



IBM Software Group

Java Batch Modernization using WebSphere Compute Grid v6.1.1

Technical Overview

An IBM Proof of Technology



Agenda

- What is Batch processing?
- IBM® WebSphere® Compute Grid Summary
 - ▶ Features
 - ▶ Architecture
 - ▶ Design consideration for high performance batch
- Example Customer Implementations
 - ▶ SwissRe
 - ▶ Typical Distributed Customer
- Future directions

What is batch computing?

batch processing *system for processing data with little or no operator intervention. This allows efficient use of the computer and is well suited to applications of a repetitive nature, such as file format conversion, payroll, or the production of utility bills.*

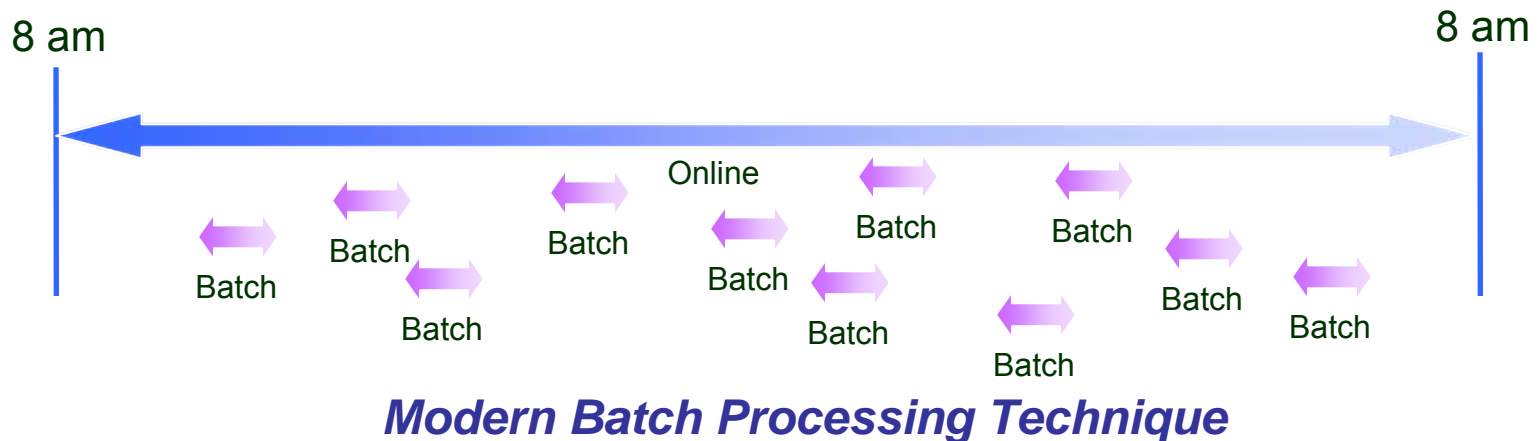
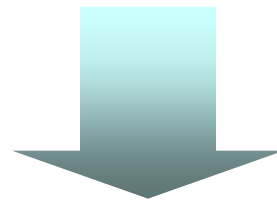
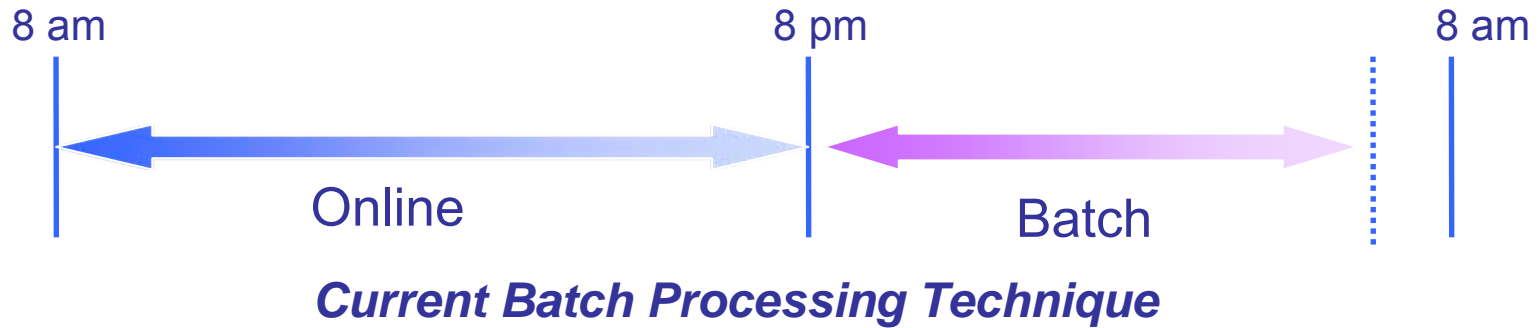
*In **interactive computing**, by contrast, data and instructions are entered while the processing program is running.*

Hutchinson Encyclopedia

- **By eliminating human reaction time, batch can process in one hour what Online Transaction Processing (OLTP) would take one month (or require 50,000 people)**
- **Batch is part of a continuous spectrum of workload**
- **Enterprise clients run a continual mix of online and batch – response to global, 24x7 business**
- **Business events as a trigger for batch – for example, end-of-day is a hallmark of a critical workload**



24x7 Batch and OLTP



WebSphere Compute Grid - Product Summary

- Provides container-managed services such as **checkpoint strategies, restart capabilities,** and threshold policies that govern the execution of batch jobs.
- **Provides a parallel processing infrastructure** for partitioning, dispatching, managing and monitoring parallel batch jobs.
- **Enables the standardization of batch processing across the enterprise;** stamping out homegrown, maverick batch infrastructures and integrating the control of the batch infrastructure with existing enterprise schedulers, disaster recovery processes, archiving, and auditing systems.
- Delivers a workload-managed batch processing platform, enabling **24x7 combined batch and OLTP capabilities.**
- Java™ **Plain-Old-Java-Object (POJO)-based application development** with **end-to-end development tooling,** libraries, and patterns for sharing business services across OLTP and batch execution paradigms.

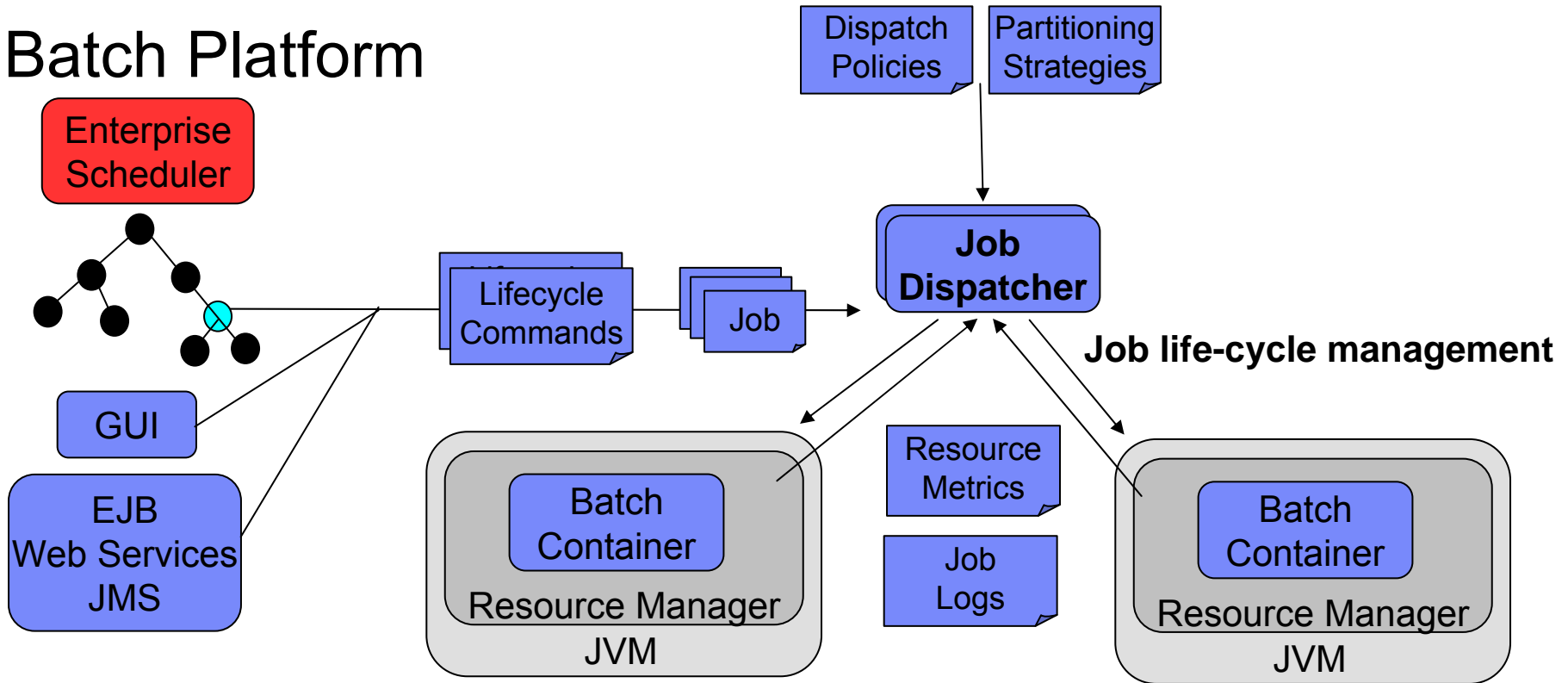


WebSphere Compute Grid – Unique Features

- **Uses J2EE™ Application Server**
 - Transactions
 - Security
 - High availability including dynamic servants
 - High availability of scheduler and end points
 - Takes advantage of the inherent WebSphere Application Server quality of service
 - Connection Pooling
 - Thread Pooling
- **Platform for executing transactional Java batch applications**
 - Checkpoint/restart
 - Batch data stream management
 - Parallel job execution
 - Operational control
 - External scheduler integration
 - IBM z/OS® System Management Facilities records for batch
 - IBM z/OS Workload Management Facility Integration



Batch Platform



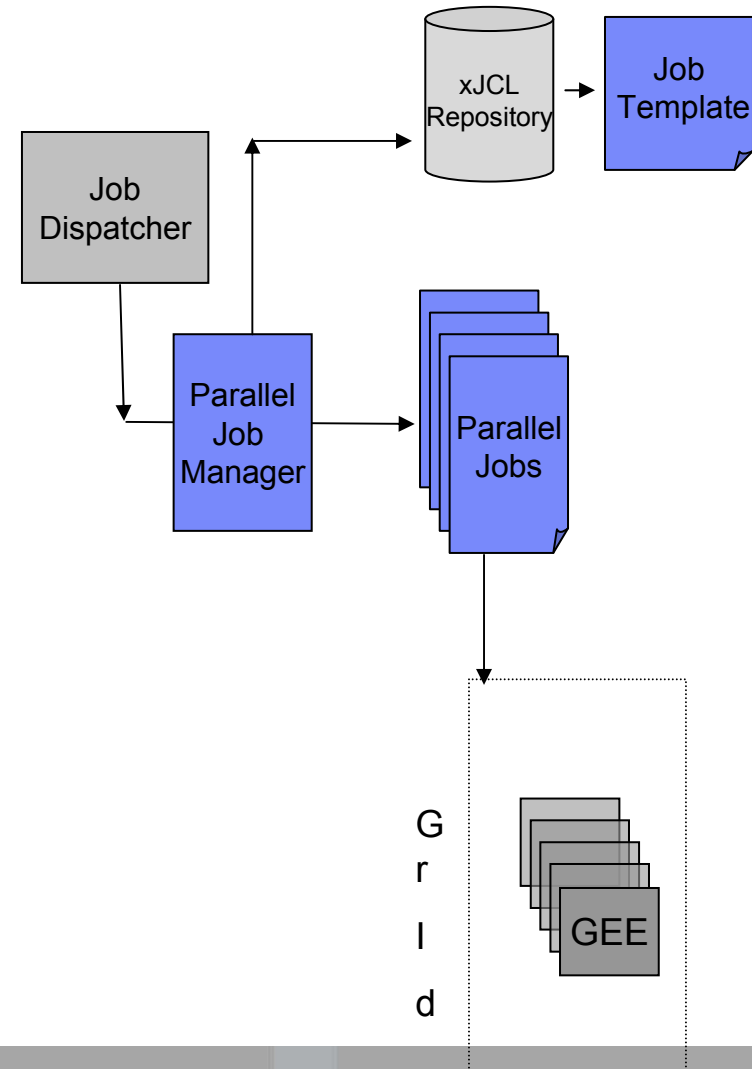
- Jobs are submitted to the system using an **enterprise scheduler**, **process server**, Job Dispatcher **GUI**, or **programmatically** through Enterprise JavaBeans™ (EJB), JMS, or web services
- Administrator can define dispatch and partitioning policies
- Job Dispatcher selects the best endpoint for job execution based on execution metrics
- Job Dispatcher aggregates job logs and provides job control (submit, cancel, restart, suspend)

Enterprise Features at a glance...

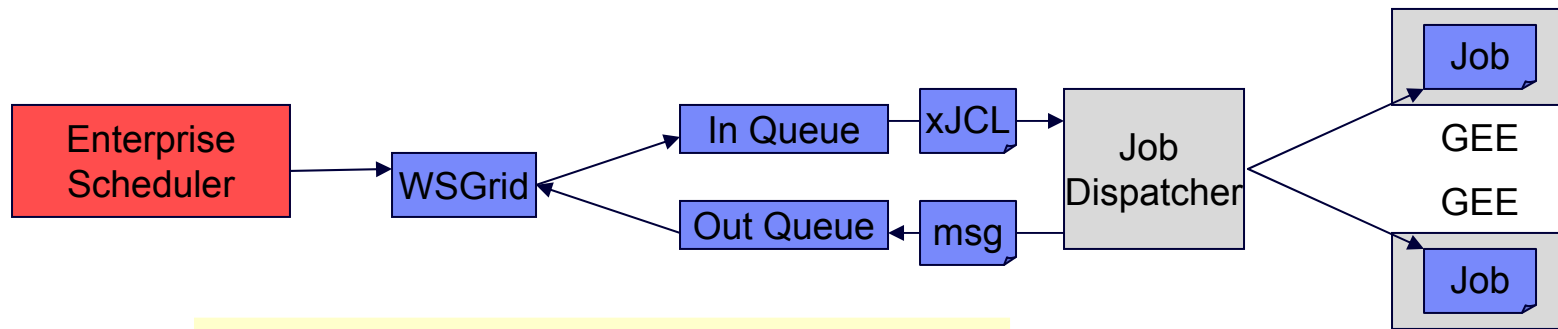
- Check Point Restart
 - ▶ Container-Managed **Checkpoint Strategies**
 - Keep track of the current input and output positions on behalf of the batch step
 - Provide flexible options: Time-based, Record-based, Custom algorithms
 - ▶ Container-Managed **Restart Capabilities**
 - Seek to the correct positions in the input and output streams
 - Restart should be **transparent** to the application
 - ▶ Dynamically adjust the checkpoint strategies based on Workload Management metrics, OLTP load, and application priorities
- Integrated Operation control
 - ▶ Provide an operational infrastructure for starting/stopping/canceling/restarting/etc batch jobs
 - ▶ Integrate with existing enterprise schedulers such as Tivoli Workload Scheduler
 - ▶ Provide log management and integration with archiving and auditing systems
 - ▶ Provide resource usage monitoring
 - ▶ Integrate with existing security and disaster recovery procedures
- High Availability
 - ▶ Clustered Job Scheduler
 - ▶ Clustered Endpoints
- Disaster recovery through multisite topology

Compute Grid Components

- **Job Scheduler/Dispatcher (JS)**
 - ▶ The job entry point to Compute grid
 - ▶ Jobs defined using XML Job Control Language (xJCL) that can be submitted directly or from a repository maintained by the dispatcher
 - ▶ Job life-cycle management (Submit, Stop, Cancel and others) and monitoring
 - ▶ Dispatches workload to either the parallel job manager (PJM) or grid endpoints (GEE)
 - ▶ Hosts the Job Management Console (JMC)
- **Parallel Job Manager**
 - ▶ Breaks large batch jobs into smaller partitions for parallel execution
 - ▶ Provides job life-cycle management (Submit, Stop, Cancel, Restart) for the single logical job and each of its partitions
 - ▶ Is not a required component in compute grid
- **Grid Endpoints**
- **Executes the actual business logic of the batch job**



Enterprise Schedulers and WebSphere Compute Grid



Version 6.1.1 has a native implementation of the WSGrid Utility optimized for IBM z/OS

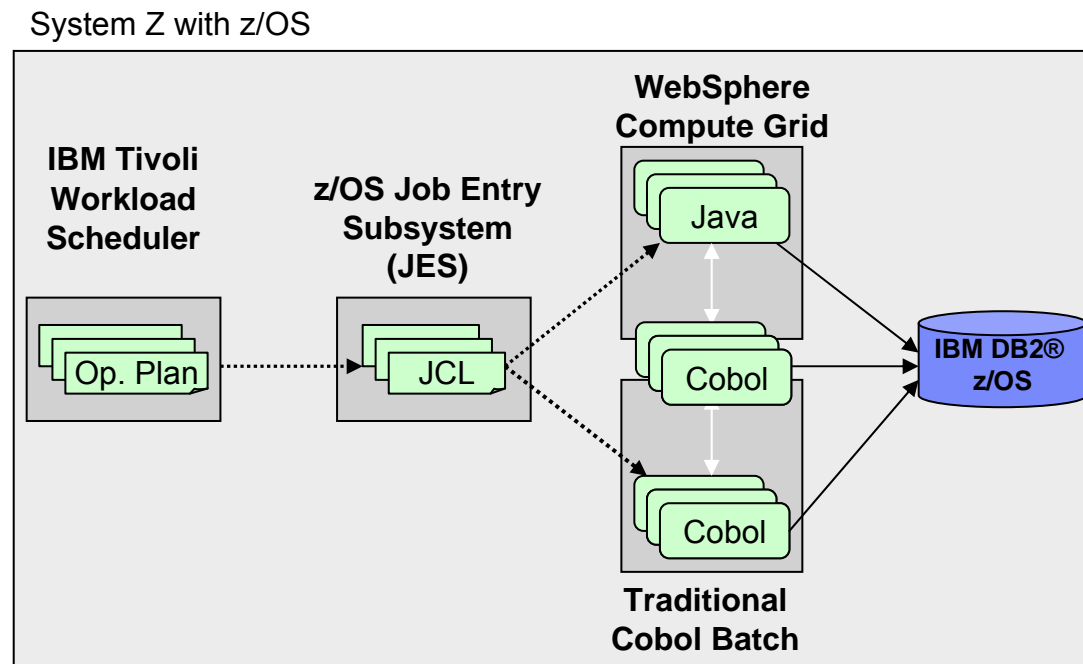
-WebSphere Compute Grid provides a utility named WSGrid that allows customers to submit jobs through the job dispatcher using from a wide variety of enterprise schedulers including IBM Tivoli® Workload Schedulers.

-To the enterprise scheduler, WSGrid behaves like an ordinary batch script that once submitted, retains control, directs log information to standard output and returns a completion code.

-While presenting this simple abstraction to the enterprise scheduler, WSGrid submits the job through the dispatcher and gathers and renders logs using asynchronous messaging with the job dispatcher. It returns the completion code of the WebSphere Compute Grid job as its own result code only after the job completes.



An Approach to Incremental Batch Modernization



Today: Executing traditional batch with COBOL

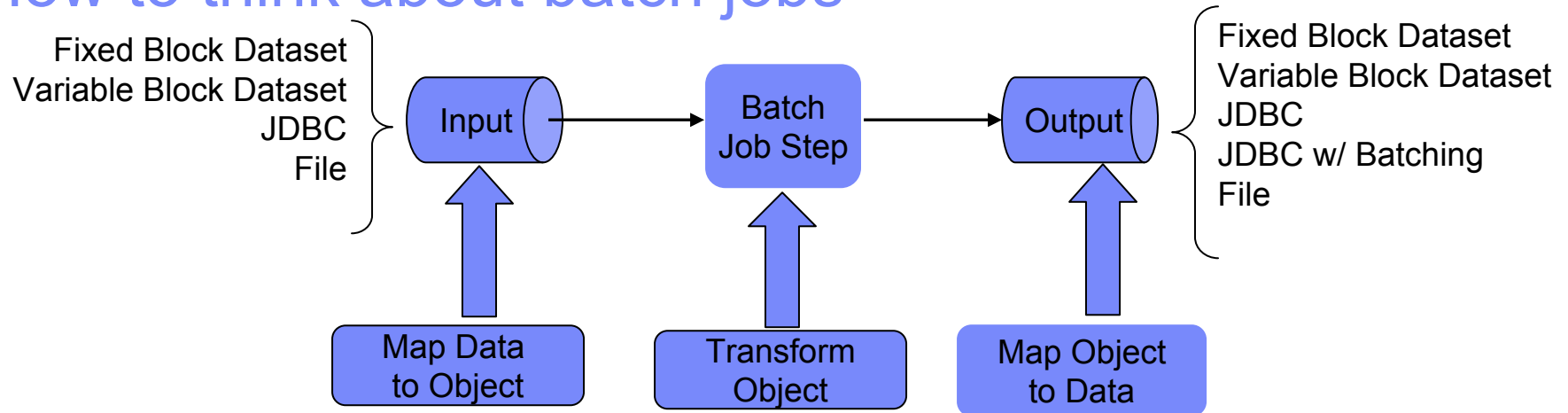
Phase 1: Implement new business logic in Java with XD Compute Grid

Phase 2: Share existing COBOL modules across both Java and COBOL domains

Phase 3: Incrementally migrate COBOL modules to Java with XD Compute Grid

Completion: All COBOL batch modules are replaced with Java, running in XD

How to think about batch jobs



- The better the contract between the container and the application, the more container-managed services can be provided.
- Batch Data Stream Framework (BDSFW) provides an application structure and libraries for building apps
 - Customer implements pattern interfaces for input/output/step
 - Pattern interfaces are *very* lightweight. They follow *typical* life-cycle activities:
 - I/O patterns: initialize, map raw data to single record, map single record to raw data, close
 - Step pattern: Initialize, process a single record, destroy.
 - Object transformation can be done in any technology that can be run within the application server.
- ***BDS Framework is the recommended approach for building applications.*** The customer is free to implement applications in many other ways though.

A job is made up of one or more steps.

Step

Two yellow horizontal bars representing input or output streams for a step.

Step

Two yellow horizontal bars representing input or output streams for a step.

Step

Two yellow horizontal bars representing input or output streams for a step.

Step

Two yellow horizontal bars representing input or output streams for a step.

StepName
Step Implementation Reference
Execution Condition (in terms of other steps)
Property <N=V>
Property <N=V>

Input Stream

Output Stream

Error Stream

Example xJCL for a Step

```
<job-step name="Step3">
<step-scheduling>
<returncode-expression step="Step2" operator="eq" value="4" />
</step-scheduling>
<jndi-name>ejb/OverdraftAccountPosting</jndi-name>
<checkpoint-algorithm-ref name="{checkpoint}" />
<results-ref name="jobsum" />

<!-- Step3 uses this bds to read the overdrawn accounts from Account Table in database -->
<batch-data-streams>
<bds>
<!-- this logical name is used by the batch step to retrieve a handle to the bds at runtime -->
<logical-name>dbread</logical-name>
<impl-class>com.ibm.websphere.samples.OverdraftInputStream</impl-class>
</bds>
</batch-data-streams>

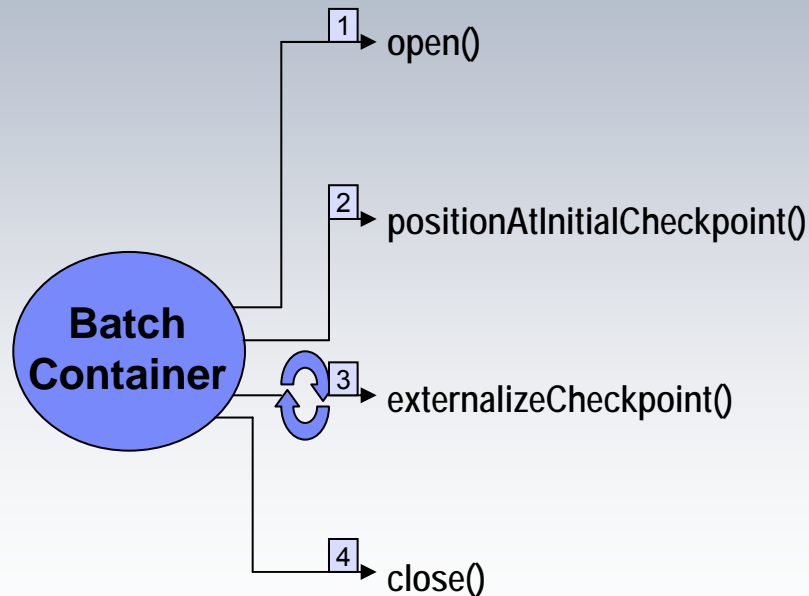
</job-step>
```



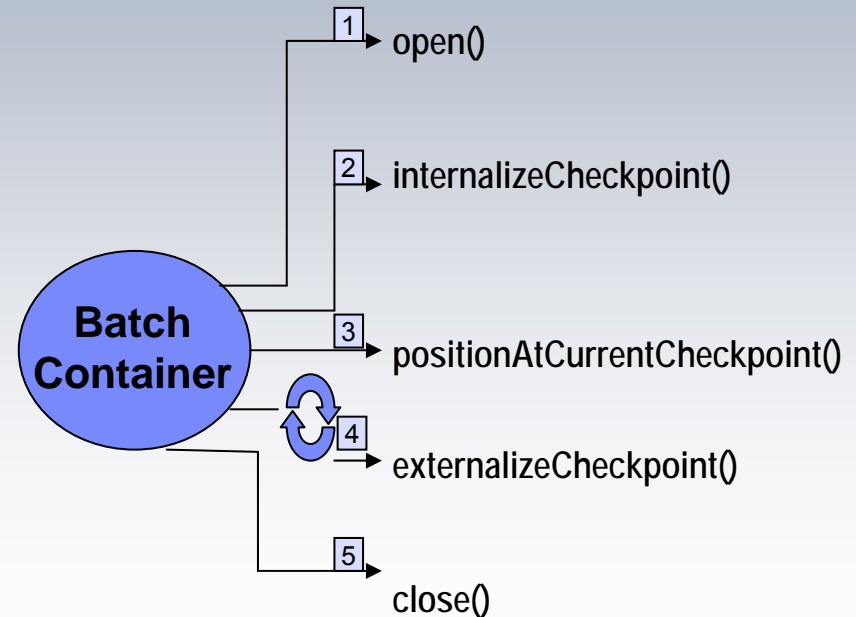
Checkpoint and Restart with Batch Data Streams

XD Compute Grid makes it easy for developers to encapsulate input/output data streams using POJOs that optionally support checkpoint/restart semantics.

Job Start

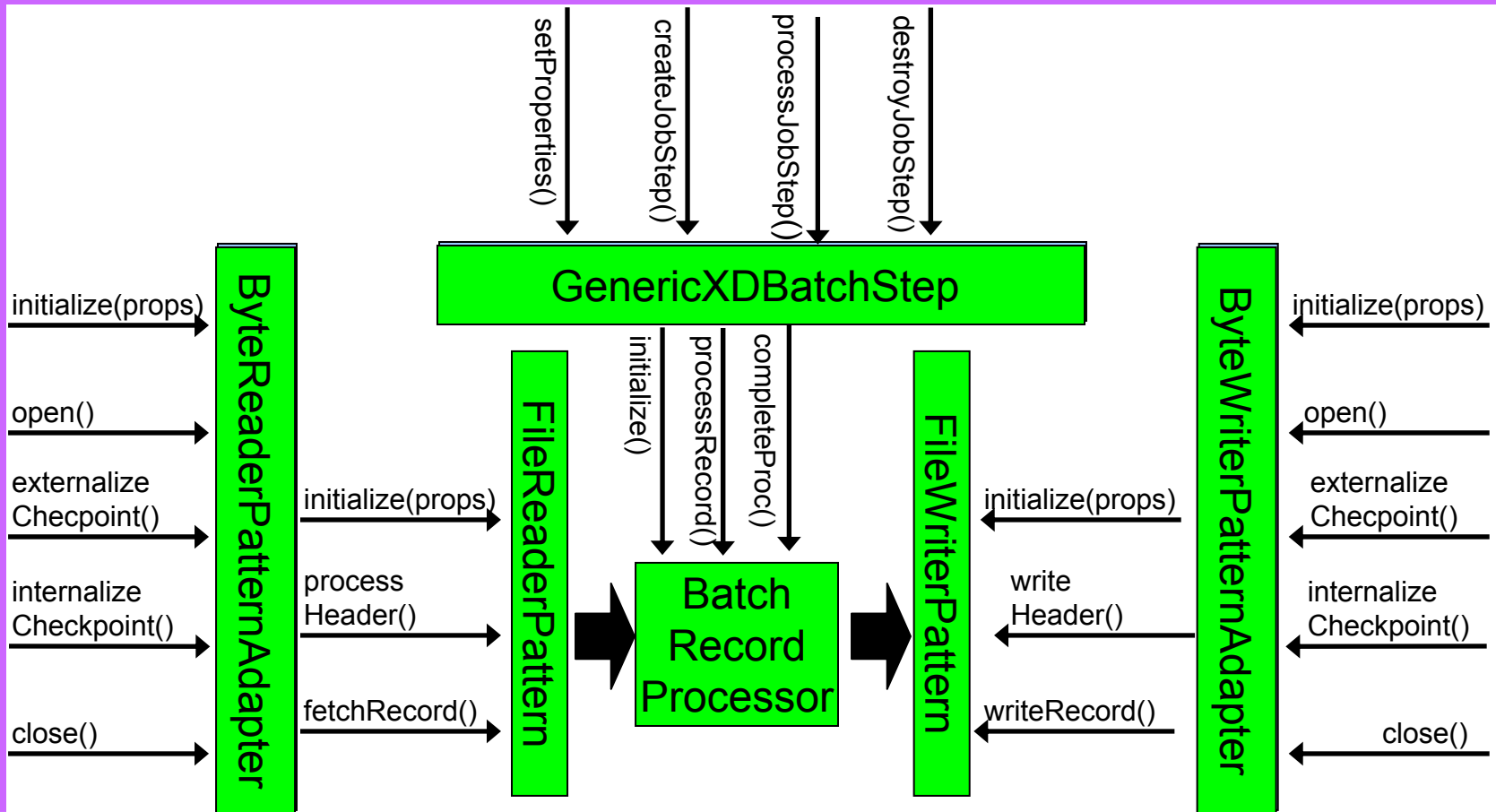


Job Restart

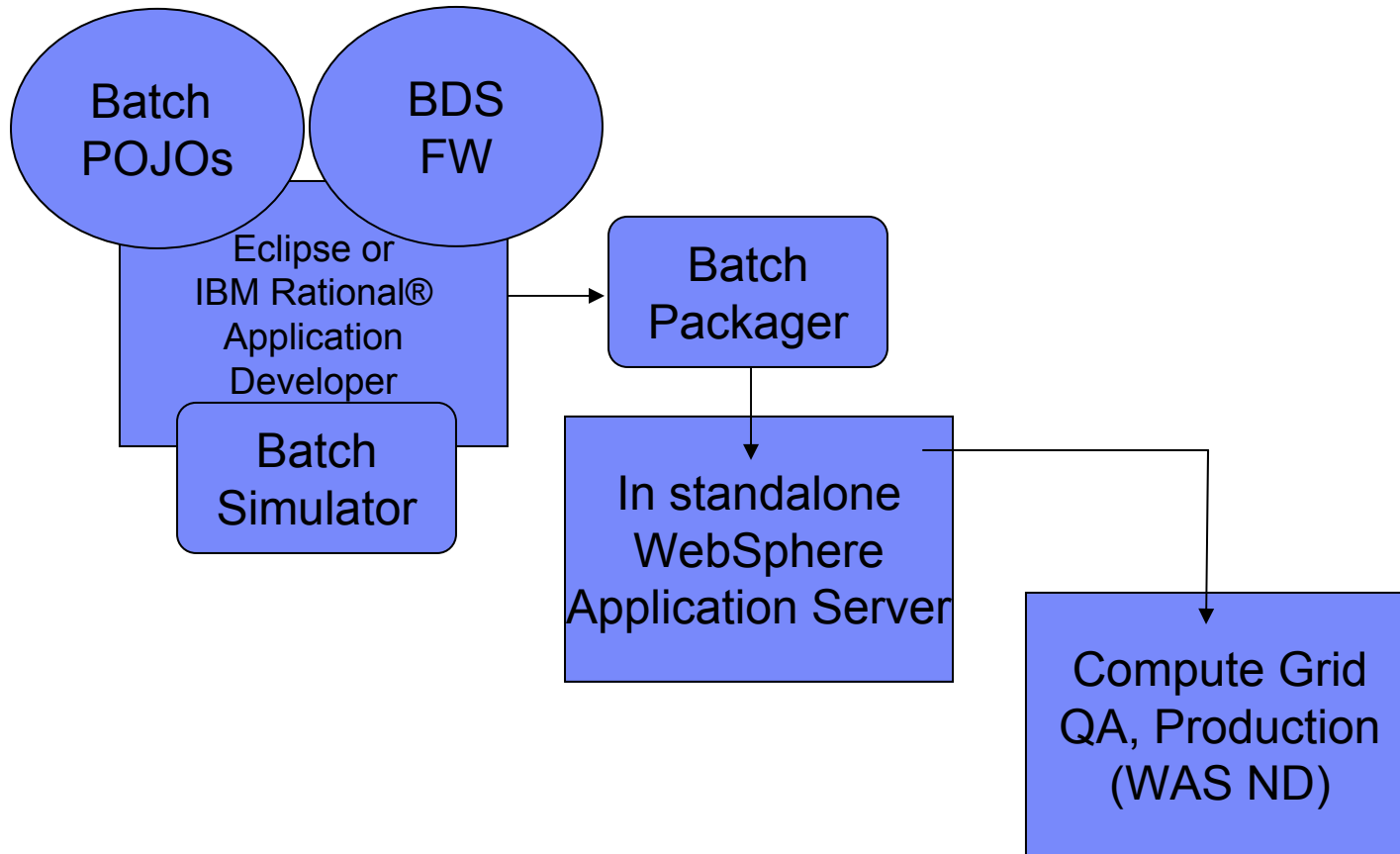


The BDS Framework

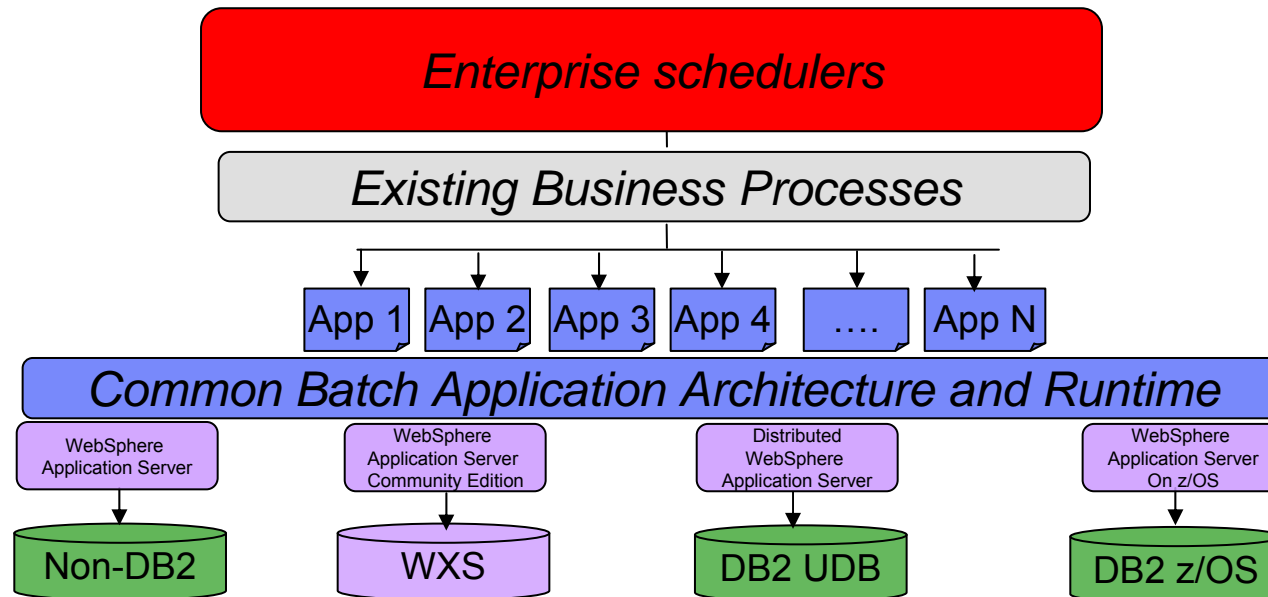
Batch Container



Compute Grid Development Lifecycle




The Batch Vision




- **Portable Batch applications** across platforms
- Location of the data dictates the placement of the batch application
- Centrally managed by your enterprise scheduler
- Integrating with existing: Disaster Recovery, Auditing, Logging, Archiving

References

 WebSphere Extended Deployment Compute Grid ideal for handling mission-critical batch workloads

http://www.ibm.com/developerworks/websphere/techjournal/0804_antani/0804_antani.html

- Enterprise Java Batch with Compute Grid WebCast
<http://www-306.ibm.com/software/os/systemz/telecon/nov15/>
- WebSphere XD Technical Overview Podcast
<http://www.ibm.com/developerworks/podcast/dysmf/dysmf-2007-ep5txt.html?ca=dwpodcastall>
- Java Batch Programming with XD Compute Grid
http://www.ibm.com/developerworks/websphere/techjournal/0801_vignola/0801_vignola.html

 Development Tooling Summary for XD Compute Grid

<http://www.ibm.com/developerworks/forums/thread.jspa?threadID=190624>

- Compute Grid Discussion forum
<http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1240>
- Compute Grid Trial Download
http://www.ibm.com/developerworks/downloads/ws/wscg/learn.html?S_TACT=105AGX10&S_CMP=ART
- Compute Grid Wiki (product documentation)
http://www.ibm.com/developerworks/wikis/display/xdcompute/grid/Home?S_TACT=105AGX10&S_CMP=ART

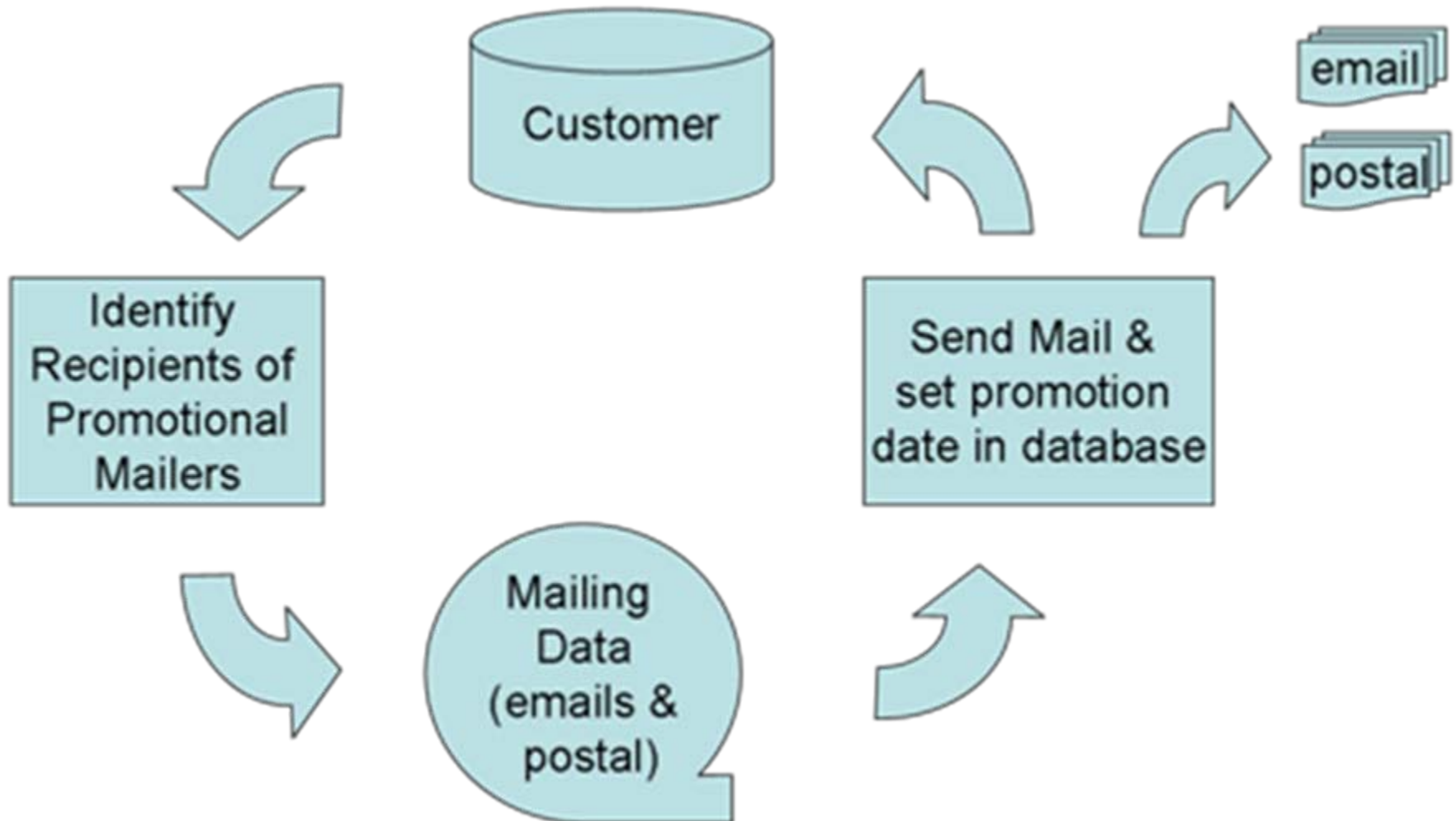


Thank You

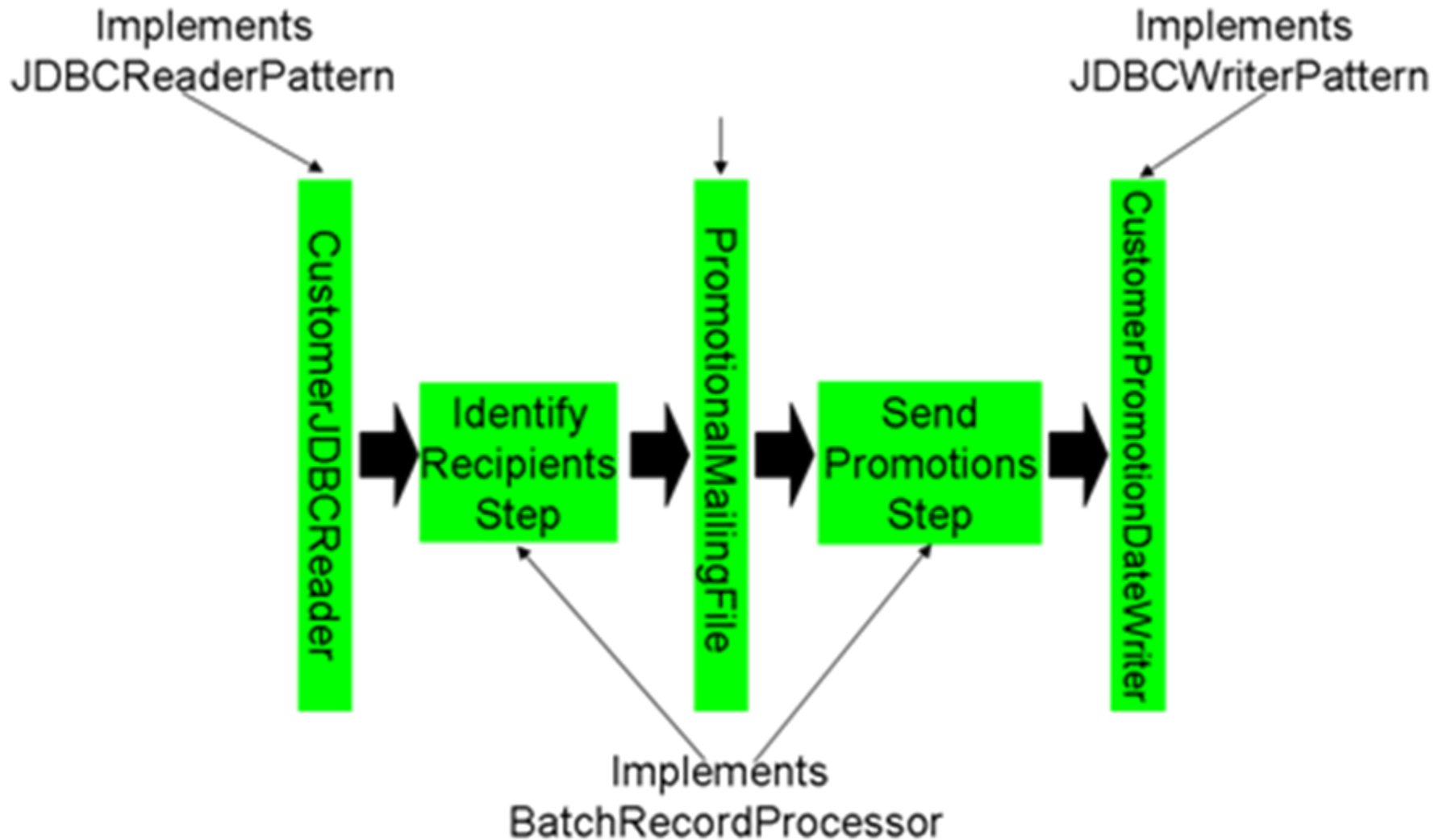
We appreciate your feedback.
Please fill out the survey form in order
to improve this educational event.



The example in the exercise – a promotional mailing



Implemented on top of the BDS Framework





IBM Software Group

Introduction to using the Compute Grid Parallel Job Management

An IBM Proof of Technology



Agenda

- Background
- Starting point – a regular CG Java batch job.
- Top level jobs and sub jobs.
- Parameterization
- The Mailer Sample Application
- Discussion of topology
- Advanced Service Provider Interfaces (SPIs)



Starting point – Substitution Properties in the xJCL

For the substitution properties to influence the range of data that the job acts upon they must be used to modify behavior or configuration of one or more of your batch input streams:

```
<bsd>
<logical-name>inputStream</logical-name>
<props>
  <prop name="EnablePerformanceMeasurement" value="false"/>
  <prop name="EnableDetailedPerformanceMeasurement" value="false"/>
  <prop name="debug" value="false"/>
  <prop name="ds_jndi_name" value="jdbc/DS2" />
  <prop name="PATTERN_IMPL_CLASS" value="demo.streams.OrderJDBCReader"/>
  <prop name="ORDERDATE_RANGE_MIN" value="${beginning_date}"/>
  <prop name="ORDERDATE_RANGE_MAX" value="${ending_date}"/>
</props>
</bsd>
```

xJCL

```
public void initialize(Properties props) {
  orderDateLowerBoundStr = props.getProperty("ORDERDATE_RANGE_MIN");
  orderDateUpperBoundStr = props.getProperty("ORDERDATE_RANGE_MAX");
}
```

Initialization of stream

```
public String getRestartQuery(String restartToken) {
  lastOrderID = Long.parseLong(restartToken);
  String query = SELECT_CLAUSE;
  if ( orderDateLowerBoundStr != null ) {
    query += " WHERE ORDERID > "+ restartToken+
      " AND \!\"ORDERDATE\" BETWEEN \""+
        orderDateLowerBoundStr+" AND \""+orderDateUpperBoundStr +"\" ORDER BY ORDERID";
  } else {
    query += " WHERE ORDERID > "+ restartToken+" ORDER BY ORDERID";
  }
  return query;
}
```

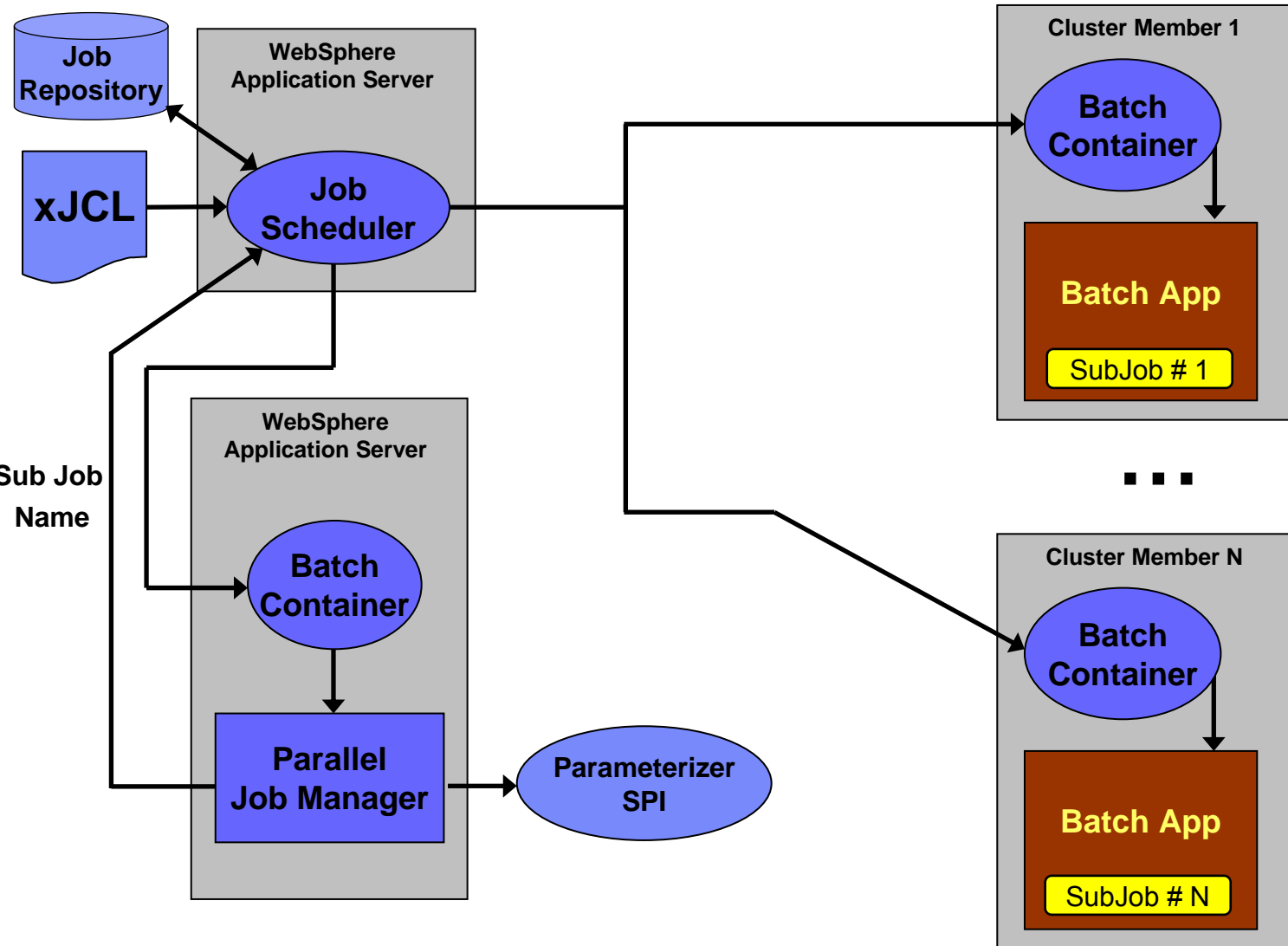
Specifying the query to the database

Background

The Parallel Job Manager is a system provided application.

- ▶ Installed using the wsadmin script parallelJobManager.py.
- ▶ Can be installed to a single server or a cluster.
- ▶ Can be installed into same server or cluster as the Job Scheduler or GEE
- The Parallel Job Manager (PJM) is the target application of a parallel job.
 - ▶ The PJM does not process batch data streams, but instead submits or restarts sub jobs under the control of step properties which identify the sub job in the job repository and the count of sub jobs to process.
 - ▶ A parallel job is submitted using XML Job Control Language (xJCL) for its “top-level” job that specifies these details.
- Sub jobs are:
 - ▶ An instance of a regular batch job that can be **bounded** by substitution properties specified in its xJCL.
 - ▶ Submitted to the job scheduler by the PJM.
 - ▶ Aggregated by the PJM into one logical top level job with regard to
 - Status
 - Result code
 - Operational commands such as submit, cancel, and restart.

Parallel Job Manager



Starting point – a regular CG java batch job.

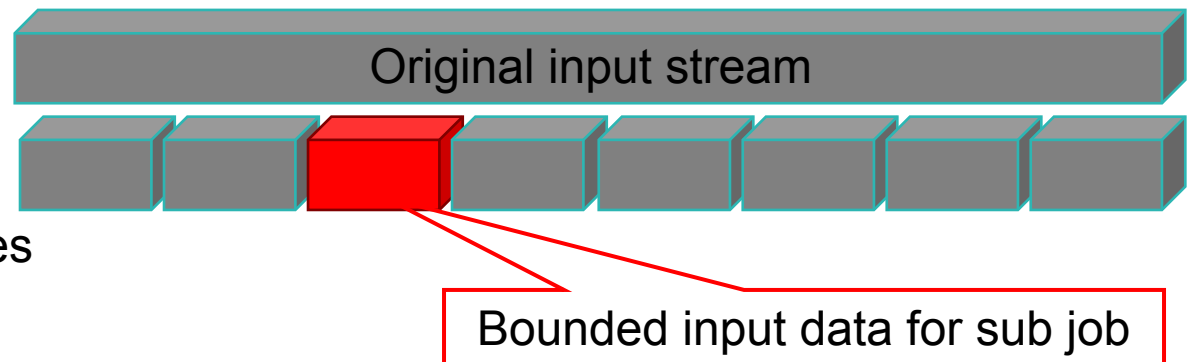
The starting point for parallel job is a regular compute grid job that can be **bounded** by **substitution properties** provided in its xJCL.

Bounded means:

A proper subset of the data stream that the job acts upon can be indentified by specifying one or more constraints that are meaningful to the underlying batch application.

Examples include:

- ▶ Date Ranges
- ▶ Sequences number ranges
- ▶ Postal code
- ▶ Customer or Account ID ranges
- ▶ Regional or organizational qualifiers or code



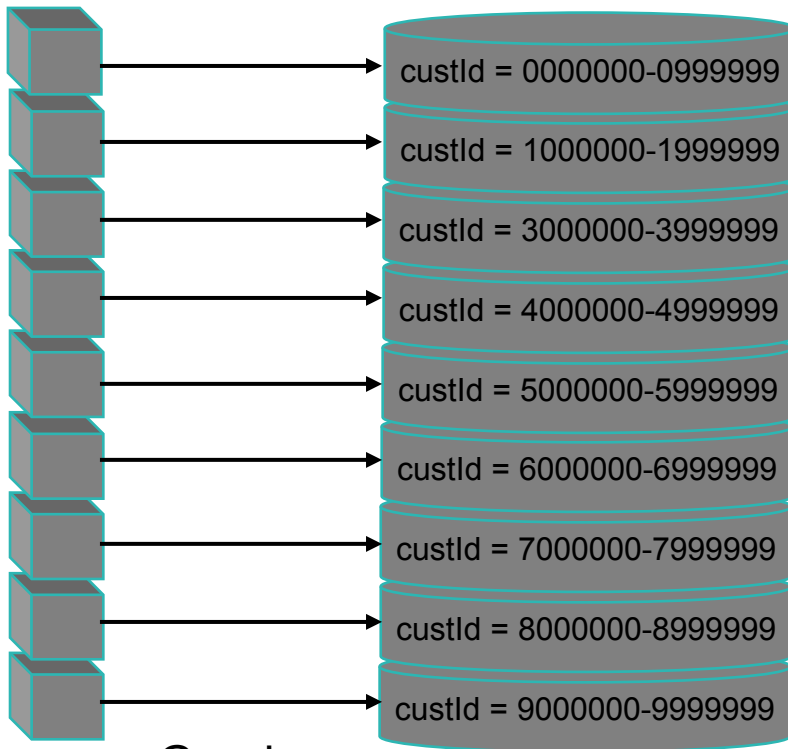
These bounds are used to partition the input streams for multiple sub jobs that will then process those input streams simultaneously.

Bounding constraints and the relational database

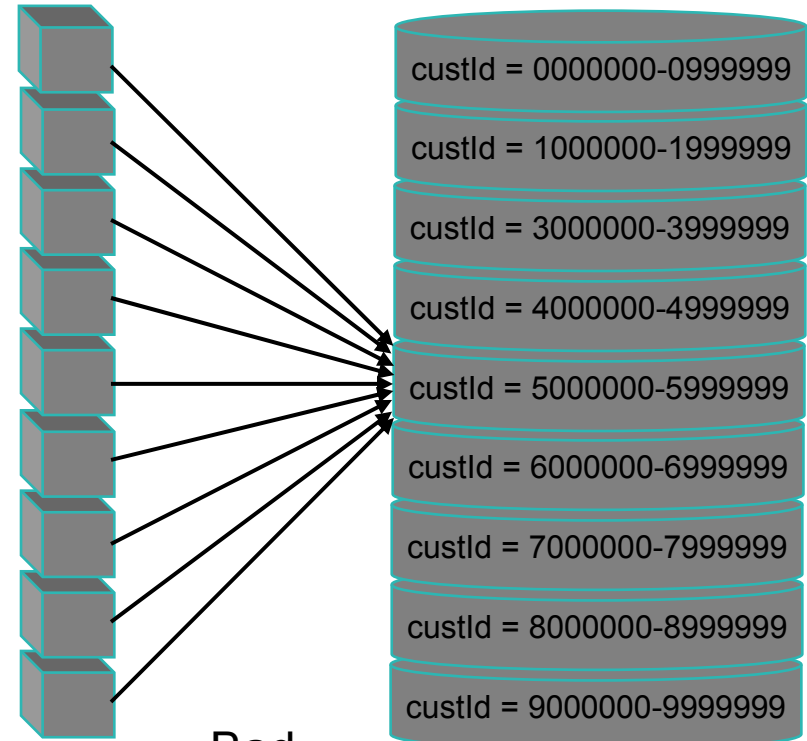
- Good bounding constraints are usually already reflected in the relational database
 - ▶ Database partitions
 - ▶ Primary keys
 - ▶ Indexed columns

Do use parallel jobs to increase throughput by leveraging concurrency.

Don't use parallel jobs to introduce contention thus decreasing throughput.



Good



Bad

Starting point – Substitution Properties

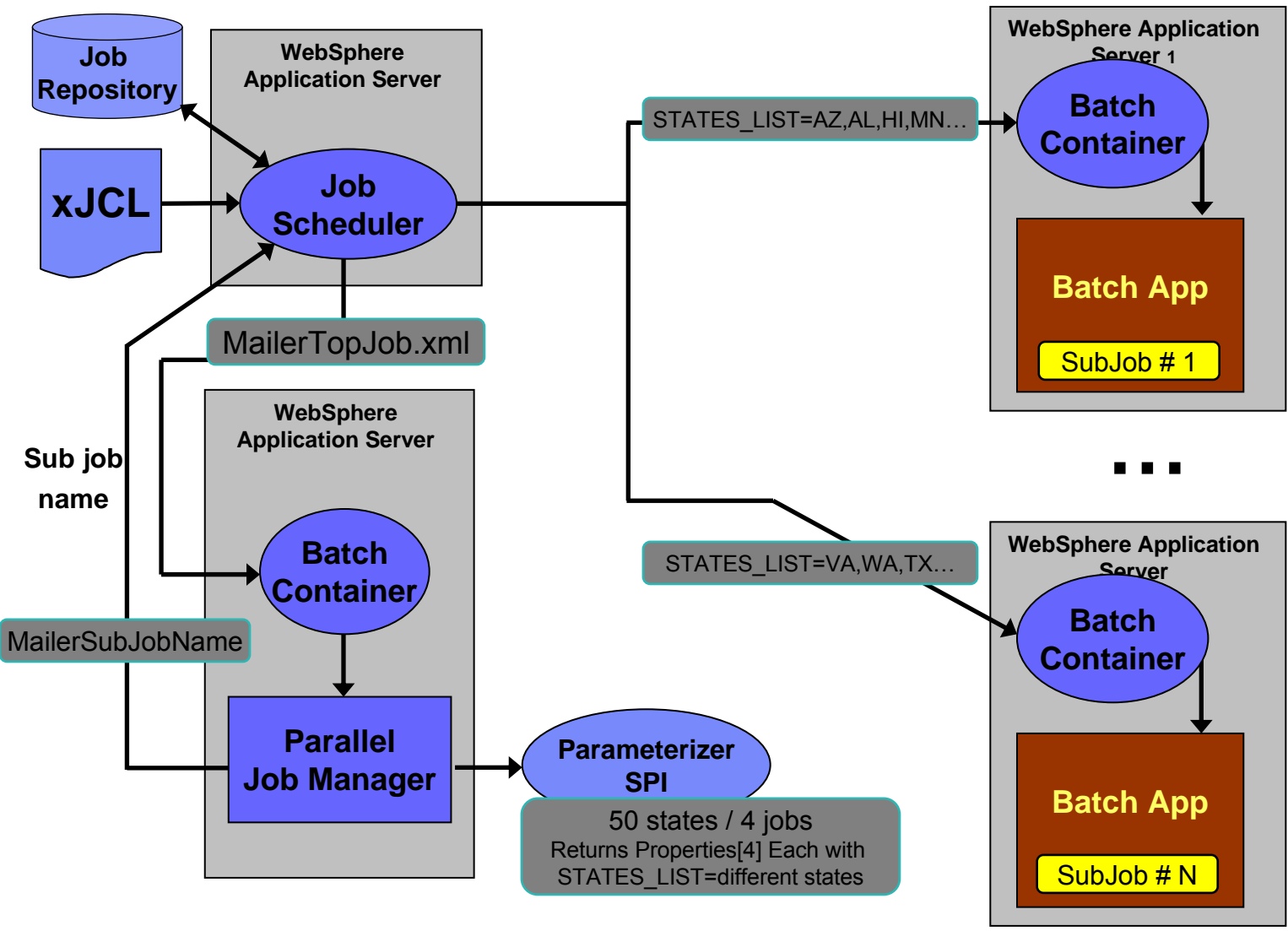
Substitution properties are name-value pairs that can be passed into a job during its submission. Substitutions are performed on the xJCL for the job being submitted.

For a key value pair $K=v$, all occurrences of $\${K}$ in the xJCL will become v

Property	Value
beginning_date	2005-06-01
chkpt_recordcount	1000
ending_date	2005-06-31
file_date_fmt	yyyy-MM-dd
input.file	ORDERS-FIXEDREC.DateSorted.CR
orders.data	/shared/working

So xJCL of the job for the job being submitted above will have occurrences of $\${beginning_date}$ and $\${ending_date}$ replaced by 2005-06-01 and 2005-06-31.

Parallel Job Manager



Mailer Sample TopJob xJCL

```
<substitution-props>
  <prop name="parallel.jobcount" value="3" />
</substitution-props>

<job-step name="Step1">
  <jndi-name>ejb/ParallelJobManager</jndi-name>
  <checkpoint-algorithm-ref name="timebased" />
  <results-ref name="jobsum" />
  <props>
    <prop name="com.ibm.wsspi.batch.parallel.subjob.name" value="MailerSampleJob" />

    <!-- The count of parallel sub jobs to be submitted -->
    <prop name="parallel.jobcount" value="{parallel.jobcount}" />

    <!-- These properties control the runtime properties generated by the Parameterizer SPI. -->
    <prop name="FILENAME" value="/tmp/PJM-TEST-DATA.txt" />

  </props>
</job-step>
```

The StatesList Parameterizer

```

public class StatesListParameterizer extends Parameterizer {
private String states[] = {
    "AL","AK","AZ","AR","CA","CO","CT","DE","FL","GA","HI","ID","IL","IN","IA","KS","KY","LA","ME","MD",
    "MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ","NM","NY","NC","ND","OH","OK","OR","PA","RI","SC",
    "SD","TN","TX","UT","VT","VA","WA","WV","WI","WY" };
public Parameters parameterize(String logicalJobName, String LogicalTransactionID, Properties props) {
    // get job count from properties
    int jobcount = Integer.valueOf(props.getProperty("parallel.jobcount","1"));

    // Establish properties that will be delegated to all without modifications
    Parameters parms = new Parameters();
    parms.setSubJobCount(jobcount);

    //Populate a Properties object for each subjob.
    Properties newprops [] = new Properties[jobcount];
    for ( int i=0; i<jobcount; i++) {
        newprops[i] = new Properties();
        // calculate slice of states array and build state listfor subjob
        int slice_size = states.length / jobcount;
        int start_index = i * slice_size;
        int end_index = (i == jobcount-1) ? states.length : (i+1)*slice_size;
        String stateList = new String();
        for ( int j = start_index; j < end_index; j++ ) {
            if ( j == start_index ) { stateList += states[j];
            } else { stateList += ","+states[j];
            }
        }
        newprops[i].put("STATES_LIST", stateList);
    }
    parms.setSubJobProperties(newprops);
    return parms;
}
}

```

Mailer Sample SubJob xJCL

```
<job-step name="IdentifyRecipientsStep">
  <jndi-name>ejb/IdentifyRecipientsStep</jndi-name>
  <checkpoint-algorithm-ref name="recordbased" />
  <results-ref name="jobsum" />
  <batch-data-streams>
<bds>
  <logical-name>inputStream</logical-name>
  <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.CursorHoldableJDBCReader</impl-class>
  <props>
    <prop name="IMPLCLASS" value="com.ibm.websphere.samples.CustomerJDBCReader" />
    <prop name="ds_jndi_name" value="jdbc/mailingnonxa" />
    <prop name="STATES_LIST" value="{STATES_LIST}" />
    <prop name="debug" value="false" />
  </props>
</bds>
<bds>
  <logical-name>outputStream</logical-name>
  <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileWriter</impl-class>
  <props>
    <prop name="IMPLCLASS" value="com.ibm.websphere.samples.PromotionalMailingFile" />
    <prop name="FILENAME" value="{EXCHANGED_FILENAME}" />
    <prop name="debug" value="false" />
  </props>
</bds>
</batch-data-streams>
<props>
  <prop name="BATCHRECORDPROCESSOR" value="com.ibm.websphere.samples.IdentifyRecipientsStep" />
  <prop name="AD_CAMPAIGN_CODE" value="A1025" />
  <prop name="PROMOTIONAL_MAILER_THRESHOLD" value="5000" />
  <prop name="DELAY_EXECUTION_TIME" value="0" />
  <prop name="debug" value="false" />
</props>
```

Specifying sub job props in top-level job xJCL

```
<job-step name="Step1">
  <jndi-name>ejb/ParallelJobManager</jndi-name>
  <checkpoint-algorithm-ref name="timebased" />
  <results-ref name="jobsum" />
  <props>
    <!-- Properties to be consumed by the Parallel Job Manager -->
    <prop name="DEBUG" value="{DEBUG}" />
    <prop name="com.ibm.wsspi.batch.parallel.subjob.name" value="ParaMailSubJob" />
    <!-- The count of parallel sub jobs to be submitted -->
    <prop name="parallel.jobcount" value="4" />
    <!-- Properties intended for all subjobs -->
    <prop name="parallel.subjobprop.All.EXCHANGED_FILENAME" value="{EXCHANGED_FILENAME}" />
    <prop name="parallel.subjobprop.*.DEBUG" value="{SUB_JOB_DEBUG}" />
    <!-- Properties intended for specific subjobs -->
    <prop name="parallel.subjobprop.1.STATES_LIST" value="AL,AK,AZ,AR,CA,CO,CT,DE,FL,GA,HI,ID,IL" />
    <prop name="parallel.subjobprop.2.STATES_LIST" value="IN,IA,KS,KY,LA,ME,MD,MA,MI,MN,MS,MO" />
    <prop name="parallel.subjobprop.3.STATES_LIST" value="MT,NE,NV,NH,NJ,NM,NY,NC,ND,OH,OK,OR" />
    <prop name="parallel.subjobprop.4.STATES_LIST" value="PA,RI,SC,SD,TN,TX,UT,VT,VA,WA,WV,WI,WY" />
  </props>
</job-step>
```

Steps to transforming a regular job into a parallel job

- Identify a regular CG Java batch job that is boundable as described earlier.
 - ▶ Modify the underlying application if necessary to base the bounding of the input stream on properties that can be passed into the job as substitution properties.
- Copy the regular jobs xJCL and add some additional “boiler plate” substitution properties required by the PJM (more on this in the next slide).
 - ▶ They won't be used by your application but rather by the PJM.
- Save the sub job xJCL to the job repository.
- Decide on whether to use the GenericParameterizer and specify the various sub job properties in the top level job xJCL or write your own parameterizer.
- Customize a top level job xJCL that specifies the Parallel Job Manager as its target batch application.
 - ▶ It must also specify subjob.name with the name that you saved the sub job under.c

'Boilerplate' to add to sub job xJCL

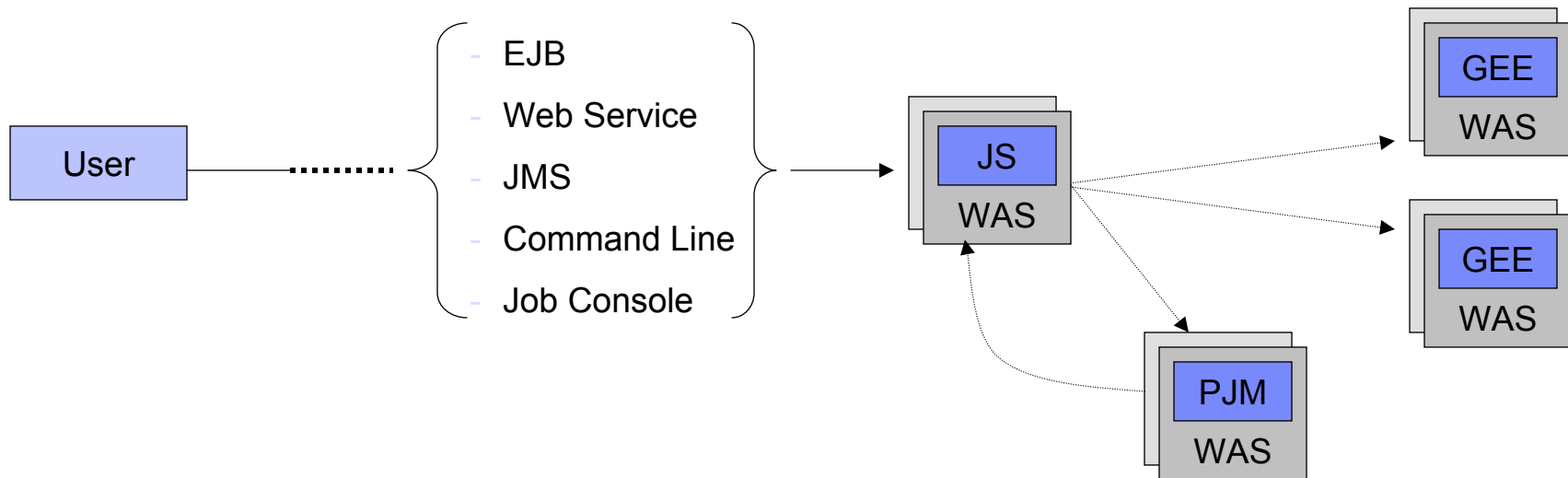
Instead of its own job name the sub job will now be assigned a job name by the PJM

```
<job name="{parallel.jobname}" default-application-name="MailerSample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-..
```

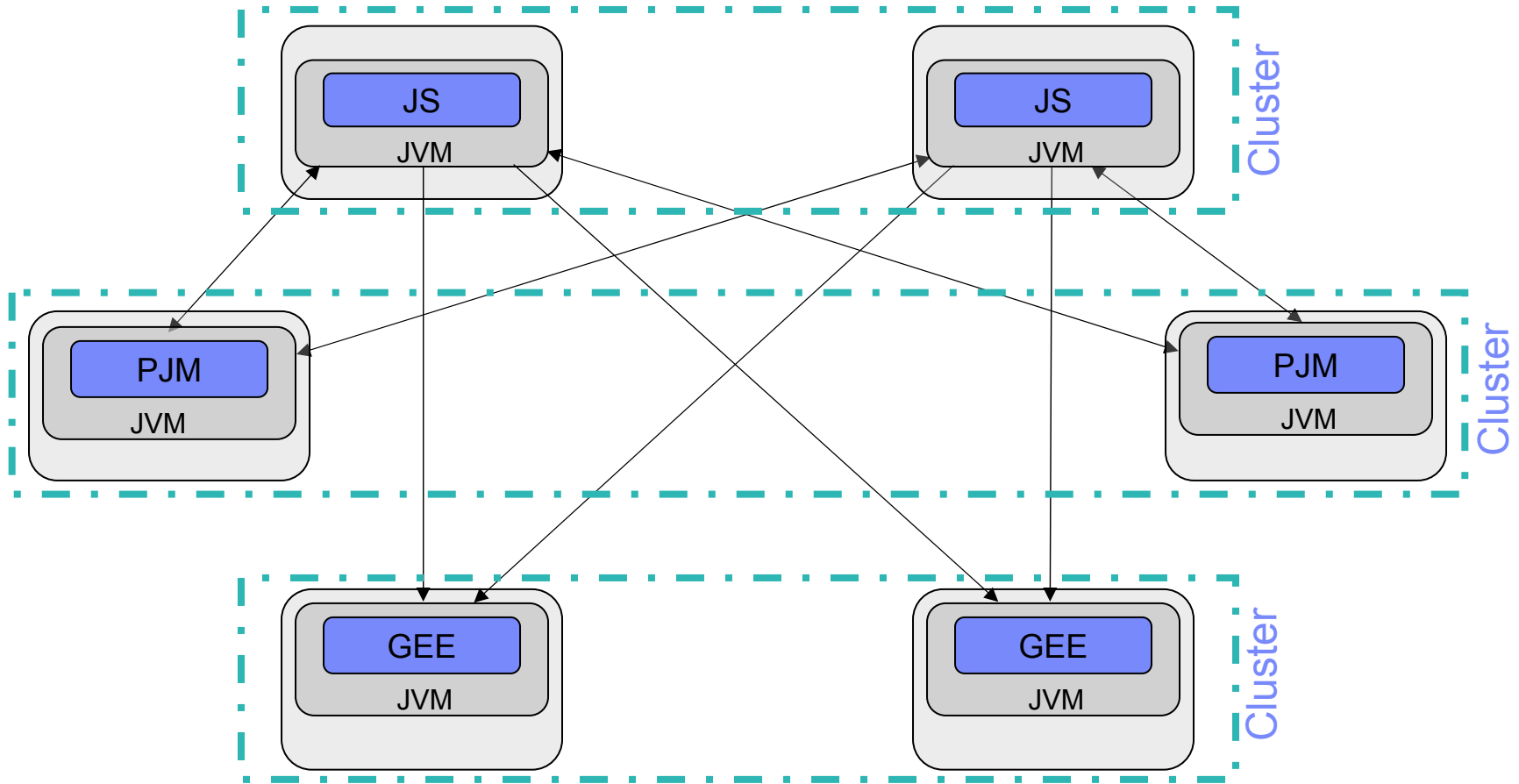
The following block should be added to each steps props section.

```
<!-- the following properties are modified at job submission time by the ParallelJobManager. -->  
<!-- SPI com.ibm.websphere.samples.parallelPostingSampleParameterizer is used to set the properties -->  
<!-- which are then passed when the sample is submitted from the job repository -->  
<!-- these three properties are REQUIRED by ParallelJobManager conventions -->  
<prop name="com.ibm.wsspi.batch.parallel.jobname" value="{parallel.jobname}" />  
<prop name="com.ibm.wsspi.batch.parallel.logicalTXID" value="{logicalTXID}" />  
<prop name="com.ibm.wsspi.batch.parallel.jobmanager" value="{parallel.jobmanager}" />
```

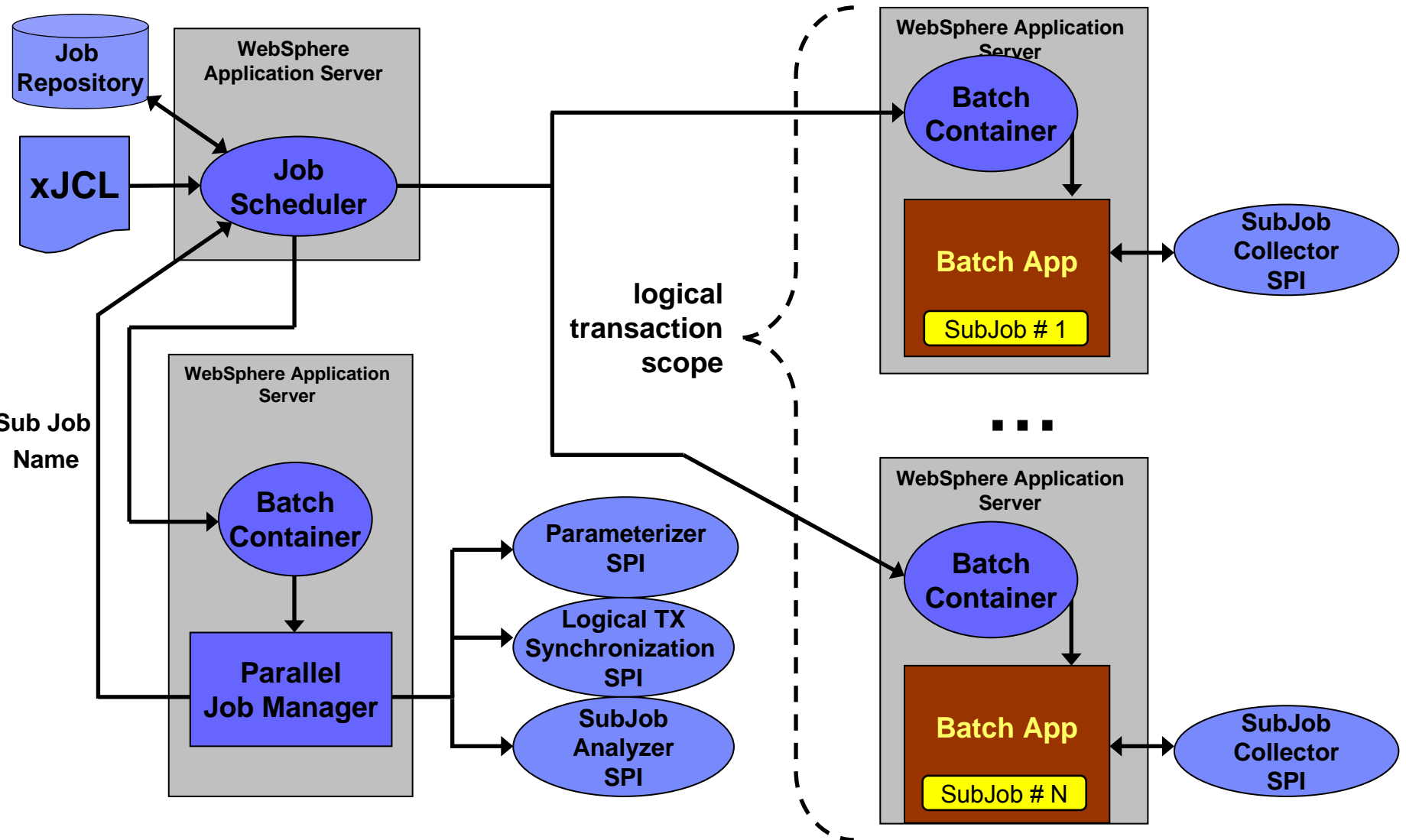
Compute Grid Components



Example Production Topology- Highly Available/Scalable Compute Grid



Parallel Job Manager



Logical Transaction

- A logical transaction provides a logical unit of work scope that spans all subjobs belonging to a parallel job.
- The Parallel Job Manager starts a logical transaction before the subjobs are submitted and completes the logical transaction after all subjobs have completed.
- logical transactions provide a containment scope only - resource enlistment and coordination with resource manager JTA transactions is not provided.

Not JCA Transactions

- Rollback is initiated in the following ways:
 - ▶ When any SPI in the `com.ibm.wsspi.batch.parallel` family throws declared exception `RollbackLogicalTXException`.
 - ▶ When any subjob completes in the `Restartable` or `Failed` state.
- When rollback is triggered, the following actions are taken:
 - ▶ In-flight subjobs are cancelled. This includes subjobs still executing on endpoints, as well as subjobs that may be awaiting dispatch.
 - ▶ The `rollBack` method on this SPI is invoked. It is the responsibility of the `rollBack` method to undo any work done by subjobs that have successfully completed one or more checkpoints. Note this includes subjobs that have completed successfully - i.e. are in the "ended" state.



The logicalTX.Synchronization SPI

Method Summary

void begin (java.lang.String logicalTXID)

Indicates the specified logical transaction has begun.

void beforeCompletion (java.lang.String logicalTXID)

Indicates all subjobs have ended and that the specified logical transaction is about to commit.

void afterCompletion (java.lang.String logicalTXID, Synchronization.TXStatus status)

Indicates the specified logical transaction has completed with the indicated status.

RestartInstructions rollBack (java.lang.String logicalTXID)

Invoked to undo changes made by subjobs within the unit of work scope of the specified logical transaction.

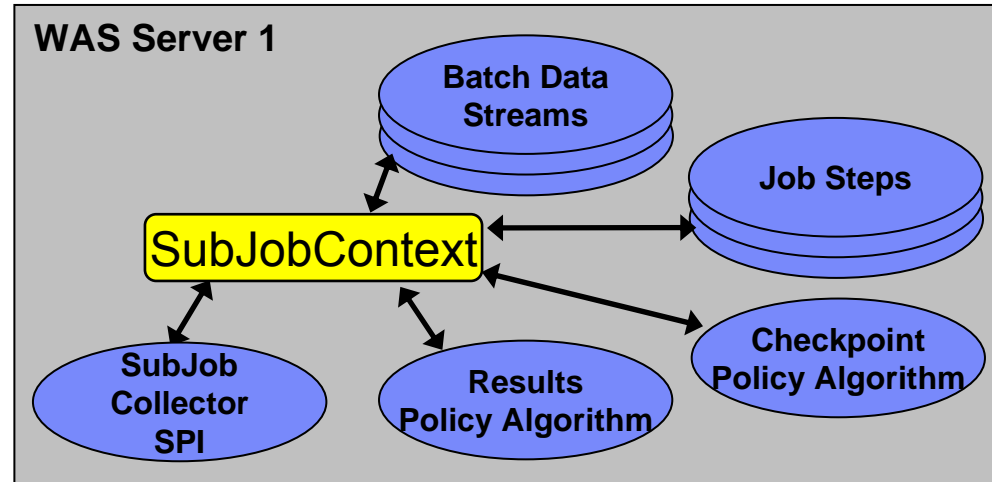
RestartInstructions contains:

- A RestartDirective that can be RESTARTABLE,ALL or SPECIFIED.
- An Array of restartable subjob IDs for the jobs that need to be run if directive is SPECIFIED.

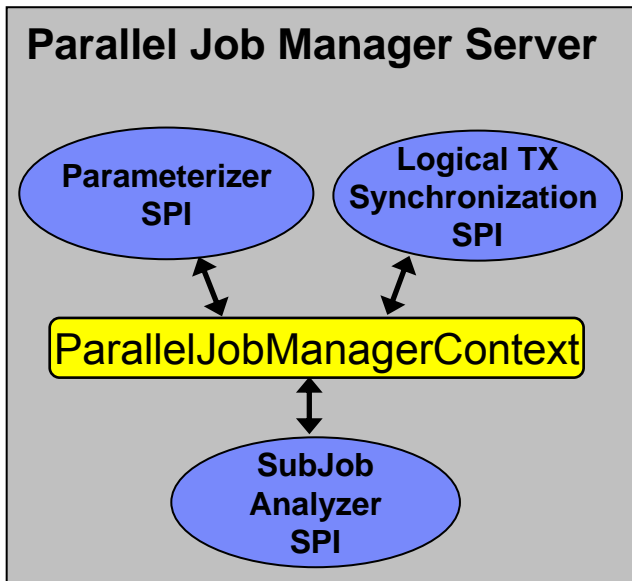
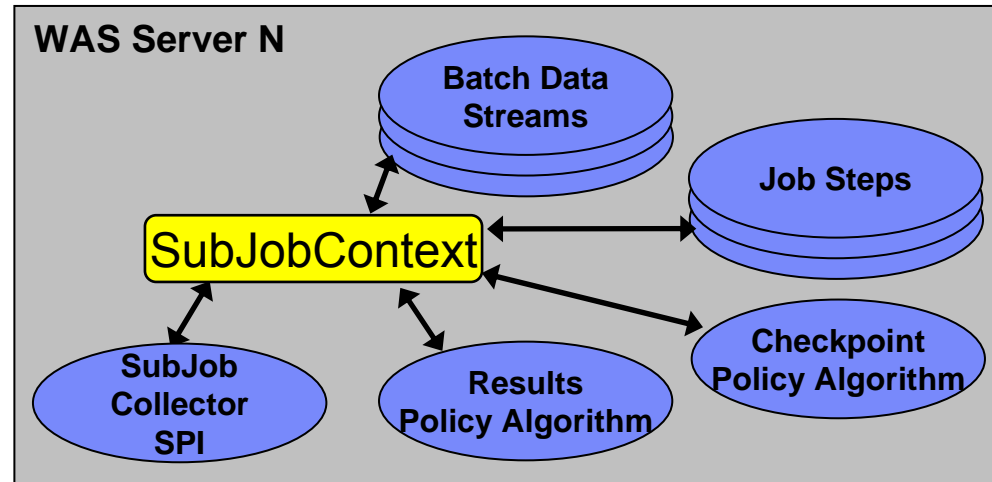


SubJob Collector, Analyzer and Contexts

- Together they provide for the gathering, propagation and aggregation of application specific status and completion data.
- This information can be useful when the `SubJobAnalyzer` calculates the overall job return code and when `Synchronization` determines `RestartInstructions`.



...



Installing the Parallel Job Manager

The `jython` script `parallelJobManager.py` is provided in `${WAS_INSTALL_ROOT}/bin`. It can be used to install to a server or a cluster (both static or dynamic)

Install to a cluster:

```
wsadmin parallelJobManager.py -install -cluster <clusterName>
```

Install to a server (must specify server name and node name):

```
wsadmin parallelJobManager.py -install -node <nodeName> -server <serverName>
```

Uninstall from a cluster:

```
wsadmin parallelJobManager.py -remove -cluster <clusterName>
```

Uninstall from a server (must specify server name and node name):

```
wsadmin parallelJobManager.py -remove -node <nodeName> -server <serverName>
```

Installing the Parallel Job Manager – Database Options

The Parallel Job Manager requires a relational database.

The `parallelJobManager.py` script provides options that can be added to `-install` to specify the schema name, backend database type and JDBC JNDI name.

The options are:

`-schema <schemaName>`

Default: PJSHEMA

`-backend <dbType>`

Default: DERBY_V100_1

`-jdbcname < jdbcname >`

Default: jdbc/parallelJobManager

Parallel Job Manager Updates

- Built-in parameterizer implementation
- Sub-job pacing
- Top-level job context persistence
- Shared library SPI deployment

Built-in parameterizer parallelization

- `com.ibm.ws.batch.parallel.BuiltInParameterizer` SPI implementation
- Number of sub-jobs specified in xJCL of top level job

com.ibm.wsspi.batch.parallel.jobs=N

- Sub-job substitution properties specified in the top level job

For properties assigned to specific sub-job:

com.ibm.wsspi.batch.parallel.prop.<property_name>.<subjob>=<value>

For properties assigned to all sub-jobs:

com.ibm.wsspi.batch.parallel.prop.<property_name>=<value>

Sub-job pacing

- In the top level job xJCL you can define a property that will specify the maximum number of concurrent sub jobs.

- Maximum number of concurrently submitted subjobs

com.ibm.ws.batch.parallel.MAXIMUM_CONCURRENT_SUBJOBS=N

- Maximum number of submission threads

com.ibm.ws.batch.parallel.MAXIMUM_SUBJOBS_SUBMISSION_THREADS=N

Top-level job context persistence

- A property in the top level job enables persistence of context.
- *com.ibm.ws.batch.parallel.PERSIST_TOP_LEVEL_JOB_CONTEXT=true*
- ParallelJobManager Context is persisted to database.
- New table added to Parallel Job Manager DDL.



Thank You





IBM Software Group

Java Batch Modernization using WebSphere Compute Grid v6.1.1

Enterprise Scheduler Integration

An IBM Proof of Technology



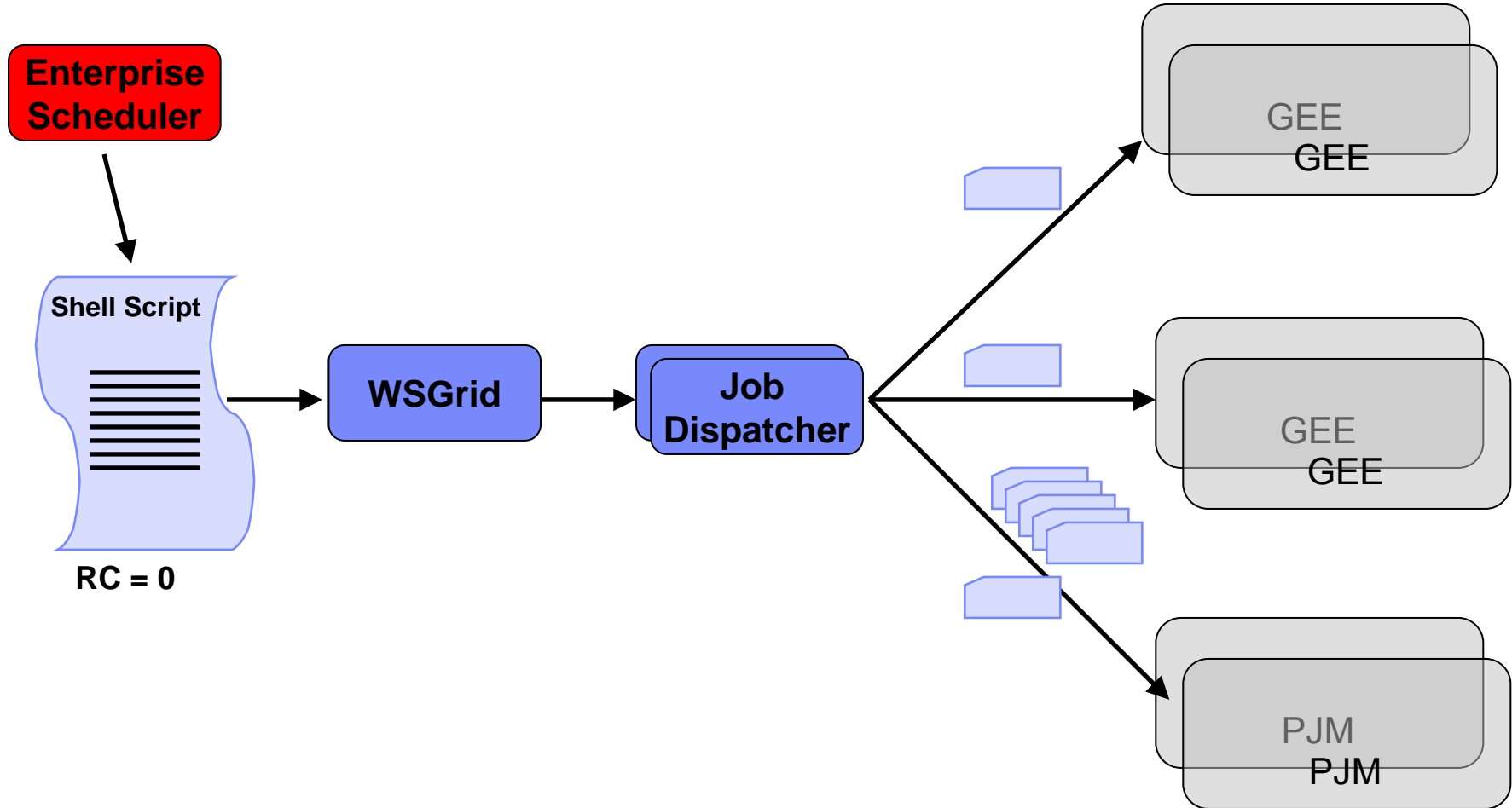
Established Enterprise Job Scheduling Environments

Customers typically have large and established enterprise scheduling environments:

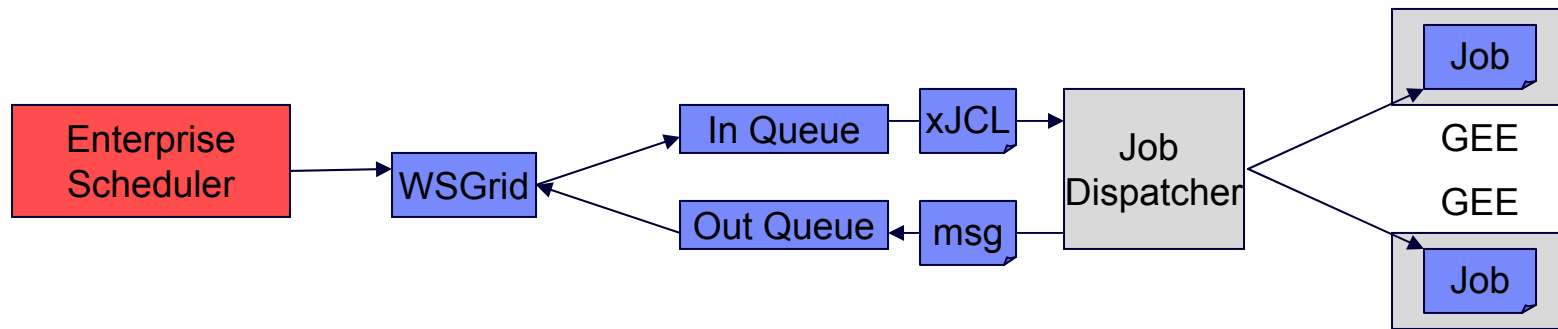
- Span various organizations and technologies.
 - Manage dependencies between jobs, notification of failure, retention of logs.
 - Frequently include both computing activities and business events, human actions.
 - Have an entire staff responsible for maintaining these jobs.
-
- WebSphere Compute Grid provides for integrating with these established scheduling environments non-disruptively and incrementally.



Simple shell script invocation facade



Enterprise Schedulers and WebSphere Compute Grid

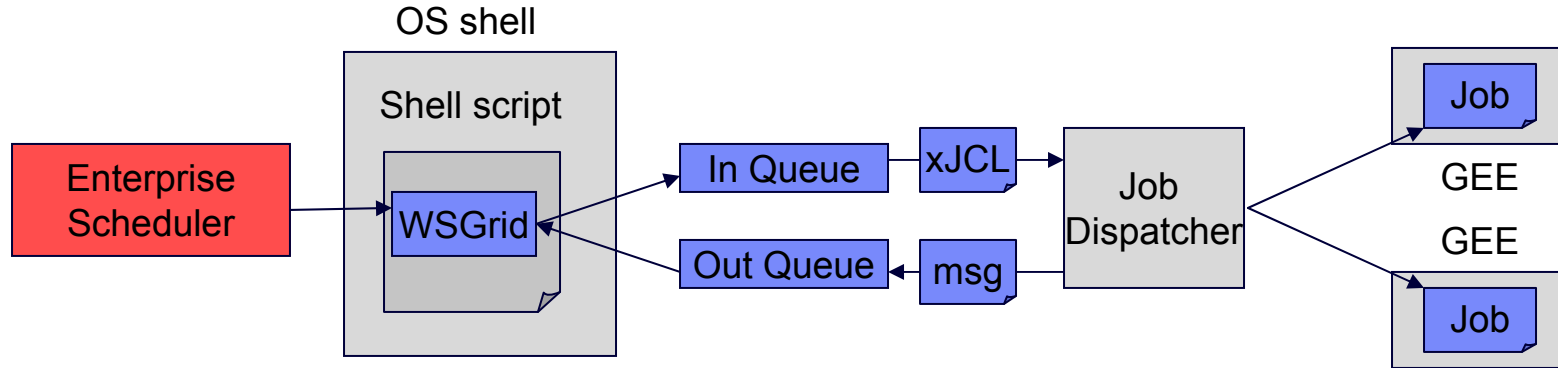


-WebSphere Compute Grid provides a utility named WSGrid that allows customers to submit jobs through the job dispatcher using from a wide variety of enterprise schedulers including IBM Tivoli® Workload Schedulers.

-To the enterprise scheduler, WSGrid behaves like an ordinary batch script that once submitted, retains control, directs log information to standard output and returns a completion code.

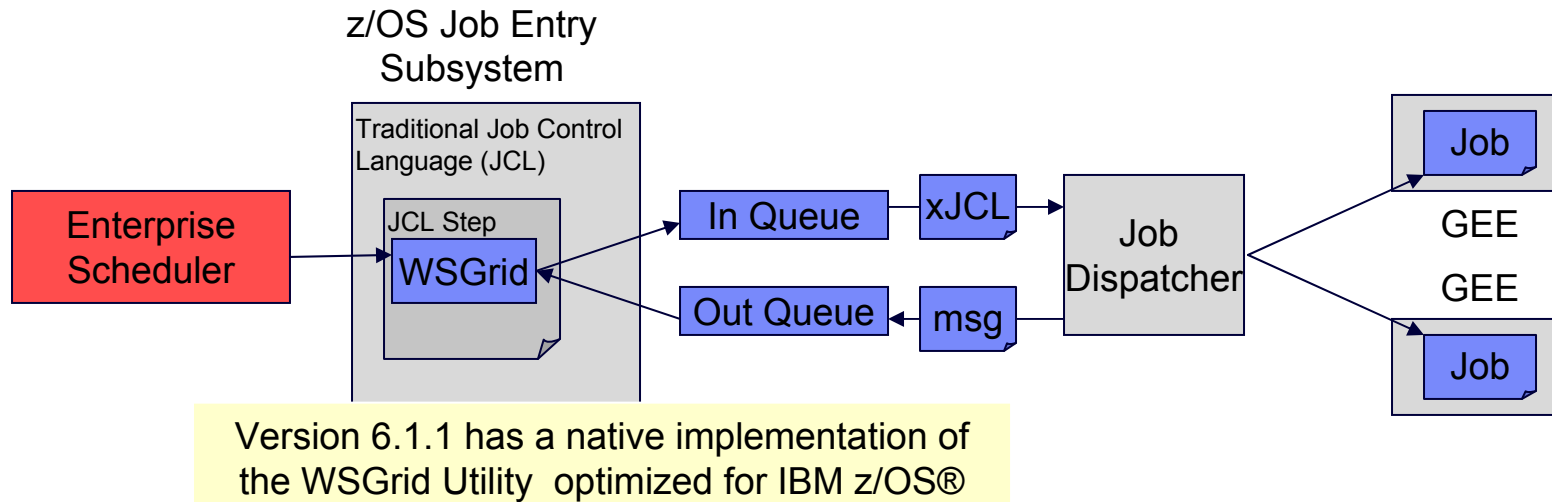
-While presenting this simple abstraction to the enterprise scheduler, WSGrid submits the job through the dispatcher and gathers and renders logs using asynchronous messaging with the job dispatcher. It returns the completion code of the WebSphere Compute Grid job as its own result code only after the job completes.

Enterprise Schedulers and WebSphere Compute Grid



- On distributed platforms the WSGrid utility is typically invoked from within a shell script.
- Logs are returned in standard output stream
- Result code is returned in $\$@$ or ERRORLEVEL variables depending on operating system.

Enterprise Schedulers and WebSphere Compute Grid



-On IBM z/OS® the WSGrid utility is typically invoked from within a job step of a traditional JCL job running in the z/OS Job Entry Subsystem(JES)

-Logs are returned SYSOUT stream of the step

-Result code is returned as the return code of the step.

Invocation Syntax

WSGrid.sh control.properties job.properties restart.properties

control.properties

Properties used to connect to WebSphere Compute Grid job dispatcher. This file will likely be the same for many different jobs

job.properties

Properties specifying the job being submitted and any substitution properties specified in the jobs xJCL.

restart.properties.

File name for a not-yet-created file, unique to a given submission of the job. If the job fails in a restartable state this file will contain the following property:

`restart-job=jobID`

Job restart can then be accomplished using the following command:

WSGrid.sh control.properties restart.properties

Customers use WSGrid with a wide variety technologies

- Traditional Enterprise Schedulers such as IBM Tivoli ® Workload Scheduler.
- Home grown shell script environments.
- Remote invocation using Secure Shell (ssh).
- Any technology that can launch a platform appropriate shell script or JCL job and consume the result codes can be used as an integration point.



Thank You

We appreciate your feedback.
Please fill out the survey form in order
to improve this educational event.



References

- **IBM WebSphere Application Server V7 Feature Pack for Modern Batch Beta**
<http://www.ibm.com/software/webservers/appserv/was/featurepacks/modernbatch/>

WebSphere Batch processing

- **Batch Processing with WebSphere Compute Grid: Delivering Business Value to the Enterprise**
<http://www.redbooks.ibm.com/Redbooks.nsf/RedpieceAbstracts/redp4566.html?Open>
- **Java-enabled batch for cost-optimized, efficient business processing**
<http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&appname=SWG WS AT USEN&htmlfid=WSW14113USEN&attachme nt=WSW14113USEN.PDF>
- **WebSphere Extended Deployment Compute Grid ideal for handling mission-critical batch workloads**
http://www.ibm.com/developerworks/websphere/techjournal/0804_antani/0804_antani.html
- **CCR2 article on SwissRe and Compute Grid**
<http://www-01.ibm.com/software/tivoli/features/ccr2/ccr2-2008-12/swissre-websphere-compute-grid-zos.html>
- **Java Batch Programming with XD Compute Grid**
http://www.ibm.com/developerworks/websphere/techjournal/0801_vignola/0801_vignola.html
- **WebSphere Compute Grid Frequently Asked Questions**
<http://www-128.ibm.com/developerworks/forums/thread.jspa?threadID=228441&tstart=0>
- **Development Tooling Summary for XD Compute Grid**
<http://www.ibm.com/developerworks/forums/thread.jspa?threadID=190624>