Batch Processing with WebSphere

Technical Overview of WebSphere XD Compute Grid

Chris Vignola

cvignola@us.ibm.com

STSM, Compute Grid Lead Architect
WebSphere XD Development, SWG, IBM
http://chris.vignola.googlepages.com

Snehal S. Antani

antani@us.ibm.com

WebSphere XD Technical Lead
SOA Technology Practice, ISSW, SWG, IBM
http://snehalantani.googlepages.com

Agenda

- WebSphere XD Compute Grid product goals
- Compute Grid Infrastructure
  - Topology
  - Performance (Parallel Processing, Proximity to Data)
  - ‘ilities (Availability, Scalability, etc)
- Batch Applications
  - Batch Data-stream (BDS) Framework and other tooling
  - Design patterns for sharing services across batch and OLTP
  - Example application architecture
  - Performance optimizations (caching, parallelization, etc)
- Short-term objectives and strategy

Goal

To Deliver a ***Modern*** Batch Processing ***Platform*** for the ***Enterprise***

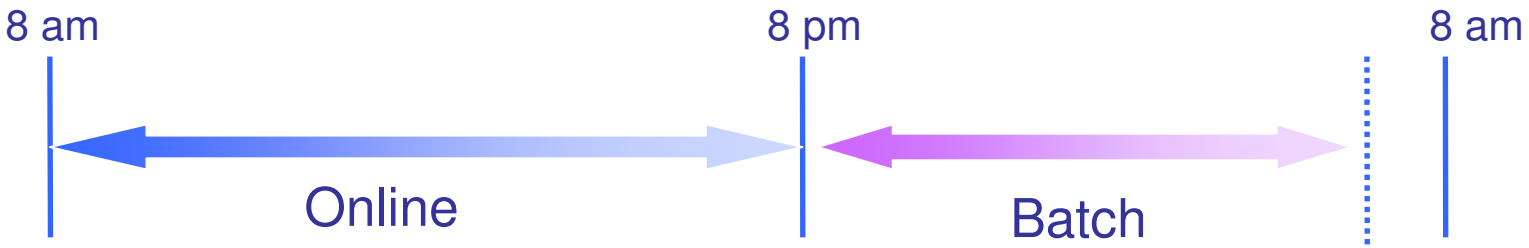
Modern: 24x7 Batch, Sharing business services across batch and OLTP, parallel-processing and caching, container-managed QoS, design patterns

Platform: runtime components (schedule, dispatch, govern), e-2-e Development Tooling, workload management integration, operational control with external scheduler integration

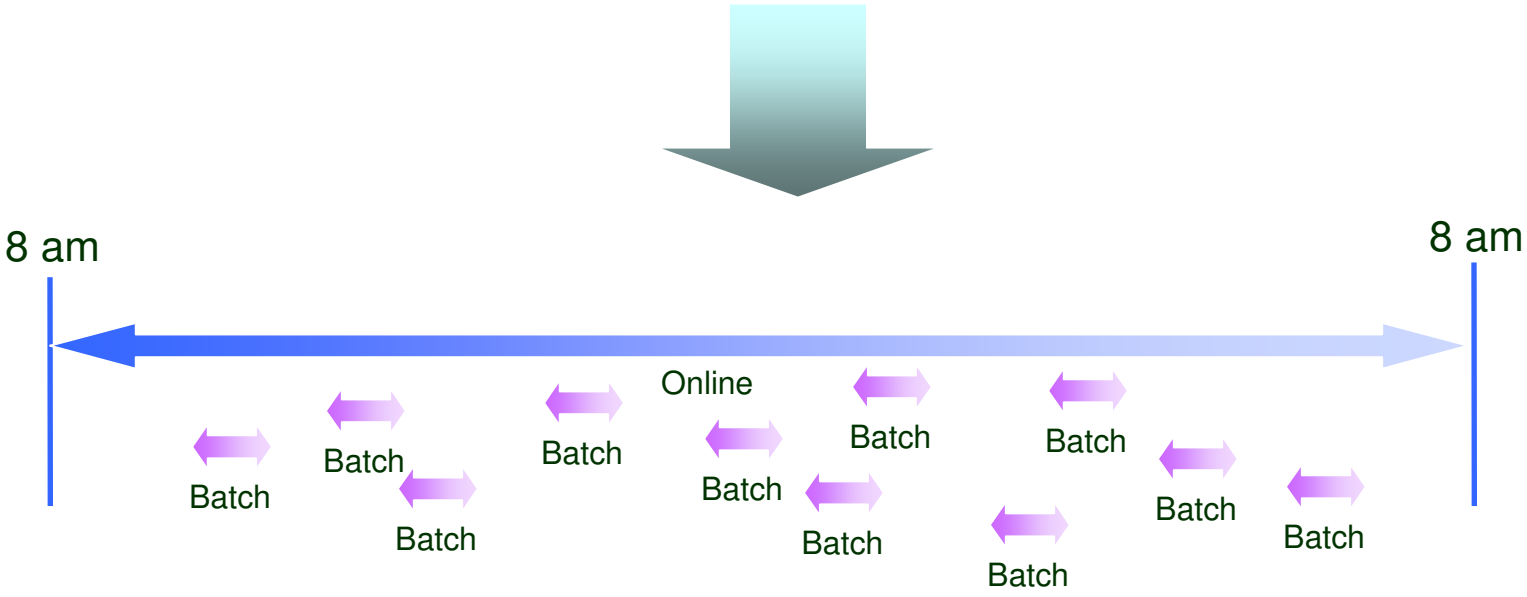
Enterprise: Platform-neutral applications, Standardized Application Architecture, Standardized Operational Procedures

Compute Grid Infrastructure

24x7 Batch and OLTP



Current Batch Processing Technique



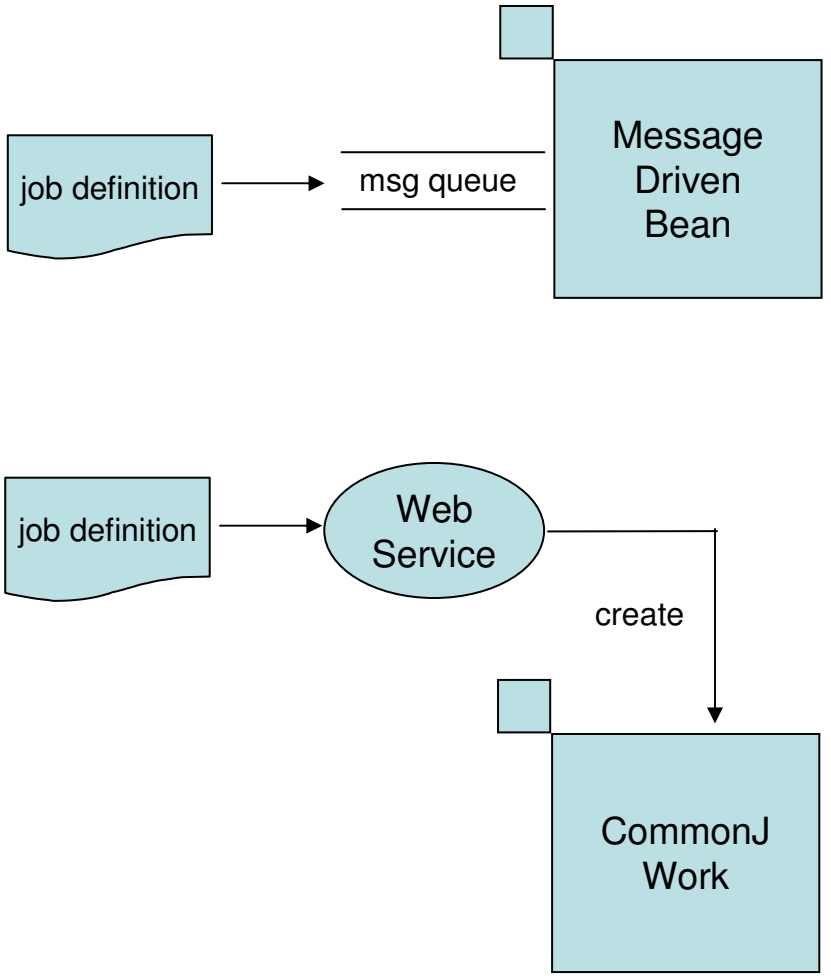
Modern Batch Processing Technique

The “Maverick” Batch Environment

- Roll Your Own (RYO)
- Seems easy – even tempting ☺
- Message-driven Beans or
- CommonJ Work Objects or …

But …

- No job definition language
- No batch programming model
- **No checkpoint/restart**
- No batch development tools
- **No operational commands**
- **No OLTP/batch interleave**
- No logging
- **No job usage accounting**
- **No monitoring**
- No job console
- No enterprise scheduler integration
- **No visibility to WLM**
- No Workload throttling/pacing/piping
- …

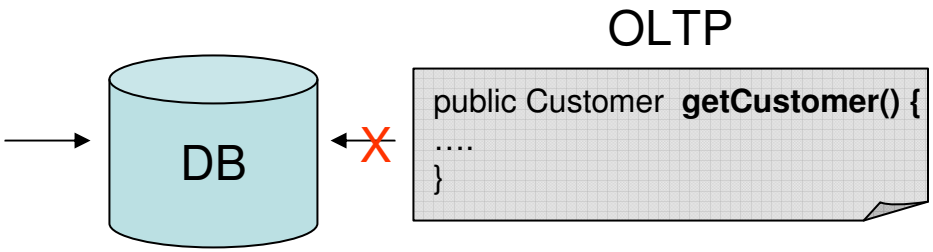


OLTP and Batch Interleave

```
public void doBatch() {
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    Cart cart = new Cart(...);
    customer.setCart(cart) // needs to be persisted as well
    session.save(customer);
    if ( i % 20 == 0 ) { //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
}
```

Source: some Hibernate Batch website

BATCH



-Batch application's hold on DB locks can adversely impact OLTP workloads

-OLTP Service Level Agreements can be breached

-How do you manage this?

-WLM will make the problem worse!

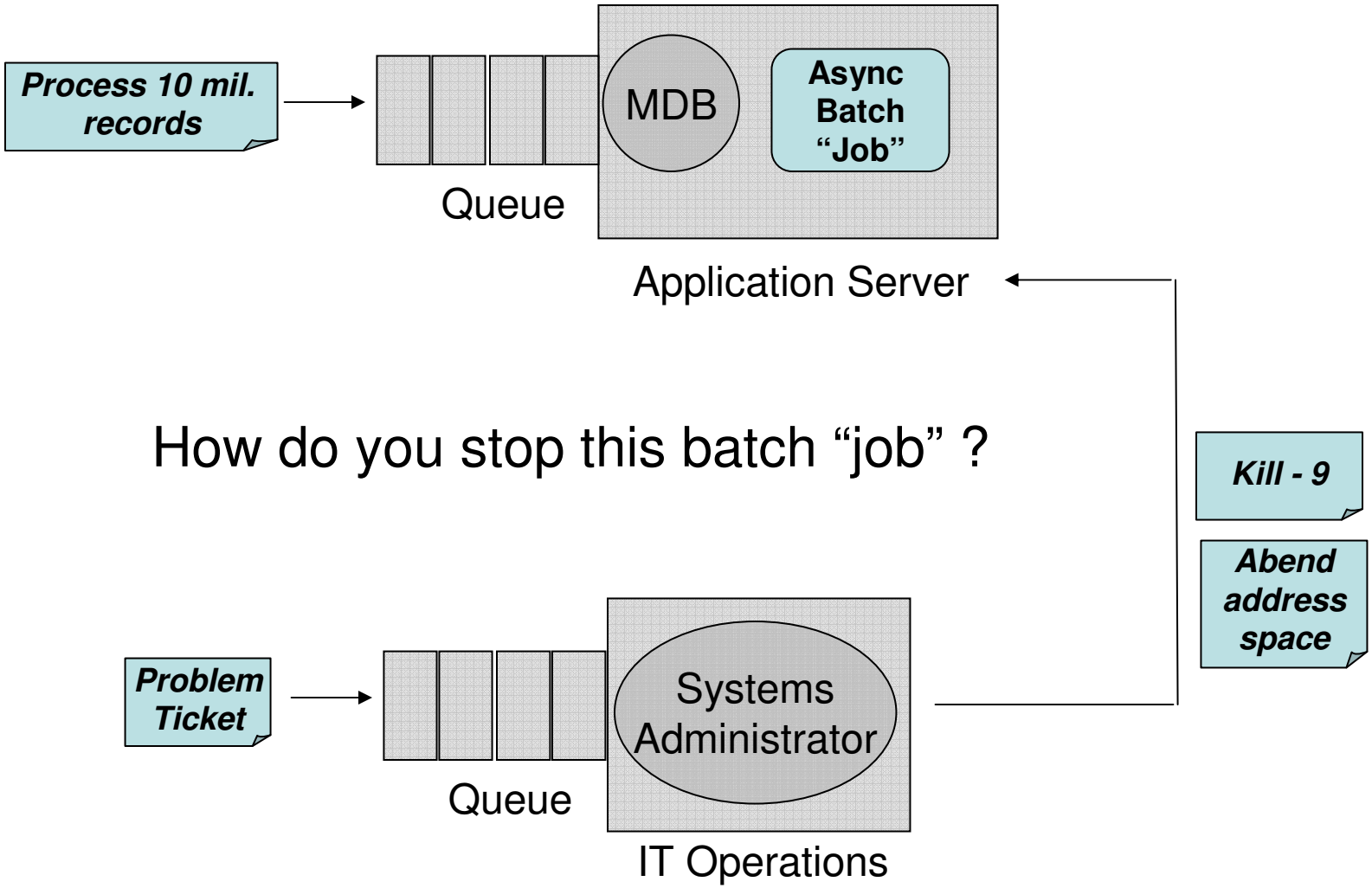
Solution: Container-Managed Services

- Container-Managed ***Checkpoint Strategies***
  - Keep track of the current input and output positions on behalf of the batch step
  - Store these values as part of the same global transaction as the business logic
  - 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

Operational Control in “Maverick” Batch



Solution: Integrated Operational Control

-Provide an operational infrastructure for starting/stopping/canceling/restarting/etc batch jobs.

- Integrate that operational infrastructure with existing enterprise schedulers such as Tivoli Workload Scheduler

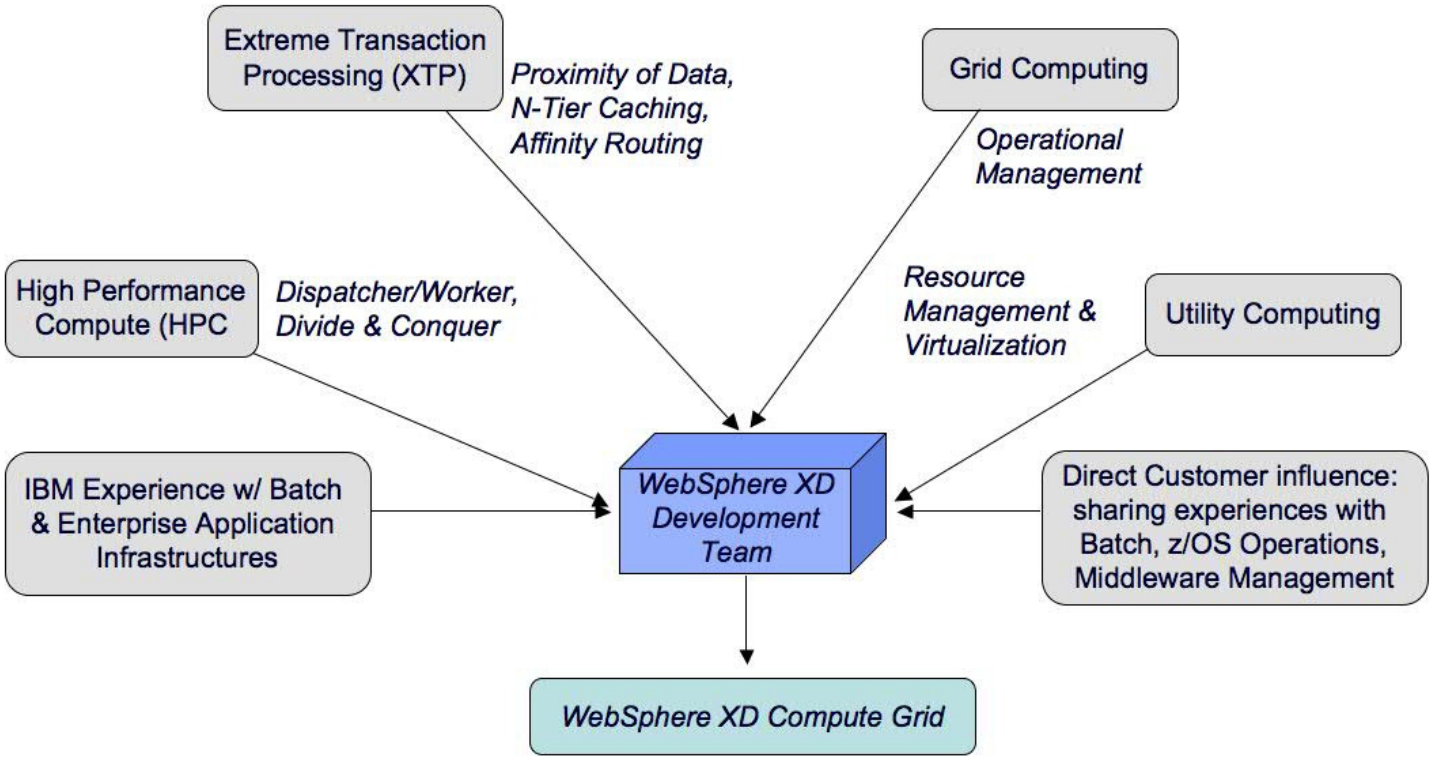
- Provide log management and integration with archiving and auditing systems

- Provide resource usage monitoring

- Integrate with existing security and disaster recovery procedures

Batch Processing with WebSphere XD Compute Grid

- Compute Grid design has been influenced by a number of domains
- Most important: Customer collaborations and partnerships
  - Continuous cycle of ***Discovery*** and ***Validation***
  - ***Discover*** new features by working directly with our clients
  - ***Validate*** ideas, features, and strategy directly with our clients



WebSphere XD Compute Grid summary

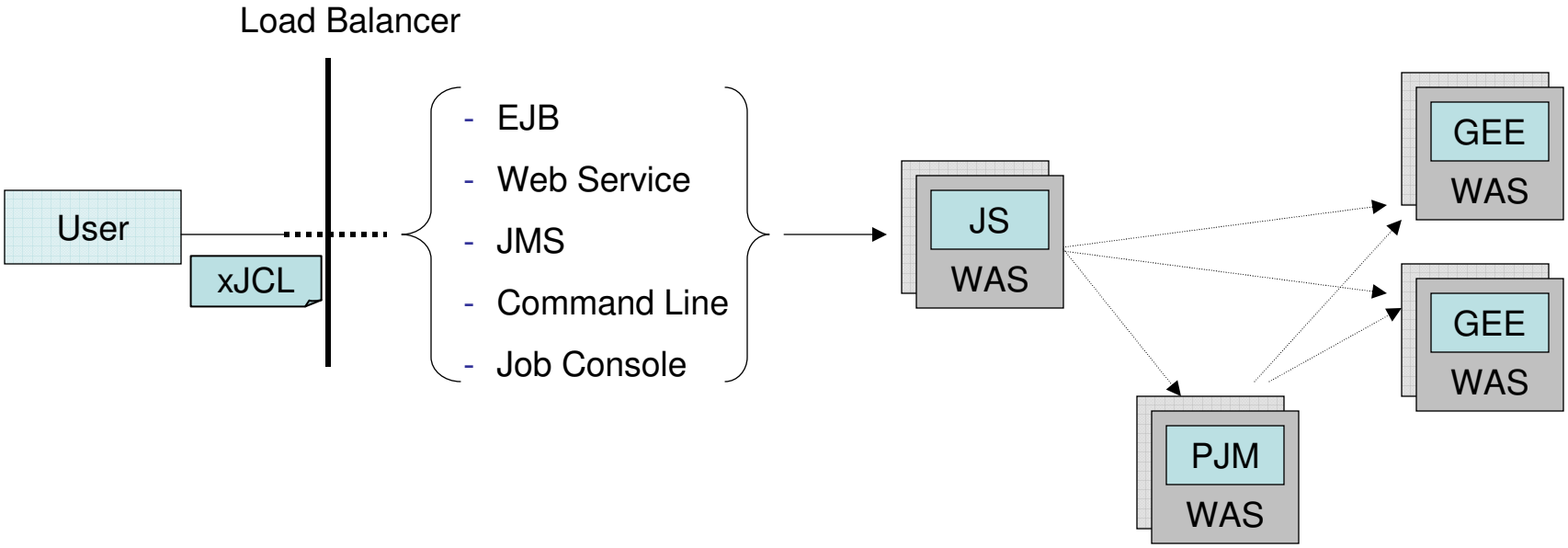
- Leverages J2EE Application Servers (WebSphere today… more tomorrow)
  - **Transactions**
  - **Security**
  - **high availability including dynamic servants**
  - **Leverages the inherent WAS QoS**
  - **Connection Pooling**
  - **Thread Pooling**

- Platform for executing transactional java batch applications
  - Checkpoint/Restart
  - Batch Data Stream Management
  - Parallel Job Execution
  - Operational Control
  - External Scheduler Integration
  - SMF Records for Batch
  - zWLM Integration

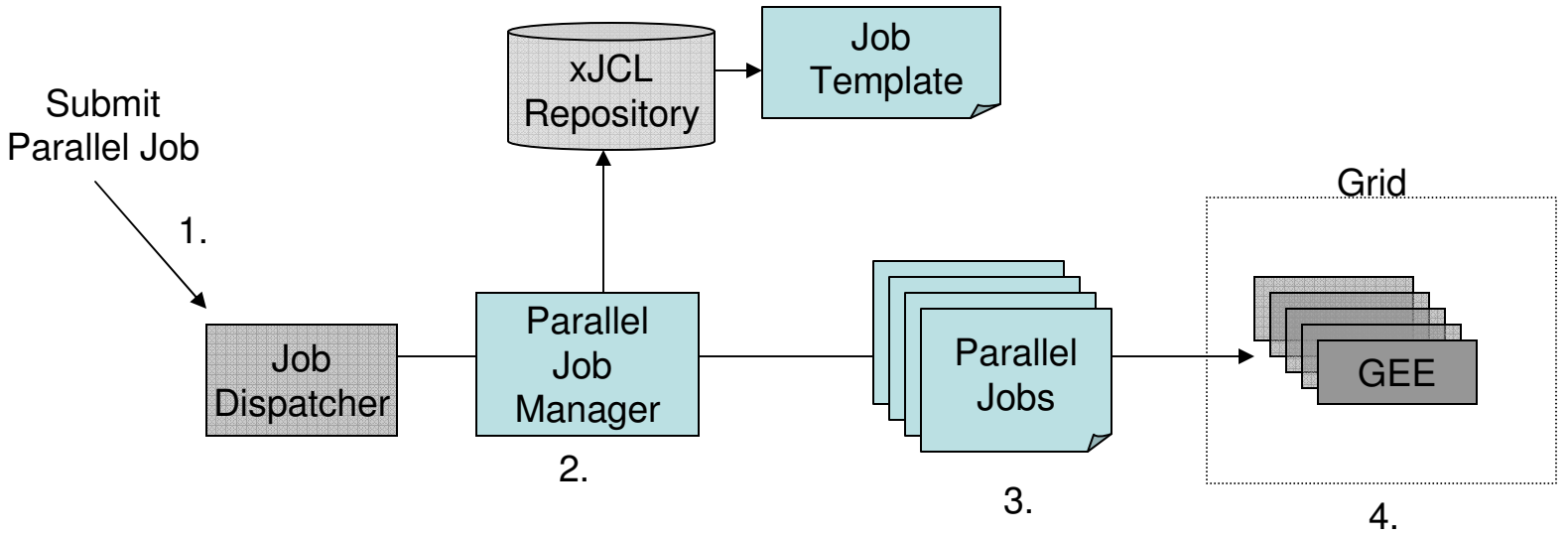
XD Compute Grid Components

- Job Scheduler (**JS**)
  - The job entry point to XD Compute grid
  - Job life-cycle management (Submit, Stop, Cancel, etc) and monitoring
  - Dispatches workload to either the PJM or GEE
  - Hosts the Job Management Console (JMC)
- Parallel Job Manager (**PJM**)-
  - 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 (**GEE**)
  - Executes the actual business logic of the batch job

XD Compute Grid Components

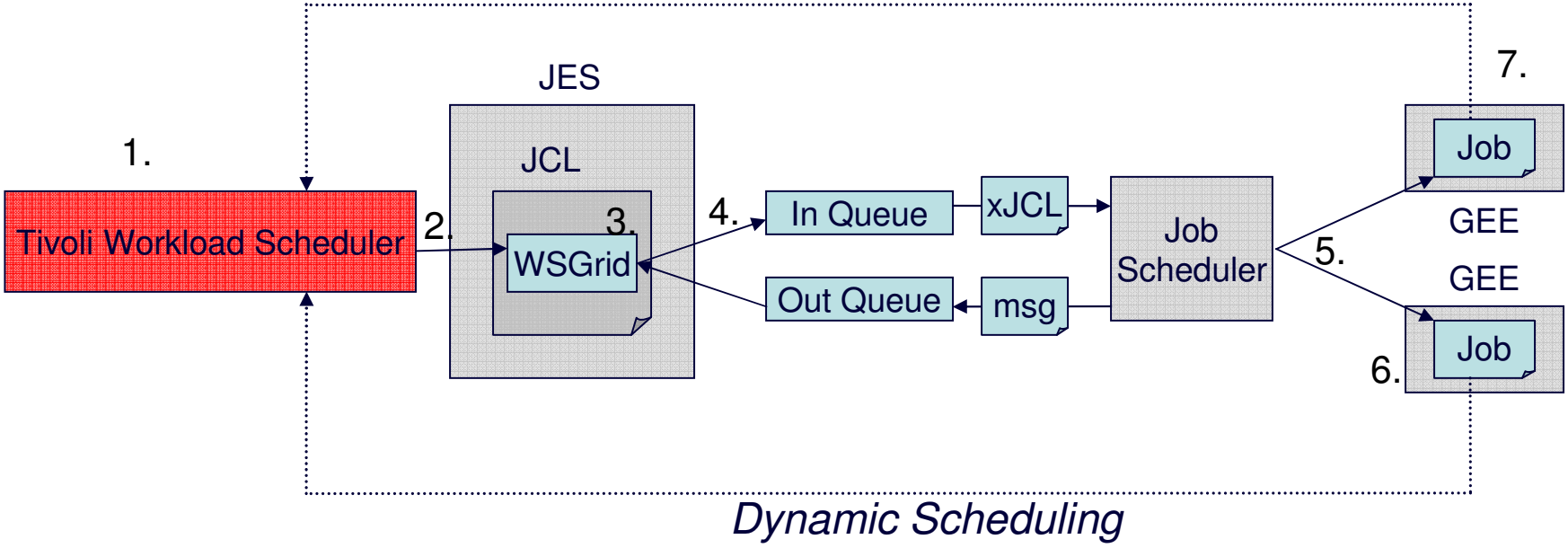


Submitting a job to the Parallel Job Manager



1. Large, single job is submitted to the Job Dispatcher of XD Compute Grid
2. The Parallel Job Manager (PJM), with the option of using job partition templates stored in a repository, breaks the single batch job into many smaller partitions.
3. The PJM dispatches those chunks across the cluster of Grid Execution Environments (GEE)
4. The cluster of GEE's execute the parallel jobs, applying qualities of service like checkpointing, job restart, transactional integrity, etc.

Enterprise Scheduler Integration



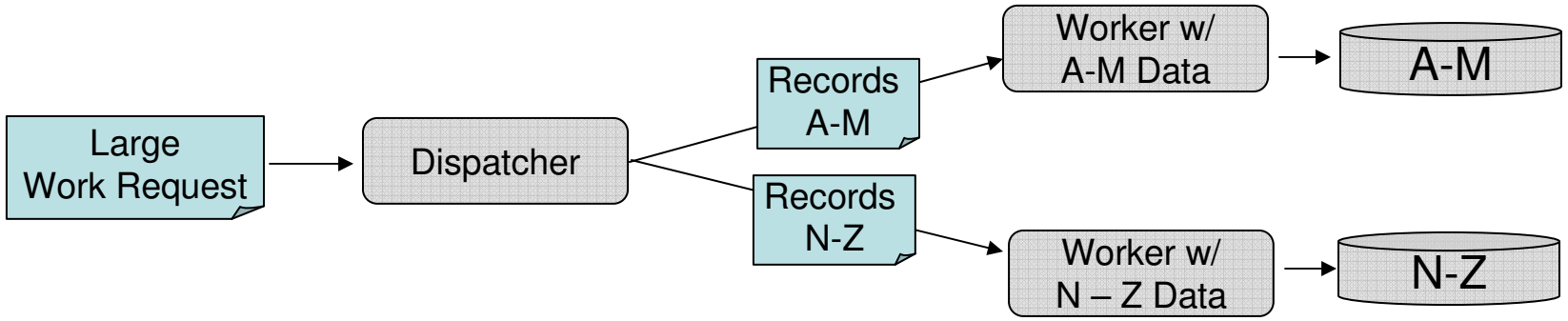
- *enterprise scheduler for operational control*
- *Jobs and commands are submitted from WSGRID*
- *Jobs can dynamically schedule ES via its EJB interface*

Key Influencers for HPC Grids

- *Proximity to the Data*
  - Bring the business logic to the data: co-locate on the same platform
  - Bring the data to the business logic: in-memory databases, caching
- *Data-aware Routing*
  - Partitioned data with intelligent routing of work
- *Divide and Conquer*
  - Highly parallel execution of workloads across the grid
- *On-Demand Scalability*

Compute Grid + XTP = eXtreme Batch

Bringing the data closer to the business logic



-Proximity of the business logic to the data significantly influences performance

-Bring data to the business logic via caching

-Bring business logic to the data via co-location

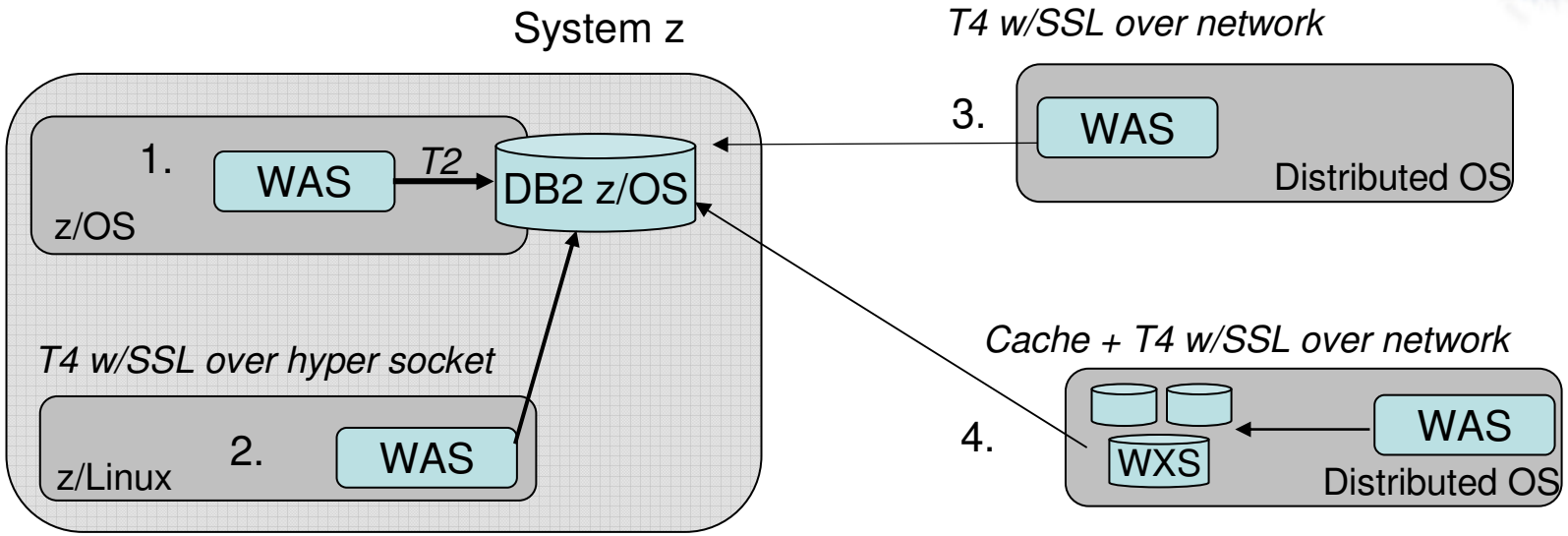
- Increase cache hits and reduce data access through affinity routing

- Data is partitioned across the cluster of workers

- Work requests are divided into partitions that correspond to the data

- Work partitions are intelligently routed to the correct work with the data preloaded.

Proximity of Data - Options

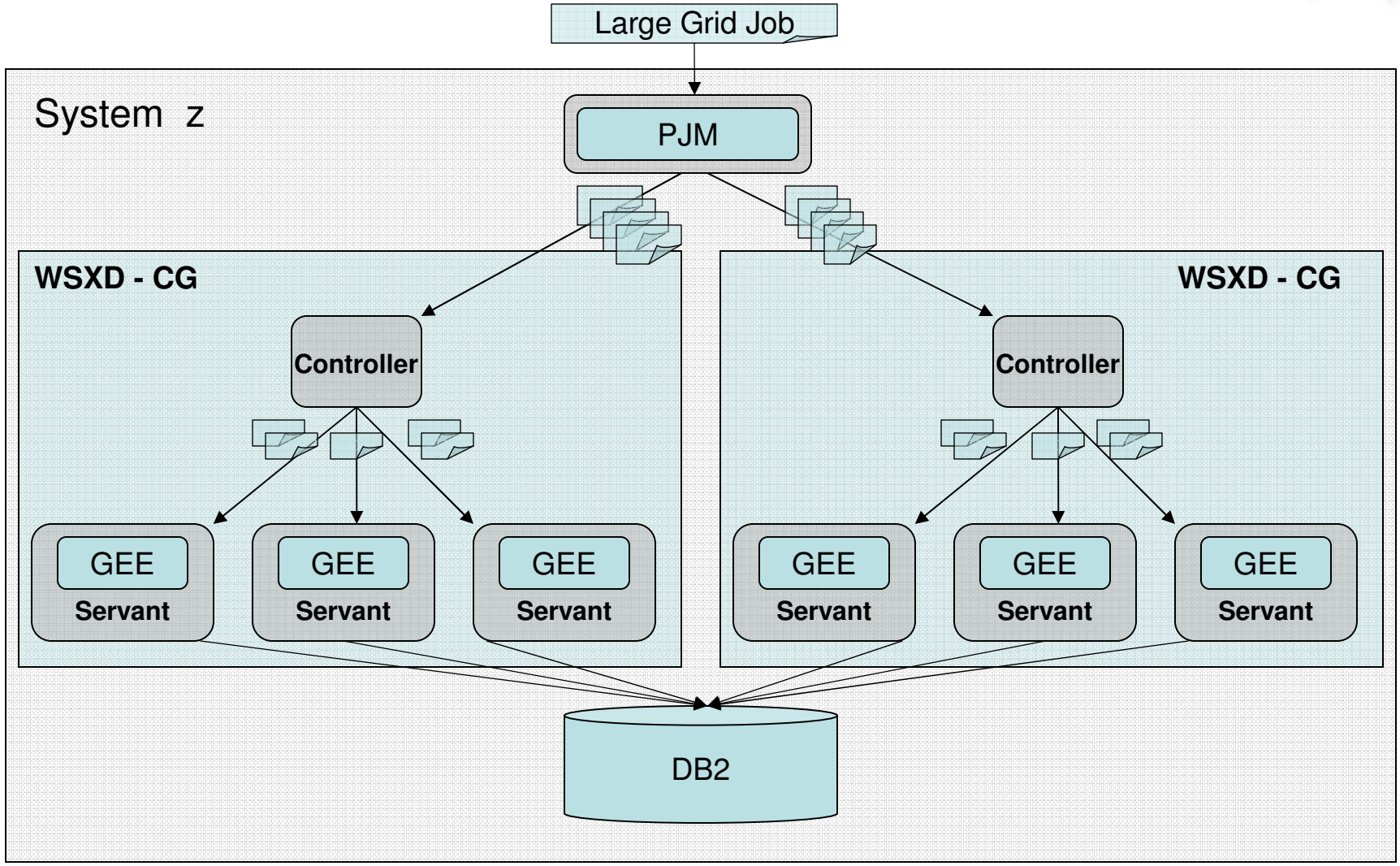


1. WAS z/OS using optimized mem-to-mem JDBC Type-2 Driver
2. WAS z/Linux using JDBC Type-4 driver and SSL over optimized z network stack
3. WAS distributed (unix/linux/windows/etc) using JDBC Type-4 driver and SSL over traditional network stack
4. WAS distributed coupled with WebSphere eXtreme Scale cache

If the data is on z/OS, the batch application should run on z/OS.

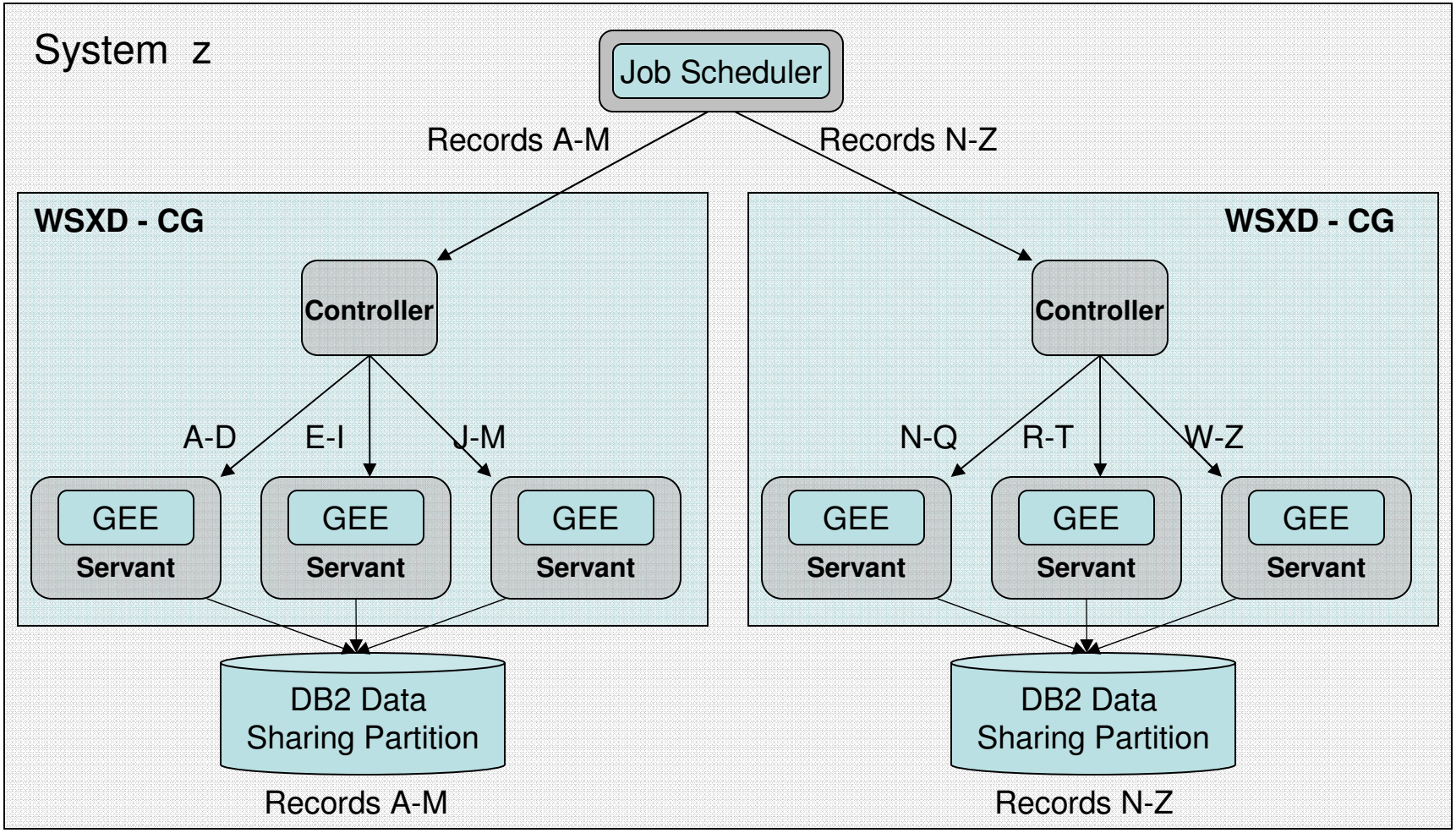
Divide and Conquer

highly parallel grid jobs



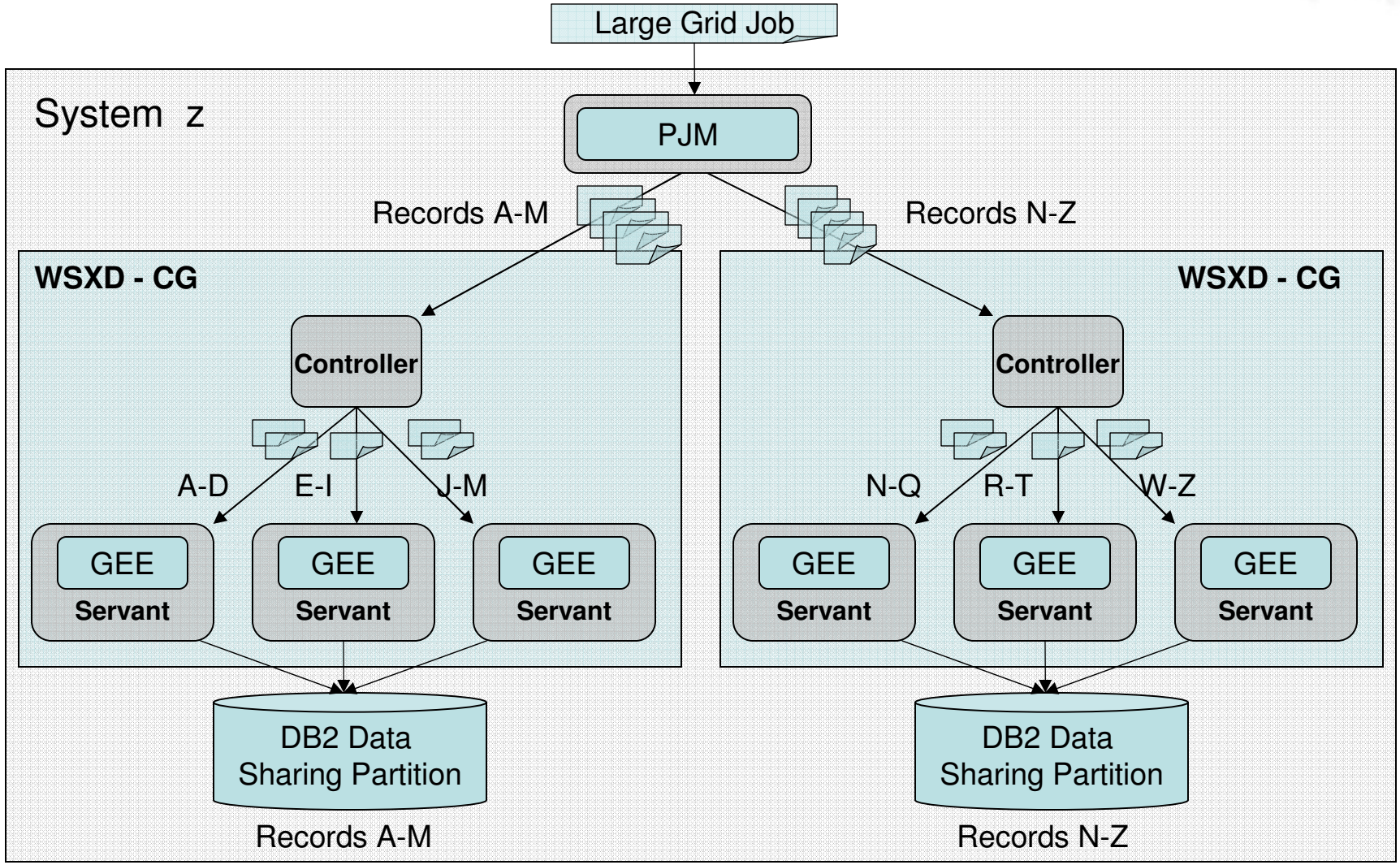
Data-aware Routing

partition data with intelligent workload routing



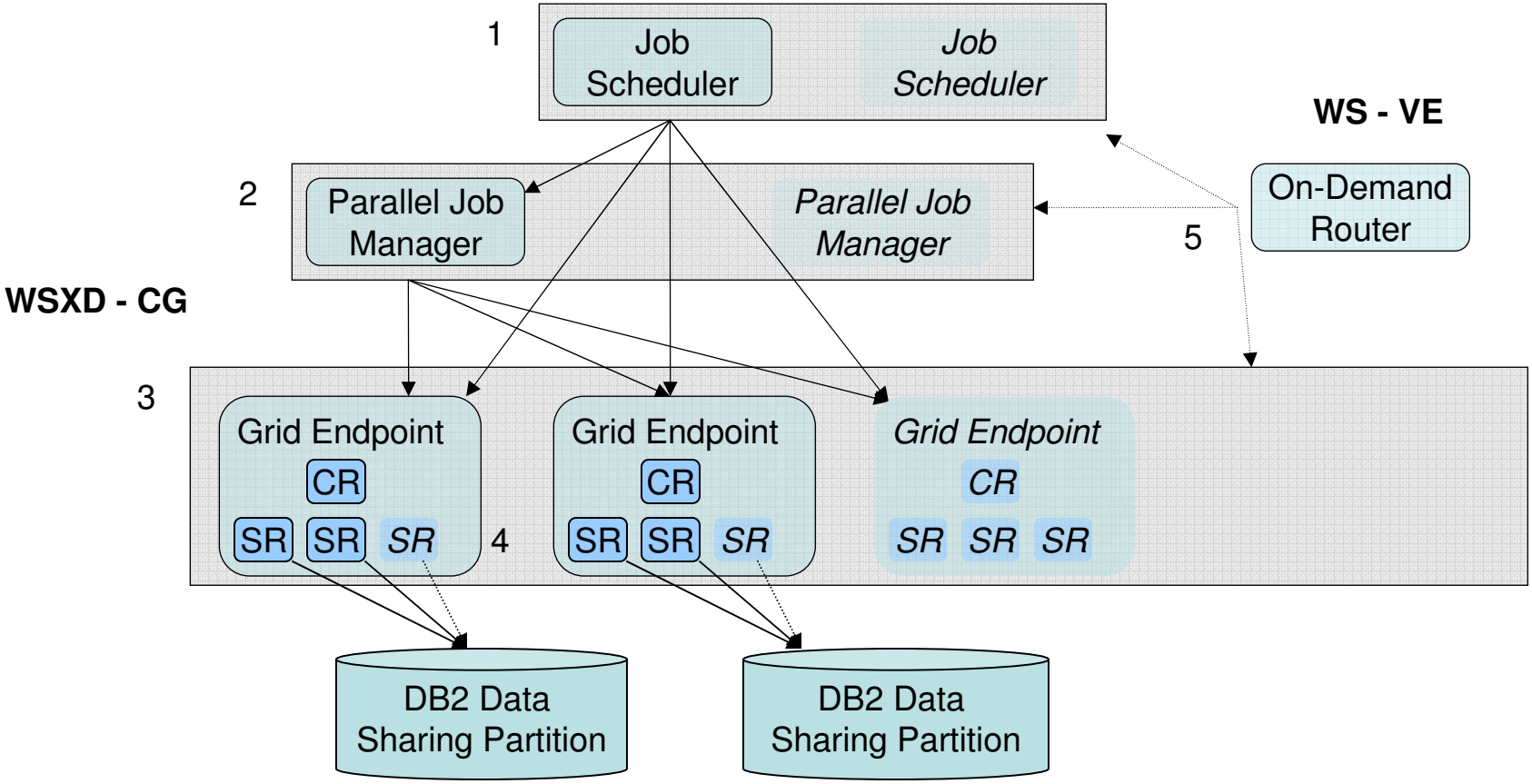
eXtreme Transaction Processing

coupling DB2 data partitioning and parallel processing



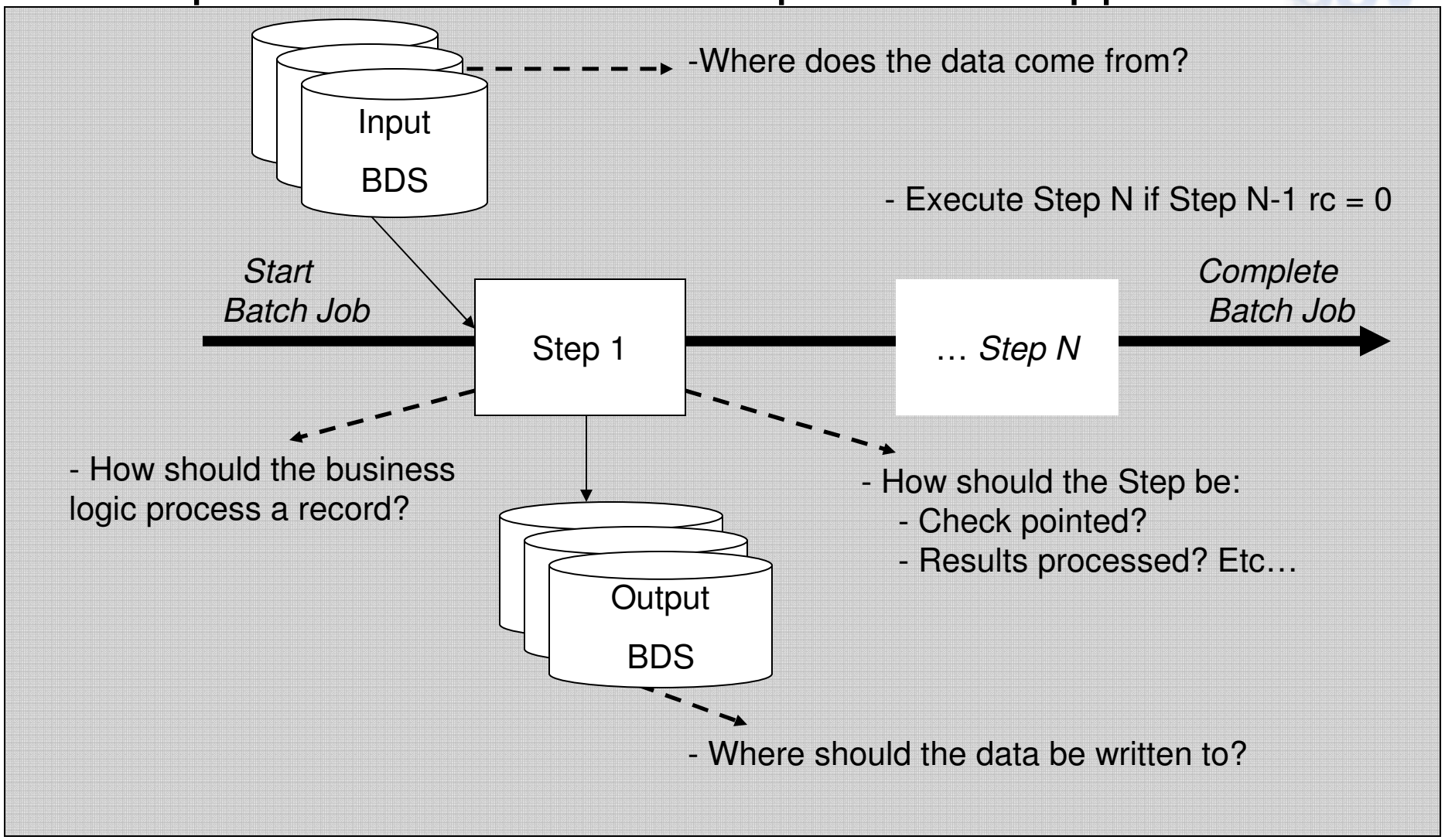
Bringing it all together

batch using CG, HPC and XTP on z/OS

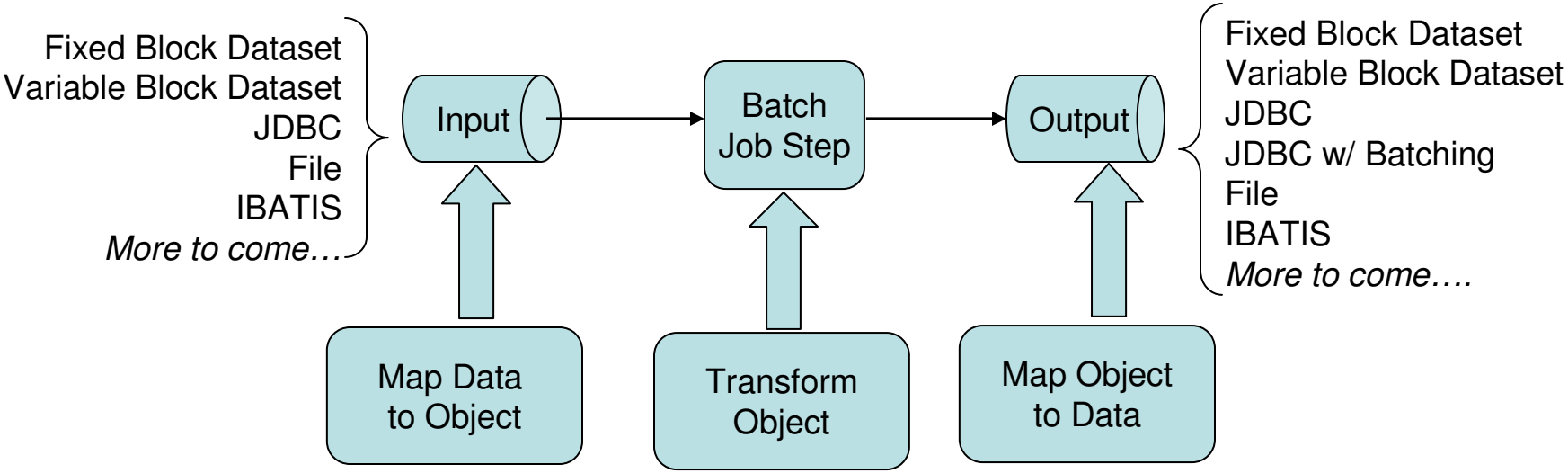


Batch Application Design

Components of an XD Compute Grid Application



How to think about batch jobs



-Customer implements pattern interfaces for input/output/step

-Pattern interfaces are *very* lightweight.

-They follow ***typical*** lifecycle 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.

End-to-end Development tooling

-Customer develops business service POJO's

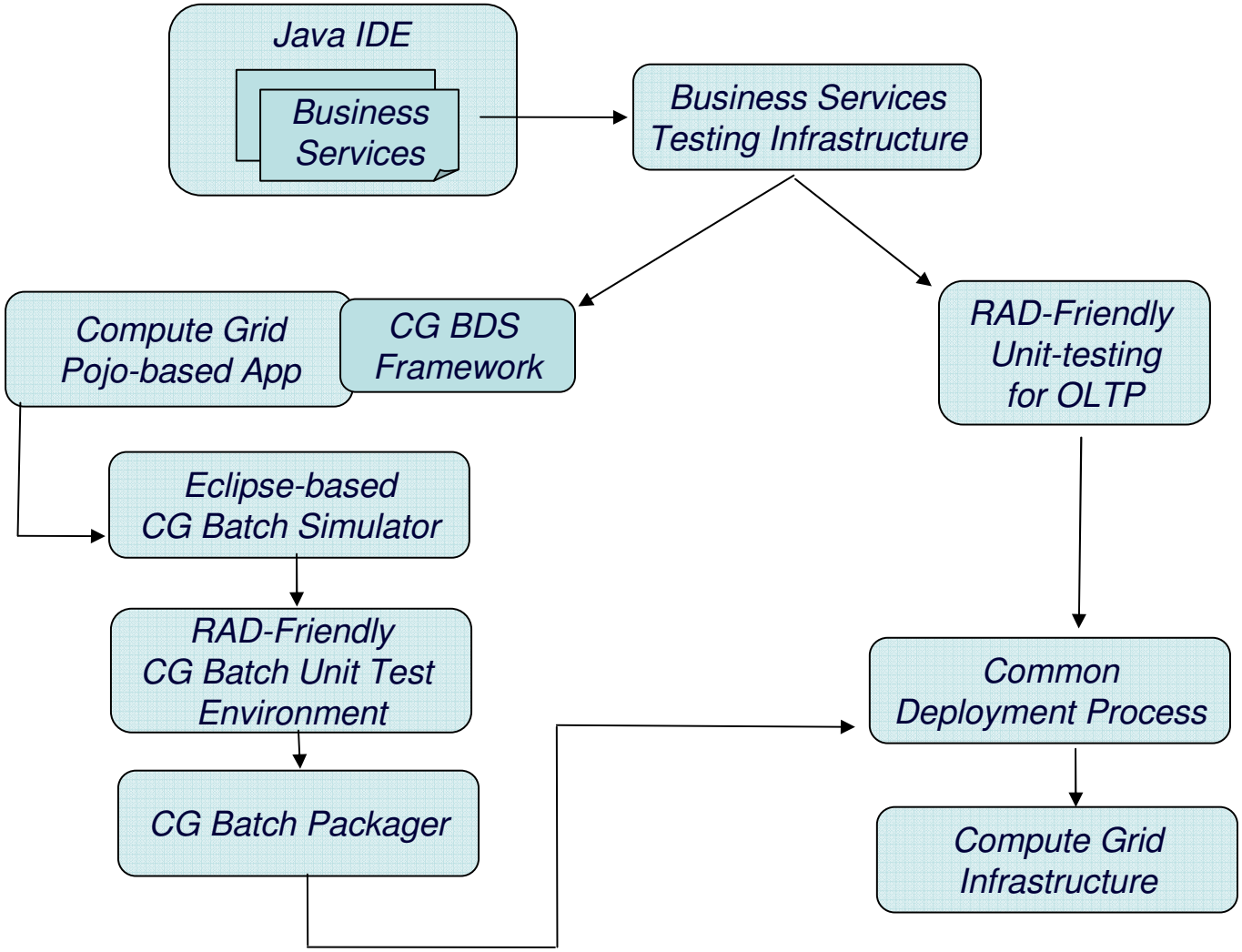
-Applications are assembled via IOC Container

-XD BDS Framework acts as bridge between job business logic and XD Compute Grid programming model

-XD Batch Simulator for development

-XD Batch Unit test environment for unit testing

-XD batch packager for .ear creation



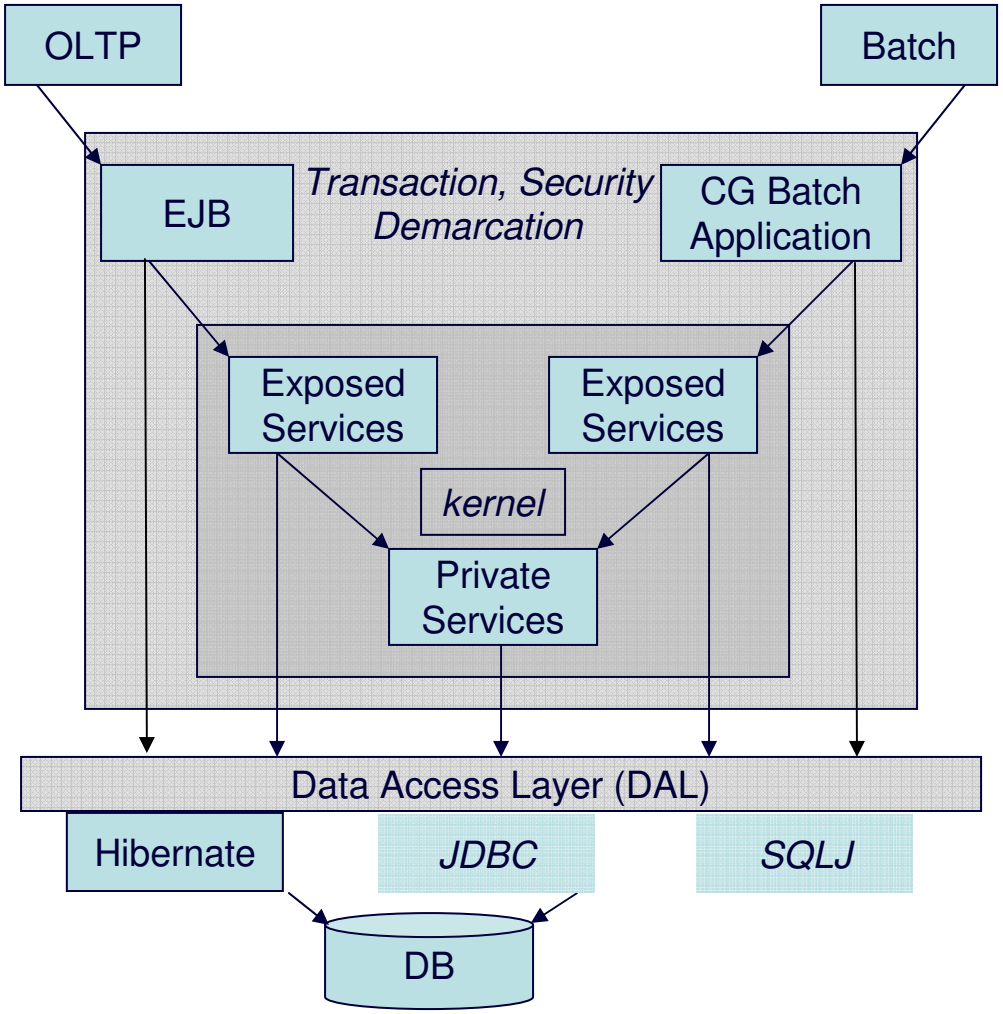
Application Design Considerations

- Strategy Pattern for well structured batch applications
  - Use the BDS Framework!!!
  - Think of batch jobs as a record-oriented Input-Process-Output task
  - Strategy Pattern allows flexible Input, Process, and Output objects *(**think "toolbox"** of input BDS, process steps, and output BDS)*
- Designing "services" shared across OLTP and Batch
  - Cross-cutting Functions (Logging, Auditing, Authorization, etc)
  - Data-injection approach, ***not*** Data-acquisition approach
  - POJO-based "services", not heavy-weight services
  - Be aware of transaction scope for OLTP and Batch. *TxRequiresNew in OLTP + TXRequires in Batch => Deadlock Possible*
- Designing the Data Access Layer (DAL)
  - DAO Factory pattern to ensure options down the road
  - Context-based DAL for OLTP & Batch in same JVM
  - Configuration-based DAL for OLTP & Batch in different JVM's

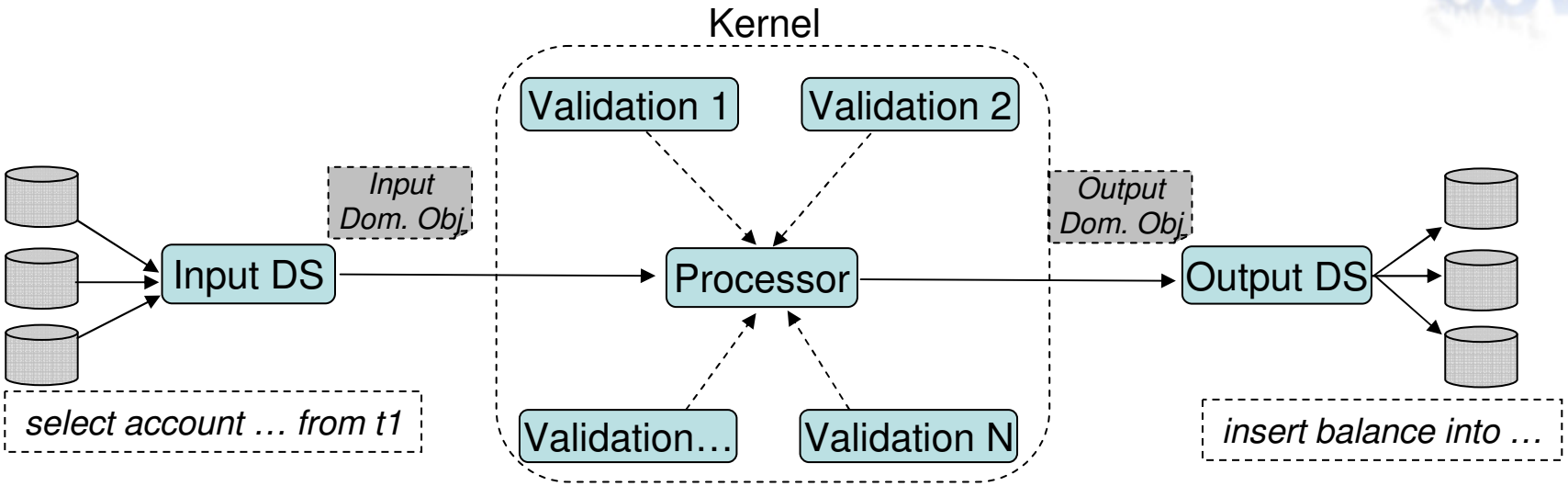
SwissRe Application Architecture for Shared OLTP and Batch Services

-*J2EE and XD manage Security, transactions*

- *Spring-based application Configuration*
- *Custom authorization service within kernel for business-level rules*
- *Initial data access using Hibernate. Investigating JDBC, SQLJ, etc*

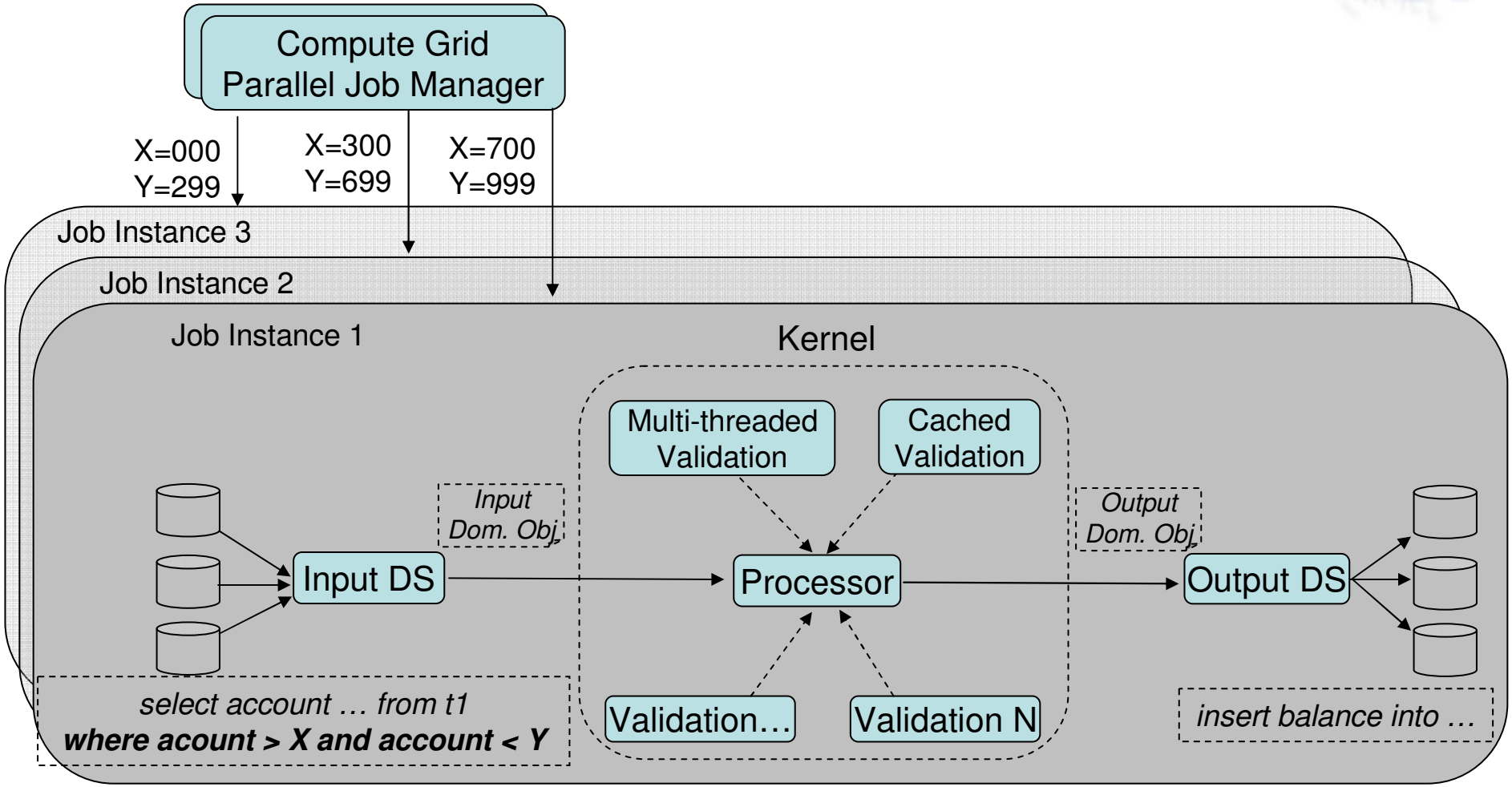


Batch Application Design



1. Batch Input Data Stream (Input DS) manages acquiring data and creating domain objects
2. Record processor applies business validations, transformations, and logic on the object
3. Batch Output Data Stream (Output DS) persists the output domain object
4. Processor and OutputDS are not dependent on Input method (file, db, etc)
5. Processor and OutputDS only operate on ***discrete business records***
6. Customers can use favorite IOC container to assemble the Kernel, and use xJCL to wire the batch flow (input -> process -> output) together.

Batch Application Design - Performance

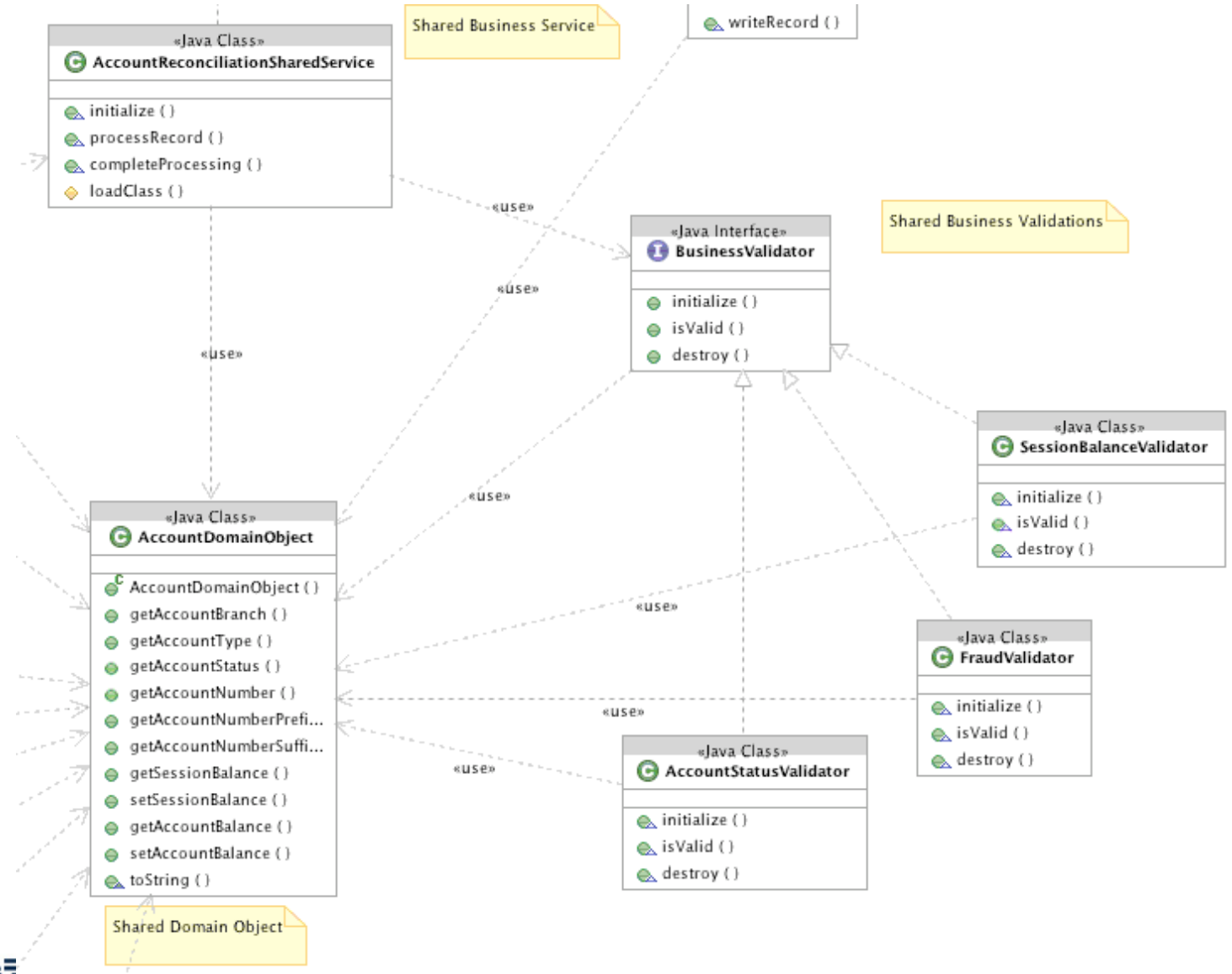


Shared Service Example - "Data Injection" Pattern

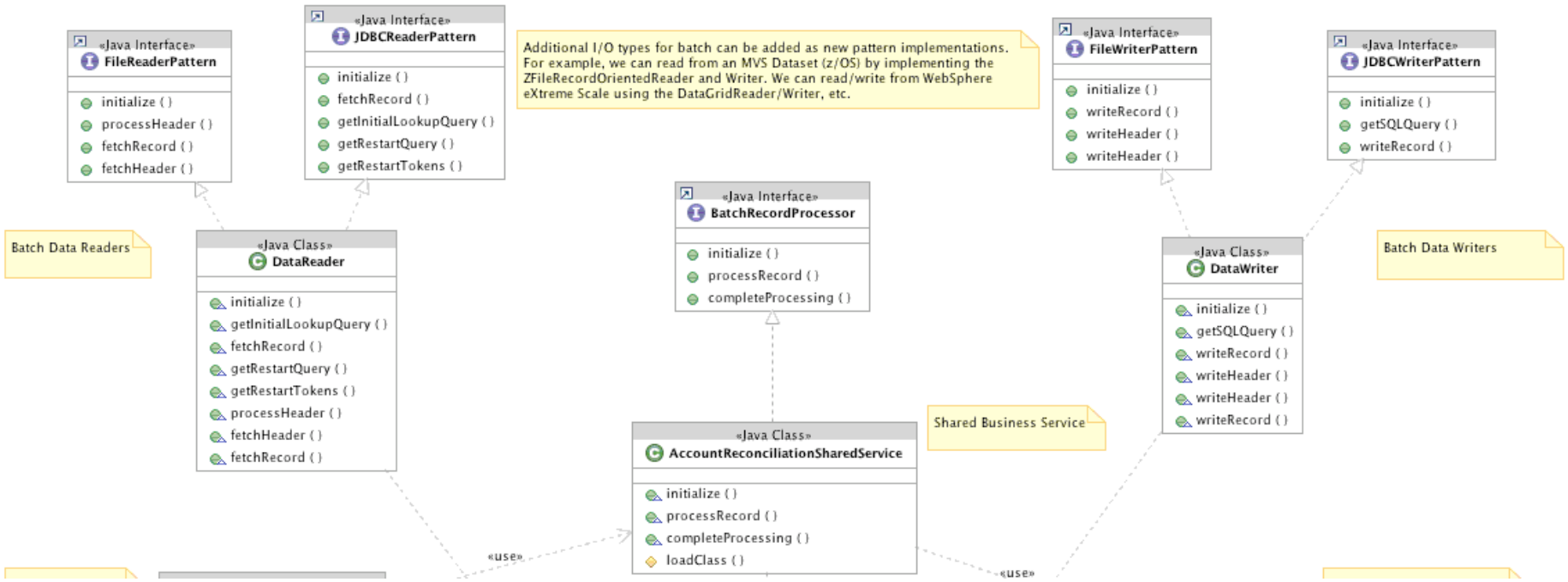
agnostic to source and destination of the data

```
public Object processRecord(Object record) throws Exception {
    AccountDomainObject obj = (AccountDomainObject) record;

    if ((accountStatusValidator.isValid(obj)) && (this.fraudValidator.isValid(obj))&& (this.sessionBalanceValidator.isValid(obj)))
    {
        BigDecimal sum= obj.getSessionBalance().add(obj.getAccountBalance());
        obj.setAccountBalance(sum);
        obj.setSessionBalance(BigDecimal.ZERO);
        return obj;
    }
    else return null;

}
```

Execution Agnostic Application “Kernel”



Compute Grid (Batch) Wrapper to the Shared Service

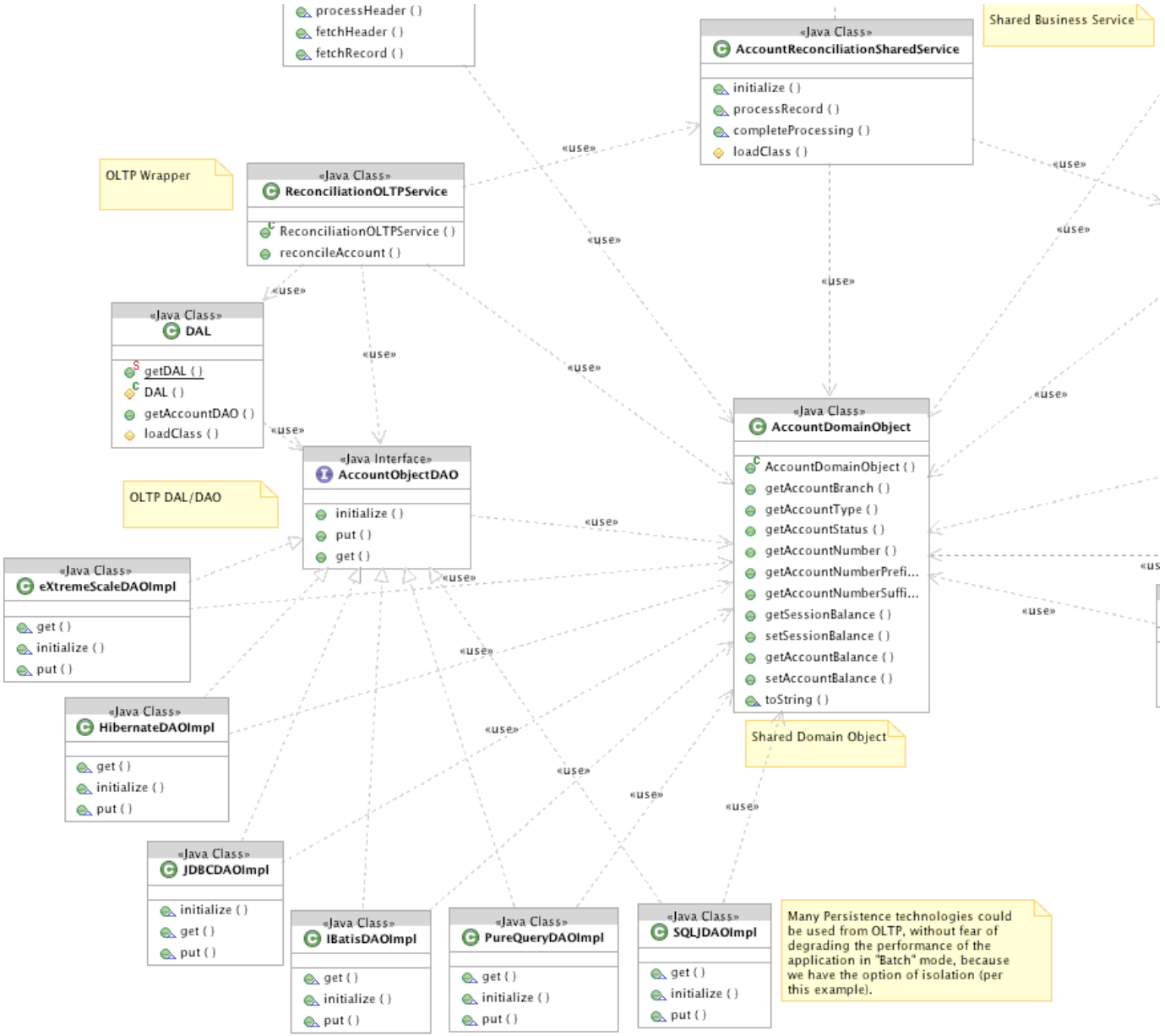


OLTP Wrapper to the Shared Service

*Data is **injected** into the shared service*

```
public class ReconciliationOLTPService {

    protected AccountObjectDAO accountDAO;
    protected AccountReconciliationSharedService reconcileService;

    public ReconciliationOLTPService()
    {
        accountDAO = DAL.getDAL().getAccountDAO();
        reconcileService = new AccountReconciliationSharedService();
        reconcileService.initialize(new Properties());
    }

    public boolean reconcileAccount(String accountId)
    {
        try
        {
            AccountDomainObject obj = accountDAO.get(accountId);
            obj = (AccountDomainObject) reconcileService.processRecord(obj);
            if (obj == null)
                return false;
            else
            {
                accountDAO.put(obj);
                return true;
            }
        }
        catch (Throwable t)
        {
            throw new RuntimeException("Failed to reconcile account. " + t);
        }

    }

}
```

OLTP Wrapper to the Shared Service



Conclusions

WebSphere XD Compute Grid Summary

- IBM WebSphere XD Compute Grid delivers a ***complete batch platform***
  - End-to-end Application Development tools
  - Application Container with Batch QoS (checkpoint/restart/etc)
  - Features for Parallel Processing, Job Management, Disaster Recovery, High Availability
  - Scalable, secure runtime infrastructure that integrates with WebSphere Virtual Enterprise and WLM on z/OS
  - Designed to integrate with existing batch assets (Tivoli Workload Scheduler, etc)
  - Supports all platforms that run WebSphere, including z/OS.
  - Experienced Services and Technical Sales resources available to bring the customer to production

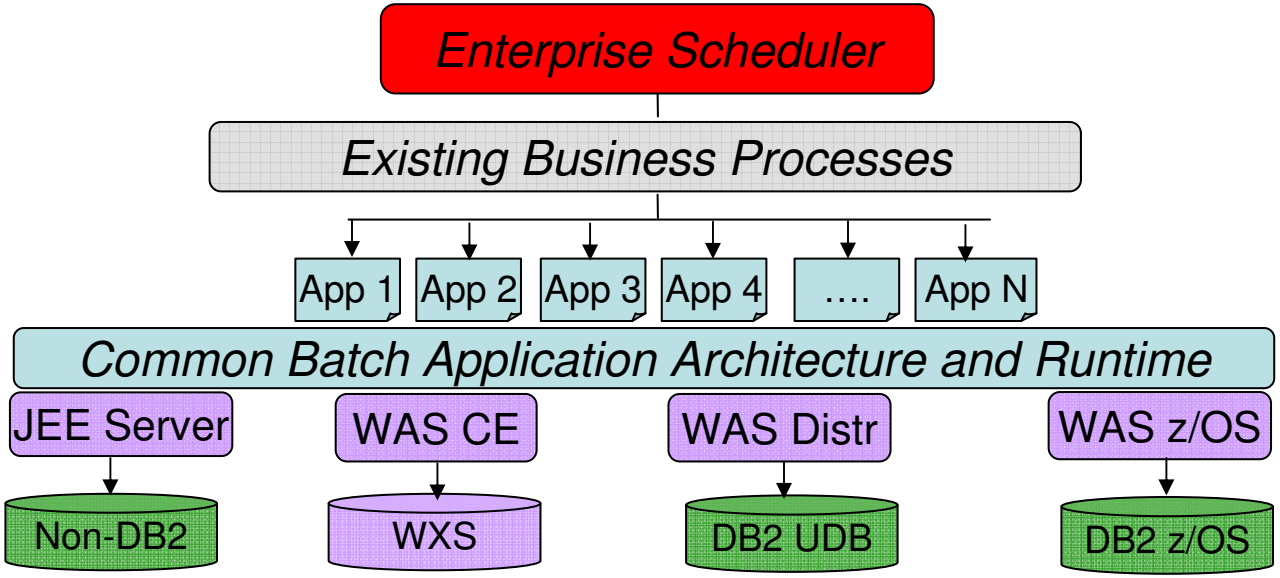
- Is ready for "prime time". Several customers in production on Distributed and z/OS today
  1. Swiss Reinsurance, Public Reference, Production 4/2008 on z/OS
  2. German Auto Insurer, Production 7/2008 on Distributed
  3. Turkish Bank, Production on Distributed
  4. Japanese Bank, Production on Distributed
  5. Danish Bank, Pre-production on z/OS
  6. Wall Street Bank (two different projects), Pre-production on Distributed
  7. South African Bank, Pre-production on Distributed
  8. Danish business partner selling a core-banking solution built on Compute Grid.
  - > 20 customers currently evaluating the product (PoC, PoT)
  - Numerous other customers in pre-production

- Vibrant Customer Community
  - Customer conference held in Zurich in September, 2008. 6 customers and > 50 people attended
  - User group established for sharing best practices and collecting product requirements
  - Over ***30,000*** hits in the Compute Grid developers forum since January 22nd, 2008.

What's next for Compute Grid?

1. 'ilities: Continue to ***Harden*** Compute Grid.
   - end-to-end HA, Scalability, Reliability
   - Further improve consumability: simplify xJCL, more BDSFW Patterns
   - Public and shared Infrastructure Verification Tests (IVT)
2. 24x7 Batch: Enterprise Workload Management.
   - Job Pacing, Job Throttling, Dynamic Checkpoint Intervals
3. Operational Control and Integration: Collaborations across IBM
   - High-performance WSGrid, TWS Integration
4. High Performance
   - Parallel Job Manager improvements, Daemon Jobs for Real-time Processing, Data pre-fetching BDSFW patterns
5. Ubiquity
   - JEE Vendor Portability (Portable Batch Container)
   - Multi-programming model support (Spring Batch, JZOS)

The Batch Vision



- ***Portable Batch applications*** across platforms and J2EE vendors
- 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
- WebSphere Compute Grid Frequently Asked Questions
  http://www-128.ibm.com/developerworks/forums/thread.jspa?threadID=228441&tstart=0
- CCR2 article on SwissRe and Compute Grid
  coming in the November 2008 issue.
- 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/xdcomputegrid/Home?S_TACT=105AGX10&S_CMP=ART

Backup

WebSphere Compute Grid- Quick Summary

- Provide 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.

- 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.

Competitive Differentiation

- **Spring Batch**
  - Only delivers an application container (no runtime!)
  - Spring Batch applications ***can not be workload-managed on z/OS***
  - Competes with the Batch Data Stream (BDS) Framework, which is part of Compute Grid's ***FREE*** application development tooling package.
  - Lacks operational controls like start/stop/monitor/cancel/etc
  - No parallel processing infrastructure
- **Datasynapse, Gigaspaces, Gridgain:**
  - No batch-oriented container services like checkpoint/restart
  - Does not support z/OS
- **Java Batch System** (JBS) and related technologies (**Condor**, **Torque**, etc)
  - No batch-oriented container services like checkpoint/restart
  - Not intended for concurrent Batch and OLTP executions
  - Does not support z/OS
- ***Note***: If the data is on z/OS, the batch application should run on z/OS

Development Tooling Story for WebSphere XD Compute Grid

- 1. The **Batch Datastream (BDS) Framework.** This is a development toolkit that implements the Compute Grid interfaces for accessing common input and output sources such as files, databases, and so on. The following post goes into more details.

  2. a **Pojo-based application development model**. As of XD 6.1, you only have to write Pojo-based business logic. Tooling executed during the deployment process will generate the necessary Compute Grid artifacts to run your application. The following developerworks article goes into more details: Intro to Batch Programming with WebSphere XD Compute Grid

  3. The **Batch Simulator**. A light-weight, non-J2EE batch runtime that exercises the Compute Grid programming model. This runs in any standard Java development environment like Eclipse, and facilitates simpler application development since you're only dealing with Pojo's and no middleware runtime. The Batch Simulator is really for developing and testing your business logic. Once your business logic is sound, you would execute function tests, system tests, and then deploy to production. You can download this from batch simulator download

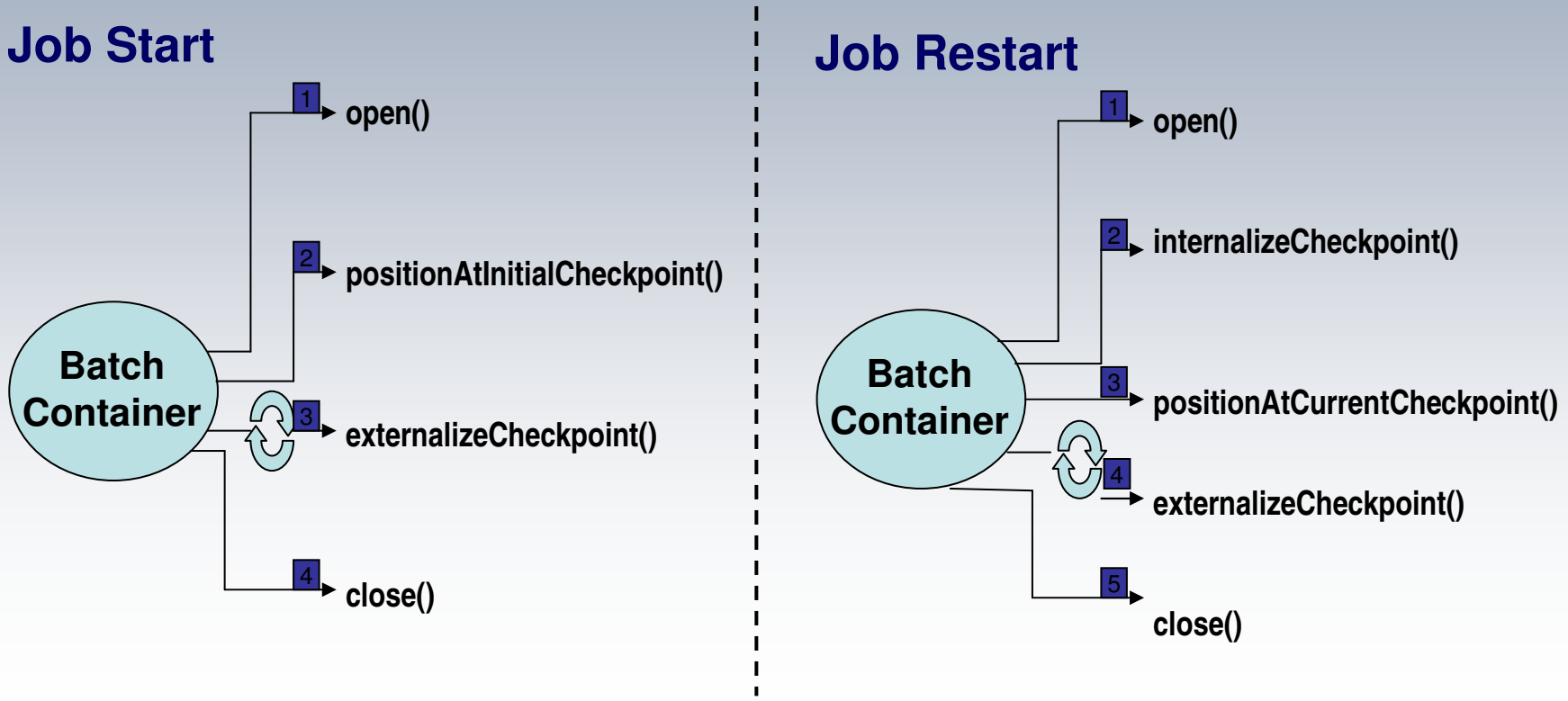
  4. The **Batch Packager**. This utility generates the necessary artifacts for deploying your Pojo-based business logic into the Compute Grid runtime. The packager is a script that can be integrated into the deployment process of your application. It can also be run independently of the WebSphere runtime, so you don't need any heavy-weight installs in your development environment.

  5. The **Unit-test environment (UTE).** The UTE package is described in the following post. The UTE runs your batch application in a single WebSphere server that has the Compute Grid runtime installed. It's important to function-test your applications in the UTE to ensure that it behaves as expected when transactions are applied.

Simple Programming Model …

The anatomy of an transactional batch application – batch data stream

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

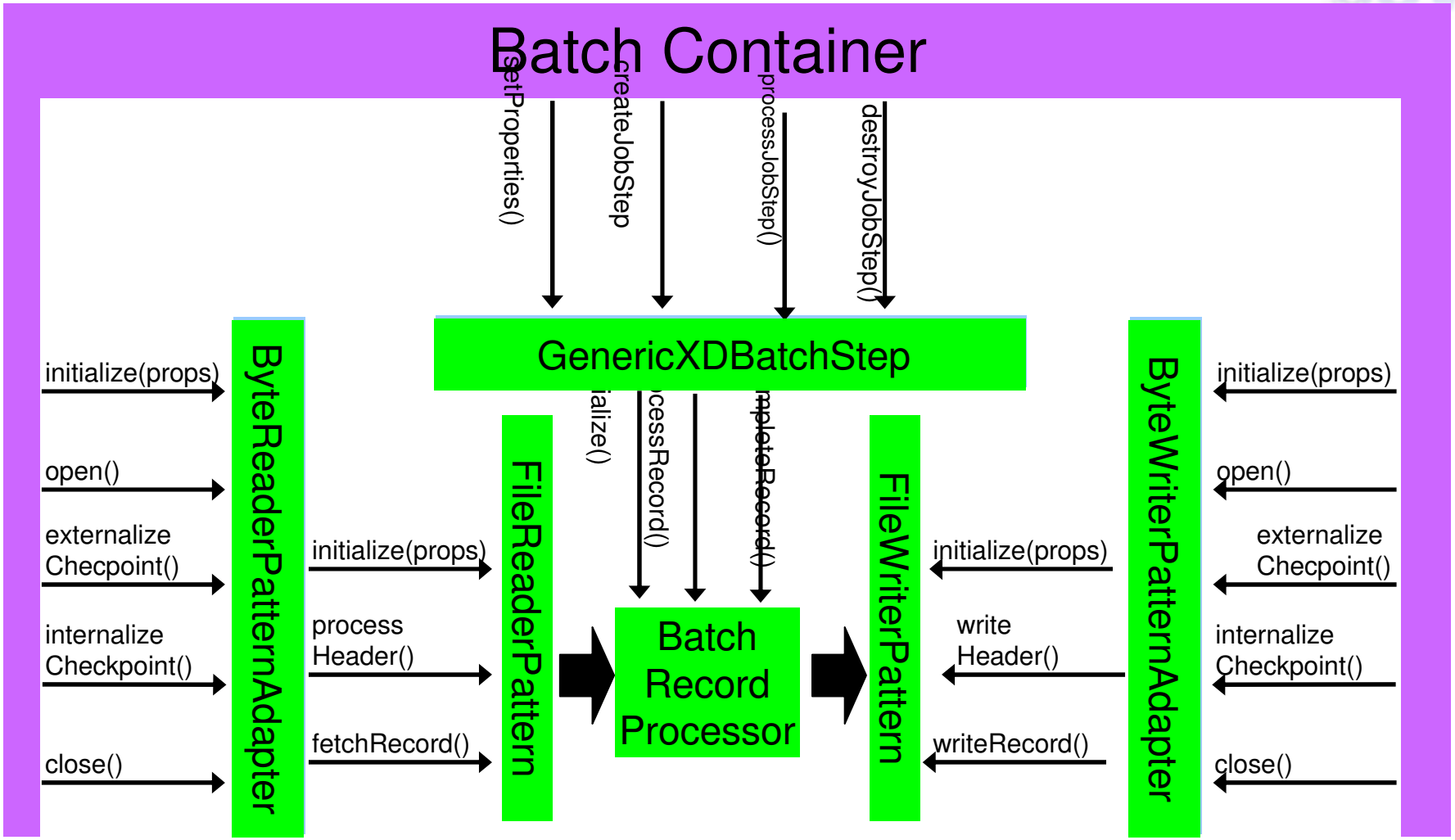


WebSphere XD Compute Grid
BDS Framework Overview

- BDS Framework implements XD batch programming model for common use-cases:
  - Accessing MVS Datasets, Databases, files, JDBC Batching
  - Provides all of the restart logic specific to XD Batch programming model
- Customer's focus on business logic by implementing light-weight pattern interfaces; doesn't need to learn or understand the details of the XD Batch programming model
- Enables XD Batch experts to implement best-practices patterns under the covers
- XD BDS Framework owned and maintained by IBM; will be reused across customer implementations to provide stable integration point for business logic.

```
package com.ibm.websphere.batch.devframework.datastreams.patternadapter;

import java.sql.PreparedStatement;

public interface JDBCWriterPattern {

    public void initialize(Properties props);
    public String getSQLQuery();
    public PreparedStatement writeRecord(PreparedStatement pstmt, Object record);
}
```

The BDS Framework

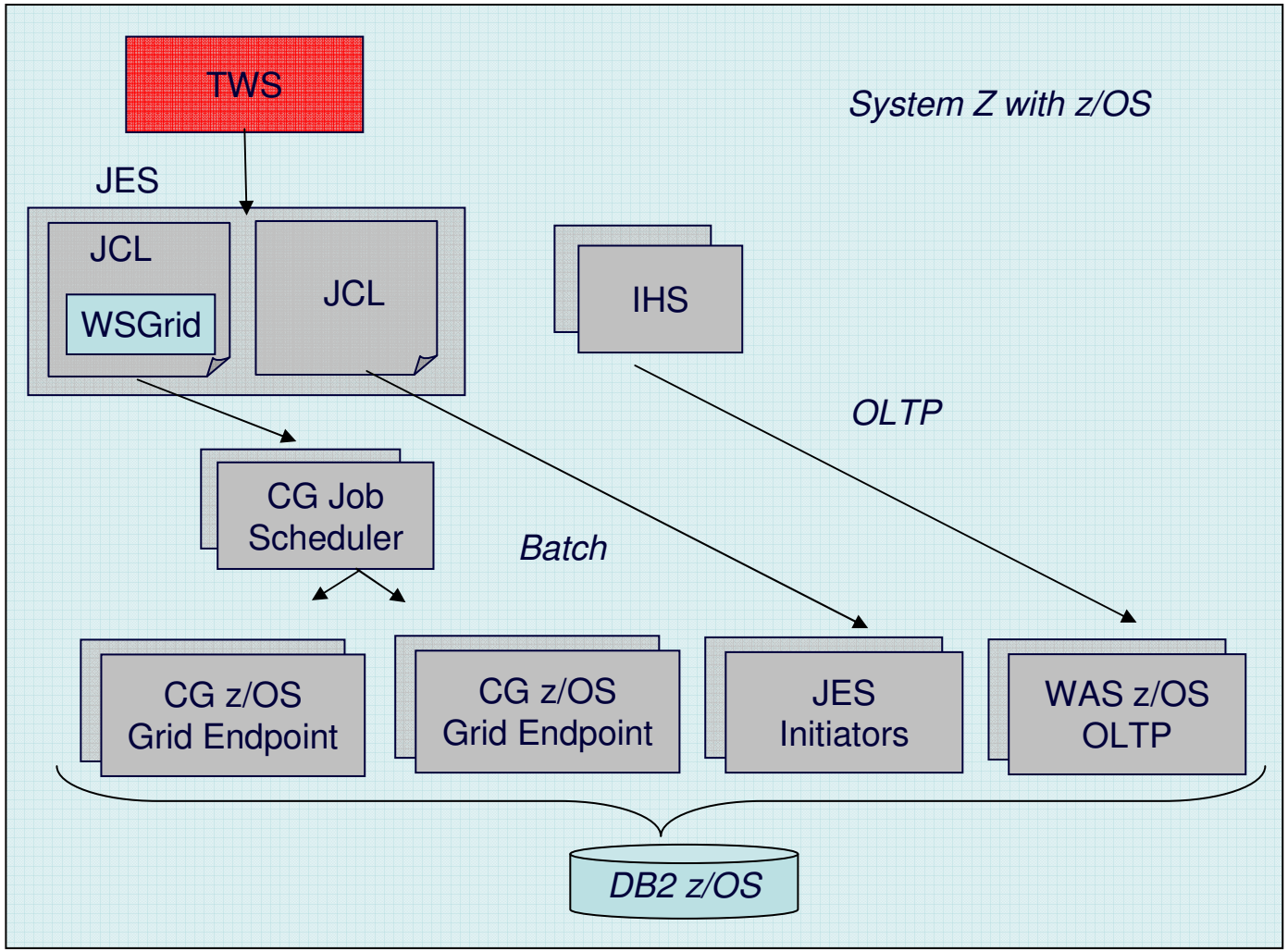


SwissRe Batch and Online Infrastructure

- Maintains close proximity to the data for performance

- Common:

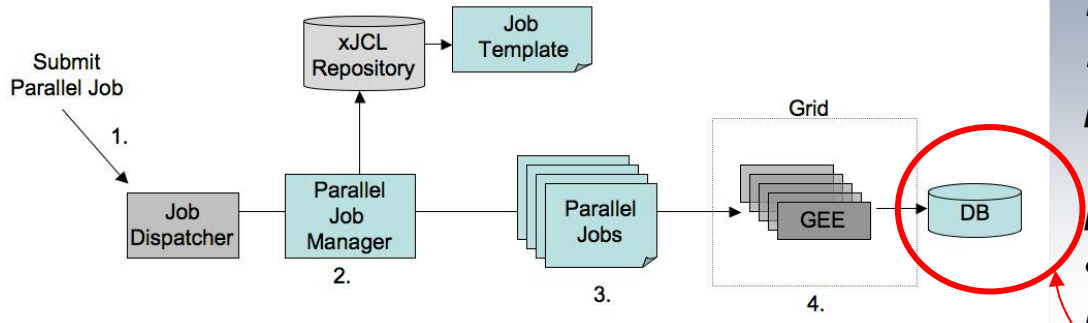
- *Security*
- *Archiving*
- *Auditing*
- *Disaster recovery*



Wall St. Bank

High Performance, Highly-Parallel Batch Jobs with XD Compute Grid and eXtreme Scale on Distributed Platforms

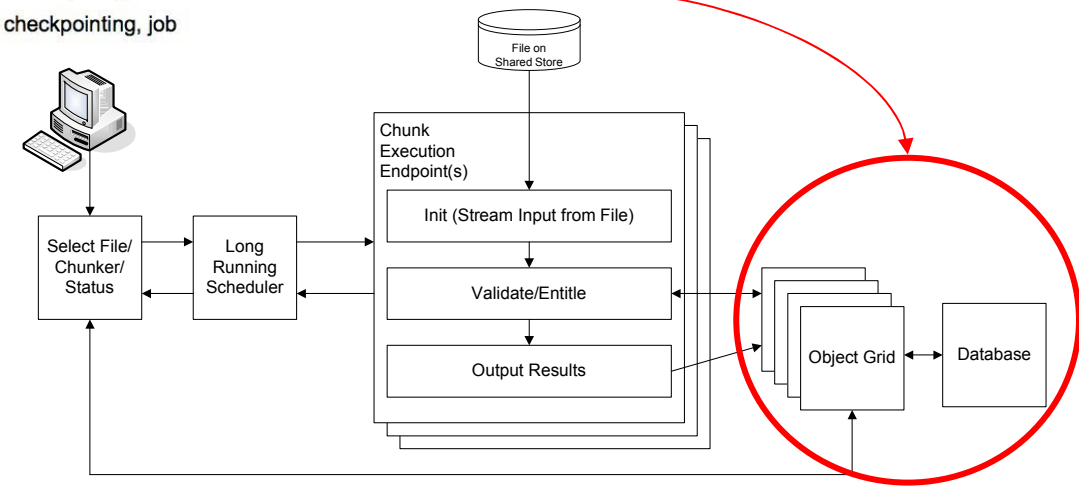
Parallel Processing with Compute Grid



1. Large, single job is submitted to the Job Dispatcher of XD Compute Grid
2. The Parallel Job Manager (PJM), with the option of using job partition templates stored in a repository, breaks the single batch job into many smaller partitions.
3. The PJM dispatches those chunks across the cluster of Grid Execution Environments (GEE)
4. The cluster of GEE's execute the parallel jobs, applying qualities of service like checkpointing, job restart, transactional integrity, etc.

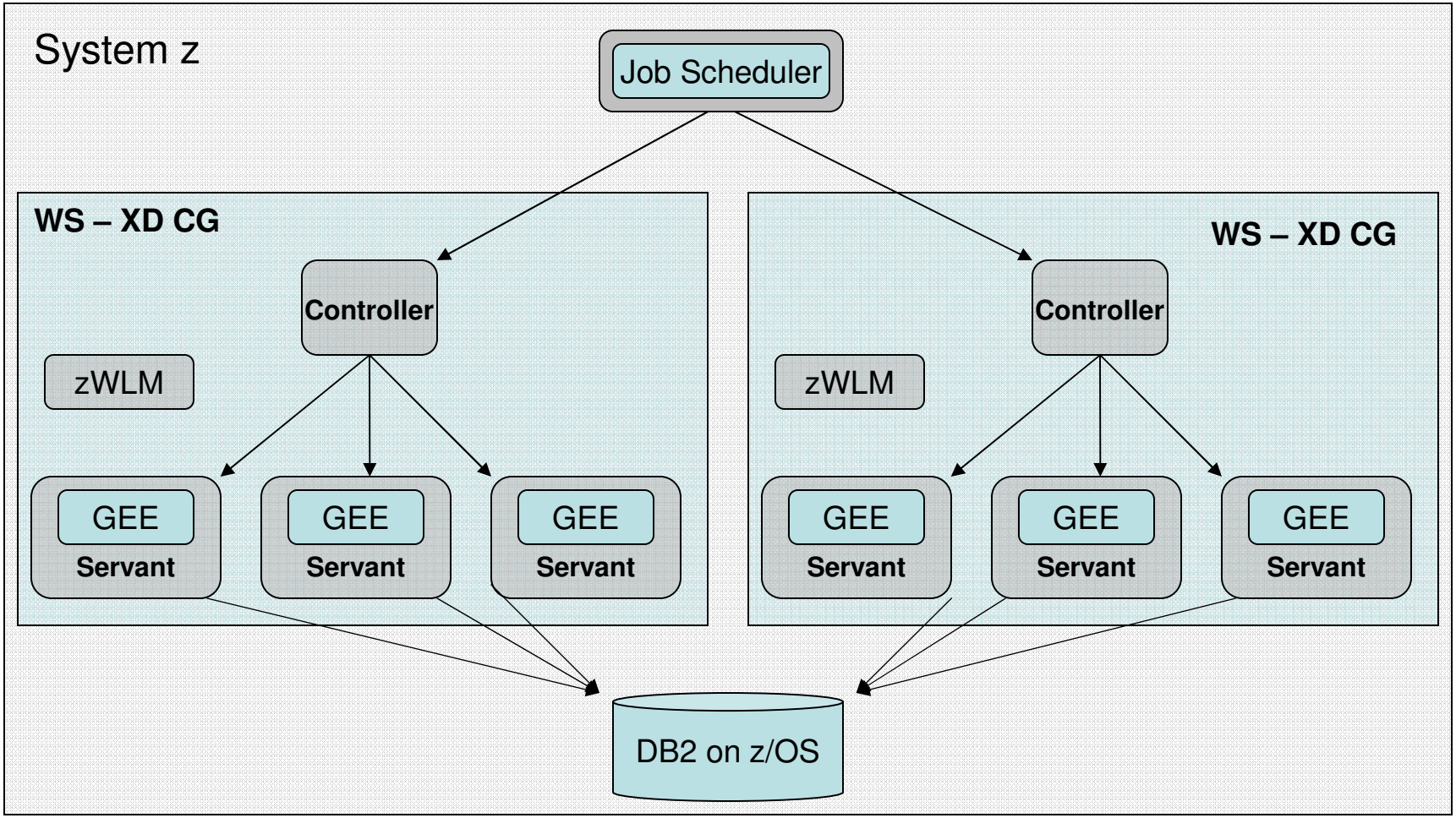
Major Wall St. Bank uses the Parallel Job Manager for highly parallel XD Compute Grid jobs with eXtreme Scale for high-performance data access to achieve a cutting edge grid platform

Applying the Pattern at the bank

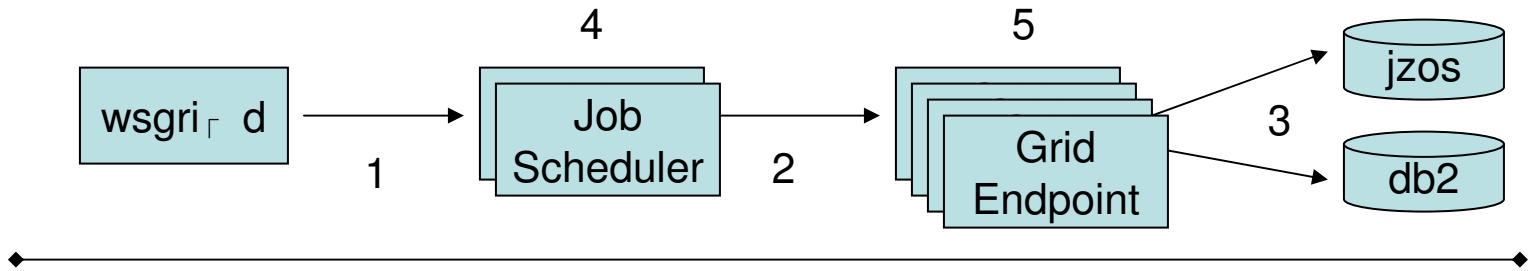


On-Demand Scalability

with WebSphere z/OS



1. WSGrid to Job Scheduler
2. Job Scheduler to Grid Endpoint
3. Grid Endpoint to resources (DB2, JZOS)
4. Job Scheduler High Availability
5. Grid Endpoint High Availability



End-to-end:

6. Monitoring
7. Security
8. Workload Management
9. Disaster Recovery
10. Life-cycle (application, xJCL, etc)

End-to-end Failure Scenarios

1. GEE servant fails (z/OS)

   A:

   - Job status is restartable.
   - wsgrid ends with non-zero return code
   - rc should be 0, 4, 8, 12, 16

   B:

   - WLM starts a new servant
   - servant failure is transparent to WSGrid
   - Job restarted transparently on available servant.

2. GEE CR fails (z/OS, but synonymous to Server failure on Distributed)

   - Job is in restartable state
   - WSGrid receives non-zero return code.

End-to-end Failure Scenarios

3. Job Scheduler SR fails (z/OS)

A: Jobs in execution (dispatched from this Scheduler)

- WSGrid continues running
- Job continues to run.
- Failure is scheduling tier is transparent to job execution.

B: New jobs being scheduled

- New SR starts, business as usual.
- SR fails to start, job should be available for other scheduler to manage.
- if any Job Scheduler SR is available in the system, the job must be scheduled! Failure should be transparent to the job submitter.

4. Job Scheduler CR fails (z/OS, but synonymous to Server failure on Distributed)

- WSGrid and Job continue to run. Any failure in scheduler tier is transparent to job and user. (goal)
- Interim: WSGrid fails with non-zero RC; job managed by this JS should be canceled

End-to-end Failover Scenarios

5. Scheduler Messaging Engine (Adjunct) fails (z/OS)
   - Jobs managed by this JS are canceled. WSGrid fails with non-zero RC.
   - Note: use of messaging engine (SIB generally) is just an interim solution. Shared queues, etc needed.

6. WSGrid is terminated
   - Job is canceled

7. Quiesce the LPAR/Node (for rolling IPL and system maintenance)
   1. No new work should be scheduled to JS on that node. Work should be routed to other JS
   2. no new work should be submitted to GEE on that node. Work should be routed to other GEE's
   3. After X time interval (3.5 hours in SwissRe's case), jobs running in that GEE should be stopped.
   4. After Y time interval (4 hours in SwissRe's case), where x < y, jobs still running in the GEE should be canceled.
   5. WSGrid gets non-zero RC for steps 3 and 4.

Execution Environment – z/OS WLM Integration

- WAS uses WLM to control the number of Servant Regions
- Control Regions are MVS started task
- Servant Regions are started automatically by WLM an a as-needed basis
- WLM queues the user work from the Controller to the Servant region according to service class
- WLM queuing places user requests in a servant based on same service class
- WLM ensures that all user requests in a given servant has been assigned to the same service class
- A Servant running no work can run work assigned to any service class
- WLM and WAS Worker thread : WLM dispatch work as long as it has worker threads
- Behavior of WAS Worker Threads (ORB workload profile)
  - ISOLATE : number of threads is 1. Servants are restricted to a single application thread
  - IOBOUND : number of threads is 3 * Number of CPUs)
  - CPUBOUND : number of threads is the Number of CPUs)
  - LONGWAIT : number of threads is 40
- XD service policies contain one or more transaction class definition
- XD service policies create the goal, while the job transaction class connects the job to the goal
- XD service policy transaction class is propagated to the Compute Grid Execution Environment
- Transaction class is assigned to a job during by the Scheduler during dispatch/classification phase
- When a job dispatch reaches GEE the Tclass is extracted from the HTTP request
- Tclass is mapped to WLM service class. An enclave is created.
- XD Service policies are not automatically defined in the z/OS WLM.

Execution Environment – z/OS Security Considerations

- Compute Grid runs jobs under server credential by default
- In order to run jobs under user credential:
  - WAS security must be enabled
  - Application security must be enabled
  - WebSphere variable RUN_JOBS_UNDER_USER_CREDENTIAL must be set to “true”
  - Enable z/OS thread identity synchronization
  - Enable RunAs thread identity

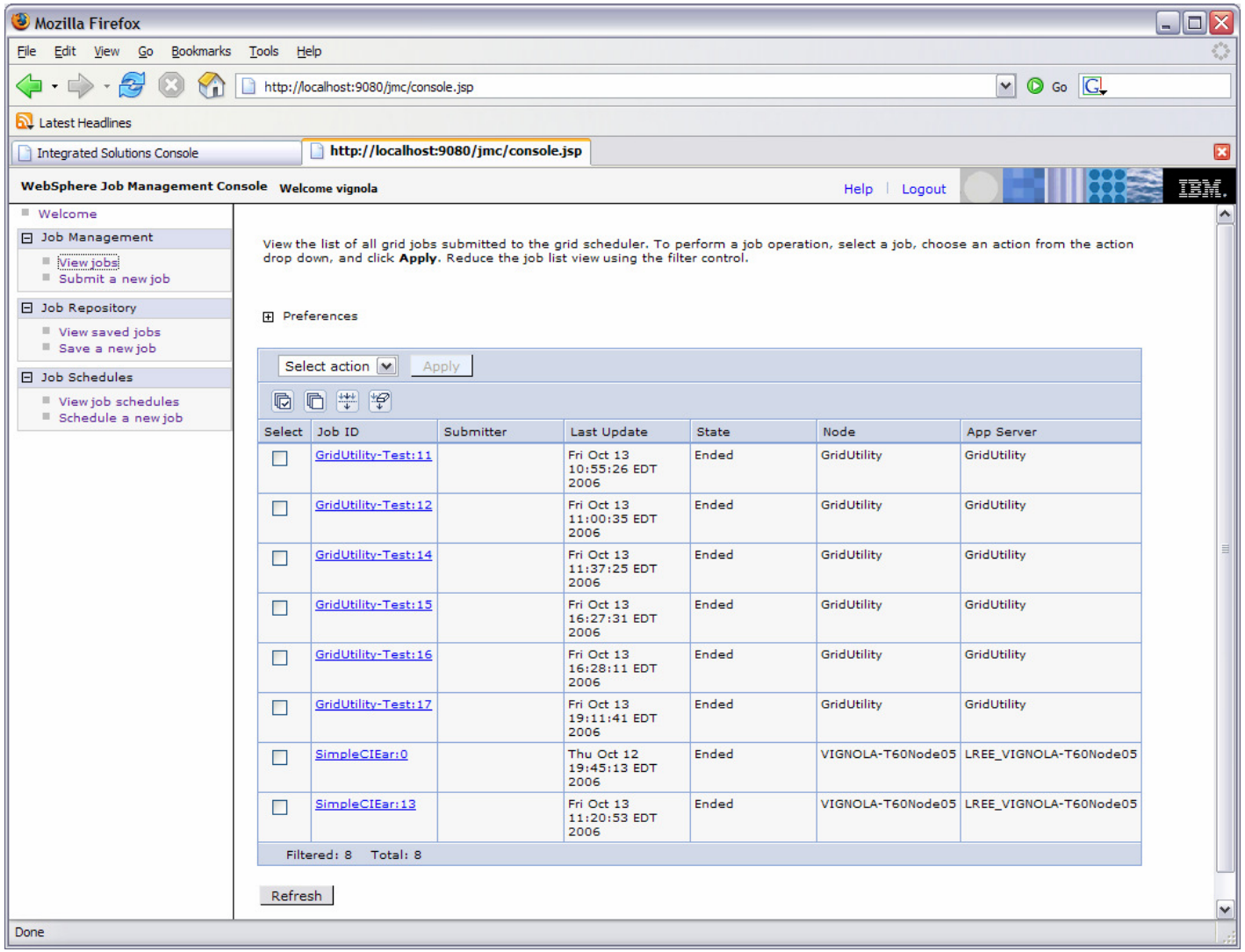
Compute Grid – z/OS Integration Summary

- SMF accounting records for J2EE batch jobs
  - SMF 120 (J2EE) records tailored to jobs
  - Record includes: job id, user, accounting string, CPU time
- Dynamic Servants for J2EE batch job dispatch
  - XD v6.1.0.0 uses pre-started servants (min=max, round-robin dispatch)
  - XD v6.1.0.1 New support will exploit WLM to start new servants to execute J2EE batch jobs on demand
- Service policy classification and delegation
  - Leverages XD job classification to select z/OS service class by propagating transaction class from Job Entry Server to z/OS app server for job registration with WLM

Compute Grid – Job Management Console

- **Web Interface to Scheduler**
  - Hosted in same server (cluster) that hosts scheduler function
  - Replaces job management function formerly found in admin console
- **Essential job management functions**
  - job submission
  - job operations
    - cancel, stop
    - suspend, resume
    - restart, purge
  - job repository management
    - save, delete job definitions
  - job schedule management
    - create, delete job schedules
- **Basic Security Model**
  - userid/password login
  - lrsubmitter, lrAdmin roles

Compute Grid – Jog Management Console



Infrastructure Design Considerations

- High Availability practices
  - Job Scheduler can be made highly available (as of 6.1)
  - Cluster GEE's
- Disaster Recovery practices
  - Today, Active/Inactive approach
  - Tomorrow, Active/Active approach
- Security
  - Job Submitter and Compute Grid Admin roles
  - Options for using Job Submitter identity or Server's identity ***(Performance degradation today!)***
- Connecting Compute Grid to the Enterprise Scheduler
  - JMS Client connector bridges enterprise scheduler to Job Scheduler
  - JMS best practices for securing, tuning, etc apply

High Availability

Topology Questions…

- First, is the Parallel Job Manager (PJM) needed, will you run highly-parallel jobs?
- What are the *high availability* requirements for the JS, PJM, and GEE?
  - Five 9's? Continuous?
- What are the s*calability* requirements for the JS, PJM, GEE?
  - Workloads are predictable and system resources are static?
  - Workloads can fluctuate and system resources are needed on-demand?
- What are the performance requirements for the batch jobs themselves?
  - They must complete within some constrained time window?
- What will the workload be on the system?
  - How many concurrent jobs? How many highly-parallel jobs? Submission rate of jobs?

Topology Considerations…

- If the Job Scheduler (JS) does not have system resources available when under load, managing jobs, monitoring jobs, and using the JMC will be impacted.
- If the PJM does not have system resources available when under load, managing highly parallel jobs and monitoring the job partitions will be impacted.
- If the GEE does not have system resources available when under load, the execution time of the business logic will be impacted.
- The most available and scalable production environment will have:
  - Redundant JS. JS clustered across two datacenters.
  - Redundant PJM. PJM clustered across two datacenters.
  - n GEE's, where n is f(workload goals). Clustered across two datacenters

Cost Considerations…

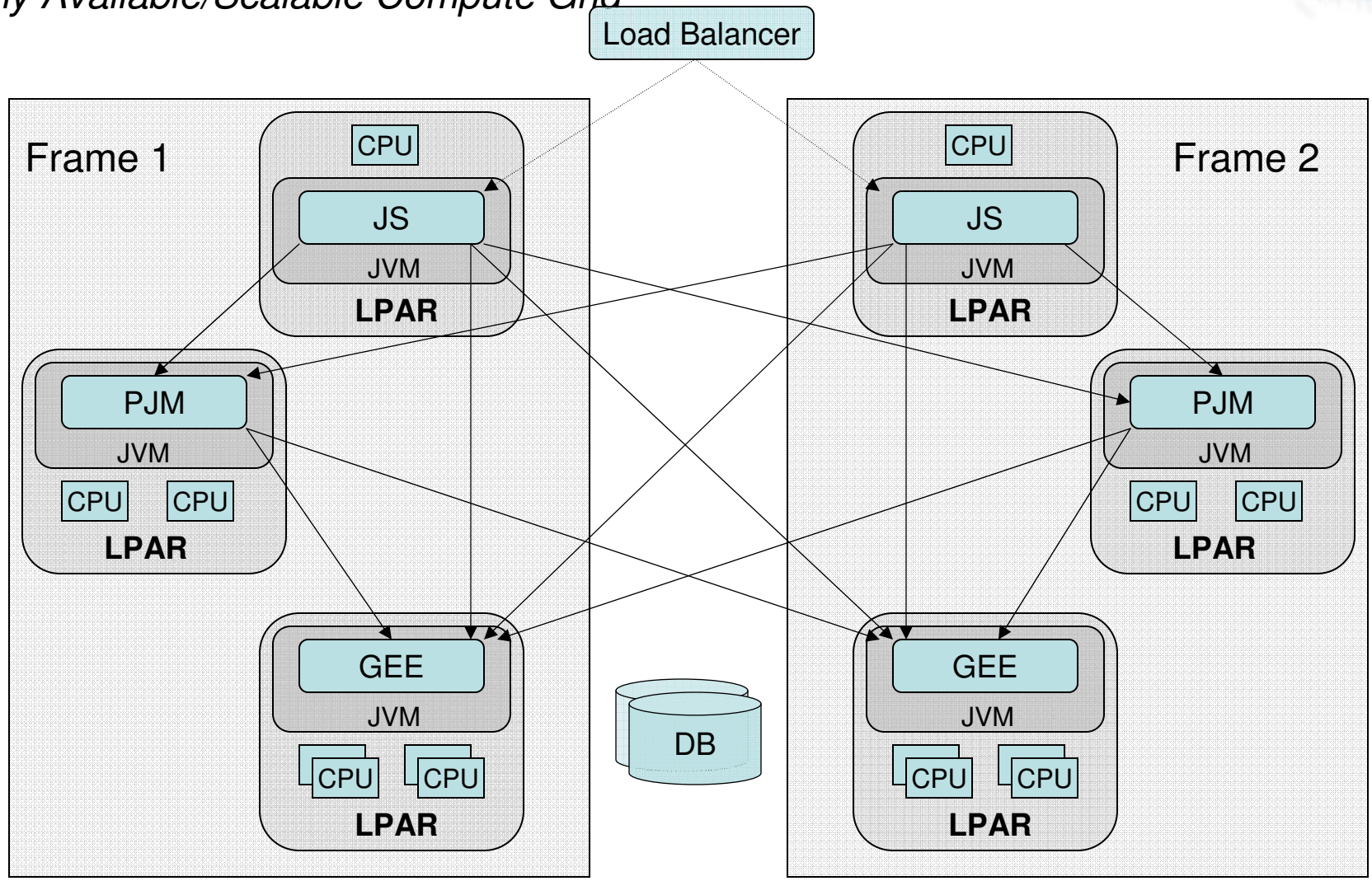
- GEE will most likely require the most CPU resources. The total number of CPU's needed is dependent on:
  - the workload goals
  - max number of concurrent jobs in the system.

- PJM will require fewer CPU's than the GEE. The total number of CPU's needed is dependent on:
  - Rate at which highly-parallel jobs are submitted
  - Max number of concurrent parallel partitions running in the system.

- Job Scheduler will require fewer CPU resources than the GEE, and perhaps the PJM too. The total number of CPU's needed is dependent on:
  - Rate at which jobs will be submitted
  - Max number of concurrent jobs in the system

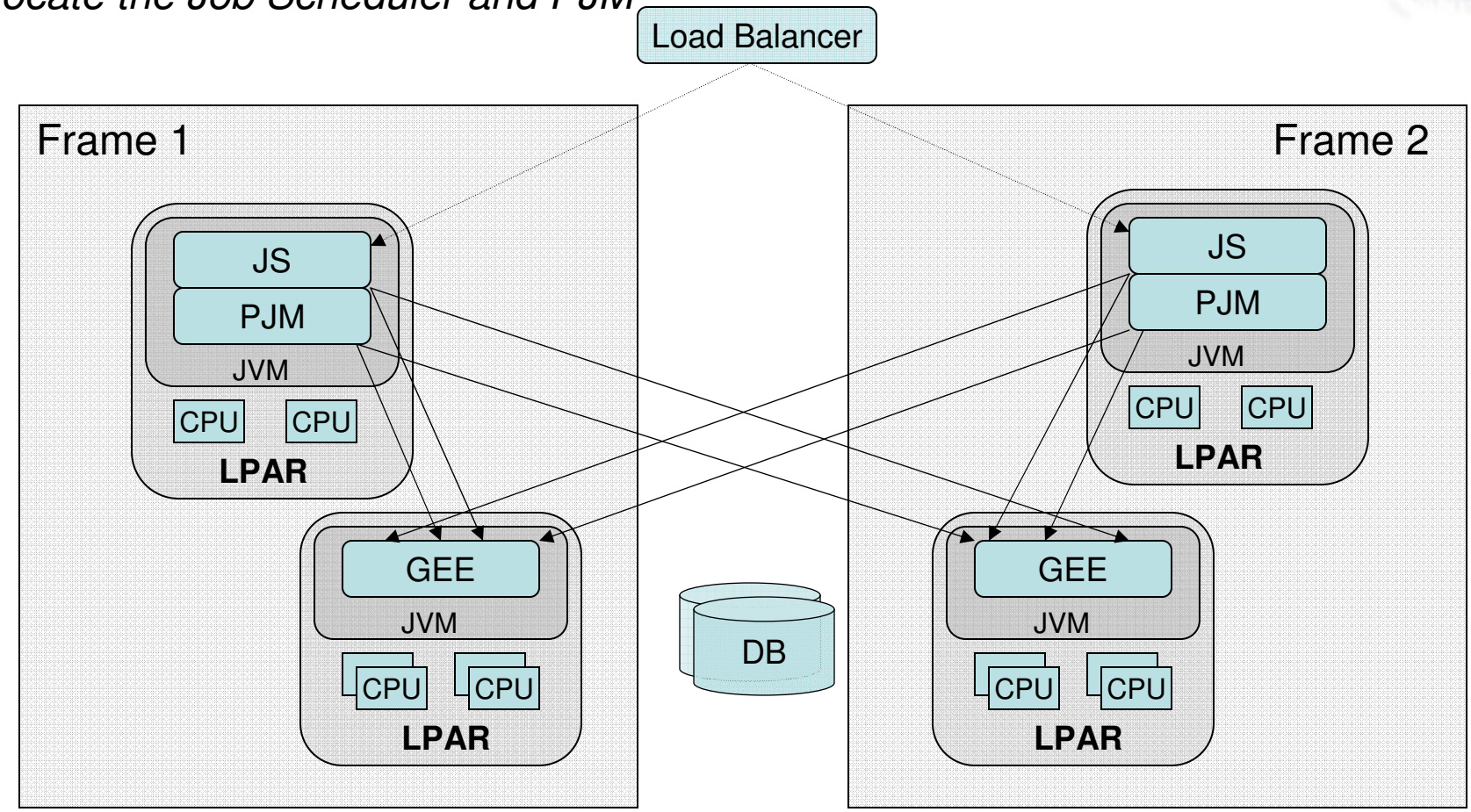
Example Production Topology-

Highly Available/Scalable Compute Grid



Example Production Topology-

Co-locate the Job Scheduler and PJM

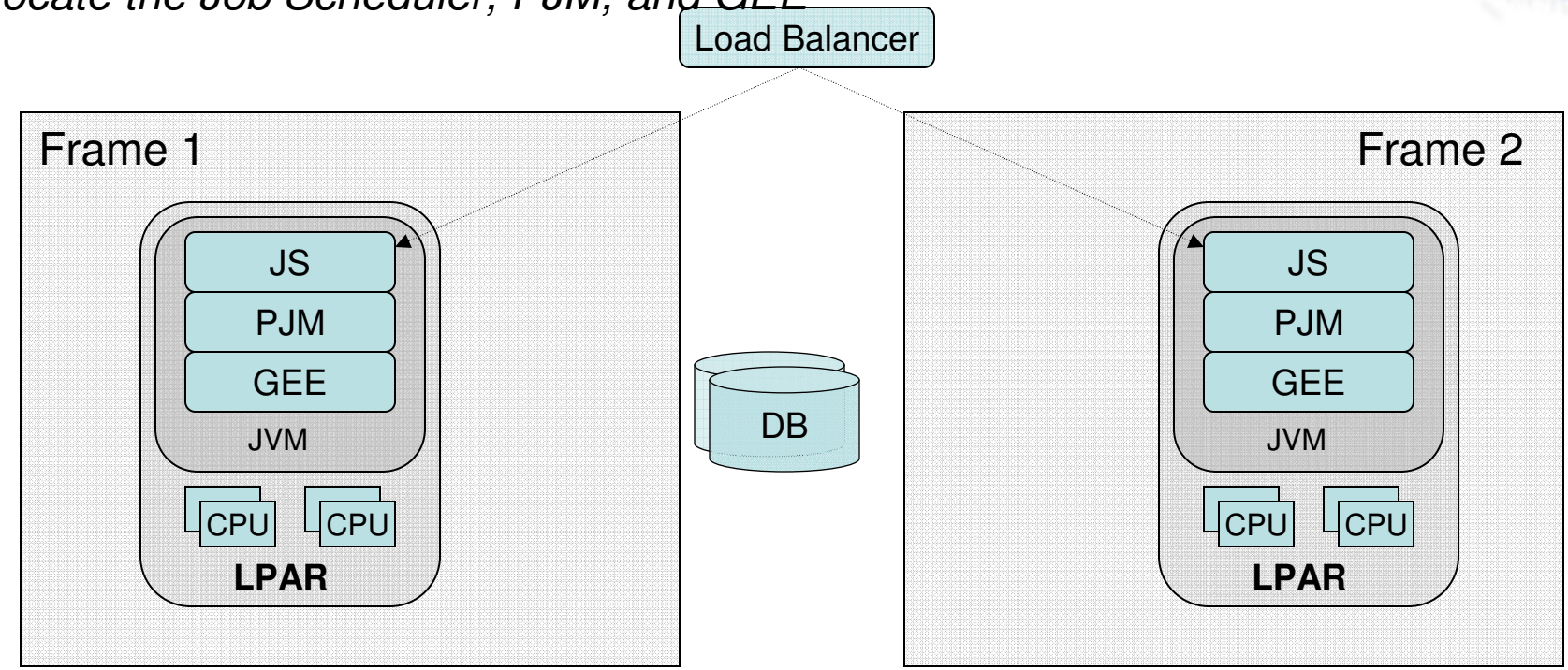


Pro: Faster interaction between JS and PJM due to co-location and ejb-local-home optimizations

Con: Possibility of starving JS or PJM due to workload fluctuations

Example Production Topology-

Co-locate the Job Scheduler, PJM, and GEE



Con: Possibility of starving JS, PJM, and GEE due to workload fluctuations

Con: Not scalable

High Availability – Summary & Key Considerations

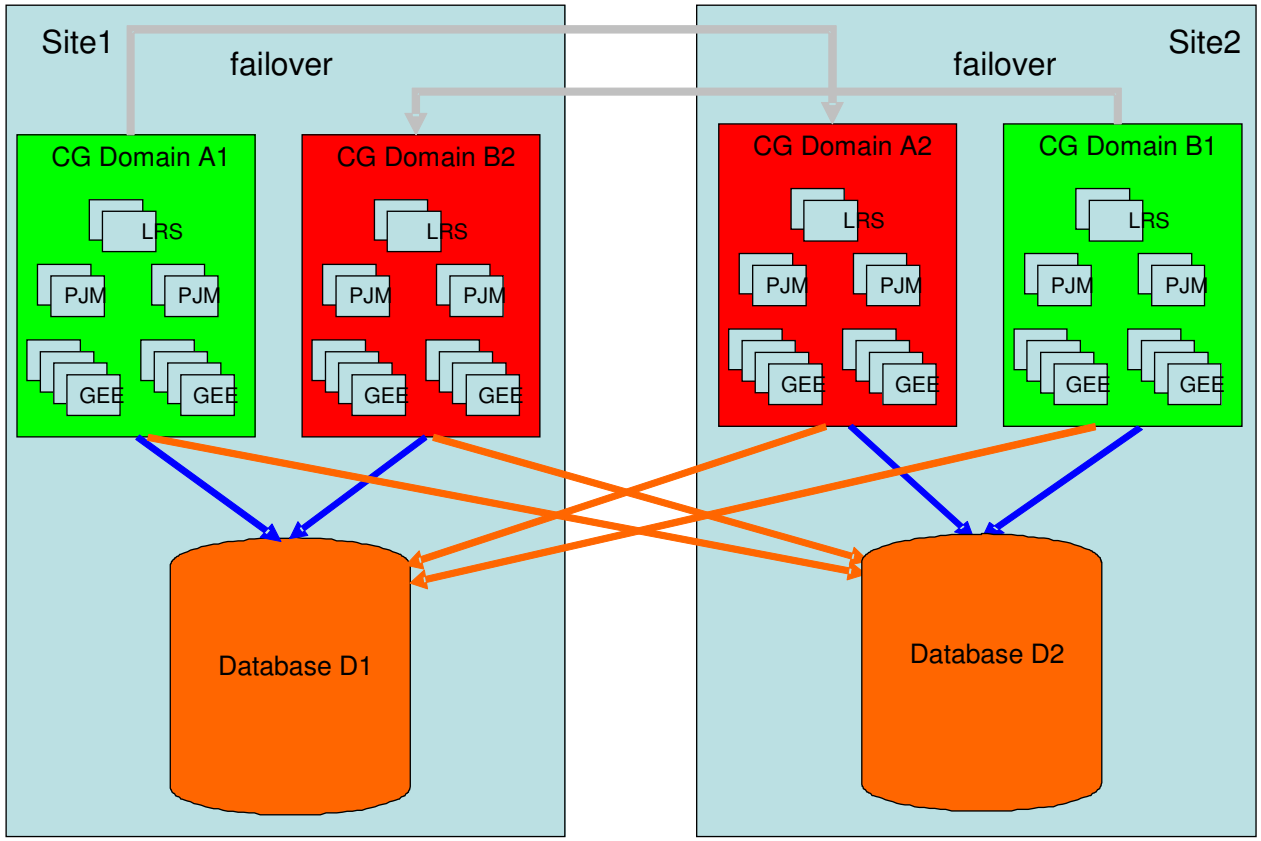
- Clustered Job Scheduler
  - Configure Job Schedulers on clusters
  - Multiple active Job Schedulers (since XD 6.1)
  - Jobs can be managed by any scheduler in your cluster
- Clustered Endpoints
  - Batch applications hosted on clusters
- Network Database
- Shared File System

Disaster Recovery

Disaster Recovery

- DR Topology
  - Build separate cells for geographically dispersed sites
  - Limit Compute Grid scheduling domains to endpoints within a cell
  - Use Active/Inactive DR domains
    - Jobs cannot be processed on primary and back domains simultaneously
  - Active/Active DR Topology is through a pair of Active/Inactive DR domains
    - Host backup (inactive) domain on a remote site
- DR Activation Process
  - Use CG provided DR scripts to prepare the inactive domain for takeover
  - Complete takeover by activating the inactive domain

Active/Active Multi-site Disaster Recovery Topology



The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.
Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM trademarks, see www.ibm.com/legal/copytrade.shtml
AIX, CICS, CICSPlex, DB2, DB2 Universal Database, i5/OS, IBM, the IBM logo, IMS, iSeries, Lotus, MQSeries, OMEGAMON, OS/390, Parallel Sysplex, pureXML, Rational, RACF, Redbooks, Sametime, Smart SOA, System i, System i5, System z , Tivoli, WebSphere, zSeries and z/OS.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.