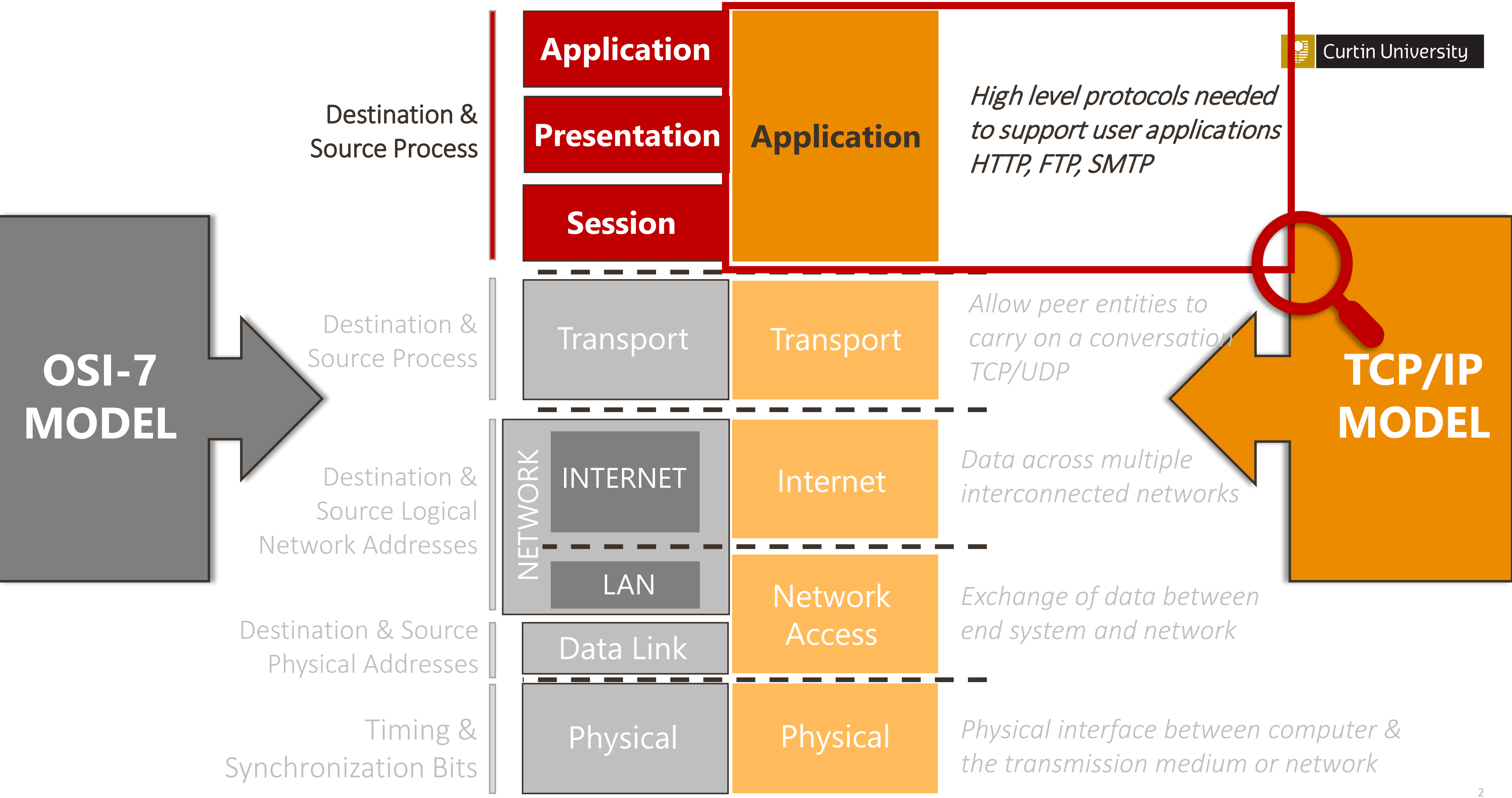
Curtin University

# Application Layer I

Prof. Ling Li | Dr. Nadith Pathirage | Lecture 09

Semester 1, 2021

Curtin University



OSI-7 MODEL

TCP/IP MODEL

Destination & Source Process

Application

Presentation

Session

Application

*High level protocols needed to support user applications HTTP, FTP, SMTP*

Destination & Source Process

Transport

Transport

*Allow peer entities to carry on a conversation TCP/UDP*

Destination & Source Logical Network Addresses

NETWORK

INTERNET

LAN

Internet

*Data across multiple interconnected networks*

Destination & Source Physical Addresses

Data Link

Network Access

*Exchange of data between end system and network*

Timing & Synchronization Bits

Physical

Physical

*Physical interface between computer & the transmission medium or network*

# Application Layer

Defines communications protocols and interface methods used in process-to-process communications

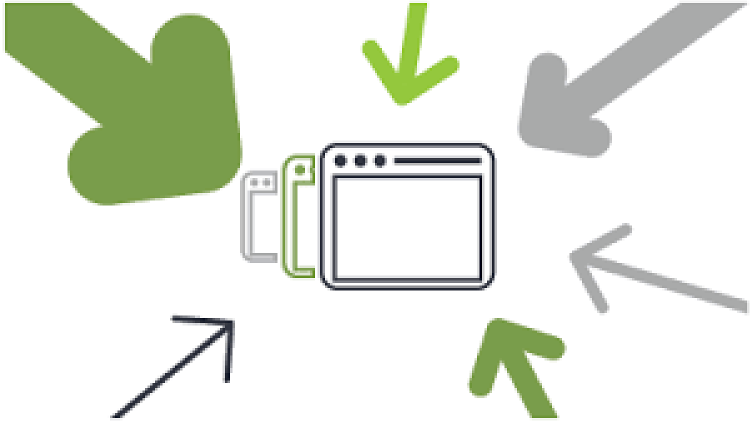
# Overview

- **Network Application Architectures**
  - Client-Server
  - Peer-to-peer
- **Application Layer Protocols**
  - Telnet
  - FTP
  - HTTP
  - SMTP/POP3/IMAP
  - DHCP

# Application Layer

Curtin University

- All the communication applications / processes
- **Layers below** are there **to provide reliable transport**
- What we appreciate is how these applications are build on **top of the lower layers**
- Application layer in TCP/IP mainly cover the Session, Presentation, and Application Layers of the OSI Reference Model



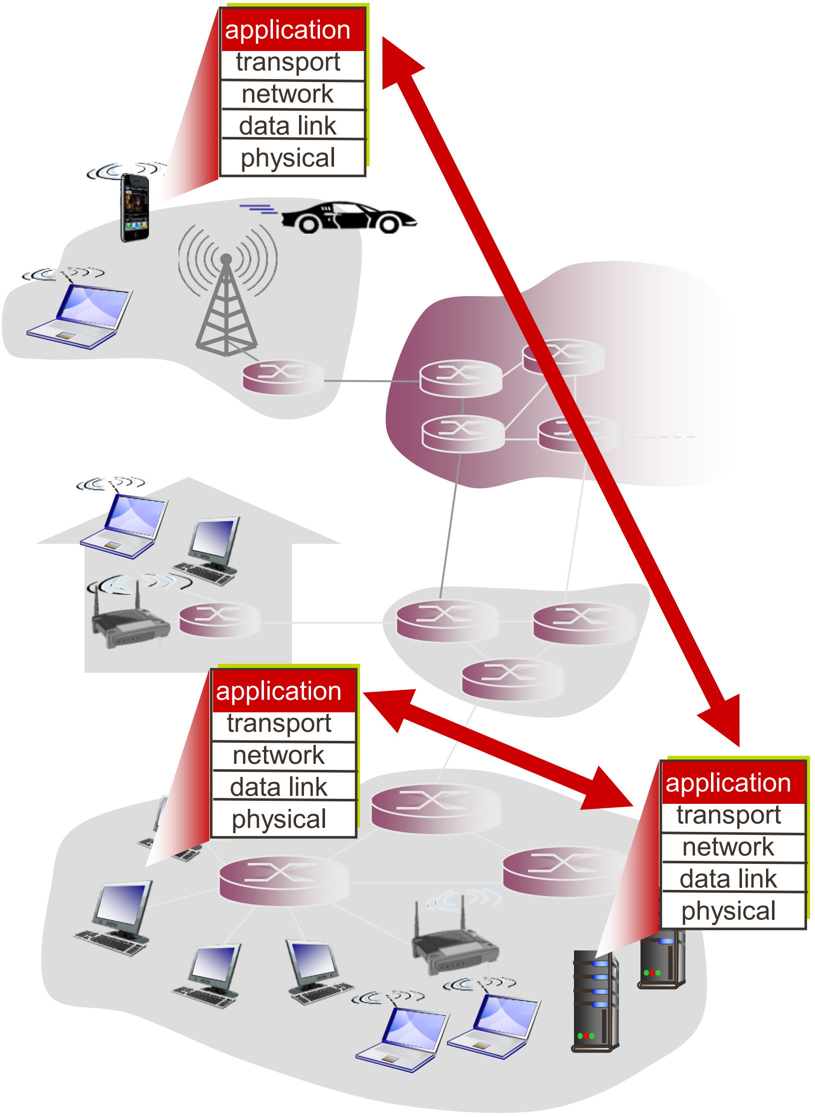
# Some Applications

- E-mail
- Web
- Remote login
- P2P file sharing
- Streaming stored video
- Voice over IP e.g. Skype
- Real-time video conferencing
- Social networking
- Search
- Multi-user network games

# Creating a Network Application

- **Write programs** that:
  - ✓ run **on** (different) **end systems**
  - ✓ communicate over network
  - ✓ i.e. web server software <-> browser software

- No need to write software for network-core devices
  - network-core devices do not run user applications
  - applications on end systems allows for rapid app development, propagation



# App-layer Protocol Defines

- **Message Type** *exchanged*
  - e.g., request, response
- **Message Syntax**
  - what fields in messages & how fields are delineated
- **Message Semantics**
  - meaning of information in fields
- **Rules** for **when and how** processes **send & respond** to messages

**Open Protocols:**

- defined in RFCs
- allows for interoperability
- e.g., **HTTP**, **SMTP**

**Proprietary Protocols:**

- i.e. **Skype**

# Transport Services Required

- **Data integrity**
  - some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
  - other apps (e.g., audio) can tolerate some loss
- **Timing**
  - some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”
- **Throughput**
  - some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
  - other apps (“elastic apps”) make use of whatever throughput they get
- **Security**
  - encryption, data integrity, …

# Transport Services vs App

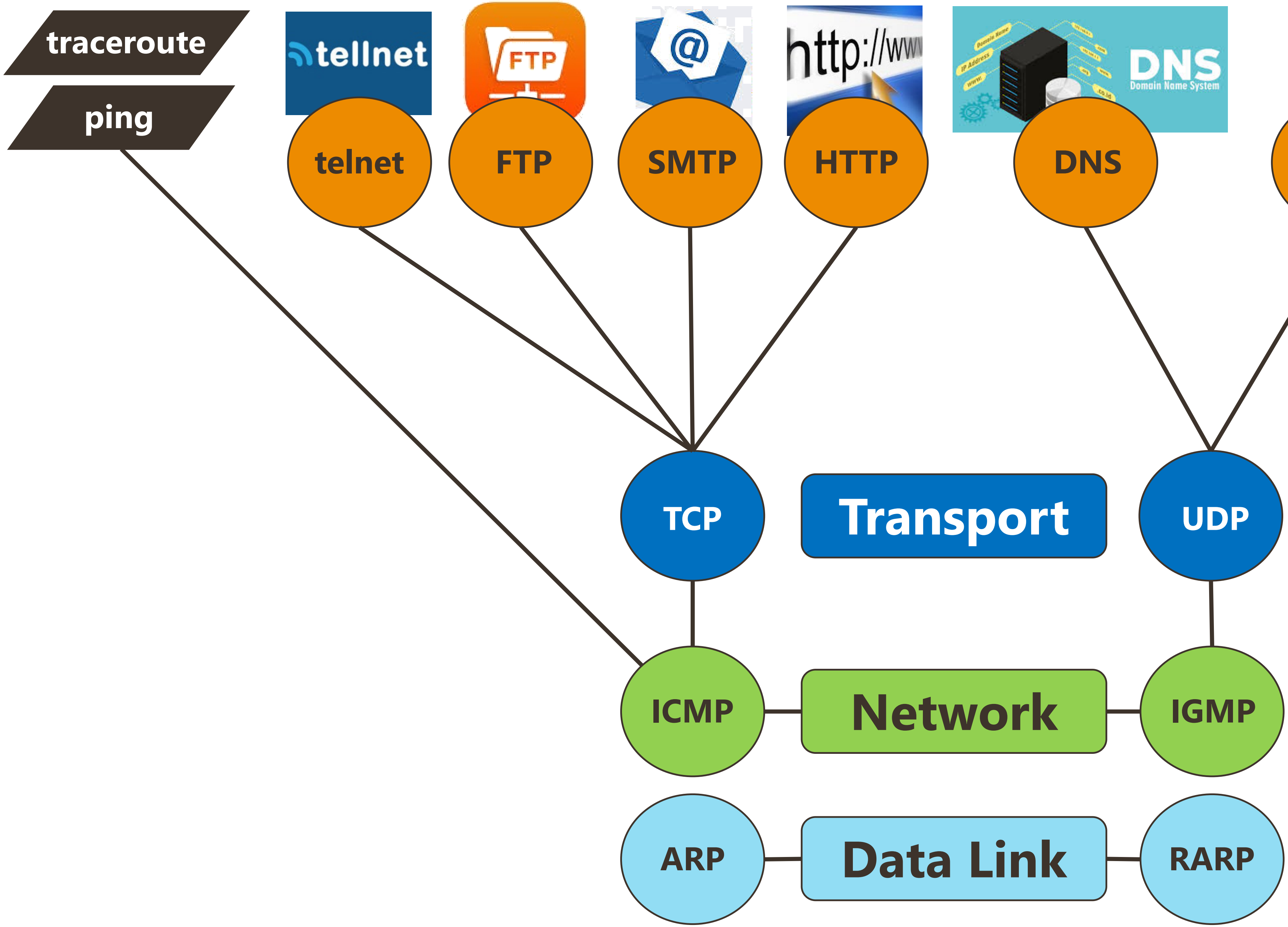
| Application | Data Loss | Throughput | Time Sensitive |
|---|---|---|---|
| **File Transfer** | No loss | Elastic | No |
| **E-mail** | No loss | Elastic | No |
| **Web Documents** | No loss | Elastic | No |
| **Real-time Audio/Video** | Loss-tolerant | Audio: 5kbps- 1Mbps<br>Video: 10kbps- 5Mbps | Yes, 100’s msec |
| **Stored Audio/Video** | Loss-tolerant | Same as above | Yes, few secs |
| **Interactive Games** | Loss-tolerant | Few kbps | Yes, 100’s |
| **Text Messaging** | No loss | Elastic | msec (yes and no) |

# Apps **underlying TCP Protocols**

Curtin University

| **application** | **application layer protocol** | **underlying transport protocol** |
|---|---|---|
| e-mail | SMTP [RFC 2821] | **TCP** |
| remote terminal access | Telnet [RFC 854] | **TCP** |
| Web | HTTP [RFC 2616] | **TCP** |
| file transfer | FTP [RFC 959] | **TCP** |
| streaming multimedia | HTTP (e.g., YouTube), RTP [RFC 1889] | **TCP or UDP** |
| Internet telephony | SIP, RTP, proprietary (e.g., Skype) | **TCP or UDP** |

Curtin University

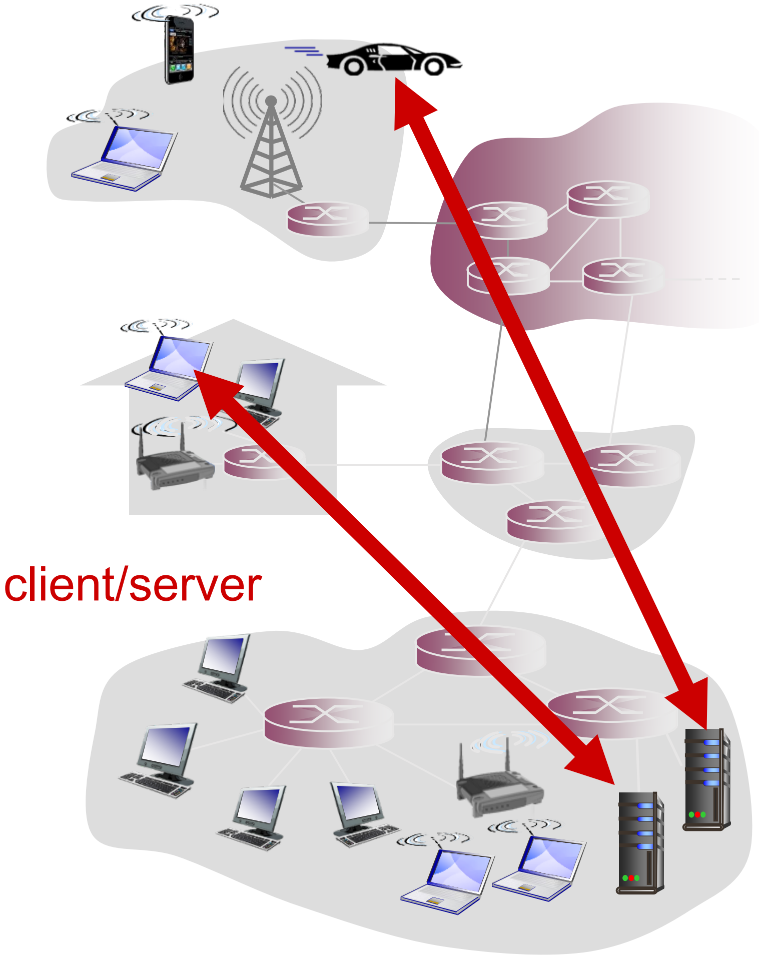


# Secure Communication

- **TCP & UDP**
  - no encryption
  - cleartext passwords sent into port (socket) traverse Internet in clear text

- **SSL**
  - provides encrypted TCP connection
  - data integrity
  - end-point authentication
- **SSL is at app layer**
  - Apps use SSL libraries, which “talk” to TCP

# Client-Server Architecture

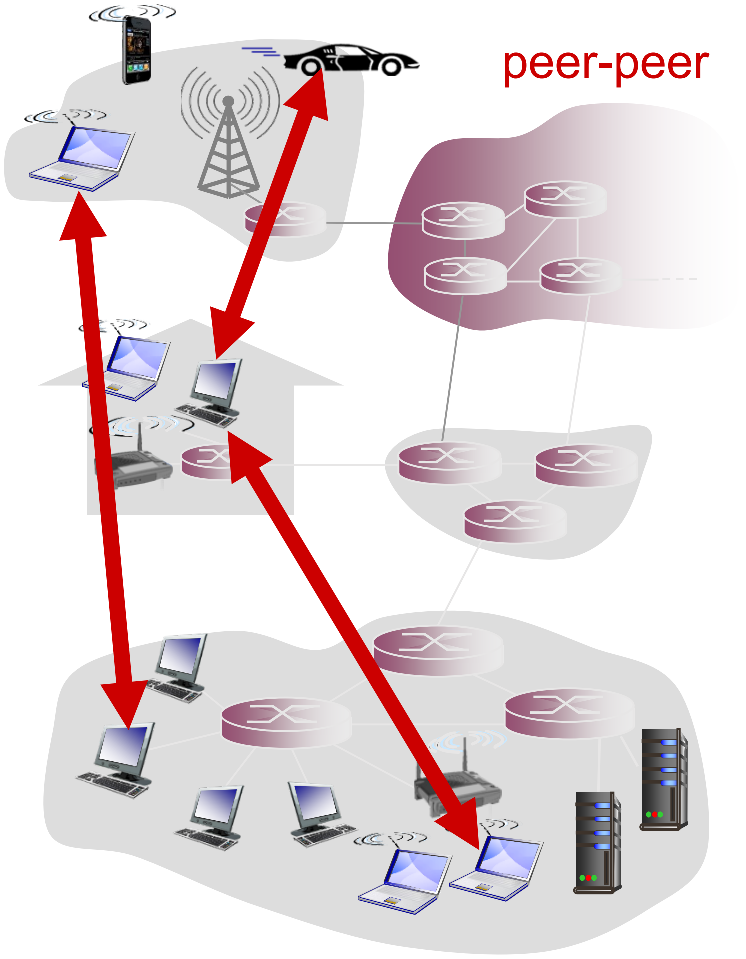


- **Server:**
  - ✓ always-on host
  - ✓ permanent IP address
  - ✓ data centers for scaling
- **Clients:**
  - ✓ communicate with server
  - ✓ may be intermittently connected
  - ✓ may have dynamic IP addresses
  - ✓ do not communicate directly with each other

# P2P Architecture

Curtin University

- **No always-on server**
- **Arbitrary end systems** directly communicate
- Peers request service from other peers, provide service in return to other peers
  - ✓ ***self scalability*** *– new peers bring new service capacity, as well as new service demands*
- Peers are intermittently connected and change IP addresses
  - ✓ ***complex management***



Curtin University

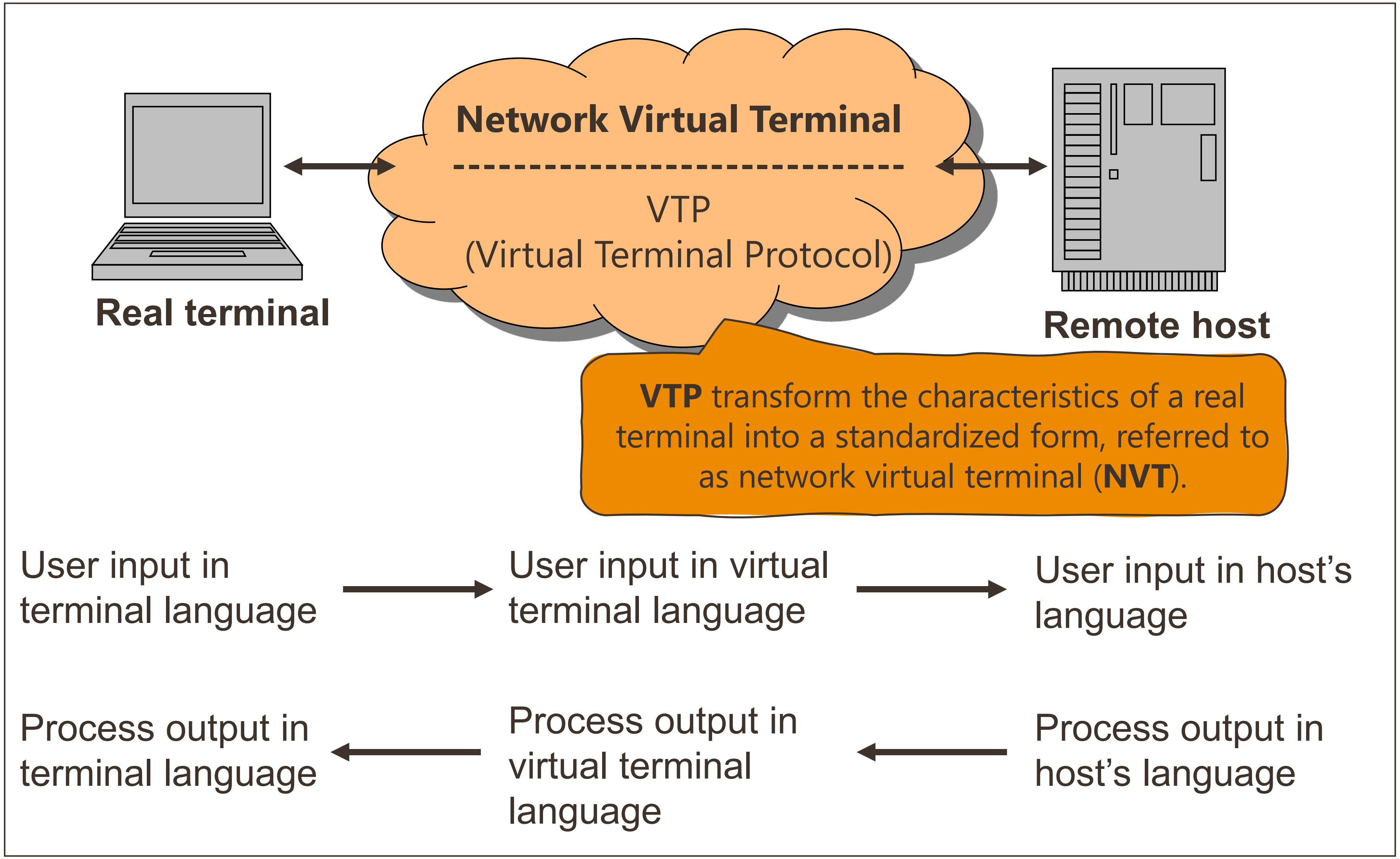
# Application Layer Protocols - Telnet

- Fundamentals
- NVT, VTP
- Connections
- Highlights

# Terminal Access – **Telnet**

- One of the oldest application (1969)
- Basis of many newer protocols
- Telnet is a **remote logon facility** based on the use of a virtual terminal protocol (**VTP**) and a network virtual terminal (**NVT**).
- Both real terminal's characteristics and a host's representation of a terminal are mapped into a network virtual terminal for data transfer

# Using TCP connection, Telnet can be used between two terminals, two processes, or a terminal and a process



**VTP** transform the characteristics of a real terminal into a standardized form, referred to as network virtual terminal (**NVT**).

User input in terminal language → User input in virtual terminal language → User input in host's language

Process output in terminal language ← Process output in virtual terminal language ← Process output in host's language

*Network Virtual Terminal Concept (Stallings)*

# Telnet Transfer Protocol

Curtin University

- **Data** sent **half-duplex**
- **Terminal-to-process:**
  - newline signifies end of user input.
- **Process-to-terminal:**
  - Telnet **Go Ahead Command** is used

    *(Returns the prompt to the user)*
- Underlying TCP **full duplex**
  - **Control Signals** sent any time regardless of current data direction

- **Data** are sent as a **stream of 8-bit bytes**
  - no other formatting to the data
  - **control** data and other non-data information are sent as **telnet commands**:
    - **Interrupt process** (IP) – code 244
    - **Break** (BRK) – code 243
    - **Interpret as Command** (IAC) – code 255

Curtin University

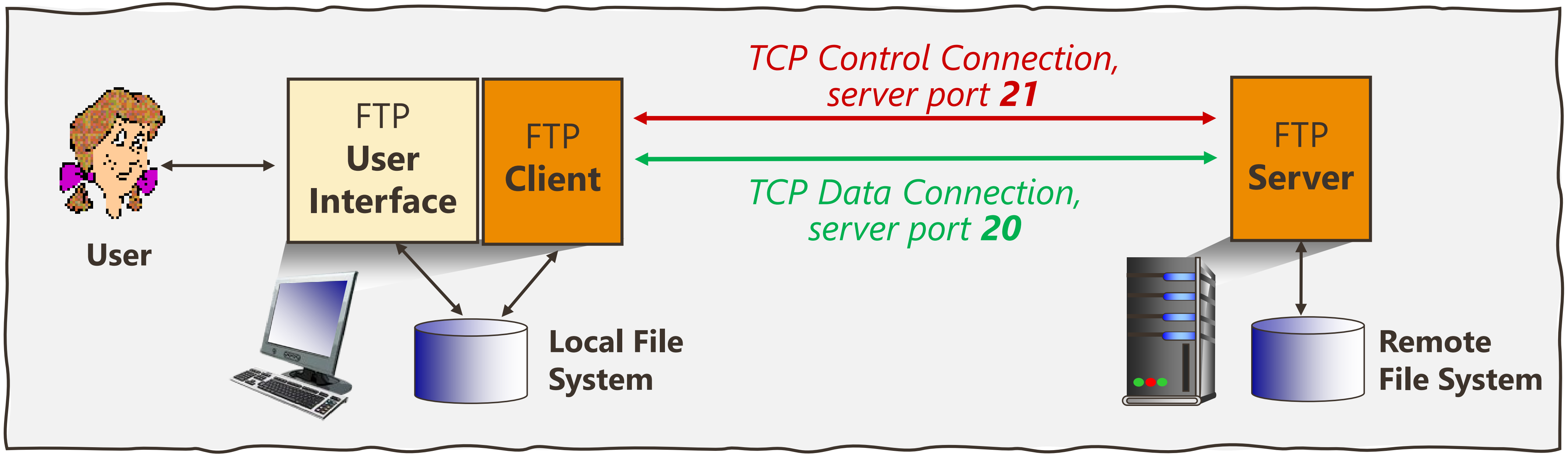
# Application Layer Protocols - FTP

- Fundamentals
- Connections
- Commands and Reponses

# File Transfer Protocol (**FTP**)

Curtin University

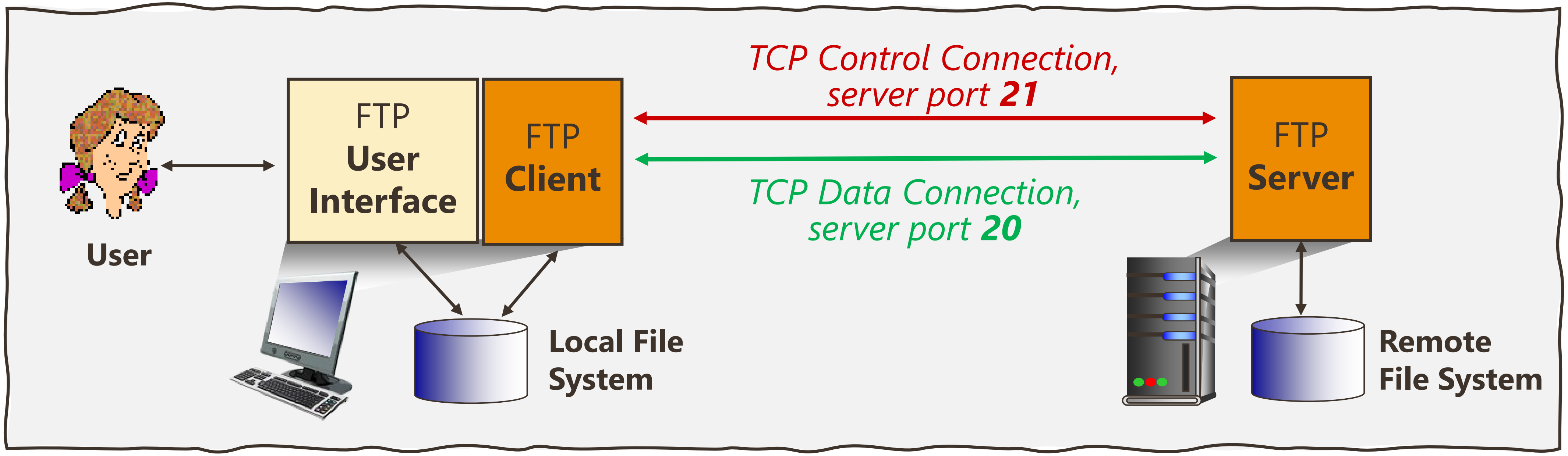
- **Client initiates** connection
- Client **authorized over control connection**
- Client browses remote directory, sends commands over control connection
- When server receives file transfer command, server opens **2nd TCP data connection** (for file transfer) to client



# Separate control, data connections

- **Control Connection: “out of band”**
- **FTP server maintains “state”**
  - current directory, earlier authentication etc.

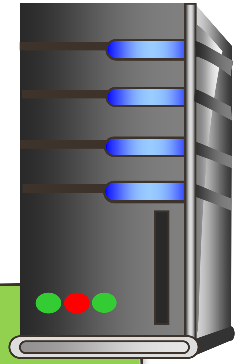
1 control connection, many data connections (i.e. each file transfer)



# FTP Commands, Responses

## Sample Commands

- Sent as **ASCII text** over control channel
  - ✓ **USER** username
  - ✓ **PASS** password
  - ✓ **LIST** return list of file in current directory
  - ✓ **RETR** filename retrieves (gets) file
  - ✓ **STOR** filename stores (puts) file onto remote host



## Sample Return Codes

- **status code** and phrase (as in HTTP)
  - ✓ **331** Username OK, password required
  - ✓ **125** data connection already open; transfer starting
  - ✓ **425** Can't open data connection
  - ✓ **452** Error writing file

Curtin University

# Application Layer Protocols - HTTP

- Review on web
  - Web resource
  - WWW
- HTTP
  - Connections (persistent/non-persistent)
  - Messages (http request / http response)
  - HTTP 1.1 / HTTP 2.0
  - Maintaining State (Cookies)
  - Web Caching (Browser Cache, Proxy Server)
  - Conditional GET
  - Web Sockets

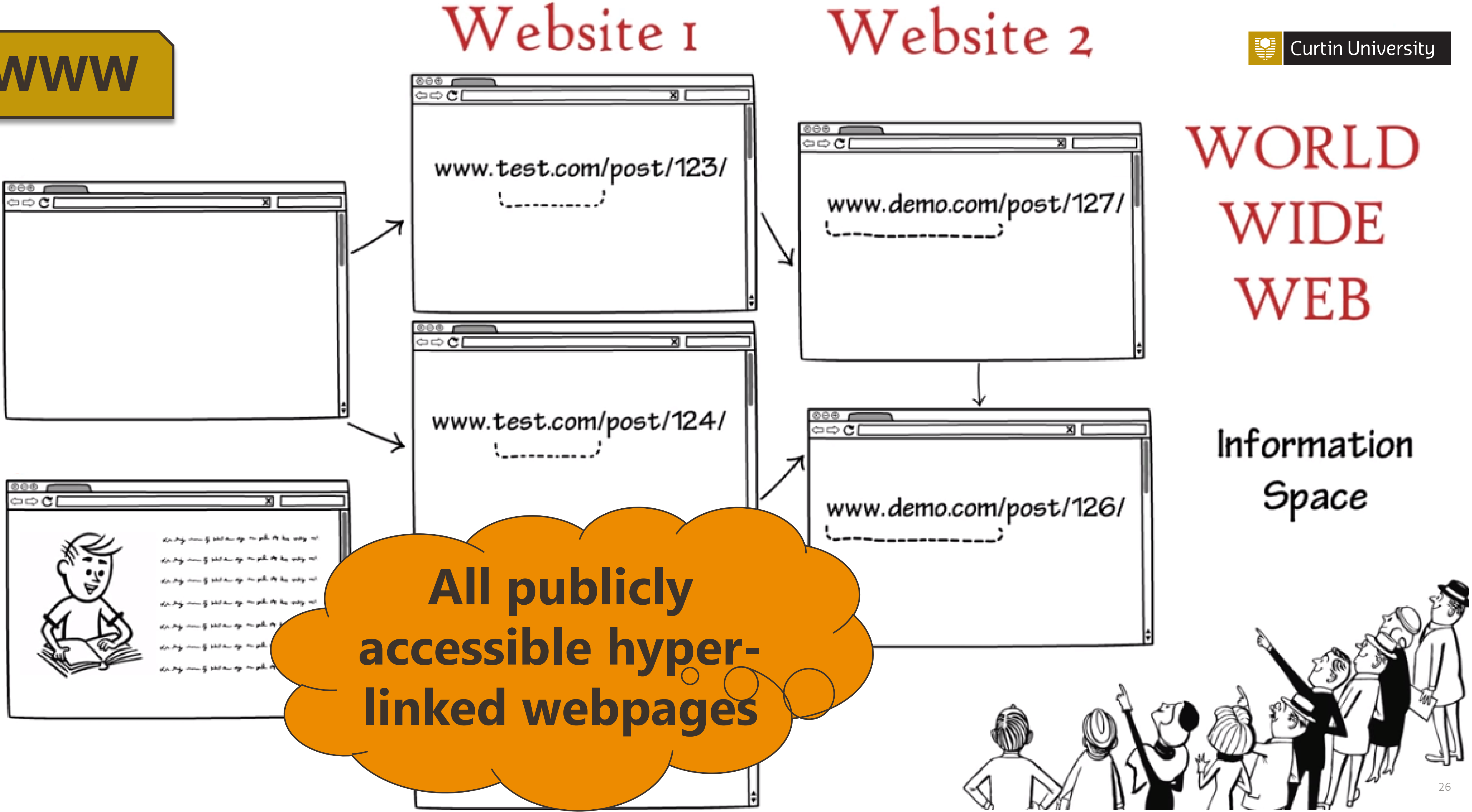
# Web and HTTP

- **First, a review…**
  - Web page consists of objects
  - Object can be HTML file, JPEG image, Java applet, audio file,…
  - Web page consists of base HTML-file which includes several referenced objects
  - Each object is addressable by a URL, e.g.

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

# WWW

Curtin University



All publicly accessible hyper-linked webpages

WORLD WIDE WEB

Information Space

# Web and HTTP – cont.

Curtin University

**Is the Internet and the World Wide Web (WWW) the same ?**

- Internet is a network of networks

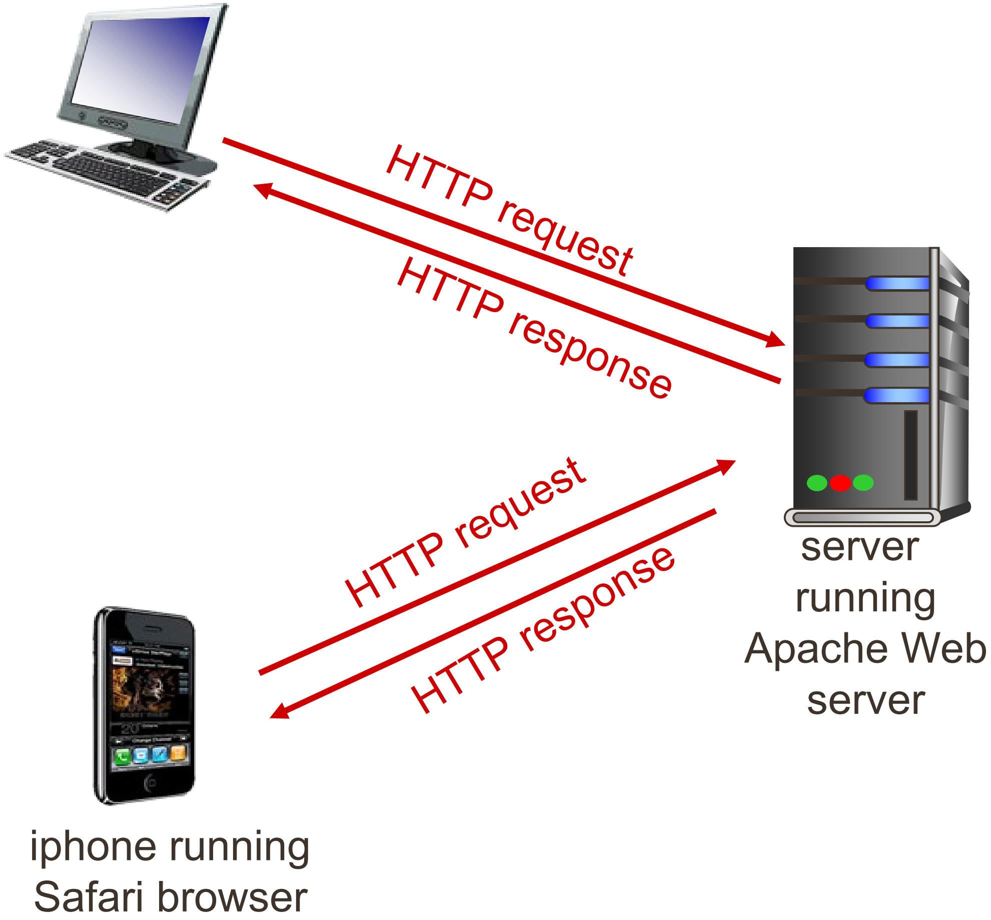
**To store information and share it | i.e. website**

- WWW is a ***distributed system*** that runs on top of the Internet

— Not a network!

# HTTP Overview

- **HTTP:** hypertext transfer protocol
- Web's application layer protocol
- **Client/Server model**
  - client: browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - server: Web server sends (using HTTP protocol) objects in response to requests


HTTP request
HTTP response
HTTP request
HTTP response
server running Apache Web server
iphone running Safari browser

# HTTP Overview – cont.

- **Uses TCP:**
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- **HTTP is "stateless"**
  - server maintains no information about past client requests

protocols that maintain "state" are complex!

- ❖ past history (state) must be maintained
- ❖ if server/client crashes, their views of "state" may be inconsistent, must be reconciled

# HTTP Connections

- **Non-persistent HTTP**
  - at most one object sent over TCP connection
  - connection then closed
  - downloading multiple objects  required multiple connections
- **Persistent HTTP**
  - multiple objects can be sent over  single TCP connection between  client and server

# Non-persistent HTTP

Suppose user enters URL:

www.someSchool.edu/someDepartment/home.index

**contains text, references to 10 jpeg images**

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

# Non-persistent HTTP – cont.

2. HTTP client sends HTTP *request* message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response* message containing requested object, and sends message into its socket

4. HTTP server closes TCP connection.

# Non-persistent HTTP – cont.

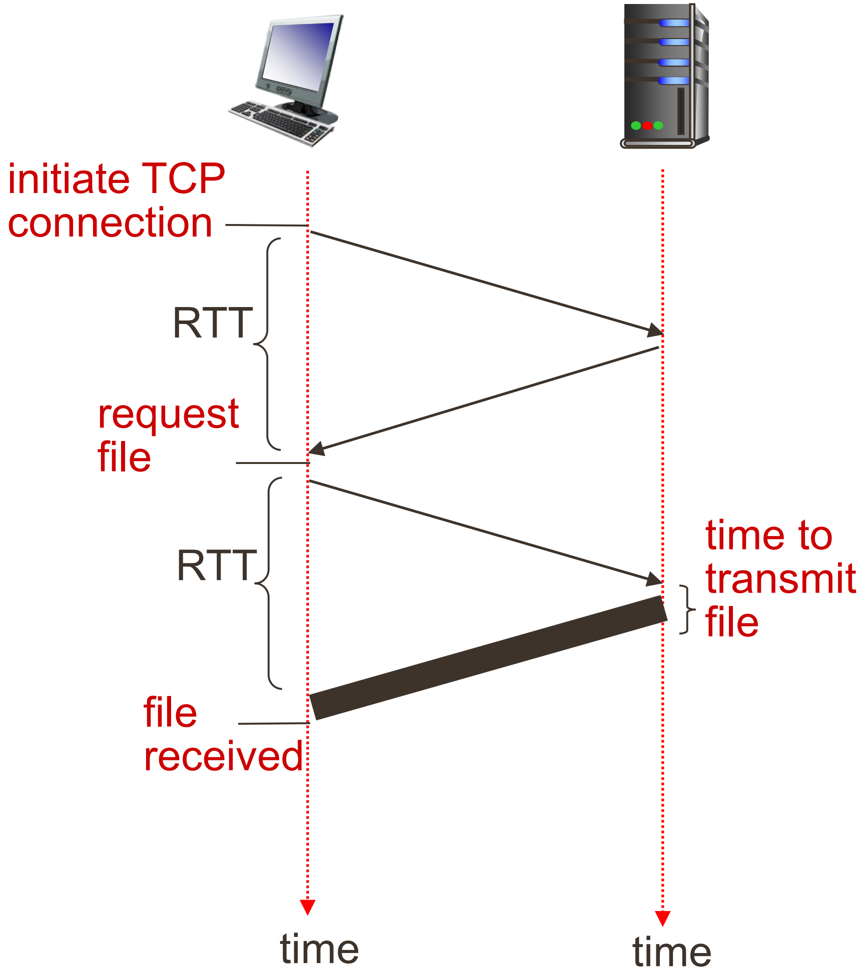
Curtin University

5. HTTP client receives *response* message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Repeat Steps 1-5 for each of the 10 jpeg objects

# Non-persistent HTTP: response time

- HTTP response time:
  - one RTT to initiate TCP connection
  - one RTT for HTTP request and first few bytes of HTTP response to return
  - file transmission time
  - non-persistent HTTP response time = 2RTT + file transmission time



# Persistent HTTP

- **Non-persistent HTTP issues:**
  - requires 2 RTTs per object
  - OS overhead for each TCP connection
  - browsers often open parallel TCP connections to fetch referenced objects

- **Persistent HTTP:**
  - server leaves connection open after sending response
  - subsequent HTTP messages between same client/server sent over open connection
  - client sends requests as soon as it encounters a referenced object
  - as little as one RTT for all the referenced objects

# HTTP Messages

## HTTP Request:

- ASCII (human-readable format)

✓ **Methods**

- GET, POST
- HEAD,
- PUT, DELETE

✓ **URL**

- Requested resource path (on server)

✓ **Version**

- HTTP/1.0
- HTTP/1.1
- HTTP/2.0

## HTTP Response:

✓ **Status code** *(similar in FTP)*

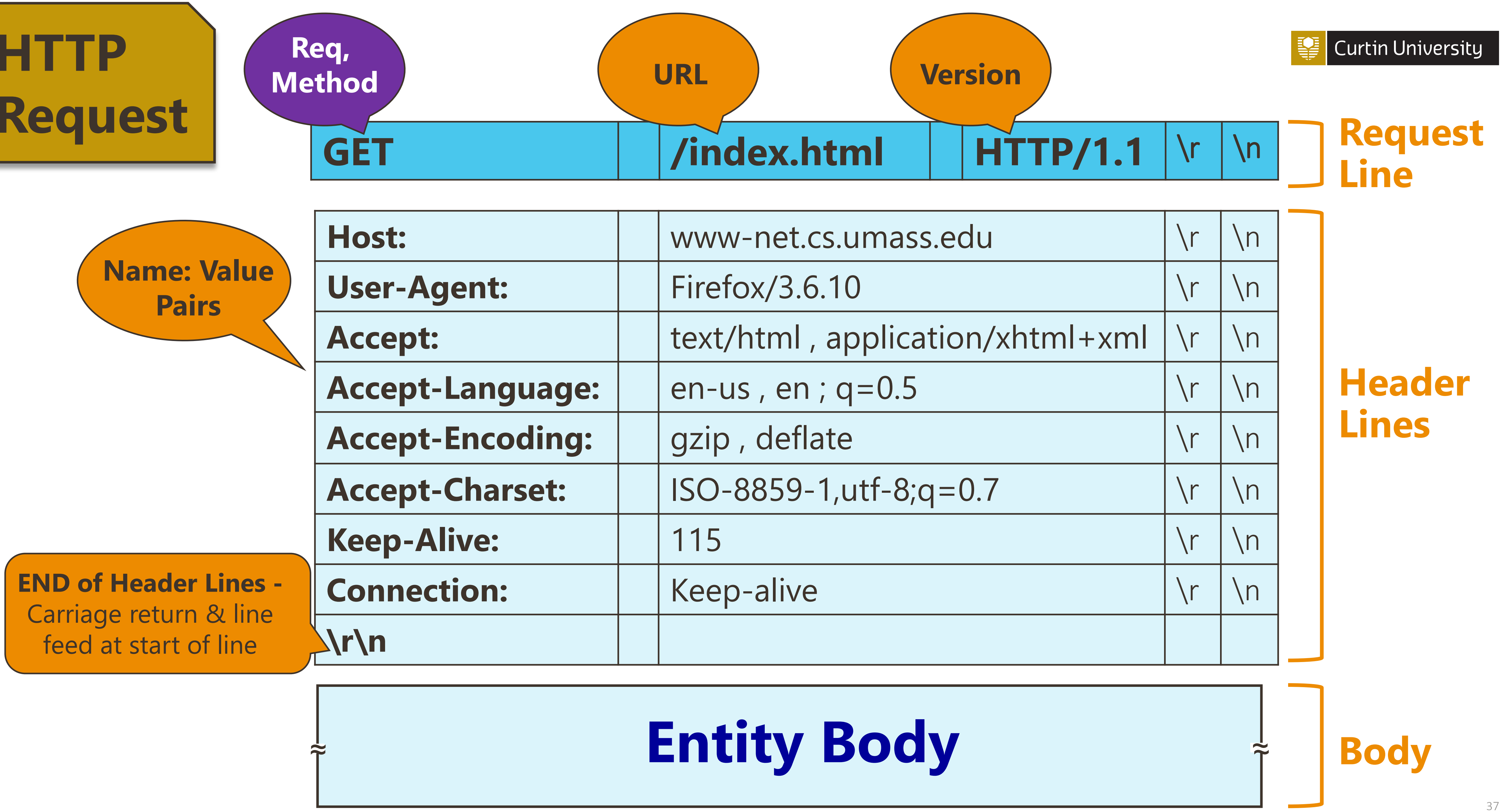
**200** OK

**404** Not Found

**400** Bad Request

**301** Moved Permanently

# HTTP Request



Req, Method

URL

Version

| | | | | | | |
|---|---|---|---|---|---|---|
| GET | | /index.html | | HTTP/1.1 | \r | \n |

Request Line

Name: Value Pairs

| | | | | |
|---|---|---|---|---|
| Host: | | www-net.cs.umass.edu | \r | \n |
| User-Agent: | | Firefox/3.6.10 | \r | \n |
| Accept: | | text/html , application/xhtml+xml | \r | \n |
| Accept-Language: | | en-us , en ; q=0.5 | \r | \n |
| Accept-Encoding: | | gzip , deflate | \r | \n |
| Accept-Charset: | | ISO-8859-1,utf-8;q=0.7 | \r | \n |
| Keep-Alive: | | 115 | \r | \n |
| Connection: | | Keep-alive | \r | \n |
| \r\n | | | | |

Header Lines

**END of Header Lines -** Carriage return & line feed at start of line

≈ Entity Body ≈

Body

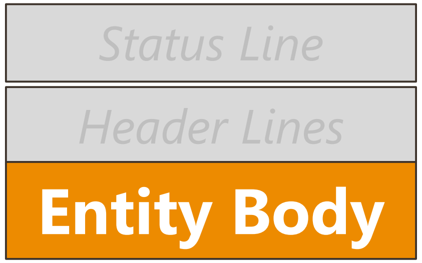
# Uploading Form Input

Curtin University

- **GET Method**

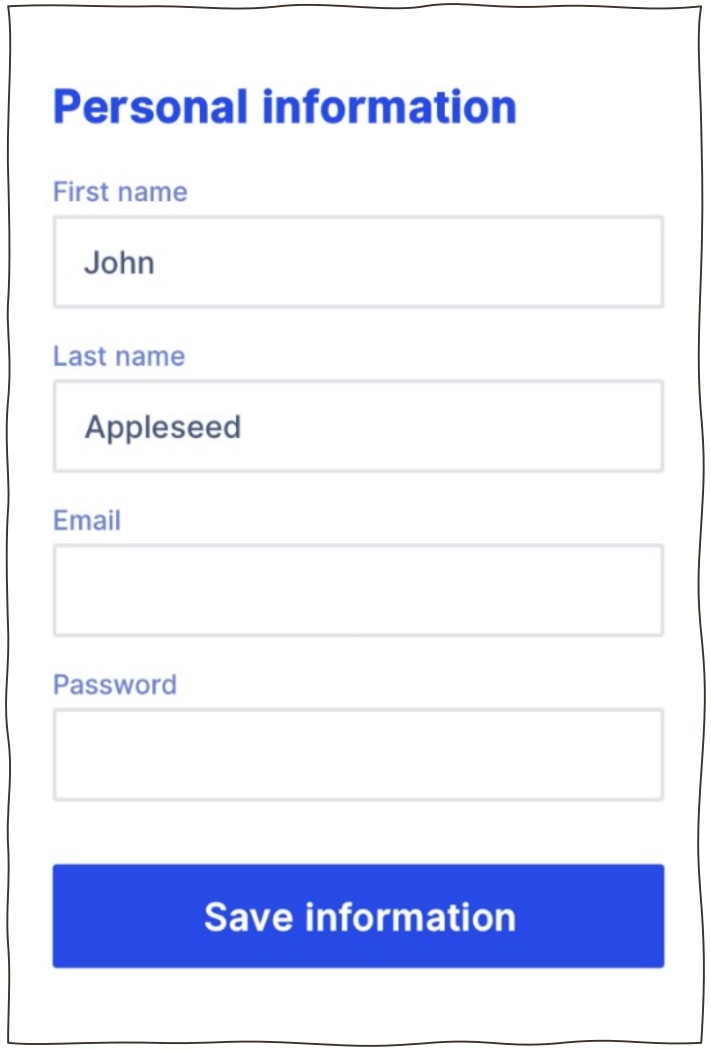
  Input is uploaded in **URL field** of request line:

```
www.w3schools.com/action_page.php?fname=John&lname=Appleseed
```



- **POST Method**

  Input is uploaded to server in the body of the request message



## Personal information

First name: John

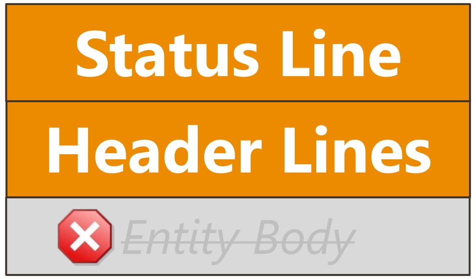
Last name: Appleseed

Email:

Password:

# Request Methods – cont.

Curtin University


Status Line

Header Lines

✗ ~~Entity Body~~

## HEAD

Asks server to leave **requested object out of response**

## PUT

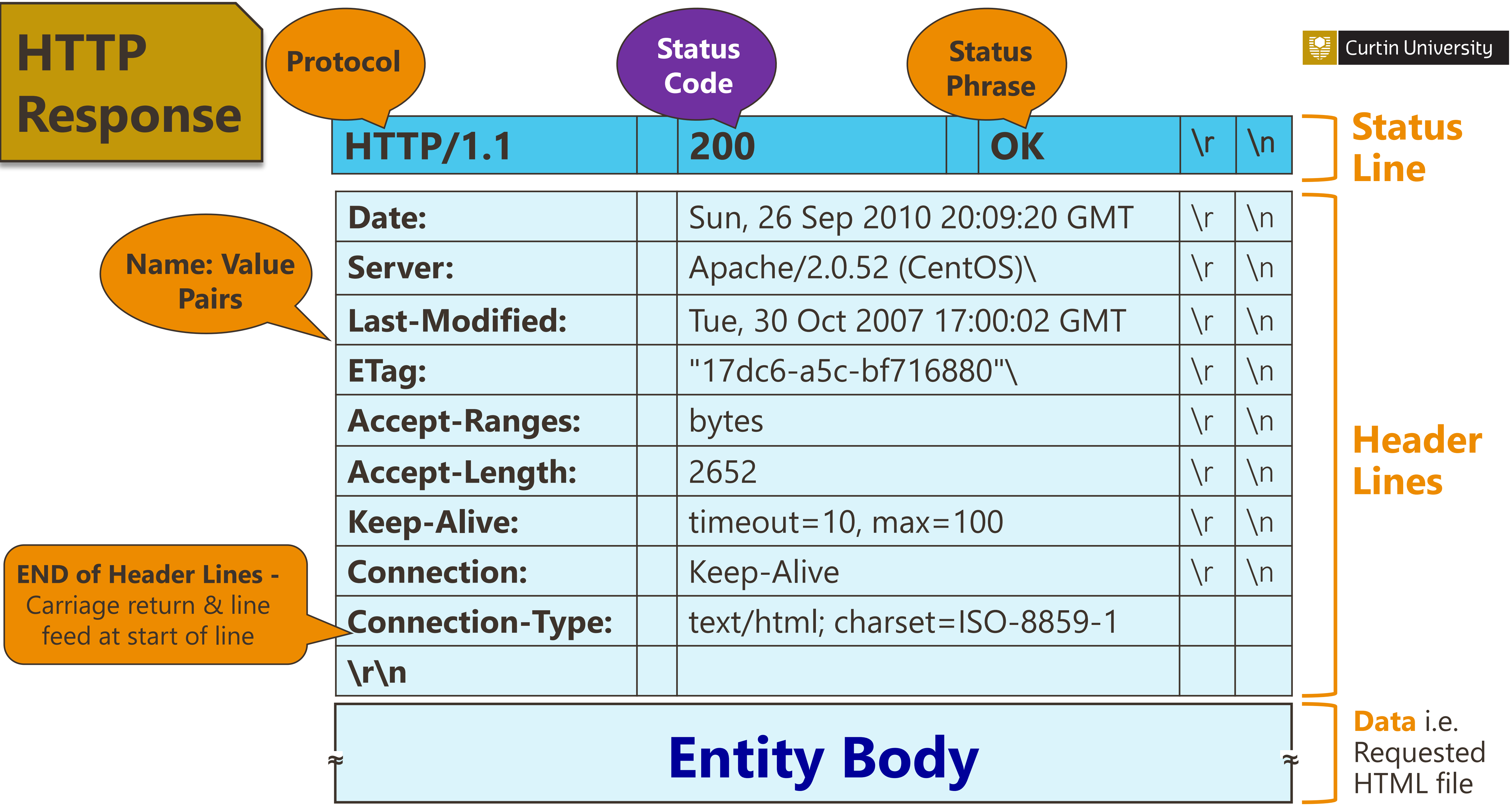
**uploads file** in entity body to path specified in **URL field**

## DELETE

**deletes** file specified in the **URL field**

Curtin University

# HTTP Response



Protocol

Status Code

Status Phrase

| | | | | | | |
|---|---|---|---|---|---|---|
| HTTP/1.1 | | 200 | | OK | \r | \n |

Status Line

| | | | | |
|---|---|---|---|---|
| Date: | | Sun, 26 Sep 2010 20:09:20 GMT | \r | \n |
| Server: | | Apache/2.0.52 (CentOS)\ | \r | \n |
| Last-Modified: | | Tue, 30 Oct 2007 17:00:02 GMT | \r | \n |
| ETag: | | "17dc6-a5c-bf716880"\ | \r | \n |
| Accept-Ranges: | | bytes | \r | \n |
| Accept-Length: | | 2652 | \r | \n |
| Keep-Alive: | | timeout=10, max=100 | \r | \n |
| Connection: | | Keep-Alive | \r | \n |
| Connection-Type: | | text/html; charset=ISO-8859-1 | | |
| \r\n | | | | |

Name: Value Pairs

**END of Header Lines -** Carriage return & line feed at start of line

Header Lines

**Entity Body**

**Data** i.e. Requested HTML file

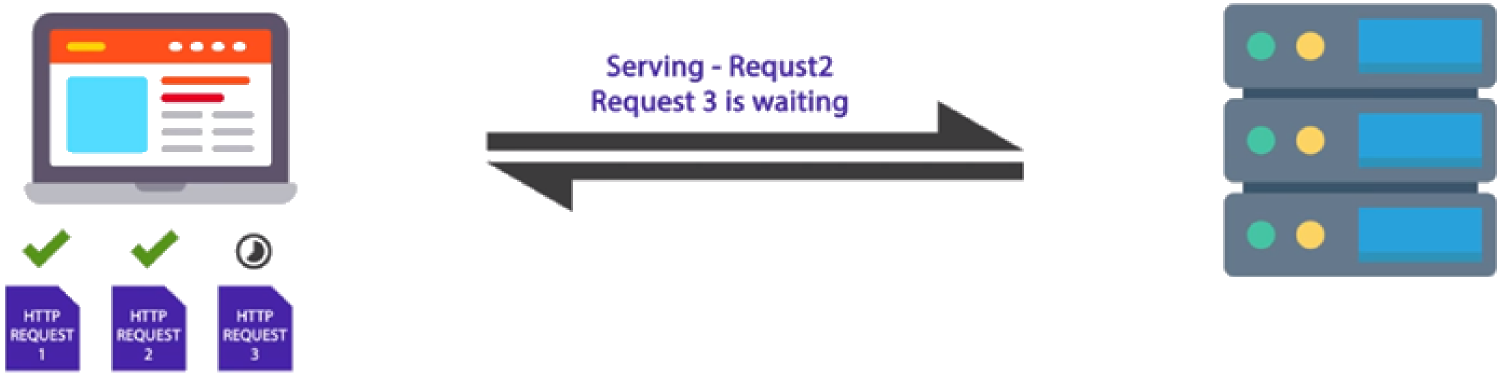
# HTTP Response - **Status Codes**

- **Some Sample Codes:**
  - **200 OK**
    request succeeded, requested object later in this msg
  - **301 Moved Permanently**
    requested object moved, new location specified later in this msg (Location:)
  - **400 Bad Request**
    request msg not understood by server
  - **404 Not Found**
    requested document not found on this server
  - **505 HTTP Version Not Supported**

# HTTP 1.1 - Problems

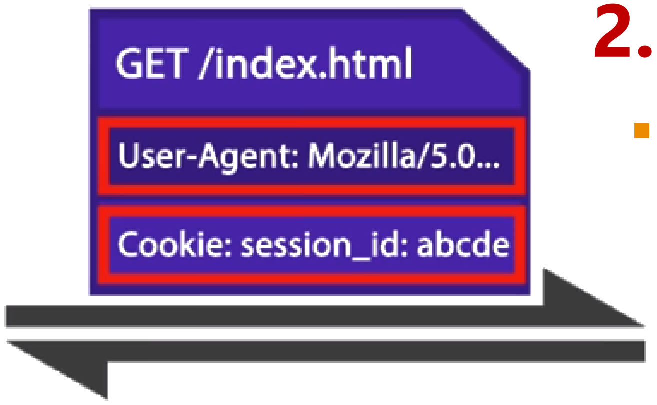
## 1. Head-of-Line Blocking

The TCP connection/channel is blocked by the preceding request



## 2. Redundancy in request header

- Sending same static header parameters again and again

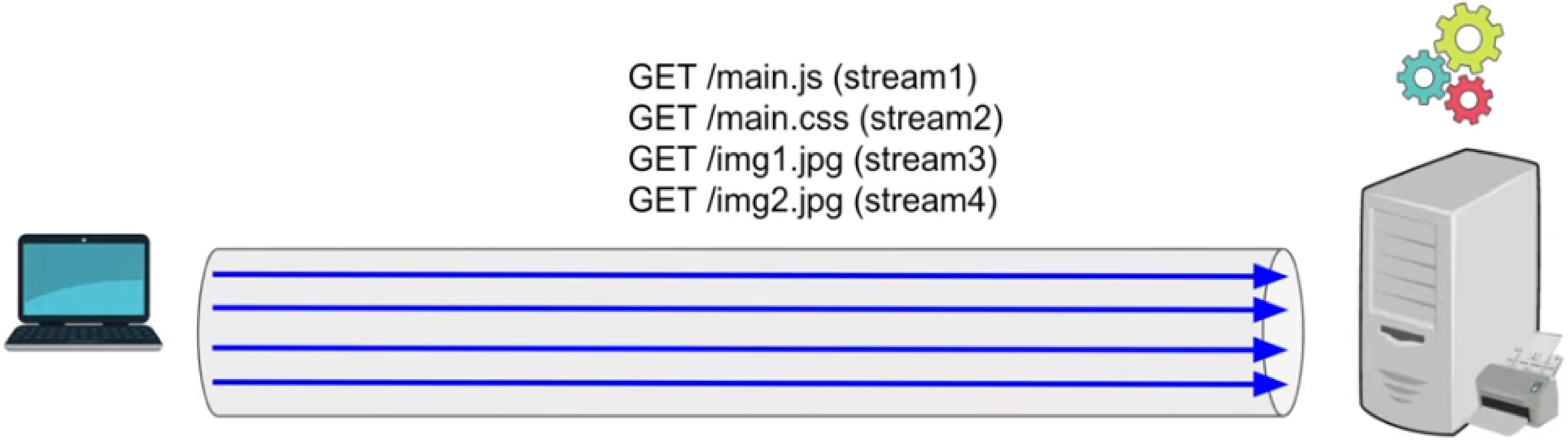


GET /index.html
User-Agent: Mozilla/5.0...
Cookie: session_id: abcde

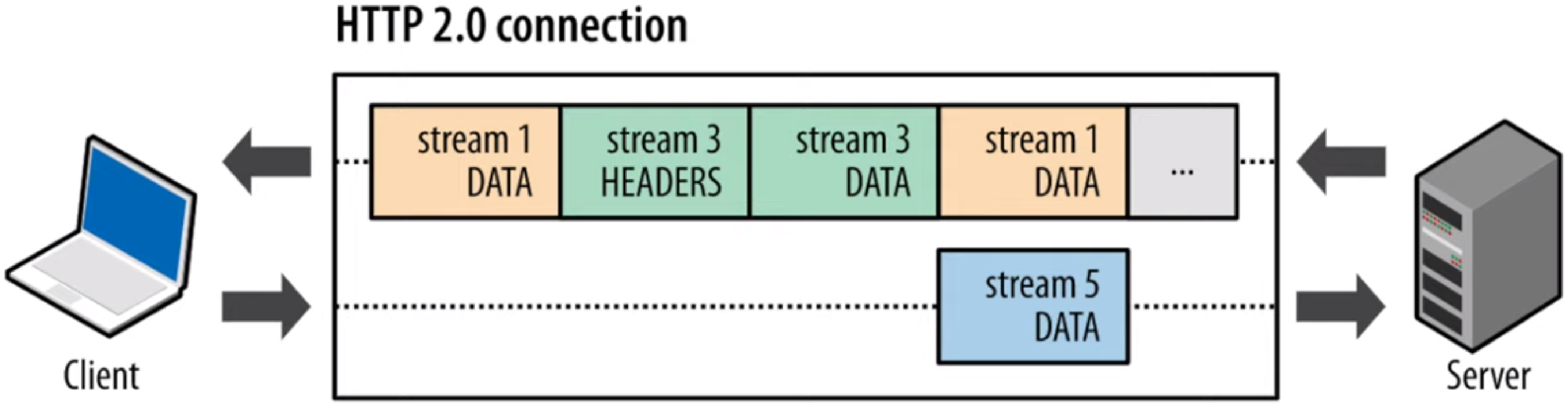
## 3. No header compression

# 1. No More Head-of-Line Blocking

- Single TCP connection serve multiple requests by multiplexing
- StreamID is used to identify request-response streams



GET /main.js (stream1)
GET /main.css (stream2)
GET /img1.jpg (stream3)
GET /img2.jpg (stream4)

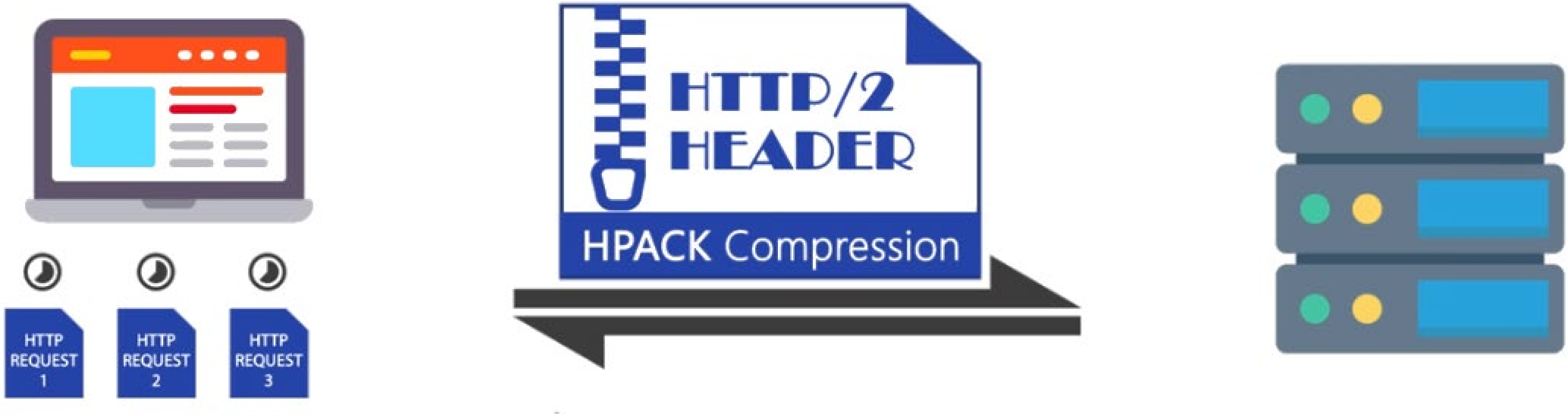


HTTP 2.0

# 2. Allows to compress HTTP Headers, Data

Curtin University

- Own compression format (HPACK)
- Compression works at connection level,
- so that headers can be shared among requests.

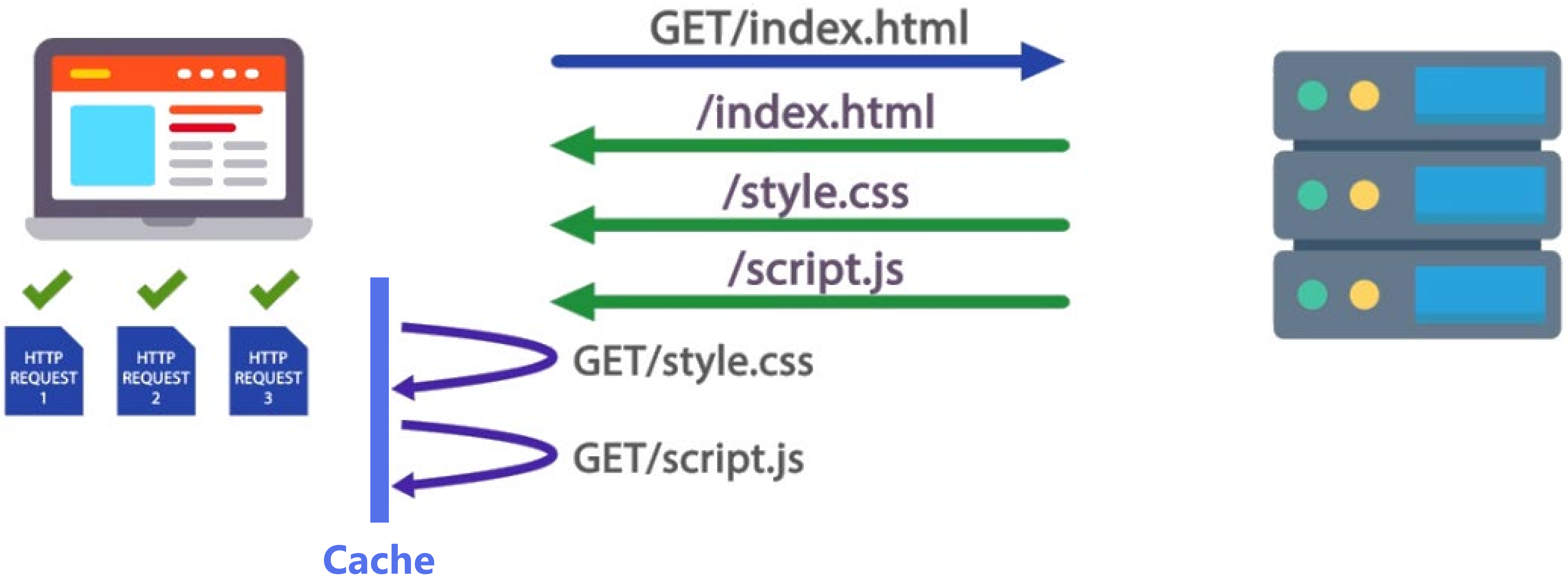


HTTP 2.0

# 3. Push (not push notifications)

Curtin University

- Allows respond to the request that hasn't even being sent.
- But you are sure the client would request it.
- During the actual request, it will be fetched from the **cache**.



# User-server state: cookies

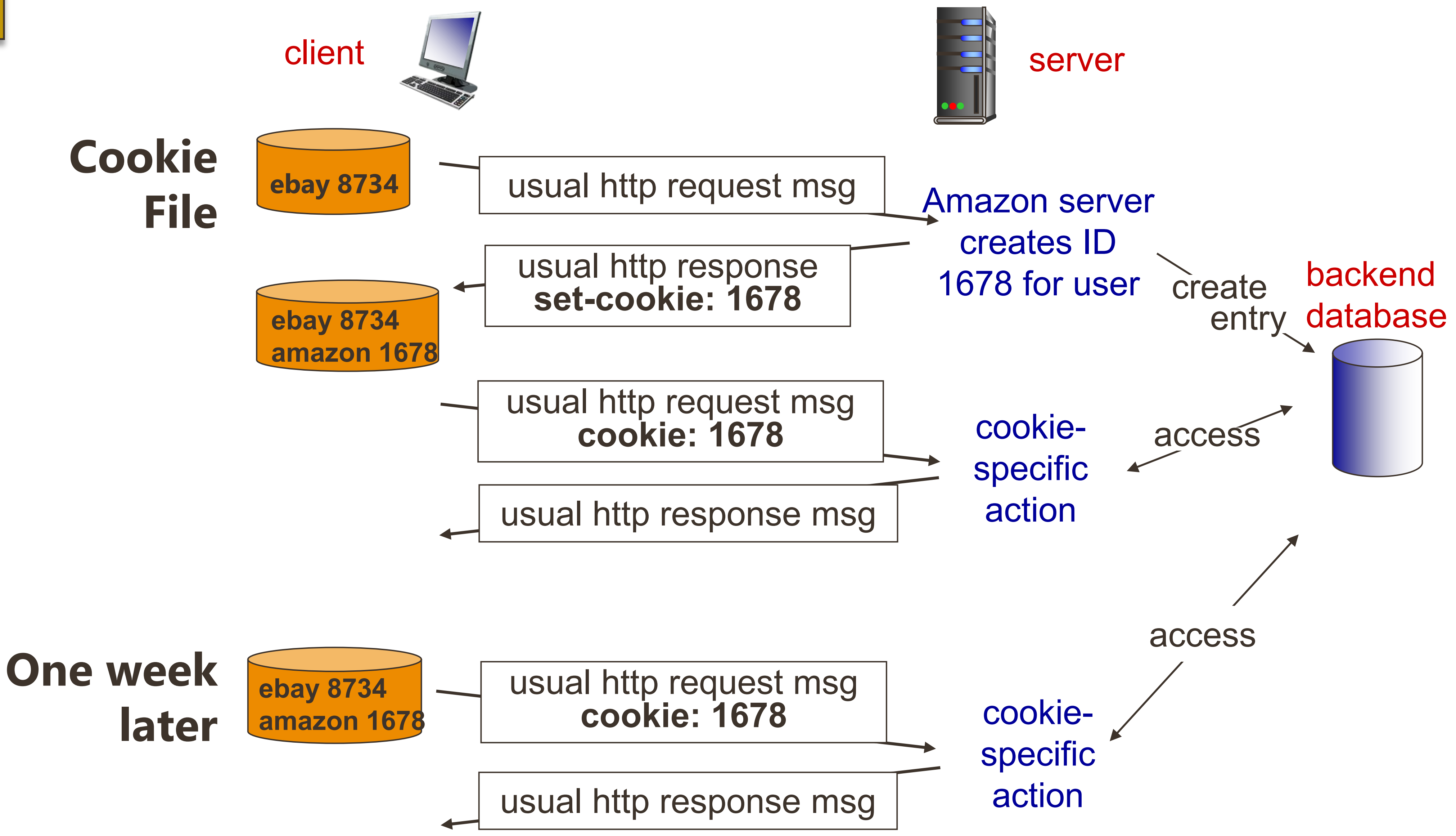
- **Many** web sites **use cookies**

- Four components:

1. **cookie header line** of **HTTP response** message *(from server)*
2. **cookie header line** in next **HTTP request** message *(to the server)*
3. **cookie file** kept **on user's host**, managed by user's browser (no extra burden on the server)
4. **back-end database** at web server

# Cookies

## Keeping "state"



# Cookies: keeping “state”

- **How to keep “state”:**
  - ✓ **protocol endpoints:**

    maintain state at sender/receiver over multiple transactions
  - ✓ **cookies:**

    http messages carry state
- **Cookies can be used for:**
  - ✓ authorization
  - ✓ shopping carts
  - ✓ recommendations
  - ✓ user session state (Web e-mail)

**Cookies:**

- permits sites to learn a lot about you
- stored in clear text

# Web Caching

- Typically cache is installed by ISP (University, Company, Residential ISP)

- **Why Web caching?**
  - ✓ Reduce response time for client request
  - ✓ Reduce traffic on an institution’s access link
  - ✓ Internet dense with caches enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

# Web Caches: Proxy Server

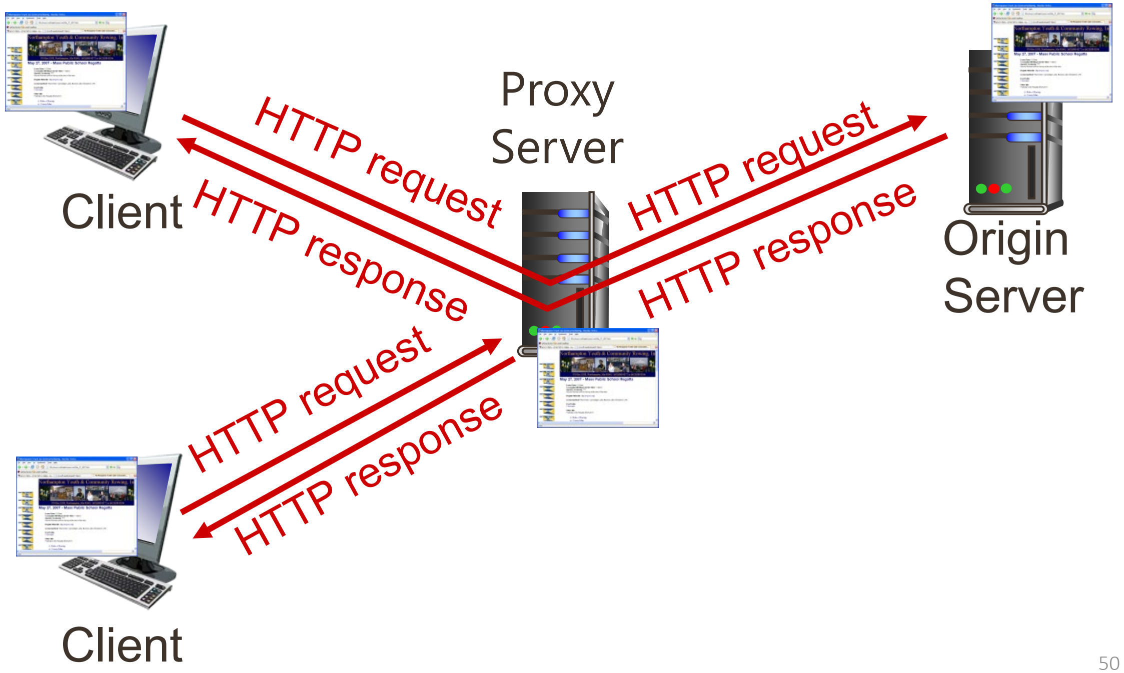
Curtin University

**Goal:** satisfy client request without involving origin server

- Browser (client) -> **Proxy Server** **(server)**
  - ✓ Object in cache, return object
  - ✓ Else, request from origin server

- **Proxy Server** **(client)**

  -> Origin Server (server)

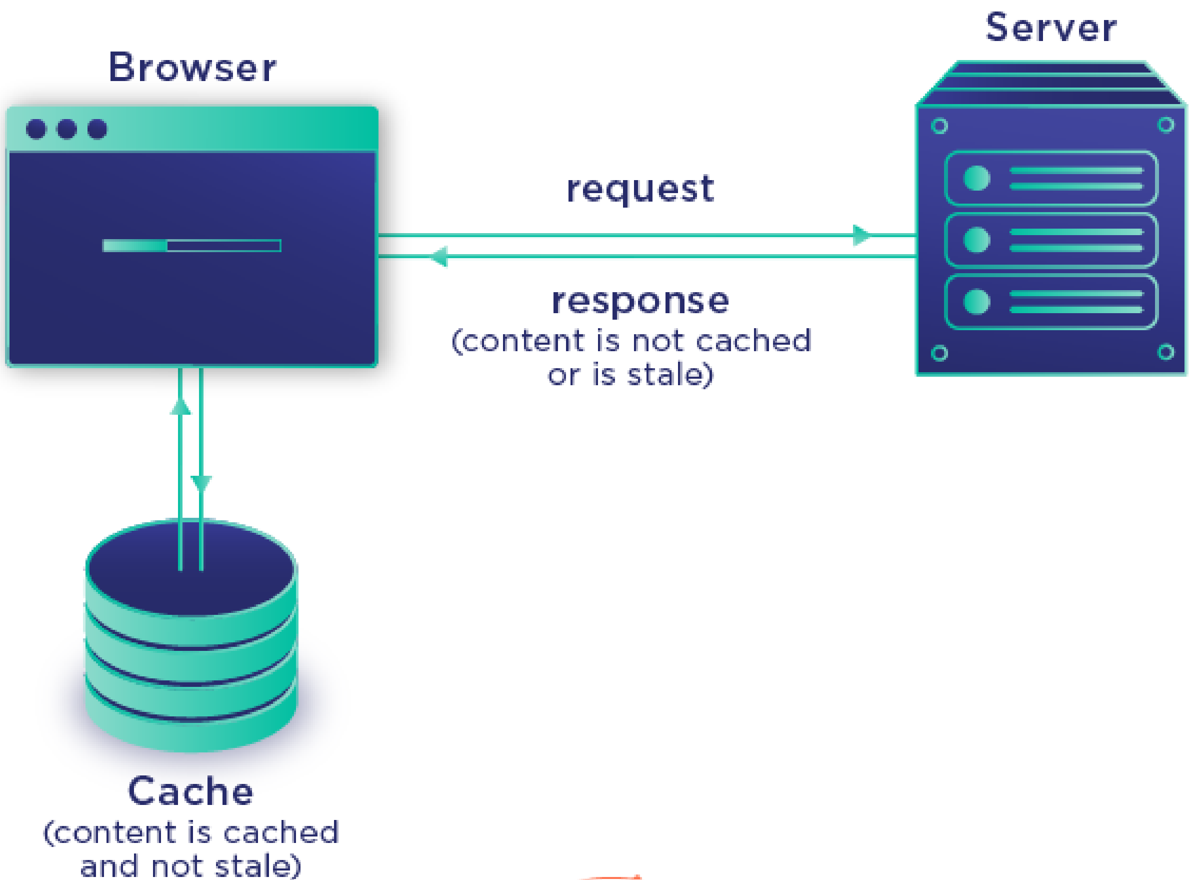


# Web Caches: Browser Cache

**Goal:** satisfy client request without involving origin server

- Typically cache **static assets**
  - ✓ Parts of a website that do not change from visit to visit
  - ✓ i.e. HTML, CSS, JavaScripts, images, etc.
- What to cache? How long?

*determined by the webserver*

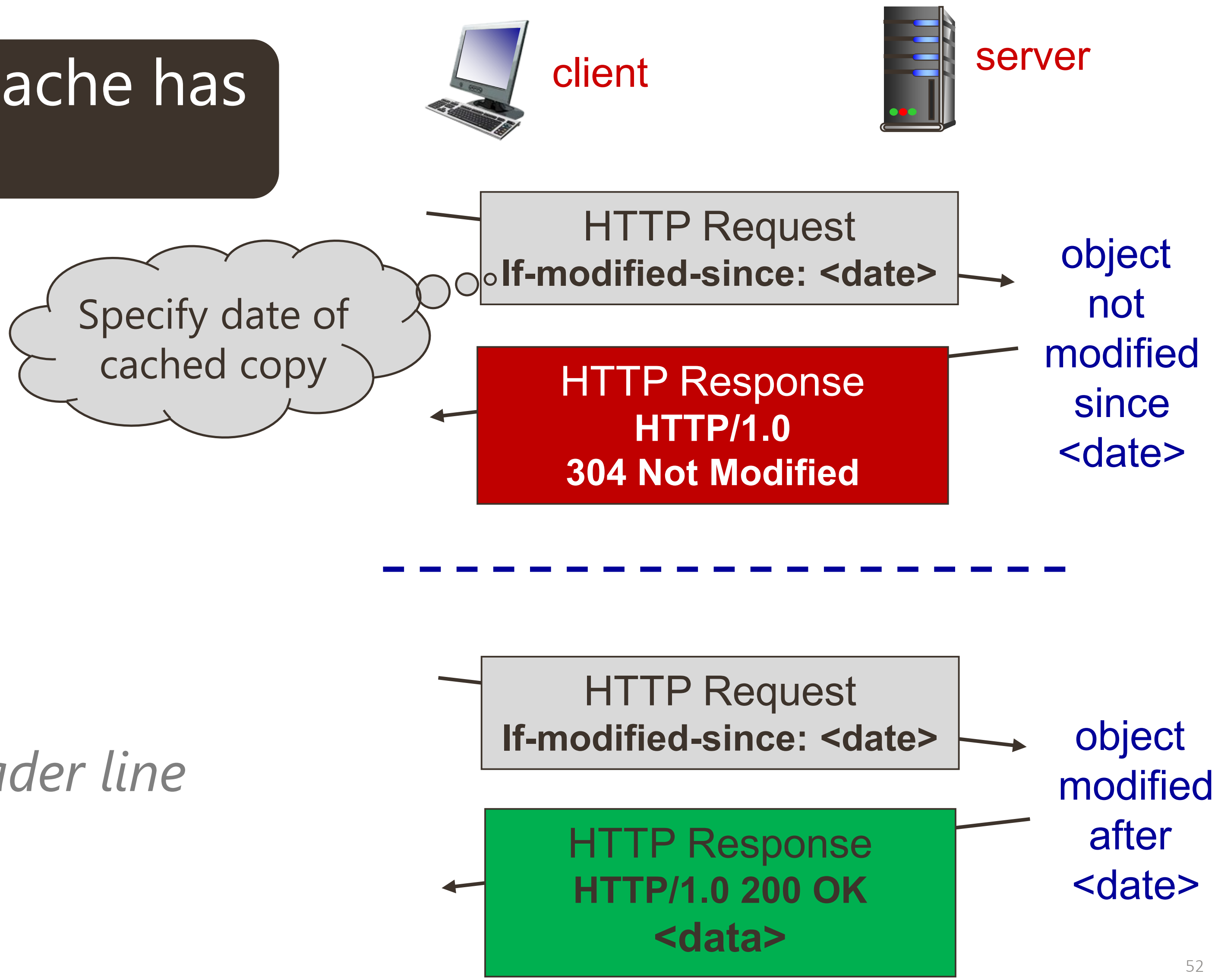

Browser
Server
request
response
(content is not cached or is stale)
Cache
(content is cached and not stale)

# Conditional GET

**Goal:** don’t send object if cache has up-to-date cached version

- No object transmission delay
- Lower link utilization

- **If-modified-since: <date>** *// header line*



client

server

Specify date of cached copy

HTTP Request
**If-modified-since: <date>**

HTTP Response
**HTTP/1.0**
**304 Not Modified**

object not modified since <date>

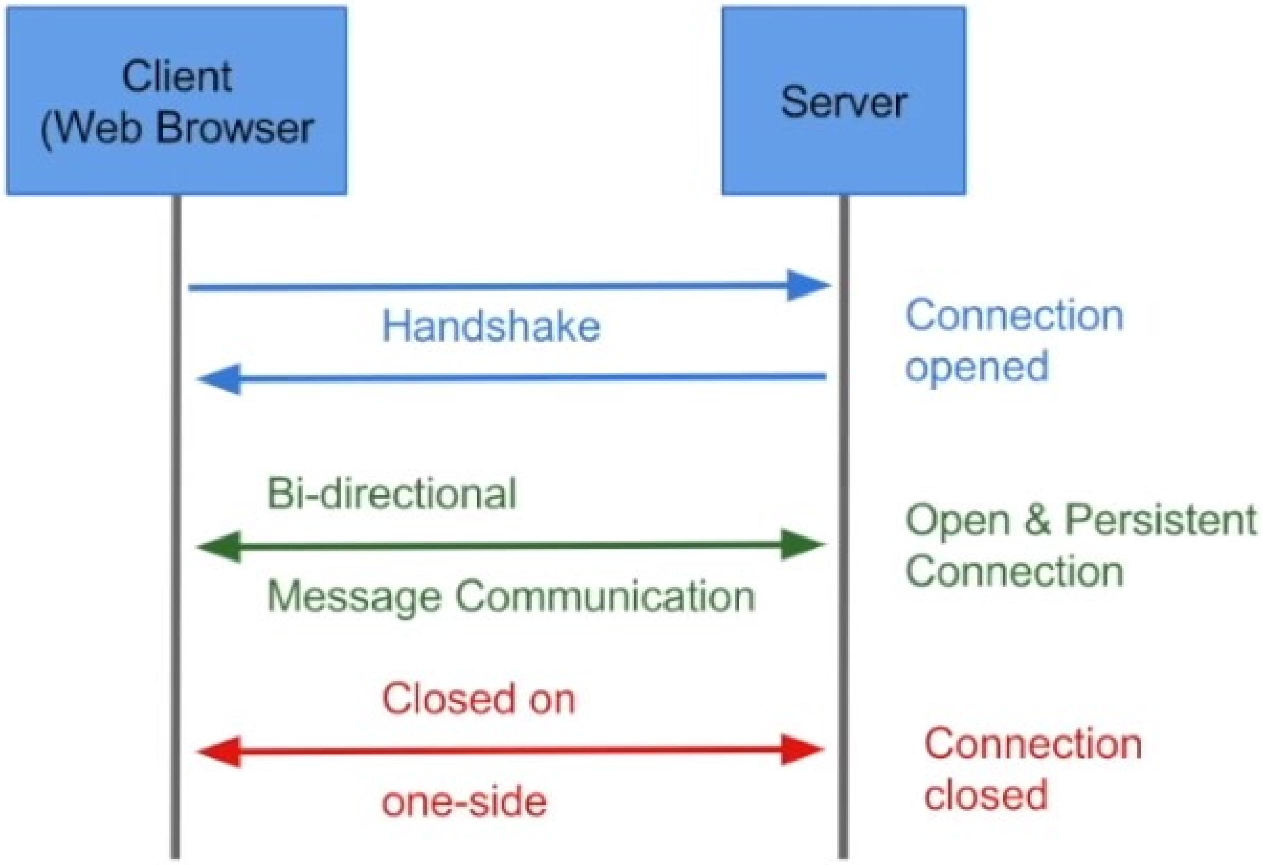
HTTP Request
**If-modified-since: <date>**

HTTP Response
**HTTP/1.0 200 OK**
**<data>**

object modified after <date>

# HTTP and **Web Sockets**

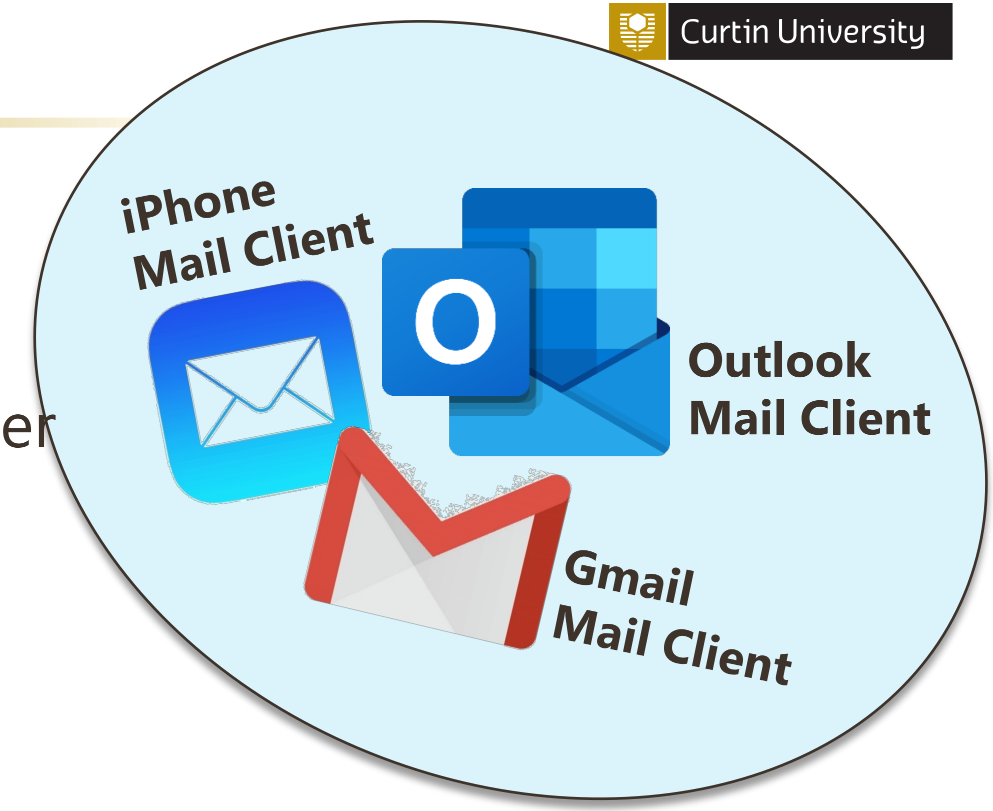
- **Bi-directional** *(unlike http uni-directional request-response)*
- **Persistent Connection**, Faster
- **Message Oriented Protocol**
- **For Real-Time** applications
  - ✓ No need to refresh UI/browser



# Application Layer Protocols - **SMTP, POP3, IMAP**

- Mail-sending Protocol
  - SMTP
- Mail-Access Protocol
  - HTTP (Web-mail)
  - POP3
  - IMAP

# Electronic Mail



## 1. User Agents

- ✓ composing, editing, reading mail messages
- ✓ **outgoing**, **incoming** messages stored on server

## 2. Mail Servers

A. **“Mailbox”:** incoming messages for user

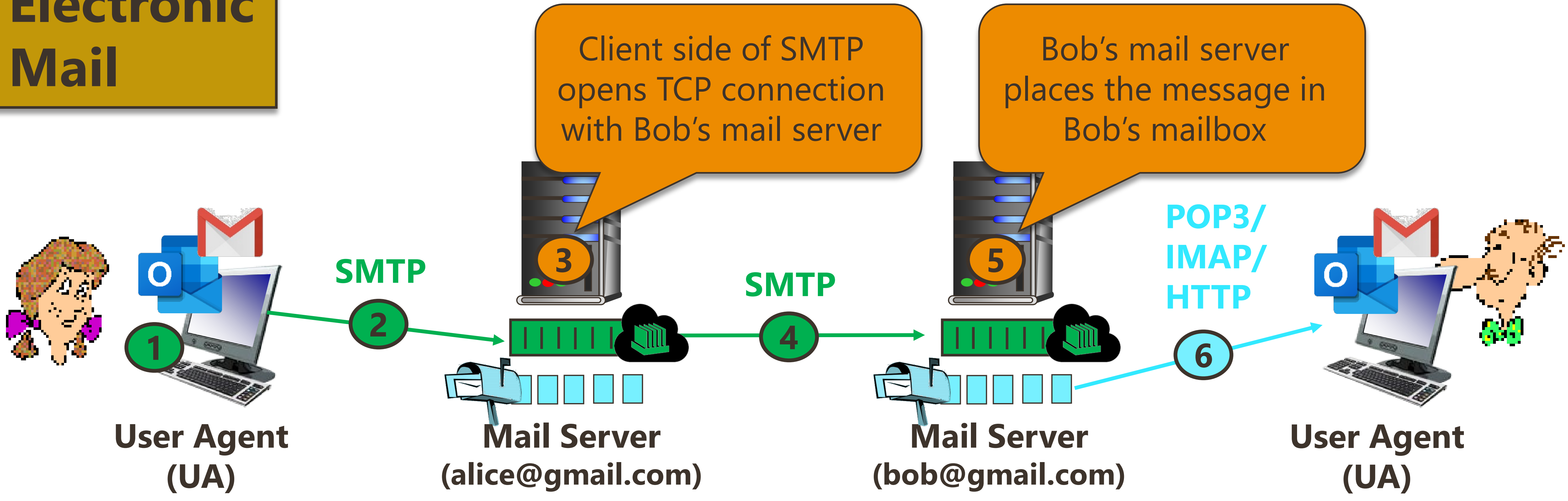
B. **Message Queue:** outgoing (to be sent) mail messages

## 3. SMTP: Simple Mail Transfer Protocol

- ✓ **“client”:** sending mail server
- ✓ **“server”:** receiving mail server

# Electronic Mail

Curtin University



Client side of SMTP opens TCP connection with Bob's mail server

Bob's mail server places the message in Bob's mailbox

1

SMTP

2

3

SMTP

4

5

POP3/ IMAP/ HTTP

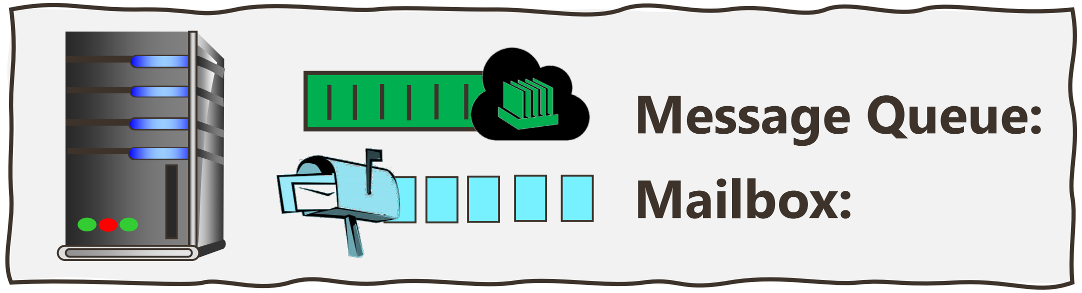
6

User Agent (UA)

Mail Server (alice@gmail.com)

Mail Server (bob@gmail.com)

User Agent (UA)



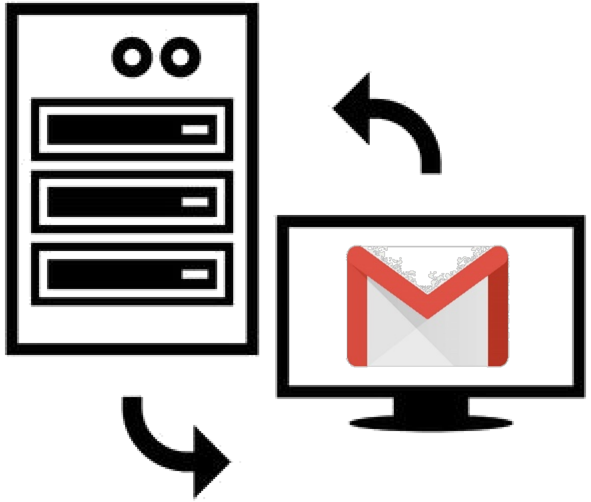
Message Queue:

Mailbox:

# Sample SMTP interaction

Curtin University

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```



# SMTP [RFC 2821]

Curtin University

- Uses TCP to reliably transfer email message from client to server, **port 25**
- **Direct Transfer:** Sending server to receiving server
- **Messages** must be in **7-bit ASCII**

- **Three phases of transfer**
  1. handshaking (greeting)
  2. transfer of messages
  3. closure

- **Command/Response Interaction** *(like HTTP, FTP)*
  - ✓ **Commands:** ASCII text
  - ✓ **Response:** Status code and phrase

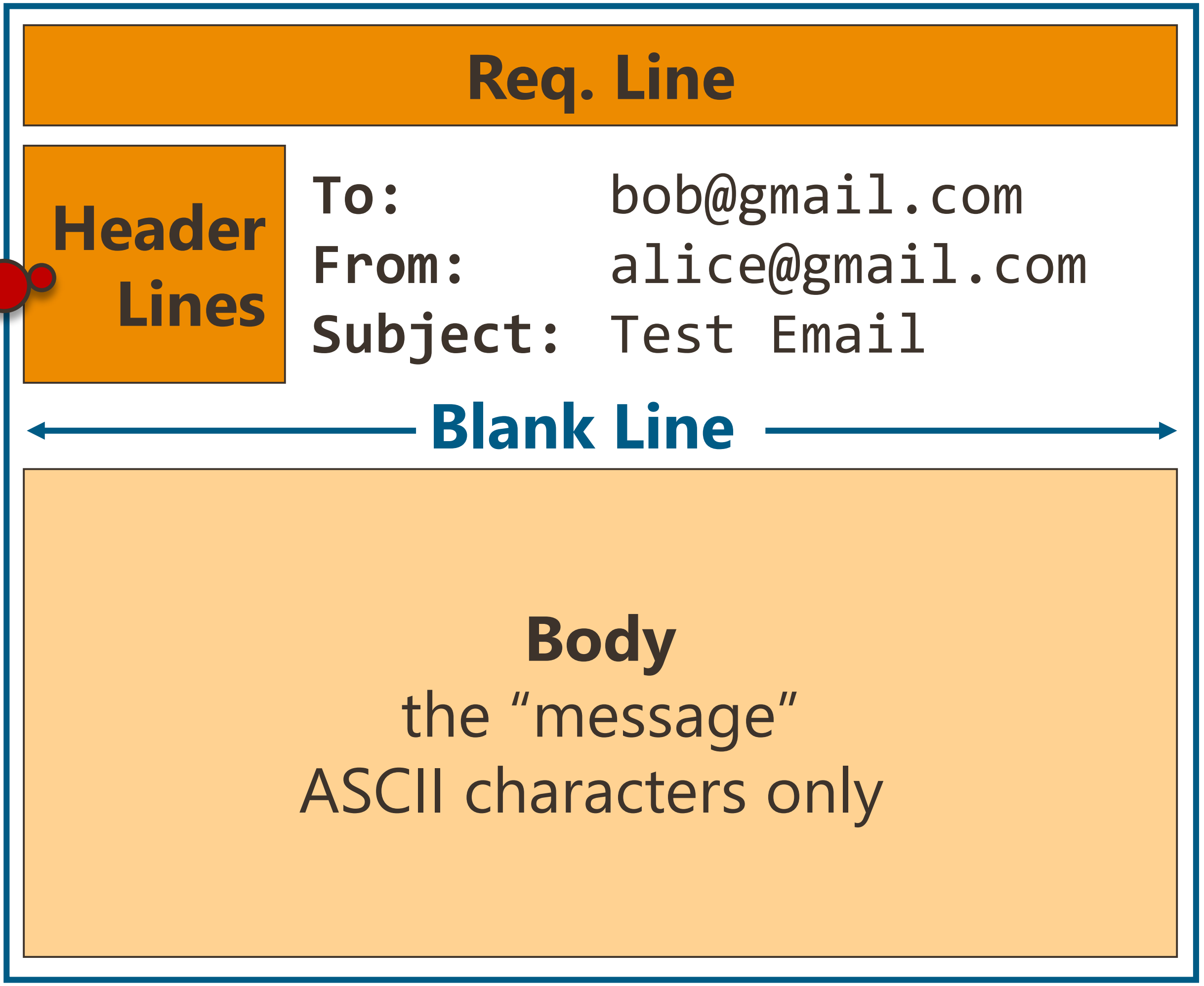
# SMTP – cont.

- Uses **persistent connections**
- Requires message (header & body) to be in **7-bit ASCII**
- Server uses **CRLF.CRLF** to determine **end of message**
- **Comparison with HTTP:**
  - **HTTP:** pull
  - **HTTP:** each object encapsulated in its own response message
  - **SMTP:** push
  - **SMTP:** multiple objects sent in multipart message
  - both have ASCII command/response interaction, status codes

# SMTP Mail Message Format

Curtin University

***different** from*
SMTP MAIL
FROM, RCPT TO:
**commands !**



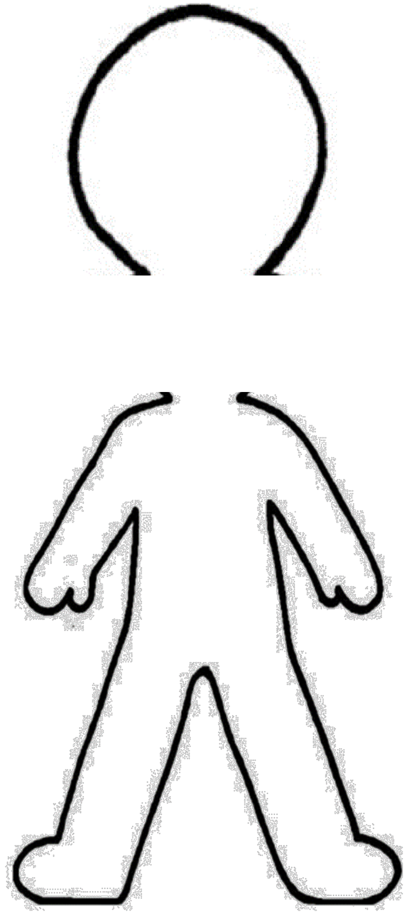
**Req. Line**

**Header Lines**

```
To:       bob@gmail.com
From:     alice@gmail.com
Subject:  Test Email
```

**Blank Line**

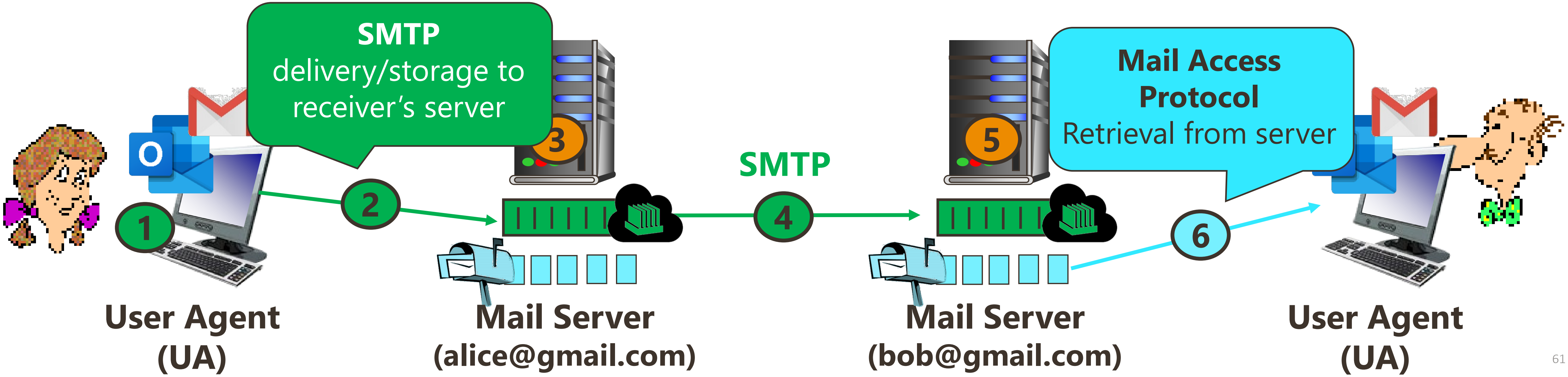
**Body**
the "message"
ASCII characters only



1

# Mail Access Protocols

1. **POP:** Post Office Protocol: **Authorization**, **Download**
2. **IMAP:** Internet Mail Access Protocol: **More Features**
   *including manipulation of stored messages on server*
3. **HTTP:** **Gmail**, **Hotmail**, Yahoo! Mail, etc.


SMTP
delivery/storage to receiver's server
1
2
3
SMTP
4
5
Mail Access Protocol
Retrieval from server
6
User Agent (UA)
Mail Server (alice@gmail.com)
Mail Server (bob@gmail.com)
User Agent (UA)

# POP3 protocol

*Authorization phase*

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

*Transaction phase,* client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Curtin University

# POP3 – cont.

- **POP3 is stateless** across sessions

- **Two Modes:**

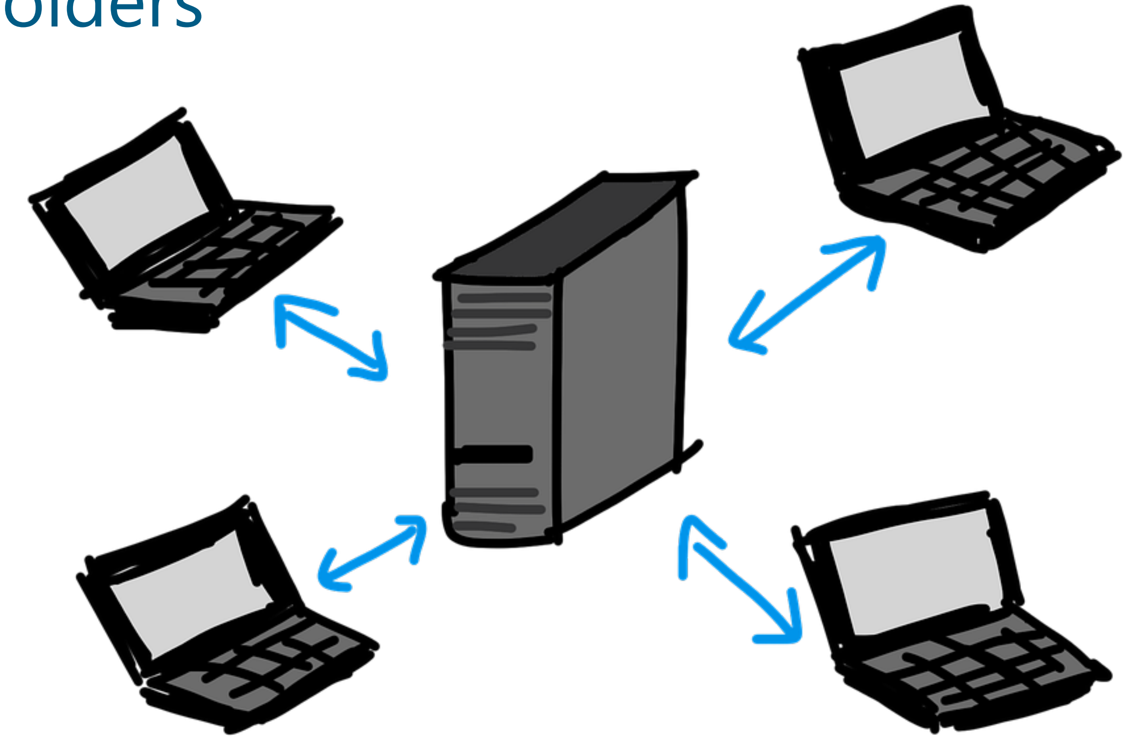
1. **Download and Delete:** Previous example uses this mode

   *Bob cannot re-read e-mail if he changes client*

2. **Download and Keep:** Copies of messages on different clients

# IMAP

- Keeps all messages in one place: at server
- Allows user to organize messages in folders
- **IMAP is stateful** across sessions
  - ✓ Names of folders and mappings between message IDs and folder name



- **Application Layer**
  - Fundamentals
  - App-protocol Contract
  - Required Transport Services
  - Apps underlying TCP Protocols
  - Secure Communication
  - Architectures
    - Client-server
    - P2P

- **App Protocols – Telnet**
  - NVT, VTP
  - Connections
  - Highlights

- **App Protocols – FTP**
  - Connections
  - Commands and Reponses

- **App Protocols – HTTP**
  - Web Basics
    - WWW
  - **HTTP**
    - Connections (persistent/non-persistent)
    - Messages (http request / http response)
    - HTTP Request Methods
    - HTTP 1.1 Problems
    - HTTP 2.0
    - Maintaining State (Cookies)
    - Web Caching (Browser Cache, Proxy Server)
    - Conditional GET
    - Web Sockets

- **App Protocols – SMTP, POP3, IMAP**
  - Mail-sending Protocol
    - SMTP
  - Mail-Access Protocol
    - HTTP (Web-mail)
    - POP3
    - IMAP

Curtin University

# THANK YOU

Make tomorrow better.