ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

5 Reliable Messaging Standards (WS-Reliability, WS-ReliableMessaging)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Background and historical perspective

RPC Call semantics

RPC is one to one

Possible semantics are:

- Maybe: no guarantees.
- At least-once: the procedure will be executed if the server does not fail, but it is possible that it is executed more than once.
- At most-once: the procedure will be executed either once or not at all.

Enforcement of these semantics involves additional mechanisms (message counters, message ids, emembering message ids, etc.)

More complex semantics require transactional RPC to be able to extend the properties to sets of calls

RPC results in tightly coupled systems

- Both ends must work
- Knowledge of both ends is needed to program an application

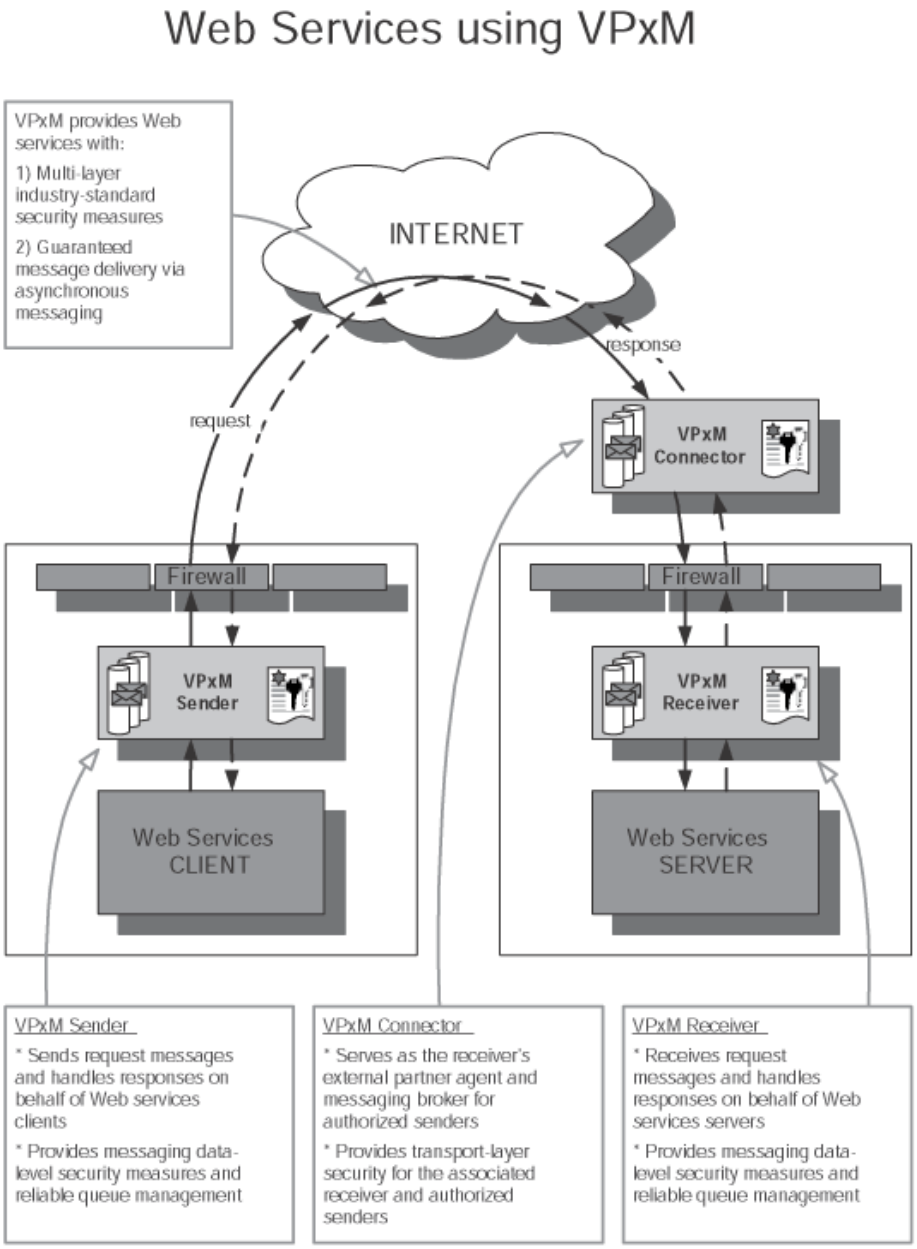
With RPC is difficult to implement any other form of communication than one-to-one:

- N to N
- Many to one
- One to many
- Content driven distribution
- Filters based on content
- Call routing

RPC hides the communication channel behind a programming language construct. That is its main advantage and it key disadvantage.

Queues as a solution (example)

- It is possible to implement the equivalent of messages queues for web services
- The techniques are similar to the technique used to implement asynchronous messaging in conventional middleware
- The problem is that Web services are meant to be platform independent and, thus, the queues have to be standardized, otherwise a WS queue might not be able to talk to another WS-queue, thereby defeating the purpose of web services



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Reliable Messaging in Web Services

Reliability In Distributed Systems

- False Assumptions:
  1. The network is reliable
  2. Latency is zero
  3. Bandwidth is infinite
  4. The network is secure
  5. Topology is stable
  6. There is one administrator
  7. Transport cost is zero
  8. The network is homogeneous

  Peter Deutsch's The Eight Fallacies of Distributed Computing, 1991

- When using Web Services to implement a distributed systems these assumptions do not hold either!

The Case for Reliable Messaging (RM)

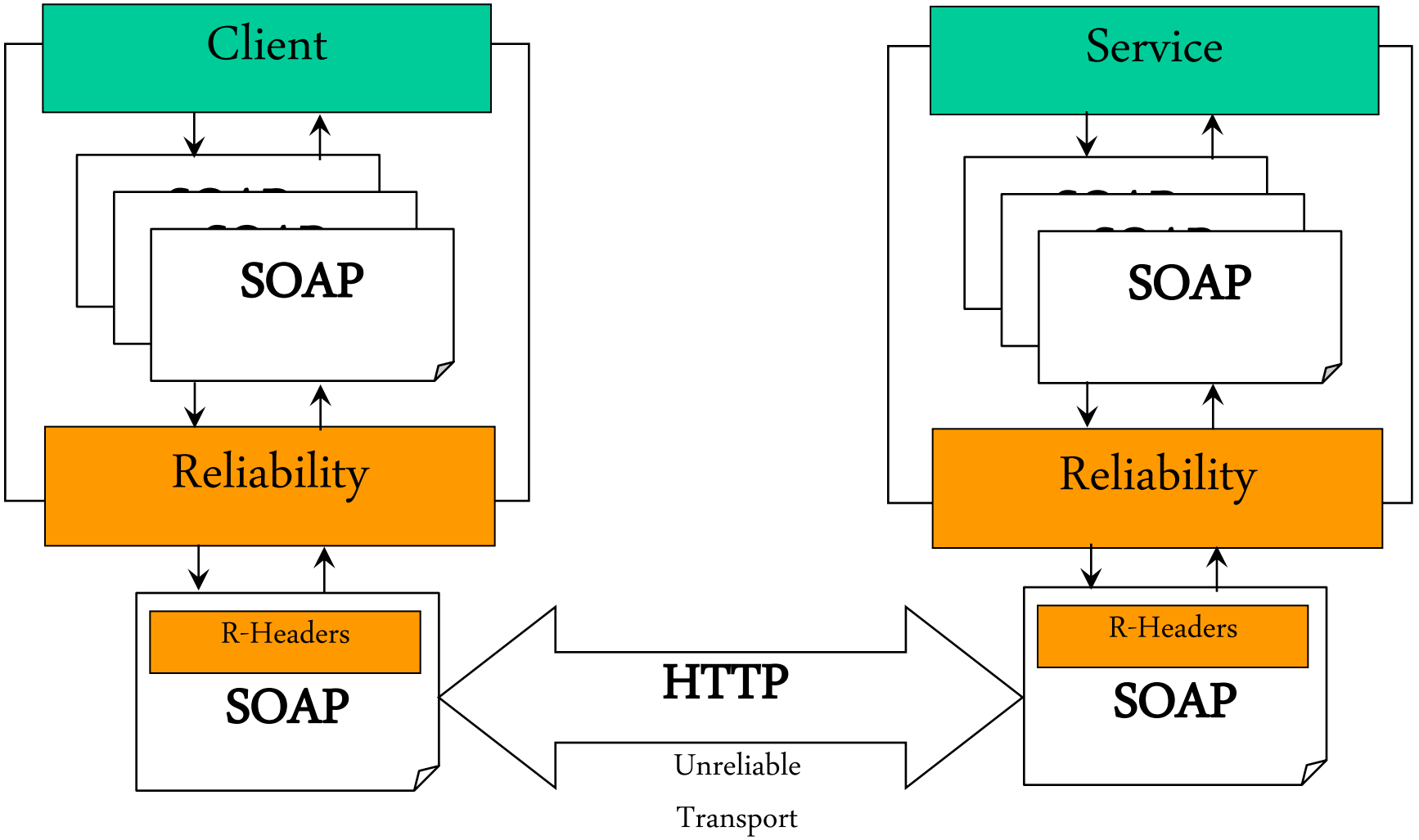
- Technical Issues
  - Compensate for unreliable network transports
  - Reduce coupling in the time-dimension
  - Simplify application development, as reliability concerns are handled by the infrastructure
  - Support for occasionally connected mobile clients
- Business Perspective
  - Reliable Messaging may be required to satisfy business rules, policies and service level agreements.
  - Especially in B2B scenarios, there may be penalties involved if messages (e.g., purchase orders) are lost.
- System Administration
  - Simplify and reduce coordination between different administrative domains.
  - Systems can be taken offline without affecting others

HTTP and Reliability

- Although the HTTP protocol is built on top of TCP, HTTP is not a reliable message transport when compared to “real” message queuing (*MQ, JMS)
- Problem:
  - The transfer of a SOAP message with HTTP may be interrupted at any time (client disconnects, server crashes, network fails)
  - The application should not have to deal with such failures during the message transfer
- Solution:
  - Let the WS stack deal with the problem and provide a reliable message transfer mechanism, which guarantees the application that a message will be delivered with certain properties

Reliable Messaging Overview

- The reliable messaging infrastructure handles transmission errors at the transport layer



Defining Reliable Messaging

- Reliability can be defined at different layers of the stack (Network, Message, Application)
- At the **Message Layer**, reliability implies these properties:
  - Guaranteed Message Delivery, with different QoS:
    - At least once (>=1)
    - Once and only once (=1)
    - At most once (<=1)
    - *Without reliability*: best effort (>=0)
  - Message identification and duplicate elimination
  - Message ordering and priority-based delivery
  - Notification of Message Status

Reliable Messaging Aspects

Message persistence

Messages survive temporary failures of sender, recipients and network

Message ordering

Messages should be received in the same order as they have been sent

Message priority

Messages are usually delivered on first-come, first-served basis.

More advanced is the ability to prioritize messages based on sender or priority metadata.

Message status

Applications should be able to determine whether a message they sent was received and when this occurred.

Message correlation

Related messages should be automatically recognized and grouped by the infrastructure

This is done automatically in case of RPC, where responses are implicitly tied to requests.

Reliable Messaging Techniques

Store and Forward

Progressively get messages closer to their destination.

Messages are stored by intermediate nodes and forwarded as soon as the recipient becomes available.

Acknowledgements

Report that a message was successfully transferred to its destination.

Acknowledgement messages can be positive or negative and can cover ranges (or sets) of messages.

Retries

Message transmission can be repeated in case of lack of acknowledgement (timeout) or negative acknowledgement.

This ensures that a message eventually reaches its destination, but may also introduce duplicates

Duplicate Elimination

Detect and remove duplicate messages before they reach the ultimate receiver.

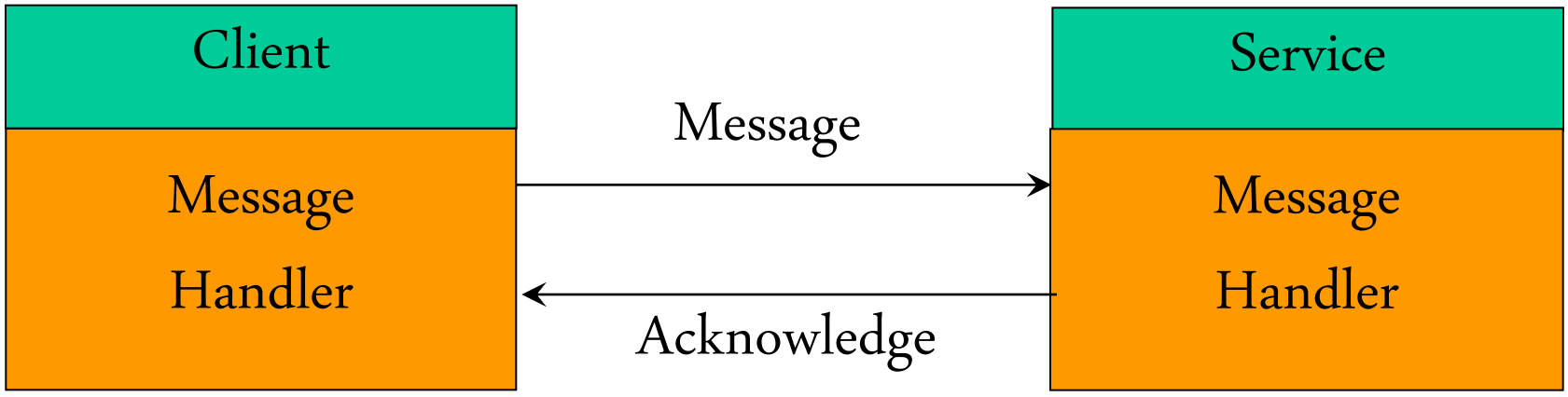
Requires a mechanism to identify Messages

Interoperable Reliable Messaging

- Currently 2 main competing and incompatible specifications:
  - **WS-Reliability (WS-R)**
    - submitted January 2003, v1.1 released November 2004 as an OASIS standard (Fujitsu, Iona, NEC, Nokia, Oracle, SAP, Sonic, Sun, ...)
  - **WS-ReliableMessaging (WS-RM)**, WS-Addressing
    - (March 2003) Microsoft, IBM, BEA, TIBCO
  - WS-Acknowledgement, WS-Callback, WS-MessageData
    - (February 2003, obsolete) BEA
- Eventually will be merged into a new OASIS standard (?).
- Latest version of WS-Reliable Messaging August 2006

Interoperable Reliable Messaging

- All of these specifications focus on the same problem and provide a similar (but not interoperable) protocols based on:
  - Standard SOAP headers
    - Message Identification
    - Message Sequencing
  - Verification of message delivery based on acknowledgement messages



WS-R, WS-RM Feature Comparison

	WS-Reliability	WS-ReliableMessaging
Acknowledgements	Yes	Yes
Message Persistence	Yes	Implied
Message Ordering	Yes (2)	Yes (1)
At-Most-Once	Yes	Yes (1)
At-Least-Once	Yes	Yes
Exactly-Once	Yes (2)	Yes (1)
Time To Live	Yes	Yes
Duplicate Detection	Yes (2)	Yes (1)

(1) The receiver is responsible for implementing this semantics

(2) Message Ordering requires Duplicate Detection and Exactly-Once.

WS-R, WS-RM Feature Comparison (2)

	WS-Reliability	WS-ReliableMessaging
Fault Handling	Yes	Yes
Retry	Yes	Yes, with exponential backoff
Ack polling	Yes	No
Negative Acks	No	Yes
Ack piggyback	No	Yes
MessageID	Yes	Yes
SequenceID	Yes (start at 0)	Yes (start at 1)
Callback location	Yes (URI)	Yes (WS-Addressing)
Description	RM Agreements	WS-Policy
Standard	OASIS	No

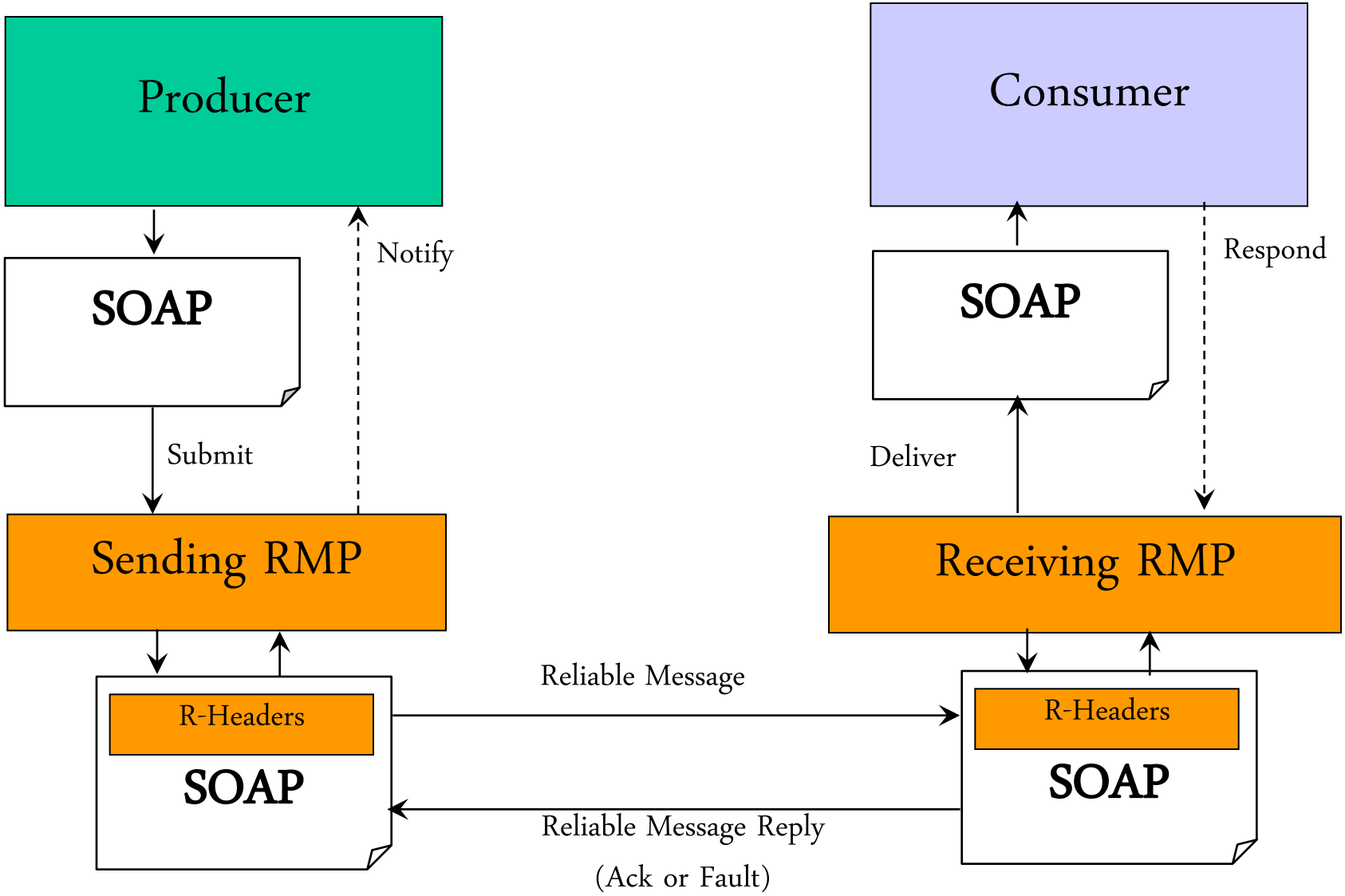
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WS-Reliability

WS-Reliability Overview

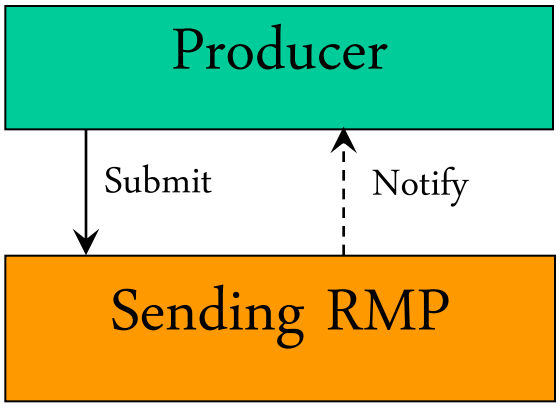
- WS-Reliability is a protocol used to provide reliable delivery of SOAP messages, independent of the underlying transport protocol (transport-agnostic).
- It features all of these message delivery semantics:
  - At least once (guaranteed delivery)
  - At most once (duplicate elimination)
  - Exactly once
  - Guaranteed ordering (within a group of messages)
- It also defines a standard interface contract for accessing the reliable messaging layer.
- As opposed to WS-ReliableMessaging, WS-Reliability is an OASIS standard, released on November 2004.
- The standard has evolved the ebXML Message Service.

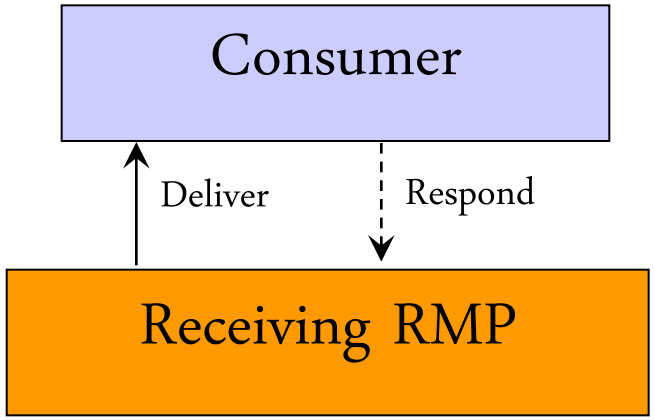
Message Processing Model



WS-Reliability QoS Contract

- The Reliable Messaging Processor offers the following interface contract to message producers and consumers:





1. **Submit** – A message payload is transferred to the RMP subject to a reliability QoS agreement.
2. **Deliver** – The RMP returns every valid received message that satisfies its associated reliability requirements (ordering, duplicate elimination).
3. **Respond** – an optional response to a delivered reliable message can be send by the consumer to the producer
4. **Notify** – the RMP informs the producer about the failed delivery of a message and returns the consumer response (if any)

Reliable Messaging Agreement

- A reliable messaging agreement describes which reliability features have been agreed upon by the sending and receiving parties when exchanging one (or a group) of messages.
  - A Sending RMP is (somehow) configured to support a certain RM agreement.
  - A Receiving RMP discovers the Sending RMP configuration because it is sent along in the header of a Reliable Message.

Quality of Service Agreement
Guaranteed Delivery
Duplicate Elimination
Ordered Delivery

Protocol Agreement
Timing
Maximum Idle Time, Expiry Time
Patterns
Response, Callback, Poll

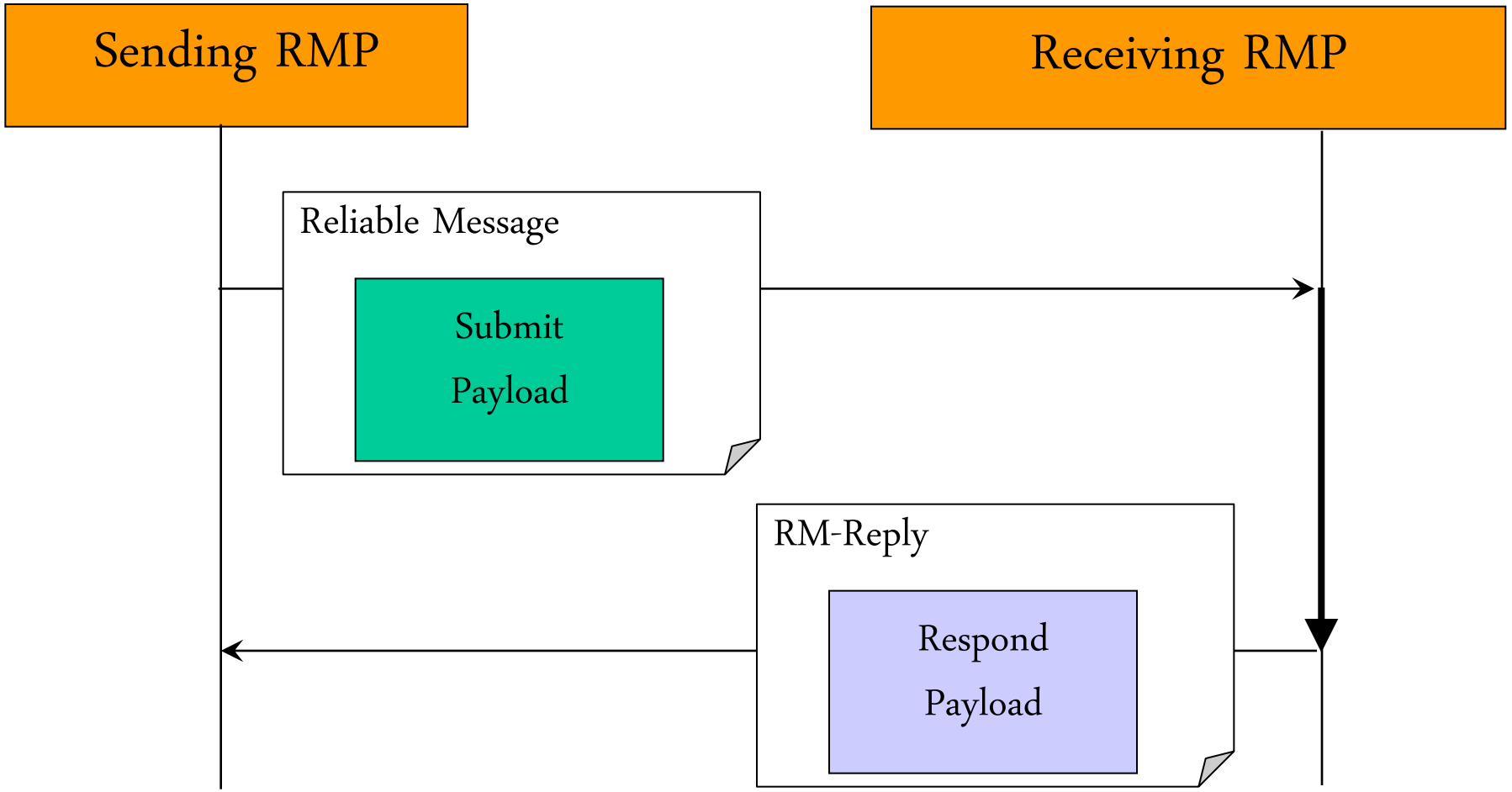
RM Agreement Quality of Service

- The RM Agreement covers these Quality of Service options:
  - Guaranteed Delivery
    - if enabled, the producer is guaranteed to be notified if a failure occurred while delivering the message
  - Duplicate Elimination
    - if enabled, the consumer is guaranteed not to receive more than one copy of the same message that was submitted by the producer.
    - Warning: this does not apply to “respond payloads”.
  - Ordered Delivery
    - a group of messages is delivered to the consumer in the same order (and without missing messages) as it was originally submitted by the producer.
    - Ordered Delivery requires that Guaranteed Delivery and Duplicate Elimination are also enabled.

RM Agreement: Protocol Options

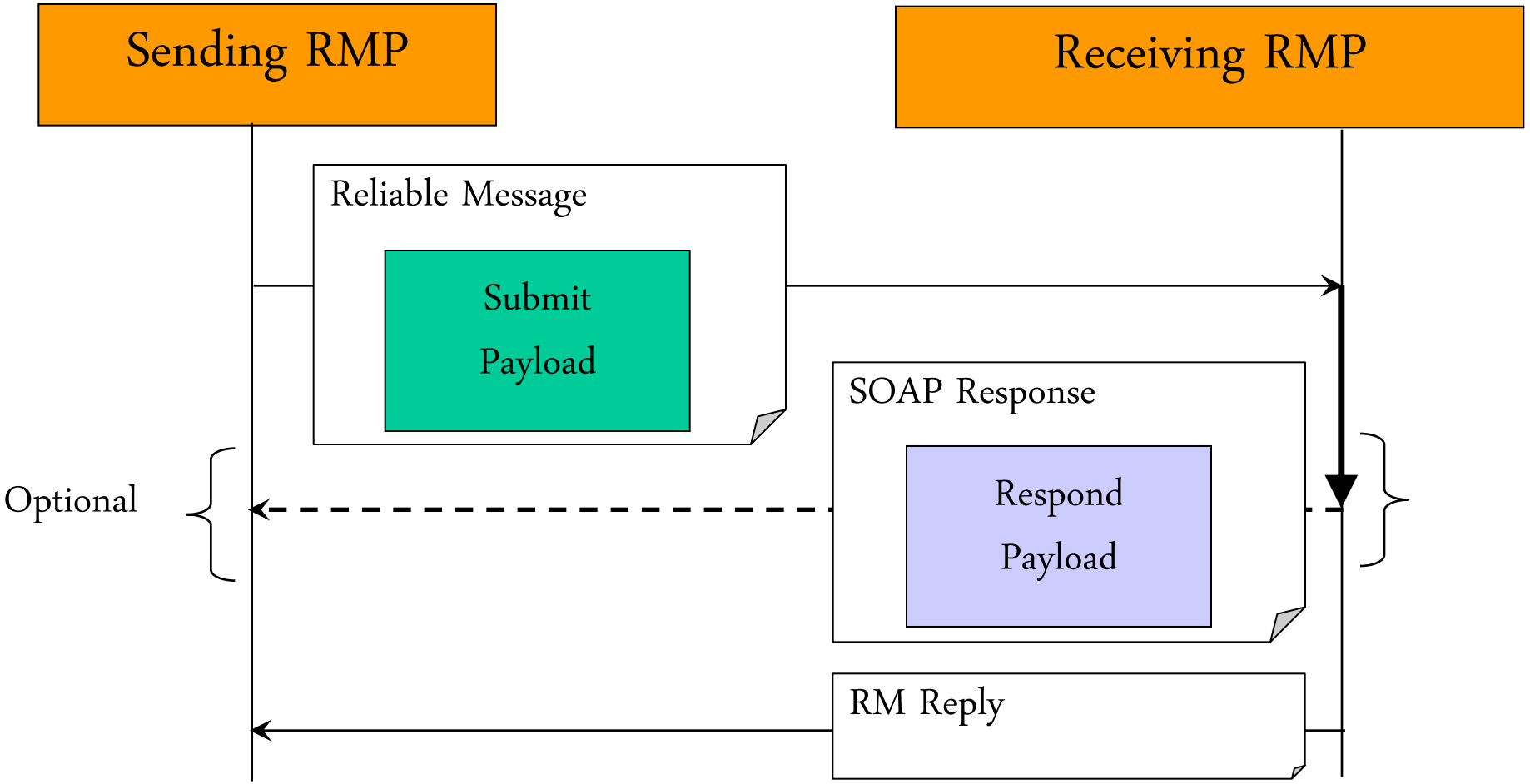
- Controlling the lifetime of a message group with either one of the two options:
  - Group Maximum Idle Duration
    - Implicit termination of a group if a message has not been exchanged for this amount of time.
  - Group Expiry Time
    - Explicit deadline to terminate a group of messages
- Options for individual messages:
  - Expiry Time
    - Deadline for resending a message to the consumer
  - Protocol patter to use for sending the RM Reply:
    - Response
    - Callback
    - Poll

WS-R Protocols: **Response** RM-Reply



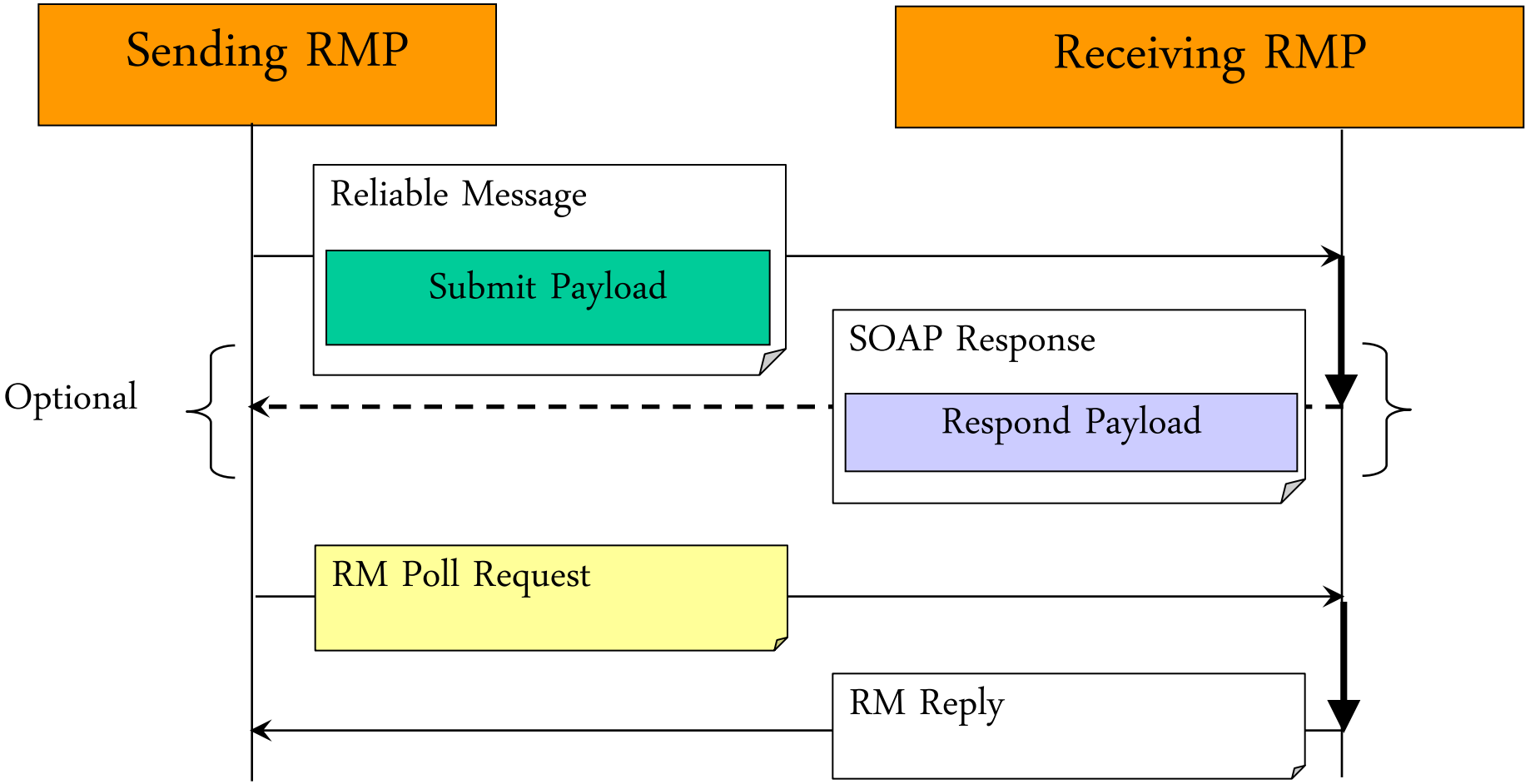
- This pattern is used for SOAP Request-Response message pairs. The RM Reply is sent together with the response.

WS-R Protocols: **Callback** RM-Reply



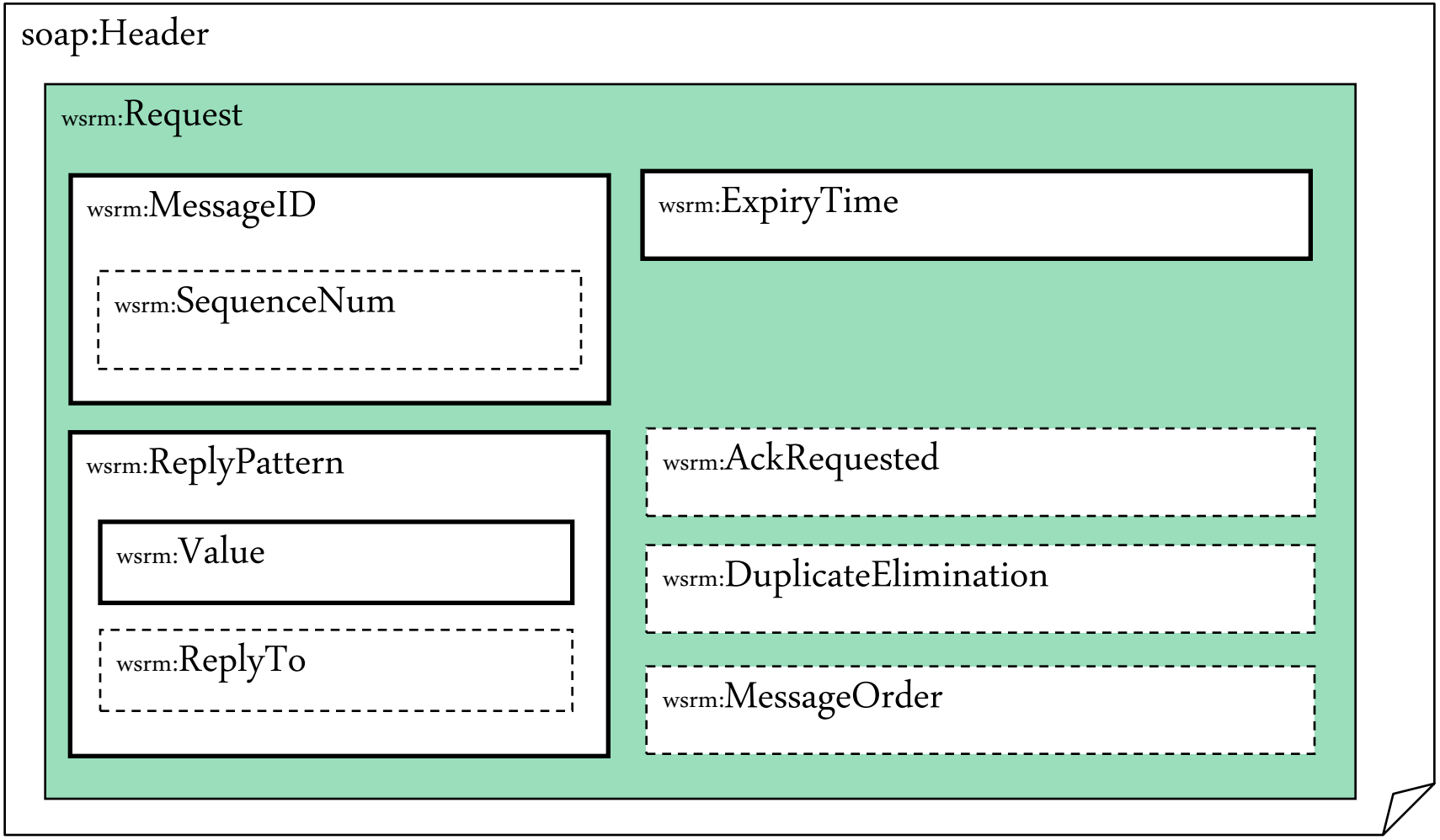
- This pattern uses a separate one-way (callback) SOAP message to report on the status of the previous message

WS-R Protocols: **Poll** RM-Reply

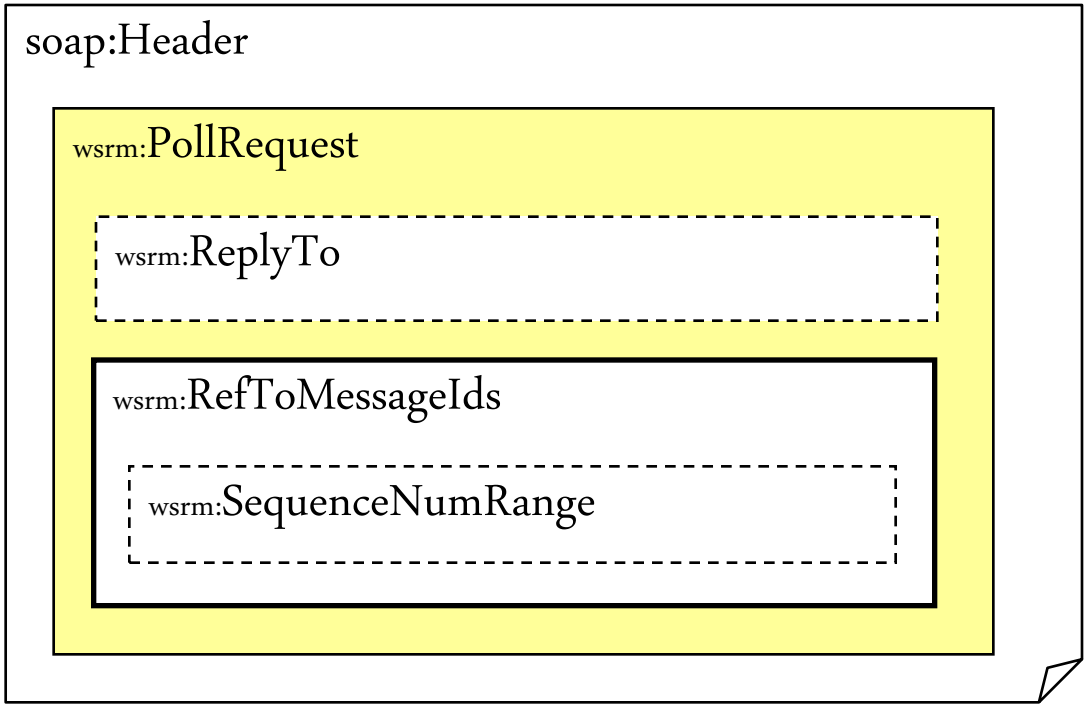


- Two variants: synchronous (RM Reply sent with response of the poll request) or asynchronous (RM Reply sent later).

RM Message Request Header Format



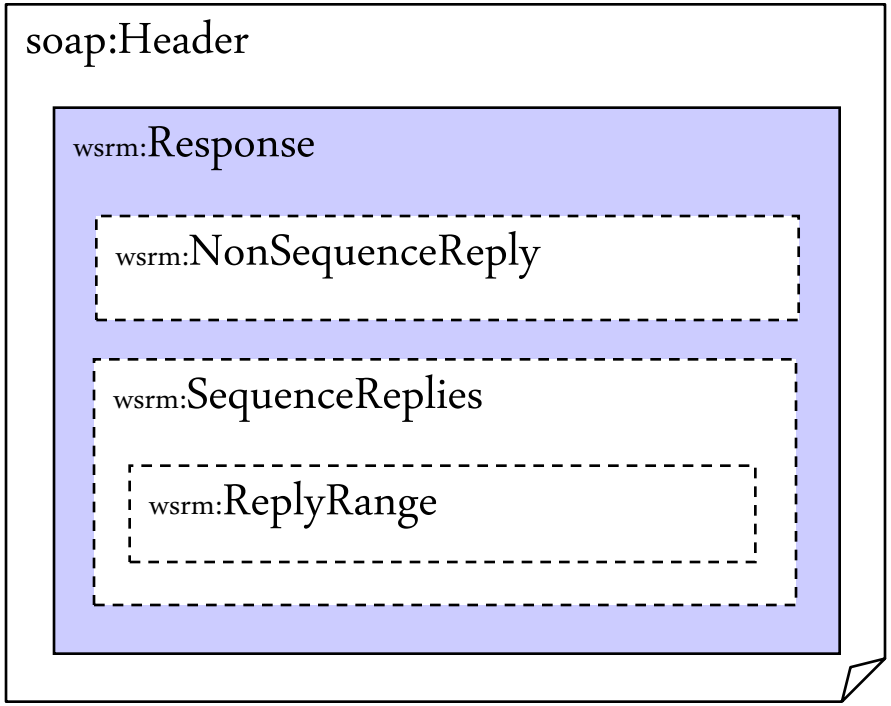
RM Message PollRequest Header Format



- Poll Request headers are used to actively ask the Receiving RMP for the status of previously sent messages.
- These messages are identified by their groupIDs and sequence number ranges.
- In case an asynchronous Poll pattern is used, the ReplyTo address includes the endpoint to which the RM Reply should be sent.

RM Message Response Header Format

- A wsrm:Response header acknowledges the successful receipt (or the failure) of one or more messages.
- It lists the set of messages that were received using their groupId and sequence ranges.
- "Fault" attributes can be attached to each message to indicate that a problem has occurred:
  - Invalid message format
  - Permanent or transient message processing failure
  - Feature not supported



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

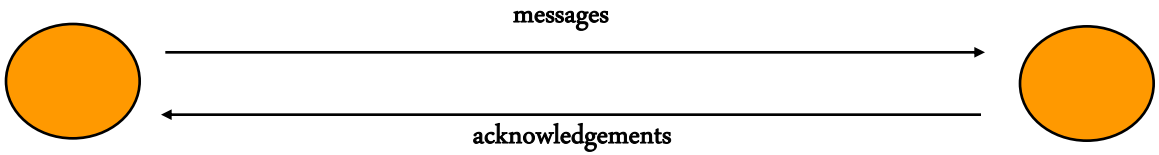
WS-ReliableMessaging

WS-ReliableMessaging Overview

- WS-ReliableMessaging defines a set of SOAP Header extension elements that enable:
  - Guaranteed Message Delivery
  - Message Ordering Preservation
  - Duplicate Detection
- Unlike other proposals, WS-RM accomplishes this with a “much simpler syntax and more efficient processing semantics”:
  - The basic protocol provides at least once and it is up to the recipient site to implement duplicate detection and ordering preservation semantics
- The protocol can be implemented using different network transport technologies. To support interoperable Web services, a SOAP binding is included in the specification.

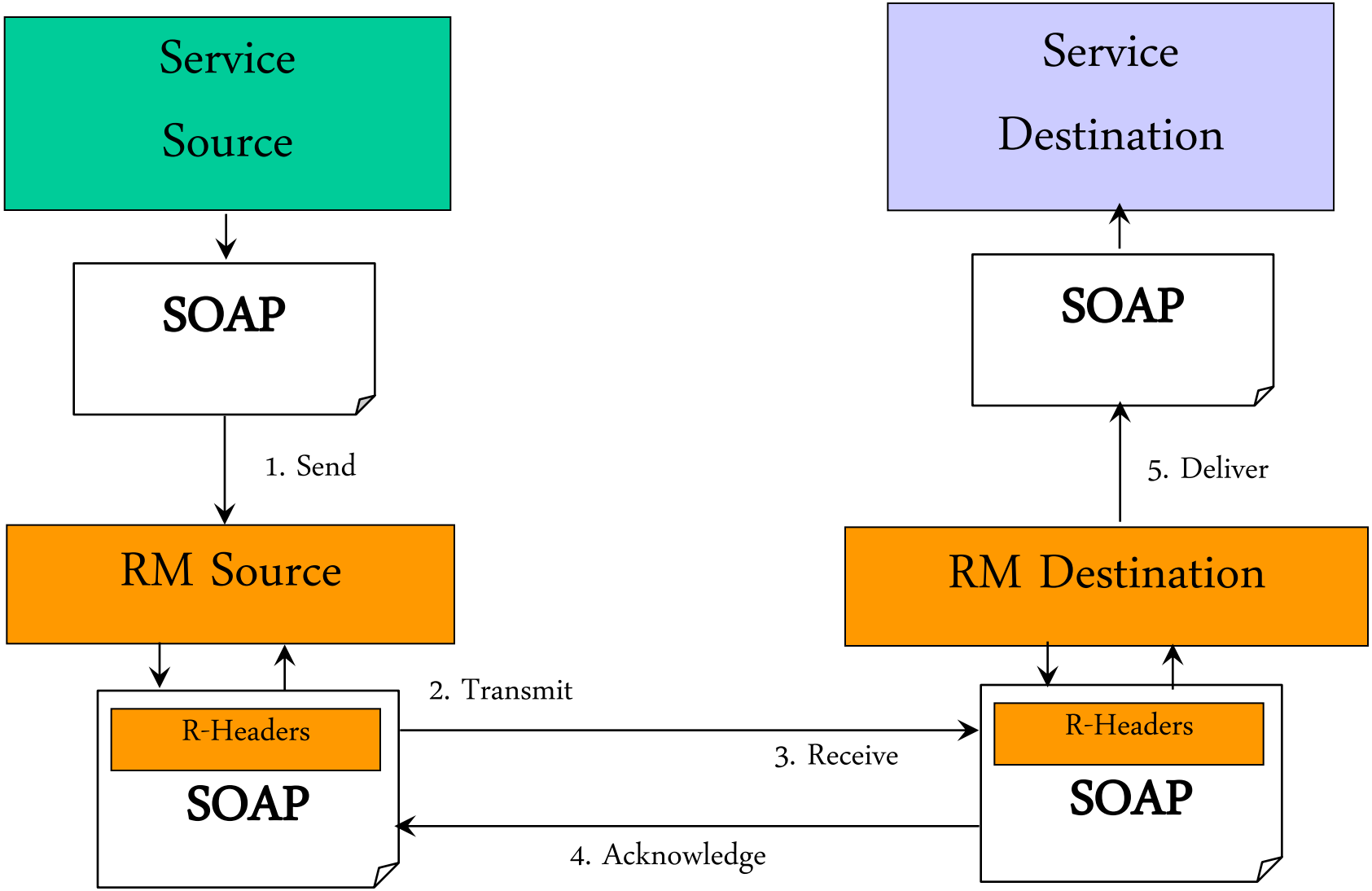
Basic model

- Based on a sender and a receiver
- Communication happens one way (from the sender to the receiver)
- Reliable two way communication requires to duplicate the set up in the other direction

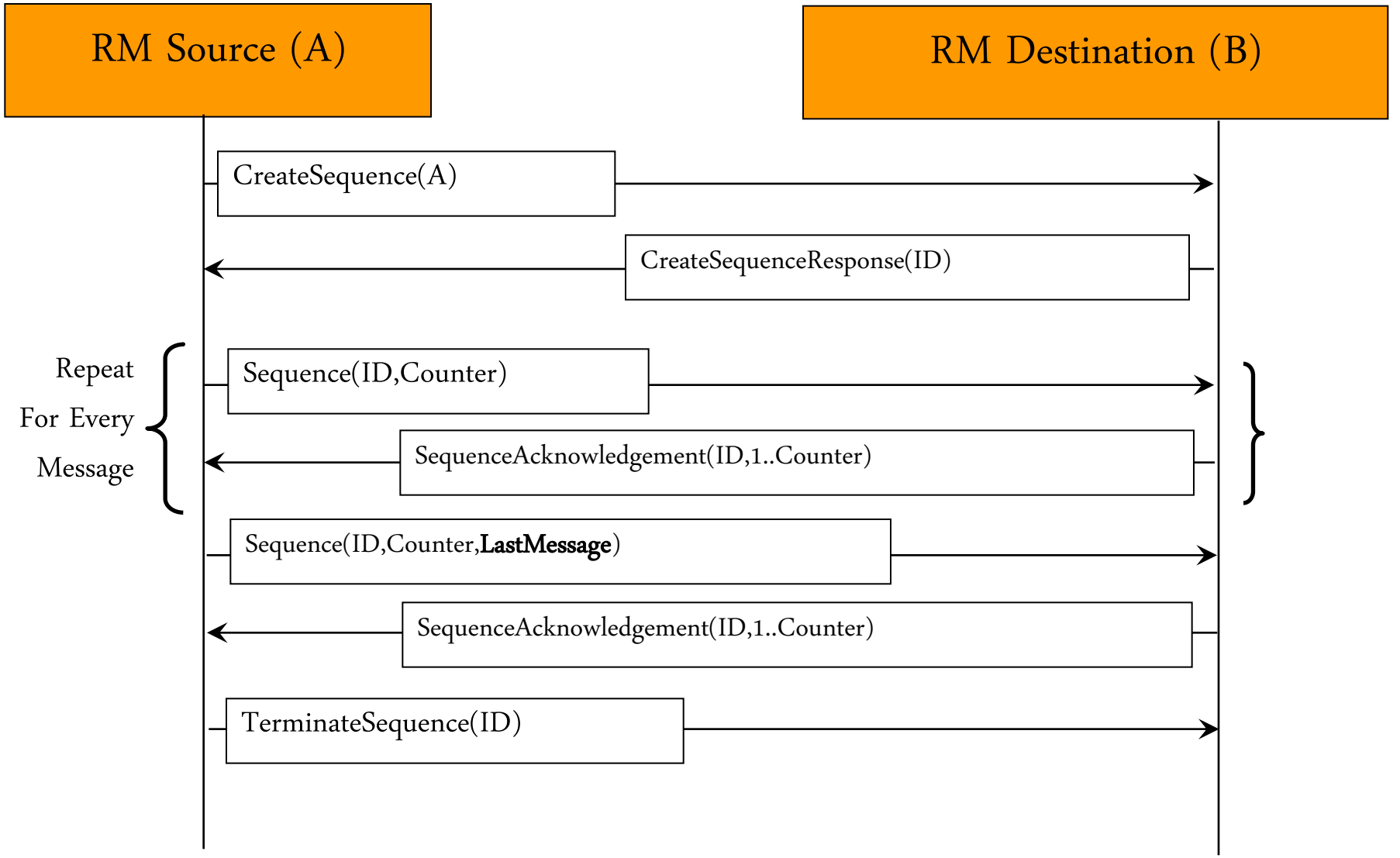


- RM Source (RMS)
  - Request creation of a sequence of reliable messages (contract)
  - Requests termination of a sequence of reliable messages
  - Add proper headers to messages
  - Requests acknowledgements
  - Resend messages if need be
- RM Destination
  - Responds to requests to create a sequence of reliable messages
  - Responds to requests to terminate a sequence of reliable messages
  - Accepts and acknowledges messages (synchronous or asynchronously)
  - May drop duplicate messages
  - May queue messages to deliver in order

Message Processing Model



Message Sequence Lifecycle



Standardized Faults

- **Sequence Terminated**
  - An endpoint chooses to terminate the sequence in response to an exceptional condition
- **Unknown Sequence**
  - A message contains an unknown sequence identifier
- **Invalid Acknowledgement**
  - A source receives an acknowledge about messages that have not yet been sent
- **Message Number Rollover**
  - Overflow of mssg. numbers
  - The message numbers for a sequence have been exhausted and the sequence is now terminated
- **Create Sequence Refused**
  - A destination cannot initiate a new message sequence
- **Sequence Closed**
  - Message sent in an already closed connection

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

6 WS Coordination
WS Transactions

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

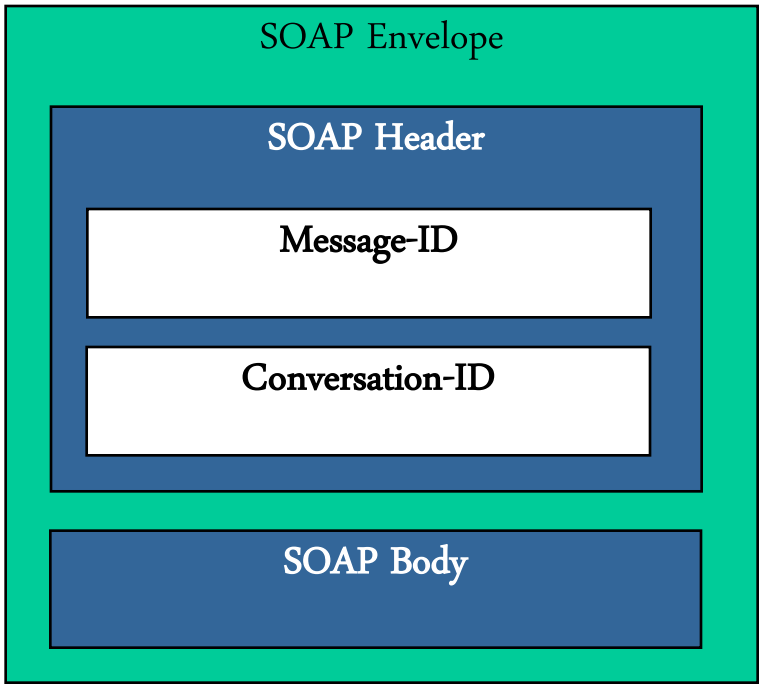
The Problem of Service Coordination

Standardizing Coordination Protocols

- SOAP has been a successful standard fostering the interoperability at the messaging layer. With it, messages can be exchanged regardless of the transport.
- Although SOAP includes the notion of message header, it does not define how such header should be used by the messaging infrastructure.
- To achieve the transparent **conversation routing** of messages, there is the need for a standardized way to generate unique conversation identifiers and store such information in the SOAP message headers.
- WSDL describes the interface of a service by only listing the operations it provides in terms of the structure of the messages that can be sent or received.
- To achieve transparent protocol compliance, a standard is needed to define what are the valid conversations supported by the service interface.
- Additionally, a meta-protocol standard is missing for two parties to agree on the set of supported protocols and how the infrastructure should coordinate them.
- Finally, standards should also cover the handling of concrete protocols (e.g., 2PC, or reliable message delivery)

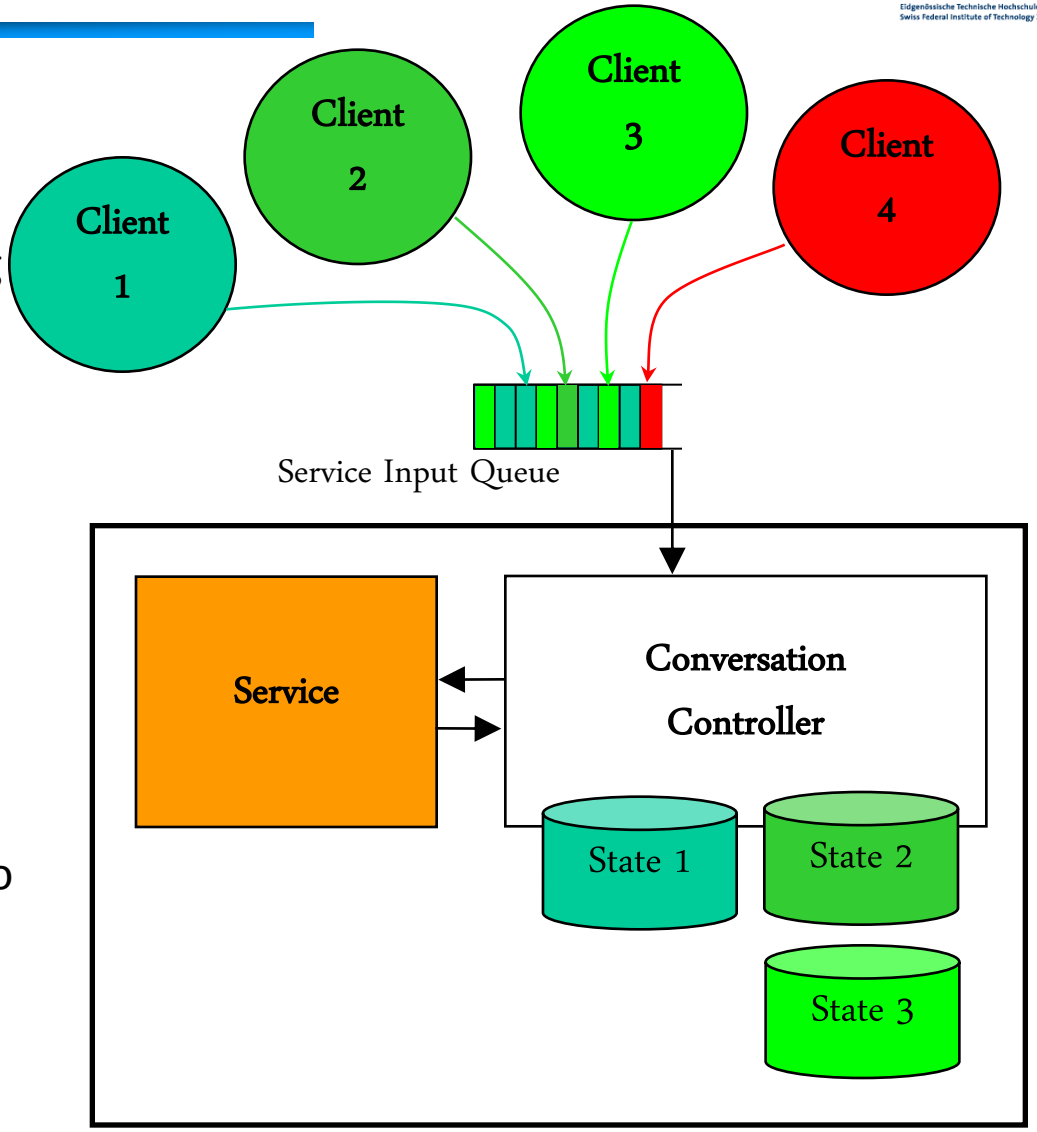
Message Correlation

- In order to support several concurrent conversations, the Web service/messaging infrastructure must define a way to uniquely identify messages and to determine which messages belong to a given conversation.
- The problem consists of reconstructing the session (and related context information) from a stateless, one-time message exchange.
- A similar problem exists with HTTP: how to establish and maintain session information across multiple request-response interactions. The standard solution is to use cookies, bits of information that persist across multiple request-response interactions.
- Reconstructing the context of a message can be done with different assumptions: Identifiers are stored in the header (correlation by the infrastructure) or the content of the message may allow the application to process it correctly (ad-hoc)



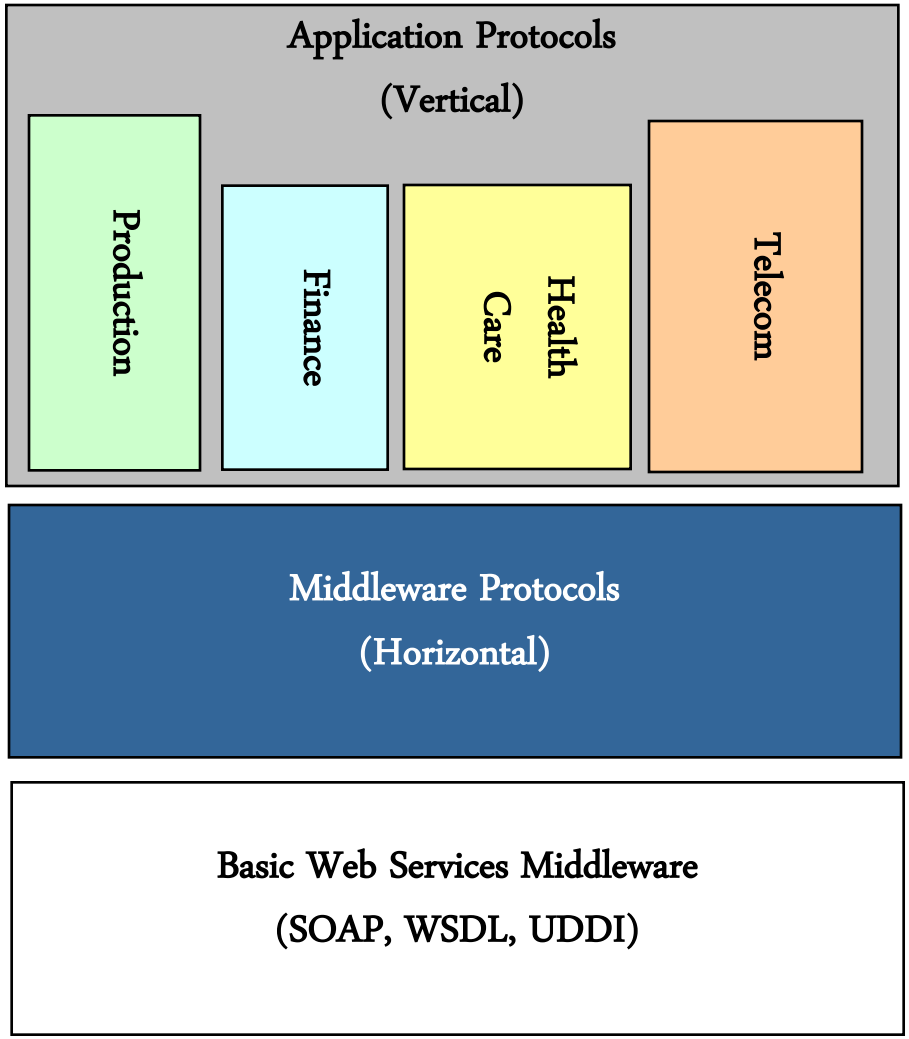
Conversation Controllers

- Conversation controllers are the key component of the coordination infrastructure built on existing reliable messaging middleware. They provide:
  - **Conversation routing.** Considering that there are multiple concurrent executions of a conversation, messages belonging to different conversations must be distinguished and correlated with the current state of the conversation.
  - **Protocol compliance.** Once a message is received, the controller must decide whether it should be accepted as it is expected as part of a conversation or rejected as it does not belong to any ongoing conversation



Horizontal and Vertical Protocols

- Horizontal protocols (or middleware protocols) define a common, application-independent infrastructure. Building on top of the basic messaging infrastructure, they extend it with additional properties (e.g., reliability, transactions, security)
- Vertical Protocols are used within a specific business area (e.g., manufacturing, health care, finance, telecommunication). They describe in detail how to conduct concrete business transactions, what are the documents involved and what is their format and semantics.
- Examples of vertical protocols are xCBL or RosettaNet. They can be seen as the XML evolution of previous ones such as EDI.

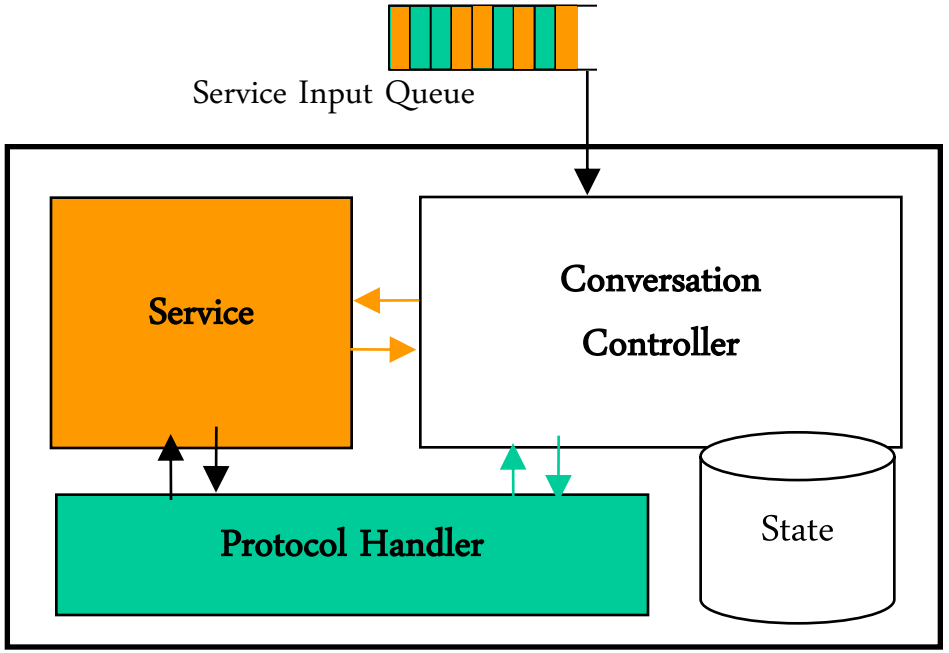


Transparent protocol handling

- In addition to stateful conversations, assuming the appropriate level of standardization, the coordination infrastructure can handle coordination protocols providing properties such as security, transactions and reliability.

- To do so, a protocol handling layer is added between the low-level messaging infrastructure and the implementation of a service.

- The protocol handler may be transparent and exchange messages automatically, without the intervention of the Web service implementation.
  For example, reliable message delivery can be implemented this way.



- Alternatively, the protocol handler provides a framework which is reused by the service implementation. For example, a generic 2PC transactional protocol handler may rely on the service to decide whether to commit or abort the transaction, while coordinating the delivery of the corresponding messages.

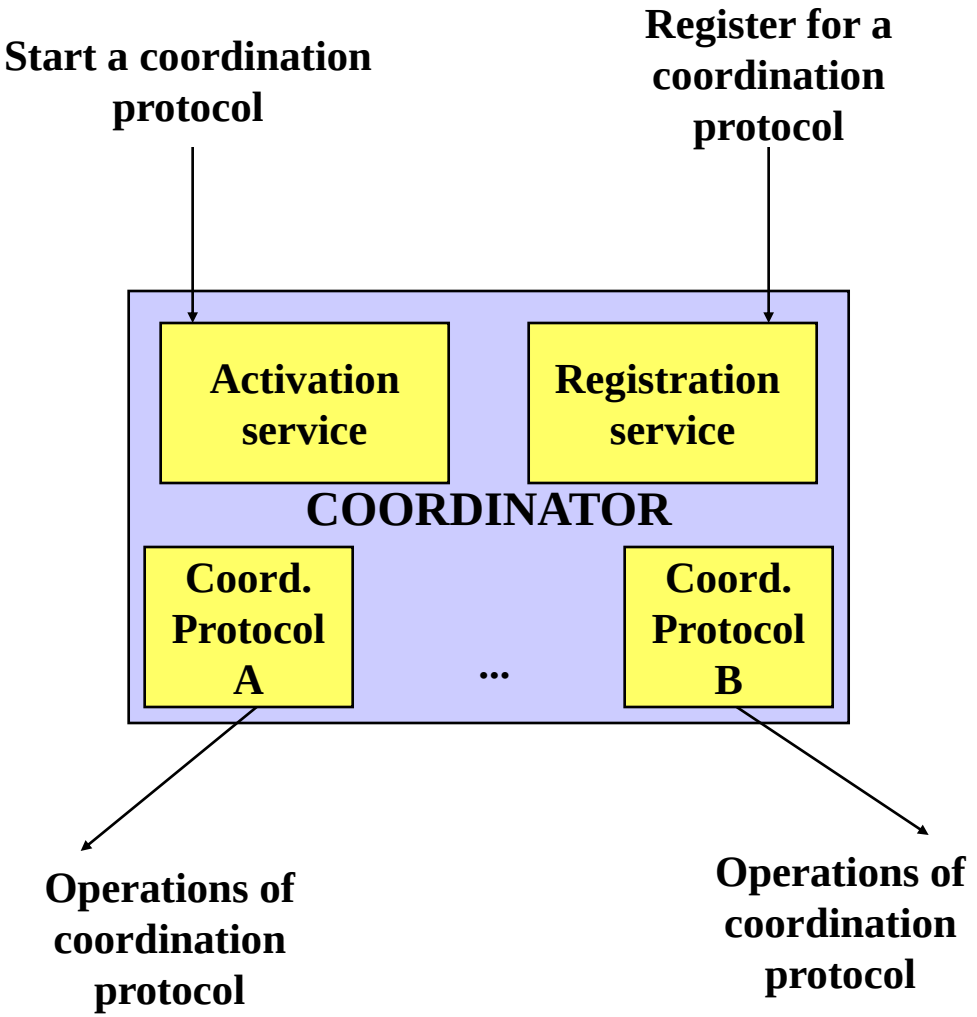
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

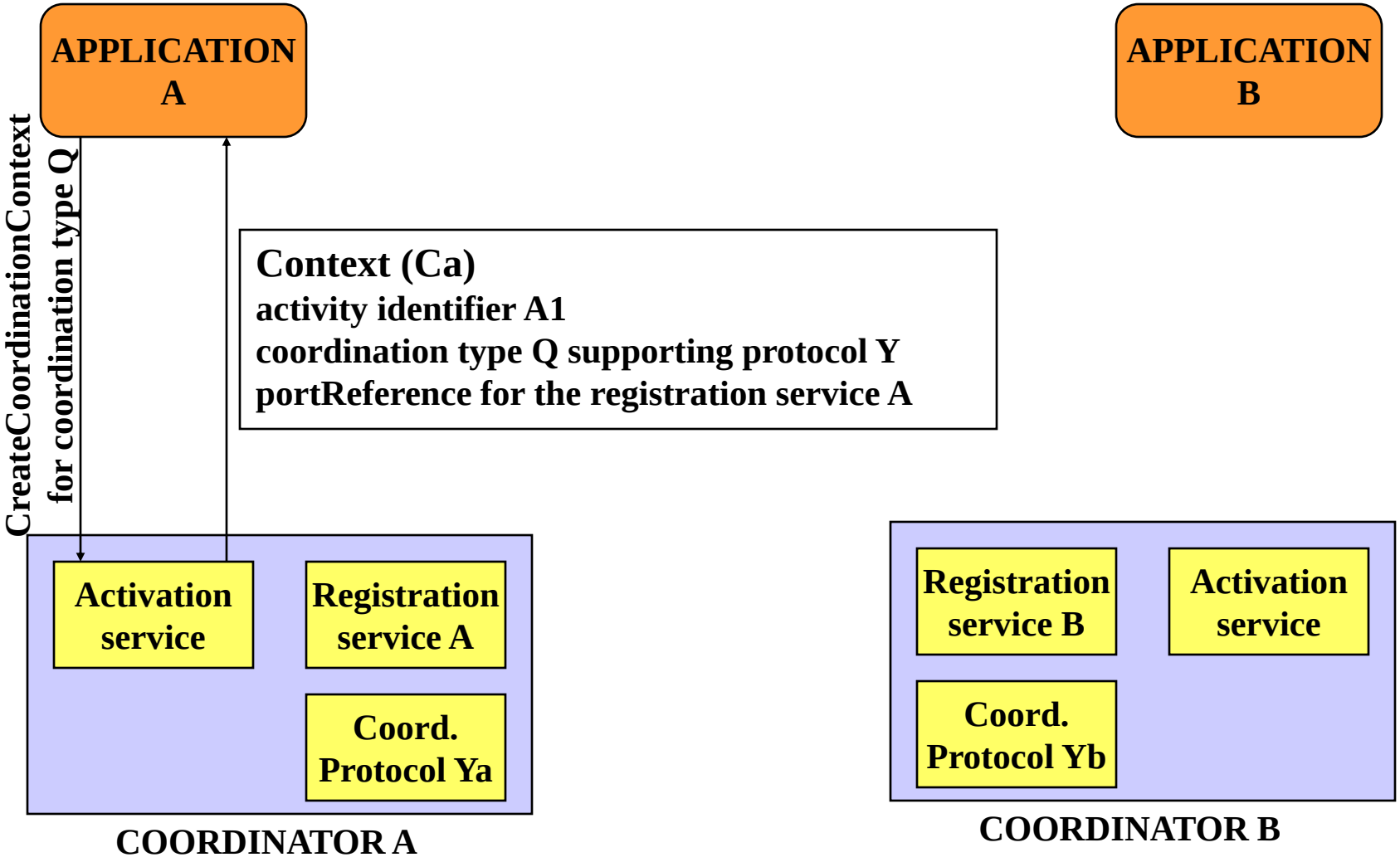
WS-Coordination

WS-Coordination

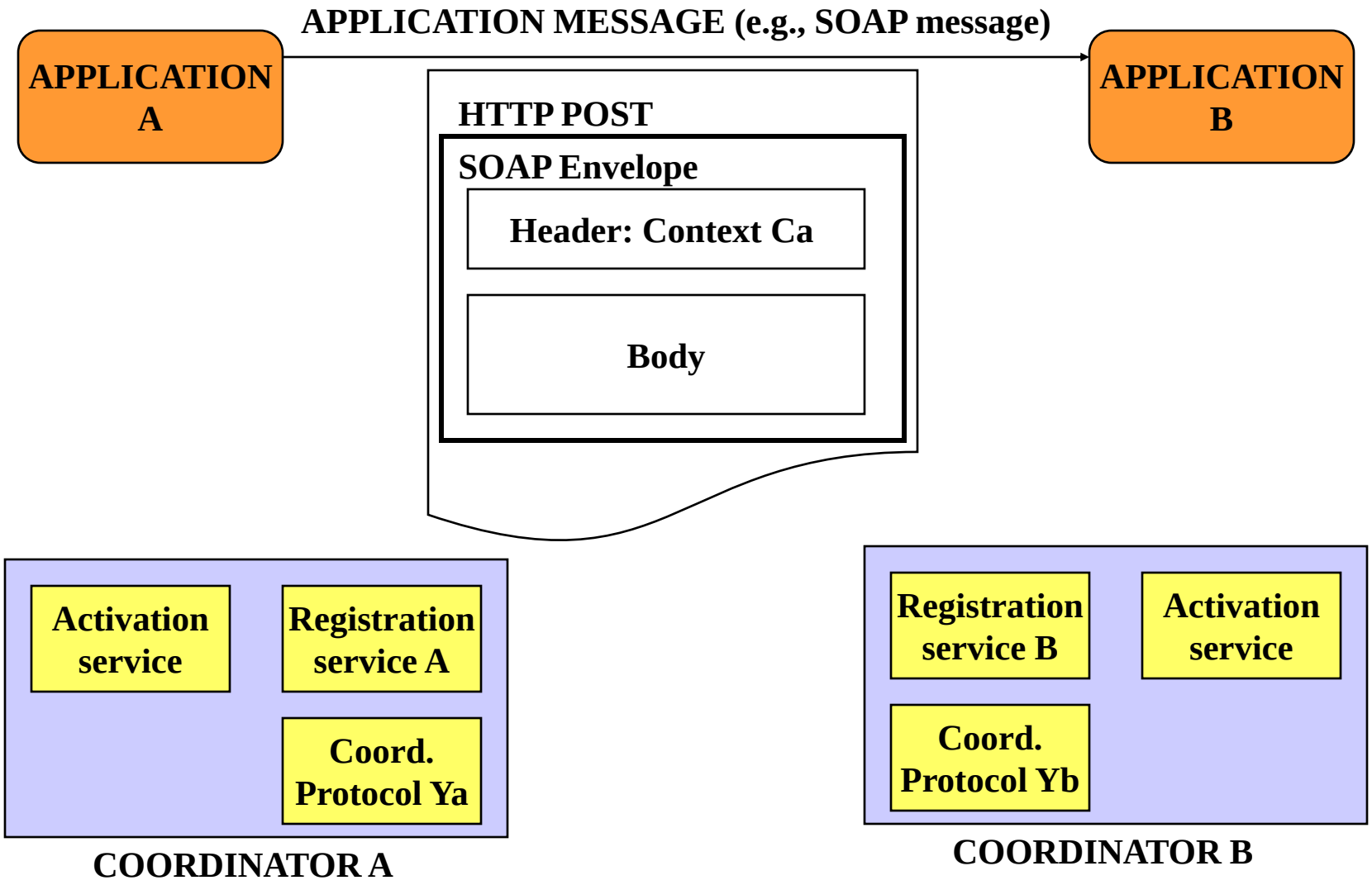
- WS-Coordination is intended as a generic infrastructure to implement coordination protocols between Web services
- Its main goal is to serve as a generic platform for implementing advanced transaction models but it can be used to implement a wide variety of coordination protocols between services (including some forms of conversations)
- WS-Coordination encompasses a set of behaviors and APIs which enable a module to extend Web services with coordination capabilities
- It mirrors the behavior of transactional services in conventional middleware platforms



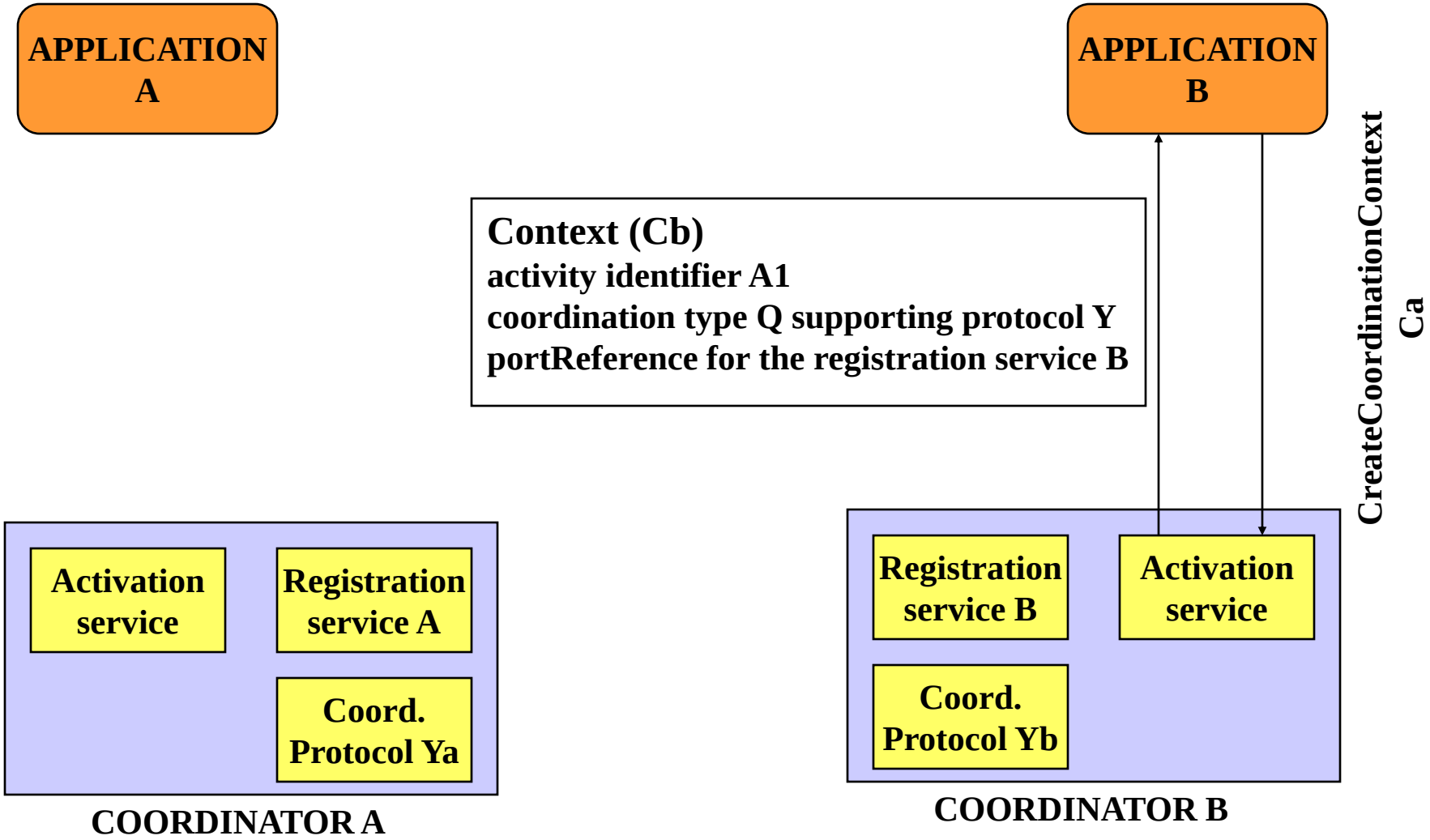
Basics of WS-Coordination (1)



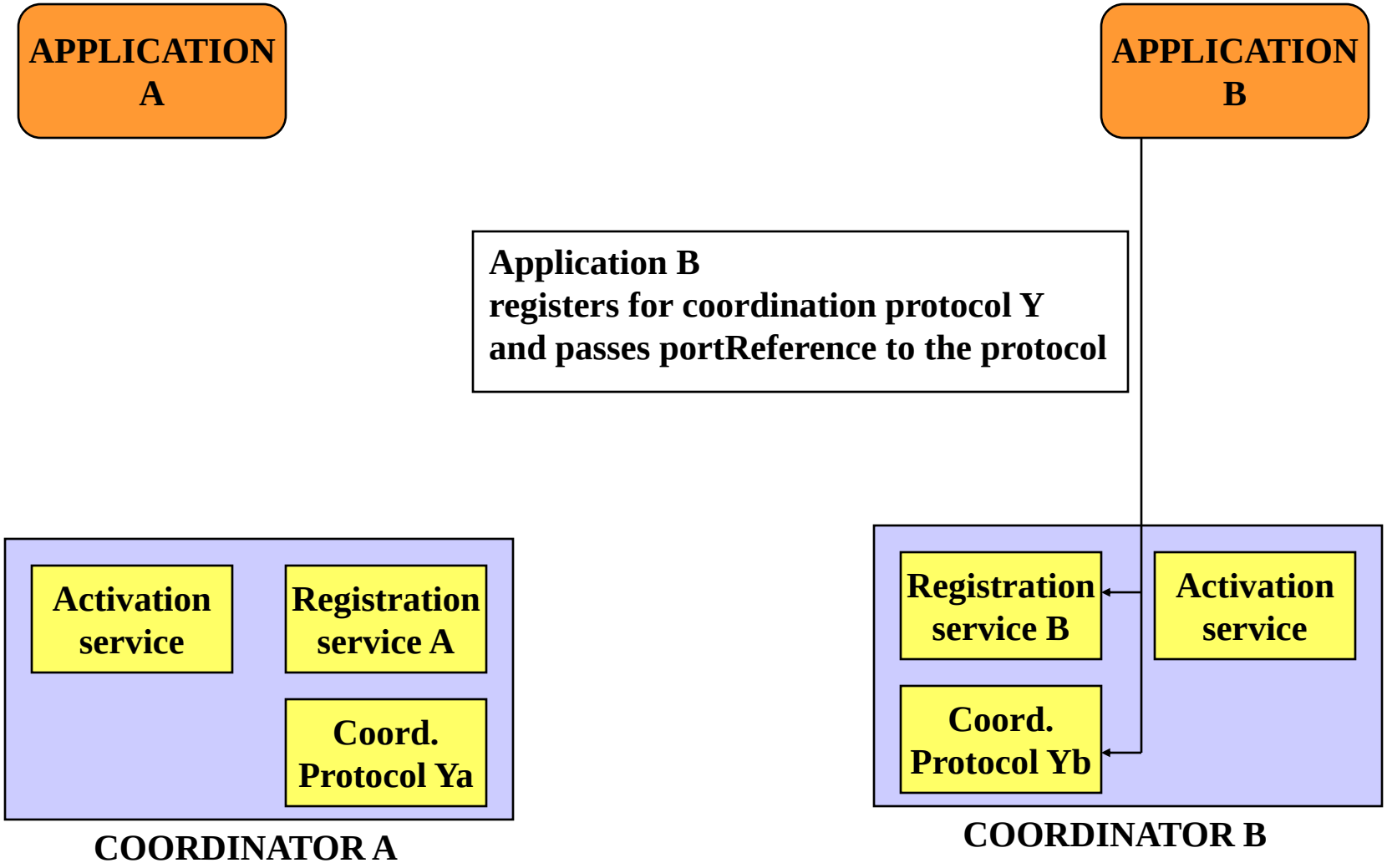
Basics of WS-Coordination (2)



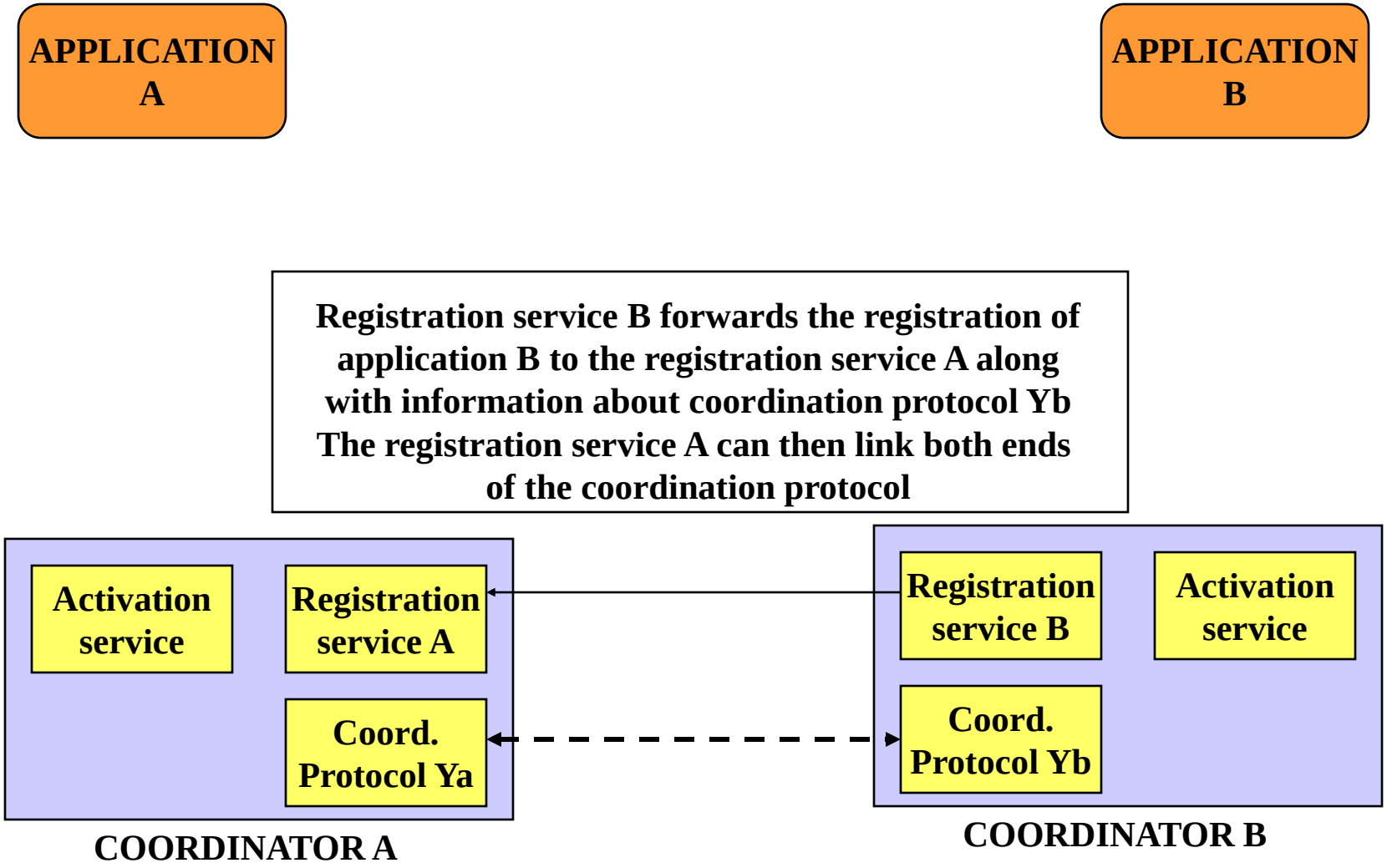
Basics of WS-Coordination (3)



Basics of WS-Coordination (4)



Basics of WS-Coordination (5)

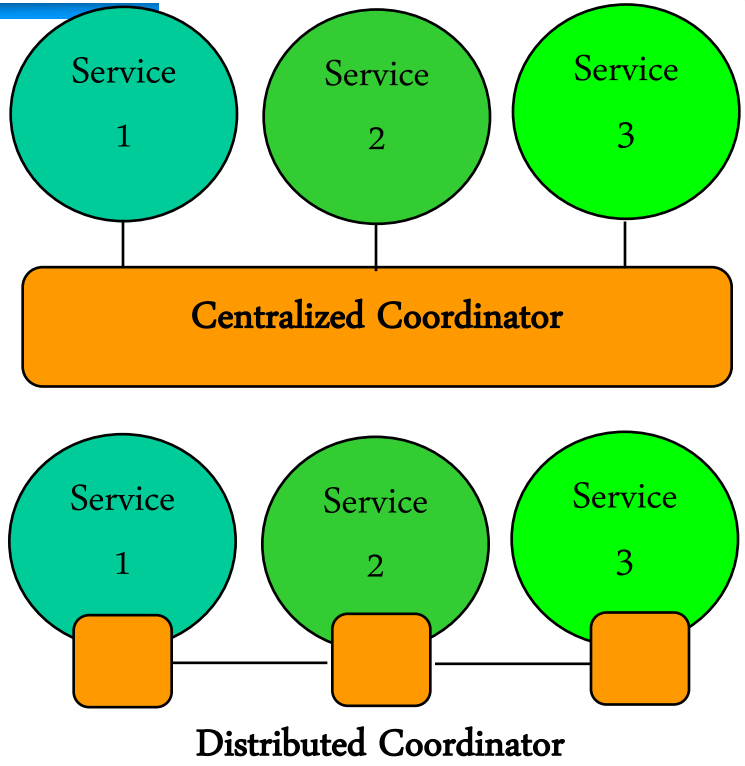


Messages and Interfaces

- The coordinator defined by WS-Coordination is described using WSDL and offers a number of services to the application.
- The application accesses these services by sending, e.g., SOAP messages to the coordinator which then responds with new SOAP messages. Interactions with the protocol would then also be in terms of SOAP messages (but other protocols are possible, one simply needs to provide alternative bindings for the coordinator services)
- The example shown considers the case where application B decides to use its own coordinator. Application B could also decide to use the same coordinator as application A but in the cases where A and B are independent services, provided by different organizations, a coordinator per application makes more sense
- WS-Coordination is an attempt at standardizing:
  - the use of SOAP headers for coordination protocols
  - the basic operations for most coordination protocols
  - the functionality a Web service middleware platform must support for allowing coordination protocols to be implemented

Summary

- WS-Coordination is a standard coordination infrastructure proposed in August 2002
- It defines a **coordination context** to be included in the header of SOAP messages. The context stores unique conversation identifiers used for routing and protocol verification.
- It includes a solution for **registering** the protocol handlers of the Web services with the coordination infrastructure. With it, protocol handlers can be notified when specific steps of the protocol are carried out.
- It contains an **activation** interface, used to create a new coordination context and inform each peer about the role it should assume while running the protocol.



- WS-Coordination attempts to generalize existing middleware solutions (e.g., CORBA OTS) to provide a common foundation of specific forms of coordination
- *Note:* WS-Coordination is not a language for defining coordination protocols.

ETH

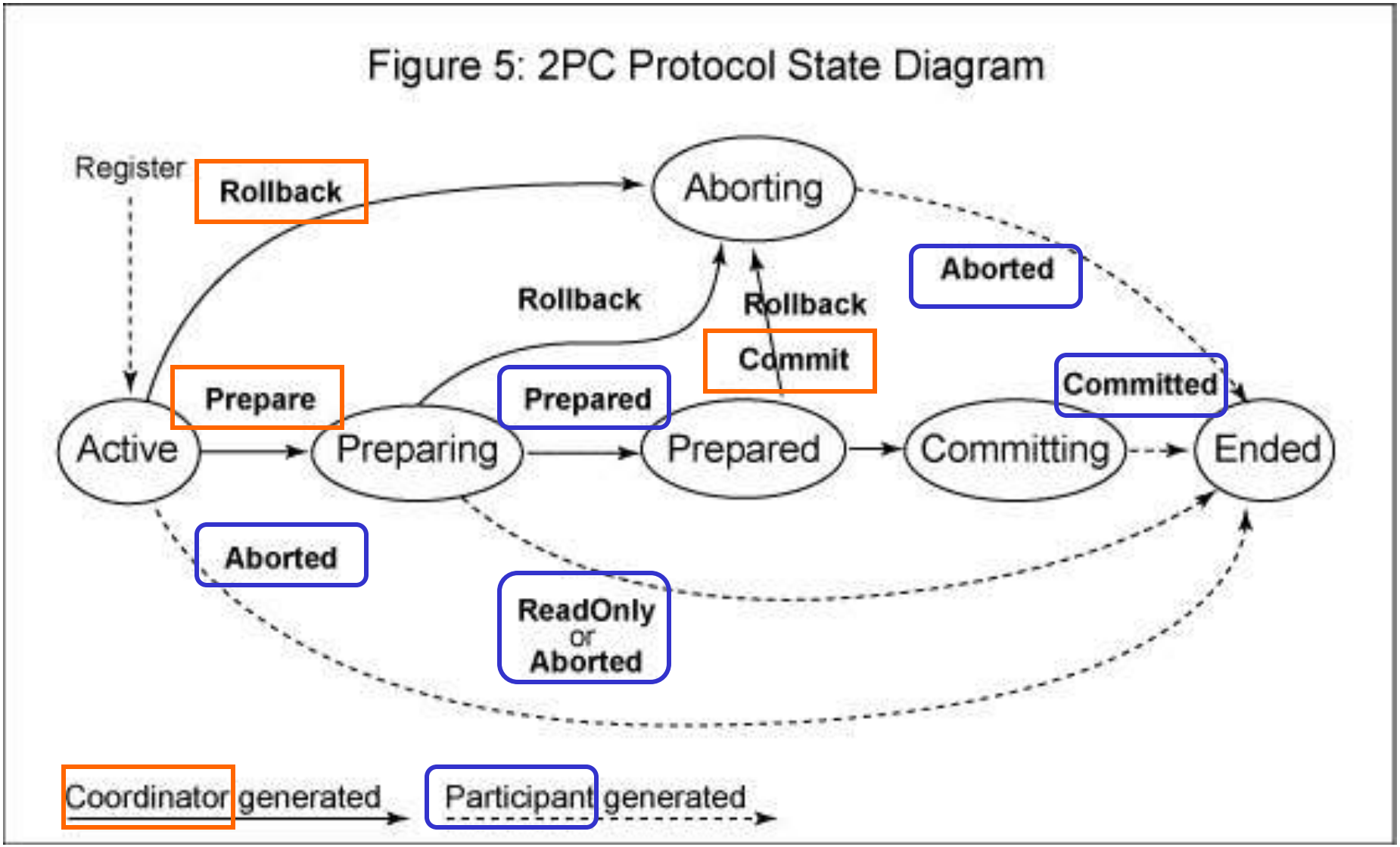
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WS-Transactions

WS-Transactions

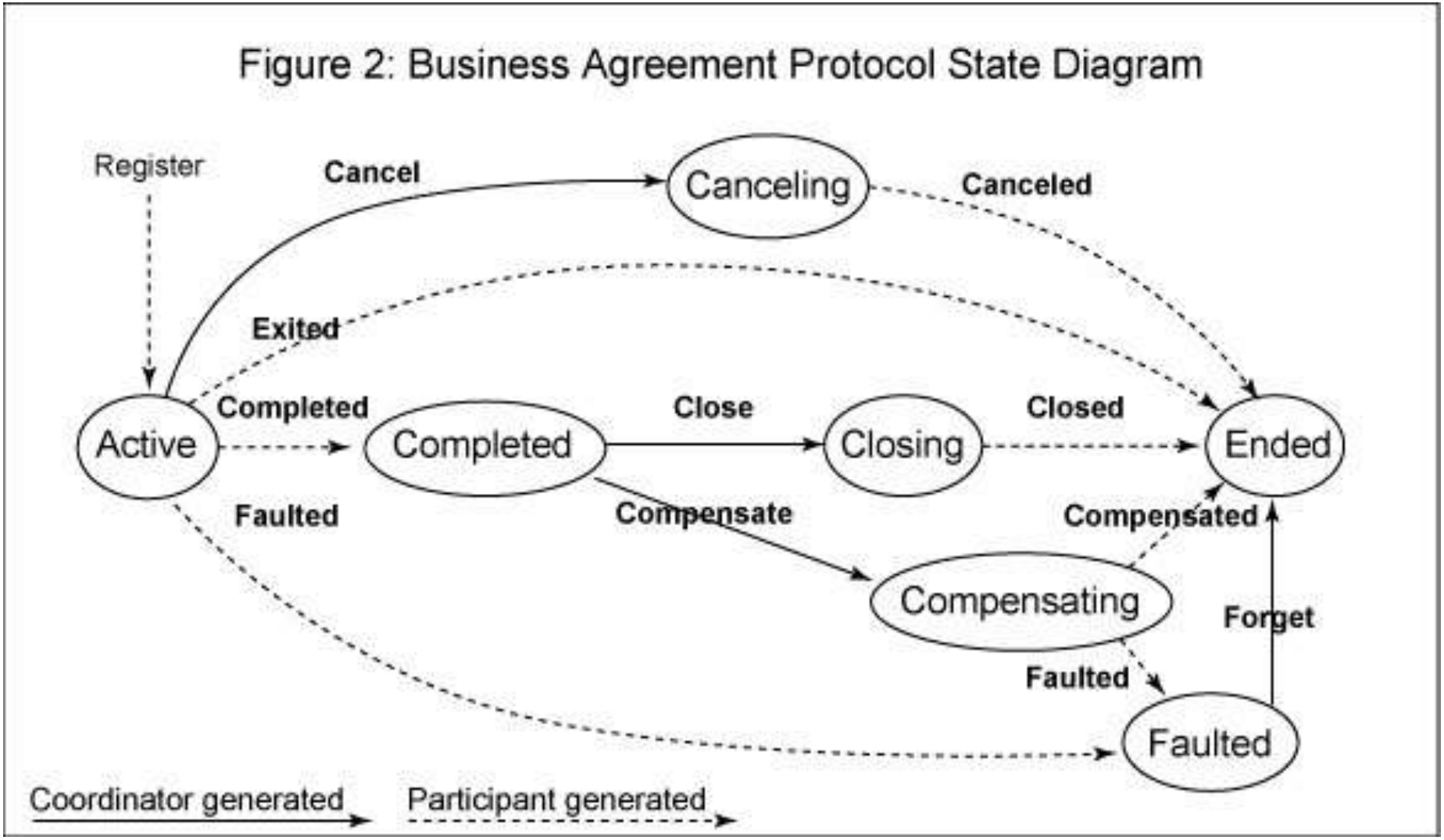
- WS-Transactions builds directly upon WS-Coordination to specify different coordination protocols related to transaction processing
  - atomic transactions (governed by 2 Phase Commit)
  - business activities (transactional but based on compensation activities)
    - business agreement
    - business agreement with complete
- WS-Transactions specify the coordination protocol to be used as part of WS-Coordination. The specification deals with the nature of the interaction, the syntax and semantics of the messages to be exchanged as part of the coordination protocol, and the expected responses of all participants involved
- Like WS-Coordination, WS-Transactions follows very closely the transactional model found in conventional middleware platforms

Coordination Protocol for 2PC



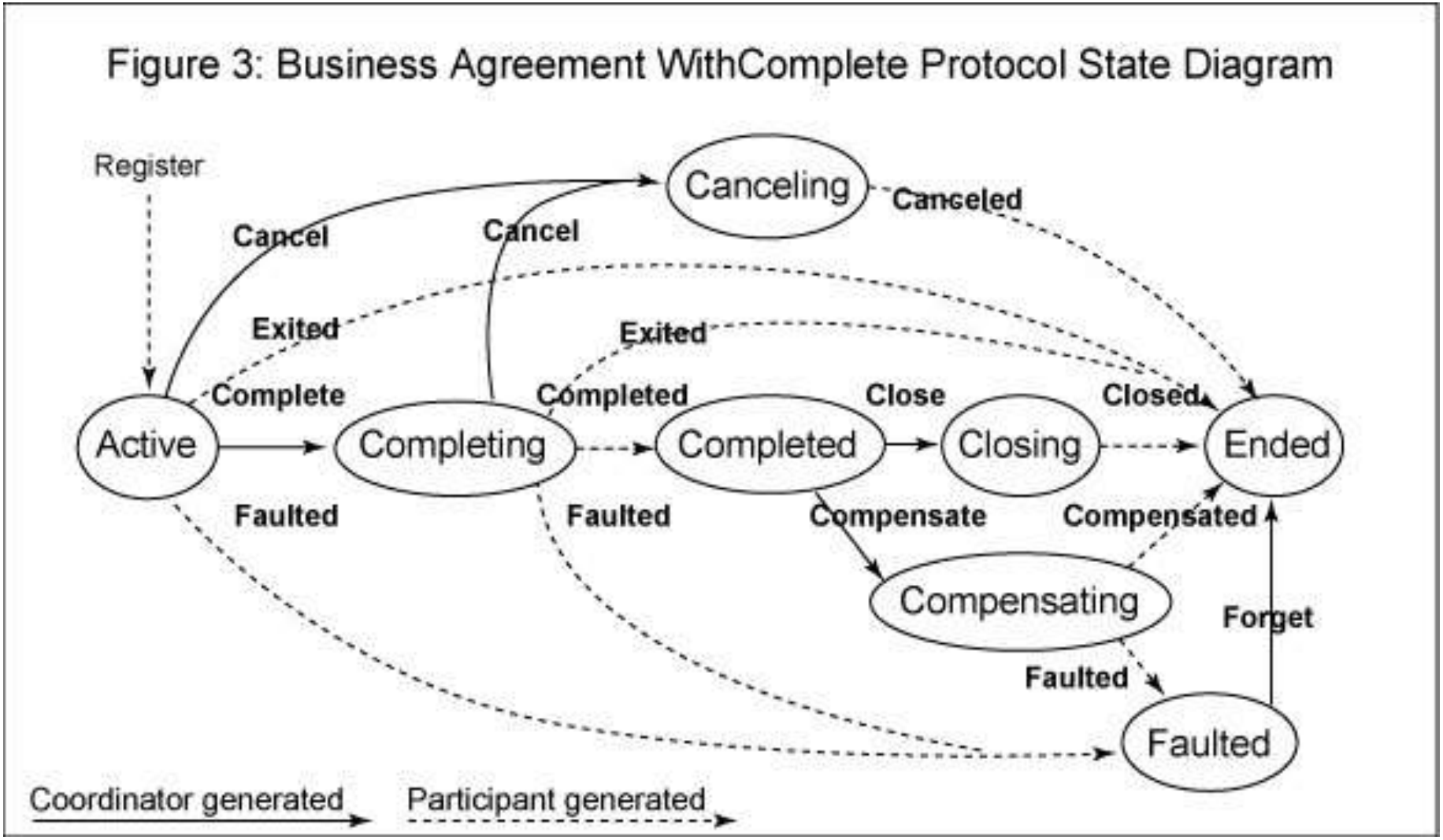
From Web Services Transaction (WS-Transaction) 9 August 2002

Business Agreement



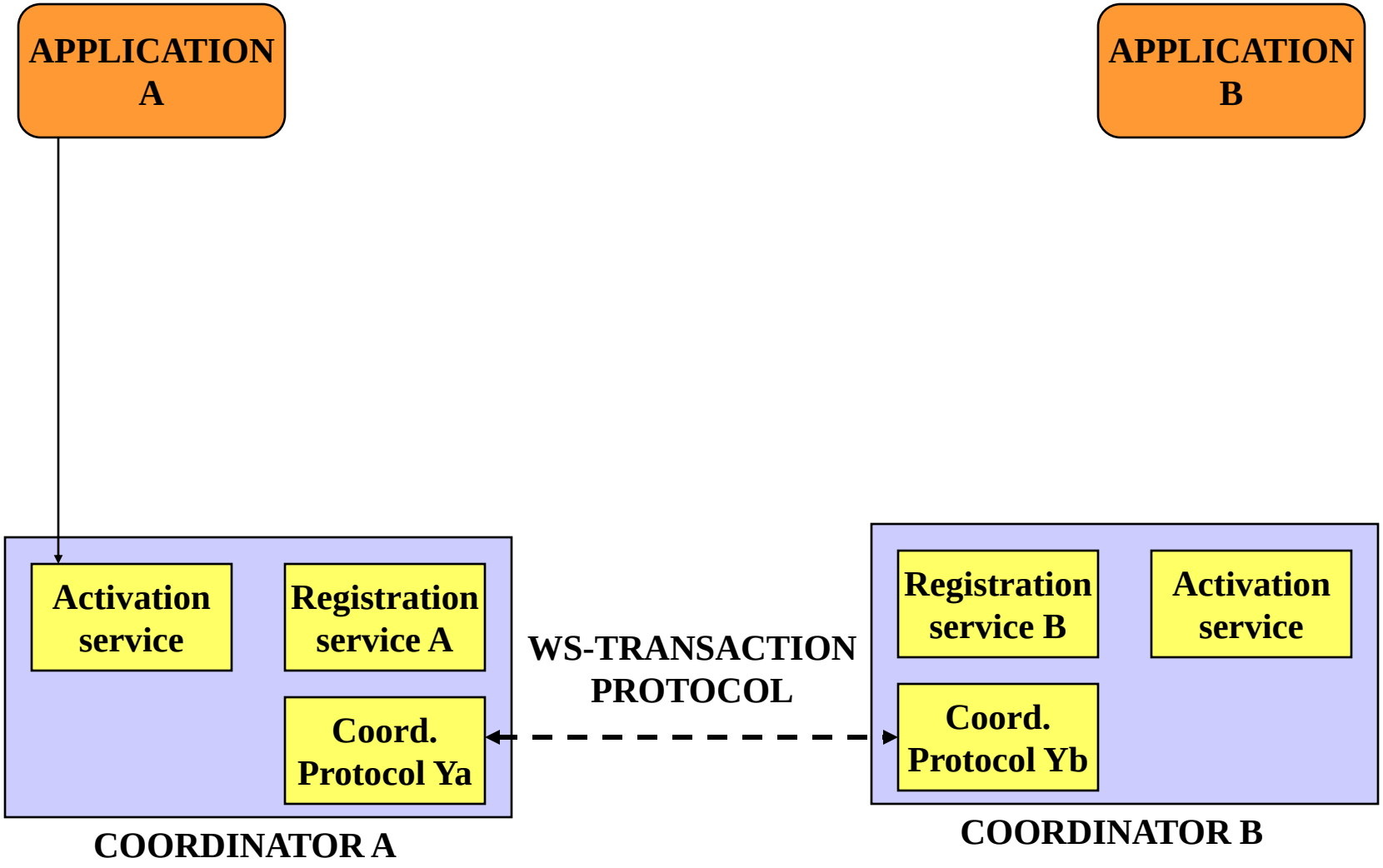
From Web Services Transaction (WS-Transaction) 9 August 2002

Business Agreement with Completion



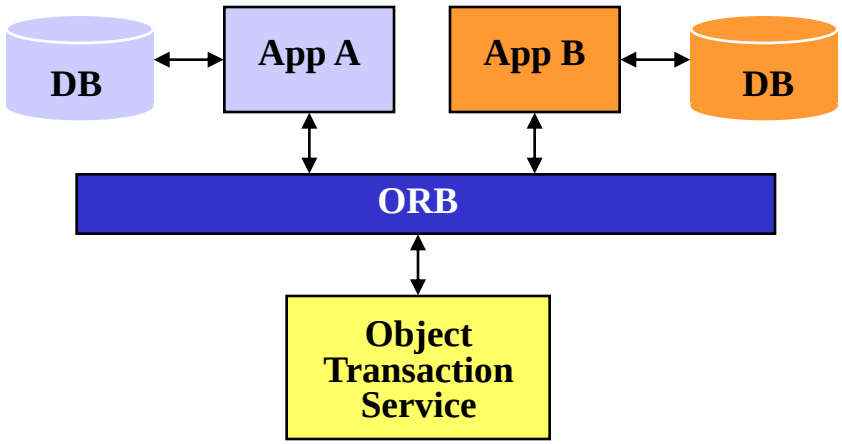
From Web Services Transaction (WS-Transaction) 9 August 2002

WS-Transactions

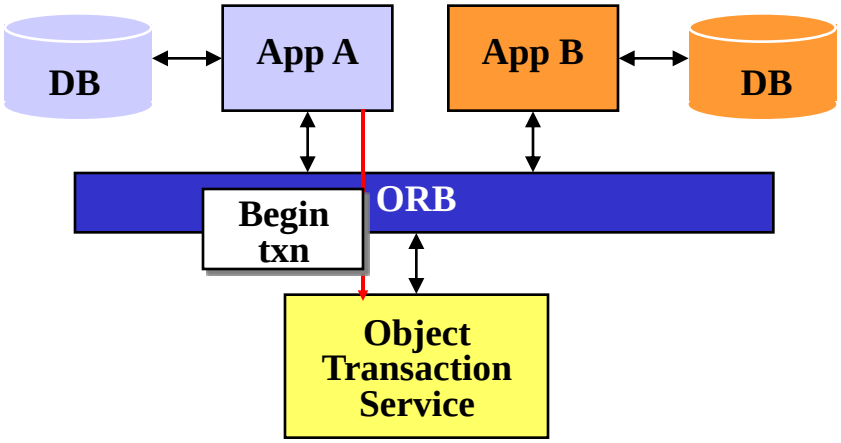


CORBA transactions (1)

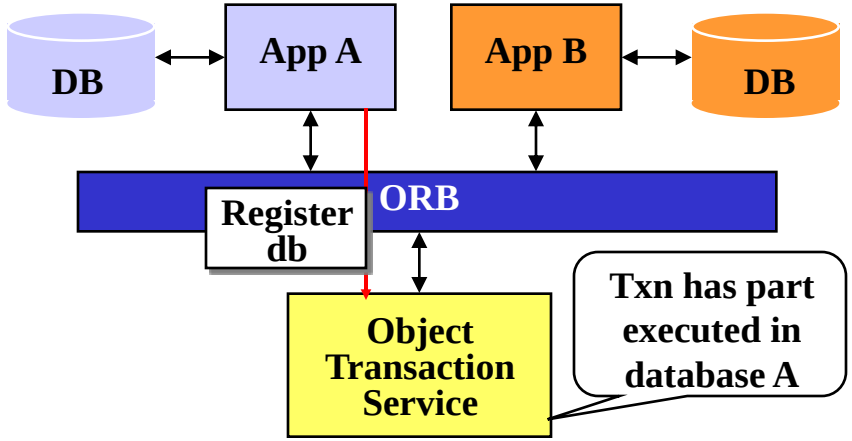
1) Assume App A wants to update databases A and B



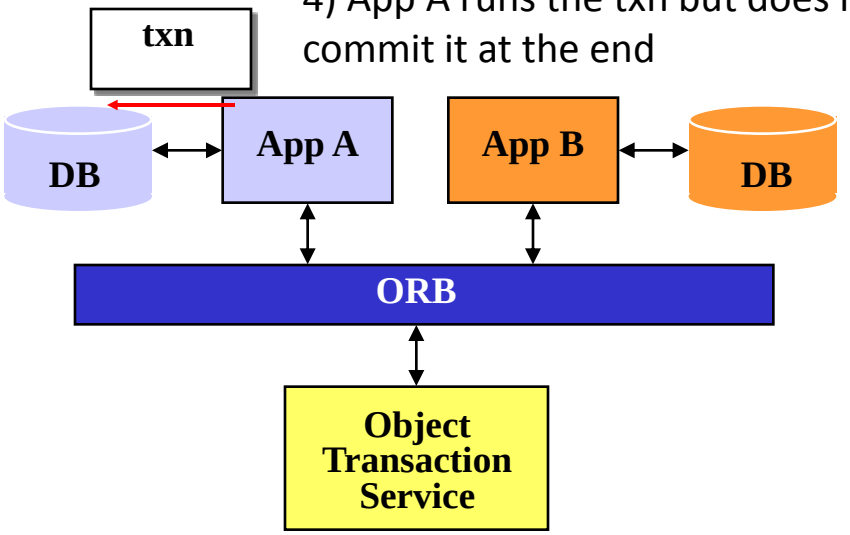
2) App A obtains a txn identifier for the operation



3) App A registers the database for that transaction

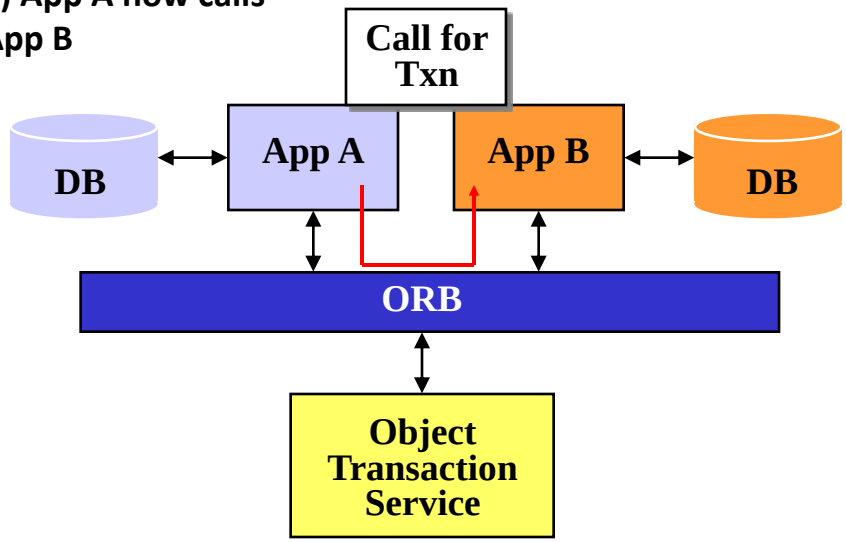


4) App A runs the txn but does not commit it at the end

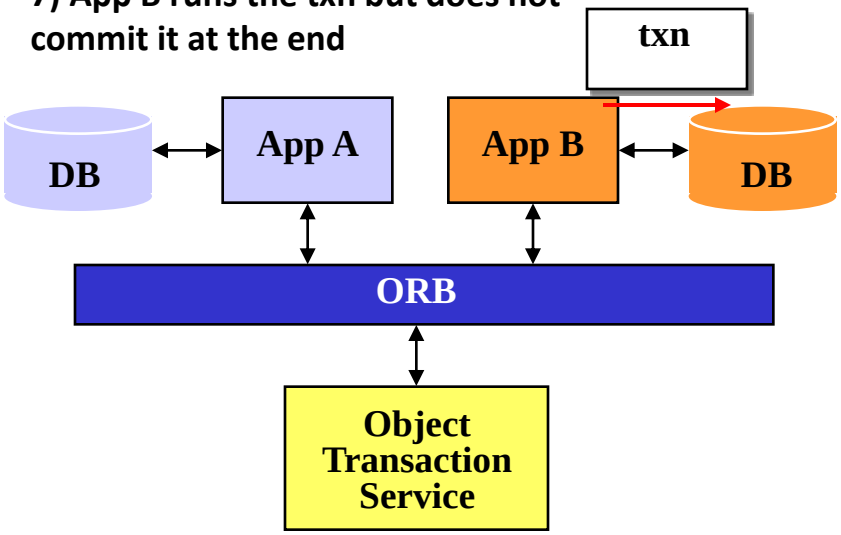


CORBA transactions (2)

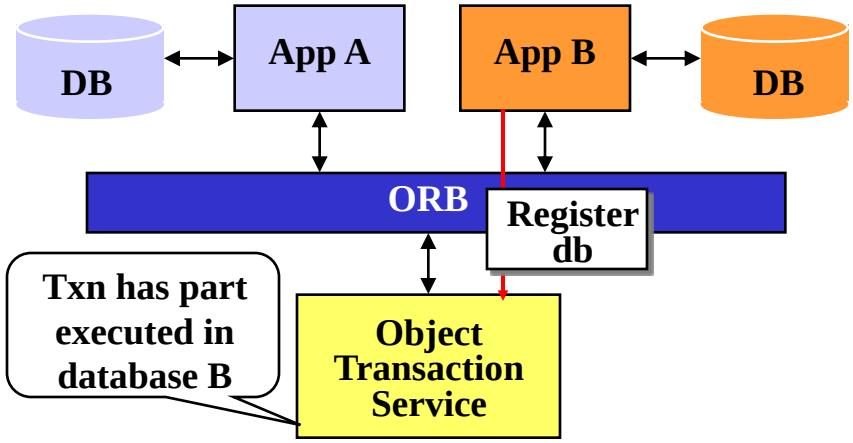
5) App A now calls App B



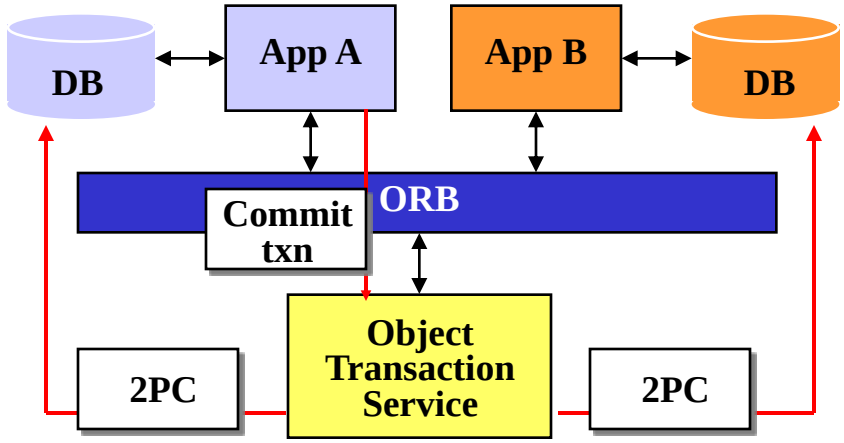
7) App B runs the txn but does not commit it at the end



6) App B registers the database for that transaction



8) App A request commit and the OTS runs 2PC



Summary WS-Transaction

- WS-Transaction is an example of how to apply the framework defined by WS-Coordination to define a specific protocol.
- WS-Transaction defines short lived **atomic transactions** standardizing the interfaces provided by traditional TP-monitor tools.
- However, in an Web services scenario, transactions may also take a long time to complete. For this case, WS-Transaction uses the notion of **business activity** and defines a protocol based on compensation (as opposed to locking) used to achieve distributed consensus on whether the results of a long-running message exchange should be made persistent.
- This standard defines the port types (WSDL interfaces) that must be implemented by each participant service depending on its role in the transaction (e.g., initiator, outcome listener). It also specifies the port types provided by the coordinator.
- The actual implementation of the operations corresponding to a commit, abort or compensate message are left unspecified as they are highly dependent on the business logic of the specific Web service.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

7 WSIF: Web Services Invocation Framework

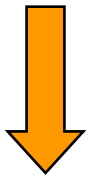
http://ws.apache.org/wsif

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/

SOA and web services

There is no problem in system **design** that cannot be solved by adding a level of indirection.

There is no **performance** problem that cannot be solved by removing a level of indirection.



Take advantage of Middleware but let the system decide what to use

- WS Invocation Framework
  - Use WSDL to describe a service
  - Use WSIF to let the system decide what to do when the service is invoked:
    - If the call is to a local EJB then do nothing
    - If the call is to a remote EJB then use RMI
    - If the call is to a queue then use JMS
    - If the call is to a remote Web service then use SOAP and XML
  - There is a single interface description, the system decides on the binding
  - This type of functionality is at the core of the notion of Service Oriented Architecture

Interfaces

- Web Services are intended to provide an standardized interface to conventional integration functionality
- SOAP provides an abstract, standardized way to exchange messages
- WSDL provides an abstract, standardized way to define interfaces
- UDDI provides an abstract and standardized way to look for services

- The problem is that the abstract description must always have a binding to a concrete implementation (e.g., RPC over HTTP)

- Web services lose some advantages if they always have to be used with a concrete implementation
  - Developers have to know the concrete implementation
  - It is difficult to change the concrete implementation once fixed
  - In most cases, the most generic implementation is chosen, which is typically the most expensive (XML format for SOAP messages)

- The ultimate goal of SOA is to use services as abstract concepts and ignore their concrete implementation (let the middleware take care of the protocol details)

The case for abstract interfaces

Fast prototyping

- Build a simple service in Java
- Test it and see how it works
- Replace it with a more robust EJB implementation incorporating additional features
- The change from prototype to final implementation is done without changing the interface and without requiring changes to the application using the service

Service evolution

- Replace an RPC based service with a queue based service for added persistence and delivery guarantees
- Include redundant service implementations working on the queue for added reliability
- The change can be made without affecting the client and application code

Alternative channels

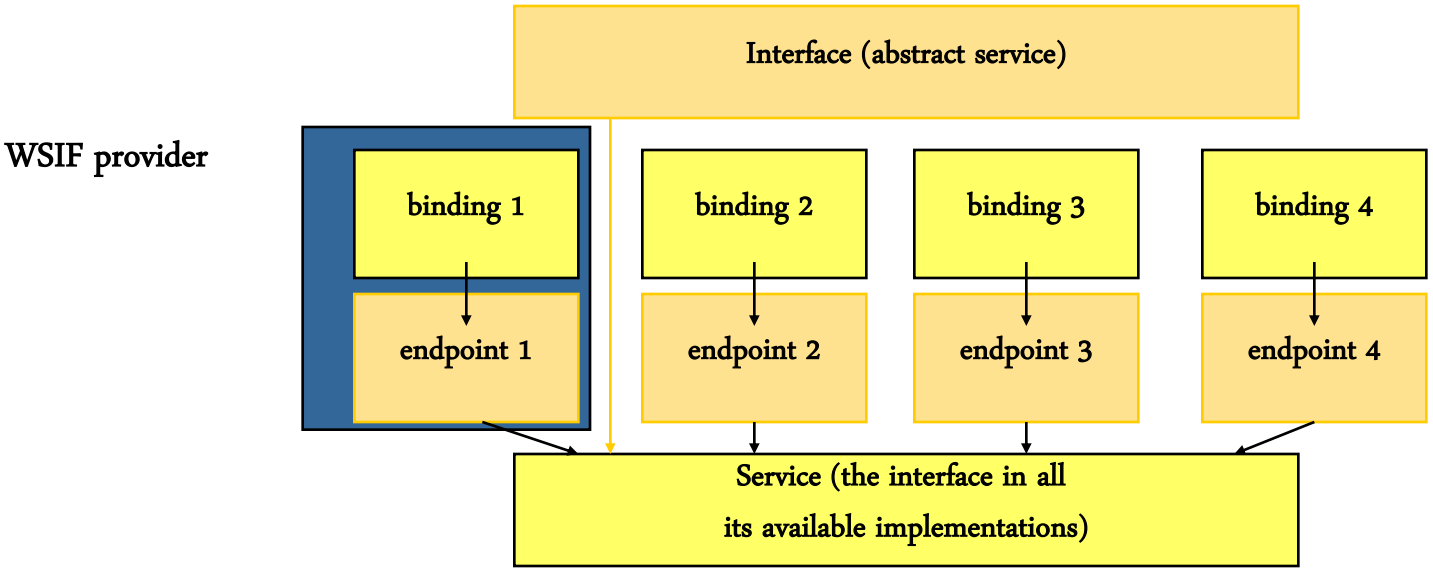
- For a given service, provide different access channels to be used according to Service Level Agreements
- Add alternative channels dynamically as requirements change

Loose coupling C/S

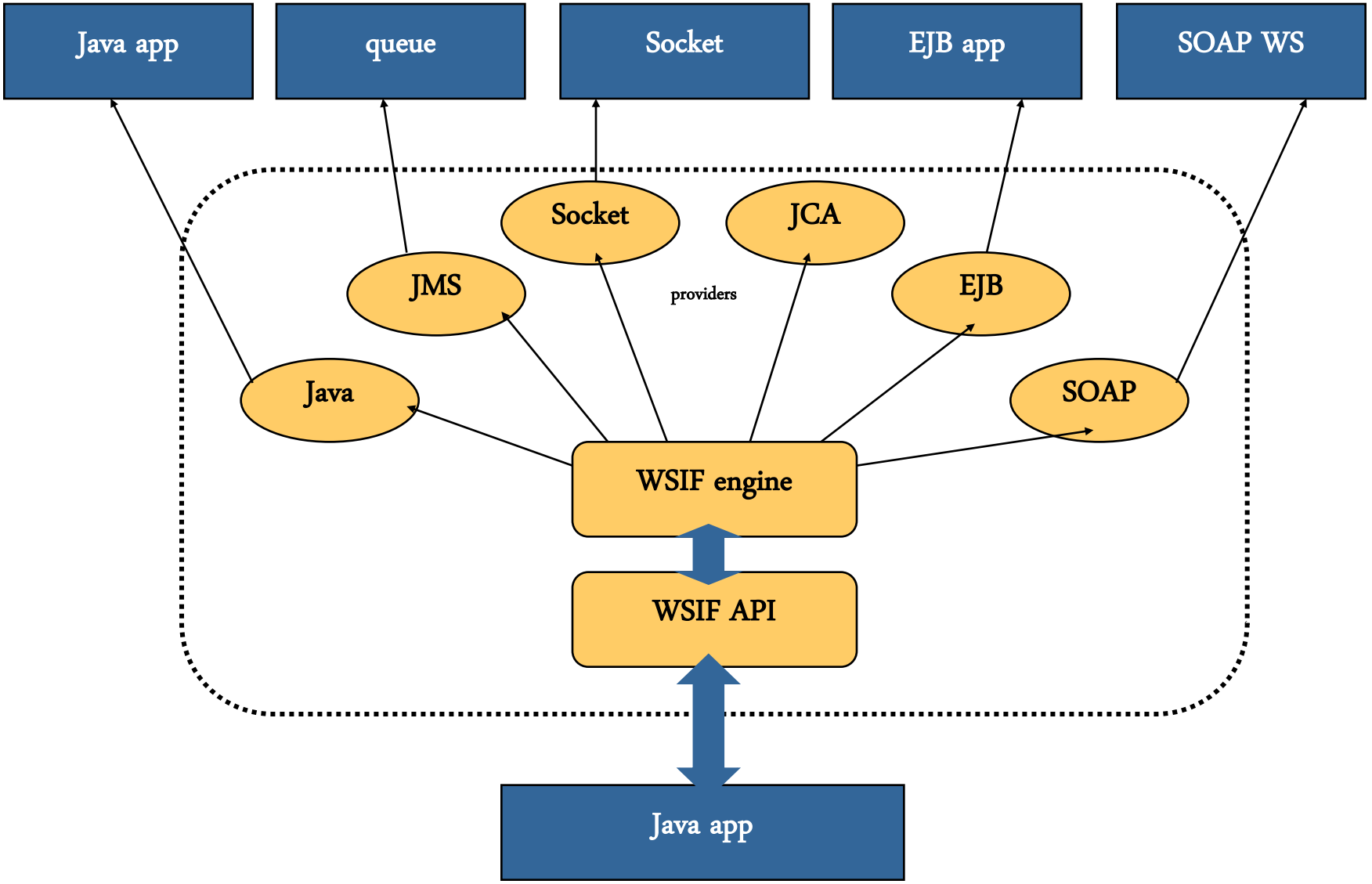
- The client invokes a service interface
- A dynamic binding determines at run time which concrete implementation of the abstract interface is to be used
- Which concrete implementation to use can change over time (e.g., based on autonomic functionality)

WSIF and WSDL

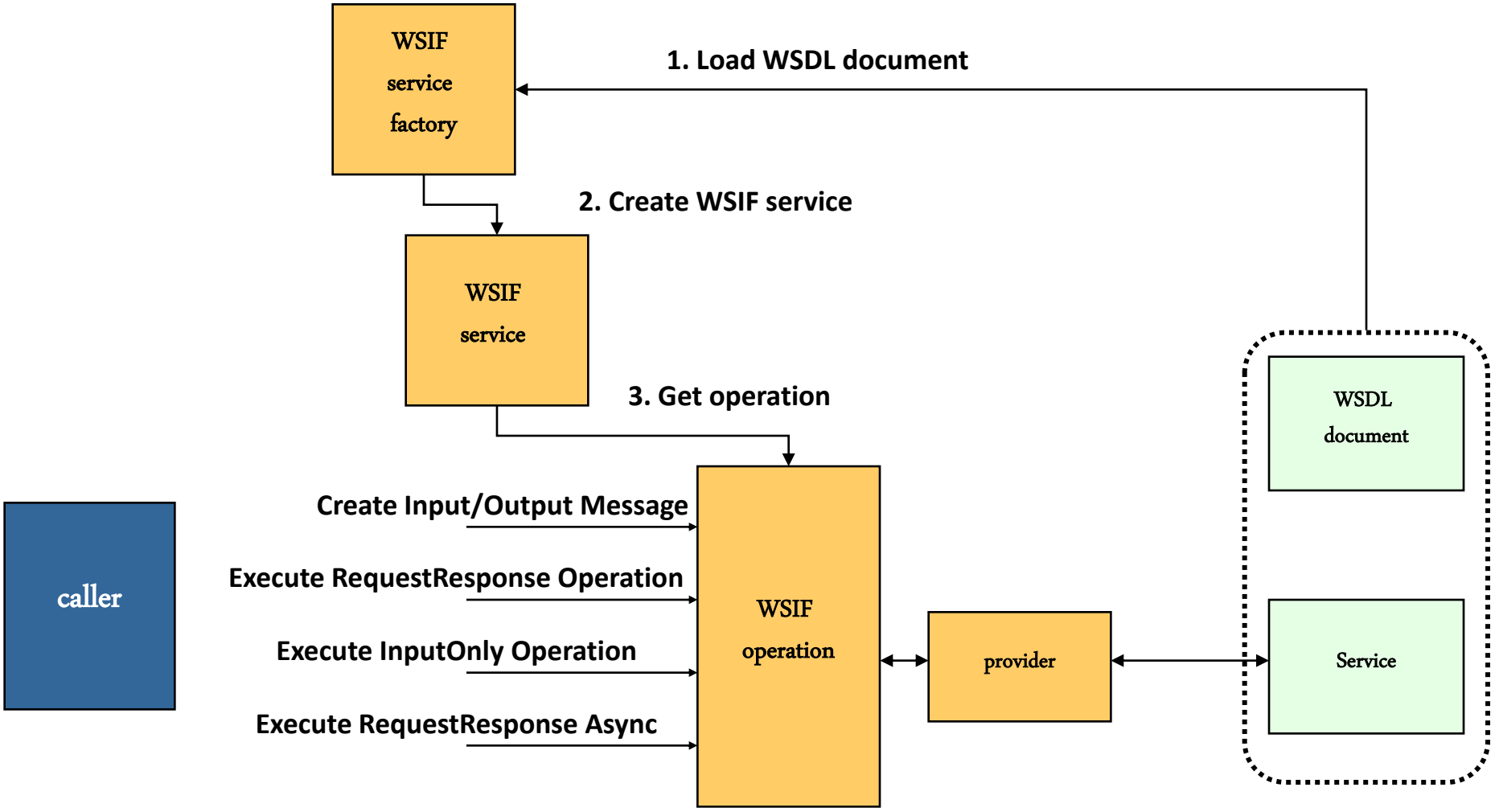
- WSIF has two parts:
  - A meta data description of the service (equivalent to the WSDL abstract service)
  - A set of providers implementing different concrete implementations of the interface
- A provider is a code module that implements a single WSDL binding and endpoint
- Providers use the J2SE JAR Service provider specification, making them discoverable at run time.



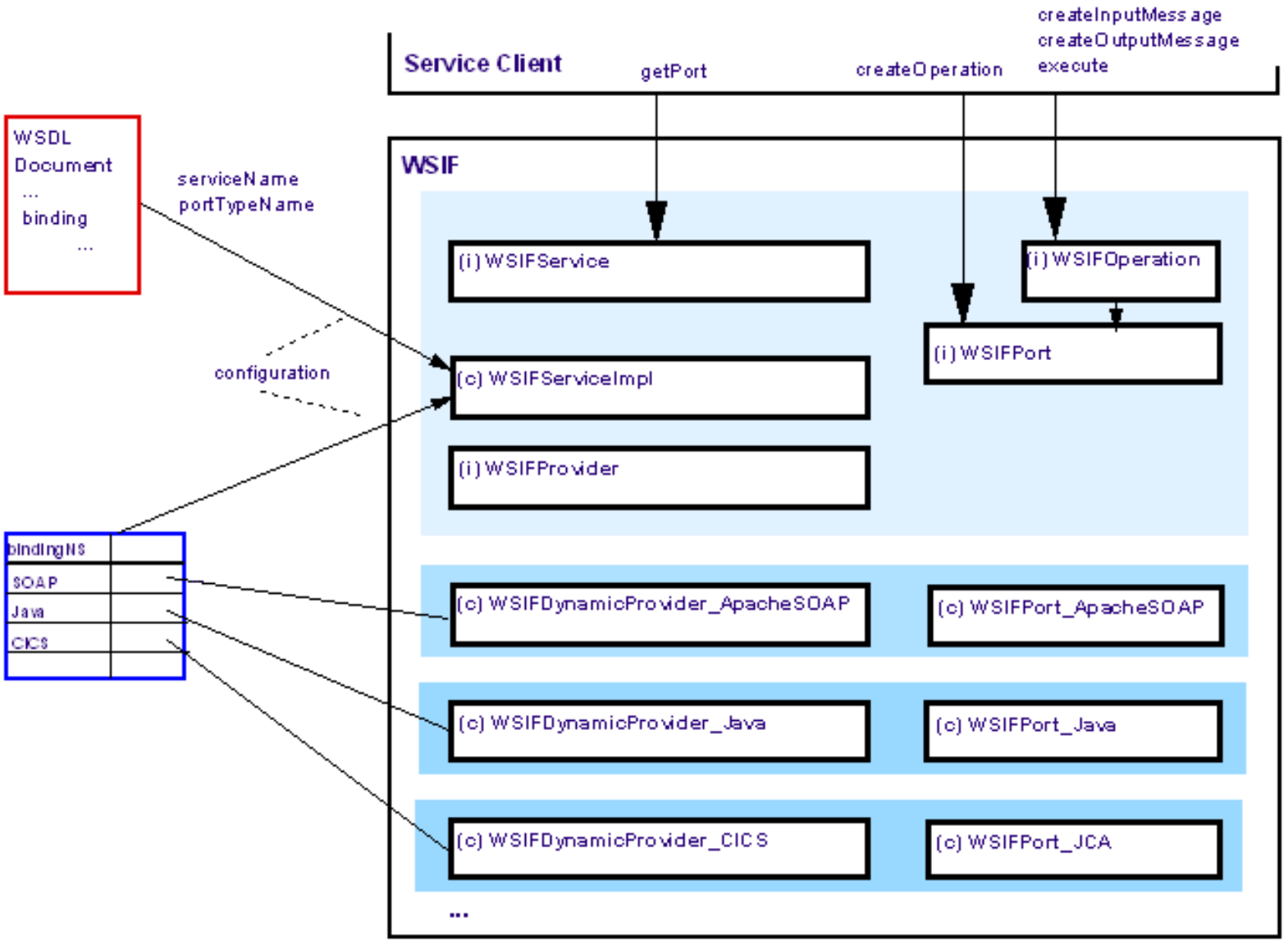
WSIF Structure



WSIF architecture



Basic interactions with WSIF



http://ws.apache.org/wsif

WSIF Use

- Create a Service Factory
- Use the Factory to read the WSDL describing the service to call and create a Service object that can be used to refer to the service
- If necessary, map the types used in the WSDL document to Java objects
- Use the Service object to create a Port object:
  - The port determines the binding
  - One can specify which one to use or choose a default
- Use the Port object to create the operation to invoke
  - Create the messages
  - Map the data to the messages
- Call the operation

- It is possible to invoke an operation asynchronously
  - By indicating a handler to be invoked when the response arrives
  - By simply updating the message output when the response arrives

- It is possible to use stubs for invocation (created at run time) through the interface of the Service object (service.getStub)