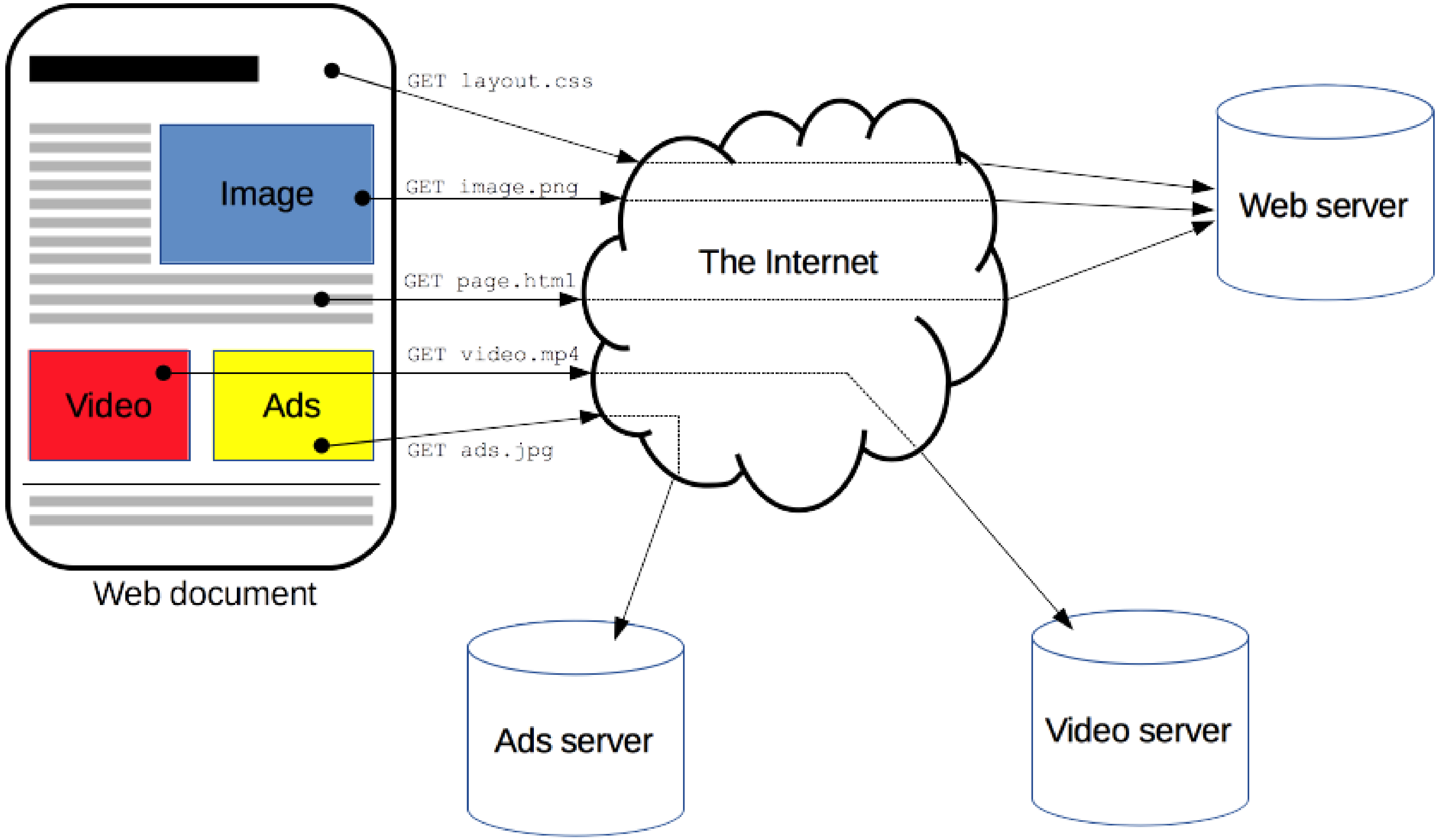
The nature of HTTP requests

WEB SCRAPING IN R

Timo Grossenbacher
Instructor

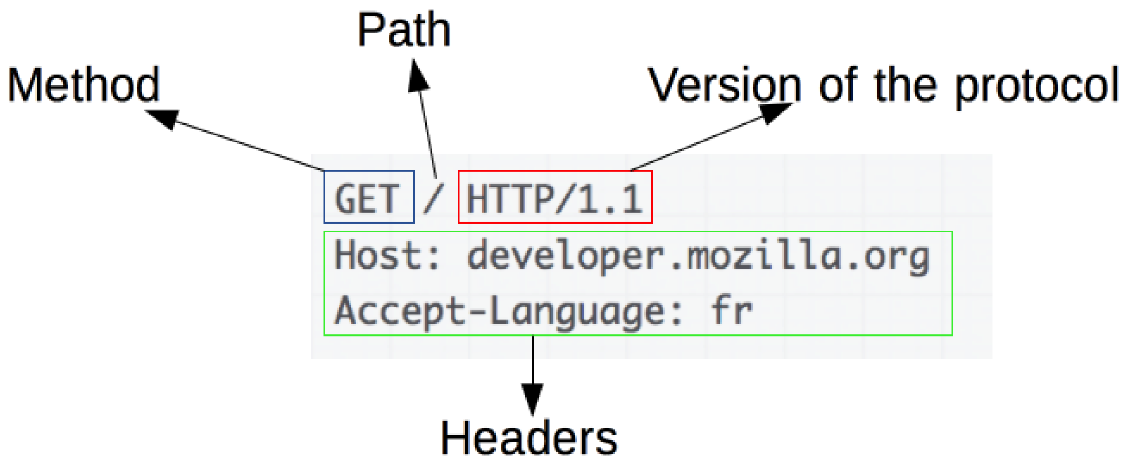
Hypertext Transfer Protocol (HTTP)



[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

The anatomy of requests

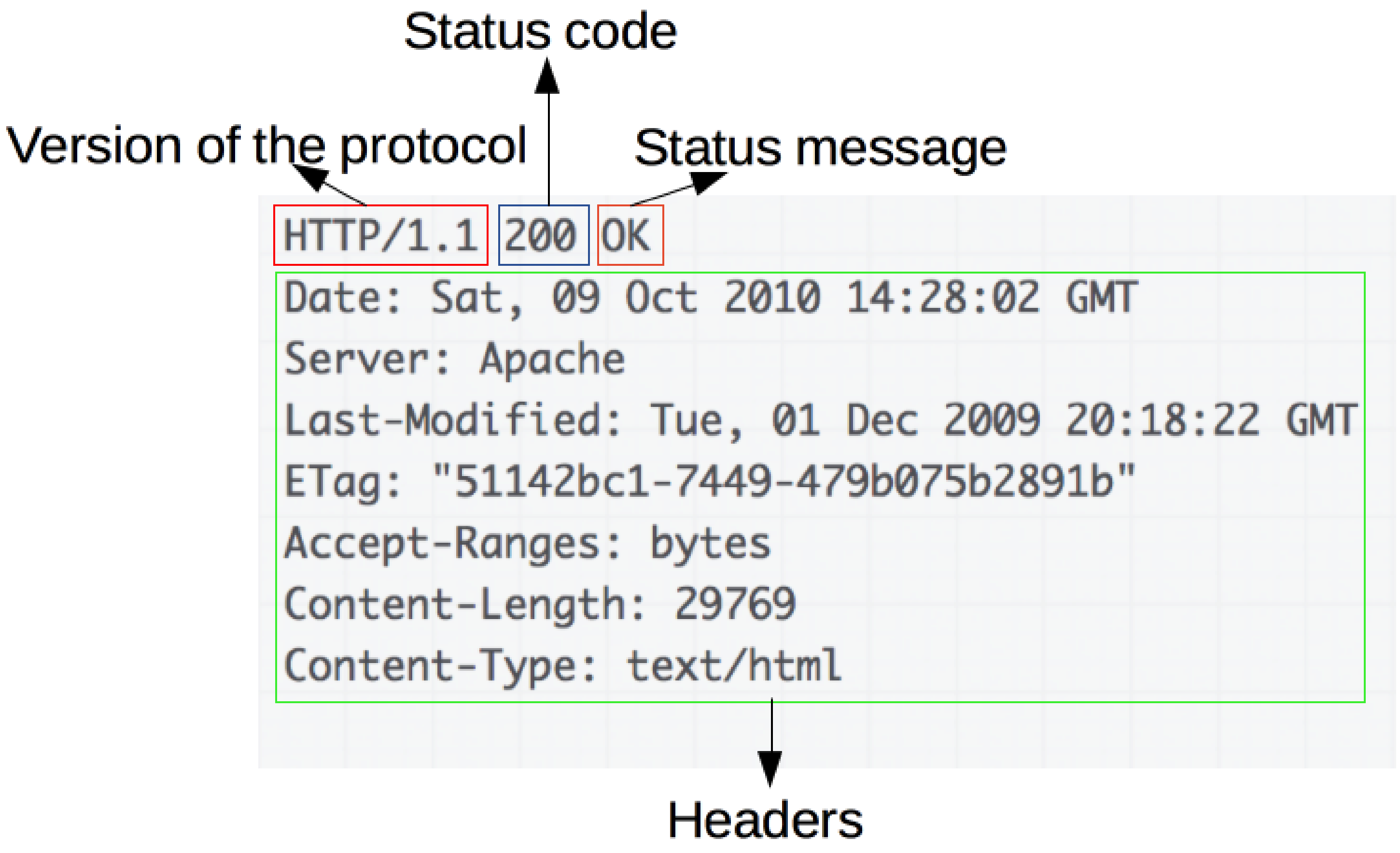
A request is sent *to* the web server



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Typical status codes: `200` (OK), `404` (NOT FOUND), `3xx` (redirects), `5xx` (server errors)

A response is received *from* the web server



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Request methods: GET and POST

- GET: Used to fetch a resource without submitting data (`GET /index.html`)
- POST: Used to send data to a server, e.g. after filling out a form on a page

```
POST /test HTTP/1.1
Host: foo.example
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

field1=value1&field2=value2
```

POST requests are also answered with a response!

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

HTTP requests with httr

```r
library(httr)
GET('https://httpbin.org')
```

```
Response [https://httpbin.org/]
  Date: 2020-09-19 13:02
  Status: 200
  Content-Type: text/html; charset=utf-8
  Size: 9.59 kB
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    ...
```

HTTP requests with httr

```
library(httr)
response <- GET('https://httpbin.org')
content(response)
```

```
{html_document}
<html lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF ...
[2] <body>\n    <a href="https://github.com/requests/httpbin" class="github ...
```

Let's practice!

WEB SCRAPING IN R

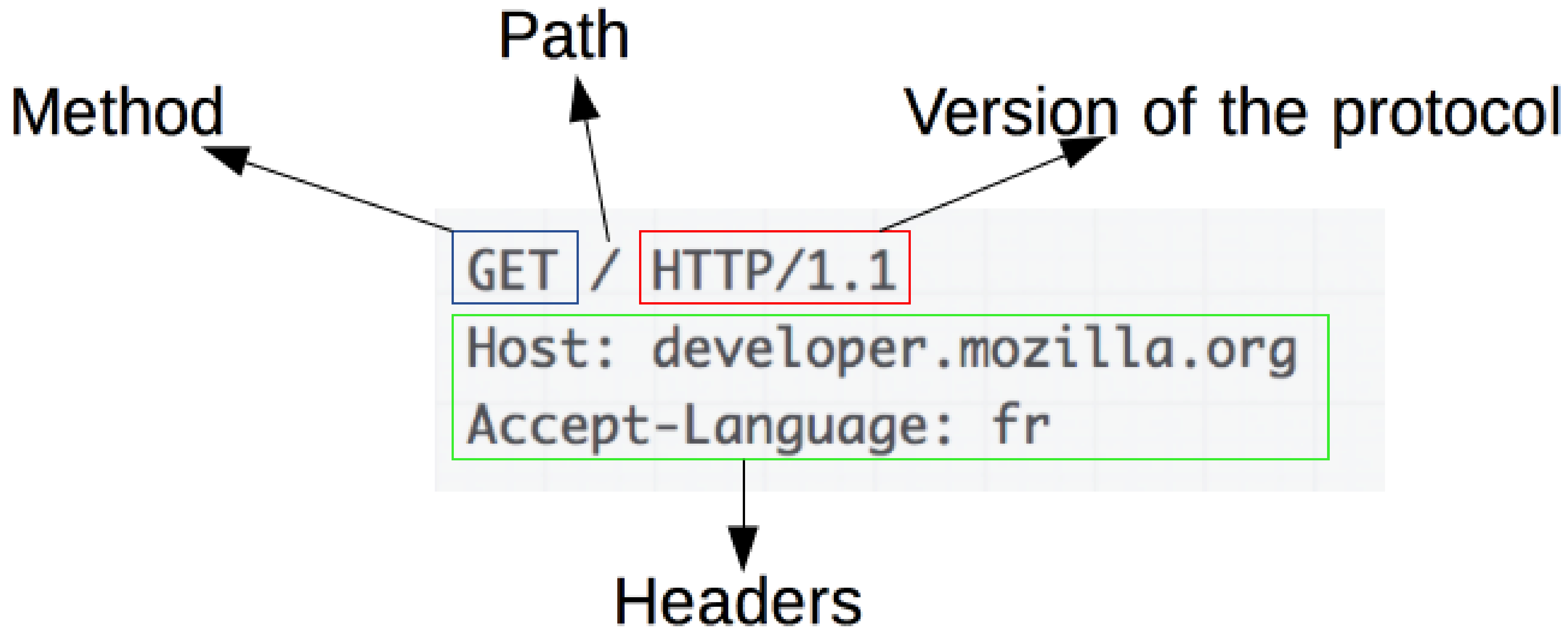
Telling who you are with custom user agents

WEB SCRAPING IN R

Timo Grossenbacher
Instructor

Show yourself!

- Web server already registers your IP address
- Better to explicitly identify yourself
- There's an HTTP header for that!



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Modify headers with httr

```
response <- GET('http://example.com', user_agent("Hey, it's me, Timo!
    Reach me at timo@timogrossenbacher.ch."))
```

Alternatively:

```
set_config(add_headers(`User-Agent` = "Hey, it's me, Timo!
    Reach me at timo@timogrossenbacher.ch."))
response <- GET('http://example.com')
```

Let's try this!

How to be gentle and slow down your requests

WEB SCRAPING IN R

Timo Grossenbacher
Instructor

datacamp

Don't try this at home!

```r
library(httr)
while(TRUE){
  print(Sys.time())
  response <-
    GET("https://httpbin.org")
  print(status_code(response))
}
```

```
[1] "2020-06-20 10:31:17 CEST"
[1] 200
[1] "2020-06-20 10:31:17 CEST"
[1] 200
[1] "2020-06-20 10:31:17 CEST"
[1] 200
[1] "2020-06-20 10:31:17 CEST"
[1] 200
[1] "2020-06-20 10:31:17 CEST"
[1] 200
[1] "2020-06-20 10:31:18 CEST"
[1] 200
...
```

A nicer way of requesting data from websites

```r
while(TRUE){
  # Wait one second
  # ...
  print(Sys.time())
  response <-
    GET("https://httpbin.org")
  print(status_code(response))
}
```

```
[1] "2020-06-20 10:36:06 CEST"
[1] 200
[1] "2020-06-20 10:36:07 CEST"
[1] 200
[1] "2020-06-20 10:36:08 CEST"
[1] 200
[1] "2020-06-20 10:36:09 CEST"
[1] 200
[1] "2020-06-20 10:36:10 CEST"
[1] 200
[1] "2020-06-20 10:36:11 CEST"
[1] 200
...
```

A tidy approach to throttling

Throttling a function = introducing a time delay between calls

```
library(httr)
library(purrr)
throttled_GET <- slowly(
  ~ GET("https://httbin.org"),
    rate = rate_delay(3))
while(TRUE){
  print(Sys.time())
  response <- throttled_GET()
  print(status_code(response))
}
```

```
[1] "2020-06-20 10:53:44 CEST"
[1] 200
[1] "2020-06-20 10:53:47 CEST"
[1] 200
[1] "2020-06-20 10:53:50 CEST"
[1] 200
[1] "2020-06-20 10:53:53 CEST"
[1] 200
[1] "2020-06-20 10:53:56 CEST"
[1] 200
...
```

Query custom URLs in a throttled function

```r
library(httr)
library(purrr)
throttled_GET <-
    # instead of GET("https://...")
    slowly(~ GET(.), rate = rate_delay(3))
while(TRUE){
  print(Sys.time())
  response <-
      throttled_GET("https://wikipedia.org")
  print(status_code(response))
}
```

```
[1] "2020-06-20 10:53:44 CEST"
[1] 200
[1] "2020-06-20 10:53:47 CEST"
[1] 200
[1] "2020-06-20 10:53:50 CEST"
[1] 200
[1] "2020-06-20 10:53:53 CEST"
[1] 200
[1] "2020-06-20 10:53:56 CEST"
[1] 200
...
```

Looping over a list of URLs

```
library(httr)
url_list <- c("https://httbin.org/anything/1",
              "https://httbin.org/anything/2",
              "https://httbin.org/anything/3")

for(url in url_list){
  response <- throttled_GET(url)
  print(status_code(response))
}
```

```
[1] 200
[1] 200
[1] 200
```

```
library(httr)
url_list <- c("https://wikipedia.org/wiki/K2",
              "https://wikipedia.org/wiki/\
    Mount_Everest")

for(url in url_list){
  response <- throttled_GET(url)
  print(status_code(response))
}
```

```
[1] 200
[1] 200
```

Let's apply this to a real world example!

WEB SCRAPING IN R

Recap: Web Scraping in R

WEB SCRAPING IN R

Timo Grossenbacher
Instructor

Concepts covered

- Chapter 1: Introduction to HTML and Web Scraping
- Chapter 2: Navigation and Selection with CSS
    - Fundamental web technologies and how to exploit them for scraping
    - The `rvest` package
- Chapter 3: Advanced Selection with XPATH
    - XPATH functions like `position()` or `text()`
    - Node selection based on surrounding nodes (e.g. children)
- Chapter 4: Scraping Best Practices
    - Behind everything: HTTP (and the `httr` package)
    - Best practices like throttling and identifying user agents

What to do with the scraped data?

DataCamp courses:

- Cleaning Data in R
- Working with Data in the Tidyverse
- Dealing with Missing Data in R
- Communicating with Data in the Tidyverse

Happy scraping!

WEB SCRAPING IN R