qaware.de

Hochschule **Rosenheim**
University of Applied Sciences

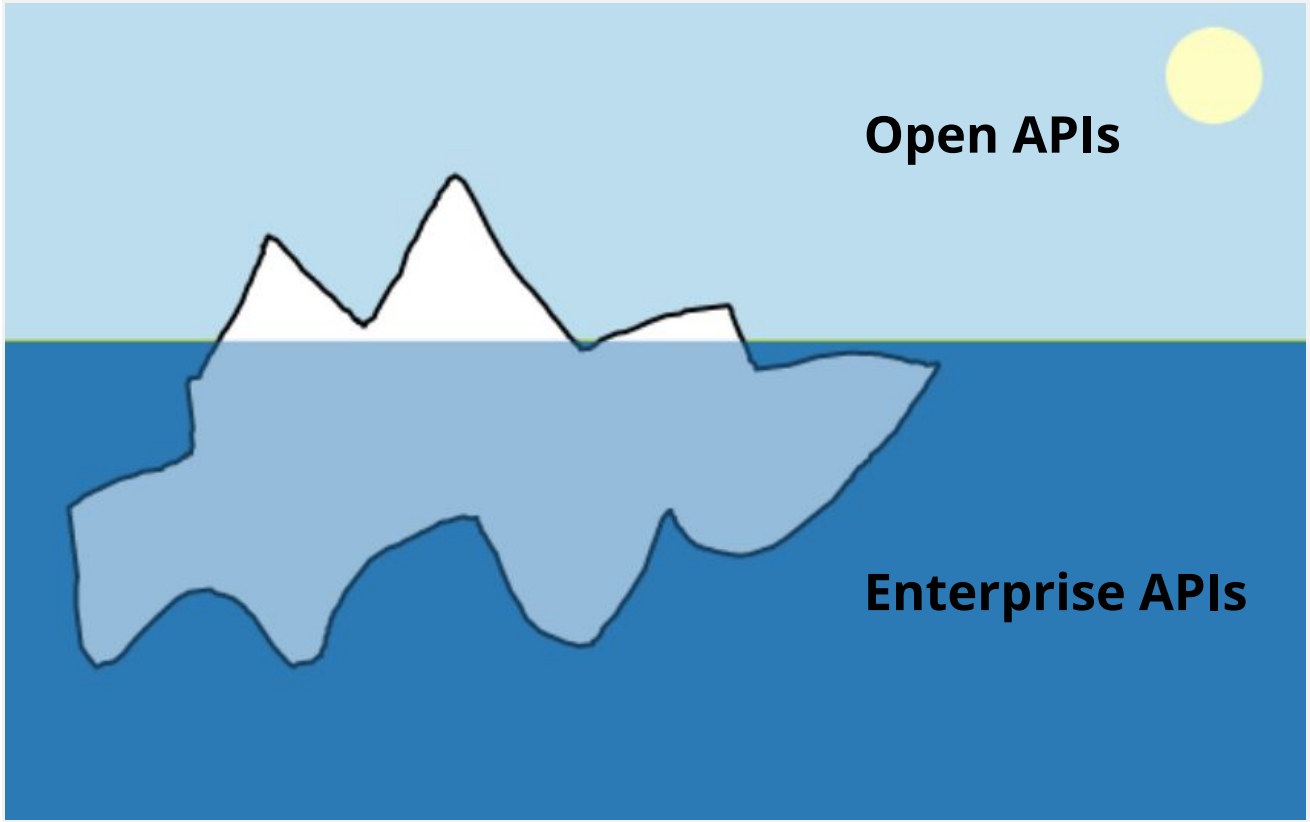
Cloud Computing Communication

Lukas Buchner

2023-10-12

"One cannot not communicate"

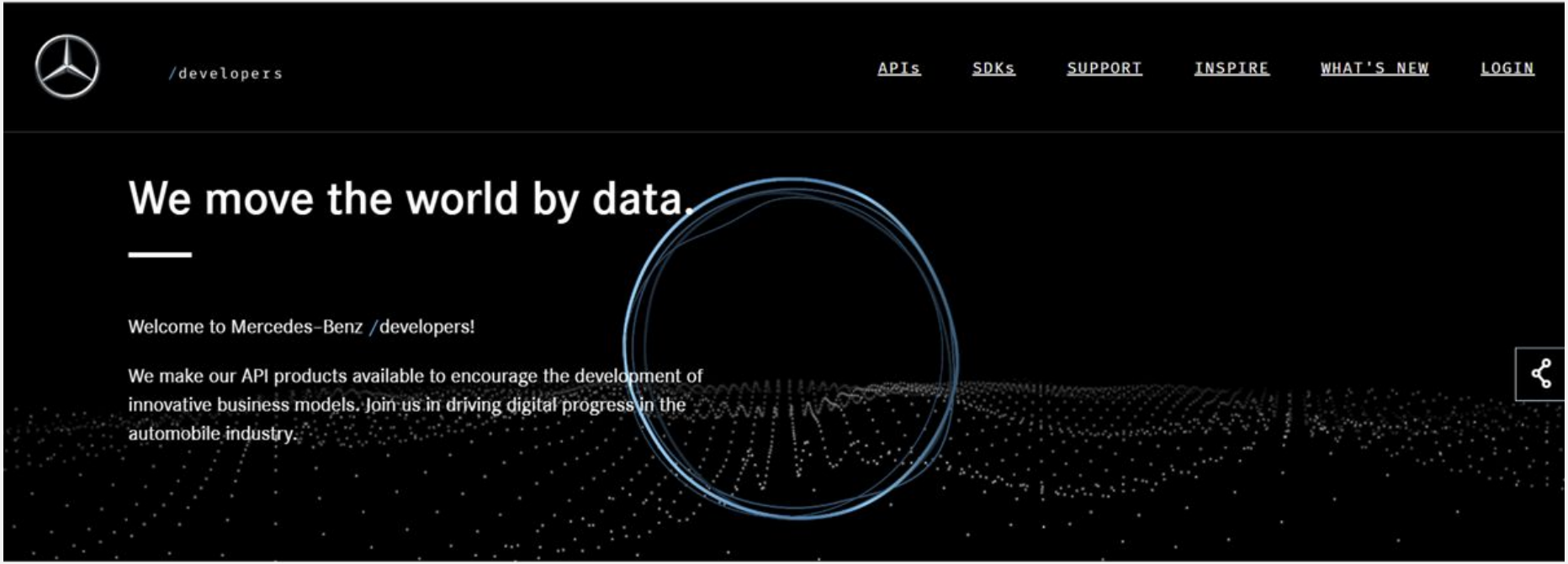
- Paul Watzlawik



- Just a fraction of all existing APIs are openly accessible
- APIs may earn money
- Some APIs are demanded to be open by law.
- Open APIs aid the innovation of new products.

https://joshdata.me/iceberger.html , inspired by https://twitter.com/GlacialMeg/status/1362557149147058178

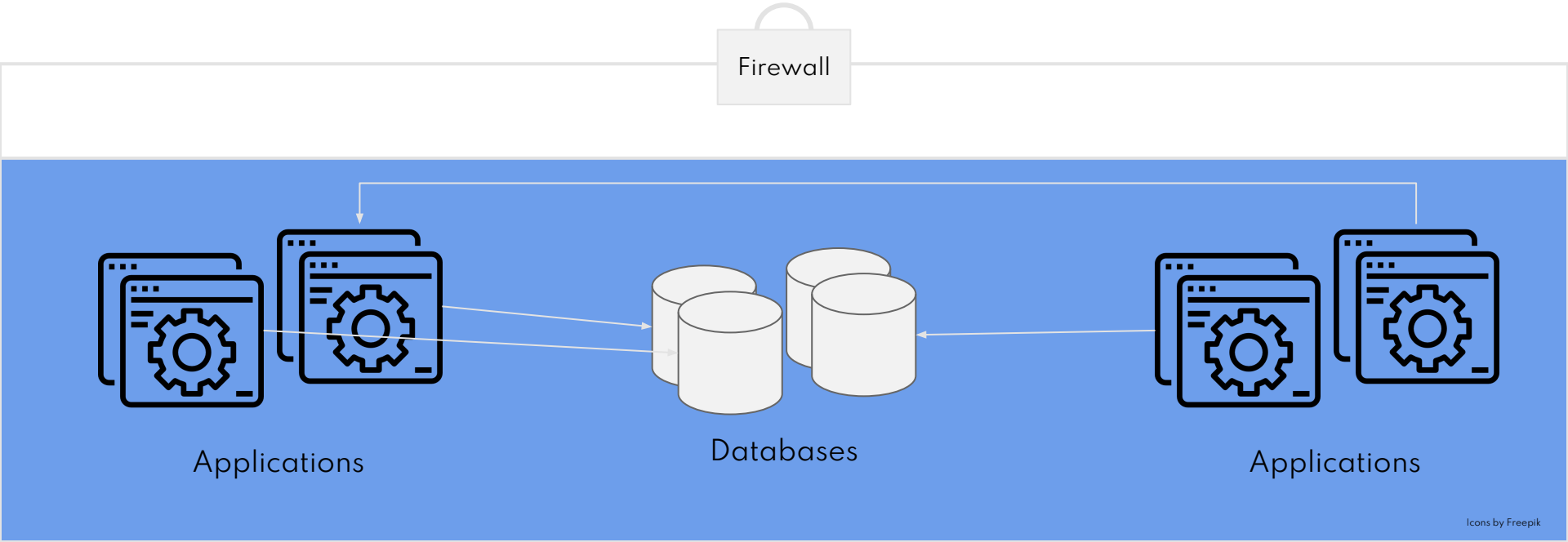
Example: Mercedes Benz Developer Portal



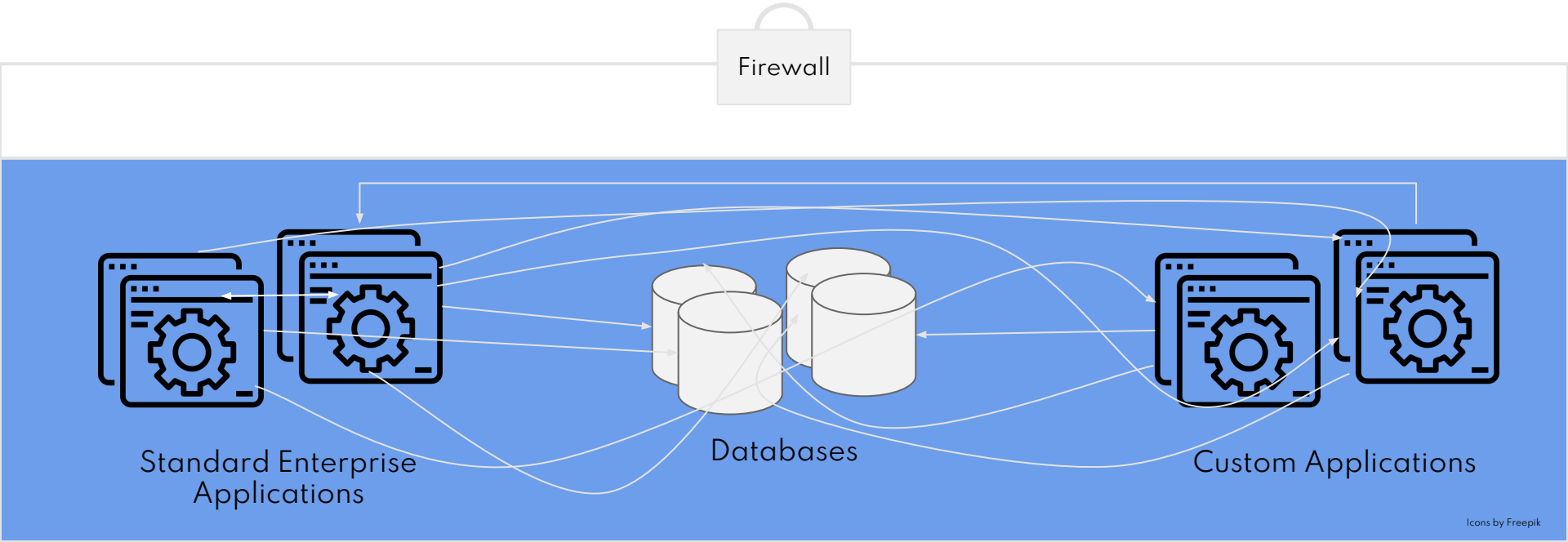
https://developer.mercedes-benz.com/

How does that affect Cloud Computing?

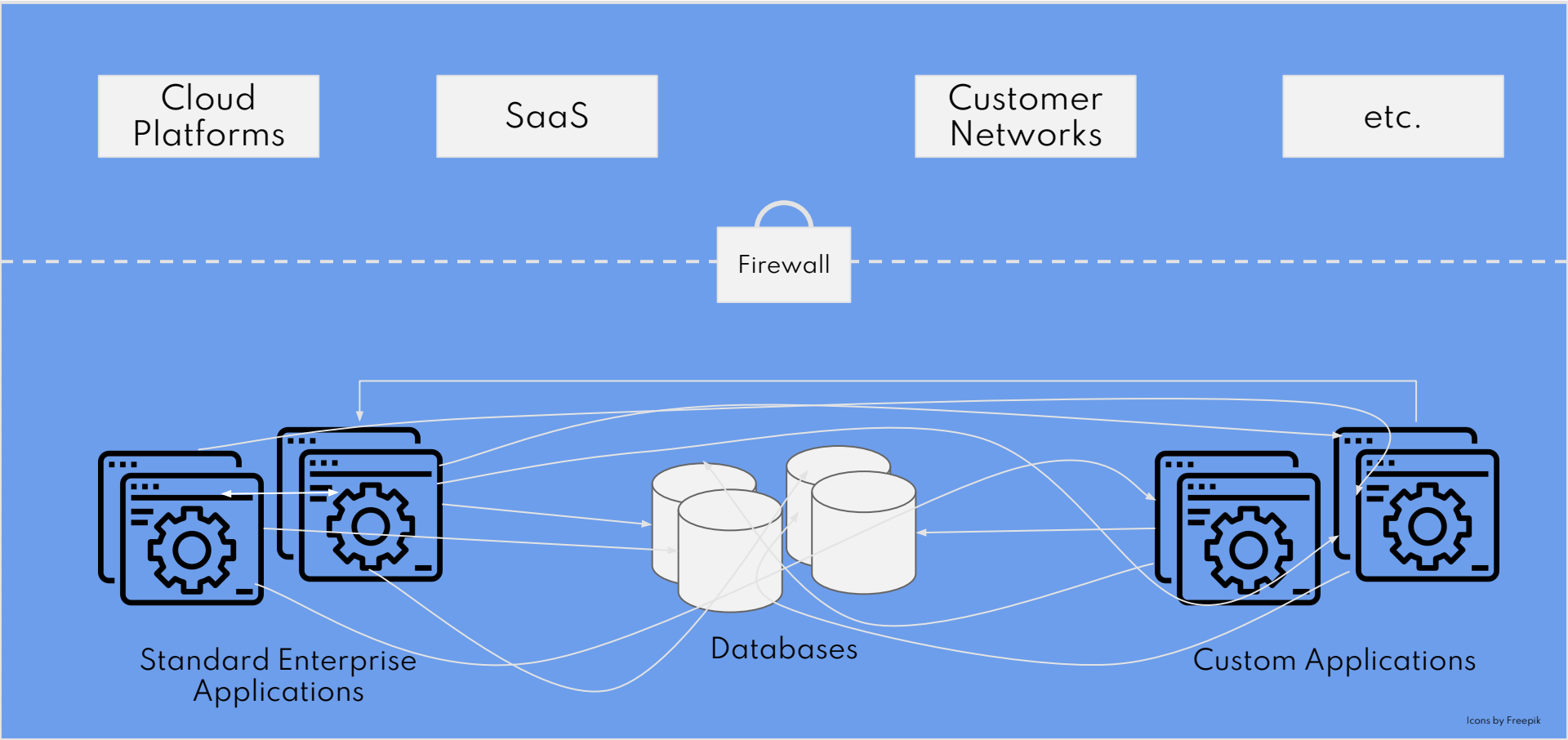
A few decades ago enterprise software was hidden behind a firewall



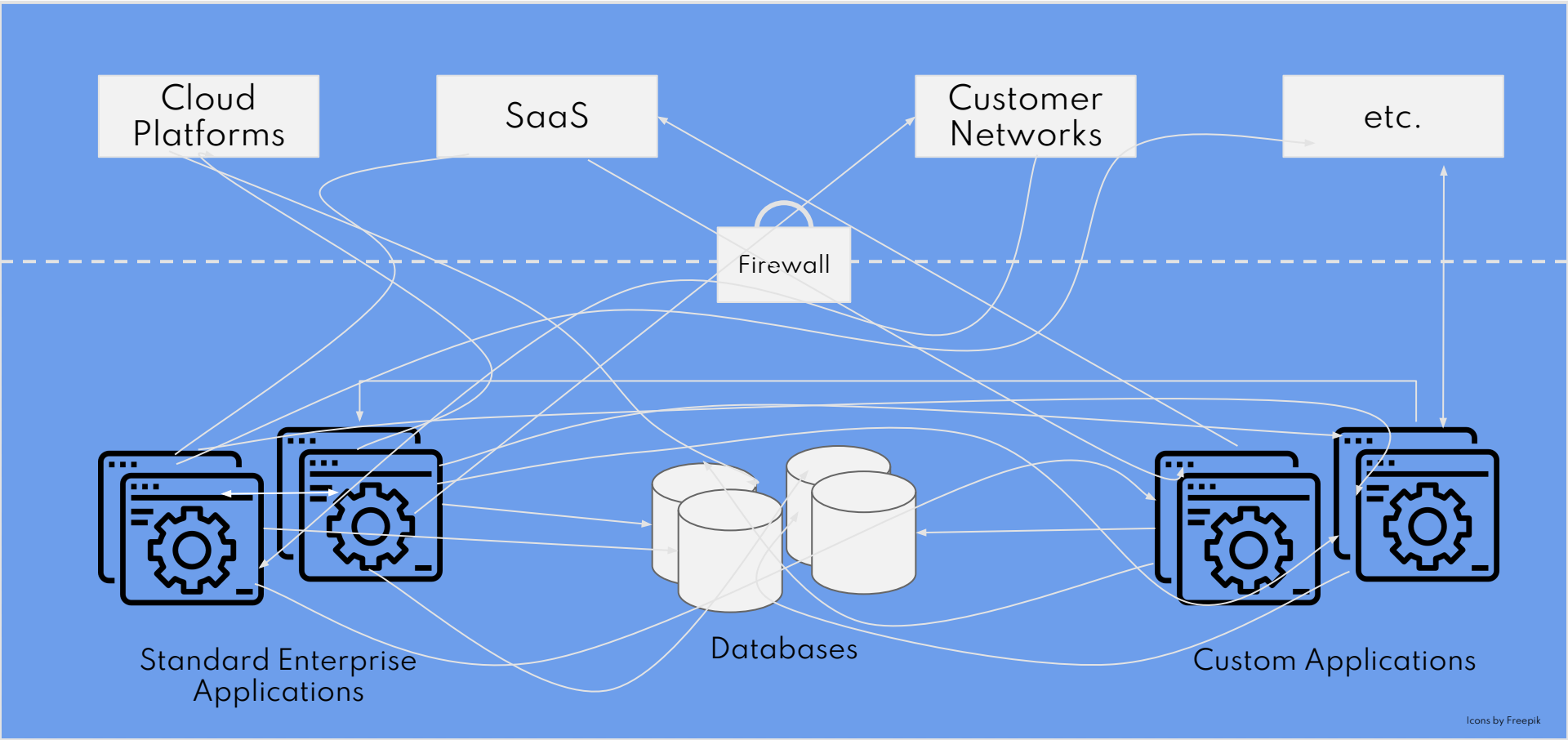
Over time...


Firewall
Standard Enterprise Applications
Databases
Custom Applications
Icons by Freepik

The network extends beyond the companies network boundaries

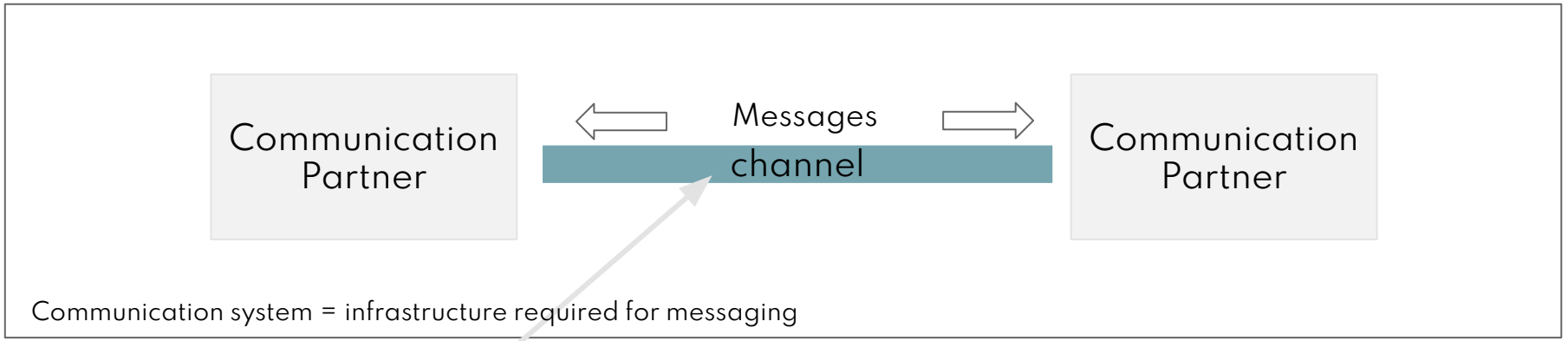


When reality kicks in...



Communication models

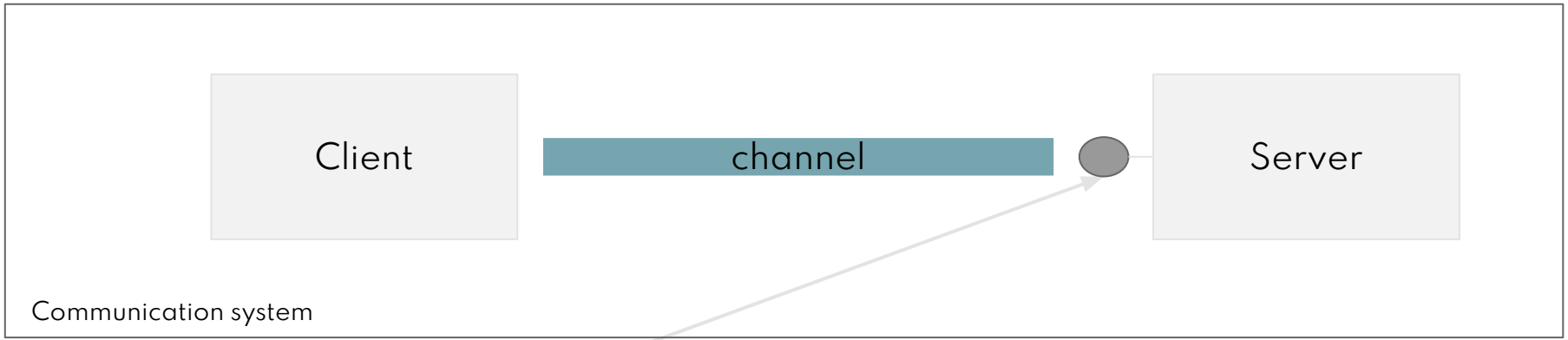
A general communication model for the internet - adapted from Shannon/Weaver.



Typical properties of a channel:

- direction
- data format
- synchronous/asynchronous
- reliability and guarantees
- security
- performance (latency, bandwidth)
- overhead (payload / total load)

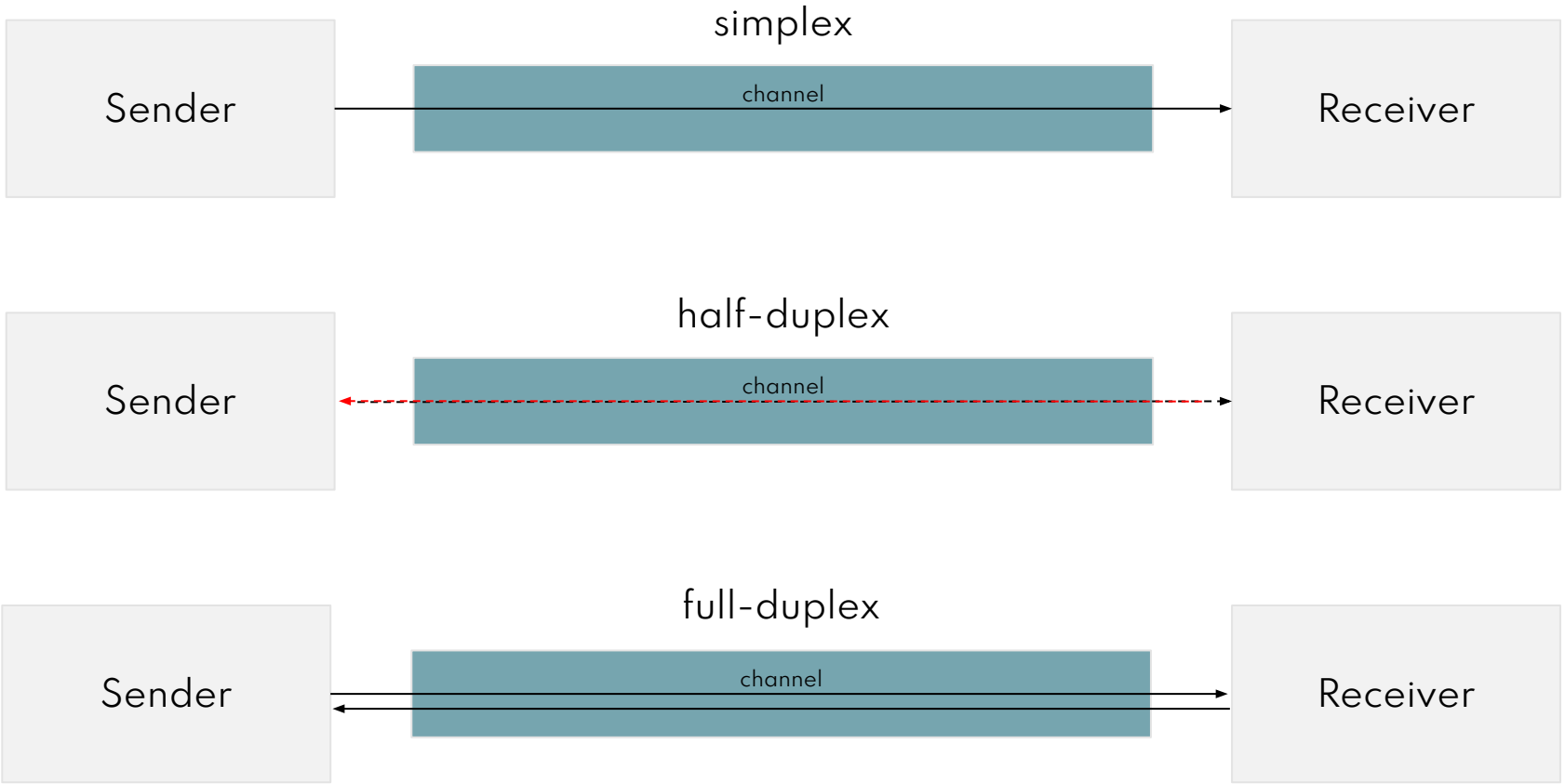
Service-orientation in a communication system: client-server-communication via services



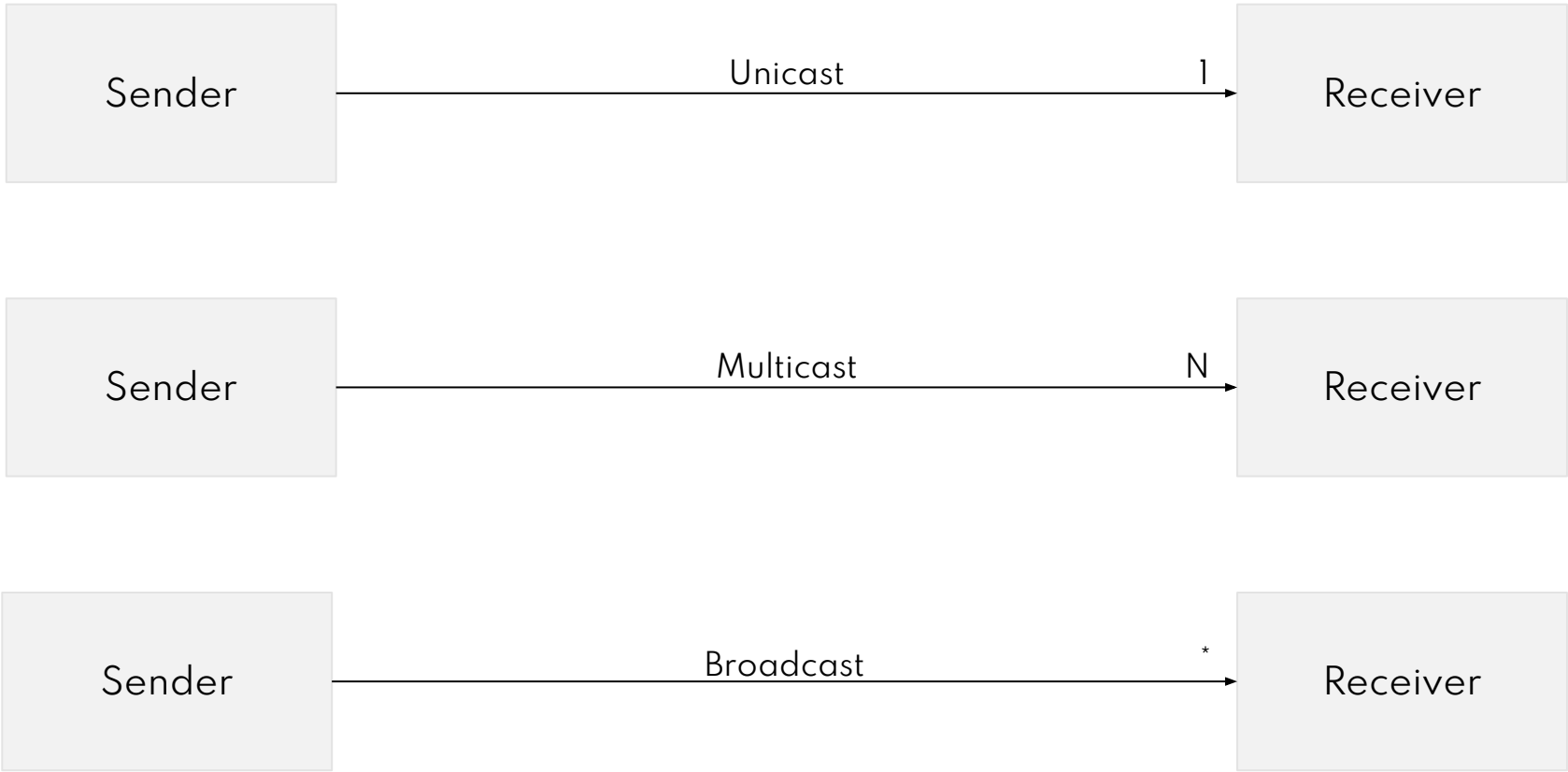
A **service** is a functionality provided through a defined interface.
Each service is defined by a service interface.

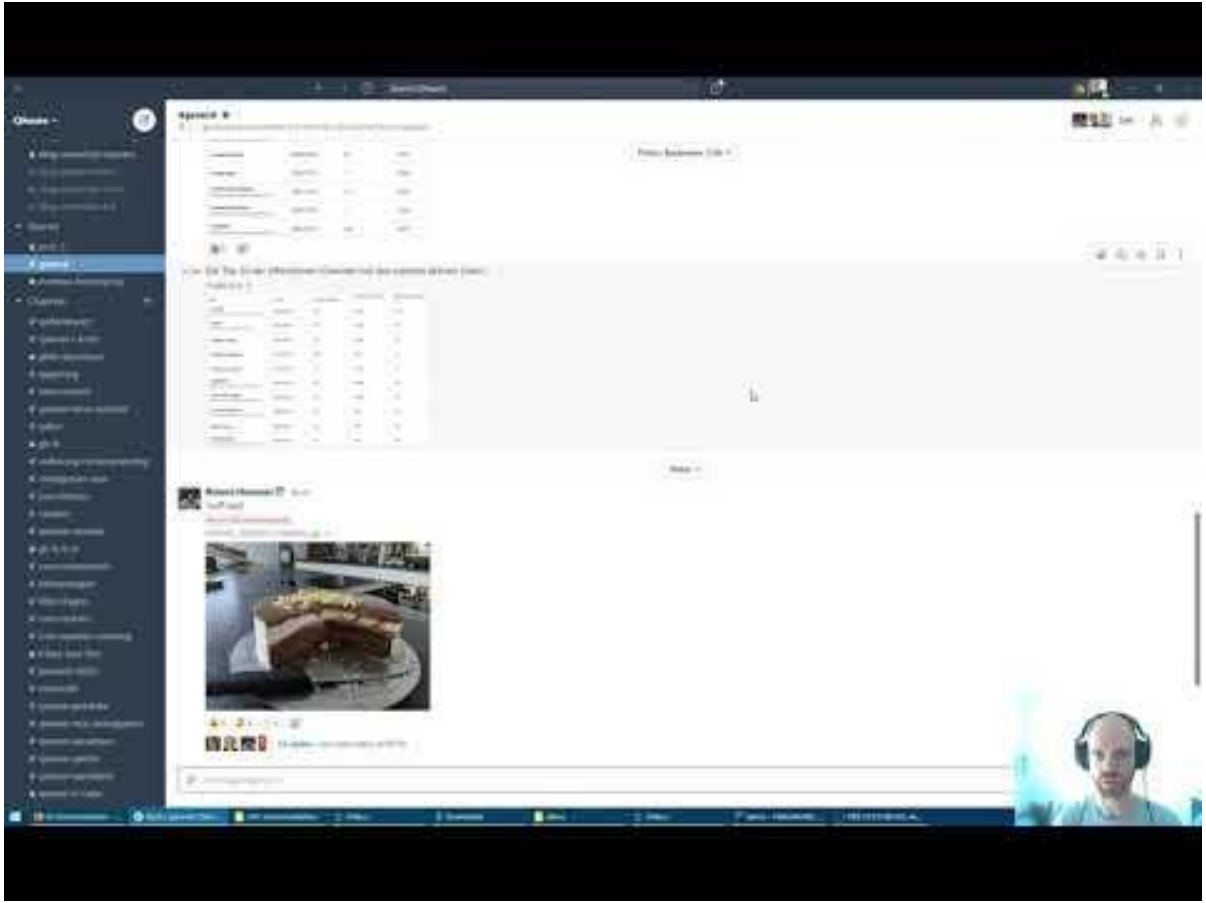
A service interface is a contract between the user and the provider regarding the syntax and semantics of service usage and optionally includes guarantees regarding the Quality of Service.

Usage patterns of a channel (Direction)



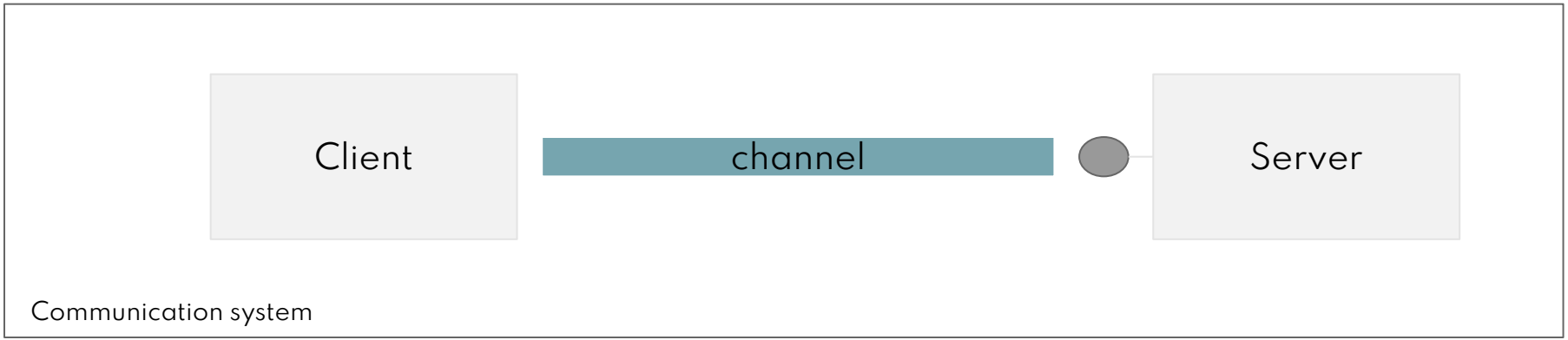
Cardinality of message receivers





https://www.youtube.com/watch?v=ED2NgbPZan0

Who initiates the communication?



Client starts communication

Request-Response

requests response from server

Unidirectional notification

Push

server starts communication

Peer-to-Peer

Both communication partners may start the communication.

HTTP

Almost all cloud communication is based on TCP and HTTP.

Version	Year introduced	Current status	Usage in August 2024	Support in August 2024
HTTP/0.9	1991	Obsolete	0	100%
HTTP/1.0	1996	Obsolete	0	100%
HTTP/1.1	1997	Standard	33.8%	100%
HTTP/2	2015	Standard	35.3%	66.2%
HTTP/3	2022	Standard	30.9%	30.9%

TCP

- Developed from 1973 and standardized in 1981
- Reliable full-duplex end-to-end connection
- An endpoint is an IP + Port.

HTTP

- HTTP 1.0: Developed in 1989 at CERN
- HTTP 1.1: Connection Pooling / Keep-Alive, HTTP Pipelining, methods PUT and DELETE
- HTTP 2.0: Binary stream, multiplexing, encryption as standard, various performance optimizations, push
- HTTP 3.0: Uses QUIC as the transport layer (which in turn uses UDP)

Communication patterns in HTTP

- **Request/Response:** Classic HTTP. The client sends a request, and the server responds with a response. The client then sends another request, and so on.
- **Push**: Server-Sent Events (SSE). The client sends a request, and the server responds with a response. Now the server can also send additional messages to the client without receiving new requests from the client.
- **Peer-to-Peer**: Websockets. The client sends a request, and the server responds with an upgrade response. Now the server can send messages to the client, and the client can send messages to the server (bidirectional channel).

An exemplary HTTP communication

Client:

```
GET / HTTP/1.1

Host:
www.example.com
```

Server:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
...
```

Data formats

JSON

```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Währung": "EURO",
  "Inhaber": {
    "Name": "Mustermann",
    "Vorname": "Max",
    "männlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Alter": 42,
    "Kinder": [],
    "Partner": null
  }
}
```

- JSON = JavaScript Object Notation (pure data).
- Exists also as binary encoding (BSON – Binary JSON).
- MIME-Typ: *application/json*
- Schema-descriptors: JSON Schema (http://json-schema.org)

Data Types::

- null value:: null
- booleans: true, false
- numbers: 42, 2e+6
- characters: “Mustermann“
- arrays: [1,2,3]
- objects with properties: {“Name“:“Mustermann“}

Protocol Buffers

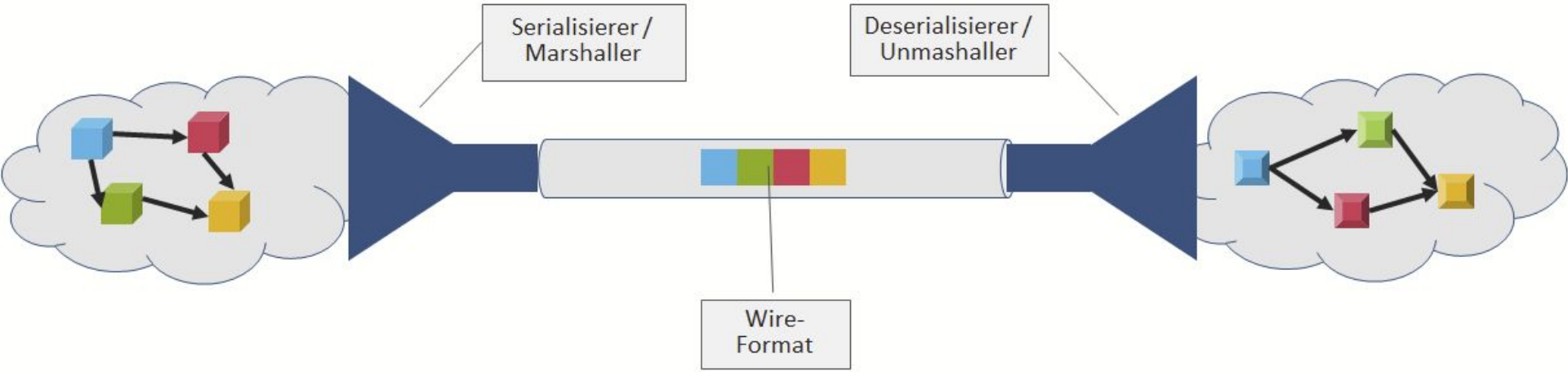
```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

- Released by Google in 2008
- Language-neutral
- Platform-independent
- Binary-encoded - memory efficient and fast
- Protobuf source is translated into programming language by the Protobuf compiler

https://www.youtube.com/watch?v=uGYZn6xk-hA

Usually serialization & deserialization is needed

Serialization/Marshalling is the process of converting an object, data structure, or complex data format (such as a Python object, Java object, etc.) into a format that can be easily stored or transmitted and later reconstructed. The resulting data can be stored in files, sent over a network, or communicated between different systems.



Performance numbers

the higher, the better

Benchmark	Mode	Cnt	Score	Error	Units
jsonDeserialize	thrpt	20	2969647,037	± 26838,990	ops/s
jsonSerialize	thrpt	20	4126749,377	± 40462,442	ops/s
protoBufDeserialize	thrpt	20	6992710,402	± 254170,355	ops/s
protoBufSerialize	thrpt	20	22019405,324	± 741610,436	ops/s
xmlDeserialize	thrpt	20	577380,698	± 9450,761	ops/s
xmlSerialize	thrpt	20	1578672,085	± 40488,728	ops/s

Service oriented Request-Response-Communication via REST

https://youtu.be/SLwpqD8n3d0

REST is a paradigm for application services based on the HTTP protocol

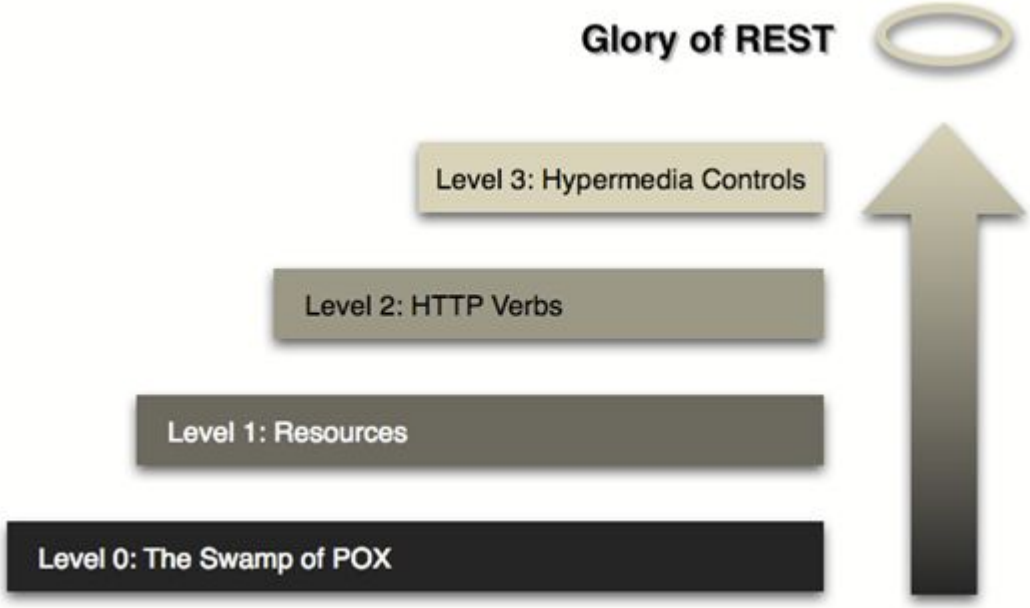
- REST is a paradigm for designing interfaces of web applications based on the HTTP protocol (verbs).
- Dissertation by Roy Fielding: "Architectural Styles and the Design of Network-based Software Architectures", 2000, University of California, Irvine.

REST is a paradigm for application services based on the HTTP protocol

Basic properties:

- **Everything is a resource**: A resource is uniquely addressable via a URI, has one or more representations (XML, JSON, any MIME type), and can link to other resources via hyperlink. Resources are, wherever possible, hierarchically navigable.
- **Uniform interfaces**: Services are based on HTTP methods (POST = create, PUT = update or create, DELETE = delete, GET = retrieve). Errors are reported through HTTP codes. Thus, services have standardized semantics and a stable syntax.
- **Stateless**: The communication between server and client is stateless. A state is maintained on the client only through URIs.
- **Connectivity**: Based on mature and ubiquitous infrastructure: The web infrastructure with effective caching and security mechanisms, powerful servers, and, for example, web browsers as clients.

The REST maturity model



https://martinfowler.com/articles/richardsonMaturityModel.html

What do **updates** for orders look like?

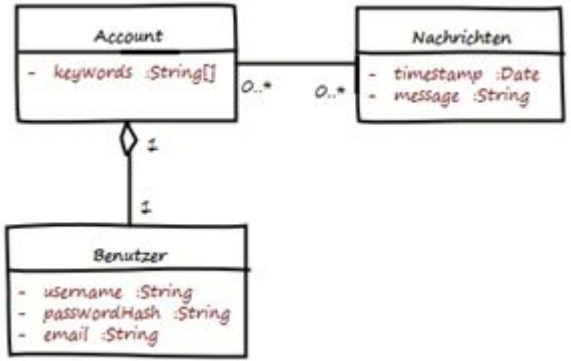
http://www.service.de/customers/4711/orders
Input: Order per PUT
Output: HTTP-Code (200, 500) +
Link to changed Order

http://www.service.de/customers/4711/orders
Input: Order per PUT
Output: HTTP-Code (200, 500)

http://www.service.de/customers/4711/orders
Input: Order per POST
Output: UpdateOrderConfirmation (OK)

http://www.service.de/customers/updateOrderService
Input: UpdateOrderRequest per POST
Output: UpdateOrderConfirmation (OK)

Developing REST APIs



In the beginning:

- Define use cases
- Create entity models
- Choose out of either alternatives:

Top-Down-Approach:

- define REST-interface (e.g. in OpenAPI)
- generate code for REST-interface (e.g. with OpenAPI Generator)
- implement REST-interface

Bottom-Up-Approach:

- implement REST-interface
- generate definition of REST-interface (e.g. with OpenAPI)

REST-API with Java's JAX-RS


```
@Path("/hello/{name}")
public class HelloWorldResource {

    @GET
    @Produces("application/json")
    public ResponseMessage getMessage(
            @DefaultValue("Hallo") @QueryParam("salutation") String salutation,
            @PathParam("name") String name) throws IOException {
        ResponseMessage response = new ResponseMessage(new Date().toString(), salutation + " " + name);
        return response;
    }

}
```

REST-API with Java's Spring Web MVC

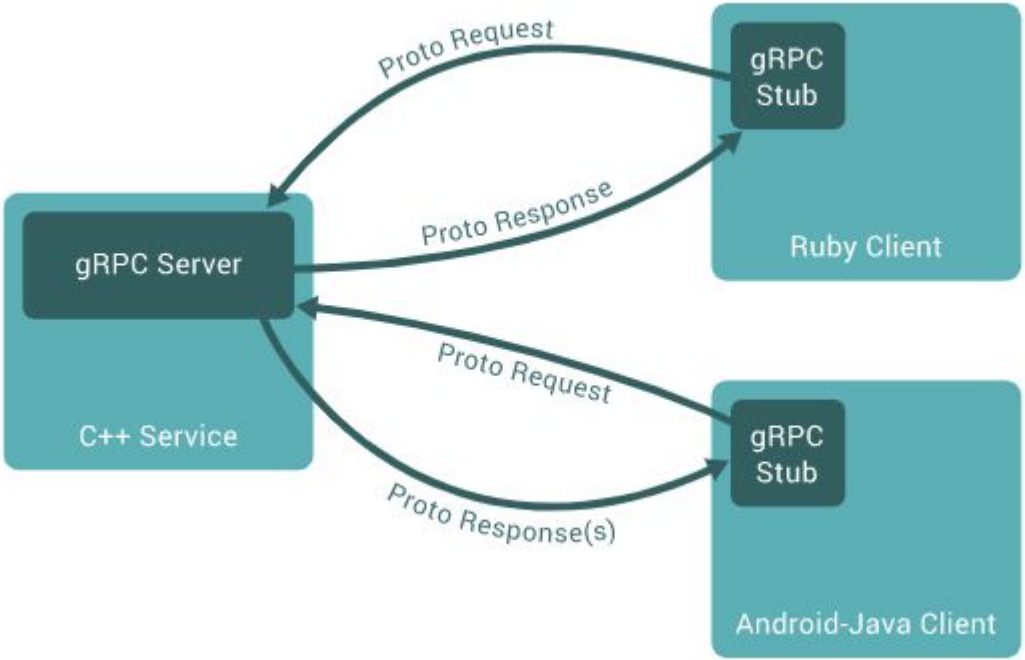
```java
@RequestMapping("/hello/{name}")
public class HelloWorldController {
   @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
   public ResponseMessage getMessage(
           @RequestParam(name = "salutation", defaultValue = "Hallo") String salutation,
           @PathParam("name") String name
   ) {
       return new ResponseMessage(Instant.now().toString(), salutation + " " + name);
   }
}
```

Service oriented Request-Response-Communication via gRPC

gRPC

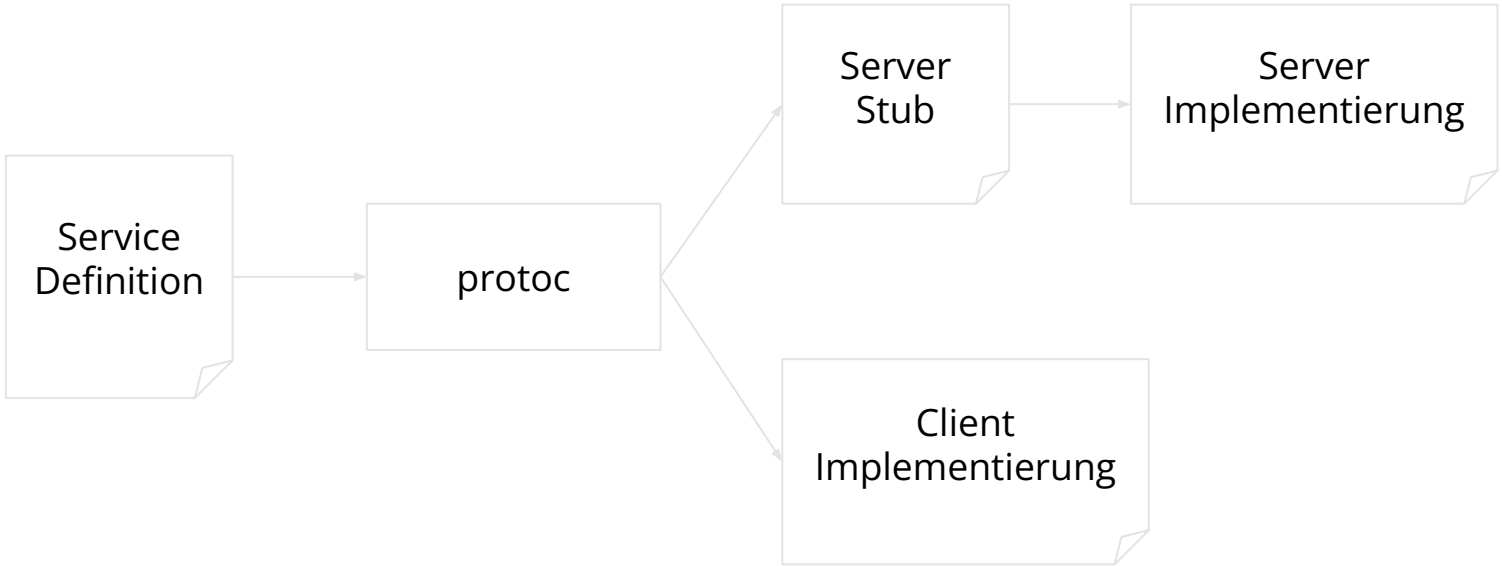
- Developed by Google in 2015
- HTTP/2 as the transport protocol
- Protocol Buffers as the serialization format
- Types of communication
  - Request-Response
  - Client-side Streaming
  - Server-side Streaming
  - Bidirectional Streaming
- Generator for server and client implementations
  - Official languages: C, C++, C#, Dart, Go, Java, Kotlin, Node.js, Objective-C, PHP, Python, Ruby

gRPC

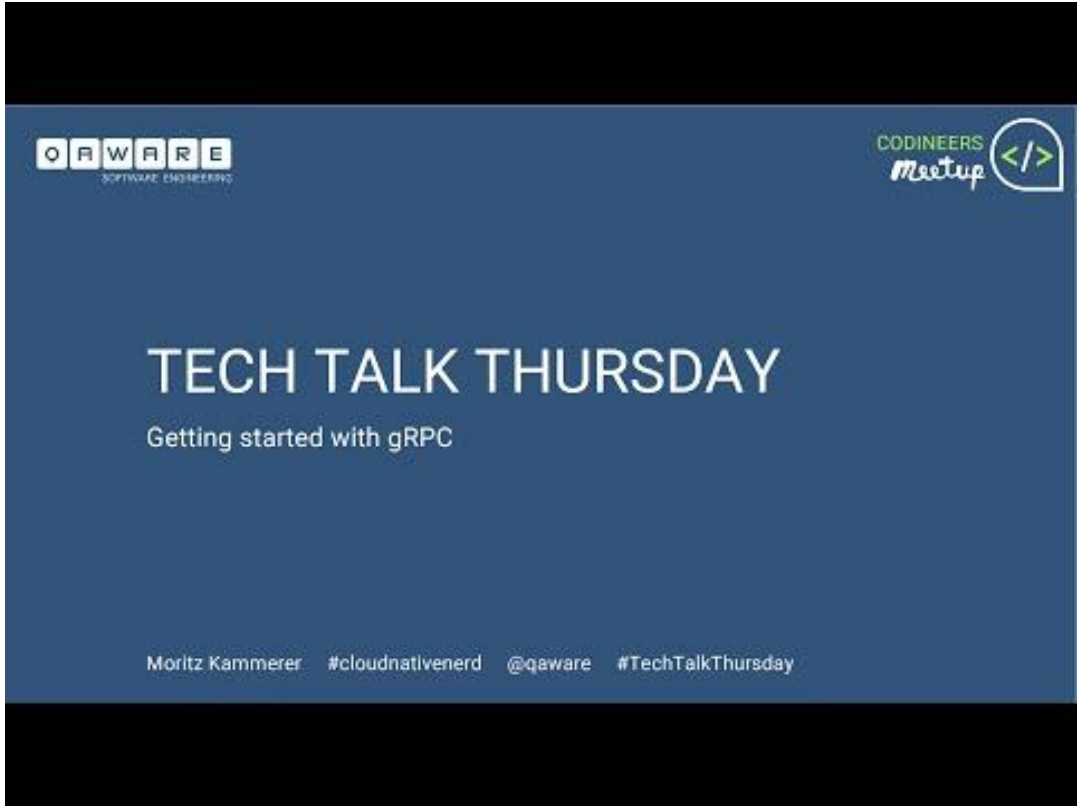


https://grpc.io/docs/what-is-grpc/introduction/

gRPC Workflow



Demo

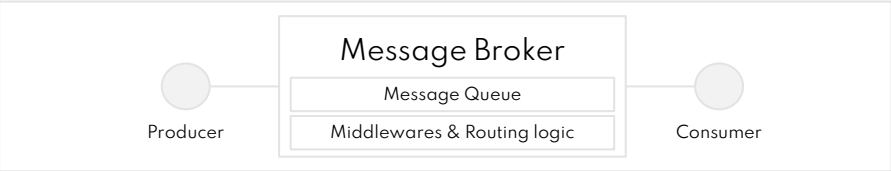


https://www.youtube.com/watch?v=Qw8bPpw8h-g

Übung

Flexible Communication Patterns via Messaging

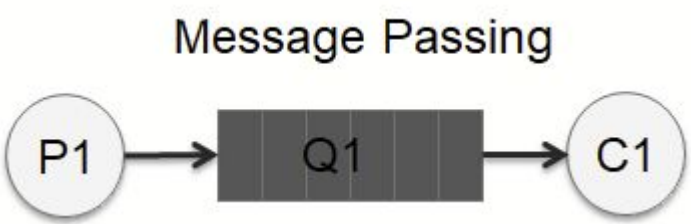
Messaging as resilient, asynchronous messaging

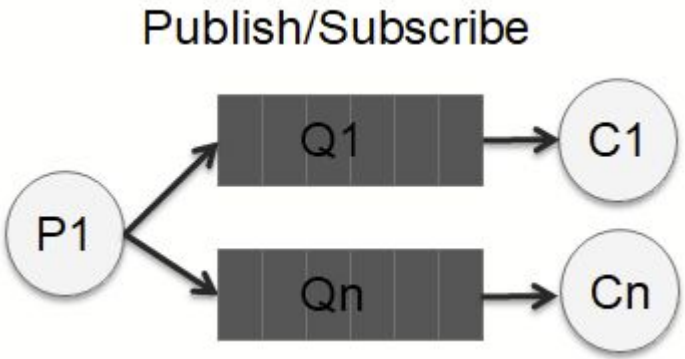


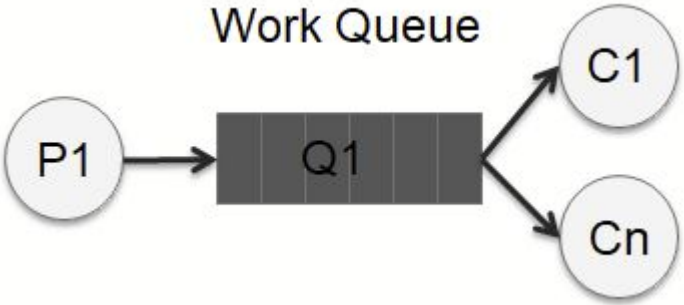
Decouples Producer and Consumer

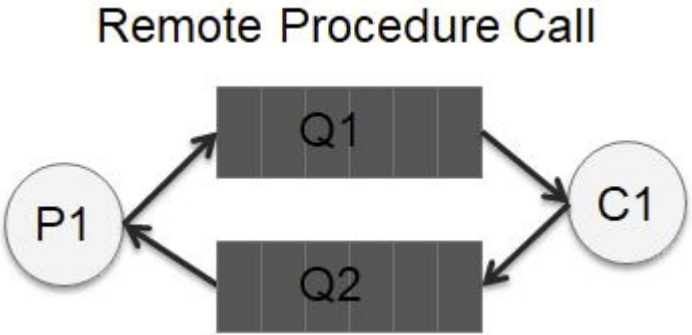
- Service interface: application format
- Message broker: no restrictions on the format
- Advantages
  - The sending and receiving times can be spaced apart as long as needed
  - Horizontal scalability: Can be delivered to multiple consumers
  - Load balancing: e.g., delivery to the consumer with the least workload
  - Handling peak loads: Message delivery can be delayed if consumers are overloaded
- Configuration options:
  - TTL (Time to Live) of the message
  - Delivery guarantee (at least once, exactly once, no guarantee)
  - Transactionality
  - Message priority
  - Order compliance

Messaging Patterns







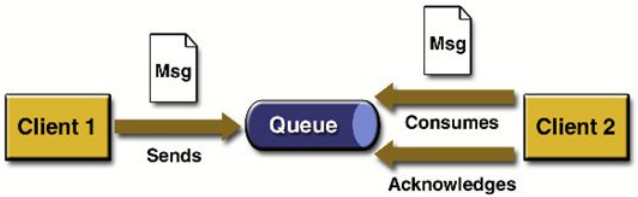


JakartaEE (JavaEE) Standard: JMS

JMS = Jakarta Messaging API (Java Messaging Service). Standardised API via Java-Enterprise-Edition-Specification. (Does not standardise message format)

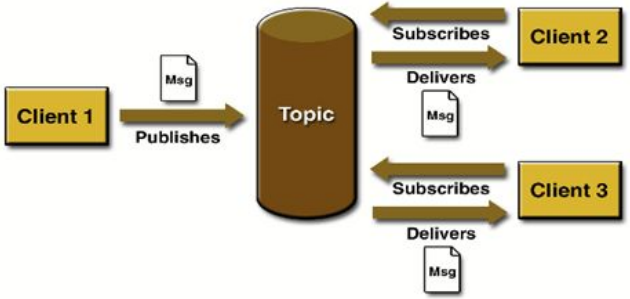
- 2002-2013: Version 1.1. Stable and widely used in the Java world.
- Since May 2013: Version 2.0 Part of the JEE 7 specification
- Since November 2020: Version 3.0 as part of the Jakarta 9 specification

Message Passing



- One consumer per message
- acknowledgement of received message

Publish / Subscribe



- Multiple consumers per message

AMQP: standardized protocol for Messaging

Problem: Message brokers were proprietary and incompatible with each other. Messaging across company boundaries was very difficult.

Solution: AMQP. Standardizes the protocol for messaging. Version 1.0 since the end of 2011.

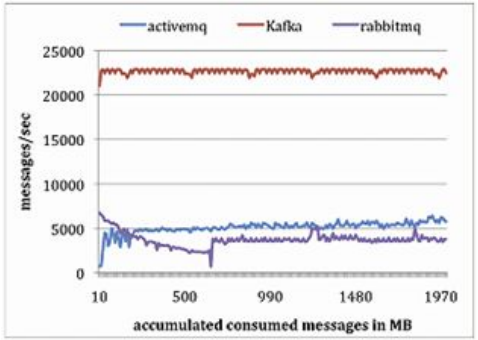
- Standardizes a network protocol for communication between clients and message brokers.
- Standardizes a model of available APIs and building blocks for message brokering and storage (Producer, Exchange, Queue, Consumer).
- Supports all known messaging patterns.
-

The standardization committee encompasses Cisco, Microsoft, Red Hat, Deutsche Börse Systems, IONA, Novell, Credit Suisse, JPMorganChase et. al.

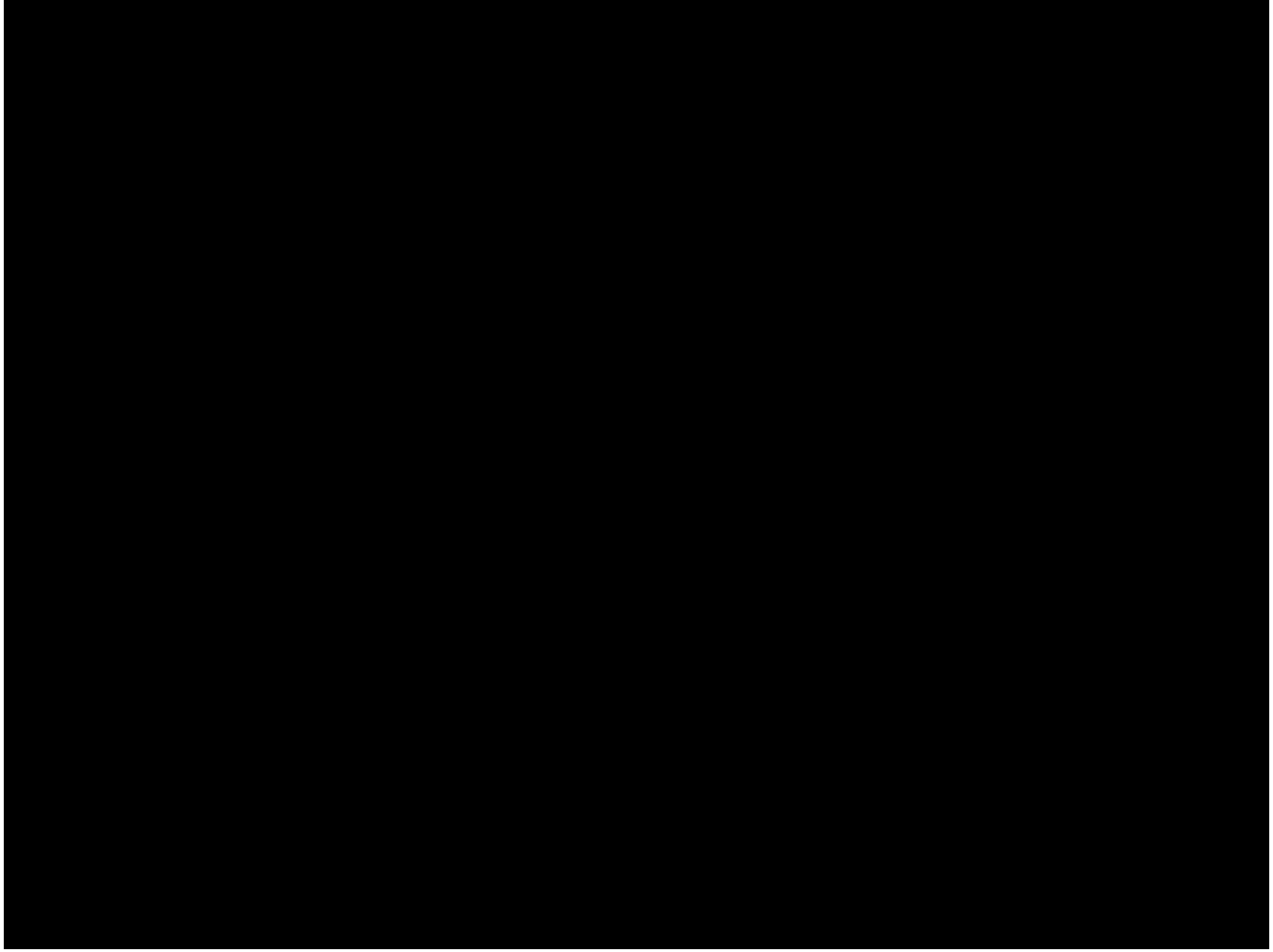
Well known Brokers include RabbitMQ and ActiveMQ.

Apache Kafka

- Developed at LinkedIn and released as an open-source project in 2011. Since 2012, it has been under the Apache Foundation.
- Kafka has become the de-facto standard for messaging in the cloud because Kafka is highly distributable and significantly faster than comparable solutions.



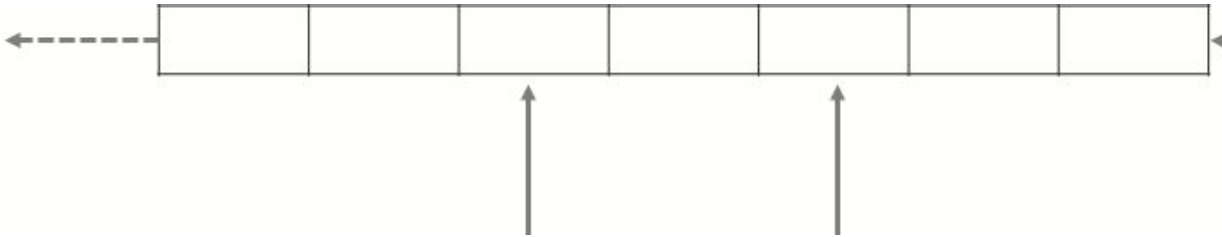
- Kafka is so fast because it intelligently uses operating system resources, has an efficient encoding format for messages, and maintains the delivery state in the clients.
- Kafka is written in Java and Scala.
- The Kafka API is proprietary and is not based on any messaging standard.



https://www.youtube.com/watch?v=k-7lz6Ex354

Kafka is based on an Event-Log data structure. Each consumer has its own state and pointer into the log.

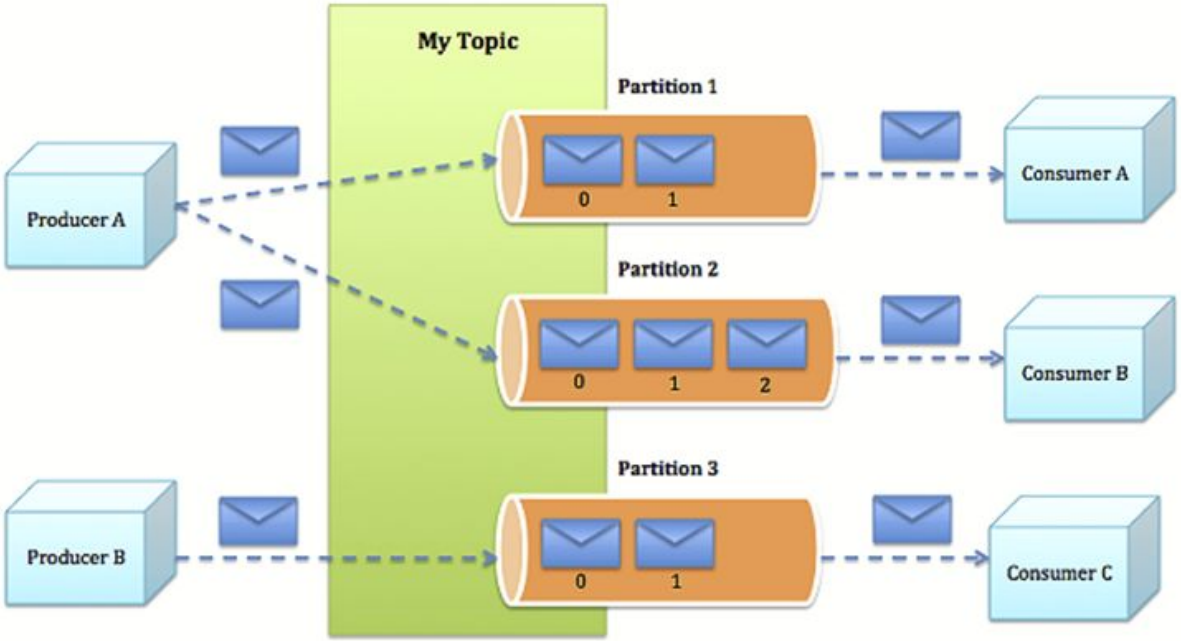
Old events are deleted according to defined criteria (e.g., age, maximum topic size)



New events are appended to the end of the event-log

Pointer to the last read event of a client (managed by the client itself, offset can be stored in Kafka)

Kafka's event log is distributed

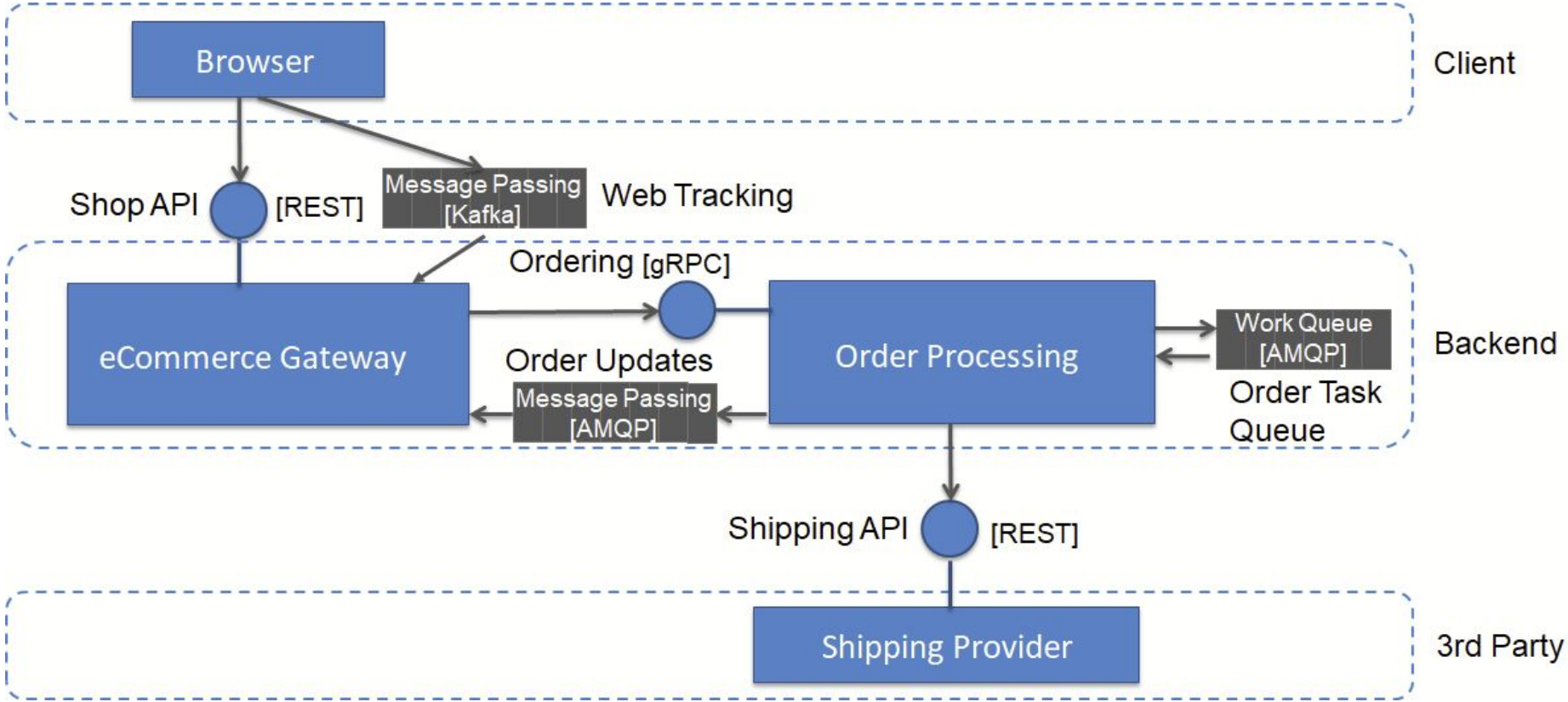


- The events in a topic are divided into partitions
- The partitions are distributed across the available broker instances
- Partitions are replicated for fault tolerance

see:

- http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node
- http://www.infoq.com/articles/apache-kafka

Exemplary architecture


Browser
Client
Shop API
[REST]
Message Passing
[Kafka]
Web Tracking
Ordering [gRPC]
eCommerce Gateway
Order Updates
Message Passing
[AMQP]
Order Processing
Work Queue
[AMQP]
Order Task
Queue
Backend
Shipping API
[REST]
Shipping Provider
3rd Party

Literatur

Literatur

Books:

- Patterns of Enterprise Application Architecture, Martin Fowler, 2002
- Computer Networks, Andrew Tanenbaum, 2010
- Inter-Process Communication, Hephaestus Books, 2011

Internet:

- Dissertation von Roy Fielding zu REST

  http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- RESTful Webservices

  http://www.ibm.com/developerworks/webservices/library/ws-restful

- Richardson Maturity Model

  https://martinfowler.com/articles/richardsonMaturityModel.html