HTTP Servers

Jacco van Ossenbruggen
CWI/VU Amsterdam

Learning goals

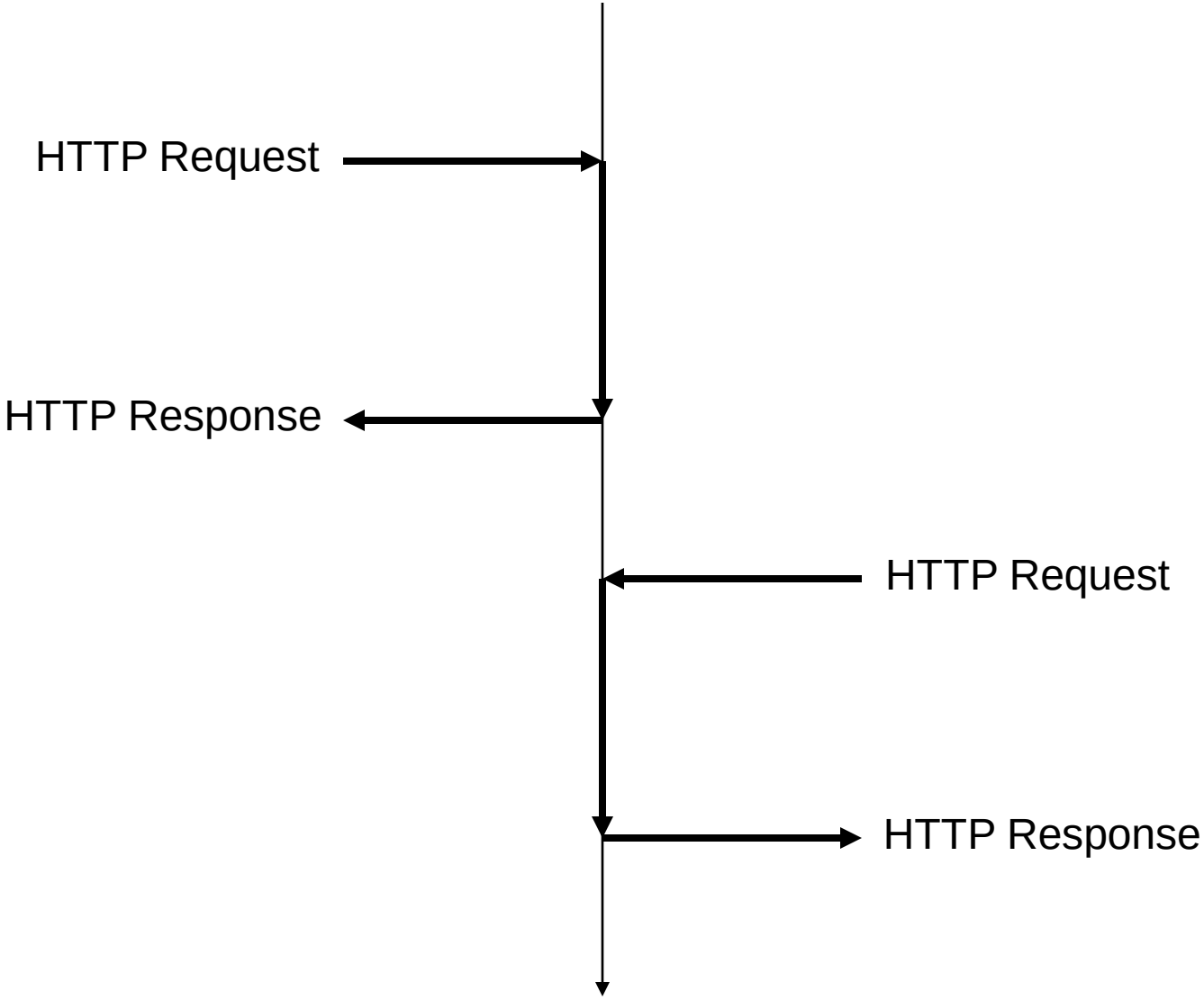
Understand:

- Basis HTTP server functionality
- Serving static content
  - from HTML and other files
- Serving dynamic content
  - from software within a HTTP server
  - from external software
- Security & privacy issues

HTTP: The Web's network protocol

- Early 90s: only a few HTTP servers, but many FTP servers helped bootstrapping the Web
  - Example: ftp://ftp.gnu.org/gnu/aspell/dict/en/
- HTTP servers based on the freely available *httpd* web server from NSCA
- NCSA stopped *httpd* support when the associated team left to start Netscape
- Webmasters started to send around software patches to further improve *httpd*
- Result was referred to as "a patchy server"
- Now the open source **Apache** server is one of the mostly used Web servers

HTTP server main loop



HTTP server main loop

```
while(forever)
  listen to TCP port 80 and wait
  read HTTP request from client
  send HTTP response to client
```

Seems not that complicated …

But: regular Apache HTTP server installation installs > 24Mb of software … ?!

What makes real servers so complex?

Static content

from files: HTML,
CSS, JavaScript, images, …

Example HTTP request

- **GET / HTTP/1.0**
-

Example HTTP request

- **`GET / HTTP/1.1`**
- **`Host: www.few.vu.nl`**

Why does the client need to tell the server the server's own hostname?

- because the server doesn't know its own name!
- **`www.cs.vu.nl`** is hosted on the same machine by the same server software
- server may need to send different responses for different host names
- "Virtual host" configuration allows web masters to tune server to do exactly this

Example HTTP request

```
GET / HTTP/1.1
Host: www.few.vu.nl
```

- Server needs to determine what resource is associated with '/'
- Also configurable, defaults to the file **`index.html`** in the server's "document root" directory, e.g. **`/var/www/www.few.vu.nl/html/index.html`**
- Security issues
  - **`GET ~yourname/../../../passwd HTTP/1.1`**
  - **`GET ~yourname/../~yourlogin/Mail HTTP/1.1`**
- Webmaster needs to configure which directories in the local file system may be served by the web server
  - Webmaster: "Oops, that dir should not have been on the Web"
  - User: "Oops, I didn't know this dir was on the Web too"

Example HTTP request

```
GET / HTTP/1.1
Host: www.few.vu.nl
```

- Server needs to send content of file **`index.html`** to the client
- Along with
  - length of the content
  - the current time/date
  - modification date
  - expiration date
  - MIME type of the content (e.g. text/html)
  - character encoding (e.g. UTF-8)
  - etc
- Most of these HTTP header values need to be looked up in a configurable way
- Results need to be **logged** in the server log for later analysis

Example: apache HTTP logs

access_log.2:**soling.few.vu.nl** - - [**11/Jan/2008:16:47:19 +0100**] "GET **/cgi-bin/wt-test?naam=&textarea=+** HTTP/1.0" 200 1341 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"

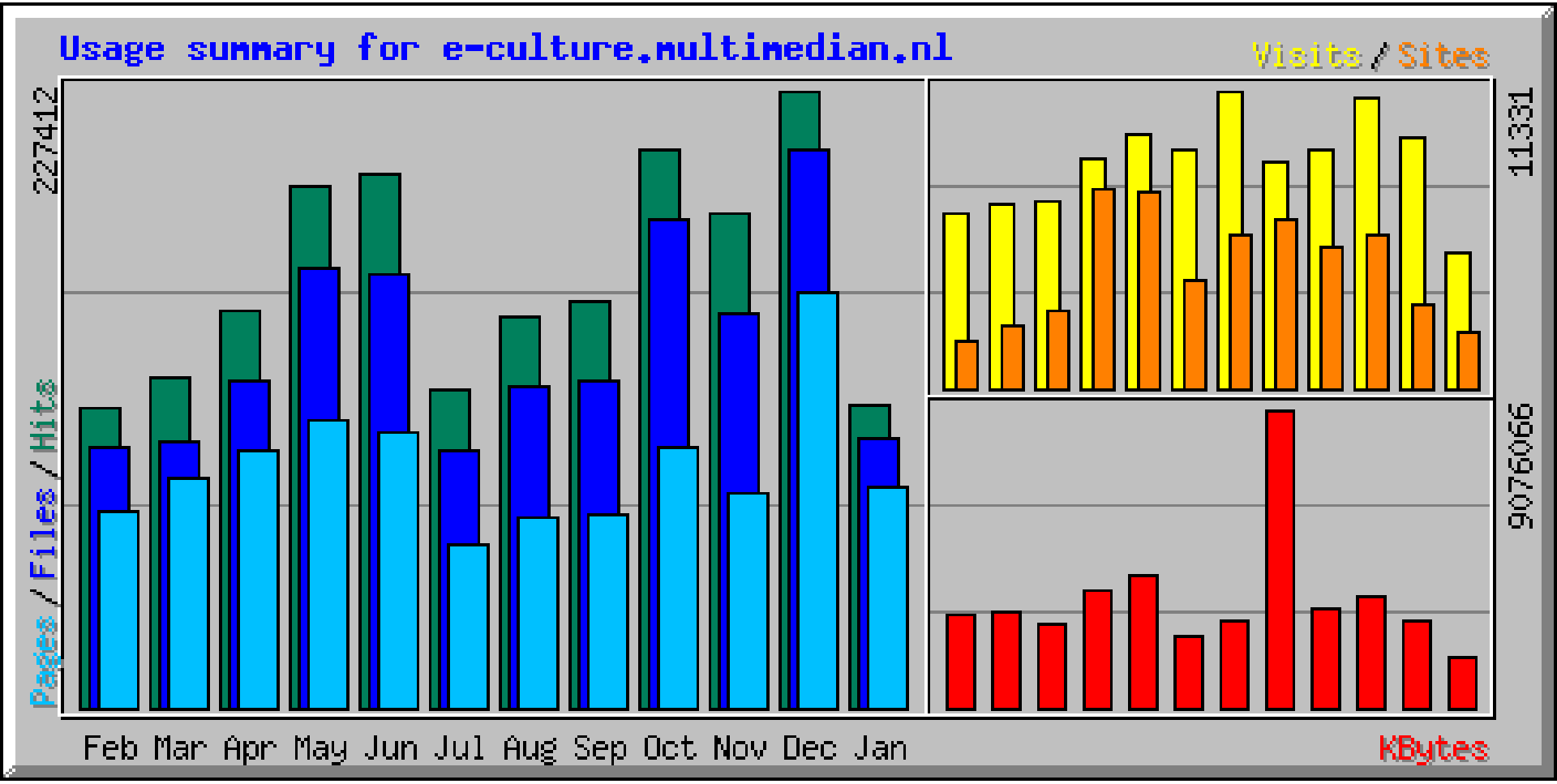
access_log.2:soling.few.vu.nl - - [11/Jan/2008:16:47:48 +0100] "GET /cgi-bin/wt-test?naam=&textarea=+ HTTP/1.0" 200 1341 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"

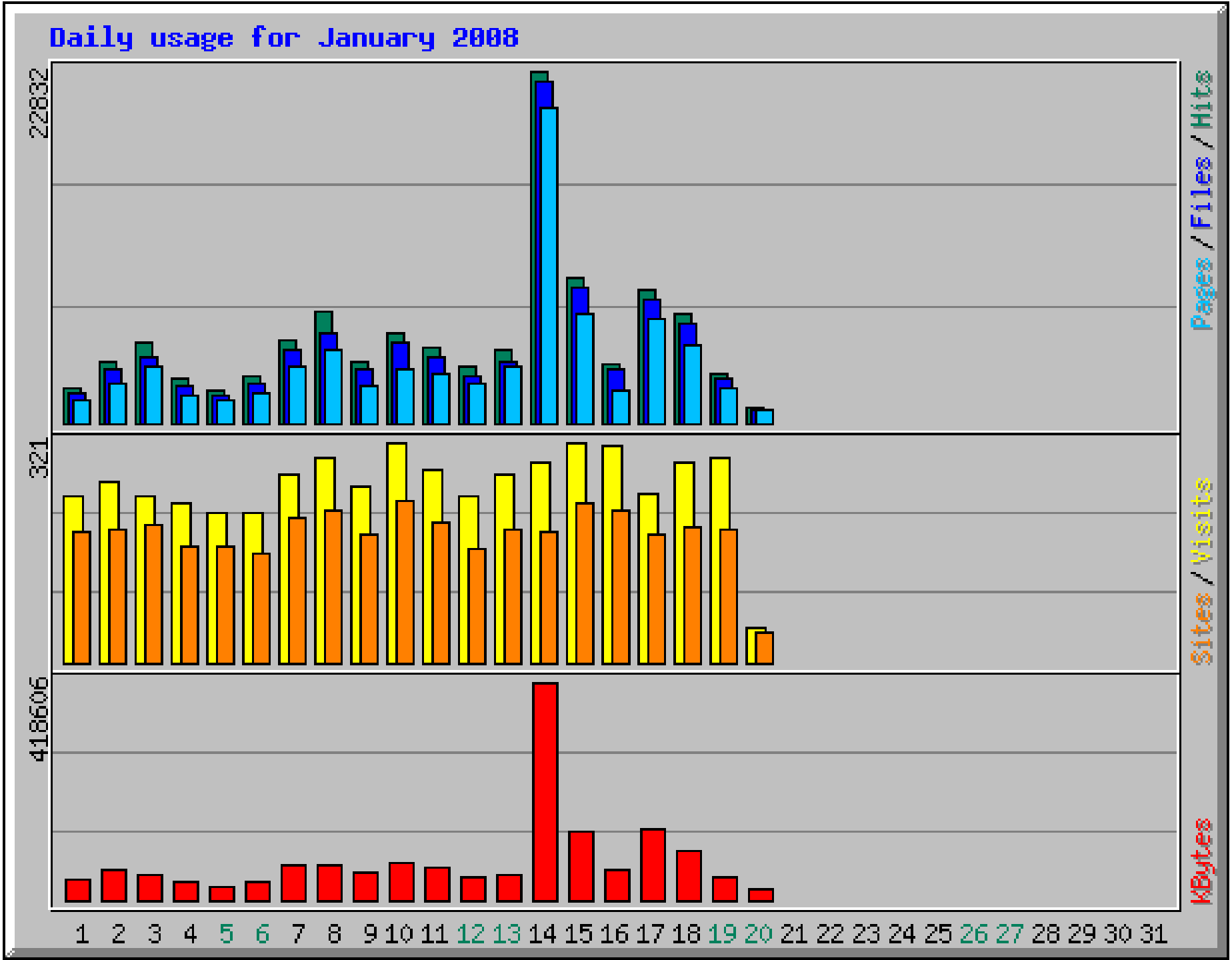
access_log.2:soling.few.vu.nl - - [11/Jan/2008:16:48:48 +0100] "GET /cgi-bin/wt-test?naam=&textarea=+ HTTP/1.0" 200 1341 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"

access_log.2:soling.few.vu.nl - - [11/Jan/2008:16:55:59 +0100] "GET /cgi-bin/wt-test?naam=&radio=inhoudelijk&textarea=+vxfvsdfsdf%0D%0A HTTP/1.0" 200 1409 "-“ "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firfox/2.0.0.6"

access_log.2:soling.few.vu.nl - - [11/Jan/2008:16:56:08 +0100] "GET /cgi-bin/wt-test?naam=Cjijij&radio=inhoudelijk&checkbox1=checkbox1&textarea=+vxfvsdfsdf%0D0A%0D%0Afsdfsdf HTTP/1.0" 200 1487 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1 en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"

access_log.2:soling.few.vu.nl - - [11/Jan/2008:16:58:25 +0100] "GET /cgi-bin/wt-test?naam=&radio=structuur1&textarea=+ HTTP/1.0" 200 1375 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"


Usage summary for e-culture.multimedian.nl
Visits / Sites
227412
11331
Pages / Files / Hits
9076066
Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan
KBytes


Daily usage for January 2008
22832
321
418606
Pages / Files / Hits
Sites / Visits
KBytes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Top N of ...

Top 10 of 2094 Total Sites

	#Hits		Files		Kbytes		Visits		Hostname
1	**28066**	25.26%	**27754**	27.89%	**529851**	34.02%	**50**	0.97%	*.search.live.com
2	**14434**	12.99%	**13899**	13.96%	**206962**	13.29%	**7**	0.14%	***.googlebot.com**
3	**8963**	8.07%	**5779**	5.81%	**47864**	3.07%	**17**	0.33%	*.speedy.telkom.net.id
4	**6142**	5.53%	**5871**	5.90%	**59502**	3.82%	**82**	1.59%	***.cwi.nl**
5	**1265**	1.14%	**1203**	1.21%	**6455**	0.41%	**3**	0.06%	ipXX.speed.planet.nl
6	**1237**	1.11%	**1228**	1.23%	**10163**	0.65%	**18**	0.35%	soling.few.vu.nl
7	**1169**	1.05%	**1026**	1.03%	**6181**	0.40%	**1**	0.02%	XX.demon.nl
8	**1050**	0.94%	**972**	0.98%	**16429**	1.05%	**5**	0.10%	XXadsl.sinica.edu.tw
9	**956**	0.86%	**904**	0.91%	**5634**	0.36%	**5**	0.10%	XX.adslsurfen.hetnet.nl
10	**908**	0.82%	**889**	0.89%	**13028**	0.84%	**21**	0.41%	XX.wise-guys.nl

Top 7 Search Strings

1	**60**	**37.97%**	**the scream**
2	**8**	**5.06%**	**vu**
3	**6**	**3.80%**	**scream**
4	**4**	**2.53%**	**eculture**
5	**4**	**2.53%**	**the scream painting**
6	**3**	**1.90%**	**the scream paintings**
7	**2**	**1.27%**	***.gif**

Example HTTP request

```
GET / HTTP/1.1
Host: www.few.vu.nl
```

- Server needs to send content of file **`index.html`** to the client
- Along with
  - length of the content
  - the current time/date
  - modification date
  - expiration date
  - MIME type of the content (e.g. text/html)
  - character encoding (e.g. UTF-8)
  - etc
- Most of these HTTP header values need to be looked up in a configurable way
- Results need to be **logged** in the server log for later analysis
  - Assume everything you do will be logged and will be traceable back to you

Example HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 21 Jan 2008 10:18:49 GMT
Server: Apache/2.0.58 (Unix) mod_ssl/2.0.58
  OpenSSL/0.9.7d DAV/2 PHP/5.2.4 mod_python/3.3.1
  Python/2.4.3
X-Powered-By: PHP/5.2.4
Expires: Mon, 21 Jan 2008 16:18:49 GMT
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
<head>
```

Example HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 21 Jan 2008 10:18:49 GMT
Server: Apache/2.0.58 (Unix) mod_ssl/2.0.58
  OpenSSL/0.9.7d DAV/2 PHP/5.2.4 mod_python/3.3.1
  Python/2.4.3
X-Powered-By: PHP/5.2.4
Expires: Mon, 21 Jan 2008 16:18:49 GMT
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
<head>
```

Example HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 21 Jan 2008 10:18:49 GMT
Server: Apache/2.0.58 (Unix) mod_ssl/2.0.58
  OpenSSL/0.9.7d DAV/2 PHP/5.2.4 mod_python/3.3.1
  Python/2.4.3
X-Powered-By: PHP/5.2.4
Expires: Mon, 21 Jan 2008 16:18:49 GMT
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
<head>
```

Static vs dynamic content

- Not all requests are for **static content** stored in a file
  - some data needs to be requested by the server from other applications
    (e.g. from an organisation's database)
  - some data needs to be computed "on the fly" in response to the request
    (e.g. results of a query on a search engine)
- Need for **dynamic content** by **programmable** server behaviour
- Note: from the browser's perspective, static and dynamic content look syntactically exactly the same ("it's just a URI")

REST

Roy Fielding

- co-author of the HTTP specification
- co-founder of Apache
- described the key principles of WWW network architecture in his PhD thesis (UCI, 2000)
- He named these principles **REST** (**RE**presentational **S**tate **T**ransfer)
- Implementations are called **RESTful**
- REST strongly influenced the early network architecture of the Web…
- … and still does:
  - 15 Jan 2008:
    W3C published the SPARQL Recommendation, a web query language based on a RESTful design

REST: key principles

- **All** sources of information (files and applications) are **resources** that are uniquely addressable using a **URI**
- Clients and servers only need to know
  - the URI of the resource (e.g. **`http://www.few.vu.nl/`**)
  - the allowed actions (e.g. **`HTTP GET`**)
  - the allowed representations (e.g. **`text/html`**)
- Client does not need to know how the server generates the representation
- Server does not need to know how the client presents it
- Both client and server do not need to be aware of intermediate proxies or caches
- There is no communication state
  - HTTP response does not depend on previous request
  - Methods are **idempotent**: requesting the same resource multiply times will yield the same content
- Simplifies global design and improves performance …
- … but sometimes makes server programming more difficult

dynamic content

computed by other software
computed by the server

CGI: common gateway interface

- Commonly agreed upon way to run batch programs in response to a HTTP request
- HTTP server executes program
  - server recognizes a CGI request and determines which program from the URL
  - supplying details about the request to the program via (OS environment) variables
  - returning program's output verbatim to the client (output needs to supply content and all required HTTP headers)

CGI Example: form URL you used in assignment 1

```
<form action="http://eculture.cs.vu.nl/cgi-bin/wt1-test"
      method="get">
```

```
#!/usr/bin/perl
##
##  cgi-bin/wt1-test -- program which just prints its environment
##

print "Content-type: text/plain\n\n";
foreach $var (sort(keys(%ENV))) {
    $val = $ENV{$var};
    $val =~ s|\n|\\n|g;
    $val =~ s|"|\\"|g;
    print "${var}=\"${val}\"\n";
}
```

CGI response

```
HTTP/1.1 200 OK
Date: Fri, 18 Jan 2008 14:09:18 GMT
Server: Apache/2.2.9
Connection: close
Content-Type: text/plain

DOCUMENT_ROOT="/export/data1/httpd/htdocs"
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT_LANGUAGE="en"
HTTP_HOST=”eculture.cs.vu.nl"
QUERY_STRING="name=value"
REMOTE_ADDR=“80.127.61.144"
REMOTE_HOST=“plan.xs4all.nl“
…
```

CGI: pros & cons

- ✓ Very flexible
  - can use programs written in any interpreted or compiled programming language
  - easy way to reuse existing software in a Web context
- – Creates a new process to re-execute program for every request
  - – very expensive: too slow for popular sites
  - – hard to maintain state between requests

    (we will look deeper into the concept of state later)
- – Mixes program logic and HTML generation
  - – hard to maintain by programmers and designers
- – Not convenient to get data from databases

CGI alternatives

- server-side scripting:
  - server has a module that keeps the language interpreter running over multiple requests
  - running little scripts at the server (“servlets”) is then relatively cheap
- Use general purpose scripting languages
  - Apache comes standard with modules for many languages: mod_python, mod_perl, …

Example HTTP response

```
HTTP/1.1 200 OK
Date: Fri, 18 Jan 2008 11:18:49 GMT
Server: Apache/2.0.58 (Unix) mod_ssl/2.0.58
  OpenSSL/0.9.7d DAV/2 PHP/5.2.4 mod_python/3.3.1
  Python/2.4.3
X-Powered-By: PHP/5.2.4
Expires: Fri, 21 Jan 2008 17:18:49 GMT
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
<head>
```

CGI alternatives: scripting

- Server-side scripting:
  - server has a module that keeps the language interpreter running over multiple requests
  - running little scripts at the server is then relatively cheap
- Use general purpose scripting languages:
  - mod_python, mod_perl, …
  - need rules to determine which URLs are deferred to script module (e.g. **`http://www.example.org/file.py`**)
- Compiled Java bytecode programs
  - server modules running a Java Virtual Machine are known as a web or **servlet** container (e.g. tomcat)
  - servlets typically use standard Java extensions to simplify programming (javax.servlet.*)
- All these solutions result in files that look like programs
  - HTML markup deeply hidden in "print" statements
  - hard to maintain by non-programmers

Example: code with hidden HTML

```
print “<html>”
…
print “<body>”
print “<ul>”
for (i=1; i<N; i++) {
     data = get_item(i);
     print “<li>” + data +</li>
}
print “</ul>”
…
```

Dedicated frameworks

- Use dedicated scripting frameworks
  - *PHP: Hypertext Preprocessor*
    - Used to implement WordPress, MediaWiki
    - mixes html, program code & database queries
  - *JSP: Java Server Pages*
    - mixes html & java
- These approaches typically result in files that look like HTML pages, with embedded code and custom tags processed by the server
  - complex func. still requires programming
  - but results are easier to reuse
  - easier to maintain, also by non-programmers

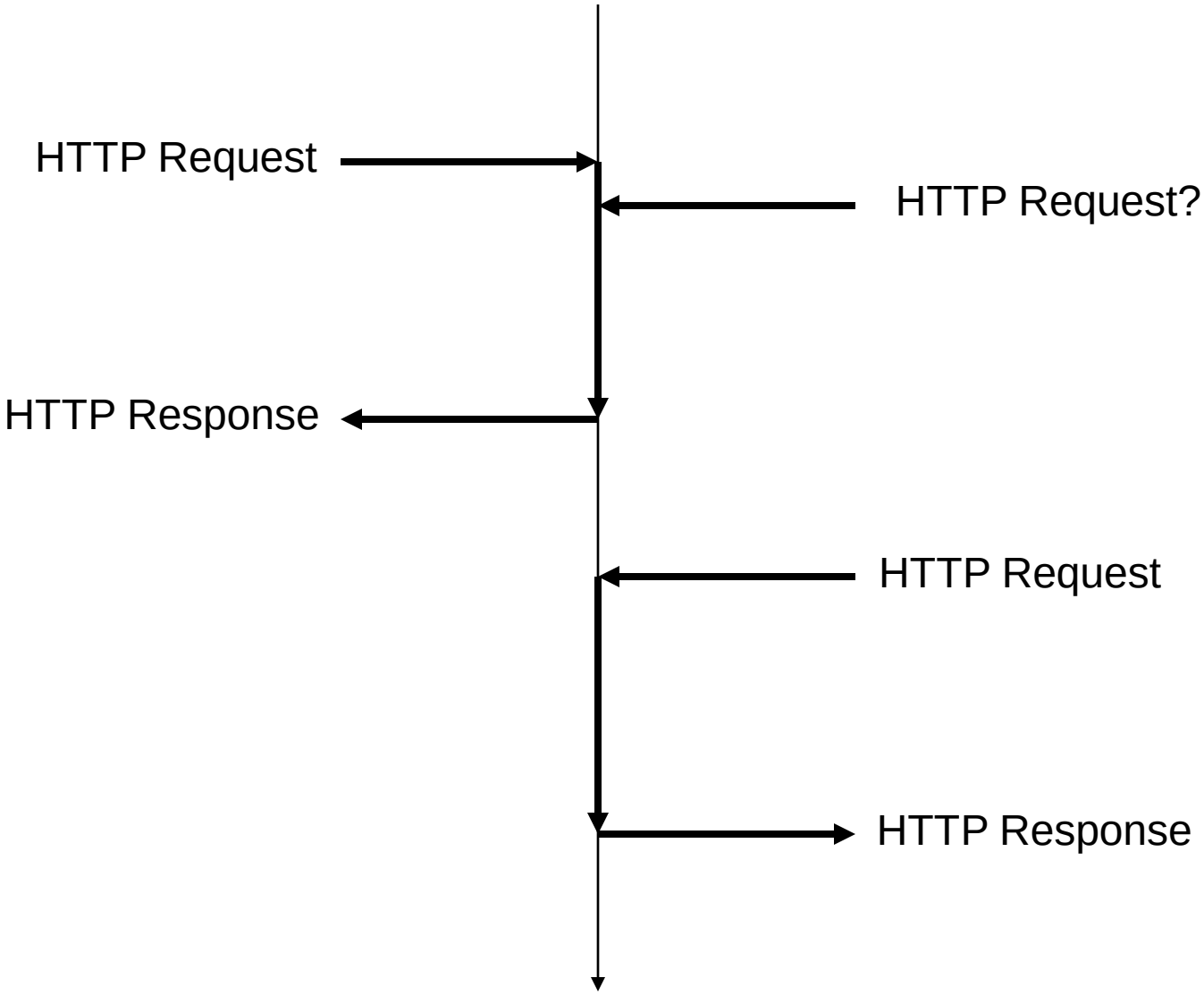
Example: HTML with hidden code

```
<html>
…
<body>
<ul>
<? generate_items(N) ?>
</ul>
</body>
</html>
```

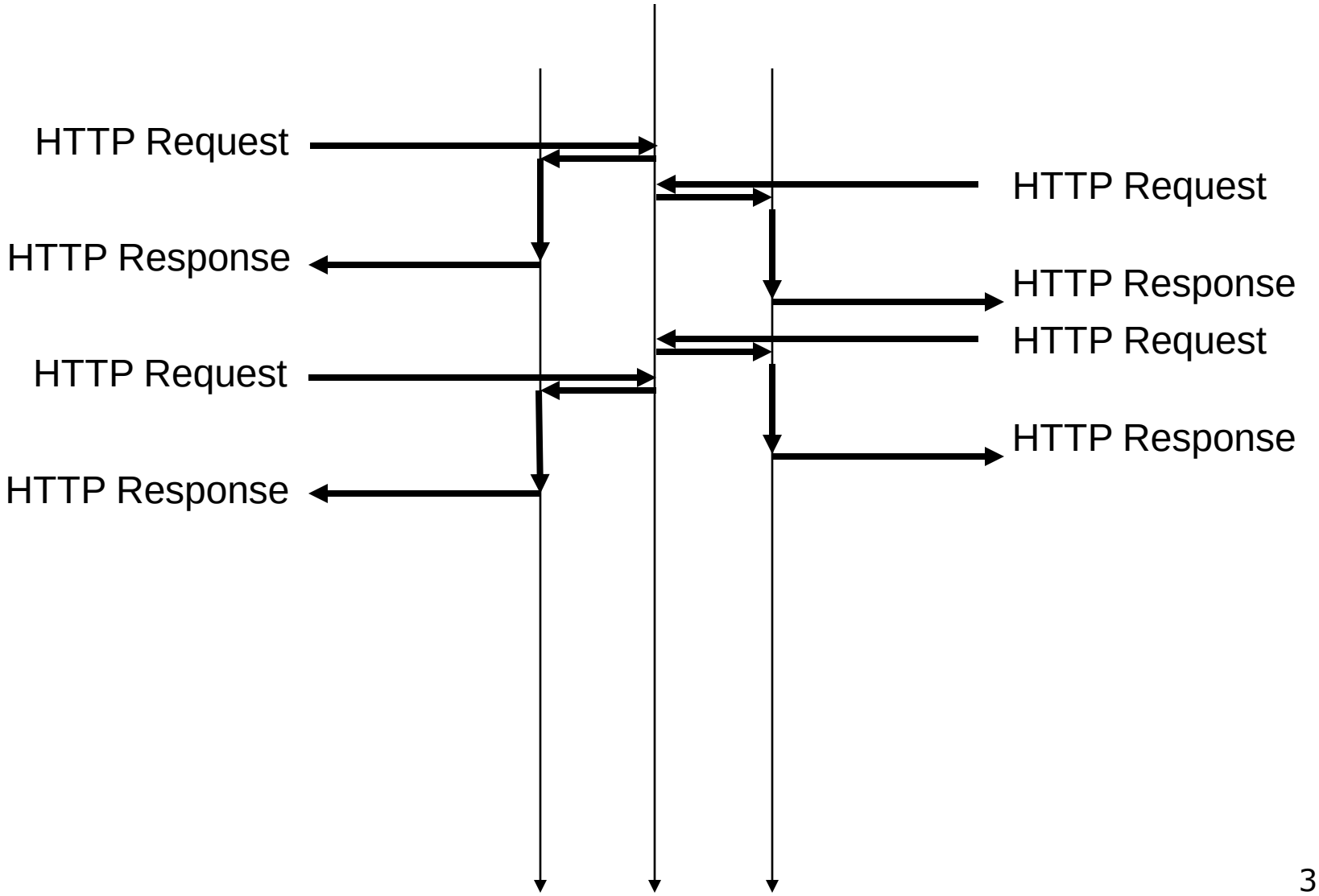
Typical problems in server programming

- Concurrency
- Session management & cookies
- Authentication & security
- Interfacing with other software (generating HTML from database content)

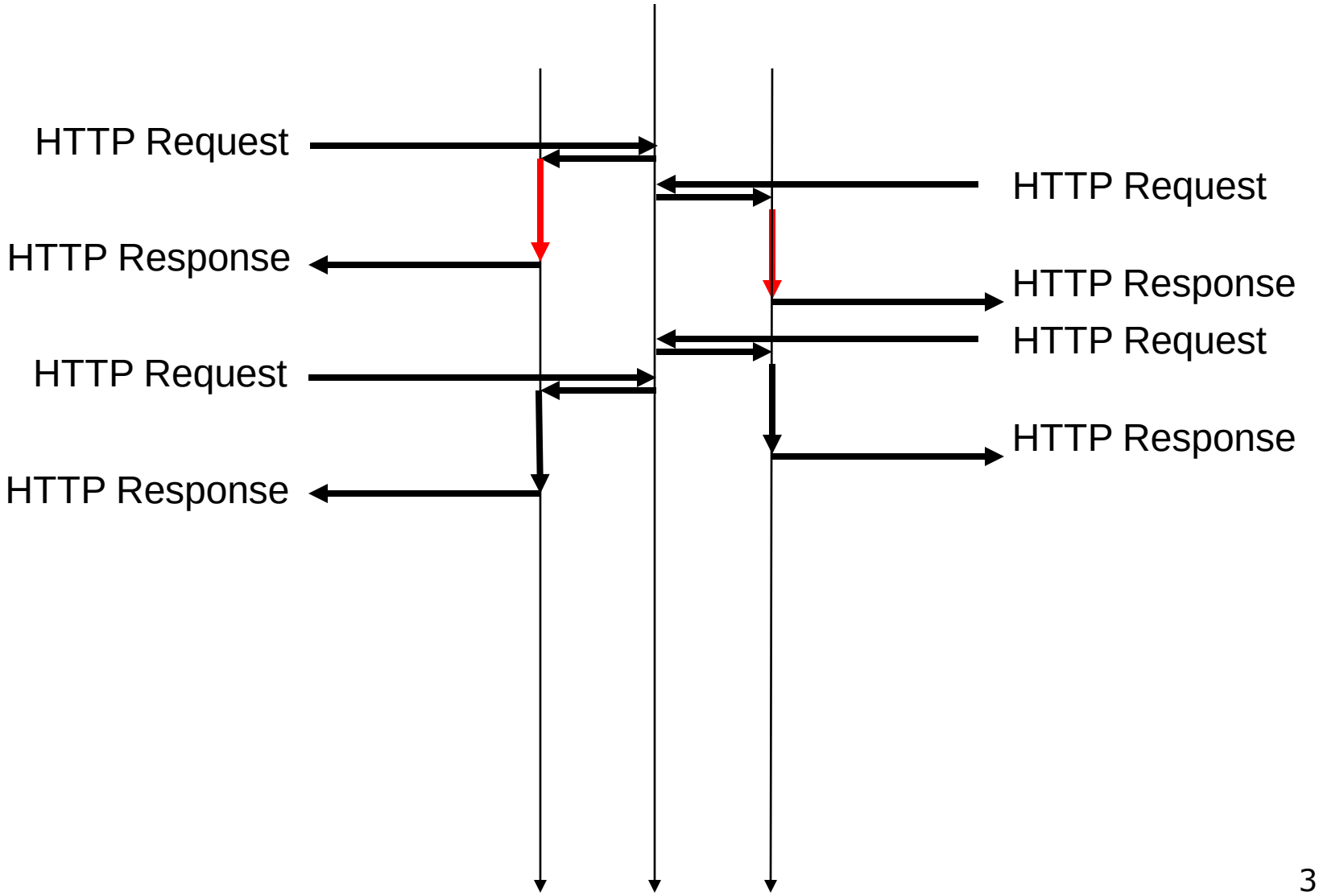
HTTP server main loop



HTTP server concurrency



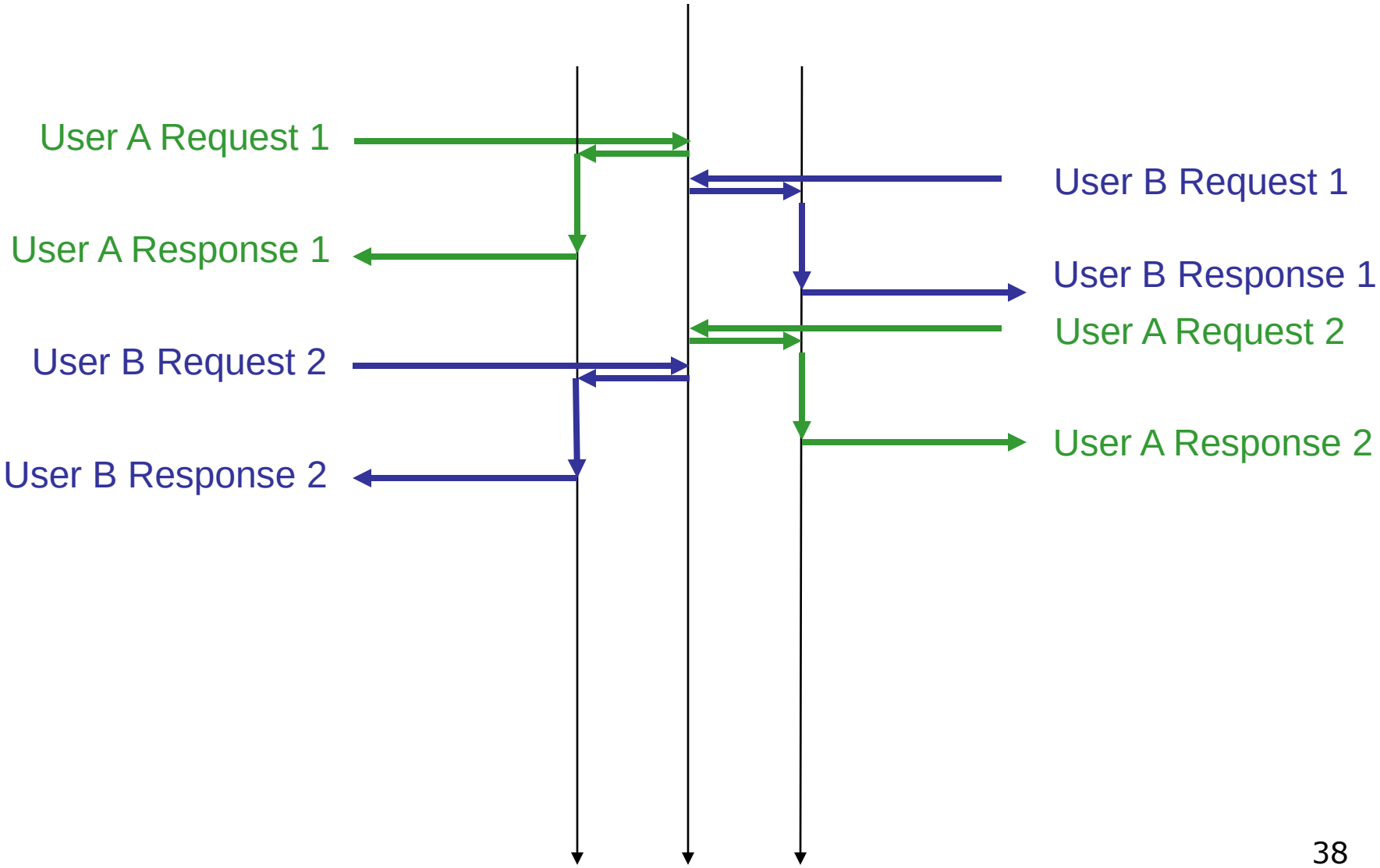
HTTP server concurrency



HTTP server concurrency

- Server-side software needs to be aware that other processes/threads processing other request **may** run at the same time ("multi-threading", "MT-safe")
  - makes accessing global resources (variables, databases, files) more complicated and error prone

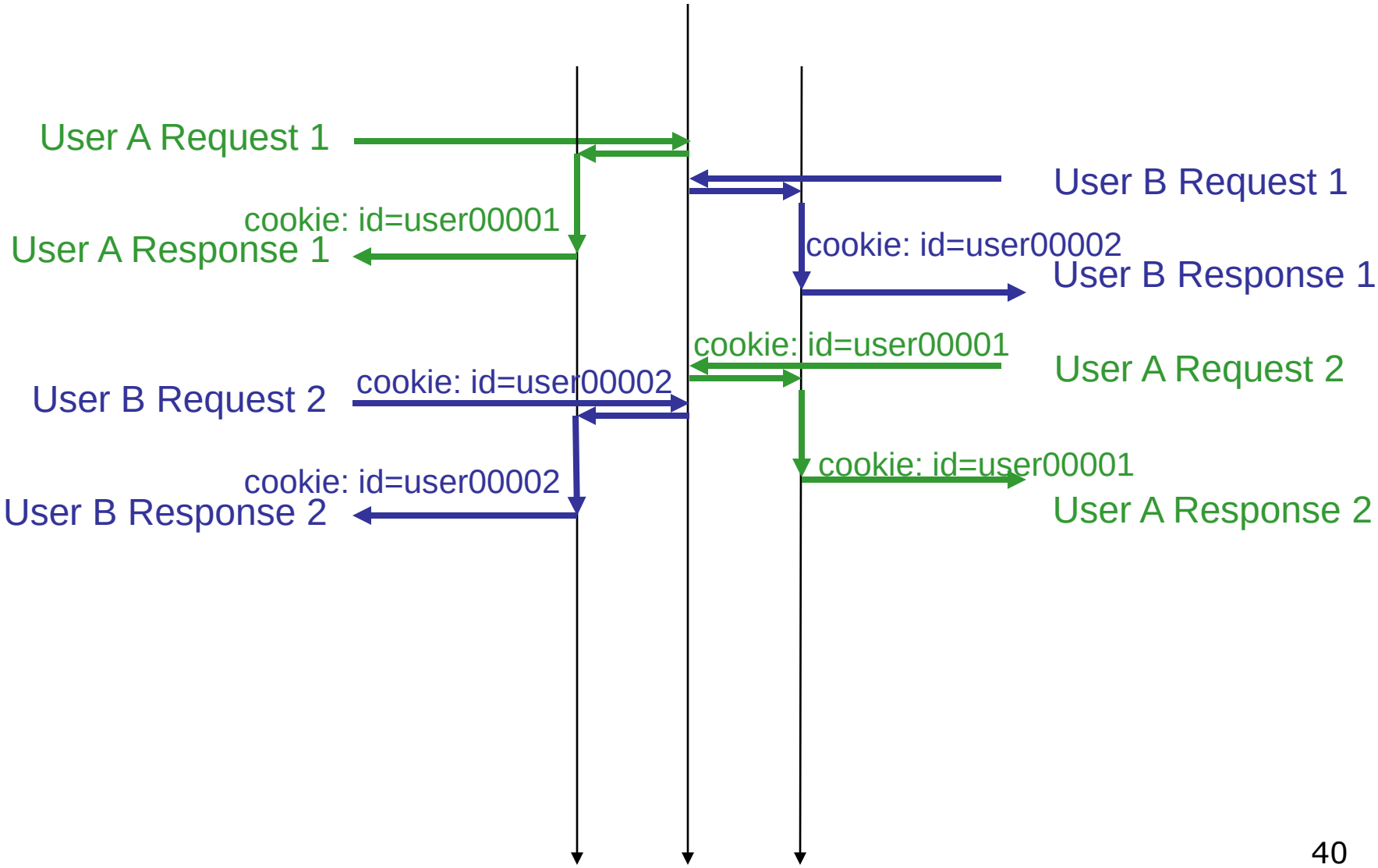
HTTP server sessions



HTTP server sessions

- How to recognize which requests belong to the same user?
  - look at client's IP address
  - in first response, send client a small but unique piece of data
  - ask client to send this back as part of the HTTP header of all following requests
  - piece of data is known as a (magic) **cookie**

HTTP server sessions



Cookie: bb.vu.nl response

```
HTTP/1.1 302 Moved Temporarily
Set-Cookie: ARPT=IZJNJNSbb3CYUQ; path=/
Date: Sun, 20 Jan 2008 20:24:23 GMT
Server: Apache/1.3.33 (Unix) mod_ssl/2.8.21 OpenSSL/0.9.7e
  mod_jk/1.2.4
Pragma: no-cache
Cache-Control: no-cache
Set-Cookie: session_id=@@BCCF1515B166A6BE2FF476EB20E9774F
Location: http://bb.vu.nl/nocookies.html
Content-Length: 0
Connection: close
Content-Type: application/octet-stream;charset=ISO-8859-1
```

Cookies

- Introduced in Mosaic browser (1994)
  - cookies were enabled by default
  - users were not informed when a site set a cookie
  - most users did not know about cookies at all
- Privacy issues became serious issue in 1996 after a publication in the Financial Times
- Now all major browsers allow users to delete cookies and to be alerted when cookies are set
- Many sites make privacy policies public on their site (P3P)

Cookies

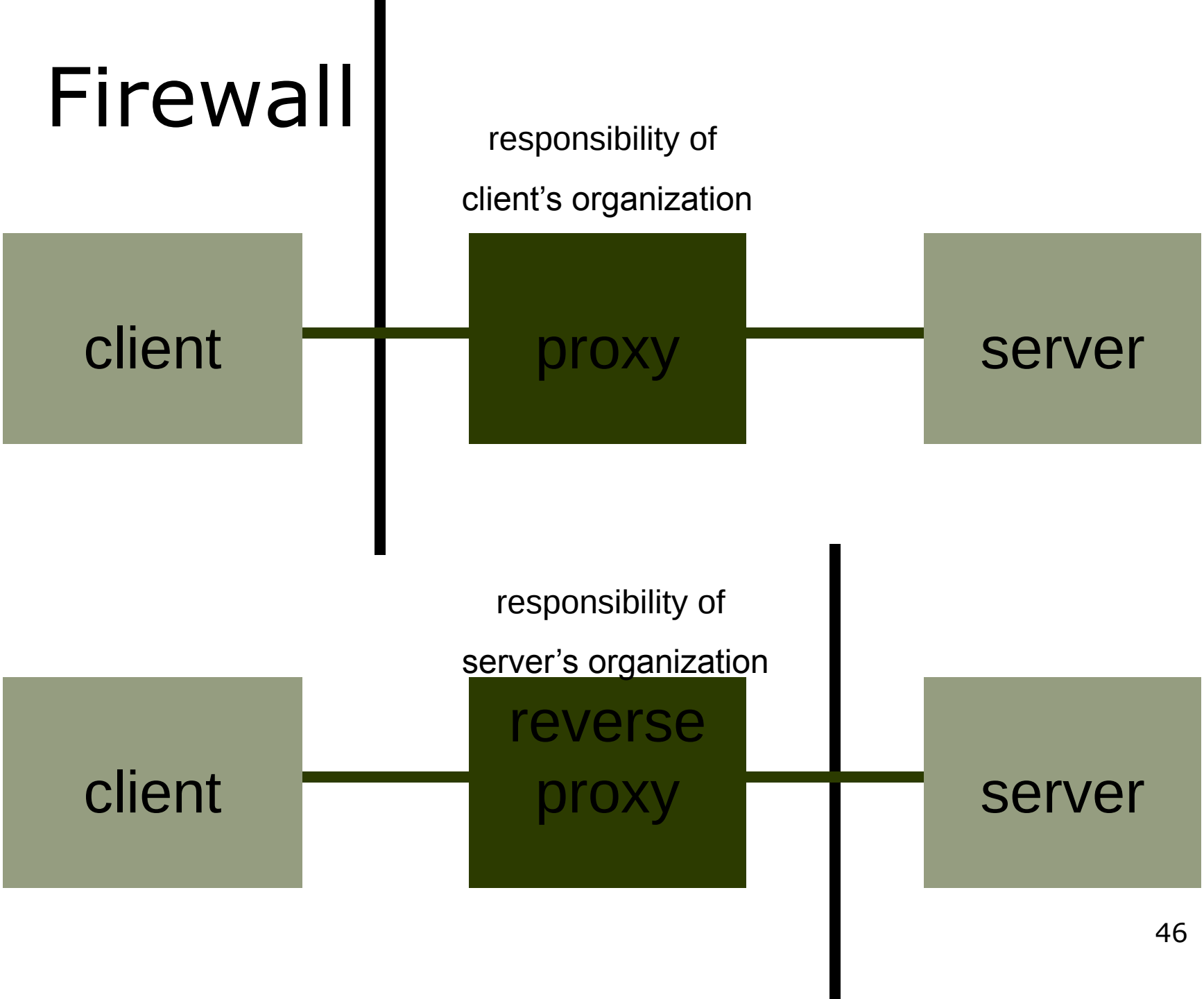
- Handy
  - Electronic shopping basket
  - Personalisation
    - user preferences
    - user profile
  - Authentication
- Tricky
  - User tracking across websites
  - Direct marketing
  - Privacy issues
- Note: sites may set cookies without knowing it or even using them…
- Check the cookies stored in your browser

Security issues

see also
guest lecture Thursday

Proxies & firewalls

- Some clients have no direct internet access to contact servers
  - Browser can use a **proxy** server
  - Content servers do not need to know
- Some servers have no direct internet access to be contacted (!)
  - Server can use a **reverse proxy** server
  - Clients do not need to know


Firewall
responsibility of
client's organization
client
proxy
server
responsibility of
server's organization
client
reverse
proxy
server

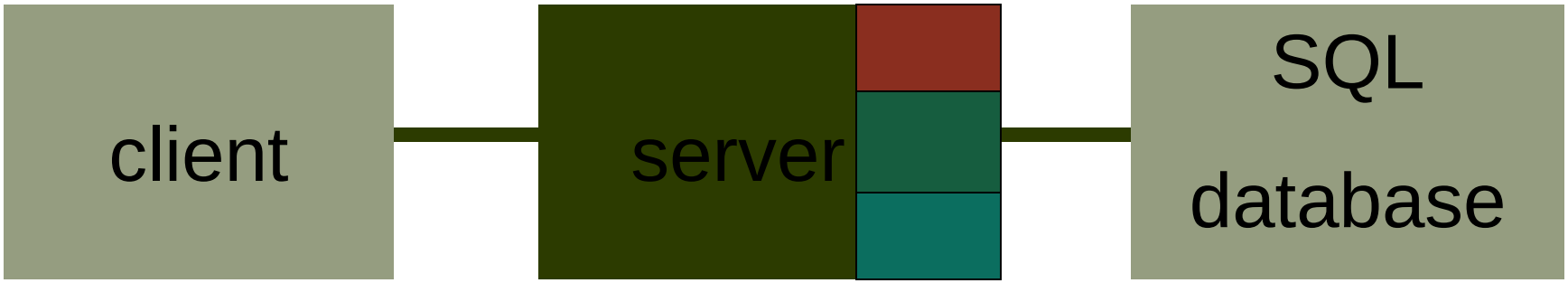
Authentication & encryption

- HTTP 1.0 Basic Access Authentication
  - username, password, content sent in plain text
- HTTP Digest Access Authentication
  - username, password encrypted
  - content still sent plain text
- HTTPS: HTTP entirely over secure layer
  - public key encryption, also for content
  - less vulnerable to man in the middle attacks

Man in the middle attack

- HTTPS requires web site to authenticate itself using a **certificate** stating its identity
- How do you know how to trust **certificate authority**?
  - many generally trusted authorities are known by your browser

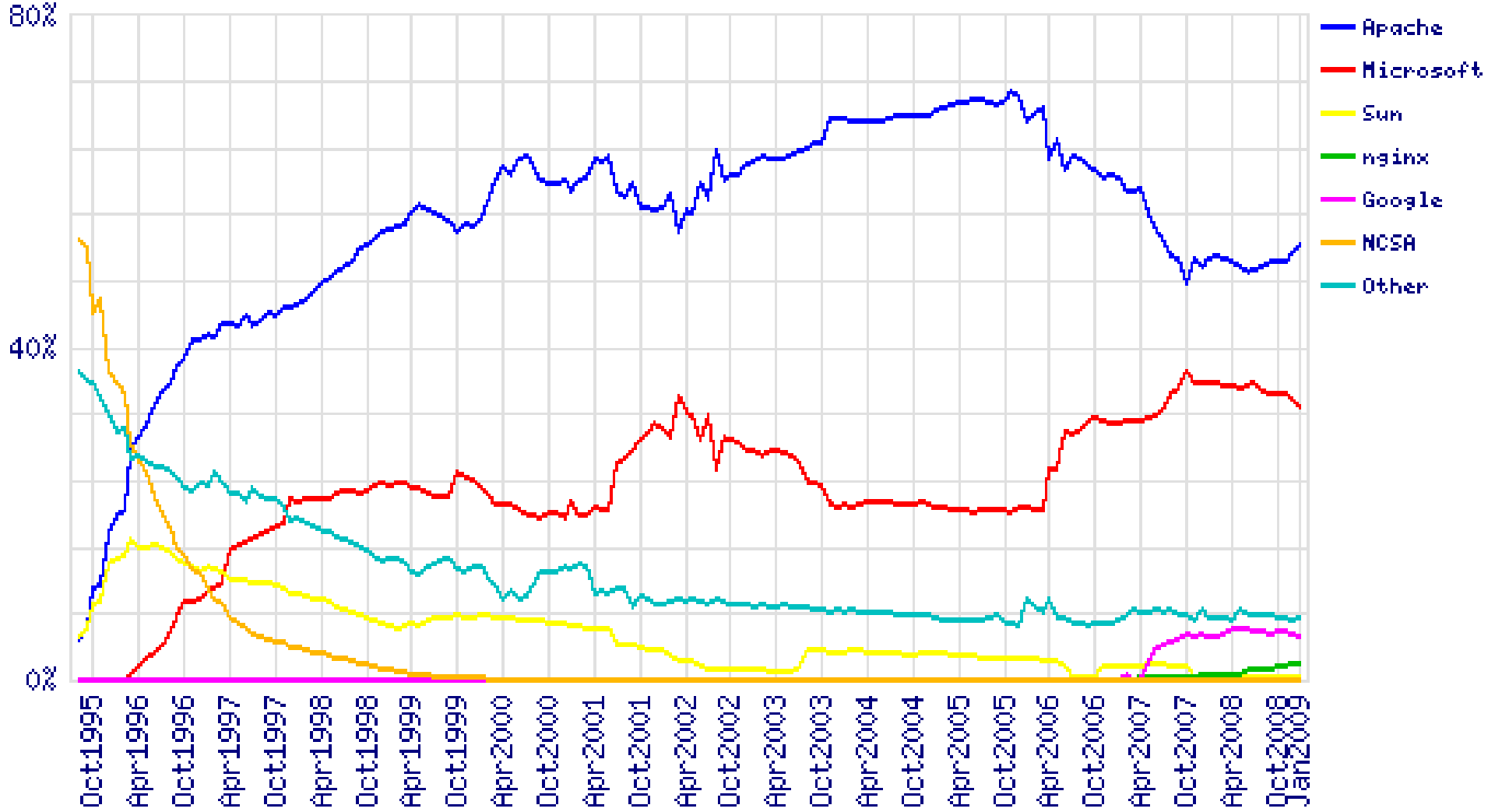
Database connectivity



- All frameworks provide ways to simplify generating HTML out of database content
  - Java Servlets, JSP
  - PHP
  - Content management systems
  - …

LAMP and the ubiquity of HTTP servers

- Typical web server needs:
  1. Operating system with good TCP/IP support
  2. HTTP server implementation
  3. Database to store content
  4. Framework for creating web pages from database content
- All these ingredients are currently commonly available (as open source software) and run on commodity PCs
- Frequently used combination is **L**inux, **A**pache, **M**ySQL and **P**HP (**LAMP**)
- Many sites are served by LAMP software running on old PC hardware …
- A "web server" is nothing special anymore!
  > 185 million servers (Netcraft, Jan 2009)


80%
40%
0%
Apache
Microsoft
Sun
nginx
Google
NCSA
Other
Oct1995
Apr1996
Oct1996
Apr1997
Oct1997
Apr1998
Oct1998
Apr1999
Oct1999
Apr2000
Oct2000
Apr2001
Oct2001
Apr2002
Oct2002
Apr2003
Oct2003
Apr2004
Oct2004
Apr2005
Oct2005
Apr2006
Oct2006
Apr2007
Oct2007
Apr2008
Oct2008
Jan2009

Learning goals

- Understand
  - Basis HTTP server functionality
  - Serving static HTML and other files
  - Serving dynamic content from software within a HTTP server
  - Serving dynamic content from external software
  - Be aware of security & privacy issues