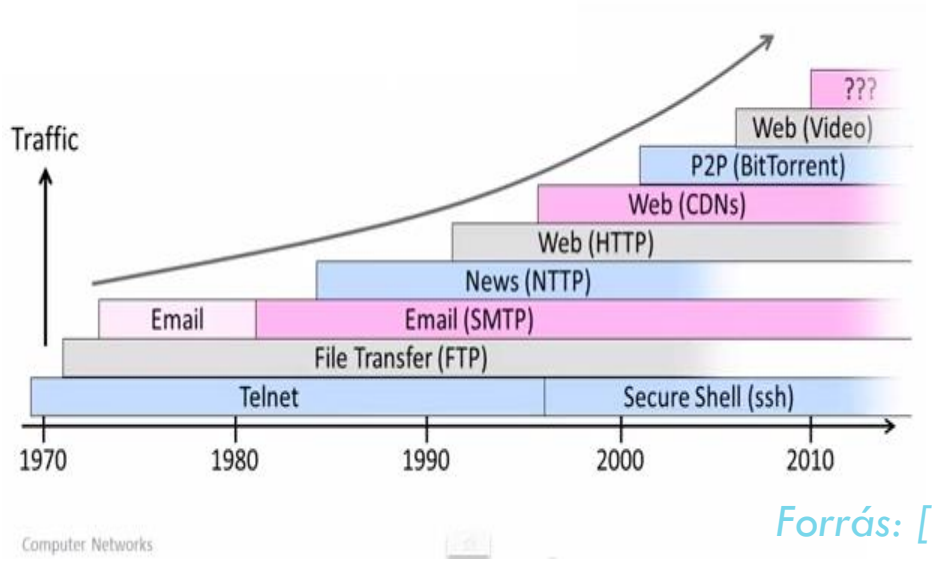
Számítógépes Hálózatok

11. Előadás: Alkalmazási réteg
DNS, HTTP, CDN

Based on slides from **Zoltán Ács ELTE** and D. Choffnes Northeastern U., Philippa Gill from StonyBrook University , Revised Spring 2016 by S. Laki

Internetes alkalmazások evolúciója



Forrás: [1]

- Folyamatosan változik, növekszik…
  - www.evolutionoftheweb.com

DNS

„8. réteg” (A szénalapú csomópontok)

- Ha…
  - Fel szeretnél hívni valakit, akkor el kell kérned a telefonszámát
    - Nem hívhatod csak úgy “P I S T Á T”
  - Levelet küldenél valakinek, akkor szükséged van a címére
- Mi a helyzet az Internettel?
  - Ha el akarod érni a Google-t, szükséges annak IP címe
  - Tudja valaki a Google IP címét???
- A probléma bennünk van:
  - Az emberek nem képesek IP címek megjegyzésére
  - Ember számára értelmes nevek kellenek, melyek IP címekre képezhetők

Internetes nevek és címek

- Címek, pl. 129.10.117.100
  - Számítógépek által használt címkék a gépek azonosítására
  - A hálózat szerkezetét tükrözi
- Nevek, pl. www.northeastern.edu
  - Ember számára értelmes címkék a gépeknek
  - A szervezeti struktúrát tükrözi
- Hogyan képezzünk az egyikről a másikra?
  - Domain Name System (DNS)

Réges régen…

- A DNS előtt minden név-IP leképezés egy *hosts.txt*-ben volt
  - */etc/hosts* - Linuxon
  - C:*Windows**System32**drivers**etc**hosts* - Windowson
- Központosított, manuális rendszer
  - A változásokat emailben kellett beküldeni a SRI-nek
    - SRI=Stanford Research Institute
  - A gépek periodikus időközönként letöltötték (FTP) a *hosts.txt* fájlt
  - Minden név megengedett volt – nem volt benne hierarchia („flat” (sík) felépítés)
    - alans_server_at_sbu_pwns_joo_lol_kthxbye

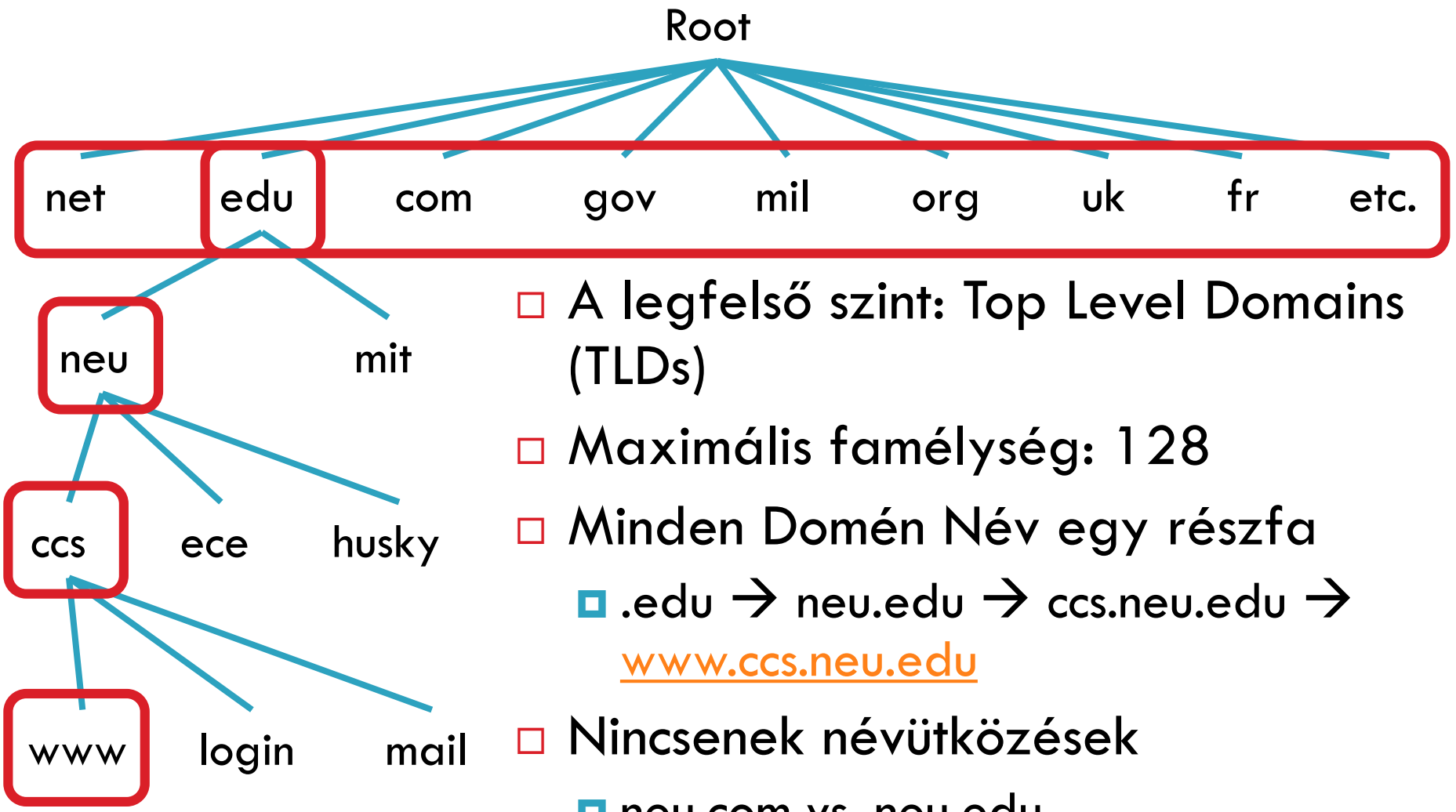
A DNS felé

- Végül a *hosts.txt* alapú rendszer szétesett
  - Nem skálázható, SRI nem bírt a terheléssel/igényekkel
  - Nehéz volt a nevek egyediségének biztosítása
    - Pl. MIT
      - Massachusetts Institute of Technology?
      - Melbourne Institute of Technology?
  - Számos gép rendelkezett nem naprakész *hosts.txt*-vel
- Ez vezetett a DNS megszületéséhez…

DNS általánosságban

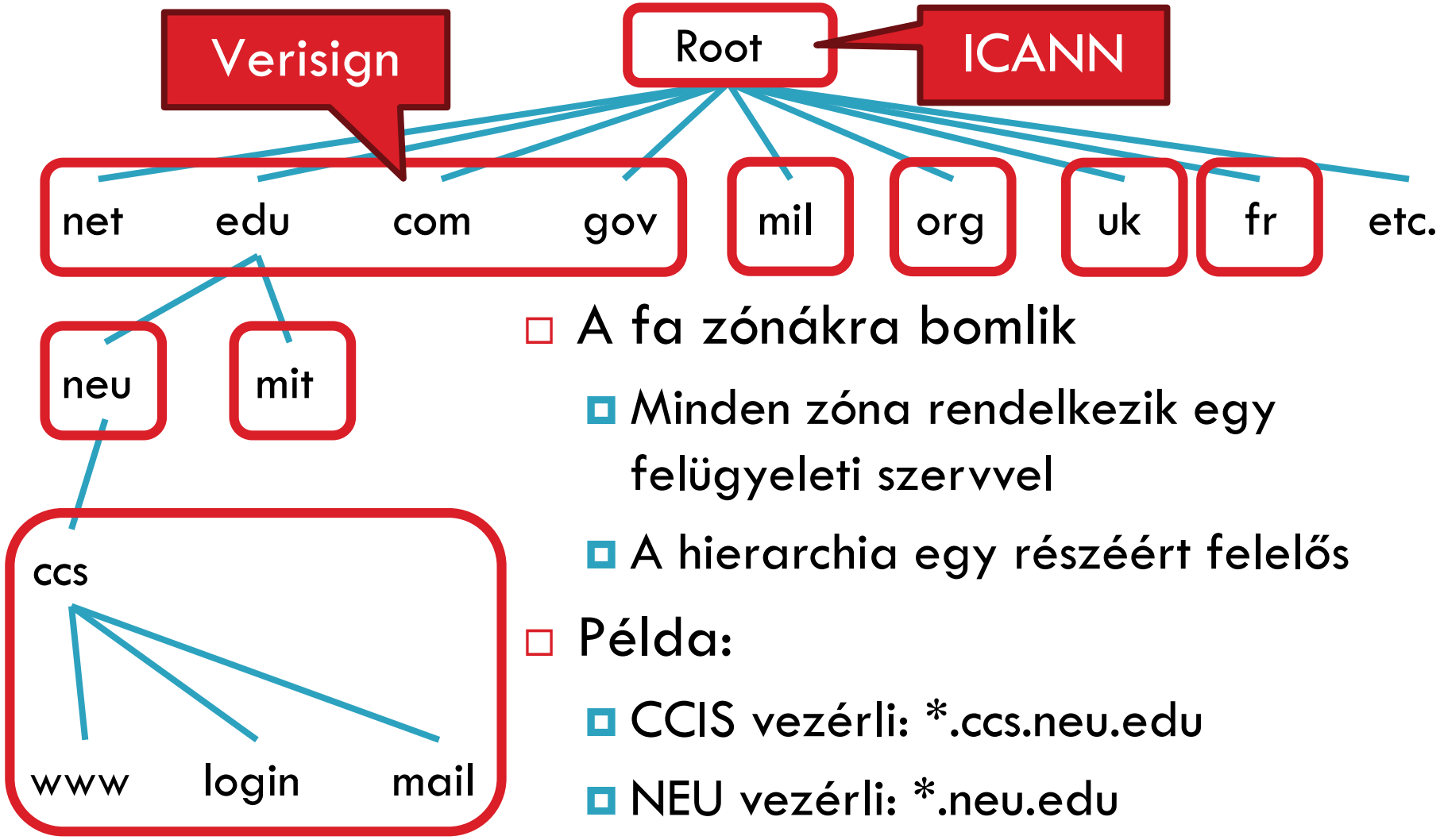
- Domain Name System
- Elosztott adatbázis
  - Nem központosított
- Egyszerű kliens-szerver architektúra
  - UDP 53-as port, vannak TCP implementációk is
  - Rövid kérések – rövid válaszok; kérés-válasz típusú kommunikáció
- Hierarchikus névtér
  - Szemben a hosts.txt alapú flat megoldással
  - pl. .com → google.com → mail.google.com

Név hierarchia



- A legfelső szint: Top Level Domains (TLDs)
- Maximális famélység: 128
- Minden Domén Név egy részfa
  - .edu → neu.edu → ccs.neu.edu → www.ccs.neu.edu
- Nincsenek névütközések
  - neu.com vs. neu.edu

Hierarchikus adminisztráció



- A fa zónákra bomlik
  - Minden zóna rendelkezik egy felügyeleti szervvel
  - A hierarchia egy részéért felelős
- Példa:
  - CCIS vezérli: *.ccs.neu.edu
  - NEU vezérli: *.neu.edu

Szerver hierarchia

- Egy DNS szerver funkciói:
  - A hierarchia egy részét felügyeli
    - Nem szükséges minden DNS nevet tárolnia
  - A zónájához tartozó összes hoszt és domén rekordjainak tárolása
    - Másolatok lehetnek a robosztusság növelés végett
  - Ismeri a root szerverek címét
    - Ismeretlen nevek feloldása miatt kell
- A root szerverek minden TLD-t ismernek
  - Azaz innen indulva fel lehet tárni…

Top Level Domains

- **I**nternet **C**orp. **A**ssigned **N**ames and **N**umbers (1998)
- 22+ **általános *TLDs*** létezik
  - klasszikusok: *.com*, *.edu*, *.gov*, *.mil*, *.org*, *.net*
  - később keletkeztek: *.aero*, *.museum*, *.xxx*
- ~250 TLDs a különböző **ország kódok**nak
  - Két betű (mint például .au, .hu), 2010-től plusz nemzetközi karakterek (például kínai)
  - Több elüzletisedett, például a .tv (Tuvalu)
  - Példa domén hack-ekre: instagr.am (Örményország), goo.gl (Grönland)

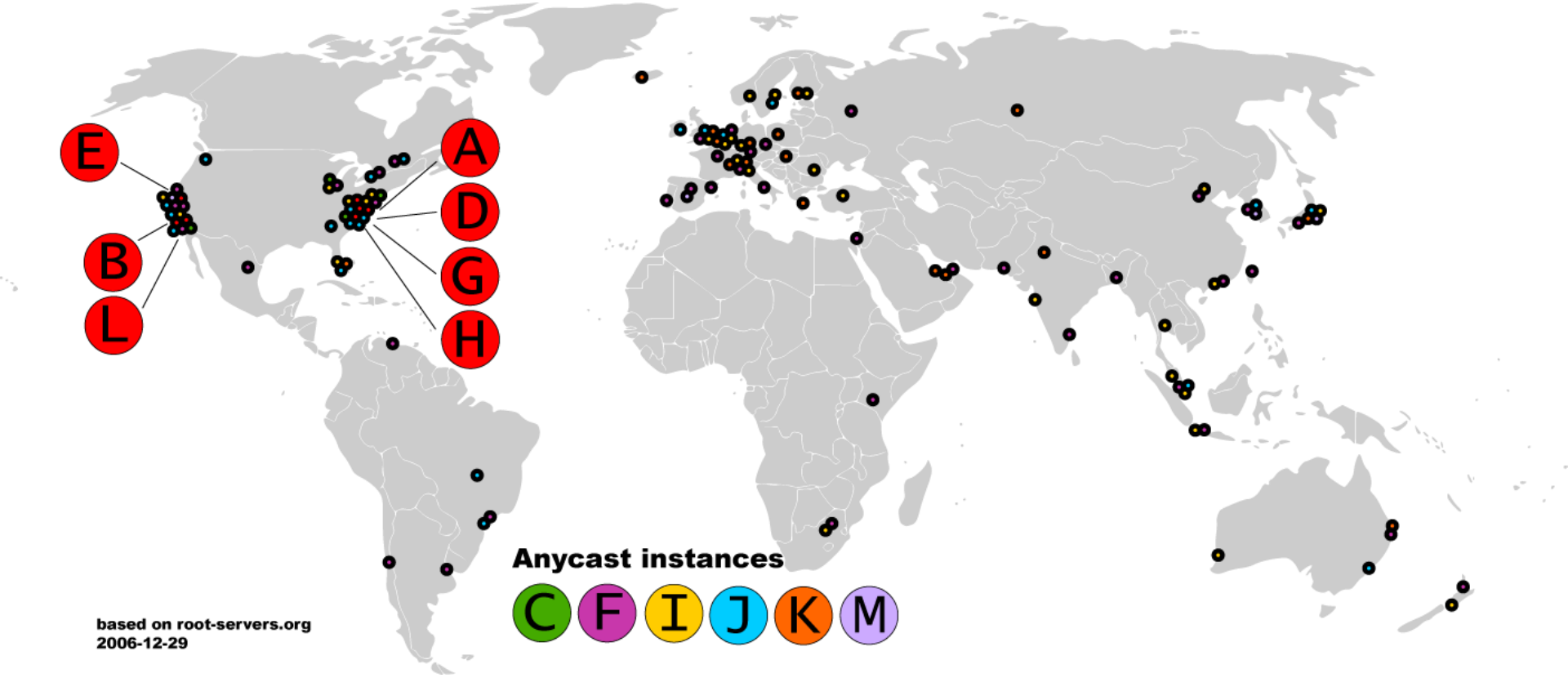
Root Name Servers

- A Root Zone Fájlért felelős
  - Listát vezet a TLD-kről és arról, hogy ki felügyeli őket.
  - ~272KB a fájl mérete
  - Pl. bejegyzése:

com.	172800	IN	NS	a.gtld-servers.net.
com.	172800	IN	NS	b.gtld-servers.net.
com.	172800	IN	NS	c.gtld-servers.net.

- Az ICANN adminisztrálja
  - 13 root szerver, címkék: A→M
  - Pl.: i.root-servers.net
  - 6 db ezek közül „anycastolt”, azaz globálisan számos replika létezik
- Ha név nem feloldható (lokálisan), akkor hozzájuk kell fordulni
  - A gyakorlatban a legtöbb rendszer lokálisan tárolja ezt az információt (cache)

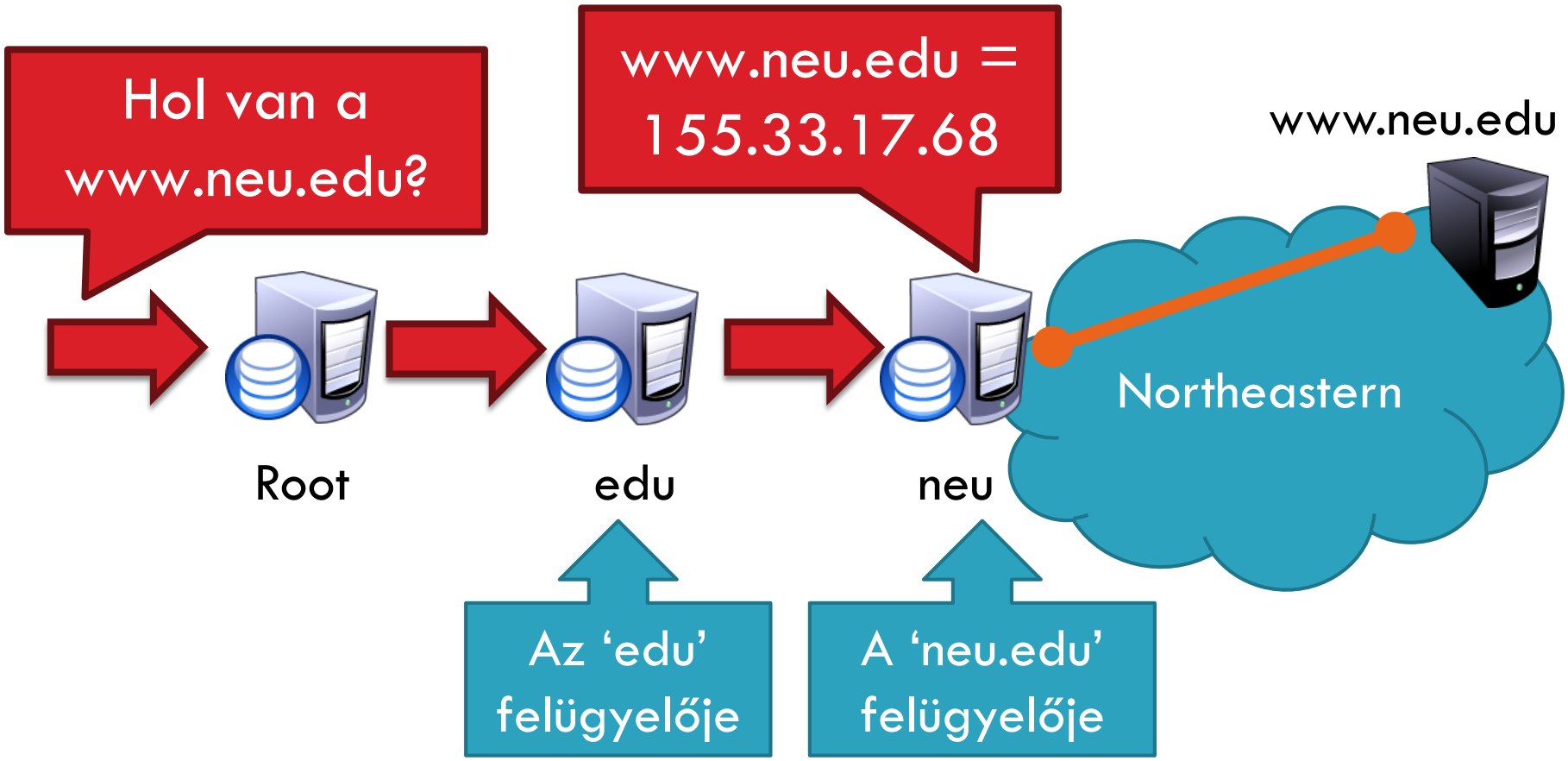
Map of the Roots



Lokális névszerverek

- Minden ISP/cég rendelkezik egy lokális, default névszerverrel
- Gyakran a DHCP konfigurálja fel
- A DNS lekérdezések a lokális névszervernél kezdődnek
- Gyakran cache-be teszik a lekérdezés eredményét

Authoratív Névszerverek



- név→IP leképezéseket tárolja egy adott hoszthoz

Egyszerű doménnév feloldás

- Minden hoszt ismer egy lokális DNS szervert
  - Minden kérést ennek küld
- Ha a lokális DNS szerver tud válaszolni, akkor kész…
  1. A lokális szerver a felügyelő szerver az adott névhez
  2. A lokális szerver cache-ében van rekord a keresett névhez
- Különben menjünk végig a teljes hierarchián felülről lefelé egészen a keresett név felügyeleti szerveréig
  - Minden lokális DNS szerver ismeri a root szervereket
  - Cache tartalma alapján bizonyos lépések átugrása, ha lehet
    - Pl. ha a root fájl tárolva van a cache-ben, akkor egyből ugorhatunk az „.edu” szerverére.

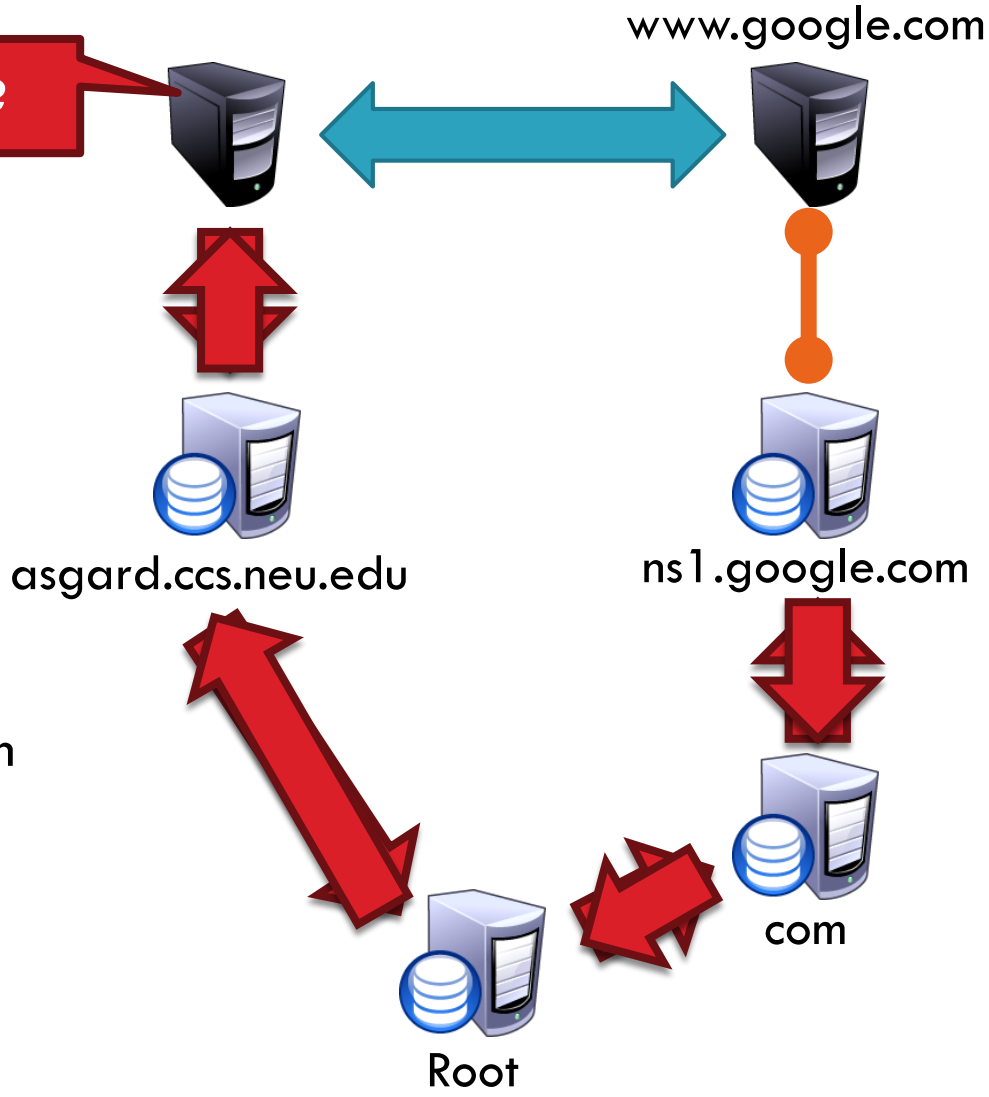
Lekérdezések

- A lekérdezésnek két fajtája van:
  - *Rekurzív lekérdezés* – Ha a névszerver végzi el a névfeloldást, és tér vissza a válasszal.
  - *Iteratív lekérdezés* – Ha a névszerver adja vissza a választ vagy legalább azt, hogy kitől kapható meg a következő válasz.
- Melyik a jobb?
  - *Rekurzív jellemzői*
    - Lehetővé teszi a szervernek a kliens terhelés kihelyezését a kezelhetőségért.
    - Lehetővé teszi a szervernek, hogy a kliensek egy csoportja felett végezzen *cache*lést, a jobb teljesítményért.
  - *Iteratív jellemzői*
    - Válasz után nem kell semmit tenni a kéréssel a névszervernek.
    - Könnyű magas terhelésű szervert építeni.

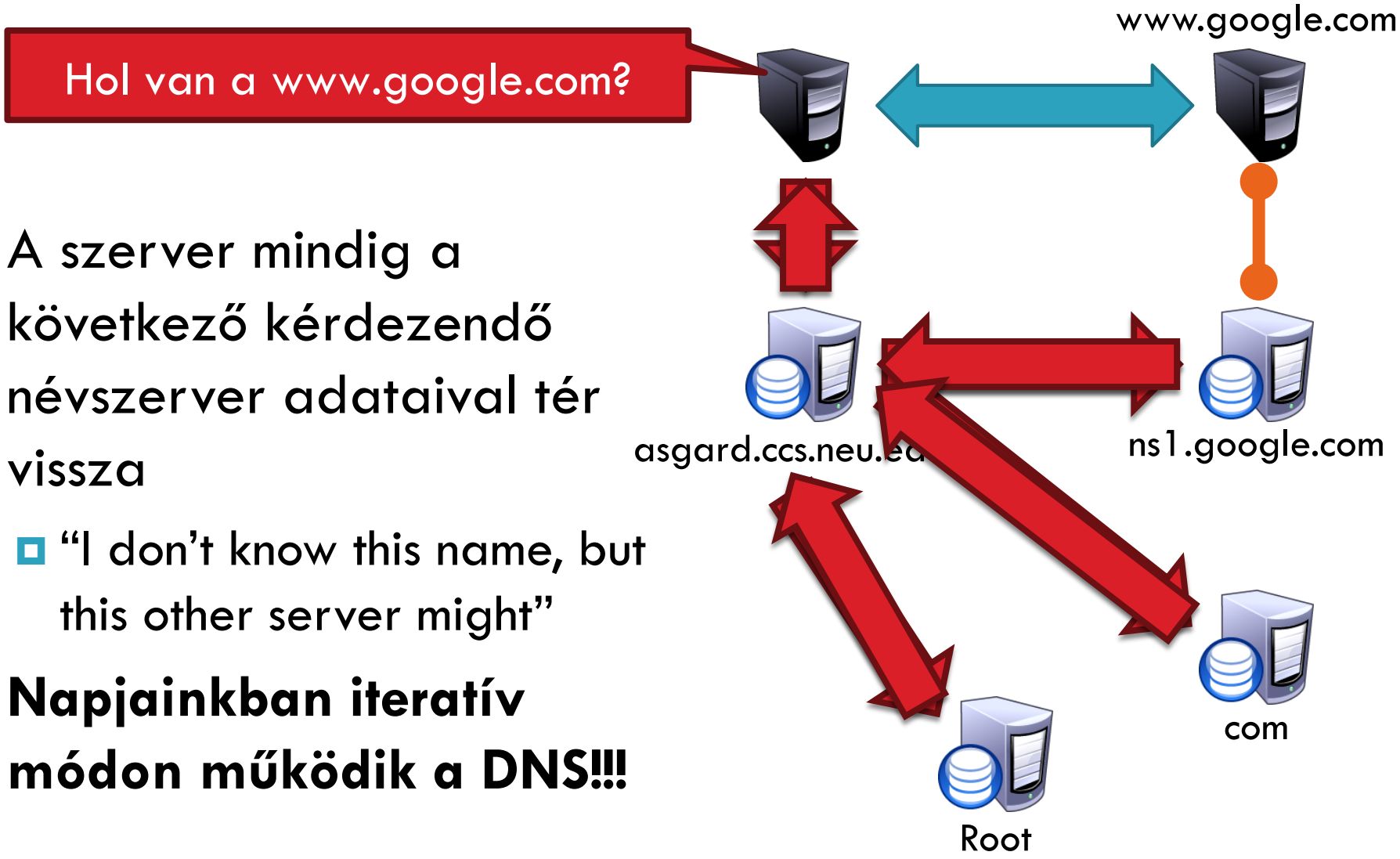
Rekurzív DNS lekérdezés

Hol van a www.google.com?

- A lokális szerver terhet rak a kérdezett névszerverre (pl. root)
- Honnan tudja a kérdezett, hogy kinek továbbítsa a választ?
  - Random ID a DNS lekérdezésben



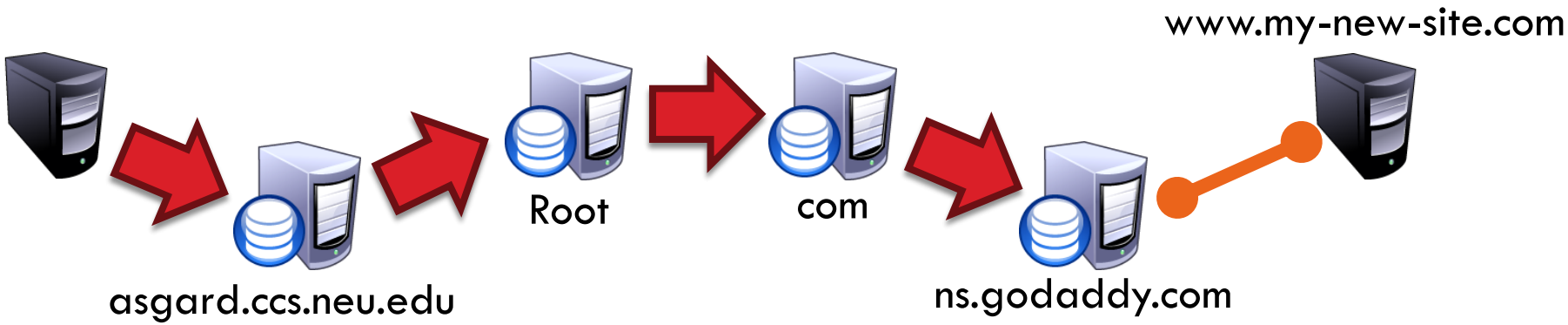
Iteratív DNS lekérdezés



- A szerver mindig a következő kérdezendő névszerver adataival tér vissza
  - "I don't know this name, but this other server might"
- **Napjainkban iteratív módon működik a DNS!!!**

DNS bejegyzés elterjedése

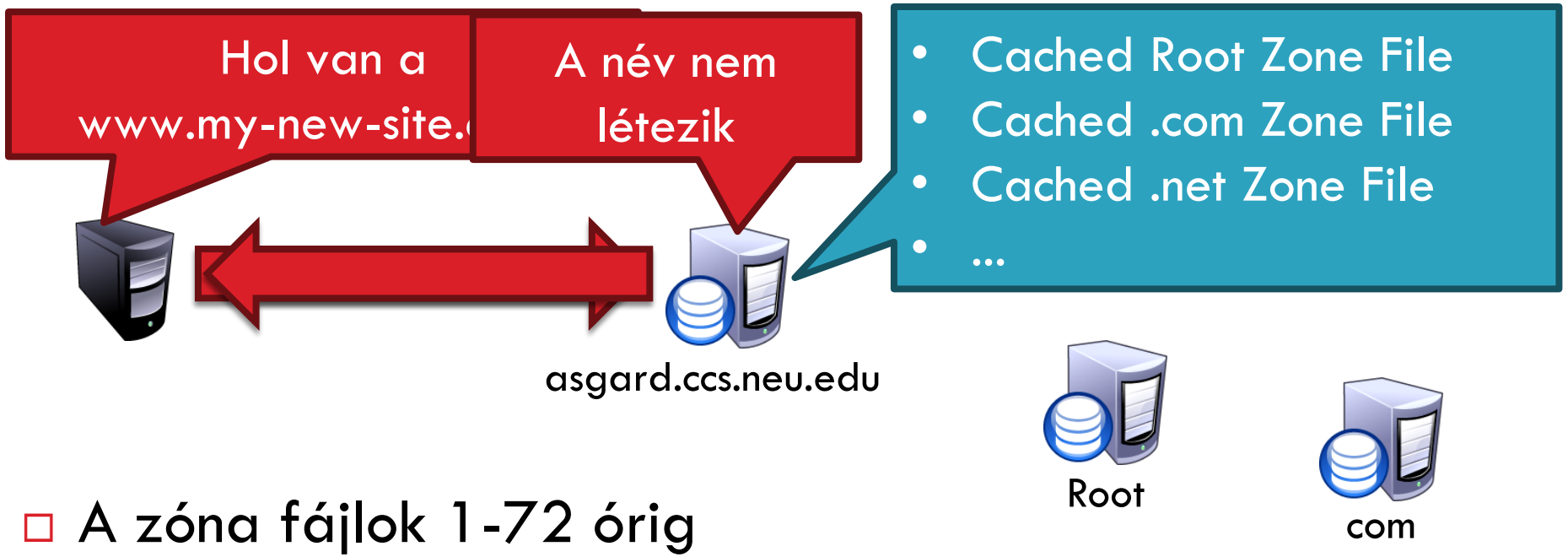
- Van-e a teremben olyan, aki vásárolt már domén nevet?
  - Észrevettétek-e, hogy kb. 72 óra kell ahhoz, hogy elérhető legyen a bejegyzés után?
  - Ez a késés a DNS Propagáció/DNS bejegyzés elterjedése



- Miért nem sikerül ez egy új DNS név esetén?

Cachelés VS frissesség

- DNS elterjedés késését a cache okozza



- A zóna fájlok 1-72 órig élnek a cache-ben

DNS Erőforrás rekordok (Resource Records)

- A DNS lekérdezéseknek két mezőjük van
  - name és type
- Az erőforrás rekord válasz egy DNS lekérdezésre
  - Négy mezőből áll: (name, value, type, TTL)
  - Egy lekérdezésre adott válaszban több rekord is szerpelhet
- Mit jelent a name és a value mező?
  - Ez a lekérdezés típusától (type) függ

DNS lekérdezés típusok

- Type = A / AAAA
  - Name = domén név
  - Value = IP cím
  - A = IPv4, AAAA = IPv6

Query
Name: www.ccs.neu.edu
Type: A

Resp.
Name: www.ccs.neu.edu
Value: 129.10.116.81

- Type = NS
  - Name = rész domén
  - Value = a rész doménhez tartozó DNS szerver neve
  - "Menj és küldd a kérésed ehhez a szerverhez"

Query
Name: ccs.neu.edu
Type: NS

Resp.
Name: ccs.neu.edu
Value: 129.10.116.51

DNS lekérdezés típusok

- Type = CNAME
  - Name = domén név
  - Value = kanonikus név
  - Alias nevek használatához
  - CDN használja

Query
Name: foo.mysite.com
Type: CNAME

Resp.
Name: foo.mysite.com
Value: bar.mysite.com

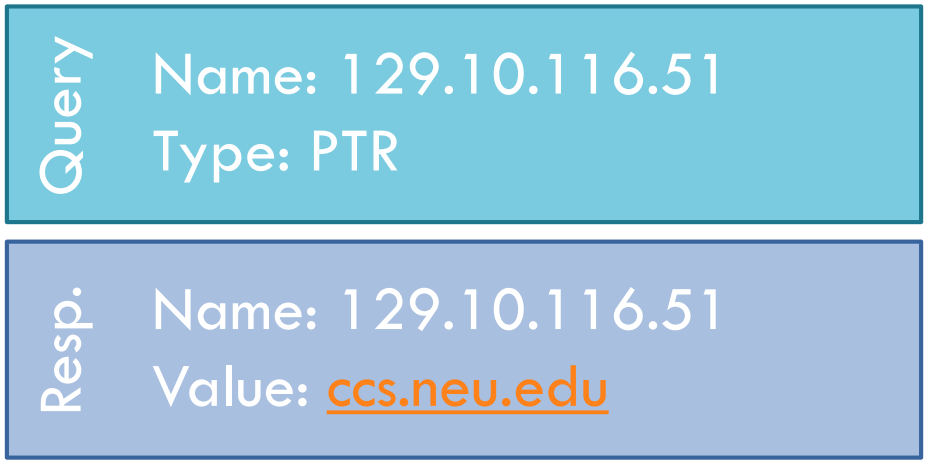
- Type = MX
  - Name = emailben szereplő domén név
  - Value = mail szerver kanonikus neve

Query
Name: ccs.neu.edu
Type: MX

Resp.
Name: ccs.neu.edu
Value: amber.ccs.neu.edu

Fordított lekérdezés (PTR rekord)

- Mi a helyzet az IP→név leképezéssel?
- Külön hierarchia tárolja ezeket a leképezéseket
  - Gyökér pont: in-addr.arpa és ip6.arpa
- DNS rekord típusa (type): PTR
  - Name = IP cím
  - Value = domén név
- Nincs garancia arra, hogy minden IP címre működik



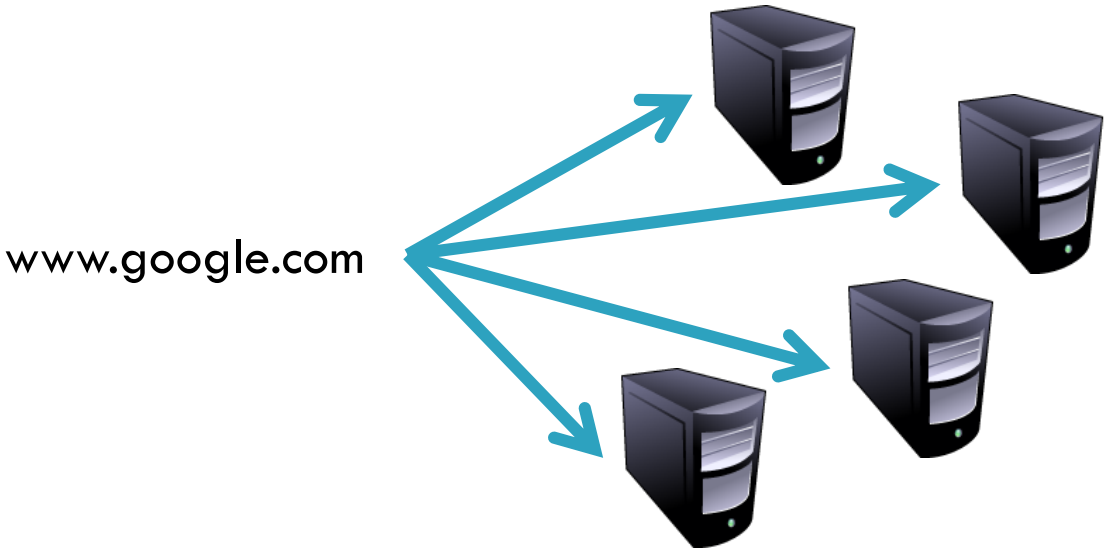
DNS as Indirection Service

- DNS számos lehetőséget biztosít
  - Nem csak a gépekre való hivatkozást könnyíti meg!
- Egy gép IP címének lecserélése is triviális
  - Pl. a web szervert átköltöztetjük egy új hosztra
  - Csak a DNS rekord bejegyzést kell megváltoztatni!

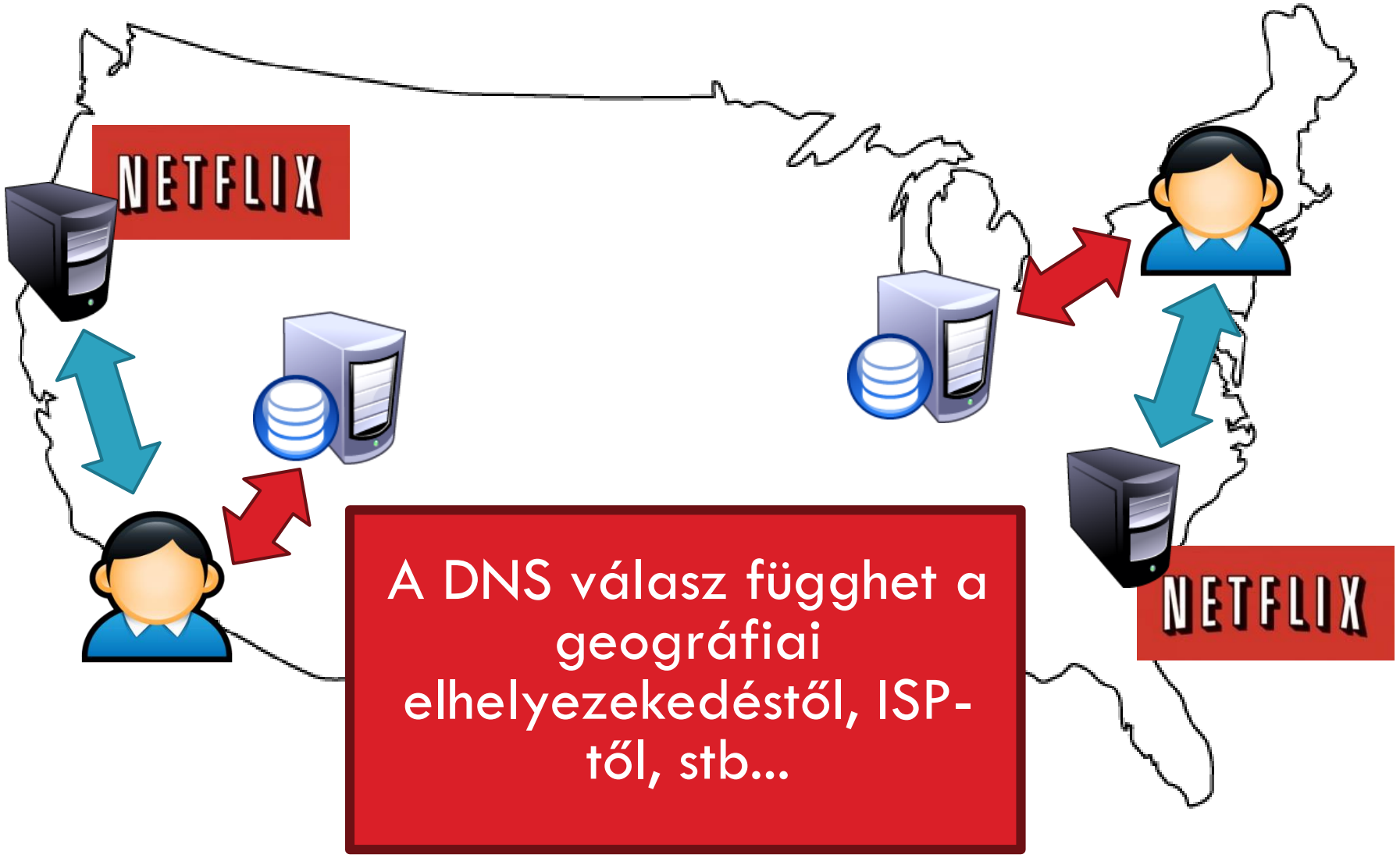
Aliasing/Kanonikus nevek és Load Balancing/Terhelés elosztás

- Egy gépnek számos alias neve lehet

- Egy domén névhez számos IP cím tartozhat



Content Delivery Networks



A DNS fontossága

- DNS nélkül…
  - Hogyan találjunk meg egy weboldalt?
- Példa: a mailszervered azonosít
  - Email címet adunk meg weboldalakra való feliratkozásnál
  - Mi van, ha valaki eltéríti a DNS bejegyzést a mailszerveredhez?
- DNS a bizalom forrása a weben
  - Amikor a felhasználó begépeli a www.bankofamerica.com címet, azt várja, hogy a bankja honlapja jelenjen meg.
  - Mi van, ha a DNS rekordot meghackelték?

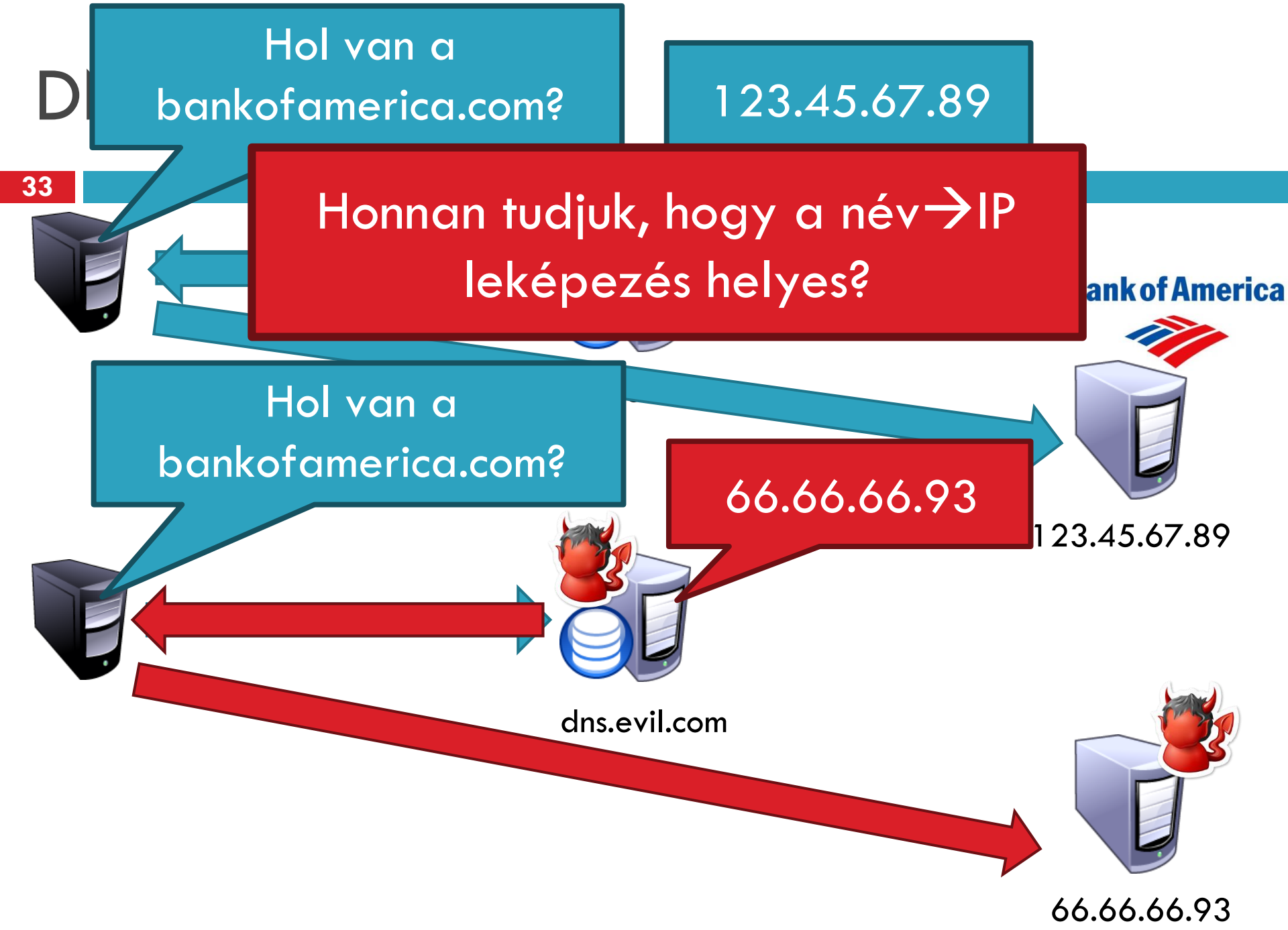
Denial Of Service (DoS)

- DNS szerverek túlterhelése lekérdezésekkel, amíg össze nem omlanak
- 2002 Október: masszív DDoS támadás a root szerverek ellen
  - Mi volt a hatása?
  - … a felhasználók észre se vették
  - Root zónák mindenhol cache-elve vannak
- Célzottabb támadás hatékonyabb lenne
  - Lokális DNS szerver → elérhetetlen DNS
  - Authoritative szerver → elérhetetlen domén

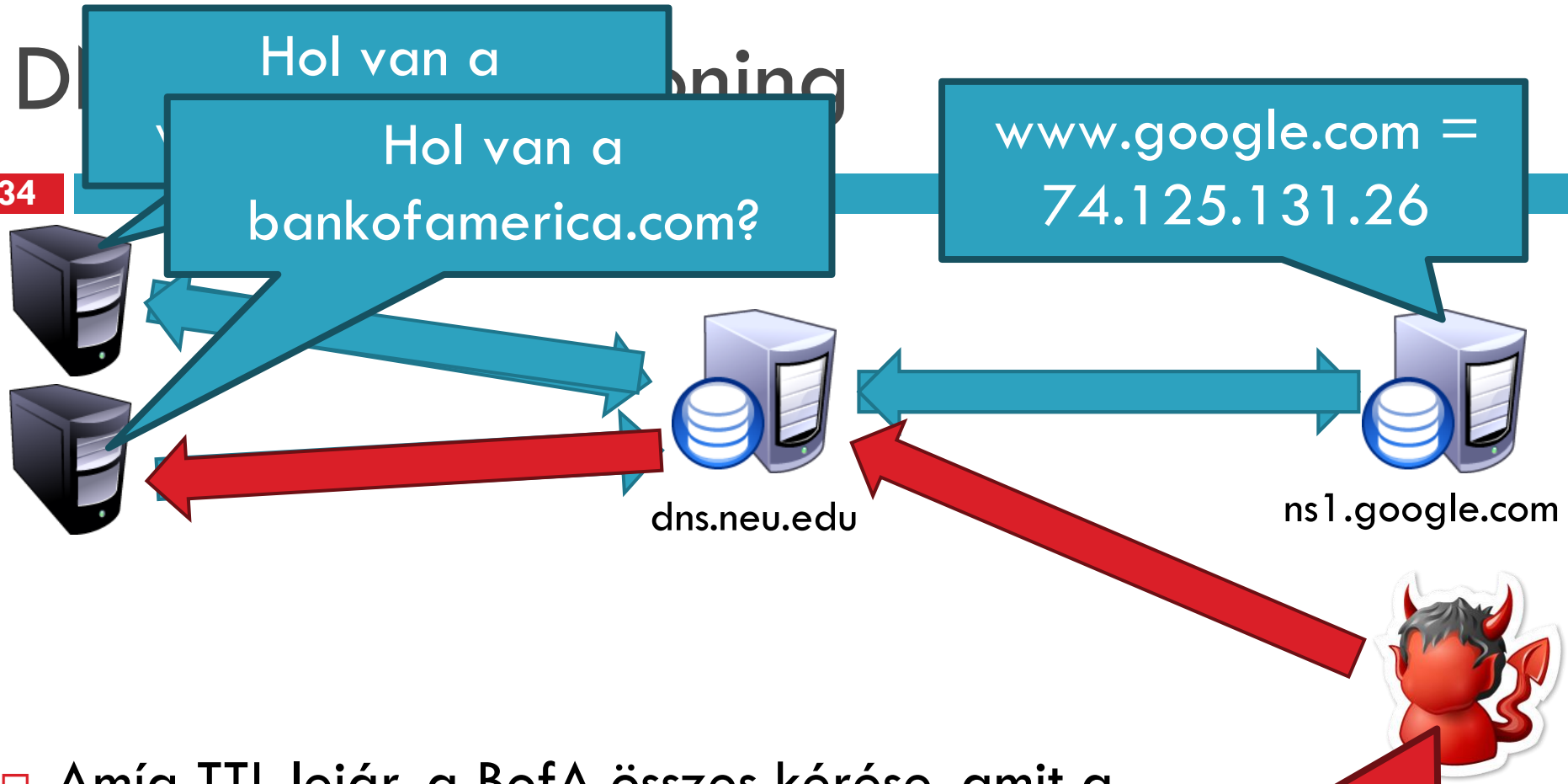
DNS Hijacking (eltérítés)

- OS vagy browser megfertőzése (virus/trojan)
  - Pl. Számos trójai megváltoztatja a bejegyzéseket a /etc/hosts fájlnan
  - *.bankofamerica.com → evilbank.com
- Man-in-the-middle

- Válasz hamisítás (spoofing)
  - Kérések lehallgatása
  - Válaszok megversenyeztetése


Hol van a bankofamerica.com?
123.45.67.89
Honnan tudjuk, hogy a név→IP leképezés helyes?
ank of America
Hol van a bankofamerica.com?
66.66.66.93
123.45.67.89
dns.evil.com
66.66.66.93

D[illegible]oning



- Amíg TTL lejár, a BofA összes kérése, amit a dns.neu.edu-nak küld, hamis/fertőzött vissza
- Sokkal rosszabb, mint a spoofing/man-in-the-middle
  - Egy teljes ISP-t érinthet!

Hogyan éri el a támadó a fertőzött bejegyzés tárolását?

- 1. Azt mondjuk a feloldónak, hogy az áldozathoz tartózó NS a támadó IP-jén érhető el
  - Kiváltó lekérdezés: subdomain.attacker.example IN A
  - Támadó válasza:
- Válasz: (nincs válasz a lekérdezésre), de
  - Authority Section: attacker.example. 3600 IN NS ns.target.example.
  - Additional Section: ns.target.example. IN A w

A támadó azt mondja, „hogy a doménem authoratív szervere a ns.target.example és mellesleg itt van az IP-je”…

Hogyan éri el a támadó a fertőzött bejegyzés tárolását?

- 2. NS rekord átirányítása a támadó doménébe
  - Kiváltó lekérdezés: subdomain.attacker.example IN A
  - Válasz: (nincs válasz)
    - Authority section:
      - Target.example. 3600 IN NS ns.attacker.example.
    - Additional section:
      - Ns.attacker.example. IN A w.x.y.z

A támadó nem releváns információt szúr be, melyet a szerver el fog tárolni a cacheben…

Megoldás: DNSSEC

- Kritikus rekordokat kriptografikus aláírással látjuk el
  - A feloldó ellenőrzi az aláírást
- Két új erőforrástípus
  - Type = DNSKEY
    - Name = Zóna domén
    - Value = A zóna publikus kulcsa
  - Type = RRSIG
    - Name = (type, name) páros, pl. a lekérdezés maga
    - Value = A lekérdezés eredményének kriptografikus aláírása
- Elterjedése
  - Root szerverek 2010 Július óta
  - Verisign lehetővé tette a .com és .net doménekben 2011 Január

Bizalmi hierarchiát hoz létre a zónák között

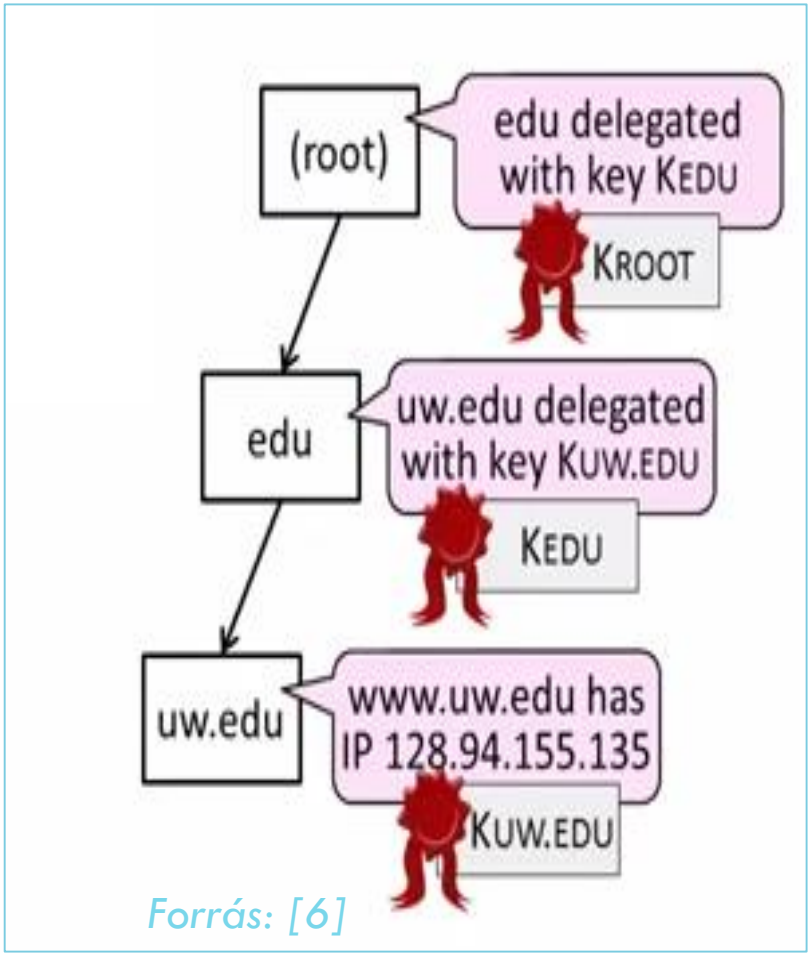
Megelőzi az eltérítést és a hamisítást

DNSSEC 2

- Új rekordtípusokat vezettek be:
  - RRSIG a rekordok digitális aláírására.
  - DNSKEY publikus kulcsok használatához a validáció során.
  - DS publikus kulcsok használatához a delegációhoz.
  - NSEC/NSEC3 a létezés hitelesített visszautasítása.
- Első változat 1997-ben jelent meg. 2005-ben újraírták.
- Ez a fejlesztés viszont a kliensek és szerverek frissítését vonja maga után. (2010 - „Root” szerverek frissítése.)
- A kliensek a szokott módon kérdezik le a DNS-t, és később ellenőrzik a válasz hitelességét.
- A *Trust Anchor* a publikus kulcsok gyökere. (DNS kliens konfigurációjának a része)
- A biztonság leköveti a DNS hierarchiát.

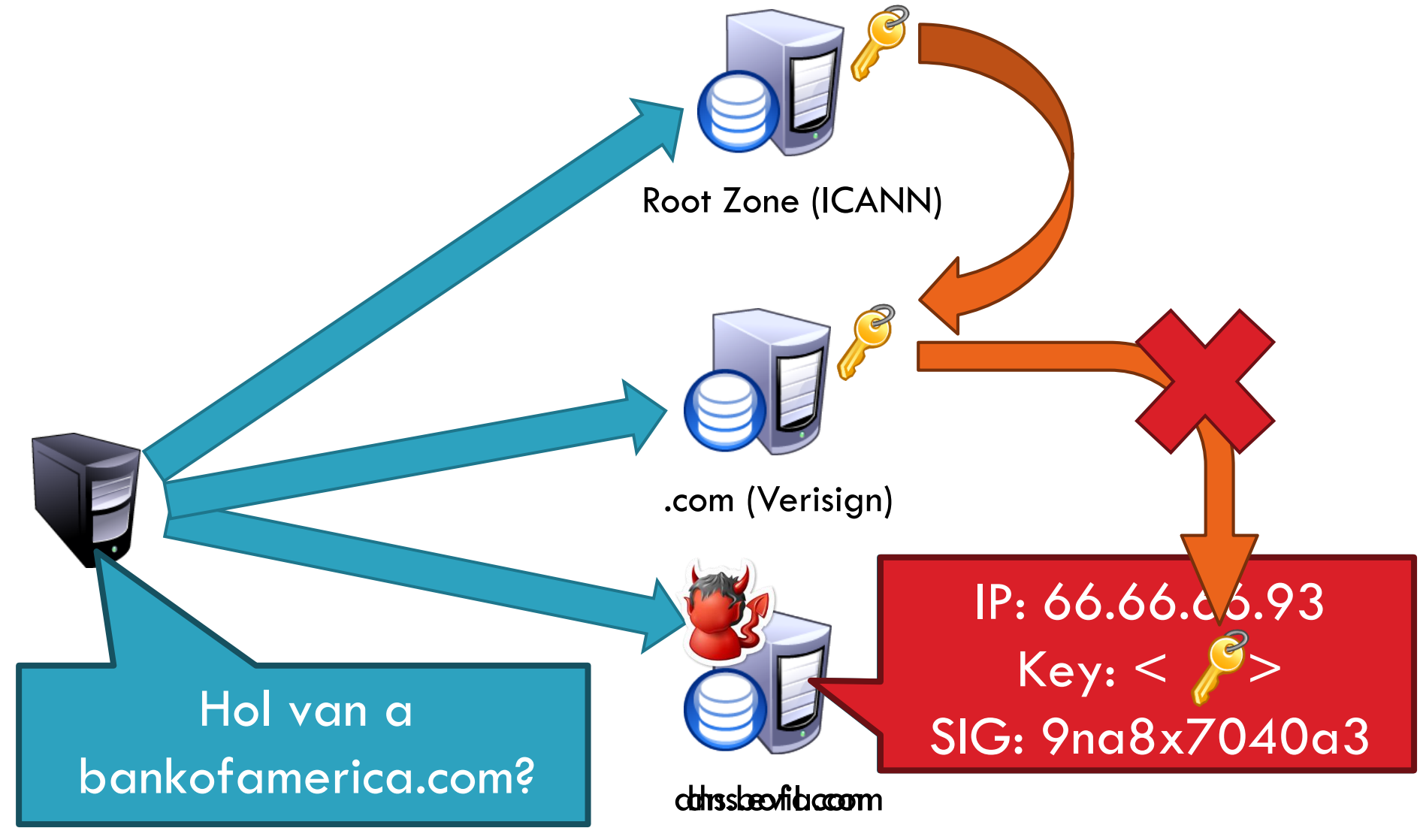
DNSSEC 3

- Példa a www.uw.edu lekérdezése egy klienssel.
- Kliens hitelesíti a választ:
  1. KROOT egy *Trust Anchor*.
  2. KROOT-ot használja a KEDU ellenőrzésére.
  3. KEDU-t használja a KUW.EDU ellenőrzésére.
  4. KUW.EDU-t használja a IP cím ellenőrzésére.



Forrás: [6]

DNSSEC Bizalmi hierarchia



Does DNSSEC Solve all our problems?

- No.
- DNS still vulnerable to reflection attacks + injected responses

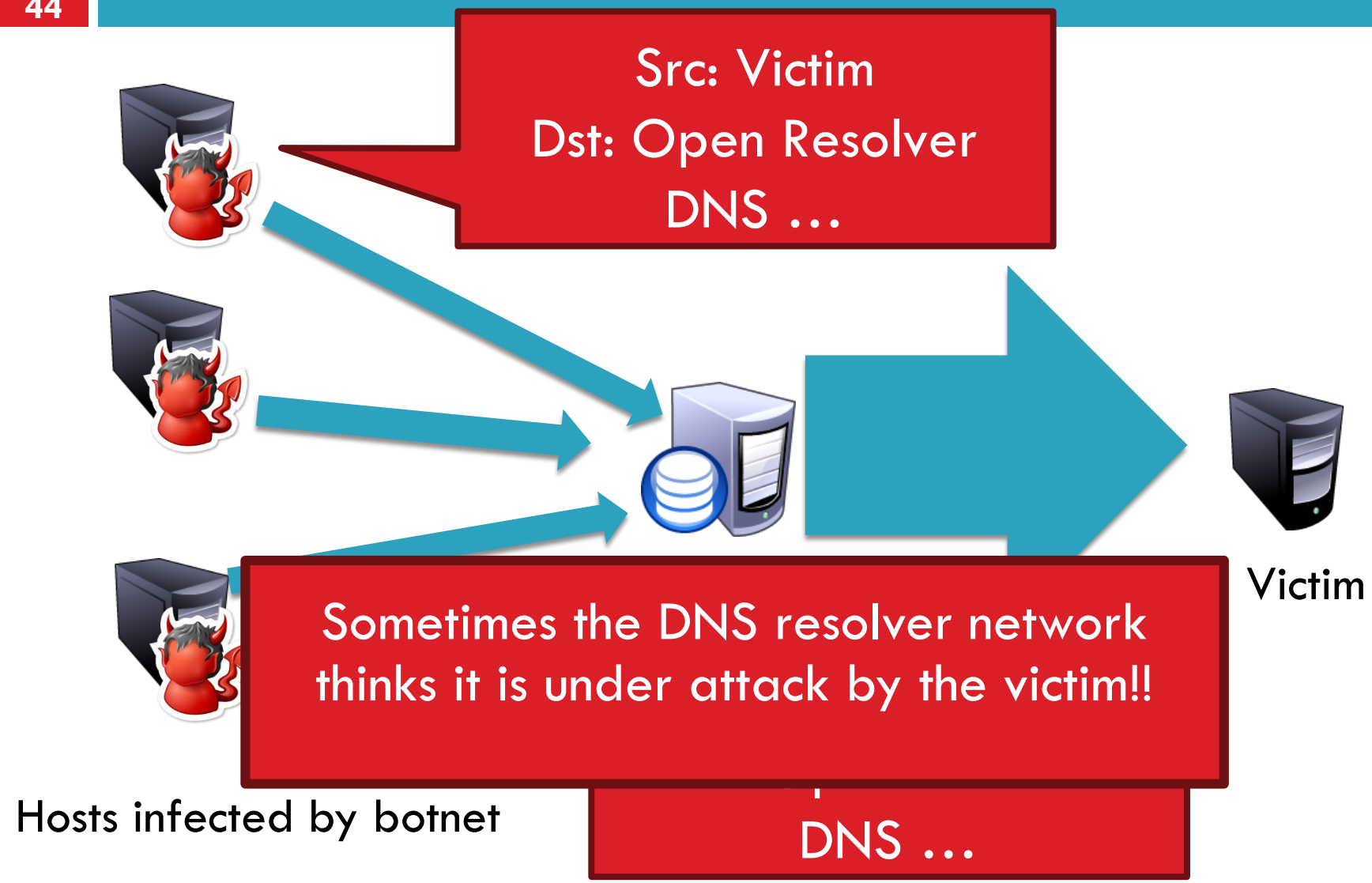
DNS Reflection

- Very big incident in 2012
  - (http://blog.cloudflare.com/65gbps-ddos-no-problem/)
  - 65 Gbps DDoS
  - Would need to compromise 65,000 machines each with 1 Mbps uplink
    - How was this attack possible?
- Use DNS reflection to amplify a Botnet attack.
- Key weak link: Open DNS resolvers will answer queries for anyone http://openresolverproject.org/

So how does this work?

- Remember: DNS is UDP
- No handshaking between endpoints
- I can send a DNS query with a forged IP address and the response will go to that IP address
  - **Secret sauce:** a small request that can elicit a large response
  - E.g., query for zone files, or DNSSEC records (both large record types).
- Botnet hosts spoof DNS queries with victim's IP address as source
  - Resolver responds by sending massive volumes of data to the victim

DNS amplification illustrated



HTTP

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

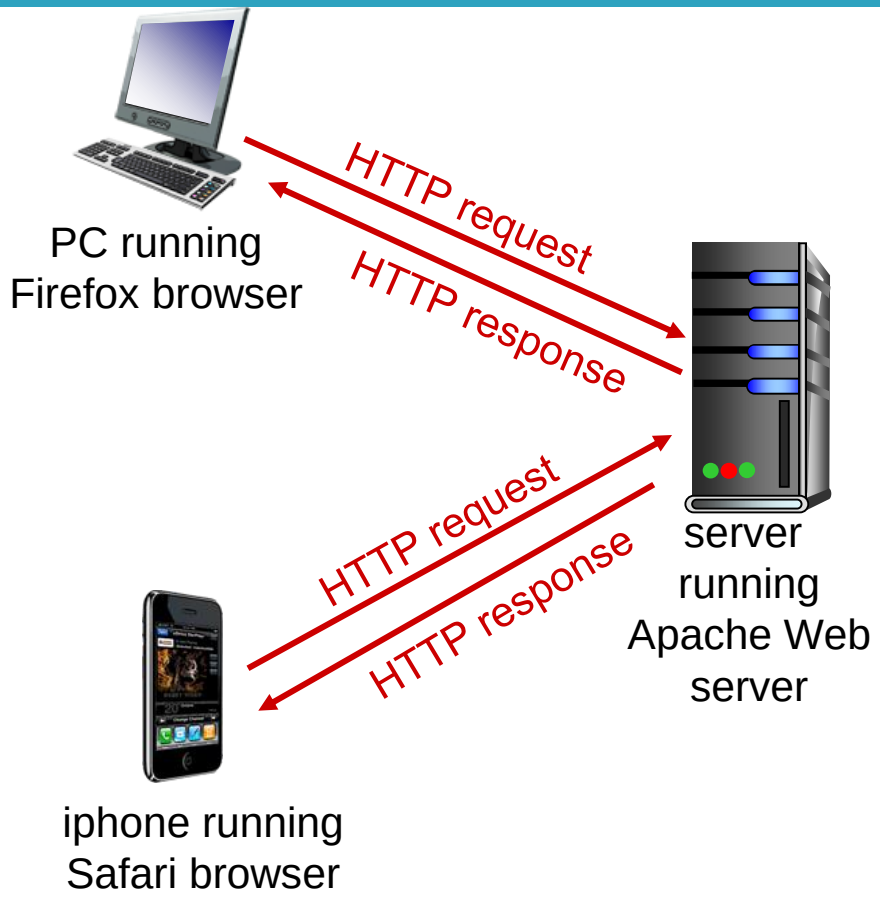
`www.someschool.edu/someDept/pic.gif`

host name | 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" (in theory…)

- server maintains no information about past client requests

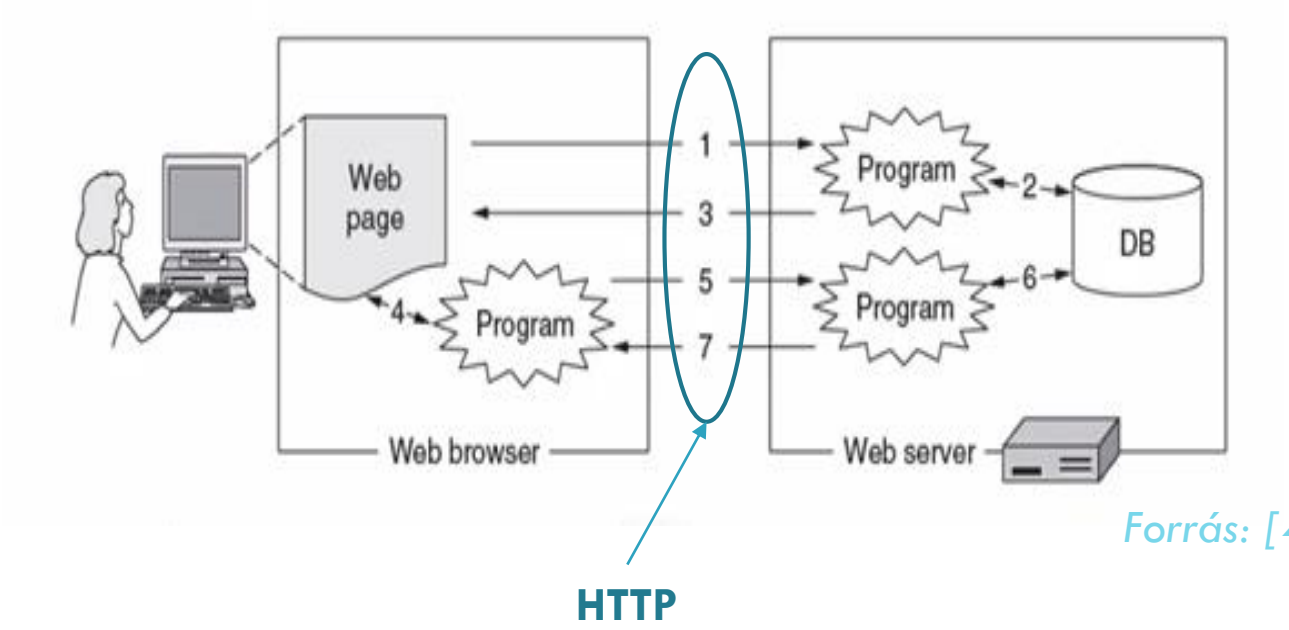
aside

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

Statikus és dinamikus weboldalak

- A *statikus weboldal* tartalma nem változik csak manuális átszerkesztéssel.
- A *dinamikus weboldal* valamilyen kód végrehajtásaként keletkezik, mint például: *javascript*, *PHP*, vagy mindkettő egyszerre.



Forrás: [4]

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

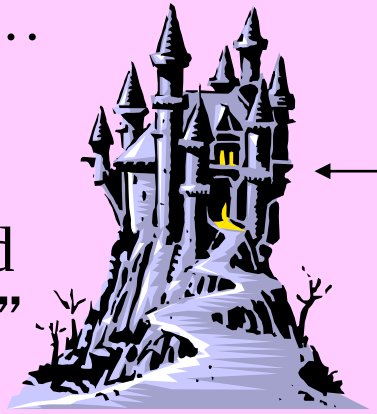
Example Web Page

page.html

Harry Potter Movies

As you all know, the new HP book will be out in June and then there will be a new movie shortly after that…

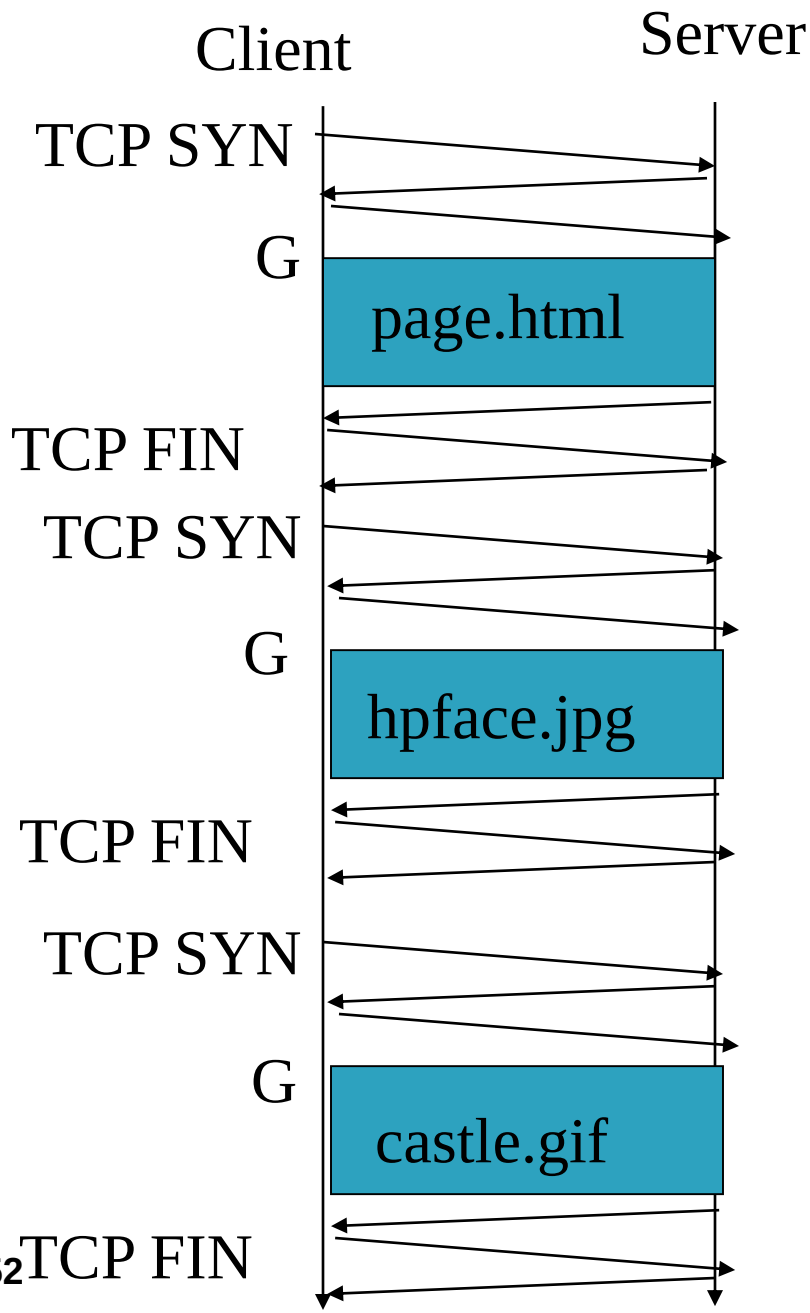
hpface.jpg



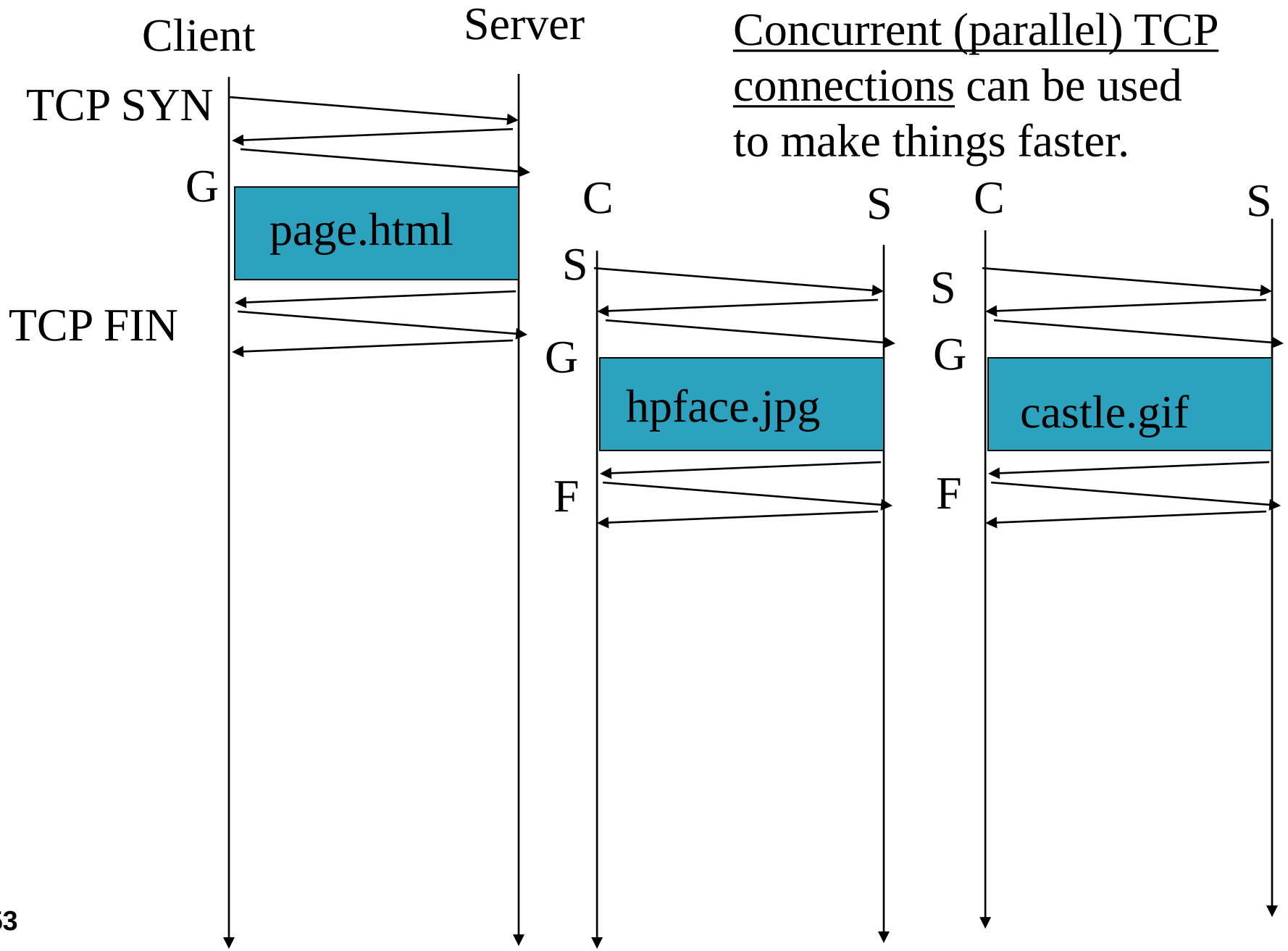
castle.gif

“Harry Potter and the Bathtub Ring”

Non-Persistent HTTP



The "classic" approach in HTTP/1.0 is to use one HTTP request per TCP connection, serially.



Concurrent (parallel) TCP connections can be used to make things faster.

Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- 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 sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

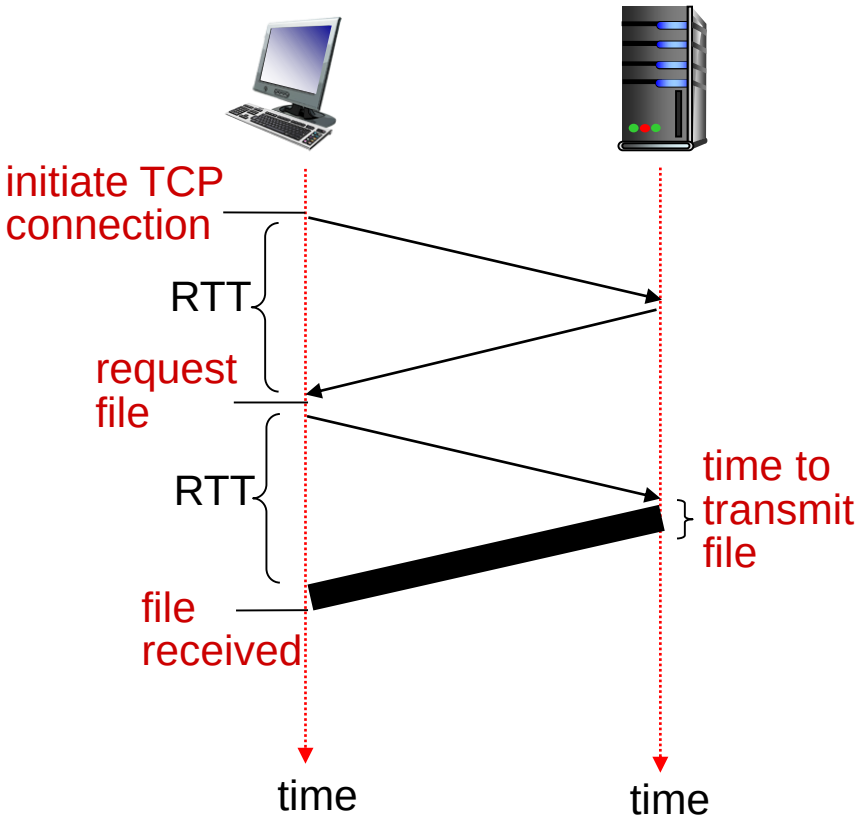
Non-persistent HTTP: response time

RTT: time for a packet to travel from client to server and back

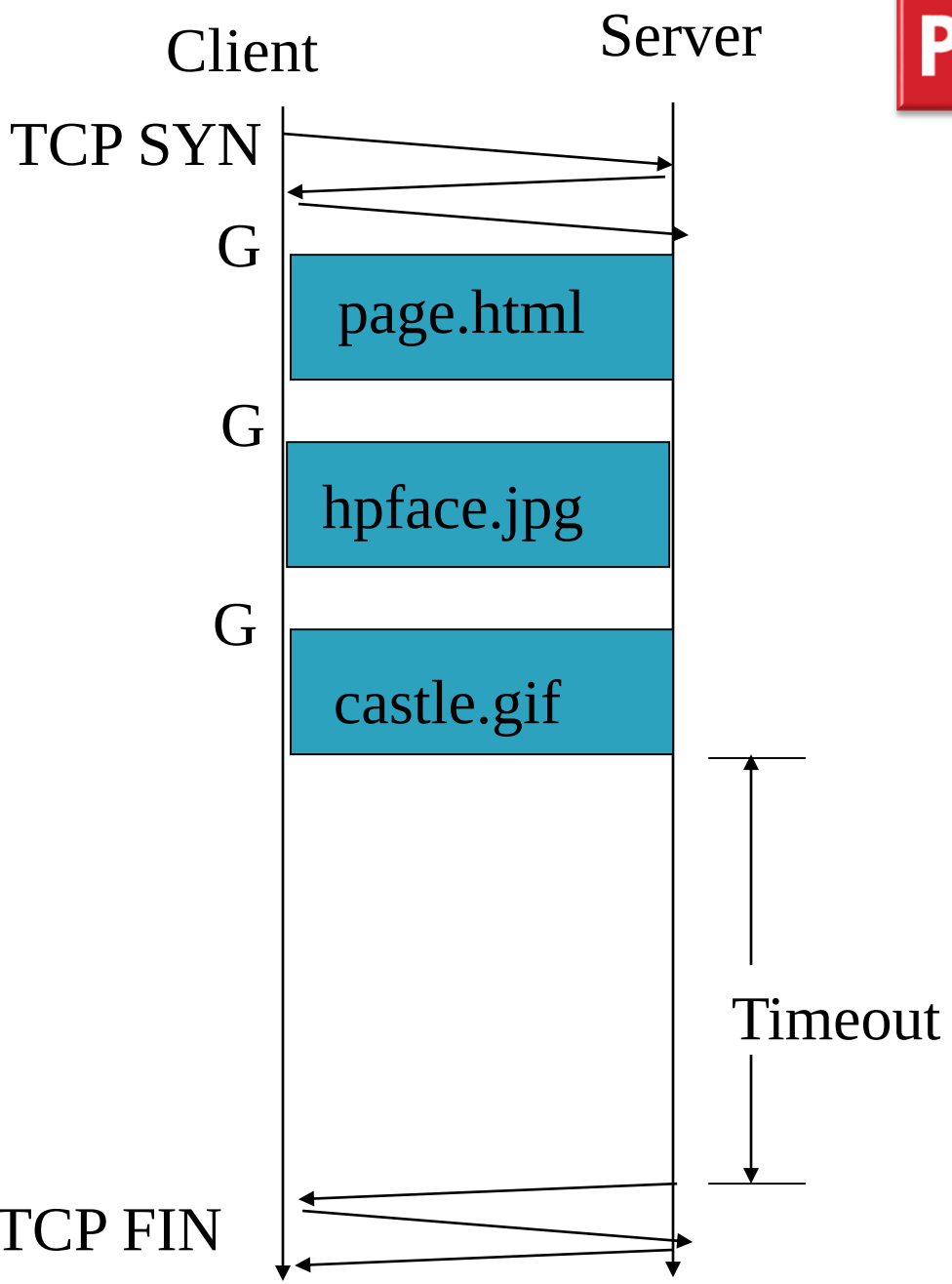
HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
  - This assumes HTTP GET piggy backed on the ACK
- file transmission time
- non-persistent HTTP response time =

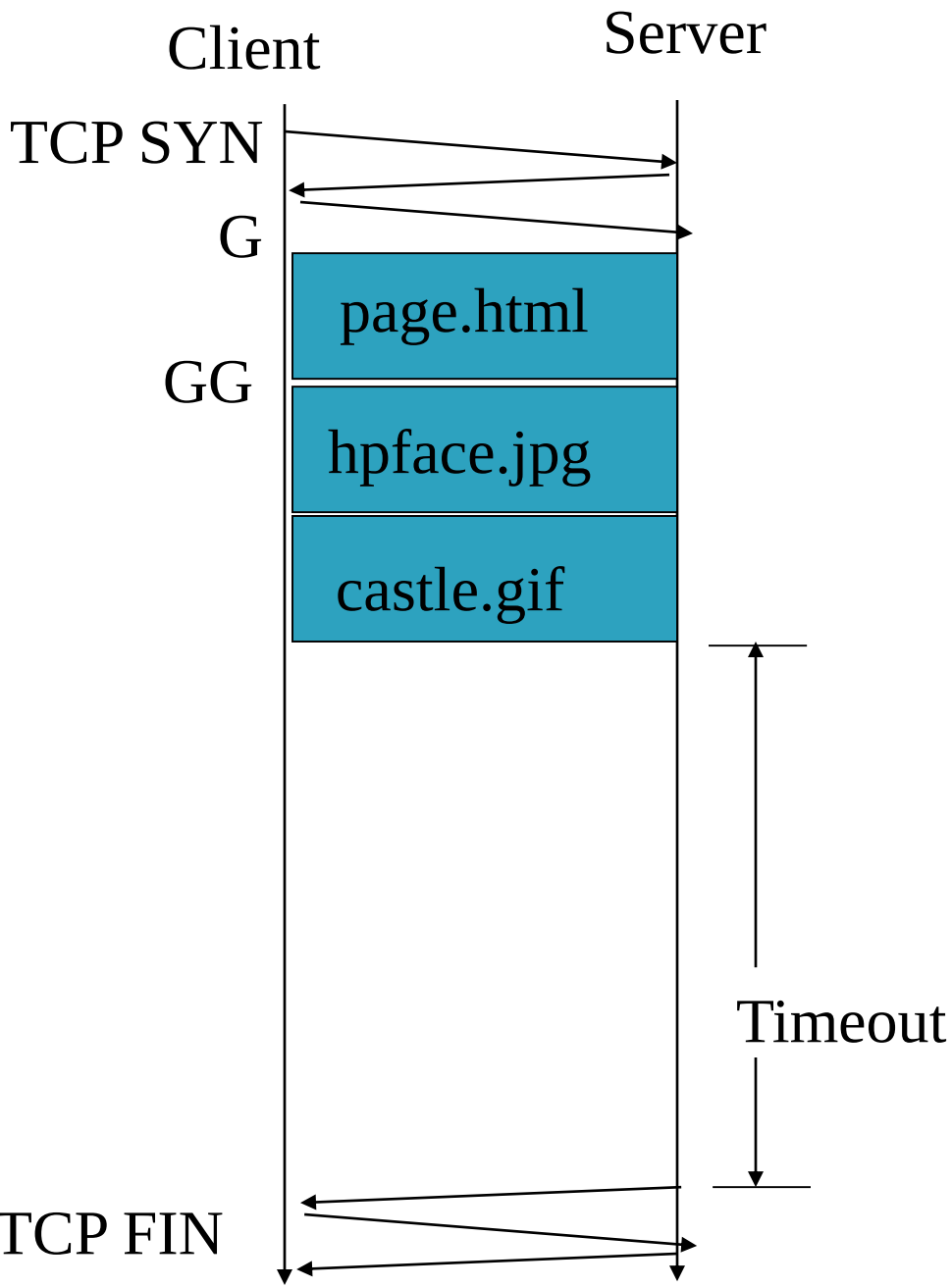
  2RTT+ file transmission time



Persistent HTTP



The "persistent HTTP" approach can re-use the same TCP connection for Multiple HTTP transfers, one after another, serially. Amortizes TCP overhead, but maintains TCP state longer at server.



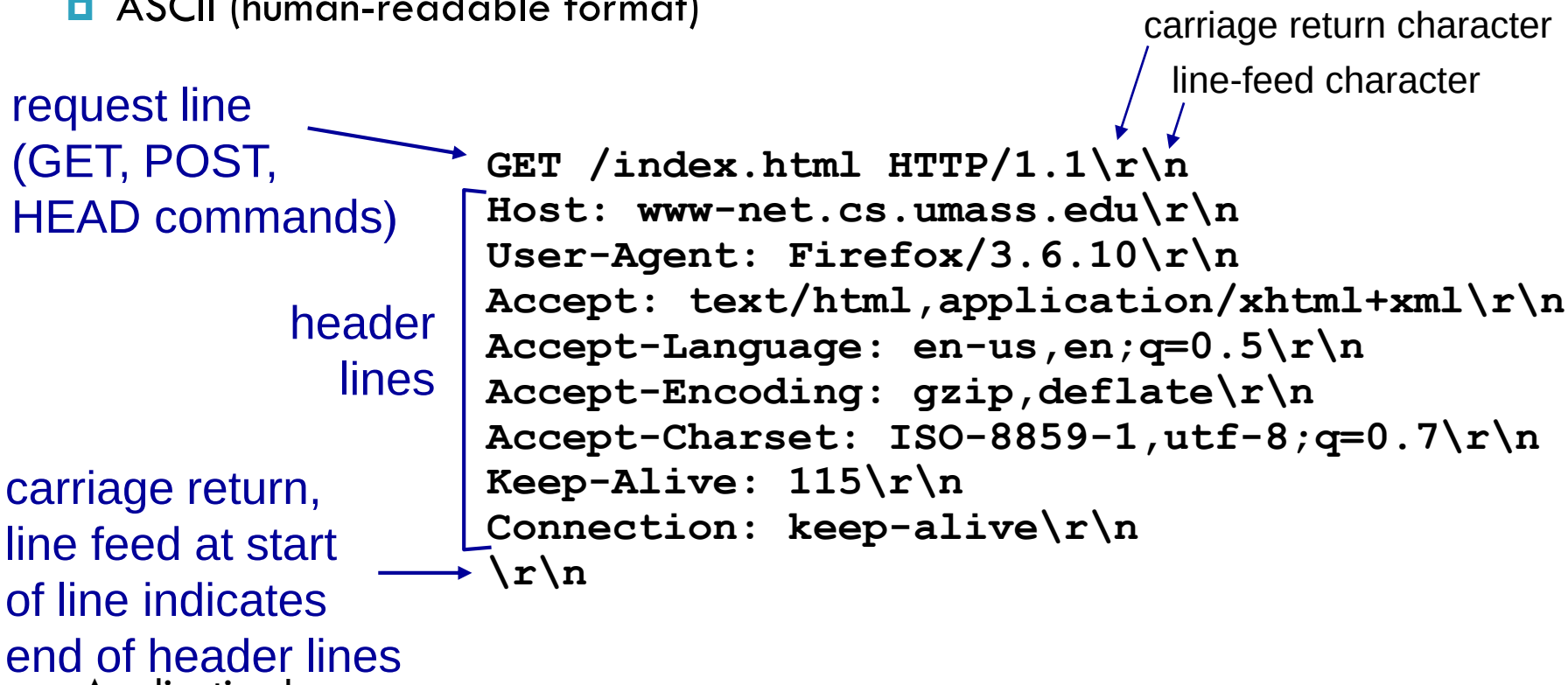
The "pipelining" feature in HTTP/1.1 allows requests to be issued asynchronously on a persistent connection. Requests must be processed in proper order. Can do clever packaging.

Outline

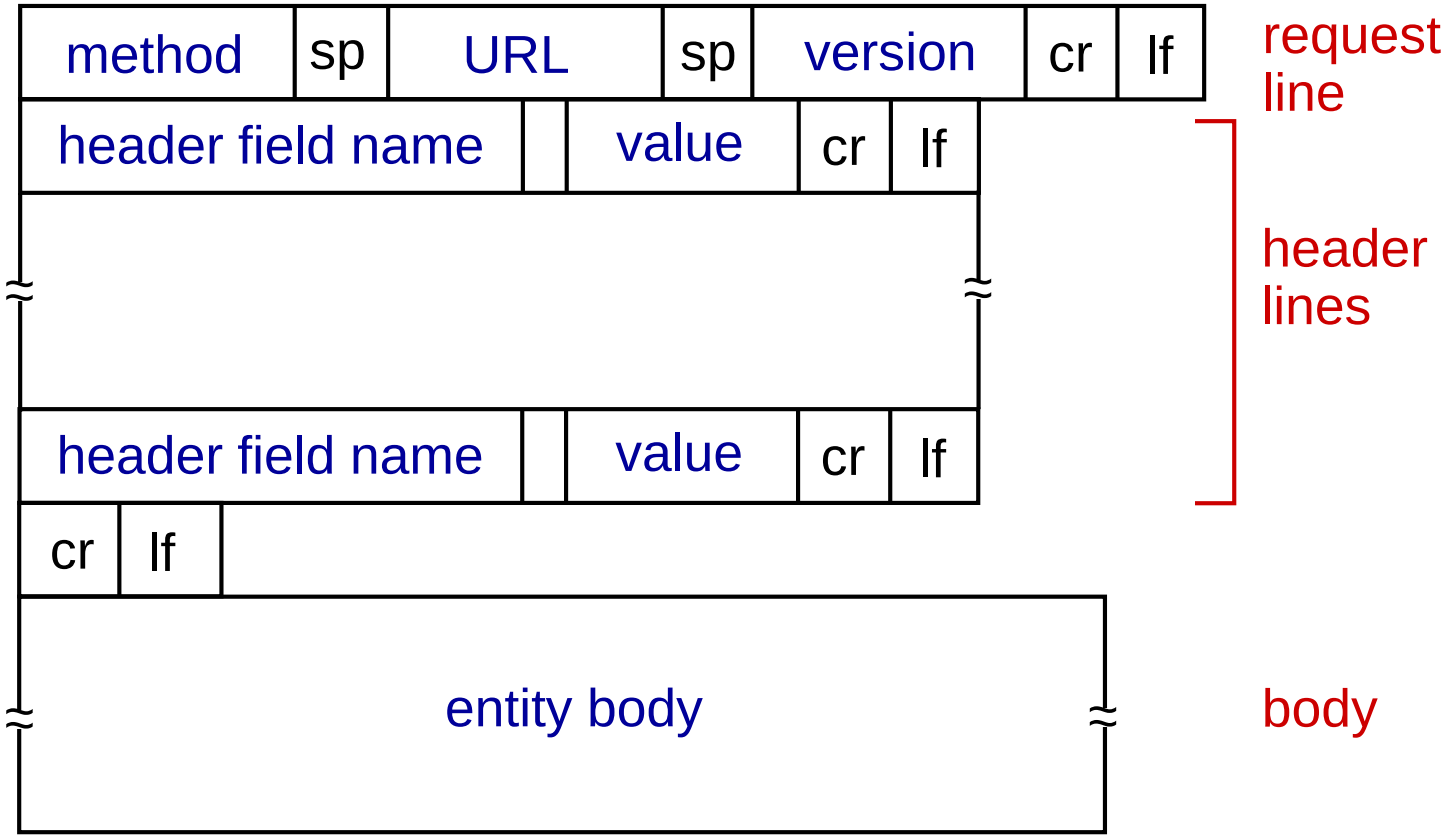
- HTTP Connection Basics
- HTTP Protocol
- Cookies, keeping state + tracking

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP request message: general format



Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

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 appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (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

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Outline

- HTTP Connection Basics
- HTTP Protocol
- Cookies, keeping state + tracking

User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)

usual http request msg

Amazon server creates ID 1678 for user

create entry

backend database

usual http response
set-cookie: 1678

usual http request msg
cookie: 1678

cookie-specific action

access

usual http response msg

one week later:

access

usual http request msg
cookie: 1678

cookie-specific action

usual http response msg

Application Layer

Cookies (continued)

what cookies can be used for:

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

aside

cookies and privacy:

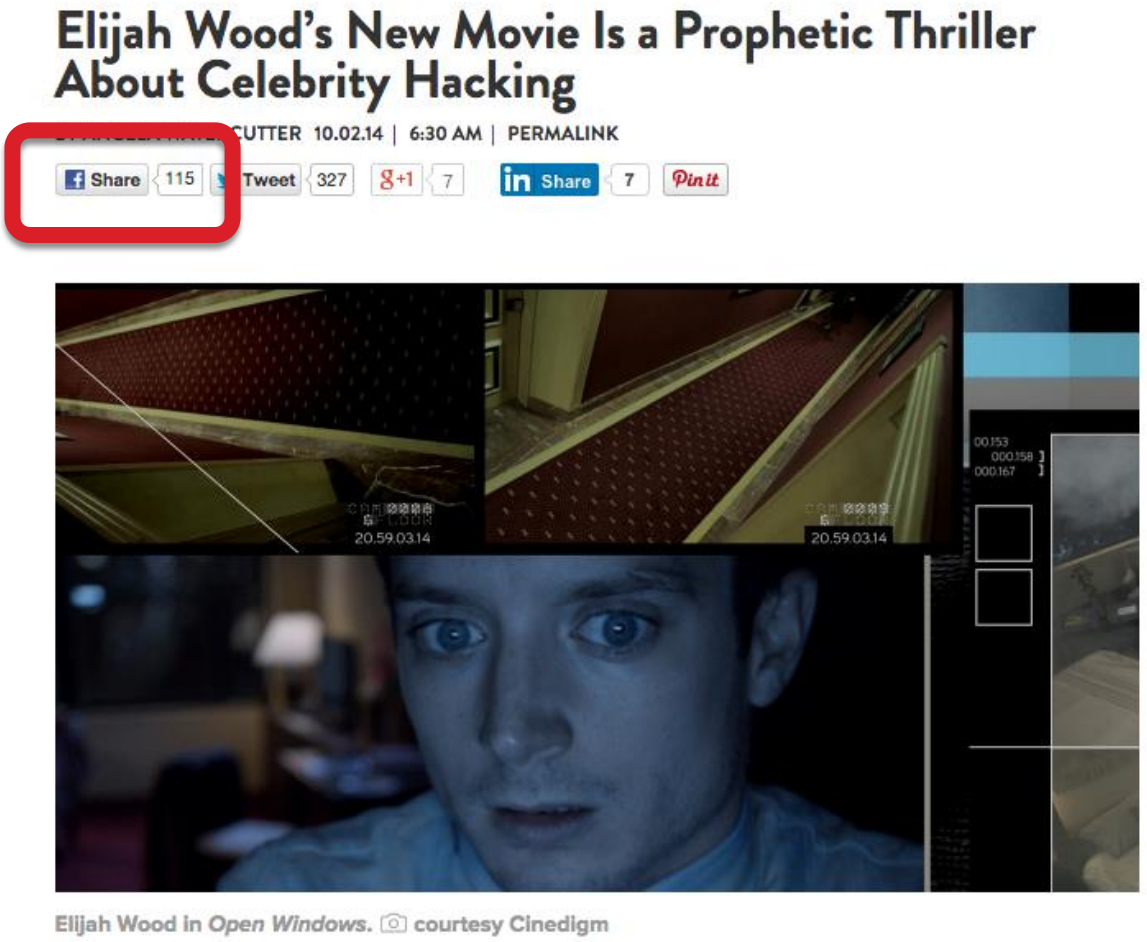
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Cookies + Third Parties

- Example page (from Wired.com)



How it works

And it's not just Facebook!

Wi

GET article.html

GET sharebutton.gif
Cookie: FBCOOKIE

Facebook now knows you visited this Wired article.
Works for all pages where 'like'/'share' button is embedded!

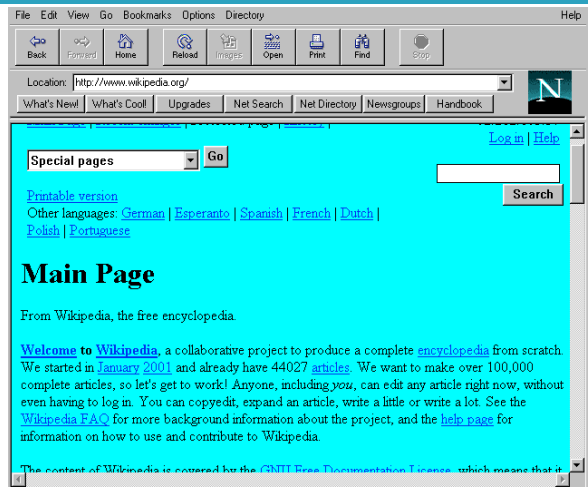
CDN

Content in today's Internet

- Most flows are HTTP
  - Web is at least 52% of traffic
  - Median object size is 2.7K, average is 85K (as of 2007)
- HTTP uses TCP, so it will
  - Be ACK clocked
  - For Web, likely never leave slow start
- Is the Internet designed for this common case?
  - Why?

Evolution of Serving Web Content

- In the beginning…
  - …there was a single server
  - Probably located in a closet
  - And it probably served blinking text



- Issues with this model
  - Site reliability
    - Unplugging cable, hardware failure, natural disaster
  - Scalability
    - Flash crowds (aka Slashdotting)

Replicated Web service

- Use multiple servers
- Advantages
  - Better scalability
  - Better reliability
- Disadvantages
  - How do you decide which server to use?
  - How to do synchronize state among servers?

Load Balancers

- Device that multiplexes requests across a collection of servers
  - All servers share one public IP
  - Balancer transparently directs requests to different servers
- How should the balancer assign clients to servers?
  - Random / round-robin
    - When is this a good idea?
  - Load-based
    - When might this fail?
- Challenges
  - Scalability (must support traffic for n hosts)
  - State (must keep track of previous decisions)
    - RESTful APIs reduce this limitation

Load balancing: Are we done?

- Advantages
  - Allows scaling of hardware independent of IPs
  - Relatively easy to maintain
- Disadvantages
  - Expensive
  - Still a single point of failure
  - Location!

Where do we place the load balancer for Wikipedia?

Popping up: HTTP performance

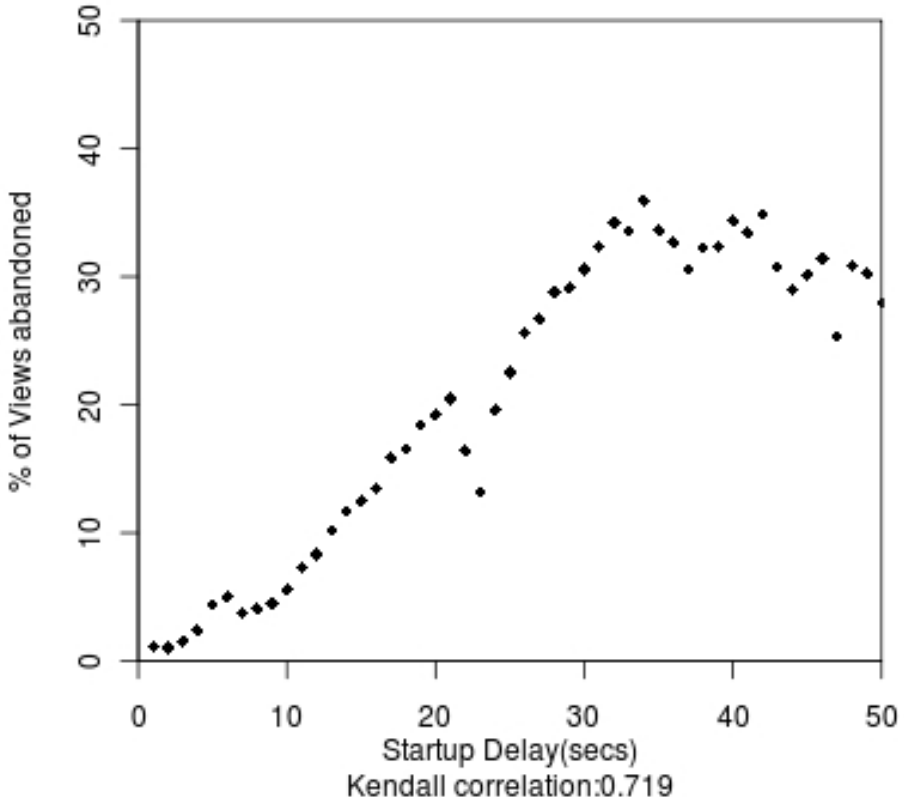
- For Web pages
  - RTT matters most
  - Where should the server go?
- For video
  - Available bandwidth matters most
  - Where should the server go?

- Is there one location that is best for everyone?

Server placement

Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay



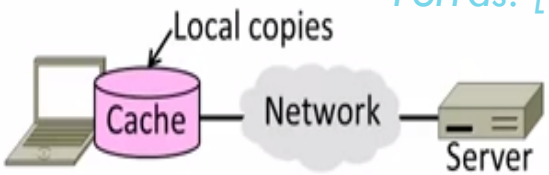
Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay
- Impact on revenue
  - Amazon: increased revenue 1% for every 100ms reduction in page load time (PLT)
  - Shopzilla:12% increase in revenue by reducing PLT from 6 seconds to 1.2 seconds
- Ping from BOS to LAX: ~100ms

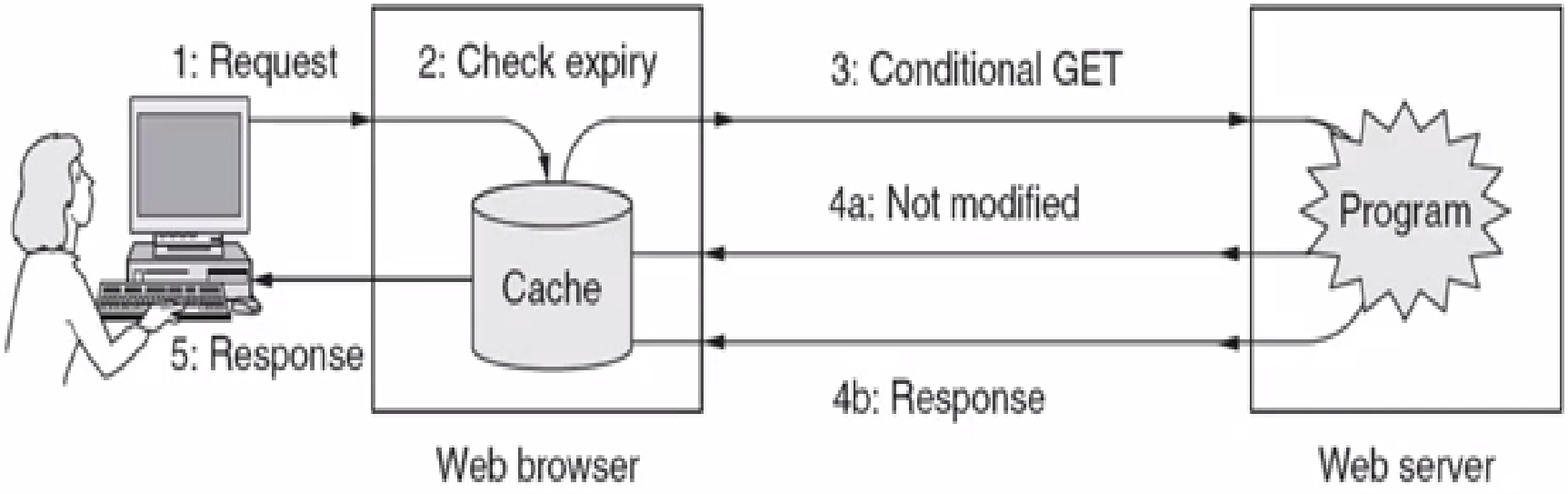
Web „caching” 1/2

- Felhasználók gyakran újra látogatják az oldalakat.
  - Használható a lokális másolata a lapnak. Ezt nevezzük „caching”-nek.
- **Kérdés**: *Mikor használhatjuk a lokális másolatot?*
  - Lokálisan meghatározni, hogy valid-e a kópia.
    - Lejárati információk alapján, mint például az „Expires” fejléc alapján a szervertől.
    - Heurisztikák használatával megtippelni.
    - **Előny**: A tartalom azonnal elérhető.
  - A másolat újra validálása a szerverrel.
    - Másolatban lévő időbélyegző alapján. (a *Last-Modified* fejléc)
    - A másolat tartalma alapján. (az *Etag* fejléc a szervertől)
    - **Előny**: A tartalom egy RTT-nyi késleltetéssel elérhető.

Forrás: [5]



Web „caching” 2/2

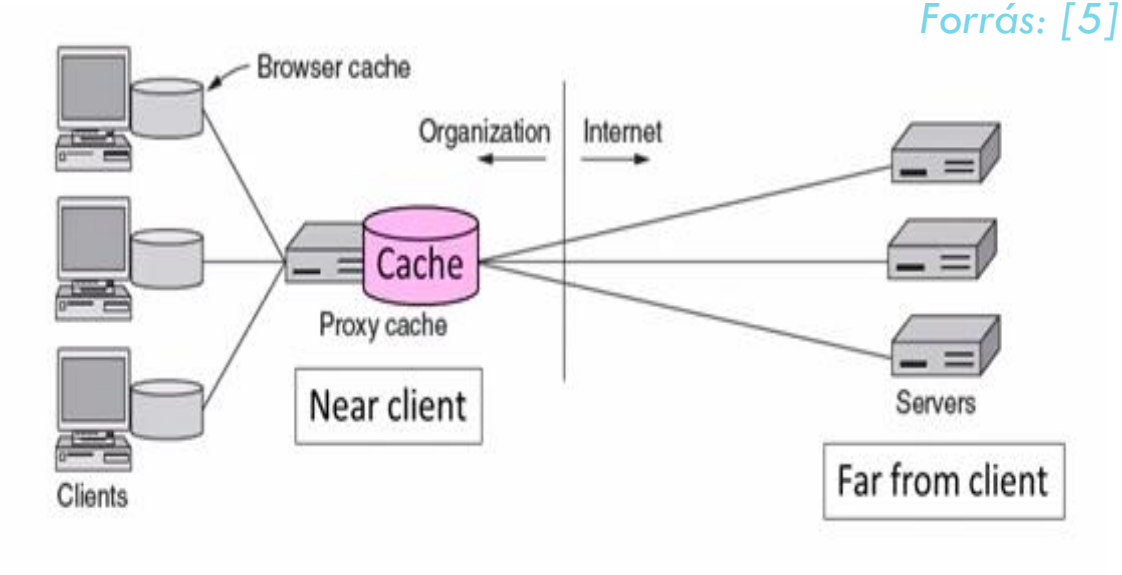


Forrás: [5]

Web proxy

- A kliensek csoportja és a külső web szerverek közé egy közbelső elem elhelyezése, ez lesz a proxy.
  - Előnyök a klienseknek: nagyobb *cache* és biztonsági ellenőrzés.
  - A szervezeti hozzáférés szabályozás.
- *Proxy cache* egy elosztott, nagy gyorsító tárat biztosít a klienseknek.
  - Korlátozások a tárolásra: biztonságos tartalmak, dinamikus tartalmak.

Forrás: [5]



Strawman solution: Web caches at ISPs

- ISP uses a middlebox that caches Web content
  - Better performance – content is closer to users
  - Lower cost – content traverses network boundary once
  - Does this solve the problem?

- No!
  - Size of all Web content is too large ☹
  - Web content is **dynamic** and **customized**
    - Can't cache banking content
    - What does it mean to cache search results?

What is a CDN?

- Content Delivery Network
  - Also sometimes called Content Distribution Network
  - At least half of the world's bits are delivered by a CDN
    - Probably closer to 80/90%
- Primary Goals
  - Create replicas of content throughout the Internet
  - Ensure that replicas are always available
  - Directly clients to replicas that will give good performance

Key Components of a CDN

- Distributed servers
  - Usually located inside of other ISPs
  - Often located in IXPs (coming up next)
- High-speed network connecting them
- Clients (eyeballs)
  - Can be located anywhere in the world
  - They want fast Web performance
- Glue
  - Something that binds clients to “nearby” replica servers

Examples of CDNs

- Akamai
  - 147K+ servers, 1200+ networks, 650+ cities, 92 countries
- Limelight
  - Well provisioned delivery centers, interconnected via a private fiber-optic connected to 700+ access networks
- Edgecast
  - 30+ PoPs, 5 continents, 2000+ direct connections
- Others
  - Google, Facebook, AWS, AT&T, Level3, Brokers

Inside a CDN

- Servers are deployed in clusters for reliability
  - Some may be offline
    - Could be due to failure
    - Also could be "suspended" (e.g., to save power or for upgrade)
- Could be multiple clusters per location (e.g., in multiple racks)
- Server locations
  - Well-connected points of presence (PoPs)
  - Inside of ISPs

Mapping clients to servers

- CDNs need a way to send clients to the "best" server
  - The best server can change over time
  - And this depends on client location, network conditions, server load, …
  - What existing technology can we use for this?

- DNS-based redirection
  - Clients request www.foo.com
  - DNS server directs client to one or more IPs based on request IP
  - Use short TTL to limit the effect of caching

CDN redirection example


```
choffnes$ dig www.fox.com


;; ANSWER SECTION:
www.fox.com.                    510     IN      CNAME   www.fox-rma.com.edgesuite.net.
www.fox-rma.com.edgesuite.net. 5139 IN  CNAME   a2047.w7.akamai.net.
a2047.w7.akamai.net.            4       IN      A       23.62.96.128
a2047.w7.akamai.net.            4       IN      A       23.62.96.144
a2047.w7.akamai.net.            4       IN      A       23.62.96.193
a2047.w7.akamai.net.            4       IN      A       23.62.96.162
a2047.w7.akamai.net.            4       IN      A       23.62.96.185
a2047.w7.akamai.net.            4       IN      A       23.62.96.154
a2047.w7.akamai.net.            4       IN      A       23.62.96.169
a2047.w7.akamai.net.            4       IN      A       23.62.96.152
a2047.w7.akamai.net.            4       IN      A       23.62.96.186
```

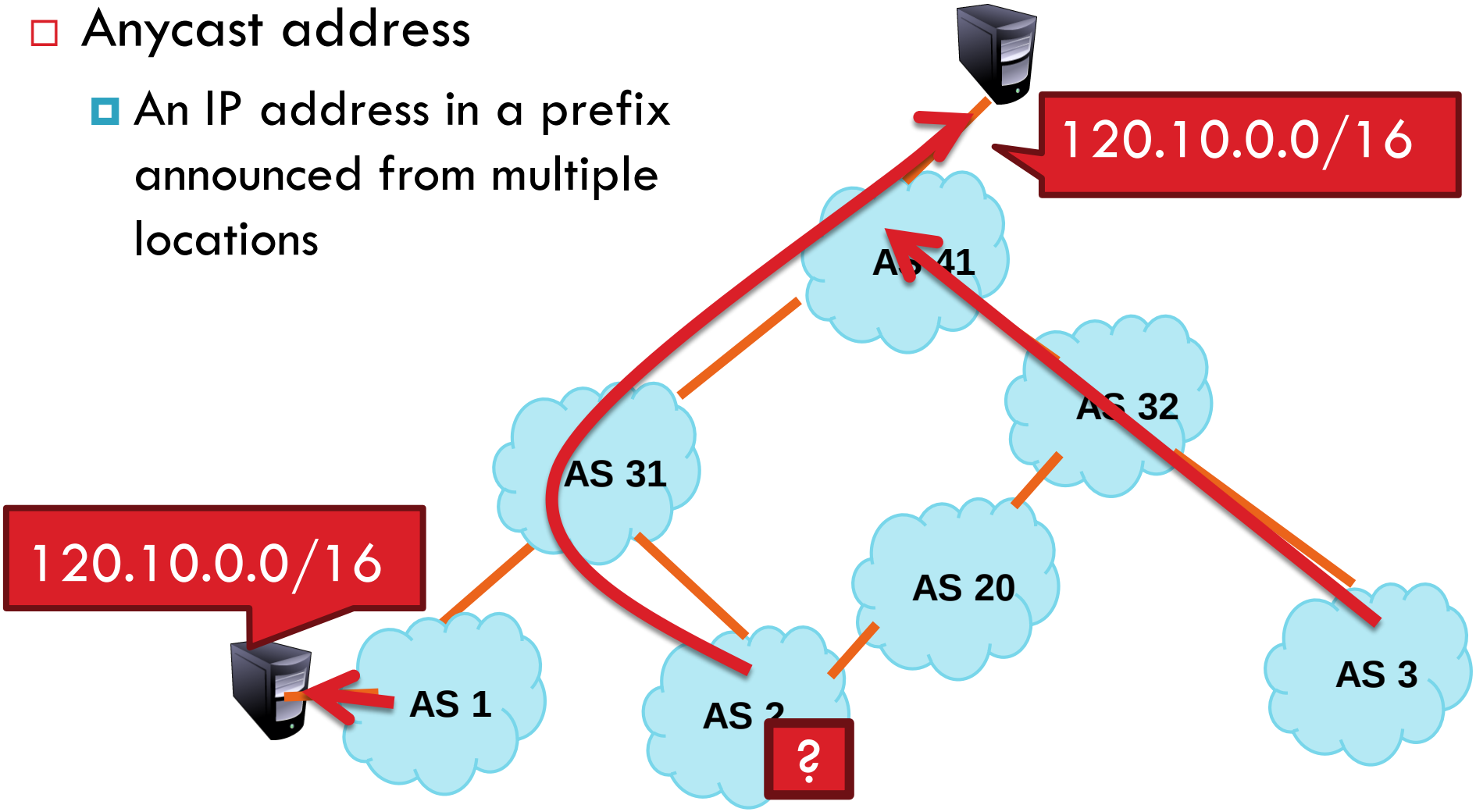
DNS Redirection Considerations

- Advantages
  - Uses existing, scalable DNS infrastructure
  - URLs can stay essentially the same
  - TTLs can control "freshness"

- Limitations
  - DNS servers see only the DNS resolver IP
    - Assumes that client and DNS server are close. Is this accurate?
  - Small TTLs are often ignored
  - Content owner must give up control
  - Unicast addresses can limit reliability

CDN Using Anycast

- Anycast address
  - An IP address in a prefix announced from multiple locations



Anycasting Considerations

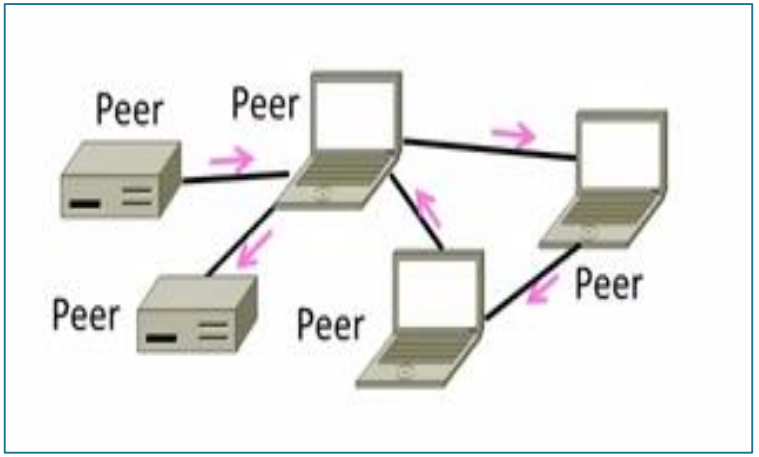
- Why do anycast?
  - Simplifies network management
    - Replica servers can be in the same network domain
  - Uses best BGP path
- Disadvantages
  - BGP path may not be optimal
  - Stateful services can be complicated

p2p tartalom megosztás 1/3

- CDNs hátrányai
  - Dedikált infrastruktúrát igényelnek.
  - Centralizált vezérlés/felügyelet kell.
- **Cél**: Olyan dedikált infrastruktúra és központi felügyelet mentes kézbesítés megvalósítása, ami még mindig hatékony és megbízható.
- **Kulcs**: a résztvevők, avagy peer-ek, segítsenek magukon
  - Napster 1999 zenei tartalomra,
  - BitTorrent 2001 bármilyen tartalomra.

P2P HÁLÓZAT JELLEMZŐI

- Nincs szerver.
- A kommunikáció peer-ek között folyik és önszerveződő.
- Skálázási problémák merülnek fel.



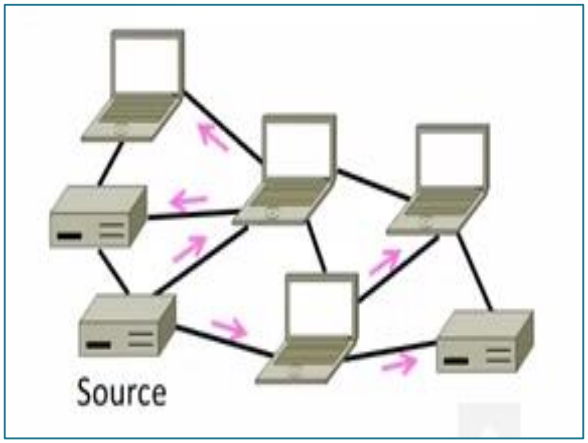
p2p hálózat
Forrás: [2]

p2p tartalom megosztás 2/3

P2P HÁLÓZAT KIHÍVÁSOK

1. Korlátozott lehetőségek
   - Hogyan képes egy *peer* tartalmat továbbítani az összes többi *peer*-nek?
2. Részvétel ösztönzése
   - Miért segítenének egymásnak a *peer*-ek?
3. Decentralizálás
   - Hogy találják meg a peer-ek a tartalmat?

- Peer-ek kettős szerepe:
  1. Feltöltés a többiek segítésére.
  2. Letöltés saját magának.

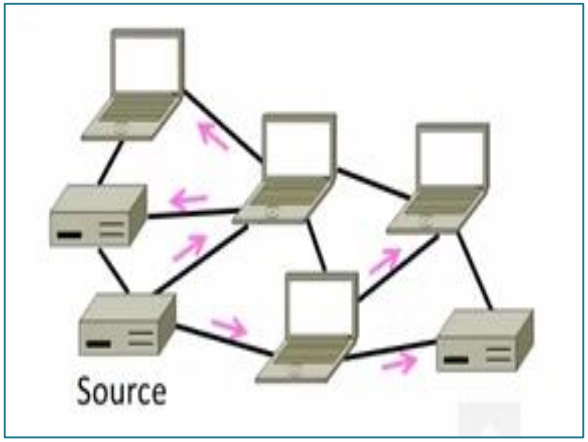


p2p megosztási fa
Forrás: [2]

p2p tartalom megosztás 3/3

P2P DECENTRALIZÁLÁS MEGVALÓSÍTÁSA

- A *peer* meg kell tanulja a tartalom helyét. (***D****istributed* ***H****ash* ***T****ables*)
- A DHTs teljesen decentralizált, hatékony algoritmusok egy elosztott indexhez.
  - Az index a *peer*-ek között terjed.
  - Az index listázza a tartalommal rendelkező *peer*-eket.
  - Bármely *peer* kikeresheti az indexet.
  - Tudományos munkaként indult 2001-ben.



p2p megosztási fa
Forrás: [2]

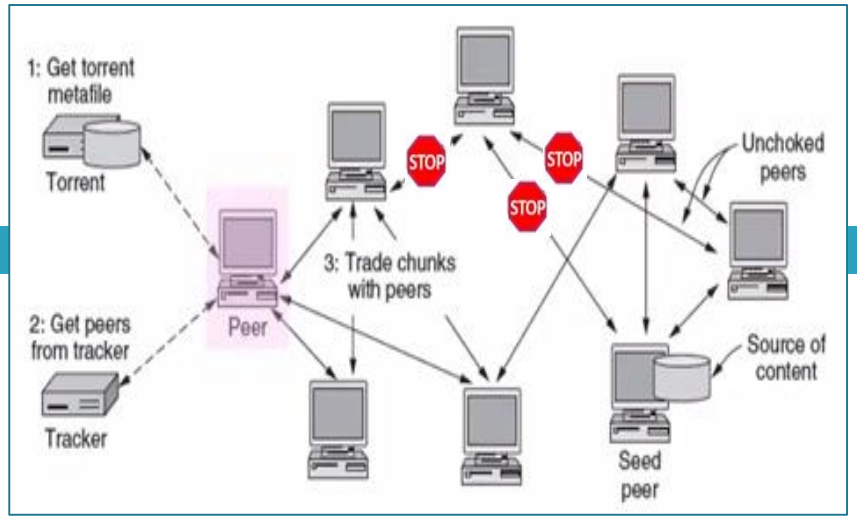
BitTorrent

- A legelterjedtebb p2p rendszer, amit:
  - 2001-ben fejlesztette ki Bram Cohen.
  - Nagyon gyorsan elterjedt, és jelenleg az Internet forgalom nagy részét teszi ki a BitTorrent forgalom.
  - Legális és illegális tartalmak megosztására is használnak.
- Az adatkézbesítés „torrent”-ek segítségével történik:
  - A fájlok megosztása darabonként történik.
  - Az ösztönzés céljára figyelemreméltó módszert alkalmaz.
  - Tracker-ek vagy decentralizált indexek használata.

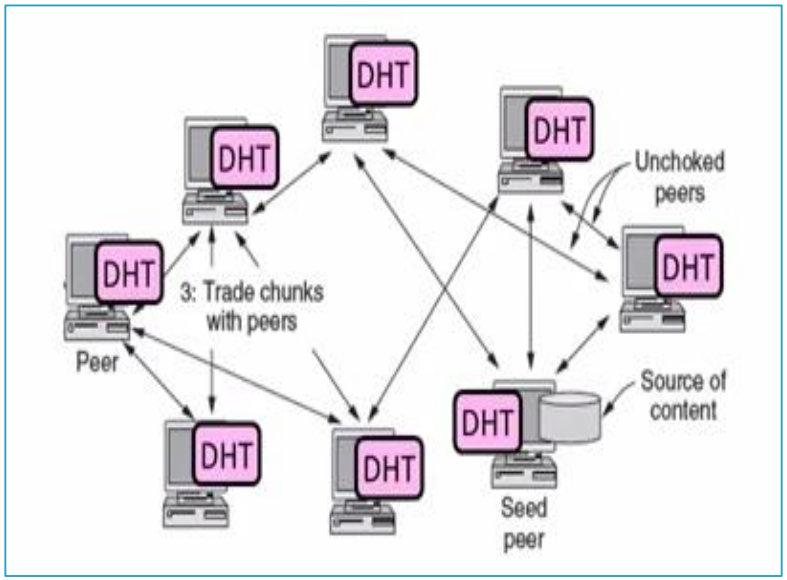
BitTorrent protokoll

EGY TORRENT LETÖLTÉSÉNEK LÉPÉSEI

1. A *torrent* leírásával kezdődik.
2. Két lehetőség van:
   a) Kapcsolatba lépni a *tracker*-rel a csatlakozáshoz, és elkérni a *peer*-ek listáját, amelyen legalább egy *seed peer* is van. (régi)
      - **seed peer** – Olyan speciális peer, aki rendelkezik a letöltendő fájl összes darabjával.
      - **leech peer** – Olyan peer, aki fel és le is tölt, azaz nem rendelkezik az összes darabbal.
   b) Vagy *DHT* index használata a *peer*-ekhez. (új)
3. A különböző peer-ekkel forgalom lebonyolítása.
4. Előnybe részesítjük azon *peer*-eket, akik gyorsan töltenek fel a részünkre.
   - **choke peer** – Olyan peer, aki korlátozza a letöltést más peer-ek részére.



BitTorrent tracker-rel (példa)
Forrás: [6]



BitTorrent DHT-val (példa)
Forrás: [2]

Köszönöm a figyelmet!