Web and Cloud Computing
Week 2 - Web Architecture

2615 - Master in Business Analytics - 23/24 T3

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

NOVA SCHOOL OF BUSINESS & ECONOMICS

Demo: The First “Hello, World!” web-server

Demo: The First Web-client Application

Writing an HTML page from scratch to report the weather, and …

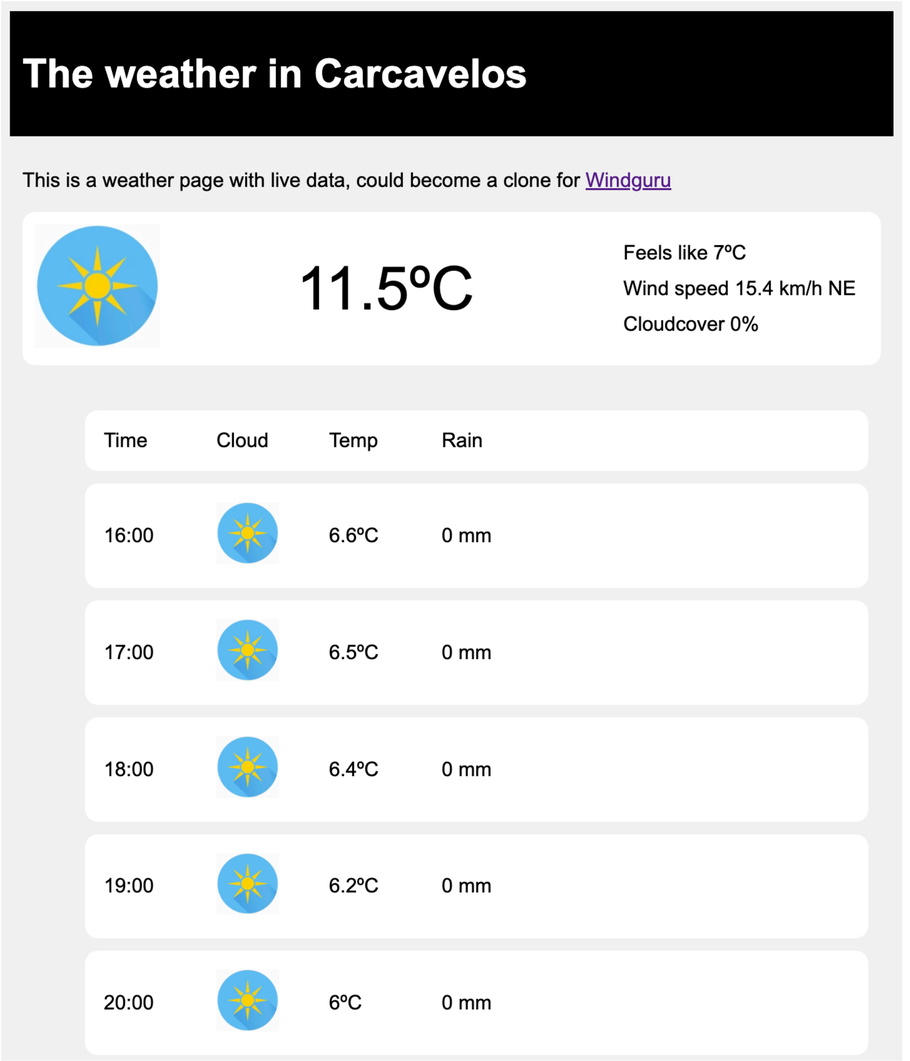
load it from the API

www. = subdomain

Domain name

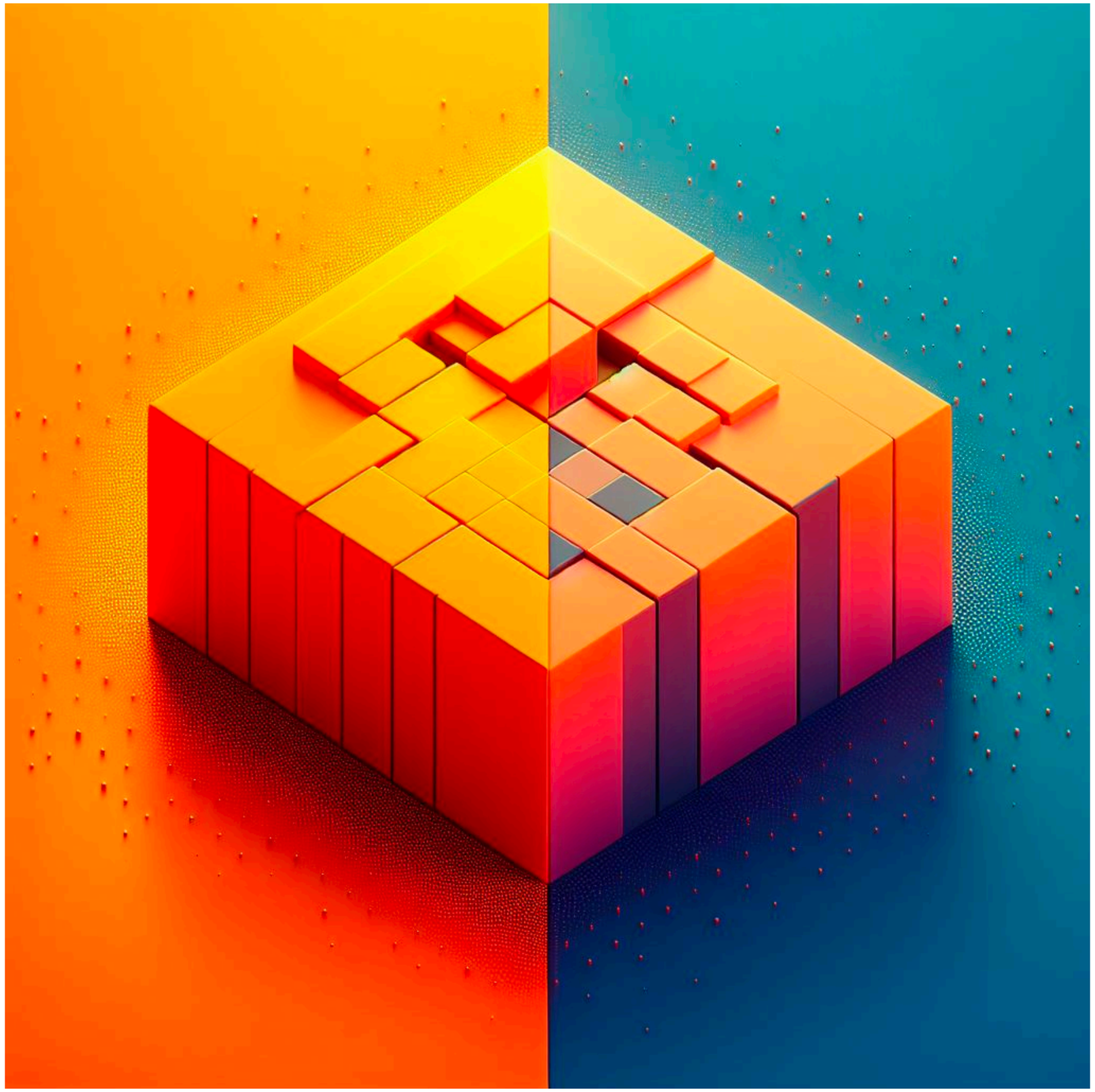
https://open-meteo.com/ path /? query

protocol



Break

Software Architecture



Software Architecture

What is software architecture?

Why is it important?

What are its benefits and challenges?

Sources:
- Software Architecture in practice
- https://martinfowler.com/architecture/

Software Architecture is…

"The way the highest level components are wired together" Martin Fowler

The fundamental structure and design of a software system, that defines its components, relations, properties, and quality attributes.

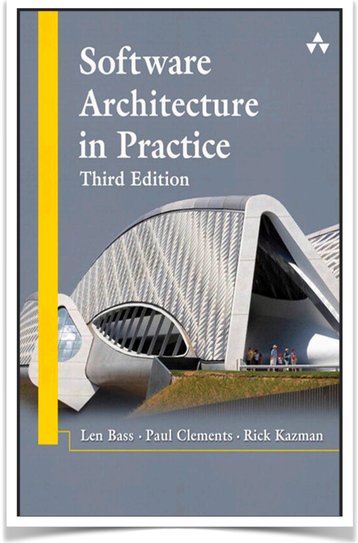
= helicopter view of the system

Software Architecture is…

1. Functionalities (modules)
2. Parts (components & connectors)
3. Resources (allocation)

The organisation of structures and views:

- Module structures (code or data units): represent functionality
- Component/Connector structures
  - Components: servers, clients, filters, etc.
  - Connectors: call-return, pipes, sync operators, etc.
- Allocation structures: resources that support components and connectors.



Software Architecture is essential because…

It affects:

now
- **Performance:** identify bottlenecks, isolate and optimise locally.
  → efficient (streamlined)

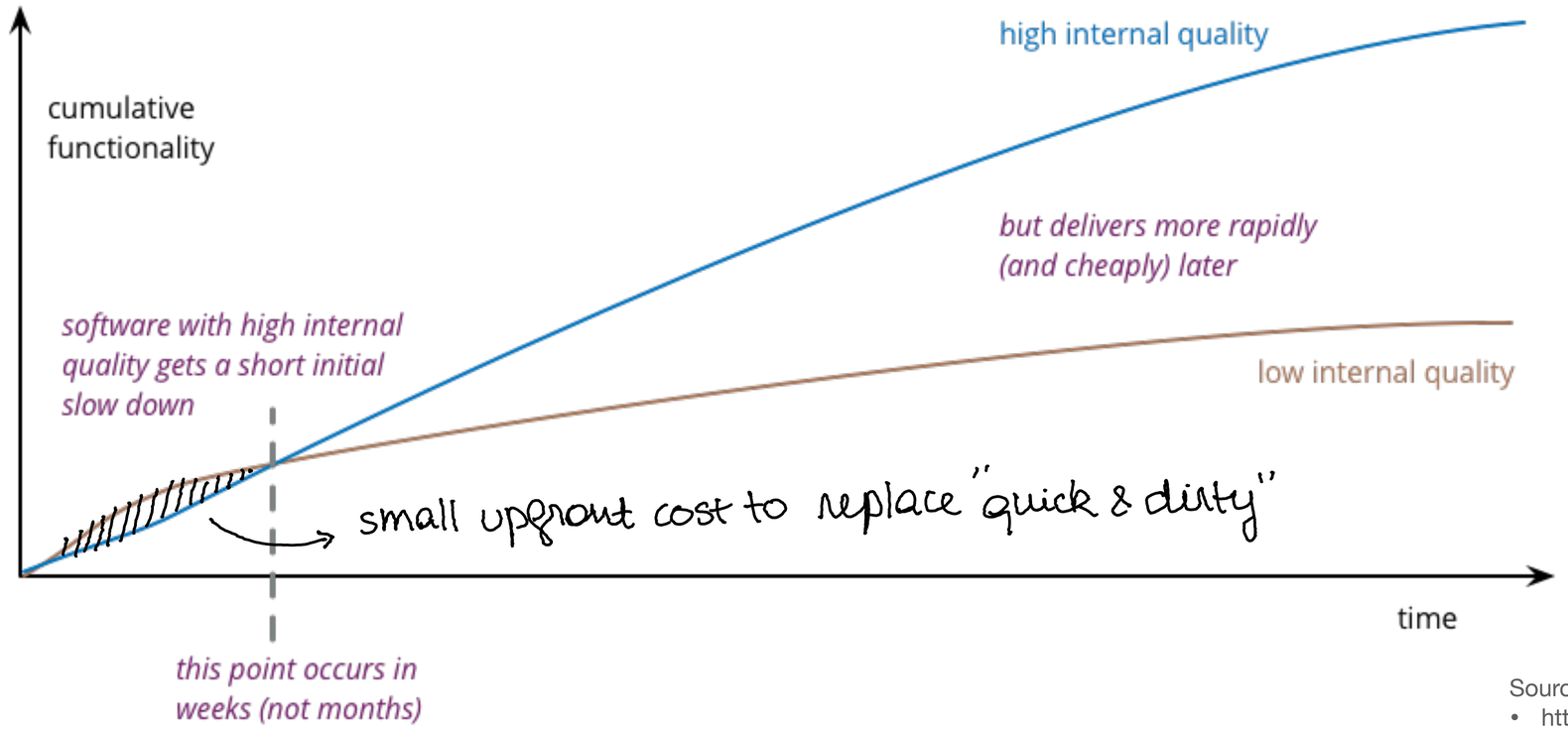
future
- **Maintainability:** localised bugs, less impact on other components.
  → Robustness and trouble shooting
- **Scalability:** replication of components, not the system.
  → ↑ modularity (more flexible)
- **Evolution:** add more components and improve the “hub”.
  → adapt easily (monolithic to modular)

Architectural patterns…

- Module patterns: ∩ functions
  - Layered (a > b > c)
  - Shared-data
  - Client-server (specific accesses)
  - Event-driven (webhooks etc.)
- Allocation patterns: ∩ resources
  - Multi-tier, Services, μServices (deployment) — data logic & presentat. — lots of APIs to embed
  - Competence center (human resources) clear assignement

Software Architecture pays off…

An organised system is much easier to extend, maintain, and reason about. However, the client does not see the architecture, it's internal.



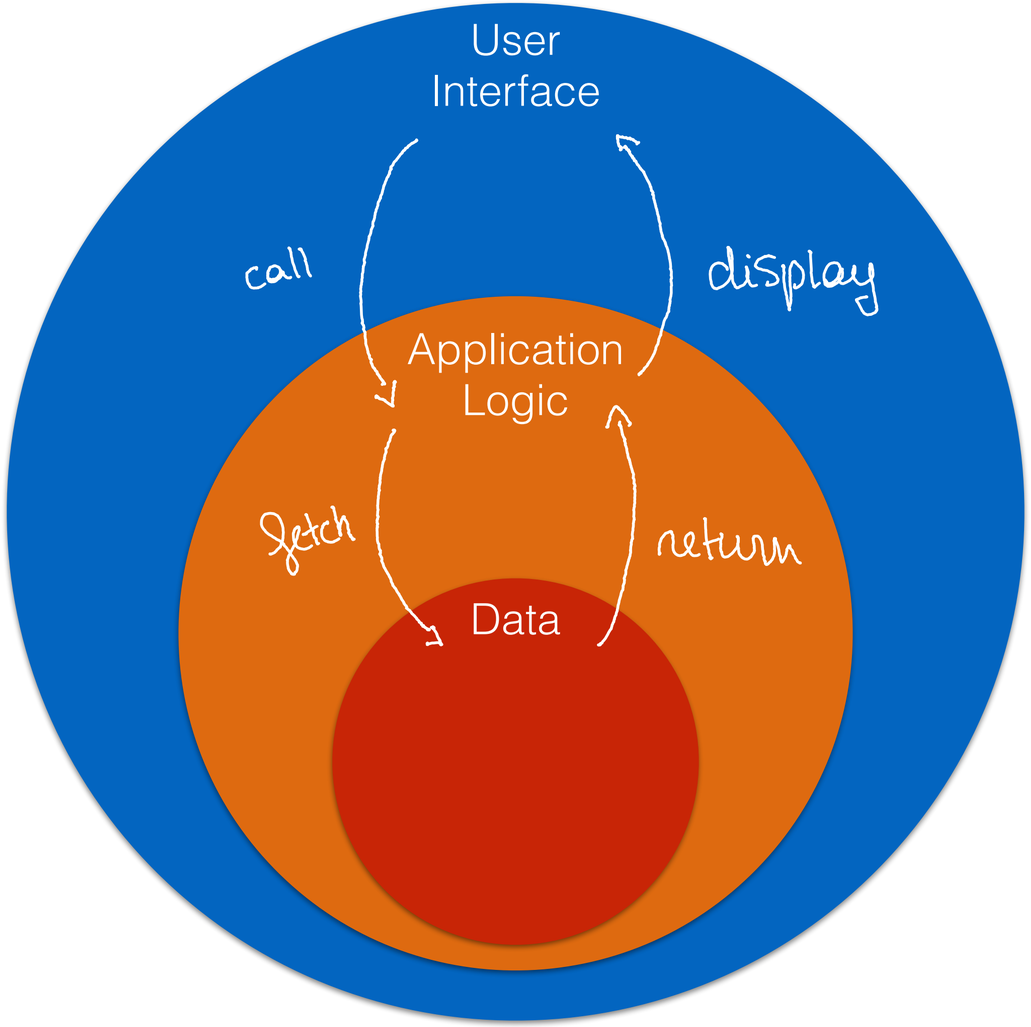
Sources:
- https://martinfowler.com/architecture/

Monoliths

Monoliths (the traditional way…)

Big data-centric systems or applications covering multiple concerns.

One server, one database, many clients (Web? Mobile?)

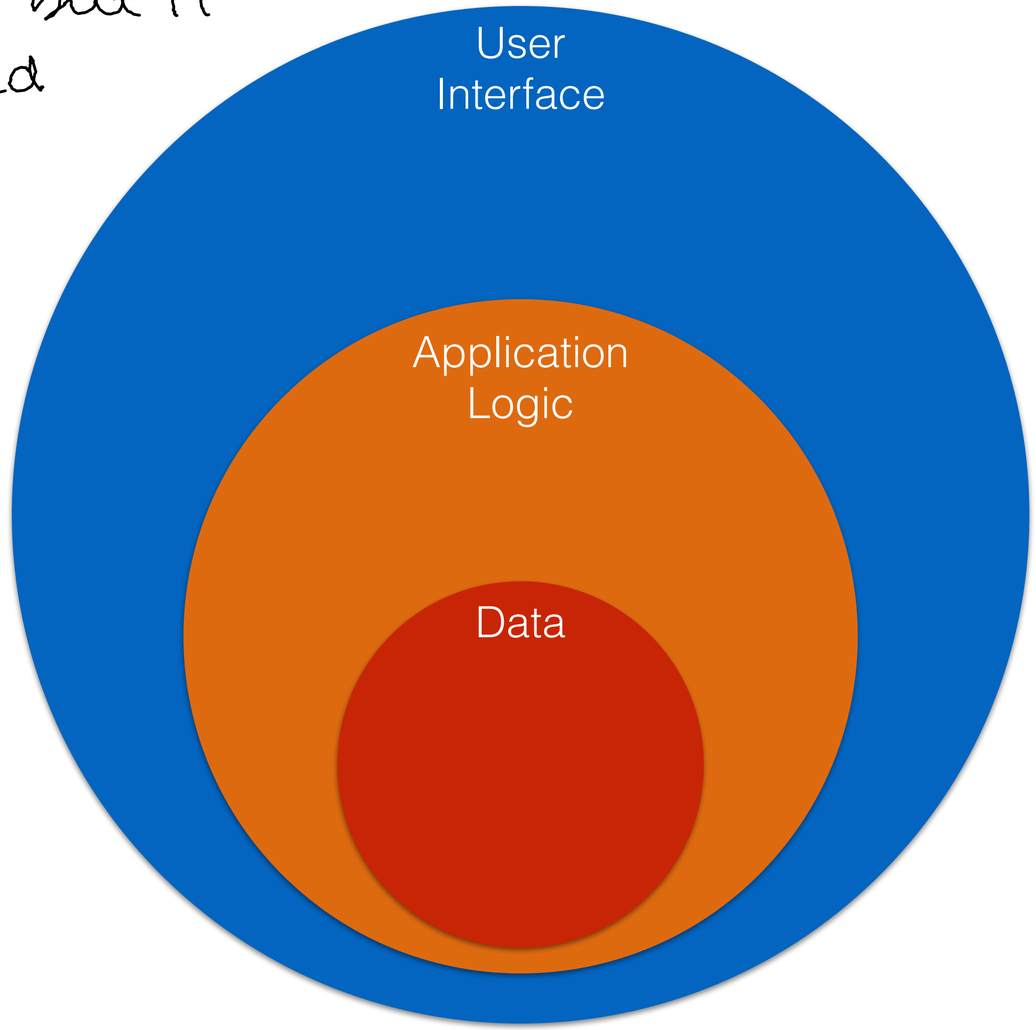


Monoliths (the traditional way…)

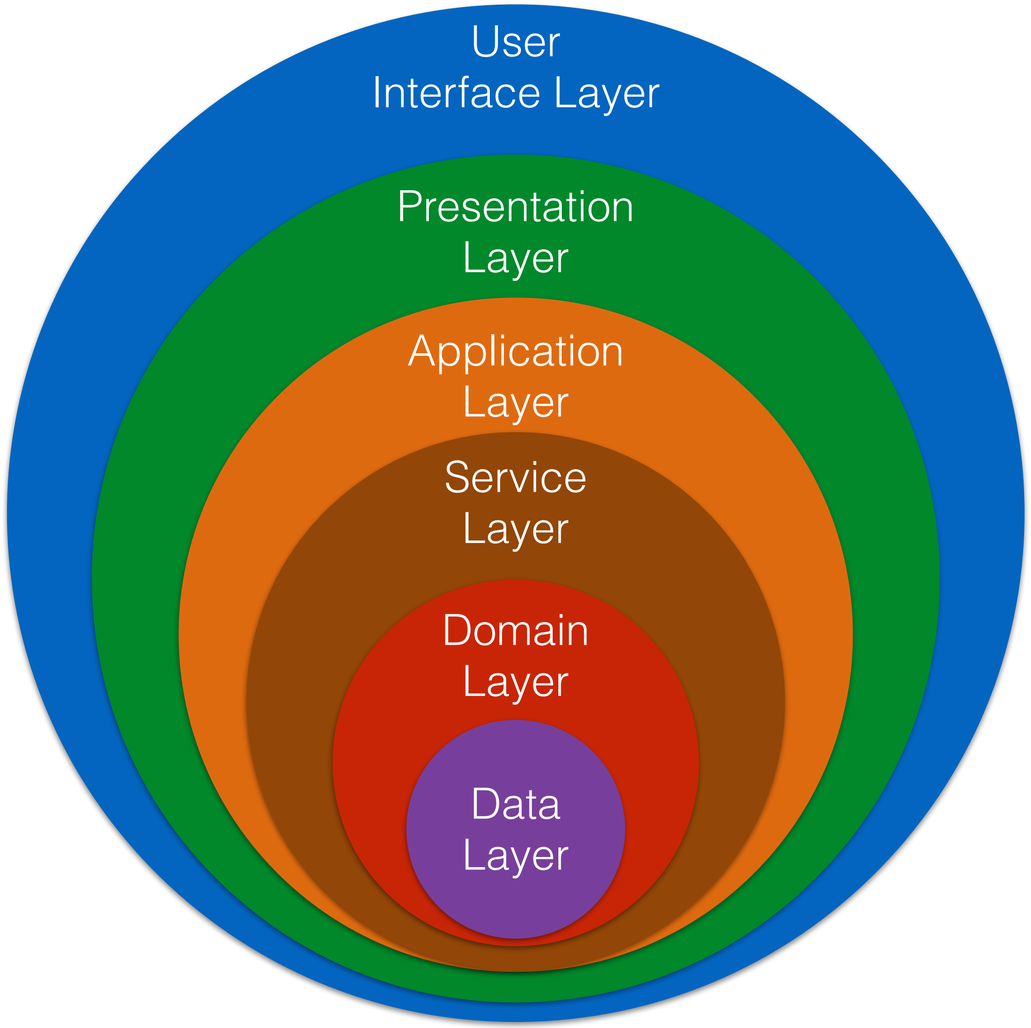
Overall, we have a high level of control but it comes at the cost of flexibility & speed

Challenges:

- Ownership/sharing of code
- Technological flexibility
- Deployment is slower and synched but under control
- Cohesion is high, and structural errors are detected early (on the build)



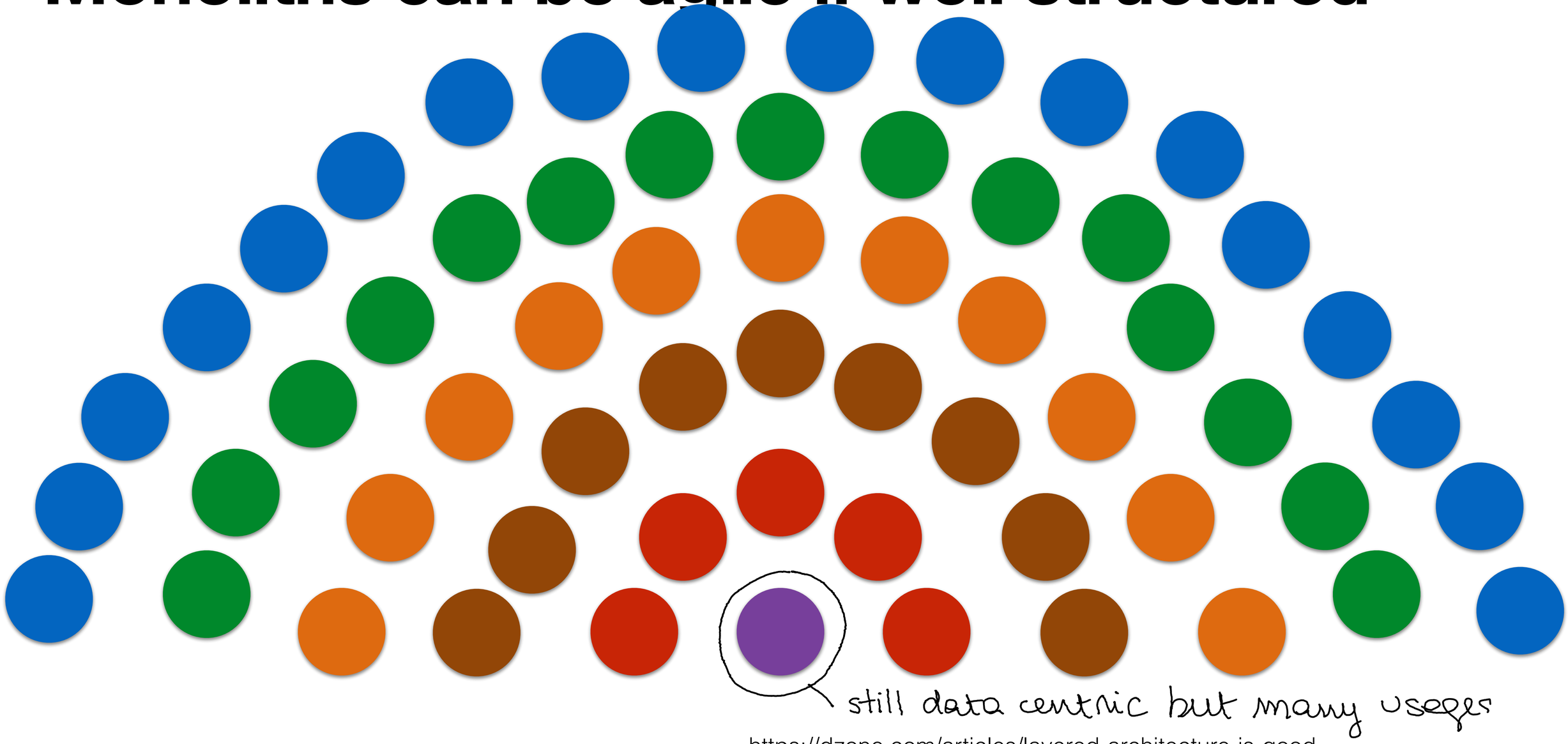
Monoliths can be agile if well structured



More layers will give more flexibility

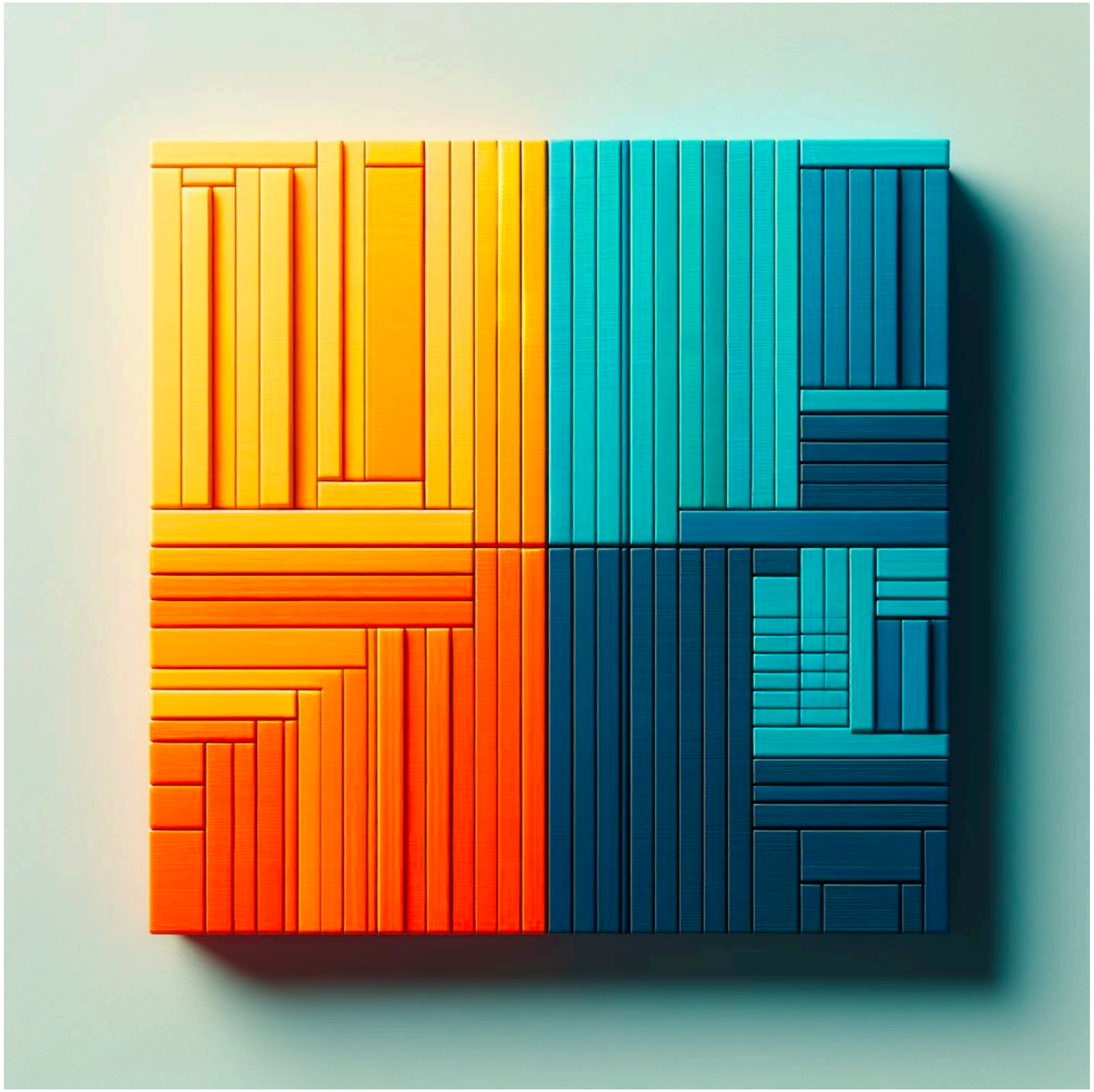
https://dzone.com/articles/layered-architecture-is-good

Monoliths can be agile if well structured



https://dzone.com/articles/layered-architecture-is-good

Services



Service-oriented architecture

~ reconfigurable

~ wired

An architectural style that favours discrete services instead of monolithic components.

Modules connect via a network protocol instead of coupled direct calls.

Modules communicate via connecting structures like brokers or message queues.

Modules are loosely coupled and can be independently deployed.

Services are self-contained, stateless, black-box components.

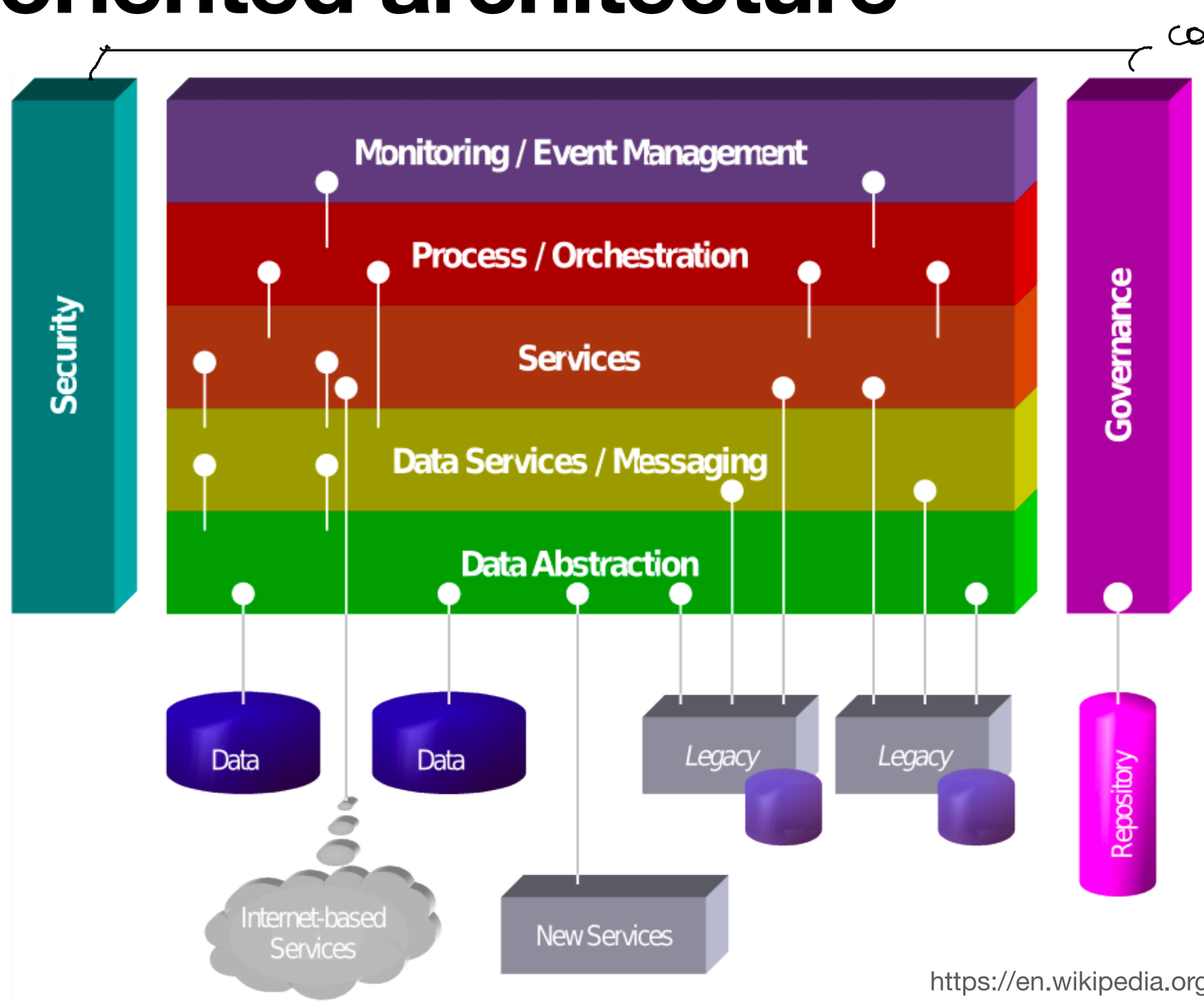
Services provide an API and may depend on other services.

~ functions

Services are described by metadata that specify how they interact.

Services execute independently of one another.

Service-oriented architecture



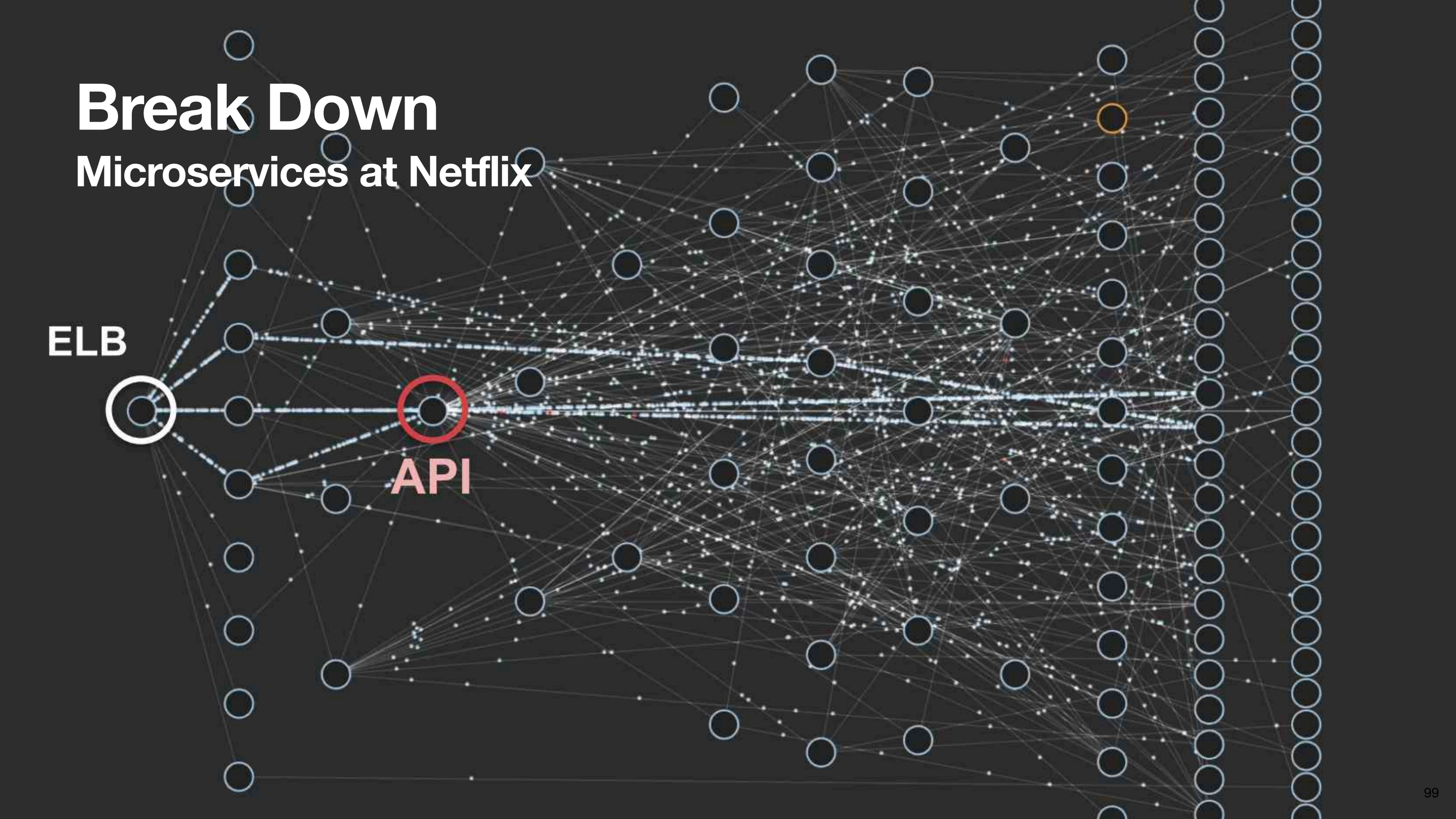
https://en.wikipedia.org/wiki/Service-oriented_architecture

Microservices

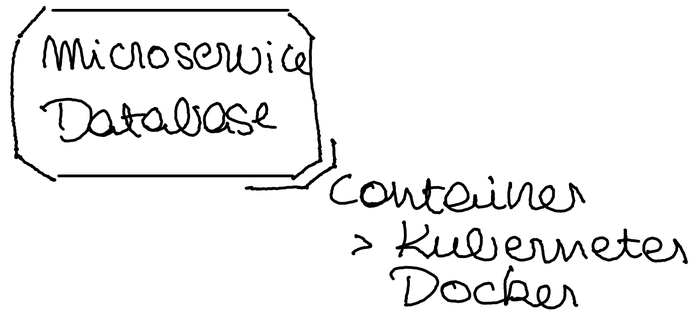
~ SOA but with more granular services

Break Down

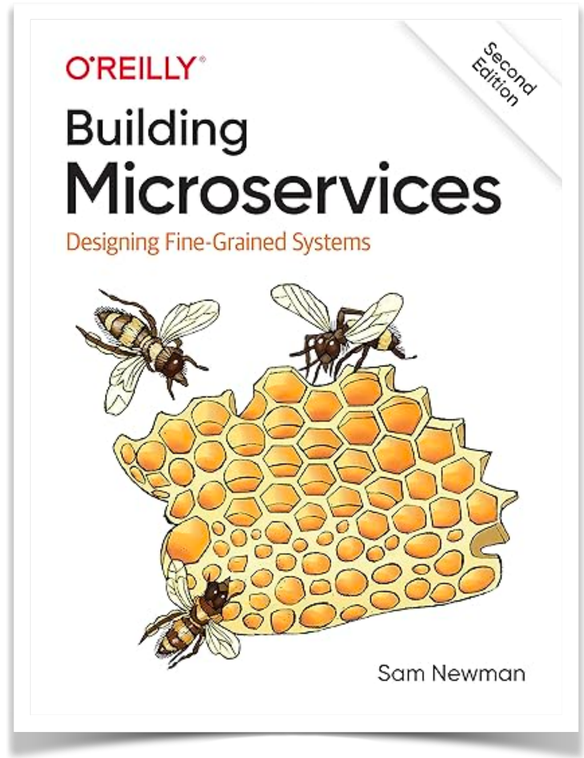
Microservices at Netflix



Microservices



- Microservices are independently releasable services modelled around a business domain (as are Services).
- Microservice architectures avoid using shared databases; each microservice encapsulates its database.
- Microservice architectures are usually containerised and managed by frameworks.
  - Containers are a lightweight way to easily provision several instances
  - Containers are often managed by orchestration platforms like Kubernetes



Do we use one container / service?

Interface Technology

REST

Representational State Transfer

Restful interface design → can support a SOA

- Follows an architectural style that promotes a simple and efficient way of providing and connecting web services. Built on top of basic HTTP. (base protocol)
- Promotes the decoupling between Data-centric server side applications and client user-centric applications.
- Implementations provide (convenient) flavours
  - Web-service style pure JSON/XML Data
  - Complete/partial HTML view responses

  (one step further with a representation layer)
- Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine

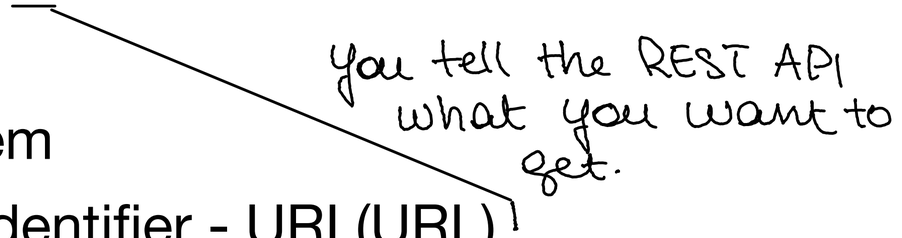
REST - Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

Domain) Path) Query

Representational State Transfer

- Resource Based
  - vs Action Based
  - Nouns and not verbs to identify data in the system
  - Identified (represented) by a Uniform Resource Identifier - URI (URL)
  - Aliasing is admissible (two URLs pointing to the same data)
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)



Representational State Transfer

- Resource Based
- Representation
  - JSON or XML representation of the state of a given resource transferred between client and server at a given verb in a given URL.
  - Well identified interface (the information retrieved at an URL — the type)
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
  - standard HTTP verbs (GET, PUT, POST, DELETE) → standard actions
  - standard HTTP response (status code, info in the response body) → std outcomes
  - Uniform structure of URIs with a name, identifying the resource → std call
  - References inside responses must be complete. → predefined schema
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
  - Server does not hold session state
  - Messages are self contained
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

the resource is released very fast.

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
  - Responses can be tagged as cacheable (in the server)
  - (also) Bookmarkable
- Layered System
- Code on Demand (not talking about it)

we can keep it in "memory" and avoid unnecessary calls

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Layered System
  - Establishes an API between a client and a “database”
- Code on Demand (not talking about it)

→ thus monolithic yet flexible because of numerous user interfaces

EXAMPLES

Google Glasses worked with REST to send info to the glasses
Twitter made the decision to only focus on the API from the beginning > other companies focussed on making the applicat.

6. Real REST Examples

Here's a very partial list of service providers that use a REST API. Note that some of them also support a WSDL (Web Services) API, in addition, so you can pick which to use; but in most cases, when both alternatives are available, REST calls are easier to create, the results are easier to parse and use, and it's also less resource-heavy on your system.

So without further ado, some REST services:

- The Google Glass API, known as "Mirror API", is a pure REST API. Here is an excellent video talk about this API. (The actual API discussion starts after 16 minutes or so.)
- Twitter has a **REST API** (in fact, this was their original API and, so far as I can tell, it's still the main API used by Twitter application developers),
- **Flickr**,
- **Amazon.com** offer several REST services, e.g., for their S3 storage solution,
- **Atom** is a RESTful alternative to RSS,
- **Tesla Model S** uses an (undocumented) REST API between the car systems and its Android/iOS apps.

) from the beginning

in … http://rest.elkstein.org/2008/02/real-rest-examples.html

Twitter Example

Twitter API v2: Early Access

Tweets

Filtered stream

resource

- GET /2/tweets/search/stream
- GET /2/tweets/search/stream/rules
- POST /2/tweets/search/stream/rules

version

Hide replies

- PUT /2/tweets/:id/hidden

Sampled stream

- GET /2/tweets/sample/stream

Search Tweets

Tesla Example

Tesla API

Search...

Tesla API

AUTHENTICATION

OAuth

User

PRODUCTS

List

VEHICLES

List

State And Settings

Commands

POWERWALLS

State And Settings

Commands

ENERGY SITES

Tesla API

How is this site organized?

This site is broken into sections for different API information:

- Authentication - API commands to generate an `{access_token}` to communicate with your vehicle.
- Vehicles - API commands to communicate between your vehicle and your client.
- Codes - Localized Tesla vehicle option codes for the Model S and Model X

What Tesla products are supported?

- Tesla Model S

State And Settings

protocol

domain

GET https://owner-api.teslamotors.com/api/1/vehicles/:id/data

↳ subdomain ↳ path

Vehicle Data (Legacy)

GET https://owne

Vehicle Data

GET https://owne

Vehicle Service Da

GET https://owne

Mobile Enabled

GET https://owne

Charge State

Commands

POST https://owner-api.teslamotors.com/api/1/vehicles/:id/wake_up

Wake Up ! That's a verb!

POST https://owner-api.teslamotors.com/api/1/vehicles/:id/command/door_unlock

Unlock Doors

POST https://owner-api.teslamotors.com/api/1/vehicles/:id/command/door_lock

Lock Doors

POST https://owner-api.teslamotors.com/api/1/vehicles/:id/command/honk_horn

Honk Horn

↳ set an attribute / data to a new value

RESTful design

- Resource = object or representation of something
- Collection = a set of resources
- URI = a path identifying **resources** and allowing actions on them + action = URL
- URL methods represents standardised actions
  - GET = request resources
  - POST = create resources
  - PUT = update or create resources
  - DELETE = deletes resources
- HTTP Response codes = operation results → standardized returns as well
  - 20x Ok
  - 3xx Redirection (not modified)
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
  - 5xx Server Error
- Searching, sorting, filtering and pagination obtained by query string parameters > part of the collection ? arg=value
- Text Based Data format (JSON, or XML) > easy to parse

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

Example of data modelling in REST

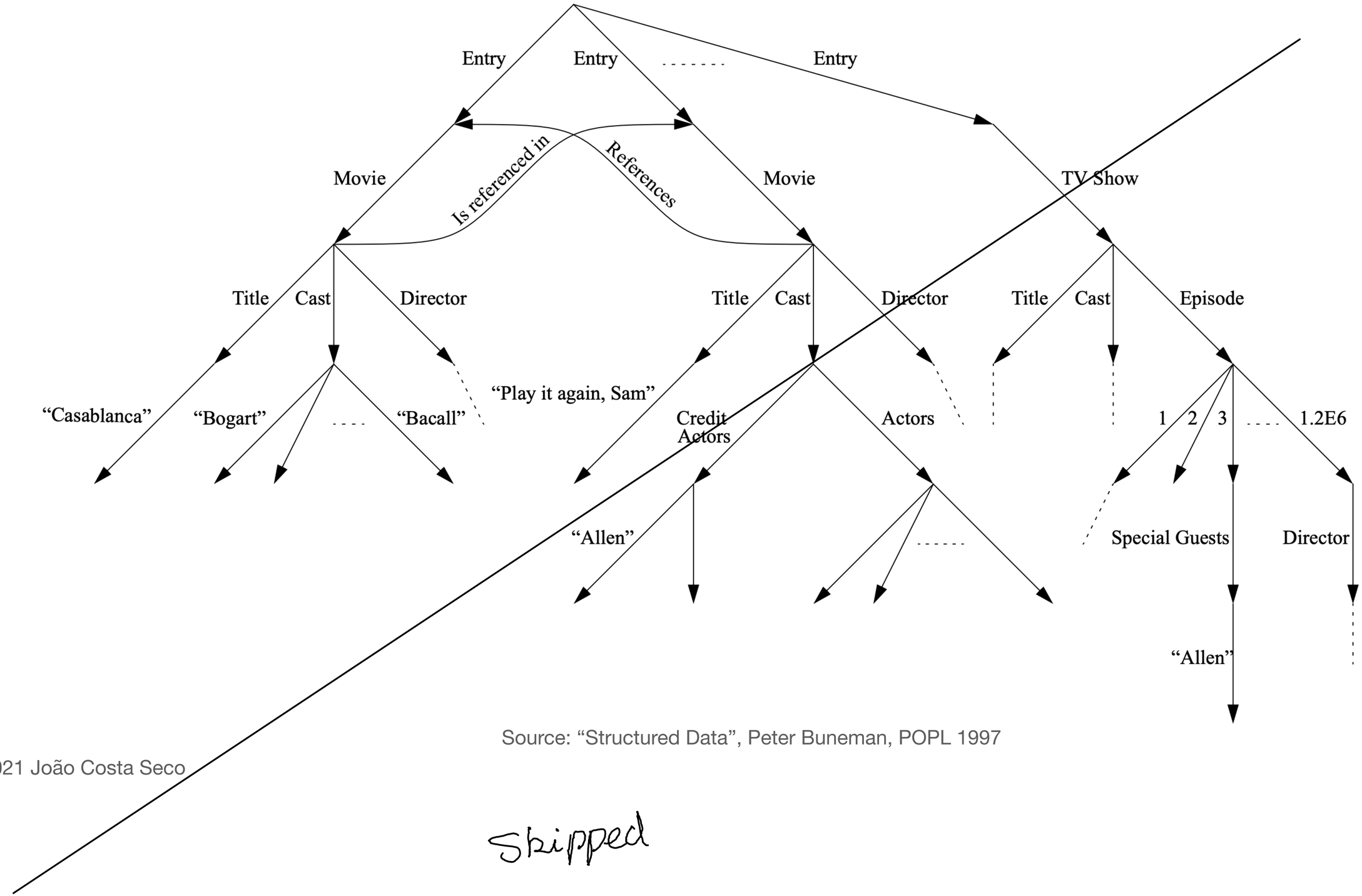
- Application to manage contacts of partner companies (e.g. for security clearance in events)
- Resources > what am I talking about & what are their attributes.
  - **Companies** (name, address, email, list of contacts (employees))
  - **Contact/Employee** (name, email, job, company)
- Operations (CRUD) /post /put /get \delete
  - List, add, update, and delete resources

Semi Structured data modelling

- The data model of document-based NoSQL databases (e.g. MongoDB)
- Favours collections of hierarchical data (objects) where ownership is well represented
- Relations between objects can be achieved with keys (identifiers)
- Collections can be indexed for better performance

Structure of a REST response

```
const initialCompanies =
    [
        {name:"EDP",
         contacts:
            [
                {name:"one at edp", email:"one@edp"}
                 , { name:"two at edp", email:"two@edp"}
                 , { name:"three at edp", email:"three@edp"}
            ]
        }
    ,   {name:"RTP",
         contacts:
            [
                {name:"one at rtp", email:"one@rtp"}
            ,   {name:"two at rtp", email:"two@rtp"}
            ,   {name:"three at rtp", email:"three@rtp"}
            ]
        }
    ]
```



Source: "Structured Data", Peter Buneman, POPL 1997

Skipped

Partner companies

- `GET /companies` - List all the companies
- `GET /companies?search=<criteria>` - List all the companies that contain the substring `<criteria>`
- `POST /companies` - Create a company described in the payload. The request body must include all the necessary attributes.
- `GET /companies/{id}` - Shows the company with identifier {id}
- `PUT /companies/{id}` - Updates the company with {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /companies/{id}` - Removes the company with {id}

) usually don't just scrape that change the attribute 'delete' of the resource instead.

Partner contacts

search=id would not follow the rest API

- `GET /contacts` - List all the contacts
- `GET /contacts?search=<criteria>` - List all the contacts that contain the substring `<criteria>`
- `POST /contacts` - Create a contact described in the payload. The request body must include all the necessary attributes.
- `GET /contacts/{id}` - Shows the contact with identifier {id}
- `PUT /contacts/{id}` - Updates the contact with {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /contacts/{id}` - Removes the contact with {id}

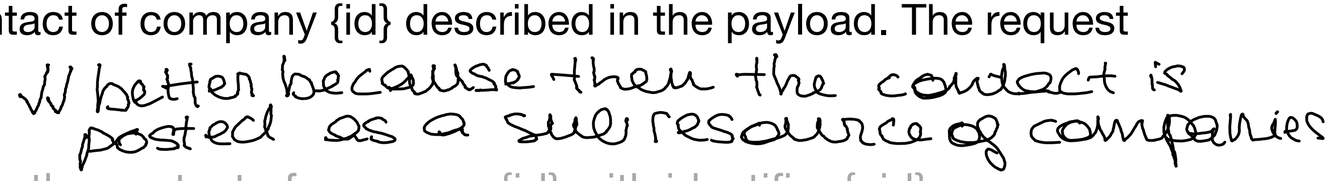
Partner contacts

- `GET /contacts` - List all the contacts
- `GET /contacts?search=<criteria>` - List all the contacts that contain the substring `<criteria>`
- `POST /contacts` - Create a contact described in the payload. The request body must include all the necessary attributes.
- `GET /contacts/{id}` - Shows the contact with identifier {id}
- `PUT /contacts/{id}` - Updates the contact with {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /contacts/{id}` - Removes the contact with {id}

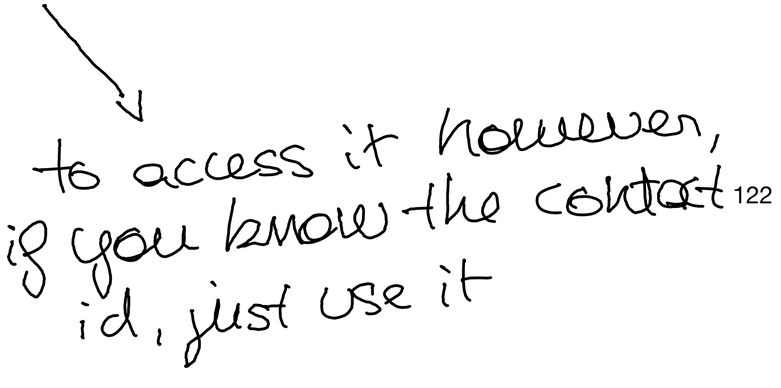
→ contacts belong to companies so you should access it through its company

Partner contacts of companies

- `GET /companies/{id}/contacts` - List all the contacts of a company
- `GET /companies/{id}/contacts?search=<criteria>` - List all the contacts of a company that contain the substring `<criteria>`
- `POST /companies/{id}/contacts` - Create a contact of company {id} described in the payload. The request body must include all the necessary attributes.



- `GET /companies/{id}/contacts/{cid}` - Shows the contact of company {id} with identifier {cid}
- `PUT /companies/{id}/contacts/{cid}` - Updates the contact with {cid} of company {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /companies/{id}/contacts/{cid}` - Removes the contact with {id}



Semistructured Data modelling in REST

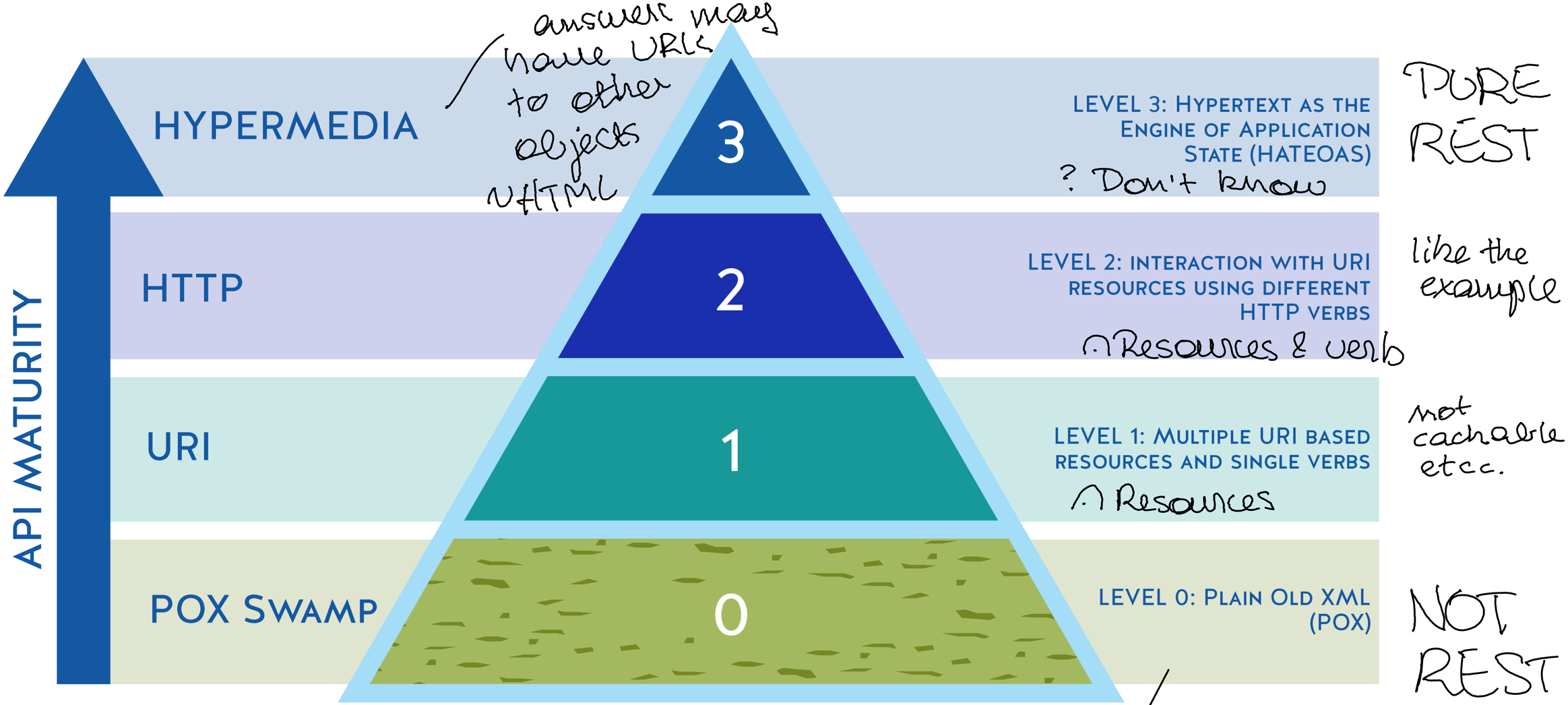
no strict schema imposed ex-ante as in SQL

- Collection- and Object- based data types
- Ownership relations by inclusion
- Other relations may use direct object references
- Flexible structure of data schema

```
[{
  "name": "John",
 },
 {
  "name" : "Jack",
  "parent": {"$ref": "$.[?(@.name=='John')]"}
 },
...
]
```

```
{
  "_id" : ObjectId("5126bbf64aed4daf9e2ab771"),
  // .. application fields
  "creator" : {
                    "$ref" : "creators",
                    "$id" : ObjectId("5126bc054aed4daf9e2ab772"),
                    "$db" : "users"
                  }
}
```

THE RICHARDSON MATURITY MODEL



GraphQL

GraphQL

- is a query language for APIs
- is supported by a server-side runtime for executing queries
- isn't tied to any specific database or storage engine

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
```

https://graphql.org/learn

Extra Reading and Research Pointers

API centric

Explore the Modularity Aspect of SOA (Logical and Technological)

indpt data blackboxes

Explore the difference between SOA and Microservices

Is it all sunshine with Microservices? → repetition of data resources