



Integration technologies and products for OpenVMS

Maybe you can teach an old
dog new tricks

Brett Cameron, presented by Gerrit Woertman

© 2011 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice



Agenda



- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ (and ZeroMQ); Erlang
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

Abstract



This talk will examine some of the technologies and products that are available for integrating OpenVMS applications with other applications on other platforms. The topics will range from data integration at the file and database level through to the exposure of existing software components as services, be it real-time or queued.

Some of the technologies and products to be examined will include:

- Change Data Capture from RMS files, JDBC, and ODBC from Attunity
- Queuing as an integration medium using RabbitMQ (and ZeroMQ); ERLANG
- Calling and exposing Web Services from and on OpenVMS using gSOAP

The advent of the internet and the web have influenced the need for very high-speed and reliable queuing systems to a very large degree, be it for social networking sites, financial systems, or the reliable distribution of data in any environment. A key aspect of this talk will be to introduce the audience to message queuing in general and to focus on emerging standards such as AMQP (Advanced Message Queuing Protocol). Discussion will include details of the implementations of said standards that are available for OpenVMS.

The integration *problem* (1)



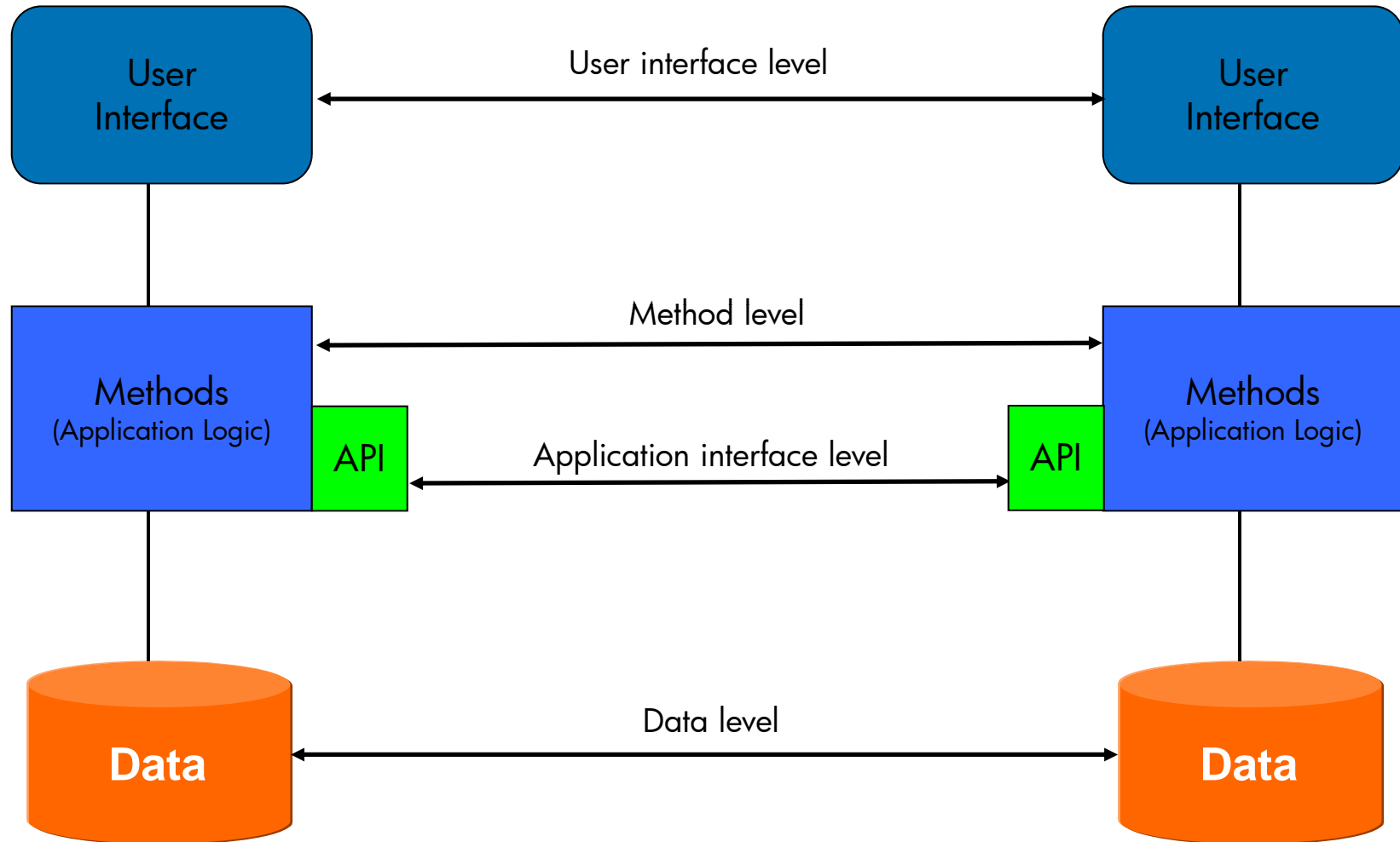
- Robust, reliable, secure, scalable, and efficient integration...
 - A critical success factor for many organizations
- Well done integration facilitates:
 - Agility
 - Flexibility
 - Better return on investment
 - Reuse of applications and code
 - Investment protection
 - Cost savings
- Integration drivers include:
 - Business automation
 - Efficiency and cost saving
 - E-commerce
 - Internet and web-enablement
 - Business-to-business (B2B)
 - Mergers and acquisitions
 - Outsourcing

The integration *problem* (2)



- Today's typical IT environment...
 - Multiple everything
 - Multiple platforms and databases
 - Multiple, disparate applications
 - Custom
 - Legacy
 - Packaged
 - Multiple transaction processors
 - Multiple data entry points
 - Multiple versions of the same data
 - Incompatible business data
- Complications...
 - Business/mission critical “legacy” applications are:
 - Implemented on a quagmire of incompatible architectures
 - Difficult to maintain
 - Nearly impossible to eliminate, at least in the short term
 - Application systems are:
 - Implemented by different groups/departments
 - At different times
 - Acting independently

Types of integration solutions



Data level integration



- Integration by directly replicating and/or manipulating the data
- Benefits:
 - Relatively inexpensive
 - Proven
 - Quick
 - Technology is available
 - Minimal risk
- Possible issues:
 - Sometimes more of a stopgap measure than a real solution?
 - Scalability needs to be very carefully considered
 - Could mask important issues with the information systems
- Typical enabling technologies:
 - Middleware
 - Database-oriented middleware and gateways
 - Message brokers and other MOM
 - Data warehouse tools and technology
 - Database replication features

Application interface level integration

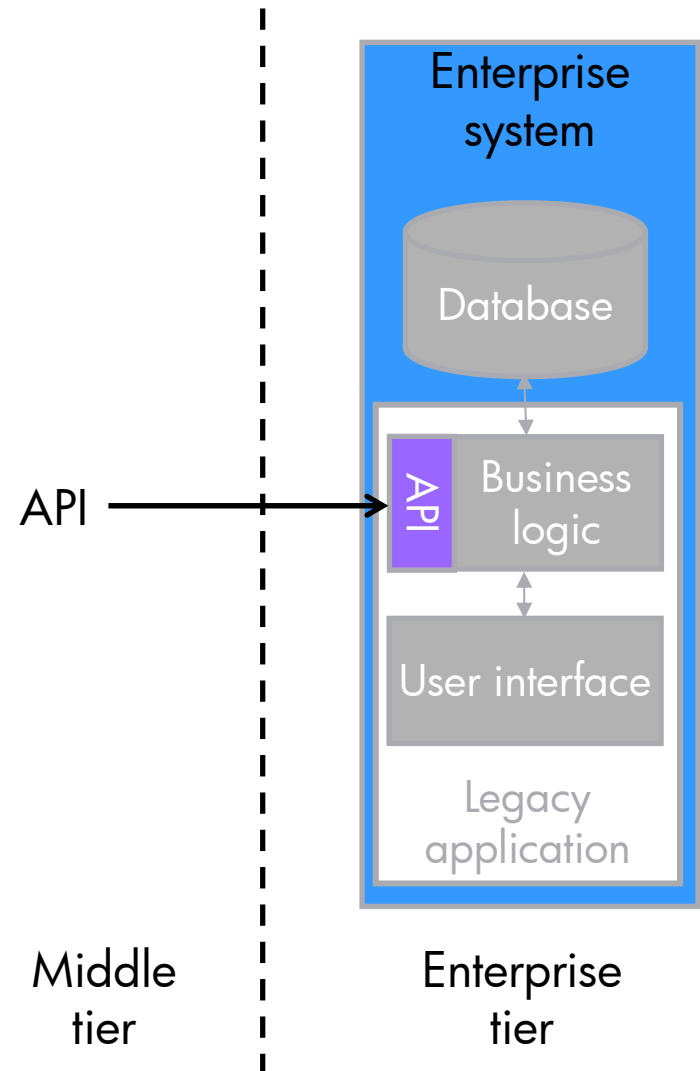


- A wealth of integration possibilities
 - APIs to access data, objects, services, ...
- Need to find points of integration
 - Existing application interfaces
 - ACMS tasks for example
 - Create interfaces from existing application code
 - Create interfaces by application wrapping
 - Packaged application interfaces
- Benefits:
 - Interface at “natural boundaries” in existing systems
 - Provides a mechanism to share business logic and data
 - Infrastructure for sharing common business processes
- Issues:
 - Existence of usable and clean application interfaces
 - Ability to create usable interfaces
 - Quality of interfaces provided by packaged applications
 - Customization might be required

Application interface level integration (2)



- Enabling technologies:
 - Middleware
 - Message brokers
 - Database-oriented middleware
 - Application servers
 - Distributed objects
 - Specialized toolkits
 - Packaged application utilities
 - Packaged applications themselves



Method level integration



- Process integration to create a composite application
 - Powerful, but complex
 - Requires significant up front design and planning
- Shares:
 - Code and business logic
 - Processing
 - Programs
 - Transactions
 - Objects
- Benefits:
 - Very powerful
 - Provides code reuse infrastructure for many applications
- Issues:
 - Can be more complex and expensive than other approaches
 - Requires significant design work, planning, and time
 - Major investment
- Enabling technologies:
 - Application servers
 - TP monitors
 - Distributed objects
 - Traditional development tools

We are pretty much talking about things like web services and SOA here

User interface level integration

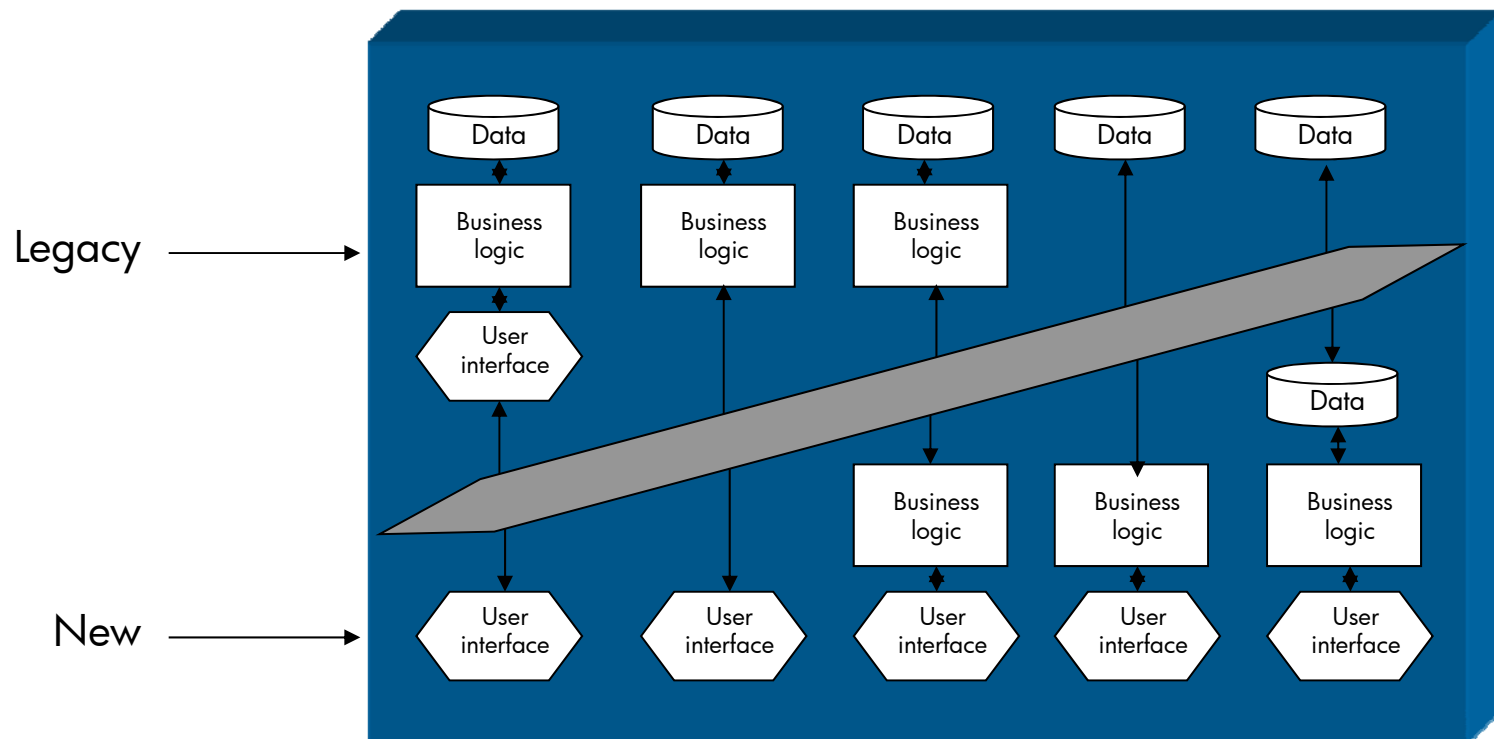


- The last resort?
- Effectively screen scraping of some form or another
 - Screens as objects or screens as data
- Benefits
 - No changes to source or target systems
 - No new interfaces
 - Low risk, low cost
 - Technology is available and stable
- Issues
 - Performance and scalability
 - Only prolongs the integration problem?
- Enabling technologies:
 - 3270 and 5250 emulators
 - Terminal application libraries
 - Screen to object translators
 - Message broker and application server adapters

Which integration option is most appropriate for you?



- That's a tricky question!
 - How open is the application to being integrated?
 - How much do you want to rewrite?
 - How important is performance to you?
 - Does the application have natural boundaries that can be exploited?

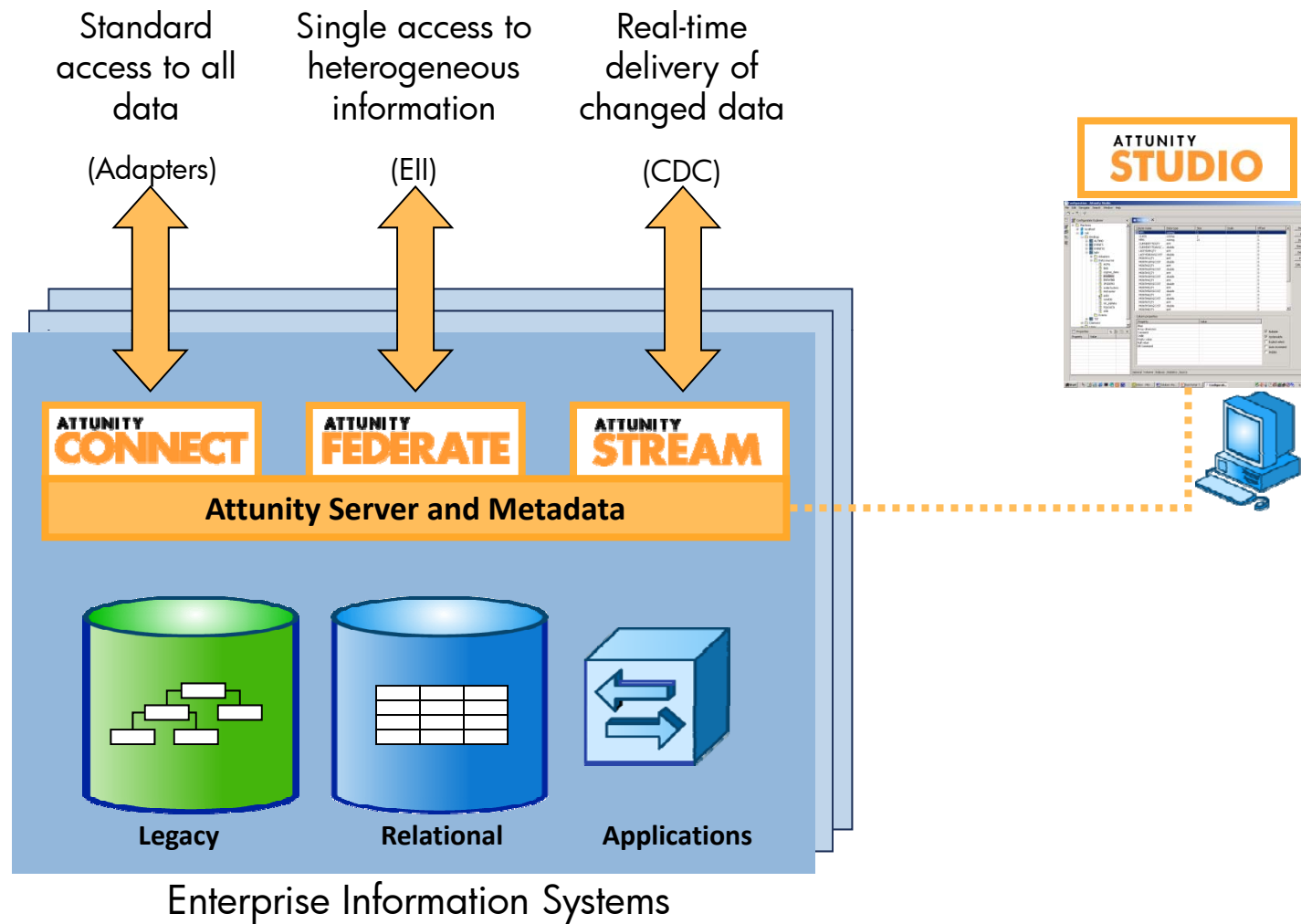


Agenda



- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ and ZeroMQ
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

Attunity Integration Suite overview (1)



Attunity Integration Suite overview (2)



- Typical uses of AIS components:
 - Web sites, portals
 - Business intelligence and data warehousing
 - Application integration and data synchronization
 - SOA
 - Legacy integration and modernization

*Now let's have a quick
look at each member of the
suite...*

Attunity Connect (1)



- A family of pre-built and standard adapters to enterprise data sources and legacy applications
 - Real-time access to heterogeneous data sources
- Key capabilities include:
 - Native adapters to legacy applications
 - Native drivers to relational and non-relational sources
 - Standard integration via SQL and XML
 - Service (XML) abstraction for data and legacy applications
 - Full read/write and transaction support
 - Query governing
 - Robust security, scalability, performance

Attunity Connect (2)



- Supported data sources and applications:

Relational

- Oracle
- DB2
- Sybase
- Informix
- Ingres
- SQL Server
- Rdb
- SQL/MP
- DBMS

Non-Relational

- VSAM
- IMS/DB
- Adabas
- QSAM
- Enscribe
- RMS
- C/D ISAM
- Flat Files
- Delimited Text

Applications

- CICS
- IMS/TM
- Pathway
- TUXEDO
- Natural
- COBOL
- RPG
- C
- Any other 3GL

Attunity Connect (3)



- Supported platforms:

Windows

- XP
- Vista
- Server 2000
- Server 2003
- Server 2008

Mid-Range

- OpenVMS
- HP UX
- AIX
- Linux
- Solaris
- OS/400

High End

- OS/390
- z/OS
- HP NonStop

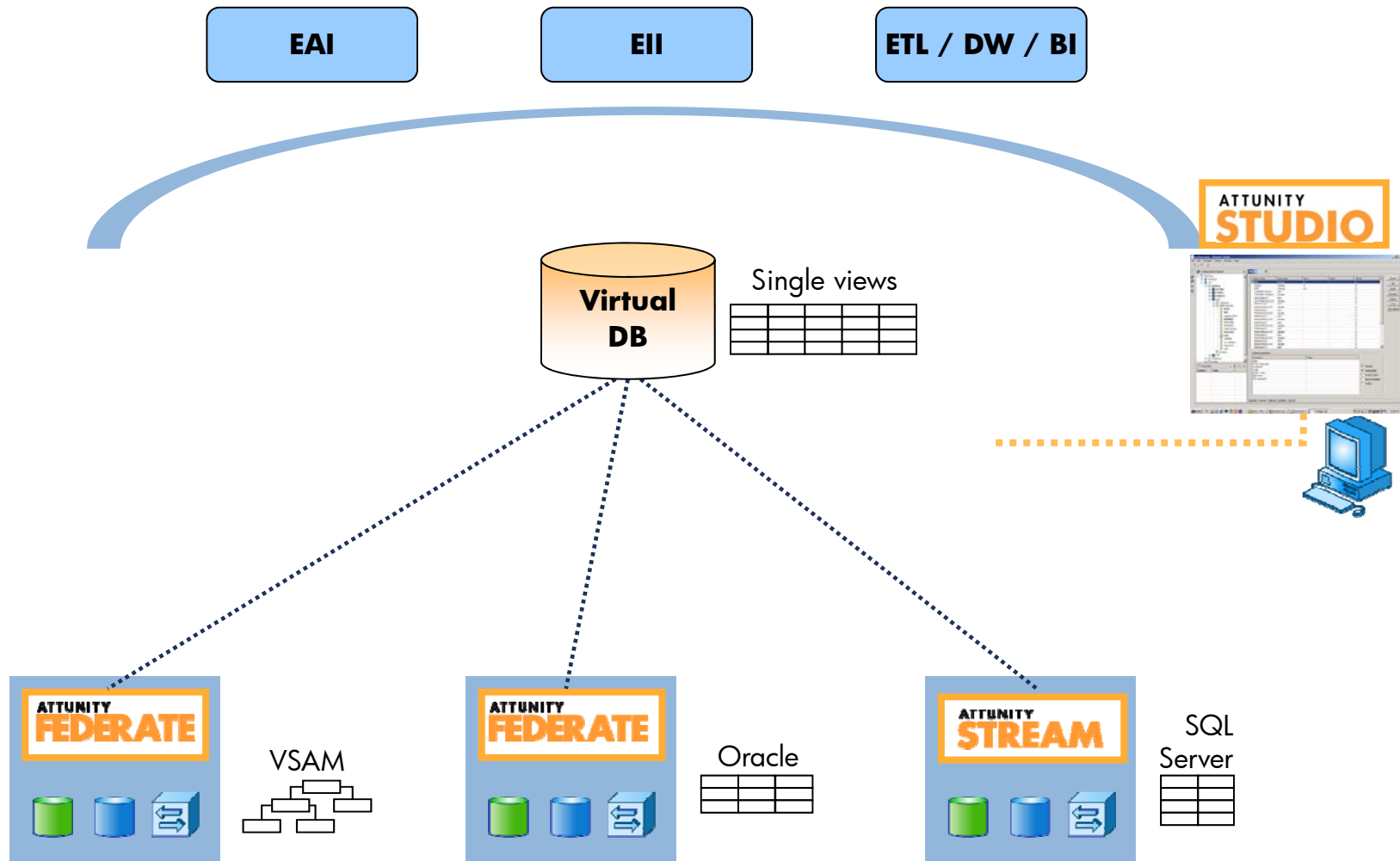
Attunity Federate (1)



- Provides virtual data federation for enterprise information integration (EII)
 - Virtual views across heterogeneous sources
- Key capabilities include:
 - Single virtual catalog across sources
 - Dynamic federated-query processing
 - Virtual views as new data models
 - Optimized, distributed query execution
 - Cost-based distributed query optimizer
 - Distributed query engines (compared to hub-based architectures)
 - Standard access via SQL and XML

*Might remind some folks
of DBI...*

Attunity Federate (2)



Attunity Stream (1)



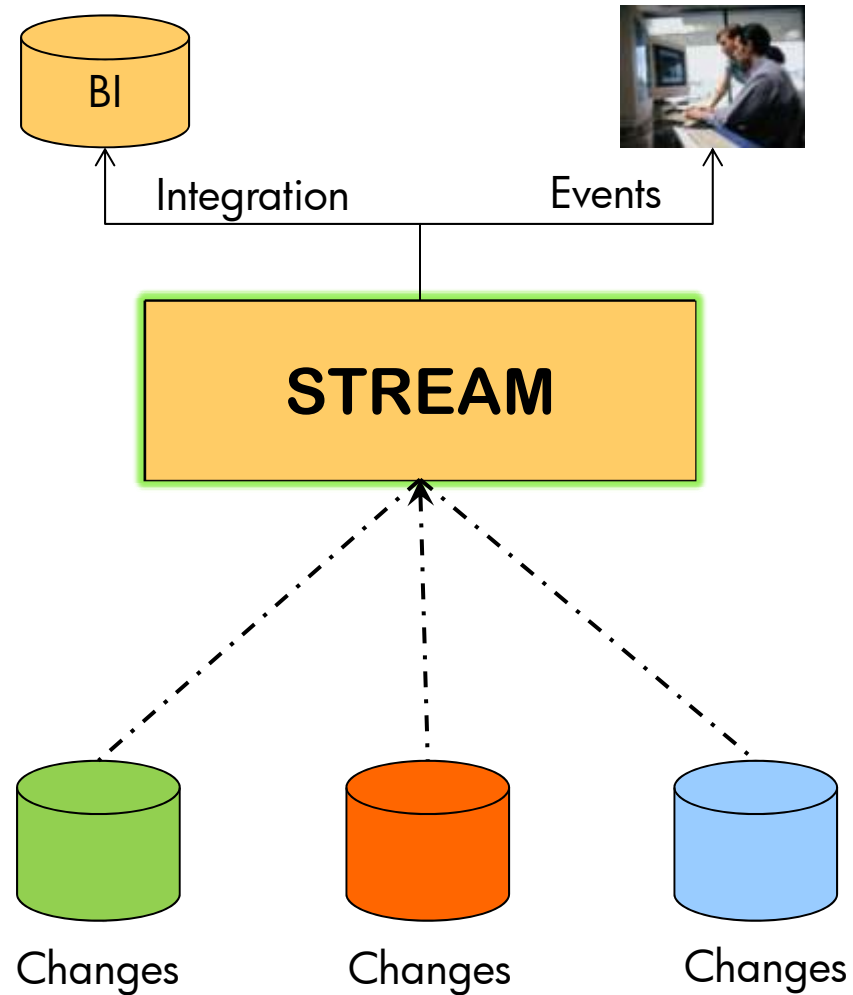
- Log-based, real-time CDC (Change Data Capture)
 - Captures and delivers only the changes made to data sources and transactional systems
- Benefits
 - No downtime and freed up processing time
 - Eliminates batch windows for moving data
 - Reduces the risk of not completing nightly batch data transfer processing
 - Reduces the required resources for ETL
 - Works with virtually any ETL tool
- Key capabilities
 - Low impact, non intrusive change data capture
 - Support for numerous data sources on many platforms (as per other Attunity products)
 - Change record filtering
 - SQL-based change delivery for ETL and data-oriented applications
 - XML-based change delivery for EAI and message-oriented applications
 - Periodic and continuous CDC
 - Reliable delivery and recovery



Attunity Stream (2)



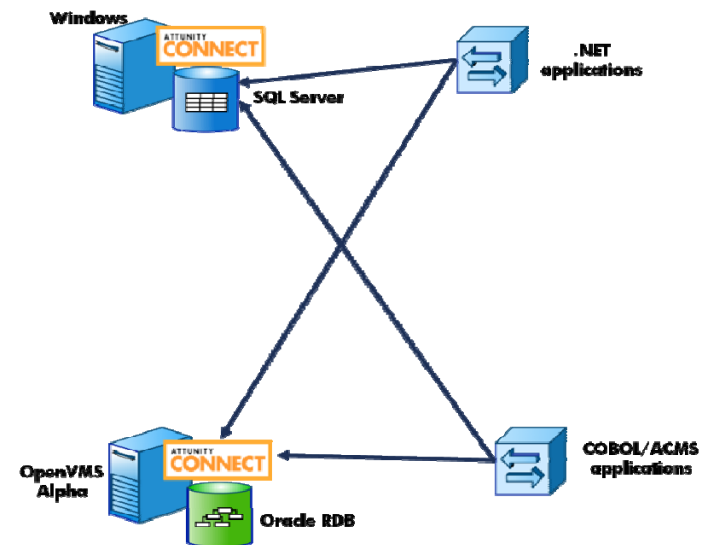
- Real-time capture of data changes and events...



The things we do...



- A large government department
 - Heterogeneous environment
 - Microsoft SQL Server and .NET
 - Oracle Rdb, ACMS, and COBOL on OpenVMS
 - Attunity Connect used to facilitate all database operations
 - Query both SQL Server and Oracle RDB from OpenVMS application code via a common interface
 - Attunity ODBC API
 - Distributed transactions
 - Simultaneous updates to both SQL Server and Oracle RDB databases



Agenda



- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ and ZeroMQ
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

What is middleware?



- Middleware is a generic term for software that interconnects systems
- The key characteristics of middleware are:
 - Application-level messages
 - The objects passed across the network are meaningful to client applications
 - Queuing and routing
 - The middleware must be able to queue messages internally, and route them to different clients in various ways
 - Asynchronous operation
 - Messages are pushed through the network rather than pulled, with queues acting to buffer slower parts of the network
 - Selectable service levels
 - Client applications can explicitly choose between different combinations of speed and reliability

What is middleware?



- An ideal middleware system must also be able to:
 - Handle all kinds of data, opaquely, and without imposing any encoding or representation
 - Handle messages of any size
 - Handle both extremes of high-volume and high-reliability, plus all sensible graduations in between
 - In the first case throughput is critical but messages can be lost
 - In the latter case messages can never be lost
 - Provide several types of routing, including:
 - Queues
 - Publish/subscribe
 - Topics
 - Messages are published according to a hierarchy of names
 - Content-based routing
 - Messages are routed according to key field values

What is middleware?



- An ideal middleware system should also:
 - Be able to interoperate with or simulate other middleware systems
 - Be able to run well on all platforms
 - From small client machines to large servers
 - Numerous operating systems
 - Be cheap enough to deploy without licensing concerns
 - Be easily adapted and extended
- And finally, middleware should:
 - Allow the creation of an abstract network of services
 - Provide ways for application developers to add services
 - Provide ways for replicating data throughout such a network of services

Agenda



- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ and ZeroMQ
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

AMQP introduction



- AMQP (Advanced Message Queuing Protocol) is an open standard application layer protocol for message oriented middleware
- The defining features of AMQP are:
 - Message orientation
 - Queuing
 - Routing (including point-to-point and publish-and-subscribe)
 - Reliability
 - Security
- AMQP mandates the behaviour of the messaging provider and client
 - Implementations from different vendors are truly interoperable
 - Previous attempts to standardise middleware have focussed at the API level
 - This approach did not create interoperability
 - Instead of merely defining an API, AMQP defines a wire-level protocol
 - A wire-level protocol is a description of the format of the data that is sent across the network as a stream of octets
 - Any tool that can create and interpret messages that conform to the defined wire-level protocol can interoperate with any other compliant tool, irrespective of implementation language

History



- AMQP was originally designed to provide a vendor-neutral protocol for managing the flow of messages across an enterprise's business systems
- AMQP was developed from mid-2004 to mid-2006 by JPMorgan Chase and iMatix
 - iMatix also developed the original AMQP implementation (OpenAMQ)
 - See <http://www.openamq.org>
- JPMorgan Chase and iMatix documented the protocol and assigned it to a working group that included Red Hat, Cisco Systems, TWIST, IONA, and iMatix
- As of August 2011, the working group consists of:
 - Bank of America, Barclays Bank, Credit Suisse, Deutsche Börse, Goldman Sachs, JPMorgan Chase Bank, Cisco Systems, HCL Technologies, INETCO Systems, Informatica (29 West), Microsoft, my-Channels, Novell, Progress Software, Red Hat, Software AG, Solace Systems, StormMQ, Tervela, TWIST Process Innovations, VMware (Rabbit Technologies), WSO2
 - See <http://www.amqp.org>
- Although AMQP originated in the financial services industry, it has general applicability to a broad range of middleware problems

Motivation



- AMQP was born of frustration!
 - Message oriented middleware needs to be everywhere in order to be useful, but...
 - Traditionally dominant solutions are typically very proprietary
 - They are frequently too expensive for everyday use
 - They invariably do not interoperate
 - The above *issues* with proprietary solutions have resulted in numerous home-grown developments
 - Custom middleware solutions
 - Custom adaptors
 - The net result for a large enterprise is *middleware hell*
 - Hundred's of applications, thousand's of links
 - Every other connection is different
 - Massive waste of effort
 - Costly to implement
 - Costly and difficult to maintain

*Why does
middleware have to
be so difficult?*

The AMQP vision



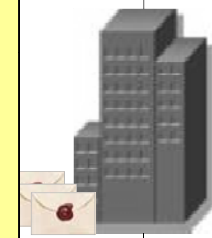
invent
Business
Partners

AMQP Aware Services
C/C++, Java JMS,
Microsoft WCF
and Business Applications

Enterprise



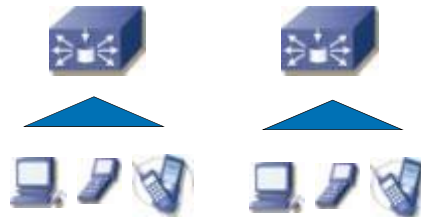
AMQP aims to become the solution for enterprise messaging!



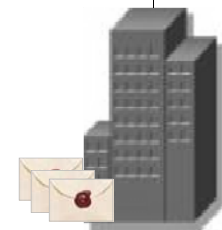
orders@supplier.com



AMQP Aware Clients
Devices & workstations



Branch Offices



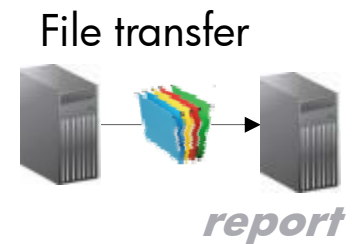
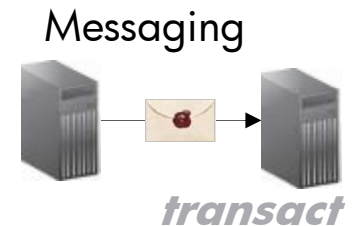
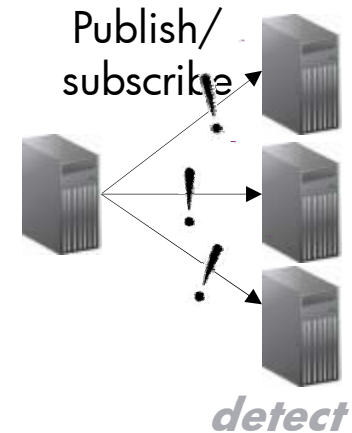
treasury@fundmanager.com

The AMQP model – key features



- Key features of AMQP include:
 - Queuing with strong delivery assurances
 - Event distribution with flexible routing
 - Large message capability (gigabytes)
 - Global addressing scheme (email-like)
 - Meets common requirements of mission-critical systems
 - Service oriented

Any language, any model, any payload, any transport, any platform, reliable, interoperable, manageable, performant, scalable



Comparison with some other protocols



Protocol	Comments
SMTP	Unreliable, slow
HTTP	Synchronous, semi-reliable, no routing
XMPP	No delivery fidelity or queue management
FTP	Point to point, transient, does not work well with NAT/SSL
MQ	Exactly once
TCP	At least once, reliable but short lived, no application-level state management
UDP	Fast but has no delivery guarantees

*AMQP can accommodate all of the above as use-cases...
and switch between them*

The AMQP model – key components



- Message broker
 - Applications connect to a broker to participate in the AMQP network
 - The connection is used to establish a session
 - Sessions provide state between connections, establish identity, ease failover
 - Connections are further subdivided into channels
 - Multiple threads of control within an application can share one connection
- Message layer and queues
 - Application logic interacts only with queues
 - Queues have well known names (they are addressable)
 - Applications do not need to know how messages get in and out of queues
 - Queues can be smart
 - They are an extension point
 - Applications will assign implied semantics to queues (for example, “StockOrderQueue”)
- Nodes and links
 - Move messages between queues and/or applications
 - Contain routing and predicate evaluation logic (similar to complex event processing)
- Peer-to-Peer wire protocol

**Application Layer
(AMQP Brokers)**

Message Layer

Nodes and Links

**Peer-to-Peer
Wire Protocol**

Scales of deployment

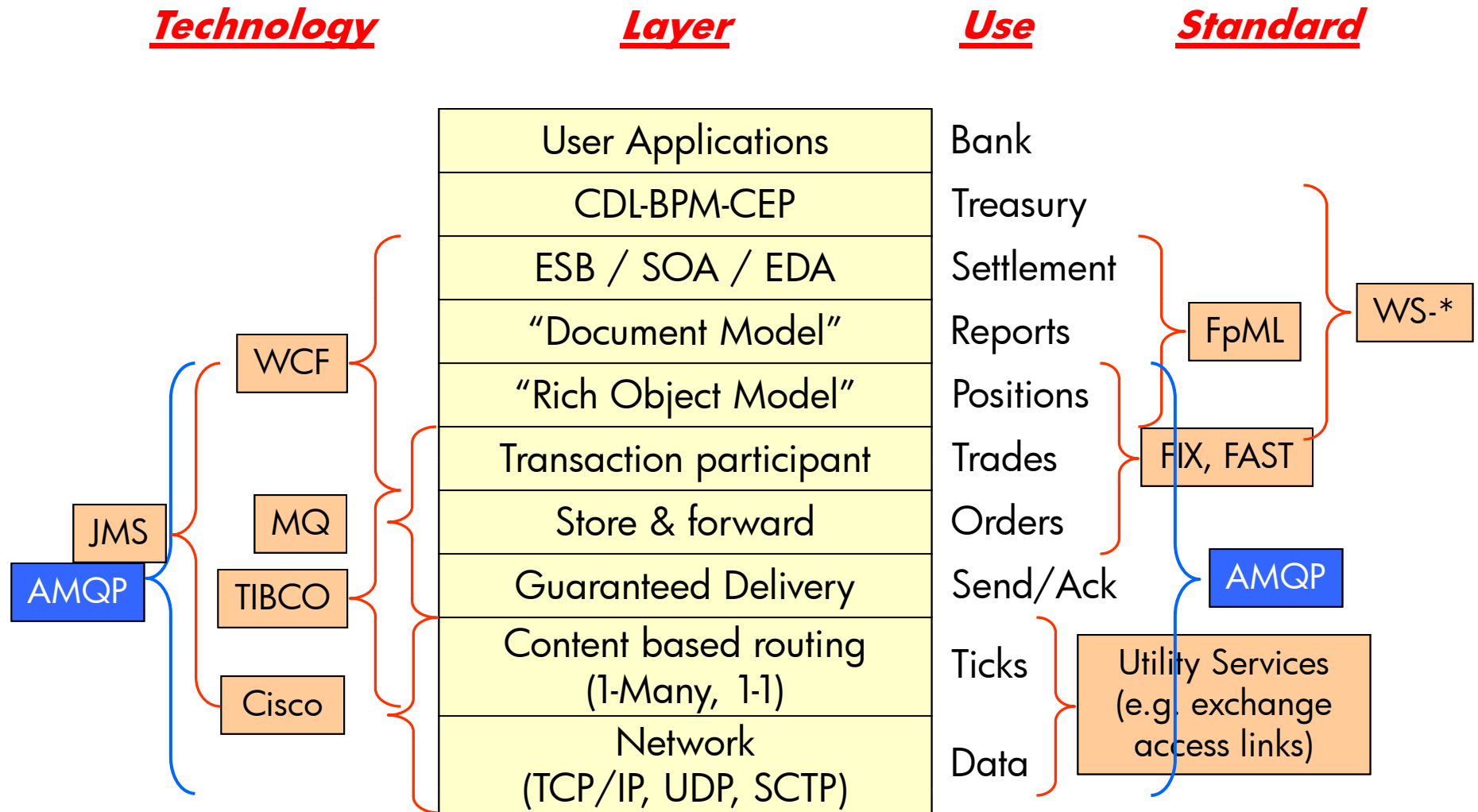


AMQP implementations can cover deployment at different levels of scale ranging from trivial to the mind-boggling:

Type of deployment	Possible scenario
• Developer/casual use	• 1 server, 1 user, 10 queues, 1 message per second
• Production application	• 2 servers, 10-100 users, 10-50 queues, 25 messages per second
• Departmental mission critical application	• 4 servers, 100-500 users, 50-100 queues, 250 messages per second
• Regional mission critical application	• 16 servers, 500-2,000 users, 100-500 queues and topics, 2,500 messages per second
• Global mission critical application	• 64 servers, 2K-10K users, 500-1000 queues and topics, 25,000 messages per second
• Market data (trading)	• 200 servers, 5K users, 10K topics, 250K messages per second

As well as volume, the latency of message transfer is often important; AMQP implementations can deliver messages with latencies of less than 200 μ s

AMQP coverage

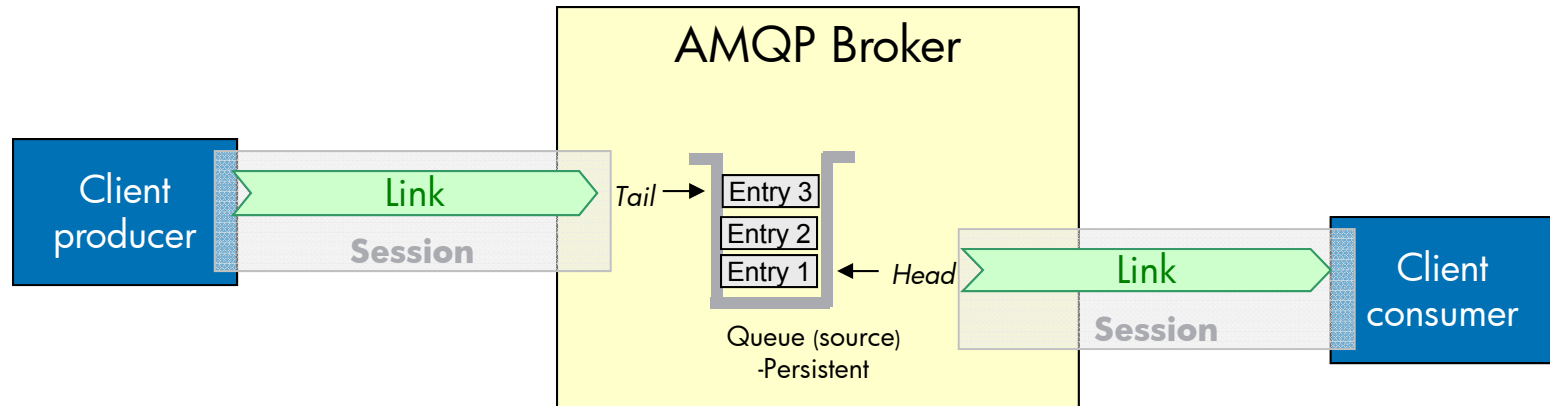


Some typical usage patterns



- Simple point-to-point queue delivery
- Abstracted point-to-point queue
- Load-balanced point-to-point queue delivery
 - Similar to a “multi-reader” queue in DEC/BEA/Oracle MessageQ
- Dynamic (non-persistent) publish/subscribe delivery
- Durable (persistent) publish/subscribe delivery
- Inter-network connectivity

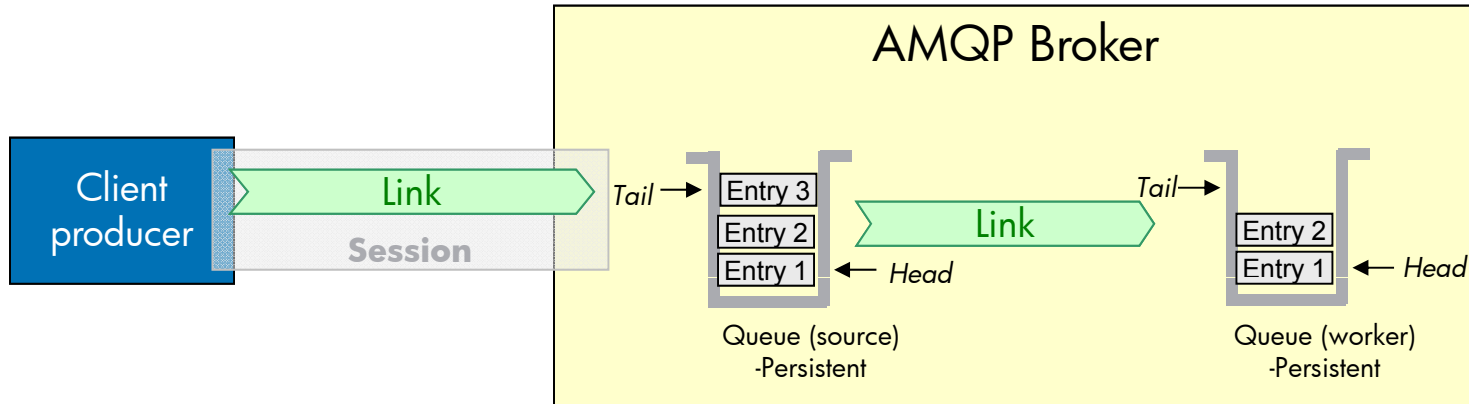
Simple point-to-point queue delivery



Comments:

- Only "source" queue is required and can be read directly by consumer over link (a dedicated consumer worker queue; bridging between source and worker unnecessary)

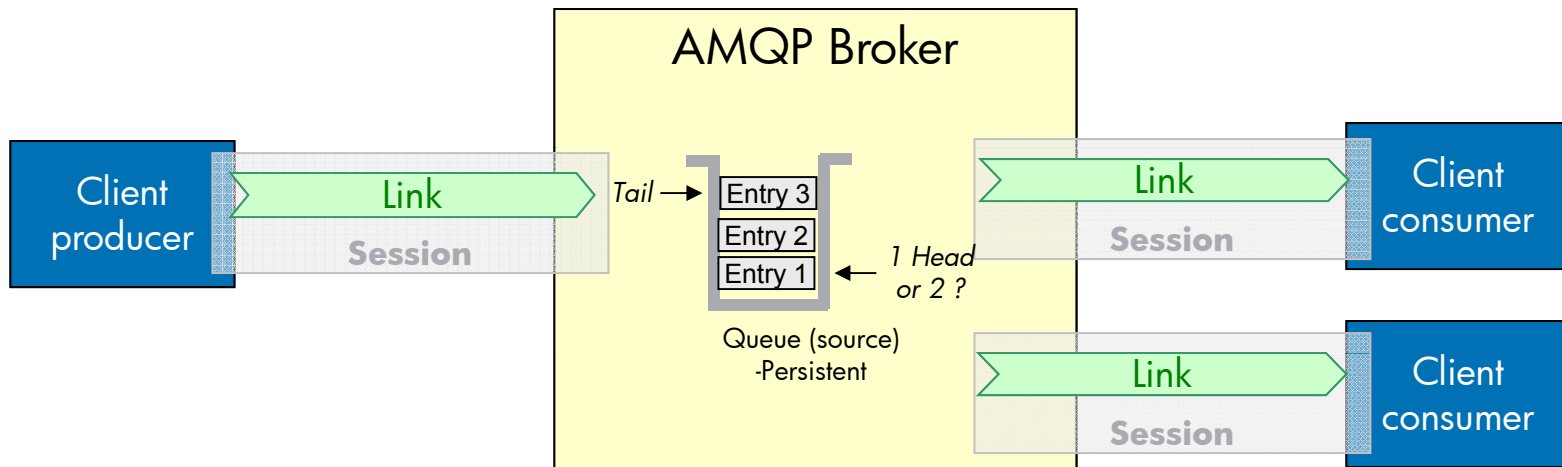
Abstracted point-to-point queue



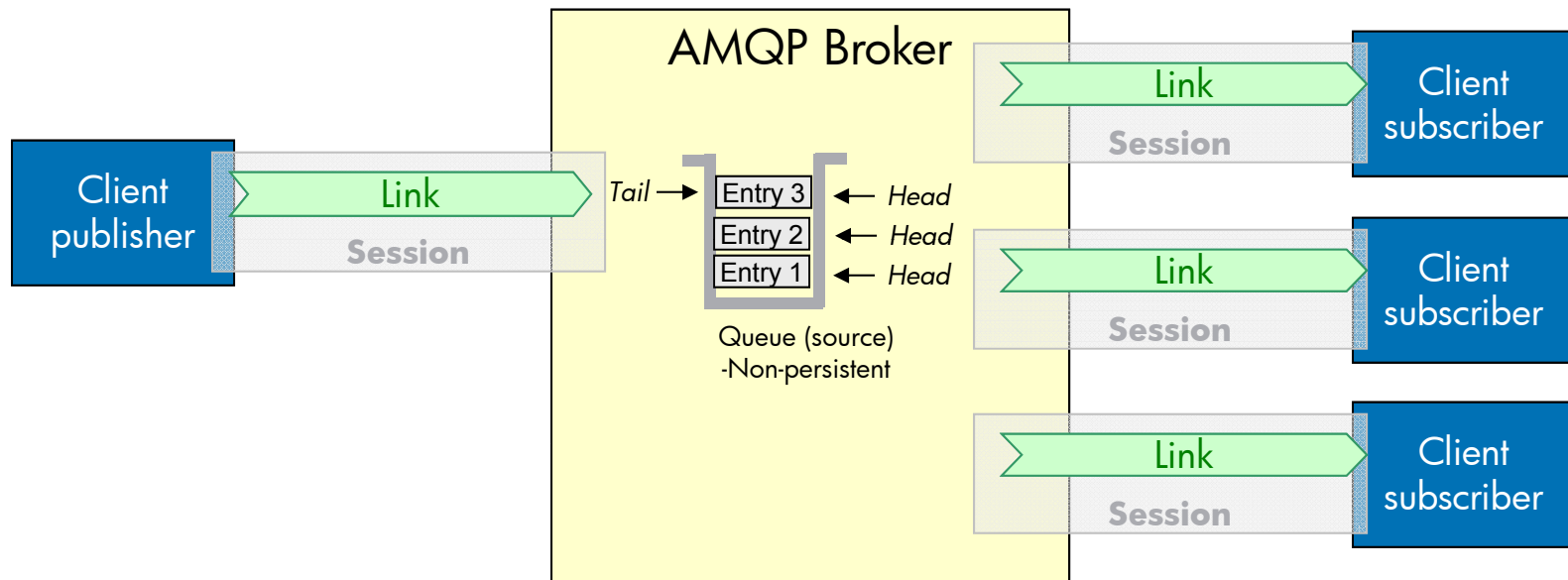
Comments:

- One queue performs the role of holding the "well known" name for the outside world
- All messages are automatically forwarded on to the real worker queue
- Allows internal topology to change without the outside world needing to know

Load-balanced point-to-point queue delivery



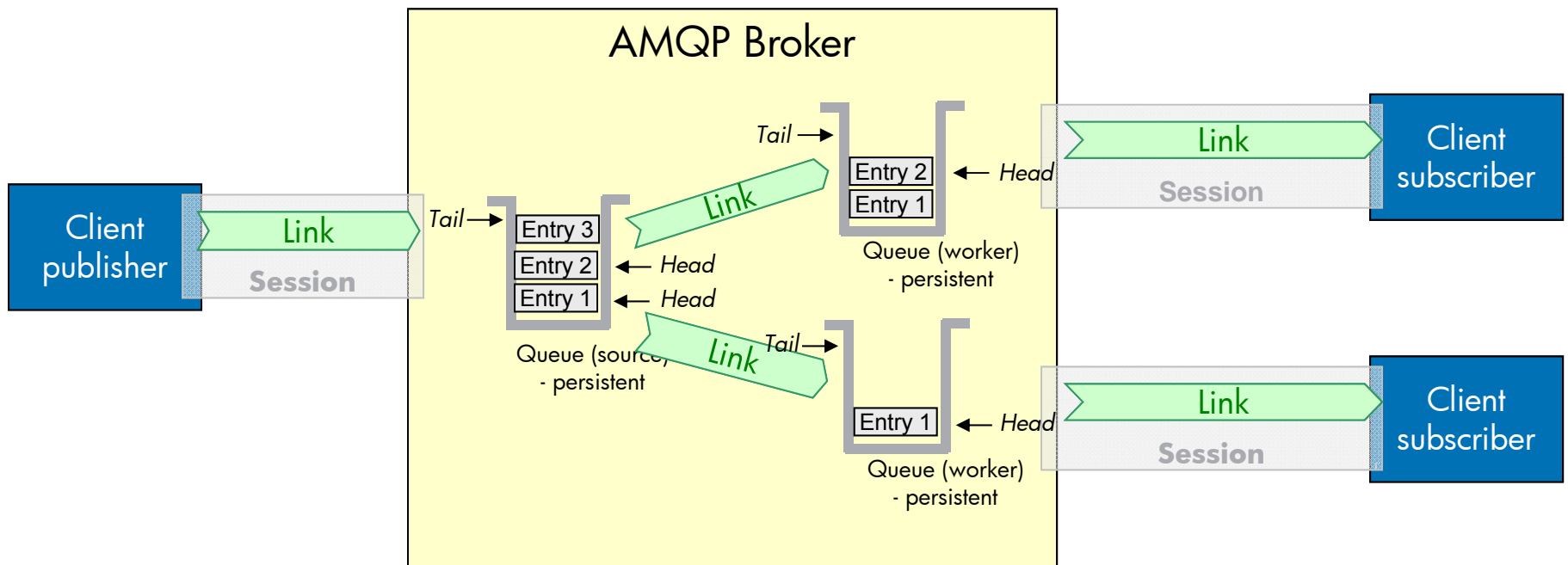
Dynamic (non-persistent) publish/subscribe delivery



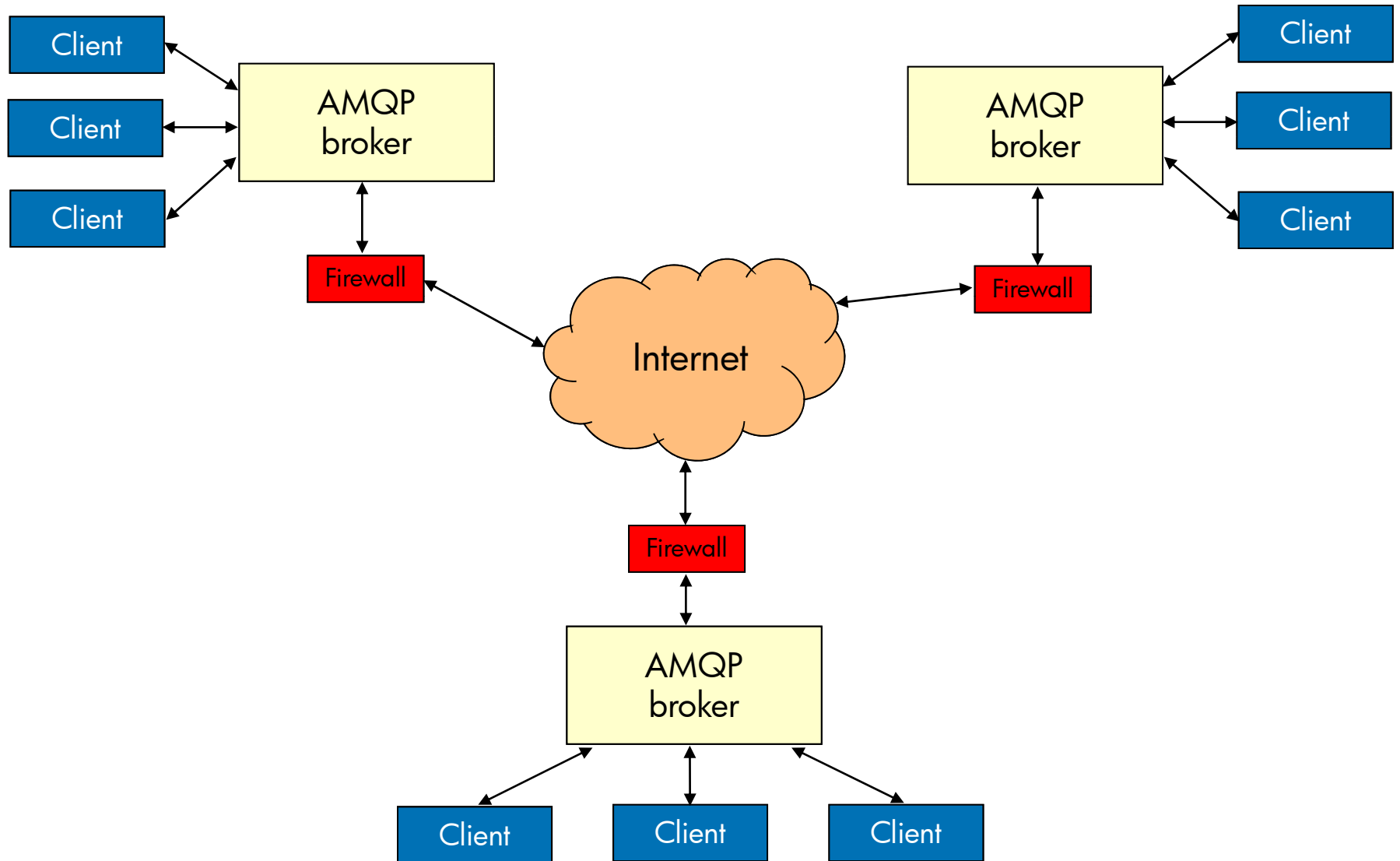
Comments:

- Messages are “garbage collected” in an implementation specific manner after delivery
- AMQP makes some guarantees about how long messages are valid for

Durable (persistent) publish/subscribe delivery



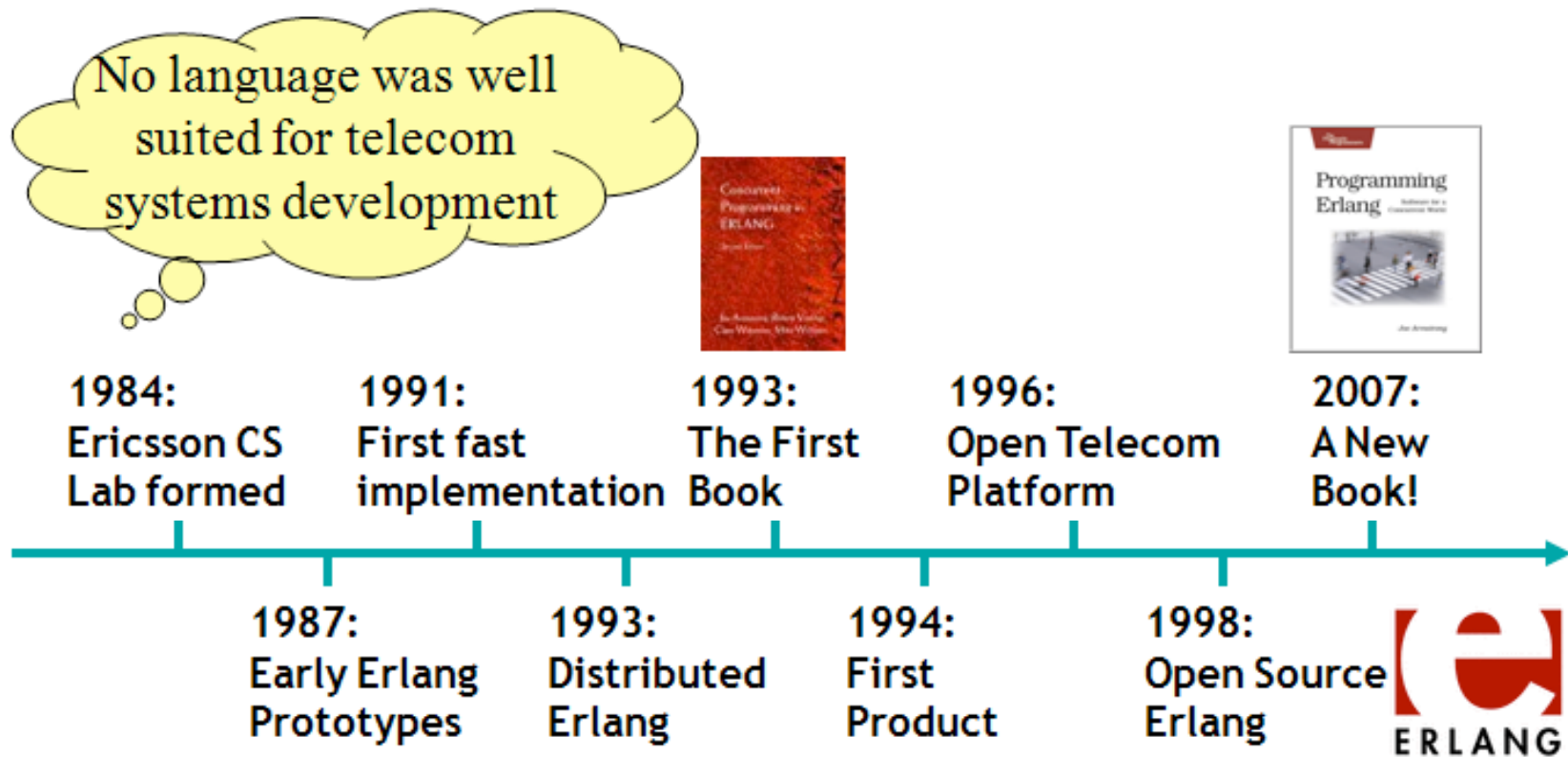
Inter-network connectivity



Erlang (1)



- In his search to find better ways to implement fault-tolerant applications, Joe Armstrong designed and implemented the first version of Erlang in 1986



Erlang (2)



- A general-purpose programming language and runtime environment that has built-in support for concurrency, distribution, and fault tolerance
 - Also supports hot swapping, allowing code to be changed without stopping a system
 - Way before Java could do such things!
- Designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications
 - Used by Ericsson in several large telecommunications systems
 - First version developed by Joe Armstrong in 1986
 - Released as Open Source in 1998
- Erlang implements a “shared-nothing” policy
 - Cooperating Erlang processes (as distinct from operating system processes) communicate using message passing instead of shared variables, which removes the need for any form of locking
- The popularity of Erlang has increased significantly in recent years due to its ability to address problems that cannot be readily or satisfactorily addressed by other languages
 - Particularly concurrency

Erlang (3)



- Erlang is ideally suited to the implementation of distributed, highly-reliable, soft real-time concurrent systems
 - The language provides a simple and powerful model for error containment and fault tolerance
 - Concurrency and message passing are a fundamental to the language
 - Context switching between Erlang processes is typically several orders of magnitude cheaper than switching between threads in a C program
 - Writing applications made of components that execute on different machines is straightforward
 - The Erlang runtime environment is a virtual machine similar to the Java virtual machine
 - Code compiled on one platform can typically be run on any other platform that supports the Erlang virtual machine
- The capabilities of the core Erlang language are extended by OTP (Open Telecom Platform)
 - OTP is a large and collection of libraries for Erlang to do everything from compiling ASN.1 to providing a web server
 - Most Erlang applications make extensive use of facilities provided by OTP

Erlang (4)



Some Erlang highlights:

Functional	<ul style="list-style-type: none">• Light-weight processes• Highly scalable• Message Passing
Concurrent	<ul style="list-style-type: none">• Erlang processes communicate by asynchronous message passing
Soft real-time	<ul style="list-style-type: none">• Response times in the low milliseconds• Per-process garbage collection
Robust	<ul style="list-style-type: none">• Simple and consistent error recovery• Supervision hierarchies• Cooperating processes
Distributed	<ul style="list-style-type: none">• Explicit or transparent distribution• Network-aware runtime system
Hot code loading	<ul style="list-style-type: none">• Easily change code in a running system• Enables non-stop operation• Simplifies testing
External interfaces	<ul style="list-style-type: none">• “Ports” to the outside world behave as Erlang processes
Portable	<ul style="list-style-type: none">• Erlang runs on a Virtual Machine (available for UNIX, Windows, VxWorks, OS X, OpenVMS, ...)

Erlang (5)



- More information:
 - See <http://www.erlang.org/> and <http://erlangonopenvms.blogspot.com/>
- Versions:
 - The current release of Erlang for OpenVMS on Integrity is based on the Erlang R13A distribution
 - Current version is R14B03
- Current status:
 - A few issues to resolve, but generally the port appears quite stable and works very well
 - Able to run several large Erlang applications on OpenVMS
 - RabbitMQ
 - Yaws
 - CouchDB
- Future plans:
 - Further refine current port
 - Investigate porting R1403 (not a small job)
 - Port other Erlang applications
 - Spend more time actually learning the Erlang language – it is brilliant stuff!

RabbitMQ (1)



Im in yr serverz,
queueing yr messagez

RabbitMQ (2)



- A powerful Open Source message broker (message-oriented middleware)
 - The leading (and arguably the most popular) implementation of AMQP
 - Provides a robust and flexible messaging platform designed to interoperate with other messaging systems
- RabbitMQ essentially comprises the following components:
 - The RabbitMQ broker (supports AMQP 0.8 and 0.9.1)
 - Gateways for HTTP, XMPP, STOMP, and other protocols
 - AMQP client libraries for Java, .NET, and C/C++
 - AMQP clients for *numerous* other languages are available from other vendors and/or the Open Source community
 - The "Shovel" plug-in that takes care of copying (replicating) messages from one broker to another
- The RabbitMQ broker is written in Erlang and leverages components of the Open Telecom Platform (OTP) framework for clustering and failover
 - The high availability and functional characteristics of Erlang/OTP make it ideally suited to use in an OpenVMS environment
 - By inference the same may be said for RabbitMQ

RabbitMQ (3)



- Other components available (or that work) on OpenVMS include:
 - libRabbitMQ
 - An API that can be used by OpenVMS-based software applications to exchange data via AMQP
 - Based on an experimental C API developed by the RabbitMQ team
 - Designed so that it can be used with most programming languages that are available on OpenVMS, including C, FORTRAN, COBOL, and Pascal
 - RabbitMQ Java and Erlang clients
 - PHP client
 - Can be used with Apache
 - Python AMQP client (Pika)
 - Plus one or two utilities to simplify development of AMQP-based applications on OpenVMS...
 - (see next slide)

The things we do... (1)

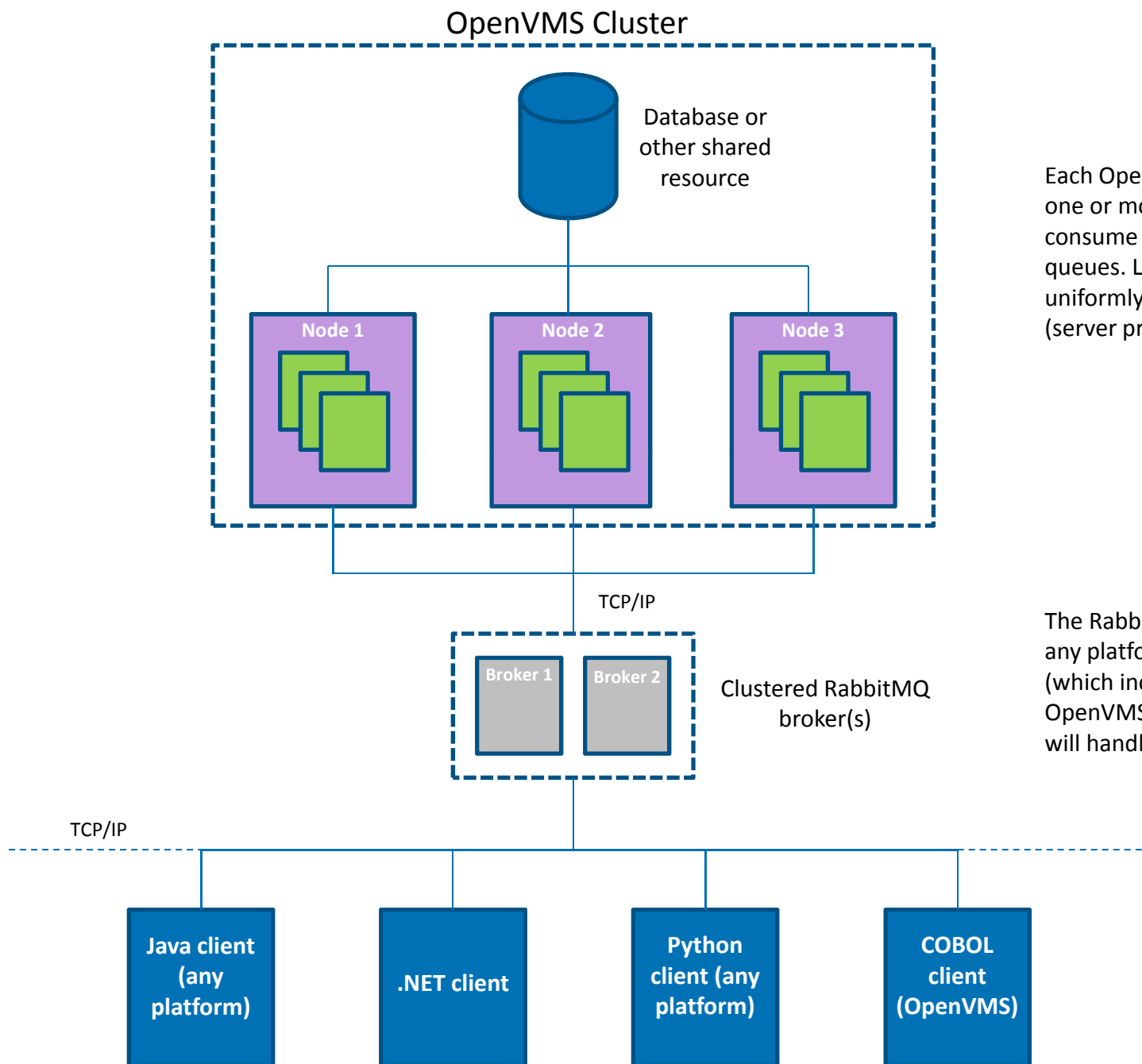


- Created a generic AMQP server (`AMQP$SERVER.EXE`)
 - Uses libRabbitMQ-C, which has been ported to OpenVMS and modified so that it can be readily used from any 3GL
 - Have working examples in FORTRAN and COBOL
 - Not just relevant to the modified WSIT solution discussed in the next slide
 - Can be used to simplify development of AMQP servers (“consumers”)
 - Can be used in an RPC mode (request/response) or to just de-queue and process messages that require no response
 - Still some things to do
 - Improved logging facilities
 - Need to develop a process to boot and monitor `AMQP$SERVER.EXE` instances
 - Endgame:
 - Provide developers with a useful set of tools and libraries with which to develop AMQP-based applications on OpenVMS

The things we do... (2)



- Created a set of modified WSIT Velocity templates to generate code for use with RabbitMQ/AMQP instead of the WSIT run-time
 - Generates C server code (as per WSIT) and server build procedure
 - Build procedure builds a shareable image that can be loaded and called by `AMQP$SERVER.EXE`
 - Generates JavaBeans that use the RabbitMQ Java client API instead of the WSIT inter-process communication classes
 - WSIT inter-process communication classes use ICC (intra-cluster communication), which is OpenVMS-specific
 - RabbitMQ Java client uses TCP/IP and is not tied to any particular platform
- Added RabbitMQ Java client jar files to WSIT
- Endgame:
 - Use WSIT to generate AMQP-aware wrappers for “legacy” application code
 - Use WSIT to generate platform-independent Java classes to interact with wrapped “legacy” code



Each OpenVMS cluster node runs one or more server processes that consume messages from AMQP queues. Load will be balanced uniformly across all consumers (server processes).

The RabbitMQ broker can run on any platform that supports Erlang (which includes HP-UX and OpenVMS). Note that the broker will handle security.

Clients can be implemented in just about any language.

Some other AMQP implementations



- OpenAMQ (<http://www.openamq.org>)
 - Original Open Source implementation of AMQP, written in C by iMatix
 - No longer supported (sadly)
 - Runs on Linux, AIX, Solaris, Windows, OpenVMS, HP-UX
 - Includes broker, APIs in C/C++ and Java JMS, remote administration shell, scripting, federation, failover, and AMQP-over-HTTP via the RestMS protocol
- Apache Qpid (<http://qpid.apache.org/>)
 - A project in the Apache Foundation
 - Includes broker and APIs that support C++, Ruby, Java, JMS, Python and .NET
- Red Hat Enterprise MRG (<http://www.redhat.com/mrg/>)
 - Implements the latest version of AMQP 0-10
 - Rich set of features including management, federation, Active-Active clustering, and APIs for C++, Ruby, Java, JMS, Python .NET

ØMQ (ZeroMQ)



- A high-performance (low-latency) messaging platform
 - Currently achieving 13.4 microseconds end-to-end latencies and up to 4,100,000 messages a second over InfiniBand
- Supports TCP, PGM, and SCTP
- Fully distributed
- Runs on numerous platforms, including OpenVMS, Windows, and pretty much any UNIX/Linux variant
- Language bindings are available for just about any language you can think of, including:
 - C, C++, Java, Python, .NET/Mono, Delphi, Ruby, PHP, Erlang, Perl, and many, many more
 - And there's an OpenVMS calling standard-compliant API's that can be used by an OpenVMS 3GL
- See <http://www.zeromq.org>



ØMQ in 100 words



ØMQ (ZeroMQ, 0MQ, zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry whole messages across various transports like in-process, inter-process, TCP, and multicast. You can connect sockets N-to-N with patterns like fanout, pub-sub, task distribution, and request-reply. It's fast enough to be the fabric for clustered products. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems. ØMQ is from [iMatix](#) and is LGPL open source.

See <http://zguide.zeromq.org/page:all#-MQ-in-a-Hundred-Words>

Expanding on that a little bit...



- ØMQ is a high-performance Open Source messaging API intended for use in the development of highly scalable distributed or concurrent applications
- The API provides message queuing facilities
 - But unlike most message-oriented middleware, a ØMQ-based system can run without the need for a dedicated message broker to coordinate the routing messages between messages
- The library uses entities called sockets (see next slide) to describe and identify connections in accordance with one of the following messaging patterns:
 - Request-reply
 - Publish-subscribe
 - Push-pull (pipeline)
 - This can be quite a complex scenario, involving multiple processes (nodes) connecting in a fan-out/fan-in pattern involving potentially multiple processing steps and loops
 - This is a parallel task distribution and collection pattern
- Messages sent by the ØMQ API are treated as opaque blobs (an array of bytes)
 - It is up to developers to construct messages in such a way that they can be interpreted and appropriately actioned by program code

What more can one say?



A ØMQ socket is what you get when you take a normal TCP socket, inject it with a mix of radioactive isotopes stolen from a secret Soviet atomic research project, bombard it with 1950-era cosmic rays, and put it into the hands of a drug-addled comic book author with a badly-disguised fetish for bulging muscles clad in spandex.

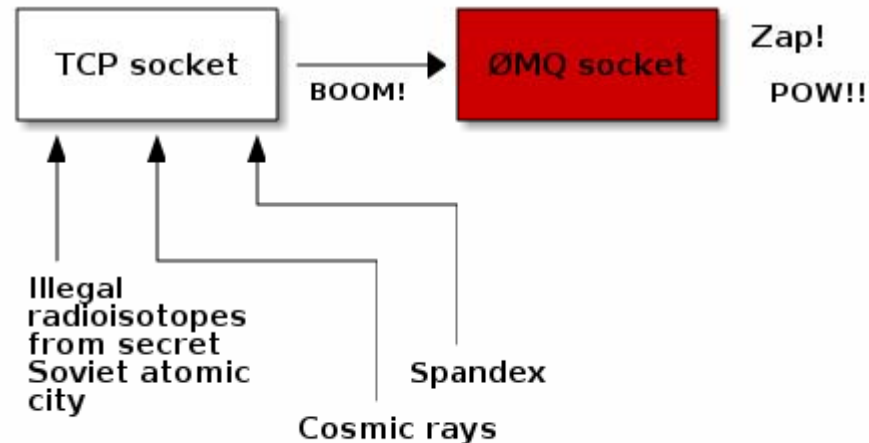


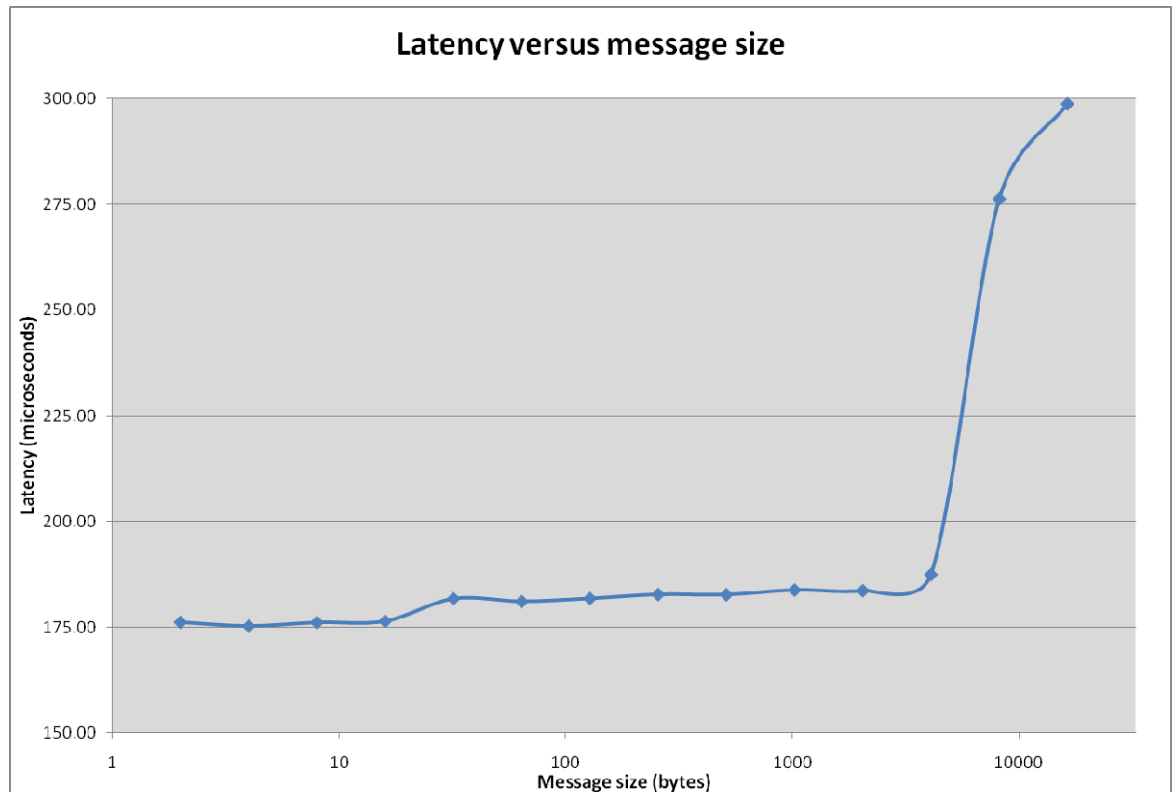
Figure 1 – A terrible accident...

If you are using ØMQ sockets already, congratulations on your purchase! If you are still struggling with products of the past, rather than the future, you will now find *An Introduction to ØMQ* has morphed into Chapter One of [The Guide](#). Enlightenment guaranteed, or your money back.

Results on OpenVMS so far (1)



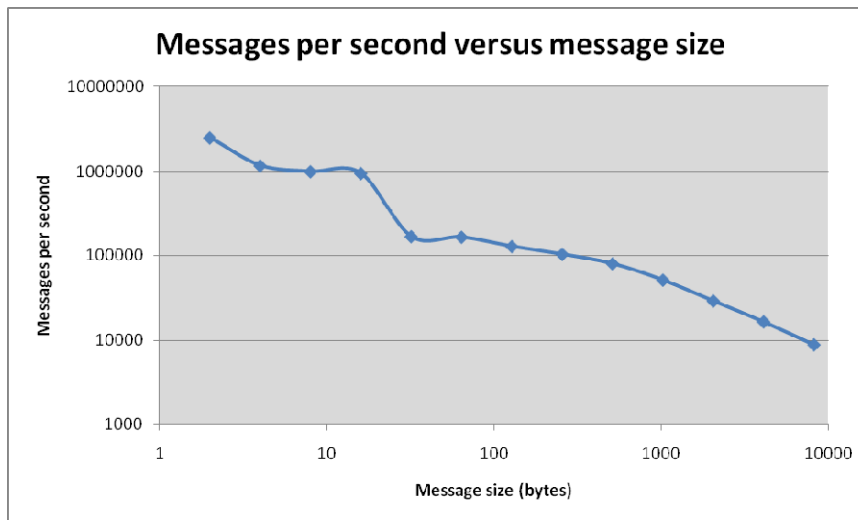
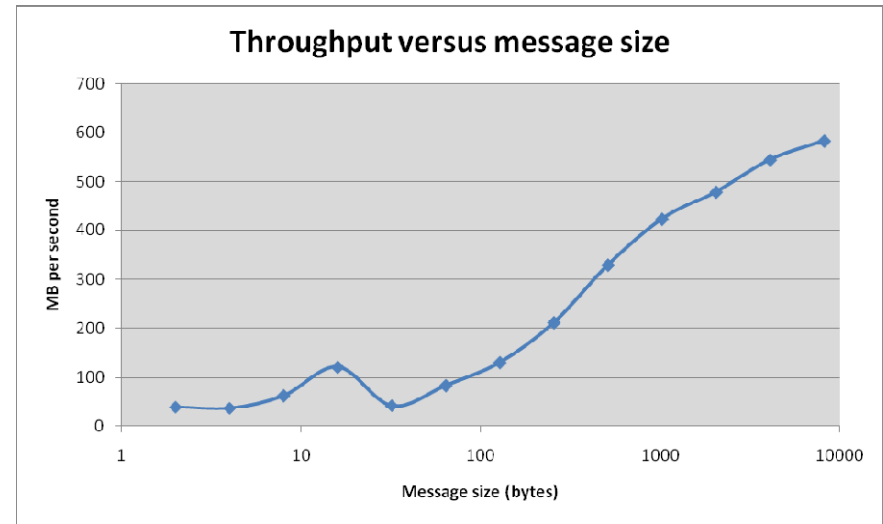
- Results obtained using a pair of HP BL860c systems with several cores and a good chunk of RAM
 - OpenVMS 8.4
 - HP TCP/IP Services V5.7
 - 10Gb NICs
- Code compiled with:
 - HP C++ V7.4-003
 - HP C V7.3-018



Results on OpenVMS so far (2)



- Results not quite on a par with the sub-100us latencies achieved on UNIX/Linux
 - Best results obtained with upcalls enabled
 - Code path very small, so optimization makes little difference
 - Believe limitation is in performance of `poll()/select()`
 - Working on it...



- Porting work identified an issue with the `socketpair()` function on OpenVMS (this has been fixed)

Agenda

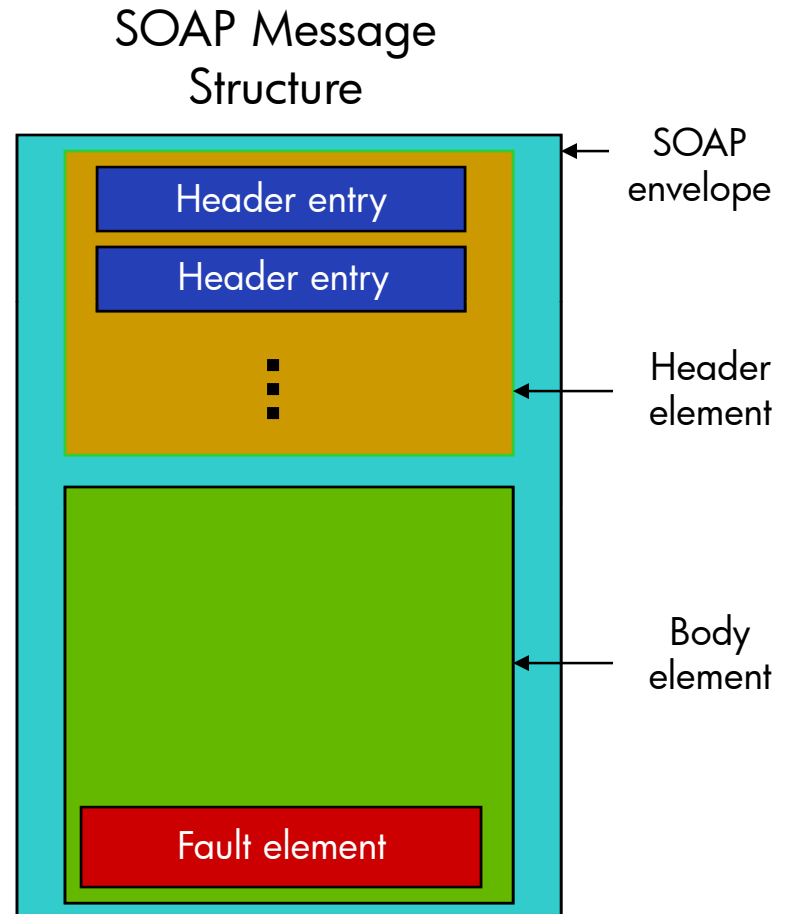


- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ and ZeroMQ
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

SOAP in one slide



- Light-weight protocol based on XML as the marshalling format for data in request and response messages
 - Encoding rules for data type instances
 - Vendor and platform-neutral
 - Language-neutral
 - Object model neutral
 - Transport neutral
- Designed for loosely-coupled distributed computing
 - No remote references
- XML allows data transformation (XSLT)
- XML enables long-term data persistence



What is gSOAP?



- Full-featured Open Source SOAP technology
 - See <http://www.cs.fsu.edu/~engelen/soap.html>
- More than 2500 registered users (including several Fortune 100 companies)
- Conforms to WS-I Basic Profile V1.0a
- Uses a source-to-source stub and skeleton compiler to automate the integration of SOAP RPC in applications
 - Client and server development
 - Automates the deployment of (legacy) 3GL applications as Web services
 - Primarily intended for use with C/C++, but can be used (with a little effort) with any 3GL
 - Automates the development of clients
- Suitable for high-performance web service computing (very fast)
- Major components available on OpenVMS (Alpha and IA64)
 - Extensions to simplify use from languages other than C/C++
 - ACMS support (threaded agent gateway)

gSOAP goals



- Application-centric
 - Minimize legacy application code adaptation
 - Support (de)marshalling of application's native data structures in SOAP/XML
 - Preserve the logical structure of data
- Minimize data migration overhead and formatting errors
 - Avoid (hand-written) wrappers
 - Generate fast (de)marshalling routines and streaming XML parsers
 - Efficient run-time remote object allocation

gSOAP tools



- Stub/skeleton compiler (`soapcpp2`)
 - Generates source code stubs and skeletons for SOAP RPC
 - Generates XML (de)marshalling routines for native and user-defined C/C++ data types
- WSDL/schema parser (`wSDL2h`)
 - Imports one or more WSDL files and XSDs to generate a C/C++ header file that defines the service prototypes and data types
 - The C/C++ header file would then be used as input to `soapcpp2`
- gSOAP runtime
 - Provides low-level HTTP, TCP, SOAP/XML handling, and memory management capabilities
- See <http://www.cs.fsu.edu/~engelen/factsheet.pdf> for a full list of features

gSOAP development

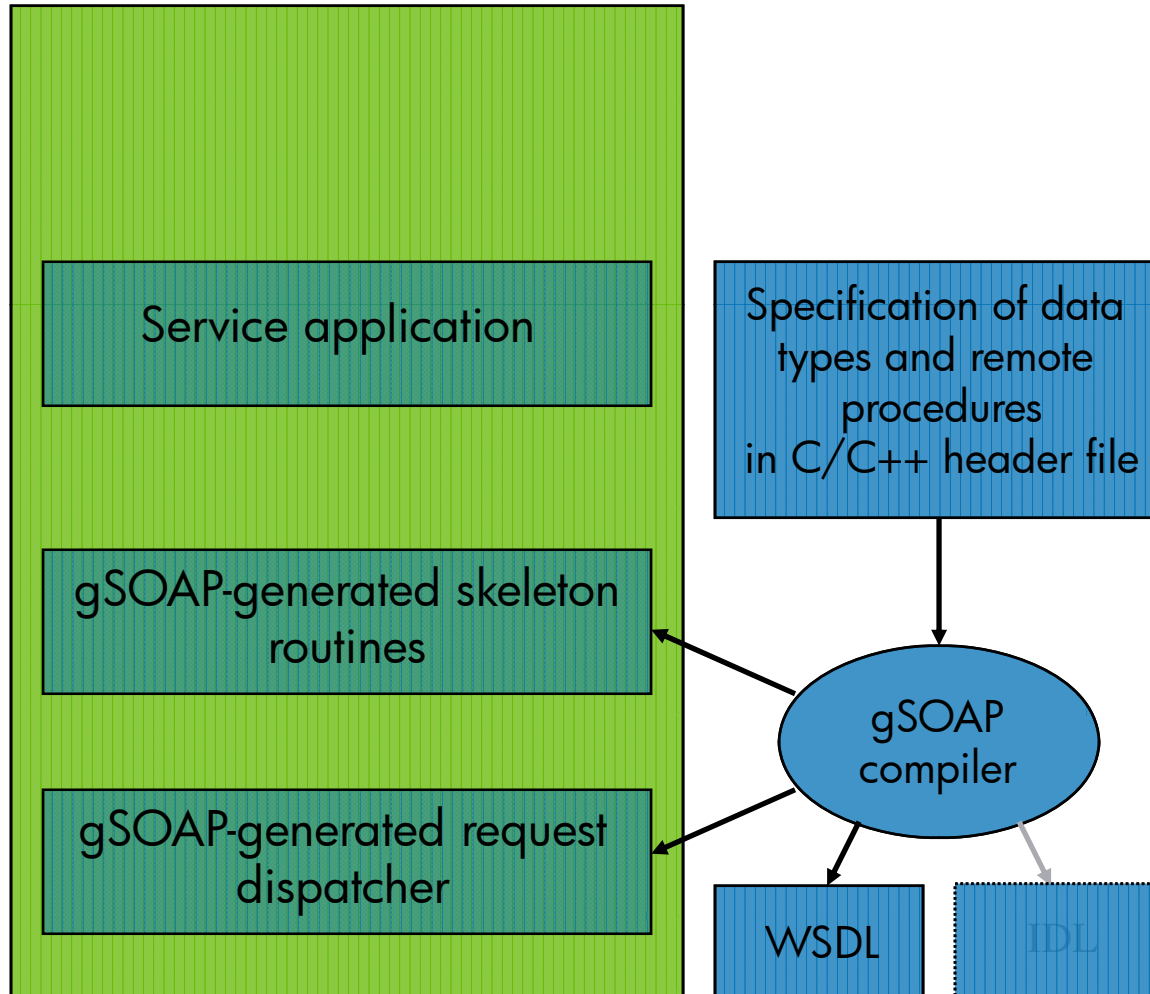


- Two basic approaches to development...
 - Start with a WSDL (*top down*)
 - Approach typically used when wanting to call an existing Web service
 - Use `wsdl2h` to convert WSDL to C/C++ header file
 - Use `soapcpp2` to generate stubs and skeletons
 - Develop client application
 - Link client application with generated code and gSOAP runtime
 - Start *without* a WSDL (*bottom up*)
 - Approach might typically be used to expose existing (legacy) functionality as Web services
 - Create a C/C++ header file containing the necessary data type and service (function prototype) definitions
 - Use `soapcpp2` to generate stubs and skeletons
 - Link generated code and gSOAP runtime with existing (or new) application code

gSOAP development



Server



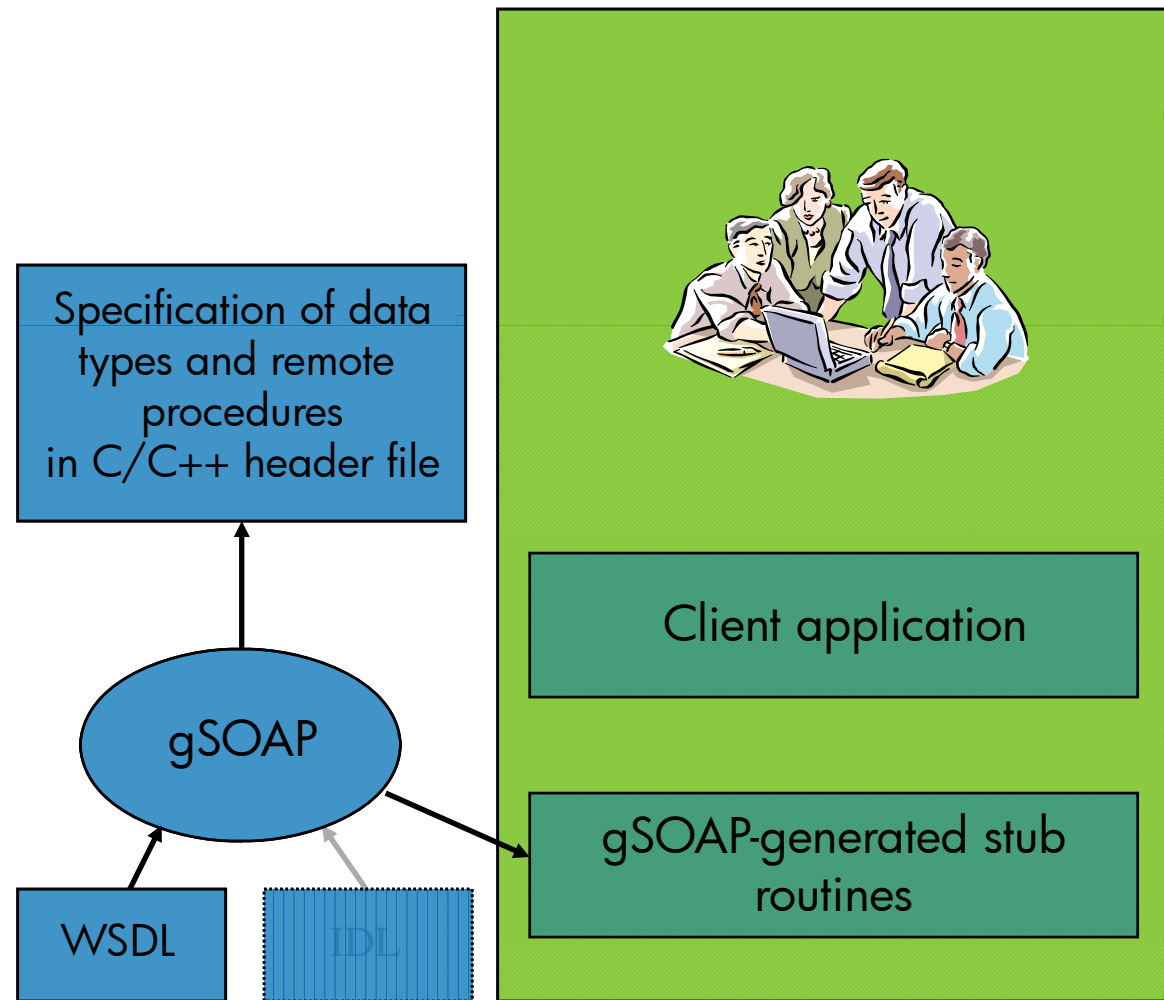
Supplied header file is processed by `soapcpp2` to generate stub and skeleton routines that are linked with user-written application code. Note that `soapcpp2` can optionally generate WSDL.

gSOAP development



An existing WSDL can be used to develop a gSOAP client (or server) application. The `wsdl2h` tool is used to convert the WSDL into a header file, which can be processed by `soapcpp2` to generate stub and skeleton routines that are linked with user-written application code.

Client



gSOAP OpenVMS port



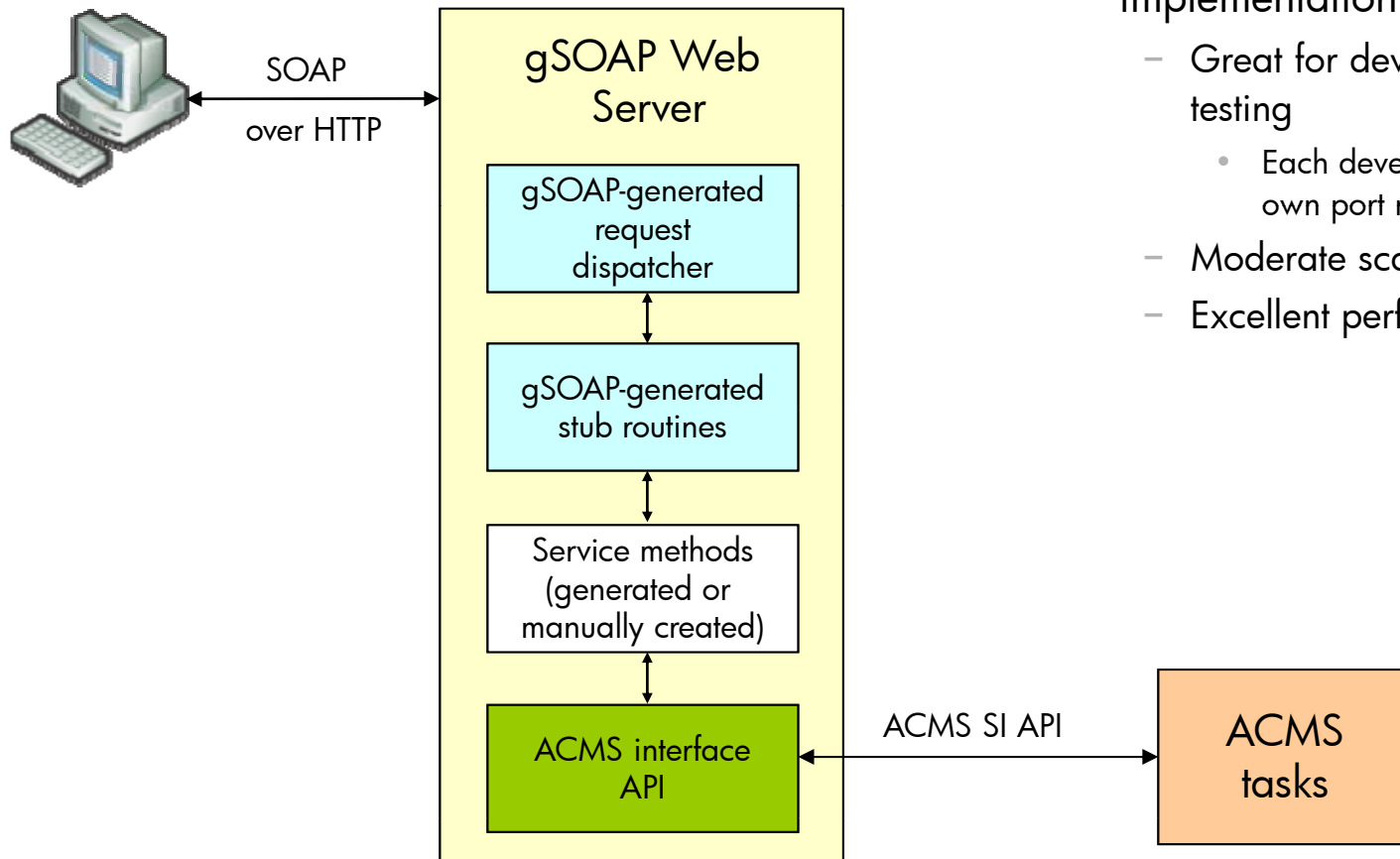
- Major components available on OpenVMS Alpha and IA64
 - soapcpp2.exe
 - Stub and skeleton compiler
 - Generates proxies (and RPC stubs)
 - Generates the C/C++ Web service skeletons
 - Can optionally generate WSDL and XSDs
 - wsdl2h.exe
 - WSDL parser
 - Converts WSDL into gSOAP header file specifications of Web services
 - Object libraries (runtime libraries)
 - Provide a transport layer with an HTTP stack on top of TCP/IP
 - Optional support for SSL and DIME/MIME attachment
 - mod_gsoap
 - Apache module for gSOAP
 - Includes some little “modifications”

Wrapping ACMS applications



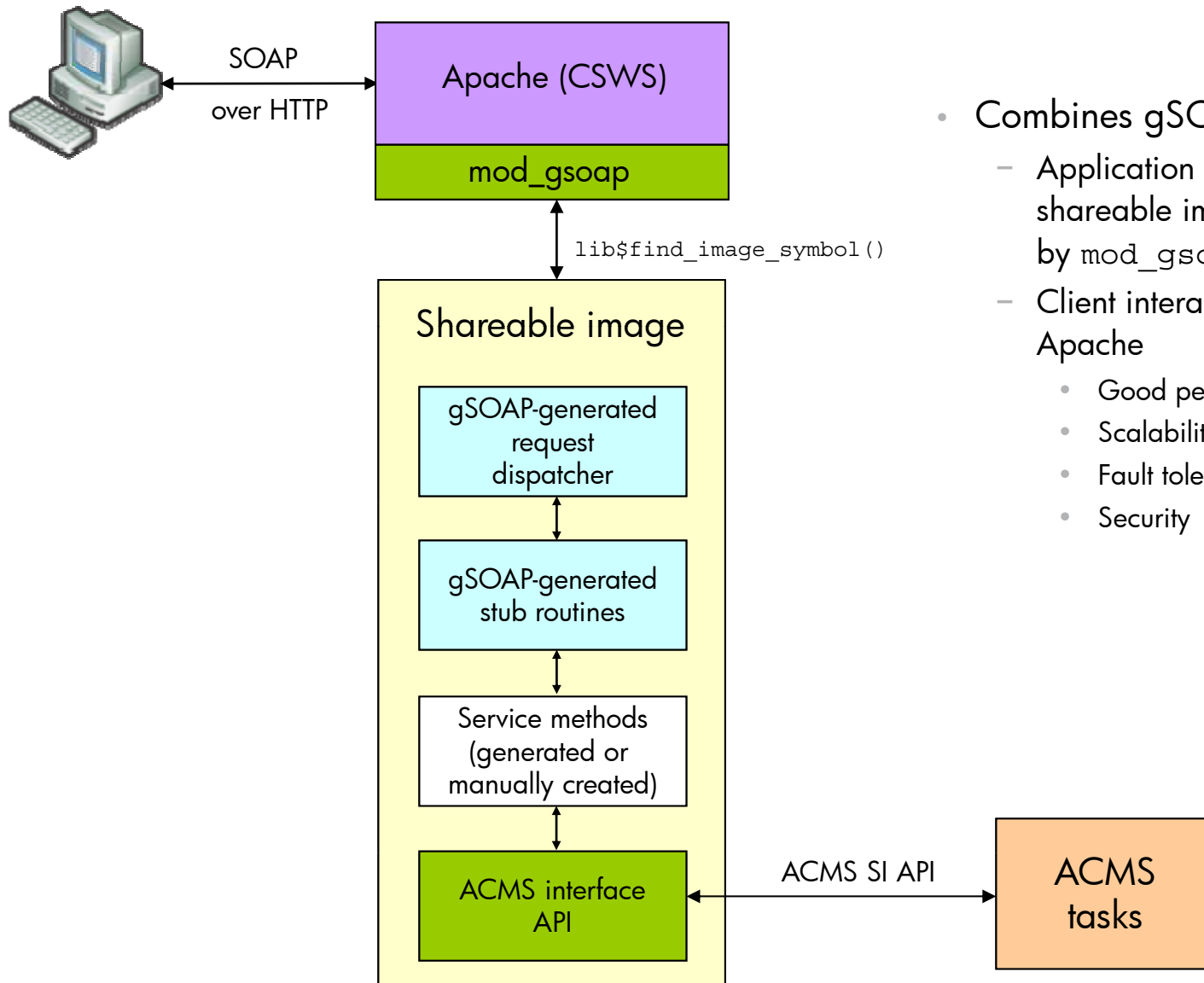
- `fiddle` is a powerful facility to generate a gSOAP-based web services wrapper for an ACMS application
 - Input is an ACMS STDL file
 - Completely defines the interface
 - Output is:
 - A gSOAP-compliant interface definition
 - Interface wrapper code
 - Can deploy as a standalone gSOAP server or run under Apache using `mod_gsoap`
 - At most all you need to do is write a simple “main” routine

Wrapping ACMS applications - standalone gSOAP server



- Provides a minimal Web server implementation
 - Great for development and unit testing
 - Each developer can use their own port number
 - Moderate scalability
 - Excellent performance

Wrapping ACMS applications - gSOAP with Apache



- Combines gSOAP and Apache
 - Application linked as a shareable image that is loaded by mod_gsoap
 - Client interaction managed by Apache
 - Good performance
 - Scalability
 - Fault tolerance
 - Security

Summary of steps (automated wrapping)



1. Obtain STDL file for the ACMS application
 - Modify STDL file as appropriate to include only the tasks you wish to expose
 - Create a simple C main routine
 - Useful for development and developer testing
 - For system testing and the production environment it is better to deploy services under Apache via `mod_gsoap`
2. Use “fiddle” tool to generate the gSOAP interface definition based on the STDL
3. Use the gSOAP `soapcpp2` tool to generate WSDL and stub code for the service(s)
4. Link generated stub code with main routine and gSOAP RTL components
5. Run your new gSOAP server process and test the interface using a tool such as `soapUI`
 - Make sure that the username under which the server is being run has permission to access the ACMS application
6. Develop clients to use your new services!

What people are doing with gSOAP



- An Australian customer is calling an Internet-based web service from an existing COBOL application
- Another customer in Australia is invoking a web service from an existing application and is using SSL for additional security
- We assisted a US customer in creating a web services interface to their existing FORTRAN application, including a content-based router that routes services based on their XML content
- A customer in Europe is calling a gSOAP-based web service running on OpenVMS from a Java-based PDA client to perform telephone number lookups
- Several customers (US and Europe) have exposed one or more ACMS tasks as web service methods
- And many more...

Agenda



- Introduction
- Data-level integration technologies from Attunity
- What is middleware?
- Queuing as an integration medium using RabbitMQ and ZeroMQ
- Calling and exposing web services from and on OpenVMS using gSOAP
- Summary and questions

Summary



- There are numerous OpenVMS users running large, complex, business-critical bespoke applications that have served the business well for many years and continue to do so
 - For whatever reason these applications may now need to interoperate and exchange data with other external systems via industry-standard mechanisms
 - Integration solutions abound for OpenVMS, be they Open Source or commercial products
 - Web services
 - Standards-based messaging
 - Low-latency messaging
 - Data-level integration solutions
 - Basically, you can do on OpenVMS pretty much anything you can do on other “more open” platforms and in many cases this can be done in an extremely cost-effective manner
- Okay, I’ve probably talked enough...

Questions?





i n v e n t