RESTful Services

https://dev.twitter.com/docs/api

Developers API Health Blog Discussions Documentation Search Sign in

Home

The Twitter REST API

Jump to

REST API version 1.1

The most recent version of the Twitter REST API.

API v1.1 Resources »

Rate Limiting in API v1.1 »

Authenticating »

Announcement »

REST API version 1

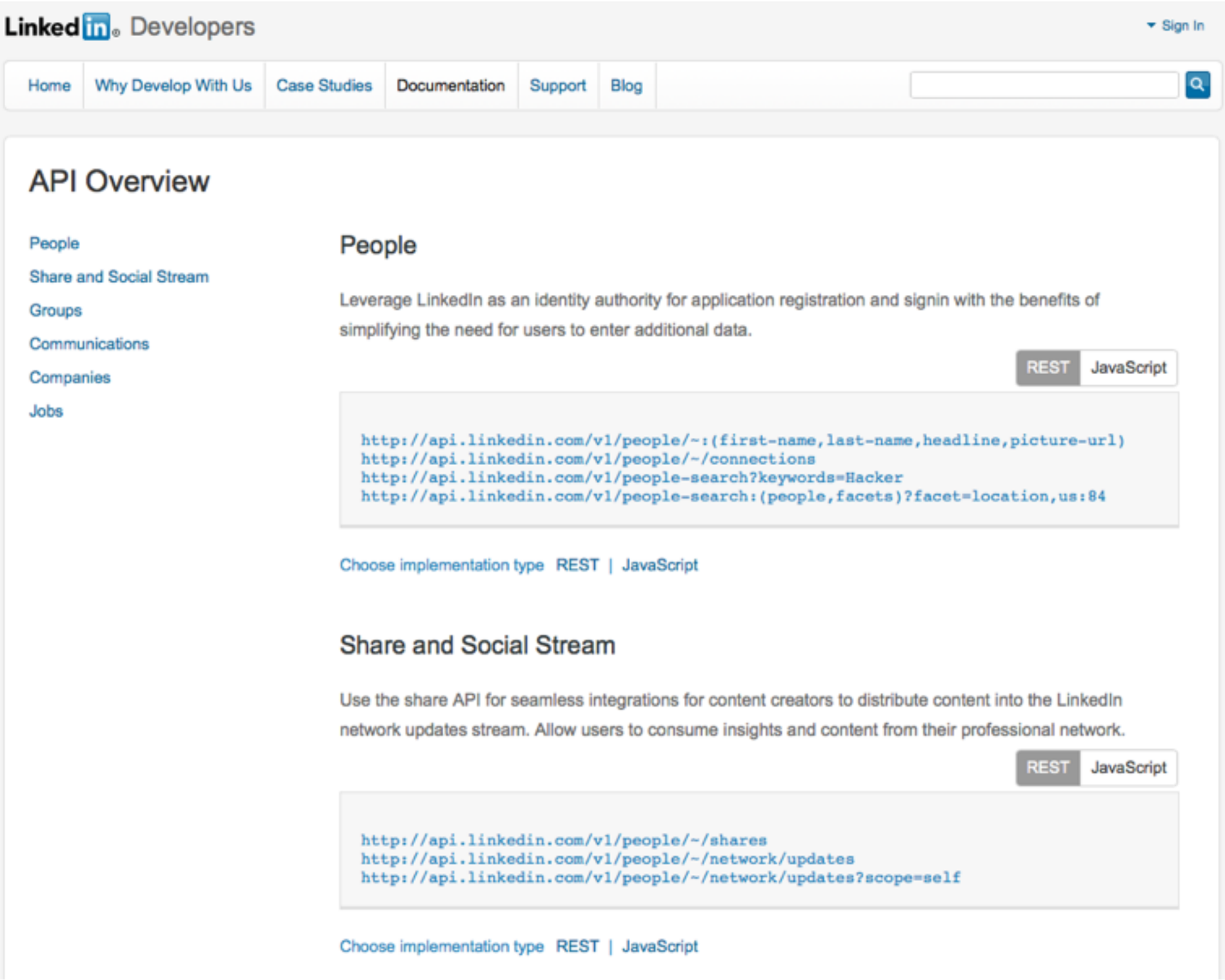
Version 1 of the REST API is now deprecated and will cease functioning in the coming months. Migrate to version 1.1 today.

Review the deprecated version 1 API »

Follow @twitterapi

API Terms API Status Blog Discussions Documentation A Drupal community site supported by Acquia

http://developer.linkedin.com/apis



http://docs.aws.amazon.com/AmazonS3/latest/API/APIRest.html

aws documentation

Amazon Simple Storage Service
API Reference (API Version 2006-03-01)

Search: Documentation – This Guide

REST API

Topics

This section contains information specific to the Amazon S3 REST API.

The examples in this guide use the newer virtual hosted-style method for accessing buckets instead of the path-style. Although the path-style is still supported for legacy applications, we recommend using the virtual-hosted style where applicable. For more information, see Working with Amazon S3 Buckets

The following example is a virtual hosted-style request that deletes the `puppy.jpg` file from the `mybucket` bucket.

```
DELETE /puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: mybucket.s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

The following example is a path-style version of the same request.

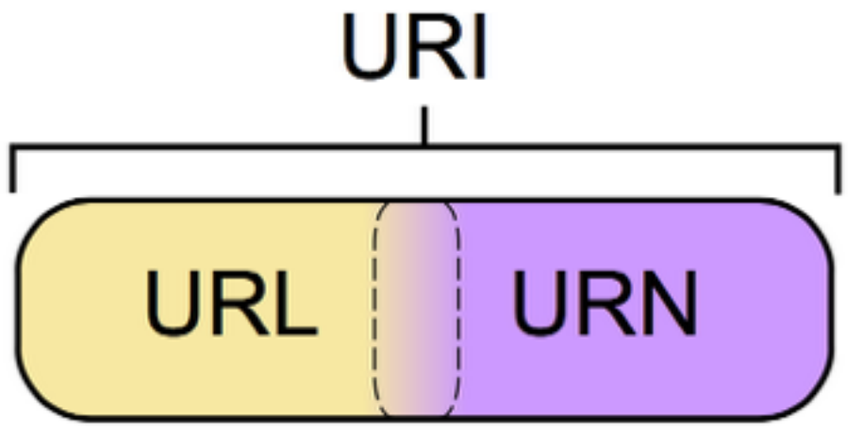
```
DELETE /mybucket/puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

Web Architectural Components

1. Identification: URI
   - uniform resource identifier
2. Interaction: HTTP
   - hypertext transfer protocol
3. Standard Document Format: HTML, XML, JSON
   - hypertext markup language
   - extensible markup language
   - javascript object notation

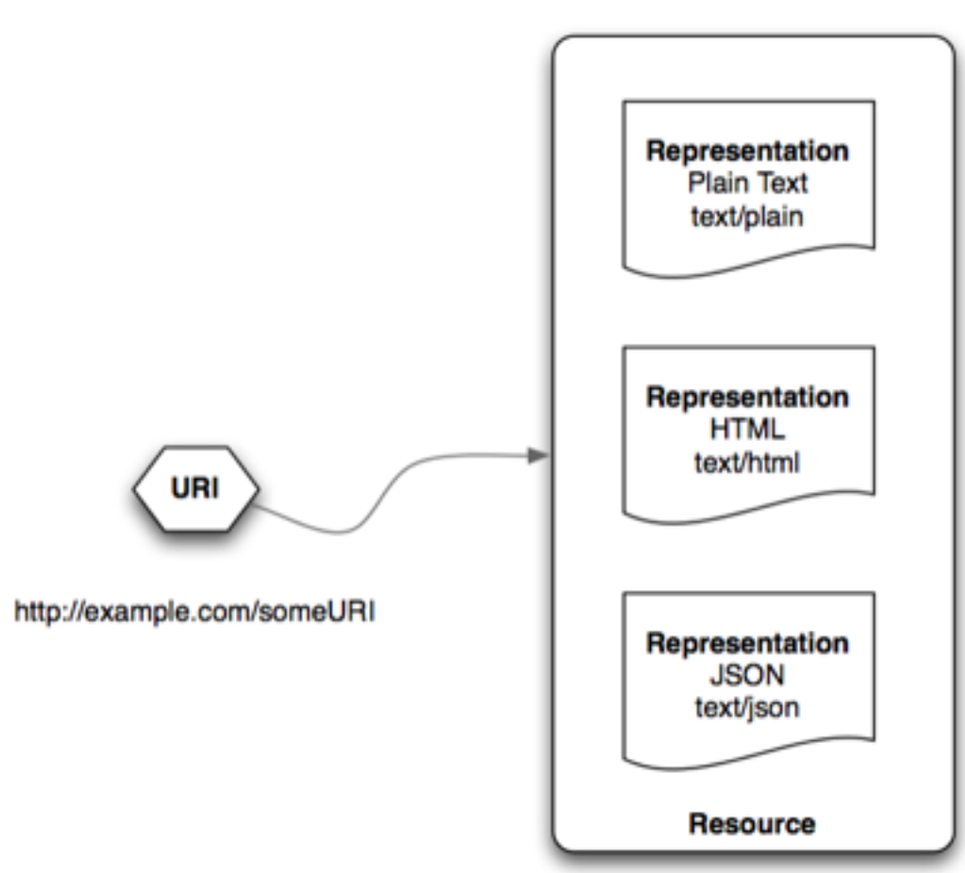
URIs

- URIs identify interesting things in the Web
  - documents on the Web
  - relevant aspects of a data set
- A URN (uniform resource name) defines an item's identity
  - A URN functions like a person's name
- A URL (uniform resource locator) provides a method for finding an item
  - A URL resembles that person's street address, while
- A URI can be a URL, a URN or both



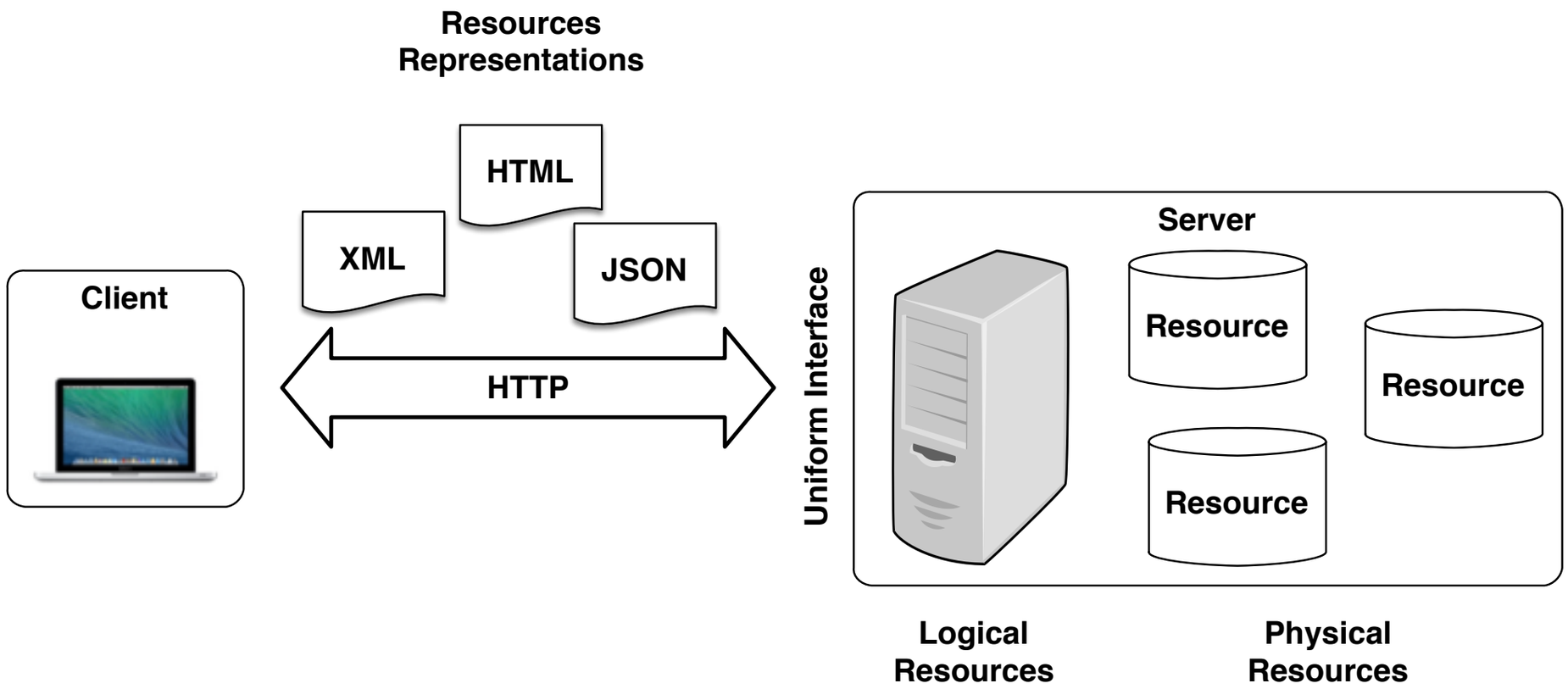
URI Examples

- HTTP URIs name and address resources in a Web system
  - A URI names and identifies one resource
  - A resource can have more than one name
    - http://example.com/software/latest-release
    - http://example.com/software/release-1.4
- A resource can have several representations:



Interacting with resources

- We interact with resource representations
  - not the resources themselves
  - representations can be in any format
    - defined by media type
- Each resource implements a standard uniform interface: HTTP
  - a small set of verbs applied to a large set of nouns
  - verbs are universal and not invented on a per-application basis



REST and ROA

- Representational State Transfer (REST)
  - Based on chapter 5 of Roy Fielding's PhD thesis (2000)
- An architectural style for building loosely coupled systems
  - The Web itself is an instance of that style
- Can be used to build Web services
- Resource Oriented Architecture (ROA)
  - A set of design principles to build RESTful Web services

Architectural Principles

- Addressability
- Uniform Interface
- Connectedness
- Statelessness

Addressability

- An addressable application
  - exposes the interesting aspects of its dataset as resources
  - exposes a URI for every piece of information it might serve
  - which is usually an infinite number of URIs
- A resource
  - is anything that is important enough to be referenced as a thing in itself
  - usually something
    - that you want to serve information about
    - that can be represented as a stream of bits
      - actors
      - movies
  - a resource must have at least one name (URI)
- Resource names
  - the URI is the name and address of a resource
  - resource's URI should be descriptive:

GOOD: http://example.com/movies

BAD: http://example.com/overview.php?list=all,type=movie

Uniform Interface

- The same set of operations applies to everything (every resource)
- A small set of verbs (methods) applied to a large set of nouns (resources)
- Verbs are universal and not invented on a per-application base
  - Natural language works in the same way (new verbs rarely enter language)
- HTTP defines a small set of verbs (methods) for acting on URI-identified resources

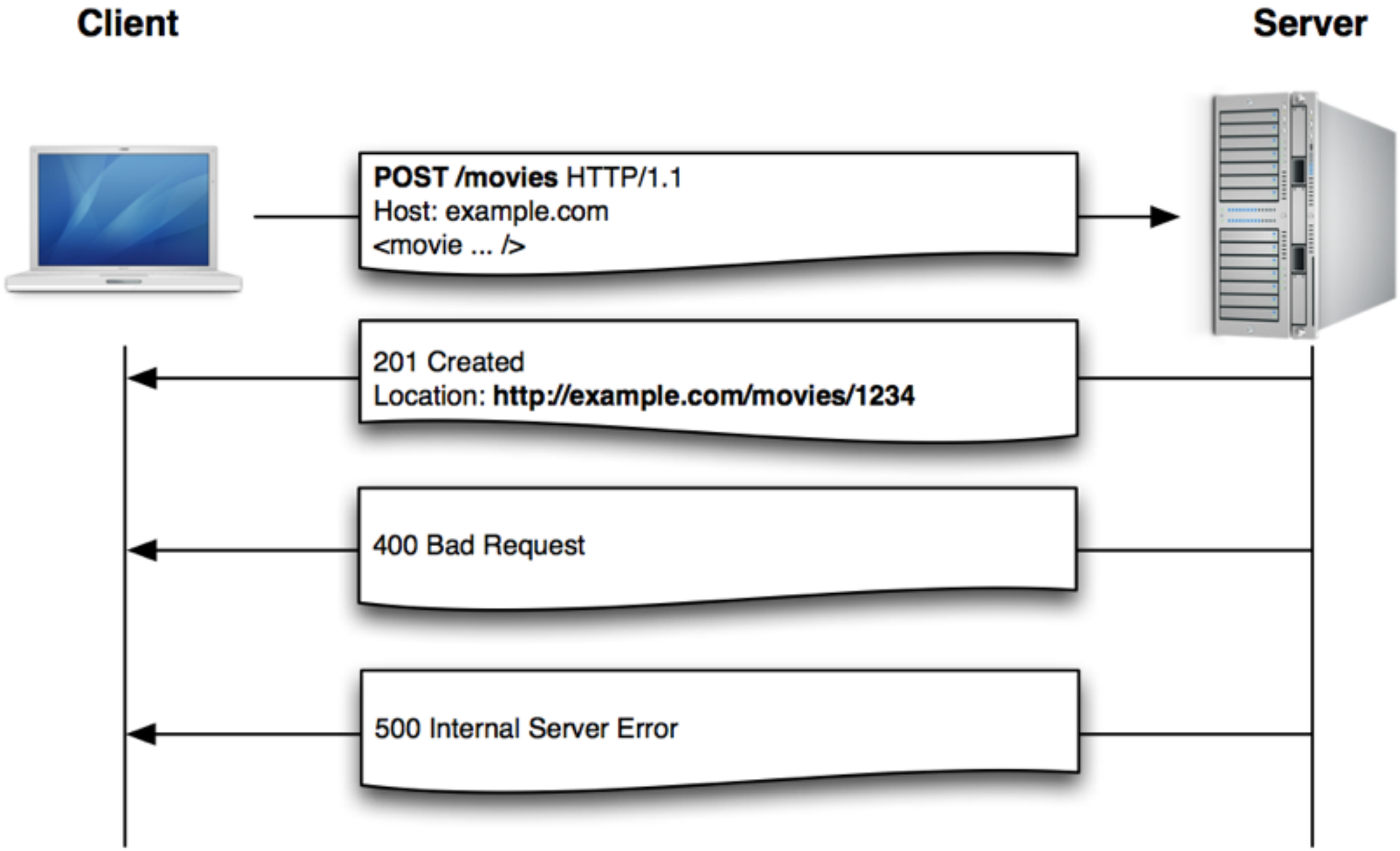
- RESTful Web Services use HTTP to its full extent
  - Methods: GET, POST, PUT, DELETE, (...)
  - Request headers: Authorization, Content-Type, Last-Modified
  - Response Codes: 200 OK, 304 Not Modified, 401 Unauthorized, 500 Internal Server Error
  - Body: an envelope for data to be transported from A to B

CRUD with HTTP

- With HTTP we have all methods we need to manipulate Web resources
- CRUD interface:
  - Create = POST (or PUT)
  - Read = GET
  - Update = PUT
  - Delete = DELETE
- Safe and idempotent behavior
  - Safe methods can be ignored or repeated without side-effects: GET and HEAD
  - Idempotent methods can be repeated without side-effects: PUT and DELETE
  - Unsafe and non-idempotent methods should be treated with care: POST

CREATE

- CREATE a new resource with HTTP POST

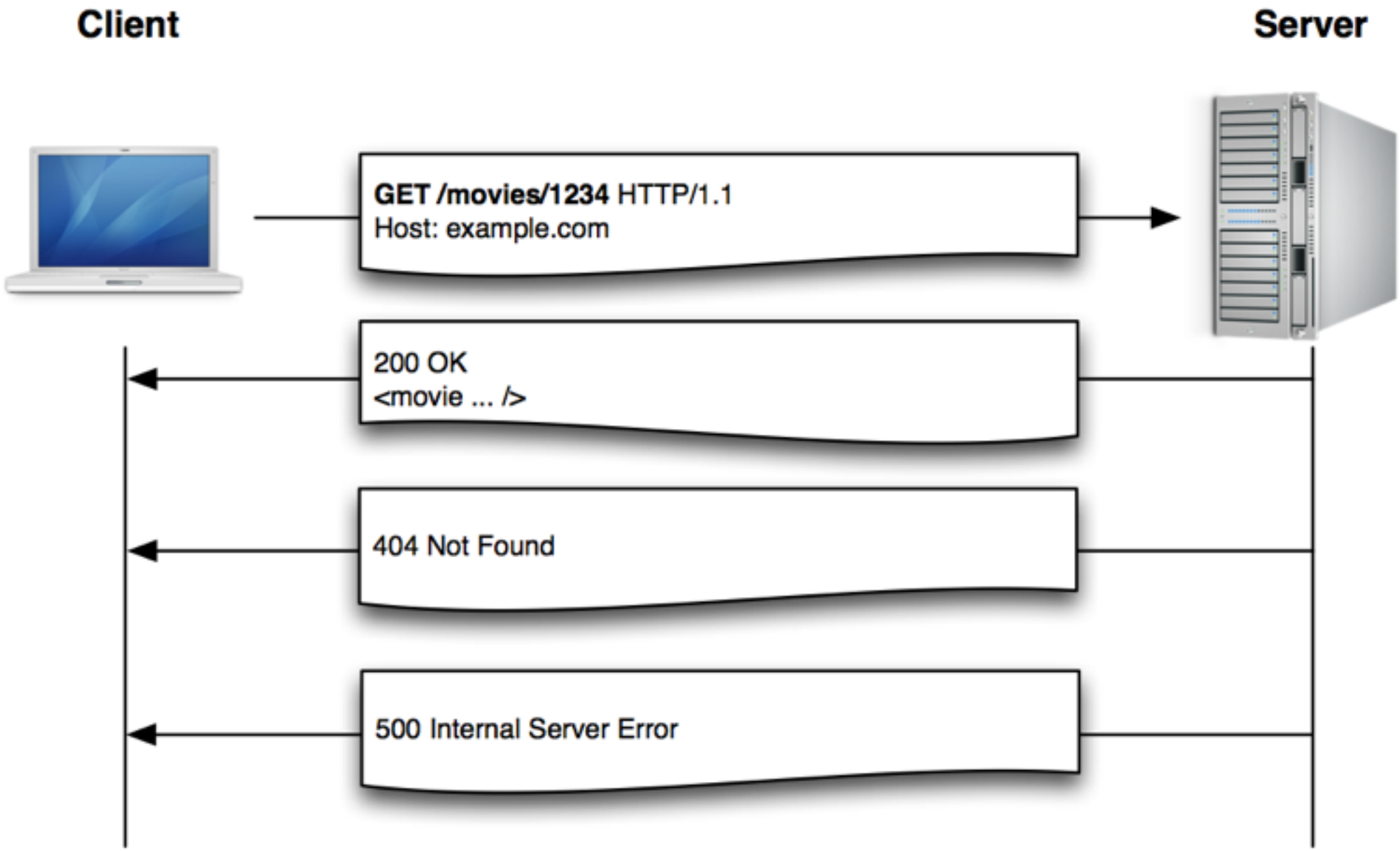


POST Semantics

- POST creates a new resource
- The server decides on the resource's URI
- POST is **not idempotent**
  - A sequence of two or more POST requests has side-effects
  - Human Web:
    - "Do you really want to post this form again?"
    - "Are you sure you want to purchase that item again?"
  - Programmatic Web:
    - if you post twice, you create two resources

READ

- READ an existing resource with HTTP GET

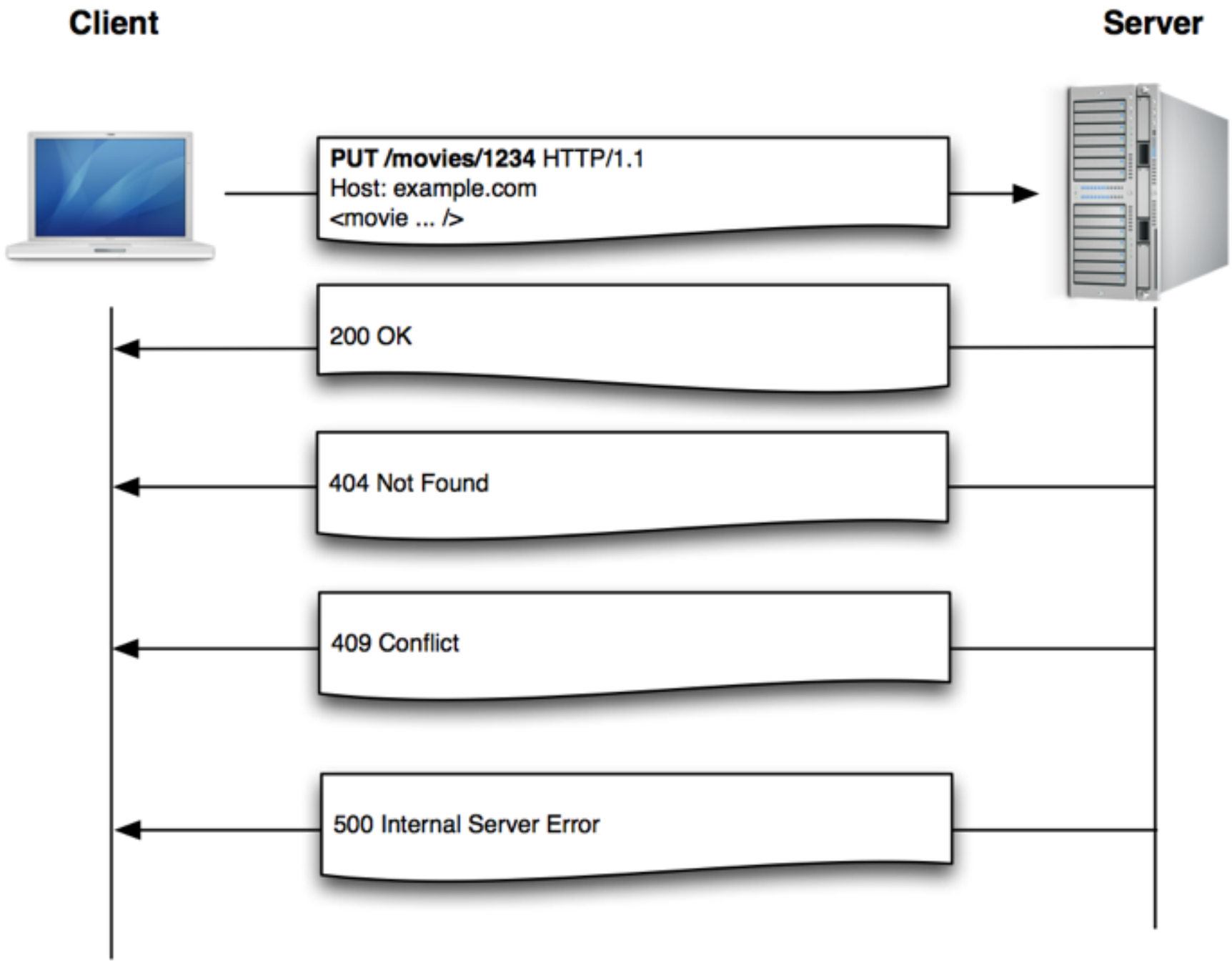


GET Semantics

- GET retrieves the representation (i.e., the current state) of a resource
- GET is **safe** (implies idempotent)
  - does not change state of resource
  - has no side-effects
- If GET goes wrong
  - GET it again!
  - no problem because it safe (and idempotent)

UPDATE

- UPDATE an existing resource with HTTP PUT

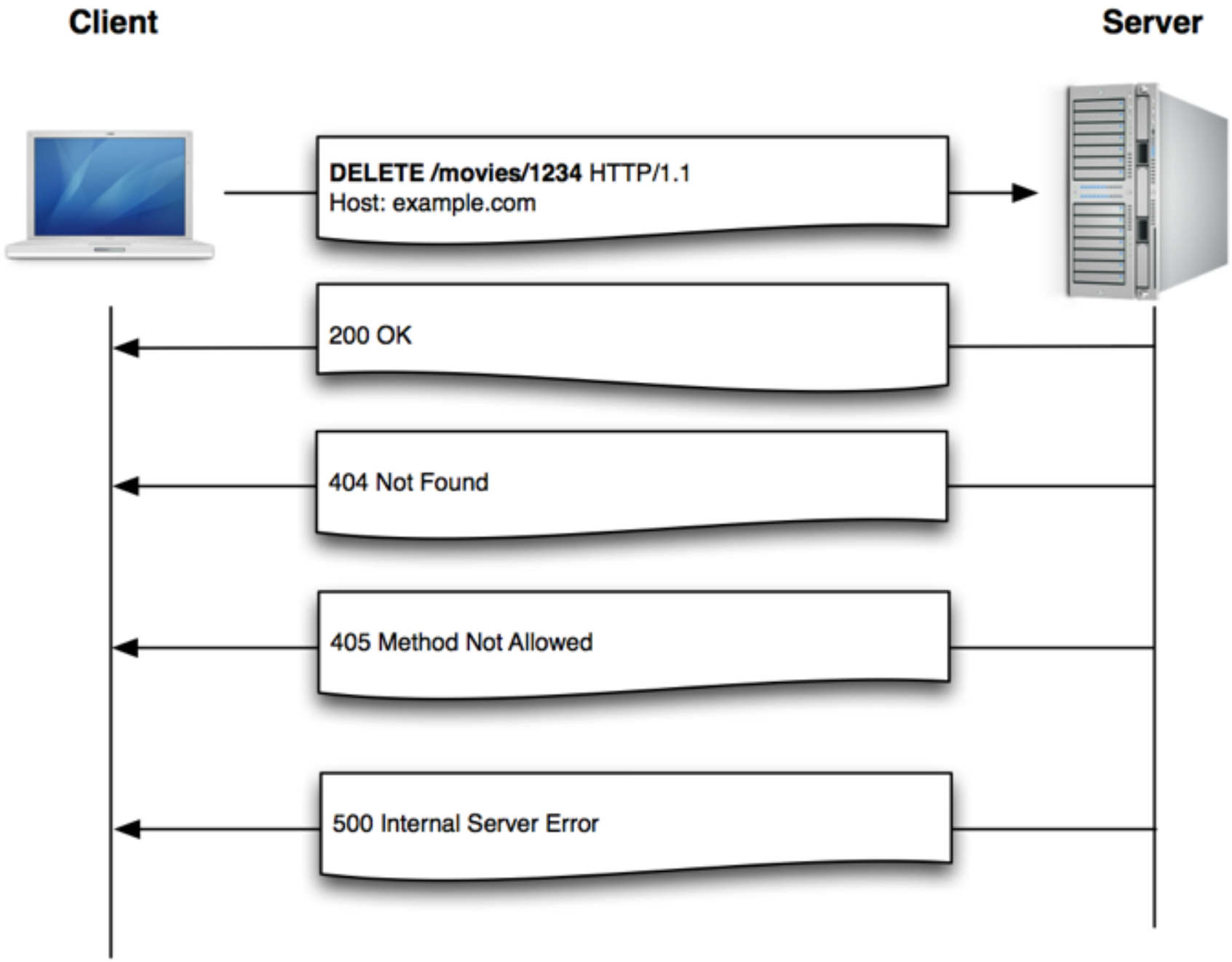


PUT Semantics

- PUT creates a new resource
- The client decides on the resource's URI
- PUT is **idempotent**
  - multiple PUT requests have no side effects
  - but it changes the resource state

DELETE

- DELETE an existing resource with HTTP DELETE



DELETE Semantics

- Stop the resource from being accessible
  - logical delete
  - not necessarily physical
- If DELETE goes wrong
  - try it again!
  - DELETE is idempotent

Representations in HTTP

- In HTTP, the format of a resource is identified through a MIME code
  - Multimedia Internet Mail Extension
  - format: type/subtype

Examples:

- Examples:
  - text/plain, text/html
  - application/xml
  - image/jpeg

Connectedness

- RESTful services representations are hypermedia documents
- These are documents that contain not just data, but links to other resources
- The server guides the client's path by serving “hypermedia”: links and forms inside hypertext representations
- The server sends the client guidelines about which states are near the current one.
- The quality of having links is called “connectedness”.
- Resources should link to each other in their representations.
- Hence, why the human web is easy to use because it is well connected

Statelessness

- Statelessness = every HTTP request executes in complete isolation
- The request contains all the information necessary for the server to fulfill that request
- The server never relies on information from a previous request
  - if information is important (e.g., user- authentication), the client must send it again
- This constraint does not say "stateless applications"!
  - for many RESTful applications, state is essential (e.g., shopping carts)
- It means to move state to clients or resources
- State in resources
  - the same for every client working with the service
  - when a client changes resource state other clients see this change as well
- State in clients (e.g., cookies)
  - specific to client and has to be maintained by each client – makes sense for maintaining session state (login / logout)

Tools and Frameworks

- Restlet - framework for mapping REST concepts to Java classes
  - http://www.restlet.org
- Django - framework for building RESTful Web applications in Python
  - http://django-rest-framework.org
- Ruby on Rails - a framework for building RESTful Web applications
  - http://www.rubyonrails.org/
- JAX-RS - a specification provides a Java API for RESTful Web Services over the HTTP protocol
  - https://jax-rs-spec.java.net
- Jersey - the open source reference implementation of JAX-RS
  - https://jersey.java.net
- RESTEasy - JBoss project that provides various frameworks for building RESTful Web Services and RESTful Java applications. Fully certified JAX-RS implementation.
  - http://www.jboss.org/resteasy/