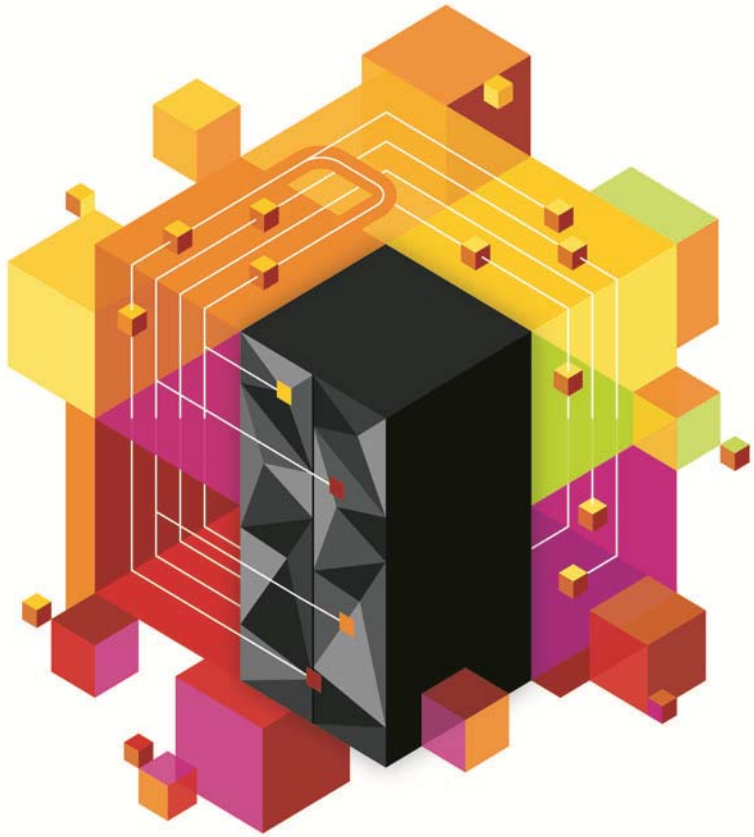
Hey, Who closed my batch window?

WebSphere Batch on System z



Agenda

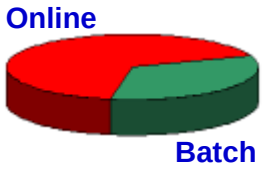
- Business Pressures on Traditional Batch
- IBM WebSphere Java Batch Overview
- IBM WebSphere Java Batch Feature Focus
- IBM WebSphere Java Batch for z/OS Focus
- IBM WebSphere Java Batch Deployment Scenarios
- Wrap-Up Summary

Business Pressures on Traditional Batch

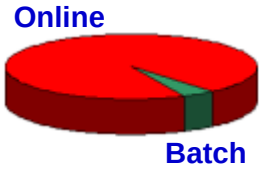
Concept of "Dedicated Batch" Window Going Away

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.

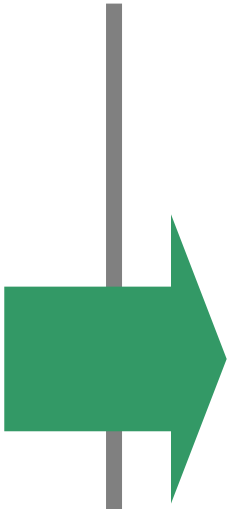
The Value of Shared Services

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

Batch Support Staff

OLTP Support Staff

- Batch Infrastructure
- OLTP Infrastructure
- Batch Applications
- OLTP Applications
- Batch Development Tools
- OLTP Development Tools
- Homegrown Middleware Infrastructure



Common Support Staff

- Batch + OLTP Common Infrastructure
- Batch Applications
- Shared Java Assets
- OLTP Applications
- Batch + OLTP Common Tooling
- Homegrown Middleware Infrastructure

Efficiencies through consolidation around common assets

Java for Batch Processing?

Yes ... for many very good reasons:

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.

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.

Processing in OLTP Runtime

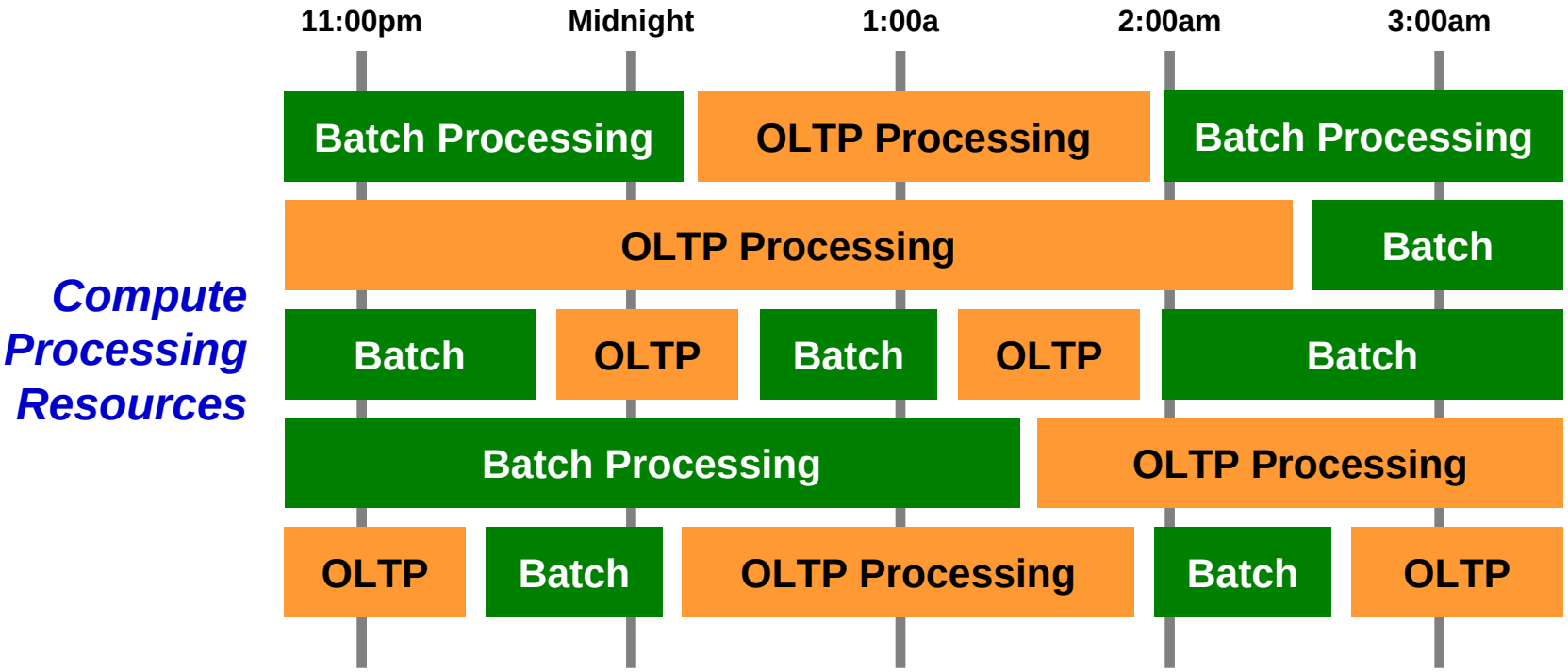
Running Java batch in the same execution runtime as Java OLTP provides an opportunity to mix and manage the two processing types together under the same management model.

Availability of Skills

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

The Objective -- OLTP and Batch Mixed and Managed:

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



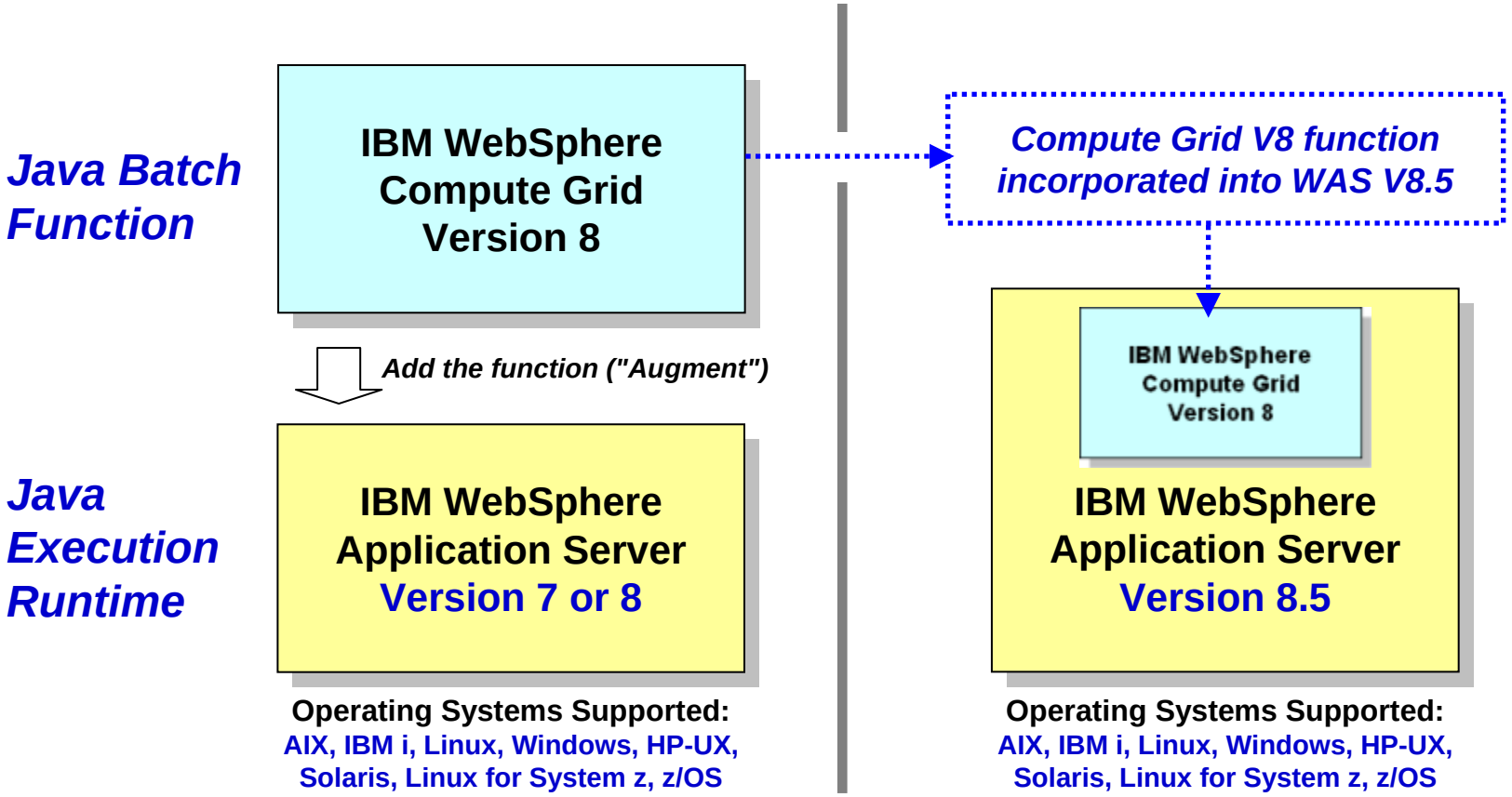
With IBM WebSphere Batch this is possible. OLTP and Batch processing within a common execution runtime (WebSphere Application Server) allows the WAS platform to mix and manage the two workload types.

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:

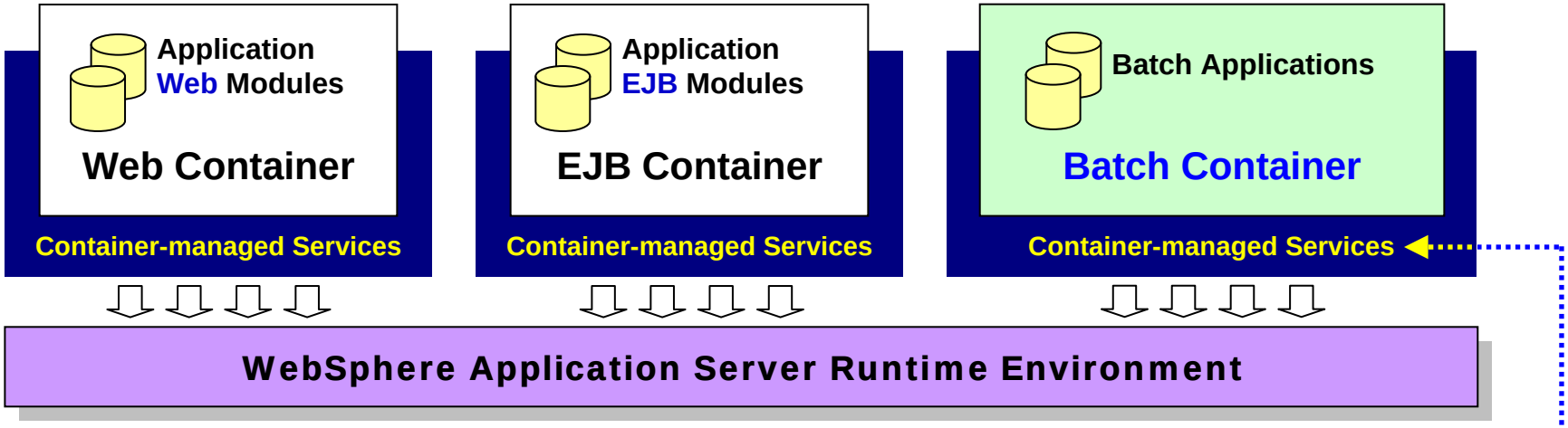


Function is identical between the two environments

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

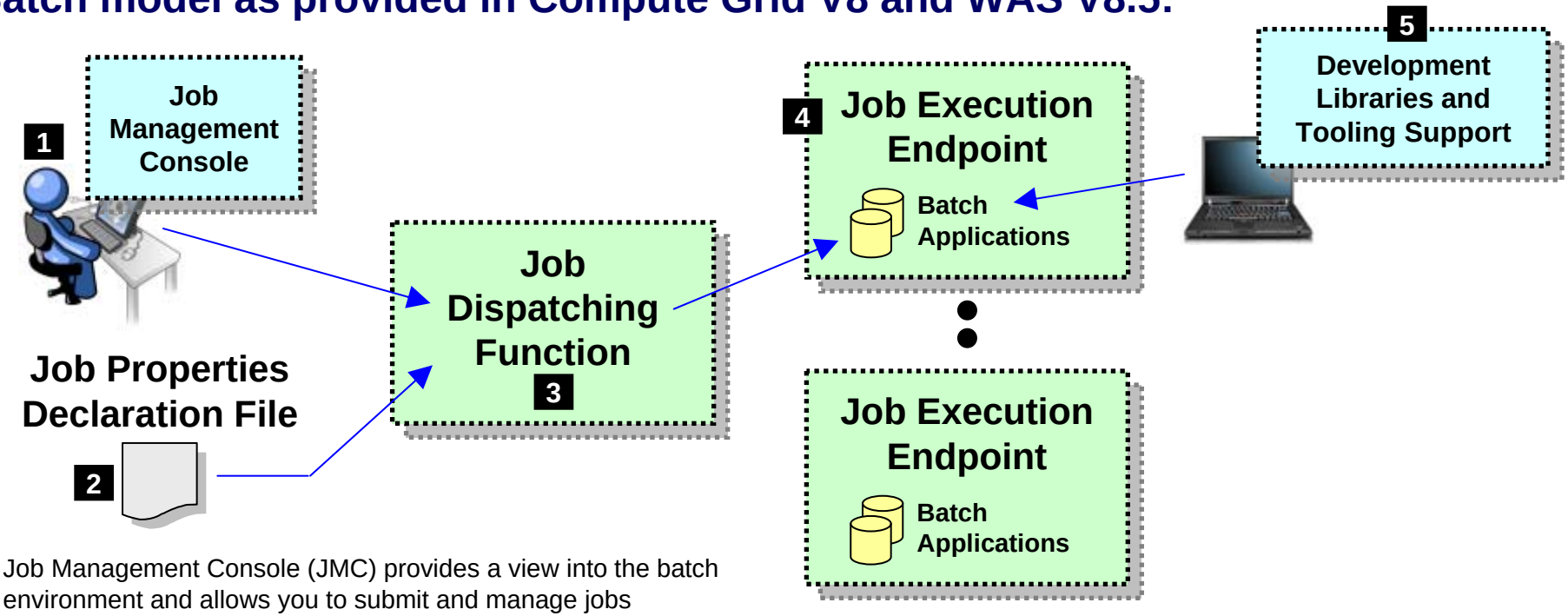
At a very high-level, you may think the IBM WebSphere Java Batch function as a "batch container" operating alongside the other containers of WAS itself:



Batch job dispatching and management system	Job resiliency services (skip record, step retry)
Data record read and write support services	Parallel job management and execution services
Checkpoint and job restart services	COBOL module call services

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

xJCL -- The Job Control Definition

Not JCL (no // and no column issues) ... but amazingly similar in concepts:

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

<job name="name" ... >
    <jndi-name>batch_controller_bean_jndi</jndi-name>
    <substitution-props>
        <prop name="property_name" value="value" />
    </substitution-props>

    <job-step name="name">
        <classname>package.class</classname>
            <checkpoint-algorithm-ref name="chkpt"/>
            <results-ref name="jobsum"/>
            <batch-data-streams>
                <bds>
                    <logical-name>input_stream</logical-name>
                        <props>
                            <prop name="name" value="value"/>
                        </props>
                </bds>
            </batch-data-streams>
    </job-step>

    <job-step

</job>
```

Roughly analogous to the JOB card

A job step

Like the EXEC PGM= statement in JCL

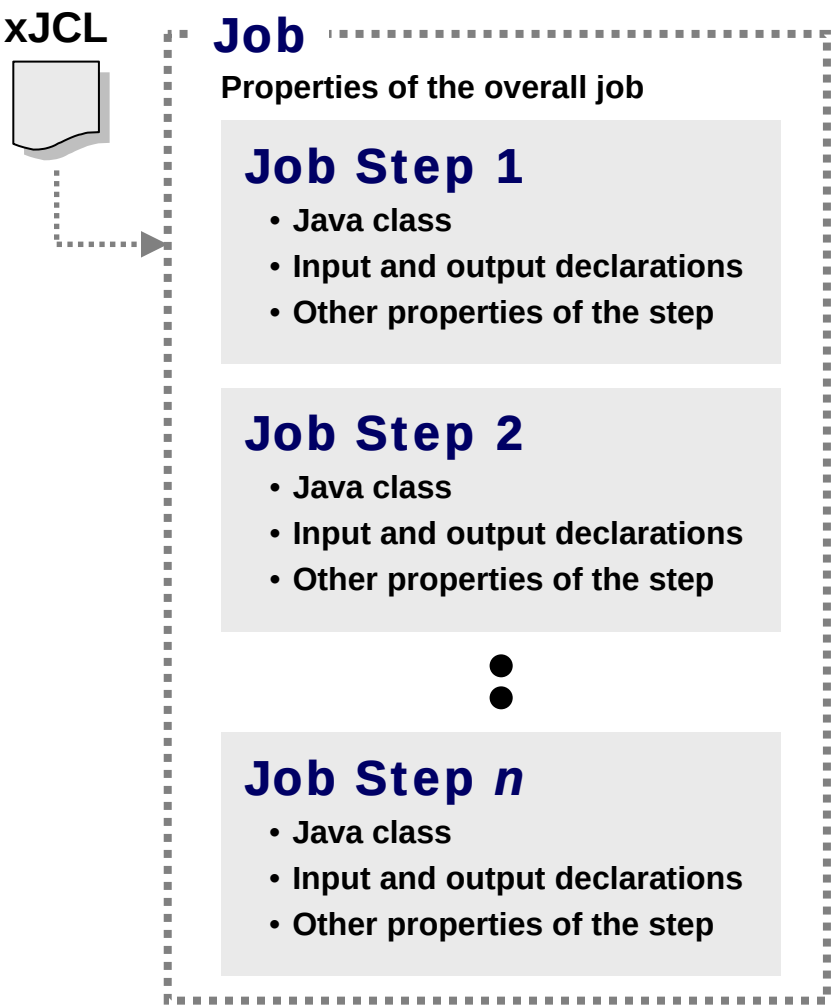
Similar to DD statements

A brief sampling ... many things not shown.

Do you see the similarity to traditional JCL?

Batch Job and Batch Job Steps

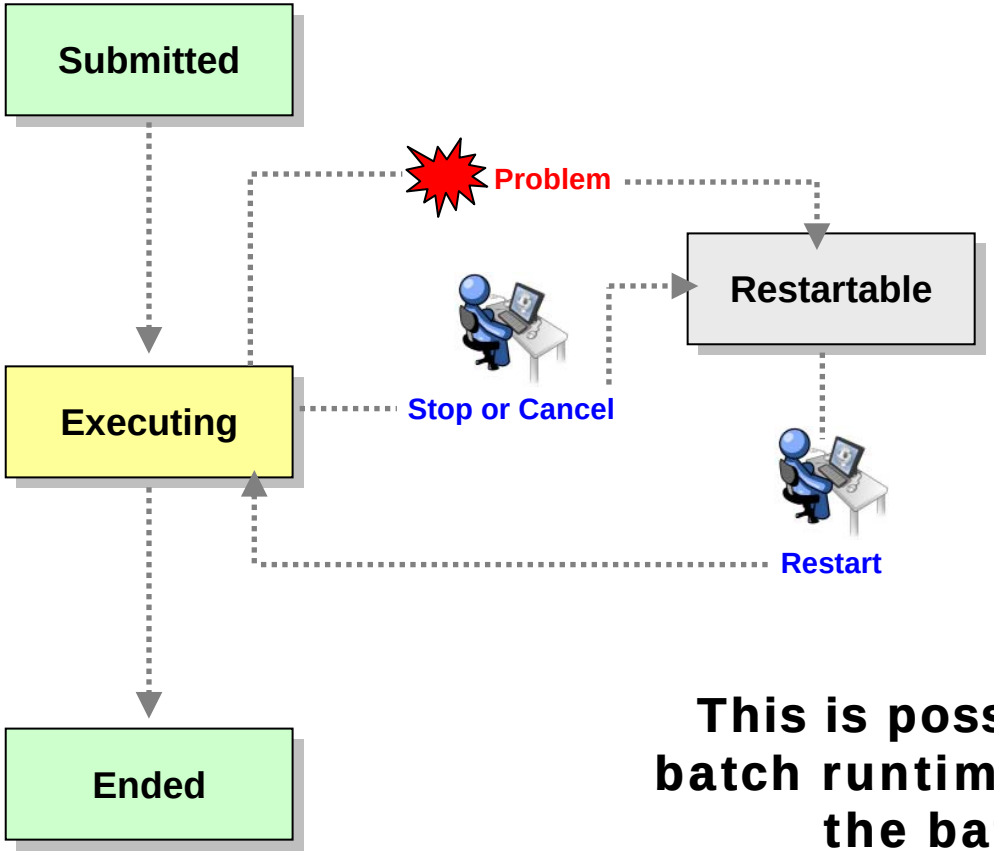
A batch job consists of one or more steps executed in order specified in xJCL:



- 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
- The Dispatching Function invokes the 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
- The Dispatching Function maintains awareness of the job state
- When the job ends, the job output file is accessible through Job Management Console

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.*



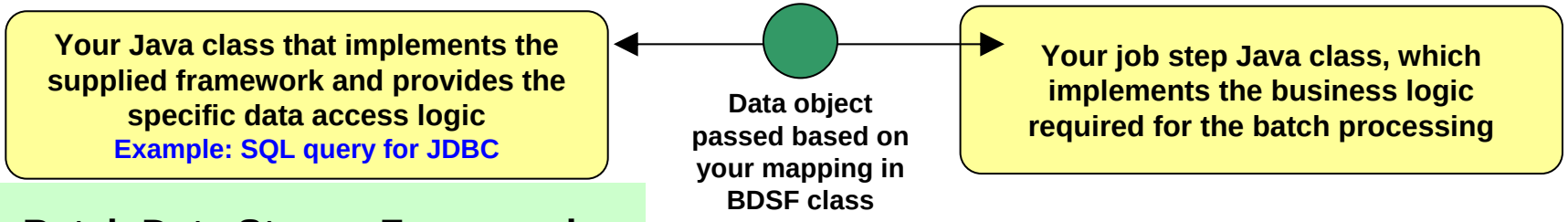
- The Job Management Console provides you ability to act upon an executing job
- The Batch Container is maintaining checkpoint status and will restart at the last checkpoint interval

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

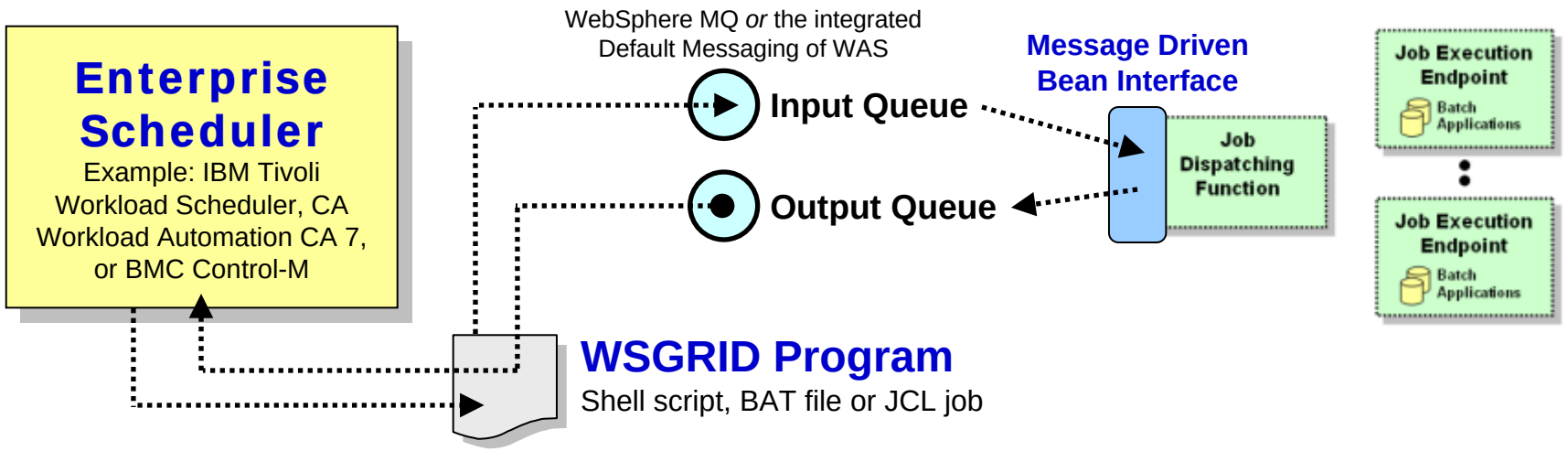
Supplied "patterns" for data access:

- **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:



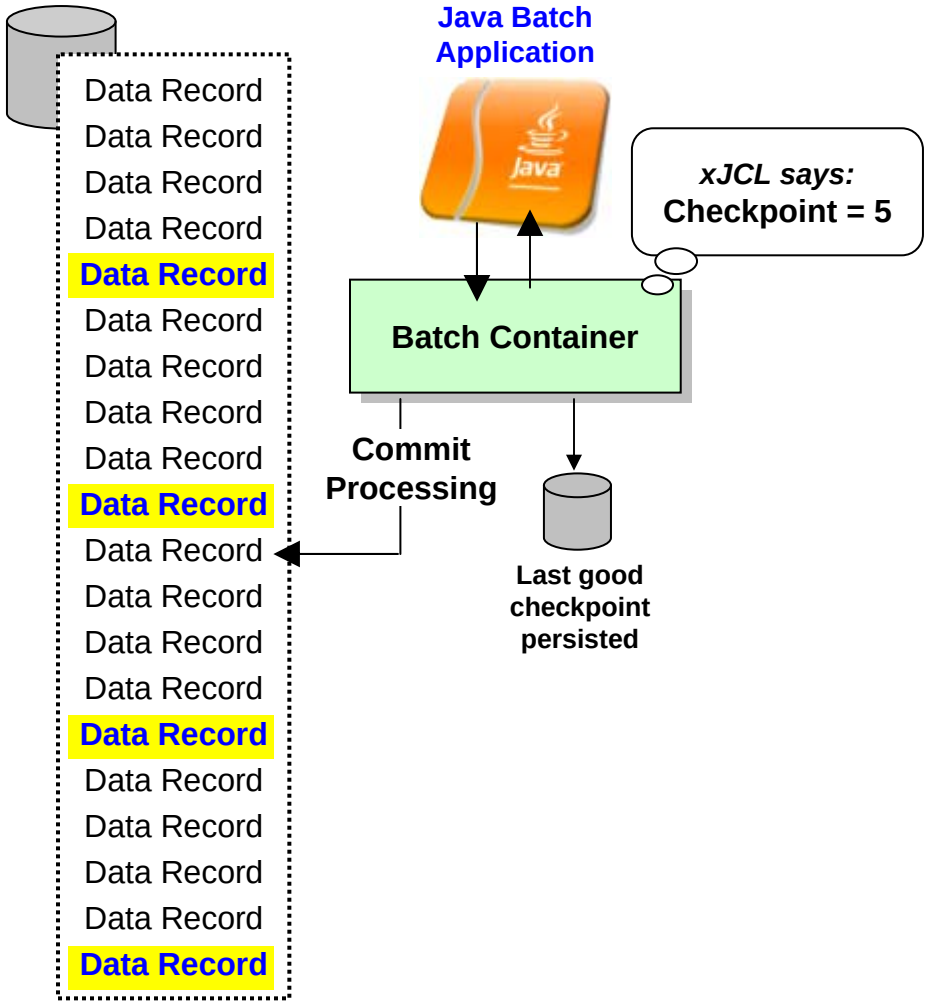
- WSGRID is seen by Scheduler as any other batch job it starts and monitors
- WSGRID interacts with Job Dispatching, submitting the job and processing Java batch job output back to STDOUT or JES Spool if z/OS
- WSGRID program stays up for life of job in WebSphere Java Batch
- To the Scheduler, WGRID *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

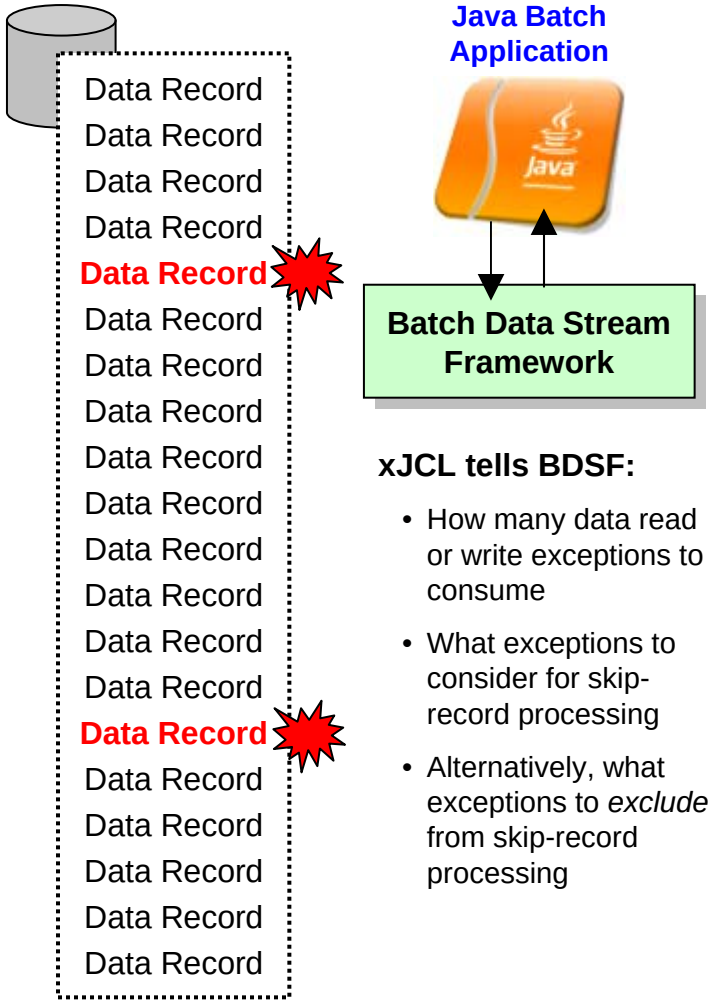
The batch container provides the ability to checkpoint at intervals based on either record count or time. The container keeps track of last checkpoint.



- 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.



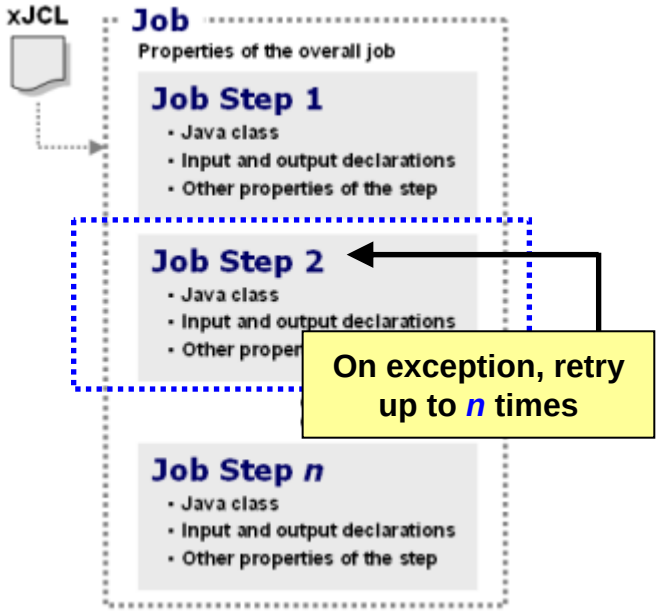
xJCL tells BDSF:

- How many data read or write exceptions to consume
- What exceptions to consider for skip-record processing
- Alternatively, what exceptions to *exclude* from skip-record processing

- 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 records
- 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.



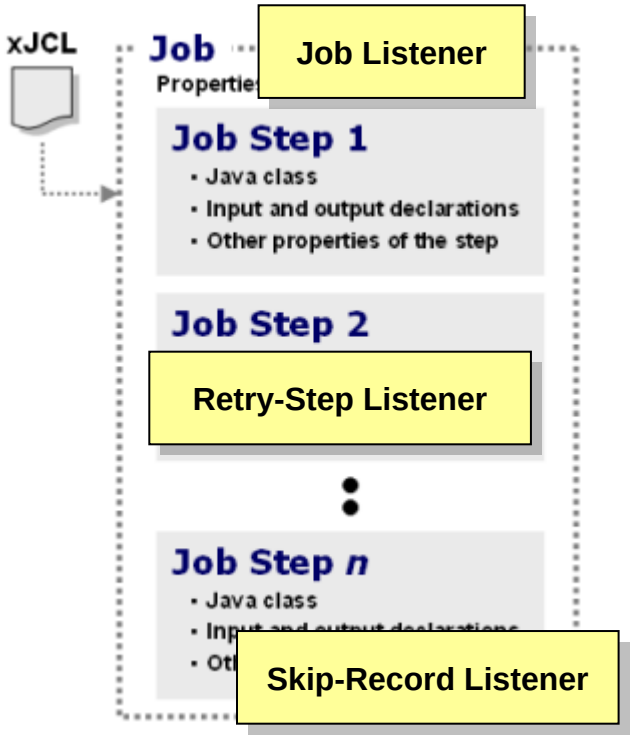
xJCL tells Container:

- 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 a 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
- 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

Batch "Listeners"

These are callout points where your customer "listener" code will be called when key events occur. The callouts are managed by the batch container:



Job Listener

- **Callouts occur:**
  Start of the job; Start of each step; End of each step; End of job
- **Register your code to container with property in xJCL**
- **Use this to perform any special setup or cleanup actions at those points in the lifecycle of a batch job**

Retry-Step Listener

- **Callouts occur:**
  When the exception is thrown; When the retry is attempted
- **Register your listener with code in application `createJobStep()` method**
- **Use this to take action at these points, such as logging information about the exception and the point in the processing where it occurred**

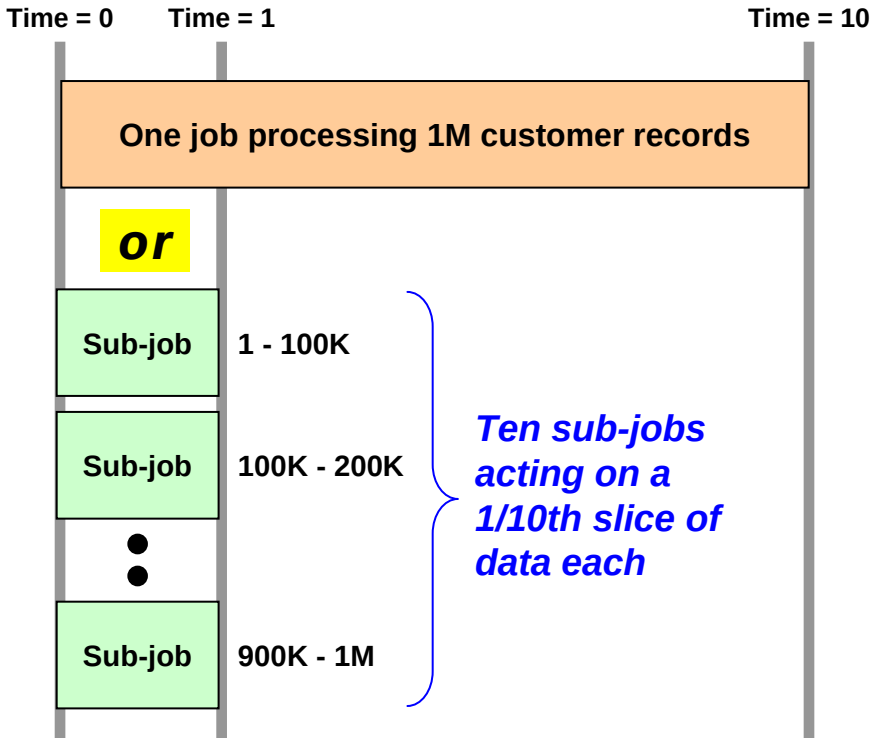
Skip-Record Listener

- **Callouts occur:**
  On skipped read or skipped write operation
- **Register your listener with code in application `createJobStep()` method**
- **Use this to take action at these points, such as logging information about the exception and the record skipped**

Listeners provide ability to have your code called at key points during batch job execution

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

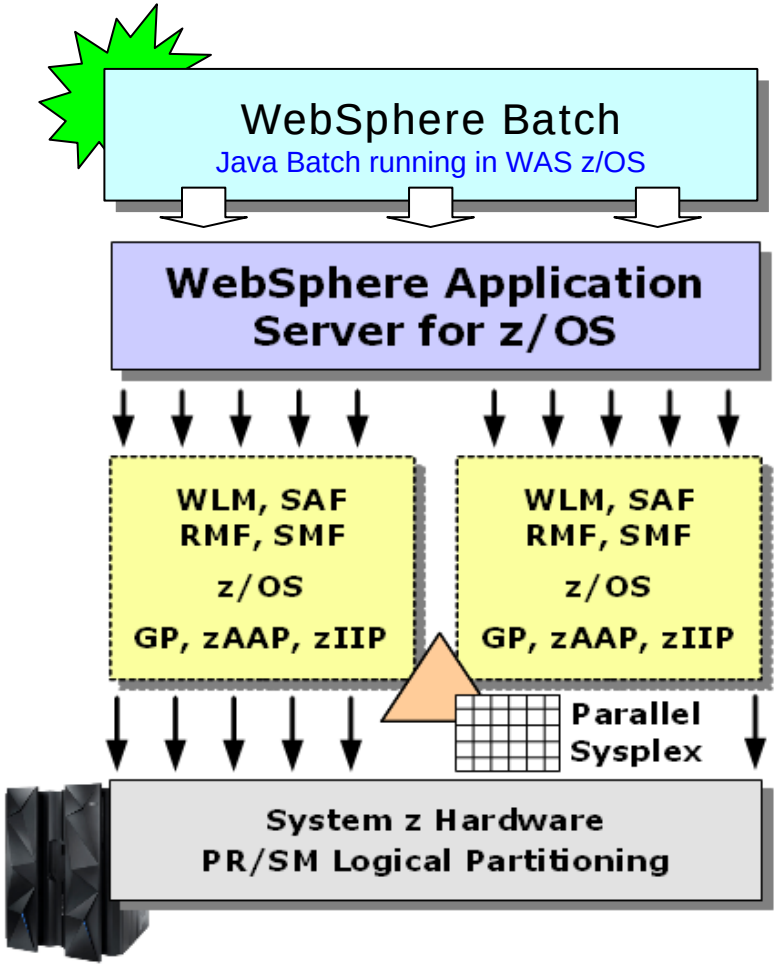
Objective is reduction in overall job completion time

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

Java Batch on z/OS

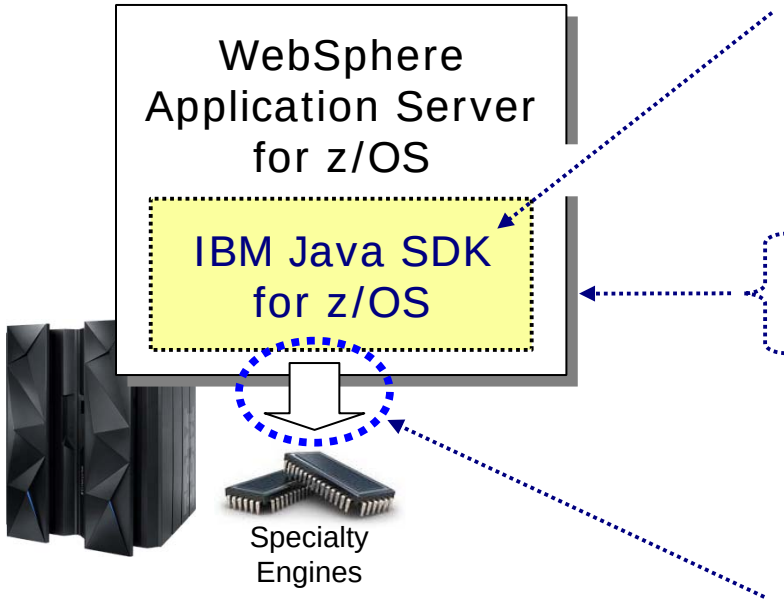
A review of what IBM WebSphere Java Batch brings specific to z/OS

The Value of WebSphere Batch on z/OS



- All the WAS z/OS value statements apply to WebSphere batch on z/OS as well, such as:
  - Reliability and availability
  - Shared data in Parallel Sysplex
  - Mapping parallel jobs onto clustered WAS servers in Sysplex with shared data infrastructure
  - Batch job classification with WLM
  - SMF recording of batch activity
  - Integration with enterprise scheduler functions with MQ BINDINGS
  - Local communications when co-located with applications and deata

IBM Java SDK for z/OS included with WAS for z/OS



WAS z/OS is a Java application runtime environment

WAS z/OS supplies the Java SDK

- The Java SDK binaries are packaged with the WAS z/OS binaries, and maintenance of Java is part of WAS z/OS maintenance

IBM coordinates Java SDK maintenance prerequisites for each fixpack of WAS z/OS

- 31-bit or 64-bit configurable by server
- With WAS V8.5 Java 6 or Java 7 is included and configurable by server

Default is Java 6 -- maintain same Java spec as previous versions of WAS if desired

Java 7 is latest and includes specific exploitation of instructions in new IBM System z EC12 machine

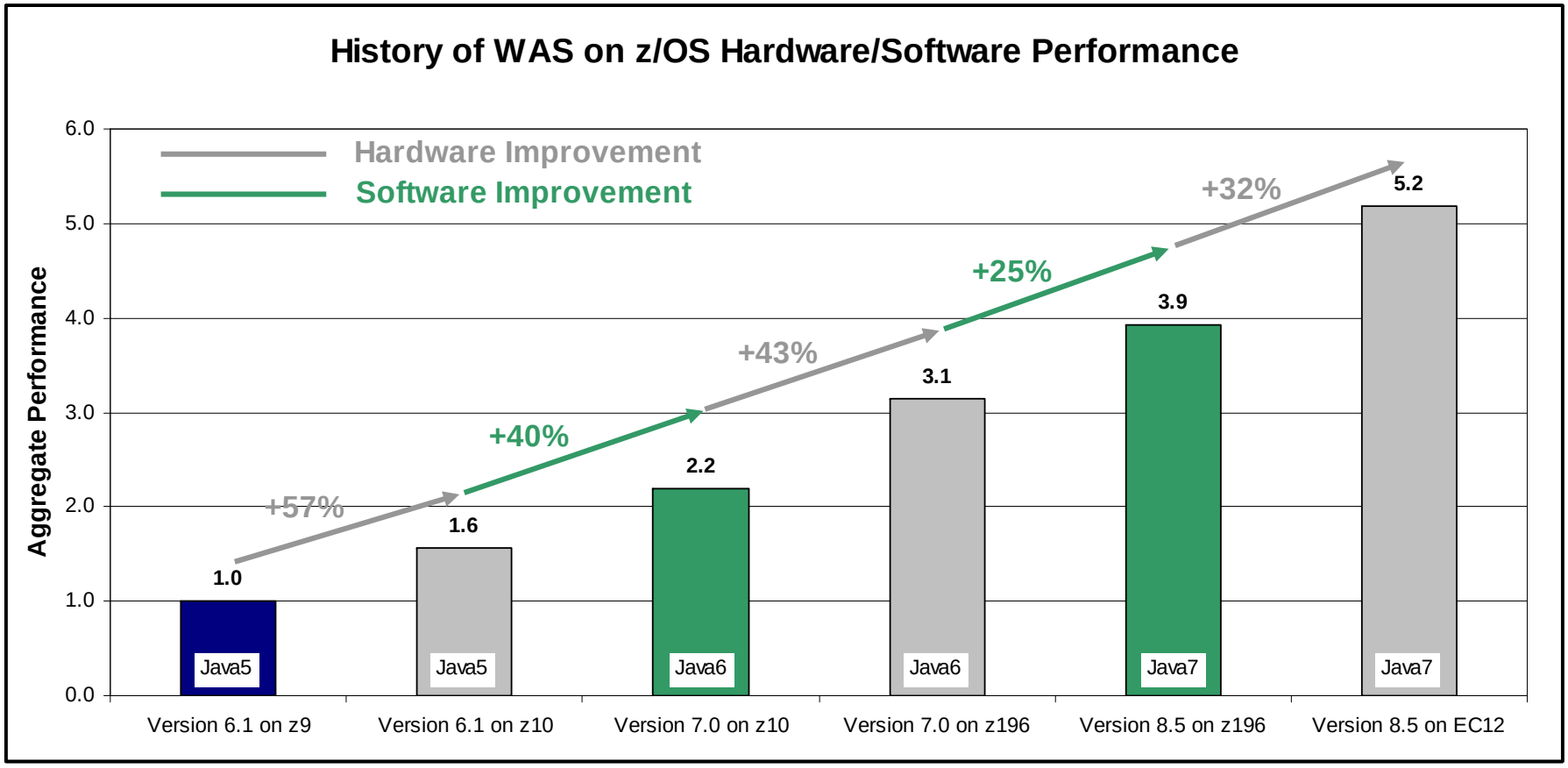
- Offload to specialty engines falls between 50% and 80% of WAS z/OS CPU

Offload *completely transparent* to the Java app in WAS

The degree of offload is dependent on many factors. An application that performance extensive Java work (XML parsing, for example) will offload relatively more. Applications that make greater use of WAS z/OS native platform services relatively less.

A Steady History of WAS on z/OS Performance Enhancements

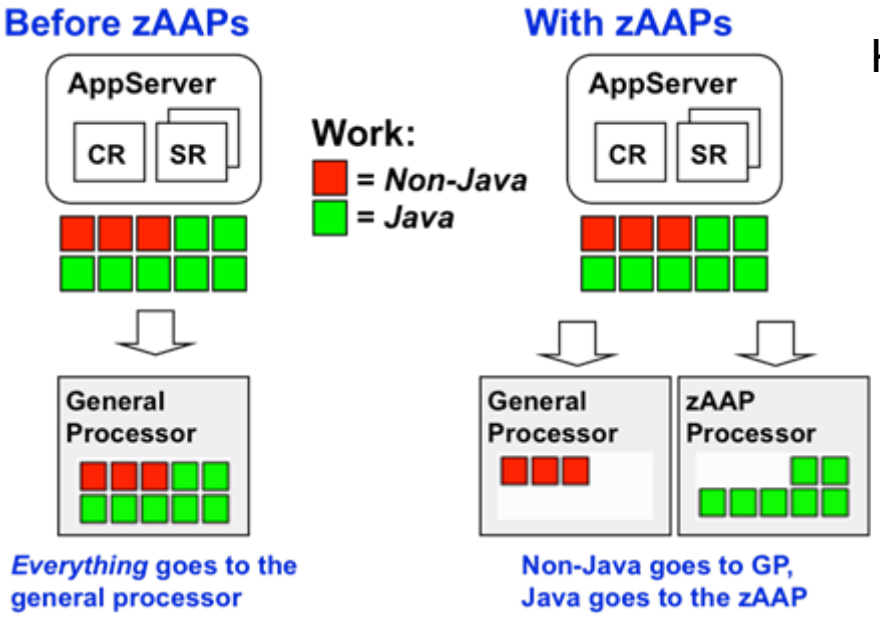
Aggregate HW, SDK and WAS Improvement: WAS 6.1 (Java 5) on z9 to WAS 8.5 (Java 7) on zEC12



~5x aggregate hardware and software improvement comparing WAS 6.1 Java5 on z9 to WAS 8.5 Java7 on zEC12

Java Offload to zAAP Specialty Engines

zAAP engines are Java offload engines. They enhance the financial picture of the z/OS platform, and they free up GP for other key subsystem processing



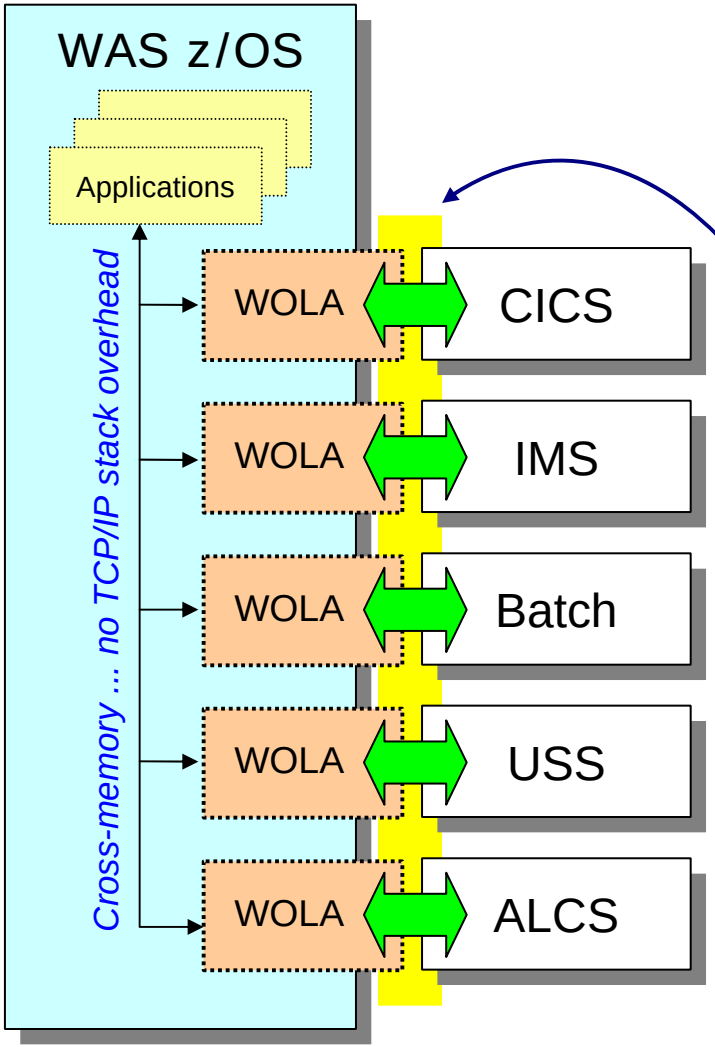
Keys to understanding value of zAAPs:

- zAAP processors have a considerably lower acquisition cost compared to GPs
- Offloading Java to zAAP frequently allows growing non-Java work to live within existing GPs, thus avoiding capital acquisition
- Monthly license charges based on capacity of the system can be influenced by the presence of zAAPs, which do not count towards charges

This is really a function of the Java SDK and the dispatcher of z/OS. The zAAP-enabled Java SDK is packaged with WAS z/OS, so WAS automatically takes advantage of zAAPs if they're present and configured

WebSphere Optimized Local Adapters (WOLA)

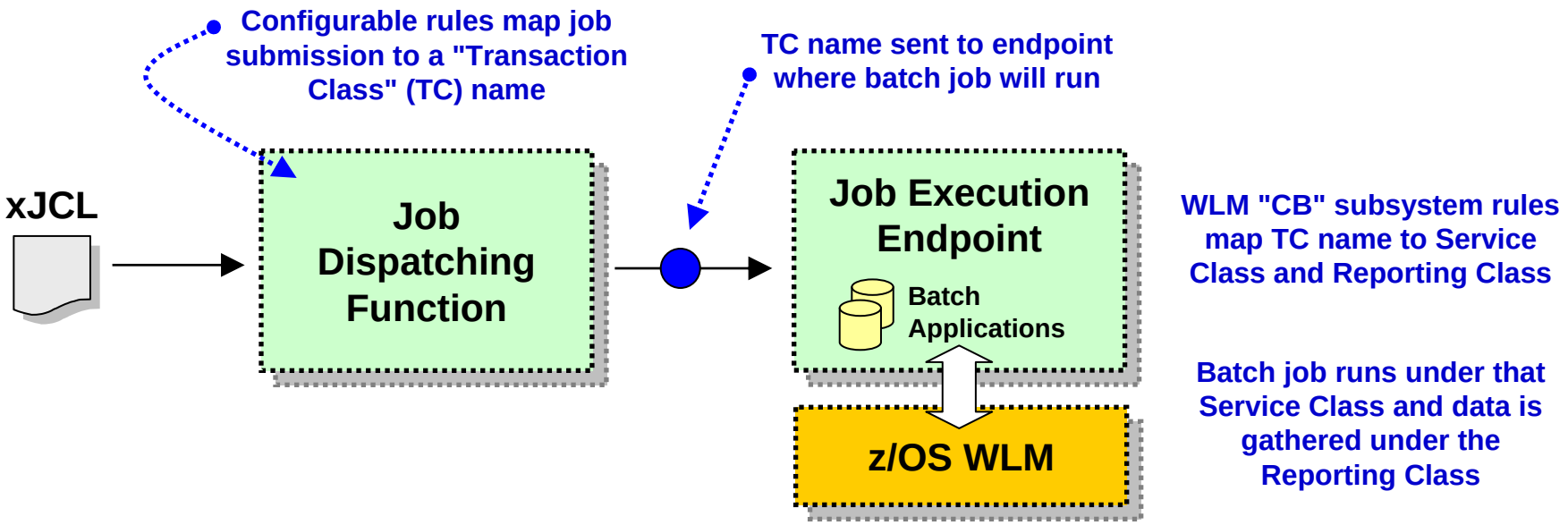
WebSphere Optimized Local Adapters provides an efficient low-latency mechanism to exchange data bi-directionally between WAS z/OS and other address spaces:



- Leverage WOLA to call out from batch programs to co-located applications
- Very efficient byte-array transfer
- First available with WAS z/OS 7.0.0.4
- Bi-directional
  - Outbound -- Java in WAS invokes program in external
  - Inbound -- Program in external invokes Java in WAS
- Two phase commit, identity assertion
- Supplied JCA resource adapter for applications going outbound
- Supplied native APIs for cases where their usage is indicated
  - COBOL, C/C++, PL/I, High Level Assembler
  - 31-bit and 64-bit modules

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

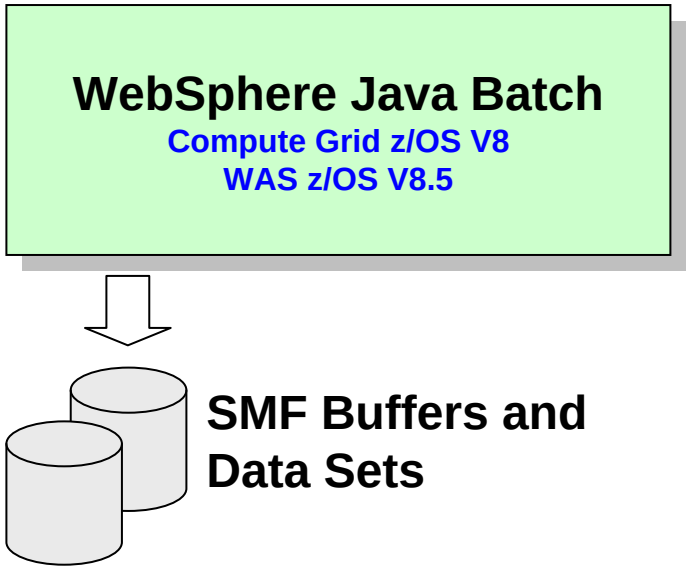
A somewhat sophisticated practice not widely used

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

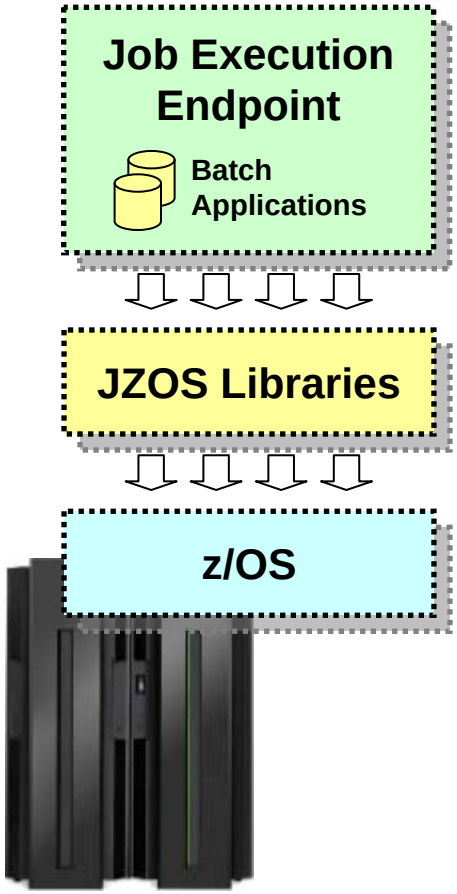
WAS z/OS supports the use of activity recording using the SMF 120.9 record. WebSphere Java Batch extends the record with batch activity information:



- 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: Total - GP = 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:



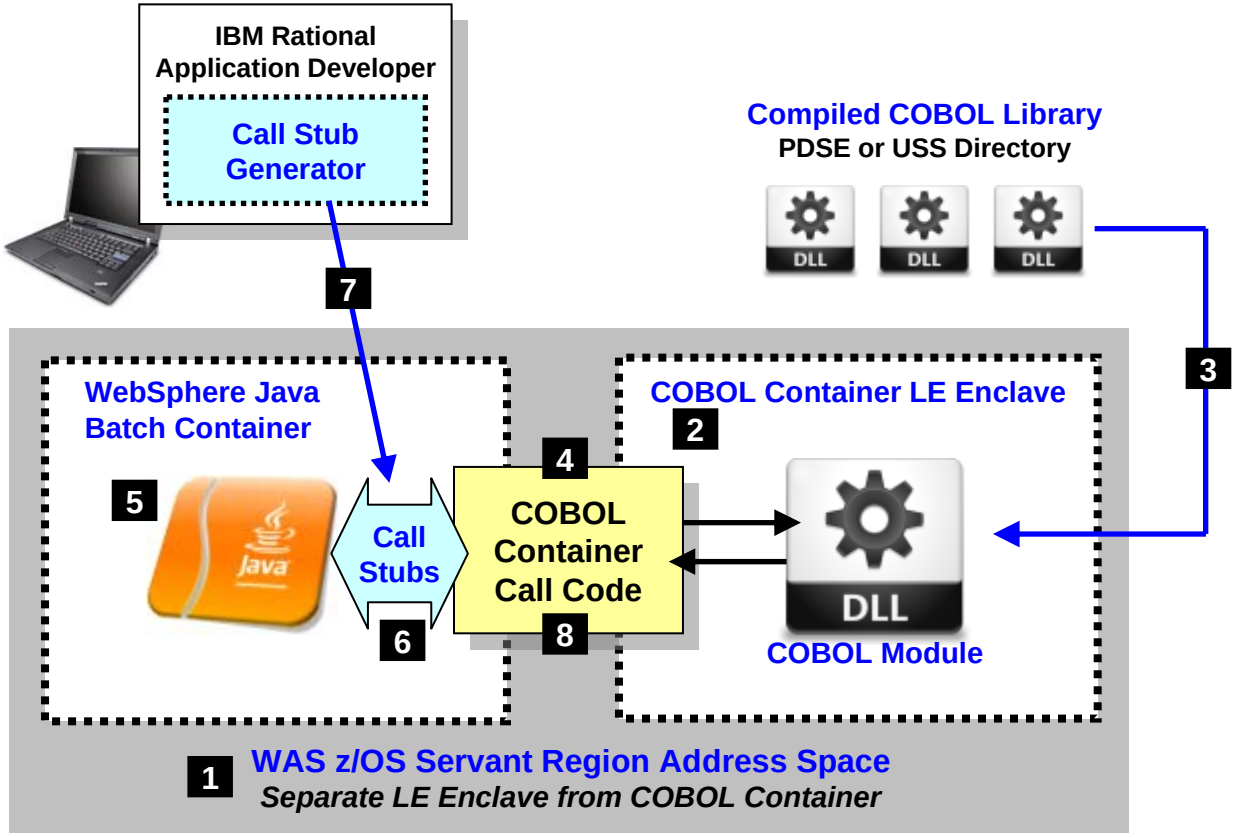
Examples of some z/OS services available:

- **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

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

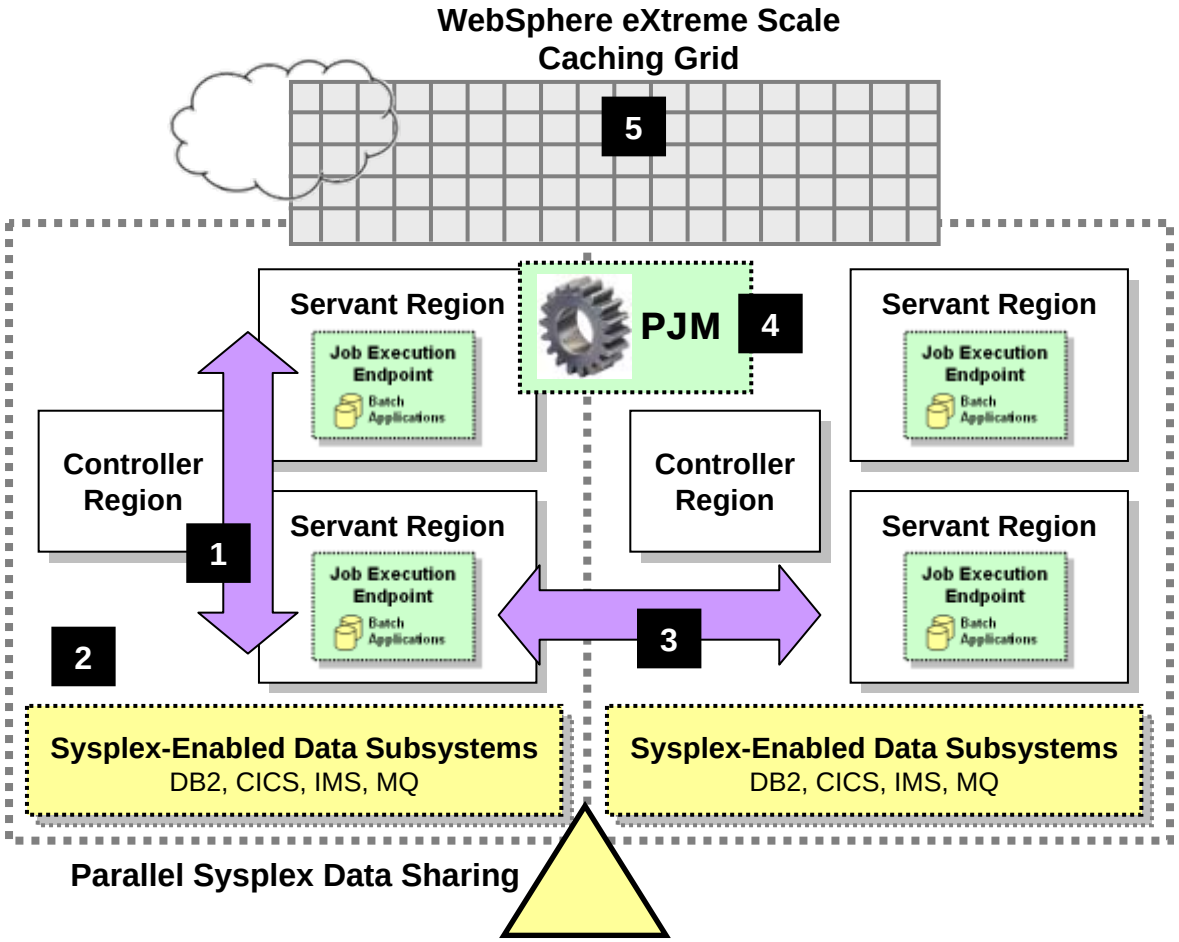


1. Batch application runs in the WAS z/OS servant region address space
2. The COBOL container is created as a separate LE enclave in the address space
3. COBOL DLLs are accessed using STEPLIB or LIBPATH
4. COBOL Container code provides the "glue" between the Java environment and the native COBOL
5. Java 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

Scaling Up the Java Batch Solution on z/OS

There are several ways in which a WebSphere Java batch solution can be scaled up to provider greater batch throughput and shorter execution windows:



1. Vertical

WAS z/OS servant regions provide a type of "vertical cluster," giving you additional batch compute resources

2. Capacity on Demand

CPU processors may be dynamically added to a z/OS LPAR, increasing the capacity for processing work

3. Horizontal

WAS z/OS clustering on top Parallel Sysplex provides near-linear scalability up to 32 nodes with a central data sharing model

4. Parallel Processing

The Parallel Job Manager may be used to partition data into sub-jobs, which may then be run on multiple threads, different servants, or different servers on other LPARs.

5. Data Caching

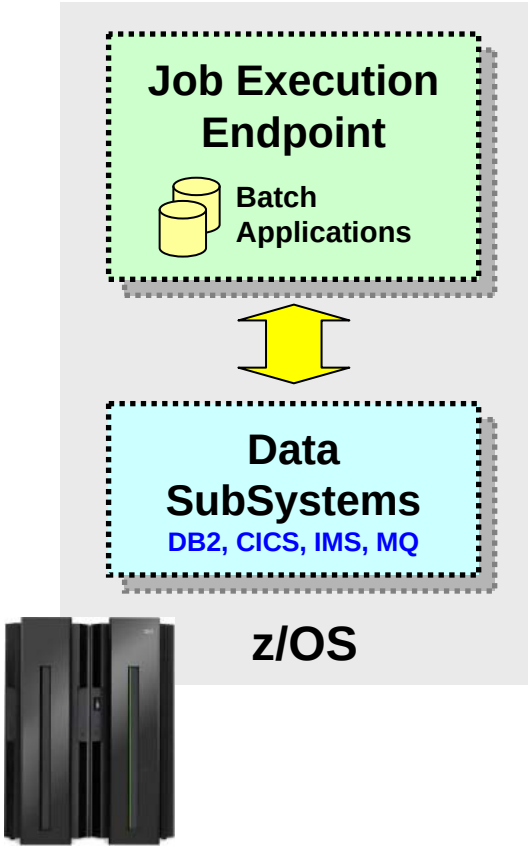
WebSphere eXtreme Scale provides a data caching grid from which Java batch may fetch and store data

Deployment Scenarios

A review of some potential ways to deploy the WebSphere Java Batch function

Co-Location on z/OS

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

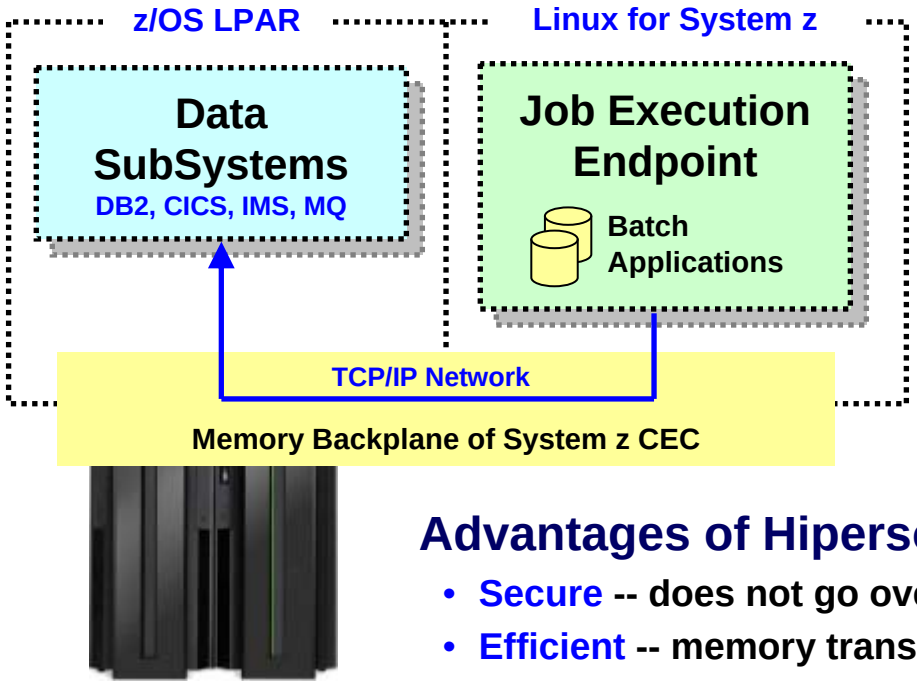


- Use of **cross-memory connectors** for high-speed and low-latency access to data
  - **JDBC Type 2 connector for access to DB2**
  - **CICS Transaction Gateway (CTG) local EXCI**
  - **WebSphere Optimized Local Adapters (WOLA)**
- Much more **secure** -- cross memory data exchanges can *not* be 'sniffed' 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

Linux for System z and Hipersocket Access to z/OS Data

Hipersockets is a technology that maps a TCP/IP network onto the memory backplane of a System z divided into multiple logical partitions (LPAR):



To programs and processes that use Hipersockets it looks like a routed TCP/IP network

Advantages of Hipersockets

- **Secure** -- does not go over adapters or external wires
- **Efficient** -- memory transfer speeds implies lower overall latency

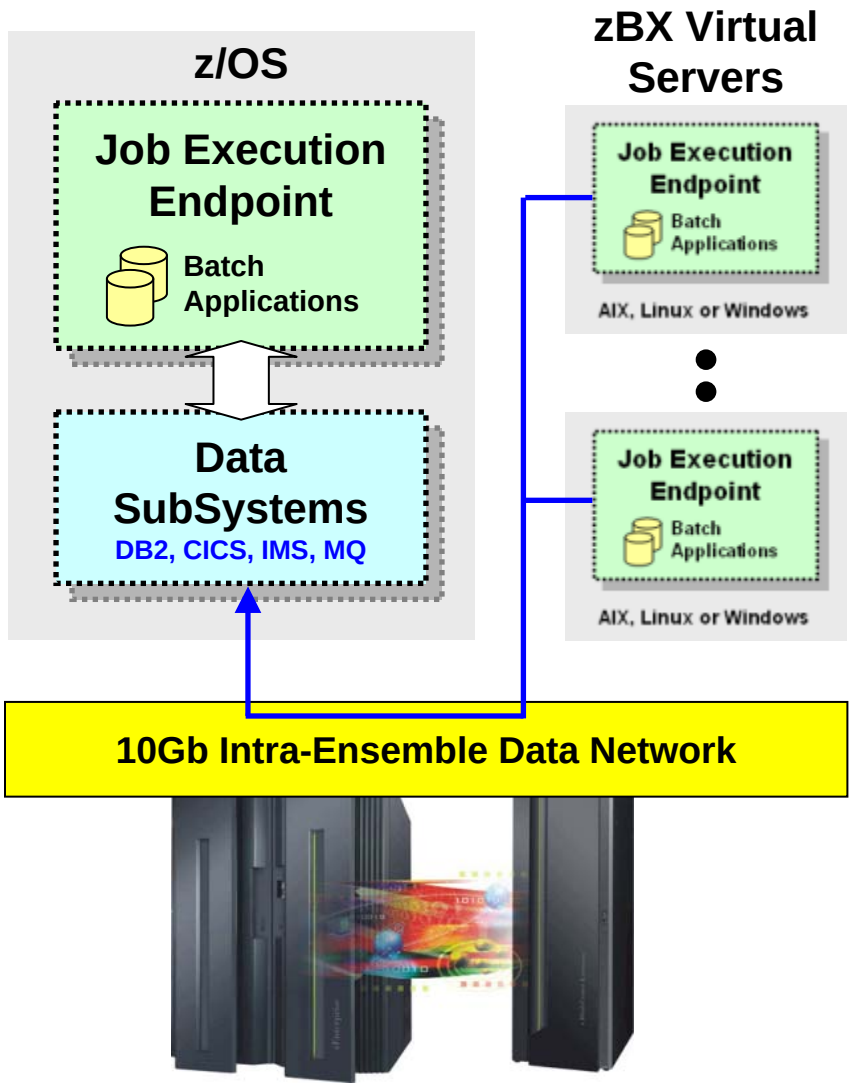
Advantages of Linux for System z

- **Consolidation** -- host many Linux images in a virtualized environment

Virtualizing on the zVM hipervisor provides a means of quickly scaling up in Linux instances to meet requirements

zEnterprise and zBX

The zEnterprise system is designed around principle of right-fit placement:



- System z LPAR serves as the anchor for a zEnterprise "node"
- A zBX blade extension rack hosts IBM p or IBM x blades capable of hosting AIX, Linux or Windows virtual servers
- A 10Gb network connects it all
- WebSphere Java Batch endpoints may be placed where the work they do makes best sense:
  - Batch processes requiring a highly available and highly secure environment may operate on z/OS
  - Batch processes that use relatively more CPU may be offloaded to zBX blade servers
  - WebSphere Java Batch Dispatching function would be able to "see" all the different endpoints and dispatch based on where batch applications were deployed

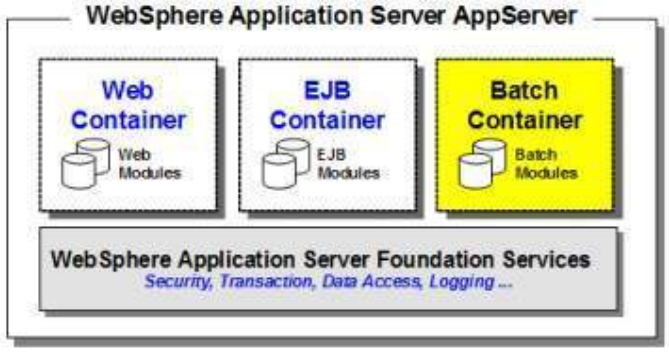
Wrap-Up and Summary

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
- Skip record processing
- COBOL support on z/OS



Compute Grid capabilities integrated into WAS 8.5

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.

WebSphere on z/OS and Java Batch – Links to Collateral

Topic	Link
Guide to WebSphere on z/OS Collateral - Updated master list of links to collateral	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102205
WebSphere Java Batch - Overview and z/OS Specifics - Presentation, whitepaper, videos	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101783
Why WAS for z/OS - Executive Brochure - Technical Presentation, videos	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101532
WAS for z/OS Liberty Profile - Executive Brochure - Quick Start Guide and Samples	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110
WebSphere Optimized Local Adapters (WOLA) - Overview, whitepapers, videos - History of WOLA updates	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490
Training – z/OS Wildfire Workshops - WAS for z/OS v8.5, WebSphere Compute Grid	http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS1778
WebSphere on z Virtual User Group - Interact with WebSphere on z users - Download webcasts (WAS for z/OS, Batch)	http://www.websphereusergroup.org/zos


Thank
You