

Internet Applications Design and Implementation

(Lecture 2 - Software Architecture)

**MIEI - Integrated Master in Computer Science and Informatics
Specialization block**

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

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

Outline

- Software Architecture - Introduction
- Software Architecture for Internet Applications
 - Three-tier architecture
 - Service-based architectures
 - Microservice-based architectures
- Frameworks at the service of Software Architecture
- The architectural style REST to instantiate webservices
- Specifying webservices with OpenAPI and Spring

Internet Applications Design and Implementation

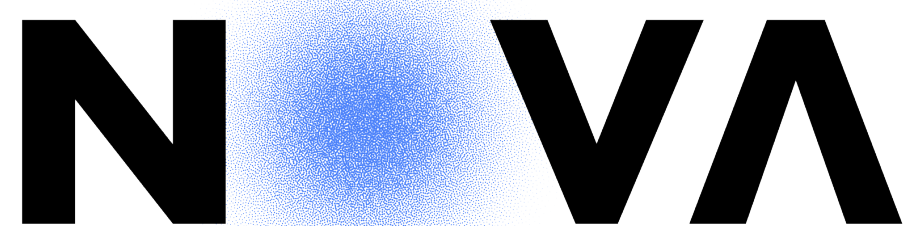
2020 - 2021

(Lecture 2, Part 1 - Software Architecture - Introduction)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

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

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))



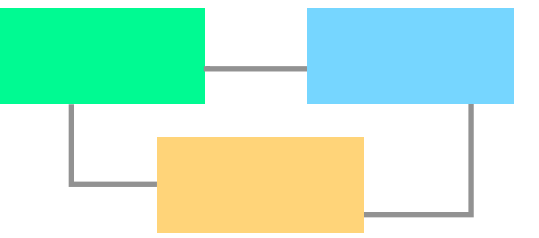
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Software Architecture

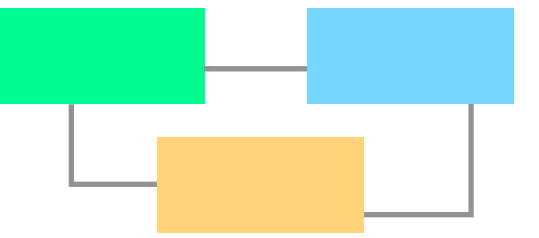
Introduction

based on the book “Software Architectures in Practice”

Software Architecture



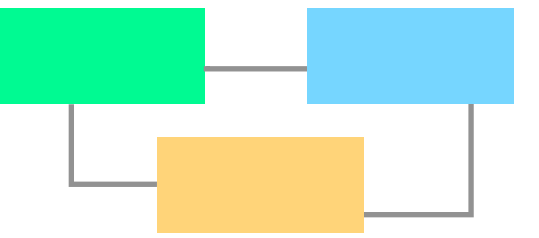
- What is software architecture?
- What are the benefits of using one?



- What is software architecture?

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.” (in Software Architectures in Practice)

- Structures can be:
 - the module decomposition structure, which divide computational responsibilities and work assignment (among teams). Identify modules or components
 - runtime structures (connectors) that deal with the communication between components (eg services)
 - organisational structures. How components are developed, tested, deployed



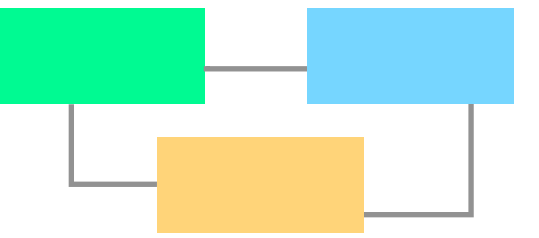
... is a form of Abstraction
(different views over the same system)

... is in every software system
(from caos to tidy)

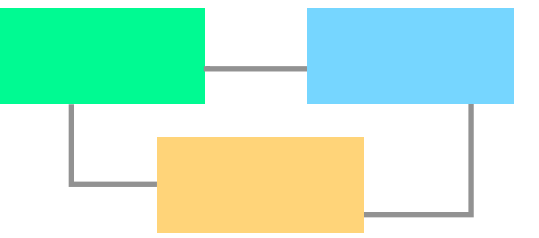
... includes Behaviour
(names and connectors have semantics)

Not All Architectures Are Good Architectures

Software Architecture (Structures)



- Module structures
 - What is the primary functional responsibility assigned to each module?
 - What are the dependencies to other software elements?
 - What modules are each module related to? by generalisation or specialisation.
- Component and connector structures
 - How do modules interact?
 - What are the shared data stores?
 - What is the data flow in the system?
 - Can the structure change? how?
 - What are the security requirements? performance bottlenecks?
- Allocation structures
 - What is the hardware/cloud infrastructure? module ownership? which are the regressions tests? etc.



- Pre-determined compositions of architectural elements provide strategies for solving common problems in software systems.
- Examples:
 - Layered pattern — Linear (unidirectional) dependencies between multiple elements
 - Shared-data pattern — Components and connectors to create and manipulate persistent data, connectors are languages like SQL.
 - Client-server pattern — Components: Clients and servers; Connectors: protocols and languages
 - Multi-tier pattern — A generic deployment structure of components in different infrastructures
 - Competence center — Work assignment division by expertise (eg. departments)

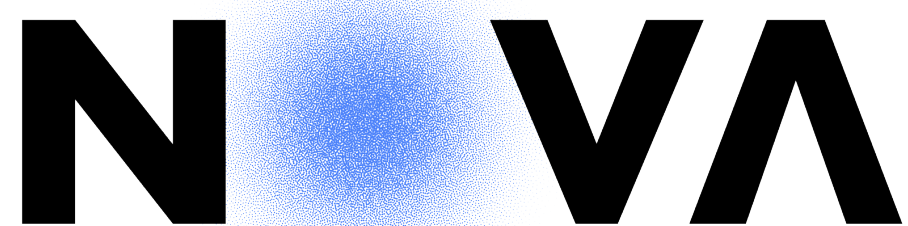
Internet Applications Design and Implementation 2020 - 2021

(Lecture 2, Part 2 - Software Architecture - Internet Applications)

**MIEI - Integrated Master in Computer Science and Informatics
Specialization block**

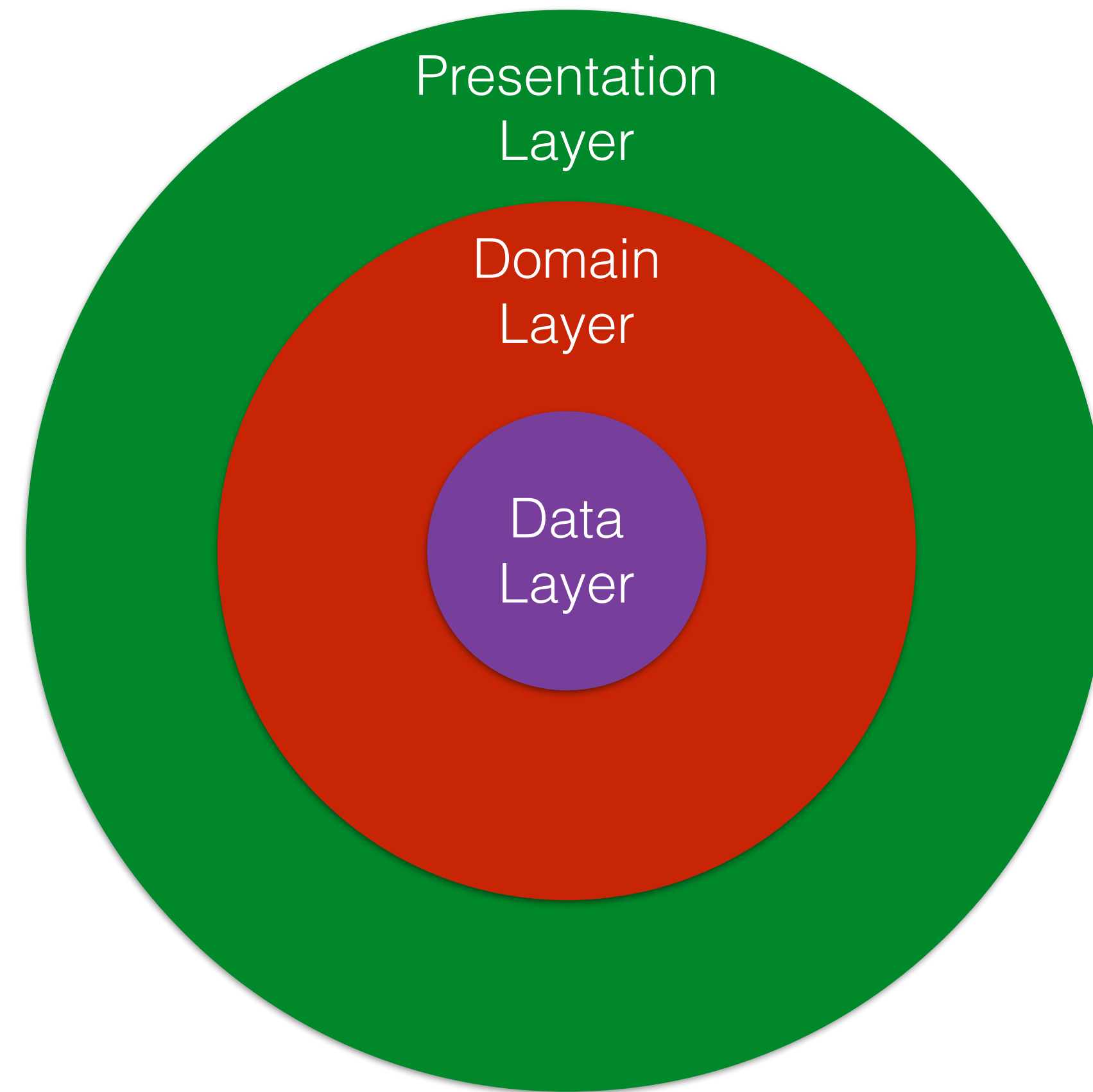
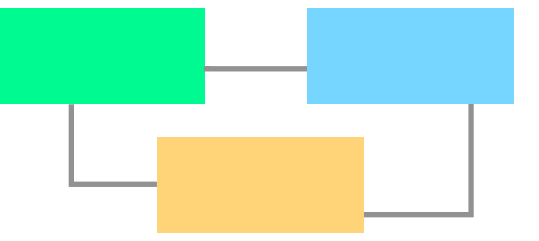
João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))



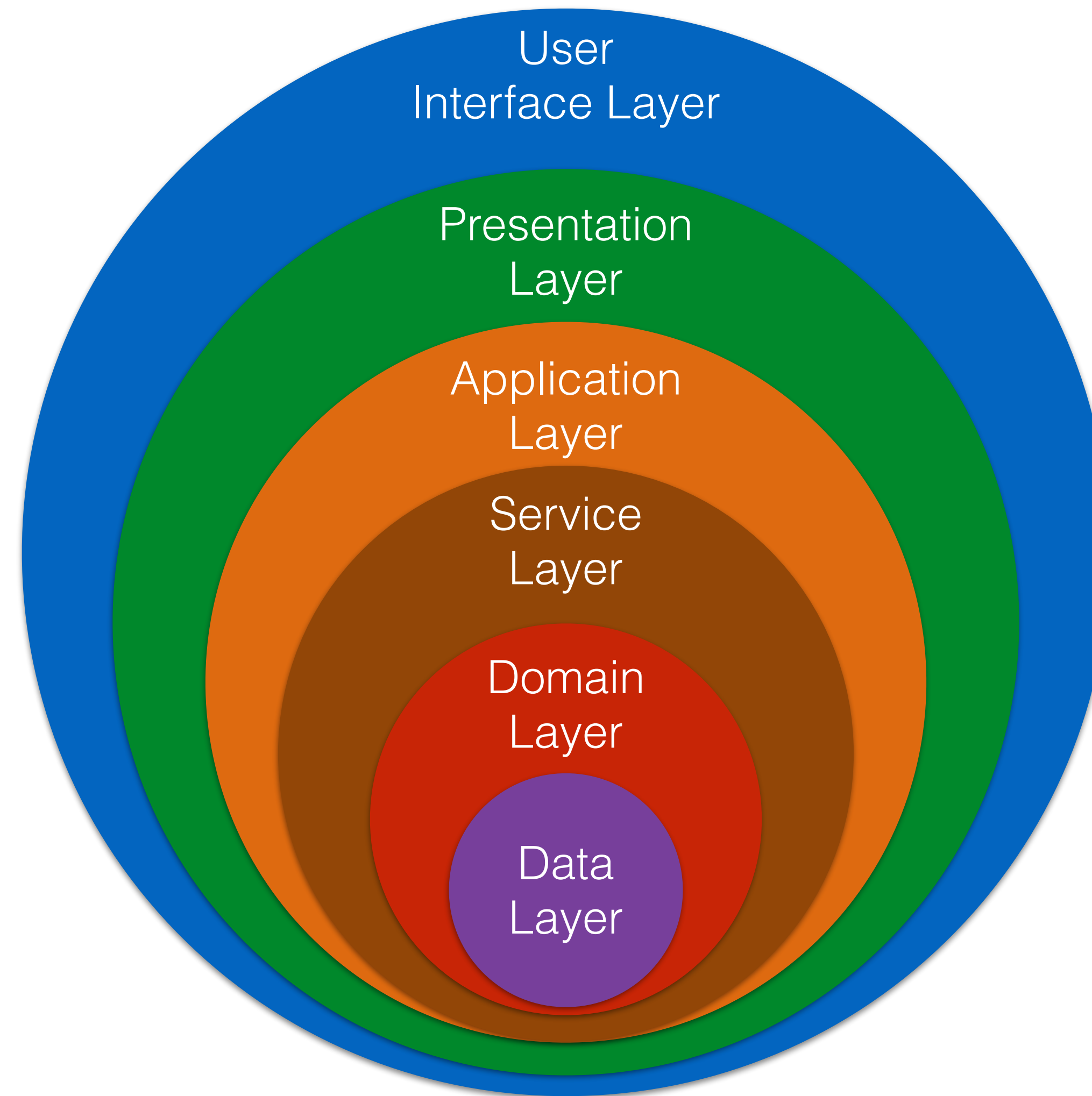
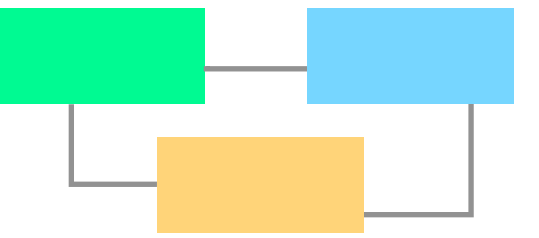
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Internet Applications are Data-Centric



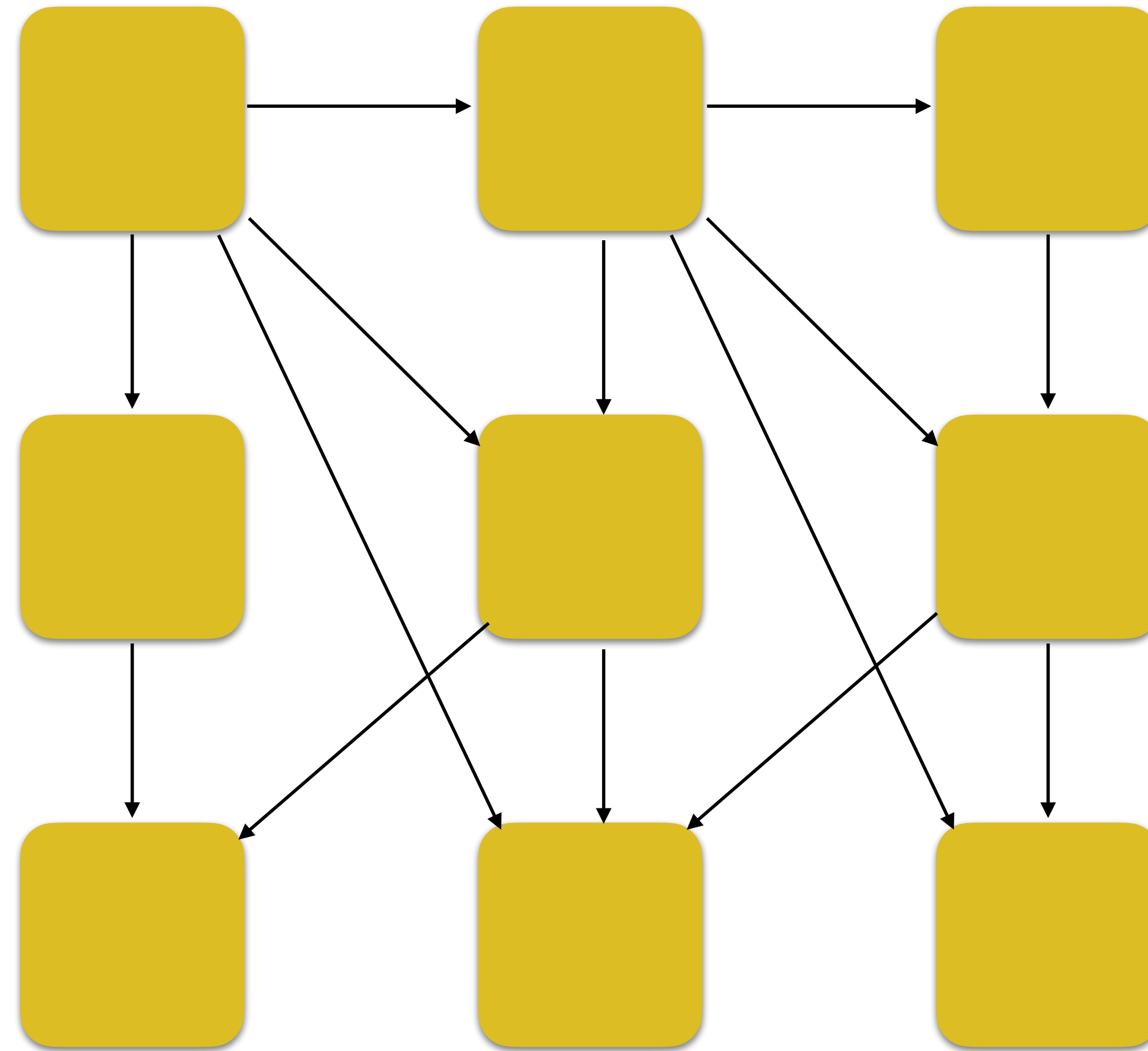
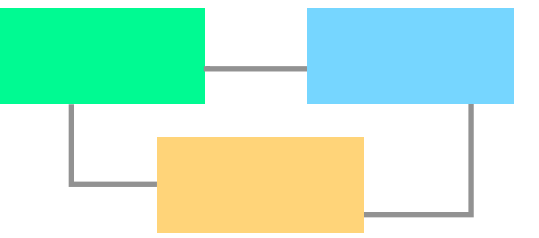
Patterns of Enterprise Application Architecture

Internet Applications are Data-Centric



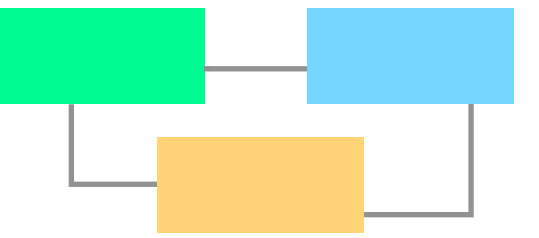
<https://dzone.com/articles/layered-architecture-is-good>

Internet Applications are also Decentralised

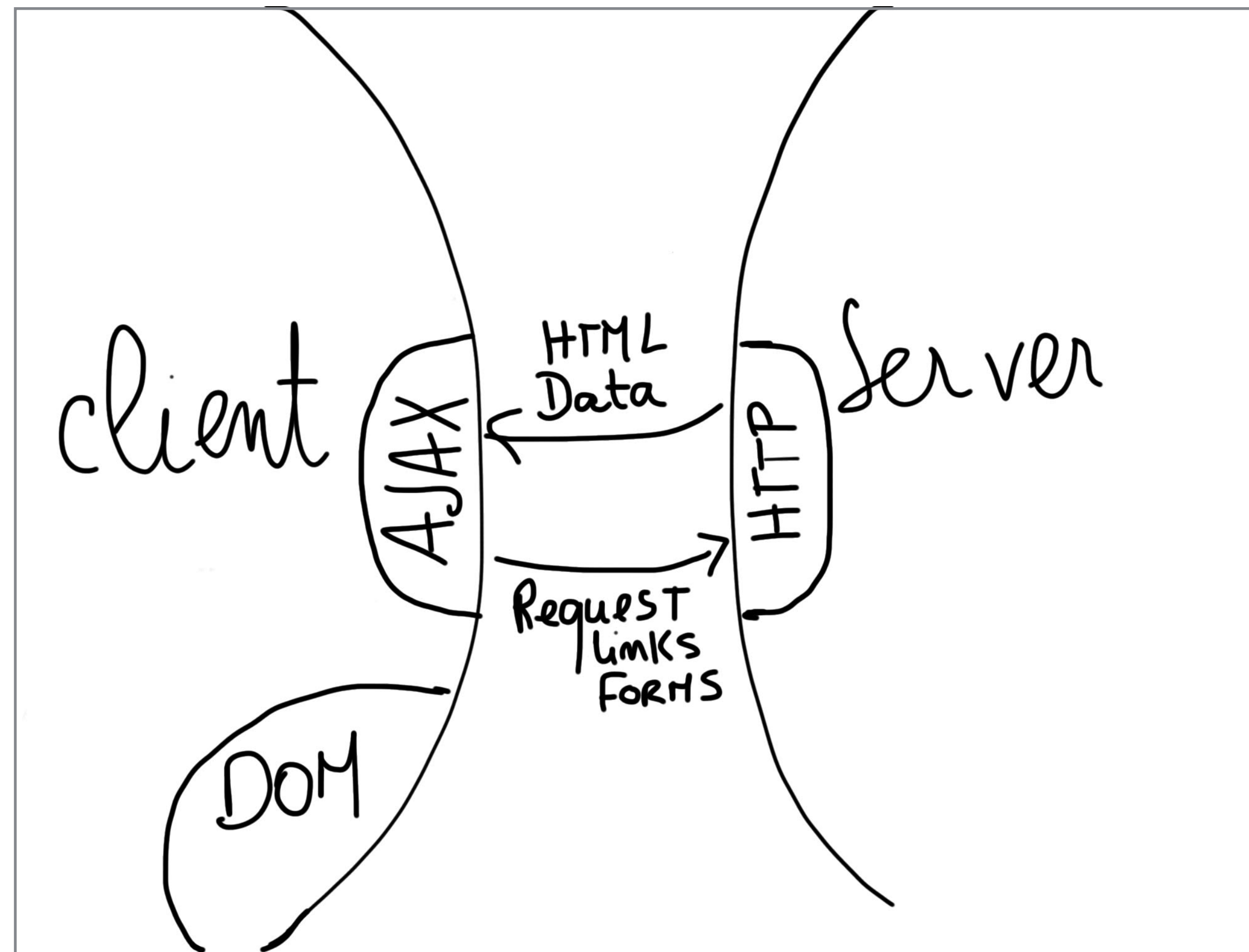


<https://dzone.com/articles/introduction-to-microservices-part-1>

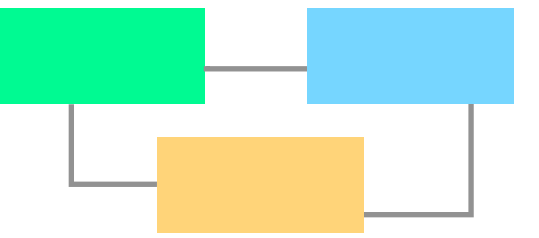
(Logical) Architecture of Web applications



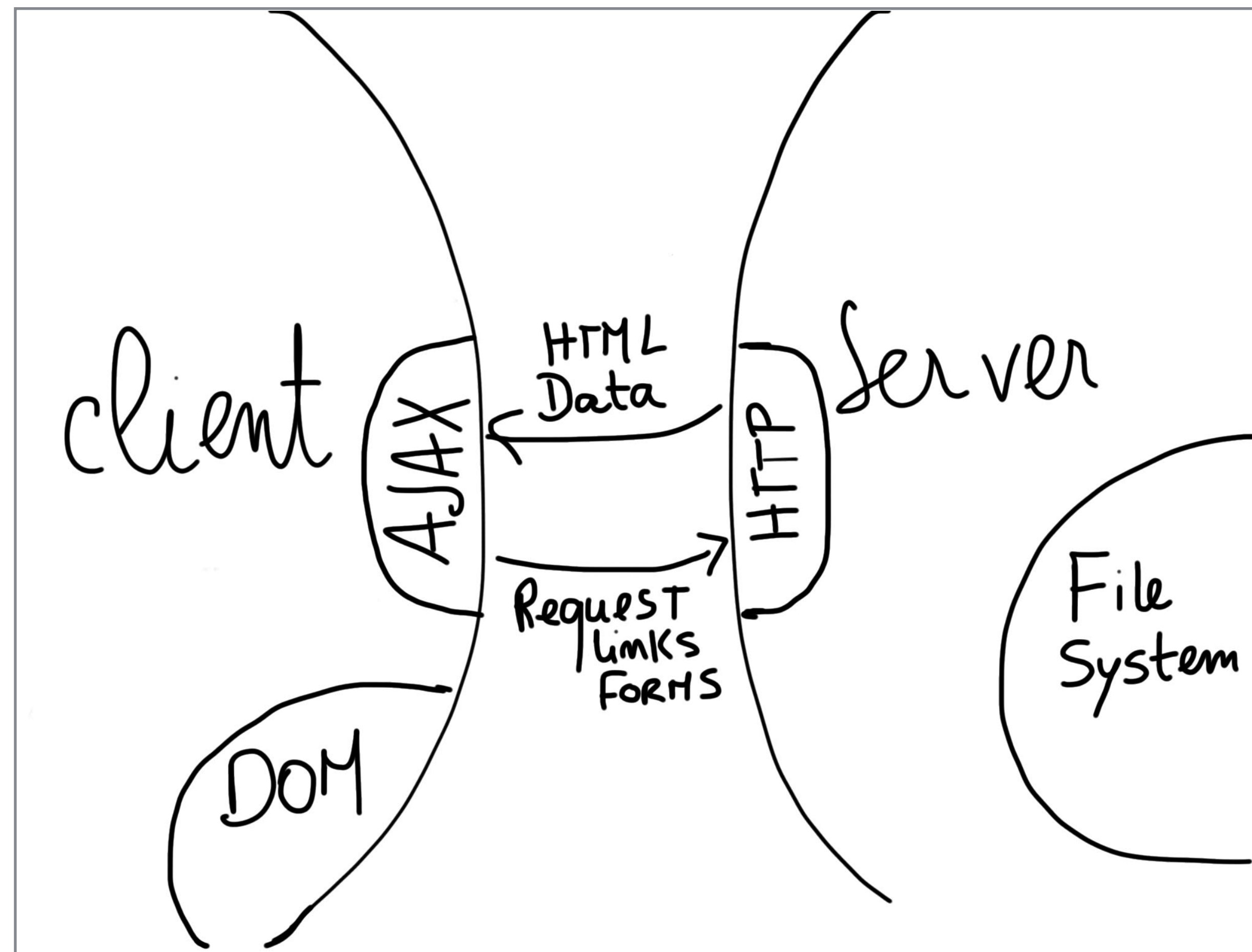
- Tech Details: Interconnection based on the HTTP protocol
 - Method, path, arguments, (a)synchrony, multi-typed response



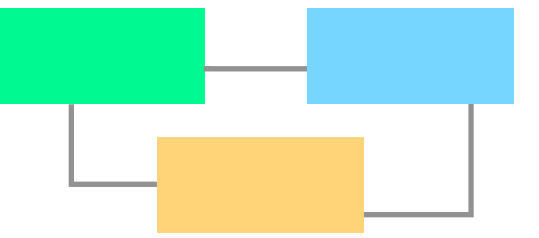
Architecture of Web applications



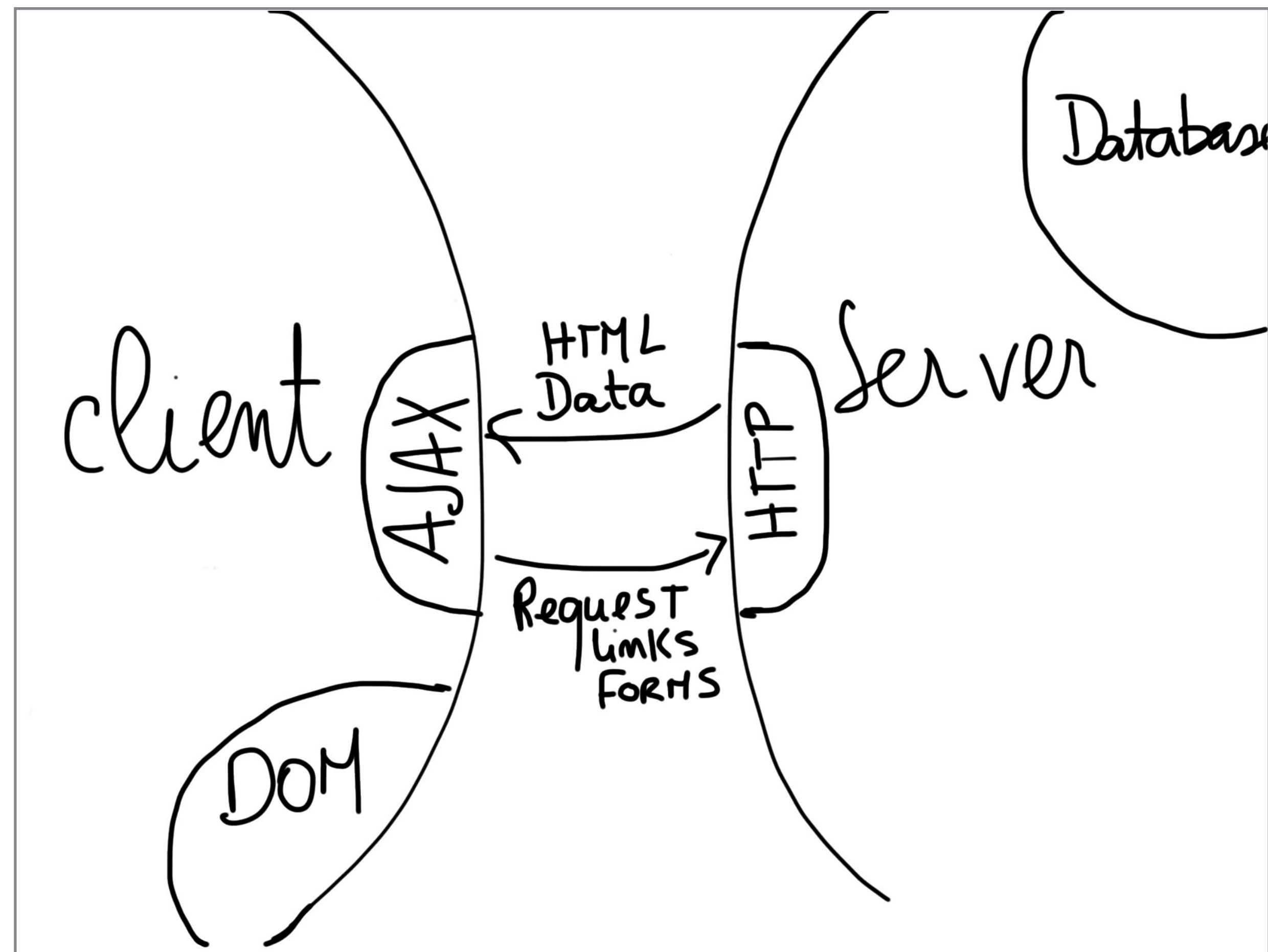
- Static HTML pages stored in the file system



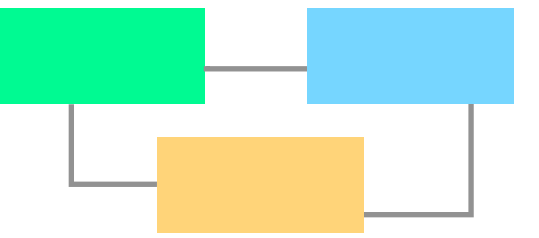
Architecture of Web applications



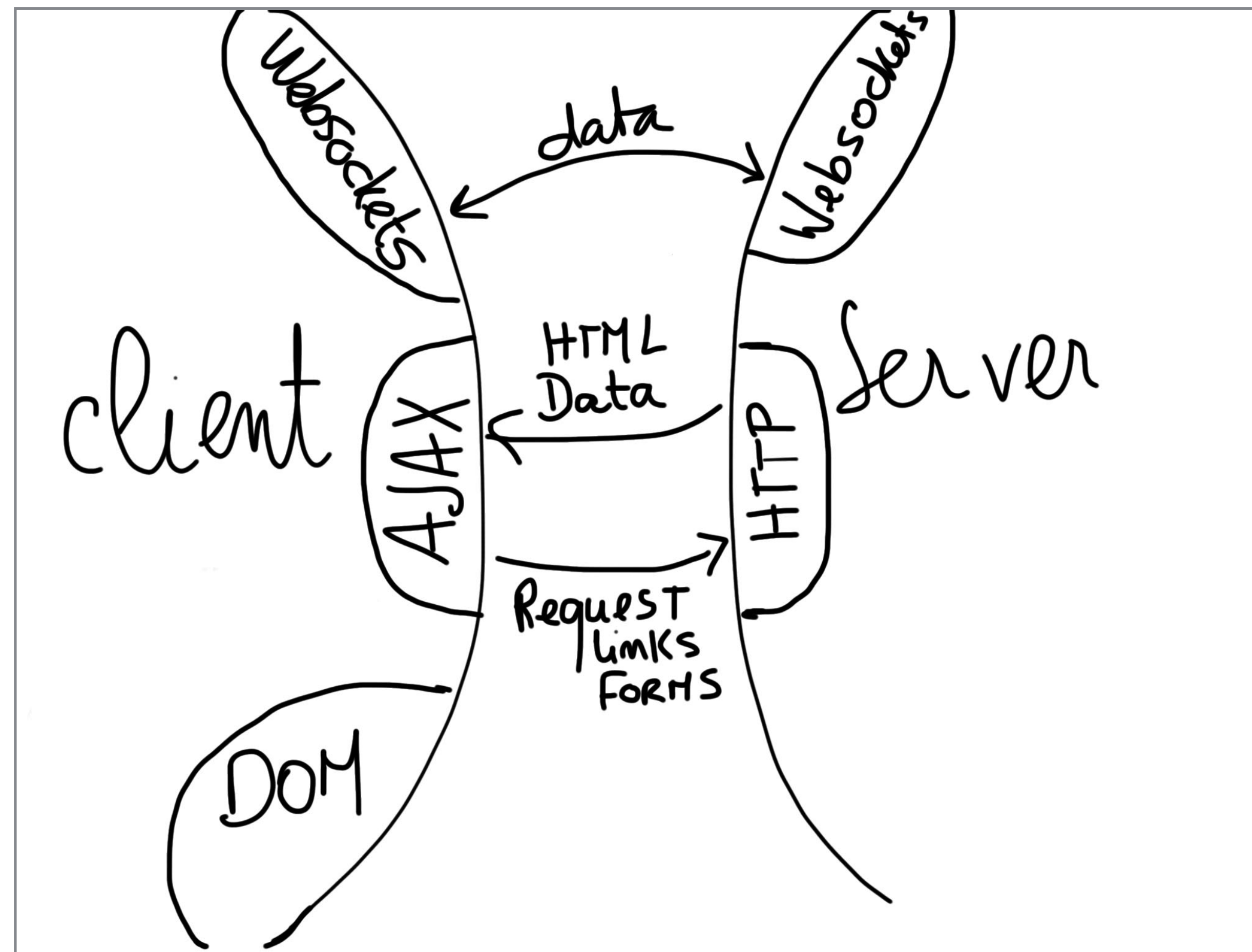
- Dynamic HTML pages based on data stored in some database.



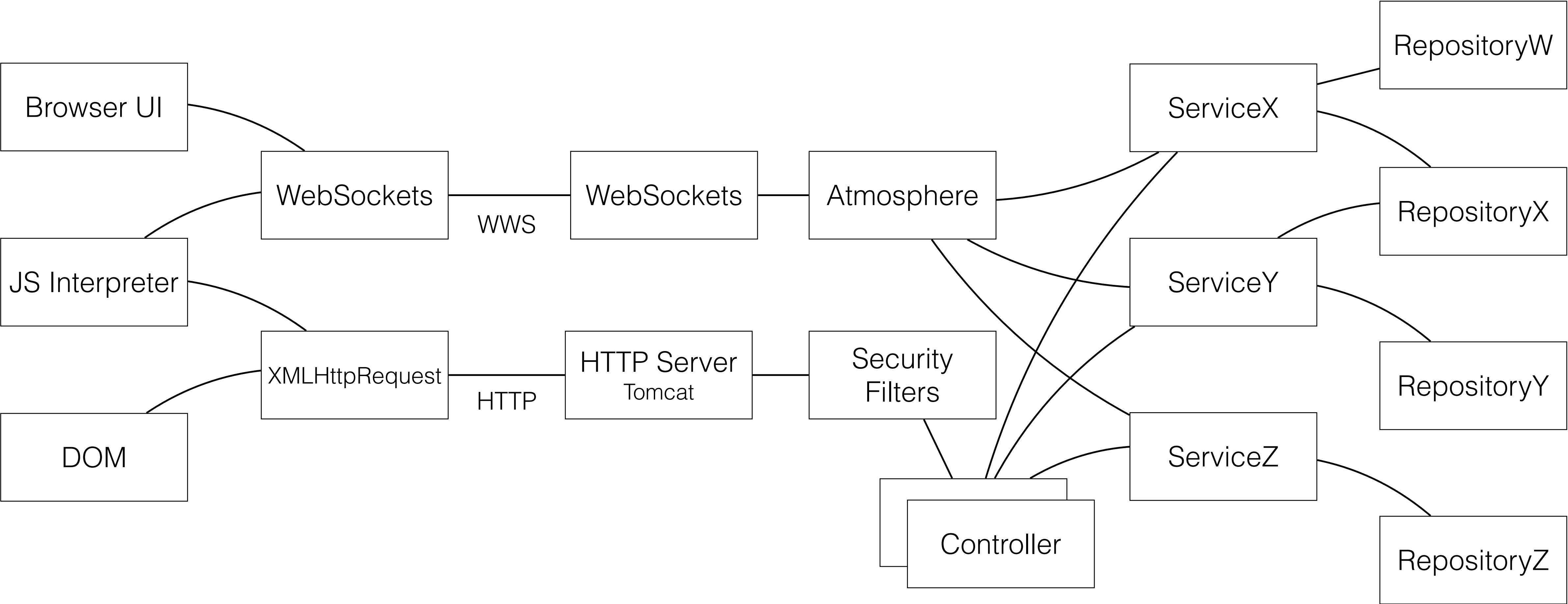
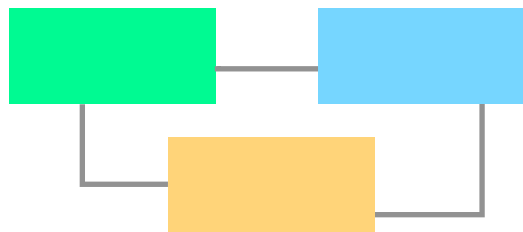
Architecture of Web applications



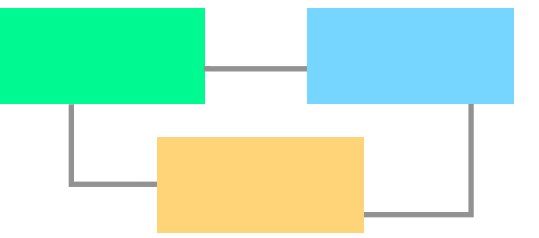
- Dynamic HTML pages with permanent connection with the server to exchange data.



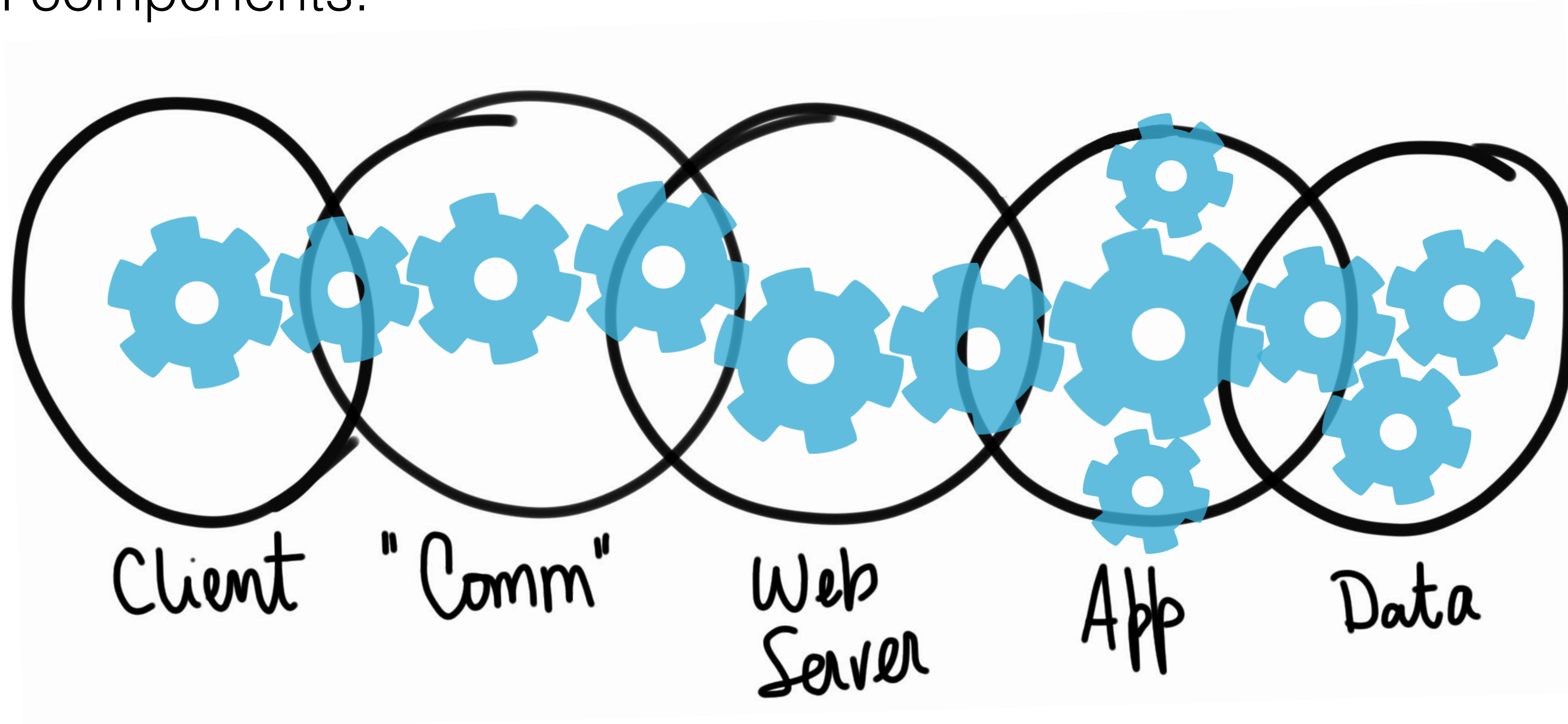
Architecture of Web applications



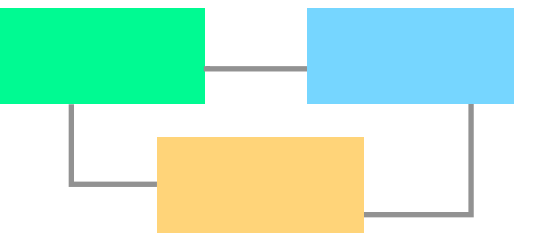
Architecture of Web applications - The Big Picture



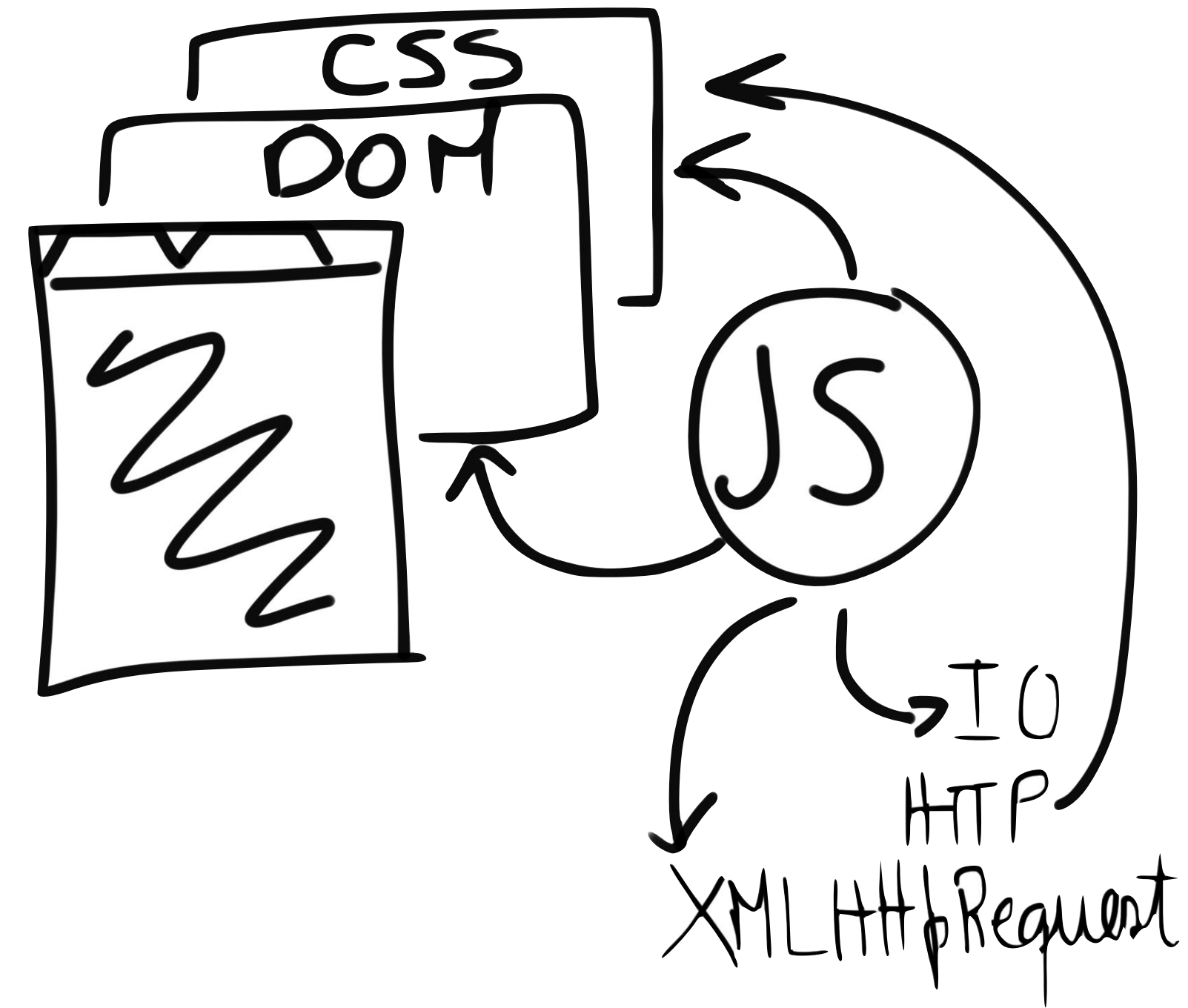
- A web application is made of code deployed to the independent and different physical components.

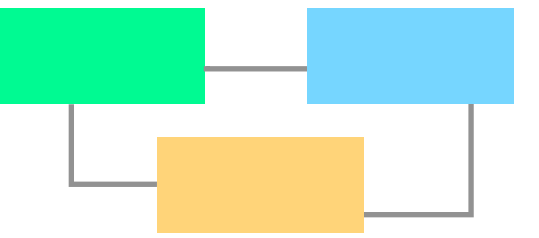


Web client architecture

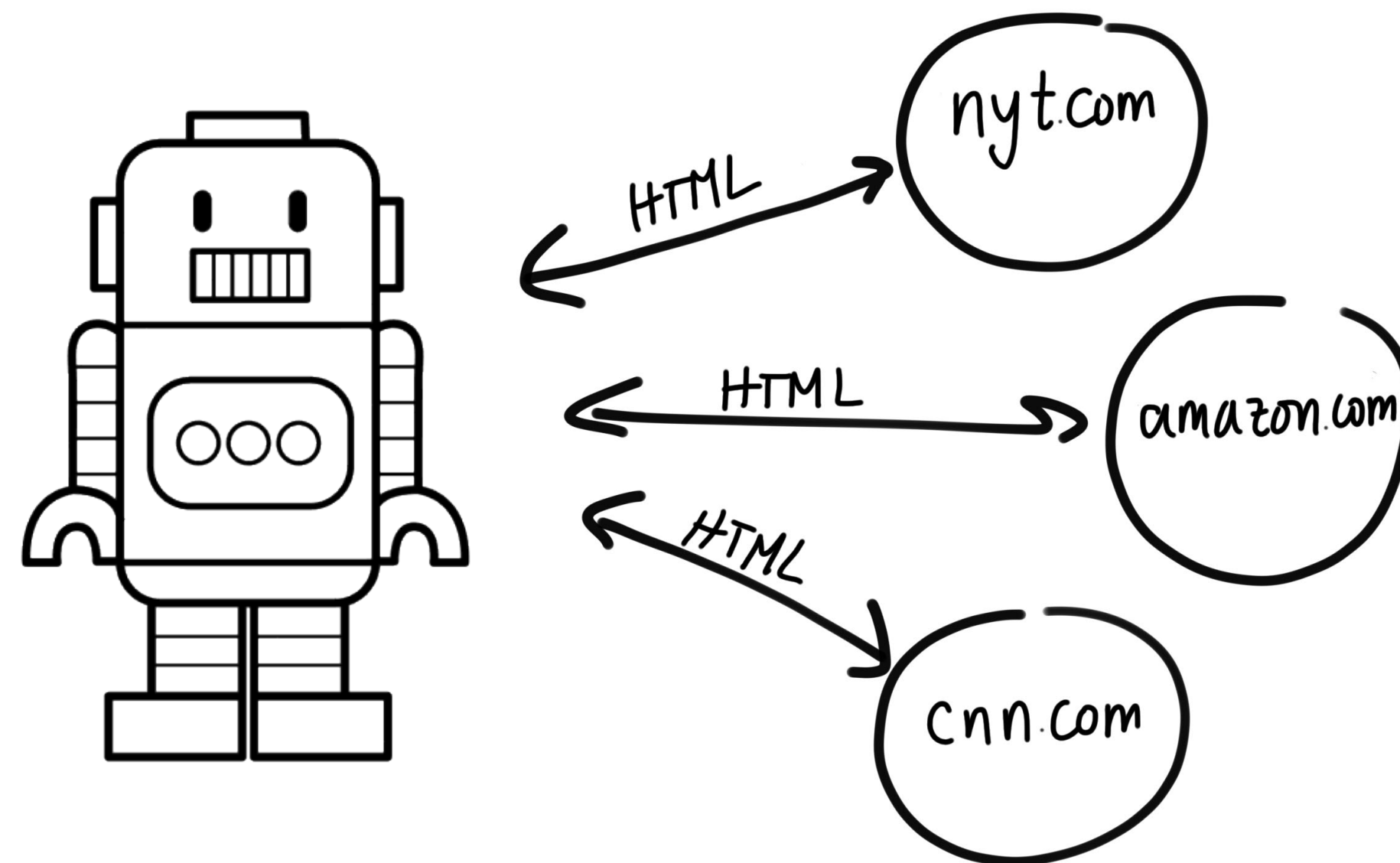


- Browser (HTML5)
 - HTML (structure and semantics)
 - CSS (style and UI behaviour)
 - JS + AJAX,
Socket interfaces (behaviour)
 - DOM
(the supporting data structure)
 - UI Events & callbacks
(mechanism for dynamic structuring of behaviour)

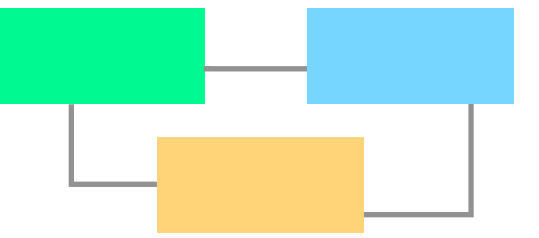




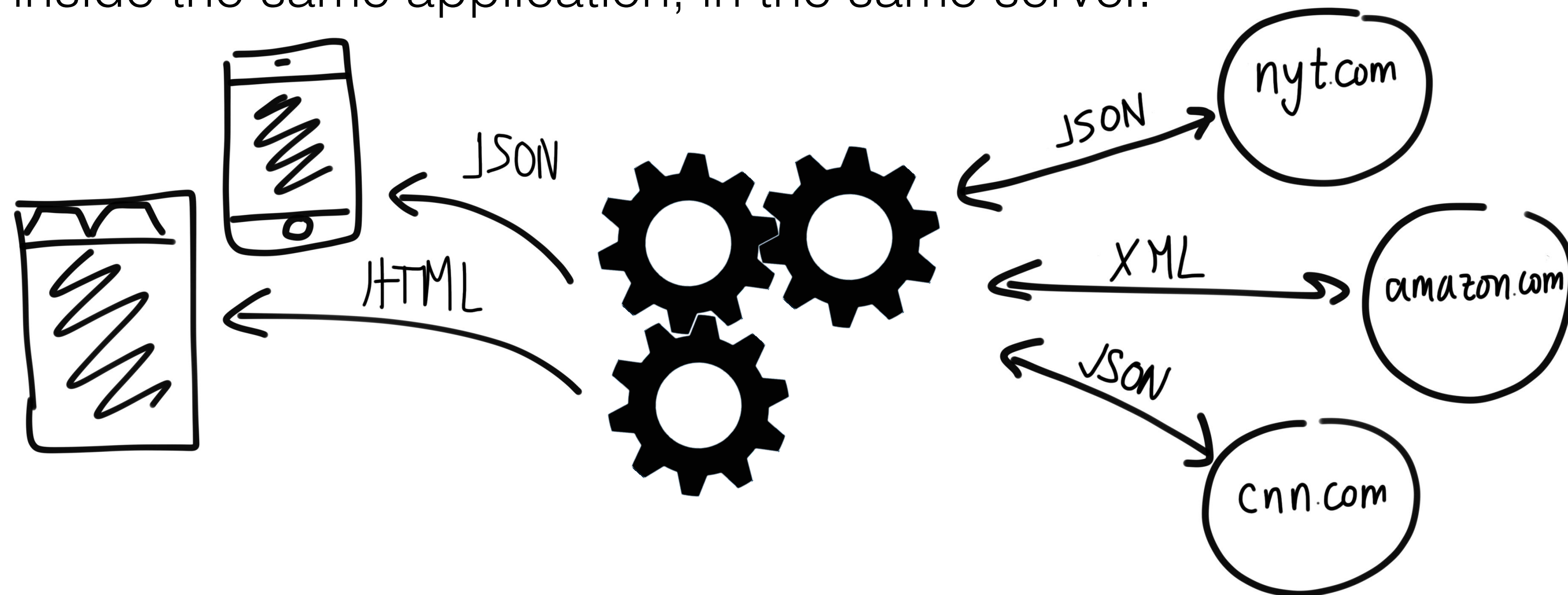
- Other kind of web clients (asking for HTML)
 - simulate web requests and sessions with web-servers
 - parse/crawl the results to extract information
 - should be built with web-services instead (if possible).



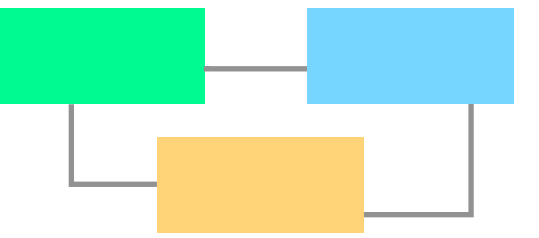
Service based web client architecture



- Provide web content based on data providing services
 - Data can be exchanged in “machine-friendly” formats. (e.g. JSON, XML)
 - Combined and refurbished in HTML or data formats.
 - Even reused inside the same application, in the same server.



Interconnection layer (low-level support)



- HTTP/1.1 (*Hypertext Transfer Protocol*) Protocol
 - Method: GET, HEAD, POST, PUT, DELETE (3 more)
 - Arguments (query string and body)
 - Multi-typed message body
 - Cookies
 - Return codes: 1XX, 2XX, 3XX, 4XX, 5XX
- Websockets (standard RFC6455 2011 - ws:// wss://)
 - supported by browsers and web servers to allow two way data communication between client and servers
- HTTP/2 approved May 2015.
 - Faster, compressed, and cyphered transmission of data
- TLS (successor of SSL)
 - Provides cryptographic support for web communications (handshake+symmetric crypto)

1xx Informational
100 Continue
101 Switching Protocols
102 Processing
2xx Success
200 OK
201 Created
202 Accepted
203 Non-authoritative Information
204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status
208 Already Reported
226 IM Used
3xx Redirection
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
308 Permanent Redirect
4xx Client Error
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot
421 Misdirected Request
422 Unprocessable Entity
423 Locked
424 Failed Dependency
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 Connection Closed Without Response
451 Unavailable For Legal Reasons
499 Client Closed Request
5xx Server Error
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected
510 Not Extended
511 Network Authentication Required
599 Network Connect Timeout Error

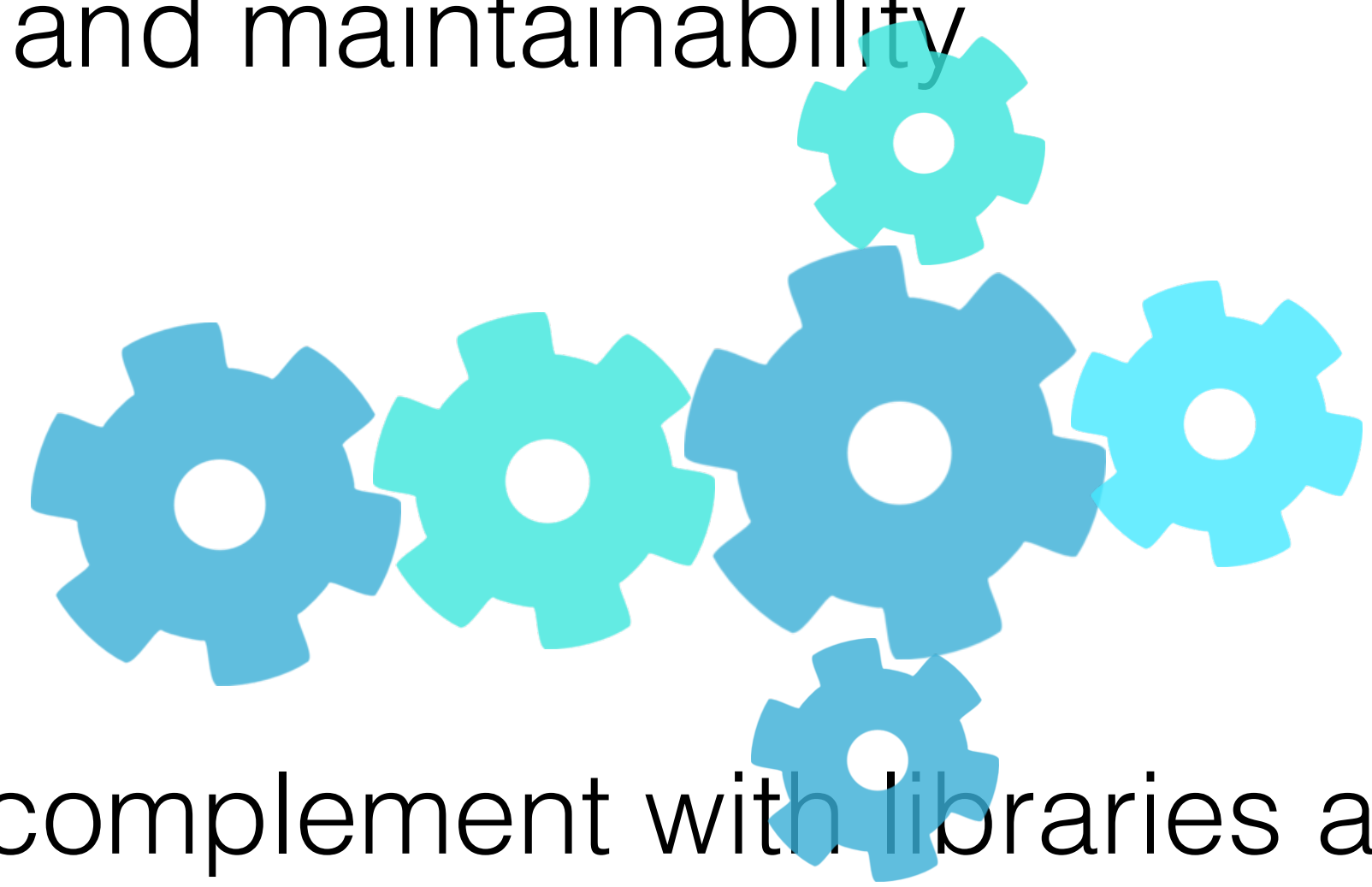
<https://httpstatuses.com/>

Web server / App server architecture

- Web servers handle HTTP requests, map URLs to local files, execute local scripts (e.g. CGI, PHP), or locally bound code (e.g. Servlets).
- App Servers modularly manage bound code, and associated resources (e.g. sessions, context, connections).
 - One web server, many applications.
 - Allows the assembly of components of applications (controllers, views, models)

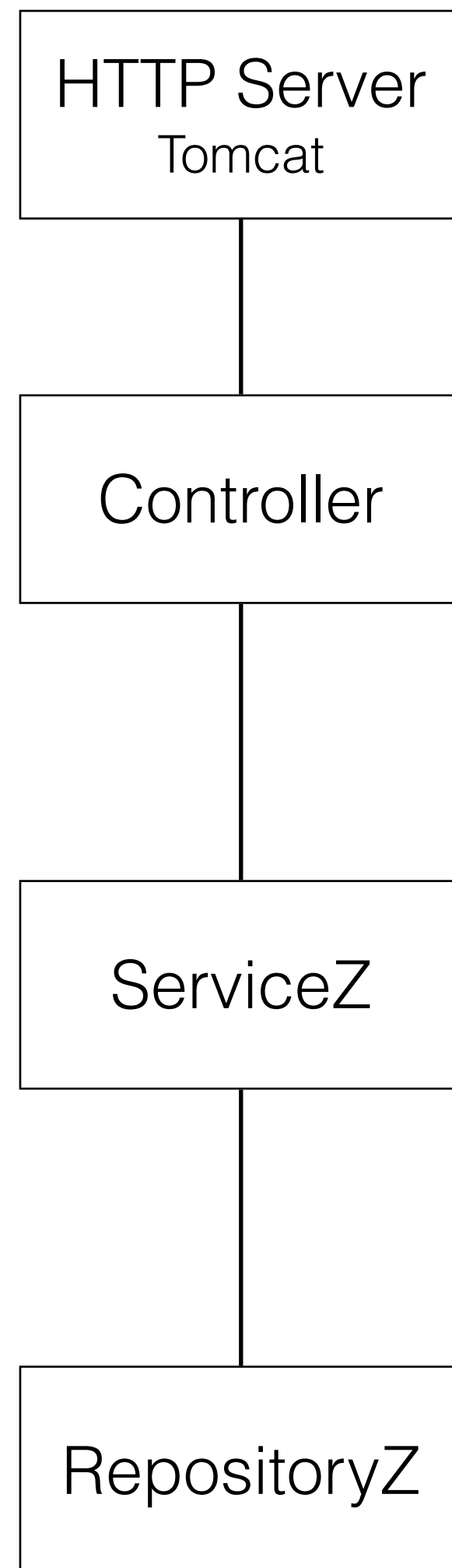
Web architectures, patterns and styles

- Increase the level of abstraction, reusability, and maintainability
 - Software Architectures
 - Architectural and design patterns
 - Architectural styles
- Software frameworks implement some, and complement with libraries and tools.
- User-defined pieces are required to specify the “core” logic, and configure general purpose code.
- All implement the “inversion of control” pattern.



An architecture built with Spring

Example of an Architecture built with Spring



- Spring is a component framework
- Resolves component dependencies by dependency injection
- Uses annotations to configure components

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

    // http GET :8080/api/projects/2/team
    @GetMapping( "/api/projects/{id}/team")
    fun teamMembersOfProject(
        @PathVariable id:String
    )
    = employees.teamMembersOfProject(id)

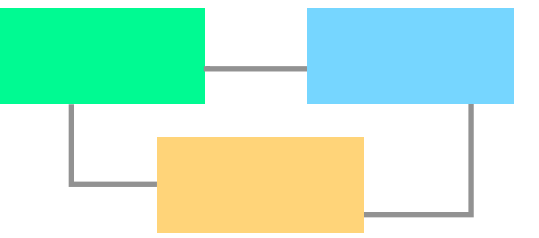
}

@Service
class EmployeeService(val employees:EmployeeRepository) {
    fun teamMembersOfProject(id:String) = employees.findAll()
}

interface EmployeeRepository : CrudRepository<Employee, Long>
```

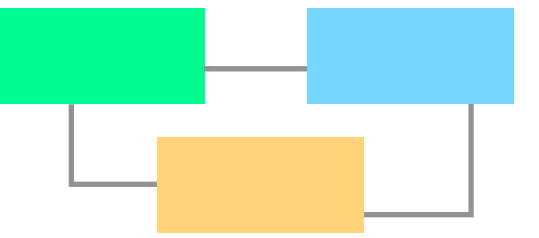
Service Based Architectures

Service Oriented Architectures



- Are the technological and methodological basis for building open-ended Internet Applications.
- Provide a model of distributed computation based on loosely coupled interactions.
- Define heterogeneous ecosystems of service implementations.
- Use implementation independent data formats (JSON, XML)
- Allows independent development of services by different vendors and technologies
- A service:
 - Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
 - Is self-contained
 - May be composed of other services
 - Is a “black box” to consumers of the service

<https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm>



- Web services are usually defined over HTTP protocol
- **SOAP** (*Simple Object Access Protocol*)
 - Operation based protocol (on HTTP) to implement web services (XML as format)
- **REST** (*Representational State Transfer*)
 - Resource based architectural style to implement web services over HTTP (or another connection protocol)

Micro Service Architectures

- Are an extreme interpretation of service based architectures.
- Have smaller grained services and interfaces.
- Provide independent and lightweight deployment.
- Allow flexible management (e.g. replicas).
- Based on a clear service ownership model (team responsible for all stages of dev&ops).
- Isolated persistent state (sometimes bad).

<https://martinfowler.com/microservices/>

Amazon's API Mandate (by Jeff Bezos, 2002)

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology you use.
- All service interfaces, without exception, must be designed from the ground up to be externalizeable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- The mandate closed with: Anyone who doesn't do this will be fired. Thank you; have a nice day!

MSc Topic on making micro-services better!

The Art, Science, and Engineering of Programming

Upcoming Submission Deadline: October 1, 2021

Robust Contract Evolution in a TypeSafe MicroServices Architecture

João Costa Seco¹, Paulo Ferreira², Hugo Lourenço³, Carla Ferreira⁴, and Lucio Ferrao⁵

The Art, Science, and Engineering of Programming, 2020, Vol. 4, Issue 3, Article 10

Submission date: 2019-10-01

Publication date: 2020-02-17

DOI: <https://doi.org/10.22152/programming-journal.org/2020/4/10>

Full text: [PDF](#)

Abstract

Microservices architectures allow for short deployment cycles and immediate effects but offer no safety mechanisms when service contracts need to be changed. Maintaining the soundness of microservice architectures is an error-prone task that is only accessible to the most disciplined development teams.

We present a microservice management system that statically verifies service interfaces and supports the seamless evolution of compatible interfaces. We define a compatibility relation that captures real evolution patterns and embodies known good practices on the evolution of interfaces. Namely, we allow for the addition, removal, and renaming of data fields of a producer module without breaking or needing to upgrade consumer services. The evolution of interfaces is supported by runtime generated proxy components that dynamically adapt data exchanged between services to match with the statically checked service code.

The model was instantiated in a core language whose semantics is defined by a labeled transition system and a type system that prevents breaking changes from being deployed. Standard soundness results for the core language entail the existence of adapters, hence the absence of adaptation errors and the correctness of the management model. This adaptive approach allows for gradual deployment of modules, without halting the whole system and avoiding losing or misinterpreting data exchanged between system nodes. Experimental data shows that an average of 69% of deployments that would require adaptation and recompilation are safe under our approach.

1. joao.seco@fct.unl.pt, Universidade NOVA de Lisboa, Portugal
2. paulo.ferreira@outsystems.com, OutSystems, Portugal
3. hugo.lourenco@outsystems.com, OutSystems, Portugal
4. carla.ferreira@fct.unl.pt, Universidade NOVA de Lisboa, Portugal
5. lucio.ferrao@outsystems.com, OutSystems, Portugal

The Journal

[About](#)
[Purpose and Operation](#)
[Boards](#)
[Awards](#)
[Publisher](#)
[Volumes](#)

Issues

[Volume 6, Issue 1](#)
[Volume 5, Issue 3](#)
[Volume 5, Issue 2](#)
[Volume 5, Issue 1](#)
[Volume 4, Issue 3](#)
[Volume 4, Issue 2](#)
[Volume 4, Issue 1](#)
[Volume 3, Essays](#)
[Volume 3, Issue 3](#)
[Volume 3, Issue 2](#)
[Volume 3, Issue 1](#)
[Volume 2, Issue 3](#)
[Volume 2, Issue 2](#)
[Volume 2, Issue 1](#)
[Volume 1, Issue 2](#)
[Volume 1, Issue 1](#)

For Authors

[Call for Papers](#)
[Timeline](#)
[Submissions](#)
[Copyright](#)

[Article feed \(atom\)](#)
[Article feed \(RSS\)](#)



MSc Thesis 3 - Contract Evolution in a Microservices Architecture

Context

Microservice architectures allow for short deployment cycles and immediate effects but offer no safety mechanisms for service contracts when they need to be changed. Maintaining the soundness of microservice architectures is an error-prone task that is only accessible to the most disciplined development teams. The strategy to evolve a producer service without disrupting its consumers is often to maintain multiple versions of the same interface and dealing with an explicitly managed handoff period and its inherent disadvantages.

In [1] the authors present a microservice management system that statically verifies service interface signatures against their references and supports the seamless evolution of compatible interfaces. This work defines a compatibility relation on types that captures real evolution patterns and embodies known good practices on the evolution of interfaces. Namely, it allows for the addition, removal, and renaming of data fields of a producer module without breaking or needing to upgrade consumer services. The evolution of interfaces is supported by runtime generated proxy components that dynamically adapt data exchanged between services to match with the statically checked service code.

Goal

The work proposed in [1] presents a theoretical work that addresses the contract (or API) evolution of microservices. The goal of this master thesis is to develop a distributed tool that checks and maintains compatibility between different services of a system. For compatible contract evolutions, the tool should create a lightweight proxy that automatically adapts services at runtime, which frees up the programmer from having to manually adapt either the existing microservices or the new one. The tool would advance current industrial practices provided by frameworks such as Protobuf, Thrift, and Avro. These tools support data schema evolution, but their support is more restricted and without the guarantees given by the approach formalised in [1].

[1] J. Costa Seco, P. Ferreira, H. Lourenço, C. Ferreira, and L. Ferrão. Robust Contract Evolution in a TypeSafe MicroServices Architecture, The Art, Science, and Engineering of Programming 4 (3).

Supervision: Carla Ferreira, João Costa Seco