Working with Caching and Concurrency

Kevin Dockx
ARCHITECT

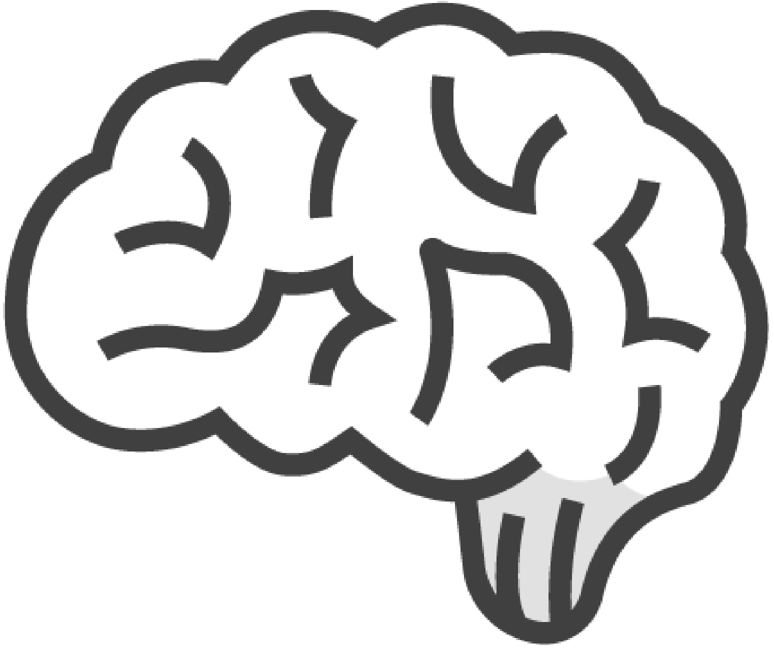
@KevinDockx https://www.kevindockx.com

Coming Up

Working with caching

Concurrency in a RESTful world

Working with Caching



Each response should define itself as cacheable or not

HTTP Caching

- http://bit.ly/2hJTTxD (RFC 2616)
- http://bit.ly/2in4uzh (RFC 7234)

> “Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases.”

HTTP standard

The Purpose of Caching

Eliminate the number of requests

Reduces network-roundtrips

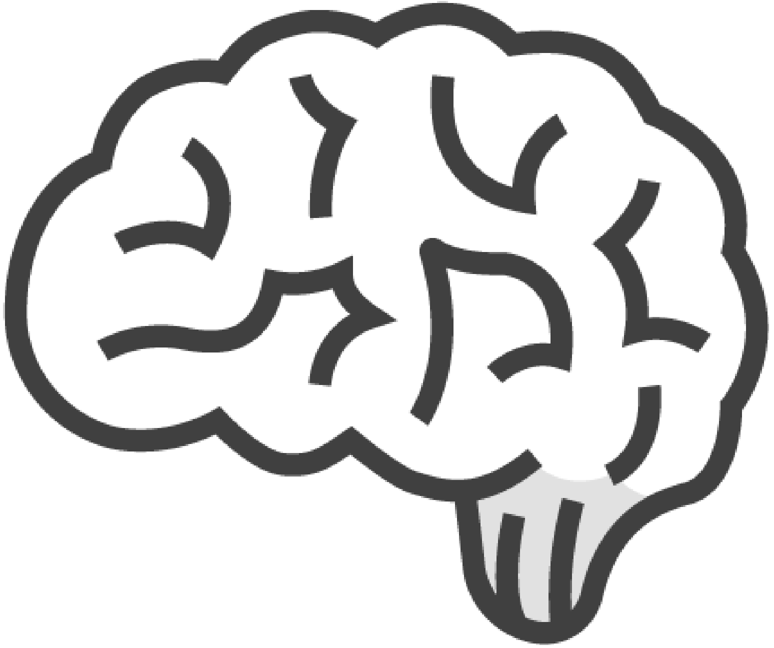
Expiration mechanism

Eliminate the need to send full responses

Reduces network bandwidth

Validation mechanism

Working with Caching



The cache is a separate component

- Accepts requests from consumer to the API
- Receives responses from the API and stores them if they are deemed cacheable

It's the middle-man of request-response communication

Cache Types

Client Cache

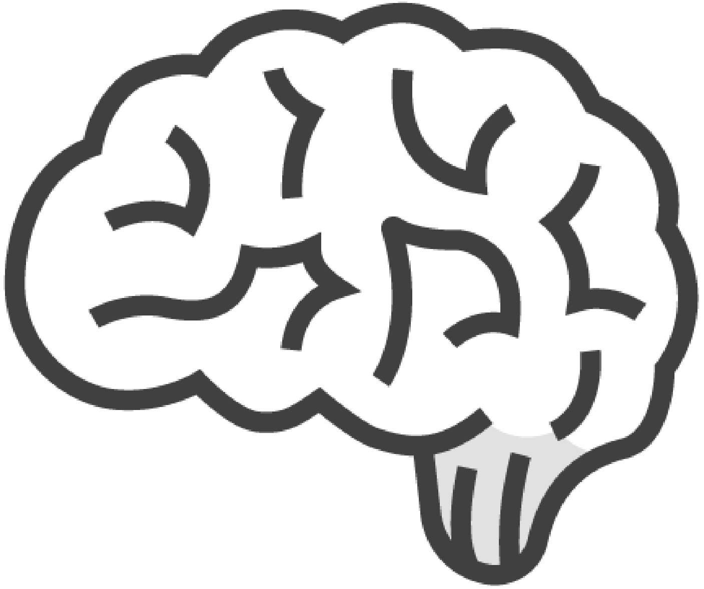
Lives on the client
Private cache

Gateway Cache

Lives on the server
Shared cache

Proxy Cache

Lives on the network
Shared cache



Expiration Model

Allows the server to state how long a response is considered fresh

Expiration Model

Expires header

Expires: Sat, 14 Jan 2017 15:23:40 GMT

Clocks must be synchronised

Offers little control

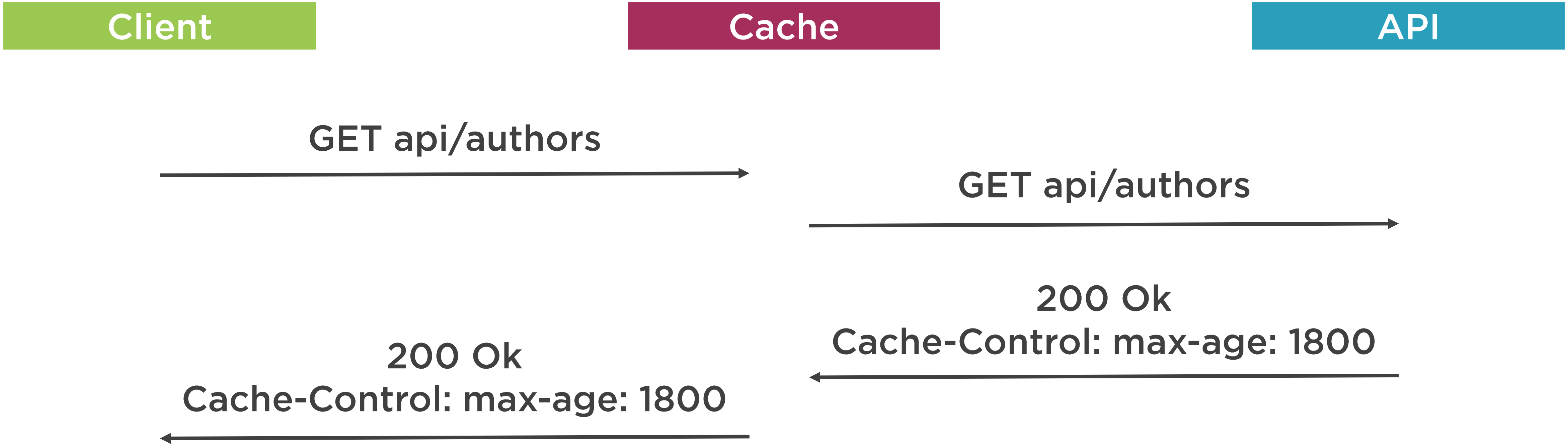
Cache-Control header

Cache-Control: public, max-age=60

Preferred header for expiration

Directives: http://bit.ly/1Ups120

Expiration Model



Client

Cache

API

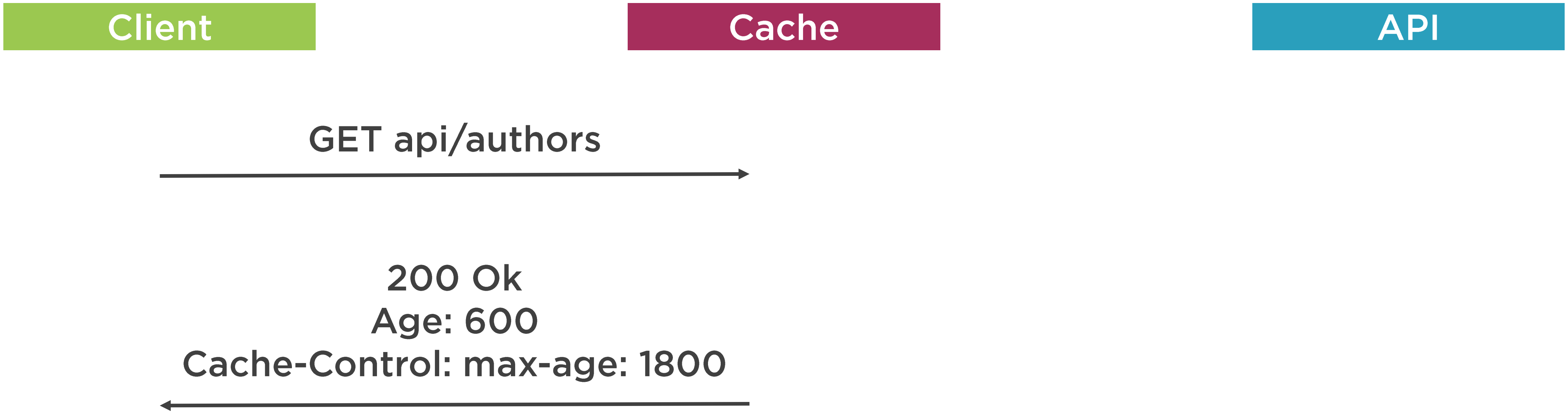
GET api/authors

GET api/authors

200 Ok
Cache-Control: max-age: 1800

200 Ok
Cache-Control: max-age: 1800

Expiration Model



Client

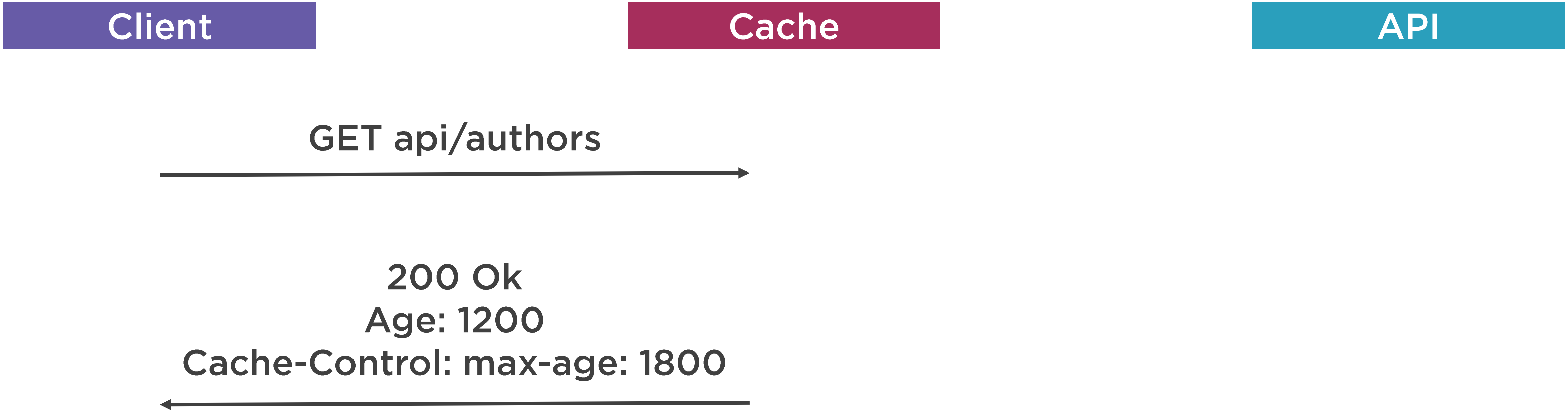
Cache

API

GET api/authors

200 Ok
Age: 600
Cache-Control: max-age: 1800

Expiration Model



Client

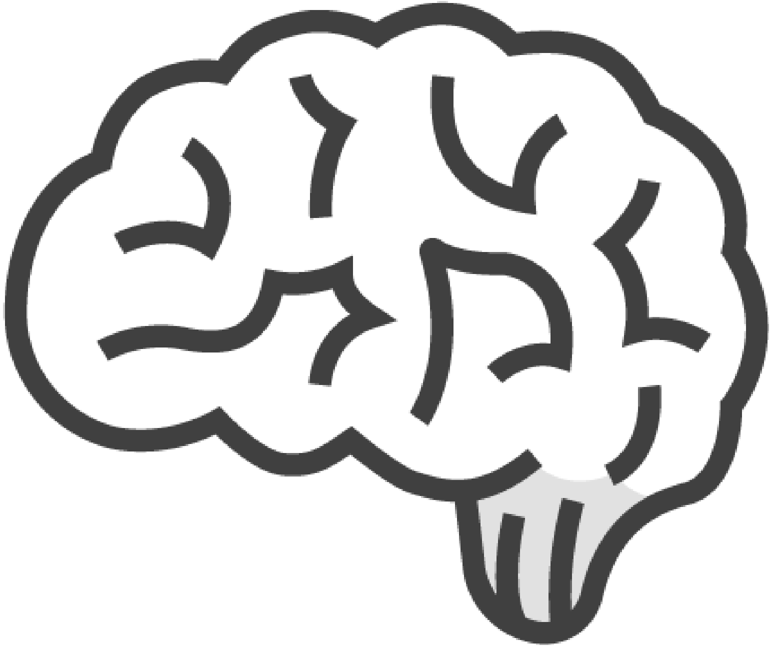
Cache

API

GET api/authors

200 Ok
Age: 1200
Cache-Control: max-age: 1800

Expiration Model and Cache Types

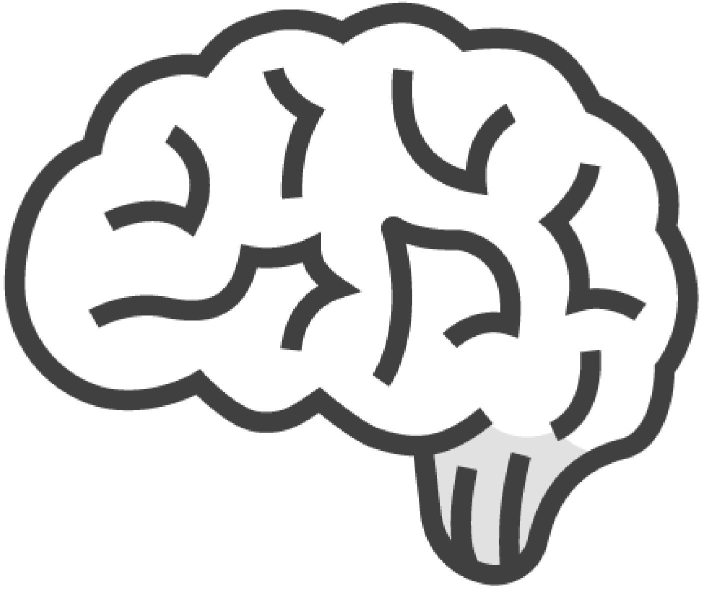


Private cache

- Reduces bandwidth requirements
- Less requests from cache to API

Shared (public) cache

- Doesn’t save bandwidth between cache and API
- Drastically lowers requests to the API



Validation Model

Used to validate the freshness of a response that's been cached

Validators

Strong validators

Change if the body or headers of a response change

ETag (Entity Tag) response header
ETag: “123456789”

Can be used in any context (equality is guaranteed)

Weak validators

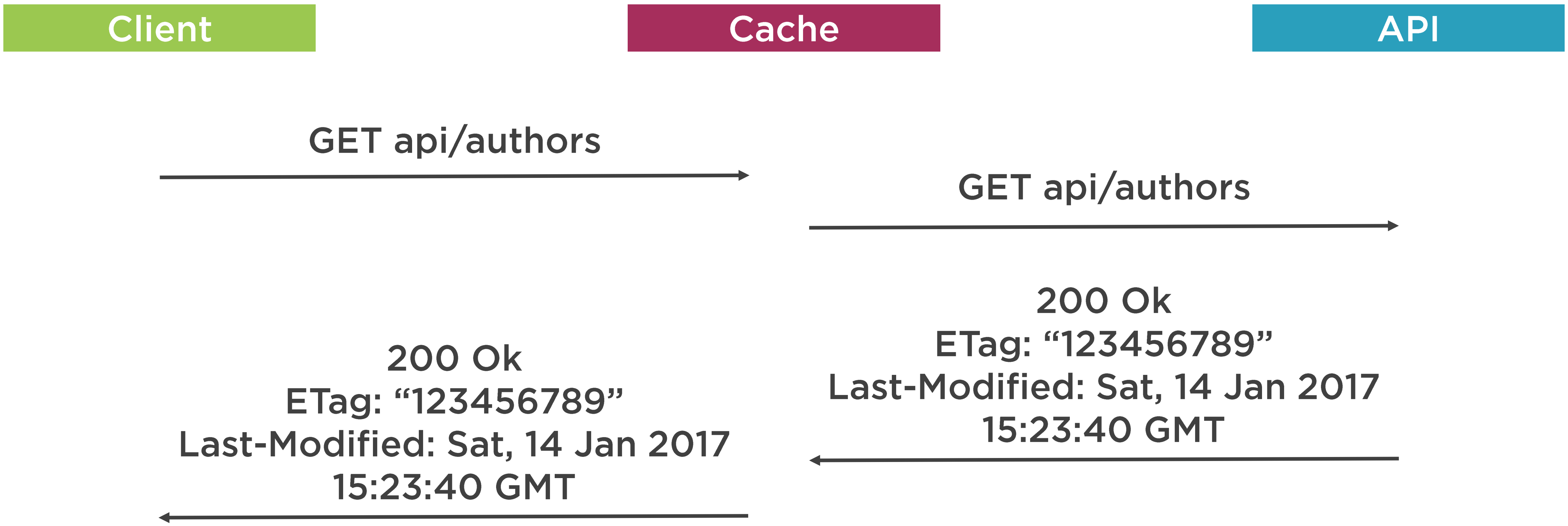
Don’t always change when the response changes (eg: only on significant changes)

Last-Modified: Sat, 14 Jan 2017 15:23:40 GMT

ETag: “w/123456789”

Equivalence, but not equality

Validation Model



Client

Cache

API

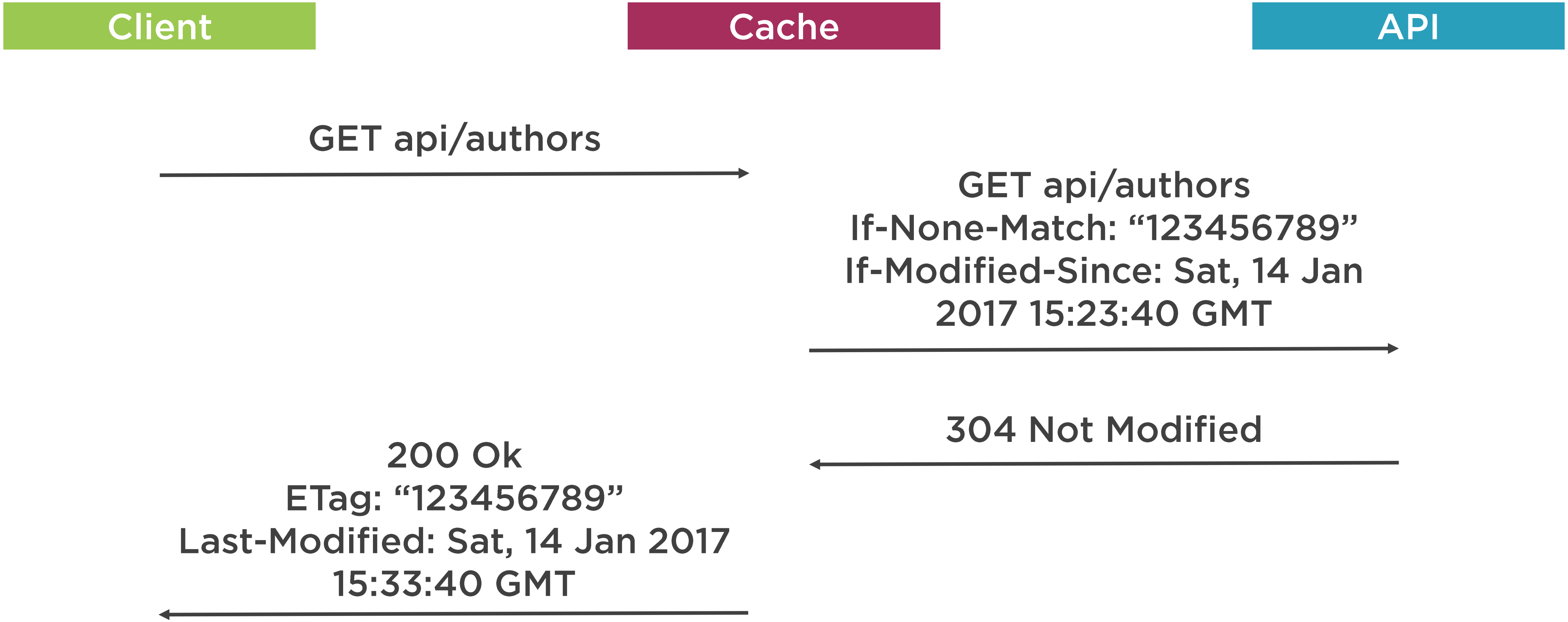
GET api/authors

GET api/authors

200 Ok
ETag: “123456789”
Last-Modified: Sat, 14 Jan 2017
15:23:40 GMT

200 Ok
ETag: “123456789”
Last-Modified: Sat, 14 Jan 2017
15:23:40 GMT

Validation Model



Client

Cache

API

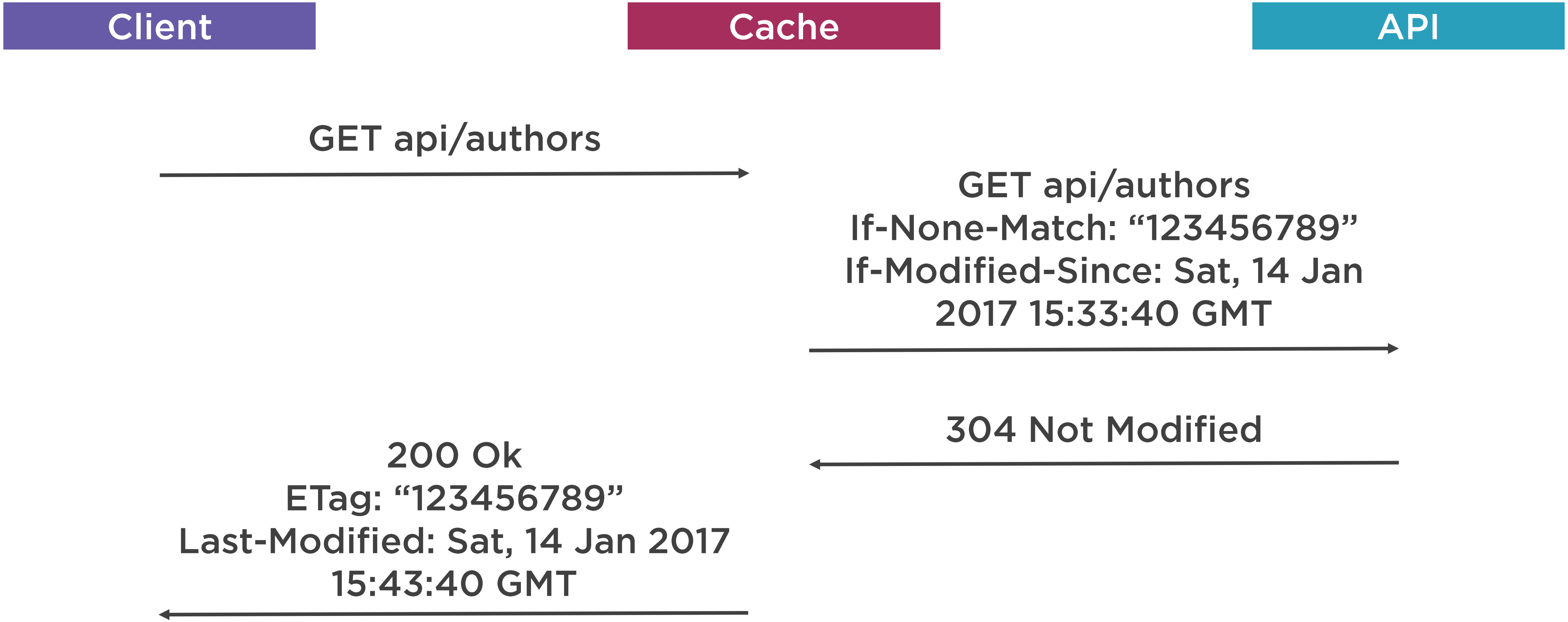
GET api/authors

GET api/authors
If-None-Match: “123456789”
If-Modified-Since: Sat, 14 Jan 2017 15:23:40 GMT

304 Not Modified

200 Ok
ETag: “123456789”
Last-Modified: Sat, 14 Jan 2017 15:33:40 GMT

Validation Model



Client

Cache

API

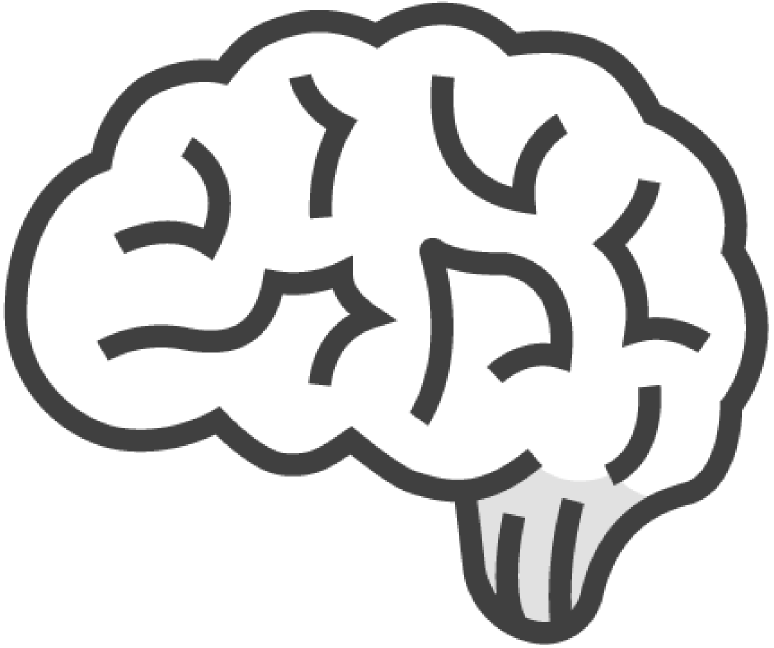
GET api/authors

GET api/authors
If-None-Match: “123456789”
If-Modified-Since: Sat, 14 Jan 2017 15:33:40 GMT

304 Not Modified

200 Ok
ETag: “123456789”
Last-Modified: Sat, 14 Jan 2017 15:43:40 GMT

Validation Model and Cache Types



Private cache

Reduces bandwidth requirements

Shared (public) cache

Saves bandwidth between cache and API

Expiration and Validation Combined

Private cache

As long as the response hasn't expired (isn't stale), that response can be returned from the cache

Reduces communication with the API (including response generation), reduces bandwidth requirements

If it has expired, the API is hit

Bandwidth usage and response generation is potentially reduced even more

Shared (public) cache

As long as the response hasn't expired (isn't stale), that response can be returned from the cache

Reduces bandwidth requirements between cache and API, dramatically reduces request to the API

If it has expired, the API is hit

Bandwidth usage between cache and API and response generation is potentially reduced

The Holy Grail of Caching

Combine private and shared caches

Cache-Control Directives

Response

Freshness
max-age, s-maxage

Cache type
public, private

Validation
no-cache, must-revalidate, proxy-revalidate

Other
no-store, no-transform

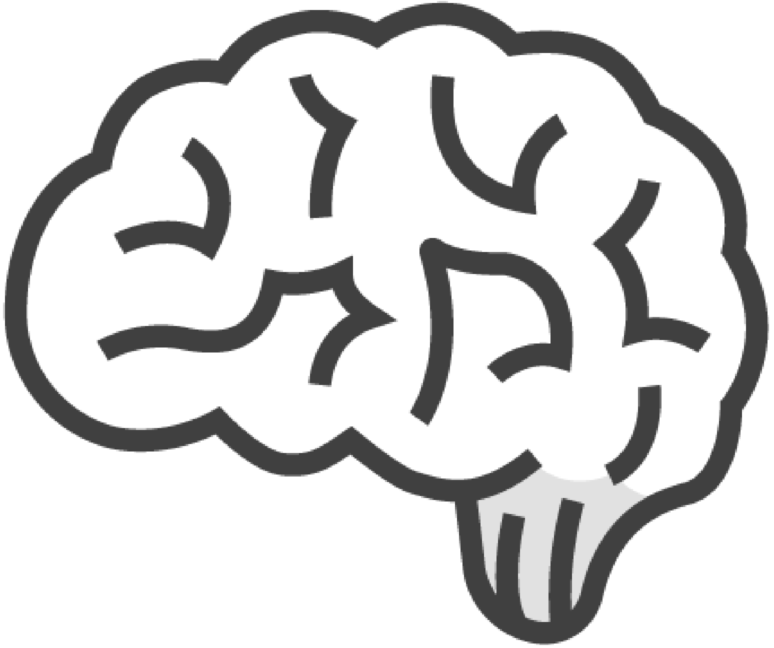
Request

Freshness
max-age, min-fresh, max-stale

Validation
no-cache

Other
no-store, no-transform, only-if-cached

Supporting Cache Headers



In ASP.NET Core: [ResponseCache]

Not sufficient for our purposes

In ASP.NET: CacheCow.Server

Not available for ASP.NET Core at the moment

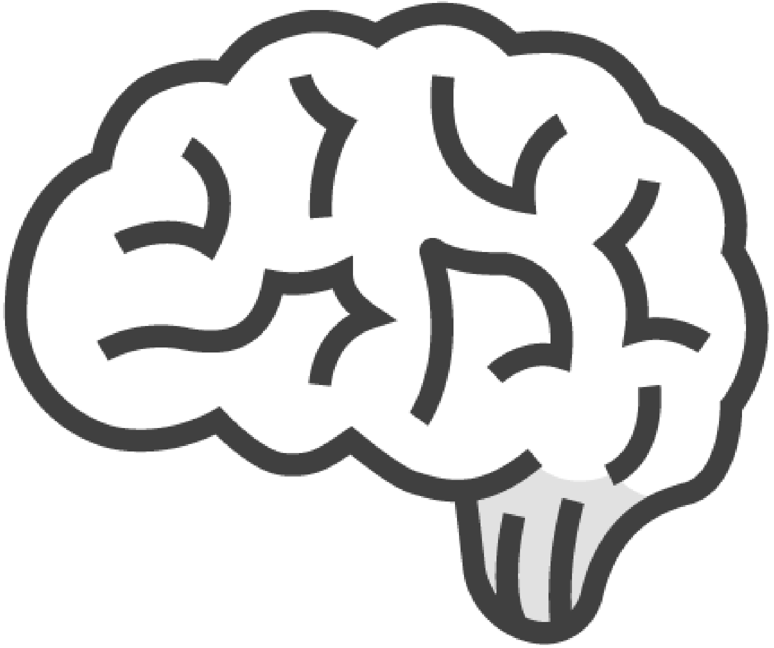
Marvin.Cache.Headers

- NuGet: http://bit.ly/2knD67M

Demo

Supporting HTTP Cache Headers

Cache Stores



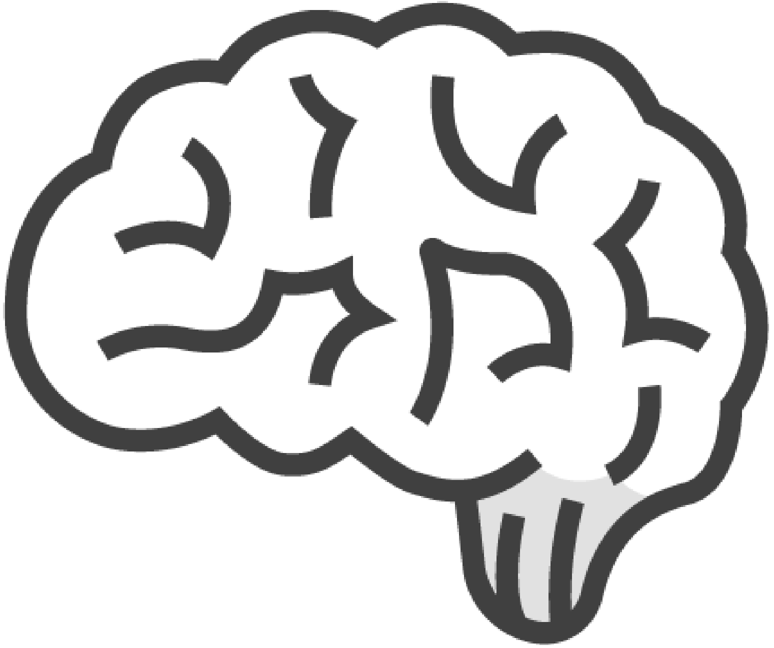
Private caches

- angular-http-etag (http://bit.ly/2jqm6hh)
- Marvin.HttpCache (http://bit.ly/2i6AuJE)
  - PCL, no .NET Core

Private and shared caches

- CacheCow.Client (http://bit.ly/2iqWIUs)
  - ASP.NET Web API, full .NET framework

Cache Stores



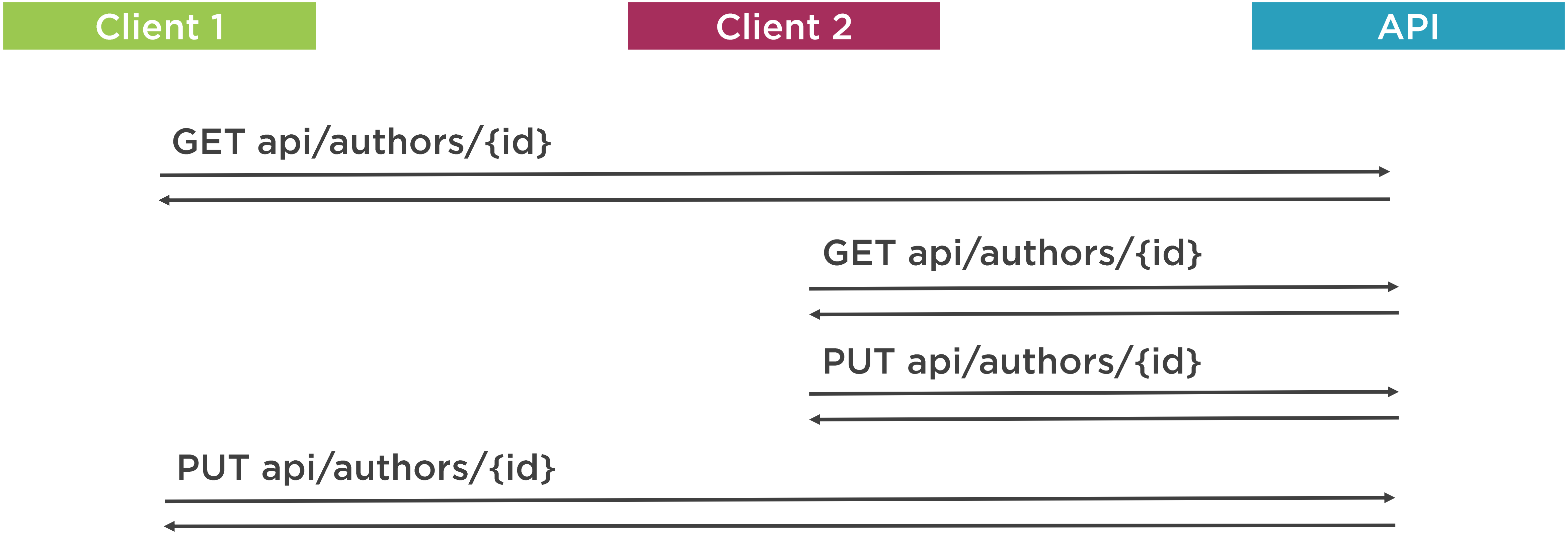
Shared caches (.NET Core)

- ASP.NET Core ResponseCaching middleware (http://bit.ly/2jq4UbM)
- Use from v2.0 onwards

Demo

Adding a Cache Store

Dealing with Concurrency in a RESTful World



Concurrency Strategies

Pessimistic concurrency

Resource is locked

While it's locked, it cannot be modified by another client

This is not possible in REST

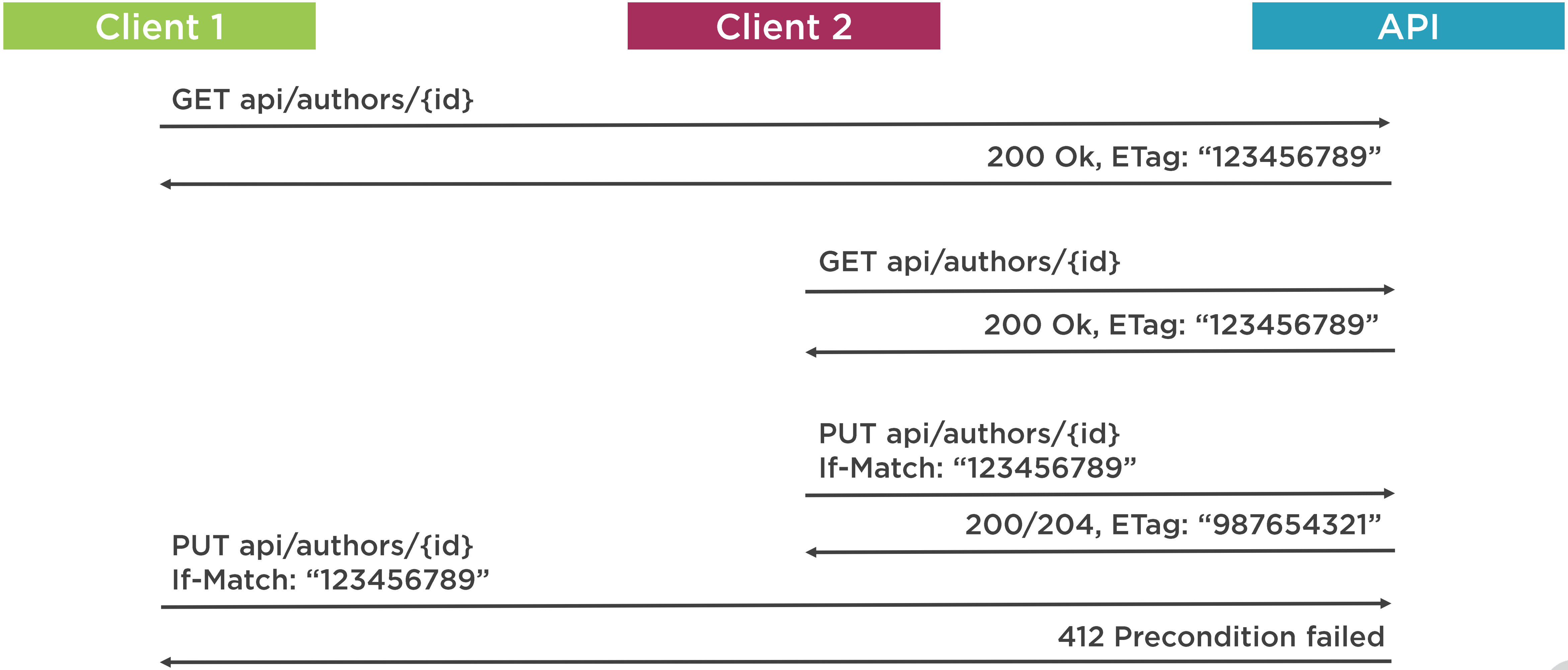
Optimistic concurrency

Token is returned together with the resource

The update can happen as long as the token is still valid

ETags are used as validation tokens

Dealing with Concurrency in a RESTful World



Client 1

Client 2

API

GET api/authors/{id}

200 Ok, ETag: “123456789”

GET api/authors/{id}

200 Ok, ETag: “123456789”

PUT api/authors/{id}
If-Match: “123456789”

200/204, ETag: “987654321”

PUT api/authors/{id}
If-Match: “123456789”

412 Precondition failed

Demo

Dealing with Concurrency

Summary

Each response must state whether or not it can be cached

Caching: expiration model

- Allows the server to state how long a response is considered fresh
- Cache-Control header

Summary

Caching: validation model

- Used to validate the freshness of a response that's been cached
- ETag, Last-Modified headers

Use ETags for optimistic concurrency