# WWW Case Study

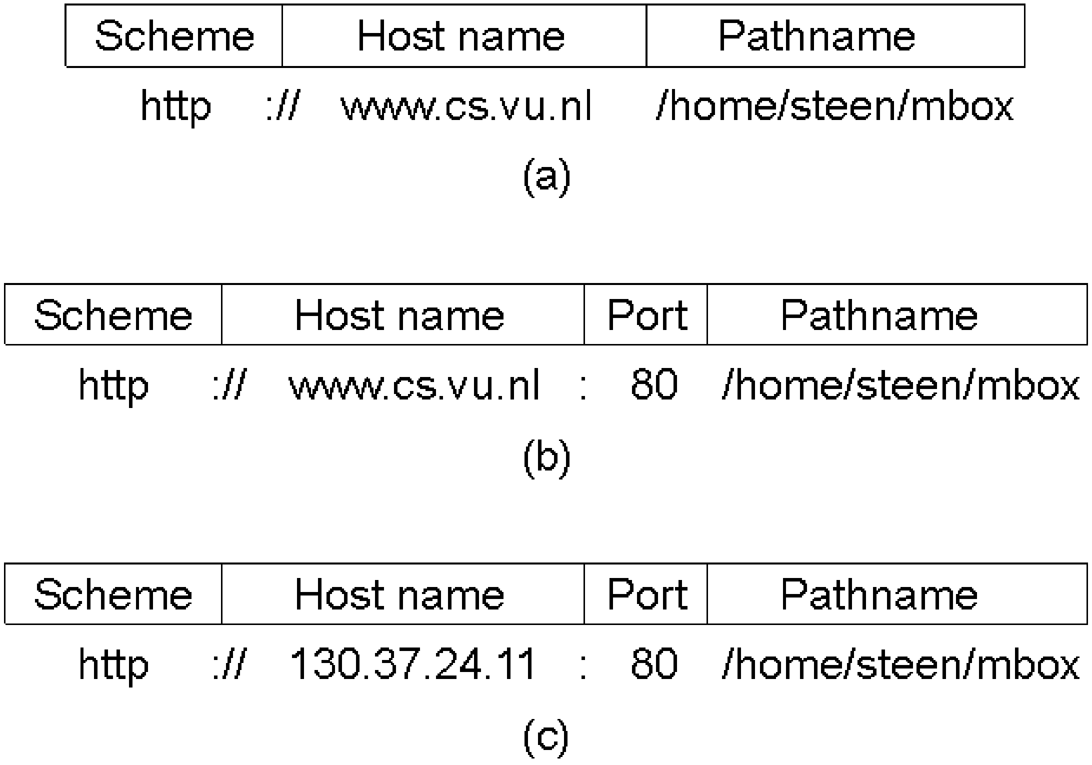
Jonathan Walpole

CSE515 Distributed Systems

# WWW Concepts

- What is the conceptual model?
- How are documents named?

# Uniform Resource Locators



| Scheme | Host name | Pathname |
|---|---|---|

http :// www.cs.vu.nl /home/steen/mbox

(a)

| Scheme | Host name | Port | Pathname |
|---|---|---|---|

http :// www.cs.vu.nl : 80 /home/steen/mbox

(b)

| Scheme | Host name | Port | Pathname |
|---|---|---|---|

http :// 130.37.24.11 : 80 /home/steen/mbox

(c)

Often-used structures for URLs.

a. Using only a DNS name.

b. Combining a DNS name with a port number

c. Combining an IP address with a port number.

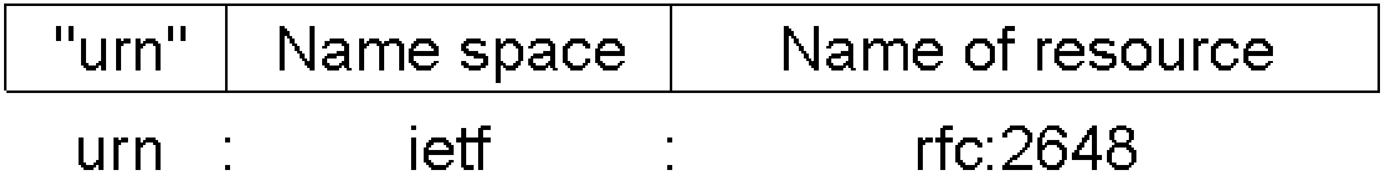
# Uniform Resource Locators (2)

| •Name | •Used for | •Example |
|---|---|---|
| •http | •HTTP | •http://www.cs.vu.nl:80/globe |
| •ftp | •FTP | •ftp://ftp.cs.vu.nl/pup/minx/README |
| •file | •Local file | •file:/edu/book/work/chp/11/11 |
| •data | •Inline data | •data:text/plain;charset=iso-8859-7,%e1%e2%e3 |
| •telnet | •Remote login | •telnet://flits.cs.vu.nl |
| •tel | •Telephone | •tel:+31201234567 |
| •modem | •Modem | •modem:+31201234567;type=v32 |

- Examples of URLs.

# Uniform Resource Names

- The general structure of a URN

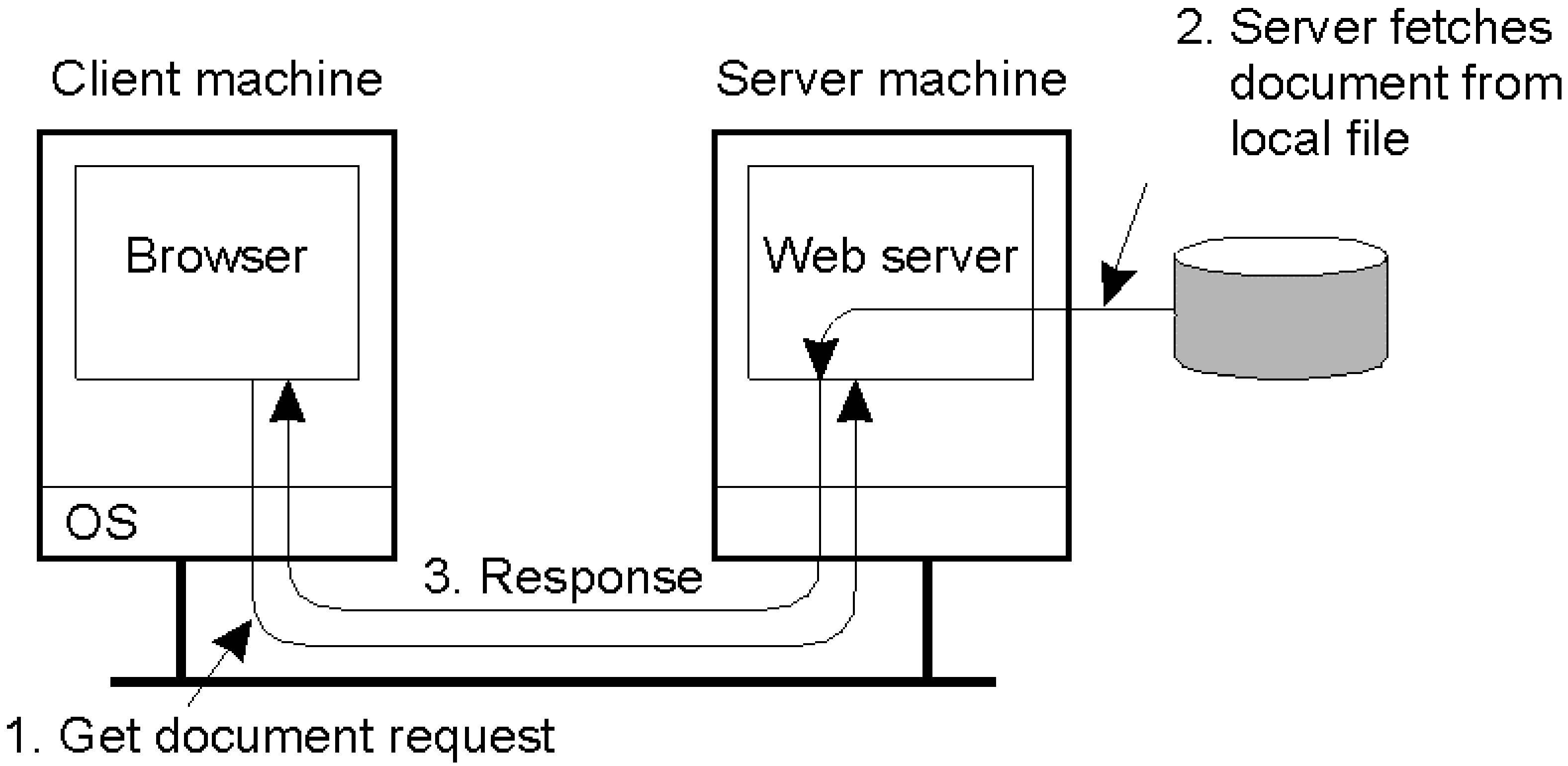


| "urn" |  | Name space |  | Name of resource |
|---|---|---|---|---|
| urn | : | ietf | : | rfc:2648 |

# WWW Concepts

- What is the architectural model?

# Simple Client Server Architecture



# WWW Concepts

- How are documents processed on the client?
- What is HTML?
- What is a client-side script?
- What is DOM and why do you need it?

# Document Model (1)

```
<HTML>                                                  <!- Start of HTML document     -->
<BODY>                                                  <!- Start of the main body      -->
<H1>Hello World/H1>                                     <!- Basic text to be displayed   -->
<P>                                                     <!- Start of a new paragraph   -->
<SCRIPT type = "text/javascript">                       <!- identify scripting language -->
  document.writeln ("<H1>Hello World</H1>;              // Write a line of text
</SCRIPT>                                               <!- End of scripting section     -->
</P>                                                    <!- End of paragraph section -->
</BODY>                                                 <!- End of main body              -->
</HTML>                                                 <!- End of HTML section          -->
```

- A simple Web page embedding a script written in JavaScript.

# WWW Concepts

- What is XML?

# Document Model (2)

```
(1)      <!ELEMENT article (title, author+,journal)>
(2)      <!ELEMENT title (#PCDATA)>
(3)      <!ELEMENT author (name, affiliation?)>
(4)      <!ELEMENT name (#PCDATA)>
(5)      <!ELEMENT affiliation (#PCDATA)>
(6)      <!ELEMENT journal (jname, volume, number?, month? pages, year)>
(7)      <!ELEMENT jname (#PCDATA)>
(8)      <!ELEMENT volume (#PCDATA)>
(9)      <!ELEMENT number (#PCDATA)>
(10)     <!ELEMENT month (#PCDATA)>
(11)     <!ELEMENT pages (#PCDATA)>
(12)     <!ELEMENT year (#PCDATA)>
```

- An XML definition for referring to a journal article.

# Document Model (3)

```
<?xml = version "1.0">
<!DOCTYPE article SYSTEM "article.dtd">
<article>
    <title> Prudent Engineering Practice for Cryptographic Protocols</title>
    <author><name>M. Abadi</name></author>
    <author><name>R. Needham</name></author>
    <journal>
            <jname>IEEE Transactions on Software Engineering</jname>
            <volume>22</volume>
            <number>12</number>
            <month>January</month>
            <pages>6 – 15</pages>
            <year>1996</year>
    </journal>
</article>
```

- An XML document using the XML definitions from previous slide

# WWW Concepts

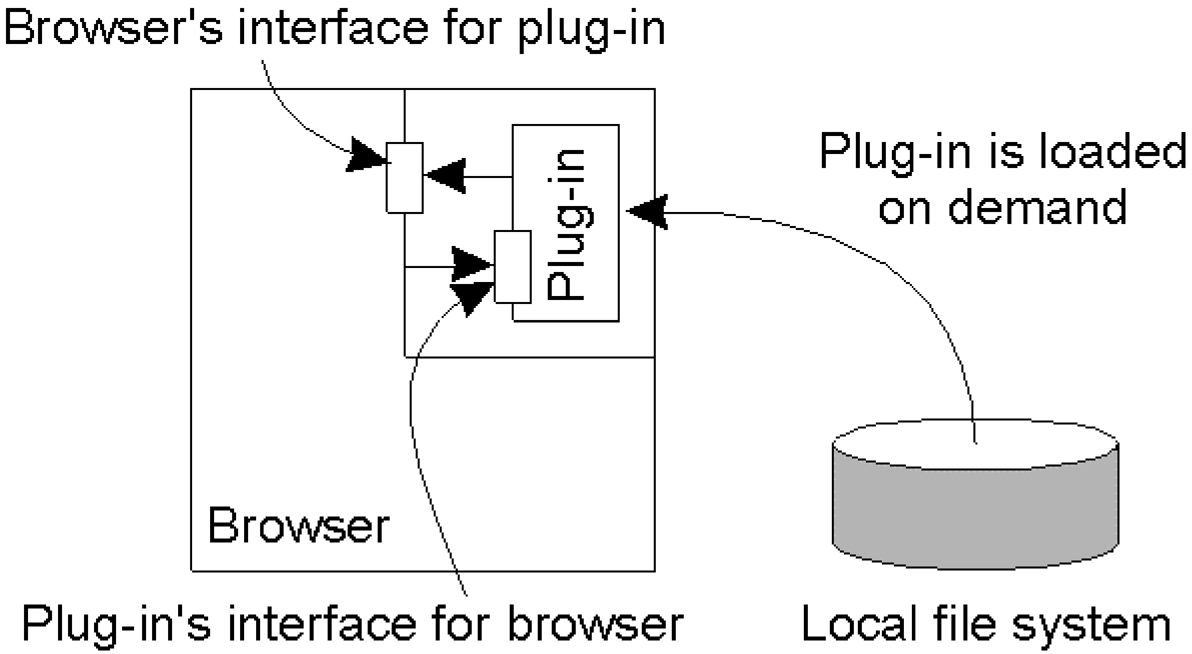
- What are MIME types?
- What are browser plug-ins?

# Document Types

| •**Type** | •**Subtype** | •**Description** |
|---|---|---|
| •Text | •Plain | •Unformatted text |
| | •HTML | •Text including HTML markup commands |
| | •XML | •Text including XML markup commands |
| •Image | •GIF | •Still image in GIF format |
| | •JPEG | •Still image in JPEG format |
| •Audio | •Basic | •Audio, 8-bit PCM sampled at 8000 Hz |
| | •Tone | •A specific audible tone |
| •Video | •MPEG | •Movie in MPEG format |
| | •Pointer | •Representation of a pointer device for presentations |
| •Application | •Octet-stream | •An uninterrupted byte sequence |
| | •Postscript | •A printable document in Postscript |
| | •PDF | •A printable document in PDF |
| •Multipart | •Mixed | •Independent parts in the specified order |
| | •Parallel | •Parts must be viewed simultaneously |

- Six top-level MIME types and some common subtypes.

# Clients (1)

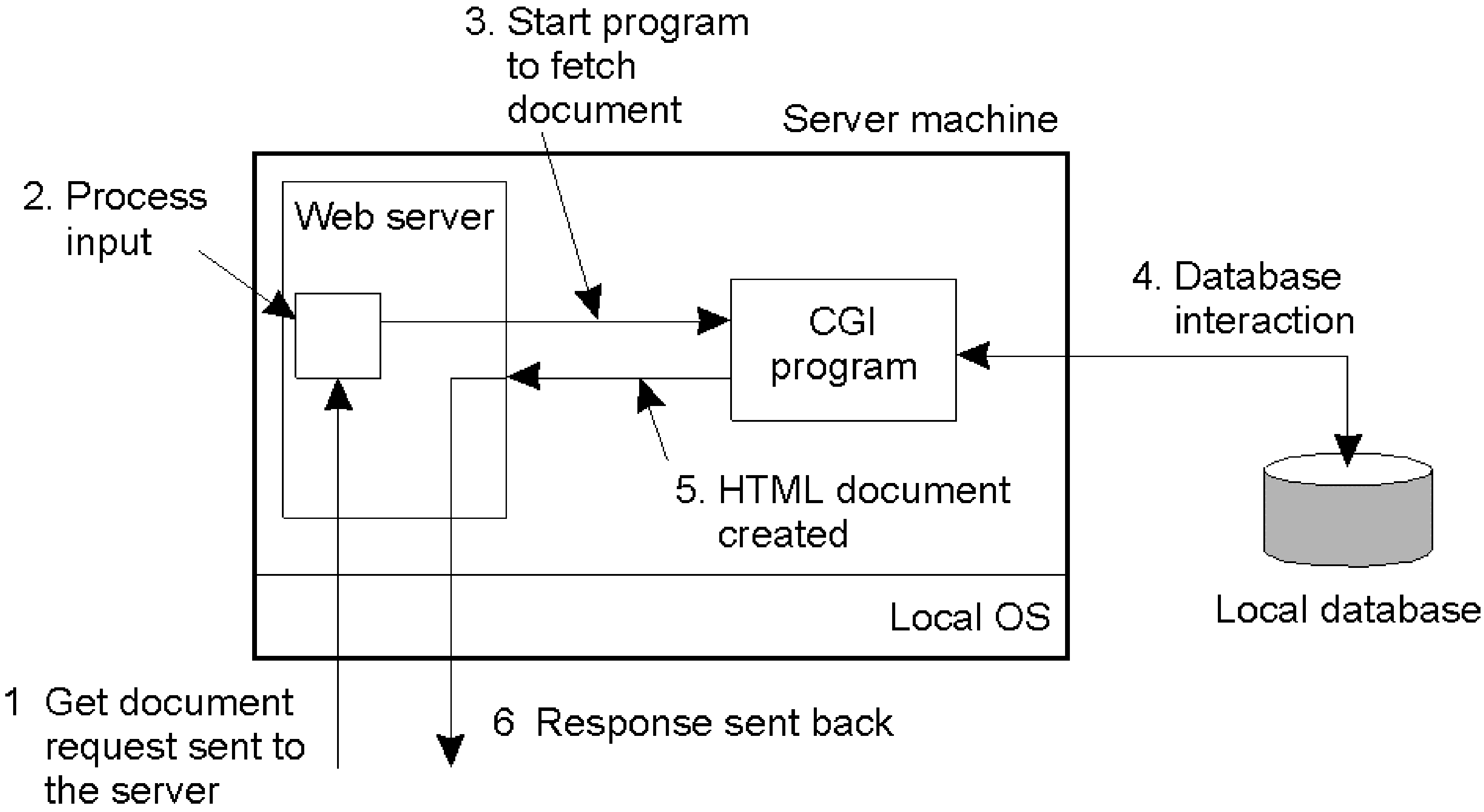

Browser's interface for plug-in
Plug-in
Plug-in is loaded on demand
Browser
Plug-in's interface for browser
Local file system

- Using a plug-in in a Web browser.

# WWW Concepts

- What is CGI?
- How are CGI programs invoked?
- What is the difference between a CGI program and a browser plug-in?

# Architectural Overview (1)



# WWW Concepts

- What is a server script?
- How does a server recognize it?
- What do clients do with server scripts?
- What is the difference between a server script and a CGI program?

# Architectural Overview (2)

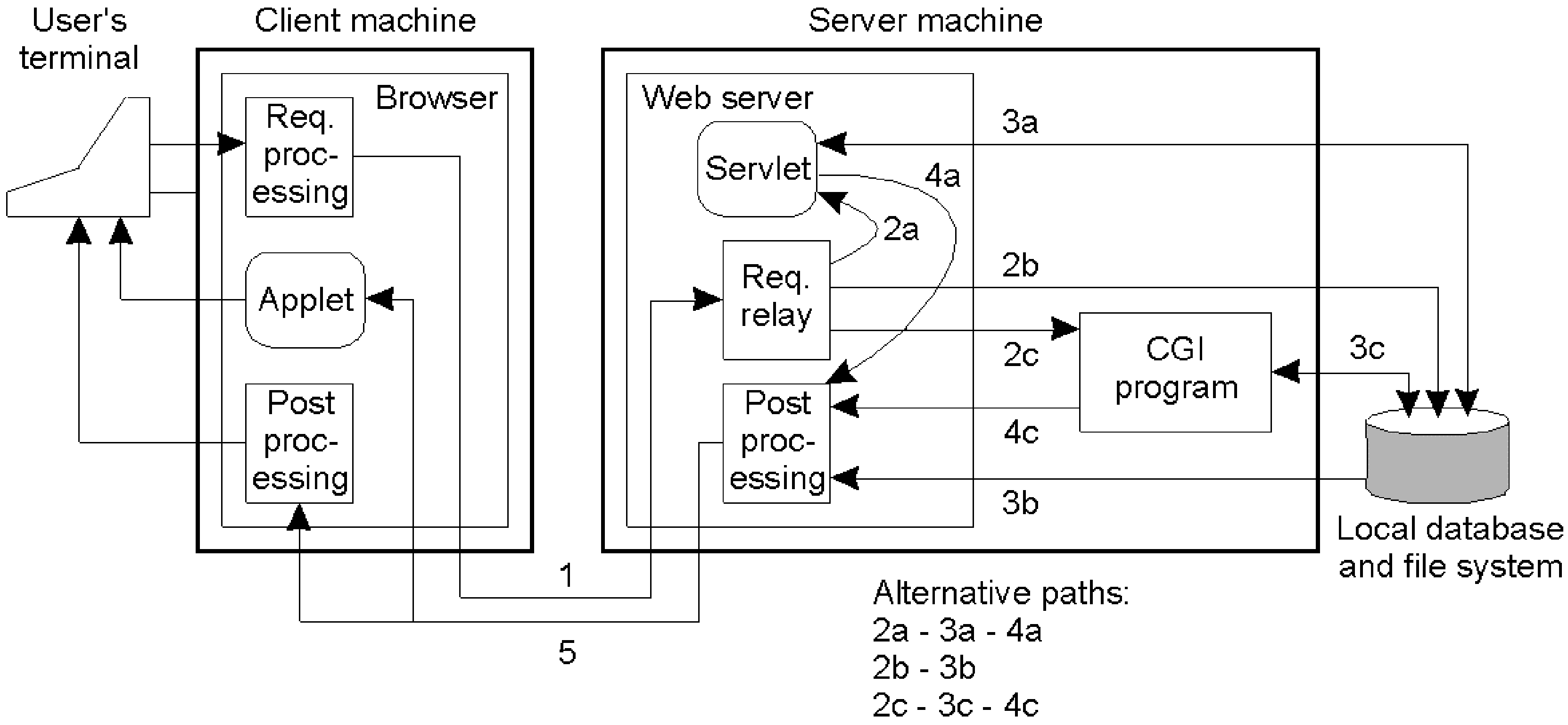
```
<HTML>
<BODY>
<P>The current content of <pre>/data/file.txt</PRE>is:</P>
<P>
<SERVER type = "text/javascript");
    clientFile = new File("/data/file.txt");
    if(clientFile.open("r")){
             while (!clientFile.eof())
                 document.writeln(clientFile.readln());
             clientFile.close();
    }
</SERVER>
</P>
<P>Thank you for visiting this site.</P>
</BODY>
</HTML>
```

- An HTML document containing a JavaScript to be executed by the server

# WWW Concepts

- What is an applet?
- How is an applet invoked?
- What is the difference between an applet and a browser plug-in?
- What is a servlet?
- How is a servlet invoked?
- What is the difference between a servlet and a CGI program?

# Architectural Overview (3)



- Architectural details of a client and server in the Web.

# WWW Concepts

- What communication protocols does the web use?
- What methods does HTTP support?

# HTTP Methods

- Some example operations supported by HTTP.

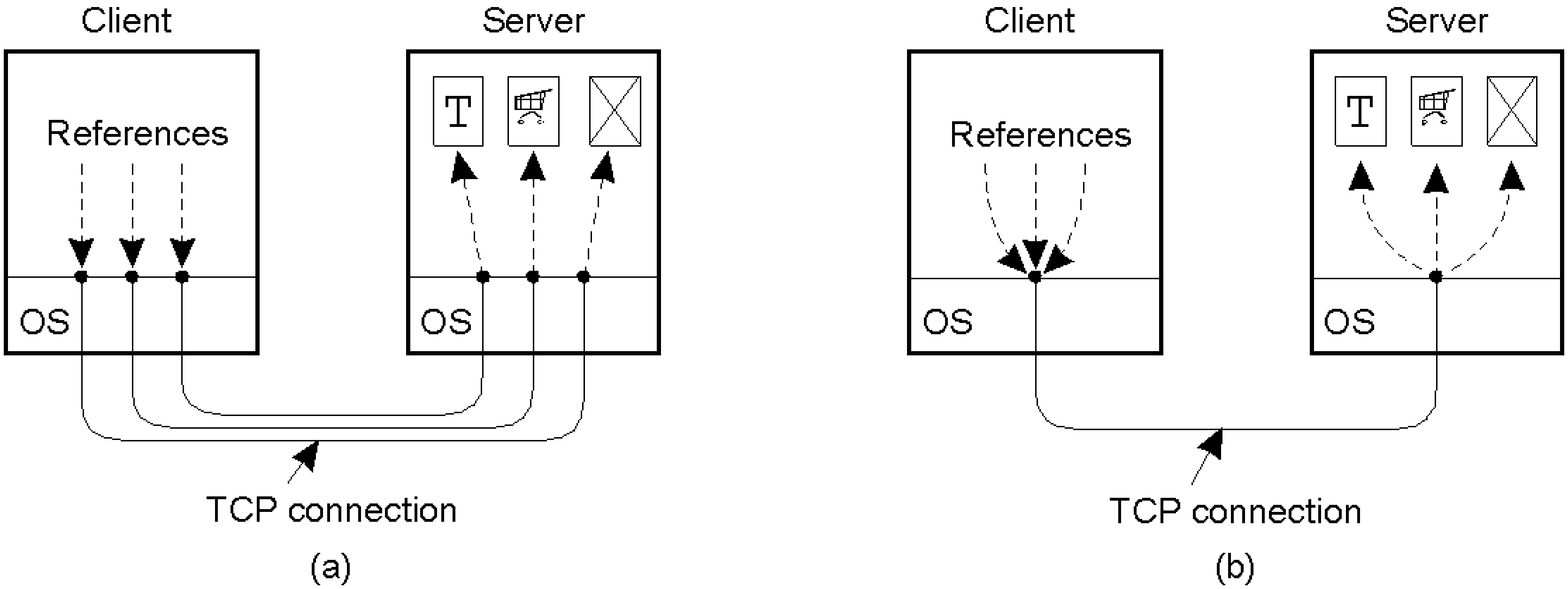
| •**Operation** | •**Description** |
|---|---|
| •Head | •Request to return the header of a document |
| •Get | •Request to return a document to the client |
| •Put | •Request to store a document |
| •Post | •Provide data that is to be added to a document (collection) |
| •Delete | •Request to delete a document |

# WWW Concepts

- Why was HTTP 1.0’s use of TCP inefficient?
- What is the difference between persistent and no-persistent TCP connections?
- What are parallel, non-persistent connections?
- What is pipelining?

# HTTP Connections

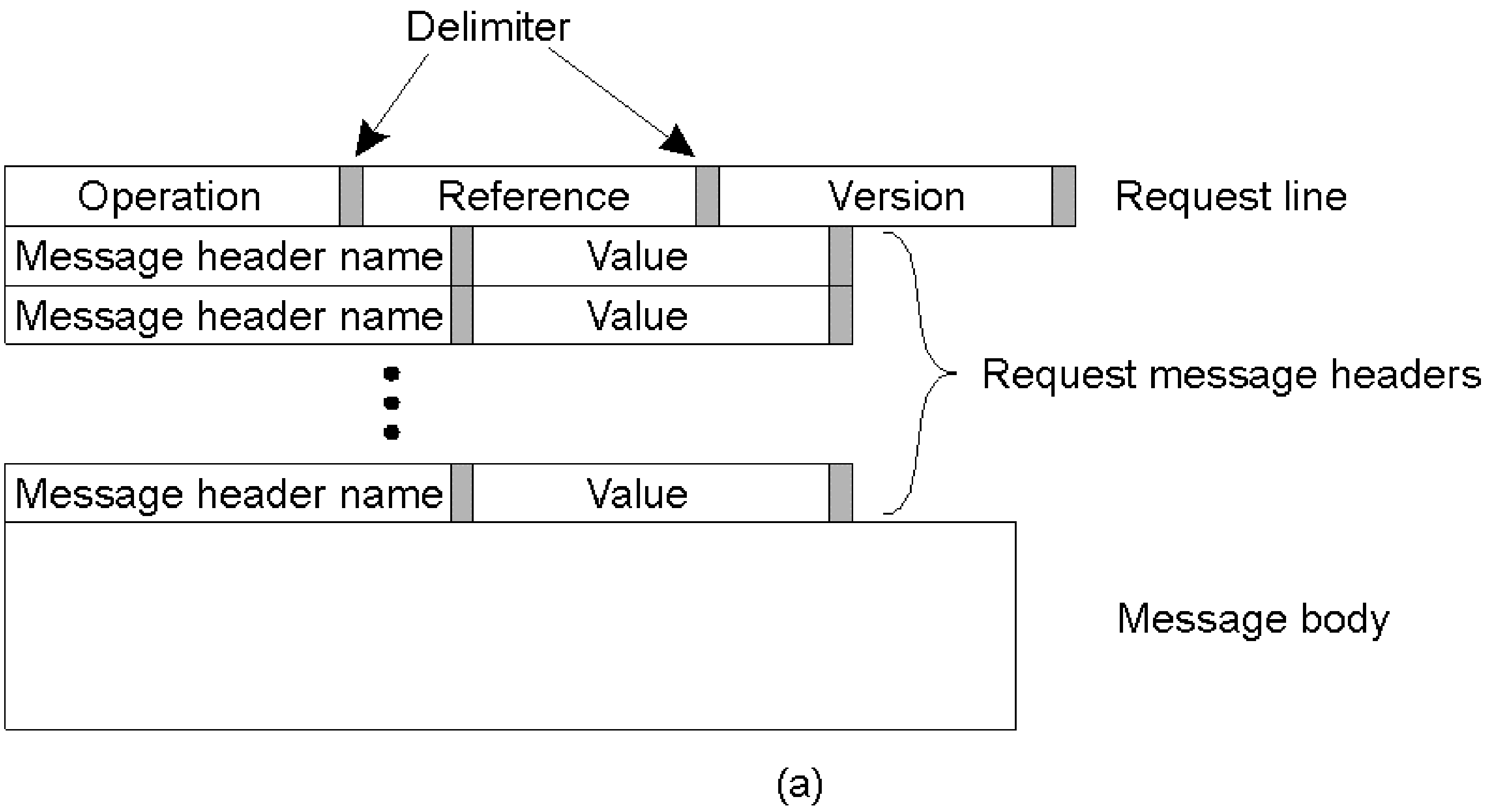
- Using nonpersistent connections.



# WWW Concepts

- How does client-side caching work?
- What is a web proxy?
- What are the consistency semantics?
- How does expiry-based caching work?

# HTTP Messages (1)



Delimiter

Operation | Reference | Version — Request line

Message header name | Value

Message header name | Value

Message header name | Value

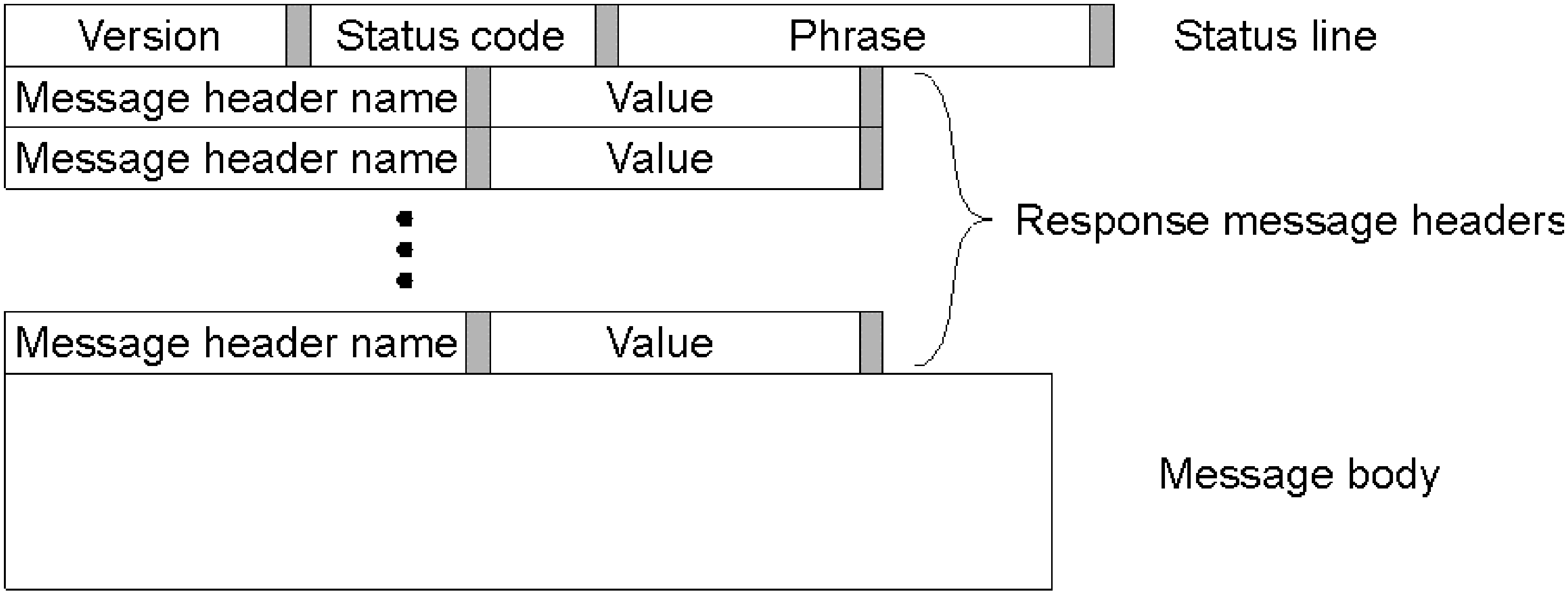
Request message headers

Message body

(a)

- HTTP request message

# HTTP Messages (2)



(b)

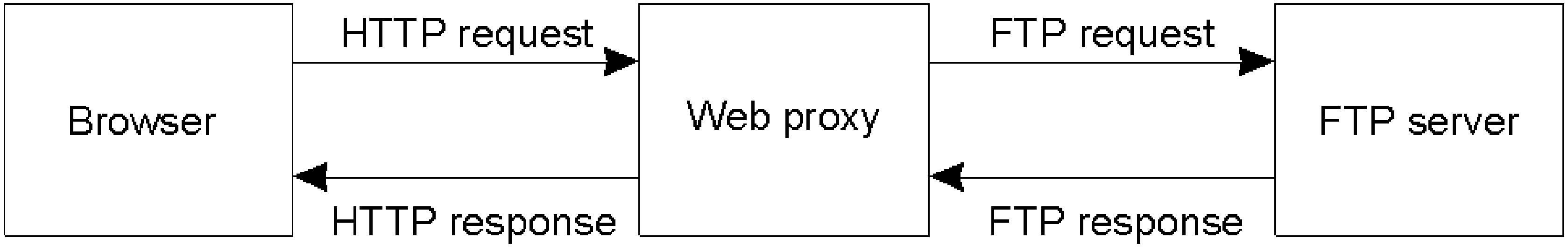
- HTTP response message.

# HTTP Messages (3)

- Some HTTP message headers.

| •Header | •Source | •Contents |
|---|---|---|
| •Accept | •Client | •The type of documents the client can handle |
| •Accept-Charset | •Client | •The character sets are acceptable for the client |
| •Accept-Encoding | •Client | •The document encodings the client can handle |
| •Accept-Language | •Client | •The natural language the client can handle |
| •Authorization | •Client | •A list of the client's credentials |
| •WWW-Authenticate | •Server | •Security challenge the client should respond to |
| •Date | •Both | •Date and time the message was sent |
| •ETag | •Server | •The tags associated with the returned document |
| •Expires | •Server | •The time how long the response remains valid |
| •From | •Client | •The client's e-mail address |
| •Host | •Client | •The TCP address of the document's server |
| •If-Match | •Client | •The tags the document should have |
| •If-None-Match | •Client | •The tags the document should not have |
| •If-Modified-Since | •Client | •Tells the server to return a document only if it has been modified since the specified time |
| •If-Unmodified-Since | •Client | •Tells the server to return a document only if it has not been modified since the specified time |
| •Last-Modified | •Server | •The time the returned document was last modified |
| •Location | •Server | •A document reference to which the client should redirect its request |
| •Referer | •Client | •Refers to client's most recently requested document |
| •Upgrade | •Both | •The application protocol the sender wants to switch to |
| •Warning | •Both | •Information about the status of the data in the message |

# Clients (2)

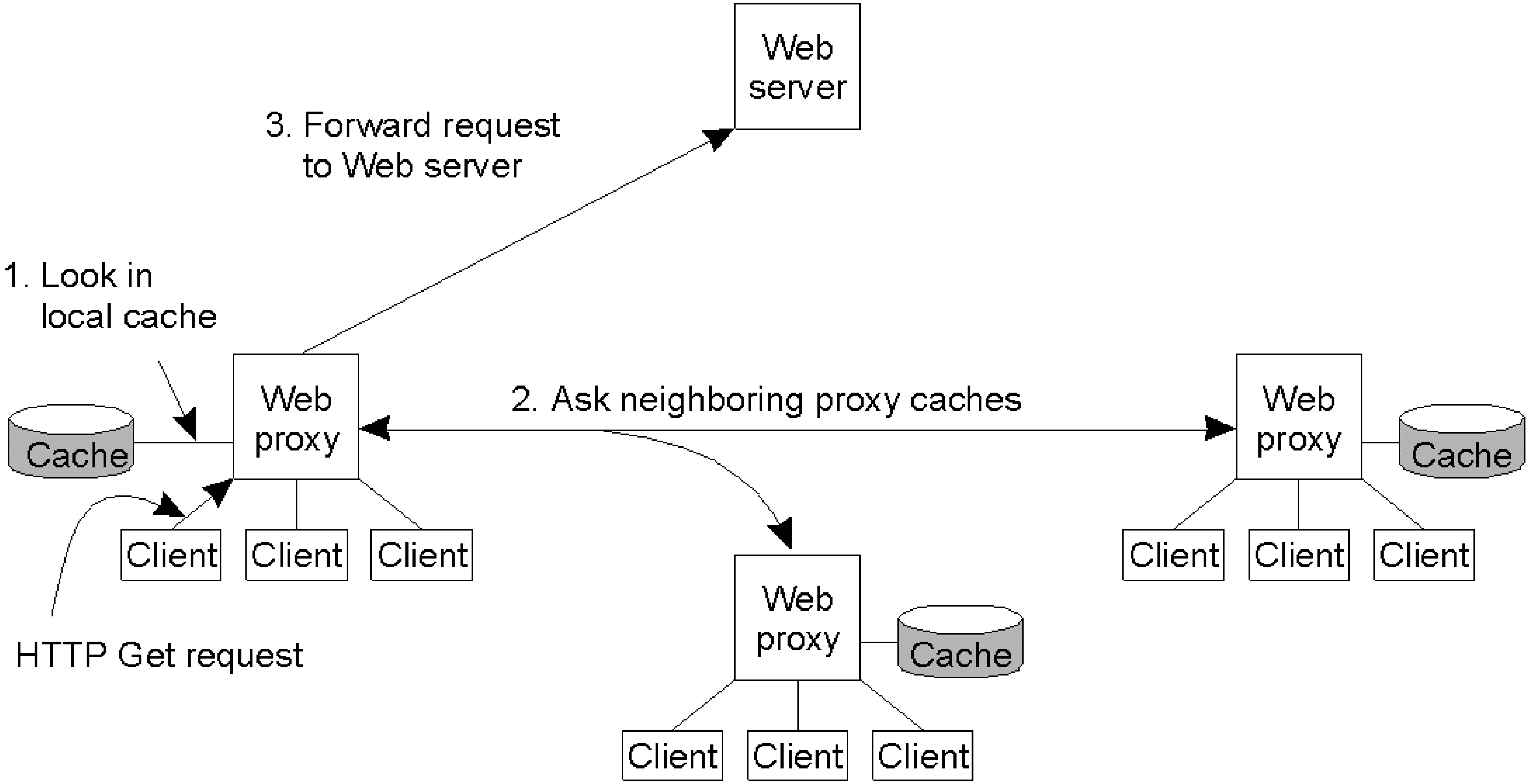


- Using a Web proxy when the browser does not speak FTP.

# WWW Concepts

- What are hierarchical and cooperative caching?
- Why do they lead to higher latency?
- What about dynamic documents?
- Why is server replication becoming  more popular than caching?

# Web Proxy Caching

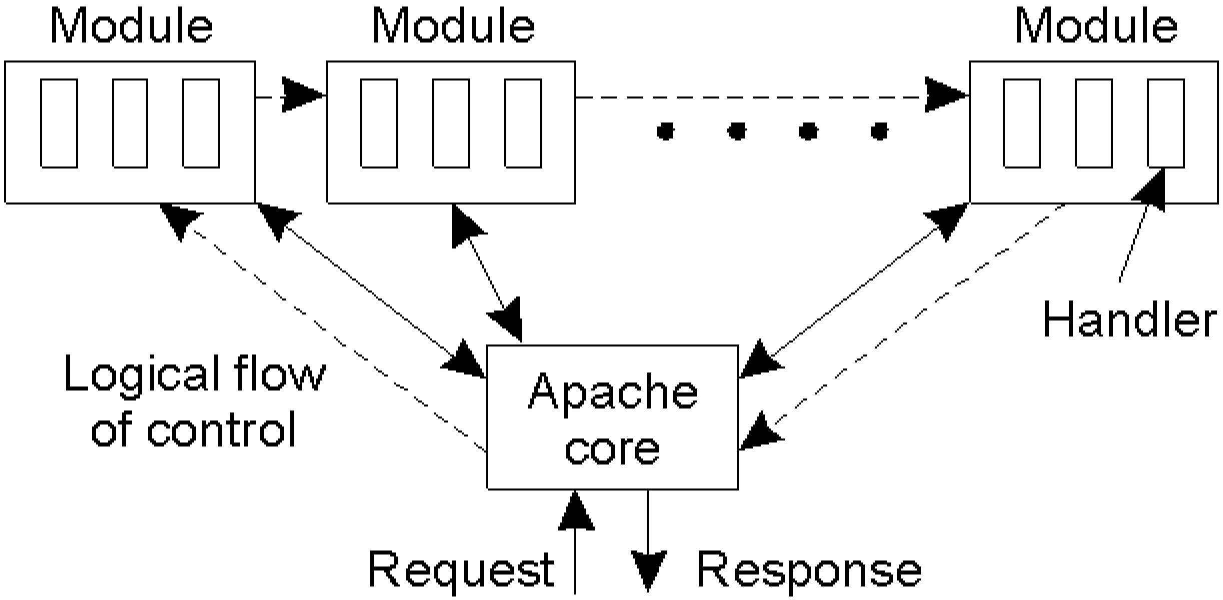


# WWW Concepts

- Why structure web servers as a pipeline of processing steps (such as Apache handlers)?

# Servers

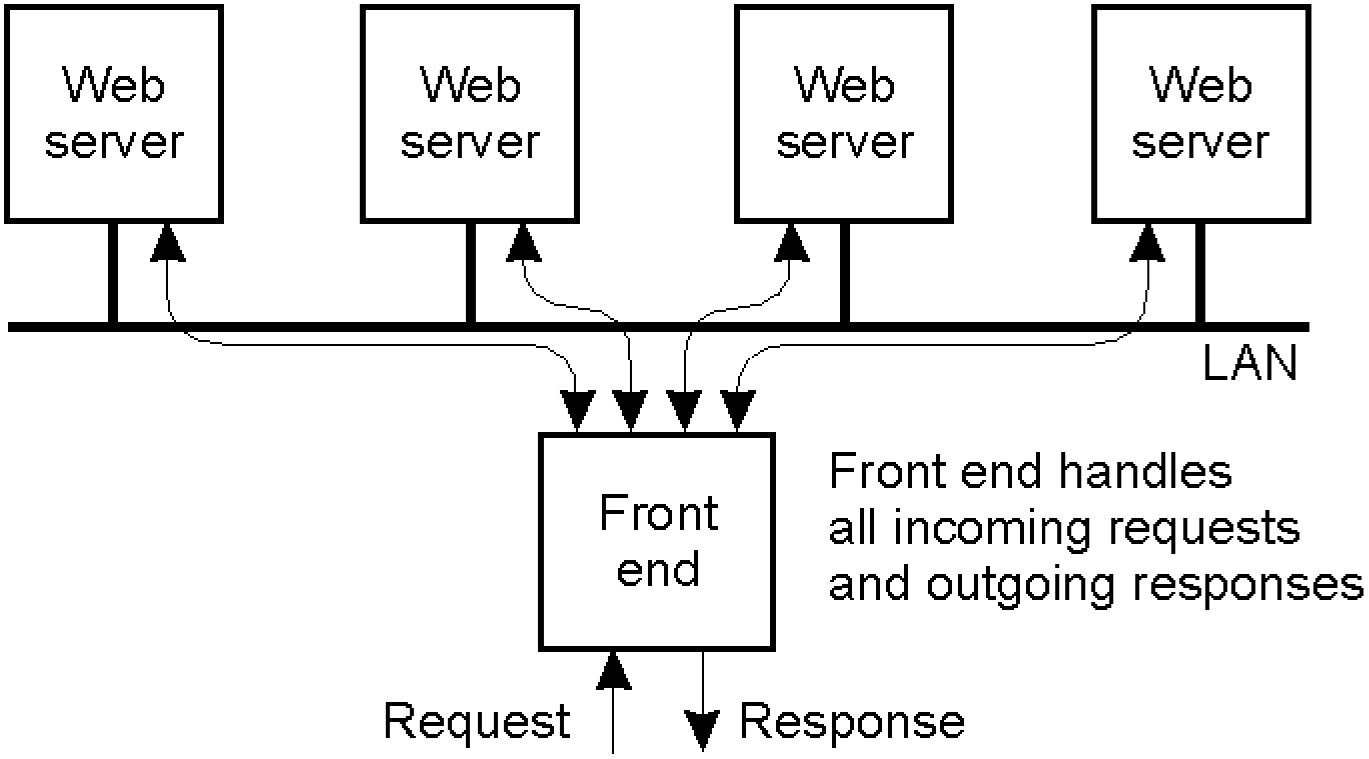
- General organization of the Apache Web server.



# WWW Concepts

- How can we build scalable web servers?
- How would you structure a server cluster?

# Server Clusters (1)



- The principle of using a cluster of workstations to implement a Web service.

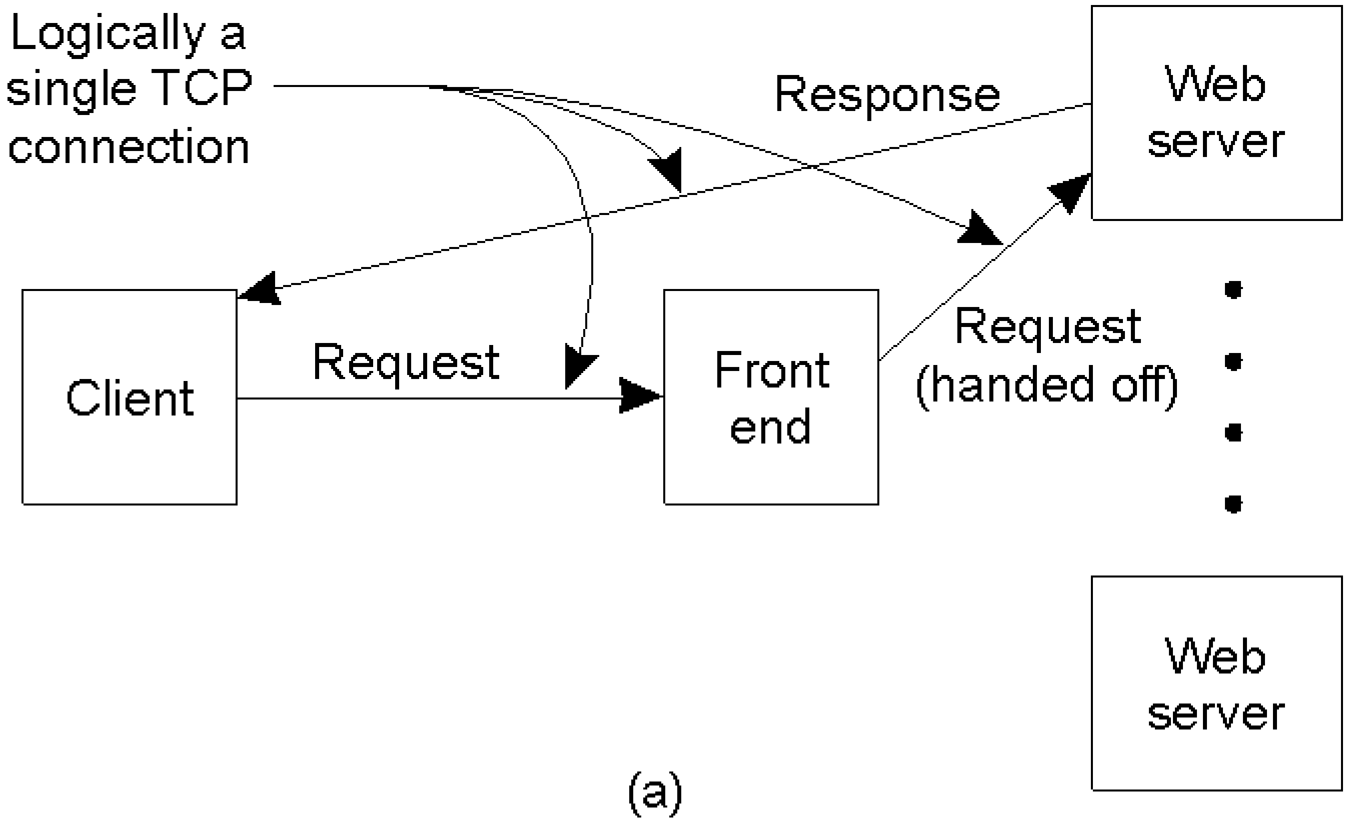
# WWW Concepts

- How can you prevent the front end from becoming a bottleneck?
- How does a transport-layer switching differ from application-layer (content-aware) switching?
- What are the relative advantages and disadvantages?

# WWW Concepts

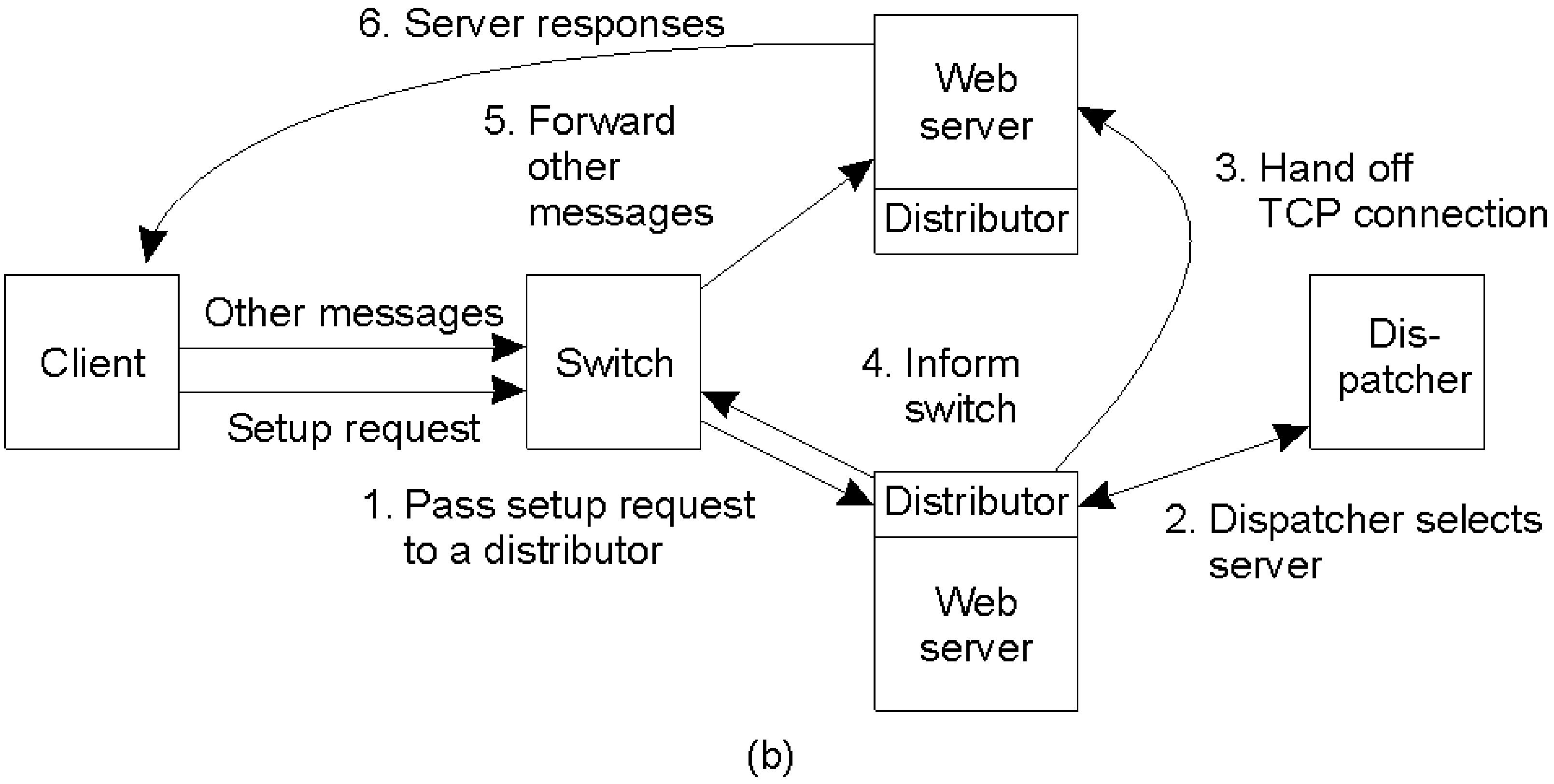
- What is TCP hand-off?
- Why does it improve scalability?

# Server Clusters (2)



- The principle of TCP handoff.

# Server Clusters (3)

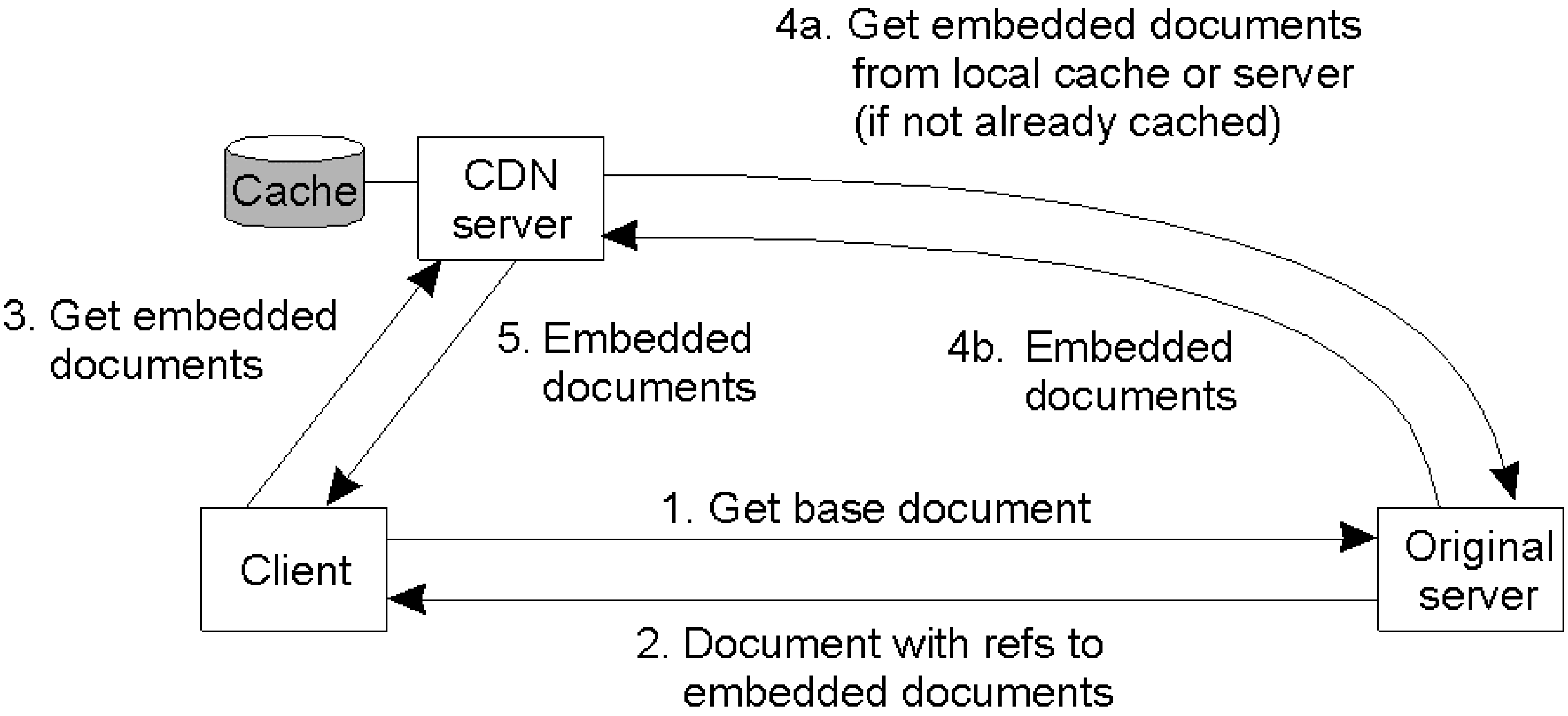


- A scalable content-aware cluster of Web servers.

# WWW Concepts

- What is a content distribution network (CDN)?
- How does a CDN discover server replicas close to the client?
- How does it redirect current and subsequent requests to that replica?
- How does this approach relate to DNS load balancing?

# Server Replication


4a. Get embedded documents from local cache or server (if not already cached)
Cache
CDN server
3. Get embedded documents
5. Embedded documents
4b. Embedded documents
1. Get base document
Client
Original server
2. Document with refs to embedded documents

- The principle working of the Akami CDN.

# Spare Slides

# Building scalable web services

- A relatively easy problem….
  - Why?
    - HTTP: stateless, request-response protocol
    - decoupled, independent requests
  - How?
    - divide and conquer
    - replicate, partition, distribute, load balance

# Outline

- Application layer tricks
  - explicit server partitioning
  - dynamic name resolution
- Transparent networking tricks
  - virtual servers
- Case studies
  - scalable content delivery (Yahoo!)
  - content transformation engines
  - transparent web caches
  - scalable secure servers

# Explicit server partitioning (static)

- Run a new server per resource/service
- Example
  - www.blah.com
  - mail.blah.com
  - images.blah.com
  - shopping.blah.com
  - my.blah.com
  - etc. etc.

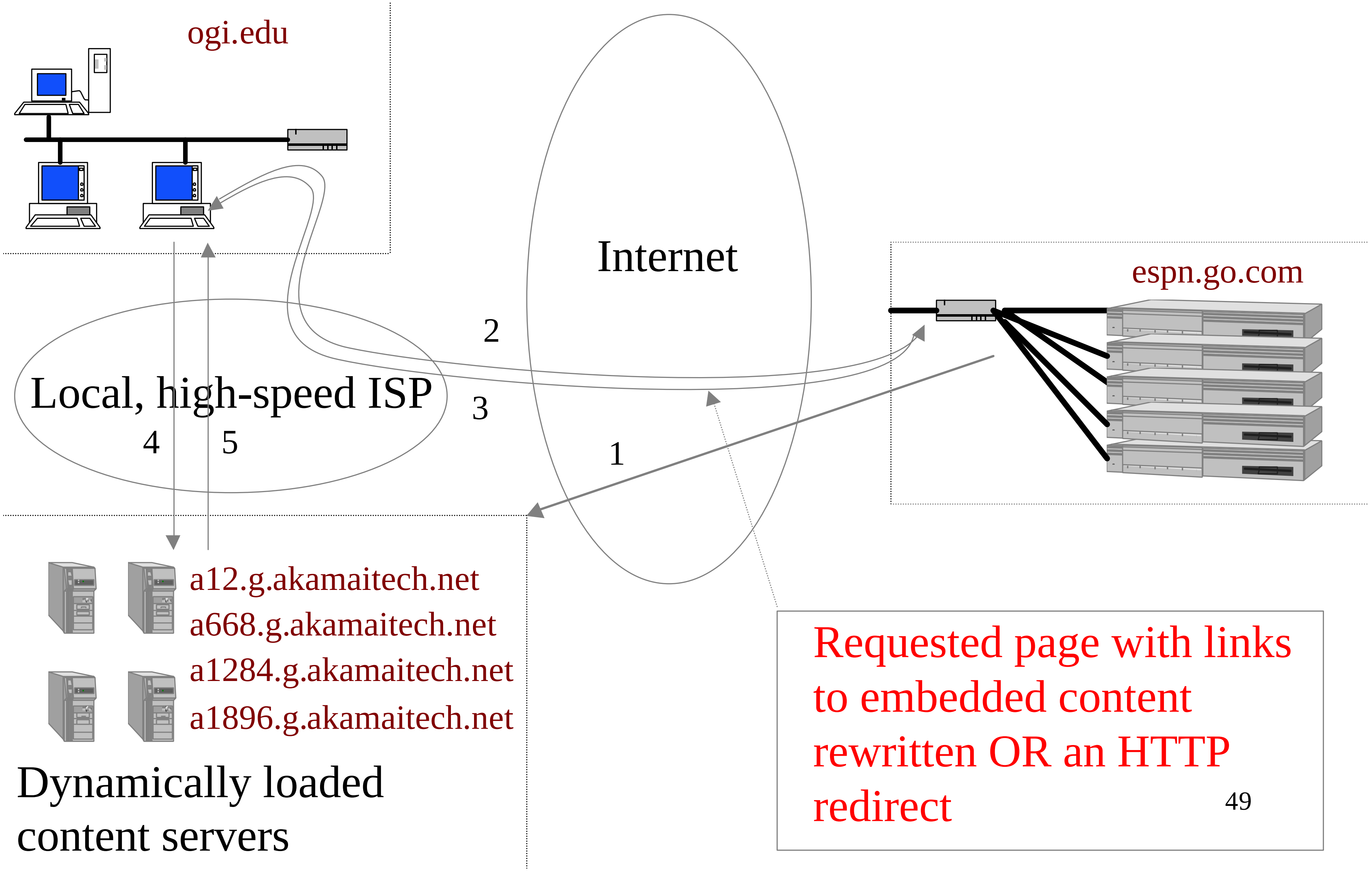
# Explicit server partitioning (static)

- Advantages
  - better disk utilization
  - better cache performance
  - protection against DOS
- Disadvantages
  - lower peak capacity
  - coarse load balancing across servers/services
  - management costs

# Explicit server partitioning (dynamic)

- Basis for CDNs (Content Distribution Networks)
- Active “forward deployment” of content to explicitly named servers near client
- Redirect requests from origin servers by
  - HTTP redirects
  - dynamic URL rewriting of embedded content
- Application-level multicast based on geographic information
- Akamai, Digital Island (Sandpiper), SightPath, Xcelera (Mirror-image), Inktomi

# Explicit server partitioning (dynamic)



ogi.edu

Internet

espn.go.com

Local, high-speed ISP

a12.g.akamaitech.net
a668.g.akamaitech.net
a1284.g.akamaitech.net
a1896.g.akamaitech.net

Dynamically loaded content servers

Requested page with links to embedded content rewritten OR an HTTP redirect

# Explicit server partitioning (dynamic)

- Advantages
  - better network utilization
  - better load distribution
- Disadvantages
  - distributed management costs
  - storage costs
  - currently OK as ($ network bw >> $ storage)

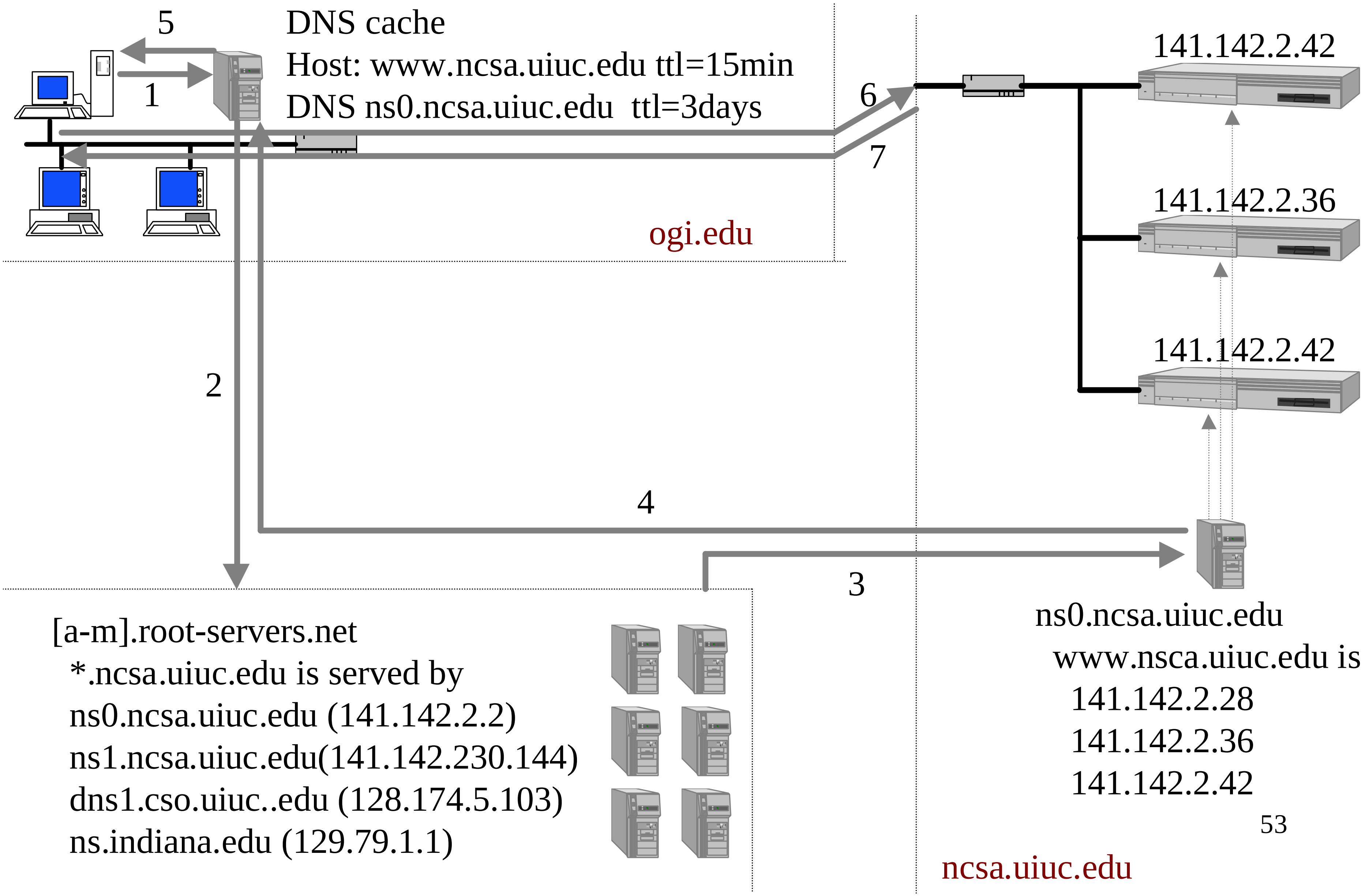
# Outline

- DNS
  - explicit server partitioning
  - transparent name resolution (DNS load balancing)
- Networking tricks
  - virtual servers
- Case studies
  - scalable content delivery (Yahoo!)
  - content transformation engines
  - transparent web caches
  - scalable secure servers

# DNS load balancing

- Popularized by NCSA circa 1993
- Fully replicated server farm
  - Centralized
  - Distributed
- IP address per node
- Adaptively resolve server name (round-robin, load-based, geographic-based)

# DNS load balancing



DNS cache
Host: www.ncsa.uiuc.edu ttl=15min
DNS ns0.ncsa.uiuc.edu ttl=3days

[a-m].root-servers.net
*.ncsa.uiuc.edu is served by
ns0.ncsa.uiuc.edu (141.142.2.2)
ns1.ncsa.uiuc.edu(141.142.230.144)
dns1.cso.uiuc..edu (128.174.5.103)
ns.indiana.edu (129.79.1.1)

ns0.ncsa.uiuc.edu
www.nsca.uiuc.edu is
141.142.2.28
141.142.2.36
141.142.2.42

# DNS load balancing

- Advantages
  - simple, easy to implement
  - uses existing infrastructure
- Disadvantages
  - coarse load balancing
  - local DNS caching affects performance
  - full server replication

# DNS RFCs

- RFC 1794
  - “DNS Support for Load Balancing”
  - http://www.rfc-editor.org/rfc/rfc1794.txt
- RFCs 1034 and 1035 (1987)
  - Replace older DNS RFCs 882 and 883 (1983)
  - http://www.rfc-editor.org/rfc/rfc1034.txt
  - http://www.rfc-editor.org/rfc/rfc1035.txt

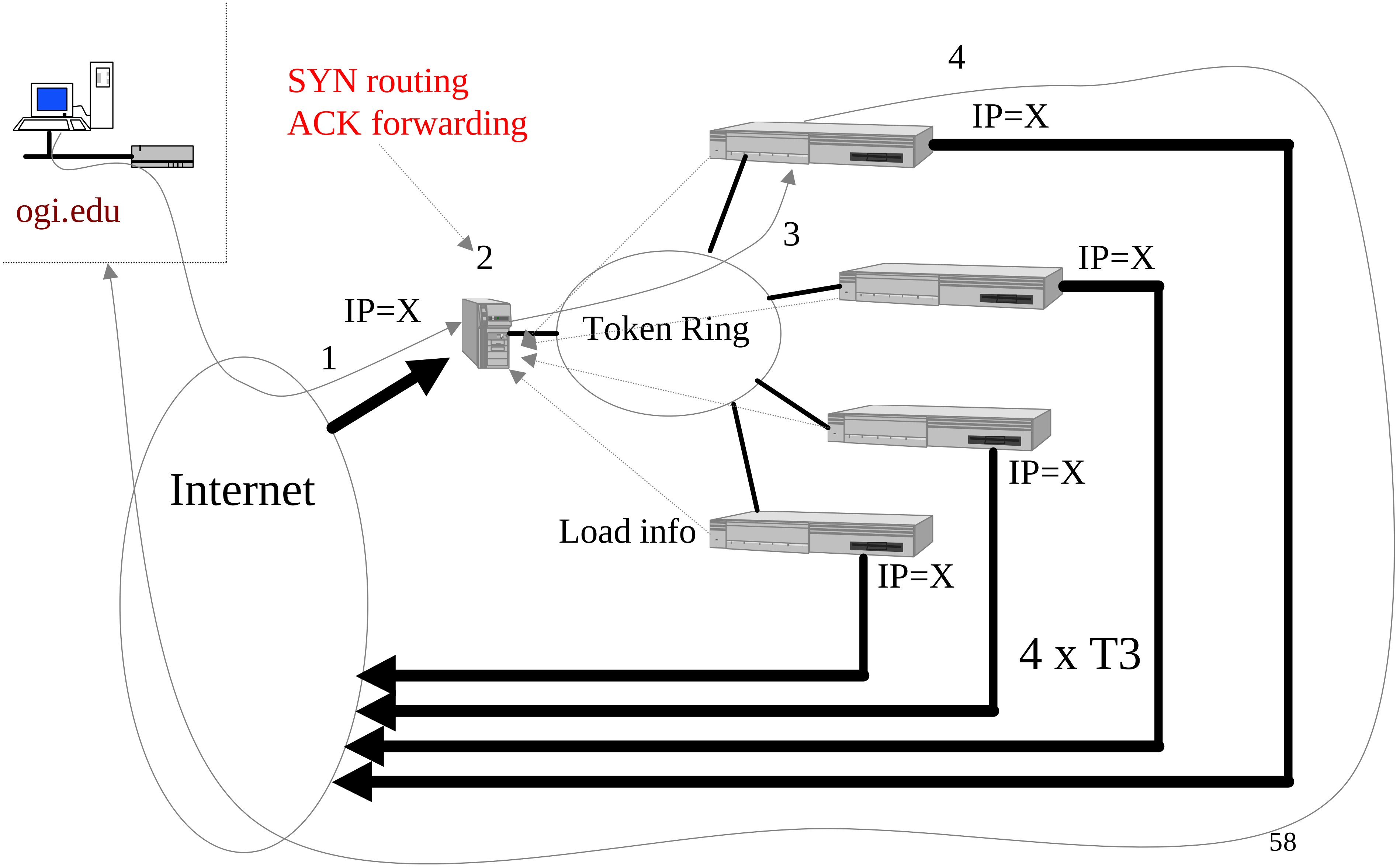
# Outline

- DNS
  - server per resource partitioning
  - dynamic name resolution
- Networking tricks
  - virtual servers
- Case studies
  - scalable content delivery (Yahoo!)
  - content transformation engines
  - transparent web caches
  - scalable secure servers

# Virtual servers

- Large server farm -> single virtual server
- Single front-end for connection routing
- Routing algorithms
  - by load (response times, least connections, server load, weighted round-robin)
  - by layer 3 info (IP addresses)
  - by layer 4 info (ports)
  - by layer 5-7 info (URLs, Cookies, SSL session IDs, User-Agent, client capabilities, etc. etc.)

# Olympic web server (1996)



# Olympic web server (1996)

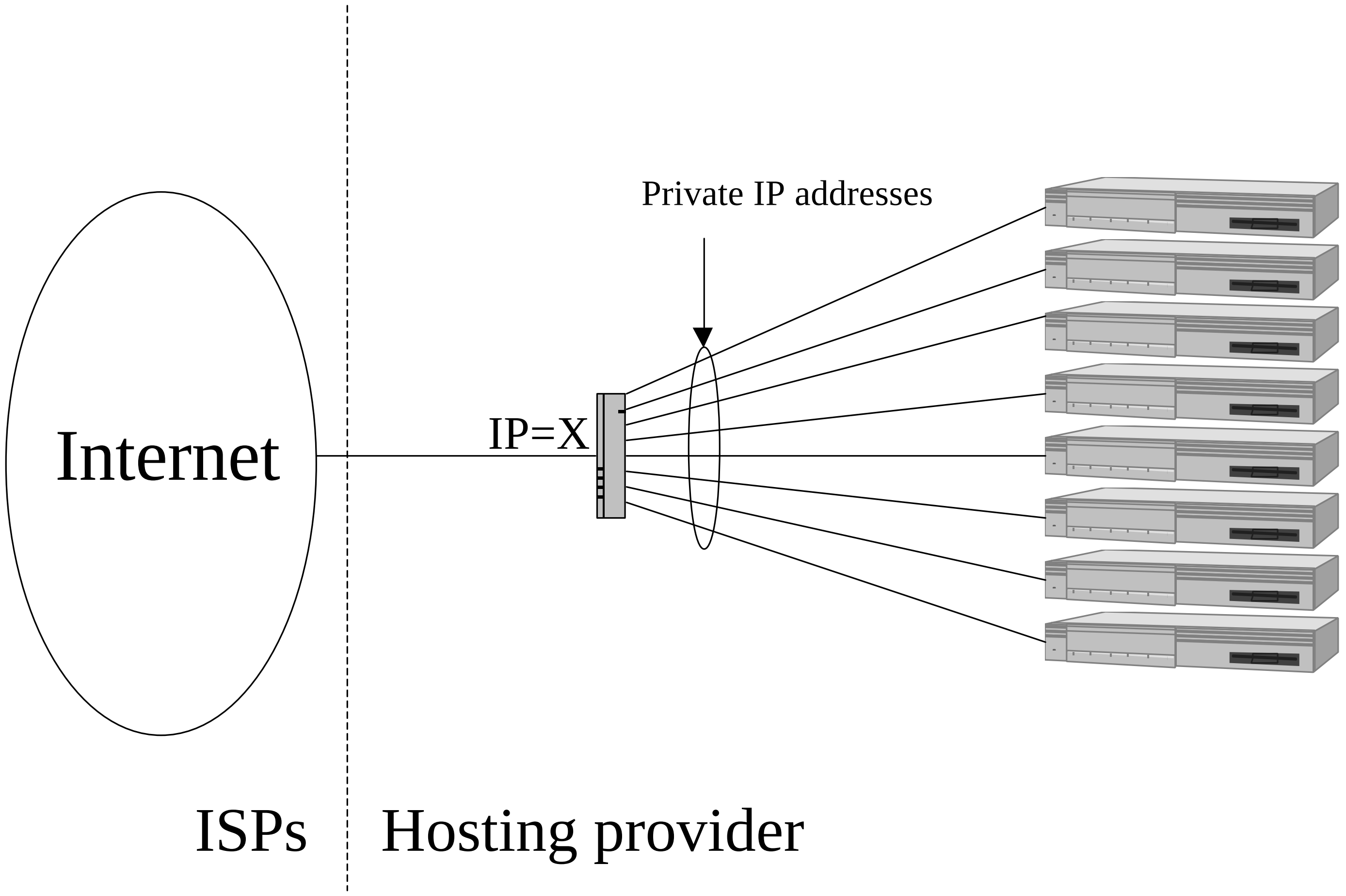
- Front-end node
  - TCP SYN
    - route to particular server based on policy
    - store decision (connID, realServer)
  - TCP ACK
    - forward based on stored decision
  - TCP FIN or a pre-defined timeout
    - remove entry
- Servers
  - IP address of outgoing interface = IP address of front-end’s incoming interface

# Olympic web server (1996)

- Advantages
  - only ACK traffic is processed
  - more reactive to load than DNS
- Disadvantages
  - non-stickiness between requests
    - SSL
    - cache performance
  - software solution (prone to DOS)
  - can’t support L5 switching
    - must proxy both ways of connection
    - need to rewrite ACKs going both ways

# Other LB variations (L2-L4)

- Hardware switches



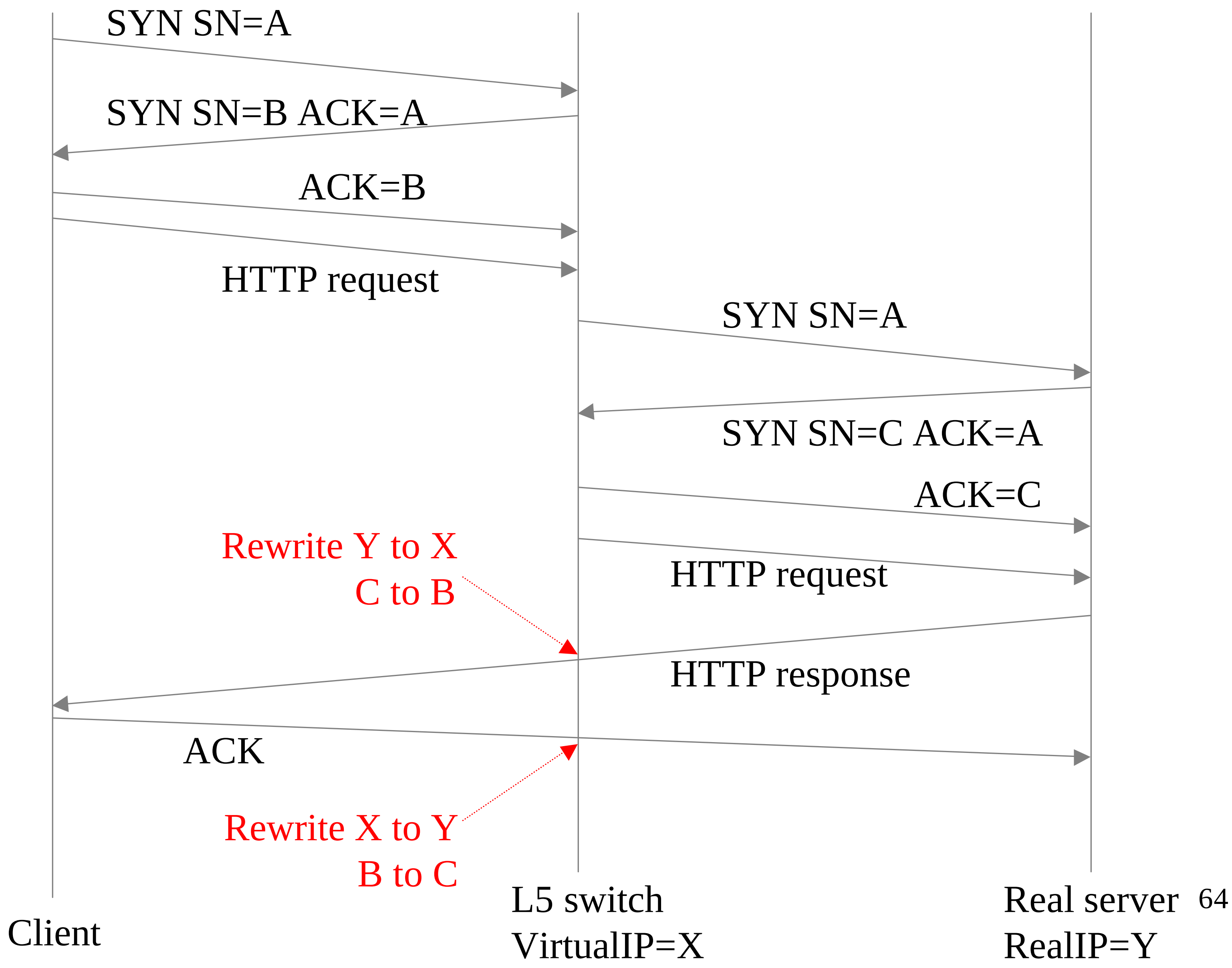
# Other LB variations (L2-L4)

- Load balancing algorithms
  - anything contained within TCP SYN packet
    - sourceIP, sourcePort, destIP, destPort, protocol
    - hash(source, dest, protocol)
  - server characteristics
    - least number of connections
    - fastest response time
    - server idle time
  - other
    - weighted round-robin, random

# Virtual servers with L5

- Spoof server connection until URL sent
- Switch based on content in request
- Server-side NAT device
- Connections proxied through switch switch terminates
- TCP handshake
  - switch rewrites sequence numbers going in both directions
  - exception
    - TCP connection migration from Rice University
    - migrate TCP state (sequence no. information) to real server
    - IP address of real server = IP address of virtual server

# L5 switches


SYN SN=A
SYN SN=B ACK=A
ACK=B
HTTP request
SYN SN=A
SYN SN=C ACK=A
ACK=C
HTTP request
Rewrite Y to X
C to B
HTTP response
ACK
Rewrite X to Y
B to C
Client
L5 switch
VirtualIP=X
Real server
RealIP=Y

# L5 switching

- Advantages
  - increases effective cache/storage sizes
  - allows for session persistence (SSL,cookies)
  - support for user-level service differentiation
    - service levels based on cookies, user profile, User-Agent, URL
- Disadvantages
  - content hot-spots
  - overhead

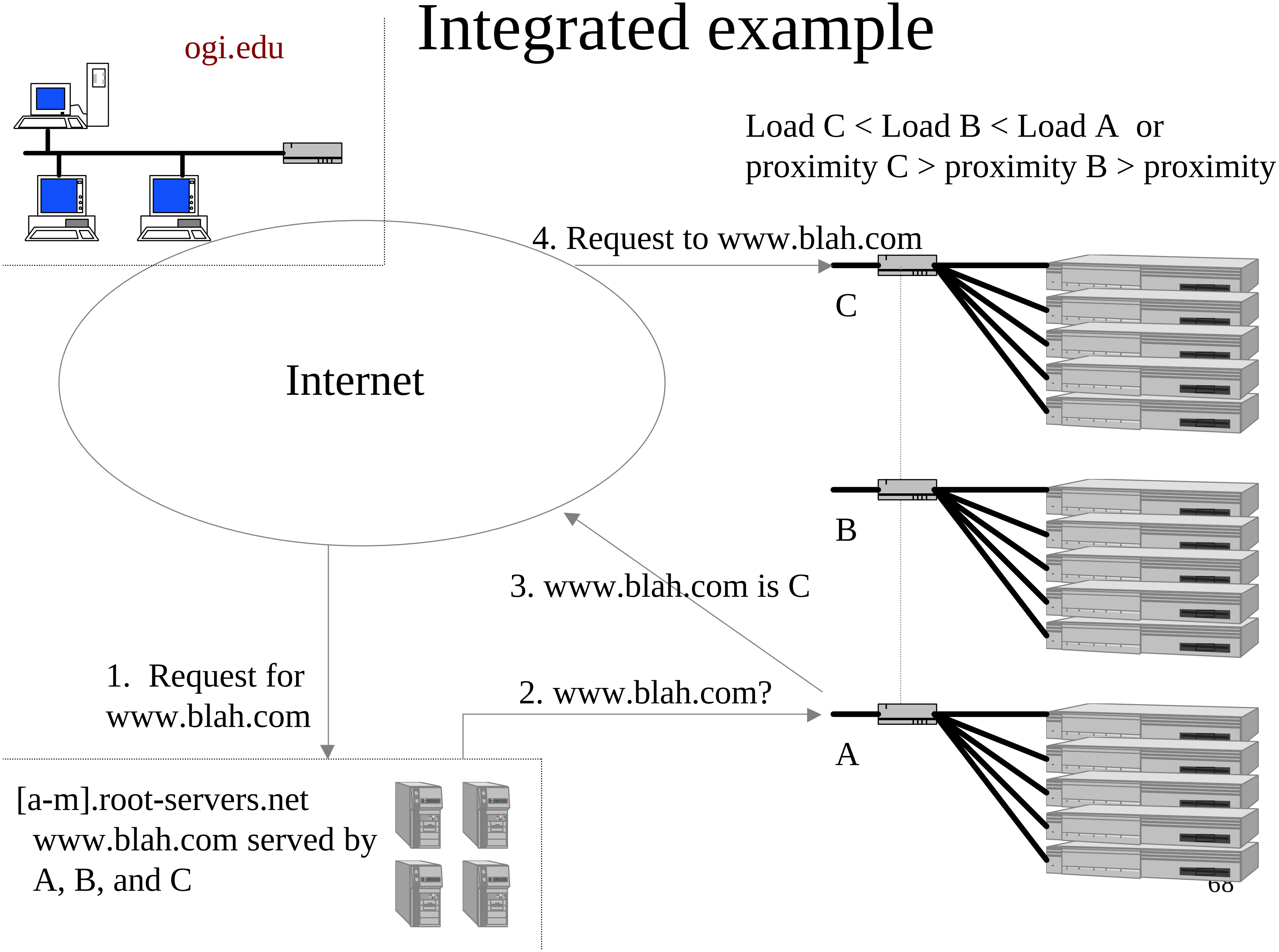
# Load balancing switches

- Cisco Local Director
- Cisco/Arrowpoint CS-100/800
- IBM Network Dispatcher
- F5 Labs BIG/ip
- Resonate Central Dispatch
- Foundry ServerIron XL
- Nortel/Alteon ACEDirector

# Integrated DNS/virtual server approaches

- LB switches coordinate and respond to DNS requests
  - based on load
  - based on geographic location
- Vendors
  - Cisco Distributed Director
  - F5 Labs BIG/ip with 3DNS
  - Nortel/Alteon ACEDirector3
  - Resonate Global Dispatch

# Integrated example



ogi.edu

Load C < Load B < Load A or
proximity C > proximity B > proximity A

4. Request to www.blah.com

Internet

C

B

A

3. www.blah.com is C

1. Request for
www.blah.com

2. www.blah.com?

[a-m].root-servers.net
www.blah.com served by
A, B, and C

# Complications to LB

- Hot-spot URLs
  - L5, URL switching bad
- Proxied sources
  - (i.e. HTTP proxies (AOL), SOCKS, NAT devices etc.)
  - L3, source IP switching bad
- Stateful requests (SSL)
  - Load-based/RR bad
- IP fragmentation
  - Breaks all algorithms unless switch defrags

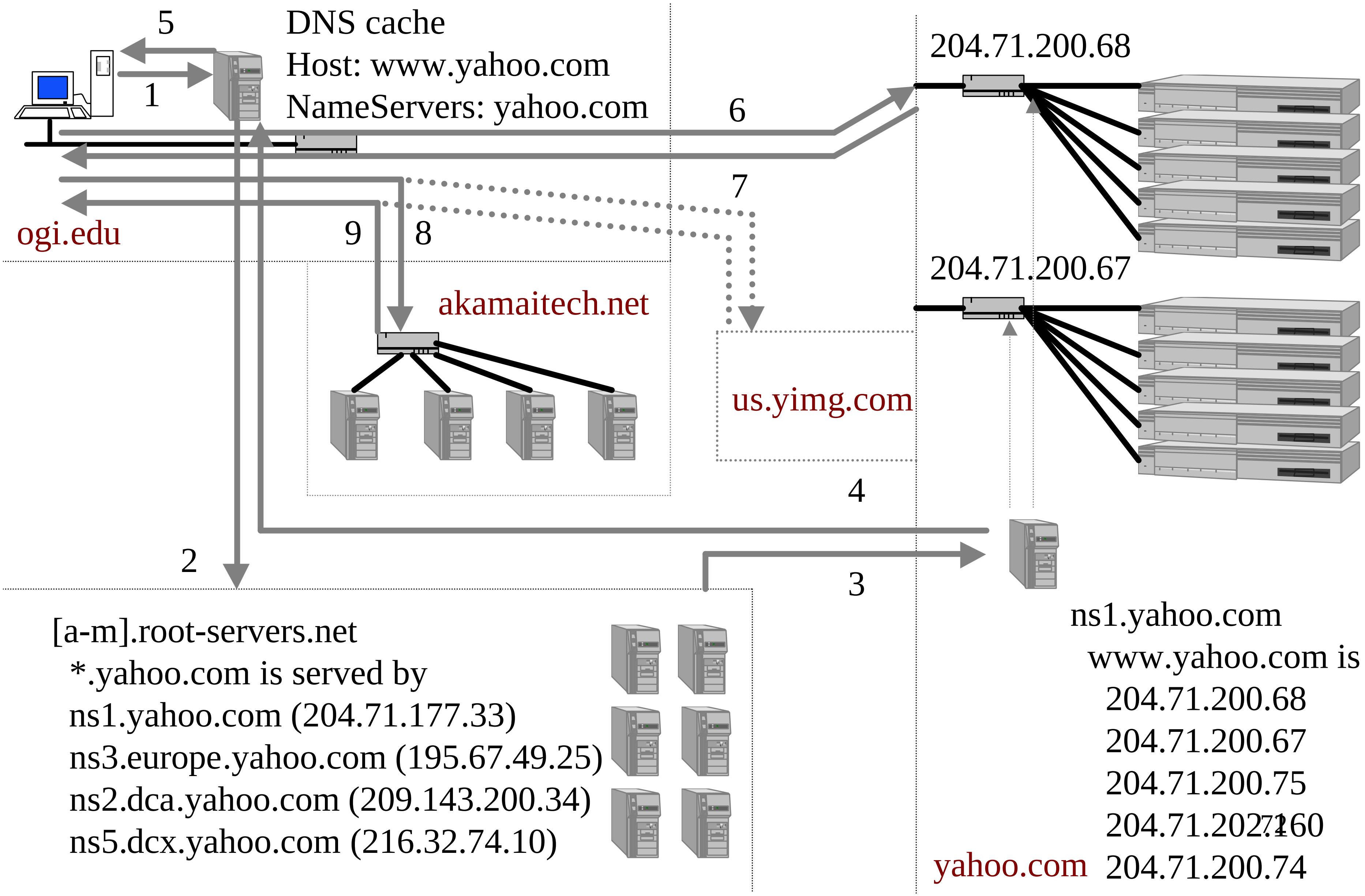
# Complications to LB

- IPsec
  - must end IPsec tunnel at switch doorstep
- Optimizing cache/disk
  - non-L5 solutions bad
- Optimizing network bandwidth
  - non-Akamai-like solutions bad

# Designing a solution

- Examine primary design goals
  - load balancing performance
  - cache hit rates
  - CPU utilization
  - network resources
- Apply solutions which fits problem

# Yahoo!



DNS cache
Host: www.yahoo.com
NameServers: yahoo.com

[a-m].root-servers.net
*.yahoo.com is served by
ns1.yahoo.com (204.71.177.33)
ns3.europe.yahoo.com (195.67.49.25)
ns2.dca.yahoo.com (209.143.200.34)
ns5.dcx.yahoo.com (216.32.74.10)

ns1.yahoo.com
www.yahoo.com is
204.71.200.68
204.71.200.67
204.71.200.75
204.71.202.160
204.71.200.74

# Proxinet Example

- Application
  - “Browser in the sky”
    - Download and rendering done on a server
    - Server does
      - HTTP protocol functions
      - HTML parsing, rendering, and layout
      - Caching
      - Transcoding of images
      - Packaging and compression
    - Client (Palm/PocketPC) does
      - Basically nothing
- Server architecture
  - CPU utilization and cache hit rates are biggest concerns
  - Network efficiency and other resources are non-issues
  - Want to support user and URL differentiation
    - L5 killed on hot-spot URLs
    - Load based algorithms killed on low cache hit rates (unless global cache is used)

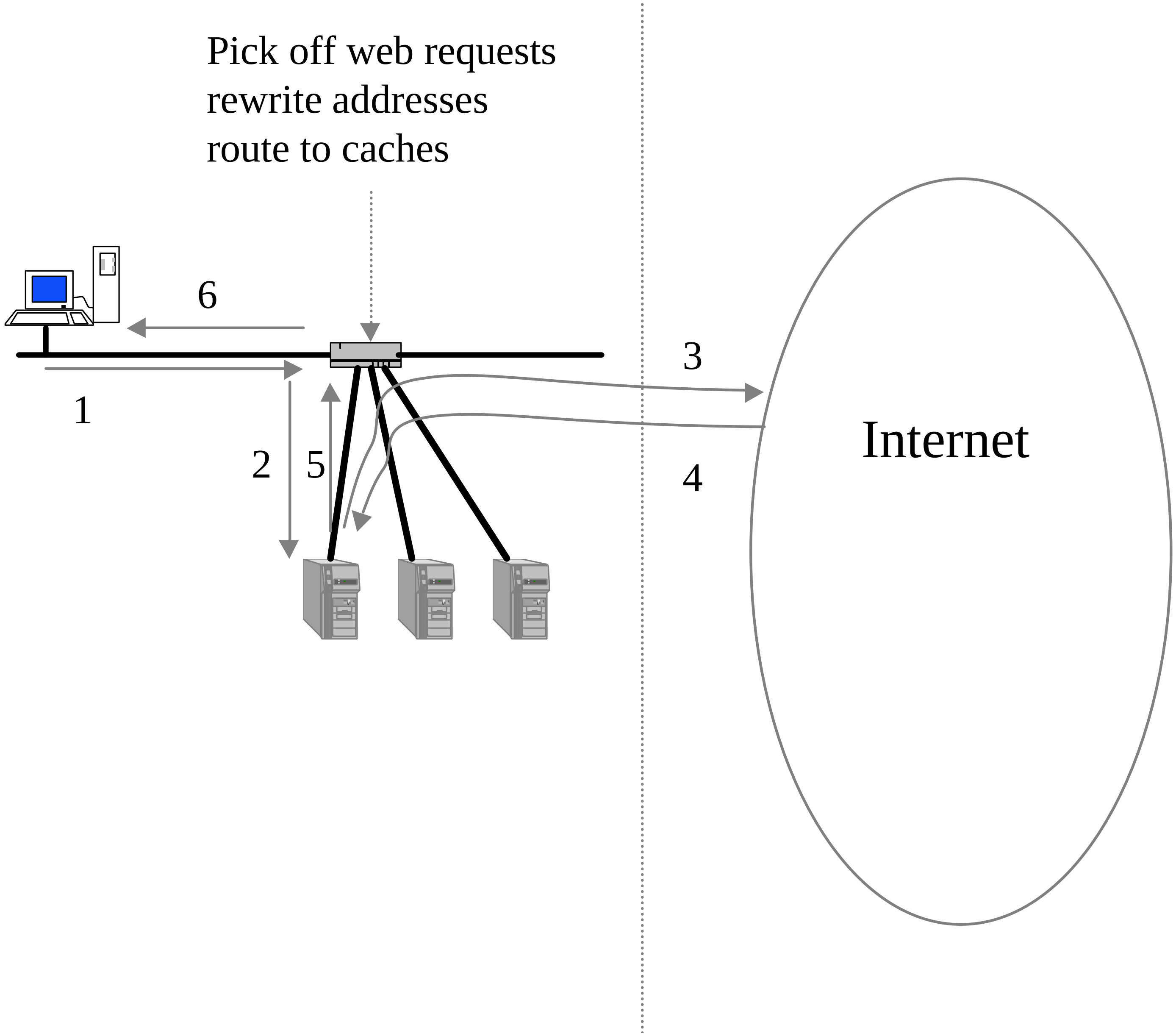
# Proxinet Example

- Solution: Use a hybrid like LARD
  - load balance with URL to a certain limit
  - load balance with least connections when load imbalanced
  - provision by URL based on Benjamins

# Transparent Web Caching

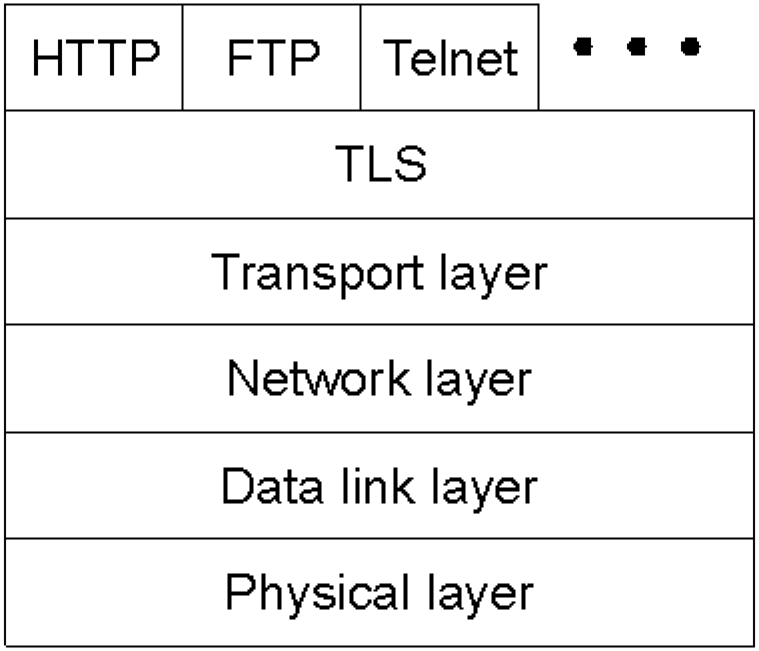
- Redirect web requests to cache transparently
- Eliminates client management costs over explicit HTTP proxies
- How?
  - Put web cache/redirector in routing path
  - Redirector
    - Pick off cacheable web requests
    - rewrite destination address and forward to cache
    - rewrite source address and return to client

# Transparent Web Caching



Pick off web requests
rewrite addresses
route to caches

# Security (1)



HTTP | FTP | Telnet • • •

TLS

Transport layer

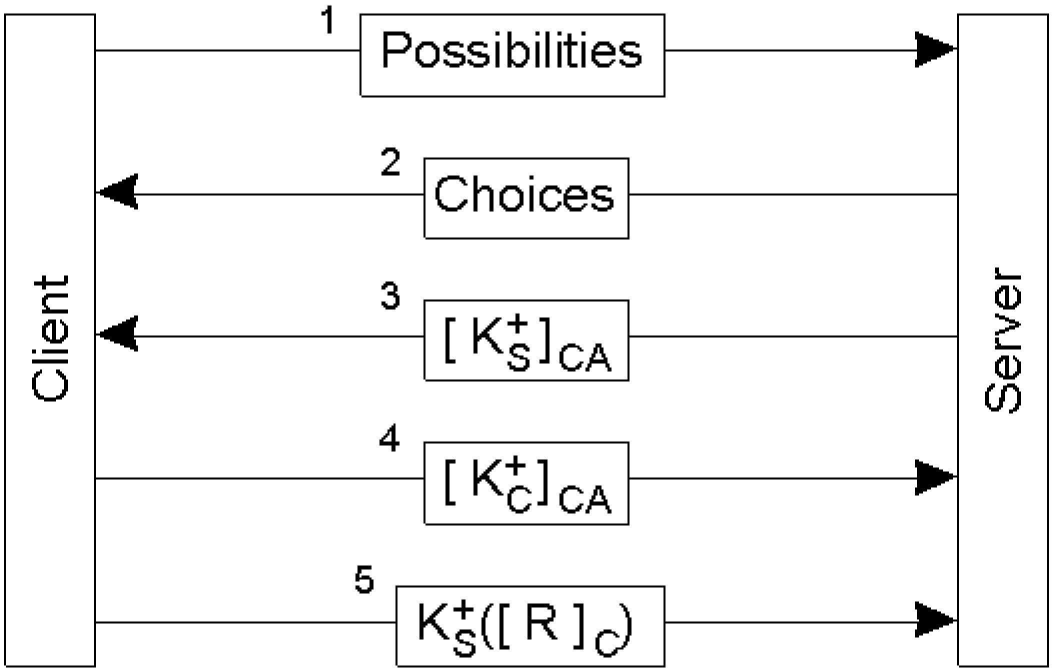
Network layer

Data link layer

Physical layer

- The position of TLS in the Internet protocol stack.

# Security (2)

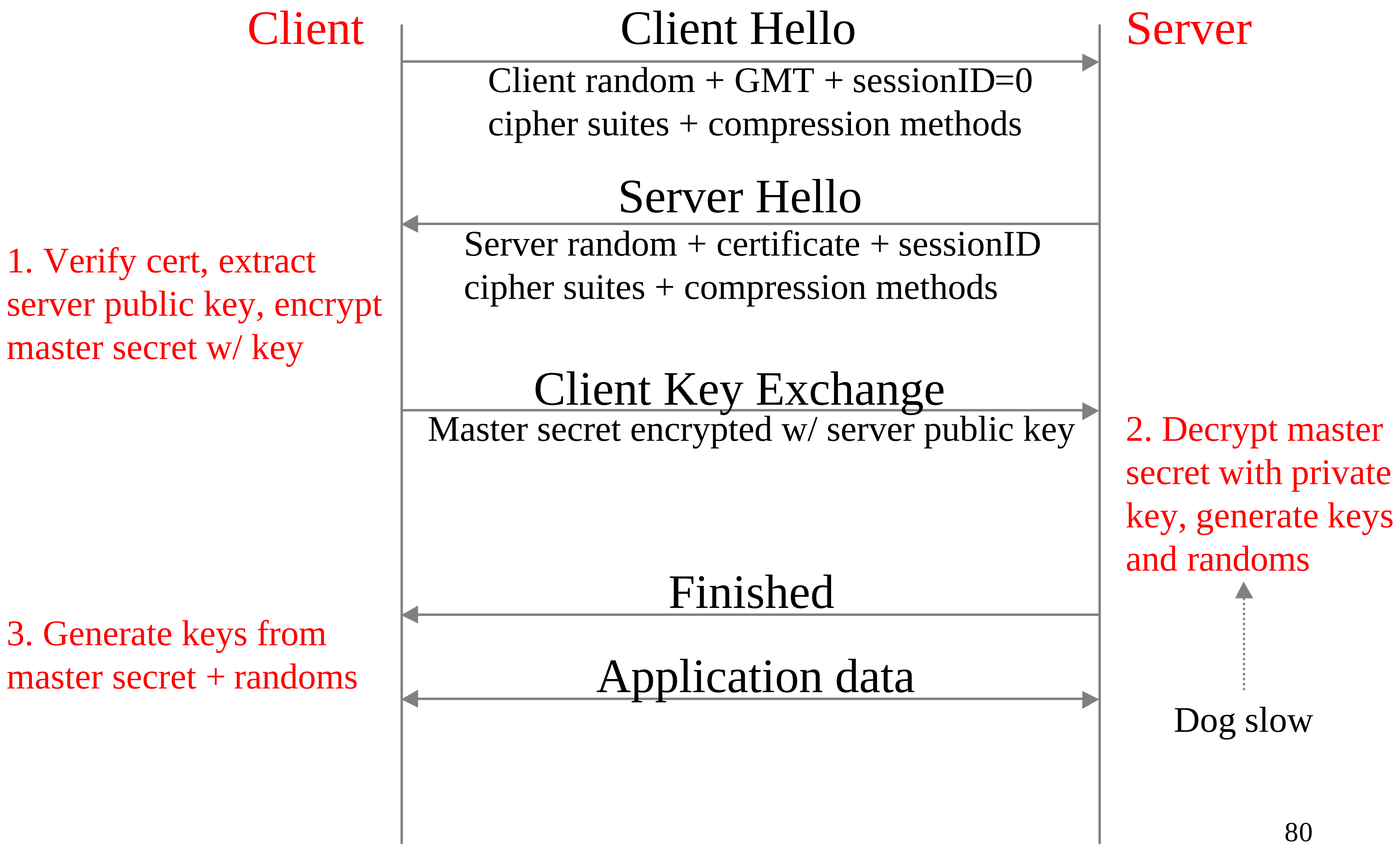


- TLS with mutual authentication.

# Scalable Secure Servers

- SSL handshake
- Server intensive processing
  - 200 MHz PowerPC
  - Client ~12ms processing
  - Server ~50ms processing
- Session reuse avoids overhead of handshake
- Subsequent requests must return to initial server

# Scalable Secure Servers



Client

Server

Client Hello

Client random + GMT + sessionID=0
cipher suites + compression methods

Server Hello

Server random + certificate + sessionID
cipher suites + compression methods

1. Verify cert, extract server public key, encrypt master secret w/ key

Client Key Exchange

Master secret encrypted w/ server public key

2. Decrypt master secret with private key, generate keys and randoms

Dog slow

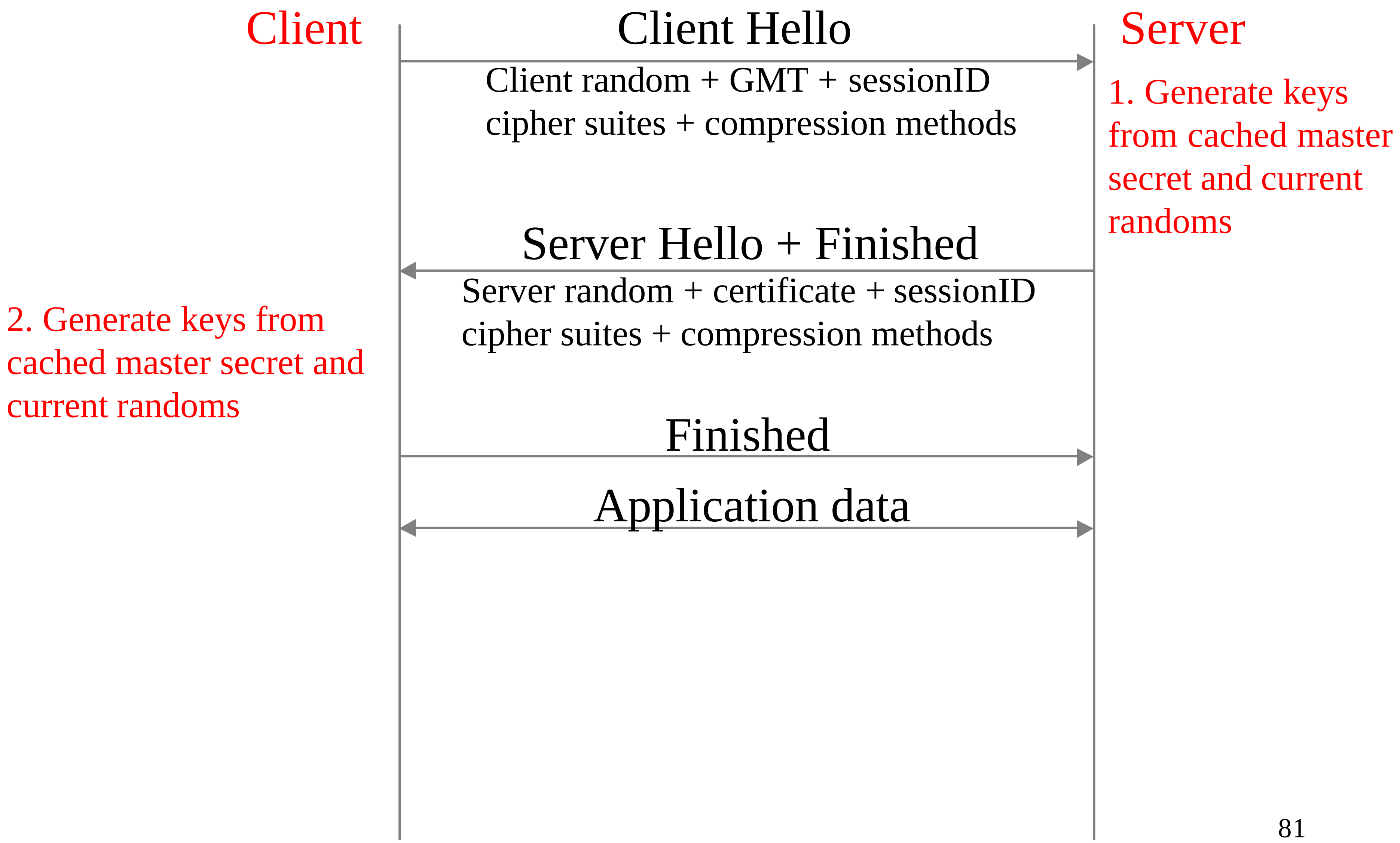
Finished

3. Generate keys from master secret + randoms

Application data

Initial SSL Handshake

# Scalable Secure Servers



Client

Server

Client Hello

Client random + GMT + sessionID
cipher suites + compression methods

1. Generate keys from cached master secret and current randoms

Server Hello + Finished

Server random + certificate + sessionID
cipher suites + compression methods

2. Generate keys from cached master secret and current randoms

Finished

Application data

SSL session reuse

# Scalable Secure Servers

- Source IP switching solution
  - solves affinity problem
  - load balancing poor
- SSL session ID switching
  - solves affinity problem
  - load balancing on initial handshake