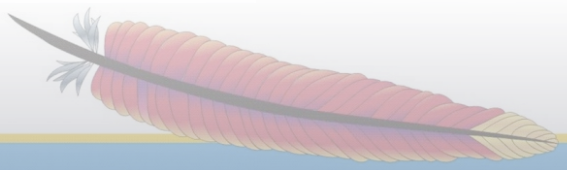
Real World Messaging With Apache ActiveMQ

Bruce Snyder
bsnyder@apache.org
7 Nov 2008
New Orleans, Louisiana

Do You Use JMS?

Agenda

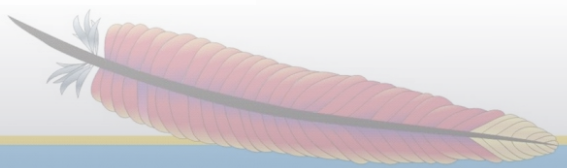
- Common questions
- ActiveMQ features



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

What kind of hardware is needed for a throughput of *X* number of messages per second?



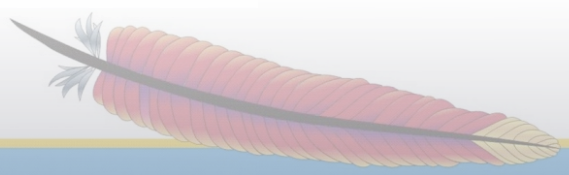
What Are Your Performance Objectives?

What Trade-Offs Can You Accept?

		1						
		2		3				4
			5			6		7
5			1	4				
	7						2	
				7	8			9
8		7			9			
4				6		3		
						5		

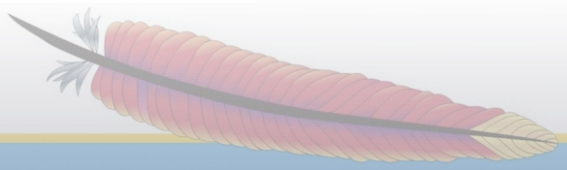
Common Trade-Offs

- Messaging Domain
- Durability v. Persistence
- Message acks v. transactions
- Message consumption types



Messaging Domains

- Do all consumers need a copy of the messages?
- What if consumers are disconnected?
- Can consumers deal with missing messages?
- Do you need request/reply messaging?

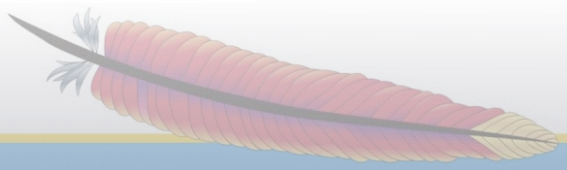


Durability v. Persistence

- What quality of service do you need?
  - **Durability** - should messages be held while consumer is offline?
  - **Persistence** - used to preserve messages in the event of a JMS provider failure

Message Acks v. Transactions

- When messages are received, the consumer must acknowledge the message
  - There is no unacknowledge!
- Transactions allow for:
  - Rollback for error handling
  - Batching for speed



Synchronous Message Consumption

```
public class MyConsumer {
...

public void doCreateConsumer() {

  Destination destination =
    consumer.getSession().createQueue("JOBS." + job);

  MessageConsumer messageConsumer =
     consumer.getSession().createConsumer(destination);

  while ((message = consumer.receive(timeout)) != null) {
    processMessage(message);
  }
}

...
}
```

Asynchronous Message Consumption

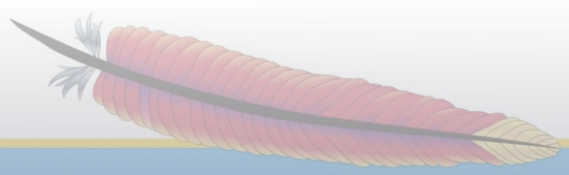
```
public class MyConsumer {
...

public void doCreateConsumer() {

  Destination destination =
    consumer.getSession().createQueue("JOBS." + job);

  MessageConsumer messageConsumer =
     consumer.getSession().createConsumer(destination);

  messageConsumer.setMessageListener(new MyMessageListener(job));
}

...
}
```

How can the broker be clustered to improve scalability or guarantee availability?

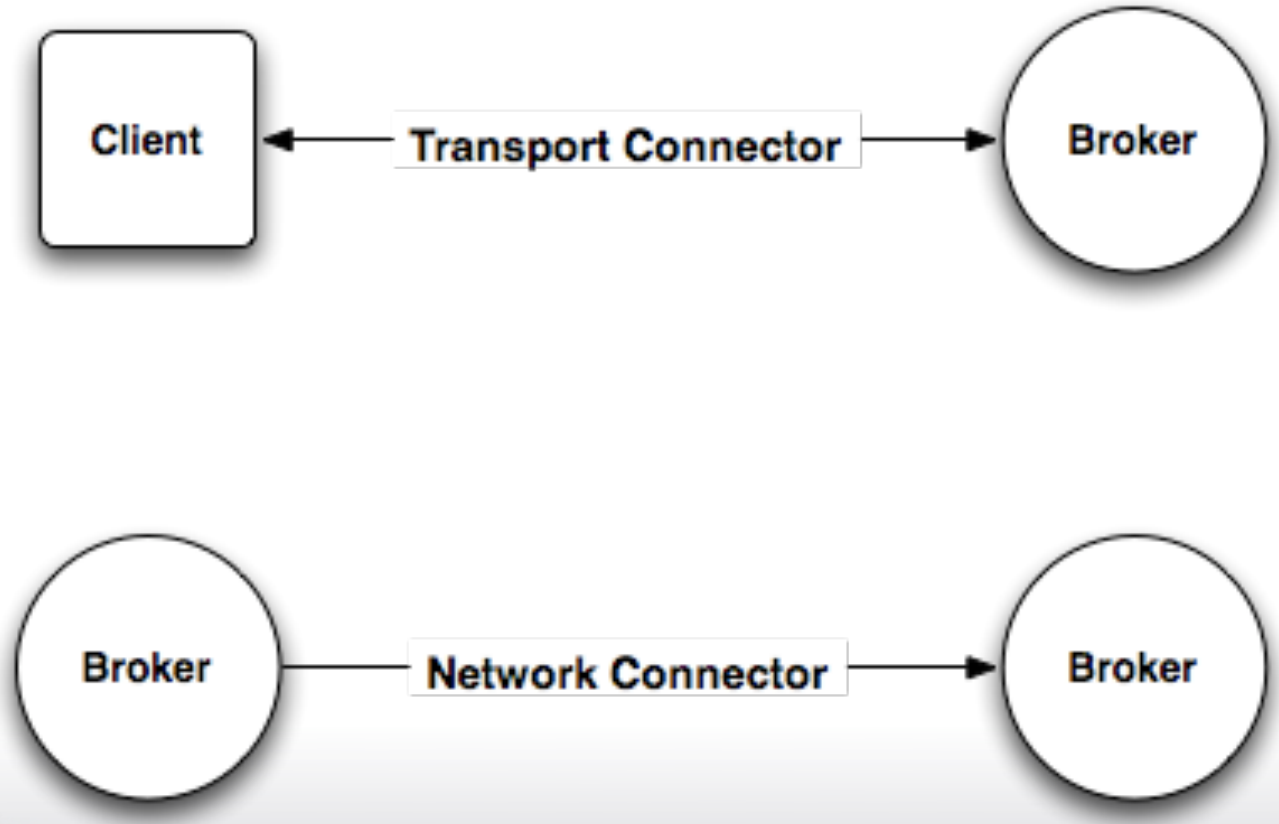
- Let's talk about two features
  - Network of brokers
  - Master/slave

Broker Options

- Connectors
- Persistence
- Master/slave
- Security

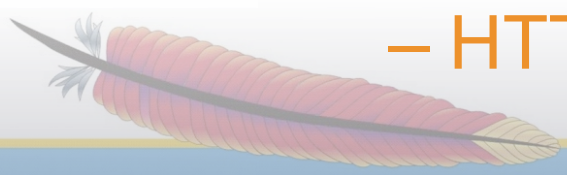


Two Types of Transports



Transport Connectors

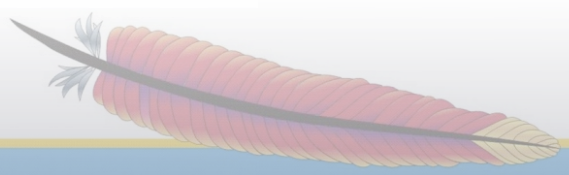
- Client-to-broker connections
  - Similar to JDBC connections to a database
- Various protocols are supported:
  - VM
  - TCP
  - NIO
  - UDP
  - SSL
  - HTTP/S



Clients Should Be Configured For Failover

- Use the failover protocol:

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

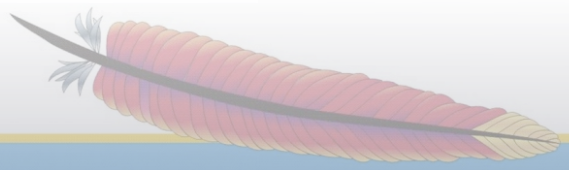


Network of Brokers

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

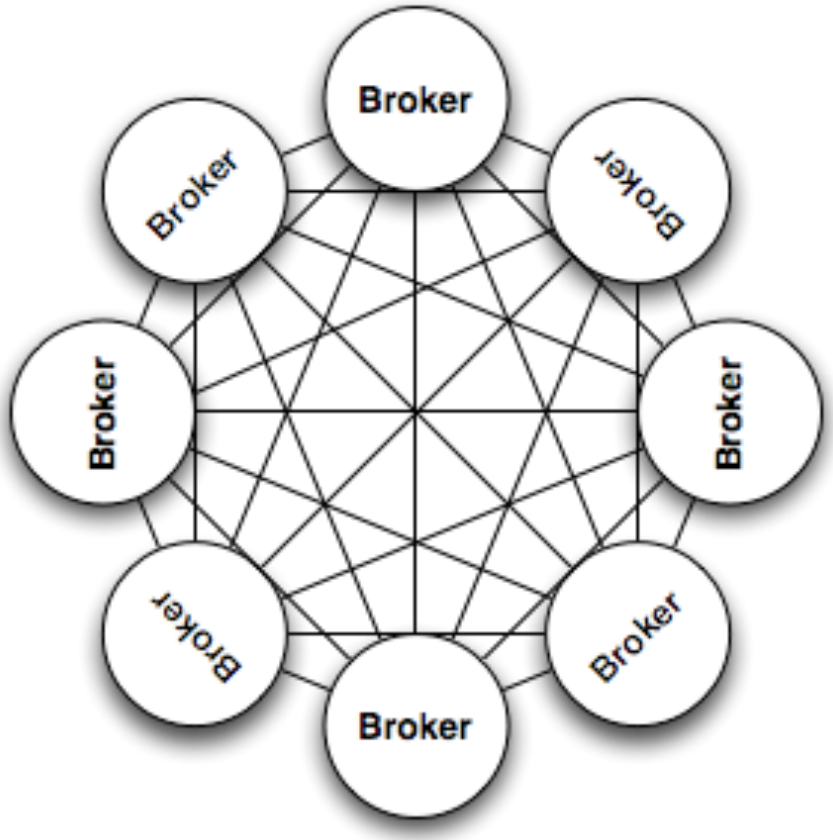
Networks of Brokers

- 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

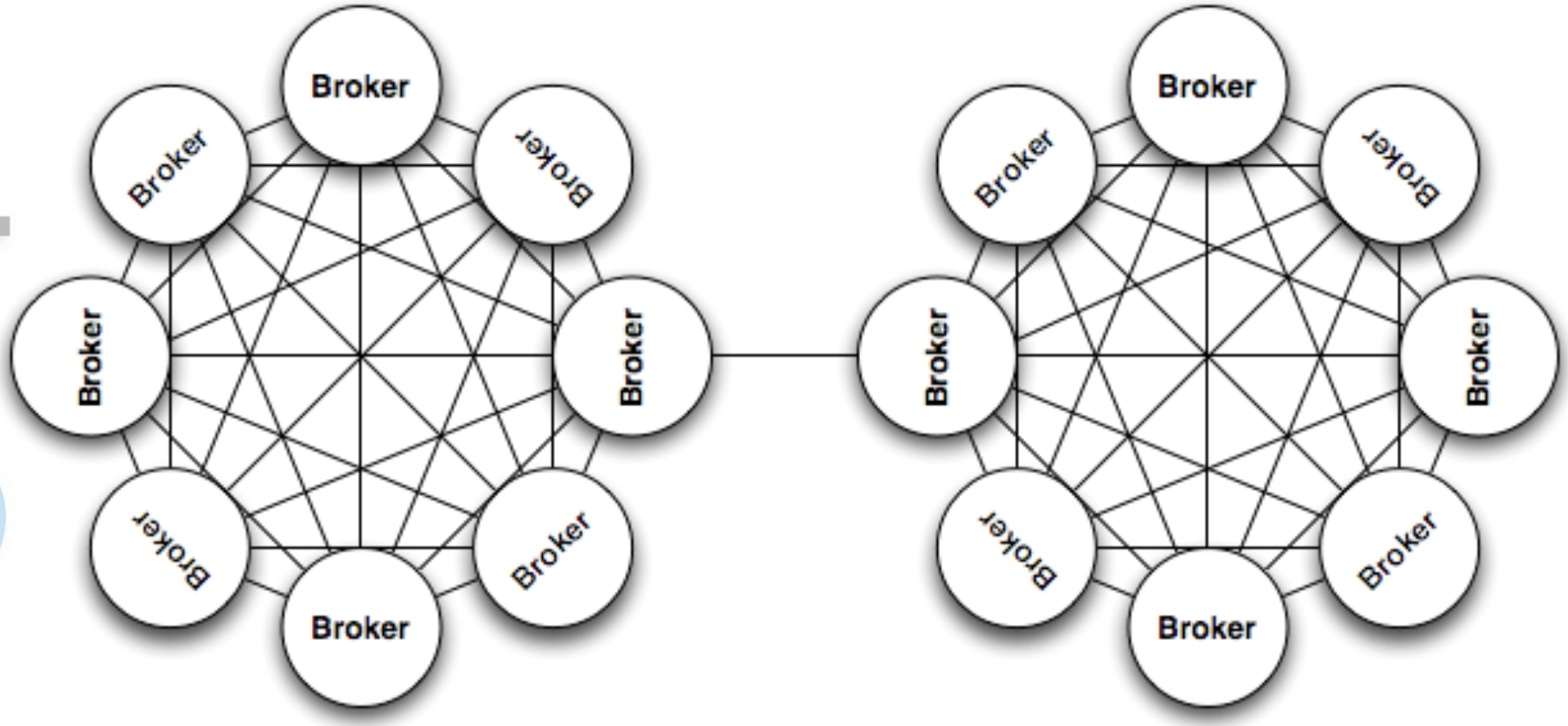


Topology Example

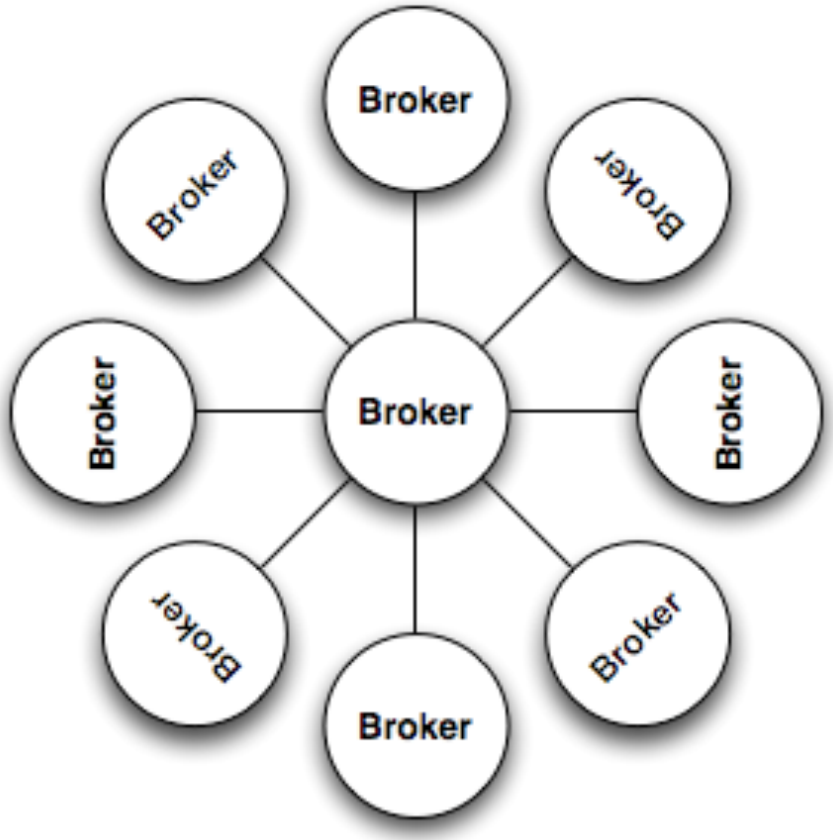
Topology Example



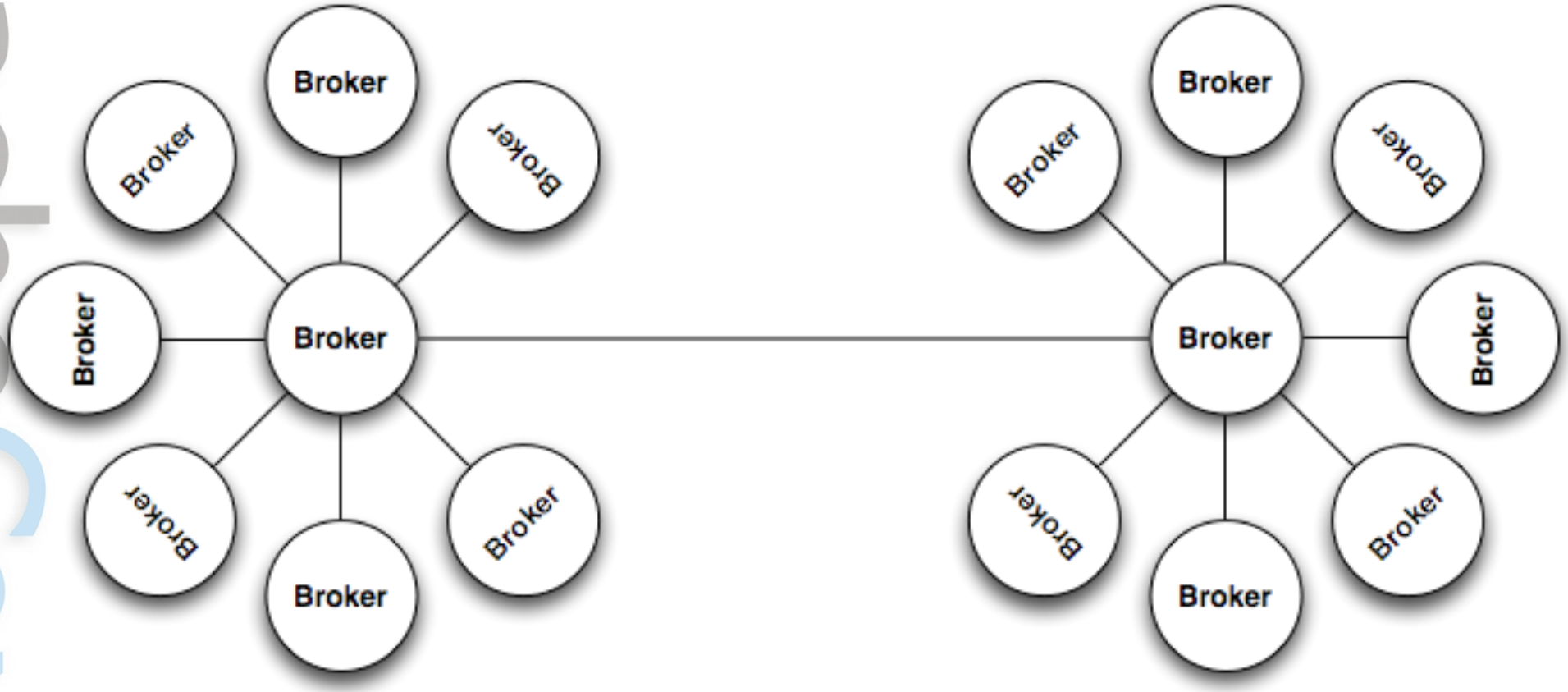
Topology Example



Topology Example



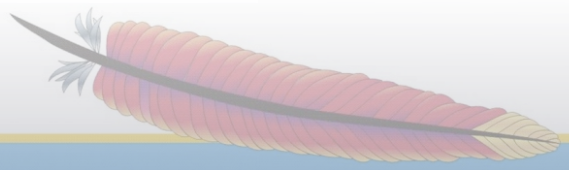
Topology Example



Master/Slave Broker Configurations

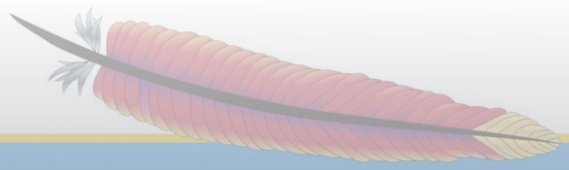
Three Types of Master/Slave

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



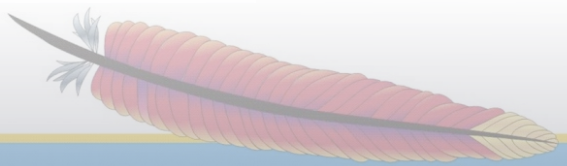
Pure Master/Slave

- 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

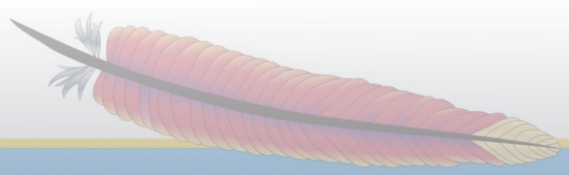


Pure Master/Slave

- 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

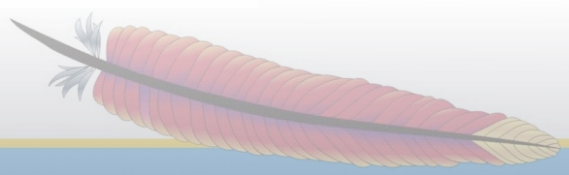
Shared Filesystem Master/Slave

- 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

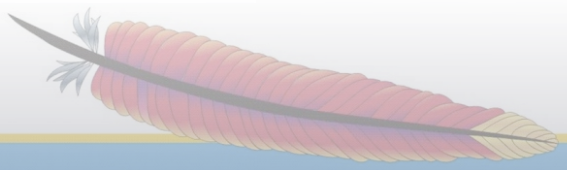
- Recommended when using a shared database
- No restriction on the number of brokers
- Simple configuration
- Clustered database negates single point of failure
- One master selected at random



Client Connectivity With Master/Slave

- Clients should use failover transport:

```
failover://(tcp://masterhost:61616, \
  tcp://slavehost:61616)?randomize=false
```

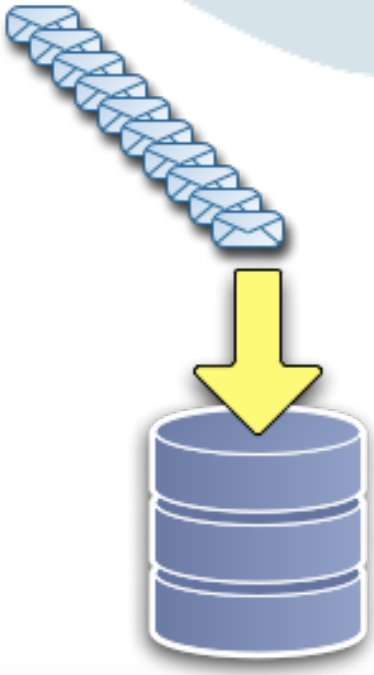
How can the broker be clustered to improve scalability or guarantee availability?

- Create a network of brokers
- Use master/slave for broker failover

High Availability and Fault Tolerance

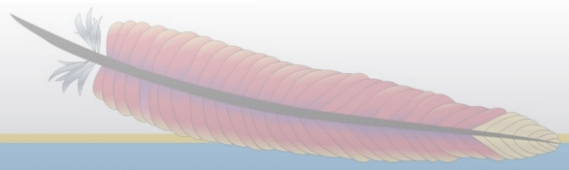
- RAIDed disks
- A Storage Area Network
- Clustered relational databases
- Clustered JDBC via C-JDBC
  - http://c-jdbc.objectweb.org/

What's the real difference between message persistence strategies?



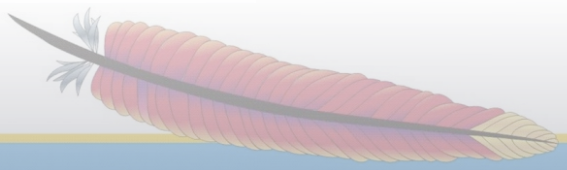
AMQ Message Store

- 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



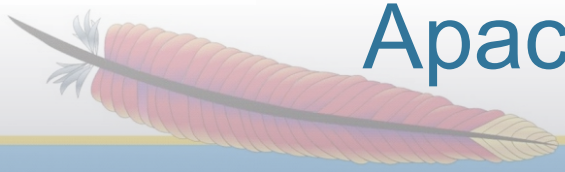
Non-Journaled JDBC

- 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

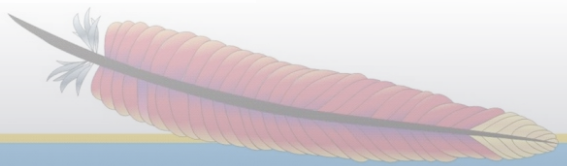
- Messages are no longer stored in memory
  - Previous to 5.1, message references were stored in memory
- Messages are paged in from storage when space is available in memory

What security options are available?

- Authentication
  - I.e., are you allowed to connect to ActiveMQ?
- Authorization
  - I.e., do you have permission to use that ActiveMQ resource?

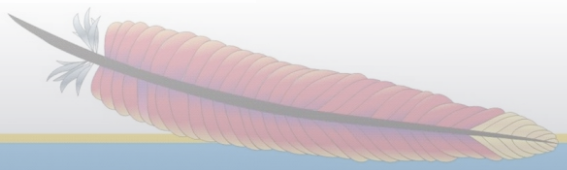
Authentication

- File based
- JAAS based



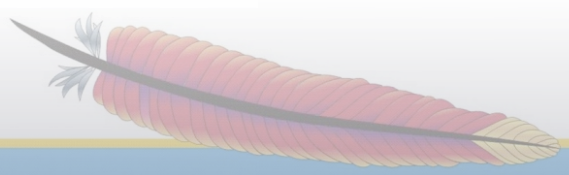
Authorization

- Destination level
- Message level via custom plugin



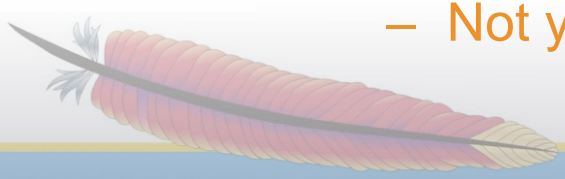
What's the best way to connect non-Java clients?

- Let’s talk about wire formats



Wire Formats

- OpenWire
  - The default in ActiveMQ; a binary protocol
- STOMP
  - Simple Text Oriented Messaging Protocol; a text based protocol
- XMPP
  - The Jabber XML protocol
- REST
  - HTTP POST and GET
- AMQP
  - Not yet fully supported

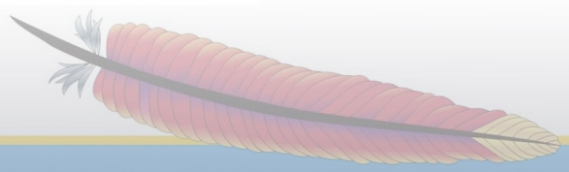


OpenWire

- A binary wire level protocol for marshaling objects to/from byte arrays
- Designed for performance
  - Sacrificed some ease of implementation
- Clients for C++, Java and .NET
  - The default is Java

STOMP

- A text-based wire level protocol
- Designed to be very easy to understand
  - Encourages implementation in additional languages
- Can be easily demonstrated via telnet
- Clients for C, Javascript, Perl, PHP, Python, Ruby and more



What's the best approach if a producer or consumer may be down for some time?

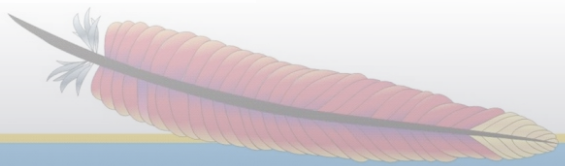
Use the failover Protocol

- Utilize the failover options
  - initialReconnectDelay - wait before reconnect
  - maxReconnectDelay - max time between reconnects
  - useExponentialBackOff - exponentially grow time between reconnects
  - backOffMultiplier - multiplier for backoff
  - maxReconnectAttempts - max number of reconnects
  - randomize - randomly choose from list of brokers
  - backup - start and hold a hot standby connection

Use Producer Flow Control

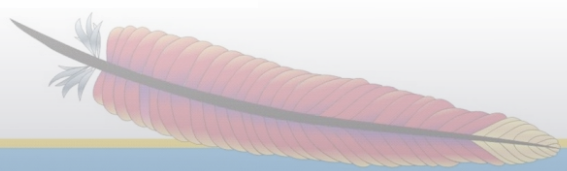
- Prevents slow consumers from being flooded

```
...
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="FOO.>"
            producerFlowControl="false"
            memoryLimit="128mb">
    </policyEntries>
  </policyMap>
</destinationPolicy>
...
```



Use Message Persistence

- Quality of service to preserve messages in the event of broker failure

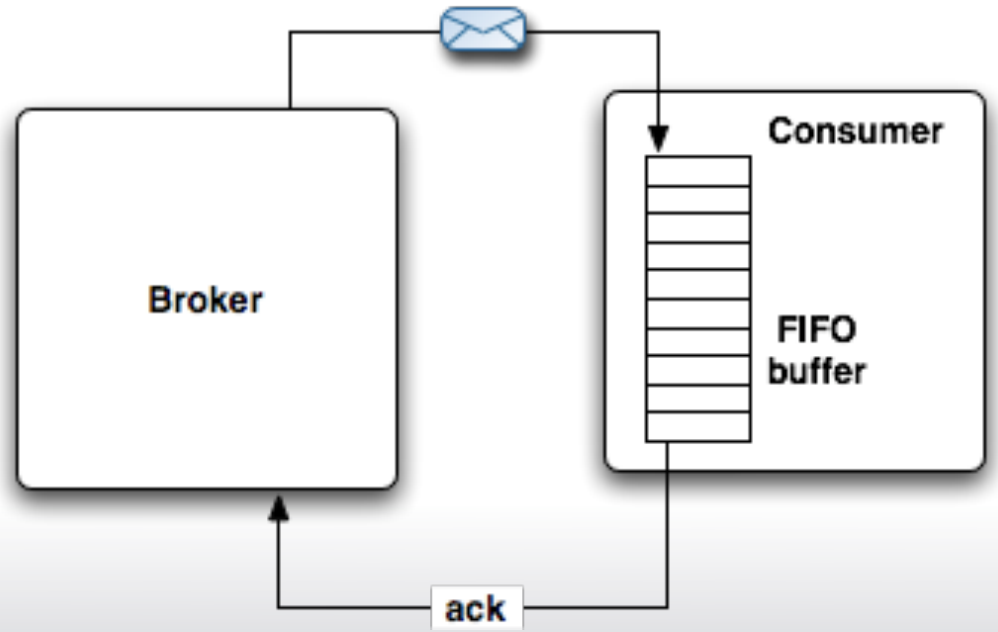


Consumer Options

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

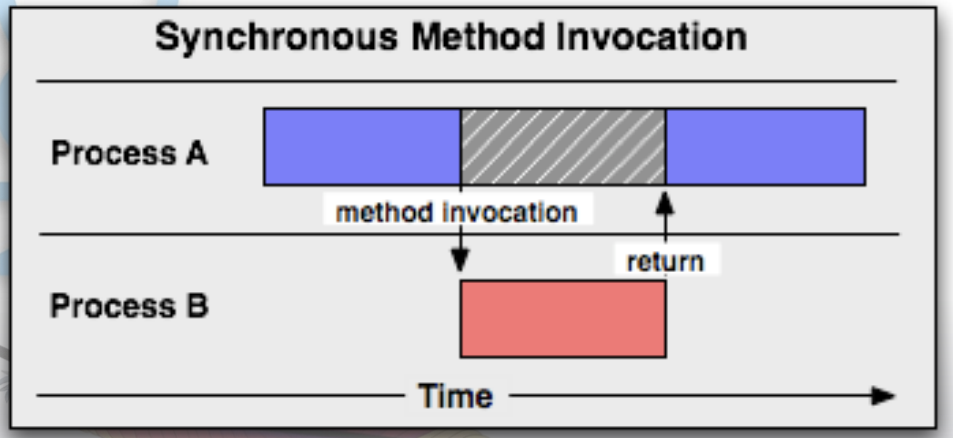
Message Prefetch

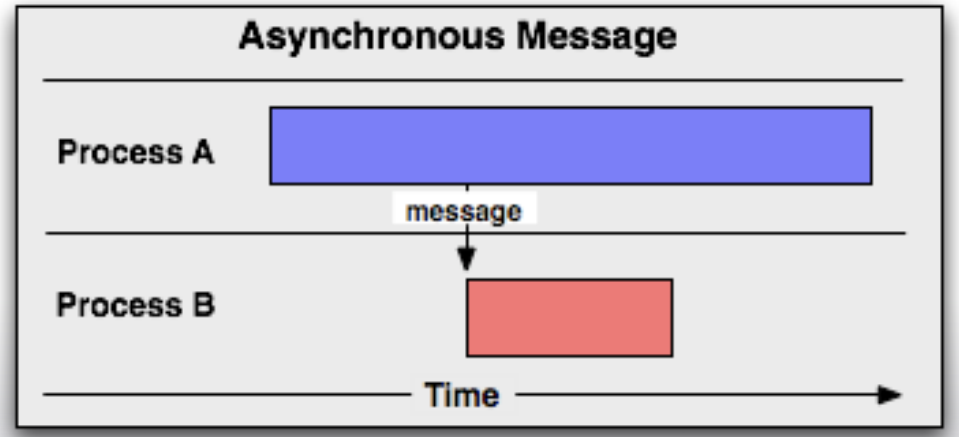
- Used for slow consumer situations
  - Consumer is flooded by messages from the broker
- FIFO buffer on the consumer side



Async Dispatch

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

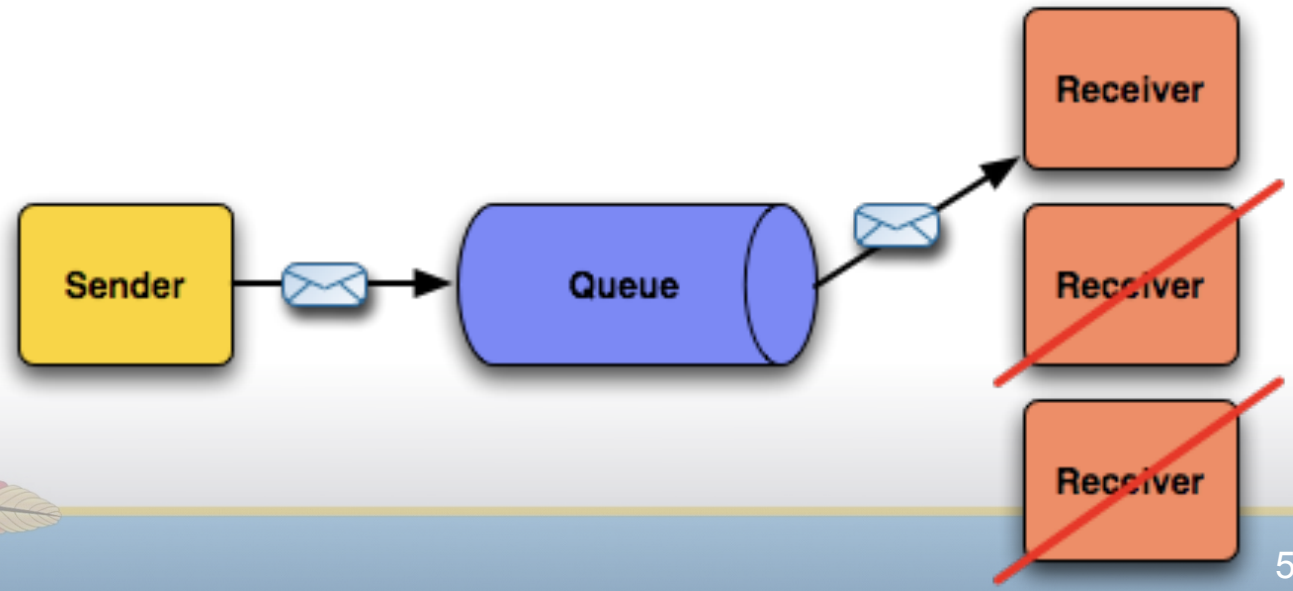




Blocked thread

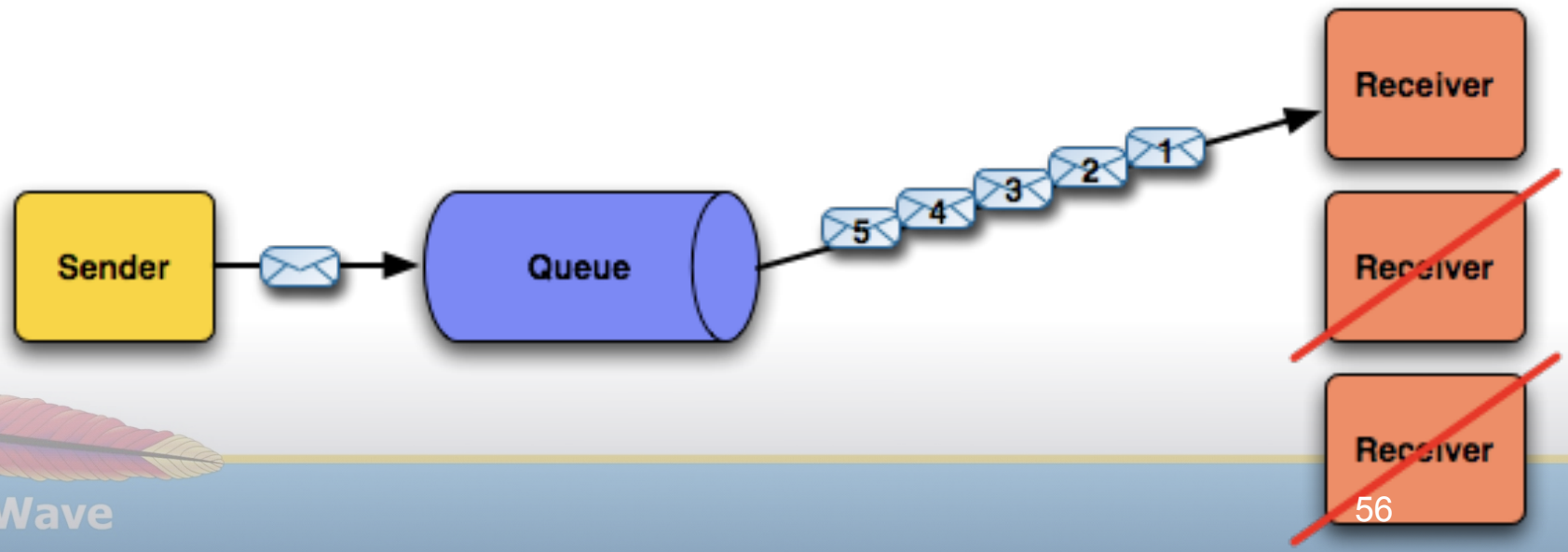
Exclusive Consumers

- 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

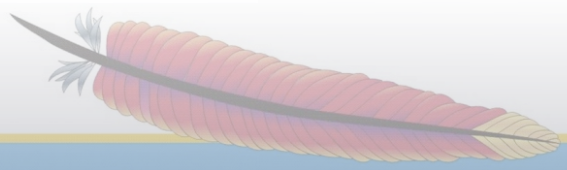


Message Groups

- 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

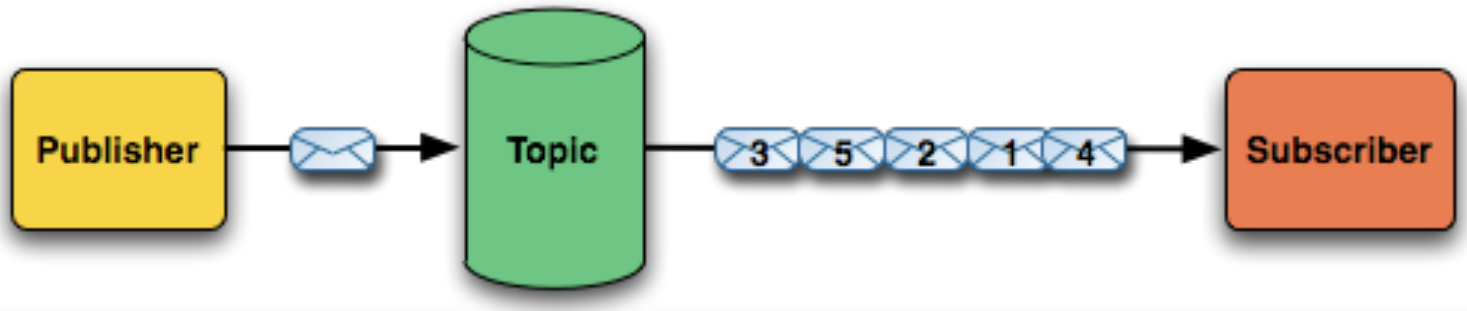
Redelivery Policy

- Messages are redelivered to a client when:
  - A transacted session is rolled back
  - A transacted session is closed before commit
  - A session is using CLIENT_ACKNOWLEDGE and Session.recover() is explicitly called



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

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

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

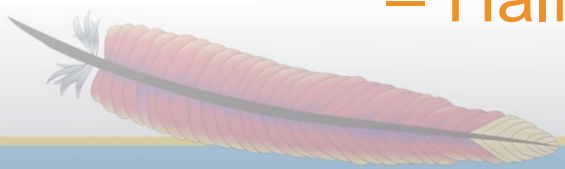
Selectors

- 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''

Slow Consumers Strategies

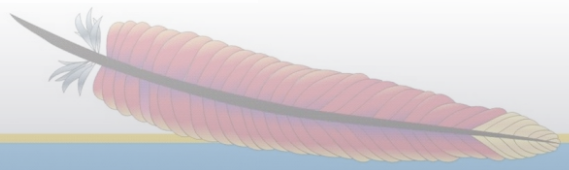
Slow Consumer Strategies

- 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



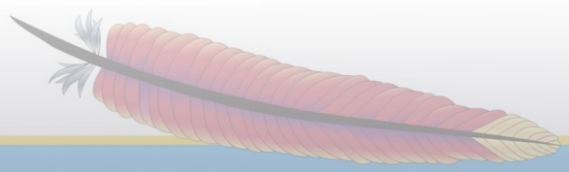
Use Message Limit Strategies

- 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



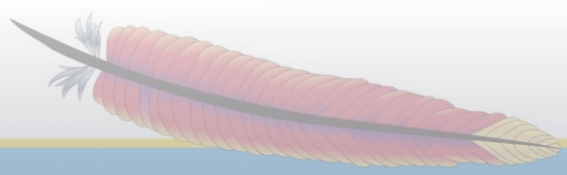
Use Prefetch and an Eviction Policy

- 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



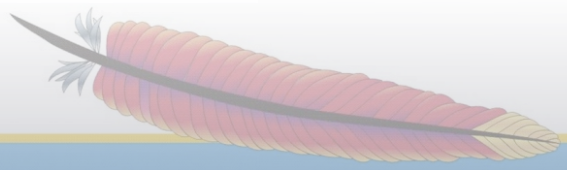
Use Destination Policies

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



Additional Tips

- Consider configuring 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

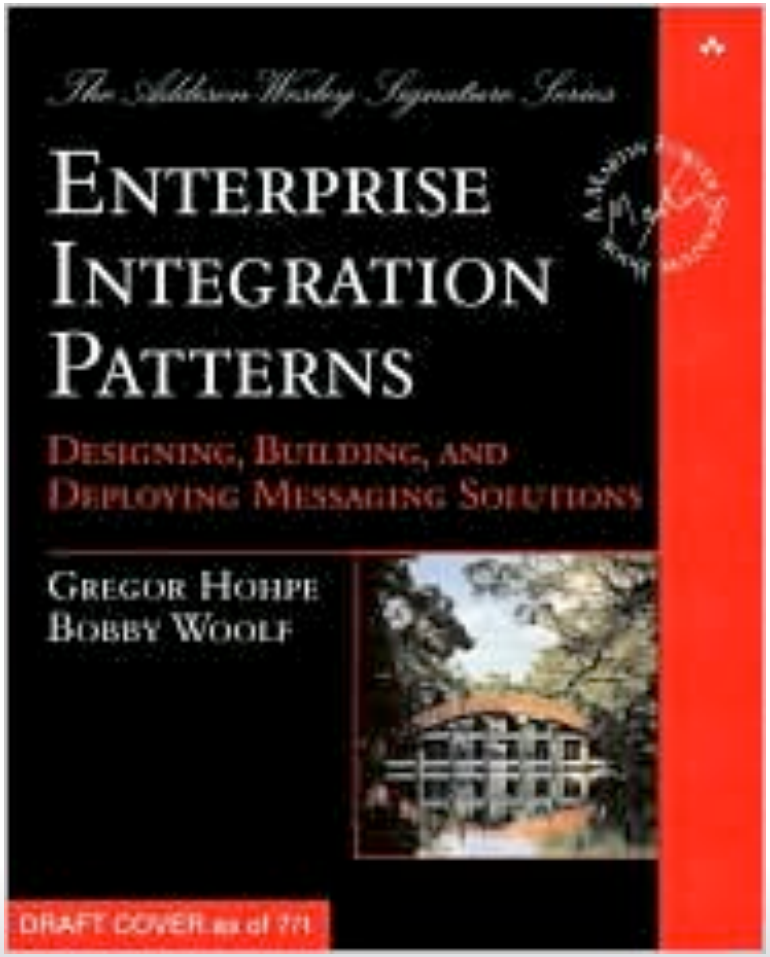


How to monitor individual messages, queue depths, or other broker statistics?

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

What is Apache Camel?

http://activemq.apache.org/camel/



Camel Components

ActiveMQ	File	JBI	MINA	RMI	TCP
ActiveMQ Journal	FIX	JCR	Mock	RNC	Test
AMQP	Flatpack	JDBC	MSMQ	RNG	Timer
Atom	FTP	Jetty	MSV	SEDA	UDP
Bean	Hibernate	JMS	Multicast	SFTP	Validation
CXF	HTTP	JPA	POJO	SMTP	Velocity
DataSet	iBATIS	JT/400	POP	Spring Integration	VM
Direct	IMAP	List	Quartz	SQL	XMPP
Esper	IRC	Log	Queue	Stream	XQuery
Event	JavaSpace	Mail	Ref	String Template	XSLT

Fluent Java API

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

XML Config

```
<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>
```

Information Overload Yet?

Thank You For Attending!

Questions?