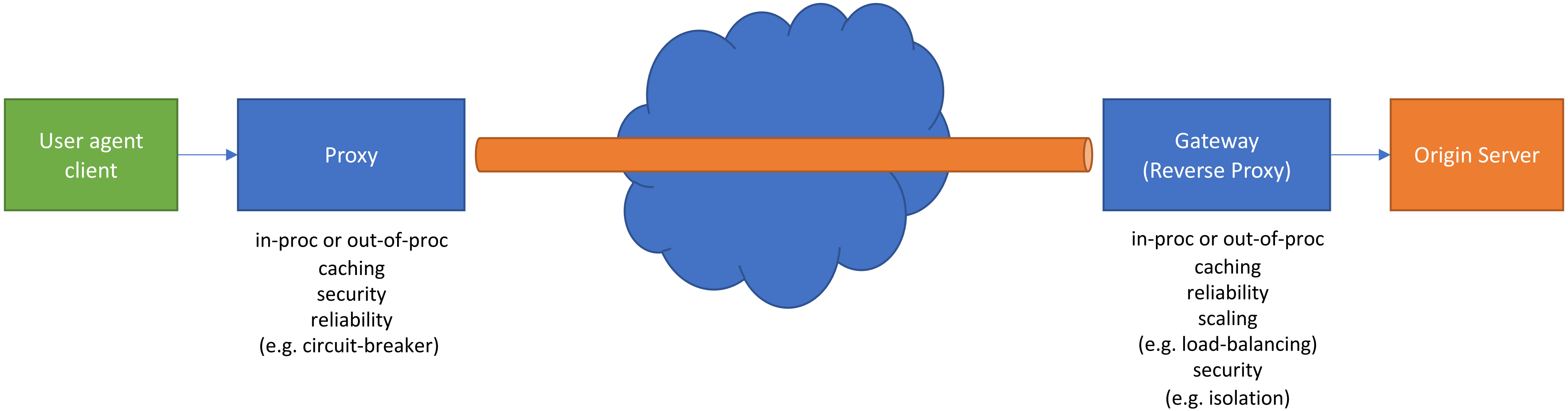
# HTTP Caching

Pedro Félix
April 2020



User agent client

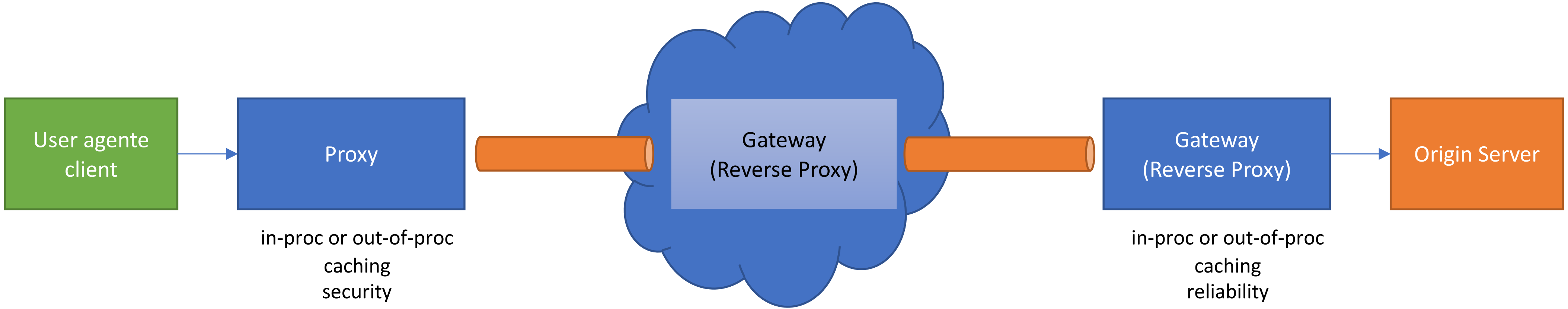
Proxy

in-proc or out-of-proc
caching
security
reliability
(e.g. circuit-breaker)

Gateway (Reverse Proxy)

in-proc or out-of-proc
caching
reliability
scaling
(e.g. load-balancing)
security
(e.g. isolation)

Origin Server



User agente client

Proxy

Gateway (Reverse Proxy)

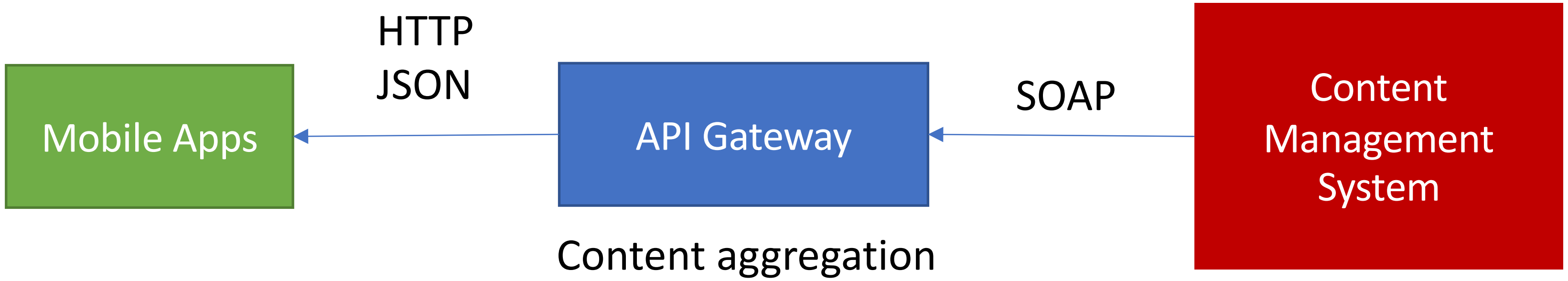
Gateway (Reverse Proxy)

Origin Server

in-proc or out-of-proc
caching
security
reliability
(e.g. circuit-breaker)

in-proc or out-of-proc
caching
reliability
scaling
(e.g. load-balancing)
security
(e.g. isolation)

# Caching: a use case



Mobile Apps

HTTP
JSON

API Gateway

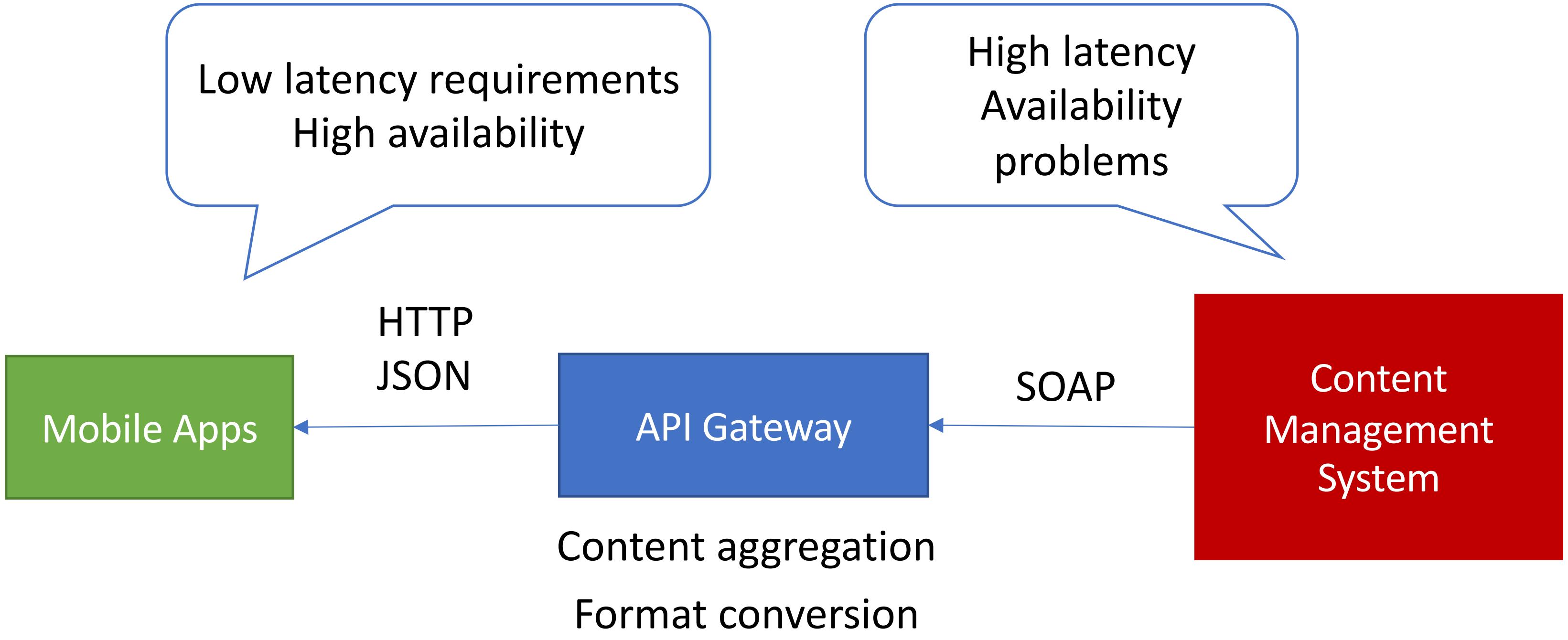
SOAP

Content Management System

Content aggregation

Format conversion

# Caching: a use case



Low latency requirements
High availability

High latency
Availability
problems

Mobile Apps

HTTP
JSON

API Gateway

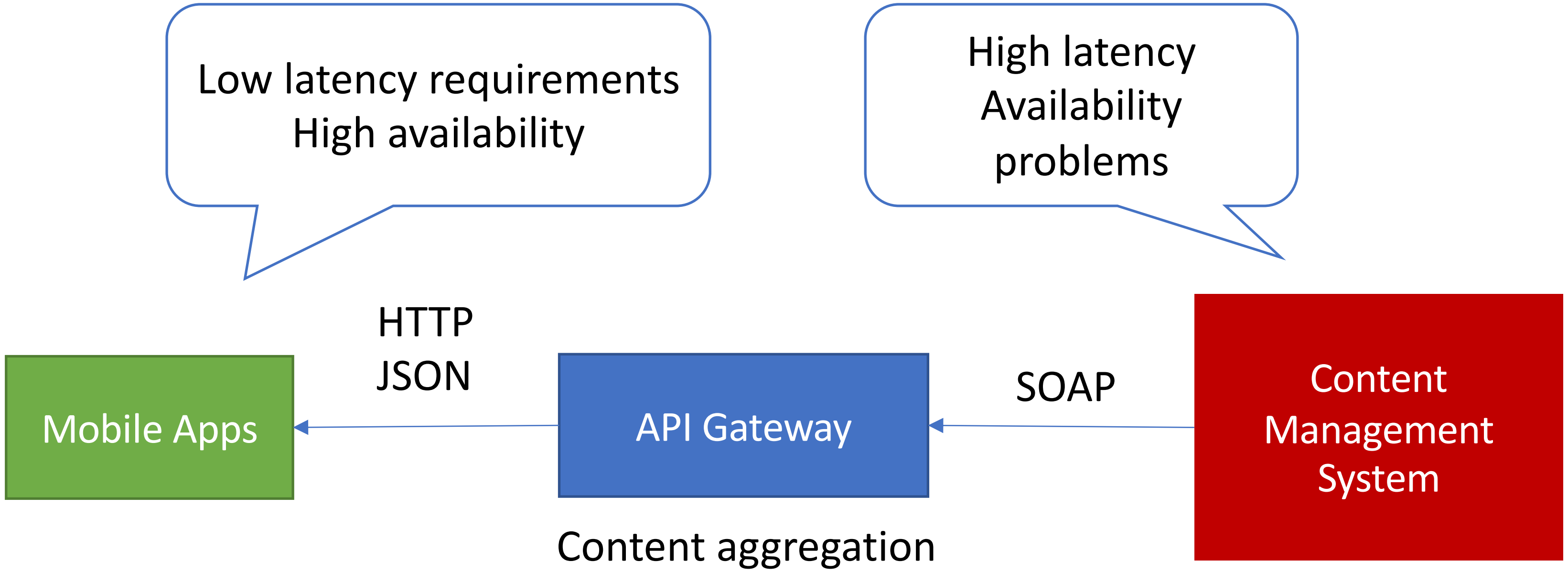
SOAP

Content
Management
System

Content aggregation

Format conversion

# Caching: a use case



Content aggregation

Format conversion

**Caching**

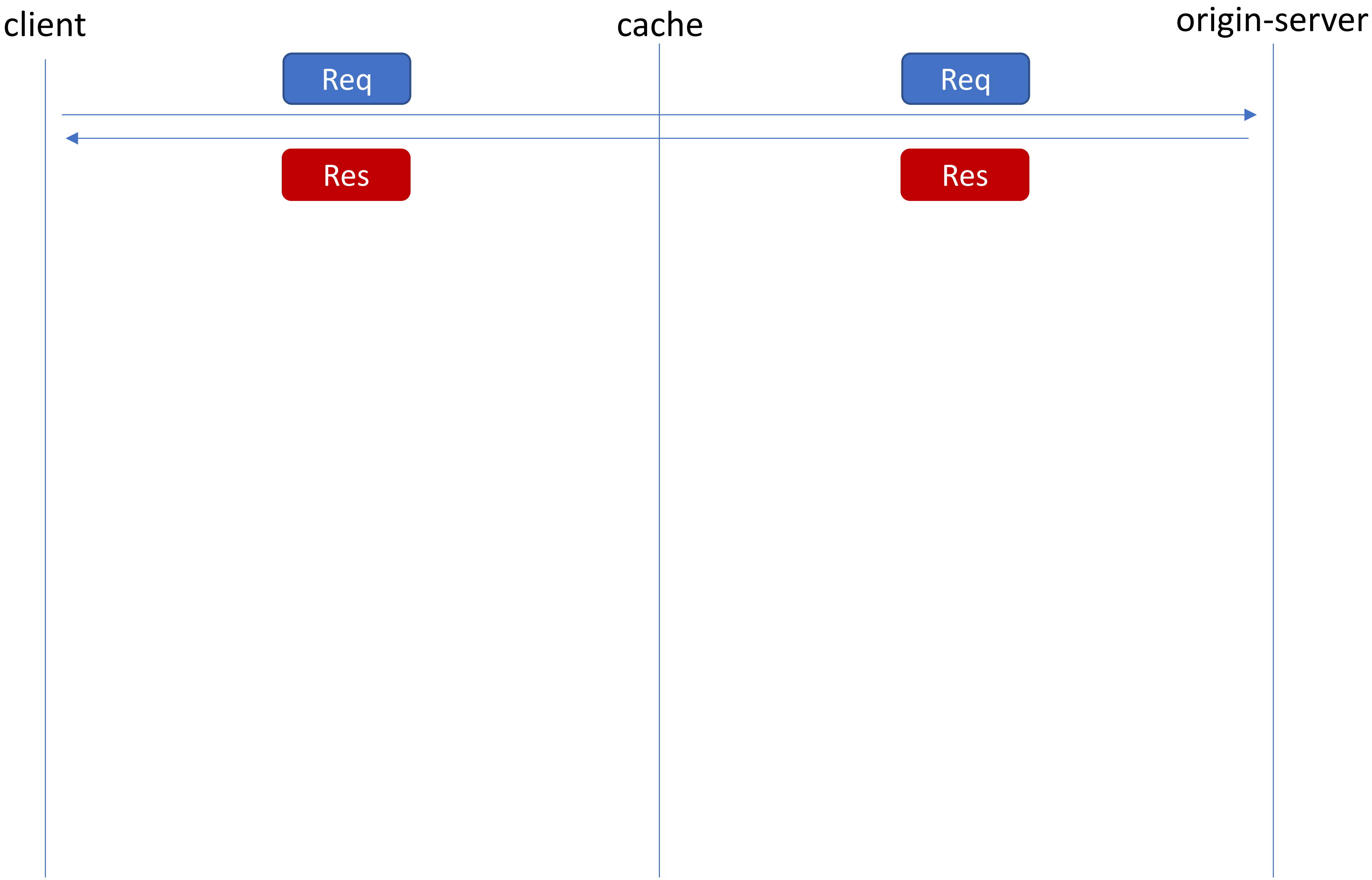
# From RFC 7234 (**bolds** are mine)

- *“An HTTP cache is a **local store of response messages** and the subsystem that controls **storage**, **retrieval**, and **deletion** of messages in it. A **cache stores cacheable responses** in order to **reduce the response time** and **network bandwidth consumption** on future, equivalent requests. **Any client or server MAY employ a cache**, though a cache **cannot** be used by a server that is **acting as a tunnel**.”*
- *“A **shared cache** is a cache that stores responses to be reused by more than one user; shared caches are usually (but not always) **deployed as a part of an intermediary**. A **private cache**, in contrast, is dedicated to a single user; often, they are deployed as a component of a **user agent**.”*
- *“A stored response is considered **"fresh"**, as defined in Section 4.2, if the response **can be reused without "validation"** (**checking with the origin server to see if the cached response remains valid for this request**). A fresh response can therefore reduce both latency and network overhead each time it is reused.”*
- *“ When a cached response is not fresh, it might still be reusable if it can be freshened by validation (Section 4.3) or if the origin is unavailable (Section 4.2.4).”*
- *“A **fresh** response is one whose **age** has not yet exceeded its **freshness lifetime**. Conversely, a **stale** response is one where it has.”*

# From RFC 7234 (**bolds** are mine)

- *“Each cache entry consists of a **cache key** and **one or more HTTP responses** corresponding to **prior requests** that used the **same key.**”*
- *“The most common form of cache entry is a **successful result of a retrieval request**: i.e., a 200 (OK) response to a GET request, which contains a representation of the resource identified by the request target (Section 4.3.1 of [RFC7231]). However, it is also **possible to cache permanent redirects**, **negative results** (e.g., 404 (Not Found))”*
- *“The **primary cache key** consists of the **request method** and **target URI**.”*
- *“If a request target is **subject to content negotiation**, its cache entry might consist of **multiple stored responses**, each differentiated by a secondary key for the **values of the original request's selecting header fields**”*.

# Caching



client

cache

origin-server

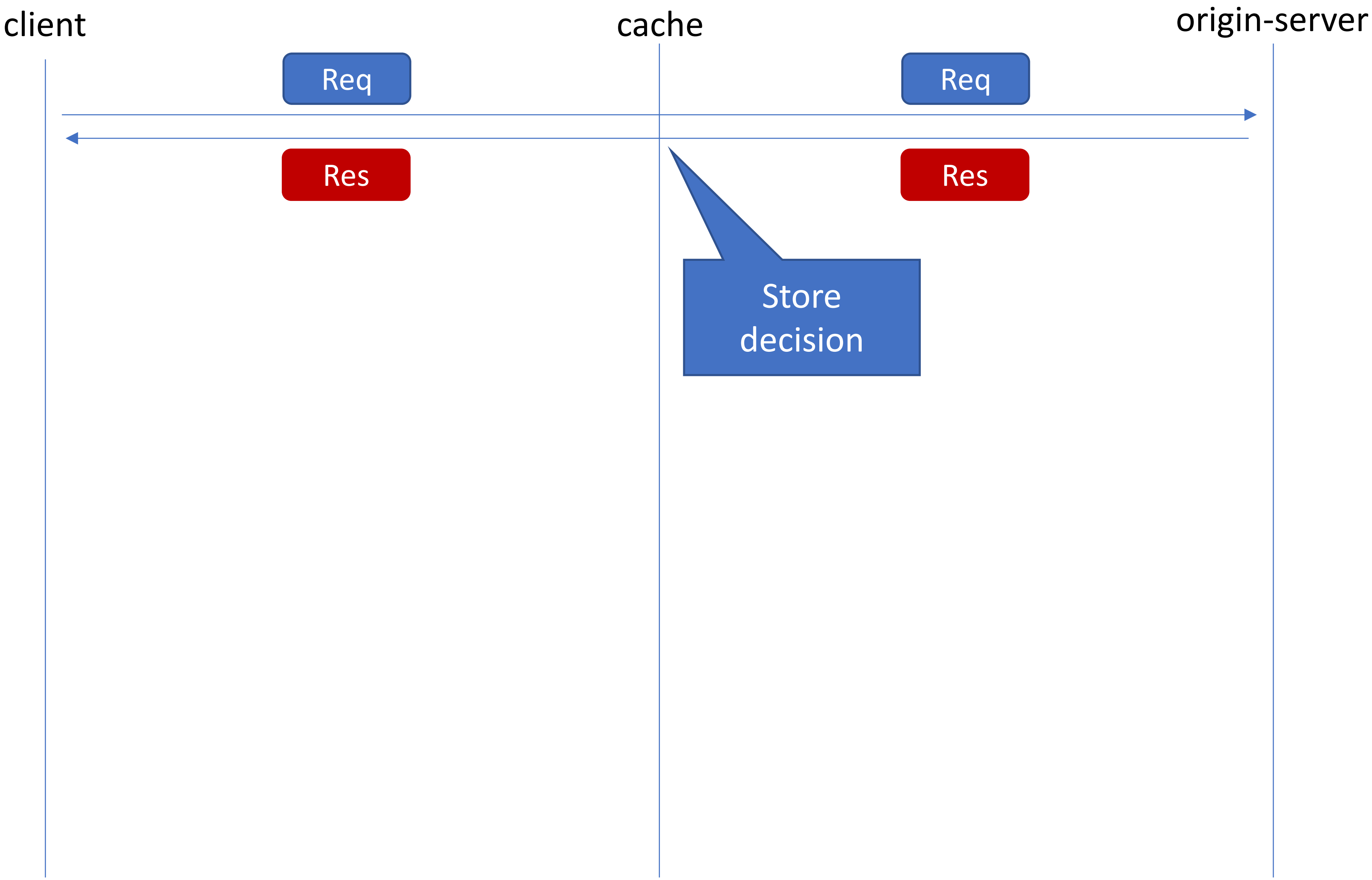
Req

Req

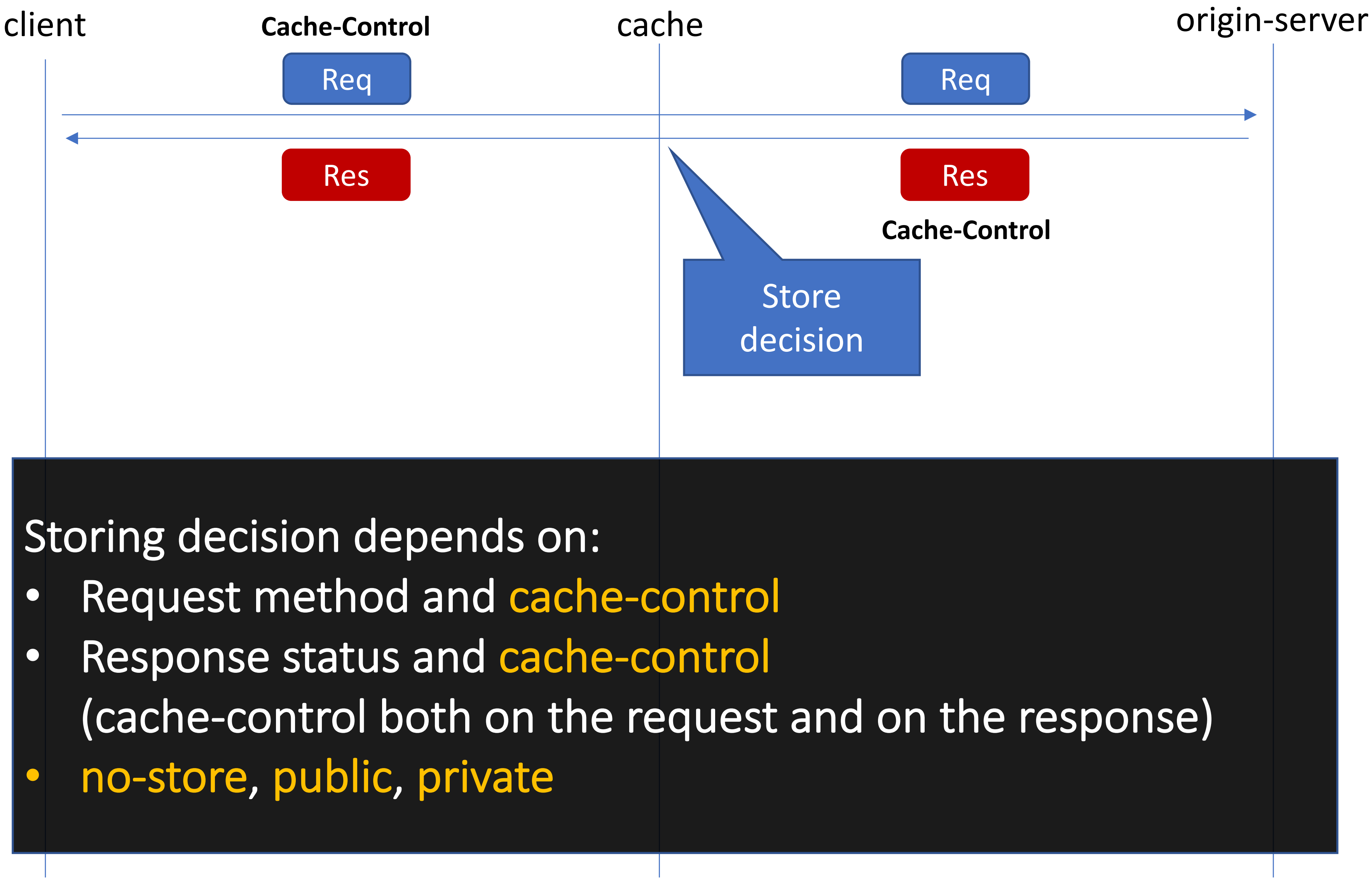
Res

Res

# Caching



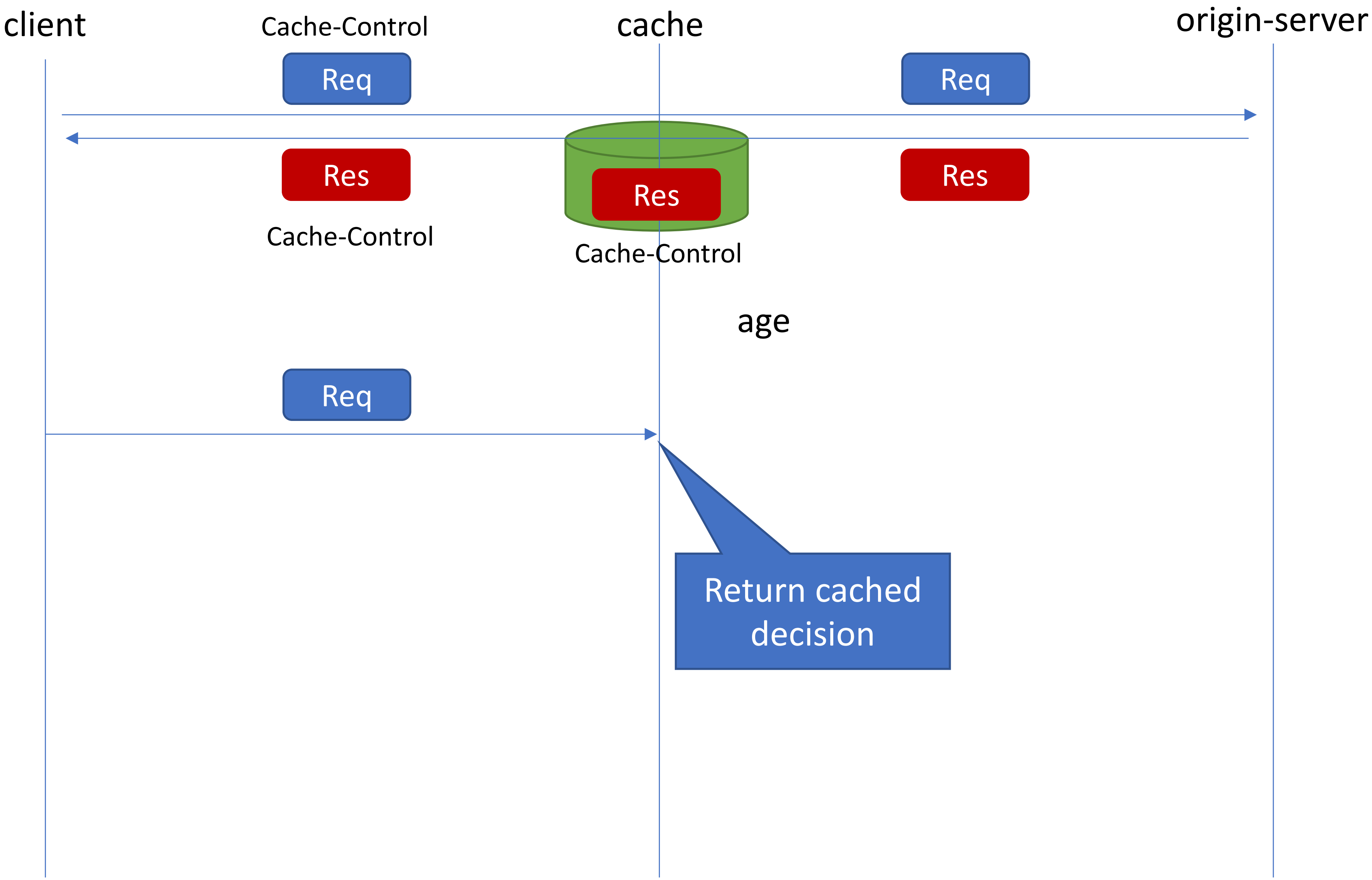
# Caching



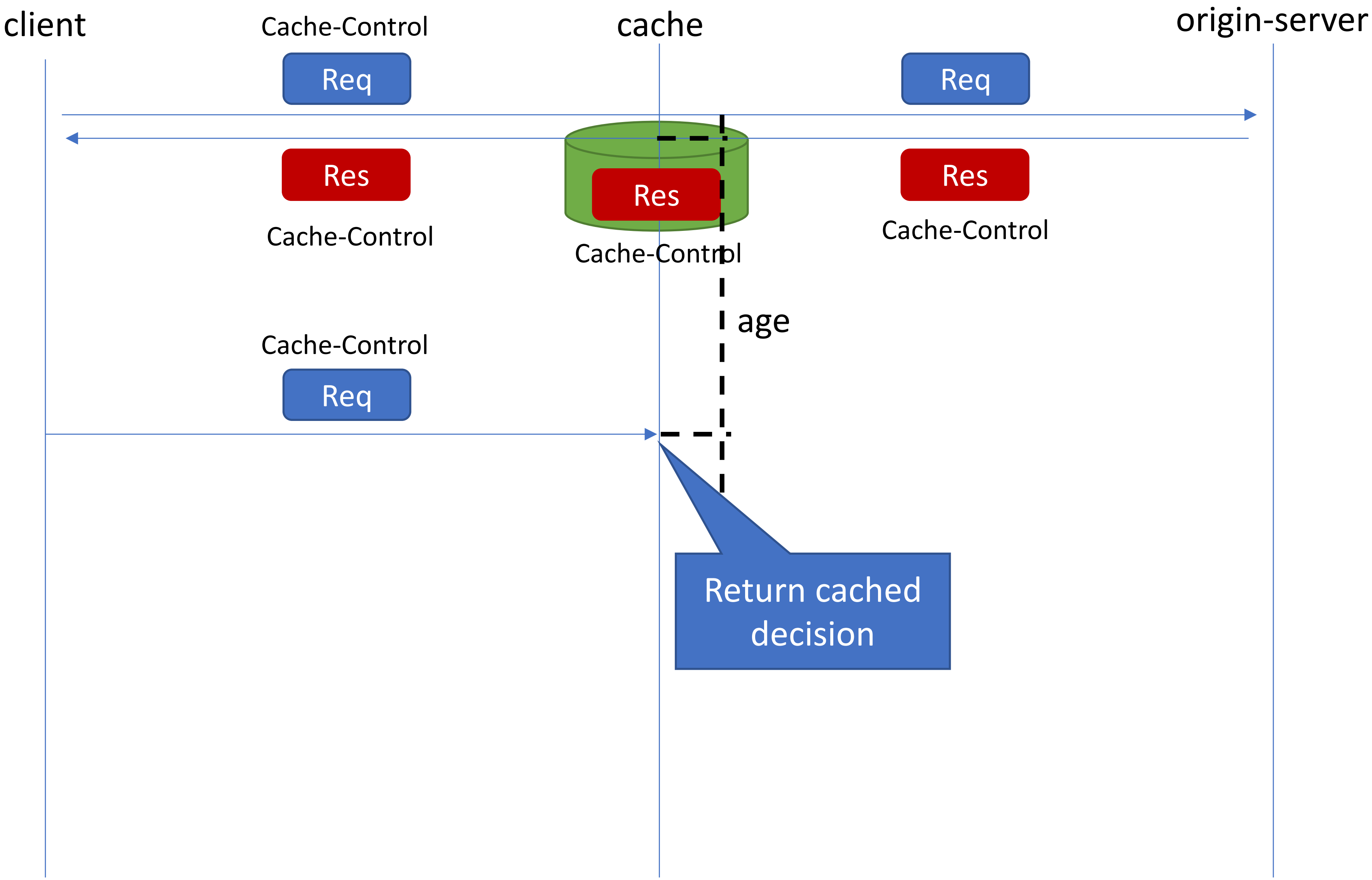
Storing decision depends on:

- Request method and cache-control
- Response status and cache-control
  (cache-control both on the request and on the response)
- no-store, public, private

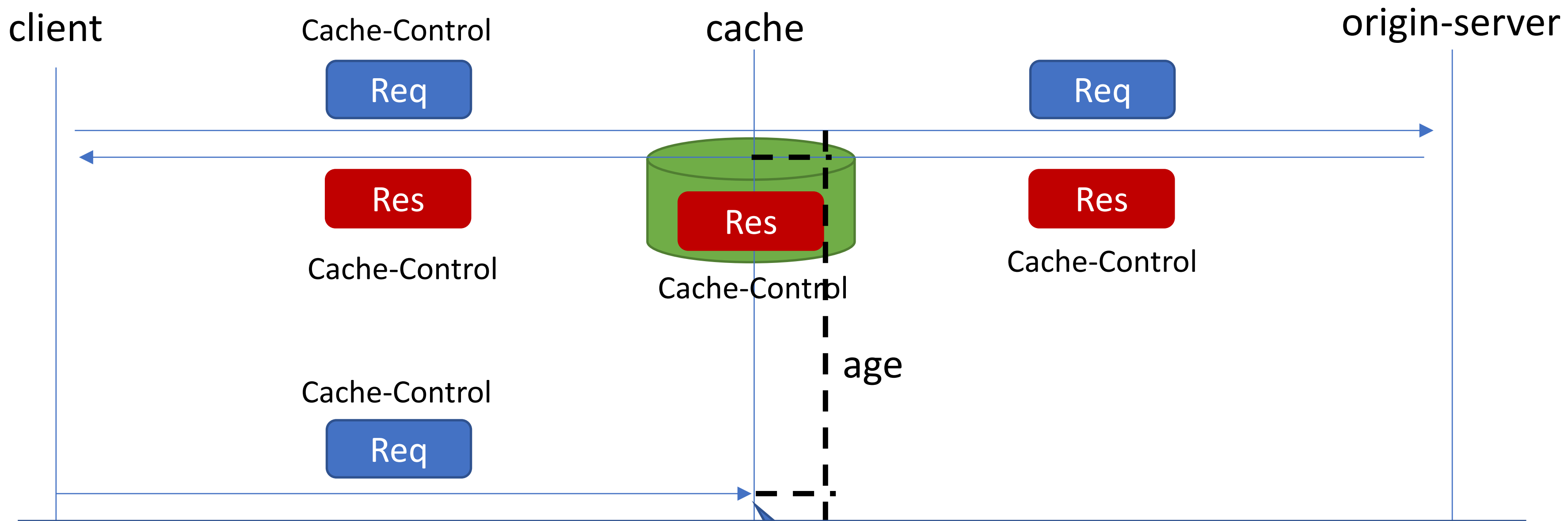
# Caching


client
Cache-Control
cache
origin-server
Req
Req
Res
Res
Res
Cache-Control
Cache-Control
age
Req
Return cached decision

# Caching


client
Cache-Control
Req
cache
origin-server
Req
Res
Res
Res
Cache-Control
Cache-Control
Cache-Control
age
Cache-Control
Req
Return cached decision

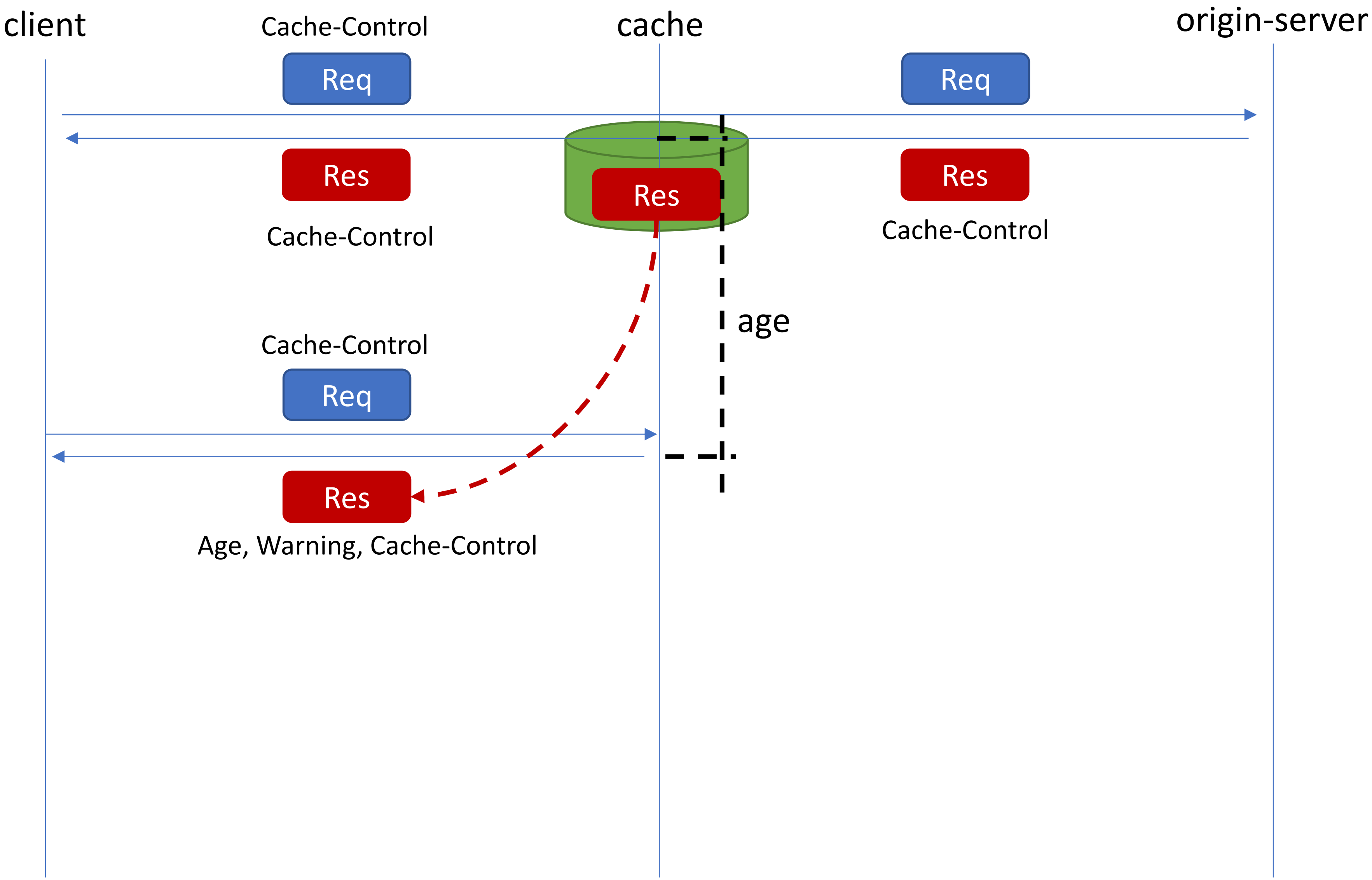
# Caching



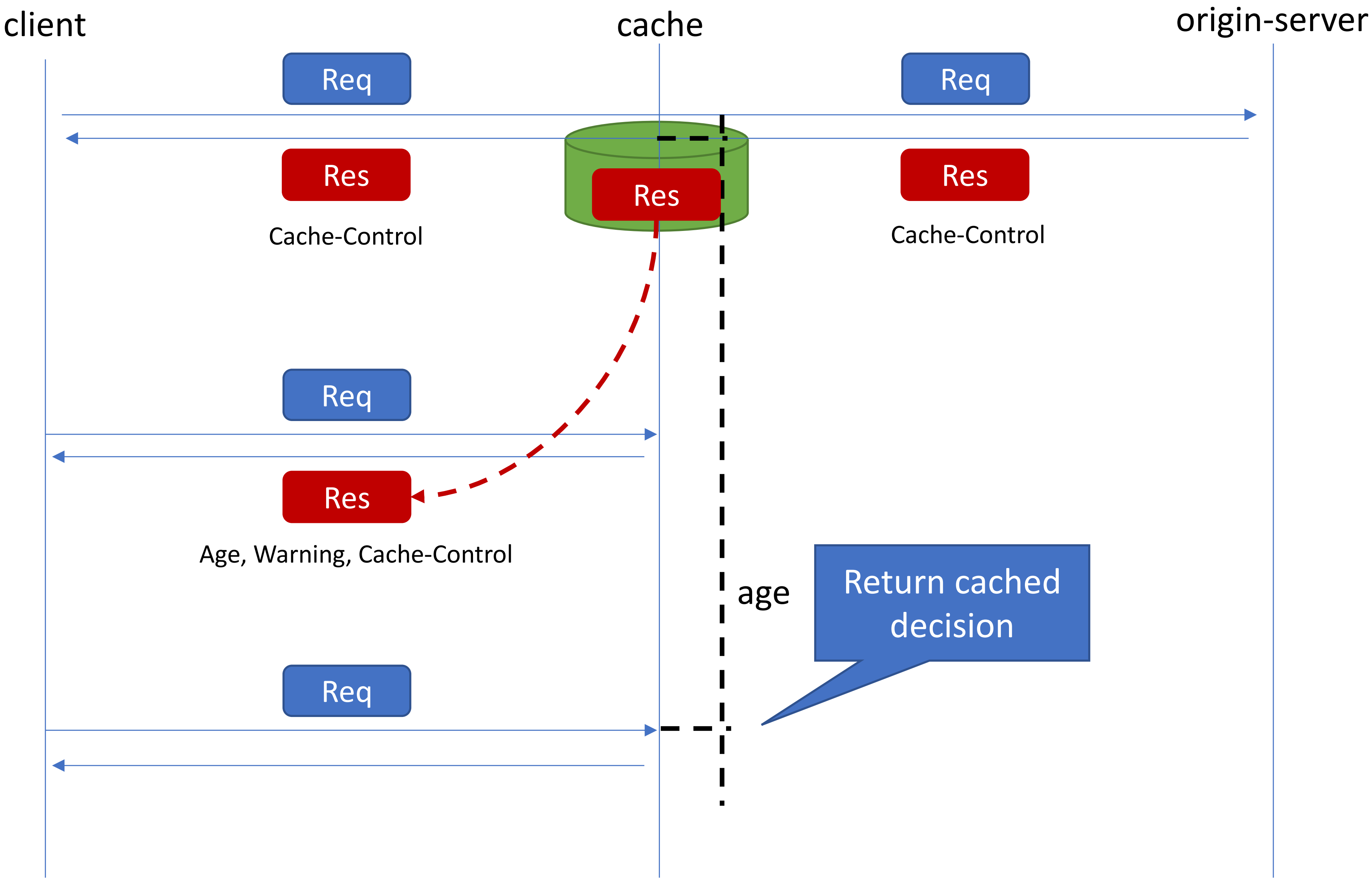
Return cached decision depends on:

- Request method and cache-control
- Stored Response status and cache-control
  (cache-control both on the request and on the response)
- no-cache, max-age, max-stale

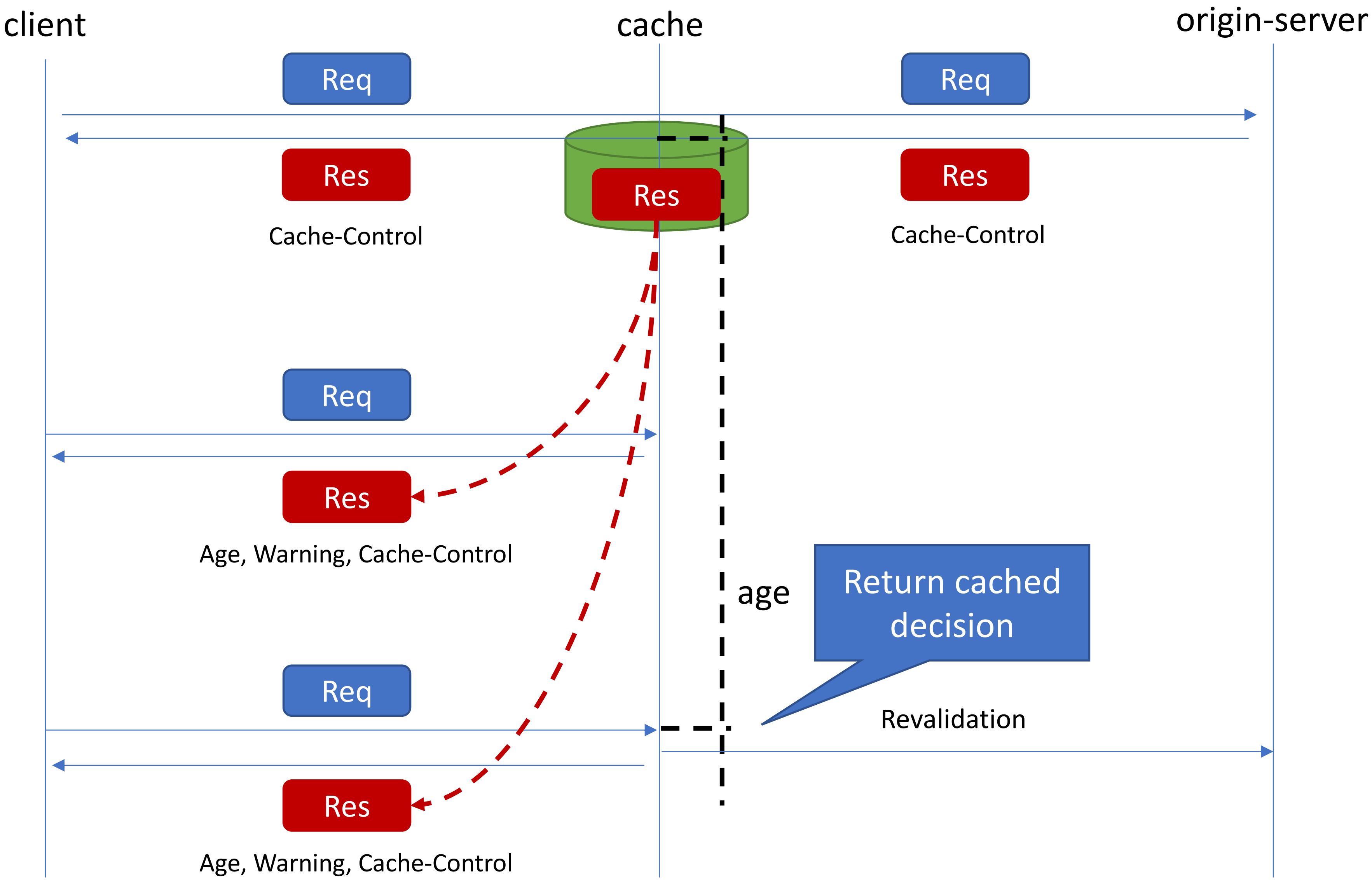
# Caching


client
Cache-Control
cache
origin-server
Req
Req
Res
Res
Res
Cache-Control
Cache-Control
age
Cache-Control
Req
Res
Age, Warning, Cache-Control

# Caching


client
cache
origin-server
Req
Req
Res
Res
Res
Cache-Control
Cache-Control
Req
Res
Age, Warning, Cache-Control
age
Return cached decision
Req

# Caching


client
cache
origin-server
Req
Req
Res
Res
Res
Cache-Control
Cache-Control
Req
Res
Age, Warning, Cache-Control
age
Return cached decision
Revalidation
Req
Res
Age, Warning, Cache-Control

# Caching

- Multiple levels
  - Client (e.g. browser cache) - private
  - Proxy – private or public
  - Reverse-proxy – public
- Cache contains (key, stored response)
  - Primary key is (request method, target URI)
  - Secondary key is a list of headers
  - **Age**
  - Fresh state
  - Stale state

# Caching

- **Age** response header

- **Cache-Control** header
  - Request
    - **no-cache**, **no-store**
    - **max-age**, **max-stale**, **min-fresh**
  - Response
    - **public**, **private**
    - **no-cache**, **no-store**
    - **must-revalidate**
    - **max-age**, **s-maxage**

# Caching

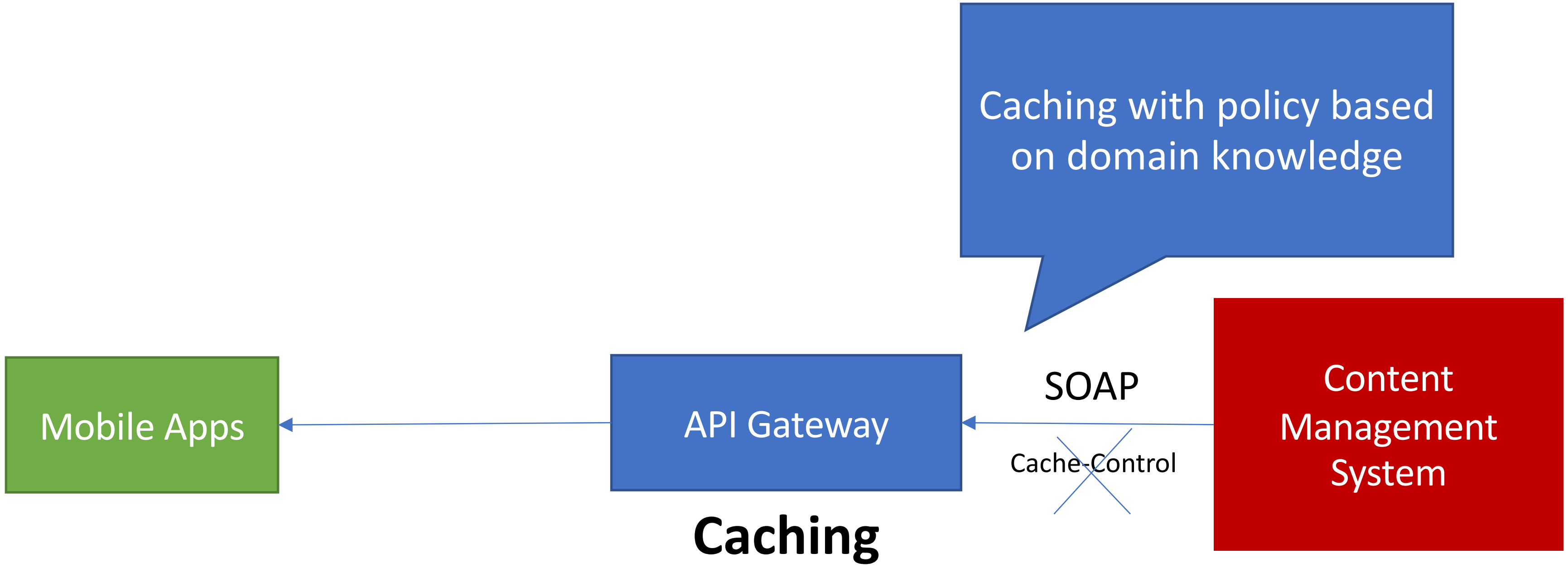
- **Warning** response header
  - **110 Response is Stale**
  - **111 Revalidation Failed**
  - **112 Heuristic Operation**

- Heuristics
  - Some HTTP methods are cacheable by default
  - Max-age is determined by cache, using heuristics
  - E.g. 10% of time since **Last-Modified**

# Caching extensions

- Two response cache control extensions
- **stale-while-revalidate** = {delta}
  - GET with cached stale response
    - May serve stale if age less than max-age + delta seconds
    - Start asynchronous revalidation
  - Revalidation does not block request
- **stale-if-error** = {delta}
  - GET with cached stale response
    - Start a synchronous revalidation
    - May serve stale if age less that max-age + delta seconds

# Returning to our use case…



Caching with policy based on domain knowledge

Mobile Apps

API Gateway

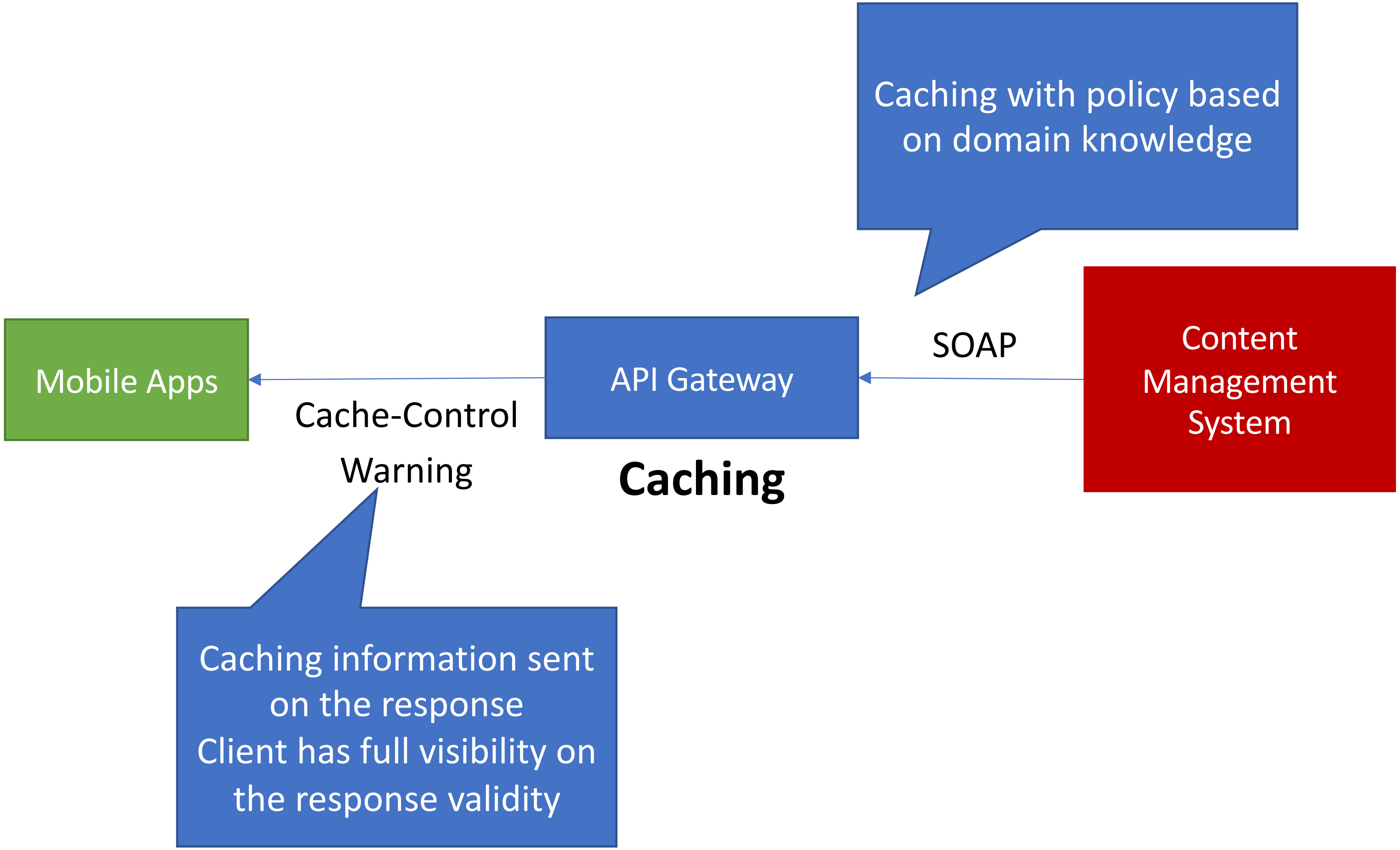
**Caching**

SOAP

~~Cache-Control~~

Content Management System

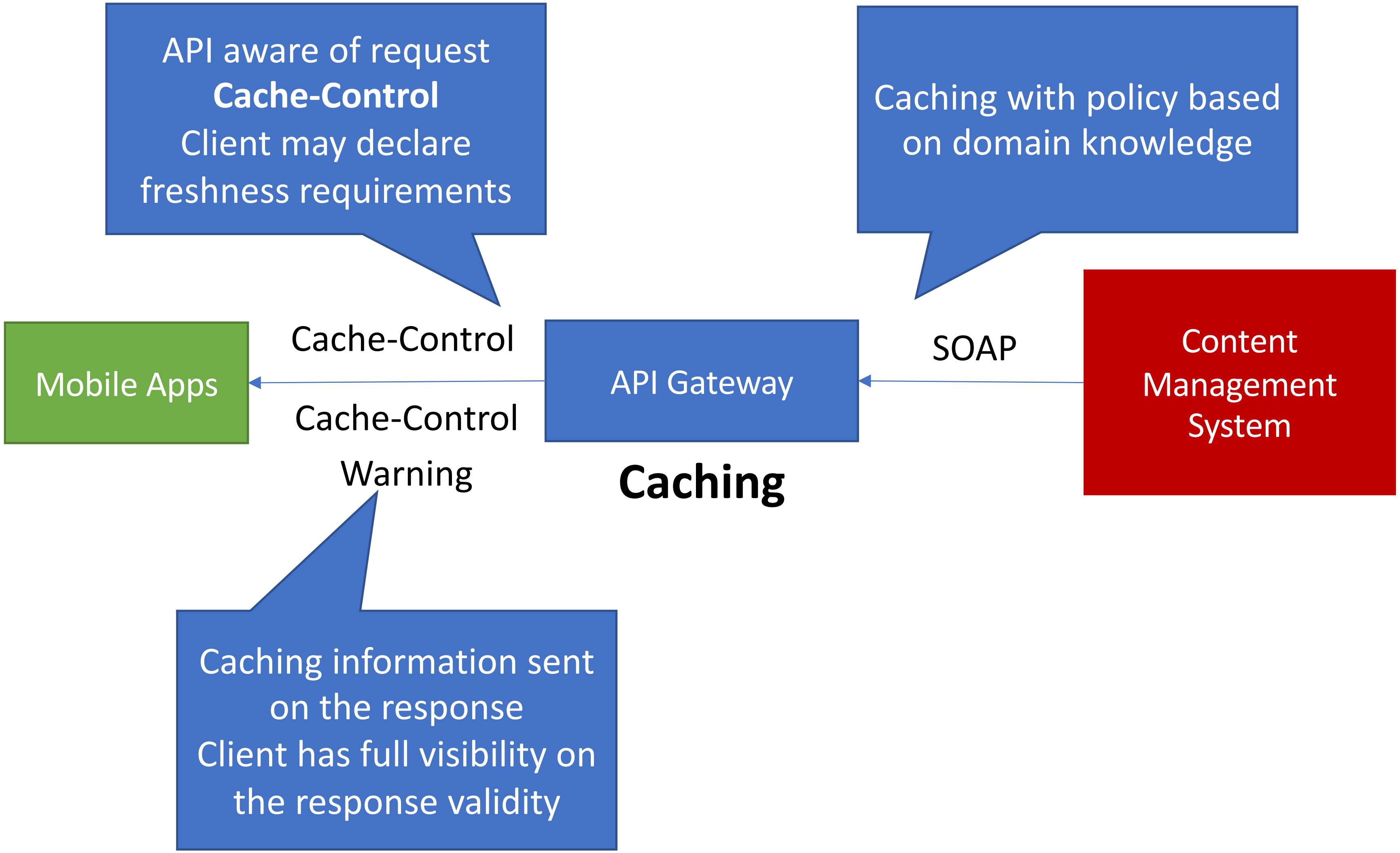
# Returning to our use case…



Caching with policy based on domain knowledge

Caching information sent on the response
Client has full visibility on the response validity

# Returning to our use case…



API aware of request **Cache-Control**
Client may declare freshness requirements

Caching with policy based on domain knowledge

Mobile Apps

Cache-Control

Cache-Control
Warning

API Gateway

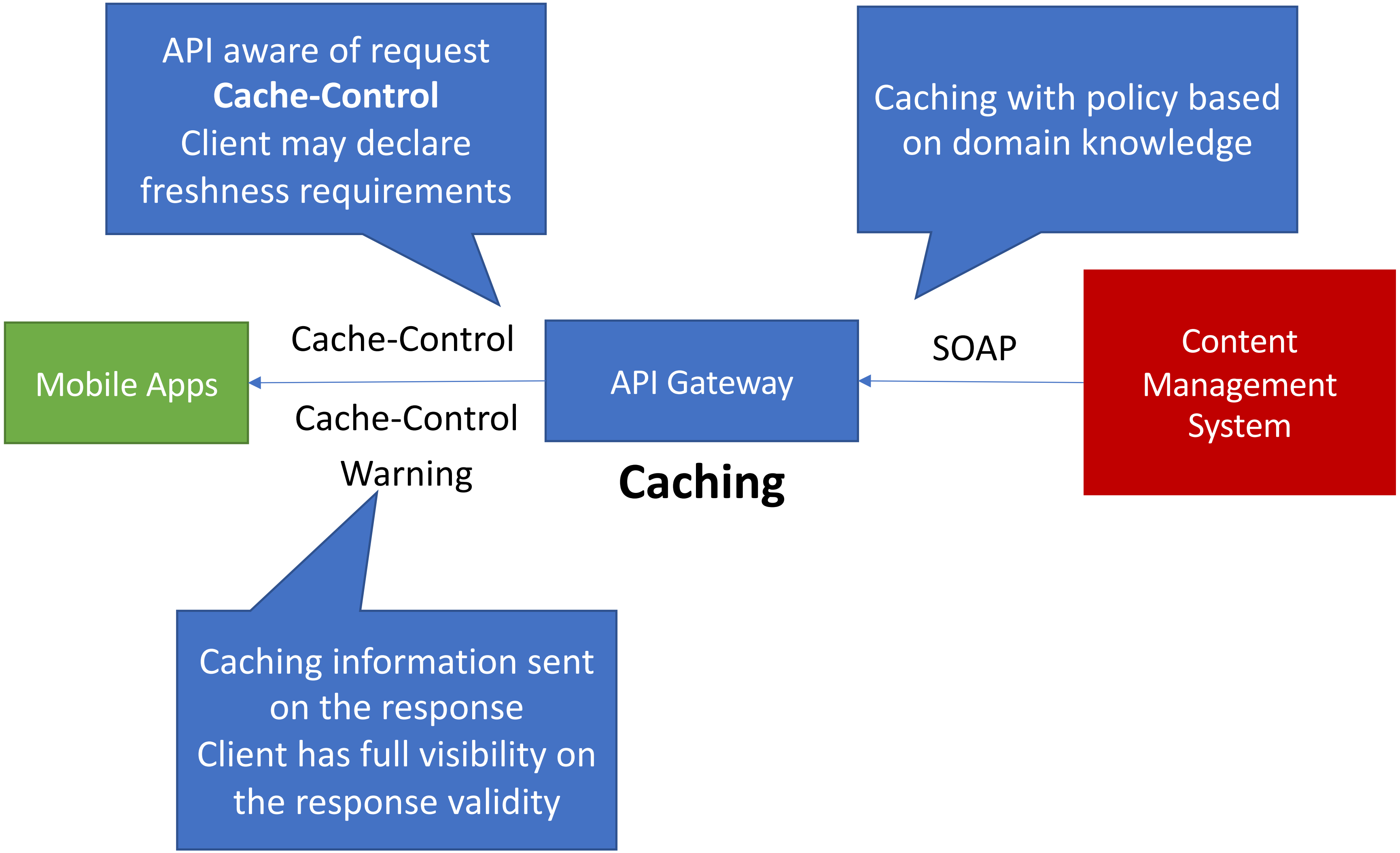
**Caching**

SOAP

Content Management System

Caching information sent on the response
Client has full visibility on the response validity

# Returning to our use case…

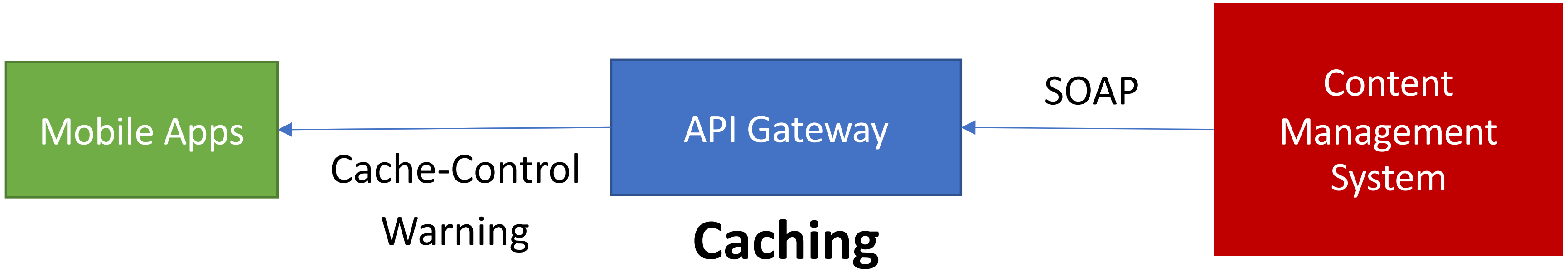


API aware of request
**Cache-Control**
Client may declare freshness requirements
Caching with policy based on domain knowledge
Cache-Control
Mobile Apps
Cache-Control
Warning
API Gateway
**Caching**
SOAP
Content Management System
Caching information sent on the response
Client has full visibility on the response validity

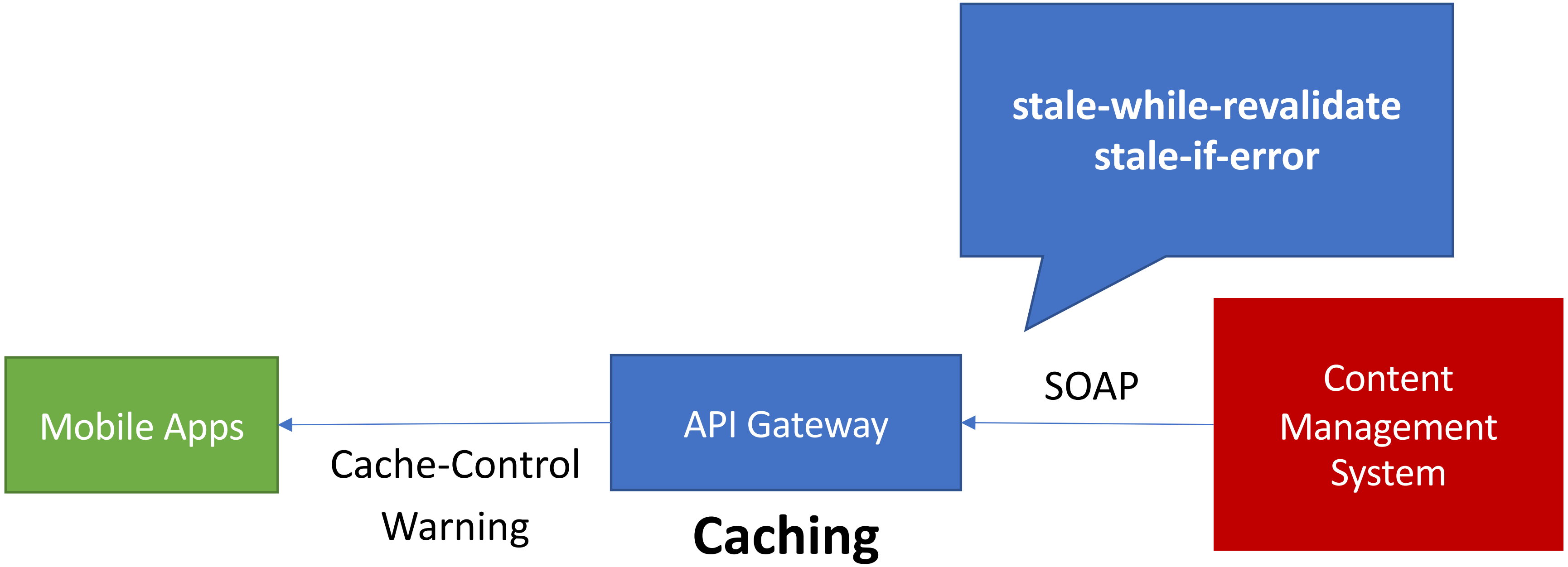
# Returning to our use case...

Still some problems

- Synchronous revalidation may incur high-latency
- Revalidation failures due to back-end unavailability



# Returning to our use case…



stale-while-revalidate
stale-if-error

Mobile Apps

Cache-Control

Warning

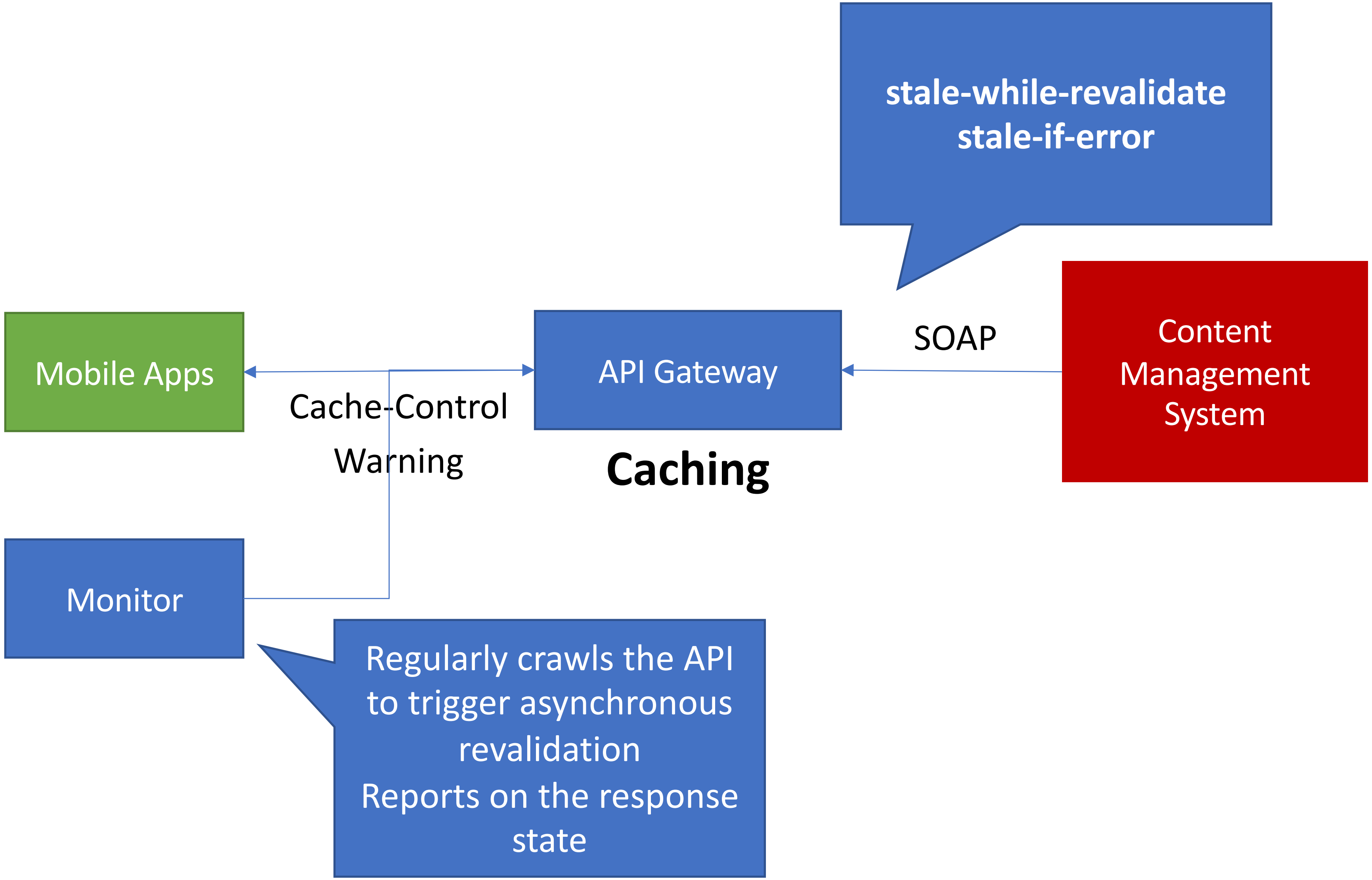
API Gateway

**Caching**

SOAP

Content Management System

# Returning to our use case…



stale-while-revalidate
stale-if-error

Mobile Apps

Cache-Control

Warning

API Gateway

Caching

SOAP

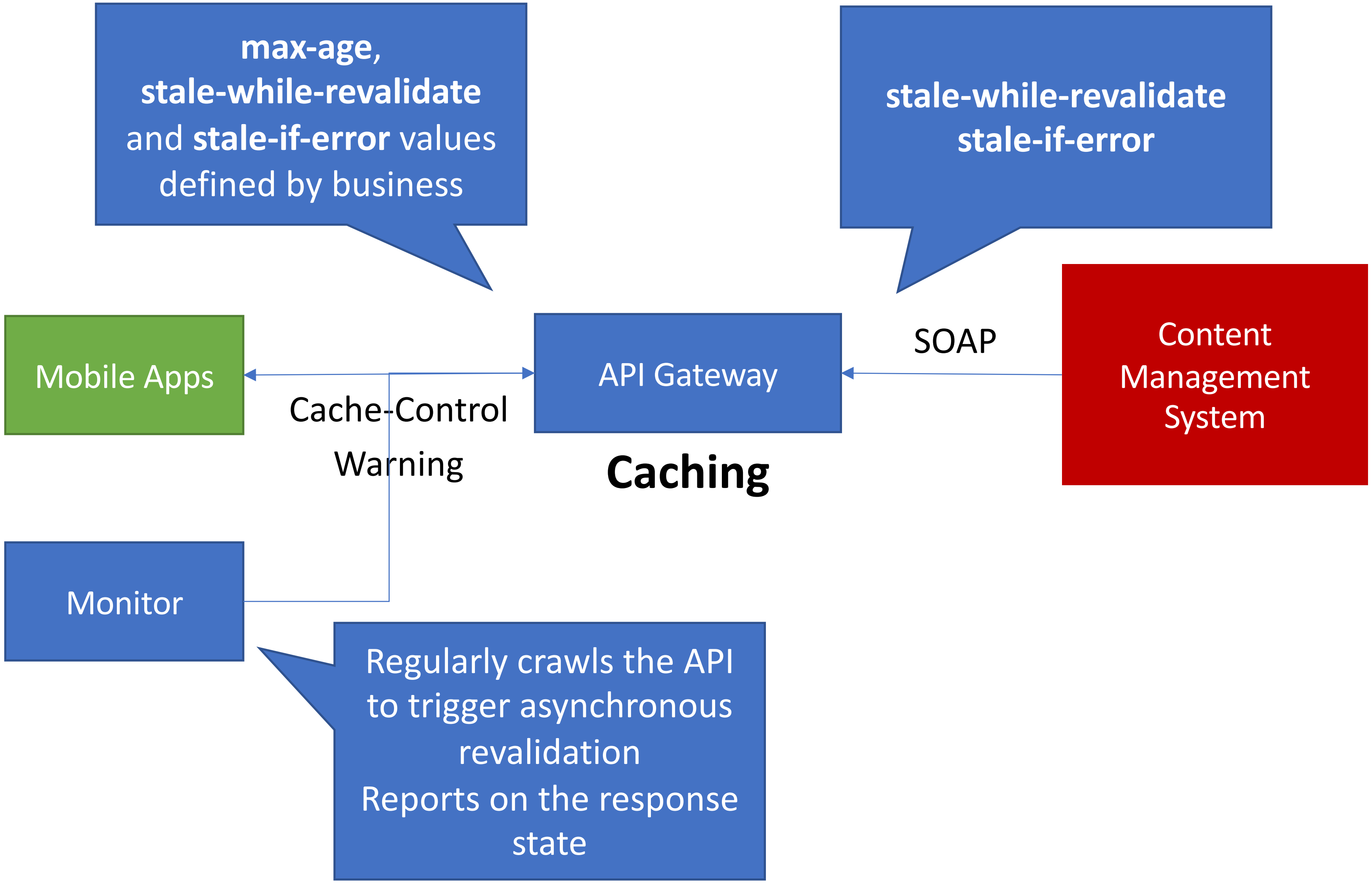
Content Management System

Monitor

Regularly crawls the API to trigger asynchronous revalidation

Reports on the response state

# Returning to our use case…

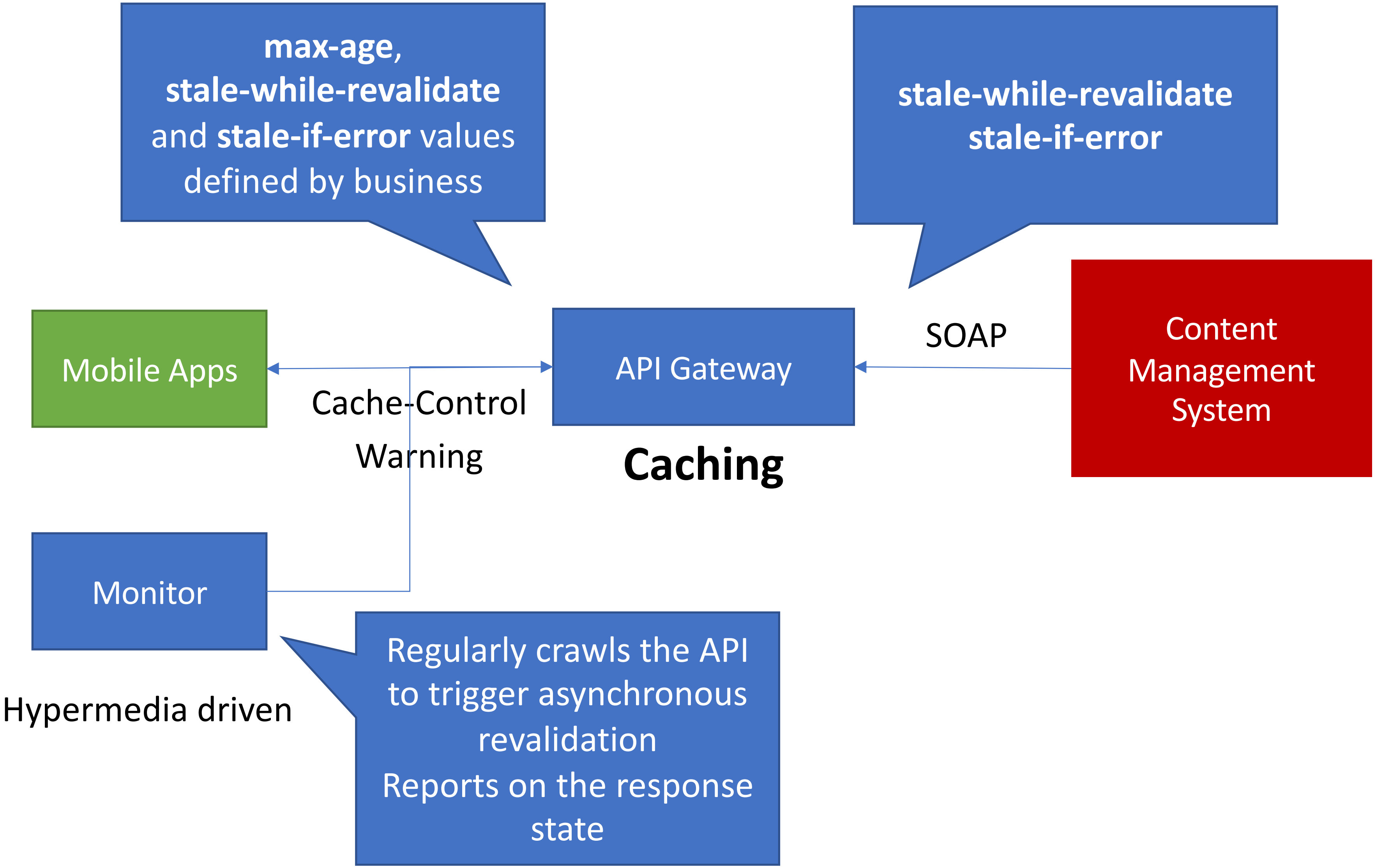


**max-age,**
**stale-while-revalidate**
and **stale-if-error** values
defined by business

**stale-while-revalidate**
**stale-if-error**

Regularly crawls the API
to trigger asynchronous
revalidation
Reports on the response
state

# Returning to our use case…



**max-age,**
**stale-while-revalidate**
and **stale-if-error** values
defined by business

**stale-while-revalidate**
**stale-if-error**

Mobile Apps

Cache-Control

Warning

API Gateway

**Caching**

SOAP

Content Management System

Monitor

Hypermedia driven

Regularly crawls the API to trigger asynchronous revalidation

Reports on the response state