

Standard Software for Enterprise Resource Planning (ERP),

as well as CRM, HR, FI, ...

Lecturer: Prof. Dr. Ralf Möller

Lab classes: Rainer Marrone, Michael Wessel

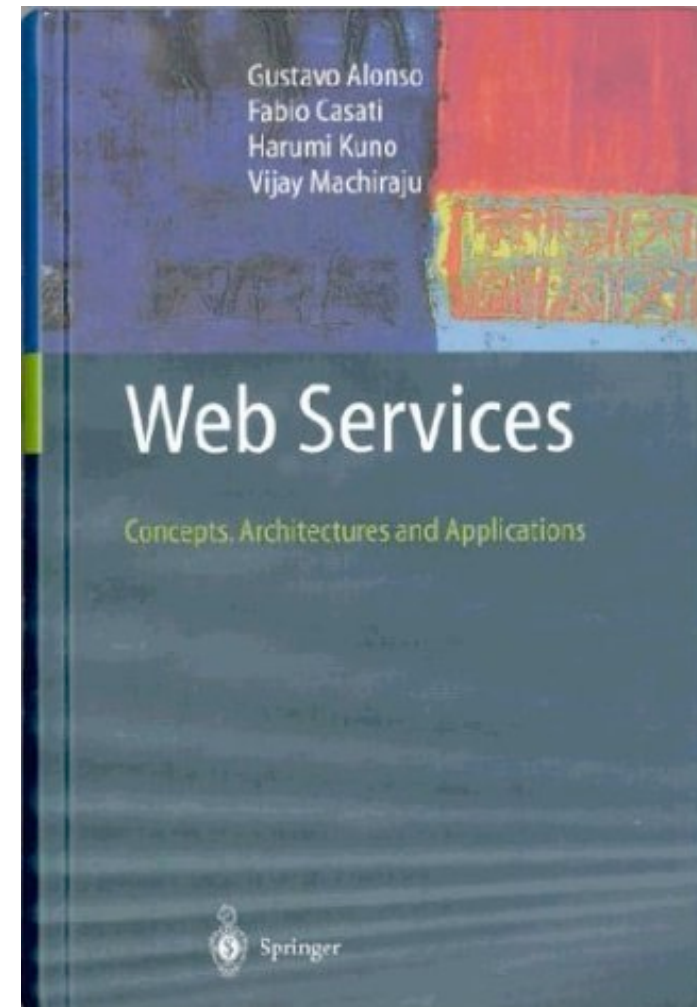
Lecture: Thursdays (90 minutes)

Lab classes: Fridays (60 minutes)

Prerequisite:

Lecture on ECommerce

This lecture is based on:





ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Web Services - Concepts, Architecture and Applications

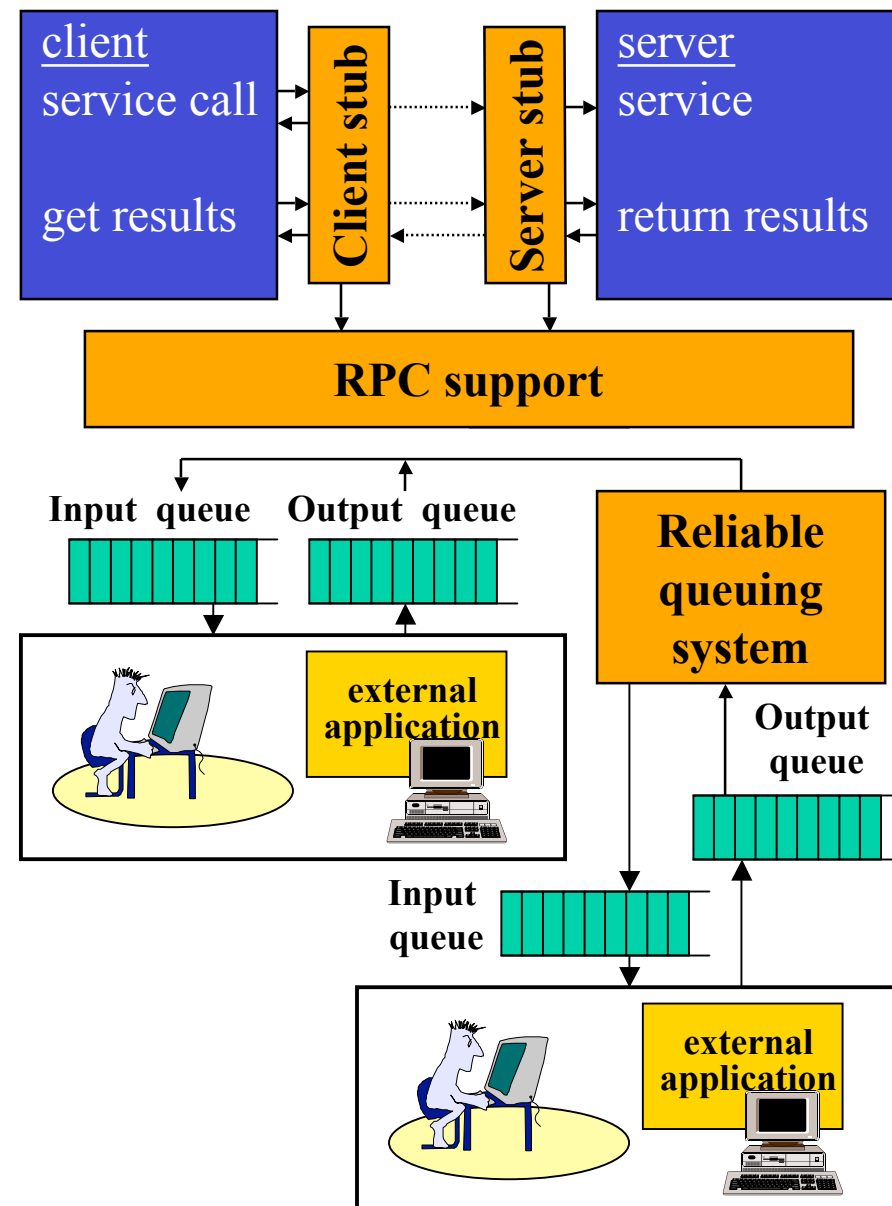
Part 3: Asynchronous middleware

Gustavo Alonso and Cesare Pautasso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch
<http://www.inf.ethz.ch/~alonso>

Message oriented middleware

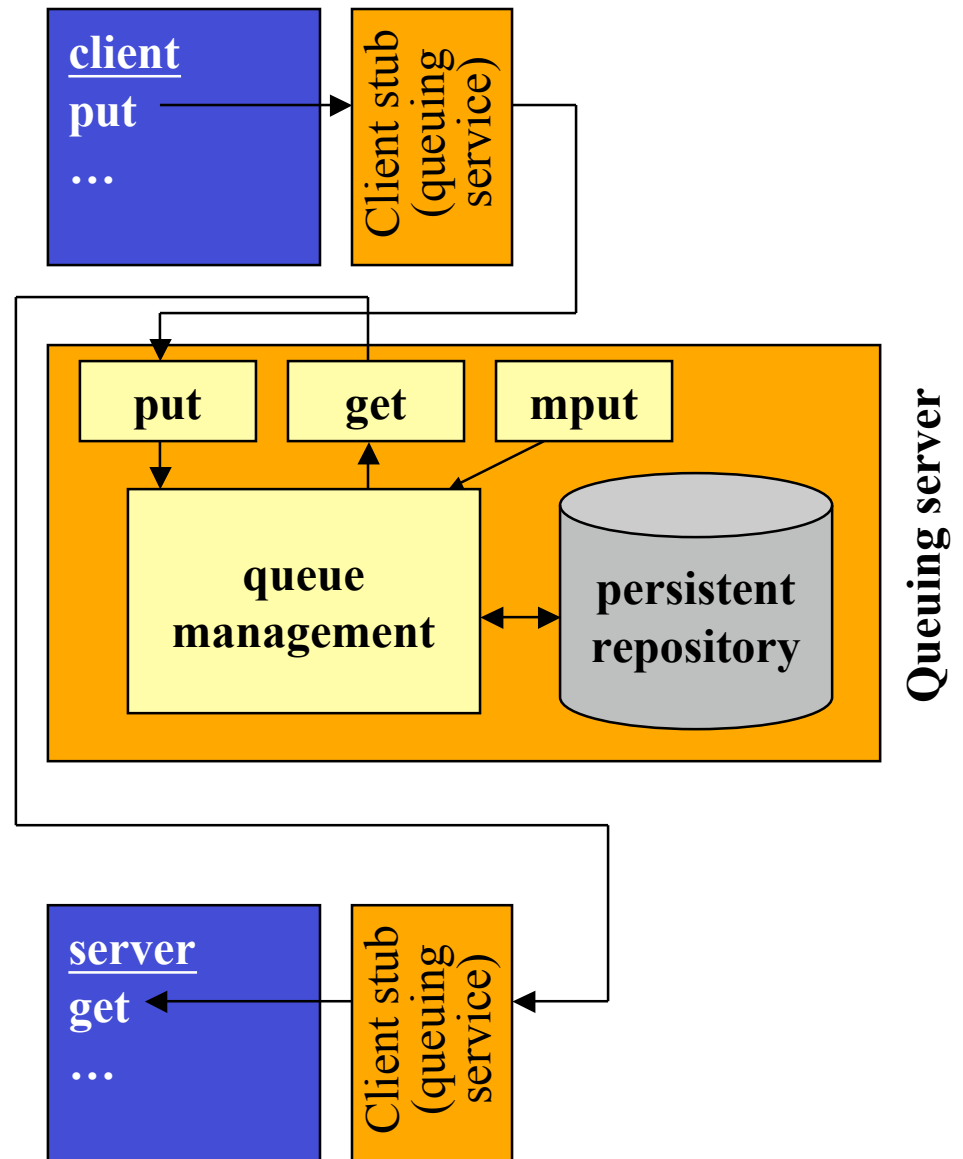
Queues in TP-Monitors

- ❑ Synchronous middleware: **tight coupling** between the caller and the called.
- ❑ Name and directory services help to minimize the effects of this coupling
- ❑ Considerable problems when trying to implement certain properties or manage the interaction between client and servers (fault tolerance, availability, etc.)
- ❑ Alternatives:
 - ➔ Asynchronous RPC: client makes a call that returns immediately; the client is responsible for making a second call to get the results
 - ➔ Reliable queuing systems (e.g., Encina, Tuxedo) where, instead of using procedure calls, client and server interact by exchanging messages.



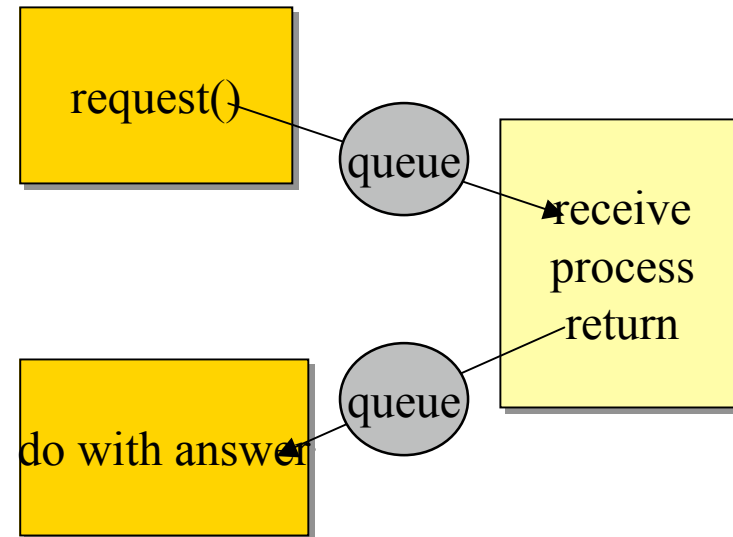
Queues in practice

- ❑ To access a queue, a client or a server uses the queuing services, e.g., :
 - ➔ put (enqueue) to place a message in a given queue
 - ➔ get (dequeue) to read a message from a queue
 - ➔ mput to put a message in multiple queues
 - ➔ transfer a message from a queue to another
- ❑ In TP-Monitors, these services are implemented as RPC calls to an internal resource manager (the reliable queuing service)
- ❑ These calls can be made part of transaction using the same mechanisms of TRPC (the queuing system uses an XA interface and works like any other resource manager)

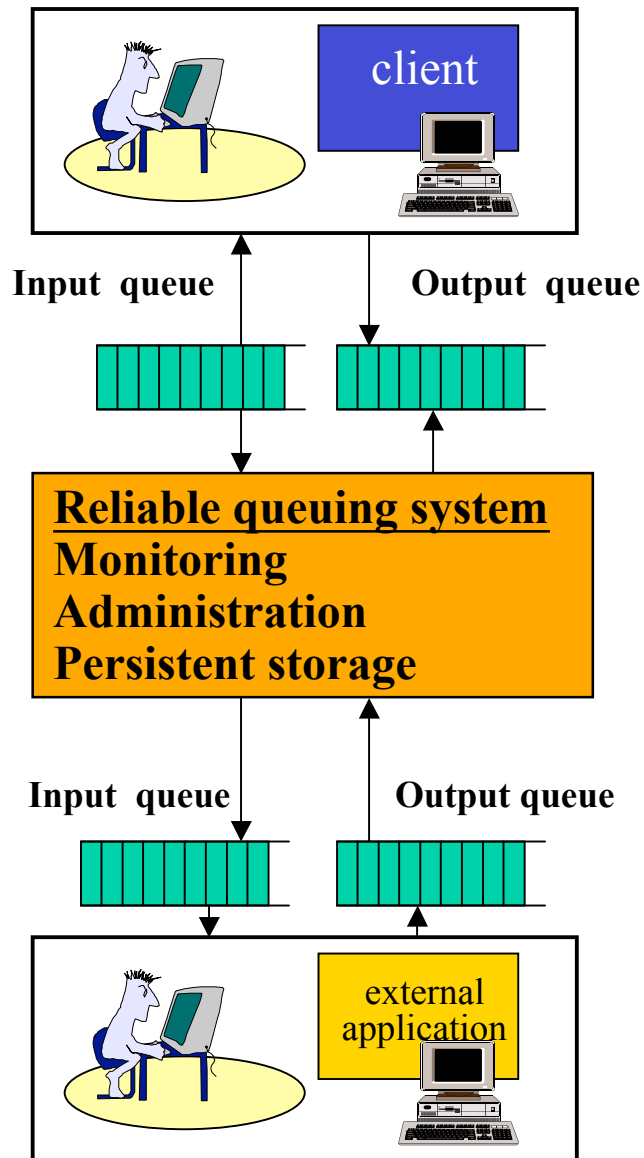


Reliable queuing

- ❑ Reliable queuing turned out to be a very good idea and an excellent complement to synchronous interactions:
 - ➔ Suitable to modular design: the code for making a request can be in a different module (even a different machine!) than the code for dealing with the response
 - ➔ It is easier to design sophisticated distribution modes (multicast, transfers, replication, coalescing messages) and it also helps to handle communication sessions in a more abstract way
 - ➔ More natural way to implement complex interactions



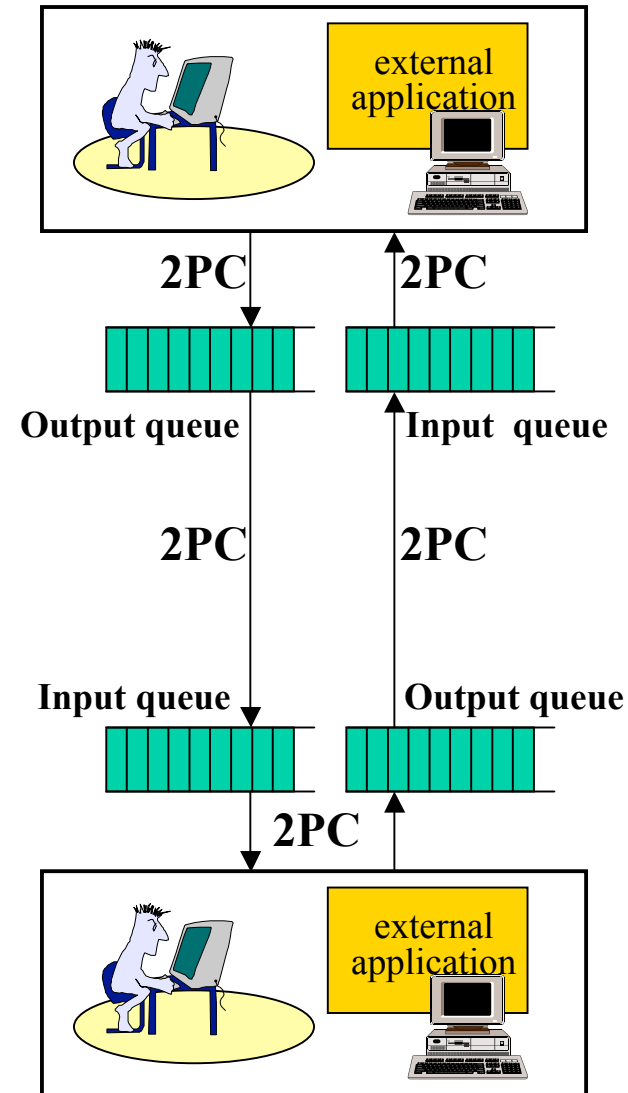
Queuing systems



- ❑ Asynchronous interactions
- ❑ Queues store messages transactionally, guaranteeing that messages are there even after failures occur
- ❑ Advantages over traditional solutions in terms of **fault tolerance** and overall **system flexibility**: applications do not need to be there at the time a request is made!
- ❑ Queues provide a way to communicate across heterogeneous networks and systems while still being able to make some assumptions about the behavior of the messages.

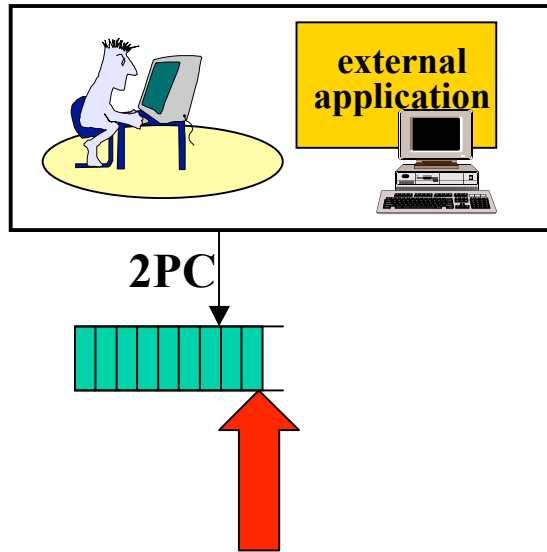
Transactional queues

- ❑ Persistent queues are closely tied to transactional interaction:
 - ➔ to send a message, it is written in the queue using 2PC
 - ➔ messages between queues are exchanged using 2PC
 - ➔ reading a message from a queue, processing it and writing the reply to another queue is all done under 2PC
- ❑ Significant overhead but it also provides considerable advantages.
 - ➔ The overhead is not that important with local transactions (writing or reading to a local queue).
- ❑ Using transactional queues, the processing of messages and sending and receiving can be tied together into one single transaction so that atomicity is guaranteed. This solves a lot of problems!



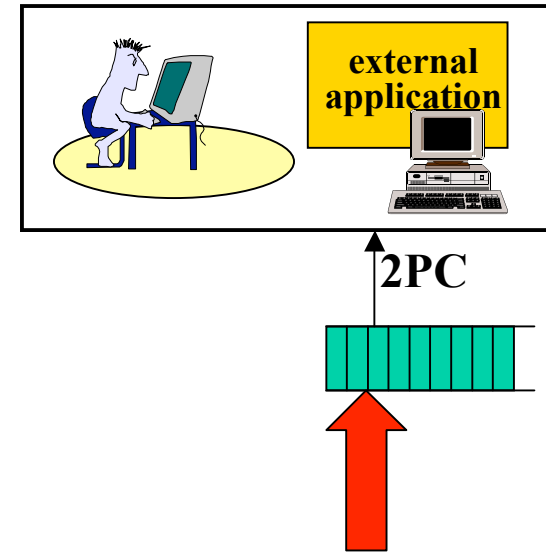
Problems solved (I)

SENDING



Message is now **persistent**. If the node crashes, the message remains in the queue. Upon recovery, the application can look in the queue and see which messages are there and which are not. Multiple applications can write to the same queue, thereby “**multiplexing**” the channel.

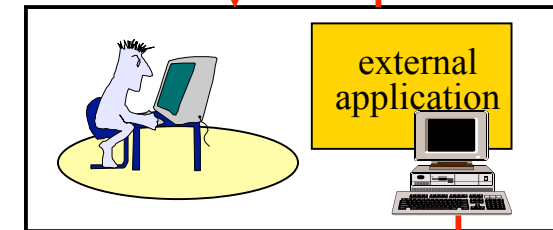
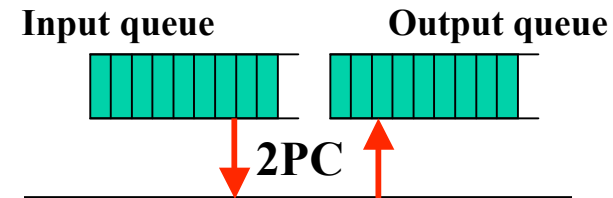
RECEIVING



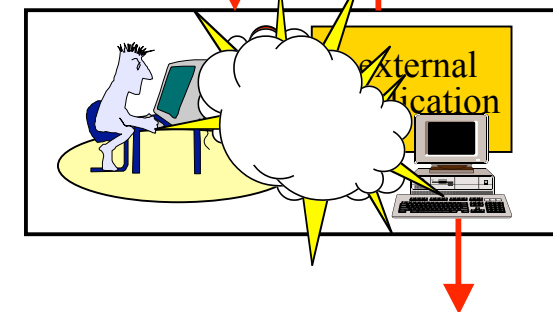
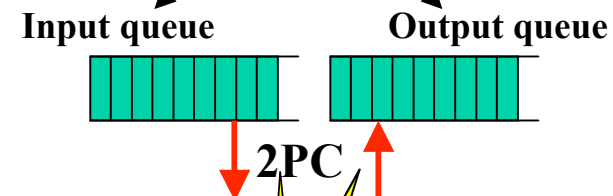
Arriving messages remain in the queue. If the node crashes, messages are not lost. The application can now take its time to process messages. It is also possible for several applications to read from the same queue. This allows to implement **replicated services**, do load balancing, and **increase fault tolerance**.

Problems solved (II)

- ❑ Bundle within a single transaction reading a message from a queue, interacting with other systems, and writing the response to a queue. If a failure occurs, in all scenarios consistency is ensured:
 - ➔ if the transaction was **not completed**, any interaction with other applications is undone and the reading operation from the input queue is not committed: the message remains in the input queue. Upon **recovery**, the message can be processed again, thereby achieving exactly-once semantics.
 - ➔ If the transaction was completed, the write to the output queue is committed, i.e., the response remains in the queue.
 - ➔ If replicated services are used, if one fails and the message remains in the input queue, it is safe for other services to take over this message.

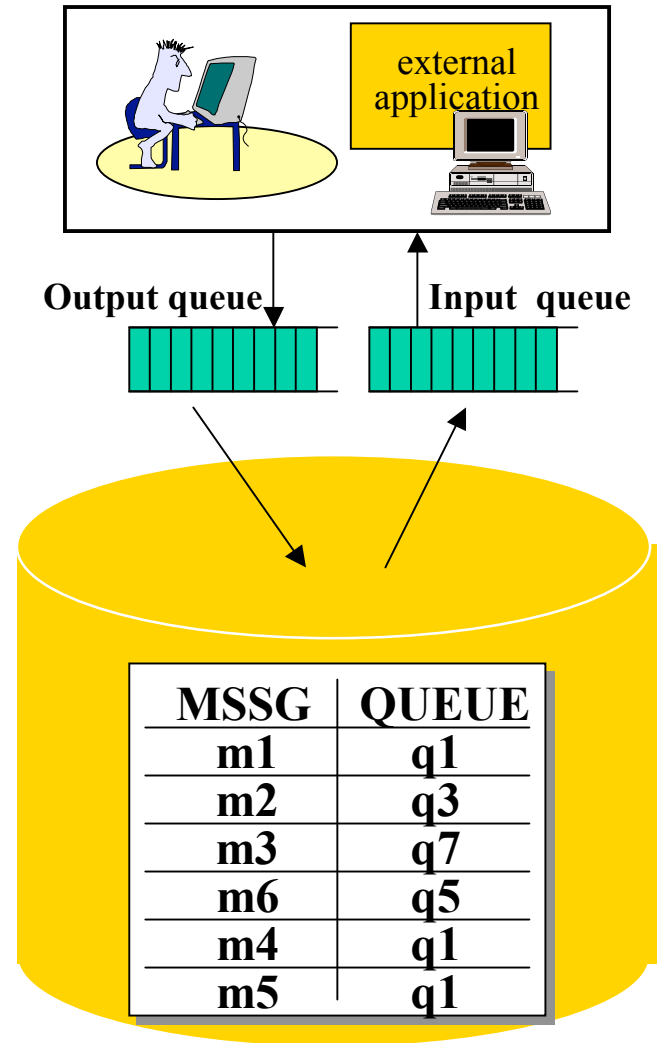


Message is either read or written



Implementation

- ❑ Persistent queues can be implemented as part of a database since the functionality needed is exactly that of a database:
 - ➔ a transactional interface
 - ➔ persistence of committed transactions
 - ➔ advanced indexing and search capabilities
- ❑ Thus, messages in a queue become simple entries in a table. These entries can be manipulated like any other data in a database so that applications using the queue can assign priorities, look for messages with given characteristics, trigger certain actions when messages of a particular kind arrive ...

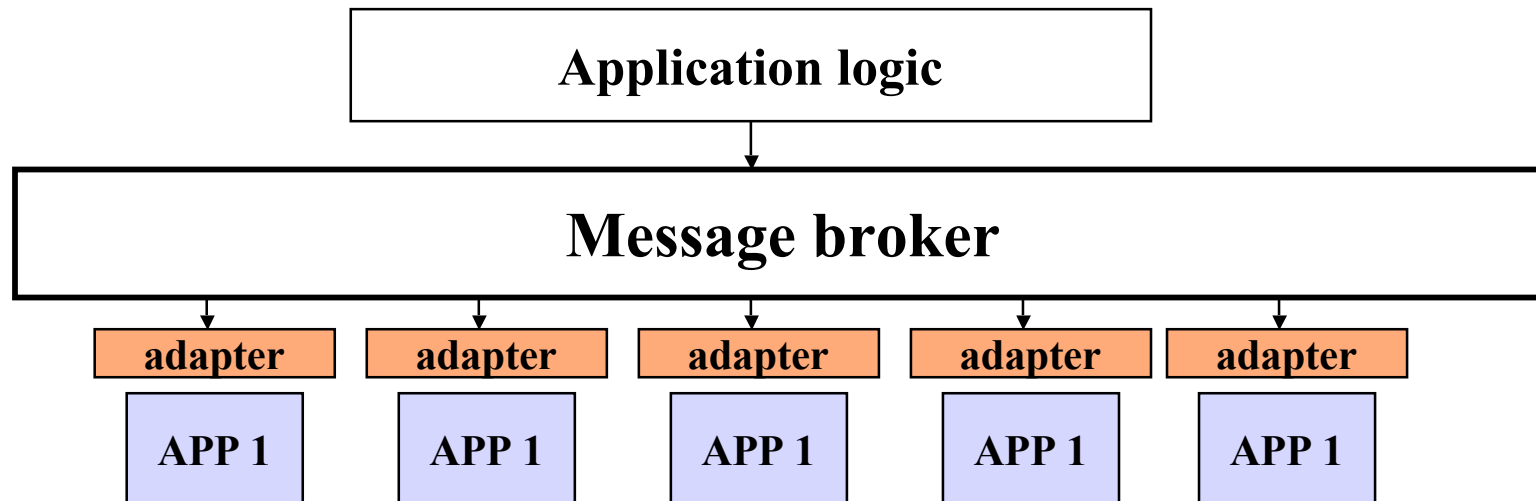


Advantages of queues in EAI

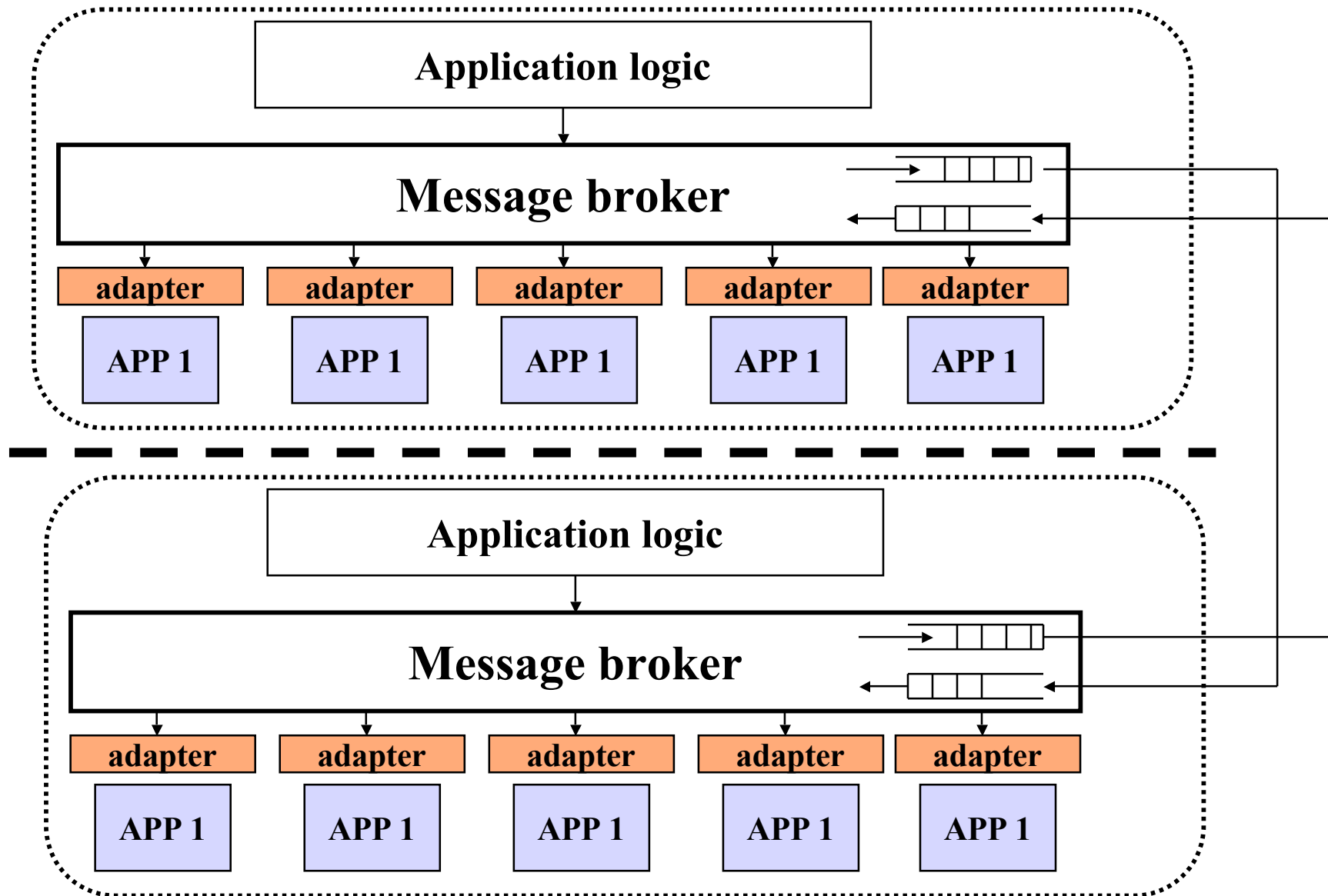
- ❑ Queuing management systems **completely detach the two sides of an interaction** as there is no direct call but a document exchange
- ❑ This makes the development of both sides of the interaction (e.g., client and server) **completely independent and very flexible**
- ❑ The interaction can be defined in terms of the document to exchange rather than the interface to invoke. This is a **more natural approach in the business world** where interactions tend to be document based
- ❑ From the scheduling point of view, it is much **easier to deal with messages than with synchronous invocations.**
- ❑ Technically, the infrastructure necessary to deal with messages is simpler than the infrastructure necessary to deal with synchronous calls. Especially if there are no fancy features added to the queues.
- ❑ Queues also have the advantage of being easily adaptable to communicate with external systems, even systems outside the organization (e.g., plug a queue to an e-mail address)
- ❑ When working with messages, the effort in describing the interaction goes into the description of the message or document within the message. This can be done without knowledge of how the message will be processed.

Message brokers

- ❑ Queuing management systems take care only of implementing queues and moving messages from one place to another. It is not possible to associate any logic to the queues.
- ❑ Message brokers are advanced queuing systems that are capable of associating application logic to the queues so that the queues no longer act as passive transmitter of messages but can **actively process, transform, or route the message according to business rules** associated with the queue.
- ❑ Plain queuing systems allow to implement asynchronous interaction. Message brokers allow to transform business processes into sequence of message exchanges between different applications, exactly what is needed for EAI.



Message brokers across organizations



Limitations of message brokers

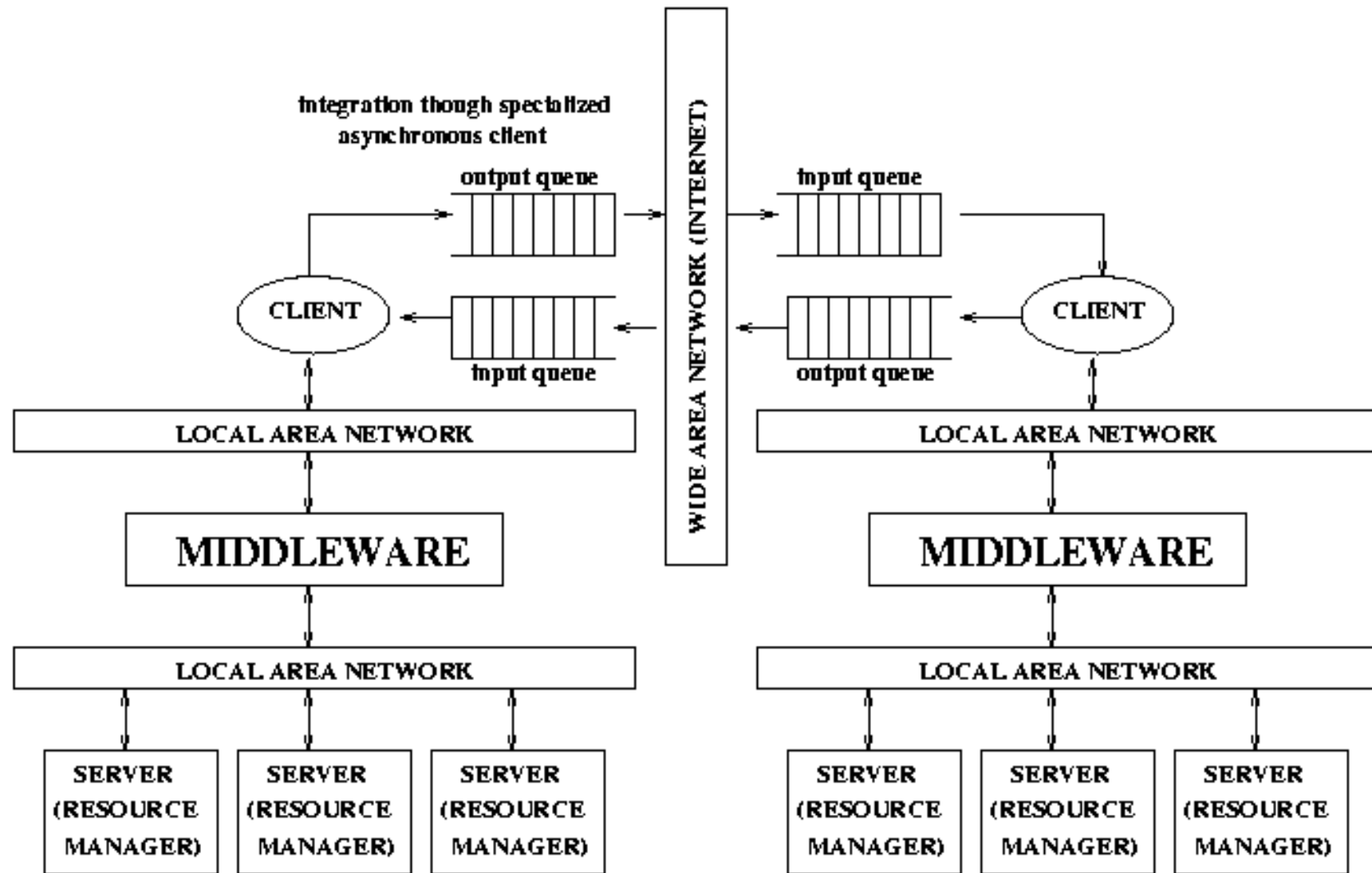


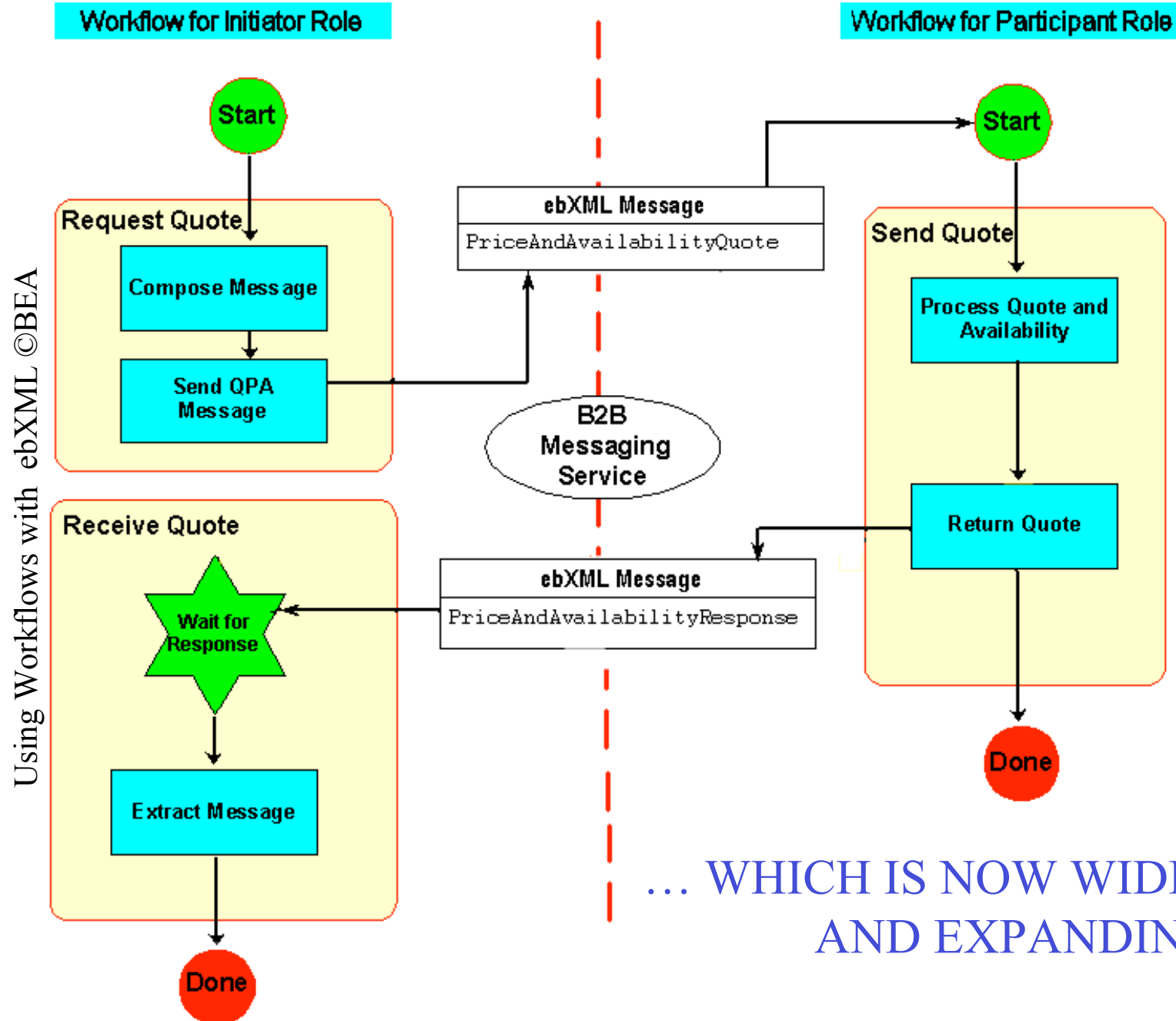
- ❑ Message brokers are very efficient and very flexible infrastructures for EAI provided:
 - ➔ messages are not too big (granularity has to be fine)
 - ➔ the mapping business process -> queues is well understood and well documented

- ❑ If one of these two constraints are not observed, problems might arise:
 - ➔ The implementation of queuing systems and message brokers is largely still based on the same technology as conventional middleware (RPC). Such procedural or method interfaces are not made for passing large objects (e.g., a large document) but mostly basic types and references.
 - ➔ If data flow is through references, it is outside the control of the system and some properties may be difficult to enforce (reliability, transactions, etc.)
 - ➔ Associating logic to the queue can be very helpful in many applications but distributing the logic of a business process across many queues makes the process very difficult to understand and there might not be a global description of the process.
 - ➔ Usually, it is not a good idea to use a message broker in the same way one would use a workflow engine due to the lack of global view over what is happening.

Message oriented business exchanges

Old fashioned e-commerce ...





Message based interaction

- ❑ Advantages over synchronous systems, particularly when the interaction involves different organizations and typical business transactions (where the response might not be immediate).
- ❑ Less of a client server flavor and more of an **exchange of information between partners**, this affects not only the way the interaction is implemented (as interconnected business processes) but also the technology used for that purpose.
- ❑ If the messages are **standardized**, then it is possible to create off-the-shelf systems that can process such messages. This was the original goal of EDI but the technology was not yet there. The Internet, XML, and the whole notion of Web services will probably now help to make this goal a reality at a lower cost and with less effort than was required before.
- ❑ Message based interaction is behind many of the most comprehensive models for electronic commerce (ebXML, xCBL, etc.) and will certainly have a decisive influence on how SOAP, WSDL and UDDI will evolve in the future.



ebXML Architecture

Anne Thomas Manes
Sun Microsystems
atm@sun.com



Agenda

- ❑ What is ebXML?
- ❑ Architecture Overview
- ❑ Architecture Details
- ❑ SOAP and UDDI
- ❑ Roadmap
- ❑ How to get involved



What is ebXML

- ❑ ebXML = Electronic Business XML
- ❑ Global Standard for electronic business
- ❑ ebXML enables anyone, anywhere to do business with anyone else over the Internet
- ❑ Specifically designed to support SME
- ❑ Complementary to existing B2B initiatives (UDDI, RosettaNet, TradeXchange, etc.)

An end-to-end B2B XML Framework

ebXML Vision



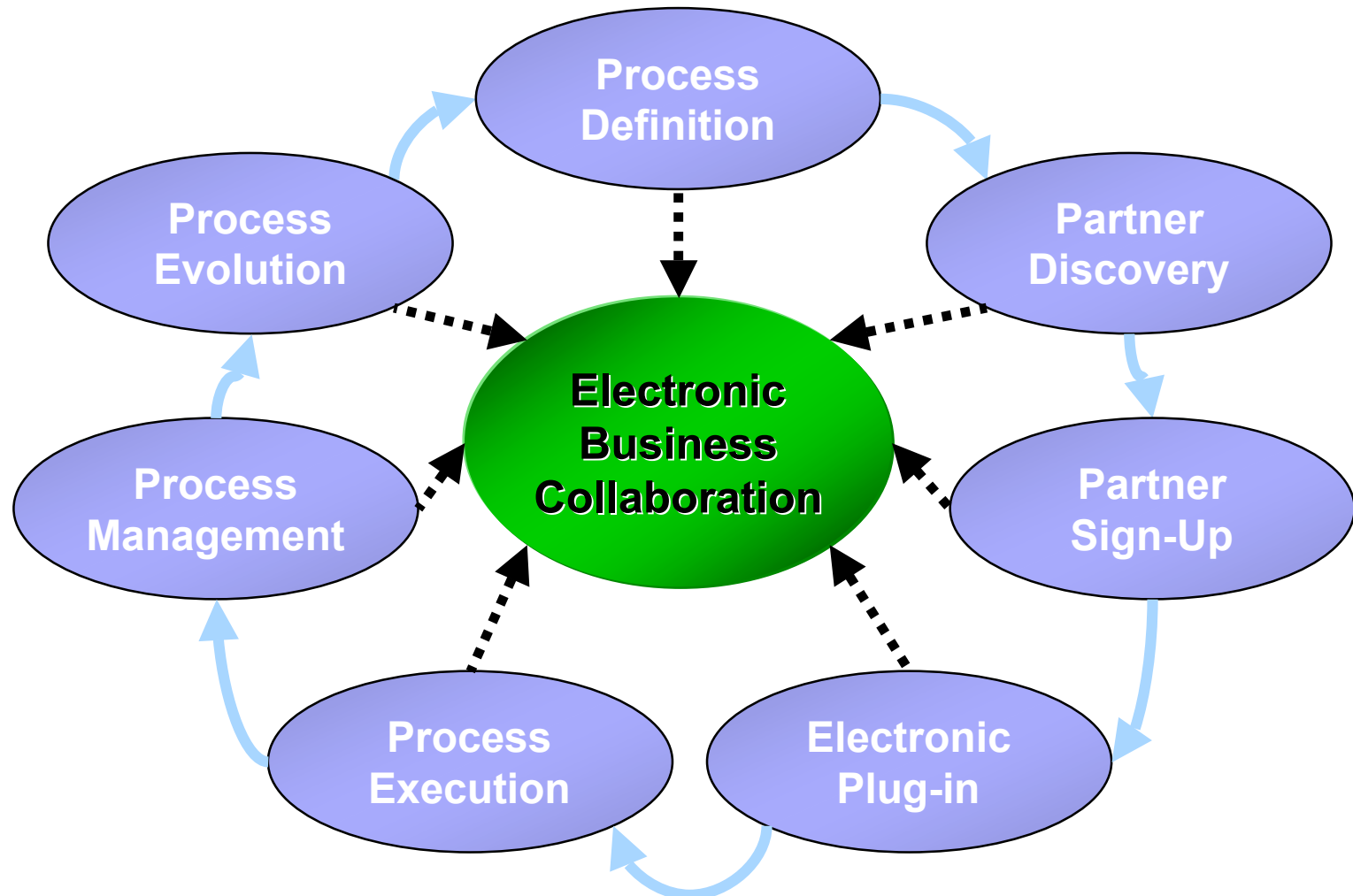
- ❑ A global electronic market place where enterprises of any size, anywhere can:
 - ➔ Find each other electronically
 - ➔ And conduct business
 - Using XML messages
 - According to standard business process sequences
 - With clear business semantics
 - According to standard or mutually agreed trading partner protocol agreements
 - Using off the shelf purchased business applications



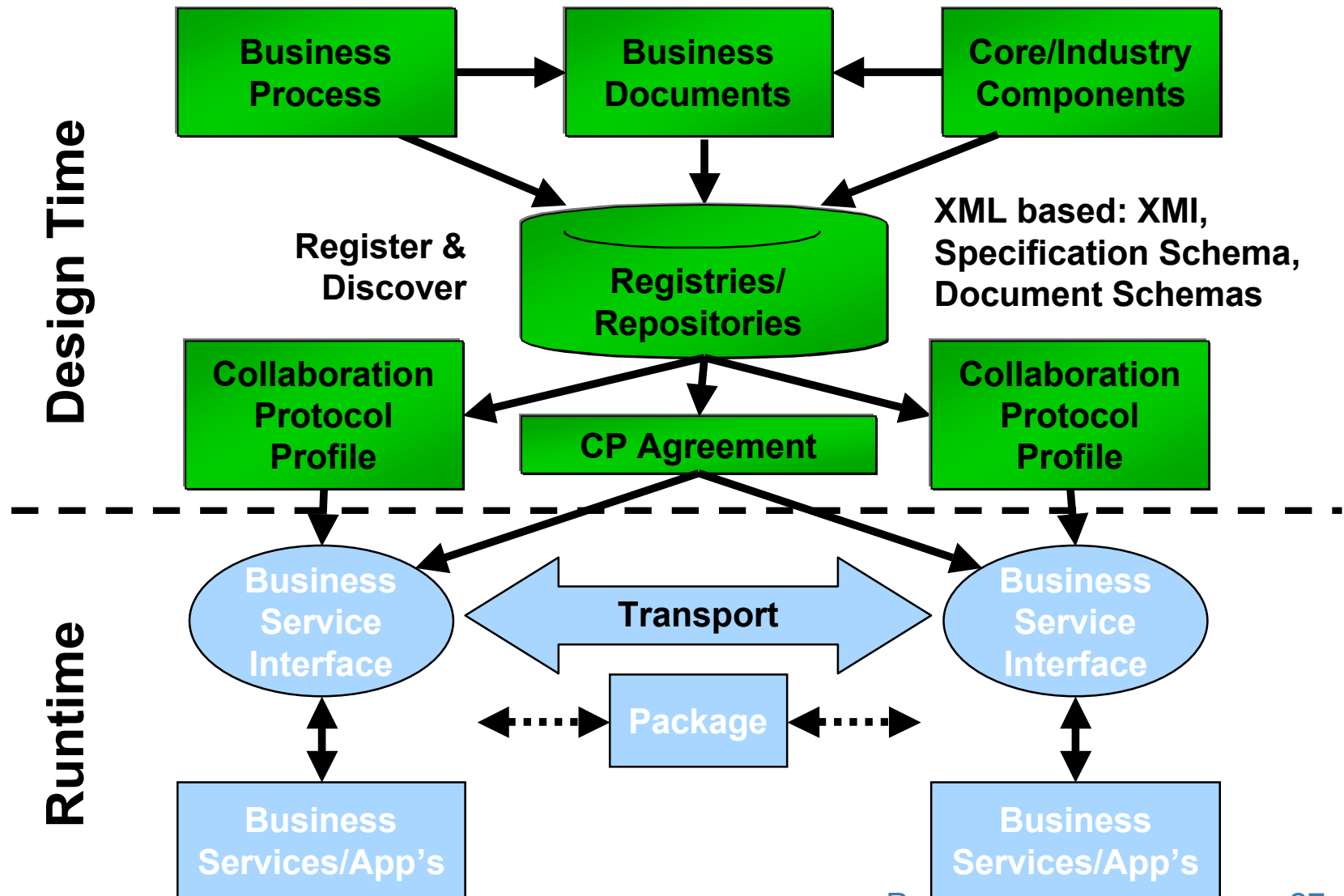
B2B Collaboration

- ❑ B2B collaboration requires more than just an XML protocol and a service registry
- ❑ You have to deal with
 - ➔ Business semantics
 - ➔ Negotiating terms and conditions
 - ➔ Interoperability
 - ➔ Security and Privacy
 - ➔ Reliability
- ❑ ebXML provides concrete specifications to enable dynamic B2B collaborations

B2B Collaboration Process



ebXML Architecture



Usage Example

Company X



ebXML BO Library
ebXML BP Model

Build local system implementation

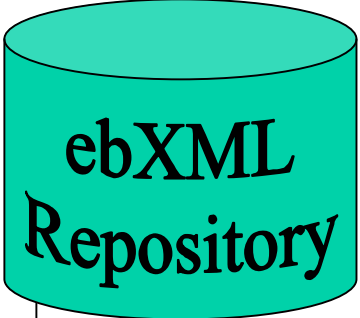
DO BUSINESS!



Company Y

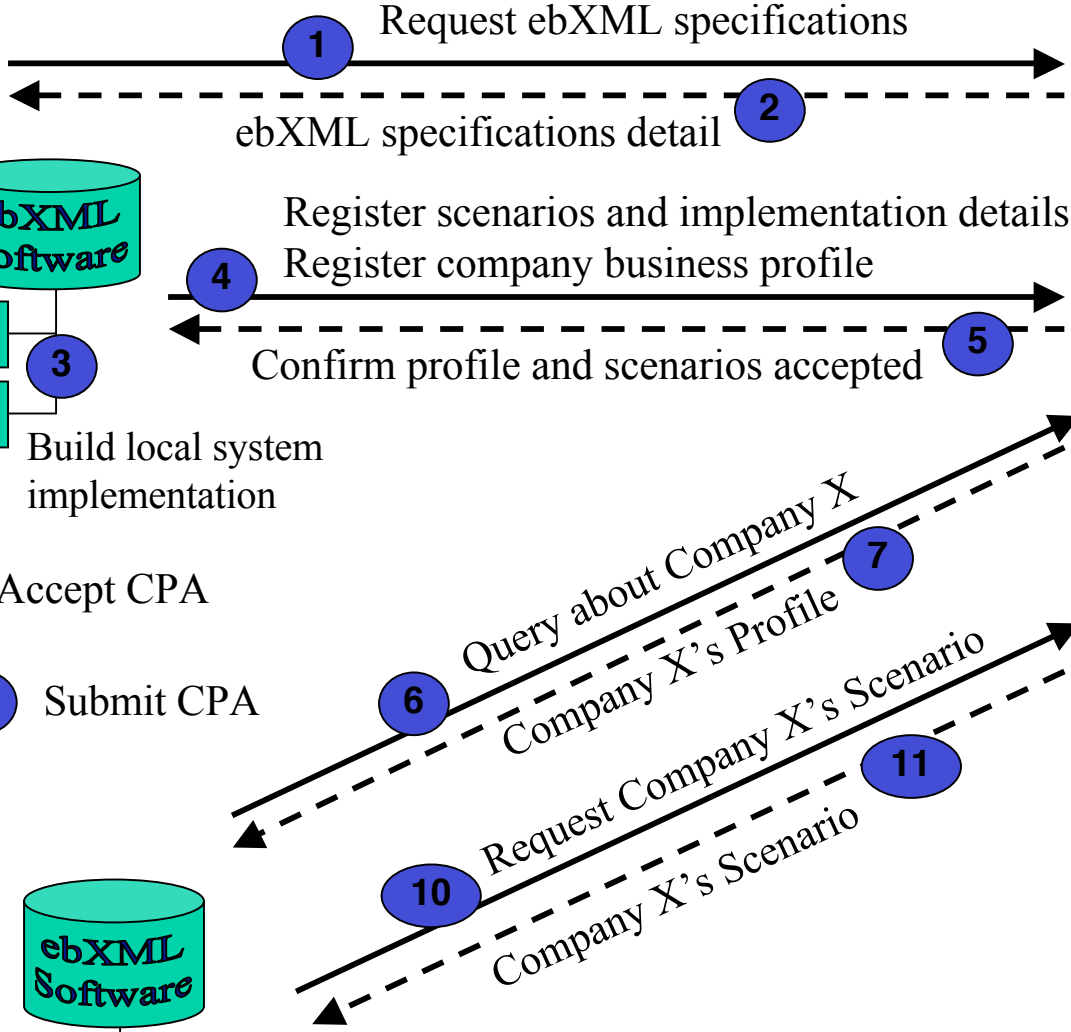


ebXML BO Library
ebXML BP Model



Specifications
Profiles
Scenarios

INDUSTRY INPUT





Company Profile

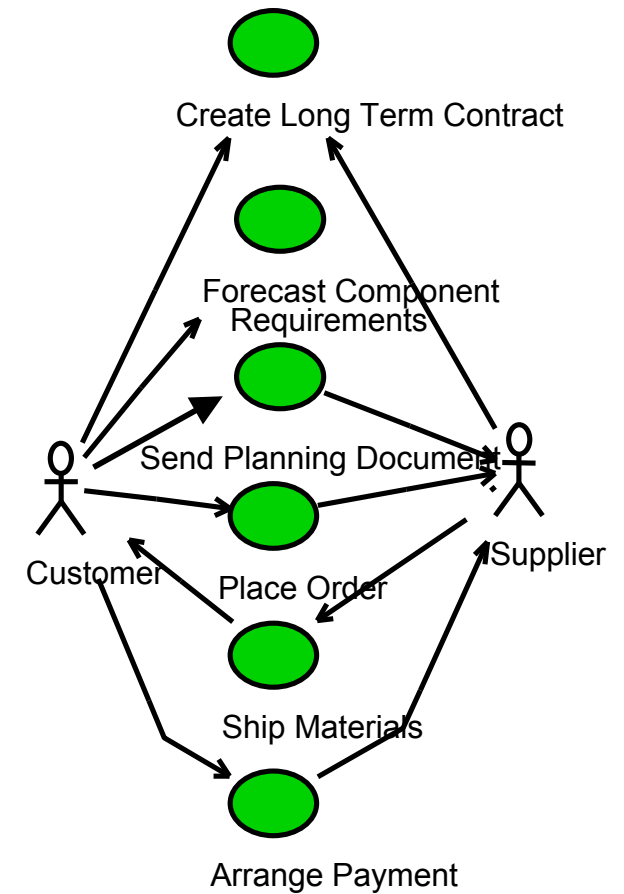
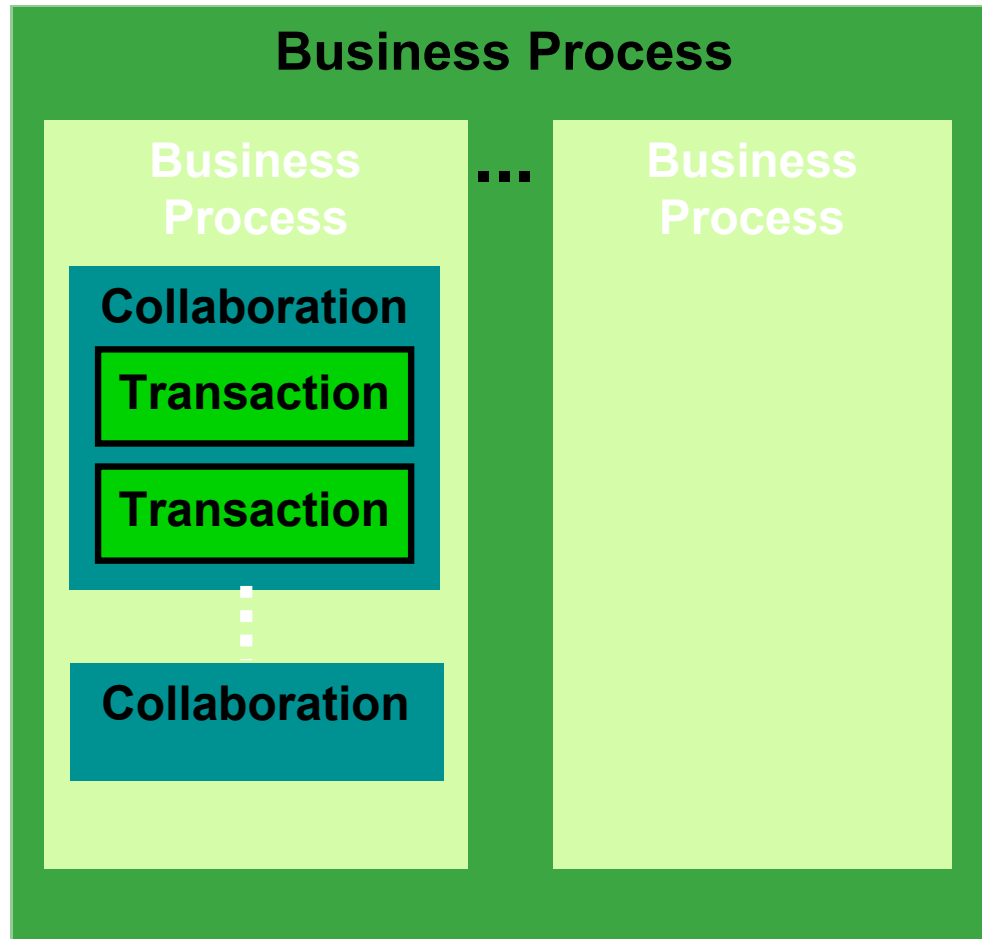
- ❑ Collaboration Protocol Profile
 - ➔ Defined using ebXML Specification Schema
 - ➔ Concrete specification of your ebusiness offerings
 - Business scenarios you support
 - Service interfaces you implement
 - Document formats exchanged
 - Technical requirements/options (protocols, security, reliability)
- ❑ Composed of
 - ➔ Business process models
 - ➔ Information models
 - ➔ Context rules



Business Scenarios

- ❑ Often defined by Industry Groups
 - ➔ Standard business scenarios remove the need for prior agreements among trading partners
- ❑ Business Process Model
 - ➔ Interactions between parties
 - ➔ Sequencing of interactions
 - ➔ Documents exchanged in each interaction
- ❑ Information Model
 - ➔ Document definition
 - ➔ Context definition
 - ➔ Context rules

Business Process



Resources

- ❑ ebXML Participation & Mailing Lists
 - ➔ Open to everyone
 - ➔ <http://www.ebxml.org/participate.htm>
- ❑ ebXML Specifications
 - ➔ http://www.ebxml.org/specdrafts/approved_specs.htm

