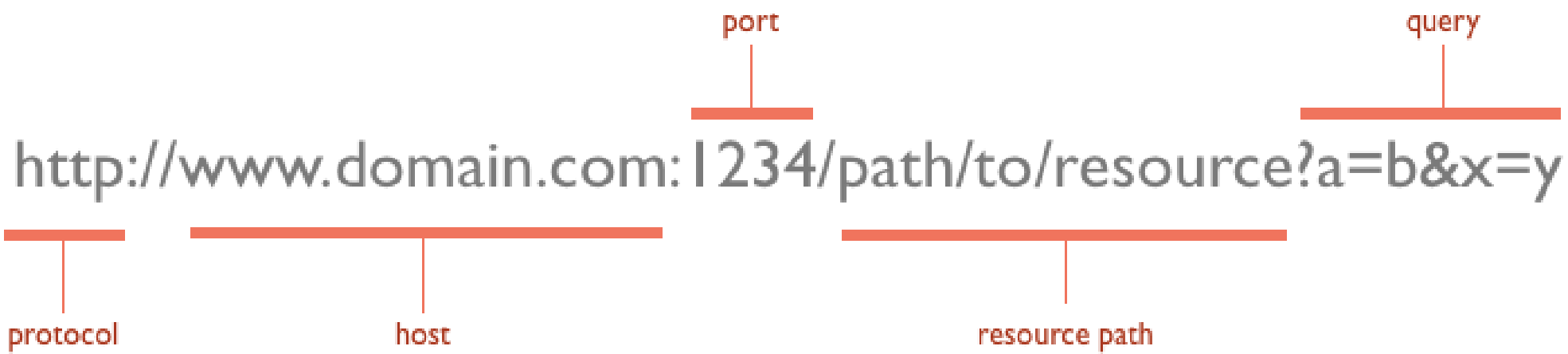
Web APIs

Lecture 13

Dr. Colin Rundel

URLs


port
query
http://www.domain.com:1234/path/to/resource?a=b&x=y
protocol
host
resource path

Query Strings

Provides named parameter(s) and value(s) that modify the behavior of the resulting page.

Format generally follows:

```
?arg1=value1&arg2=value2&arg3=value3
```

Some quick examples,

- `http://lmgtfy.com/?q=hello%20world`
- `http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=1600+Amphitheatre+Parkway`
- `https://nomnom-prod-api.dennys.com/mapbox/geocoding/v5/mapbox.places/raleigh,%20nc.json?types=country,region,postcode,place&country=us,pr,vi,gu,mp,ca`

URL encoding

This is will often be handled automatically by your web browser or other tool, but it is useful to know a bit about what is happening

- Spaces will encoded as '+' or '%20'
- Certain characters are reserved and will be replaced with the percent-encoded version within a URL

!	#	$	&	'	(	)
%21	%23	%24	%26	%27	%28	%29
*	+	,	/	:	;	=
%2A	%2B	%2C	%2F	%3A	%3B	%3D
?	@	[	]			
%3F	%40	%5B	%5D			

- Characters that cannot be converted to the correct charset are replaced with HTML numeric character references (e.g. a Σ would be encoded as Σ)

Examples

```
URLencode("http://lmgtfy.com/?q=hello world")
```

```
[1] "http://lmgtfy.com/?q=hello%20world"
```

```
URLdecode("http://lmgtfy.com/?q=hello%20world")
```

```
[1] "http://lmgtfy.com/?q=hello world"
```

```
URLencode("!#$&'()*+,/:;=?@[]")
```

```
[1] "!#$&'()*+,/:;=?@[]"
```

```
URLencode("!#$&'()*+,/:;=?@[]", reserved = TRUE)
```

```
[1] "%21%23%24%26%27%28%29%2A%2B%2C%2F%3A%3B%3D%3F%40%5B%5D"
```

```
URLencode("!#$&'()*+,/:;=?@[]", reserved = TRUE) |>
  URLdecode()
```

```
[1] "!#$&'()*+,/:;=?@[]"
```

```
URLencode("Σ")
```

```
[1] "%CE%A3"
```

```
URLdecode("%CE%A3")
```

```
[1] "Σ"
```

Examples (httpuv)

```
1 httpuv::encodeURI("http://lmgtfy.com/?q=hello world")
```

```
[1] "http://lmgtfy.com/?q=hello%20world"
```

```
1 httpuv::decodeURI("http://lmgtfy.com/?q=hello%20world")
```

```
[1] "http://lmgtfy.com/?q=hello world"
```

```
1 httpuv::encodeURI("!#$&'()*+,/:;=?@[]")
```

```
[1] "!%23$&'()*+,/:;=?@%5B%5D"
```

```
1 httpuv::encodeURI("!#$&'()*+,/:;=?@[]") |>
2   httpuv::decodeURI()
```

```
[1] "!#$&'()*+,/:;=?@[]"
```

```
1 httpuv::encodeURIComponent("!#$&'()*+,/:;=?@[]")
```

```
[1] "!%23%24%26'()*%2B%2C%2F%3A%3B%3D%3F%40%5B%5D"
```

```
1 httpuv::encodeURIComponent("!#$&'()*+,/:;=?@[]")
2   httpuv::decodeURIComponent()
```

```
[1] "!#$&'()*+,/:;=?@[]"
```

```
1 httpuv::encodeURI("Σ")
```

```
[1] "%CE%A3"
```

```
1 httpuv::decodeURI("%CE%A3")
```

```
[1] "Σ"
```

RESTful APIs

REST

*RE*presentational *S*tate *T*ransfer

- describes an architectural style for web services (not a standard)
- all communication via HTTP requests
- Key features:
  - client–server architecture
  - addressible (specific URL endpoints)
  - stateless (no client information stored between requests)
  - layered / hierarchical
  - cacheability

GitHub API

GitHub provides a REST API that allows you to interact with most of the data available on the website.

There is extensive documentation and a huge number of endpoints to use - almost anything that can be done on the website can also be done via the API.

GitHub REST API

Demo 1 - GitHub API
Basic access

Get a user

List organization repositories

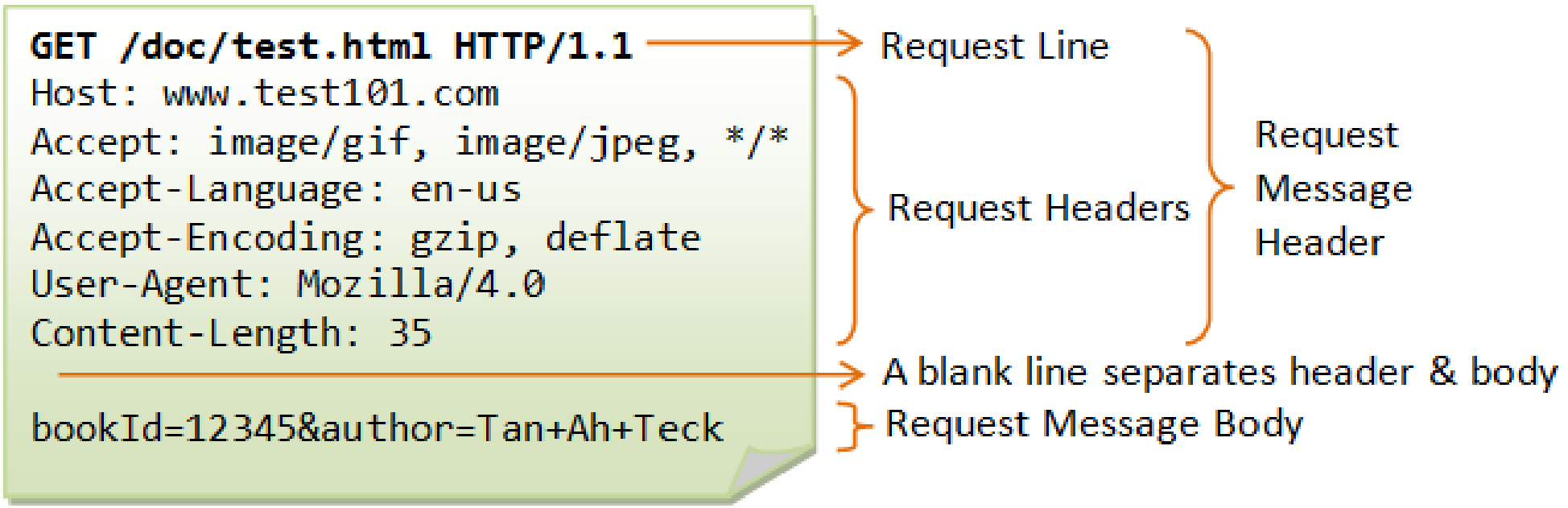
httr2

Background

`httr2` is a package designed around the construction and handling of HTTP requests and responses. It is a rewrite of the `httr` package and includes the following features:

- Pipeable API
- Explicit request object, with support for
  - rate limiting
  - retries
  - OAuth
  - Secrure secret storage
- Explicit response object, with support for
  - error codes / reporting
  - common body encoding (e.g. json, etc.)

Structure of an HTTP Request



HTTP Methods / Verbs

- *GET* - fetch a resource
- *POST* - create a new resource
- *PUT* - full update of a resource
- *PATCH* - partial update of a resource
- *DELETE* - delete a resource.

Less common verbs: *HEAD*, *TRACE*, *OPTIONS*.

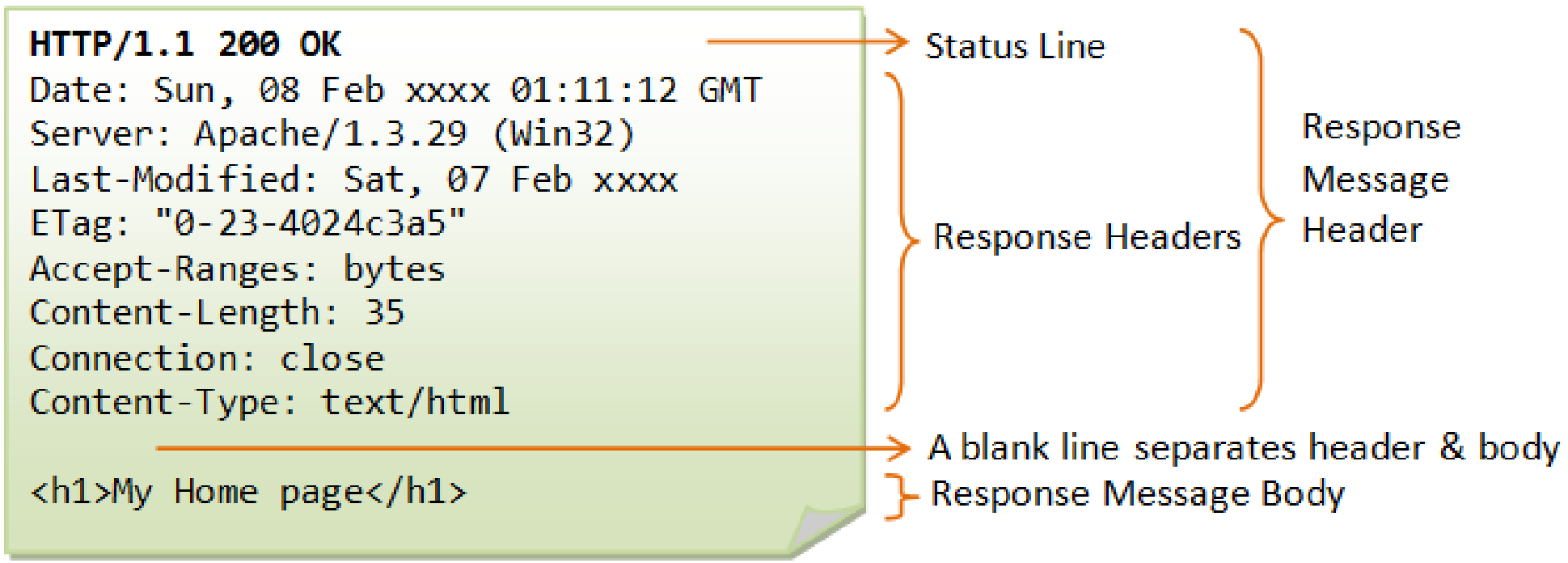
httr2 request objects

A new request object is constructed via `request()` which is then modifed via `req_*()` functions

Some useful functions:

- `request()` - initialize a request object
- `req_method()` - set HTTP method
- `req_url_query()` - add query parameters to URL
- `req_url_*()` - add or modify URL
- `req_body_*()` - set body content (various formats and sources)
- `req_user_agent()` - set user-agent
- `req_dry_run()` - shows the exact request that will be made

Structure of an HTTP Response



Status Codes

- 1xx: Informational Messages
- 2xx: Successful
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

httr2 response objects

Once constructed a request is made via `req_perform()` which returns a response object (the most recent response can also be retrieved via `last_response()`). Content of the response are accessed via the `resp_*()` functions

Some useful functions:

- `resp_status()` - extract HTTP status code
- `resp_status_desc()` - return a text description of the status code
- `resp_content_type()` - extract content type and encoding
- `resp_body_*()` - extract body from a specific format (`json`, `html`, `xml`, etc.)
- `resp_headers()` - extract response headers

Demo 2 - httr2 + GitHub