Universidad de Oviedo

EN
English

School of Computer Science

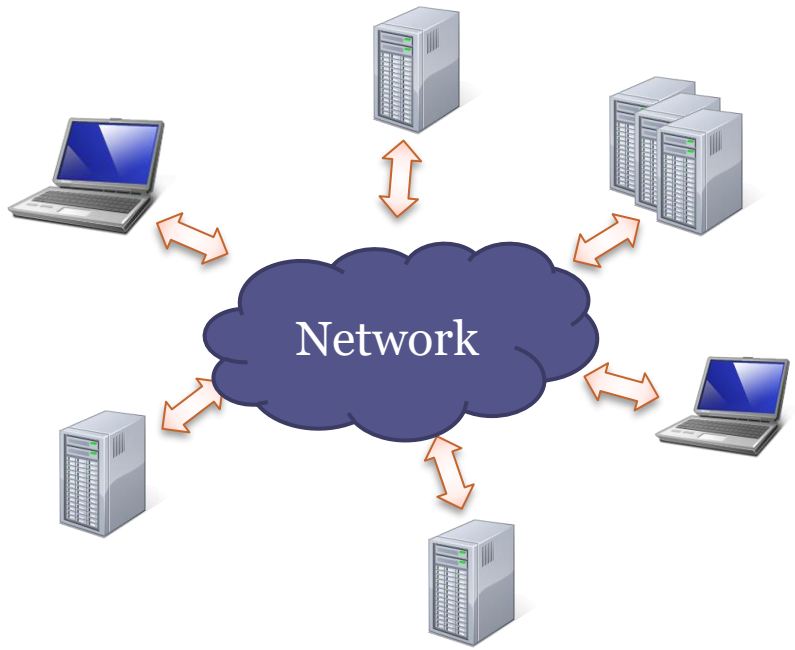
Distributed Systems
Scalable and Big Data systems

2025-26

Jose E. Labra Gayo

Distributed systems

- Integration styles
- Topologies: Hub & Spoke, Bus
- Broker pattern
- Peer-to-peer
- Service Oriented Architectures
  - WS-* vs REST
  - Service based architectures
  - Microservices
  - Serverless

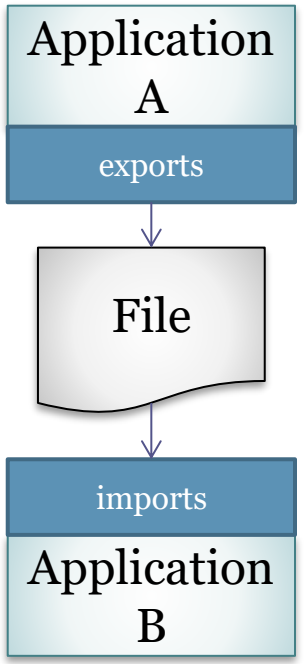


Integration styles

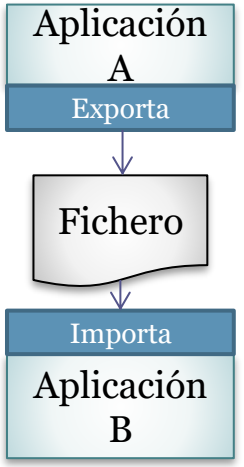
File transfer
Shared database
Remote procedure call
Messaging

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



Advantages

- Low coupling
  - A and B are independent
- Debugging
  - Analyzing intermediate files

Challenges

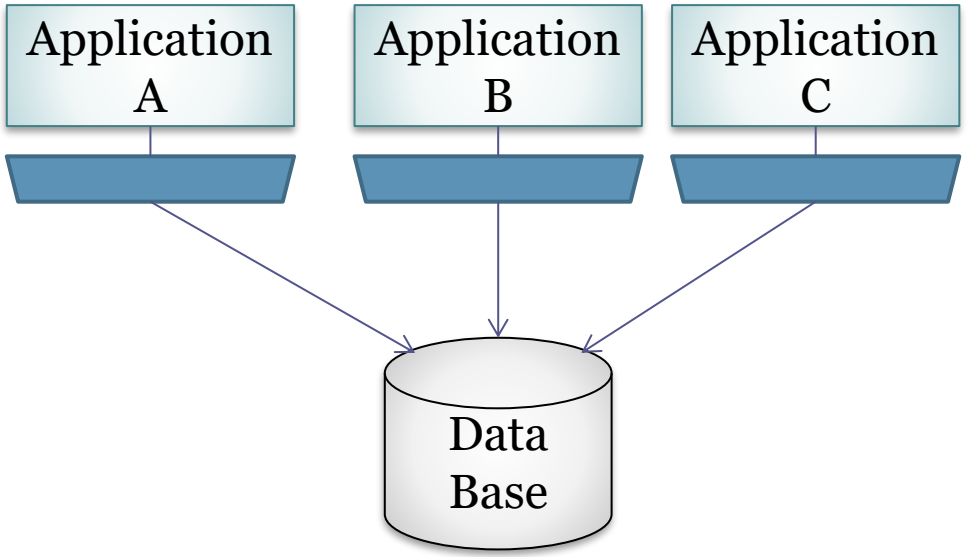
- Agreeing on a common file format
  - Can increase coupling
- Coordination
  - Once the file is sent, B can modify it ⇒ 2 files!
- It usually requires manual adjustments

Shared database

Applications store their data in a shared database

Advantage

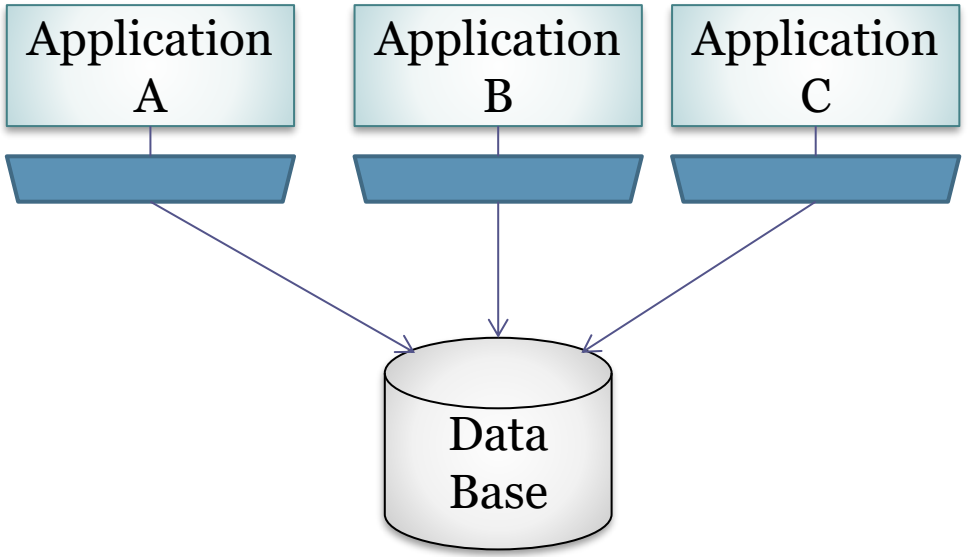
- Data are always available
- Everyone has access to the same information
- Consistency
- Familiar format
  - SQL for everything



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



Shared database

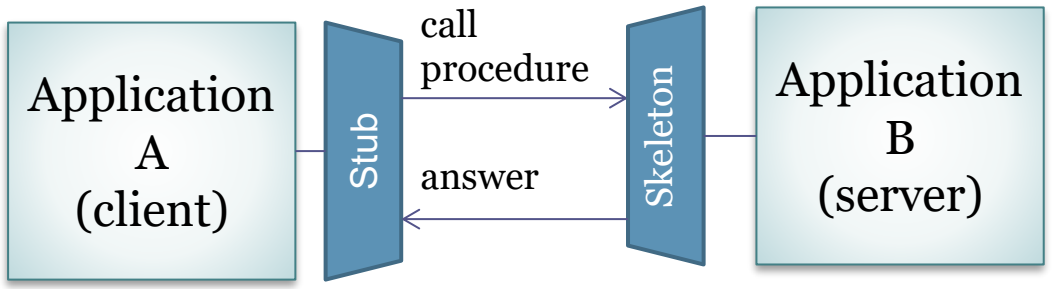
Variants

- *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 (RPC)

An application calls a function from another application that could be in another machine

- Invocation can pass parameters
- Obtains an answer

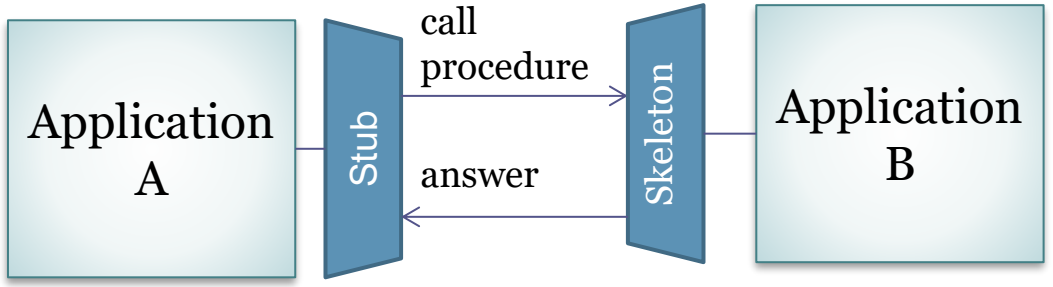


- Stub: It acts in the client side, it simulates the client that it is invoking a local method
- Skeleton: It resides on the server, acts as a gatekeeper for the service implementation

Remote Procedure Call (RPC)

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 (RPC)

Challenges

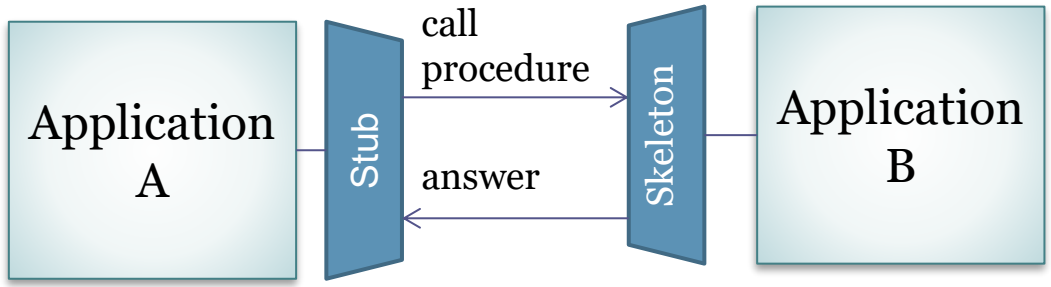
False sense of simplicity

Remote procedure ≠ 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

https://www.youtube.com/watch?v=UZxLYv5RFyI&t=54s

Remote Procedure Call

- Applications
  - RPC, RMI, CORBA, .Net Remoting, ...
- Variants
  - Basic web services (SOAP, …)
  - gRPC

Remote procedure call - gRPC

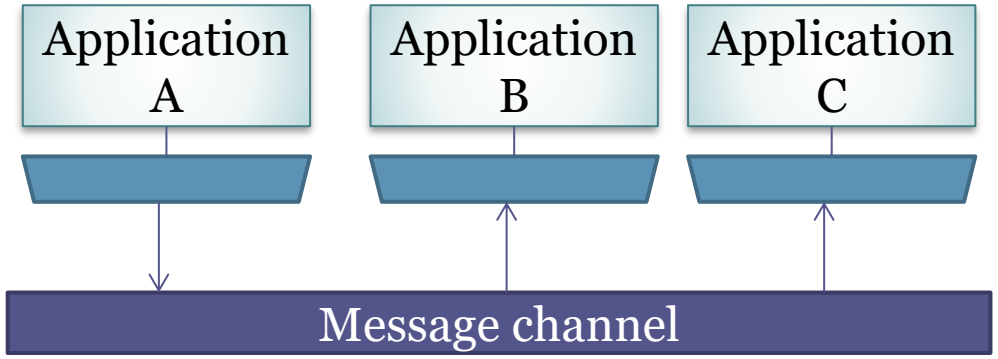
- gRPC (https://grpc.io/), proposed by Google
  - Emphasis on performance:
    - Based on Protocol Buffers
      - Binary serialization instead of XML or JSON
    - Built on http/2 transport protocol (Streaming)
  - Strongly typed contracts using .proto files

Messaging

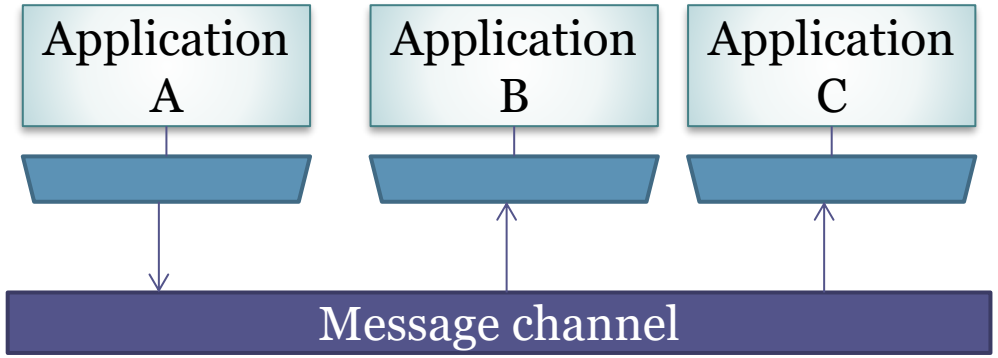
Multiple independent applications communicate sending messages through a channel

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

Challenges

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

Integration topologies

Hub & Spoke

Bus

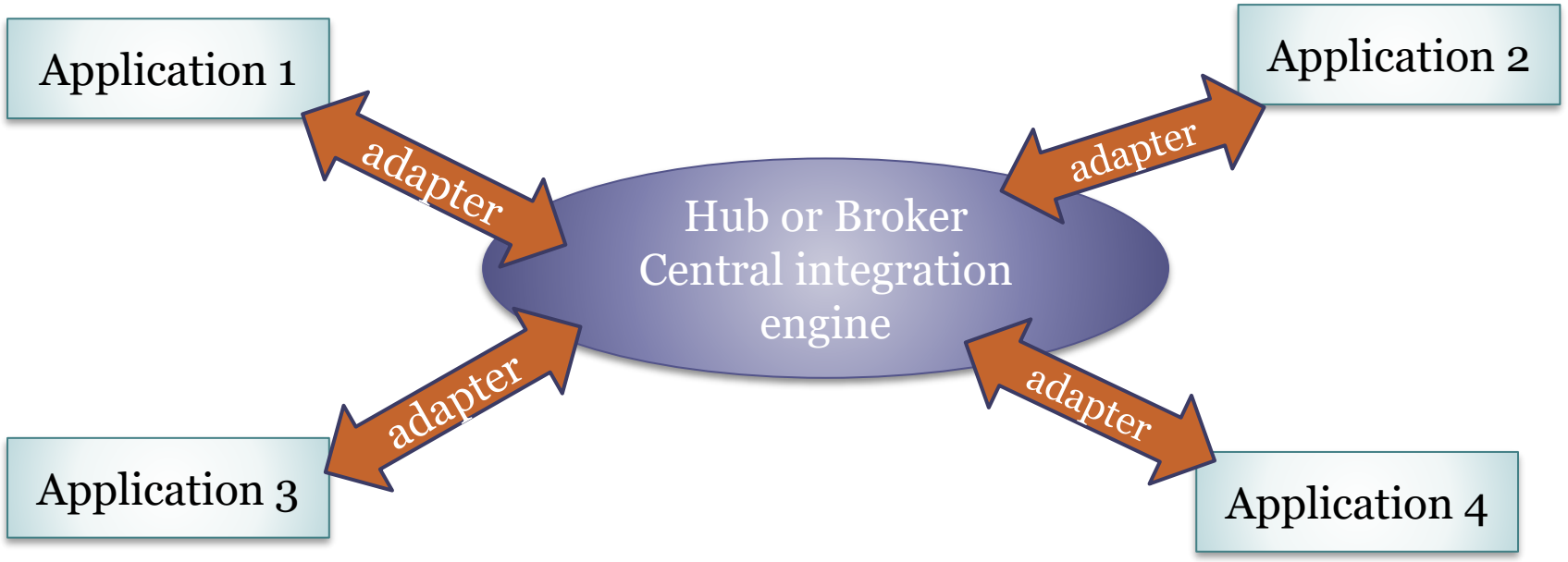
Hub & Spoke (Centralized)

Hub = Centralized message Broker

It is in charge of integration

Every spoke (≈ Application) connects directly with the Hub

Hub handles the integration logic = single point of failure

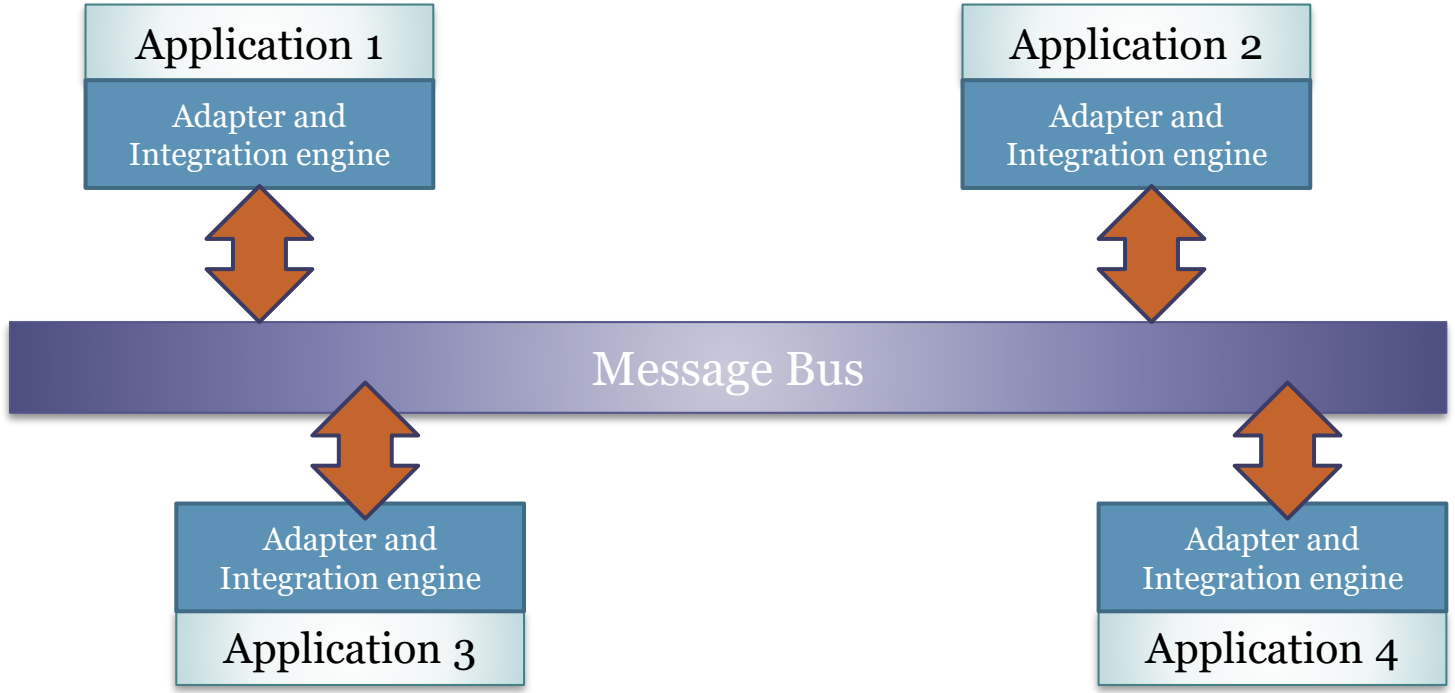


Bus

Each application contains its own integration machine

- The logic (intelligence) is moved to the endpoints
- Bus ≈ dumb pipe that carries "smart" messages

Publish/Subscribe style



Enterprise Service Bus (ESB)

- Defines the messaging backbone
  - Protocol conversion
  - Data transformation
  - Routing
- Despite the "Bus", ESB is closer to Hub & Spoke than to bus

MOM (Message Oriented Middleware)

Message infrastructure

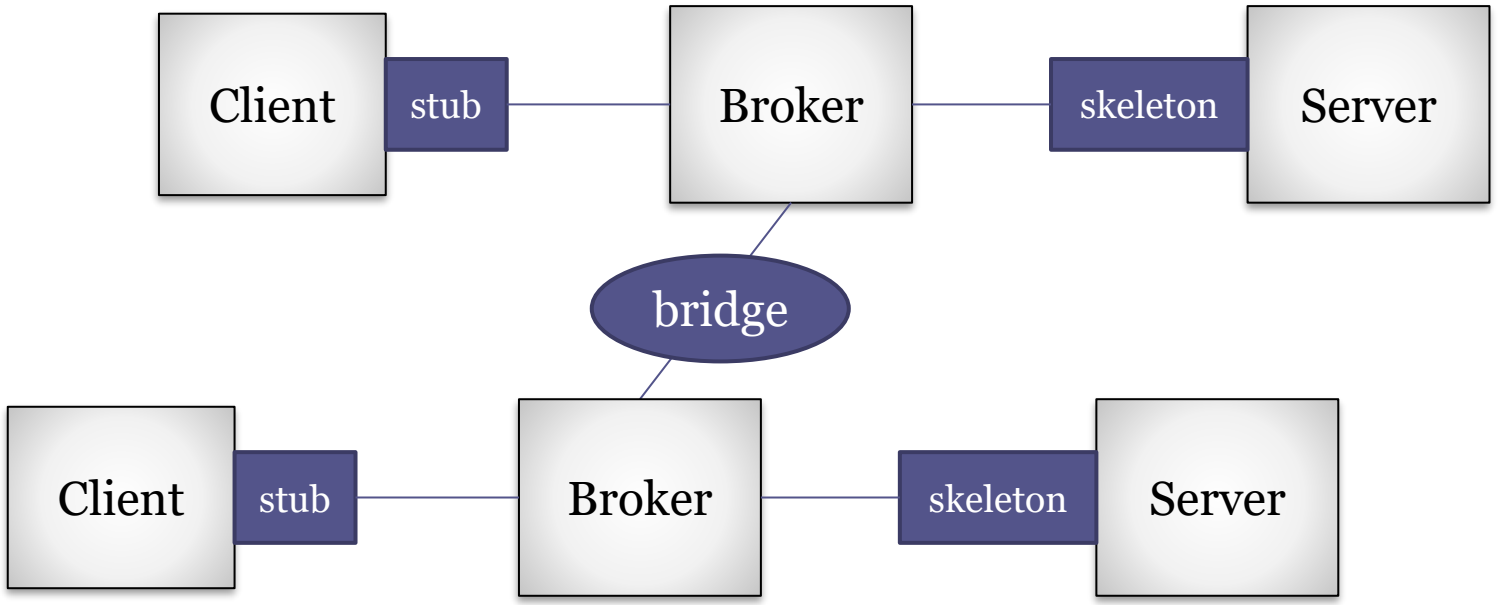
Software or API that is focused on sending/receiving messages

MOM handles:

- Asynchronicity (Fire and forget):
  - A system can send a message without waiting for a response
- Reliability and persistence:
  - In case of a crash, the messages are not lost
- Throttling and load leveling
  - If a system receives too many messages, the MOM can act as a buffer

Broker

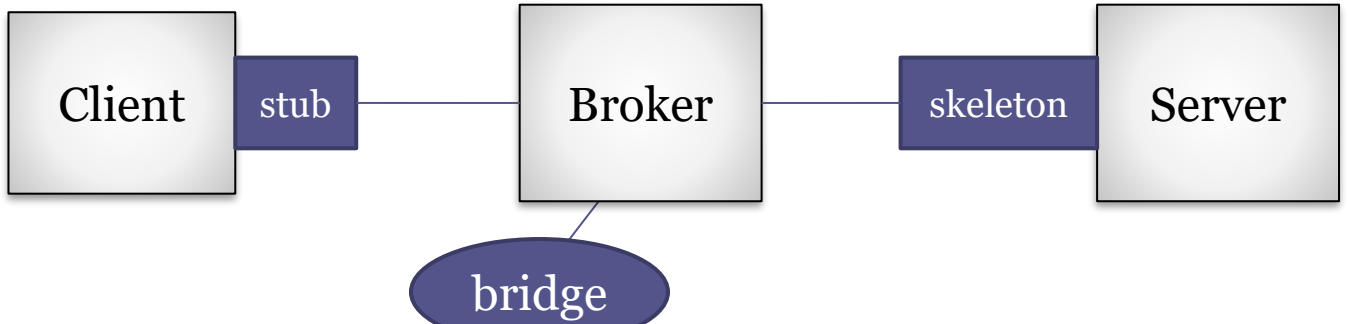
Intermediate node that manages communication between a client and a server



Broker

Elements

- Broker
  - Manages communication
- Client: Sends requests
  - Client Proxy: *stub*
- Server: Returns answers
  - Server Proxy: *skeleton*
- Bridge: Can connect brokers



Broker

Advantages

- Separation of concerns
  - Delegates low level communication aspects to the broker
  - Separate maintenance
  - Reusability
- Servers are independent from clients
- Portability
  - Broker = low level aspects
- Interoperability
  - Using *bridges*

Challenges

- Performance
  - Adds an indirection layer
- Can increase coupling between components
- Broker = single point of failure

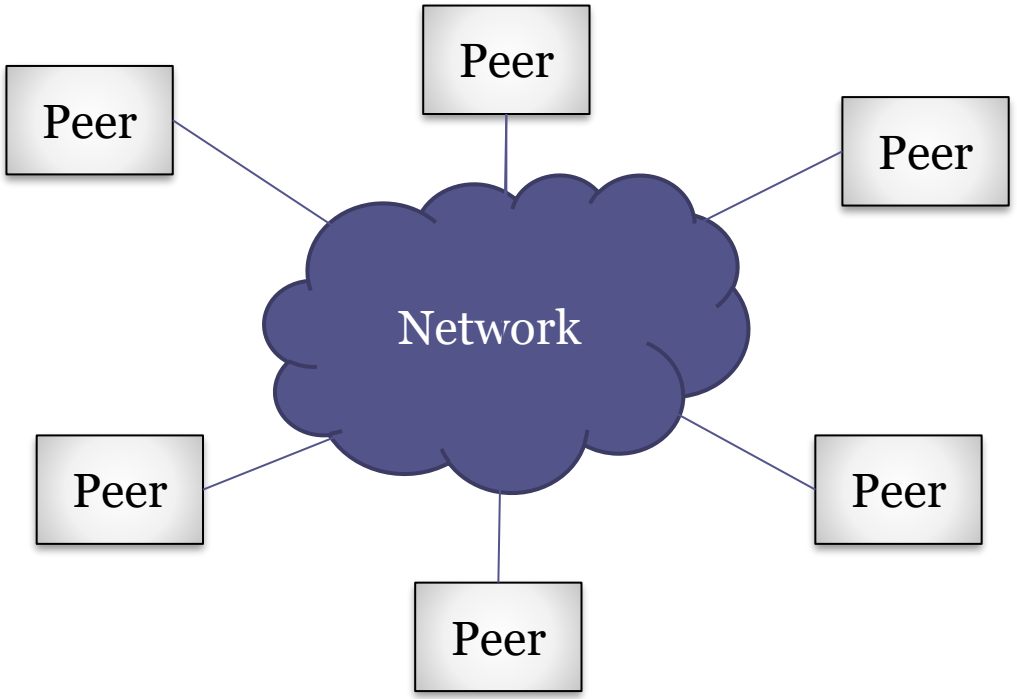
Broker

Applications

- CORBA and distributed systems
- Android uses a variation of Broker pattern

Peer-to-Peer

Equal and autonomous nodes (*peers*) that communicate between them.



Peer-to-Peer

- Elements
  - Computational nodes: *peers*
    - They contain their own state and control thread
  - Network protocol
- Constraints
  - There is no main node
  - All peers are equal

Peer-to-Peer

Advantages

- Decentralized information and control
- Fault tolerance
  - There is no single point of failure
  - A failure in one peer does not compromise the whole system

Challenges

- Keeping the state of the system
  - Complexity of the protocol
- Bandwidth Limitations
  - Network and protocol latency
- Security
  - Detect malicious *peers*

Peer-to-Peer

- Popular applications
  - Napster, BitTorrent, Gnutella, ...
  - This architecture style is not only to share files
    - e-Commerce (B2B)
    - Collaborative systems
    - Sensor networks
    - Blockchain
    - ...
- Variants
  - Super-peers

Service Oriented Architectures

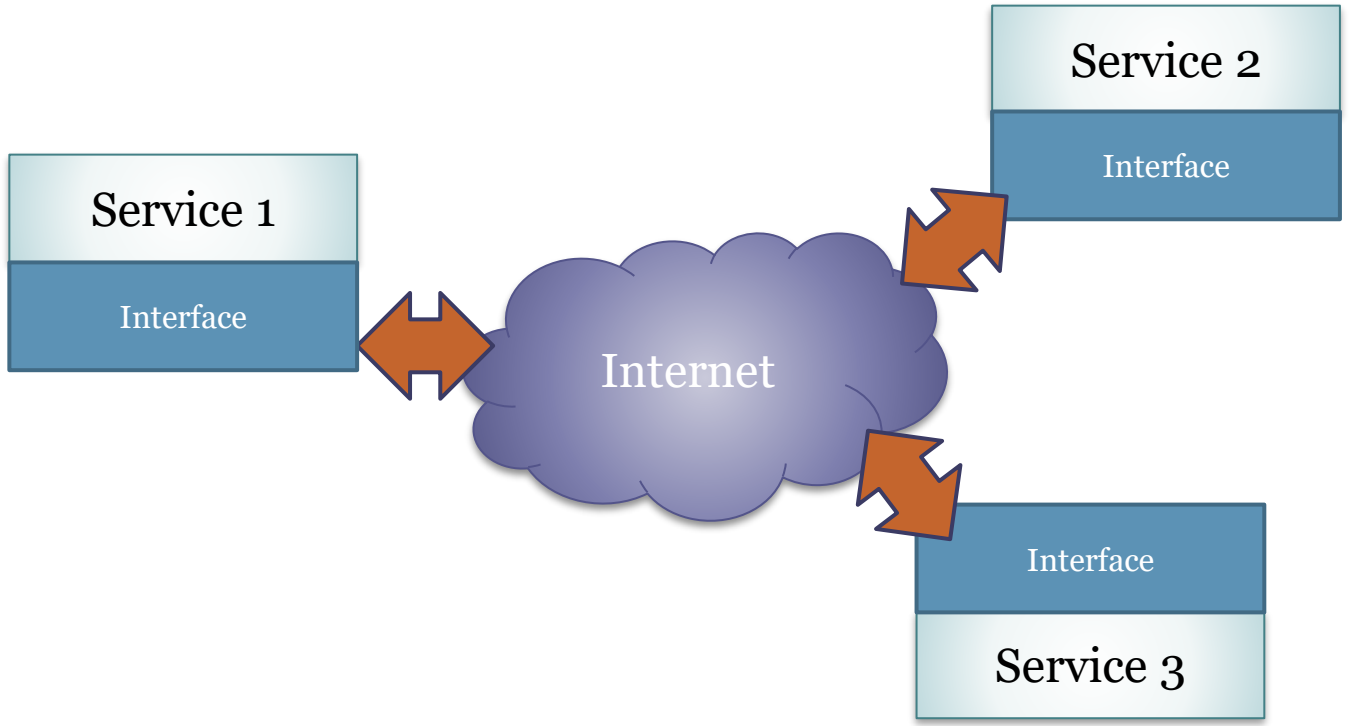
SOA – Service Oriented Architectures

- SOA
- WS-* vs REST
- Service based architectures
- Microservices
- Serverless

SOA

SOA = Service Oriented Architecture

Services are defined by an interface



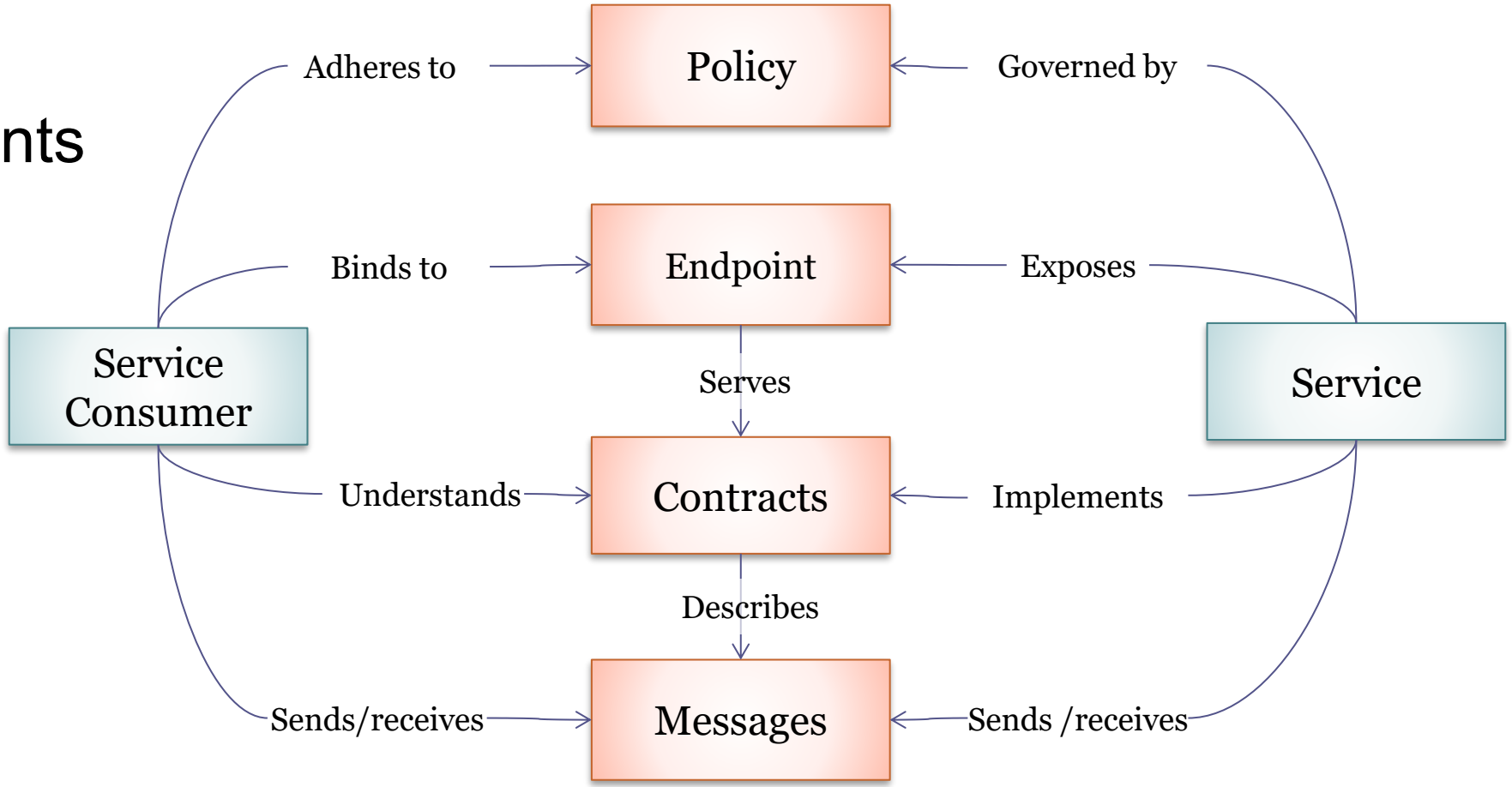
SOA

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

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

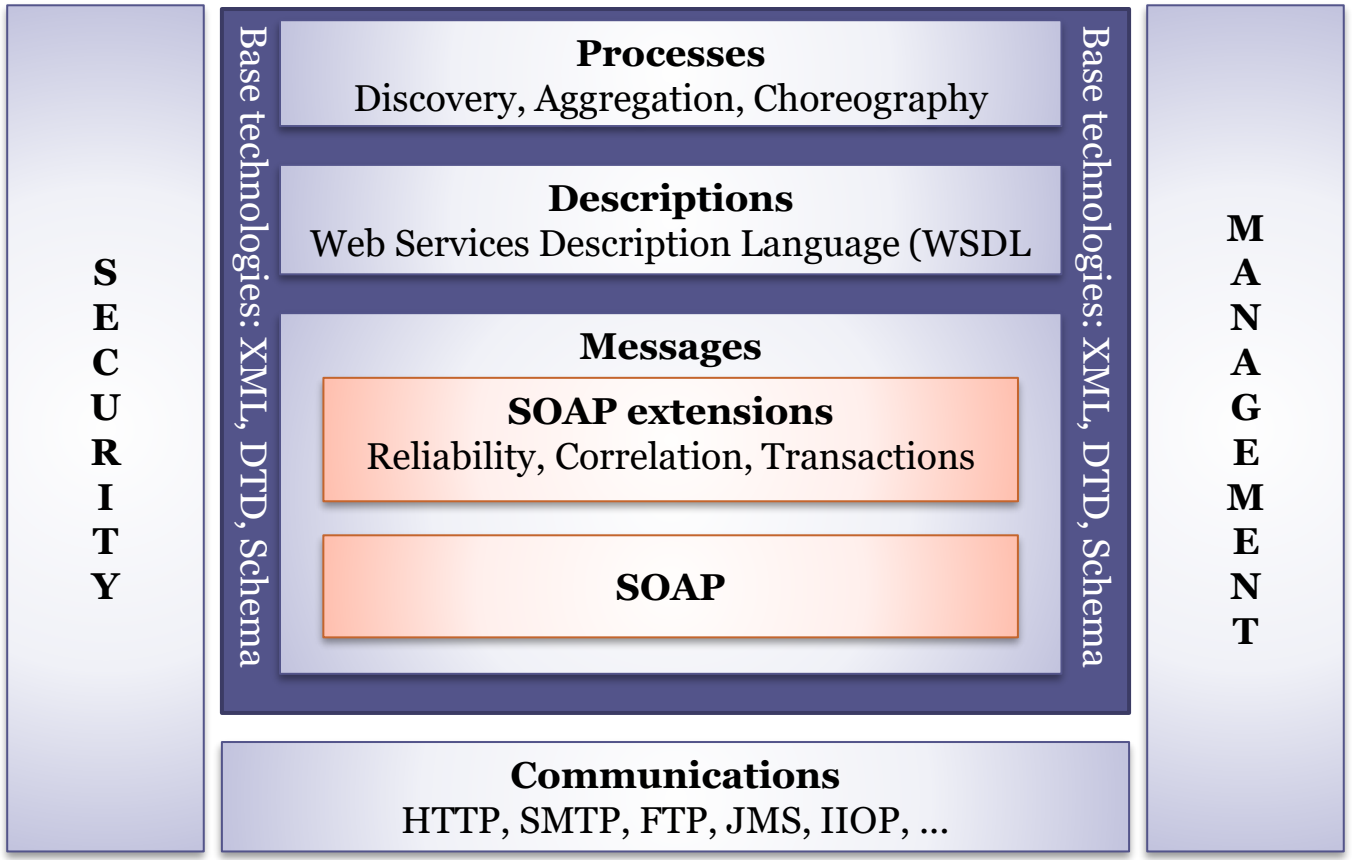
WS-*

WS-* model = Set of specifications

- SOAP, WSDL, UDDI, etc....
- Proposed by W3C, OASIS, WS-I, etc.
- Goal: Reference SOA implementation

WS-*

Web Services Architecture



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

Interoperability Issues

- Basic Profile 1.1 - WS-I - Final Specification
- Attachments Profile 1.0 - WS-I - Final Specification
- Simple SOAP Binding Profile 1.0 - WS-I - Final Specification
- Basic Security Profile WS-I - Working Group Draft1
- REL Token Profile WS-I - Working Group Draft1
- SAML Token Profile WS-I - Working Group Draft1
- Conformance Claim Attachment Mechanism (CCAM) 1.0 - WS-I - Final Specification
- Reliable Asynchronous Messaging Profile (RAMP) 1.0 - IBM, Fred Motor Company

Business Process Specifications

- Business Process Execution Language for Web Services (BPEL4WS) - 1.1 - IBM, Systems, IBM, Microsoft, SAP, Siebel Systems - OASIS-Standard
- WS-Choreography Model Overview - 1.0 - W3C - Working Draft
- Web Service Choreography Interface (WSCI) - 1.0 - W3C - Sun Microsystems, SAP, BEA Systems and Intalio - Note
- Business Process Management Language (BPML) - 1.0 - BPMI.org - Final Draft
- Web Service Choreography Description Language (CDL4WS) - 1.0 - W3C - Working Draft

Management Specifications

- Web Services Management Foundation (WSMF-Foundation) - 2.0 - Hewlett-Packard - Working Draft
- WS-Events - 2.0 - Hewlett-Packard - Working Draft
- Web Services Management (WSMF-WSM) - 2.0 - Hewlett-Packard - Working Draft
- WS-Management - AMD, Dell, Intel, Microsoft and Sun Microsystems - Published Specification
- Management Using Web Services (MUWS) - 1.0 - OASIS - OASIS-Standard
- Management Of Web Services (MOWS) - 1.0 - OASIS - Committee Draft

Metadata Specifications

- WS-Policy - BEA Systems, IBM, Microsoft, SAP, Sonic Software and VeriSign - Public Draft
- WS-PolicyAssertions - BEA Systems, IBM, Microsoft, SAP - Public Draft
- WS-PolicyAttachment - BEA Systems, IBM, Microsoft, SAP, Sonic Software and VeriSign - Released Draft
- WS-Discovery - Microsoft, BEA Systems, Canon, Intel and webMethods - Draft
- WS-MetadataExchange - BEA Systems, Computer Associates, IBM, Microsoft, SAP, Sun Microsystems, and webMethods - Public Draft
- Universal Description, Discovery and Integration (UDDI) - v2 - OASIS-Standard - Technical Committee
- Web Service Description Language - WSDL - 2.0 - W3C - Working Draft
- Web Service Description Language - WSDL - 1.1 - W3C - Note

Web Service Description Language (WSDL) is an XML-based language for describing Web services and how to access them. Specifically, WSDL describes a service and the operations (or methods) the service exposes.

Reliability Specifications

- WS-ReliableMessaging - BEA Systems, IBM, Microsoft, and Tibco - Draft
- WS-Reliability - 1.1 - OASIS - OASIS-Standard

Security Specifications

- WS-Security - 1.0 - OASIS
- WS-Security: Username Token Profile - 1.0 - OASIS - OASIS-Standard
- WS-Security: SOAP Message Security - 1.0 - OASIS - OASIS-Standard
- WS-SecurityPolicy - 1.0 - IBM, Microsoft, RSA Security, VeriSign - Public Draft
- WS-Security: Kerberos Binding - 1.0 - OASIS - Working Draft
- WS-Trust - 1.1 - BEA Systems, Computer Associates, IBM, Layer 7 Technologies, Microsoft, Netegrity, Ofelia, OpenNetwork, Ping Identity Corporation, Reactivity, RSA Security, VeriSign and Westbridge Technology - Initial Draft
- WS-Security: SAML Token Profile - 1.0 - OASIS - OASIS-Standard
- WS-Federation - 1.0 - IBM, Microsoft, IBM,Systems, RSA Security, VeriSign - Initial Draft
- WS-Security: X.509 Certificate Token Profile - 1.0 - OASIS - OASIS-Standard
- WS-SecureConversation - 1.1 - BEA Systems, Computer Associates, IBM, Layer 7 Technologies, Microsoft, Netegrity, Ofelia, OpenNetwork, Ping Identity Corporation, Reactivity, RSA Security, VeriSign and Westbridge Technology - Public Draft

Transaction Specifications

- WS-Business Activity - Microsoft, BEA Systems, IBM - Published
- WS-Atomic Transaction - Microsoft, BEA Systems, IBM - Published
- WS-Coordination - Microsoft, BEA Systems, IBM - Published
- WS-Composite Application Framework (WS-CAF) - 1.0 - Arjuna Technologies, Fujitsu, IONA, Oracle and Sun Microsystems - Committee Draft
- WS-Context (WS-CTX) - 1.0 - OASIS - Committee Draft
- WS-Coordination Framework - 1.0 - OASIS - Committee Draft
- WS-Transaction Management (WS-TXM) - 1.0 - OASIS - Committee Draft

Resource Specifications

- Web Services Resource Framework (WSRF) - OASIS
- WS-BaseFaults (WSRF) - 1.2 - OASIS - Working Draft
- WS-ServiceGroup (WSRF) - 1.2 - OASIS - Working Draft
- WS-ResourceProperties - 1.2 - OASIS - Working Draft
- WS-ResourceLifetime - 1.2 - OASIS - Working Draft
- WS-Transfer - BEA Systems, Computer Associates, Microsoft, Sonic Software and Systinet - Working Draft
- Resource Representation SOAP Header Block (RRSHB) - W3C - Recommendation

Messaging Specifications

- WS-Notification - 1.2 - OASIS - Working Draft
- WS-Eventing - BEA Systems, Computer Associates, IBM, Microsoft, Sun Microsystems and TIBCO Software - Public Draft
- WS-Topics - 1.2 - OASIS - Working Draft
- WS-BrokeredNotification - 1.2 - OASIS - Public Review Draft
- WS-Addressing - 1.0 - W3C - Candidate Recommendation
- WS-BaseNotification - 1.2 - OASIS - Public Review Draft
- WS-Enumeration - BEA Systems, Computer Associates, Microsoft, Sonic Software and Systinet - Public Draft

SOAP

- SOAP - 1.2 - W3C - Recommendation
- SOAP Message Transmission Optimization Mechanism - 1.0 - W3C - Recommendation
- SOAP - 1.1 - W3C - Note

XML Specifications

- XML - 1.1 - W3C - Recommendation
- XML - 1.0 - W3C - Recommendation
- Namespaces in XML - 1.0 - W3C - Recommendation
- XML Information Set - 1.0 - W3C - Recommendation
- XML Information Set - 1.0 - W3C - Recommendation
- XML Schema - 1.0 - W3C - Recommendation
- XML-binary Optimized Packaging (XOP) - 1.0 - W3C - Recommendation

Standards Bodies

- OASIS — The Organization for the Advancement of Structured Information Standards (OASIS) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. The consortium produces more Web services standards than any other organization along with standards for security, business, and standardization efforts in the public sector and for application-specific markets.
- W3C — The World Wide Web Consortium (W3C) was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C is designing the infrastructure, establishing the architecture and the core technologies for Web services. In January 2002, the Web Services Activity was launched, subsuming the XML Protocol Activity and extending its scope.
- WS-I — The Web Services Interoperability Organization (WS-I) is an open industry organization chartered to promote Web services interoperability across platforms, operating systems and programming languages. The organization's diverse community of Web services leaders helps customers to develop interoperable Web services by providing guidance, recommended practices and supporting resources.

Dependencies

Messaging Specifications

- WS-Notification
- WS-BaseNotification
- WS-Topics
- WS-BrokeredNotification
- WS-Addressing
- WS-Eventing
- WS-Enumeration

Resource, Security, Metadata

Metadata Specifications

- WS-Policy
- WS-PolicyAssertions
- WS-PolicyAttachment
- WS-Discovery
- WS-MetadataExchange
- Universal Description, Discovery and Integration
- Web Service Description Language

Security, Messaging

Security Specifications

- WS-Security
- WS-Security: SOAP Message Security
- WS-Security: Kerberos Binding
- WS-Security: SAML Token Profile
- WS-Security: X.509 Certificate Token Profile
- WS-Security: Username Token Profile
- WS-SecurityPolicy
- WS-Trust
- WS-Federation
- WS-SecureConversation

Reliability, Messaging, Metadata

Reliability Specifications

- WS-ReliableMessaging
- WS-Reliability

Transaction, Basic Profile, Security, Metadata

Resource Specifications

- Web Services Resource Framework
- WS-BaseFaults
- WS-ServiceGroup
- WS-ResourceProperties
- WS-ResourceLifetime
- WS-Transfer
- Resource Representation SOAP Header Block (RRSHB)

Transaction, Security, Messaging

Management Specifications

- WS-Management
- Management of Web Services
- Management Using Web Services
- WS-Events
- WS-Eventing
- WS-Topics
- WS-BrokeredNotification
- WS-BaseNotification
- WS-Enumeration
- Web Services Management
- Web Services Management Foundation

Basic Profile, Basic, Reliability, Security, Messaging

Business Process Specifications

- Business Process Execution Language for Web Services
- Web Service Choreography Description Language
- Web Service Choreography Interface
- WS-Choreography Model Overview
- Business Process Management Language

Reliability, Security, Messaging, Transaction

Transaction Specifications

- WS-Business Activity
- WS-Atomic Transaction
- WS-Coordination
- WS-Composite Application Framework
- WS-Transaction Management
- WS-Context
- WS-Coordination Framework

Metadata, Messaging, Security

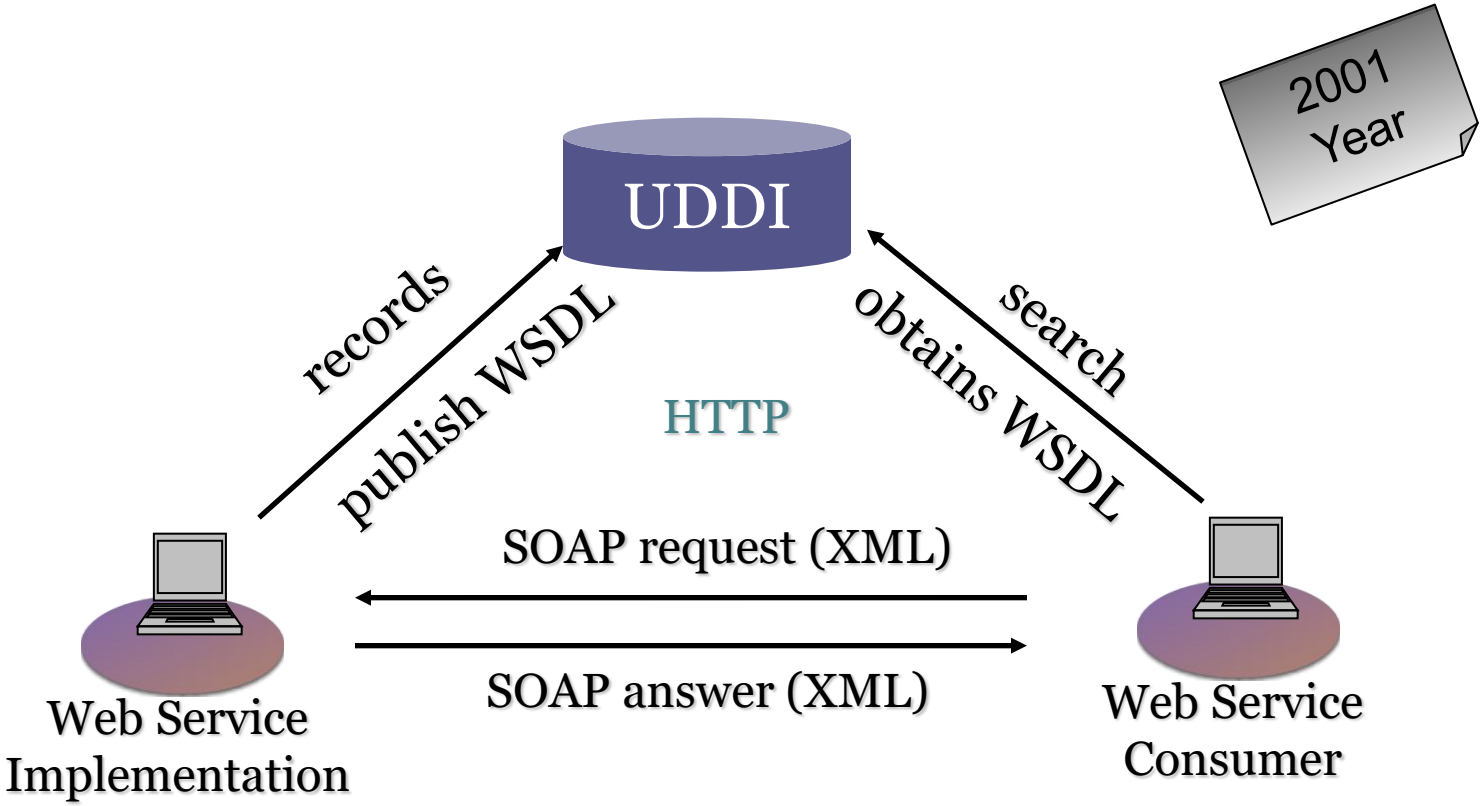
innoQ

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Telefon +49 (0) 2102 - 77 162 - 100
Telefax +49 (0) 2102 - 77 16 - 01
info@innoq.com · www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Telefon +41 (0) 41 - 743 01 11
Telefax +41 (0) 41 - 743 01 19

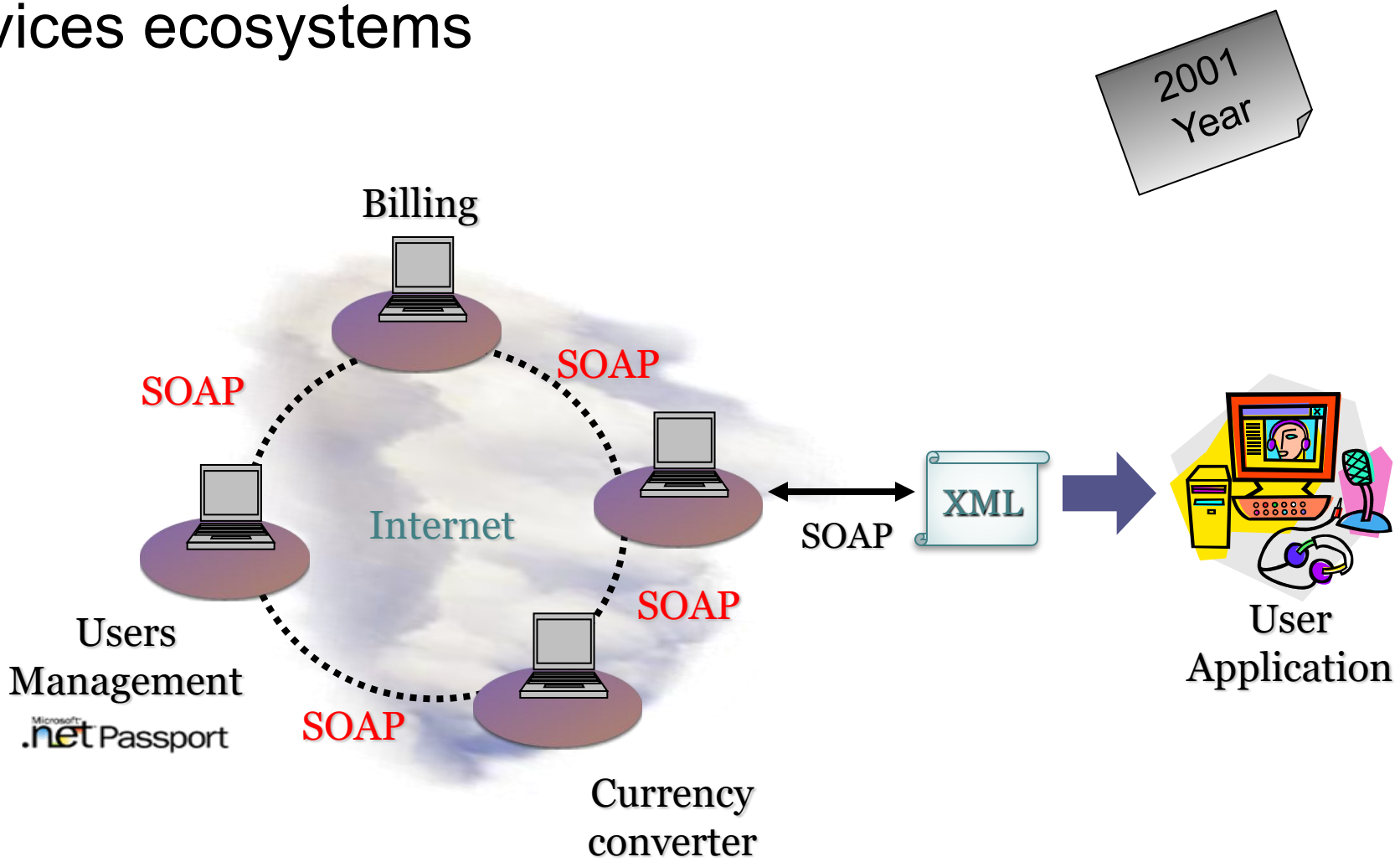
Version 2.0 · September 2005

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

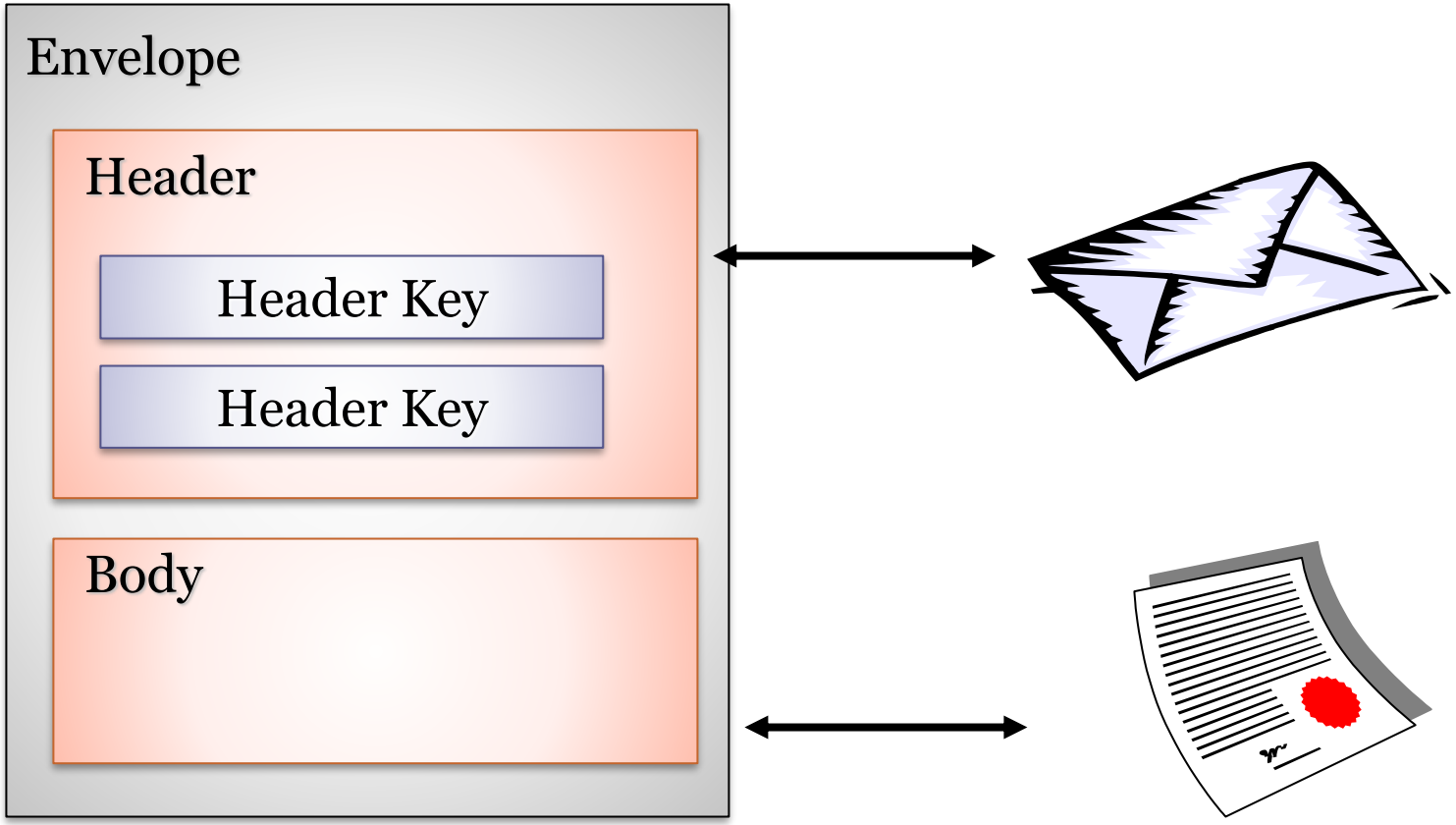
- Initially *Simple Object Access Protocol*

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 ?

```
POST /Suma/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: longitud 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>
```

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.

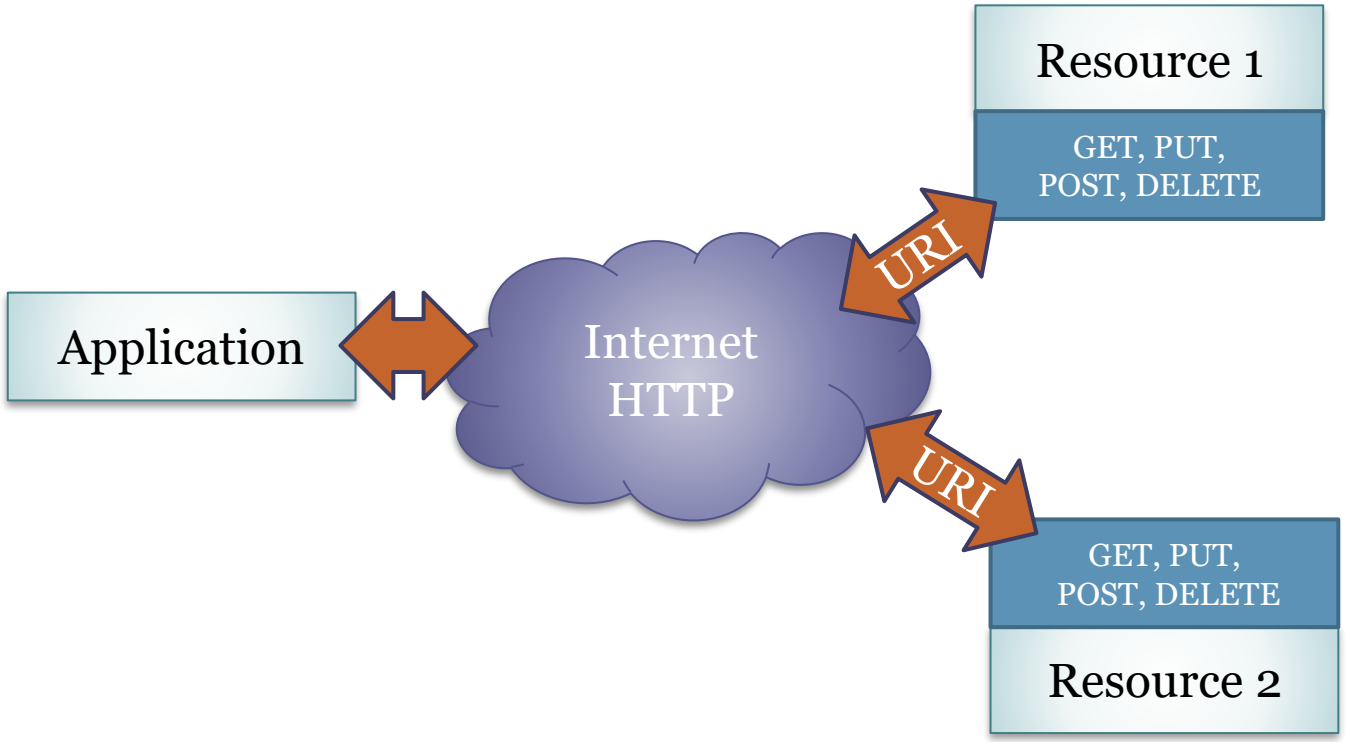
REST

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 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

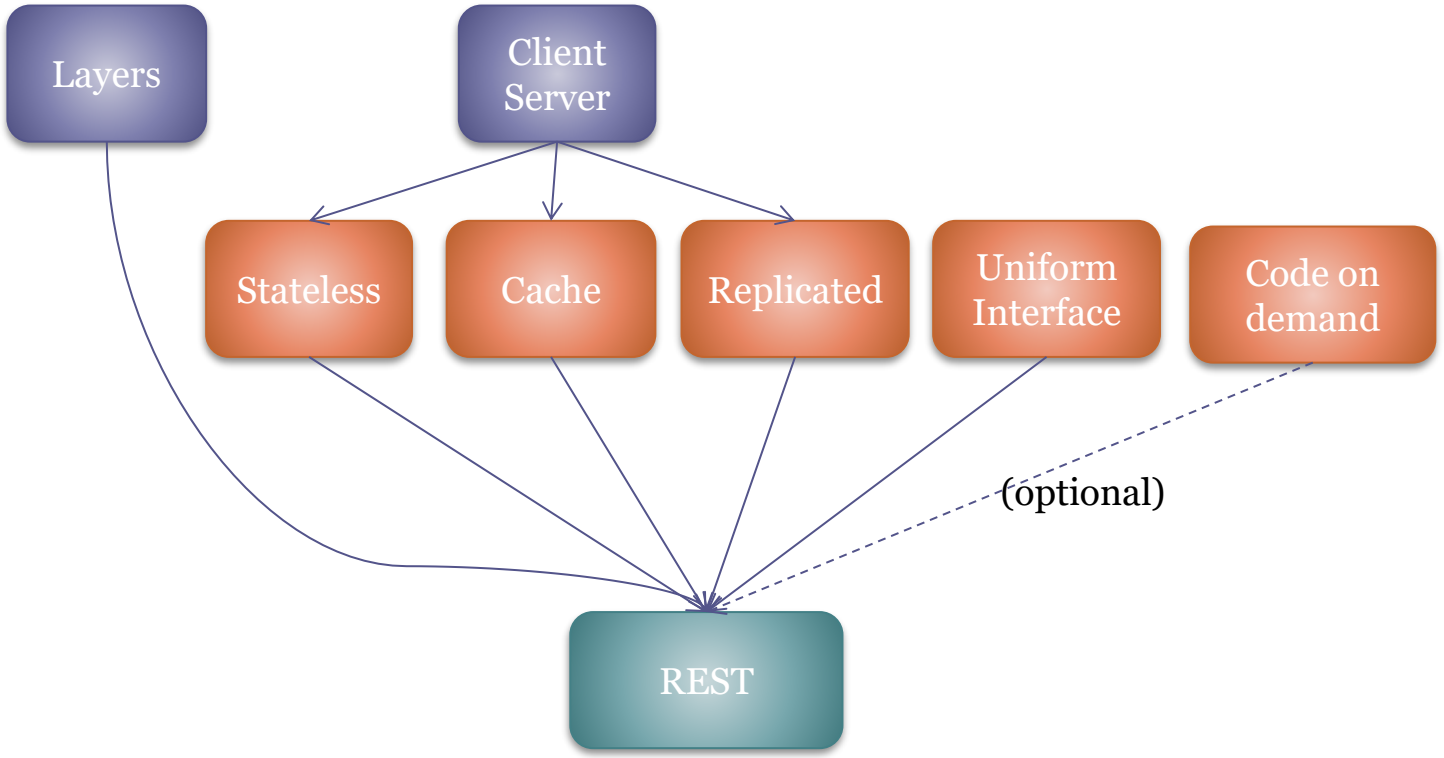
Challenges

- 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

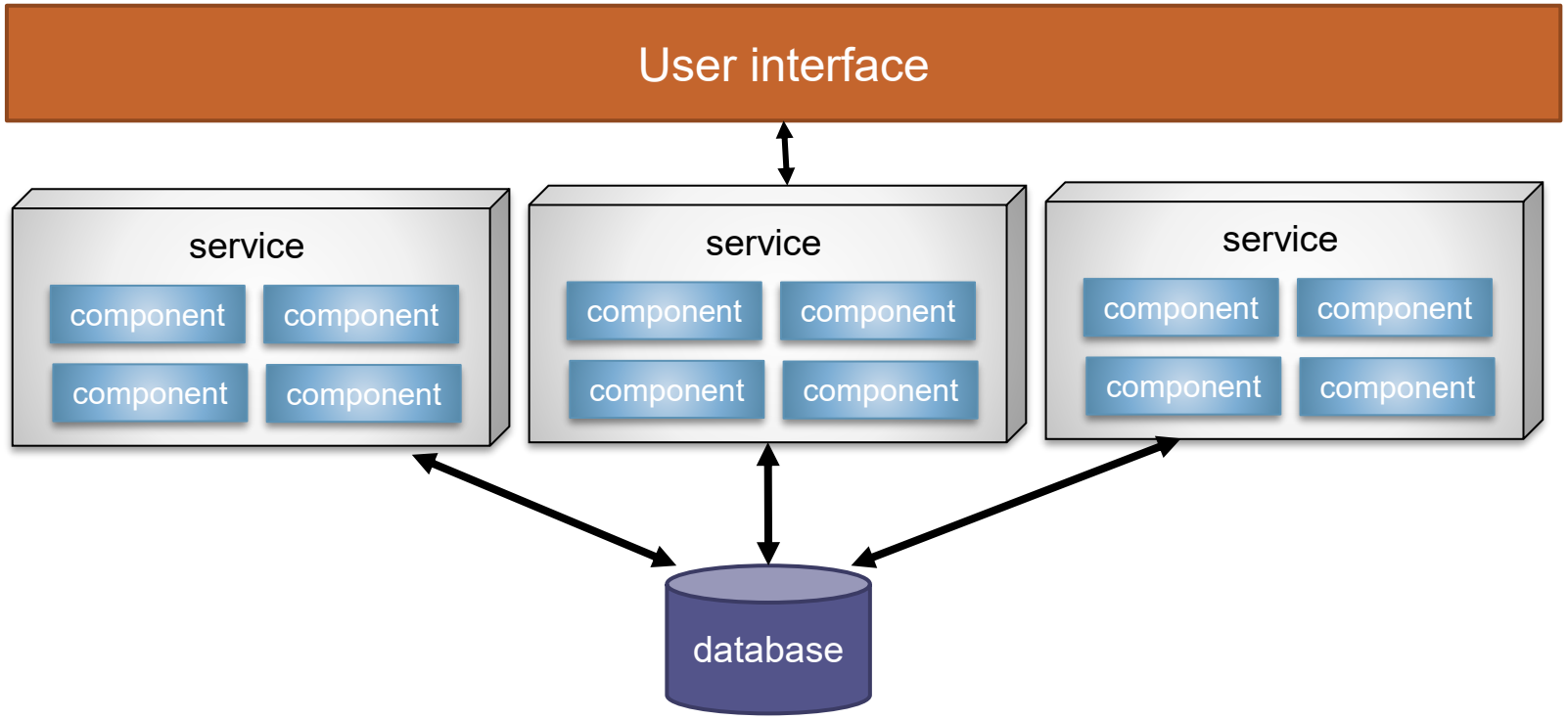
- Layers
- Client-Server
  - Stateless
  - Cached
  - Replicated server
- Uniform interface
  - Resource identifiers (URIs)
  - Auto-descriptive messages (MIME types)
  - Links to other resources (HATEOAS)
- Code on demand (optional)

REST as a composed style



Service based architecture

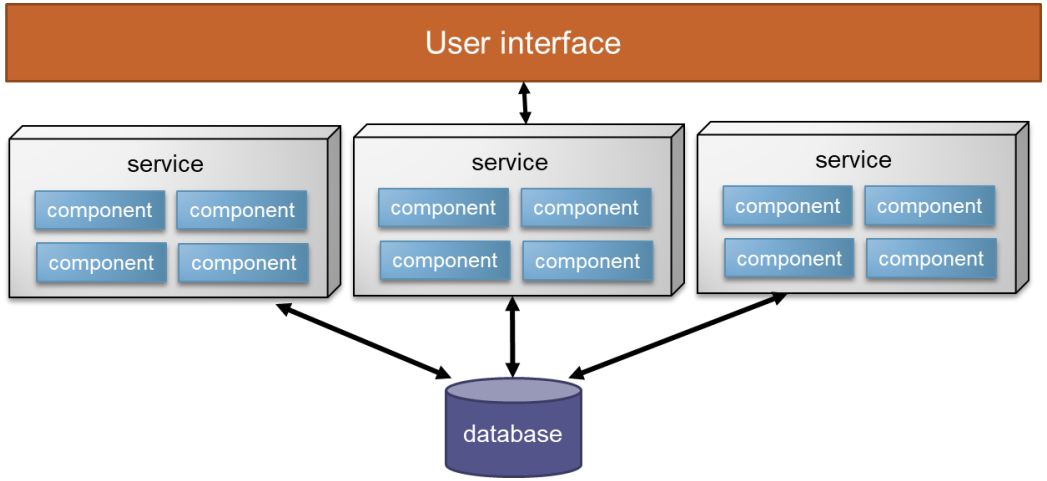
Pragmatic architectural style based on SOA



Service based architecture

Elements

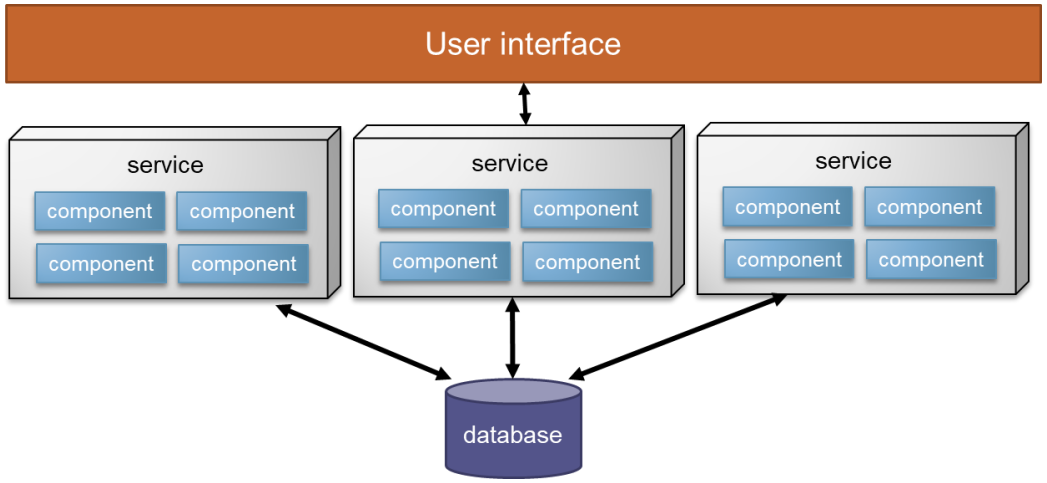
- Services = independently deployed units
  - Usually composed of different components
- User interface accesses services remotely (Internet)
- Database shared by those services



Service based architecture

Constraints

- Each service is independently deployed
- Services are usually coarse grained
- User interface can be divided (different topologies)
- Database is usually shared by each service



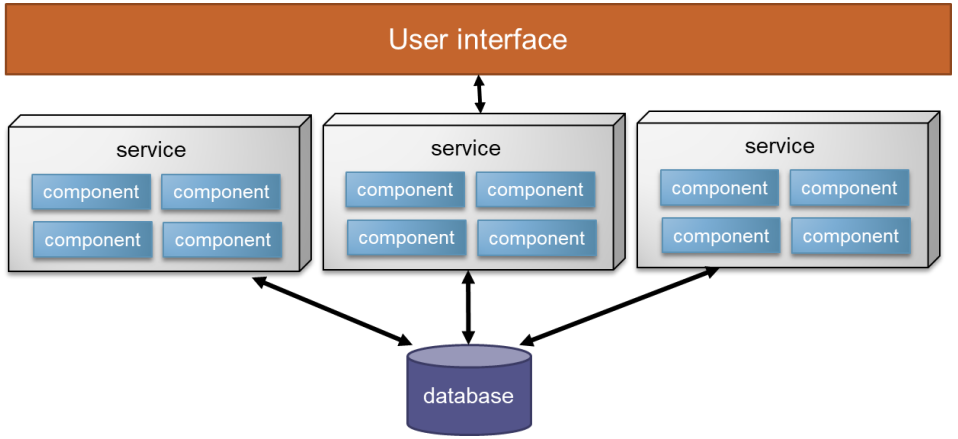
Service based architecture

Advantages

- Modularity of development
  - Services can be independently developed
- Technology diversity
  - Each service can be developed using a different programming language & technology
- Time to market
  - Several frameworks
- Availability
- Reliability

Challenges

- Scalability (database partitioning)
- Evolution of services
  - Adaption to change is usually difficult
  - Services can be monoliths
- Conway's law
  - Database team
  - User interface team
  - Programmers



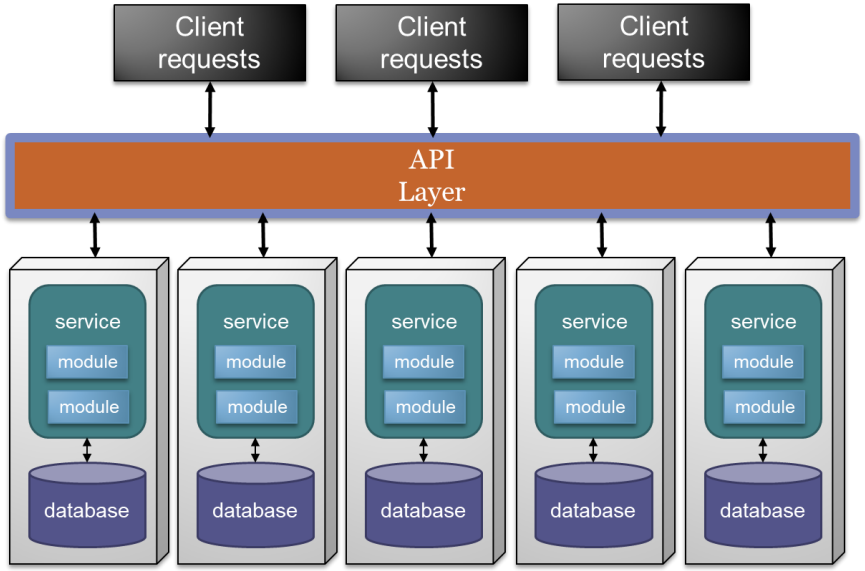
Microservices

Applications decomposed in microservices

Microservice = small, autonomous services that work together

Each microservice = independent building and deployment block

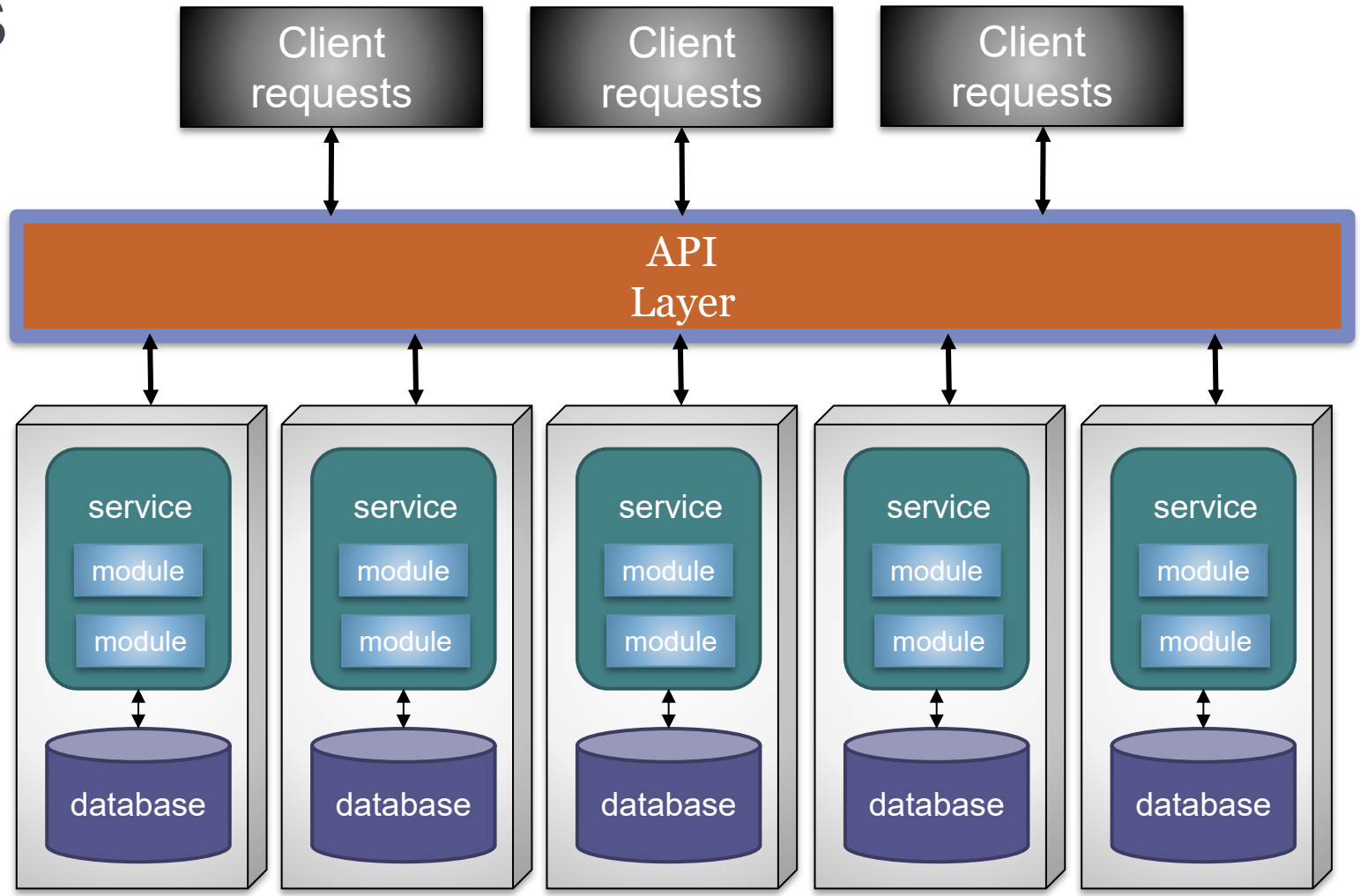
- Highly uncoupled
- Focus on a specific task
- Manage their own data



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

Microservices

Diagram



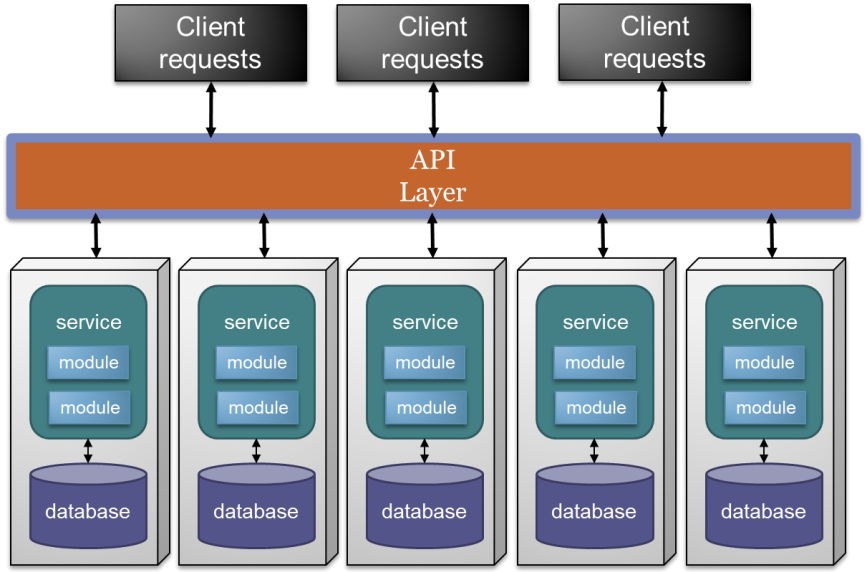
Microservices

Elements

A service + database form a deployed component

A service contains several modules and its own database

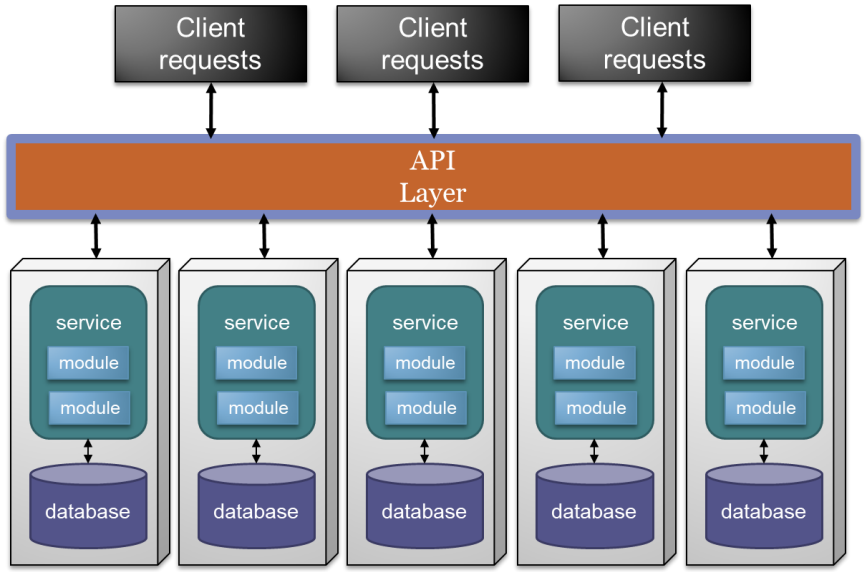
API layer (optional) offers a proxy or naming service



Microservices

Constraints

- Distributed
- Bounded context:
  - Each service models a domain or workflow
- Data isolation
- Independency:
  - No mediator or orchestrator



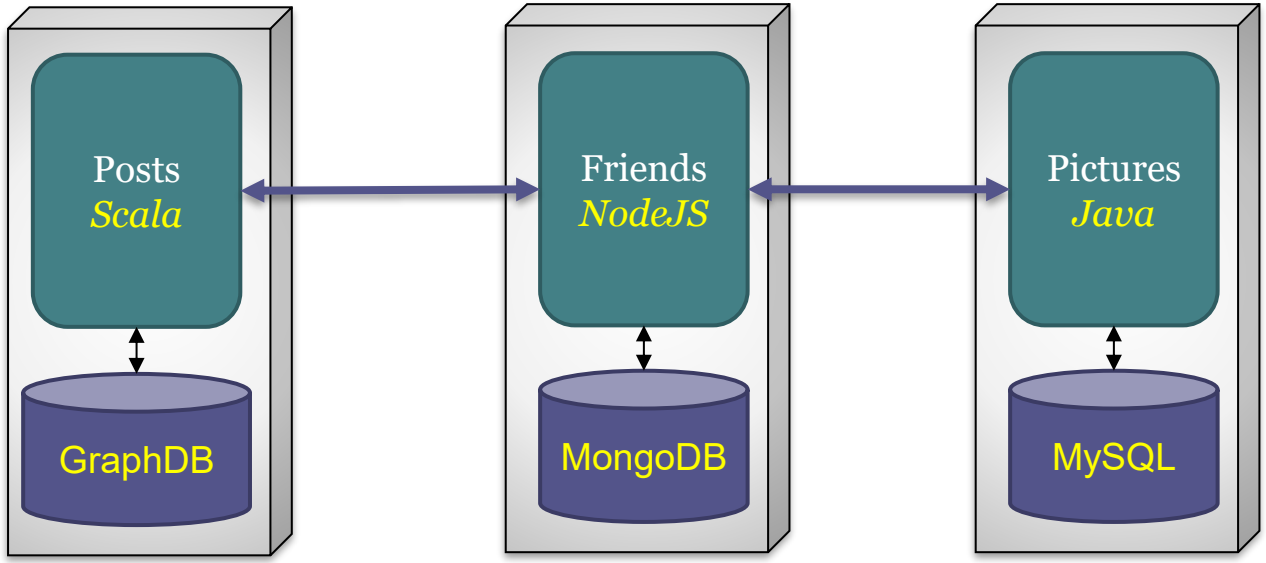
Features/advantages

Technology heterogeneity
Resilience
Scalability
Deployability
Organizational alignment
Decentralized data management
Optimizing for replaceability

Technology heterogeneity

Each microservice can be implemented in its own programming language and technology stack

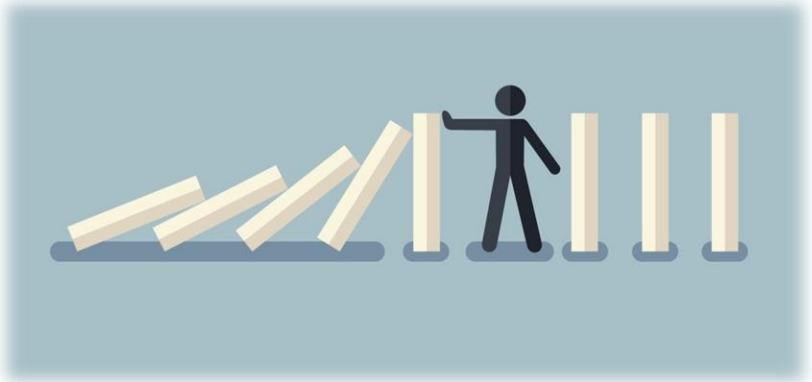
- Facilitates experimentation with new technologies
- Flexibility



Resilience

If a component of a system fails and the failure doesn't scale, the system can carry on working

In a monolithic system if a component fails, the whole system stops working



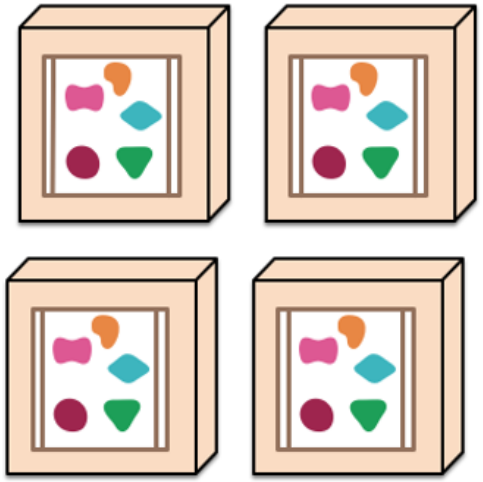
Scalability

- It is possible to scale on demand specific services
  - Monolithic systems require to scale the whole system
  - Not all components have the same needs
    - Microservices can be replicated as needed

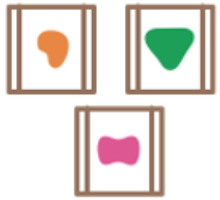
Scalability

Monolithic: all functionality in a single process

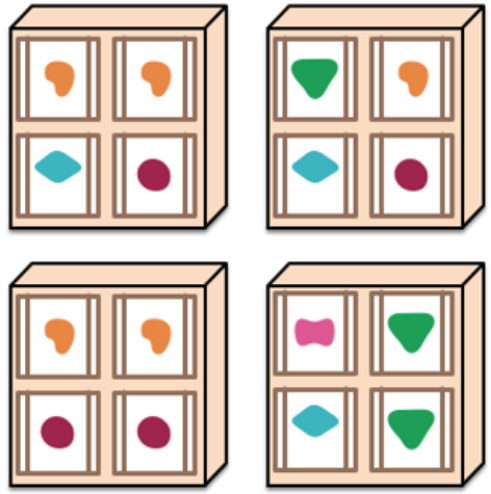
...scales replicating the monolith on multiple services



Microservices: each element of functionality into a separate service



... scales distributing these services and replicating as needed



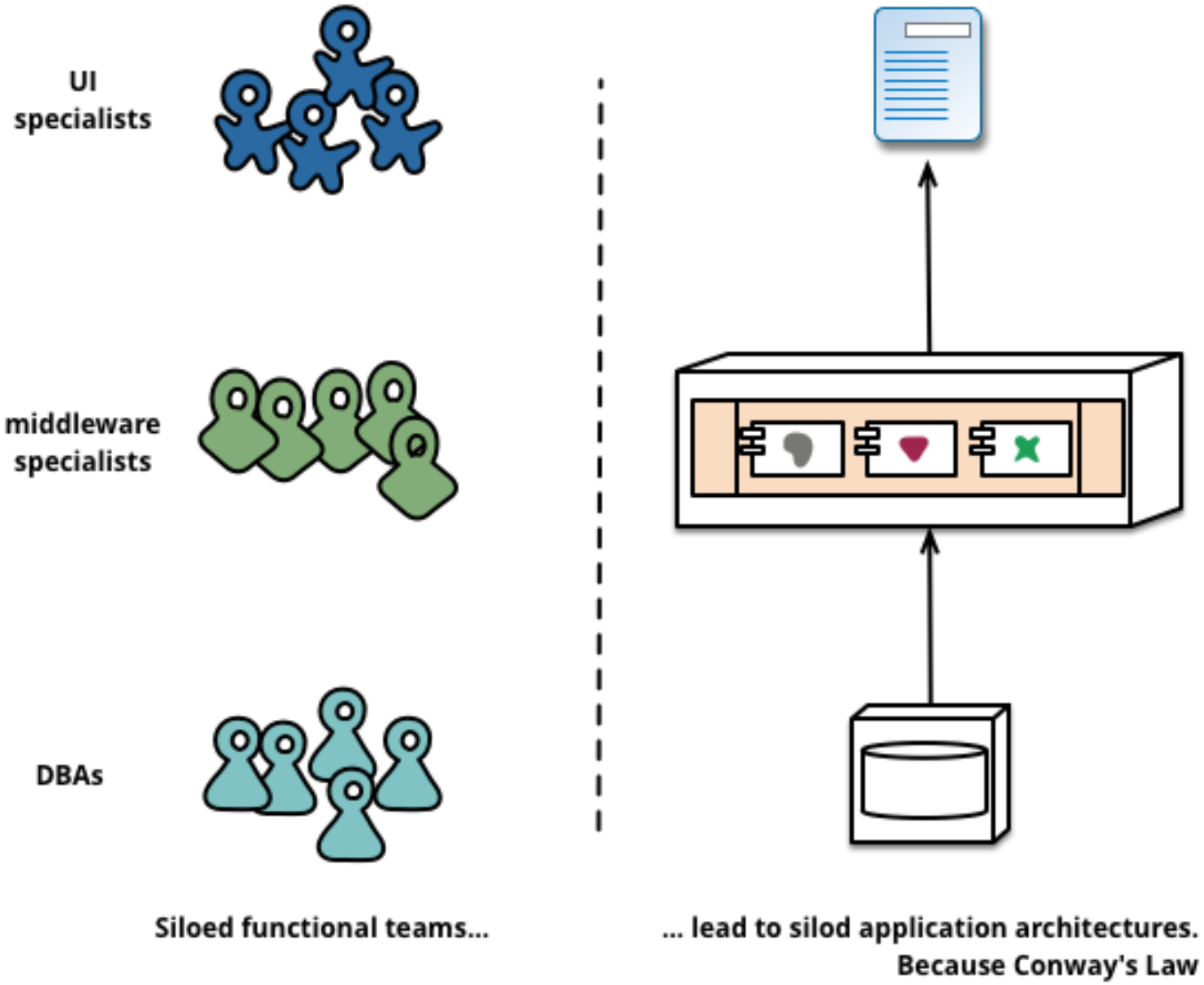
Deployability

- Deploy each service independently
  - Enables to do a change in a service and deploy it inmediately
  - Towards continuous deployment

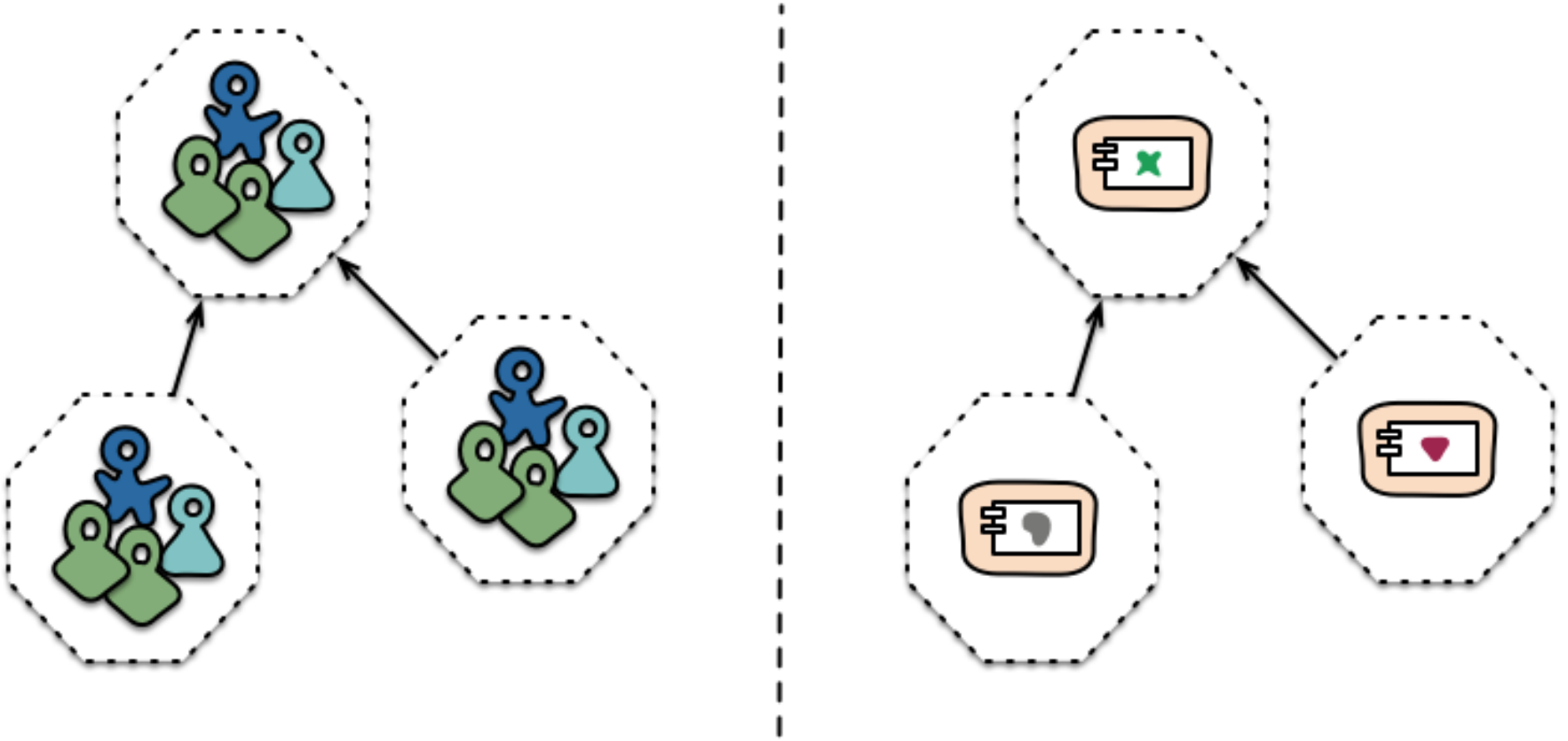
Organizational alignment

- Inverse Conway Law maneuver
  - Evolve teams and organizational structure to promote the desired architecture
- Create teams following the modular decomposition
  - Cross-functional teams
- Service ownership: the team owning a service is responsible for making changes and deploying it
  - "You build it, you run it" (Amazon)
- Goal: increased autonomy and speed of delivery

Traditional applications



With microservices



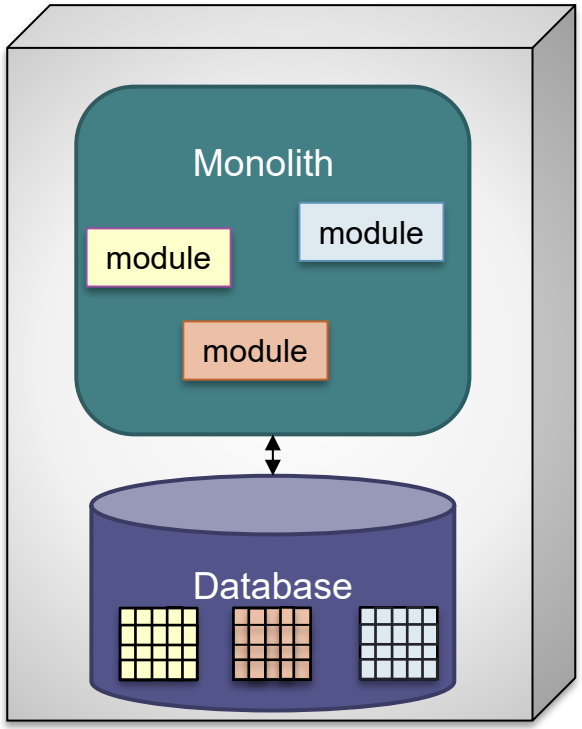
Cross-functional teams...

... organised around capabilities
Because Conway's Law

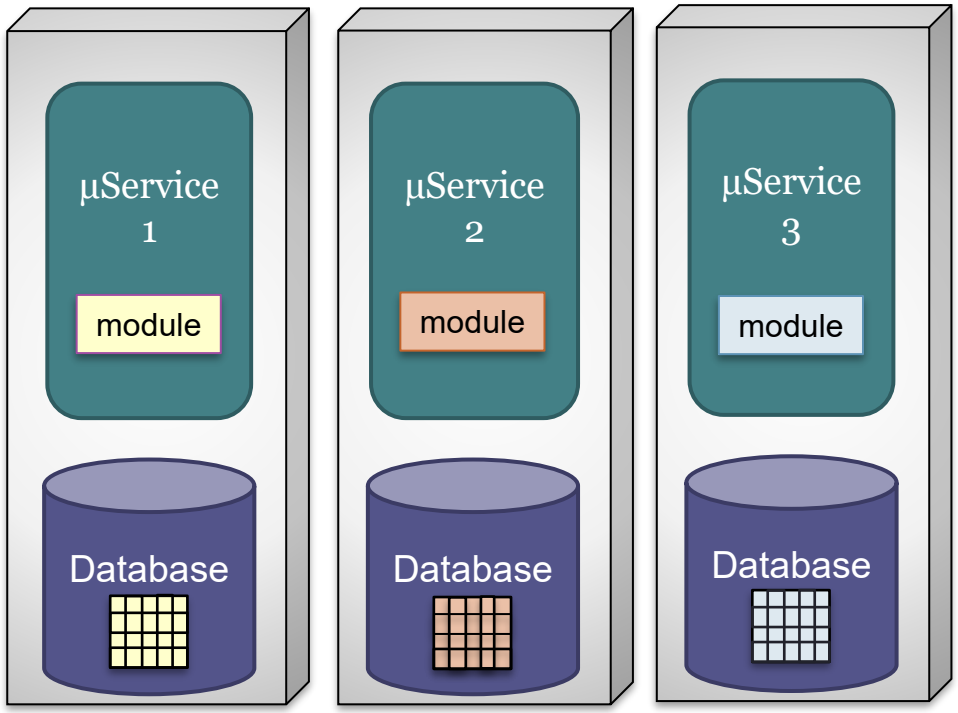
Decentralized data management

Each team/service handles its own data

Monolith - single database



Microservices - application databases



Optimizing for replaceability

- Traditional systems usually contained old legacy systems which no one wants to touch
- With microservices
  - Less cost to replace a microservice with a better implementation
  - Or even delete it

Challenges of microservices

- Managing lots of microservices
  - Too much microservices = antipattern (nanoservices)
  - Ensure application consistency
- Complexity of distributed system management
  - New challenges: latency, message format, load balance, fault tolerance, etc.
- Testing & deployment
  - Operational complexity
- Antipattern: distributed monolith
  - Microservices tangled that are not independently deployed
- Structural decay (*see next slide*)

http://martinfowler.com/articles/microservice-trade-offs.html
https://www.ufried.com/blog/microservices_fallacy_1/

Microservices structural decay

- Code dependencies between services
  - Too much shared libraries
- Too much inter-service communication
- Too many orchestration requests
- Database coupling

Analyzing architecture (microservices)
https://www.youtube.com/watch?v=U7s7Hb6GZCU

Some microservices patterns

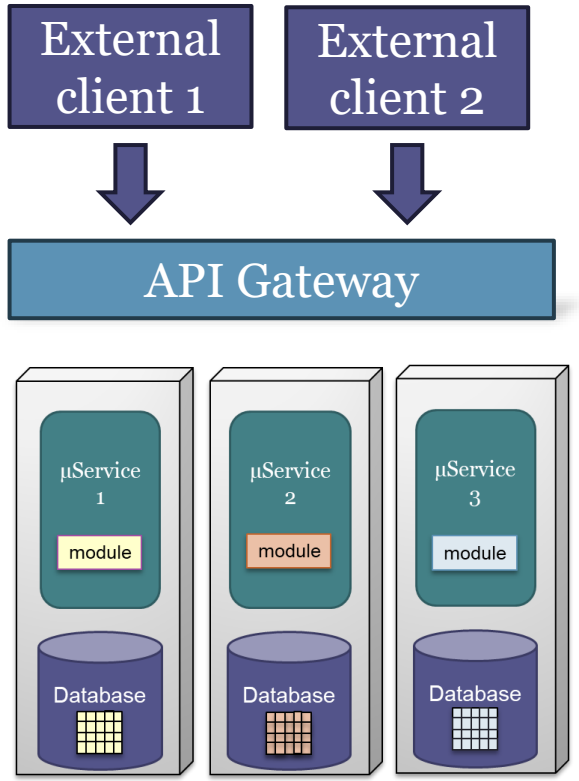
- API Gateway: External Access
  - GraphQL
- Sidecar: Deployment
- Strangler Fig: Migration
- Access token: Security

More info: https://microservices.io/

API Gateway

Offer a single point of entry
- A client shouldn't have to know about a lot of different endpoints

Consequences
- Centralized entry point
  - Map a single public URL to internal microservices
- Protocol translation
  - Can translate REST/GraphQL requests to internal gRPC calls
- Security
  - Handle OAuth/JWT so internal microservices don't need to handle it
- Request aggregation
  - One call to the external API can be translated to several microservice calls
- Rate limiting
  - Protecting internal microservices from spikes of traffic or DDoS attacks



GraphQL

- Also known as: Query based integration
  - Initially developed by Meta
- Client specifies required data as a query
  - Which fields needs in a single request
    - Goal: eliminate:
      - Over-fetching (obtaining more data than necessary)
      - Under-fetching (needing multiple calls to get related data)
  - Single smart endpoint
    - Single URL with resolvers to pull data from different microservices
  - Strongly typed schema
    - Schema Definition Language defines data models
  - More info: https://graphql.org/

Sidecar

Offload cross-cutting tasks from the main application into a separate container

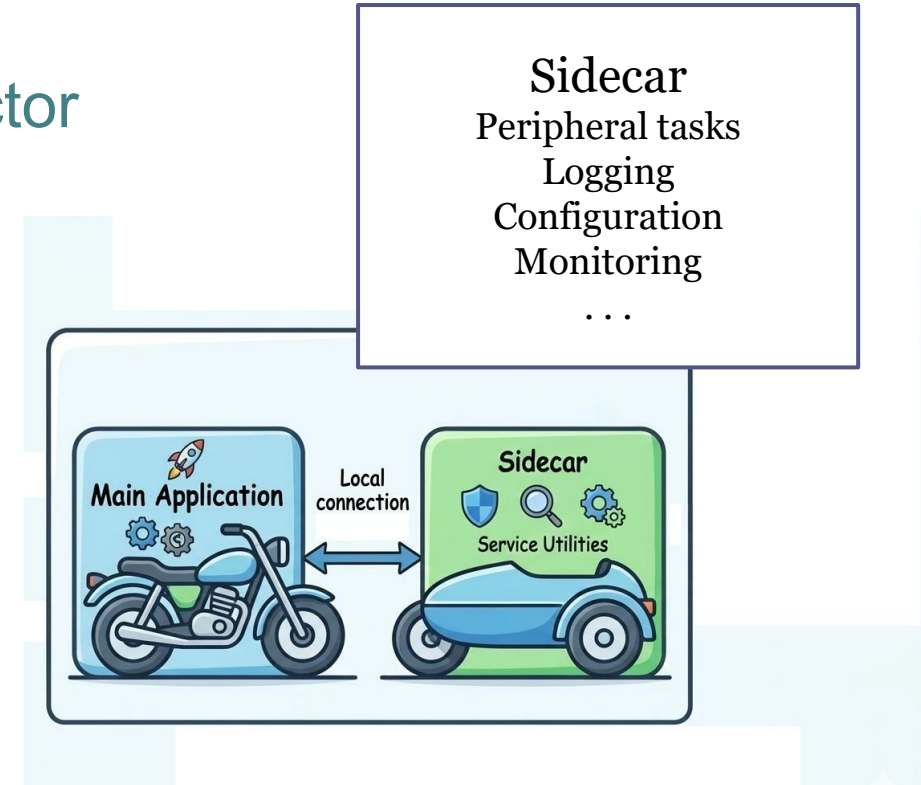
- Cross-cutting tasks = logging, configuration, monitoring, …
- The sidecar can be written in a different language or technology
- It starts and stops with the main application

Some use cases

- Monitor one application
- Collect logs and stream them to a central collector
- Healthchecking and restarting

Consequences

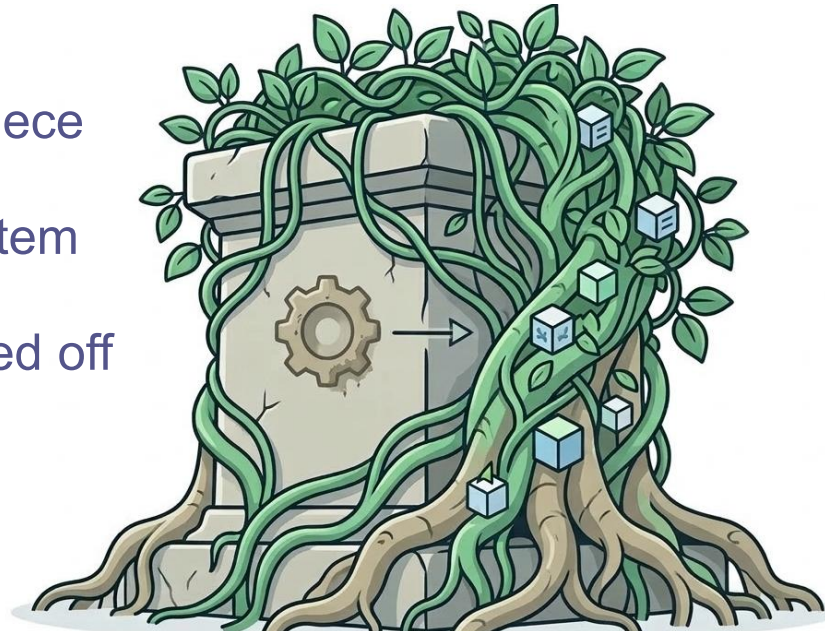
- Separation of concerns
- Consistent interface for platform services



Strangler Fig

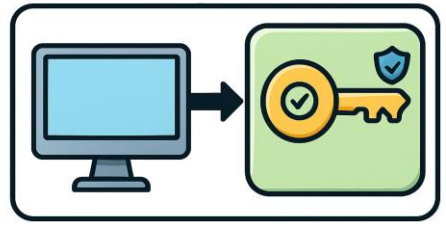
Pattern for migration from a monolith to microservices
Main idea: Incremental migration instead of "big bang" rewrite
Steps:

- Legacy preservation:
  - Wrap existing monolith in a facade
- Routing layer
  - Send requests to legacy system or new microservices
- Coexistence
  - Both systems run in production simultaneously
- Gradual replacement
  - Over time, functionality is strangled out of the monolith piece by piece
- Risk mitigation
  - If a newly service fails, it is easy to route traffic back to legacy system
- Decommisioning
  - Once the monolith is an empty Shell with no traffic it is safely turned off



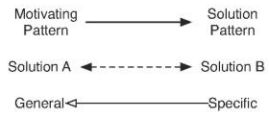
More info: https://martinfowler.com/bliki/StranglerFigApplication.html

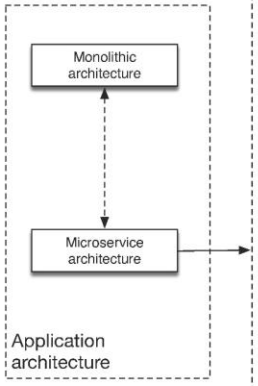
Access Token

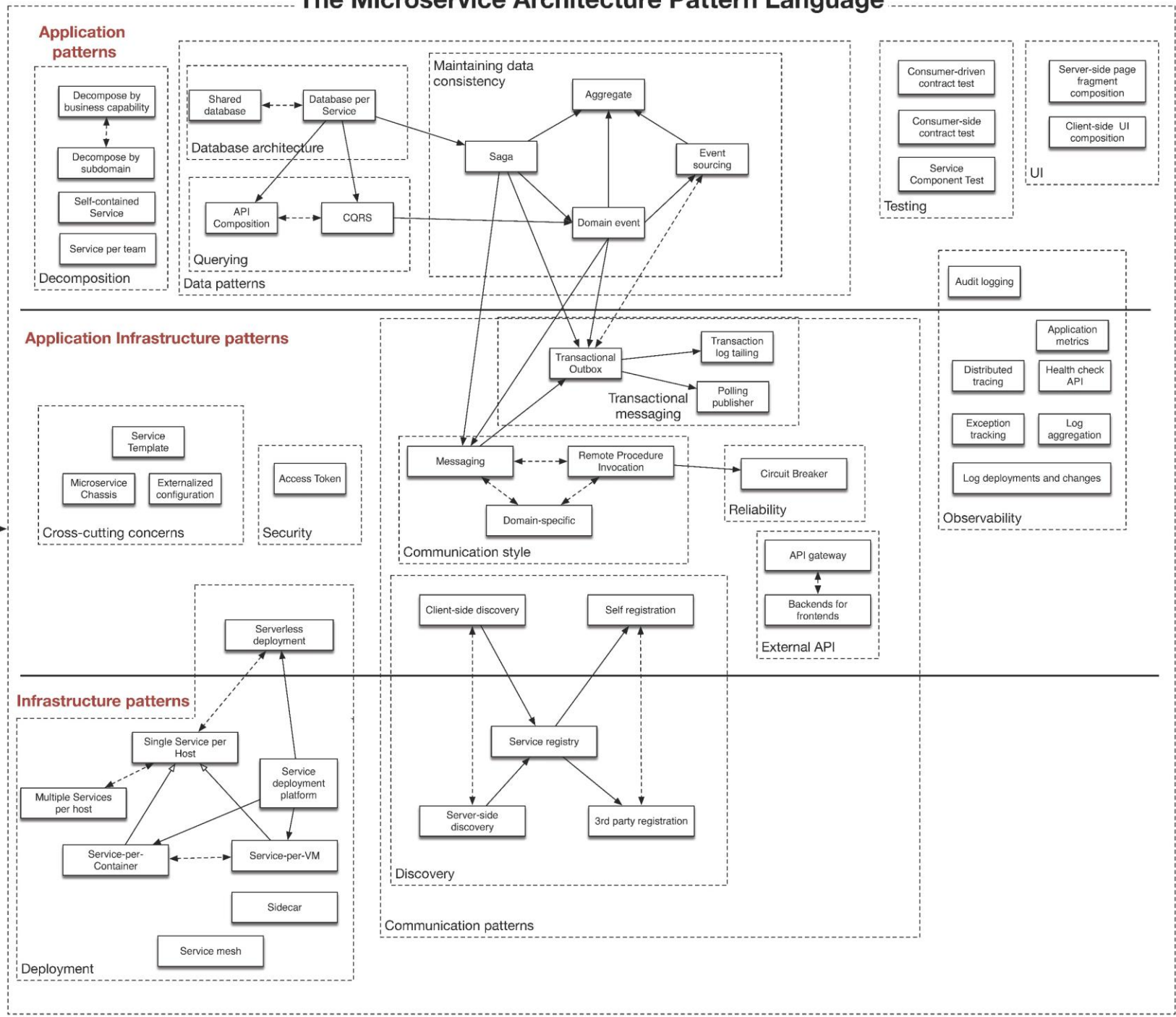


- The clients carry a self-contained *password* or token
  - It avoids using user or session ids
- The token proves the identity in each microservice
- Self-described claims
  - The token contains claims (User Ids, roles, expiration…)
- Each microservice can verify the token's authenticity locally
  - Stateless validation
- Decoupled architecture
  - Microservices don't need to know how a user logged in
  - They only care if the token is valid

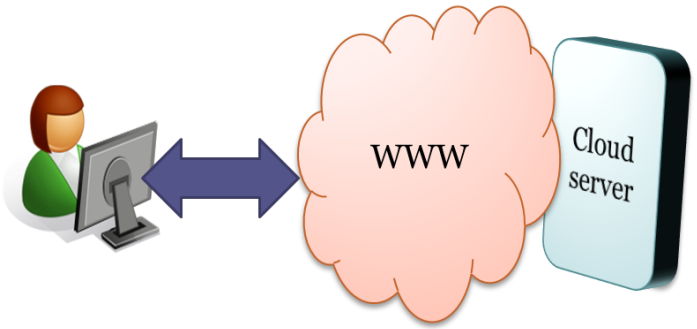
Other Microservices patterns







Serverless



Also known as:
- Function as a service (FaaS)
- Backend as a service (BaaS)

Applications depend on third-party services

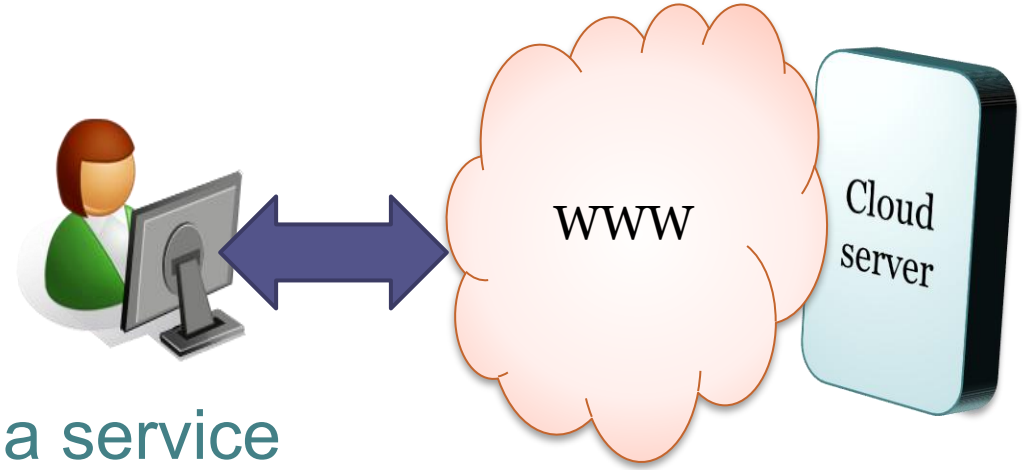
Developers don't need to care about servers
- Automatic scalability

Rich clients
- *Single Page Applications, Mobile apps*

Examples:
- AWS Lambda, Google Cloud Functions/Firebase, Ms Azure Functions

https://en.wikipedia.org/wiki/Serverless_computing
https://martinfowler.com/articles/serverless.html

Serverless



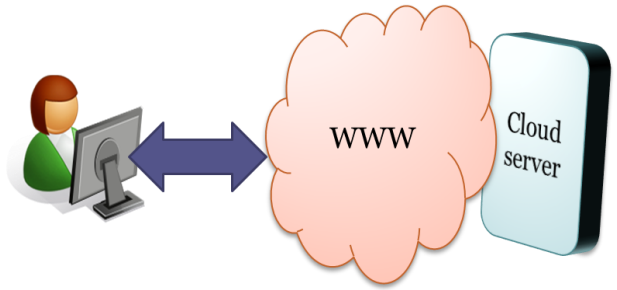
Elements

- Client that runs functions as a services
- Cloud server which provides backend as a service

Constraints

- No management of server hosts
- Ephimeral lifecycle: stateless and short-lived functions
- Automatic scalability and provisioning based on load
- Costs based on usage
- Operational abstraction
  - Developers focus on business-logic
  - Server-management tasks shifted to the provider
    - OS updates, cluster sizing, HW provisioning

Serverless



Advantages

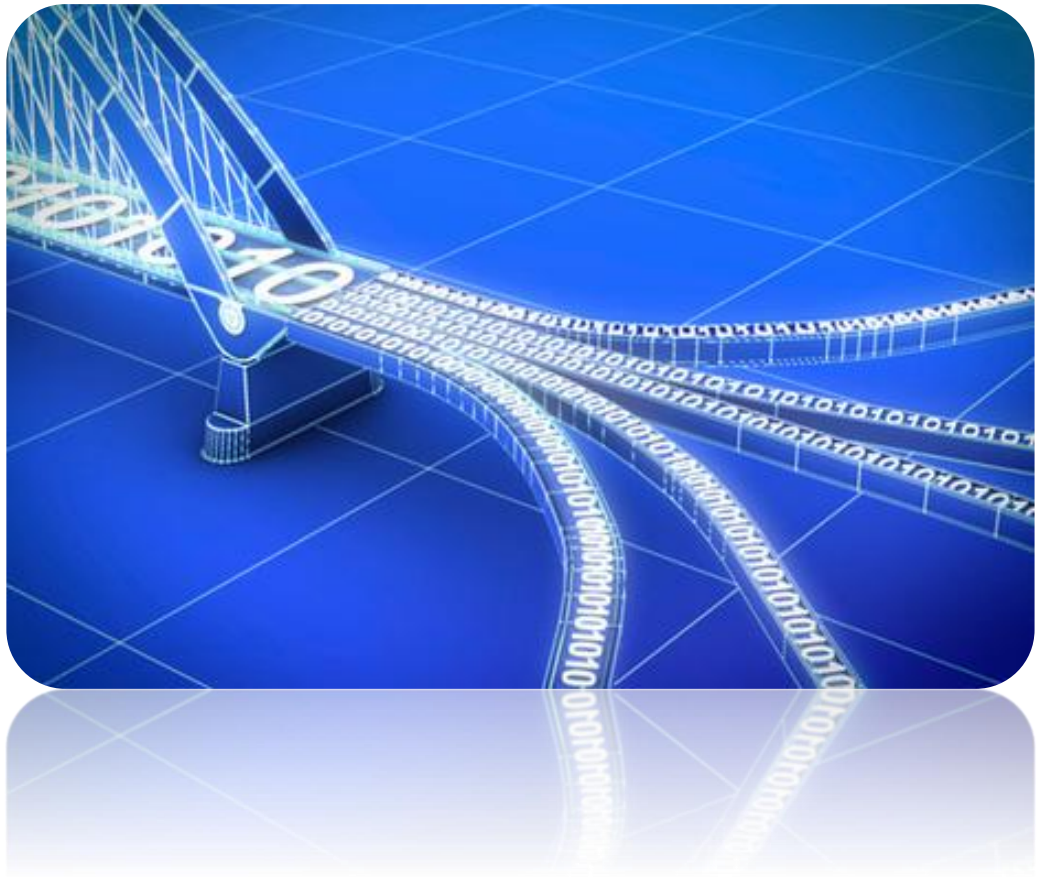
- Automatic scalability
- Implicit high availability
- Performance not defined in terms of host size/cost
- Costs based on precise usage
  - Only pay for the compute you need
  - Ideal for unpredictable/spiky workloads
- Time to market

Challenges

- Vendor lock-in
  - Incompatibility between vendors
- Resource constraints
  - Limits on execution time/available memory
- Startup latency (Cold start)
- Integration testing
- Monitoring/debugging

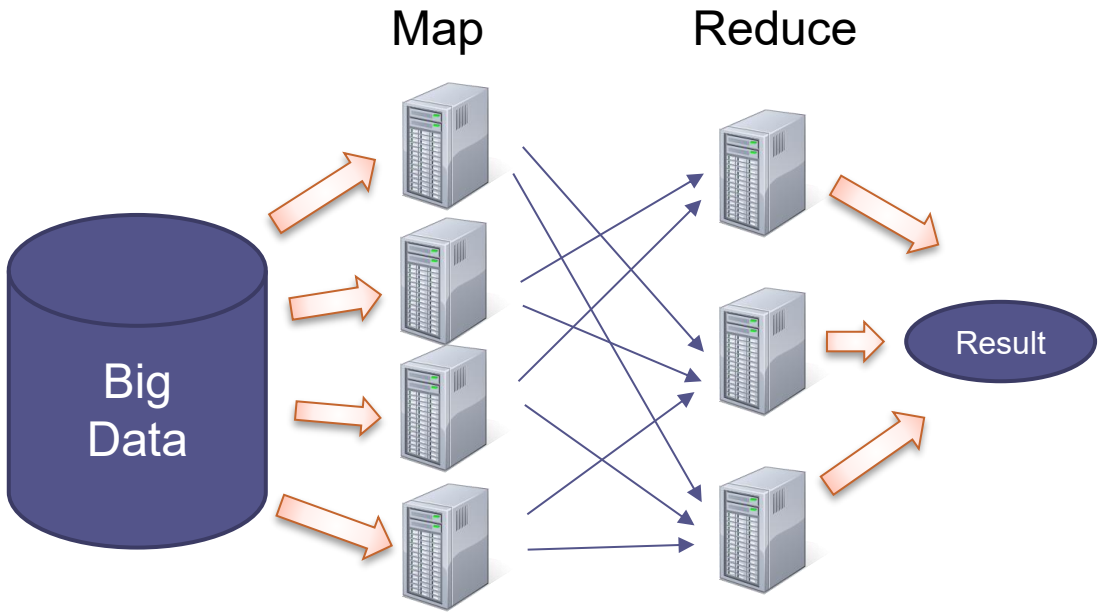
Big data and scalable systems

MapReduce
Lambda architecture
Kappa architecture



MapReduce

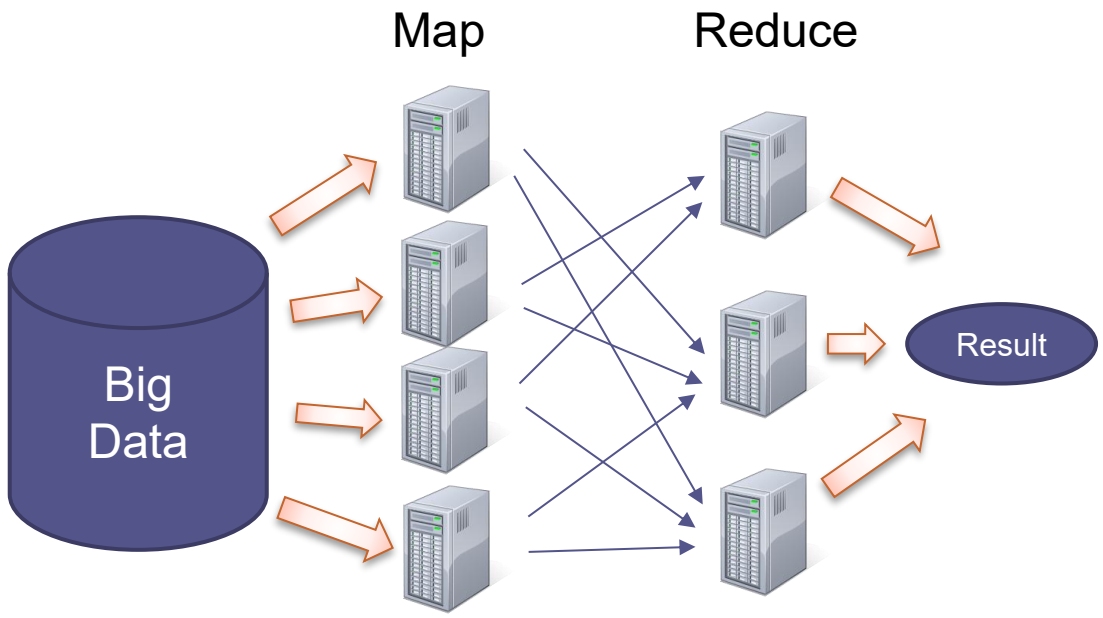
- Proposed by Google
  - Published in 2004
  - Internal implementation by Google
- Goal: big amounts of data
  - Lots of computational nodes
  - Fault tolerance
  - Write-once, read-many
- Style composed of:
  - Master-slave
  - Batch



MapReduce

Elements

- *Master* node: Controls execution
  - Node table
  - It manages replicated file system
- *Slave nodes*
  - Execute mappers, reducers
  - Contain replicated data blocks



MapReduce - Scheme

Inspired by functional programming

- 2 components: mapper and reducer

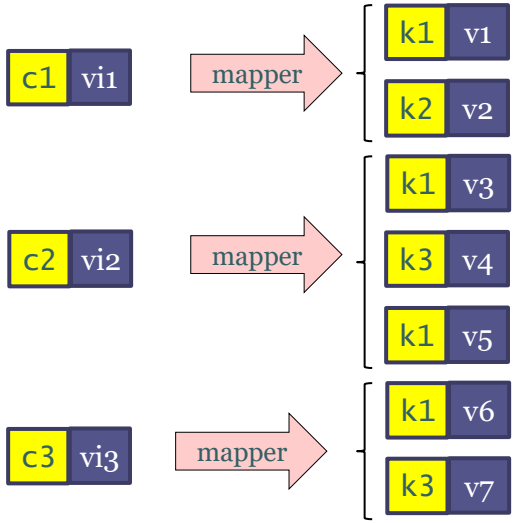
Data are divided for their processing

Each data is associated with a key

- Transforms `[(key1,value1)] to [(key2,value2)]`

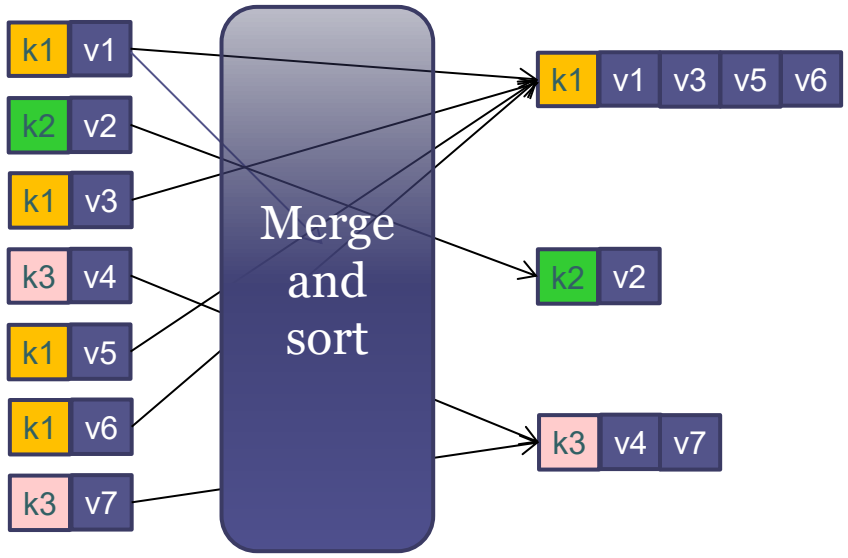
Step 1: mapper

mapper: (Key1, Value1) → [(Key2,Value2)]



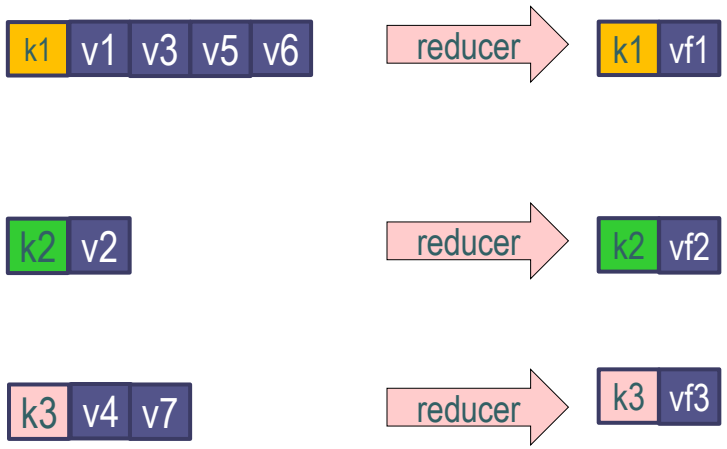
Step 2: Merge and sort

System merges and sorts intermediate results according to the keys

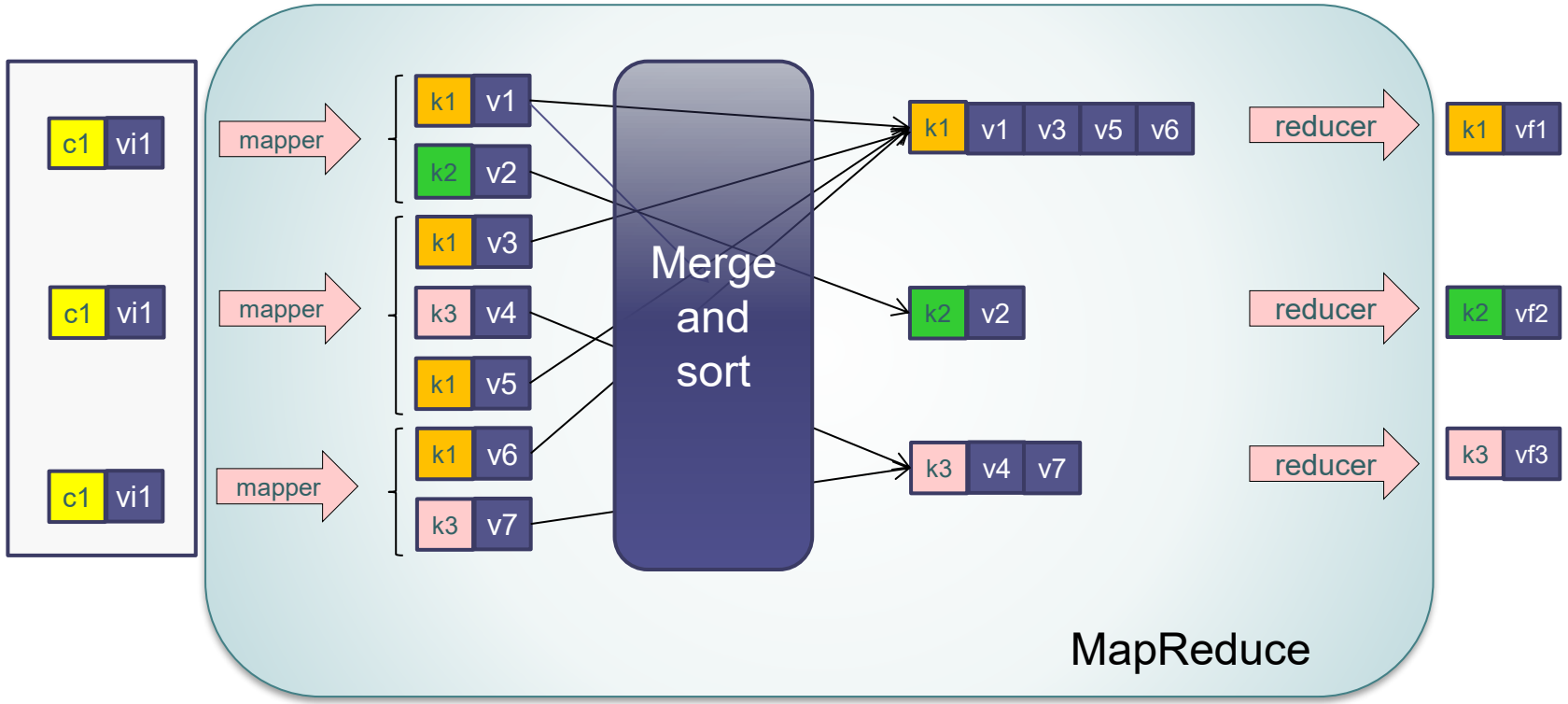


Step 3: Reducers

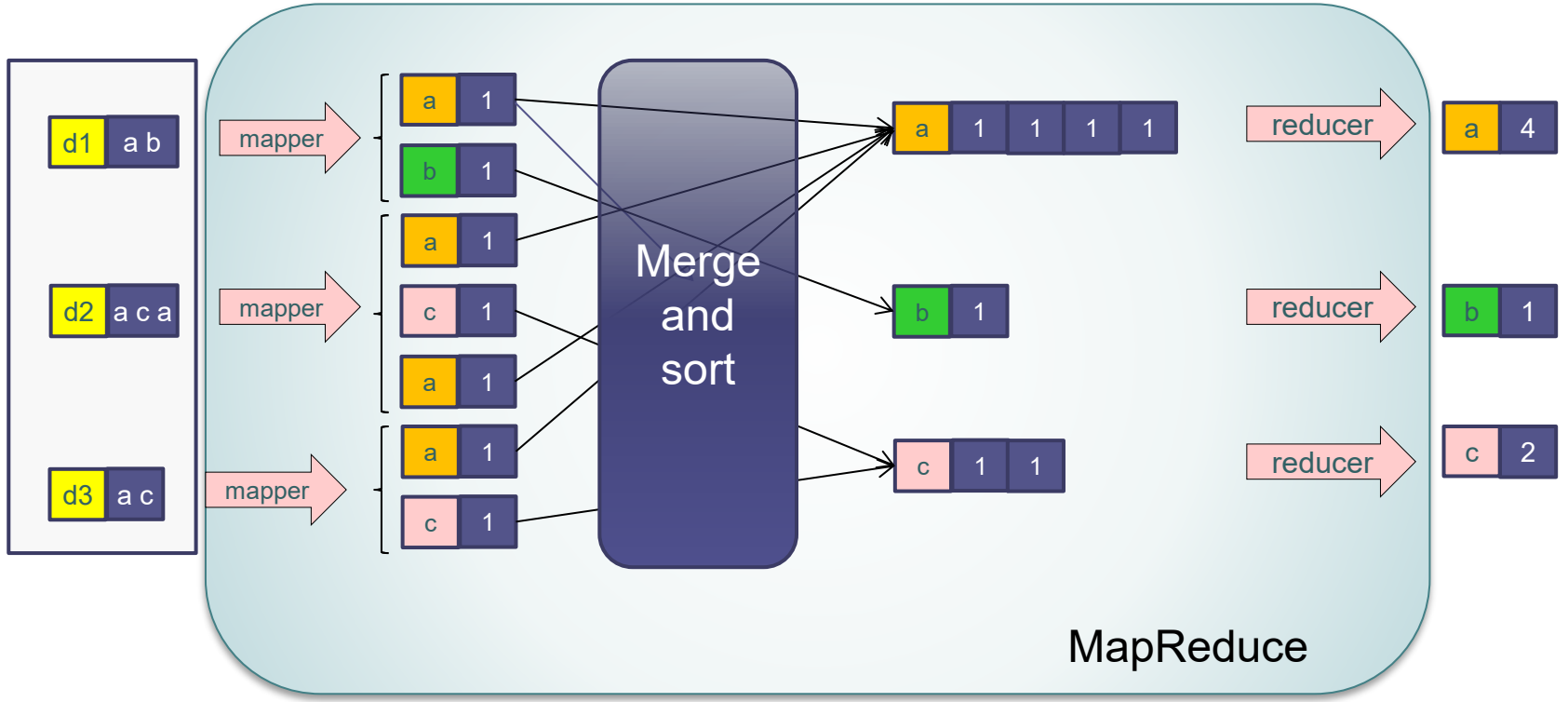
reducer: (Key2, [Value2]) → (Key2,Value2)



MapReduce - general scheme



MapReduce - count words




```
// return each work with 1
mapper(d,ps) {
 for each p in ps:
   emit (p, 1)
}
```

```
// sum the list of numbers of each word
reducer(p,ns) {
  sum = 0
  for each n in ns { sum += n; }
  emit (p, sum)
}
```

MapReduce - execution environment

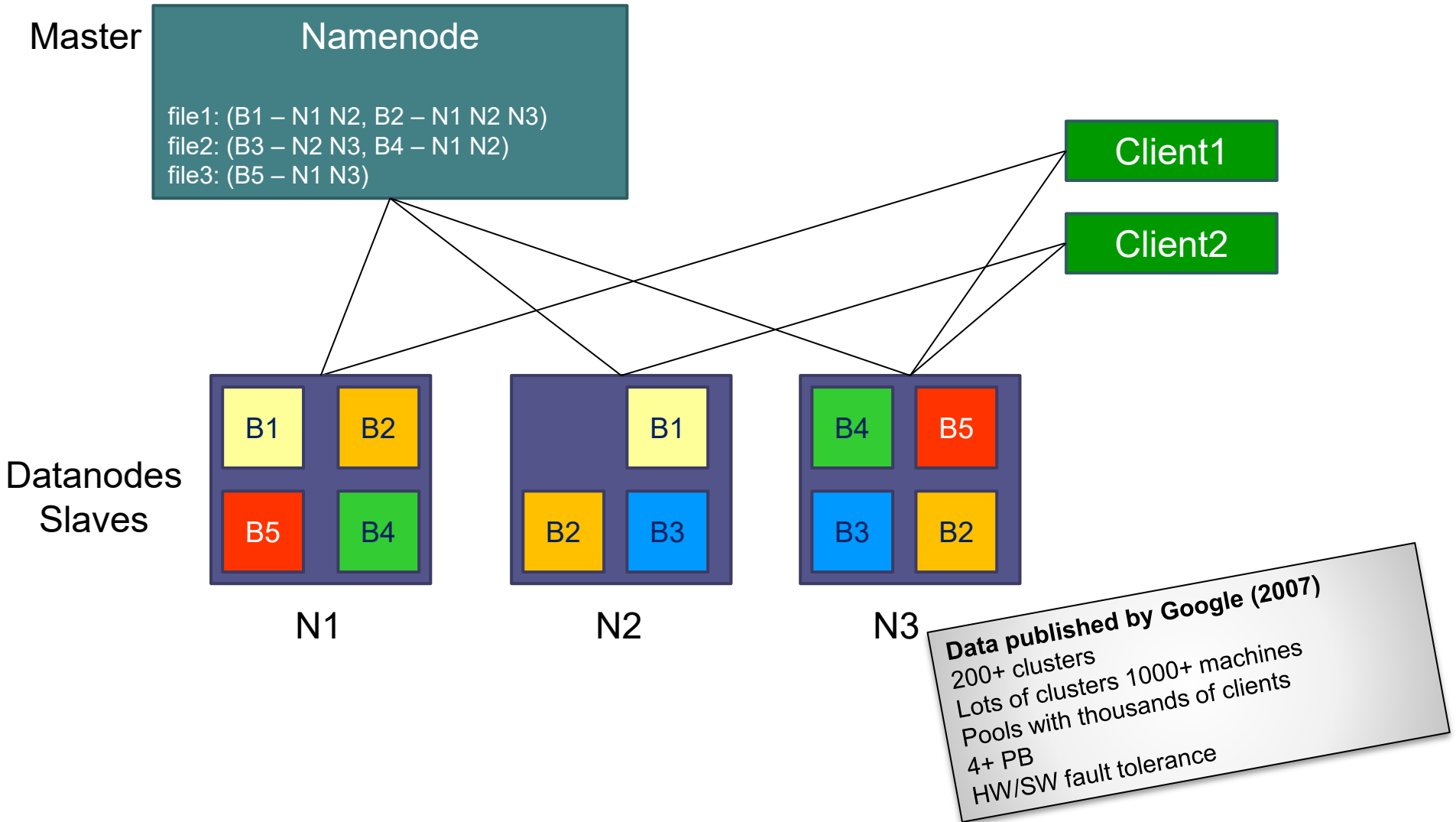
Execution environment is in charge of:

- Planning: Each job is divided in tasks
- Placement of data/code
  - Each node contains its data locally
- Synchronization:
  - *reduce* tasks must wait *map* phase
- Error and failure handling
  - High tolerance to computational nodes failures

MapReduce - File system

- Google developed a distributed file system - GFS
- Hadoop created HDFS
  - Files are divided in chunks
  - 2 node types:
    - Namenode (master), datanodes (data servers)
  - Datanodes store different chunks
    - Block replication
  - *Namenode* contains metadata
    - Where is each chunk
    - Direct communication between clients and datanodes

MapReduce - File system



MapReduce

Advantages

- Distributed computations
  - Split input data
    - Replicated repository
  - Fault tolerant
    - Hardware/software heterogeneous
- Large amount of data
  - Write-once. Read-many

Challenges

- Dependency on master node
- Non interactivity
- Data conversion to MapReduce
  - Adapt input data
  - Convert output data

MapReduce: Applications

Lots of applications:

- Google, 2007, 20petabytes/day, around 100,000 mapreduce jobs/day
- PageRank algorithm can be implemented as MapReduce
- Success stories:
  - Automatic translation, similarity, sorting, ...
- Other companies: last.fm, facebook, Yahoo!, twitter, etc.

MapReduce: Applications

- Implementations
  - Google (internal)
  - Hadoop (*open source)*
  - …
- Libraries
  - Hive (Hadoop): query language inspired by SQL
  - Pig (Hadoop): specific language that can define data flows
  - Cascading: API that can specify distributed data flows
  - Flume Java (Google)
  - Dryad (Microsoft)

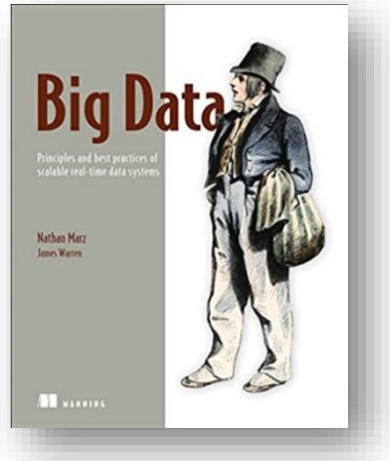
Lambda architecture

Handle Big Data & real time analytics

Proposed by Nathan Marz, 2011

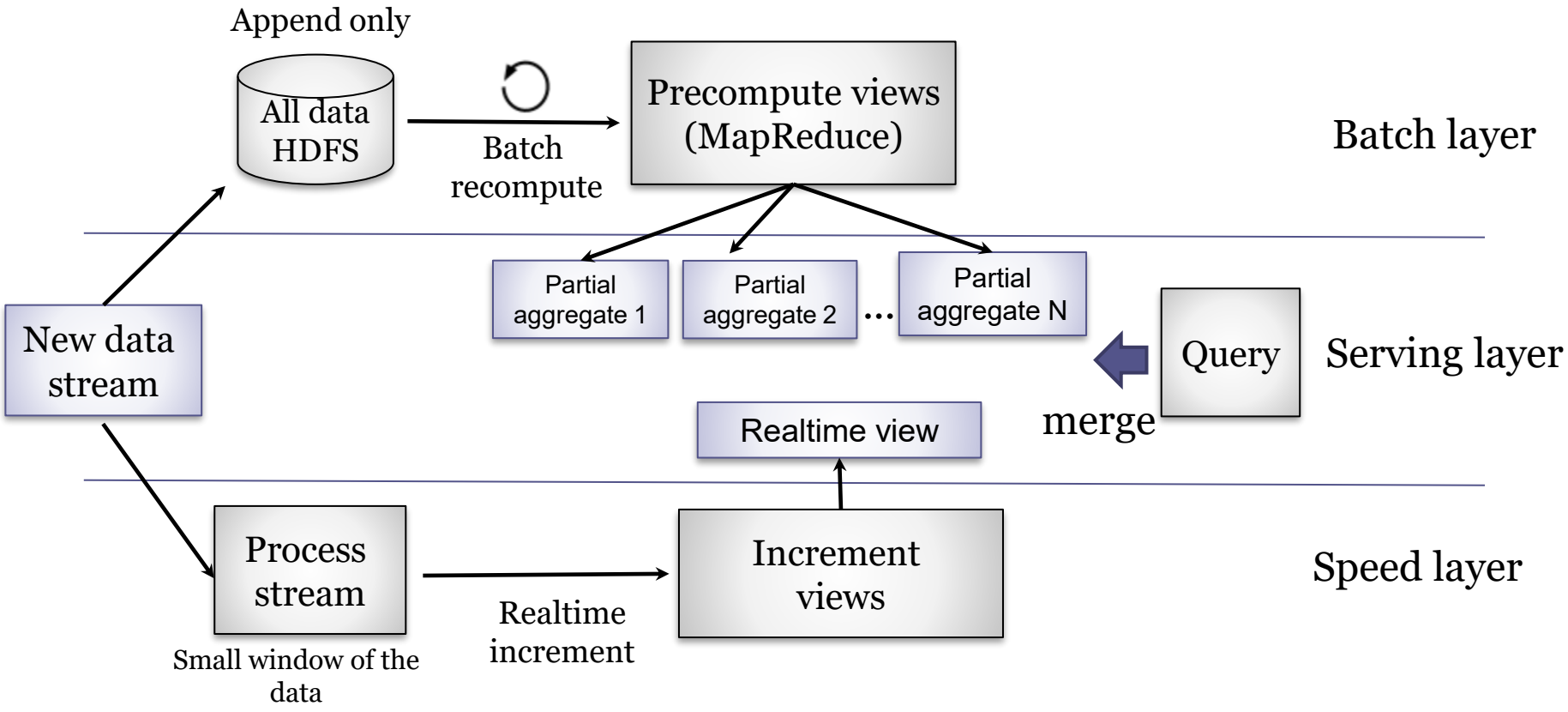
3 layers

- Batch layer: precomputes all data with MapReduce
  - Generates partial aggregate views
  - Recomputes from all data
- Speed layer: real time, small window of data
  - Generates fast real time views
- Serving layer: handles queries
  - Merges the different views

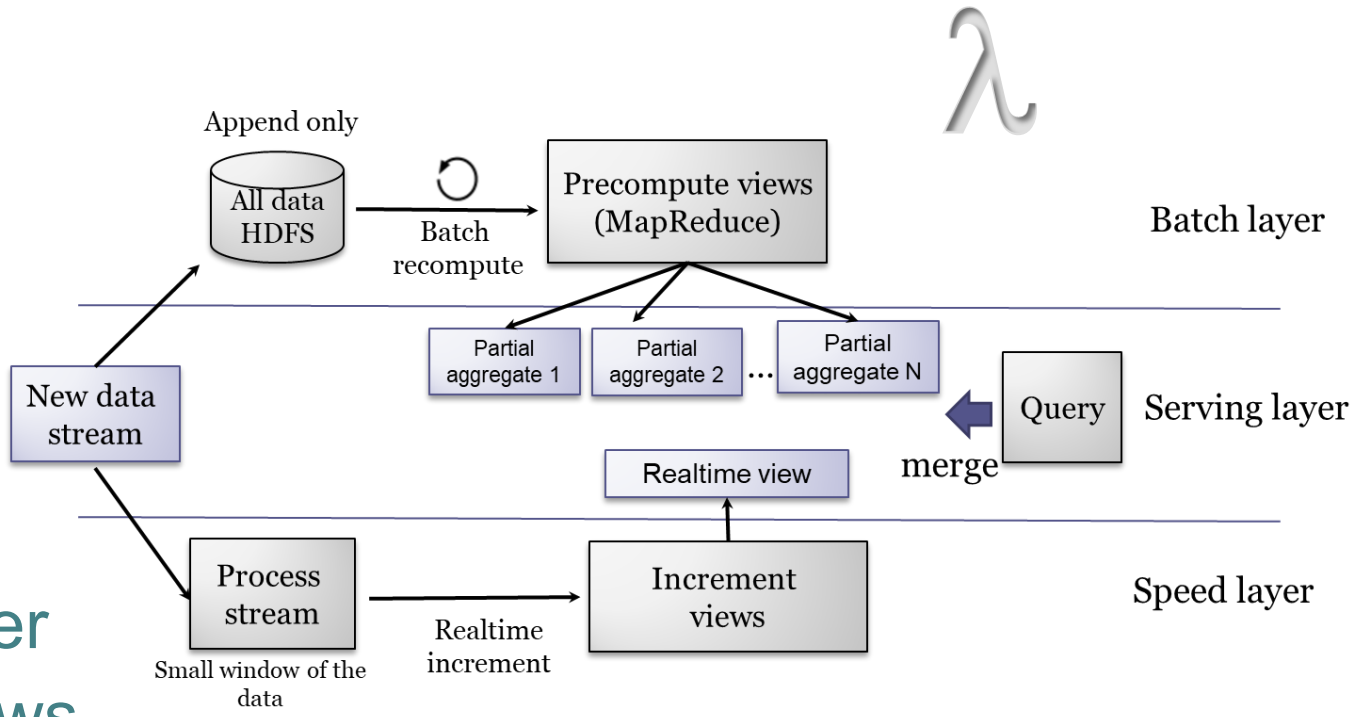


Lambda architecture

Combines Real time with batch processing



Lambda architecture



Constraints

- All data is stored in the batch layer
- The batch layer precomputes views
- The results of the speed layer may not be accurate
- Serving layer combines precomputed views
- The views can be simple DBs for querying

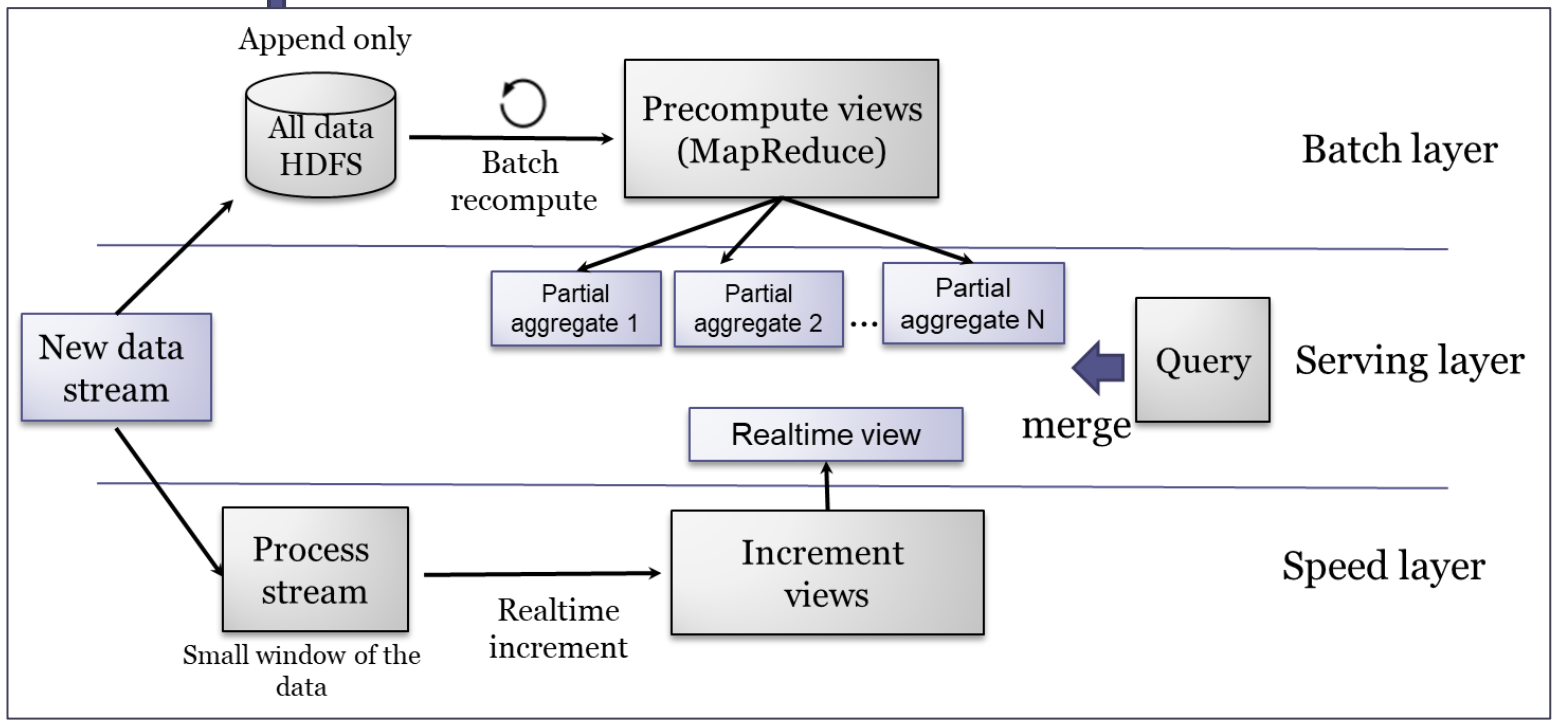
Lambda architecture

Advantages

- Scalability (Big data)
- Real time
- Decoupling
- Fault tolerant
- Keep all input data
- Reprocessing

Challenges

- Inherent complexity
- Merging views can be inaccurate
  - Losing some events



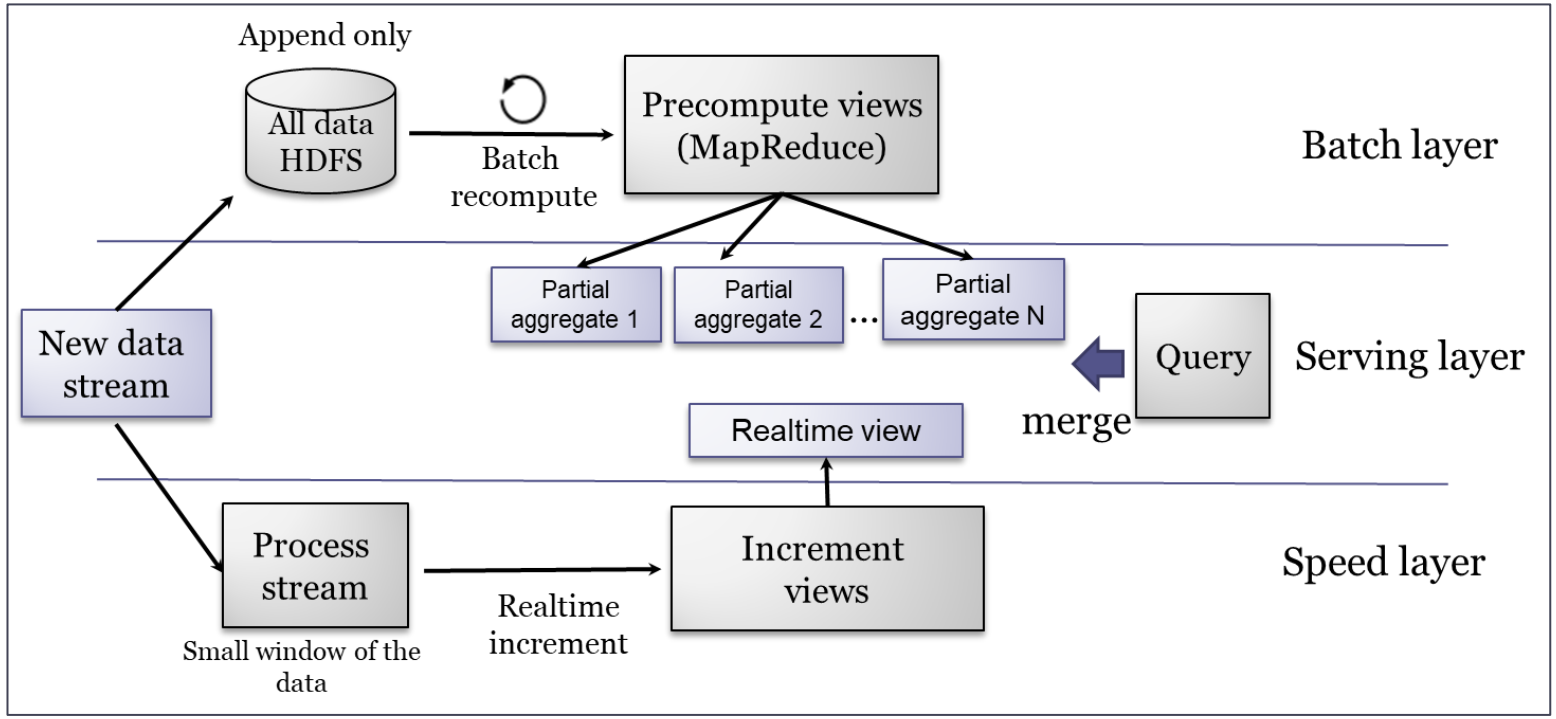
Lambda architecture

Applications

- Spotify, Alibaba, …

Libraries

- Apache Storm
- Netflix Suro project

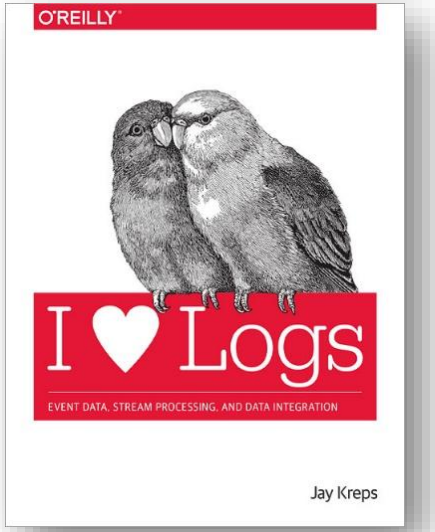


Kappa architecture

Proposed by Jay Krepps (Apache Kafka), 2013

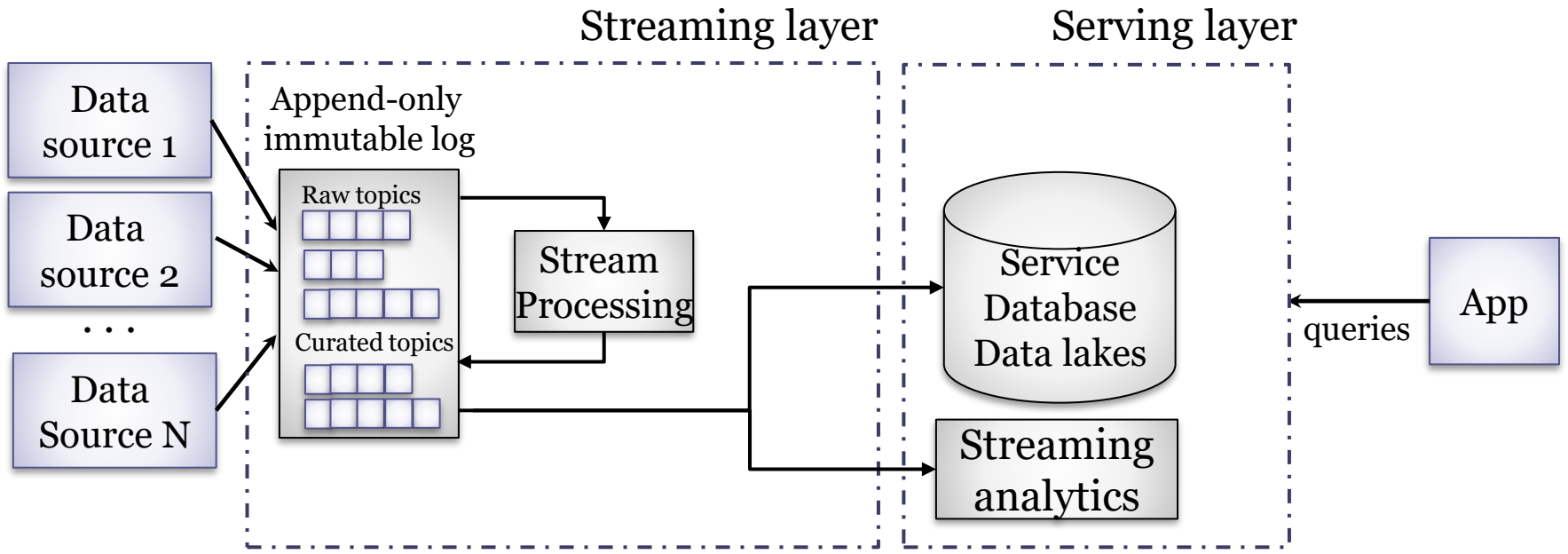
Handle Big data & Real time with logs

- Simplifies Lambda architecture
  - Removes the batch layer
- Based on a distributed ordered log
  - Replicated cluster
  - The log can be very large



Kappa architecture

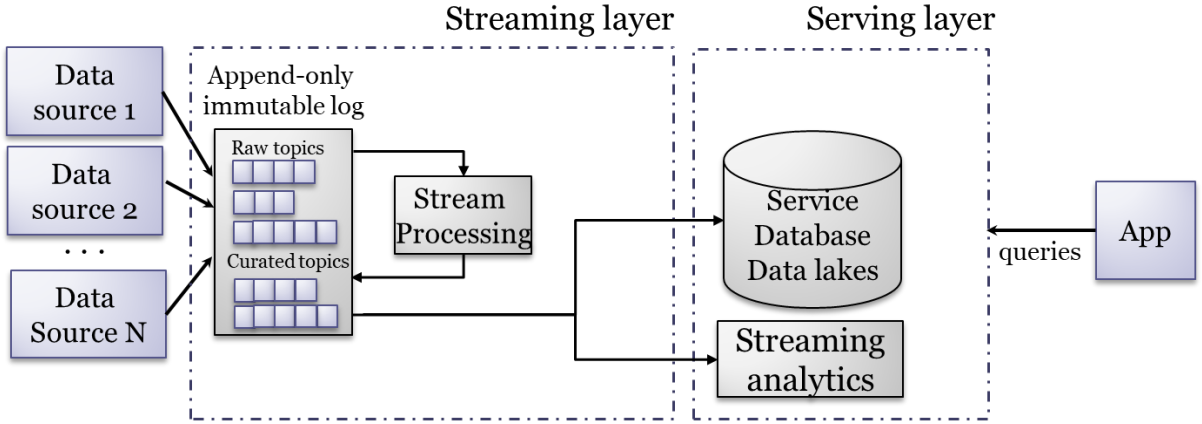
Diagram



Kappa architecture

Constraints

- The event log is append-only
- The events in the log are immutable
- Stream processing can request events at any position
  - To handle failures or doing recomputations



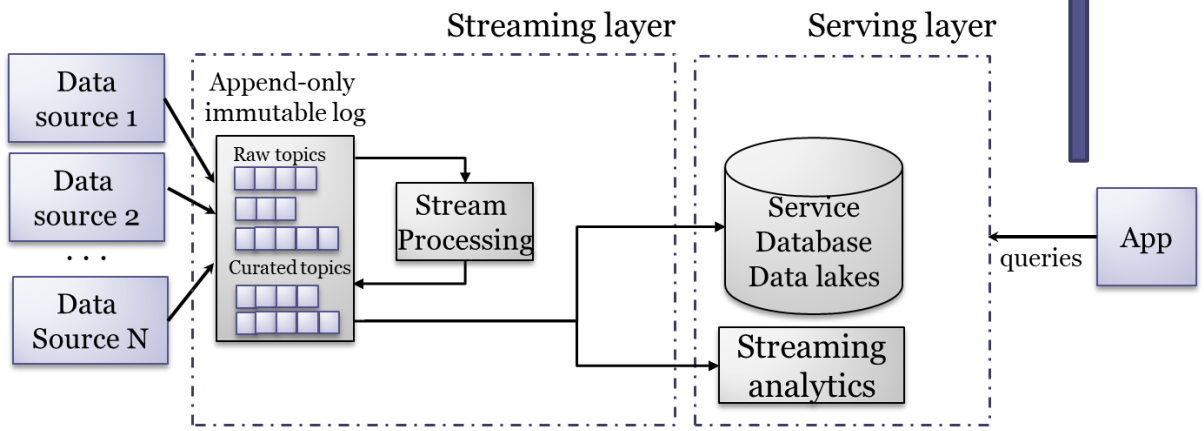
Kappa architecture

Advantages

- Scalable (big data)
- Real time processing
- Simpler than lambda
  - No batch layer

Challenges

- Space requirements
  - Duplication of log and DB
  - Log compaction
- Ordering of events
- Delivery paradigms
  - At least once
  - At most once (it may be lost)
  - Exactly once

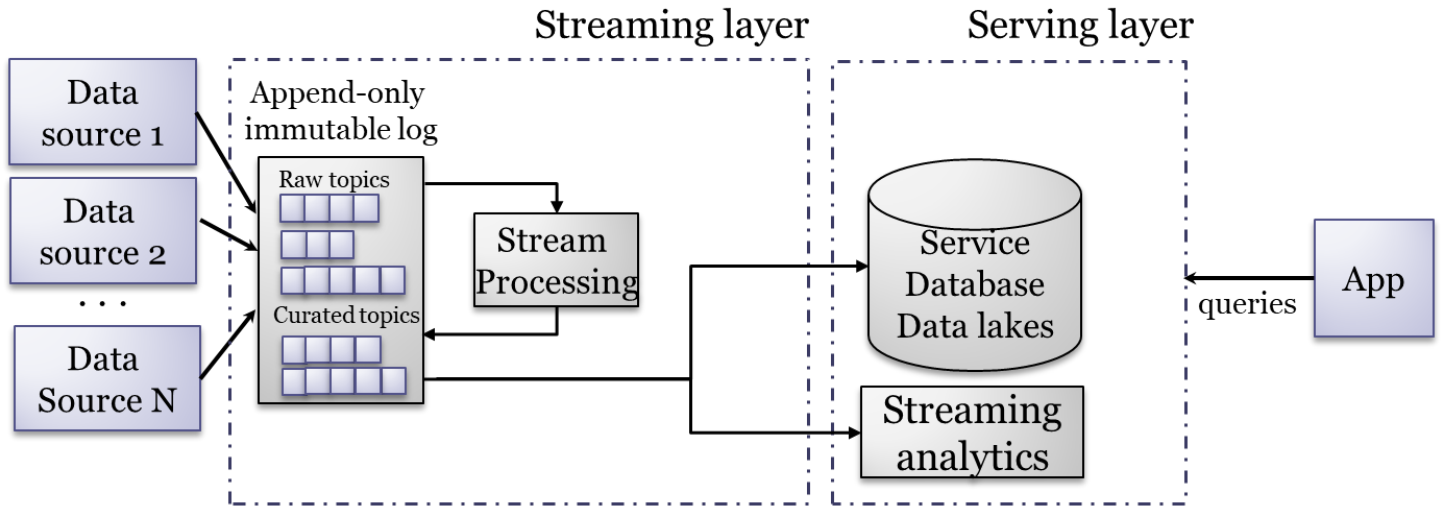


Kappa architecture

Applications & libraries

- Apache Kafka
- Apache Samza
- Spark Streaming

LinkedIn



End of presentation