Network Programming:
The Tiny Web Server

Tiny Web Server

- **Tiny Web server described in textbook**
  - Tiny is a sequential Web server
  - Serves static and dynamic content to real browsers
    - text files, HTML files, GIF, PNG, and JPEG images
  - 239 lines of commented C code
  - Only GET requests
  - Not as complete or robust as a real Web server
    - You can break it with poorly-formed HTTP requests (e.g., terminate lines with “\n” instead of “\r\n”)

Tiny Operation

- **Accept connection from client**
- **Read request from client (via connected socket)**
- **Split into <method> <uri> <version>**
  - If method not GET, then return error
- **If URI contains "`cgi-bin`" then serve dynamic content**
  - (Would do wrong thing if had file "`abcgi-bingo.html`")
  - Fork process to execute program
  - NOT SAFE! (`/cgi-bin/../../usr/bin/halt`)
- **Otherwise serve static content**
  - Copy file to output

Tiny Serving Static Content

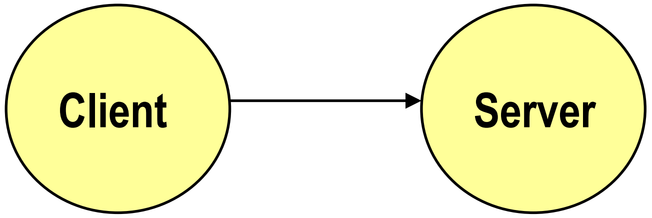
```c
void serve_static(int fd, char *filename, int filesize)
{
    int srcfd;
    char *srcp, filetype[MAXLINE], buf[MAXBUF];

    /* Send response headers to client */
    get_filetype(filename, filetype);
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);
    sprintf(buf, "%sConnection: close\r\n", buf);
    sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);
    sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);
    Rio_writen(fd, buf, strlen(buf));

    /* Send response body to client */
    srcfd = Open(filename, O_RDONLY, 0);
    srcp = Mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0);
    Close(srcfd);
    Rio_writen(fd, srcp, filesize);
    Munmap(srcp, filesize);
}
```

tiny.c

Serving Dynamic Content

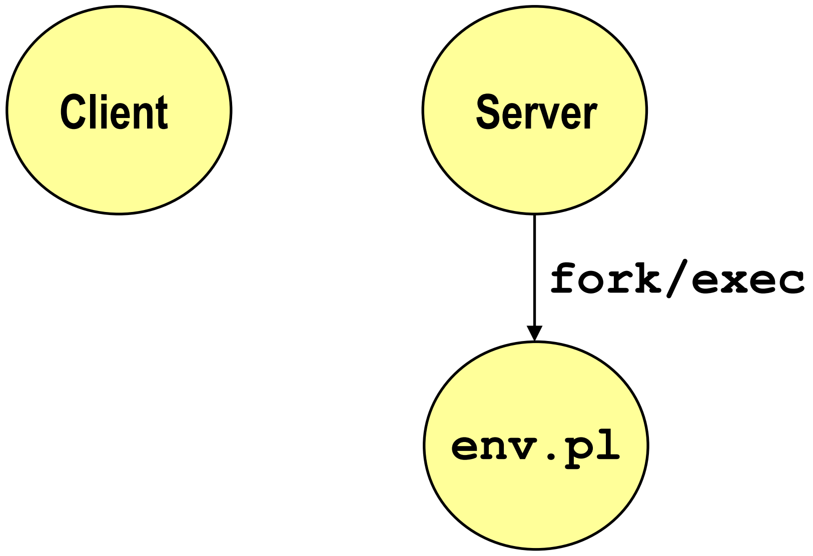
- Client sends request to server
- If request URI contains the string "`cgi-bin`", the Tiny server assumes that the request is for dynamic content

```
GET /cgi-bin/env.pl HTTP/1.1
```



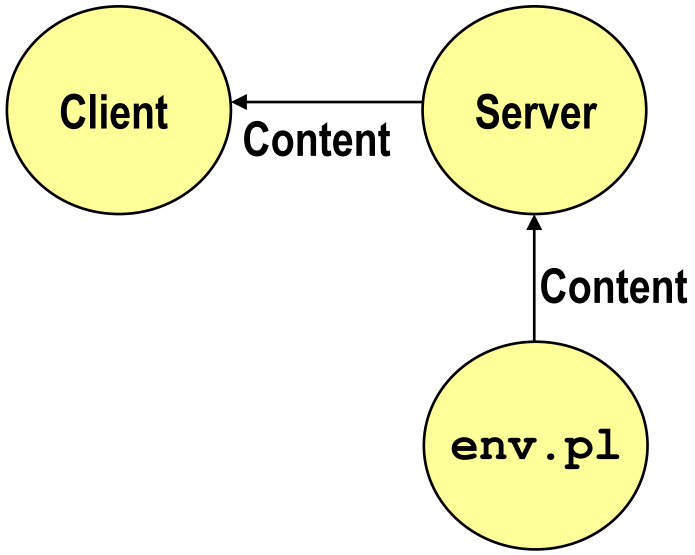
Serving Dynamic Content (cont)

- The server creates a child process and runs the program identified by the URI in that process



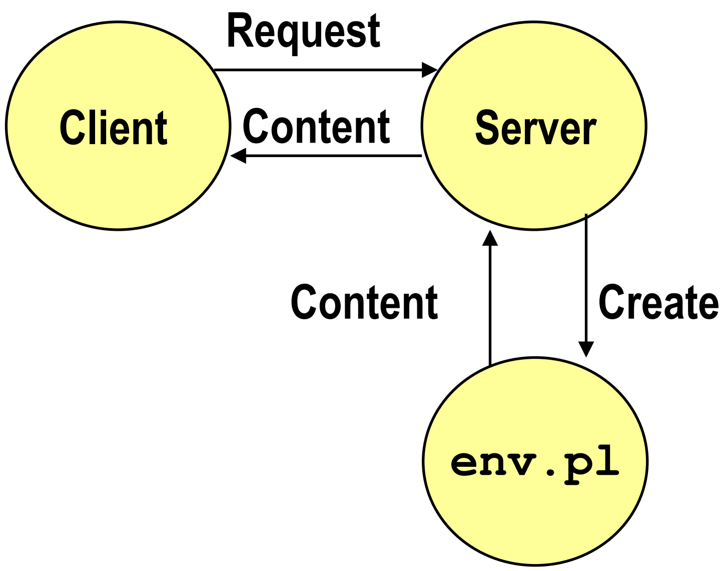
Serving Dynamic Content (cont)

- The child runs and generates the dynamic content
- The server captures the content of the child and forwards it without modification to the client



Issues in Serving Dynamic Content

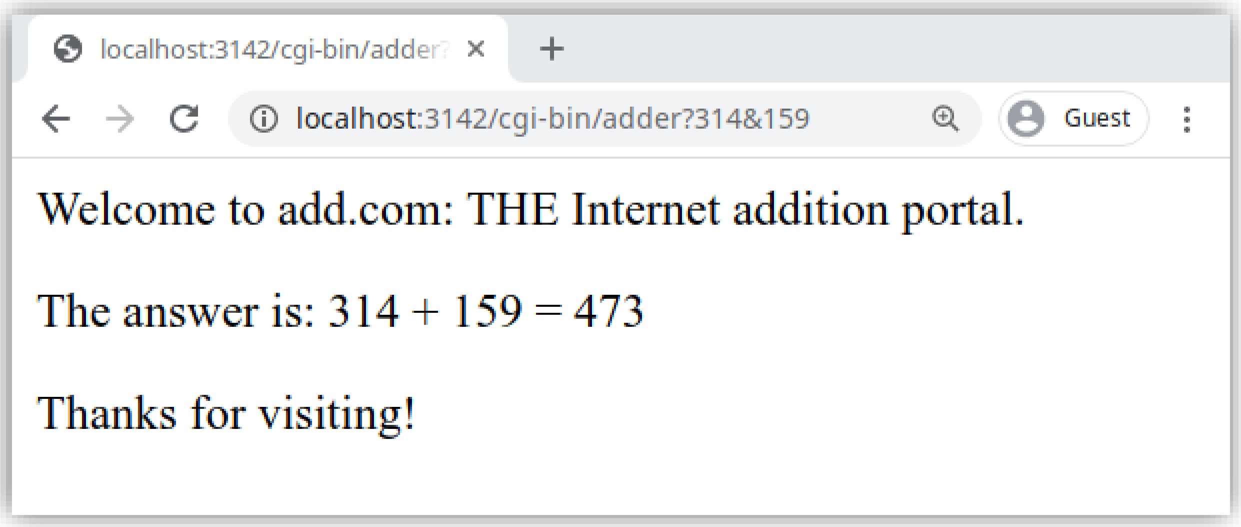
- How does the client pass program arguments to the server?
- How does the server pass these arguments to the child?
- How does the server pass other info relevant to the request to the child?
- How does the server capture the content produced by the child?
- These issues are addressed by the **Common Gateway Interface (CGI)** specification.



CGI

- **Because the children are written according to the CGI spec, they are often called *CGI programs/scripts.***
- **However, CGI really defines a simple standard for transferring information between the client (browser), the server, and the child process.**
- **CGI is the original standard for generating dynamic content. Has been largely replaced by other, faster techniques:**
  - E.g., fastCGI, Apache modules, Java servlets, Rails controllers
  - Avoid having to create process on the fly (expensive and slow).

The add.com Experience

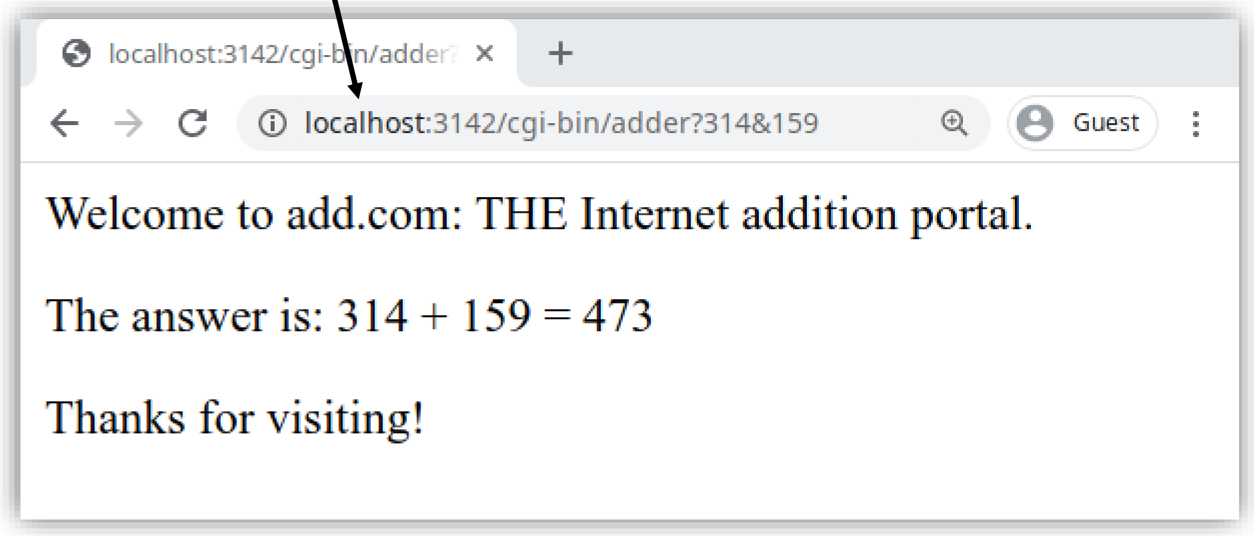


localhost:3142/cgi-bin/adder?314&159

```
Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!
```

The add.com Experience

host



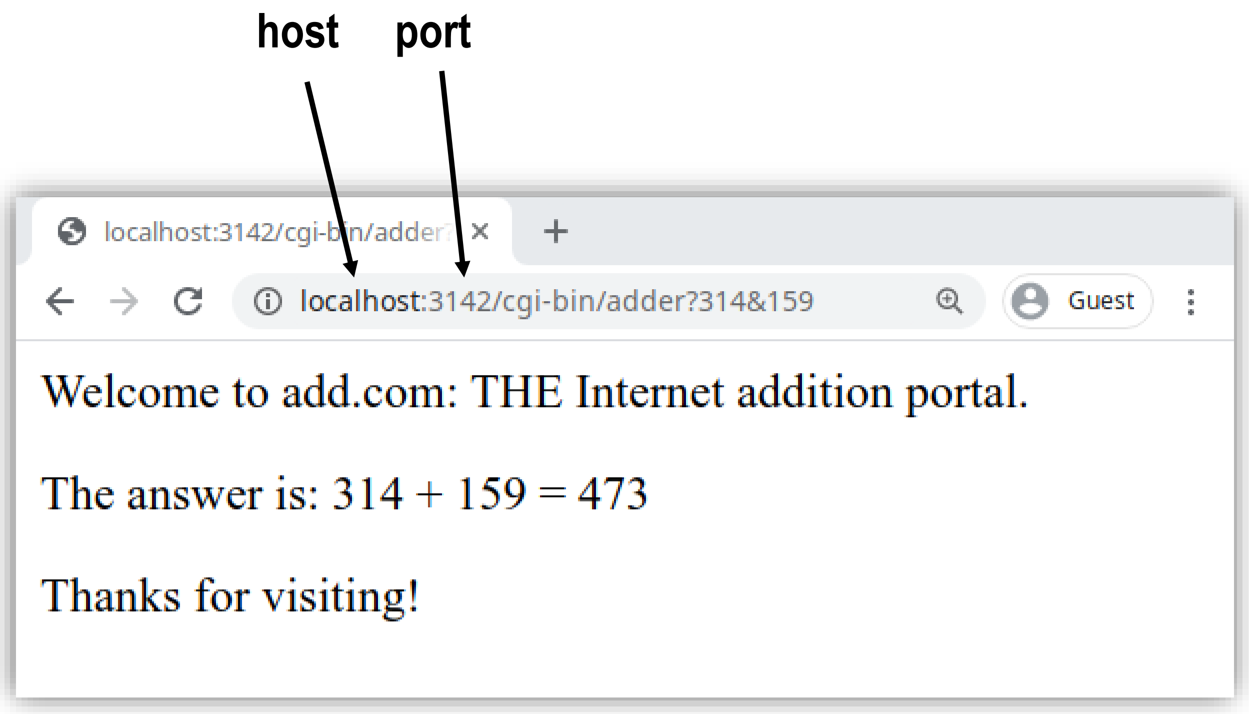
localhost:3142/cgi-bin/adder?314&159

Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!

The add.com Experience

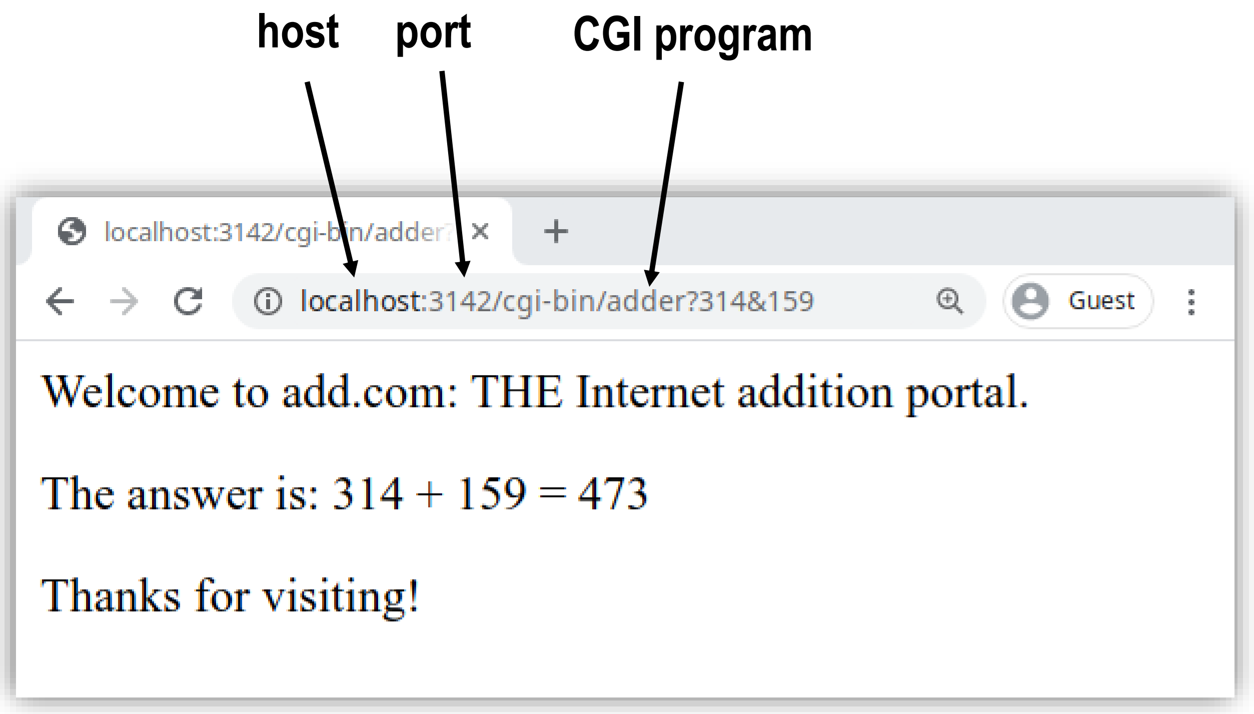

host
port
localhost:3142/cgi-bin/adder?
localhost:3142/cgi-bin/adder?314&159
Guest

Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!

The add.com Experience



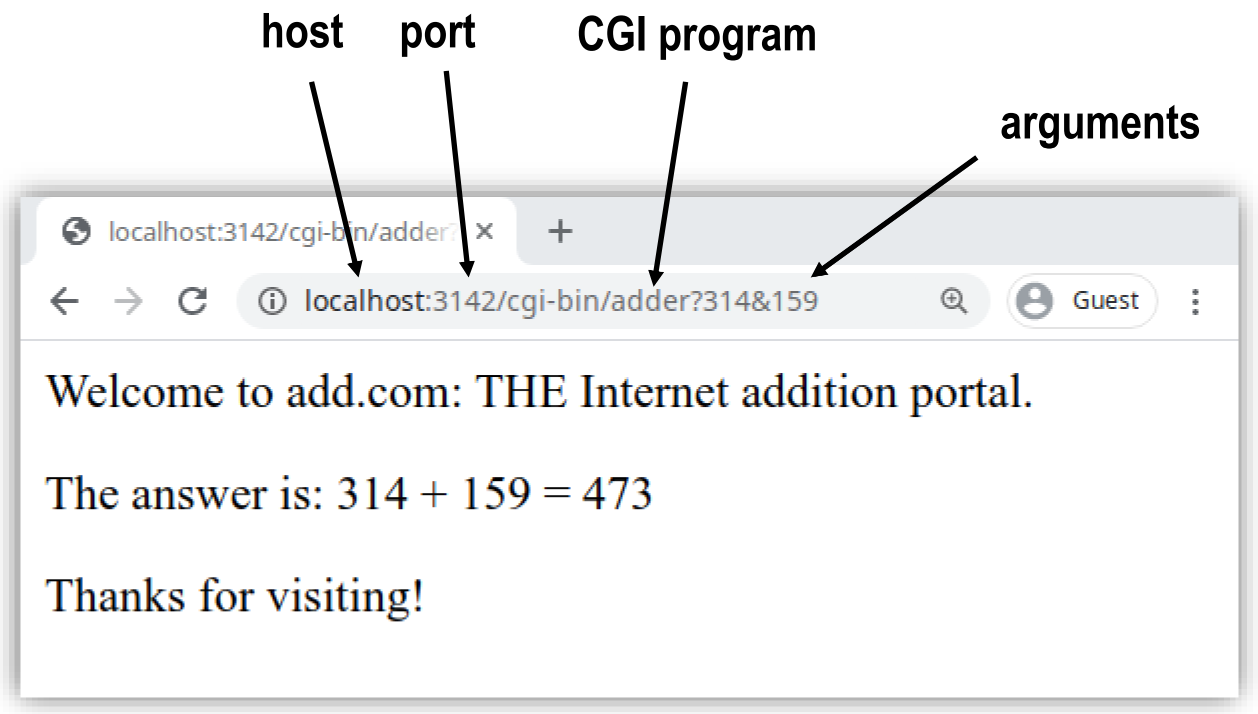
localhost:3142/cgi-bin/adder?314&159

Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!

The add.com Experience

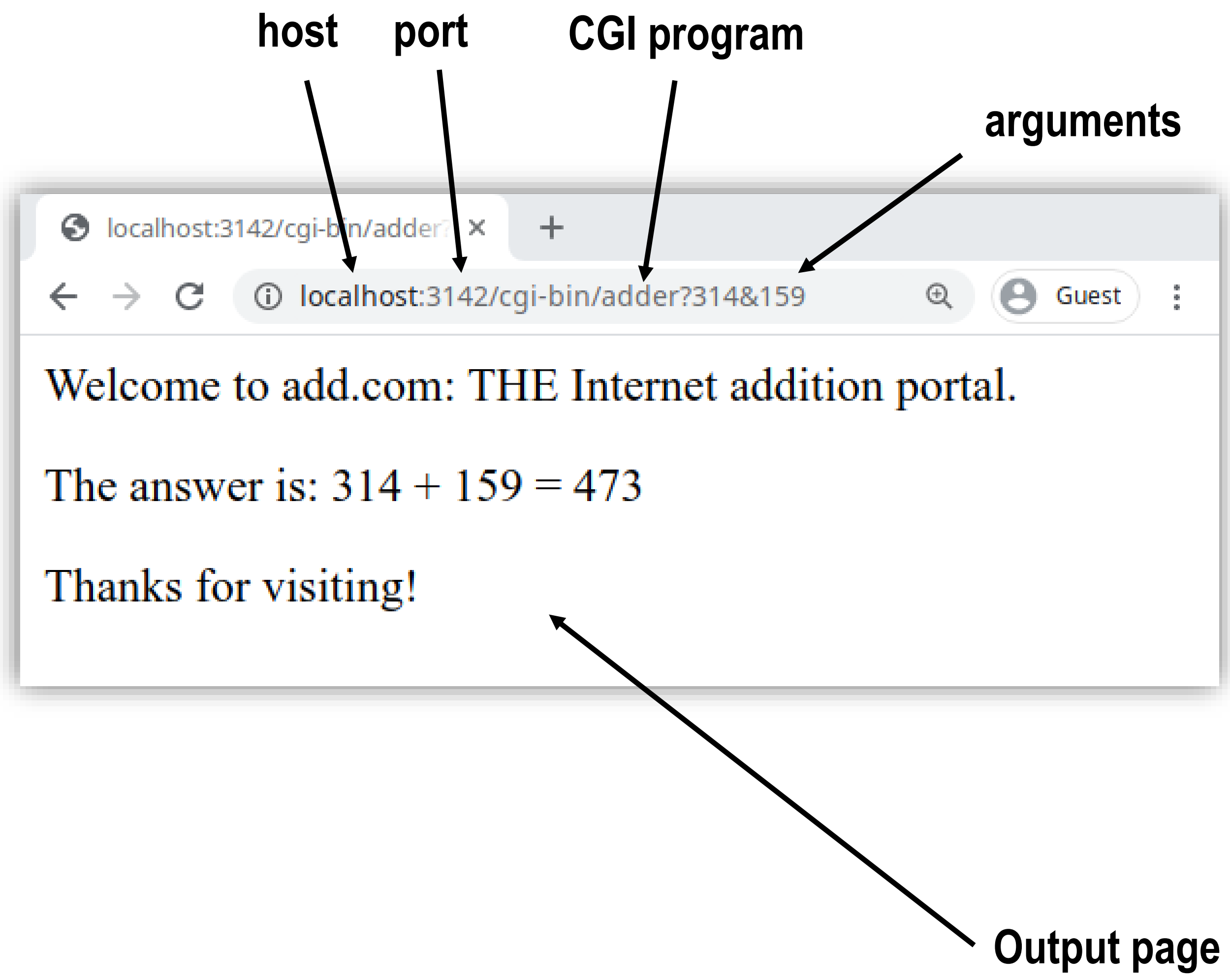

host
port
CGI program
arguments
localhost:3142/cgi-bin/adder?
localhost:3142/cgi-bin/adder?314&159
Guest

Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!

The add.com Experience



```
localhost:3142/cgi-bin/adder?314&159
```

Welcome to add.com: THE Internet addition portal.

The answer is: 314 + 159 = 473

Thanks for visiting!

Serving Dynamic Content With GET

- **<u>Question:</u> How does the client pass arguments to the server?**
- **<u>Answer:</u> The arguments are appended to the URI**

- **Can be encoded directly in a URL typed to a browser or a URL in an HTML link**
  - `http://add.com/cgi-bin/adder?15213&18213`
  - `adder` is the CGI program on the server that will do the addition.
  - argument list starts with `"?"`
  - arguments separated by `"&"`
  - spaces represented by `"+" or "%20"`

Serving Dynamic Content With GET

- **URL suffix:**
  - `cgi-bin/adder?15213&18213`
- **Result displayed on browser:**

```
Welcome to add.com: THE Internet
addition portal.

The answer is: 15213 + 18213 = 33426

Thanks for visiting!
```

Serving Dynamic Content With GET

- **Question: How does the server pass these arguments to the child?**
- **Answer: In environment variable `QUERY_STRING`**
  - A single string containing everything after the "?"
  - For add: `QUERY_STRING` = "`15213&18213`"

```
    /* Extract the two arguments */
    if ((buf = getenv("QUERY_STRING")) != NULL) {
        p = strchr(buf, '&');
        *p = '\0';
        strcpy(arg1, buf);
        strcpy(arg2, p+1);
        n1 = atoi(arg1);
        n2 = atoi(arg2);
    }
```

adder.c

Serving Dynamic Content with GET

- **Question:** How does the server capture the content produced by the child?
- **Answer:** The child generates its output on `stdout`. Server uses `dup2` to redirect `stdout` to its connected socket.

```
void serve_dynamic(int fd, char *filename, char *cgiargs)
{
    char buf[MAXLINE], *emptylist[] = { NULL };

    /* Return first part of HTTP response */
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Server: Tiny Web Server\r\n");
    Rio_writen(fd, buf, strlen(buf));

    if (Fork() == 0) { /* Child */
        /* Real server would set all CGI vars here */
        setenv("QUERY_STRING", cgiargs, 1);
        Dup2(fd, STDOUT_FILENO);          /* Redirect stdout to client */
        Execve(filename, emptylist, environ); /* Run CGI program */
    }
    Wait(NULL); /* Parent waits for and reaps child */
}
```

tiny.c

Serving Dynamic Content with GET

- Notice that only the CGI child process knows the content type and length, so it must generate those headers.

```
    /* Make the response body */
    sprintf(content, "Welcome to add.com: ");
    sprintf(content, "%sTHE Internet addition portal.\r\n<p>", content);
    sprintf(content, "%sThe answer is: %d + %d = %d\r\n<p>",
            content, n1, n2, n1 + n2);
    sprintf(content, "%sThanks for visiting!\r\n", content);

    /* Generate the HTTP response */
    printf("Content-length: %d\r\n", (int)strlen(content));
    printf("Content-type: text/html\r\n\r\n");
    printf("%s", content);
    fflush(stdout);

    exit(0);
```

adder.c

Serving Dynamic Content With GET

```
$ telnet localhost 15213
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
GET /cgi-bin/adder?15213&18213 HTTP/1.0

HTTP/1.0 200 OK
Server: Tiny Web Server
Connection: close
Content-length: 117
Content-type: text/html

Welcome to add.com: THE Internet addition portal.
<p>The answer is: 15213 + 18213 = 33426
<p>Thanks for visiting!
Connection closed by foreign host.
$
```

HTTP request sent by client

HTTP response generated by the server

HTTP response generated by the CGI program

Serving Dynamic Content with POST

- **POST is like GET but provides raw data in addition**
- **Rule of thumb:**
  - GET requests should not change the status of server
  - `http://example.com/?arg=val` can be cached!
  - Use POST requests for side-effects and ever-changing pages
- **CGI scripts receive `?arg=val` as env variable (as usual) POST data on the stdin**

```
    /* Extract the two arguments */
    if (fgets(buf, MAXLINE, stdin) > 0) {
        p = strchr(buf, '&');
        *p = '\0';
        strcpy(arg1, buf);
        strcpy(arg2, p+1);
        n1 = atoi(arg1);
        n2 = atoi(arg2);
    }
```

Serving Dynamic Content with POST

```
$ telnet mc.cdm.depaul.edu 80
Trying 216.220.181.74...
Connected to mc.cdm.depaul.edu.
Escape character is '^]'.
POST /cgi-bin/echo.cgi HTTP/1.1
Host: mc.cdm.depaul.edu
Content-length: 6

Hello!

HTTP/1.1 200 OK
Server: nginx/1.14.1
Date: Wed, 26 Aug 2020 20:14:19 GMT
Content-Type: text/html
Content-length: 46

<html>Echo POST:<br/><pre>Hello!</pre></html>
$
```

HTTP request sent by client

HTTP response generated by the server

HTTP response generated by the CGI program

For More Information

- **W. Richard Stevens et al. "Unix Network Programming: The Sockets Networking API", Volume 1, Third Edition, Prentice Hall, 2003**
  - THE network programming bible.
- **Michael Kerrisk, "The Linux Programming Interface", No Starch Press, 2010**
  - THE Linux programming bible.
- **Newer technologies:**
  - HTTP/2 (2015, discontinued chunked transfer)
  - HTTP/3 (2020, draft stage, no TCP anymore, based on QUIC)
  - Ajax (1999, asynchronous, one-way, *not a protocol*)
  - Websockets (2011, full-duplex, "through" HTTP, but independent)