ECS524
Application Layer

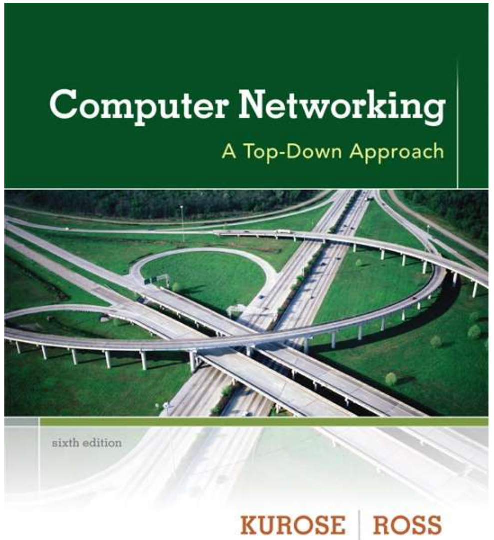
Prof. Steve Uhlig
steve.uhlig@qmul.ac.uk
Office: Eng.202

Dr. Felix Cuadrado
felix.cuadrado@qmul.ac.uk
Office: Eng 202

Slides

Disclaimer:
Some of the slides' content is borrowed directly from those provided by the authors of the textbook. They are available from

http://www-net.cs.umass.edu/kurose-ross-ppt-6e



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

The Application Layer

- **Internet applications**
- HTTP
- DNS
- Content delivery
- P2P

Some networked apps

Networked applications

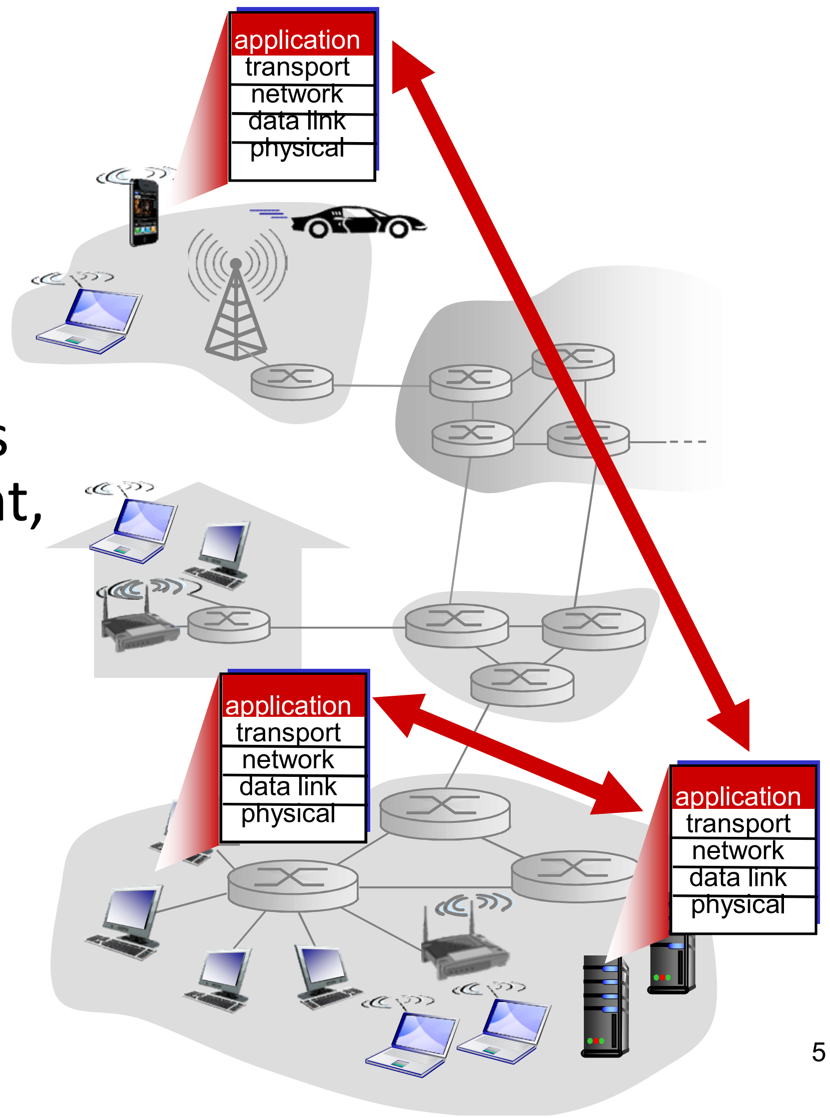
Programs that:

- run on (different) ***end systems***
- communicate over network

  e.g., web server software communicates with mobile browser software
- applications only on end systems allows for rapid app development, propagation
- two main interaction types

  client – server

  peer to peer



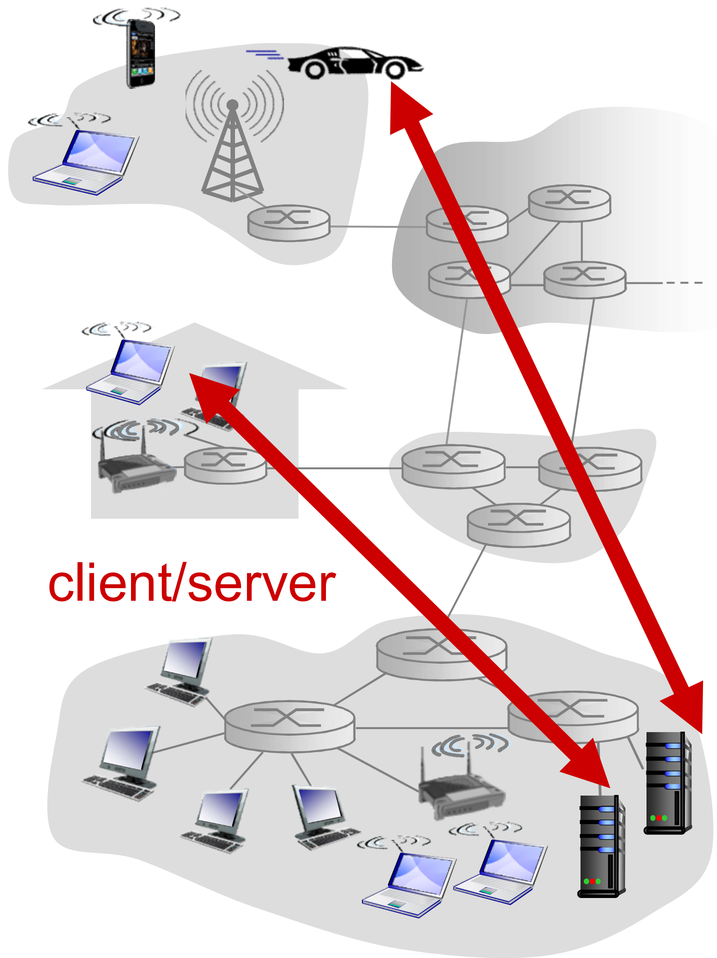
Client-server architecture

server:

- always-on host
- permanent IP address
- serves multiple clients

  data centres for scaling

  cloud computing



clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Anatomy of a datacentre

http://www.youtube.com/watch?v=avP5d16wEp0

P2P architecture

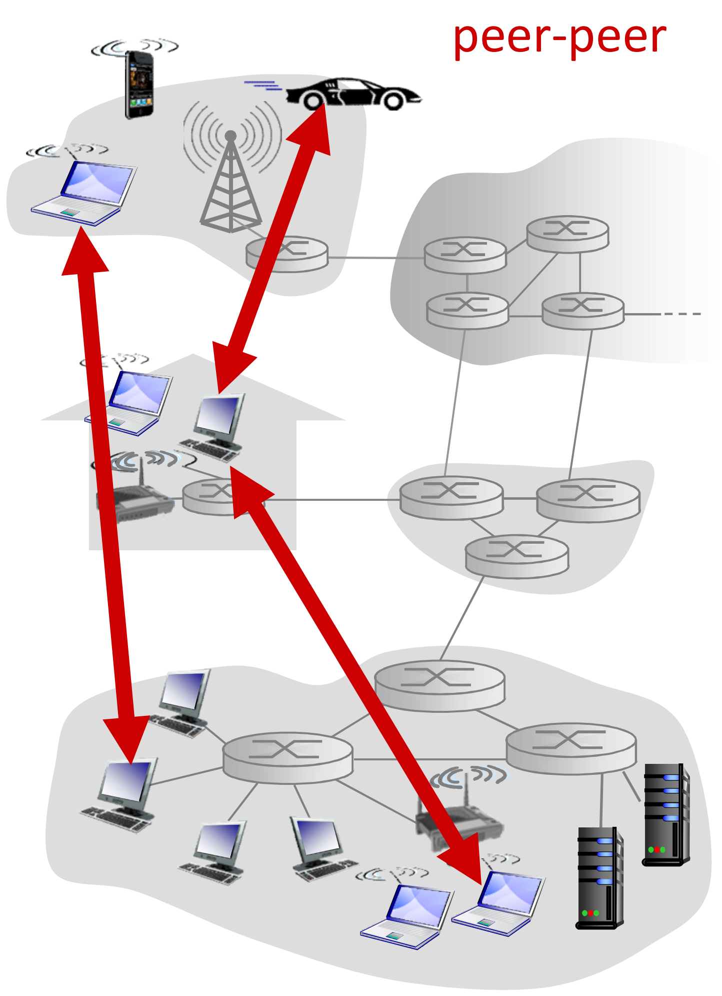
- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers

  every peer acts as client and server

  self scalability – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses

  complex management

  hard to provide guarantees



App-layer protocol defines

- types of messages exchanged
  e.g., request, response
- message syntax
  what fields in messages & how fields are delineated
- message semantics
  meaning of information in fields
- rules
  when and how processes send & respond to messages

The Application Layer

- Internet applications
- **HTTP**
- DNS
- Content delivery
- P2P

Web and HTTP

web page consists of *objects*

object can be HTML file, XML-Json data, js client-side code, JPEG image, audio file,…

web page consists of *base HTML-file* which includes *several referenced objects*

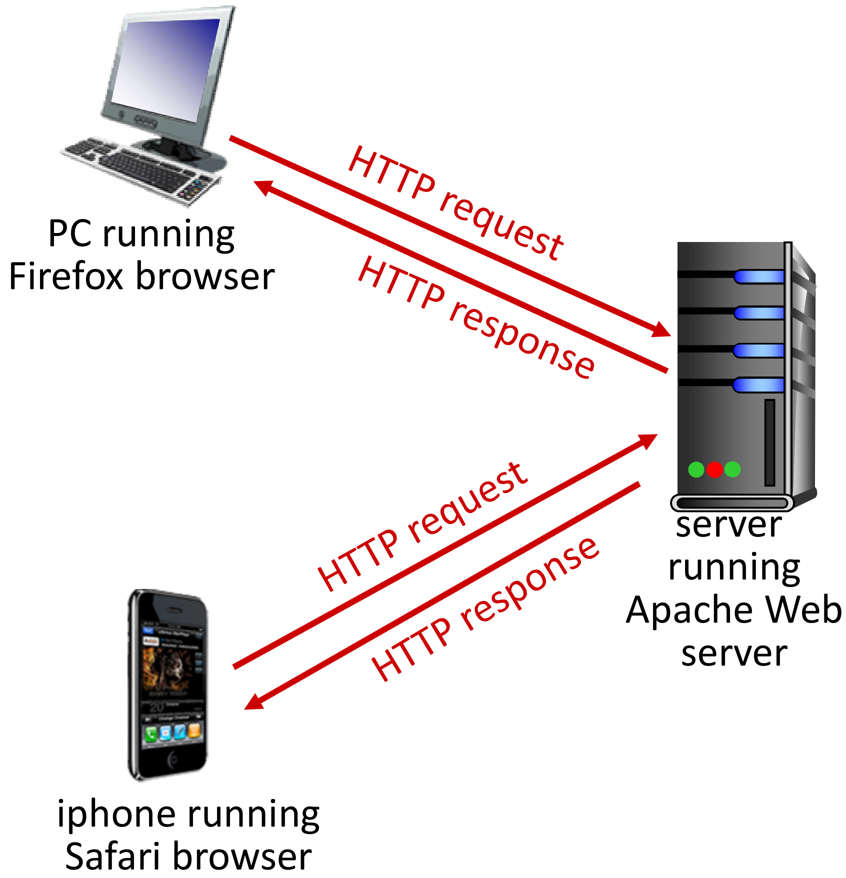
each object is addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

uses TCP:

client initiates TCP connection (creates socket) to server, port 80

server accepts TCP connection from client

HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

TCP connection closed

HTTP is "stateless"

server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Sample HTTP interaction

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP (cont.)

4. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

5. Steps 1-4 repeated for each of 10 jpeg objects

6. HTTP server closes TCP connection.

HTTP request message

two types of HTTP messages: *request, response*

HTTP request message:

ASCII (human-readable format)

carriage return character

line-feed character

request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Request Methods

GET Method

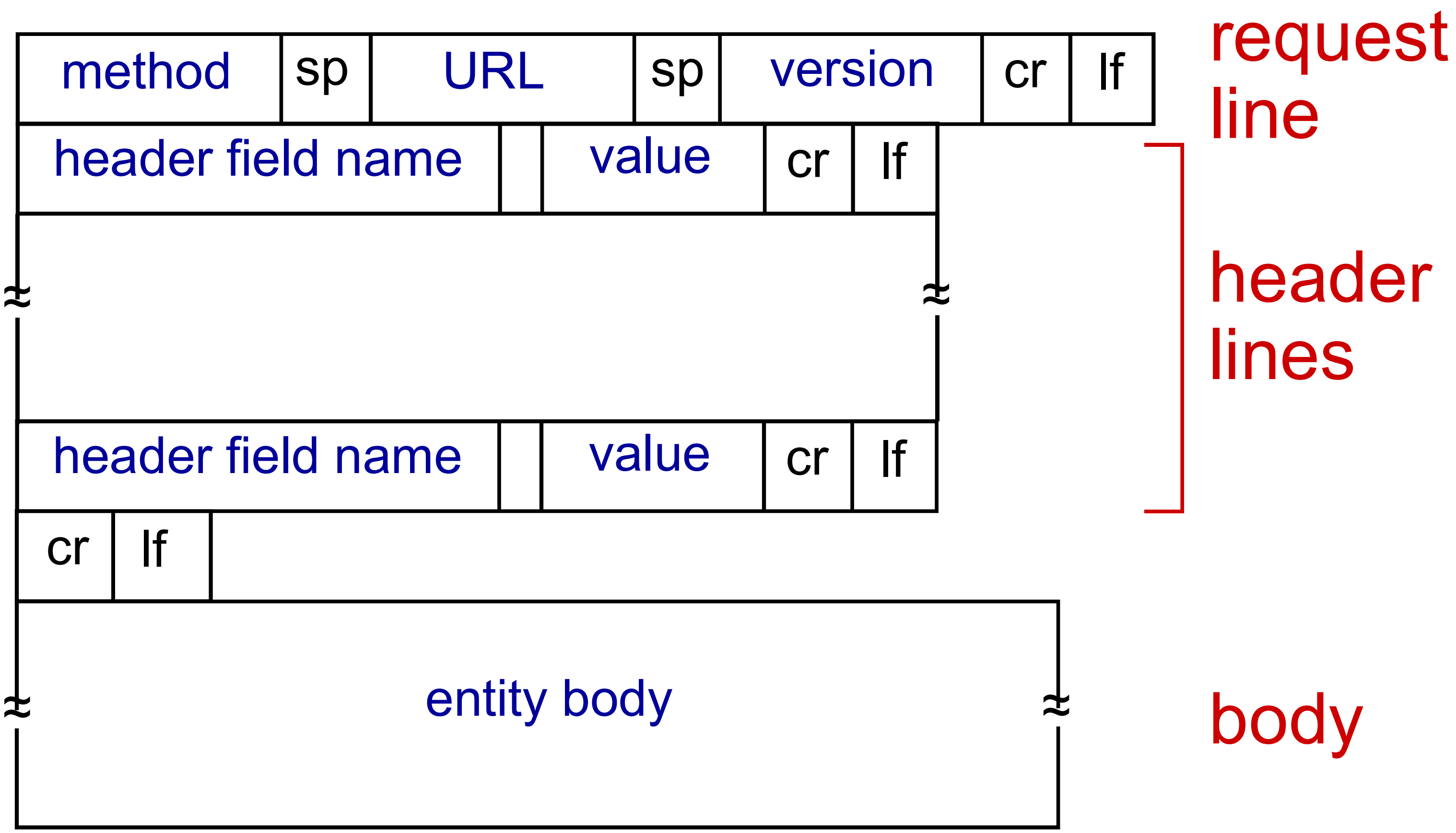
- Requests an object to the host
- Request body is empty
- Information can be encoded in the request path
- Server returns the requested object

POST Method

- Request body contains data to be uploaded to the server
- Used for uploading complex forms and data
- e.g. Facebook Photo Upload

HEAD, PUT, DELETE,…

request message format



HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
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: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

- status code in 1st line in server-to-client response
- Informs about the result of the client request
- codes are predefined, numeric in the form XXX

`200 OK`

request succeeded, requested object embedded in message body

`301 Moved Permanently`

requested object moved, new location specified later in this msg (Location:)

client can send a new request to the updated location

`400 Bad Request`

request msg not understood by server

`404 Not Found`

requested document not found on this server

`505 HTTP Version Not Supported`

State tracking: cookies

Problem:

http is stateless: servers do not keep track of their clients.

This complicates several tasks:

- authorization
- shopping carts
- personalized content (recommendations)
- user session state (Web e-mail)

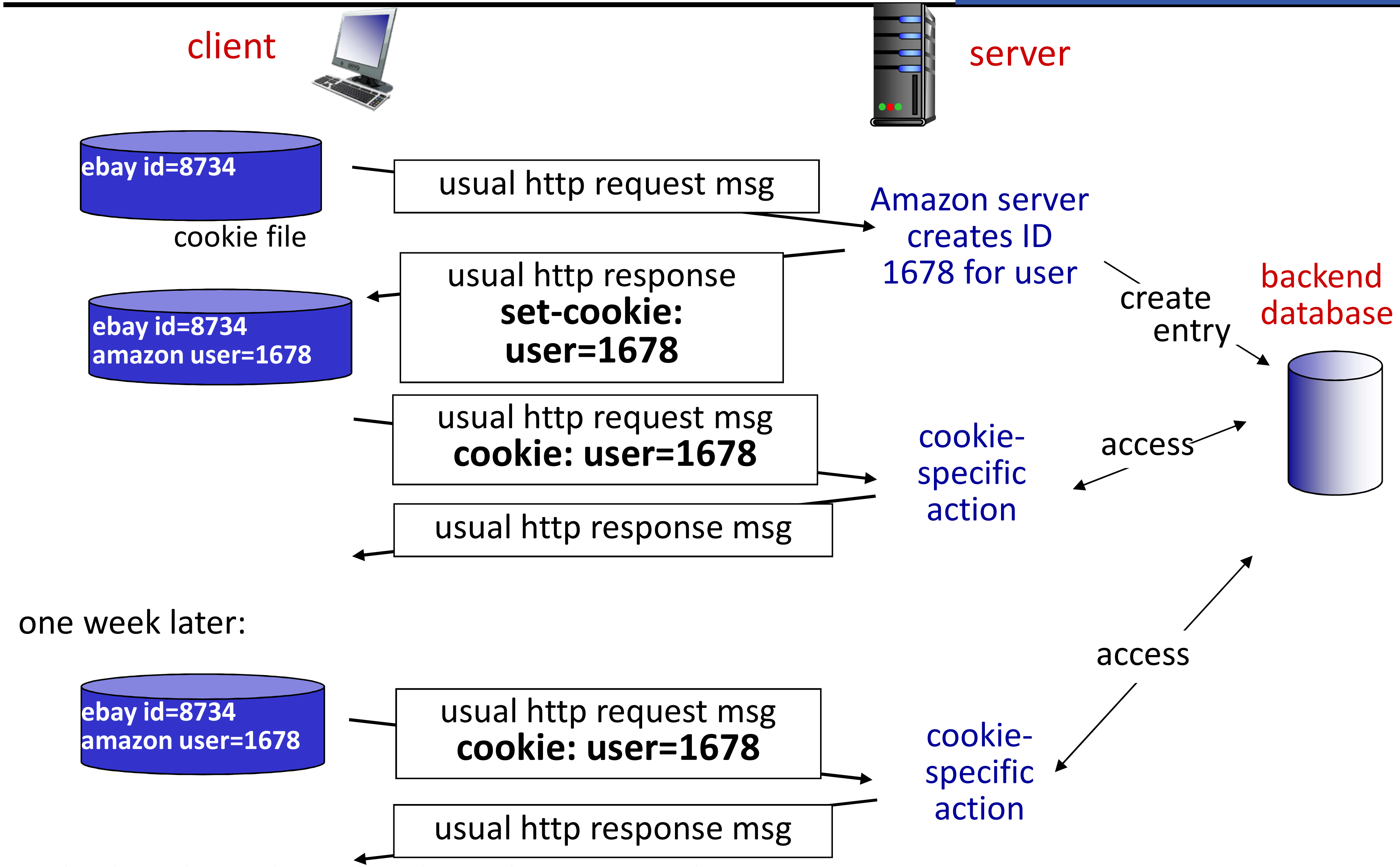
cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- this has forced current UK notification normative

Solution:

- cookies are small pieces of data, provided by the server and stored at client side.
- when a browser contacts a server, a related cookie is sent to the server, after which a page can be displayed relevant to that cookie/client.
- cookies belong to a domain (server), and can have a set expiration date (Expires field)

Cookies: keeping "state"

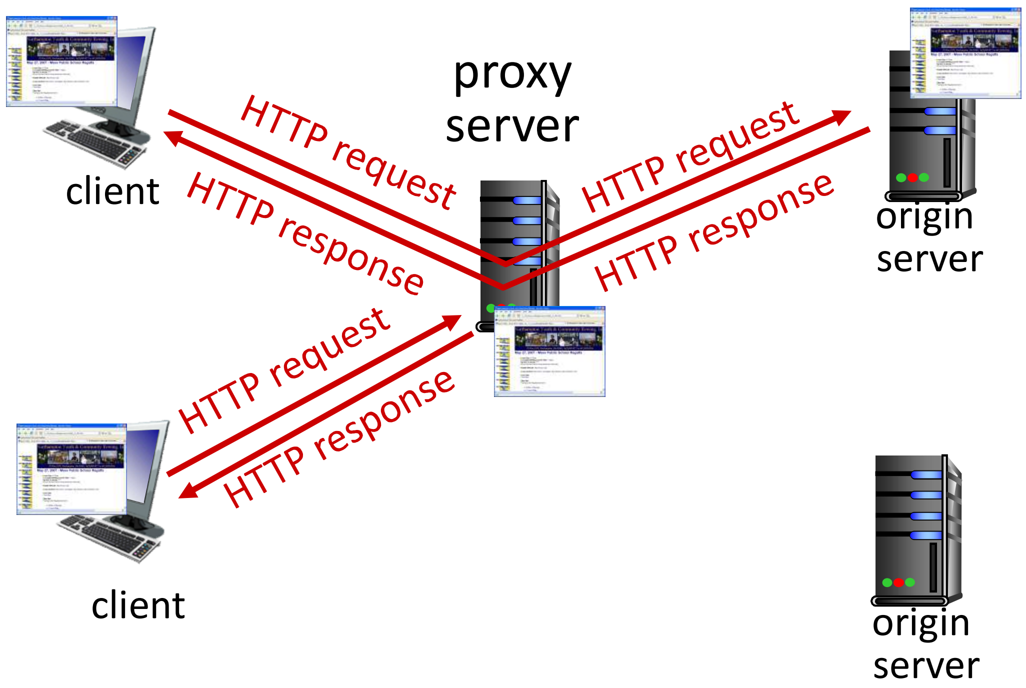


Web caches (proxy server)

cache (proxy server)
broker server to improve efficiency

browser sends all HTTP requests to cache

- object in cache: cache returns object
- else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Conditional GET

Goal: don't send object if cache has up-to-date cached version

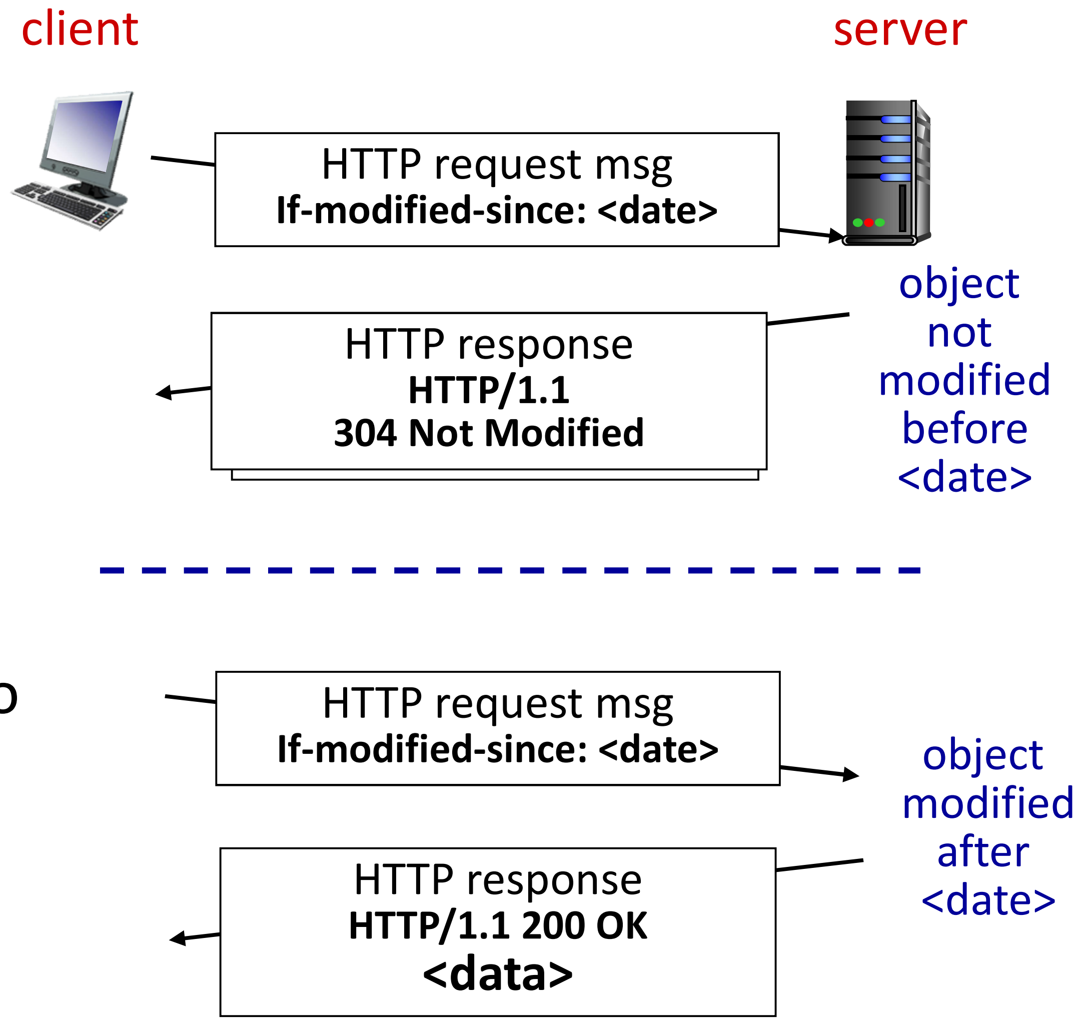
- no object transmission delay
- lower link utilization

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/1.1 304 Not Modified



RESTful applications

- REpresentational State Transfer. Roy Fielding, 2000
- Client-server architecture for distributed applications
- Information is abstracted as resources
  - Resources are stored at the server, they are stateful
  - Resources have identity (can be referenced)
  - Uniform resource access interface
  - Client and server exchange resource representations (eg XML or JSON format)
- Http is perfectly suited for RESTful services

REST Resource Identification

URI: Uniform Resource Identifier

`scheme://host:port/path?queryString#fragment`

A resource is identified by its URI

URLs act as templates

Part of the address represents the type of the resource, and the other part the individual resource

`http://phones.com/whitepages/{userPhone}`

Accept: header specifies the accepted data representation formats (json nowadays)

REST HTTP methods

Clients interact with resources by invoking HTTP methods

Request method dictates what the client wants to do

Method	Meaning	Request Body	Response Body
GET	***Retrieve*** resource representation	Empty	Resource representation
DELETE	***Delete*** the resource	Empty	Empty (or status)
PUT	***Create new*** resource	Proposed representation	Resource representation
POST	Edit/***Update*** resource	Proposed representation	Resource representation

Resource representation (json)

Json has become the de-facto standard

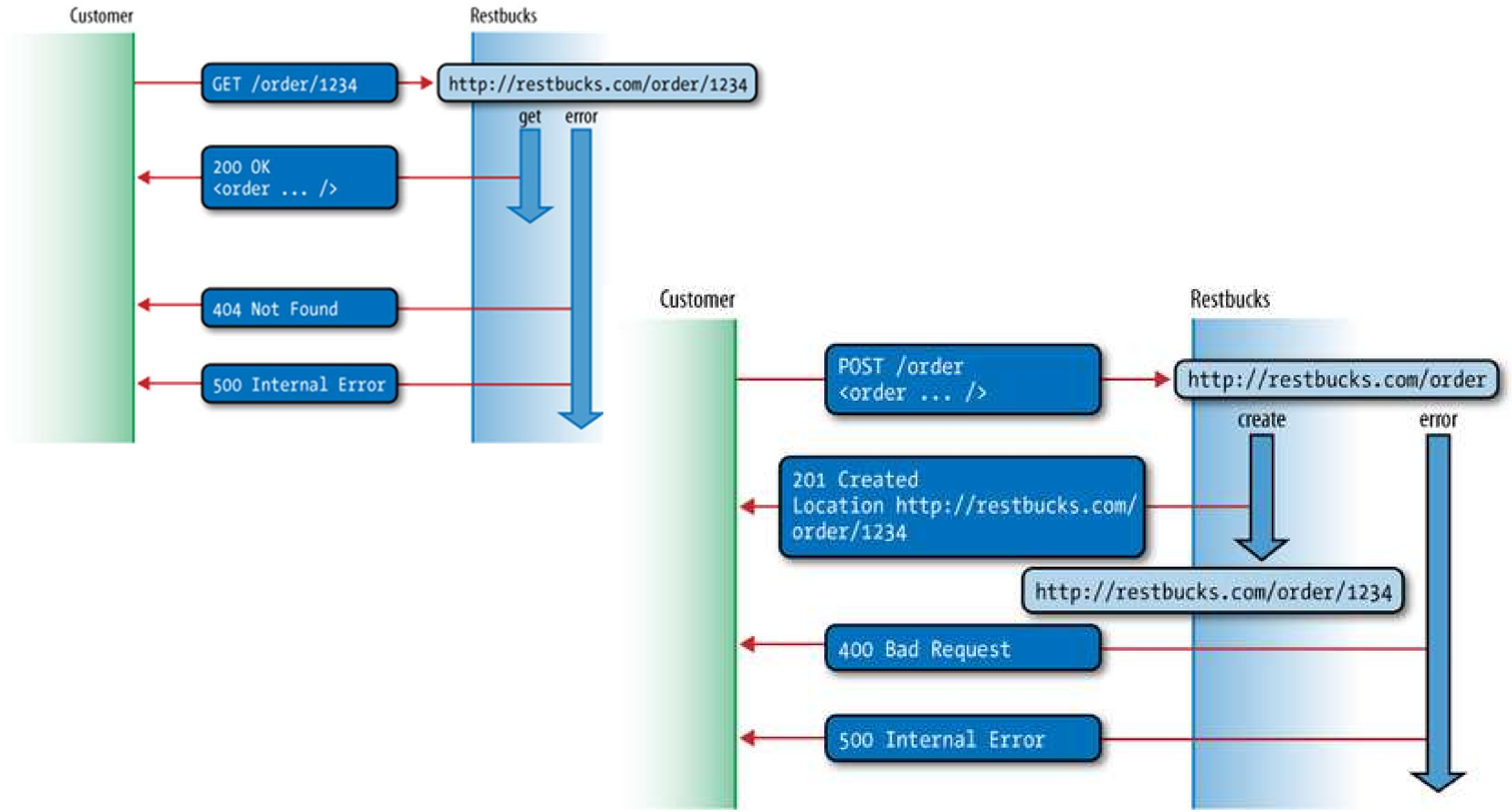
Originally part of javascript, text-like format for describing tree-like structures


```
{ "id":"26007494656", "user_id":"23161357",
"game_id":"417752",
"community_ids":[ "5181e78f-2280-42a6-873d58e25a7c313",
"848d95be-90b3-44a5-b143-6e373754c382", "fd0eab99-832a-4d7e-
8cc0-04d73deb2e54" ],
"type":"live",
"title":"Hey Guys, It's Monday - Twitter: @Lirik",
"viewer_count":32575,
"started_at":"2017-08-14T16:08:32Z",
"language":"en"}, }
```

Example REST API for orders

Method	Meaning
GET */order/{orderId}*	***Requests*** the current status of the order whose identifier is *orderId*
DELETE */order/{orderId}*	***Removes*** the order with the provided *orderId*
PUT */order/{orderId}*	***Updates*** the order with the provided *orderId*. Request body includes updated representation.
POST */order*	***Creates*** new order. Response message contains the id for the newly created resource

REST service interactions



Exchange of HTTP mesages

```
POST /order HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 239
{   "order": {
     "location": "takeAway",
     "items": {
       "item": {
         "name": "latte",
         "quantity": "1",
         "milk": "whole",
         "size": "small"
   }   }   }
}
```

```
HTTP/1.1 201 Created
Content-Length: 267
Content-Type: application/xml
Date: Wed, 26 Sep 2017 08:45:03
Location:
http://restbucks.com/order/1234
{ "order" : { …
```

```
GET /order/1234 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Content-Length: 241
Content-Type: application/xml
Date: Wed, 26 Sep 2017 09:02:10
{   "order": {
     "location": "takeAway",
     "items": {
       "item": {
         "name": "latte",
         "quantity": "1",
         "milk": "whole",
         "size": "small"
   }   }
   "status": delivered }
}
```

Using REST in client applications

It is possible to create a REST client by creating manually http messages, and sending them

… but popularity of REST has led to a myriad of high-level libraries for every framework

In Java, Jackson eases json management

```
public Book[] getAllBooks() throws
Exception {
   HttpResponse<Book[]> response =
Unirest.get(URI_BOOK).asObject(Book[].cl
ass);
   Book[] books = response.getBody();
   return books;
}
```

https://howtoprogram.xyz/2016/07/27/java-rest-client-using-unirest-java-api/

The Application Layer

- Internet applications
- HTTP
- **DNS**
- Content delivery
- P2P

DNS: domain name system

- *People:* many identifiers:
  name, passport #
- *Internet hosts, routers:*
  - IP address (32 bit) - used for addressing datagrams
  - "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing

  canonical, alias names
- mail server aliasing
- load distribution

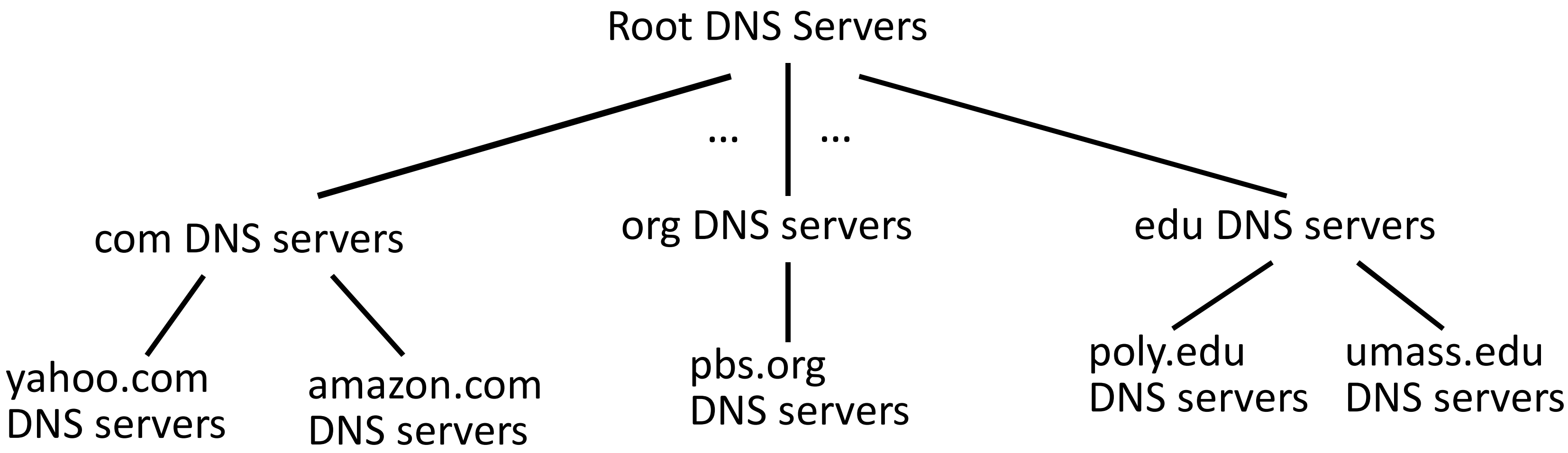
  replicated Web servers: many IP addresses correspond to one name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

DNS: a distributed, hierarchical database

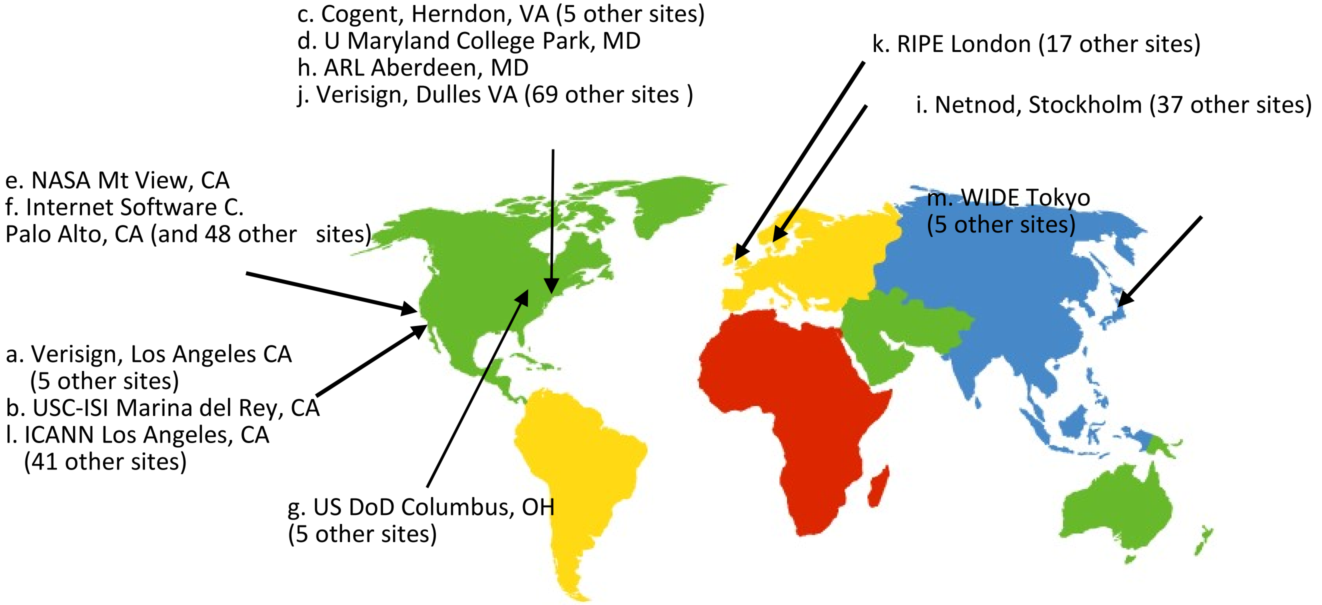


Client wants IP for www.amazon.com; 1st approx:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- Contacted by local resolver that cannot resolve name
- Root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



TLD & authoritative servers

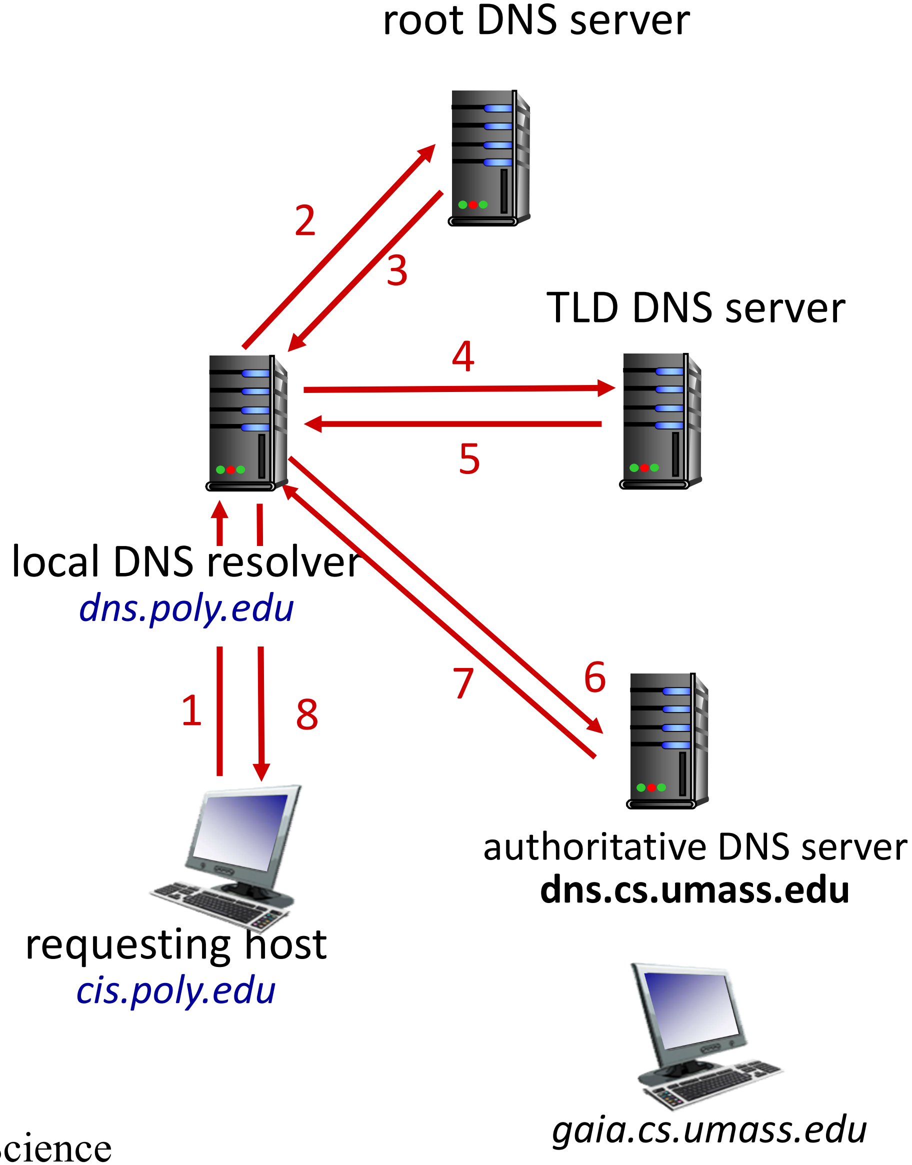
- *Top-level domain (TLD) servers:*
  - responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
  - Network Solutions maintains servers for .com TLD
  - Educause for .edu TLD
- *Authoritative DNS servers:*
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
  - can be maintained by organization or service provider

Local DNS resolver

- Client, not part of DNS hierarchy
- Each ISP (residential ISP, company, university) has one
  - also called "default resolver"
- When host makes DNS query, query is sent to its local DNS resolver
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as a proxy, forwards query into DNS hierarchy
- 3rd party resolvers
  - OpenDNS
  - GoogleDNS

DNS name resolution example

host at cis.poly.edu wants IP address for gaia.cs.umass.edu



Iterative query:

- Contacted server replies with name of server to contact (referral) = "I don't know this name, but ask this server"
- Resolver iterates until it finds the answer

DNS caching

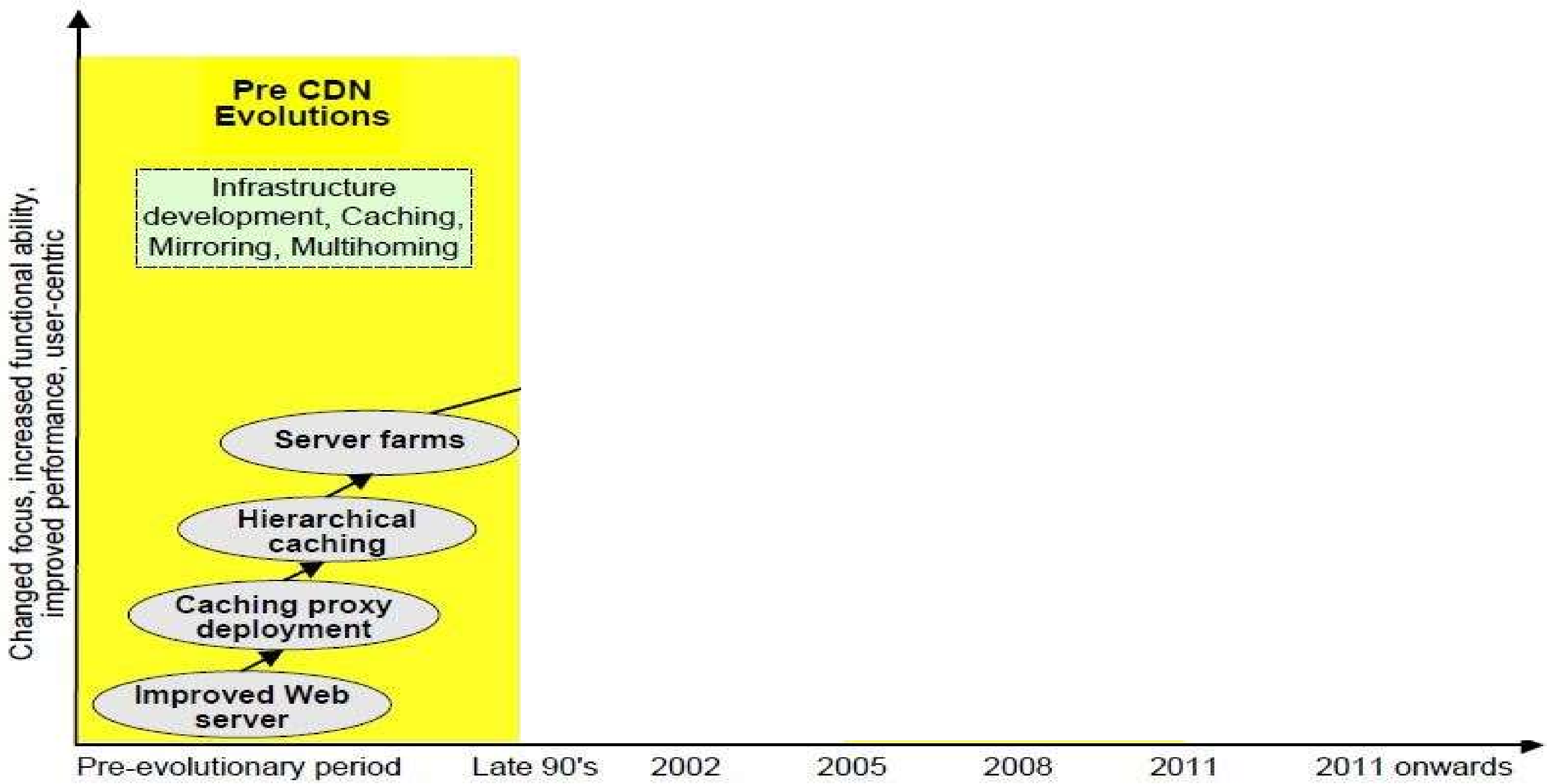
- Once (any) name server/resolver learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers

    => root name servers not often visited
- Cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire

Agenda

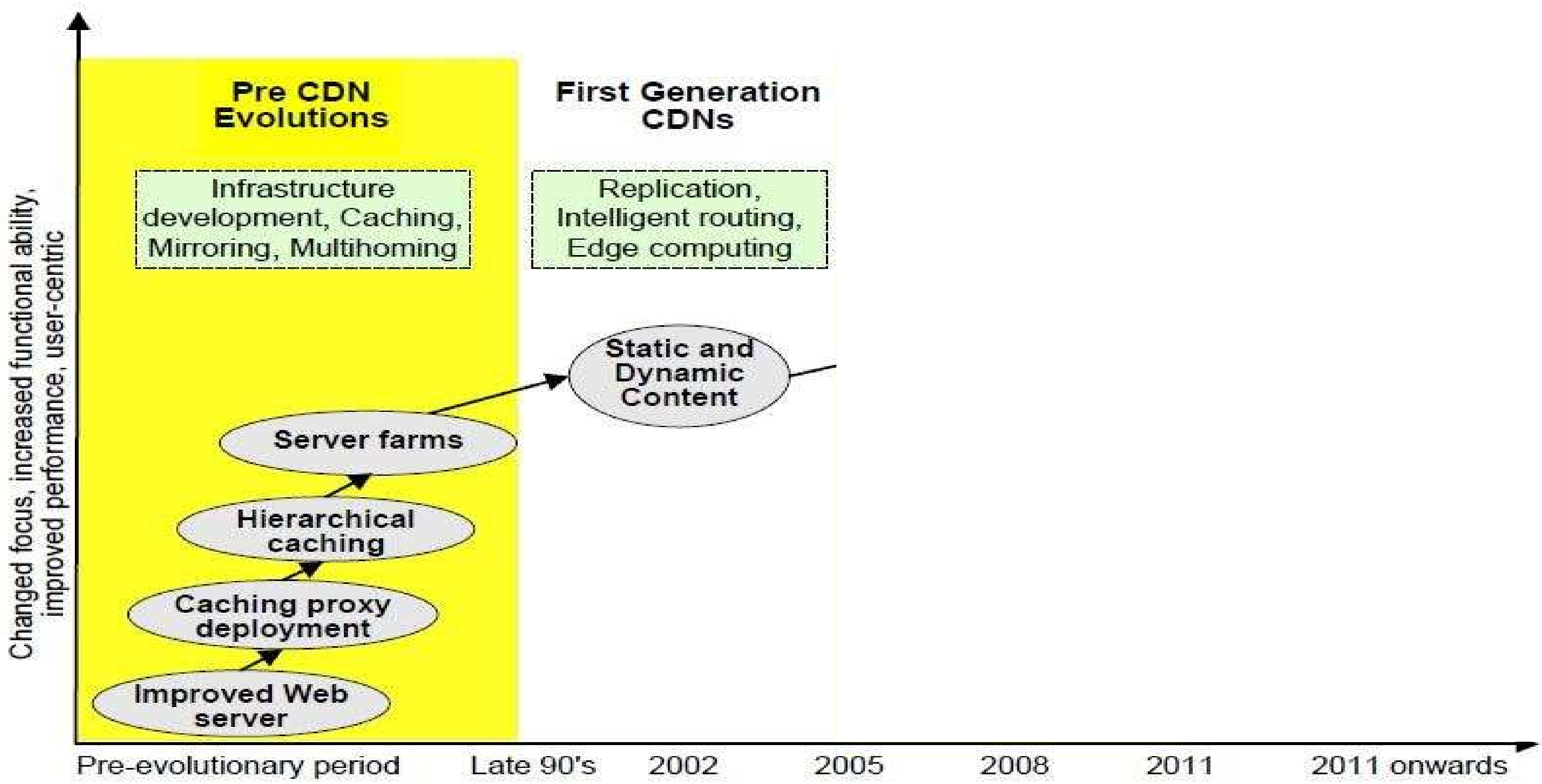
- Internet applications
- HTTP
- DNS
- **Content delivery**
- P2P

The Early Web



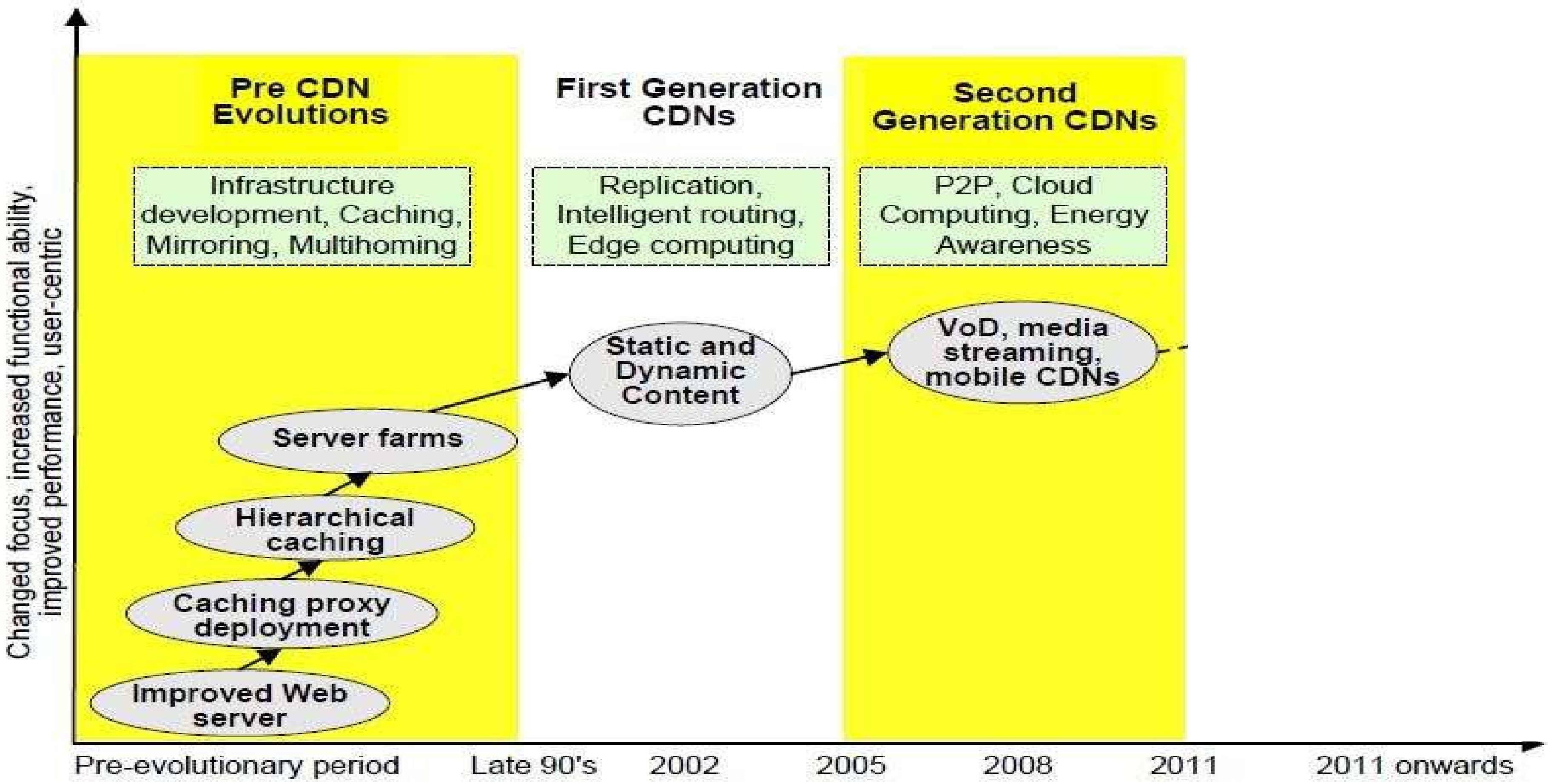
Pathan Mukaddim. Ongoing Trends and Future Directions in Content Delivery Networks (CDNs). Available online from: http://amkpathan.wordpress.com/article/ongoing-trends-and-future-directions-in-3uxfz2buz8z1w-2/

CDNs 1.0



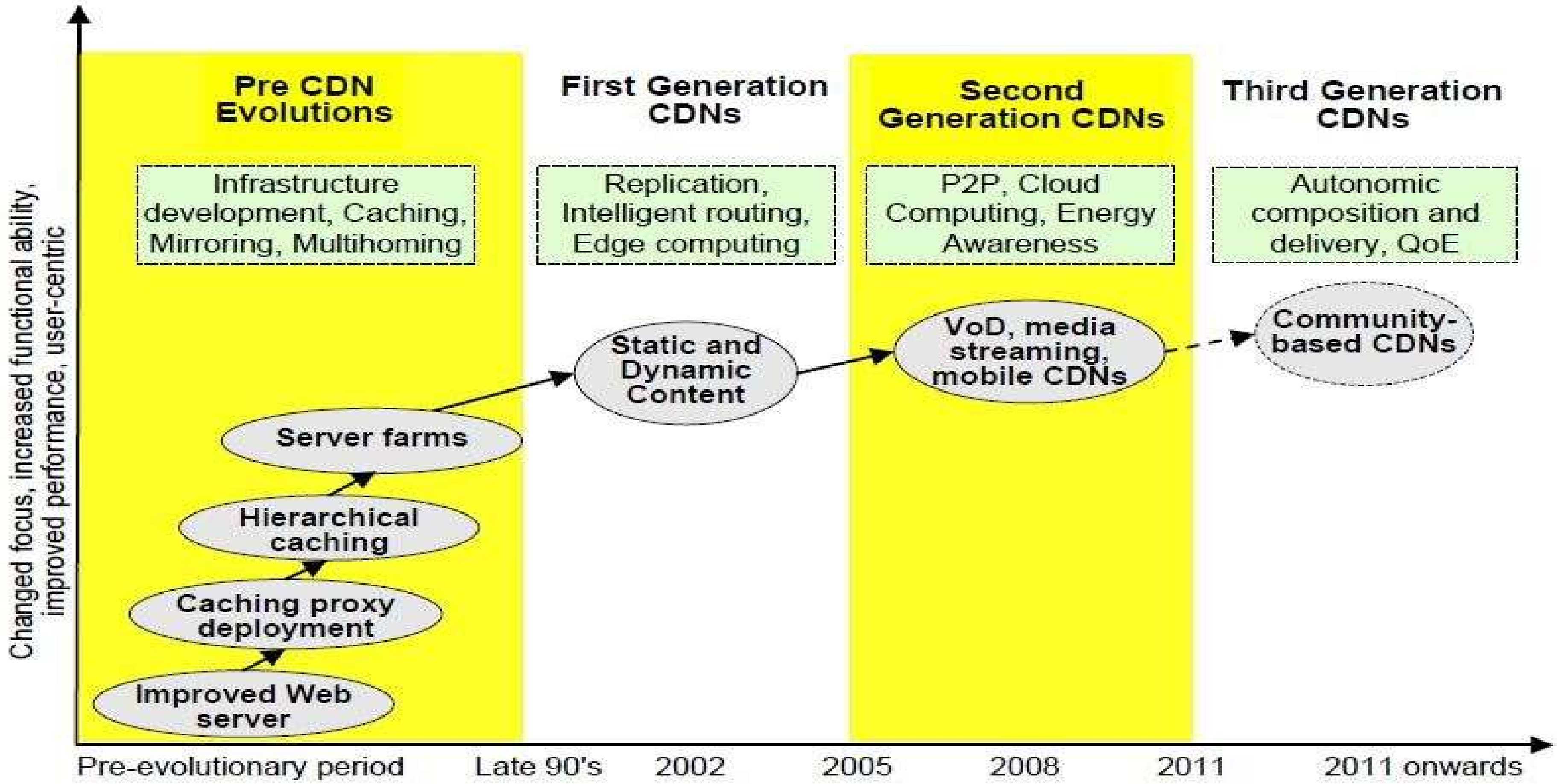
Pathan Mukaddim. Ongoing Trends and Future Directions in Content Delivery Networks (CDNs). Available online from: http://amkpathan.wordpress.com/article/ongoing-trends-and-future-directions-in-3uxfz2buz8z1w-2/

CDNs 2.0



Pathan Mukaddim. Ongoing Trends and Future Directions in Content Delivery Networks (CDNs). Available online from: http://amkpathan.wordpress.com/article/ongoing-trends-and-future-directions-in-3uxfz2buz8z1w-2/

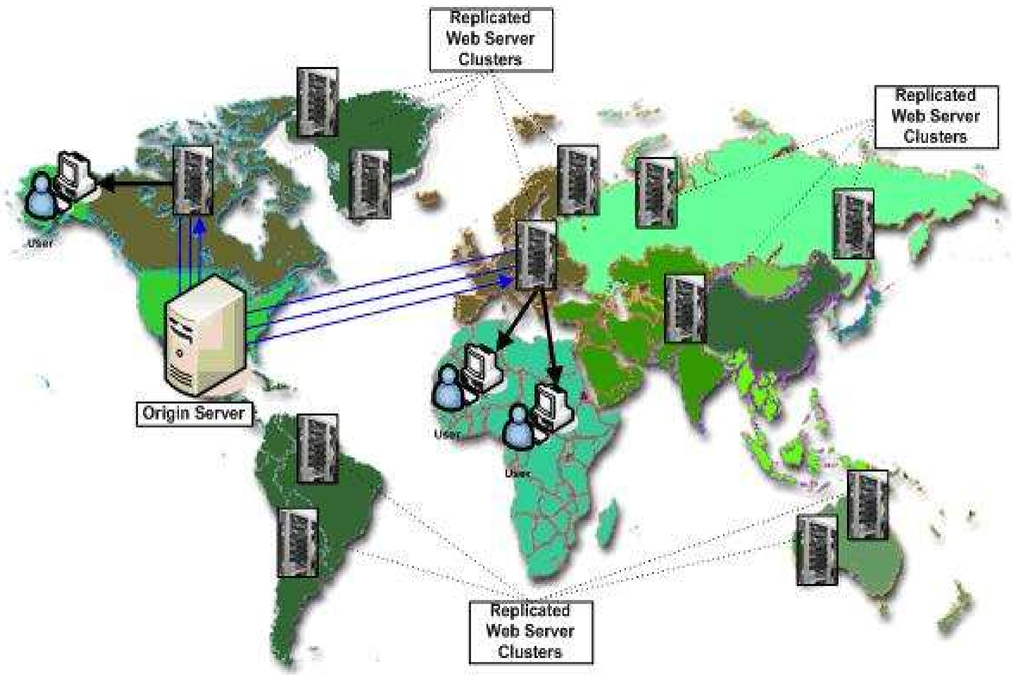
Autonomic CDNs



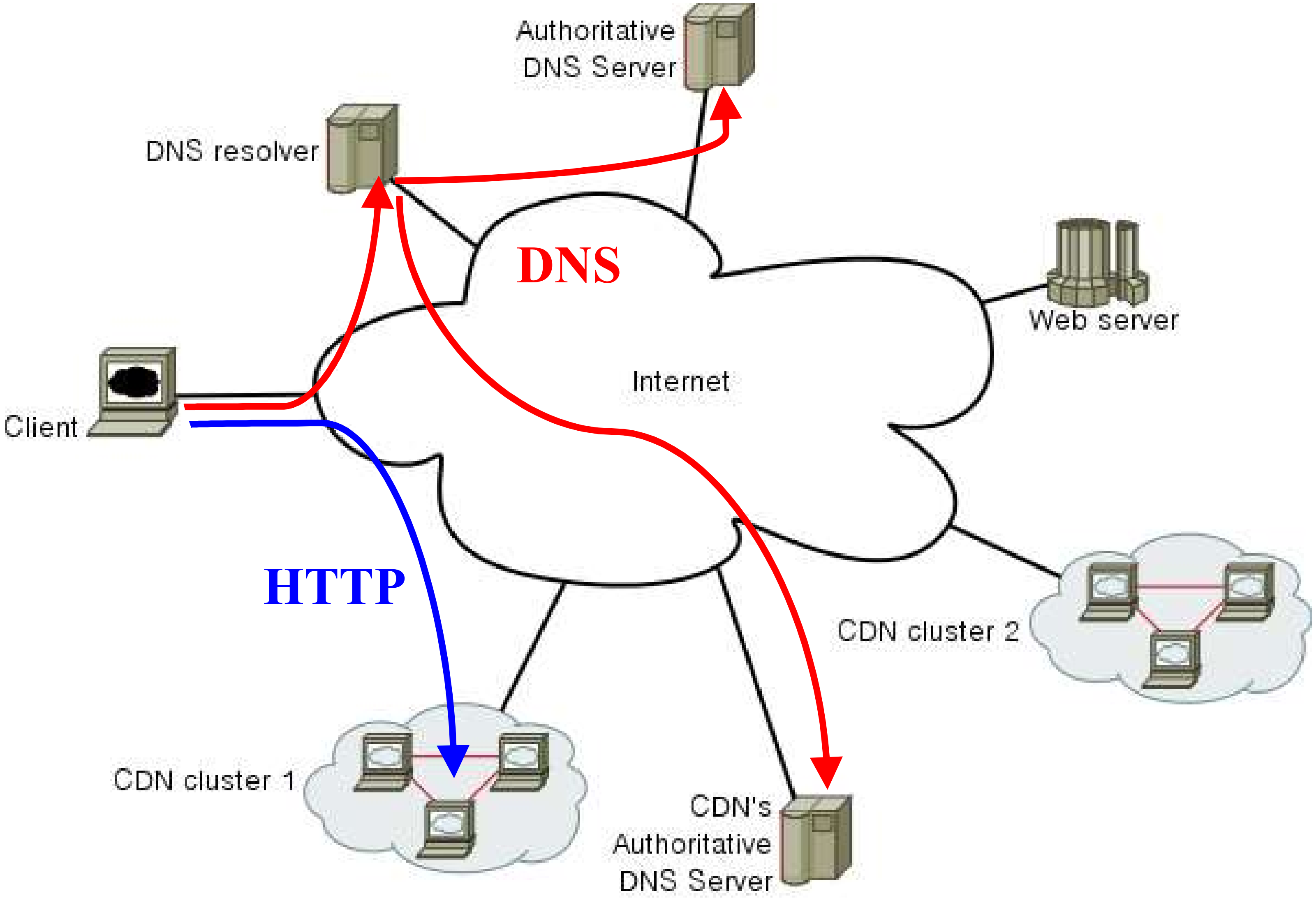
Pathan Mukaddim. Ongoing Trends and Future Directions in Content Delivery Networks (CDNs). Available online from: http://amkpathan.wordpress.com/article/ongoing-trends-and-future-directions-in-3uxfz2buz8z1w-2/

The CDN Monsters

- Servers deployed around the world
- Serve the same content from multiple (all) locations
- Examples:
  - Akamai
  - Google
  - Netflix
  - Limelight
  - Amazon CloudFront
  - Level 3
  - Windows Azure



HTTP server selection



CDNs and CNAME

- DNS redirection: CNAME = canonical name
- Popular for CDNs, e.g., Akamai

```
; <<>> DiG 9.8.3-P1 <<>> www.whitehouse.gov
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11921
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;www.whitehouse.gov.            IN      A
;; ANSWER SECTION:
www.whitehouse.gov.     3596    IN      CNAME      www.whitehouse.gov.edgesuite.net.
www.whitehouse.gov.edgesuite.net. 896 IN CNAME     www.eop-edge-lb.akadns.net.
www.eop-edge-lb.akadns.net. 300 IN      CNAME      a1128.dsch.akamai.net.
a1128.dsch.akamai.net.  20      IN      A     195.59.54.235
a1128.dsch.akamai.net.  20      IN      A     195.59.126.161
;; Query time: 43 msec
;; SERVER: 138.37.0.88#53(138.37.0.88)
;; WHEN: Fri Oct 10 10:49:15 2014
;; MSG SIZE  rcvd: 183
```

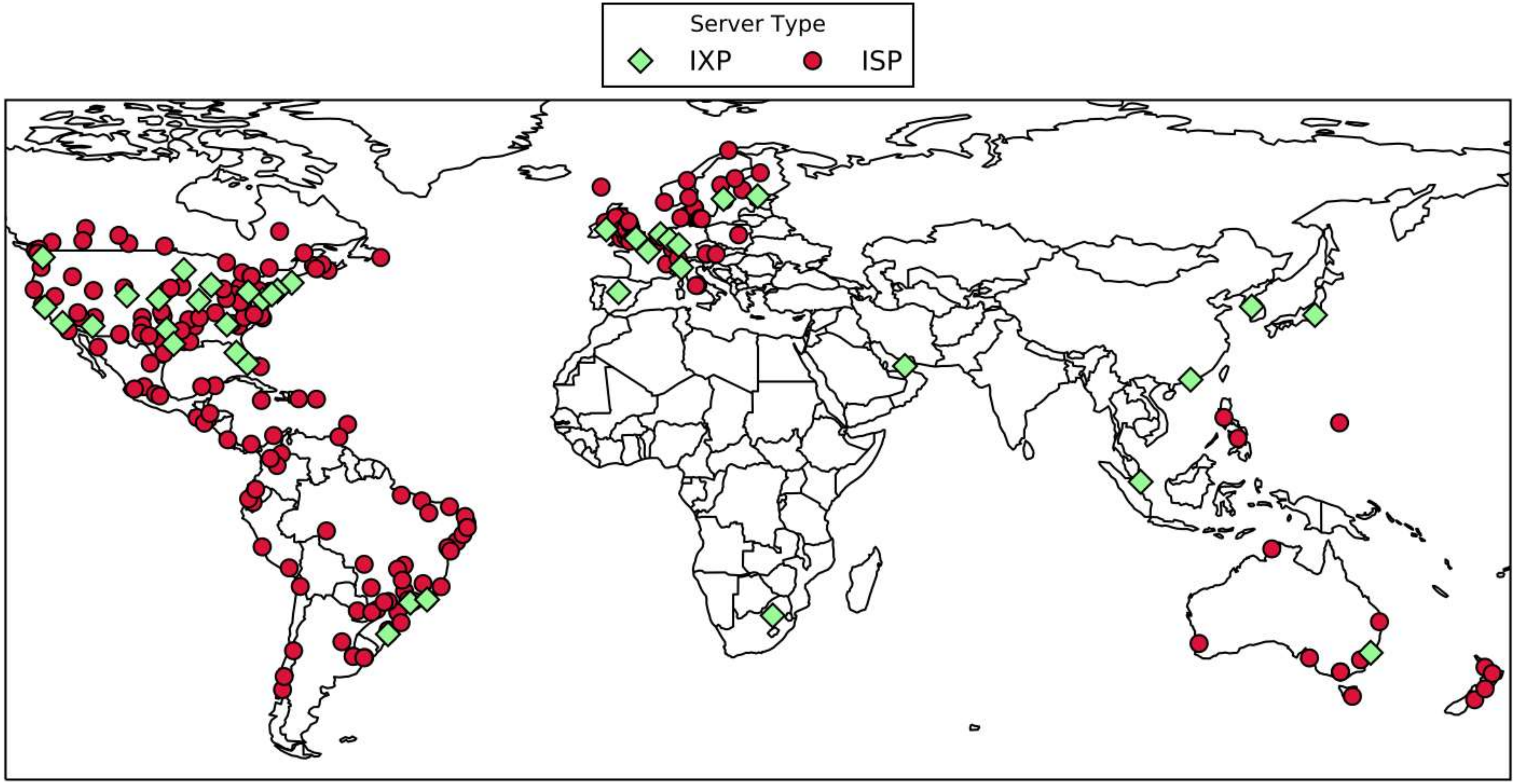
World data centers

http://www.datacentermap.com/

Google data centers

http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/

Netflix



Where is popular Web content hosted?

- California
- China 2nd
- USA: 9 among top 20
- Other developed countries: limited own content

Rank	Country	Potential	Normalized potential
1	USA (CA)	0.254	0.108
2	China	0.128	0.107
3	USA (TX)	0.190	0.061
4	Germany	0.183	0.058
5	Japan	0.163	0.051
6	France	0.146	0.034
7	Great Britain	0.157	0.030
8	Netherlands	0.144	0.029
9	USA (WA)	0.135	0.027
10	USA (unknown)	0.164	0.027
11	Russia	0.038	0.027
12	USA (NY)	0.130	0.026
13	Italy	0.122	0.018
14	USA (NJ)	0.125	0.016
15	Canada	0.028	0.015
16	USA (IL)	0.116	0.014
17	Australia	0.118	0.013
18	Spain	0.116	0.013
19	USA (UT)	0.111	0.012
20	USA (CO)	0.113	0.012

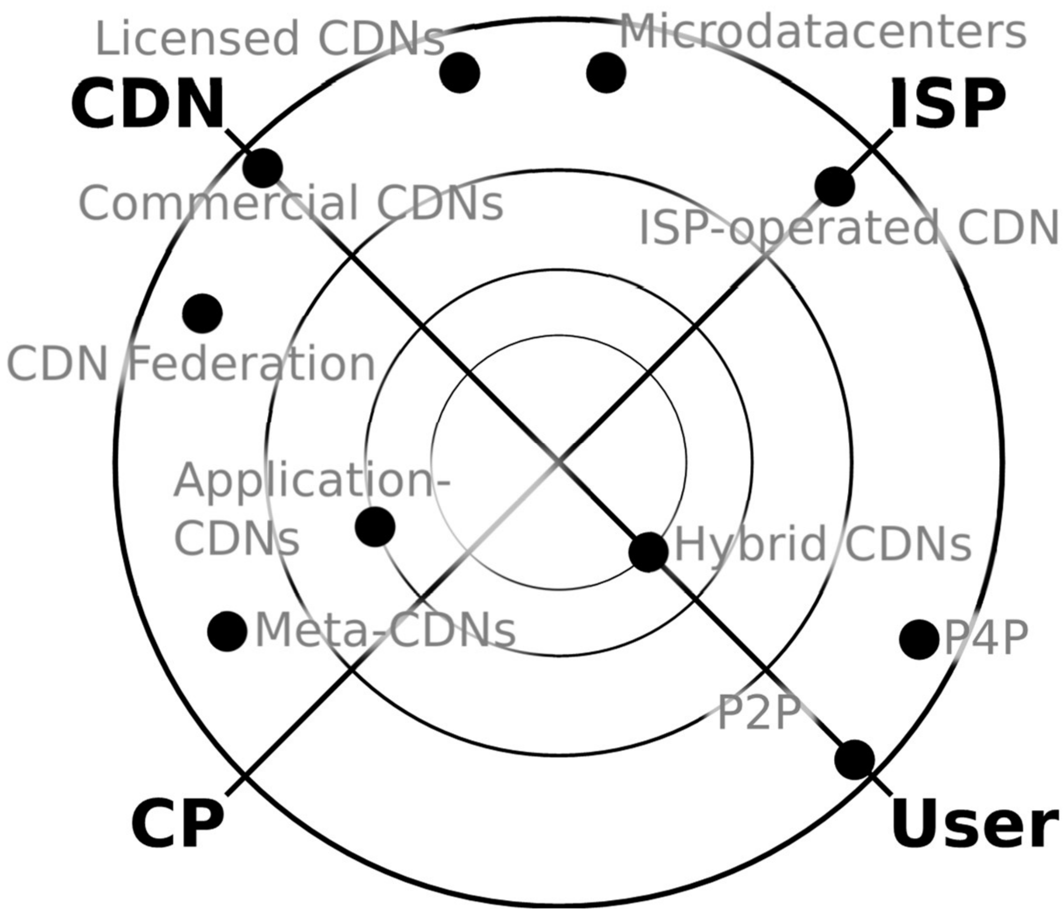
Top western Web CDNs

Rank	# hostnames	# ASes	# prefixes	owner	content mix
1	476	79	294	Akamai	
2	161	70	216	Akamai	
3	108	1	45	Google	
4	70	35	137	Akamai	
5	70	1	45	Google	
6	57	6	15	Limelight	
7	57	1	1	ThePlanet	
8	53	1	1	ThePlanet	
9	49	34	123	Akamai	
10	34	1	2	Skyrock	
11	29	6	17	Cotendo	
12	28	4	5	Wordpress	
13	27	6	21	Footprint	
14	26	1	1	Ravand	
15	23	1	1	Xanga	
16	22	1	4	Edgecast	
17	22	1	1	ThePlanet	
18	21	1	1	ivwbox.de	
19	21	1	5	AOL	
20	20	1	1	Leaseweb	

only on TOP, both on TOP and EMBEDDED, only on EMBEDDED, TAIL.

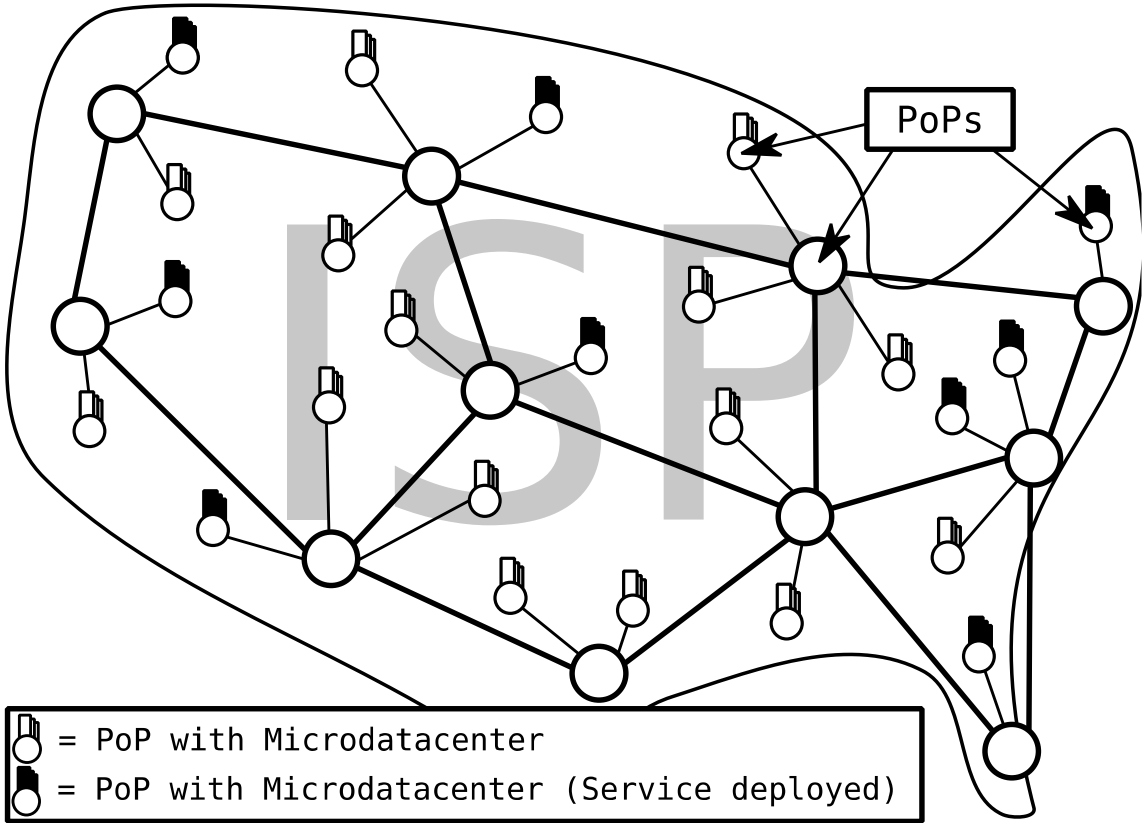
Collaborative content delivery

- Importance of stakeholders
- Spectrum in the solution space



CDN 2.0

- Hybrid infrastructures: Akamai, PPTV
- Meta-CDNs, e.g., Conviva
- Virtual CDNs through ISP micro-datacenters

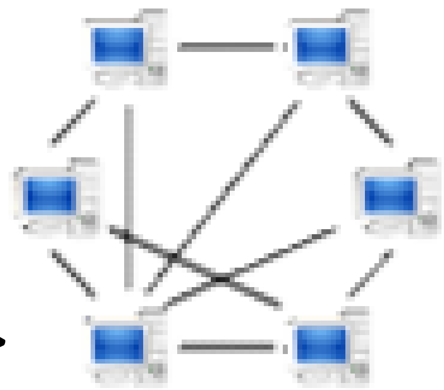


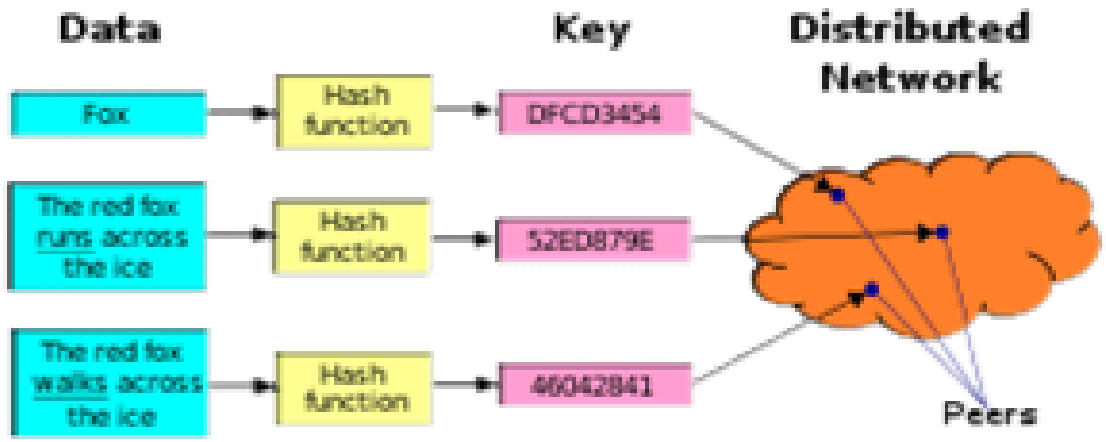
The Application Layer

- Internet applications
- HTTP
- DNS
- Content delivery
- **P2P**

P2P

- Alternative to client-server model
- Each node has the same role
- Structured or unstructured
- Often built on DHT
  - Content stored as *<hash,value pair>*
  - Locating the data based on hash
  - Content replicated across P2P network
- Examples:
  - Gnutella
  - eDonkey
  - Bittorrent

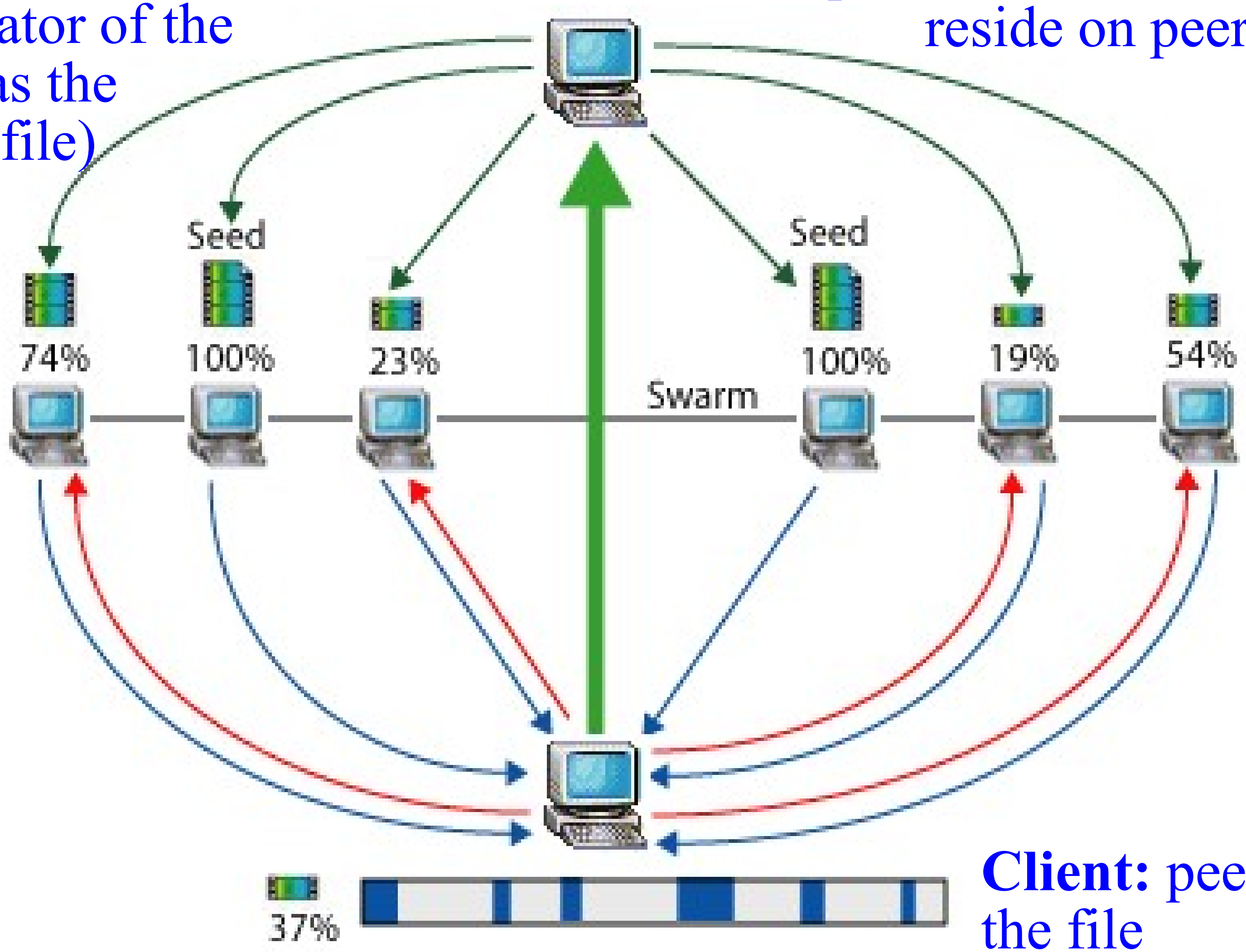




P2P: Server selection

Seed:
Originator of the file (has the whole file)

Tracker: keeps track of where file copies reside on peer machines



Swarm: peers that have at least one chunk

Client: peer who wants the file

P2P: pro's and con's

- Pro's:
  - ✓ Easy to share content
  - ✓ Anonymous thanks to DHT
  - ✓ Storage scales
- Con's
  - \- Reliability (nodes uptime), e.g., skype
  - \- Lack of visibility of content (copyrighted material)
  - \- P2P structure dictates where content is replicated

The "P2P CDN"

#	Country	U. Peers
1	US	1379462
2	ES	887480
3	GB	800308
4	CA	551820
5	IN	514246
6	AU	322009
7	BR	318294
8	IT	295339
9	PL	288780
10	PT	220124

TOP 10 ISPs (BT VIDEO USER)

	AS#	Peers	AS Name-Internet Service Provider
1	3352	165469	TELEFONICA-DATA-ESPANA(TDE)
2	3662	129047	DNEO-OSP7-COMCAST CABLE
3	6461	127297	MFNX MFN-METROMEIDA FIBER
4	2119	113597	TELENOR-NEXTEL T.NET
5	19262	101390	VZGNI-TRANSIT-Verizon ISP
6	3301	97658	TELIANET-SWEDEN TELIANET
7	3462	96564	HINET-DATA CBG
8	4134	87392	CHINANET-BACKBONE
9	6327	86964	SHAW-SHAW COMMUNICATION
10	174	74453	COGENT COGENT/PSI

P2P popularity