

Application Layer: The Web & HTTP

CPSC 433/533, Spring 2021
Anurag Khandelwal

The Web: Precursor



Ted Nelson

- **1967, Ted Nelson, Project Xanadu:**
 - A world-wide publishing network that would allow information to be stored not as separate files, but as connected literature
 - Owners of documents would be automatically paid via electronic means for the virtual copying of their data
- **Coined the term “Hypertext”**

The Web: History



Tim Berners-Lee

- World Wide Web (WWW): A distributed database of “pages” linked through the **Hypertext Transport Protocol (HTTP)**
 - First implementation - 1990
 - Tim Berners-Lee at CERN
 - HTTP/0.9 - 1991
 - Simple GET command for the Web
 - HTTP/1.0 - 1992
 - Client/server information, simple caching
 - HTTP/1.1 - 1996
 - HTTP/2 - 2015

Web Components

- **Infrastructure:**
 - Clients
 - Servers
- **Content:**
 - URL: naming content
 - HTML: formatting content
- **Protocol for exchanging information: HTTP**

Uniform Resource Locator (URL)

`protocol://host-name[:port]/directory-path/resource`

- Extend the idea of hierarchical hostnames to include anything in a filesystem
 - `http://cpsc.yale.edu/people/anurag-khandelwal/myfile.txt`
- Extend to program executions as well...
 - `http://us.f4l3.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`
 - Server-side processing can be incorporated in the name

Uniform Resource Locator (URL)

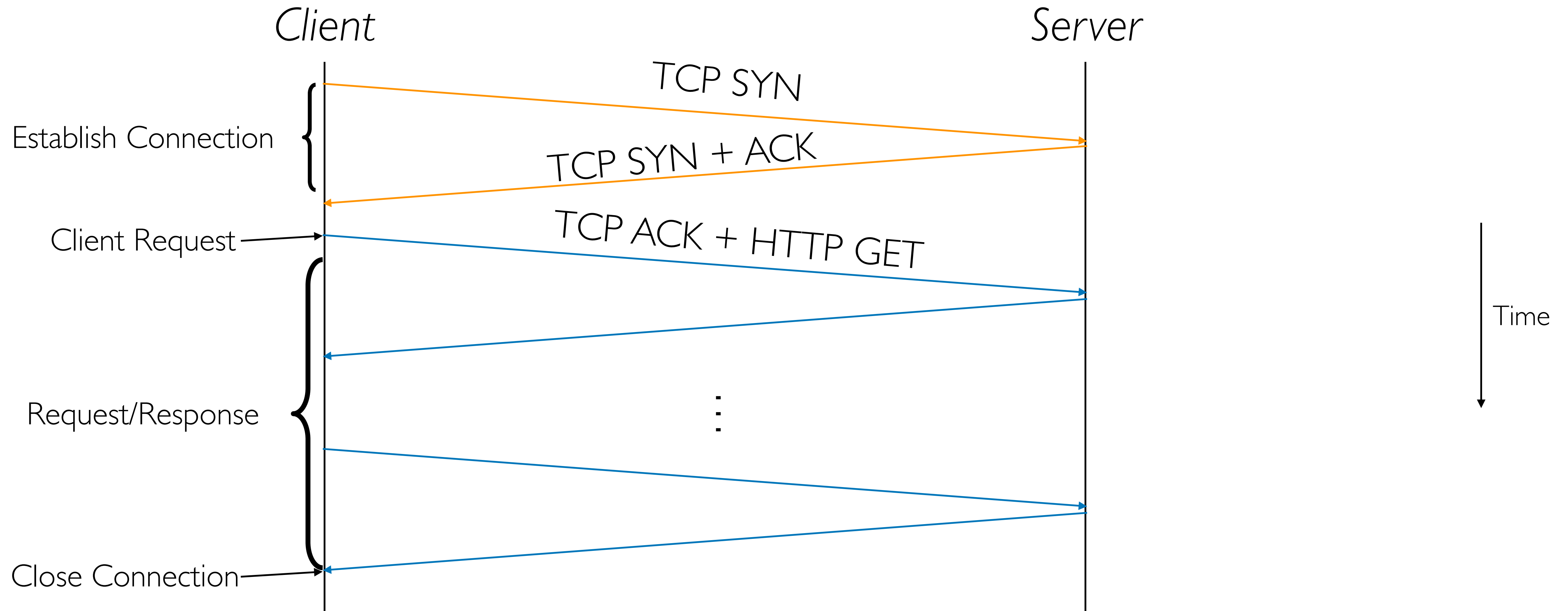
`protocol://host-name[:port]/directory-path/resource`

- `protocol`: http, ftp, https, smtp, rtsp, etc.
- `host-name`: DNS name, IP address
- `port`: defaults to protocol's standard port; e.g., http: 80, https: 443
- `directory-path`: hierarchical, reflecting filesystem
- `resource`: identifies the desired resource

Hyper Text Transfer Protocol (HTTP)

- **Client-server architecture**
 - Server is “always on” and “well known”
 - Clients initiate contact to server
- **Synchronous request/reply protocol**
 - Runs over TCP, port 80
- **ASCII format**
- **Stateless**

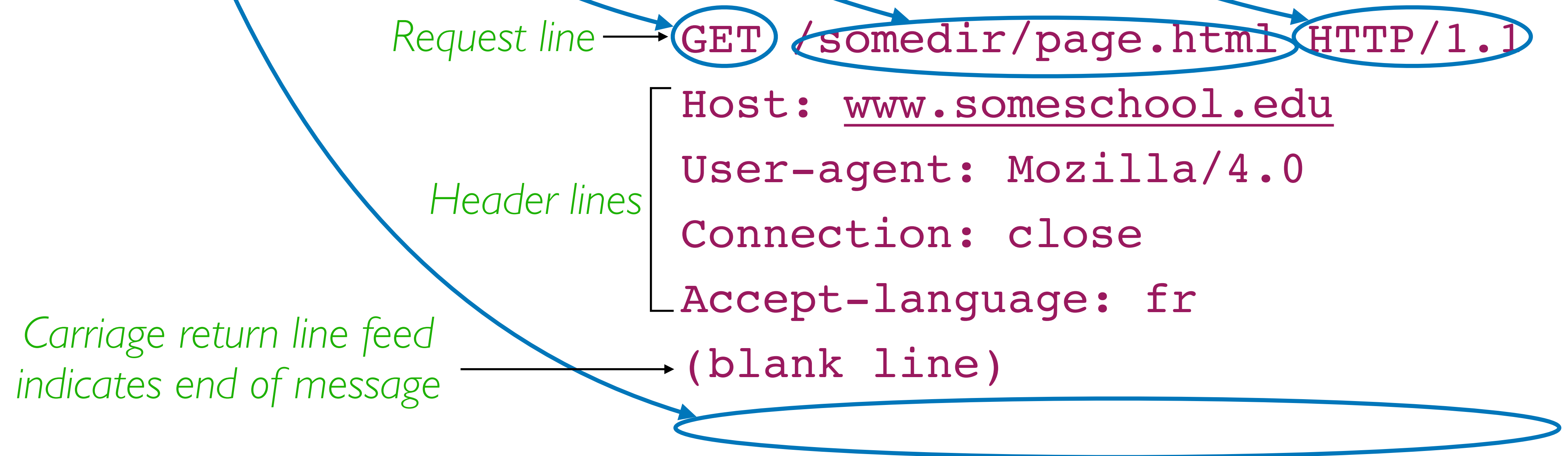
Steps in HTTP Request/Response



Client-to-Server Communication

- HTTP Request Message

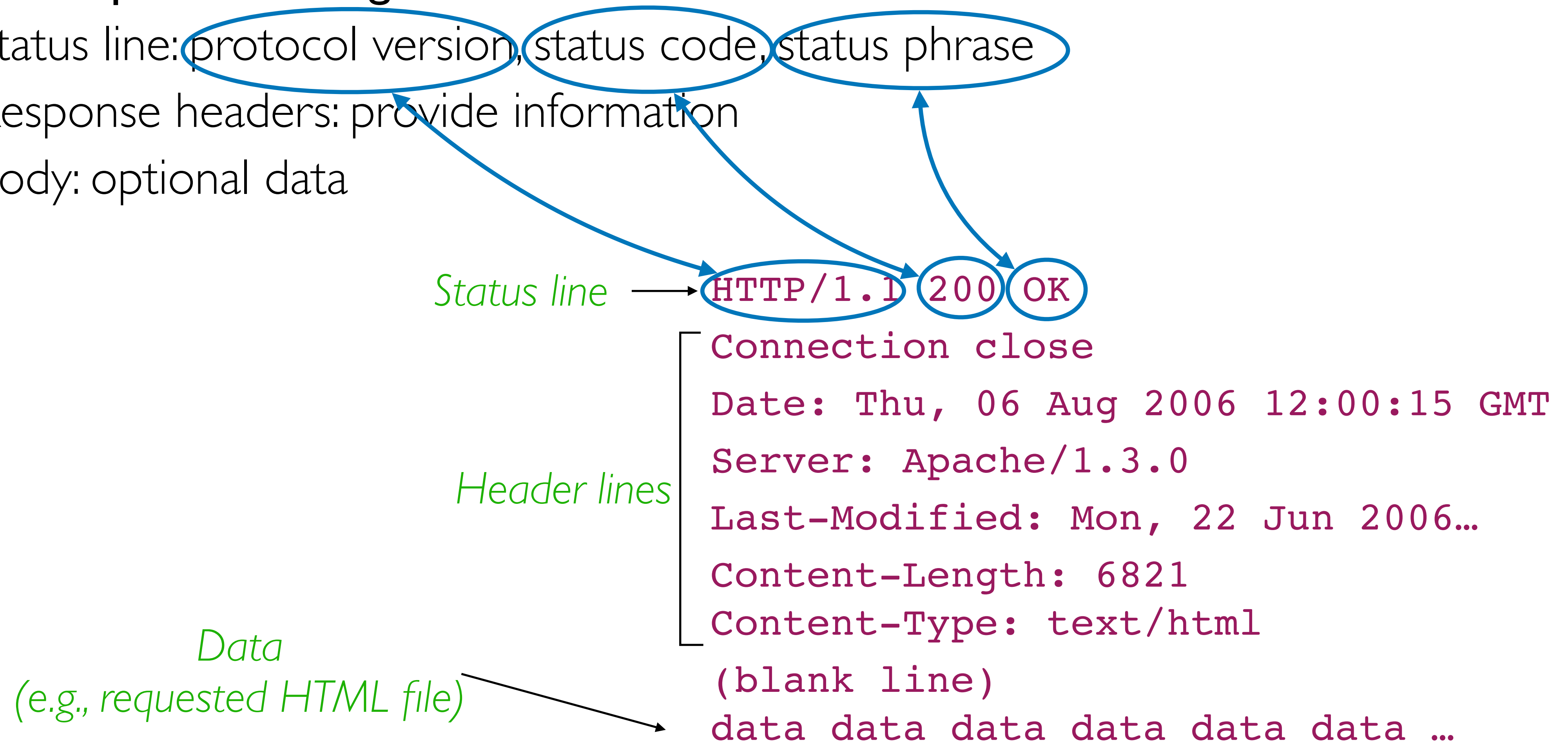
- Request line: method, resource, and protocol version
- Request headers: provide information or modify request
- Body: optional data (e.g., to “POST” data to the server)



Server-to-Client Communication

- HTTP Response Message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional data



Questions?

HTTP is *Stateless*

- **Each request-response treated independently**
 - Servers *not* required to retain state from clients
 - Pros vs cons?
- **Good: Improves scalability on the server-side**
 - Failure handling is easier
 - Can handle higher rate of requests
 - Order of requests doesn't matter
- **Bad: Some applications need persistent state**
 - Need to uniquely identify user or store temporary information
 - e.g., Shopping cart, user profiles, usage tracking, ...

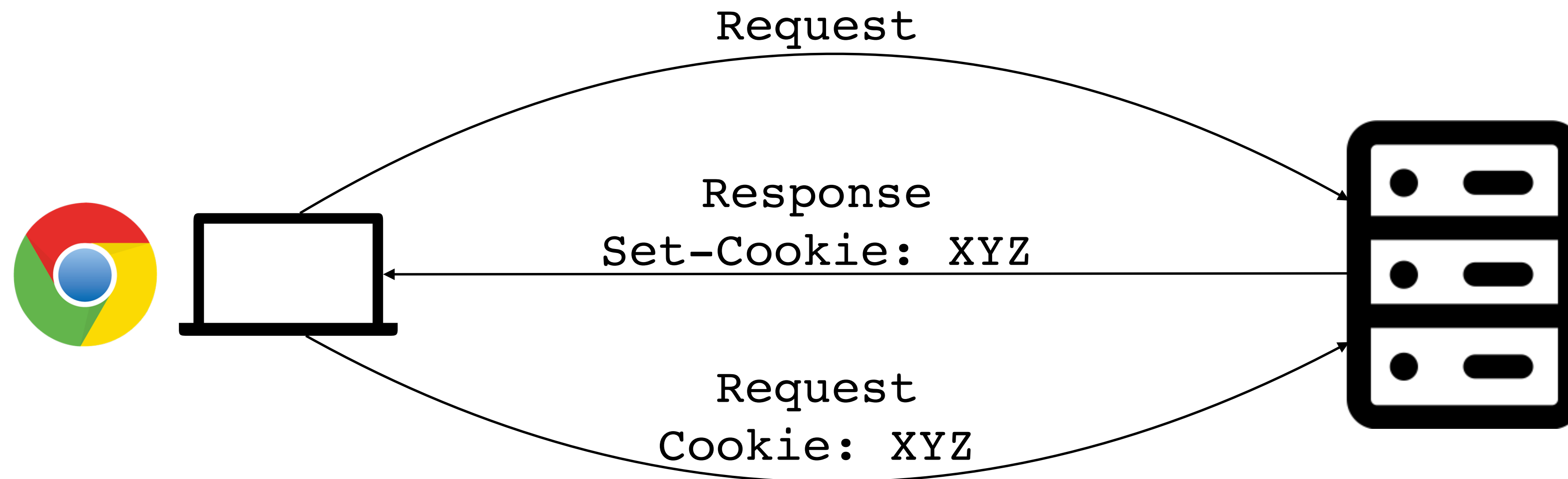
Question

- How does a stateless protocol keep state?



State in a Stateless Protocol: Cookies

- *Client-side* state maintenance
 - Client store small state on behalf of server
 - Client sends state in future requests to the server
- Can provide authentication



Questions?

HTTP Performance Issues

Performance Goals

- **Users**
 - Fast downloads (not the same as low-latency communication!)
 - High availability
- **Content provider**
 - Happy users (hence, above)
 - Cost-effective delivery infrastructure
- **Network (secondary)**
 - Avoid overload

Solutions?

- **Users**

- Fast downloads (not the same as low-latency communication!)
- High availability

Improve HTTP to compensate for TCP weak-spots

1

- **Content provider**

- Happy users (hence, above)
- Cost-effective delivery infrastructure

Caching & Replication

2

- **Network (secondary)**

- Avoid overload

Exploit economies of scale
(Webhosting, CDNs, datacenters)

3

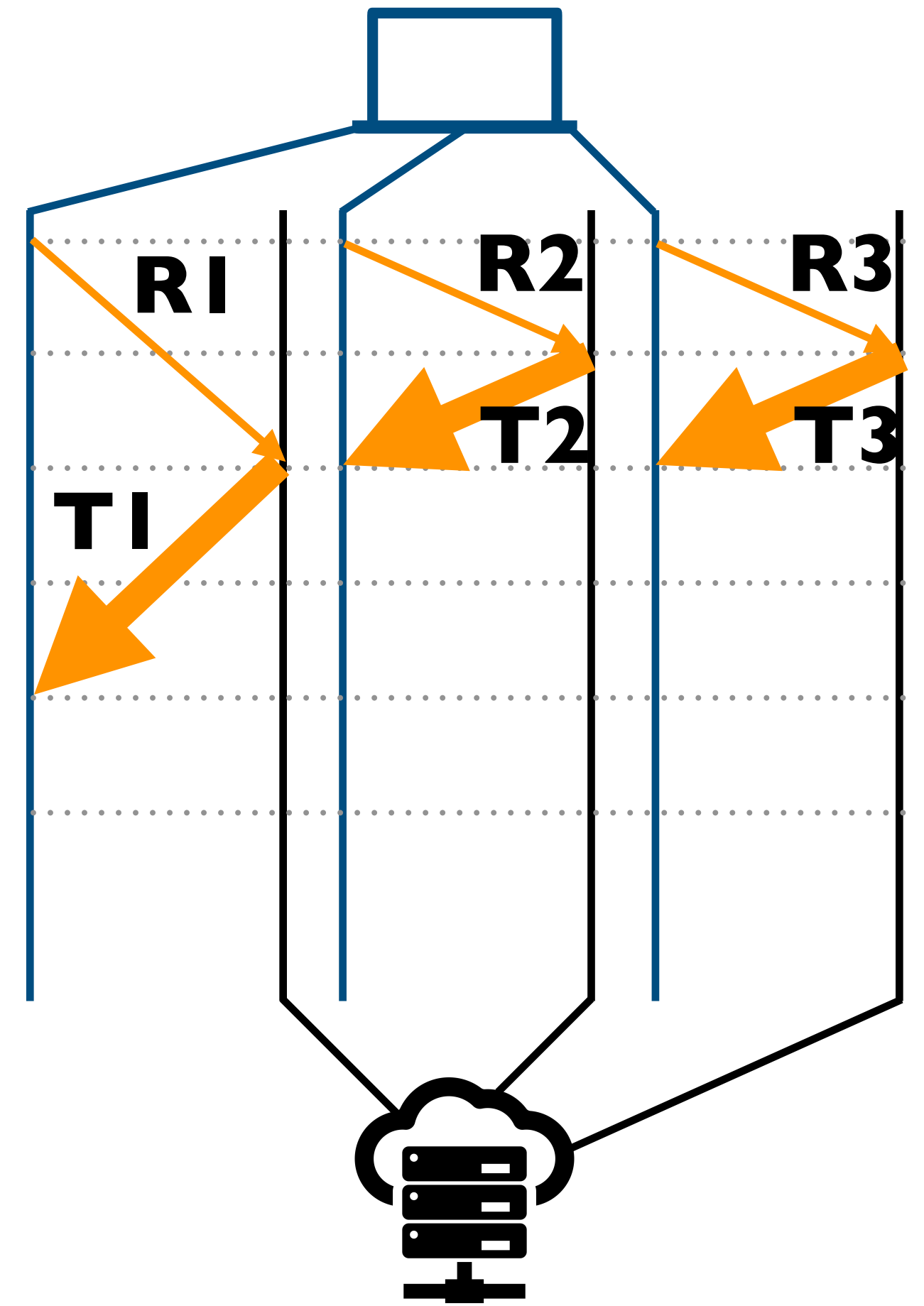
HTTP Performance on TCP

- Most webpages have multiple objects
 - e.g., HTML and a bunch of embedded images
- How do you retrieve those objects (naively)?
 - One object at a time
- New TCP connection per (small) object!

Improving HTTP Performance:

Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
- Client: 😊
- Content provider: 😊
- Network: 😞 (Why?)



Improving HTTP Performance:

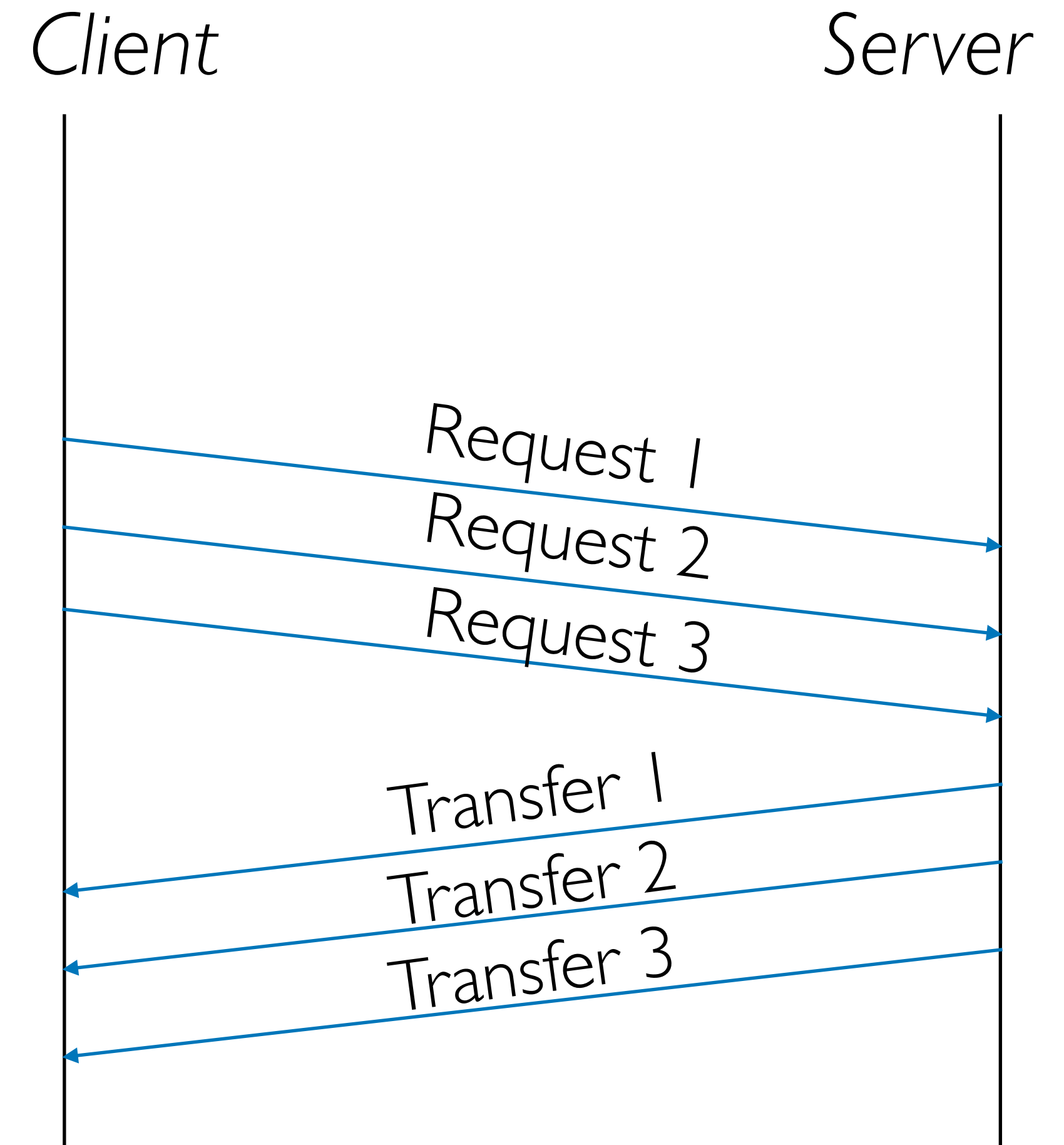
Persistent Connections

- **Maintain TCP connection across multiple requests**
 - Including transfers after current page
 - Client or server can tear down connection
- **Performance advantages**
 - Avoid overhead of connection setup and teardown
 - Allow TCP to learn more accurate RTT estimate
 - Allow TCP congestion window to increase, i.e., leverage previously discovered bandwidth
- **Default in HTTP/1.1**

Improving HTTP Performance:

Pipelined Requests & Responses

- Batch requests & responses to reduce #of packets
- Multiple requests can be contained in 1 TCP segment



Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time:
- M concurrent:
- Persistent:
- Pipelined:
- Pipelined/Persistent:

Scorecard: Getting n Large Objects of Size F

Time dominated by bandwidth

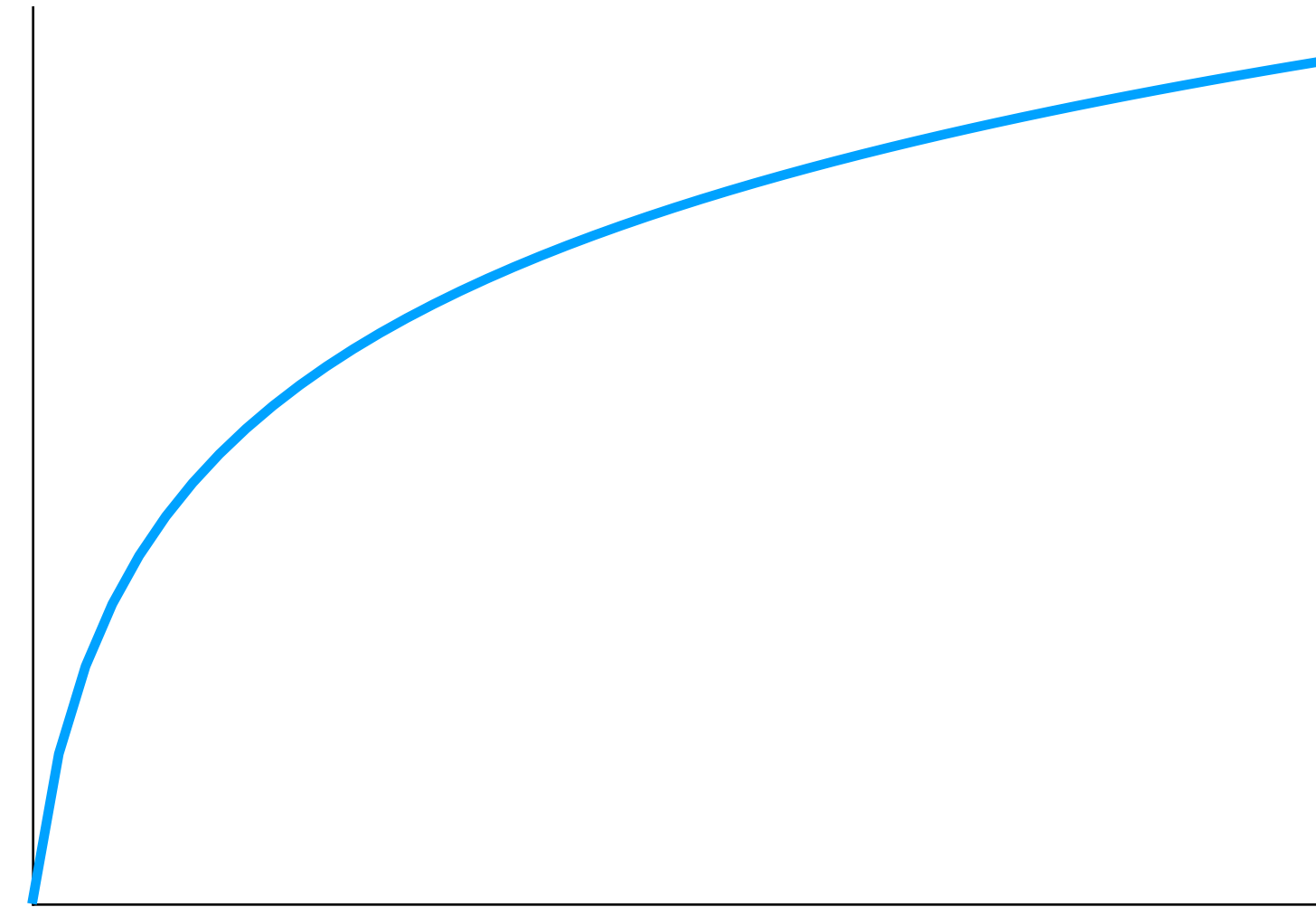
- **One-at-a-time:**
- **M concurrent:**
 - Assuming shared with large population of users
 - And each TCP connection gets the same bandwidth
- **Pipelined and/or Persistent:**
 - The only thing that helps is getting more bandwidth..

Questions?

Improving HTTP Performance:

Caching

- **Why does caching work?**
 - Exploits *locality of reference*
- **How well does caching work?**
 - Very well, up to a limit
 - Large overlap in content
 - But many unique requests
 - *A universal story!*
 - *Effectiveness of caching grows logarithmically with size*



Caching: How

- Modifier to GET requests:
 - **If-modified-since** — returns “not modified” if resource not modified since specified time

- Client specifies “if-modified-since” time in request
- Server compares this against “last modified” time of resource
- Server returns “not modified” if resource has not changed
- ... or an “OK” with the latest version otherwise

Caching: How

- Modifier to GET requests:
 - **If-modified-since** — returns “not modified” if resource not modified since specified time
- Response header:
 - **Expires** — how long its safe to cache the resource
 - **No-cache** — ignore all caches; always get resource directly from server

Improving HTTP Performance:

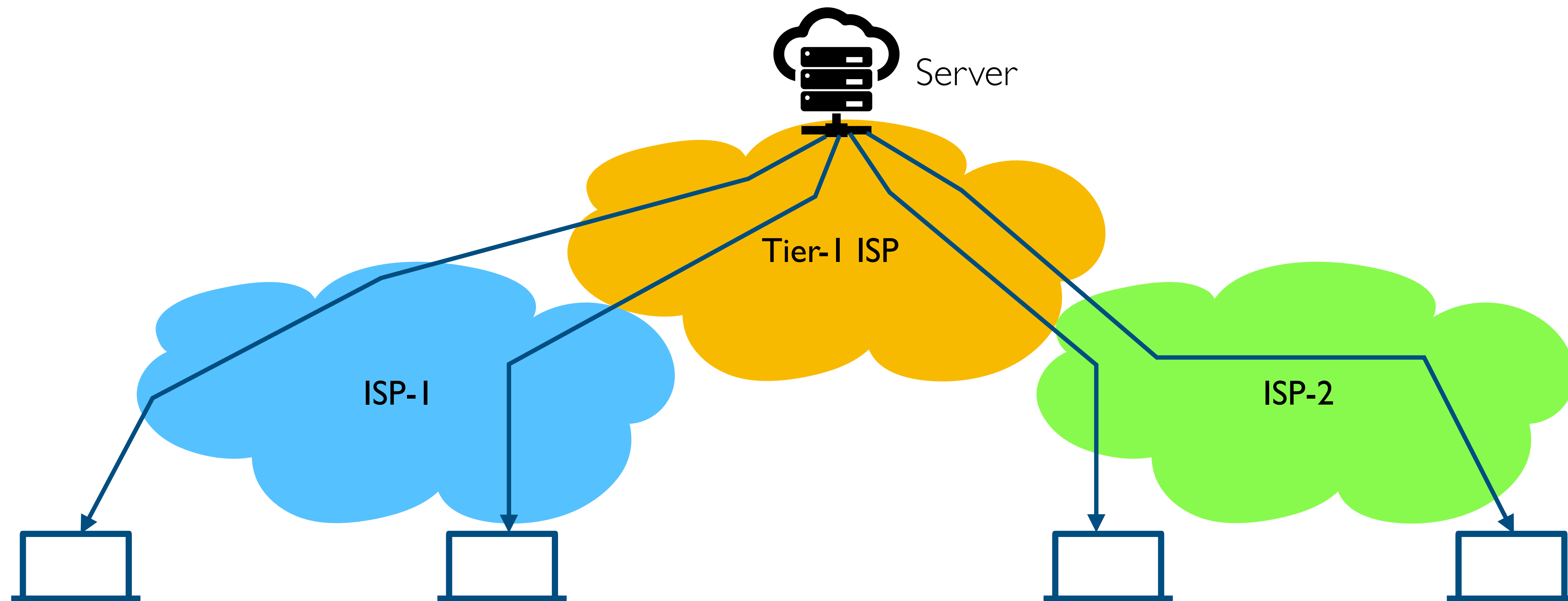
Caching: Where

- Options:
 - Client
 - Forward proxies
 - Reverse proxies
 - Content Distribution Network

Improving HTTP Performance:

Caching: Where

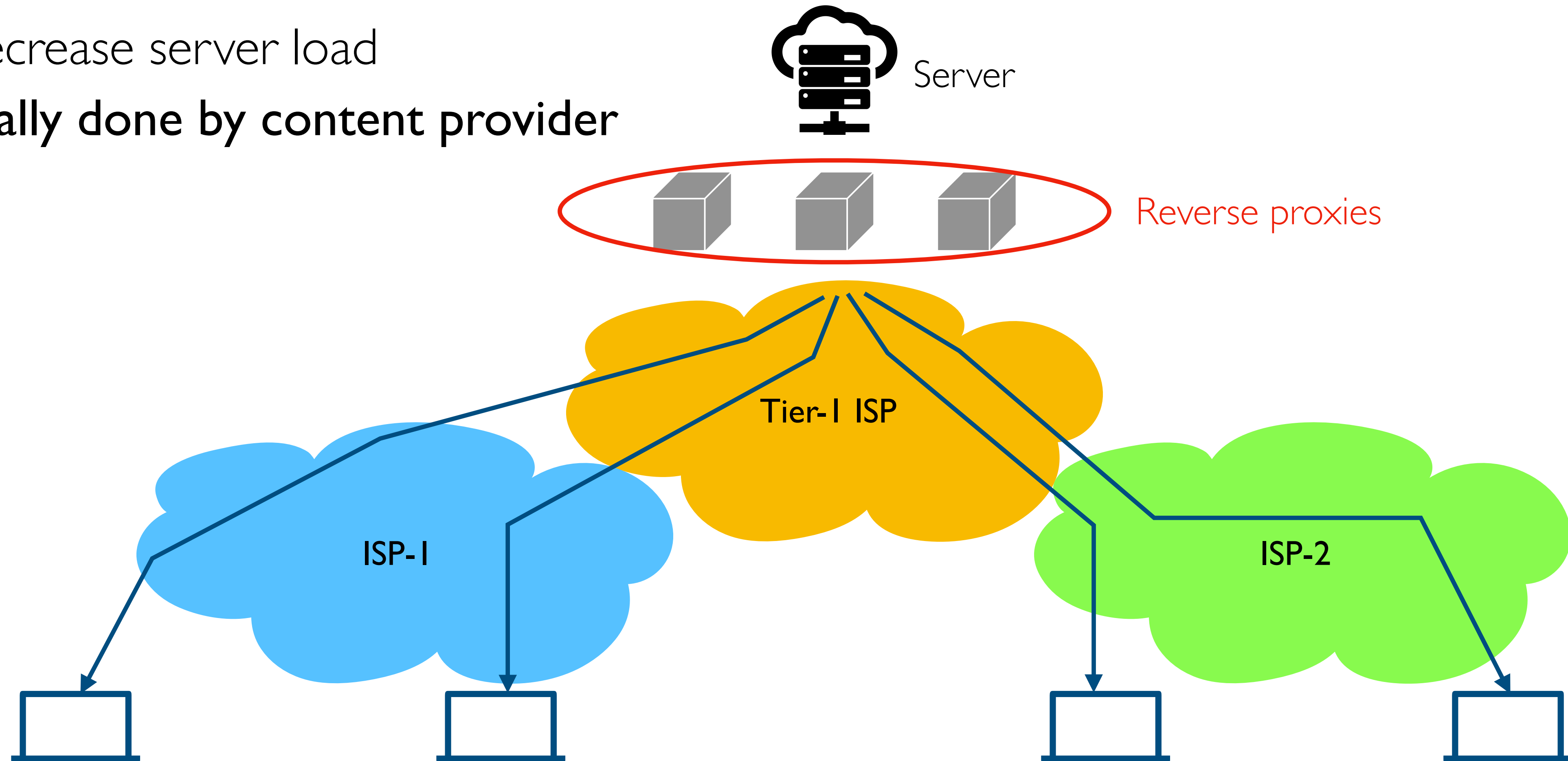
- **Baseline:** many clients transfer same information
 - Generate unnecessary server and network load
 - Clients experience unnecessary latency



Improving HTTP Performance:

Caching: Where

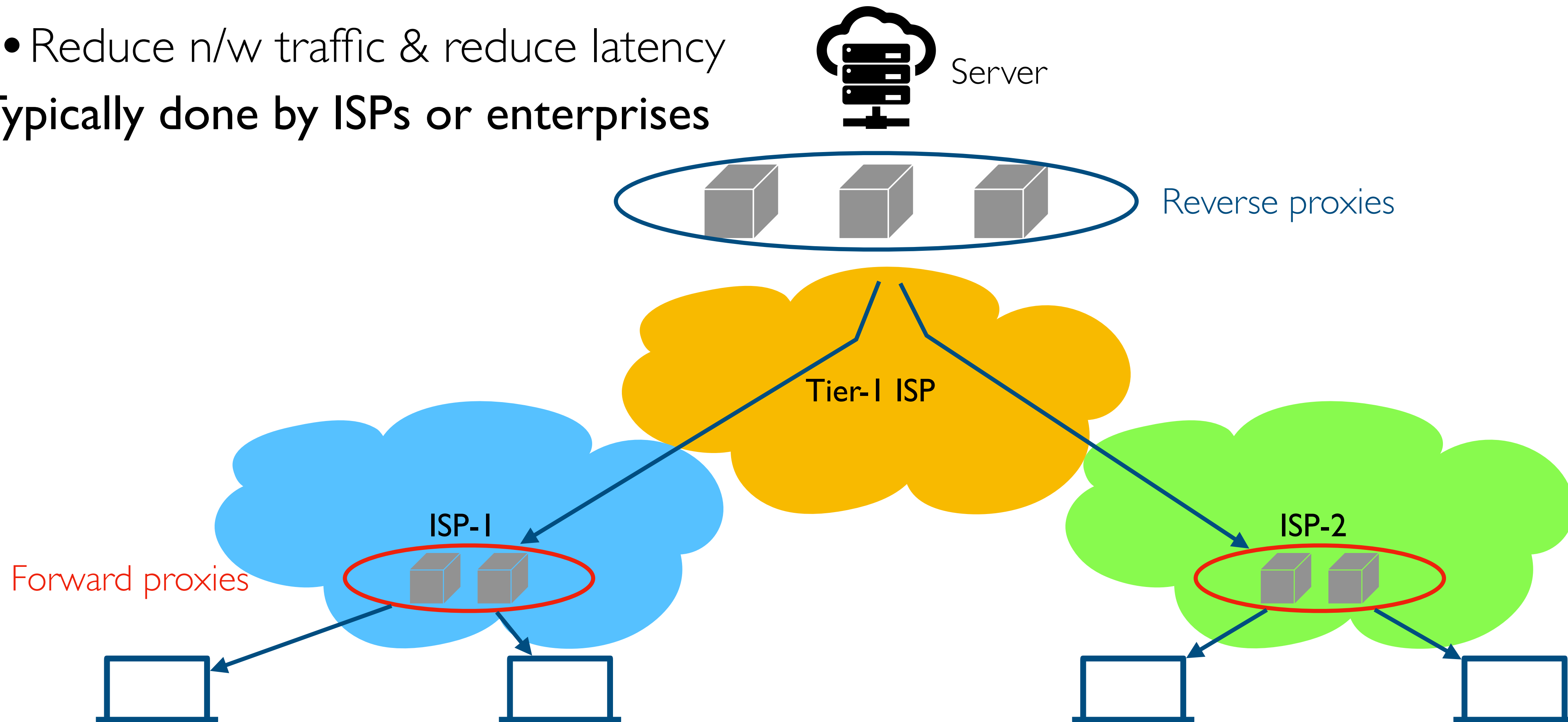
- Cache documents close to **server**
 - Decrease server load
- Typically done by content provider



Improving HTTP Performance:

Caching: Where

- Cache documents close to **clients**
 - Reduce n/w traffic & reduce latency
- Typically done by ISPs or enterprises



Improving HTTP Performance:

Replication

- **Replicate popular websites across many machines**
 - Spreads load on servers
 - Places content closer to clients
 - Helps when content isn't cacheable
- **Problem: Want to direct client to particular replica**
 - Balance load across server replicas
 - Pair clients with nearby servers
- **Common solution:**
 - DNS returns different addresses based on clients geo-location, server load, etc.