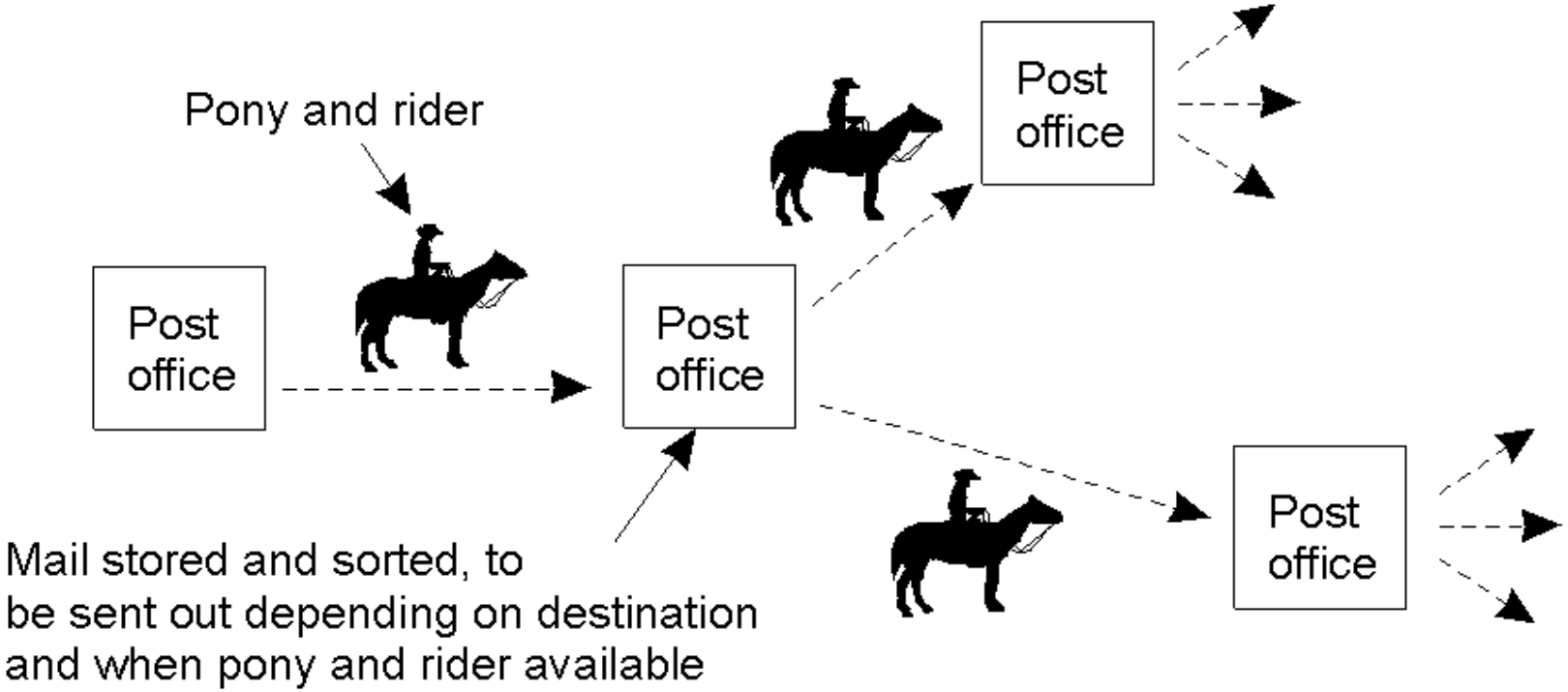
Lecture 6:
Message Oriented Middleware

- Messaging introduction
- Java Message Service (JMS)
- Messaging patterns
  - Message routing
  - Message transformation
  - Message system management

Message oriented communication

- Remote Procedure Call (RPC) and Distributed Objects (DO) extend familiar (single-process) programming model to the distributed environment. They make *communication* transparent or *implicit*.
- Message Oriented Middleware (MOM) introduces a new model for programming distributed applications, based on *explicit* communication.
- Communication mechanisms may be *transient* or *persistent*, *synchronous* or *asynchronous*

Persistent vs. Transient



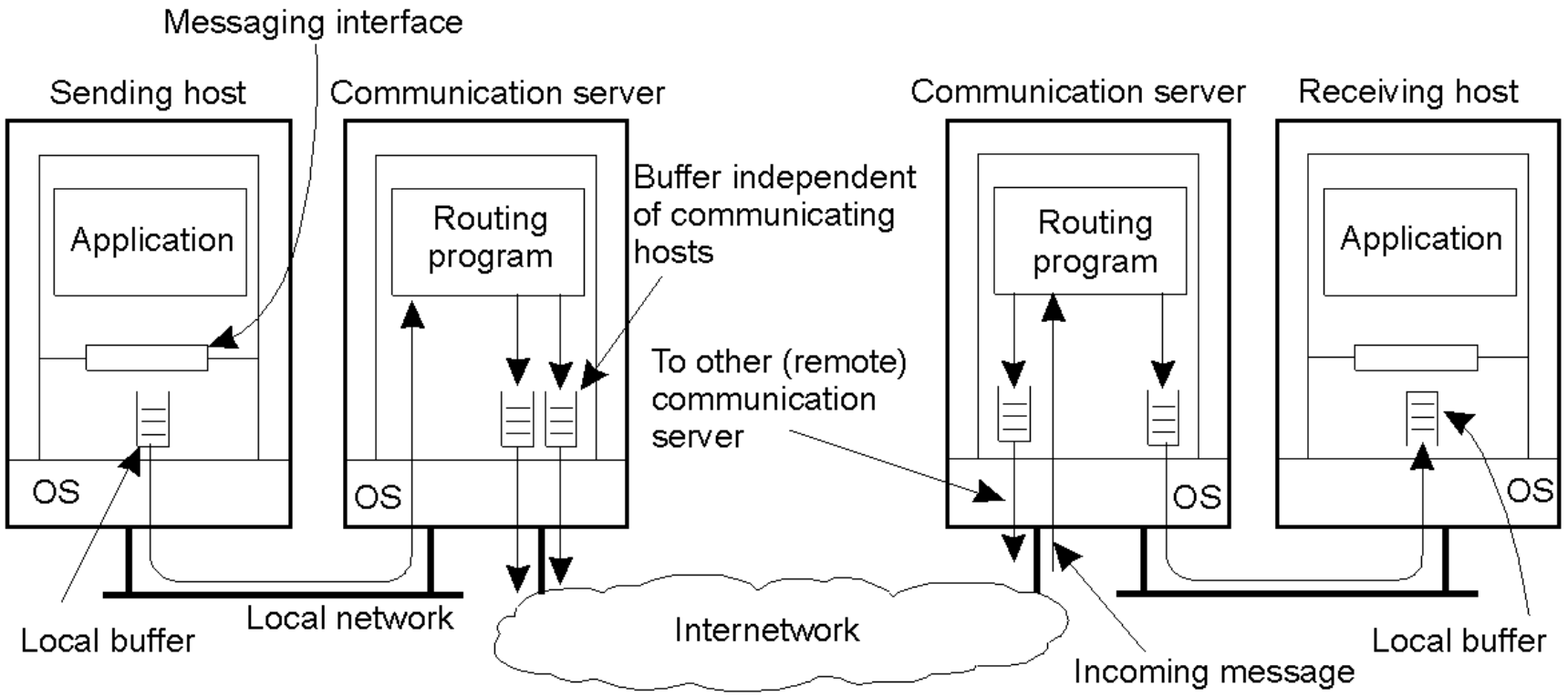
- *Persistent* communication: Message is stored in the communication system.
- *Transient* communication: Message is stored only as long as sender an receiver are executing

Synchronous vs. Asynchronous

- Asynchronous: Sender continues immediately after submission
- Synchronous: Sender is blocked until:
  - Buffered at receiving host
  - Delivered to receiver
  - Receiver has processed the message

➔ Various combinations of persistence and synchronicity.

Model of Message Oriented Middleware

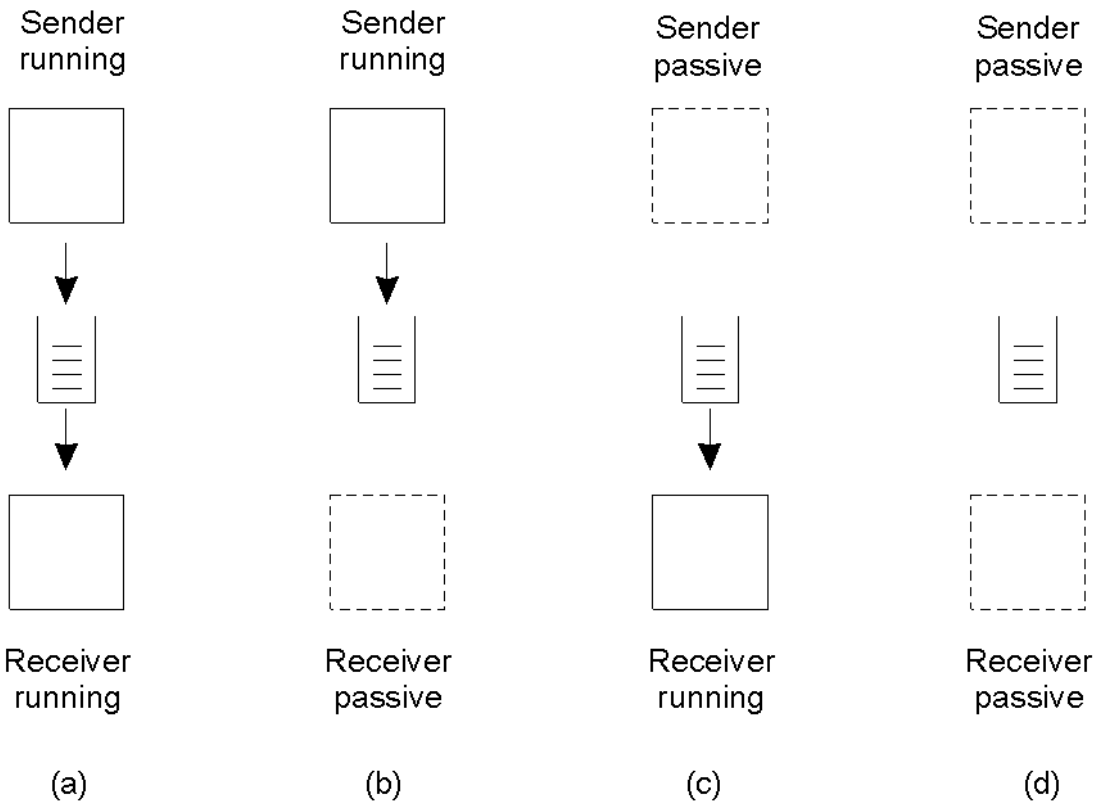


MOM characteristics

- Message queuing
- Persistent (asynchronous) communication
- Intermediate storage capacity for messages in the communication network
- Communication may take minutes (not ms)
- Basic idea: insert message into queue

→ Loosely-coupled systems!

Message-queuing model



- Four combinations for *loosely-coupled* communications using queues (for persistent communication).
- (email is a message queuing system.)

Message-Queuing Model Primitives

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

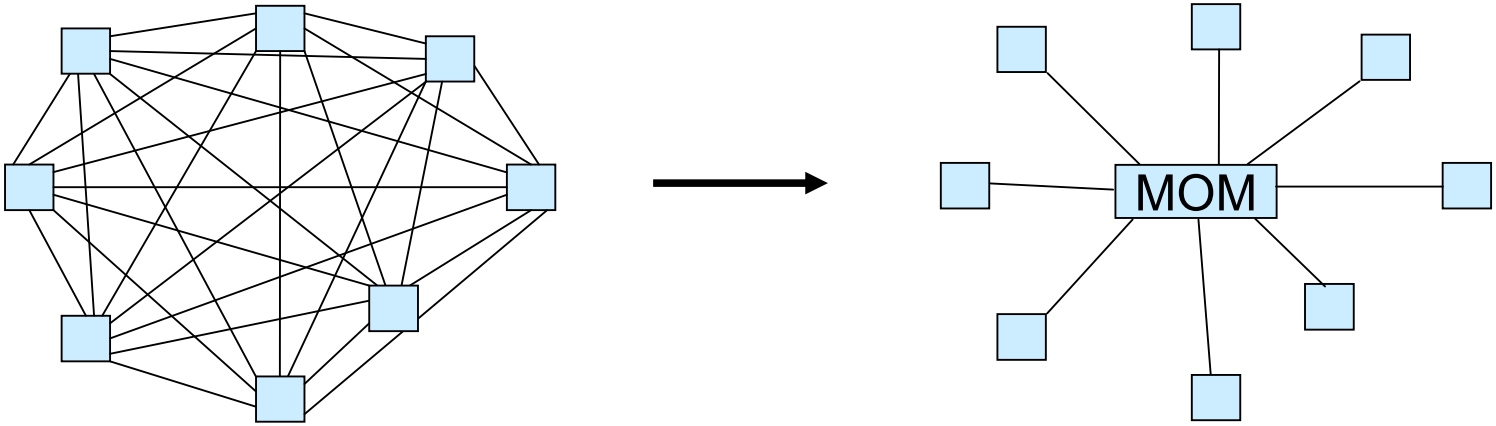
Basic interface to a queue in a message-queuing system.

Message format

- Enterprise Application Integration (EAI), legacy systems
- Each approach to replace N technologies by one technology usually ends with N+1 technologies

➔ Learn to live with different formats and provide the means to make conversion as easy as possible

Reduction of communication formats

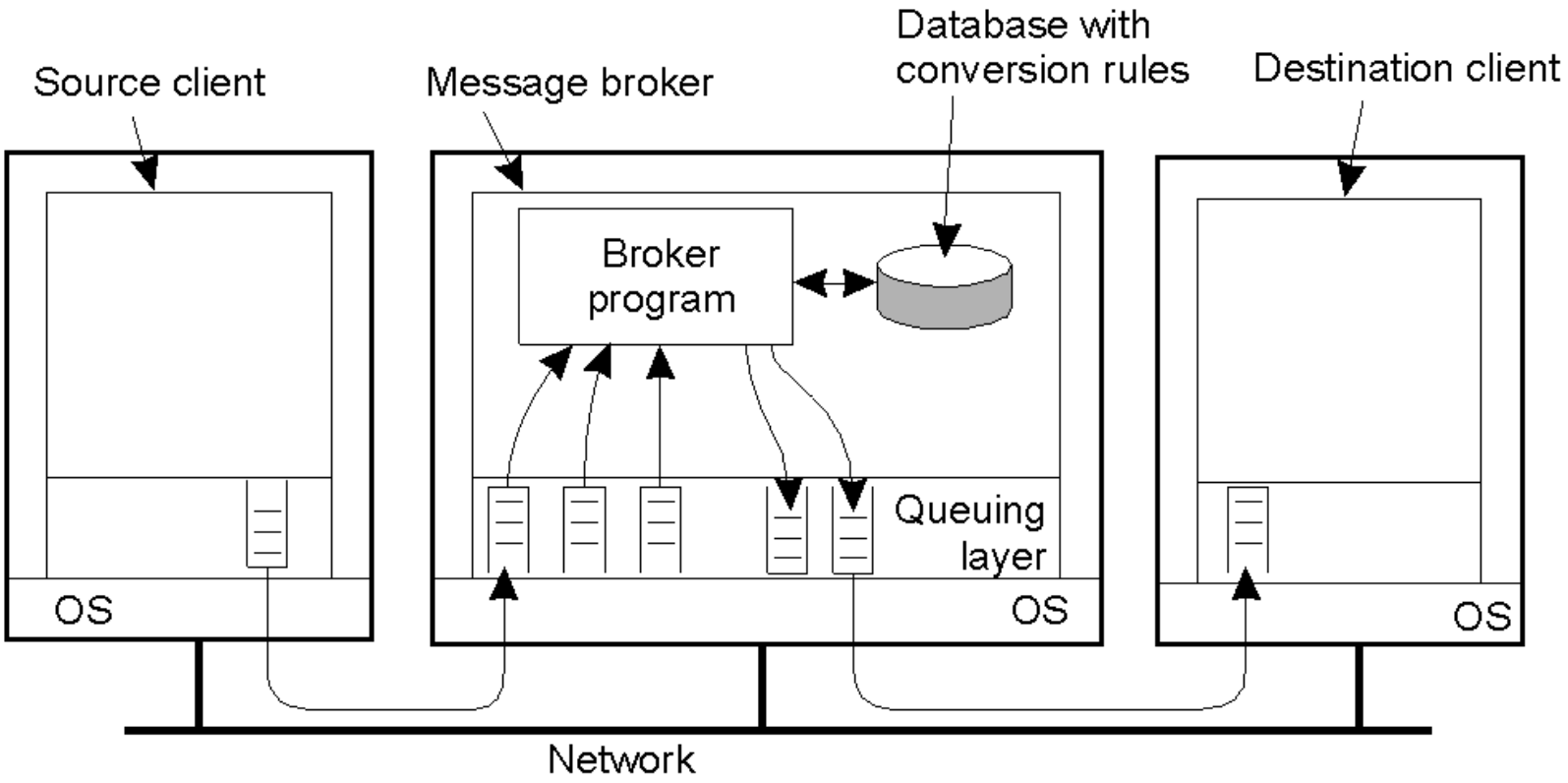


Potentially N*(N-1)/2 communication technologies and/or message formats.

Potentially N communication technologies and/or message formats.

- If we have several systems using different communication formats (technologies and/or message formats), using MOM reduces potentially N*(N-1)/2 formats to potentially N formats.

Message brokers



The general organization of a message broker in a message-queuing system. Message broker can “adapt” format of messages before delivering them.

Lecture 6:
Message Oriented Middleware

- Messaging introduction
- Java Message Service (JMS)
- Messaging patterns
  - Message routing
  - Message transformation
  - Message system management

Java Message Service (JMS)

- Message-Oriented Middleware products are an essential component for integrating intra-company operations (Enterprise Application Integration, EAI).
- JMS provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages.
- JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.

JMS objectives and design considerations

- Several messaging solutions already existed before the development of JMS.
- JMS should support features of existing messaging products in order to allow interoperability, but
- should not be too complex and reduce portability due to supporting all the possible features.
- → Include functionality required to implement sophisticated enterprise applications.

JMS definitions (1)

- *JMS provider:* is the entity that implements JMS for a messaging product
- *JMS clients:* should have a consistent API for creating and working with messages that is independent of the JMS provider.
- *Non-JMS clients:* use a message system's native client API instead of JMS

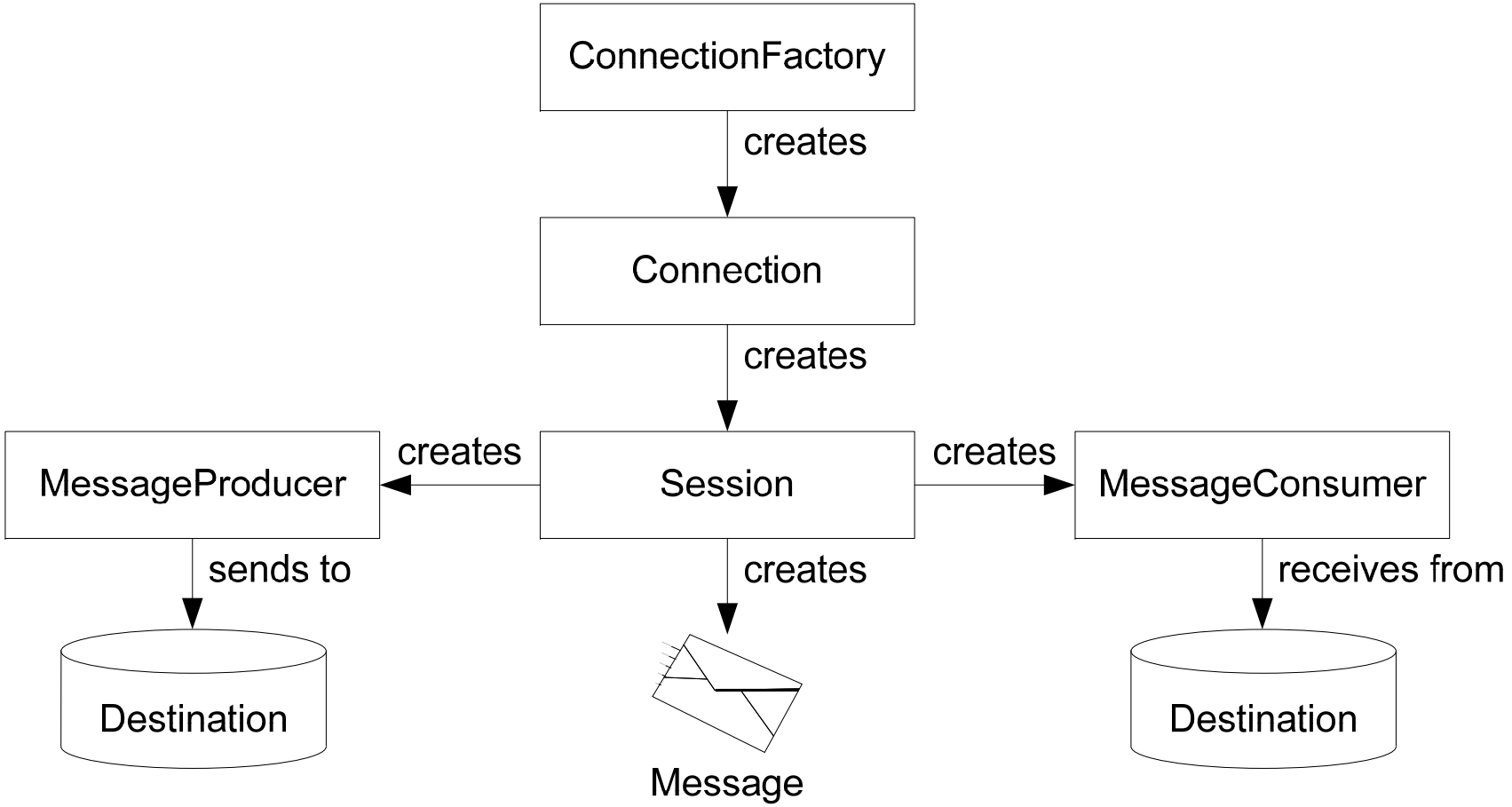
JMS definitions (2)

- *JMS message:* JMS defines a set of message interfaces that can be used for exchanging information.
- *JMS domains:* can be broadly classified as either *point-to-point* or *publish-subscribe* systems.
- *Administered objects:* preconfigured JMS objects created by an administrator for the use of clients: ConnectionFactories, Destinations.

JMS does not define

- Load balancing/fault tolerance
- Error/advisory notification
- Administration
- Security
- Wire protocol
- Message type repository

JMS Overview



Developing a JMS Client

- Lookup **`ConnectionFactory`** from JNDI.
- Lookup **`Destination`**(s) from JNDI
- Use **`ConnectionFactory`** to create a JMS **`Connection`** with message delivery inhibited.
- Use the **`Connection`** to create JMS **`Sessions`**.
- Use **`Session`** and **`Destinations`** to create the **`MessageProducers`** and **`MessageConsumers`** needed.
- Tell the **`Connection`** to **`start()`** the delivery of messages.

→ Client is ready with basic JMS setup needed to produce and consume messages.

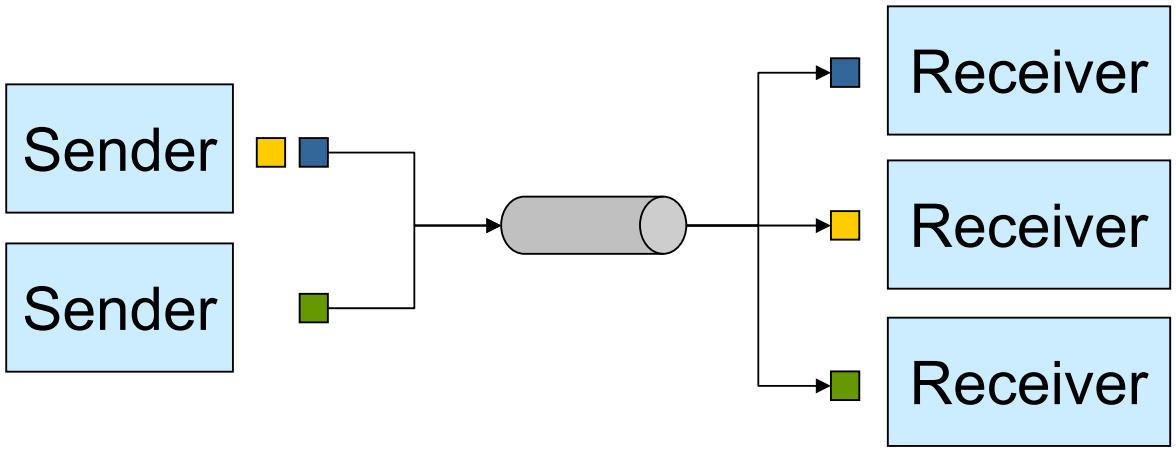
JMS message model

- JMS message split into message header and message body.
- Message header with mandatory header fields, for example:
  - *JMSMessageID*: identifies a JMS message
  - *JMSTimestamp*: set by the JMS provider while sending
  - *JMSPriority*: message priority allows „overtaking“ messages
  - *JMSDeliveryMode*: persistent (exactly-once-delivery) or transient (at-most-once-delivery).
  - *JMSExpiration*: Messages can be dropped if their expiration time is reached.

JMS message model

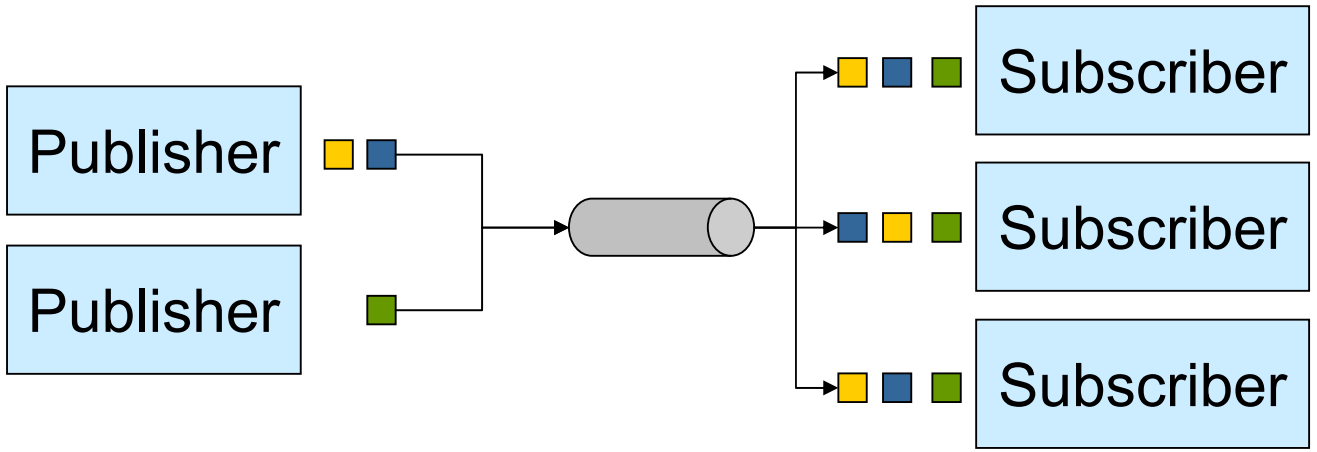
- Optional message header properties
  - Can be set by the JMS client with application-specific name.
  - Different data types (String, Integer, Boolean, …)
  - Example:
    message.setStringProperty(“productCategory”, “computer”);
- Message body with different types
  - TextMessage: contains arbitrary text.
  - MapMessage: contains name-value pairs.
  - ObjectMessage: contains a serializable object.
  - BytesMessage: sequence of uninterpreted bytes (byte[]).
  - StreamMessage: provides stream-based interface to read/write message content.

JMS Queue – point-to-point



- M senders, N receivers
- One message only delivered to a single receiver (*point-to-point communication*).
- Increases scalability
  - Can be used for load balancing between the receivers.

JMS Topic – publish/subscribe paradigm



- M Publisher, N Subscribers
- One message delivered to all subscribers.
- No guarantee on message ordering.

JMS communication models

- Point-to-point communication via JMS queues
  - If no receiver listens at the queue, messages are kept until they can be delivered or expire.
- Publish/subscribe communication via JMS topics
  - *Ordinary subscribers* only receive messages published while the subscriber is active.
  - Messages for *durable subscribers* published while the subscriber is inactive will be delivered to the subscriber the next time it becomes active.
  - Messages are only kept for a durable subscriber if
    - the topic is durable (allows to store messages) and
    - the message is persistent.

JMS message delivery

- Messages delivered to the application have to be acknowledged to be removed from the JMS system.
- Normally, a consumer *fully processes each message before acknowledging* its receipt to JMS. This insures that JMS does not discard a partially processed message due to machine failure, etc.
- Full processing is accomplished by using either a *transacted or CLIENT_ACKNOWLEDGE session*.
- Unacknowledged messages redelivered due to system failure must have the *JMSRedelivered* message header field set by the JMS provider.