02 - Application Layer (HTTP)

▼ Application Layer Overview

- The Application Layer is the top layer of the OSI model, responsible for end-to-end communication between applications over a network.
- It provides the protocols and formats that ensure proper communication between client applications (like web browsers) and servers.

Analogy: Think of the Application Layer as the post office. It's where the process of sending and receiving messages happens, using specific rules (protocols), like sorting letters in different ways (HTTP, SMTP, etc.).

▼ Network Models: Client-Server vs. Peer-to-Peer

- **Client-Server Model:** One device (client) requests resources or services from another device (server).
- Example: When you open a website, your browser (client) requests data from the web server (server).
- **Peer-to-Peer (P2P) Model:** Devices (peers) are both servers and clients at the same time, sharing resources directly.
- Example: File sharing systems like TorrentBD.

Analogy: In a Client-Server model, imagine a library (server) where people (clients) come to borrow books. In P2P, every person (peer) has books to share and can borrow from others without needing a central library.

▼ Web and HTTP

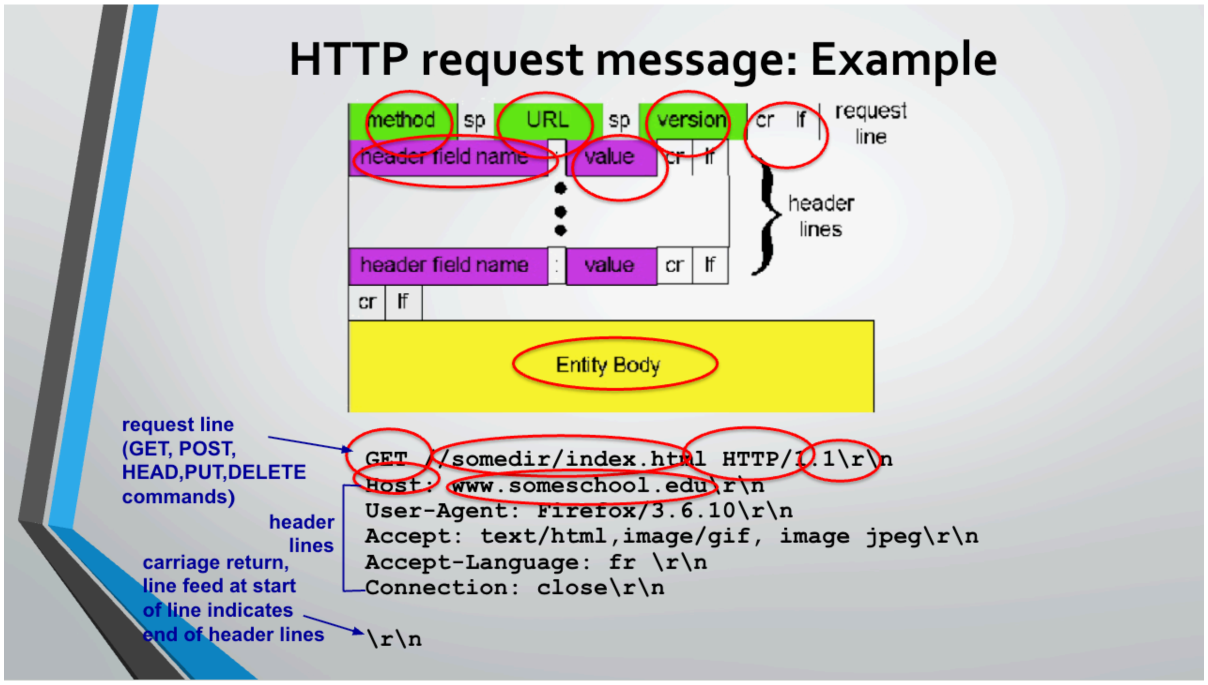
- The World Wide Web (WWW) is a collection of web pages viewed in a browser, containing text, images, and links.
- A web page is mainly composed of an HTML file (for structure) and referenced objects (like images, Java applets, etc.).
- HTTP (Hypertext Transfer Protocol) is the protocol that governs how web browsers and servers communicate to retrieve web pages.

Analogy: A web page is like a book that you read in a library (browser). Each page (file) may reference other pages (images, scripts, etc.). The HTTP is like the set of rules that the library staff (server) and readers (clients) follow to borrow books.

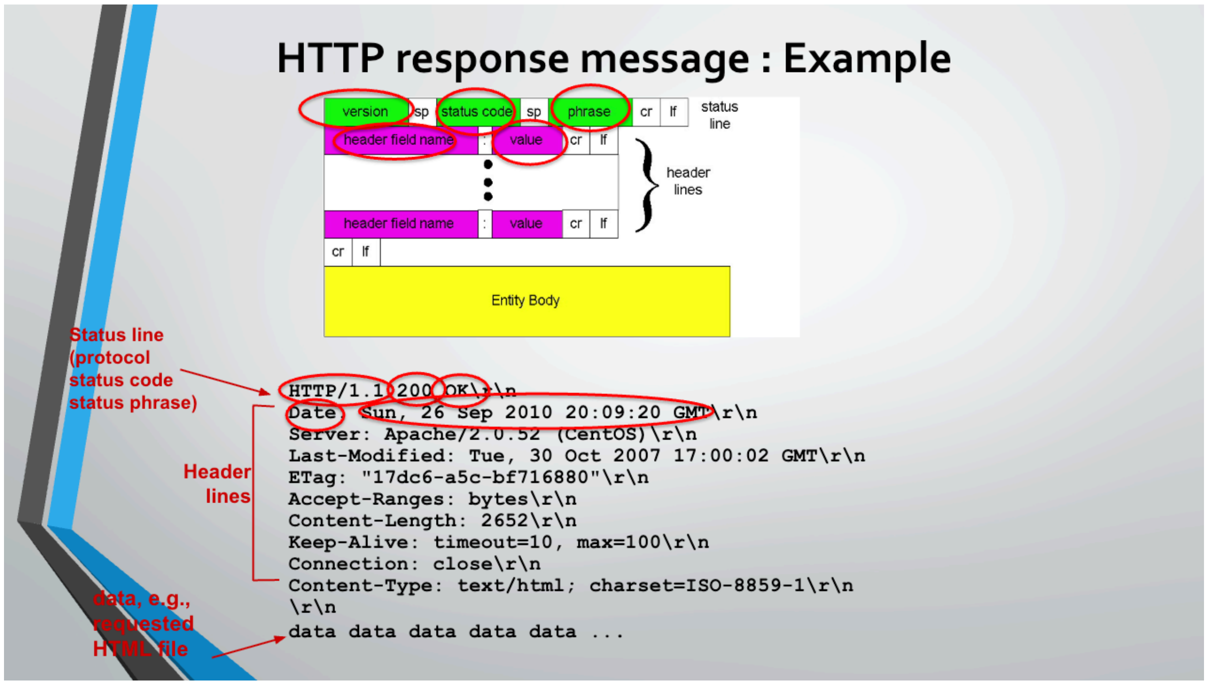
▼ HTTP Request and Response

- **HTTP Request:** The client sends a request to the server, asking for a specific resource. It can use methods like GET, POST, PUT, and DELETE.
- Example: A request like GET /index.html asks the server to return the index.html page.
- **HTTP Response:** The server sends back a response, containing a status code (like 200 OK or 404 Not Found) and the requested content.

Analogy: Sending a letter to a restaurant asking for a menu (request). The restaurant sends back the menu (response) with a note saying "Here it is!" (status code).


HTTP request message: Example
method
sp
URL
sp
version
cr
lf
request line
header field name
value
cr
lf
header lines
header field name
value
cr
lf
cr
lf
Entity Body
request line (GET, POST, HEAD,PUT,DELETE commands)
header lines
carriage return, line feed at start of line indicates end of header lines

```
GET /somedir/index.html HTTP/1.1\r\n
Host: www.someschool.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,image/gif, image jpeg\r\n
Accept-Language: fr \r\n
Connection: close\r\n
\r\n
```



HTTP response message : Example

```
HTTP/1.1 200 OK\r\n
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
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: close\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

▼ HTTP Methods

- **GET:** Requests data without altering it.
- **POST:** Sends data to the server, often used to submit forms.
- **PUT/PATCH:** Used to update or modify resources.
- **DELETE:** Removes a resource from the server.

Analogy: When ordering food,

- GET is like asking the waiter for a menu.
- POST is like submitting your food order.
- PUT/PATCH is like modifying your order (e.g., asking for no salt).
- DELETE is like canceling your order.

▼ HTTP Response Status Codes

These codes help the client understand the result of the request.

- **200 OK:** Success.
- **404 Not Found:** The requested resource does not exist.

- **301 Moved Permanently:** The resource has been permanently moved to a new location.
- **304 Not Modified:** The resources are not modified, same as previous one.
- **400 Bad Request:** The server doesn't understand the request.
- **404 Not Found:** The server cannot find the requested resource.
- **500 Internal Server Error:** There’s an issue on the server side.

▼ HTTP Connections

▼ Non-Persistent HTTP

- Every object (text, images, CSS) requires **a new TCP connection**.
- For a web page with 10 objects:
    - **10 separate TCP connections** are opened and closed.
    - Each connection has **three-way handshake and teardown overhead**, increasing latency.

Analogy:

If you want to visit a library, you check out each book one by one, returning to the front desk each time before checking out the next book.

▼ Persistent HTTP

- A **single TCP connection** is used for **multiple HTTP requests/responses**.
- Reduces **connection setup overhead**.
- Two modes:

Without Pipelining:

- Each request is sent **after the previous response is received**.
- The server handles one request at a time on the connection.
- **Reduces connection overhead but still has request-response wait times.**

Analogy:

You can borrow multiple books in a single visit, but you wait for the librarian to fetch each book before asking for the next one.

With Pipelining:

- Multiple requests are sent **back-to-back without waiting for responses**.
- Responses are received **in the same order** as requests.
- Reduces **RTT waiting delays** for sequential requests.
- Supported in **HTTP/1.1**, but not widely used due to **head-of-line blocking**.

Analogy:

You give the librarian a list of all the books you want, and they fetch them in order, allowing you to get multiple books faster without waiting between each request.

Summary Table

Aspect	Non-Persistent HTTP	Persistent HTTP (no pipeline)	Persistent HTTP (with pipeline)
TCP Connections	One per object	One per page	One per page
Request Sending	Sequential, new TCP	Sequential, same TCP	Parallel requests, same TCP
Latency	High (many handshakes)	Reduced	Further reduced
Complexity	Low	Medium	Higher
Analogy	Return each time	Wait before each book	Give list, get books in order

▼ Cookies

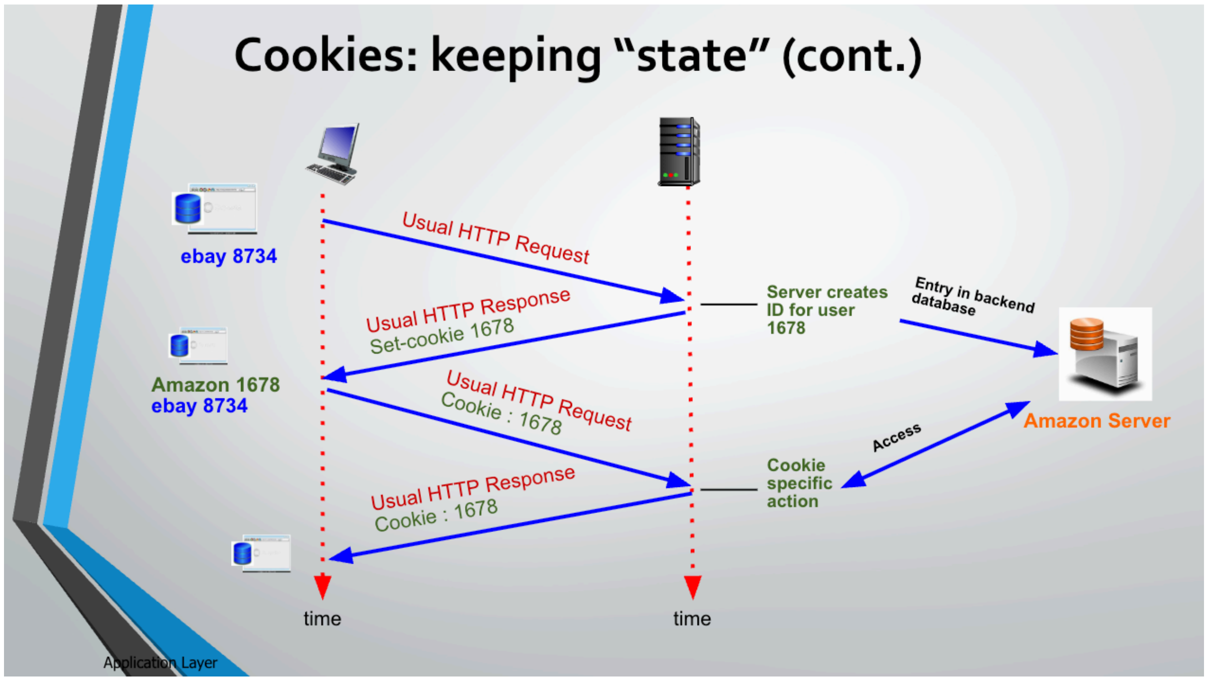
- Cookies are small pieces of data sent by the server to the client (browser) to store information about the user.

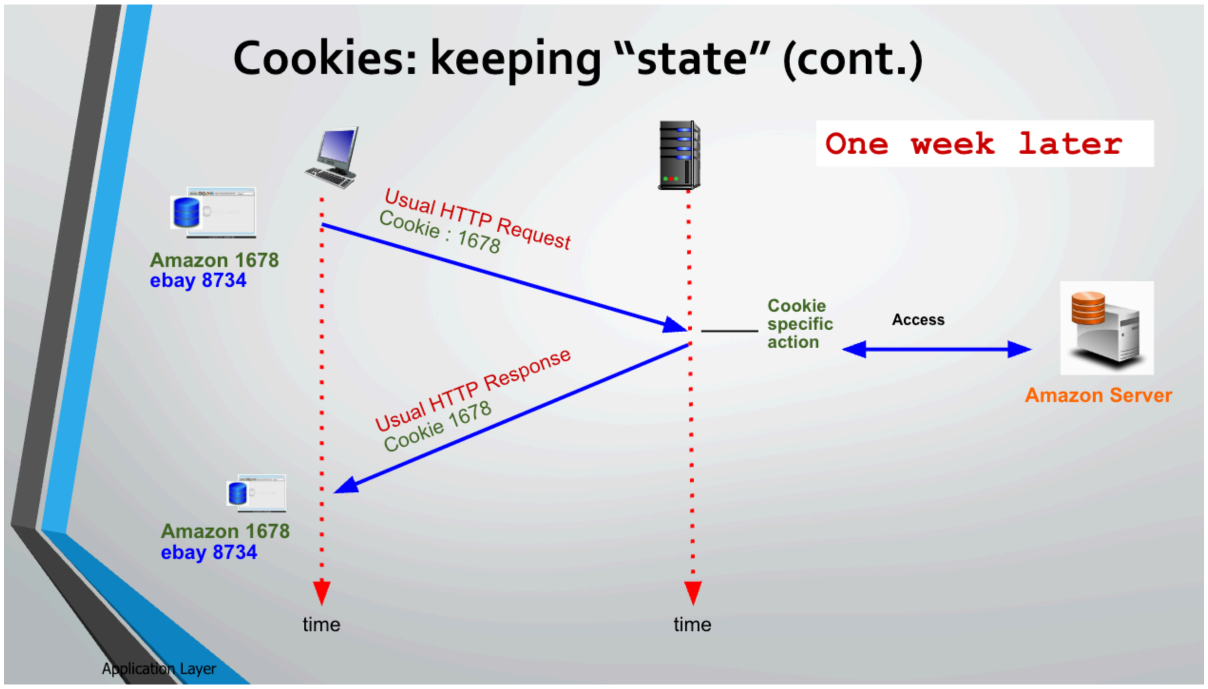
- They help in maintaining state, like keeping you logged in or storing items in your shopping cart.
- Cookie ID is added by the server
- Browser dependent. For example, if I visit facebook from both Chrome and Firefox, two browser will have two separate Cookie ID, they might be same too.

Components of cookies:

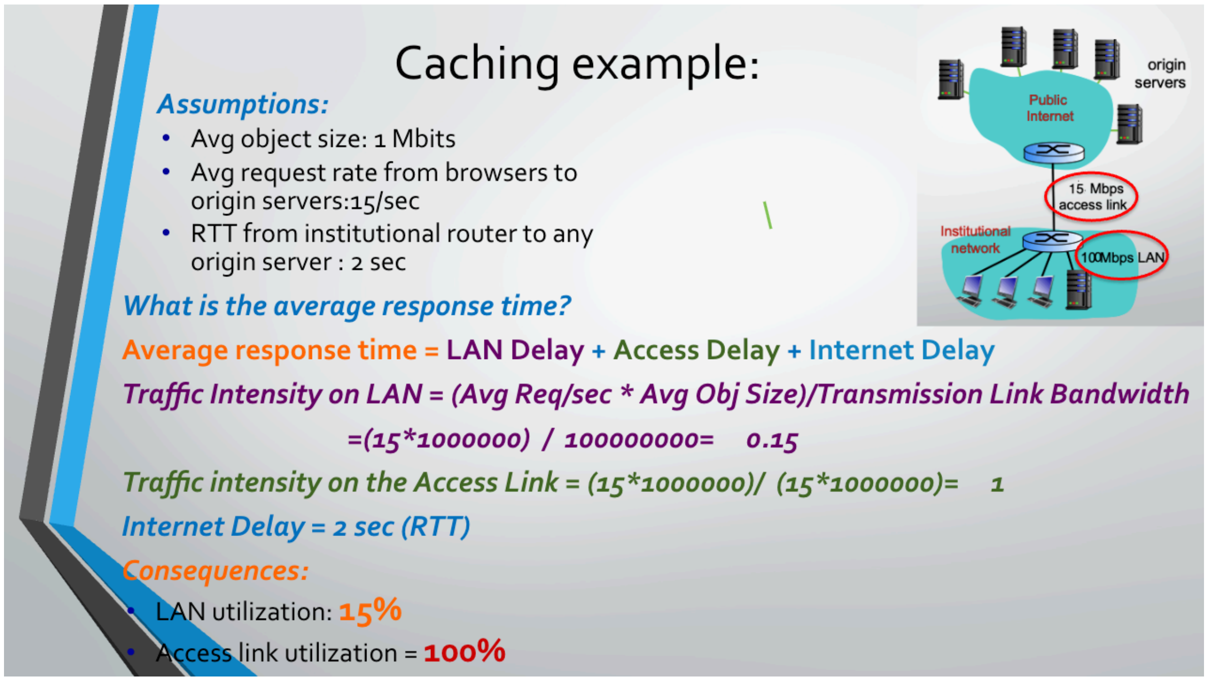
1. Cookie header line in HTTP response.
2. Cookie header line in HTTP request.
3. Cookie file on the user's computer.
4. Server's back-end database.

Analogy: Cookies are like a store that gives you a loyalty card. Every time you return, they remember who you are and your previous purchases.





Cookies: keeping "state" (cont.)



Caching example:

Assumptions:

- Avg object size: 1 Mbits
- Avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server : 2 sec

What is the average response time?

Average response time = LAN Delay + Access Delay + Internet Delay

*Traffic Intensity on LAN = (Avg Req/sec * Avg Obj Size)/Transmission Link Bandwidth*

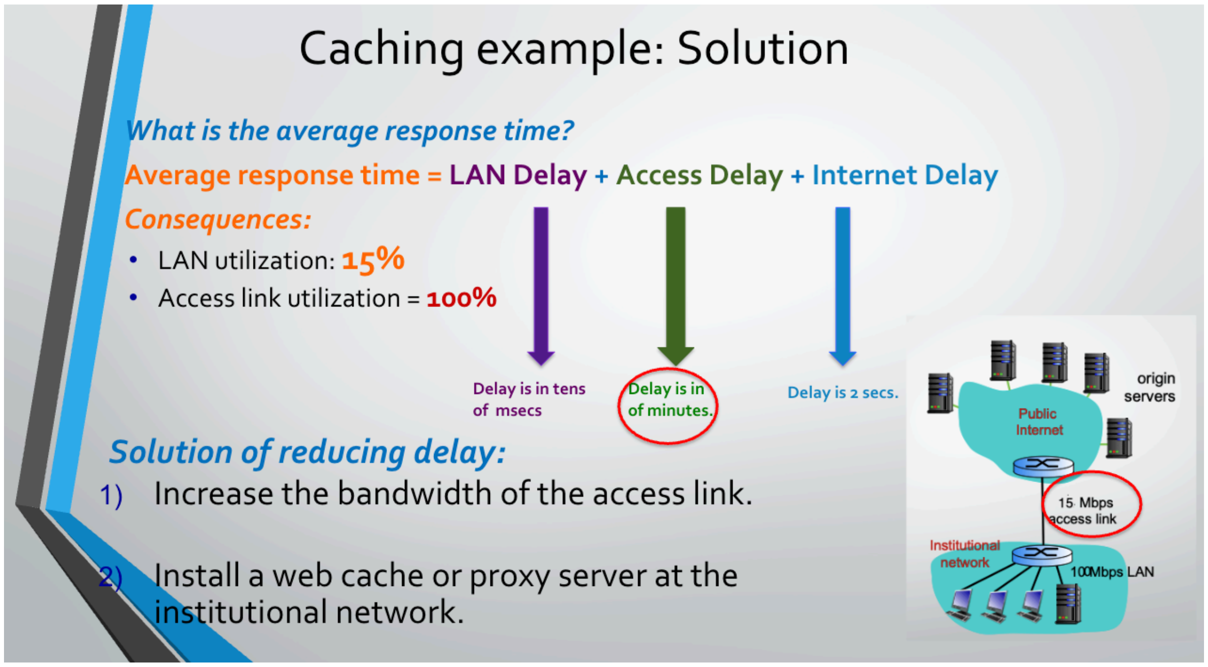
*=(15*1000000) / 100000000= 0.15*

*Traffic intensity on the Access Link = (15*1000000)/ (15*1000000)= 1*

Internet Delay = 2 sec (RTT)

Consequences:

- LAN utilization: 15%
- Access link utilization = 100%



Caching example: Solution

What is the average response time?

Average response time = LAN Delay + Access Delay + Internet Delay

Consequences:

- LAN utilization: 15%
- Access link utilization = 100%

Solution of reducing delay:

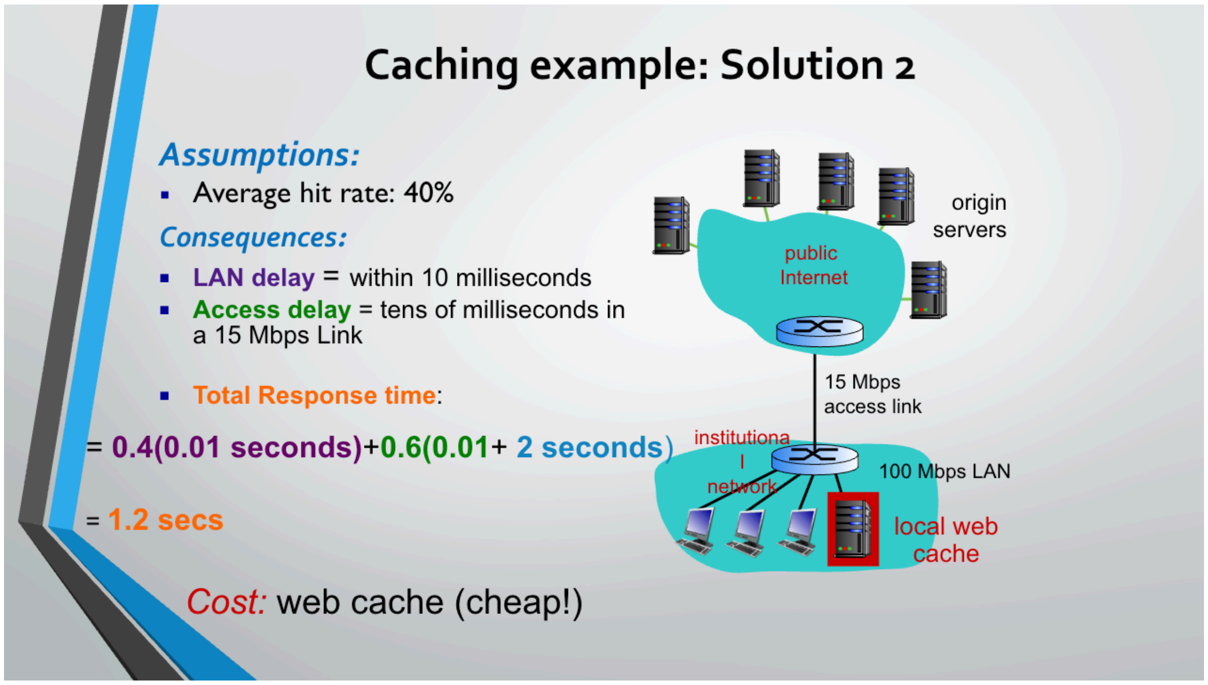
1) Increase the bandwidth of the access link.
2) Install a web cache or proxy server at the institutional network.

- Solution 01 is very costly. So, we won’t follow the solution.
- Solution 02 web caching is cheap, and a good solution.

▼ Web Caching (Proxy Servers)

- Web Cache: A proxy server that stores copies of objects to reduce loading time and server load.
- When a user requests an object, the cache checks if it’s stored locally. If yes, it serves it from there. If no, it fetches it from the origin server.
- Benefits: Faster response time, reduced bandwidth usage, and fewer requests to the origin server.

Analogy: Imagine a library (web cache) that keeps a copy of books (objects). When you request a book, if they already have it, they give it to you quickly without sending a staff member to retrieve it from the warehouse (origin server).



Caching example: Solution 2

Assumptions:
- Average hit rate: 40%

Consequences:
- LAN delay = within 10 milliseconds
- Access delay = tens of milliseconds in a 15 Mbps Link
- Total Response time:

= 0.4(0.01 seconds)+0.6(0.01+ 2 seconds)

= 1.2 secs

Cost: web cache (cheap!)

▼ Stale Cache

One problem of using Proxy server

- **Problem:** Cached object at the client (or proxy) may be **outdated if the original object on the web server has been modified** since caching.
- This leads to **clients receiving stale (old) content.**

HTTP provides a mechanism to **verify if cached objects are up to date**, preventing stale cache issues.

Conditional GET

- Uses the **GET method**.
- Includes the header: `If-Modified-Since: <date>`

  This tells the server:

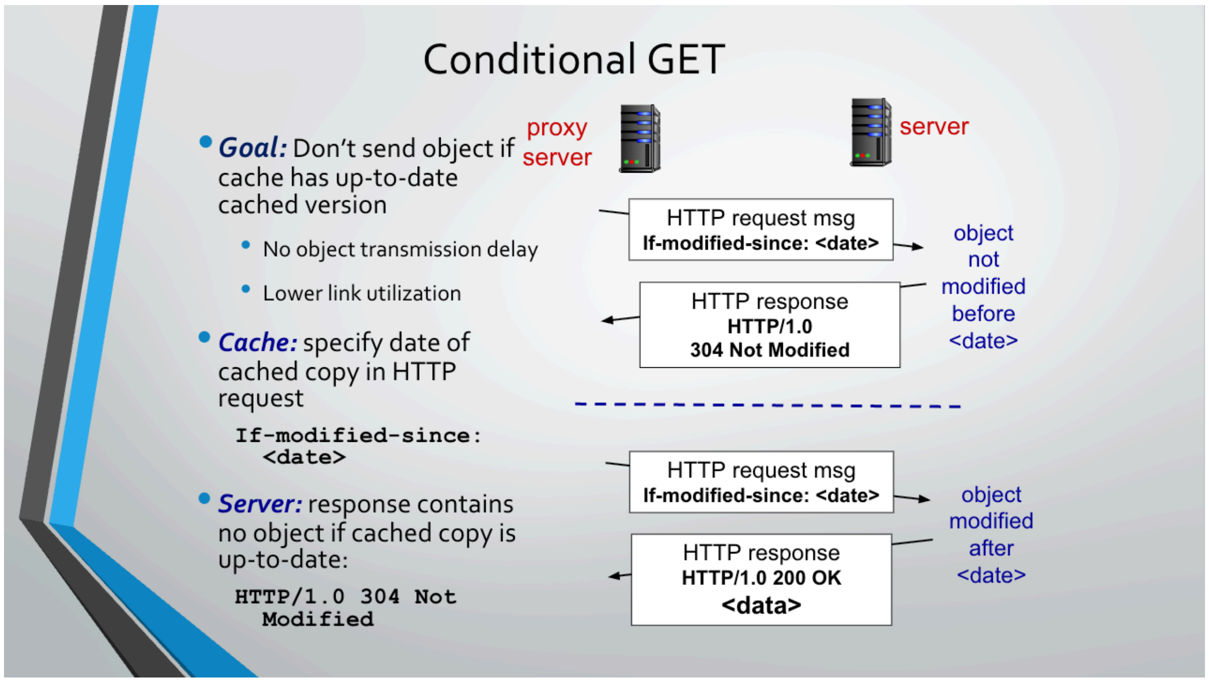
  "Send the object only if it has been modified since the specified date."

Server Response:

- If **not modified:** `HTTP/1.0 304 Not Modified`

  allowing the cache to **reuse the stored copy** without re-downloading.
- If **modified:** `HTTP/1.0 200 OK`

Sends the **updated object** to the client or proxy cache.



Conditional GET

- ***Goal:*** Don't send object if cache has up-to-date cached version
  - No object transmission delay
  - Lower link utilization
- ***Cache:*** specify date of cached copy in HTTP request

```
If-modified-since:
  <date>
```

- ***Server:*** response contains no object if cached copy is up-to-date:

```
HTTP/1.0 304 Not
  Modified
```

▼ HTTPS

- HTTPS (Hypertext Transfer Protocol Secure): It's a secure version of HTTP
- It ensures data security, like encrypting login credentials, ensuring privacy during online transactions.
- Transport Layer Security
- Use Secure Socket Layer(SSL)
- Transmit data over 443 port number
- Website need to install the signed SSL certificate
- SSL protocol encrypts the data which the client transmit to the server

Analogy: HTTPS is like sending a letter in a locked envelope that only the recipient can open, ensuring that no one else can read it during delivery.

Digital Certificate is a file that contains

- Information about the organization to help it authenticate

- Cryptographic information that helps site users communicate with it securely through encryption

SEO advantage - GOOGLE gives the preferences to those websites that use HTTPS rather than the websites that use HTTP.