UNIVERSITY OF
Southampton

UNIVERSITY OF Southampton

Proxies and Caching

COMP3220 Web Infrastructure

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

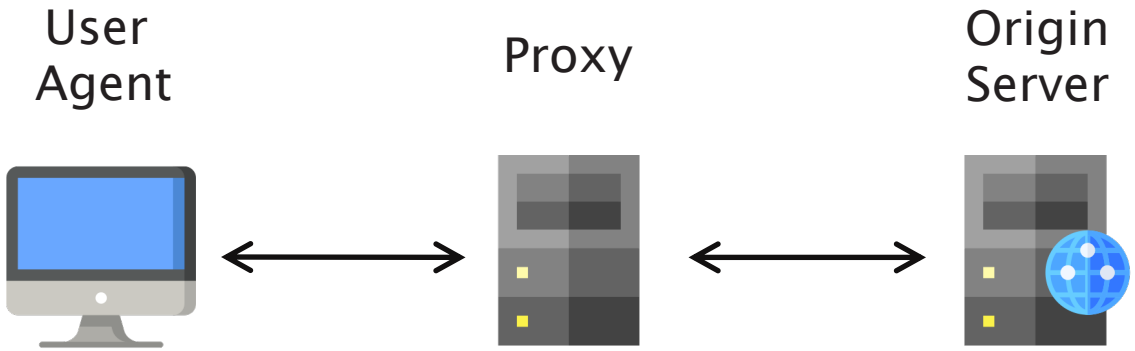
Proxies

Intermediary that acts as server and client

- Receives request from user agent and forwards to origin server
- Returns response to user agent
- Can be chained in sequence

Multiple uses:

- Filtering
- Caching
- Content routing
- Anonymising
- Access control
- Transcoding



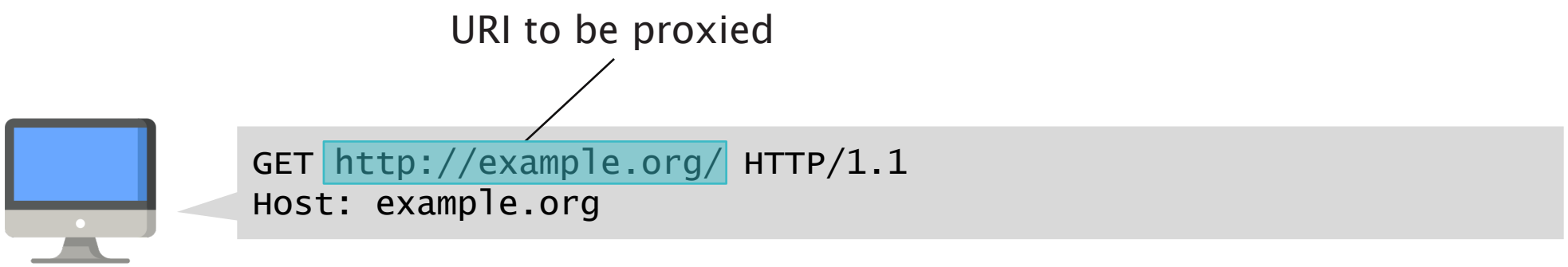
Fielding, R. and Reschke, J. (2014) Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC7230. Available at: https://tools.ietf.org/html/rfc7230

Proxies and HTTP

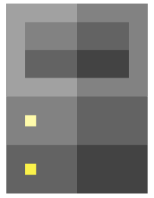
When sending requests via a proxy, user agent replaces the path in the request line with the full URI

Proxies add `Via:` headers to all messages they send

- HTTP version, hostname of proxy
- Append to existing header if chaining proxies



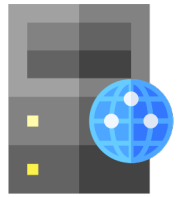
```
GET http://example.org/ HTTP/1.1
Host: example.org
```



```
GET / HTTP/1.1
Host: example.org
Via: HTTP/1.1 proxy.net
```

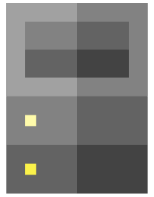
```
HTTP/1.1 200 OK

...
```



```
HTTP/1.1 200 OK
Via: HTTP/1.1 proxy.net

...
```



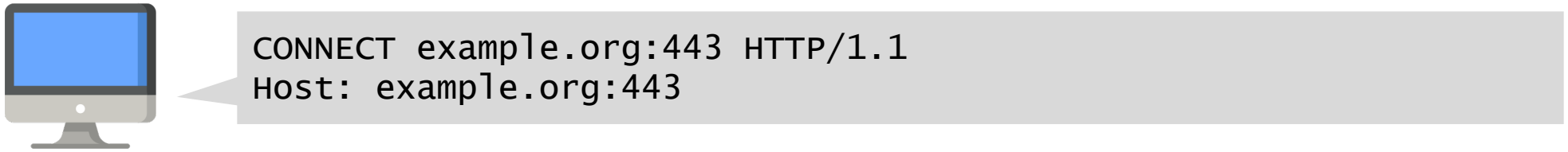
Tunnels

But what about HTTPS?

- TLS guarantees end-to-end encryption of both message body and headers
- Proxies can't understand the request

CONNECT method establishes a tunnel to the destination origin server

- Proxy will forward all subsequent packets in both directions



Proxy Autoconfiguration

`proxy.pac` file containing sandboxed JavaScript defining `FindProxyForURL()`

- Configured in browser (specify URI of `proxy.pac`)
- Configured via DHCP (option 252, containing URI of `proxy.pac`)
- Configured via DNS (looks for host named `wpad`, requests `wpad.dat`)
- Note: common practice, not a (formal) standard

```
function FindProxyForURL(url, host) {
  // ...
}
```

Return values:

- `DIRECT`
- `PROXY` *`host:port`*

Caching

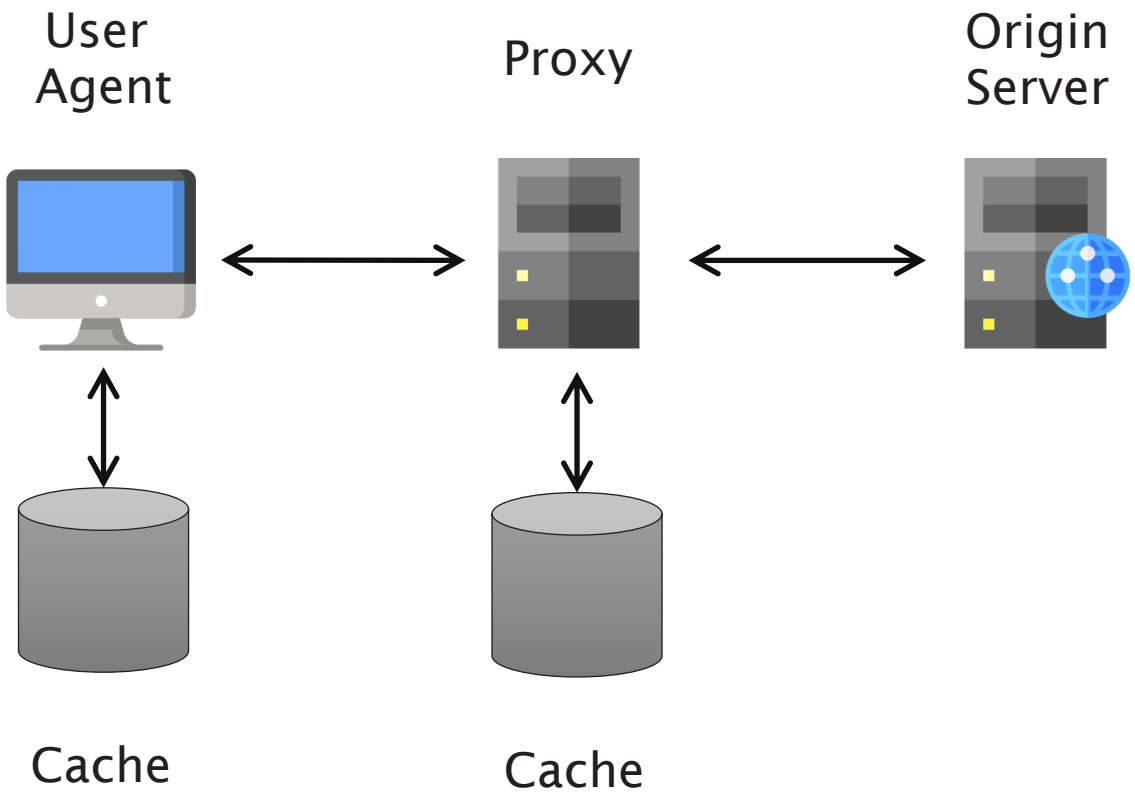
Recall the following REST constraint: response data must be labelled as cacheable or non-cacheable

In practice, both user agents and proxies may cache representations

- User agent private cache
- Proxy shared cache

How do we know what to cache?

How do we avoid stale data?



Fielding, R. et al (2014) *Hypertext Transfer Protocol (HTTP/1.1): Caching*. RFC7234. Available at: https://tools.ietf.org/html/rfc7234

Key concepts

Freshness/staleness

- How long is it since the representation was generated by the origin server?
- Is there a lifetime (or expiration time) associated with the representation?

Revalidation

- If a cache contains a stale representation, we need to refresh it by either:
  1. Checking that it is still the same as the representation served by the origin server, or
  2. Fetching a new (fresh) representation with which to replace it

Cache headers

`Age:` *`seconds`*

- Sent by a proxy server to indicate how long ago a representation was generated by the origin server

`Expires:` *`timestamp`*

- Indicates when a representation should be considered stale

`Cache-Control:` *`directive, directive, ...`*

- Specifies caching behaviour, may be sent by user agents, proxies and origin servers

Request `Cache-Control:` directives

`no-cache`

- Client will not accept cached representations that have not been validated

`no-store`

- No part of this request or response may be cached

`max-age=`*`seconds`*

- Client will not accept cached representations more than *seconds* old

`max-stale=`*`seconds`*

- Client will accept cached representations that are up to *seconds* past their freshness

`min-fresh=`*`seconds`*

- Client will accept cached representations that will remain fresh for at least *seconds*

`no-transform`

- Intermediary must not transform the representation (note: orthogonal to caching)

Response `Cache-Control:` directives

`no-cache`

- Representations may not be reused without being revalidated

`no-store`

- No part of this request or response may be cached

`must-revalidate`

- Stale representations may not be reused without being revalidated

`max-age=`*seconds*

- Representation is to be considered stale after *seconds*

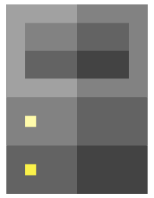
`public`

- Representation may be stored in a shared cache and used for other users

`private`

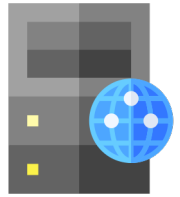
- Representation may not be used for other users

```
GET http://example.org/ HTTP/1.1
Host: example.org
```

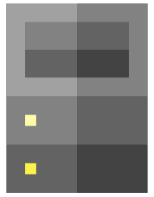


```
GET / HTTP/1.1
Host: example.org
```

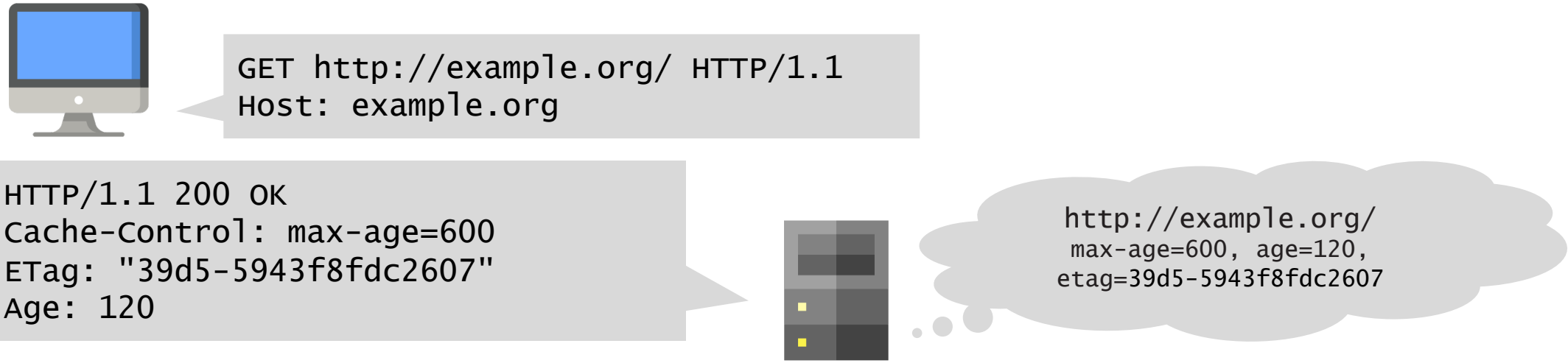
```
HTTP/1.1 200 OK
Cache-Control: max-age=600
ETag: "39d5-5943f8fdc2607"
```



```
HTTP/1.1 200 OK
Cache-Control: max-age=600
Age: 0
ETag: "39d5-5943f8fdc2607"
```

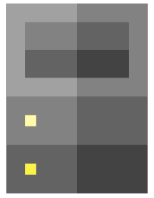


```
http://example.org/
max-age=600, age=0,
etag=39d5-5943f8fdc2607
```


GET http://example.org/ HTTP/1.1
Host: example.org
HTTP/1.1 200 OK
Cache-Control: max-age=600
ETag: "39d5-5943f8fdc2607"
Age: 120
http://example.org/
max-age=600, age=120,
etag=39d5-5943f8fdc2607

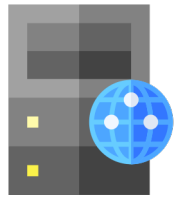
```
GET http://example.org/ HTTP/1.1
Host: example.org
```

```
http://example.org/
max-age=600, age=720,
etag=39d5-5943f8fdc2607
```

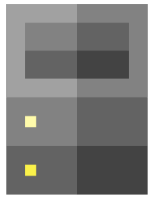


```
GET / HTTP/1.1
Host: example.org
If-None-Match: "39d5-5943f8fdc2607"
```

```
HTTP/1.1 200 OK
Cache-Control: max-age=600
ETag: "4b4f-77a6c3ab117a2"
```



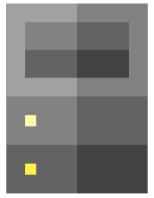
```
HTTP/1.1 200 OK
Cache-Control: max-age=600
Age: 0
ETag: "4b4f-77a6c3ab117a2"
```



```
http://example.org/
max-age=600, age=0,
etag=4b4f-77a6c3ab117a2
```

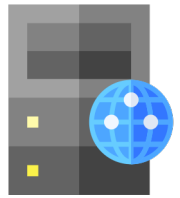
```
GET http://example.org/ HTTP/1.1
Host: example.org
```

```
http://example.org/
max-age=600, age=720,
etag=39d5-5943f8fdc2607
```

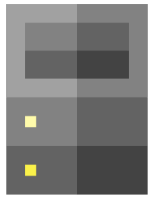


```
GET / HTTP/1.1
Host: example.org
If-None-Match: "39d5-5943f8fdc2607"
```

```
HTTP/1.1 304 Not Modified
Cache-Control: max-age=600
ETag: "39d5-5943f8fdc2607"
```



```
HTTP/1.1 200 OK
Cache-Control: max-age=600
Age: 0
ETag: "39d5-5943f8fdc2607"
```



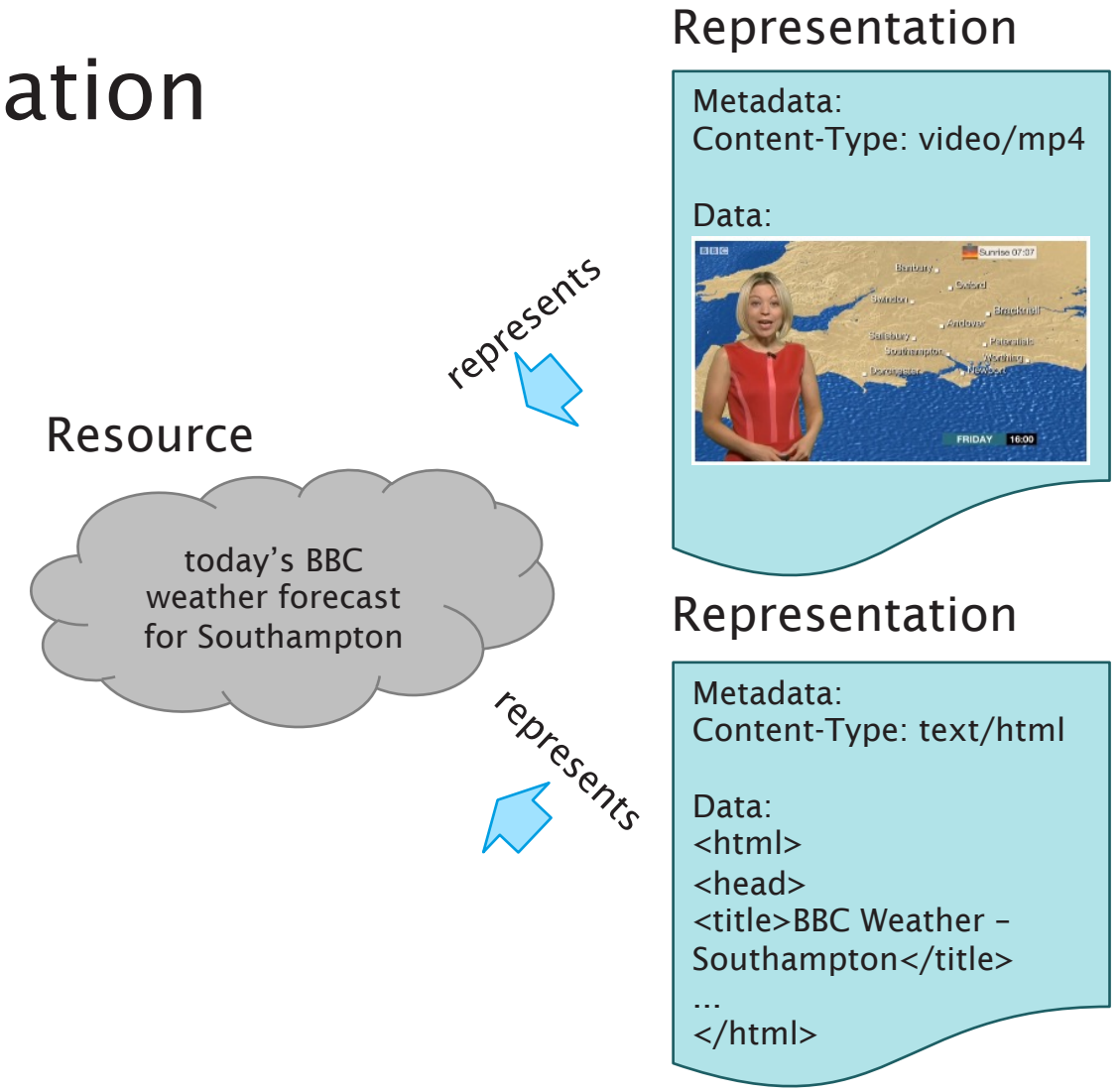
```
http://example.org/
max-age=600, age=0,
etag=39d5-5943f8fdc2607
```

Caching and content negotiation

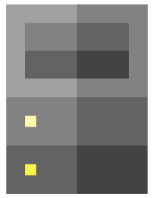
We're caching representations, but there may be multiple representations of a given resource

How do when distinguish between the different cached representations of a resource?

Cached representations record the values of the headers listed in the `Vary:` header

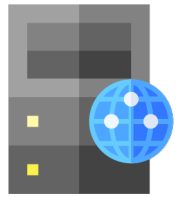


```
GET http://example.org/ HTTP/1.1
Host: example.org
Accept: text/html
```

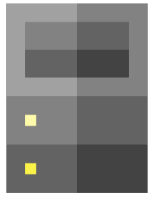


```
GET / HTTP/1.1
Host: example.org
Accept: text/html
```

```
HTTP/1.1 200 OK
Location: http://example.org/
Content-Type: text/html
Vary: accept, accept-language
```



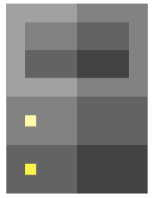
```
HTTP/1.1 200 OK
Content-Type: text/html
Vary: accept, accept-language
```



```
http://example.org/
max-age=600, age=0, text/html
```

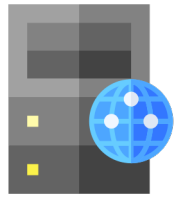
```
GET http://example.org/ HTTP/1.1
Host: example.org
Accept: text/plain
```

http://example.org/
max-age=600, age=120, text/html

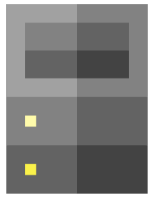


```
GET / HTTP/1.1
Host: example.org
Accept: text/plain
```

```
HTTP/1.1 200 OK
Content-Type: text/plain
Vary: accept, accept-language
```



```
HTTP/1.1 200 OK
Content-Type: text/plain
Vary: accept, accept-language
```



http://example.org/
max-age=600, age=120, text/html
max-age=600, age=0, text/plain

Further reading

Fielding, R. and Reschke, J. (2014) *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC7230.

https://tools.ietf.org/html/rfc7230

Fielding, R. et al (2014) *Hypertext Transfer Protocol (HTTP/1.1): Caching*. RFC7234.

https://tools.ietf.org/html/rfc7234

UNIVERSITY OF Southampton

Next lecture: Authentication and Authorisation