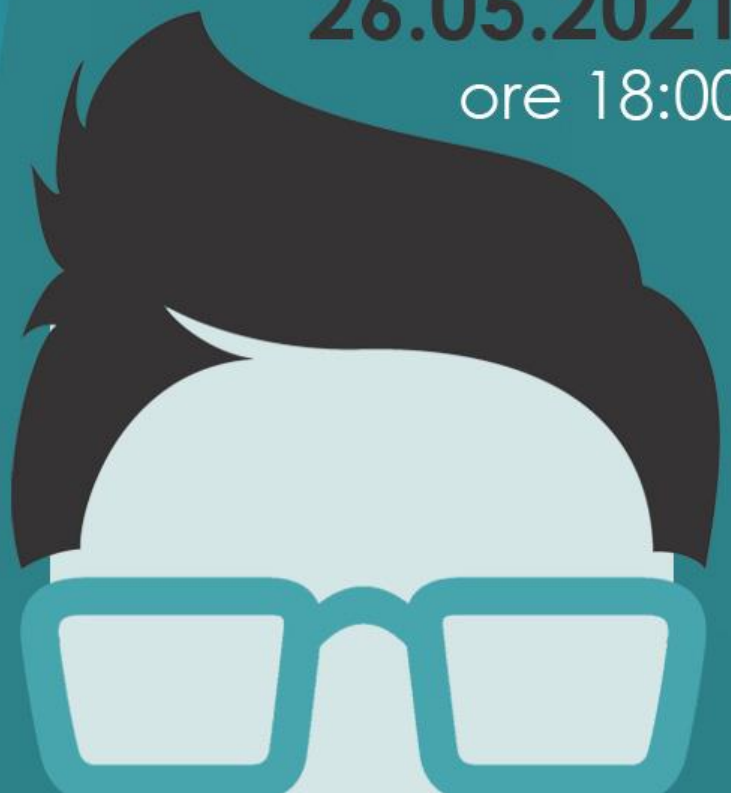
# NextCODE
Generation

AD ALTO TASSO TECNOLOGICO

26.05.2021
ore 18:00

## Ma quindi cos'è questo GraphQL di cui si sente tanto parlare?

Tiziano Pessa e Lorenzo Spinelli

## *Tiziano Pessa*

CTRL+C/CTRL+V'ing @ ( )REGISTER

*pixel13*

*tizianopessa*

*tizianopessa*

*@tizianopessa*

## *Lorenzo Spinelli*

I write code @ ( )REGISTER

*spyna*

*SpinaLorenzo*

*lorenzospinelli*

*@spyna*

REGISTER

***We're hiring!*** 

Sviluppatori PHP/Java/Javascript
Fully remote

https://tinyurl.com/register-is-hiring

# Il newsfeed di Facebook

- Nel 2012, la diffusione dei *mobile devices* spinge Facebook a scrivere una app nativa per iOS

{ REST }

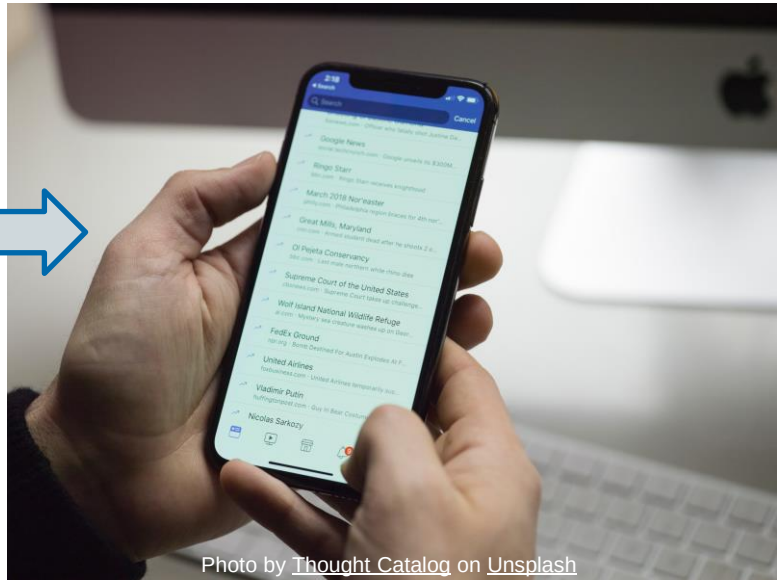
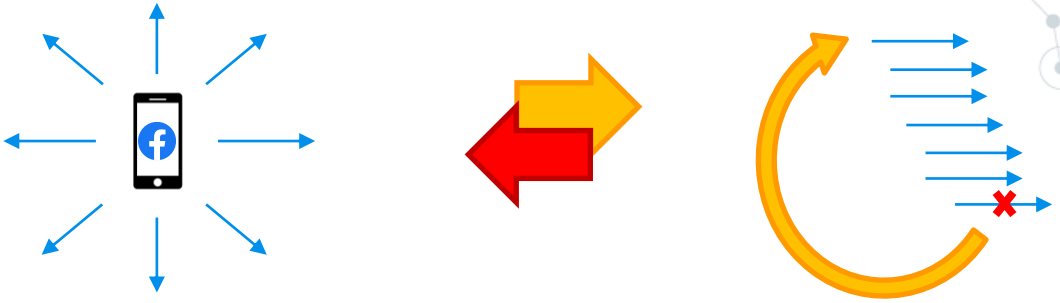


Photo by Thought Catalog on Unsplash

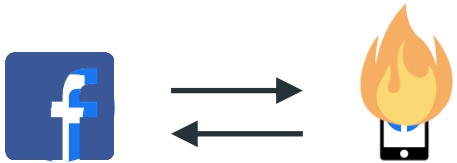
Emergono subito grossi problemi, in particolare per il newsfeed

# Il newsfeed di Facebook con le API REST

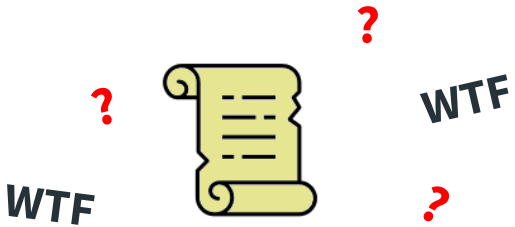
- Lento



- Fragile



- Difficile da sviluppare



# Principali vantaggi di GraphQL

- Permette ai client di scegliere **solo i dati necessari**
- Ha un unico endpoint che aggrega i dati da **sorgenti multiple**
- Utilizza un "**sistema tipizzato**" per descrivere i dati e le operazioni

# Cos'è GraphQL

## GraphQL è un linguaggio per creare API

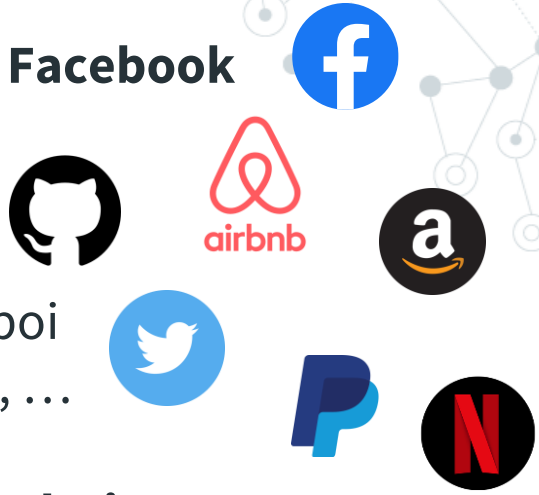
Il prefisso deriva dal fatto che i dati sono strutturati come in un **grafo**, in cui i nodi rappresentano gli oggetti e gli archi le relazioni tra di essi

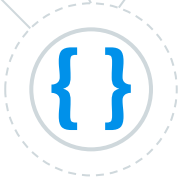
**Query Language** sta a indicare che si tratta di un linguaggio per interrogare e manipolare dati

Oltre al linguaggio per interagire con le API, GraphQL crea uno **strato applicativo** lato server per l'esecuzione delle operazioni

# Evoluzione di GraphQL

- **2012**: viene sviluppato come progetto interno da **Facebook**
- **2015**: diventa **open source**
- **2016**: è adottato da **GitHub**, **Airbnb** e **Twitter** e poi negli anni successivi da **Netflix**, **Paypal**, **Amazon**, …
- **2018**: passa sotto il controllo della **GraphQL Foundation**
- **2018**: ultima release stabile delle **specifiche**





*Playground: query*

# Il modello di GraphQL

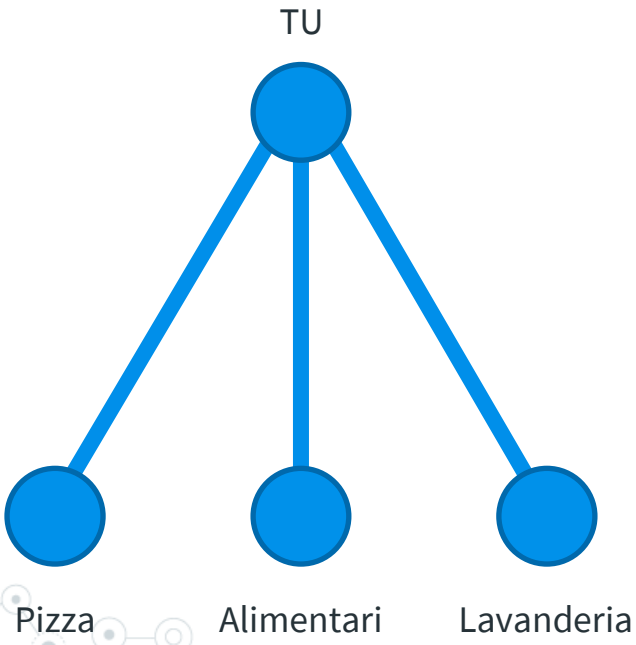
- Il layer GraphQL si inserisce tra il client e una o più sorgenti di dati

*Invece di tanti endpoint stupidi*

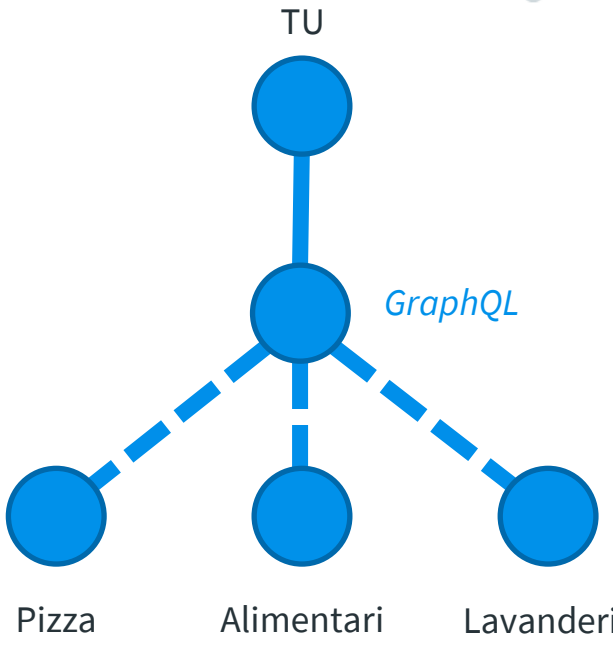
*Un solo endpoint intelligente*

# Il modello di GraphQL

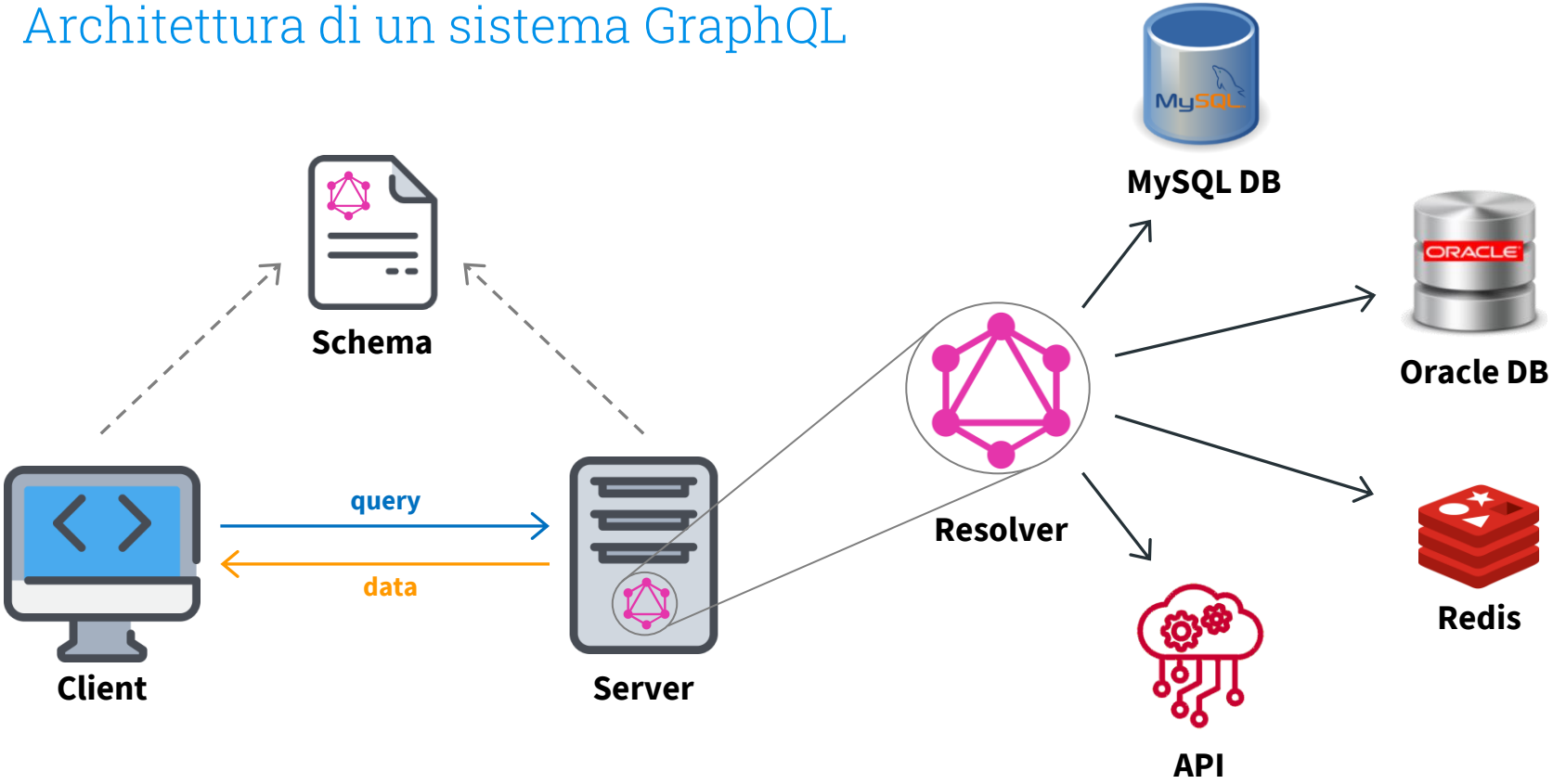
## REST



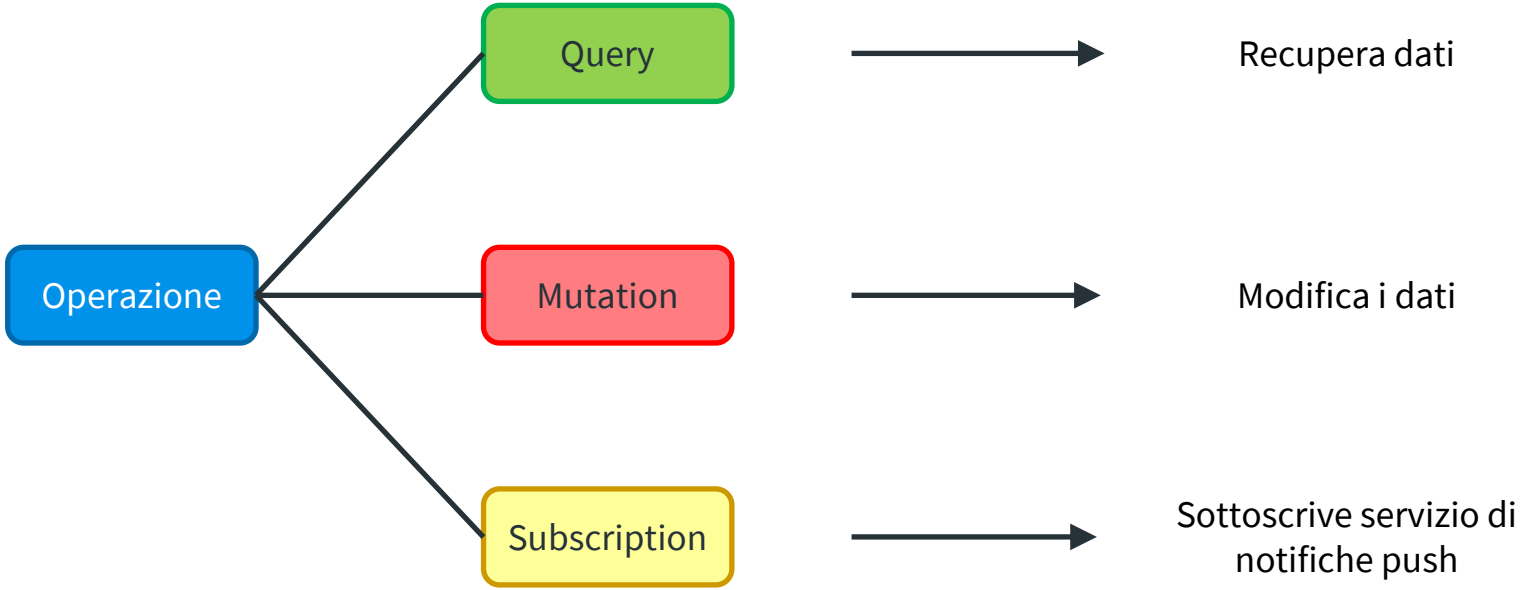

## GRAPHQL



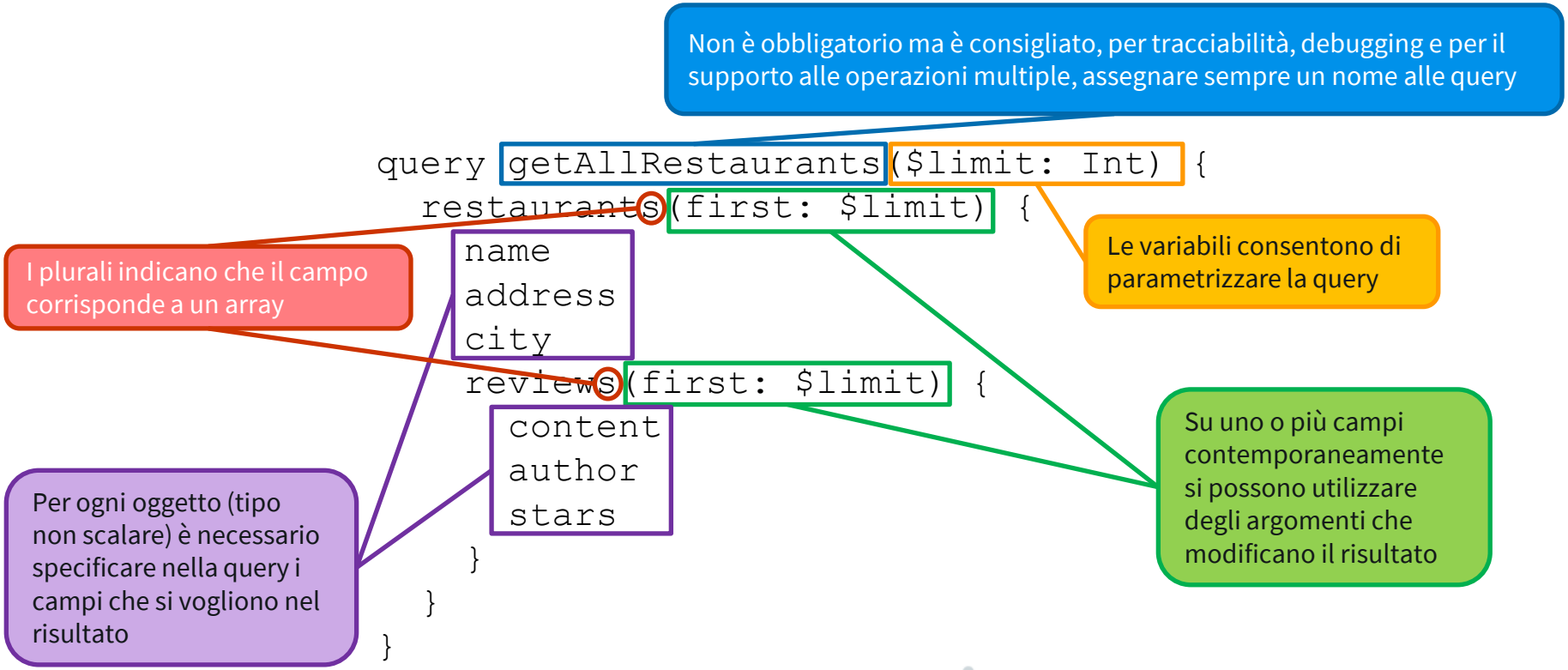
# Architettura di un sistema GraphQL

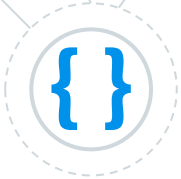


# Operazioni in GraphQL



# Riepilogo


Non è obbligatorio ma è consigliato, per tracciabilità, debugging e per il supporto alle operazioni multiple, assegnare sempre un nome alle query
query getAllRestaurants($limit: Int) {
restaurants(first: $limit) {
name
address
city
reviews(first: $limit) {
content
author
stars
}
}
}
I plurali indicano che il campo corrisponde a un array
Le variabili consentono di parametrizzare la query
Per ogni oggetto (tipo non scalare) è necessario specificare nella query i campi che si vogliono nel risultato
Su uno o più campi contemporaneamente si possono utilizzare degli argomenti che modificano il risultato



*Playground: mutation*

# Lo schema

```
type User {
    username: String!
    firstName: String!
    lastName: String!
}
```

Per ogni campo è specificato il tipo
Esistono 5 tipi scalari: Int, Float, String, Boolean e ID

```
type Review {
    id: ID!
    message: String
    rating: Int!
    restaurant: Restaurant!
    replies: [Reply]
}
```

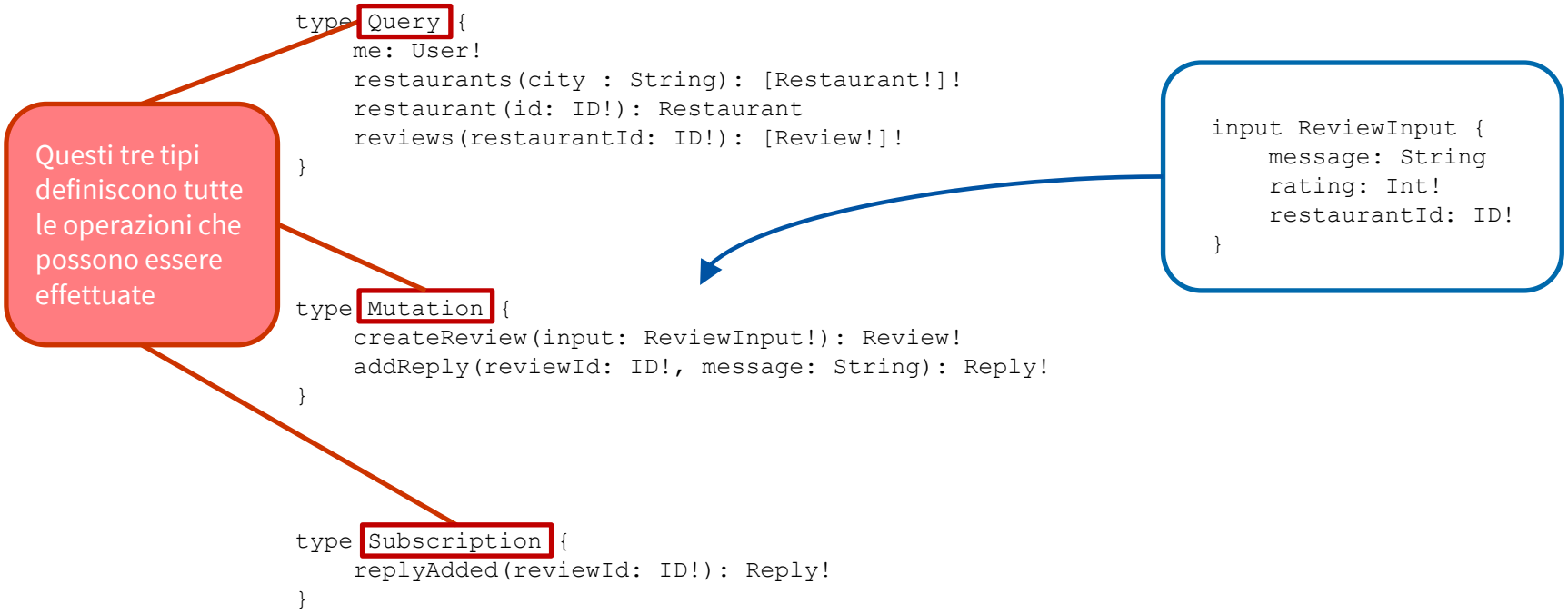
I campi di tipo oggetto fanno riferimento ad altri tipi definiti nello schema

```
type Restaurant {
    id: ID!
    name: String!
    address: String!
    city: String!
    rating: Float
    reviews(rating: Int): [Review]
    numberOfReviews: Int
}
```

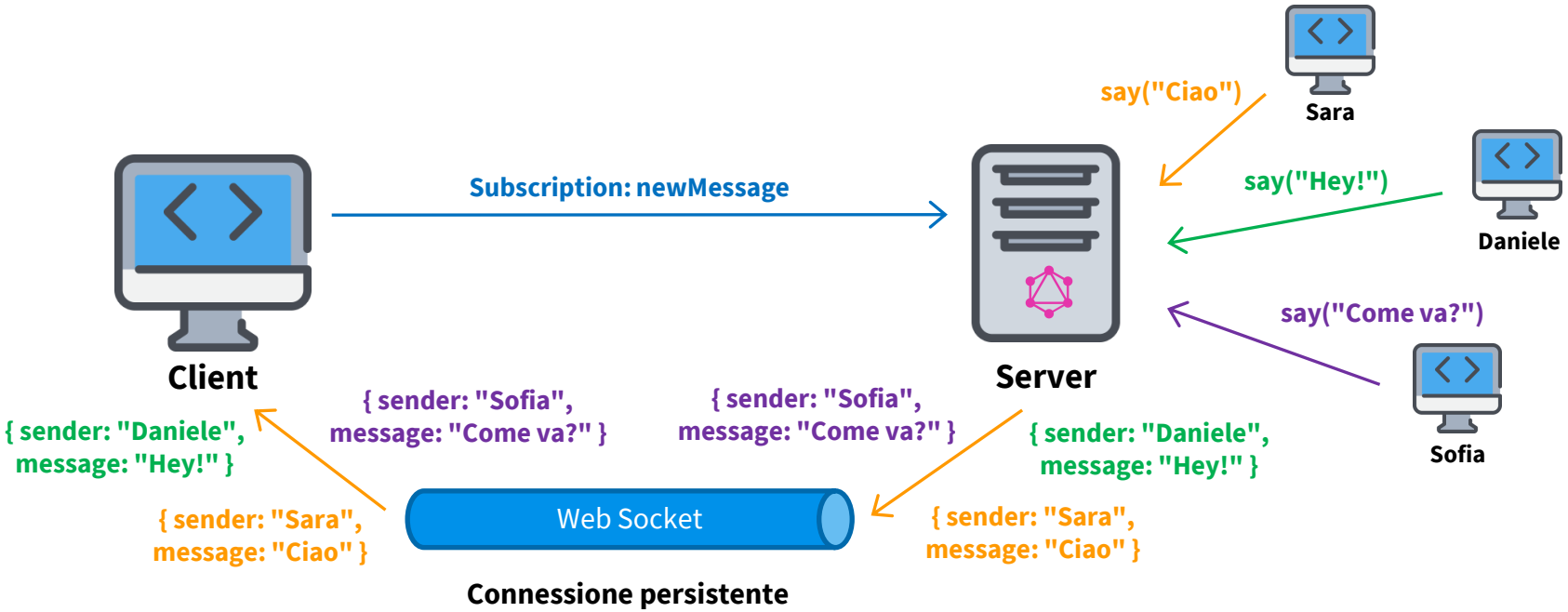
I campi possono prevedere l'uso di argomenti e anche restituire array

```
type Reply {
    id: ID!
    message: String
    review: Review
}
```

# Lo schema

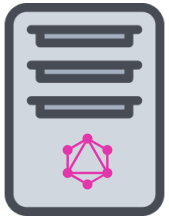

type Query {
me: User!
restaurants(city : String): [Restaurant!]!
restaurant(id: ID!): Restaurant
reviews(restaurantId: ID!): [Review!]!
}
Questi tre tipi definiscono tutte le operazioni che possono essere effettuate
type Mutation {
createReview(input: ReviewInput!): Review!
addReply(reviewId: ID!, message: String): Reply!
}
type Subscription {
replyAdded(reviewId: ID!): Reply!
}
input ReviewInput {
message: String
rating: Int!
restaurantId: ID!
}

# Subscription



# Come funziona un server GraphQL

```
query {
  restaurant(id: "1") {
    name
    city
    reviews {
      content
      stars
    }
  }
}
```



**GraphQL Server**

## 1. Parse

Viene controllata la sintassi della query e viene generato l'**AST** (Abstract Syntax Tree)

## 2. Validate

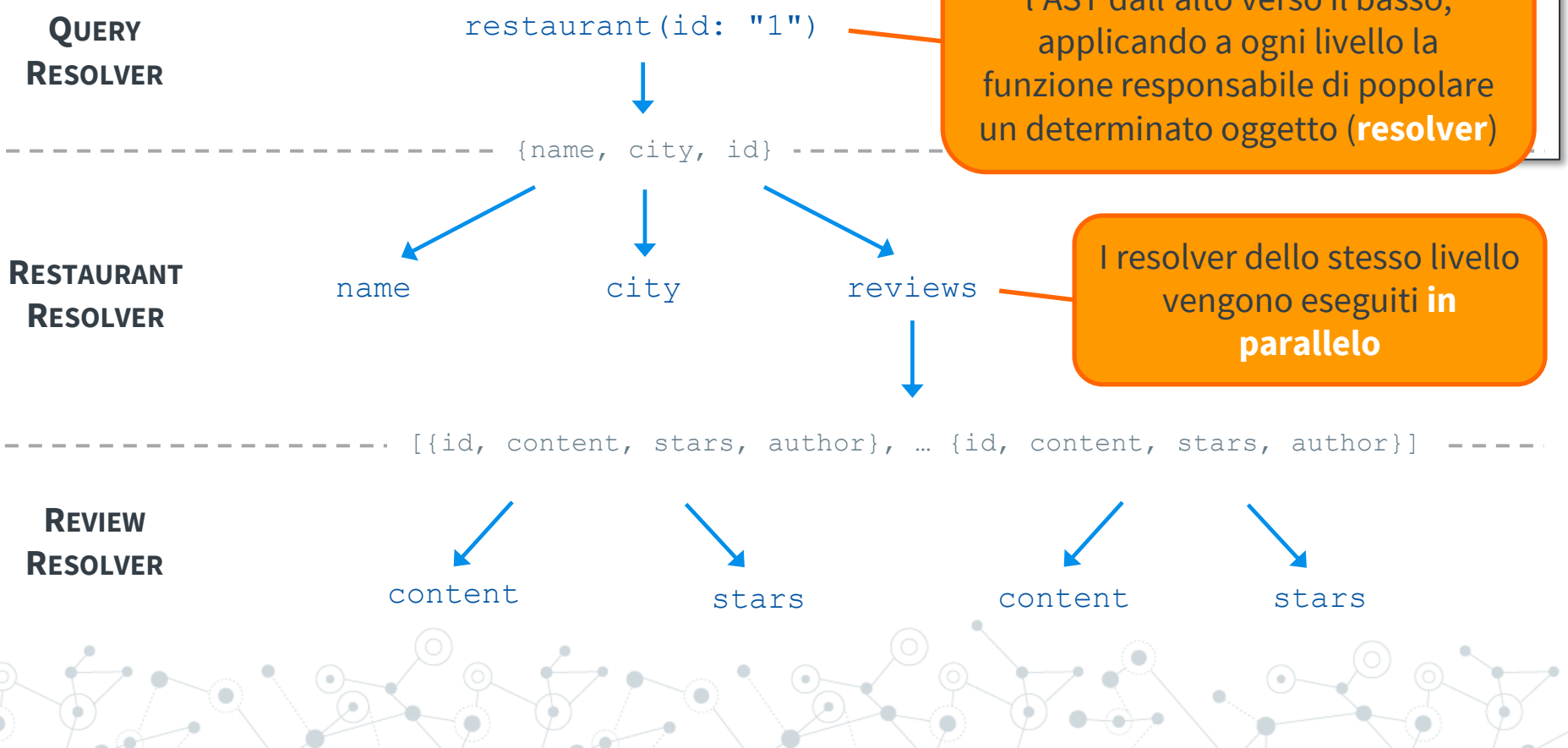
La query viene validata rispetto allo **schema**

## 3. Execute

La query viene **eseguita** e viene generato il risultato

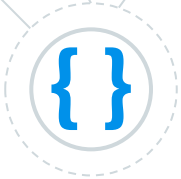
# Esecuzione di una query

```
query {
                                    {
```




Il processo di esecuzione scorre l'AST dall'alto verso il basso, applicando a ogni livello la funzione responsabile di popolare un determinato oggetto (**resolver**)

I resolver dello stesso livello vengono eseguiti **in parallelo**



*Live coding*

https://tinyurl.com/graphql-resources

# Q&A

# Domande?

## *Tiziano Pessa*

CTRL+C/CTRL+V'ing @ ( )REGISTER

*pixel13*

*tizianopessa*

*tizianopessa*

*@tizianopessa*

## *Lorenzo Spinelli*

I write code @ ( )REGISTER

*spyna*

*SpinaLorenzo*

*lorenzospinelli*

*@spyna*

REGISTER

***We're hiring!*** 

Sviluppatori PHP/Java/Javascript
Fully remote

https://tinyurl.com/register-is-hiring