Web infrastructures

https://github.com/heig-vd-dai-course

Web · PDF

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Based on the original course by O. Liechti and J. Ehrensberger.

Objectives

- Understand the concepts of web infrastructures
- Understand how HTTP features can help to build web infrastructures
- Understand the concepts of a reverse proxy
- Understand the concepts of load balancing
- Understand the concepts of caching

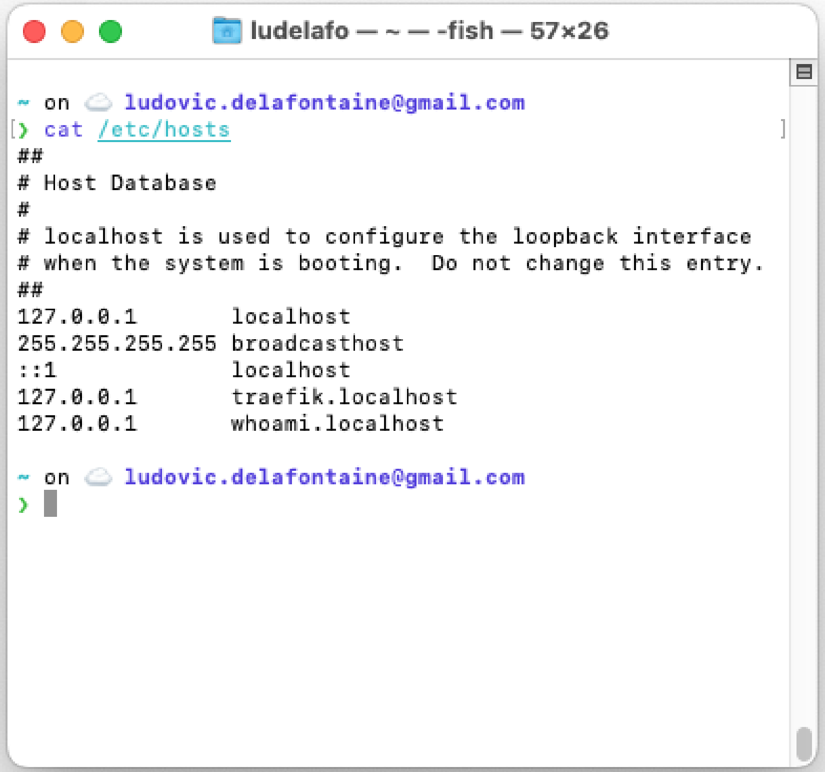
Prepare and setup your environment

More details for this section in the course material. You can find other resources and alternatives as well.

Access your `hosts` file

File to map hostnames to IP addresses, just as DNS does, but only for your computer.

- Windows: `%WinDir%\System32\Drivers\Etc\Hosts`
- Linux and macOS: `/etc/hosts`



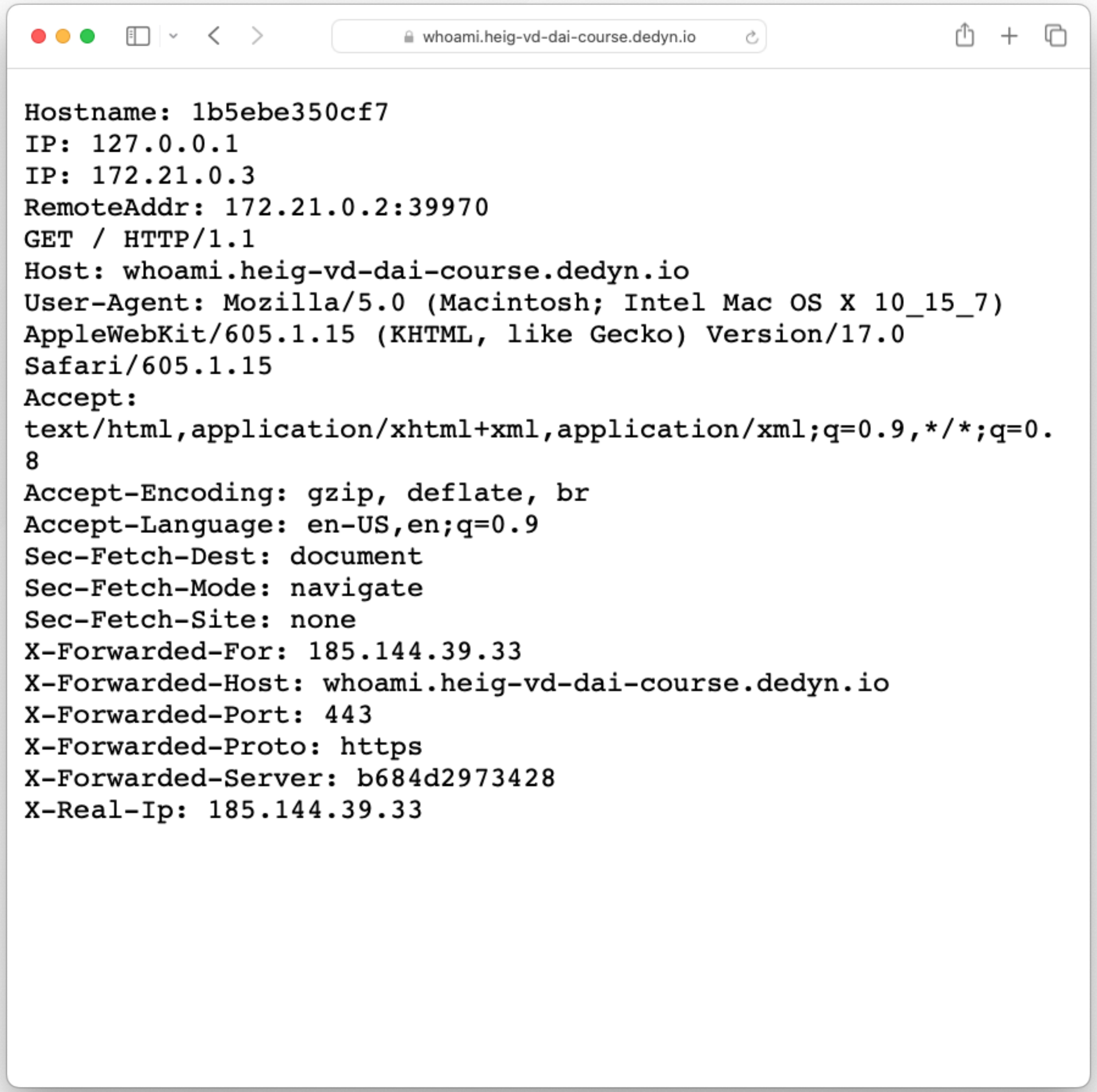
```
~ on ☁ ludovic.delafontaine@gmail.com
❯ cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
127.0.0.1       traefik.localhost
127.0.0.1       whoami.localhost

~ on ☁ ludovic.delafontaine@gmail.com
❯
```

Traefik

- An open source reverse proxy (more on this later)
- Works well with Docker Compose and Kubernetes
- Issue and renew Let's Encrypt (HTTPS) certificates automatically
- Easy to use with Docker Compose labels

whoami

- A tiny Go webserver that prints os information and HTTP request to output
- Used to demonstrate the use of HTTP with a reverse proxy and a load balancer



```
Hostname: 1b5ebe350cf7
IP: 127.0.0.1
IP: 172.21.0.3
RemoteAddr: 172.21.0.2:39970
GET / HTTP/1.1
Host: whoami.heig-vd-dai-course.dedyn.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.0
Safari/605.1.15
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
X-Forwarded-For: 185.144.39.33
X-Forwarded-Host: whoami.heig-vd-dai-course.dedyn.io
X-Forwarded-Port: 443
X-Forwarded-Proto: https
X-Forwarded-Server: b684d2973428
X-Real-Ip: 185.144.39.33
```

Functional and non-functional requirements

More details for this section in the course material. You can find other resources and alternatives as well.

Functional and non-functional requirements

- Requirements used to **define the scope of a system**.
- An **abstract representation** of the system that will be implemented.
- Can help to **define the architecture of the system**.
- **Functional requirements**: features that a system must have to satisfy the needs of its users. It is the "what" of a system.
- **Non-functional requirements**: constraints on the system. It is the "how" of a system.

Examples of **functional requirements**:

- **User management**: Users must be able to register, login, logout, etc.
- **Product management**: Users must be able to create, read, update, delete products, etc.
- **Order management**: Users must be able to create, read, update, delete orders, etc.
- **Payment management**: Users must be able to pay for their orders, etc.

Examples of **non-functional requirements**:

- **Response time**: Time between a request and a response (end user)
- **Throughput**: Number of requests/interval (service provider)
- **Scalability**: Property of a system to handle a varying amount of work
- **Availability**: Percentage of time that the system provides a satisfactory service
- **Maintainability**: How easily the system can be managed
- **Security**: Confidentiality, authentication, authorization, etc.
- ...and many, many more

Web infrastructure definition

More details for this section in the course material. You can find other resources and alternatives as well.

Web infrastructure definition

Software and hardware components that are necessary to support:

- the **development**
- the **deployment**
- the **management**

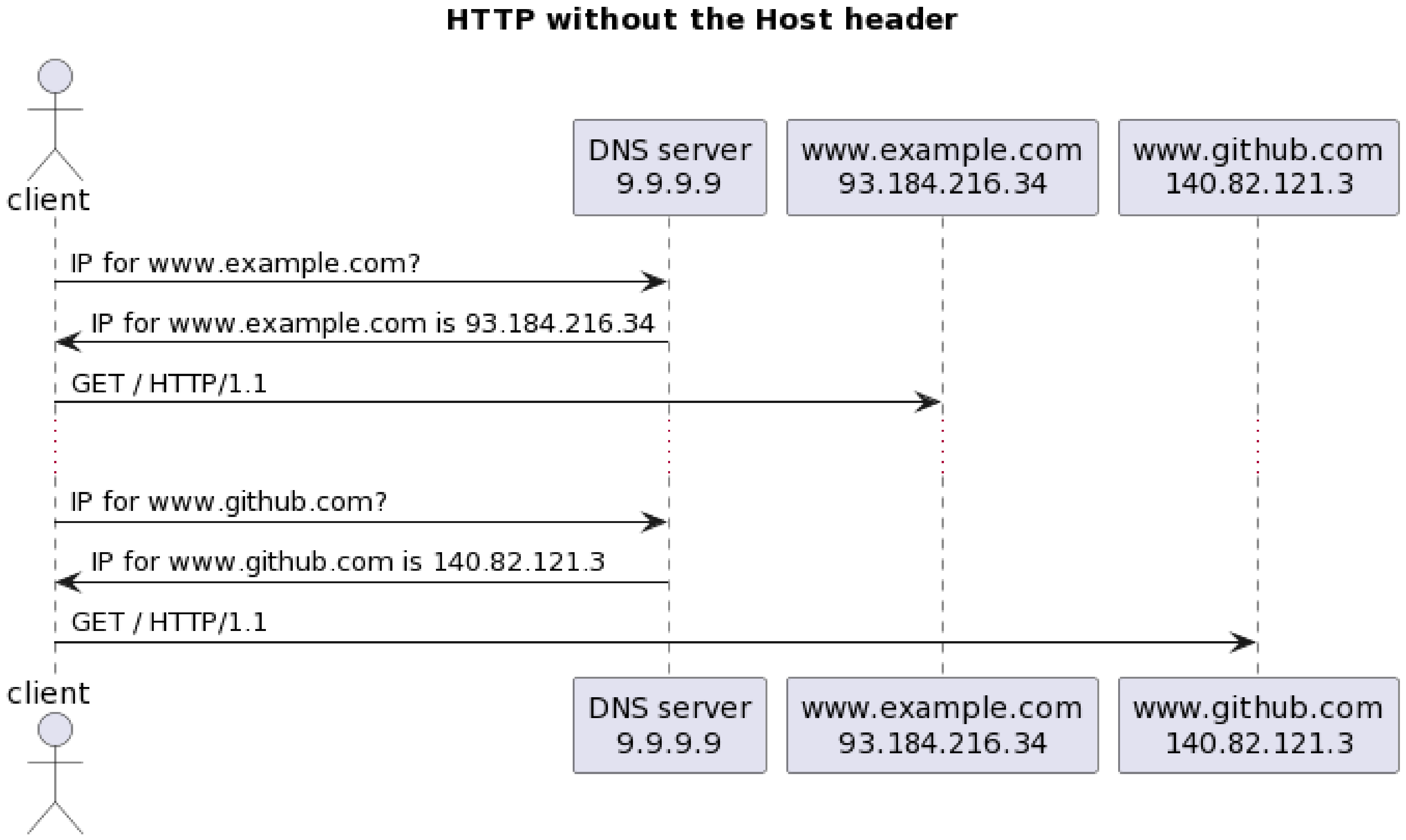
of web applications.

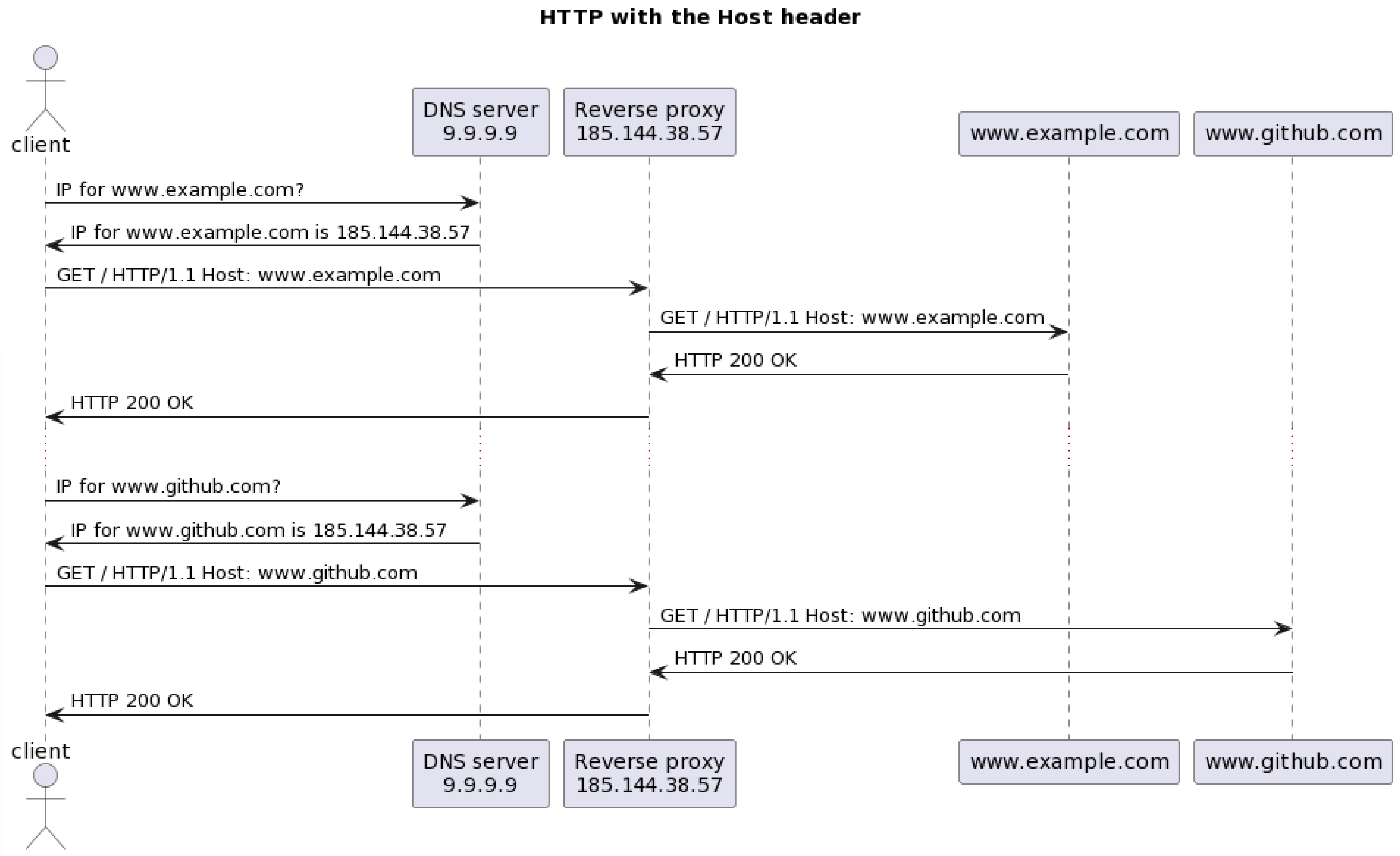
The `Host` header

More details for this section in the course material. You can find other resources and alternatives as well.

The `Host` header

- Part of the HTTP request
- Used to specify the domain name of the server
- Can be used to **host multiple websites on the same server** using a reverse proxy
- The reverse proxy will **route the request to the correct website** based on the `Host` header


HTTP without the Host header
client
DNS server 9.9.9.9
www.example.com 93.184.216.34
www.github.com 140.82.121.3
IP for www.example.com?
IP for www.example.com is 93.184.216.34
GET / HTTP/1.1
IP for www.github.com?
IP for www.github.com is 140.82.121.3
GET / HTTP/1.1
client
DNS server 9.9.9.9
www.example.com 93.184.216.34
www.github.com 140.82.121.3


HTTP with the Host header
client
DNS server 9.9.9.9
Reverse proxy 185.144.38.57
www.example.com
www.github.com
IP for www.example.com?
IP for www.example.com is 185.144.38.57
GET / HTTP/1.1 Host: www.example.com
GET / HTTP/1.1 Host: www.example.com
HTTP 200 OK
HTTP 200 OK
IP for www.github.com?
IP for www.github.com is 185.144.38.57
GET / HTTP/1.1 Host: www.github.com
GET / HTTP/1.1 Host: www.github.com
HTTP 200 OK
HTTP 200 OK
client
DNS server 9.9.9.9
Reverse proxy 185.144.38.57
www.example.com
www.github.com

Forward proxy and reverse proxy

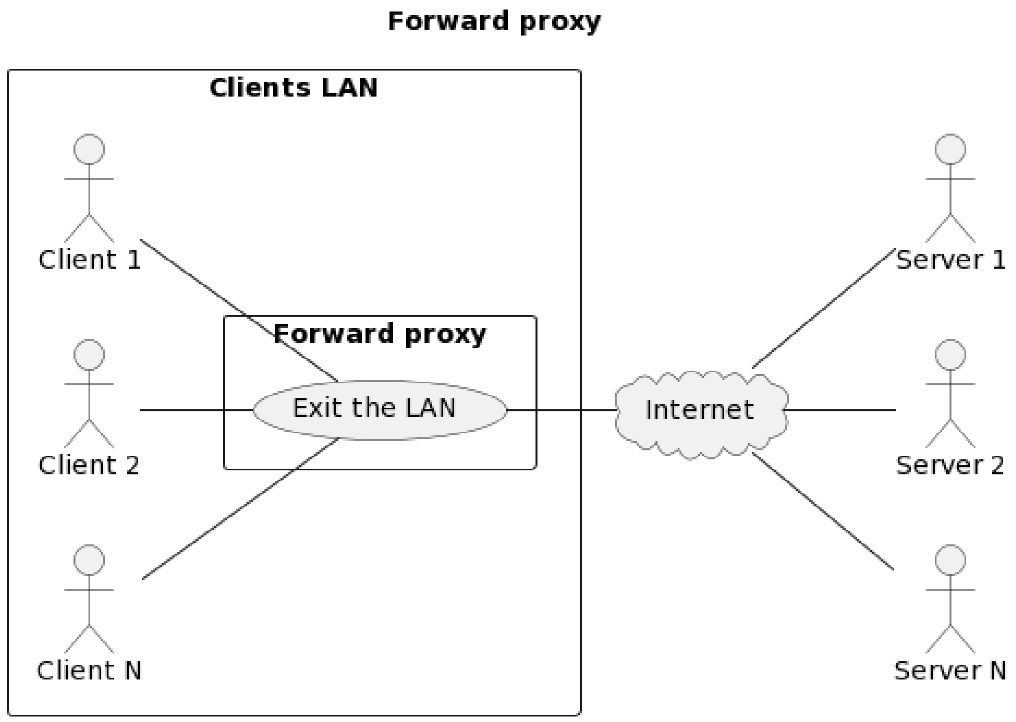
More details for this section in the course material. You can find other resources and alternatives as well.

Forward proxy and reverse proxy

- Proxies are components that **intercept** requests and responses and **filter/forward/change** them to another component.
- **Forward proxy**: used by a client to access external servers
- **Reverse proxy**: used by an external server to access internal servers

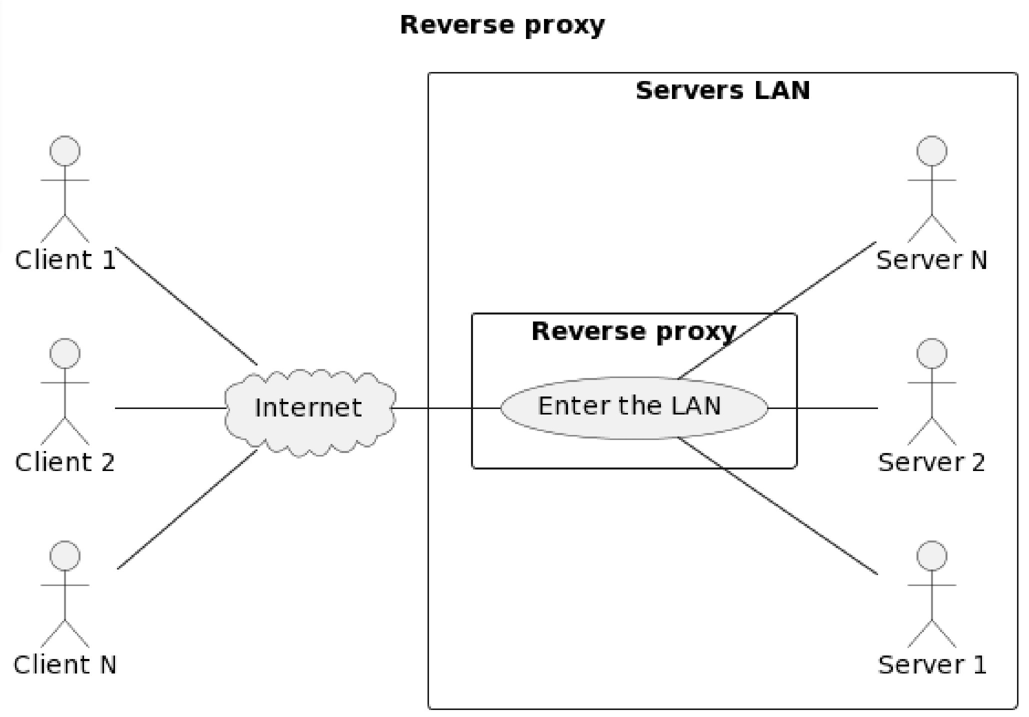
Forward proxy

- Operates between clients and external systems
- Can be used to:
  - Restrict access to external systems
  - Regulate traffic
  - Mask the identity of the client
  - Enforce security policies



Reverse proxy

- Operates between external systems and internal systems
- Can be used to:
    - Restrict access
    - Load balance requests to internal systems
    - Cache responses from internal systems



System scalability

More details for this section in the course material. You can find other resources and alternatives as well.

System scalability

- Capacity of a system to handle a varying amount of work
- Can be achieved by:
  - Vertical scaling (scale up)
  - Horizontal scaling (scale out)
- Can be achieved by:
  - Adding more resources
  - Adding more servers

Vertical scaling

Add more resources to a server:

- More RAM
- More CPU
- etc.

Limited by the hardware: at a certain point, you cannot add more/better resources to a server.

Horizontal scaling

Add more servers to a system and distribute the load between them.

Limited by the software - your software must be able to run on multiple servers:

- Backends/API accessing the same database(s)
- Frontends accessing backends/API

When to use scale up or scale out?

- Determined by the **non-functional requirements** of the system
- You need metrics to determine when to scale up or scale out to identify bottlenecks
- Once the bottleneck is identified (from monitoring), you can decide to scale up or scale out

How to monitor a system?

Out of scope for this course, but here are some tools you can use:

- Prometheus
- Grafana
- Sentry
- LibreNMS

Load balancing

More details for this section in the course material. You can find other resources and alternatives as well.

Load balancing

Process of **distributing the load** between multiple servers.

This can work thanks to the **stateless** nature of HTTP and the `Host` header.

The load balancer must know the **pool of servers** it can forward the requests to.

Multiple strategies can be used to distribute the load:

- **Round-robin**: each server in the pool in turn (covered in this course)
- **Least connections**: least number of active connections
- **Least response time**: least response time
- **Hashing**: based on a hash of the request (e.g. the IP address of the client, the URL of the request, etc.)

An issue with load balancing is **session management**: the load balancer could forward requests from the same client to different servers, loosing their session.

As HTTP is stateless, the load balancer must know how to forward requests from the same client to the same server. A solution is **sticky sessions** with the help of a cookie.

Caching

More details for this section in the course material. You can find other resources and alternatives as well.

Caching

Process of **storing a copy of a resource to serve it faster**.

Caching can **improve the performance** of a system and **reduce the load** on the backend.

Caching can be done on the **client-side** or **server-side**.

Managing cache with HTTP

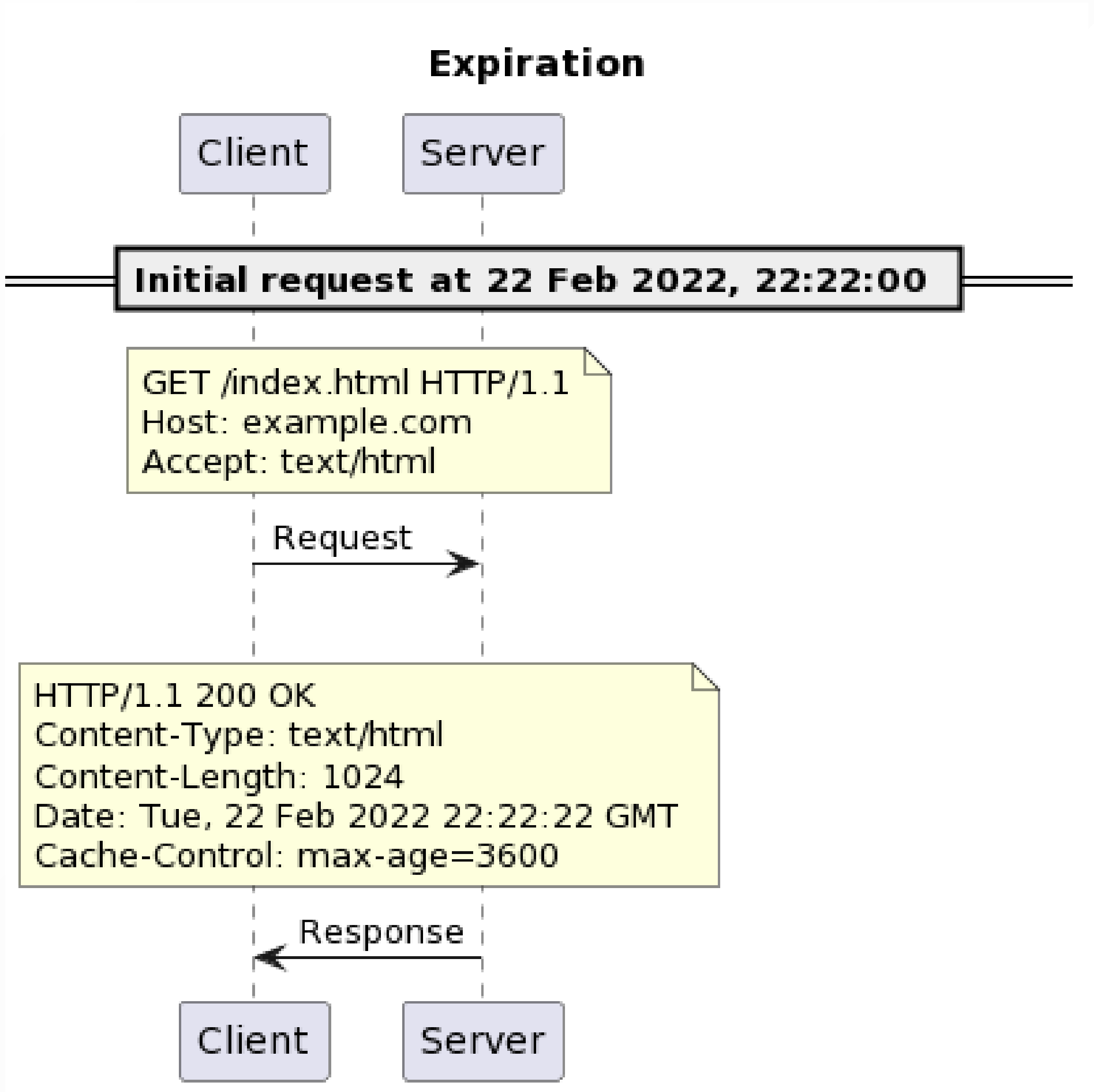
Managing chache is challenging because it is difficult to know when to invalidate the cache (the data can be stale).

Two main caching models:

- **Expiration model**: the cache is valid for a certain amount of time
- **Validation model**: the cache is valid until the data is modified

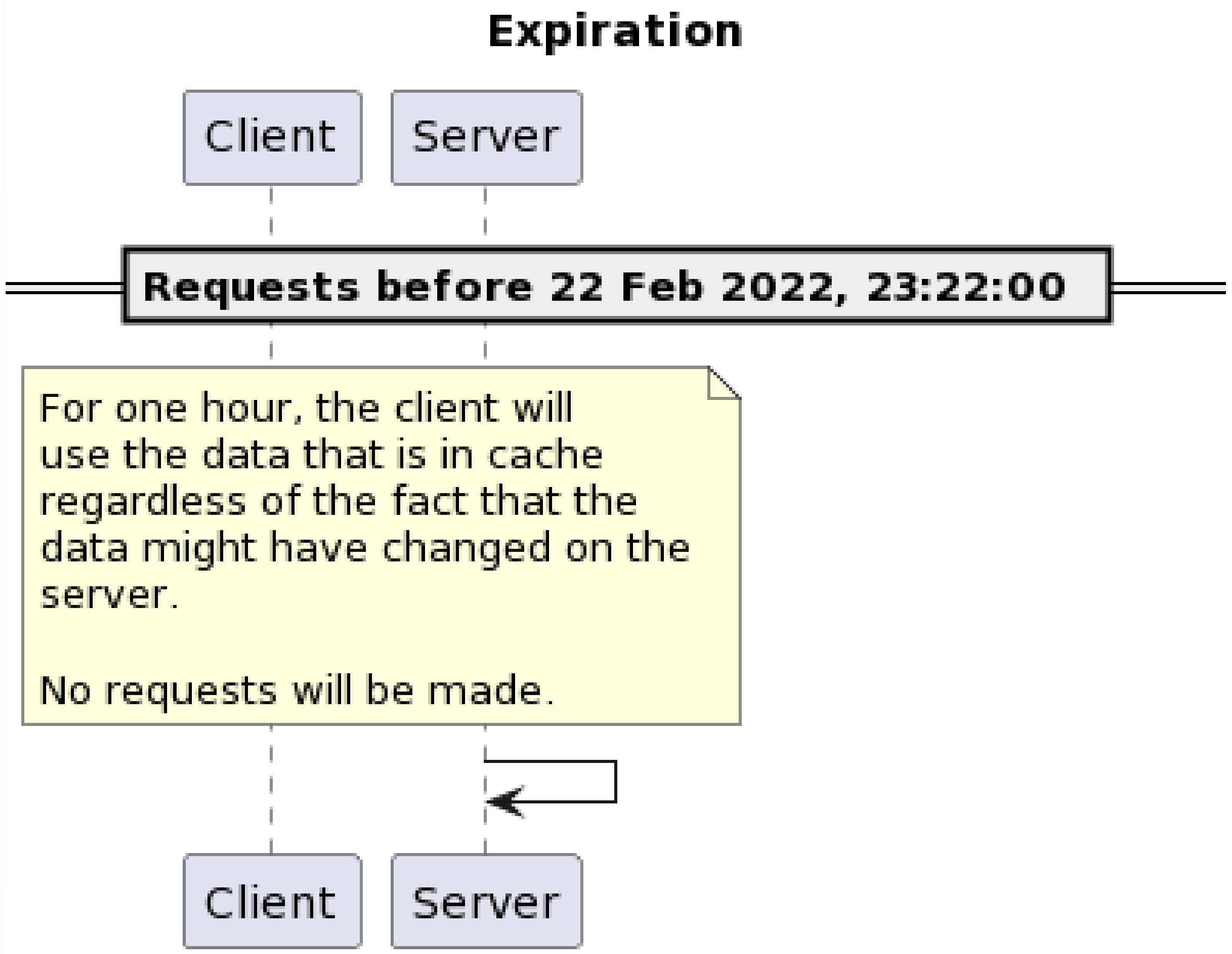
Expiration model

- The cache is valid for a certain amount of time
- If the cache is not expired, the cache is used
- Uses the `Cache-Control: max-age=<secondes>` header
- The cache is invalidated after the expiration time



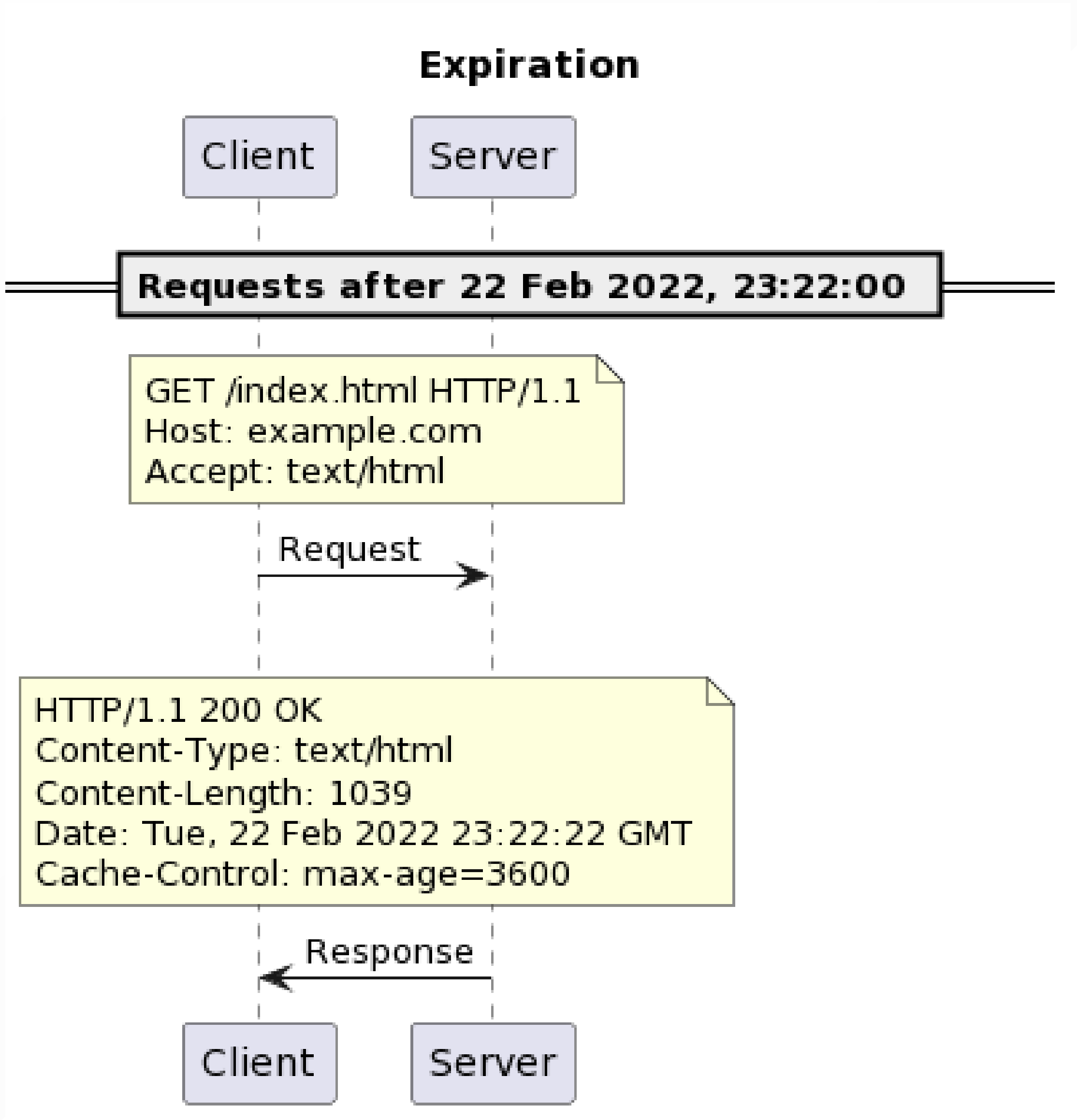
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=3600
```



For one hour, the client will use the data that is in cache regardless of the fact that the data might have changed on the server.

No requests will be made.

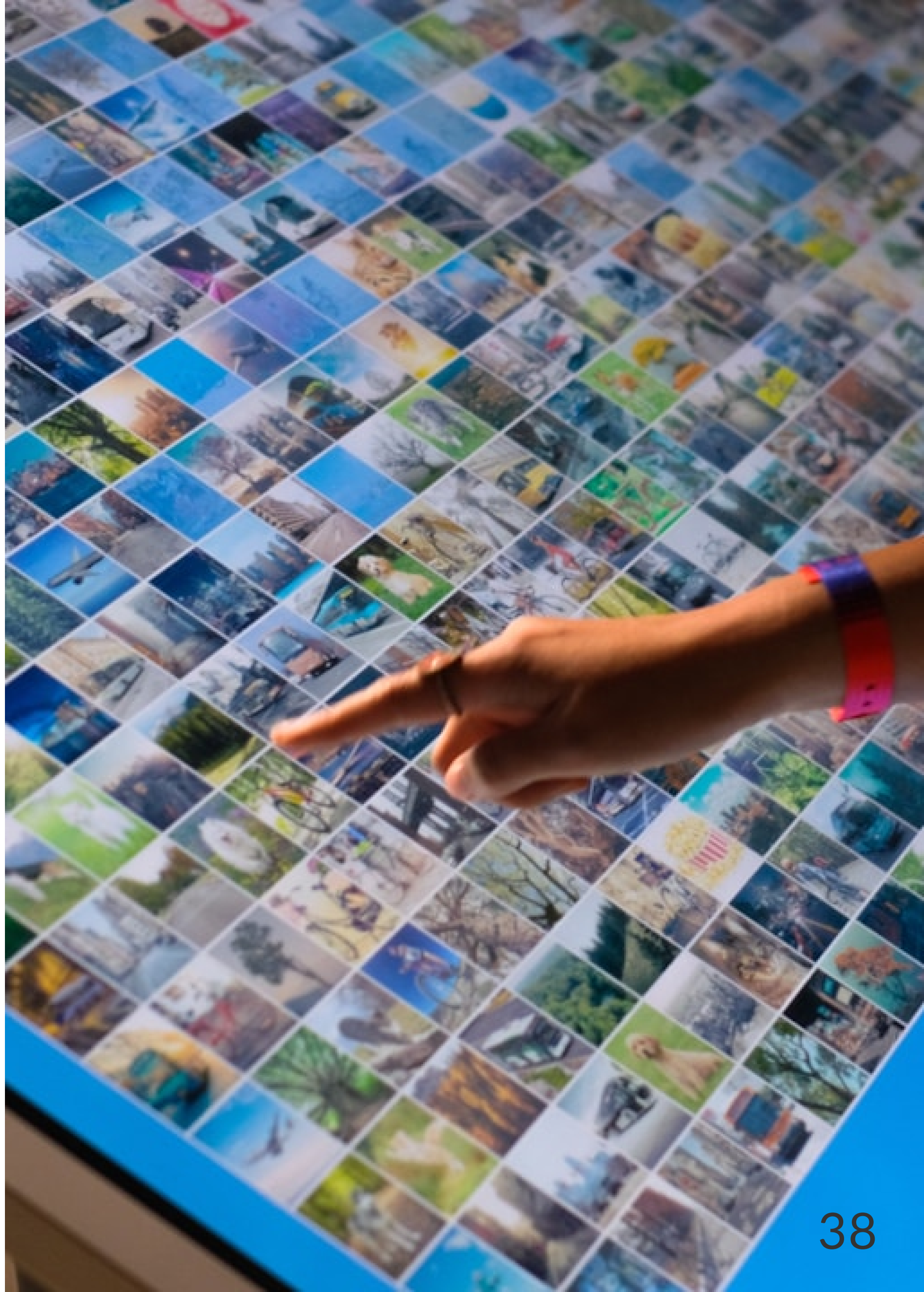


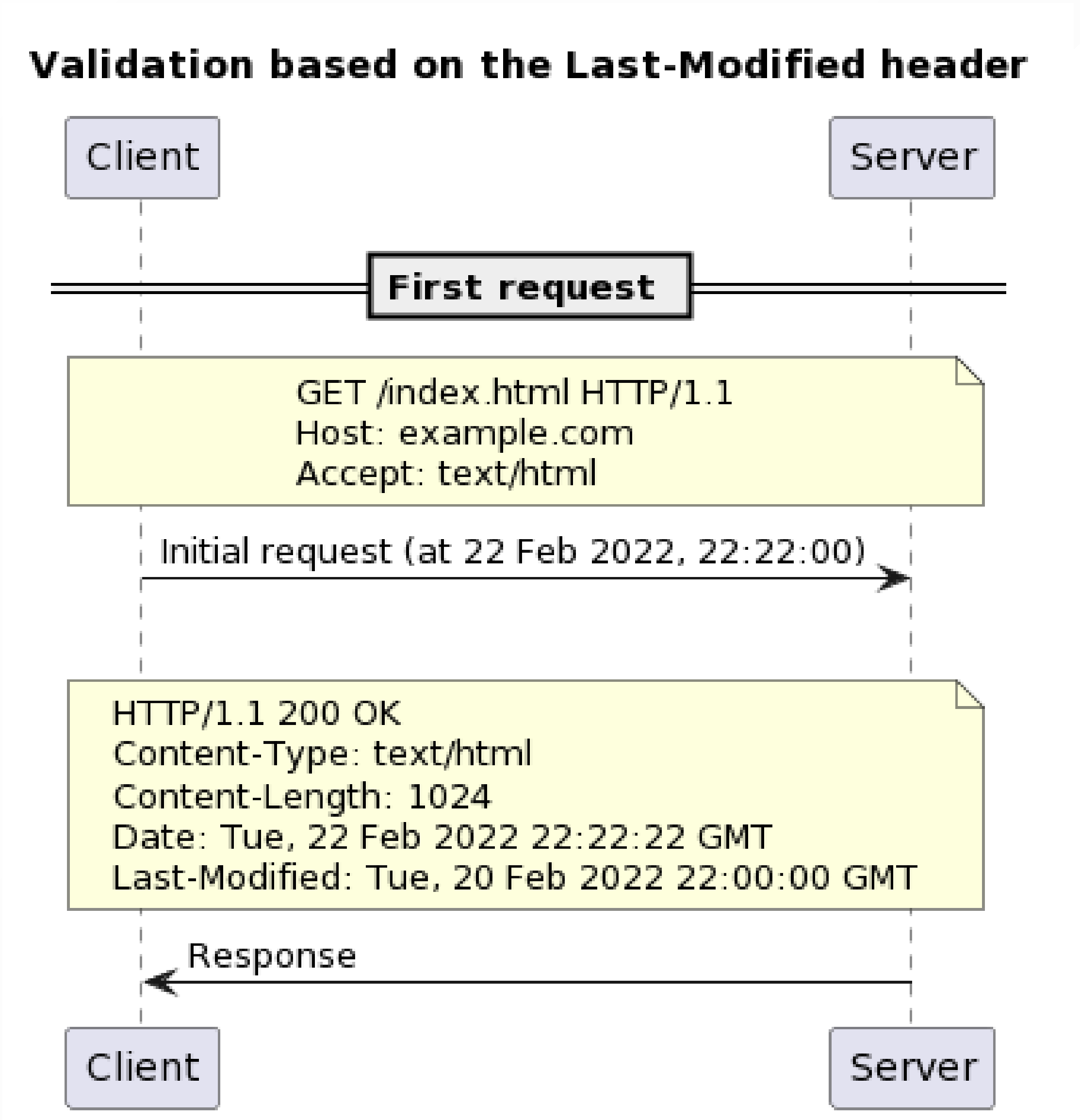
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1039
Date: Tue, 22 Feb 2022 23:22:22 GMT
Cache-Control: max-age=3600
```

Validation model

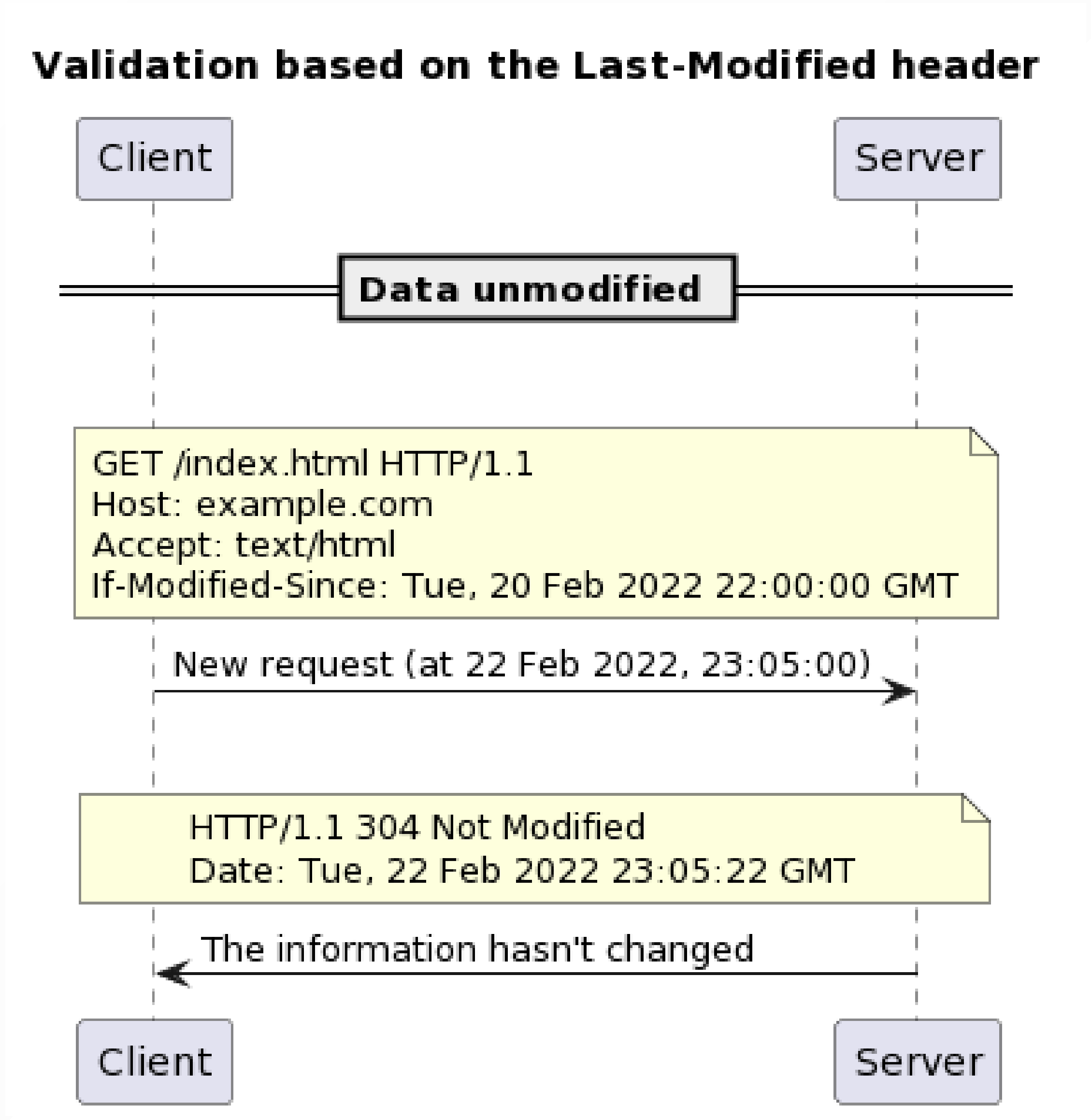
- The cache is valid until the data is modified
- If the cache is not expired, the cache is used
- Two ways to validate the cache:
  - **Last-Modified**: `Last-Modified` and `If-Modified-Since` headers
  - **ETag**: `ETag` and `If-None-Match` headers




Validation based on the Last-Modified header
Client
Server
First request
Initial request (at 22 Feb 2022, 22:22:00)
Response
Client
Server

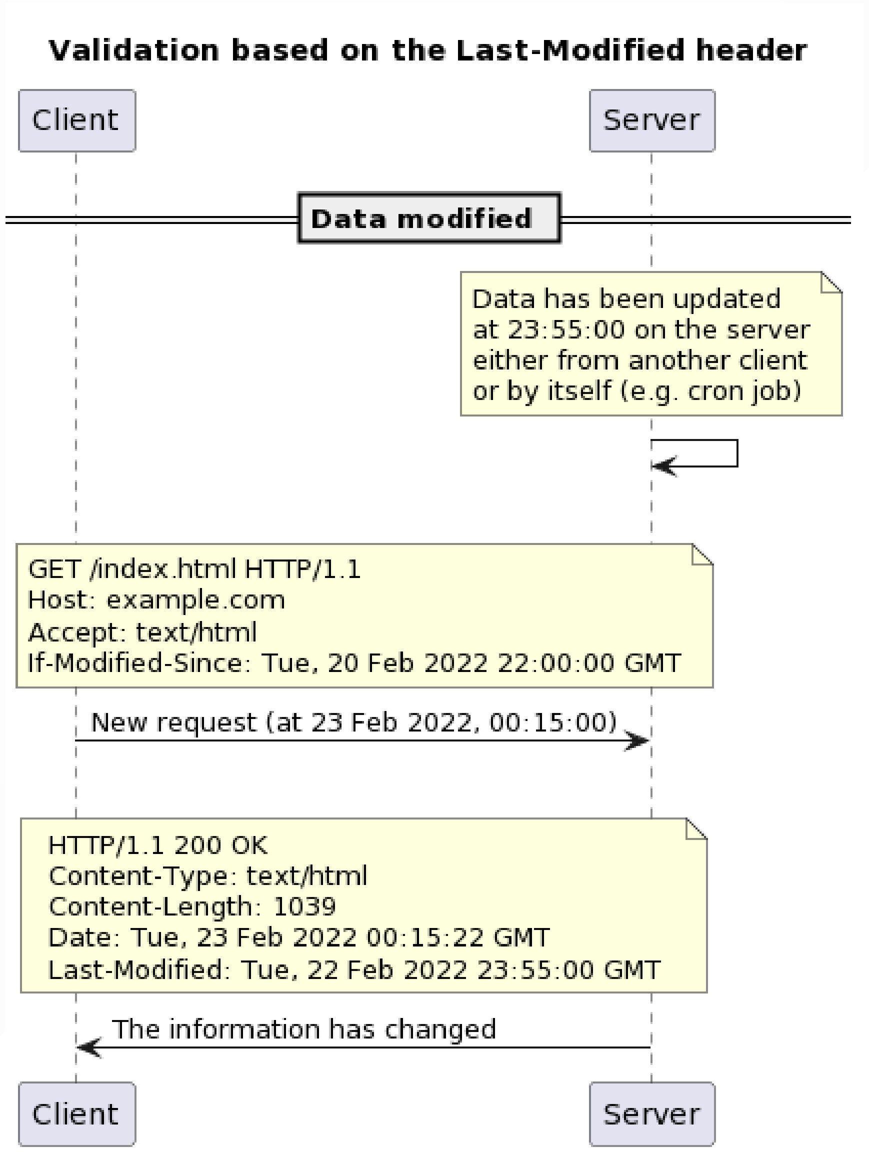
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 20 Feb 2022 22:00:00 GMT
```


Validation based on the Last-Modified header
Client
Server
Data unmodified
New request (at 22 Feb 2022, 23:05:00)
The information hasn't changed
Client
Server

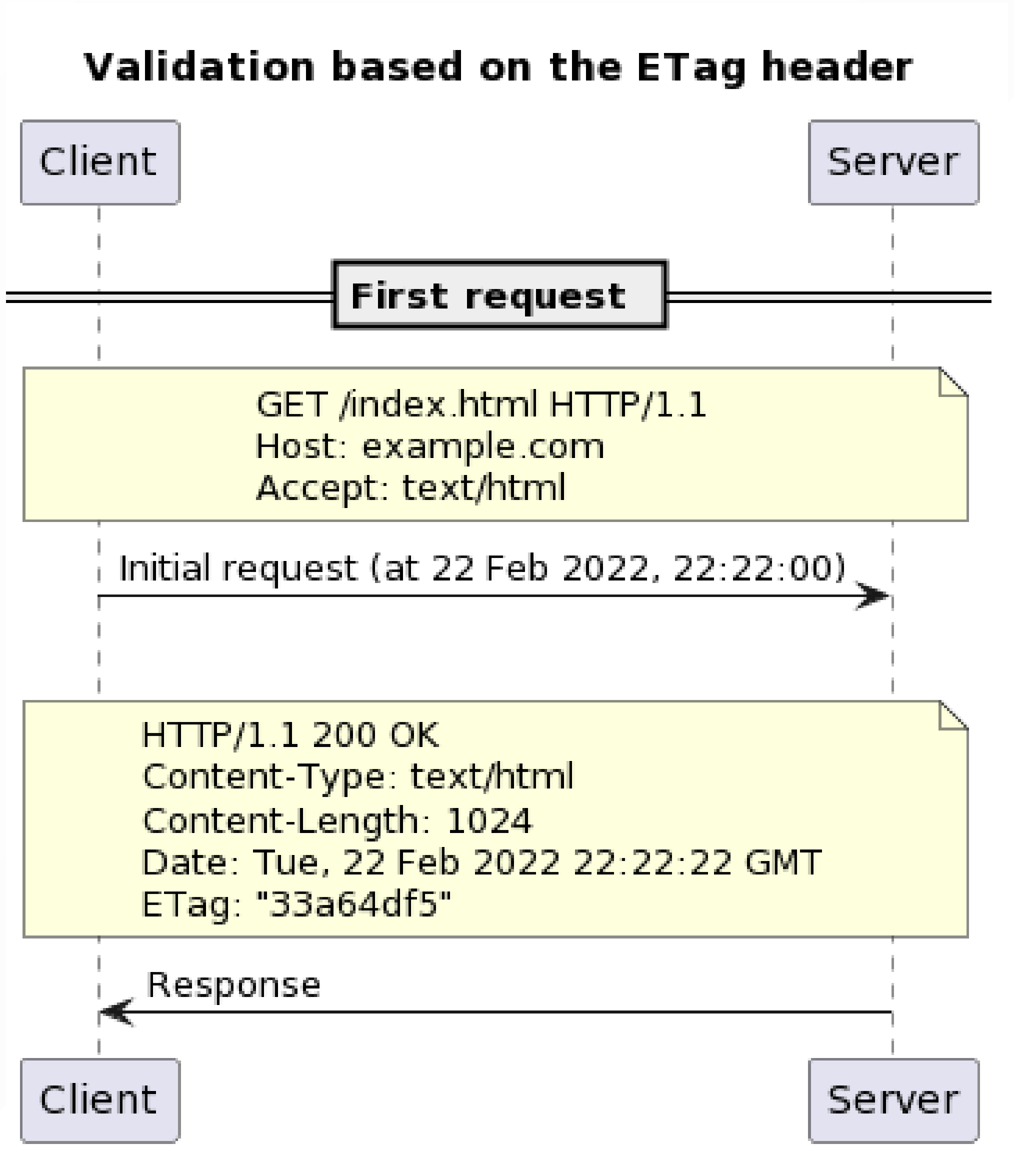
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-Modified-Since: Tue, 20 Feb 2022 22:00:00 GMT
```

```
HTTP/1.1 304 Not Modified
Date: Tue, 22 Feb 2022 23:05:22 GMT
```


Validation based on the Last-Modified header
Client
Server
Data modified
Data has been updated at 23:55:00 on the server either from another client or by itself (e.g. cron job)
New request (at 23 Feb 2022, 00:15:00)
The information has changed
Client
Server

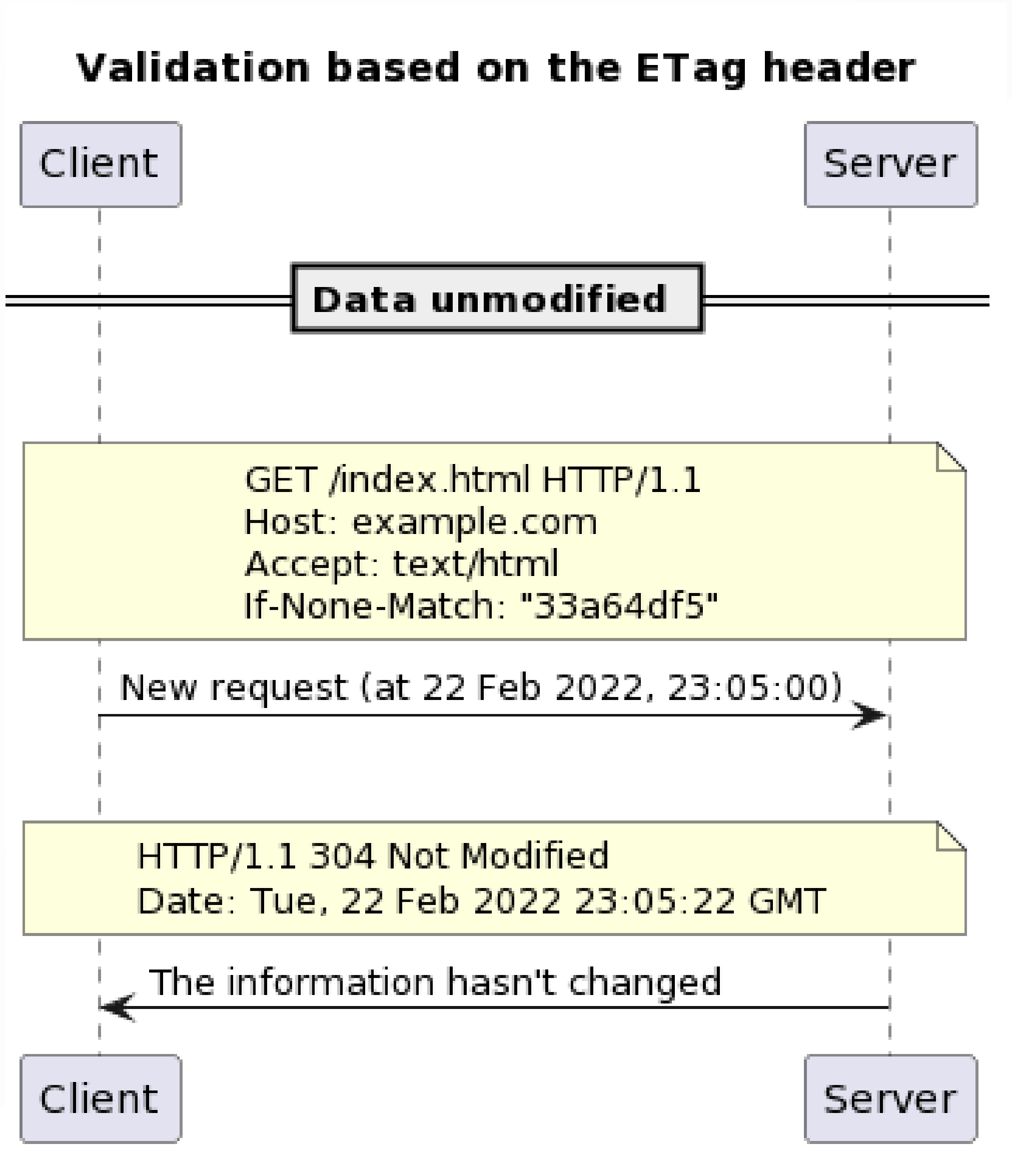
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-Modified-Since: Tue, 20 Feb 2022 22:00:00 GMT
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1039
Date: Tue, 23 Feb 2022 00:15:22 GMT
Last-Modified: Tue, 22 Feb 2022 23:55:00 GMT
```


Validation based on the ETag header
Client
Server
First request
Initial request (at 22 Feb 2022, 22:22:00)
Response
Client
Server

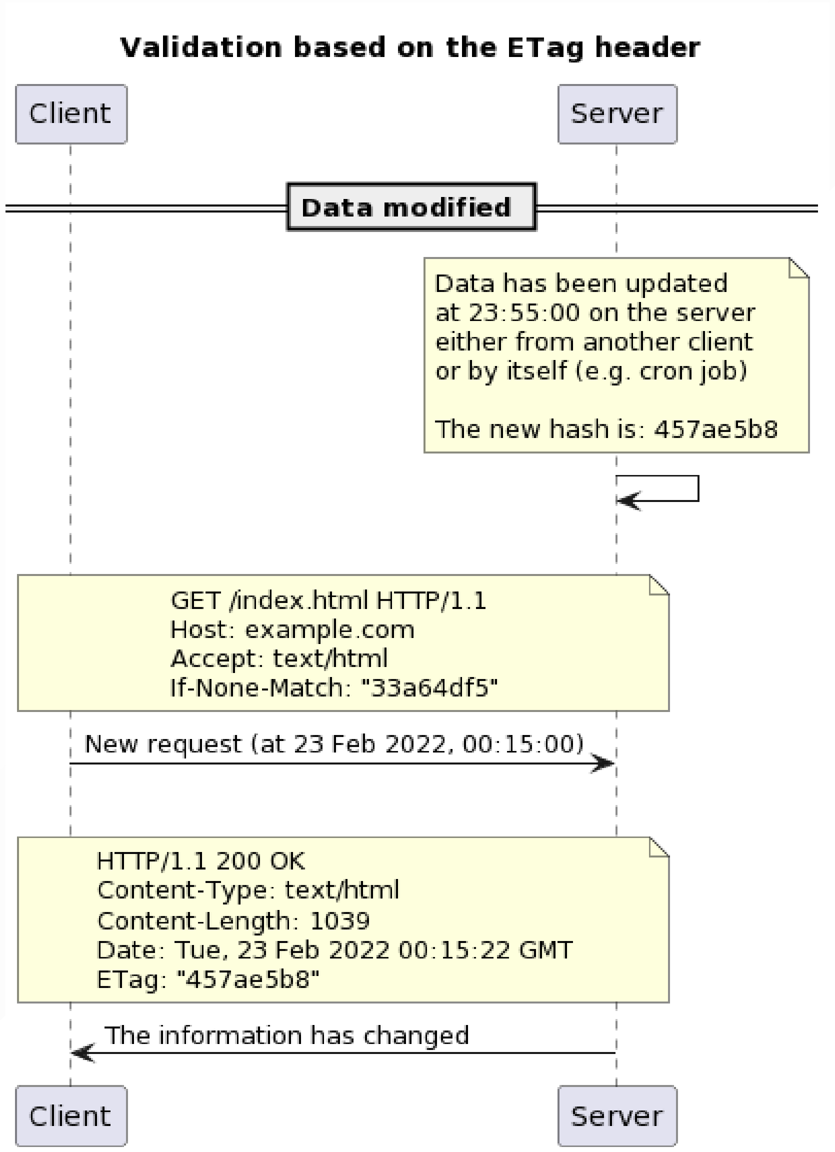
```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
ETag: "33a64df5"
```


Validation based on the ETag header
Client
Server
Data unmodified
New request (at 22 Feb 2022, 23:05:00)
The information hasn't changed
Client
Server

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-None-Match: "33a64df5"
```

```
HTTP/1.1 304 Not Modified
Date: Tue, 22 Feb 2022 23:05:22 GMT
```


Validation based on the ETag header
Client
Server
Data modified
Data has been updated
at 23:55:00 on the server
either from another client
or by itself (e.g. cron job)

The new hash is: 457ae5b8

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-None-Match: "33a64df5"
```

New request (at 23 Feb 2022, 00:15:00)

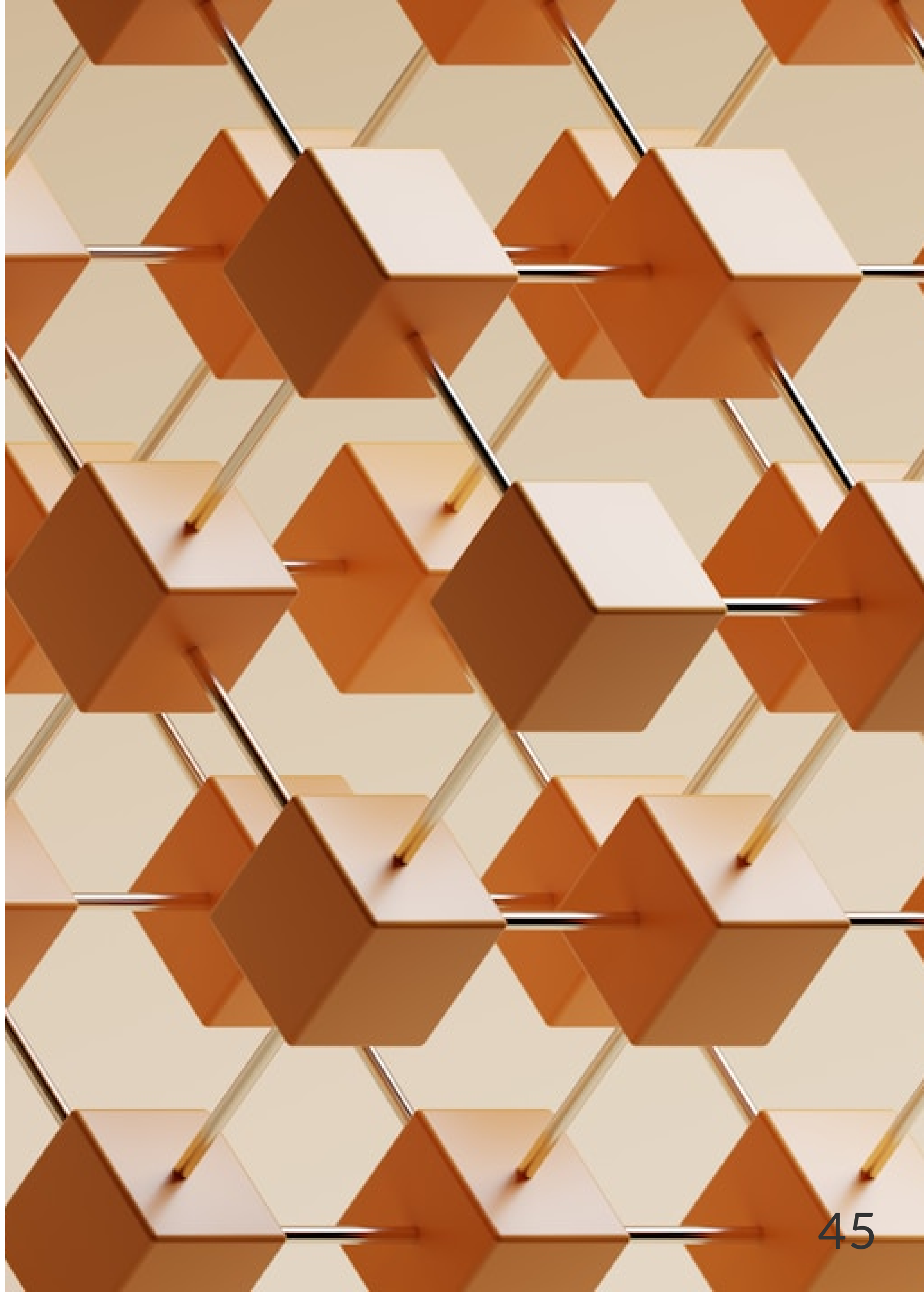
```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1039
Date: Tue, 23 Feb 2022 00:15:22 GMT
ETag: "457ae5b8"
```

The information has changed
Client
Server

CDN

Content Delivery Network (CDN) is a network of servers that are geographically distributed around the world.

Improve performance by serving static content (images, videos, etc.) from the closest server.

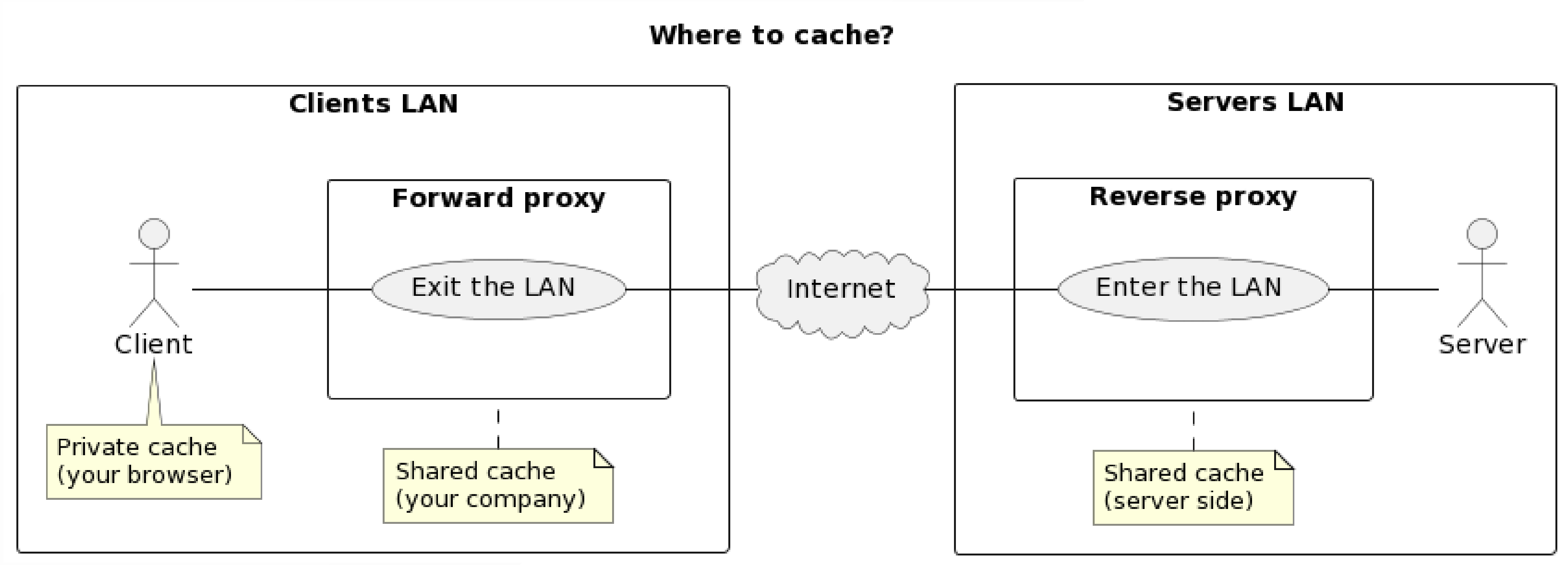


Where to cache?

The best would be to cache at each level of the system to ensure the best performance but it is not always possible or faisable:

- **Client-side**: the cache is stored on the client
- **Server-side**: the cache is stored on the server
- **CDN**: the cache is stored on a CDN

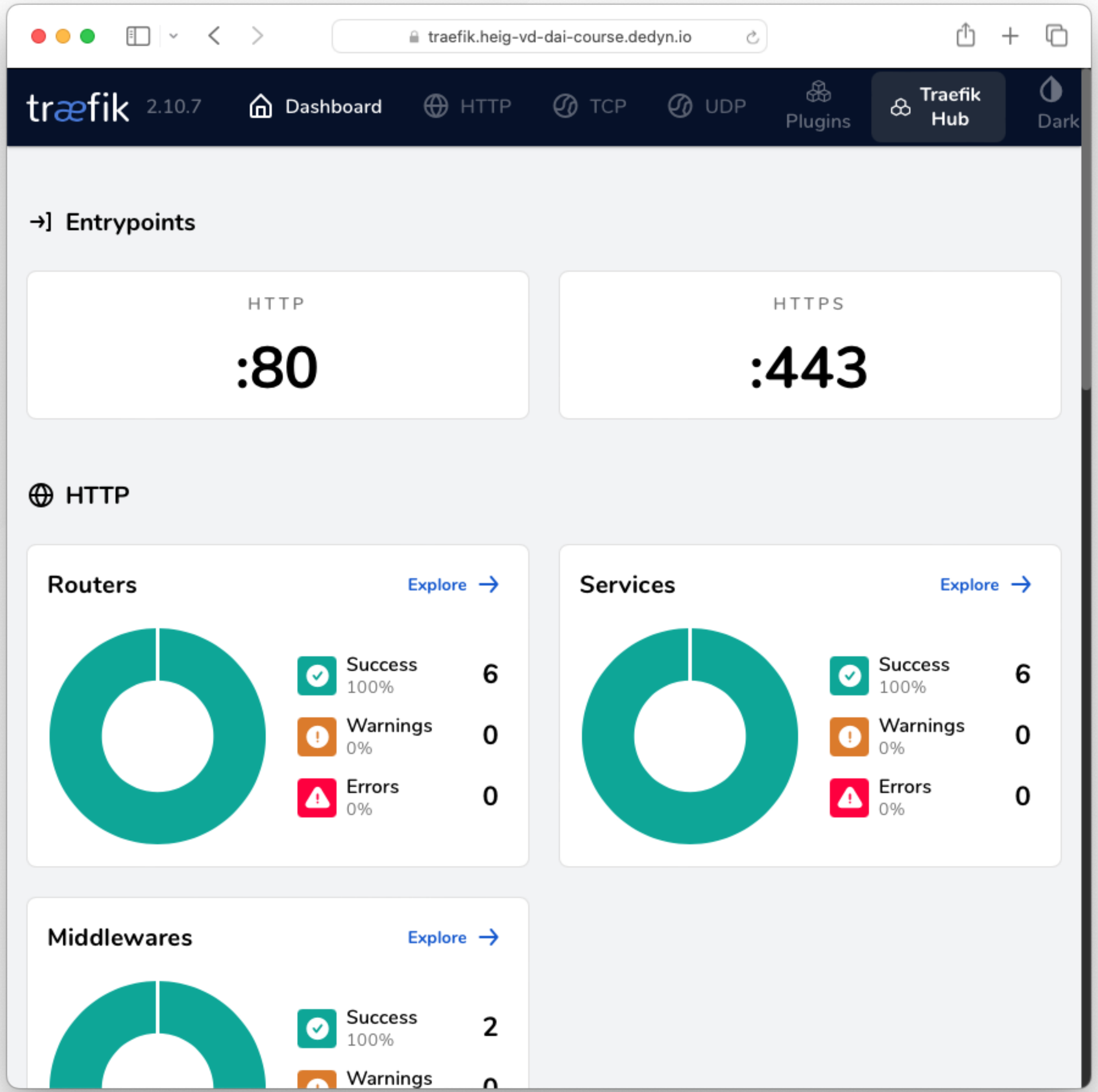
Private caches are caches that are only used by one client. Public caches are caches that are used by multiple clients.


Where to cache?
Clients LAN
Forward proxy
Client
Exit the LAN
Private cache (your browser)
Shared cache (your company)
Internet
Servers LAN
Reverse proxy
Enter the LAN
Server
Shared cache (server side)

Practical content

What will you do?

- Set up a reverse proxy
- Set up whoami
- Explore the features of the reverse proxy:
  - `PathPrefix` rule
  - `Host` rule
  - `StripPrefix` middleware
  - Sticky sessions



Entrypoints

HTTP	HTTPS
:80	:443

HTTP

Routers Explore

Success	100%	6
Warnings	0%	0
Errors	0%	0

Services Explore

Success	100%	6
Warnings	0%	0
Errors	0%	0

Middlewares Explore

Success	100%	2
Warnings		0

Find the practical content

You can find the practical content for this chapter on GitHub.

Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

➡️ GitHub Discussions

You can use reactions to express your opinion on a comment!

What will you do next?

You will start the practical work!

Sources

- Main illustration by Nicolas Picard on Unsplash
- Illustration by Aline de Nadai on Unsplash
- Illustration by Mohammadreza alidoos on Unsplash
- Illustration by Gaurav Dhwaj Khadka on Unsplash
- Illustration by Imre Tömösvári on Unsplash
- Illustration by Kelvin T on Unsplash
- Illustration by Alfred Kenneally on Unsplash
- Illustration by Mikhail Vasilyev on Unsplash

- Illustration by Ibrahim Boran on Unsplash
- Illustration by Elena Mozhvilo on Unsplash
- Illustration by Evan Krause on Unsplash
- Illustration by Markus Spiske on Unsplash
- Illustration by Fermin Rodriguez Penelas on Unsplash
- Illustration by Andrik Langfield on Unsplash
- Illustration by Karen Grigorean on Unsplash
- Illustration by Shubham's Web3 on Unsplash