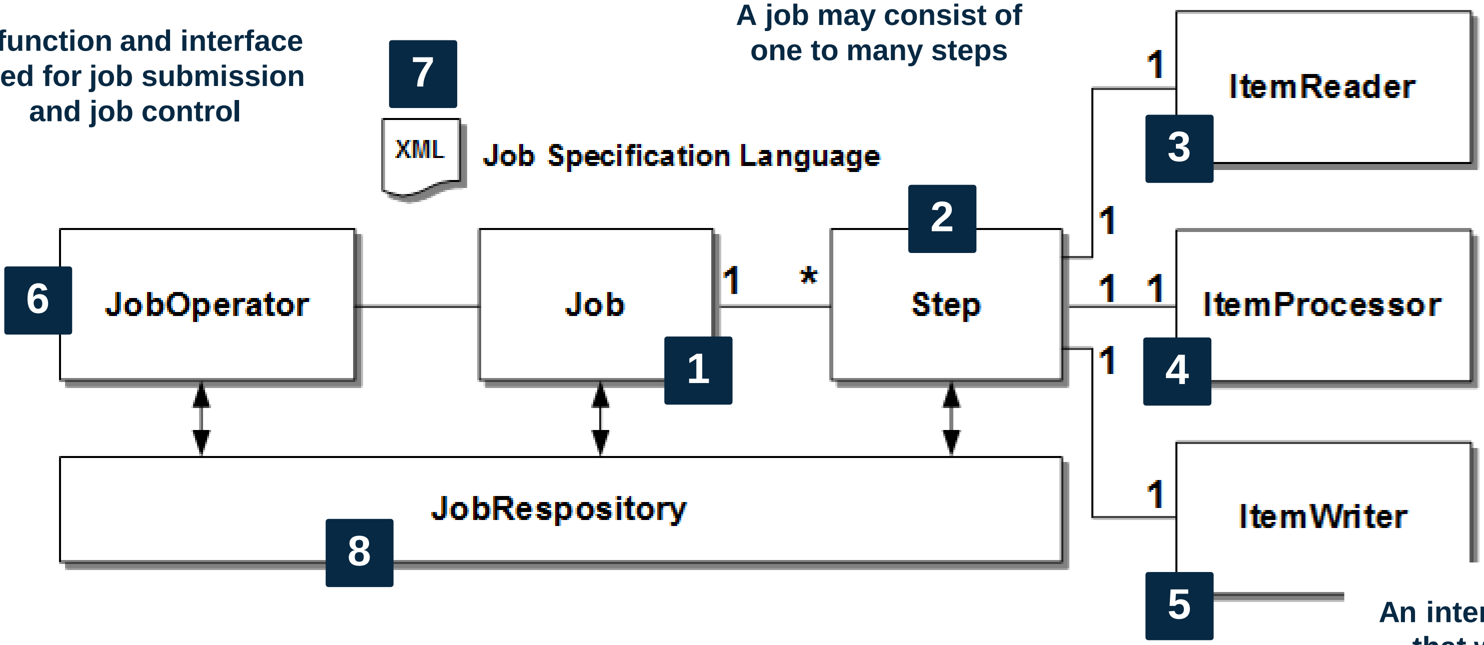
A file that declares the specifics of the job and the steps contained in the job

A job is a logical representation of the batch processing

An interface to code that reads data

A job may consist of one to many steps

A function and interface used for job submission and job control



An interface to code that processes data

An interface to code that writes data

A mechanism to persist information about the state of jobs in the environment. For example, a set of relational database tables.

You'll see how this is implemented in an upcoming section of this presentation

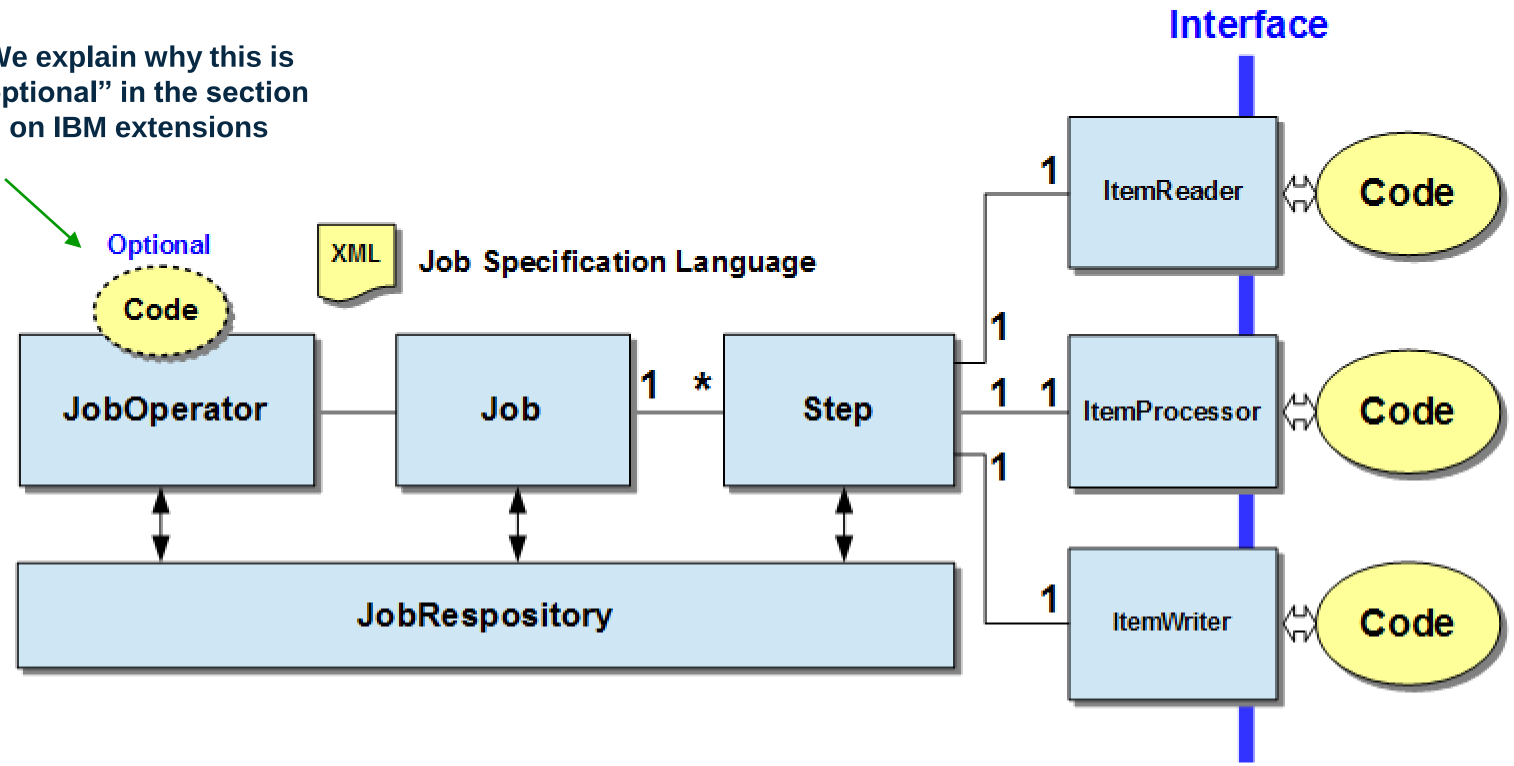
Complet

www.SHARE.org/Orlando-Eval

SHARE in Orlando 2015

How Much of that Picture Do *I* Have to Code?

We explain why this is "optional" in the section on IBM extensions



It turns out … relatively little

Much of the processing is handled by the vendor implementation of the JSR 352 standard. Your code sits behind standard interfaces and is called by the JSR 352 runtime.

Job Step Types – Chunk and Batchlet

Chunk Step

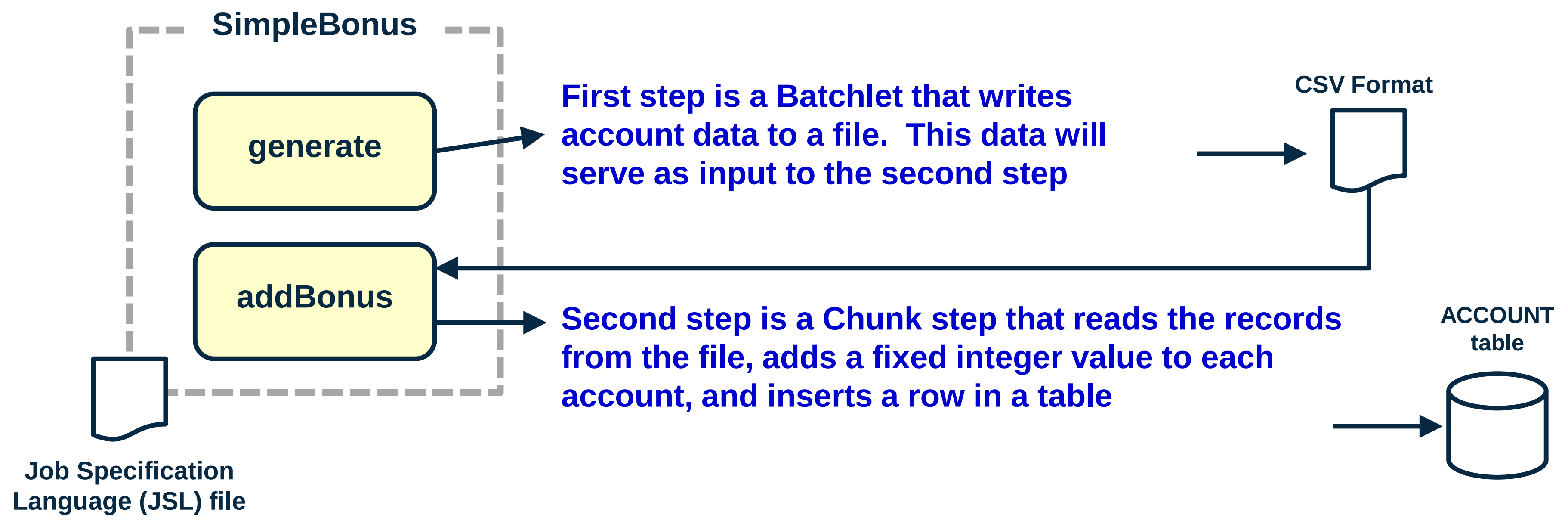
- What we typically think of as a “batch job” – an iterative loop through data, with periodic commits of data written out during processing
- This involves the ItemReader, ItemProcessor and ItemWriter interfaces shown earlier.

Batchlet Step

- A job step with much less structure … it is called, it runs and does whatever is in the code, and ends
- This job step type is useful for operations that are not iterative in nature, but may take some time … a large file FTP, for example
- This is also useful for encapsulating existing Java `main()` programs into the JSR 352 model

A multi-step job may consist of either … or both

High-Level Example … to Illustrate the Key Concepts



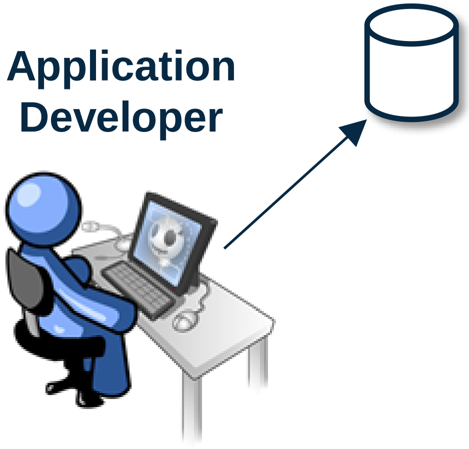
First step is a Batchlet that writes account data to a file. This data will serve as input to the second step

Second step is a Chunk step that reads the records from the file, adds a fixed integer value to each account, and inserts a row in a table

Not real-world, but useful to illustrate essential JSR 352 concepts. What does packaging look like?

A Peek Inside the Sample Application WAR file

Application Developer



```
BonusPayout-1.0.war
 WEB-INF
  ├\classes\com\ibm\websphere\samples\batch
  │  ├── \artifacts
  │  │     GenerateDataBatchlet.class
  │  │     GeneratedCSVReader.class
  │  │     BonusCreditProcessor.class
  │  │     AccountJDBCWriter.class
  │  │     (other class files)
  │  ├── \beans
  │  │     (data bean class files)
  │  └── \util
  │        (utility class files)
  └\classes\META-INF\batch-jobs
      SimpleBonusPayoutJob.xml
```

Step 1 Batchlet

Step 2 Chunk
ItemReader
ItemProcessor
ItemWriter

The "How to write JSR 352 applications" topic is important, but outside the scope of this overview discussion.

The "Job Specification Language" (JSL) file, which we'll look at next …

This deploys into the Liberty Profile server's `/dropins` directory, or pointed to with `<application>` tag like any other application

JSL: Job Specification Language, Part 1

Properties are a way to get values into your batch job. They can be specified in the JSL as shown, and overridden at submission time using IBM's REST interface (shown later)

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="SimpleBonusPayoutJob">

  <properties>
    <property name="numRecords" value="#{jobParameters['numRecords']}?:1000;" />
    <property name="chunkSize" value="#{jobParameters['chunkSize']}?:100;" />
    <property name="dsJNDI" value="#{jobParameters['dsJNDI']}?:java:comp/env/jdbc/BonusPayoutDS;" />
    <property name="bonusAmount" value="#{jobParameters['bonusAmount']}?:100;" />
    <property name="tableName" value="#{jobParameters['tableName']}?:BONUSPAYOUT.ACCOUNT;" />
  </properties>

  <step id="generate" next="addBonus">
    <batchlet ref="com.ibm.websphere.samples.batch.artifacts.GenerateDataBatchlet">
      <properties>
        <property name="numRecords" value="#{jobProperties['numRecords']}" />
      </properties>
    </batchlet>
  </step>
        :
```

(second part on next chart)

The first step is defined as a Batchlet. The Java class file that implements the Batchlet is indicated. The property to tell the Batchlet how many records to create is specified.

The job specification is taking shape. What about the second step? That's shown next …

JSL: Job Specification Language, Part 2

(first part on previous chart)

```
  :
<step id="addBonus">
  <chunk item-count="#{jobProperties['chunkSize']}">

    <reader ref="com.ibm.websphere.samples.batch.artifacts.GeneratedCSVReader"/>

    <processor ref="com.ibm.websphere.samples.batch.artifacts.BonusCreditProcessor">
      <properties>
        <property name="bonusAmount" value="#{jobProperties['bonusAmount']}" />
      </properties>
    </processor>

    <writer ref="com.ibm.websphere.samples.batch.artifacts.AccountJDBCWriter">
      <properties>
        <property name="dsJNDI" value="#{jobProperties['dsJNDI']}" />
        <property name="tableName" value="#{jobProperties['tableName']}" />
      </properties>
    </writer>

  </chunk>
</step>

</job>
```

The second step is defined as a Chunk step. The "chunkSize" (commit interval) is a property from earlier.

The "reader," "processor" and "writer" Java classes are specified

A property on the processor provides the integer bonus to add to each account. Properties on the writer indicate how to reach the database and what table to use

Summary: the JSR 352 runtime provides the infrastructure to run batch jobs; this JSL tells it what Java classes to use and other details related to the operation of the job

Parallel Job Processing

- Splits and Flows provide a mechanism for executing job steps concurrently at the orchestration layer
- A flow is a sequence of one or more steps which execute sequentially, but as a single unit.
- A Split is a collection of flows that may execute concurrently
  - A split may only contain “flows”; a step is not implicitly a flow
- This is done entirely in the JSL descriptor
  - Imposed on the batch application with no code changes!

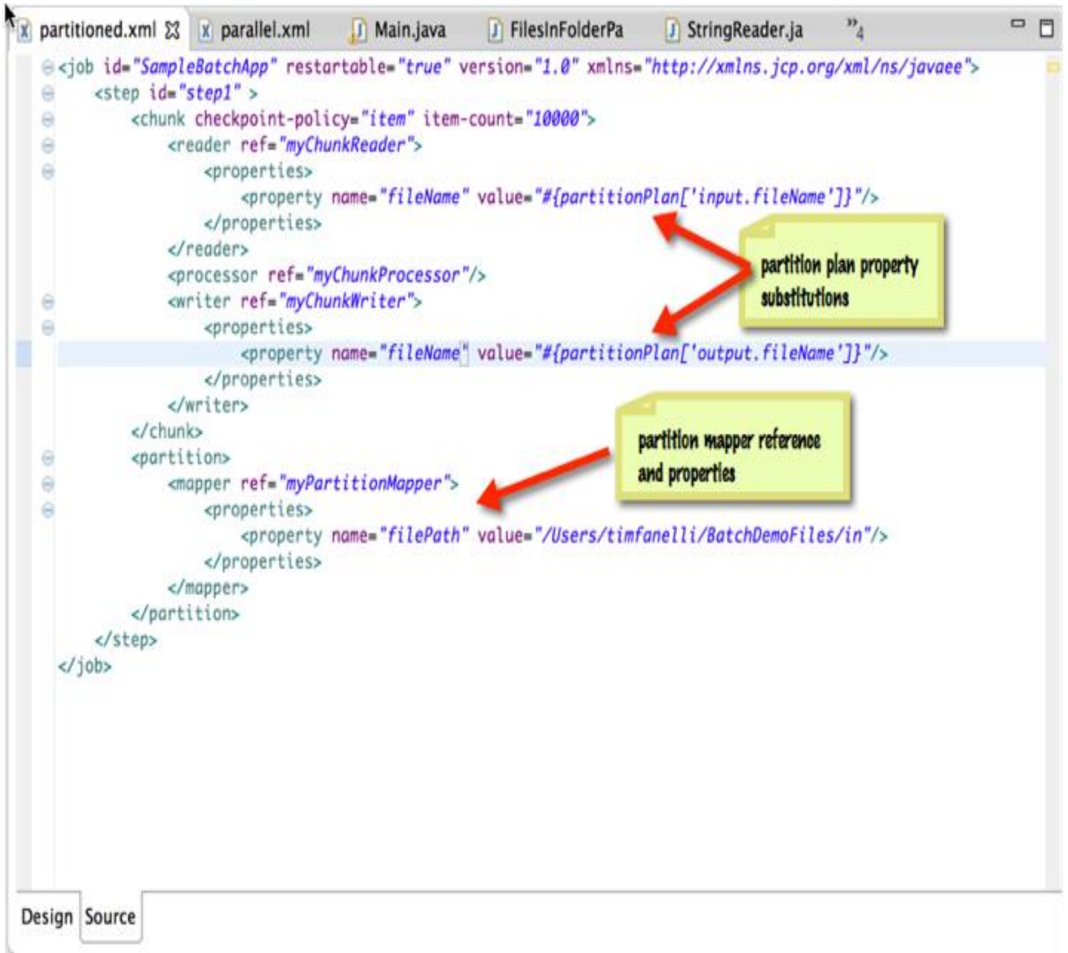
Step-level parallelism

- Step-level parallelism can be achieved programmatically using step partitioning
- A partitioned step runs as multiple instances with distinct property sets
- PartitionMapper defines the number of partitions, and property values for each partition
  - Can be a fixed set of partitions in JSL
  - Can be dynamic using a PartitionMapper implementation

Step-level parallelism

- No new Java artifacts
- Not necessarily the general case
- Might have to
  - Coalesce Exit Status (PartitionAnalyzer)
  - Process Intermediate results on parent thread (PartitionCollector->PartitionAnalyzer)
  - Perform other tasks on end of partition (PartitionReducer)
- Also might want to programmatically partition (PartitionMapper) rather than via JSL

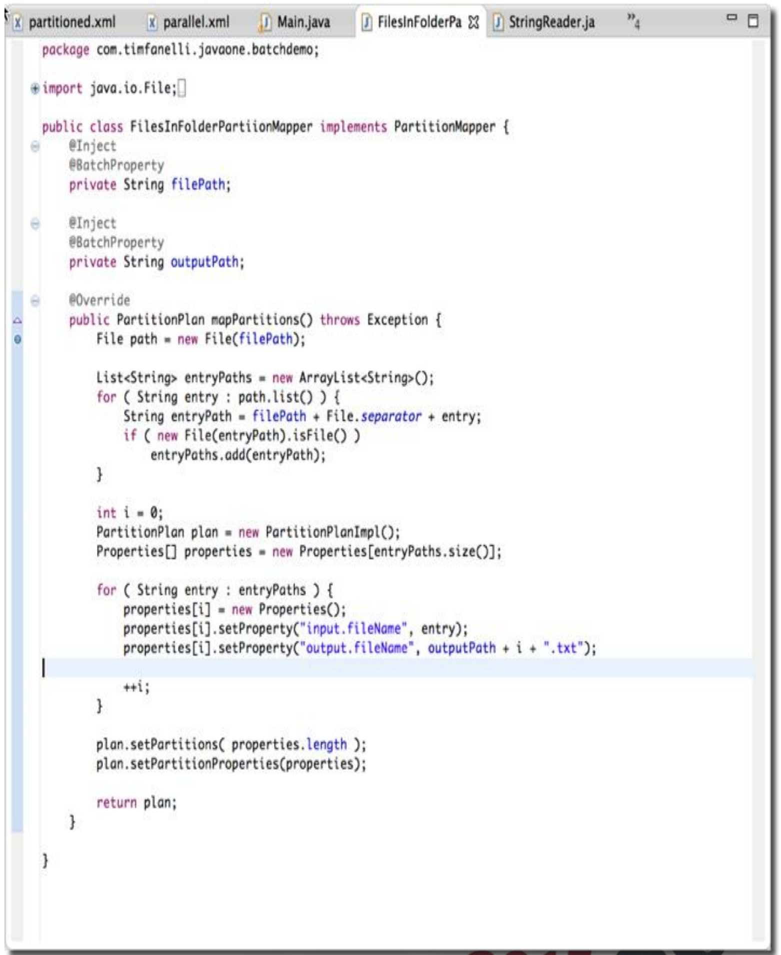
Parallel Job Processing



```
<job id="SampleBatchApp" restartable="true" version="1.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee">
    <step id="step1" >
        <chunk checkpoint-policy="item" item-count="10000">
            <reader ref="myChunkReader">
                <properties>
                    <property name="fileName" value="#{partitionPlan['input.fileName']}"/>
                </properties>
            </reader>
            <processor ref="myChunkProcessor"/>
            <writer ref="myChunkWriter">
                <properties>
                    <property name="fileName" value="#{partitionPlan['output.fileName']}"/>
                </properties>
            </writer>
        </chunk>
        <partition>
            <mapper ref="myPartitionMapper">
                <properties>
                    <property name="filePath" value="/Users/timfanelli/BatchDemoFiles/in"/>
                </properties>
            </mapper>
        </partition>
    </step>
</job>
```

partition plan property substitutions

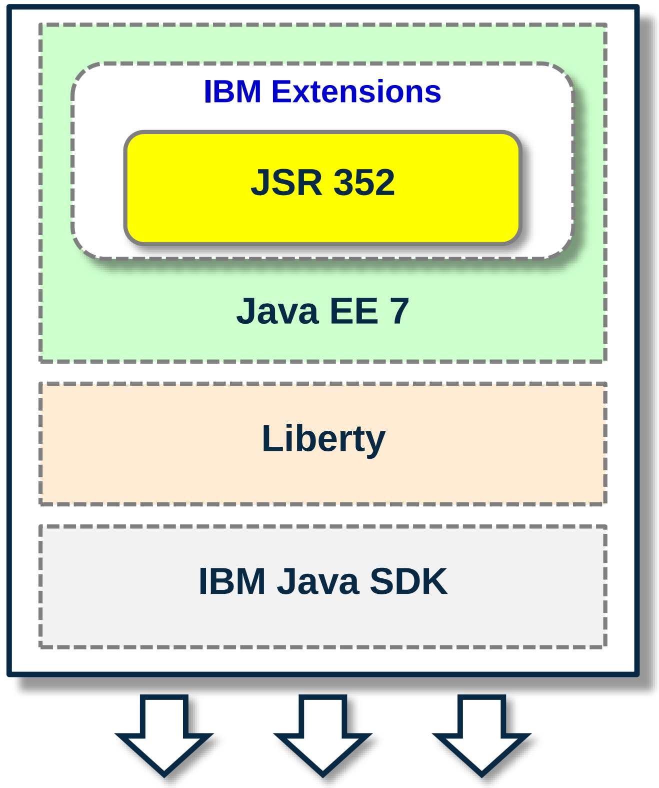
partition mapper reference and properties



```
package com.timfanelli.javaone.batchdemo;

import java.io.File;

public class FilesInFolderPartiionMapper implements PartitionMapper {
    @Inject
    @BatchProperty
    private String filePath;

    @Inject
    @BatchProperty
    private String outputPath;

    @Override
    public PartitionPlan mapPartitions() throws Exception {
        File path = new File(filePath);

        List<String> entryPaths = new ArrayList<String>();
        for ( String entry : path.list() ) {
            String entryPath = filePath + File.separator + entry;
            if ( new File(entryPath).isFile() )
                entryPaths.add(entryPath);
        }

        int i = 0;
        PartitionPlan plan = new PartitionPlanImpl();
        Properties[] properties = new Properties[entryPaths.size()];

        for ( String entry : entryPaths ) {
            properties[i] = new Properties();
            properties[i].setProperty("input.fileName", entry);
            properties[i].setProperty("output.fileName", outputPath + i + ".txt");

            ++i;
        }

        plan.setPartitions( properties.length );
        plan.setPartitionProperties(properties);

        return plan;
    }

}
```

IBM Implementation And Extensions

Built on Liberty as the Java Runtime Server



All Platforms Supported By Liberty

Liberty 8.5.5.6 and above

- IBM's fast, lightweight, composable server runtime
- Dynamic configuration and application updates

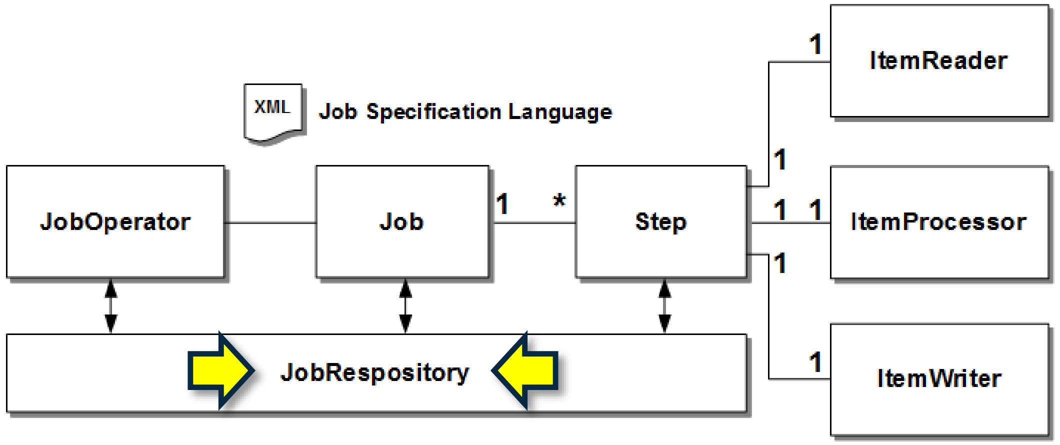
JVM Stays Active Between Jobs

- Avoids the overhead of JVM initialize and tear down for each job

IBM Extensions to JSR 352

- JSR 352 is largely a *programming* standard
- IBM extensions augment this with valuable *operational* functions
- Includes:
  - Job logs separated by job execution
  - REST interface to JobOperator
  - Command line client for job submission
  - Integration with enterprise scheduler functions
  - Multi-JVM support: dispatcher and endpoint servers provide a distributed topology for batch job execution

JobRepository Implementation



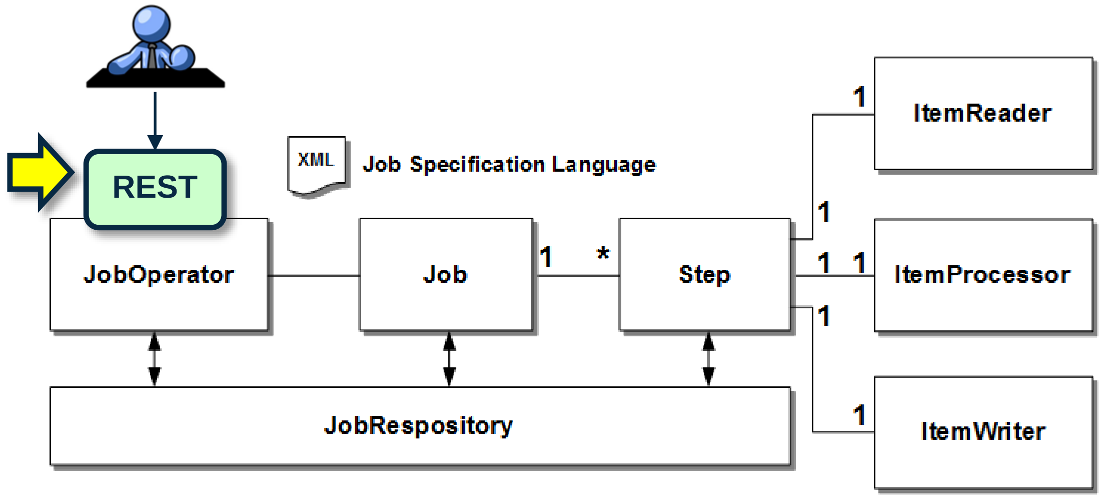
The JSR 352 standard calls for a JobRepository to hold job state information, but it does not spell out implementation details

IBM JSR 352 provides three options for this:

1. **An in-memory JobRepository**
   For development and test environments where job state does not need to persist between server starts
2. **File-based Derby JobRepository**
   For runtime environments were a degree of persistence is desired, but a full database product is not needed
3. **Relational database product JobRepository**
   For production and near-production environments where a robust database product is called for

Table creation is automatic. Relatively easy to drop one set of tables and re-configure to use a different data store.

REST Interface to JobOperator



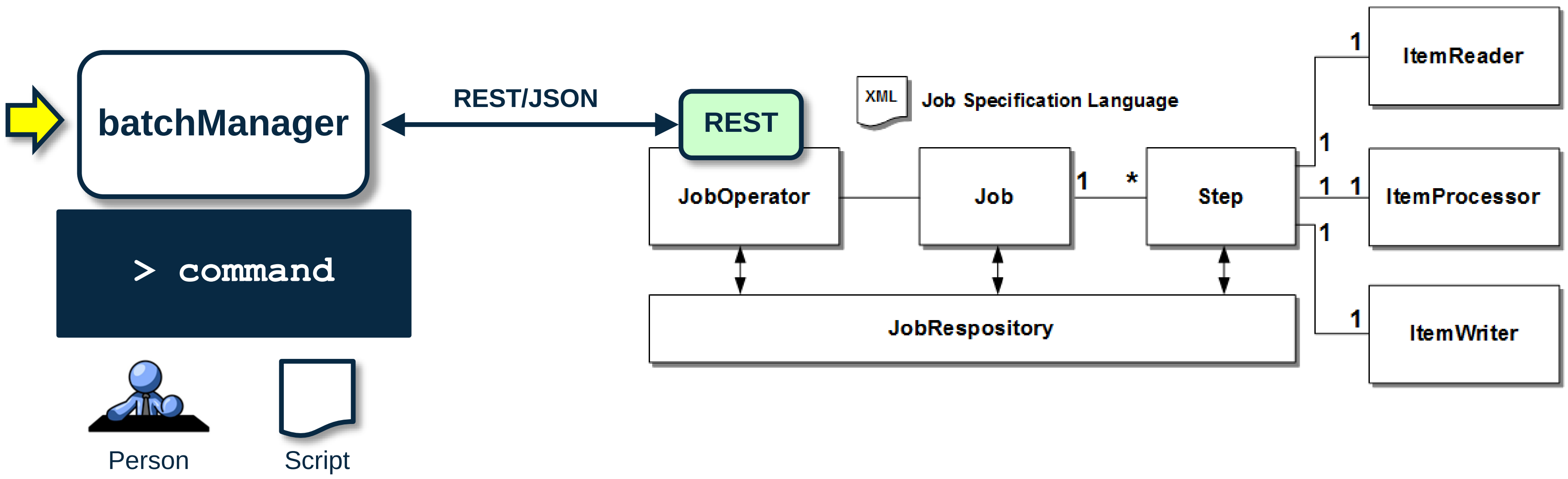
The JSR 352 standard calls for a JobOperator *interface*, but leaves to vendors to implement function to handle external requests for job submission, control and monitoring

The IBM JSR 352 REST interface provides:

1. **A RESTful interface for job submission, control and monitoring**
   Job submission requests may come from outside the Liberty Profile runtime
2. **Security model for authentication and authorization**
   Authorization is role-based: administrator, submitter, monitor
3. **JSON payload carries the specifics of the job to be submitted**
   With information such as the application name, the JSL file name, and any parameters to pass in

This permits the remote submission and control of jobs; it provides a way to integrate with external systems such as schedulers

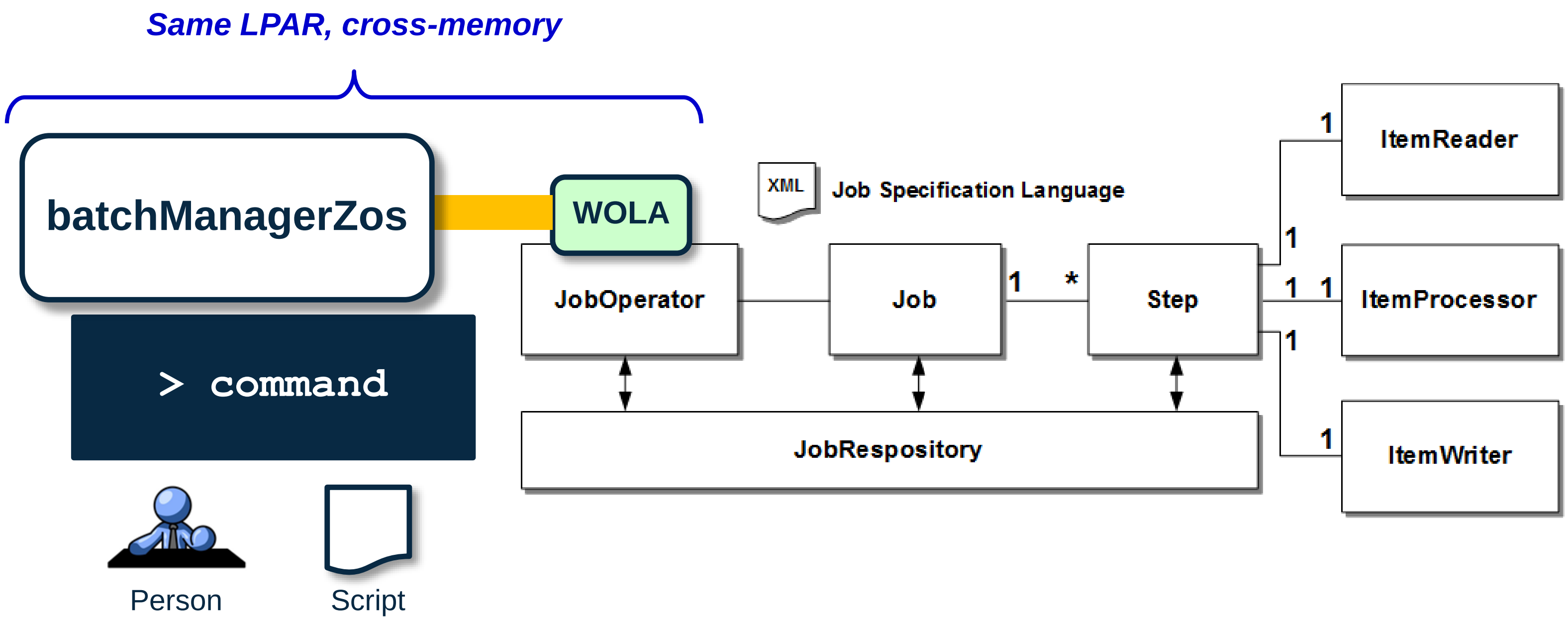
Command Line Client to REST Interface



The batchManager command line interface client provides:

1. **A way to submit, monitor and control jobs remotely using a command line interface**
   On the same system, or a different system … different OS … doesn't matter: TCP/IP and REST/JSON
2. **Uses the REST interface on the IBM Java Batch server**
   Which means the same security model is in effect: SSL, authentication, role-based access
3. **External schedulers can use this to submit and monitor job completion**
   batchManager parameters allow the script to "wait" for Java to complete. Parameters allow for discovery of job log information, and a mechanism to retrieve the job log for archival if desired.

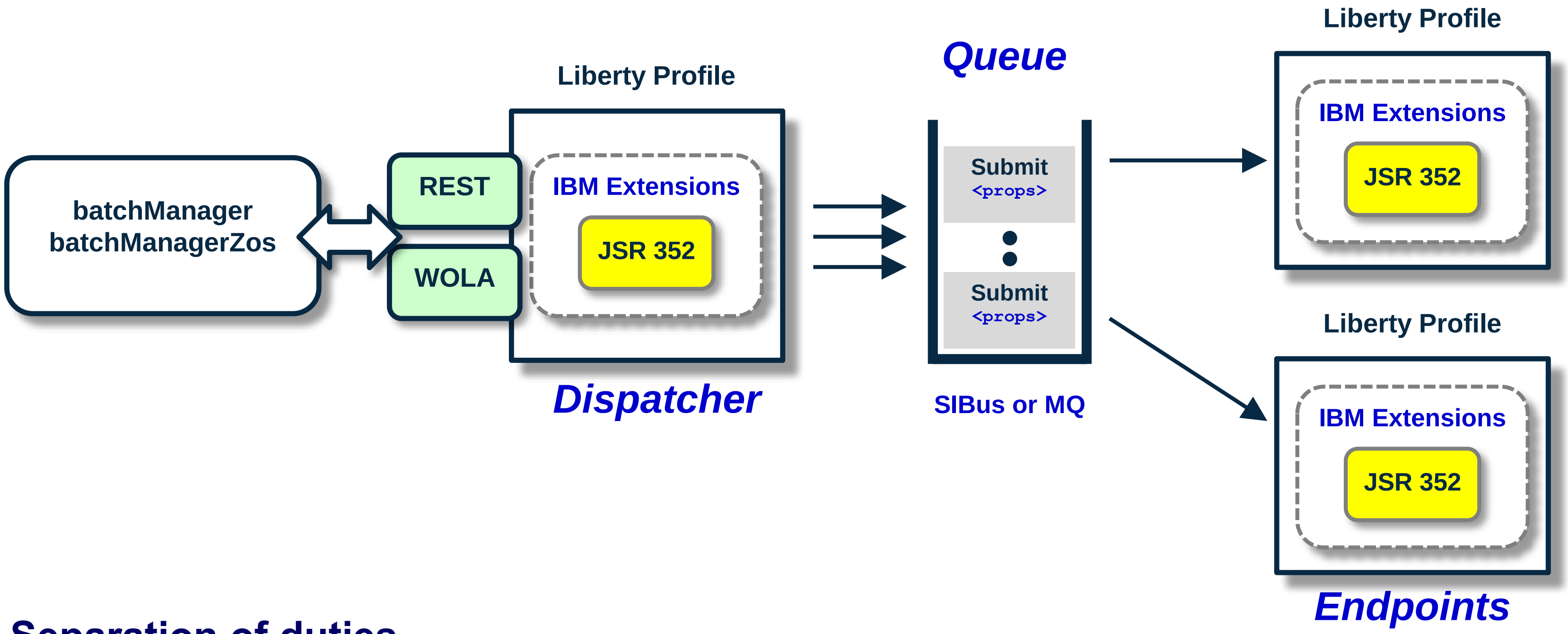
z/OS: Native Program Command Line Interface



Same batchManager command line function, but …

1. Not a Java client, so do not need to spin up a JVM for each invocation
   Saves the CPU associated with initiating the JVM, and when there's a lot of jobs this can be significant
2. Cross-memory
   Very low latency, and since no network then no SSL and management of certificates
3. Same access security model
   Once the WOLA connection is established, the same "admin," "submitter" and "monitor" roles apply

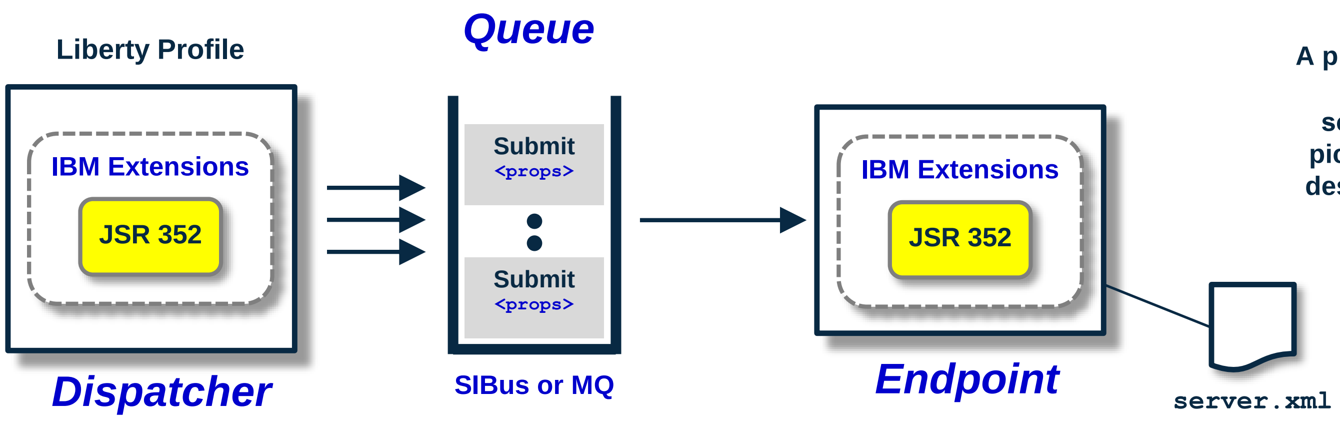
Multi-JVM Support: Job Dispatchers, End-Points



Separation of duties …

1. **Server designated as dispatchers handle job requests, and places them on JMS queue**
   The endpoints listen on the JMS queues and pick up the job submission request based on criteria you set to indicate which jobs to pick up (*more on that next chart*)
2. **Endpoint servers run the batch jobs**
   Deploy the batch jobs where most appropriate; co-locate some batch jobs and others have their own server
3. **JMS queues (either Service Integration Bus or MQ) serve as integration between two**
   This provides a mechanism for queuing up jobs prior to execution

Multi-JVM Support: Get Jobs Based on Endpoint Criteria



A property in the `server.xml` defines the "message selector" criteria to use to pick up messages. You can designate – by server – what criteria to use.

```
… messageSelector="com_ibm_ws_jbatch_applicationName = 'BatchJobA'"
```

1

```
… messageSelector="com_ibm_ws_jbatch_applicationName = 'BatchJobA'
                 AND com_ibm_ws_jbatch_myProperty = 'myValue'"
```

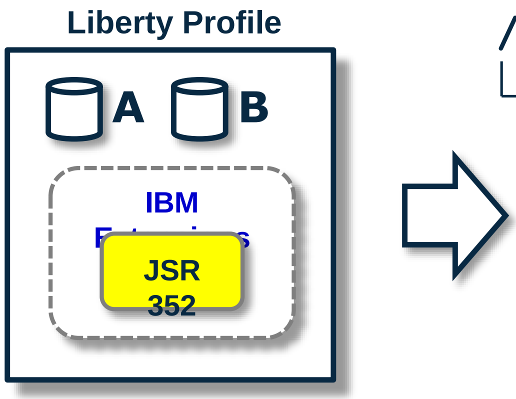
2

Submit jobs and have them run only when intended server starts and picks up the submission request

Have jobs run in intended servers based on selection criteria of your choice

Not limited to system, not limited to platform … may span systems and platforms

Job Logging



```
/<server_directory>/logs
└── /joblog
    ├── /<application_name_A>
    │   └── /<date>
    │       └── /instance.#
    │           └── /execution.#
    │                   part.#.log
    └── /<application_name_B>
        └── <date>
                etc.
```

Job logs separate from the server log, separate from each other

1. **Each job's logs are kept separate by application name, date, instance and execution**
2. **The IBM JSR 352 REST interface has a method for discovery and retrieval of job logs**
   This is accessible through the batchManager command line interface as well. This is how job log retrieval and archival can be achieved if needed.

Liberty `server.xml`

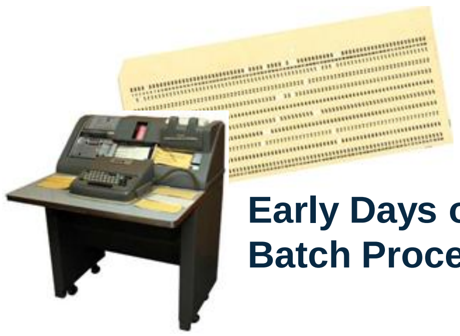
```
    :
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
</featureManager>
    :
<batchPersistence databaseStoreRef="BatchDatabaseStore" />
<databaseStore id="BatchDatabaseStore"
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />
    :
```

Relatively simple updates to `server.xml` …

1. The `batch-1.0` feature enables the JSR 352 core functionality
2. The `batchManagement-1.0` feature enables the REST interface, job logging, and the ability to configure the multi-JVM support.
3. The `<batchPersistence>` element provides information about where the JobRepository is located

Some details left out of this chart, of course … but the key point is that configuring the support is based on updates to `server.xml`

Overall Summary



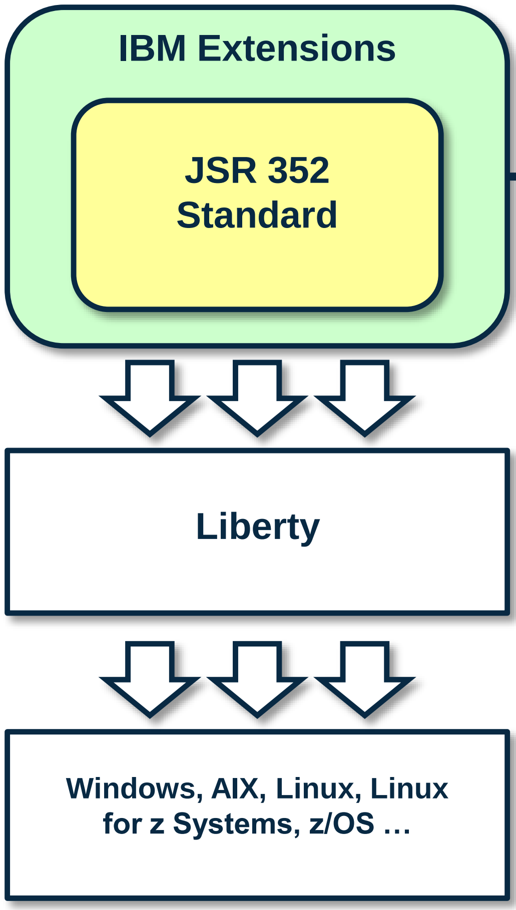
Early Days of
Batch Processing

Over time …

Modernization

Java

JSR 352



- Multiple JobRepository
- Job logging
- REST interface
- Command line client
- z/OS: native client
- Multi-JVM capability

IBM JSR 352 Java Batch

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

SHARE in Orlando 2015