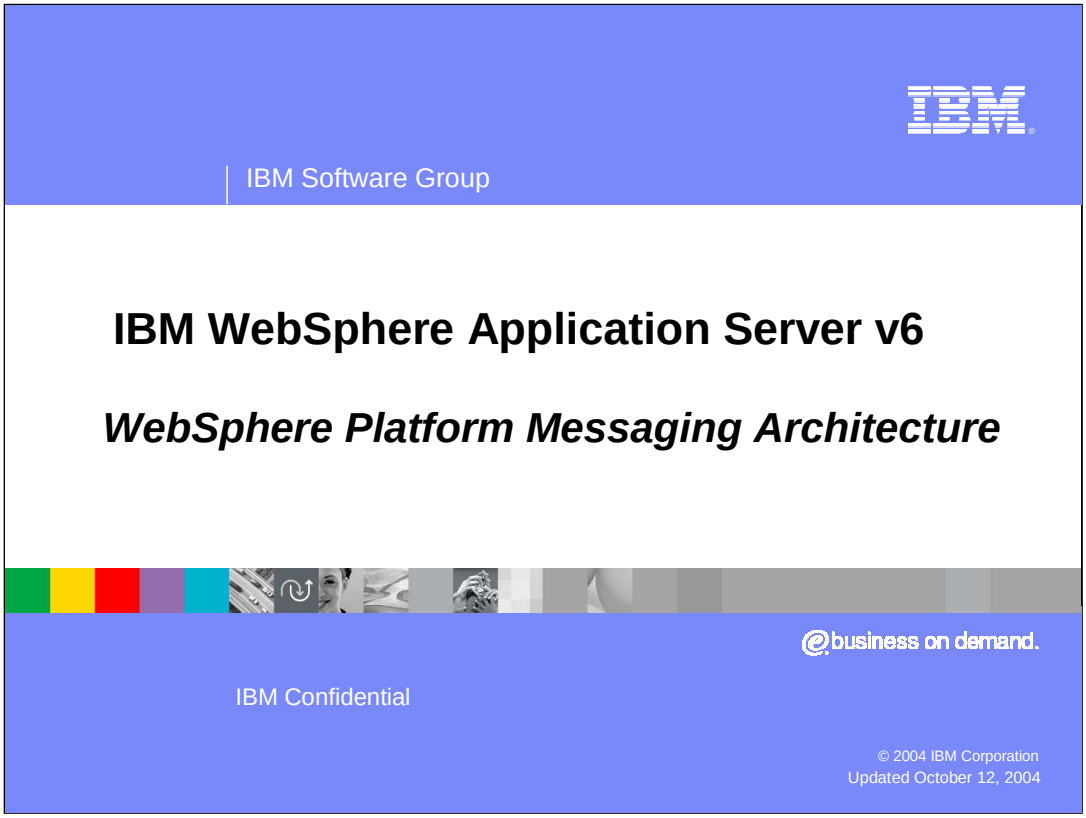

IBM Software Group
IBM WebSphere Application Server v6
WebSphere Platform Messaging Architecture
e business on demand.
IBM Confidential
© 2004 IBM Corporation
Updated October 12, 2004

Agenda

- Platform messaging architecture
  - Service Integration Bus
  - Bus Member
  - Messaging Engine
  - Destinations
  - Message Store
- Mediation Support
- High Availability and Scalability
- Summary

Section

Overview

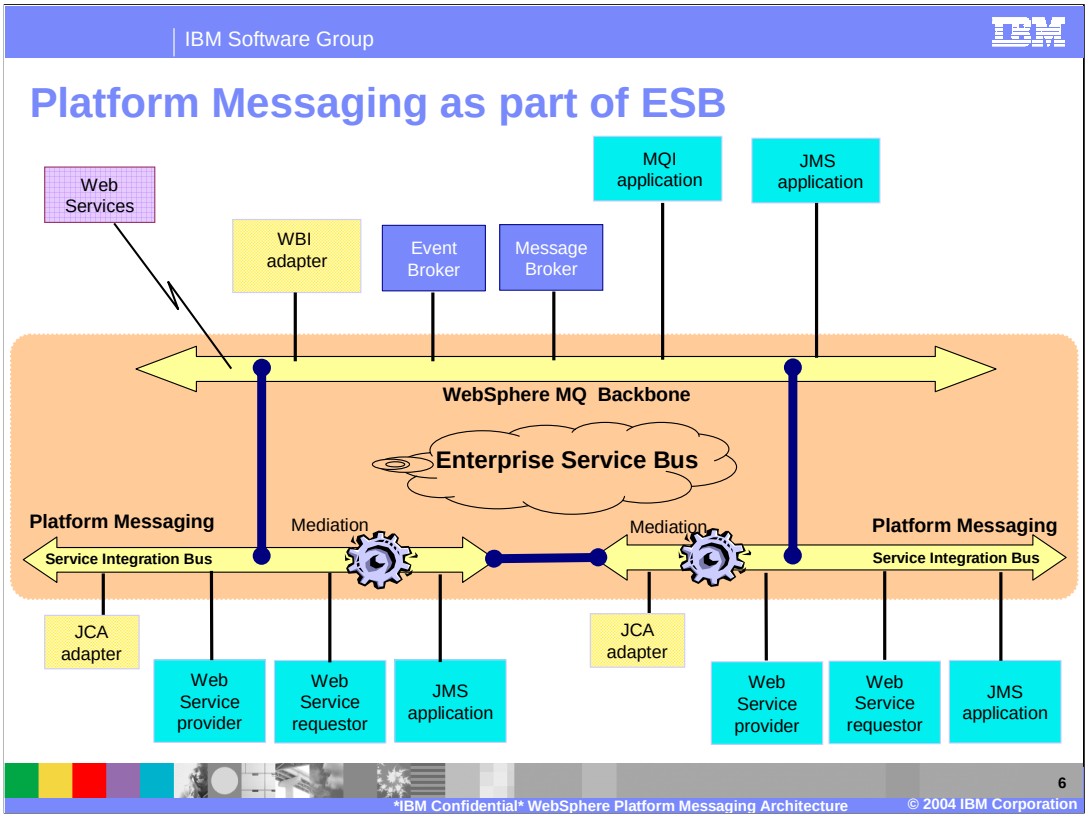
WebSphere Platform Messaging

- WebSphere Application Server Platform Messaging consists of Service Integration Bus (SIBus) which provides the framework needed to support SOA
- Its an important component of the Enterprise Service Bus (ESB) concept
- Even though JMS Messaging is based on platform messaging, platform messaging is more than JMS messaging

The service integration functionality within WebSphere Application Server provides a highly-flexible messaging fabric that supports a service-oriented architecture with a wide spectrum of quality of service options, supported protocols, and messaging patterns. It supports both message-oriented and service-oriented applications.

WebSphere Platform Messaging: Big Picture

- Integrated asynchronous messaging capability for the WebSphere Platform
  - Built-in messaging service for WebSphere Application Server
  - Fully compliant JMS 1.1 provider
- Service Integration Bus
  - WebSphere component that implements bus-style (interconnected) messaging
- Complements and extends WebSphere MQ



The existing WebSphere messaging products provide a comprehensive range of functions which enable customers to build an ESB today

This will be augmented by the new Platform Messaging which provides greater support for J2EE and WebServices standards

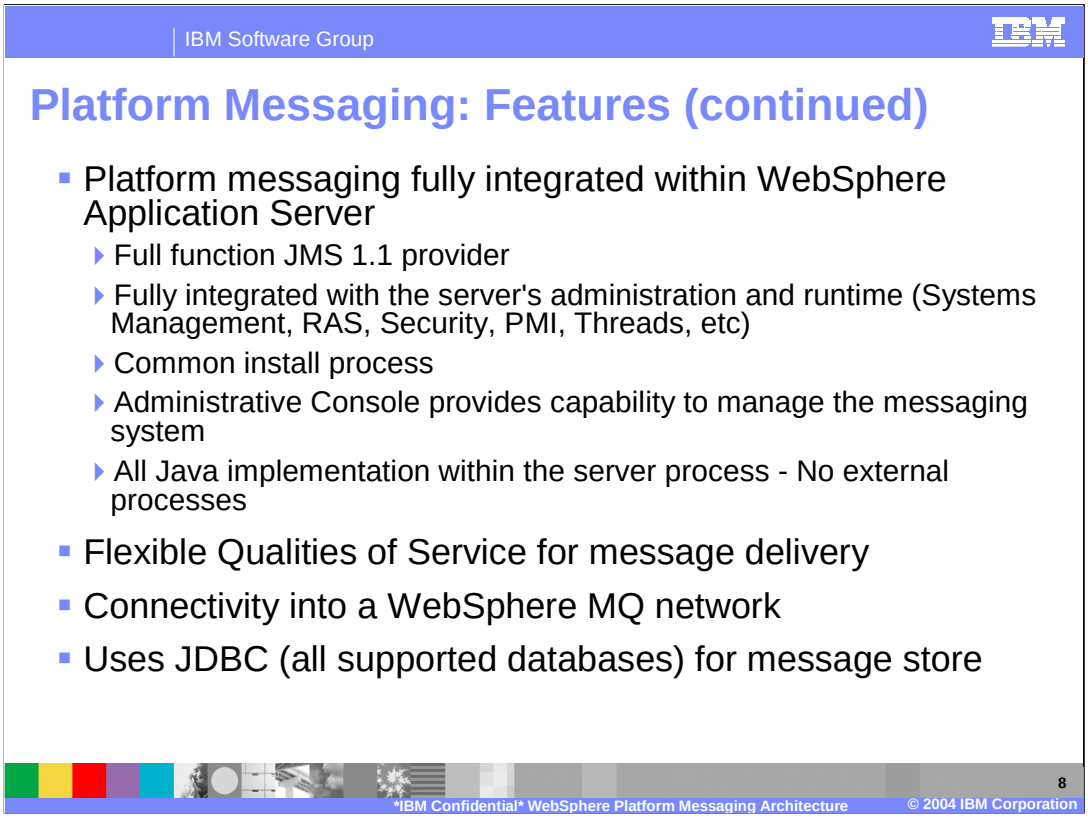
Platform Messaging provides a more flexible and integrated messaging solution for the Application Server with connectivity onto the ESB

Platform Messaging: Features

- Multiple messaging patterns (APIs) and protocols - message-oriented and service-oriented applications
  - Default messaging provider as a J2EE 1.4 compliant JMS provider, supporting JMS APIs
  - Implementation of relevant Web Services standards, supporting JAX-RPC APIs
- Reliable message transport capability
- Intermediary logic (*mediations*) to intelligently adapt message flow in the network
- Clustering enablement for scalability and High Availability (HA)

Different quality of service options are supported

Support for the WebSphere Business Integration programming model, which converges functions from workflow, message brokering, collaborations, adaptors and the Application Server



Improved performance for in-process messaging

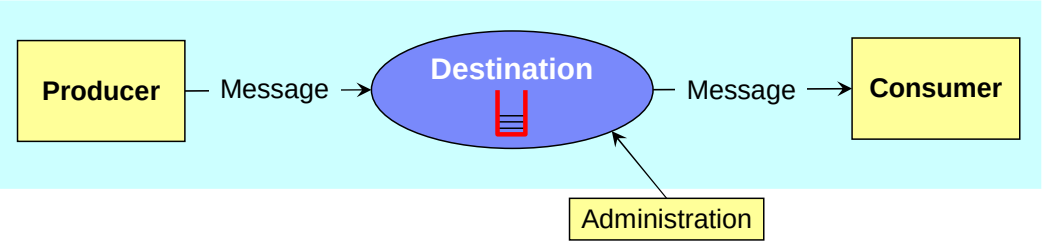
Messages traveling between applications running in the application server process can be passed in memory

Section

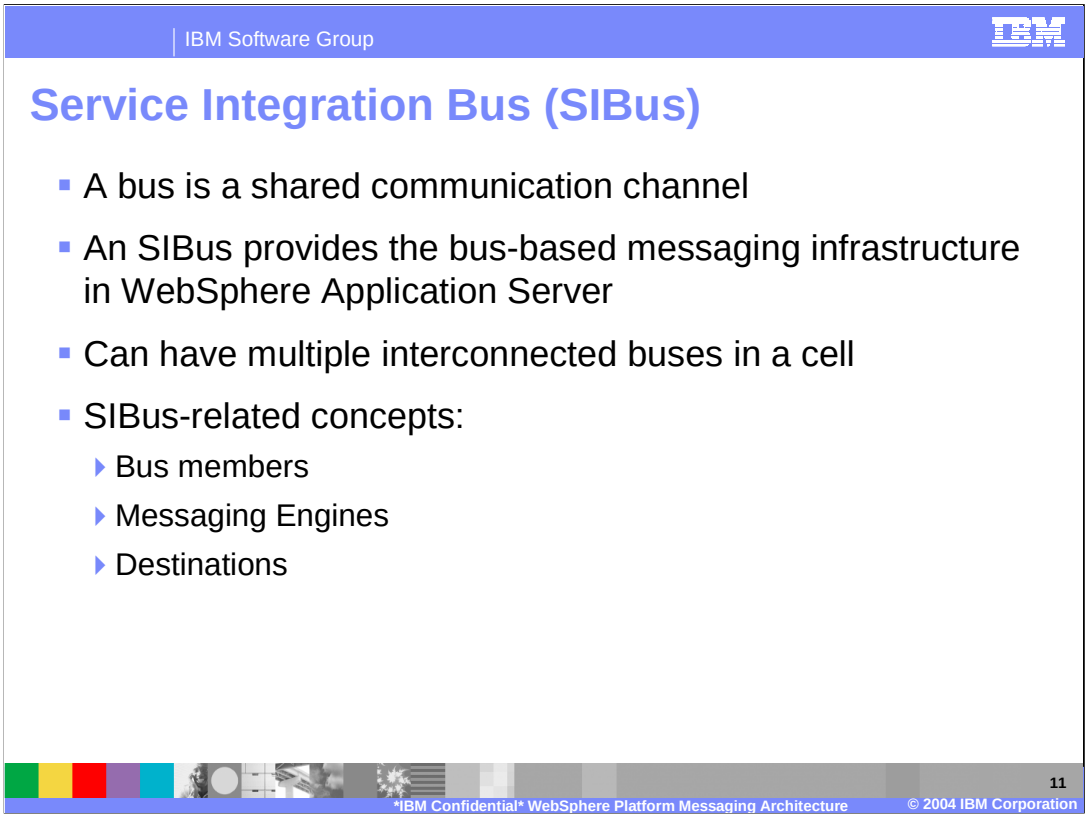
Service Integration Bus (SIBus)

Architecture and Components

Messaging: Basic Flow



- Producers send/put messages to destinations
- Consumers receive/get messages from destinations
- Destinations are Platform Messaging points of communication
  - Example: JMS Queues, topics, Web Service endpoints



The following capabilities are provided by the bus:

•Any application can exchange messages with any other application by using a *destination* to which one application sends, and from which the other application receives.

•A message-producing application, that is, a *producer*, can produce messages for a destination regardless of which messaging engine the producer uses to connect to the bus.

•A message-consuming application, that is, a *consumer*, can consume messages from a destination (whenever that destination is available) regardless of which messaging engine the consumer uses to connect to the bus

The bus supports the following types of messaging:

•Sending messages synchronously (this requires the consuming application to be running and reachable)

•Sending messages asynchronously (possible whether the consuming application is running or not and whether or not the destination is reachable). Both point-to-point and publish/subscribe messaging are supported

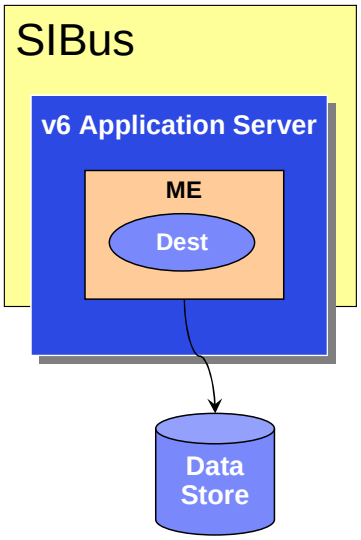
•Publishing events or other notifications. Notification messages can also be generated by the bus itself

Bus Member

- Bus Members are definitions that associate a cluster (or standalone server) with an SIBus
- When a new Bus Member is defined, one Messaging Engine (ME) is automatically created on the corresponding Cluster

Messaging Engine (ME)

- ME runs inside an Application Server and manages messaging resources
- Destinations are defined on an ME
- Each cluster has an ME for each SIBus it is associated with
- The ME has an associated data store for message persistence and overflow



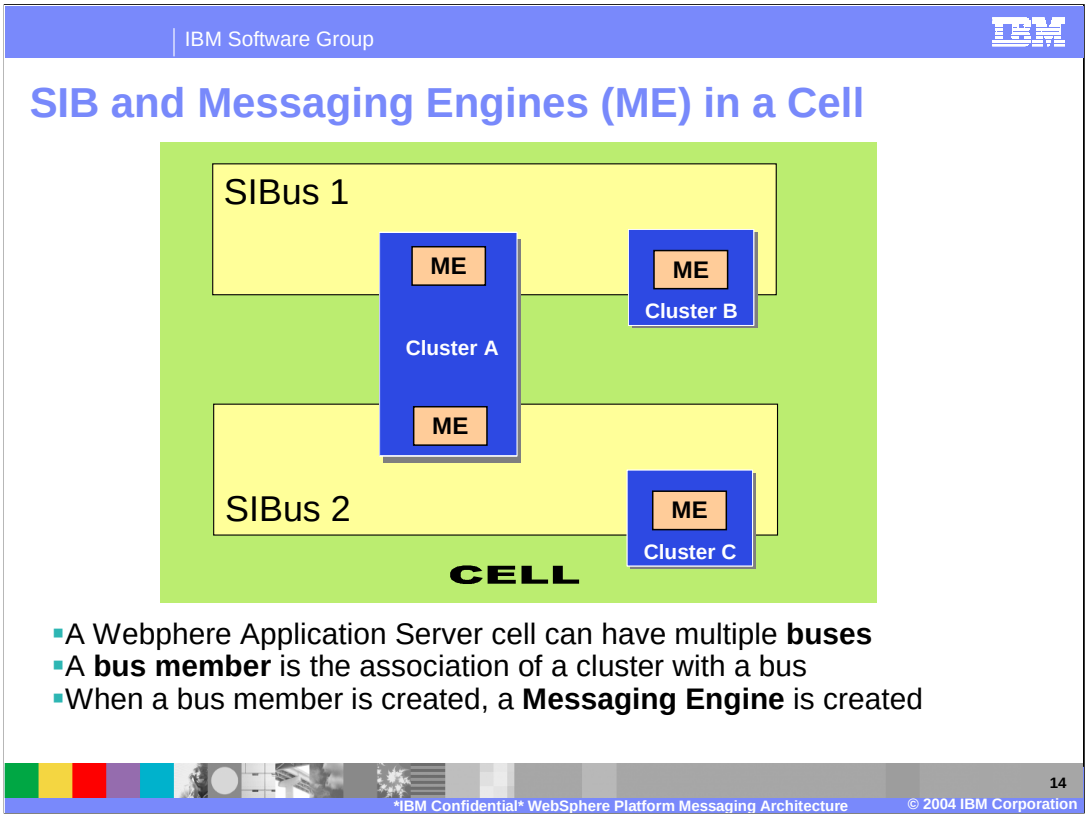
One ME is automatically created for the application server or the cluster when defining a new bus member (Application Server or Cluster)

Can have multiple MEs running within the same cluster

A default bus with a single default ME is created automatically at installation time of the Standalone Application Server

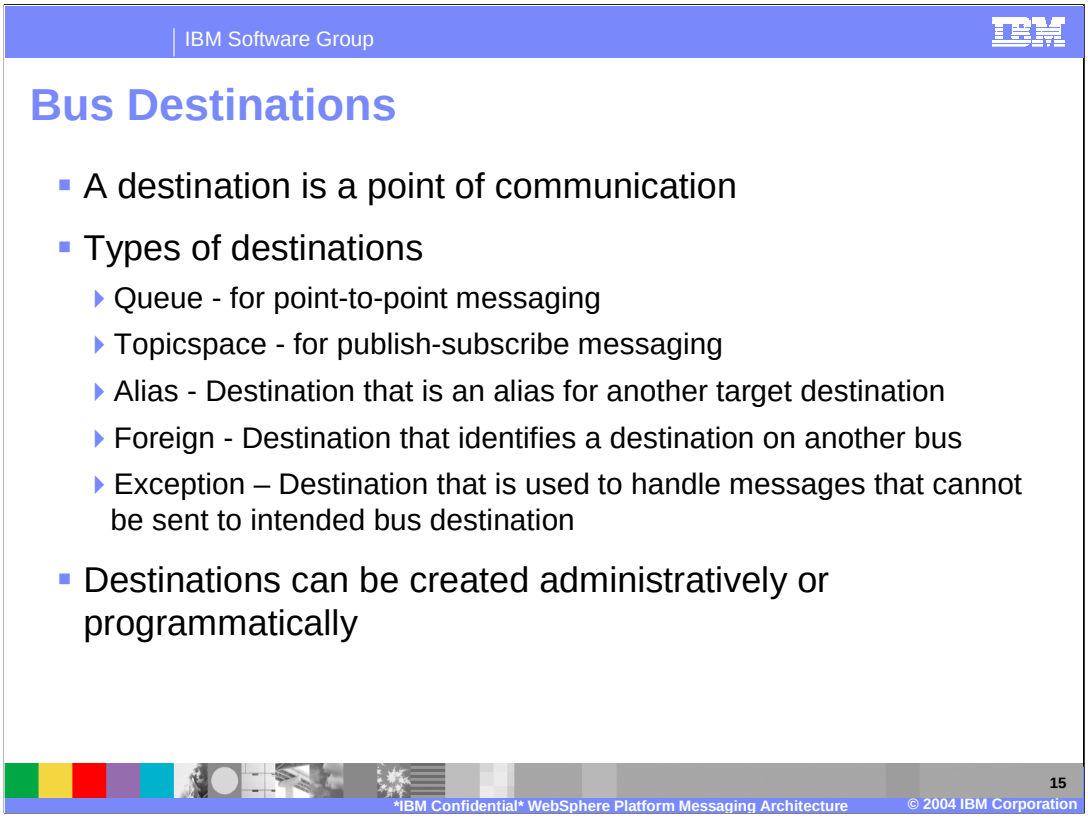
Within a bus, each ME has a unique identity - made up of bus name and server name

The data store preserves messages, subscriptions, and so on, so that they survive if the server or messaging engine is stopped and restarted. It is also used for the overflow of the non-persistent messages in some QOS options



A typical WebSphere Application Server configuration is to define a single bus for each WebSphere cell and to run a default ME in each application server that is a member of this bus. This serves as the embedded JMS provider, and provides SI bus capabilities for Web Services applications

Messaging Engines run inside an application server, alongside J2EE applications and their containers. You can have multiple MEs running in the same Application Server. This is to allow the processing related to different sets of messaging applications running within the process to be separated, and for the different applications to be using different buses, each with their own topology and set of resources. There is no architectural requirement for MEs within the same process to be members of different buses, it is possible to have two MEs that are part of the same bus both running in the same process.



Temporary destinations are bus destinations that are created and deleted automatically for API-specific temporary destinations. For example, if an application creates a temporary JMS queue for use with the default messaging provider, which is based on service integration technologies, the SIB service automatically creates a temporary destination on the bus. Temporary destinations appear on the list of destinations for a service integration bus, but normally need no administration.

A topicspace is a hierarchy of topics used for publish/subscribe messaging. Topics with the same name can exist in multiple topicspaces.

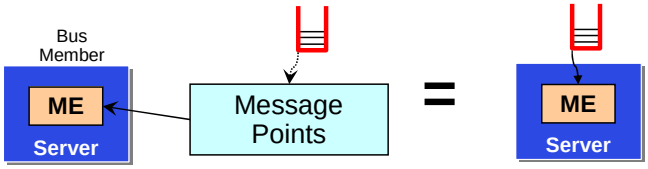
Alias destinations provide a level of abstraction between applications and the underlying target bus destinations that hold messages. Applications interact with the alias destination, so the target bus destination can be changed without changing the application. Each alias destination identifies a target bus destination and target service integration bus. Applications can use an alias destination to route messages to a target destination in the same bus or to another (foreign) bus (including across an MQLink to a queue provided by WebSphere MQ).

Each *messaging engine* has a default exception destination, SYSTEM.DEFAULT.EXCEPTION.DESTINATION.*me_name*, that can be used to handle undeliverable messages for all bus destinations that are localized to the messaging engine

The foreign destination identifies a destination on another bus, and enables applications on one bus to access directly the destination on another bus

Linking Destinations to Bus Members

- Bus destinations are the points of communication for bus members
  - A destination is associated with one Bus Member



Messages are transported through a 'Bus' (e.g. an SIBus). A Bus is a virtual messaging environment that spans Servers. If you want to send or receive messages through a Bus you have to connect to it. Servers (e.g. Server1) can be added to a bus, and in doing so you will start/create a Messaging Engine (ME) in that Server.

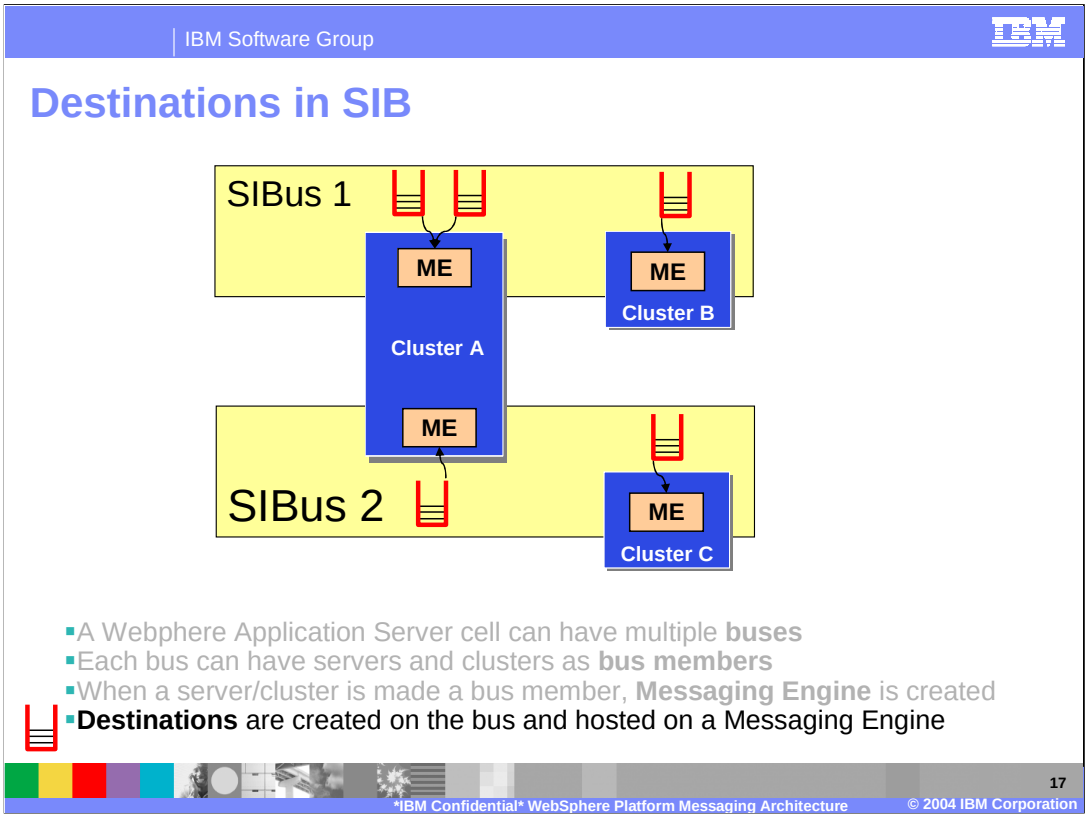
SIBDestinations are defined on the Bus (rather than on an ME). Queues will be localised to a particular ME (e.g. messages sent to that Queue will be stored in a database on one particular ME within the Bus), where as TopicSpaces (groups of Topics) are localised on every ME

A message point is the general term for the location on a messaging engine where messages are held for a bus destination.

For point-to-point messaging, the administrator selects one bus member, an application server or cluster, as the assigned bus member that is to implement the runtime state of a queue destination. This action automatically defines a queue point for each messaging engine in the assigned bus member.

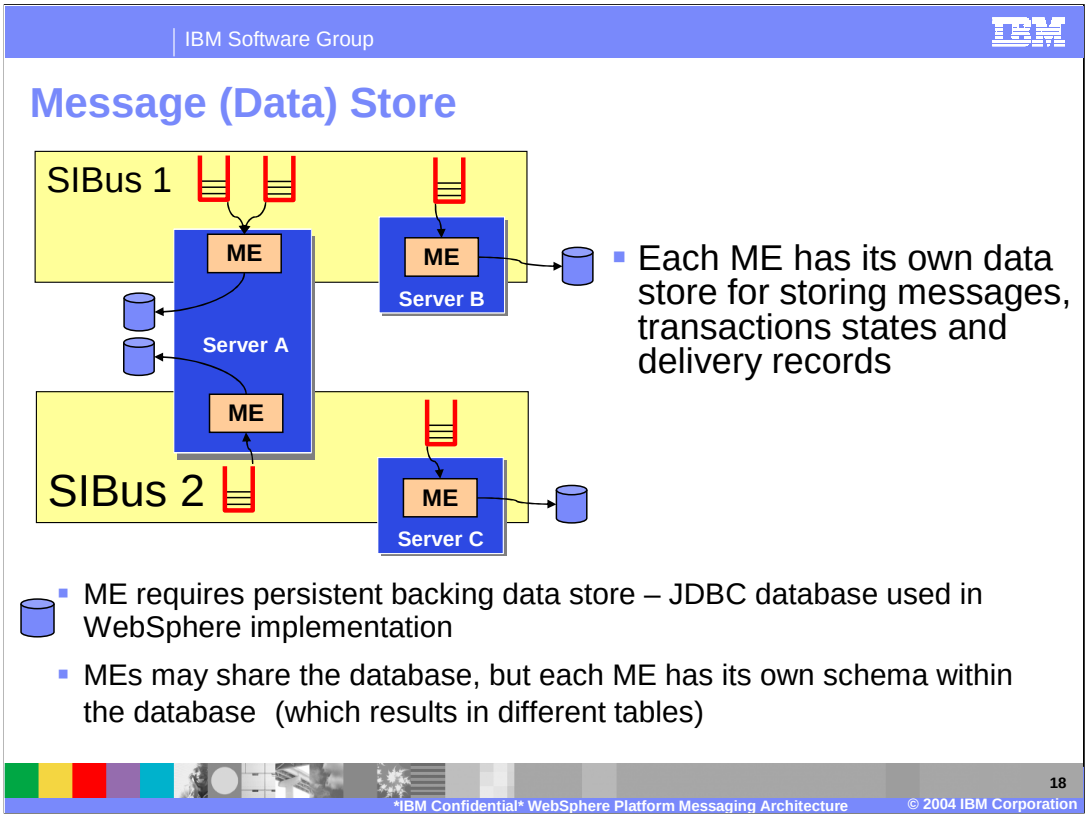
For a queue destination assigned to an application server, all messages sent to that destination are handled by the messaging engine of that server, and message order is preserved.

For publish/subscribe messaging, the administrator configures the destination as a topic space (a hierarchy of topics), but does not need to select any assigned bus member for the topicspace. A topic space has a publication point defined automatically for each messaging engine in the bus



A typical WebSphere Application Server configuration is to define a single bus for each WebSphere cell and to run a default ME in each application server that is a member of this bus. This serves as the embedded JMS provider, and provides SI bus capabilities for Web Services applications

Messaging Engines run inside an application server, alongside J2EE applications and their containers. You can have multiple MEs running in the same Application Server. This is to allow the processing related to different sets of messaging applications running within the process to be separated, and for the different applications to be using different buses, each with their own topology and set of resources. There is no architectural requirement for MEs within the same process to be members of different buses, it is possible to have two MEs that are part of the same bus both running in the same process. However this is not a recommended configuration



The Message Store is a sub-component of the messaging engine. This is used to buffer in-flight messages and hold a number of other pieces of information (for example records of message delivery when delivering multiple copies of a single message).

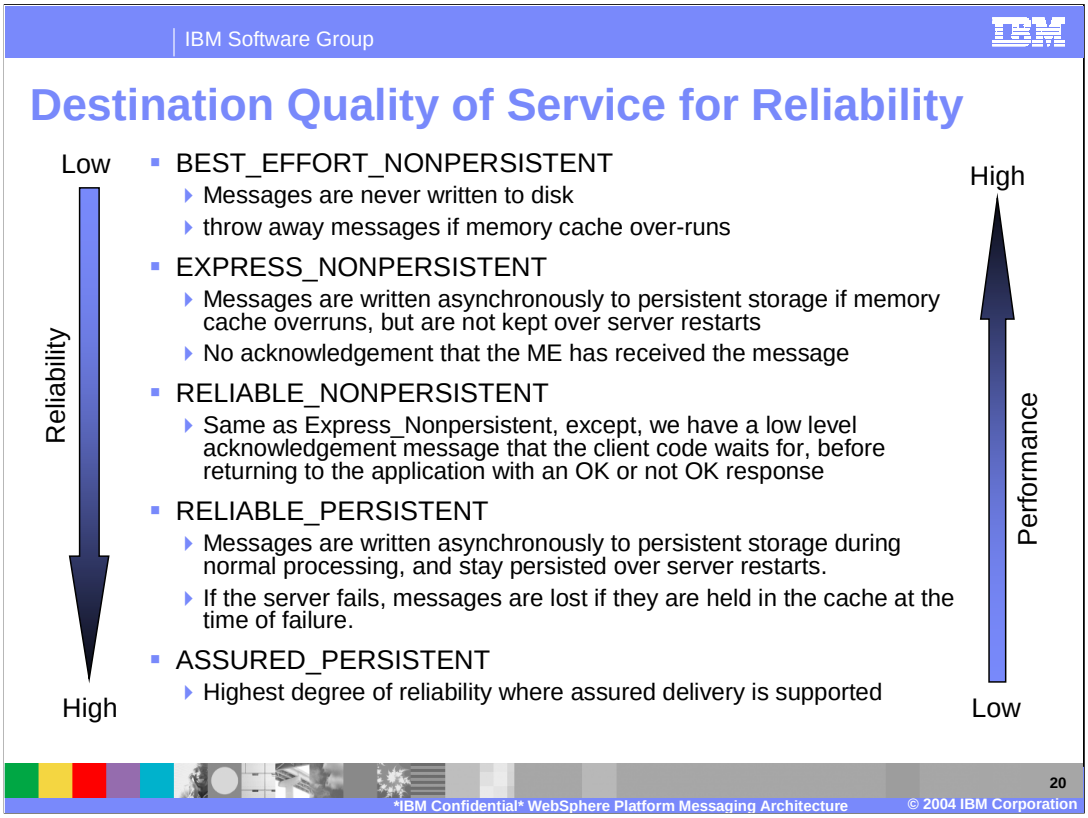
A Message Store can be either **Persistent** or **Non-Persistent**. A persistent message store is capable of holding both persistent objects (state which survives after an engine stops for whatever reason) and volatile objects (state which does not survive an ME failure, and may or may not survive an orderly shutdown of the messaging engine). In contrast a Non-Persistent Message Store can only hold volatile objects, it cannot hold persistent objects.

ME requires a persistent back end data store, even for non-persistent messages (example, for spilling).

Out of the box support for persistence using Cloudscape. Support for DB2, Oracle, Sybase etc. to enable customers to use an RDBMS of choice. These databases should also offer higher performance than Cloudscape.

Quality of Service (QOS) specification

- QOS can be specified on
  - A Destination
    - Maximum QOS can be defined on a Destination (e.g. this Destination can only accept non-persistent messages)
  - A message
    - By individual Producers and Consumers, typically under application control via a combination of API calls (e.g. JMS Persistent or Non-persistent) and the Connection Factory used (which defines the Persistent/NonPersistent Mapping)



Express non-persistent and reliable-persistent are defaults for non-persistent and persistent.

In express non-persistent, messages are sent from a producer to the ME, but there is never an acknowledgement flow (at the low level communications layer) to indicate that the ME has the message. We return to the application immediately and assume that all is well. In the reliable non-persistent we have a low level acknowledgement message that the client code waits for before returning to the application with an OK, Not OK response. So - express runs faster, but with a slightly lower level of reliability

Section

Platform Messaging

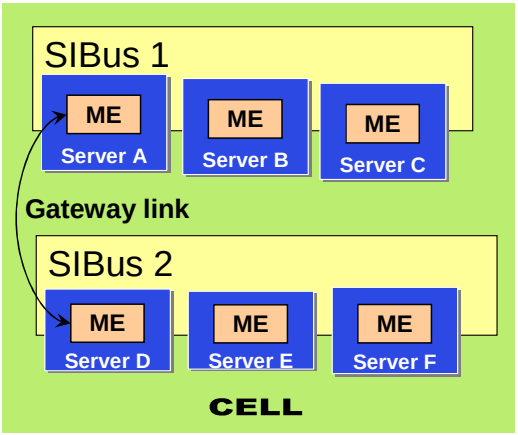
Topologies

Messaging Engine topology

- One messaging engine per bus member is adequate for most applications
  - This is the default topology
- Advantages of deploying more than one messaging engine, and linking them together
  - Spreading messaging workload across multiple servers
  - Placing message processing close to the applications that are using it
  - Accommodating firewalls or other network restrictions that limit the ability of network hosts all to connect to a single messaging engine

Bus Topology

- An enterprise might deploy multiple interconnected messaging buses for organizational reasons
  - e.g., Separately administered buses for each department
- A bus can connect to other buses which are known as **foreign buses**
  - By way of a gateway link
  - Gateway links are created administratively



A bus can connect to other buses, which are referred to as foreign buses.

If messaging engines are on different buses, applications can use those different buses, each with its own topology and set of resources.

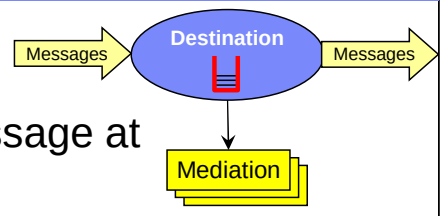
The inter-bus links might reflect the distribution of buses across organizations, across departments within organizations, or perhaps the separation of test and production facilities.

To create a link to a foreign bus, the administrator first creates a virtual link from the local bus to the foreign bus, then creates a physical gateway link from a messaging engine in the local bus to the foreign bus.

Section

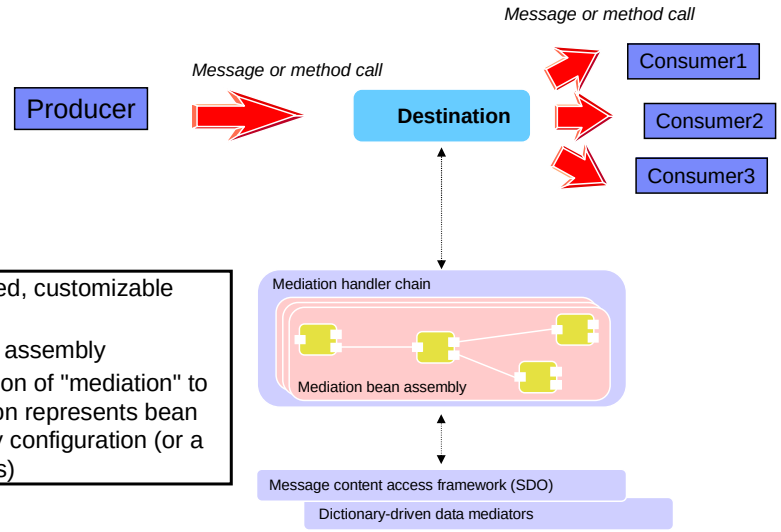
Mediation

Mediation Overview



- The ability to manipulate a message at a destination
  - Transform or reroute the message
  - Copy and route the message to additional destinations
  - Allow interaction with non-messaging resource managers (e.g. Databases)
- Mediation attached administratively to a Destination

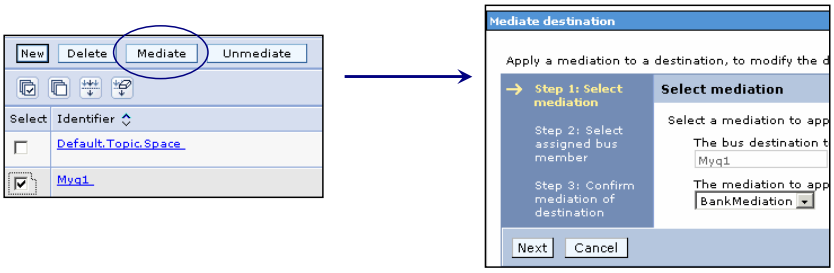
Mediation in the messaging engine



- Predefined, customizable beans
- Dynamic assembly
- Association of "mediation" to destination represents bean assembly configuration (or a java class)

Associating Mediation to destination

- After the mediation handler is deployed, the mediation is associated with the destination
  - Only one mediation can be applied to a destination
  - A mediation can contain multiple mediation beans
- Mediations are started at a mediation point
  - A mediation point is a location in a messaging engine at which messages are mediated



After a mediation is deployed to the service integration bus, it is associated with a destination.

A mediation is invoked at a mediation point when it receives a message from the runtime. The mediation operates on an instance of the message, for example transforming it, or forwarding it to other destinations.

Section

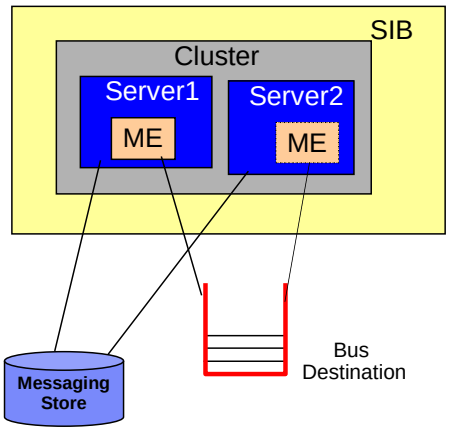
High Availability and Scalability

High Availability (HA)

- High availability is achieved by using clusters
  - An ME is created in a cluster when a Bus Member is defined on that cluster
- Any server in the cluster can run the messaging engine
  - But, only one ME is active at a given time
- HA Manager decides which server runs the Messaging Engine at any given time
  - It activates the messaging engine in the server.
  - If the ME then fails, the HA Manager activates a ME on another server in the cluster
  - WLM routes JMS clients to the member currently hosting the ME

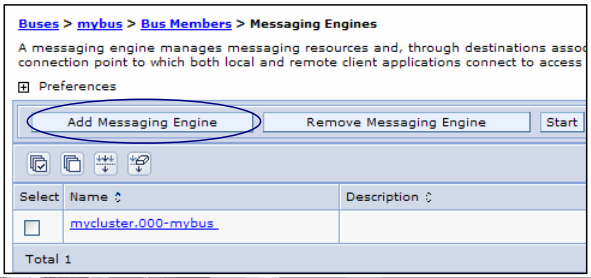
High Availability Details

- The destination is localized to a cluster because it is associated with a Bus Member
- The ME can therefore failover to a different cluster member
- Data store used by the ME needs to be accessible from all cluster members

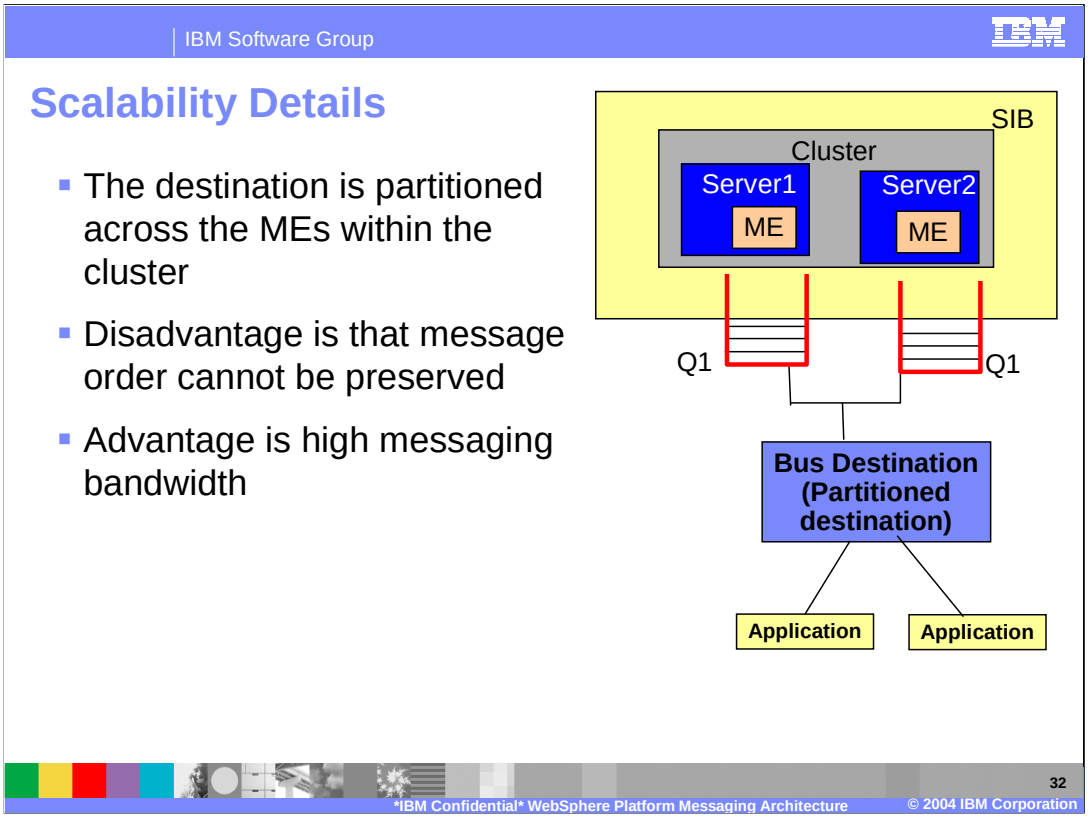


Scalability

- Scalability is achieved by deploying *multiple messaging engines per bus* to a cluster
- With this flexibility, there are **one or more** concurrently-active messaging engines running in a cluster on the same bus
  - When a destination is hosted on such a cluster, then the destination is partitioned
- Failover is handled by the HAManager



e.g.: a single logical queue might be partitioned across the cluster (with each of *n* partitions perhaps holding an *n*th of the messages)



This case provides scalability. The destination has been localized to a ClusterME. It is therefore partitioned across the MEs within the cluster. With a partitioned Destination, the recoverable objects associated with the destination are split between separate Message Stores and hence separate Data stores. This configuration has the disadvantage that message order cannot be preserved, but has advantages. One is that multiple consumers (or producers) can be deployed across the same Cluster to provide high messaging bandwidth. Messaging operations would always be locally fulfilled.

Scalability can be increased by introducing additional MEs into a cluster.

A set of MEs localizing a partitioned destination must all be in the same cluster. Likewise, an availability group of instances of the same ME must all be in the same cluster.

Summary

- WebSphere Platform Messaging provides a JMS v1.1 compliant JMS Provider
- SIBus is the communication layer for WebSphere Platform Messaging
- Messaging Engines manage messaging resources
- MQLinks can be created to communicate with WebSphere MQ Queue Managers

Section

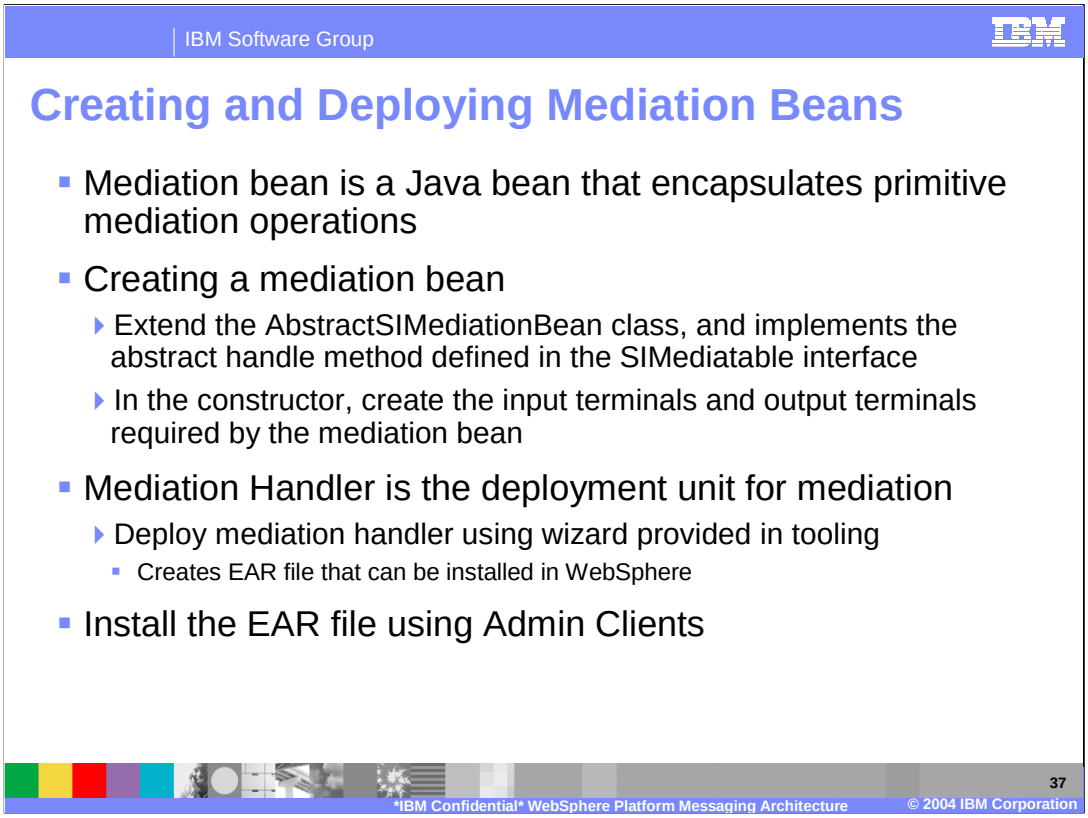
Appendix

HA settings for Messaging

- The HA policies determine the high availability characteristics of a ME
  - Can be set using Admin Console or wsadmin
- By default, all MEs have the default SIB policy (one of N)
  - All clusters belong to the “DefaultCoreGroup” by default

HA policy options for messaging

- Static
  - ME is static – cannot be failed over
  - This makes WebSphere 6.0 behave like 5.0
- 1 of N (default behavior)
  - The HA manager elects one member to host ME and if that member fails then another takes over very quickly
  - Admin can configure a set of preferred servers
- OS Managed
  - WebSphere does not do fail-over
  - A third party must tell WebSphere using JMX where to place the ME



Connecting, or wiring, mediation beans together provides more complex mediation functionality than is supplied by a single mediation bean. A group of connected mediation beans is called a mediation bean assembly. To wire two mediation beans together, use the SIMediationBean interface in package com.ibm.websphere.sib.mediation.beanframework

Mediation beans can be aggregated to provide more complex functions

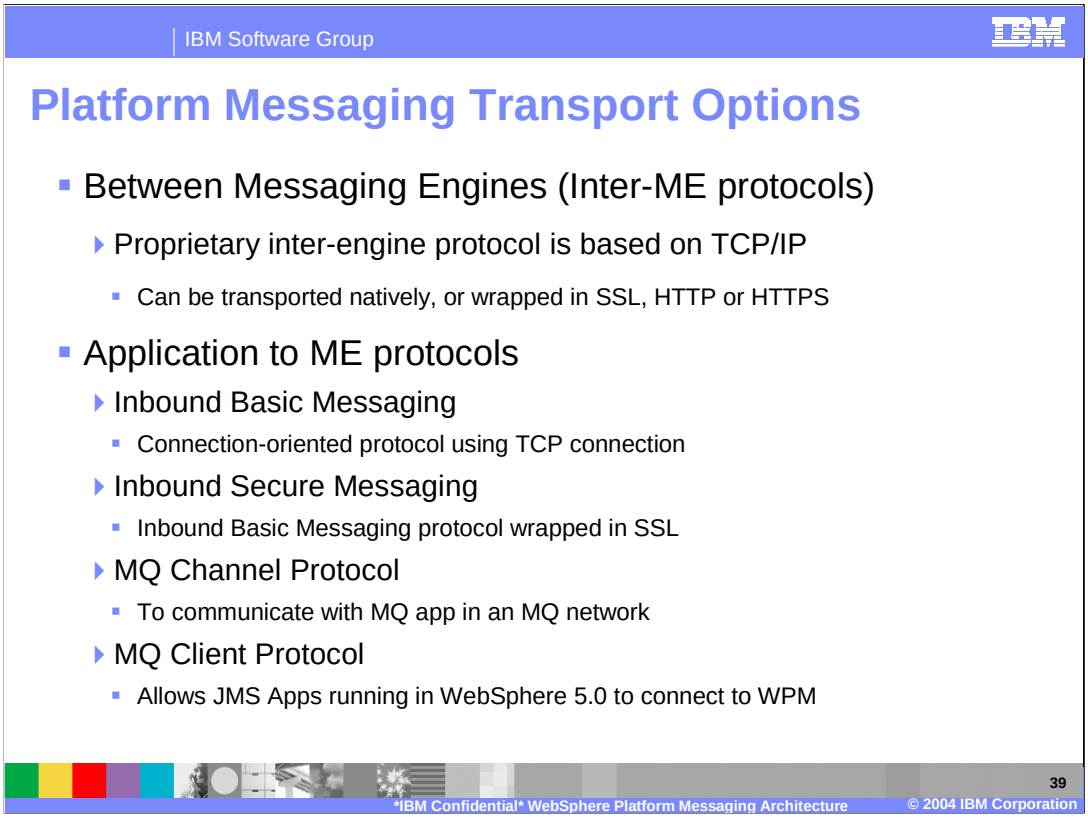
Default values that are set when a MediationBean is created may be overridden when a mediation is deployed

At runtime, a handler list is associated with one or more specific destinations. This makes the handler list eligible to receive messages when they arrive at the destination. The handler list can be configured to either run inside a single global transaction, or to run as a local transaction.

A mediation handler is deployed either as a simple JavaBean, or as stateless session Enterprise JavaBean (EJB). The behavior of a mediation may be influenced by modifying its properties at deployment. A mediation handler is deployed using the Deploy wizard available in the Assembly Toolkit, and is installed into WebSphere Application Server as an EAR file.

Section

Platform Messaging Protocols



Formats and Protocols (FAP) :The WebSphere MQ FAPs define how queue managers communicate with one another, and also how WebSphere MQ clients communicate with server queue managers.

JFAP :The formats and protocols used to communicate between messaging engines.

MEs communicate with each other using an efficient proprietary protocol. This carries the actual message data, along with control information that allows the work of the bus to be distributed across the various engines that make it up.

As with the PM client FAP, the inter-engine FAP is based on TCP/IP and can be transported natively, or wrapped in SSL, HTTP or HTTPS and uses the Channel Framework to provide the I/O support.

InboundBasicMessaging

This is a connection-oriented protocol, using a standard TCP/IP connection (JFAP-TCP/IP). It includes support for two-phase transactional (remote XA) flows, so that a message producer or consumer, running on a client or server system, can participate in a global transaction managed on that client or server system. The specific use for the XA flows is to support access from an application running in one server to a messaging engine on second server, perhaps because the first server does not have a suitable messaging engine. If the remote XA flows are used, a transaction coordinator must be available local to the application.

InboundSecureMessaging

This is the InboundBasicMessaging protocol wrapped in SSL.

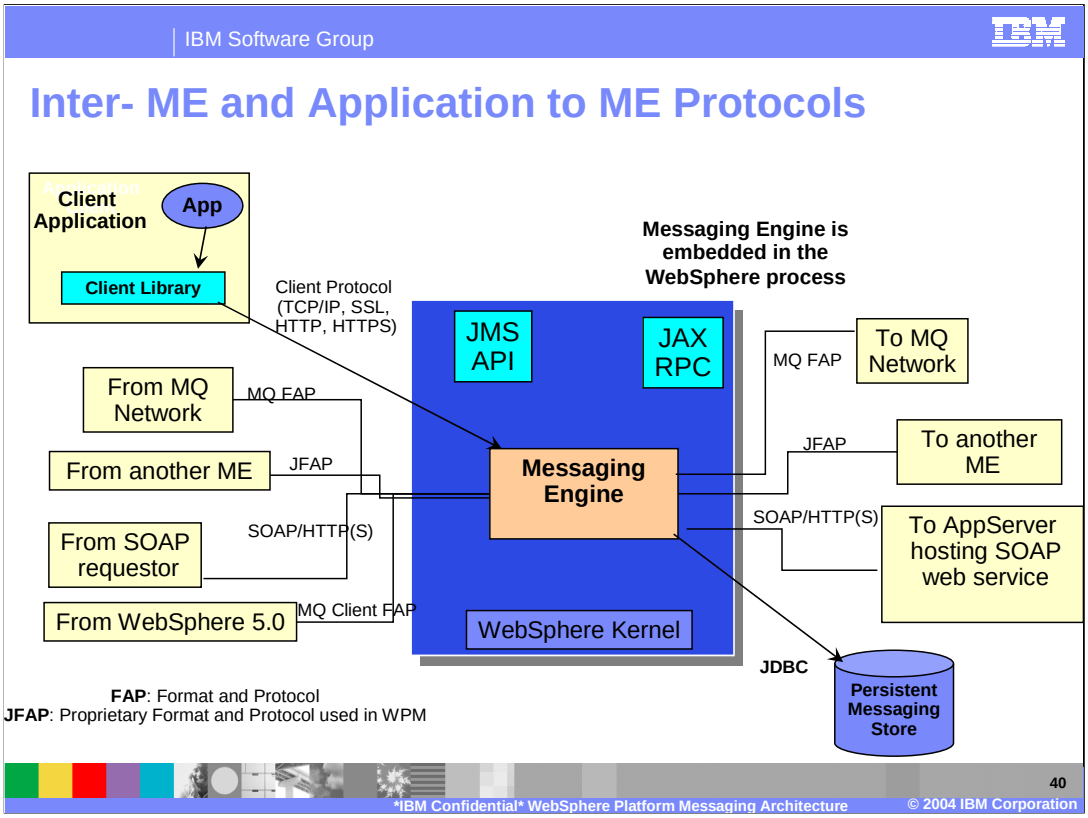
InboundBasic and Secure Messaging settings are made when creating the connection factory

MQ channel protocol

This allows a PM bus to send messages to or to receive messages from an MQ application running against a queue manager in an MQ network.

MQ client protocol

Allows JMS applications running on WAS 5.0 using the embedded JMS provider to connect to WPM. Note this support is provided for the MQ client protocol only. There is no support in WPM for WAS 5.0 clients using the 'Direct' attachment.



Applications can attach to WPM by using the JMS API. WPM provides API libraries that connect the application to a messaging engine, either via an in-process call, or across a network using a remote client. A remote client may run in the J2EE Application Client environment and the J2EE Application Server environment.

Template Revision: 10/01/2004 4:46 PM

Trademarks and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	iSeries	OS/400	Informix	WebSphere
IBM(logo)	pSeries	AIX	Cloudscape	MQSeries
e(logo)business	xSeries	CICS	DB2 Universal Database	DB2
Tivoli	zSeries	OS/390	IMS	Lotus

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are
trademarks of Intel Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon 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 throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.

This Page Intentionally Left Blank