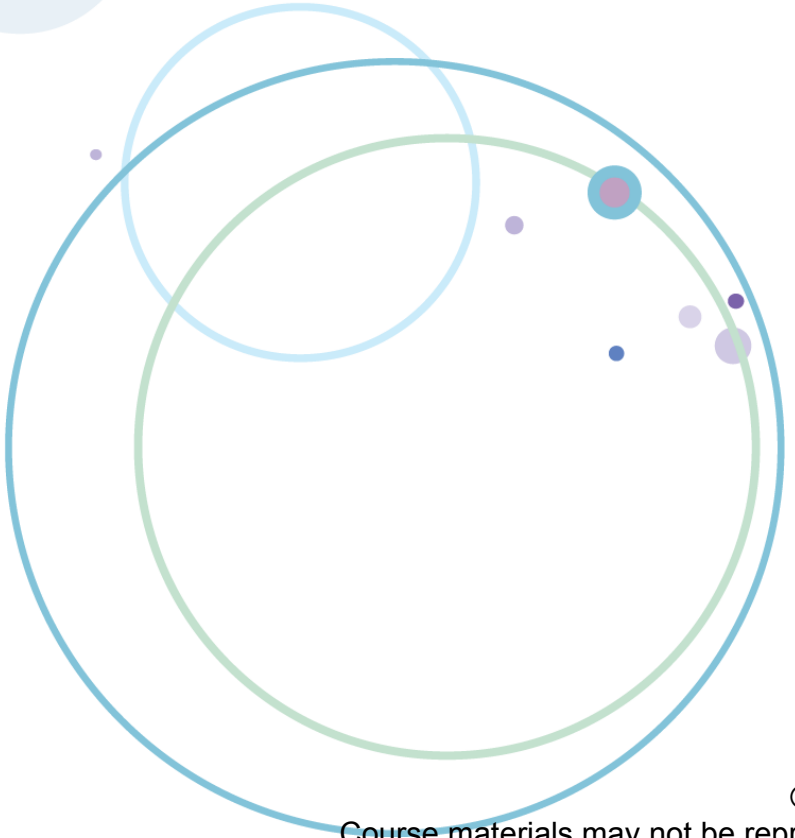
Introduction to WebSphere Messaging



Unit objectives

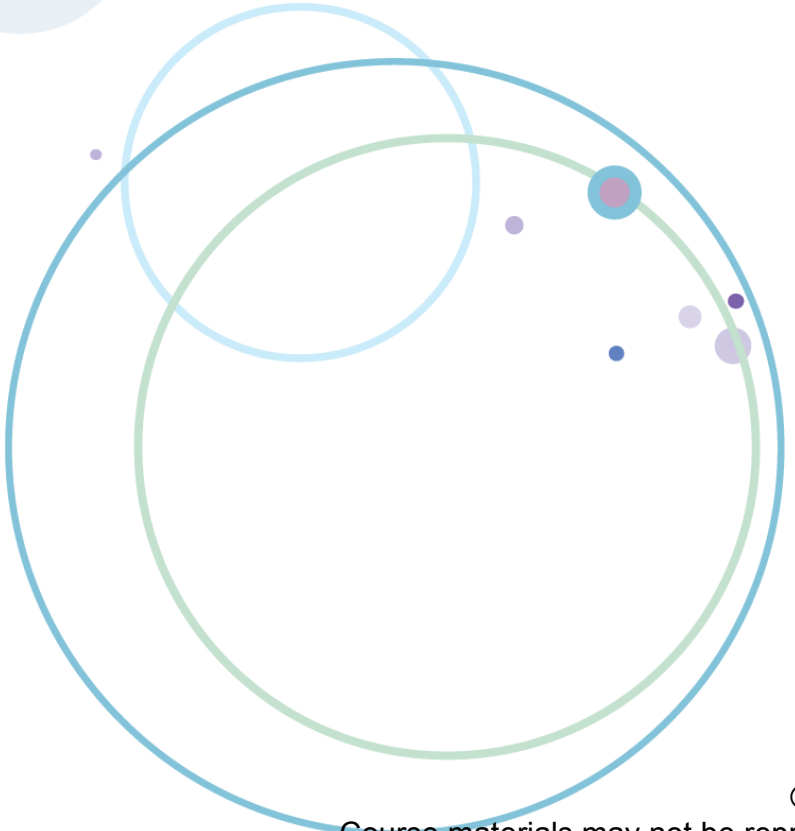
After completing this unit, you should be able to:

- Describe what WebSphere Messaging is and how it is used
- Describe messaging components such as JMS providers, the service integration bus (SIBus), and messaging engines
- Configure resources to support messaging applications such as queues, topics, and connection factories
- Implement various clustered messaging engine policies for high availability and scalability
- Create links to foreign buses and WebSphere MQ
- Describe how JMS and WebSphere MQ use the SIBus to support application messaging services

Topics

- Overview of messaging concepts
- Messaging engine clustering
- SIBus and messaging engine topologies
- Additional messaging considerations

Overview of messaging concepts

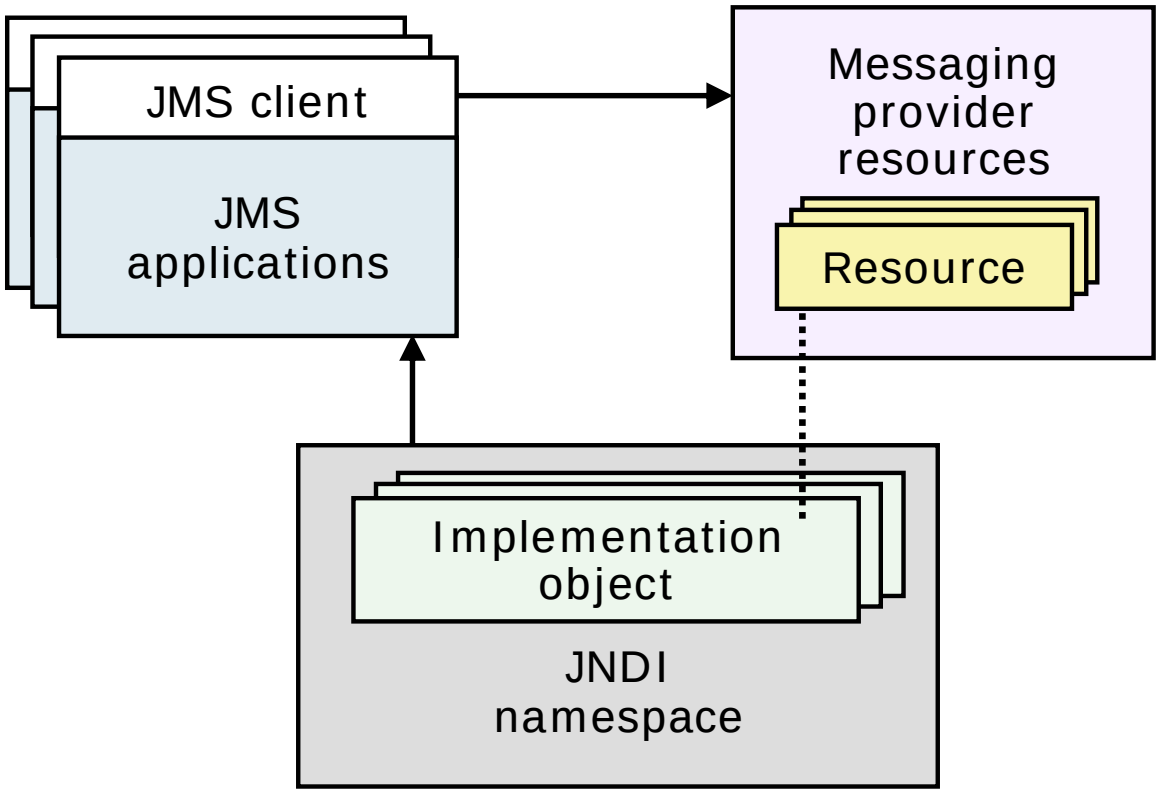


What is JMS?

- The Java Message Service (JMS) is an API for accessing enterprise messaging systems
- WebSphere Application Server V8.5.5 supports JMS 1.1 as part of the Java Platform, Enterprise Edition (Java EE) 6 specification
- The Java Message Service allows Java EE applications to asynchronously send and receive business data and events
- JMS supports two styles of asynchronous messaging:
  - Point-to-point (queues)
  - Publish and subscribe (topics)

JMS applications

- The JMS specification defines the interfaces that JMS applications use
- JMS applications do not need to know how the interfaces are implemented
  - Java objects that implement the interface are returned through JNDI lookups
  - The implementation objects are vendor-specific
  - The implementation objects use vendor-specific properties to access the messaging resources



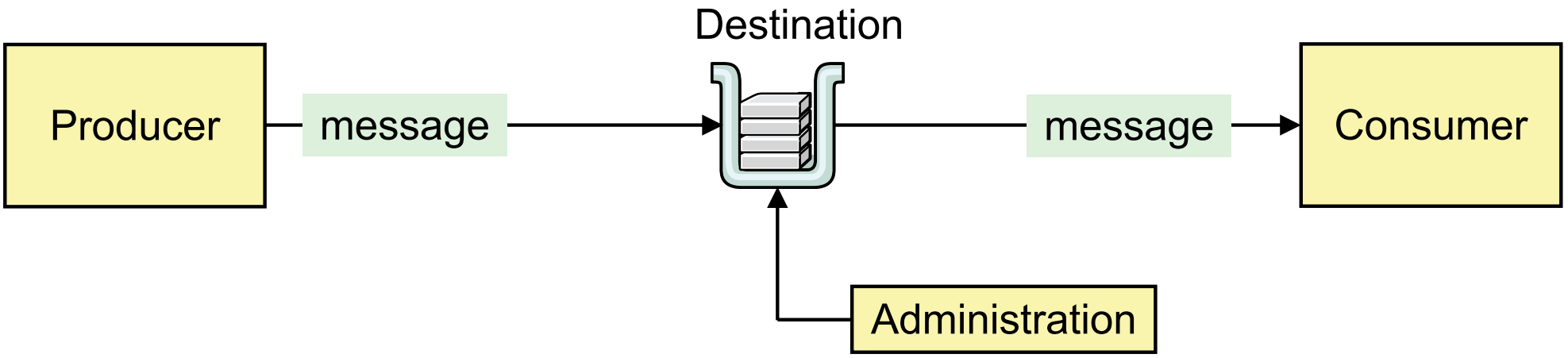
What is a JMS provider?

- A **JMS provider** is the implementation of the JMS API
- The following JMS providers are supported:
  - WebSphere Application Server default messaging provider
  - V5 default messaging provider
  - WebSphere MQ messaging provider
  - Generic JMS provider: Click **New** and define a third-party provider

New Delete

Select	Name	Description	Scope
You can administer the following resources:			
	Default messaging provider	Default messaging provider	Cell=was8hostCell01
	V5 default messaging provider	V5 default messaging provider	Cell=was8hostCell01
	WebSphere MQ messaging provider	WebSphere MQ messaging provider	Cell=was8hostCell01
Total 3			

Basic messaging flow



- Producers send or put messages to destinations
- Consumers receive or get messages from destinations
- Destinations are managed points of communication:
  - JMS queues
  - JMS topics (publish and subscribe)
  - Web service endpoints
- While the messaging flow is inherently asynchronous, it can be used to provide synchronous communication
  - Uses temporary REPLY-TO destinations

WebSphere default messaging

- Default messaging is the JMS provider that is delivered with WebSphere Application Server
- Messaging capabilities are fully integrated into the WebSphere Application Server
- Based on service integration bus (SIBus) technology
- Complements and extends WebSphere MQ and WebSphere Application Server
- Other WebSphere products use default messaging

Service integration bus (SIBus)

- A service integration bus is an administrative concept for configuring and hosting messaging resources
  - Buses are scoped to a network deployment (ND) cell
- An SIBus contains **bus members** and **destinations**
- Producers and consumers connect to SIBus bus members
- The SIBus manages communication with destinations
  - The SIBus can hold messages for a destination until a consumer becomes available

SIBus members

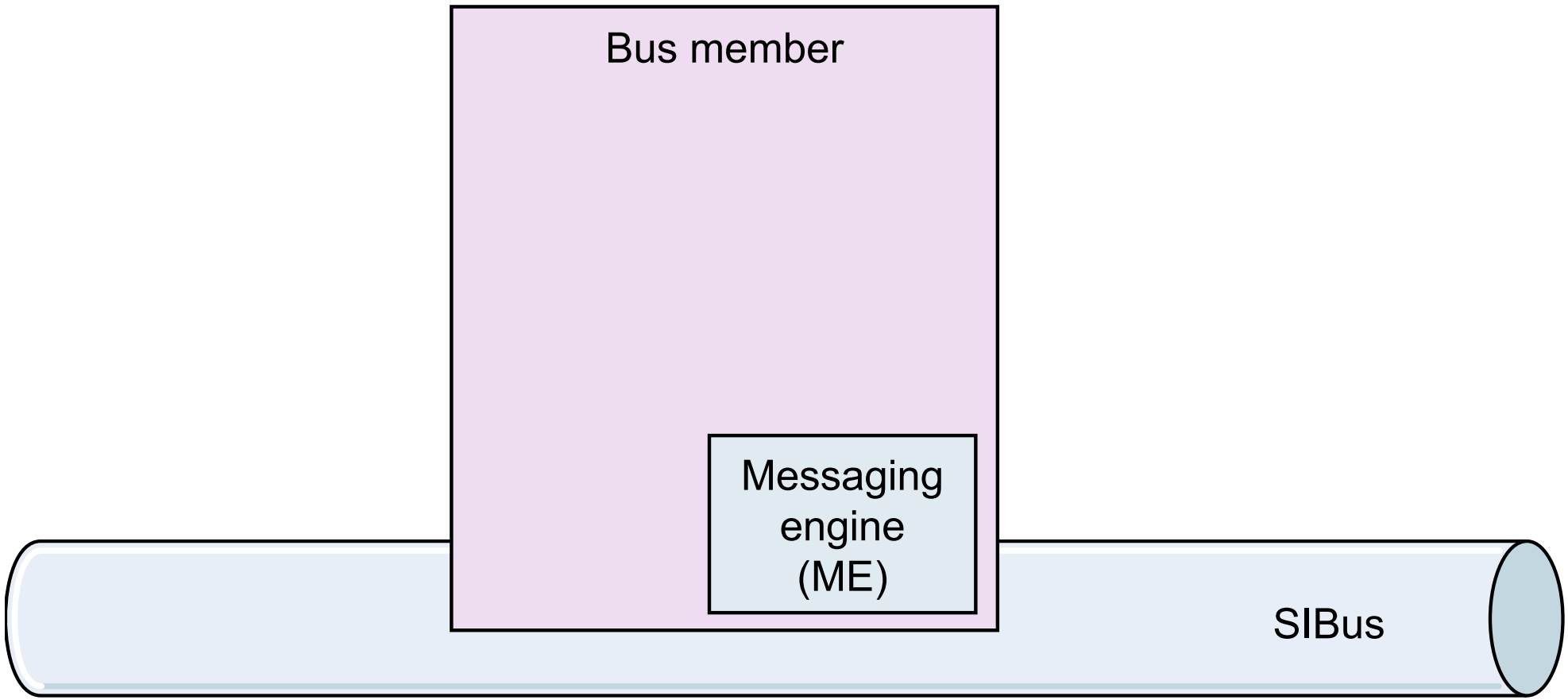
- SIBus members can be:
  - Application servers or clusters
- When a new bus member is defined, one or more **messaging engines** are automatically created
  - When adding a cluster as a bus member, more than one messaging engine might be created
- Bus members can be added or removed from the bus
  - This action effectively adds or removes messaging engines from the servers

Messaging engine (ME)

- MEs run inside the application server or cluster member, and manage messaging resources
  - A common pattern is one ME per bus member
- Each ME has a unique identity that is made up of the SIBus name and the name of the bus member
  - For example, `PlantsCluster.000-msgBus`, `PlantsCluster.001-msgBus`
  - `was85hostNode01.MyServer01-MyBus`
- MEs provide a connection point for clients to put or get messages
- All MEs are visible and accessible from anywhere on the bus, no matter which ME the client has an actual network connection with

Bus member and messaging engine

- An SIBus bus member can be a server or a cluster
- Each bus member contains at least one messaging engine (ME)
- The ME provides the runtime functions for the SIBus

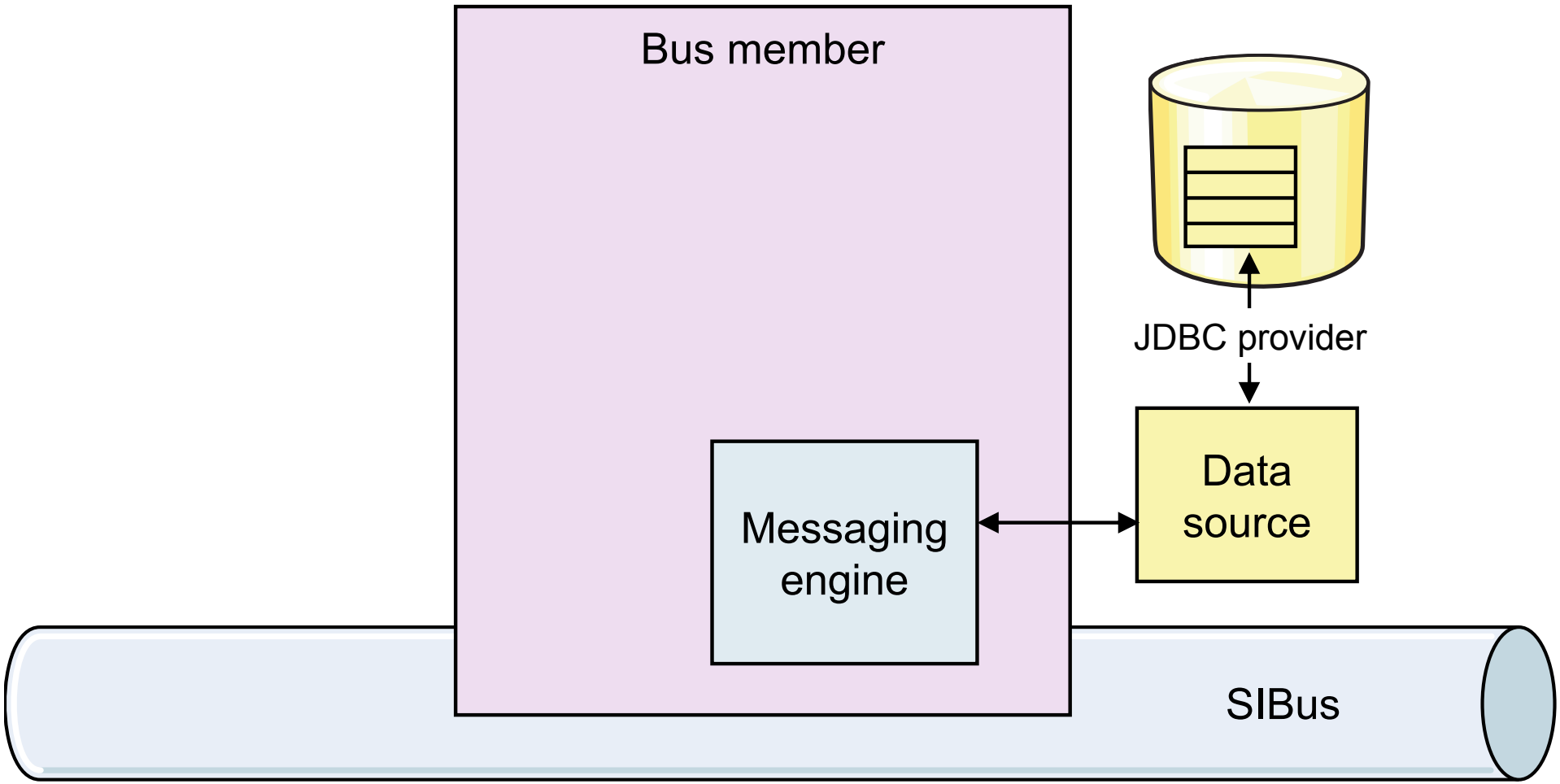


Message stores

- An ME requires a persistent backing store for storing recoverable data such as messages
- Two types of message stores
  - File stores (flat files in the file system)
  - Data stores (relational database tables)
- Multiple MEs can share a database, but each ME has its own schema within the database (which results in different tables)
- Derby database is used as the default data store in a stand-alone server
- For cluster bus members, a distributed database, such as DB2, or a shared file system is required

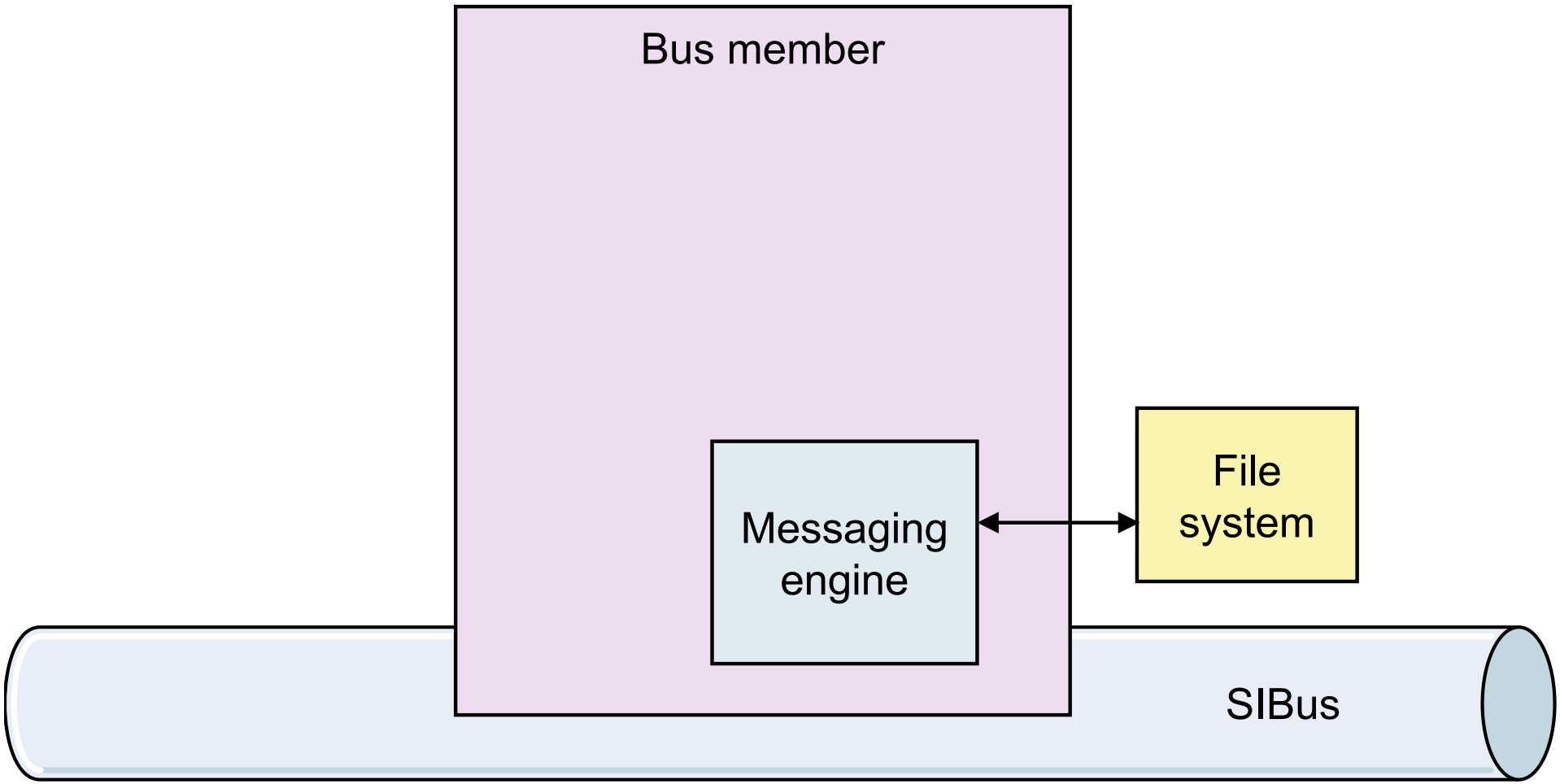
Messaging engine data stores

- An ME can be configured to use data source to connect to its message store



Messaging engine data stores

- An ME can also use the file system for its message store



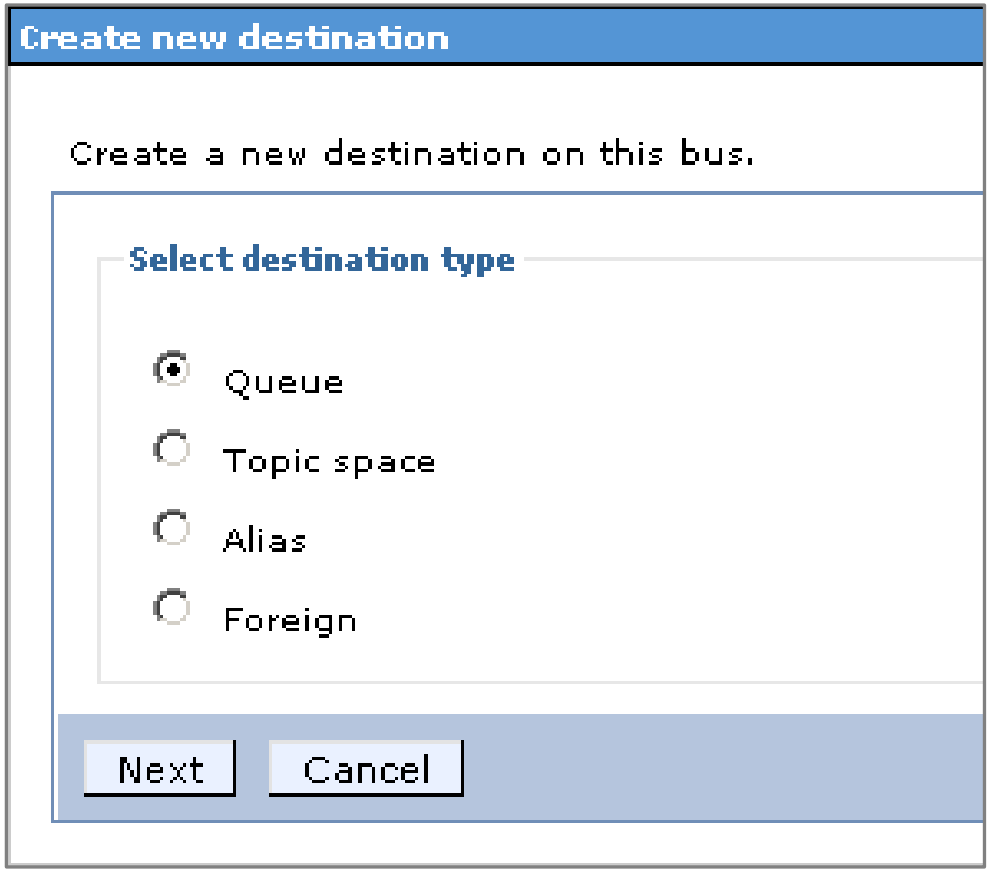
What is a bus destination?

- A bus destination is a **virtual place** within an SIBus, which applications (producers and consumers) use to exchange messages

- An SIBus destination is associated with bus members, therefore associating it with the corresponding MEs
  - MEs associated with a destination have a **message point** for that destination
  - Allows administrator to control which message store is used for persistence

- Any destination on a bus is visible and accessible to applications connected anywhere on the bus

SIBus destinations



- A logical name which applications use to exchange messages
- Queue
  - For point-to-point messaging
- Topic space
  - For publish and subscribe messaging
  - Represents hierarchies of topics
- Alias
  - Provide a level of abstraction between applications and the target bus destinations that hold messages
- Foreign
  - Identifies a destination on another bus
  - Applications on one bus can directly access the destination on another bus
- Exception
  - Automatically created for each messaging engine

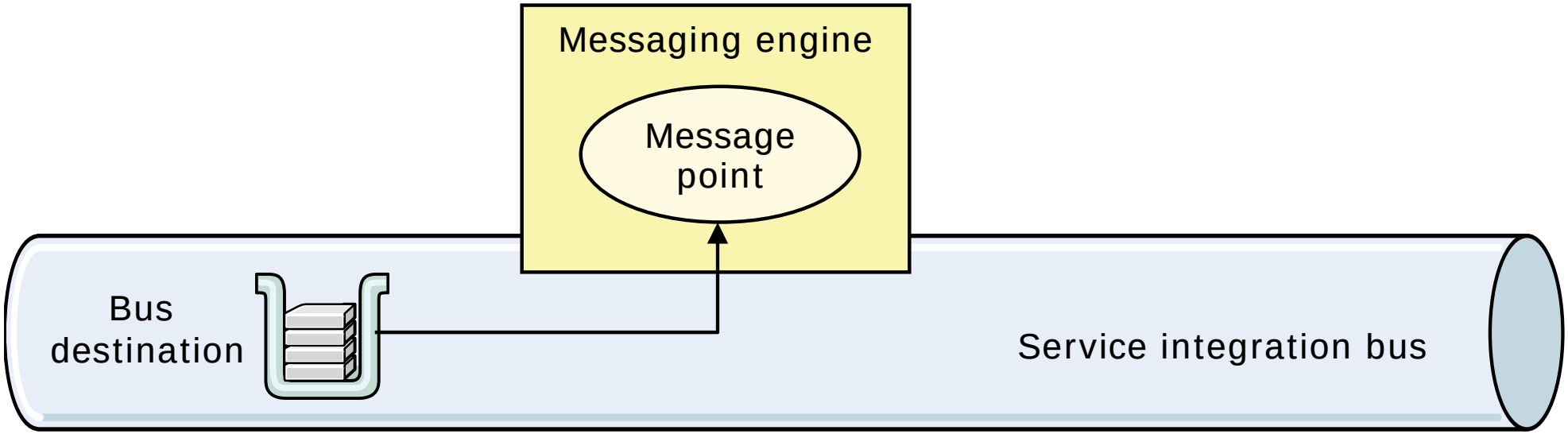
Linking destinations to bus members

- A bus destination is associated with one or more bus members, therefore associating it with the corresponding MEs
  - Allows the administrator to control which database is used for persistence
  - In most cases, a destination is associated with one ME
  - Multiple MEs provide scalability
- Use a queue for point-to-point messaging
  - The administrator defines a queue destination on one assigned bus member
  - Each ME in that assigned bus member has a **queue point** where messages are held
- Use a topic space for publish and subscribe messaging
  - Every ME in the SIBus is a **publication point** where messages are held

Message points (1 of 2)

- A message point is the physical location on a messaging engine where messages are held for a bus destination

- A message point can be a
  - Queue point
  - Publication point

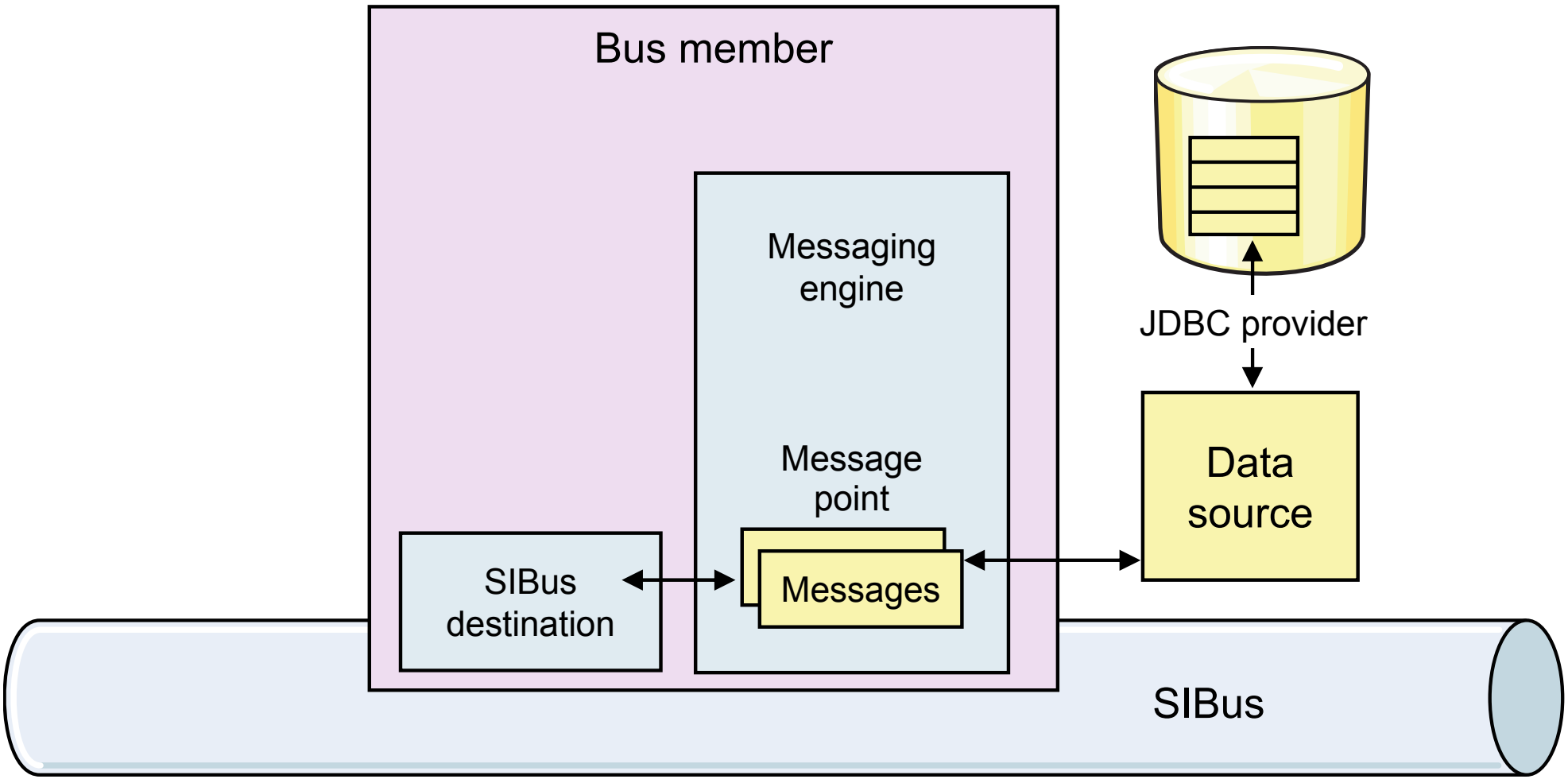


Message points (2 of 2)

- A queue point is the message point for a queue destination
- When creating a queue destination on a bus:
  - Specify the bus member that holds the messages for the queue
  - A queue point is automatically defined for each messaging engine that is associated with the specified bus member
- A publication point is the message point for a topic space
- When creating a topic space destination:
  - Creating a topic space destination automatically defines a publication point on each messaging engine within the bus

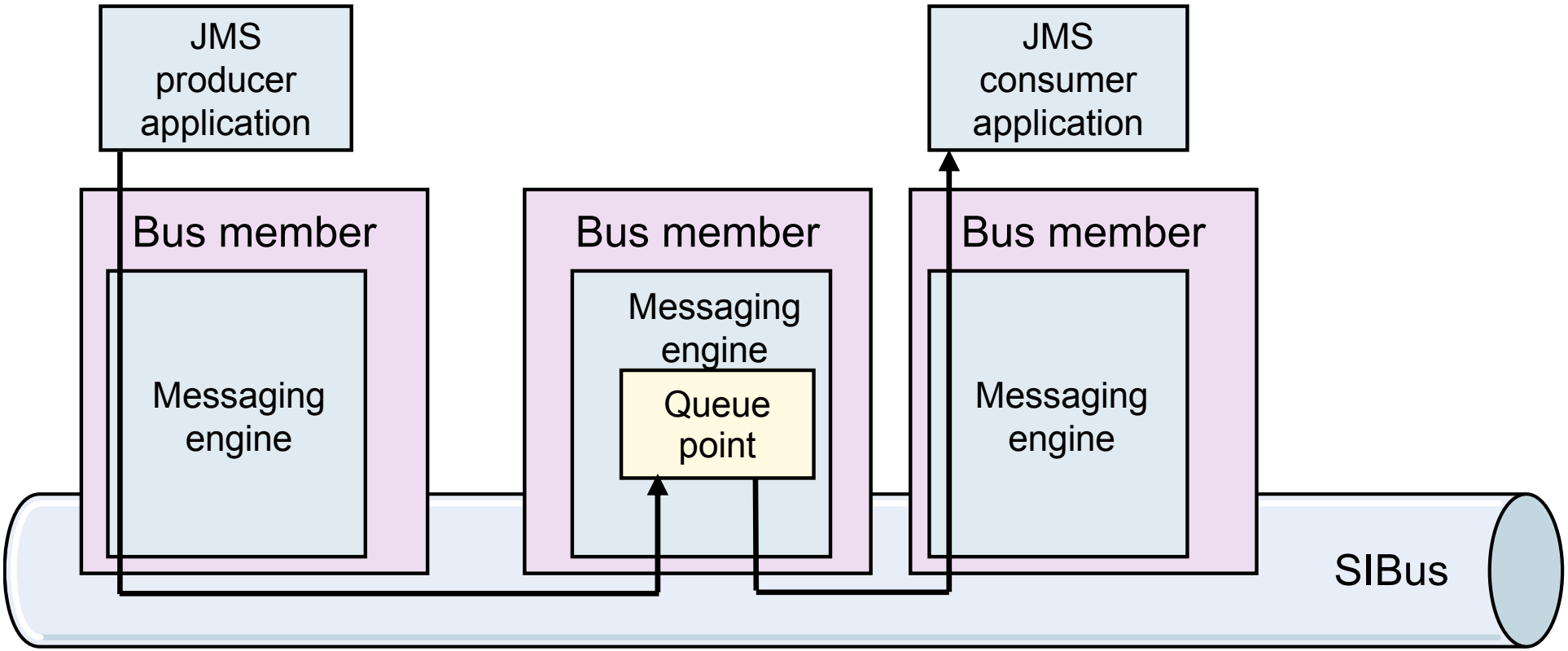
SIBus destinations

- SIBus destinations are associated with one or more MEs
  - Queues are explicitly assigned to a bus member
  - Topic spaces are associated with all bus members



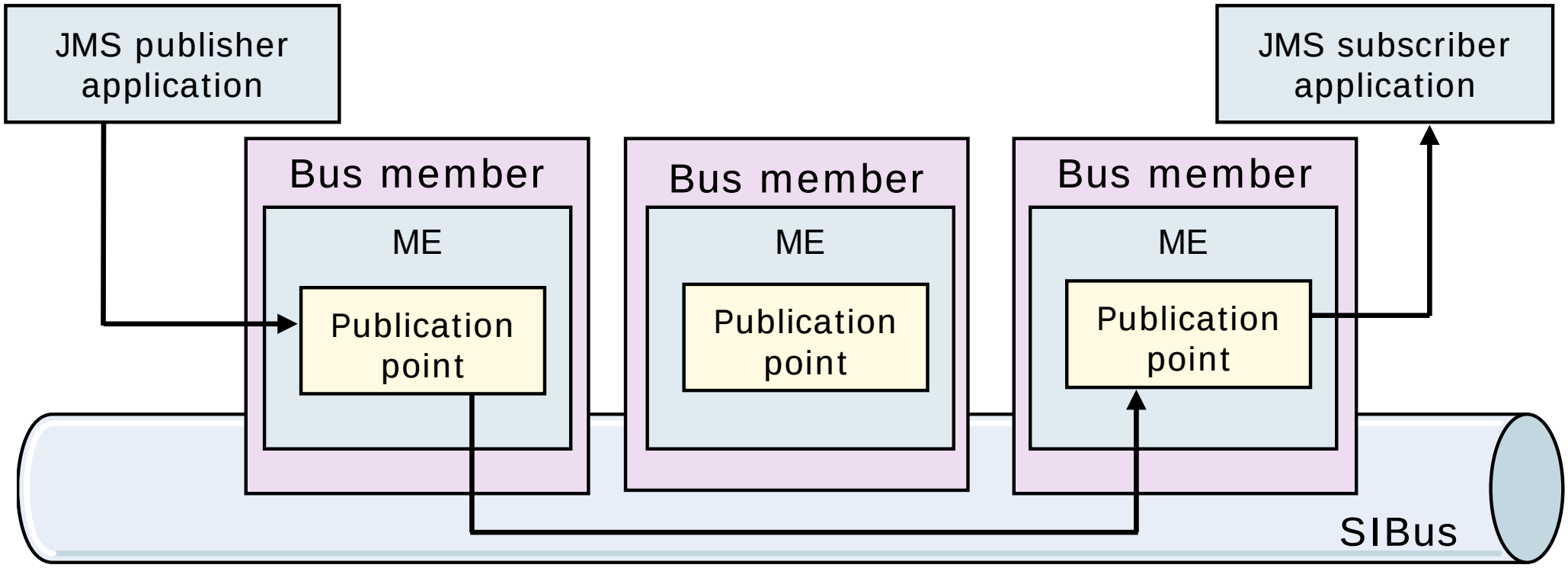
SIBus queue destinations

- Queues are visible across the entire bus
  - Physically on one bus member
  - Applications can connect to any ME on the bus and produce or consume messages from a queue
  - Even if the queue is associated with a different bus member



SIBus topic space destinations

- Each topic space is a namespace for dynamically created topics
  - Topics are hierarchical: for example, A/B/C
- They are located across the whole bus as publication points
- Subscriber subscriptions are localized to a single ME
- Messages that are published anywhere on the bus are propagated only to MEs with subscribers for the topics that are published to



Java EE access to bus members

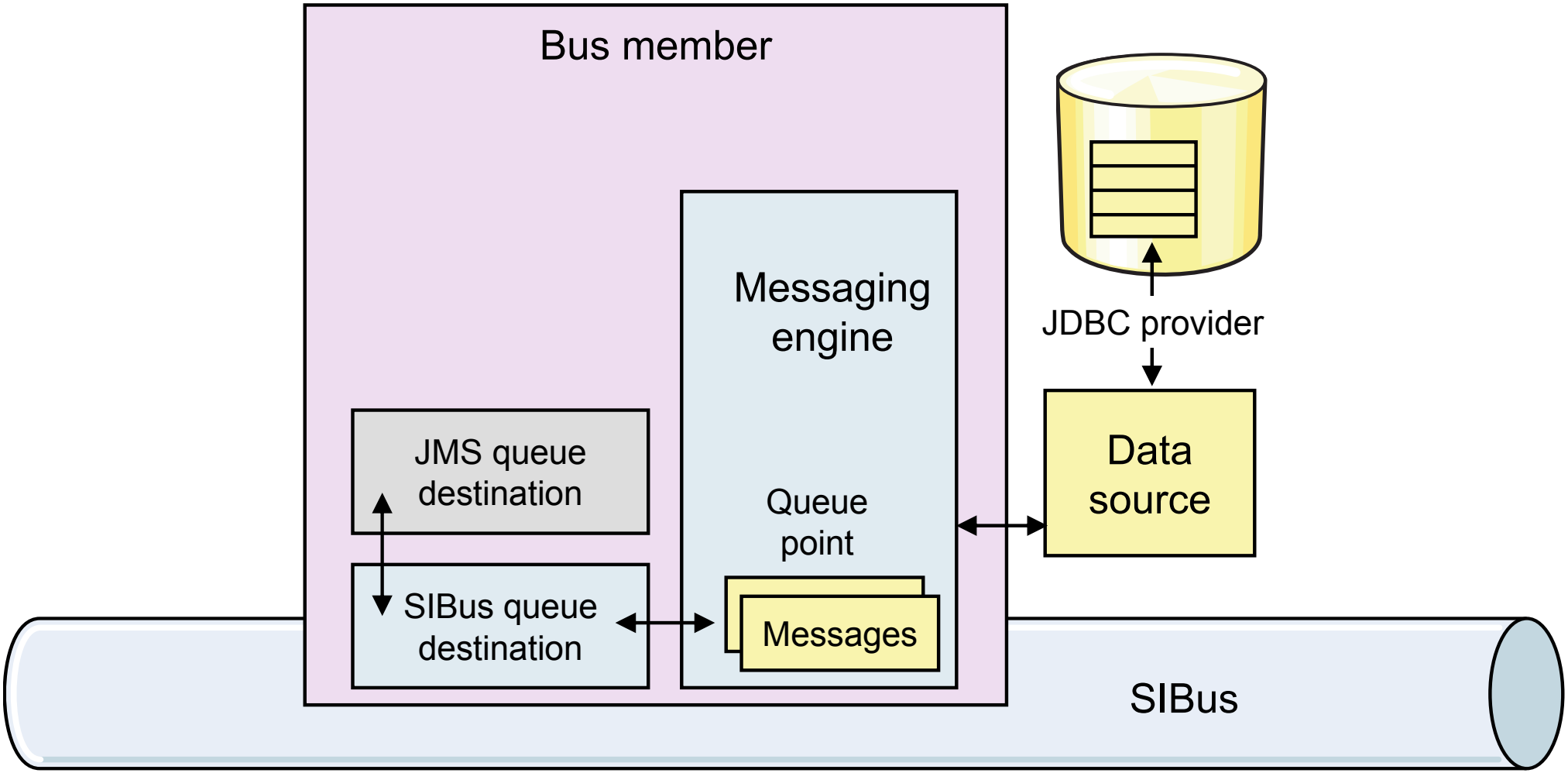
- Java EE applications (producers, consumers) access the SIBus and the bus members through the JMS API

- JMS defines queues and topics
  - Applications access JMS destinations
  - JMS destinations have JNDI names
  - For example, `jms/tradeQueue`, `jms/storeTopic`

- JMS defines interfaces for accessing destinations
  - **ConnectionFactory**: Java EE components use it to connect to the SIBus through a messaging engine (Typically used by producer applications)
  - **ActivationSpec**: message-driven beans use it to connect to the SIBus through a messaging engine (Used by consumer applications)

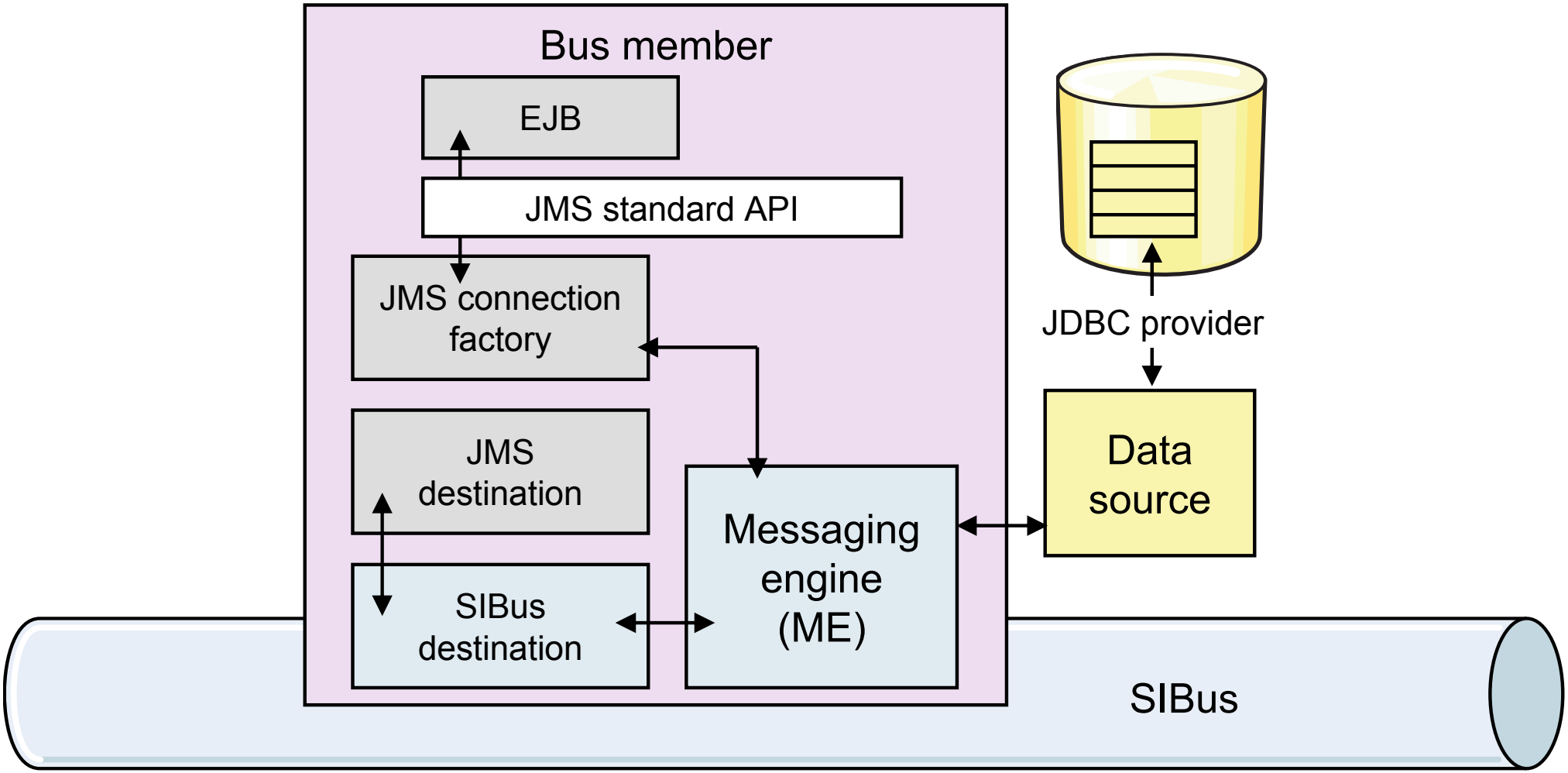
JMS destinations

- JMS destinations are associated with SIBus destinations
- The SIBus destination implements the JMS destination function



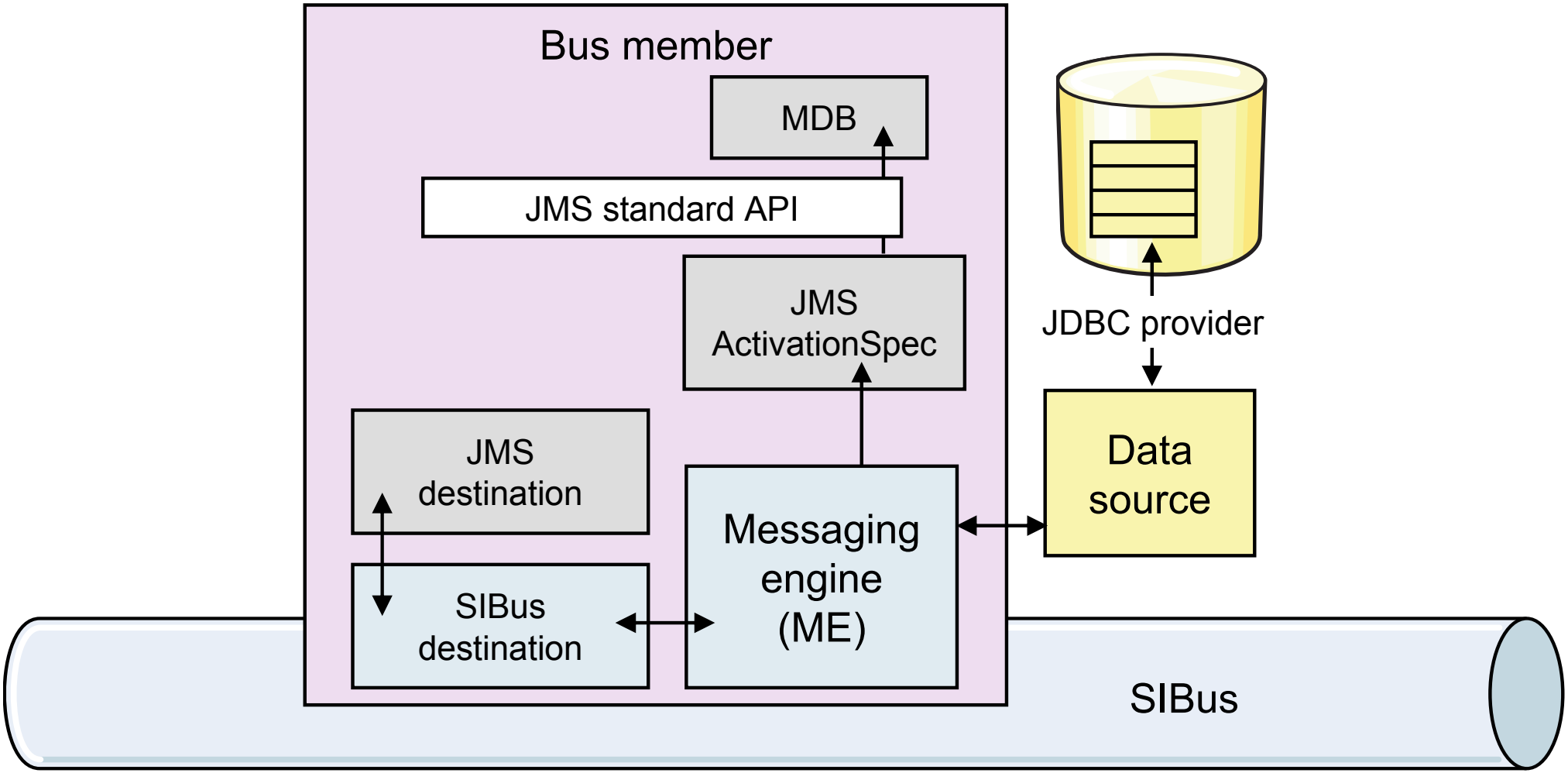
JMS connection factory

- Producers and consumers use the JMS API to access destinations
- Java EE components (Session EJBs for example) use a JMS connection factory to connect to the JMS provider (ME)



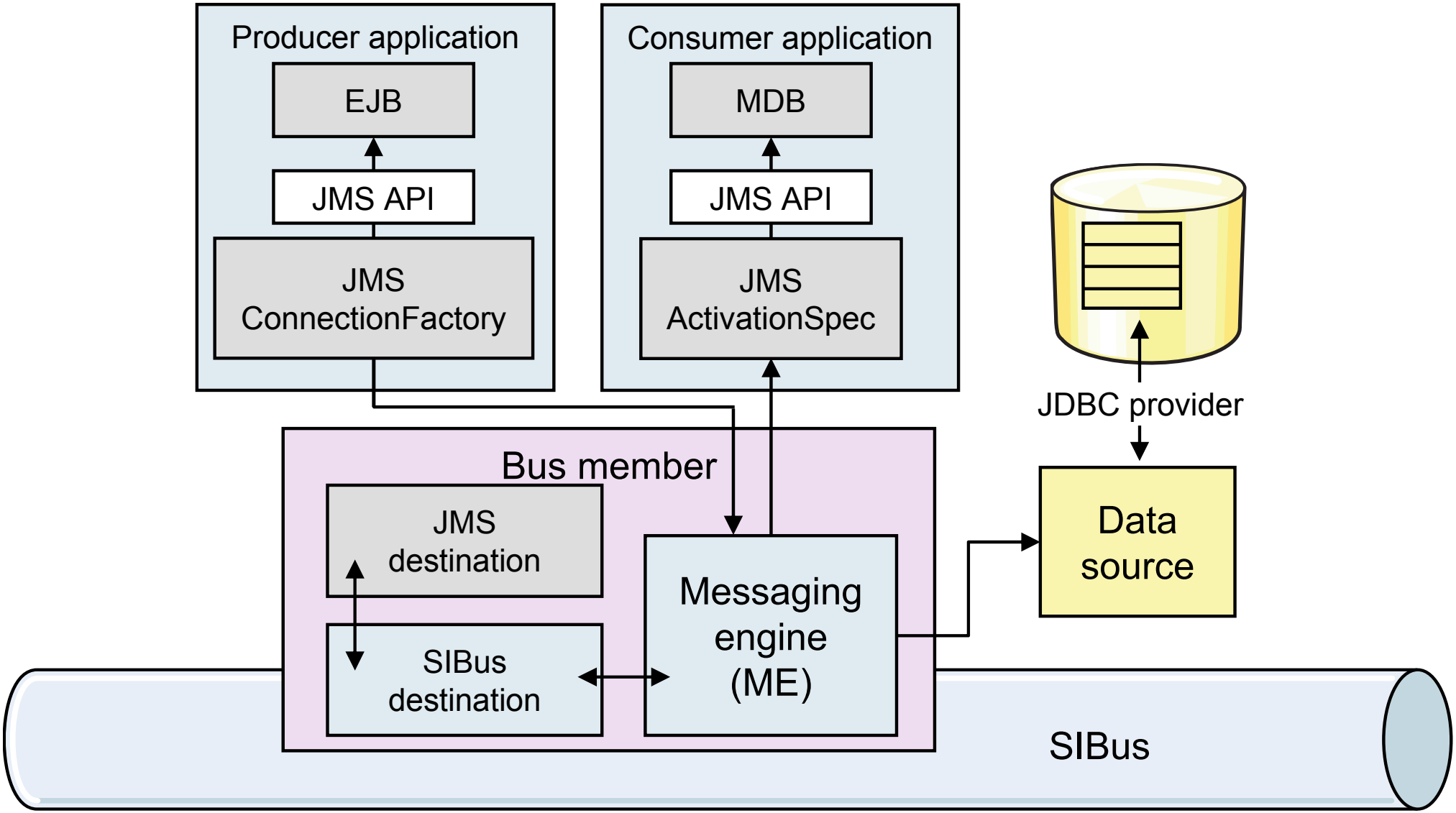
JMS ActivationSpec

- Message-driven beans (MDBs) use a JMS ActivationSpec to connect to the JMS provider (ME)

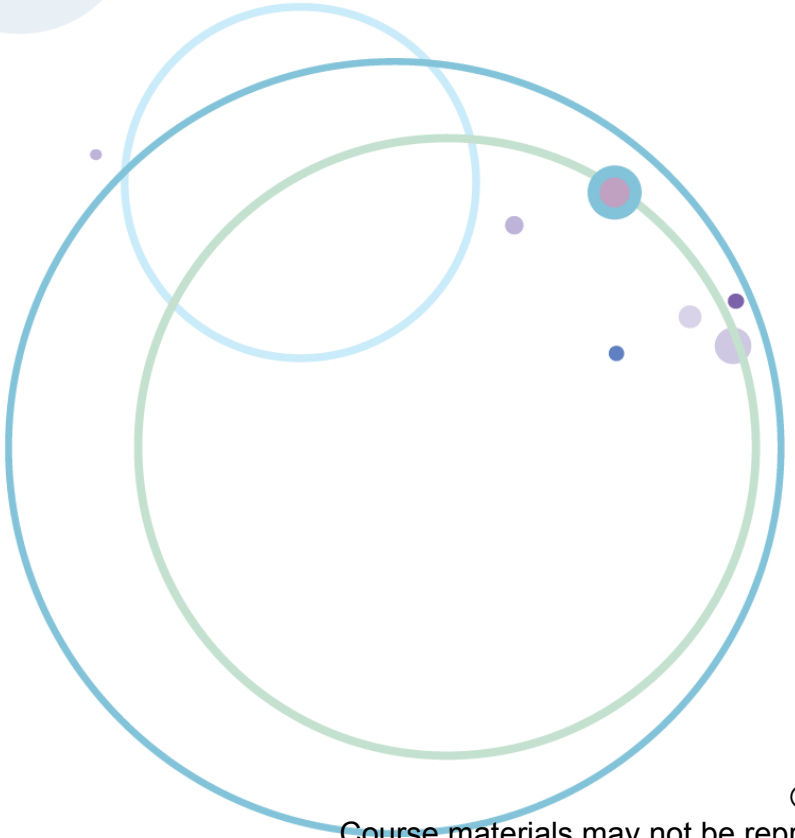


Applications can run outside bus members

- Producers and consumers can also run outside of the server that is hosting the ME



Messaging engine clustering



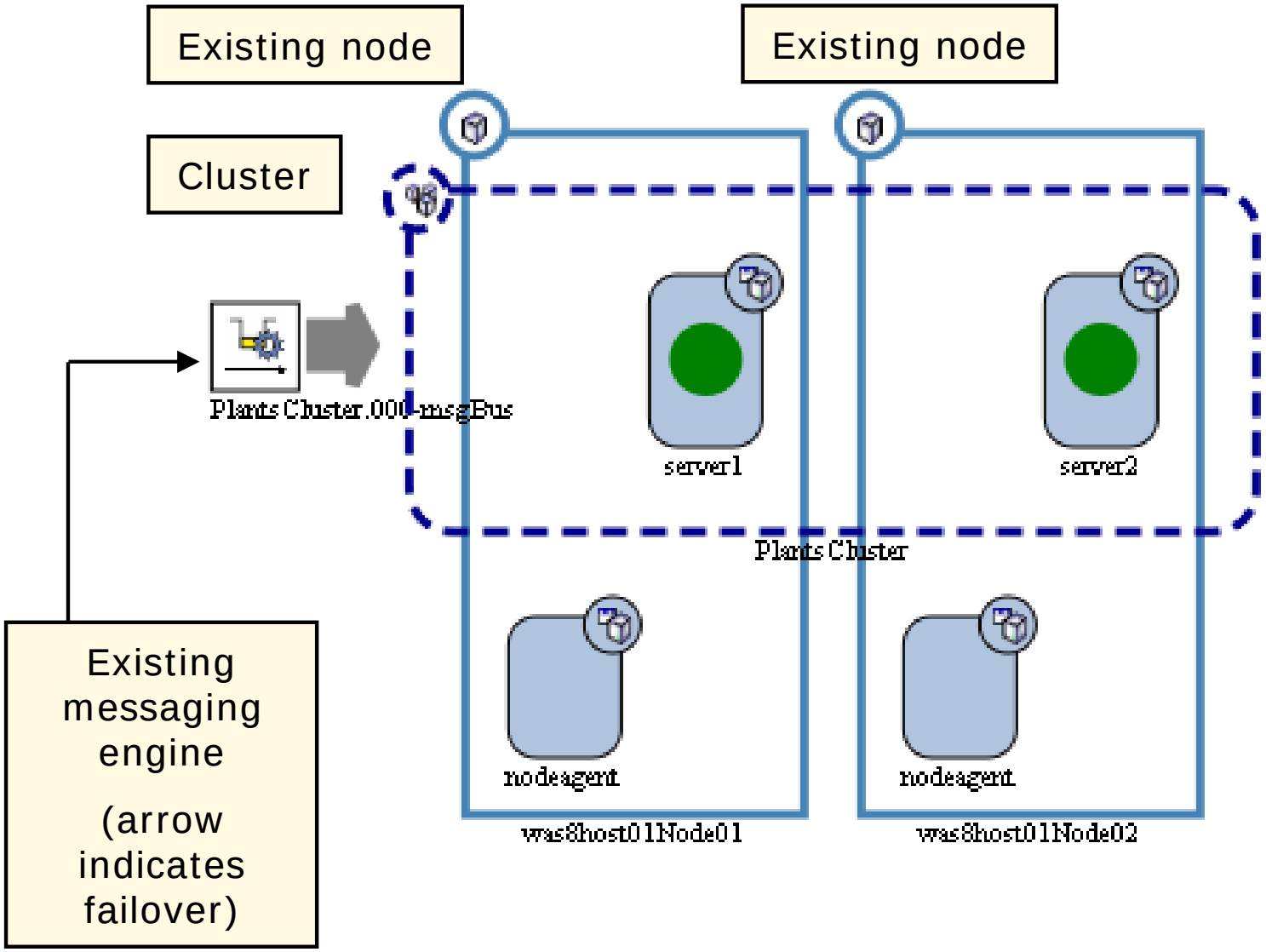
Messaging engine policy assistance

- Administrative console feature in WebSphere Application Server
- When you add a cluster to an SIBus, you can choose one of three predefined messaging engine policies:
  - High availability
  - Scalability
  - Scalability with high availability
- A custom policy option is also available

Select	Policy type	Is further configuration required?
◉	High availability	No
○	Scalability	⚠ You need to add the following number of messaging engines: 1. ⚠ You need to correct the following number of messaging engine policies: 1.
○	Scalability with high availability	⚠ You need to add the following number of messaging engines: 2. ⚠ You need to correct the following number of messaging engine policies: 1.
○	Custom	Advice is not available for a custom configuration.

Example: High availability policy (1 of 2)

- **High availability:** Ensures that a single ME is always available

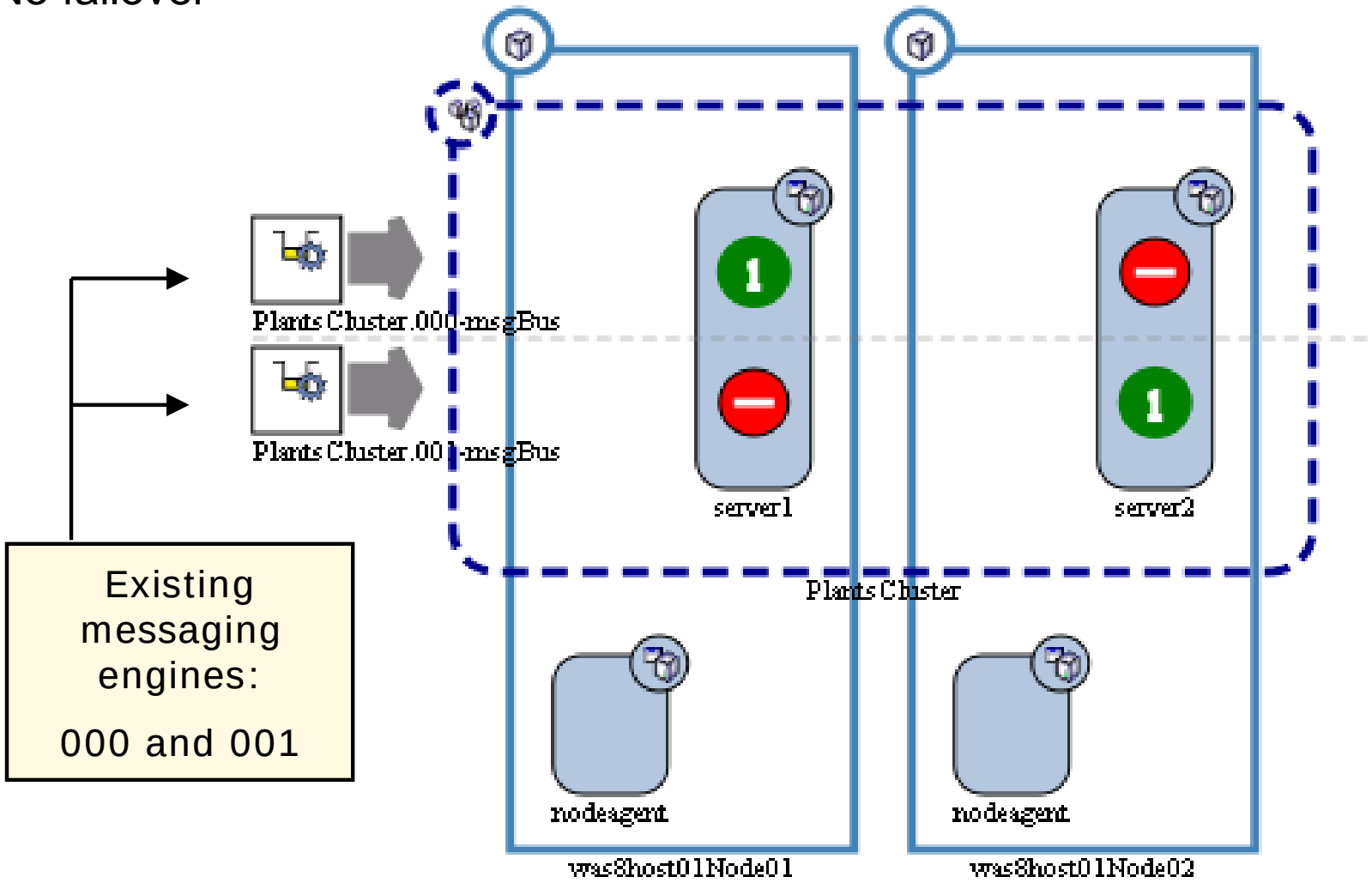


Example: High availability policy (2 of 2)

- The ME runs on only one cluster member
  - The HA manager decides which cluster member
- If the host cluster member fails, ME fails over to another cluster member
- No failback
- By default, if the consumer app runs on the same cluster as the bus member, only MDBs with an active ME get messages from the destination
  - You can configure all MDBs on all cluster members to receive messages
  - Configure the activation specification by using the option: always activate MDBs in all servers

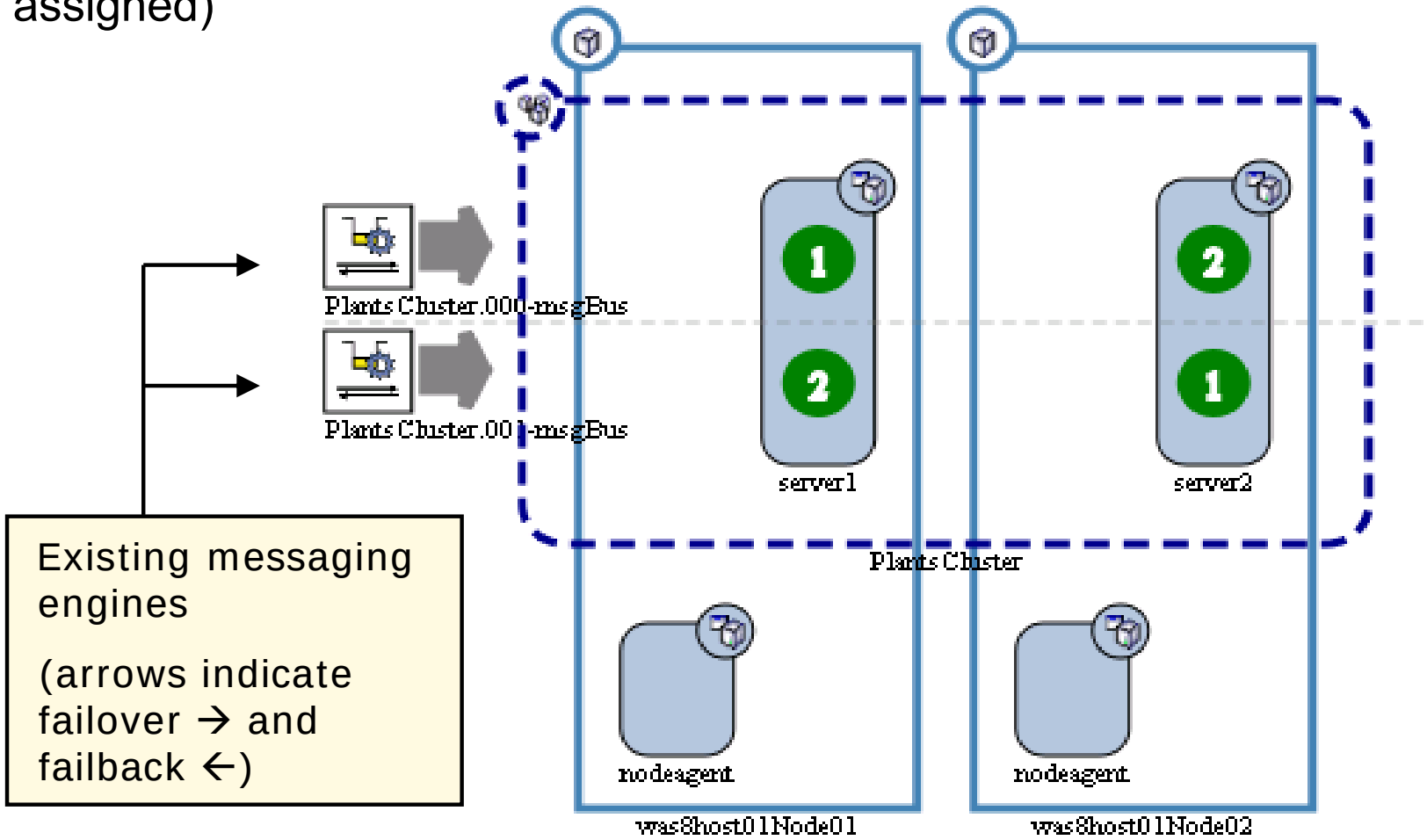
Example: Scalability policy

- **Scalability:** Creates an ME for every server in the cluster
  - Each ME is always hosted by its assigned server
  - No failover

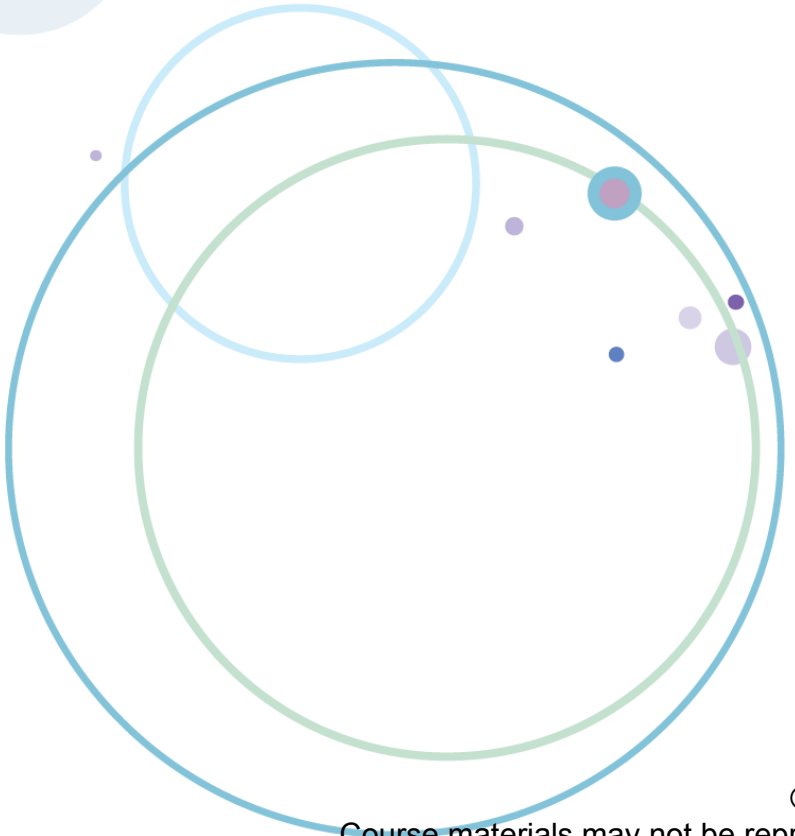


Example: Scalability with high availability policy

- Scalability with high availability:
  - Creates an ME for every server in the cluster
  - One other server can host each ME (in addition to the one to which it is assigned)



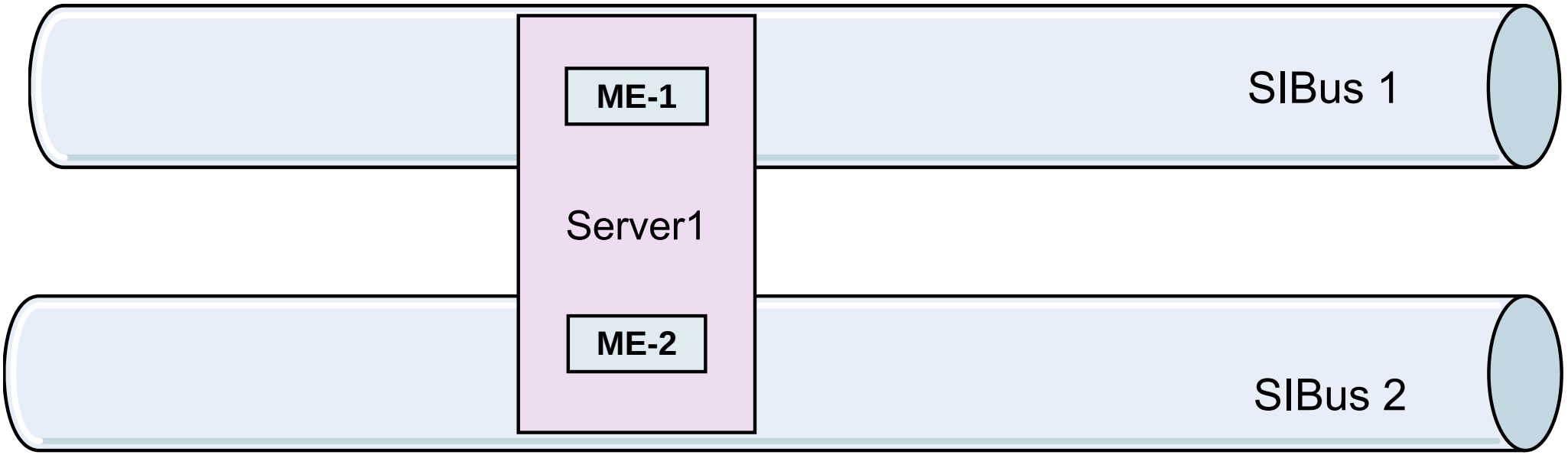
SIBus and messaging engine topologies



Messaging engine topology

- You can have multiple interconnected buses in a cell or stand-alone node (single server)
  - A common pattern is to have one SIBus in a stand-alone single server
- A topology that consists of just one cluster bus member by using a high availability policy is adequate for many applications
  - This results in a single ME
- Advantages in creating more than one ME are:
  - Spreading messaging workload across multiple servers
  - Placing message processing close to the applications that are using it
  - Improving availability in the face of system or link failure
  - Accommodating firewalls or other network restrictions that limit the ability of network hosts all to connect to a single ME

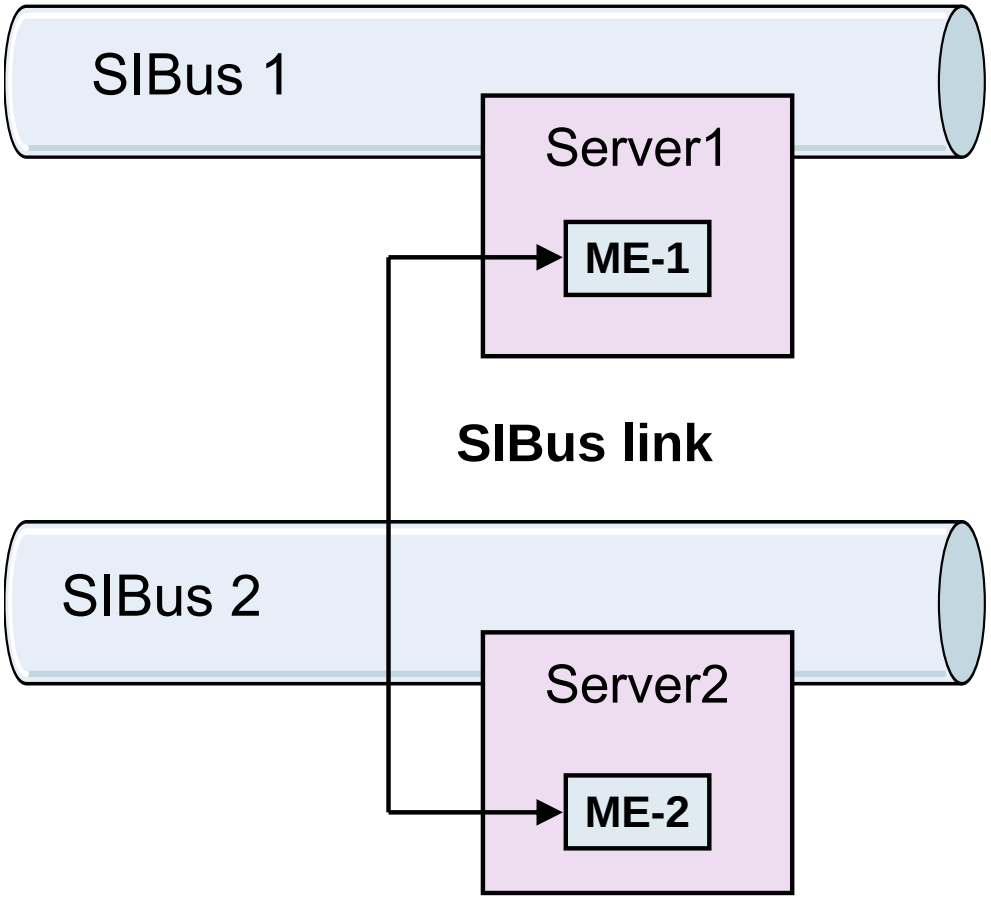
SIBus and MEs in a stand-alone server



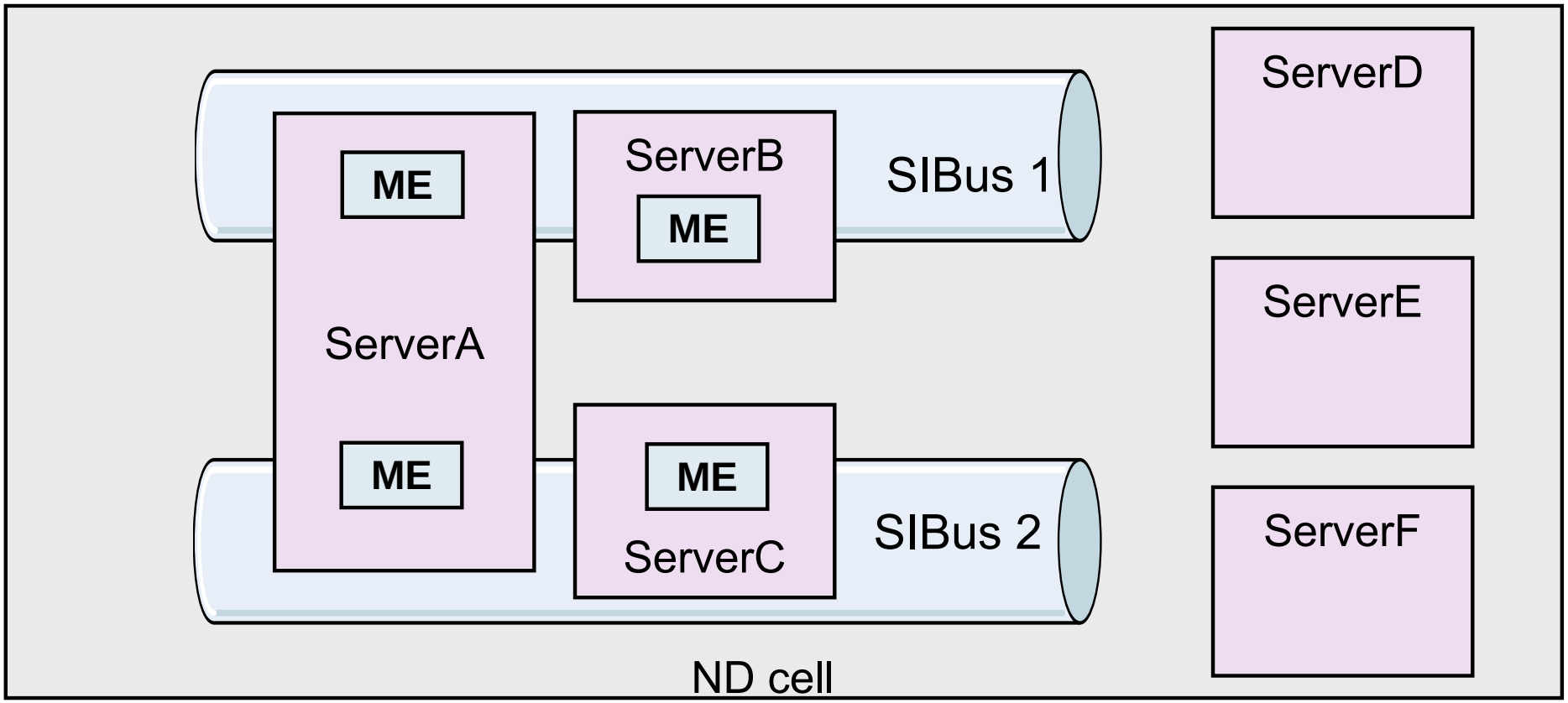
- A stand-alone server can be a member of multiple **buses**
- When the server is made a bus member, a **messaging engine** is created

Bus topology: Stand-alone servers

- An enterprise might deploy multiple interconnected messaging buses for organizational reasons
  - For example, separately administered buses for each department
- A bus can connect to other buses, which are known as **foreign buses**
- The administrator can create
  - An SIBus link
  - A WebSphere MQ link



SIBus and MEs in a Network Deployment cell

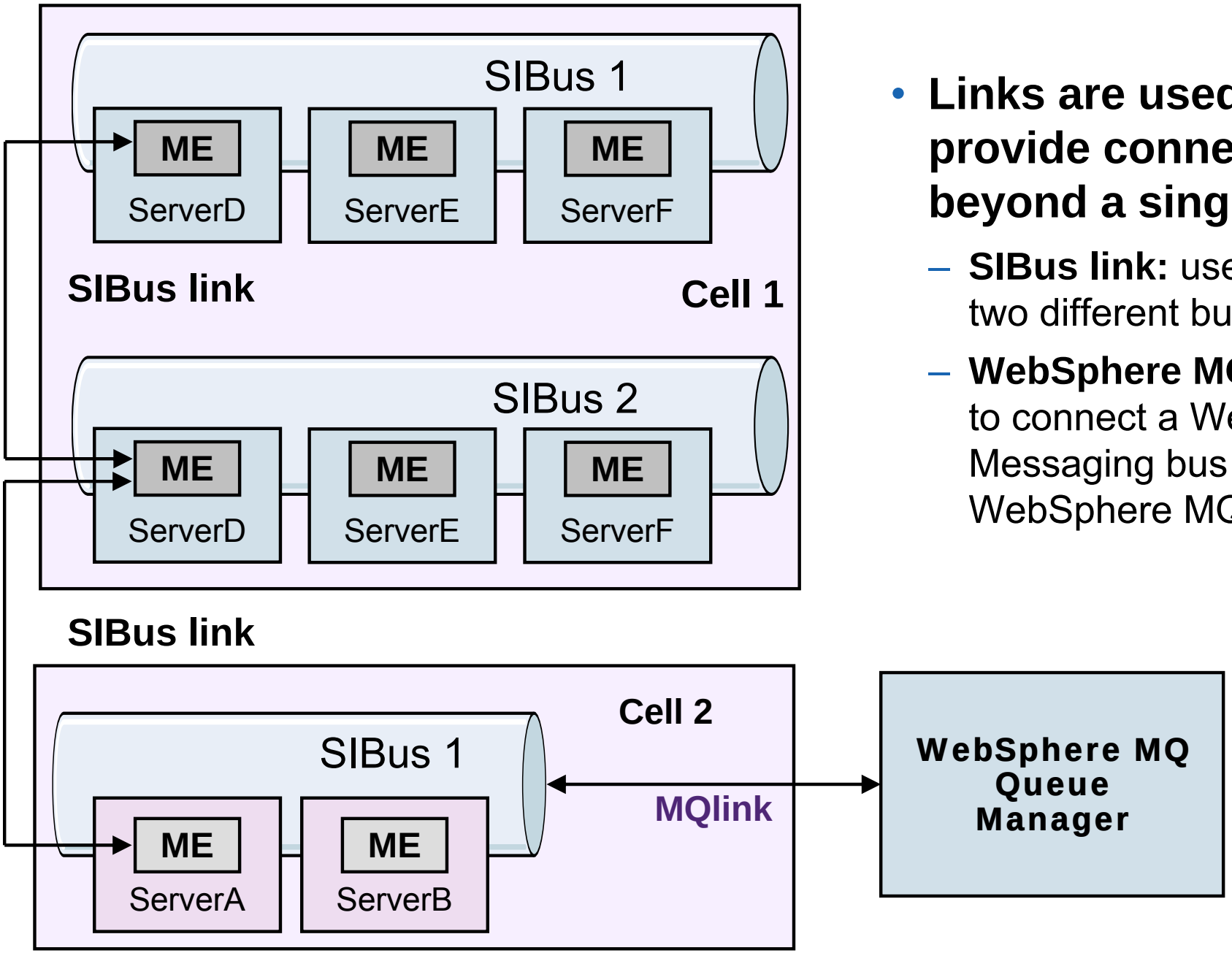


- A WebSphere Application Server ND cell can have multiple **buses**
- Each bus can have servers and clusters as **bus members**
- When a server, or cluster, is made a bus member, a **messaging engine** is created
- The cell can have servers that are not bus members

Network Deployment cell topologies (1 of 2)

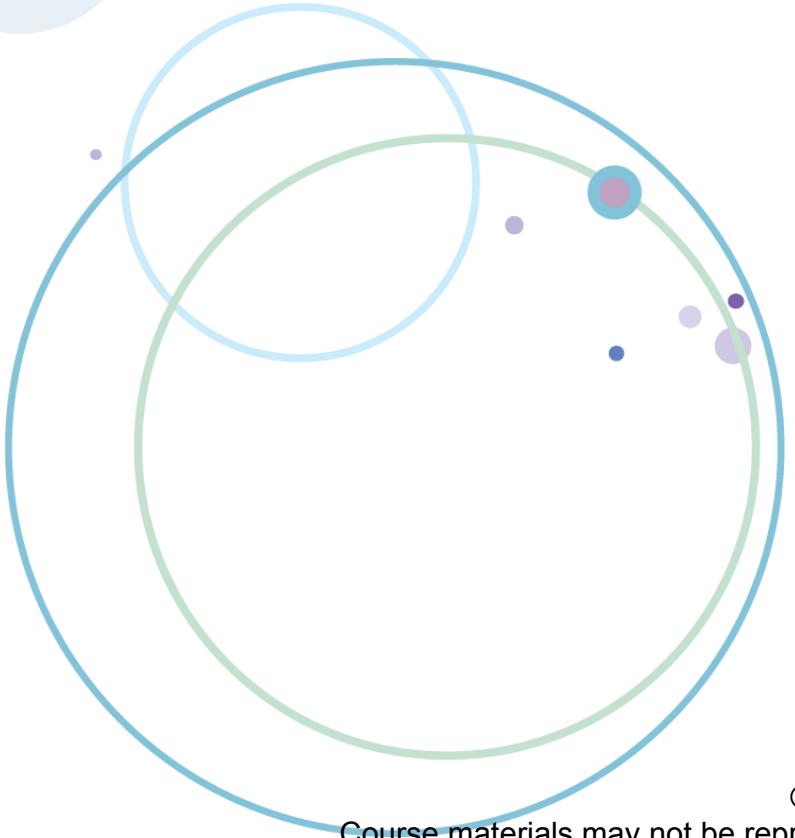
- In WebSphere Messaging, the administrative unit is the cell
  - Assumes uniform access to all MEs within the cell
  - All MEs on a bus are fully interconnected
  - A cell can host multiple buses
  - A bus in one cell can have a link to a bus in another cell

Network Deployment cell topologies (2 of 2)



- **Links are used to provide connectivity beyond a single bus**
  - **SIBus link:** used to connect two different buses
  - **WebSphere MQ link:** used to connect a WebSphere Messaging bus and a WebSphere MQ network

Additional messaging considerations

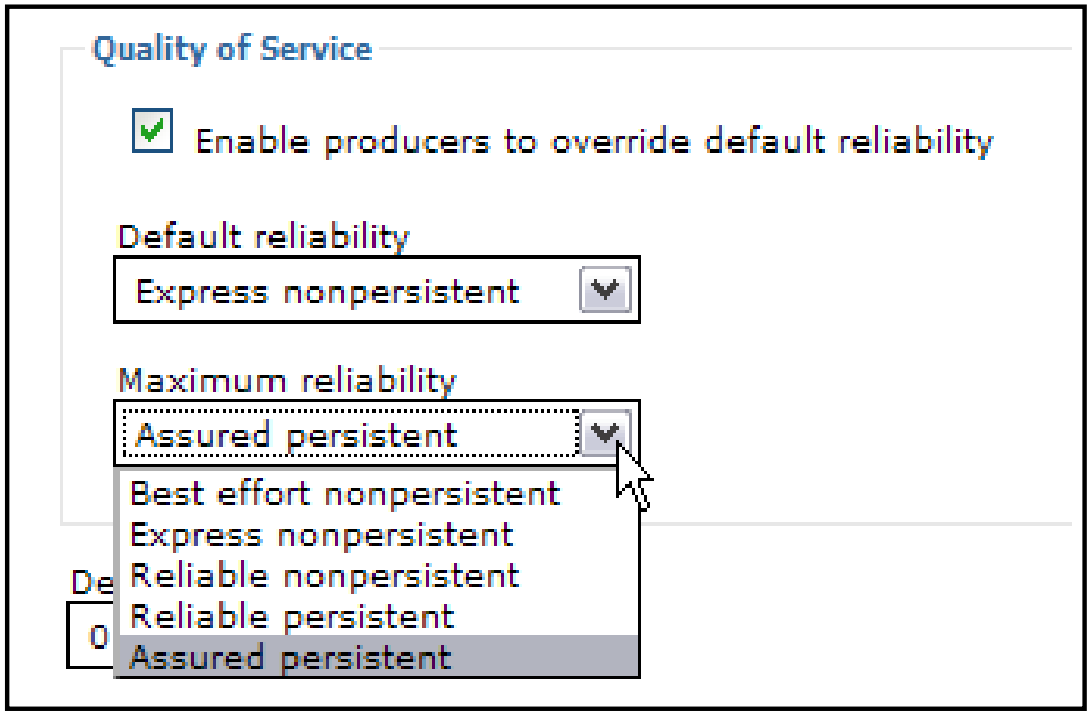


Additional messaging considerations

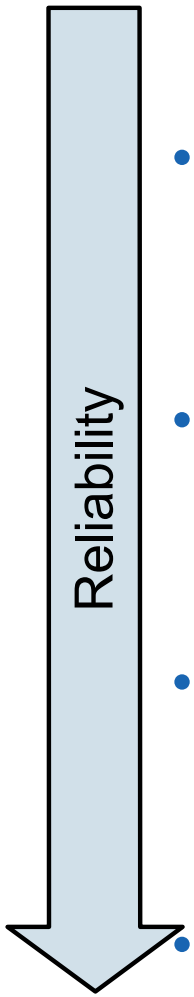
- For a complete configuration of WebSphere Messaging, consider:
  - Quality of service
  - Mediation
  - SIBus security
  - Interoperability with WebSphere MQ

SIBus destination quality of service

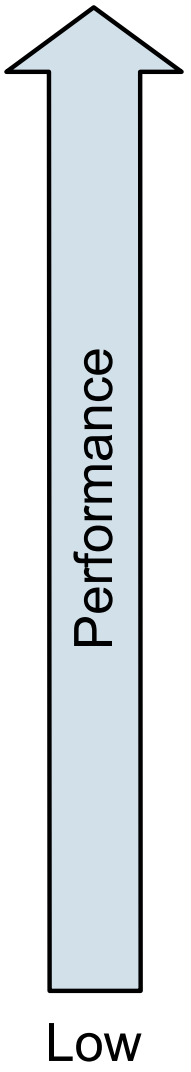
- Quality of service can be configured for SIBus destinations
- **Producers can override default reliability**: Select this option so that producers can override the default reliability that is set on the destination
- **Default reliability**: The reliability that is assigned to a message that is produced to this destination when the producer does not set an explicit reliability
- **Maximum reliability**: The maximum reliability of messages that this destination accepts



Destination quality of service for reliability

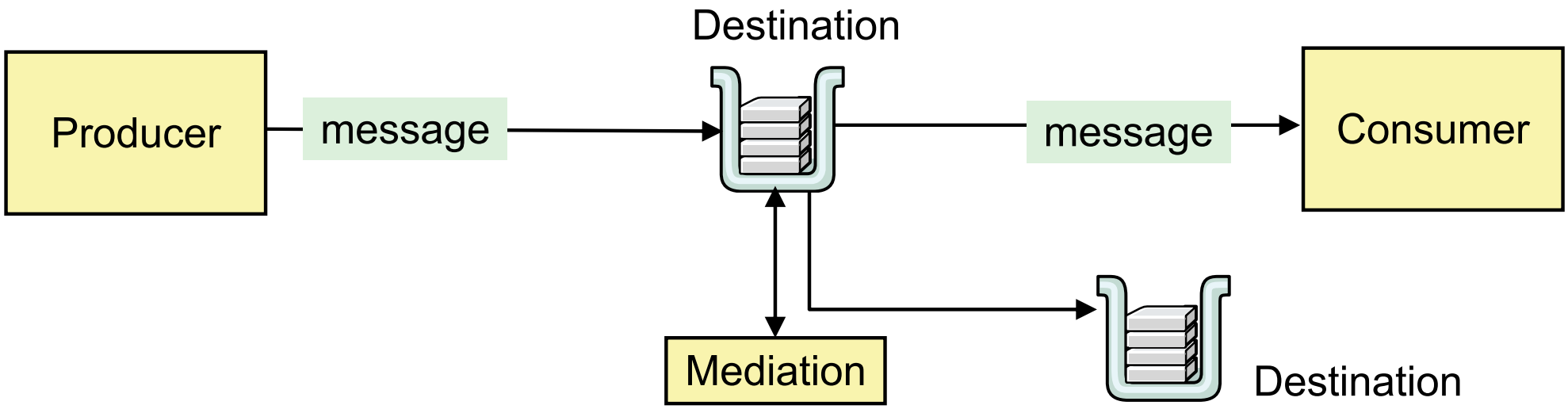


- BEST_EFFORT_NONPERSISTENT
  - Messages are never written to disk
- EXPRESS_NONPERSISTENT
  - Messages are written asynchronously to persistent storage if memory cache overruns
- RELIABLE_NONPERSISTENT
  - Same as EXPRESS_NONPERSISTENT, except you have a low-level acknowledgement message
- RELIABLE_PERSISTENT
  - Messages are written asynchronously to persistent storage during normal processing
- ASSURED_PERSISTENT
  - Highest degree of reliability where assured delivery is supported



What is a mediation?

- **Mediation:** The ability to manipulate a message as it traverses the messaging bus (destination)
  - Transform the message
  - Copy or reroute the message to a different destination, or sequence of destinations
  - Allow interaction with nonmessaging resource managers (for example, databases)
- Mediations are attached administratively to a destination



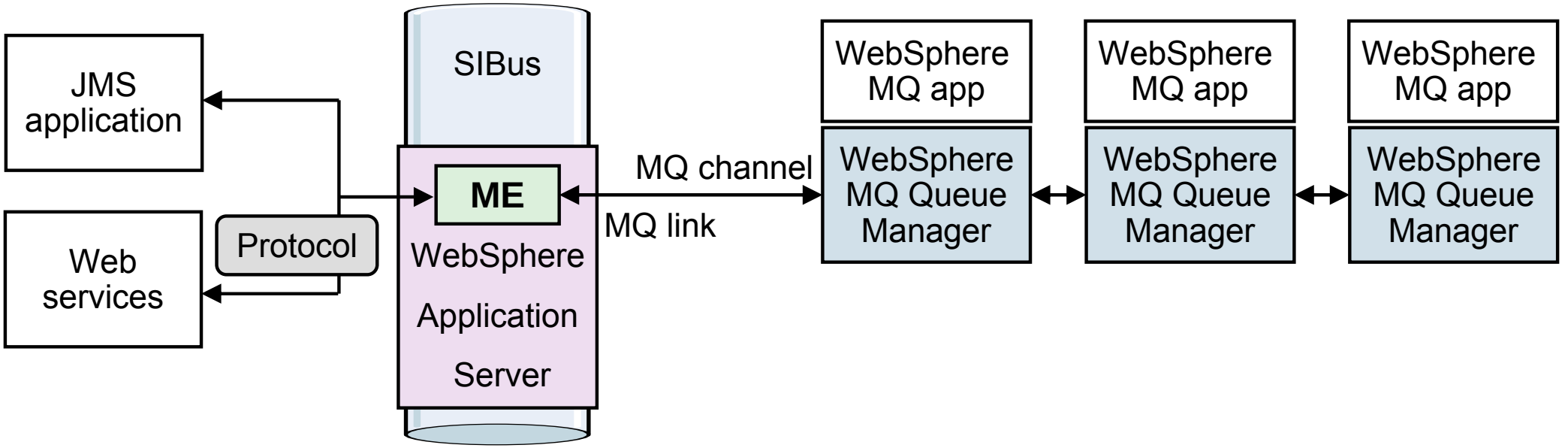
SIBus security

- SIBus security can be enabled when the bus is created
- Authentication:
  - Inherit cell level security domain (default)
  - Use a selected security domain
- Authorization: available roles for users and groups
  - Bus connector: can connect to the bus
  - Sender: can send (produce) messages to the destination
  - Receiver: can read (consume) messages from the destination
  - Browser: can read (non-destructive) messages from the destination
- Secure message transportation
  - Encrypt transport channel with SSL

Interoperating with a WebSphere MQ network

- WebSphere Application Server can interoperate with WebSphere MQ in the following ways:
  - By using the WebSphere MQ messaging provider to configure WebSphere MQ as an external JMS provider
  - By using the default messaging provider and a WebSphere MQ link to connect a service integration bus to a WebSphere MQ network
  - By using the default messaging provider and a WebSphere MQ server to integrate WebSphere MQ queues into a bus

Connect SIBus to WebSphere MQ Network



- You use a WebSphere MQ link to connect an SIBus to a WebSphere MQ network
- WebSphere MQ treats the SIBus ME as a virtual queue manager
- SIBus treats WebSphere MQ network as a foreign bus
- WebSphere MQ applications can send messages to queues hosted on WebSphere Application Server SIBuses
- WebSphere Application Server messaging applications can send messages to WebSphere MQ queues

Unit summary

Having completed this unit, you should be able to:

- Describe what WebSphere Messaging is and how it is used
- Describe messaging components such as JMS providers, the service integration bus (SIBus), and messaging engines
- Configure resources to support messaging applications such as queues, topics, and connection factories
- Implement various clustered messaging engine policies for high availability and scalability
- Create links to foreign buses and WebSphere MQ
- Describe how JMS and WebSphere MQ use the SIBus to support application messaging services

Checkpoint questions

1. Name two types of JMS destinations.

2. What can be added as a member of an SIBus?

3. True or False: An application server always has at least one message engine that is running.

4. True or False: Producer and consumer applications connect to the SIBus through a Messaging Engine.

Checkpoint answers

1. Name two types of JMS destinations.

   Queues and topics

2. What can be added as a member of an SIBus?

   Application servers and clusters

3. True or False: An application server always has at least one message engine that is running.

   False

4. True or False: Producer and consumer applications connect to the SIBus through a Messaging Engine.

   True

Exercise 11

Configuring the service integration bus

Exercise objectives

After completing this exercise, you should be able to:

- Explain some of the design decisions that are required to set up a messaging environment
- Explain how to configure the service integration bus, messaging engines, and bus destinations in WebSphere Application Server
- Explain how to set up basic SIBus security
- Explain how to configure JMS queues, connection factories, and activation specifications for message-driven beans
- Explain how to install and test the messaging features of the two example programs

Exercise: Messaging tasks