EN
English

Universidad de Oviedo

School of Computer Science

Integration

SOFTWARE
ARCHITECTURE

Course 2018/2019

Jose E. Labra Gayo

Integration

Application Integration = Biggest challenge

Integration

- Integration styles
  - File transfer
  - Shared database
  - Remote procedure call
  - Messaging
    - Event log
- Topologies
  - Hub & Spoke, Bus
- Service Oriented Architectures
  - WS-*, REST
- Microservices
- Serverless

Integration styles

File transfer

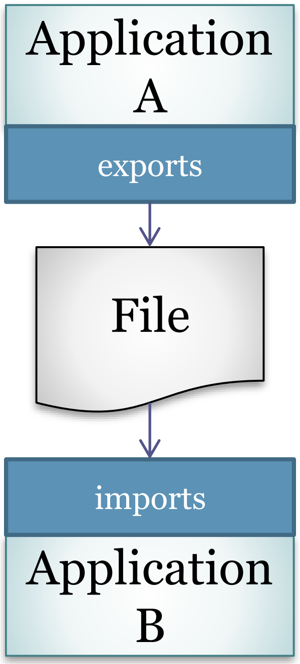
Shared database

Remote procedure call

Messaging

File transfer

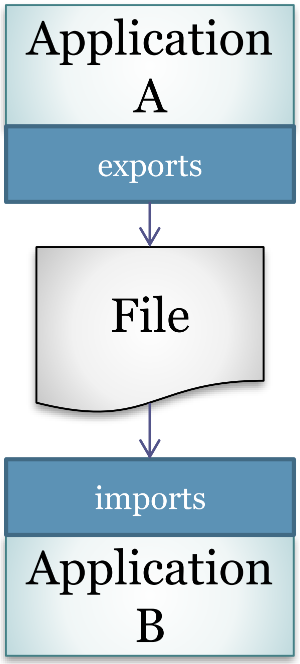
- An application generates a data file that is consumed by another
  - One of the most common solutions
- Advantages
  - Independence between A and B
  - Low coupling
  - Easier debugging
    - By checking intermediate files



File transfer

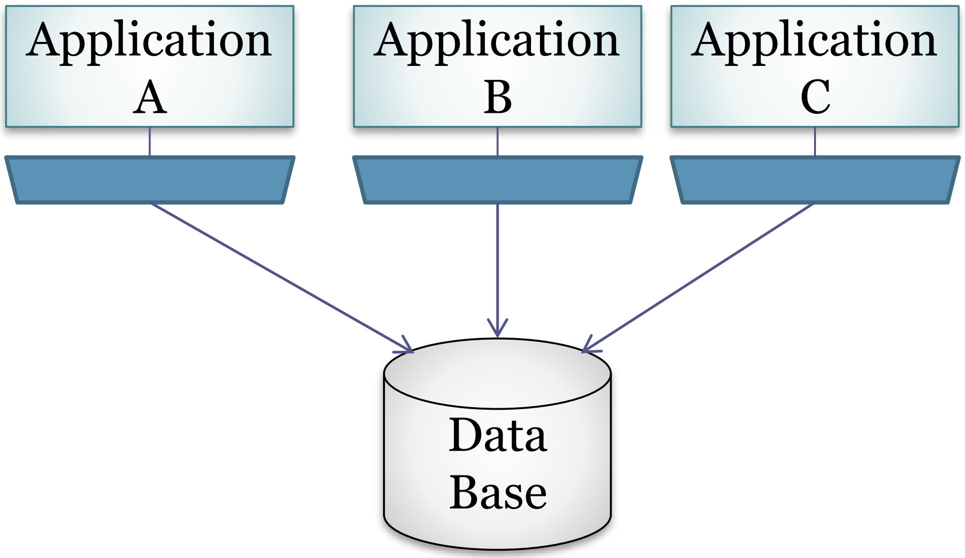
Challenges

- Both applications must agree a common file format
  - It can increase coupling
- Coordination
  - Once the file has been sent, the receiver could modify it $\Rightarrow$ 2 files!
- It may require manual adjustments



Shared database

- Applications store their data in a shared database
- Advantage
  - Data are always available
  - Everyone has access to the same information
  - Consistency
  - Familiar format
    - SQL for everything



Shared database

Challenges

- Database schema can evolve
  - It requires a common schema for all applications
    - That can cause problems/conflicts
  - External packages are needed (common database)
- Performance and scalability
  - Database as a bottleneck
- Synchronization
  - Distributed databases can be problematic
  - Scalability
  - NoSQL ?

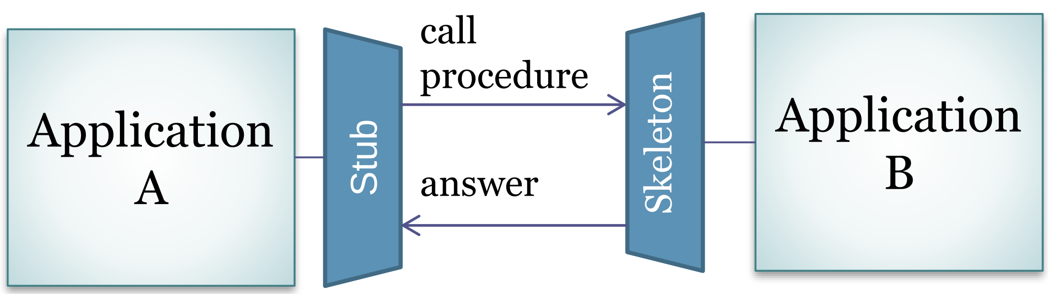
Shared database

Variants

- *Data warehousing*: Database used for data analysis and reports
- ETL: process based on 3 stages
  - Extraction: Get data from heterogeneous sources
  - Transform: Process data
  - Load: Store data in a shared database

Remote procedure call

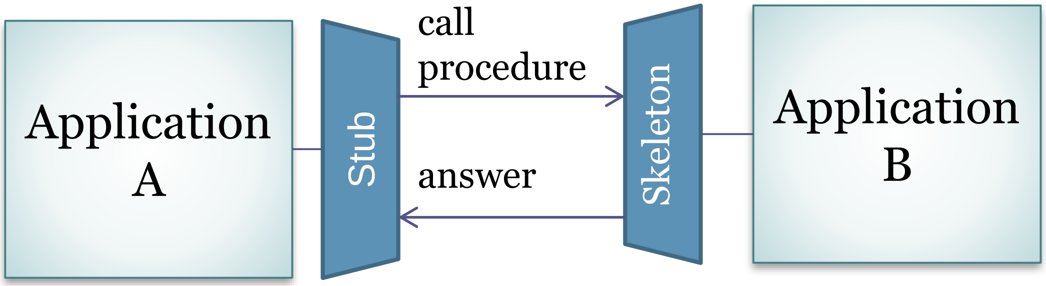
- An application calls a function from another application that could be in another machine
  - Invocation can pass parameters
  - Obtains an answer
- Lots of applications
  - RPC, RMI, CORBA, .Net Remoting, ...
  - Web services, ...



Remote procedure call

Advantages

- Encapsulation of implementation
- Multiple interfaces for the same information
  - Different representations can be offered
- Model familiar for developers
  - It is similar to invoke a method



Remote procedure call

Challenges

False sense of simplicity

Remote procedure $\neq$ procedure

8 fallacies of distributed computing

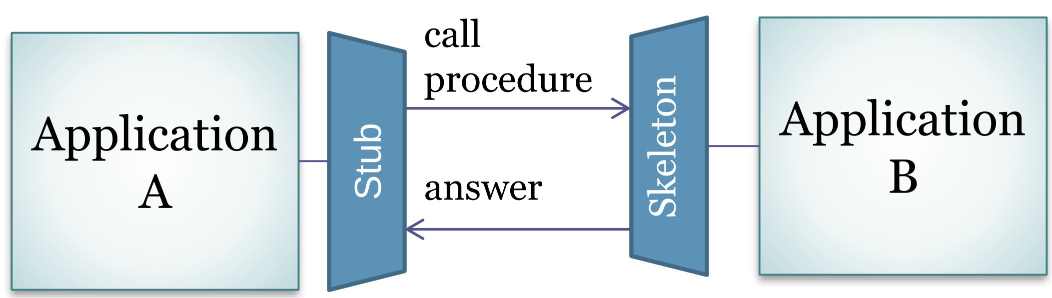
Synchronous procedure calls

Increase application coupling

The network is reliable
Latency is zero
Bandwidth is infinite
The network is secure
Topology doesn't change
There is one administrator
Transport cost is zero
The network is homogeneous

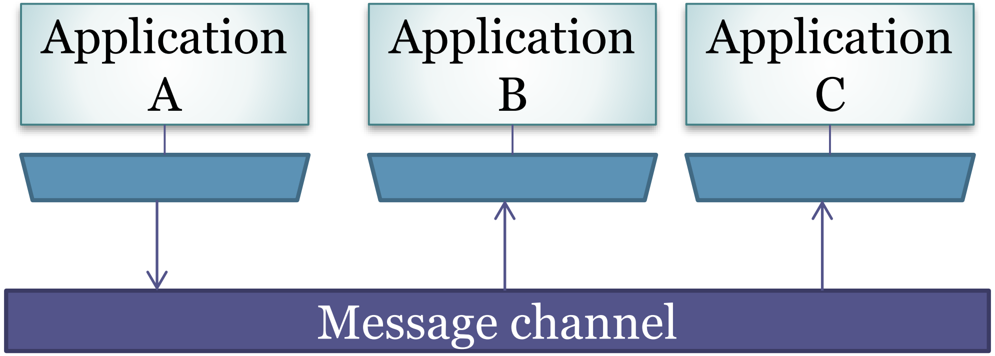
8 fallacies of distributed computing

http://en.wikipedia.org/wiki/Fallacies_of_distributed_computing



Messaging

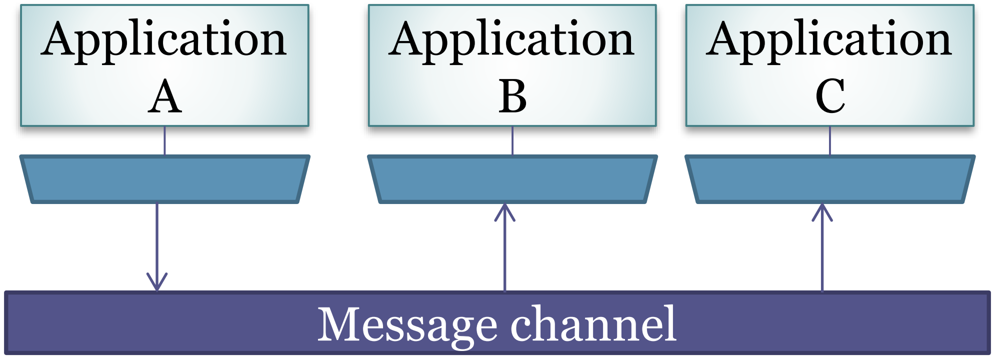
- Multiple independent applications communicate sending messages through a channel
- Asynchronous communication
  - Applications send messages a continue their execution



Messaging

Advantages

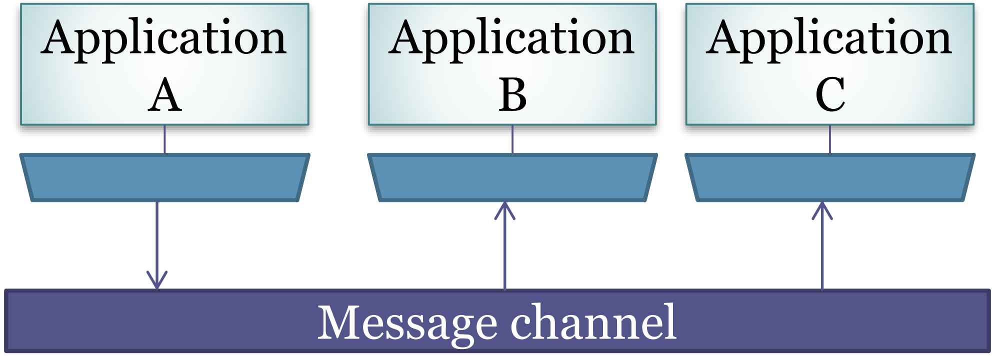
- Low coupling
  - Applications are independent between each other
- Asynchronous communication
  - Applications continue their execution
- Implementation encapsulation
  - The only thing exposed is the type of messages



Messaging

Challenges

- Implementation complexity
  - Asynchronous communication
- Data transfer
  - Adapt message formats
- Different topologies



Integration topologies

Hub & Spoke

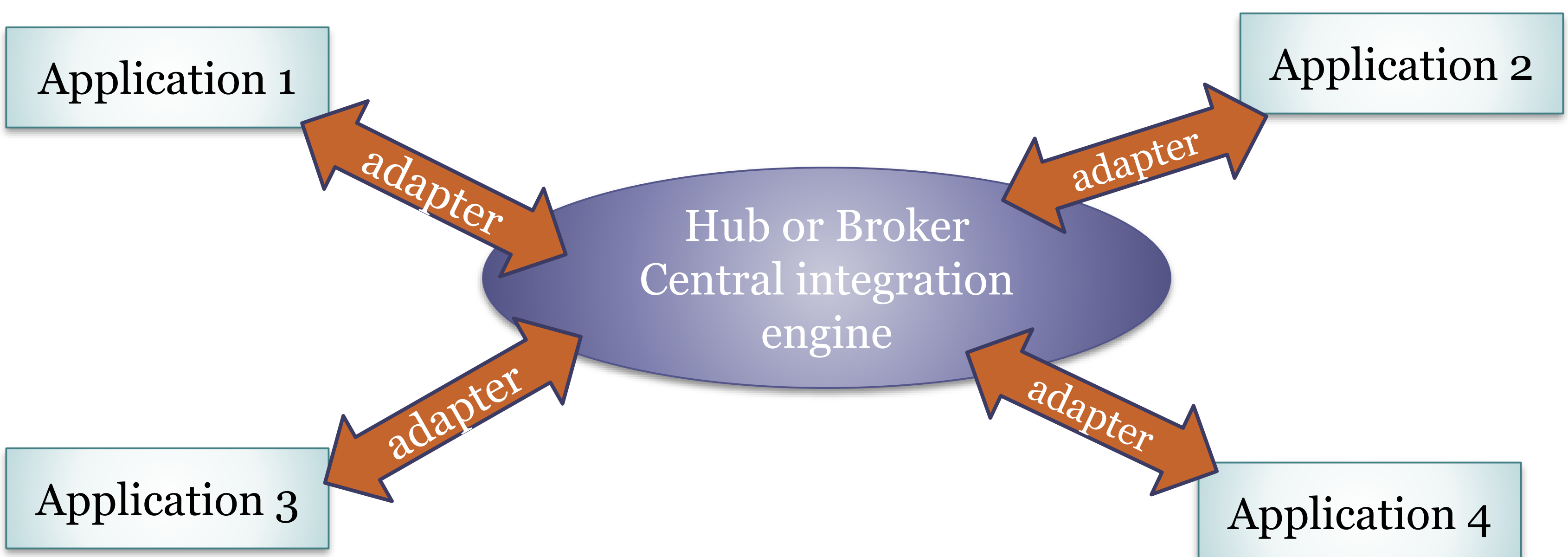
Bus

Hub & Spoke

Related with Broker pattern

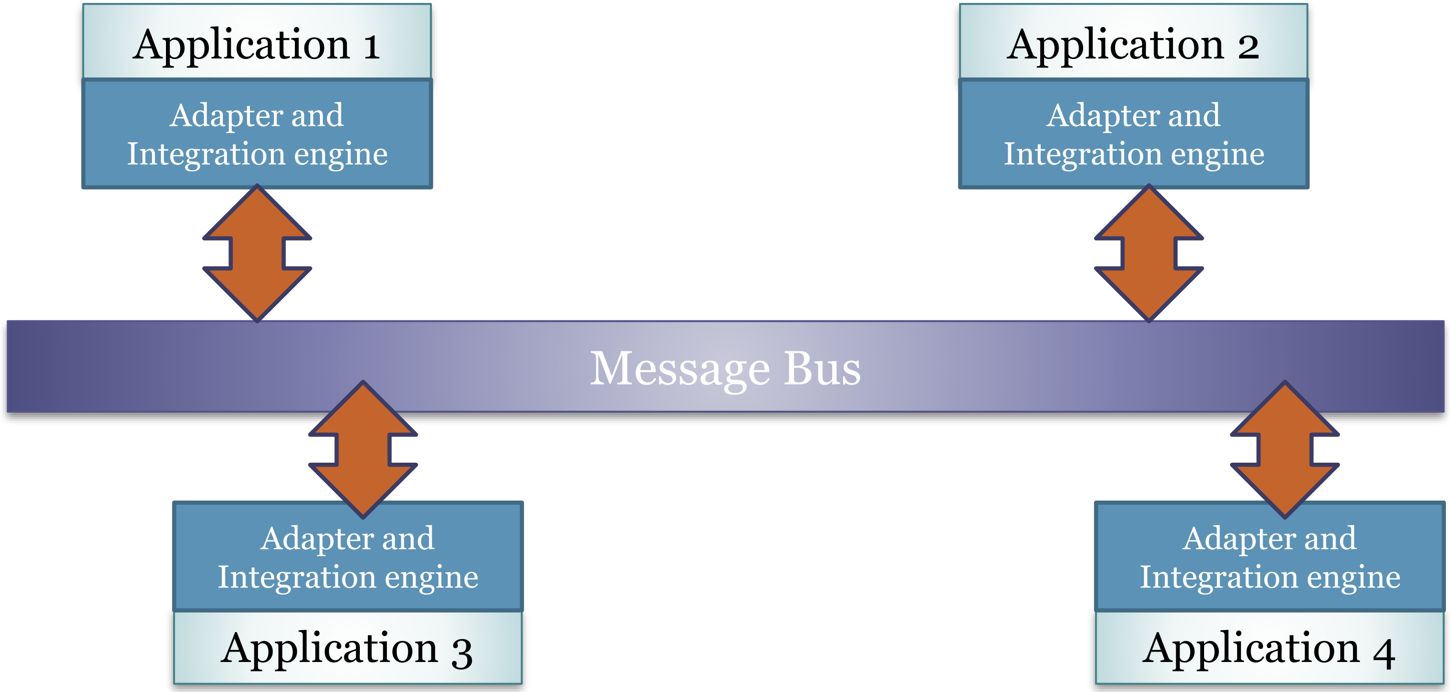
Hub = Centralized message Broker

- It is in charge of integration



Bus

- Each application contains its own integration machine
- Publish/Subscribe style



Bus

- ESB - Enterprise Service Bus
- Defines the messaging backbone
  - Some tasks
    - Protocol conversion
    - Data transformation
    - Routing
- Offers an API to develop services
  - MOM (Message Oriented Middleware)

Service Oriented Architectures

SOA

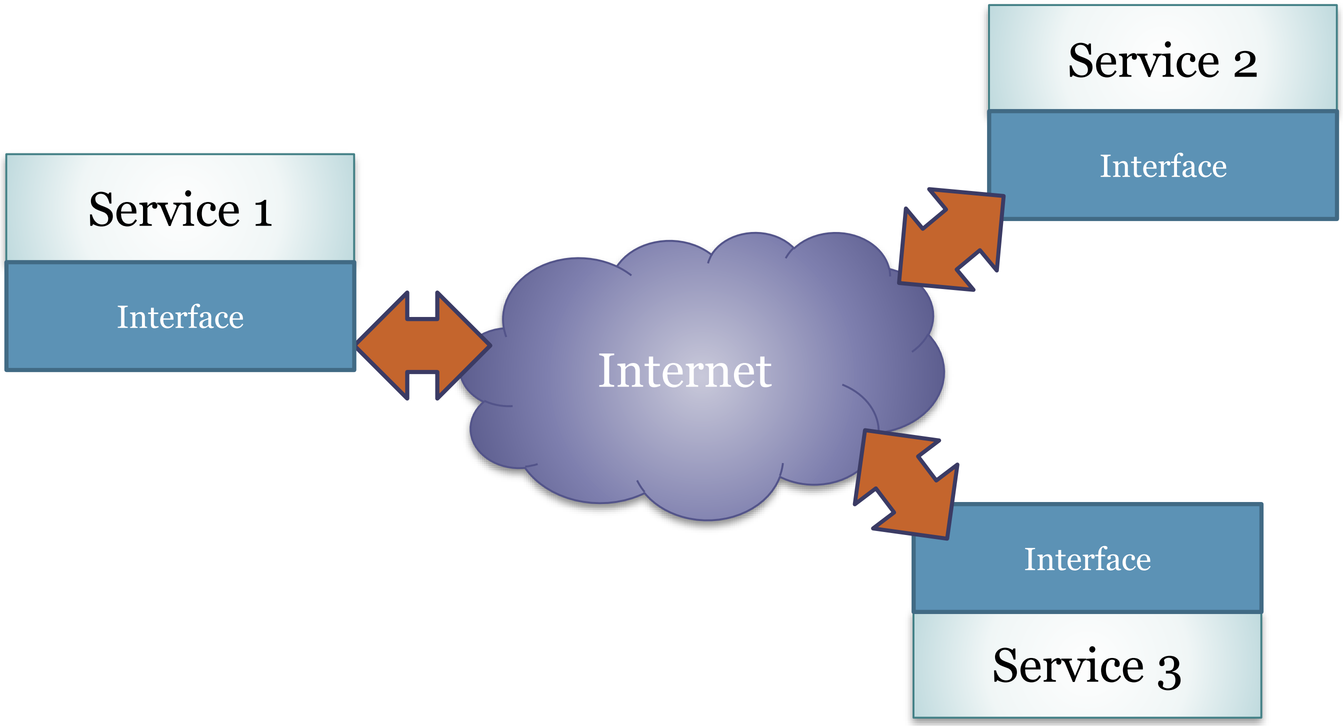
WS-*

REST

SOA

SOA = Service Oriented Architecture

Services are defined by an interface



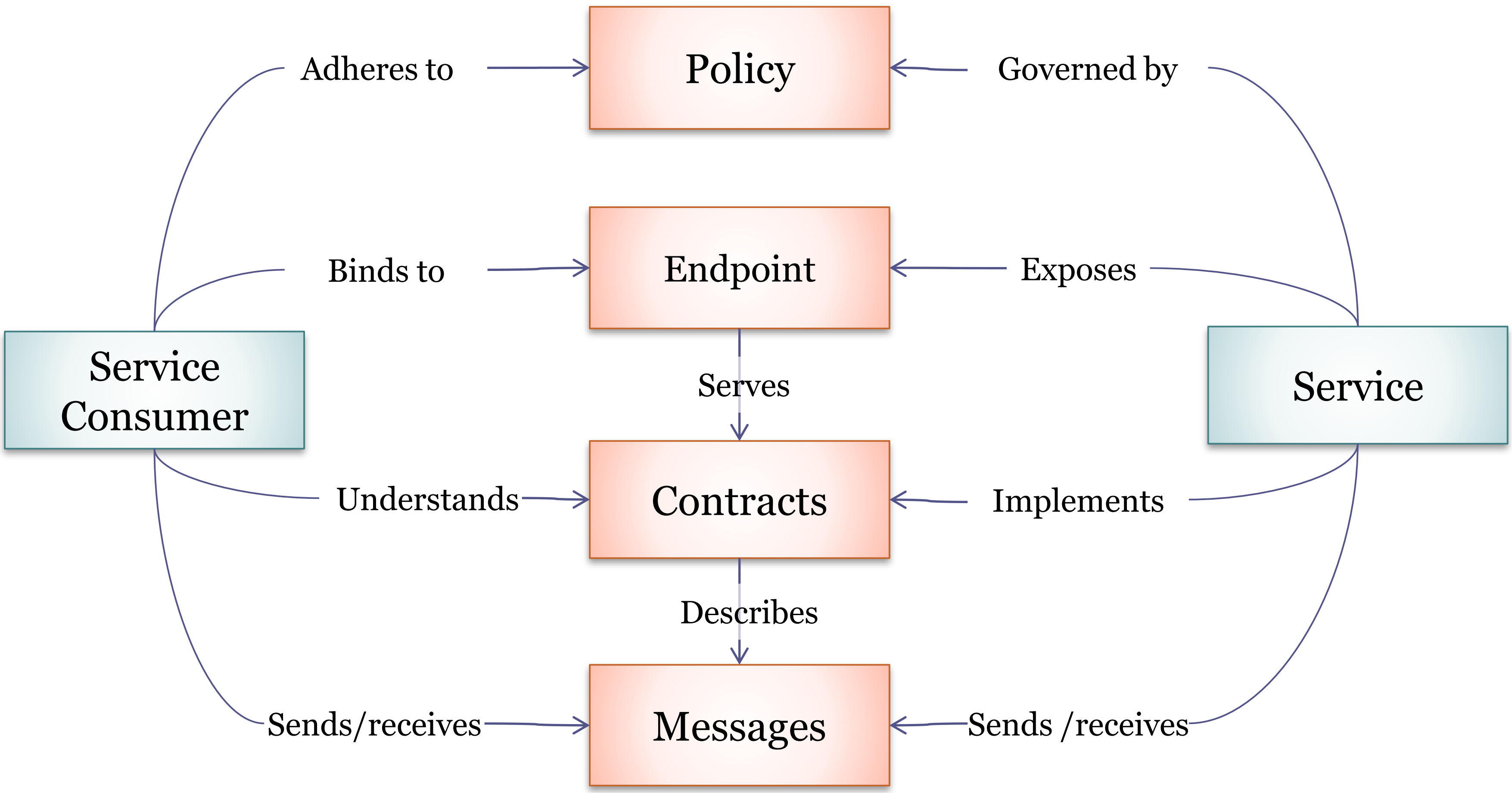
SOA

Elements

- Provider: Provides service
- Consumer: Does requests to the service
- Messages: Exchanged information
- Contract: Description of the functionality provided by the service
- Endpoint: Service location
- Policy: Service level agreements
  - Security, performance, etc.

SOA

Constraints



Fuente: SOA Patterns, A. Rottem Gal Oz

University of Oviedo
School of Computer Science

SOA

Advantages

- Independent of language and platform
- Interoperability
  - Use of standards
- Low coupling
- Decentralized
- Reusability
- Scalability
  - one-to-many vs one-to-one
- Partial solution for legacy systems
  - Adding a web services layer

Challenges

- Performance
  - E.g. real time systems
- Overkill in very homogeneous environments
- Security
  - Risk of public exhibition of API to external parties
  - DoS attacks
- Service composition and coordination

SOA

Variants:

- WS-*
- REST

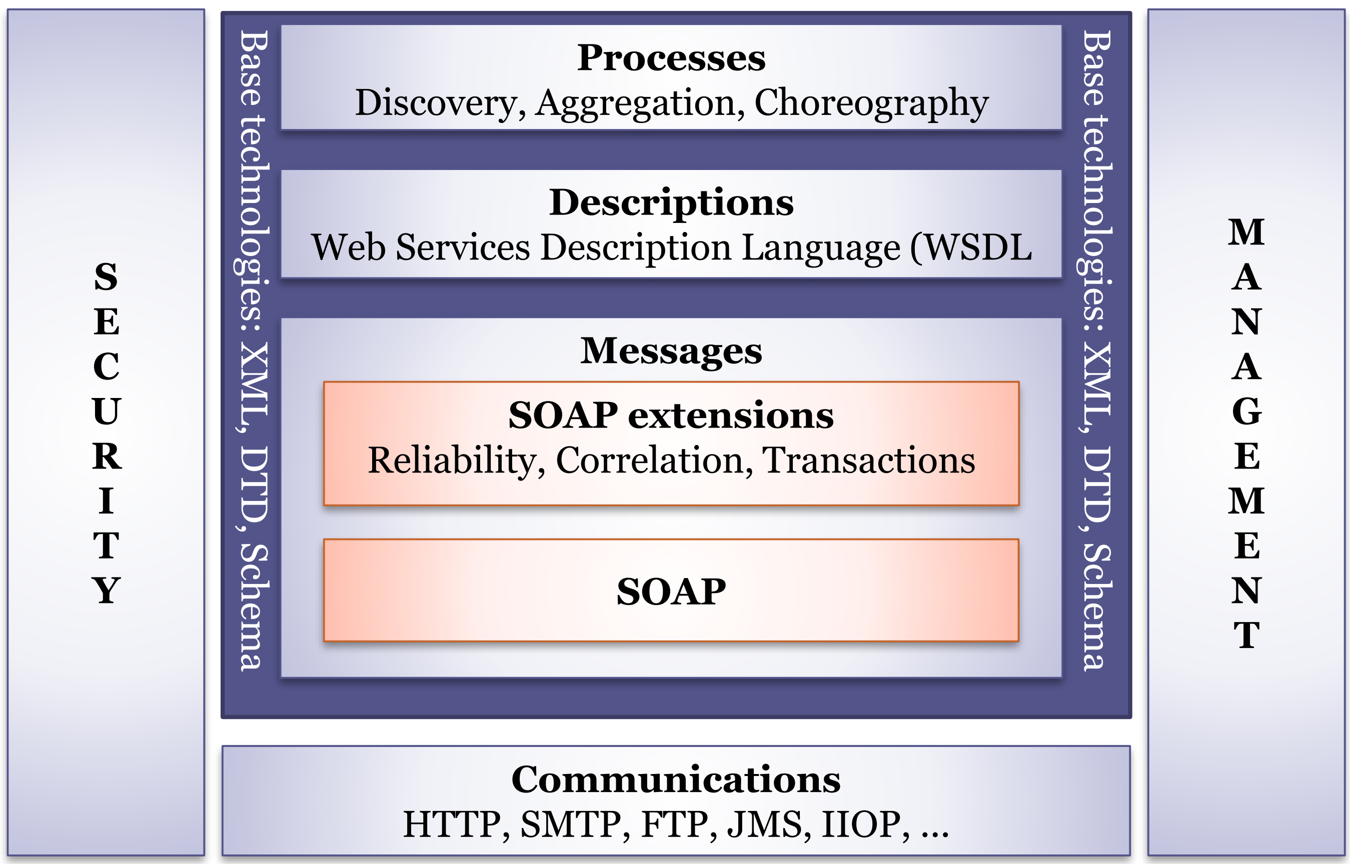
WS-*

- WS-* model = Set of specifications
  - SOAP, WSDL, UDDI, etc....
  - Proposed by W3c, OASIS, WS-I, etc.
  - Goal: Reference SOA implementation

University of Oviedo

WS-*

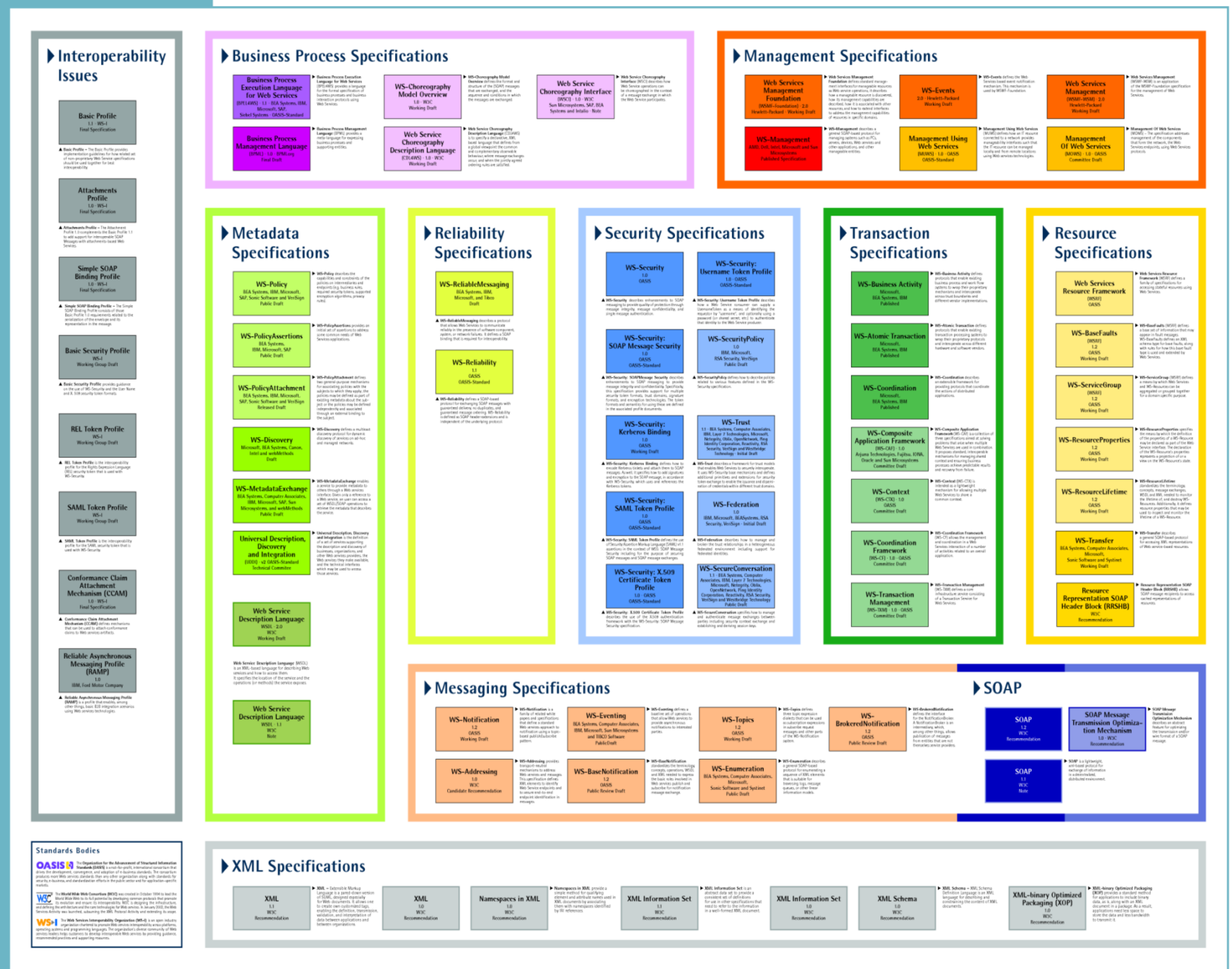
Web Services Architecture



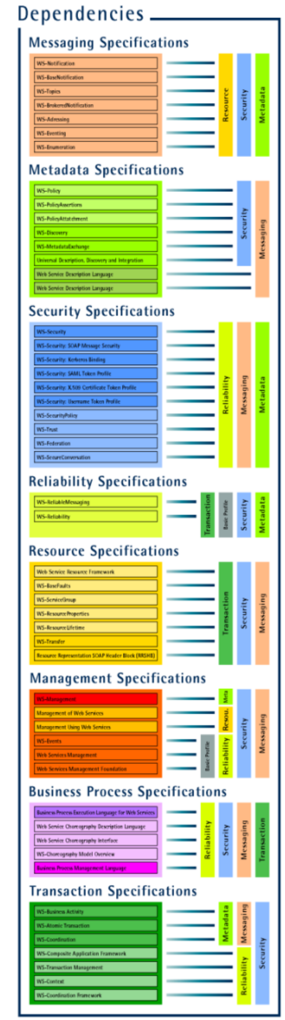
School of Computer Science

Web Services Standards

Deutsche Post
Deutsche Post AG
SOP Group
Straesschensweg 10
53113 Bonn
Phone +49 (2 28) 182 19 019
Fax +49 (2 28) 182 19 099
SOP-Group@DeutschePost.de
www.SOP-Group.com

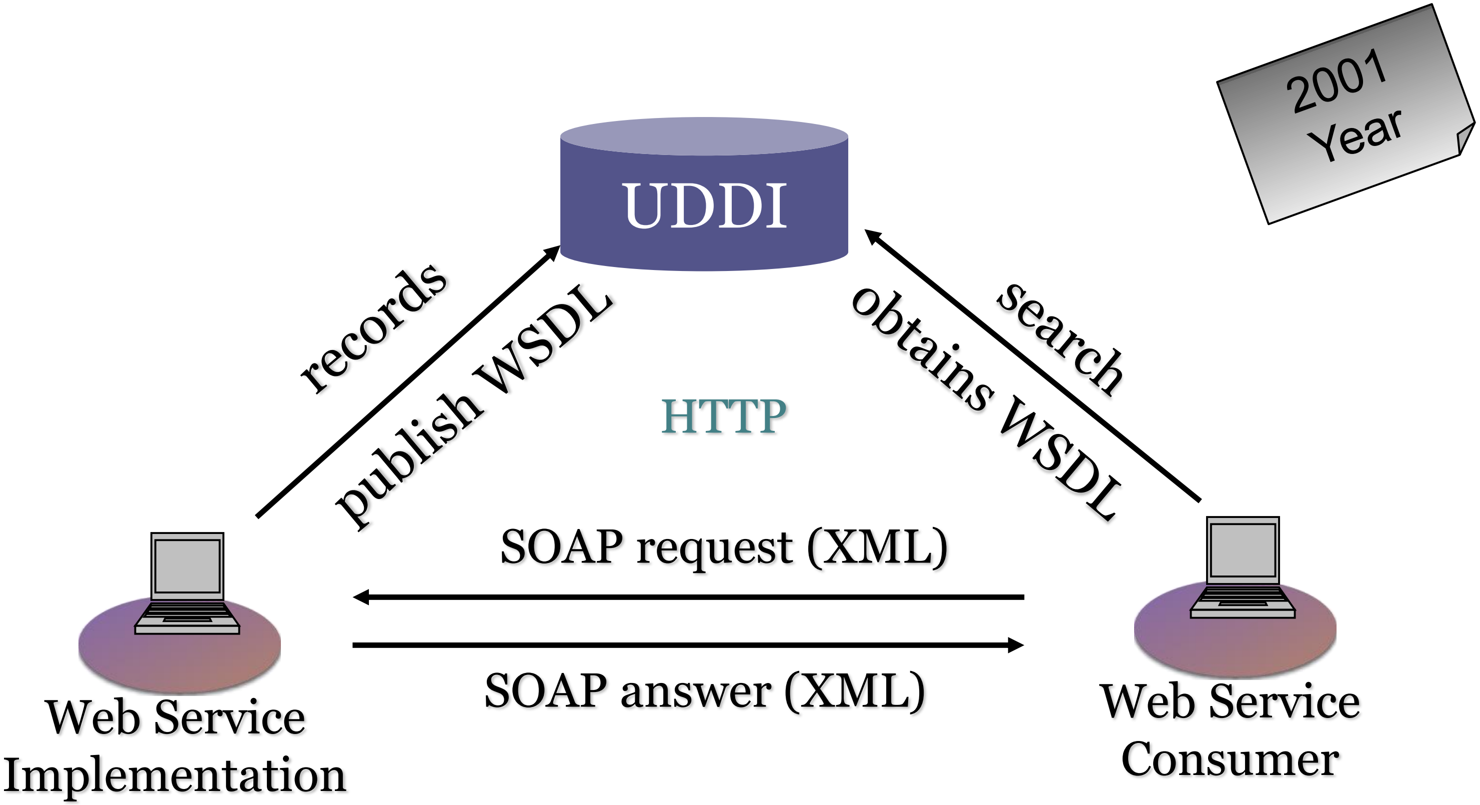


Version 2.0 · September 2005



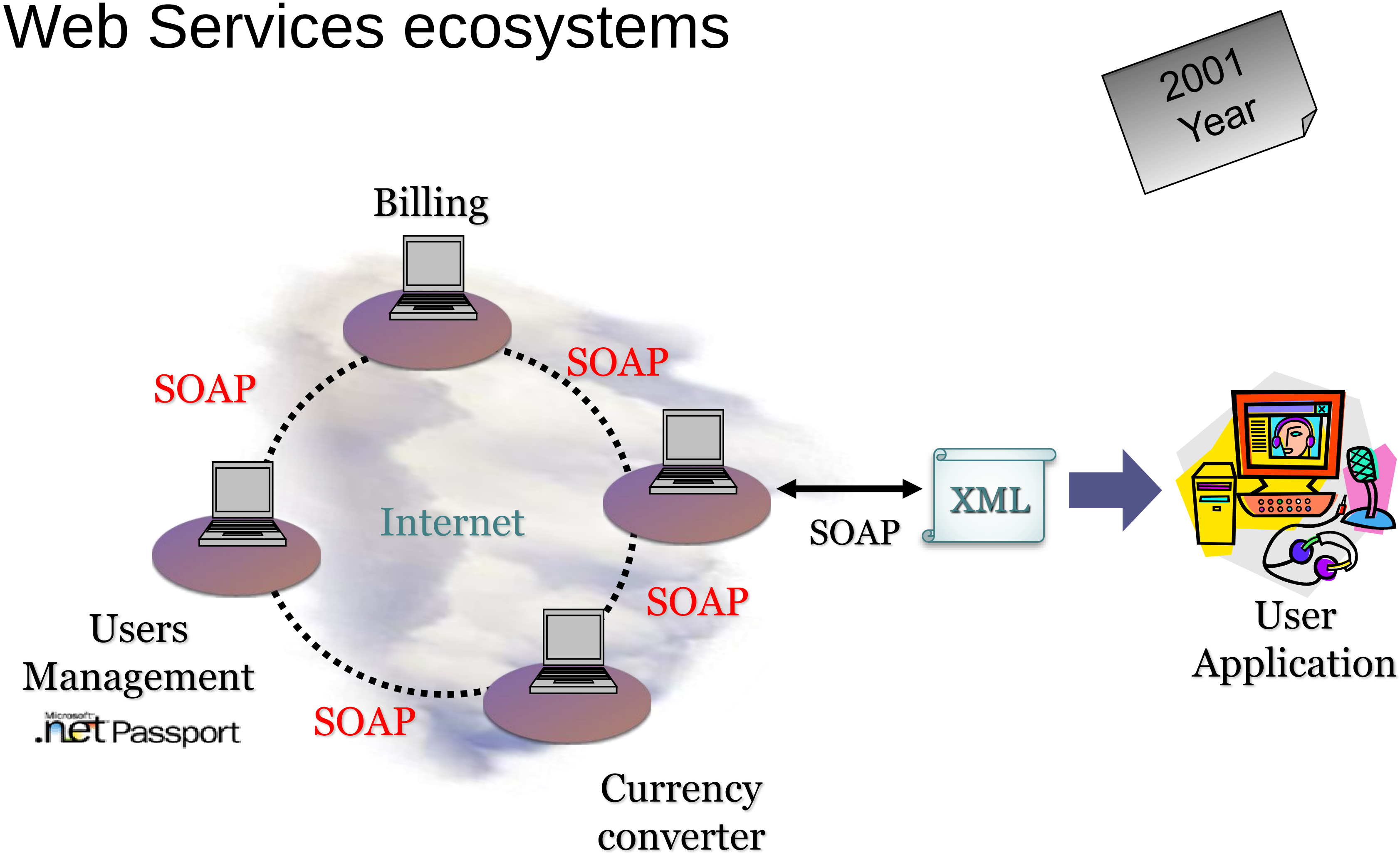
innoQ
innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Telefon +49 (0) 21 02 - 77 162 - 100
Telefax +49 (0) 21 02 - 77 16 - 01
innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Telefon +41 (0) 41 - 743 01 11
Telefax +41 (0) 41 - 743 01 19
info@innoq.com · www.innoq.com

WS-*


2001
Year
UDDI
records
publish WSDL
search
obtains WSDL
HTTP
SOAP request (XML)
SOAP answer (XML)
Web Service
Implementation
Web Service
Consumer

WS-*

Web Services ecosystems



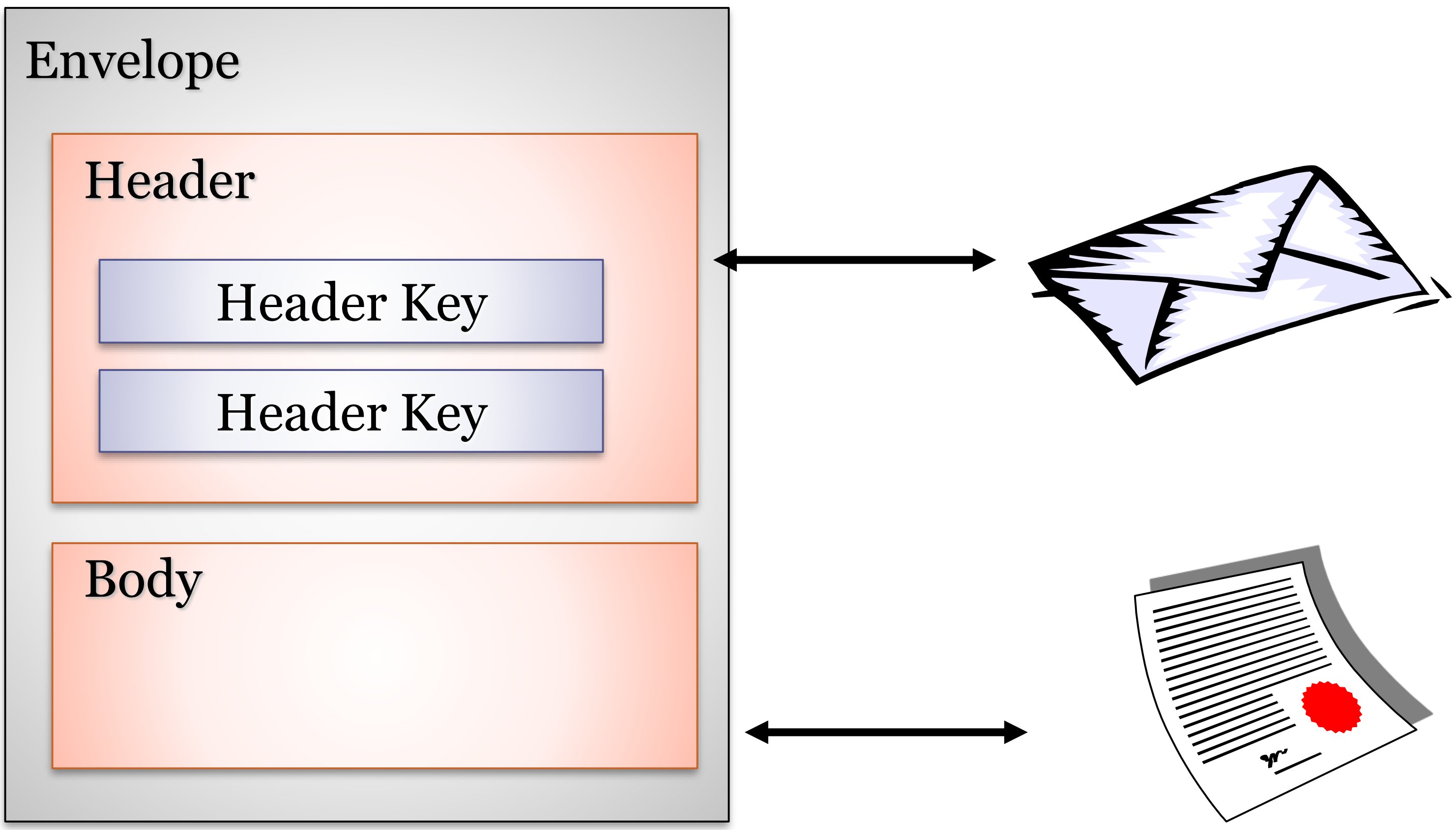
WS-*

SOAP

- Defines messages format and bindings with several protocols
  - Initially *Simple Object Access Protocol*
- Evolution
  - Developed from XML-RPC
  - SOAP 1.0 (1999), 1.1 (2000), 1.2 (2007)
  - Initial development by Microsoft
  - Posterior adoption by IBM, Sun, etc.
  - Good Industrial adoption

WS-*

Message format in SOAP



WS-*

Example of SOAP over HTTP

2001
Year

```
POST /Suma/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: longitod del mensaje
SOAPAction: "http://tempuri.org/suma"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <sum xmlns="http://tempuri.org/">
     <a>3</a>
     <b>2</b>
  </sum>
 </soap:Body>
</soap:Envelope>
```

POST ?

WS-*

Advantages

- Specifications developed by community
  - W3c, OASIS, etc.
- Industrial adoption
  - Implementations
- Integral view of web services
- Numerous extensions
  - Security, orchestration, choreography, etc.

Challenges

- Not all specifications were mature
  - Over-specification
  - Lack of implementations
- RPC style abuse
  - Uniform interface
  - Sometimes, bad use of HTTP architecture
    - Overload of GET/POST methods

WS-*

Applications

- Lots of applications have been using SOAP
  - Example: eBay (50mill. SOAP transactions/day)
- But…some popular web services ceased to offer SOAP support
  - Examples: Amazon, Google, etc.

REST

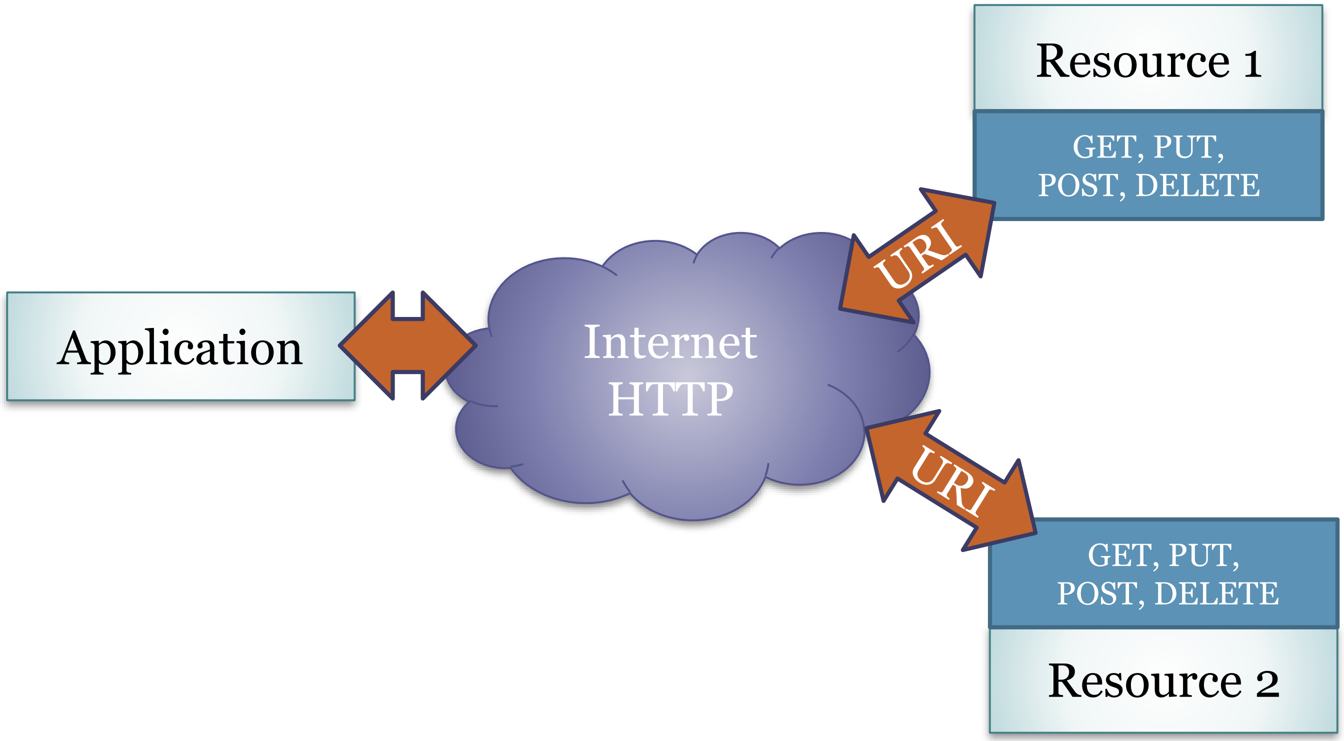
REST = REpresentational State Transfer

- Architectural style
- Source: Roy T Fielding PhD dissertation (2000)
- Inspired by Web architecture (HTTP/1.1)

REST

REST - Representational State Transfer

Diagram



REST

- Set of constraints
  - Resources with uniform interface
    - Identified by URIs
    - Fixed set of actions: GET, PUT, POST, DELETE
  - Resource representations are returned
  - Stateless
- REST = Architectural style
  - Some levels of adoption:
    - RESTful
    - REST-RPC hybrid

REST as a composed style

- Layers
- Client-Server
  - Stateless
  - Cached
  - Replicated server
- Uniform interface
  - Resource identifiers (URIs)
  - Auto-descriptive messages (MIME types)
  - Links to other resources (HATEOAS)
- Code on demand (optional)

REST uniform interface

Fixed set of operations

GET, PUT, POST, DELETE

Method	In databases	Function	Safe?	Idempotent?
PUT	≈Create/Update	Create/update	No	Yes
POST	≈Update	Create/ Update children	No	No
GET	Retrieve	Query resource info	Yes	Yes
DELETE	Delete	Delete resource	No	Yes

Safe = Does not modify server data
Idempotent = The effect of executing N-times is the same as executing it once

REST

- Stateless client/server protocol
- State handled by client
  - HATEOAS *(Hypermedia As The Engine of Application State)*
    - Representations return URIs to available options
    - Chaining of resource requests

Example: Student management

1.- Get list of students

```
GET http://example.org/student
```

Returns list of students with each student URI

2.- Get information about an specific student

```
GET http://example.org/student/id2324
```

3.- Update information of an specific student

```
PUT http://example.org/student/id2324
```

REST

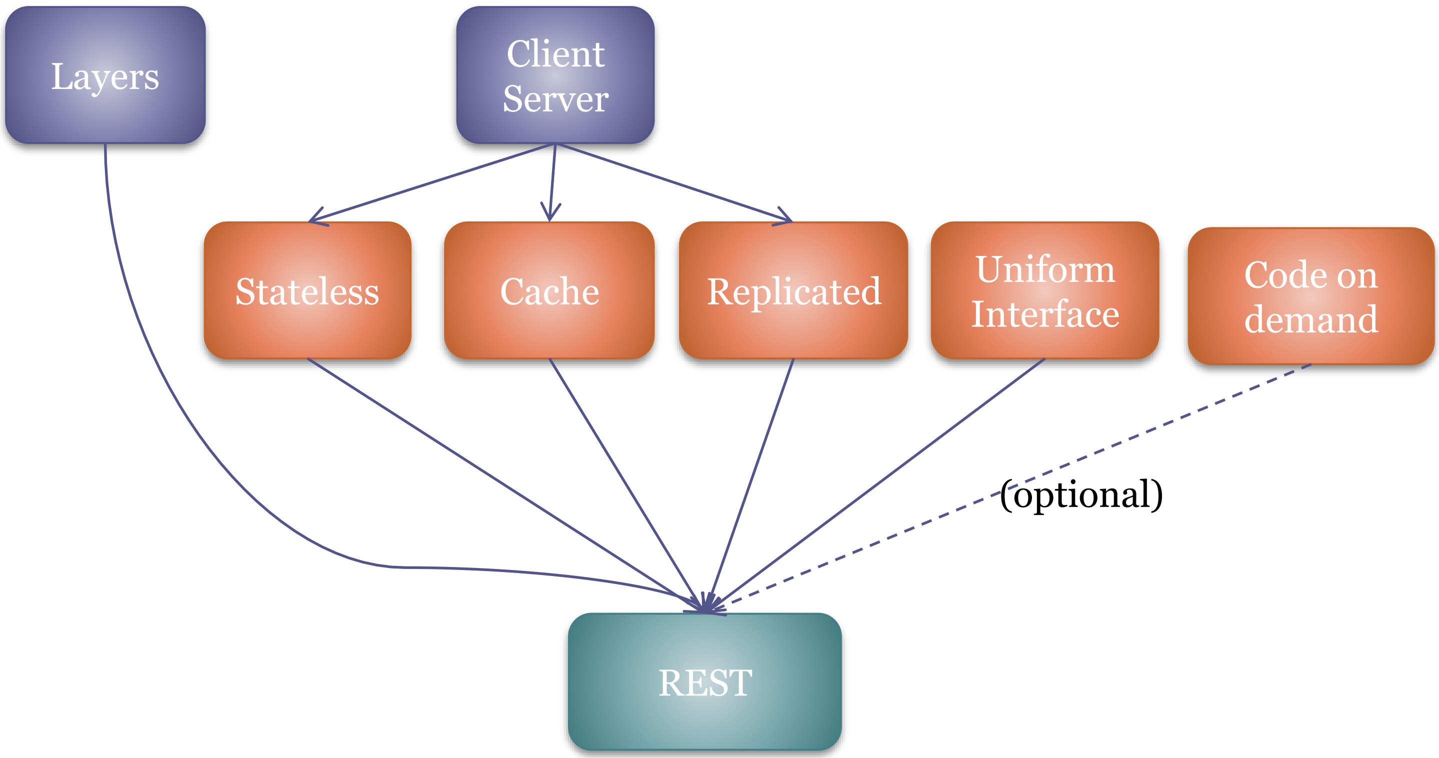
Advantages

- Client/Server
  - Separation of concerns
  - Low coupling
- Uniform interface
  - Facilitates comprehension
  - Independent development
- Scalability
- Improves answer times
- Less network load (cached)
  - Less bandwidth

Challenges

- REST partially adopted
  - Just using JSON or XML
    - Web services without contract or description
  - RPC style REST
- Difficult to incorporate other requirements
  - Security, transaction, composition, etc.

REST as a composed style



Microservices

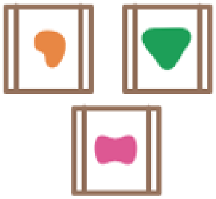
- Applications divided in small components called microservices
- Each microservice = small building block
  - Highly uncoupled
  - Focus on a specific task
- Difference with SOA
  - In SOA, services are in different applications
  - Microservices belong to the same application

http://martinfowler.com/articles/microservices.html

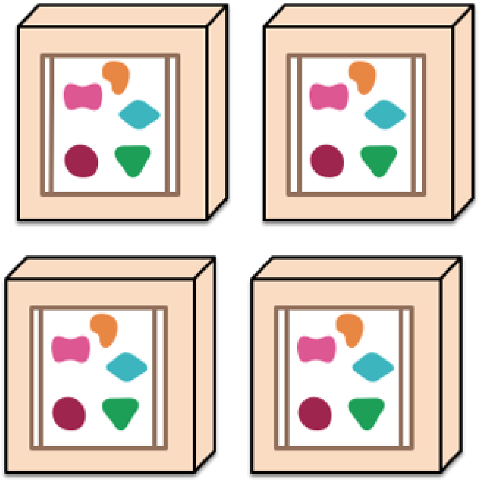
Microservices & scalability

A monolithic application puts all its functionality in a single process

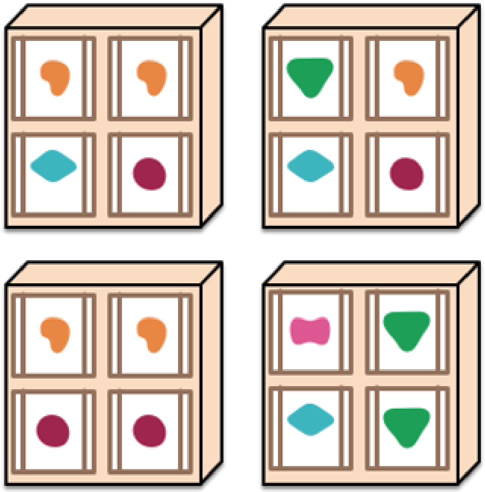
A microservices architecture puts each element of functionality into a separate service



...and scales by replicating the monolith on multiple services



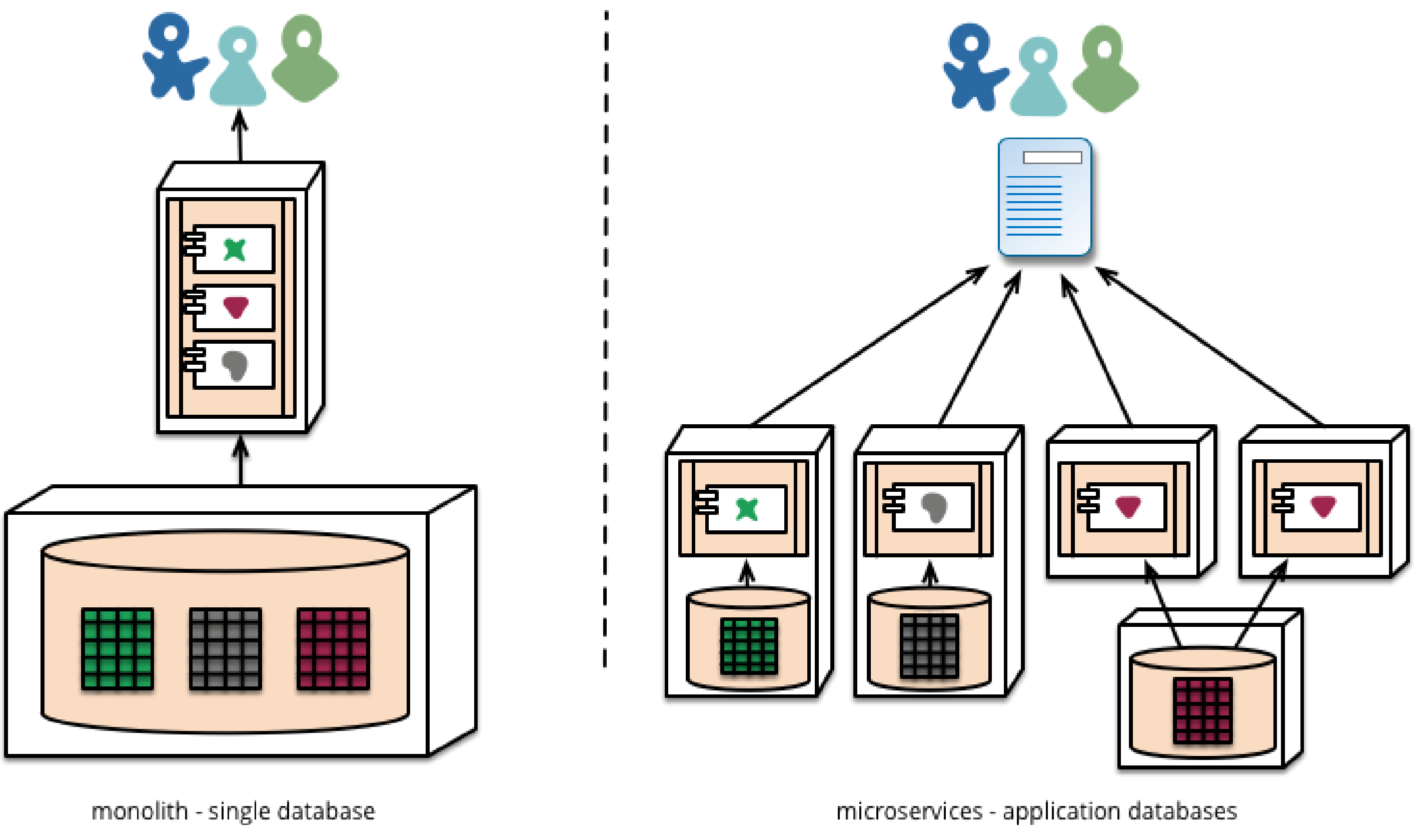
...and scales by distributing these services, replicating as needed



University of Oviedo

Microservices

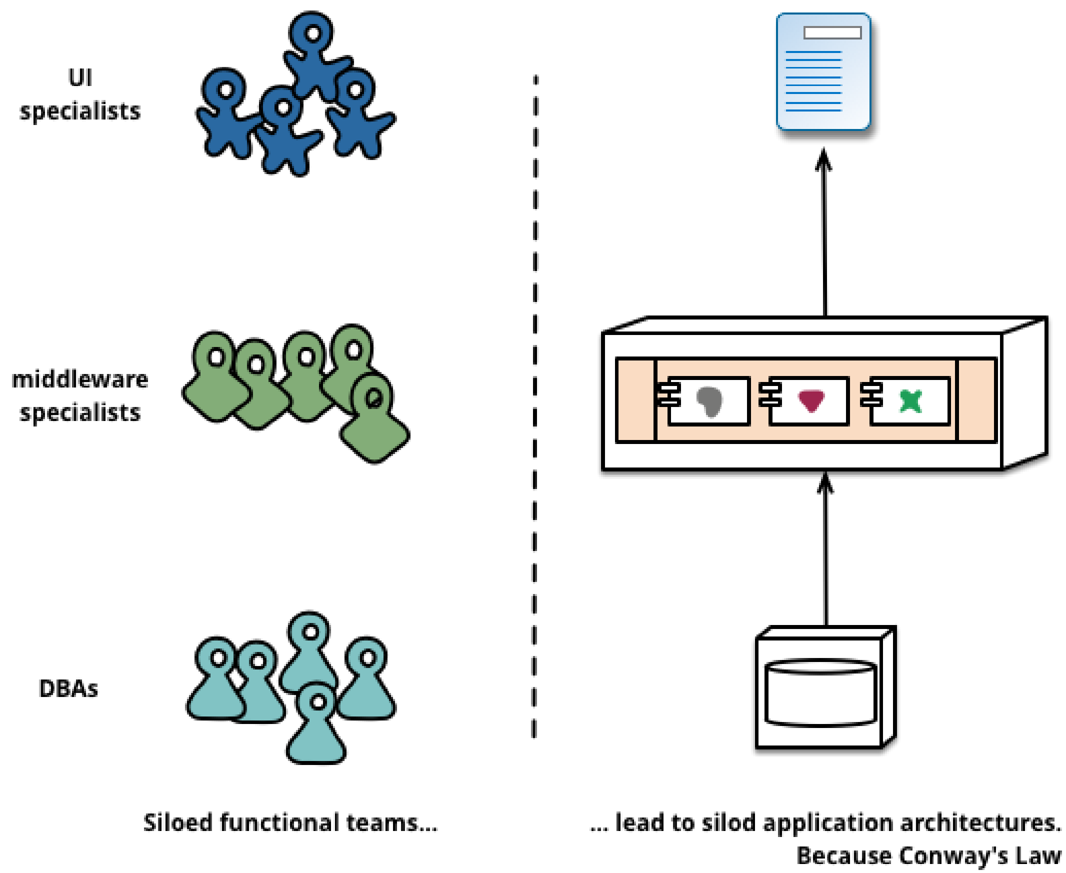
Decentralized data management



School of Computer Science

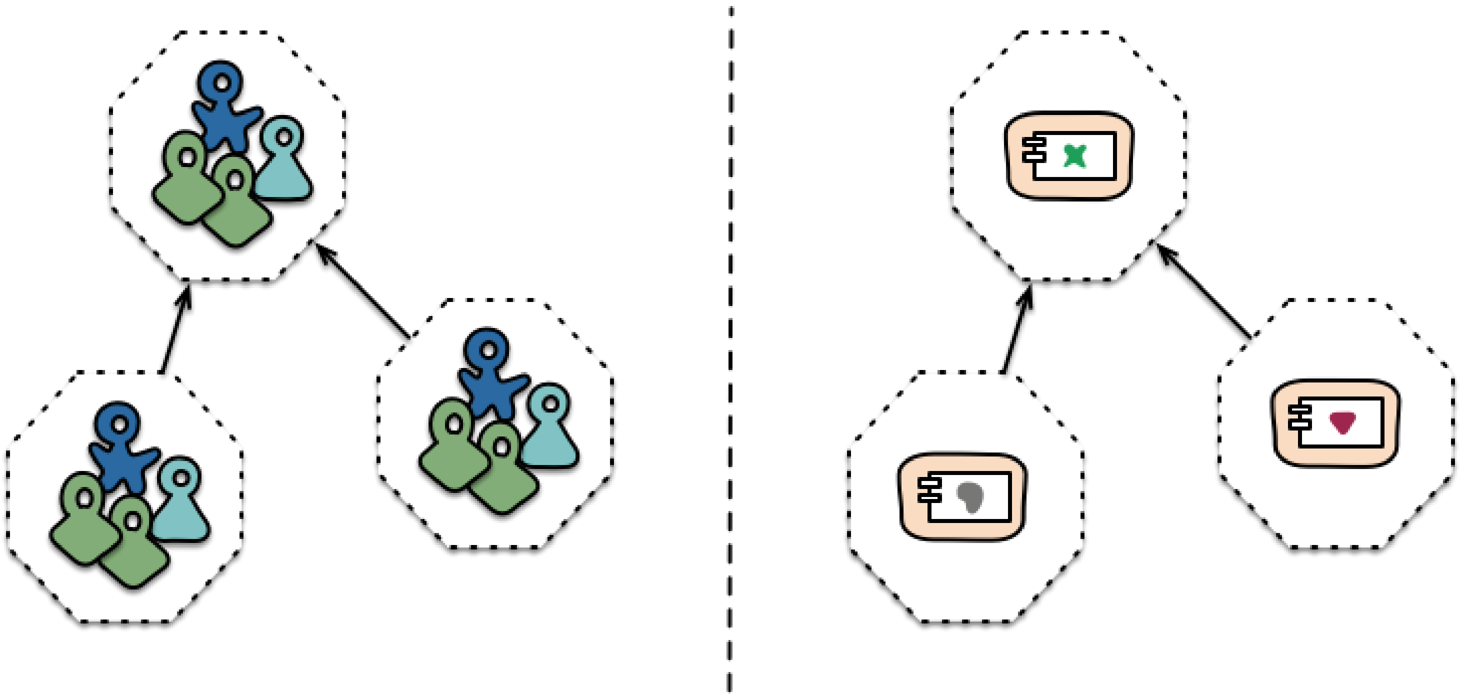
Microservices

Conway Law (traditional application)



Microservices

Conway Law (microservices): Teams are decomposed around capabilities



Cross-functional teams...

... organised around capabilities
Because Conway's Law

University of Oviedo

Microservices

Advantages

- Strong Modularity of development
  - Microservices reusability
- Independent development and deployment
  - Scalability
  - Decentralization
- Technology diversity
  - Each service can be developed using a different programming language & technology

Challenges

- Management of lots of microservices
  - Too much microservices = antipattern (nanoservices)
  - How to ensure application consistency
- Complexity
  - Distributed system management
  - New challenges: latency, message format, load balance, fault tolerance, etc.
- Testing & deployment
  - Operational complexity

http://martinfowler.com/articles/microservice-trade-offs.html

School of Computer Science

Serverless

- Also known as:
  - Function as a service (FaaS)
  - Backend as a service (BaaS)
- Applications depend on third-party services
- Developers don't need to care about servers
  - Automatic scalability
- Rich clients
  - *Single Page Applications, Mobile apps*
- Examples:
  - AWS Lambda, Google Cloud Functions, Ms Azure Functions

Serverless

Advantages

- Scalability
- Availability
- Performance
- Reduce costs
  - Operational cost
  - Only pay for the compute you need
- Time to market

Challenges

- Vendor control
- Vendor lock-in
  - Incompatibility between vendors
- Security
- Startup latency
- Integration testing
- Monitoring/debugging

End of presentation