University of St Andrews

CS2003: Internet and the Web

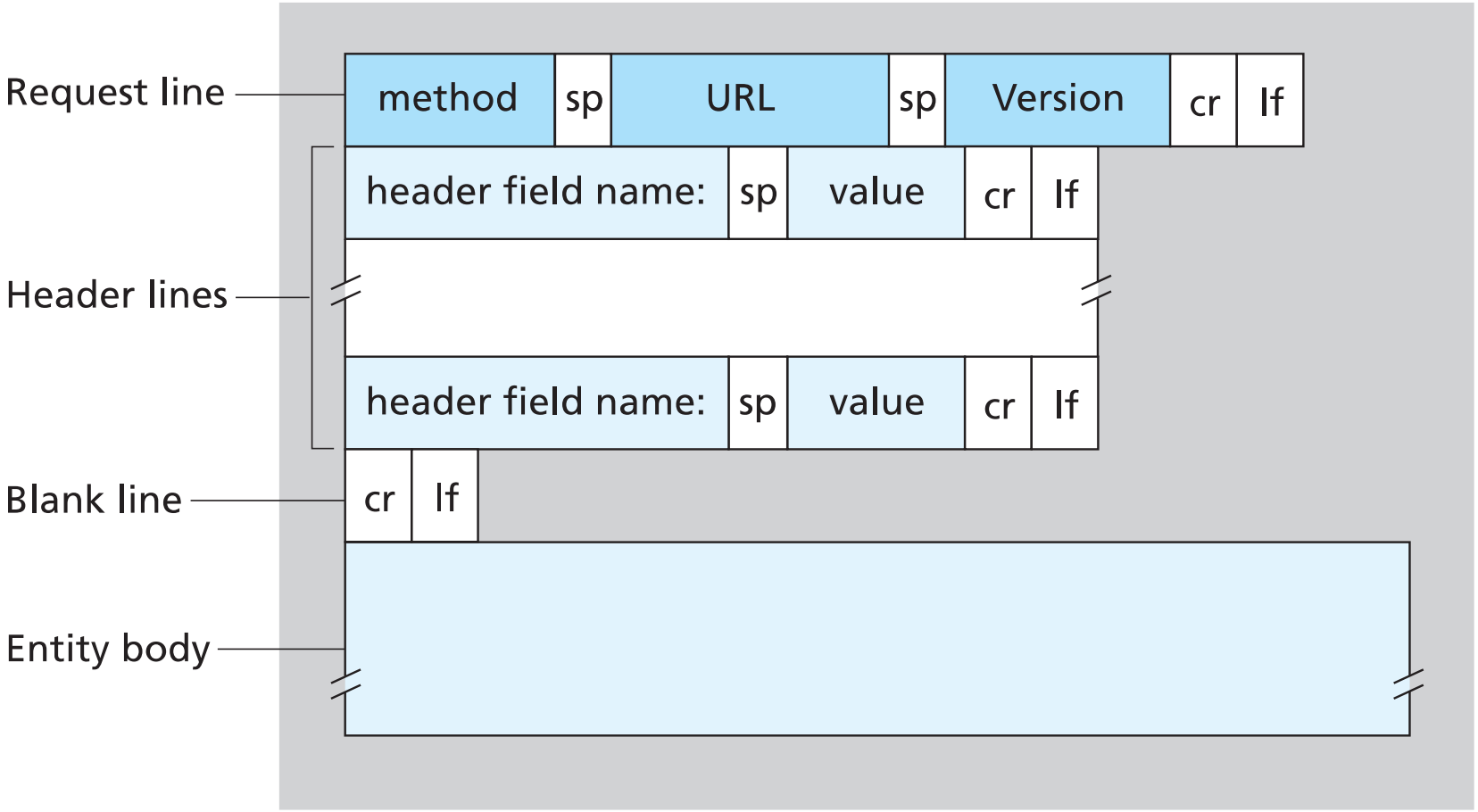
HTTP

Hyper Text Transfer Protocol (HTTP)

University of St Andrews

- HTTP is the protocol for the World Wide Web.
- Several versions: we will focus on HTTP/1.1:
  - RFC7230 (PS) (also RFC7231/2/3/4/5 (PS) !)
- A client-server protocol.
- Application level protocol (running over TCP).
- Session: stateless (single request, single response).
- Presentation (Data representation):
  - Printable strings for message headers (and content).
  - HTML + CSS (+ other media) and content / payload

HTTP request message format (1)



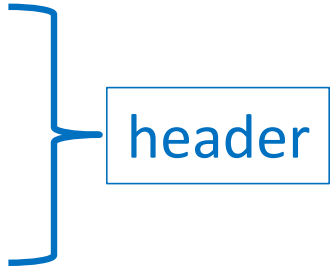
(From section 2.2.3 of Kurose and Ross, 7th ed.)

HTTP request message format (2)

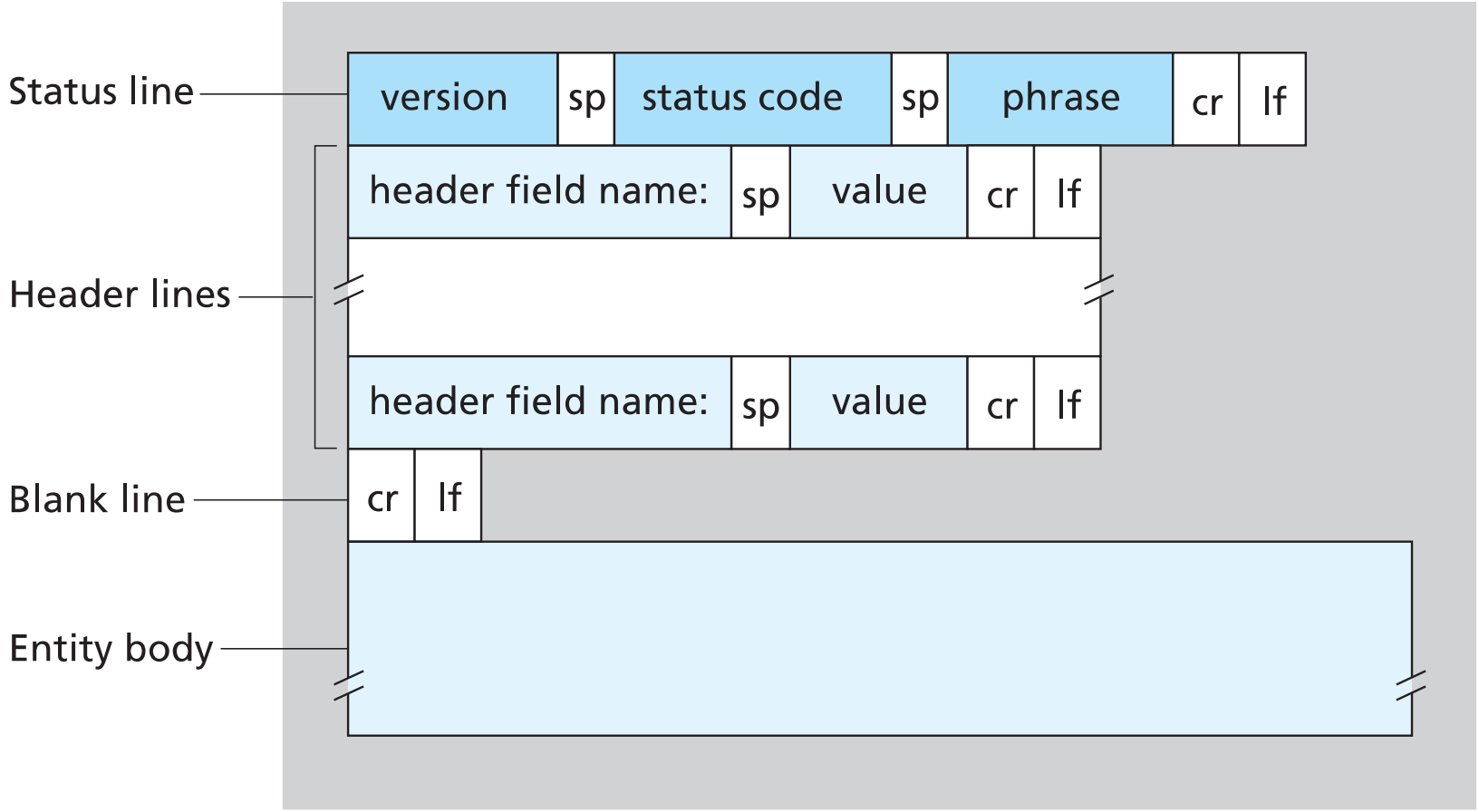
```
curl -v https://studres.cs.st-andrews.ac.uk/CS2003/

. . . <TLS exchange omitted, curl output tidied up> . . .

GET /CS2003/ HTTP/1.1
Host: studres.cs.st-andrews.ac.uk
User-Agent: curl/7.61.1
Accept: */*
```



HTTP response format (1)



(From section 2.2.3 of Kurose and Ross, 7th ed.)

HTTP response format (2)

```
curl -v https://studres.cs.st-andrews.ac.uk/CS2003/

. . . <TLS exchange omitted, curl output tidied up> . . .

HTTP/1.1 200 OK
Date: Mon, 26 Oct 2020 11:20:53 GMT
Server: Apache
Strict-Transport-Security: max-age=15552000; includeSubdomains;
Content-Length: 1868
Content-Type: text/html;charset=ISO-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of CS2003</title>
   . . .
```

header

payload

Content-type – Media types

- In request:

  `Accept: */*`

  Accepts anything.

- In response:

  `Content-type: A/B`

  Type A

  Subtype B

  Type of media / content in the body.

- Examples:

  `text/html`
  `text/css`
  `text/plain`

  `image/jpeg`
  `image/gif`
  `image/png`

  `video/mp4`

HTTP: text-based

- Presentation is text:
  - Headers etc are printable strings
  - Some text-based protocols may require binary data to be encoded in a printable format (e.g. base64)
- Contrast with IP, TCP: binary
  - More efficient?
  - Higher performance?
  - What about development, debugging?
  - What about extending?

Common methods

- GET:
  - Request a specific page or object.
  - Request has header only, no message body.
- HEAD:
  - Request only the header, not the object itself.
  - Can check timestamp against local cache.
  - Request has header only, no message body.
- POST:
  - Send form data to server.
  - Request body contains form data.

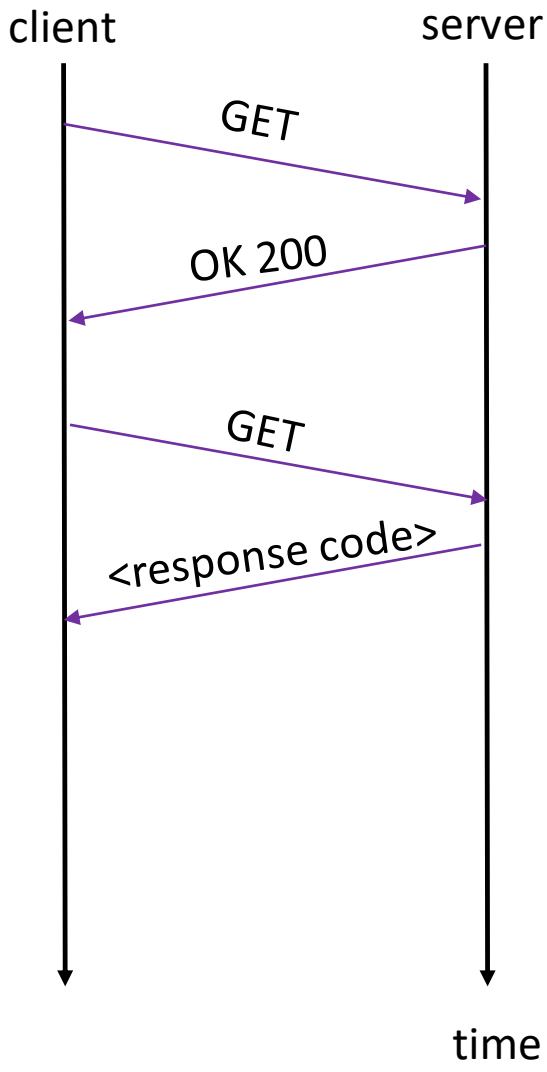
HTTP response codes

Response code types

1xx	Informational
2xx	Successful
3xx	Redirection
4xx	Client error
5xx	Server error

Response code examples

200	OK
301	Moved permanently
404	Not found
502	Bad Gateway



HTTP is stateless

- Each client request is treated independently.
- Server does not maintain state or history of any previous requests from the client:
  - The notion of an **application-level session** must be built to the application that is using HTTP.
- Different applications maintain state in different ways, depending on the nature of the application:
  - e.g. **application layer protocol** on top of HTTP.
  - e.g. DNS over HTTPS (DoH), RFC8484 (PS)
  - **lots** of applications! Some call HTTP the new "narrow waist"

Fetching a “whole page”

- The client must fetch all page contents:
  - Not part of HTTP, e.g. `curl` only fetches the URL requested, not any other linked objects.
- Client must:
  - Fetch “main” HTML page.
  - Look for additional links to objects in HTML, e.g.:

```
<link rel="stylesheet” . . . >
<img . . . >
```

    (and others also), then fetch each with separate GET.
  - Render / display as required.

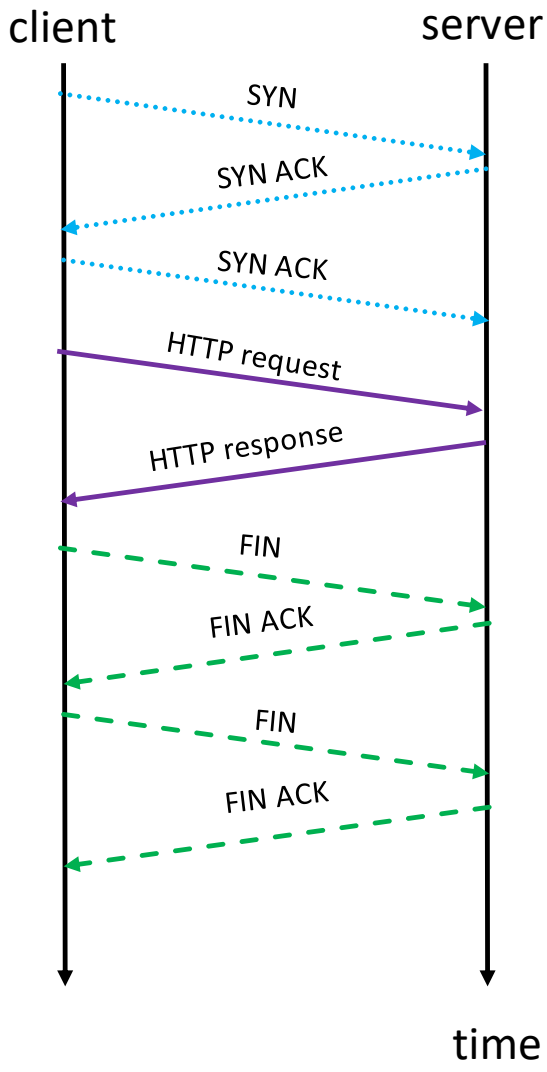
HTTP and TCP (1)

University of St Andrews

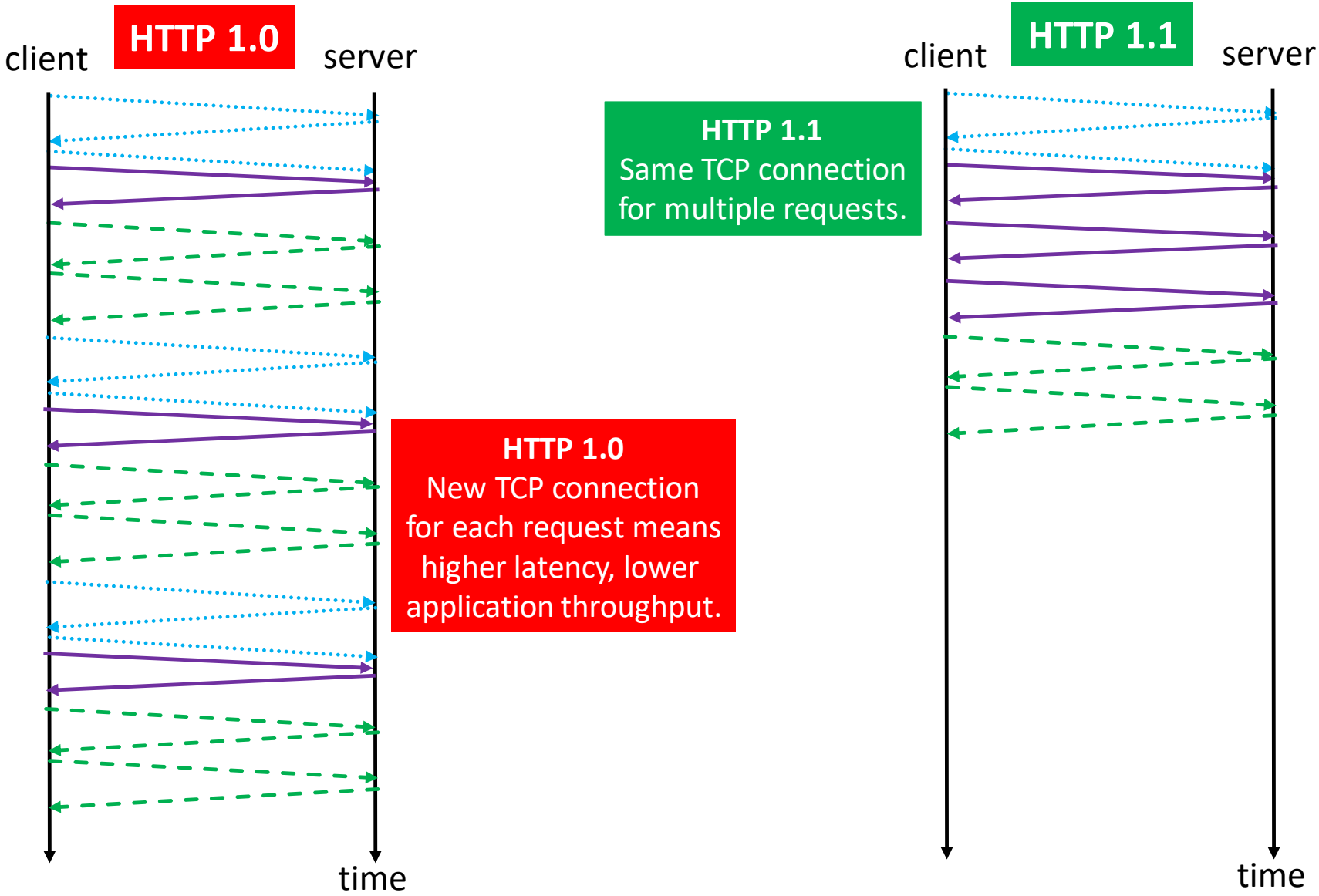
- A TCP connection must first be established.
- Then a HTTP request can be sent.
- HTTP 1.0:
  - New TCP for every object linked on page.
    `Connection: close`
- HTTP 1.1:
  - TCP connection kept open to fetch multiple objects.
    `Connection: keep-alive`
  - Greatly reduced overhead compared to HTTP 1.0.

HTTP and TCP (2)

- TCP connection required for HTTP request.
  - TCP connection set-up.
  - HTTP request.
  - HTTP response.
  - TCP connection release.
- TCP connection overhead:
  - Latency in requests.
  - Impacts client and server.



HTTP 1.0 vs HTTP 1.1



HTTP/2

University of St Andrews

- Supported by many server systems:
  - RFC7540/1
  - Derived from Google SPDY protocol
  - Some clients still use HTTP 1.1 with TLS.
- Improved security for HTTP communications:
  - HTTP/2 defaults to use HTTP over TLS (HTTPS) (all deployed HTTP/2 uses TLS).
- Binary encoding of protocol and content:
  - No need to encode binary data to printable strings.
  - (Header compression.)
- Support for request multiplexing and pipelining:
  - Single TCP connection used for parallel requests.
  - Can cause problems since single connection can be blocked ("head-of-line blocking")
- Server PUSH:
  - Server pre-emptively sends objects linked on page (useful for caching).

HTTP/3

- (Still being developed, not yet widely supported.)
- Will not use TCP, HTTP/3 will run over QUIC (which runs over UDP):
  - https://tools.ietf.org/html/draft-ietf-quic-http
  - (QUIC is another Google-originated protocol)
- Faster connection set-up (lower latency).
- Built-in security (does not use TLS).
- Many other features as for HTTP/2:
  - Binary encoding of data.
  - Multiplexing and pipelining of requests.
  - Push for caching.

Caching (1)

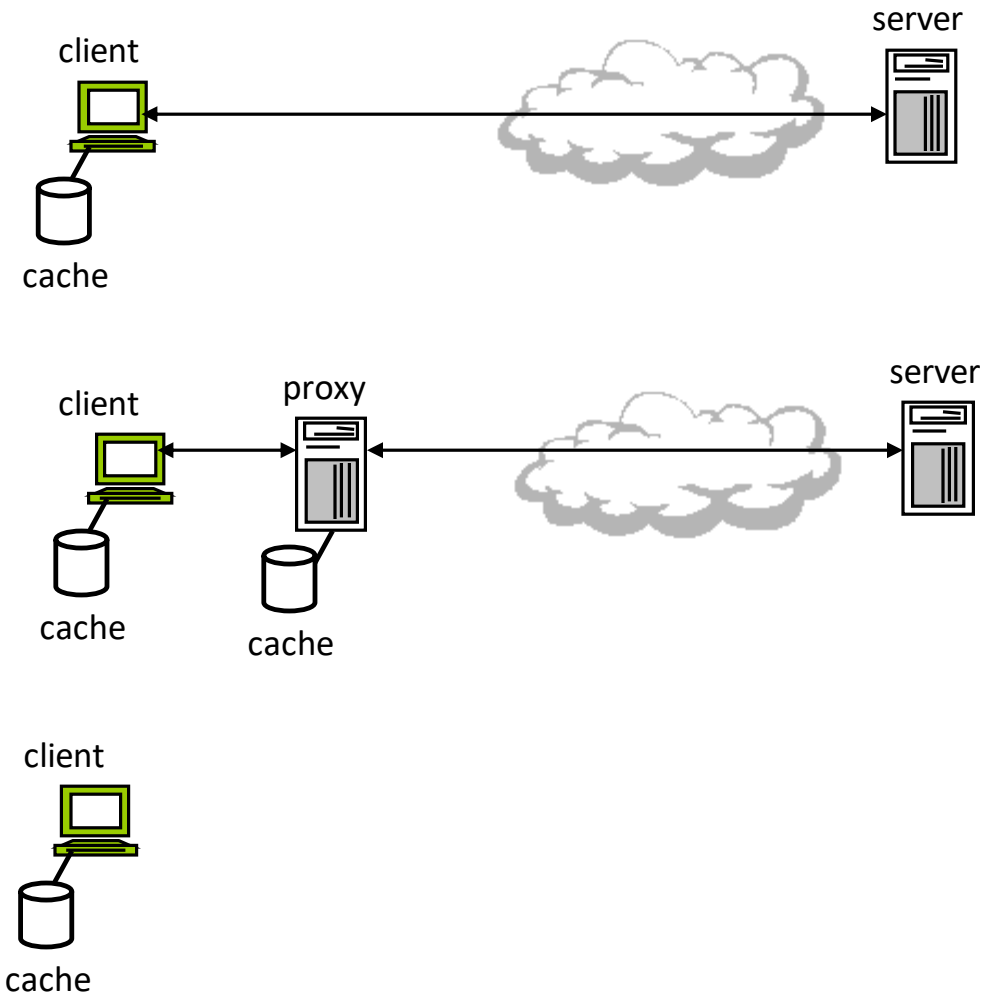
University of St Andrews

- A **cache** is a performance optimisation tool:
  - a cache is a **local** ("conveniently-placed") copy of data.
- A cache can improve performance:
  - data is more readily available because:
    - » (i) it is "closer" to where it needs to be for use (**spatial locality**).
    - » (ii) it can be copied to cache ahead of time (**temporal locality**).
- **Any data in the cache is a copy!**
  - if the original data changes, the cached version is "stale"!
  - stale data could perturb the behaviour of the application.
- What if content is **dynamically generated**?

Caching (2)

- Need mechanism and protocol to manage cache:
  - "local" storage: **privacy**!
- Should not use stale information:
  - need to check when original has changed.
- Share cached information?
  - improve availability of data.
  - reduce load on network and server.
  - **security and privacy**!
- Use pre-emptive caching?
  - reduce wait times for users.

Caching, computation & processing (1)



- Client and server need to communicate:
  - input from user
  - feedback to user
- Data has to be processed:
  - client-side processing
  - server-side processing
  - depends on application
- Proxy does not normally undertake processing:
  - but may be used for site-wide policy.
- **Security and privacy**!

Caching, computation & processing (2)

University of St Andrews

Client-side

- Local processing , fast response.
- Takes load off server.
- Reduces network load.
- May incur local storage and processing overheads.
- “Private” to client:
  - Potential security and privacy issues.

Server-side (proxy)

- Caching provides benefit for whole site with access to data.
- Common content generated on demand.
- Higher overhead at server for each page/document for content generation.
- Potential security and privacy issues.

Cache control in HTTP/1.1

University of St Andrews

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control
- Many possibilities: fine-grained control, e.g.:
- Example header fields from **Requester**:

```
Cache-Control: max-age=<seconds>
Cache-Control: min-fresh=<seconds>
Cache-Control: max-stale[=<seconds>]
(Other possibilities also ...)
```

Example header fields from **Responder**:

```
Cache-Control: must-revalidate
Cache-Control: no-cache
Cache-Control: no-store
Cache-Control: public
Cache-Control: private
Cache-Control: proxy-revalidate
(Other possibilities also ...)
```

Summary

- HTTP 1.1:
  - HTTP message format.
  - HTTP client-server interaction.
  - Use of TCP.
- Summary of HTTP/2 and HTTP/3:
  - HTTP/2: HTTP / TLS / TCP.
  - HTTP/3: HTTP / QUIC / UDP.
- Cache control in HTTP/1.1
- Further reading: Peterson & Davie Ch 9.1 and also the *Perspective: HTTP is the New Narrow Waist* at the end of Ch 5; Kurose & Ross Ch 2.2