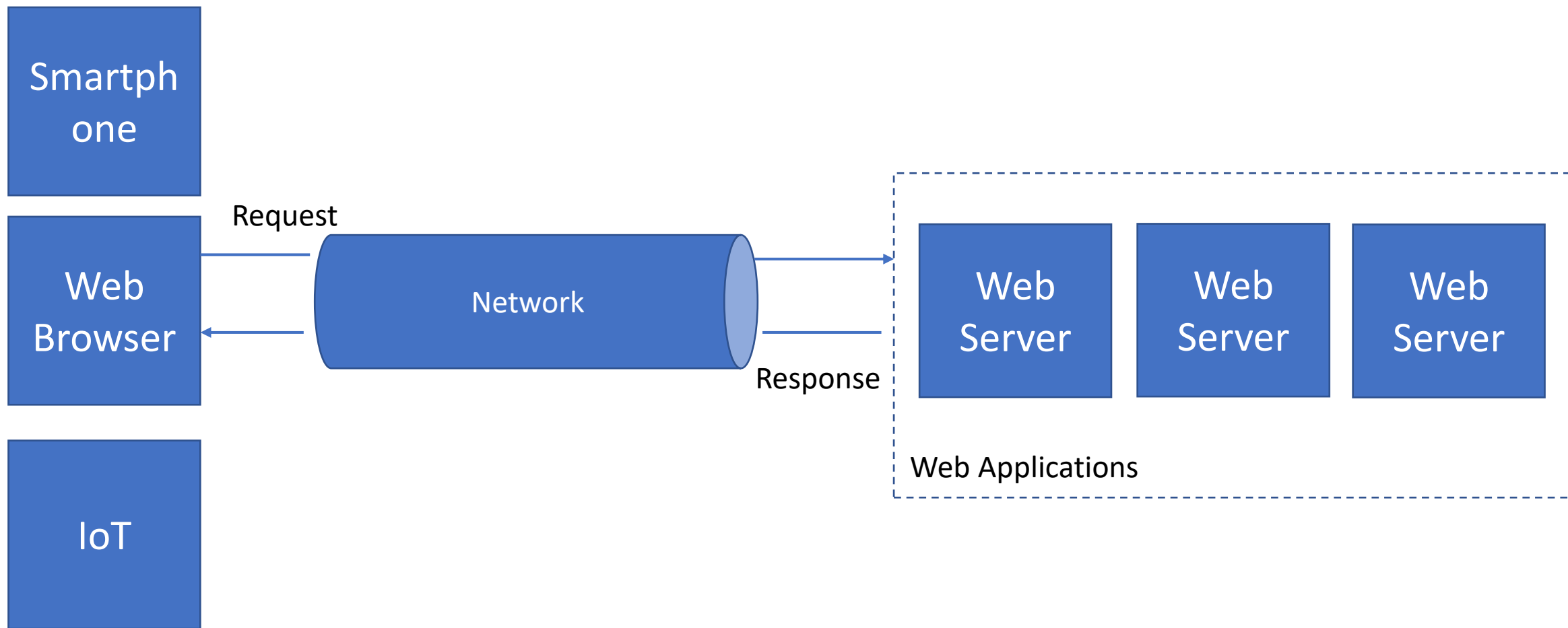




# HTTP Revisited

Chandreyee Chowdhury

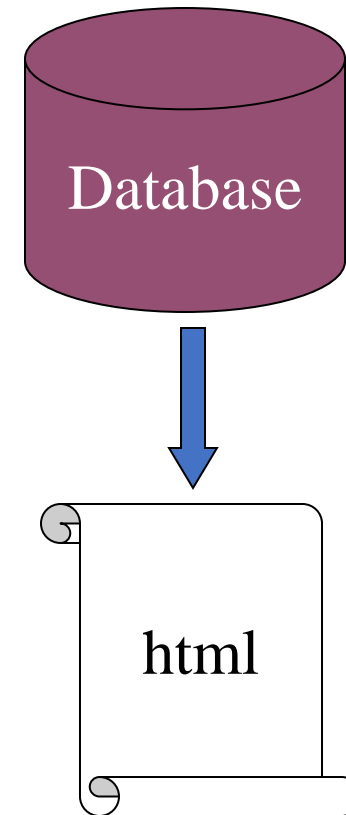


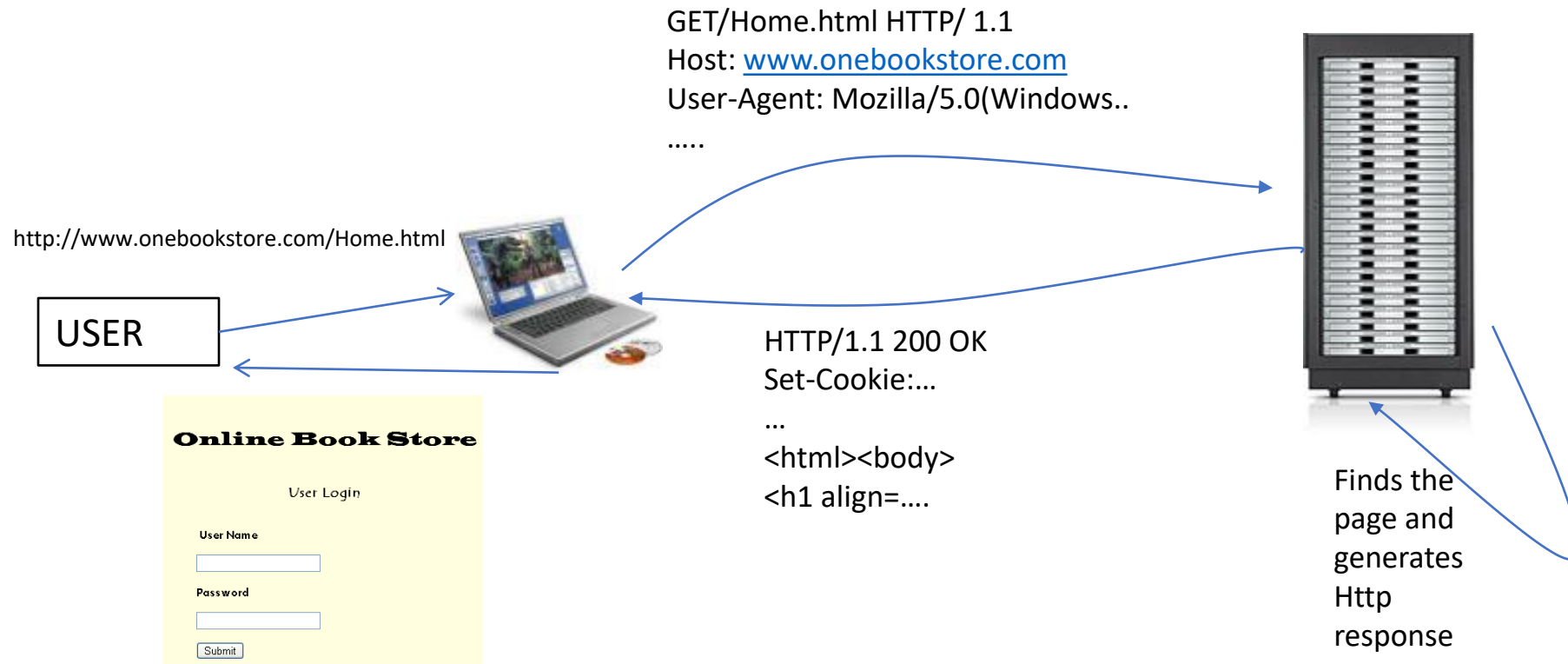
# Why HTTP

- It provides a uniform interface to access the resources and services from a web server or cloud
- Reuse infrastructure for ubiquity
  - Web is ubiquitous
- Reusing
  - application frameworks and libraries
  - Load balancing infrastructure for different applications including interacting with cloud
- Distribute requests throughout the servers

# Data-Driven Websites

- Websites that provide access to:
  - Lots of data
  - Dynamic data
  - Customized views of data
  - E.g. ebay.com
- Scripts map data to html





## Web App1

Application Layer-HTTP request

TCP

IP

MAC

# Http Request

- ***HTTP GET***

- The total amount of characters in a GET is really limited (depending on the server)
- The data you send with the GET is appended to the URL up in the browser bar, so whatever you send is exposed
- Because of this, the user can bookmark a form submission if you use GET

- **HTTP POST**

- The data is included in the request body
- More data can be sent
- General purpose sending of data

# Http Methods

- Put
  - Data to be stored in the server
- Delete
  - Remove some information from the server

GET	PATH + Resource
POST	
PUT	
DELETE	



## Request line

- GET/com/Kolkata/Home.html HTTP/ 1.1
- Method
- <path+resource>

**method**

**path**

**protocol**

GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1

Host: net.tutsplus.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

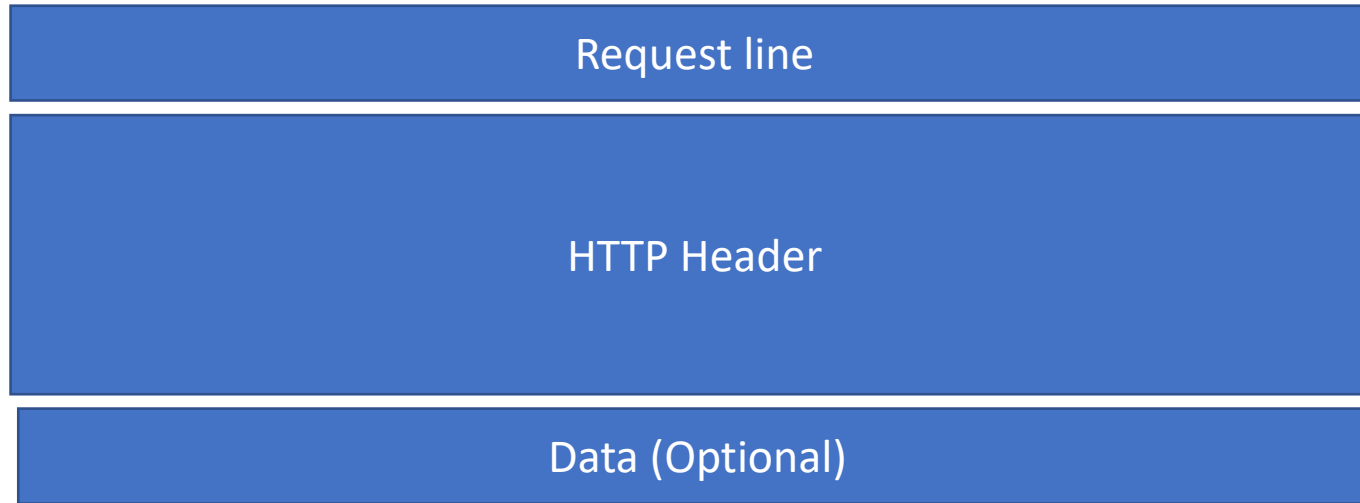
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120

Pragma: no-cache

Cache-Control: no-cache

**HTTP headers as Name: Value**

# HTTP Headers



- Headers are meta information but body of a HTTP message contains pure data
  - These are the extra information that the client is giving the server to help it complete that Request.
- If message body is sent without the header then the server may process the request
  - May not send the response in the expected format
- When body of a message is missing when it was required, the relevant information would not be processed

# Uniform Resource Locator

`https://wishnet.in/home/login.html`

- The way resources are identified is called a URL
- `http://<host name>:<port number>/path/resource?key1=value1&key2=value2`
- Using the query parameters we can pass extra information about a specific aspect of a resource to the server
- URL encoding encodes any character that is not allowed in the query param spec
- For dynamically constructed URLs with data, it is better to encode all URLs as data may not follow the spec
- It is good to provide the correct file extension in the encoded URLs but not a requirement

# Data Types

Image/jpg

Image/png

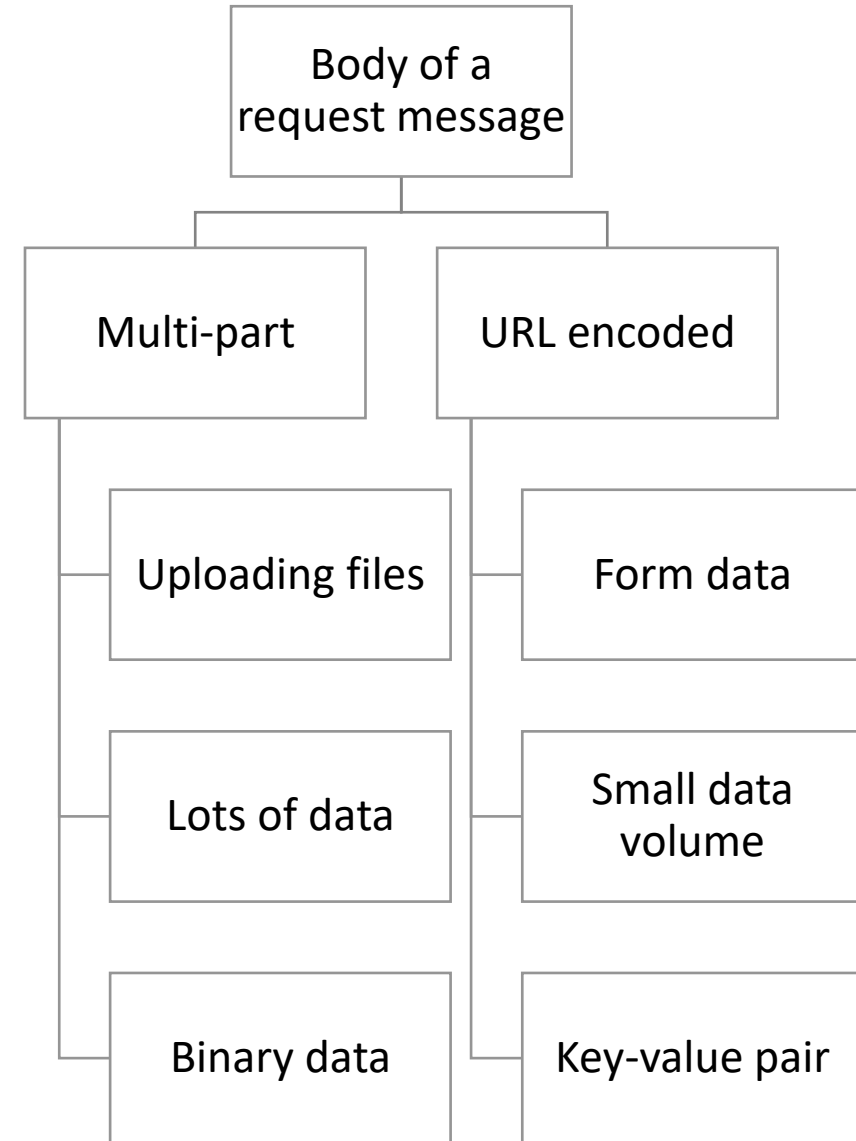
Text/plain

Text/html

- The way data is stored in the server and the way it is sent in the body may differ
  - MIME type allows this adaptation
- There should be some way of interpreting the type of data sent in body
  - Image data
- MIME type is an identifier for a particular type or format information
- All of these different MIME types are identifiers for a well known format for the data in the body of either a request or a response.
  - Based on the MIME type the data will be processed
- MIME types are changed between client and server

# Request Body Encoding

- These are written as content types



# HTTP Response

- We can't be present at the server to check what has happened
- 1XX
- 2XX
- 3XX
- 4XX
- 5XX

## HTTP Response - Read lines from socket

The diagram illustrates the structure of an HTTP response. It is divided into two main sections: the **Header** and the **Body**, indicated by red curly braces on the left. The **Header** section includes the status line 'HTTP/1.1 200 OK', where 'HTTP/1.1' is labeled as the **Version**, '200' as the **Status**, and 'OK' as the **Status Message**. Following the status line are the header fields: 'Date: Fri, 16 Mar 2018 17:36:27 GMT', 'Server: \*Your server name\*', 'Content-Type: text/html;', and 'Content-Length: 1846'. A red label '*blank line*' points to the empty line between the headers and the body. The **Body** section contains the HTML content, starting with the XML declaration '<?xml ... >', followed by the DOCTYPE declaration '<!DOCTYPE html ... >', the opening tag '<html ... >', an ellipsis '...', and the closing tag '</html>'.

```
Version      Status      Status Message
  ↓          ↓          ↓
Header { HTTP/1.1 200 OK
        Date: Fri, 16 Mar 2018 17:36:27 GMT
        Server: *Your server name*
        Content-Type: text/html;
        Content-Length: 1846
        blank line
Body { <?xml ... >
      <!DOCTYPE html ... >
      <html ... >
      ...
      </html>
```

# Response Codes

- 1XX- informal continuing process
- 2XX- successful
  - 200 means the client can assume that the server has successfully handled the request
- 3XX- redirection
  - Resend the request as the requested resource may have been moved
- 4XX- client error
  - Requested resource not found
  - Problem in request formatting
- 5XX- server error
  - The response body may contain the detailing of the error
- Depending on the response code and the MIME type, the body of the response is processed

# Cookies

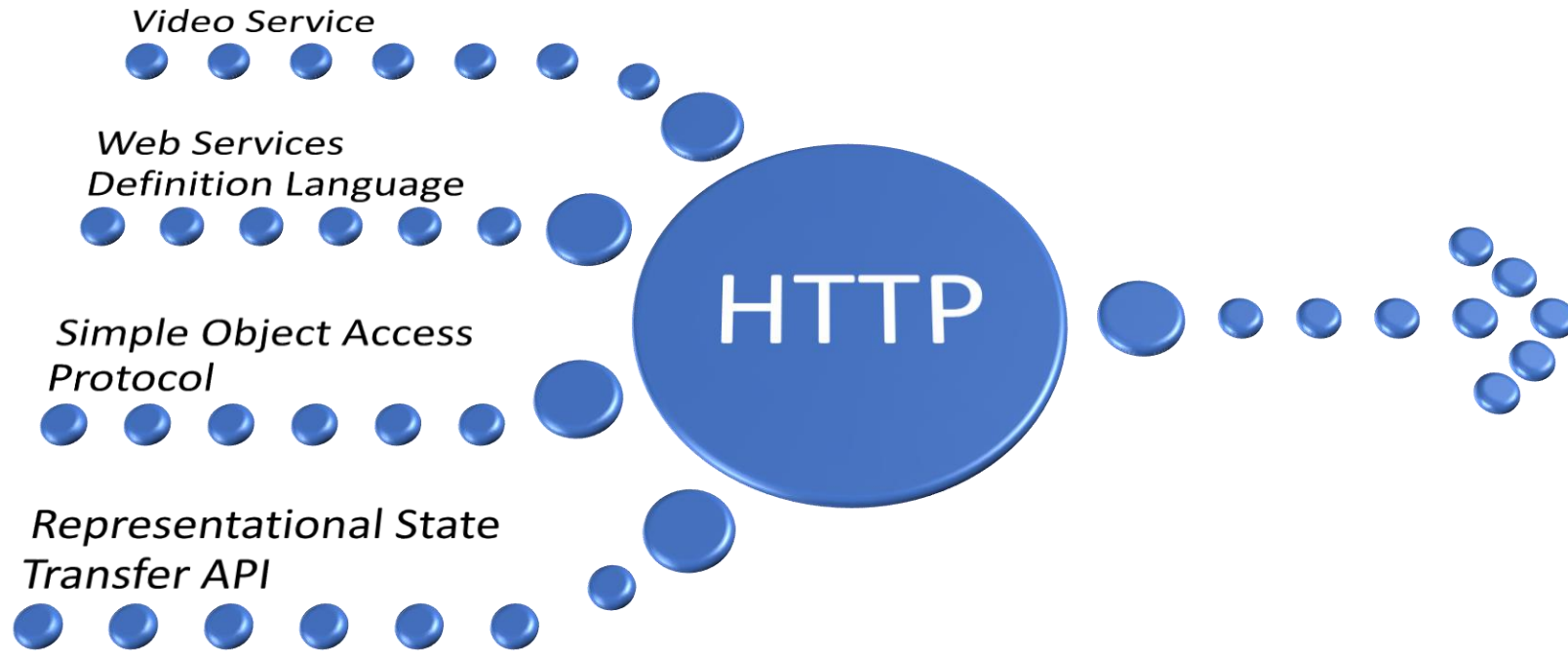
- They are very small limited pieces of data, that the server sends back to the client
- Goes through response header
- Size of the cookie has to be small
- Date/lifetime
- Encryption while sending the cookie
  - client only sends this cookie back to the server if a secure link, or an HTTPS communication protocol is being used



# Protocol Layering

- Specifying rules for interaction and exchange of messages
- formatting of those messages
- If a nice interface is designed for the protocol then other protocols could even be layered on top of this protocol
- So HTTP is just a protocol layer among many in the application
- Existing protocols on HTTP describe how you take messages or services and describe them using HTTP-based interactions

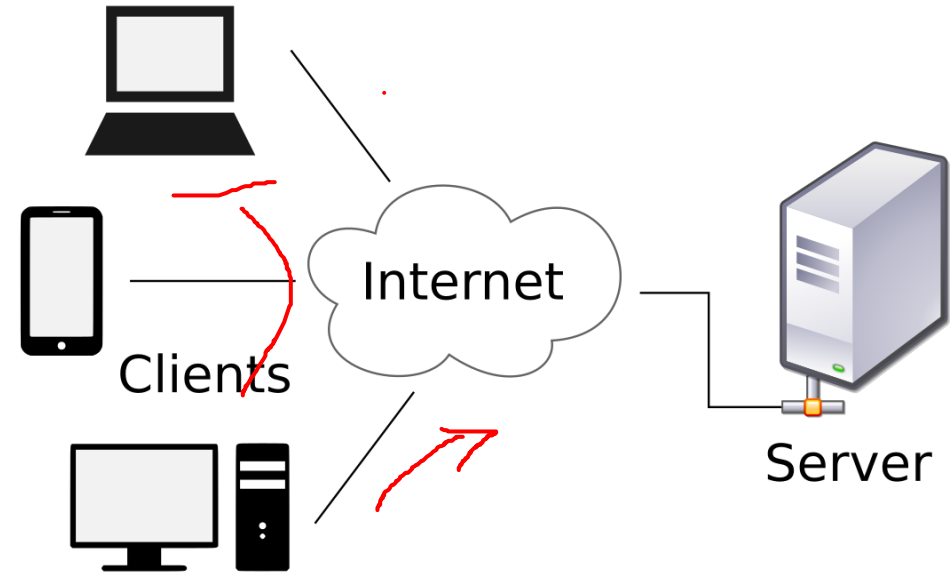
# Services on top of HTTP



# Other protocols


- Other protocols may have other protocols
- How to interface HTTP through that protocol
- How to encode parameters in the body or URL
- Web services and REST are methodologies for building on top of HTTP that describes very specific formats and principles for building things on HTTP
- Most applications support REST like services or webservices like properties
- Building in some object which can live in the cloud or some service which can live in the cloud and can receive information and be interacted with over HTTP

# HTTP is client driven



- If client 2 updates important data that client 1 just pulled in, unless client 1 makes a second request for an update the server cannot notify client 1
- Server is not able to push data as and when needed

# Pushing Data

- Manual- User may click on “refresh” when (s)he has decided to pull the data
  - Every time the window is opened data is pulled from the server
  - Periodic update
    - Overhead on the server processes as it needs to process the request even when no update is to be notified
    - Polling
    - Exponential backoff
  - Server Sent Event (SSE)
  - HTTP 2.0
    - In addition to the response to the original request, the server can push additional resources to the client
- 

# WebSockets

- communication protocol which features bi-directional, full-duplex communication over a persistent TCP connection
- Any party can push data anytime
- Single TCP connection for full duplex traffic
- Message transfer on websockets does not require all parts of HTTP to be sent (header, URL, content type, body etc.)
- Simply send binary messages or some other format back and forth in a server
- By default, port 80 is used
- Port 443 is used for connection tunneled over the TLS

# Web Socket HTTP compatibility



# Web Socket Advantages

- Stateful connection
- Message overhead of polling is less than web socket
- STOMP- Simple Text Oriented Messaging Protocol



# Web Socket

- Web sockets enable a server to push data only if the client is connected
- How to handle intermittent internet connections
  - Event mechanisms and notifications should be designed to detect disconnection and reconnection
  - Periodically reconnect/ or getting an event from Android
- Web sockets are heavy weight difficult to synchronize with more clients
- Keeping an open connection can have substantial resource impact
- For shared hosting servers, web socket is not a scalable option
- http responses can be cached by browser or by proxies
  - There is no such built-in mechanism for requests sent via webSockets

# REST or REST like Services

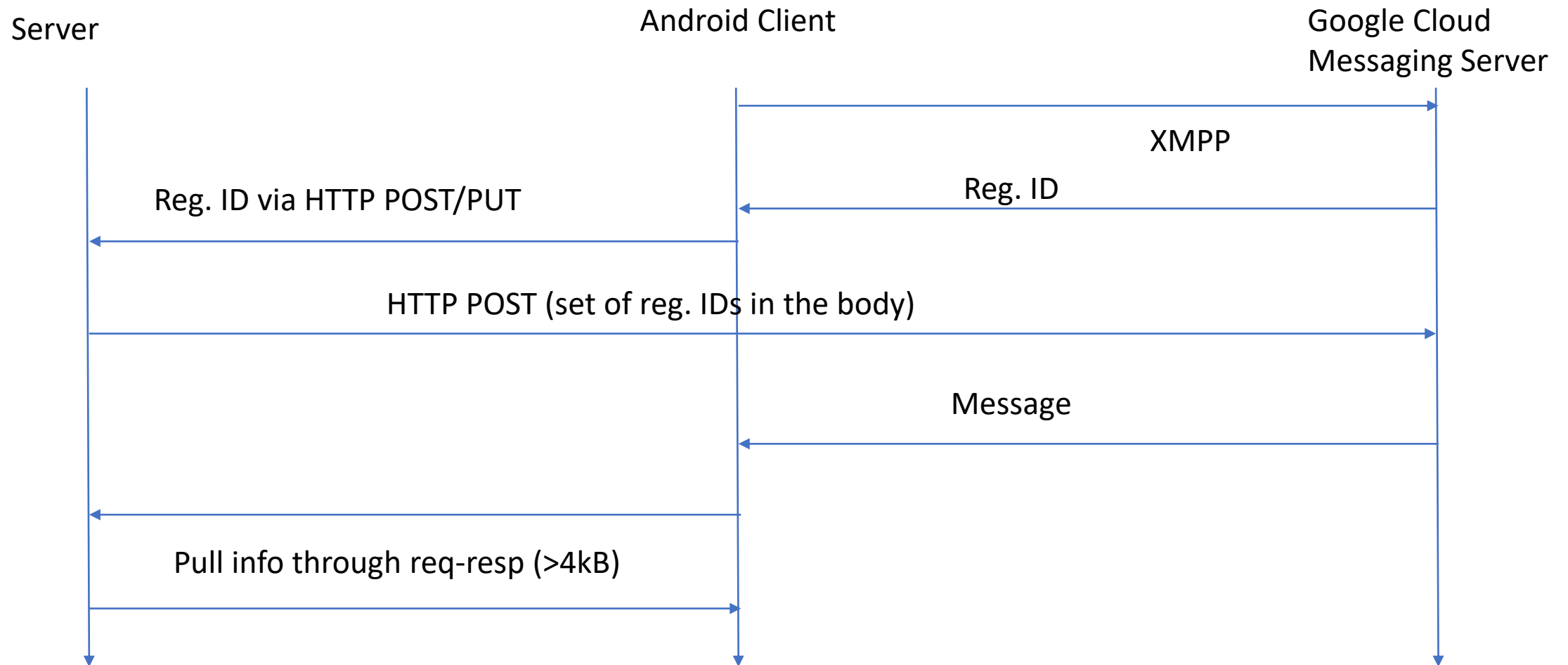
- URL Addressing scheme
- /video 

All videos list
- /video/1 

Specific video using an identifier
- /video/1/duration 

Duration of a video or specific part of the video
- GET-retrieve data
- PUT-create or replace
- POST-To add a new video and get back its URL identifier
- DELETE-remove resources at the server side

# Push Notification



# Push to Sync Model

- GCM handles the disconnection and reconnection issues through Android Cloud libraries
- Server sends an event pushed to the client that results in a pull by the client from the server
- Better than polling as the client now knows when to pull information
- Both message size and privacy issues dictate the exact design of the push notification
- Advantages
  - No need to push out large amount of information
  - The user can control the data dissemination and security over that dissemination of data