



Batch-Modernisierung auf Basis von Java, CICS und MQ

Joerg-Ulrich Vesper

Client Technical Professional for
WebSphere on System z
jveser@de.ibm.com

WAS V8.5 Delivers

Unparalleled Application Development and Management Environment, Rich User Experiences...Faster

Developer Experience



Fast, flexible, and simplified application development

- Liberty Profile
- Expanded Tooling and WAS Tooling Bundles
- OSGi programming model enhancements
- EJB support in OSGi apps
- JDK7 Support
- Migration toolkit
- Web 2.0 & Mobile Toolkit; IBM Worklight Integration
- SCA OASIS programming model

Application Resiliency



Intelligent Management & Enhanced Resiliency

- Application Edition Management
- Application Server Health Management
- Dynamic Clustering
- New Intelligent Routing capabilities
- Messaging infrastructure resiliency
- Memory leak detection & protection in WAS

Operations and Control



Improved Operations, Security, Control & Integration

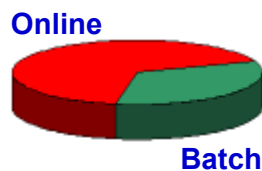
- Selectable JDK
- **WebSphere Batch enhancements**
- Admin Security Audit
- OSGi Blueprint security improvements
- Cross Component Trace (XCT)
- Enhanced IBM Support Assistant
- Better log and trace filtering

Concept of "Dedicated Batch" Window Going Away

Windows of time which used to be dedicated to batch processing are shrinking.
The demands of online processing require more and more ...



In the past ...



Today ...



24 x 7 x 365 Access

Users of your online systems expect availability at all hours
Users from other parts of the world means availability is expected around the clock



Mobile Users

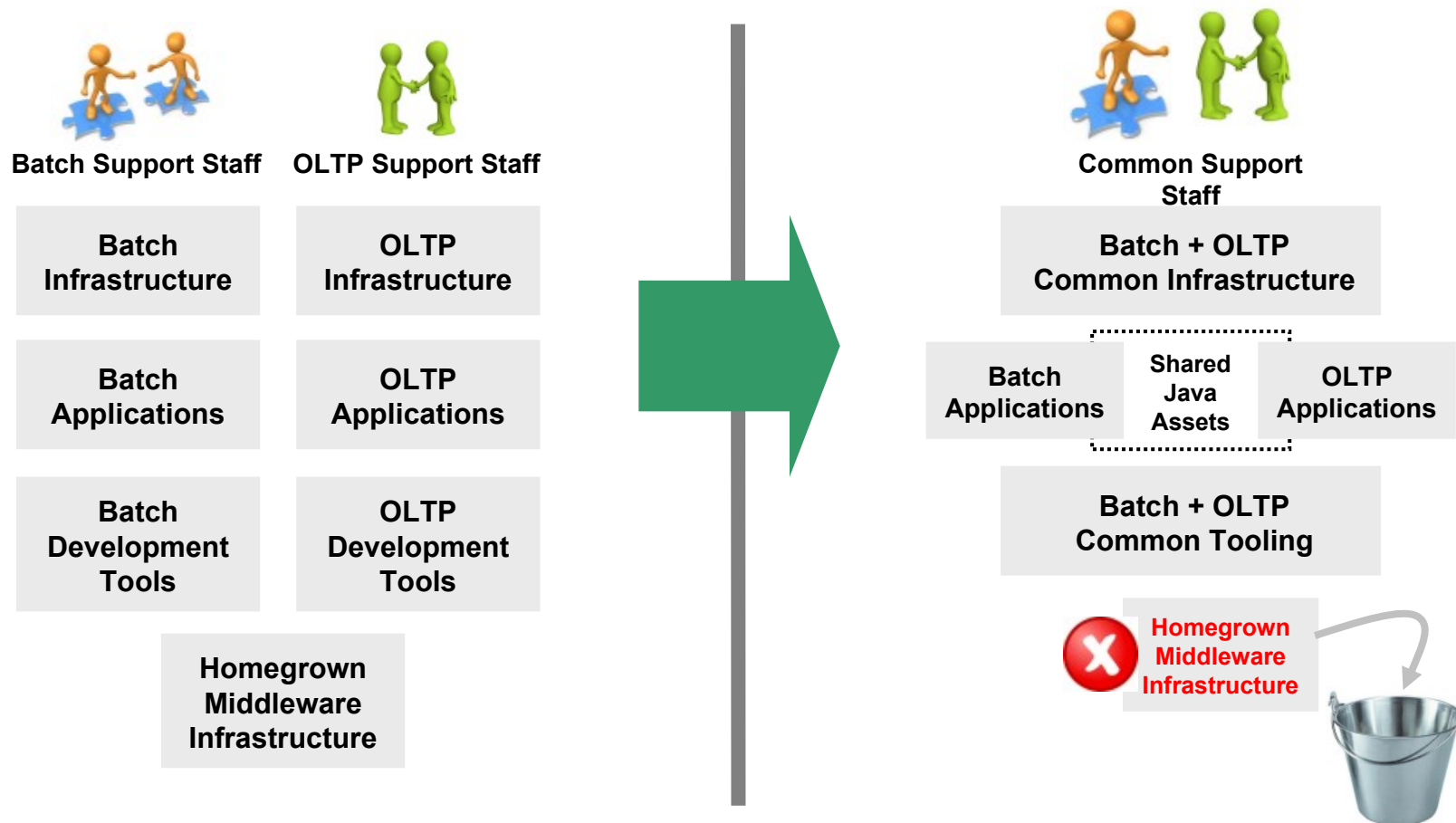
Users are no longer tied to a desk and a computer. Today users have access to mobile computing devices that are with the user wherever they may be. Day or night, home or office.

The need to process batch work has *not* gone away.

The need to perform the work concurrent with OLTP has emerged.

The Value of Shared Services

It's not *just* that the window is shrinking ... it's also the cost pressures on maintaining the batch and OLTP environments:



Efficiencies through consolidation around common assets

Java for Batch Processing?

Yes ... for many very good reasons:



Availability of Skills

Java is a programming language with wide adoption in the industry. Skills for Java programming are common and affordable.

Tooling Support

Development tooling for Java has advanced to the point where some tools (IBM Rational Application Developer) are very powerful and sophisticated.

This also provides an opportunity to consolidate to a common tooling environment for both OLTP and batch development.

z/OS Specialty Engines

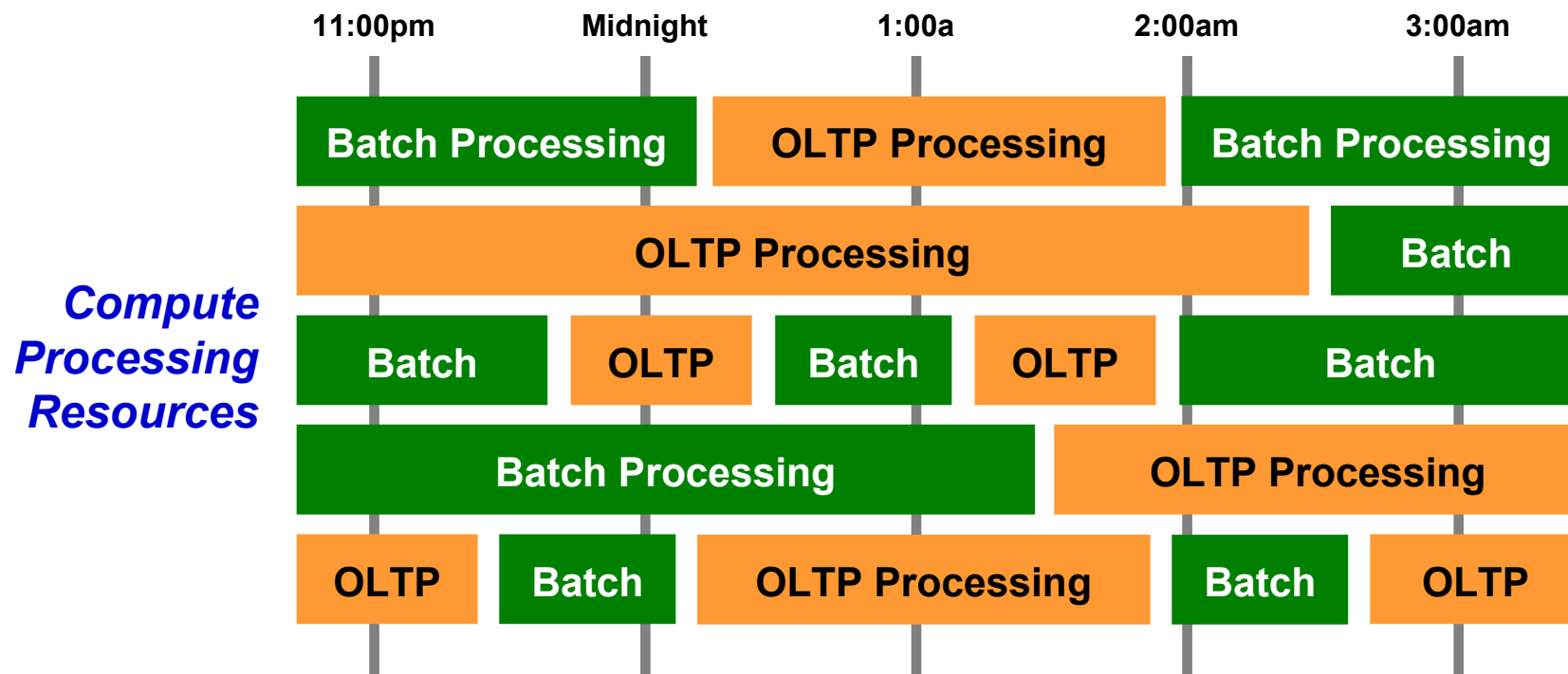
Pressures on cost containment often dictate greater use of z/OS specialty engines. Java offloads to zAAP. Java batch does as well.

Processing in OLTP Runtime

Running Java batch in the same execution runtime as Java OLTP provides an opportunity to mix and manage the two processing types together under the same management model.

The Objective -- OLTP and Batch Mixed and Managed:

OLTP and Batch do not need to be "either / or" ... it can be "both":



With IBM WebSphere Batch this is possible. OLTP and Batch processing within a common execution runtime (WebSphere Application Server) allows the WAS platform to mix and manage the two workload types.

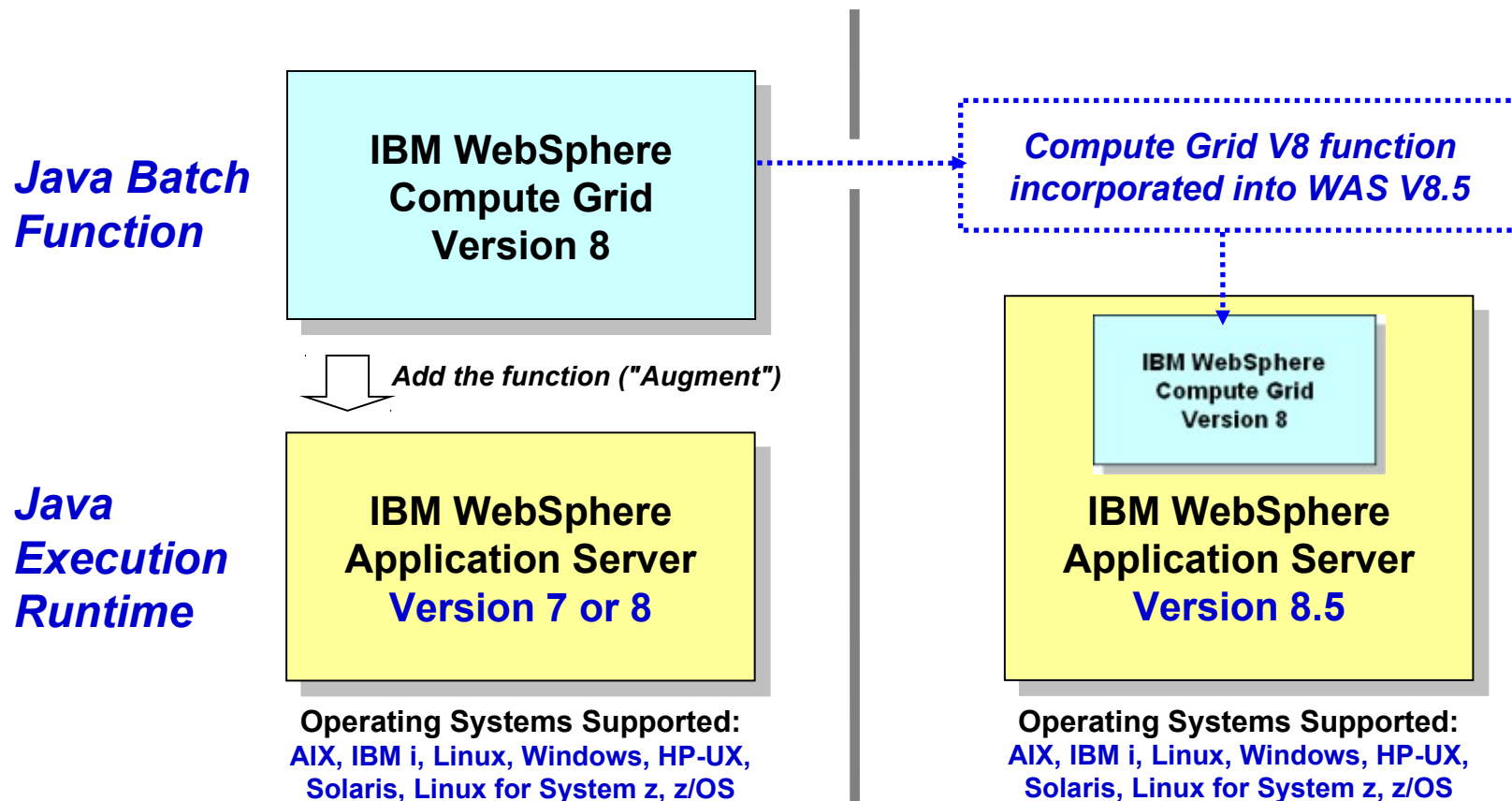


Overview

**A high-level look at the IBM WebSphere
Java Batch model**

IBM Compute Grid V8 and IBM WAS V8.5

The IBM WebSphere Java Batch function is provided in two ways today:

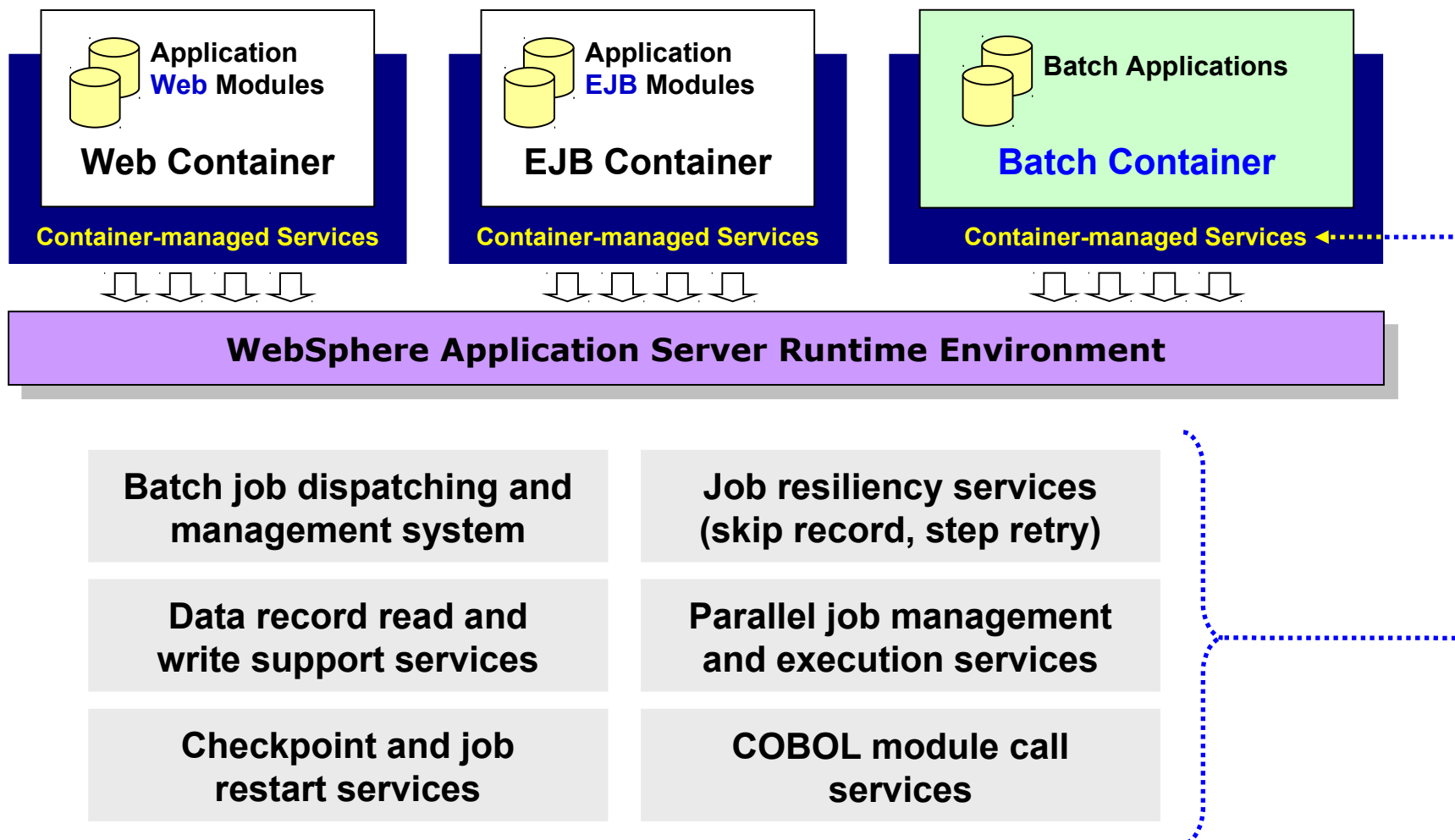


Function is identical between the two environments

Compute Grid V8 available for those who have not yet migrated their execution runtimes to WAS V8.5

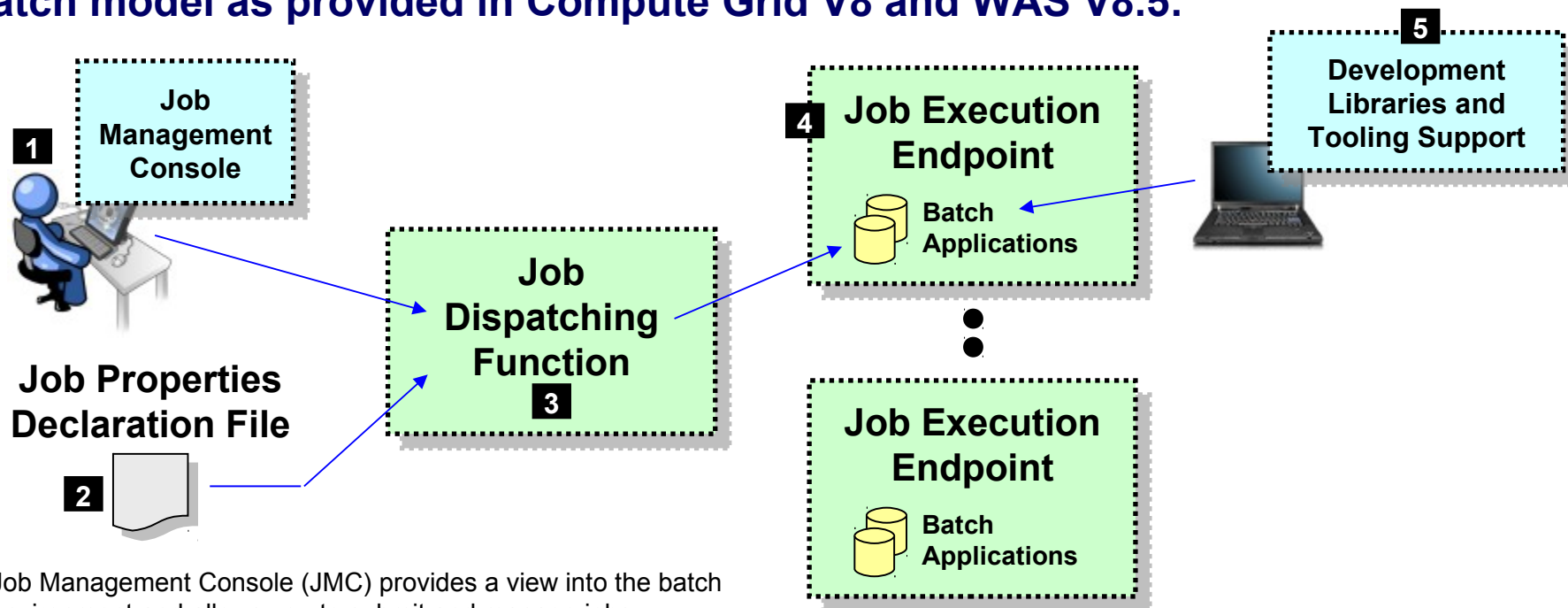
Batch Container Added to the WAS Runtime

At a very high-level, you may think the IBM WebSphere Java Batch function as a "batch container" operating alongside the other containers of WAS itself:



Overview of the Management and Execution Model

This picture illustrates some of the key components of the WebSphere Java Batch model as provided in Compute Grid V8 and WAS V8.5:



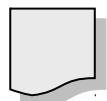
1. Job Management Console (JMC) provides a view into the batch environment and allows you to submit and manage jobs
2. Job declaration file (xJCL) provides information about the job to be run, such as the steps, the data input and output streams and the batch class files to invoke
3. The Job Dispatching function interprets the xJCL, dispatches the job to the endpoint where the batch application resides, and provides ability to stop and restart jobs
4. The Execution Endpoint is a WAS server in which the deployed batch applications run
5. The development libraries and tooling assist in the creation of the batch applications

**A comprehensive Java
batch execution platform**
Built on the proven Java runtime environment
of WebSphere Application Server

Batch Job and Batch Job Steps

A batch job consists of one or more steps executed in order specified in xJCL:

xJCL



Job

Properties of the overall job

Job Step 1

- Java class
- Input and output declarations
- Other properties of the step

Job Step 2

- Java class
- Input and output declarations
- Other properties of the step



Job Step n

- Java class
- Input and output declarations
- Other properties of the step

The xJCL is submitted through the Job Management Console

Interfaces provided: [HTTP browser](#), [command Line](#), [Web Services](#), [RMI](#)

The Job Dispatching function interprets xJCL and determines which endpoint has batch application class files deployed

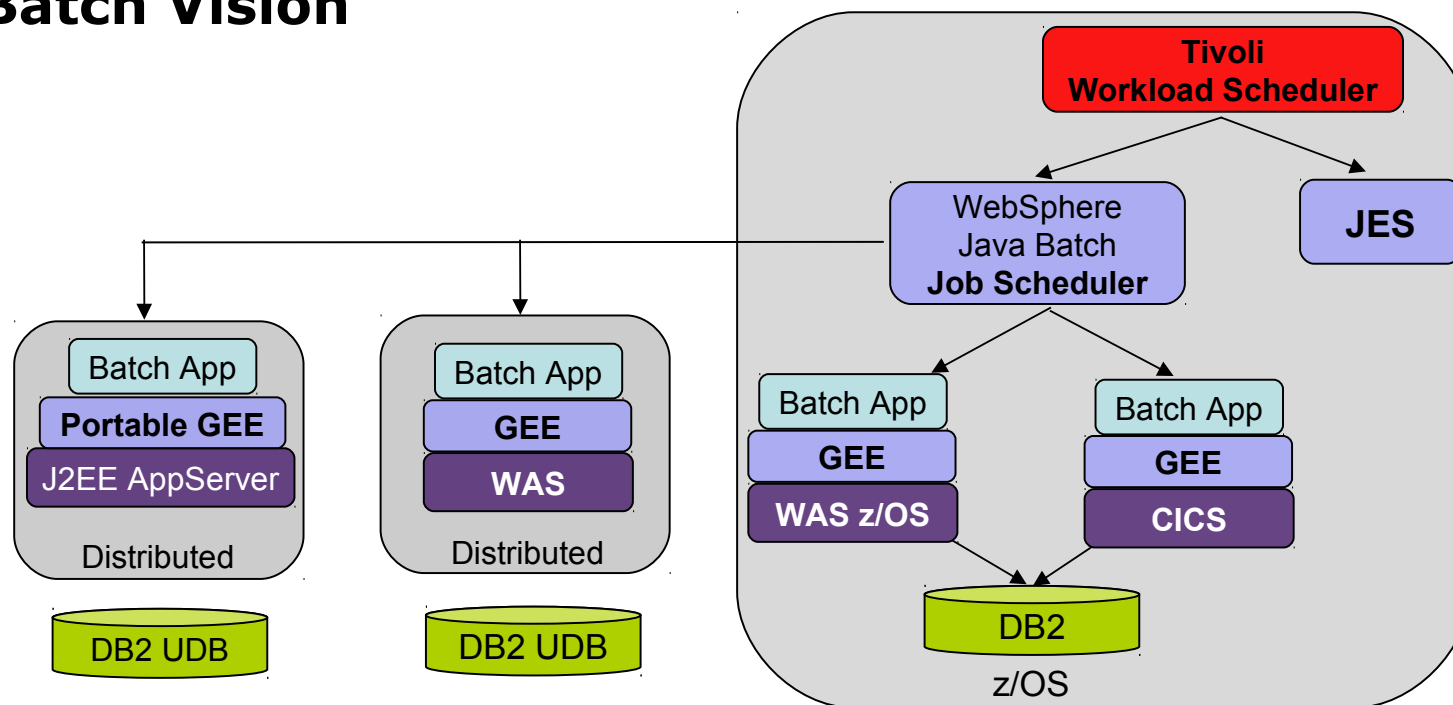
Dispatching Function invokes job and passes to the endpoint an object containing all the properties in xJCL

Steps are executed in order, with conditional step processing if declared

Dispatching Function maintains awareness of job state

When job ends, job output file accessible through Job Management Console

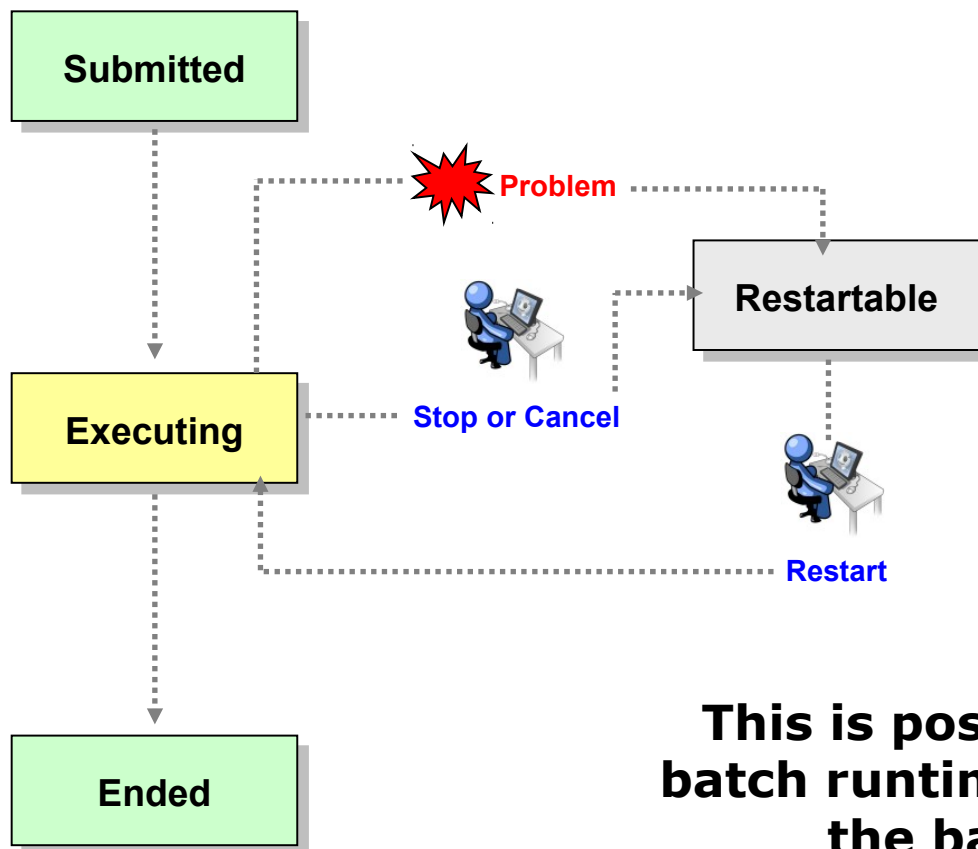
The Batch Vision



- **Portable Batch applications** across platforms and J2EE vendors
- Location of the data dictates the placement of the batch application
- Flexible programming model, will host Spring Batch, JZOS, Compute Grid apps
- Centrally managed by your enterprise scheduler
- z/OS operational procedures manage batch across all platforms

Job Execution "State"

The following picture illustrates a simplified view of the job states ... it helps illustrate a key point: *executing jobs can be acted upon; failed jobs restarted.*



The Job Management Console provides you ability to act upon an executing job

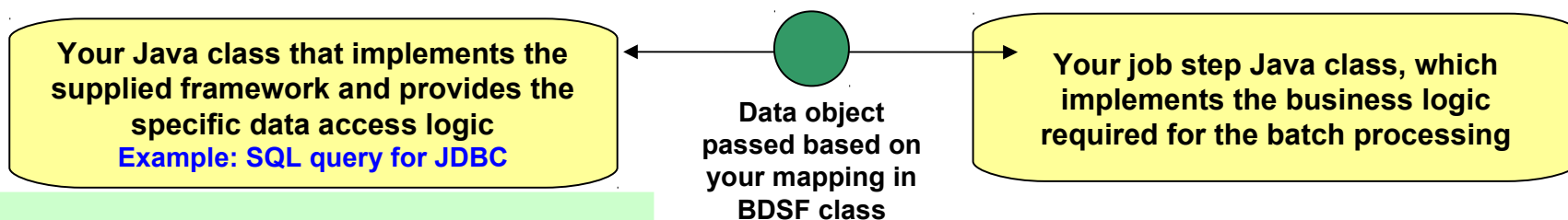
The Batch Container is maintaining checkpoint status and will restart at the last checkpoint interval

This is possible because of the Java batch runtime services that are part of the batch container model

If you were to write this yourself then just what's shown here would require a significant amount of custom batch middleware code. IBM WebSphere Java Batch provides that as part of the product.

Batch Data Stream Framework (BDSF)

This is a key function service provided by the batch container - it abstracts data read and write operations so your code may focus on the business logic:



Batch Data Stream Framework

Supplied "patterns" for data access:

- JDBC read or write operations
- JPA read or write operations
- File read or write operations
- z/OS Data Set read or write operations



Batch Data Stream retrieves result set from data persistence store (DB, file, etc.)

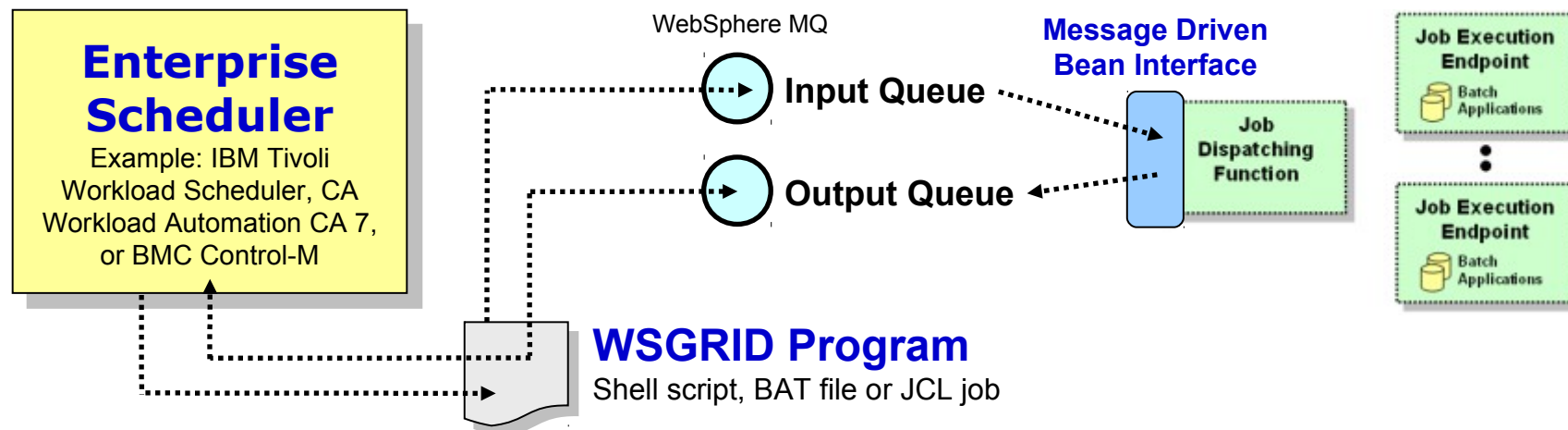
Batch Data Stream maps data fields to data object

For each record in result set, BDSF invokes your job step, passing a data object mapped to your specifications

Your job step code stays focused on business logic, not Java stream handling and data object formatting

Integration with Enterprise Scheduler Functions

The Job Dispatching Function has a Message Driven Bean (MDB) interface. IBM supplies a program that integrates schedulers with WebSphere Java Batch:



WSGRID is seen by Scheduler as any other batch job it starts and monitors

WSGRID interacts with Job Dispatching, submitting the job and processing Java batch job output back to STDOUT or JES Spool if z/OS

WSGRID program stays up for life of job in WebSphere Java Batch

To the Scheduler, WGRID is the Java Batch job ... but behind WSGRID is all the WebSphere Java Batch function we'll discuss

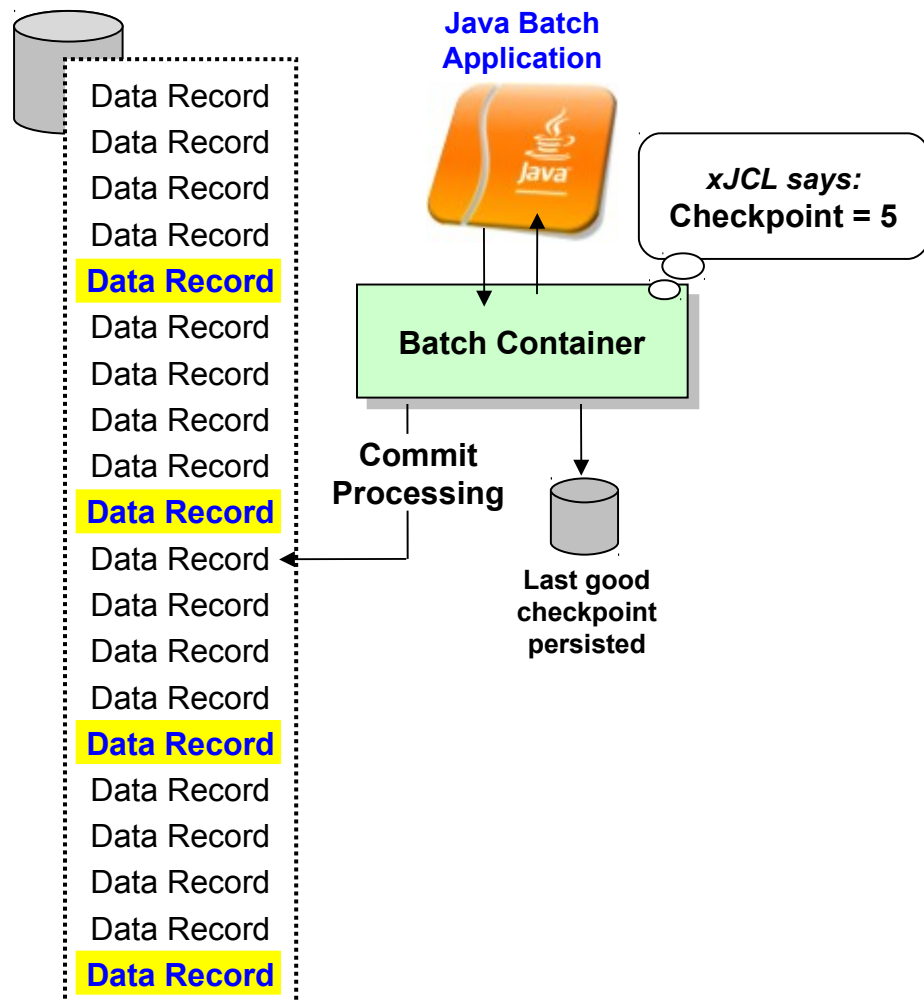


Feature Focus

**A closer look at some of the features
and functions of the IBM WebSphere
Java Batch model**

Transactional Checkpoint Processing

The batch container provides the ability to checkpoint at intervals based on either record count or time. The container keeps track of last checkpoint.



Checkpoint interval (record or time) specified in the xJCL

This is a function of the batch container, *not* your application code

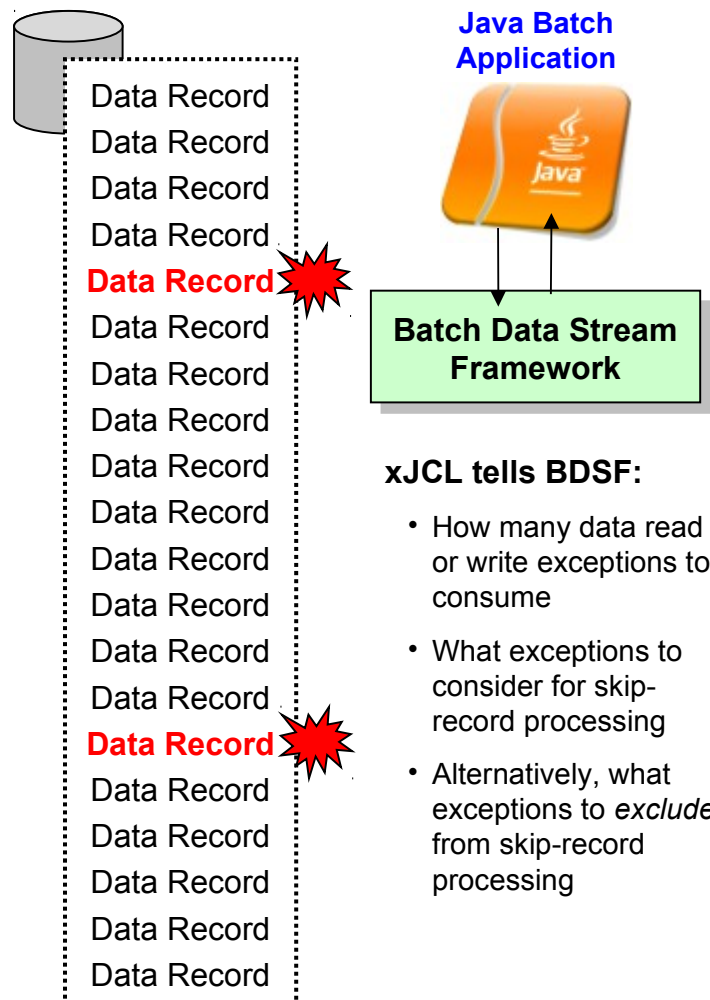
As checkpoint intervals are reached, container commits and records the checkpoint attained

In the event of a failure, job may be restarted at the last good checkpoint

Set the checkpoint interval based on your knowledge of balance between recoverability and efficiency

Skip-Record Processing

Provides a container-managed way of tolerating data read or write errors so the job itself may continue on. Information about data errors may be logged.



Objective: allow job to continue if a data read or write exception occurs in BDSF

Why fail a million-record job just because of one or two read or write exceptions? Better to complete the job and allow auditors to go back and investigate the few exceptions.

Skip-Record processing allows BDSF to keep exception and *not* surface it to your application

This takes burden off your application code to explicitly handle data read or write exceptions that may occur

A "skip-record listener" may be called so your code may log information about skipped record

More on "batch listeners" coming up

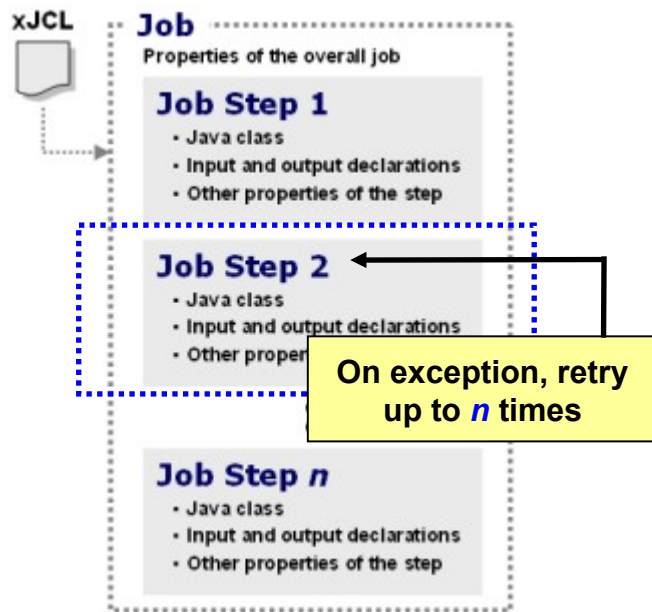
xJCL properties allow you to specify how many records may be skipped and what exceptions to include or exclude from consideration

When skip limit is reached, further exceptions are surfaced to application. That may result in job failing and going into a restartable state

Normal restart-at-checkpoint would occur

Retry-Step Processing

Provides a means of retrying a job step in the event of an exception thrown. If successful on retry then the job continues and your processing completes.



xJCL tells Container:

- How many step retries may be attempted
- What exceptions to consider for retry-step processing
- Alternatively, what exceptions to *exclude* from retry-step processing
- Whether to process a delay before attempting a retry of the step

Objective: retry step in attempt to allow overall job to continue and complete when an unanticipated exception is thrown

This is at level higher than skip-record ... this is if an unhandled exception is thrown when the job step function is called

Batch container falls back to last good checkpoint and restarts from there

A "retry-step listener" may be called so you can perform custom action upon retry-step processing

[More on "batch listeners" coming up](#)

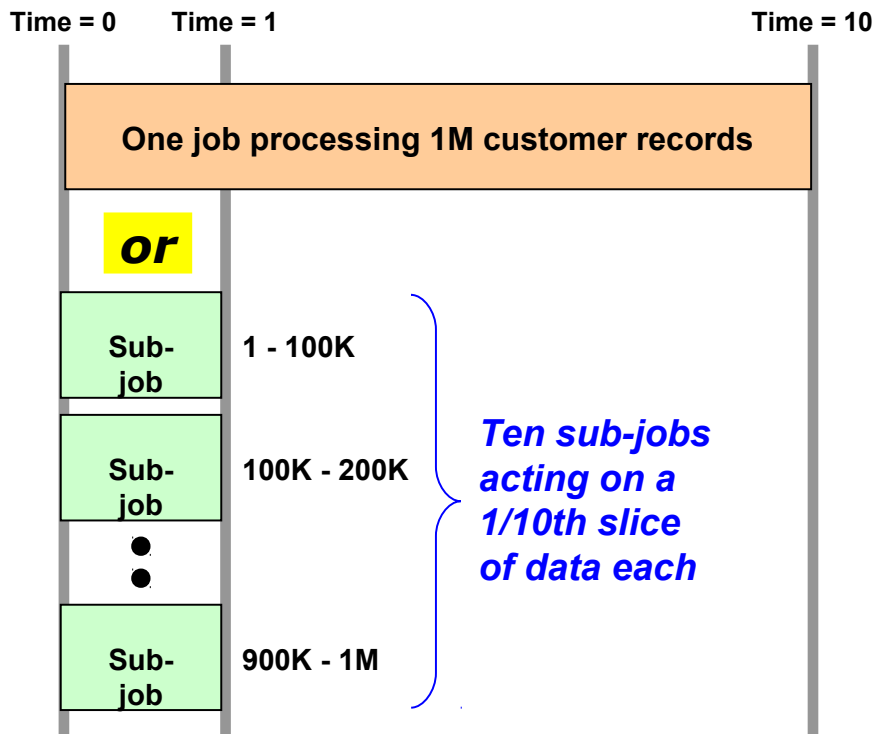
xJCL properties allow you to specify how many retry attempts will be performed and what exceptions to include or exclude from consideration

When retry limit is reached, job will go into restartable state

[Normal restart-at-checkpoint would occur](#)

Parallel Job Manager

The Parallel Job Manager (PJM) provides a way to "parameterize" logic so parallel sub-jobs may act on a slice of the overall batch job data:



xJCL specifies whether job is to be run in parallel, and if so how:

- One JVM, multiple threads
- Multiple JVMs

Your "parameterizer" code is called at start so data range may be segmented into sub-job slices

Job is submitted, then PJM dispatches "sub-jobs" to act on each data range

"Parameterizer" code constructs data range query strings to be used by each sub-job

PJM manages "top-job" and all subordinate "sub-jobs" to completion

Objective is reduction in overall job completion time

Which shortens overall batch window if other jobs are dependent on this job for completion

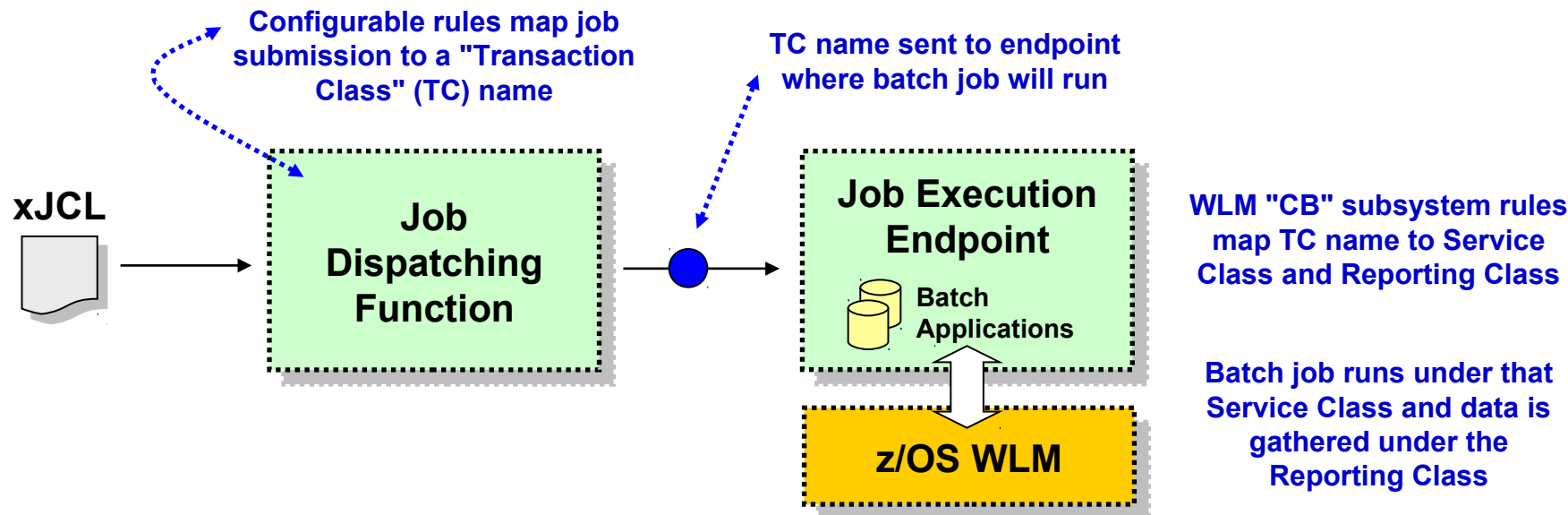


Java Batch on z/OS

**A review of what IBM WebSphere Java
Batch brings specific to z/OS**

WLM Classification

The submitted job can be tagged with a WLM "transaction class," which may then be used to map the job to a WLM Service Class or Reporting Class:



Classifying to a Service Class
allows WAS z/OS to place work
into separate servant regions
based on Service Class

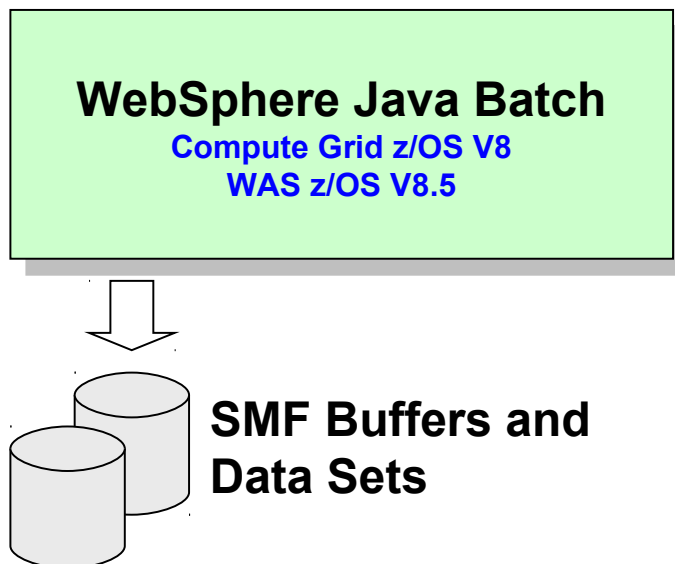
A somewhat sophisticated practice not widely used

Classifying to a Reporting Class
allows WLM to gather system
information for all work running
under that Class

*A much more common practice that is very useful for
understanding usage patterns and for capacity planning*

SMF 120.9 Activity Recording

WAS z/OS supports the use of activity recording using the SMF 120.9 record. WebSphere Java Batch extends the record with batch activity information:



Job activity records allow you to understand how your system is being used and to provide chargeback data

Activity recording available on all platforms, but only z/OS uses SMF, which is an extremely efficient logging mechanism

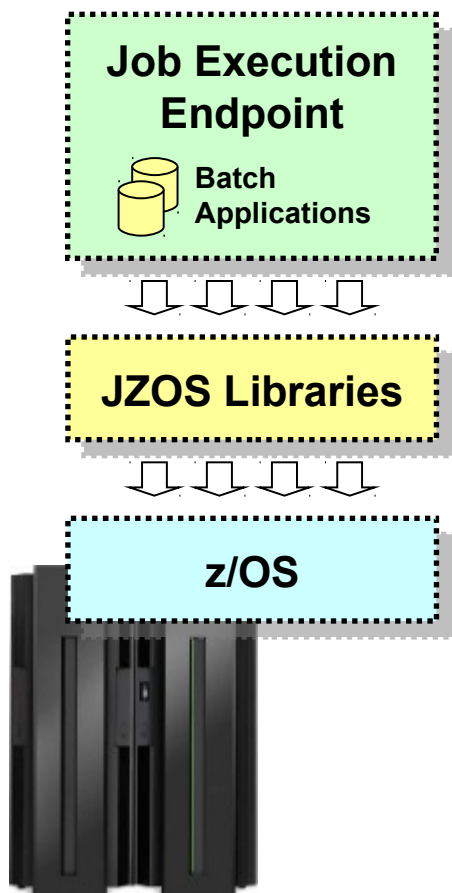
Provides historical records for usage analysis and batch capacity planning

Information captured:

- Job submitter
- Date and time of submission
- Final job state
- Total CPU used for job
- General processor used for job
- zAAP usage derived: $\text{Total} - \text{GP} = \text{zAAP}$

Use of JZOS Services

JZOS is a set of functions that make using Java on z/OS much easier and useful. The JZOS class libraries may be used in batch application development:



JZOS is technology acquired by IBM from Dovetail Technologies* and incorporated into z/OS**

Examples of some z/OS services available:

DfSort - interface for invoking DFSORT

MvsConsole - class with static methods to interface with the MVS console.

MvsJobSubmitter - class for submitting batch jobs to JES2 or JES3 from a Java program

PdsDirectory - class for opening a PDS directory and iterating over its members.

WtoMessage - simple data object/bean for holding a WTO message and its parameters.

ZUtil - static interface to various z/OS native library calls other than I/O.

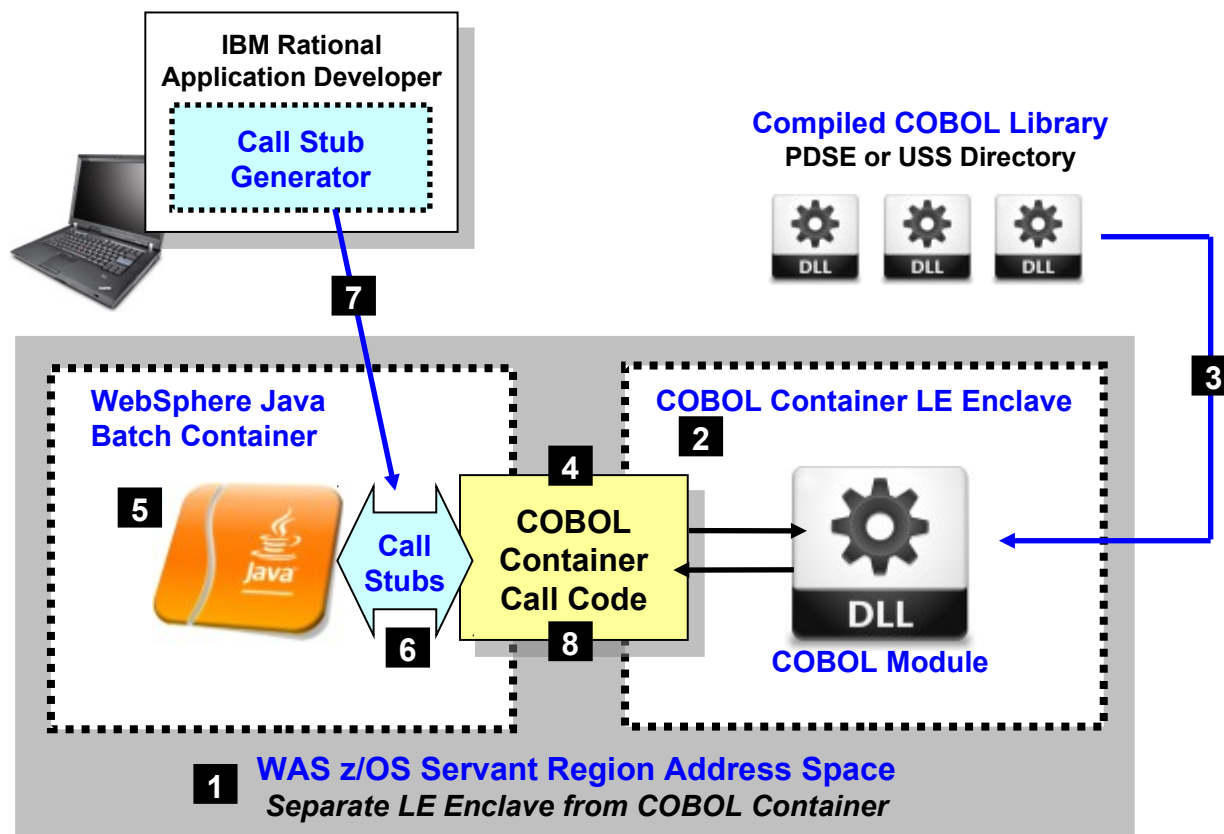
WebSphere Java Batch and JZOS are not mutually exclusive ... the JZOS class libraries may provide exactly what you need for your batch application to access z/OS functions and services

* www.dovetail.com

** <http://www-03.ibm.com/systems/z/os/zos/tools/java/products/jzos/overview.html>

COBOL Container

The COBOL Container provides a way to call and execute COBOL modules in the WAS z/OS server address space ... a *very efficient* way to call COBOL



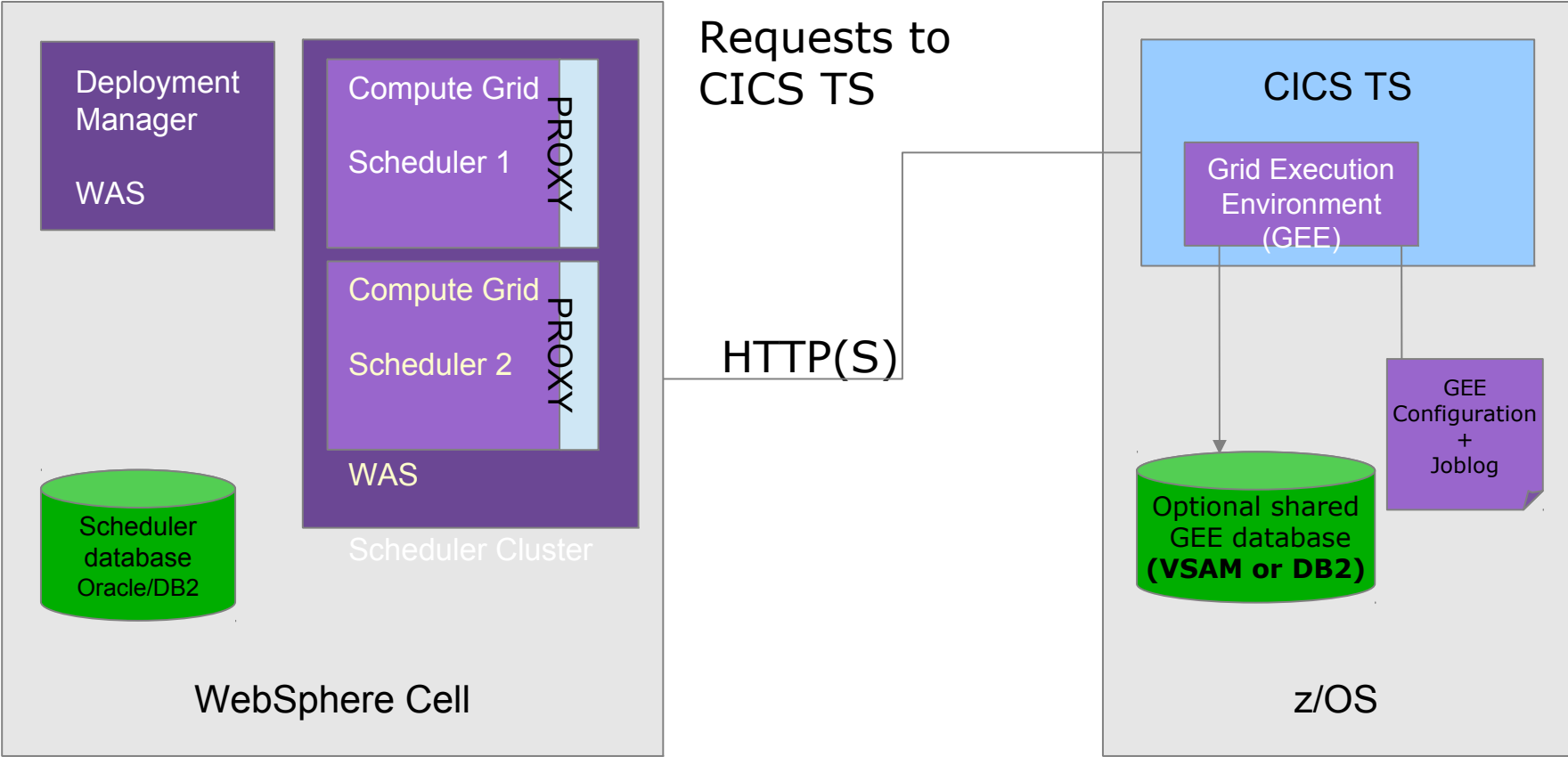
1. Batch application runs in the WAS z/OS servant region address space
2. The COBOL container is created as a separate LE enclave in the address space
3. COBOL DLLs are accessed using STEPLIB or LIBPATH
4. COBOL Container code provides the "glue" between the Java environment and the native COBOL
5. Java batch code uses supplied class methods to create the container and use it
6. Call stubs provide an easy way to call the COBOL DLL and marshal data back and forth
7. The call stubs are generated by a supplied utility that uses COBOL source to understand data bindings
8. JDBC Type 2 connections created in the Java batch program may be shared into the COBOL module in the COBOL Container

**Lines of code needed to invoke COBOL many times
less than other means of calling COBOL from Java**

Why run batch inside CICS?

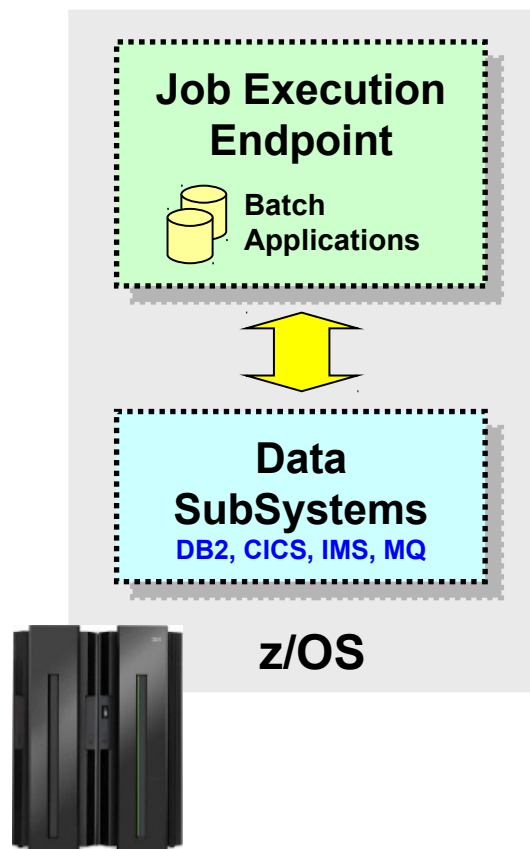
- Batch jobs and online transactions can run in parallel sharing the same data
- Resources and business logic shared
- VSAM files not closed and hence availability is increased
- Use JCICS API to talk to CICS COBOL programs from Java with transaction support
- Compute Grid provides
 - General job dispatching, management, execute control, monitoring
- Works with other schedulers, eg. Tivoli Workload Scheduler
 - Higher throughput to address larger data volumes
- Process jobs in parallel across multiple CICS regions
 - Locking of data
- Updates are synchronised at configurable Checkpoints
- Checkpoint includes positions in input and output resources
 - Failure/Recovery scenarios
- If batch jobstep fails, rollback updates, restore last checkpoint, cursors to input and output resources restored, and jobstep retried

WebSphere Java Batch and CICS TS with SupportPac CN11



Co-Location on z/OS

With the WebSphere Java Batch function on z/OS several advantages surface:



Use of cross-memory connectors for high-speed and low-latency access to data

- JDBC Type 2 connector for access to DB2
- CICS Transaction Gateway (CTG) local EXCI
- WebSphere Optimized Local Adapters (WOLA)

Much more secure -- cross memory data exchanges can *not* be 'sniffed' or intercepted

Parallel Sysplex data sharing provides highly available clustered environment *without* reliance on a single instance of a data subsystem

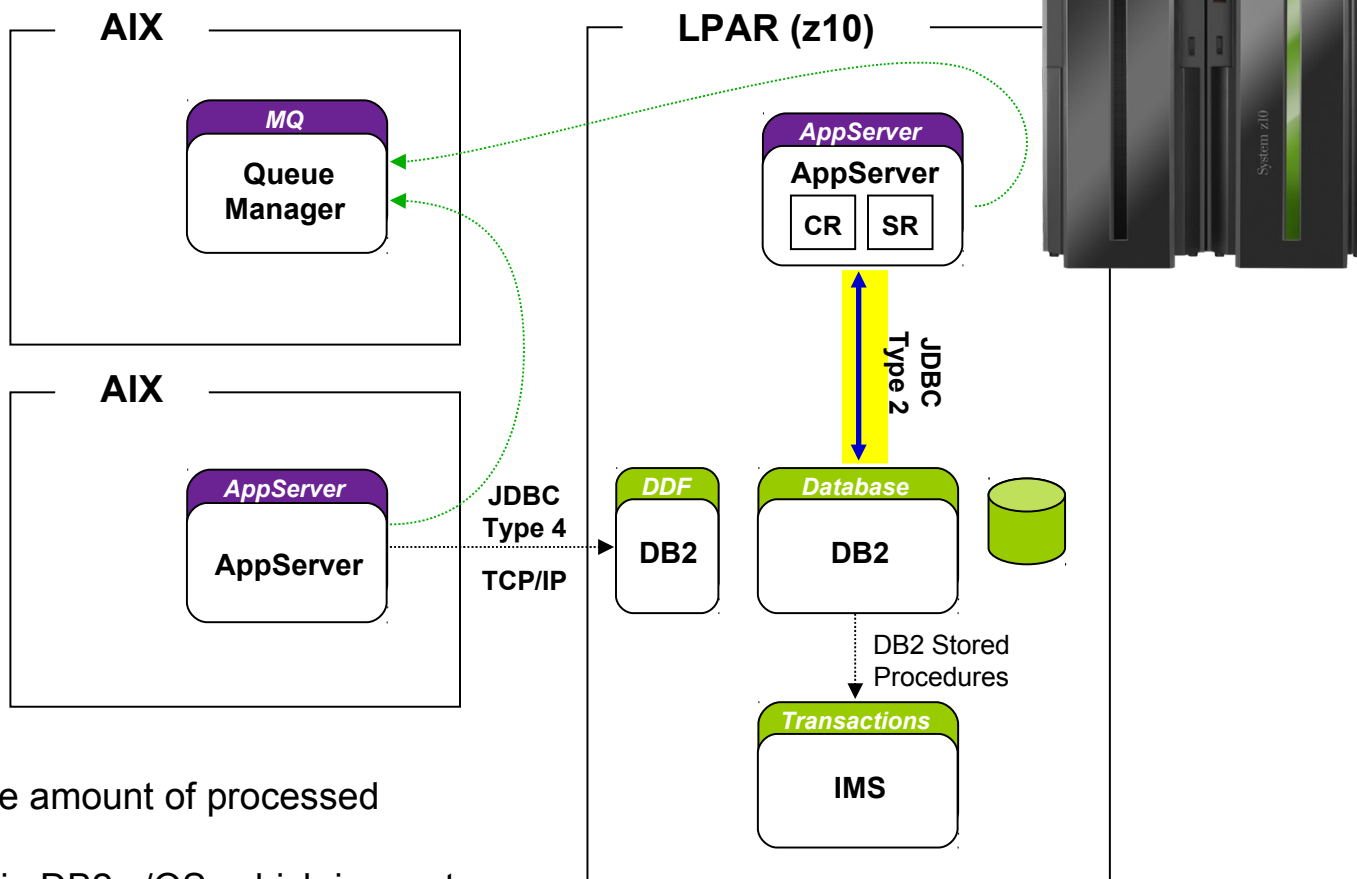
Use of COBOL Container technology for re-use of COBOL assets in very efficient calling pattern

Use of WebSphere MQ Bindings Mode for integration with Enterprise Scheduler for very fast job submission and job output return

Reduction of per-access latency is critical when dealing with large volumes of records where job completion time is important

WAS z/OS Java Batch PoC Architecture

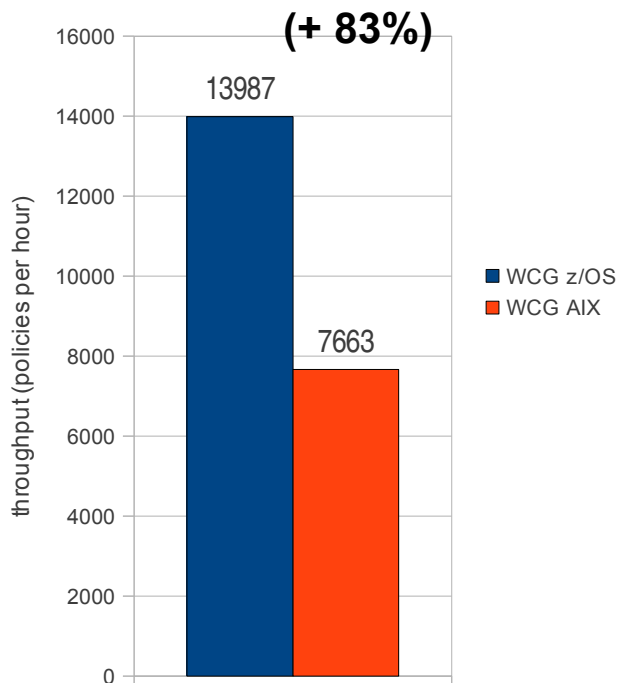
An insurance company has an existing WCG infrastructure on distributed AIX with data intensive JEE applications, which accesses DB2 z/OS via JDBC type 4. Within the scope of this PoC the Java batch job, which calculates the dynamic of the accident insurance is evaluated on WCG z/OS using JDBC type 2 cross memory adapter.



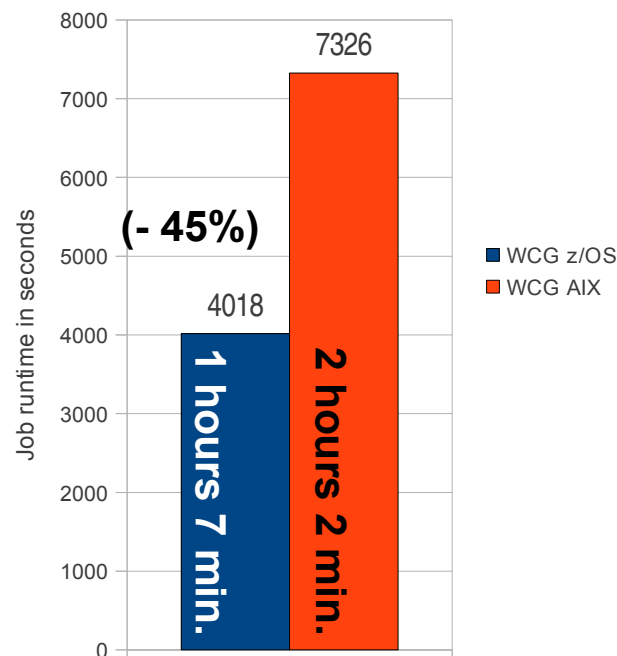
- ✓ Same job with same amount of processed insurance policies
- ✓ Same data basis in DB2 z/OS, which is reset after each run

WAS z/OS Java Batch PoC Results

Throughput platform comparison
based on 15592 policies



Job runtime platform comparison
based on 15592 policies



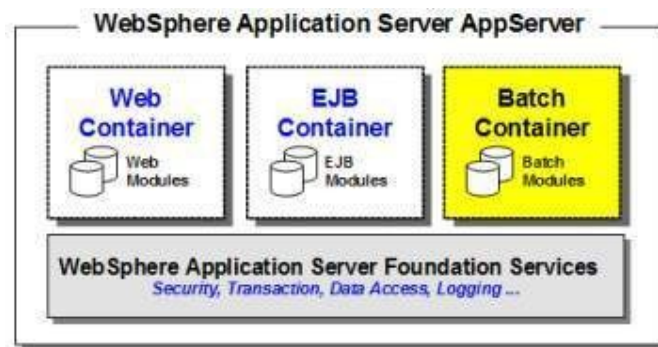
The same job runs on both platforms WCG AIX and WCG z/OS. Because of the proximity to data in DB2 z/O the job, which runs on **WCG z/OS** has **83.5% more throughput** and the **job runtime is shortened by 45%**.

WebSphere Java Batch

WebSphere Application Server v8.5 integrates capabilities from WebSphere Compute Grid and delivers a complete **enterprise level Java batch** processing solution

Key Features:

- Java Batch programming model
- Java Batch container built on WAS QoS
- Development and deployment tooling
- Batch execution environment
- Concurrent OLTP and batch workloads
- Enterprise scheduler integration
- Parallel processing of batch jobs
- Container based checkpoint and restart
- Mixed batch workloads
- COBOL support on z/OS



Compute Grid
capabilities
integrated
into
WAS 8.5

WebSphere Java Batch - Value Proposition

Move batch into the WebSphere environment and **integrate with OLTP** to gain the benefits of **concurrent processing**, **shared business logic**, and **cost efficiencies**

- **Reliable batch infrastructure** – Built on the proven Qualities of Service delivered by WebSphere Application Server.
- **Incremental modernization** – Move at your pace to reduce risk.
- **Resource efficiencies** – Focus resources on business logic and leave the infrastructure to WebSphere
- **Enterprise integration** – Integrate with existing enterprise schedulers to help deliver a robust end-to-end solution.
- **Enables new execution patterns** – Dynamic OLTP and Batch runtime environment built on WebSphere; highly parallel batch jobs; and many others.
- **Supports a SOA strategy of reuse** – Enable the cost effective sharing of business logic across both the OLTP and Batch paradigms.
- **Eliminate batch windows** – Transition from traditional batch windows to running batch 24x7 concurrent with OLTP.

WebSphere Java Batch - Key Use Cases

Evolve to a **single infrastructure for both OLTP and Batch** that enables you to **leverage existing applications** and **focus resources on business logic**

- **Batch Modernization** – Migrate from a native batch runtime, typically developed in programming languages like C, C++, PL/I, and COBOL, to Java.
- **Highly Parallel Batch Jobs** – Execute a single large batch job that is broken into chunks and executed concurrently across a grid of resources.
- **Dynamic OLTP & Batch Runtime** – Dynamically provision resources for execution to meet operational goals.
- **Batch as a Service** – Expose business capabilities as a service and leverage usage accounting features for tracking and chargeback.
- **Replace Homegrown Batch Frameworks** – Eliminate costly proprietary batch infrastructures and focus development resources on business logic.
- **Shared business logic across OLTP and Batch** – Leverage the proven WebSphere platform to share logic across both batch and OLTP.