

REST, gRPC, GraphQL, Webhooks

Dans quelles situations ?



ippon POSITIVE
TECHNOLOGY

VOXXED DAYS
LUXEMBOURG

REST, gRPC, GraphQL, Webhooks

Dans quelles situations ?



REST, gRPC, GraphQL, Webhooks

In what situations?



\$ whoami

François Delbrayelle (@fdelbrayelle)   

Cht'Ipmon @ **Ippon** Lille since 2019

- ***Senior Software Engineer***
- ***Tech Community Ambassador***
- **Trainer**
- Blog Team Member (blog.ippon.fr)

Contributor @ OSS / **JHipster**







27/9/21
LILLE

CHALEUREUX!

ARTHUR, TAS
GAGNE!
VAN
STRATEGY
OFFICER

ACCOMPAGNEMENT

ON FOUT DES BAFFES!



CRAFT

ADEO

NEOLIA

VEROY
MERLIN

EDF

NEXITY

CAMPÉS
LAFAYETTE

ACCUEIL
SEGAIGRES

FORMATION

SITE
WEB

CONTRIBUTION



MEEETUP



DENFEST



PARLEAGE



DE

Design



CONNAISSANCES

COMMUNAUTÉS



CLOUD

NORD



JAVA



DESIGN



AURA DANS L'IT



POURQUOI?

VOILÀ COMMENT...

POSITIVE TECHNOLOGY

SCALE ACCÉLÉRALE INNOVATE!

2024



OLIVIER SAMPSON-



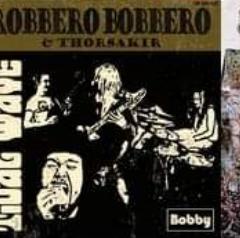
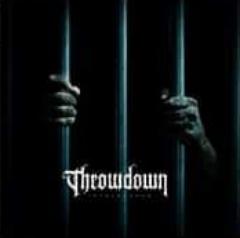
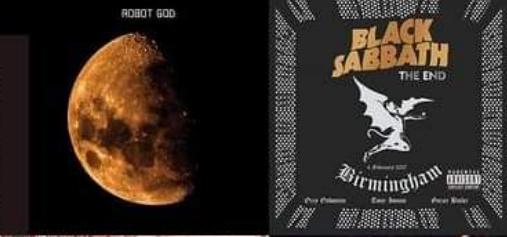
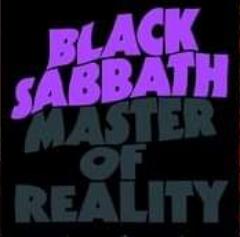
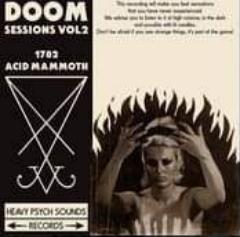
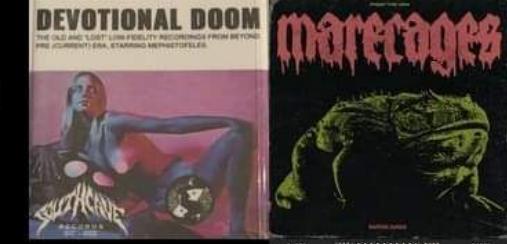
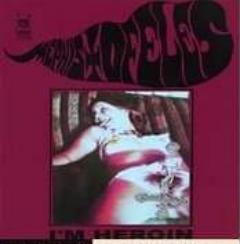












Disclaimer

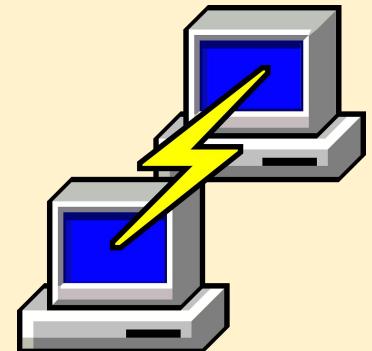
-  Pas de mise en avant d'une techno par rapport aux autres
-  Pas une présentation détaillée de chaque techno
-  Pas d'ordre logique entre les technos abordées
-  Pas de biais cognitifs (au moins essayer)
-  Pas abordé : *pub/sub* et *queues*
-  Pas de *silver bullets*





Make actors communicate

📜 A brief history of client / server communication



Client / Server Communication

Client / Serveur: network architecture

Sockets

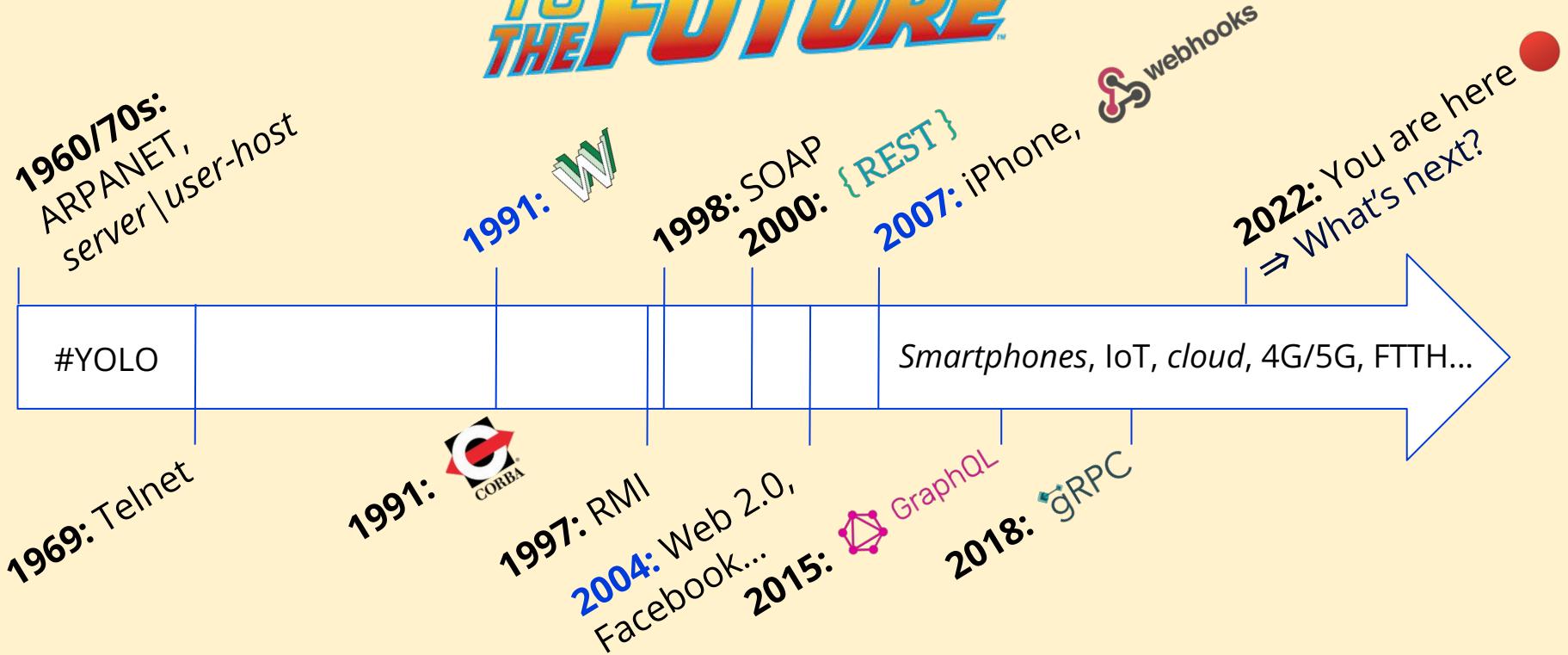
TCP/IP: common transport protocol

Inter-Process Communication (IPC)

Non-Functional Requirements



BACK TO THE FUTURE



```
<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

    <soap:Header>
    ...
    </soap:Header>

    <soap:Body>
    ...
        <soap:Fault>
        ...
        </soap:Fault>
    </soap:Body>

    </soap:Envelope>
```



2010...



REST IS THE NEW SOAP

2015...

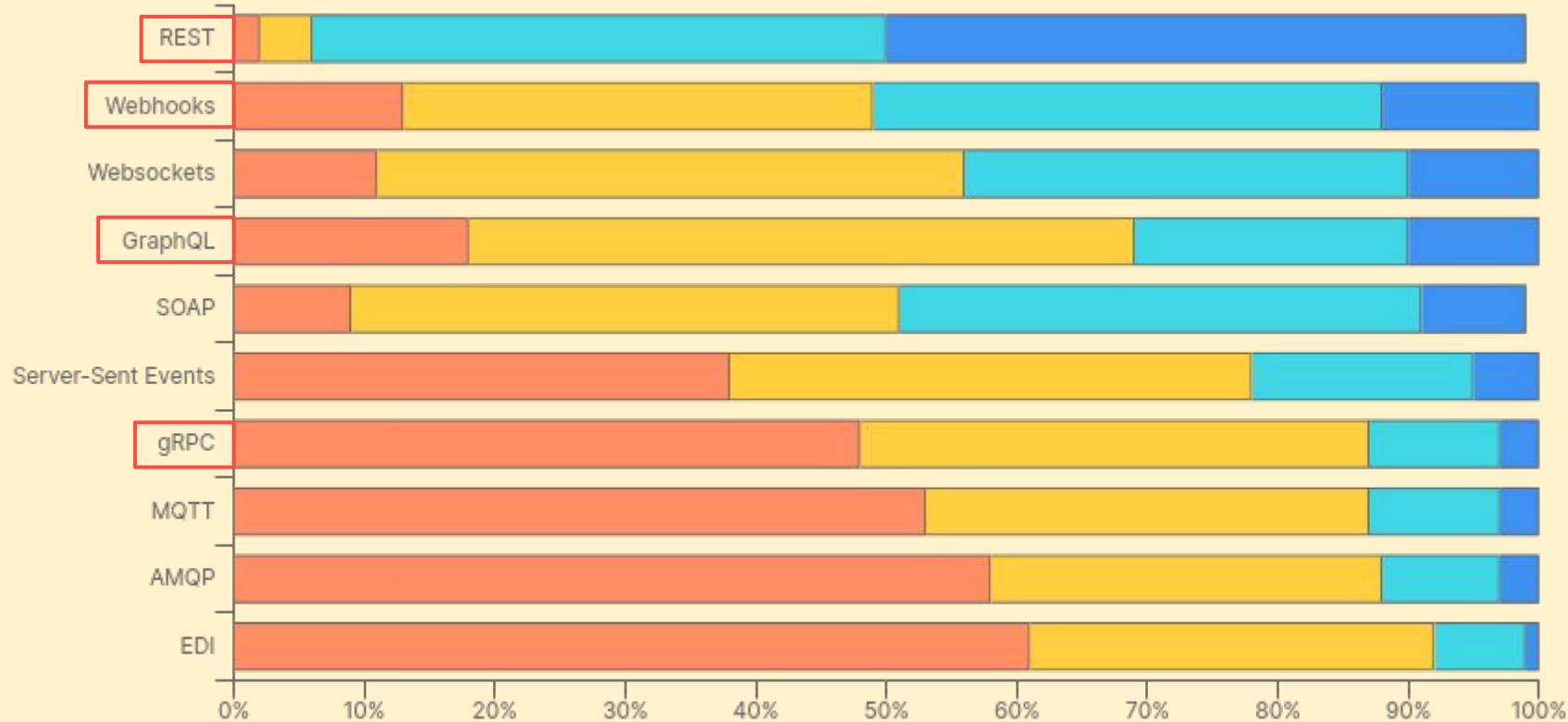


**GRAPHQL IS THE NEW REST
(IT'S MORE VERSATILE!)**

2018...



**GRPC IS THE NEW GRAPHQL
(IT'S FASTER!)**



Source : Postman's 2021 State of the API Report (<https://www.postman.com/state-of-api>)

Légende : uses and loves - = uses - = knows but doesn't use - = never heard





C'EST UNE BONNE SITUATION ÇA ?

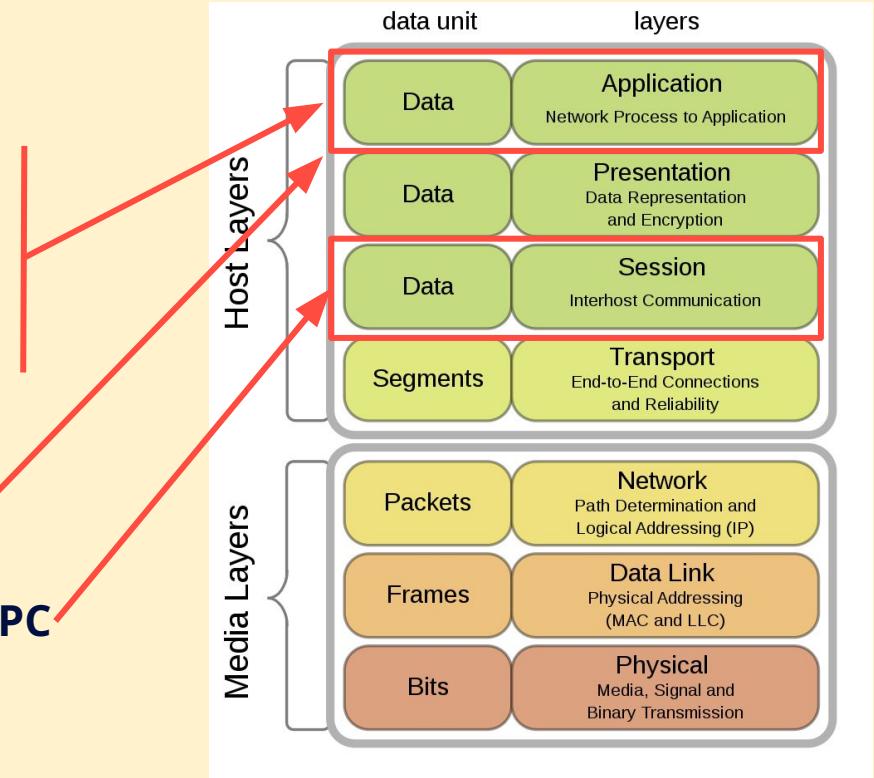
Different typologies

REST: architecture (often with **HTTP**)

GraphQL: language (**HTTP POST**)

Webhooks: **HTTP POST**

gRPC : framework based on **HTTP/2 et RPC**

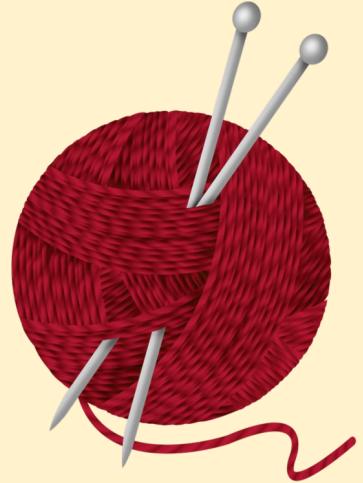


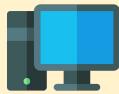
Source (picture): Wikipédia



Case Study

Extract from an e-commerce platform with:
Web / Mobile Applications
PIM (Product Information Management)





E-Commerce



Mobile



Store Checkout



Store



Marketplace



Product API



Price API



Stock API



Asset API



What is an API?

Application Programming Interface

Client / Server

Exchange informations

Use standards

Proper lifecycle



{ REST }



What is REST?

REpresentational State Transfer

Architecture for distributed hypermedia system

Thought by **Roy Fielding** in 2000:

Architectural Styles and the Design of Network-based Software Architectures

Can be based on **HTTP/1.1 (RFC 2616)**



Key Concepts

Ressource

- Entity, model, identifiable concept, concrete or abstract
- URI (*Uniform Resource Identifier*) = identity
- REST Service binds URLs to URIs

HTTP Verb

- Idempotent: GET, PUT, DELETE
- POST, PATCH...

Representation of resource state: exchange format (JSON, XML, HTML, raw text...)



Architectural Constraints

Client / Server

Stateless

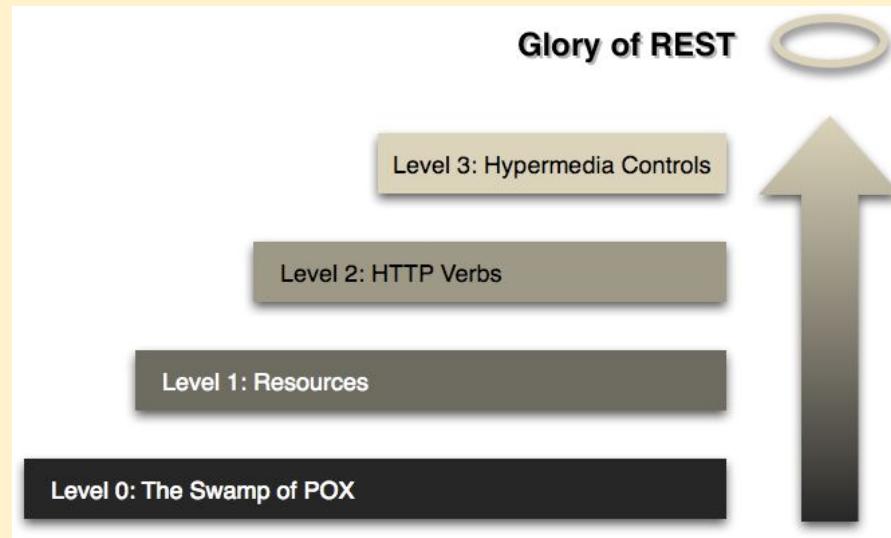
Cache

Uniform Interface

Layered System



Richardson Maturity Model



Source : <https://www.martinfowler.com/articles/richardsonMaturityModel.html>





Expose business value with hypermedia

Business centric

Beware anemic APIs

Don't expose directly databases

⇒ Use **hypermedia** (**HATEOAS** = *Hypermedia as the Engine of Application State*)



REST Call Example

```
GET /characters/42 HTTP/1.1  
Host: api.h2g2.com  
Content-Type: application/json
```



```
{  
  "id": 42,  
  "name": "Arthur Dent",  
  "links": [  
    {  
      "href": "42/messages",  
      "rel": "messages",  
      "type": "GET"  
    }  
  ]  
}
```



{ REST }

Pros

- Based on web standards
- Universal support (browsers)
- A lot of use cases
- Well-known by developers
- CRUD
- HATEOAS

Cons

- Over|underfetching
- 1 to 1
- HTTP/1.1 sometimes limited
- Payload size





Why choose REST?

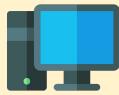
Motivations:

- Wide web browser support
- Business data / behaviors exposition
- Headless
- CRUD-based simple actions

Use cases:

- Resource-driven applications
- Web applications
- Mobile applications
- Microservices

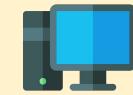




E-Commerce



Mobile



Store Checkout



Store



Marketplace



Product API



Price API

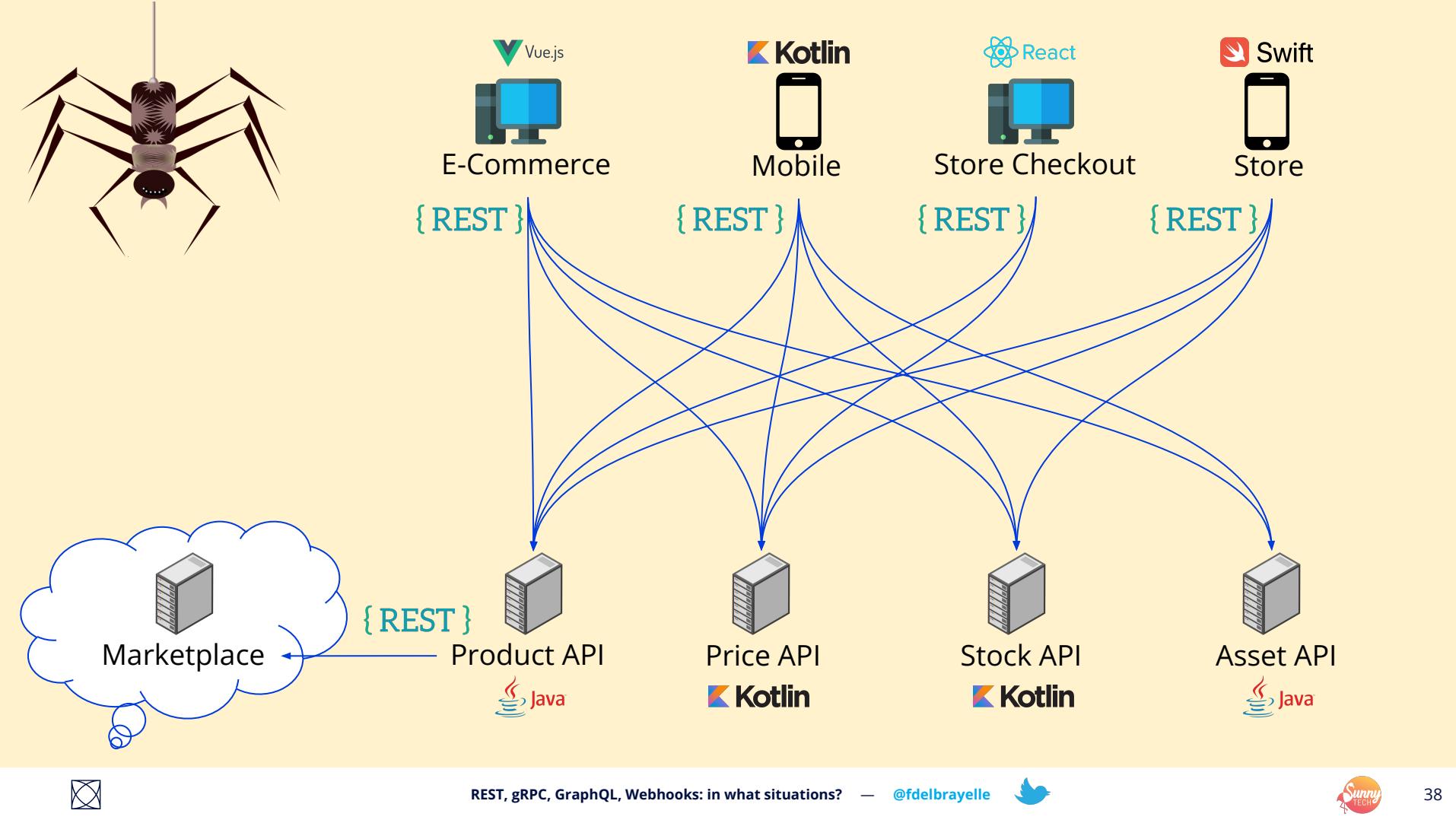


Stock API



Asset API







What is gRPC?

Framework created by **Google** in **2016**

Open-sourced (Apache 2.0) : <https://github.com/grpc/grpc>

Part of the **CNCF Foundation**

Based on **RPC protocol (*Remote Procedure Call*)**

3 main implementations: GRPC-JAVA, GRPC-GO et GRPC-C (from which derive C++, Python, Ruby, Objective C, PHP and C# ones)

Java : <https://github.com/grpc/grpc-java> (through Maven/Gradle plugins)





HTTP/2

Standard since 2015 (RFC 7540)

Used for transport

Streams composed of **frames** (with specific formats and sizes)

Multiplexing: Multiple frames in a request / connection

45,3 % of websites are using it in June 2022

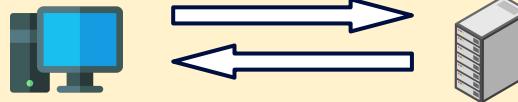
<https://w3techs.com/technologies/details/ce-http2>



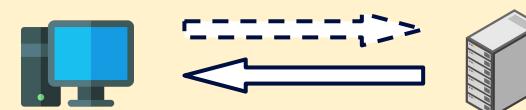


Communication Modes

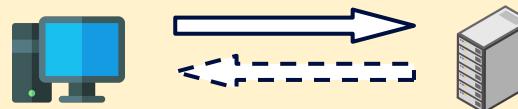
Unary



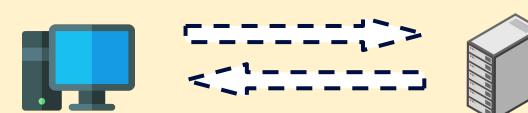
Client Streaming



Server Streaming



Bidirectional Streaming





Protocol Buffers

Interface Description Language (IDL) to define **messages** and **services**

Agnostic

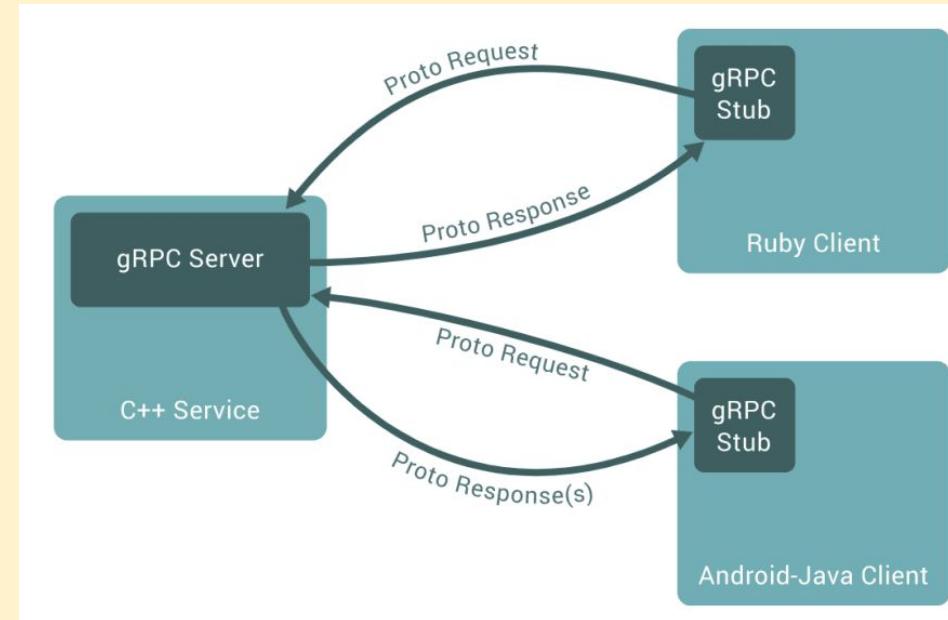
Interface independent from implementation

Binary data and (de)serialization (more efficient than JSON):

- less bandwidth (binary = small payloads)
- closer to the machine representation of the data
- less CPU intensive



Client / Server Example



Source: <https://grpc.io/docs/what-is-grpc/introduction/>



```
// todo.proto
syntax = "proto3";

package todoPackage;

service Todo {
    rpc createTodo(TodoItem) returns (TodoItem);
    rpc readTodos(voidNoParam) returns (TodoItems);
    rpc readTodosStream(voidNoParam) returns (stream TodoItem);
}

message voidNoParam {}

message TodoItem {
    int32 id = 1;
    string text = 2;
}

message TodoItems {
    repeated TodoItem items = 1;
}
```



```
// server.js

const grpc = require("grpc");
const protoLoader = require("@grpc/proto-loader");
const packageDef = protoLoader.loadSync("todo.proto", {});
const grpcObject = grpc.loadPackageDefinition(packageDef);
const todoPackage = grpcObject.todoPackage;

const server = new grpc.Server();
server.bind("0.0.0.0:40000", grpc.ServerCredentials.createInsecure());
server.addService(todoPackage.Todo.service,
{
    "createTodo": createTodo,
    "readTodos": readTodos,
    "readTodosStream": readTodosStream
});
server.start();

const todos = [];
function createTodo(call, callback) {
    // Impl
}
```



```
// client.js
const client = new todoPackage.Todo("localhost:40000",
  grpc.credentials.createInsecure());

const text = process.argv[2];
client.createTodo({
  "id": -1,
  "text": text
}, (err, response) => {
  console.log("Received from server " + JSON.stringify(response))
});
```





CRUD (REST) vs Actions (RPC)

```
POST /users/501/messages HTTP/1.1  
Host: api.example.com  
Content-Type: application/json  
  
{"message": "Hello!"}
```



```
POST /SendUserMessage HTTP/1.1  
Host: api.example.com  
Content-Type: application/json  
  
{"userId": 501, "message": "Hello!"}
```



Pros

- HTTP/2
- ProtoBuf
- Progressive feedback
- Actions oriented
- More complex procedures

Cons

- Schema
- ProtoBuf (hard to sniff on a network)
- Learning curve
- Not ready (yet) for the web



	{ REST }	gRPC
Protocols	HTTP/1.1 HTTP/2 but... (request / response)	HTTP/2 + RPC (client / response)
Schema	No but...	Yes (.proto)
Formats	JSON XML PDF HTML ...	Binary (Protocol Buffers)
Versioning	Yes	Yes
Streaming/bidirection	No but...	Yes
Browsers Support	Yes	No but gRPC-web + proxy
Code Generation	Swagger / OpenAPI / Postman ...	protoc (native in gRPC)
Security	JWT / OAuth	SSL/TLS / ALTS / JWT / OAuth





Why choose gRPC?

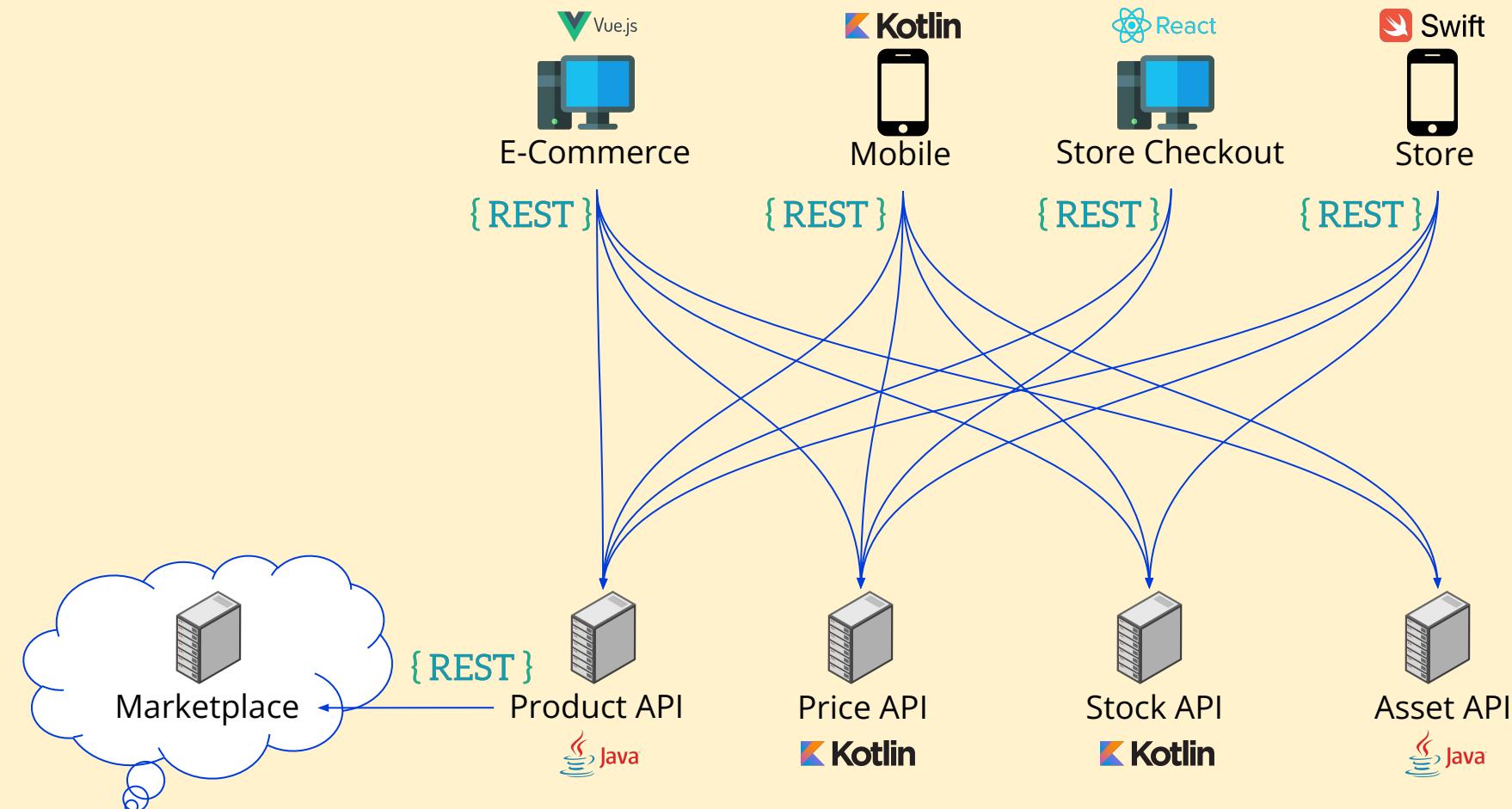
Motivations:

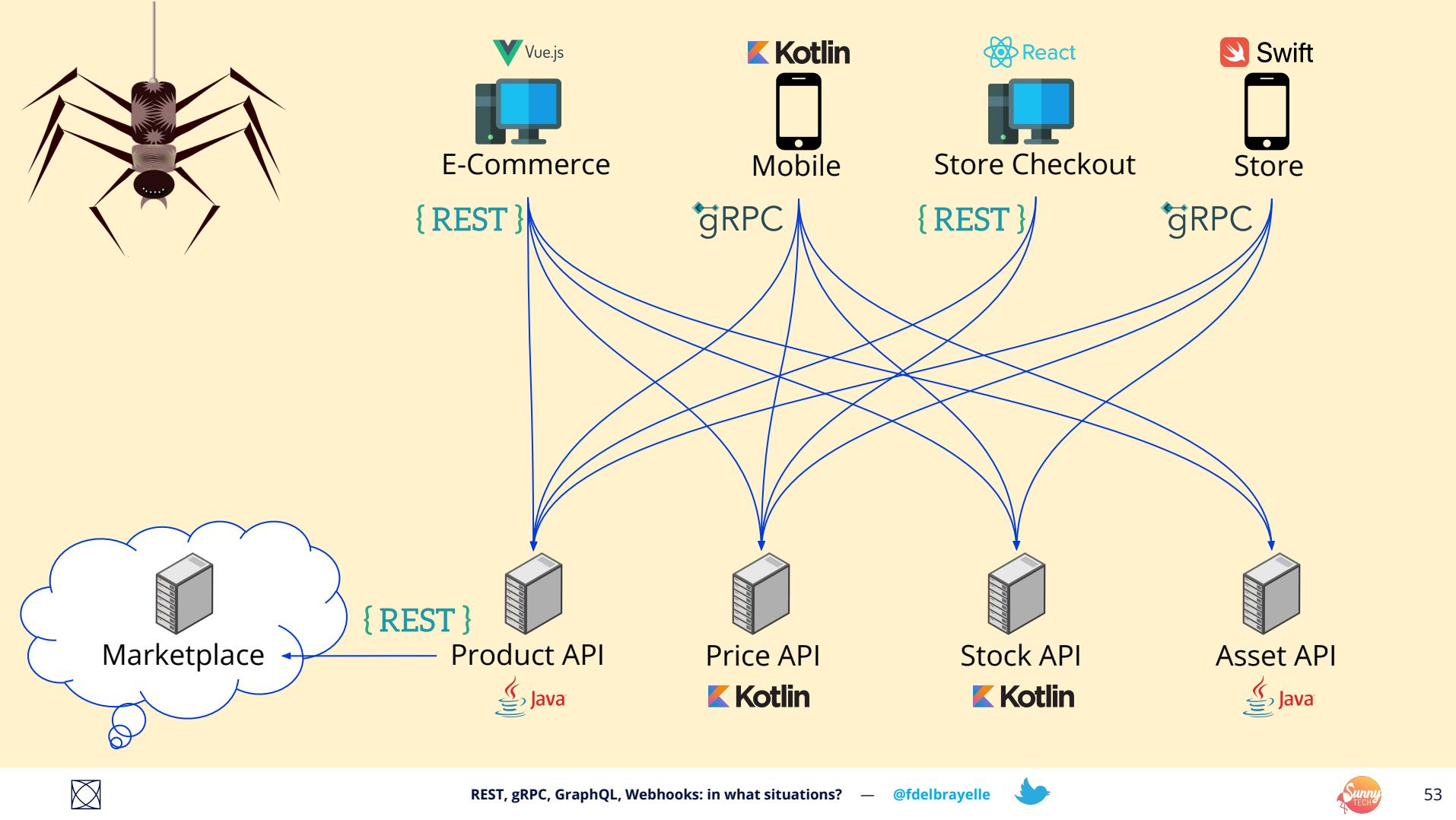
- Performance
- Payloads size
- Bilateral communication and real-time streaming
- More complex actions

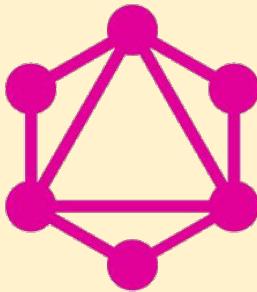
Use cases:

- Microservices
- Mobile applications
- Internet of Things (IoT)
- Multi-languages systems









GraphQL





What is GraphQL?

Language created by **Facebook** in **2012**

Open-sourced in **2015**: <https://github.com/graphql>

Extension .graphql (s)

Responses in **JSON**

Specification: <https://spec.graphql.org>

*Can be transported by **HTTP POST***

Client / Server (Apollo Server, Spring GraphQL...)





Key Concepts

Schema (+ introspection)

Type, interface, input, fields, arguments

Scalar type (Int, Float, String, Boolean, ID, scalar Date...), enum, fragment

Operation : **query** (parallel read), **mutation** (serial create/update/delete),
subscription (streaming) + variables

Directive : @include | skip (if: \$isAwesome)

Non-nullable with !





Patterns

Composite pattern: Compose multiple calls (API, external service, *in-memory...*) to aggregate data - API Gateway / BFF (*Back-end for Front-end*)

Proxy pattern: Add features to an API (often an old legacy one)

Facade pattern: Simplify call to an API





Graphs... Graphs everywhere!

« It's Graphs All the Way Down »

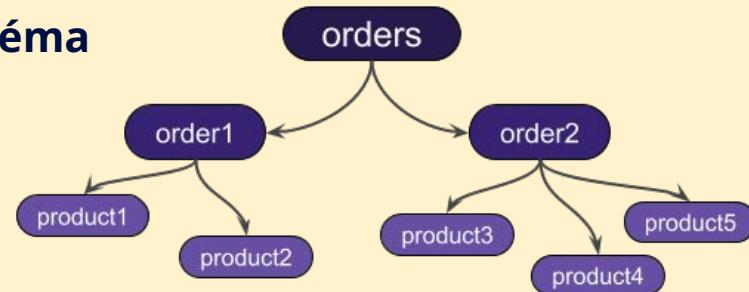
Domaine métier = graphe défini par un **schéma**

Différents types (de nœuds)

Connexions entre les nœuds

Références de types à types

<https://graphql.org/learn/thinking-in-graphs>





GraphQL Call Example

```
# Query
{
  orders {
    id
    products {
      product {
        id
        name
      }
      quantity
    }
    amount
  }
}
```



```
{
  "data": {
    "orders": [
      {
        "id": 0,
        "products": [
          {
            "product": {
              "id": 1,
              "name": "Crème solaire"
            },
            "quantity": 62
          }
        ],
        "amount": 100.00
      }
    ]
  }
}
```





GraphQL

Pros

- No over|underfetching
- Composition
- Limit client perimeter
- Auto-generated documentation
- Most languages supported
- Operation registry

Cons

- Errors management
- Up|download not native
- 1 platform = 1 implementation
- Learning curve



	{ REST }		 GraphQL
Architecture	<i>Server driven</i>	<i>Server driven</i>	<i>Client driven</i>
Endpoints	n	n	1
Schema	No but...	Yes	Yes
Operations	<i>Create, Read Update, Delete</i>	Functions	<i>Query, Mutation, Subscription</i>
Formats	JSON XML HTML...	Binary (ProtoBuf)	GraphQL & JSON
Over underfetching	Yes	Yes	No
Versioning	Yes	Yes	No
Cache	GET et POST	?	Yes but...
Security	JWT / OAuth	SSL/TLS / JWT / OAuth	Yes but...





Why choose GraphQL?

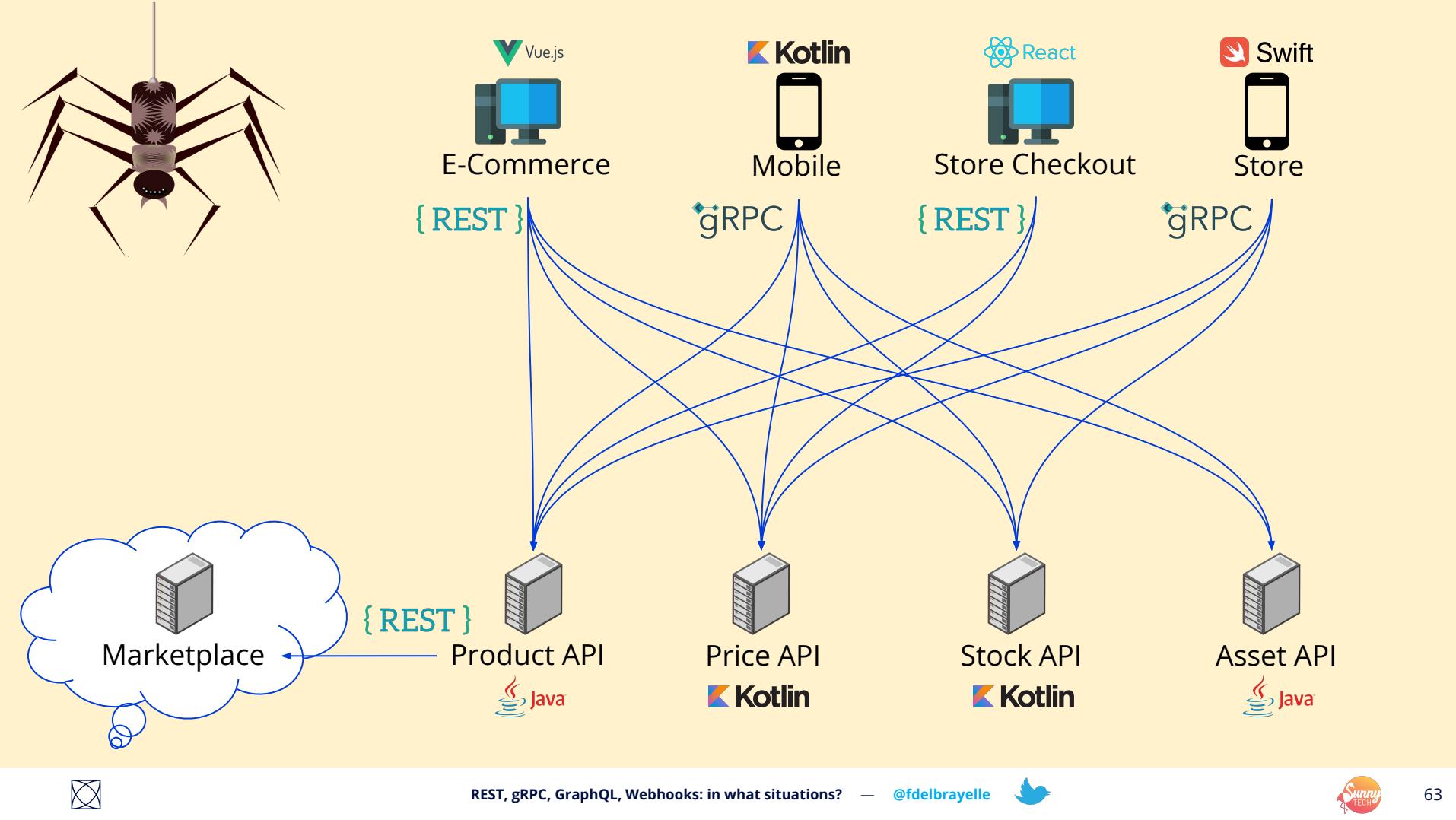
Motivations:

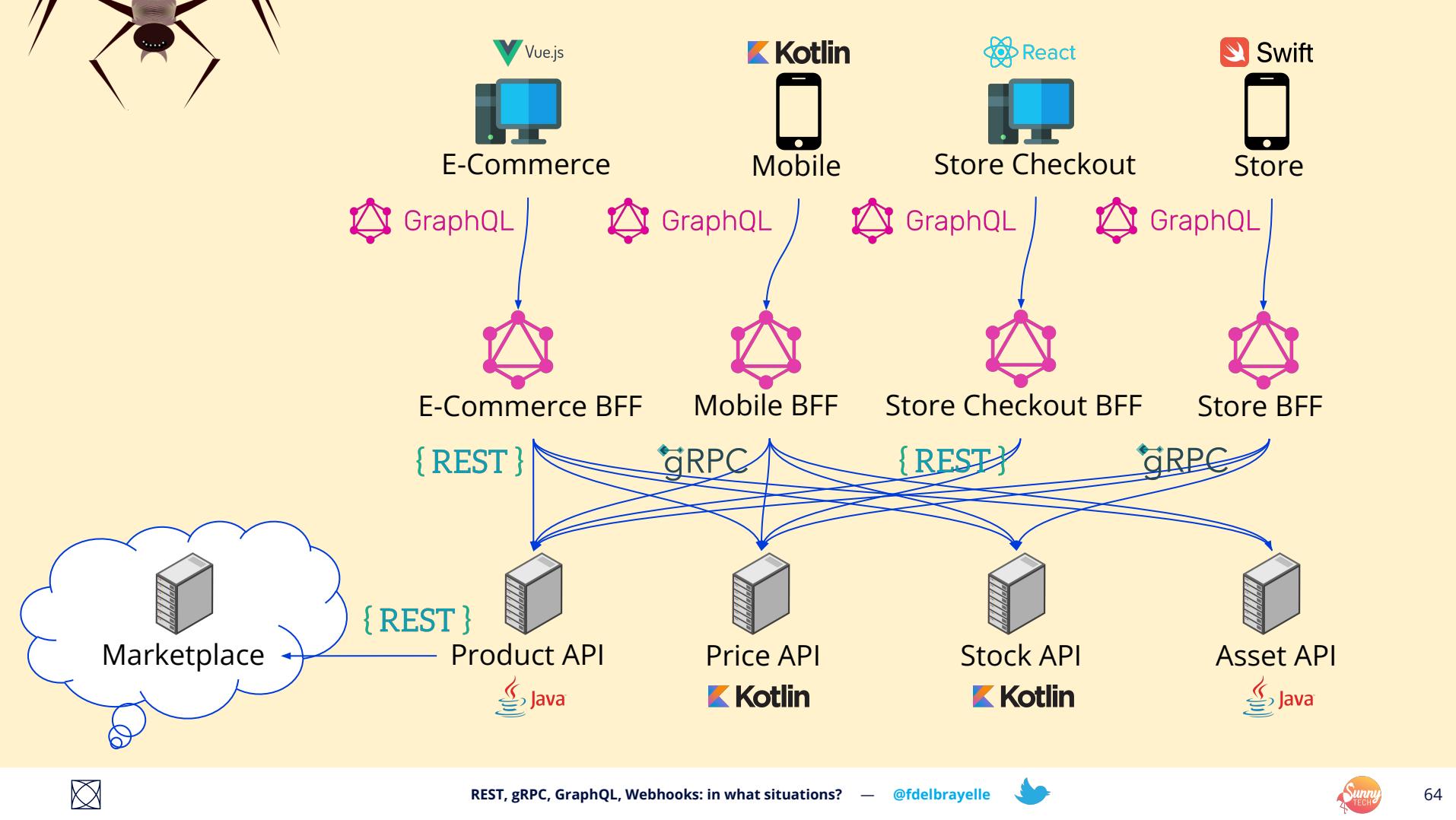
- More control on retrieved data (client-driven)
- Use multiple data source
- Limited bandwidth

Use cases:

- Big web applications (e-commerce...)
- Mobile applications
- Legacy applications with old and poor maintainability APIs
- Production use cases examples (GitHub...)









webhooks



What are Webhooks?

Coined in **2007** by **Jeff Lindsay**

NOT a standard

Event types (event-driven API)

(Un)subscription

Send data through HTTP POST to a **callback URL** in response to an **event type**



Do not mix up...

Webhooks: Send a message in response to a server event to a client through a callback URL containing a secret - One-way communication

Websockets: Two-way low-latency client/server communication, client opens a connection, scalability challenges, decrease HTTP requests amount...

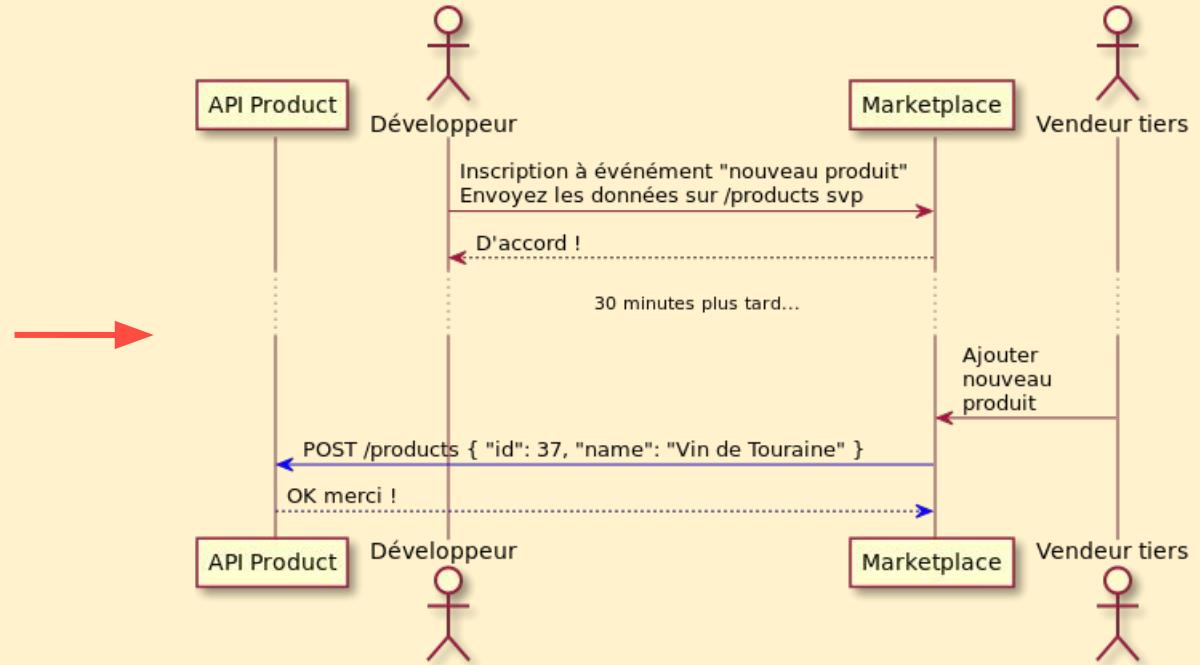
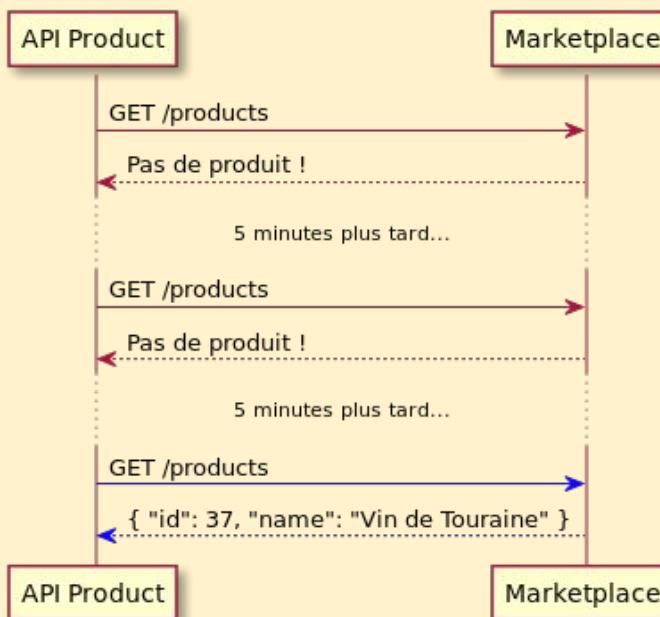
Server-sent events (SSE): Events transmissions to a web client from a server - One-way communication

...

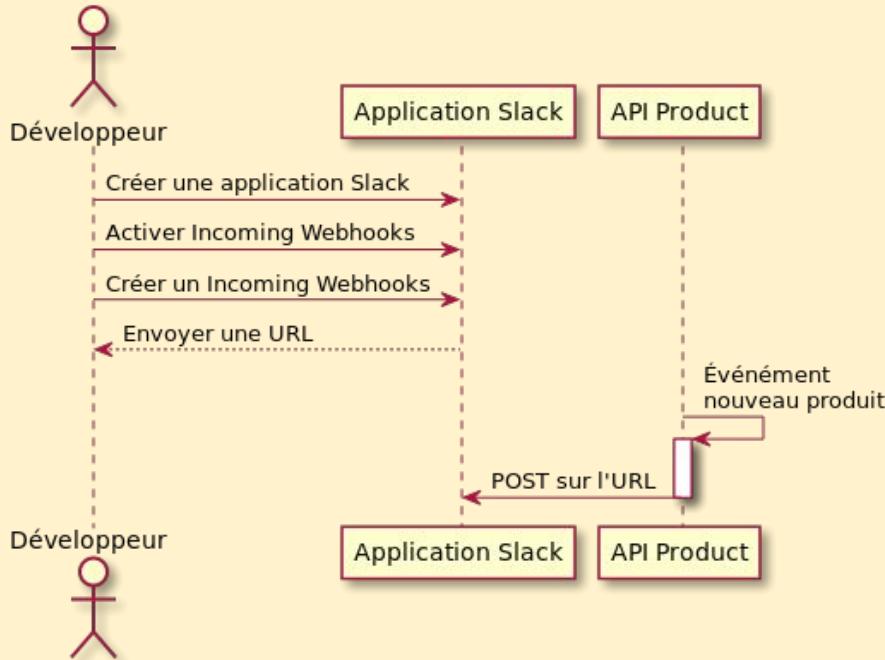




Pull vs Push



Webhooks Call Example



POST

```

https://hooks.slack.com/services/T00000000/B00000000/xxxxxxxxxxxxxxxxxxxxxx
Content-type: application/json
{
  "text": "New product added!"
}

```





Pros

- Push instead of pull
- Send data outside a system

Cons

- External clients must subscribe
- 1 webhook = 1 event (“noise”)
- Server down = lost data
- Less control (data flow)



	{ REST }	 webhooks
Mecanism	<i>pull</i> data (<i>request based</i>)	<i>push</i> data (<i>event based</i>)
Communication	Two-way (⚠️ not streaming)	One-way
Real-time	No but...	Yes
HTTP Verbs	GET POST PUT DELETE PATCH ...	POST GET (rare)
Formats	JSON XML PDF HTML ...	JSON XML
Payloads size	Can be huge	Small, in general





Why choose webhooks?

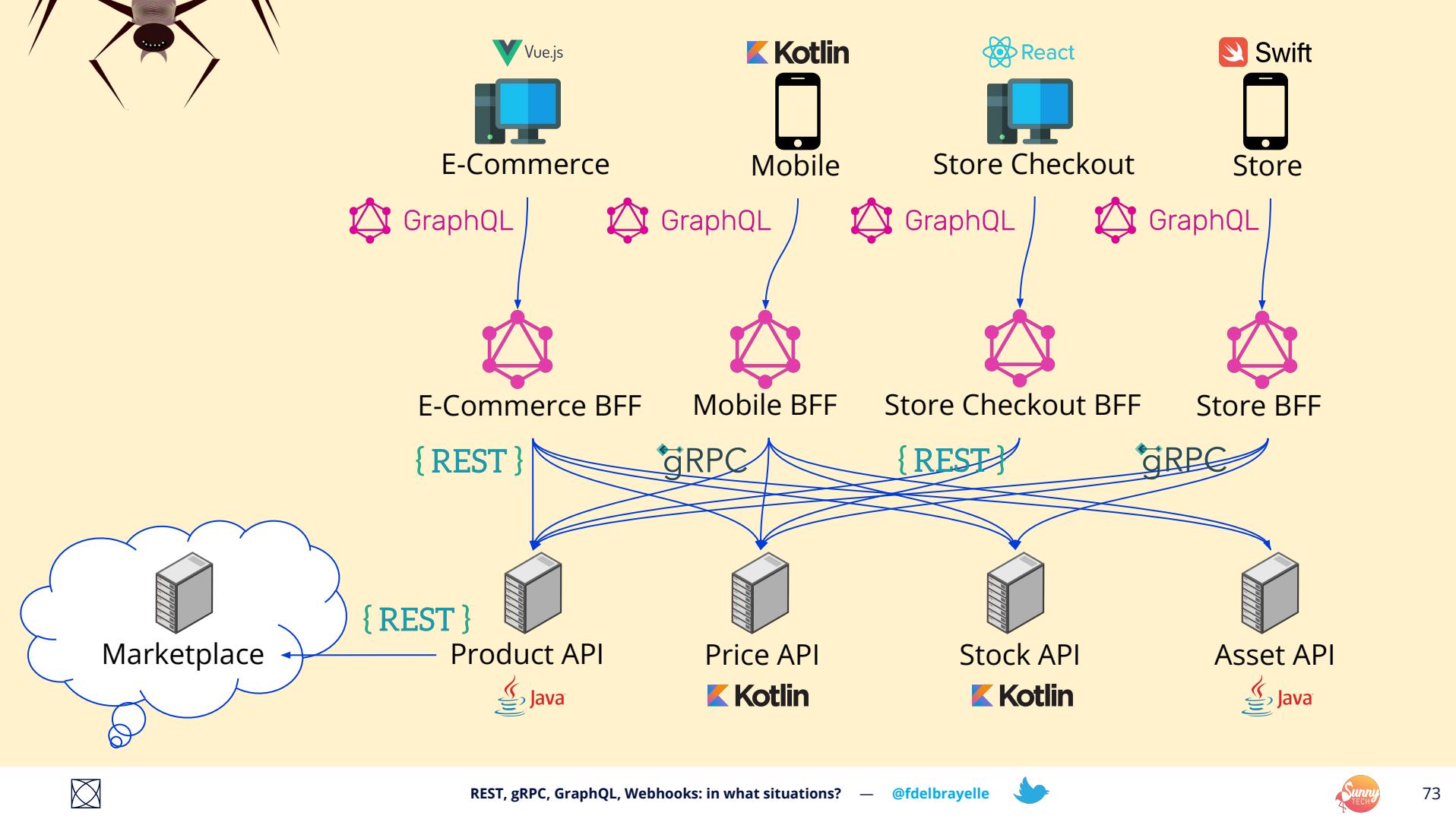
Motivations:

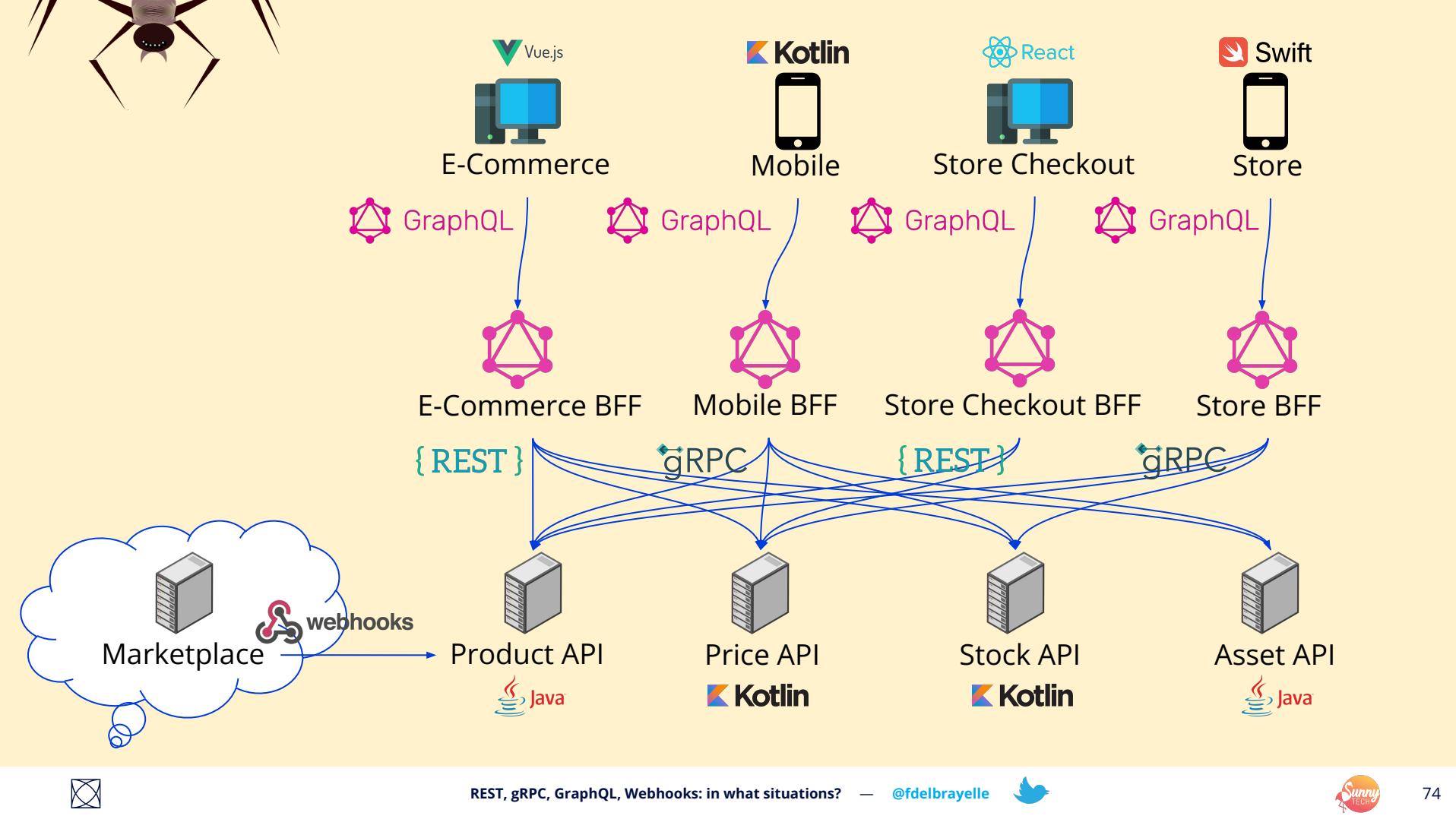
- Need to push instead to pull
- Trigger punctual actions
- Open a system externally

Use cases:

- Real-time systems synchronization
- Notifications (alerts, informations...)







So, is that a good situation?

🤔 It depends!™ There is no good or bad situation...





Diversity of communication modes

REST: Stateless Architecture - Business data transfer - Standards (HTTP, URI, JSON) - Need to iterate quickly - Web - CRUD - File up | download

gRPC: RPC Framework + HTTP/2 + ProtoBuf - Need a well defined data amount or a routine operation - Consumers with limited power and resources (mobile, IoT) or microservices

GraphQL: Query Language - Compose multiple data sources - Client-driven - Avoid over|underfetching

Webhooks: HTTP POST - Auto data updates in response to events - Push vs Pull





The final word: stay pragmatic!

⚒️ **With the right tools**, you will work

SOS **On communities**, you will rely on

❤️ **Multiple technologies at the same time**, you will consider

👨‍💻 **That your team masters the chosen technologies**, you will ensure

🟢 **That the systems in place respond with the given parameters**, you will check

↔️ **Business needs with the appropriate approach**, you will align





Thank you! 🙏

Do you have any questions?

www.ippontech.fr

contact@ippontech.fr — +33 1 46 12 48 48 — @ippontech