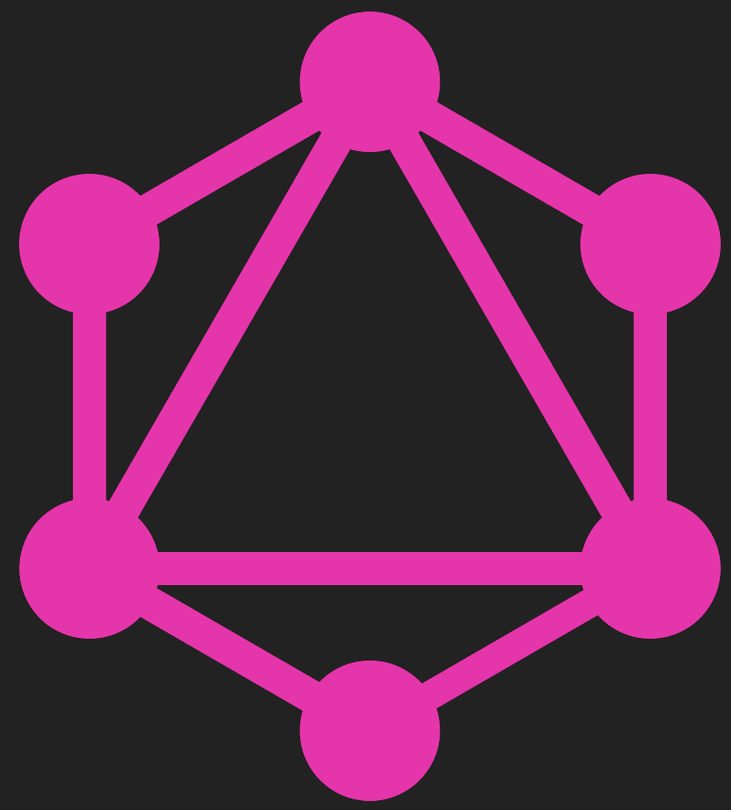


It's Graphs All the way Down !

With GraphQL, You model your business domain as graph.

INTRODUCTION TO GraphQL

Chester Cheng

@Mediatek IT/EC2

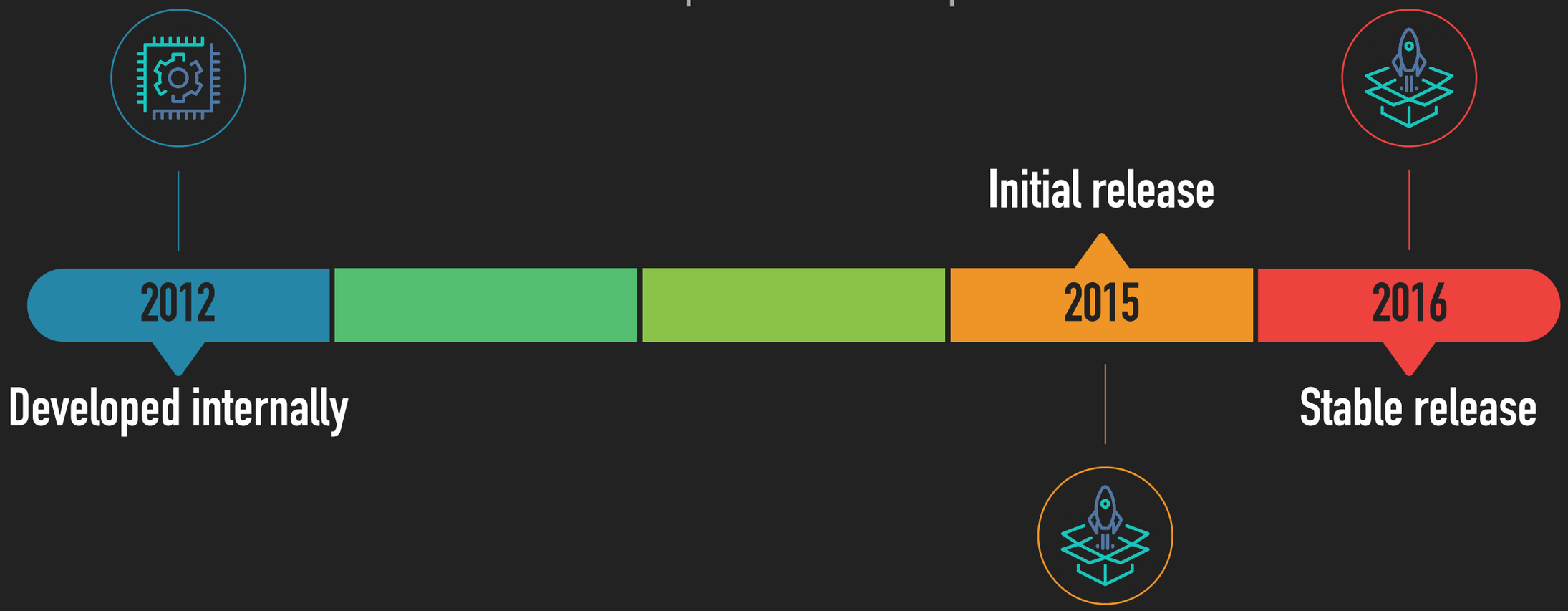
github.com/chestercheng

www.linkedin.com/in/chestercheng626

What is GraphQL

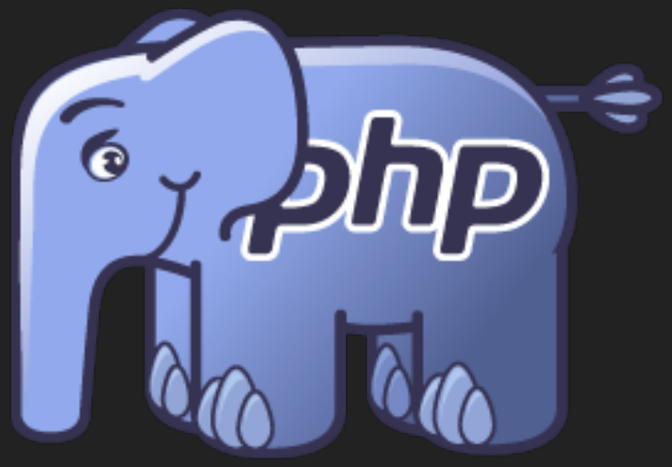
A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

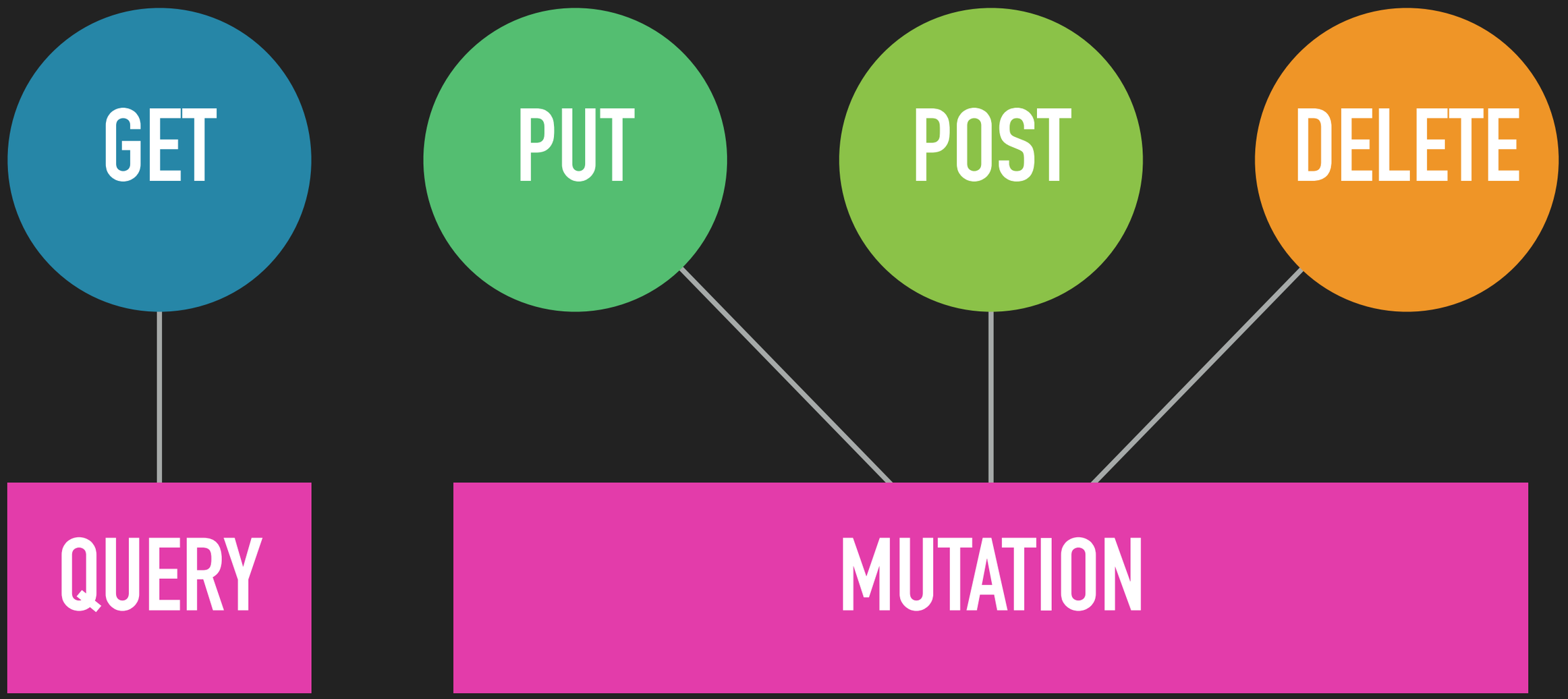


Who's using

Programming languages



Operations



HTTP Methods

POST

application/graphql

```
query {
    me {
        name
    }
}
```

POST

application/json

```
{
  "query":
      "{me{name}}"
}
```

GET

HTTP query string

```
?query={me{name}}
```

HTTP is the most common choice for client-server protocol when using GraphQL because of its ubiquity. Here are some guidelines for setting up a GraphQL server to operate over HTTP.

Newsfeed Resources

Post

Comment

Author

REST v.s. GraphQL

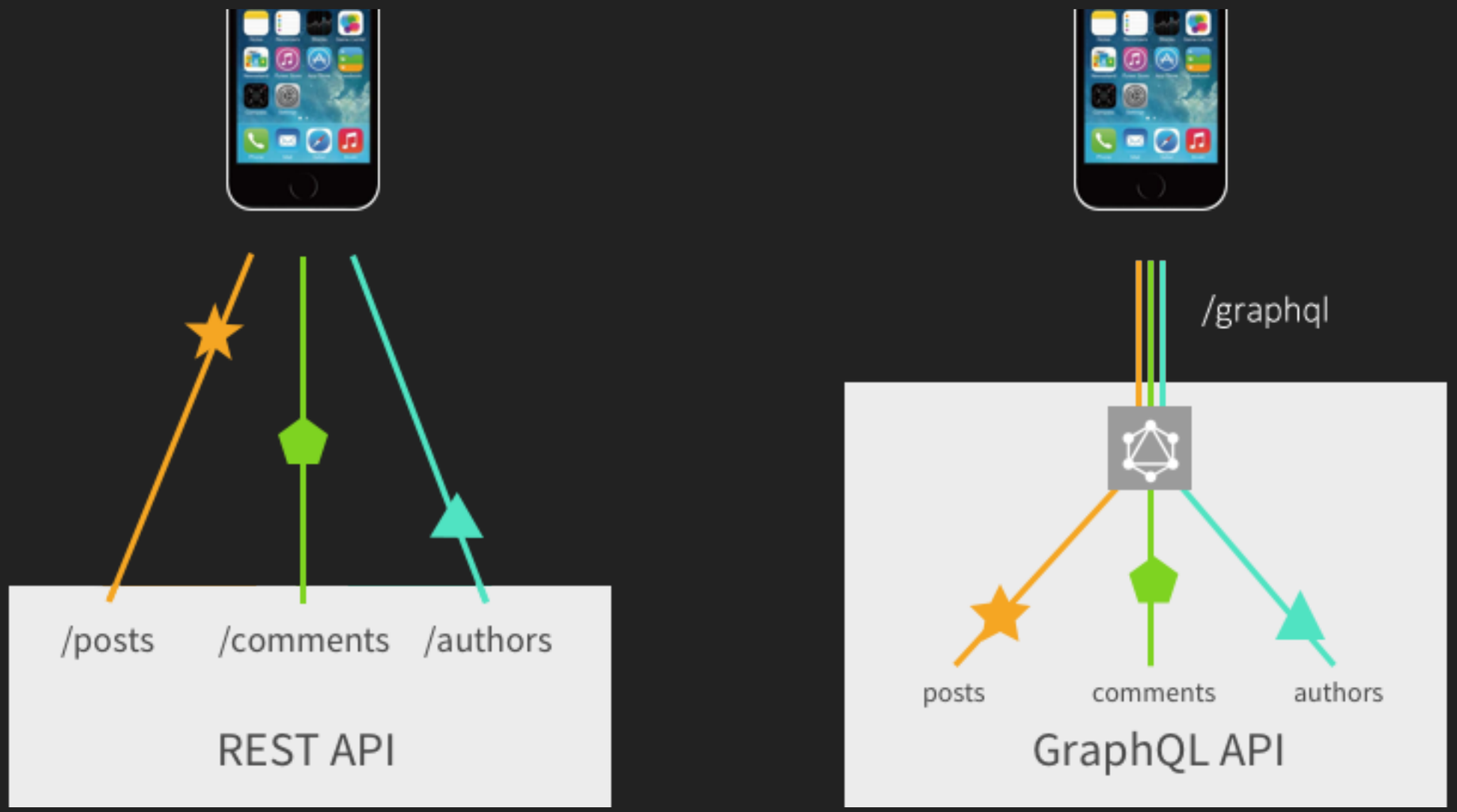
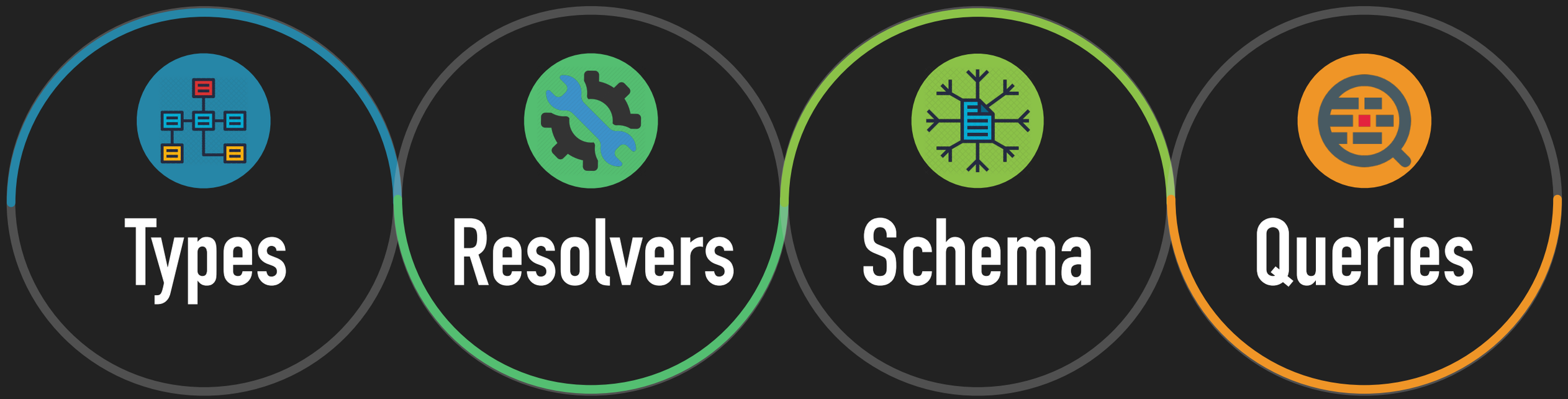


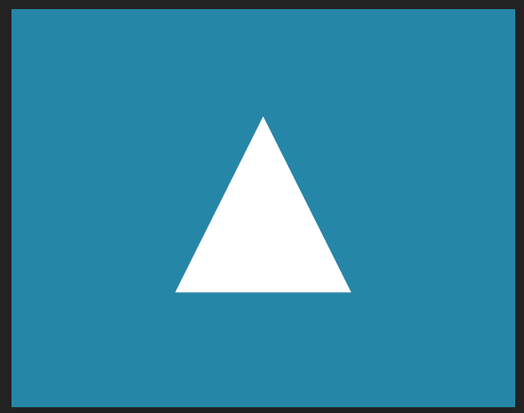
Image credit: https://dev-blog.apollodata.com/graphql-vs-rest-5d425123e34b

How GraphQL Works?



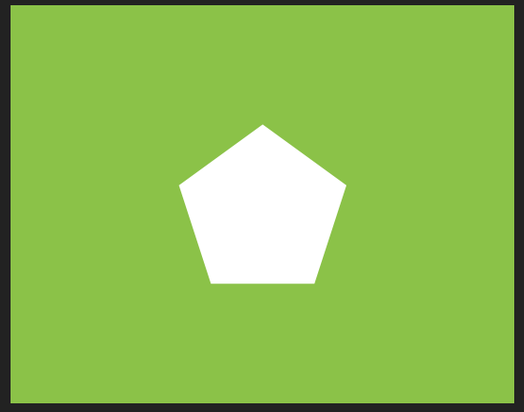
A GraphQL schema describes your data model, and provides a GraphQL server with an associated set of resolve methods that know how to fetch data.

Types



```python
class Author(ObjectType):

    id = ID()
    name = String(required=True)
```



```python
class Comment(ObjectType):

    id = ID()
    author = Field(Author, required=True)
    text = Field(String, required=True)
```

```python
class Post(ObjectType):

    id = ID()
    author = Field(Author, required=True)
    text = String(required=True)
    comments = List(Comment)
```

Graphene defines the following base Scalar Types: `String`, `Int`, `Float`, `Boolean`, `ID`

Resolvers

```python
class MyRootQuery(ObjectType):

    newsfeed = List(Post)
    post = Field(Post, id=Int())

    def resolve_newsfeed(self, info):
        return get_newsfeed()

    def resolve_post(self, info, id):
        return get_post(id)
```

A resolver is a method that resolves certain fields within a ObjectType. If not specifiifed otherwise, the resolver of a field is the `resolve_{field_name}` method on the ObjectType.

By default resolvers take the arguments `info` and `*args`.

Schema

```
my_schema = Schema(
    query=MyRootQuery,
    mutation=MyRootMutation,
)
```

A Schema is created by supplying the root types of each type of operation, `query` and `mutation` (optional). A schema definition is then supplied to the validator and executor.

Queries

GraphQL Request

```
query {
    post(id: 1) {
        text
        author {
            name
        }
        comments {
            text
            author {
                name
            }
        }
    }
}
```

JSON Result

```
{
  "data": {
    "posts": {
        "author": {"name": "Chester"},
        "comments": [
            {
                "author": {"name": "ming"},
                "text": "Good!"
            },
            {
                "author": {"name": "DAiNESE"},
                "text": "Awesome!"
            }
        ],
        "text": "It's Graphs All the way Down!"
    }
}
```

Python Modules

- `graphene` - GraphQL framework for Python
- `graphene-django` - Graphene Django integration
- `flask-graphql` - Adds GraphQL support to your Flask application
- `sanic-graphql` - Adds GraphQL support to your Sanic app
- `graphene-sqlalchemy` - Graphene SQLAlchemy integration
- `aiodataloader` - Asyncio DataLoader for Python3