REST API

Sridhar Alagar

REST

Acronym for Representational State Transfer

An architectural style by Roy Fielding as Ph.D dissertation

These days, “REST service,” “REST API” or “RESTful API” more-than-likely mean

- an HTTP or Web-based server that accepts requests over HTTP and responds in human-readable JSON.

The six constraints

1. Uniform Interface

2. Stateless

   Each request from client to server contains all necessary information

3. Cacheable

   Responses can be cached to improve performance

4. Client Server

   Separation of concerns help client and server evolve independently

5. Layered System

   Composed of hierarchical layers, e.g., MVC pattern

6. Code on demand

Uniform interface

- Identification of resources
  - The interface must uniquely identify each resource involved in the interaction between the client and the server.
- Manipulation of resources through representations
  - The resources should have uniform representations in the server response. API consumers should use these representations to modify the resource state in the server.
- Self-descriptive messages
  - Each resource representation should carry enough information to describe how to process the message. It should also provide information of the additional actions that the client can perform on the resource.
- Hypermedia as the engine of application state
  - The client should have only the initial URI of the application. The client application should dynamically drive all other resources and interactions with the use of hyperlinks.

REST structure and endpoints

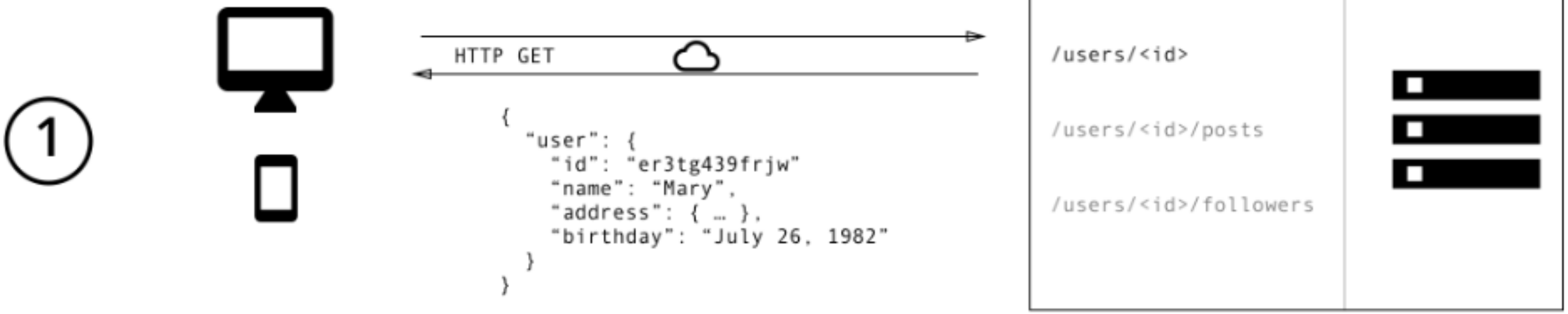
- Resources are represented as URLS
  - Provide sensible resource name, e.g.,
    - /users/12345
    - /customers/33245/orders/8769/lineitems/1
- Use HTTP verb to mean something
  - GET /users: Retrieves all users
  - POST /users: Creates a new user
- Use HTTP response codes to indicate status
- Favor JSON
- Response should contain links wherever possible

Limitations of REST

- Over-fetching
  - Fetching a user data with additional fields that are not needed
- Under-fetching
  - Fetching a user without posts requires additional fetches
- Complex associated data (nested) requires multiple fetches
- Fixed data structure per endpoint

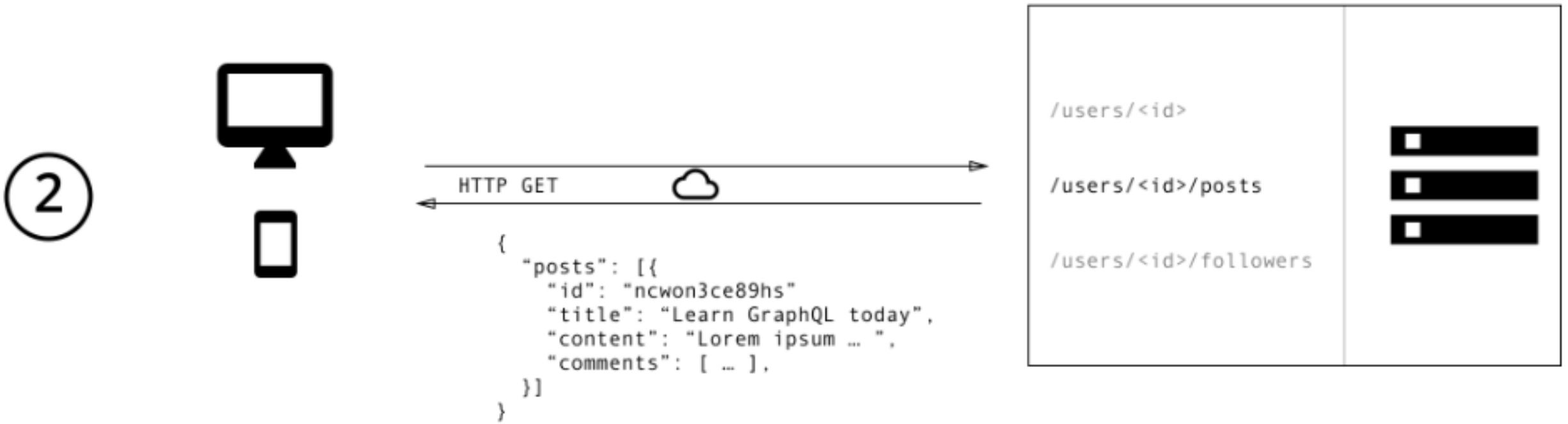
Data fetching with REST

Scenario: In a blogging app, need to display the titles of the posts of a specific user. Same page should also display last 3 followers



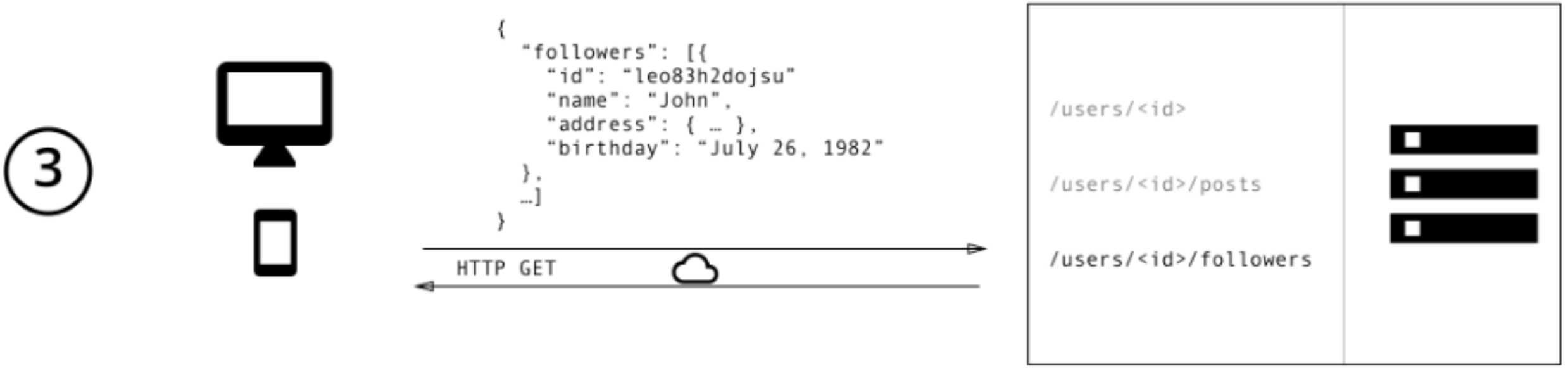
Data fetching with REST

Scenario: In a blogging app, need to display the titles of the posts of a specific user. Same page should also display last 3 followers



Data fetching with REST

Scenario: In a blogging app, need to display the titles of the posts of a specific user. Same page should also display last 3 followers



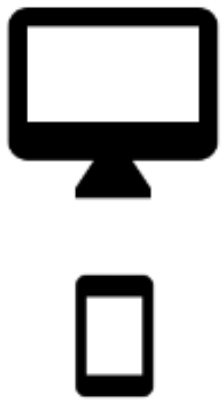
GraphQL is the better REST

- GraphQL is a query language and runtime for APIs, allowing clients to specify the exact data structure they need

- A single endpoint receives queries, providing only requested fields

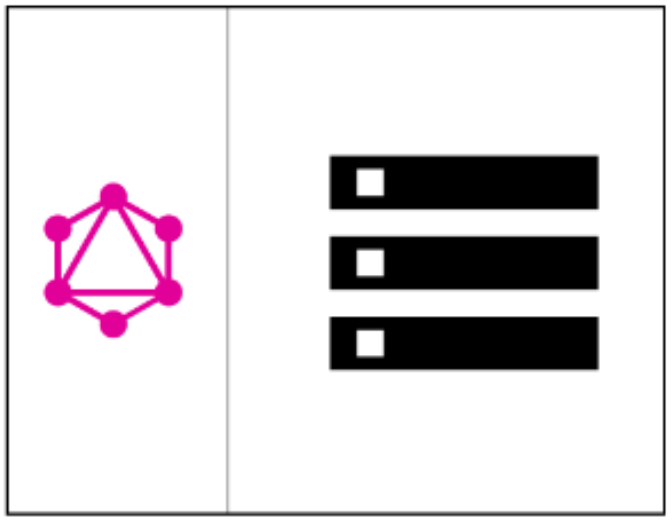
Data fetching with GraphQL in the blogging APP

```
query {
  User(id: "er3tg439frjw") {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```



HTTP POST


```
{
  "data": {
    "User": {
      "name": "Mary",
      "posts": [
        { title: "Learn GraphQL today" }
      ],
      "followers": [
        { name: "John" },
        { name: "Alice" },
        { name: "Sarah" },
      ]
    }
```



GraphQL – core concepts

- Query: Retrieve data
- Mutation: Modify data
- Subscription: Listen for real-time updates

Sources

1. Learn REST API Design
2. REST API Tutorial
3. GraphQL is the better REST
4. GraphQL
5. CS 142 Lectures