Messaging With Apache ActiveMQ

Bruce Snyder

bsnyder@apache.org

19 Nov 2008

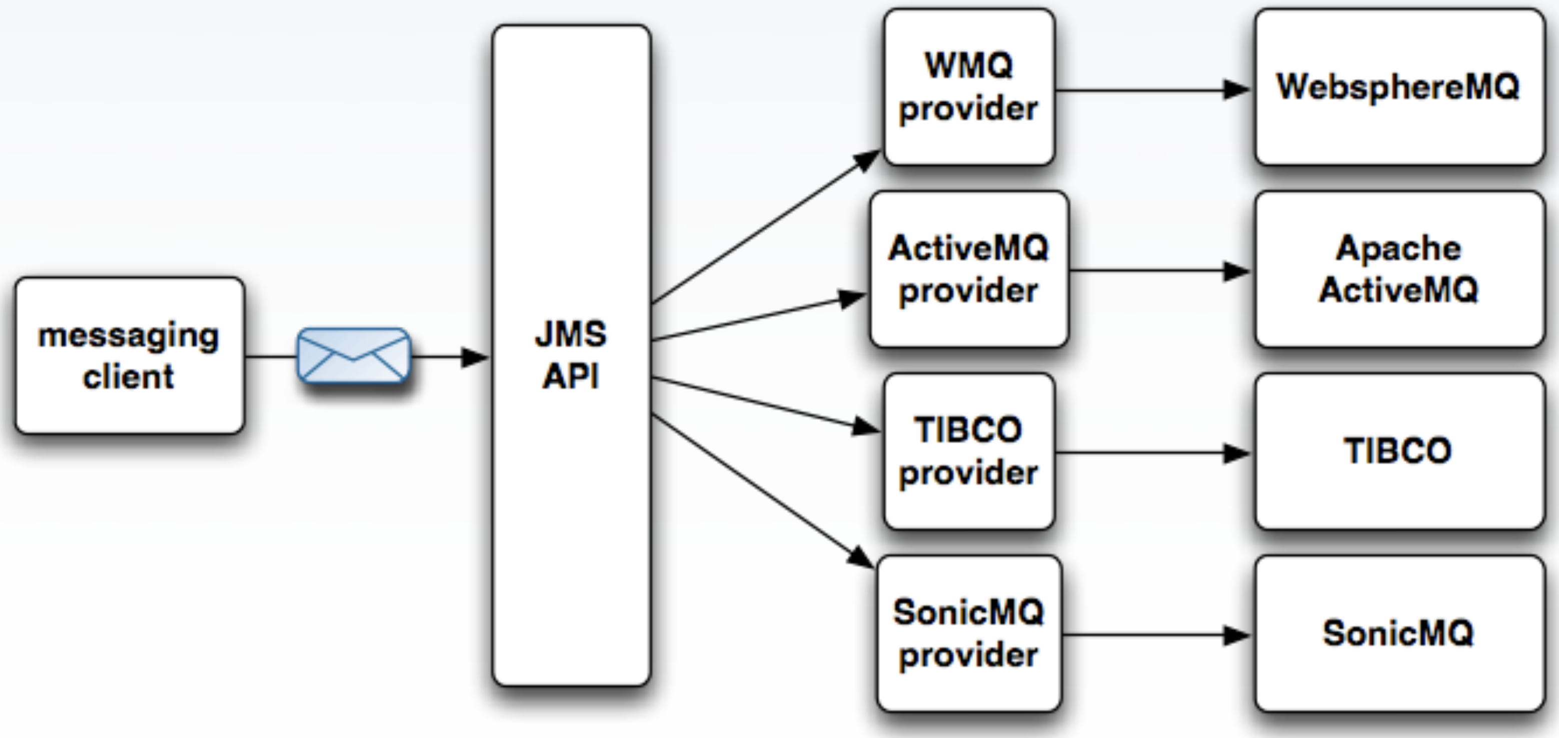
Malmo, Sweden

Do You Use JMS?

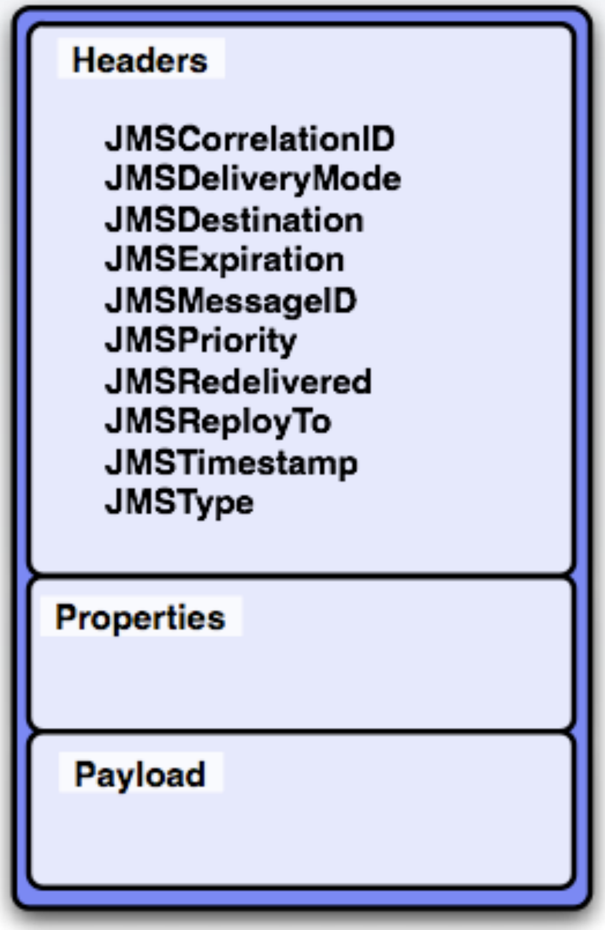
A Crash Course in Messaging

- JMS is:
  - An API for enterprise messaging
  - Included in Java EE
    - Also available stand alone
  - Loosely coupled
- JMS is not:
  - A message broker implementation

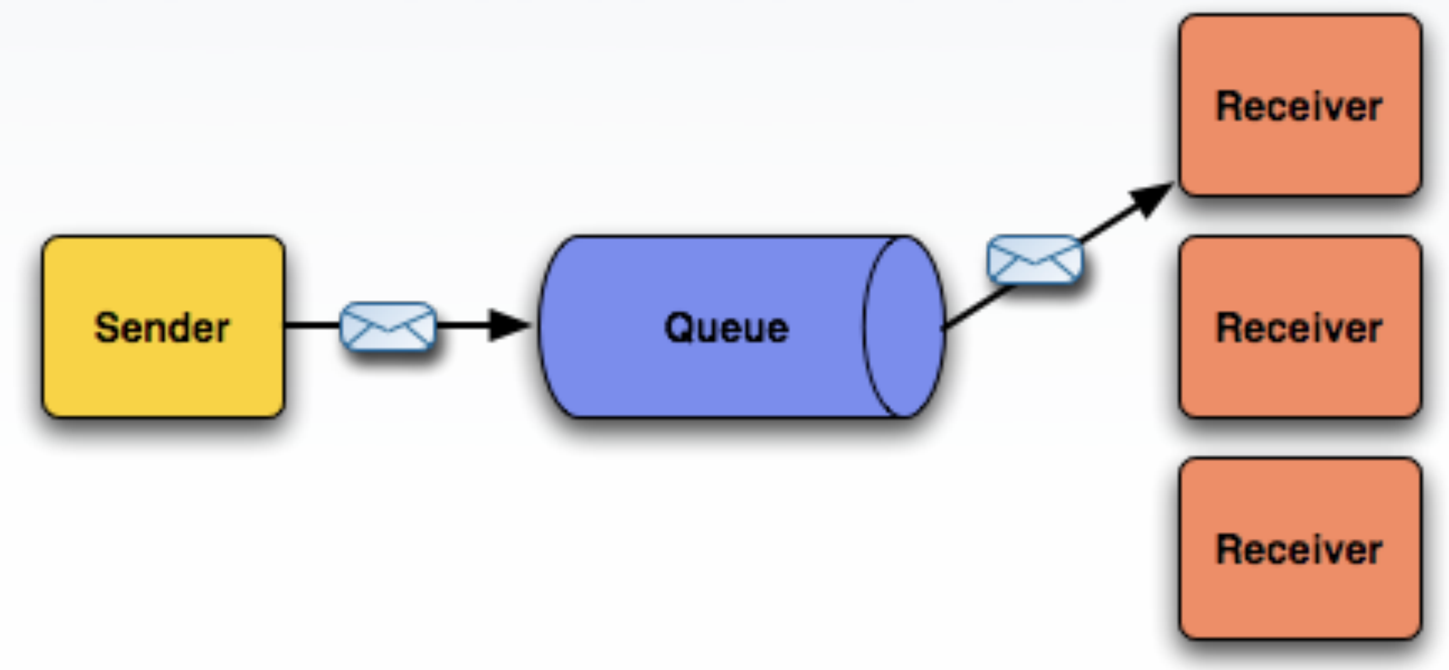
JMS Absracts Message Brokers



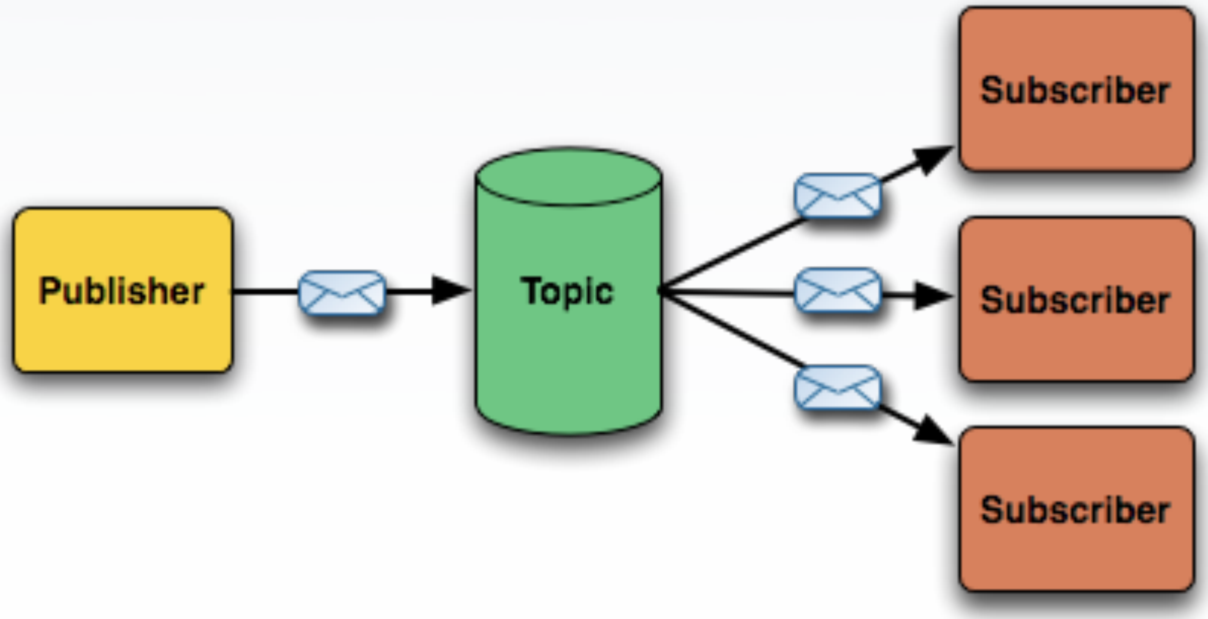
The JMS Message



Point-to-Point Messaging



Publish-Subscribe Messaging Domain



What is ActiveMQ?

- Message-oriented middleware
- Apache project
  - http://activemq.apache.org/
- Apache licensed
- JMS 1.1 compliant

- Goal:
  - Standards-based, message-oriented application integration across many languages and platforms

Examples Demo

Easily send and receive messages using the default examples

Configuration

(conf/activemq.xml)

ActiveMQ Uses URIs

```
<protocol>://<host>:<port>?<transport-options>
```

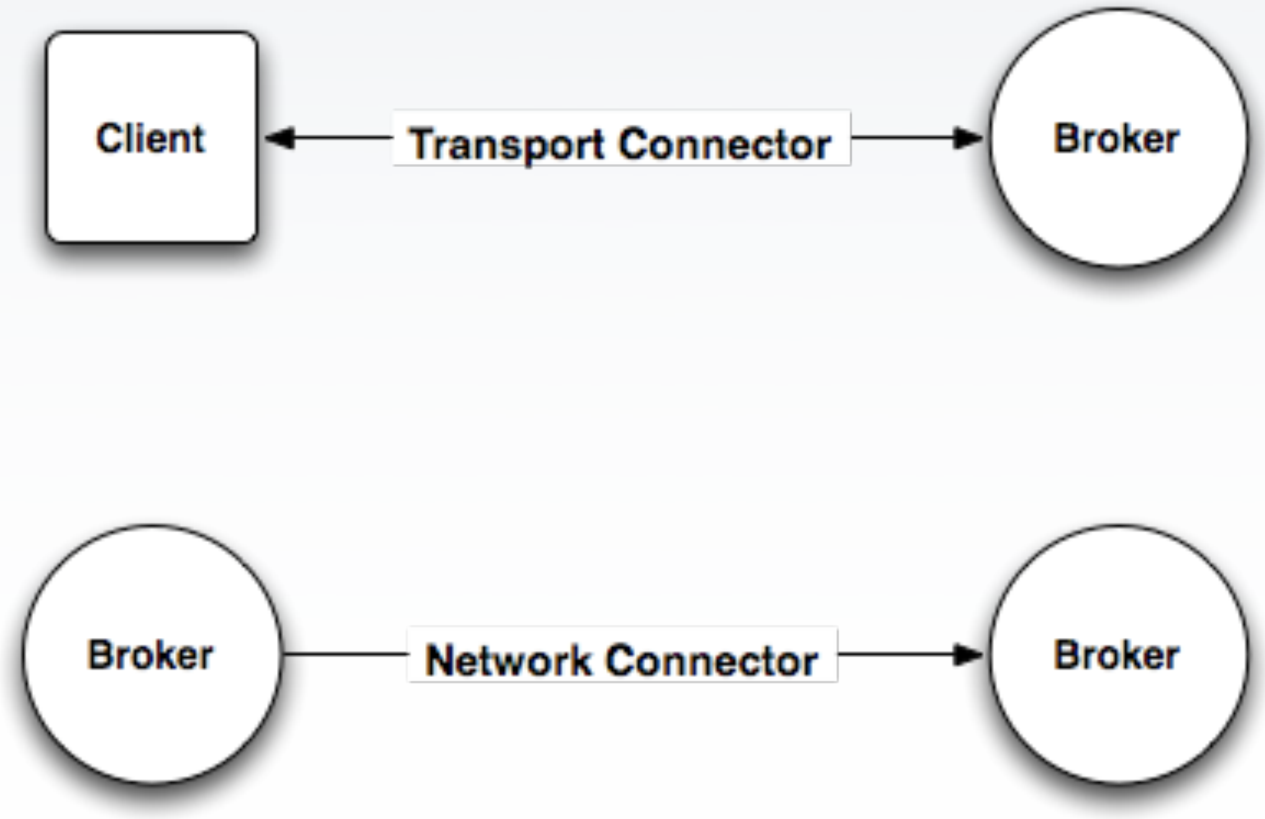
Example URIs

```
vm://localhost?broker.persistent=false

tcp://localhost:61616?jms.useAsyncSend=true

stomp://localhost:61613

failover:(tcp://host1:61616,tcp://host2:61616)?
initialReconnectDelay=100
```

Wire Formats

- OpenWire
  - The default in ActiveMQ; a binary protocol
  - Clients for C++, Java and .NET
- STOMP
  - Simple Text Oriented Messaging Protocol; a text based protocol
  - Clients for C, Javascript, Perl, PHP, Python, Ruby and more
- XMPP
  - The Jabber XML protocol
- REST
  - HTTP POST and GET
- AMQP
  - Not yet fully supported

Two Types of Transports



Transport Connectors

- Client-to-broker connections
  - Similar to JDBC connections to a database
- Protocols are supported:
  - TCP
  - UDP
  - NIO
  - SSL
  - HTTP/S
  - VM
  - XMPP

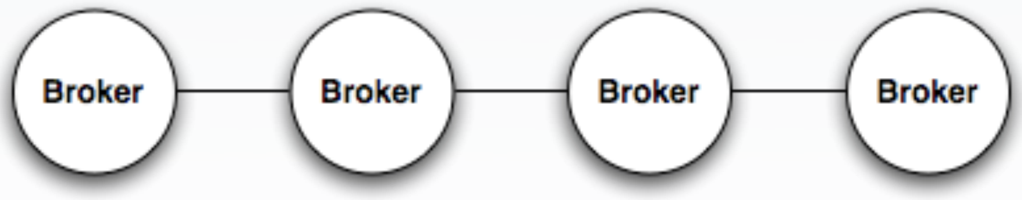
Network of Brokers

- Broker-to-broker connections
- Protocols supported:
  - Static
  - Failover
  - Multicast
  - Zeroconf
  - Peer
  - Fanout
  - Discovery

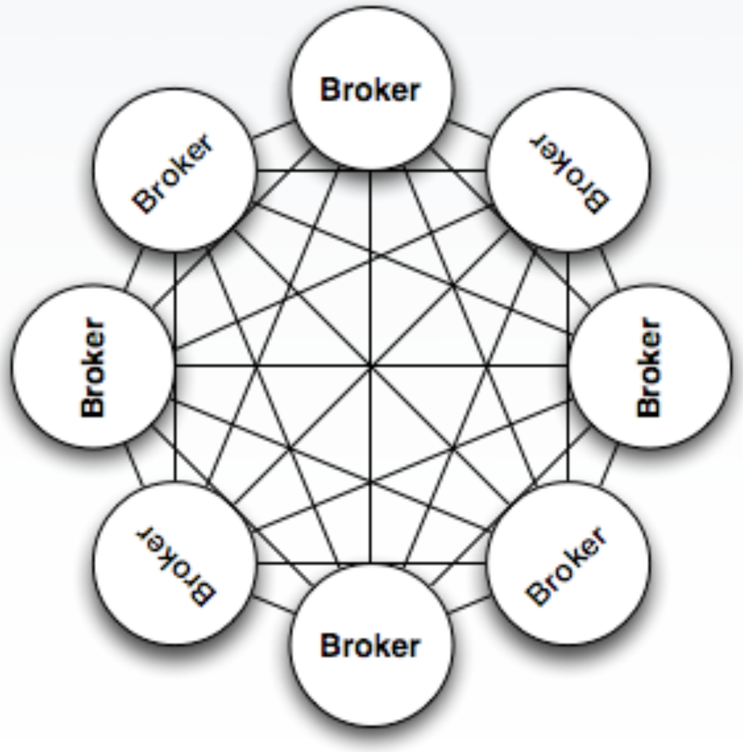
Networks of Brokers

- Provides large scalability
- ActiveMQ store-and-forward allows messages to traverse brokers
  - Demand-based forwarding
  - Some people call this distributed queues
- Many possible configurations or topologies are supported

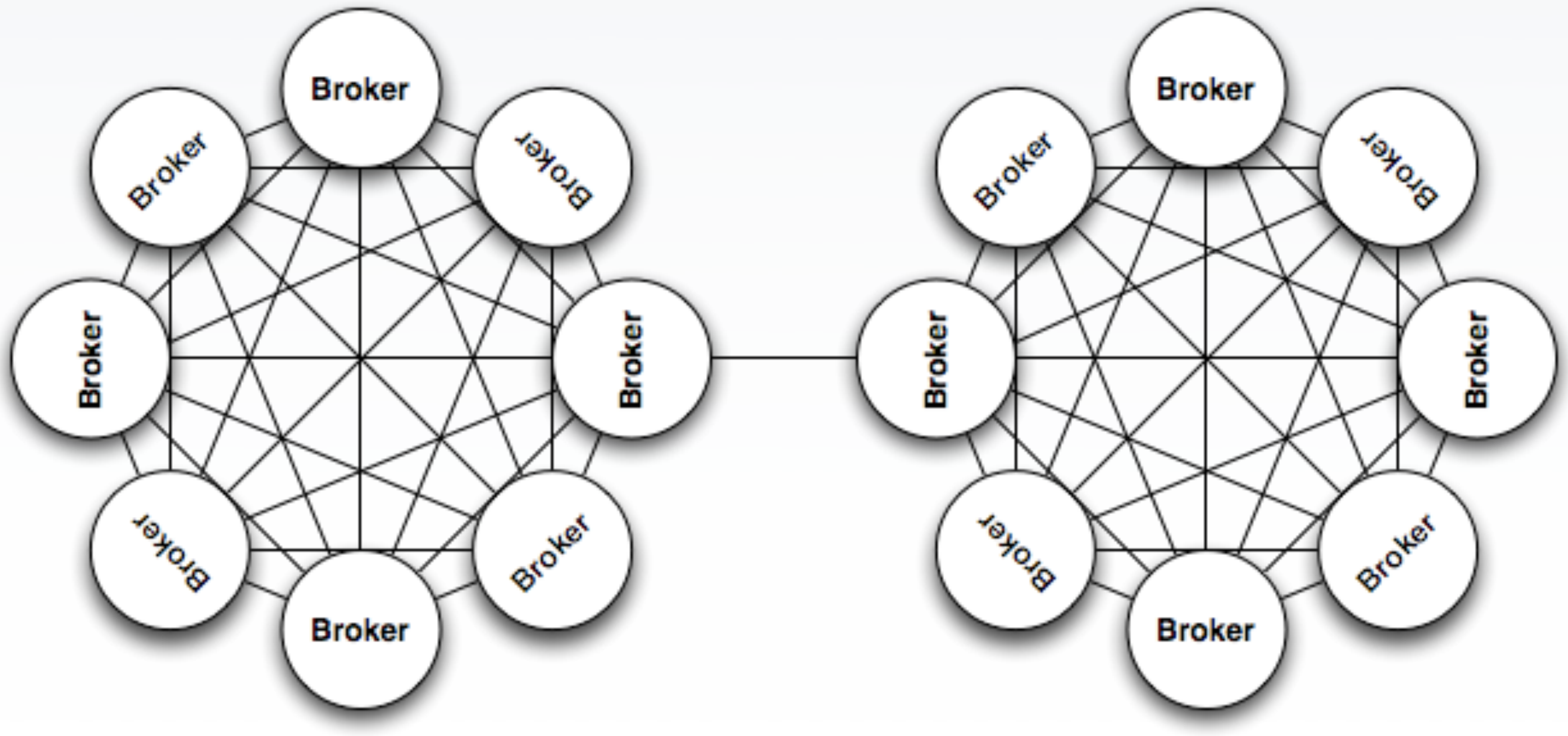
Topology Example



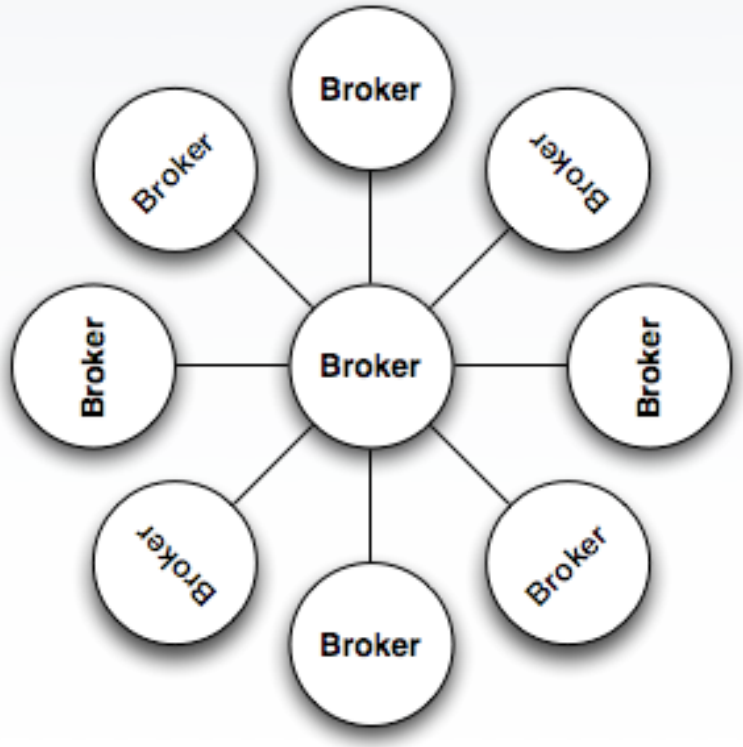
Topology Example



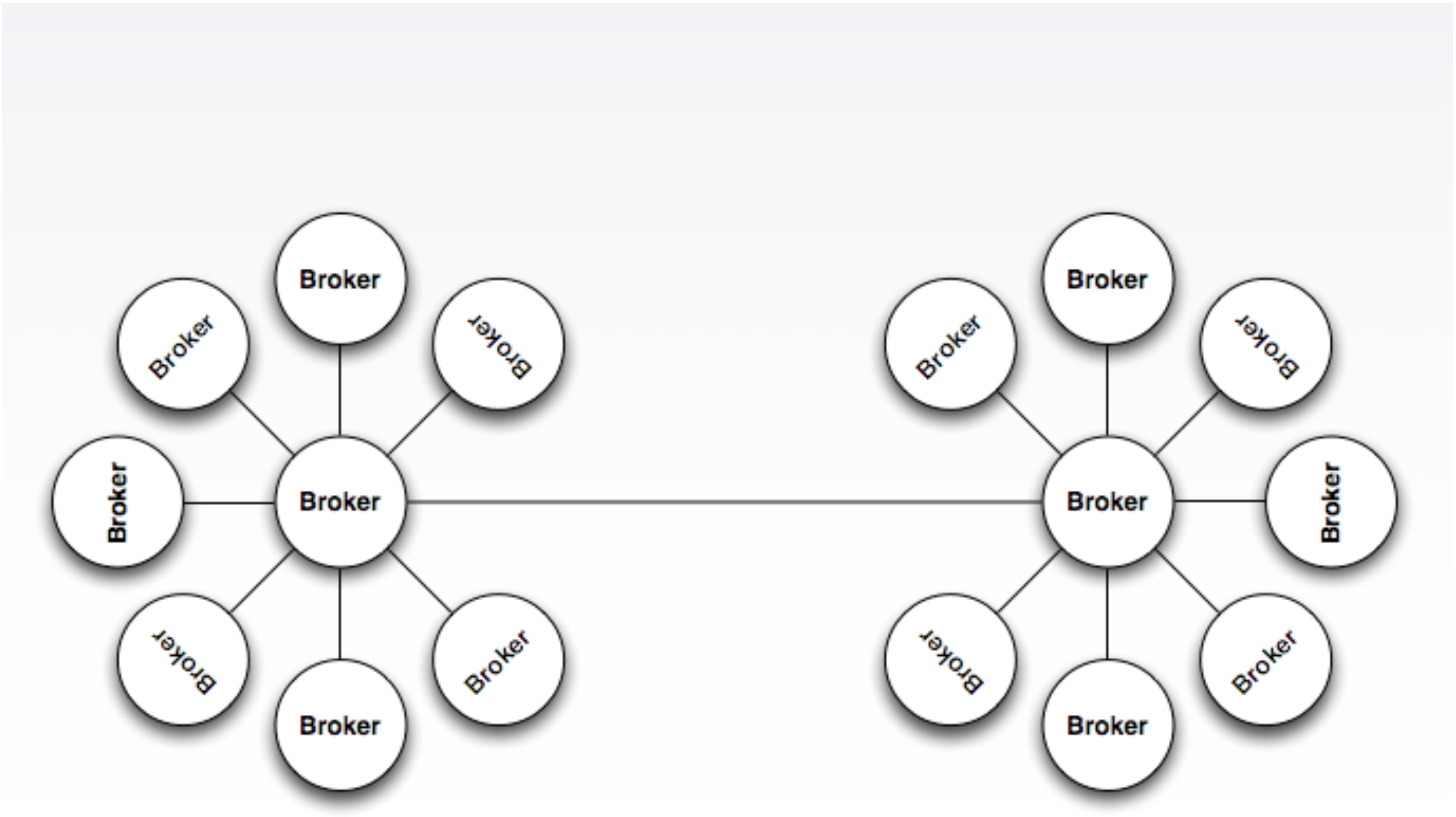
Topology Example



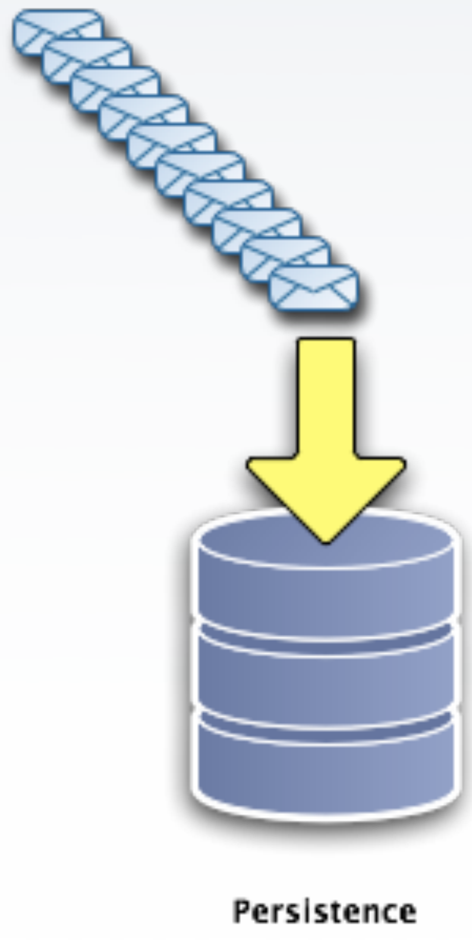
Topology Example



Topology Example



Message Persistence



AMQ Message Store

- Transactional message storage solution
- Fast and reliable
- Composed of two parts:
  - Data Store - holds messages in a transactional journal
  - Reference store - stores message locations for fast retrieval
- The default message store in ActiveMQ 5

Non-Journaled JDBC

- Transactional message storage solution
- Reliable but not fast
  - JDBC connection overhead is prohibitively slow

Journaled JDBC

- Transactional message storage solution
- Reliable and faster than non-journaled
- Two-piece store
    - Journal - A high-performance, transactional journal
    - Database - A relational database of your choice
- Default database in ActiveMQ 4.x is Apache Derby

Message Cursors

- Messages are no longer stored in memory
  - Previous to 5.1, message references were stored in memory
- Messages are paged in from storage when space is available in memory

Master/Slave Broker Configurations

Three Types of Master/Slave

- Pure master/slave
- Shared filesystem master/slave
- JDBC master/slave

Pure Master/Slave

- Shared nothing, fully replicated topology
  - Does not depend on shared filesystem or database
- A Slave broker consumes all message states from the Master broker (messages, acks, tx states)
- Slave does not start any networking or transport connectors

Pure Master/Slave

- Master broker will only respond to client when a message exchange has been successfully passed to the slave broker

Pure Master/Slave

- If the master fails, the slave optionally has two modes of operation:
  - Start up all it’s network and transport connectors
    - All clients connected to failed Master resume on Slave
  - Close down completely
    - Slave is simply used to duplicate state from Master

Shared Filesystem Master/Slave

- Utilizes a directory on a shared filesystem
- No restriction on number of brokers
- Simple configuration (point to the data dir)
- One master selected at random

JDBC Master/Slave

- Recommended when using a shared database
- No restriction on the number of brokers
- Simple configuration
- Clustered database negates single point of failure
- One master selected at random

Client Connectivity With Master/Slave

:: Again, clients should use failover transport:

```
failover:(tcp://broker1:61616,tcp://broker2:61616, \
tcp://broker3:61616)?initialReconnectDelay=100
```

Tips for HA and Fault Tolerance

- RAIDed disks
- A Storage Area Network
- Clustered relational databases
- Clustered JDBC via C-JDBC
  - http://c-jdbc.objectweb.org/

Security

Broker Security

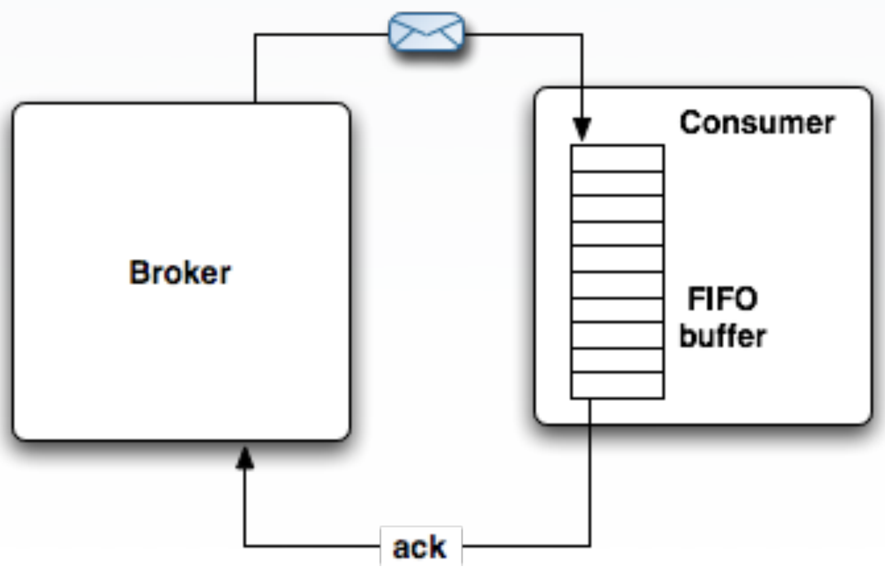
- Authentication
  - I.e., are you allowed to connect to ActiveMQ?
  - File based implementation
  - JAAS based implementation
- Authorization
  - I.e., do you have permission to use that ActiveMQ resource?
  - Destination level
  - Message level via custom plugin

Consumer Options

- Message prefetch
- Consumer dispatch async
- Exclusive consumer
- Consumer priority
- Message groups
- Redeliery policies
- Retroactive consumer
- Selectors
- **Some slow consumer strategies**

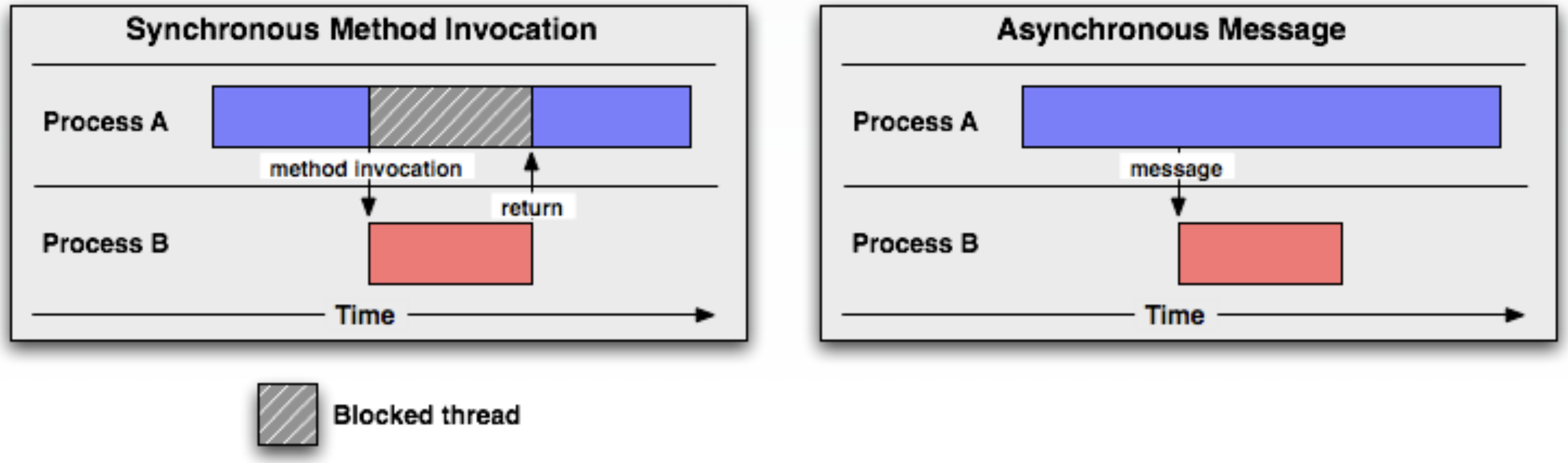
Message Prefetch

- Used for slow consumer situations
  - Consumer is flooded by messages from the broker
- FIFO buffer on the consumer side



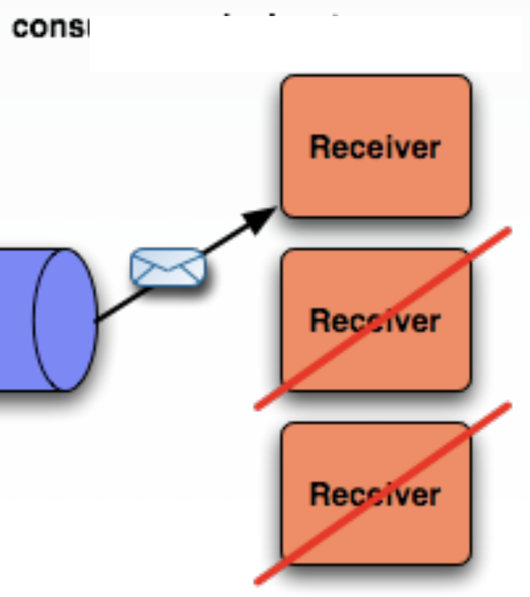
Async Dispatch

- Asynchronous message delivery to consumers
  - Default is true
- Useful for slow consumers
  - Incurs a bit of overhead



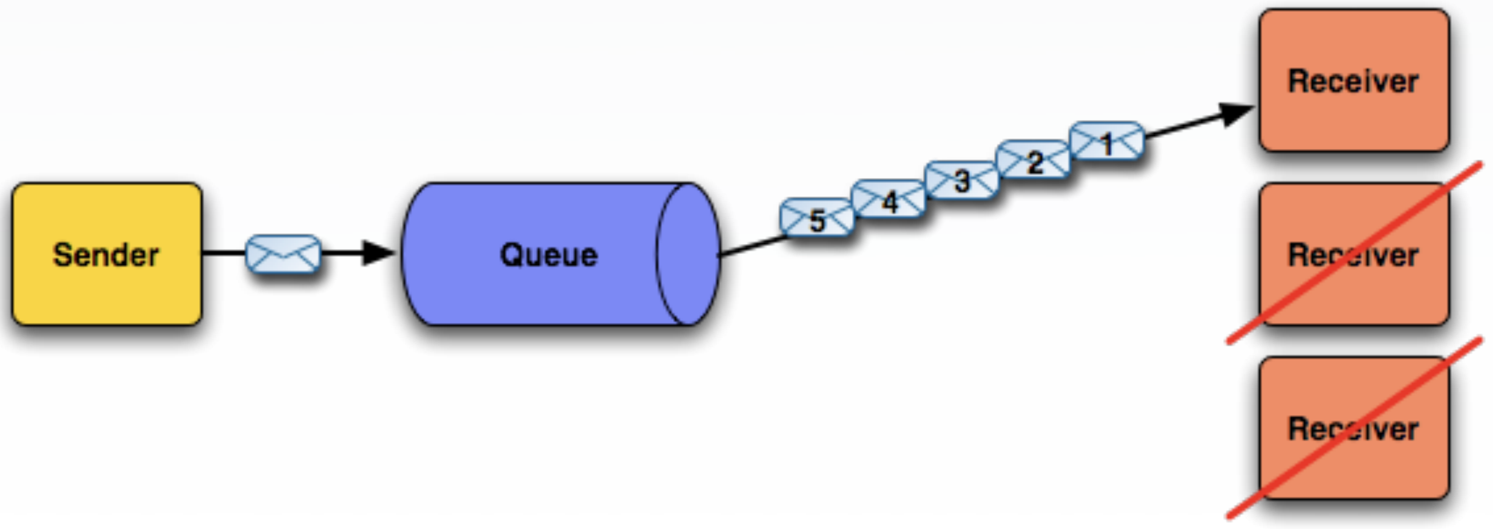
Exclusive Consumers

- Anytime more than one consumer is consuming from a queue, message order is lost
- Allows a single consumer to consume all messages on a queue to maintain message ordering



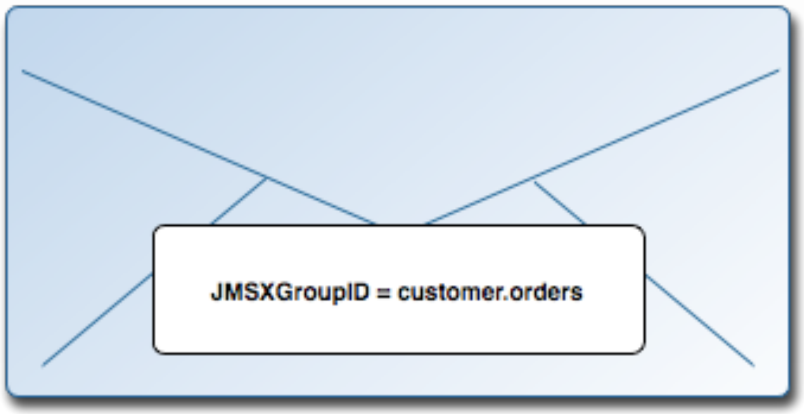
Consumer Priority

- Just like it sounds
  - Gives a consumer priority for message delivery
  - Allows for the weighting of consumers to optimize network traversal for message delivery



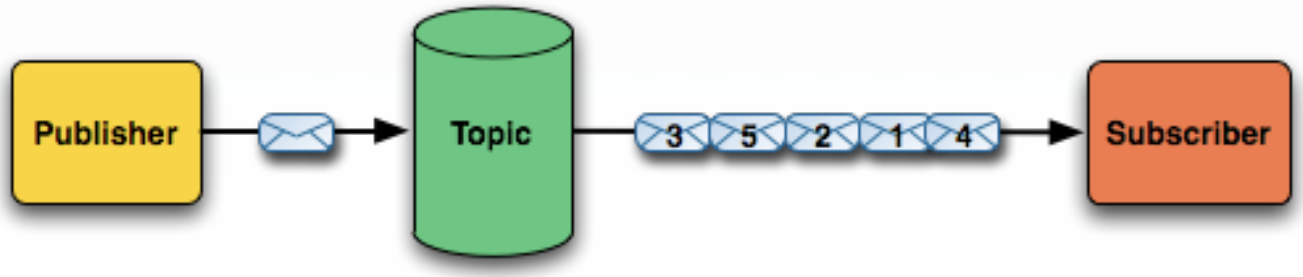
Message Groups

- Uses the JMSXGroupID property to define which message group a message belongs
  - Guarantees ordered processing of related messages across a single destination
  - Load balancing of message processing across multiple consumers
  - HA/failover if consumer goes down



Retroactive Consumer

- Message replay at start of a subscription
  - At the start of every subscription, send any old messages that the consumer may have missed
  - Configurable via policies



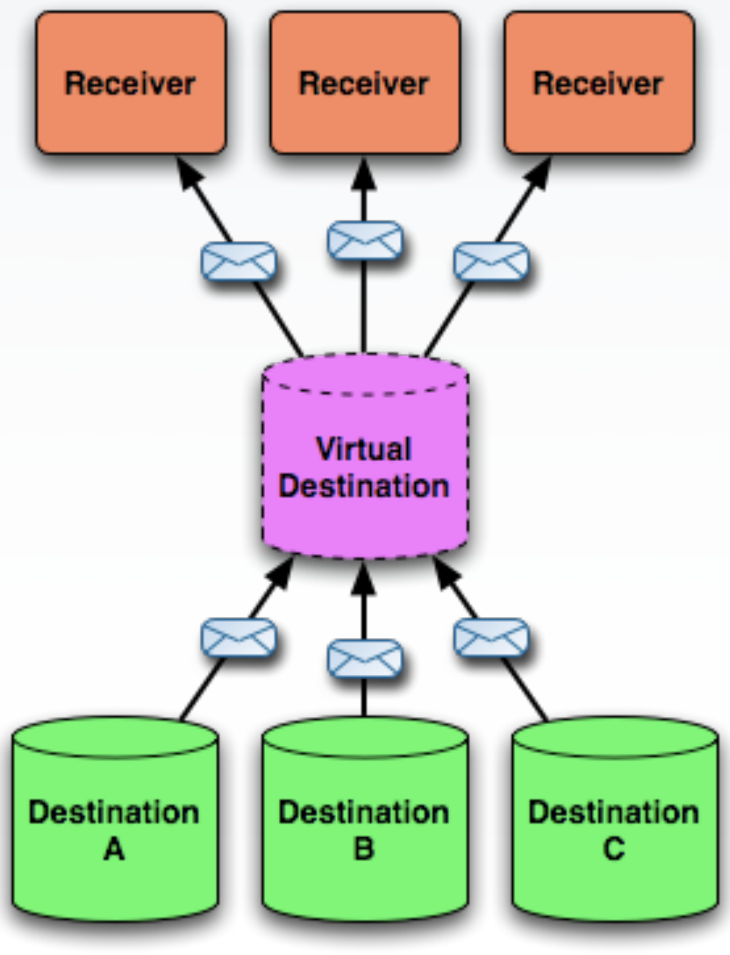
Wildcards on Destinations

EXAMPLE

```
...
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="Price.Stock.>"
          memoryLimit="128mb">
    </policyEntries>
  </policyMap>
</destinationPolicy>
...
```

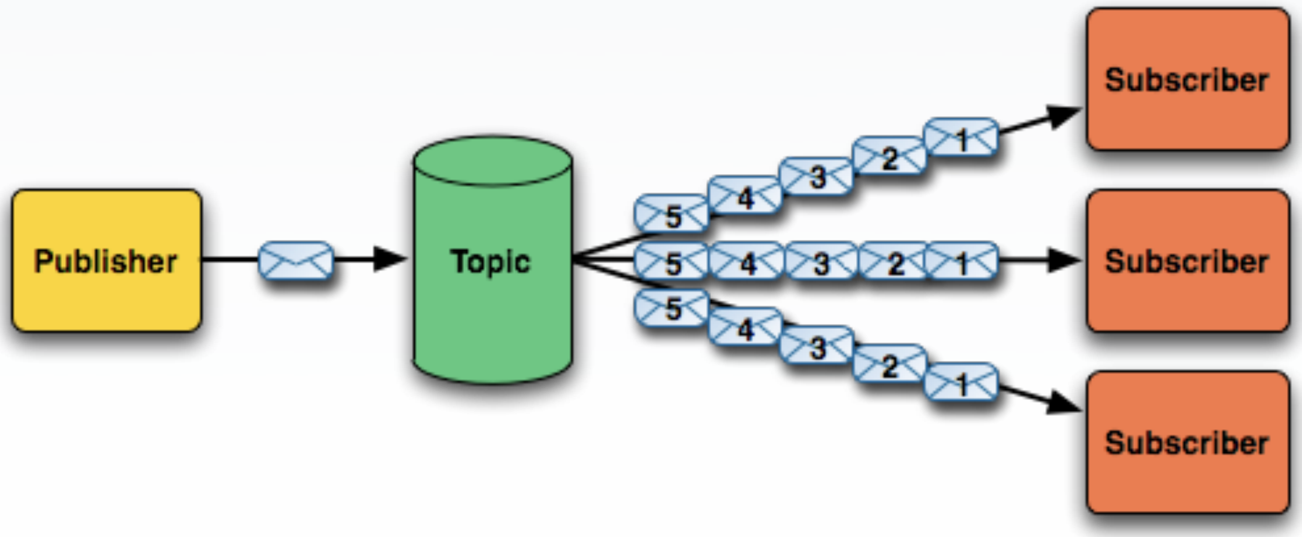
- Price.>
- Price.Stock.>
- Price.Stock.NASDAQ.*
- Price.Stock.*.IBM

Virtual Destinations

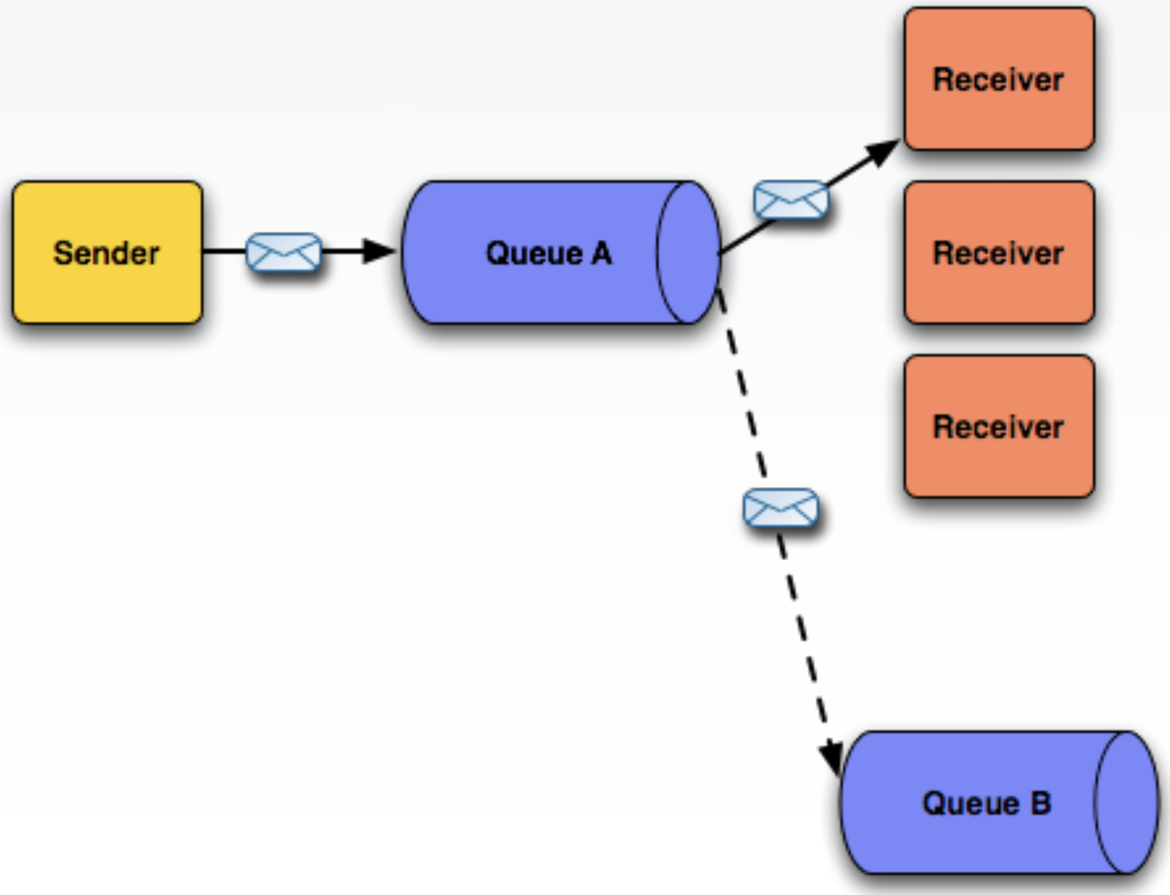


Total Ordering

- A guaranteed order of messages for each consumer



Mirrored Queues



Message Selectors

- Used to attach a filter to a subscription
- Defined using a subset SQL 92 syntax
- JMS selectors
  - Filters only message properties
    - JMSType = ‘stock’ and trader = ‘bob’ and price < ‘105’
- XPath selectors
  - Filters message bodies that contain XML
    - ‘/message/cheese/text() = 'swiss''

Retroactive Consumer

- Used to go back in time
  - In terms of messages
- At the start of a subscription, send old messages the consumer may have missed
- Configurable via timed or fixed size recovery

Slow Consumer Strategies

- Various configurable strategies for handling slow consumers
- Slow consumer situations are ***very*** common
- Caused by:
  - Slow network connections
  - Unreliable network connections
  - Busy network situations
  - Busy JVM situations
  - Half disconnects with sockets

Use Message Limit Strategies

- PendingMessageLimitStrategy
  - Calculates the max number of pending messages to be held in memory for a consumer above its prefetch size
- ConstantPendingMessageLimitStrategy
  - A constant limit for all consumers
- PrefetchRatePendingMessageLimitStrategy
  - Calculates the max number of pending messages using a multiplier of the consumers prefetch size

Use Prefetch and an Eviction Policy

- Use the prefetch policy
  - The prefetch policy has a property named maximumPendingMessageLimit that can be used on a per connection or per consumer basis
- Use a message eviction policy
  - OldestMessageEvictionStrategy
    - Evict the oldest messages first
  - OldestMessageWithLowestPriorityEvictionStrategy
    - Evict the oldest messages with the lowest priority first

Use Destination Policies

- Configured on the destination policies in the ActiveMQ XML configuration file
- Combined with wildcards, this is very powerful

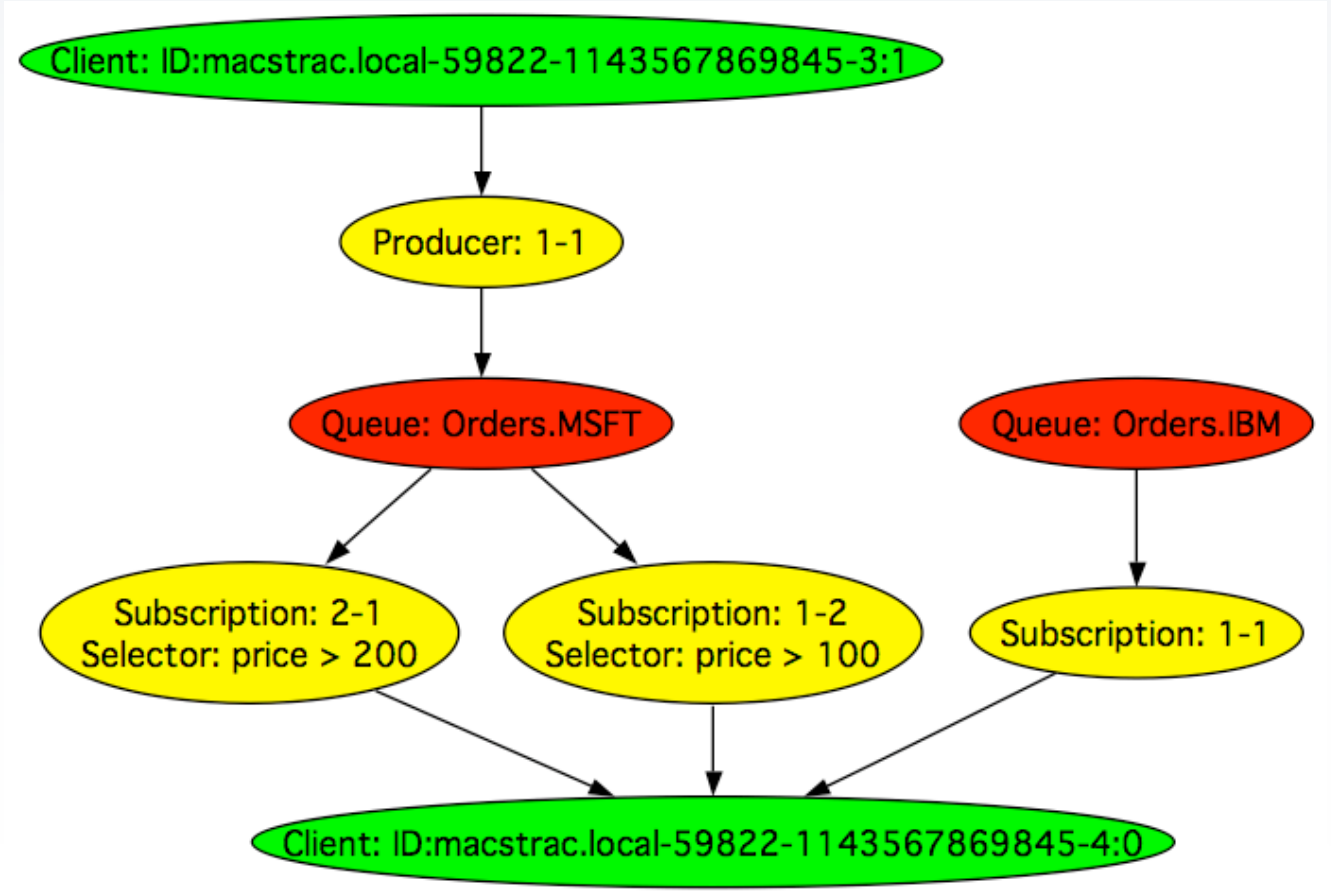
Additional Tips

- Consider configuring different message cursors
- The status of slow consumers can be monitored via JMX properties
  - discarded - The count of how many messages have been discarded during the lifetime of the subscription due to it being a slow consumer
  - matched - The current number of messages matched and to be dispatched to the subscription as soon as some capacity is available in the prefetch buffer. So a non-zero value implies that the prefetch buffer is full for this subscription

Monitoring

- JMX
- ActiveMQ web console
- Additional consumers
  - Camel routes
- SpringSource AMS
  - Based on Hyperic
- IONA FuseHQ
  - Based on Hyperic

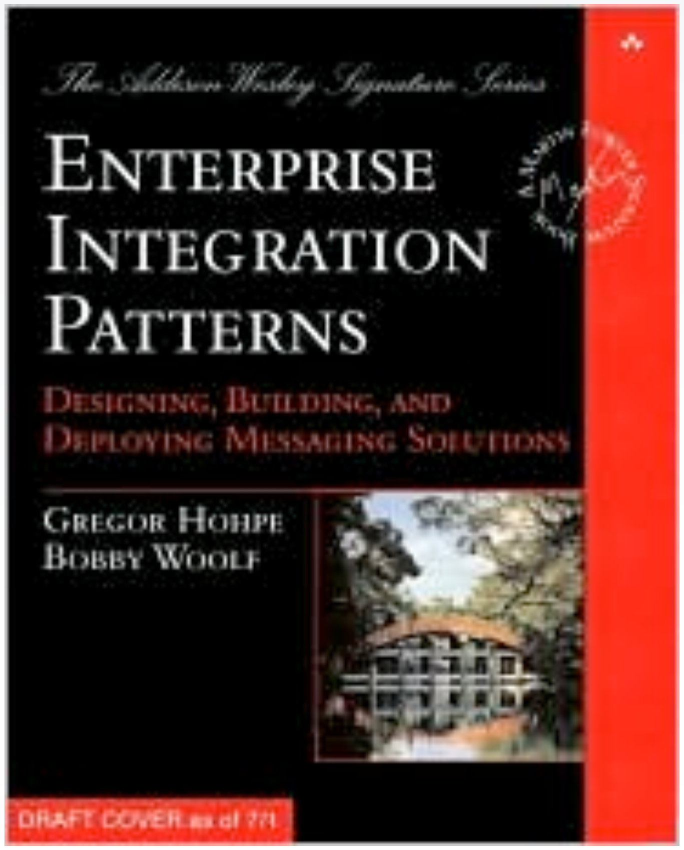
Visualization



ActiveMQ Broker: ID:macstrac.local-59822-1143567869845-0:0

What is Apache Camel?

What is EIP?


The Addison-Wesley Signature Series
ENTERPRISE INTEGRATION PATTERNS
DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS
GREGOR HOHPE
BOBBY WOOLF
DRAFT COVER as of 7/1

Fluent Java API

EXAMPLE

```
RouteBuilder MyRoute = new RouteBuilder() {
  public void configure() {
    from("activemq:TEST.QUEUE").
      to("file:///Users/bsnyder/camelinbox/text.txt").
      to("log:MyLog");
  }
};
```

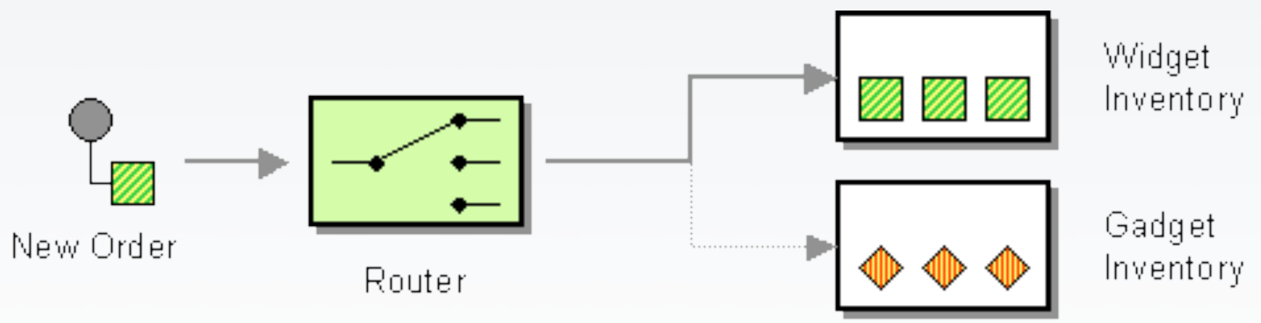
XML Config

EXAMPLE

```
<camelContext id="camel"
  xmlns="http://activemq.apache.org/camel/schema/spring">
  <package>com.mycompany</package>
  <route>
    <from uri="activemq:example.A" />
    <to uri="file:///Users/bsnyder/camelinbox/text.txt" />
    <to uri="log:MyLog?showProperties=true" />
  </route>
</camelContext>
```

EIP Pattern: Content Based Router

EXAMPLE



```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {

from("seda:a").choice().when(header("foo").isEqualTo("bar
")).to("seda:b")
            .when(header("foo").isEqualTo("cheese")).to("
seda:c").otherwise().to("seda:d");
    }
};
```

EIP Pattern: Content Based Router

EXAMPLE

```
<camelContext id="buildSimpleRouteWithChoice"
    xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="seda:a"/>
    <choice>
      <when>
        <predicate>
          <header name="foo"/>
          <isEqualTo value="bar"/>
        </predicate>
        <to uri="seda:b"/>
      </when>
      <when>
        <predicate>
          <header name="foo"/>
          <isEqualTo value="cheese"/>
        </predicate>
        <to uri="seda:c"/>
      </when>
      <otherwise><to uri="seda:d"/></otherwise>
    </choice>
  </route>
</camelContext>
```

Do You Have Information Overload Yet? ;-)

Thank You For Attending!

Questions?