Programmable Web Project
Part 1: Introduction

Spring 2025

Services and APIs

The World Wide Web

Technologies for the World Wide Web

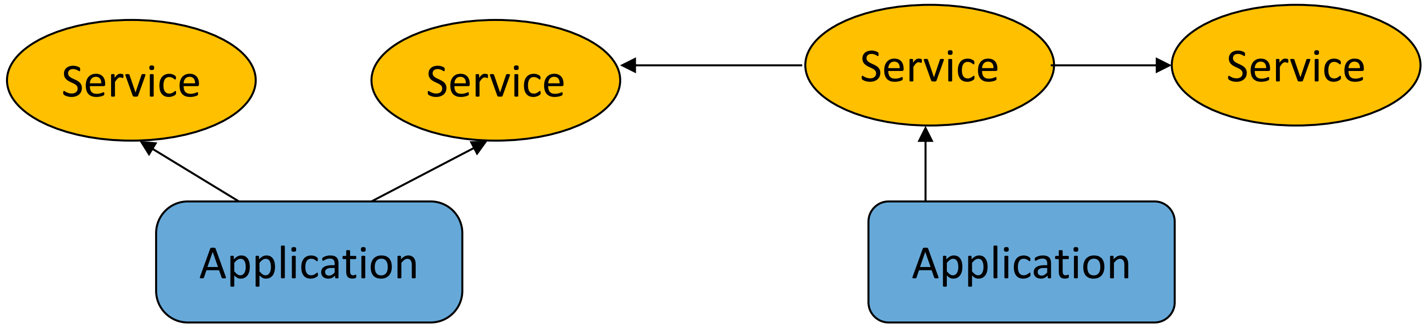
- **Backend: Business logic + data storage (databases)**
- **Transport protocol: HTTP**
- **Data serialization languages**
- **Clients**

Programmable Web

SERVICES AND APIS

Web Services

- Web services are logical units that provides certain functionality.
- They are **application independent**
  - services can be used by other services and applications.
  - services can incorporate the functionality of other services (**composite service**)



- Services need to communicate to the service consumer:
  - what **functionality** they provide
  - which **data formats** they accept and produce
  - what protocol they use

Web Services

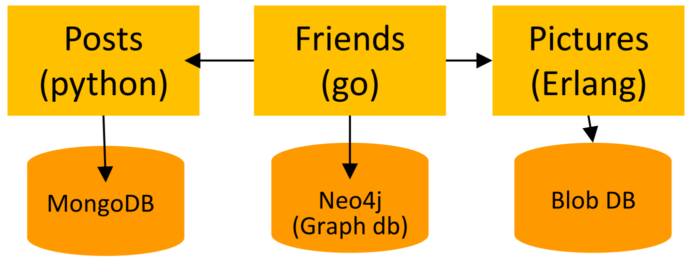
https://jsonplaceholder.typicode.com/

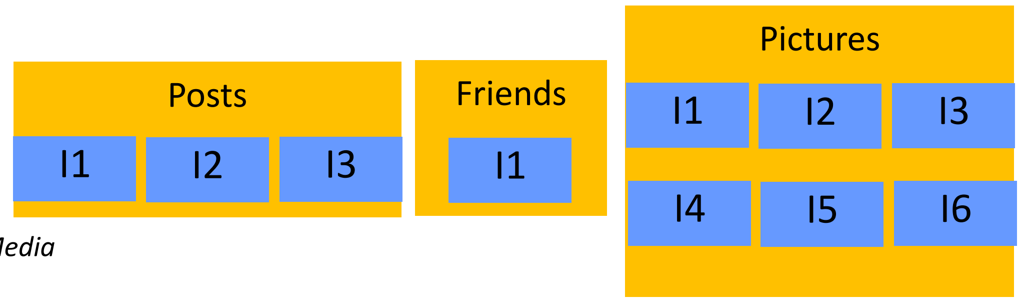
Web Services

- Online services that are not prepared to *human consumption* (in opposite to websites), but mainly machine-to-machine communication.
  - Web services require an architectural style to provide **clear and unambiguous interaction** (clearly defined interfaces), because there’s no smart human being on the client end to keep track.

Microservices

- Set of small and autonomous services that work together.
- SRP -> SINGLE RESPONSIBILITY PRINCIPLE
  - Business boundaries clear defined -> just a piece of functionality
  - Each microservice runs in its own OS process.
    - Change independently of each other
- Benefits:
  - Technology heterogeneity
  - Resilience
  - Scaling
  - Easy of deployment
  - Organizational alignment
  - Composability



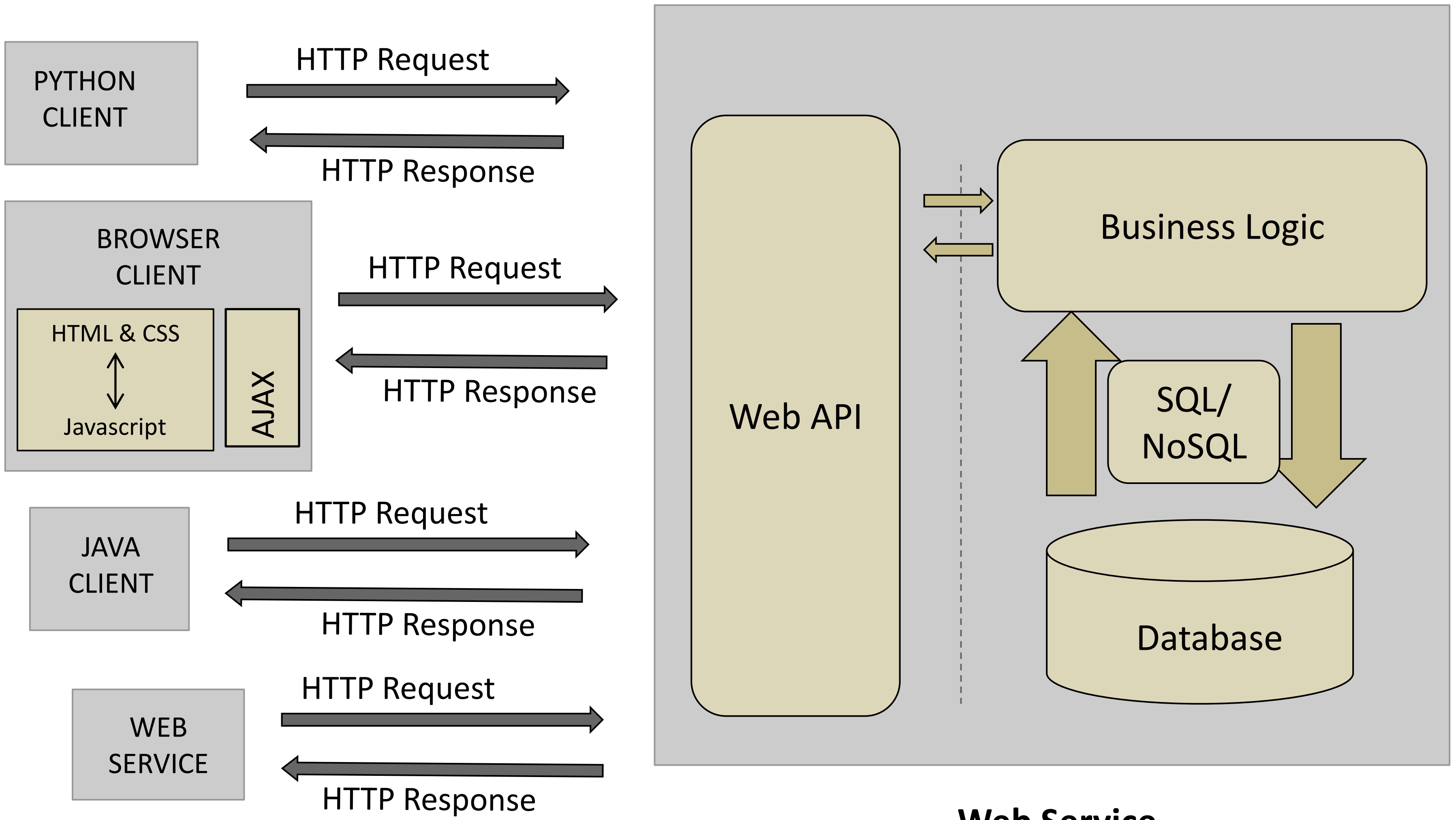


Building microservices. Sam Newman. O'Really Media

APIs and Web APIs

- **API = Application Programming Interfaces**
- Defines how the service functionality is exposed by means of one or more endpoints:
  - Protocol semantics
  - Application semantics
- **Nowadays, web service word is in disuse => We use Web API instead**

Web API



Web Service
Or
Web API

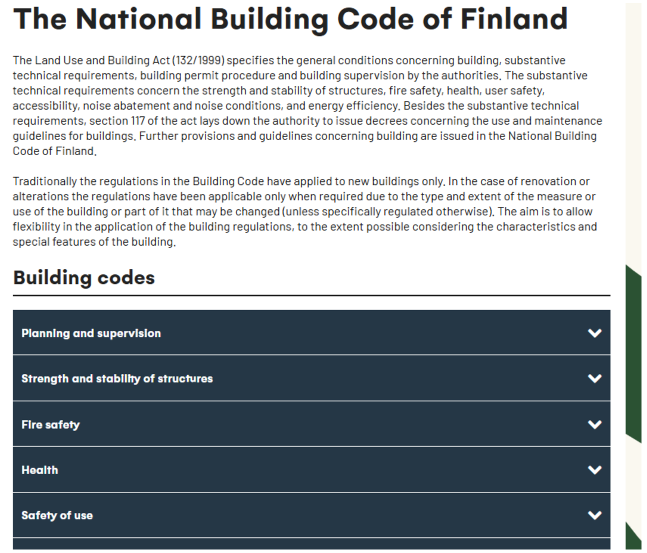
Website vs Web API

- Gist:
  - Github tool that allows sharing code and applications
  - Website at: https://gist.github.com/
  - API at https://developer.github.com/v3/gists/
  - Gist clients: https://gist.github.com/defunkt/370230
    - For instance, Sublime Text client: https://github.com/condemil/Gist

ARCHITECTURAL STYLES

Architectural styles

- Defines a set of design principles and constraints for building and interacting with software systems over a network
  - It provides guidelines for structuring API communication and determines how clients and servers exchange data.
  - Does not define an architecture but requirements for the architecture



The National Building Code of Finland

The Land Use and Building Act (132/1999) specifies the general conditions concerning building, substantive technical requirements, building permit procedure and building supervision by the authorities. The substantive technical requirements concern the strength and stability of structures, fire safety, health, user safety, accessibility, noise abatement and noise conditions, and energy efficiency. Besides the substantive technical requirements, section 117 of the act lays down the authority to issue decrees concerning the use and maintenance guidelines for buildings. Further provisions and guidelines concerning building are issued in the National Building Code of Finland.

Traditionally the regulations in the Building Code have applied to new buildings only. In the case of renovation or alterations the regulations have been applicable only when required due to the type and extent of the measure or use of the building or part of it that may be changed (unless specifically regulated otherwise). The aim is to allow flexibility in the application of the building regulations, to the extent possible considering the characteristics and special features of the building.

Building codes

Architectural styles

REST

- CRUD
- Hypermedia (HATEOAS)

RPC

- SOAP
- GraphQL

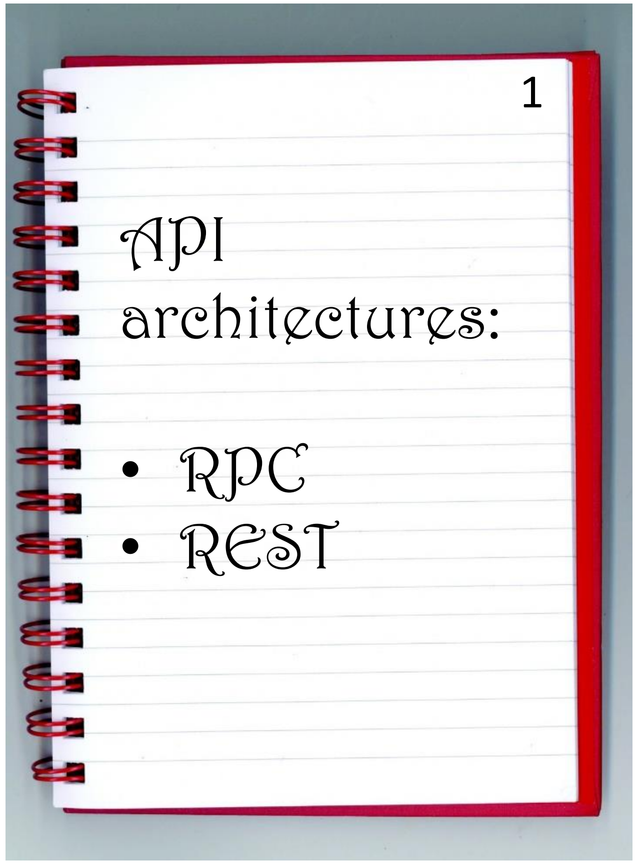
Pub/Sub (Asynchronous Event-Based Collaboration)

REST AND HYPERMEDIA

REST (Representational State Transfer)

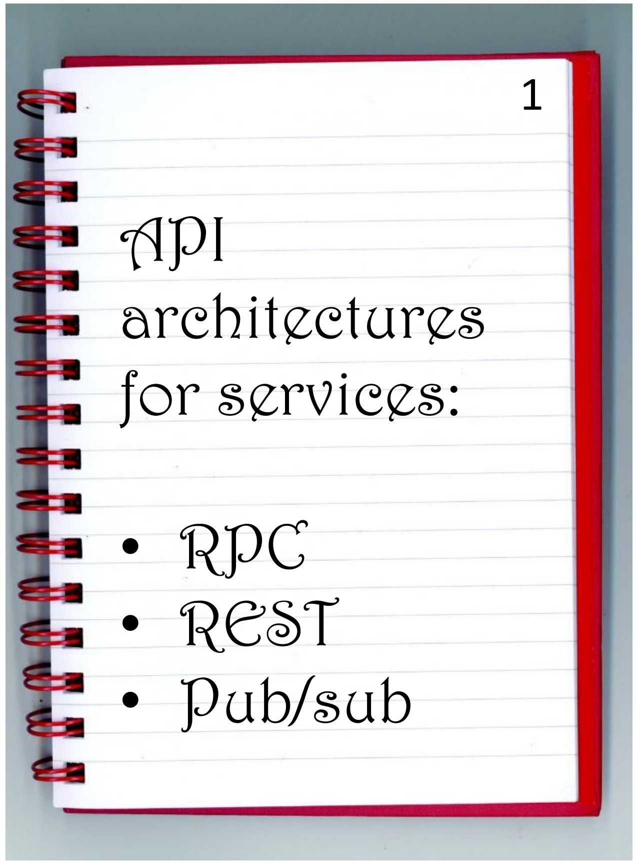
- Architectural style proposed by Roy Thomas Fielding. http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
  - Does not define an architecture but requirements for the architecture
- **Re**presentation
  - Resource-oriented: operates with resources.
    - **Resource**: Any piece of information that can be named. Identified generally by URL
- **S**tate:
  - value of all properties of a resource at the certain moment.
- **T**ransfer: State can be transferred
  - Clients can:
    1) retrieve the state of a resource and
    2) modify the state of the resource
  - UNIFORM interface

REST (Representational State Transfer)



I want to know the content of the page 1 of the notebook.

REST (Representational State Transfer)

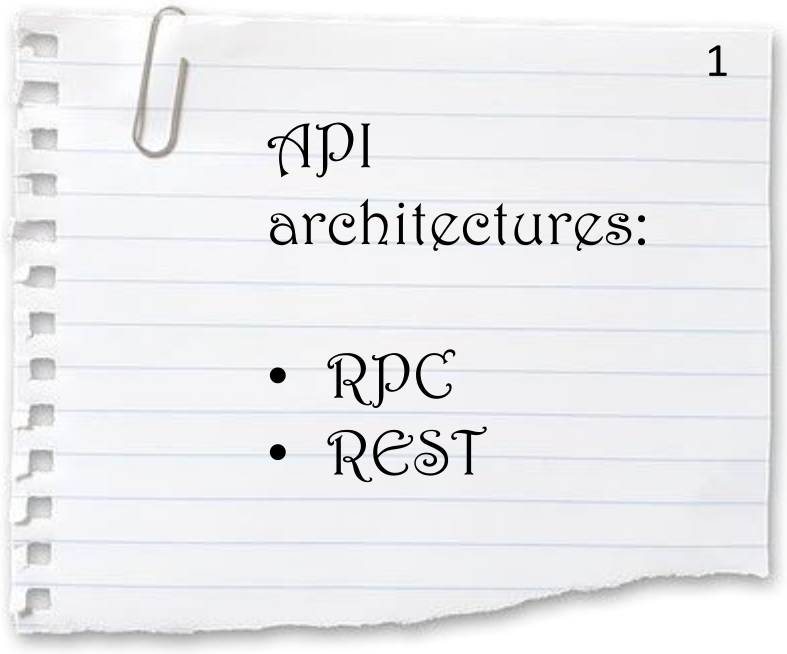

1

API architectures for services:

- RPC
- REST
- Pub/sub

I want to edit the content of the page 1 of the notebook

REST (Representational State Transfer)


1

API architectures:

- RPC
- REST

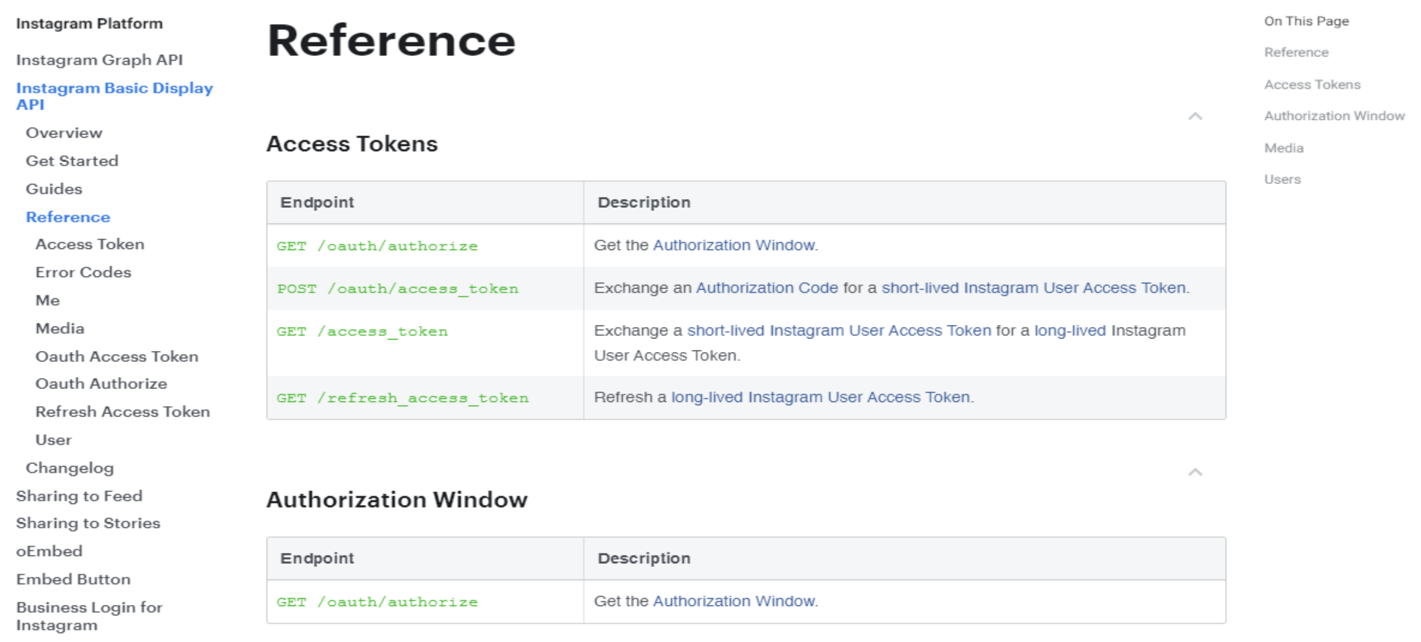
I want to tear off the page 1 of the notebook.

OULUN YLIOPISTO

REST (Representational State Transfer)

I want start writing in a different page of the notebook

Instagram API



Instagram Platform
Instagram Graph API
Instagram Basic Display API
Overview
Get Started
Guides
Reference
Access Token
Error Codes
Me
Media
Oauth Access Token
Oauth Authorize
Refresh Access Token
User
Changelog
Sharing to Feed
Sharing to Stories
oEmbed
Embed Button
Business Login for Instagram

On This Page
Reference
Access Tokens
Authorization Window
Media
Users

Reference

Access Tokens

Endpoint	Description
`GET /oauth/authorize`	Get the Authorization Window.
`POST /oauth/access_token`	Exchange an Authorization Code for a short-lived Instagram User Access Token.
`GET /access_token`	Exchange a short-lived Instagram User Access Token for a long-lived Instagram User Access Token.
`GET /refresh_access_token`	Refresh a long-lived Instagram User Access Token.

Authorization Window

Endpoint	Description
`GET /oauth/authorize`	Get the Authorization Window.

Media

Endpoint	Description
`GET /{media-id}`	Get fields and edges on an image, video, or album.
`GET /{media-id}/children`	Get a list of images and videos on an album.
`GET /{user-id}/media`	Get a list of images, videos, or albums on a User.

https://developers.facebook.com/docs/instagram-api/

e.g. Comment Moderation: https://developers.facebook.com/docs/instagram-api/guides/comment-moderation

REST APIs

CRUD

- Most extended approach.
- Most of Web APIs nowadays
- Not follow strictly REST principles

Hypermedia

- Follows strictly REST principles

Hypermedia driven Web APIs

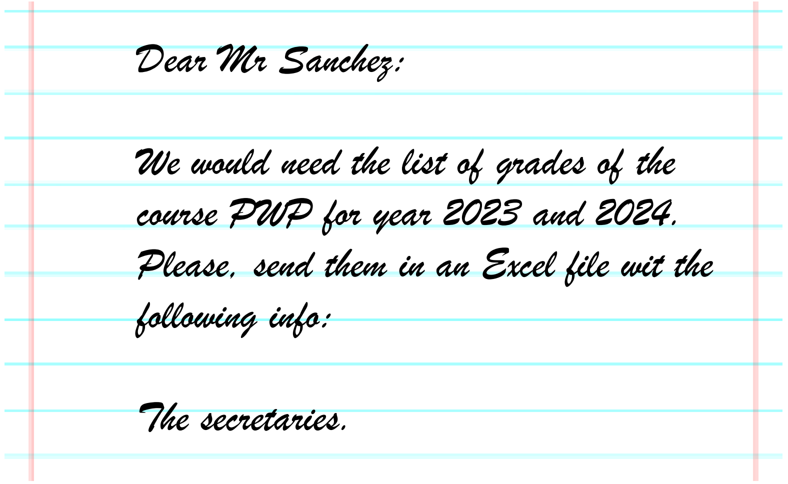
- Follows strictly Fielding dissertation principles.
  - REST APIs must be hypertext driven: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- Uses Hypermedia as the Engine of the Application State
  - Hypermedia describes the actions that you can perform with the resources.
    - Client does not memorize operations nor workflow. Everything is in the messages
- Documentation reduced drastically: messages are documented by themselves
  - A REST API should spend almost all of its descriptive effort in defining the media type used for representing resources and driving application states
- Easier to create general clients
  - Example: RSS and Atom PUB. Multiple clients can read the same RSS feed.

RPC

RPC-style Web APIs

- **RPC: Remote procedure call**
  - A method or subroutine is executed in another address space, without the programmer explicitly encoding the details of the remote interaction.
- An RPC-style Web API accepts an envelope full of data from its client and sends a similar envelope back.
  - The method and the scoping information are kept inside the envelope, or on stickers applied to the envelope.

Air Mail
TOP PRIORITY
To: Iván Sánchez
UBICOMP



Dear Mr Sanchez:

We would need the list of grades of the course PWP for year 2023 and 2024. Please, send them in an Excel file wit the following info:

The secretaries.

RPC-style Web APIs

- Every RPC-style Web API defines a brand new vocabulary: method name, method parameters
- Some examples:
  - XML-RPC
  - SOAP
  - gRPC

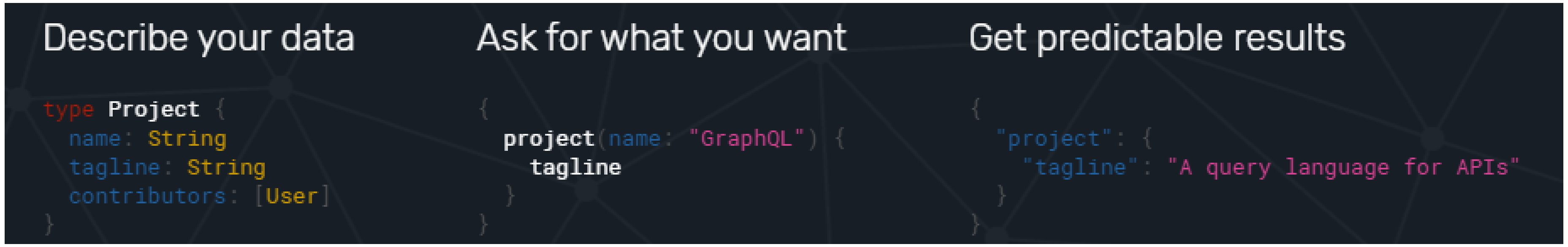
RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

GraphQL

- Mixed of RPC and REST API concepts
  - Created by Facebook.

- GraphQL is a query language APIs, and a server-side runtime for executing queries by using a type system defined for the data.



Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

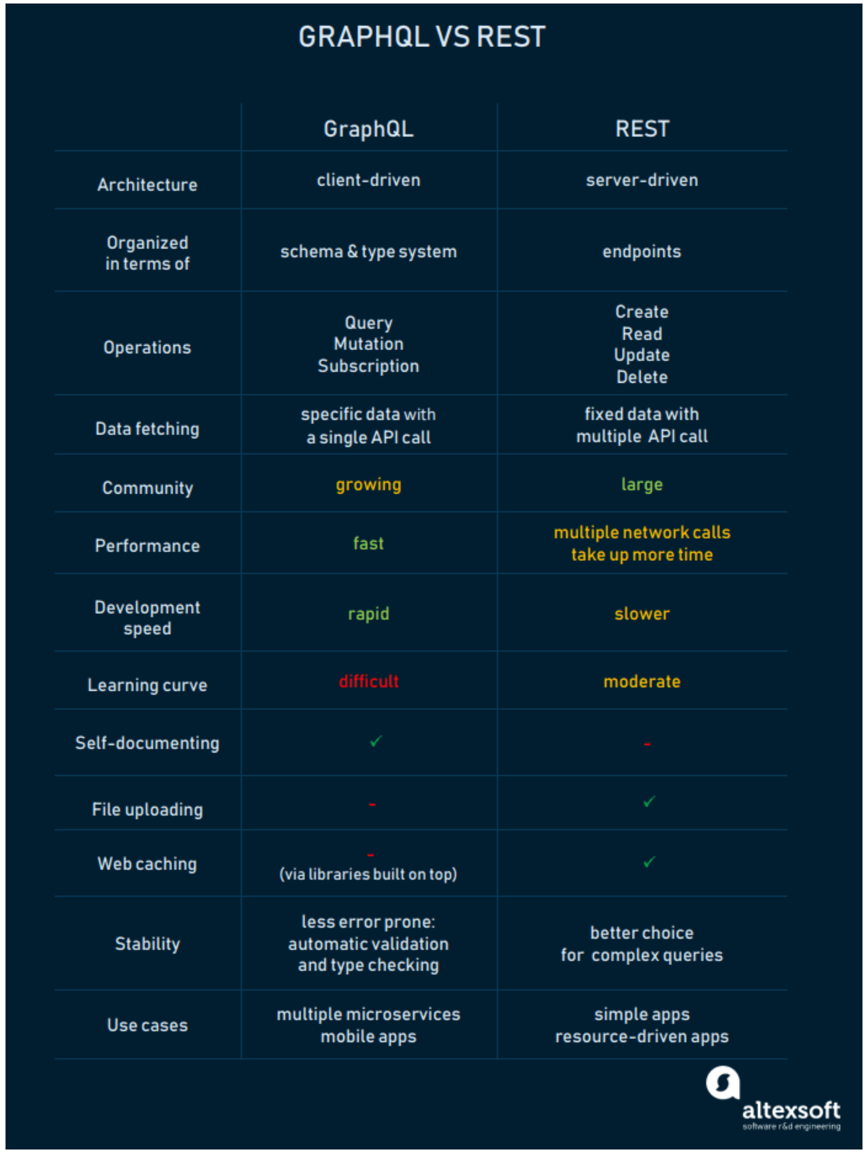
Get predictable results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

https://graphql.org/
https://graphql.org/learn/queries/

GraphQL

https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/



GRAPHQL VS REST

	GraphQL	REST
Architecture	client-driven	server-driven
Organized in terms of	schema & type system	endpoints
Operations	Query Mutation Subscription	Create Read Update Delete
Data fetching	specific data with a single API call	fixed data with multiple API call
Community	growing	large
Performance	fast	multiple network calls take up more time
Development speed	rapid	slower
Learning curve	difficult	moderate
Self-documenting	✓	-
File uploading	-	✓
Web caching	- (via libraries built on top)	✓
Stability	less error prone: automatic validation and type checking	better choice for complex queries
Use cases	multiple microservices mobile apps	simple apps resource-driven apps

altexsoft
software r&d engineering

RPC TECHNOLOGIES EXAMPLES

GRPC

OULUN YLIOPISTO

OLD SOAP WEB SERVICES

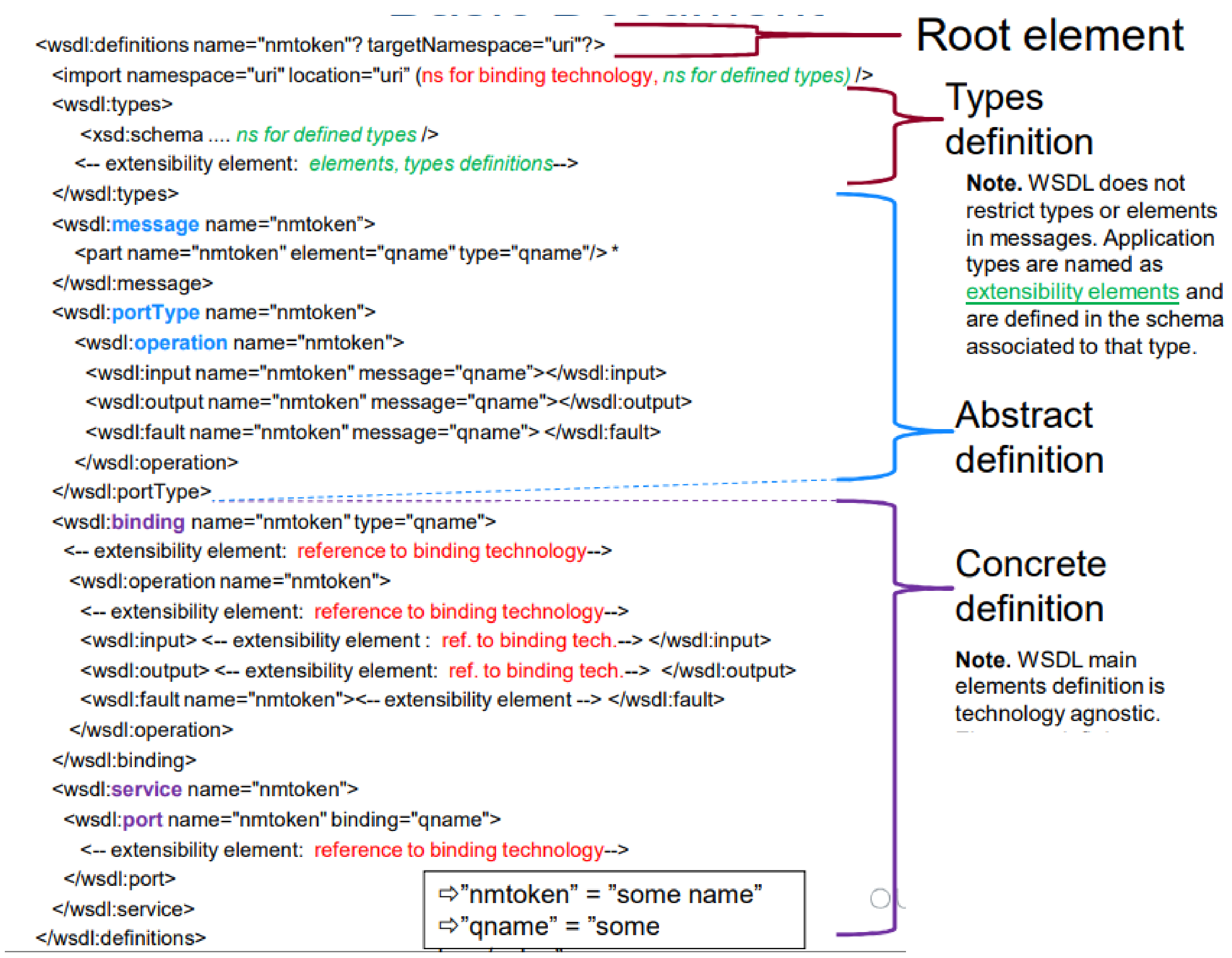
REQUEST

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
<soap:Body>
    <m:getUserFirstName xmlns:m="http://service.forum.rsi.isg.oulu.fi">
        <m:userId>user-3</m:userId>
    </m:getUserFirstName>
</soap:Body>
</soap:Envelope>
```

RESPONSE

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
    <getUserFirstNameResponse xmlns="http://service.forum.rsi.isg.oulu.fi">
        <getUserFirstNameReturn>Axel</getUserFirstNameReturn>
    </getUserFirstNameResponse>
</soap:Body>
</soap:Envelope>
```

WSDL



```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
  <import namespace="uri" location="uri" (ns for binding technology, ns for defined types) />
  <wsdl:types>
      <xsd:schema .... ns for defined types />
     <-- extensibility element:  elements, types definitions-->
  </wsdl:types>
  <wsdl:message name="nmtoken">
     <part name="nmtoken" element="qname" type="qname"/> *
  </wsdl:message>
  <wsdl:portType name="nmtoken">
     <wsdl:operation name="nmtoken">
       <wsdl:input name="nmtoken" message="qname"></wsdl:input>
       <wsdl:output name="nmtoken" message="qname"></wsdl:output>
       <wsdl:fault name="nmtoken" message="qname"> </wsdl:fault>
     </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="nmtoken" type="qname">
    <-- extensibility element:  reference to binding technology-->
    <wsdl:operation name="nmtoken">
      <-- extensibility element:  reference to binding technology-->
      <wsdl:input> <-- extensibility element :  ref. to binding tech.--> </wsdl:input>
      <wsdl:output> <-- extensibility element:  ref. to binding tech.-->  </wsdl:output>
      <wsdl:fault name="nmtoken"><-- extensibility element --> </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="nmtoken">
    <wsdl:port name="nmtoken" binding="qname">
      <-- extensibility element:  reference to binding technology-->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

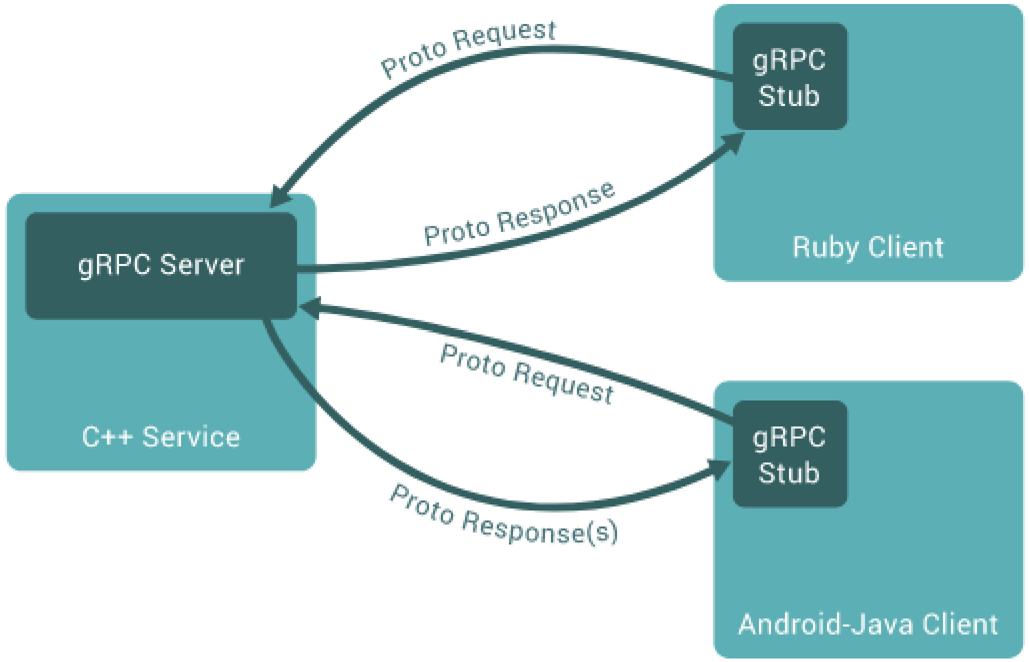
⇨"nmtoken" = "some name"
⇨"qname" = "some

Note. WSDL does not restrict types or elements in messages. Application types are named as extensibility elements and are defined in the schema associated to that type.

Note. WSDL main elements definition is technology agnostic.

GRPC intro

- Based on Protocol Buffers
  - Google technology for serializing data structures



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

GRPC. Proto

```
syntax = "proto3";

enum BookCategory {
    MYSTERY = 0;
    SCIENCE_FICTION = 1;
    SELF_HELP = 2;
}

message RecommendationRequest {
    int32 user_id = 1;
    BookCategory category = 2;
    int32 max_results = 3;
}

message BookRecommendation {
    int32 id = 1;
    string title = 2;
}

message RecommendationResponse {
    repeated BookRecommendation recommendations = 1;
}

service Recommendations {
    rpc Recommend (RecommendationRequest) returns (RecommendationResponse);
}
```

Compiled to programming language structure (objects)

https://realpython.com/python-microservices-grpc/

GRPC. Server

```
class RecommendationService(recommendations_pb2_grpc.RecommendationsServicer):
    def Recommend(self, request, context):
        if request.category not in books_by_category:
            context.abort(grpc.StatusCode.NOT_FOUND, "Category not found")

        books_for_category = books_by_category[request.category]
        num_results = min(request.max_results, len(books_for_category))
        books_to_recommend = random.sample(
            books_for_category, num_results
        )

        return RecommendationResponse(recommendations=books_to_recommend)
```

https://realpython.com/python-microservices-grpc/

GRPC. Client

```
>>> channel = grpc.insecure_channel("localhost:50051")
>>> client = RecommendationsStub(channel)
>>> request = RecommendationRequest(
...     user_id=1, category=BookCategory.SCIENCE_FICTION, max_results=3
... )
>>> client.Recommend(request)
```

https://realpython.com/python-microservices-grpc/

Types of service methods

- **Unary:** the client sends a single request to the server and gets a single response back, just like a normal function call.
- **Server streaming RPCs:** client sends a request to the server and gets a stream to read a sequence of messages back.
  - The client reads from the returned stream until there are no more messages.
  - gRPC guarantees message ordering within an individual RPC call.
- **Client streaming RPCs:** the client writes a sequence of messages and sends them to the server, again using a provided stream
  - Once the client has finished writing the messages, it waits for the server to read them and return its response.
  - gRPC guarantees message ordering within an individual RPC call.
- **Bidirectional streamming RPCs:** Mix of two previous.

PUB/SUB

Pub / Sub

- Some services emit events (user entered the room)
- Some services are subscribed to those events
  - When the publisher publish the events the subscriber receives the event
- Generally a **broker** is in charge of coordination:
  - Producers publish event to the broker
  - Broker handle subscriptions and inform when an event arrives

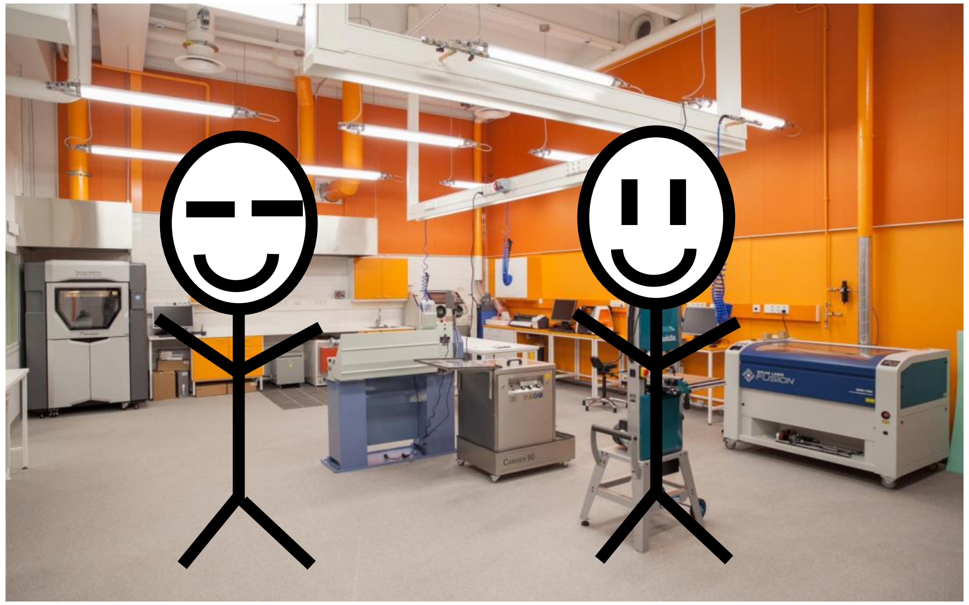
- Complex solution BUT creates effective loosely-couple solutions.

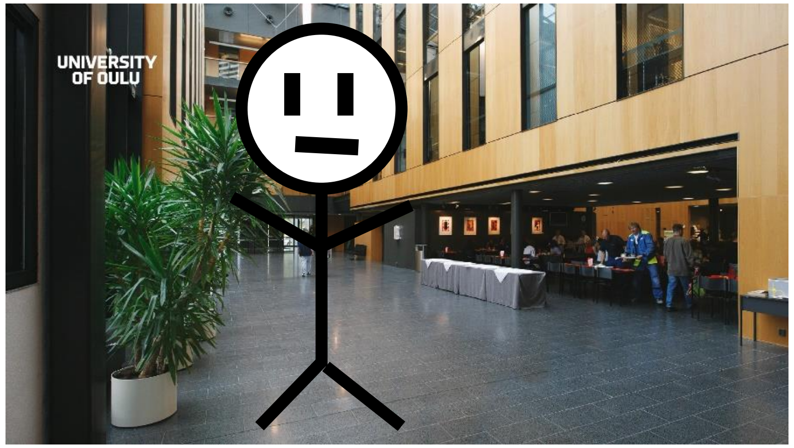
- E.g. mqtt, rabitmq …

Pub / Sub

Let me know when a student use the 3D printer
Me too
OK
UNIVERSITY OF OULU

Pub / Sub





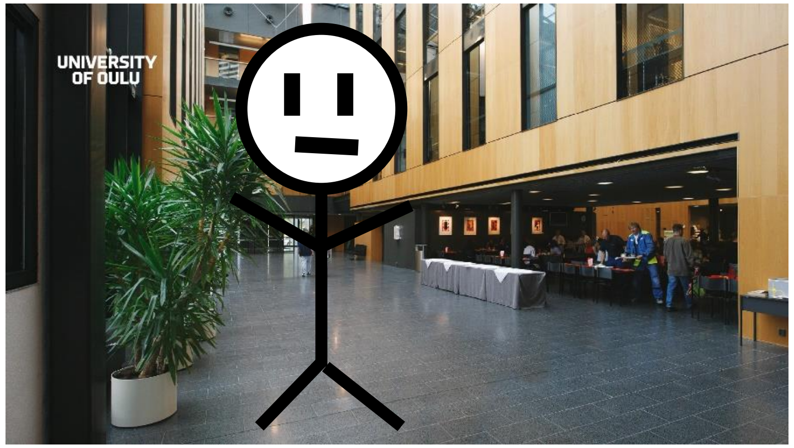
Pub / Sub

Instructor 1, Matti wants to 3D print right now

UNIVERSITY OF OULU

Pub / Sub

Instructor 2, Matti wants to 3D print right now


UNIVERSITY OF OULU

The World Wide Web

What is the World Wide Web?

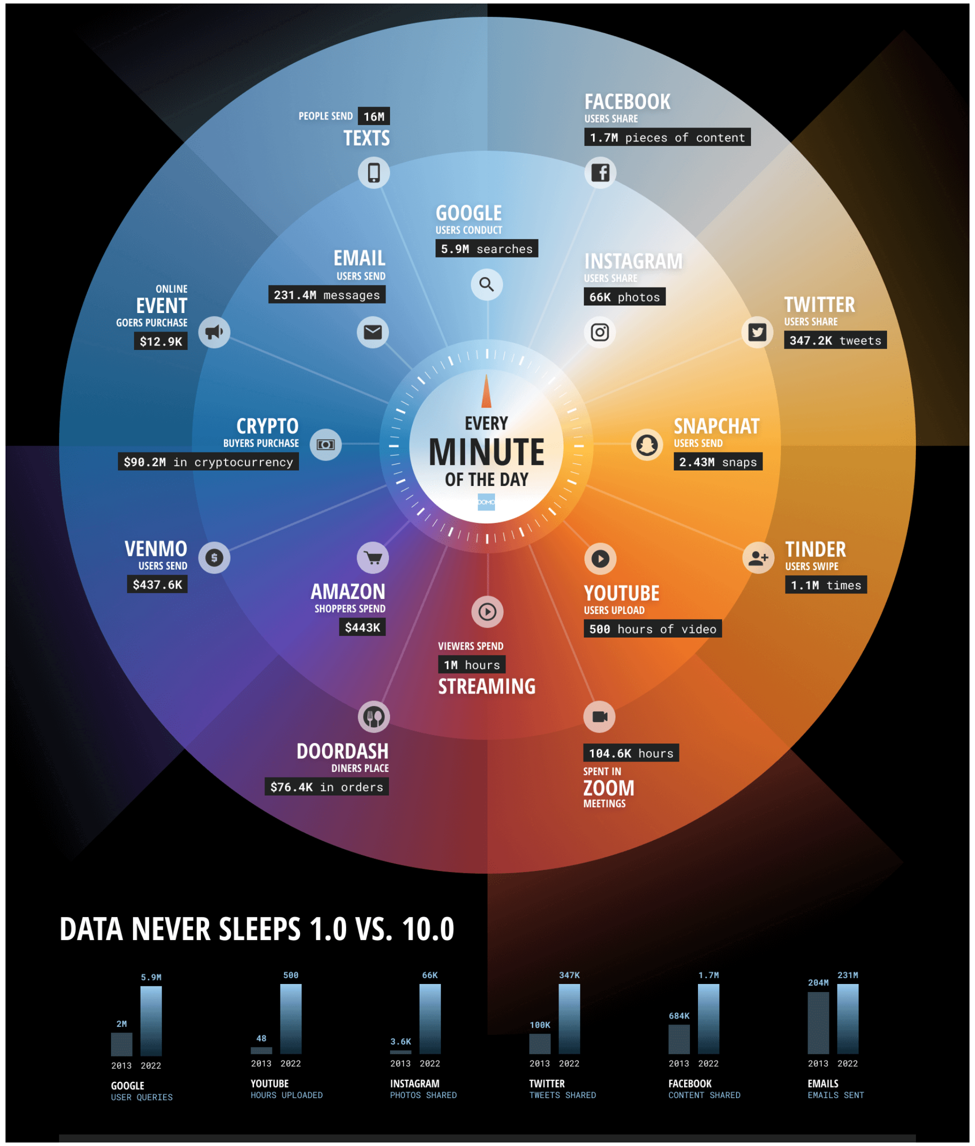
https://www.youtube.com/watch?v=OM6XIICm_qo&start=18&end=190&autoplay=1

What is the World Wide Web?

Goal: Distribute data

- Human consumption (H2M)
- Hypertext
- Uniform API and technologies
- Single client (Web browser)

World Wide Web success. Scalability



Web is distributed

Web is massively decoupled

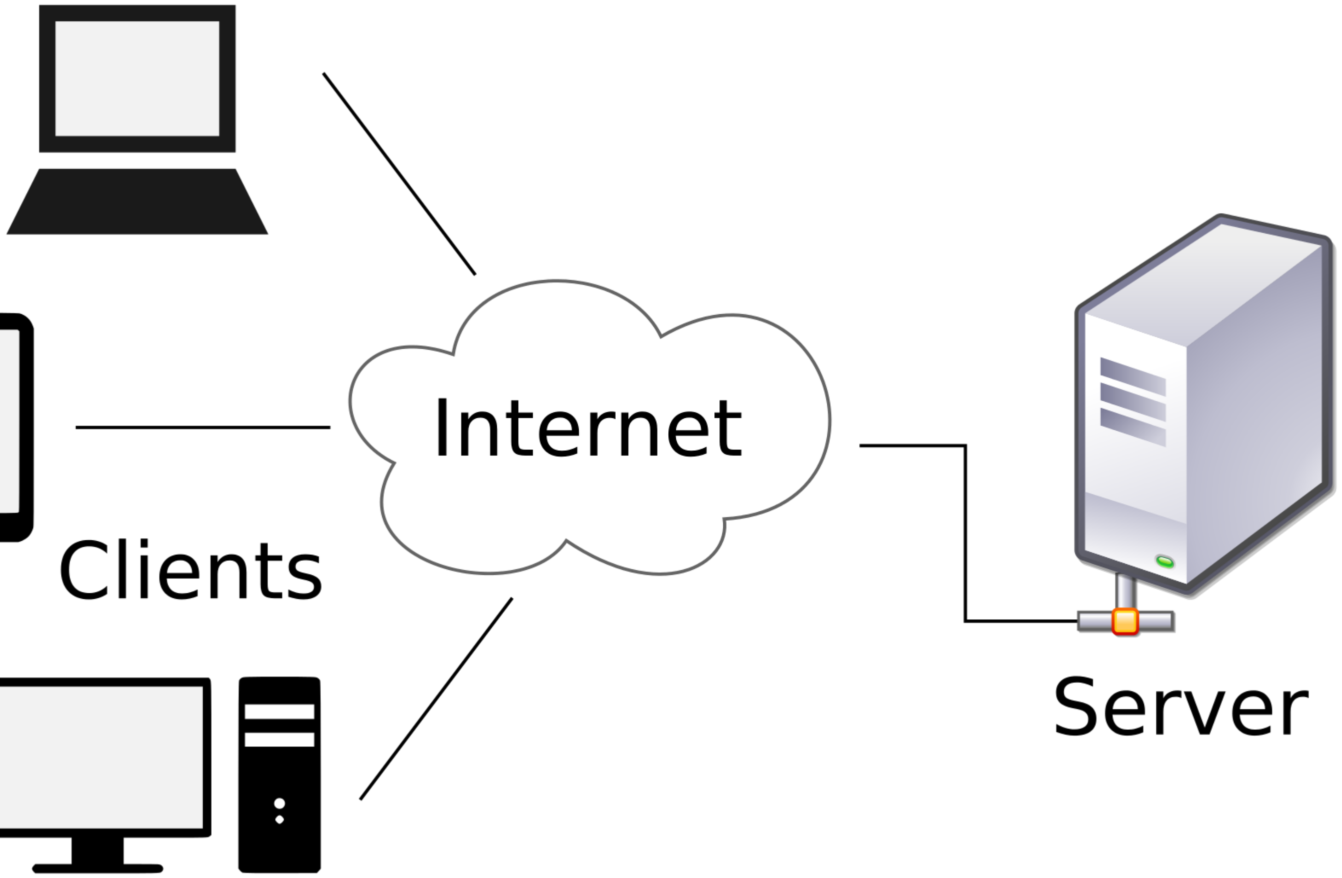
Web is dynamic

Source (2024) https://www.domo.com/data-never-sleeps

TECHNOLOGIES FOR THE WWW

- Backend: Business logic + data storage (databases)
- Transport protocol: HTTP
- Data serialization languages
- Clients

Client server model



BACKEND

Backend

- Stores application data persistently
  - **DATABASE**
- Defines how to process request from the client and process the data according to the requests coming from the client
  - **BUSINESS LOGIC**
- Expose the data using a defined API

DATABASES

Definition

- Databases emerged to solve challenges of storing and managing huge amounts of data
- A database:
  - is a data structure
  - stores organized information
  - can be easily accessed, managed and updated
- DBMS (Database Managing System) is the software that allows creating, managing and storing database structures.
  - Responsible for data integrity, recovery and access
  - Provides a way for extract or modify the data
- There are different ways to model the data in the database
  - Lately divided into relational models and non-relational models

ACID properties

- Atomicity
  - Each transaction is atomic.
  - If one part of the transaction fails the whole transaction fails and the database is not modified.
- Consistency
  - Databases moves from one valid state to another valid state in each transaction.
  - A state is valid if meets all the constraints
- Isolation
  - Concurrent access is processed as serial access.
  - Not completed transactions might not be visible to other users
- Durability
  - Once a transaction is committed it remains in the db.

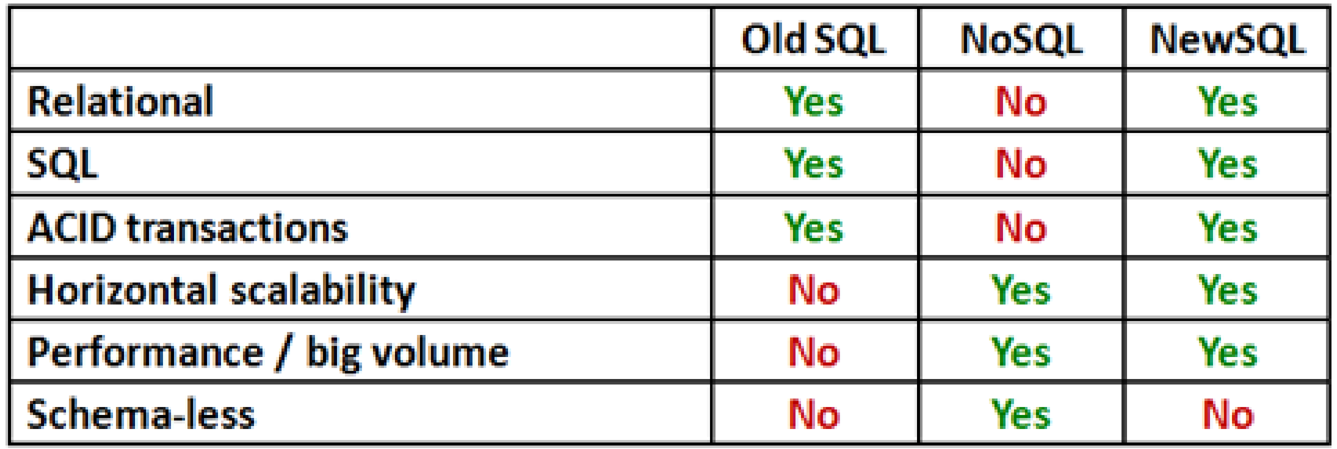
Relational – Non-relational

- Relational:
  - Database model developed by E.F. Codd in 1970
    - Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks"
  - Data is represented in terms of tuples (rows), grouped into relations (tables) that can be linked with each other.
  - Developed almost in parallel with SQL language
- Non-Relational:
  - Sometimes miscalled Non SQL databases
  - Umbrella that gathers different databases that are not relational.
  - Data is not organized in related tables.
    - Some store objects, some store key-value pairs, some store documents
  - More flexible and scalable

RDBMS Concepts

- **CRUD**
  - Databases stores data persistently
  - There are four basic functions to manage persistent data:
    - **C**reate
    - **R**ead
    - **U**pdate
    - **D**elete
- **ORM (Object relational mapping)**
  - To access a relational database from an object oriented language context (PHP, Python, Java…)
    - interface translating relational logic to objects logic is needed.
    - Such interface is called **Object-relational mapping** (**ORM**, **O/RM**, and **O/R mapping**).

SQL vs NoSQL vs NewSQL



	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

Examples - Relational

- Relational databases are still the most commonly used.
- Relational databases are mainly composed by tables.
- A table is formed by zero (empty) or more rows.
- A row consists of one or more fields
  - Each has a certain datatype. (columns)

FirstName	Surname	PersonalId
John	Smith	3321
Jack	Johnson	4352
Mary	Smith	9807

- Some examples are: PostgreSQL, MySQL, SQLite

Examples – Non-relational

– MongoDB
- Scalable, open source database
- JSON based data store: BSON
- Document-oriented database
  – Database formed by Collections of Documents
- Example of MongoDB document:

```
{
  name: “jim”,
  surname: “smith”,
  grade: 3
}
```

- Example of MongoDB query:

```
db.students.find({grade:{$gt:3}});
```

TRANSPORT PROTOCOL: HTTP

HTTP

- **The Hypertext Transfer Protocol** (HTTP):

> *”an **application-level protocol** for distributed, collaborative, hypermedia information systems”*
>
> RFC 2616 (http://www.faqs.org/rfcs/rfc2616.html)

  - HTTP communication usually takes place over TCP/IP connections.
  - Most used **application** protocol in the World Wide Web.
  - Also used as a transport protocol for other application protocols, such as SOAP, XML-RPC …
- HTTP allows bidirectional transfer of resources representations between client and server.
  - **Resource**: network data object identified by a URI

HTTP Request parts

- HTTP request example to http://www.cse.oulu.fi

The HTTP method. Here, the client (web browser) is trying to GET some information from the server (www.cse.oulu.fi).

The path In this example the path points to the root of the host (just /)

REQUEST LINE

```
GET  /  HTTP/1.1
Keep-Alive: 300
Connection: keep-alive
Host: www.cse.oulu.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The request headers Since the request does not have entity, it only contains general and request specific headers.

The entity-body This particular request has no entity body, which means the envelope is empty! This is typical for a GET request, where all the information needed to complete the request is in the path and the headers.

HTTP Response parts

- Response Example: http://www.cse.oulu.fi

The HTTP response code. In this case the GET operation must have succeeded, since the response code is 200 (“OK”).

STATUS LINE

The response headers: general, response and entitity headers

```
HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Date:  Wed, 05 Oct 2011 17:26:03 GMT
Server: Apache/2.2.3 (CentOS)
Vary: Cookie,User-Agent,Accept-Language
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN“ "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<meta name="robots" content="index,follow">
<MainPage - Department of Computer Science and Engineering</title>
…
```

The entity-body. In this case, the entity body is a HTML document representing a web page.

HTTP Methods

Defined in RFC2616

Method	Description
GET	Returns the resource representation
HEAD	Identical to GET except that the server returns only headers information in the response
PUT	Changes the state of the resource Creates a new resource when the URL is known
POST	Create subordinate resources (no URL known beforehand) Appends information to the current resource state
DELETE	Removes a resource from the server

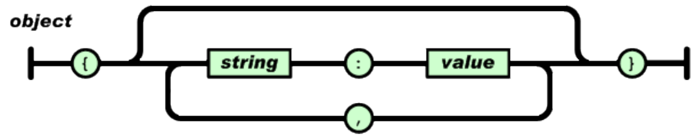
DATA SERIALIZATION LANGUAGES

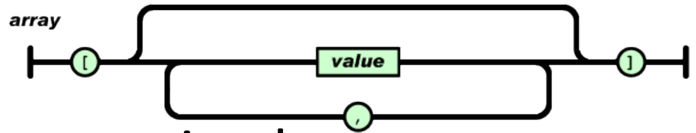
JSON and XML

- Formats used for representing data that are heavily used to share data among heterogeneous peers
  - Text format (not binary)
  - Language independant
- Although the two of them can be used for M2M and H2M
  - XML is more human readable oriented
  - JSON is more machine readable oriented
- In the Programmable Web, they are mainly used for data exchange, although the may be used also for data storage.

JSON

- JavaScript Object Notation
- Based on a subset of the JavaScript Language
- Built on two structures:
  - A collection of name/value pairs
  - An ordered list of values
- These structures can be mapped to structures in almost any programming language
- Example





```
{"widget": {
          "debug": "on",
          "window": {
                    "title": "Sample Konfabulator Widget",
                    "name": "main_window",
                    "width": 500,
                    "height": 500 }
}}
```

http://www.json.org

XML

- Extensible Markup Language
  - Markup language: system for annotating a document,
- First intended for data publishing
- Markup based in tags:

```
<tag>content</tag>
```

- More info
  - Appendix 1: App1_XML_Basics
  - http://www.w3.org/XML/
- Example

```
<widget>
          <debug>on</debug>
          <window title="Sample Konfabulator Widget">
                    <name>main_window</name>
                    <width>500</width>
                    <height>500</height>
          </window>
</widget>
```

(http://www.json.org)

Hypermedia

- Techniques to integrate content in multiple formats (text, image, audio, video…) in a way that all content is connected and accessible to the user.

"*Hypertext […]* the *simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions. Machines can follow links when they understand the data format and the relations type*"

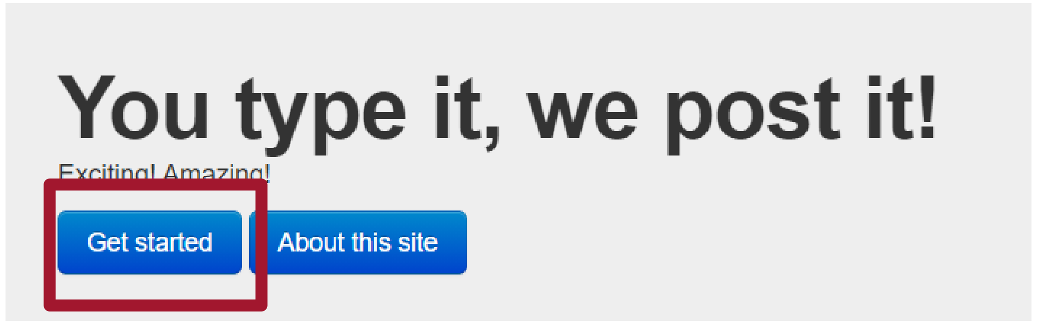
Roy Fielding, "A little REST and Relaxation*"

- Hypermedia
  - **Data**
  - **Hypermedia controls**. Indicates what actions could I do next, what are the target resource to perform the action (link) and how can I perform those actions (http method / response).

* http://www.slideshare.net/royfielding/a-little-rest-and-relaxation

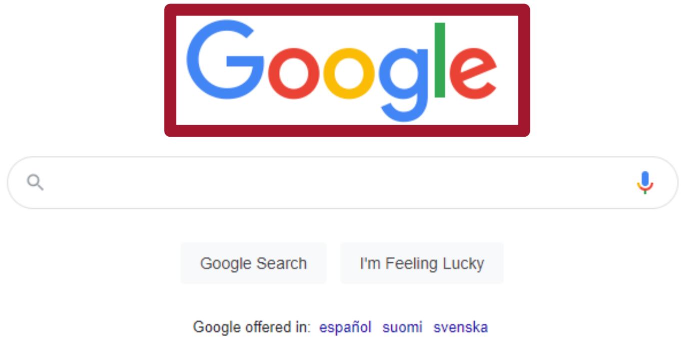
Hypermedia (HTML)

```
<a href="http://www.youtypeitwepostit.com/messages/">
        Get started
</a>
```


You type it, we post it!

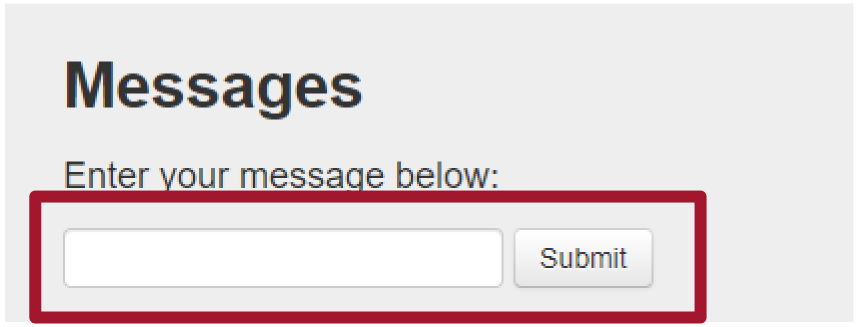
Exciting! Amazing!

```
<img alt="Google" height="92" id="hplogo"
src="/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png" rel="icon"/>
```



Hypermedia (HTML)

```
<form action="http://www.youtypeitwepostit.com/messages" method="post">
        <input type="text" name="message" value="" required="true" />
        <input type="submit" value="Post" />
</form>
```


Messages

Enter your message below:

Hypermedia (Collection+JSON)

Mime type: application/vnd.collection+json
Link: http://amundsen.com/media-types/collection/

```
{ "collection":
    {
        "version" : "1.0",
        "href" : "http://www.youtypeitwepostit.com/api/",
        "items" : [
          { "href" : "http://www.youtypeitwepostit.com/api/messages/21818525390699506",
            "data" : [
                { "name" : "text", "value" : "Test." },
                { "name" : "date_posted", "value" : "2013-04-22T05:33:58.930Z" }
            ],
            "links" : []
           },

          { "href" : "http://www.youtypeitwepostit.com/api/messages/3689331521745771",
            "data" : [
                { "name" : "text", "value" : "Hello." },
              { "name" : "date_posted", "value" : "2013-04-20T12:55:59.685Z" }
            ],
            "links" : []
           },
        "template" : {
            "data" : [
                {"prompt" : "Text of message", "name" : "text", "value" : ""}
            ]
         }
    }
}
```

LIST OF HYPERMEDIA FORMATS IN APPENDIX 3: Hypermedia formats

CLIENTS

Web browser. An Human Driven client.

- A web browser is the client for ALL websites and web applications.

- **TECHNOLOGIES:**
  - **HTML**-> Markup language which defines the content to be rendered by the browser
  - **CSS**-> Style sheet language used for describing the look and formatting of a document
  - **JAVASCRIPT->** Scripting language that listen for events triggered by the users, the network or the host system and execute predefined actions.
  - **AJAX**-> A set of techniques based on Javascript which enable asynchronous interaction between a web browser and a server
  - **WebSocket**-> Computer communication protocol over TCP that provides full-duplex communication. It enables for instance, pub/sub.

Types of clients

- Human driven clients
  - Decisions made by humans. IMPORTANT: how to represent information to humans
- Crawlers
  - It starts following all links iteratively from certain web, executing an algorithm for each link followed
  - E.g. Google
- Monitors
  - Checks the state of a resource periodically
  - E.g. RSS aggregator
- Scripts
  - Simulate an human repeating a determined set of actions (eg. Accessing sequentially a list of links).
- Agents
  - Try to emulate humans who are actively engaged with a problem. Looks to representation and take autonomous decisions based on states.

PROGRAMMABLE WEB

What about current Web APIs (RPC or CRUD)?

- Need excessive documentation
  - Exhaustive description of required protocol: HTTP methods, URLs …
- Integrating a new API inevitably requires writing custom software
  - Similar applications required totally different clients
- When an application API changes, clients break and have to be fixed
  - For instance a change in the object model in the server or the URL structure => change in the client.
- Clients need to store a lot of information
  - Protocol semantics
  - Application semantics

Web vs Programmable Web

- The **Programmable Web** use the same technologies and communication protocols as the WWW in order to cope with current problems.
- Current differences
  - The data is not delivered necessarily for human consumption (M2M)
  - Nowadays an **specific client** is needed per application at least until we solve the problems derivated from the **semantic challenge**
  - A client can be implemented using any programming language
    - Data is encapsulated and transmitted using any serialization languages such as **JSON, XML, HTML, YAML**