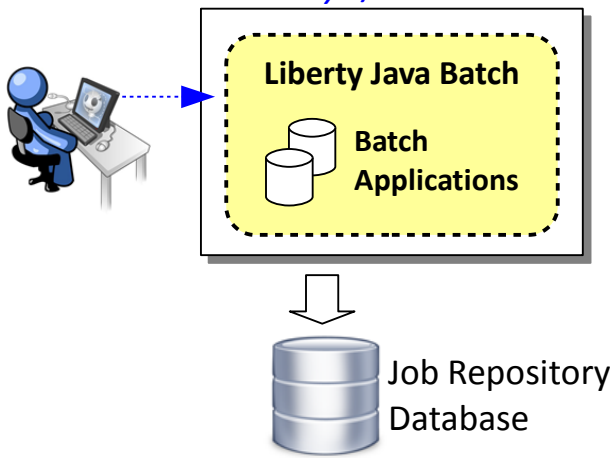
WebSphere Application Server

Unit 5

Multi-JVM Configuration

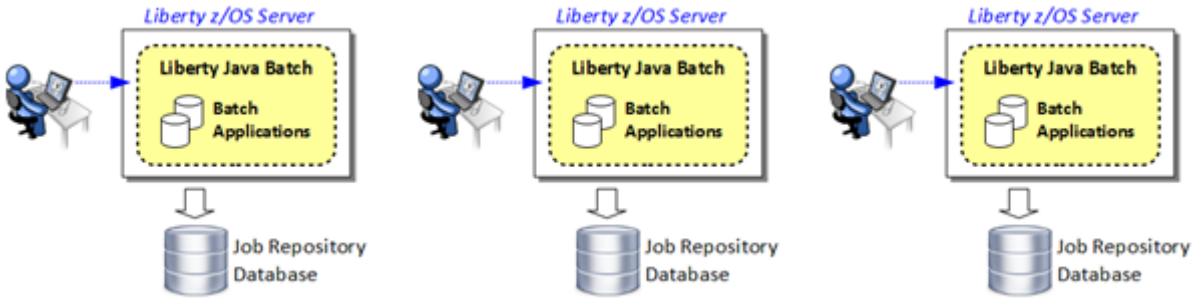
Up to This Point in the Workshop We Have Focused on a Single Server Configuration

"Lone Wolf"

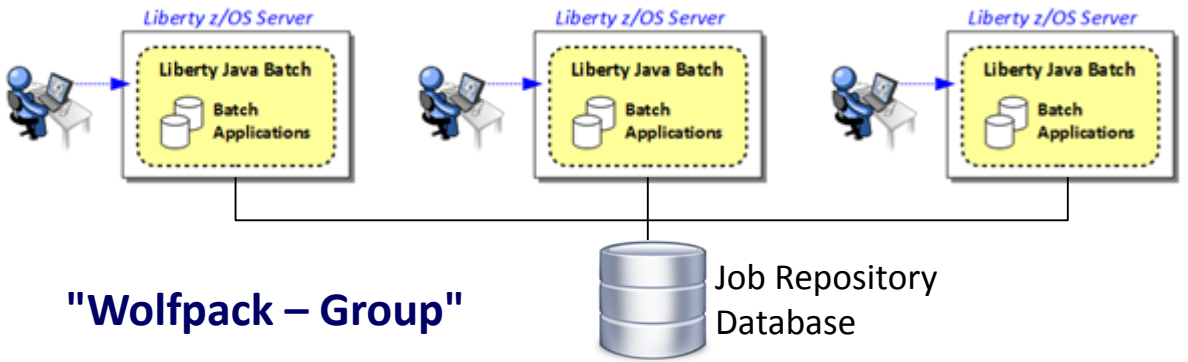


We started here because it's relatively simple to understand, setup, and operate.

Some variations ⇨ ⇨ ⇨



"Wolfpack – Independent"



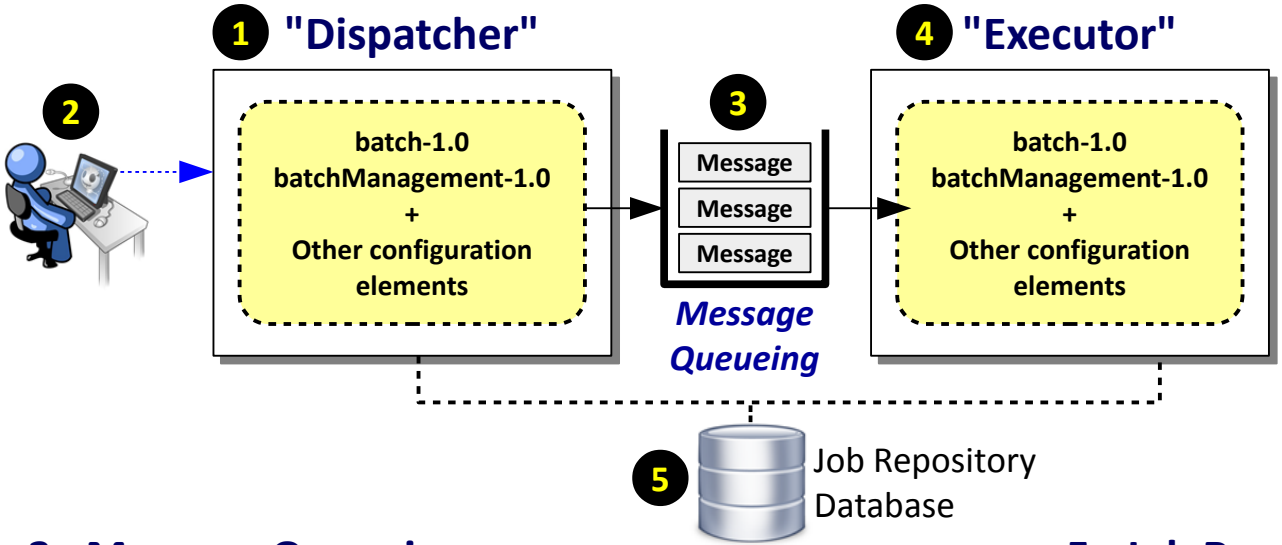
"Wolfpack – Group"

You could do this, but it could get confusing as one server would have visibility to jobs the other servers have run. So you should avoid this.

Techdoc http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102626

Multi-JVM model ...

The Multi-JVM Model



1. Dispatcher Server(s)

This server is configured with the Java Batch features as well as other XML elements that define it as a Dispatcher and provide information about the JMS queue to use.

2. Job Interfaces

The same interfaces we've seen earlier apply: batchManager, batchManagerZos, the REST interface, the AdminCenter.

3. Message Queueing

A message queue sits between the Dispatcher and the Executor. The Dispatcher places a job submission message on the queue; the Executor picks it up and executes.

4. Executor Server(s)

This server is configured with the Java Batch features as well as other XML element that define it as an Executor server.

Configurable JMS activation specs are defined to indicate which messages to pick up from the queue.

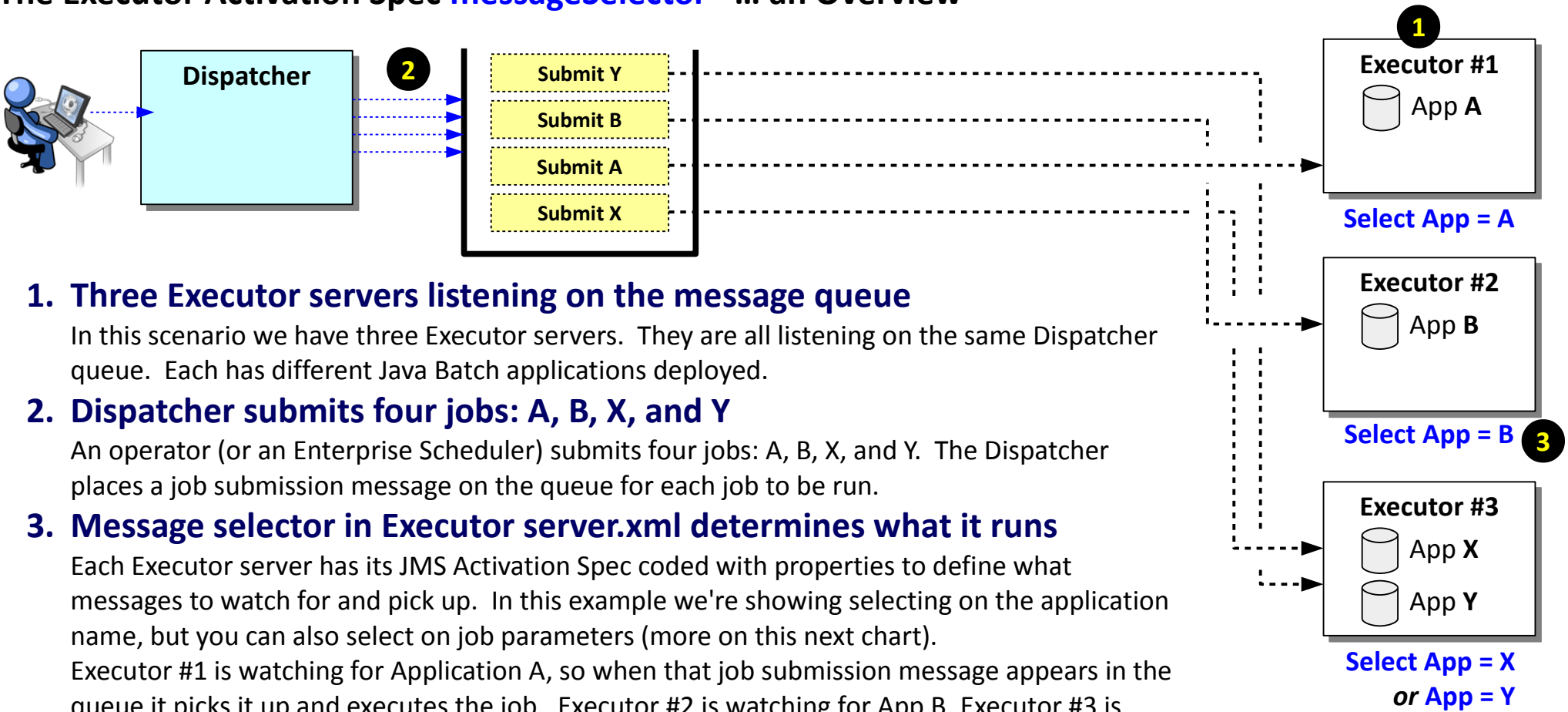
5. Job Repository Database

To use the Multi-JVM design you must have employ a relational database capable of sharing data between multiple servers.

This part is key, so let's take a quick look at that before getting into other details ...

messageSelector overview ...

The Executor Activation Spec messageSelector= … an Overview



1. Three Executor servers listening on the message queue

In this scenario we have three Executor servers. They are all listening on the same Dispatcher queue. Each has different Java Batch applications deployed.

2. Dispatcher submits four jobs: A, B, X, and Y

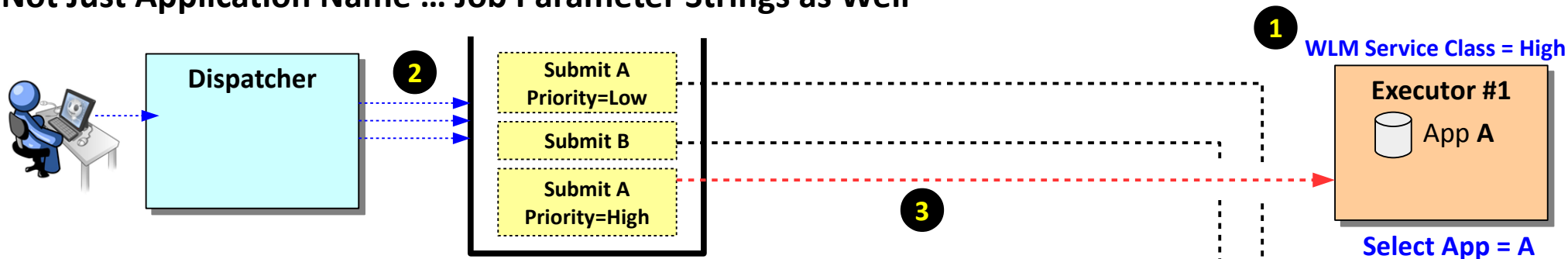
An operator (or an Enterprise Scheduler) submits four jobs: A, B, X, and Y. The Dispatcher places a job submission message on the queue for each job to be run.

3. Message selector in Executor server.xml determines what it runs

Each Executor server has its JMS Activation Spec coded with properties to define what messages to watch for and pick up. In this example we're showing selecting on the application name, but you can also select on job parameters (more on this next chart).

Executor #1 is watching for Application A, so when that job submission message appears in the queue it picks it up and executes the job. Executor #2 is watching for App B. Executor #3 is watching for X or Y, and will pick up job submission messages for either.

Not Just Application Name … Job Parameter Strings as Well



1. **Two Executor servers listening on the message queue**
   We simplified things by reducing number of servers to two. Executor #1 has just App A deployed, Executor #2 has App A and App B. Executor #1 has a WLM Service Class assigned to it that gives jobs run here a relatively higher priority than jobs run elsewhere. This is where "high priority" jobs are run.
2. **Dispatcher submits four jobs: A, B, X, and Y**
   An operator (or an Enterprise Scheduler) submits three jobs: B, A with `Priority=Low` and another instance of A with `Priority=High`.
3. **Message selector behavior in this scenario**
   Executor #1 is looking for the –JobParameter string of Priority=High. It will only pick up messages that meet that criteria. So it will ignore the Priority=Low.
   Executor #2 is looking for App A with a 'Priority' value of anything other than 'High,' as well as picking up any submission messages for App B.

The Multi-JVM Model Provides You With ...

- **Run a batch job on one of several eligible servers**

  This could be part of a "high availability" design where servers capable of running the batch job are on several LPARs

  This could be part of a "load balancing" design where, say, 100 jobs are submitted at one time and are picked up by the Executor servers that are listening on the queue.

- **Target a batch job to a specific server based on message selector**

  This is the scenario we illustrated where Priority=High, but you could assign based on any criteria you wish. The JMS activation specification message selector syntax provides considerable flexibility.

  For MQ: `https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.dev.doc/q023010_.htm`

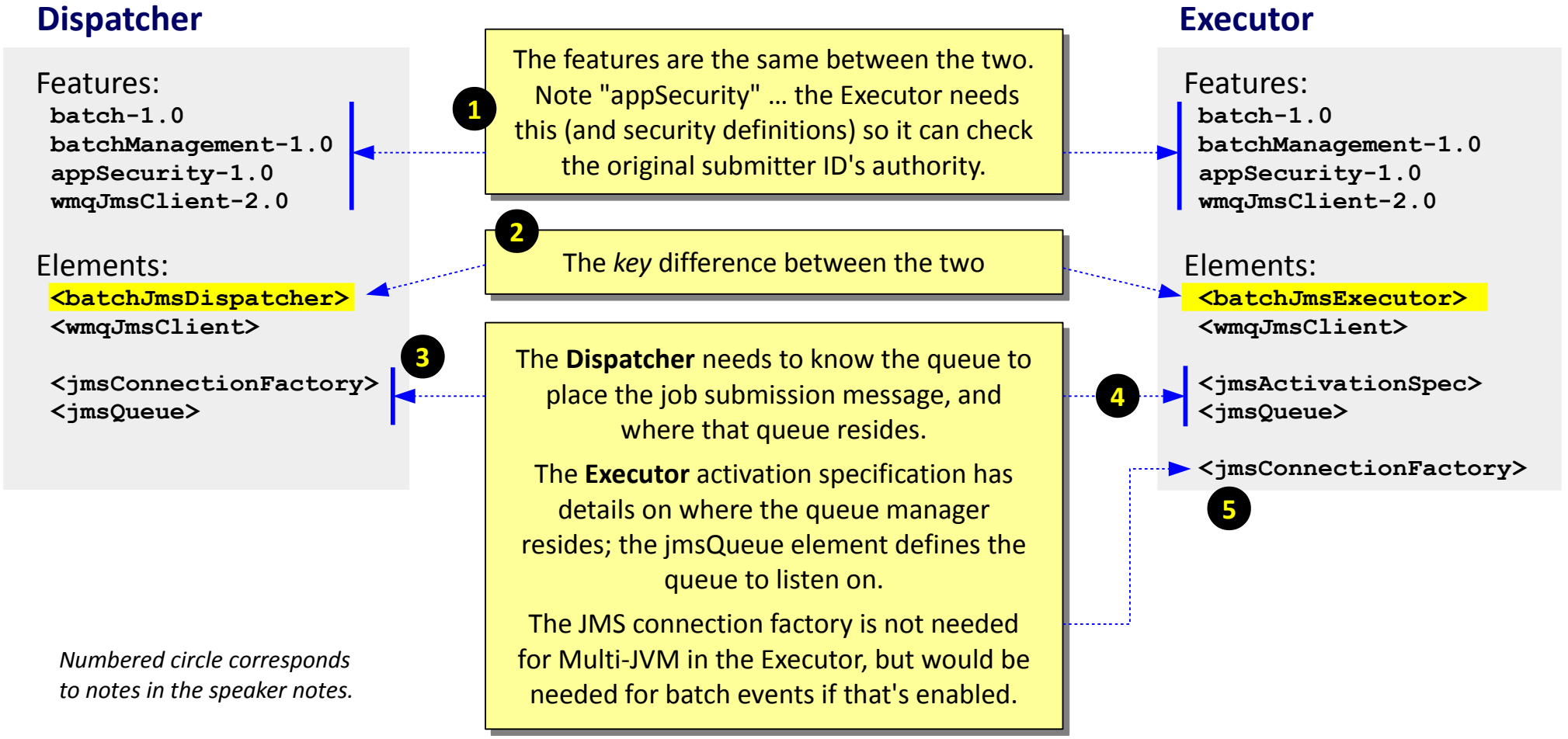
- **Separate job submission from job execution**

  The asynchronous nature of message queueing allows a job to be submitted even though no elible server is available to run it.

  You could submit all your 'batch window' jobs at any point during the day, and then start your Executor servers at the start of the batch window. When the servers came active, they would pull job submission messages off the queue and run them.

This enhances the flexibility of your Java Batch topology design

High-Level of XML Updates (some details not shown here)



WP102544 Techdoc PDF Files

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102544

Step-by-Step Implementation Guide

This document is 80+ pages long and provides a fairly detailed step-by-step instructions for configuring and using the WebSphere Liberty Java Batch solution. The document's focus is on the z/OS platform, but a great deal of the configuration information is common across platforms.

This guide can serve as a self-guided "Proof of Technology" for IBM WebSphere Liberty Java Batch.

WP102544 - WLB Step-by-Step Implementation Guide.pdf

Sample Configuration Illustration

This document provides an illustration of an actual WebSphere Liberty Java Batch configuration we have running in the test lab. This is a multi-server configuration using a "dispatcher" server and two "executor" servers, with IBM MQ as the queueing mechanism between the dispatcher and executors. This also illustrates how batch events operates.

Note: this is a *sample*, and is intended as an illustration only. Please review all security samples and configuration values and modify as needed to comply with your local standards and policies.

WP102544 - Sample Configuration.pdf

This has a detailed illustration of how to enable using the default messaging of Liberty

This has a detailed illustration of how to enable using IBM MQ.

The topology illustrated in this PDF is what we'll be doing in the lab as well.

The Dispatcher JMS Definitions in XML (illustrating IBM MQ here)

```
<batchJmsDispatcher                                    1
  connectionFactoryRef="batchConnectionFactory"
  queueRef="batchJobSubmissionQueue" />

<jmsConnectionFactory id="batchConnectionFactory"
  jndiName="jms/batch/connectionFactory">              2
  <properties.wmqJms
    hostName="wg31.washington.ibm.com"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    port="1414"
    queueManager="MQS1">
  </properties.wmqJms>
</jmsConnectionFactory>

<jmsQueue id="batchJobSubmissionQueue"                 3
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wmqJms baseQueueName="JSR.BATCH.QUEUE"
    priority="QDEF"
    baseQueueManagerName="MQS1">
  </properties.wmqJms>
</jmsQueue>
```

1. **<batchJmsDispatcher>**
   This is what enables the server to be a Dispatcher. It has two pointers: (1) to a connection factory, and (2) to a queue definition.

2. **<jmsConnectionFactory>**
   The JMS Connection Factory provides the details on how to get to the messaging engine that hosts the queue. In this example it is IBM MQ, and we're using MQ CLIENT mode (network) to access.

3. **<jmsQueue>**
   This defines the queue onto which the job submission message will be placed.

   For MQ the key is this queue must be defined as shareable with input shared.

The Executor JMS Activation Specification XML (illustrating IBM MQ here)

```
<batchJmsExecutor activationSpecRef="batchActivationSpec1"
    queueRef="batchJobSubmissionQueue"/>

<jmsActivationSpec id="batchActivationSpec1" >
  <properties.wmqJms
    destinationRef="batchJobSubmissionQueue"
    messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
    maxPoolDepth="1"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    destinationType="javax.jms.Queue"
    queueManager="MQS1"
    hostName="wg31.washington.ibm.com"
    port="1414">
  </properties.wmqJms>
</jmsActivationSpec>

<jmsQueue id="batchJobSubmissionQueue"
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wmqJms baseQueueName="JSR.BATCH.QUEUE"
     baseQueueManagerName="MQS1">
  </properties.wmqJms>
</jmsQueue>
```

1. **<batchJmsExecutor>**
   Enables server as Executor; points to activation spec and queue.
2. **<jmsActivationSpec>**
   Provides details for the activation spec, including how to access the messaging engine (or QMGR).
3. **messageSelector**
   Defines which messages to pick up. We'll focus on this in more detail in charts that follow.
4. **<jmsQueue>**
   Defines the queue to listen on.

messageSelector property ...

The messageSelector Property on the Activation Specification

Note the starting and ending double quotes. You will use single quotes inside these double quotes. Be careful not to prematurely terminate the double-quote bounary.

This is where you will code your selector attributes. You can break this definition across lines in the XML file. The key is to start with double-quote and end with it.

You can ...

- **Not code messageSelector at all**

  This works, but be very careful: the server will pick up *any* job submission message, even those for applications not deployed in that server. Coding at a minimum messageSelector= that selects on application name is recommended.

- **Code messageSelector with one selector attribute**

  There are two main types of selectors: (1) Dispatcher properties, and (2) user-defined properties:

Dispatcher Properties:	**`com_ibm_ws_batch_applicationName`** – the name of the batch application **`com_ibm_ws_batch_moduleName`** – module name of the batch application **`com_ibm_ws_batch_componentName`** – the component name of the batch application
User-defined Properties:	Any jobParameter name/value pair passed in at time of job submission

Examples coming up!

- **Code messageSelector with multiple attributes and conditional logic**

  A combination of the items shown in the table above, using AND, OR, and other conditional logic.

Simple "Application Name" Message Selector

messages.log file

```
CWWKZ0018I: Starting application SleepyBatchletSample-1.0.
SRVE0169I: Loading Web Module: SleepyBatchletSample-1.0.
SRVE0250I: Web Module SleepyBatchletSample-1.0 has been bound to default_host.
CWWKZ0001I: Application SleepyBatchletSample-1.0 started in 0.153 seconds.
```

For the SleepyBatchlet sample the module name and application name are the same. But that is not always the case.

Note the matching double quotes

```
messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
```

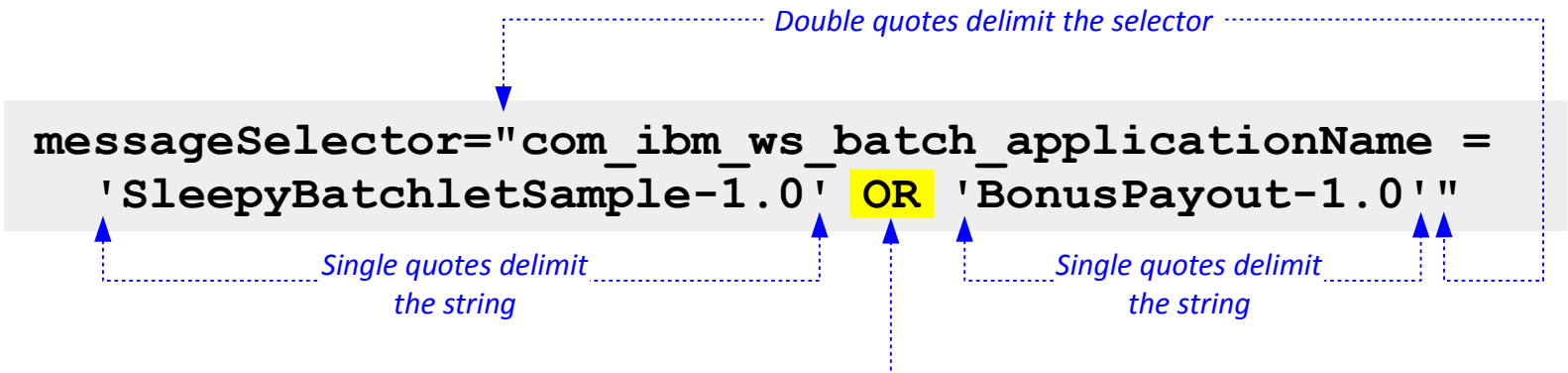
One of the three Dispatcher properties; this one for 'applicatName'

Single quotes delimit the string value

```
messageSelector="com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0'"
```

Exact same thing, just broken across two lines in the XML file

Conditional OR on Application Names



Conditional with Application Name and User Property

Executor #1

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '1'"
```

Executor #2

```
messageSelector="(com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0' AND NOT jobPriority = '1')
 OR com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
```

```
./batchManager submit ... --applicationName=SleepyBatchletSample-1.0 --jobParameter=jobPriority=1
```

This will go to Executor #1 … both conditions of AND are met: application name and jobPriority='1'

```
./batchManager submit ... --applicationName=SleepyBatchletSample-1.0 --jobParameter=jobPriority=2
```

This will go to Executor #2 … the application name matches and the AND NOT condition on jobPriority='1' is met. Notice the parenthesis that delimit that conditional expression from the OR for the other application.

```
./batchManager submit ... --applicationName=BonusPayout-1.0
```

This will go to Executor #2 based on the match of application name

One more ...

One More … Conditional with Application Name and User Property

Executor #1

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '1'"
```

Only if application name matches and jobPriority=1

Executor #2

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '2'"
```

Only if application name matches and jobPriority=2

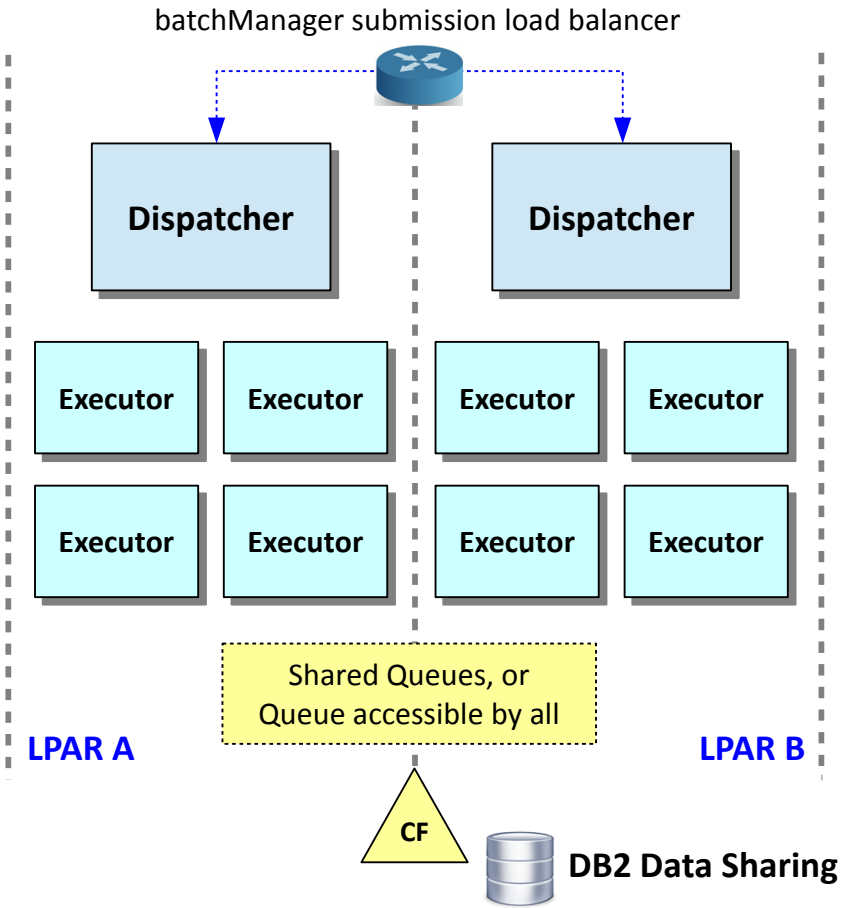
Executor #3

```
messageSelector="(com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0'
  AND NOT jobPriority = '1'
  AND NOT jobPriority = '2')
  OR com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
```

*This will pick up any SleepyBatchlet where jobPriority is **not** 1 or 2. However, jobPriority must be specified as a parameter, otherwise the message selector can't evaluate on that property and it won't select the message even if the application name matches.*

This also picks up any job submissions for the BonusPayout application.

What About Multiple Dispatchers?



Yes, this is possible … and in fact can provide a highly available Java Batch runtime environment

With Liberty's ability to share applications and configuration elements with <include> processing, along with z/OS shared file systems, some creative things can be accomplished.

Summary

The multi-JVM design allows you to asynchronously separate job submission from job execution

Properties on the JMS activation specification allows you to have servers pick up certain job submissions and ignore others

You can use this for a number of reasons: availability, priority, deferring execution until servers are started