Getting Started with ActiveMQ

Scott Cranton
Solution Consultant Manager

FuseSource
integration everywhere

Agenda

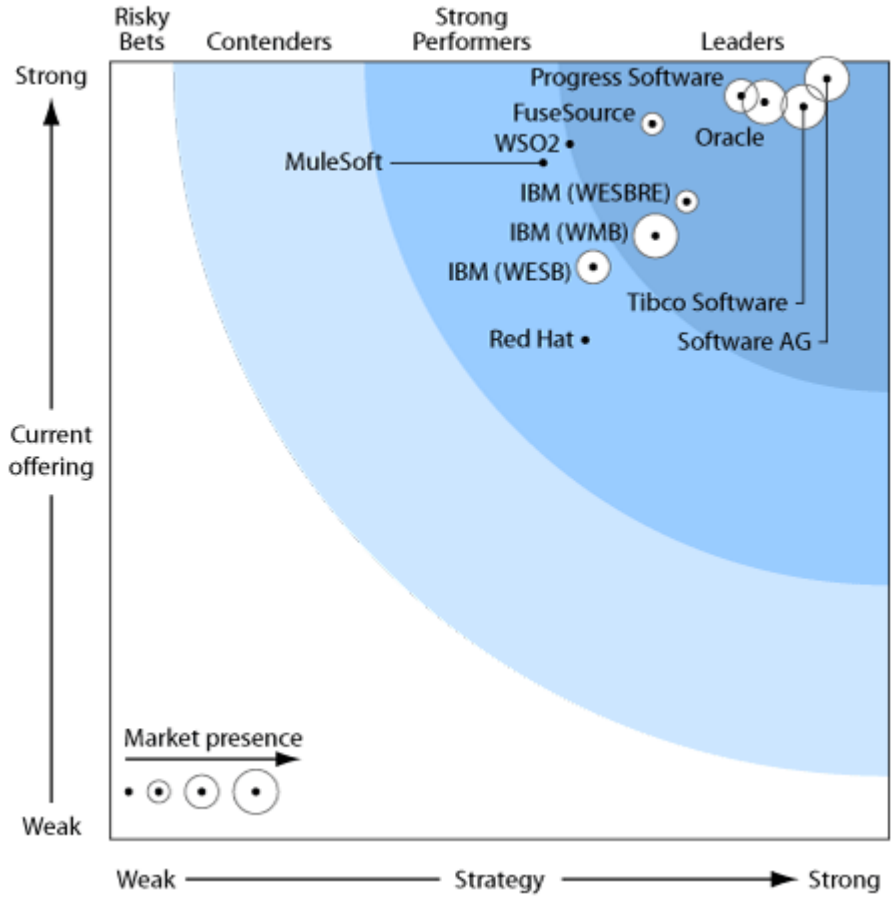
- Who's FuseSource?
- ActiveMQ Overview
  - Core capabilities
  - Managing client connections
  - Managing persistence
  - High availability
  - Network of brokers
- Demo
  - Walk through install
  - Start/Stop with alternative configuration
  - Management through JMX
  - High availability: failover and back

FuseSource - the Leading Open Source Integration and Messaging Vendor

- **Company built on success**
  - Founded in 2005
  - Double digit year-over-year growth
  - Offices in 9 time zones
- **Enterprise products that integrate everything**
  - Community-backed, open source products
  - Proven track record in mission-critical apps
  - Leader status in Forrester ESB Wave

Forrester Wave Report Q2 2011: Fuse ESB is a "Leader"

- FuseSource placed in "Leader" category in company with large, established vendors
- One of few open source solutions considered for this report
- Highest ranked open source solution



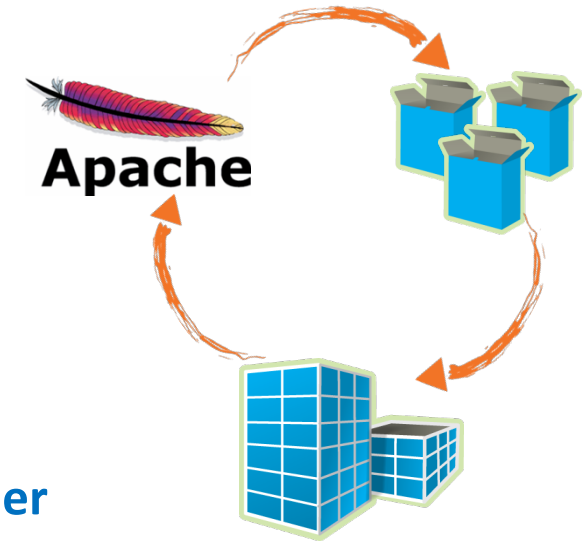
Bringing Open Source Integration & Messaging to Enterprise IT

Apache Software Foundation

- ActiveMQ (reliable messaging)
- Camel (Ent. Integration Patterns)
- ServiceMix / Karaf (containers)

Enterprise OSS Products:

- Integrated solutions
- Tested and certified
- Documented



Subscriptions

Collaborative relationship with your software provider

- Enterprise tooling
- Services level agreement
- WW support organization

Training & Consulting

- Expert training on site or via the Web
- Packaged services for all phases of the lifecycle

FuseSource: the Leaders in Open Source Integration

- Open source community expertise
  - Co-founders and PMC members of ServiceMix, Karaf, ActiveMQ, Camel, and others
  - Over 25 active committers on 11 Apache projects
- Enterprise training and consulting designed for success
  - Training: getting started -> production readiness -> management
  - Consulting: PoC Workshop -> Architectural Assessments -> Tuning & HA configuration -> Go-Live Assessment
- Proven enterprise IT success
  - 200+ customers
  - 3 of top 5 retailers in the world

FuseSource Subscription: Collaborative relationship with your software provider

Support

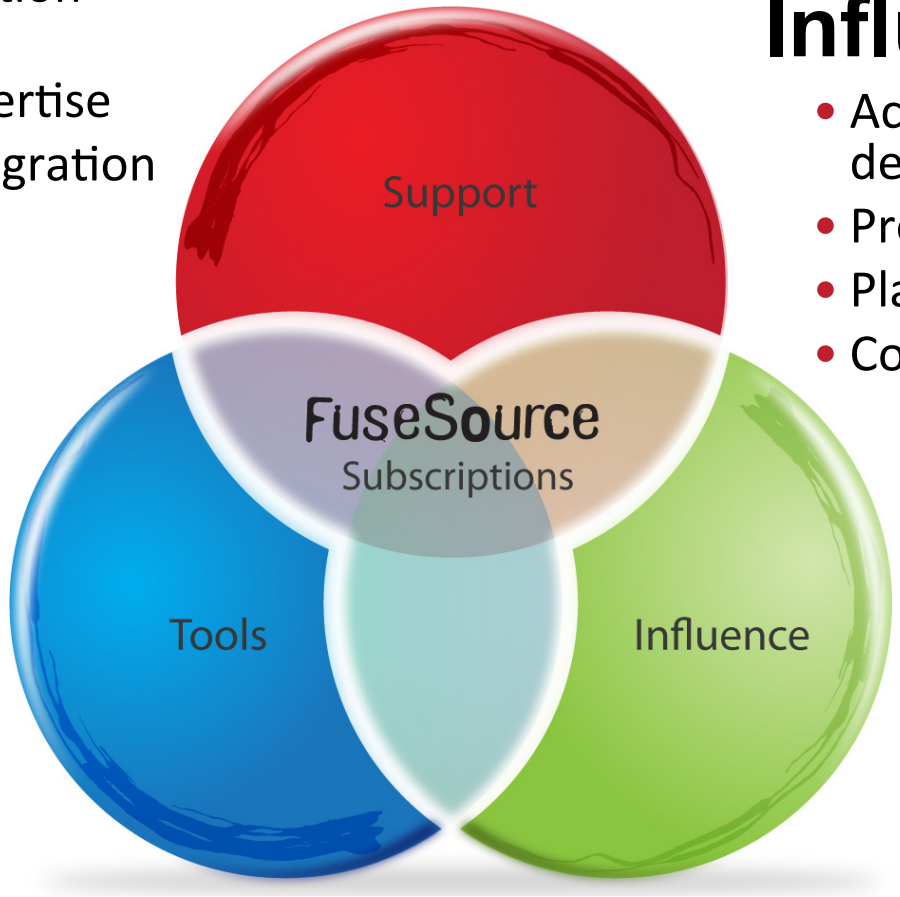
- Enterprise-class 24x7 coverage
- Global organization
- Mission-critical integration expertise
- Updates and migration assistance

Influence

- Access to the development team
- Product roadmaps
- Planning processes
- Conduit to Apache

Tools

- Certified distributions
- Dev, ops, and management tools
- Performance tuning
- Documentation

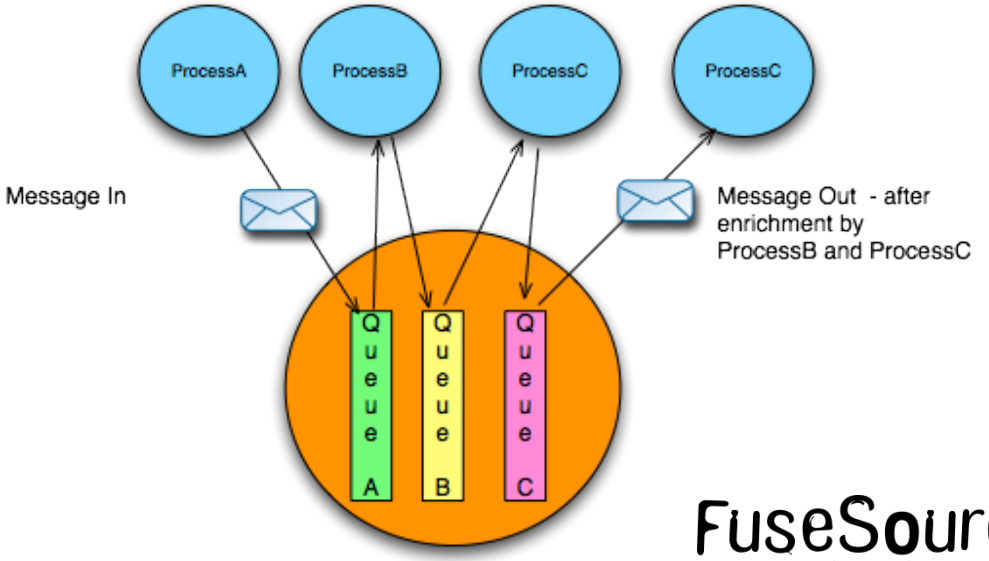


What is Apache ActiveMQ?

- Top level Apache Software Foundation project
- Wildly popular, high performance, reliable message broker
  - Supports JMS 1.1; adding support for AMQP 1.0 and JMS 2.0
  - Supports publish/subscribe, point to point, message groups, out of band messaging and streaming, distributed transactions, …
  - Fault Tolerance and High Availability
- Myriad of connectivity options
  - Native Java, C/C++, and .NET
  - STOMP protocol enables Ruby, JS, Perl, Python, PHP, ActionScript, ...
- Embedded and standalone deployment options
  - Pre-integrated with open source integration and application frameworks
  - Deep integration with Spring Framework and Java EE

Why Use Messaging?

- Reliable remote communication between applications
- Asynchronous communication
  - De-couple producer and consumer (loose coupling)
- Platform and language integration
- Fault tolerant - processing can survive Processor outage
- Scalable - multiple consumers of each queue
  - Distributes processing



What is “High Availability”?

- Always Ready – Infrastructure **appears** to always be ready
- Change Transparency – Infrastructure changes are **transparent** to application
- Fault Tolerance – maintain quality of service **despite** system failures
- Scalable – ability to grow (and shrink) infrastructure to meet needs

High Availability Concerns

- Application (messaging client)
  - (Re)Connect to messaging infrastructure automatically
  - Ignorant (mostly) of messaging infrastructure configuration
  - Balancing messaging quality of service tradeoffs
- Infrastructure (messaging broker)
  - Fault Tolerance – balance cost of outage prevention
  - Ability to scale out (and in) horizontally as needed

Application Recommendations for High Availability

- Always use Connection Pooling
  - Single biggest issue with large deployments
  - Most applications work fine with a **pool** of 1 connection
- Always use Failover transport
  - Transparently re-connect on connection failure
  - Works with discovery
  - Can rebalance connections across group of brokers
- Example connection URI
  - failover:(tcp://master:61616,tcp://slave:61616)?random=false
  - failover:(tcp://broker:61616)

Application Recommendations for High Availability

Balancing messaging quality of service

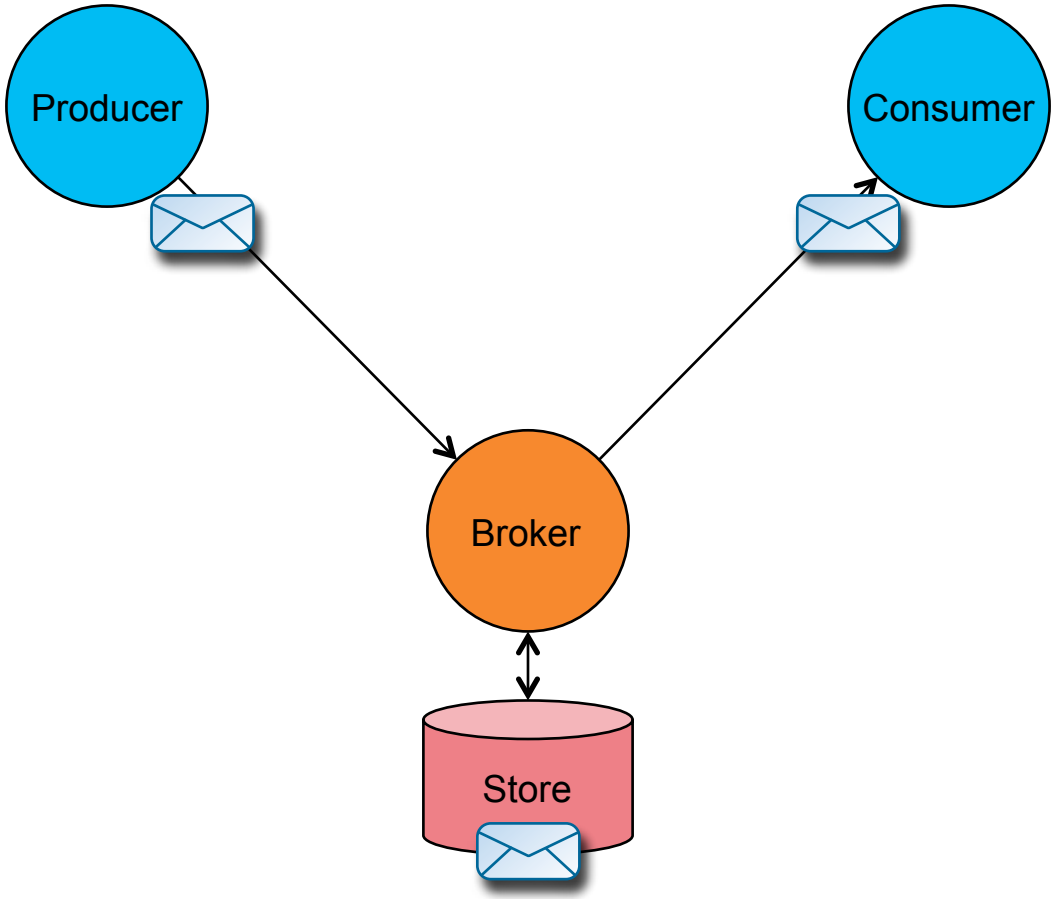
- Persistent versus non-persistent – Fault Tolerance
  - Persistent messages saved to store (most reliable, but slower)
  - Non-persistent are not (fastest, but lost on broker failure)
- Synchronous versus Asynchronous send - Throughput
  - Synchronous (default) means client thread waits for broker to acknowledge message receipt
  - Asynchronous (JMS non-standard) means client thread does not block
- Transactions – not a silver bullet
  - XA / 2 phase transactions are most reliable (in theory), but can be difficult to get working, and are slow
  - Using JMS (local) transactions for message batch sending can increase throughput

Infrastructure Recommendations for High Availability

Understand difference between Master/Slave and Network for Brokers

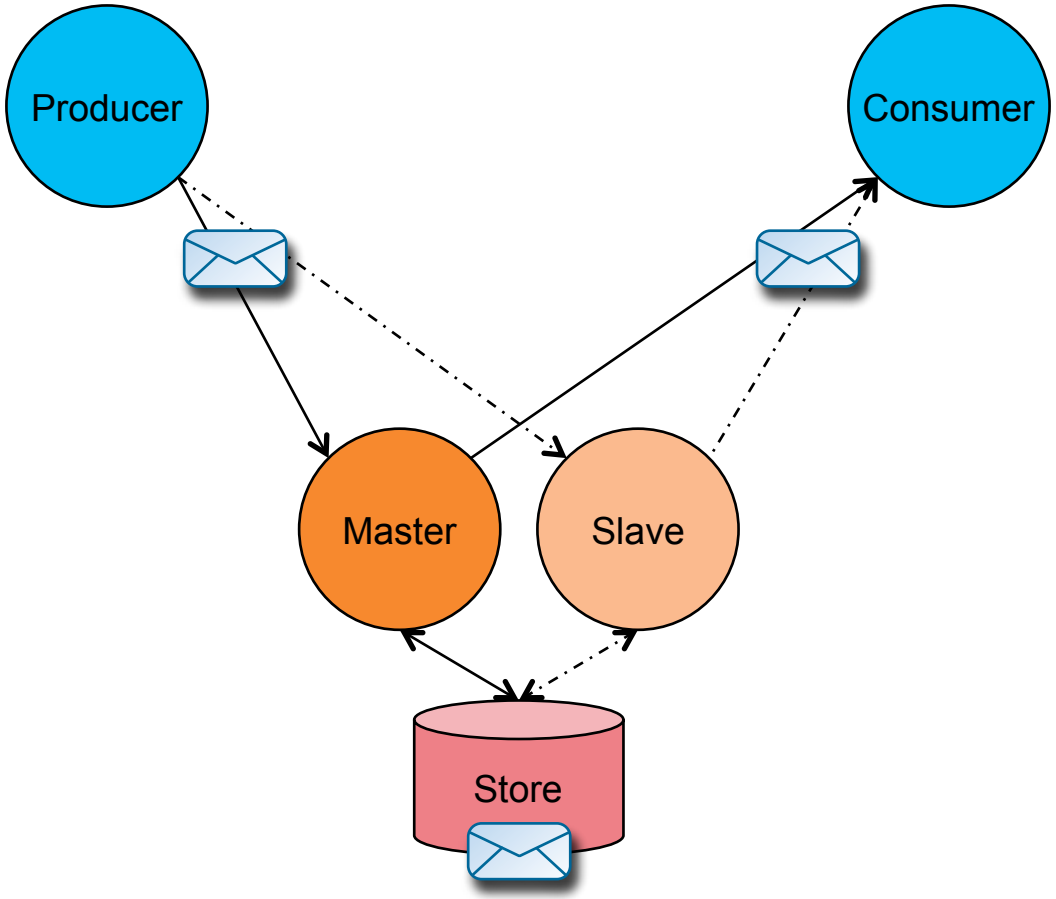
- Master/Slave
  - Multiple broker instances (generally 2) on same persistence store
  - Helps ensure **timeliness** of message delivery
- Network of Brokers (Broker Federation)
  - Connect group of brokers together
  - Messages forward through network of brokers
  - Enables horizontal scaling and multi-location reliable messaging
- Combination
  - Network of Master/Slave pairs
- Note: Persistent messages depend on integrity of persistence store technology (disk or database)

Single Broker



failover:(tcp://broker:61616)

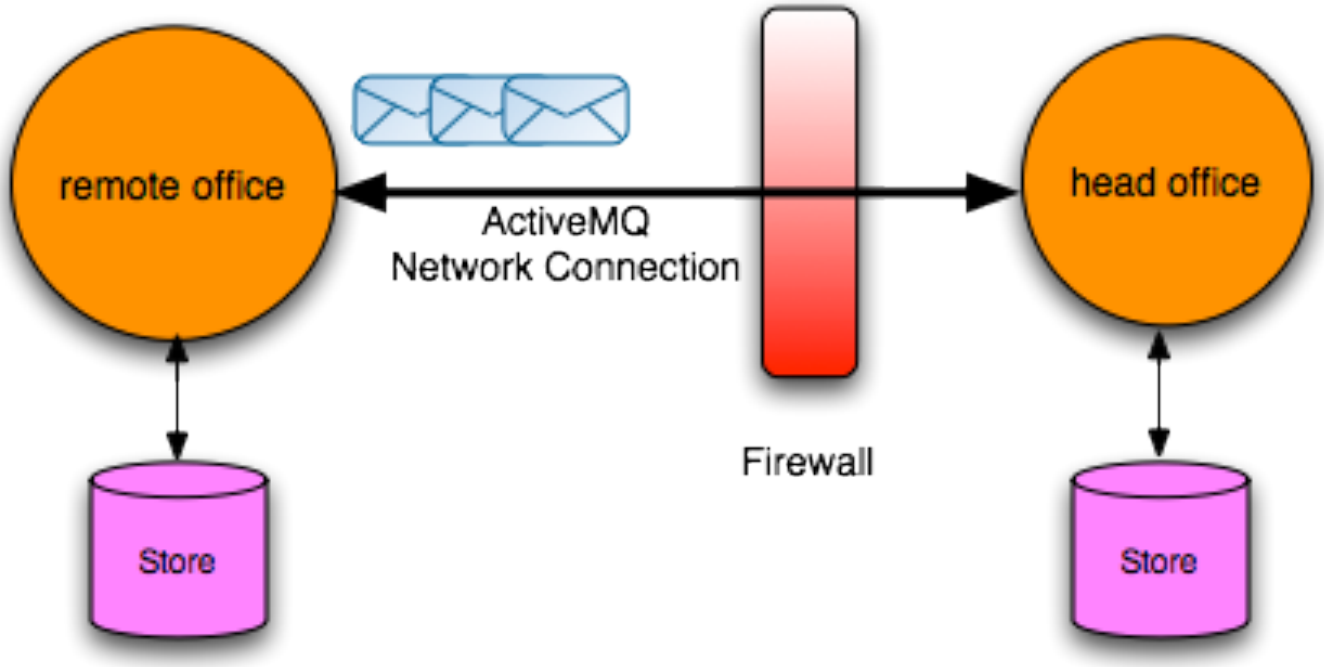
Master / Slave Brokers



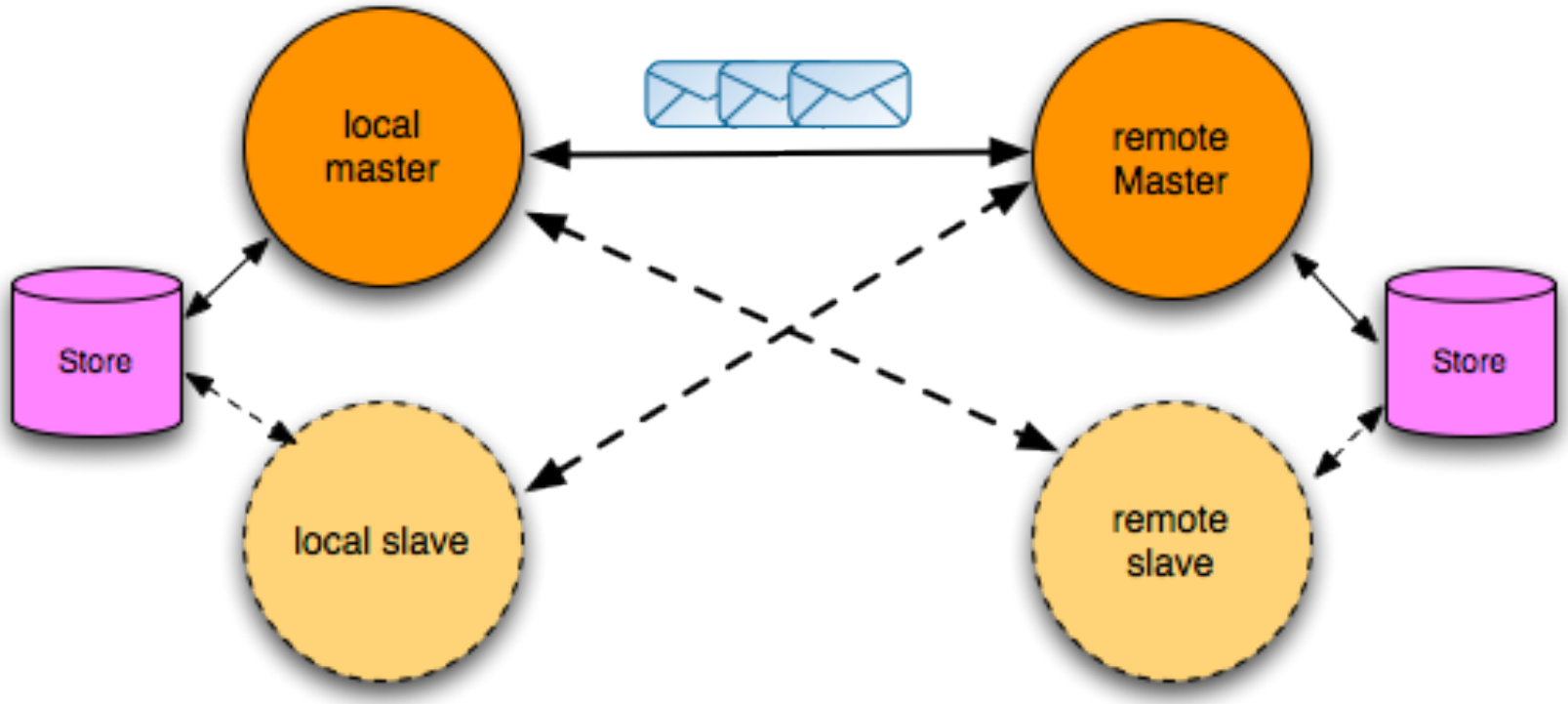
failover:(tcp://master:61616,tcp://slave:61616)?random=false

Network of Brokers : Geographically Dispersed

Network of Brokers : Geographically Dispersed



Network of Brokers : Network with Master/Slave



Network of Broker Notes of Caution

- Network Connector does two (2) things
  - Manages a network (socket) connection
  - Bridges message destinations
- Dynamic versus static bridging
  - Dynamic bridging uses Advisory Topics
    - Four+ Topics per Destination
  - Advisory messages sent when
    - Clients connect and disconnect
    - Destinations are created and destroyed
  - Static bridging requires manual configuration of destinations
    - Can use destination wildcards
    - Does not need Advisory Topics

Network of Broker Notes of Caution

Network of Master / Slave pairs

- Fixed as of ActiveMQ 5.5 (AMQ-3542)
  - static:(failover:(tcp://master,tcp://slave)?**maxReconnectAttempts=0**
- Failover transport usage means only connect to one and only one of listed brokers
- maxReconnectAttempts=0 means failover transport should NOT transparently reconnect on failure; allow static discovery transport to reconnect
  - Bridge code needs to know when connection failures happen
- As of ActiveMQ 5.6
  - masterslave:(tcp://master,tcp://slave)

Code Time

FuseSource
integration everywhere