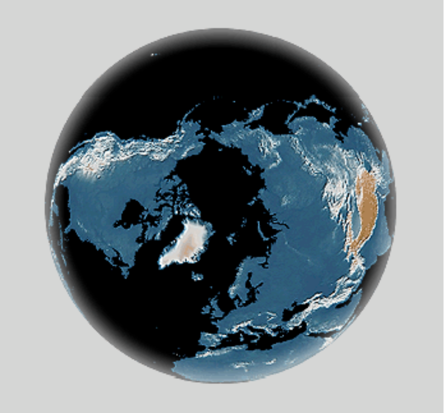


HTTP and Web Content Delivery

COS 461: Computer Networks
Spring 2011

Mike Freedman
http://www.cs.princeton.edu/courses/archive/spring11/cos461/

Outline

- Layering
- HTTP
- HTTP connection management and caching
- Proxying and content distribution networks
  - Web proxies and hierarchical networks
  - Modern distributed CDNs (Akamai)

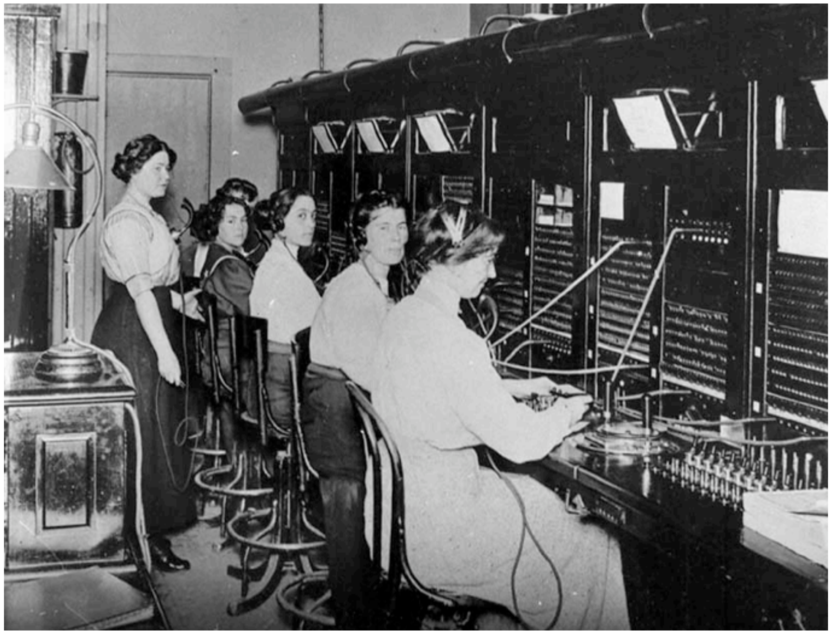
- Assignment #1 (available next week):
  - Write a basic Web proxy
    - (It will work with your browser and real web pages!)

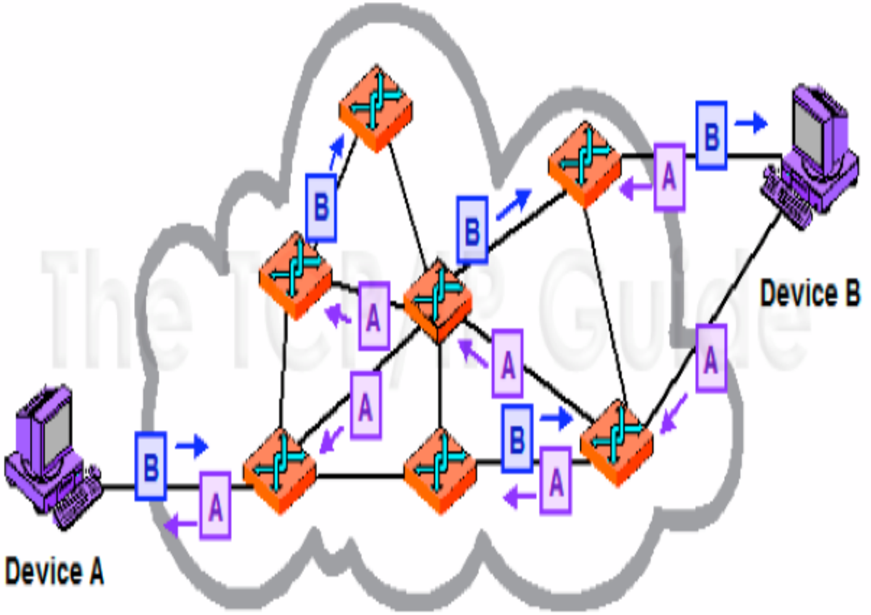
HTTP Basics

- 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
- 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

HTTP needs a stream of data

Circuit Switching → Packet switching





http://www.tcpipguide.com/free/t_CircuitSwitchingandPacketSwitchingNetworks-2.htm

Today’s networks provide packet delivery, not streams!

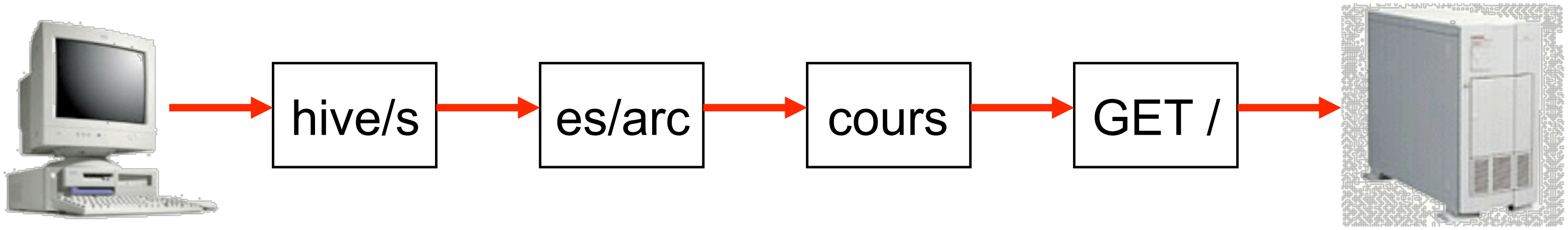
What if the Data Doesn’t Fit?

```
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



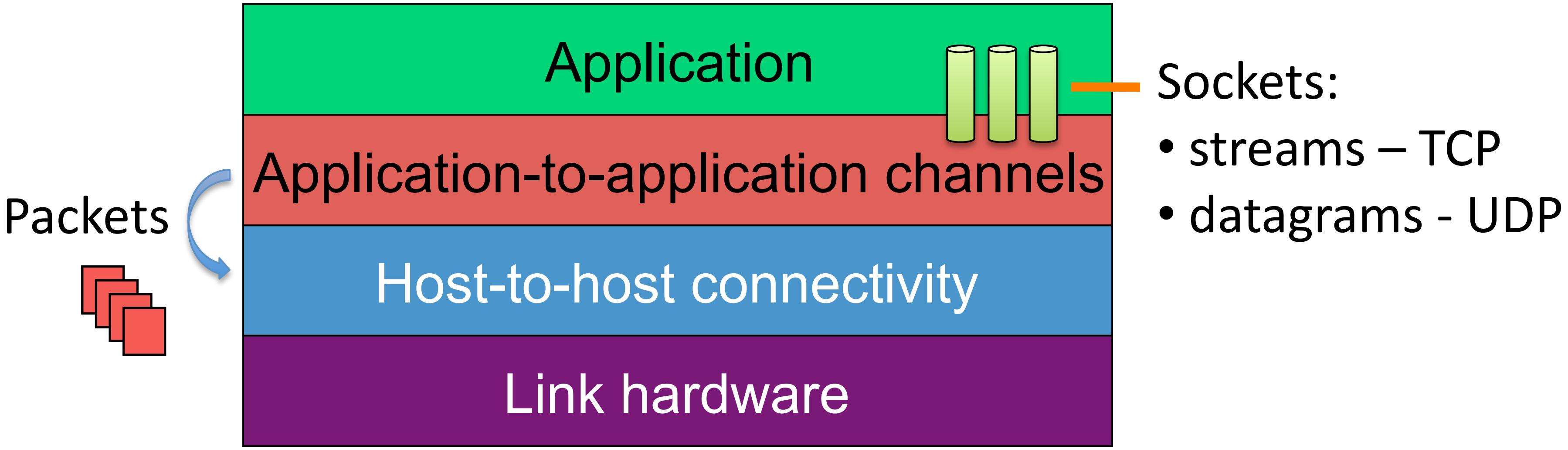
hive/s → es/arc → cours → GET /

GET index.html

Solution: Split the data across multiple packets

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



Application

Application-to-application channels

Host-to-host connectivity

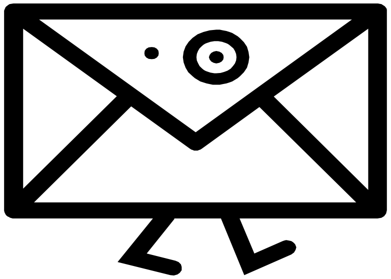
Link hardware

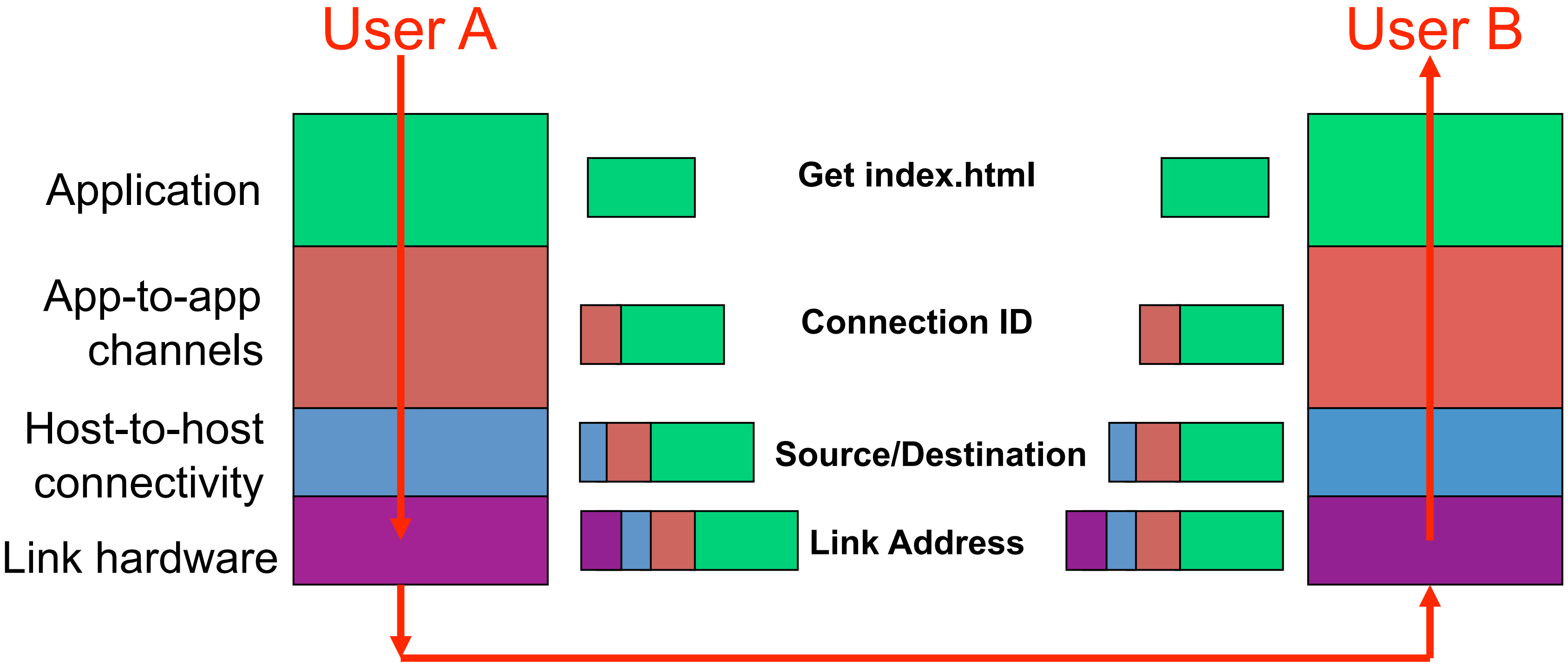
Packets

Sockets:

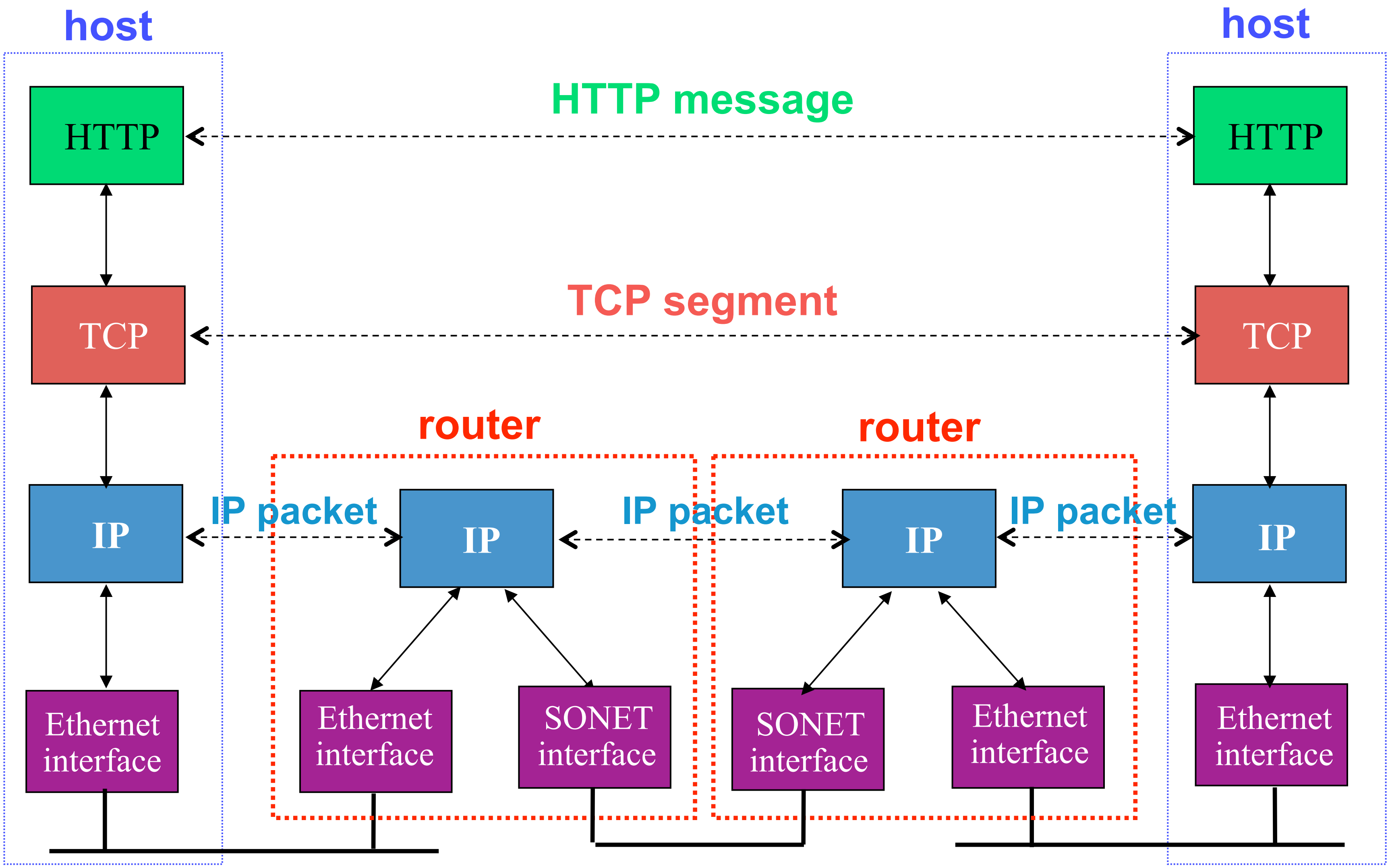
- streams – TCP
- datagrams - UDP

Layer Encapsulation in HTTP





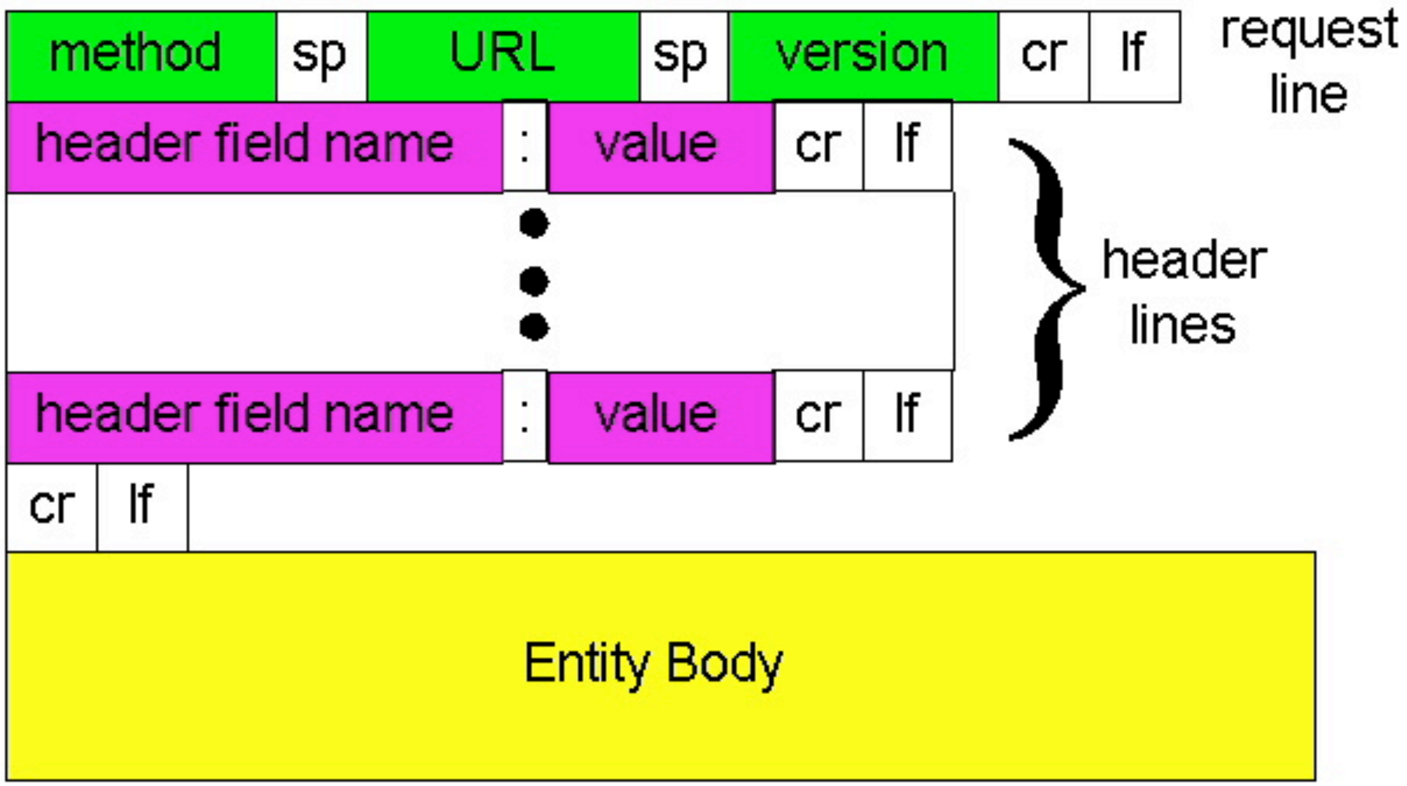
IP Suite: End Hosts vs. Routers



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
```

HTTP Request



method sp URL sp version cr lf

request line

header field name : value cr lf

header field name : value cr lf

header lines

cr lf

Entity Body

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">
```

How to Mark End of Message?

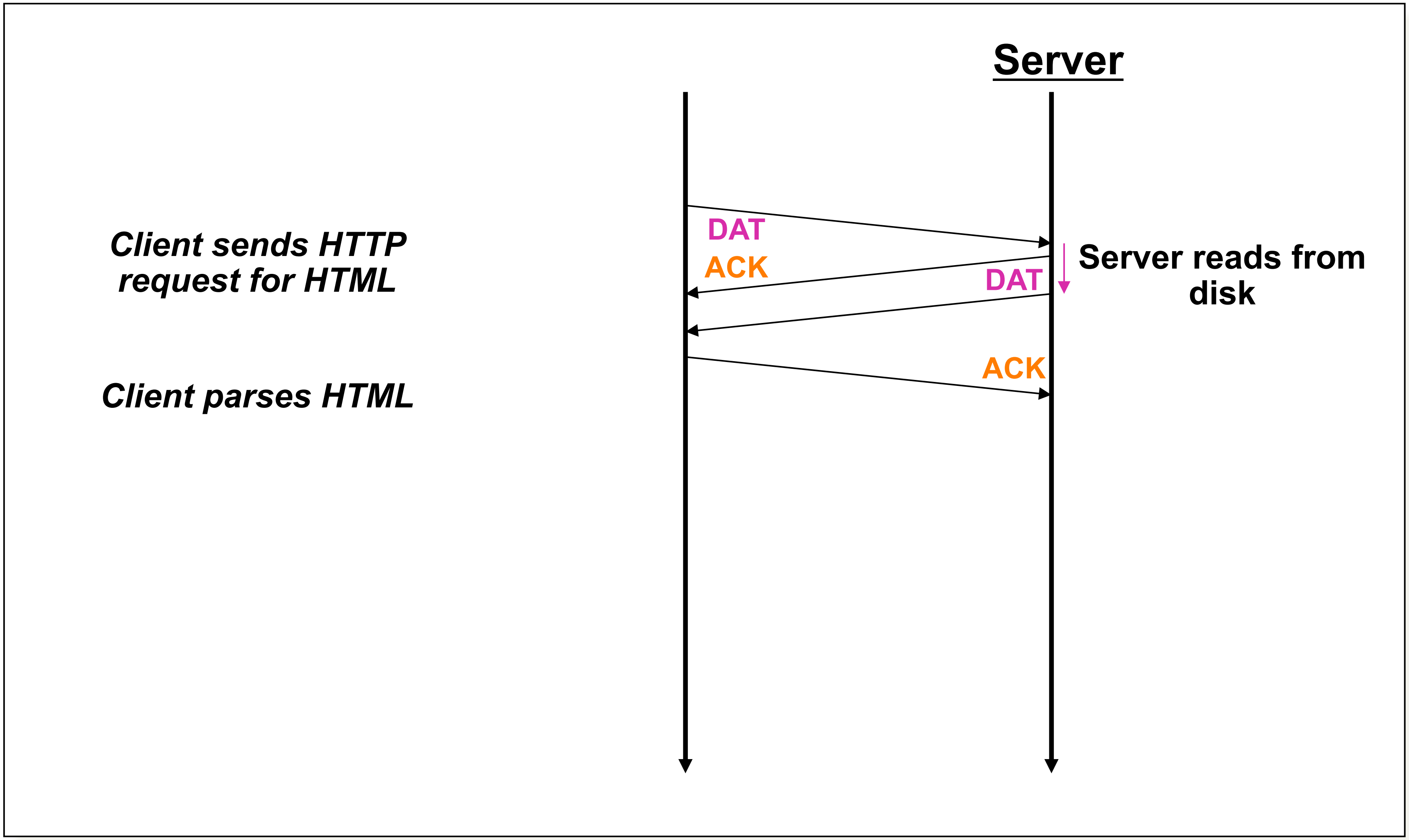
- Close connection
  - Only server can do this
- Content-Length
  - Must know size of transfer in advance
- Implied length
  - E.g., 304 (NOT MODIFIED) never have body content
- Transfer-Encoding: chunked (HTTP/1.1)
  - After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.

Example: Chunked Encoding

```
HTTP/1.1 200 OK <CRLF>
Transfer-Encoding: chunked <CRLF>
<CRLF>
25 <CRLF>
This is the data in the first chunk <CRLF>
1A <CRLF>
and this is the second one <CRLF>
0 <CRLF>
```

- Especially useful for dynamically-generated content, as length is not a priori known

Single Transfer Example

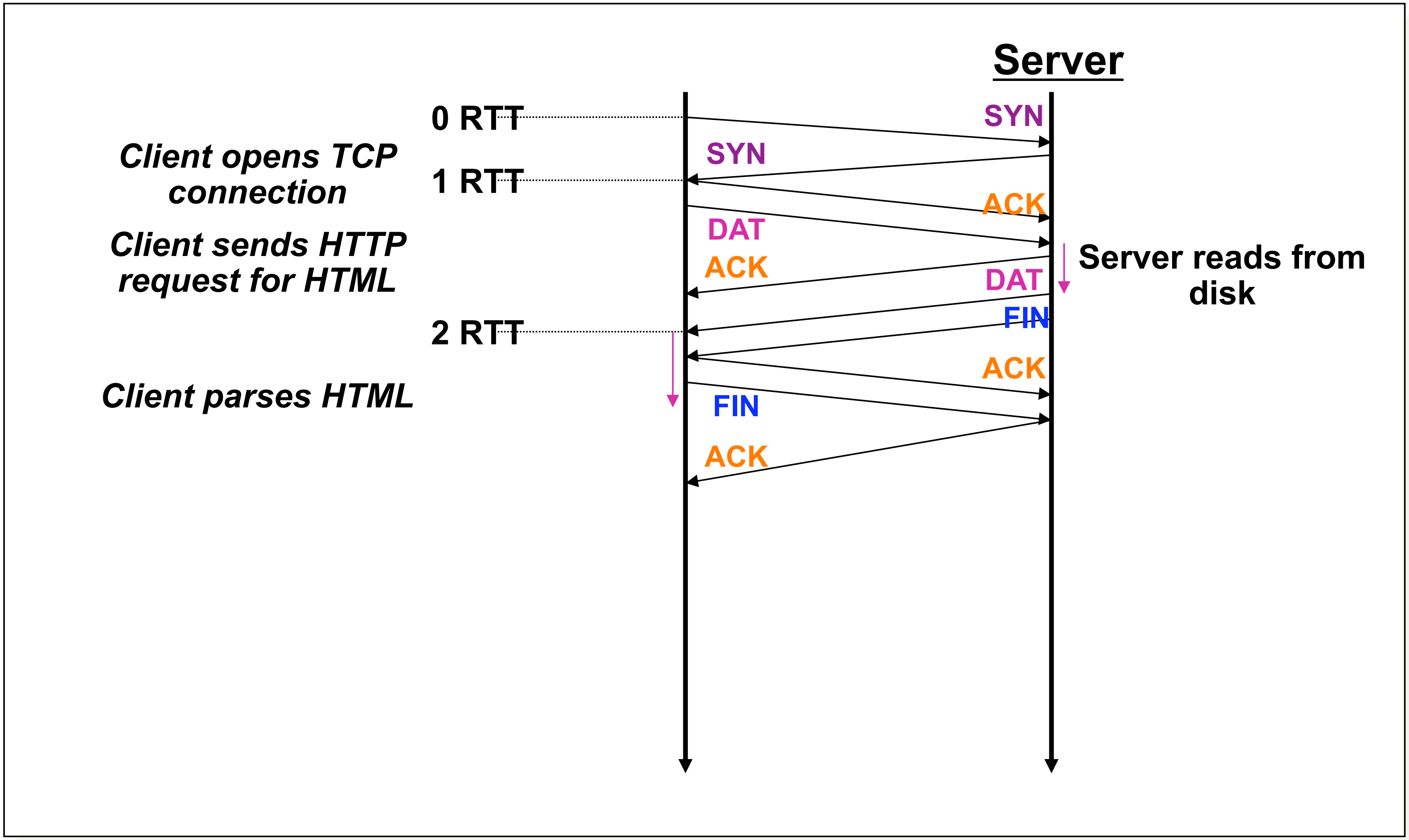


Client sends HTTP request for HTML

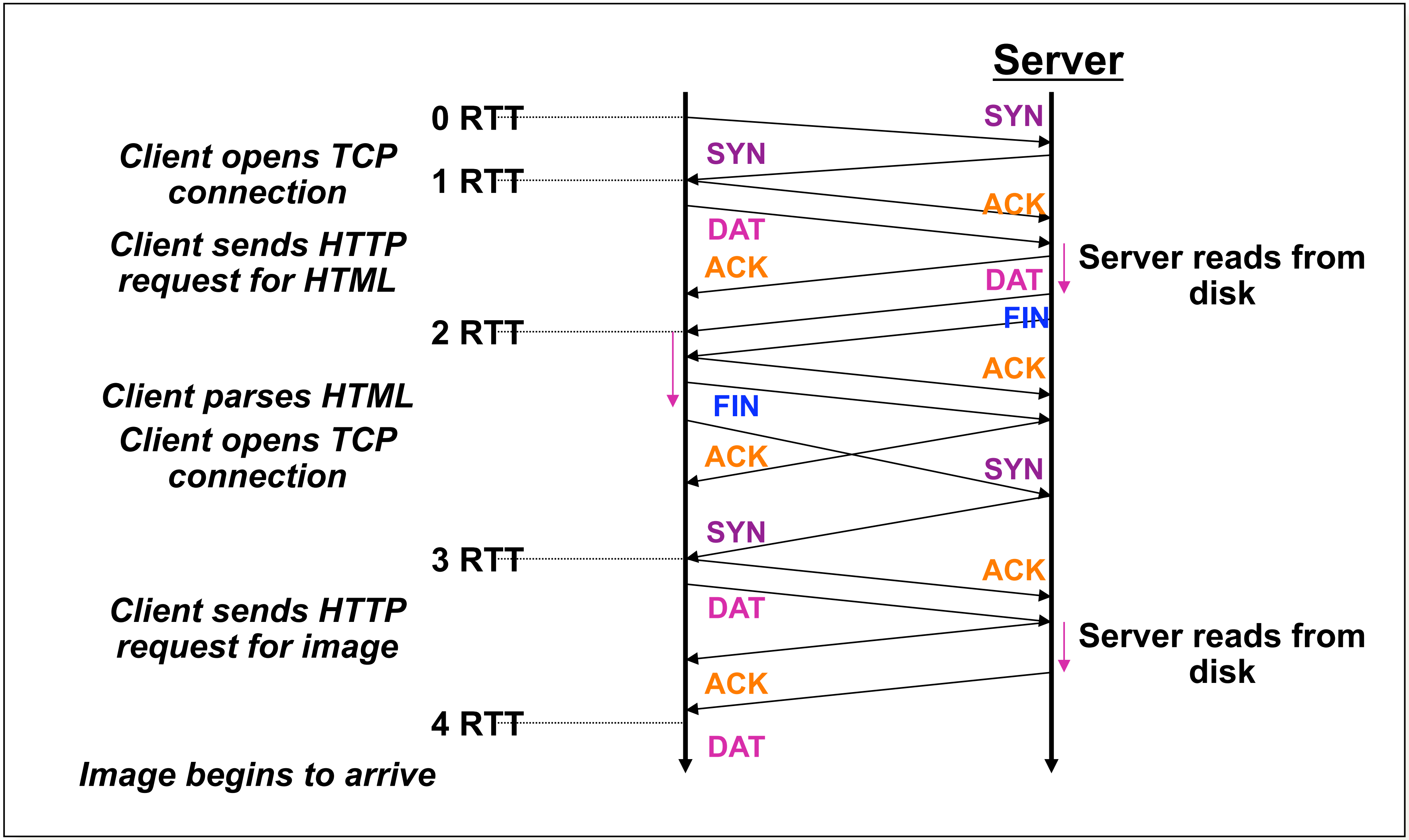
Server reads from disk

Client parses HTML

Single Transfer Example



Single Transfer Example



Problems with simple model

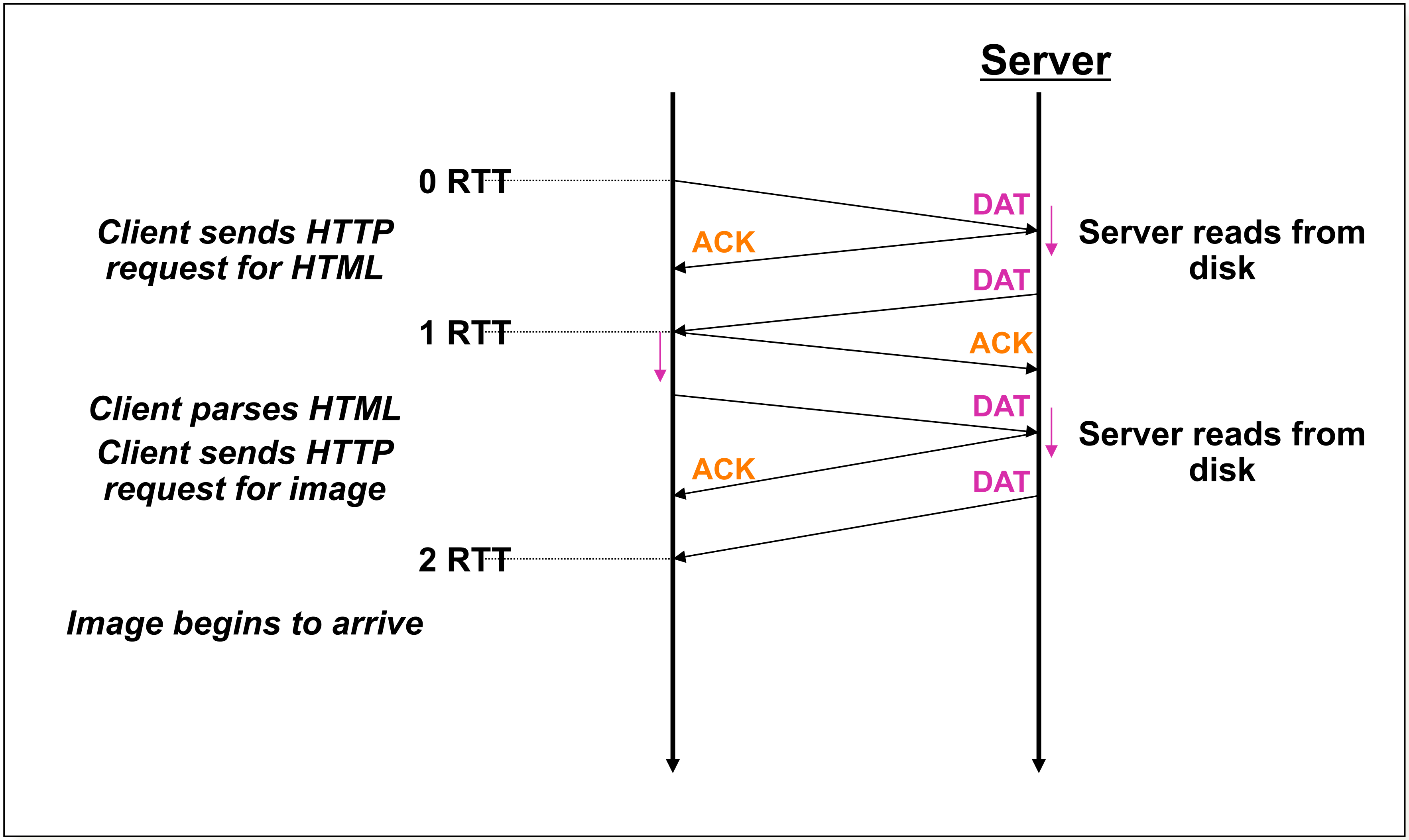
- Multiple connection setups
  - Three-way handshake each time (TCP "synchronizing" stream)
- Short transfers are hard on stream protocol (TCP)
  - How much data should it send at once?
  - Congestion avoidance: Takes a while to "ramp up" to high sending rate (TCP "slow start")
  - Loss recovery is poor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

Outline

- Layering
- HTTP
- HTTP connection management and caching
- Proxying and content distribution networks
  - Web proxies and hierarchical networks
  - Modern distributed CDNs (Akamai)

- Assignment #1 (available next week):
  - Write a basic Web proxy
    - (It will work with your browser and real web pages!)

Persistent Connection Example



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

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?
  - Small objects, equal time to serve each object
  - Small because pipelining simply removes additional 1 RTT delay to request new content
- Alternative design?
  - Multiple parallel connections (~2-4). Easier server parallelism
  - No "head-of-line blocking" problem like pipelining
    - Dynamic content makes HOL blocking possibility worse
- In practice, many servers don't support, and many browsers do not default to pipelining

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

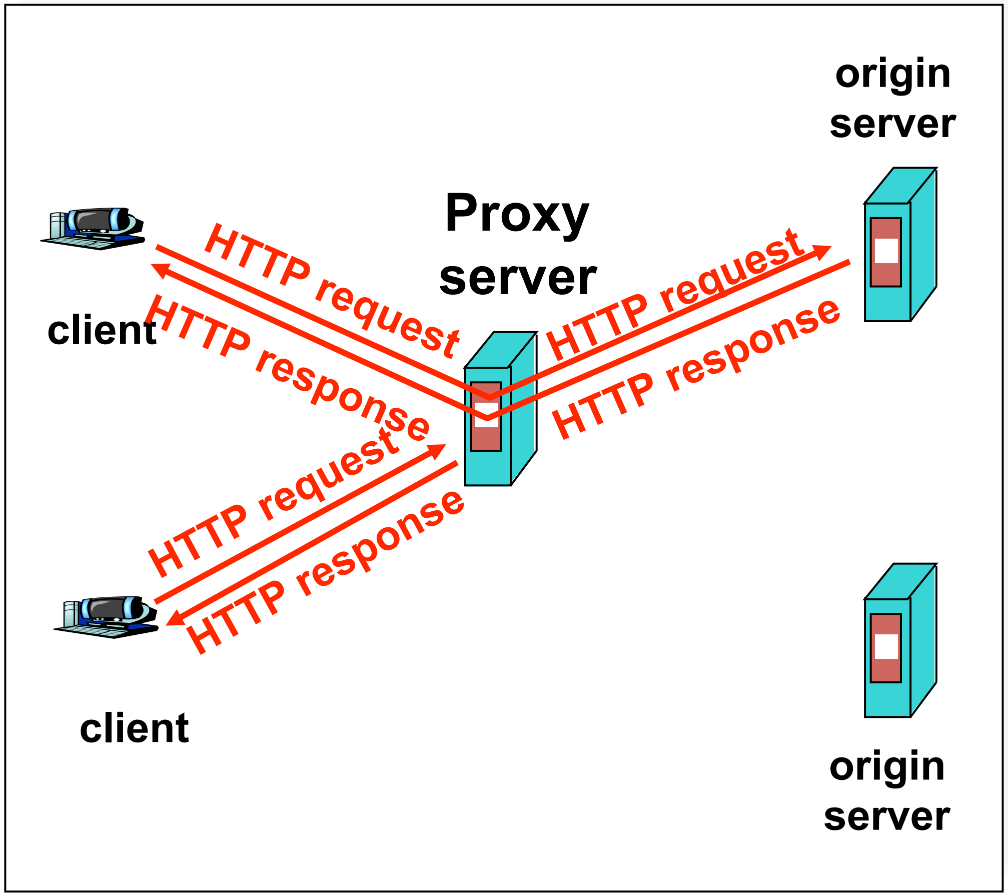
HTTP Conditional Request

```
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
```

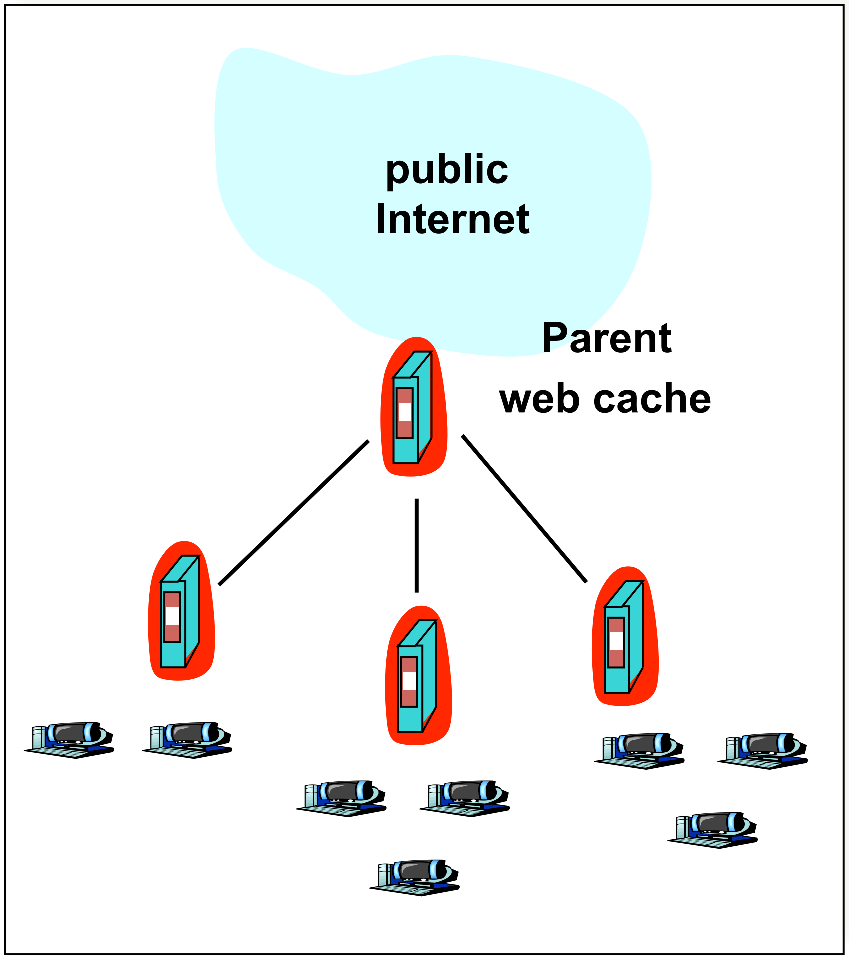
Web Proxy Caches

- 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

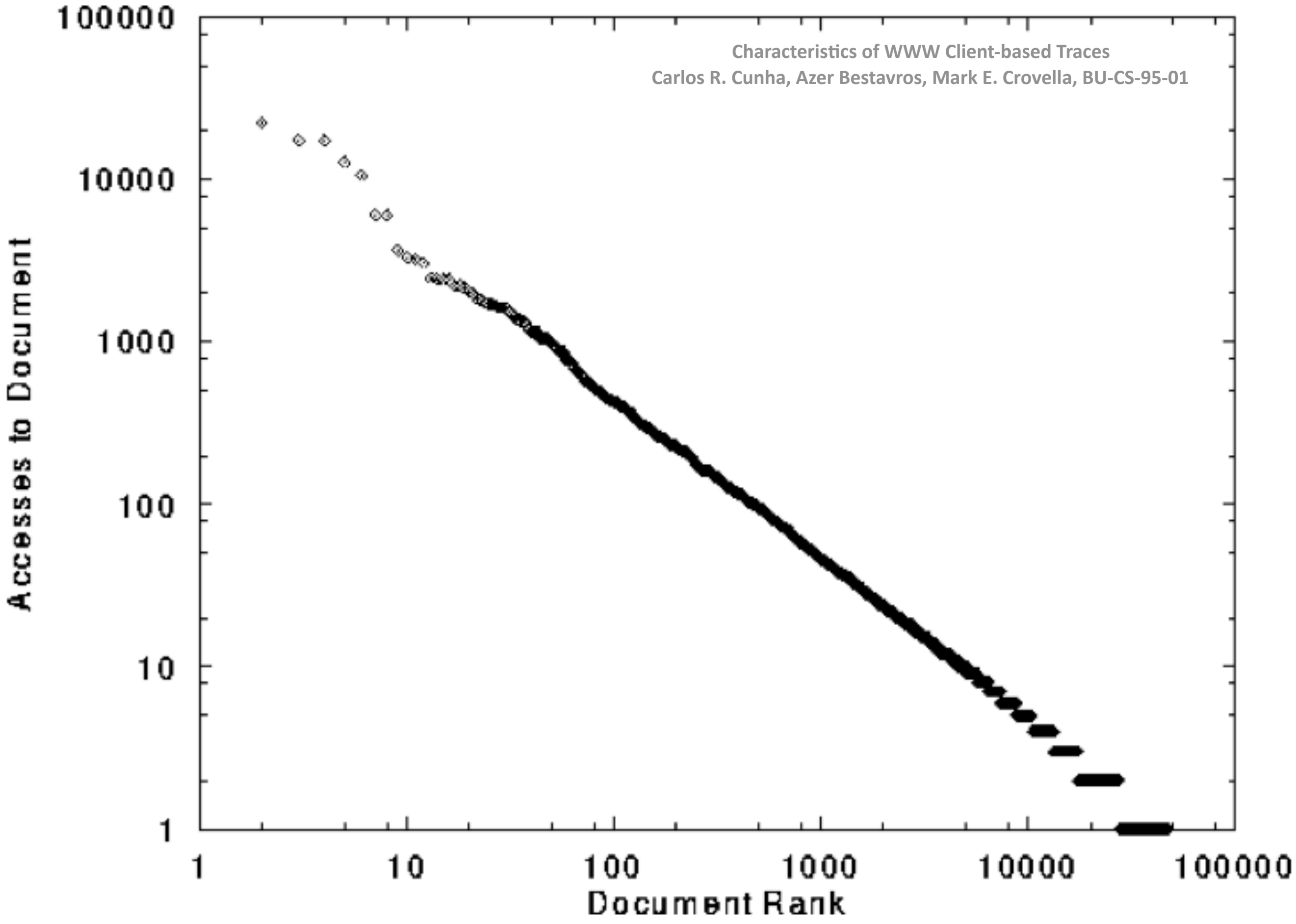


When a single cache isn't enough

- 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



Web traffic has cacheable workload



Characteristics of WWW Client-based Traces
Carlos R. Cunha, Azer Bestavros, Mark E. Crovella, BU-CS-95-01

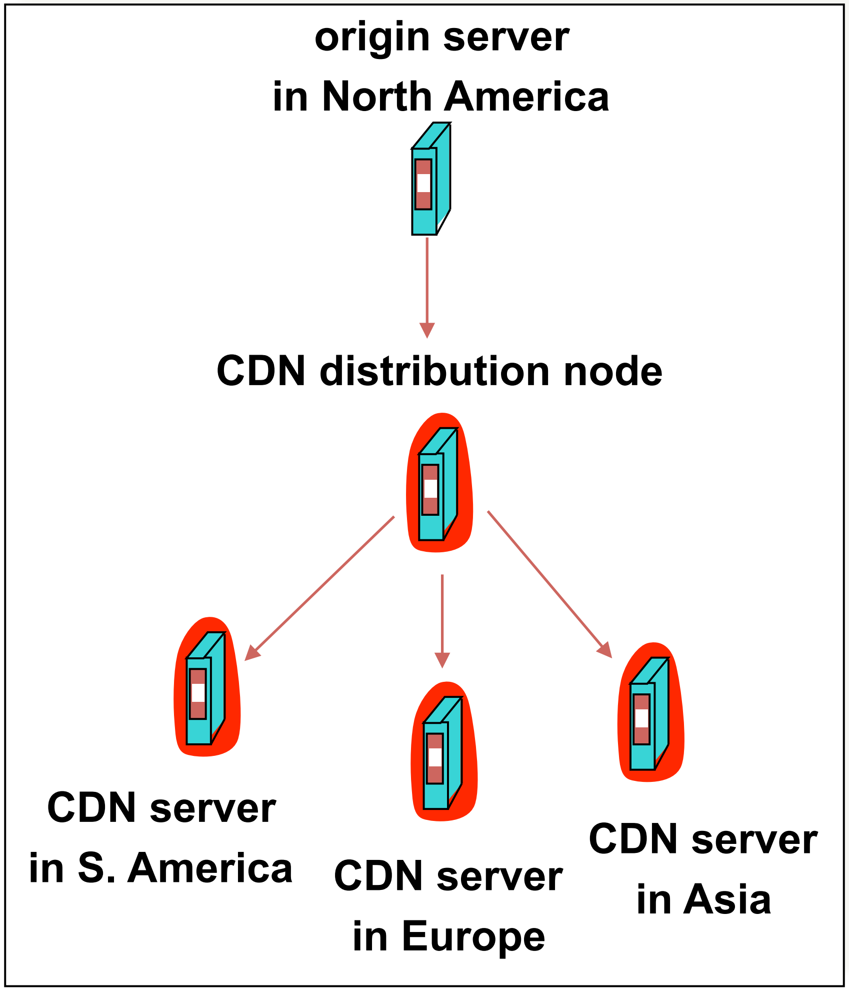
"Zipf" or "power-law" distribution

Content Distribution Networks (CDNs)

- Content providers are CDN customers

Content replication

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



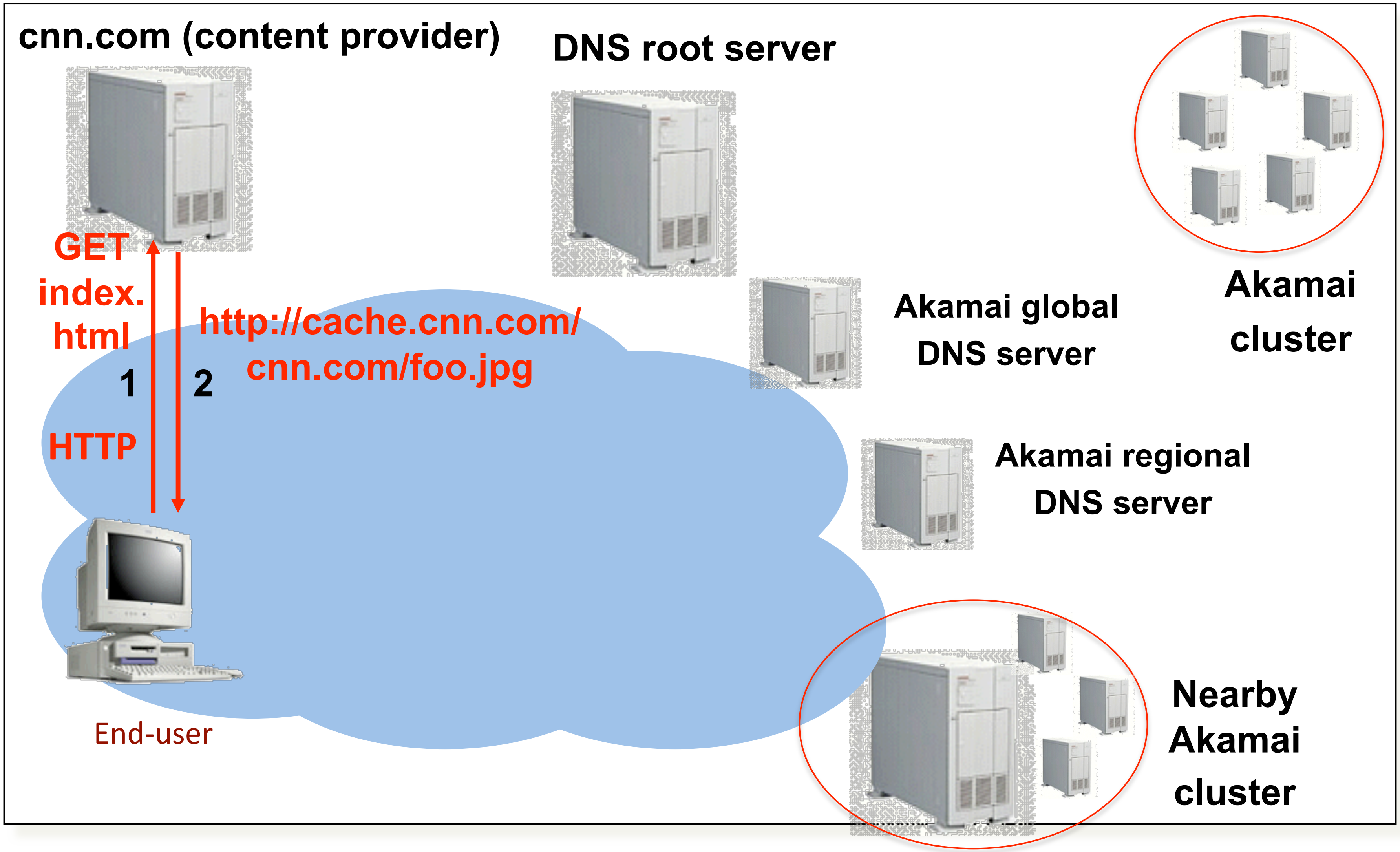
Content Distribution Networks & Server Selection

- Replicate content on many servers
- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among know replicas
  - How to direct clients towards replica

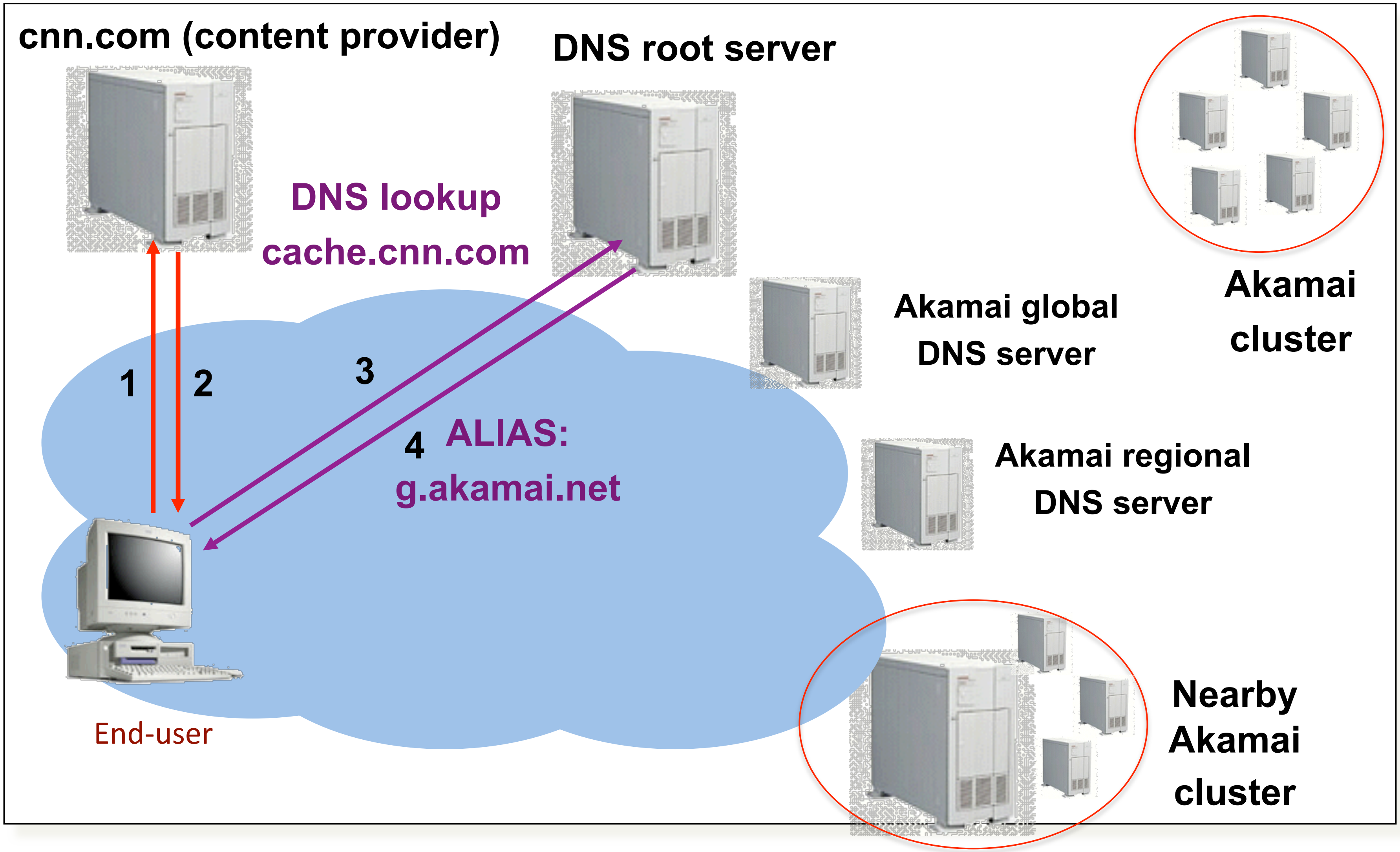
Server Selection

- 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

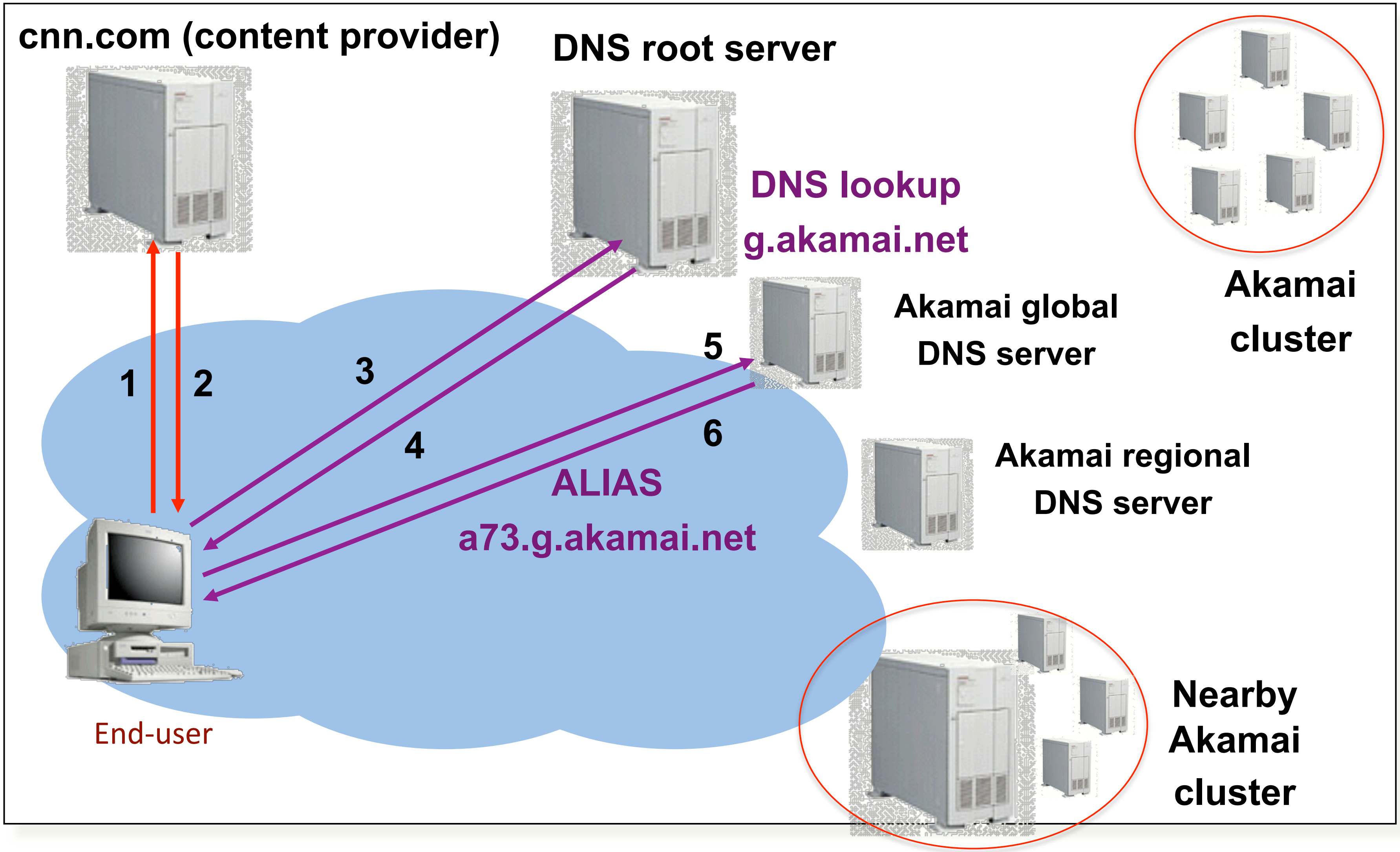
How Akamai Works



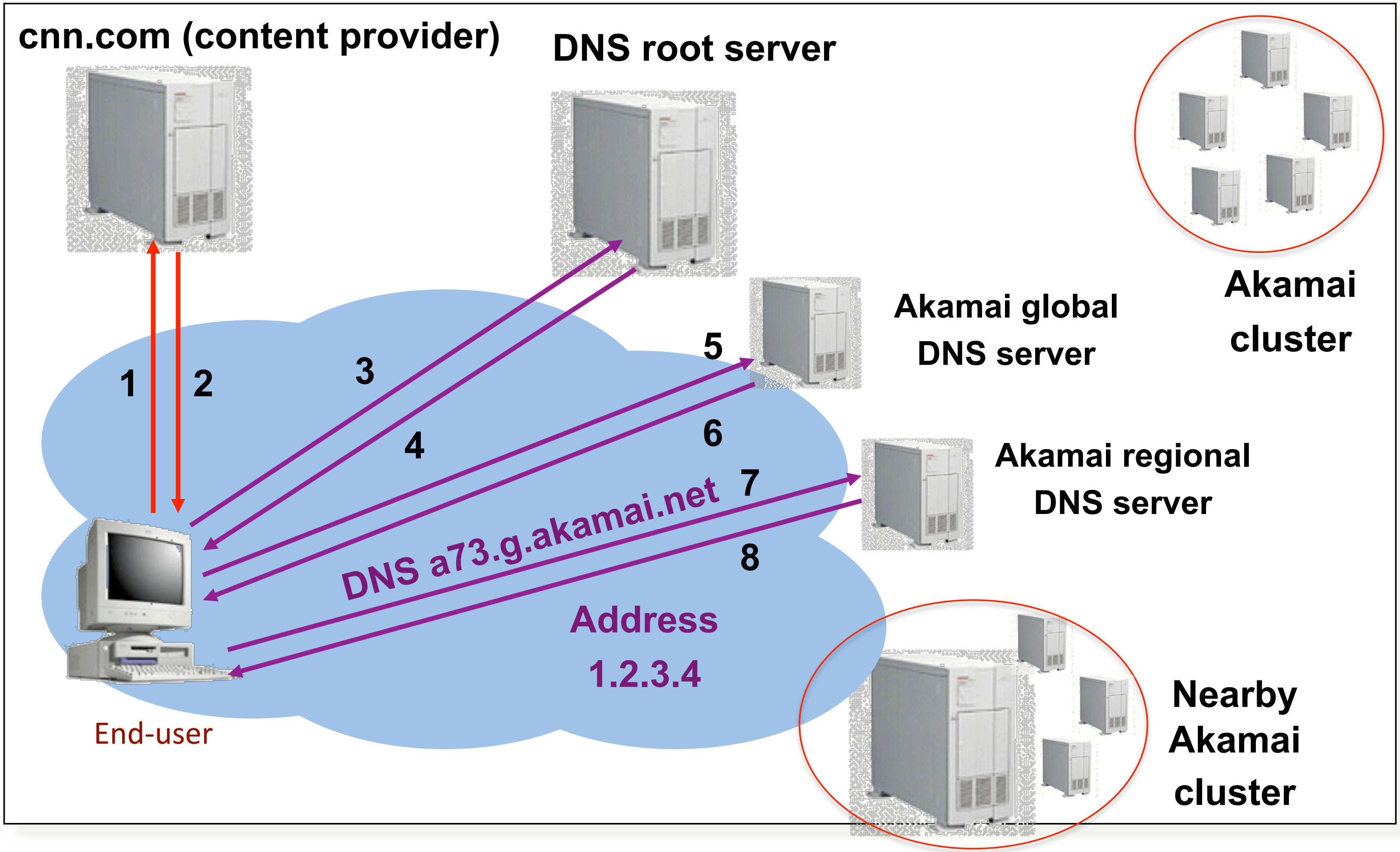
How Akamai Works



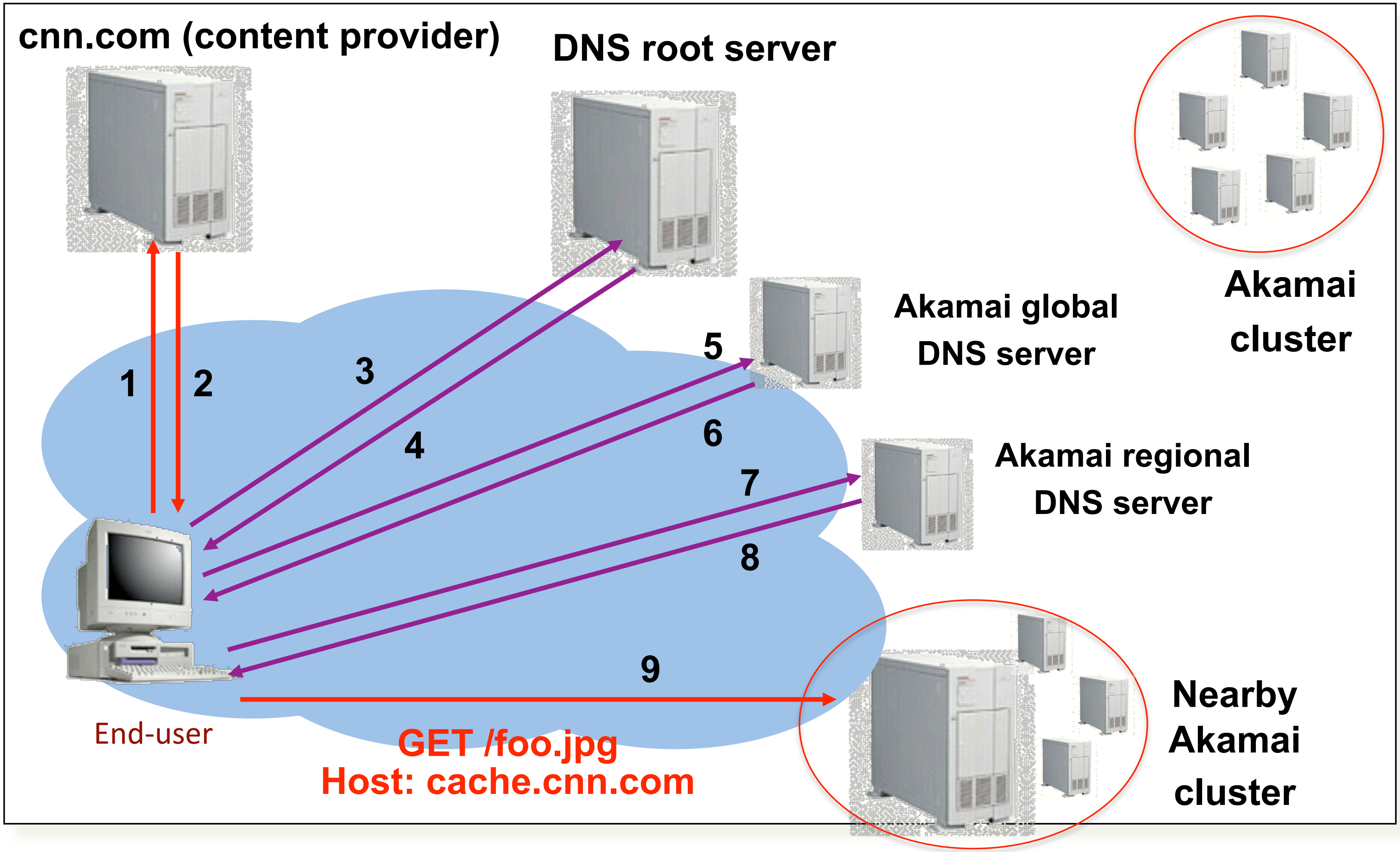
How Akamai Works



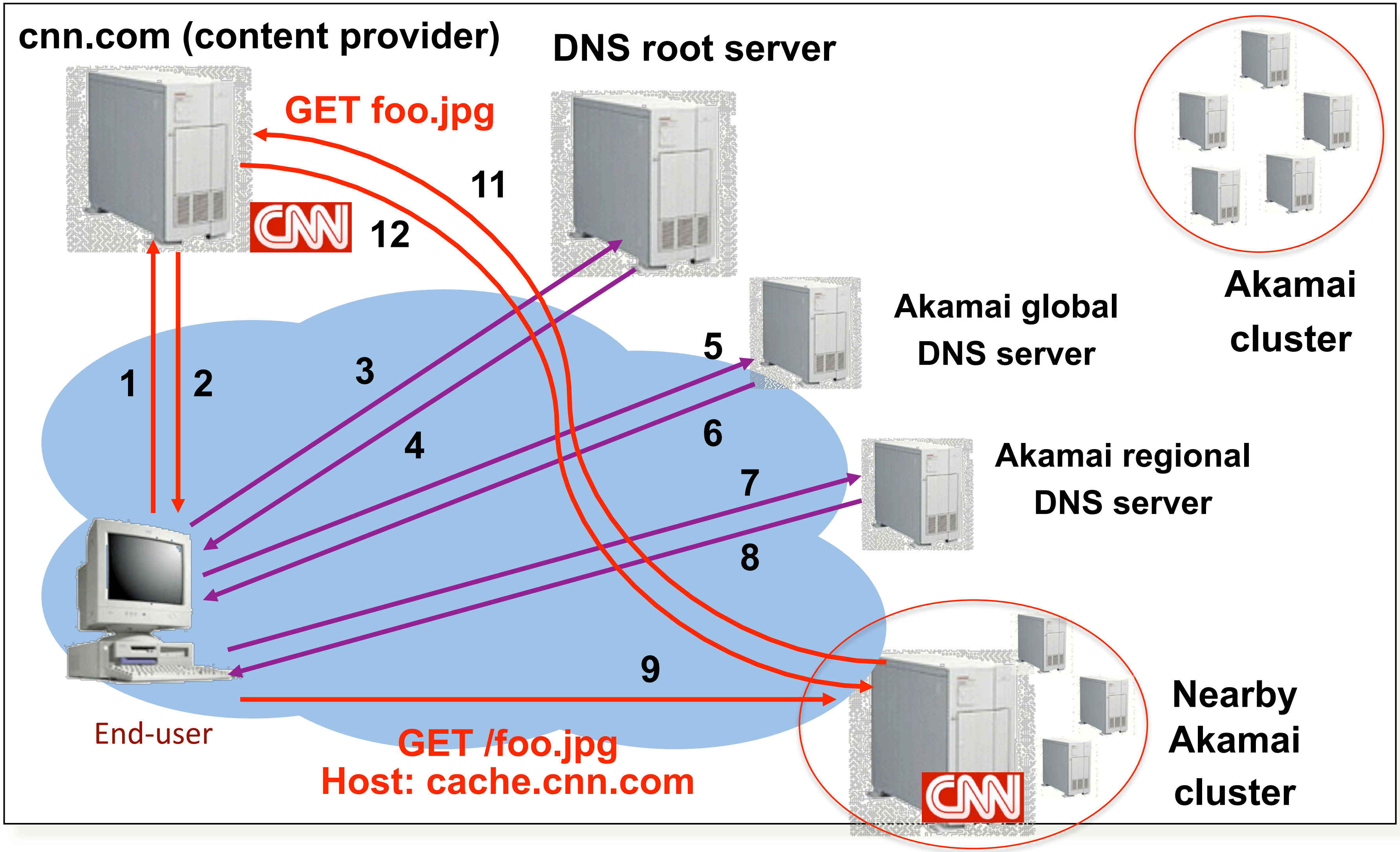
How Akamai Works



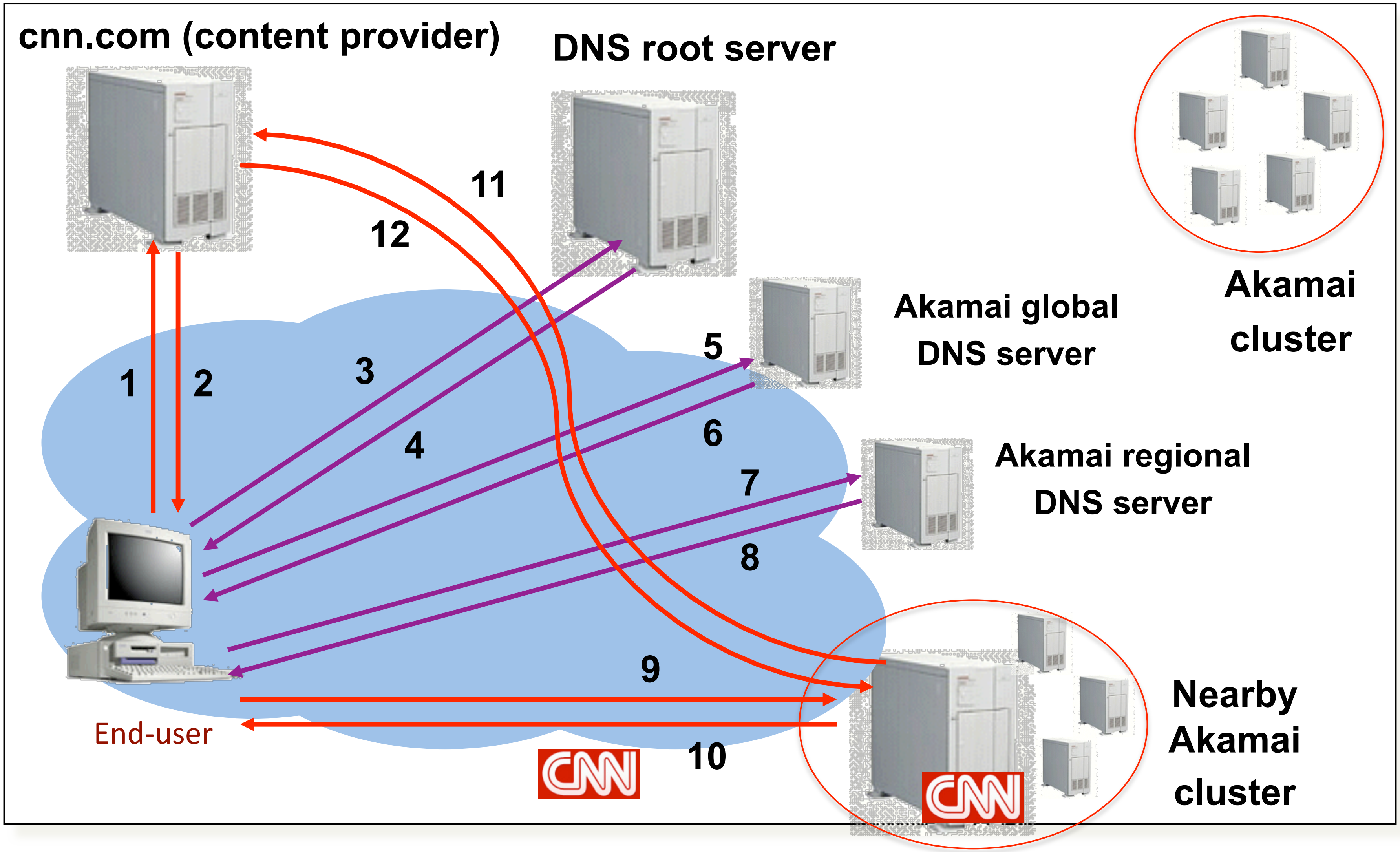
How Akamai Works



How Akamai Works



How Akamai Works



Summary

- HTTP: Simple text-based file exchange protocol
  - Support for status/error responses, authentication, client-side state maintenance, cache maintenance
- Interactions with TCP
  - Connection setup, reliability, state maintenance
  - Persistent connections
- How to improve performance
  - Persistent and pipelined connections
  - Caching
  - Replication: Web proxies, cooperative proxies, and CDNs