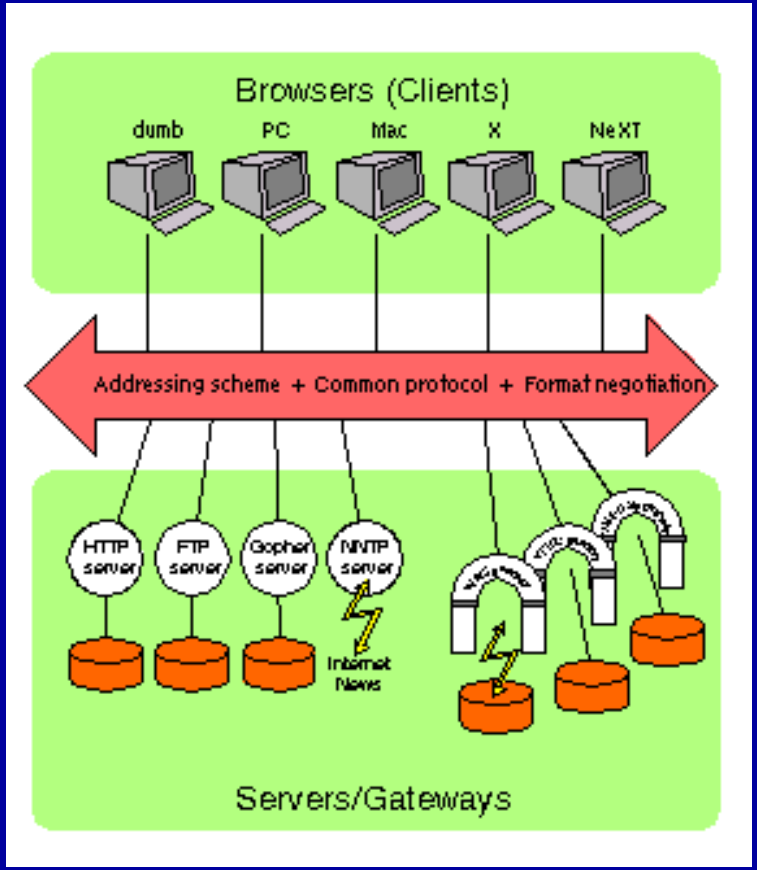
Web Programming

Based on Notes by D. Hollinger

Also Java Network Programming and Distributed Computing, Chs. 9,10

Also Online Java Tutorial, Sun.

World-Wide Web
(Tim Berners-Lee & Cailliau '92)



Topics

- HTTP – HyperText Transfer Protocol
- HTML – HyperText Markup Language
- URI – Uniform Resource Identifiers
  - URL – Uniform Resource Locators
  - URN – Uniform Resource Names
  - URC – Uniform Resource Citations
- Server-Side Programming
  - HTML Forms

See Online Resources

Only URLs are widely deployed in today's Web!

HTTP
Hypertext Transfer Protocol

Refs:

RFC 1945 (HTTP 1.0)

RFC 2616 (HTTP 1.1)

HTTP Usage

- HTTP is the protocol that supports communication between web browsers and web servers.
- A “Web Server” is a HTTP server
- We will look at HTTP Version 1.0 +

From the RFC

"HTTP is an application-level protocol with the lightness and speed necessary for distributed, hypermedia information systems."

Transport Independence

- The RFC states that the HTTP protocol generally takes place over a TCP connection, but the protocol itself is not dependent on a specific transport layer.

Request - Response

- HTTP has a simple structure:
  - client sends a request
  - server returns a reply.
- HTTP can support multiple request-reply exchanges over a single TCP connection.

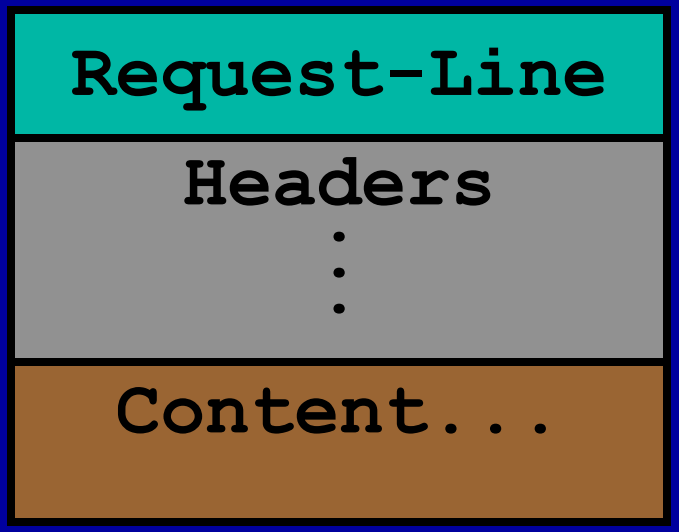
Well Known Address

- The “well known” TCP port for HTTP servers is port 80.
- Other ports can be used as well...

HTTP Versions

- The original version now goes by the name “HTTP Version 0.9”
  - HTTP 0.9 was used for many years.
- Starting with HTTP 1.0 the version number is part of every request.
- HTTP is still changing...

HTTP 1.0+ Request



- Lines of text (ASCII).
- Lines end with CRLF `"\r\n"`
- First line is called “Request-Line”

Request Line

Method URI HTTP-Version \r\n

- The request line contains 3 *tokens* (words).
- space characters " " separate the tokens.
- Newline (\n) seems to work by itself (but the protocol requires CRLF)

Request Method

- The Request Method can be:

GET	**HEAD**	**PUT**
POST	**DELETE**	**TRACE**
OPTIONS		

future expansion is supported

Methods

- GET: retrieve information identified by the URI.
- HEAD: retrieve meta-information about the URI.
- POST: send information to a URI and retrieve result.

Methods (cont.)

- PUT: Store information in location named by URI.
- DELETE: remove *entity* identified by URI.

More Methods

- TRACE: used to trace HTTP forwarding through proxies, tunnels, etc.
- OPTIONS: used to determine the capabilities of the server, or characteristics of a named resource.

Common Usage

- GET, HEAD and POST are supported everywhere.
- HTTP 1.1 servers often support PUT, DELETE, OPTIONS & TRACE.

URI: Uniform Resource Identifier

- URIs defined in RFC 2396.
- Absolute URI:

```
scheme://hostname[:port]/path
```

```
http://www.cs.rpi.edu:80/blah/foo
```

- Relative URI: `/path`

`/blah/foo`

No server mentioned

URI Usage

- When dealing with a HTTP 1.1 server, only a *path* is used (no scheme or hostname).
  - HTTP 1.1 servers are required to be capable of handling an absolute URI, but there are still some out there that won't…
- When dealing with a *proxy* HTTP server, an absolute URI is used.
  - client has to tell the proxy where to get the document!
  - *more on proxy servers in a bit….*

HTTP Version Number

"`HTTP/1.0`" or "`HTTP/1.1`"

HTTP 0.9 did not include a version number in a request line.

If a server gets a request line with no HTTP version number, it assumes 0.9

The Header Lines

- After the *Request-Line* come a number (possibly zero) of HTTP *headers*.
- Each header line contains an attribute name followed by a ":" followed by the attribute value.

Headers

- Request Headers provide information to the server about the client
  - what kind of client
  - what kind of content will be accepted
  - who is making the request
- There can be 0 headers

Example HTTP Headers

```
Accept: text/html

From: neytmann@cybersurg.com

User-Agent: Mozilla/4.0

Referer: http://foo.com/blah
```

End of the Headers

- Each header ends with a CRLF
- The end of the header section is marked with a blank line.
  - **`just CRLF`**
- For GET and HEAD requests, the end of the headers is the end of the request!

POST

- A POST request includes some *content (some data)* after the headers (after the blank line).
- There is no format for the data (just raw bytes).
- A POST request must include a Content-Length line in the headers:

```
Content-Length: 267
```

Example GET Request

```
GET /~hollingd/testanswers.html HTTP/1.0
Accept: */*
User-Agent: Internet Explorer
From: cheater@cheaters.org
Referer: http://foo.com/
```

There is a blank line here!

Example POST Request

```
POST /~hollingd/changegrade.cgi HTTP/1.1
Accept: */*
User-Agent: SecretAgent V2.3
Content-length: 35
Referer: http://monte.cs.rpi.edu/blah

stuid=6660182722&item=test1&grade=99
```

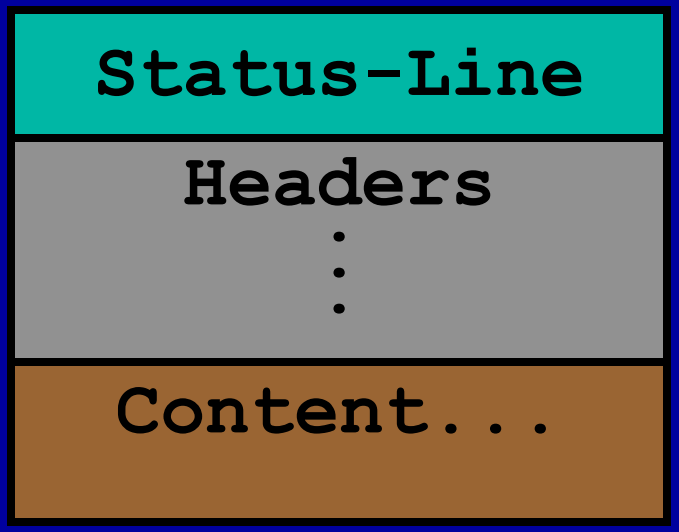
Typical Method Usage

GET used to retrieve an HTML document.

HEAD used to find out if a document has changed.

POST used to submit a form.

HTTP Response



- ASCII Status Line
- Headers Section
- Content can be anything (not just text)
  - typically is HTML document or some kind of image.

Response Status Line

HTTP-Version *Status-Code* *Message*

- *Status Code* is 3 digit number (for computers)
- Message is text (for humans)

Status Codes

- `1xx` Informational
- `2xx` Success
- `3xx` Redirection
- `4xx` Client Error
- `5xx` Server Error

Example Status Lines

```
HTTP/1.0 200 OK

HTTP/1.0 301 Moved Permanently

HTTP/1.0 400 Bad Request

HTTP/1.0 500 Internal Server Error
```

Response Headers

- Provide the client with information about the returned *entity* (document).
  - what kind of document
  - how big the document is
  - how the document is encoded
  - when the document was last modified
- Response headers end with blank line

Response Header Examples

```
Date: Wed, 30 Jan 2002 12:48:17 EST

Server: Apache/1.17

Content-Type: text/html

Content-Length: 1756

Content-Encoding: gzip
```

Content

- Content can be anything (sequence of raw bytes).
- Content-Length header is required for any response that includes content.
- Content-Type header also required.

Single Request/Reply

- The client sends a complete request.
- The server sends back the entire reply.
- The server closes it's socket.

- If the client needs another document it must open a new connection.

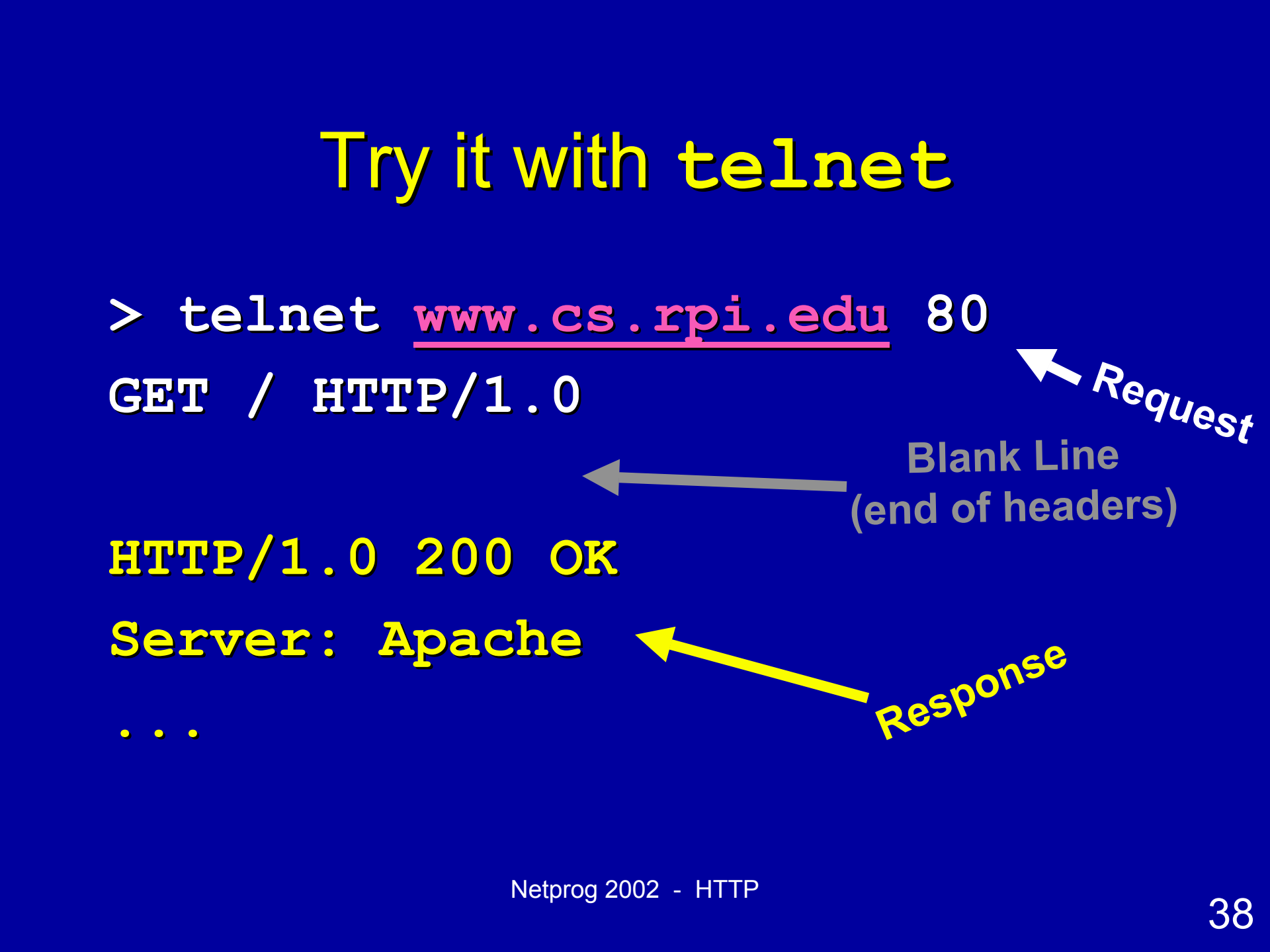
Persistent Connections

- HTTP 1.1 supports persistent connections (this is supposed to be the default).
- Multiple requests can be handled.
- Most servers seem to close the connection after the first response…

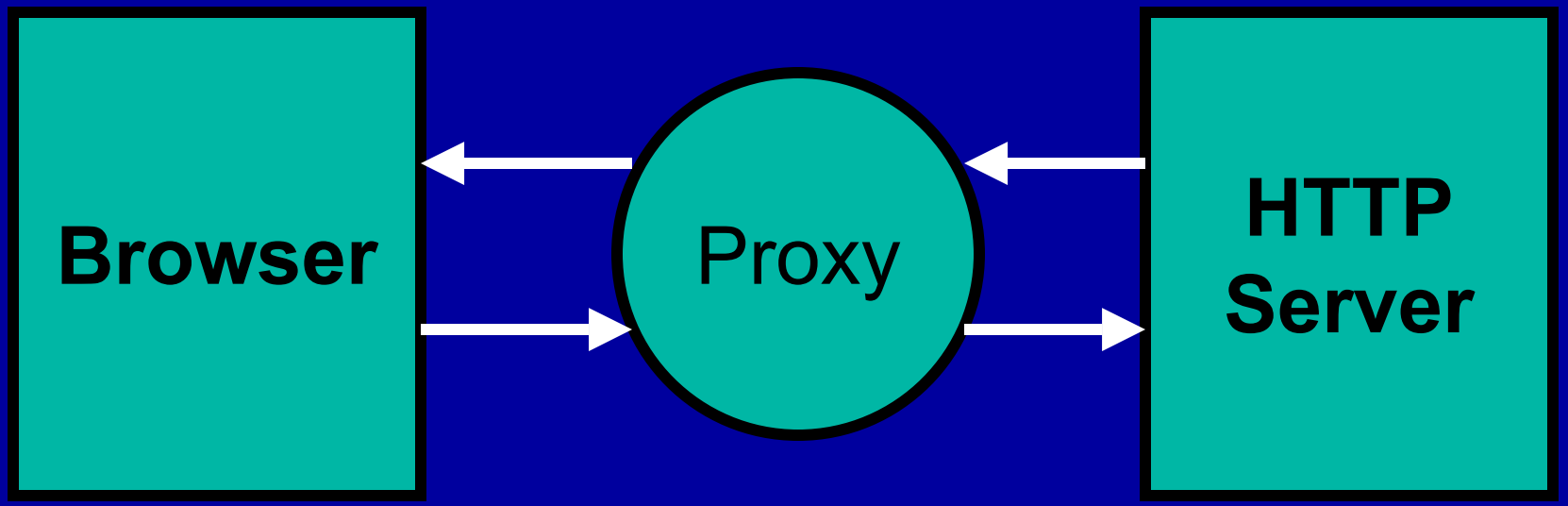
Try it with `telnet`

```
> telnet www.cs.rpi.edu 80
GET / HTTP/1.0

HTTP/1.0 200 OK
Server: Apache
...
```



HTTP Proxy Server



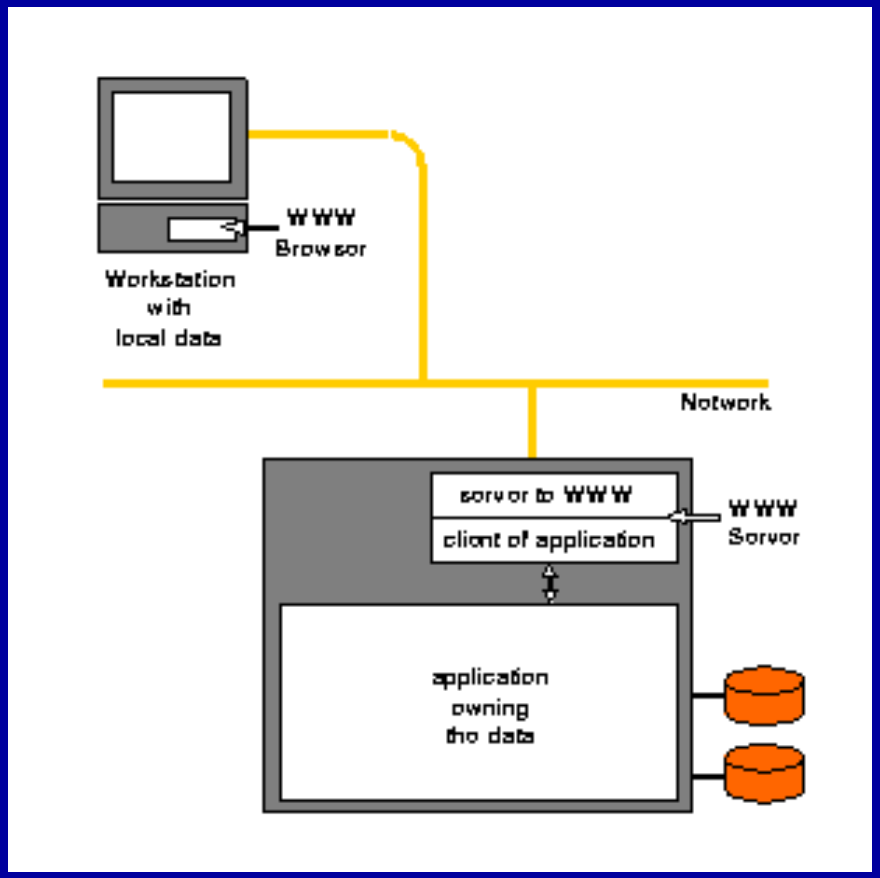
Tyba: A simple (and incomplete) HTTP Server Implementation in Java

- See:

  http://yangtze.cs.uiuc.edu/~cvarela/tyba/

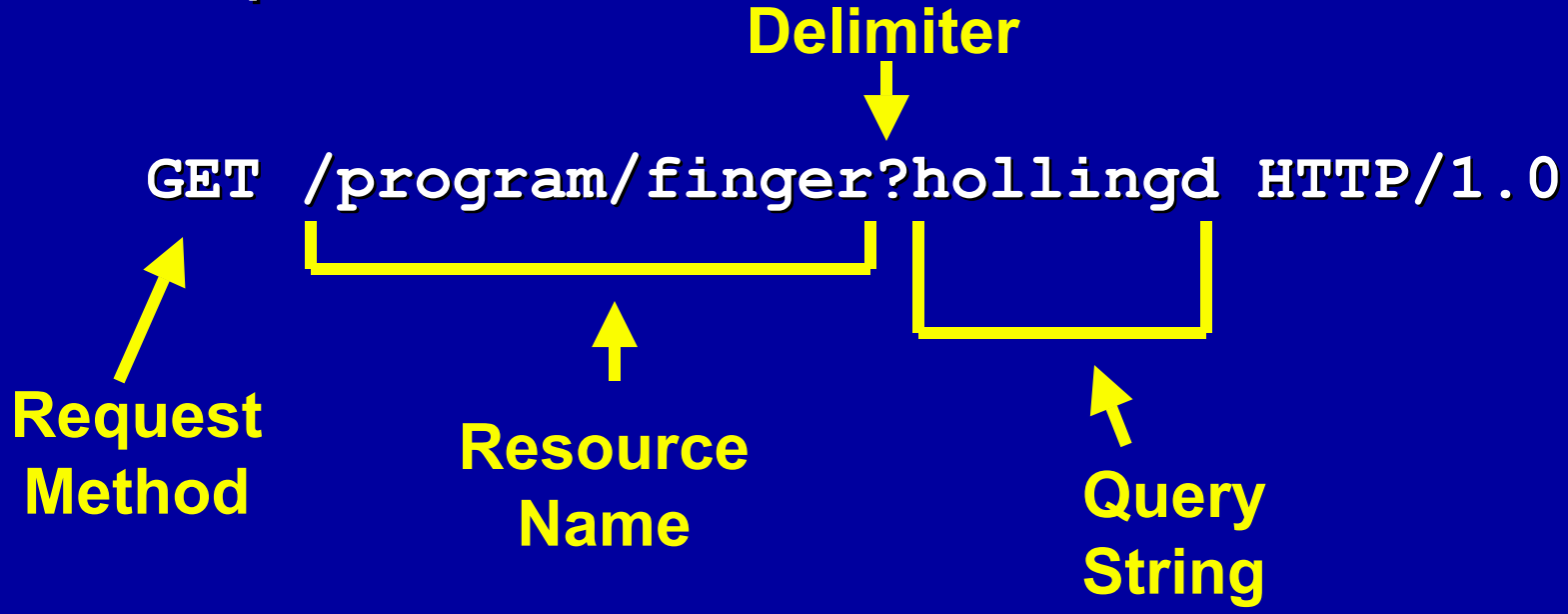
Server-Side Programming

Web Server Architecture
(Berners-Lee & Cailliau '92)



Request Method: Get

- GET requests can include a *query string* as part of the URL:



/program/finger?hollingd

- The web server treats everything before the '?' delimiter as the resource name
- In this case the resource name is the name of a program. (could be a CGI script, a servlet, or your own HTTP server)
- Everything after the '?' is a string that is passed to the server program (in the case of CGI and servlets)

Simple GET queries - ISINDEX

- You can put an <ISINDEX> tag inside an HTML document.
- The browser will create a text box that allows the user to enter a single string.
- If an ACTION is specified in the ISINDEX tag, when the user presses ***Enter,*** a request will be sent to the server specified as the ACTION.

ISINDEX Example

```
Enter a string:
<ISINDEX ACTION=http://foo.com/search>
Press Enter to submit your query.
```

If you enter the string “blahblah”, the browser will send a request to the http server at `foo.com` that looks like this:

```
GET /search?blahblah HTTP/1.1
```

URL-encoding

- Browsers use an encoding when sending query strings that include special characters.
  - Most nonalphanumeric characters are encoded as a '%' followed by 2 ASCII encoded hex digits.
  - '=' (which is hex 3D) becomes "%3D"
  - '&' becomes "%26"

More URL encoding

- The space character ‘ ‘ is replaced by ‘+’.
  - Why?
- The ‘+’ character is replaced by “**%2B**”

Example:

“**foo=6 + 7**” becomes “**foo%3D6+%2B+7**”

URL Encoding in Java

- `java.net.URLEncoder` class

```
String original = “foo=6 + 7”;
System.out.println(
  URLEncoder.encode(original));
```

foo%3D6+%2B+7

URL Decoding in Java

- `java.net.URLDecoder` class

```
String encoded = "foo%3D6+%2B+7";
System.out.println(
  URLDecoder.decode(encoded));
```

foo=6 + 7

Beyond ISINDEX - Forms

- Many Web services require more than a simple field in the web form.
- HTML includes support for forms:
  - lots of field types
  - user answers all kinds of annoying questions
  - entire contents of form must be stuck together and put in the query by the web client.

Form Fields

- Each field within a form has a name and a value.
- The browser creates a query that includes a sequence of "`name=value`" sub-strings and sticks them together separated by the '`&`' character.

Form fields and encoding

- 2 fields - name and occupation.
- If user types in “Dave H.” as the name and “none” for occupation, the query would look like this:

```
“name=Dave+H%2E&occupation=none”
```

HTML Forms

- Each form includes a METHOD that determines what http method is used to submit the request.
- Each form includes an ACTION that determines where the request is made.

An HTML Form

```
<FORM METHOD=GET
  ACTION=http://foo.com/signup>
Name:
<INPUT TYPE=TEXT NAME=name><BR>
Occupation:
<INPUT TYPE=TEXT
  NAME=occupation><BR>
<INPUT TYPE=SUBMIT>
</FORM>
```

What the server will get

- The query will be a URL-encoded string containing the name,value pairs of all form fields.
- The server program (or a CGI script, or a servlet) must decode the query and separate the individual fields.

HTTP Method: POST

- The HTTP POST method delivers data from the browser as the content of the request.
- The GET method delivers data (query) as part of the URI.

GET vs. POST

- When using forms it's generally better to use POST:
  - there are limits on the maximum size of a GET query string (environment variable)
  - a post query string doesn't show up in the browser as part of the current URL.

HTML Form using POST

Set the form method to POST instead of GET.

```
<FORM METHOD=POST ACTION=...>
```

The browser will take care of the details...

Server reading POST

- If the request is a POST, the query is coming in the body of the HTTP request.
- The “Content-length” header tells us how much data to read.

HTML Forms (in more detail)

Form Elements

- Each HTML form contains the following:
  - `<FORM>, </FORM>` tags
  - The `<FORM>` tag has two required attributes:
    - `METHOD` specifies the HTTP method used to send the request to the server (when the user submits the form).
    - `ACTION` specifies the URL the request is sent to.

FORM Method

- We have seen the two common methods used:
  - **`GET`**: any user input is submitted as part of the URI following a “?”.

```
GET foo?name=joe&cookie=oreo HTTP/1.0
```

  - **`POST`**: any user input is submitted as the content of the request (after the HTTP headers).

Sample POST Request

```
POST /dir/foo HTTP/1.0
User-Agent: Netscape
Content-Length: 20
Cookie: favorite=chocolatechip
ECACChamps: RPI

name=joe&cookie=oreo
```

Form ACTION attribute

- The ACTION attribute specifies the URL to which the request is sent. Some examples:

```
ACTION="http://www.cs.rpi.edu/CGI_BIN/foo"

ACTION="myprog"

ACTION="mailto:hollingd@cs.rpi.edu"
```

<FORM> Tag Examples

```
<FORM   METHOD="POST"
 ACTION="http://www.foo.com/cgi-bin/myprog">

<FORM   METHOD="GET" ACTION="/cgi-bin/myprog">

<FORM   METHOD="POST"
 ACTION="mailto:shirley@pres.rpi.edu">
```

Inside a form

- Between the <FORM> and </FORM> tags you define the text and *fields* that make up the form.
- You can use normal HTML tags to format the text however you want.
- The *fields* are defined using tags as well.

Form Fields

- There are a variety of types of form fields:
  - text fields: text, password, textarea
  - radio buttons
  - checkboxs
  - buttons: user defined, submit, reset (clear)
  - hidden fields

Input Fields

- There are a number of field types that allow the user to type in a string value as input.

- Each field is created using an `<INPUT>` tag with the attribute `TYPE`.

Input Attributes

- The `TYPE` attribute is used to specify what kind of input is allowed: `TEXT`, `PASSWORD`, `FILE`, ...
- Every `INPUT` tag must have a `NAME` attribute.

TEXT Fields

- TEXT is the most common type of input:
  - user can enter a single line of text.
  - Additional attributes can specify:
    - the maximum string length - `MAXLENGTH`
    - the size of the input box drawn by the browser - `SIZE`
    - a default value - `VALUE`

TEXT INPUT Examples

```
<INPUT TYPE=TEXT NAME=FOO>


<INPUT TYPE="TEXT"
       NAME="PIZZA"
       SIZE=10
       MAXLENGTH=20
       VALUE="Pepperoni">
```

An example form

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
Your Name:
<INPUT TYPE=TEXT NAME="Name"><BR>

Your Age:
<INPUT TYPE=TEXT NAME'"Age"><BR>

</FORM>
```

Submission Buttons

- Another type of INPUT field is the submission button.
- When a user clicks on a submit button the browser submits the contents of all other fields to a web server using the METHOD and ACTION attributes.

```
<INPUT TYPE=SUBMIT VALUE="press me">
```

Reset Buttons

- An INPUT of type RESET tells the browser to display a button that will clear all the fields in the form.

```
<INPUT TYPE=RESET
       VALUE="press me to clear form">
```

A Complete Form Example

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
Your Name:
<INPUT TYPE=TEXT NAME="Name"><BR>

Your Age: <INPUT TYPE=TEXT NAME="Age"><BR>

<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET>
</FORM>
```

Tables and Forms

Tables are often used to make forms look pretty - remember that you can use any HTML tags to control formatting of a form.

Table/Form example

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
<TABLE><TR>
   <TD>Your Name: </TD>
   <TD><INPUT TYPE=TEXT NAME="Name"></TD>
</TR><TR>
   <TD>Your Age:</TD>
   <TD> <INPUT TYPE=TEXT NAME="Age"></TD>
</TR><TR>
   <TD><INPUT TYPE=SUBMIT VALUE="Submit"></TD>
   <TD><INPUT TYPE=RESET></TD>
</TR></TABLE>
</FORM>
```

Other Inputs

- Checkboxes
  - present user with items that can be selected or deselected. Each checkbox has a name and a value and can be initially selected/deselected
  - Example checkbox definitions:

```
<INPUT TYPE=checkbox name=chocchip value=1>
  <INPUT TYPE=checkbox name=oreo value=1>
```

Checkbox example

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
Select all the cookies you want to order:<BR>

<INPUT TYPE=CHECKBOX NAME=Oreo Value=1>
   Oreo<BR>
<INPUT TYPE=CHECKBOX NAME=Oatmeal Value=1>
  Oatmeal<BR>
<INPUT TYPE=CHECKBOX CHECKED NAME=ChocChip
   Value=1>
  Chocolate Chip<BR>

<INPUT TYPE=SUBMIT VALUE=Submit>
</FORM>
```

Radio Buttons

- Radio Buttons are like checkbox except that the user can select only one item at a time.
- All radio buttons in a group have the same `NAME`.

```
<INPUT TYPE=radio name=cookie value=chocchip>
<INPUT TYPE=radio name=cookie value=oreo>
<INPUT TYPE=radio name=cookie value=oatmeal>
```

Radio Button Example

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
Select all the cookies you want to order:<BR>

<INPUT TYPE=RADIO NAME=Cookie Value=Oreo> Oreo <BR>
<INPUT TYPE=RADIO NAME=Cookie Value=Oatmeal> Oatmeal
   <BR>
<INPUT TYPE=RADIO CHECKED NAME=Cookie Value=ChocChip>
   ChocolateChip<BR>

<INPUT TYPE=SUBMIT VALUE=Submit>
</FORM>
```

Multiline Text

- The TEXTAREA tag creates an area where the user can submit multiple lines of text.
- This is **not** another type of <INPUT> tag!

TEXTAREA Attributes

- Each TEXTAREA tag has attributes NAME, COLS and ROWS.

```
<TEXTAREA name=address rows=5 cols=40>
default text goes here (or can be empty)
</TEXTAREA>
```

TEXTAREA example

```
<FORM METHOD=POST ACTION=cgi-bin/foo>
Please enter your address in the space
  provided:<BR>
<TEXTAREA NAME=address COLS=40 ROWS=5>
</TEXTAREA>
<BR>
<INPUT TYPE=SUBMIT VALUE=Submit>
</FORM>
```

textarea1.html

Form Submission

- When the user presses on a SUBMIT button the following happens:
  - browser uses the FORM method and action attributes to construct a request.
  - A query string is built using the (name,value) pairs from each form element.
  - Query string is URL-encoded.

Input Submissions

- For each checkbox selected the name,value pair is sent.
- For all checkboxes that are not selected - nothing is sent.
- A single name,value pair is sent for each group of radio buttons.

Other Form Field Types

- There are other form field types:
  - SELECT - pulldown menu or scrolled list of choices.
  - Image Buttons
  - Push Buttons (choice of submit buttons)

Hidden Fields

- Nothing is displayed by the browser.
- The name,value are sent along with the submission request.

```
<INPUT TYPE=HIDDEN
       NAME=SECRET
       VALUE=AGENT>
```

Hidden does not mean secure!

- Anyone can look at the source of an HTML document.
  - hidden fields are part of the document!
- If a form uses GET, all the name/value pairs are sent as part of the URI
  - URI shows up in the browser as the location of the current page