Building Reliable Messaging Systems using Web Services and WSRM

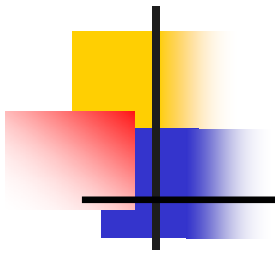
Paul Fremantle

VP of Technology

WSO2

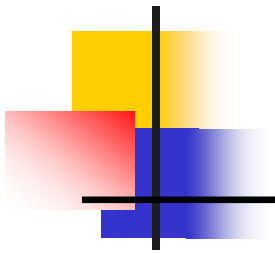
About me

- Co-founder and VP of Technology and Partnerships at WSO2
  - The leading Open Source Web services company
- Co-Chair of the OASIS WSRX Technical Committee
- Committer on Apache Synapse
- Member of the Apache Software Foundation
- Co-author of *Building Web Services in Java 2nd Edition*
- http://bloglines.com/blog/paulfremantle
- Ex IBM Senior Technical Staff Member
  - developed the IBM Web Services Gateway
  - Apache WSIF
  - Apache Axis C/C++
  - JWSDL/WSDL4J – now Apache Woden



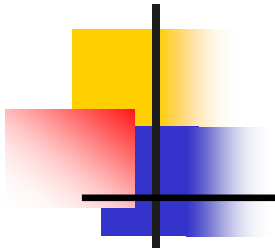
Contents

- WSRM
  - Overview
  - History
  - 1.1 *vs*. 1.0
  - Futures
- Implementing RM with Apache Sandesha2
  - Axis2
  - Synapse
- Wrapup



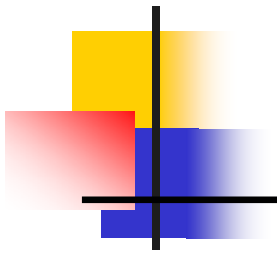
WSRM Aims

- To help ensure that messages are delivered to their destination
  - Exactly Once In Order is the most common requirement
- "Composable" with other specifications and existing systems
  - Doesn't re-implement those aspects
  - Has the ability to work with other standards (security, transactions, addressing)



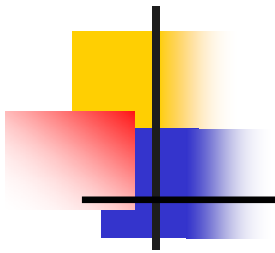
History

- A specification designed by IBM, Microsoft, BEA and Tibco
- Designed to compose with WS-Addressing, WS-Policy
- Originally published in March 2003
- Refreshed March 2004, Feb 2005
- Submitted to OASIS June 2005



This Presentation

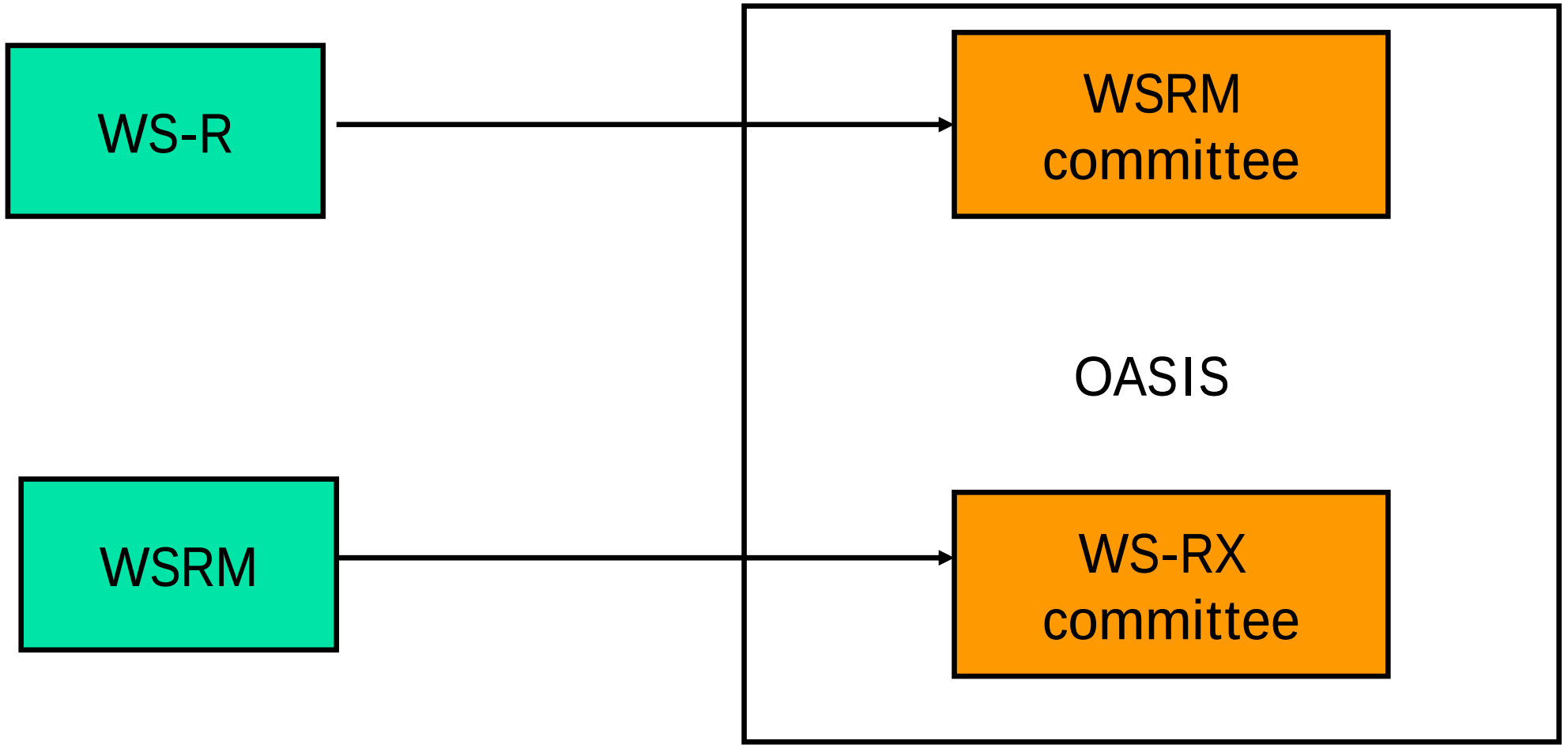
- Mainly based on the public review (CD4) of WSRM 1.1 (OASIS)
- With appropriate comments on the Submission spec (WSRM 1.0)



WS-Reliability

- An alternative specification proposed by Fujitsu, Hitachi, NEC, Oracle, Sun
- Submitted to OASIS and standardised, currently at v1.1
- One available implementation:
  - RM4GS Reliable Messaging for Grid Service
  - A sample/reference implementation

WSRM and WS-RX and WS-Reliability



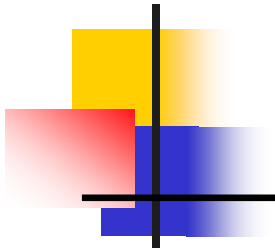
WS-RX TC Members Include Representatives of:

- Actional Corporation
- Adobe Systems
- BEA Systems, Inc.
- BTplc
- Choreology Ltd.
- Ericsson
- Fujitsu Limited
- Hitachi, Ltd.
- IBM
- IONA Technologies
- JBoss Inc.
- Microsoft Corporation
- NEC Corporation
- Nortel Networks Limited
- Novell
- Oracle Corporation
- SAP AG
- Sonic Software Corp.
- Sonoa Systems, Inc.
- Sun Microsystems
- Tibco Software Inc.
- webMethods, Inc.
- WSO2

While the TC is working from the submitted spec, it aims to meet the requirements of all the TC members within the scope of the charter

WSA Revision

- Web Services Addressing has a couple of weird URIs I'm going to refer to
  - http://www.w3.org/2005/08/addressing/anonymous
    - The 'anon' URI
    - Means the backchannel of the transport – *e.g.* in replyTo
  - http://www.w3.org/2005/08/addressing/none
    - Don't send me a message



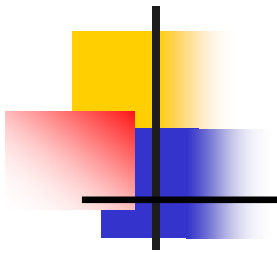
A key point

- The WSRM specification is a

  CONNECTION ORIENTED PROTOCOL

  Designed to support multiple messages in a single "Sequence"

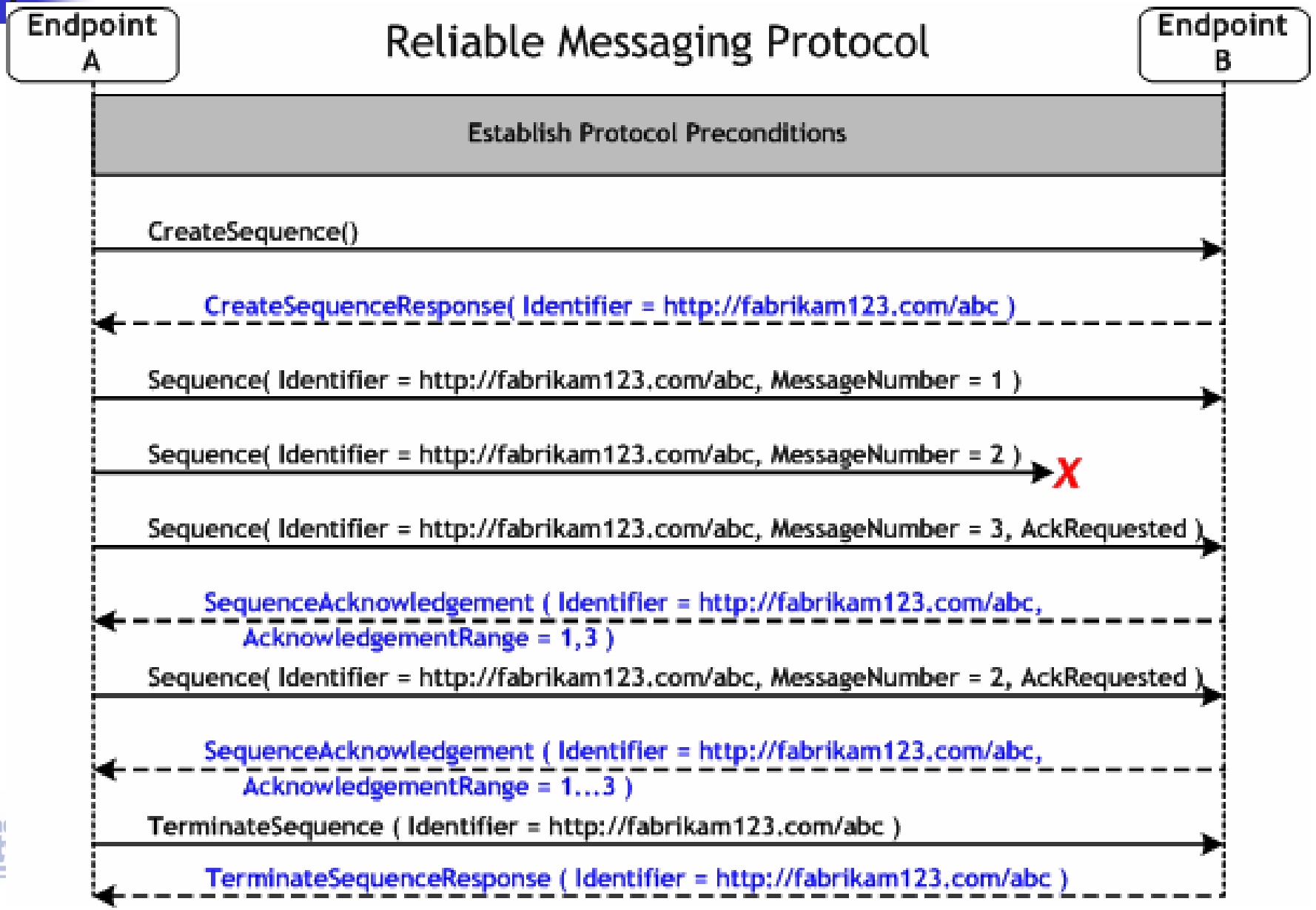
  Think of the sequence as a ***reliability contract*** between a sender and a receiver



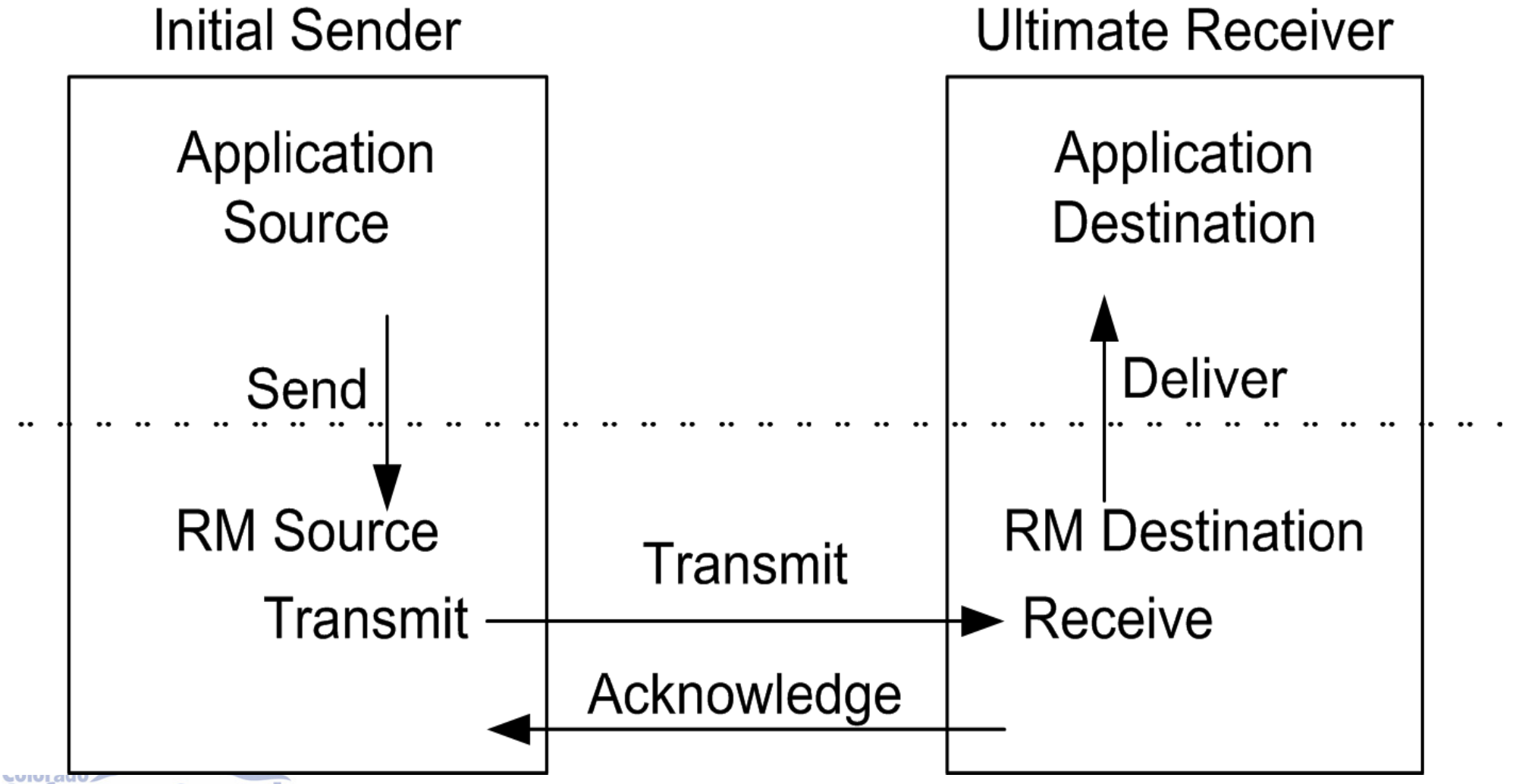
The Core Protocol

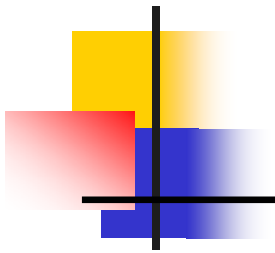
- CreateSequence and CreateSequenceResponse
- Messages allocated to the sequence
- Acknowledgement
- Resend of unacknowledged messages
- TerminateSequence and TerminateSequenceResponse

Simple Example



WS-RM Core Model





Protocol Pre-conditions

- For any single message exchange the RM Source MUST have an endpoint reference that uniquely identifies the RM Destination Endpoint.
- The RM Source MUST have successfully created a Sequence with the RM Destination.
- The RM Source MUST be capable of formulating messages that adhere to the RM Destination's policies.
- If a secure exchange of messages is REQUIRED, then the RM Source and RM Destination MUST have a security context.

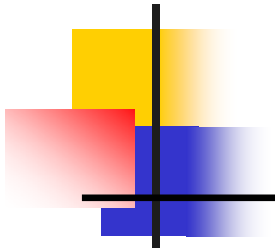
Protocol Invariants

- RMS MUST assign each message a message number beginning at 1 and increasing by 1 for each subsequent message. These numbers MUST be assigned in the same order in which messages are sent by the Application Source.

- Every acknowledgement issued by RMD MUST include the sequence number of every message successfully received and MUST exclude sequence numbers of any messages not yet received.
  - This one is wrong – we're fixing it ☺
  - Why?

Maybe Not Exactly How You Imagine It

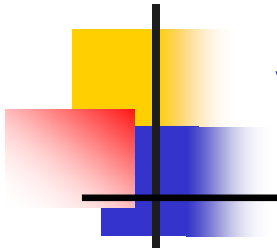
- The protocol ONLY concerns itself with transmission of the messages
- Delivery
  - From the RMD to the application
  - Not specified by this specification
- Could be
  - composed with WS-AT
  - locally transacted
  - Persisted at the RMD and/or RMS
- But that is implementation specific



Submitted Specification

- Explicitly called out the delivery assurances:
  - At least once
  - At most once
  - Exactly once (at least+at most)
  - InOrder

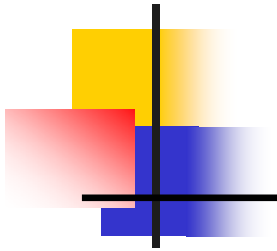
- But the protocol – and on-the-wire behaviour was identical in every case
  - OASIS TC decided to concentrate on the wire protocol
  - Removed that language



Verbs

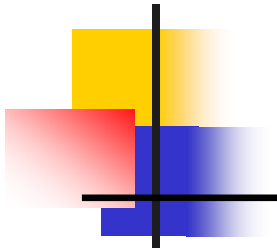
- SOAP Body:
  - CreateSequence/CreateSequenceResponse
  - CloseSequence*/CloseSequenceResponse*
  - TerminateSequence/TerminateSequenceResponse*
- SOAP Header:
  - Sequence
  - AckRequested
  - SequenceAcknowledgement

* Indicates added since submitted to OASIS



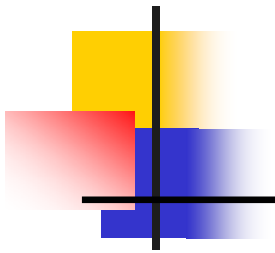
CreateSequence

```
<wsrm:CreateSequence ...>
  <wsrm:AcksTo ...>wsa:EndpointReferenceType</wsrm:AcksTo>
  <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
  <wsrm:Offer ...>
      <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
      <wsrm:Expires ...> xs:duration </wsrm:Expires> ?
      ...
  </wsrm:Offer> ?
  ...
</wsrm:CreateSequence>
```



CreateSequenceResponse

```
<wsrm:CreateSequenceResponse …>
  <wsrm:Identifier …> xs:anyURI </wsrm:Identifier>
  <wsrm:Expires> xs:duration </wsrm:Expires> ?
  <wsrm:AcknowledgementInterval Milliseconds="xs:unsignedLong"  />?
  <wsrm:Accept …>
     <wsrm:AcksTo …> wsa:EndpointReferenceType </wsrm:AcksTo>
  </wsrm:Accept> ?
  …
</wsrm:CreateSequenceResponse>
```



CreateSeq/CSR

- Sent in the body
- AcksTo specifies the place where acknowledgements are sent
- Expires is optional
- Offer is an optional way of initiating a Sequence in the reverse direction
- Can respond with *CreateSequenceRefused* fault

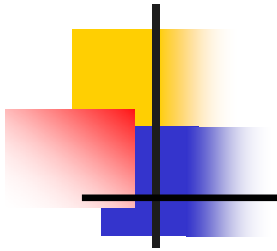
Offer and Accept

- Offered sequence is a way of cutting down the “back-and-forth”
- There is no specific requirement that the Offered sequence is used for responses
  - But usual behaviour

CloseSequence

```
<wsrm:CloseSequence …>
  <wsrm:Identifier …>xs:anyURI</wsrm:Identifier>
  …
</wsrm:CloseSequence>
```

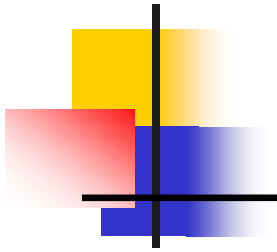
- Sent from the RMS to the RMD
- Added to help close down incomplete sequences
- After a sequence is closed, no new messages are received
- But still responds to AckRequested
  - Allowing the final state of the sequence to be ascertained



TerminateSequence

```
<wsrm:TerminateSequence …>
  <wsrm:Identifier …> xs:anyURI </wsrm:Identifier>
  …
</wsrm:TerminateSequence>
```

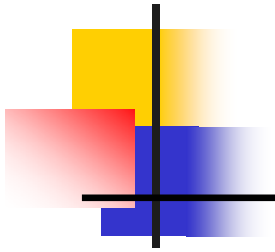
- Sent from the RMS to the RMD - indicates that the sequence will no longer be used
- Normally sent when the sequence is complete – all sent messages have been acknowledged
- Can be sent at any time



TerminateSequenceResponse

```
<wsrm:TerminateSequenceResponse …>
  <wsrm:Identifier …> xs:anyURI </wsrm:Identifier>
  …
</wsrm:TerminateSequenceResponse>
```

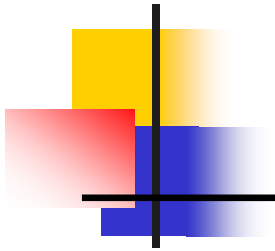
- This was added by the TC to allow the RMS to know that the RMD successfully terminated the sequence



Sequence Header

```
<wsrm:Sequence ...>
   <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
   <wsrm:MessageNumber> wsrm:MessageNumberType
   </wsrm:MessageNumber>
...
</wsrm:Sequence>
```

- The primary header – added to every message that is in a Sequence
- Adds the sequence identifier and the message number



AckRequested

```
<wsrm:AckRequested ...>
  <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
  ...
</wsrm:AckRequested>
```

- Allows a RMS to request an acknowledgement from the RMD
- RMD must reply at once
- Must be a full acknowledgement (no Partial Answer Mode)

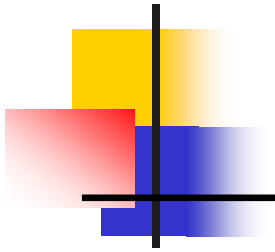
SequenceAcknowledgement

```
<wsrm:SequenceAcknowledgement ...>
      <wsrm:Identifier ...> xs:anyURI </wsrm:Identifier>
[ [ [ <wsrm:AcknowledgementRange ...
            Upper="wsrm:MessageNumberType"
            Lower="wsrm:MessageNumberType"/> +
    | <wsrm:None/> ]
      <wsrm:Final/> ? ]
|     <wsrm:Nack> wsrm:MessageNumberType </wsrm:Nack> + ]
      ...
</wsrm:SequenceAcknowledgement>
```

- A complete set of ranges
  - May be Final – indicating no more messages will be accepted
- Or None (may be Final)
- Or Nacks of individual messages

A Possible Model "Optimistic"

- CS/CSR initiates
- RMS sends messages under the sequence
- Assumes delivery
- Any undelivered messages will be Nacked by the RMD
  - Preferably by PiggyBacking a header on an existing response
- RMS will AckRequested when finished sending – or at a reasonable interval
- Once all messages acked, TerminateSequence

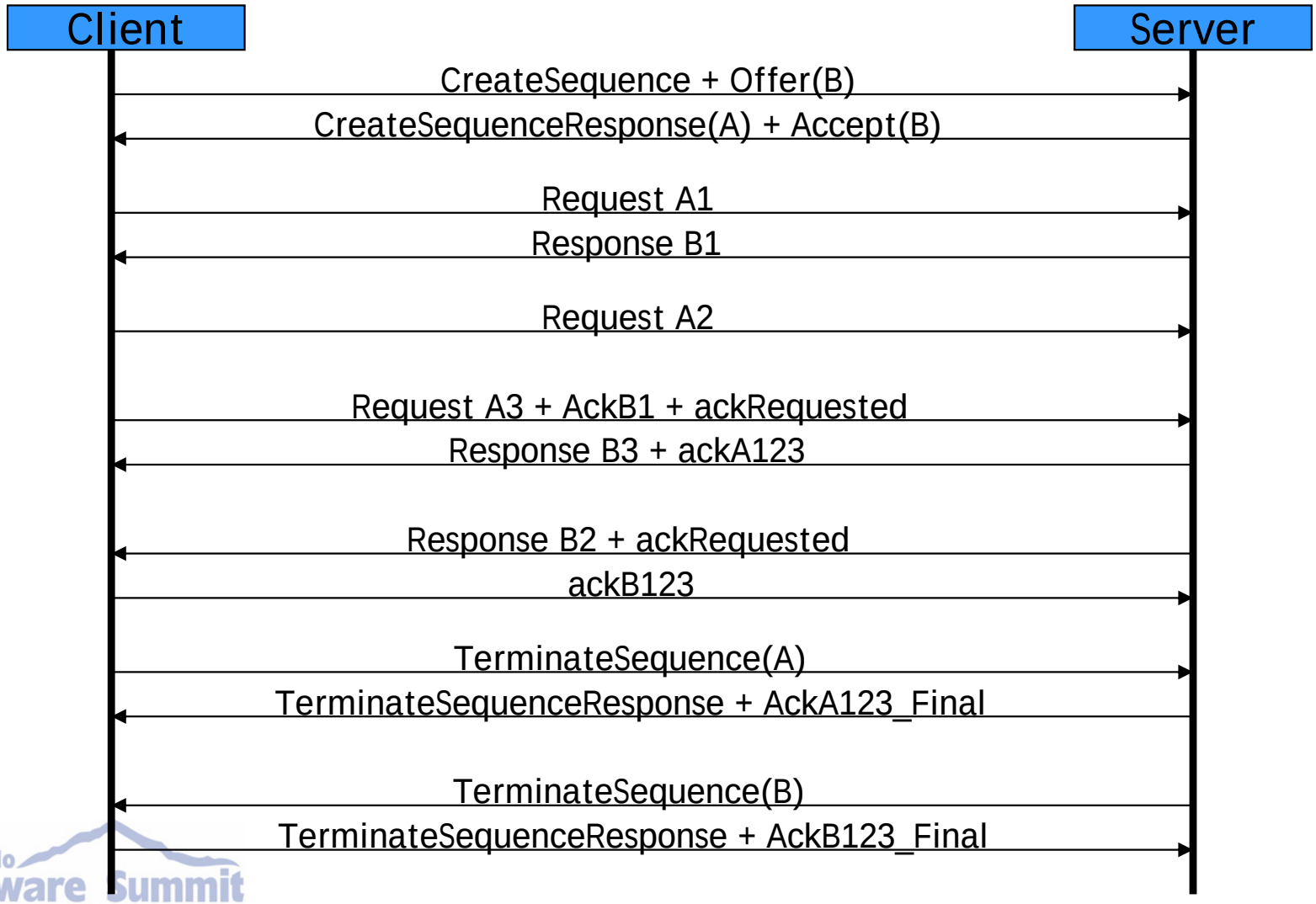


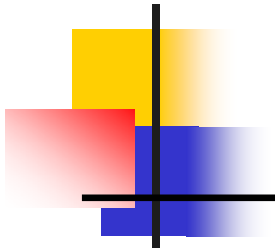
WSRM Policy

<wsrmp:RMAssertion
wsp:Optional=true/>

- The submitted spec had a number of parameters available in policy.
- AcknowledgementInterval was moved to CS
- Dynamic optimization of protocol timing is more effective than static parameters.

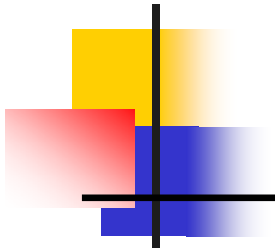
A Two-way Reliable Interaction





Piggybacking

- This is where an acknowledgement is added onto an existing message
  - AcksTo == ReplyTo
- Allows
  - Efficient use of the protocol
  - RMD to send unasked for acknowledgements even in the anonymous HTTP case

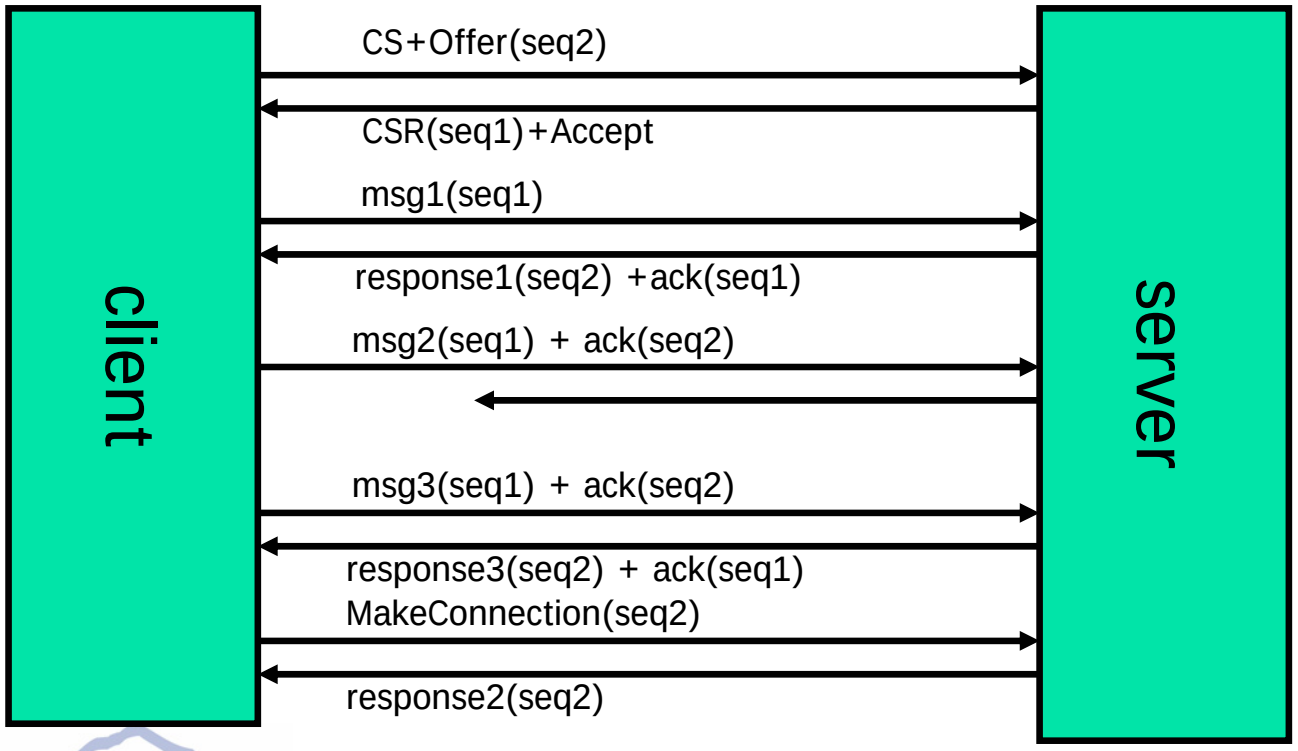


Anonymous Clients

- The protocol supports one-way reliability with anonymous clients
  - May be reliable-in/unreliable-out
  - acksTo will be anonymous
  - Uses piggybacked acks

Anonymous Clients and Two-way

- The WSRM1.1 specification has added a new verb to support this: MakeConnection

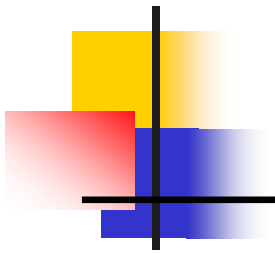


Summary of Changes

- Namespaces!
- General tidy up of definitions
- Removal of LastMessage
- Addition of None, Final, CloseSequence
- Change in highest permissible number
- Simplification of WSRM Policy
- Composition with WSSec and other security
  - Added support for composing with SSL/TLS
- Two-way reliability with an anonymous endpoint
  - MakeConnection

Uptake

- Still gated by lack of a “standard”
- Industry has definitely moved to WSRM spec
- Examples:
  - Canadian government project built a solution on WSRM spec
  - A major Wall Street bank is looking at widespread adoption of WSRM
  - Ford and Daimler Chrysler are planning major implementations
  - Other groups such as RAMP, FIXprotocol, will profile RM



WS-I RSP

- Reliable Secure Profile
  - Proposed by IBM, Ford, DaimlerChrysler
  - Aiming at creating a supplier network based on secure reliable SOAP
  - WSRM 1.1
  - WS-SecureConversation 1.3

RX TC Status and Roadmap

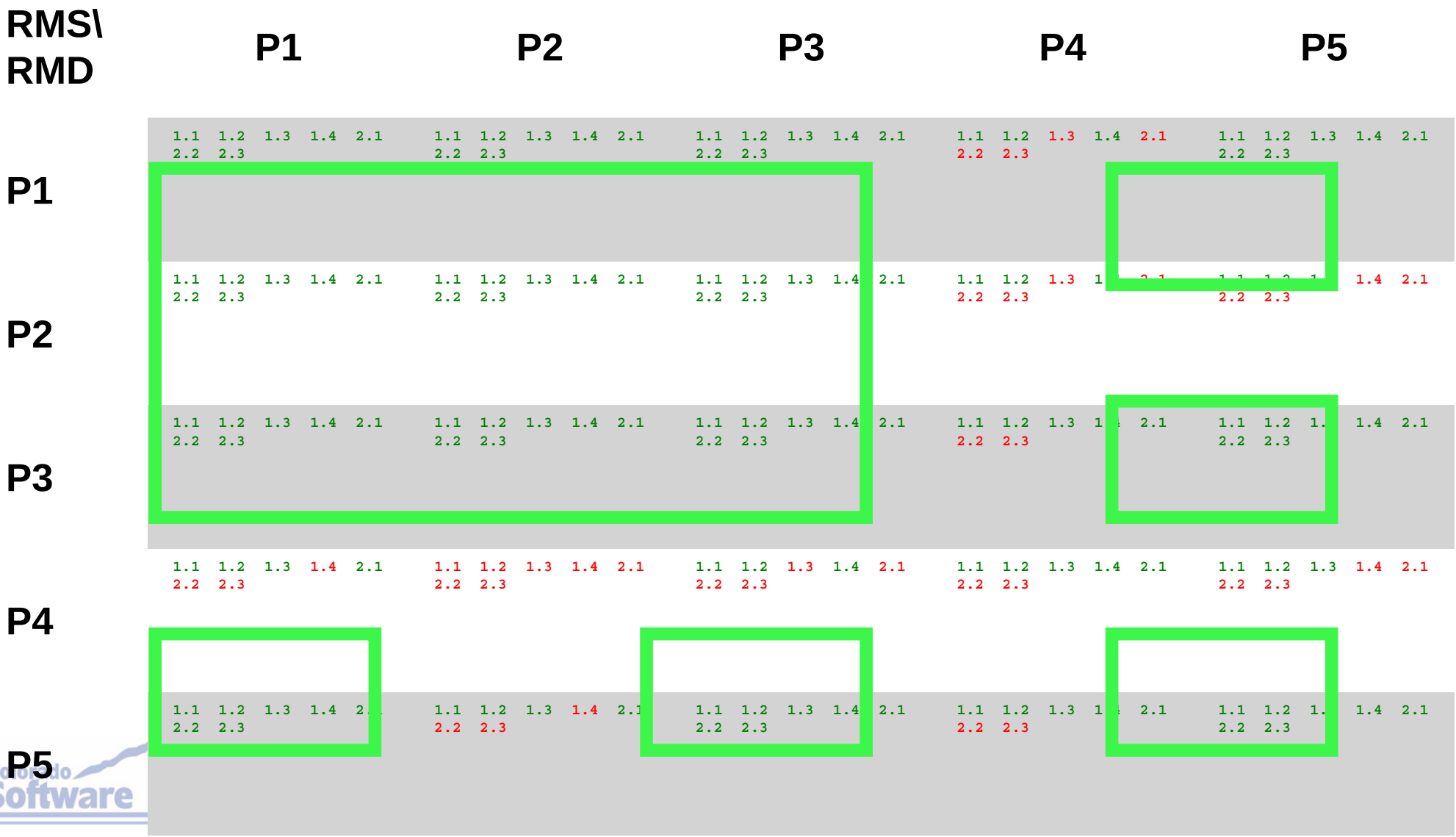
- 60-day Public Review
  - Just finished
- Aiming to have an OASIS standard by end of year

Implementations

- Microsoft Windows Comm Framework
  - WCF aka Indigo
  - Supports the submitted spec
- IBM WebSphere 6.1
  - "IBM intends to make a fully supported Web Services Reliable Messaging solution available for WebSphere Application Server V6.1 early in 2007"
- Apache Sandesha 1.0
  - Supports the submitted spec and CD3 of WSRM 1.1
- WSO2 Tungsten
  - Supported package of Apache Axis2, WSS4J, Sandesha2
- Plus implementations from:
  - Systinet, Oracle, SAP, Sun, BEA

OASIS Interop Results

IBM, MSFT, WSO2, SAP, Oracle



RM *versus* JMS

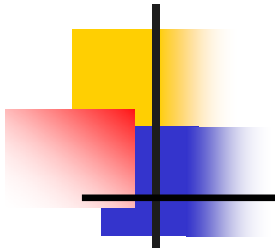
- Firstly, it's like comparing apples with oranges
  - JMS is an API
  - RM is a wire-format
- JMS has no interoperability between different implementations
  - RM is all about interop
- RM has no standard programming model
  - JMS is all about programming models

RM *vs.* SOAP/JMS

- OK, so I'm biased
- But I think RM is much cleaner and simpler
  1. No need to define new entities

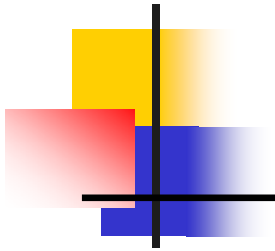
     Queues and Topics are "extras" required for SOAP/JMS, whereas RM uses existing HTTP (or other) URLs
  2. No standard for SOAP/JMS

     There is work going on but still unpublished
  3. Composes more cleanly with SOAP



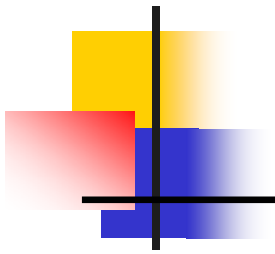
Sandesha

- Sanskrit word meaning “Message”
- Apache implementation of WSRM
  - Sandesha1 = Axis1
    - Not much activity
  - Sandesha2 = Axis2
    - Up to date with the latest spec and participating in the interops



Sandesha2

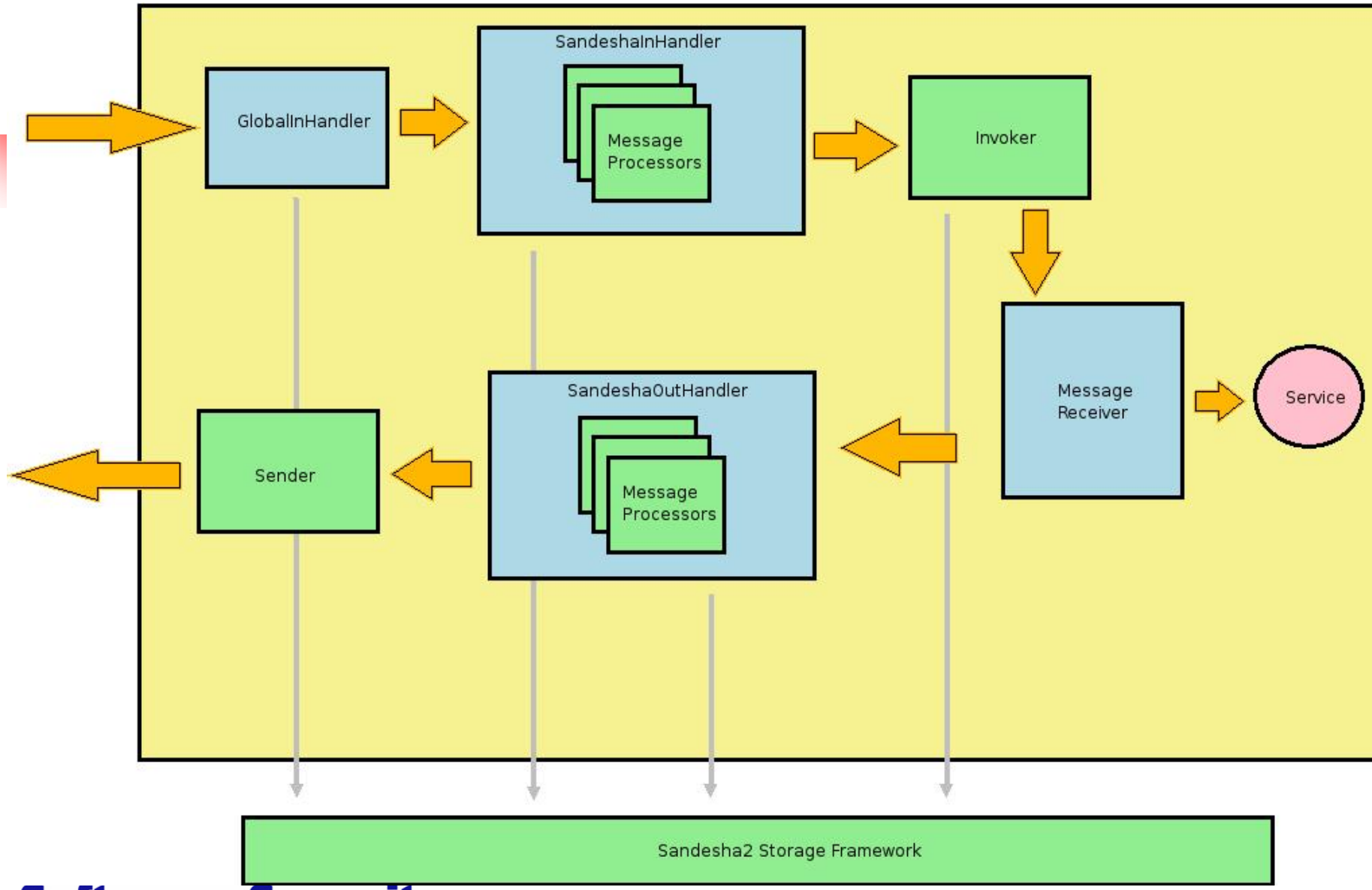
- A reliable messaging module for Axis2
- Full support for WSRM 1.0
- Full support up to the Committee Draft 3 of WSRM 1.1

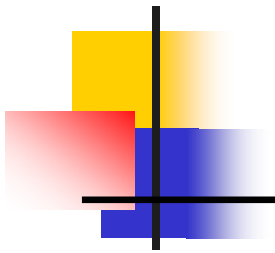


Sandesha2 on Axis2

- When module deployed adds
  - ***Two handlers to the inFlow***
  - ***One handler to the outFlow***
- Can be engaged to the Axis2 engine in various levels
  - ***Globally***
  - ***To a single service***
  - ***To a single operation***

Architecture of Sandesha2



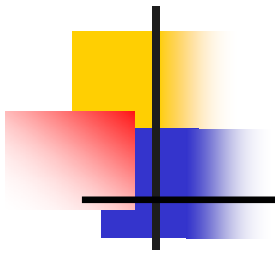


Architecture *(Continued)*

- Handlers
  - ***SandeshaInHandler***
  - ***SandeshaOutHandler***
  - ***SandeshaGlobalInHandler***
- Sender – A thread that keeps sending and resending messages.
- Invoker – A thread that is responsible for doing the ordered invocation of messages.
- Message Processors
  - ***There is a message processor responsible for processing each type of message.***

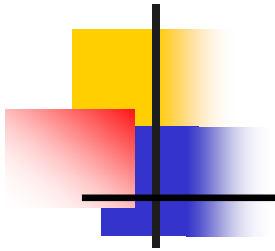
Sandesha2 Storage Framework

- Gives a common interface to store the data of the Sandesha2 system:
  - Defines a set of interfaces and beans
- By providing different implementations of these interfaces Sandesha can support varying levels of durability
  - The rest of the code is unaware of the storage type
- WSO2 Tungsten provides a fully persistent storage manager based on Hibernate
  - Derby as the default database, but pluggable



Adding RM to Your Application

- Three approaches
  - No code
  - A little code
  - Lots of code

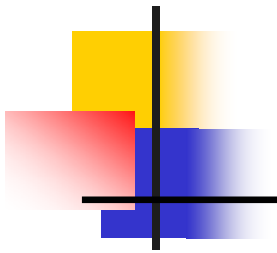


No Code

- Simply enable the module at the client or server side
- Default timeouts and sequence behavior

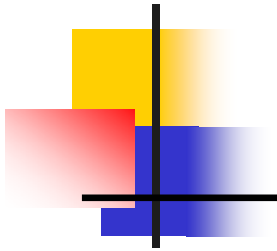
Some Code (at the client side)

- Add context to messages
  - You can specify an “internal sequence id”
    - Could be used to isolate messages from different web-users to different sequences
  - For RM1.0 you can specify “LastMessage”



Full API

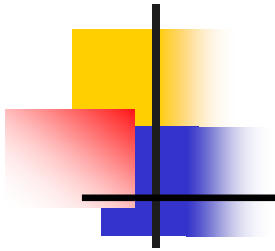
- See which sequences are in play
- Find out the status of messages
- Register to be notified of changes
- Create and terminate sequences



SandeshaClient

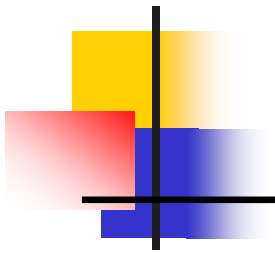
- Provides a set of functions, to control the Sandesha2 client side
- Not all clients may require these
- Mostly for advanced users

```
getOutgoingSequenceReport (ServiceClient serviceClient):SequenceReport

getIncomingSequenceReports (ServiceClient serviceClient):SequenceReport

createSequence (ServiceClient serviceClient, boolean offer)

terminateSequence (ServiceClient serviceClient)

waitUntilSequenceCompleted(ServiceClient serviceClient)

sendAckRequest(ServiceClient serviceClient)
```



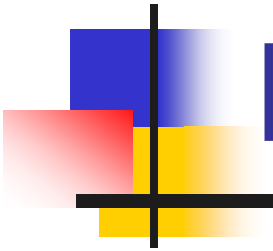
Sandesha2 Listener Feature

- Clients can register a Sandesha2 Listener
- Get notified when specific events happen
  - ***When a RM message receives a fault***
  - ***When a sequence timeouts***



Interoperability with Microsoft

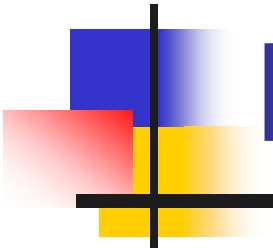
- Microsoft WCF (used to be known as Indigo)
- Fully interoperates
  - Tested at multiple “plug-fests”



Demo

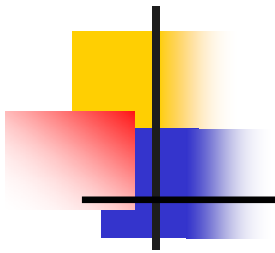
Sandesha in Other Languages

- Sandesha2/C is (almost) working
- Since Axis2/C is used by PHP, Axis2/PHP also gets RM support
  - Simple as “engageModule(“sandesha2”)!



Demo2

PHP and Java reliably sending messages



Why Use WSRM?

1) It's a standard (almost!)
2) Interoperable approach
3) It's free – either OSS or part of Windows
4) Easy to add to existing Web services
5) Support from all the major IT vendors (and a few of the rising stars ☺)
6) Its Grrrrrrrrreat!

Resources

- WSRX TC Homepage
  - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-rx
- Submitted specification
  - http://www.oasis-open.org/committees/download.php/16889/
- Cover pages Reliable Messaging
  - http://xml.coverpages.org/reliableMessaging.html
- Apache Sandesha
  - http://ws.apache.org/sandesha2/