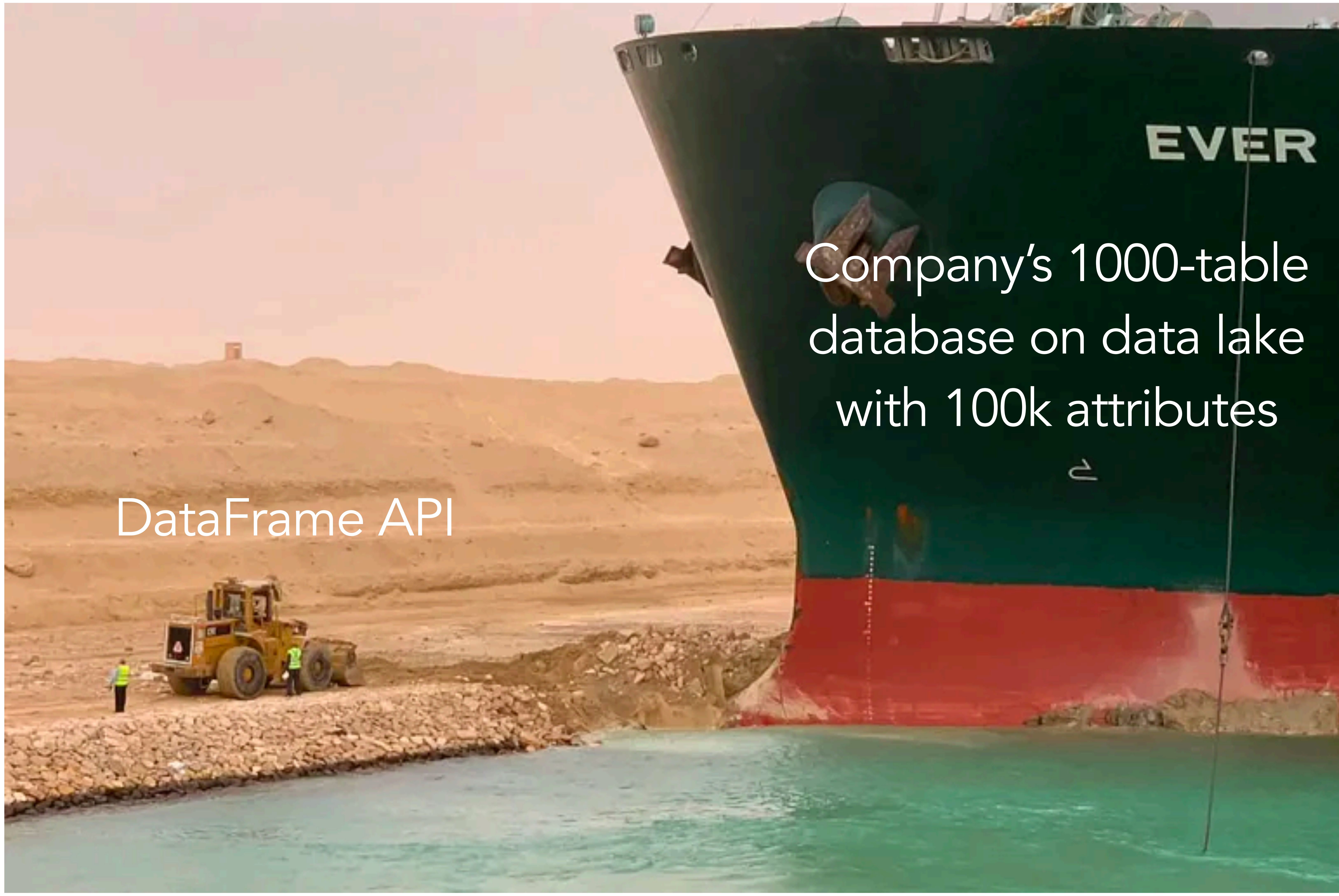
DSC 204a Scalable Data Systems

- Haojian Jin


Company's 1000-table database on data lake with 100k attributes
DataFrame API
EVER

Meme idea credit: https://datasystemsfun.tumblr.com/

Where are we in the class?

Foundations of Data Systems (2 weeks)

- Digital representation of Data → Computer Organization → Memory hierarchy → Process → Storage

Scaling Distributed Systems (3 weeks)

- Cloud → Network → Distributed storage → Parallelism → Partition and replication

Data Processing and Programming model (5 weeks)

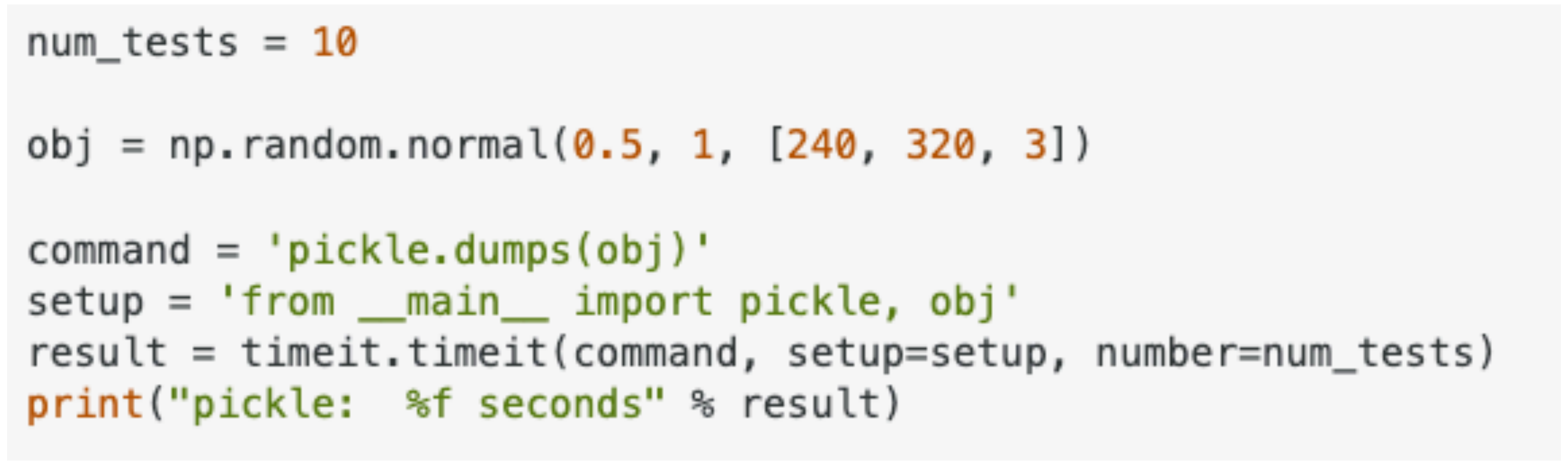
- Data Models evolution → **Data encoding evolution** → IO & Unix Pipes → Batch processing (MapReduce) → Stream processing (Spark)

Today’s topic: Data Encoding

- Formats for Encoding Data
  - Language-Specific Formats
  - JSON, XML, and Binary Variants
  - BINARY ENCODING
- Modes of dataflow
  - Database
  - REST
  - RPC
  - GraphQL
- Summary

Why encoding?

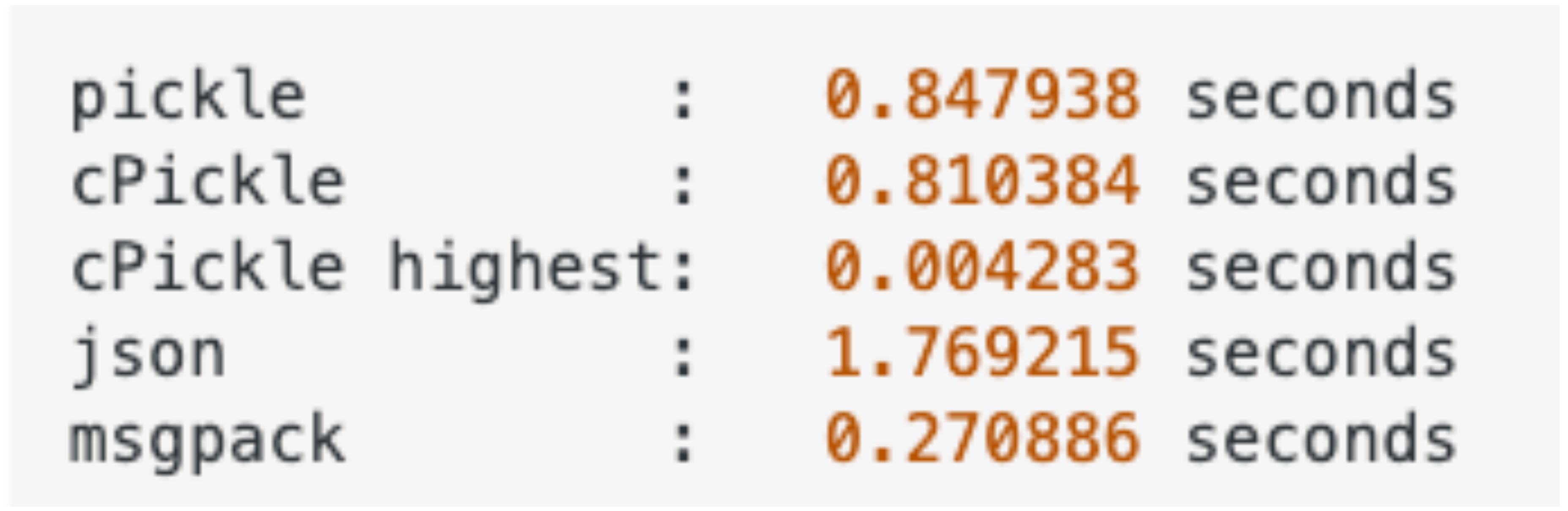
- Data in memory
  - e.g., objects, structs, lists, arrays, hash tables, trees
  - Efficient access and manipulation by the CPU (typically using pointers)
  - Why pointers? => Random address access
- Data in storage or network
  - No pointers.
  - Self-contained sequence of bytes.



```python
num_tests = 10

obj = np.random.normal(0.5, 1, [240, 320, 3])

command = 'pickle.dumps(obj)'
setup = 'from __main__ import pickle, obj'
result = timeit.timeit(command, setup=setup, number=num_tests)
print("pickle:  %f seconds" % result)
```



```
pickle         :   0.847938 seconds
cPickle        :   0.810384 seconds
cPickle highest:   0.004283 seconds
json           :   1.769215 seconds
msgpack        :   0.270886 seconds
```

https://stackoverflow.com/questions/2259270/pickle-or-json

Language-Specific Formats

- Java: java.io.Serializable;
- Python has pickle;
- Pros:
  - Convenient: in-memory objects to be saved and restored.
- Cons:
  - Tied to a programming language.
  - Decoding may lead to over-privileged behaviors.
    - e.g., remote execution.
  - Versioning, forward and backward compatibility
  - Efficiency
- Summary: quick, dirty, small individual projects

JSON, XML, CSV

- Python: Json dump.
- JSON, XML, CSV: human-readable but verbose.
- JSON: browser friendly and simple
- Common cons:
  - too verbose and unnecessarily complicated
  - ambiguity around the encoding of numbers
    - XML and CSV don’t distinguish a number and a string that happens to consist of digits
    - JSON doesn’t distinguish integers and floating-point numbers, and it doesn’t specify a precision.

JSON, XML, CSV

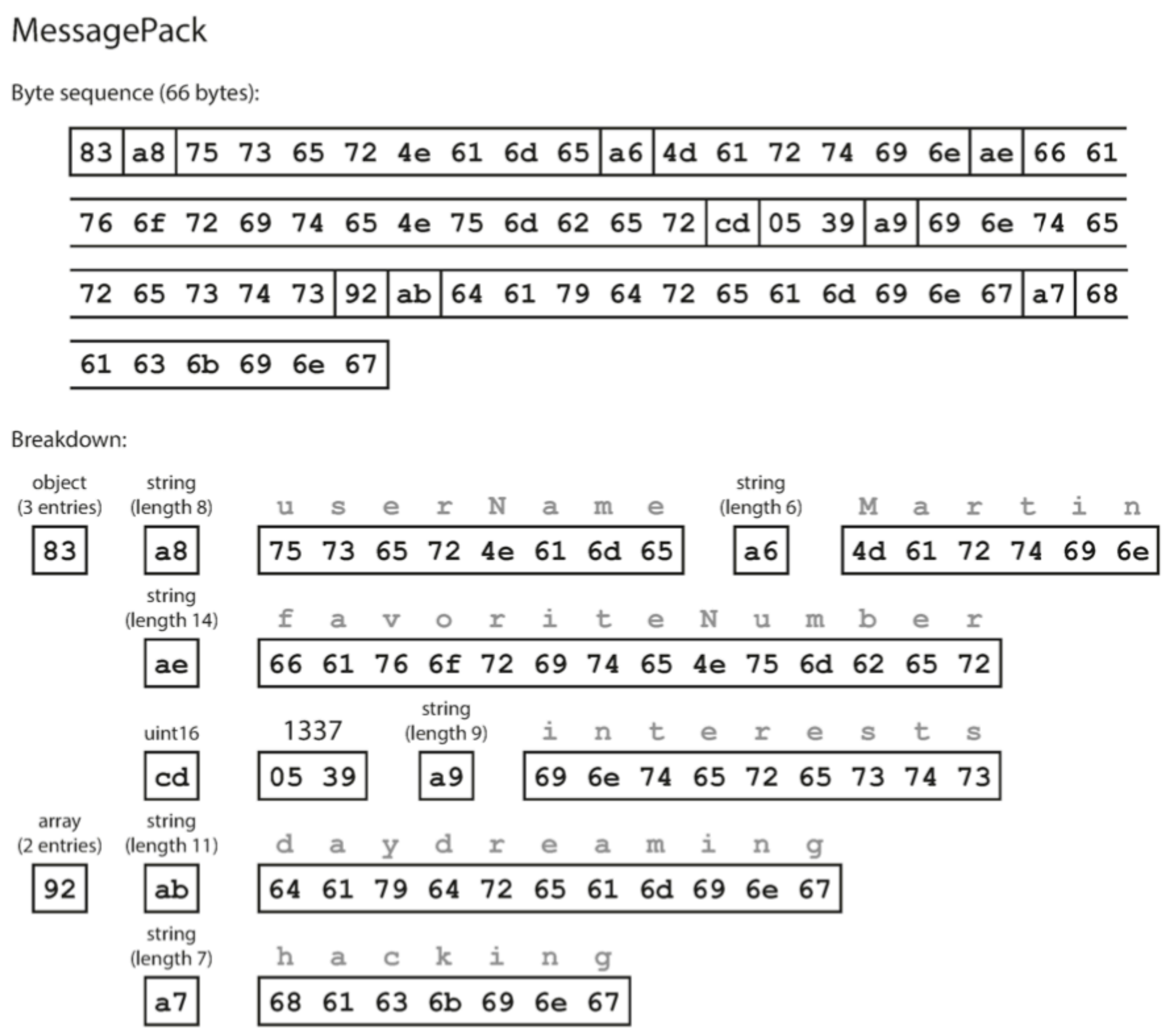
- Python: Json dump.
- Common cons:
  - JSON and XML have good support for Unicode character strings.
  - There is optional schema support for both XML and JSON
  - CSV does not have any schema,

Example

```json
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

MessagePack

```
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```


MessagePack

Byte sequence (66 bytes):

```
83 a8 75 73 65 72 4e 61 6d 65 a6 4d 61 72 74 69 6e ae 66 61
76 6f 72 69 74 65 4e 75 6d 62 65 72 cd 05 39 a9 69 6e 74 65
72 65 73 74 73 92 ab 64 61 79 64 72 65 61 6d 69 6e 67 a7 68
61 63 6b 69 6e 67
```

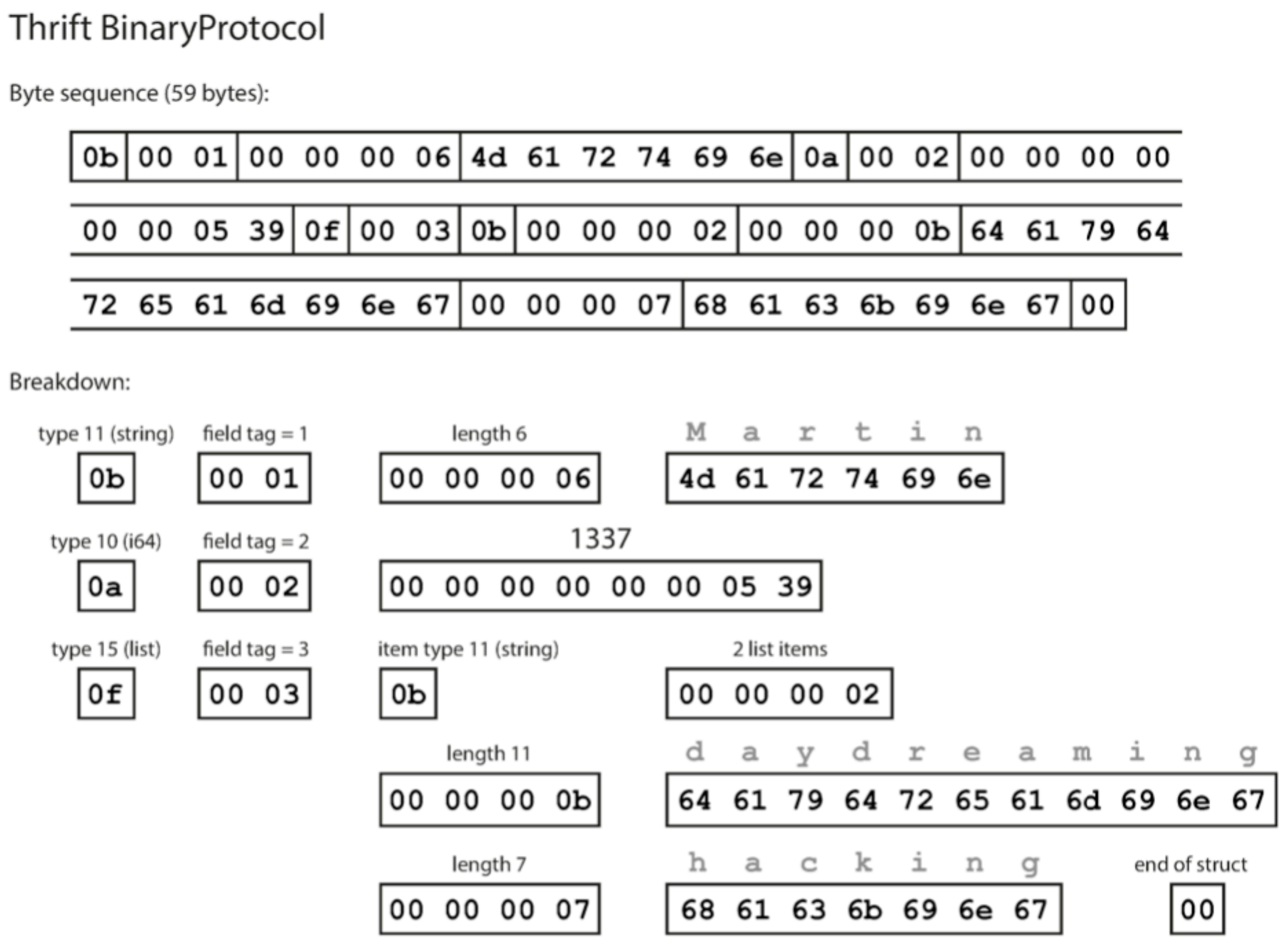
Breakdown:

object (3 entries) 83
string (length 8) a8
u s e r N a m e 75 73 65 72 4e 61 6d 65
string (length 6) a6
M a r t i n 4d 61 72 74 69 6e
string (length 14) ae
f a v o r i t e N u m b e r 66 61 76 6f 72 69 74 65 4e 75 6d 62 65 72
uint16 cd
1337 05 39
string (length 9) a9
i n t e r e s t s 69 6e 74 65 72 65 73 74 73
array (2 entries) 92
string (length 11) ab
d a y d r e a m i n g 64 61 79 64 72 65 61 6d 69 6e 67
string (length 7) a7
h a c k i n g 68 61 63 6b 69 6e 67

Thrift BinaryProtocol

```
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

```
struct Person {
  1: required string       userName,
  2: optional i64          favoriteNumber,
  3: optional list<string> interests
}
```

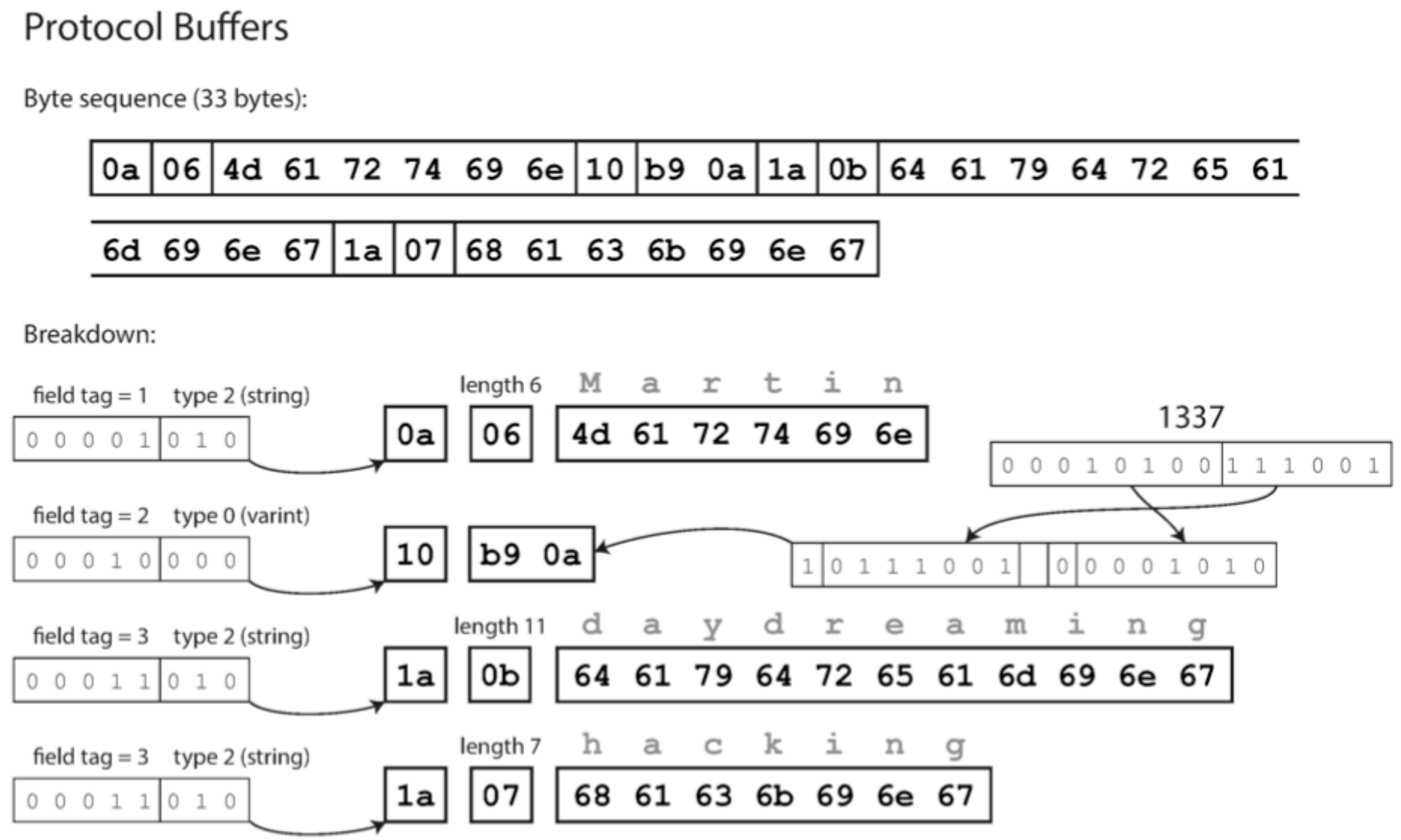


Thrift Binary Protocol v.s. MessagePack

- Same:
  - each field has a type annotation
  - a length indication
  - strings also encoded as ASCII
- Diff:
  - there are no field names
  - Instead, contains field tags
  - 59 bytes vs. 81 bytes

More system performance

- Protobuff, Thrift CompactProtocol
- Key ideas:
  - Packing the field type and tag number into a single byte
  - Using variable-length integer.



Schema evolution:

- Field tags
    - to maintain backward compatibility, every field you add after the initial deployment of the schema must be optional or have a default value.
    - only remove a field that is optional (a required field can never be removed)
    - never use the same tag number again
- Data types:
    - Possible but huge cost. May lose precision or get truncated.
    - Many language specific tricks.

The Merits of Schemas

- more compact than the “binary JSON” variants => omit field names.
- The schema & documentation.
- Hard to manually maintained.
- Keeping a database of schemas allows you to check forward and backward compatibility of schema changes.
- Code generation.

Today's topic: Data Encoding

- Formats for Encoding Data
  - Language-Specific Formats
  - JSON, XML, and Binary Variants
  - BINARY ENCODING
- Modes of dataflow
  - Database
  - REST, RPC, GraphQL
  - Message-passing
- Summary

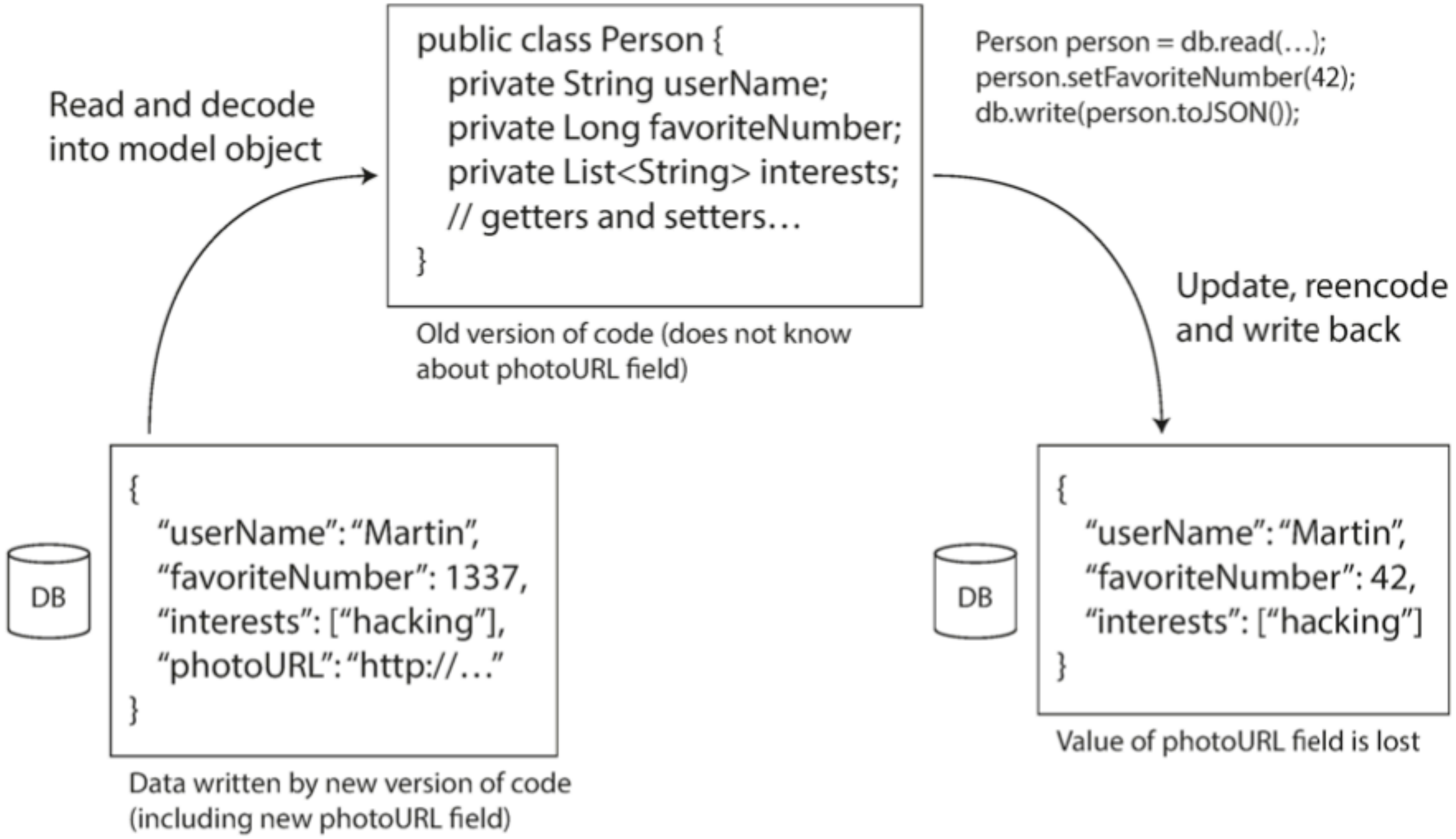
Modes of dataflow

- Whenever you want to send data to another process with which you don’t share memory, …
- Dataflow:
    - Encoding + Sharing (flowing) + Decoding
- Via databases (see “Dataflow Through Databases”)
- Via service calls (see “Dataflow Through Services: REST and RPC”)
- Via asynchronous message passing (see “Message-Passing Dataflow”)

Dataflow Through Databases

- The write process encodes data
- The read process decodes data
- Can be the same process.
- Backward compatibility
  - The write process may not match with the read process.

Backward compatibility



Read and decode into model object
Old version of code (does not know about photoURL field)
Update, reencode and write back
DB
Data written by new version of code (including new photoURL field)
DB
Value of photoURL field is lost

```
public class Person {
  private String userName;
  private Long favoriteNumber;
  private List<String> interests;
  // getters and setters…
}
```

```
Person person = db.read(…);
person.setFavoriteNumber(42);
db.write(person.toJSON());
```

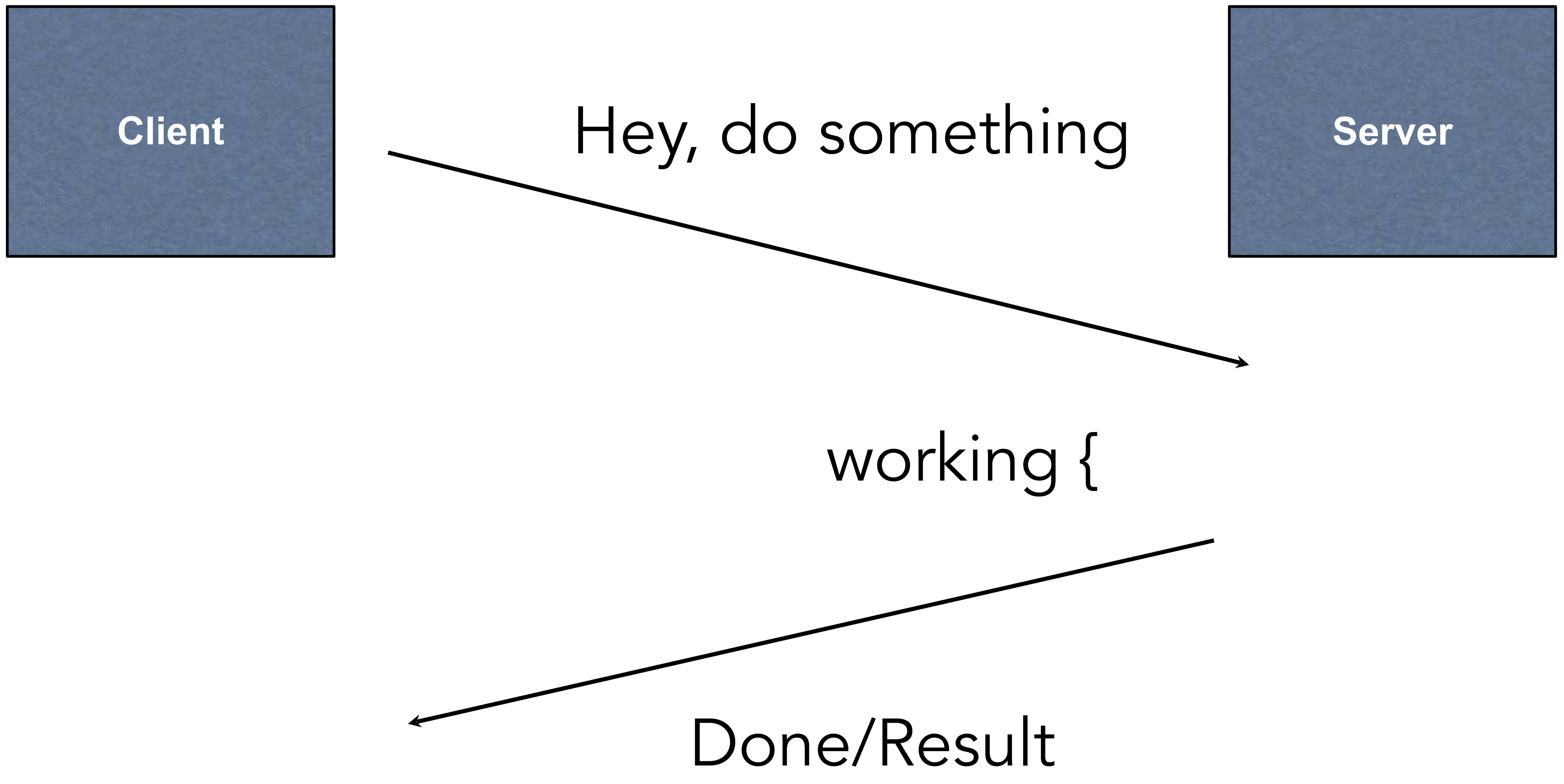
```
{
  "userName": "Martin",
  "favoriteNumber": 1337,
  "interests": ["hacking"],
  "photoURL": "http://…"
}
```

```
{
  "userName": "Martin",
  "favoriteNumber": 42,
  "interests": ["hacking"]
}
```

Multiple strategies

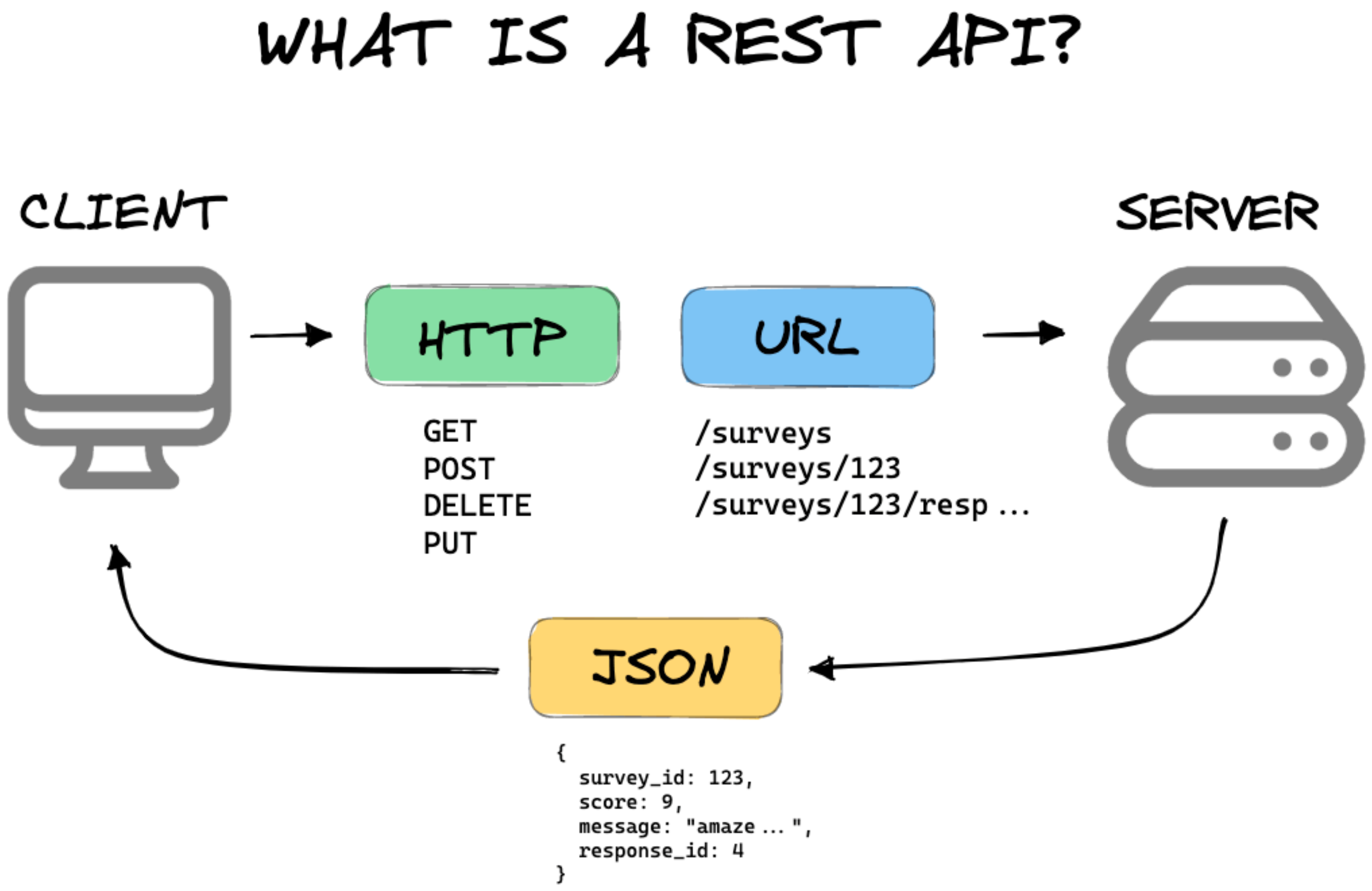
- Rewriting (migrating)
  - OLAP?
- Relational db allow simple schema changes
  - Adding a new column with a null default value
  - Old row behaviors?

Dataflow Through Services: REST & RPC & MPF



REST

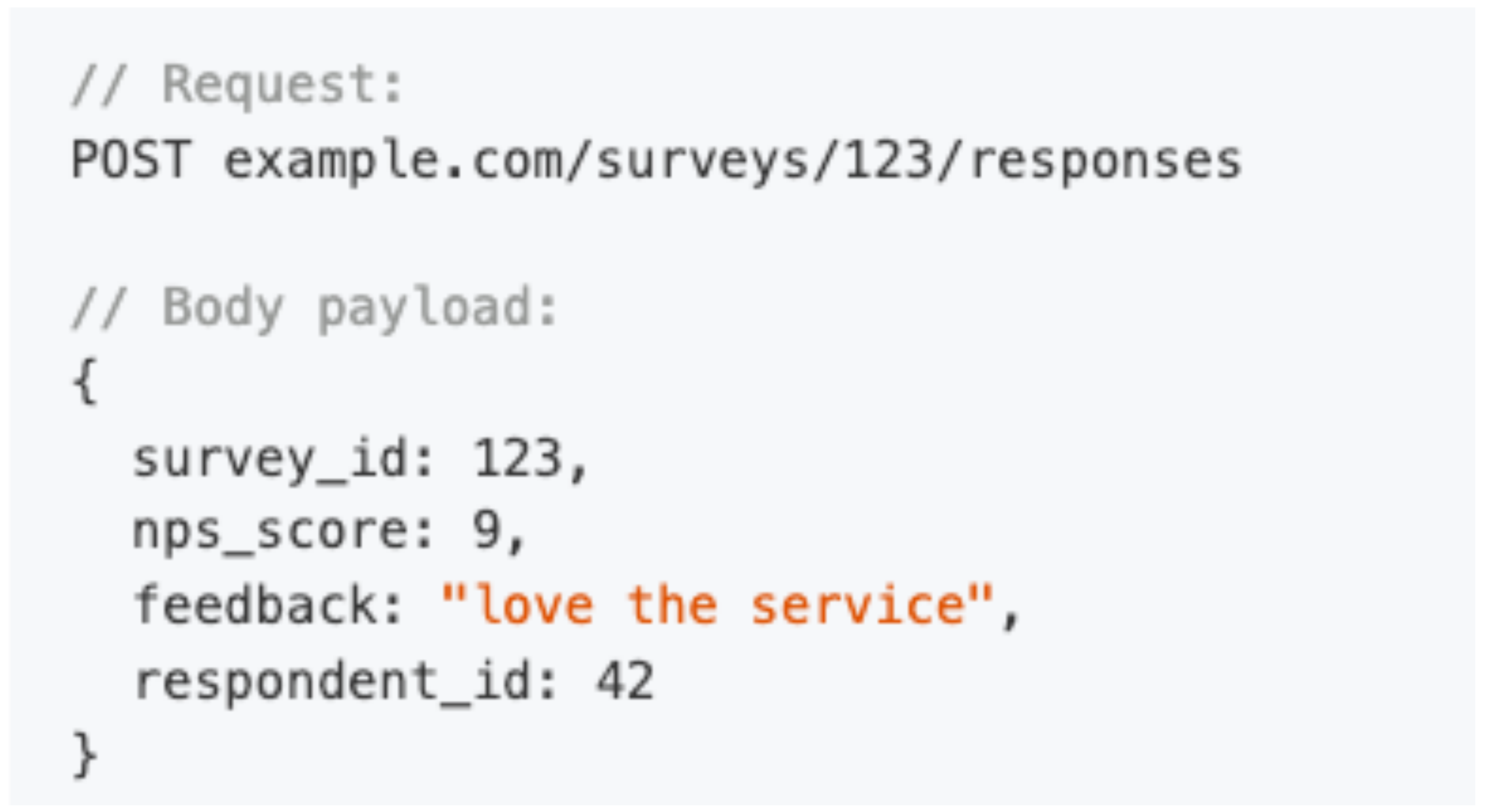
- Basic format
- HTTP verbs
- URL endpoints
- Status codes



```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```

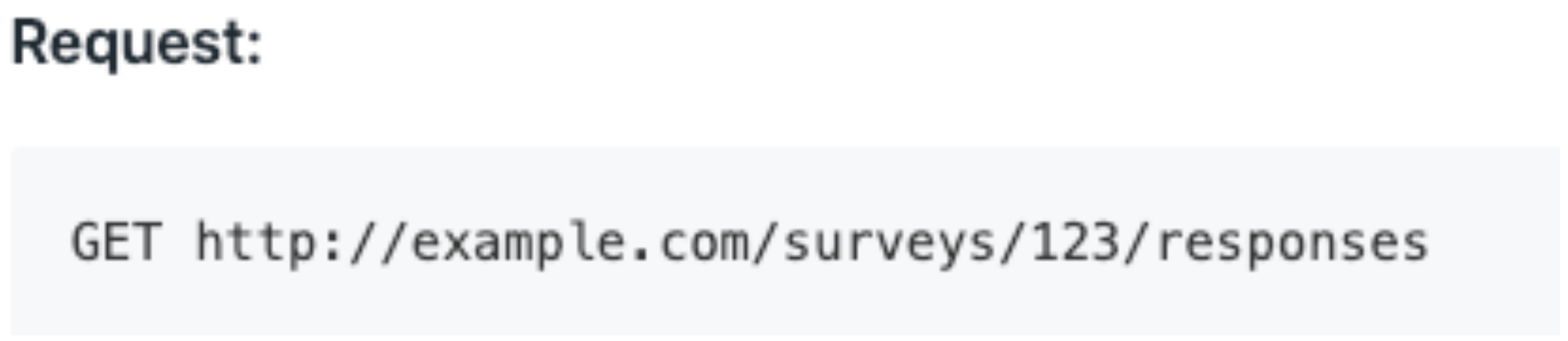
REST - Basic format

- URL endpoint
- HTTP verbs
- Body
- Response



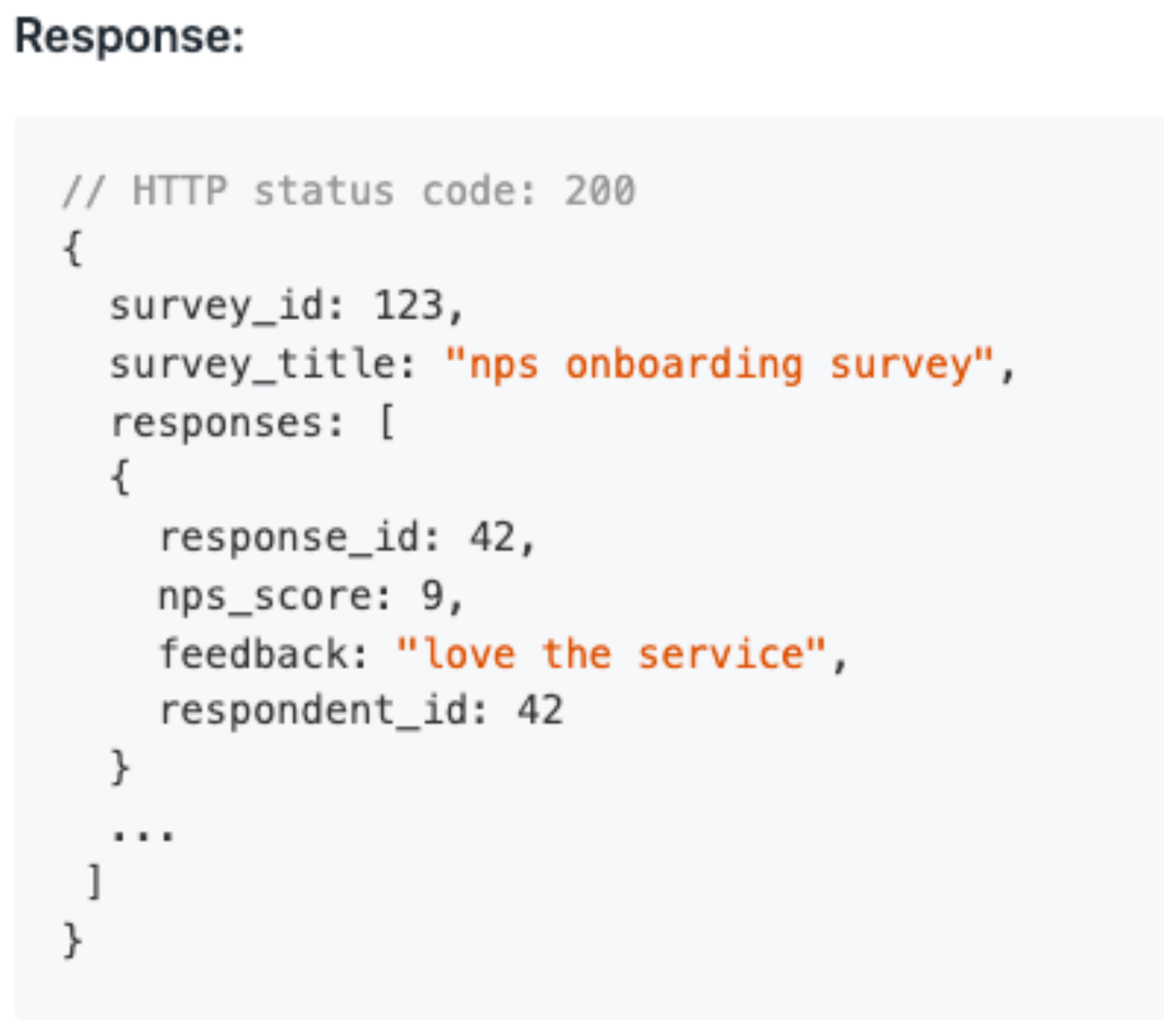
```
// Request:
POST example.com/surveys/123/responses

// Body payload:
{
  survey_id: 123,
  nps_score: 9,
  feedback: "love the service",
  respondent_id: 42
}
```



Request:

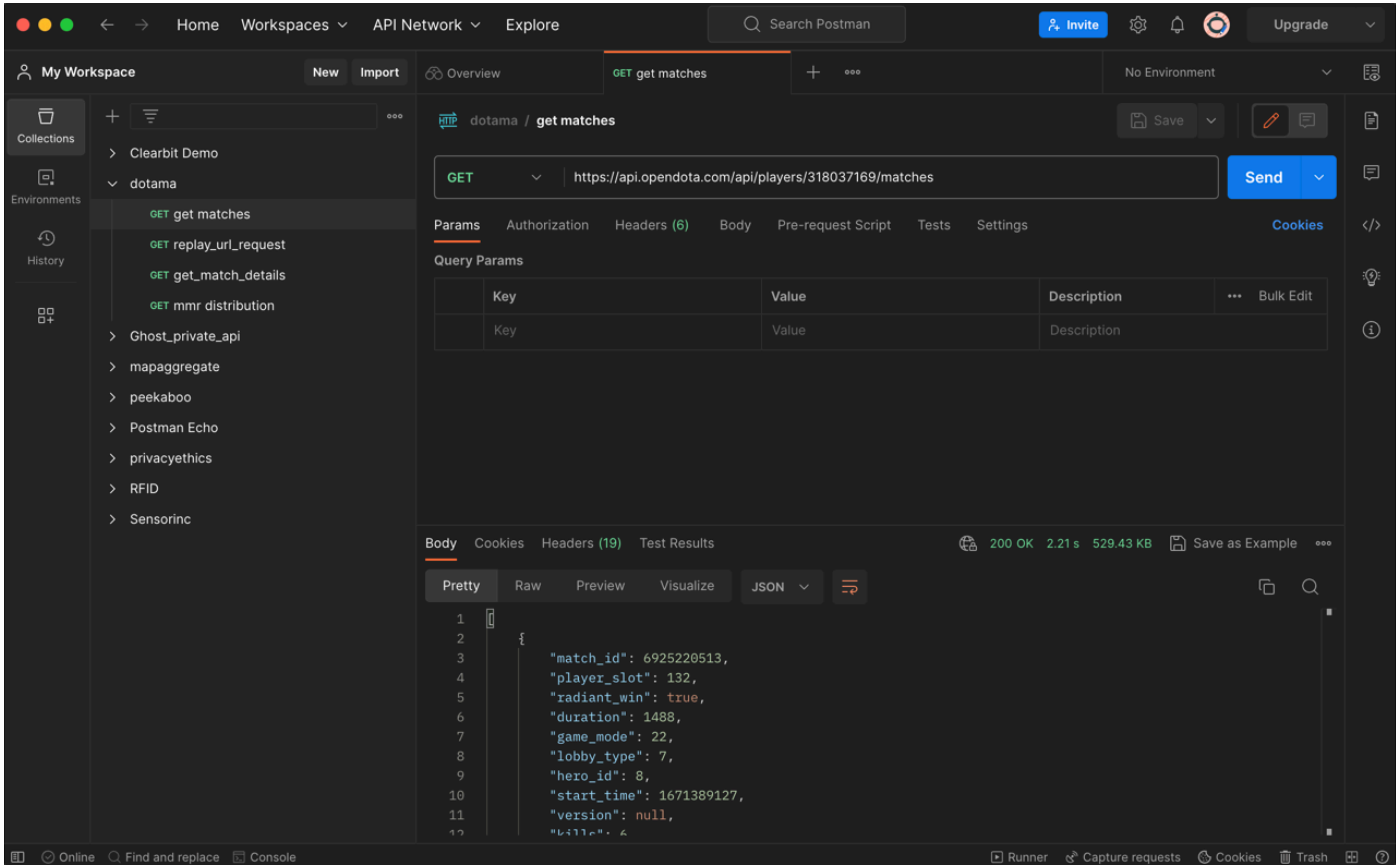
```
GET http://example.com/surveys/123/responses
```



Response:

```
// HTTP status code: 200
{
  survey_id: 123,
  survey_title: "nps onboarding survey",
  responses: [
   {
     response_id: 42,
     nps_score: 9,
     feedback: "love the service",
     respondent_id: 42
   }
   ...
  ]
}
```

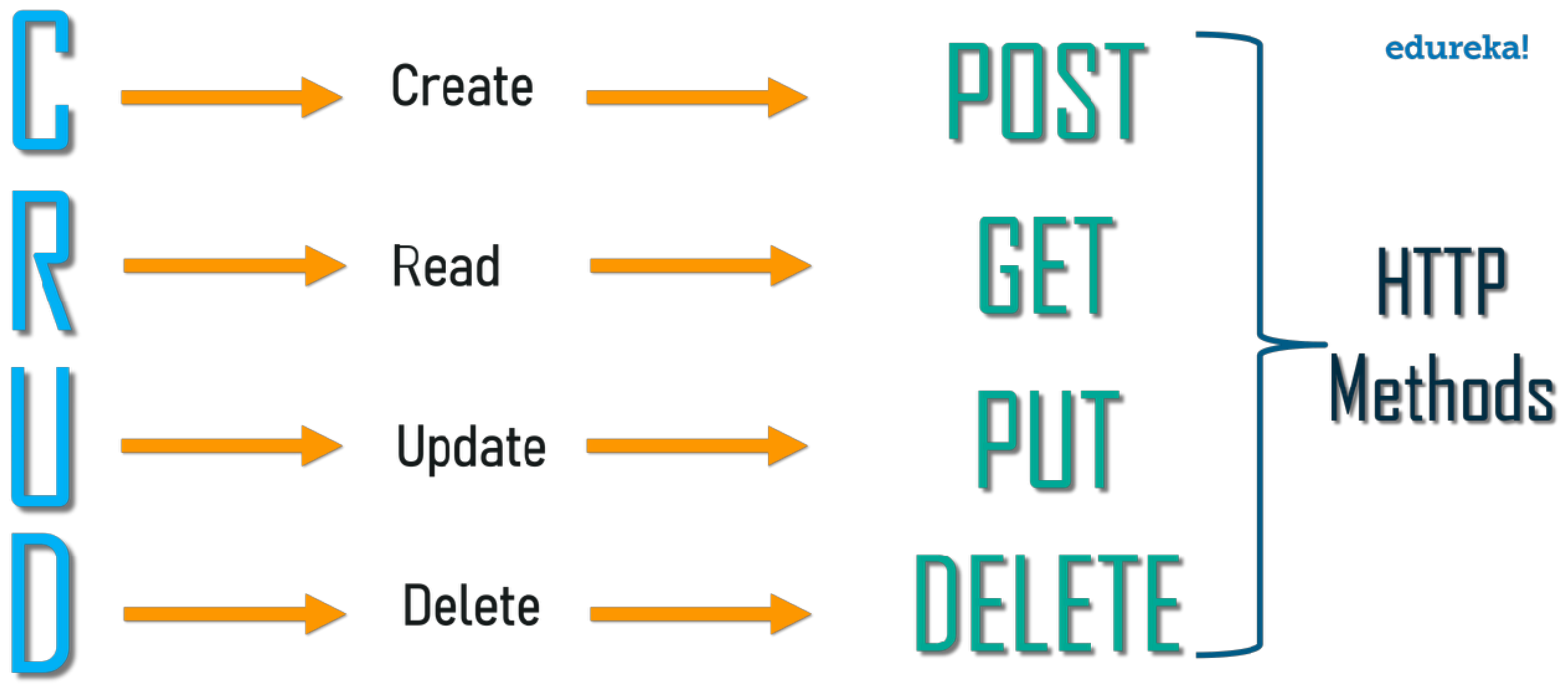
REST - Postman



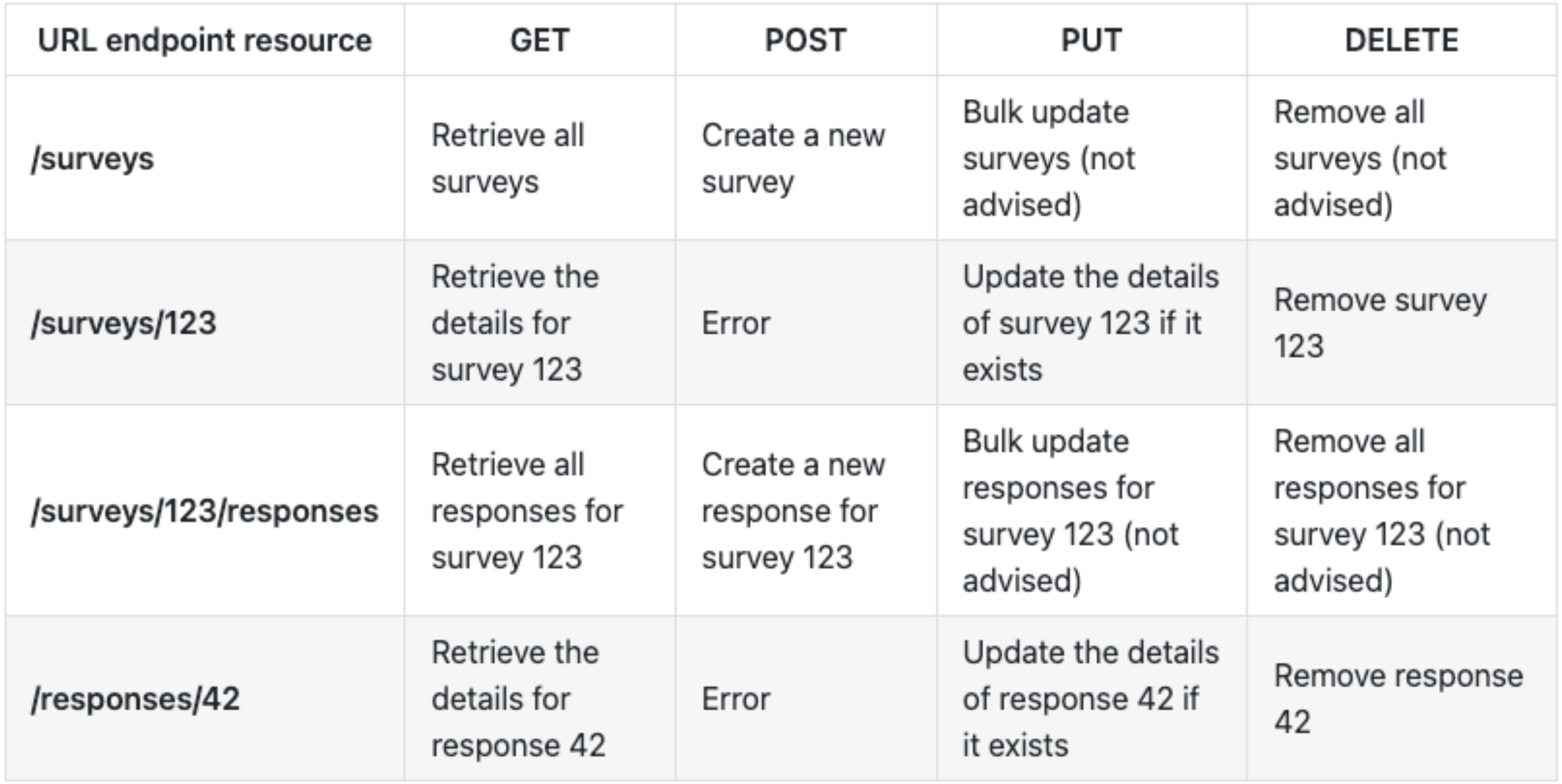
GET https://api.opendota.com/api/players/318037169/matches

```
[
    {
        "match_id": 6925220513,
        "player_slot": 132,
        "radiant_win": true,
        "duration": 1488,
        "game_mode": 22,
        "lobby_type": 7,
        "hero_id": 8,
        "start_time": 1671389127,
        "version": null,
```

REST - HTTP verbs

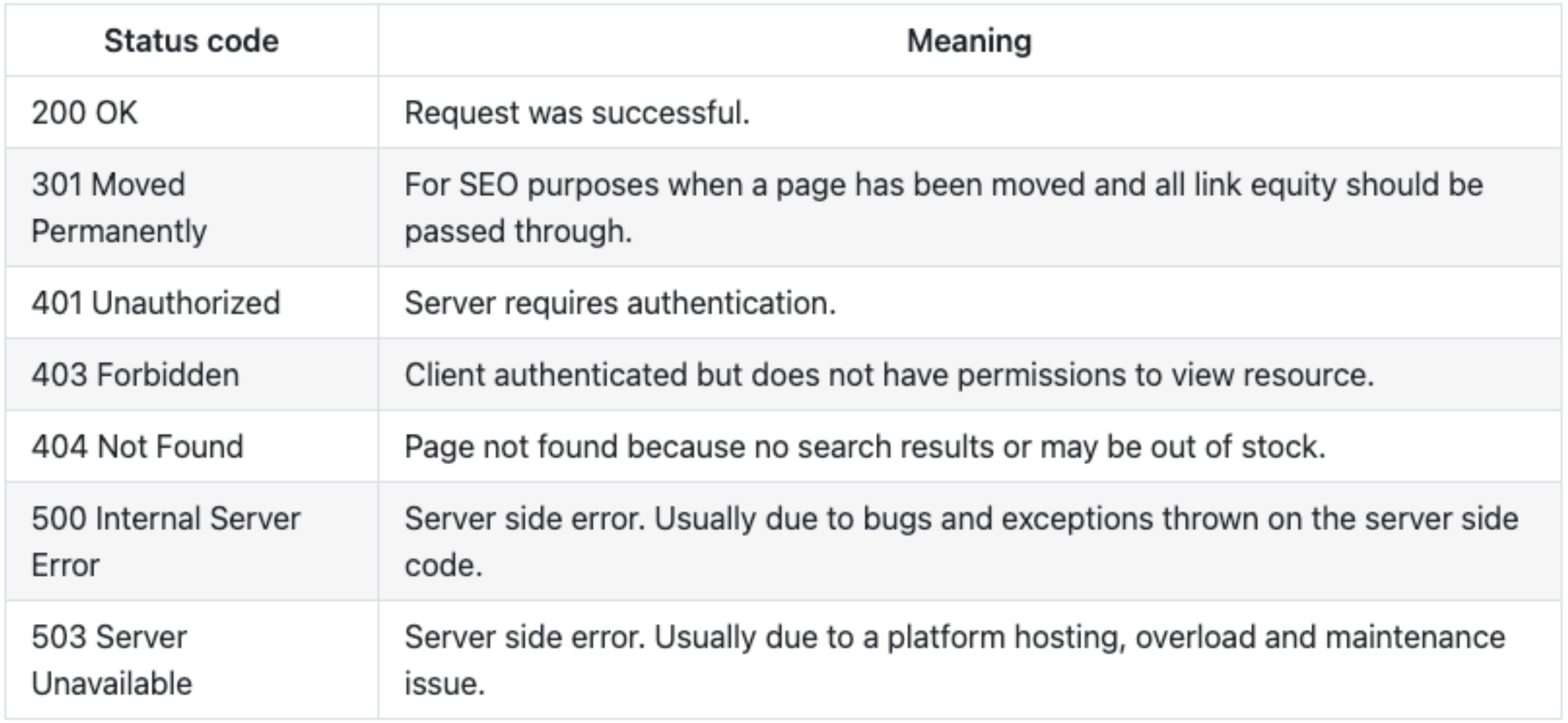


REST - URL endpoint



URL endpoint resource	GET	POST	PUT	DELETE
/surveys	Retrieve all surveys	Create a new survey	Bulk update surveys (not advised)	Remove all surveys (not advised)
/surveys/123	Retrieve the details for survey 123	Error	Update the details of survey 123 if it exists	Remove survey 123
/surveys/123/responses	Retrieve all responses for survey 123	Create a new response for survey 123	Bulk update responses for survey 123 (not advised)	Remove all responses for survey 123 (not advised)
/responses/42	Retrieve the details for response 42	Error	Update the details of response 42 if it exists	Remove response 42

REST - Status codes

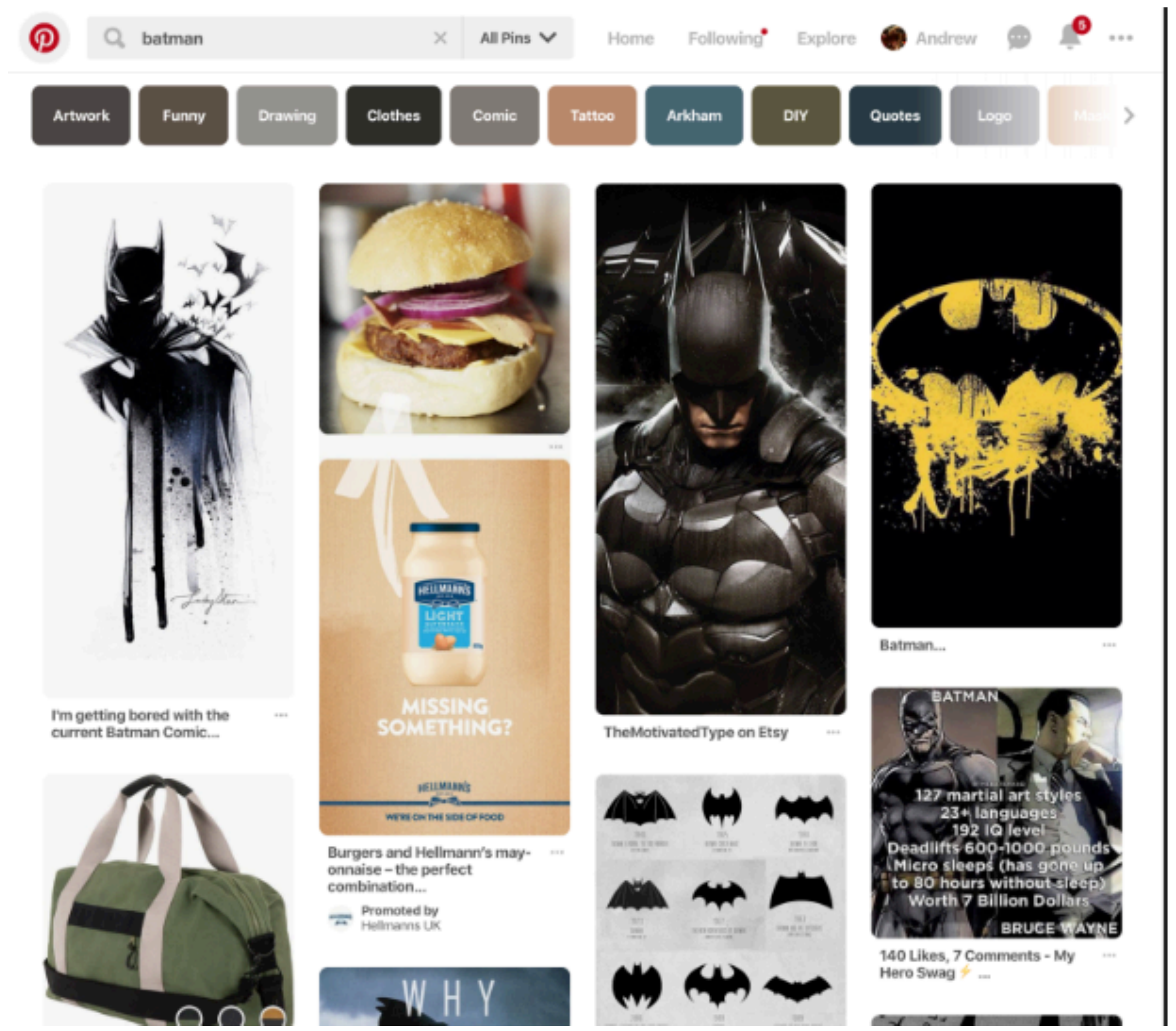


Status code	Meaning
200 OK	Request was successful.
301 Moved Permanently	For SEO purposes when a page has been moved and all link equity should be passed through.
401 Unauthorized	Server requires authentication.
403 Forbidden	Client authenticated but does not have permissions to view resource.
404 Not Found	Page not found because no search results or may be out of stock.
500 Internal Server Error	Server side error. Usually due to bugs and exceptions thrown on the server side code.
503 Server Unavailable	Server side error. Usually due to a platform hosting, overload and maintenance issue.

GraphQL

- Created by Facebook in 2012
- GraphQL is a replacement for REST
- GraphQL uses a QUERY LANGUAGE
- GraphQL is a PATTERN not a technology

Why GraphQL



- One API call for collection
- Then one call per item for details

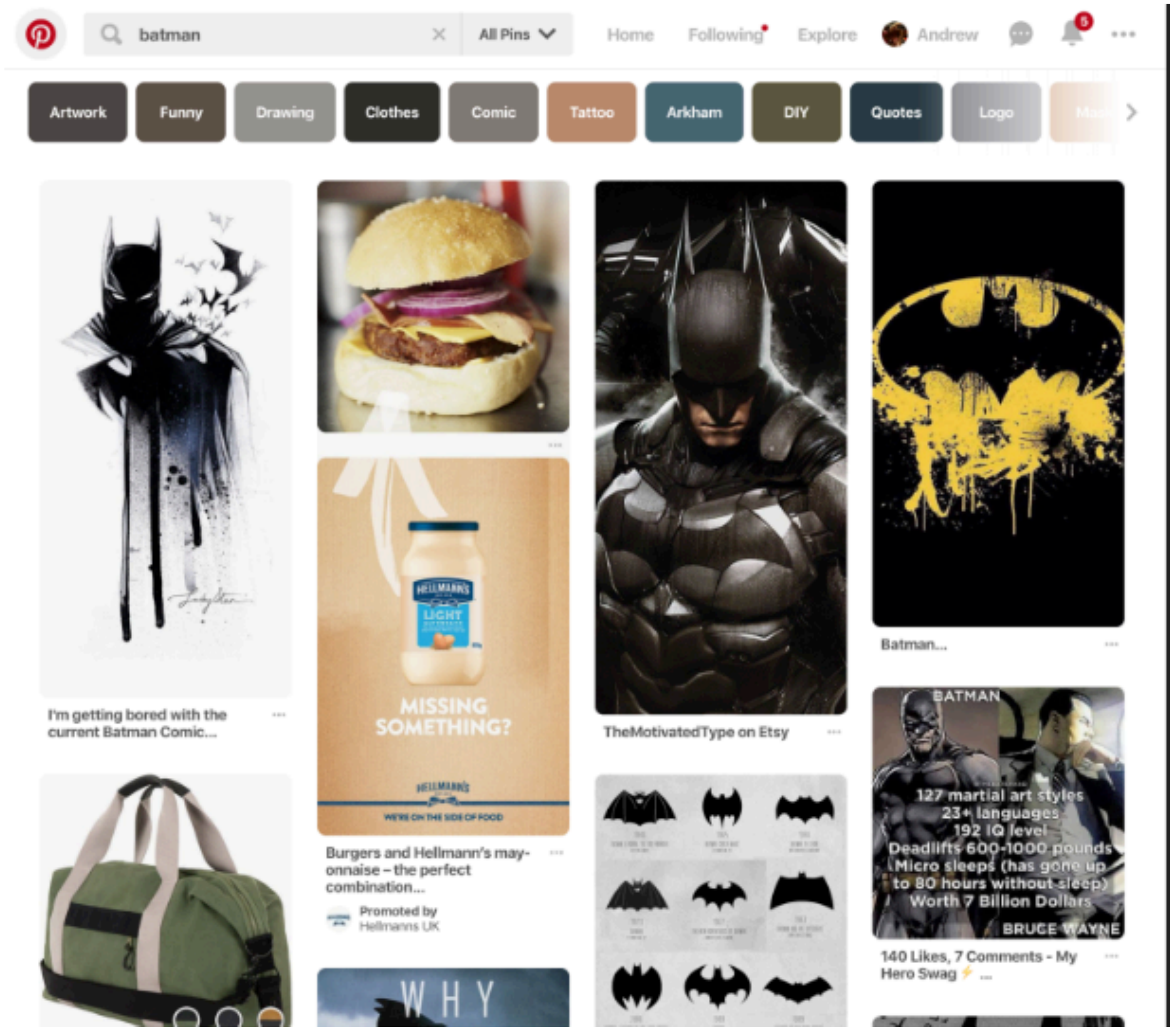
GET /results

GET /results/1234

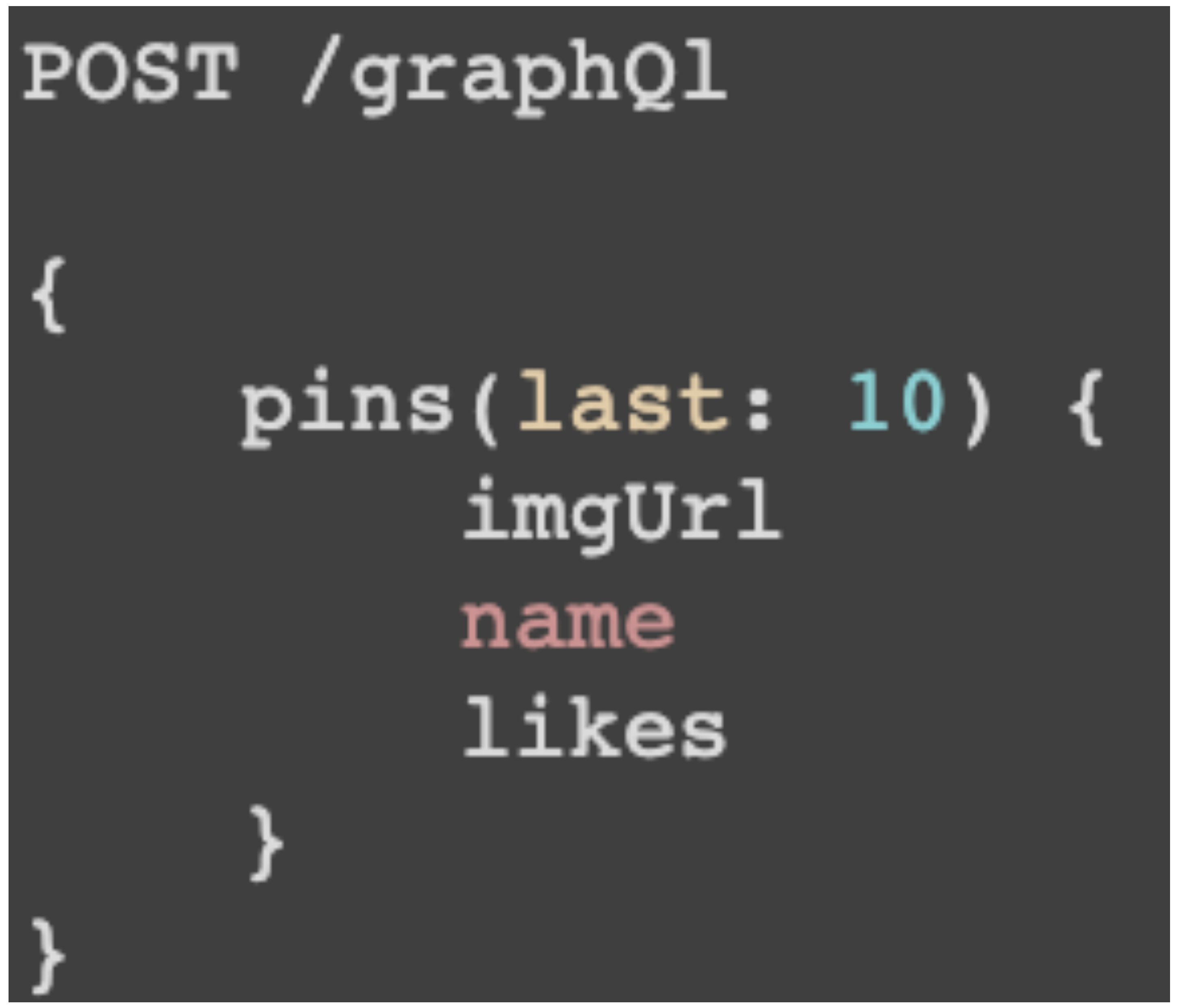
GET /results/5678

GET /results/9012

GraphQL vs REST



- GraphQL can bundle lots of data into one query instead of many

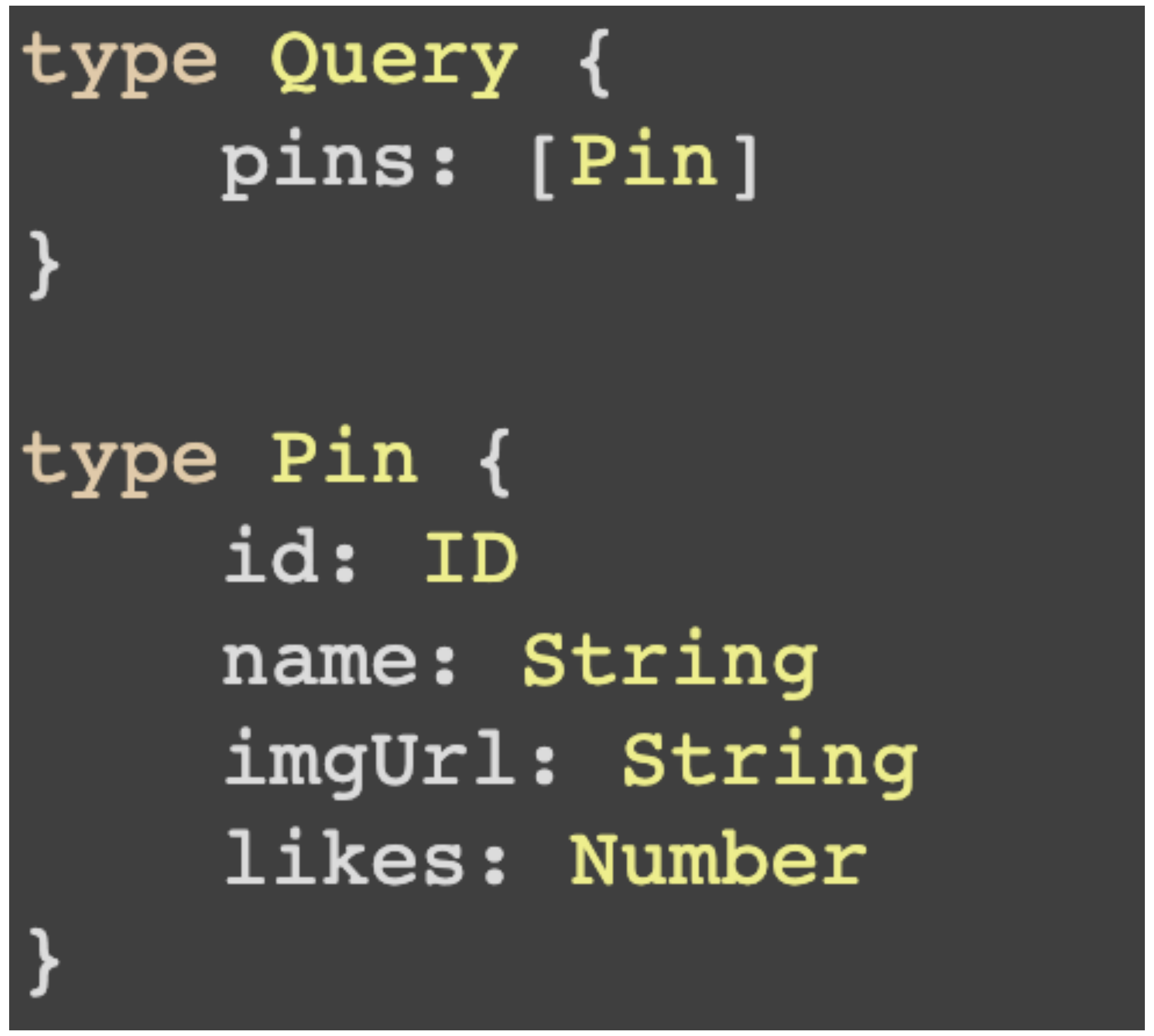


```
POST /graphQl

{
    pins(last: 10) {
        imgUrl
        name
        likes
    }
}
```

How does it work? (Schema)

First, you describe your data with a schema



```
type Query {
    pins: [Pin]
}

type Pin {
    id: ID
    name: String
    imgUrl: String
    likes: Number
}
```

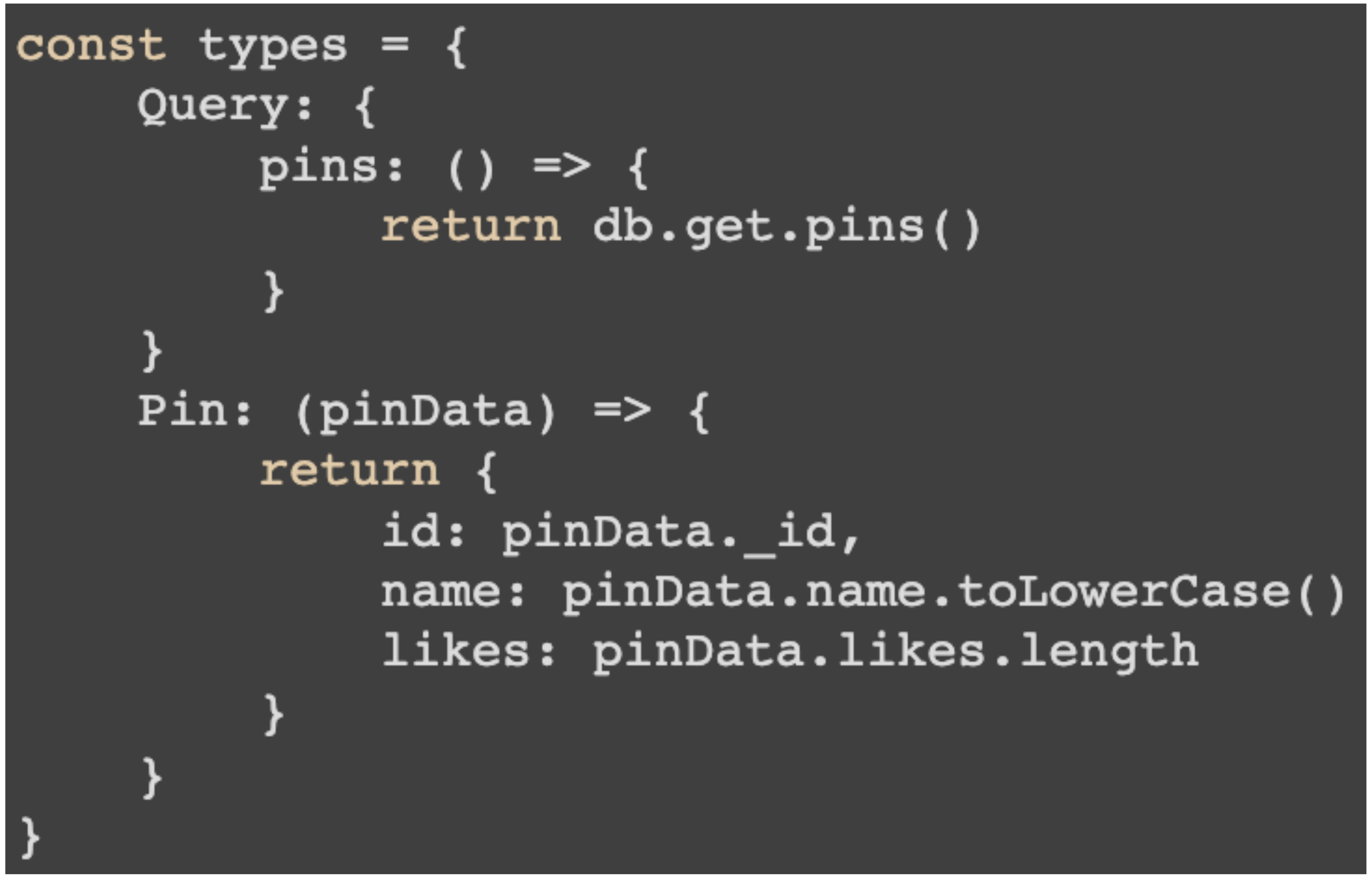
Strongly Typed

Types can be a Scalar (boolean, string, ID) or Object with fields

This schema can generate types in Typescript, or Flow.

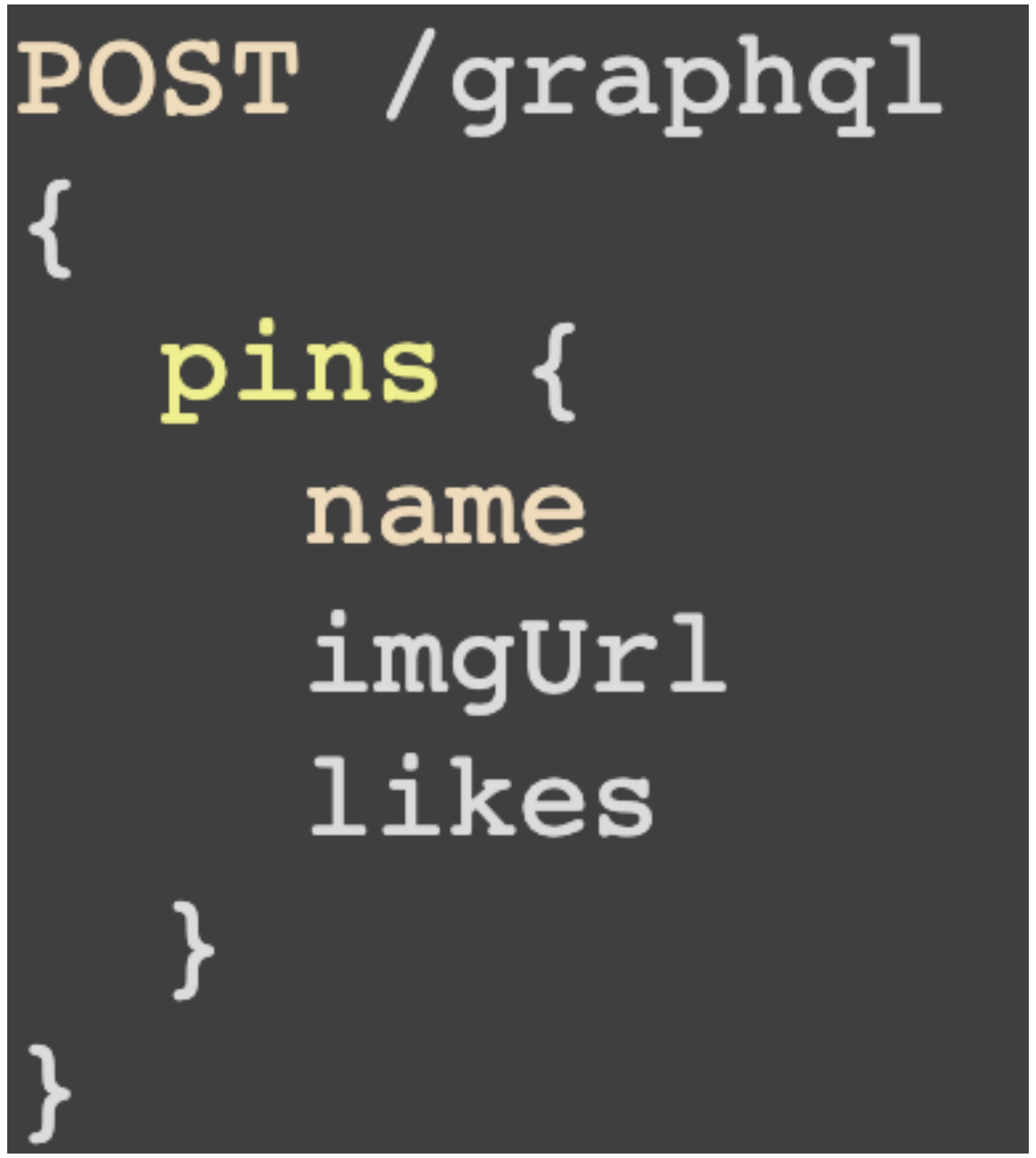
How does it work? (Resolvers)

Resolvers work at the object property level.



```
const types = {
    Query: {
        pins: () => {
            return db.get.pins()
        }
    }
    Pin: (pinData) => {
        return {
            id: pinData._id,
            name: pinData.name.toLowerCase()
            likes: pinData.likes.length
        }
    }
}
```

How does it work? (Queries)

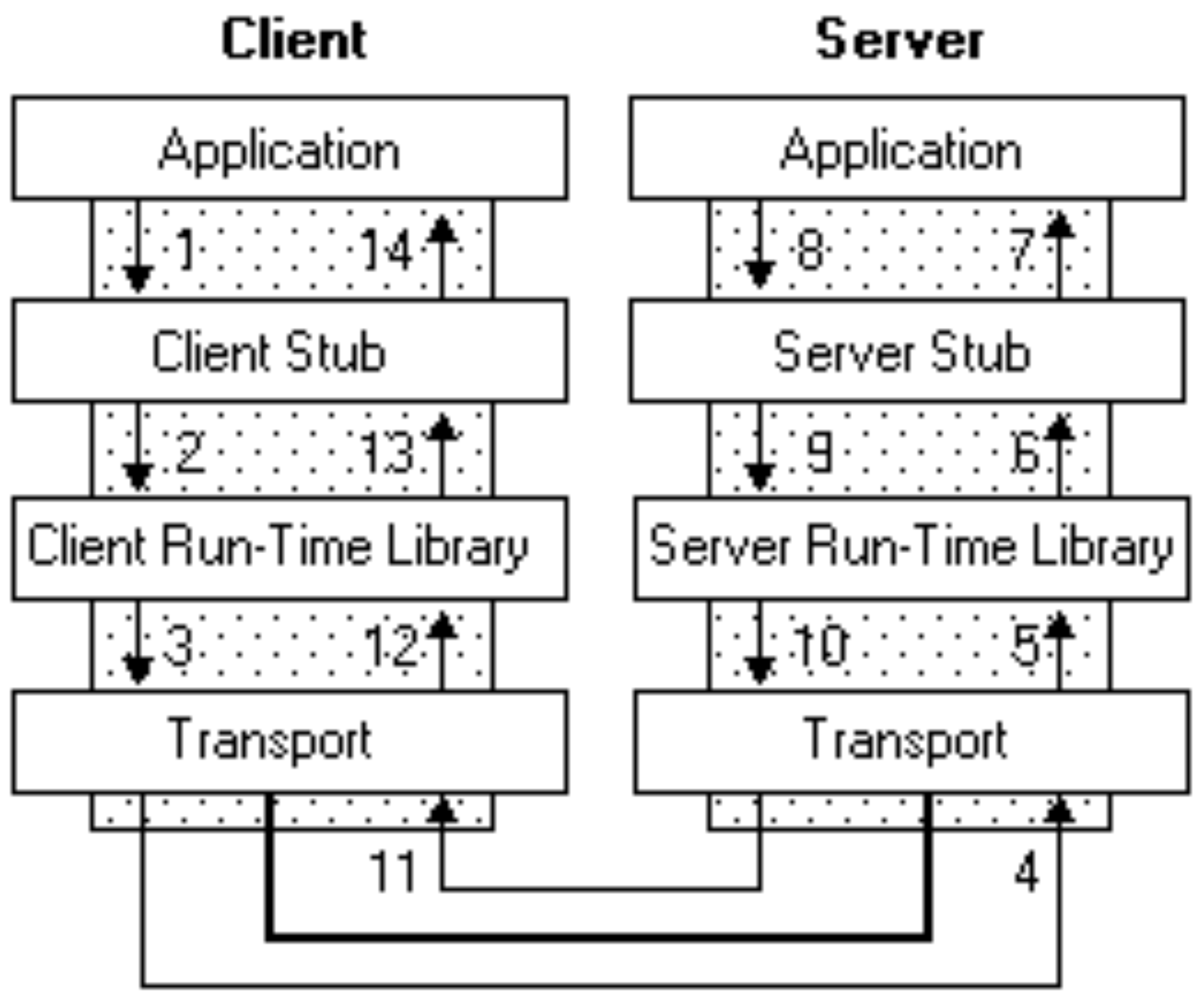


```
POST /graphql
{
  pins {
    name
    imgUrl
    likes
  }
}
```

Retrieve pins, for each pin return the name, image url, and the number of likes.

RPC - Remote Procedure Call

- A type of client/server communication
- Attempts to make remote procedure calls look like local ones



```
{ ...
   foo()
}
void foo() {
  invoke_remote_foo()
}
```

Go example

- **Client first dials the server**

```
client, err := rpc.DialHTTP("tcp", serverAddress + ":1234")
if err != nil { log.Fatal("dialing:", err) }
```

- **Then it can make a remote call:**

```
// Synchronous call
args := &server.Args{7,8}
var reply int
err = client.Call("Arith.Multiply", args, &reply)
if err != nil {
    log.Fatal("arith error:", err)
}
fmt.Printf("Arith: %d*%d=%d", args.A, args.B, reply)
```

RPC Goals

- Ease of programming
- Hide complexity
- Automates task of implementing distributed computation
- Familiar model for programmers (just make a function call)

RPC

- A remote procedure call makes a call to a remote service look like a local call
  - RPC makes transparent whether server is local or remote
  - RPC allows applications to become distributed transparently
  - RPC makes architecture of remote machine transparent
- Challenges:
  - Calling and called procedures run on different machines, environments, OS
  - Must convert to local representation of data
  - Machines and network can fail

RPC Procedure

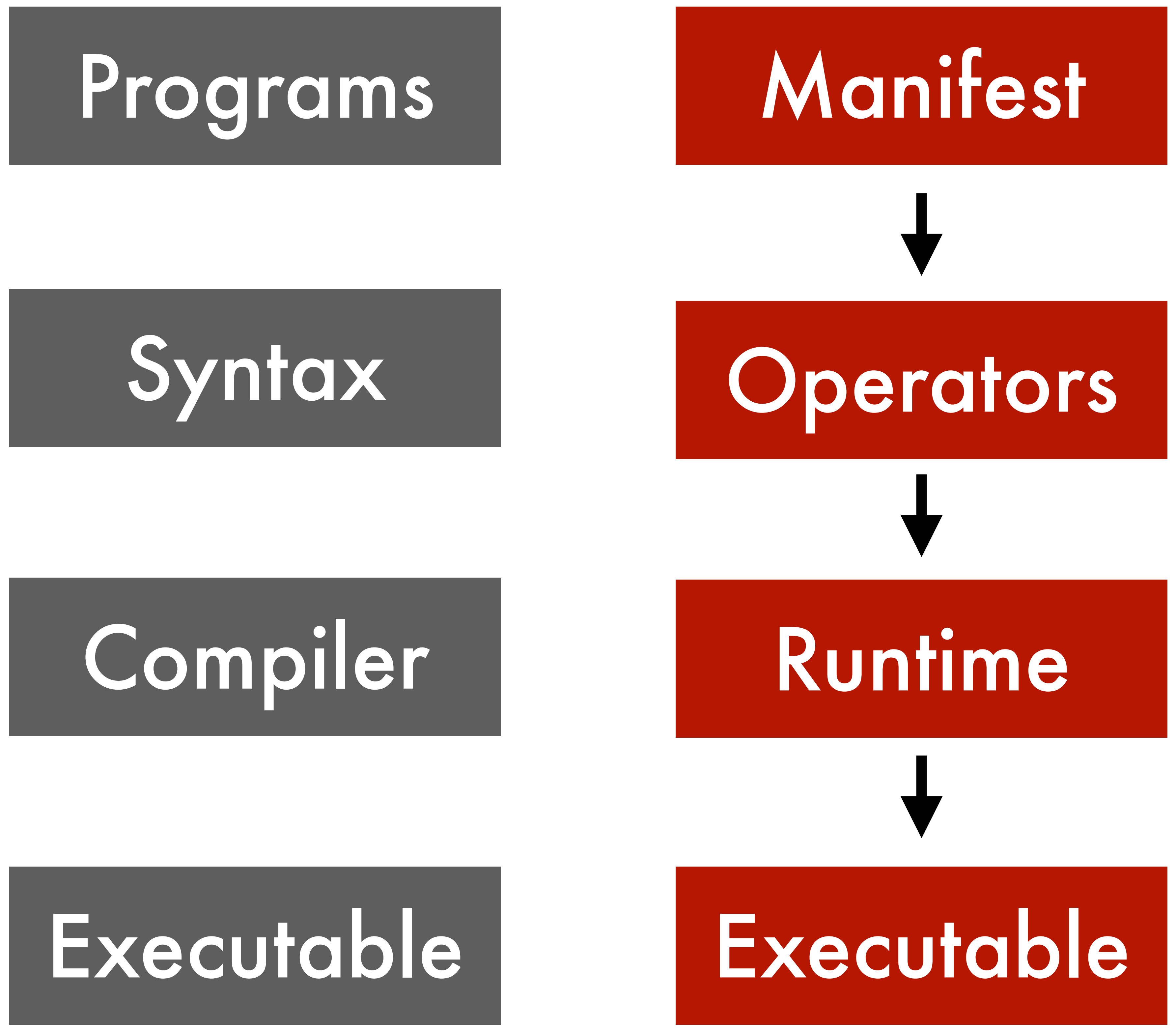
1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

Continued …

RPC Procedure

6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

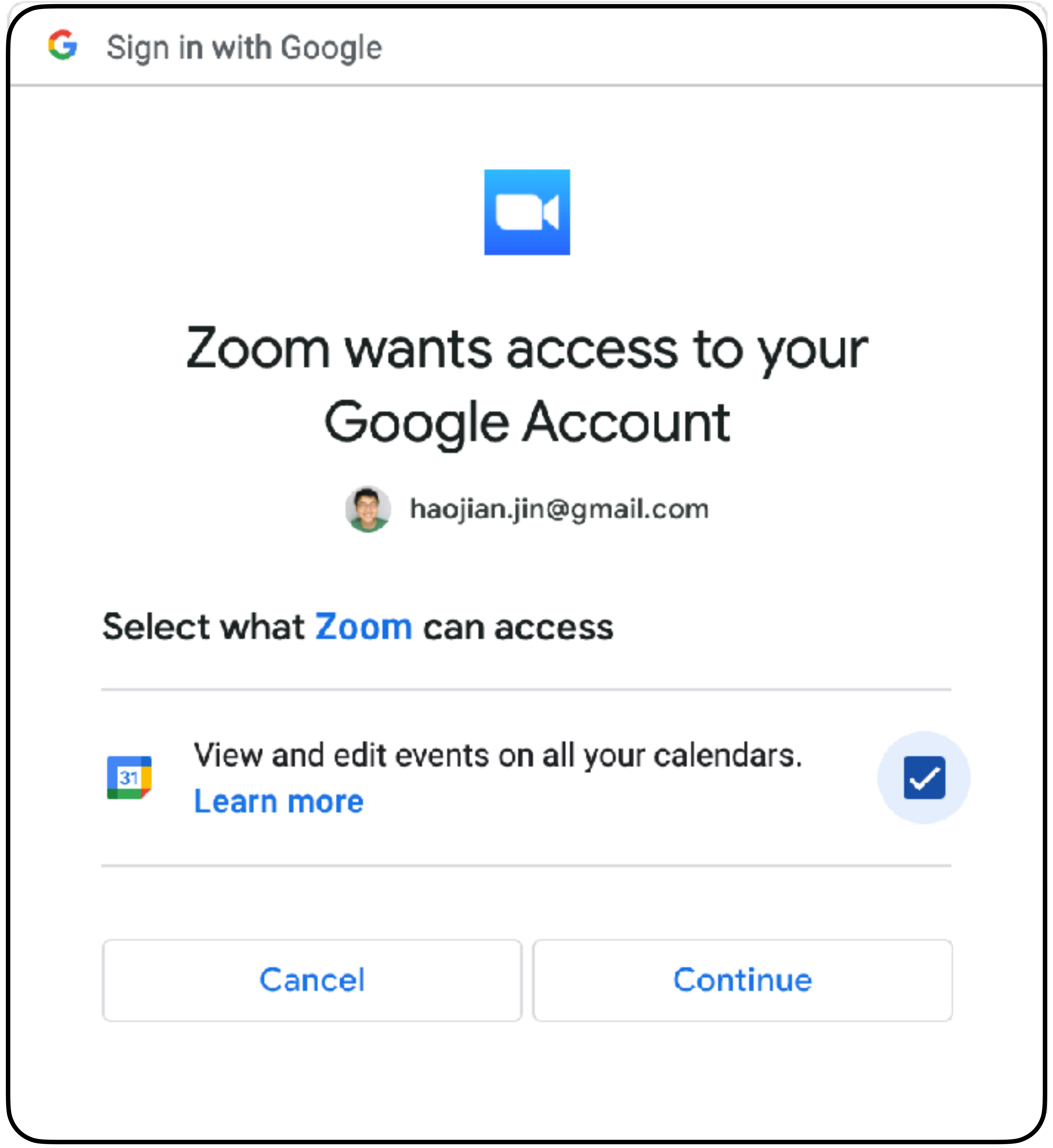
MPF - Modular Privacy Flows



A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

Zoom accesses **all** your calendar events **continuously**!


Sign in with Google

Zoom wants access to your Google Account

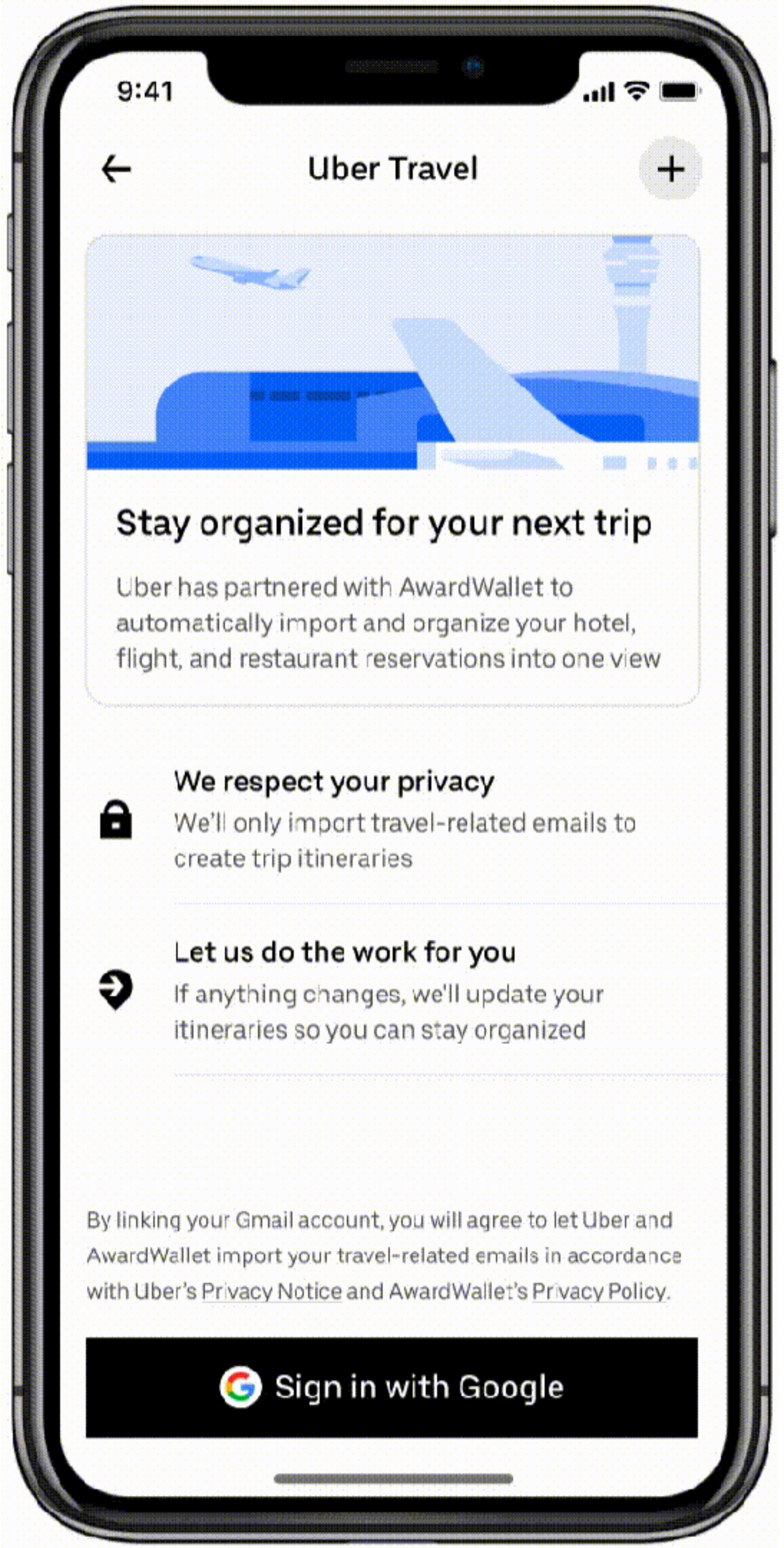
haojian.jin@gmail.com

Select what Zoom can access

View and edit events on all your calendars. Learn more ✓

Calendar events that contain https://zoom.us/xxxxx

Uber wants to see all your emails.


9:41

Uber Travel

Stay organized for your next trip

Uber has partnered with AwardWallet to automatically import and organize your hotel, flight, and restaurant reservations into one view

We respect your privacy

We'll only import travel-related emails to create trip itineraries

Let us do the work for you

If anything changes, we'll update your itineraries so you can stay organized

By linking your Gmail account, you will agree to let Uber and AwardWallet import your travel-related emails in accordance with Uber's Privacy Notice and AwardWallet's Privacy Policy.

Sign in with Google

Principle of data minimization

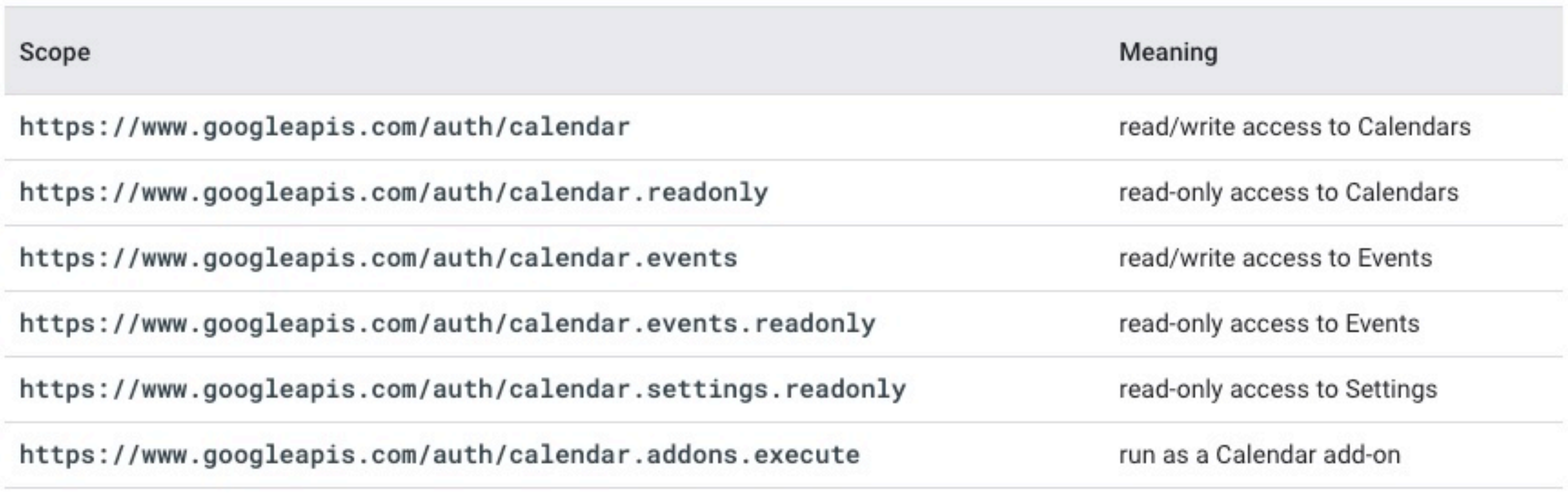
"Personal data shall be limited to ***what is necessary*** *in relation to the* ***purposes*** *for which they are processed."*

- GDPR, Article 5 (1) (c)

Principle of least privilege

"A security architecture should be designed so that each entity is granted the ***minimum system resources*** *and authorizations that the entity needs to perform its function.."*

Google APIs - All-or-nothing binary permissions

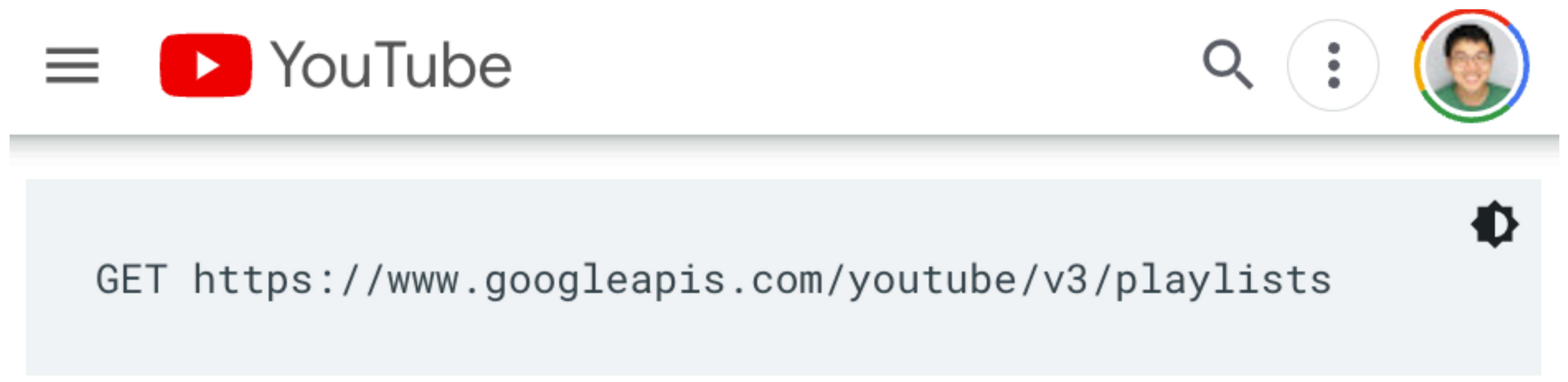


Scope	Meaning
`https://www.googleapis.com/auth/calendar`	read/write access to Calendars
`https://www.googleapis.com/auth/calendar.readonly`	read-only access to Calendars
`https://www.googleapis.com/auth/calendar.events`	read/write access to Events
`https://www.googleapis.com/auth/calendar.events.readonly`	read-only access to Events
`https://www.googleapis.com/auth/calendar.settings.readonly`	read-only access to Settings
`https://www.googleapis.com/auth/calendar.addons.execute`	run as a Calendar add-on

https://developers.google.com/calendar/api/guides/auth

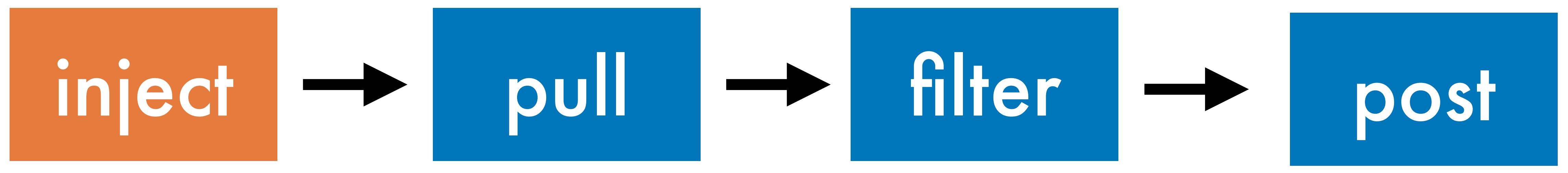
Program data transformation functions using chainable *operators*

URL-based APIs

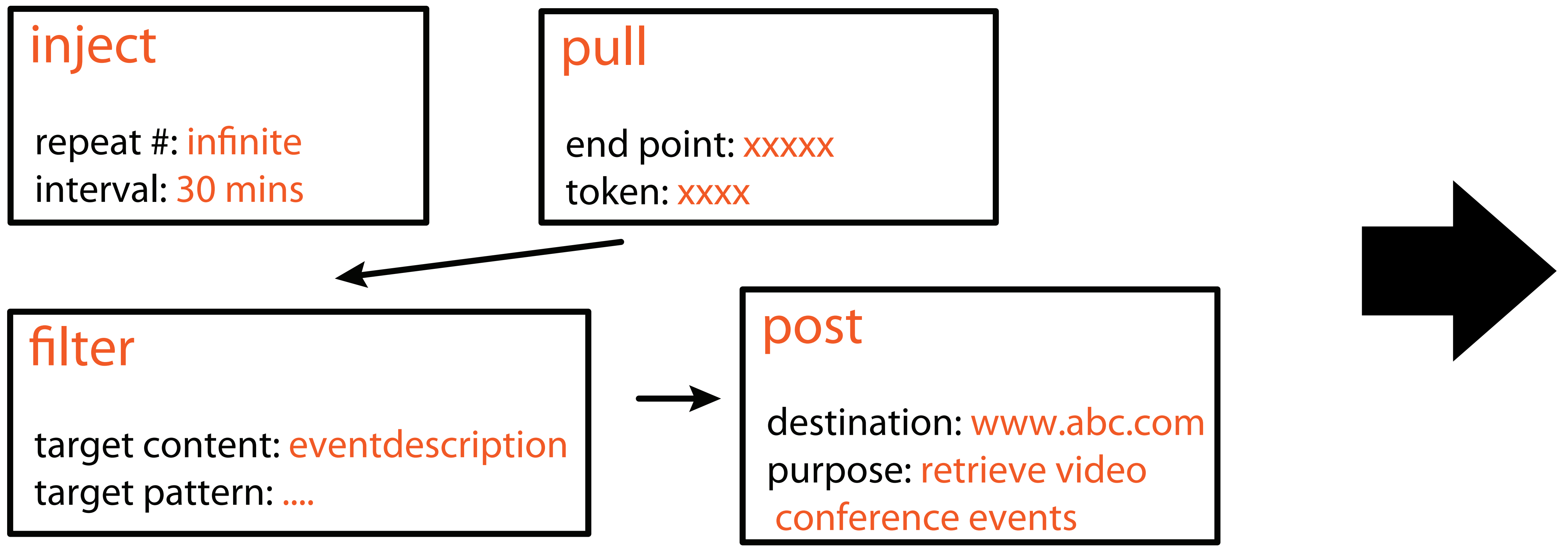


```
GET https://www.googleapis.com/youtube/v3/playlists
```

Operator-based APIs

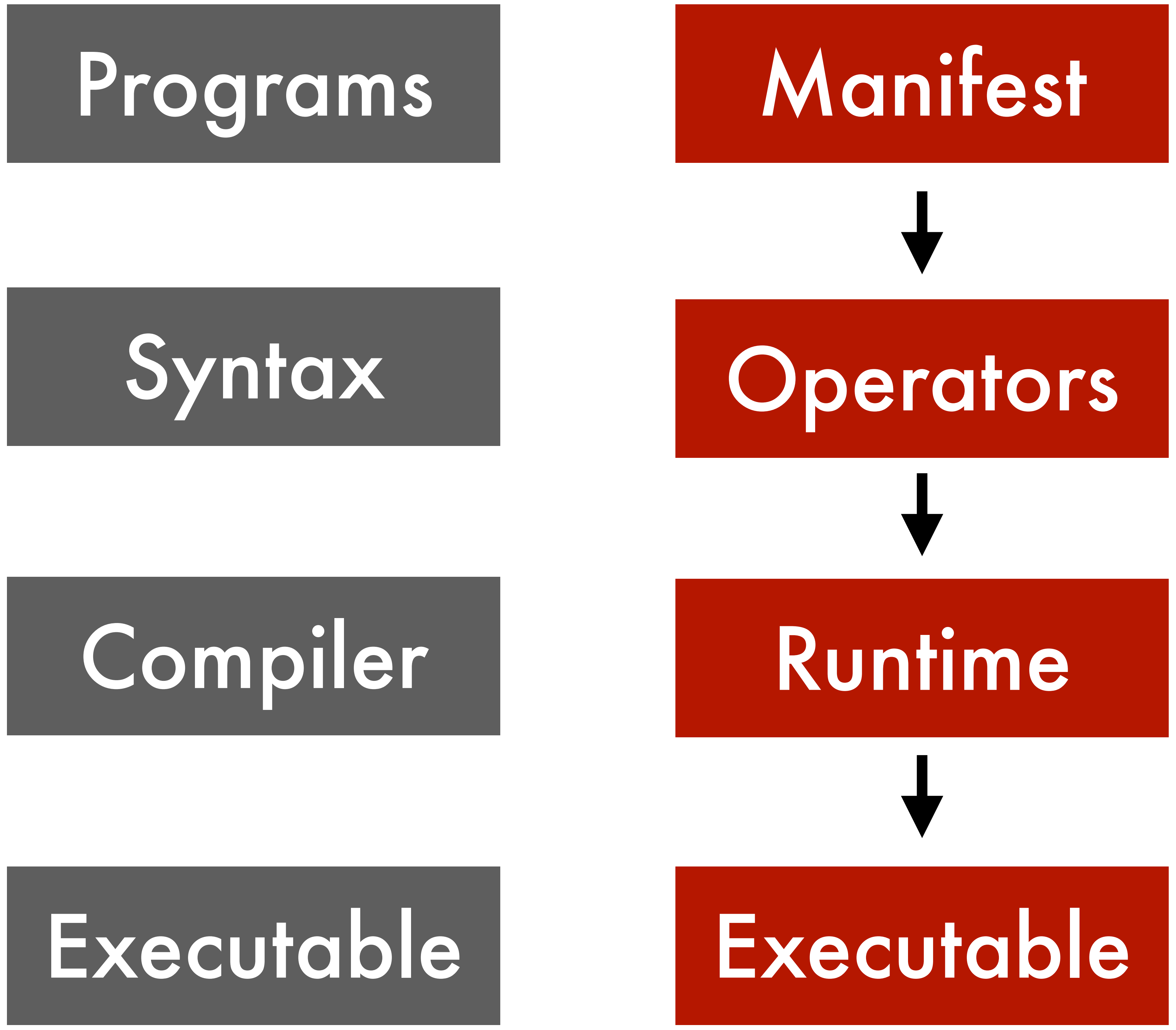


A text-based whitelist *manifest* (i.e., program representation)



```
@purpose: The app can access calendar events
which contains a zoom link.
ZoomCalendarIntegration{
  // operator [properties]
  inject[...] -> pull Calendar[...] ->
  filter [Zoom join link pattern] ->
  post [Zoom events]
}
```

MPF - Modular Privacy Flows



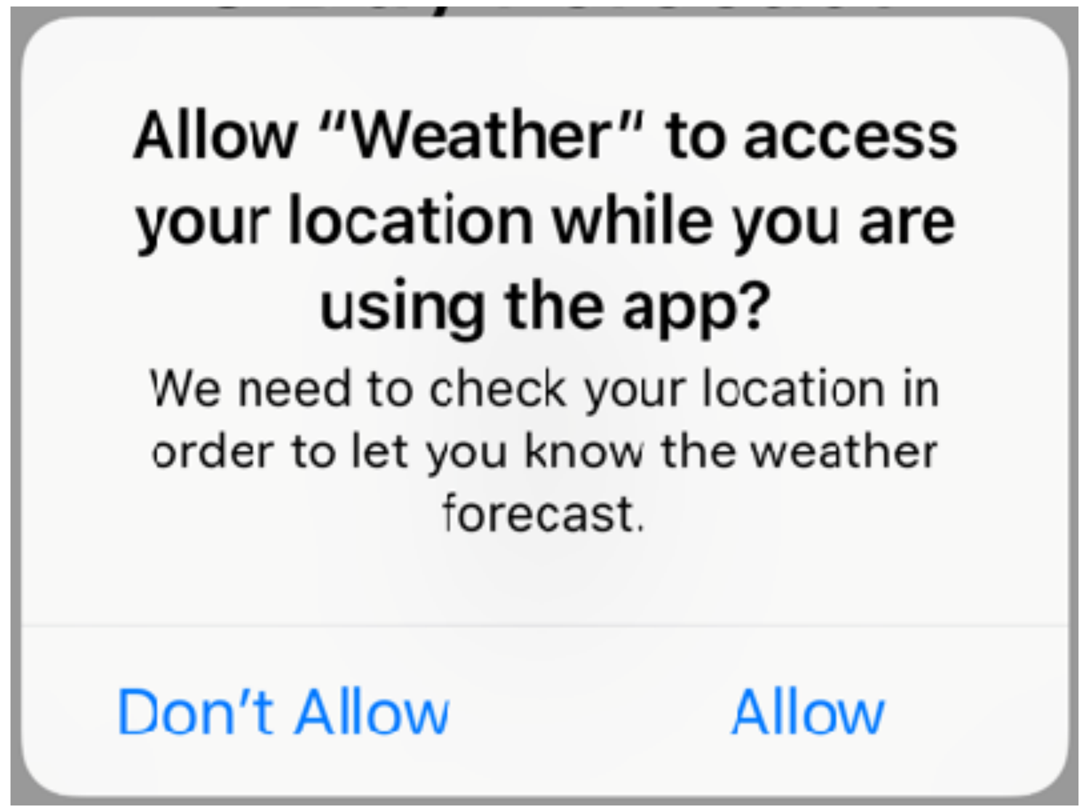
A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

MPF v.s. Binary permissions

```
<manifest ...>
  <uses-permission android:name="android.permission.
      ACCESS_COARSE_LOCATION" />
</manifest>
```

Android Permission Manifest

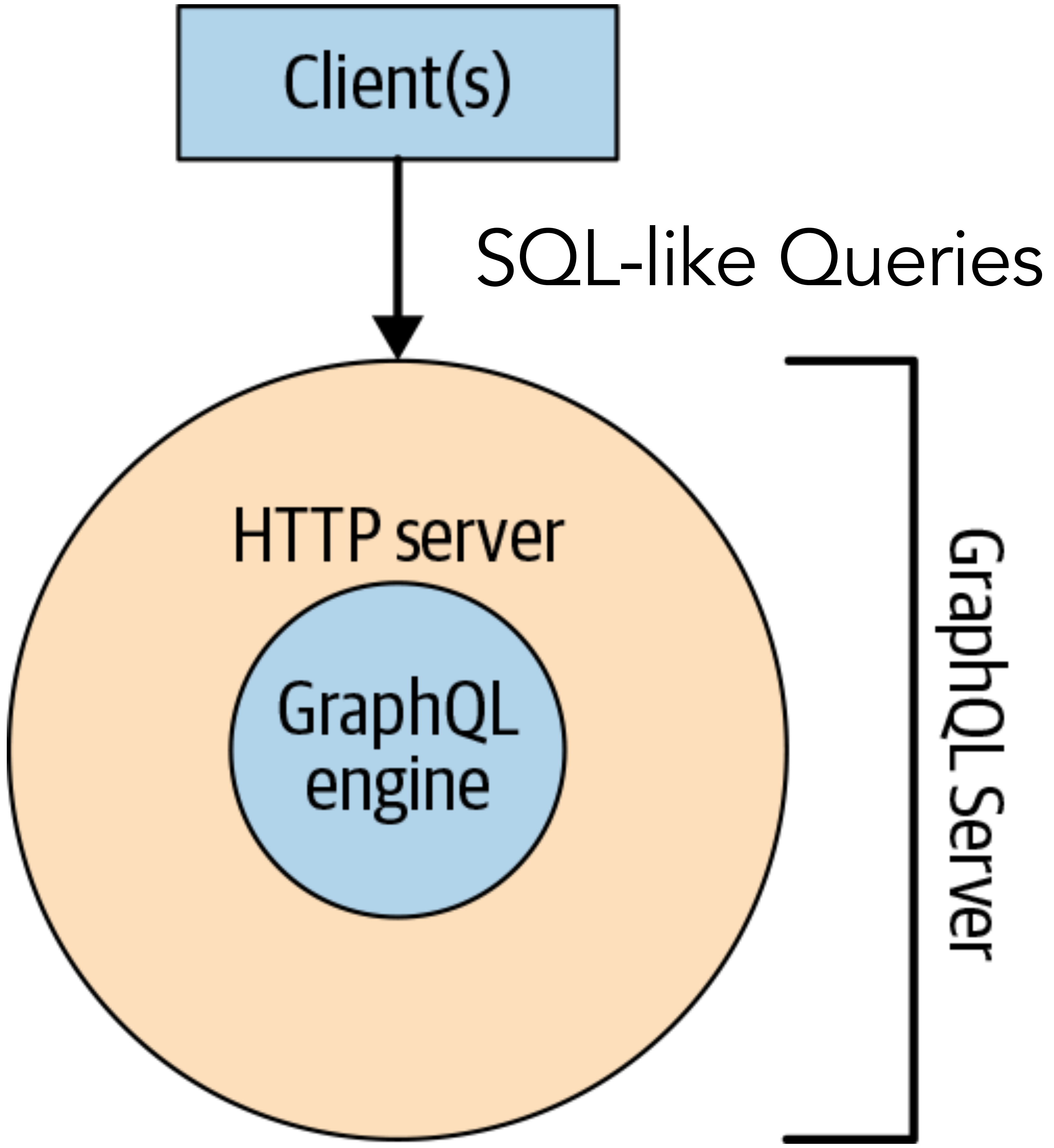

Allow "Weather" to access your location while you are using the app?

We need to check your location in order to let you know the weather forecast.

Popup window

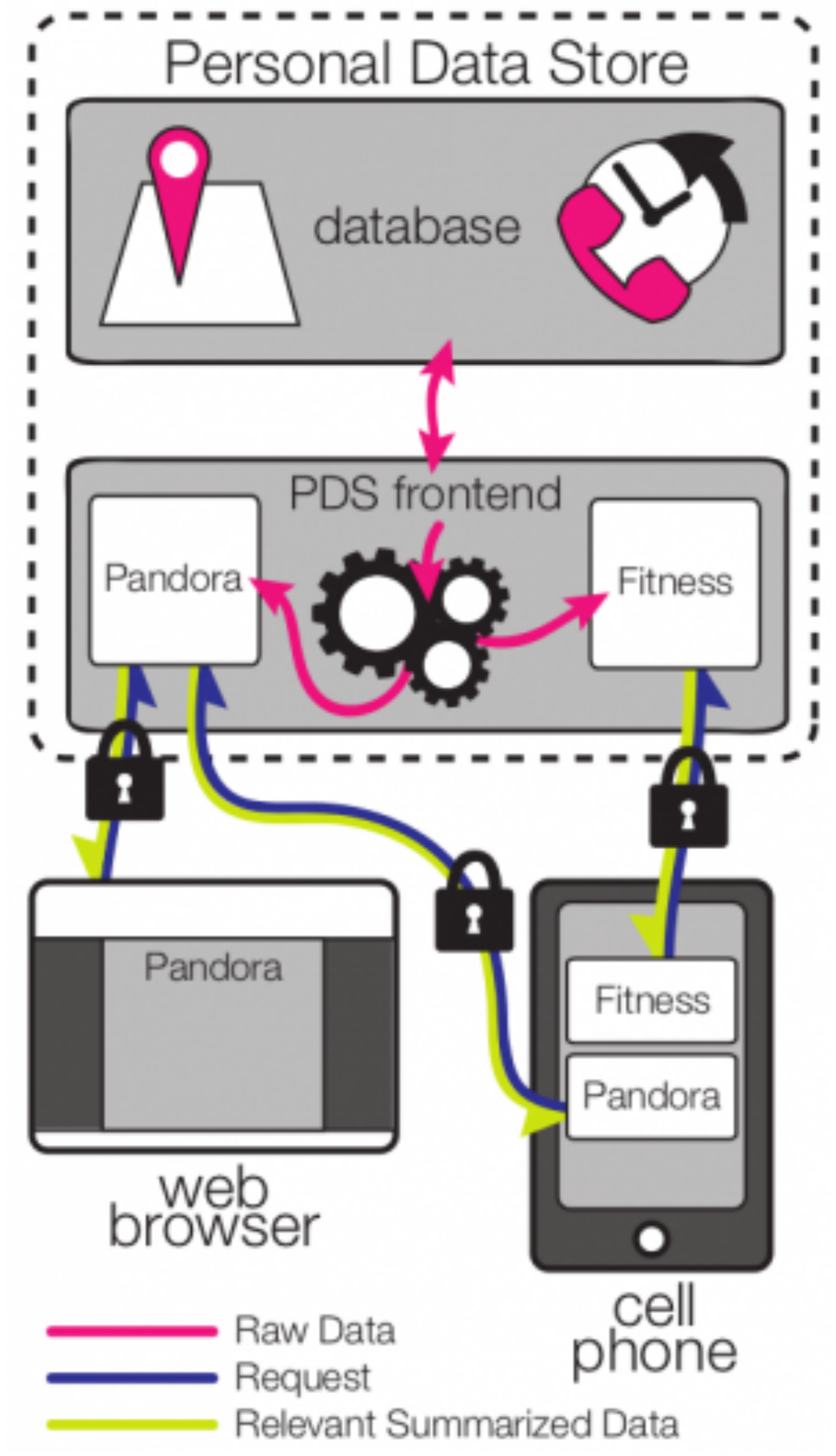
1. System implementation
2. API complexity
3. End-user management

MPF v.s. Database approaches (e.g., GraphQL)



1. Flexibility/Extendability
2. Auditability

MPF v.s. Remote Code Execution

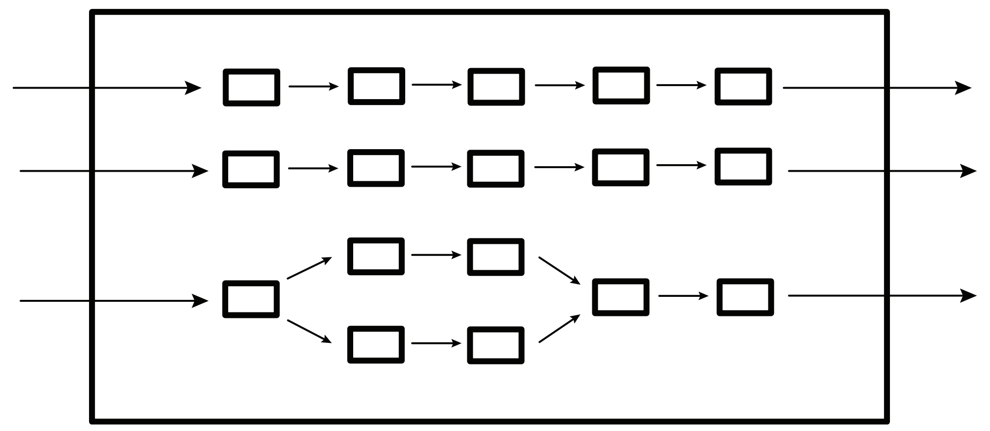


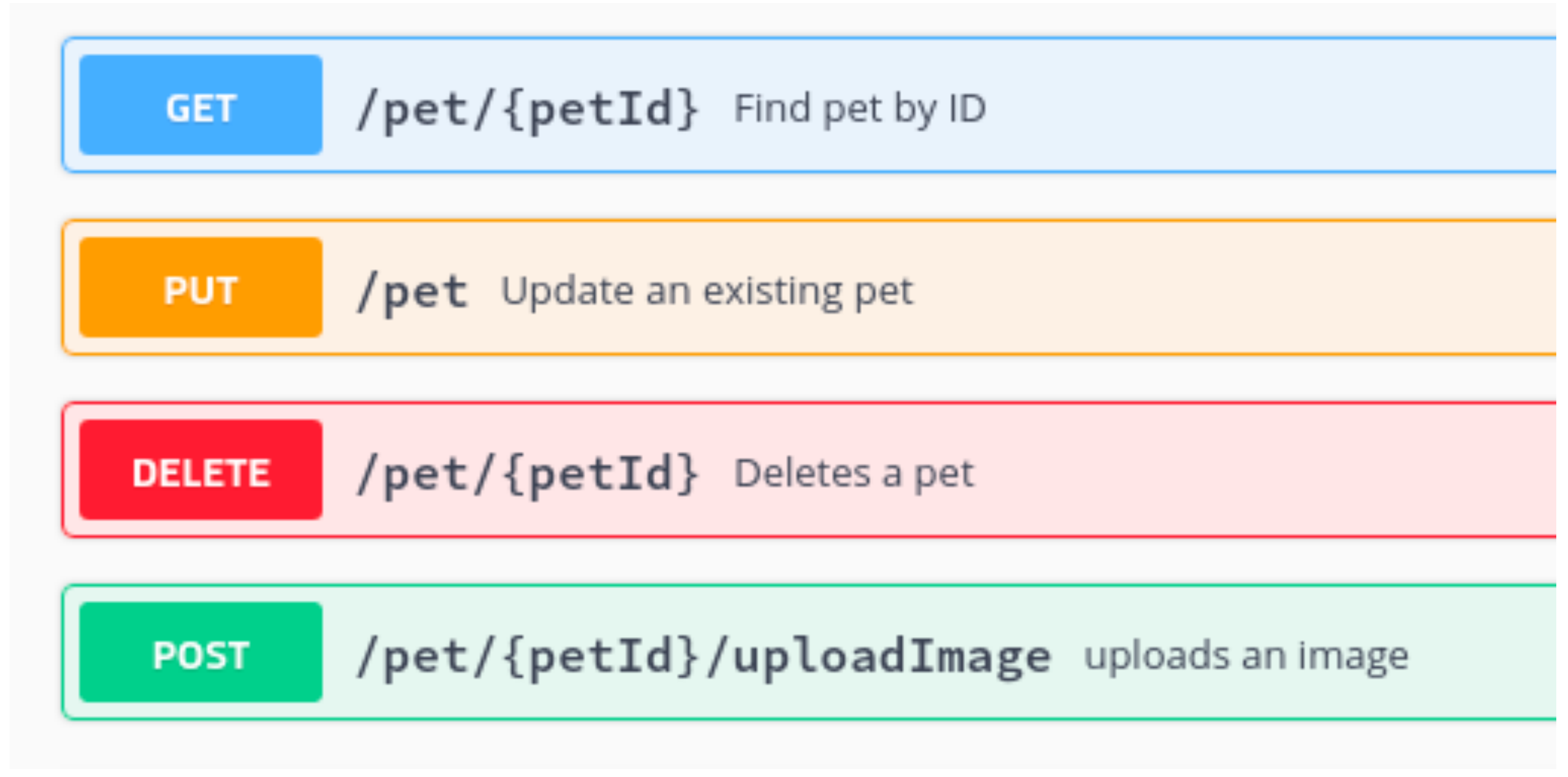
Developer-uploaded arbitrary code

1. Auditability
2. App development
3. Security

https://openpds.media.mit.edu/

Modular Privacy Flows today is REST in 2000.



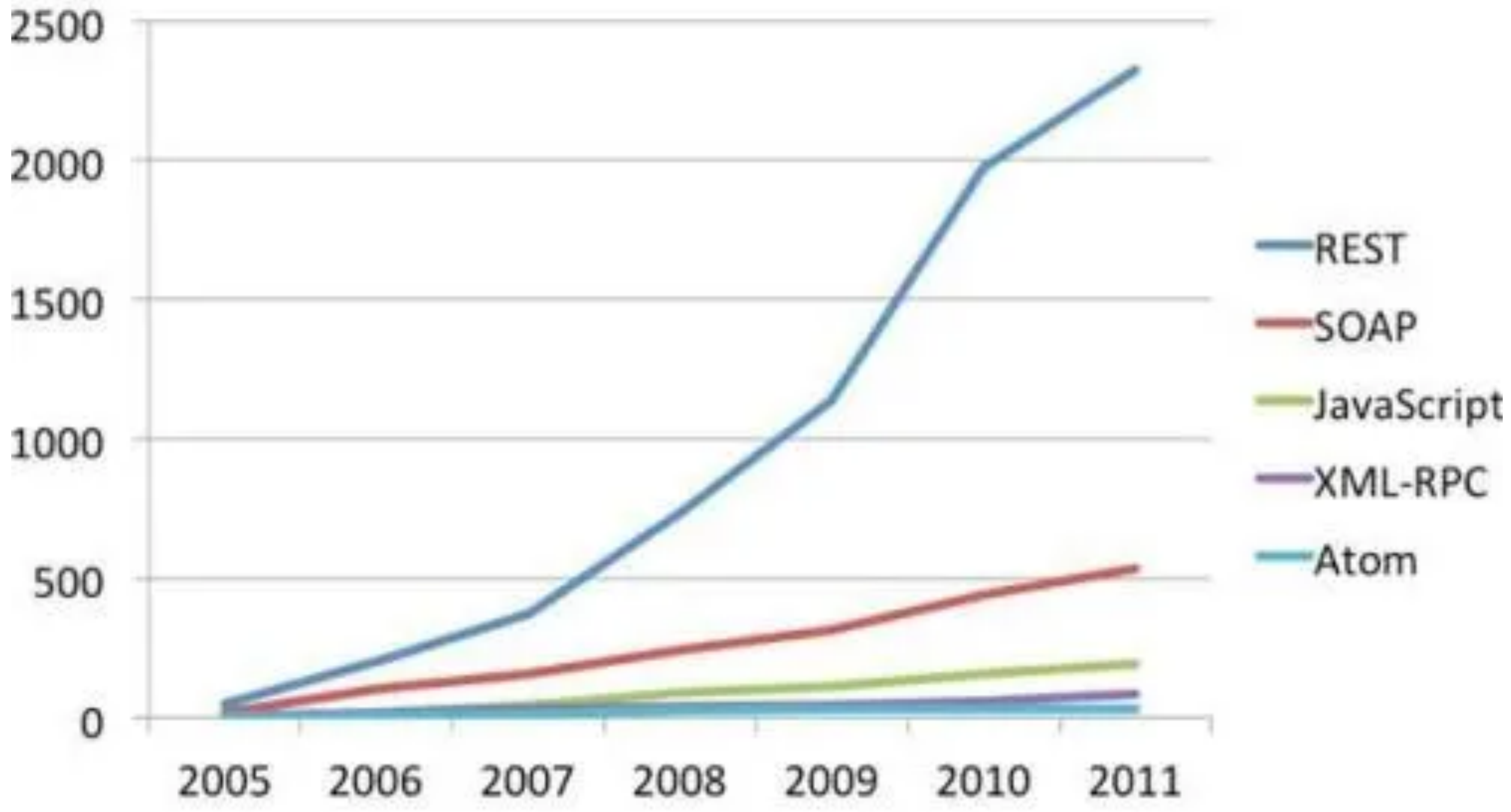


GET /pet/{petId} Find pet by ID

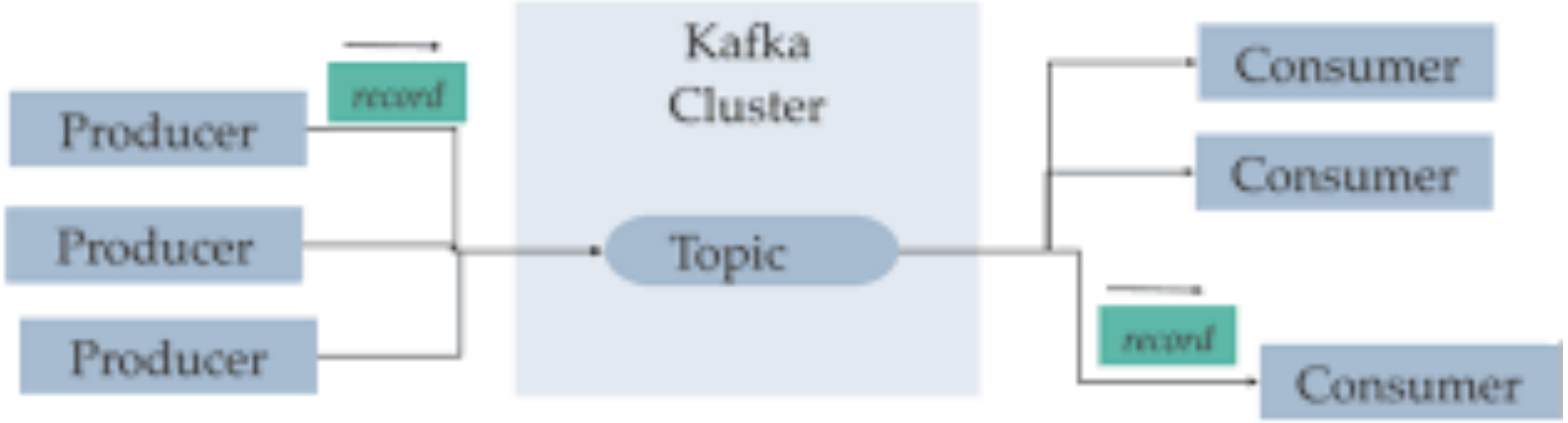
PUT /pet Update an existing pet

DELETE /pet/{petId} Deletes a pet

POST /pet/{petId}/uploadImage uploads an image

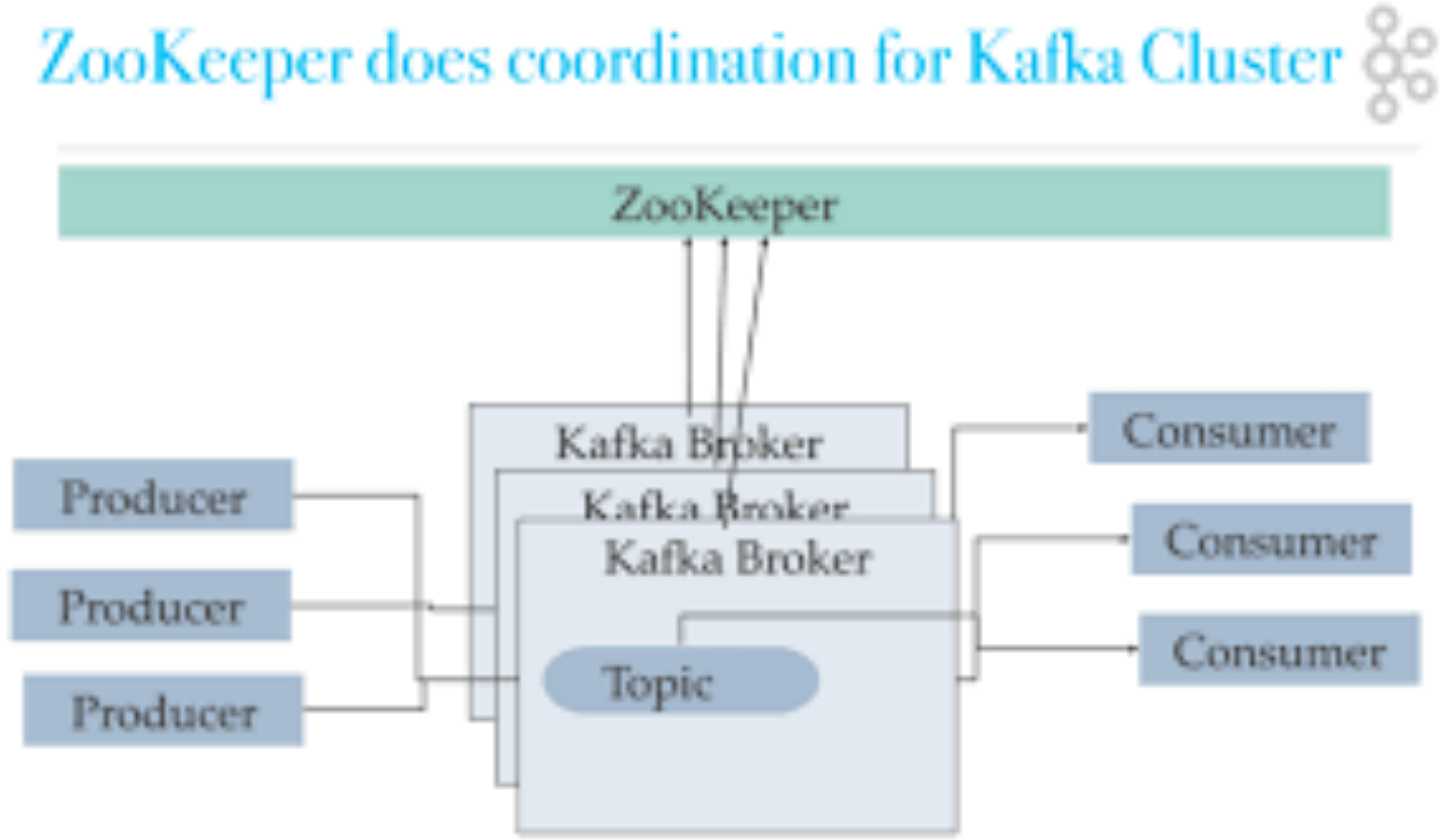


Message-Passing Dataflow (Kafka)



Message-Passing Dataflow (Kafka)

- Act as a buffer if the recipient is unavailable
- Automatically redeliver
- Senders do not need to know the IP address
- Broadcast
- Logically decoupling


ZooKeeper does coordination for Kafka Cluster
ZooKeeper
Kafka Broker
Kafka Broker
Kafka Broker
Producer
Producer
Producer
Topic
Consumer
Consumer
Consumer