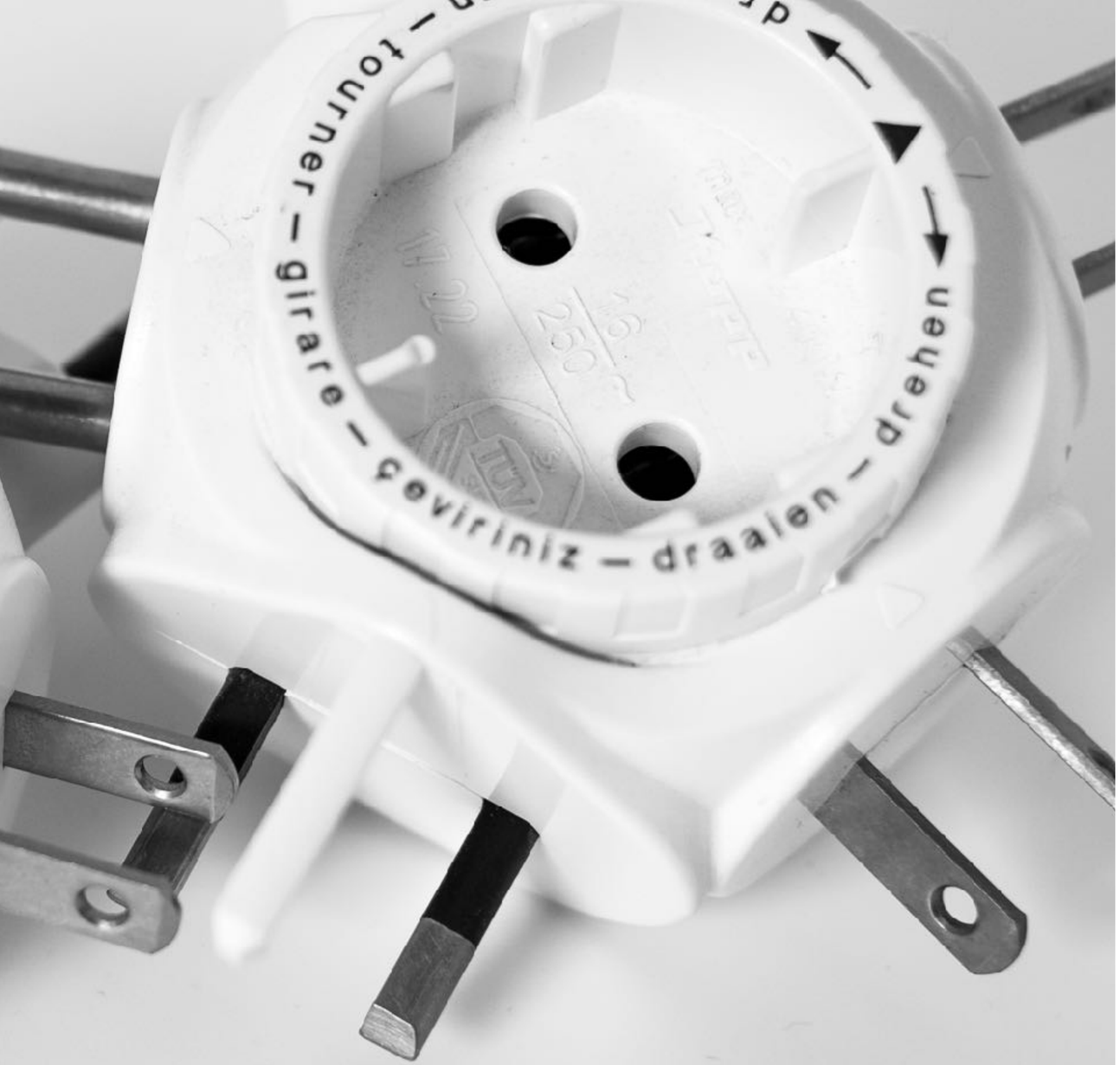
Ben Ramsey, PHP Tek, 21 May 2025

Return to REST

We’re talking about APIs.



APIs:

Fun for mash-ups circa 2008.

APIs:

Super-big business in 2025.

The rising shift from code-first to API-first

74% of respondents are API first in 2024, up from 66% in 2023.

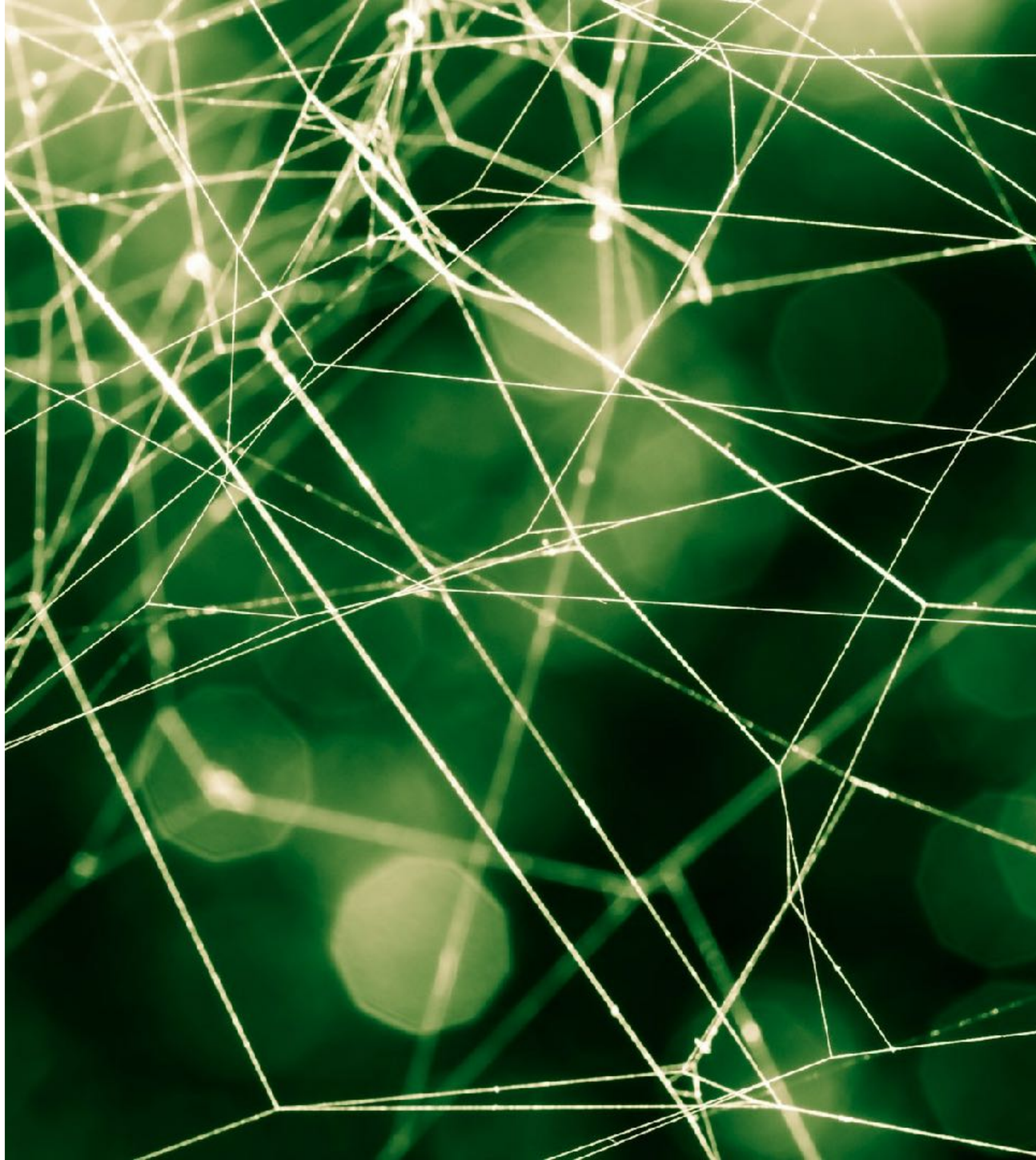
APIs front and center

APIs are no longer an afterthought but the foundation of development, with between 26 and 50 APIs powering the average application. The API-first paradigm prioritizes API design, streamlining integration and boosting efficiency across teams. To deliver top-tier APIs, engineering organizations are realizing they need more than just an API gateway. Proper tooling, enablement, and processes are essential to building and scaling high-quality APIs.

The world can’t work (together) without APIs.

HTTP APIs won.

We're talking about APIs, but



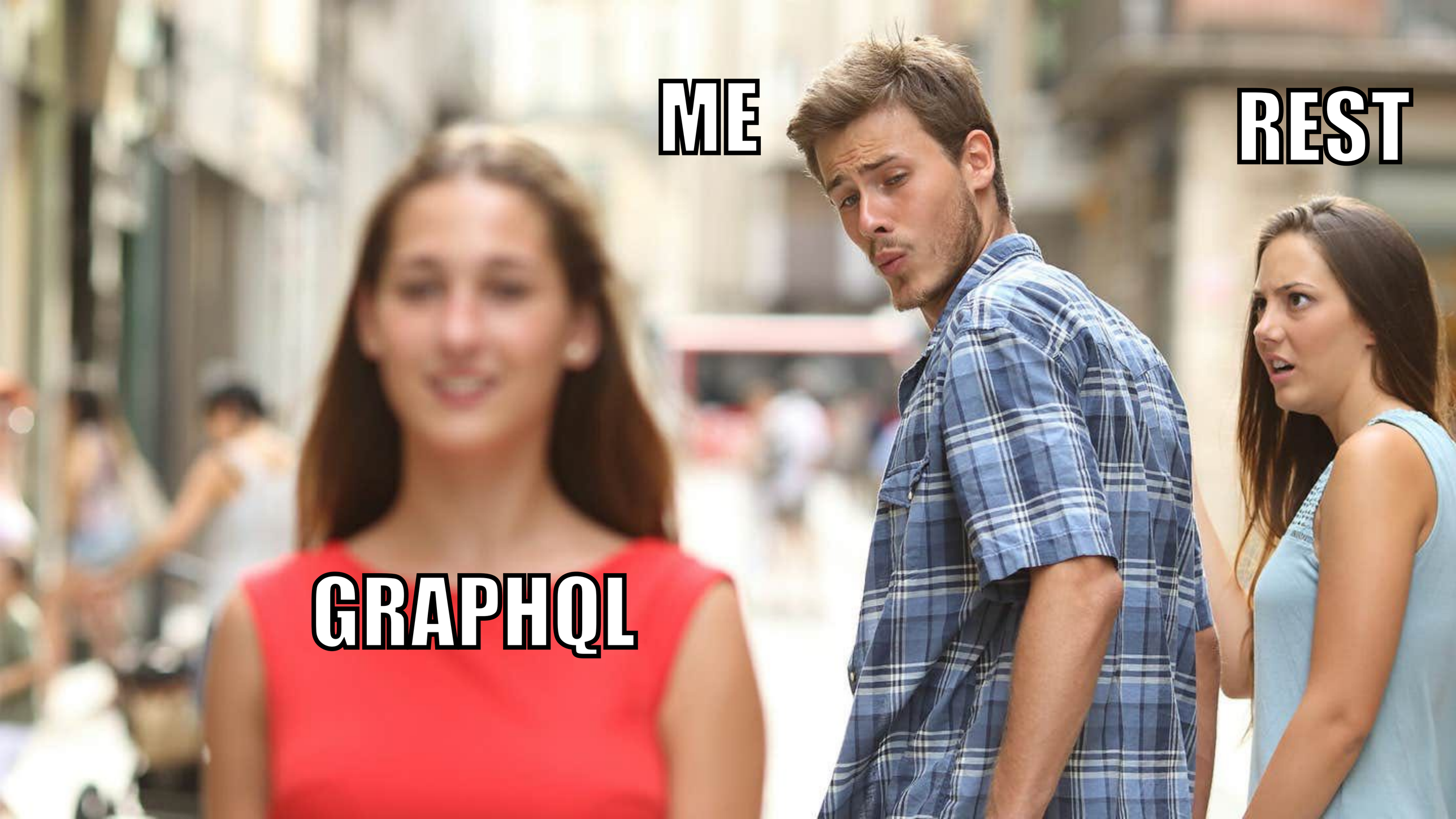
Specifically, we're talking about REST APIs.

Web 2.0

When REST was hot stuff.

What happened?

Why don’t we talk about REST anymore?


ME
REST
GRAPHQL

GraphQL is really popular.

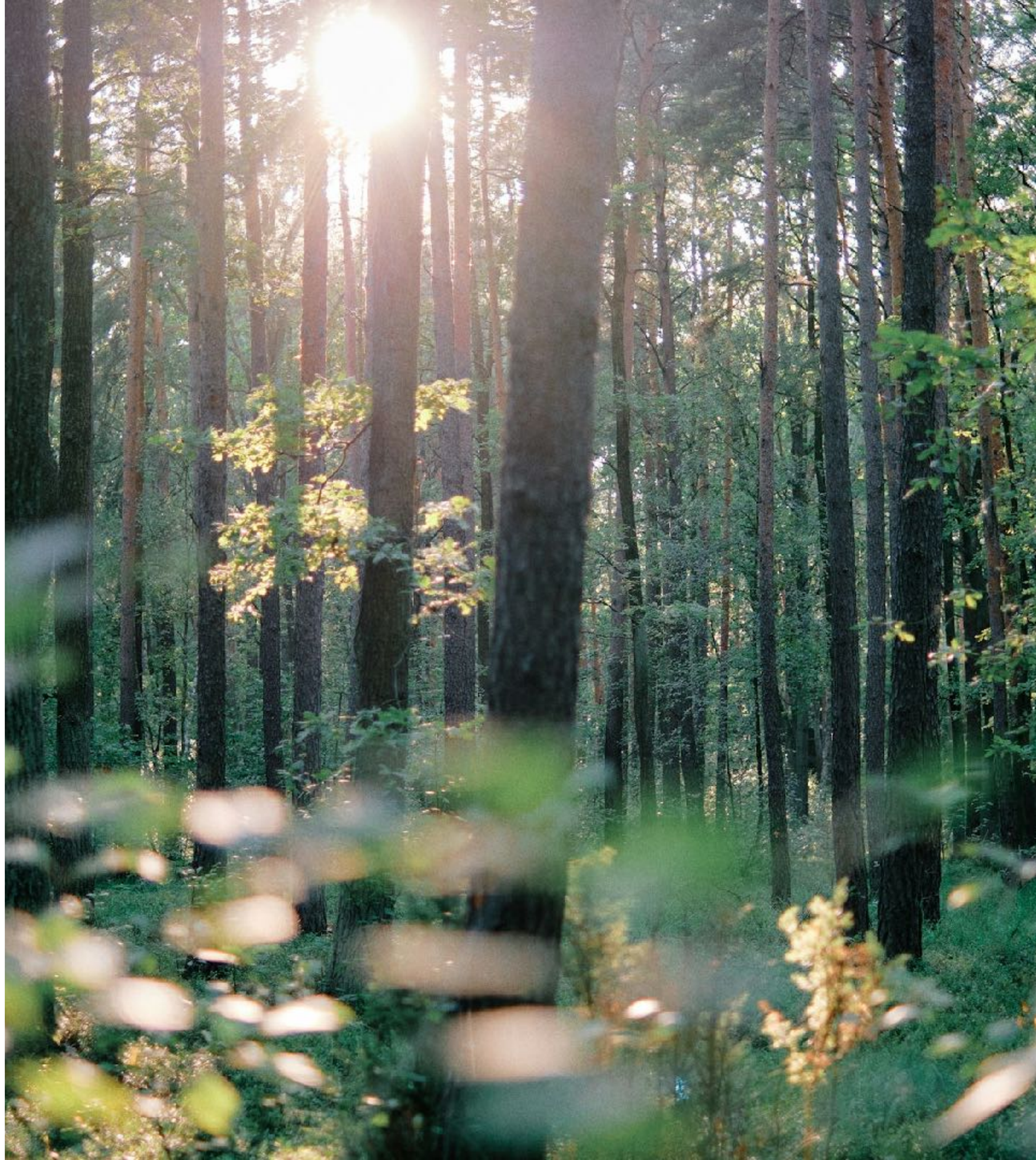
What does it have that REST doesn’t?

Representational State Transfer

Representational what now?

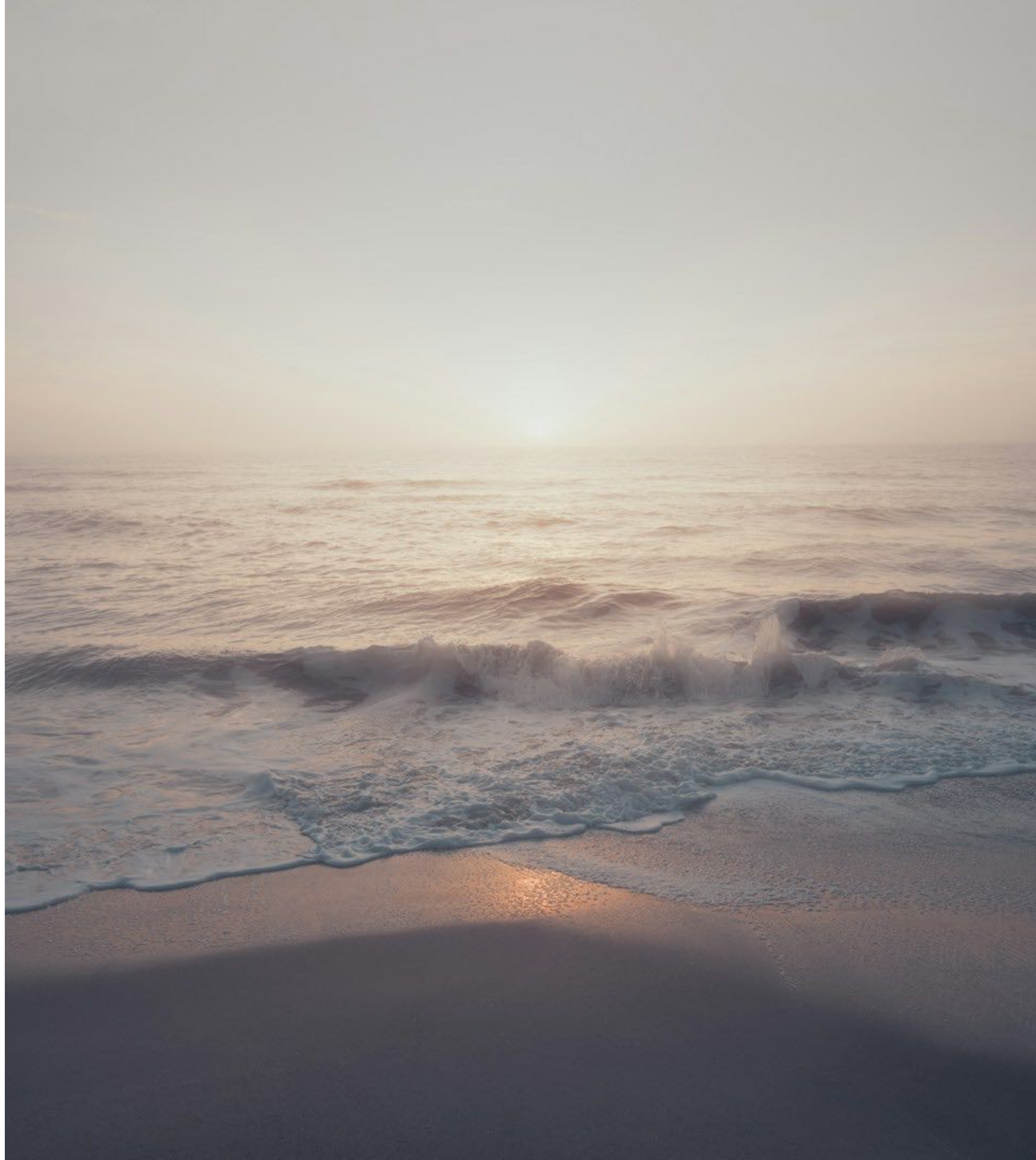
REST

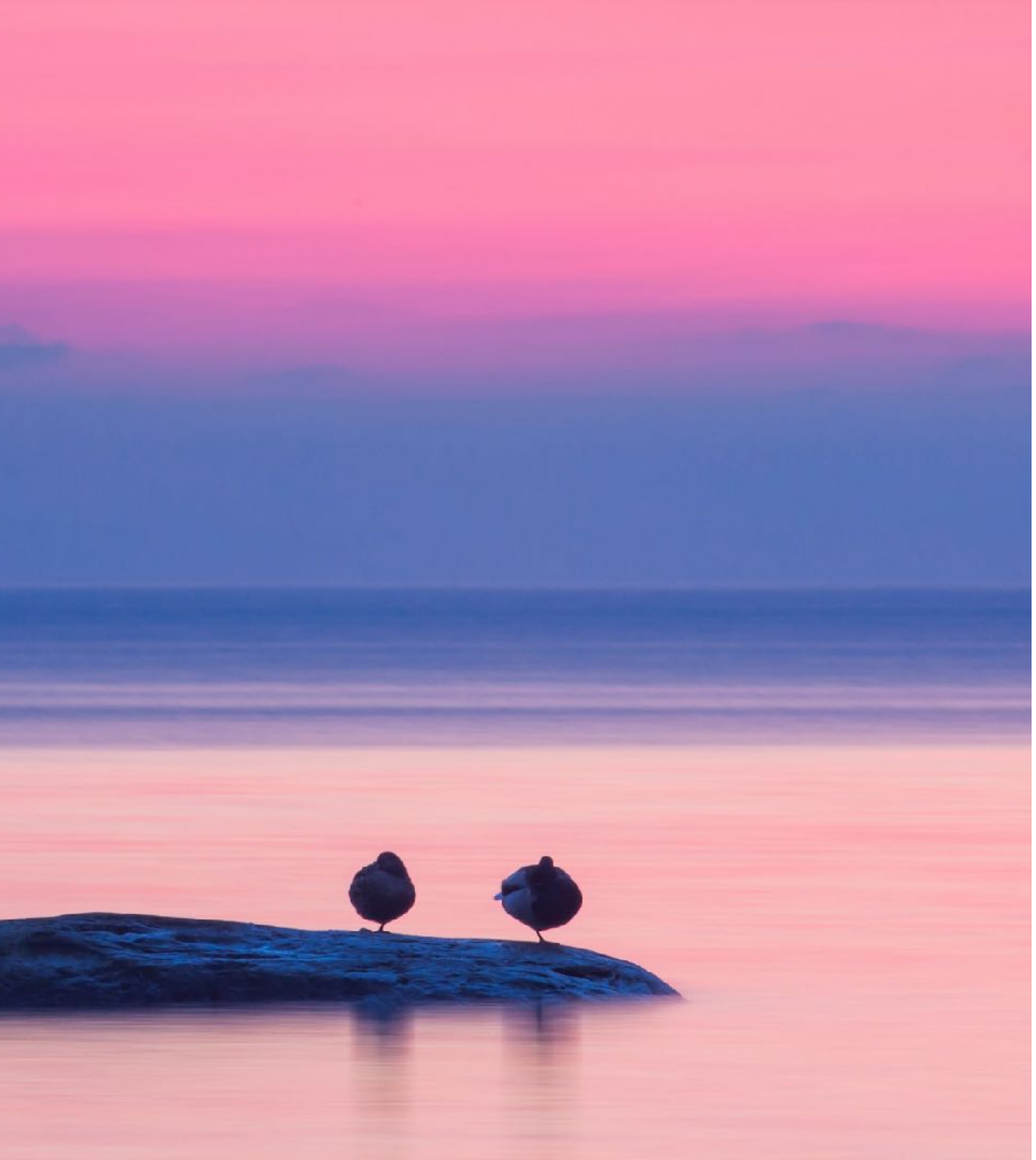
An architectural style that defines six constraints.



1. Client/server

2. Stateless

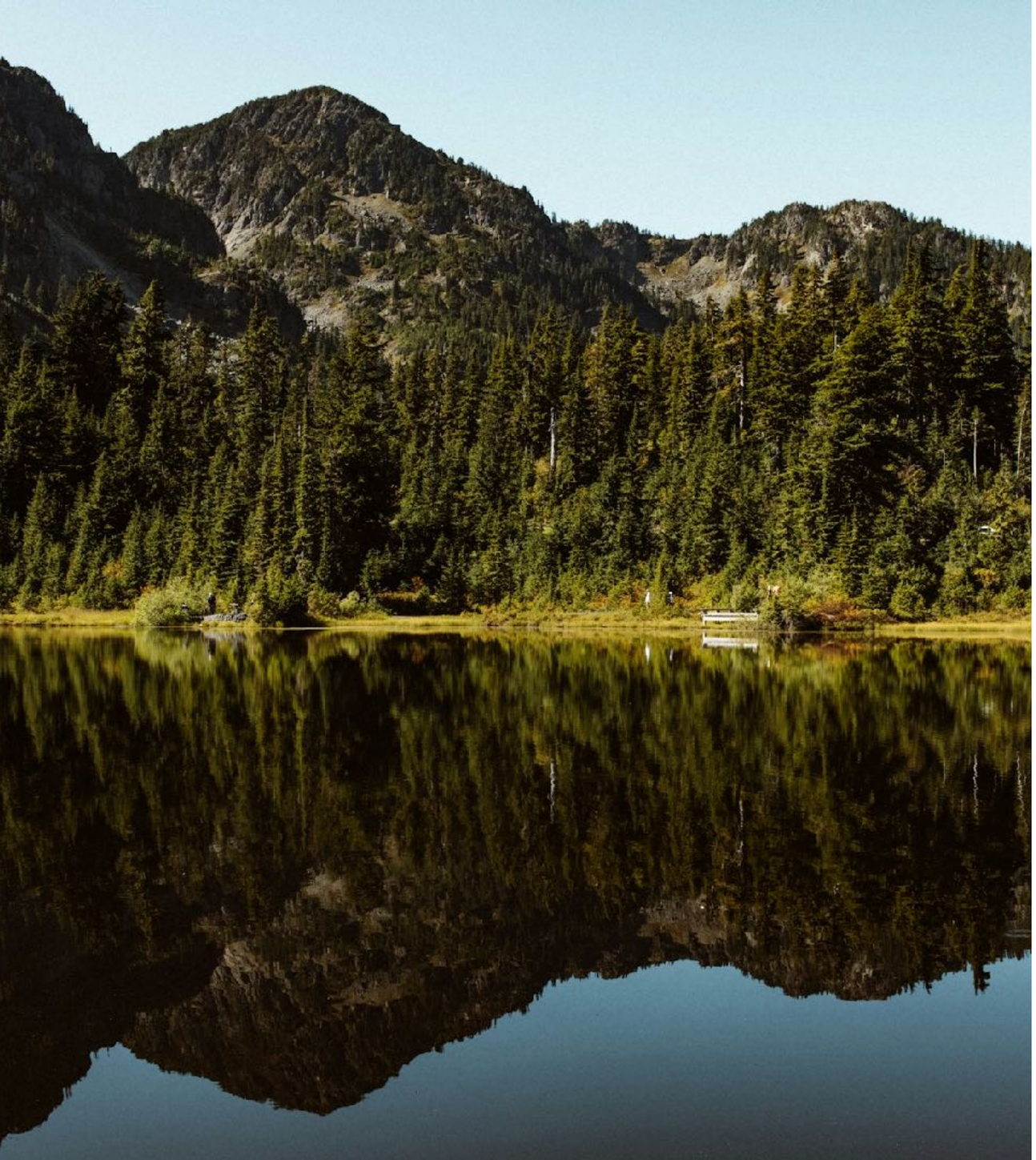




3. Cache

4. Uniform interface

- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as the engine of application state



5. Layered system

6. Code on demand

(Optional)

REST

Representational State Transfer

1. Client/server
2. Stateless
3. Cache
4. Uniform interface
5. Layered system
6. Code on demand

Cool, Ben.

So, now what?

What exactly does that get me?

REST

An architecture that is...

- Loosely coupled
- Simple
- Evolvable
- Portable
- Reliable
- Scalable

If it’s supposed to be so simple, why is it so hard?!

simple != easy

- We’re humans
- We’re lazy
- We need this yesterday.

- We don’t plan for the future.
- REST requires investments.
- We all suck.

GraphQL


REST
ME
GRAPHQL

GraphQL

GraphQL

- Reduces number of network requests and amount of data transferred
- From a company whose motto was once “move fast and break things”
- The primary trade-off: extreme tight coupling


ME
REST
GRAPHQL

REST story time...

REST

Level 0: The Swamp of POX

Request

```
POST /rpc.php HTTP/1.1
Host: api.example.com
Content-Length: 125

{
    "method": "saveCampaignData",
    "params": {
        "email": "foo@example.com",
        "postalCode": "12345"
    }
}
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 16:31:57 GMT
Content-Length: 23

{
    "status": "success"
}
```

Level 1: Resources

Level 0: The Swamp of POX

Request

```
GET /record/create?
    campaignID=o0zJeEcJ
    &email=foo@example.com
    &postalCode=12345 HTTP/1.1
Host: api.example.com
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 16:31:57 GMT
Content-Length: 23

{
    "status": "success"
}
```

Request

```
GET /campaign/o0zJeEcJ HTTP/1.1
Host: api.example.com
```

Response

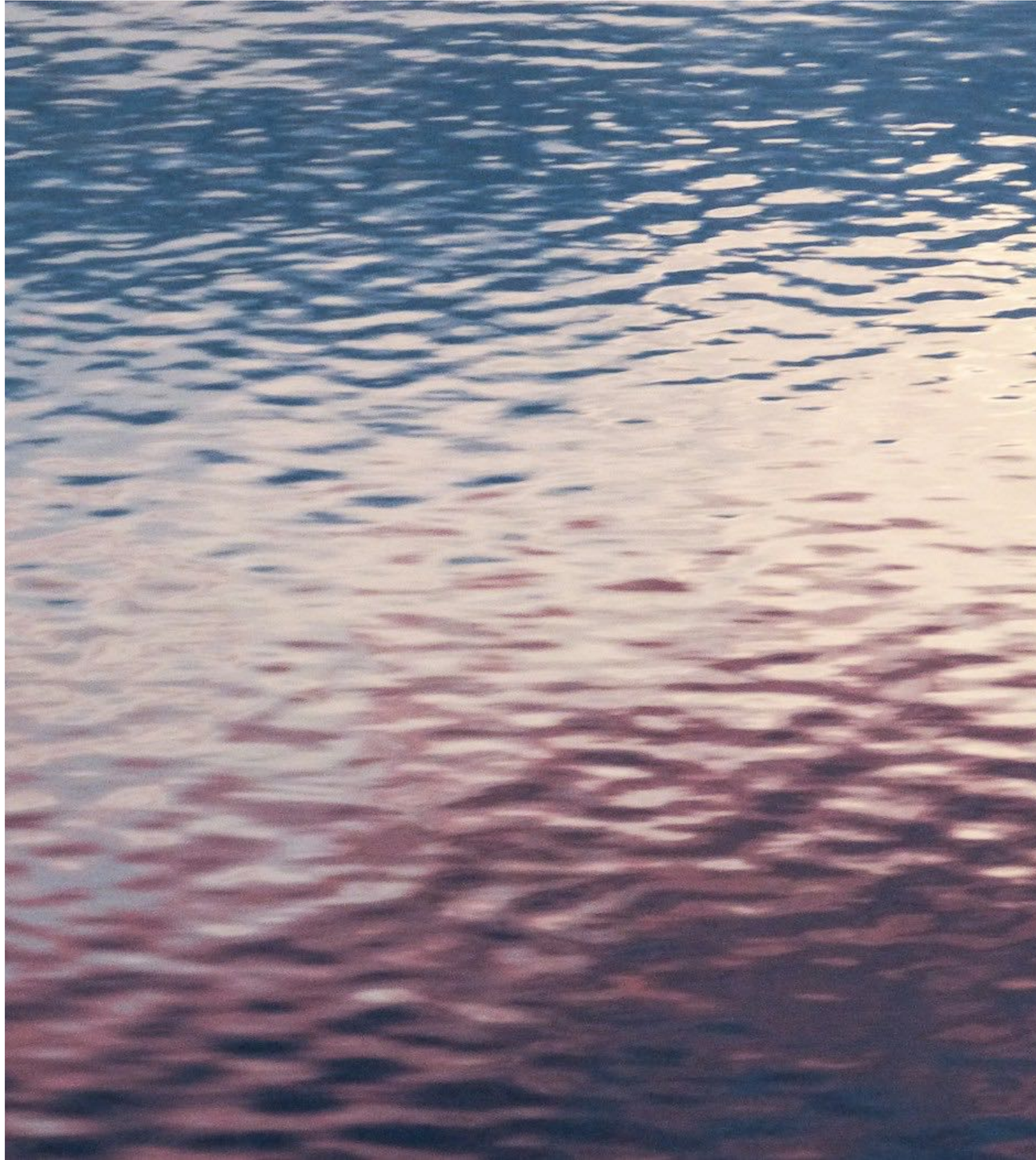
```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 16:43:23 GMT
Content-Length: 120

{
    "records": [{
        "email": "foo@example.com",
        "postalCode": "12345"
    }]
}
```

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

- Use proper HTTP semantics for resources
- POST to create a new record, GET to fetch a campaign
- What else can we do with that campaign resource?
  - Create new campaigns with POST
  - Update campaigns with PUT
- What about the record?
  - Fetch an individual record with GET
  - Use PATCH to partially update a record



Request

```
POST /campaign HTTP/1.1
Host: api.example.com
Content-Length: 37

{
    "name": "My Awesome Campaign"
}
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:14:11 GMT
Content-Length: 67

{
    "campaignId": "o0zJeEcJ",
    "name": "My Awesome Campaign"
}
```

Request

```
PUT /campaign/o0zJeEcJ HTTP/1.1
Host: api.example.com
Content-Length: 143

{
    "campaignId": "o0zJeEcJ",
    "name": "Totally Rad Campaign",
    "startDate":
        "2015-09-06T18:12:47+00:00",
    "endDate":
        "2015-10-06T18:13:06+00:00"
}
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:22:04 GMT
Content-Length: 173

{
    "campaignId": "o0zJeEcJ",
    "name": "Totally Rad Campaign",
    "startDate":
        "2015-09-06T18:12:47+00:00",
    "endDate":
        "2015-10-06T18:13:06+00:00"
}
```

Request

```
POST /record HTTP/1.1
Host: api.example.com
Content-Length: 91

{
    "campaignID": "o0zJeEcJ",
    "email": "foo@example.com",
    "postalCode": "12345"
}
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:28:45 GMT
Content-Length: 119

{
    "recordId": "5s7ytJlT",
    "campaignID": "o0zJeEcJ",
    "email": "foo@example.com",
    "postalCode": "12345"
}
```

Request

```
GET /record/5s7ytJlT HTTP/1.1
Host: api.example.com
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:29:31 GMT
Content-Length: 119

{
    "recordId": "5s7ytJlT",
    "campaignID": "o0zJeEcJ",
    "email": "foo@example.com",
    "postalCode": "12345"
}
```

Request

```
PATCH /record/5s7ytJlT HTTP/1.1
Host: api.example.com
Content-Length: 38

{
    "email": "samwise@example.com"
}
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:28:45 GMT
Content-Length: 123

{
    "recordId": "5s7ytJlT",
    "campaignID": "o0zJeEcJ",
    "email": "samwise@example.com",
    "postalCode": "12345"
}
```

Level 3: Hypermedia

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

Request

```
POST /campaign HTTP/1.1
Host: api.example.com
Accept: application/hal+json
Content-Type: application/json
Content-Length: 37

{
    "name": "My Awesome Campaign"
}
```

Response

```
HTTP/1.1 201 Created
Date: Sat, 10 Oct 2024 17:45:27 GMT
Content-Type: application/hal+json
Content-Length: 838
Location: /campaign/o0zJeEcJ
```

```
{
    "_links": {
        "self": { "href": "https://api.example.com/campaign/o0zJeEcJ" },
        "curies": [{
            "name": "ex",
            "href": "https://api.example.com/docs/rels/{rel}",
            "templated": true
        }],
        "ex:campaigns": { "href": "https://api.example.com/campaign/" },
        "ex:records": { "href": "https://api.example.com/record/" }
    },
    "name": "My Awesome Campaign",
    "_embedded": {
        "ex:record": [{
            "_links": {
                "self": {
                    "href": "https://api.example.com/record/5s7ytJlT"
                },
                "ex:campaign": {
                    "href": "https://api.example.com/campaign/o0zJeEcJ"
                }
            },
            "email": "foo@example.com",
            "postalCode": "12345"
        }]
    }
}
```


Live,
Laugh,
Rest

HATEOAS

Hypermedia As The Engine Of Application State

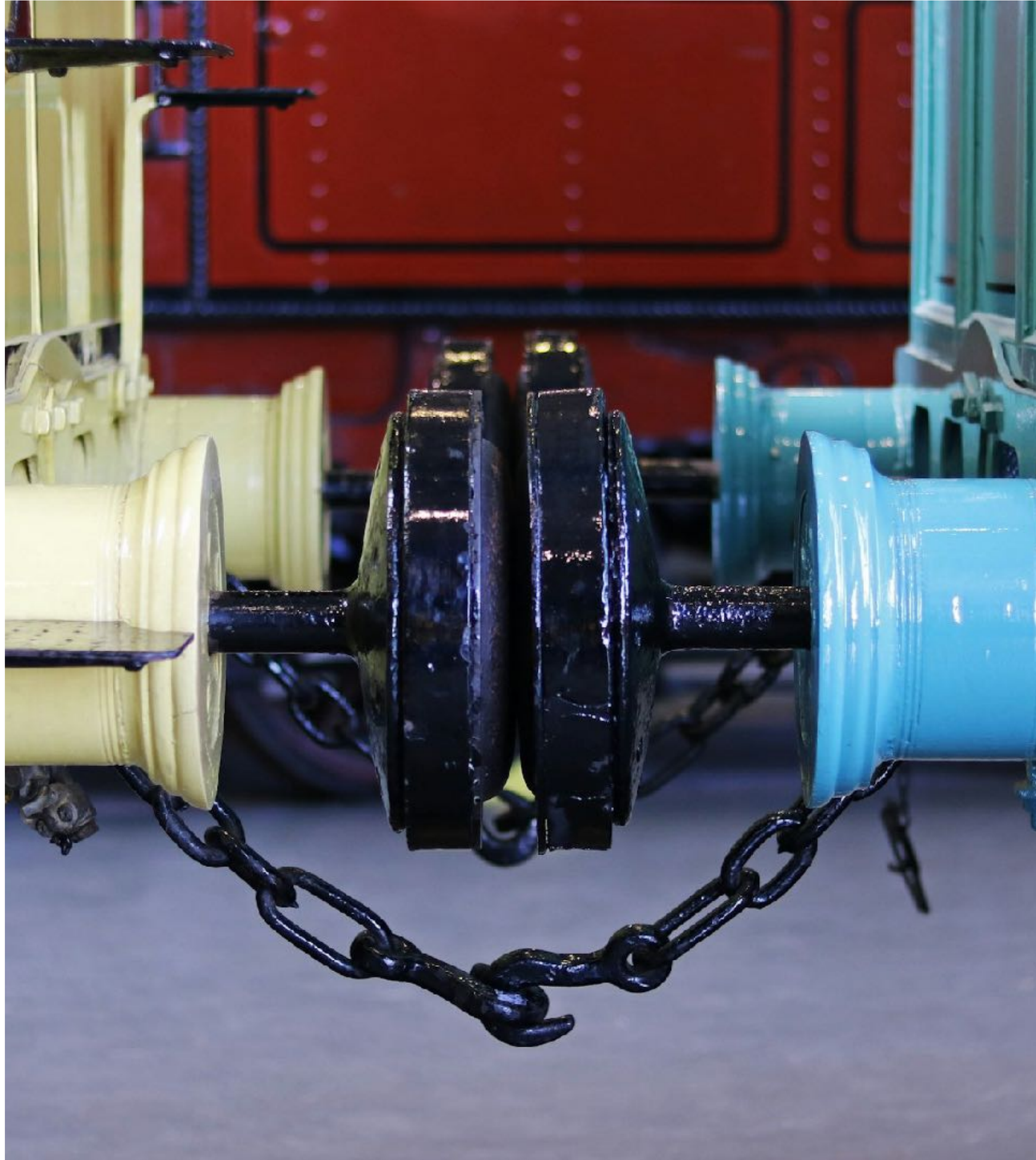
Hypermedia

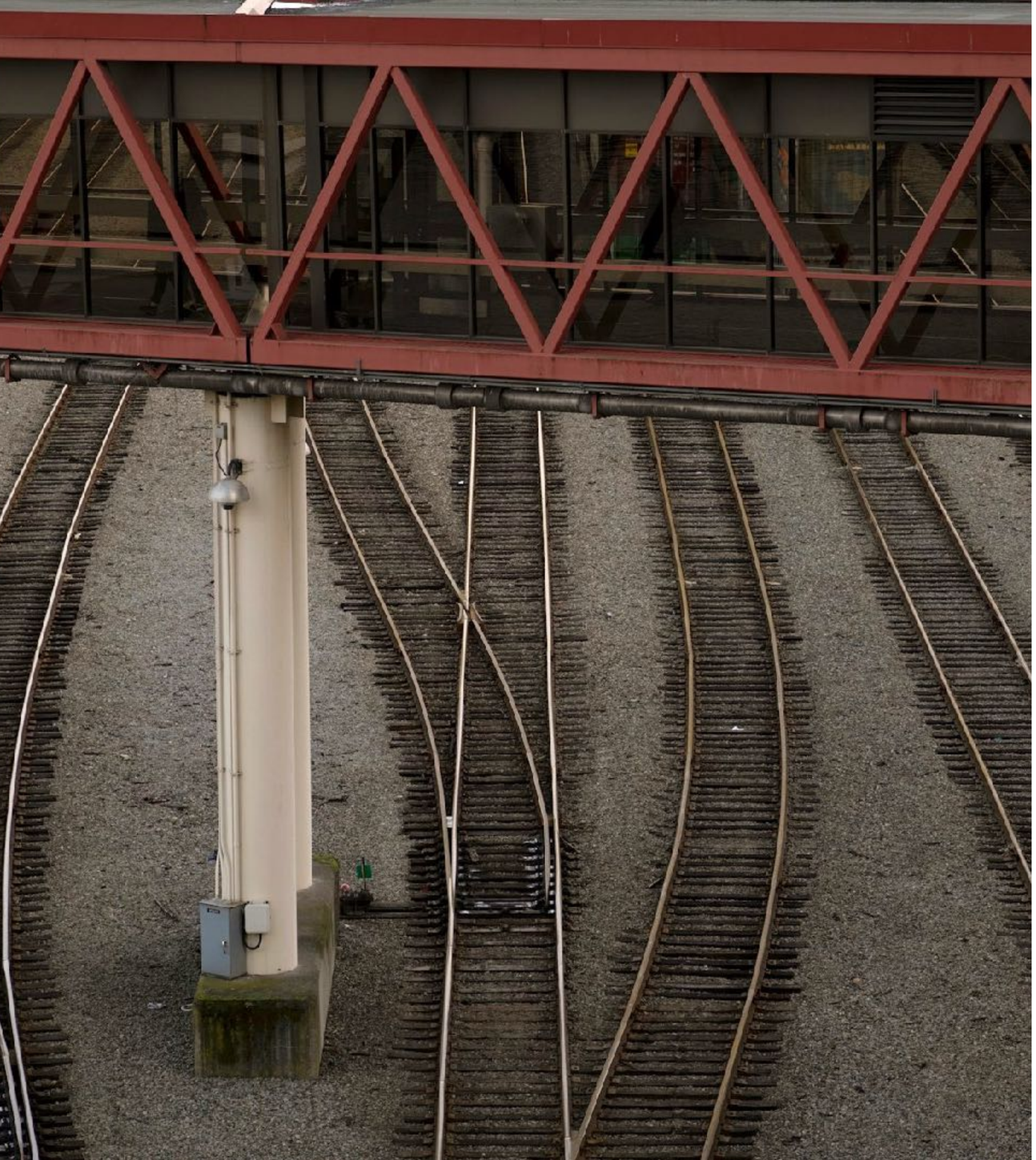
In a nutshell

- Use links to provide relationships between resources
- Use media types to describe how to process a representation

Hypermedia
Benefits

- Clients discover locations and operations
- Link relations express possible options
- URLs can change
- Workflow is abstracted
- Media type can be versioned
- Clients do not break if the implementation is updated!





Hypermedia

It answers these questions

- How does a client know what to do with resources?
- How do you go to the “next” operation?
- What are the URLs for creating subordinate resources?
- Where is the contract for the service?

Make the most of HTTP

- Use HTTP methods for the purpose they were created:
  - GET, POST, PUT, PATCH, DELETE, BREW…
- Uniquely identify each resource with a URI.
- Be self-documenting through headers (e.g., Accept, Allow, Link, etc.)
- Use hypermedia (links)!
- Put thought into your media types.

Request

```
OPTIONS /record/5s7ytJlT HTTP/1.1
Host: api.example.com
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:28:45 GMT
Accept: application/hal+json
Allow: GET, PUT, PATCH, DELETE, OPTIONS, HEAD
Link: </record/5s7ytJlT>; rel="self"
Link: </record>; rel="collection"
Link: </campaign/o0zJeEcJ>; rel="https://example.com/rel/campaign"
```

Request

```
OPTIONS /record HTTP/1.1
Host: api.example.com
```

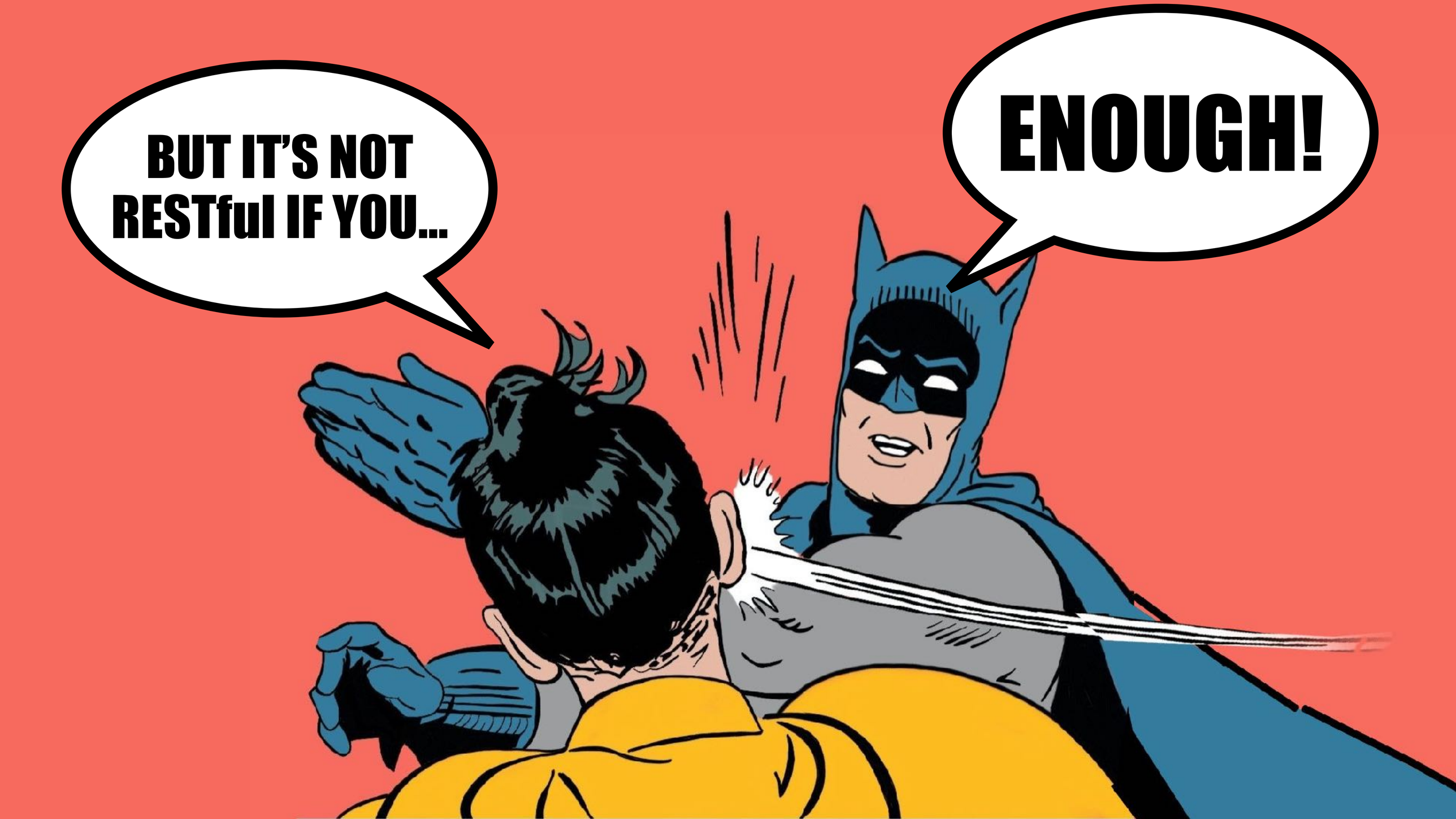
Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:28:45 GMT
Accept: application/hal+json
Allow: GET, POST, OPTIONS, HEAD
Link: </record>; rel="self"
Link: </record>; rel="start"
Link: </record?page=2>; rel="next"
Link: </record?page=1234>; rel="last"
```

Our API is self-documenting!

> “If you think you have control over the system or aren’t interested in evolvability, don’t waste your time arguing about REST.”

Roy T. Fielding


BUT IT'S NOT
RESTful IF YOU...
ENOUGH!

Evolvability is what we want.

- If we change the URLs, will we break clients?
- If we add or remove properties, will we break clients?
- If we change the input values we accept, will we break clients?

Versioning in APIs exists because we're afraid of breaking clients.

https://api.example.com/v2/campaign

Roy T. Fielding
@fielding

The reason to make a real REST API is to get evolvability ... a "v1" is a middle finger to your API customers, indicating RPC/HTTP (not REST)

5:33 PM · Sep 8, 2013

Hypermedia resolves the versioning problem.

- I can change URLs and nothing breaks.
- I can add properties and nothing breaks.
- If I remove properties or require new inputs, I can use content-negotiation to version the API:

  ```
  application/vnd.example+json; version=2
  ```

- Everybody’s happy. (Maybe?)

Request

```
OPTIONS /record/5s7ytJlT HTTP/1.1
Host: api.example.com
```

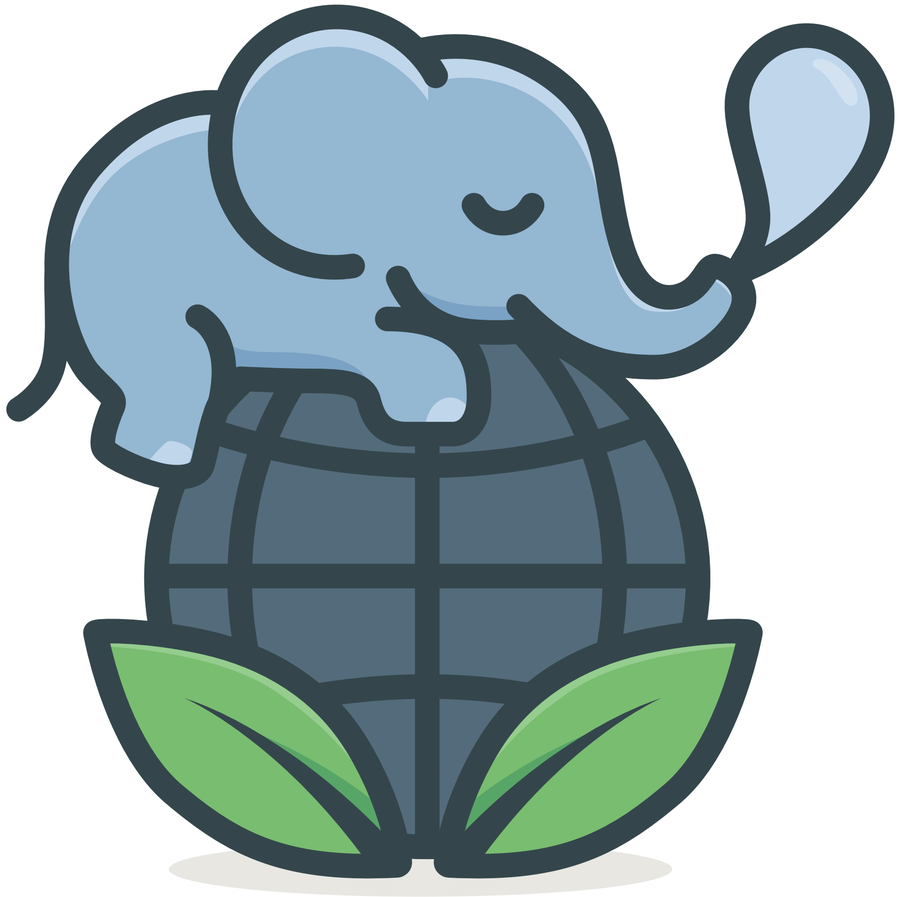
Response

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 2024 17:28:45 GMT
Accept: application/vnd.example+json; q=0.8,
        application/vnd.example+json; version=2; q=0.9,
        application/vnd.example+json; version=3
Allow: GET, PUT, PATCH, DELETE, OPTIONS, HEAD
Link: </record/5s7ytJlT>; rel="self"
Link: </record>; rel="collection"
Link: </campaign/o0zJeEcJ>; rel="https://example.com/rel/campaign"
```

REST Certain

Testing DSL for REST APIs

- Express your API tests in a natural and intuitive way
- Facilitates and encourages TDD and BDD for APIs
- Supports validation with JSON Schema
- rest-certain/rest-certain
- Ports of REST Assured from Java to PHP



Thank you!

Keep in touch

- ben.ramsey.dev
- phpc.social/@ramsey
- github.com/ramsey
- speakerdeck.com/ramsey
- www.linkedin.com/in/benramsey
- ben@ramsey.dev

bram.se/phptek-rest

Resources

- IANA HTTP Method Registry
- IANA HTTP Field Name Registry
- IANA Link Relations Registry
- IANA Media Types Registry
- RFC 9110: HTTP Semantics
- RFC 5789: PATCH Method for HTTP
- RFC 8288: Web Linking
- Hypermedia Application Language (HAL)
- Architectural Styles and the Design of Network-based Software Architectures, Roy T. Fielding

Photo Credits

- Clear Inner Vision. "Restful Summer." CC BY-NC-ND 2.0.
- Wouter de Bruijn. "Day 90." CC BY-NC-SA 2.0.
- George Rosema. "Spider Web in morning dew." Unsplash License.
- Shannon Potter. Untitled. Unsplash License.
- Ibrahim Fareed. Untitled. Unsplash License.
- Alice Olivier. "Popular Web 2.0 Logos." CC BY 3.0.
- Carlos Torres. "From the plane." Unsplash License.
- Ante Hamersmit. "Man sitting on a wooden bridge." Unsplash License.
- Jacek Smoter. "Forest in central Poland." Unsplash License.
- Artem Kniaz. "picturesque colorful autumn landscape….” Unsplash License.
- Todd Trapani. "Up early watching the sun rise along the beach." Unsplash License.
- Kaupo Kalda. "Sleepy sunset." Unsplash License.
- Meina Yin. Untitled. Unsplash License.
- Nathan Dumlao. Untitled. Unsplash License.
- Rythik. "Silhouette." Unsplash License.
- Shirley Chen. Untitled. Unsplash License.
- Claudio Schwarz. Untitled. Unsplash License.
- Chris Lynch. "Man sleeping on couch." Unsplash License.
- Richard Bell. Untitled. Unsplash License.
- Richard Beatson. Untitled. Unsplash License.
- David Clode. "I am very glad that I got some photos…." Unsplash License.
- Kym MacKinnon. "Ombre Sunrise Reflections." Unsplash License.
- Jacqueline O'Gara. Untitled. Unsplash License.
- Tim Johnson. Untitled. Unsplash License.
- Howei Wang. Untitled. Unsplash License.
- Xavier von Erlach. "The sky above the Swiss Alps after midnight." Unsplash License.