IBM Software Group

Guide Share WebSphere – 14 février 2008

JMS overview

WebSphere software

Catherine Ezvan
IT Specialist
WebSphere Application Infrastructure

Introduction

- JMS is an API
  - Application Programming Interface to access an Enterprise Messaging System (or MOM)
  - To execute the programs, you need access to a vendor implementation of JMS.

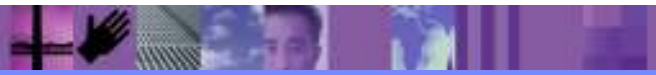
- “Enterprise Messaging”
  - Reliability – “once-and-once-only”
  - Integration – transformation of message formats, protocol bridging

- Java Message Service (JMS) is a standard enterprise messaging API
  - Defined by Sun, as part of J2EE
  - JMS interfaces supported since J2EE 1.2 (WAS v4.5)
  - JMS implementation required since J2EE 1.3 (WAS v5)
  - J2EE version 1.4 requires J2EE products to include a JMS version 1.1 provider that supports both point-to-point and publish/subscribe messaging.

Objectives of JMS

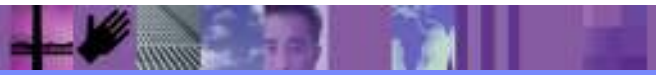
- The objectives of JMS, as stated in the specification, are to:
  - Define a common set of messaging concepts and facilities.
  - Minimize the concepts a programmer must learn to use enterprise
  - messaging.
  - Maximize the portability of messaging applications.
  - Minimize the work needed to implement a provider.
  - Provide client interfaces for both point-to-point and pub/sub *domains*.

"Domains" is the JMS term for the messaging models



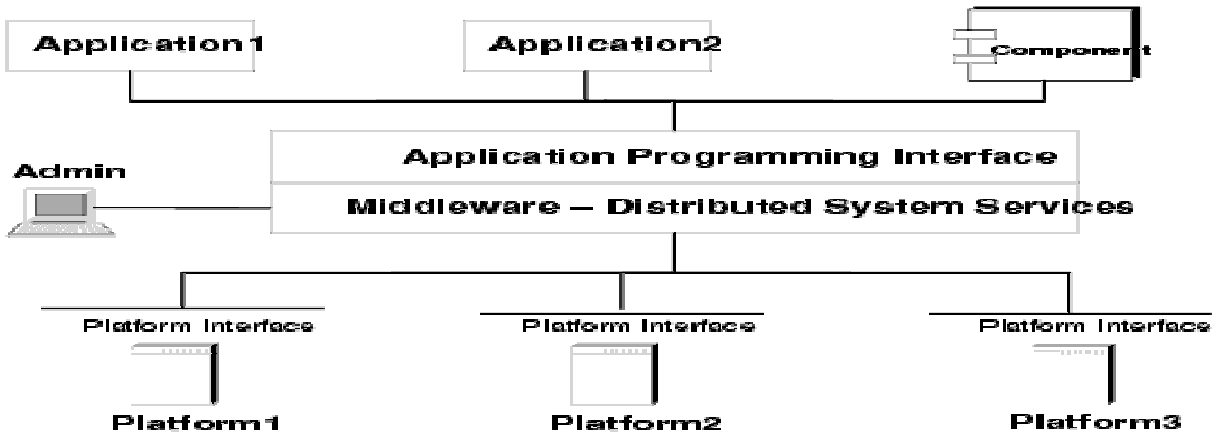
What JMS does not provide?

- The following features, common in MOM products, are not addressed by the JMS specification. Although acknowledged by the JMS authors as important for the development of robust messaging applications, these features are considered JMS provider-specific.
- JMS providers are free to implement these features in any manner they please, if at all:
  - Load balancing and fault tolerance
  - Error and advisory system messages and notification
  - Administration
  - Security
  - Wire protocol
  - Message type repository



Message Oriented Middleware (MoM)

- What Does MoM Provide?
  - Proper distribution of messages
  - Fault Tolerance
  - Scalability
  - Transaction Support
  - Asynchronous Delivery
  - De-coupling of the Sender and Receiver

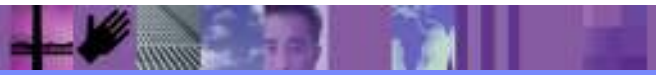


JMS – Peeling the first layer

- JMS Glossary
  - JMS Application – a system comprised of JMS clients and at least one JMS Provider
  - JMS Provider – the MOM with associated JMS classes
  - JMS client – messaging clients in a JMS environment
  - Producer – a client that produces and sends messages
  - Consumer – a client that receives and consumes messages
- Defines APIs for:
  - Point to Point - Queues
  - Publish/Subscribe - Topics
  - Five message classes
    - TextMessage, BytesMessage, ObjectMessage, StreamMessage, MapMessage
  - User defined message properties
  - Asynchronous message delivery

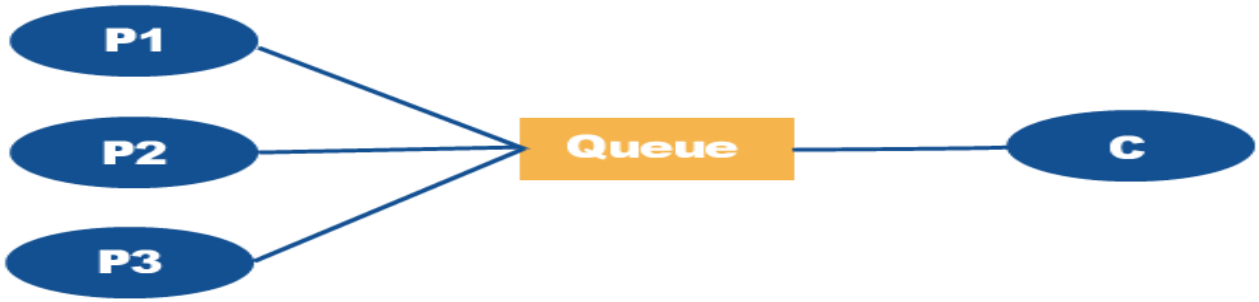
Administered Objects – JMS Business Cards

- Encapsulate vendor specific details
  - Aids application portability across JMS implementations
- Administered objects are stored externally and accessed by JNDI
  - allows application to be reconfigured without recompilation
- Two types of administered objects
  - Connection Factory
    - Describes how to connect to messaging infrastructure
  - Destination
    - Describes an end point for sending to or receiving from



Messaging - Point to Point

- Only one consumer may consume each message
- Pull Model
- Asynchronously *or "synchronously"*
- Allows consumers to browse a queue

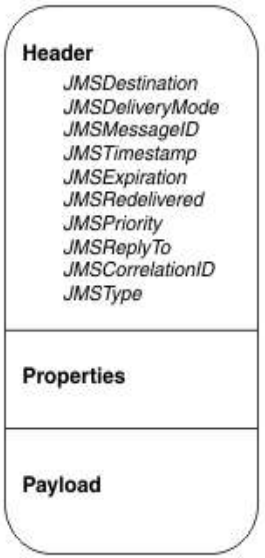


Messaging - Publication/Subscription

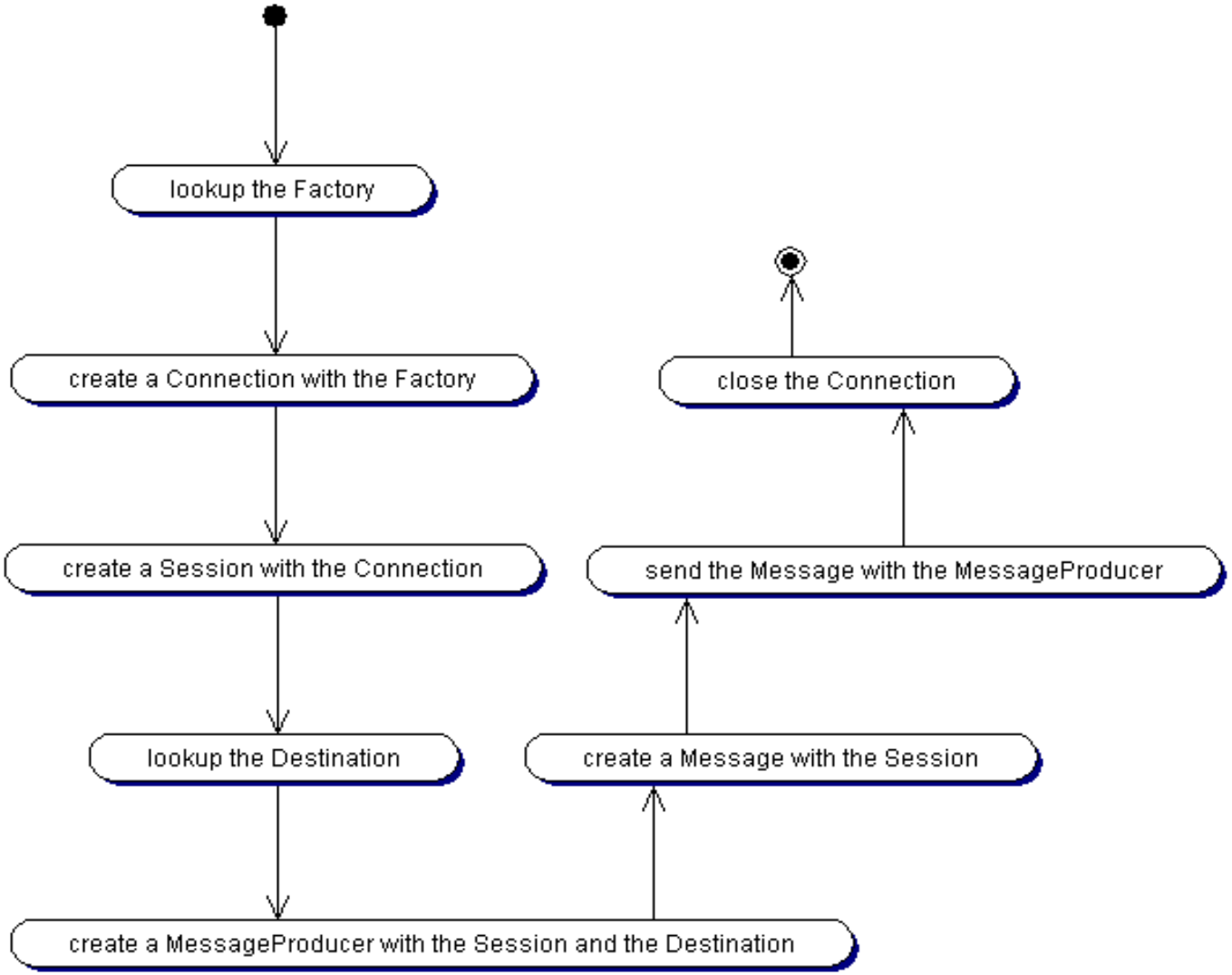
- Administrator creates a topic
- Producers publish through a virtual channel called a Topic
- Any number of consumers subscribe to the topic
- All subscribed consumers receive messages published to the topic (uses Broker to distribute the messages)
- Durable subscriptions
- Push model
- Asynchronously *or "Synchronously"*

JMS Message - Anatomy

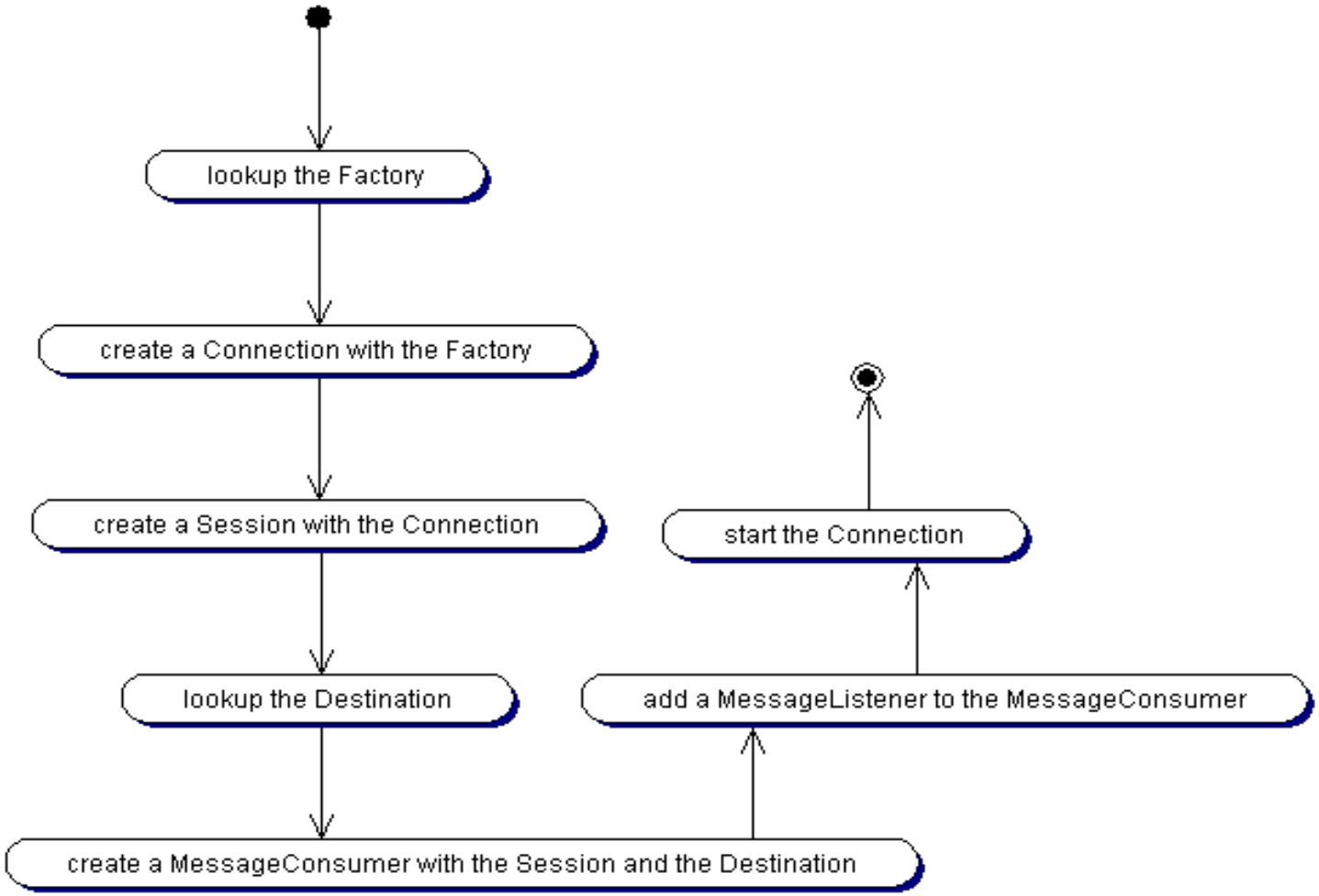
- Headers
  - JMSDestination
  - JMSDeliveryMode
  - JMSMessageID
  - JMSTimeStamp
  - JMSCorrelationID
  - JMSReplyTo
  - JMSRedelivered
  - JMSType
  - JMSExpiration
  - JMSPriority
- Properties
  - JMSXUserID
  - JMSXAppliID
  - JMSXDeliveryCount
  - MSXGroupID
  - etc
- Payload



Generic Programming Model – JMS Producer

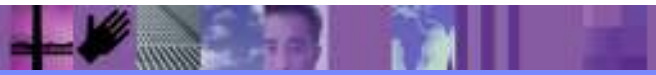


Generic Programming Model – JMS Consumer

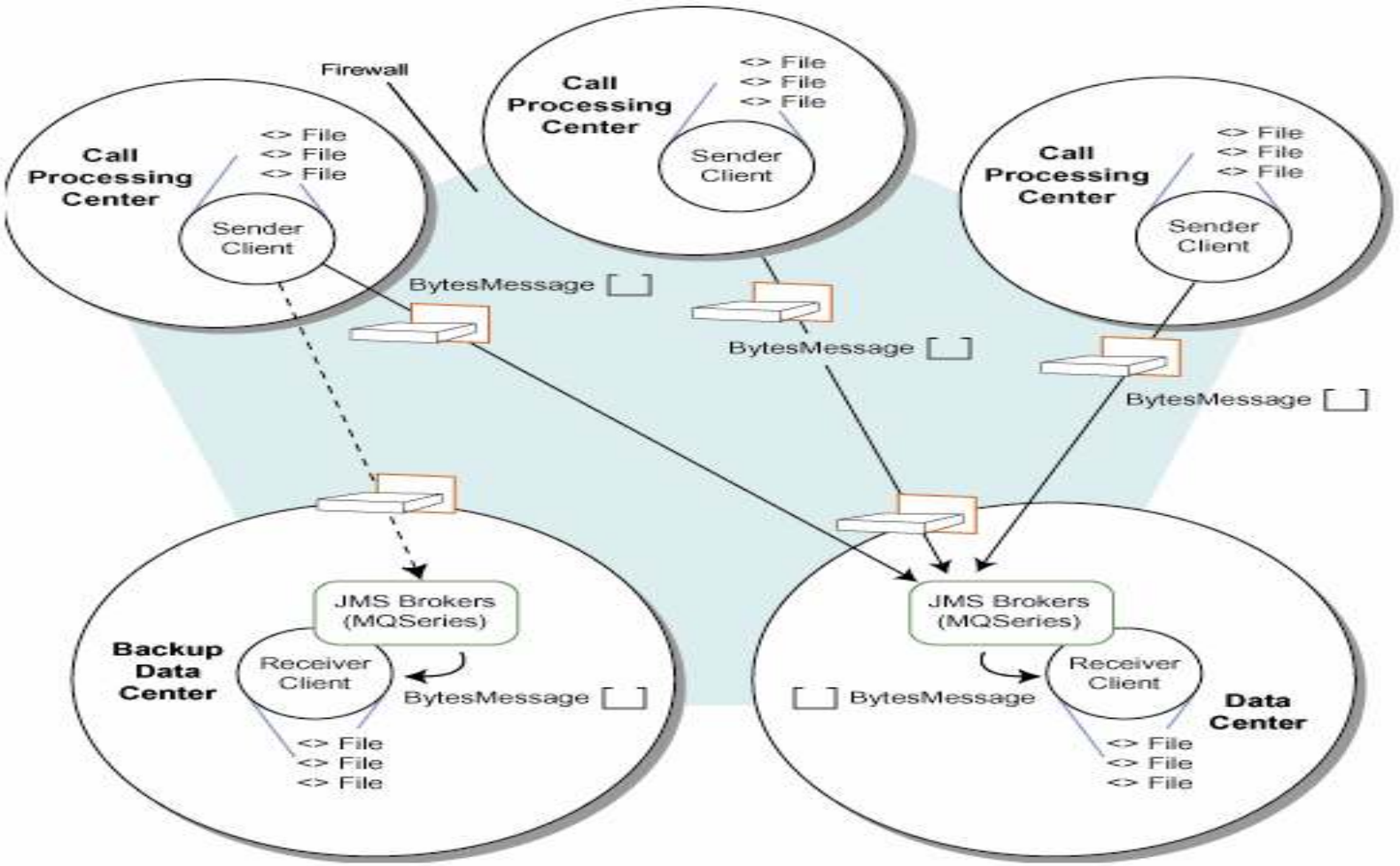


JMS-based solution Example #1 / Point to Point - Context

- Customer
  - Applied Reasoning customer
- The goal
  - To implement a file distribution and replication solution
  - Not only did JMS provide the necessary infrastructure for transferring information, but it also handled all of the infrastructure issues related to quality of service, security, reliability, and performance that the customer required.
- The problem
  - A significant distributed data challenge
    - A number of call centers around the country where operators record interactions with customers
    - The recordings had to be quickly and reliably indexed and archived in remote data centers
    - The existing solution was falling far short of the performance and reliability goals

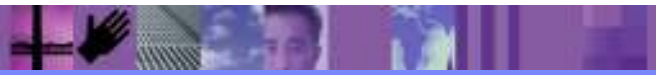


JMS-based solution Example #1 / Point to Point - Solution



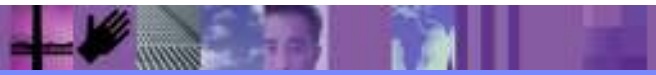
JMS-based solution Example #1 / Point to Point

- The JMS-based system
  - Provides reliable archiving for recorded multimedia files
  - Allows extendability to let multiple data centers receive the files
  - Allows additional data types to be archived.
- It was important to the client to limit vendor lock-in
- Additional elements (message compression, etc) were easy to add because of the open model JMS provides
- The entire system took six weeks to build and quickly replaced the existing, labor-intensive one the customer had been using
- JMS is a viable solution not only for small, message-oriented applications, but also for large-scale, mission-critical data transfer operations



JMS-based solution Example #2 / Pub/Sub - Context

- WebSphere eXtended Deployment Object Grid
  - ObjectGrid is a grid-enabled, in-memory, Java database
  - Supports grid-based data centric transaction processing
  - Designed to overcome pitfalls with conventional database architectures
    - Alternative architecture for very high performance applications
  - Implements persistence using policy based
    - Memory replication
    - Data placement
  - Provides consistent millisecond response times as the system scales up
- The problem
  - Local cache copies must be kept consistent across replicas

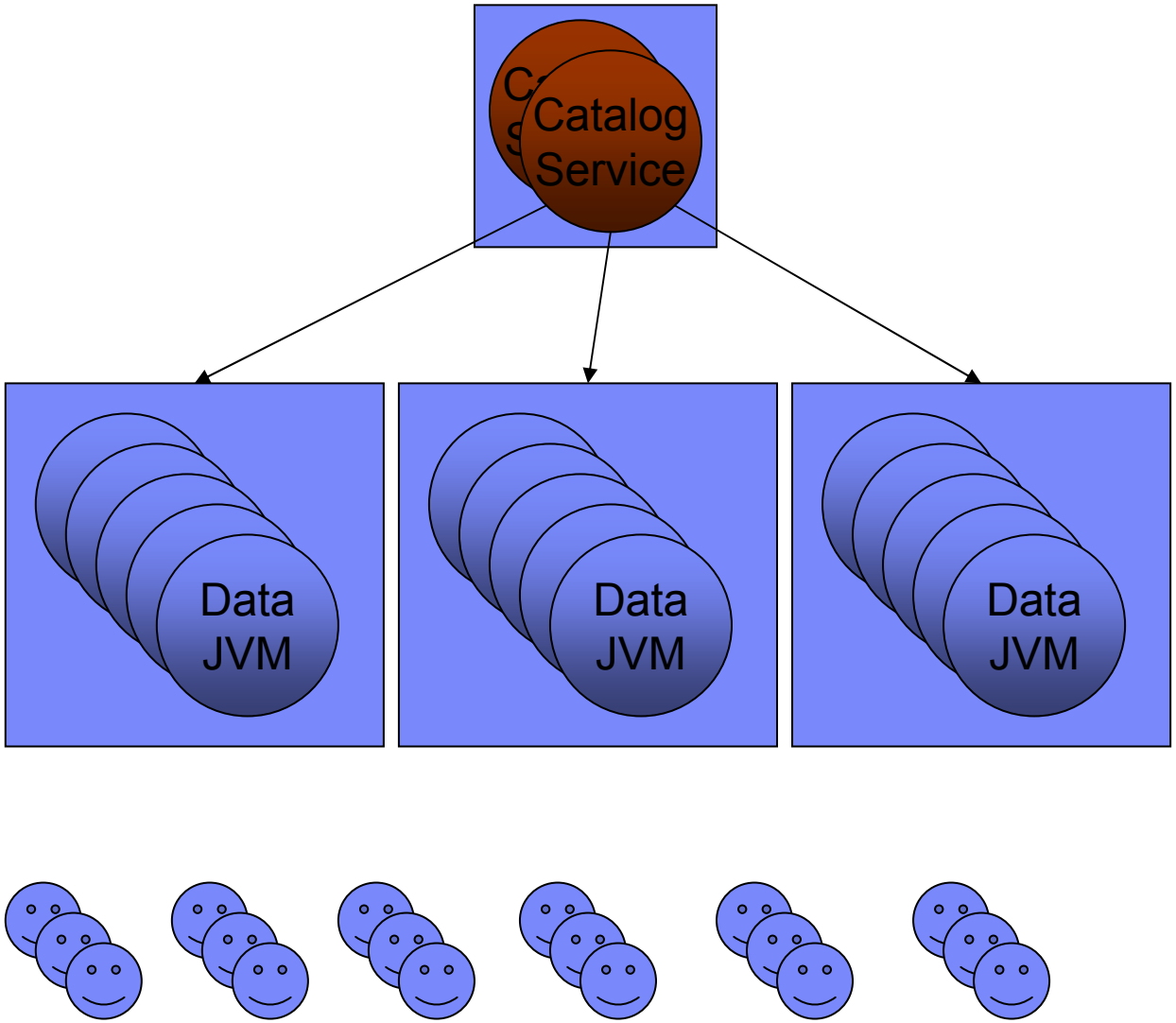


JMS-based solution Example #2 / Pub/Sub - Architecture

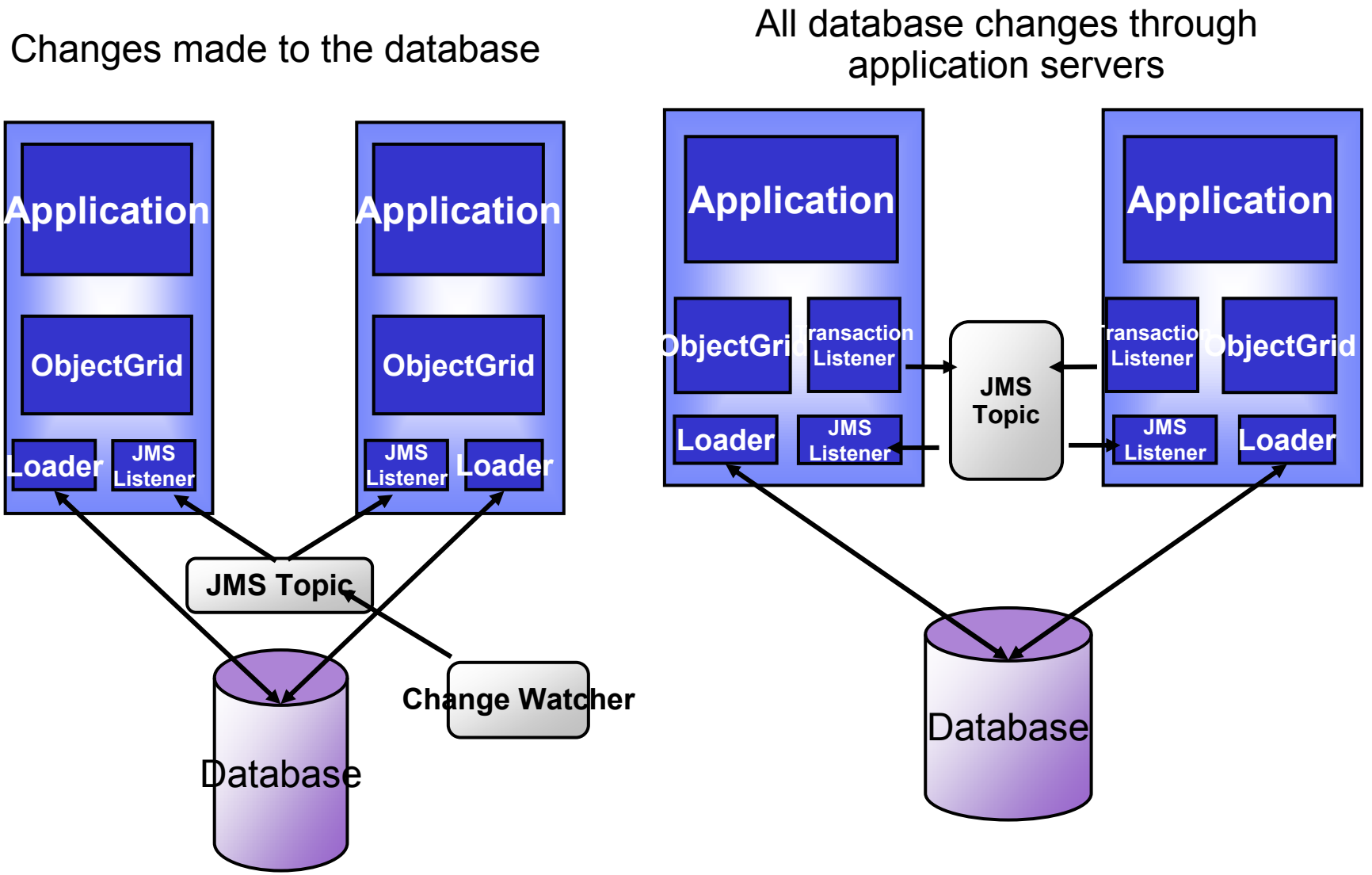
Fault tolerant catalog service
- Federates and manages server JVMs across cells/coregroups into one ObjectGrid
- Ensures the fault tolerance and allows clients to bootstrap to the correct JVMS.

Multiple cells with dynamic core groups, designed to **scale to 1000s of servers** hosting the data.

Clients using the ObjectGrid



JMS-based solution Example #2 / Pub/Sub - Solution



Références

- Introducing the Java Message Service
  - (DeveloperWorks) http://www.ibm.com/developerworks/edu/j-dw-jms-i.html
- Get the message? Using JMS as a data replication solution
  - (DeveloperWorks) http://www.ibm.com/developerworks/library/i-jms/