Getting Started with Caching Resources

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Cacheable constraint

HTTP cache

- Cache types
- Expiration and validation models
- `Cache-Control` header

Response caching middleware

Supporting the Cacheable Constraint

Each response should define itself as cacheable or not

Supporting the Cacheable Constraint

HTTP caching

- https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html (RFC 2616)
- https://datatracker.ietf.org/doc/html/rfc7234 (RFC 7234)
- https://datatracker.ietf.org/doc/html/rfc9111 (RFC9111)

“Caching would be useless if it did not significantly improve performance. The goal of caching 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

The Purpose of Caching

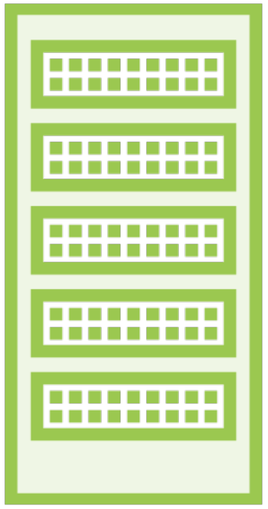
The cache is a separate component

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

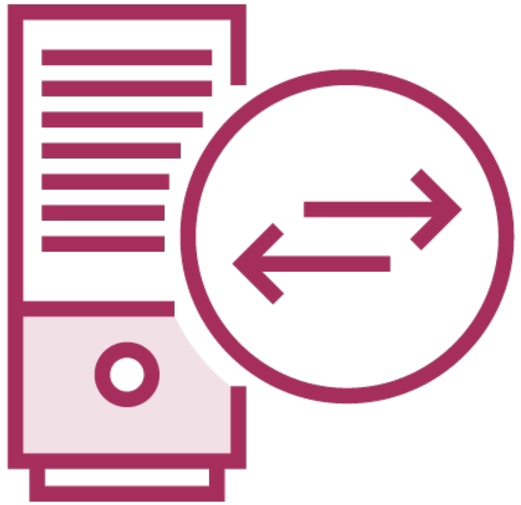
It's the middle-man of request-response communication

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

Response Cache Attribute and Middleware

State for each resource whether or not it's cacheable

- `Cache-Control`: max-age=120
- `[ResponseCache]` attribute
- This does not actually cache anything

Cache store

- Response caching middleware

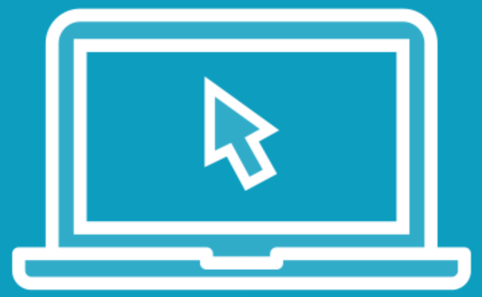
Demo

Adding cache headers to the response

Demo

Adding a cache store with the `ResponseCaching` middleware

Demo



Using cache profiles to apply the same rules to different resources

Expiration model

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

Expiration Model

Expires header

```
Expires: Thu, 21 Jul 2022
15:23:40 GMT
```

Clocks must be synchronised

Offers little control

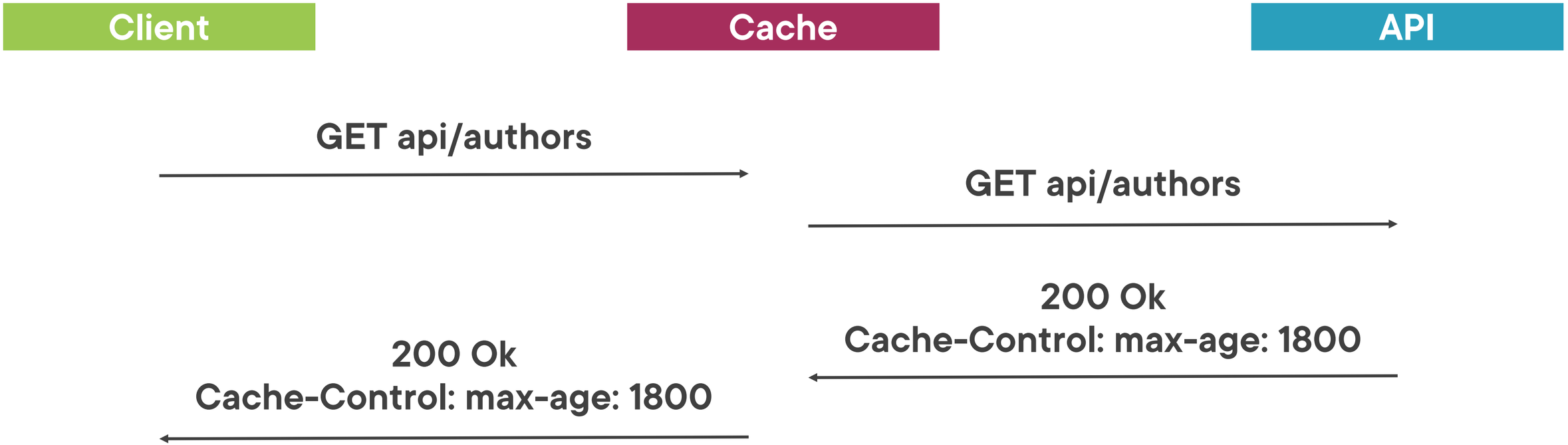
Cache-Control header

```
Cache-Control: public, max-age=60
```

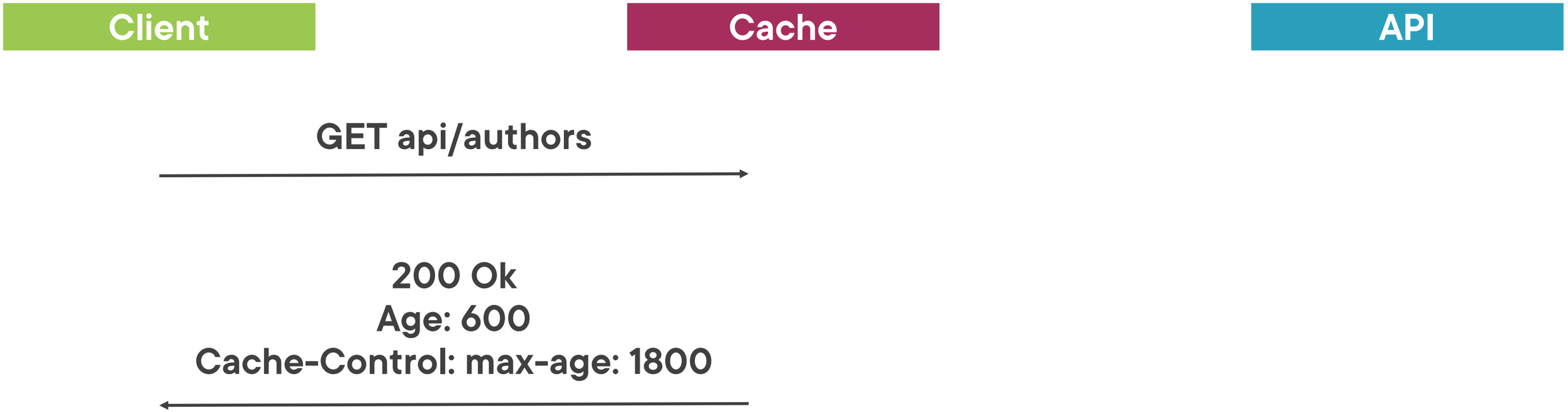
Preferred header for expiration

Directives:
https://datatracker.ietf.org/doc/html/rfc9111

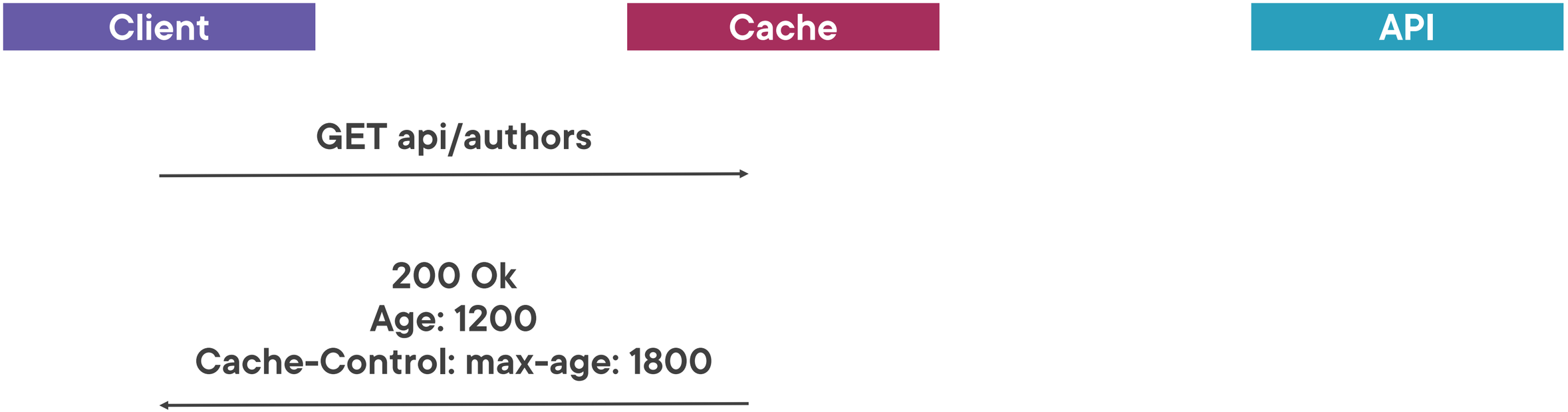
Expiration Model



Expiration Model



Expiration Model



Expiration Model and Cache Types

Private cache

- Reduces bandwidth requirements
- Less requests from cache to API

Shared (public) cache

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

Validation model

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

Validation Model

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

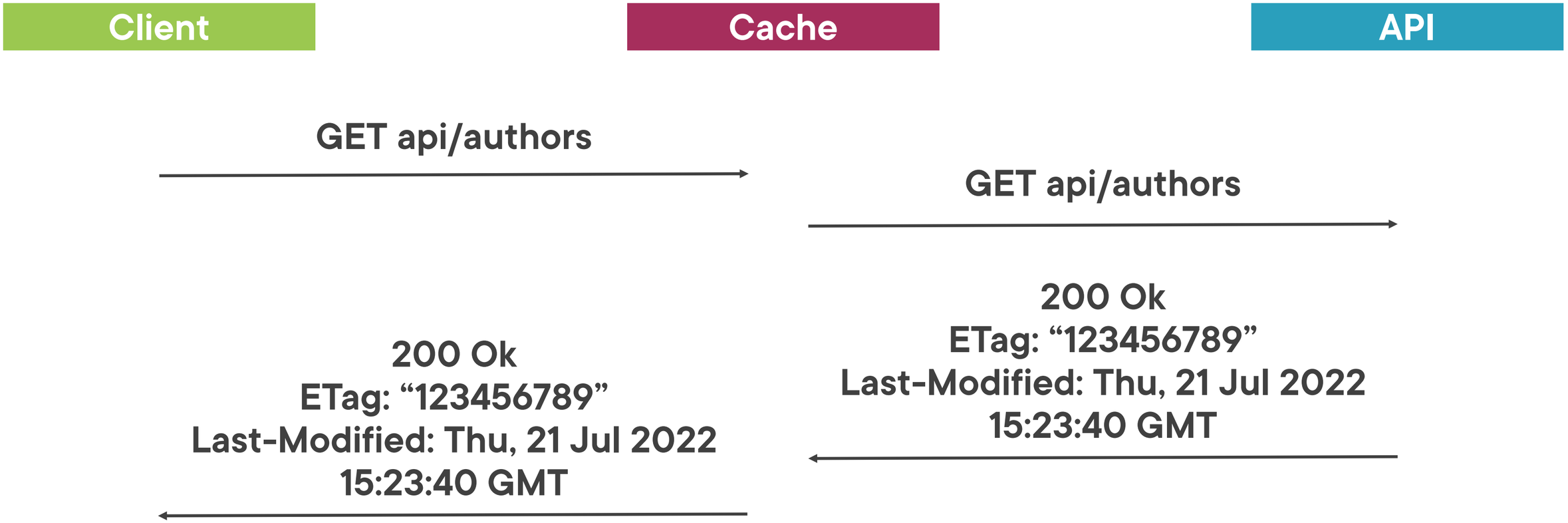
Don’t always change when the response changes (eg: only on significant changes)

```
Last-Modified: Thu, 21 Jul 2022
15:23:40 GMT
```

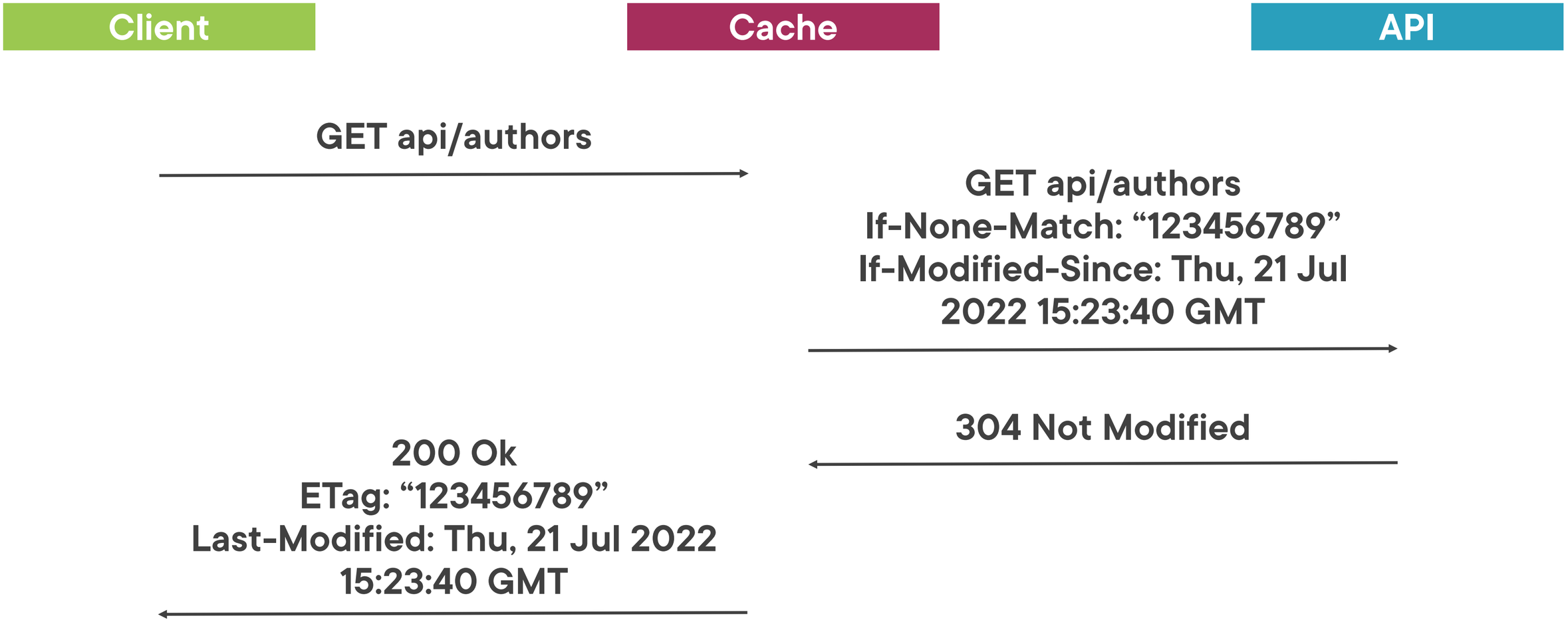
```
ETag: “w/123456789”
```

Equivalence, but not equality

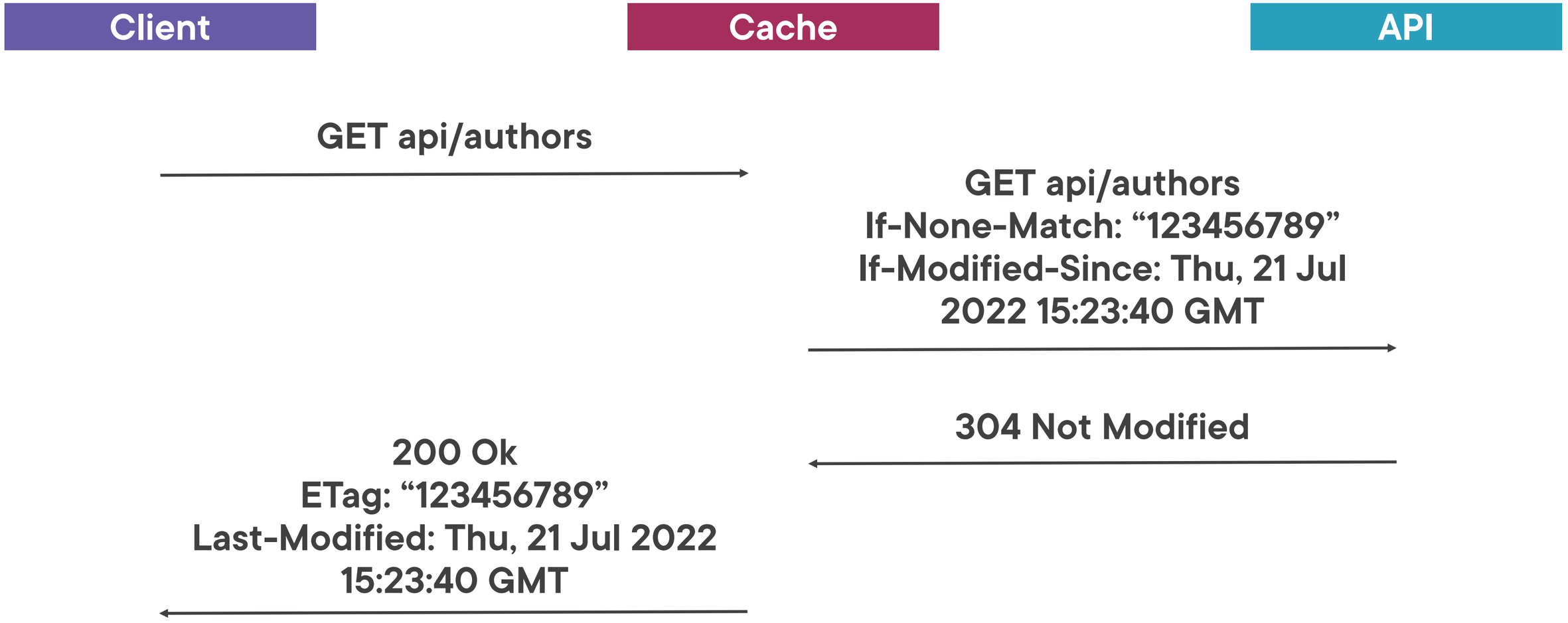
Validation Model



Validation Model



Validation Model



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

Exploring 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

Summary

Each response must state whether or not it can be cached

Cache types

- Private or shared
- Depending on the type, a model can reduce bandwidth requirements and/or network roundtrips

Summary

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

Summary

Implementation

- Control the `Cache-Control` header with the `[ResponseCache]` attribute
- Store responses with the response caching middleware

Up Next:

Supporting HTTP Cache for ASP.NET Core APIs