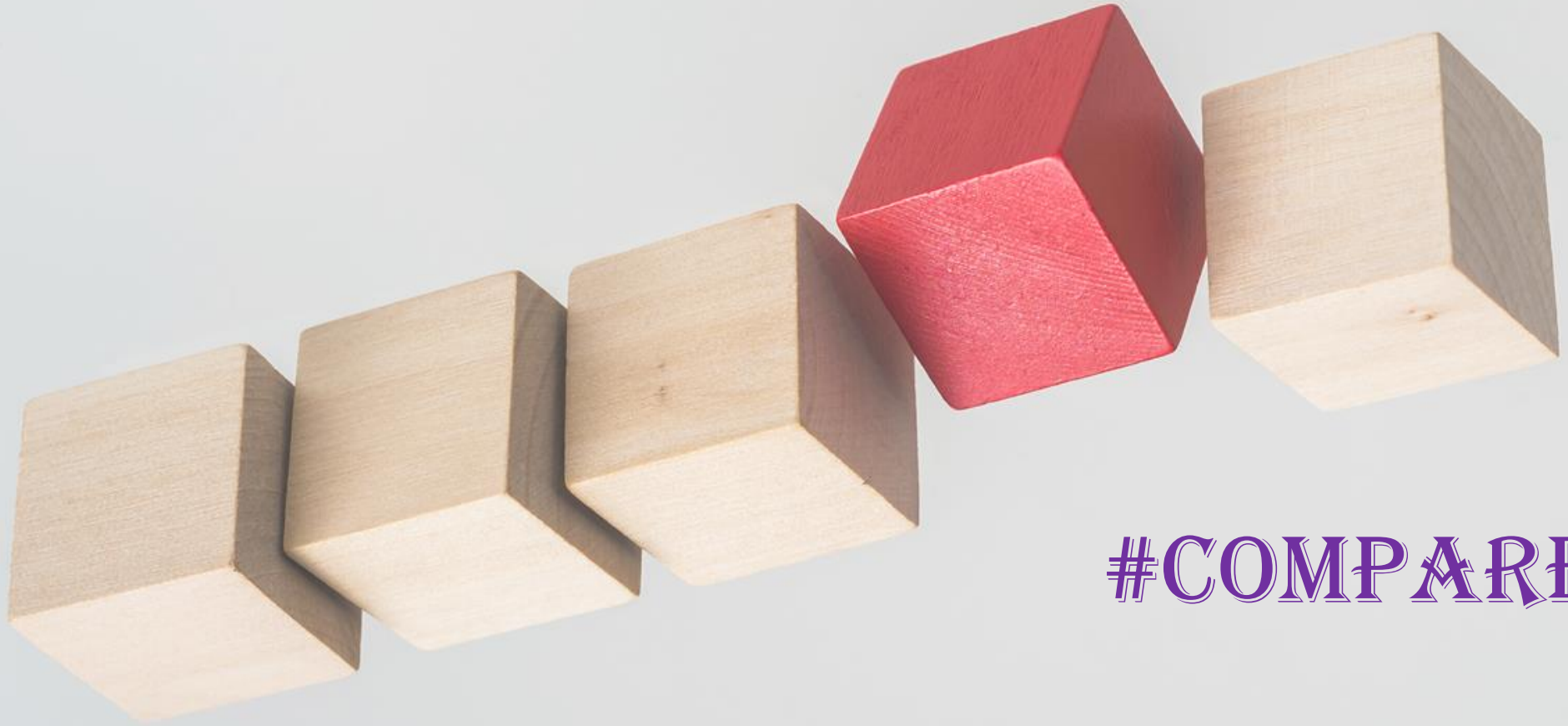
REST, GRAPHQL & GRPC – A COMPARISON

- Poornima Nayar
- Freelance .NET Developer
- Langley, Berkshire
- Mother, Reading, Carnatic Music
- @poornimanayar

#NOTADEBATE



#COMPARE

gRPC

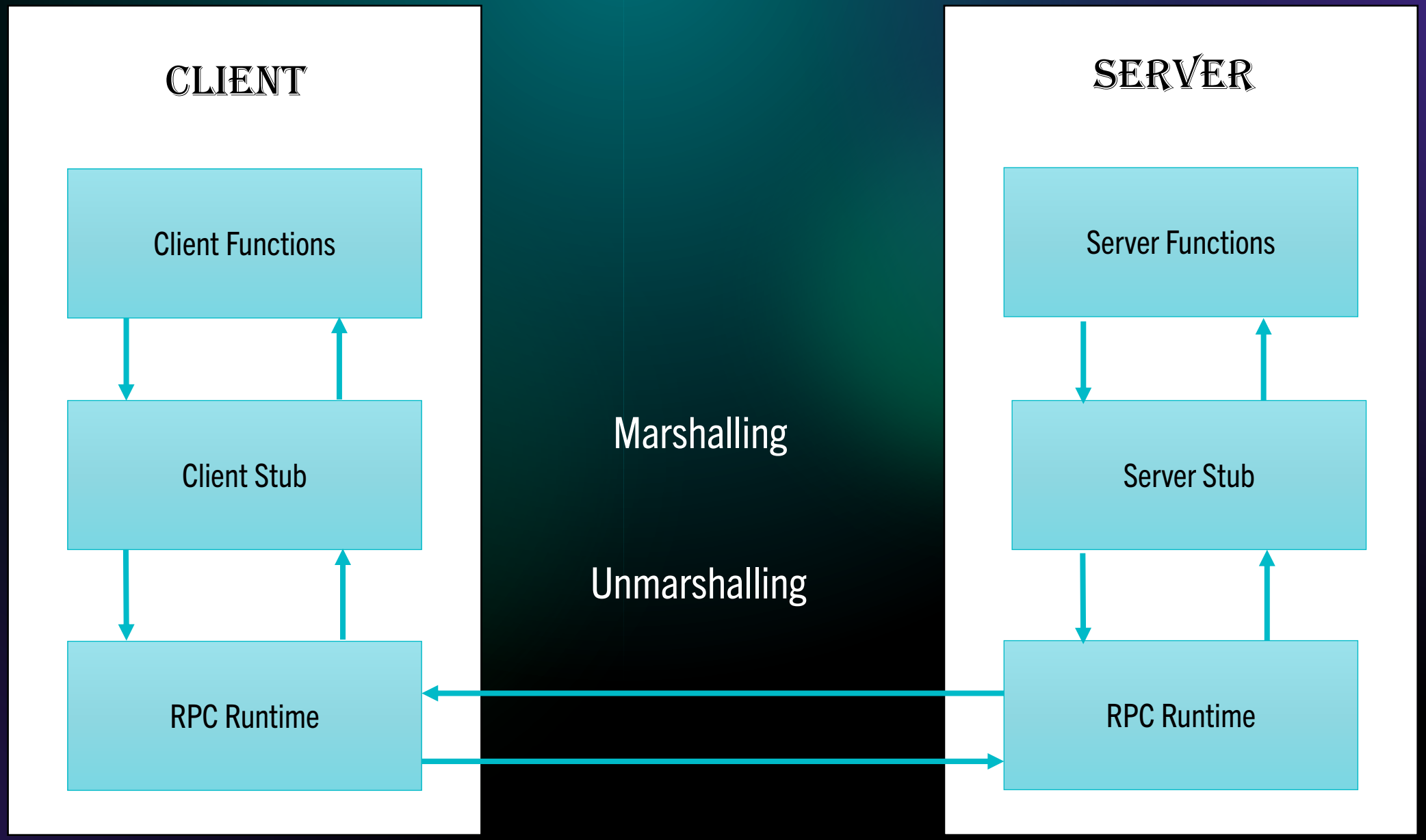
gRPC is a modern open-source, high performance Remote Procedure Call (RPC) framework

REMOTE PROCEDURE CALL

“A remote procedure call is when a computer program causes a procedure to execute in another address space (commonly another computer) , which is coded as if it were a normal procedure call, without the programmer explicitly coding the details for the remote interaction.”

Functions

Call a function on another server as though it was a local function


CLIENT
SERVER
Client Functions
Client Stub
RPC Runtime
Marshalling
Unmarshalling
Server Functions
Server Stub
RPC Runtime

gRPC

- Google’s implementation of RPC
- Initially called Project Stubby
- Made open-source and called gRPC in March 2015 (next version)
- Officially supported in .NET Core 3.0+
- Contract-based API development
- Designed for HTTP/2

PROTOCOL BUFFERS

- Google's open-source mechanism to serialize structured data
- Language-neutral, platform-neutral, extensible
- Data is structured as messages
- Each message is a record of key-value pairs called fields
- Messages are transmitted in binary format

```
message PersonReply{
    string first_name = 1;
    string last_name = 2;
    bool has_a_pet_unicorn = 3;
}
```

.PROTO FILES

- The contract of the API
- Definition of the gRPC service
- Procedures
- Messages sent between the client and the server

```
syntax = "proto3";

option csharp_namespace = "gRPC.Demo";

package greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

```
option csharp_namespace = "gRPC.Demo";

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}
```

Greeter.GreeterBase

```
// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}
```

HelloRequest

```
// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

HelloReply

```
option csharp_namespace = "gRPC.Demo";

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}
```

Greeter.GreeterClient

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
</ItemGroup>
```

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
</ItemGroup>
```

- Both (default)
- None

- GetUsers()
- GetUserById()
- CreateUser()
- UpdateUser()
- DeleteUser()

- GetStoriesForUser()
- GetStoryById()
- GetTopUserStories()
- GetStoryComments()
- GetLatestActivity()

gRPC METHODS

- Unary
- Server Streaming
- Client Streaming
- Bidirectional Streaming

METADATA

- Request headers
- Response headers
- Response trailers
- Response trailers are served after the response is complete

- Lightweight messages
- High performance
- Built in code generation
- Streaming options
- Reduced network usage with protocol buffers
- Scalable

- Tightly coupled client and server
- Limited browser support
- Relies on HTTP/2
- Zero discoverability
- Function explosion
- Not human readable (binary format)

- Acronym for REpresentational State Transfer
- First presented by Roy Fielding in 2000
- Architectural style for distributed hypermedia systems

Resource

Here are some resources and here is what you can do with them!

Resource identifier

Data + Metadata + Hypermedia links = Resource Representation

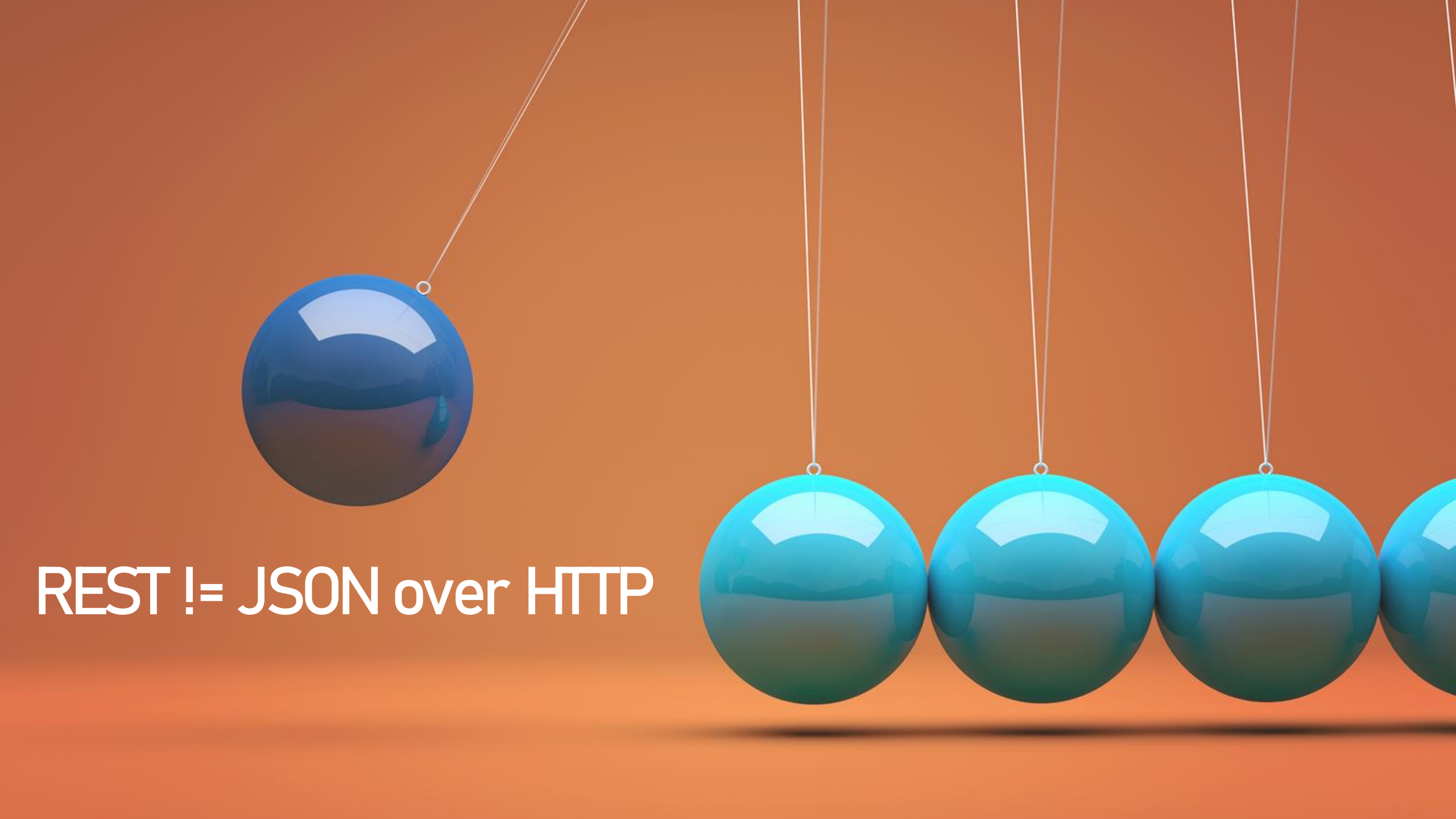

```
{
    "id": 1,
    "name": "Harry Potter",
    "headline": "The boy who lived",
    "links": [
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "self",
            "method": "GET"
        },
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "update-user",
            "method": "PUT"
        },
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "delete-user",
            "method": "DELETE"
        }
    ]
}
```



```
content-length: 381
content-type: application/json; charset=utf-8
date: Sat25 Sep 2021 11:15:19 GMT
server: Microsoft-IIS/10.0
x-powered-by: ASP.NET
```

REST CONSTRAINTS

- Client-server
- Stateless
- Cacheable
- Uniform Interface
- Layered Systems
- Code on demand(optional)


REST != JSON over HTTP


REST != 'RESTFUL' PROCEDURE CALLS


REST != HTTP

Reuse HTTP

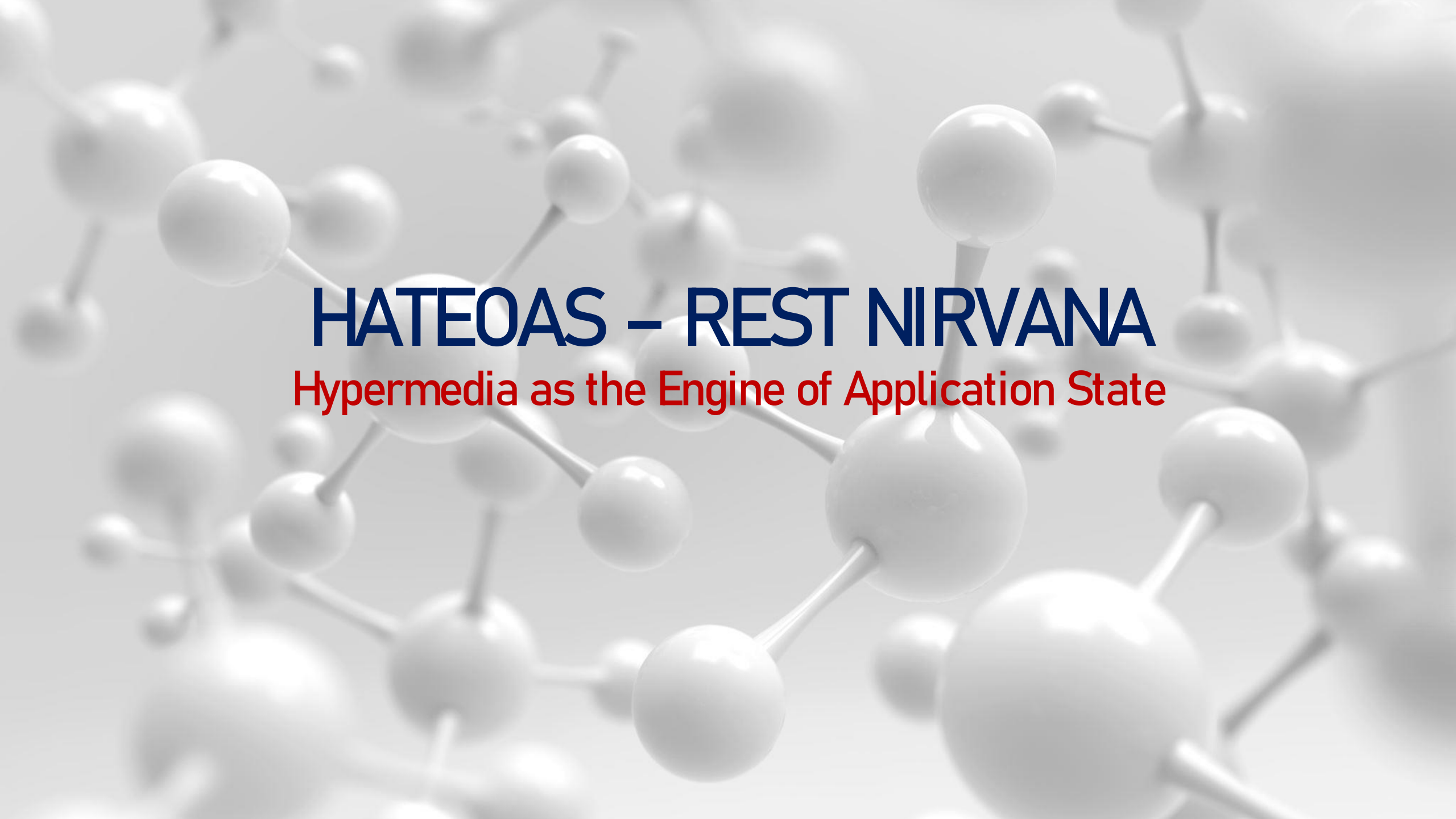
ROOT Entry Point

https://api.xyz.com

```
{
  "users": {
    "href": "https://localhost:44310/api/Users",
    "rel": "[collection]",
    "method": "GET"
  }
}
```

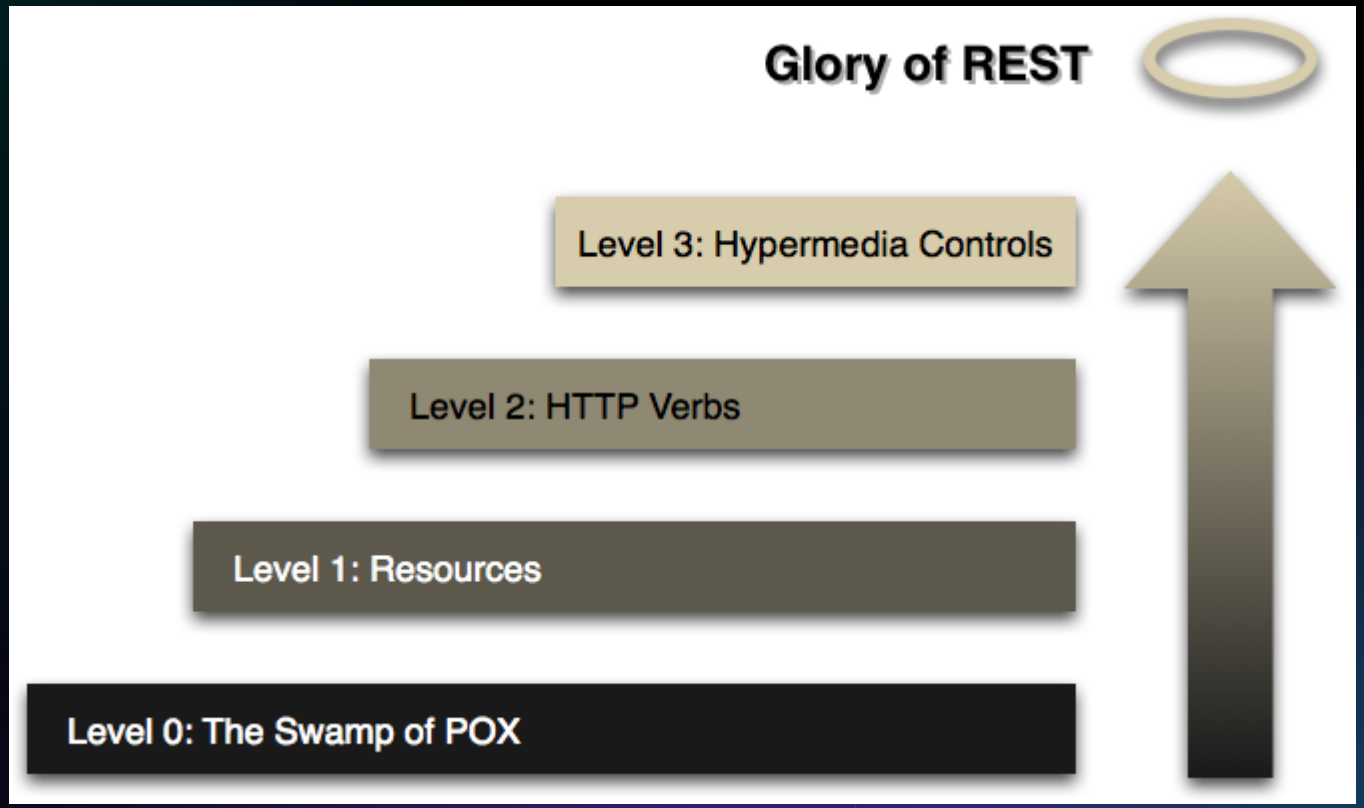
HTTP Verb	URI	Request Body	Response Body	Status Code
GET	/users /users/{id}	✗	✓	200 Ok/ 404 Not Found
POST	/users	✓	✓	201 Created with the URI of resource in the Location Header
PUT	/users/{id}	✓	✗	200 Ok / 204 No Content/ 202 Accepted/ 409 Conflict
DELETE	/users/{id}	✗	✗	204 No Content / 404 Not Found

- GetStoriesForUser()
- GetStoryComments()



HATEOAS – REST NIRVANA

Hypermedia as the Engine of Application State



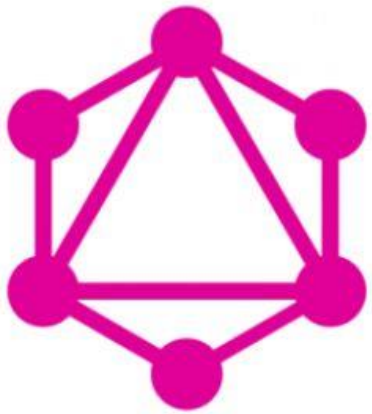
Picture Courtesy : Richardson Maturity Model (martinfowler.com)

```
{
    "id": 1,
    "name": "Harry Potter",
    "headline": "The boy who lived",
    "links": [
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "self",
            "method": "GET"
        },
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "update-user",
            "method": "PUT"
        },
        {
            "href": "https://localhost:44310/api/Users/1",
            "rel": "delete-user",
            "method": "DELETE"
        },
        {
            "href": "https://localhost:44310/api/Users/1/Stories",
            "rel": "user-stories",
            "method": "GET"
        }
    ]
}
```

- HAL (Hypertext Application Language) , WIP standard convention
- application/hal+xml , application/hal+json

- Decoupled client and server
- API can evolve over time
- Scalable
- Seamless integration and reuse HTTP
- Easy to consume
- Superior browser support

- Chatty
- No single spec
- Big payloads
- Overfetching
- Underfetching
- N+1 problem



GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.

Specify types and ask for specific fields on those types

Single, smart endpoint that takes in complex queries and builds the data output

The clients specify the shape of the data that they need, and the server responds back with the exact same data graph as the query.

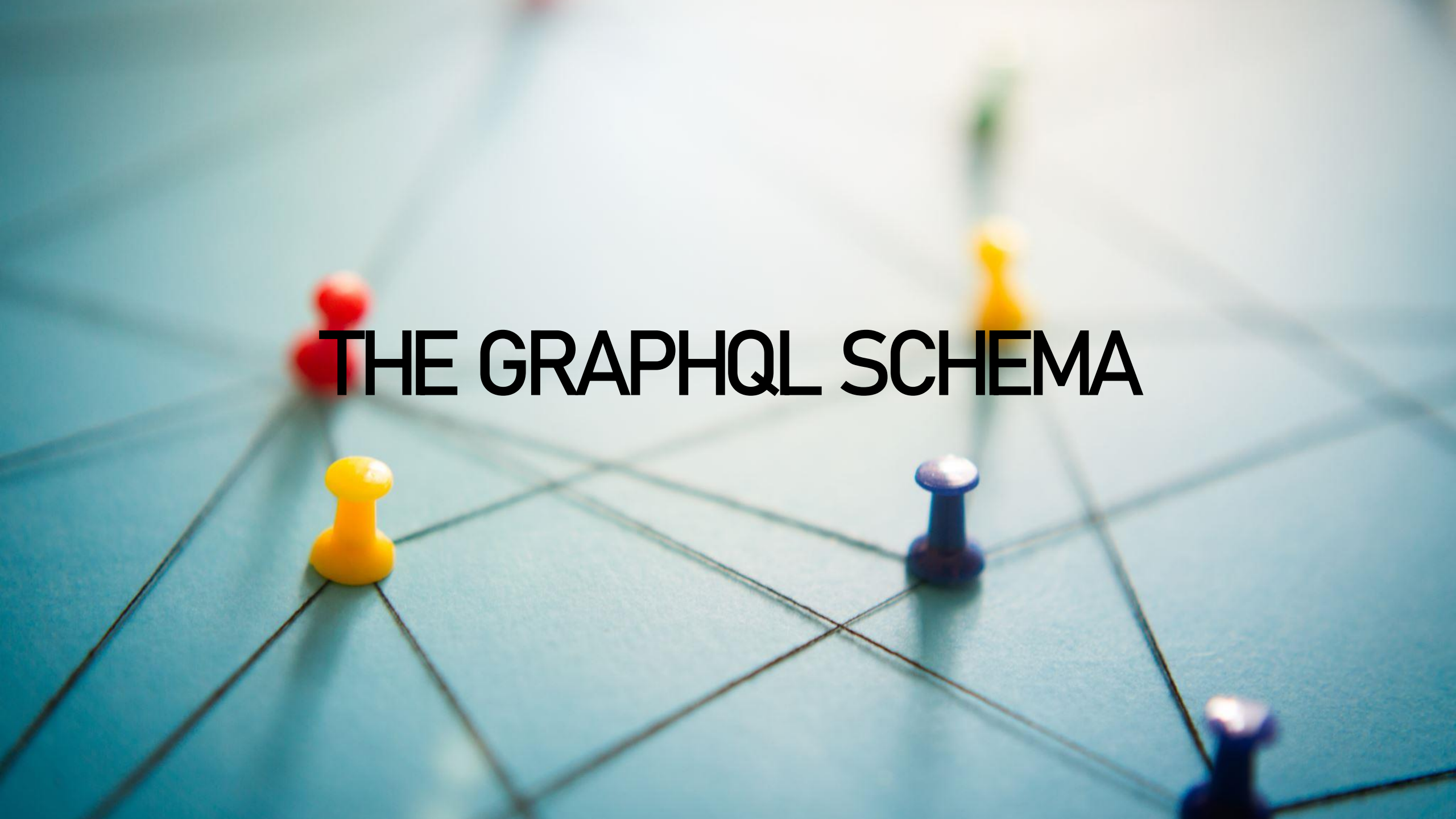
```
query{
  course(id:1045){
    id,
    credits,
    title,
    enrollments{
      student{
        firstName,
        lastName,
        emailAddress
      }
    }
  }
}
```


```
{
  "data": {
    "course": {
      "id": 1045,
      "credits": 4,
      "title": "Calculus",
      "enrollments": [
        {
          "student": {
            "firstName": "Meredith",
            "lastName": "Alonso",
            "emailAddress": "Meredith@graphqlschool.com"
          }
        },
        {
          "student": {
            "firstName": "Peggy",
            "lastName": "Justice",
            "emailAddress": "Peggy@graphqlschool.com"
          }
        }
      ]
    }
  },
  "extensions": { ↔ }
}
```

Query

Ask for data and get the data in the same shape

- Strongly Typed
- Specification
- Introspective
- Hierarchical
- JSON response that mirrors the query
- Application Layer
- Version Free
- Transport-layer agnostic


THE GRAPHQL SCHEMA

SCHEMA

TYPES

Object Types	Queries	Mutations	Subscriptions
Representation of object	*Get data*	*Insert/Update/ Delete data*	*Pushed Updates*

FIELDS

Name	Scalar/Complex Types	Resolvers
Name of the field	Type of the field	Resolve the field to concrete data

QUERY

- One of the operation types in GraphQL
- Top-level entry point for read operations
- Get/fetch the data, ask for specific fields on objects using a POST operation
- query keyword, although not needed, best practice
- Hierarchical JSON response that mirrors the query

MUTATIONS

- Top-level entry point for write operations
- HTTP POST
- mutation keyword
- Accepts a InputType as an argument

SUBSCRIPTIONS

- Updates are pushed from the server, not polled by client
- Clients can choose to listen
- Works over WebSockets
- HTTP POST
- subscription keyword
- Subscriptions are stateful

Type	Fields
Query	users, user
Mutation	createUser, updateUser, deleteUser
Subscription	userAdded, userUpdated, userDeleted

```
query allusers{
  users{
    id,
    name,headline
  }
}
```

```
query singleuser($id:Int!){
  user(id:$id){
    id,
    name,
    headline
  }
}
```

```
mutation addUser($addUserInput:UserInput!){
createUser(userInput:$addUserInput){
  id,name,headline
}
}
```

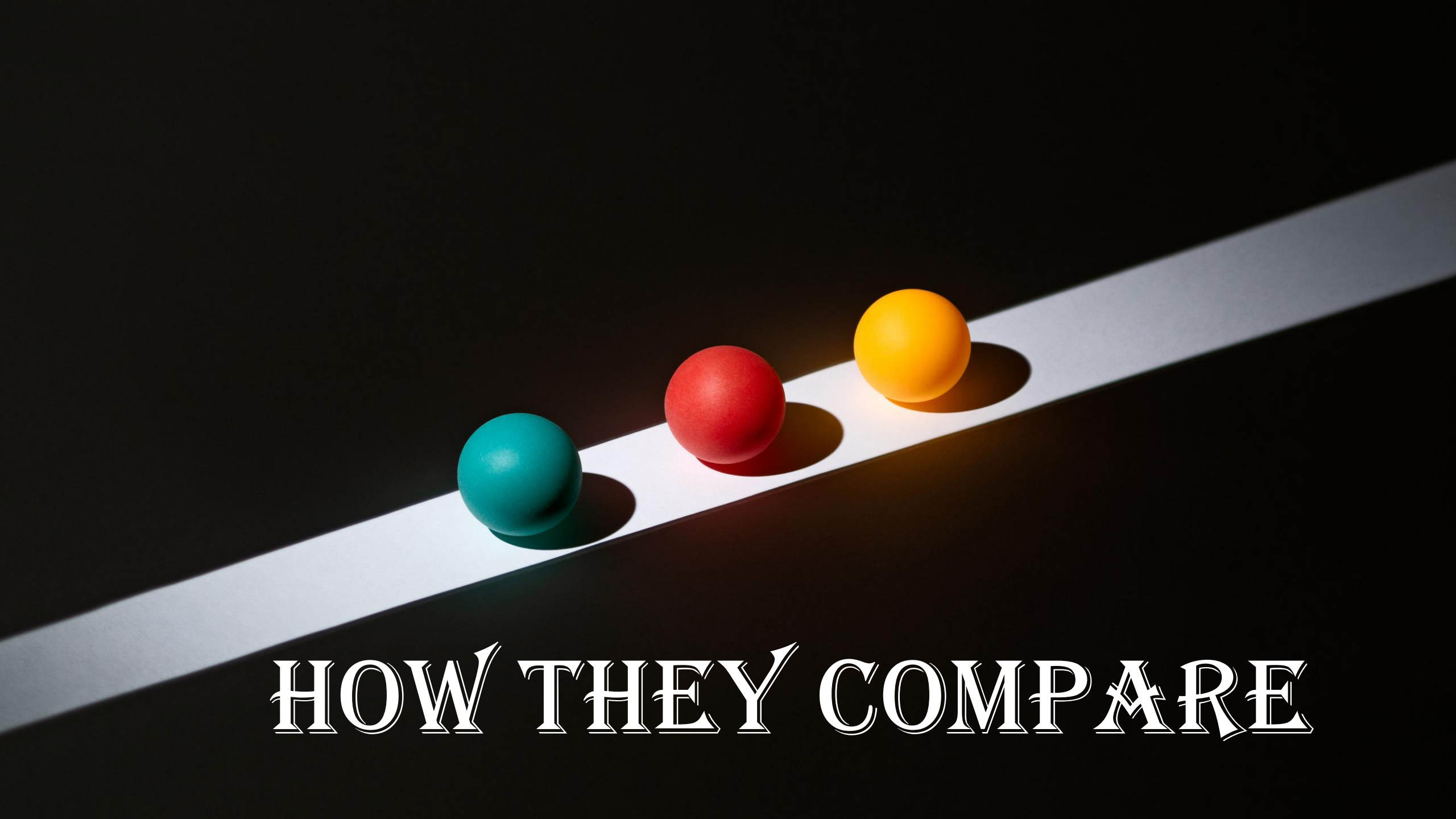
```
mutation updateUser($updateUserInput:UserUpdateInput!){
  updateUser(userInput:$updateUserInput){
    id,
    name,
    headline
  }
}
```

```
mutation deleteUser($id:Int!){
  deleteUser(id:$id){
  text
  }
}
```

```
subscription subscribeAddUser{
  userAdded{
    id,
    name,
    headline
  }
}
```

- Exact Fetching
- Loosely coupled client and server
- Autogenerating API documentation
- Version Free

- Caching complexity
- File uploading
- Performance with complex queries
- Learning curve
- Growing community


HOW THEY COMPARE

	REST	GRAPHQL	gRPC
Fundamental Unit	Resources	Query	Function
Coupling	Low	Low	High
Chattiness	High	Low	Medium
Caching	HTTP	Custom	Custom
Discoverability	Good	Good	Bad
Versioning	Can be needed	No versioning	Can be needed
Statefulness	Stateless	Stateless(Query, Mutation)	Stateless

	REST	GRAPHQL	gRPC
Performance	Good	Can suffer on complex queries	Good
Payload size	Can get heavy	Medium	Lightweight
Payload format	Human Readable (JSON)	Human Readable (JSON)	Binary
Browser support	Yes	Yes	No (requires gRPC-web)
Standard	Only best practices	Formal Specification	Formal Specification
Maturity	Mature	Still early	Still early
Learning Curve	Medium	High	Medium
Learning Resources	Many resources	Few resources	Few resources

	REST	GRAPHQL	gRPC
.NET Support	Out-of-the-box	Third-party implementations	Out-of-the-box
Visual Studio Support	Out-of-the-box templates	External Templates	Out-of-the-box templates
Hosting (Azure)	App Service, Azure Functions	App Service, Azure Functions	Azure Kubernetes Service, App Service


TOOLS

REST

- Swagger UI
- Postman
- Command Line
- REST Client (VS Code extension)

GRAPHQL

- GraphiQL
- GraphQL Playground
- Insomnia
- Banana Cake Pop

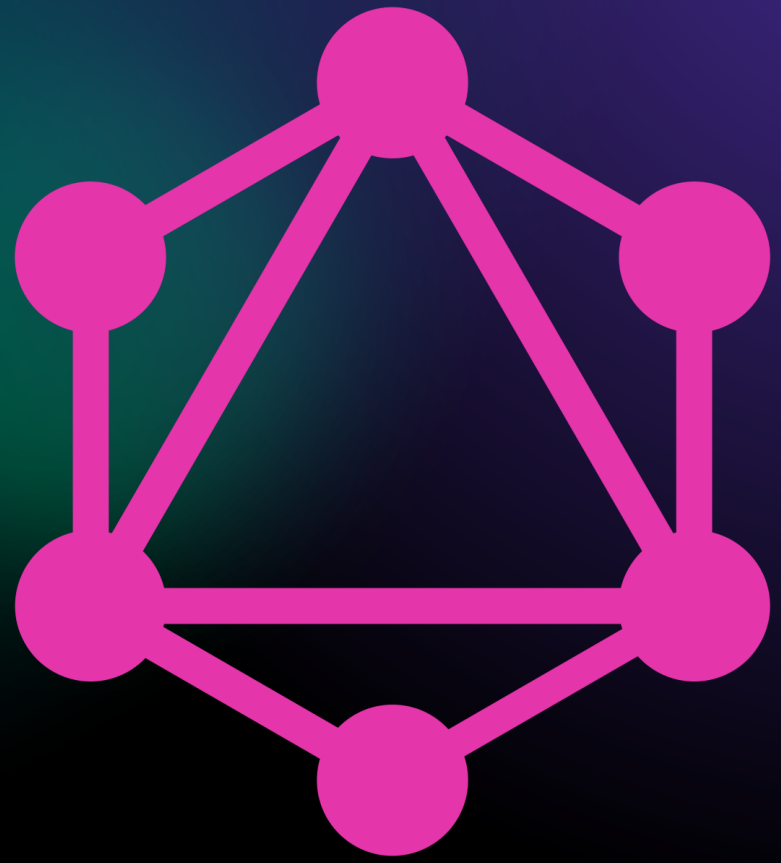
gRPC

- gRPCCurl
- grpcui
- BloomRPC

USE CASES

- Management APIs (CRUD)
- Focus on objects / resources
- Varied client usage
- Microservices

- Mobile API
- Where data is like a graph/relational
- Where data needs fetching from multiple APIs



- Action oriented APIs
- Internal services
- Microservices
- IOT
- Polyglot environments

https://bit.ly/ddd-api-compare


THANK YOU