COMP28112
The Integration Game

Lecture 17

(material put together by Dean Kuo, some time ago!)

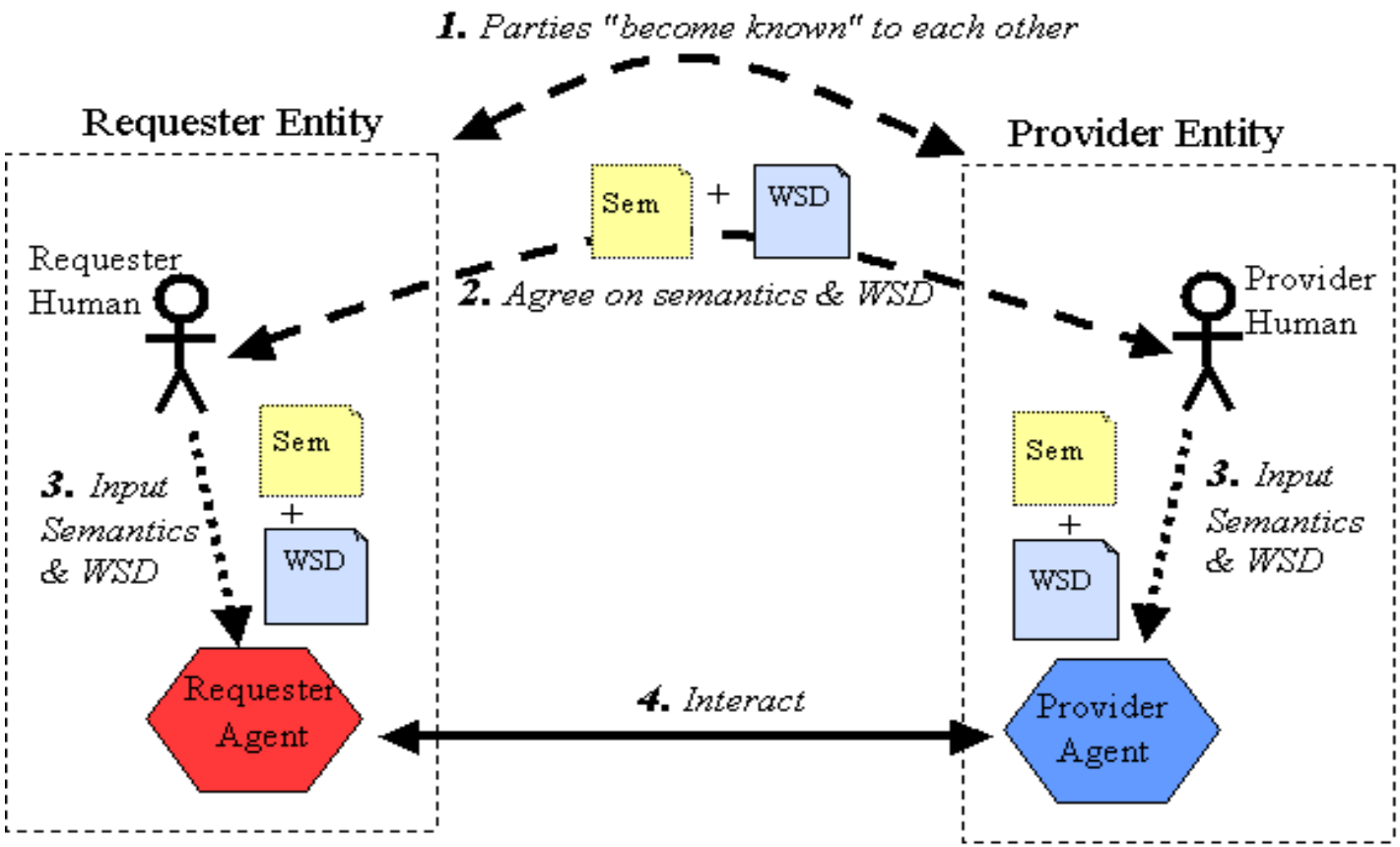
"One view holds that computer science is simply the art of realising successive layers of abstraction!"

Building large-scale distributed systems requires us to strike a balance between:

- Performance
- Reliability
- User Requirements
- Other constraints (eg, money…)

As well as software aspects:
http://static.googleusercontent.com/media/research.google.com/en//people/jeff/stanford-295-talk.pdf

Service is the idea that links requestor and provider



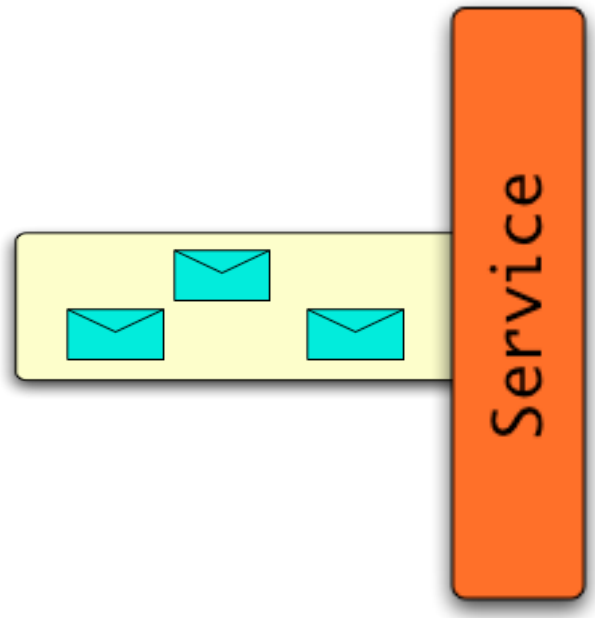
Semantic descriptions are needed to ensure that the requestor gets the “right” service.

Service Oriented Architecture (SOA)

- Principles for building systems meeting the requirements:
  - Reliable, fault-tolerant, responsive, scalable and secure
  - Interoperate across computing platforms and administrative domains

Service

- A service is a collection of code and data that stands alone
- The ONLY way in and out of a service is via messages
- Services are durable and survive crashes



What follows are known as the four tenets of SOA

Service boundaries are explicit
Services are autonomous
Services share Schema and Contracts
Service Compatibility is based on policy

Service Boundaries are Explicit

- No ambiguity as to where a piece of code and data resides
- It is explicit if a piece of code is inside or outside a service
  - E.g. Internet banking - the code and data live inside the banking service

Services are Autonomous

- Services are developed and managed independently
- A service can be re-written without impacting other services
  - With some constraints

Services Share Schema and Contracts

- Schema defines the messages a service can send and receive
- Contract defines permissible message sequences
- Services do not share implementations
  - Independent of computing platform or languages

Service Compatibility is Based on Policy

- Defines the rules for using a service
  - E.g. security policy specifies who can use the service
  - Introduces the idea of roles into distributed computing
  - Question of whether services are organised into an overall Virtual Organisation or are accessed over an open network or market.

More Complex Services

- Purchase of goods or resource
  - Get a quote, make a reservation, make changes, make payment, cancel ticket
  - Book 100 hours and 500 Gbytes of storage on a computing cloud

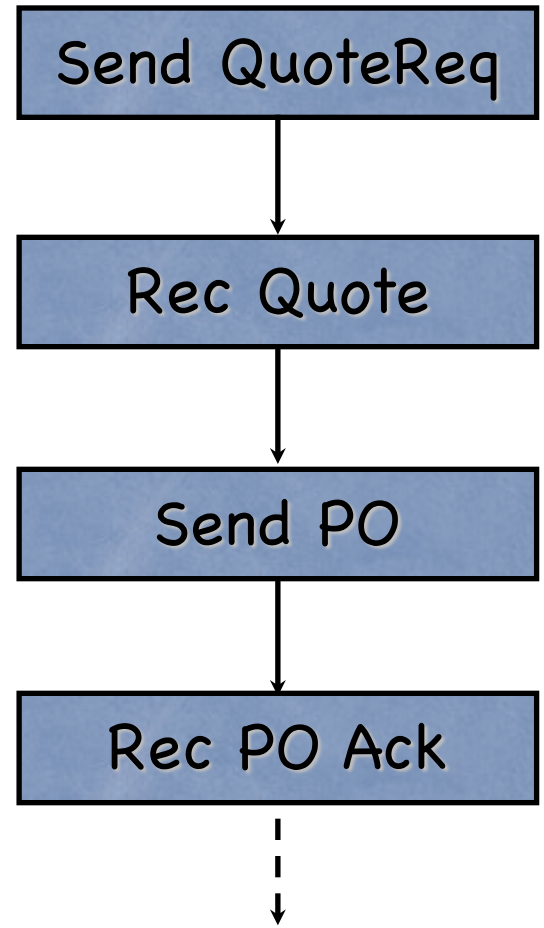
  All these messages are related to a particular order
  - Need an unique identifier to correlate the messages

Service Schema

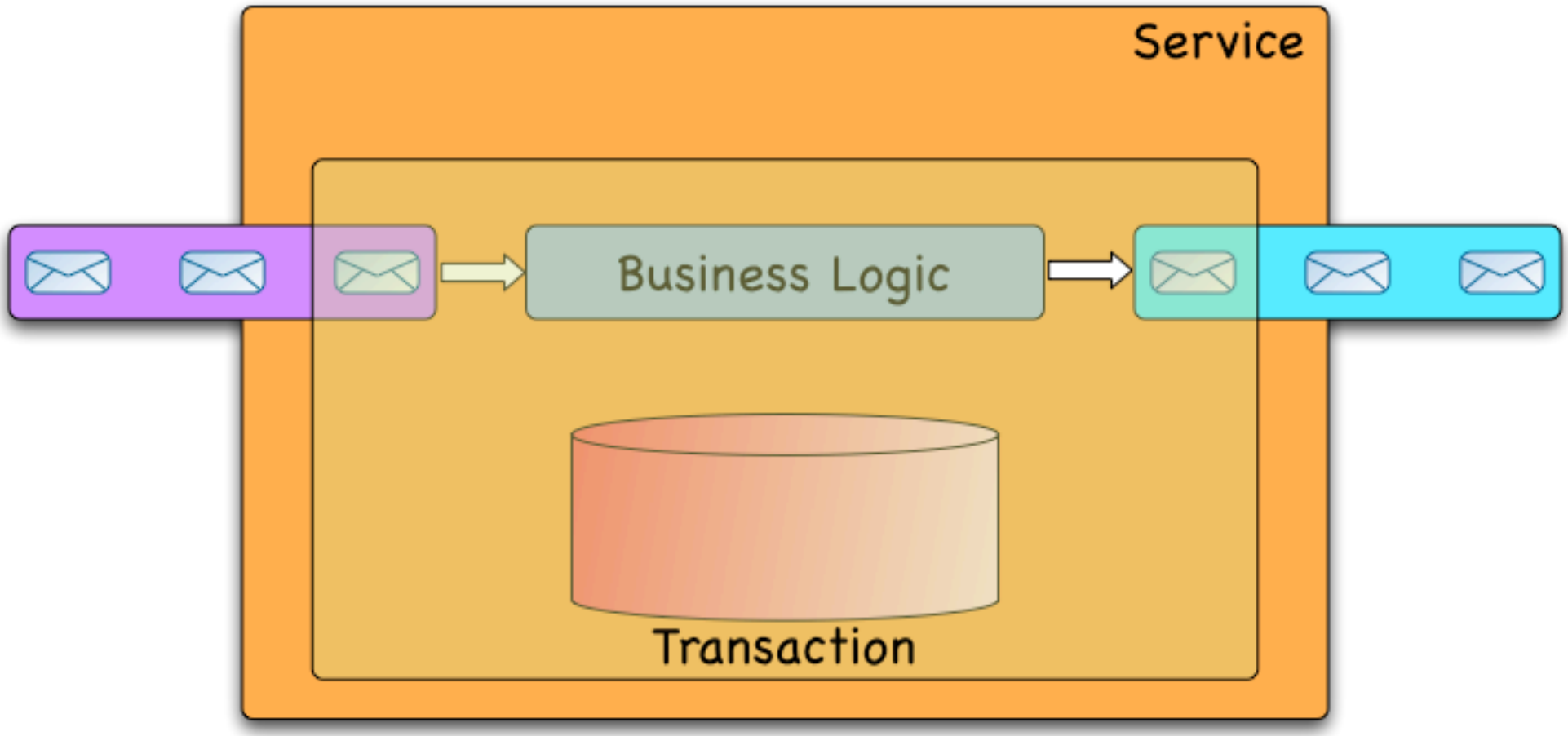
- Defines the message schema for
  - Quote request
  - Quote
  - Purchase order
  - Invoice
  - ...

Service Contract

- Defines the causal relationship between the messages - e.g.
  - Quote request before quote
  - Specifies if quote is optional or mandatory
- Co-relation identifier to link the messages in a conversation



Putting It All Together



Transactions Inside a Service

Crash
- Begin_Transaction

Crash
- Get message from the in queue

Crash
- Process message

Crash
- Put message on the out queue

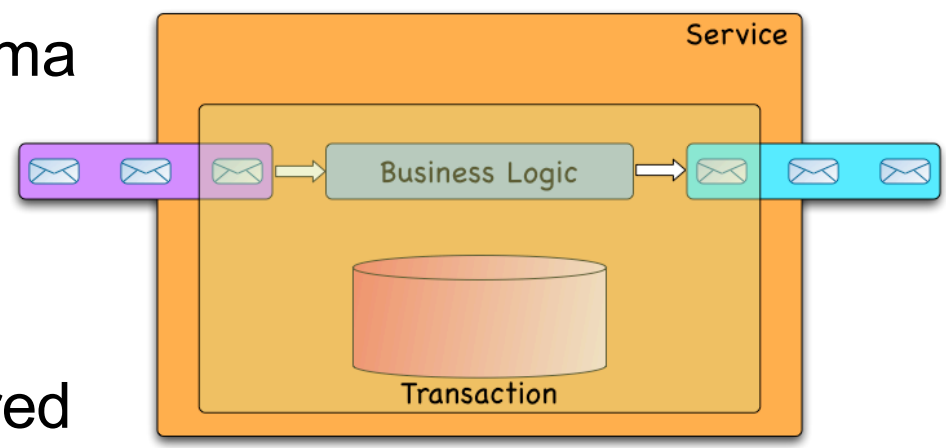
Crash
- Commit

Transactions Across Queuing system and Database

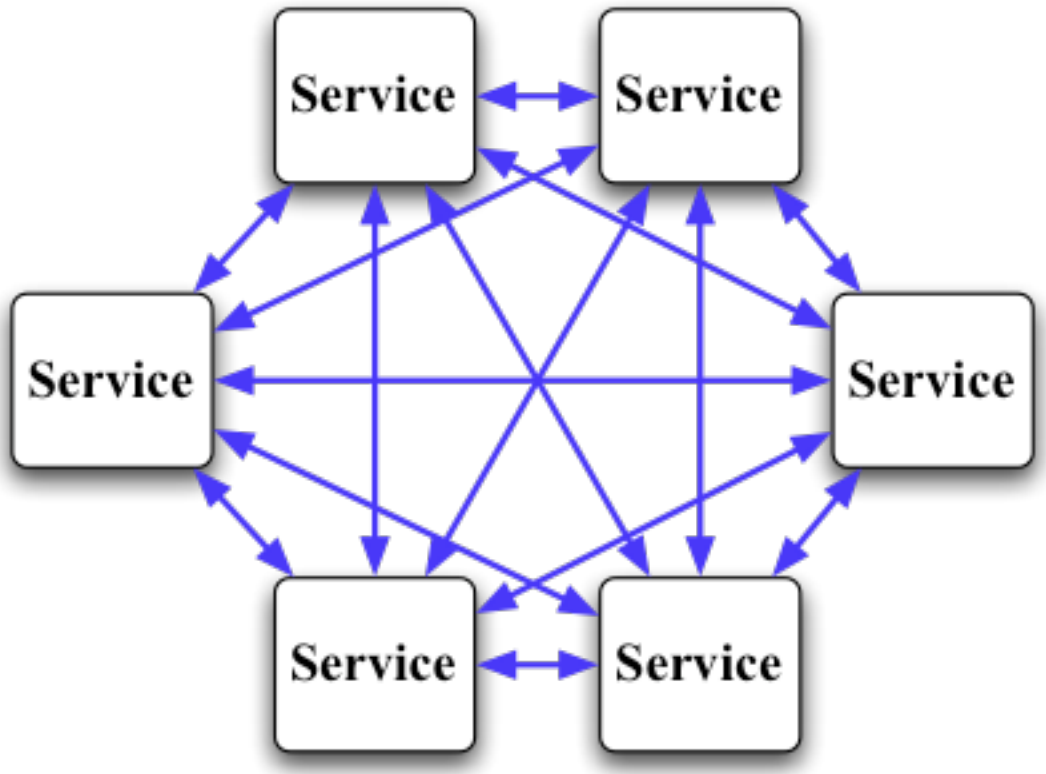
- Two-phase commit between the queuing system and the database

Services Share Schema

- Incoming message
  - Service transforms the message from the “shared schema” to its internal schema
- Outgoing message
  - Service transforms from its internal schema to the “shared schema”



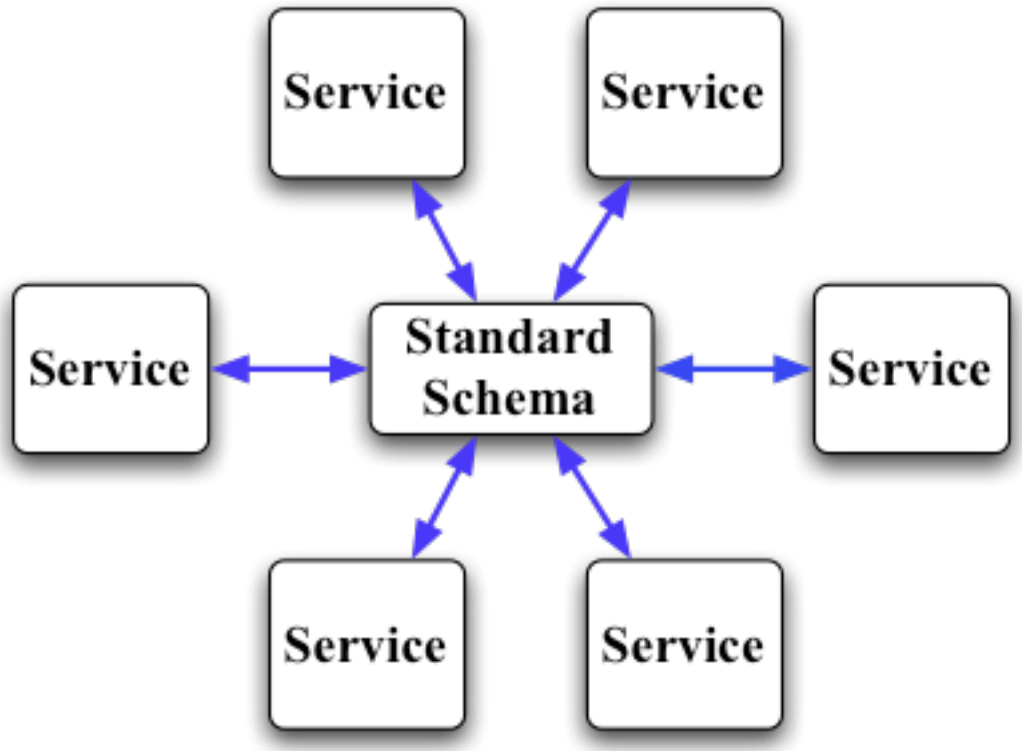
Shared Schema Between Every Pair of Services



N x (N-1) number of transformations

12 services => 132 transformations

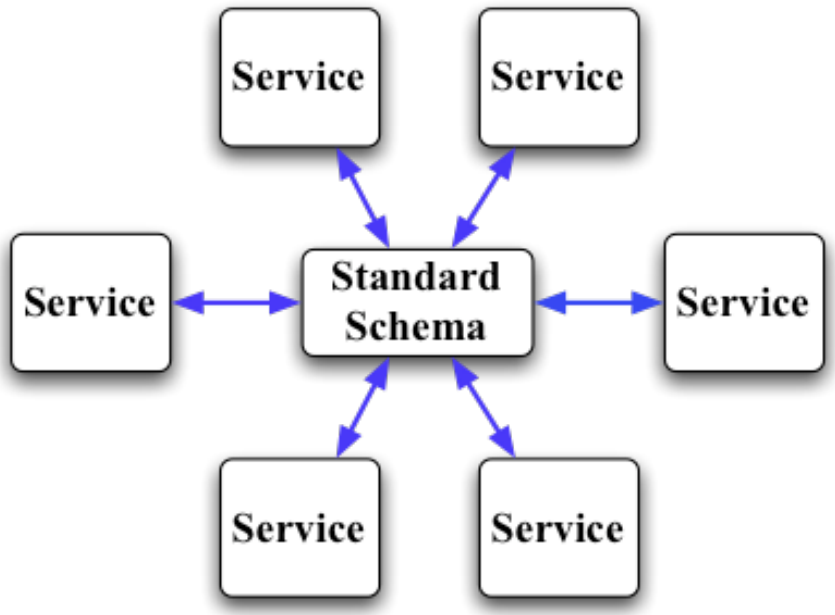
The Need for Standards

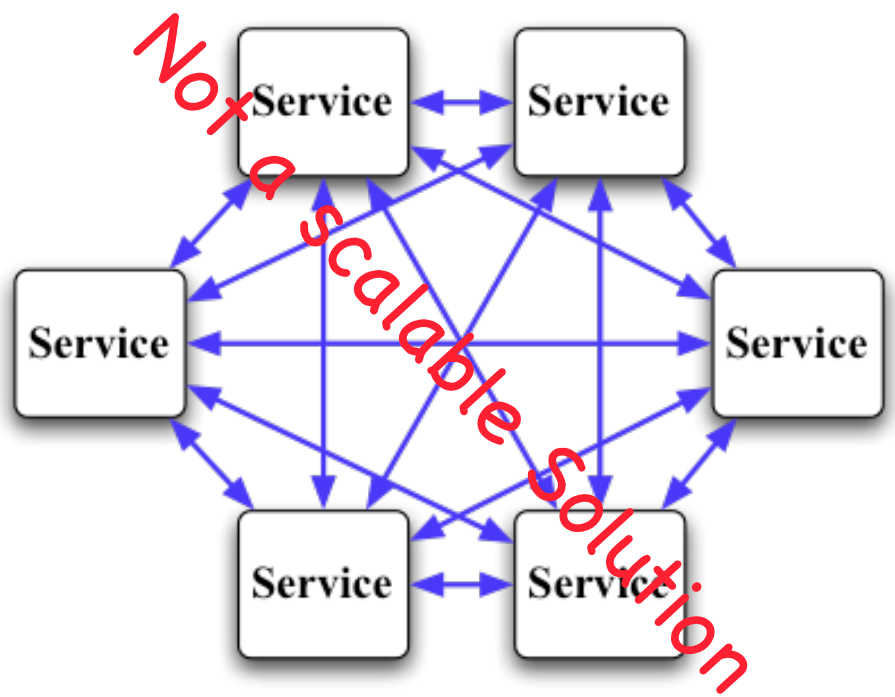


2N number of transformations

12 services => 24 transformations

Which one is simpler?





Schema Mapping Problems

- There are difficulties in mapping a service's internal schema to another schema
- Unfortunately, it can be very difficult and at times impossible to achieve

Name Conflicts

- Same concept but different names
- Same name but different concepts
  - One price includes VAT and the other does not

```
<slot_id> 3 </slot_id>
<price> 45.10 </price>
```

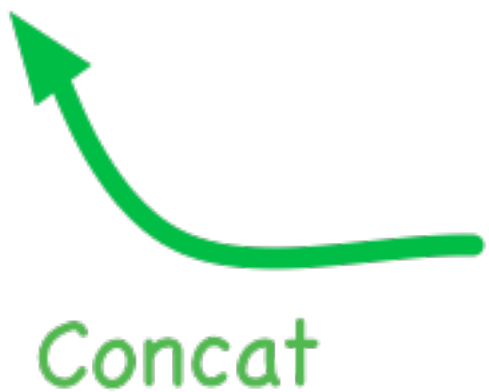
```
<slot> 3 </slot>
<price> 53 </price>
```

Structural Mismatches

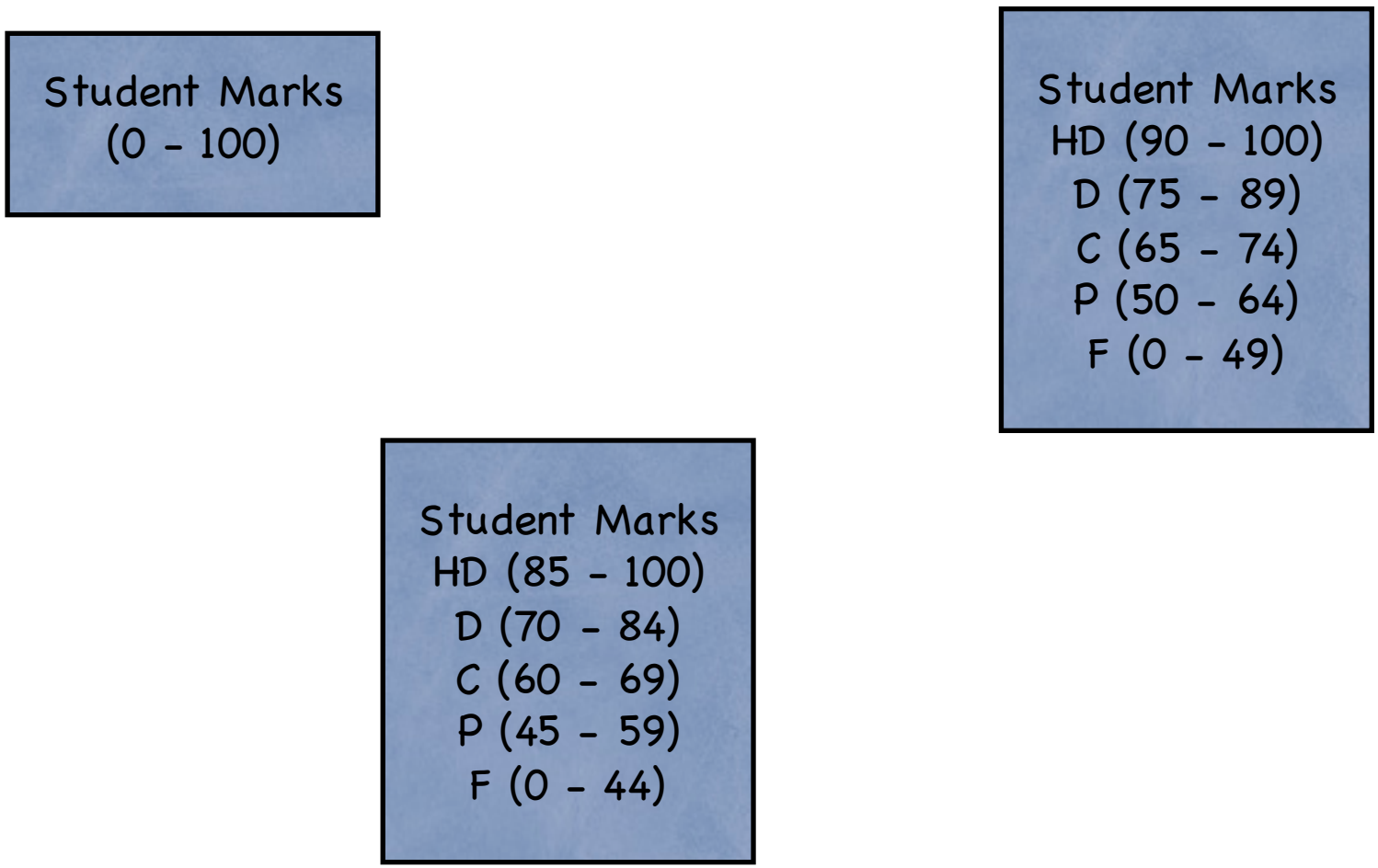
- Structure of the information is different

```
<address>
        303 Deansgate Manchester M3 4LQ United
Kingdom
</address>
```


```
<address>
        <street> 303 Deansgate
</street>
        <city> Manchester </city>
        <postcode>  M3 4LQ
</postcode>
        <country> United Kingdom
</country>
</address>
```



Different Representation



Ontologies and semantics

- If we can say what the information “means” we can have some hope of reconciling different syntax for the “same” thing.
- This is known as a semantic description of the data.
- Ontologies are tools used to capture the “meaning” of data or services.
- An ontology is the representation of a set of concepts within a domain and it allows reasoning about those concepts, e.g. to assert equivalence.

Agreement on a standard is HARD!

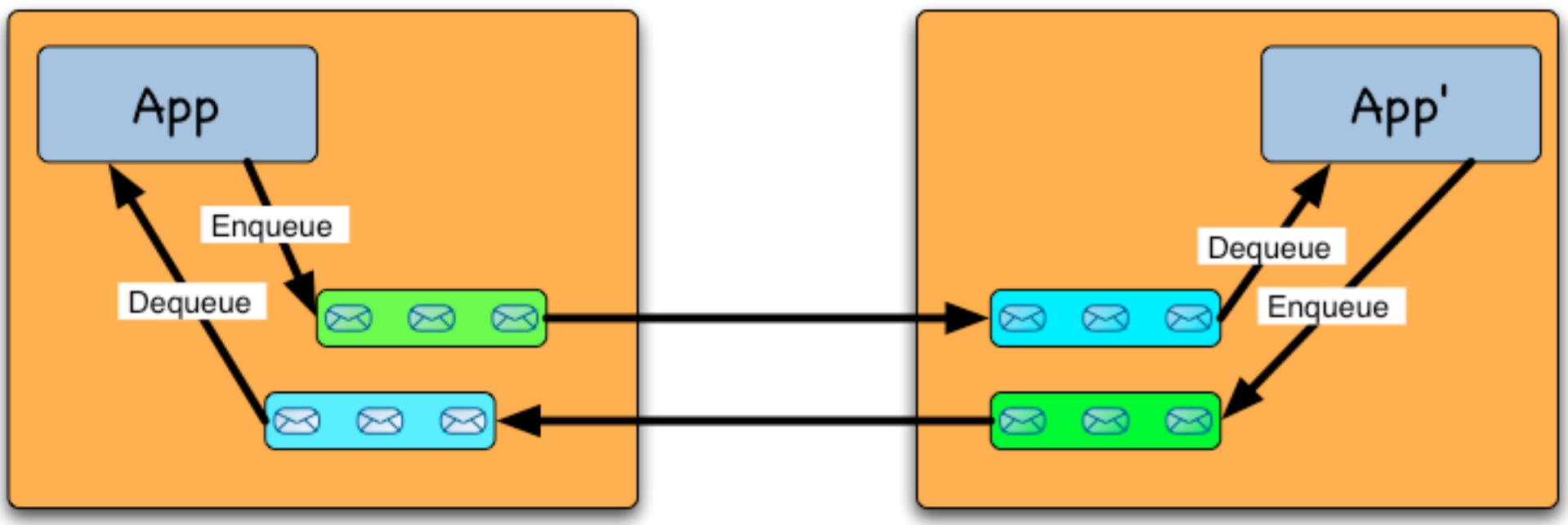
- But often not for technological reasons
- Each service, for economical reasons, will want to reduce the costs of writing code to transform their data to and from the standard
- Sometimes it is extremely hard or impossible to transform the data
- Standards will evolve slowly:
  - Driven by large organisations such as TESCO, ... or
  - Mandated by government or
  - By a community
- Slow, painful and expensive process

Message-Oriented Middleware (MOM)

- Plumbing for shipping messages between the services
  - SOA is all about using messages to connect services
- Asynchronous and non-blocking message passing
  - Sender application sends and does not wait for a response - similar to sending emails
  - Recipient may or may not be actively processing incoming messages when the message is sent

Point-to-Point messaging

- Sender specifies a recipient
- Message is delivered to ONE recipient



MOM supports Three Types of Delivery Modes

- At most once
- At least once
- Exactly once

At Most Once
(Best Effort)

- MOM sends and forgets
- Most appropriate when it is not essential that the message is delivered
  - E.g. live cricket and football scores, stock prices, ...
- MOM does not need to write to stable storage
  - Message may be lost due to failures

At Least Once

- Sender sends a message
  - Recipient will “eventually” receive at least one copy of the message
    - Survives network and service failures
- Used when a message must be received and application receiving the message can cope with duplicates

At Least Once

- If the recipient does not acknowledge within a time-out, sender resends the message
  - Repeat until sender receives an acknowledgment
    - Safe to resend
    - Safe for recipient to deliver a duplicate to the application
  - Require the sender and receiver to write to stable storage to survive crashes

Application Processing a Message

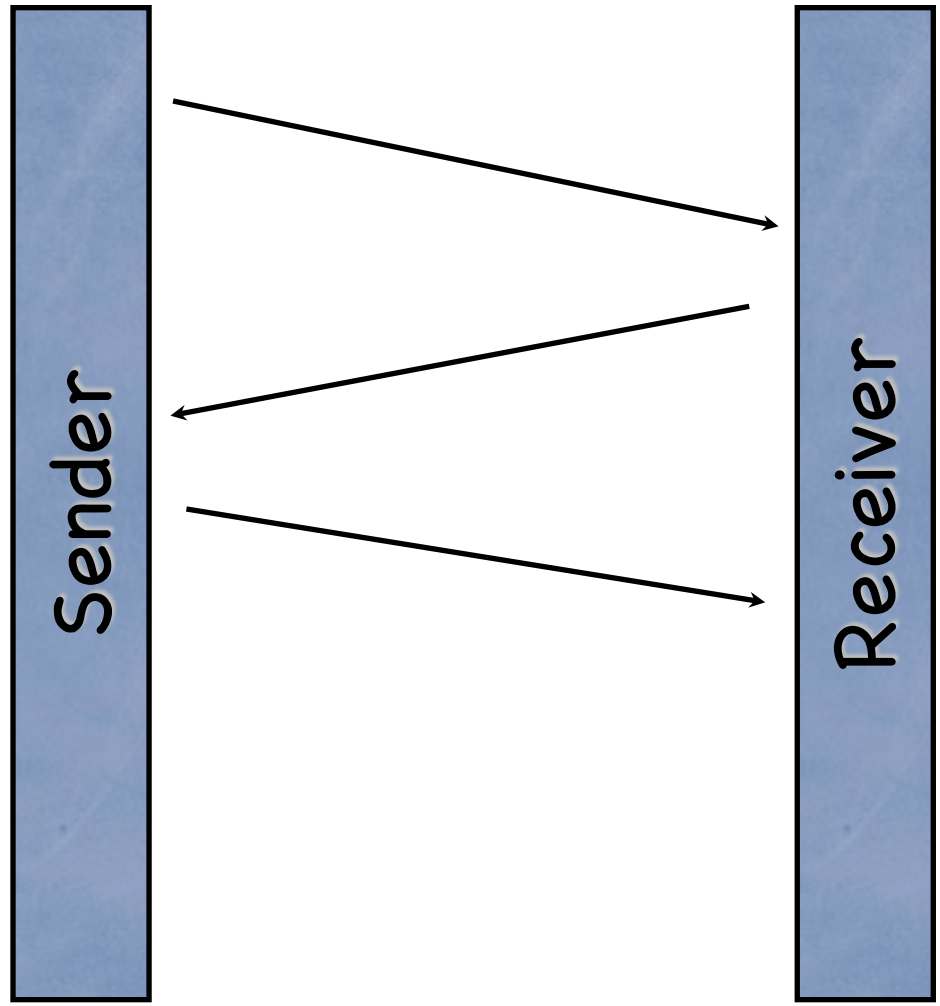
- At least once delivery - duplicates can be received
  - Determine if this message is a duplicate
    - If not then process the message; otherwise ignore the message
  - Don’t need to worry about duplicates if it is a “read” request
    - e.g. get bank balance
- Duplicates will not cause inconsistencies

Exactly Once

- Recipient is guaranteed that it will “eventually” receive exactly one copy of the message
- Application processing the message “knows” that it has never processed this message before and a duplicate will never be delivered
  - Makes it easier to write the application

Exactly once

- Uses a variant of 2PC to ensure exactly once delivery
- Requires the sender and receiver to write to stable storage
- Message must be received and recipient can't cope with duplicates



In-Order Delivery

- Messages are delivered to the receiving application in the same order as they were sent by the sending application

Throughput

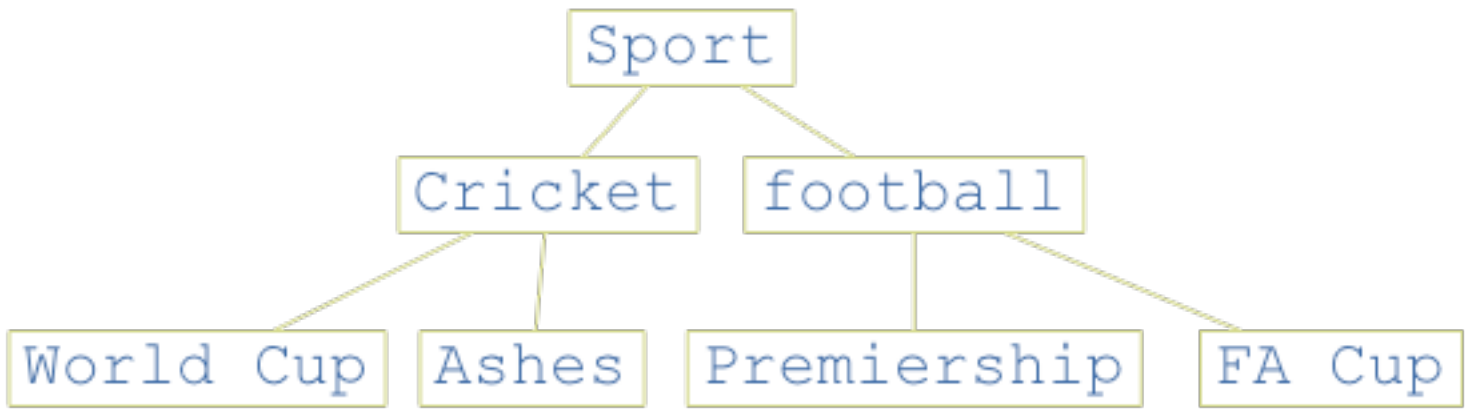
- From greatest to least
  - At most once
  - At least once
  - Exactly once

Publish Subscribe

- Sender publishes (sends) a message on a topic rather than a destination
  - Subscriber subscribe to topics
  - Similar to the idea of mailing lists

Topics

- Topics are organised in a hierarchy
  - Subscriber will receive all messages published on the subscribed topic and its sub-topics



Delivery Modes

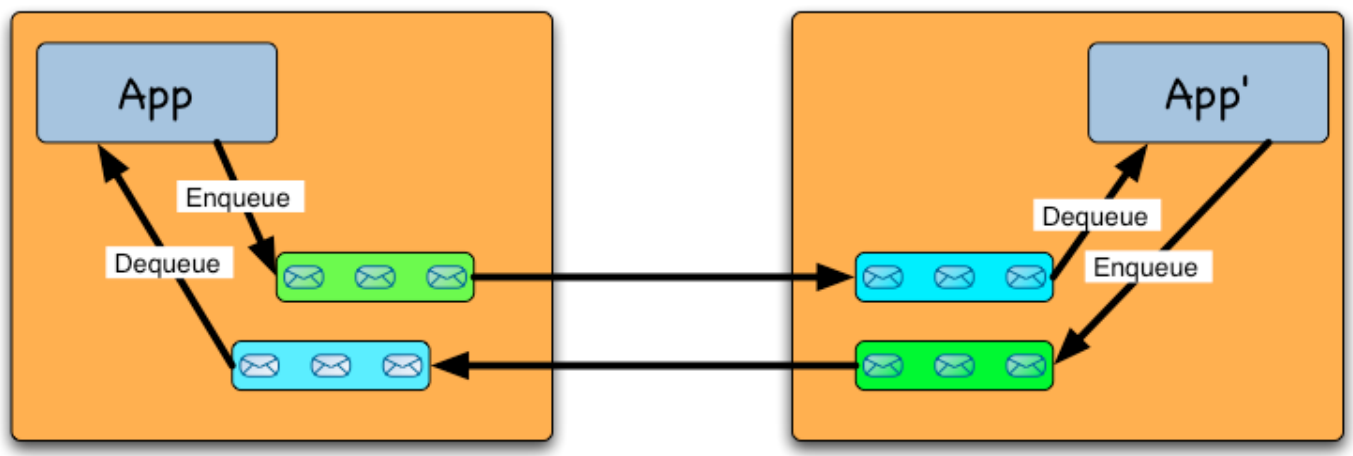
- Identical to point to point
  - At most once
  - At least once
  - Exactly once
  - In-order delivery

Messaging Technologies

- IBM WebSphere MQ (formerly MQSeries)
- MSMQ
- SQLServer
  - Native support for queues
- ActiveMQ (open source)
- TIBCO Rendevouz
- ...

The Need for Interoperability

- Services must be able to exchange messages
  - Remember, services do not share implementation
  - Services should be free to choose the MOM of their choice



Unfortunately

- MOMs from different vendors don't interoperate
- All applications must use THE SAME MOM implementation
  - Violates service autonomy

Emerging Standards

- Web Services Reliable Messaging (WS-Reliable Messaging)
- Advanced Message Queuing Protocol (AMQP) - a message queuing protocol by JP Morgan Chase & Co, Red Hat, ...

SOA/Messaging

- It WORKS and has worked for many centuries
  - Initially, paper based systems
  - Architecture for EAI (enterprise application integration) and B2B (business to business) integration
    - Towards the paperless world
  - Follows the KISS principle
    - Keep It Simple Stupid

Summary I

- Service-oriented architectures
  - Simple set of design principles for building applications that is technology independent
  - The use of messages to connect services
  - A service is a collection of code and data and the only way in and out of a service is through “messages”
- Four tenets of SOA
  - Boundaries are explicit
  - Services are autonomous
  - Services share schema and contracts and not implementation
  - Service compatibility is based on policy

Summary II

- Message-Oriented Middleware
  - Provides the infrastructure for SOA
  - Semantic descriptions, can promote service discovery, matching and interoperability.
- Messaging
  - At most once, at least once and exactly once delivery
  - Point-to-point and publish/subscribe
- Transactions across queuing system and database
- The need for standard schemas and MOM interoperability

Engineering Internet Scale Distributed Systems

Requirements

- High availability
- Responsive
- Scalable
- Reliable and fault-tolerant
- Maintains data consistency in the presence of failures and concurrency

How should these systems be designed?

Architectural Styles

- Define guiding principles for engineering systems
- Technology independent
- SOA is an architectural style based on messaging, idempotent operations, immutable messages, ...
- Are there others for building Internet-scale distributed systems?
- Can we learn from systems that work?

The Web

- Has many of the key characteristics that we want
  - High availability, scalable, responsive, ...
- What can we learn from the success of the Web in building distributed systems?
- What is the architectural style behind the Web?
  - Focus of the rest of this lecture is the study of the architectural style behind the Web

The architectural style behind the Web is REST - REpresentational State Transfer

REST

- Term coined by Roy Fielding in his PhD thesis to describe an architectural style for networked (distributed) systems
  - Course web site provide links to his thesis and other REST material
  - REST is described in Chapter 5 of the thesis

Why is it called REST?

- Key abstraction in REST is “resources”
  - A resource can be anything that can be named
    - COMP28112 course is a resource, everyone in the lecture is a resource ...
- A resource can have a number of representations
  - The HTML page at https://studentnet.cs.manchester.ac.uk/ugt/COMP28112/ is a HTML representation of this course
  - A resource can have links to other resources

REST

- Provides the image of a network of linked representation of resources
  - Each “representation” provides information about the “state” of the resource
- Client navigate through the resources via links
  - Each link “transfers” the client to the representation (state) of another resource
- So we have “REpresentational State Transfer” - REST

Implementation of REST

- The idea is to make use of the existing technology, URL,XML,HTTP,SMTP etc. to build Web services.
- For example HTTP supports these “verbs”:
  - HEAD (get the Meta data about the resource)
  - POST (create a resource)
  - GET (get the state of the resource)
  - PUT (modify the state of the resource)
  - DELETE (destroy the resource)

REST

- REST is independent of any middleware
  - J2EE, .NET, CORBA, Web Services....
- HTTP follows many of the principles prescribed by REST:
  - URLs identify resources
    - E.g. http://www.manchester.ac.uk identifies the resource "University of Manchester" while https://studentnet.cs.manchester.ac.uk/ugt/COMP28112/ identifies this course
    - Opaque and exposes no details of the implementation
  - Only representations are exposed

Representations of Resources

- A resource can have multiple representations
  - e.g. HTML, XML, JPEG, ...
- Client specifies in the http “accept” header the representation it wants to receive
  - HTML, XML, JPEG, ...

Uniform Interface

- Resources have the same interface
  - Simplifies the overall architecture
- in HTTP, every resource has the same methods
  - GET - retrieves a representation
  - POST - creates a new resource
  - PUT - updates an existing resource
  - DELETE - deletes a resource

Stateless Interactions

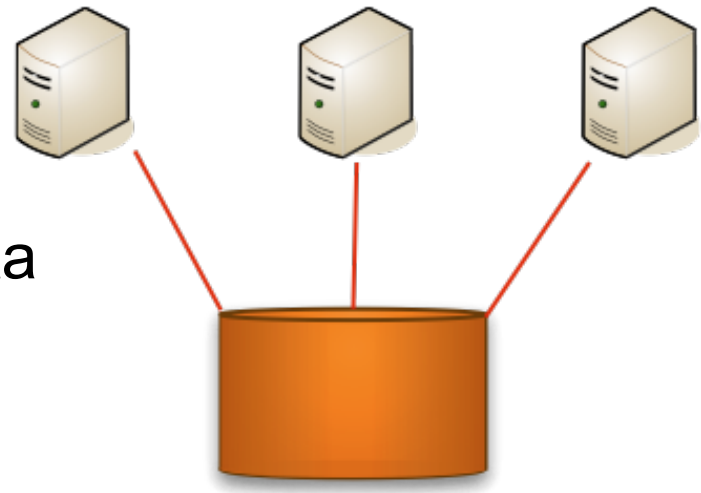
- Each request contains all the information necessary for the server to understand the request independent of any request that has preceded it
- No context is stored on the server
  - Should avoid the use of cookies

Benefits of statelessness:

- Reliability
  - Easy to manage server failures
- Scalability
  - Easy to add a new server to process messages

Resources

- State of resources are stored in databases shared by the servers
- Databases are good at storing data
  - Reliable, fast, fault-tolerant, manage concurrency, ...



Summary

- REST is an architectural style
  - Defines a set of constraints for building internet scale distributed applications
- Key abstraction is “resource”
- Key constraints
  - Anything that can be named can be a resource, URLs identify resources and uniform interface
- Simple and elegant