




## JBoss Messaging Advanced Session


Tim Fox  
JBoss Messaging Technical Lead

Berlin, November 2006



## Agenda

- Summary
- Team
- Features
- JBoss Messaging architecture
  - JMS Façade
  - Messaging core building blocks
  - Very large queue support
  - Post offices
  - Distributed destinations
  - Failover and High Availability (HA)
  - In memory message replication
- Roadmap
- Questions?




## JBoss Messaging - Summary

- **What is JBoss Messaging?**
  - JBoss Messaging is JBoss' next generation asynchronous reliable messaging system.
  - Replacement for JBoss MQ
  - Part of the JEMS suite of products



## Team and community


- **Community:**
  - We have an active user community – growing all the time
  - Always looking for new contributors – please lend a hand!
  - Opportunity to be part of one of the most widely deployed messaging products in existence.
- **Permanent team:**
  - Ovidiu Feodorov – Project Lead
  - Tim Fox – Technical Lead
  - Juha Lindfors
  - Ron Sigal (part time borrowed from the remoting team)
  - Clebert Suconic



## JBoss Messaging Features

**JBoss Messaging 1.0 features**

- Full JMS 1.1 compliance
- Persistence support for Oracle, DB2, Sybase, PostgreSQL, MySQL, SQL Server
- JTA integration – JBoss Messaging can do work as part of a larger global transaction
- JMX management interface
- JAAS based security – plug in to your existing security infrastructure
- SSL transport



## JBoss Messaging Features (2)

**JBoss Messaging 1.2 features**

- JBoss Transactions integration (also available in 1.0.2)
- Fully distributed queues and topics
- Shared durable subscriptions
- High availability with seamless failover
- Intelligent message redistribution
- HTTP transport
- "Unreliable link" scenario
- Non single point of failure (SPOF) or single point of bottleneck (SPOB)
- In memory message replication
- NIO transport
- Very large message support

## JBoss Messaging features (3)

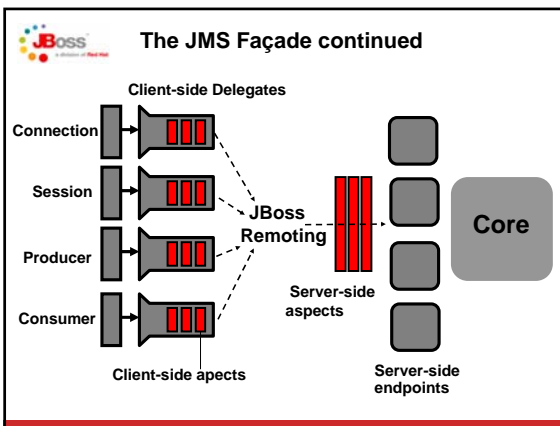
**JBoss Messaging 2.0 and later features**

- AMQP support
- WAN support

## Architectural Overview

A Messaging server instance consists of two layers

- **The JMS Façade**
  - Gives JMS "personality" to the Messaging Core
  - Implements the JMS API
  - Its inner workings are built in top of JBoss AOP
  - Stack of aspects plus a set of services
- **The Messaging Core**
  - A generic, reliable asynchronous messaging transport system
  - Does one thing and does it well: guarantees the *reliability* of a message submission, for those messages that have been configured to be reliable
  - Supports generic messages (not necessarily JMS)
  - Has a proprietary API



## Messaging Core

The main Messaging Core building blocks are

- Receiver
- Channel
- Filter

• From these more complex building blocks can be built

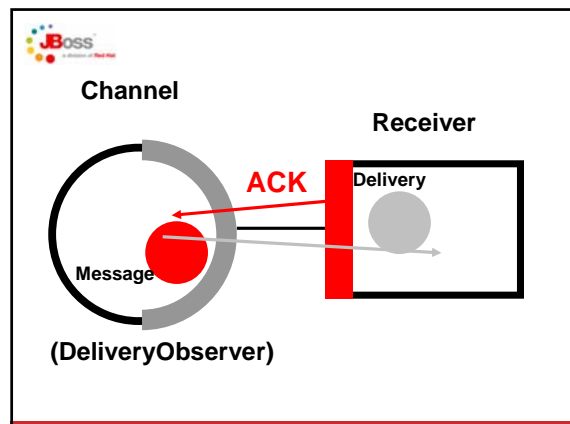
- Queue

• Also, a key component is:

- PostOffice

## Receiver

- A Receiver is the basic message handling component, that
  - Receives messages for consumption or forwarding
  - Returns a Delivery object instance for each message it receives
- The receiver then uses the Delivery instance to acknowledge the message, immediately or later
- The sender (implementing DeliveryObserver) hangs on to the Delivery, until acknowledgment (or NACK) arrives



**JBoss** **Channel**

- Channel guarantees reliability for a message
- In Core, messages flow from senders to receivers and acknowledgments flow back
- As long as a Delivery's corresponding message is reliably stored, the message IS NOT LOST even if both the sender and the receiver crash
- NOT losing messages is the Channel's job
- Channel is an extension of Receiver that can reliably hold messages

**JBoss** **Filter**

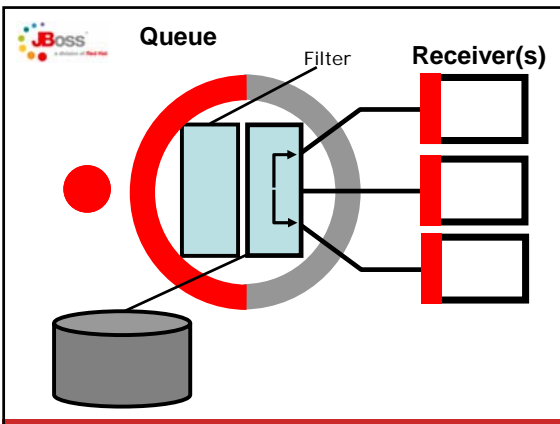
- Filter implementations contain logic for whether to accept or reject a message.
- A JMS Selector is an example of a filter
- Other filter implementations could be built for content based routing, for example

**JBoss** **All together now ...**

- Receivers, Channels and Filters are just the basic building blocks
- They are used to assemble more complex structures, such as Queues...

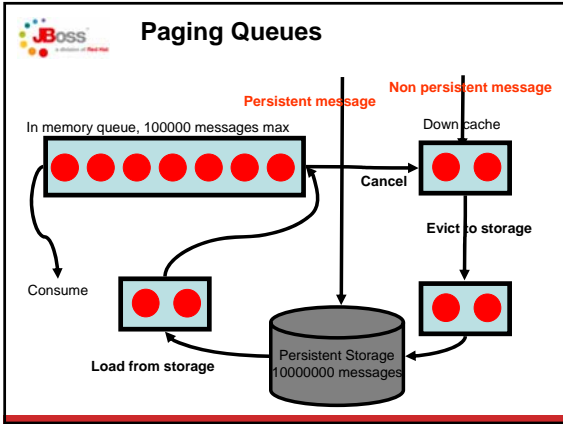
**JBoss** **Queue**

- A Queue is an implementation of Channel
- It also contains an optional Filter – it can accept or reject a message based on some criteria. (e.g. a JMS selector)
- It knows how to page messages to and from storage transparently when there are many messages in the queue
- Handles priorities
- Queue is used to implement a JMS Queue
- Queue is used to implement a JMS Topic subscription
- A Queue can have one or more Receivers attached. (E.g. JMS consumers)



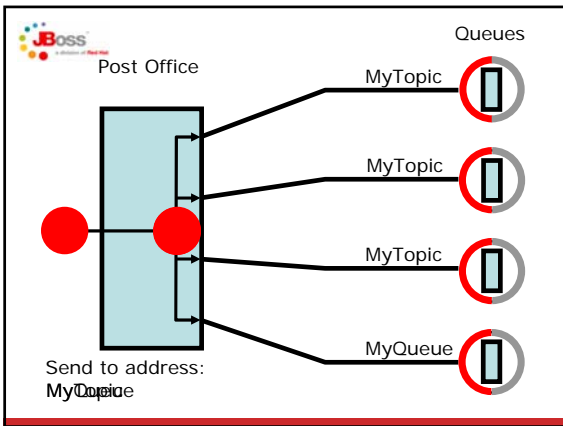
**JBoss** **Very large queues**

- Queues may need to hold many millions of messages.
- Cannot store in memory at once
- Page messages to and from storage as necessary
- JBoss Messaging can handle very large queues



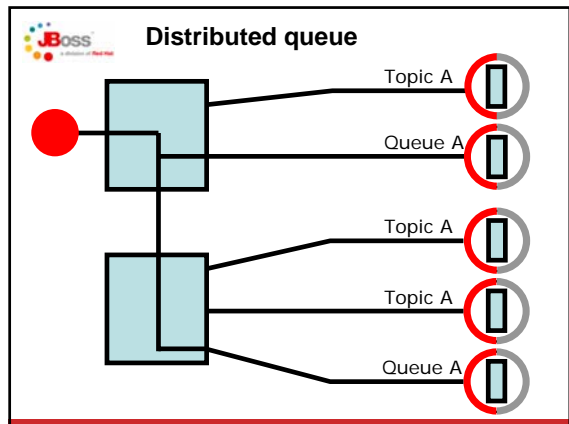
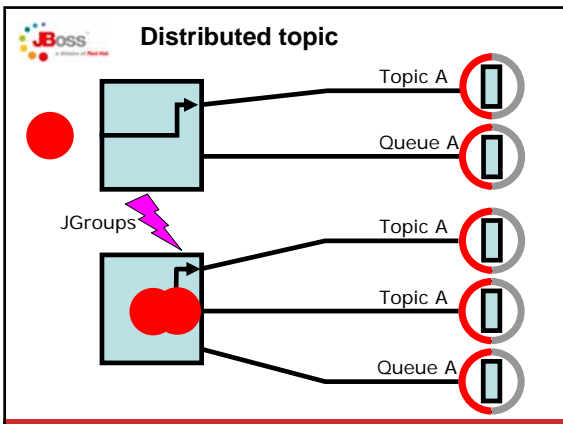
### Post Office

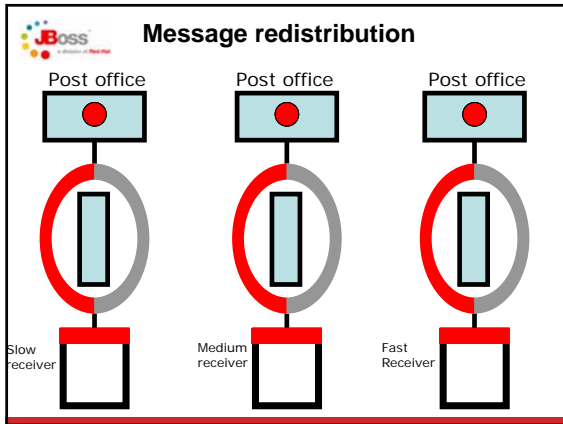
- Queues are registered with a post office under an "address"
- The post office knows how to route messages to Queues depending on the "address" given.
- A JMS Topic is just a set of Queues registered under the same "address". (The topic name)
- A JMS Queue is a single Queue registered under a unique "address" (The queue name)
- More complex implementations can be created to route based on a wildcard, for instance.
- Clustering of destinations is achieved by clustering post offices (more on this later)



### Clustered Post Office

- Post Office instances can be clustered on a LAN
- Each post office instance knows about every other post office instance in the cluster.
- Each post office instance knows exactly what queues are bound to every other queue on the cluster and what addresses are used to attach them to the post office.
- This means we can create distributed destinations....





### JBoss Distributed Destinations

**Summary:**

- Fully distributed Queues
- Fully distributed Topics
- Pluggable routing policy
- Intelligent message redistribution

### JBoss Server side failover

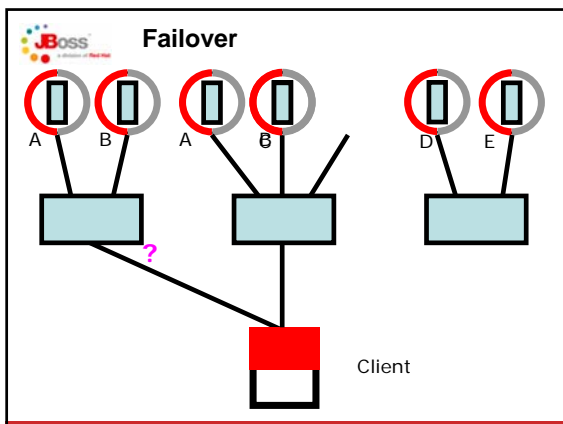
**When a server fails, the following happens on the server side:**

- The other servers detect the failure (using JGroups)
- The failover server knows which one it is, and takes over responsibility for the failed nodes Queues
- It loads the persistent state from the database (Not necessary if in memory message replication is being used)
- The server is now ready to accept new or failed over connections from JMS clients.

### JBoss Client side failover


**When a server fails, the following happens on the client side:**

- The JMS client detects the failure.
- The client tries to reconnect to what it thinks is the correct server.
- It may have the wrong server, in which case the server will redirect it to the actual correct server.
- The client carries on with its sessions.
- All this is handled automatically, no coding of failover logic is required, unlike JBossMQ.



### JBoss In memory message replication

- Instead of storing persistent messages in a shared database, messages can be replicated in memory between nodes.
- This can give a high degree of durability, comparable to persisting in a database if the hardware is arranged well.
- Replicating messages across the LAN between nodes is likely to be much faster than persisting to disk and disk syncing.
- This eliminates the shared database as a bottleneck.
- This could provide a highly scalable and highly performant solution without sacrificing reliability.
- Due in 1.2.1
- Overcomes problems inherent with local databases too




## Putting it all together

- Reliable queues that can cope with millions of messages
- Queues attached to post offices according to certain "addresses".
- Post offices clustered in groups to form fully distributed queues and topics.
- Message redistribution to smoothly move messages between queues on nodes according to load.
- In memory message replication to avoid a shared database as a bottleneck. (TODO)


Enables:

- Highly scalable, highly performant, highly reliable, grid based model for message processing.



## JBoss Messaging Roadmap

- **v1.0 – Available since March 2006**
  - Early adopters, developer focused release
  - High performance core, JMS 1.1 compliant
  - Single node focused
  - Silver support only
- **v1.2 - the "Clustering" release – 1Q 2007**
  - Enterprise production release
  - JBossMQ upgrade path
  - Full clustering implementation
  - "ESB ready" foundation
  - Unreliable link scenario
  - Silver, Gold, Platinum support
- **V1.2.1 – 2Q 2007**
  - NIO transport
  - Very large message support
  - Tuning and optimization for high end and ESB scenarios
- **v2.0 – AMQP and optimization – 4Q 2007**
  - Support AMQP protocol
  - WAN support



## Any questions?