


In the good old days...

- Years ago... the WWW was made up of (mostly) static documents.
 - Each URL corresponded to a single file stored on some hard disk.
- Today - many of the documents on the WWW are built at request time.
 - URL doesn't correspond to a single file.

Dynamic Documents

- Dynamic Documents can provide:
 - automation of web site maintenance
 - customized advertising
 - database access
 - shopping carts
 - date and time service
 - jobs for IT students.

Cool idea!



Web Programming

- Writing programs that create dynamic documents has become very important.
- There are a couple of general approaches:
 - Develop a “smart” web server
 - SSI, scripting, server APIs.
 - Have web server run external programs.
 - CGI

↑
“external” means a different program than the actual web server program (but on the same machine)

Smart Server Approach

- Take a general purpose Web server (that can handle static documents) and have it *process* requested documents as it sends them to the client.
- The documents could contain commands that the server understands (the server includes some kind of interpreter).

Example Smart Server

- Have the server read each HTML file as it sends it to the client.
- The server could look for something like this:
`<SERVERCODE> some command </SERVERCODE>`
- The server doesn't send this part to the client, instead it interprets the command and sends the result to the client.
- Everything else is sent normally.

A Possible Example

```
<H1>Welcome to TimeDate.com</H1>

<H2>The Time is:
    <SERVERCODE> Time </SERVERCODE>
</H2>

<H2>The Date is:
    <SERVERCODE> Date </SERVERCODE>
</H2>

<SERVERCODE>ShowSponsor</SERVERCODE>
```

The smart server needs to know what the commands **Time**, **Date** and how **Sponsor** mean.

Real Life - Server Side Includes

- Many real web servers support this idea (but not the syntax I've shown).
- Server Side Includes (SSI) provides a set of simple commands that a server will interpret.
- Typically the server is configured to look for commands only in specially marked documents (so normal documents aren't slowed down).

SSI Directives

- SSI commands are called *directives*
- Directives are embedded in HTML comments. A comment looks like this:

```
<!-- this is an HTML comment -->
```

- A directive looks like this:

```
<!--#command parameter="arg"-->
```

Some SSI Directives

echo: inserts the value of an environment variable into the page.

SSI servers keep a number of useful things in environment variables:

DOCUMENT_NAME, DOCUMENT_URL

Example usage:

This page is located at

```
<!--#echo var="DOCUMENT_URL"-->.
```

SSI Directives

include: inserts the contents of a text file.

```
<!--#include file="banner.html">
```

lastmod: inserts the time and date that a file was last modified.

```
Last modified <!--#lastmod file="foo.html">
```

SSI Example

```
<!--#INCLUDE FILE="header"-->
```

It is now:

```
<!--#config timefmt="%I:%M %p (%Z)"-->
```

```
<!--#echo var="DATE_LOCAL"-->
```

```
<BR>
```

Today is:

```
<!--#config timefmt="%A, %B %e, %Y"-->
```

```
<!--#echo var="DATE_LOCAL"--><BR>
```

```
<!--#INCLUDE FILE="footer"-->
```

```
<!--#config timefmt="%D"-->
```

This file last modified

```
<!--#echo var="LAST_MODIFIED"-->
```

More Power

- Some servers support elaborate scripting languages.
- Scripts are embedded in HTML documents, the server interprets the script:
 - Microsoft *Active Server Pages* (ASP)
 - JScript, VBScript, PerlScript
 - *Java Server Pages* (JSP)
 - PHP
 - ColdFusion
 - There are others...

Server Mapping and APIs

- Some servers include a programming interface that allows us to extend the capabilities of the server by writing modules.
- Specific URLs are mapped to specific modules instead of to files.
- You *could* develop a web application as a module and merge it with the web server.

↑
Not commonly done!

External Programs

- Another approach is to provide a standard interface between external programs and web servers.
 - We can run the same program from any web server.
 - The web server handles all the **http**, we focus on the special service only.
 - It doesn't matter what language we use to write the external program.

Common Gateway Interface

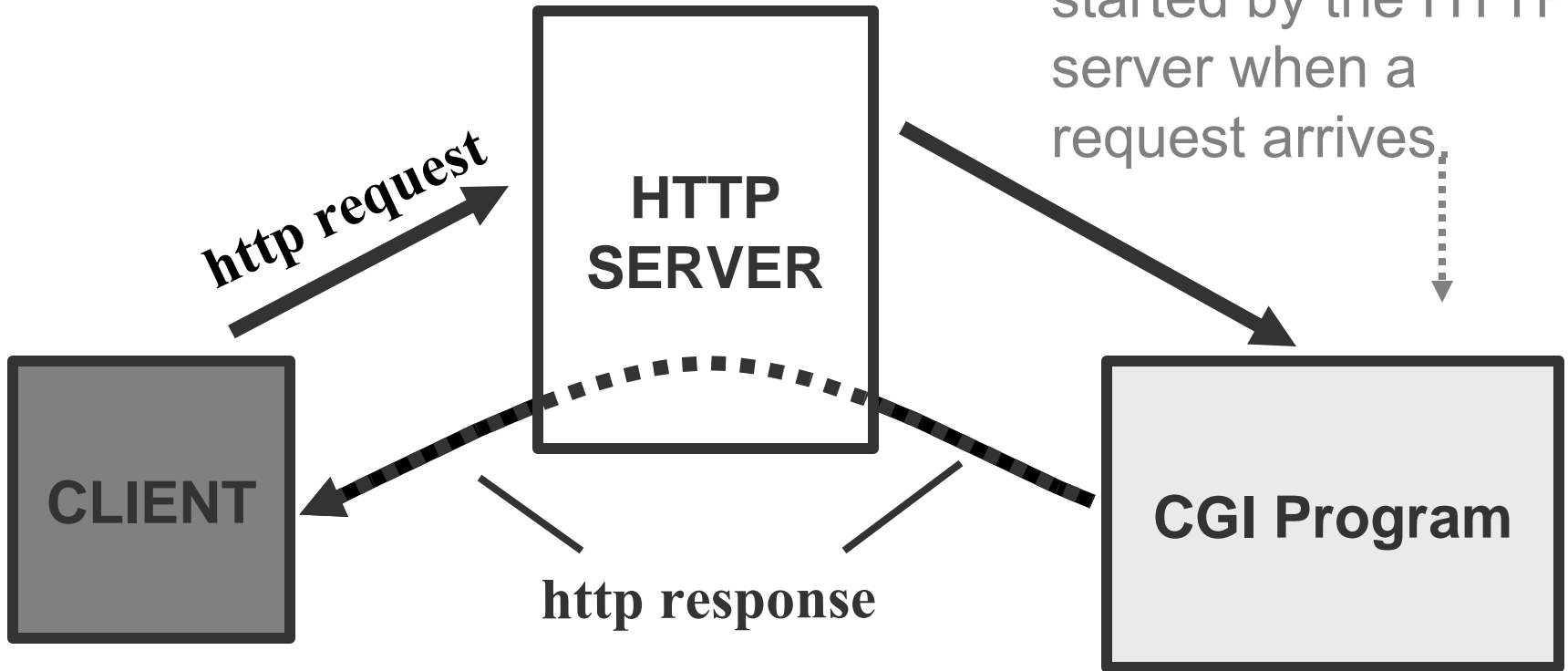
- CGI is a standard interface to external programs supported by most web servers.
- The interface that is defined by CGI includes:
 - Identification of the service (external program).
 - Mechanism for passing the request to the external program.
 - How the response generated by the external program makes it back to the client.

CGI Programming

- CGI programs are often written in scripting languages (Perl, Tcl, Python), although you can write a CGI program in any language.
- We will concentrate on Perl
 - great for handling text strings
 - HTML generator library
 - Database access
 - tons of examples

CGI Setup

The CGI program is started by the HTTP server when a request arrives.



CGI URLs

- There is some mapping (association) between URLs and CGI programs provided by the web sever.
- The exact mapping is not standardized (web server admin can set it up).
- The server need to know whether to send back the requested document, or to run it!

Common URL to CGI mappings

- requests that start with `/CGI-BIN/` , `/cgi-bin/` or `/cgi/`, etc. refer to CGI programs (not to static documents).

GET /cgi-bin/login

-or-

- file names that end in `.cgi` or `.pl` refer to CGI programs.

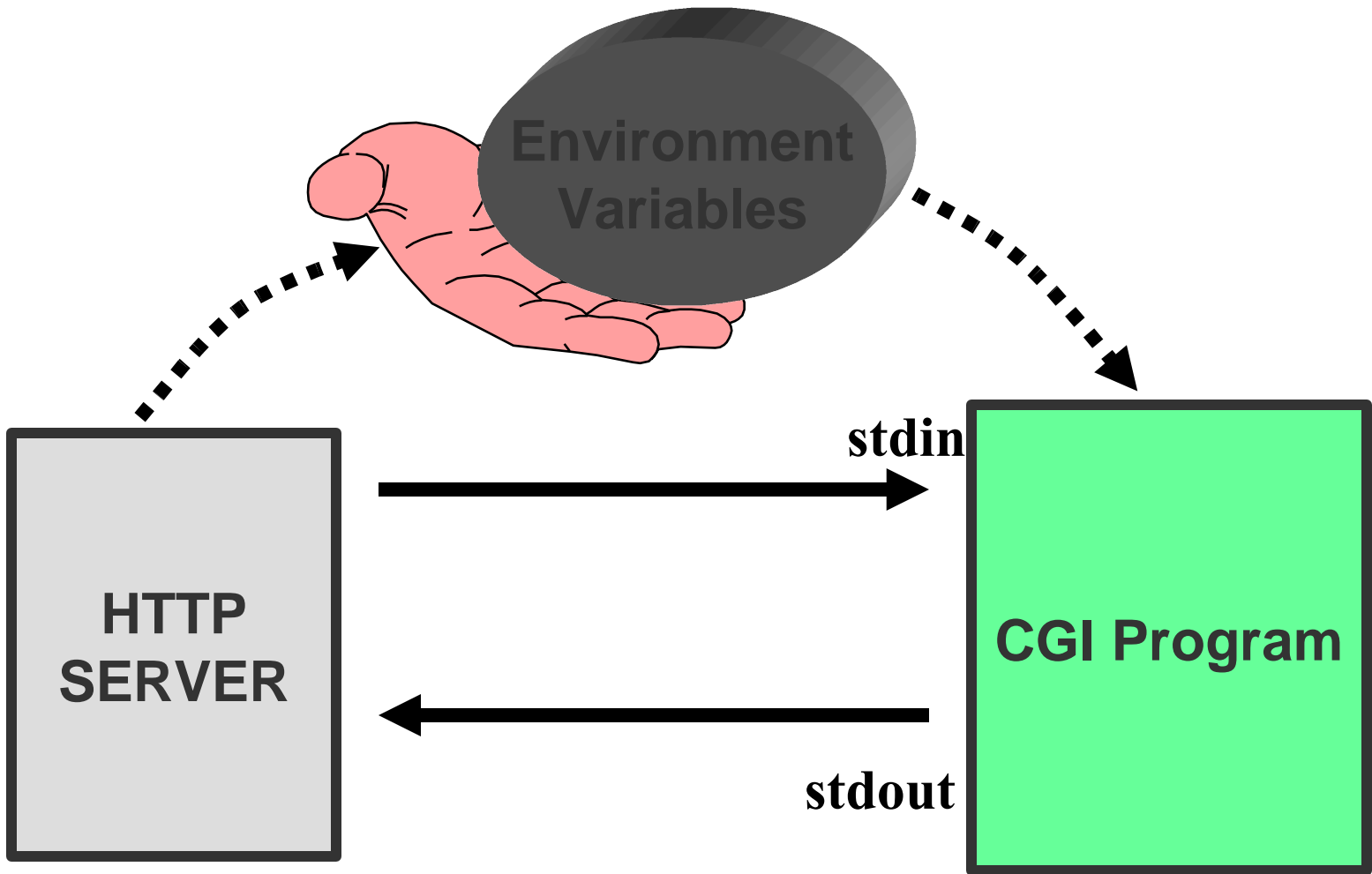
GET /login.pl

Request → CGI program

- The web server sets some *environment variables* with information about the request.
- The web server starts up the CGI program.
- The CGI program gets information about the request from environment variables.

STDIN, STDOUT

- Before starting the CGI program, the web server sets up pipes so that `stdin` comes from the web server and `stdout` goes to the web server.
- In some cases part of the request is read from `stdin`.
- Anything written to `stdout` is forwarded by the web server to the client.



Important CGI Environment Variables

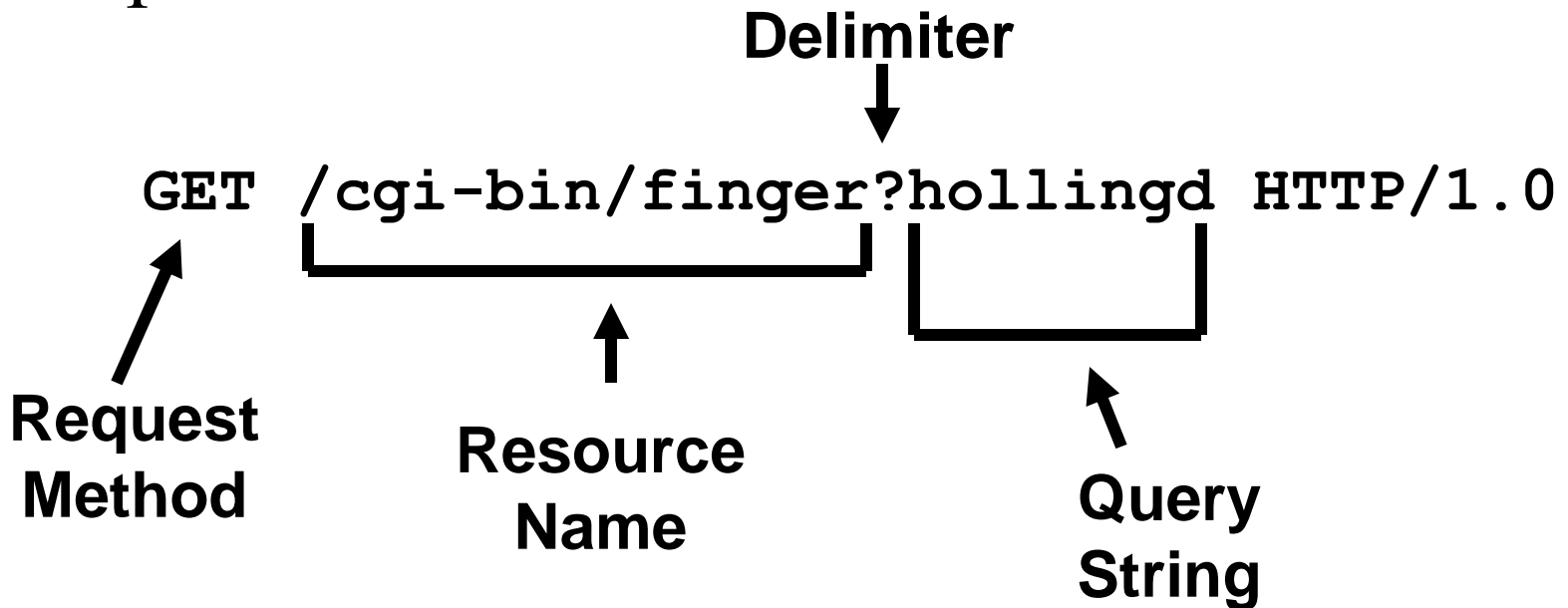
REQUEST_METHOD

QUERY_STRING

CONTENT_LENGTH

Request Method: Get

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



`/cgi-bin/finger?hollind`

- The web server treats everything before the ‘?’ delimiter as the resource name
- In this case the resource name is the name of a program.
- Everything after the ‘?’ is a string that is passed to the CGI program.

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.cgi>
```

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.cgi?blahblah HTTP/1.1
```

What the CGI sees

- The CGI program can find out about the request method (GET) from *environment variables*.
- The CGI program can also get the query string from *environment variables*.
- *This code is already written, lots of libraries are available to take care of these tasks.*

HTTP Response

- The CGI program can send back an HTTP status line and headers.
 - *often we let fancy CGI libraries handle this for us.*
- The CGI program created the content of the response and prints it to STDOUT.
 - typically this is an HTML document.

Simple GET handler in C++

```
int main() {
    char *method, *query;
    method = getenv("REQUEST_METHOD");
    if (method==NULL) ... /* error! */
    query = getenv("QUERY_STRING");
    cout << "Content-type: text/html\r\n\r\n";
    cout << "<H1>Your query was " << query <<
    "</H1>\n";
    return(0);
}
```

URL-encoding

- Browsers use an encoding when sending query strings:
 - Most non-alphanumeric 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 `+`.
- The `+` character is replaced by `%2B`
- Example:

`"foo=6 + 7"`

becomes

`"foo%3D6+%2B+7"`

URL Decoding

- A CGI program typically needs to decode the query string (to get back to the original string entered by the user in a form).
- Once again, we have libraries to do all this!

Beyond ISINDEX - Forms

- Many Web services require more than a simple ISINDEX.
- 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 QUERY_STRING by the Web server.

Form Field Review

- Each field within a form has a name/id and a value.
- The browser creates a query that includes a sequence of “**name=value**” substrings and sticks them together separated by the ‘&’ character.

HTML Form Review

- Each form includes 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.cgi>  
Name :  
<INPUT TYPE=TEXT NAME=name><BR>  
Occupation :  
<INPUT TYPE=TEXT NAME=occupation><BR>  
<INPUT TYPE=SUBMIT>  
</FORM>
```

What a CGI will get

- The query will be a URL-encoded string containing the name,value pairs of all form fields.
- The CGI must decode the query and separate the individual fields.

HTTP POST Method

- The query string is sent as the *content* of the request.
- The content follows the blank line that ends the request headers (just like in a response).
- The default encoding is URLencoding.
 - there are other encodings...
- There is always a Content-length header.

HTTP Post Example



A login form with a green background. It contains two input fields: the first is labeled 'Username:' and contains the text 'Dave'; the second is labeled 'Password:' and contains the text 'evad'. Below the password field is a button labeled 'Login'.

```
POST /cgi-bin/wlogin HTTP/1.1
```

```
User-agent: Mozilla
```

```
Content-length: 27
```

```
username=Dave&password=evad
```

POST vs. GET

- 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 URL.
- When using forms it's generally better to use POST:
 - there are limits on the maximum size of a GET query string (environment variable limitations)
 - 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...

CGI reading POST

- If REQUEST_METHOD is a POST, the query is coming in STDIN.
- The environment variable CONTENT_LENGTH tells us how much data to read.
- *These details are handled by CGI libraries!*