cohesiveFT

Cohesive Flexible Technologies

AMQP Business Messaging
for Predictable, Scalable, Available SOA
Alexis Richardson

Microsoft Architects Insight Conference 2008

Agenda

- The business problem facing SOA - ‘more of everything’
- Focus on a key aspect: using messaging to scale predictably
- A solution: anew wire level open business messaging protocol - AMQP
- The AMQP Working Group
- Using AMQP at CohesiveFT

Three definitions of SOA

An approach to business/IT alignment: driven by business instead of technology, relying on strong governance and implemented using any technology.

A technical architecture: service oriented with clearly defined interfaces, and could be technology-independent.

Web Services: business data as XML messages implemented using WS-* stack.

SOA is part of a larger problem

Trends

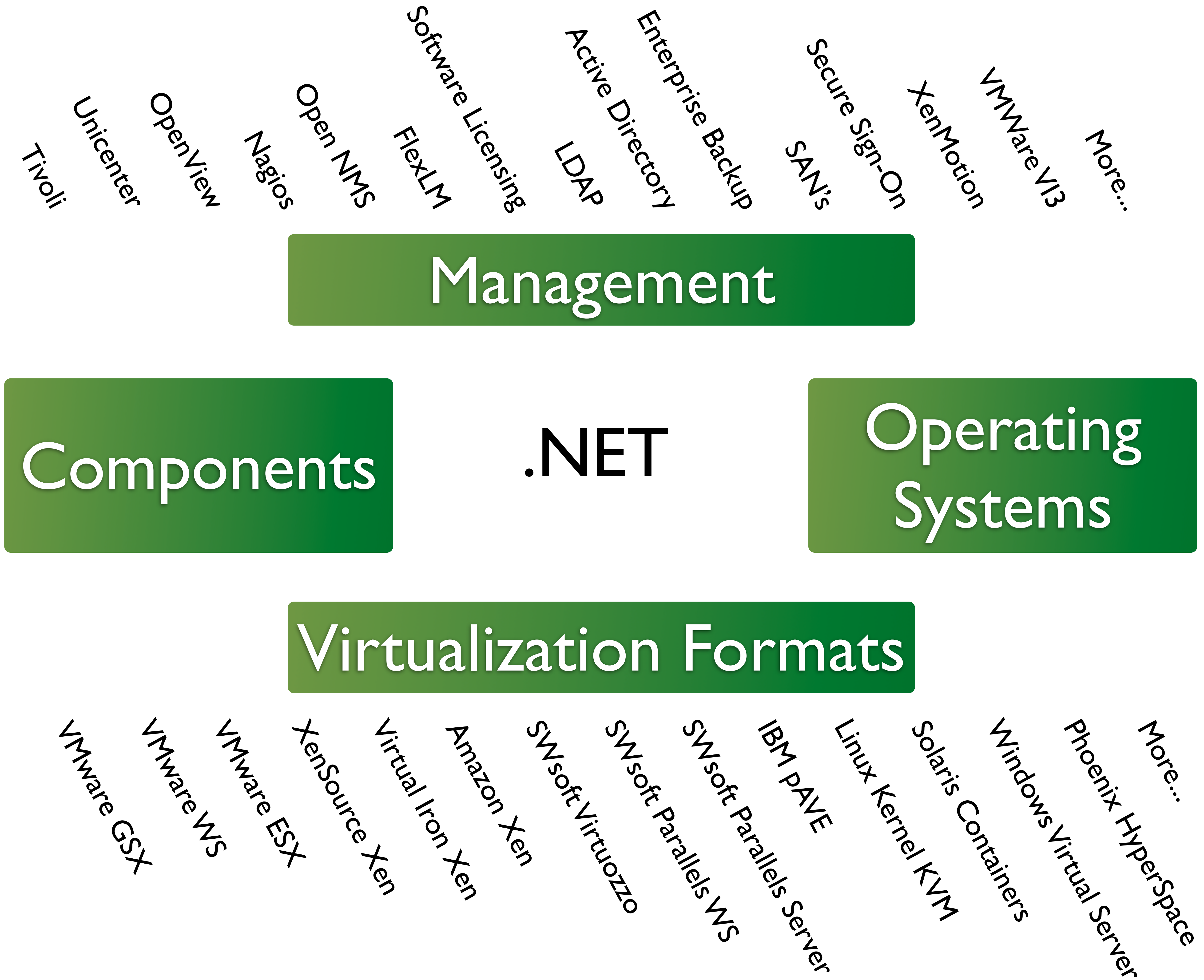
	Traditional	*Emerging*
Vendor	Single Source	Many Suppliers, including Open Source
Standards	Proprietary	Open Standards
Application Architecture	Monolithic Apps	SOA
Application Stack	Single Vendor Middleware Stack	Service Oriented Infrastructure (SOI)
Deployment	Static/Physical	Dynamic/ Virtual

>

Impact

“EVEN MORE OF EVERYTHING” PROBLEM

MUCH MORE INTEGRATION NEEDED



Tivoli
Unicenter
OpenView
Nagios
Open NMS
FlexLM
Software Licensing
LDAP
Active Directory
Enterprise Backup
SAN's
Secure Sign-On
XenMotion
VMWare VI3
More...

Management

Products
Community
Certified Community
Open Source
ISV /Commercial
IBM
.NET
Tibco
BEA, Oracle
More..
and more ...
Customer Proprietary
Vertical Industry
More....

Components

.NET

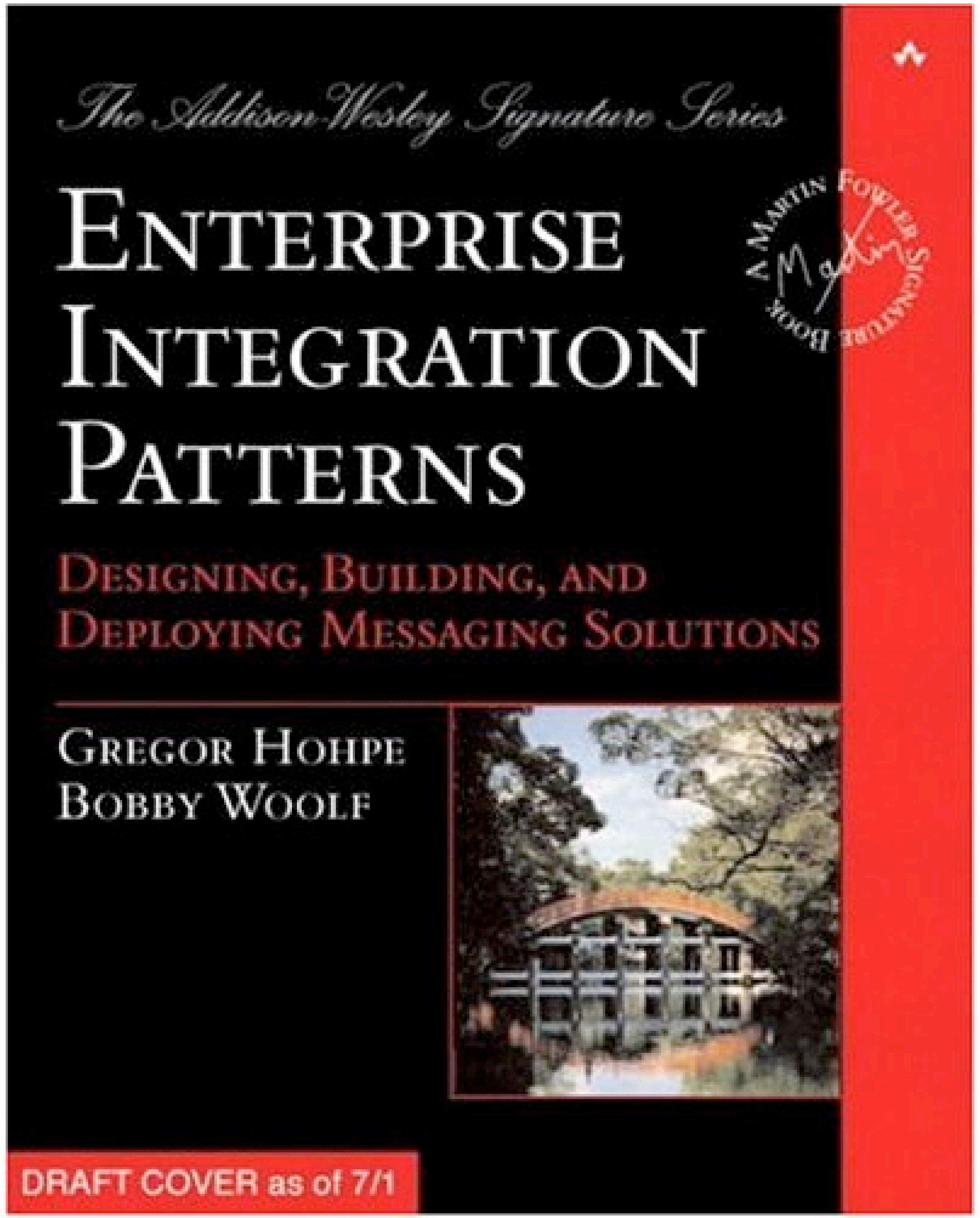
Operating Systems

Debian
Free Ubuntu
Canonical JEOS
Canonical Ubuntu
Fedora
RedHat
RedHat AOS
BEA WLS OS Shim
openSUSE
SUSE Enterprise
Windows Server
Red Flag Linux
CentOS
Gentoo
Mac OS X Server
More...

Virtualization Formats

VMware GSX
VMware WS
VMware ESX
XenSource Xen
Virtual Iron Xen
Amazon Xen
SWsoft Virtuozzo
SWsoft Parallels WS
SWsoft Parallels Server
IBM pAVE
Linux Kernel KVM
Solaris Containers
Windows Virtual Server
Phoenix HyperSpace
More...

MANY PATTERNS - MANY PLATFORMS



The Addison-Wesley Signature Series

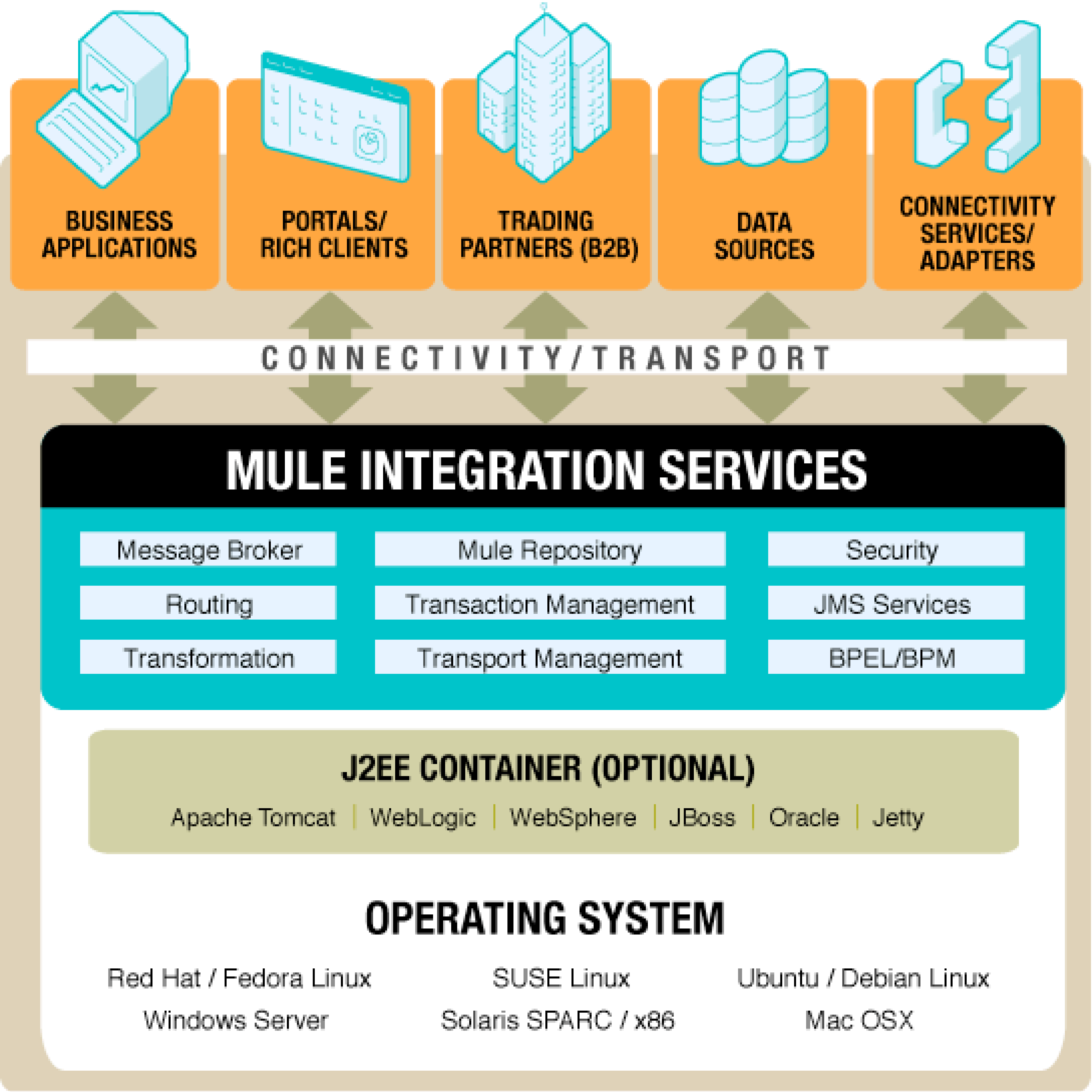
ENTERPRISE INTEGRATION PATTERNS

DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS

GREGOR HOHPE
BOBBY WOOLF

A MARTIN FOWLER SIGNATURE BOOK

DRAFT COVER as of 7/1



BUSINESS APPLICATIONS

PORTALS/ RICH CLIENTS

TRADING PARTNERS (B2B)

DATA SOURCES

CONNECTIVITY SERVICES/ ADAPTERS

CONNECTIVITY/TRANSPORT

MULE INTEGRATION SERVICES

Message Broker | Mule Repository | Security

Routing | Transaction Management | JMS Services

Transformation | Transport Management | BPEL/BPM

J2EE CONTAINER (OPTIONAL)

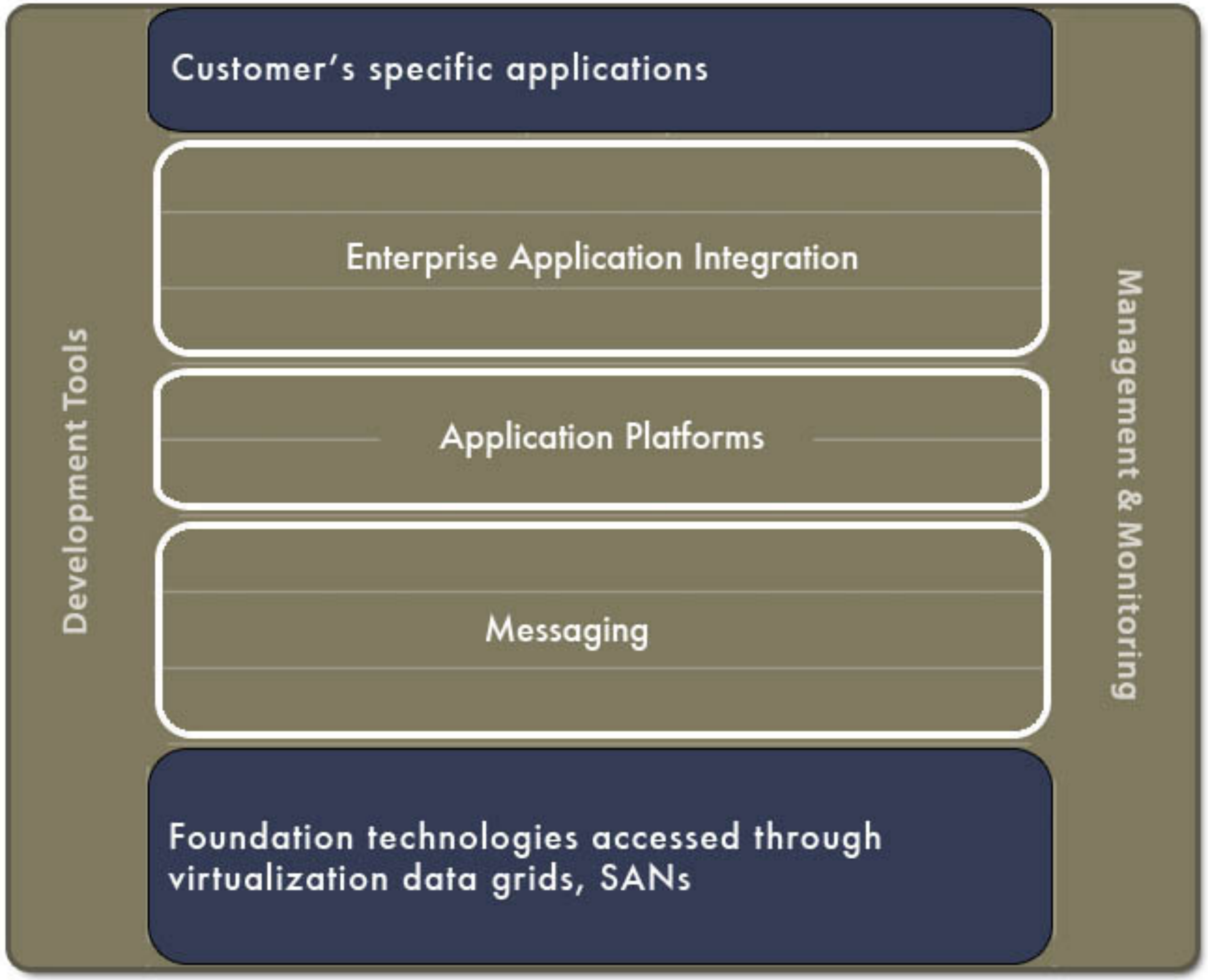
Apache Tomcat | WebLogic | WebSphere | JBoss | Oracle | Jetty

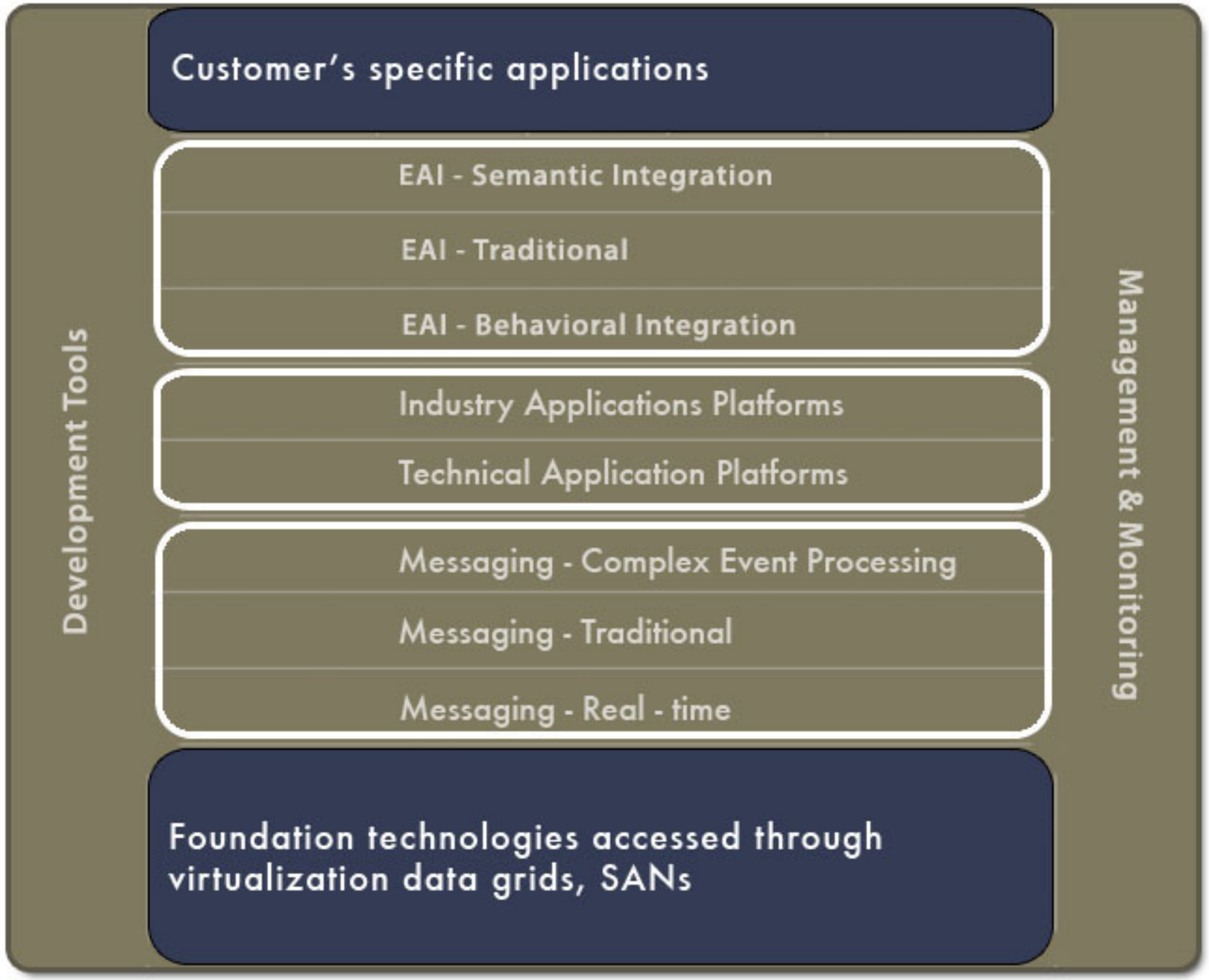
OPERATING SYSTEM

Red Hat / Fedora Linux | SUSE Linux | Ubuntu / Debian Linux

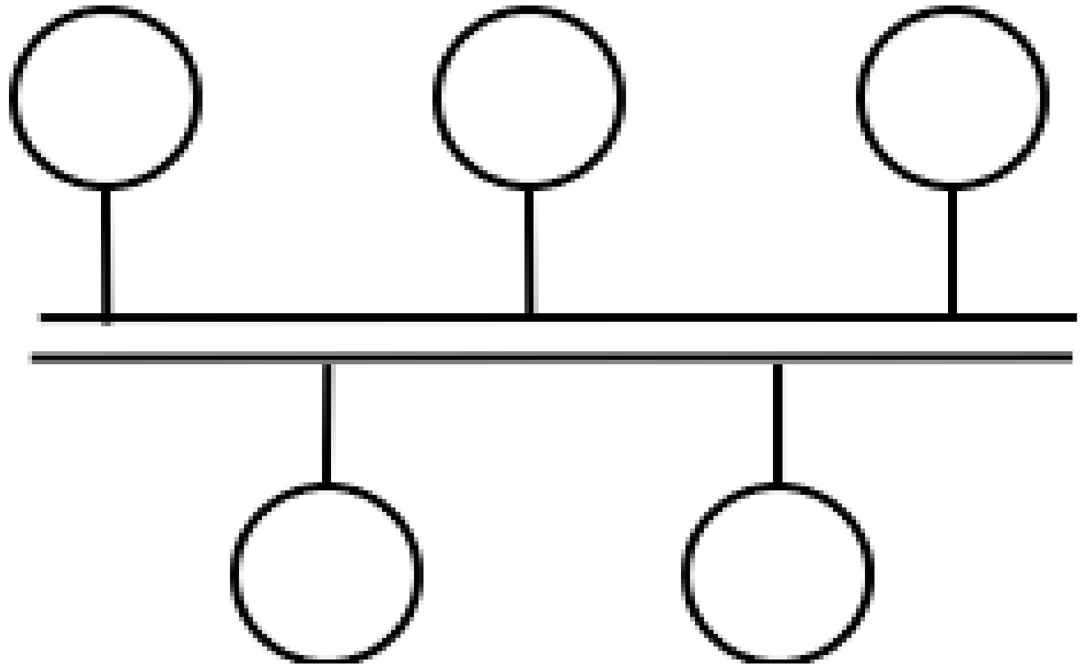
Windows Server | Solaris SPARC / x86 | Mac OSX

NO ‘ONE STACK’- PATTERNS ARE ‘FAMILIES’

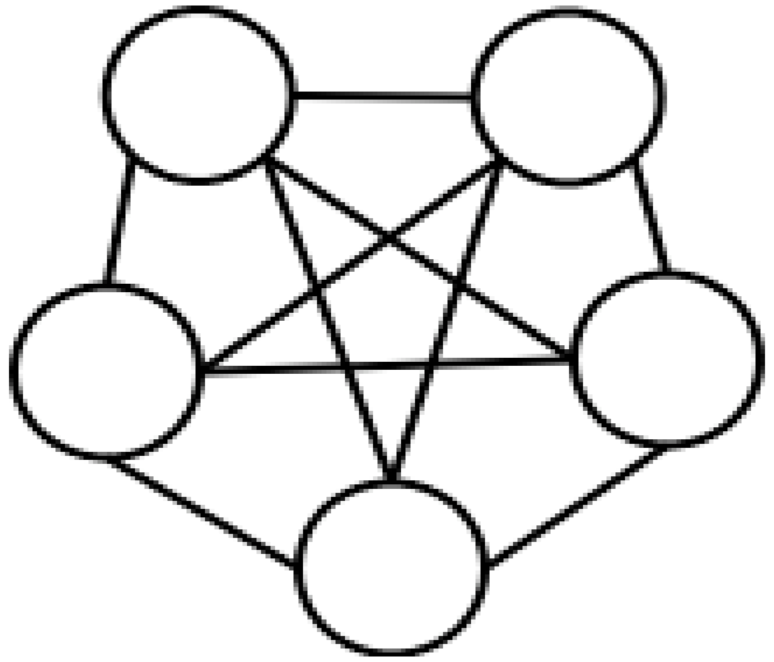

Customer’s specific applications
Enterprise Application Integration
Application Platforms
Messaging
Foundation technologies accessed through virtualization data grids, SANs
Development Tools
Management & Monitoring


Customer’s specific applications
EAI - Semantic Integration
EAI - Traditional
EAI - Behavioral Integration
Industry Applications Platforms
Technical Application Platforms
Messaging - Complex Event Processing
Messaging - Traditional
Messaging - Real - time
Foundation technologies accessed through virtualization data grids, SANs
Development Tools
Management & Monitoring

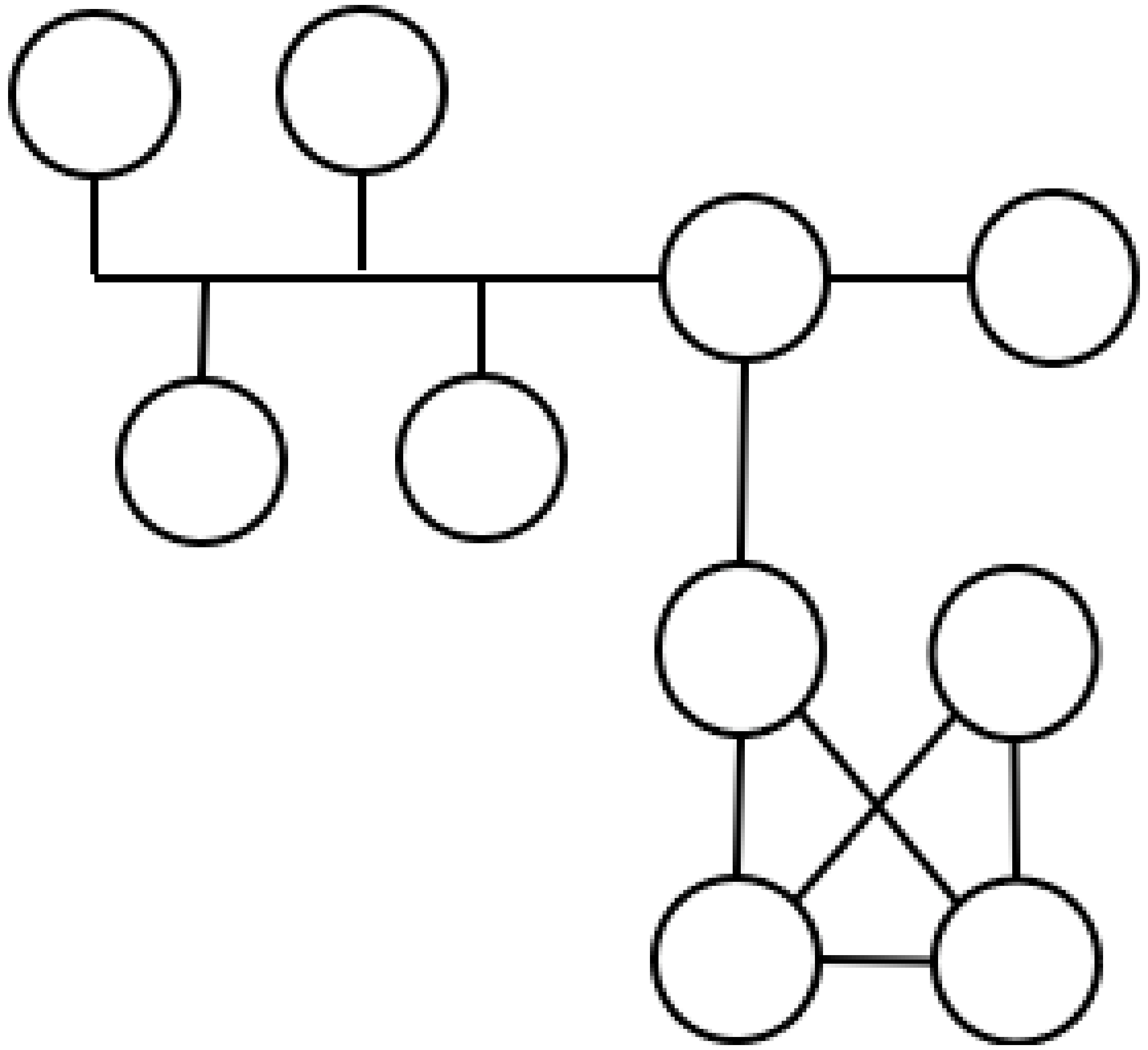
MANY TOPOLOGIES



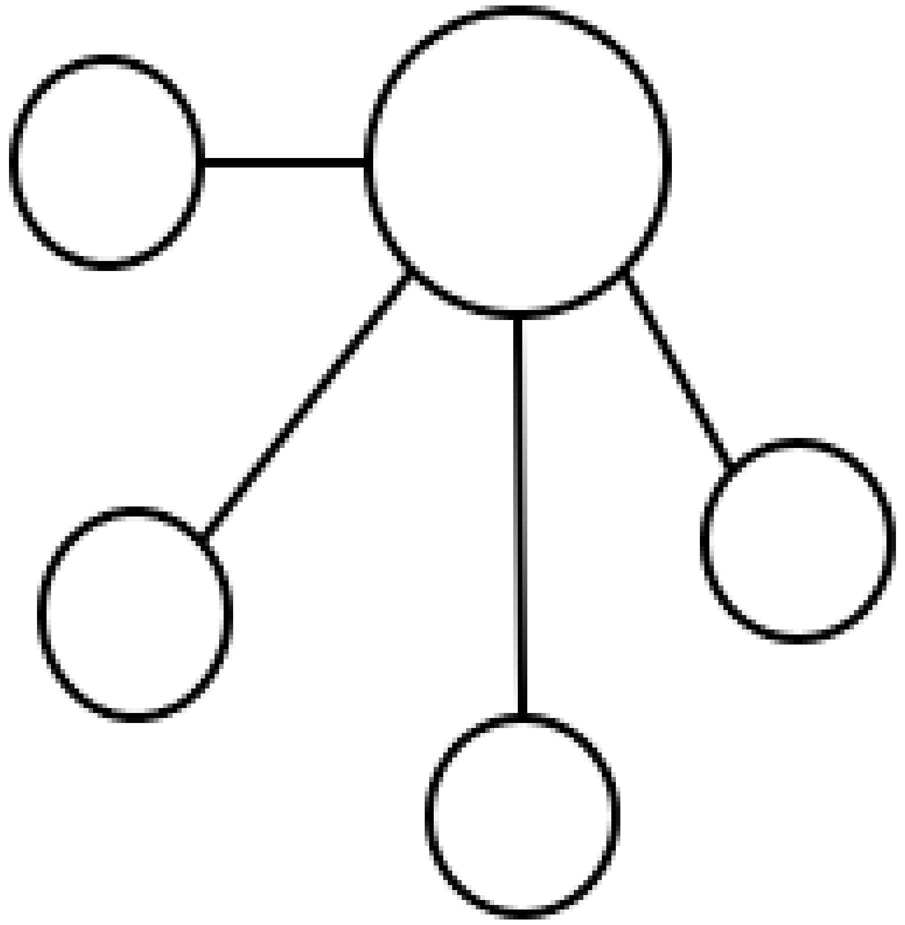
Enterprise Service Bus



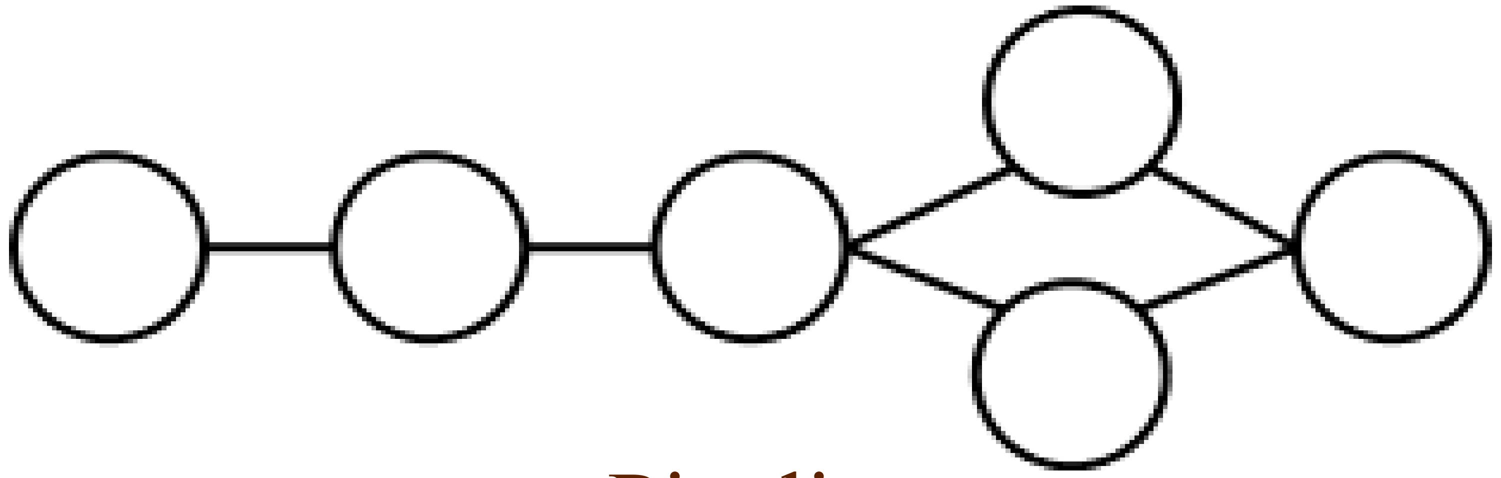
Peer Network



Enterprise Service Network



Client/Server and Hub n' Spoke



Pipeline

Impact on scalability

- More complex
- Harder to integrate
- Too much custom tuning
- Local dependencies make it harder to scale uniformly
- Not ready for the new world - eg of ‘the cloud’, virtualization, mesh, ..

SOLUTION REQUIREMENTS

Solution requirements

- Very simple - low level - preferably wire level
- Integration is based on interoperability - like TCP - not like portable APIs
- Underlying fabric is itself reliable, available, scalable, performant
- Can be embedded with the technology you already use
- Not just LAN - must be WAN, virtualization, Cloud and Mesh ‘ready’

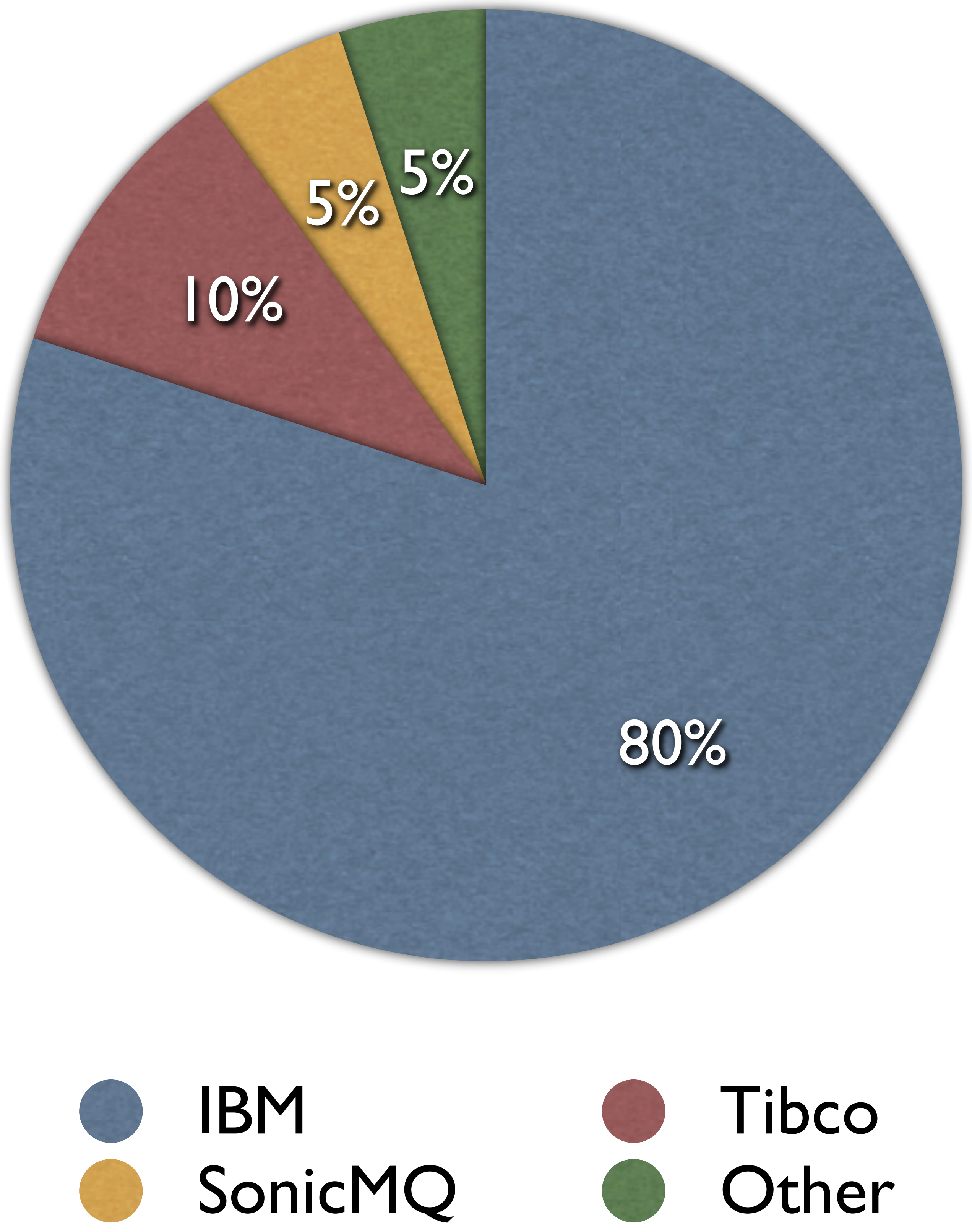
Messaging

- Asynchronous
- Not language bound as such
- Well understood model - predictable project outcomes
- Proven to scale - can be made reliable, available
- Lots of solution patterns, works with .NET WCF, and Mule, (and, and, and...)
- Is not LAN-bound or silo-bound - can be ‘cloud ready’ (licensing aside..)

WHY MESSAGING AS WE KNOW IT IS NOT ENOUGH

A long time ago, in a market far far away

Message oriented middleware products = $2.5bn-3bn



Middleware versus protocols

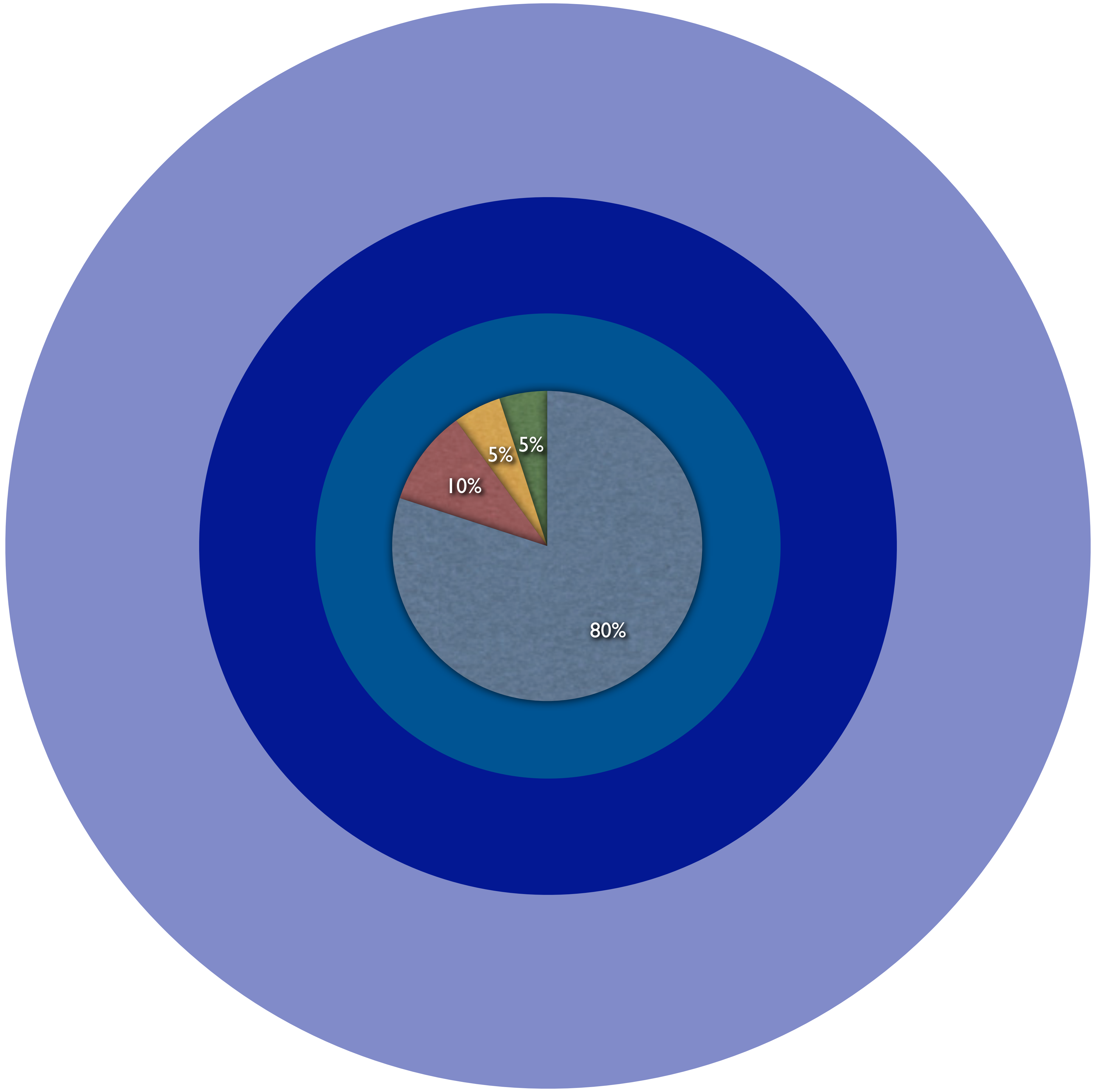
- Middleware = complex, proprietary, requires installation and customisation, integration services from consultants with knowledge of many platforms or languages, then maintenance is done by the customer, which is then followed by system aging, bloat, and eventual heat death.

- TCP, SCTP, HTTP, SMTP = simple, standard, ubiquious and no customisation needed, no integration required from consultants, maintenance is done by the vendor, and is proven to outlast the lifetime of the average software company (and as we now see, some banks)

Protocols vs Products

- Products have APIs
  - installed - requires consultants
  - customised / integrated
  - maintained by customer
  - proprietary, or ‘language bound’
  - complex, ..
- Protocols guarantee behaviours (and interop.)
  - ubiquitous - preinstalled everywhere - and anywhere
  - deliver integration for free instead of needing to be integrated
  - maintained as part of larger product or solution
  - standard and open
  - simple - plug and play - language neutral
- Imagine if we had no TCP and had to use ‘IBM NetSphere’
- Imagine if we had no HTTP and had to use ‘Microsoft Home Network’
- Imagine if we had no SMTP and had to pay per message like SWIFT

But market potential is MUCH larger for the right technology



$70-80bn?

Business Messaging = like email or IM but you can send money over it

A protocol for business messaging?

	unreliable	reliable
async	SMTP	?
sync	HTTP	IIOP

What goes in here will clean up if it is OPEN, UBIQUITOUS, & ADAPTABLE

Introducing Advanced Message Queue Protocol

Everyone uses email without thinking, so then why is commercial business messaging so hard?

→ need an ***Open Standard Protocol*** for *Message Oriented Middleware*

AMQP aims to become
THE standard for business messaging

Made to satisfy real needs:

- created by users and technologists working together (www.amqp.org)
- in development for 3+ years, went public with AMQP 0-8 June 20th 2006
- AMQP 0-10 adds reliable delivery “what goes in, must come out”

“business dialtone”

Business messaging - why a protocol is new technology

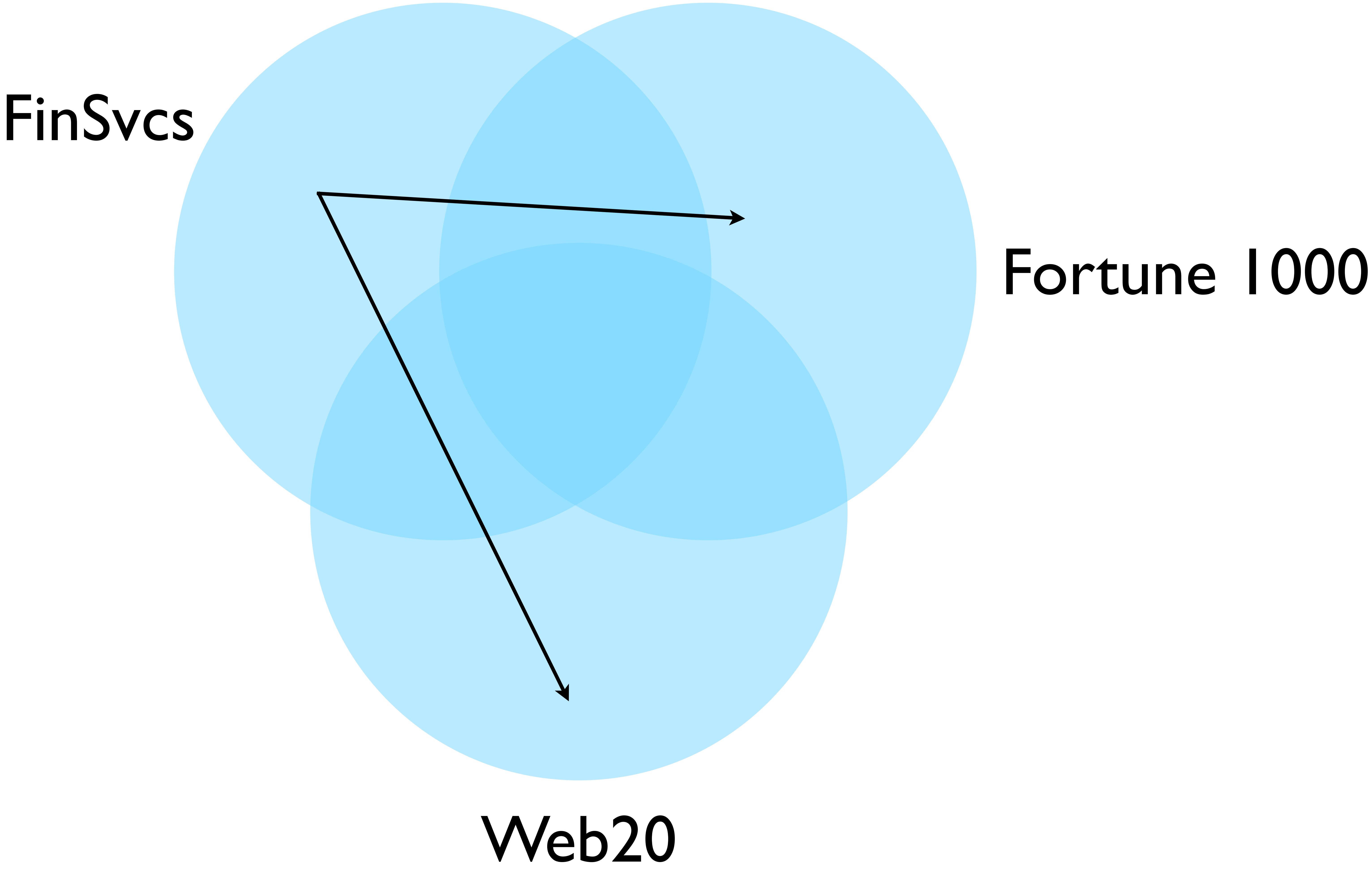
- Transport level - just above TCP, UDP, .. just below SOA tools and WS-*
- “As easy as email” but wire level, binary, for business messages
- Push ‘hard’ requirements down to standard wide-area fabric
  - “A reliable and scalable mesh for SOA”
- Technical specifics
  - Fidelity - “what goes in must come out” - can cope with failure
  - Security - transactions - “you can send money over it”
  - Many to many conversations, long running streams
  - Management, entitlements, addressing
  - Relays - addressing, smart routing, federation
  - Essentially - an intermediated protocol

Who stands to gain from this? YOU

- JPMorgan
- Goldman
- Credit Suisse
- Deutsche Boerse ..... all have ‘put up’ rather than ‘shut up’

- WHY?
- Integration that scales - made cheaper by a standard interoperable protocol

Who might use this?



Protocol design

Comparison with other protocols

- SMTP – unreliable, slow
- HTTP – synchronous, unreliable, no routing
- XMPP – not binary, no delivery fidelity, no pubsub or routing, no queue management
- FTP – point to point, transient, does not work well with NAT/SSL
- MQ – exactly once, proprietary, non-interoperating even when portable
- TCP – at least once, reliable but short lived, no app level state mgmt
- UDP – fast but has no delivery guarantees

AMQP can do all of the above as 'use cases'

… and switch between them

Intermediation - delegation - a trusted brokered model

- On the one hand we want wire level interoperability - like TCP - it just works
- Unlike TCP and HTTP, delivers true MESSAGING

This requires intermediation - a ‘protocol for brokers’

- Routing and addressing - “to:phil@cohesiveft.com”
- Smart routing - “buy.ibm.100”
- Delegation - trusted delivery
  Think: “give this to …”
  Or: “what goes in must come out”

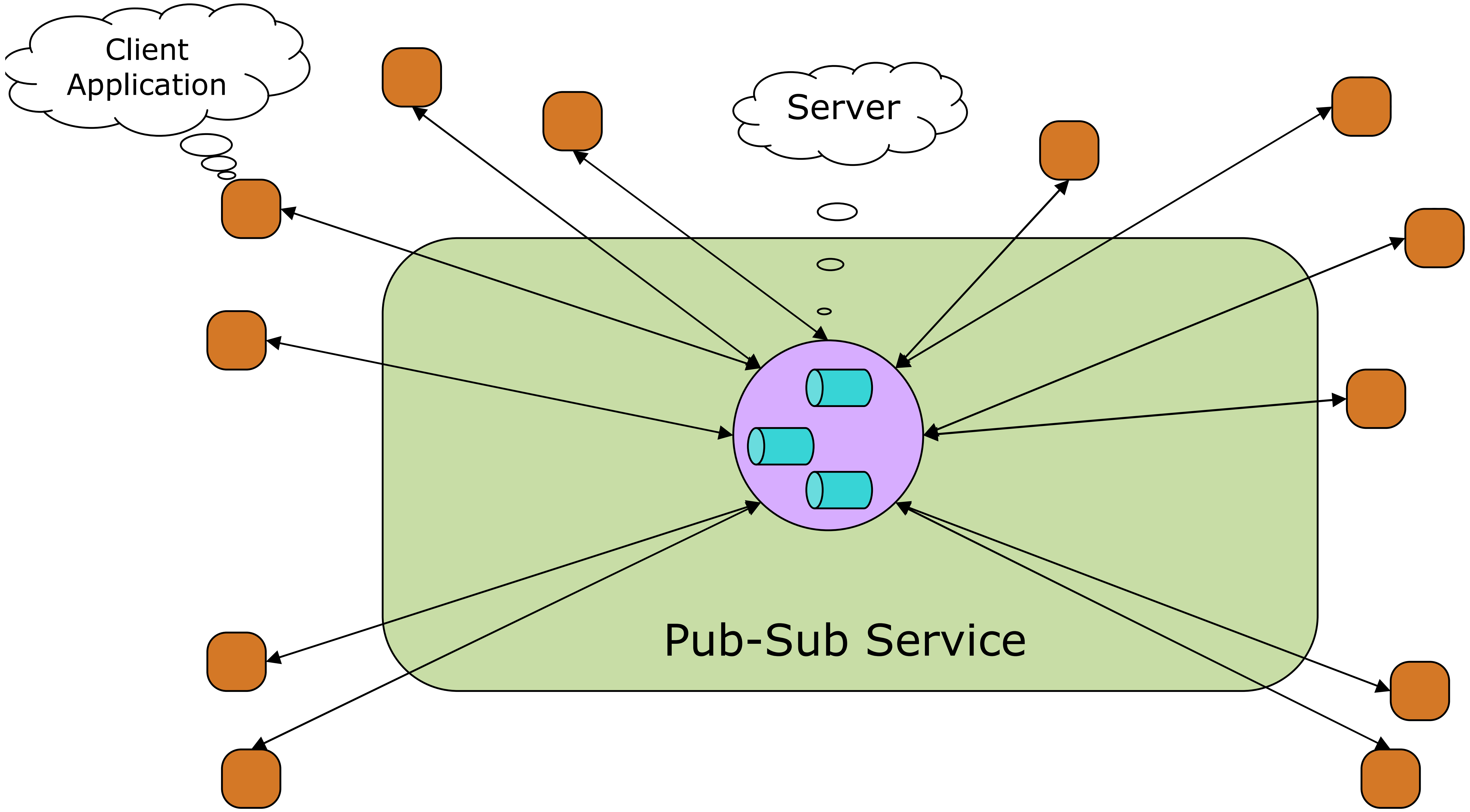
- Delegation = the concept of a middleman
- Brings security, reliability, guaranteed delivery, translation, …

Centralised broker pub-sub service, store and forward service

One central server materializes all middleware entities

All traffic flows via server

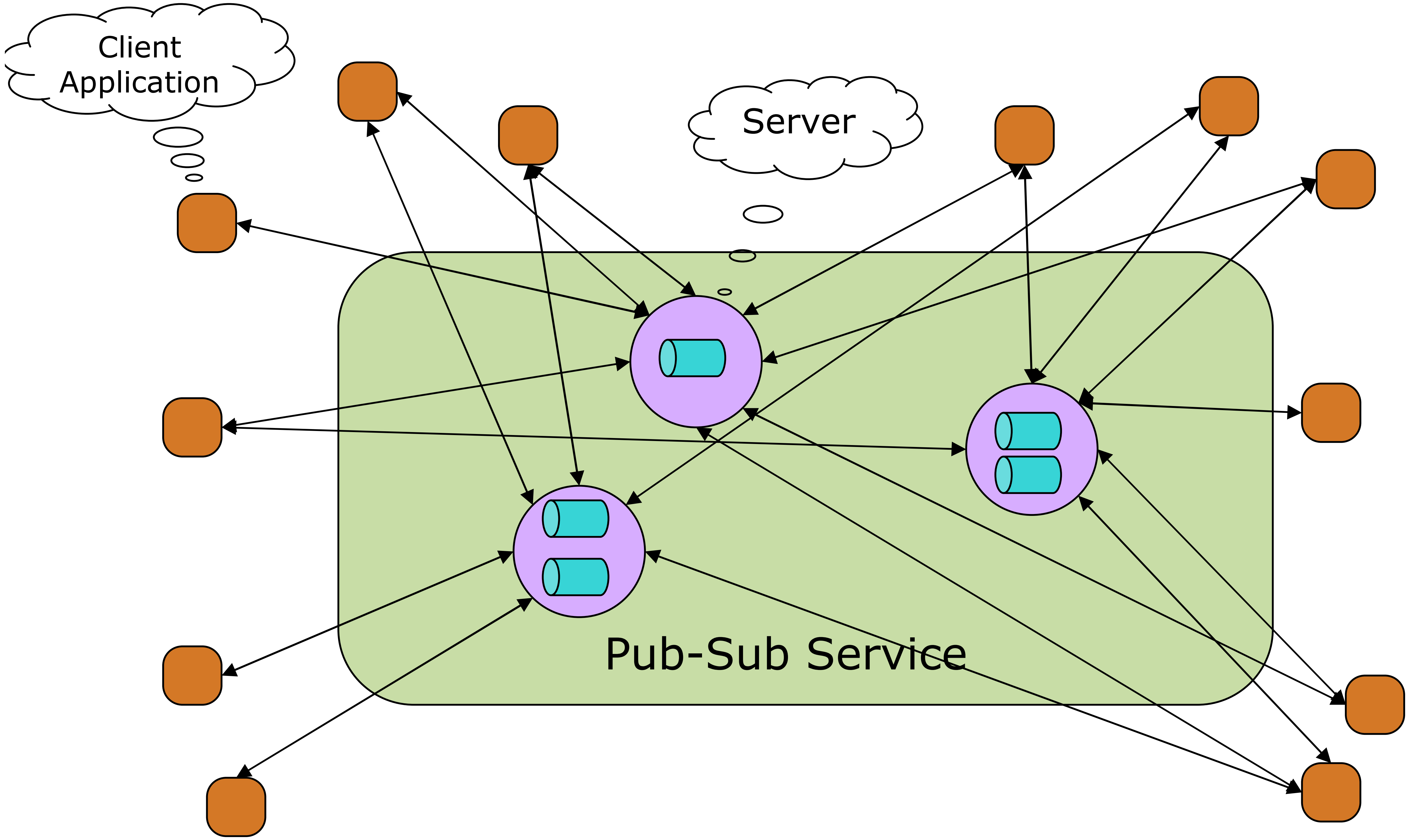
E.g. “naïve” implementations of JMS, CORBA Notification, etc.



Centralised multi-broker service

Each Queue/Topic Can be placed on a different Server

E.g. Better implementations of JMS, CORBA Notification, etc.



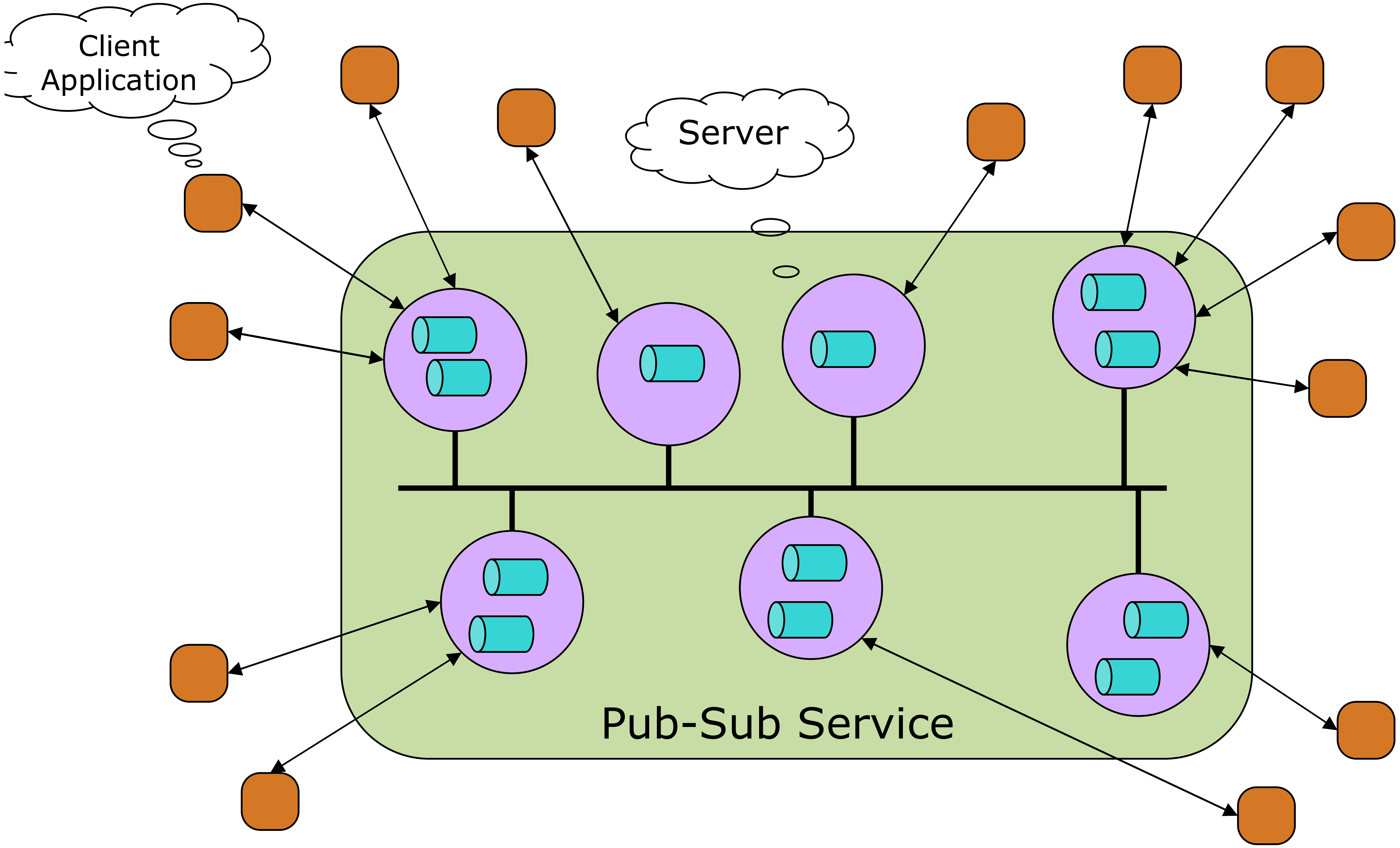
confidential

Decentralised multi-brokered service

App uses messaging or RMI to interact with Service Access points

Pub-Sub Service distributes messages internally between servers

Internally PS-Service can be peer-to-peer, hub-and-spoke, multicast

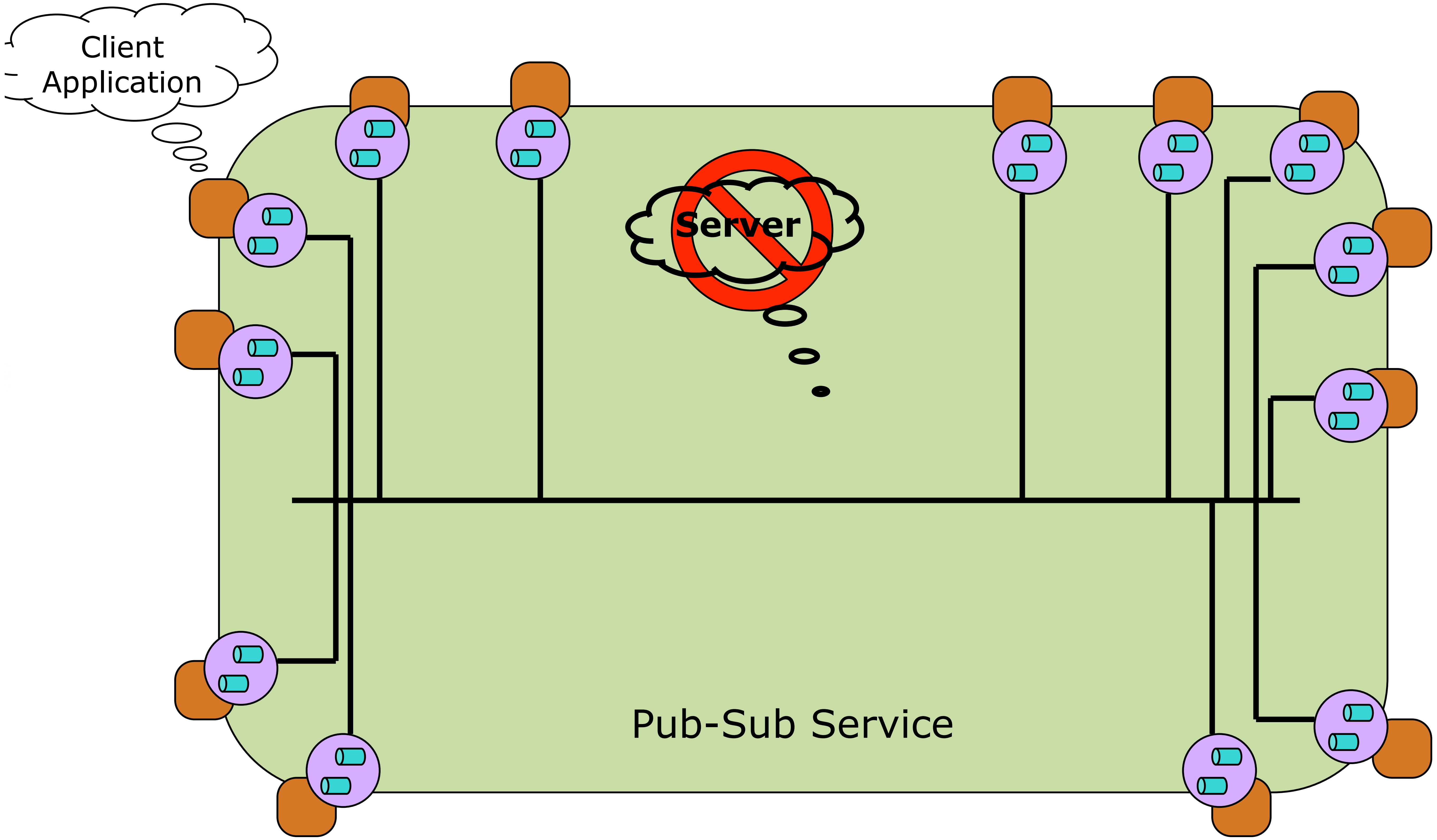


Decentralised ‘unbrokered’ - push multiple brokers to nodes

App links (binds) directly with the Pub-Sub service

Queuing occurs locally on each client

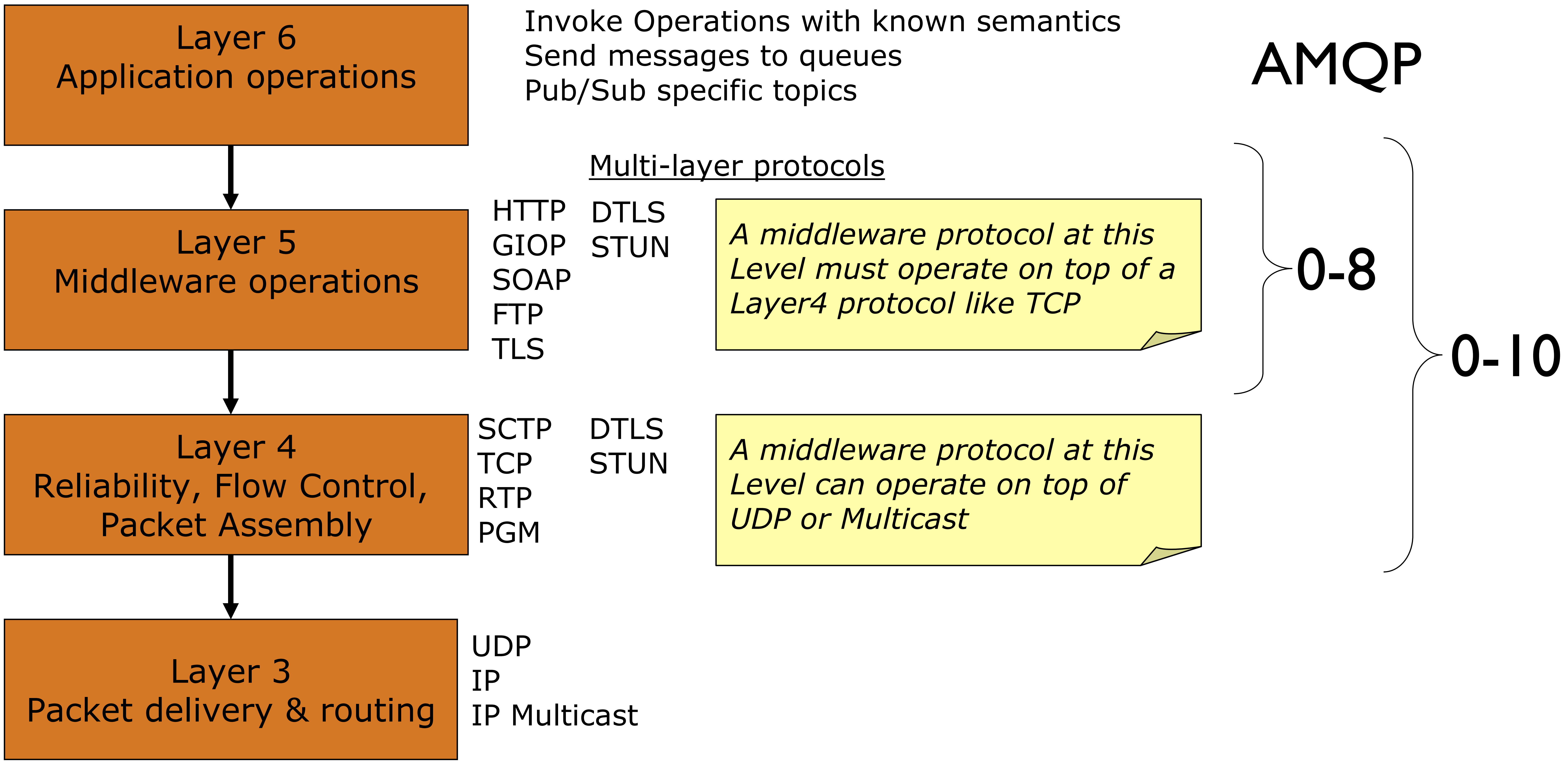
Clients communicate peer-to-peer



Service model architecture examples before AMQP

Model	*Examples*
Centralized *Brokered*	Typical JMS implementations
Centralized *Multi-Brokered*	Better JMS implementations CORBA event & notification service
De-centralized Brokered	TIBCO RendezVous TIBCO SmartSockets IBM WebSphere MQ (MQSeries) using client connection
De-centralized *Un-brokered* *(Peer-to-peer)*	Most DDS implementations: RTI DDS, OpenSplice, Tao-DDS IBM WebSphere MQ (MQSeries) using Binding connection

Middleware and protocol layering



Layer 6
Application operations

Invoke Operations with known semantics
Send messages to queues
Pub/Sub specific topics

AMQP

Multi-layer protocols

Layer 5
Middleware operations

HTTP
GIOP
SOAP
FTP
TLS

DTLS
STUN

A middleware protocol at this Level must operate on top of a Layer4 protocol like TCP

0-8

0-10

Layer 4
Reliability, Flow Control, Packet Assembly

SCTP
TCP
RTP
PGM

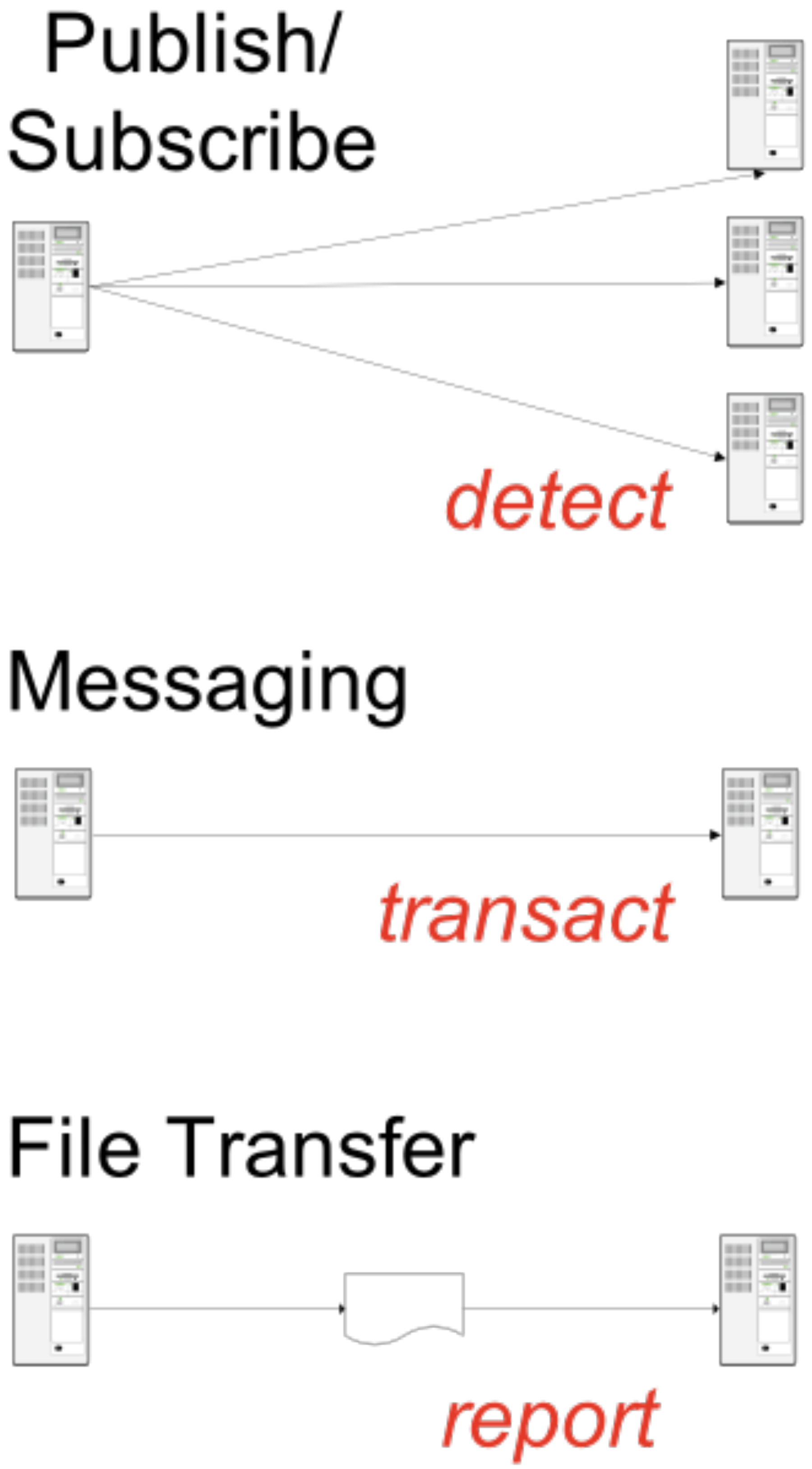
DTLS
STUN

A middleware protocol at this Level can operate on top of UDP or Multicast

Layer 3
Packet delivery & routing

UDP
IP
IP Multicast

Let’s break this down…



Publish/ Subscribe

detect

Messaging

transact

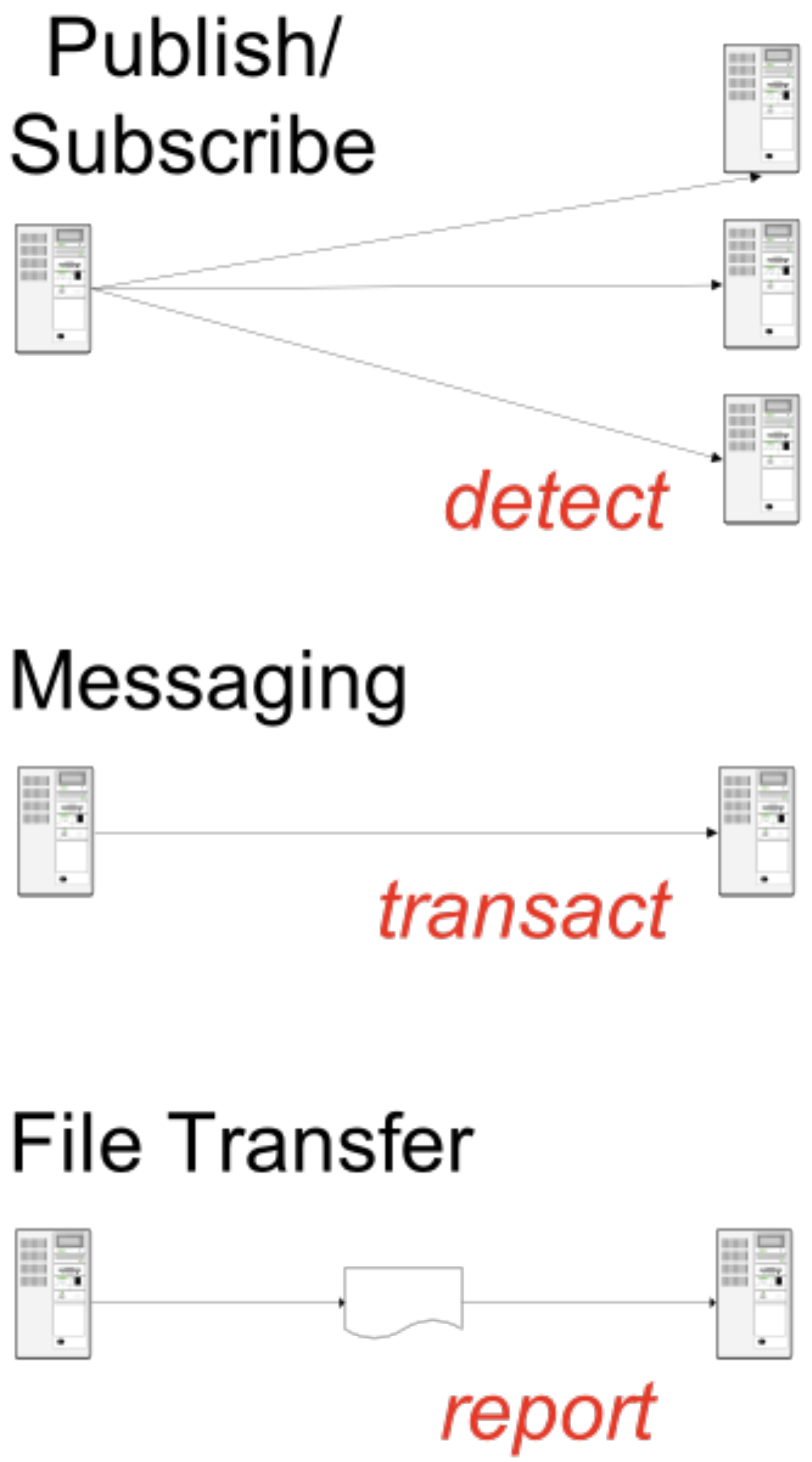
File Transfer

report

.. and put it back together ...

- Provide **event notification, messaging**, **file transfer**
  - Deals with business transaction processing
  - Technology agnostic (there is more than Java)
- Meet **real-world requirements**

  of **mission-critical** systems
- Be **Trustworthy**
  - Robust, available, scalable, secure, resilient
  - Stable - like TCP interoperating better than JMS
- Provide a **common infrastructure** for the enterprise



Publish/ Subscribe

detect

Messaging

transact

File Transfer

report

Into one protocol (to rule them all?)

Providing a **common infrastructure** for the enterprise

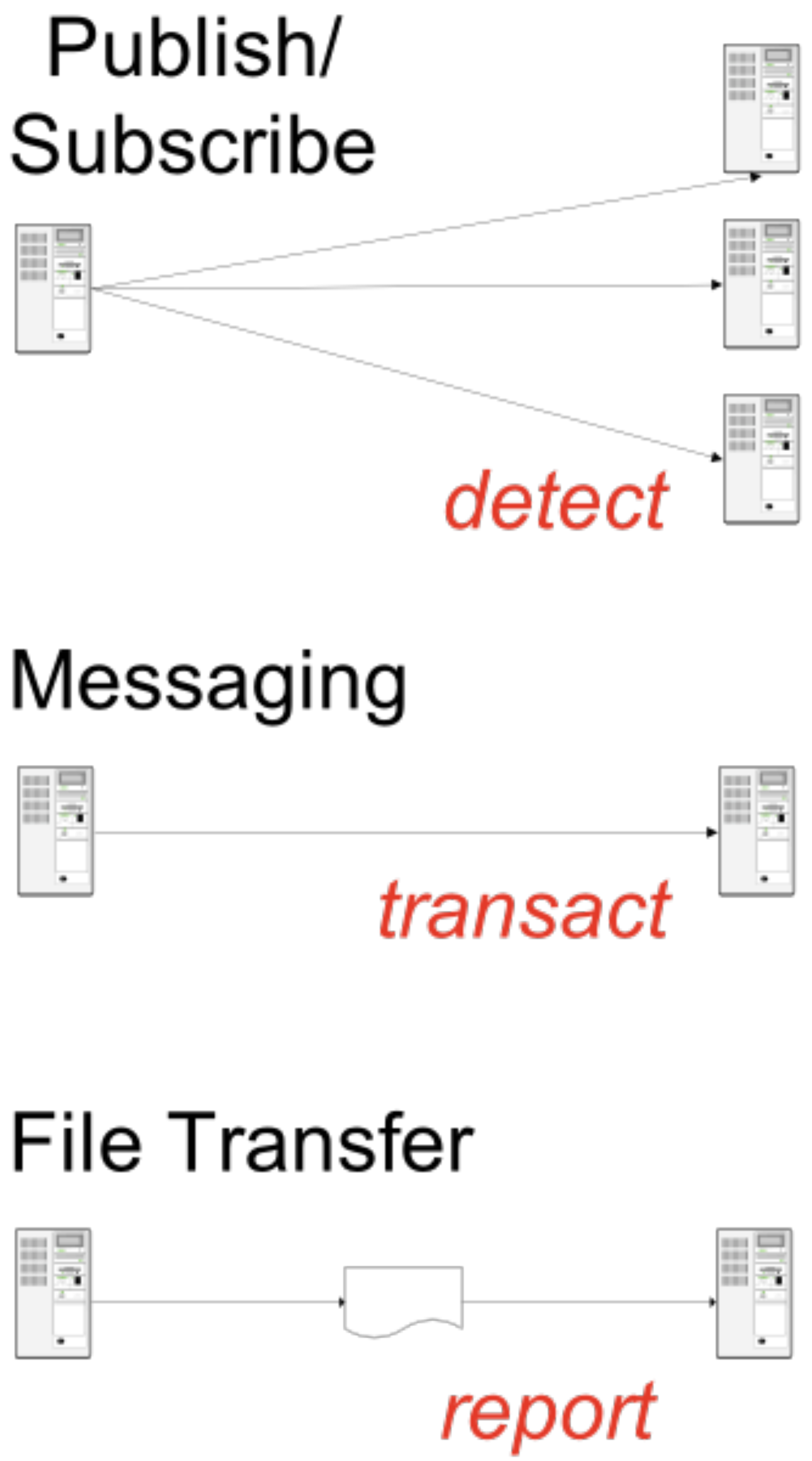
Covering the use cases we described and new ones too

Doing the difficult or ‘fiddly’ bits for you

Pub/sub, reliable messaging?

we now do all these in ONE protocol - AMQP

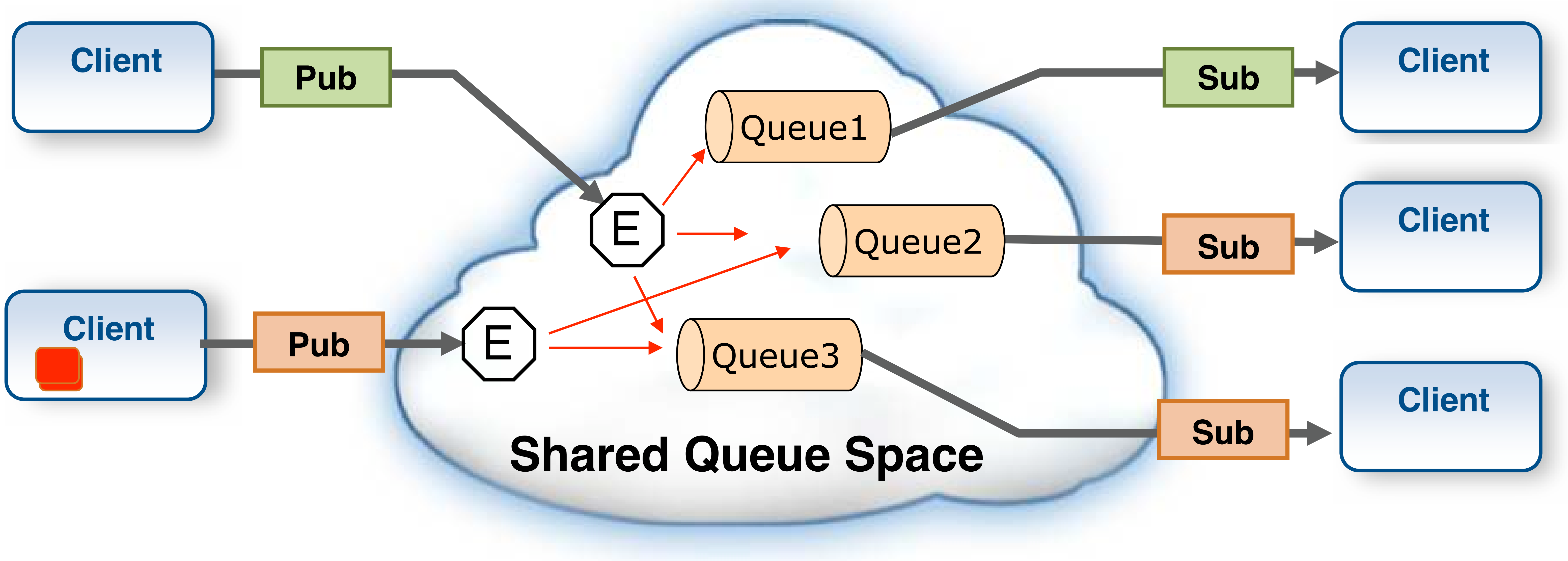
- Usually provided by 3 different proprietary products
- One solution reduces costs, increases efficiency and simplifies management



AMQP Communication model

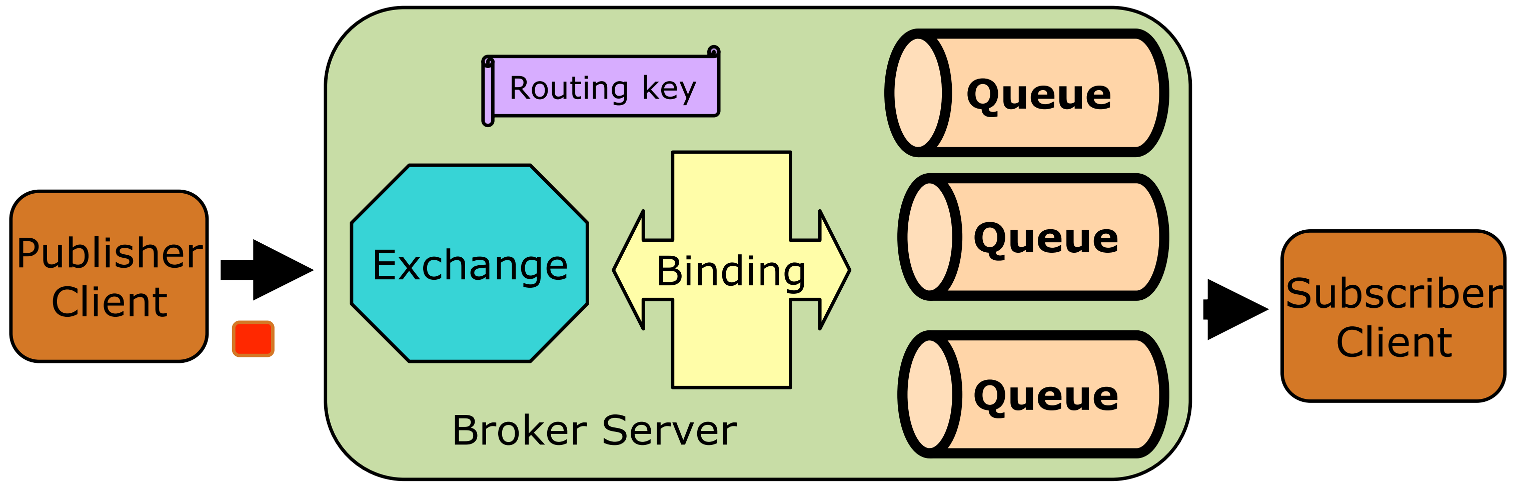
Provides a "**Shared Queue Space**" that is accessible to all interested applications.

- **Message** are sent to an **Exchange**
- Each message has an associated **Routing Key**
- **Brokers** forward messages to one or more **Queues** based on the **Routing Key**
- Subscriber get messages from **named Queues**
- Only **one subscriber** can get a given message from each **Queue**



Queues are smart buffers

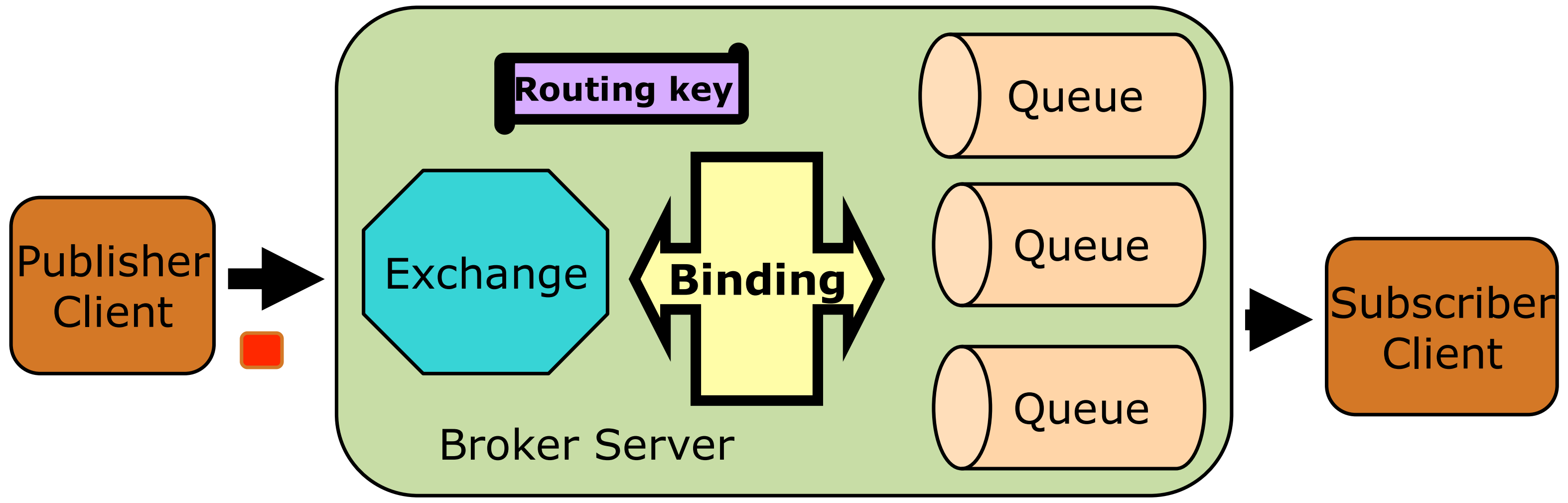
- Stores and distributes messages
- Each message delivered to a single client consumer
- Properties (on creation):
  - Name
    - Client Named or Server Named
  - Durable
    - Durable remains present after re-start
      - But may lose non-persistent messages
  - Auto-delete
    - Will auto-delete when all clients have finished using it
  - Private (Exclusive)/Shared
    - Private (Exclusive) ⇔ read by a single consumer



Exchanges do routing based on keys

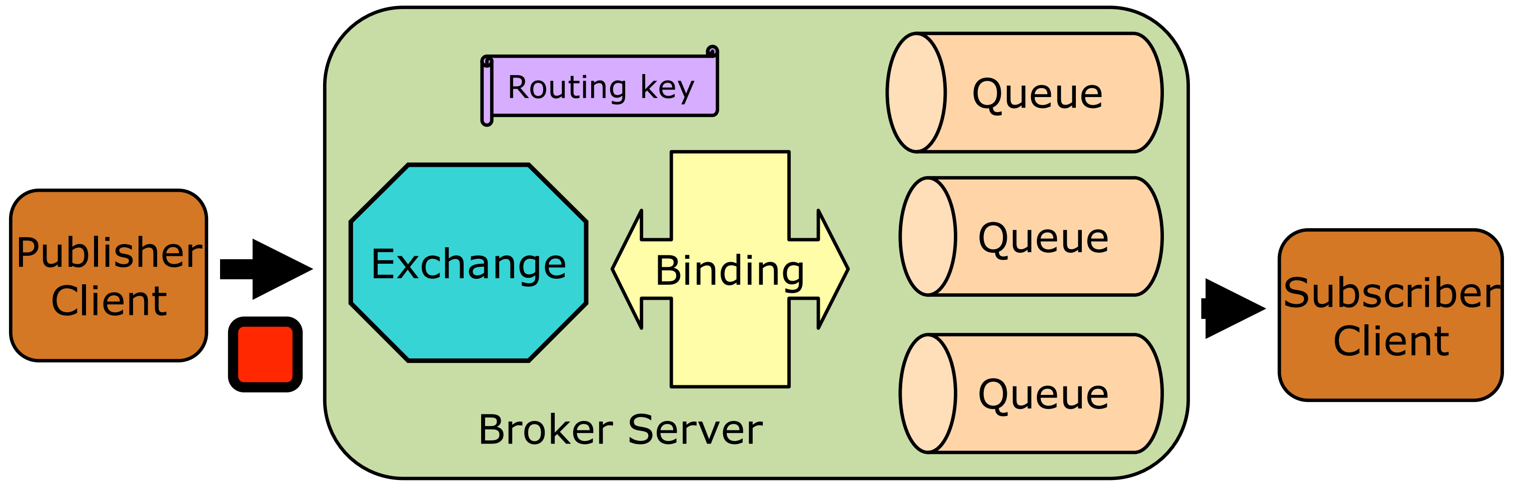
- Binding
  - Tells exchange how to route messages:

    Queue.Bind <queue> TO <exchange> WHERE <condition>
- The **<condition>** can involve:
  - Message Properties
  - Header fields
  - Content
  - In most cases uses a single field: the “routing key”
- Routing key = virtual address used in the binding
  - For Point2point routing-key = name of msgQ
  - For Topic PubSub routing-key = topic hierarchy value
  - In other cases routing-key may be combined with msg header and content

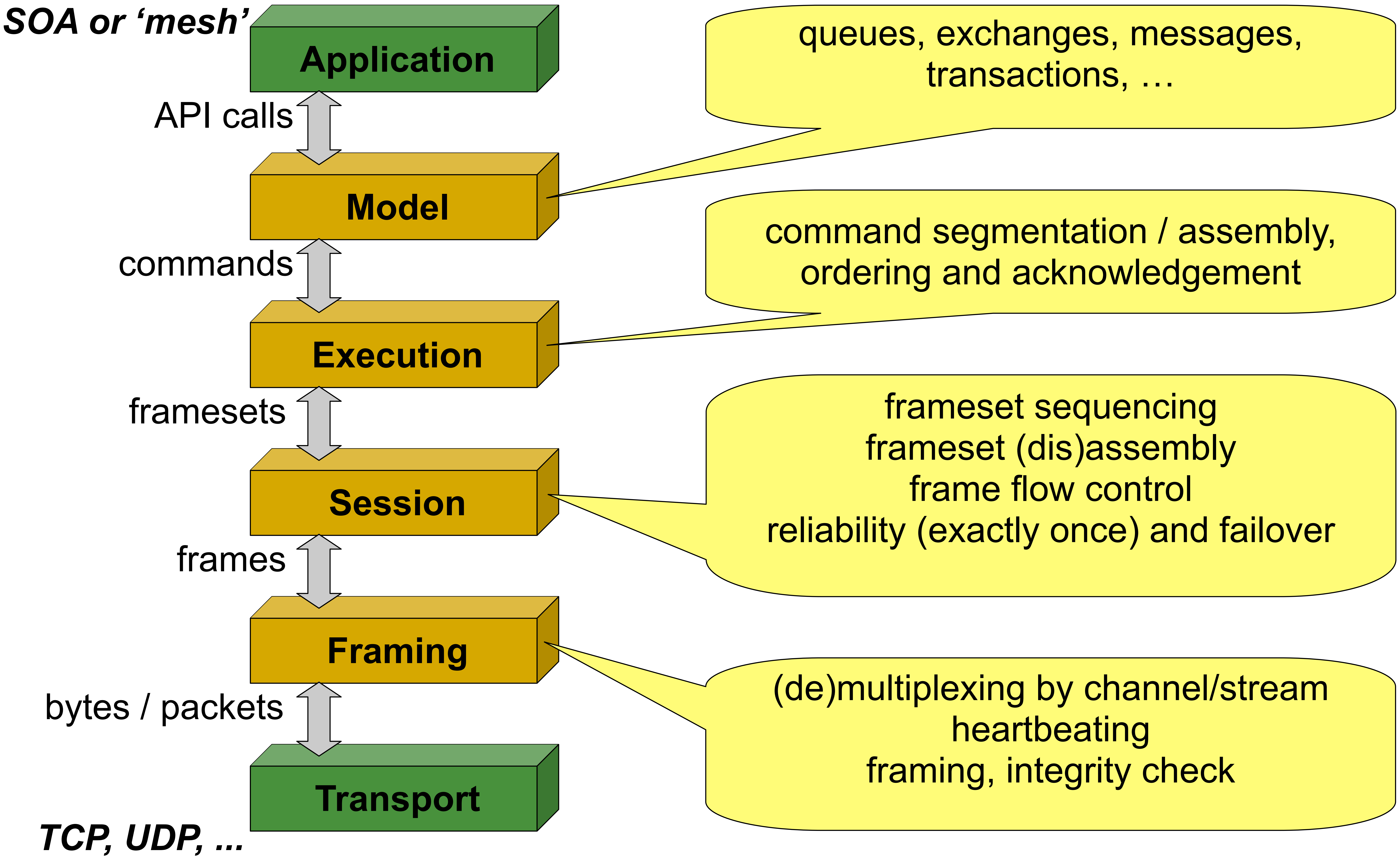


From soup to nuts ....

- Client producer creates message
- Producer fill content, properties and routing information
- Producer sends msg to Exchange
- Exchange route msg to set of Queues. Each is treated as a separate copy (no common identifier)
- Queue passes message to a single consumer if present or else buffers it.
  - Upon ‘delivery’ msg removed from queue.
- 2 kind of acks: Automatic or Explicit
  - Explicit requires app to indicate so for each message



Layered stack gives transport and model independence



SOA or ‘mesh’

Application

API calls

Model

queues, exchanges, messages, transactions, …

commands

Execution

command segmentation / assembly, ordering and acknowledgement

framesets

Session

frameset sequencing
frameset (dis)assembly
frame flow control
reliability (exactly once) and failover

frames

Framing

(de)multiplexing by channel/stream
heartbeating
framing, integrity check

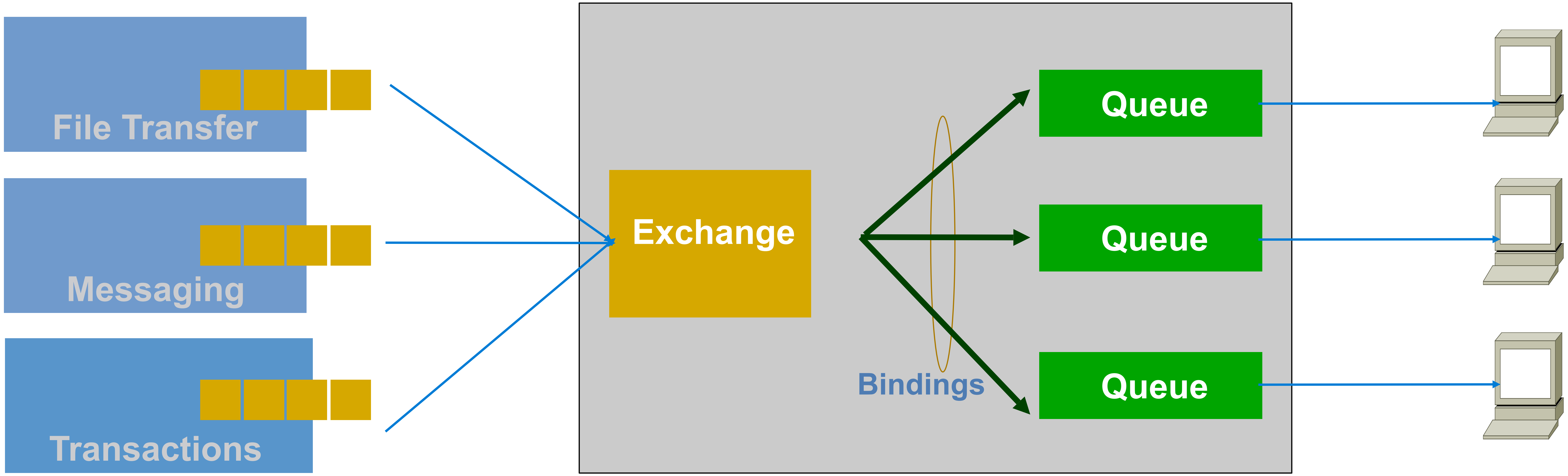
bytes / packets

Transport

TCP, UDP, ...

al

“Wouldn’t it be great if we had a GOOD messaging solution”



- any language *(C, C++, C#, Python, Java, Javascript, Erlang, Lisp, Ruby, Tcl, PHP, …)*
- any model *(native, .NET WCF, JMS, Mule, can do Caching)*
- any payload *(binary, XML, SOAP, JSON, …)*
- any transport *(TCP, SCTP, UDP, HTTP, …)*
- any scenario *(desktop, router, wan, mobile, mesh, cloud, …)*

- reliable
- interoperable
- manageable
- performant
- scaleable

SUMMARY - another win in the battle vs ‘more of everything’

- A general solution, learning from and improving messaging, to deliver a way to scale across multiple integration and SOA scenarios with a single protocol

- VERY simple to use - complexity is hidden in the protocol

- A straight-forward and complete solution for business messaging
  SMTP/TCP for business --> cost effective for pervasive deployment
  Lightweight - interoperates with anything, eg over WAN or ‘cloud’

- Does the hard messaging stuff but as a reliable high speed fabric
  Introduces reliable intermediation at lowest possible level - wire binary
  Can be invisible and fast! eg in hardware

- Does new stuff that people want - routing, wide area, ...

Open Standards - HOWTO

An open specification

Unlike Tibco and MQ, the specification is completely open

Anyone can implement it - all of it, or some of it

Users and vendors working together on real needs since 2006

History:

- 0-8 - routing and reliability (availability)
- 0-10 - guaranteed delivery and transport independence
- Service packs - security, management, addressing
- 1.0 - release

ONE BILLION MESSAGES PER DAY - NOW

AMQP Working Group - end users and vendors

JPMorgan

Deutsche Boerse

Credit Suisse and Goldman Sachs

Cisco and Red Hat and Novell

CohesiveFT - LShift - RabbitMQ

and others from WS-* world too

AMQP Working Group shares your goals

Reduce systems integration costs

- Incumbent vendor charges are high
  … for little business value add
- Messaging and integration account for 10-30% of IT costs
- AMQP will boost competition and accelerate commoditization among solution providers

AMQP Working Group shares your goals

Compete on value add

Eliminate vendor lock-in

- Lack of interoperability is a friction cost on your business
- AMQP will let you switch suppliers and spur competition
- UNLIKE - WebsphereMQ vs Tibco EMS
- Value add - eg RabbitMQ for .NET, or eg file streaming

AMQP Working Group shares your goals

“Messaging everywhere”

→ Remove barriers to a liquid services market to grow it

- Not interested in ‘rip&replace’ - extend existing investments
- Working to integrate with FIX, FpML in financial services
- Envisage SWIFT-like reliability with the openness of the Web…

AMQP aims to become..

THE standard for business messaging

Using AMQP

Messaging is not just about interoperability

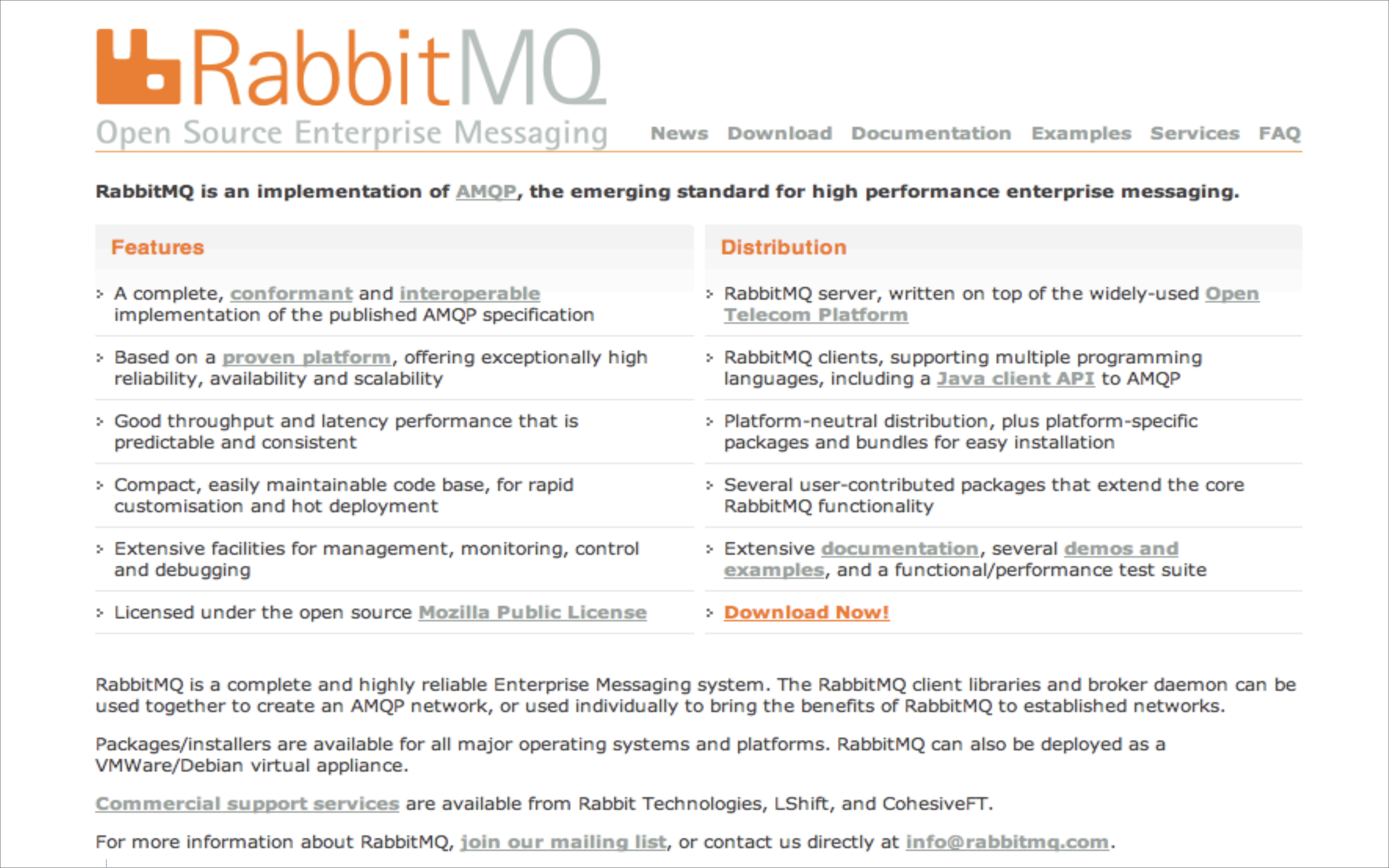
You want it to be …

- Reliable
- Scalable
- Distributed
- Maintainable
- Highly Available
- Long-Lived
- Portable
- Pluggable

- Manageable
- Certified interoperable
- Pre-integrated with many tools
- Handle diverse, variable workloads
- Secure
- ....

http://www.rabbitmq.com

http://es.cohesiveft.com/site/rabbitmq



RabbitMQ

Open Source Enterprise Messaging

News Download Documentation Examples Services FAQ

RabbitMQ is an implementation of AMQP, the emerging standard for high performance enterprise messaging.

Features

- A complete, conformant and interoperable implementation of the published AMQP specification
- Based on a proven platform, offering exceptionally high reliability, availability and scalability
- Good throughput and latency performance that is predictable and consistent
- Compact, easily maintainable code base, for rapid customisation and hot deployment
- Extensive facilities for management, monitoring, control and debugging
- Licensed under the open source Mozilla Public License

Distribution

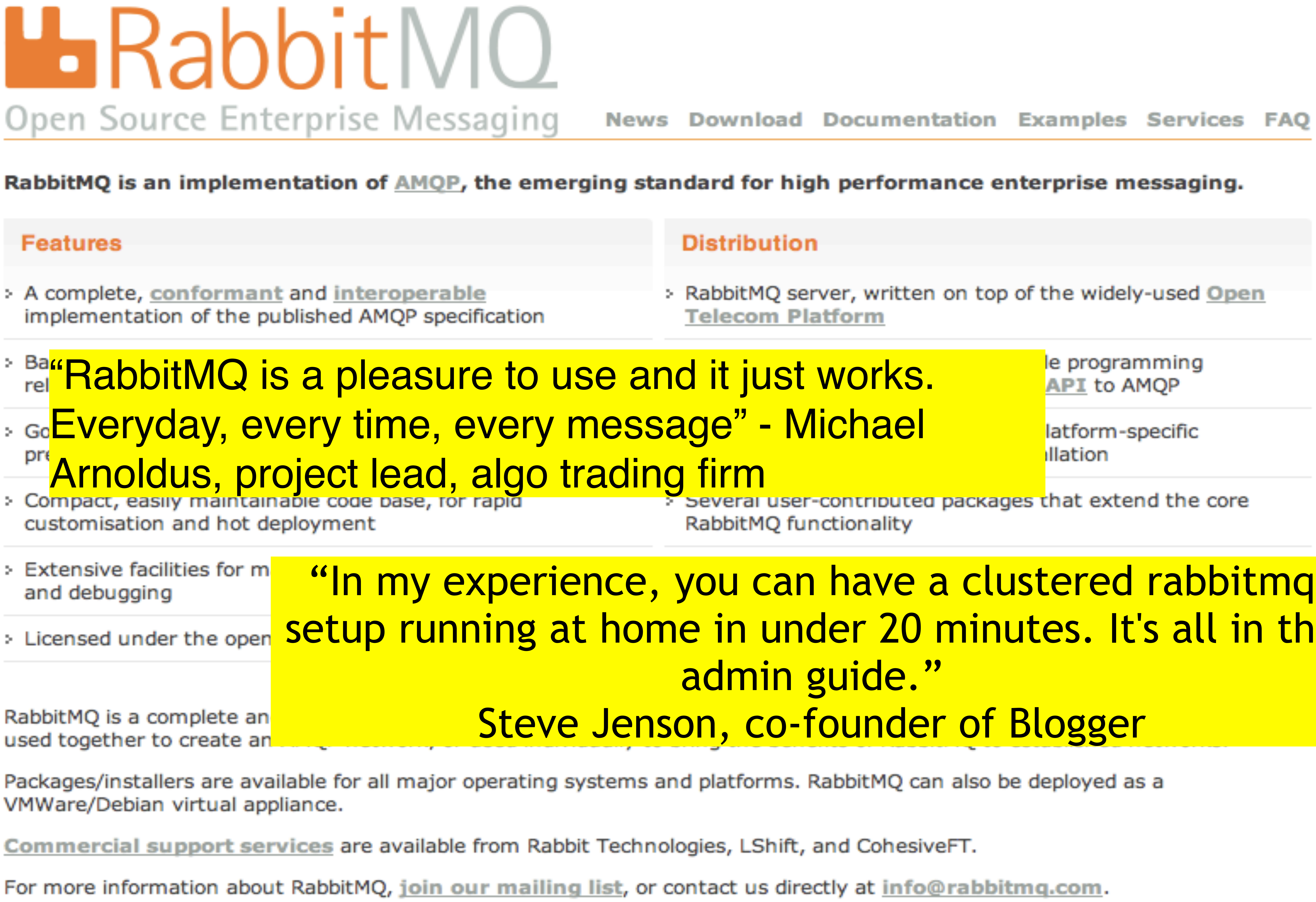
- RabbitMQ server, written on top of the widely-used Open Telecom Platform
- RabbitMQ clients, supporting multiple programming languages, including a Java client API to AMQP
- Platform-neutral distribution, plus platform-specific packages and bundles for easy installation
- Several user-contributed packages that extend the core RabbitMQ functionality
- Extensive documentation, several demos and examples, and a functional/performance test suite
- Download Now!

RabbitMQ is a complete and highly reliable Enterprise Messaging system. The RabbitMQ client libraries and broker daemon can be used together to create an AMQP network, or used individually to bring the benefits of RabbitMQ to established networks.

Packages/installers are available for all major operating systems and platforms. RabbitMQ can also be deployed as a VMWare/Debian virtual appliance.

Commercial support services are available from Rabbit Technologies, LShift, and CohesiveFT.

For more information about RabbitMQ, join our mailing list, or contact us directly at info@rabbitmq.com.



RabbitMQ

Open Source Enterprise Messaging

RabbitMQ is an implementation of AMQP, the emerging standard for high performance enterprise messaging.

Features

- A complete, conformant and interoperable implementation of the published AMQP specification
- Ba rel
- Go pr
- Compact, easily maintainable code base, for rapid customisation and hot deployment
- Extensive facilities for m and debugging
- Licensed under the open

Distribution

- RabbitMQ server, written on top of the widely-used Open Telecom Platform
- le programming API to AMQP
- latform-specific llation
- Several user-contributed packages that extend the core RabbitMQ functionality

RabbitMQ is a complete an used together to create an

Packages/installers are available for all major operating systems and platforms. RabbitMQ can also be deployed as a VMWare/Debian virtual appliance.

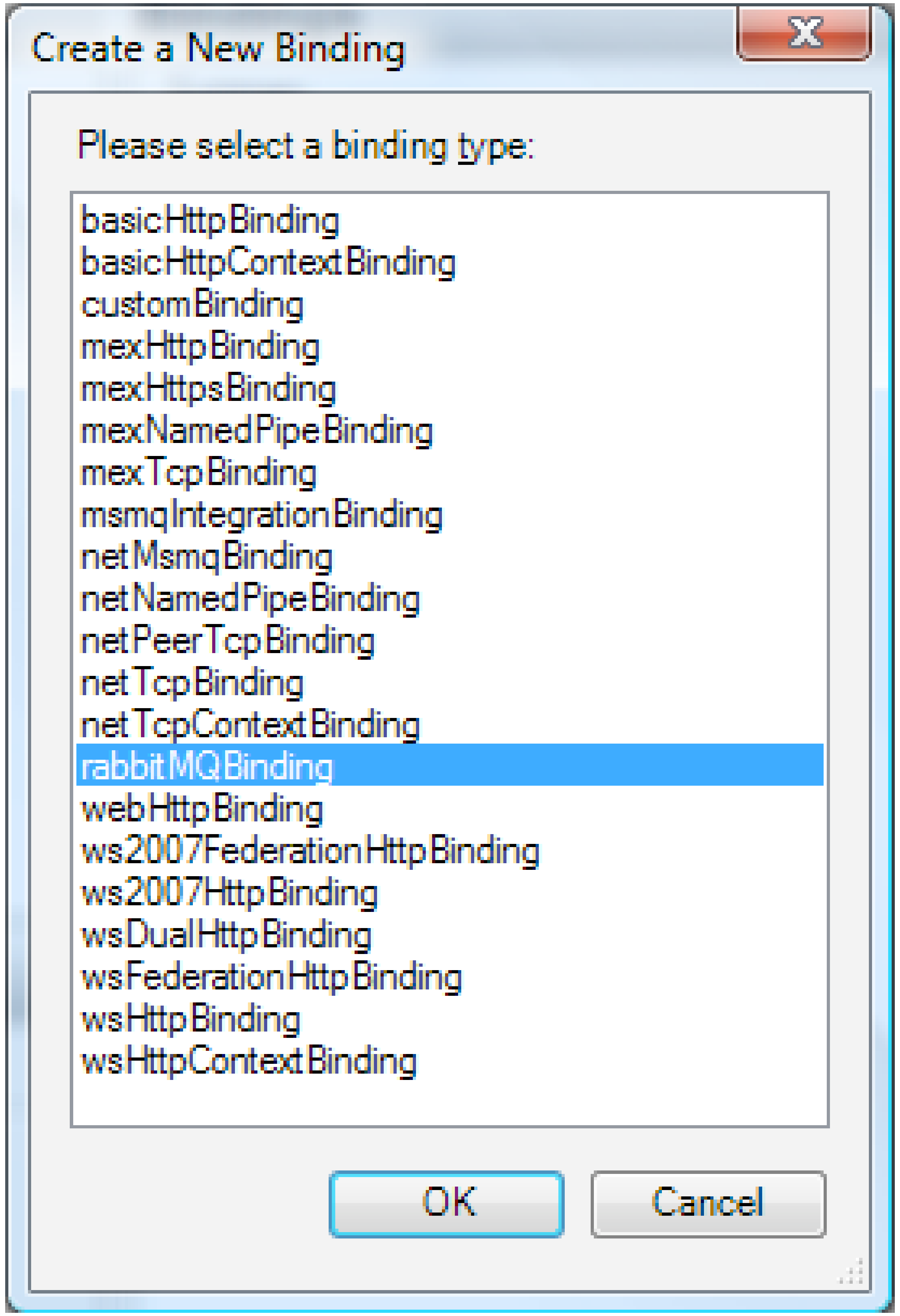
Commercial support services are available from Rabbit Technologies, LShift, and CohesiveFT.

For more information about RabbitMQ, join our mailing list, or contact us directly at info@rabbitmq.com.

"RabbitMQ is a pleasure to use and it just works. Everyday, every time, every message" - Michael Arnoldus, project lead, algo trading firm

"In my experience, you can have a clustered rabbitmq setup running at home in under 20 minutes. It's all in the admin guide."
Steve Jenson, co-founder of Blogger

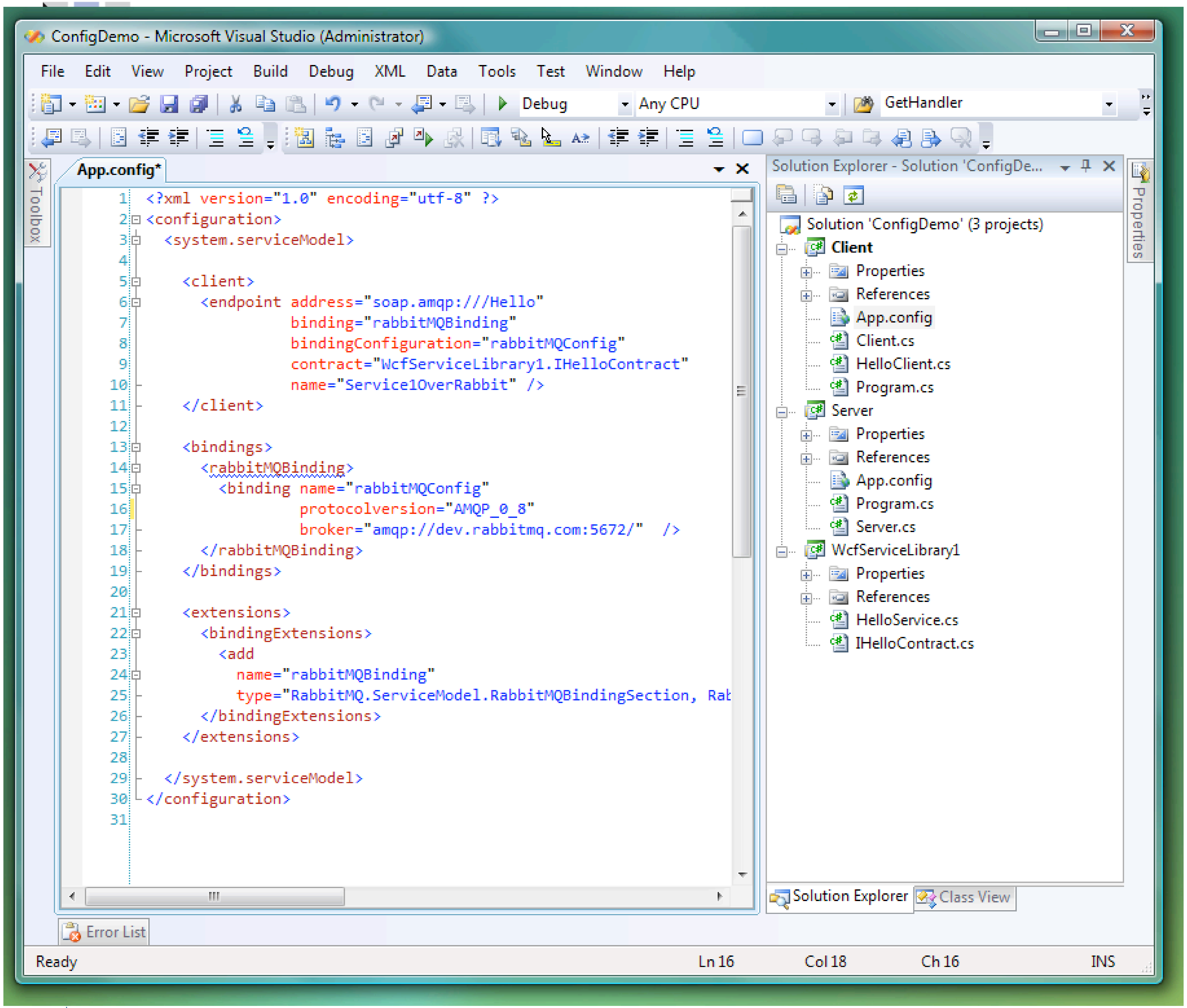
In Visual Studio, RabbitMQ is just another .NET transport



Create a New Binding

Please select a binding type:

basicHttpBinding
basicHttpContextBinding
customBinding
mexHttpBinding
mexHttpsBinding
mexNamedPipeBinding
mexTcpBinding
msmqIntegrationBinding
netMsmqBinding
netNamedPipeBinding
netPeerTcpBinding
netTcpBinding
netTcpContextBinding
rabbitMQBinding
webHttpBinding
ws2007FederationHttpBinding
ws2007HttpBinding
wsDualHttpBinding
wsFederationHttpBinding
wsHttpBinding
wsHttpContextBinding



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>

    <client>
      <endpoint address="soap.amqp:///Hello"
                binding="rabbitMQBinding"
                bindingConfiguration="rabbitMQConfig"
                contract="WcfServiceLibrary1.IHelloContract"
                name="Service1OverRabbit" />
    </client>

    <bindings>
      <rabbitMQBinding>
        <binding name="rabbitMQConfig"
                 protocolversion="AMQP_0_8"
                 broker="amqp://dev.rabbitmq.com:5672/"  />
      </rabbitMQBinding>
    </bindings>

    <extensions>
      <bindingExtensions>
        <add
          name="rabbitMQBinding"
          type="RabbitMQ.ServiceModel.RabbitMQBindingSection, Rab
      </bindingExtensions>
    </extensions>

  </system.serviceModel>
</configuration>

```

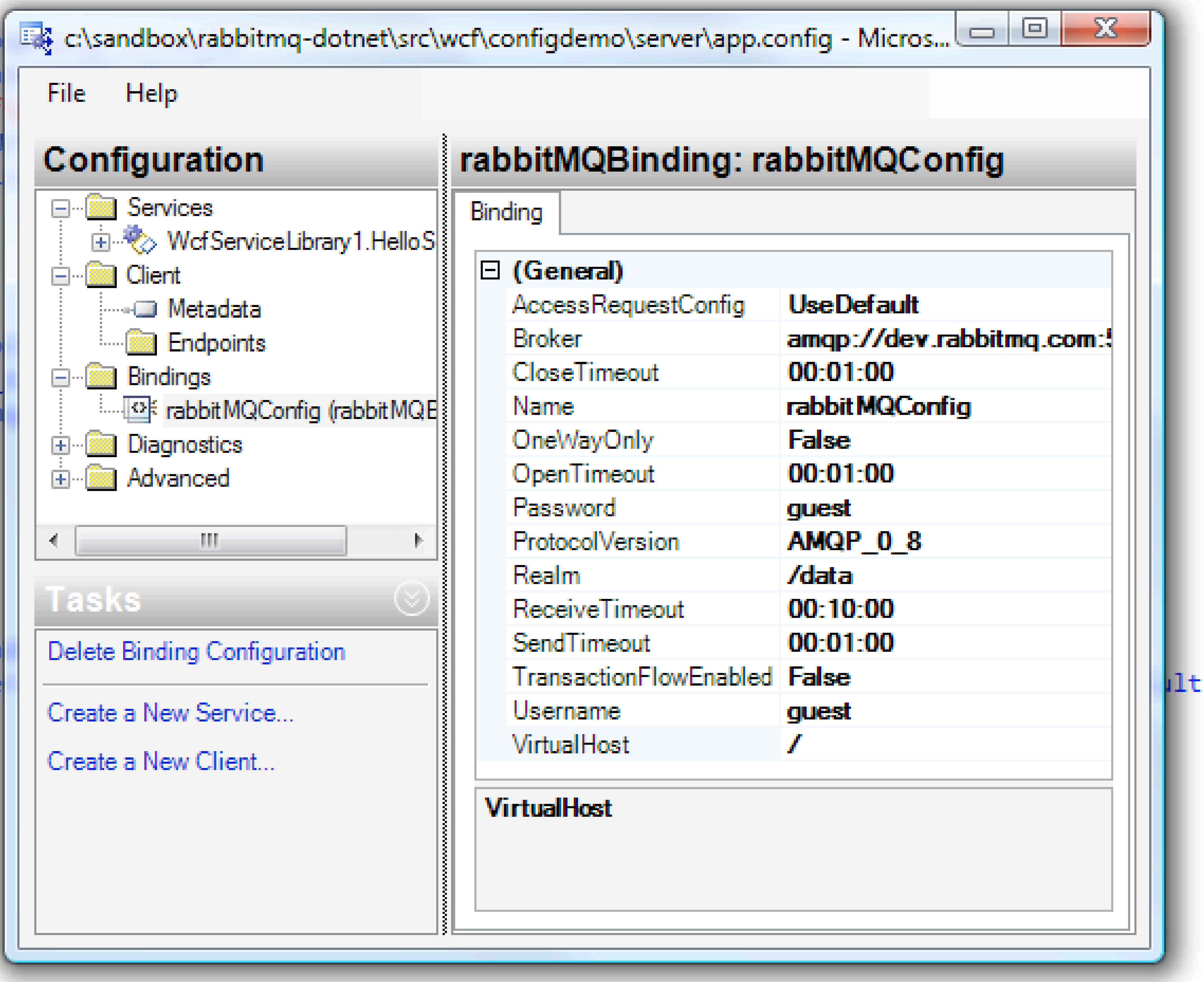
In Visual Studio the bindings and metadata are completely intuitive for developers to manage

```
<client>
  <endpoint address="so
            binding="ra
            bindingConf
            contract="W
            name="Servi
</client>

<bindings>
  <rabbitMQBinding>
    <binding name="rabb
  protocolversion="AMQP_
            broker="am
  </rabbitMQBinding>
</bindings>

<extensions>
  <bindingExtensions>
    <add
      name="rabbitMQBin
      type="RabbitMQ.Se                                      ltur
  </bindingExtensions>
</extensions>

</system.serviceModel>
:onfiguration>
```



c:\sandbox\rabbitmq-dotnet\src\wcf\configdemo\server\app.config - Micros...

Configuration

- Services
  - WcfServiceLibrary1.HelloS
- Client
  - Metadata
  - Endpoints
- Bindings
  - rabbitMQConfig (rabbitMQB
- Diagnostics
- Advanced

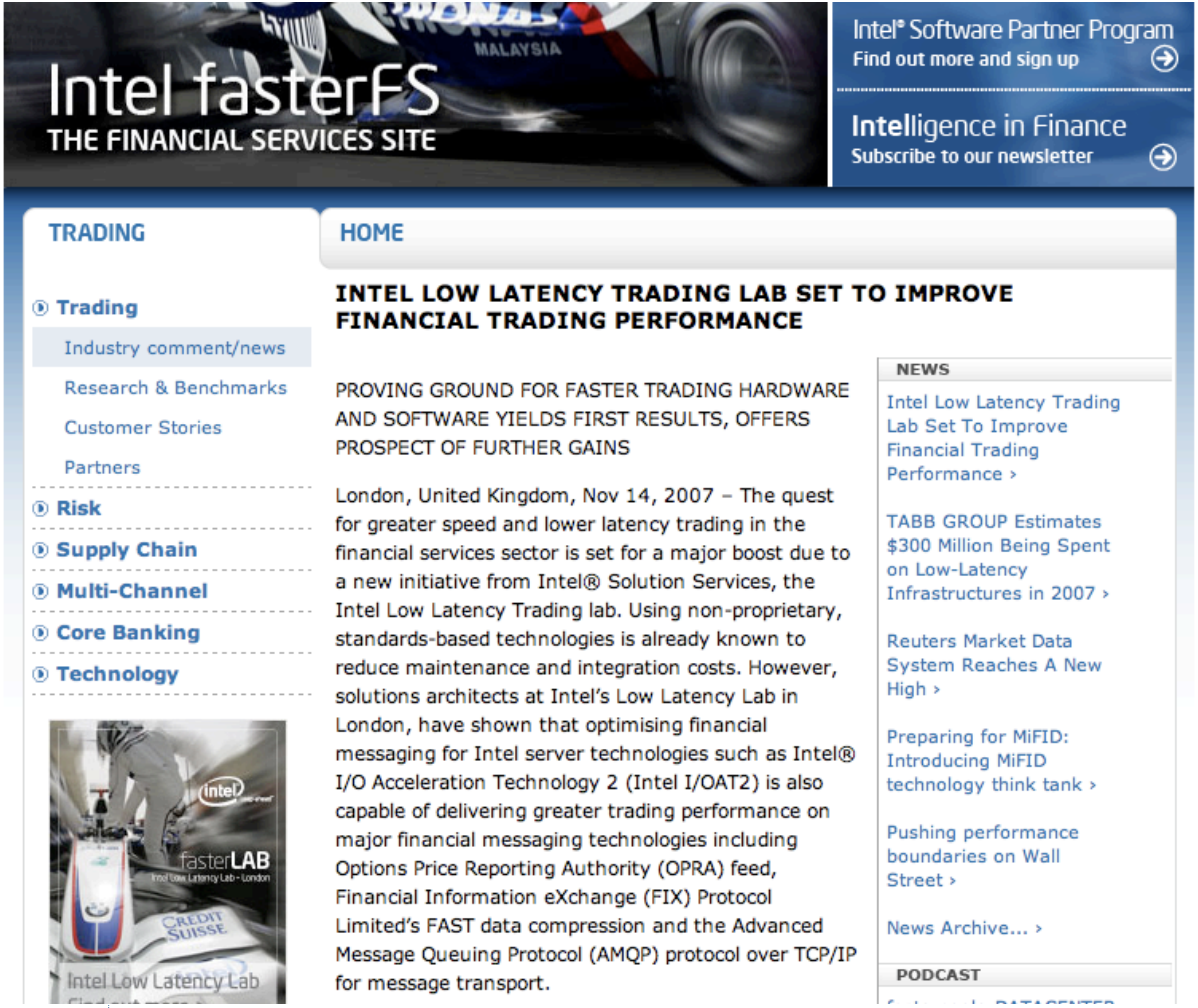
Tasks

rabbitMQBinding: rabbitMQConfig

(General)	
AccessRequestConfig	UseDefault
Broker	amqp://dev.rabbitmq.com:5
CloseTimeout	00:01:00
Name	rabbitMQConfig
OneWayOnly	False
OpenTimeout	00:01:00
Password	guest
ProtocolVersion	AMQP_0_8
Realm	/data
ReceiveTimeout	00:10:00
SendTimeout	00:01:00
TransactionFlowEnabled	False
Username	guest
VirtualHost	/

VirtualHost

Doing OPRA feed on a $5k box, by working with open standards and INTEL



INTEL LOW LATENCY TRADING LAB SET TO IMPROVE FINANCIAL TRADING PERFORMANCE

PROVING GROUND FOR FASTER TRADING HARDWARE AND SOFTWARE YIELDS FIRST RESULTS, OFFERS PROSPECT OF FURTHER GAINS

London, United Kingdom, Nov 14, 2007 – The quest for greater speed and lower latency trading in the financial services sector is set for a major boost due to a new initiative from Intel® Solution Services, the Intel Low Latency Trading lab. Using non-proprietary, standards-based technologies is already known to reduce maintenance and integration costs. However, solutions architects at Intel's Low Latency Lab in London, have shown that optimising financial messaging for Intel server technologies such as Intel® I/O Acceleration Technology 2 (Intel I/OAT2) is also capable of delivering greater trading performance on major financial messaging technologies including Options Price Reporting Authority (OPRA) feed, Financial Information eXchange (FIX) Protocol Limited's FAST data compression and the Advanced Message Queuing Protocol (AMQP) protocol over TCP/IP for message transport.

NEWS
Intel Low Latency Trading Lab Set To Improve Financial Trading Performance ›
TABB GROUP Estimates $300 Million Being Spent on Low-Latency Infrastructures in 2007 ›
Reuters Market Data System Reaches A New High ›
Preparing for MiFID: Introducing MiFID technology think tank ›
Pushing performance boundaries on Wall Street ›
News Archive... ›
PODCAST

Meez.com / AOL - messaging in the cloud

6 million users growing fast - 52 EC2 AMIs

Has never gone down

“I think that RabbitMQ has been doing

a bang up job over here”

Overview of Miyu framework

- Written in Ruby to be used from Ruby or shell
- Uses RabbitMQ for all messaging
- Developers are encouraged to instrument alarms in their code (push) instead of poll-based monitoring
- Never blocks as long as local disk is available
- Does not require dedicated monitoring servers
- Allows selective monitoring of a subset of events
- Decouples presentation from logic
- Agent infrastructure (like all similar products)

Questions?