

# Navigating WS-(death?)\*

# Our Starting Point

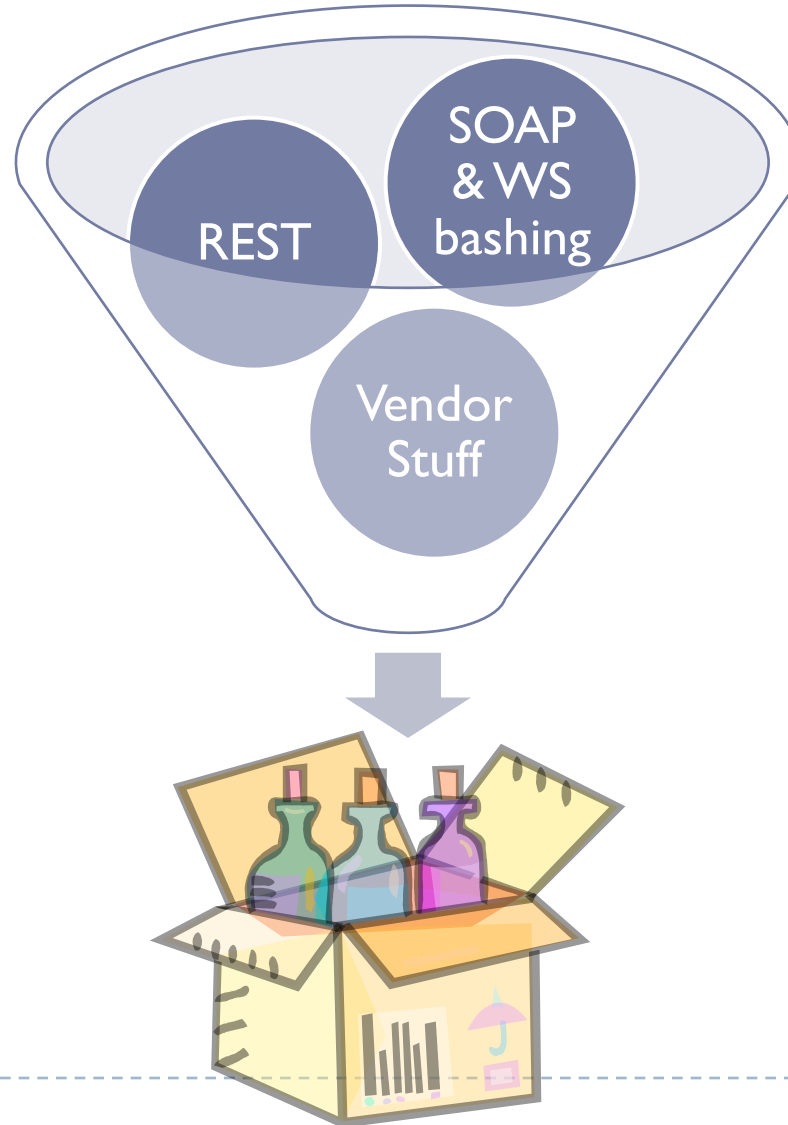
---

- ▶ Message Oriented
- ▶ Transport Agnostic
- ▶ SOAP
- ▶ If you want to question these points, lets grab a beer later!



# What this talk is not about

---



# Goals

---

- ▶ Discover the *major* specifications associated with SOAP
- ▶ Discover the *motivations* for these specifications
- ▶ Discover how these specifications can be *composed*
- ▶ Answer:
  - ▶ When should I use WS-Foo?
  - ▶ What platforms and toolkits interoperate with WS-Foo?
  - ▶ Where is WS-Foo going in the future?



# The Major Specifications

---

- ▶ WS Addressing
- ▶ WS Policy & Friends
- ▶ WS Reliable Messaging & Friends
- ▶ WS SX
  - ▶ WS Security
  - ▶ WS Secure Conversation
  - ▶ WS Trust



## Some of the “less major” specifications

---

- ▶ **WS-AtomicTransactions**
- ▶ **WS-BusinessActivity**
- ▶ **WS-Coordination**
- ▶ **WS-DistributedManagement**
- ▶ **WS-Eventing**
- ▶ **WS-MetadataExchange**
- ▶ **WS-Notification**
- ▶ **WS-Transfer**
- ▶ **Others...**

# WS-Addressing

# WS-Addressing

---

- ▶ SOAP works with any transport
- ▶ If there is no URL, how do we address services?
  - ▶ Example: JMS only has queues and topics
- ▶ How do we address multiple services hosted at the same endpoint?
- ▶ How do we tell the endpoint where to send replies?
- ▶ And faults?
- ▶ How do we reference a specific message?





# Concepts

---

- ▶ **Action:** the action to be taken by the message
- ▶ **Message ID:** Unique id which makes it possible to reference the message
- ▶ **To:** A URI which represents the server being addressed
- ▶ **ReplyTo:** EPR telling the server where to send replies
- ▶ **FaultTo:** EPR telling the server where to send faults



# Endpoint Reference

---

- ▶ An endpoint reference is the equivalent of URIs for web services
- ▶ Includes:
  - ▶ Address
  - ▶ PortType
  - ▶ ReferenceParameters
  - ▶ ServiceName
- ▶ Only address is required (and typically the only one used)



# Example

---

```
<S:Envelope ...>
  <S:Header>
    <wsa:MessageID>http://example.com/6B29FC40-CA47-1067-
    B31D-00DD010662DA</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>
        http://example.com/business/client1
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>
      http://example.com/fabrikam/Purchasing
    </wsa:To>
    <wsa:Action>
      http://example.com/fabrikam/SubmitPO
    </wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

# WS-Addressing Versions

---

- ▶ **2004-08**

- ▶ First version to pick up real adoption.
- ▶ Used in WS-ReliableMessaging 1.0

- ▶ **2005-08:**

- ▶ In most major frameworks: XFire, CXF, Axis, WCF

- ▶ **1.0**

- ▶ Recently standardized



# When should I use it?

---

- ▶ If you're addressing multiple services on the same endpoint
- ▶ If you're using non addressable transports
- ▶ If you're using another specification which relies on it (i.e. WS-RM)



# WSDL binding

---

- ▶ WS-Addressing defines a binding to put addressing information inside the WSDL
- ▶ Supported as part of JAX-WS 2.1 and WC



# WSDL Binding

---

```
<binding ...>  
  <wsaw:UsingAddressing  
    wsdl:required="true" />  
  <operation>  
    ...  
  </operation>  
</binding>
```



# WSDL Binding

---

```
<portType name="customerService">
  <operation name="getCustomer">
    <input message="tns:getCustomer"
      wsaw:Action="http://foo.com/getCustomer" />
    <output message="tns:getCustomerResponse"
      wsaw:Action="http://foo.com/getCustomerResponse" />
  </operation>
</portType>
```





# The Matrix

---

Version	Axis 2	CXF	Glassfish	JBossWS	.NET/WS E 2.0	.NET/WS E 3.0	XFire	WCF
03/04					X			
08/04	X	X	X	X		X	X	X
08/05	X	X					X	
05/06 (1.0)	X	X	X	X				X



# WS-ReliableMessaging & Friends

# WS-ReliableMessaging

---

- ▶ **Not all transports are reliable**
  - ▶ Notably HTTP
- ▶ **How do we ensure that:**
  - ▶ Each message was received?
  - ▶ In order?
  - ▶ And only once?

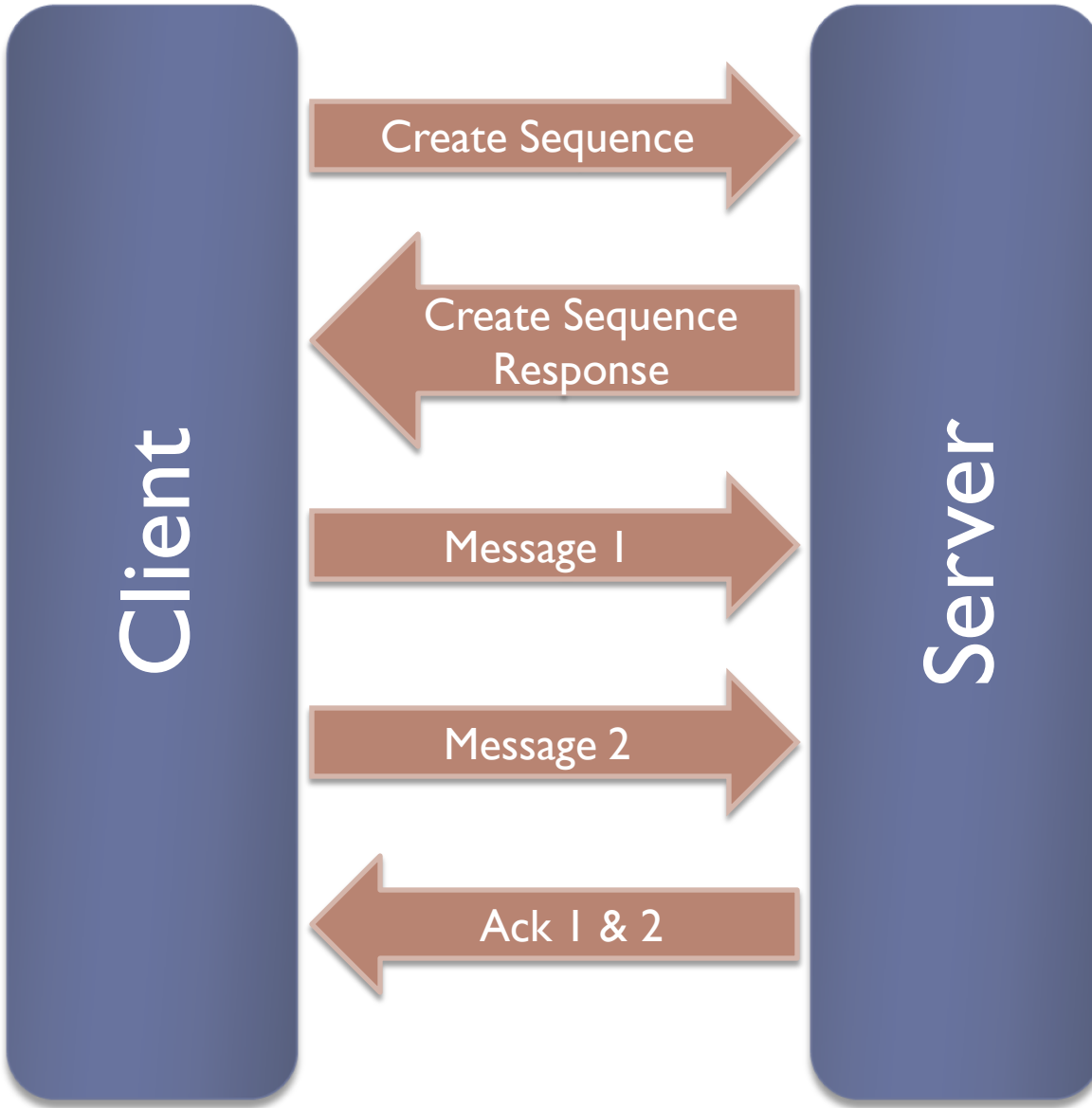


# Main Concepts

---

- ▶ A series of message exchanges between a client and server is called a sequence
- ▶ *CreateSequence* establishes a sequence
- ▶ Each message contains a *SequenceId*
- ▶ Every so often a *SequenceAcknowledgement* is sent
- ▶ *TerminateSequence* ends the sequence





# Sequence Creation

---

```
<s:Envelope>  
  <S:Body>  
    <wsrm:CreateSequence>  
      <wsrm:AcksTo>  
        <wsa:Address>  
http://Business456.com/serviceA/789  
        </wsa:Address>  
      </wsrm:AcksTo>  
    </wsrm:CreateSequence>  
  </S:Body>  
</S:Envelope>
```

# CreateSequenceResponse

---

**<S:Body>**

**<wsrm:CreateSequenceResponse>**

**<wsrm:Identifier>**

**http://Business456.com/RM/ABC**

**</wsrm:Identifier>**

**</wsrm:CreateSequenceResponse>**

**</S:Body>**

# Normal Message Exchange

---

```
<s:Envelope>
<S:Header>
  <wsa:MessageID>...</wsa:MessageID>
  <wsa:To>http://example.com/serviceB/123</wsa:To>
  <wsa:From>
    <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
  </wsa:From>
  <wsa:Action>http://example.com/serviceB/123/request</wsa:Action>
  <wsrm:Sequence>
    <wsrm:Identifier>
      http://Business456.com/RM/ABC
    </wsrm:Identifier>
    <wsrm:MessageNumber>1</wsrm:MessageNumber>
  </wsrm:Sequence>
</S:Header>
<S:Body>
  <!-- Some Application Data -->
</S:Body>
</S:Envelope>
```



# Message Acknowledgement

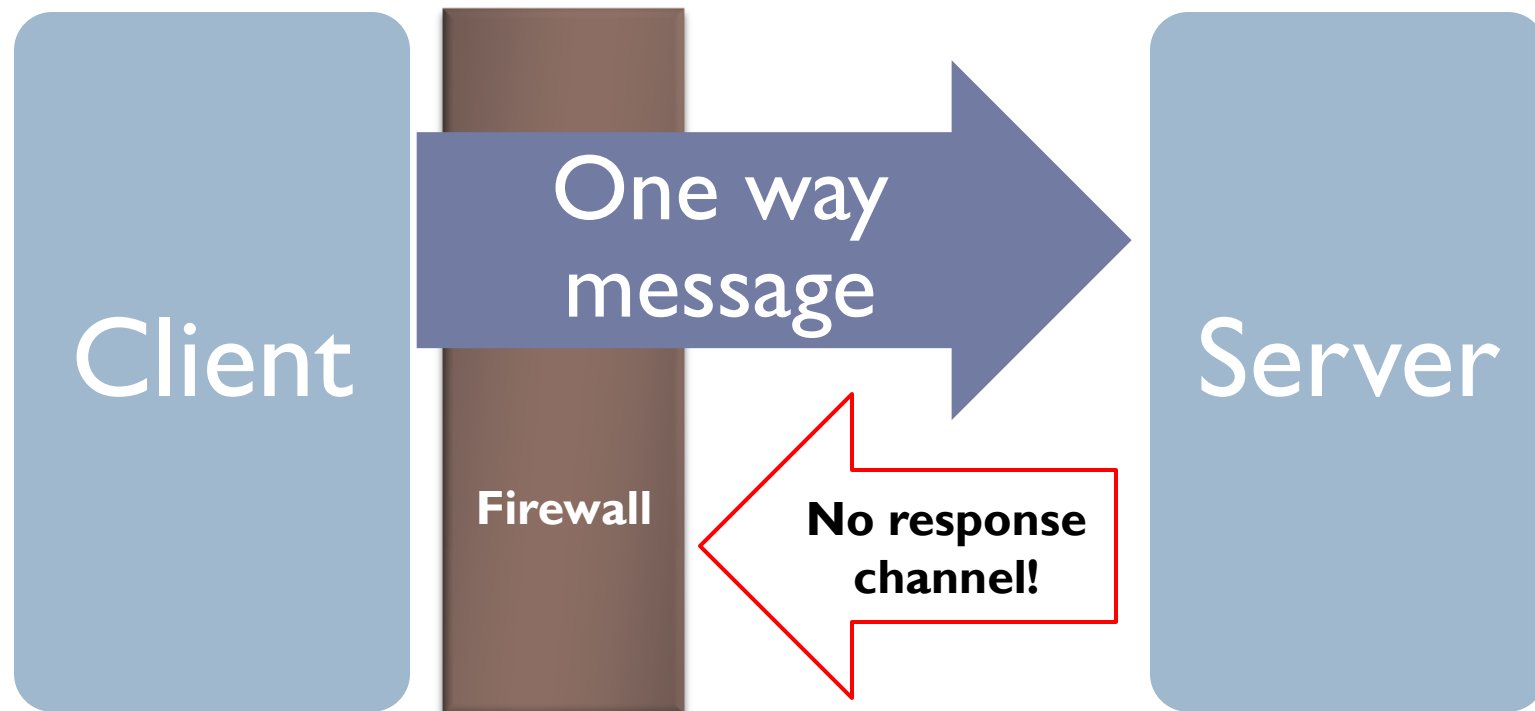
---

```
<S:Envelope>
<S:Header>
  <wsa:MessageID>...</wsa:MessageID>
  <wsa:To>http://Business456.com/serviceA/789</wsa:To>
  <wsa:From>
    <wsa:Address>http://example.com/serviceB/123</wsa:Address>
  </wsa:From>
  <wsa:Action>
    http://docs.oasis-open.org/ws-
    rx/wsrn/200608/SequenceAcknowledgement
  </wsa:Action>
  <wsrm:SequenceAcknowledgement>
    <wsrm:Identifier>
      http://Business456.com/RM/ABC
    </wsrm:Identifier>
    <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
    <wsrm:AcknowledgementRange Upper="3" Lower="3"/>
  </wsrm:SequenceAcknowledgement>
</S:Header>
<S:Body/>
</S:Envelope>
```

# Firewall issues

---

- ▶ Server has to send an acknowledgement and lost messages back to the client
- ▶ What if there is a firewall?



# Firewall Issues

---

- ▶ SequenceAcknowledgement can be piggybacked on one way synchronous response
  - ▶ Even though that's really against the BasicProfile...
- ▶ WS-RM 1.1 introduces a MakeConnection operation
  - ▶ Client sends MakeConnection to the server
  - ▶ Server can respond with any messages it wants to send



# Order and delivery assurances

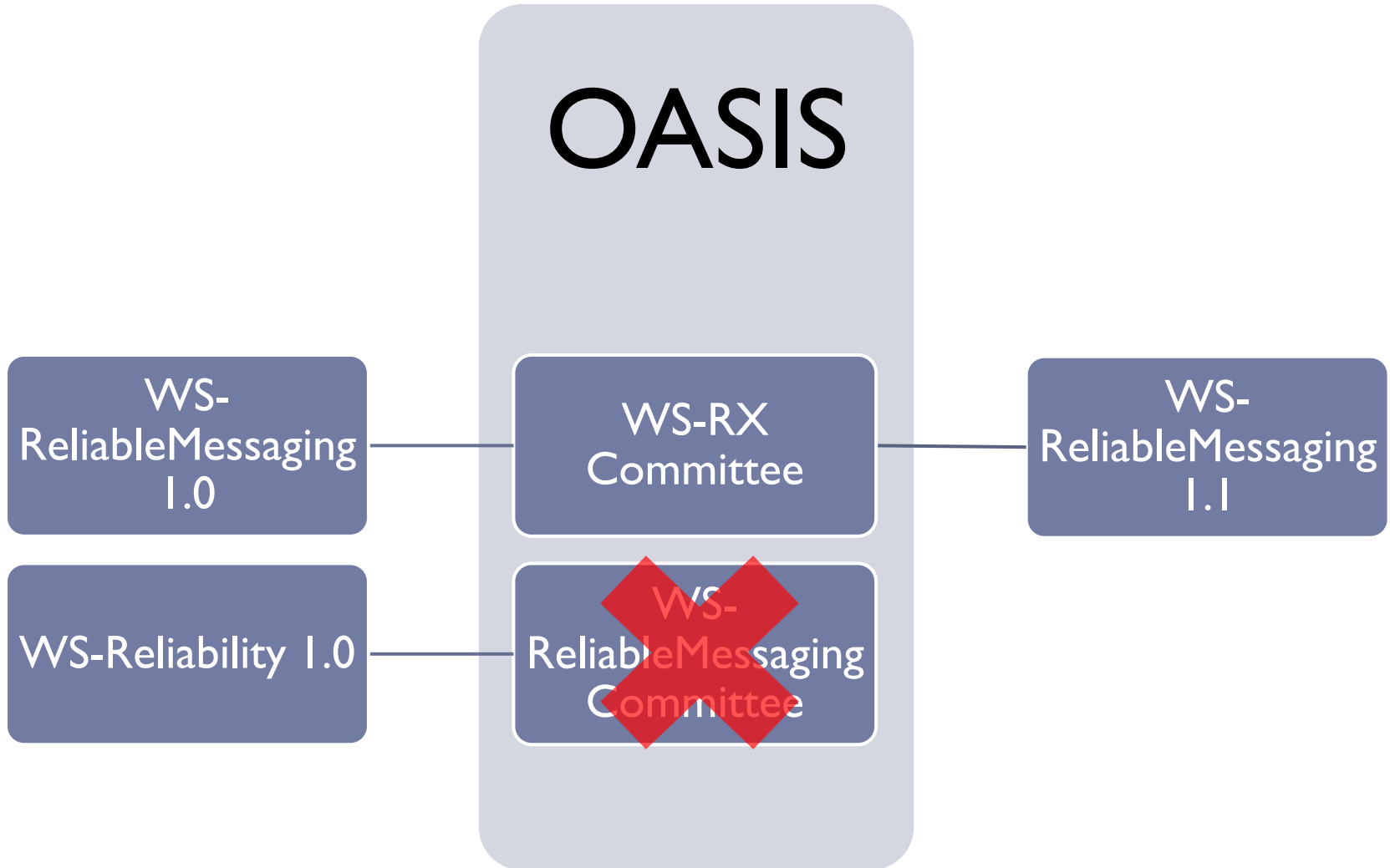
---

- ▶ WS-RM 1.1 removes in order and exactly once delivery requirements from the spec
- ▶ These are really the responsibility of your WS-RM implementation
- ▶ There are no durability assurances from provider to provider.



# WS-RM Roadmap

---



# When should I use WS-RM?

---

- ▶ Need delivery assurances over an unreliable protocol (HTTP)
- ▶ Reliability is not built into the application



# The Matrix

Version	Axis 2	CXF	Glassfish	JBossWS	.NET/W SE 2.0	.NET/W SE 3.0	WCF	Systinet
WS-RM 1.0	X	X	X	X			X	X
WS-RM 1.1 (not final)	X							





WS-Policy



# WS-Policy

---

- ▶ If my service uses WS-ReliableMessaging or WS-Security or MTOM or... how will consumers know?
- ▶ Out of band communication
- ▶ Or WS-Policy...



# What is WS-Policy

---

“WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. WS-Policy defines a framework and a model for the expression of these properties as policies.”



# Example

---

```
<wsp:Policy
  xmlns:sp="http://../securitypolicy"
  xmlns:wsp="http://../policy">
  <wsp:ExactlyOne
    <sp:Basic256Rsa15 />
    <sp:TripleDesRsa15 />
  </wsp:ExactlyOne>
</wsp:Policy>
```

# What kind of policies are there?

---

- ▶ WS-ReliableMessaging
- ▶ WS-Security (includes HTTP transport related assertions)
- ▶ MTOM
- ▶ Addressing (in development)

## WS-RM Example

---

```
<wsp:Policy wsu:Id="RmPolicy">
  <rmp:RMAssertion>
    <rmp:InactivityTimeout
      Milliseconds="600000" />
    <rmp:BaseRetransmissionInterval
      Milliseconds="3000" />
    <rmp:ExponentialBackoff />
    <rmp:AcknowledgementInterval
      Milliseconds="200" />
  </rmp:RMAssertion>
</wsp:Policy>
```

# What frameworks support WS-Policy?

---

Spec	Axis2	CXF	Glassfish	.NET WSE 3.x	.NET WCF	Systinet
1.2	X	X	X	X	X	X
1.5	X	X				
MTOM	X	X				
WS-RM	X	X				X
SecurityPolicy	X		X	X	X	X?



# A segue about public key cryptography

The adventures of Alice, Bob and Eve

# Public Key Cryptography

---

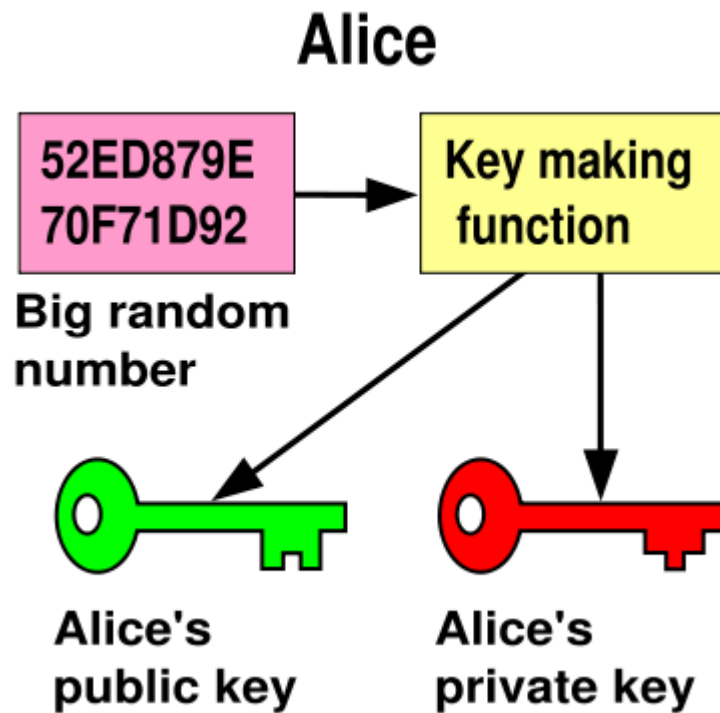
- ▶ Encryption and Signing are done with public and private keys
- ▶ Public key is advertised to the world
- ▶ Private key is your **SECRET**
- ▶ Asymmetric cryptography
  - ▶ Slower than symmetric where there is a shared key





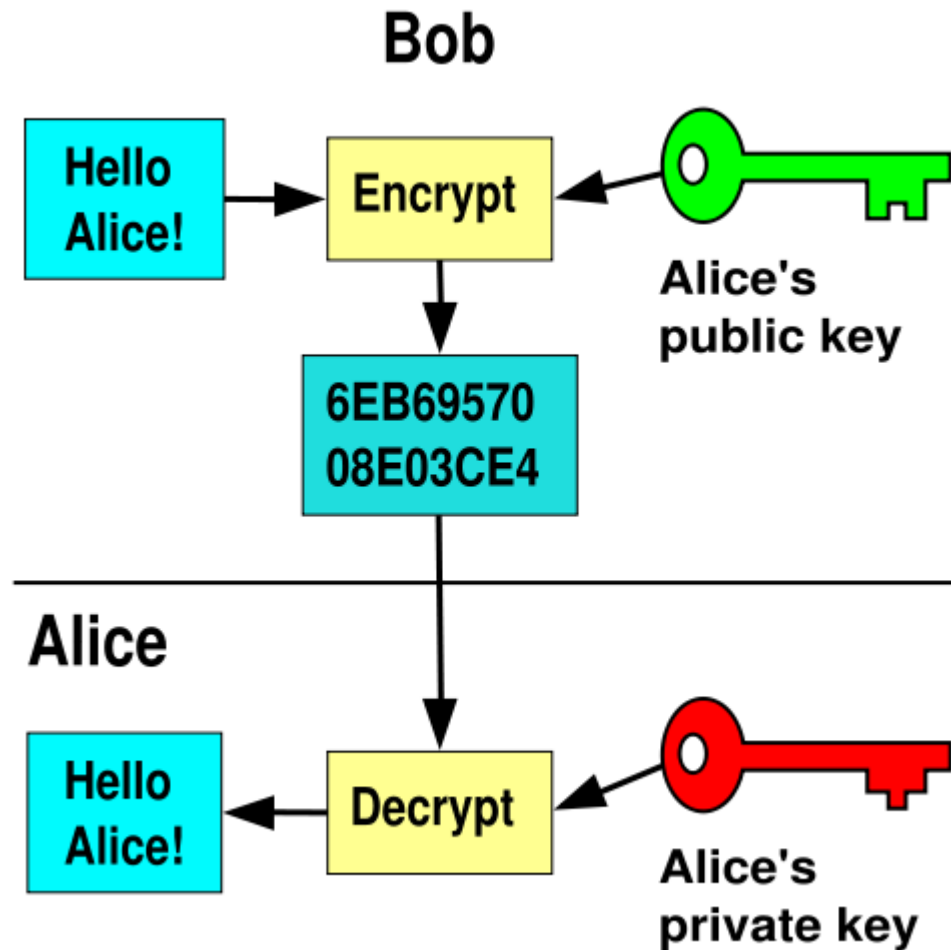
# Key Creation

---



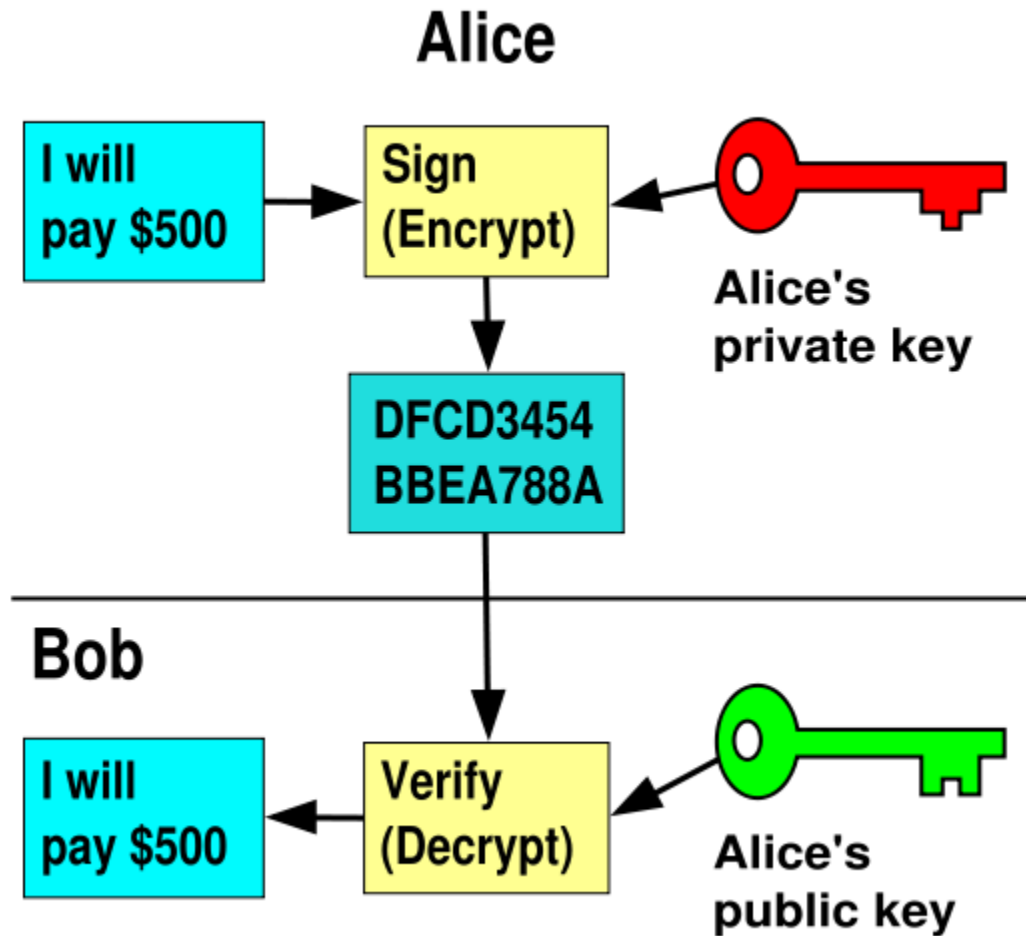
# Public Key Encryption

---



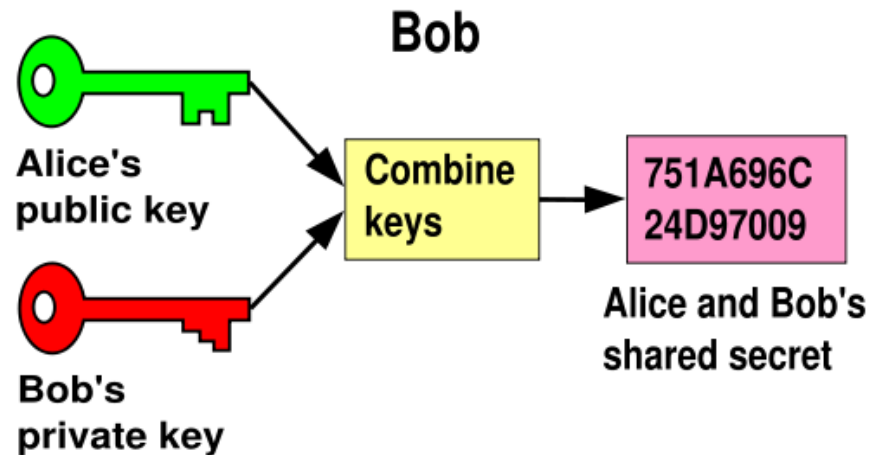
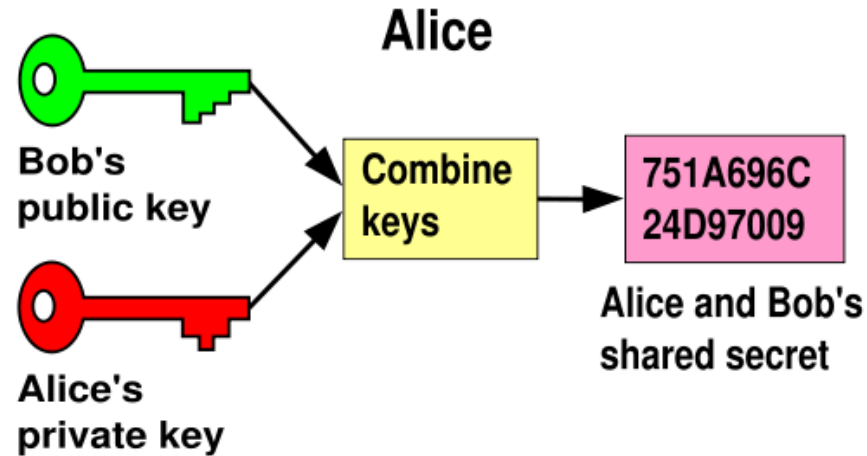
# Public Key Signature

---



# Shared Secrets

---





# WS-Security

# WS-Security

---

- ▶ Includes mechanisms for
  - ▶ Encrypting messages
  - ▶ Signing messages
  - ▶ Setting expiration dates for messages
  - ▶ Sending authentication tokens
- ▶ Builds heavily on the XML Signature and Encryption publications



# Why?

---

- ▶ Sometimes we want message level security
  - ▶ Intermediaries
  - ▶ Multiple readers
- ▶ Need a standard way to exchange a variety of security token types



# Security tokens

---

- ▶ Defines an abstract way to represent security tokens:
  - ▶ UsernameToken
  - ▶ BinarySecurityToken (X.509, Kerberos)
  - ▶ Other XML tokens - SAML
- ▶ UsernameToken:
  - ▶ Support both a password digest and clear text
  - ▶ Clear text should only be used if the transport is secure and there are no intermediaries



## Example: UsernameToken Header

---

```
<wsse:Security xmlns:wsse="...">  
  <wsse:UsernameToken Id="MyID">  
    <wsse:Username>Zoe</wsse:Username>  
    <wsse:Password>pass</wsse:Password>  
  </wsse:UsernameToken>  
</wsse:Security>
```



## Example: BinarySecurityToken Header

---

```
<wsse:BinarySecurityToken
  ValueType="...#X509v3"
  EncodingType="...#Base64Binary"
  wsu:Id="X509Token">
  MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
</wsse:BinarySecurityToken>
```



# Signature

---

- ▶ Various parts of the SOAP Body can be signed
- ▶ The signatures reside in the SOAP Header
- ▶ A signature references a message part via a `wsu:Id` attribute



# Canonicalization

---

- ▶ Before we can sign a document, we must agree on how that document is represented
- ▶ If the xml attributes are in a different order on either side, the signature value will differ
- ▶ We must *canonicalize* the document to avoid these problems.



# A digital signature part 1

---

```
<Envelope>
  <Header>
    <Signature>...</Signature>
    <BinarySecurityToken
      ValueType="...#X509v3"
      EncodingType="...#Base64Binary"
      wsu:Id="X509Token">
      MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
    </BinarySecurityToken>
  </Header>
  <Body wsu:Id="myBody">
    <FooBar>

    ...
  </FooBar>
</Body>
</Envelope>
```

# A digital signature part 2

---

```
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm=
      "http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod Algorithm=
      "http://www.w3.org/2000/09/xmlsig#rsa-sha1" />
    <ds:Reference URI="#myBody">
      <ds:Transforms>
        <ds:Transform Algorithm=
          "http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Transforms>
      <ds:DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmlsig#sha1" />
      <ds:DigestValue>EULddyts01...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
```

...

## A digital signature part 3

---

...

```
<ds:SignatureValue>  
  BL8jdfToEb11/vXcMZNNjPOV...  
</ds:SignatureValue>  
<ds:KeyInfo>  
  <wsse:SecurityTokenReference>  
    <wsse:Reference URI="#X509Token"/>  
  </wsse:SecurityTokenReference>  
</ds:KeyInfo>  
</ds:Signature>
```



# Encryption

---

- ▶ Uses XML-Encryption standard to encrypt various parts of the message
- ▶ Encrypted data can use a key that is:
  1. Exchanged out of band
  2. Inside the message (Symmetric)





# Encryption

---

```
<Envelope>
  <Header>
    <Signature>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#bodyID" />
      </xenc:ReferenceList>
    </Signature>
  </Header>
  <Body>
    <EncryptedData Id="bodyID">
      <ds:KeyInfo>
        <ds:KeyName>CN=Hiroshi Maruyama,
          C=JP</ds:KeyName>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...</xenc:CipherValue>
      </xenc:CipherData>
    </EncryptedData>
  </Body>
</Envelope>
```

# Timestamps and Message Expiration

---

- ▶ Need a way to say that a message is only valid up to a certain time
- ▶ Prevents replay attacks to some extent



# Timestamp example

---

```
<Envelope>  
<Header>  
  <wsse:Security>  
    <wsu:Timestamp wsu:Id="timestamp">  
      <wsu:Created>  
        2001-09-13T08:42:00Z  
      </wsu:Created>  
      <wsu:Expires>  
        2001-10-13T09:00:00Z  
      </wsu:Expires>  
    </wsu:Timestamp>  
    ...  
  </wsse:Security>  
</Header>  
...  
</Envelope>
```

# Who supports WS-Security

---

- ▶ **Better question: who doesn't?**
  - ▶ Dynamic languages...



# Where WS-Security falls short

---

- ▶ It depends on public key cryptography which is slow
- ▶ There is no way to establish trust relationships
- ▶ Out of band communication is required unless you're trusting all certificates from a specific authority





WS-Trust

# What is it?

---

- ▶ Defines a Security Token Service
- ▶ A way to broke trust relationships through the exchange of security tokens
  - ▶ Trust must still be bootstrapped out of band.
- ▶ Issue, renew, validate, cancel and challenge security tokens
- ▶ The building block of WS-SecureConversation



## Problem #1: Token is not understood

---

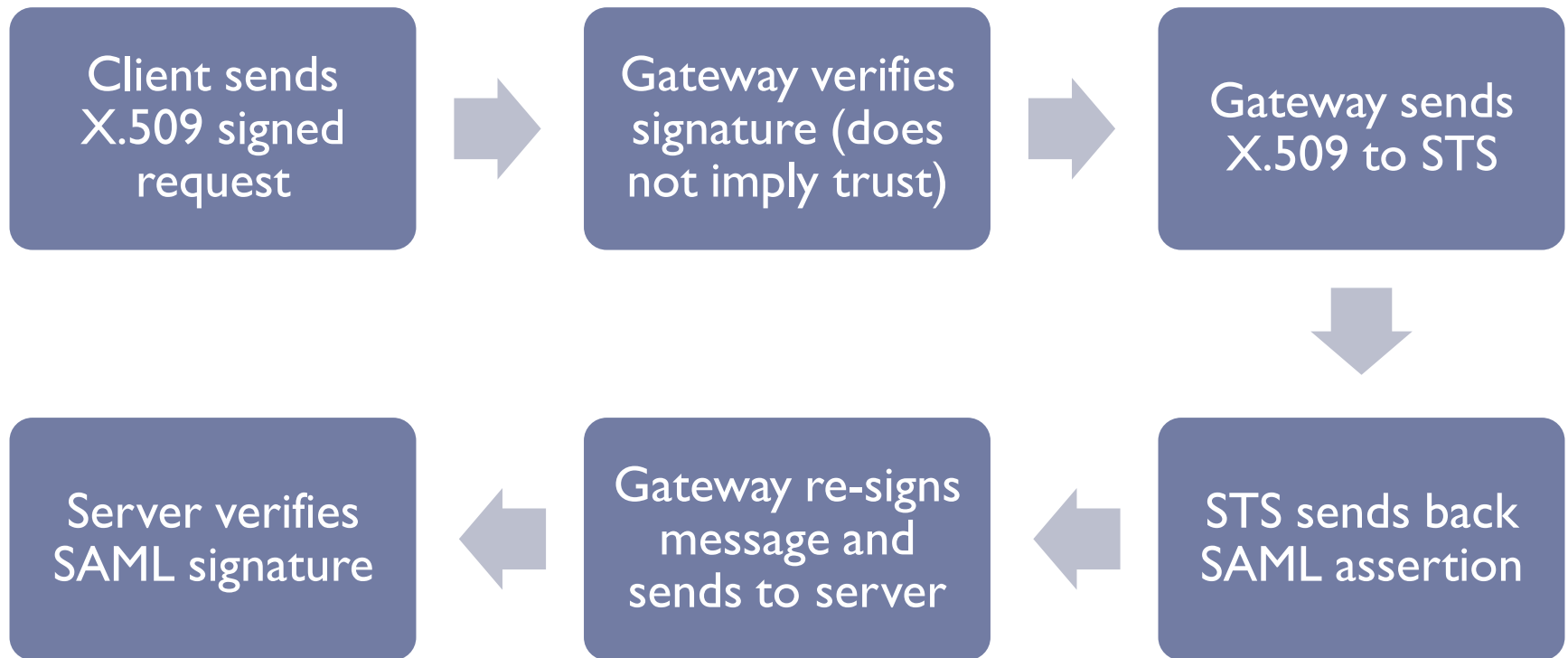
- ▶ If an endpoint does not understand a particular token, WS-Trust allows the endpoint to exchange that token type for another
- ▶ Example: Client sends X.509 certificate, server expects SAML





# Exchanging X.509 certificate for SAML

---



# Example Request

---

```
<soap:Body>
  <wstrust:RequestSecurityToken>
    <wstrust:TokenType>SAML</TokenType>
    <wstrust:RequestType>
      ReqExchange
    </RequestType>
    <wstrust:OnBehalfOf>
      <ws:BinarySecurityToken
        id="originaltoken"
        ValueType="X.509"
        sdfOIDFKLSoidfsdflk ...
      </ws:BinarySecurityToken>
    </wstrust:OnBehalfOf>
  </wstrust:RequestSecurityToken>
</soap:Body>
```

# Example Response

---

```
<soap:Body>
  <wstrust:RequestSecurityTokenResponse>
    <wstrust:TokenType>SAML</TokenType>
    <wstrust:RequestedSecurityToken>
      <saml:Assertion>
        ...
      </saml:Assertion>
    </wstrust:RequestedSecurityToken>
  </wstrust:RequestSecurityTokenResponse>
</soap:Body>
```

## Problem #2: Token is untrusted

---

- ▶ If A trusts B and B trusts C, does A trust C?
- ▶ WS-Trust server can store and manage trust relationships for you



## Problem #3: How do I issue new tokens?

---

- ▶ What if we don't want to use asymmetric cryptography?
- ▶ What if we want to create a shared secret for symmetric cryptography?
- ▶ WS-Trust allows issuance of new tokens



# WS-SecureConversation

# Why?

---

## ▶ Problem:

- ▶ WS-Security is inherently slow as it revolves around public key cryptography. Symmetric cryptography allows us to speed things up
- ▶ No way to reference established security sessions



# Security Contexts

---

- ▶ Refers to an established authentication state and negotiated keys
- ▶ A *SecurityTokenContext* is the on-the-wire representation of this state





# A digital signature with WS-SC

---

```
<SecurityTokenContext wsu:Id="SomeID">
  <Identifier>uuid:...</Identifier>
</SecurityTokenContext>
<ds:Signature>
  <ds:SignatureValue>
    BL8jdfToEb11/vXcMZNNjPOV...
  </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#SomeID"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
```

## What does this give us

---

- ▶ Using WS-Trust we can issue a new security token based on a shared secret
- ▶ This token can be used to create symmetrically encrypted messages – which is much faster
- ▶ Also allows us to create security sessions





# Takeaways

# Interoperability

---

- ▶ Java & .NET exhibit strong interoperability for the major specifications
- ▶ Most have been battle tested for a while



# Dynamic Languages

---

- ▶ There are no dynamic languages which have open source WS-\* implementations at the moment
- ▶ Some movement by the Axis2 community to provide a C version for PHP, Ruby, etc.
- ▶ However – there is no love in general from the dynamic language community for WS-\*



## WS-\* Thoughts

---

- ▶ I don't see equivalent security solutions elsewhere in the “Just HTTP” world
  - ▶ Might be one of the killer applications of WS-\*
- ▶ Message Oriented + Transport Neutral leads to WS-Addressing & WS-RM
  - ▶ Instead of URIs and Idempotent Operations
- ▶ Limited understanding, uneasiness about interoperability, and concerns about the future of WS-\* is a hindrance to adoption



# Questions?

---

- ▶ Blog: <http://netzoid.com>
- ▶ Email: [dan@envoisolutions.com](mailto:dan@envoisolutions.com)

