# Network Apps: Overview of Socket API
# Network Apps: HTTP & Content Delivery

David Marchant

Based on slides compiled by Marcos Vaz Salles, with adaptiona by Vivek Shah and Michal Kirkedal Thomsen

# Admin

- No lecture or exercises on Wednesday 10th
- Will be held on Friday 12th
- Same time, same place, different date

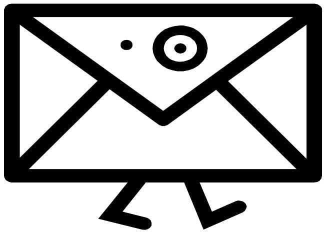
Source: Freedman

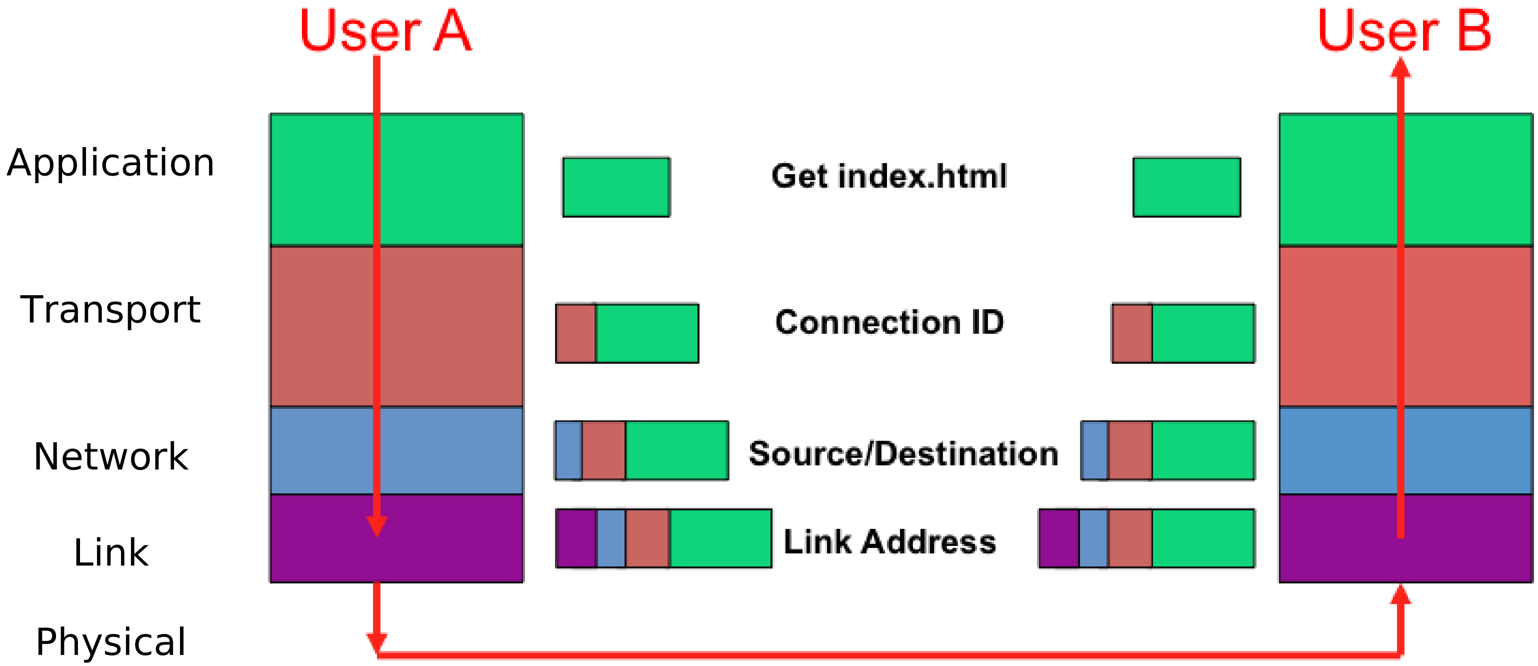
# Recap: Key Concepts in Networking

- Protocols
  - Speaking the same language
  - Syntax and semantics
- Layering
  - Standing on the shoulders of giants
  - A key to managing complexity
- Resource allocation
  - Dividing scarce resources among competing parties
  - Memory, link bandwidth, wireless spectrum, paths
- Naming
  - What to call computers, services, protocols, …

Source: Freedman

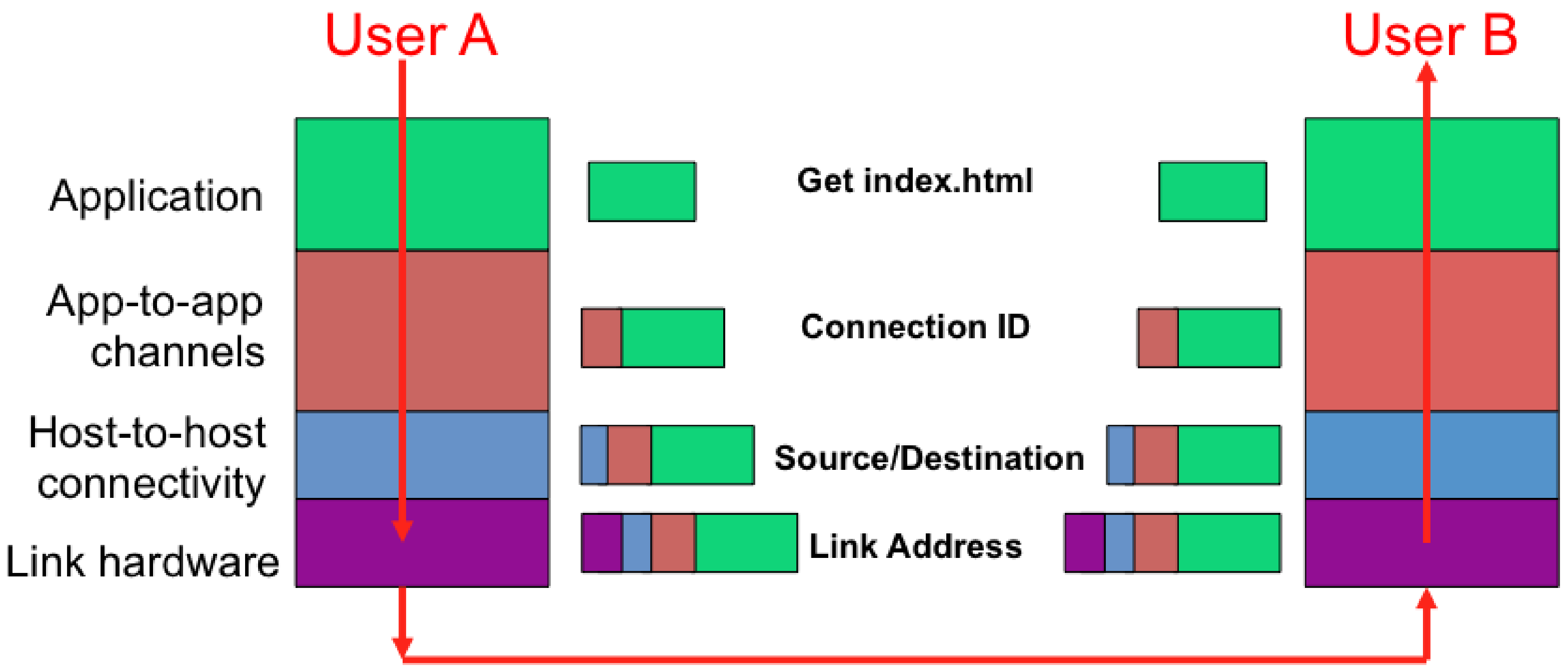
# Layer Encapsulation in HTTP






Source: Freedman

# Layer Encapsulation in HTTP




Source: Freedman

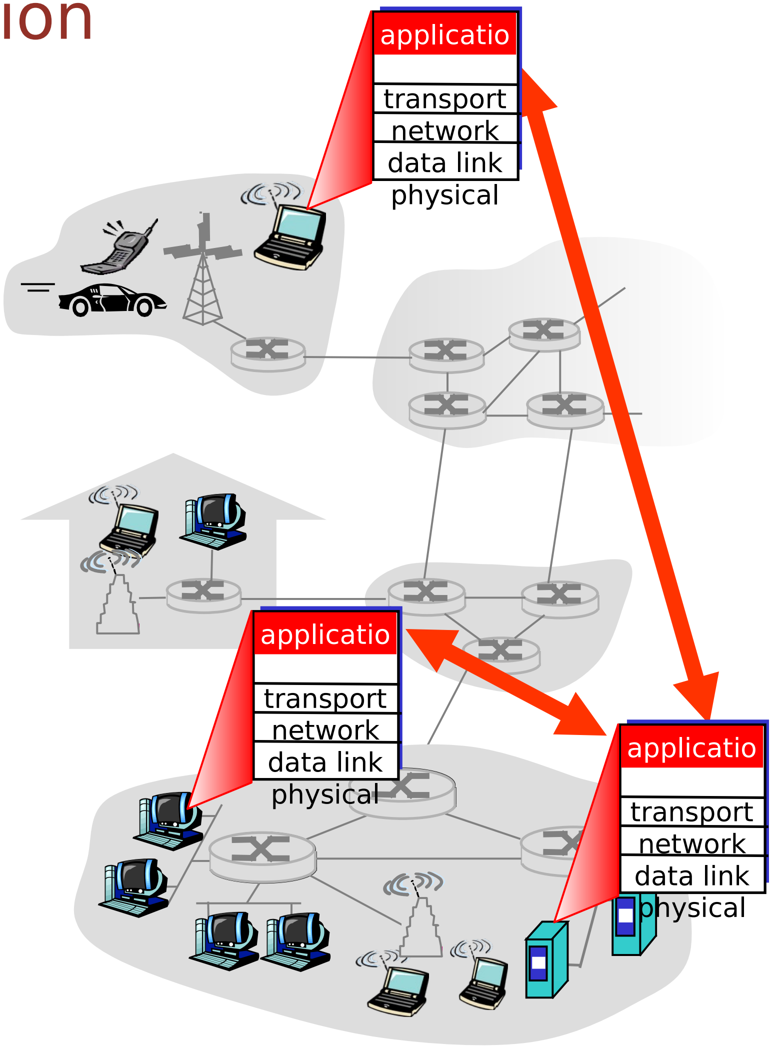
# Creating a network application

write programs that

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with 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




Source: Kurose & Ross

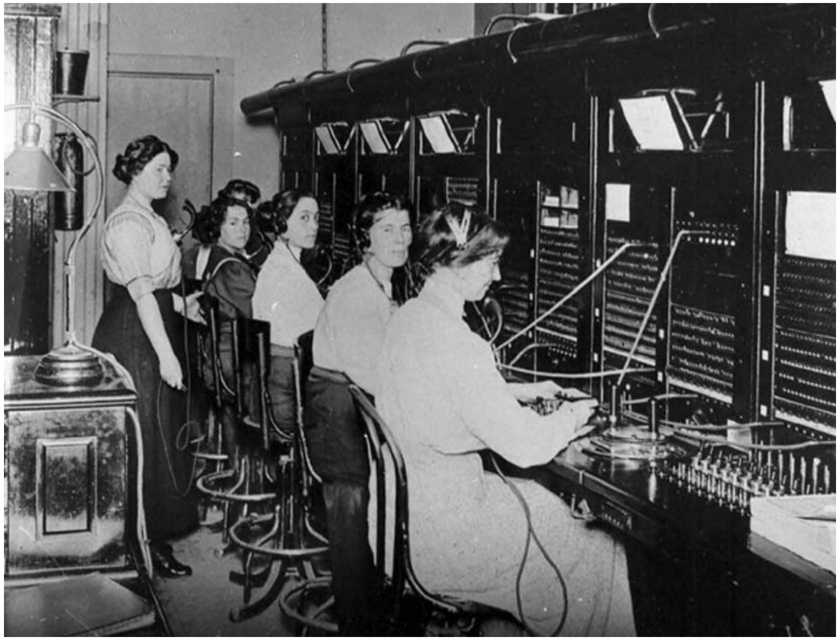
# Some network apps

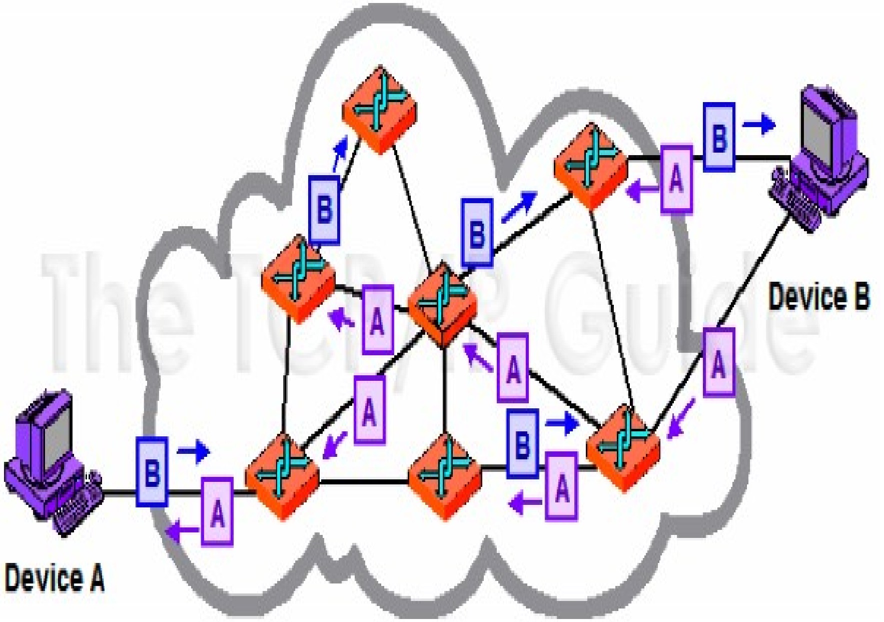
- e-mail
- remote login
- web
- instant messaging
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube)
- voice over IP
- real-time video conferencing
- social networking
- cloud computing
- …
- …
-

Source: Kurose & Ross

# Network applications need streams of data

Circuit switching → Packet switching






Today's networks provide packet delivery, not streams!

Source: Freedman (partial)

# What if the Data Doesn't Fit?

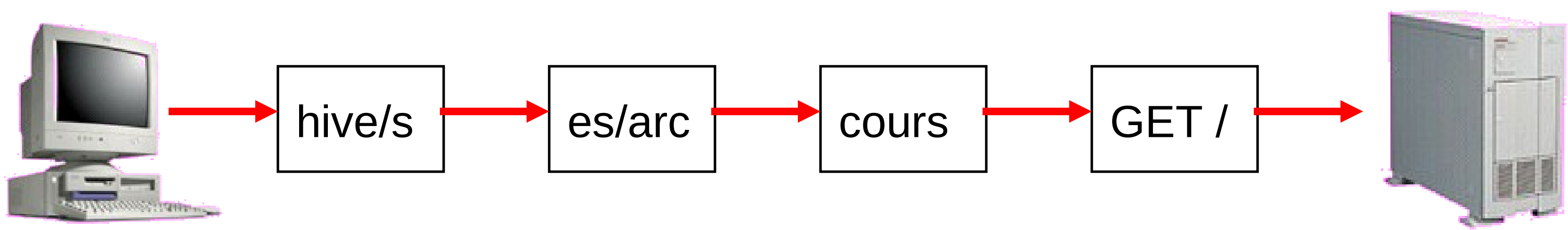
GET /courses/archive/spr09/cos461/ HTTP/1.1
Host: www.cs.princeton.edu
User-Agent: Mozilla/4.03
CRLF

Request

## Problem: Packet size

- Typical Web page is 10 kbytes
- On Ethernet, max IP packet is 1500 bytes

GET index.html

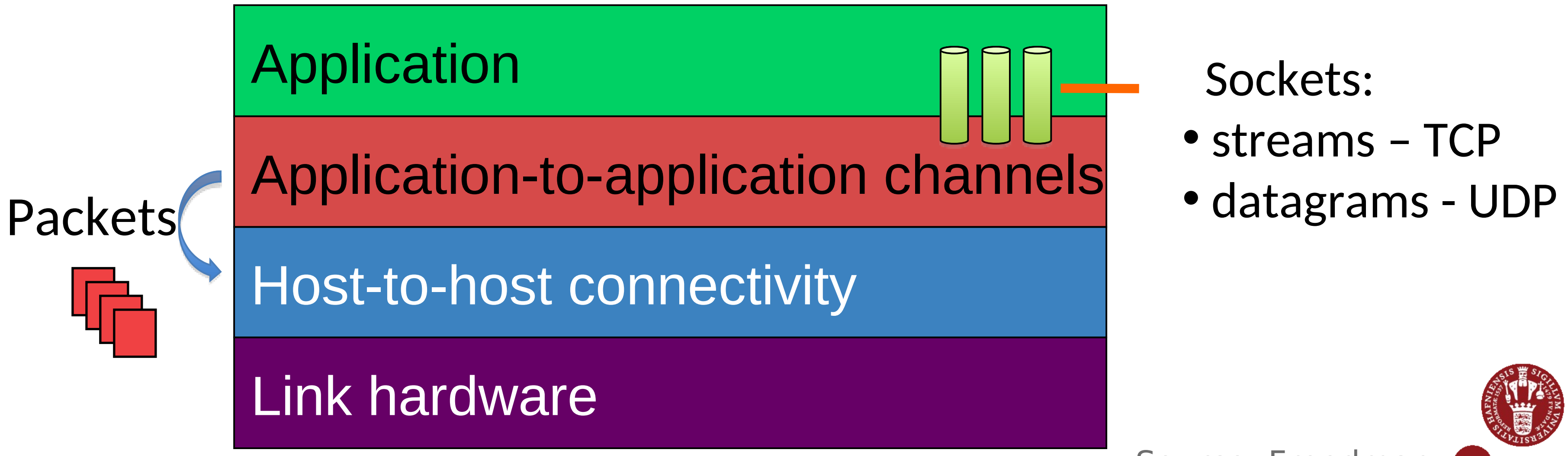



## Solution: Split the data across multiple packets

Source: Freedman

# Layering = Functional Abstraction

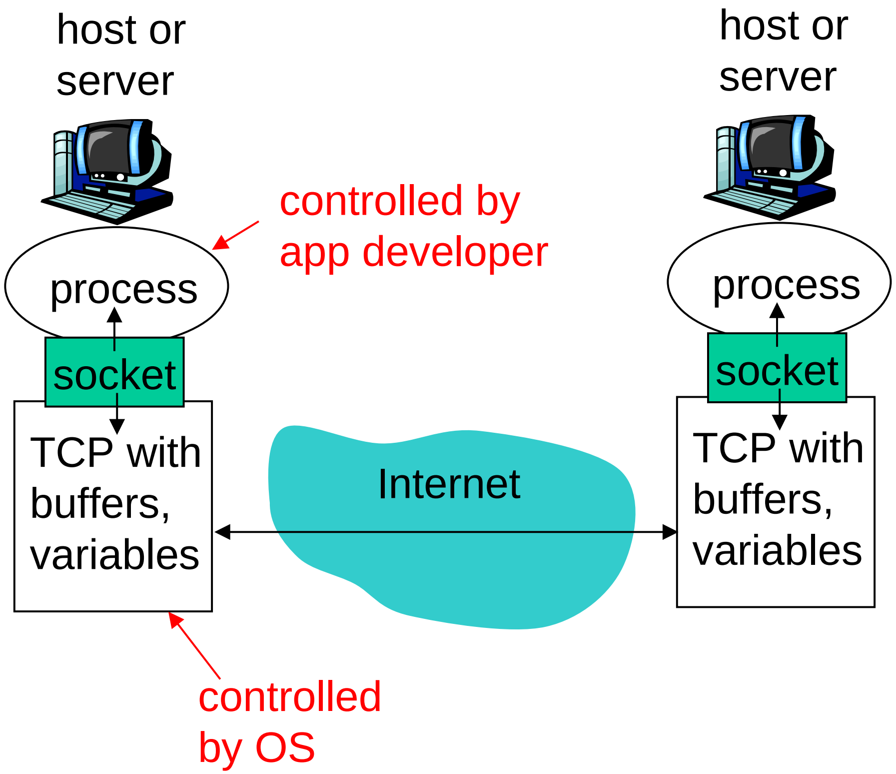
- Sub-divide the problem
  - Each layer relies on services from layer below
  - Each layer exports services to layer above
- Interface between layers defines interaction
  - Hides implementation details
  - Layers can change without disturbing other layers




Source: Freedman

# Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process




API: (1) choice of transport protocol; (2) ability to fix a few parameters (more on this in next lecture!)

Source: Kurose & Ross (partial)

# Internet transport protocols services

## TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport:* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantees, security

## UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Q: why bother? Why is there a UDP?

Source: Kurose & Ross

# UNIX Socket API

- Socket interface
  - Originally provided in Berkeley UNIX
  - Later adopted by all popular operating systems
  - Simplifies porting applications to different OSes
- In UNIX, everything is like a file
  - All input is like reading a file
  - All output is like writing a file
  - File is represented by an integer file descriptor
- API implemented as system calls
  - E.g., connect, read, write, close, …

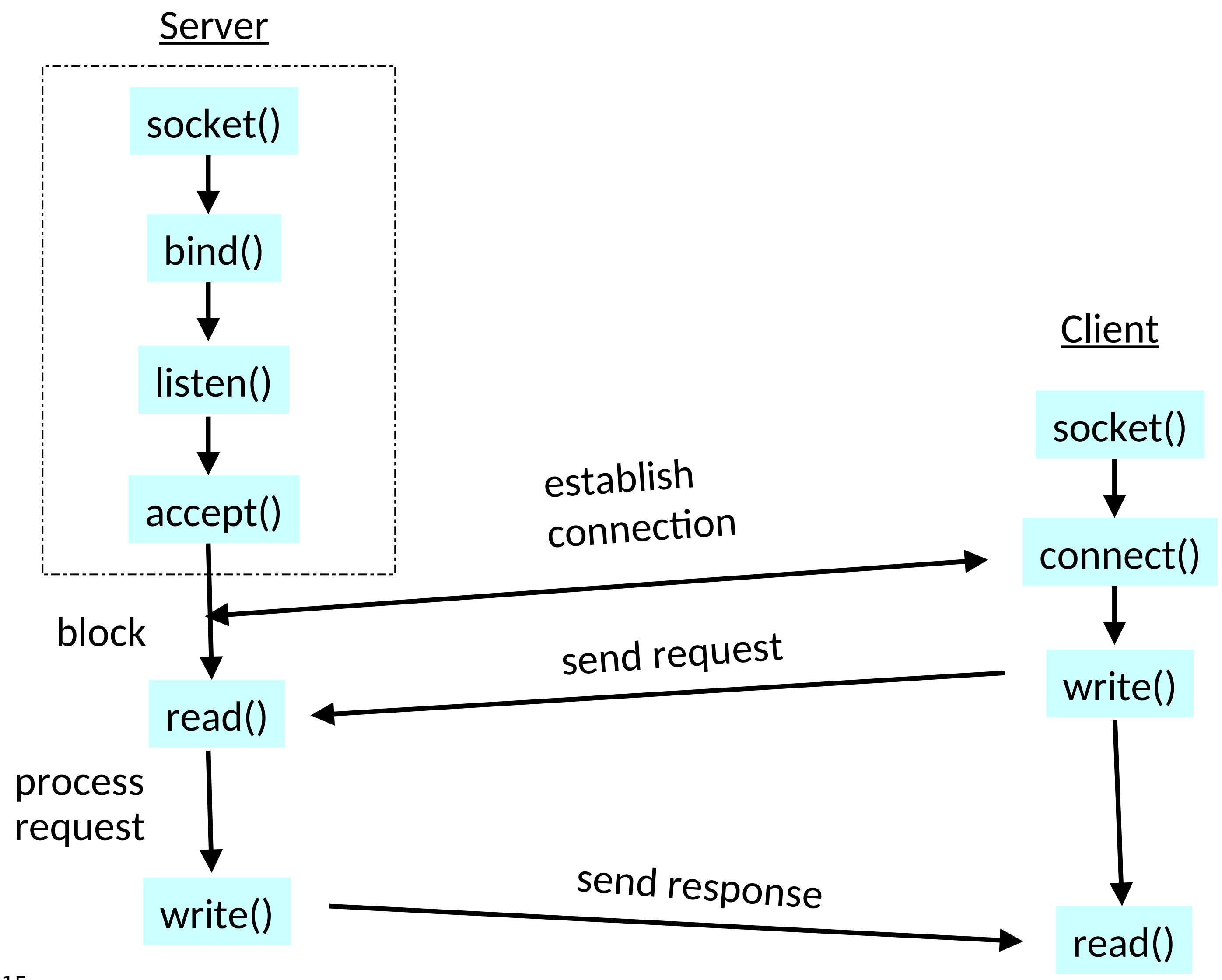
Source: Freedman

# Identifying the Receiving Process

- Sending process must identify the receiver
  - The receiving end host machine
  - The specific socket in a process on that machine
- Receiving host
  - Destination address that uniquely identifies the host
    - Typically, high-level name translated to IP address (DNS)
    - For example, www.diku.dk → 130.225.96.108
  - An IP address is a 32-bit quantity
- Receiving socket
  - Host may be running many different processes
  - Destination port that uniquely identifies the socket
  - A port number is a 16-bit quantity

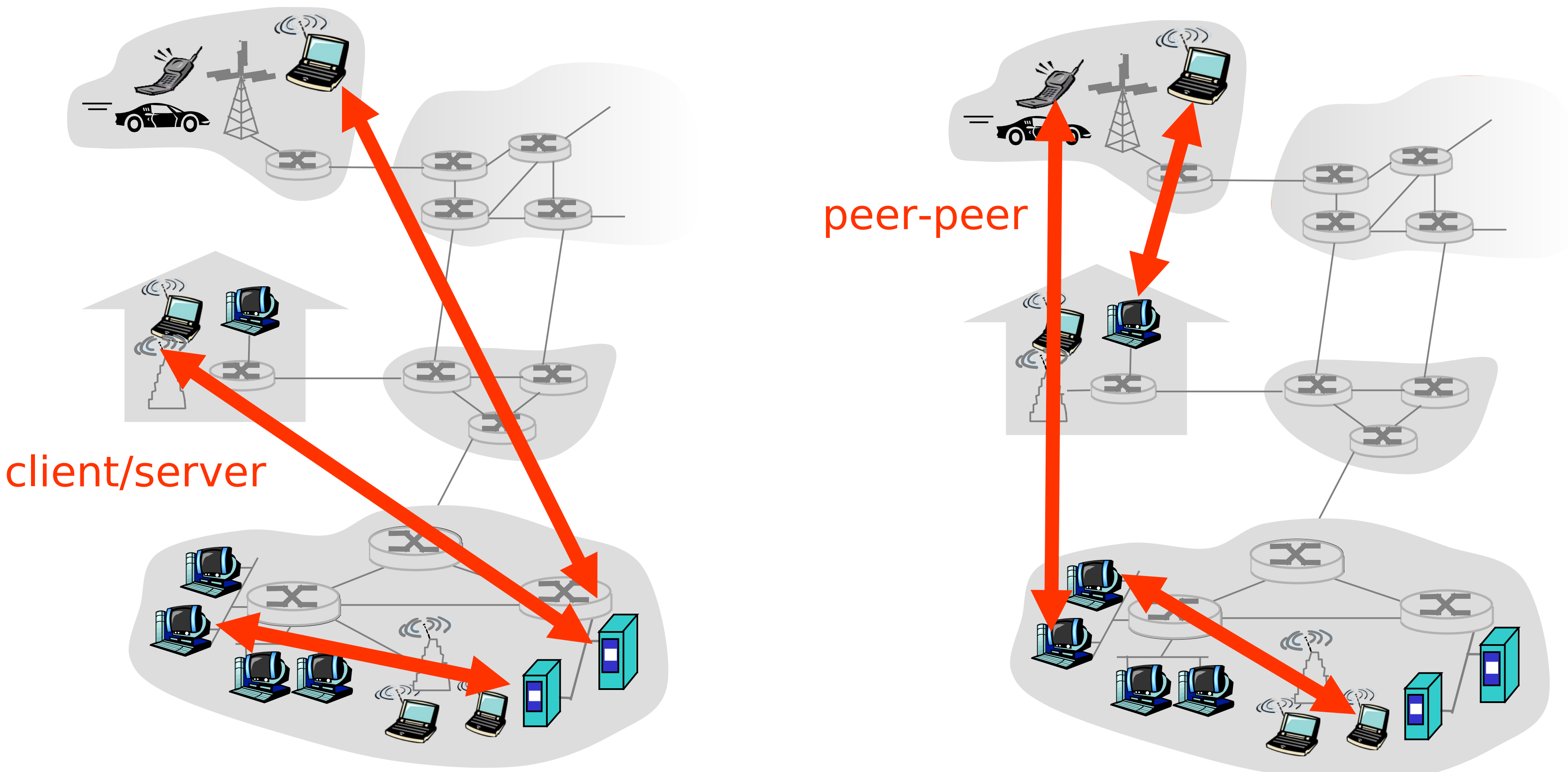
Source: Freedman (partial)

# Client-Server TCP Sockets




Source: Freedman (partial)

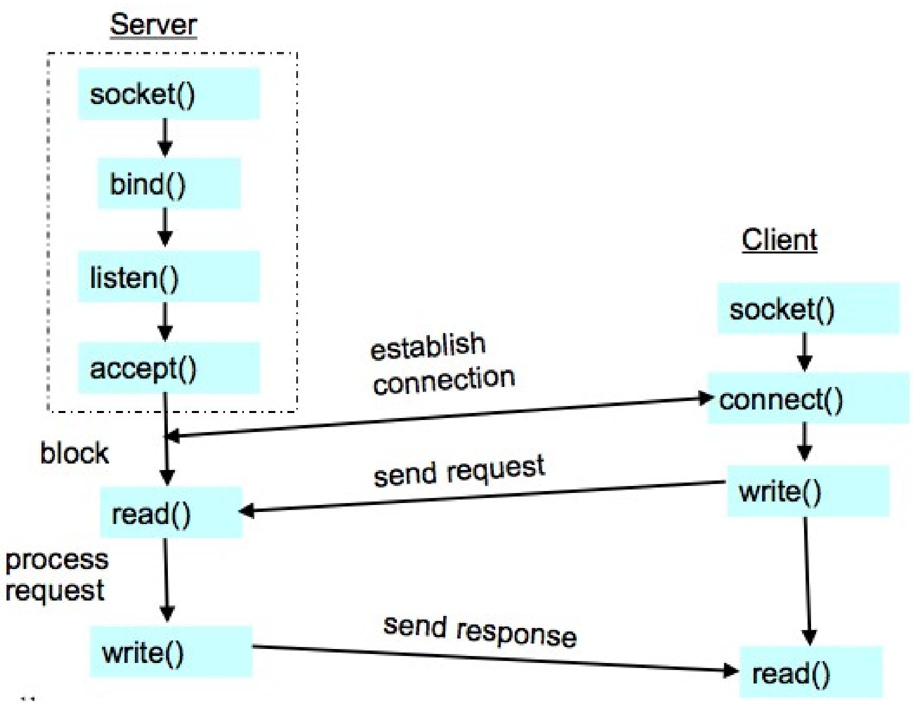
# Application Architectures

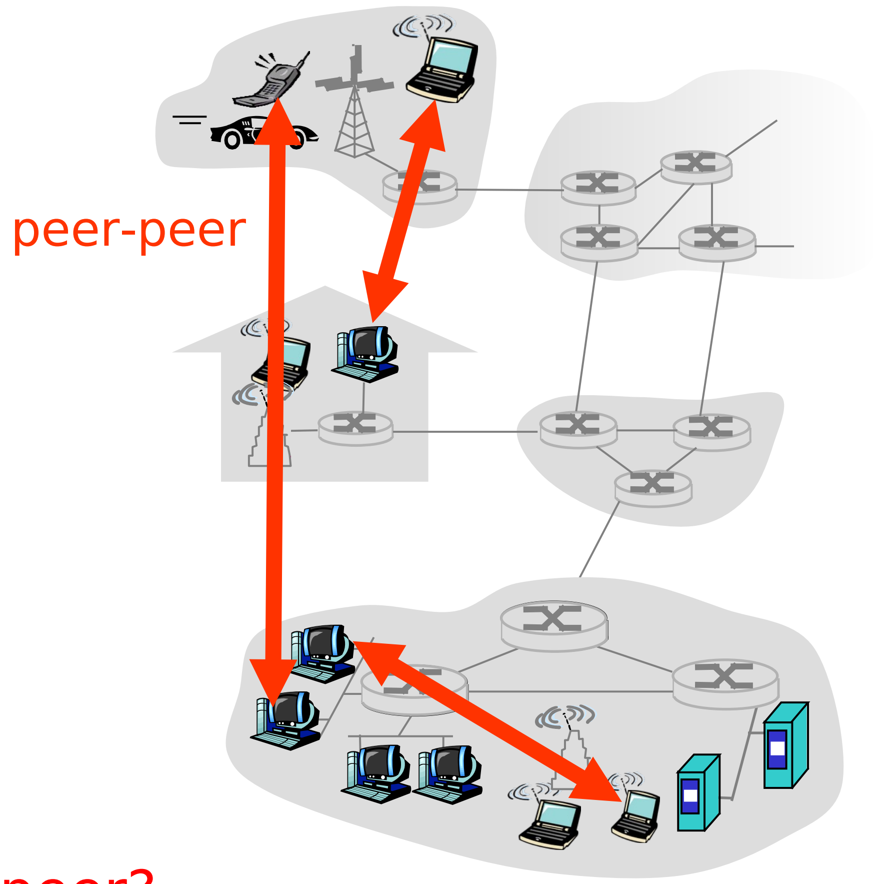



- P2P highly scalable, but difficult to manage
- Hybrids also possible, e.g., Skype

Source: Kurose & Ross (partial)

# Discussion: How do you set sockets up in a P2P application?




- How many sockets in each peer?
- What if many peers connect to one peer?

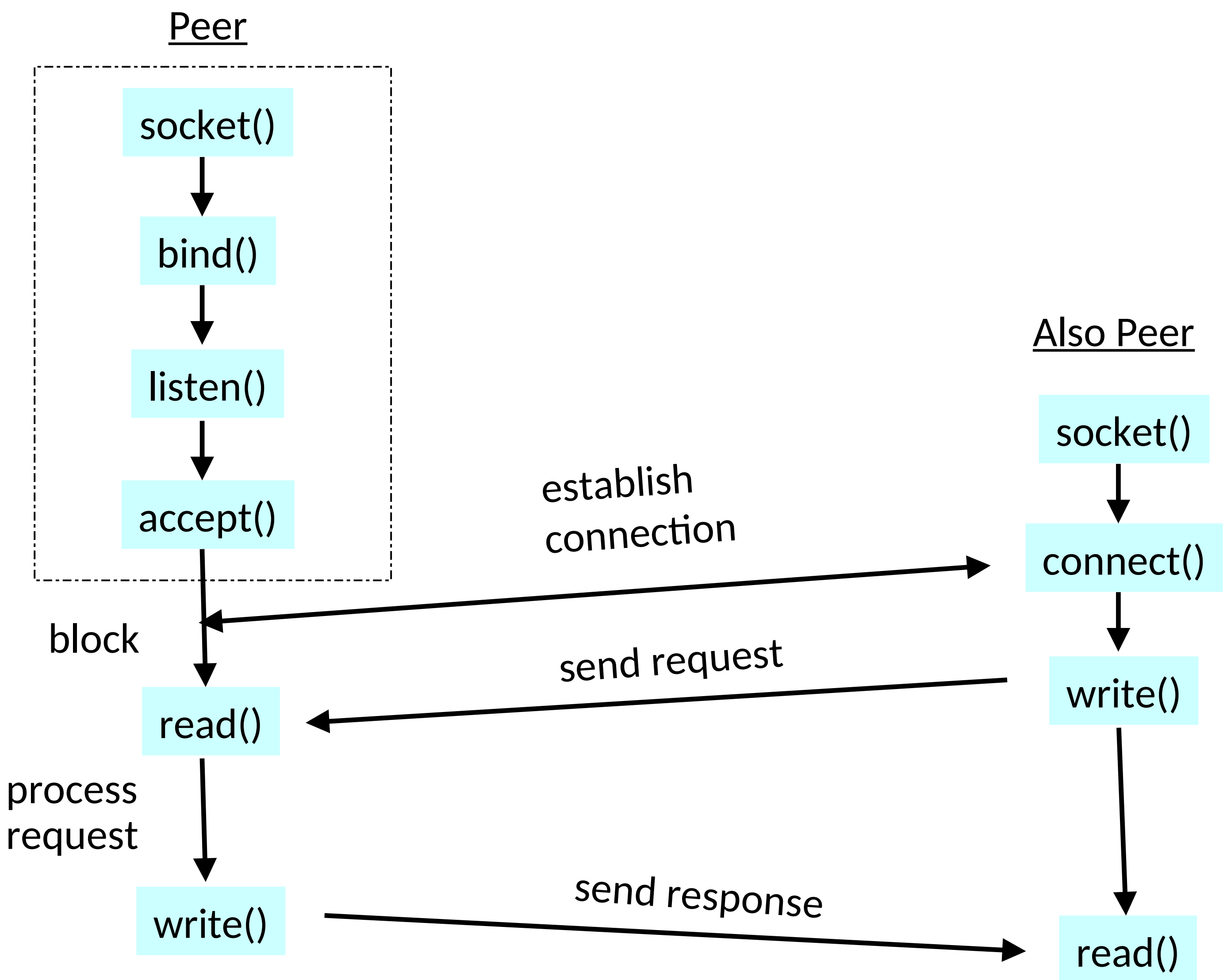
# P2P Sockets

- *The proof is trivial and left as an exercise to the reader.*

  Carl-Johannes Johnsen

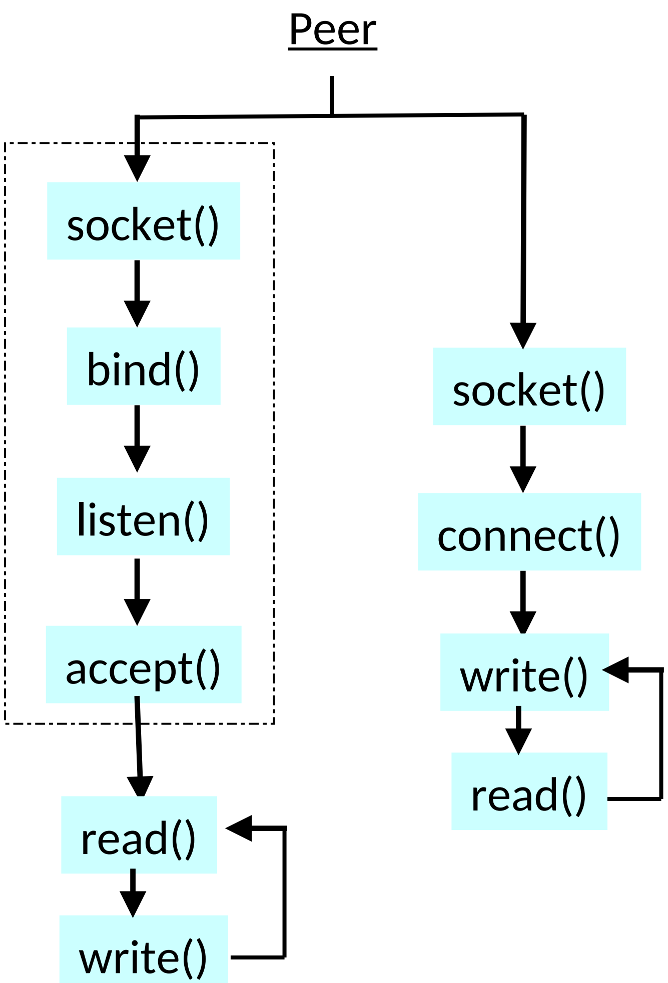
Source: Freedman (partial)

# P2P Sockets

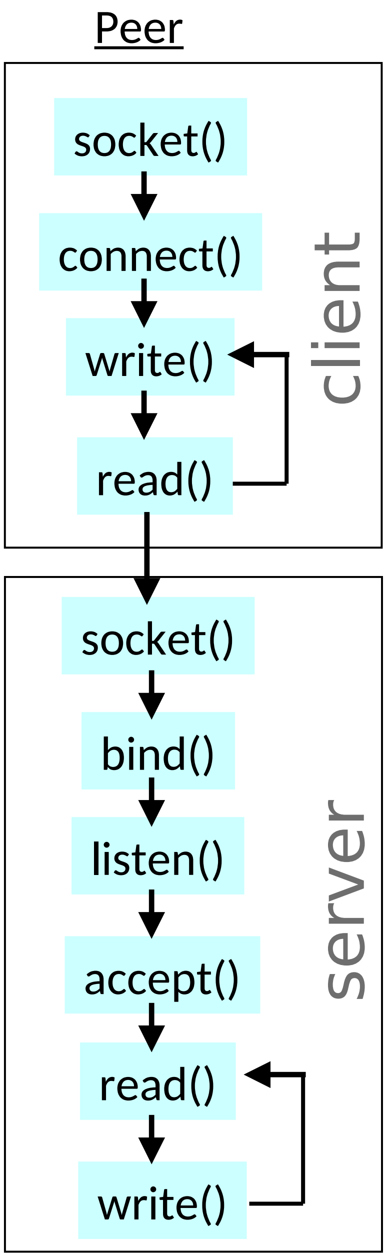



Source: Freedman (partial)

# P2P Sockets




OR




Source: Freedman (partial)

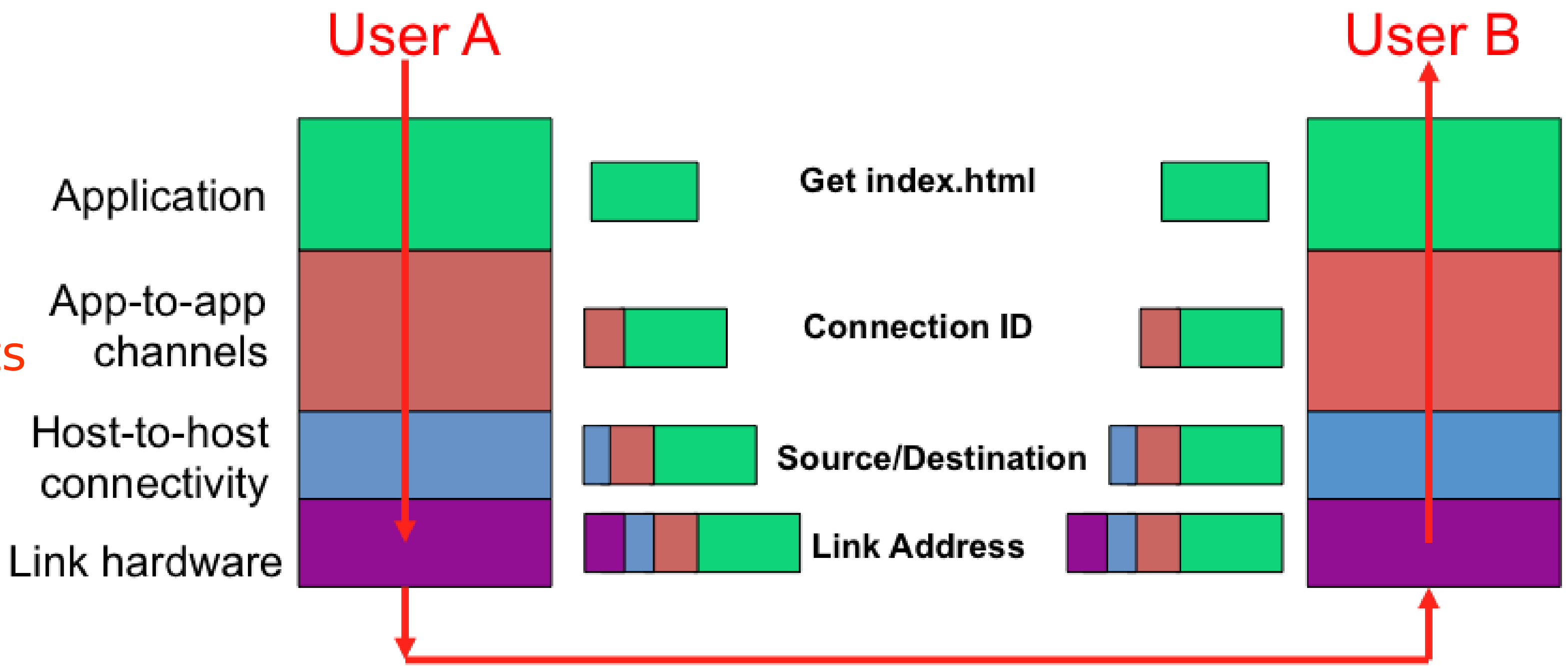
# HTTP Basics

- HTTP layered over bidirectional byte stream
- Interaction
  - Client sends request to server, followed by response from server to client
  - Requests/responses are encoded in text
- Targets access to web objects
  - GET, POST, HEAD → HTTP/1.0
  - GET, POST, HEAD, PUT, DELETE → HTTP/1.1
- Stateless
  - Server maintains no info about past client requests
    - What about personalization? Data stored in back-end database; client sends " web cookie" used to lookup data

Source: Freedman (partial)

# Layer Encapsulation in HTTP

TCP sockets




Source: Freedman (partial)

# HTTP Request Example

GET / HTTP/1.1

Host: sns.cs.princeton.edu

Accept: */*

Accept-Language: en-us

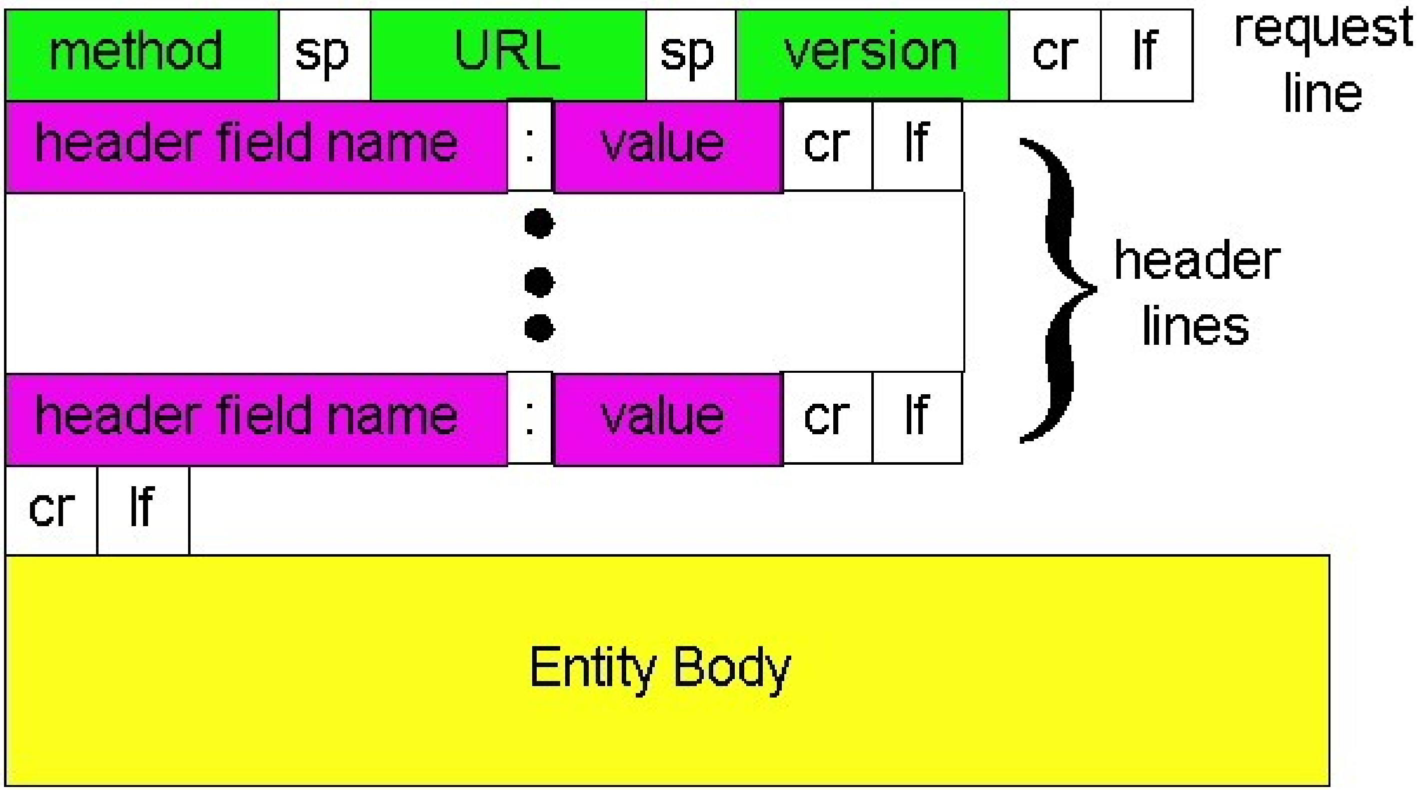
Accept-Encoding: gzip, deflate

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13

Connection: Keep-Alive

Source: Freedman

# HTTP Request




Source: Freedman

# HTTP Request

GET / HTTP/1.1

Host: sns.cs.princeton.edu

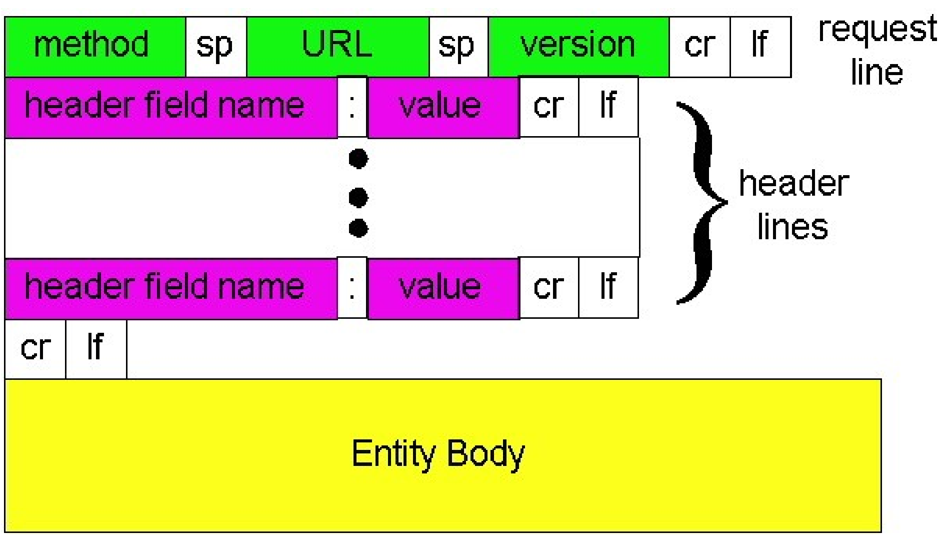
Accept: */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13

Connection: Keep-Alive




Source: Freedman

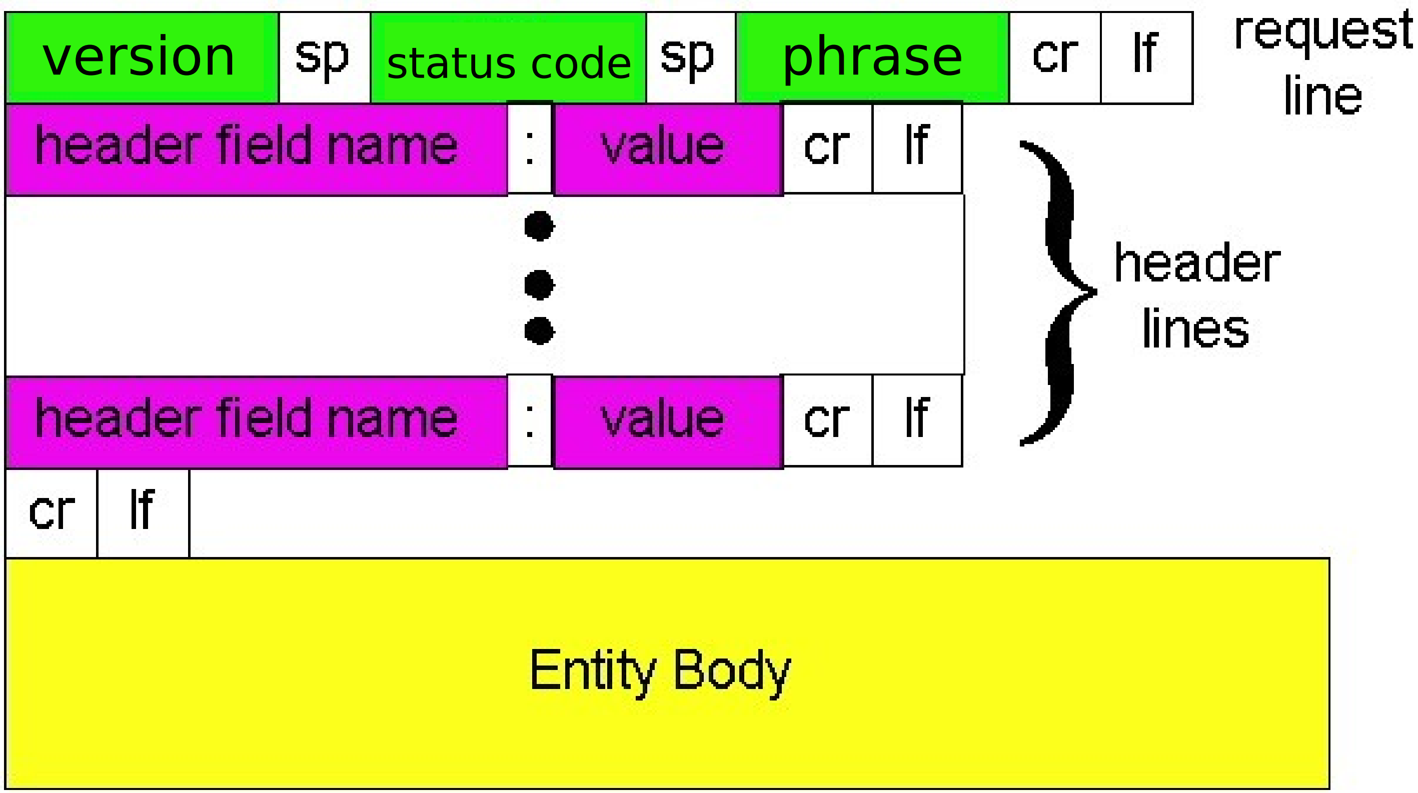
# HTTP Response Example

```
HTTP/1.1 200 OK
Date: Wed, 02 Feb 2011 04:01:21 GMT
Server: Apache/2.2.3 (CentOS)
X-Pingback: http://sns.cs.princeton.edu/xmlrpc.php
Last-Modified: Wed, 01 Feb 2011 12:41:51 GMT
ETag: "7a11f-10ed-3a75ae4a"
Accept-Ranges: bytes
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="en-US">
```

Source: Freedman

# HTTP Response




Source: Freedman

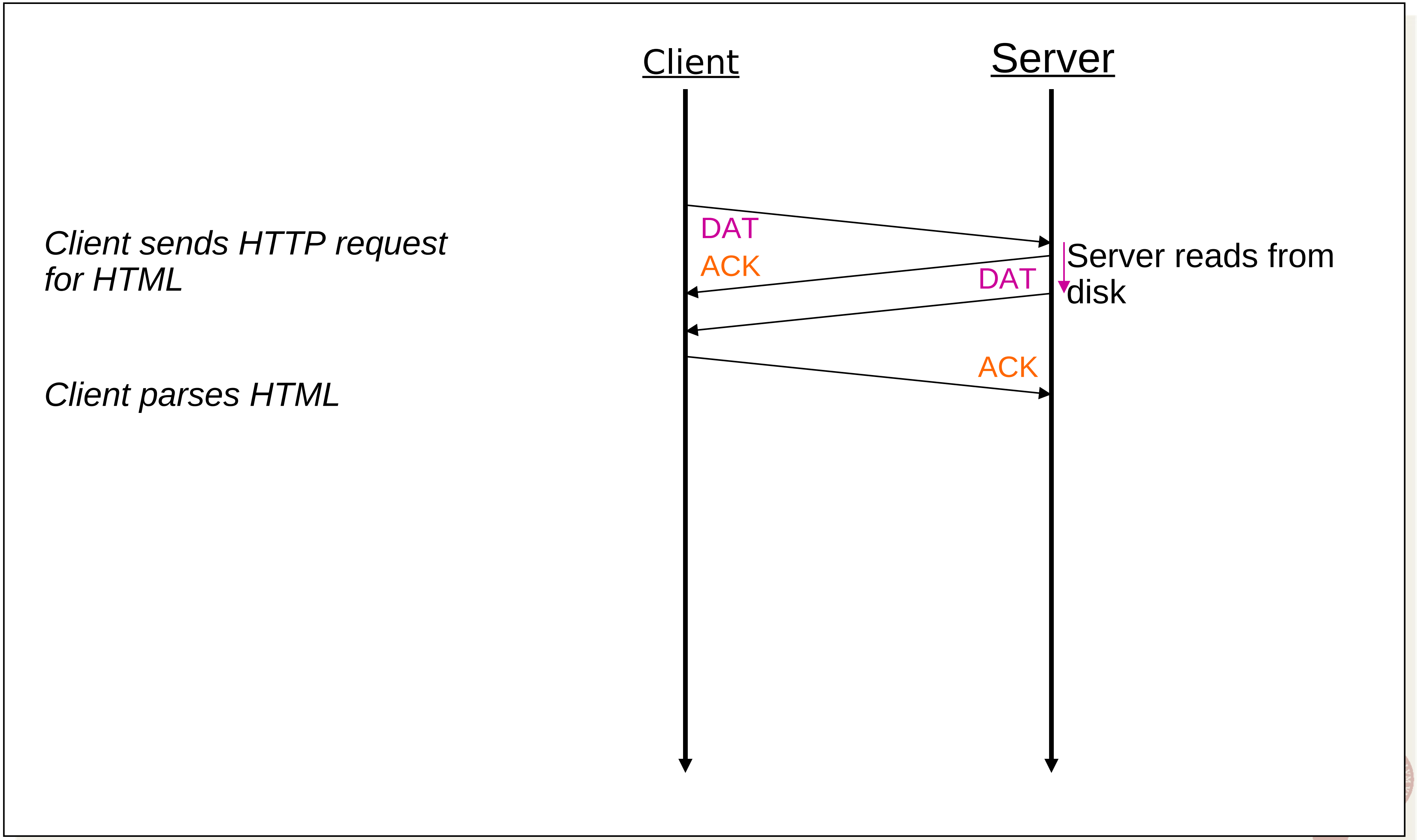
# Some HTTP Response codes

| | |
|---|---|
| 200 | OK |
| 204 | No Content |
| 301 | Moved Permanently |
| 400 | Bad Request |
| 403 | Forbidden |
| 404 | Not Found |
| 418 | I’m a teapot |
| 500 | Internal Server Error |

Source: Freedman

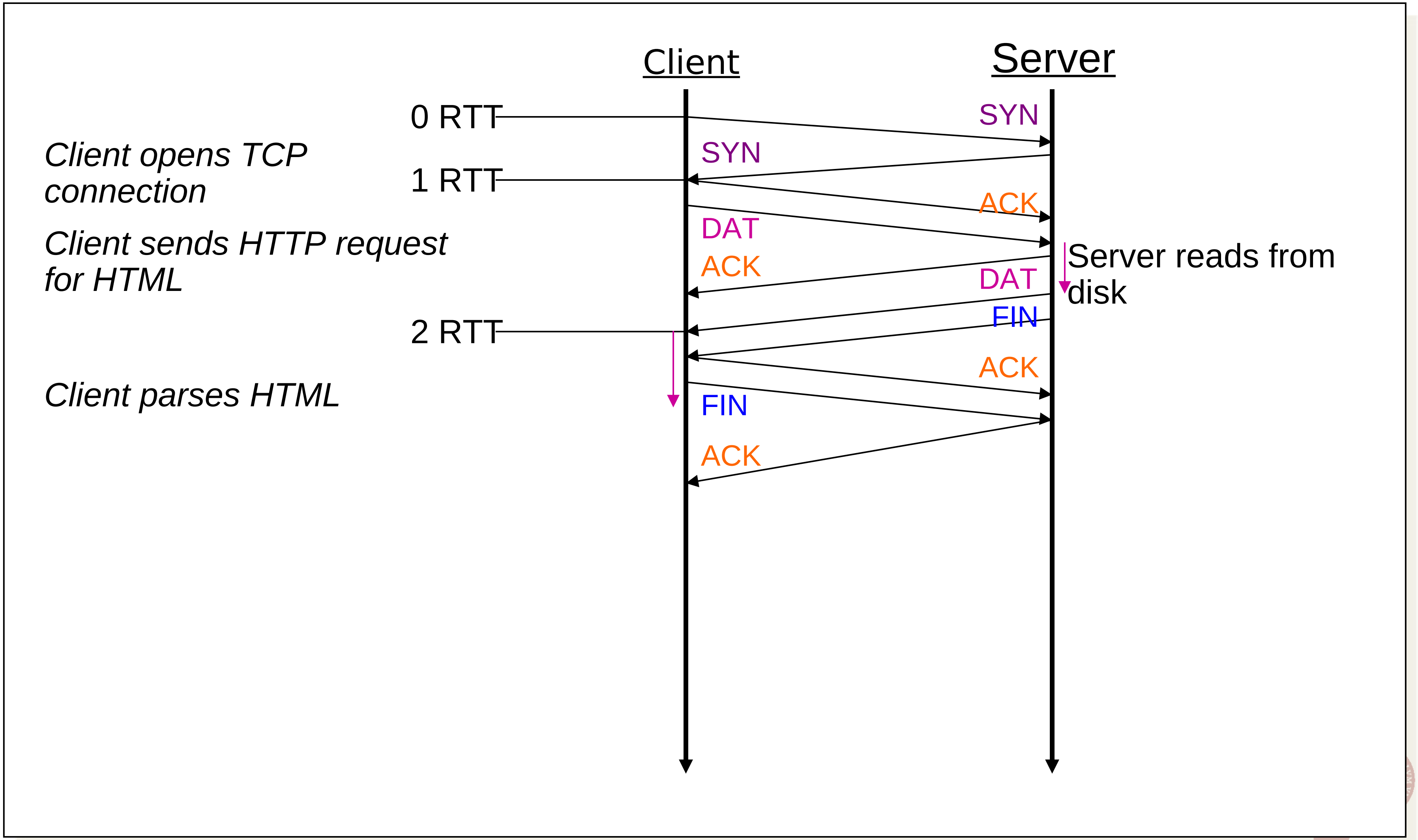
# Single Transfer Example

Source: Freedman



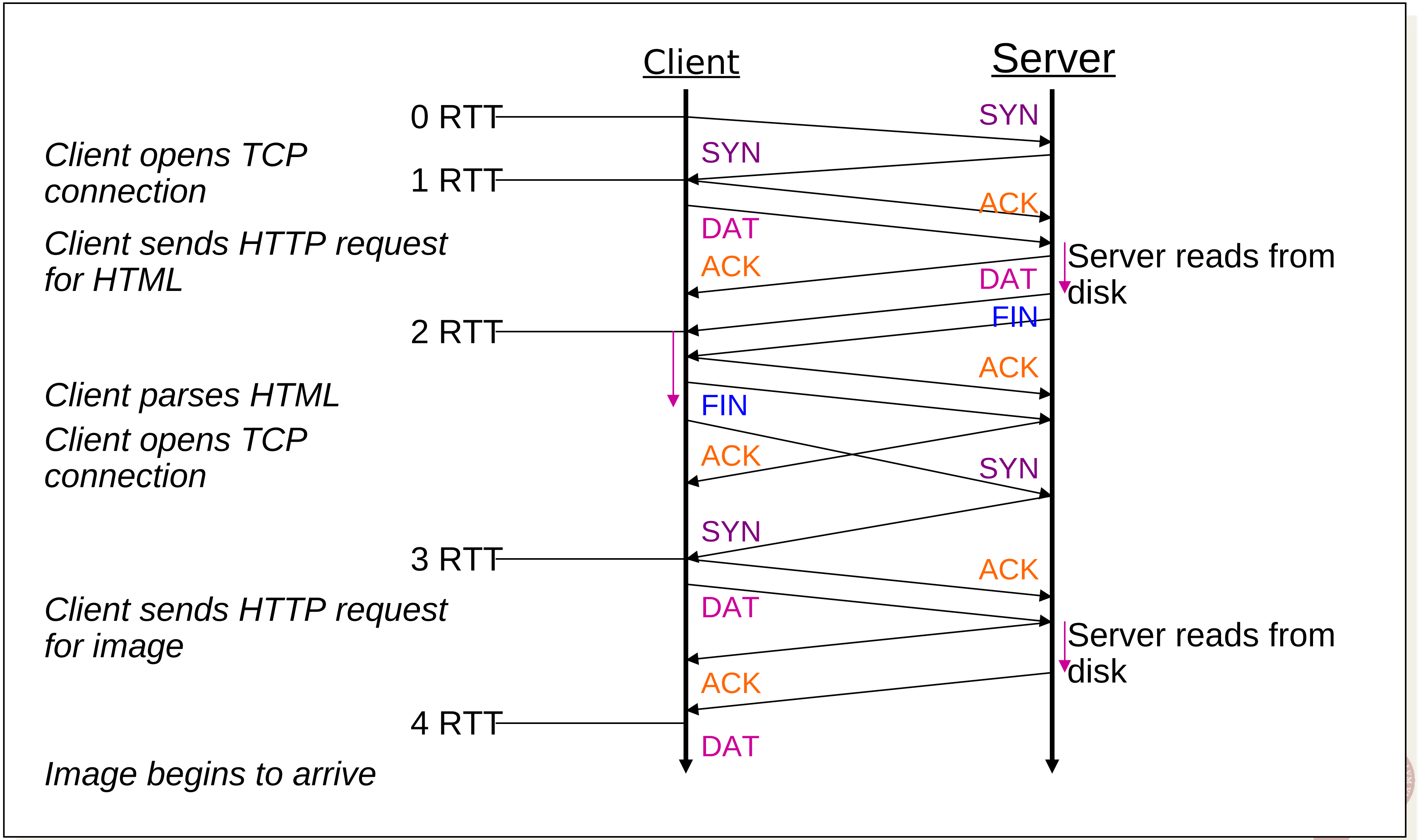
# Single Transfer Example

Source: Freedman



# Single Transfer Example

Source: Freedman



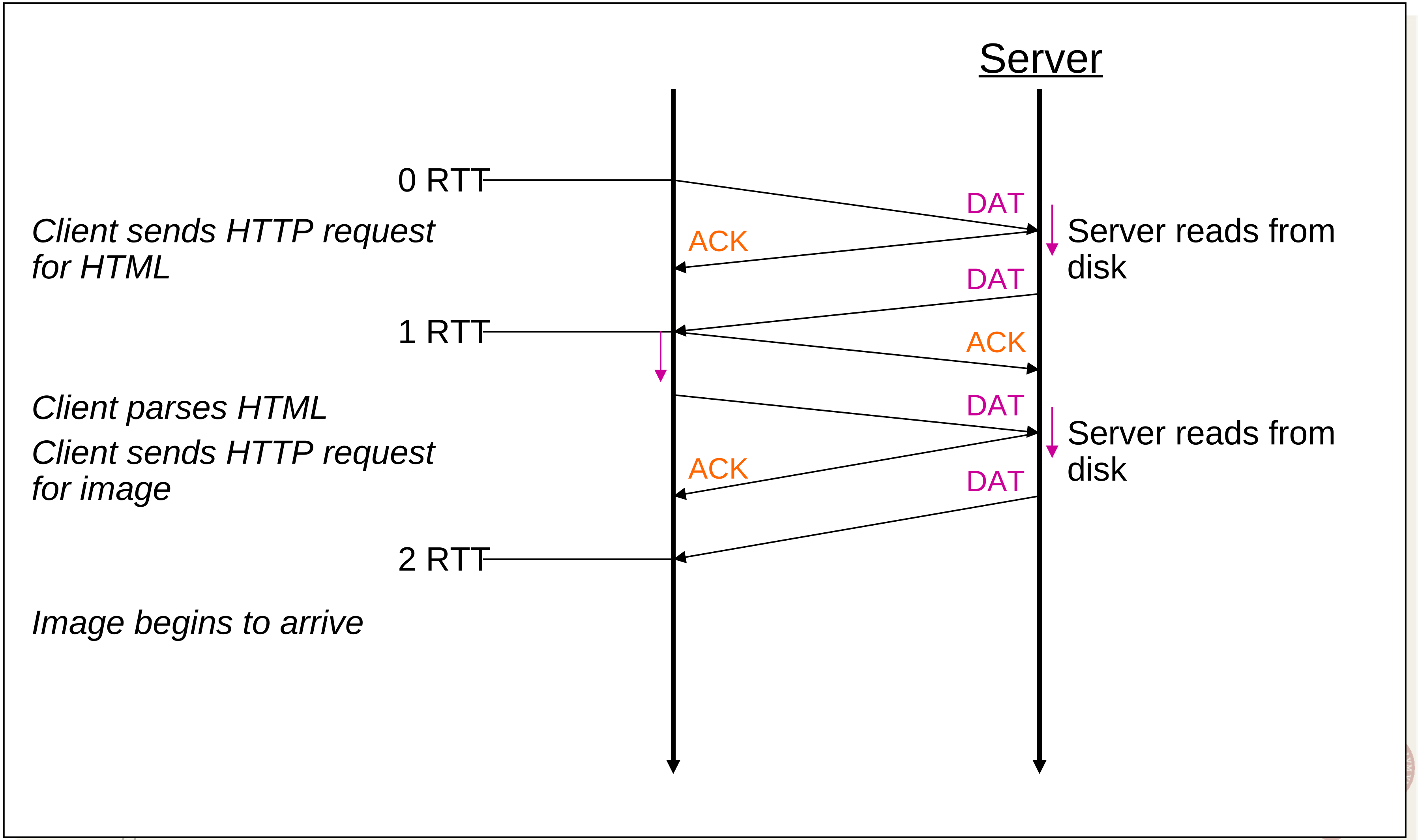
# Problems with simple model

- Multiple connection setups
  - Three-way handshake each time (TCP " synchronizing" stream)
- Lots of extra connections
  - Increases server state/processing
  - Server forced to keep connection state
- Later we will see also that
  - Short transfers are hard on stream protocol (TCP)
    - How much data should it send at once?
    - Congestion avoidance: Takes a while to " ramp up" to high sending rate (TCP " slow start" )
    - Loss recovery is poor when not " ramped up"

Source: Freedman (partial)

# Persistent Connection Example

Source: Freedman



# Persistent HTTP

Non-persistent HTTP issues:

- Requires 2 RTTs per object
- OS must allocate resources for each TCP connection
- But 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 are sent over connection

Source: Freedman

# Persistent HTTP

**Persistent without pipelining:**

- Client issues new request only when previous response has been received
- One RTT for each object

**Persistent with pipelining:**

- Default in HTTP/1.1 spec
- Client sends requests as soon as it encounters referenced object
- As little as one RTT for all the referenced objects
- Server must handle responses in same order as requests

- Persistent without pipelining most common: When does pipelining work best?
- Multiple parallel requests or pipelined requests?

Source: Freedman (partial)

# HTTP Caching

- Clients often cache documents
  - When should origin be checked for changes?
  - Every time? Every session? Date?

- HTTP includes caching information in headers
  - HTTP 0.9/1.0 used: "Expires: <date>"; "Pragma: no-cache"
  - HTTP/1.1 has "Cache-Control"
    - "No-Cache", "Private", "Max-age: <seconds>"
    - "E-tag: <opaque value>"

- If not expired, use cached copy
- If expired, use condition GET request to origin
  - "If-Modified-Since: <date>", "If-None-Match: <etag>"
  - 304 ("Not Modified") or 200 ("OK") response

Source: Freedman

# HTTP Conditional Request

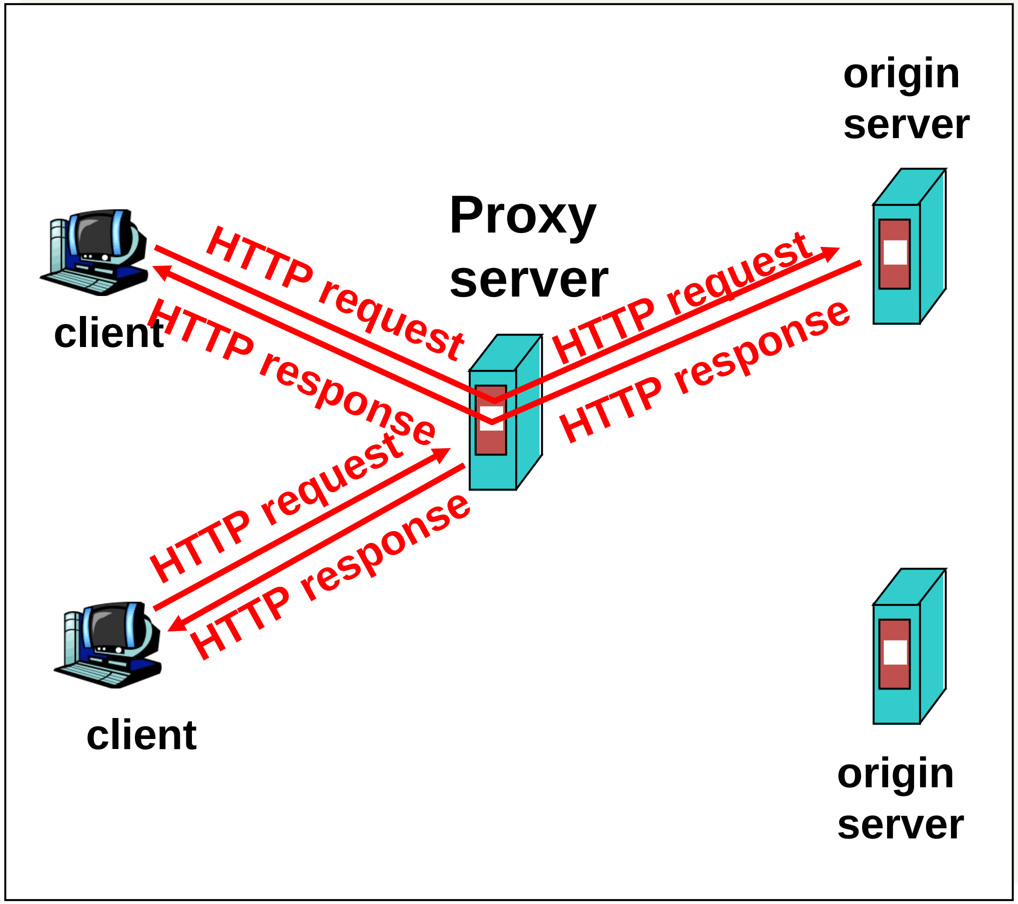
GET / HTTP/1.1
Host: sns.cs.princeton.edu
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.13)
Connection: Keep-Alive
If-Modified-Since: Tue, 1 Feb 2011 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"

HTTP/1.1 304 Not Modified
Date: Wed, 02 Feb 2011 04:01:21 GMT
Server: Apache/2.2.3 (CentOS)
ETag: "7a11f-10ed-3a75ae4a"
Accept-Ranges: bytes
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive

Source: Freedman

# Web Proxy Caches

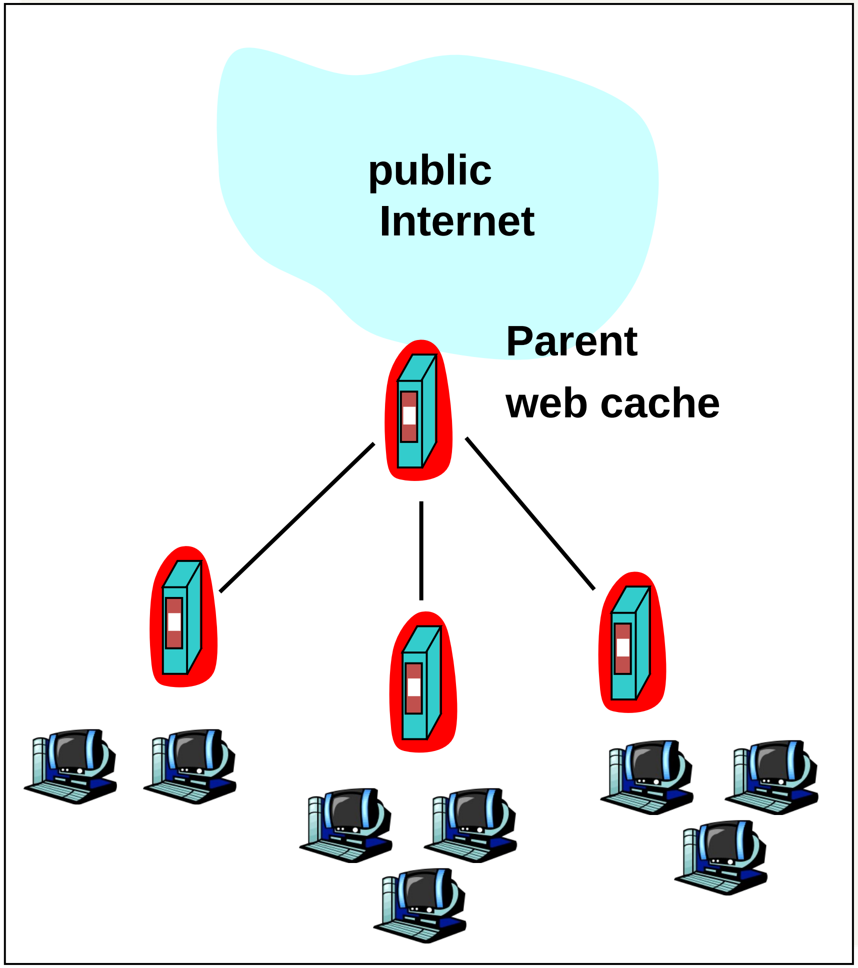
- User configures browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - Object in cache: cache returns object
  - Else: cache requests object from origin, then returns to client




Source: Freedman

# When a single cache isn't enough

- What if the working set is > proxy disk?
  - Cooperation!
- A static hierarchy
  - Check local
  - If miss, check siblings
  - If miss, fetch through parent
- Internet Cache Protocol (ICP)
  - ICPv2 in RFC 2186 (& 2187)
  - UDP-based, short timeout




Source: Freedman

# Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client throughput
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current throughput
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

Source: Kurose & Ross

# Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

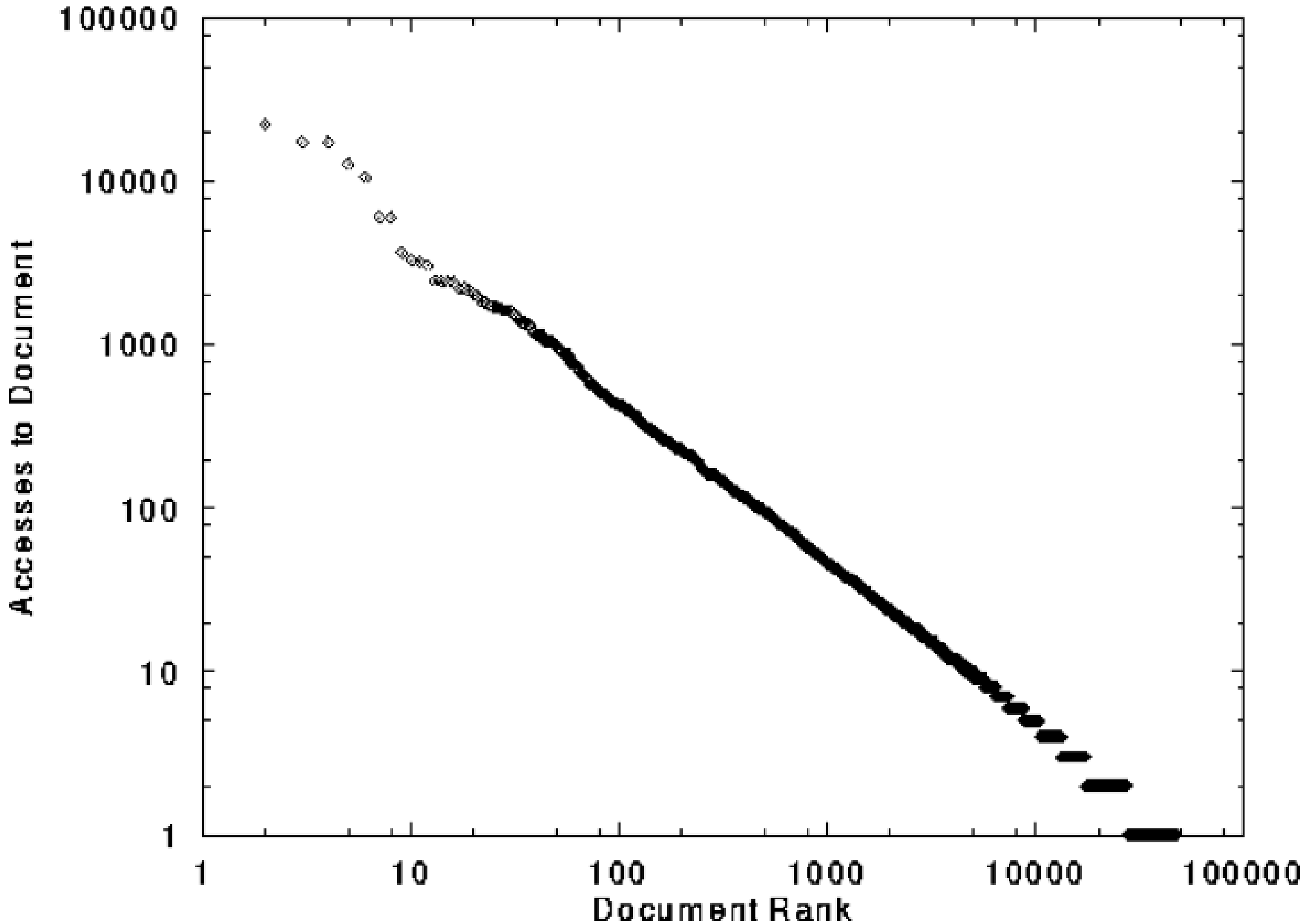
Source: Kurose & Ross

# Distributing Data to Clients

- Data needs to be close to clients
- Sending repeated data needs to be minimised
- Points of failure need to be minimised

Source: Kurose & Ross

# Web traffic has cacheable workload




" Zipf" or " power-law" distribution

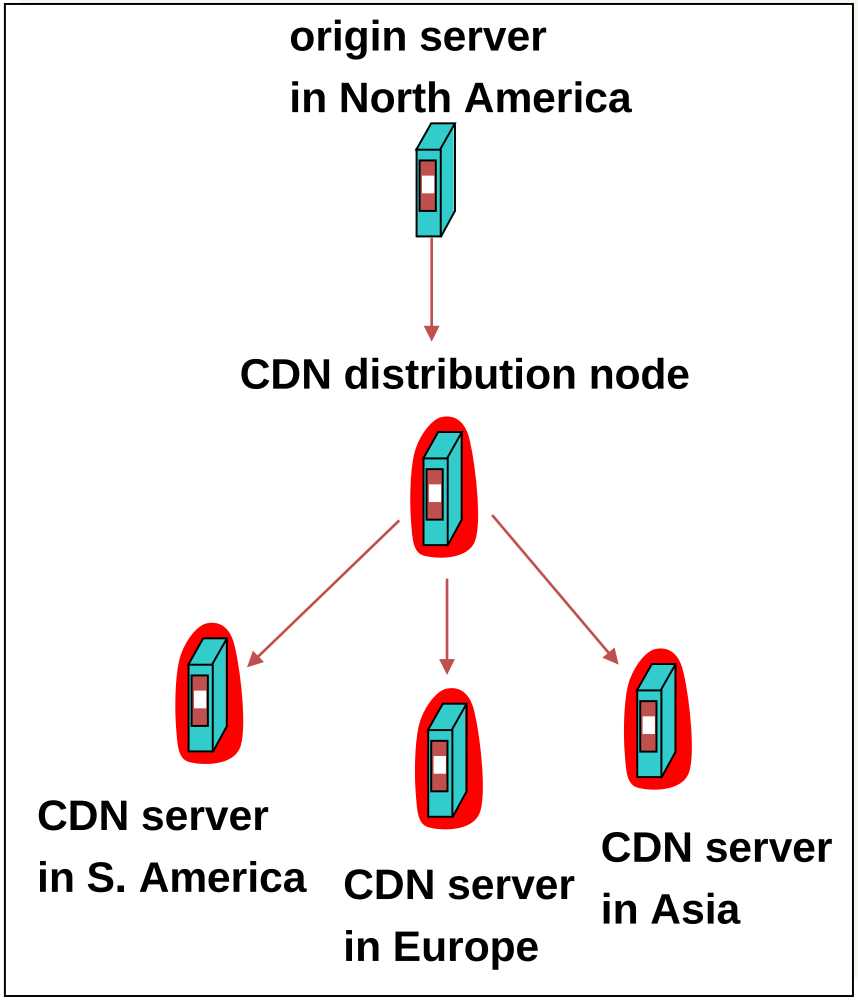
Source: Freedman

# Content Distribution Networks (CDNs)

- Content providers are CDN customers

## Content replication

- CDN company installs thousands of servers throughout Internet
  - In large datacenters (Bring Home)
  - Or, close to users (Enter Deep)
- CDN replicates customers' content
- When provider updates content, CDN updates servers




Source: Freedman

# Content Distribution Networks & Server Selection

- Replicate content on many servers

- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among known replicas
  - How to direct clients towards replica

Source: Freedman

# Server Selection

- Which server?
  - Lowest load: to balance load on servers
  - Best performance: to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node: to provide fault tolerance

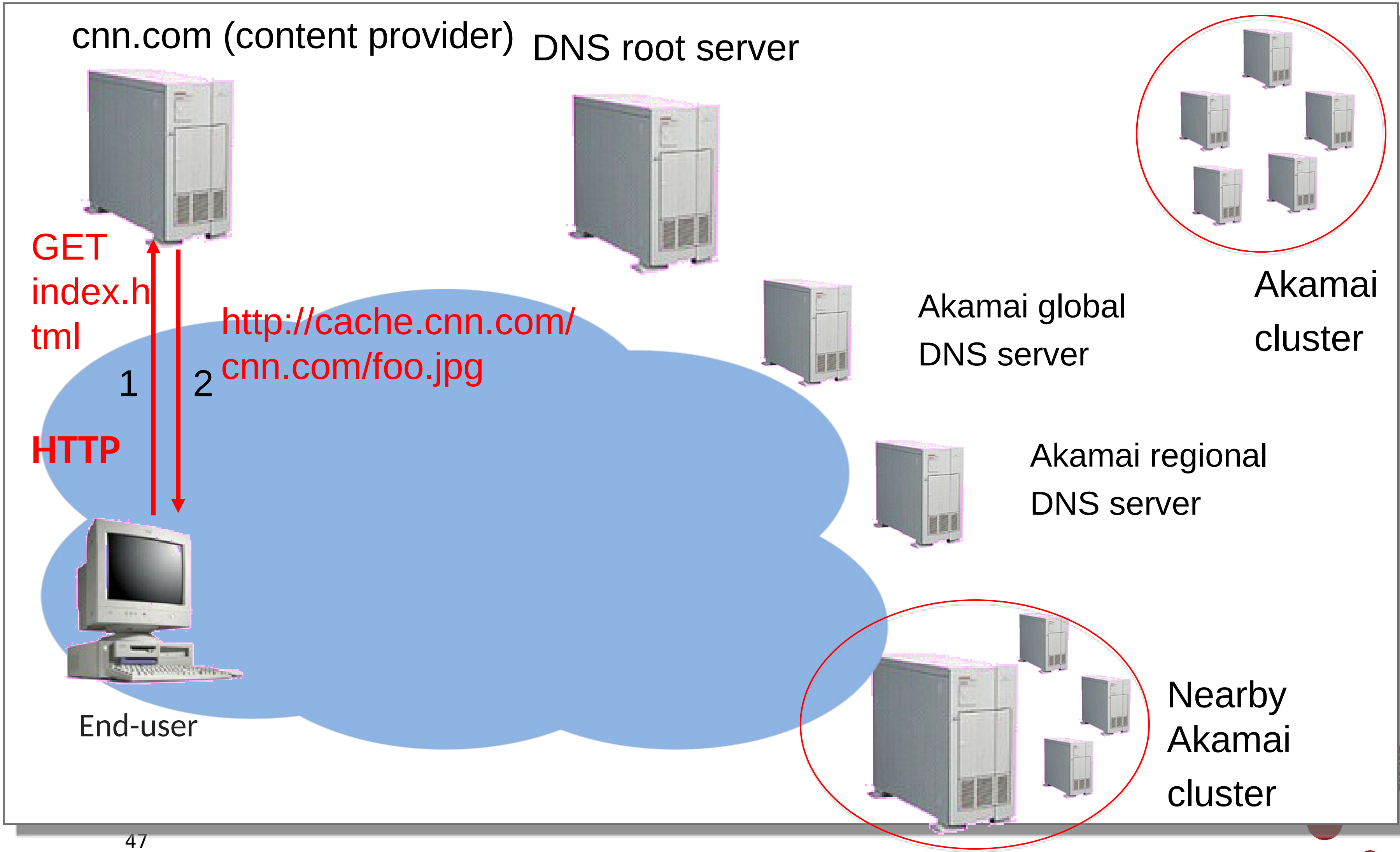
- How to direct clients to a particular server?
  - As part of routing: anycast, cluster load balancing
  - As part of application: HTTP redirect
  - As part of naming: DNS

  - We will explain some of these techniques better later in the course!

Source: Freedman (partial)

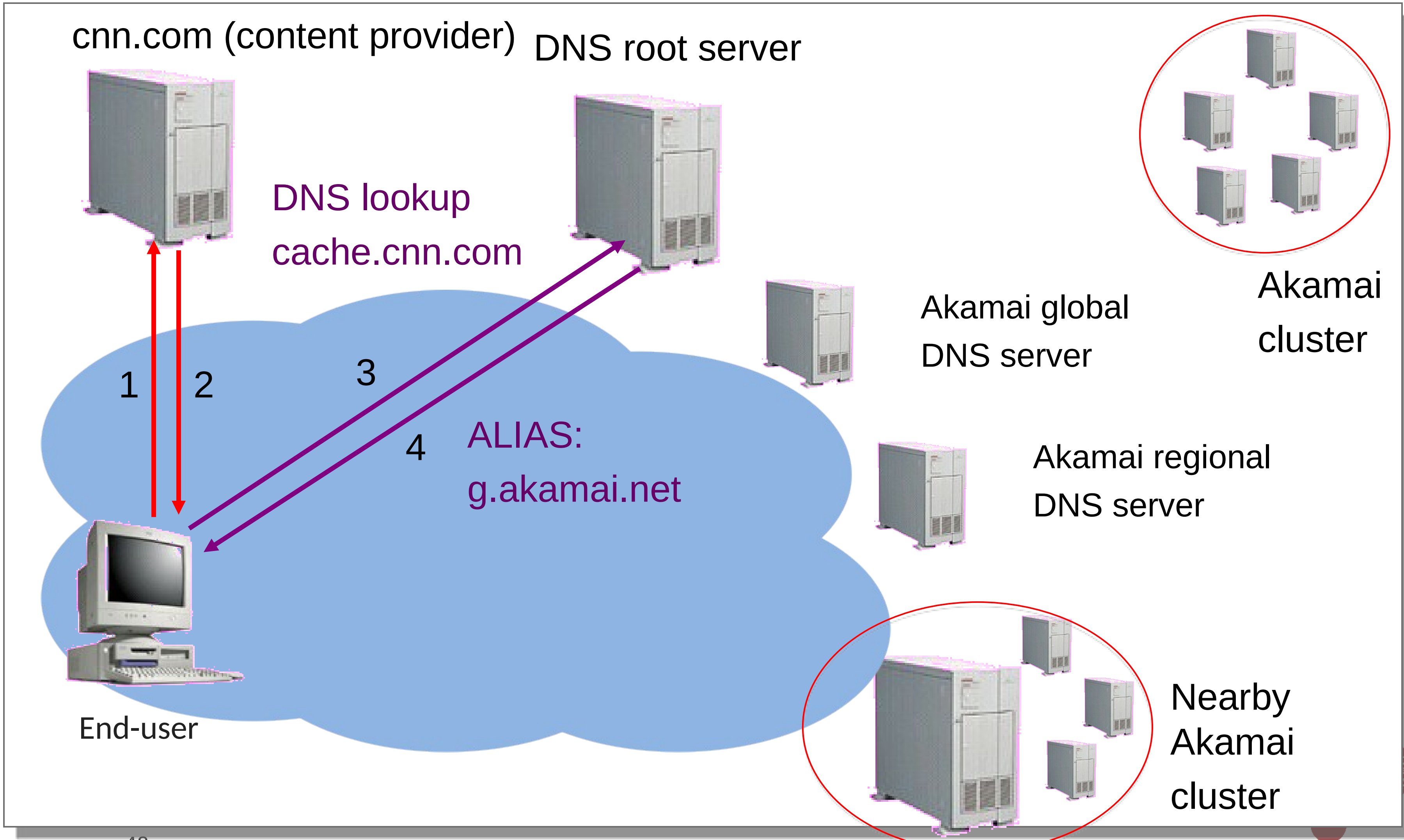
# How Akamai Works

Source: Freedman



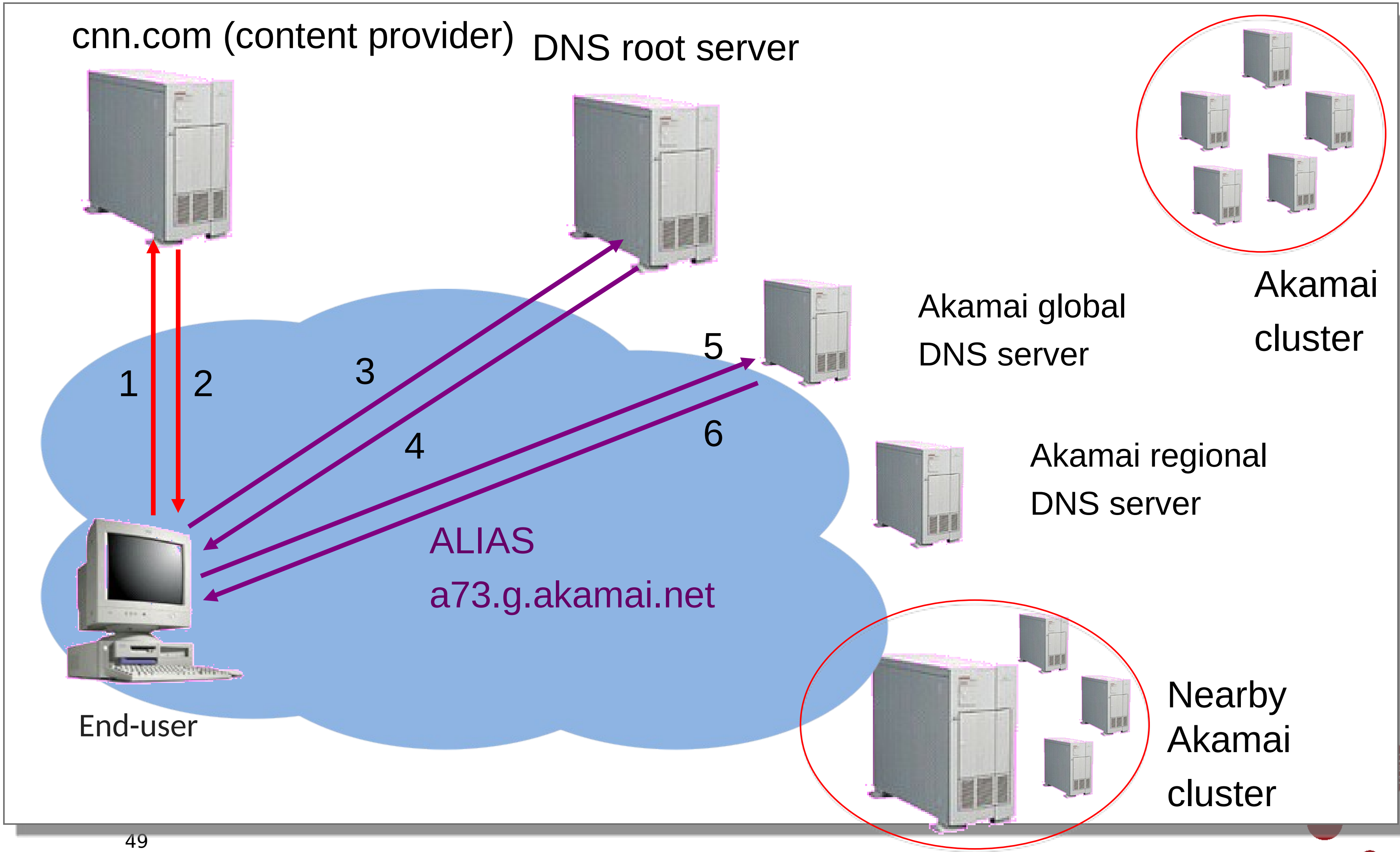
# How Akamai Works

Source: Freedman



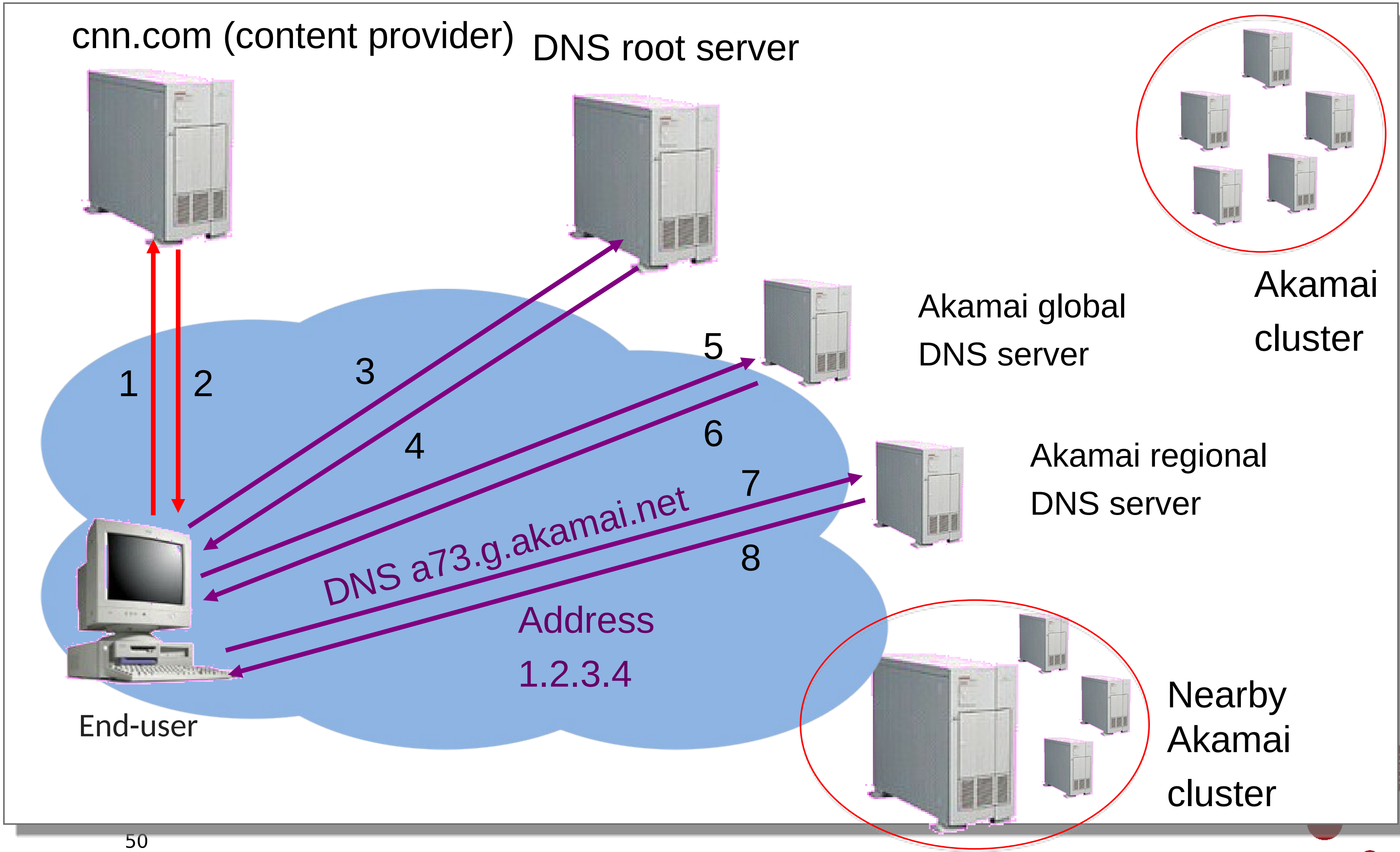
# How Akamai Works

Source: Freedman



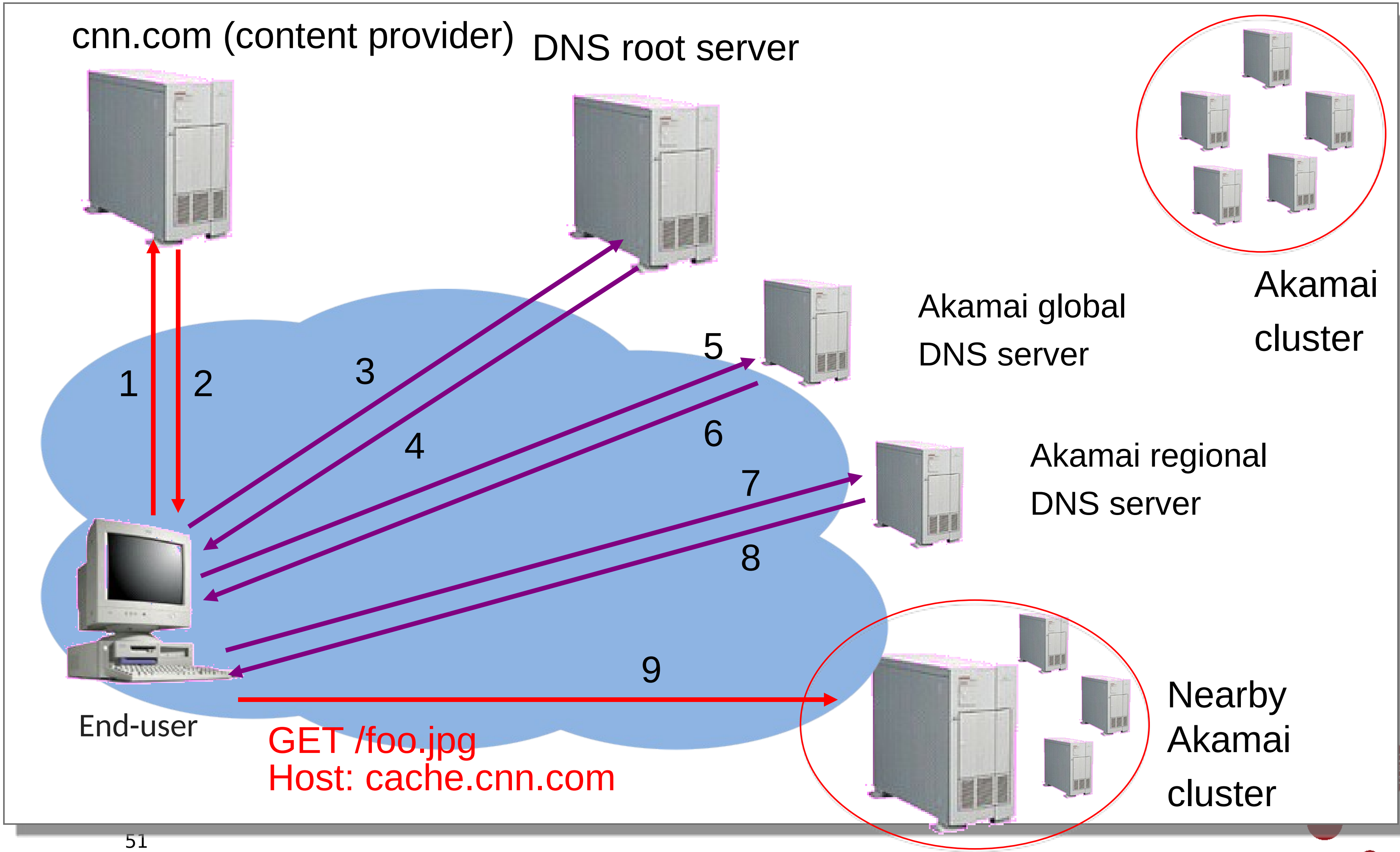
# How Akamai Works

Source: Freedman



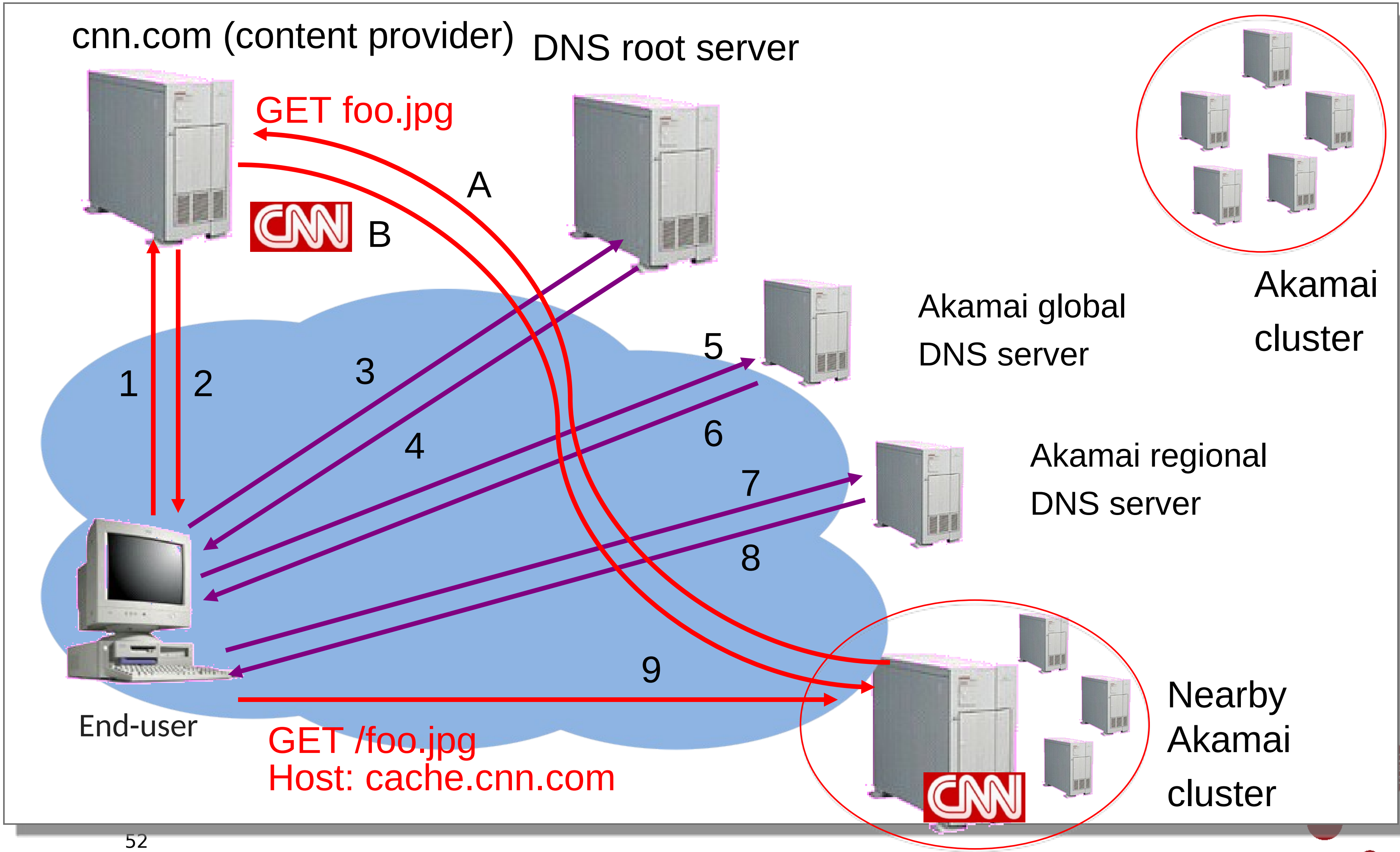
# How Akamai Works

Source: Freedman



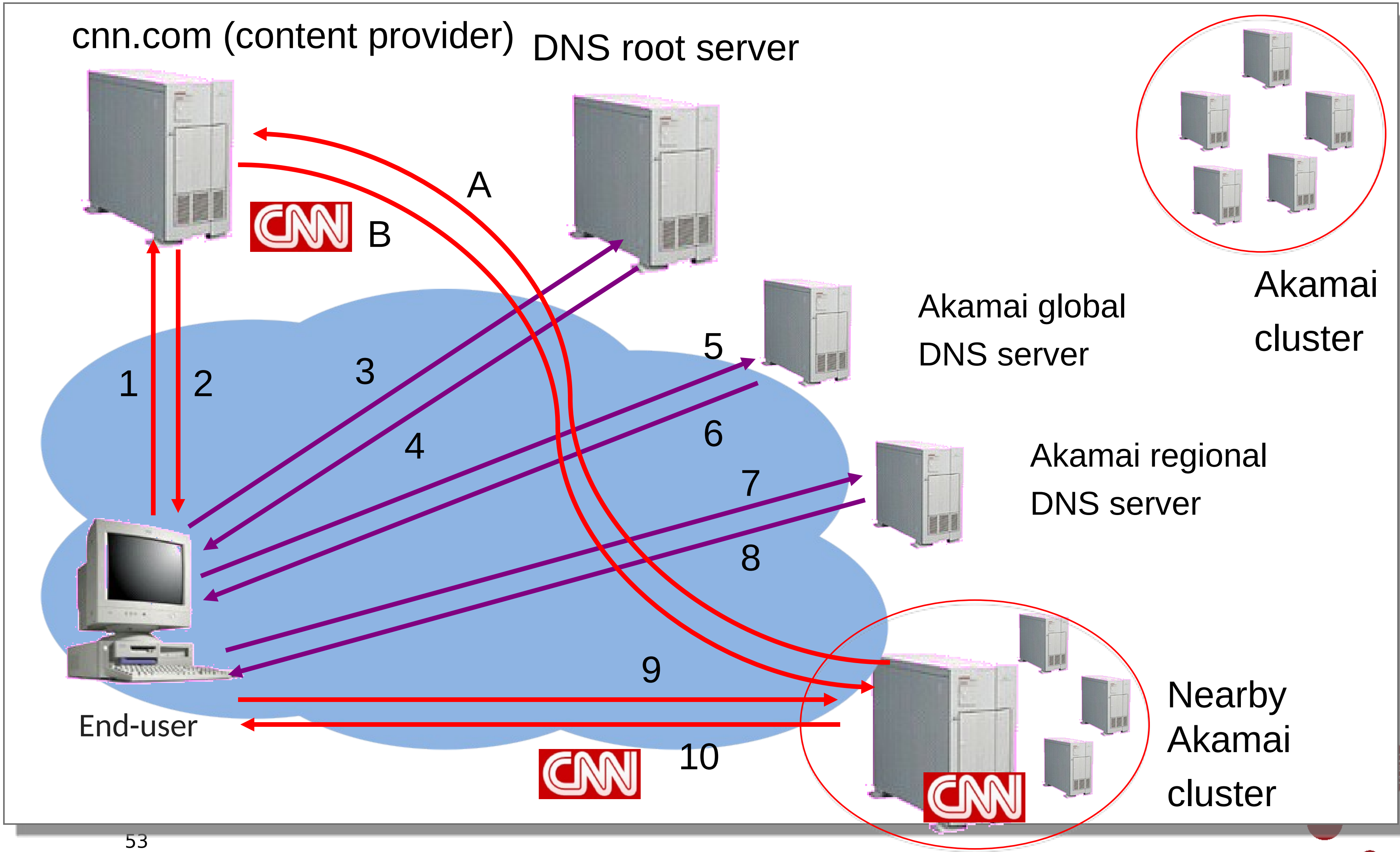
# How Akamai Works

Source: Freedman



# How Akamai Works

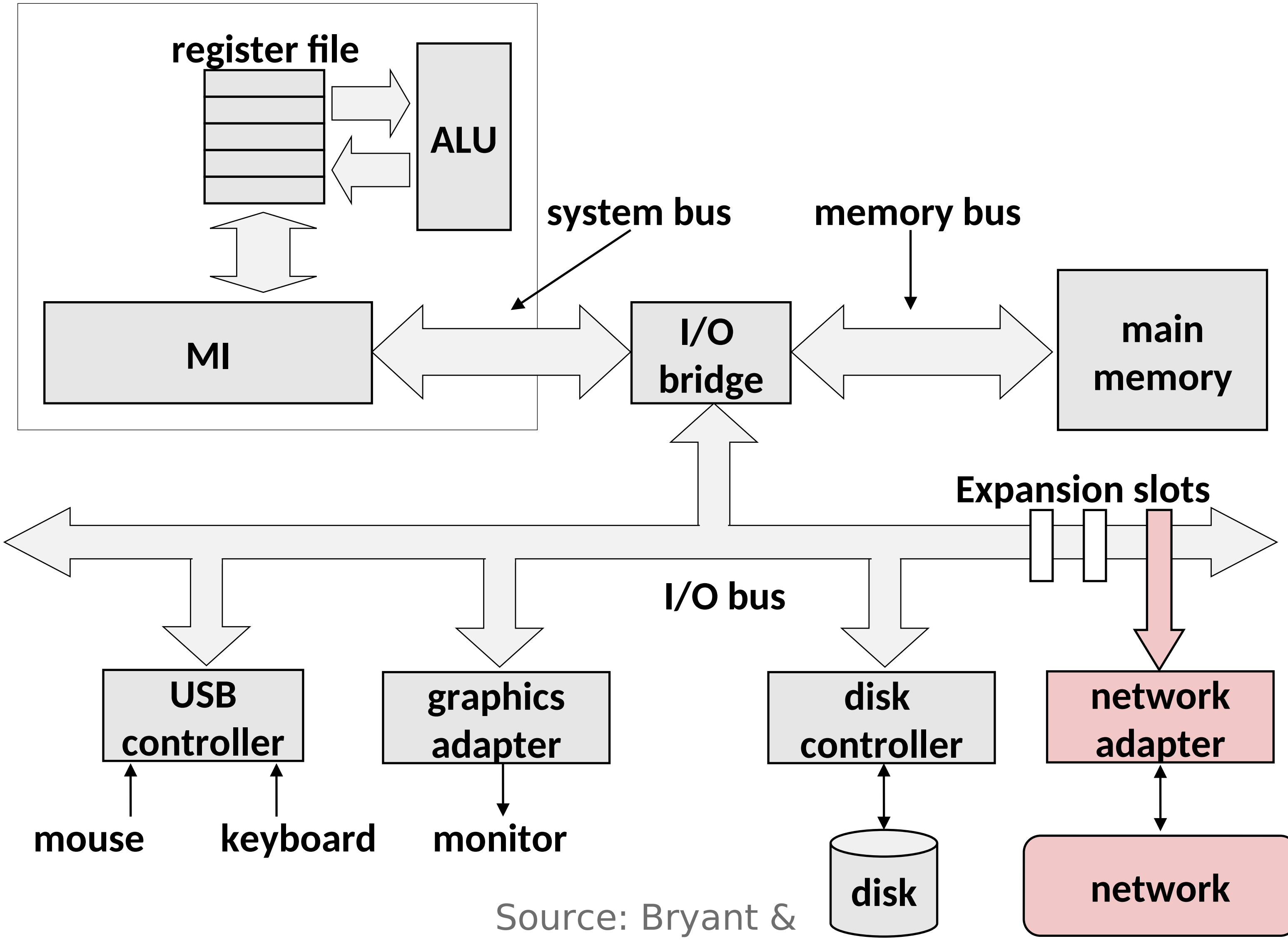
Source: Freedman



# Summary

- Network applications
  - Email, Web → more in textbook, Chapter 2!
- Socket abstraction
  - Communication between processes
  - Client / Server, Peer-to-Peer
- HTTP concepts
  - Web objects, request / response (pull)
  - Persistent connections, web proxies
  - Caching, content delivery

# What’s next? More network programming in C ?




Source: Bryant & Halloran