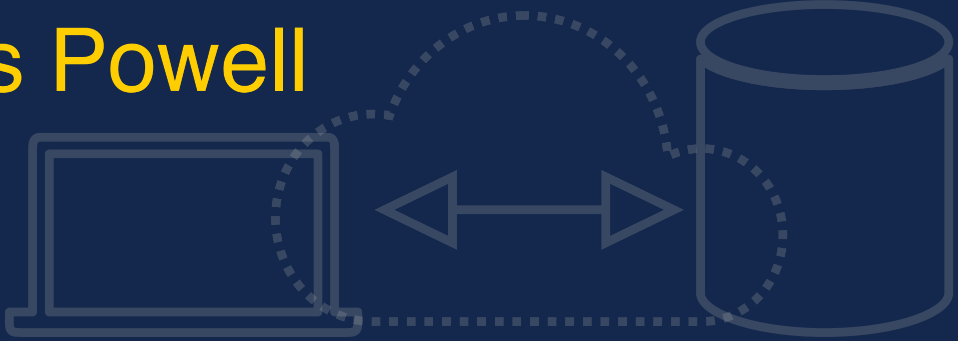
Server-Side Programming Intro

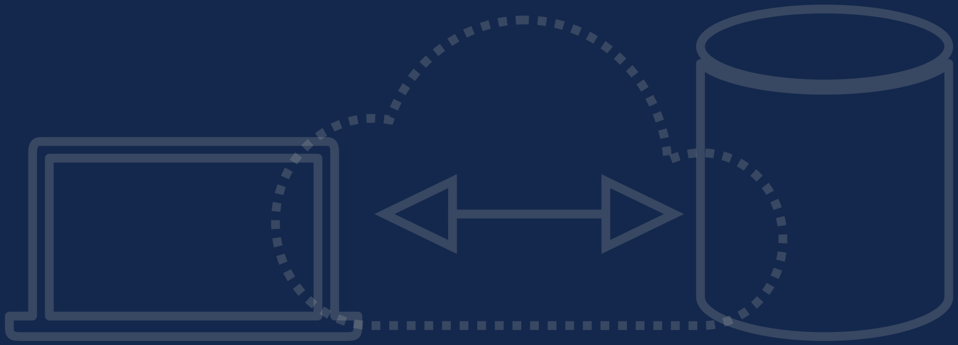
with Thomas Powell



Web Server Roles

- Web Servers as…
  - File Servers (static asset serving, web pages)
  - Application Servers (running code and serving results)

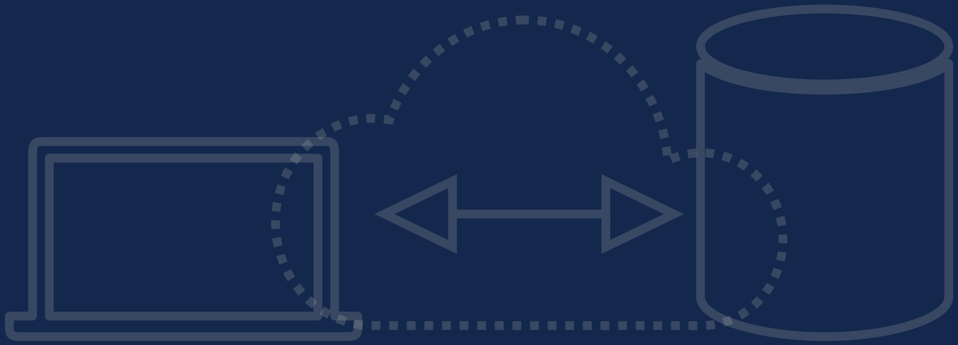
- When web servers are running apps we need to
  - Understand how programs are triggered
    - Route / URL
    - File Extension
  - How data is sent in
    - Query String, Message Body / Payload, Headers
  - How data is returned



Simple Programming Flow

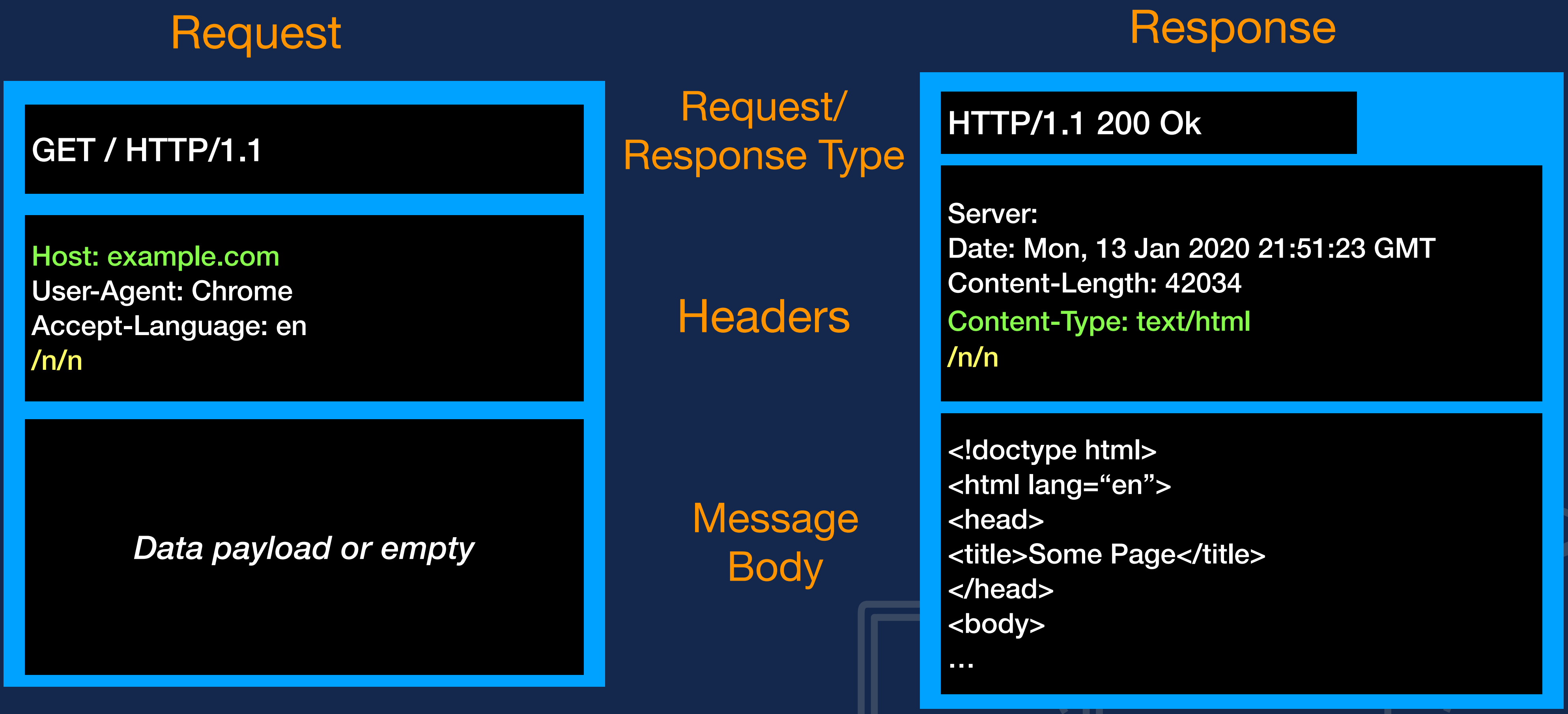
1. Read some data
2. Perform some processing on the data
3. Output some result

- While this is an extremely rudimentary way of viewing a piece of code it actually isn't that far off from the simple way a user request is made and some web page response is given in Web dev.



The Key: The HTTP Law of Three

HTTP packets have three pieces and all that matters are those three pieces



Request

```
GET / HTTP/1.1
```

```
Host: example.com
User-Agent: Chrome
Accept-Language: en
/n/n
```

Data payload or empty

Request/
Response Type

Headers

Message
Body

Response

```
HTTP/1.1 200 Ok
```

```
Server:
Date: Mon, 13 Jan 2020 21:51:23 GMT
Content-Length: 42034
Content-Type: text/html
/n/n
```

```
<!doctype html>
<html lang="en">
<head>
<title>Some Page</title>
</head>
<body>
...
```

Getting Data to the Program

- From last slide, the in-bound HTTP request has data in its query string, headers or message body.

- They query string or message body is likely provided from an HTML form fill out or from JavaScript API that can form an HTTP request (ex: `XMLHttpRequest`, `fetch`, etc.)

- The headers may be populated automatically based upon the browser in use or set via options to `fetch` or a `setRequestHeader()` call on an XHR

The Program

- It can be anything you like…
- In theory we treat the HTTP request as “standard input” and your “standard output” is then piped back to the browser
- Any program in a sense can be a web program as long as it understands what it is reading and writing!

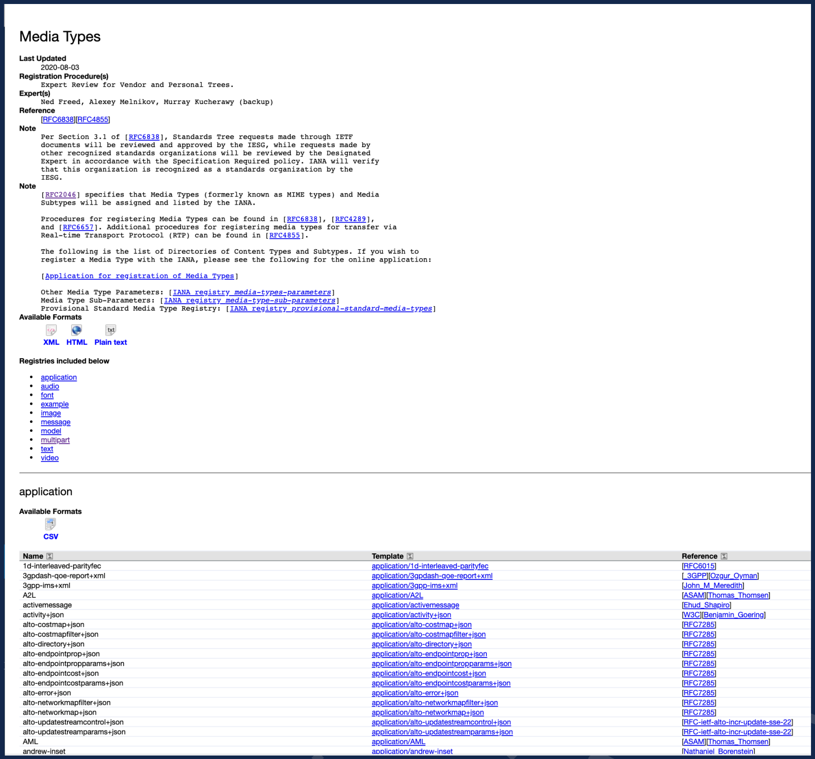
Getting Data Back to the Client

- As long as you know what you are outputting you can send absolutely anything you like back to a browser
- A key header you must understand is `Content-Type` and you would need to set it to the appropriate MIME type value corresponding to the data you are returning.
- Common Web Types might include:
  - text/html, text/css, application/javascript, text/xml, application/json, image/png, etc.
- Note: You are not limited to generated text responses, you could certainly generate an image and send it back to the server assuming you can programmatically do that.

MIME Types Yet Again

- Mime types are broken into a type and subtype. For example, text would be the type and html the submit resulting in **`text/html`**

- An official list can be found at https://www.iana.org/assignments/media-types/media-types.xhtml



Media Types

Last Updated
2020-08-03
Registration Procedure(s)
Expert Review for Vendor and Personal Trees.
Expert(s)
Ned Freed, Alexey Melnikov, Murray Kucherawy (backup)
Reference
[RFC6838][RFC4855]
Note

```
Per Section 3.1 of [RFC6838], Standards Tree requests made through IETF
documents will be reviewed and approved by the IESG, while requests made by
other recognized standards organizations will be reviewed by the Designated
Expert in accordance with the Specification Required policy. IANA will verify
that this organization is recognized as a standards organization by the
IESG.
```

Note

```
[RFC2046] specifies that Media Types (formerly known as MIME types) and Media
Subtypes will be assigned and listed by the IANA.

Procedures for registering Media Types can be found in [RFC6838], [RFC4289],
and [RFC6657]. Additional procedures for registering media types for transfer via
Real-time Transport Protocol (RTP) can be found in [RFC4855].

The following is the list of Directories of Content Types and Subtypes. If you wish to
register a Media Type with the IANA, please see the following for the online application:

[Application for registration of Media Types]

Other Media Type Parameters: [IANA registry media-types-parameters]
Media Type Sub-Parameters: [IANA registry media-type-sub-parameters]
Provisional Standard Media Type Registry: [IANA registry provisional-standard-media-types]
```

Available Formats

Registries included below

- application
- audio
- font
- example
- image
- message
- model
- multipart
- text
- video

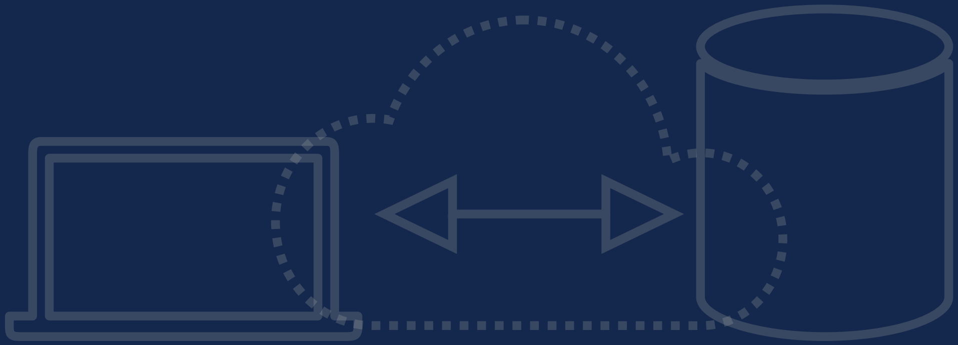
application

Available Formats

Name	Template	Reference
1d-interleaved-parityfec	application/1d-interleaved-parityfec	[RFC6015]
3gpdash-qoe-report+xml	application/3gpdash-qoe-report+xml	[_3GPP][Ozgur_Oyman]
3gpp-ims+xml	application/3gpp-ims+xml	[John_M_Meredith]
A2L	application/A2L	[ASAM][Thomas_Thomsen]
activemessage	application/activemessage	[Ehud_Shapiro]
activity+json	application/activity+json	[W3C][Benjamin_Goering]
alto-costmap+json	application/alto-costmap+json	[RFC7285]
alto-costmapfilter+json	application/alto-costmapfilter+json	[RFC7285]
alto-directory+json	application/alto-directory+json	[RFC7285]
alto-endpointprop+json	application/alto-endpointprop+json	[RFC7285]
alto-endpointpropparams+json	application/alto-endpointpropparams+json	[RFC7285]
alto-endpointcost+json	application/alto-endpointcost+json	[RFC7285]
alto-endpointcostparams+json	application/alto-endpointcostparams+json	[RFC7285]
alto-error+json	application/alto-error+json	[RFC7285]
alto-networkmapfilter+json	application/alto-networkmapfilter+json	[RFC7285]
alto-networkmap+json	application/alto-networkmap+json	[RFC7285]
alto-updatestreamcontrol+json	application/alto-updatestreamcontrol+json	[RFC-ietf-alto-incr-update-sse-22]
alto-updatestreamparams+json	application/alto-updatestreamparams+json	[RFC-ietf-alto-incr-update-sse-22]
AML	application/AML	[ASAM][Thomas_Thomsen]
andrew-inset	application/andrew-inset	[Nathaniel_Borenstein]

But…. State Preview

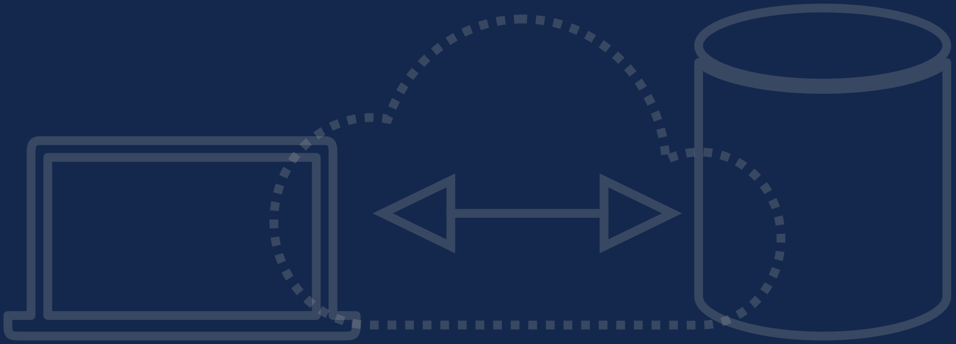
- HTTP is stateless so we won’t have any memory from page to page. To address this we need to save information some how
- Two approaches
  - Save the data itself
  - Save a “dereference to the data” an ID or “pointer” to saved data
- Three places
  - Query String
  - Message Body
  - Headers (Cookies!)
- Deck and demos later!



Help me do what I want

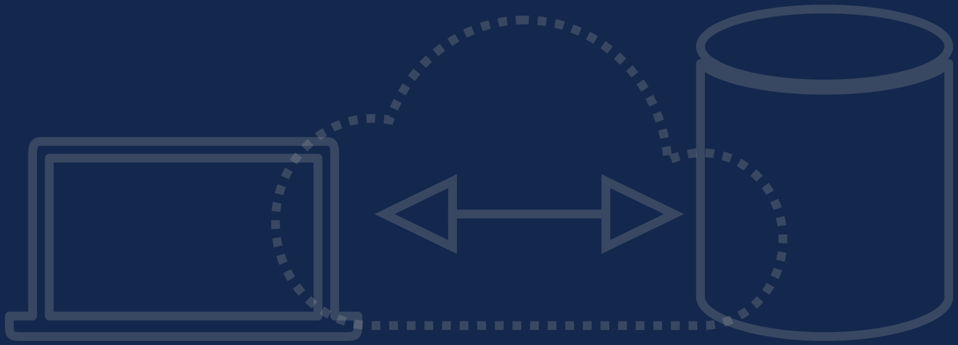
- I don’t want to print a bunch of HTML strings
- I don’t want to deal with data (saving and retrieving)
  - Databases!
- I don’t want to handle sessionization to solve for state
- I don’t want to do …

- Frameworks hopefully will help us be more productive, we will use them for sure once we understand what our problems are



To make a Web server based “program”

- You have to get data in (from user-agent to server)
- Then process the data, perform some task, etc.
- You have get data out (from server to user-agent)
  - Input: <form> data, URLs, HTTP headers (browser type, IP, cookie, etc.)
  - Output: Some form of data usually HTTP, GIF, JPEG, etc. and the appropriate header (MIME type ,cookie, etc.)
- You will need to fix the stateless nature of HTTP to do anything meaningful
  - Sessionization via cookies, URLs, hidden fields, etc.
- You should provide features to make programming easy not only do the previous things simply but allow for access to DBs, output HTML and Web site widgets, etc.



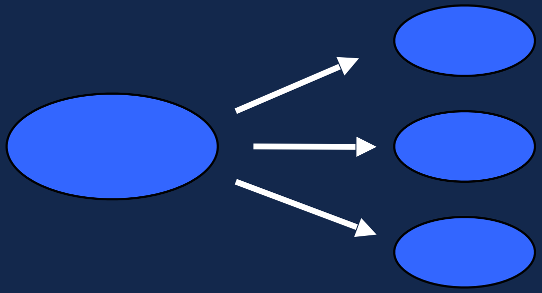
The “Hamburger” Model

- A funny way you might think of this is that your “program” – the thing you do is the “meat” of your hamburger and the buns are the input and the output
  - Input - ”Top Bun” (Msg body, query string, environment variables)
  - The Program – “Meat” (what does the work takes the input and does some calc, fetches data, etc.)
  - Output – “Bottom Bun” (The response + headers sent back)

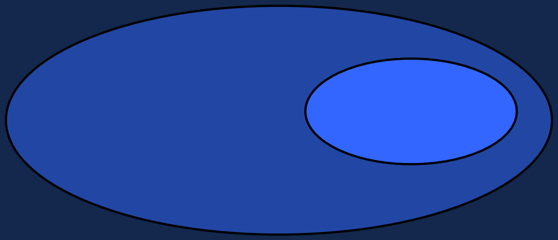
Redux -3 Server-Side Programming Models

- Revisit this again because it is that important – there are just 3 models at play here really
    - #1) Classic CGI model – “fork and exec”
        - Web server creates new child process, passing it request data as environment variables
        - CGI script issues response using standard I/O stream mechanisms
    - #2) Server API model
        - Web server runs additional request handling code inside its own process space
    - #3) Web application frameworks
        - Web server calls API application, which may manage request within its own pool of resources and using its native objects

3 Server-Side Programming Models

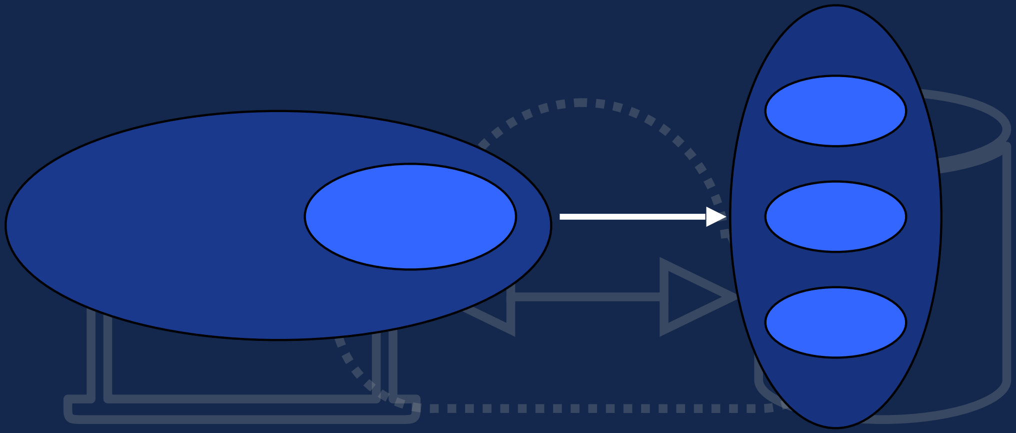


Classic CGI “fork and exec”



Server API running inside
Web server’s address space

Web application framework running
inside Web server process but managing
its own pool of resources via IPC



3 Server-Side Programming Models

- Each model has its pros and cons
    - Classic CGI model
        - Pro: isolation means easiest in principle to secure, least damaging if something goes wrong
        - Con: isolation makes it slow & resource intensive
    - Server API model
        - Pro: very fast & low overhead if written properly
        - Con: hard to write; blows up server if done wrong
    - Web application frameworks
        - Pro: ideally combines efficiency of API model with safety of CGI; adds helpful encapsulation of routine tasks like state management
        - Con: built-in tools can be resource hogs in wrong hands; ease of use may encourage carelessness

3 Server-Side Programming Models

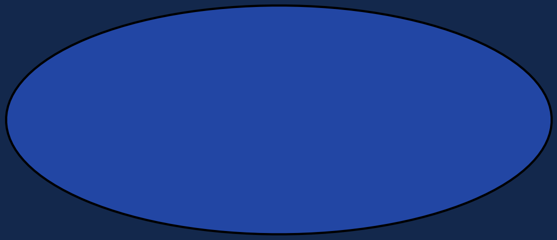
- Many examples of each
    - Classic CGI
        - Scripts written in Perl
        - Programs written in C
    - Server API
        - Apache modules
        - ISAPI filters and extensions
    - Web application frameworks
        - All descended from Server Side Includes (SSI), original “parsed HTML” solution that allowed interspersing of executable code with markup
        - ASP, ASP.NET, Cold Fusion, JSP/Servlets, Python, PHP, etc.

Theoretical Trade-offs

- As we saw there are many approaches to accomplishing Web programming
    - No one approach works for all situations
- Speed of ISAPI vs. simplicity of a PHP script
    - Some show you low-level details like headers, query strings, etc. very explicitly and some do not
    - Some are harder to work with than other
    - Trade-off: Configurability vs. Simplicity
    - We also see that some higher level frameworks will trade-off ease of programmer effort for overhead and thus speed or scale
    - Portability or reliance to a particular platform is also seen
- We start with low level old style CGI to see the underlying sameness of all server-side Web coding

Model 4?

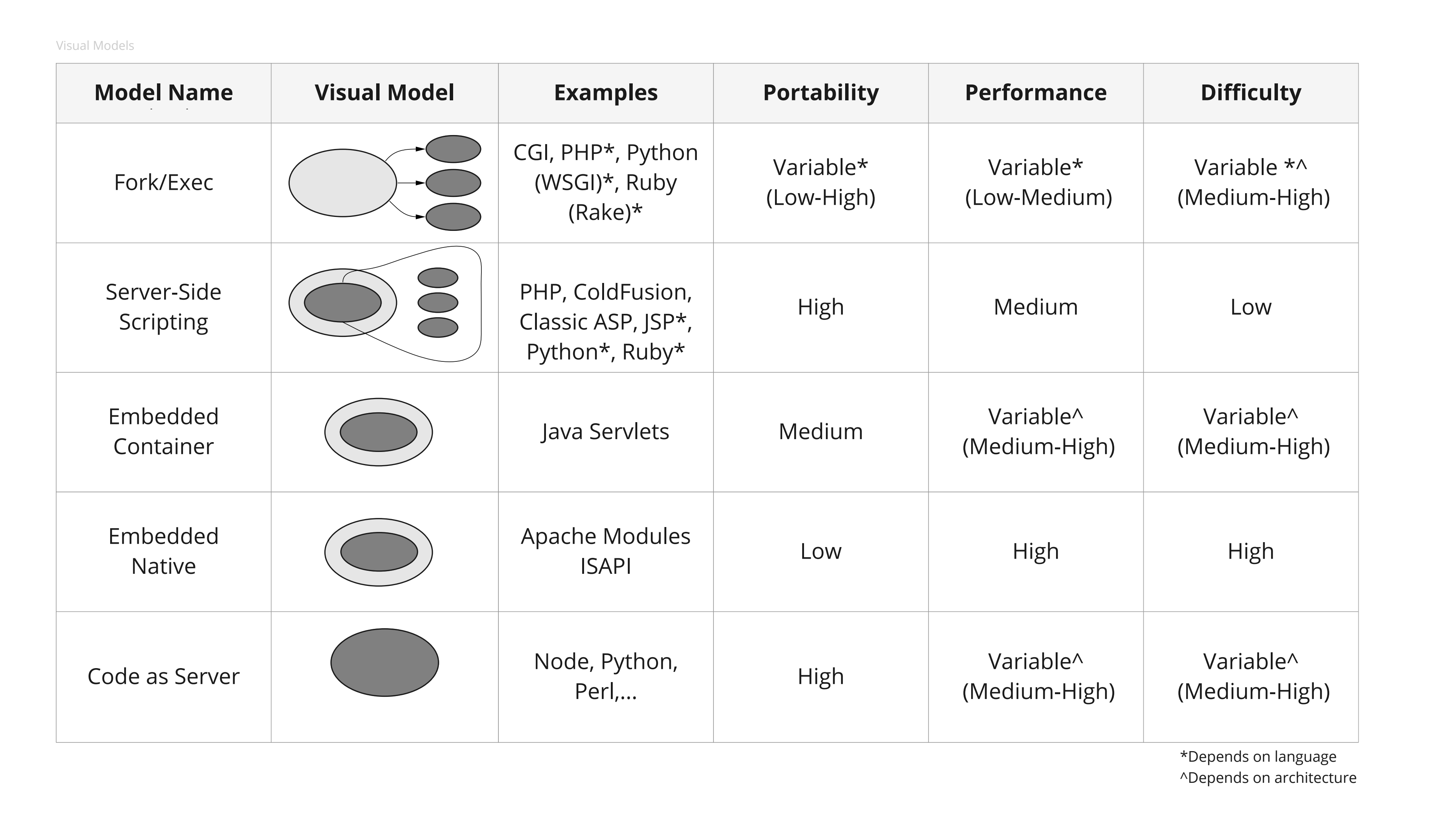
The Web Server is the app / the app is the Web server?



Common in the NodeJS world our code defines routes and the app is the server. This is both a good idea (limited surface area) and an insane idea (you have no idea what serving HTTP really means)
Suggestion; Use Node behind a HTTP proxy or in a co-server model

Interestingly this is an old model that was abandoned and is now back

Server Side Model Summary



Visual Models

Model Name	Visual Model	Examples	Portability	Performance	Difficulty
Fork/Exec		CGI, PHP*, Python (WSGI)*, Ruby (Rake)*	Variable* (Low-High)	Variable* (Low-Medium)	Variable *^ (Medium-High)
Server-Side Scripting		PHP, ColdFusion, Classic ASP, JSP*, Python*, Ruby*	High	Medium	Low
Embedded Container		Java Servlets	Medium	Variable^ (Medium-High)	Variable^ (Medium-High)
Embedded Native		Apache Modules ISAPI	Low	High	High
Code as Server		Node, Python, Perl,...	High	Variable^ (Medium-High)	Variable^ (Medium-High)

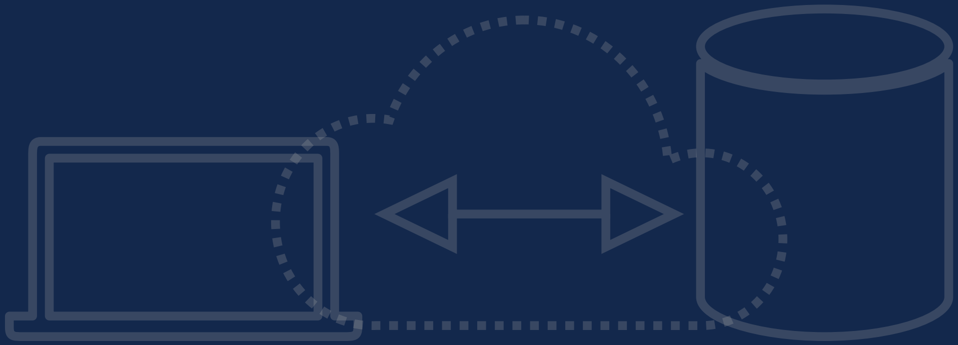
*Depends on language
^Depends on architecture

A Choice

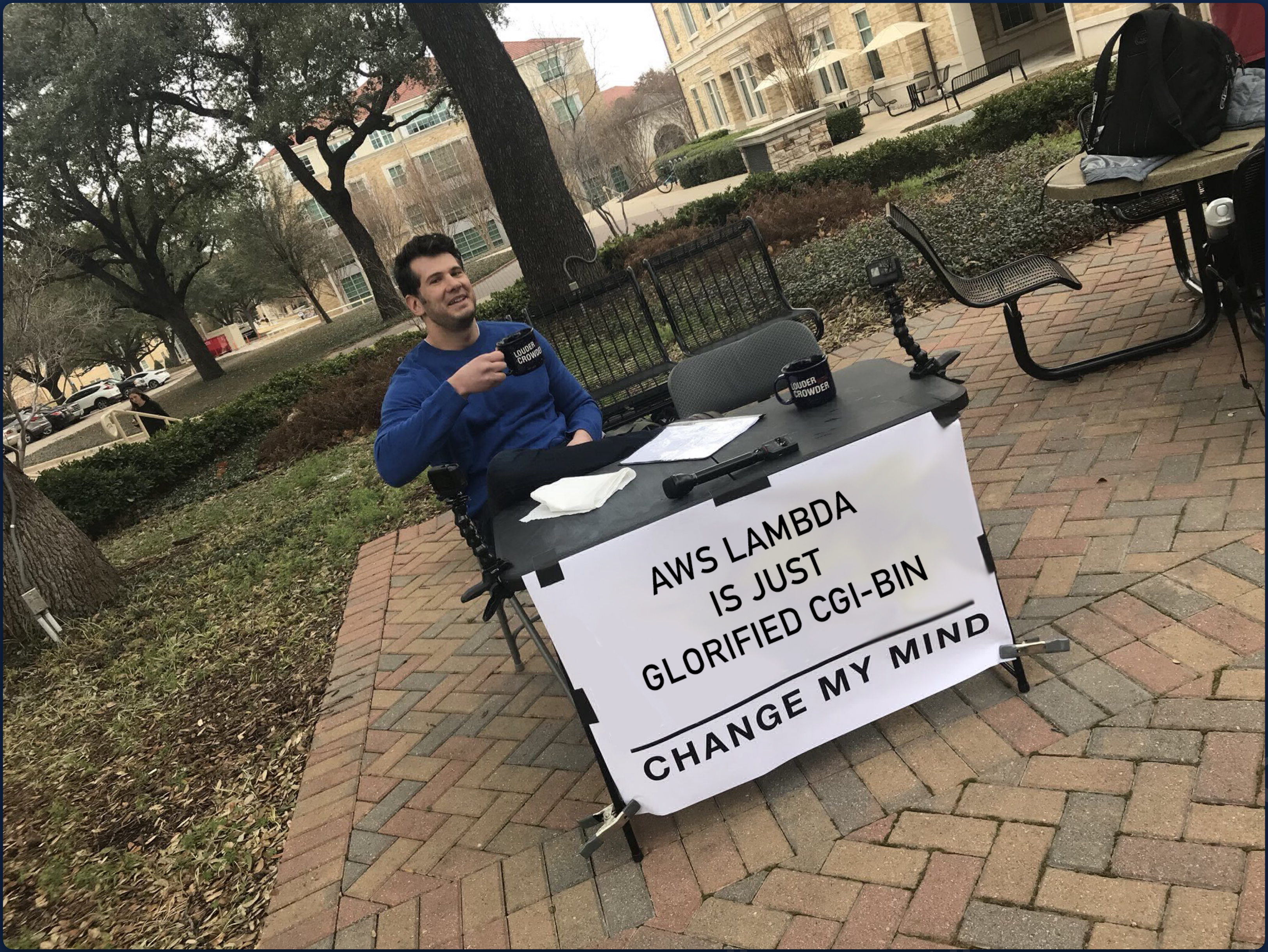
So it boils down to is my code being served and run by something else speaking HTTP or is my code going to be responsible to be an HTTP server?

Big Worry: When it comes to HTTP you often don't know what you don't know until you know it (usually after a fail)

Mastering the Basics
Revisiting Old Time CGI



IL TRIVELLA
CGI
the grill h
old school
like a boss!


LOUDER CROWDER
LOUDER CROWDER
AWS LAMBDA IS JUST GLORIFIED CGI-BIN
CHANGE MY MIND

CGI (Common Gateway Interface)

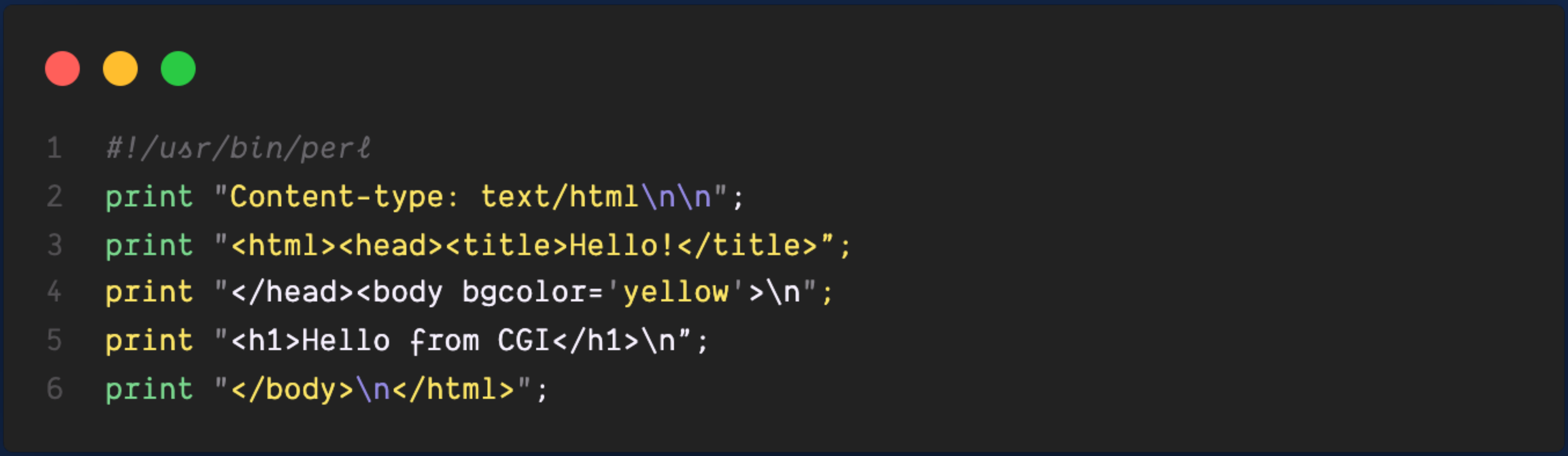
- Simple standard defining the way for external programs to be run on Web servers
- In short CGI defines the way that data is read in by external programs and what is expected to be returned
    - Prof's "Hamburger" model, others have some "onion model"

- Useful CGI related resources
    - https://en.wikipedia.org/wiki/Common_Gateway_Interface
    - http://www.cgi-resources.com
    - http://www.w3.org/CGI/
    - O'Reilly's CGI Book - http://www.oreilly.com/catalog/cgi2/

CGI and Perl

- CGI is often mentioned in conjunction with Perl, this is somewhat misleading
- CGI does not specify the usage of a particular language.
    - If this is the case why Perl?
    - Perl was commonly found on UNIX machines which were the first Web servers
    - Perl was commonly known to sysadmins at the time running said web servers and tasked as the first webmasters by the
    - Perl is very good at string manipulation which is one of the main tasks when writing a web program
    - Perl is pretty easy to hack around with
    - Of course as an interpreted language you can see Perl partial to blame for CGI's speed problems
        - Solution: Use mod_perl
        - Best Solution: Recompile as a C program

First Example in Perl



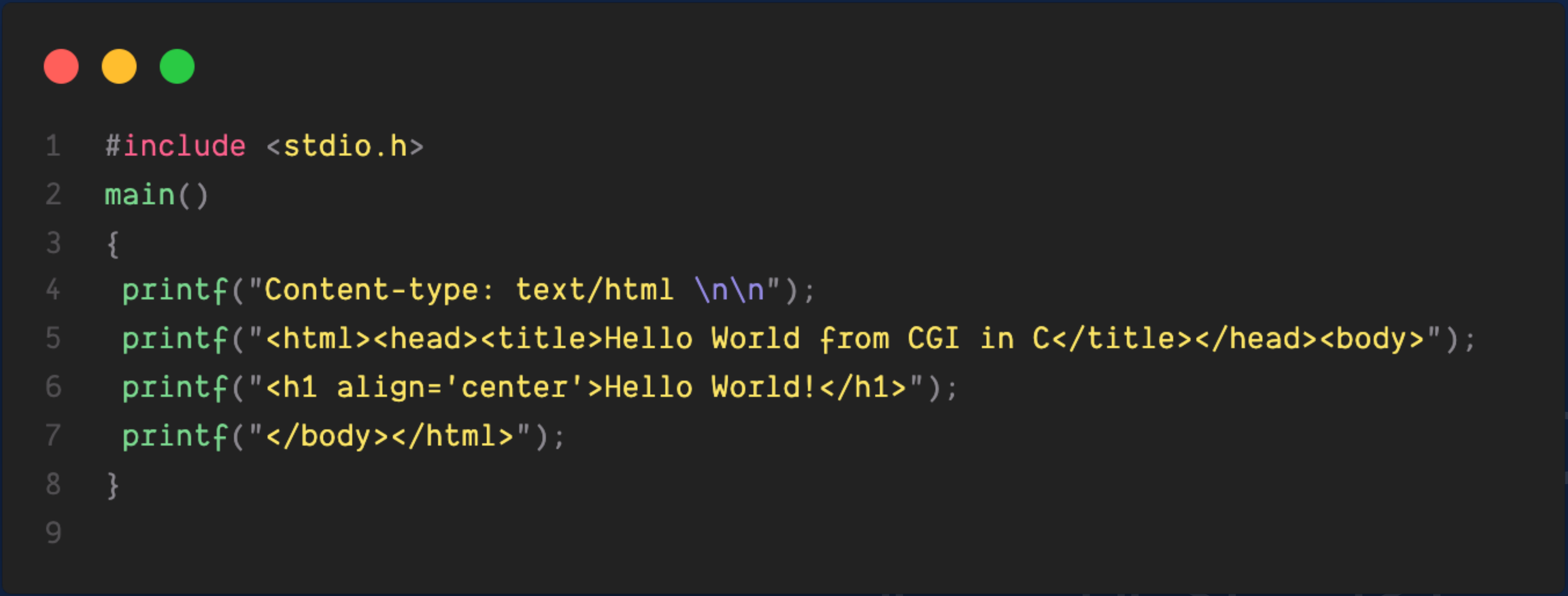
```perl
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "<html><head><title>Hello!</title>”;
print "</head><body bgcolor='yellow'>\n";
print "<h1>Hello from CGI</h1>\n”;
print "</body>\n</html>";
```

- Notice the Content-type: header. That’s 50% of CGI’s “magic”
- In this case we use a .pl extension though you might want to use .cgi or even better something else or even nothing.
- We also probably place the code in the cgi-bin directory
  - Good practice in some ways, but again change the name
-

Note: We probably should use the #!/usr/bin/perl -wT to invoke Perl. The w flag shows warnings and the T flag turns on taint checking which checks incoming data a little more carefully.

First Example in C

- As said before language is irrelevant in CGI, so we recode helloworld in C as a demo



```c
#include <stdio.h>
main()
{
 printf("Content-type: text/html \n\n");
 printf("<html><head><title>Hello World from CGI in C</title></head><body>");
 printf("<h1 align='center'>Hello World!</h1>");
 printf("</body></html>");
}

```

Don’t Reinvent the Wheel

- There isn’t much to do in CGI (nor really any server-side enviro)
  - Main tasks read in user submitted data and write out HTML and headers
  - The real programming is outside of this input/output!

- Plenty of room to screw-up
  - Assuming that what is sent in is correct or safe
  - Not handling error conditions or exposing sensitive mechanisms.

- Rather than reinventing the wheel you could utilize a library

Input/Output with CGI

- There are two forms of input to CGI programs
  - User supplied data
  - Environment supplied data
- User data generally comes from form fill-outs, clicks, etc. and is passed either using the GET or POST method while environment data is related to the environment in which the program is run and is mostly related to the various HTTP headers passed by the invoking user agent in combination with some local server variables.
- As shown by the previous example, output from CGI is a Content-type: header with the appropriate type (often text/html) followed by the specified content (often HTML)

Environment Variables

- The environment under which the request is served and contains three types of values:
    1. Server or program related information
    2. Submitted data from the client
    3. HTTP related headers

2. The next slides show a selection of these values

- Note: In Perl, environment variables are stored in a hash named %ENV so we can easily access them (see example)
- Note: ALL environments will have some way to reference this type of data it is a question of how.

Select Environment Variables

Variable Name	Description	Example
SERVER_SOFTWARE	Name and version of software running request	Apache/2.2.21
SERVER_NAME	Name of server handling the request may be an IP.	www.example.com
SERVER_PORT	Hostname of server which may be an IP address	8080
REQUEST_METHOD	Indicates the CGI Version used. Likely of little use for you.	POST
PATH_TRANSLATED	Contains the full path to the program from the server point of view	/var/www/htdocs/cgi-bin/mypgoram.pl
SCRIPT_NAME	Contains the path to the executing program	/cgi-bin/myprogram.pl
QUERY_STRING	Contains the part after the ? In a URL (aka the query string)	username=tpowell&role=prof&year=2020
REMOTE_HOST	Contains the clients fully qualified name if available	someclient.somedomain.com
REMOTE_ADDR	Contains the IP address of the requester	200.102.121.9
CONTENT_TYPE	Contains the content type of any sent data	application/x-www-form-urlencoded
CONTENT_LENGTH	The length of the data sent in bytes	9004

The HTTP Environment Variables

- All the HTTP headers are put into the environment for us to access. They will be prefixed with HTTP_
- Given the unlimited range of headers they are mapped with - to _ for example **`X-Powered-By`** becomes **`HTTP_X_POWERED_BY`**

Variable Name	Description	Example
HTTP_ACCEPT	The contents of the Accept header in the request	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*.*
HTTP_USER_AGENT	The contents of the User-Agent header in the request	Mozilla/5.0 (Macintosh; Intel;) Chrome 85.0 4143.105
HTTP_COOKIE	Value of the Cookie header in the request	Session_ID=324dasaca3
HTTP_X_WHAT_HEADER	An extension header showing that in effect the pattern allows for any arbitrary headers	Some arbitrary header value found in X-What-Header:

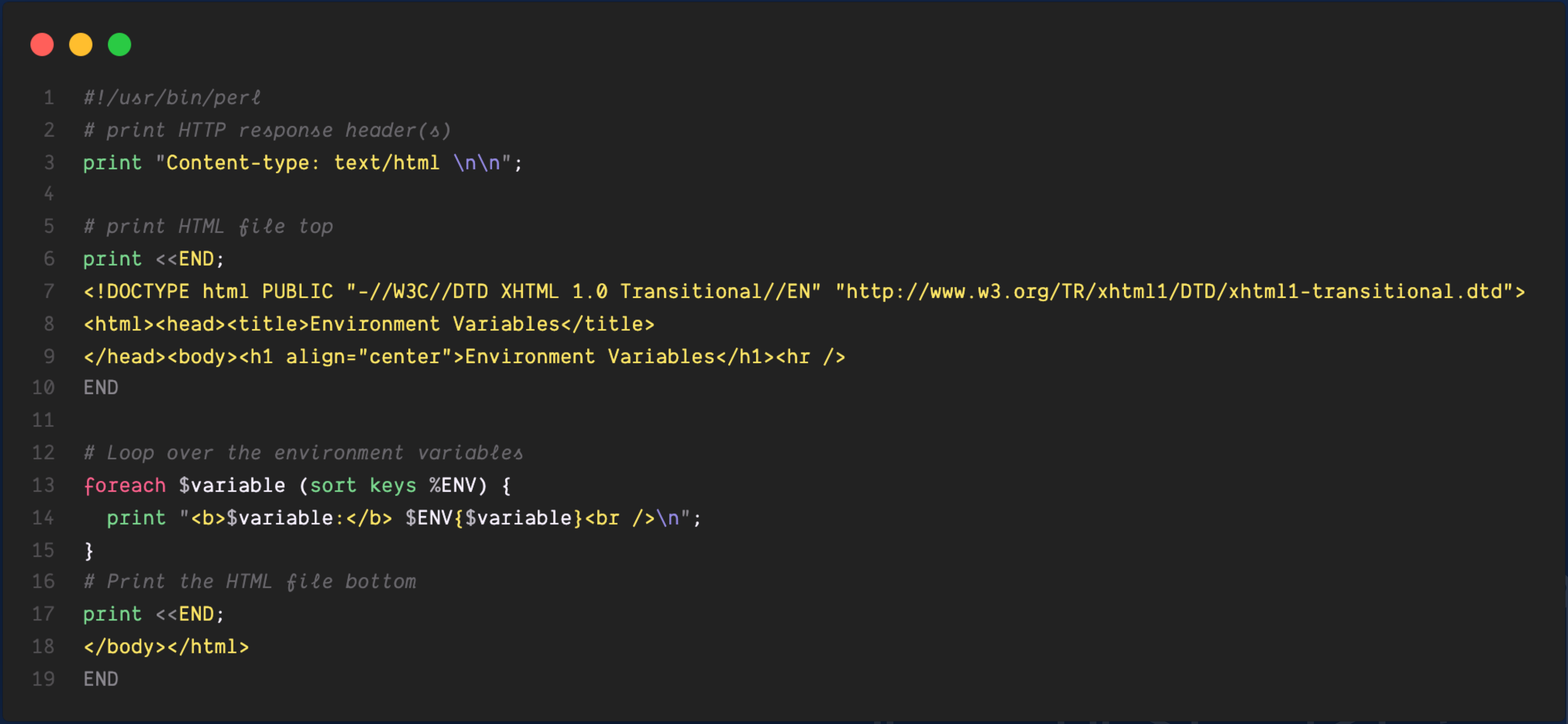
Accessing Environment Variables

- In Perl, environment variables are stored in a hash named %ENV so we can easily access them (see next example)

- ALL server-side programming environments will have some way to reference this type of data from an incoming request and state of server it is a question of how.
    - Example: PHP - getallheaders(), $_SERVER, etc. https://www.php.net/manual/en/reserved.variables.server.php

- You should note there is an underlying sameness, just the specifics of how the items are read and written is what changes in various environments

Environment Variables Example without CGI.pm



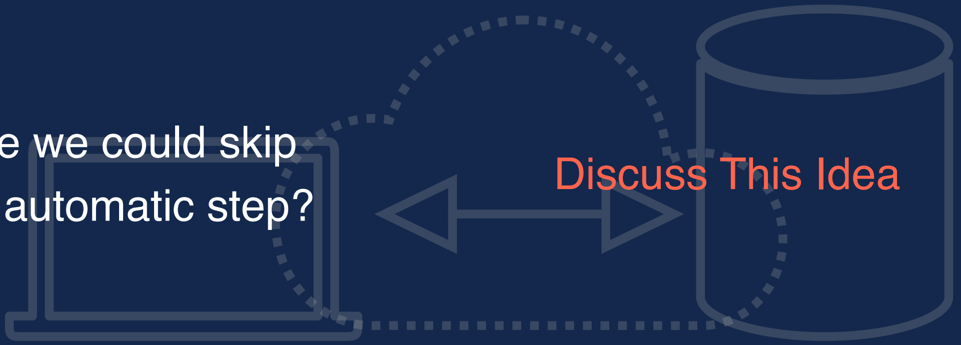
```perl
#!/usr/bin/perl
# print HTTP response header(s)
print "Content-type: text/html \n\n";

# print HTML file top
print <<END;
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><head><title>Environment Variables</title>
</head><body><h1 align="center">Environment Variables</h1><hr />
END

# Loop over the environment variables
foreach $variable (sort keys %ENV) {
  print "<b>$variable:</b> $ENV{$variable}<br />\n";
}
# Print the HTML file bottom
print <<END;
</body></html>
END
```

Manual and Automatic

- The problem we often run into with development is that after a while it becomes tedious to do a bunch of low level stuff to accomplish a pretty basic task like reading a URL query string
  - Tip: After you feel that hate, it’s time to automate!

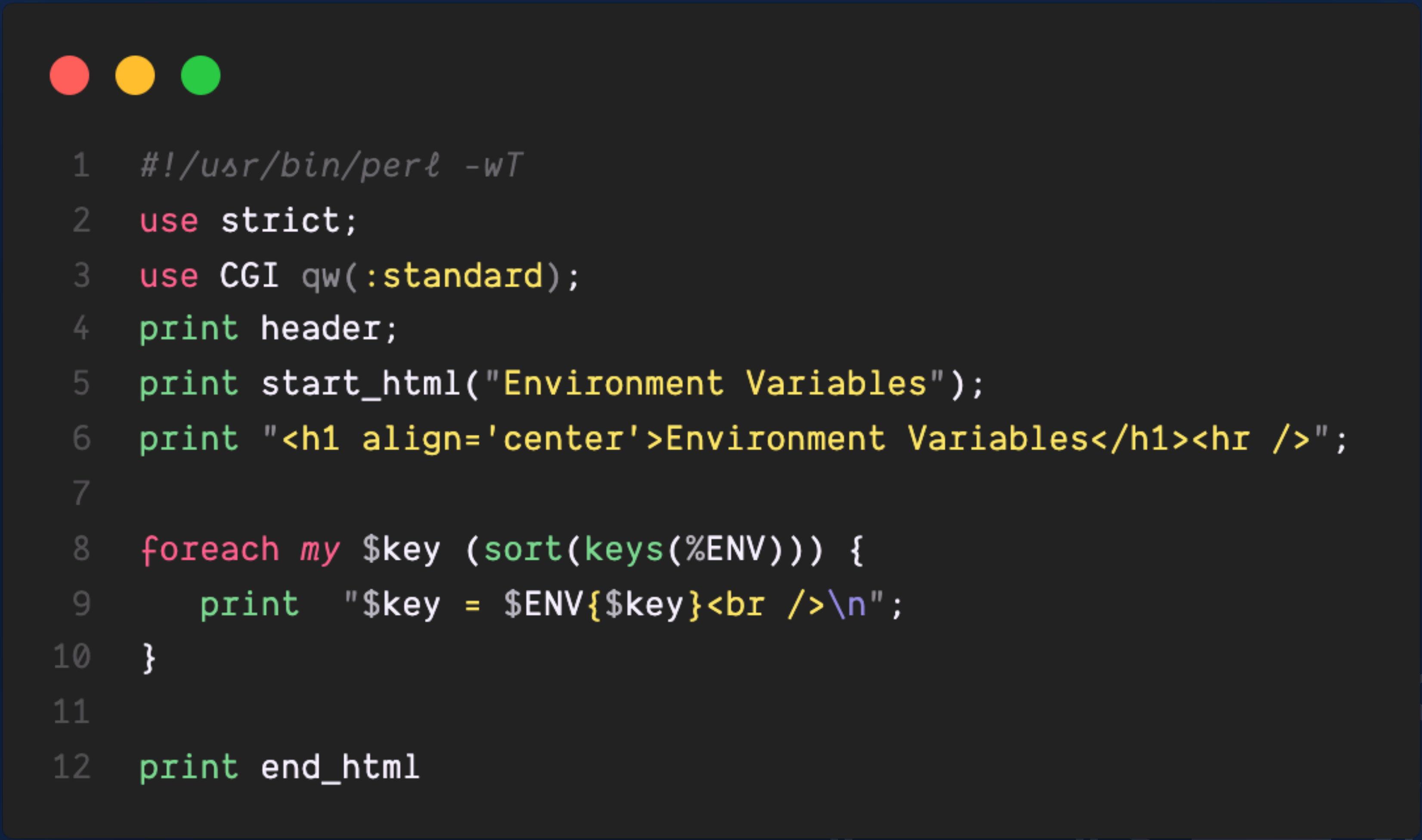
- At this point we understand what we are doing and we probably need to reach to a library to make the task easier. Even in the CGI days such a need existed (see next demo)

If we had some quality certainty maybe we could skip this evaluate and learn before we use automatic step?



Discuss This Idea

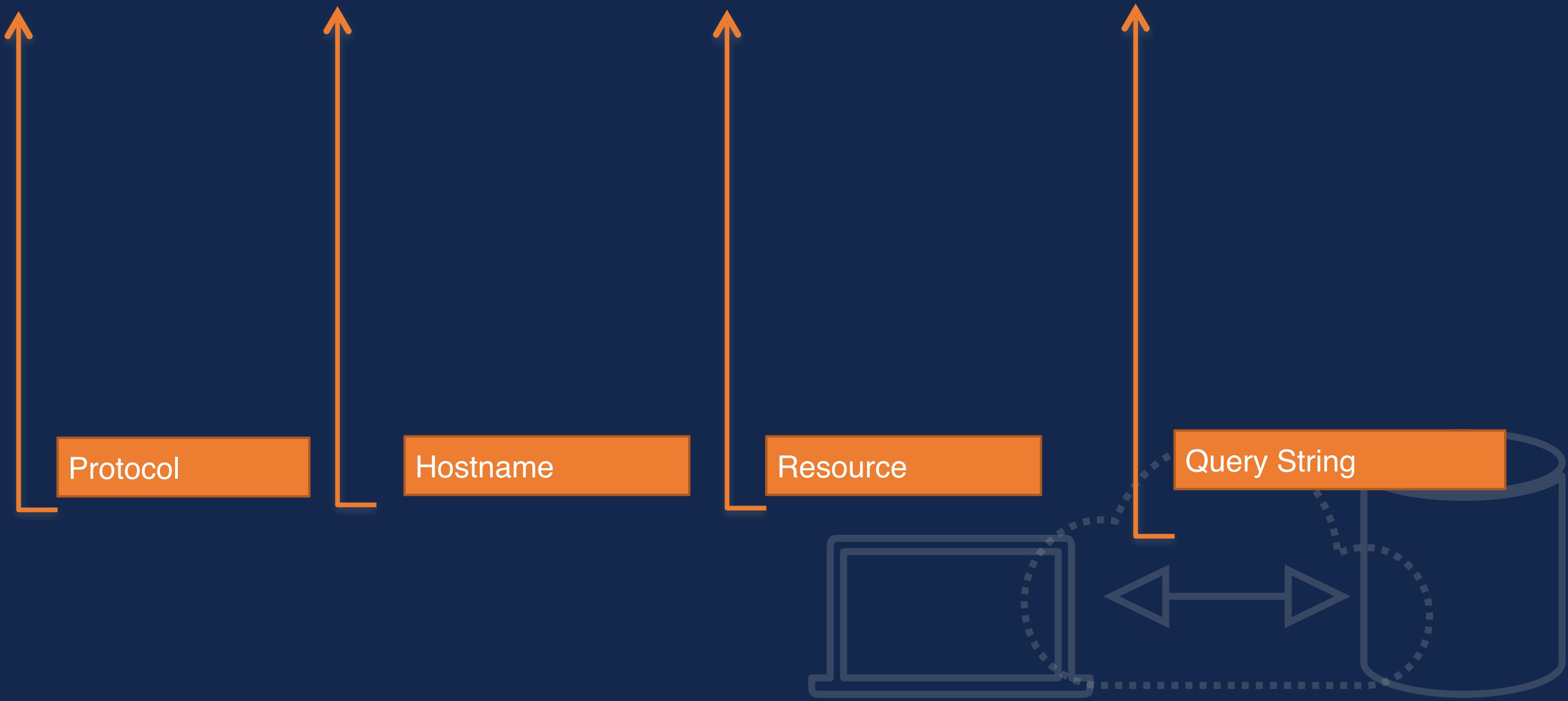
Environment Variables Example with CGI.pm



```perl
#!/usr/bin/perl -wT
use strict;
use CGI qw(:standard);
print header;
print start_html("Environment Variables");
print "<h1 align='center'>Environment Variables</h1><hr />";

foreach my $key (sort(keys(%ENV))) {
    print  "$key = $ENV{$key}<br />\n";
}

print end_html
```

Back to the URL

`http://www.example.com/catalog.php?dept=gadgets&product=7`



Protocol

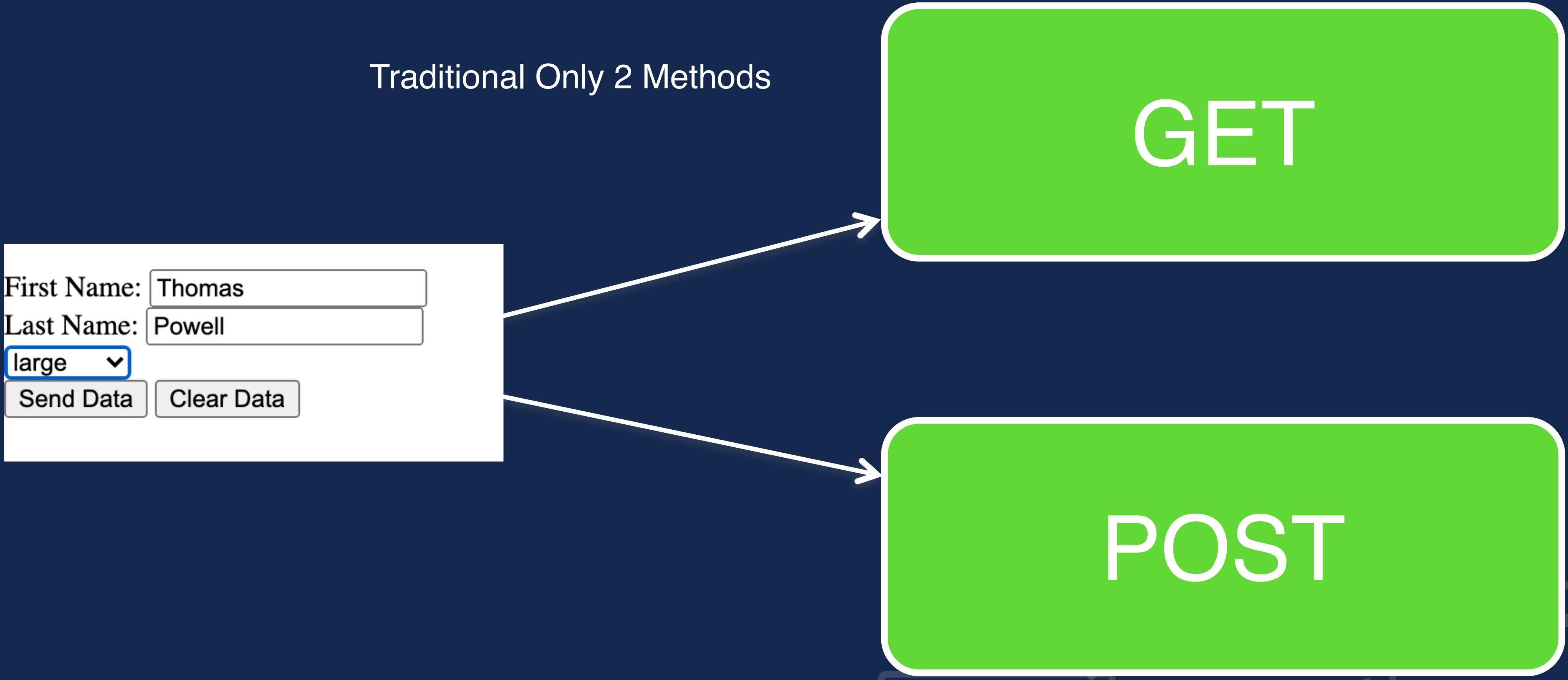
Hostname

Resource

Query String

HTML Forms

Traditional Only 2 Methods



First Name: Thomas

Last Name: Powell

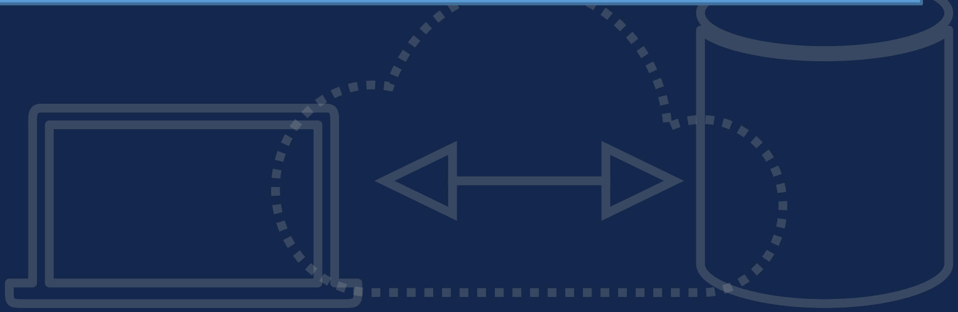
large

GET

POST

HTTP GET Data Example

```
GET /doathing.php?login=thomas&password=ZomgInClear HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Chrome 121)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate, br, super-krazy-compress
Referer: http://example.com/simplehttp
..
```



And the Response

```
GET /doathing.php?login=thomas&password=ZomgInClear HTTP/1.1
Host: example.com
User-Agent: Chrome 121
......
```

```
HTTP/1.1 200 OK
Date: Mon, 30 Jul 2029 01:25:48 GMT
Server: Apache/9.67 (Unix) PHP/15.2.5 mod_ssl/2.8.30 OpenSSL/2.9.8g
X-Powered-By: PHP/15.2.5
X-Meme: It’s Over 9000!
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

..... HTML data follows .....
```

HTTP POST Data Pass

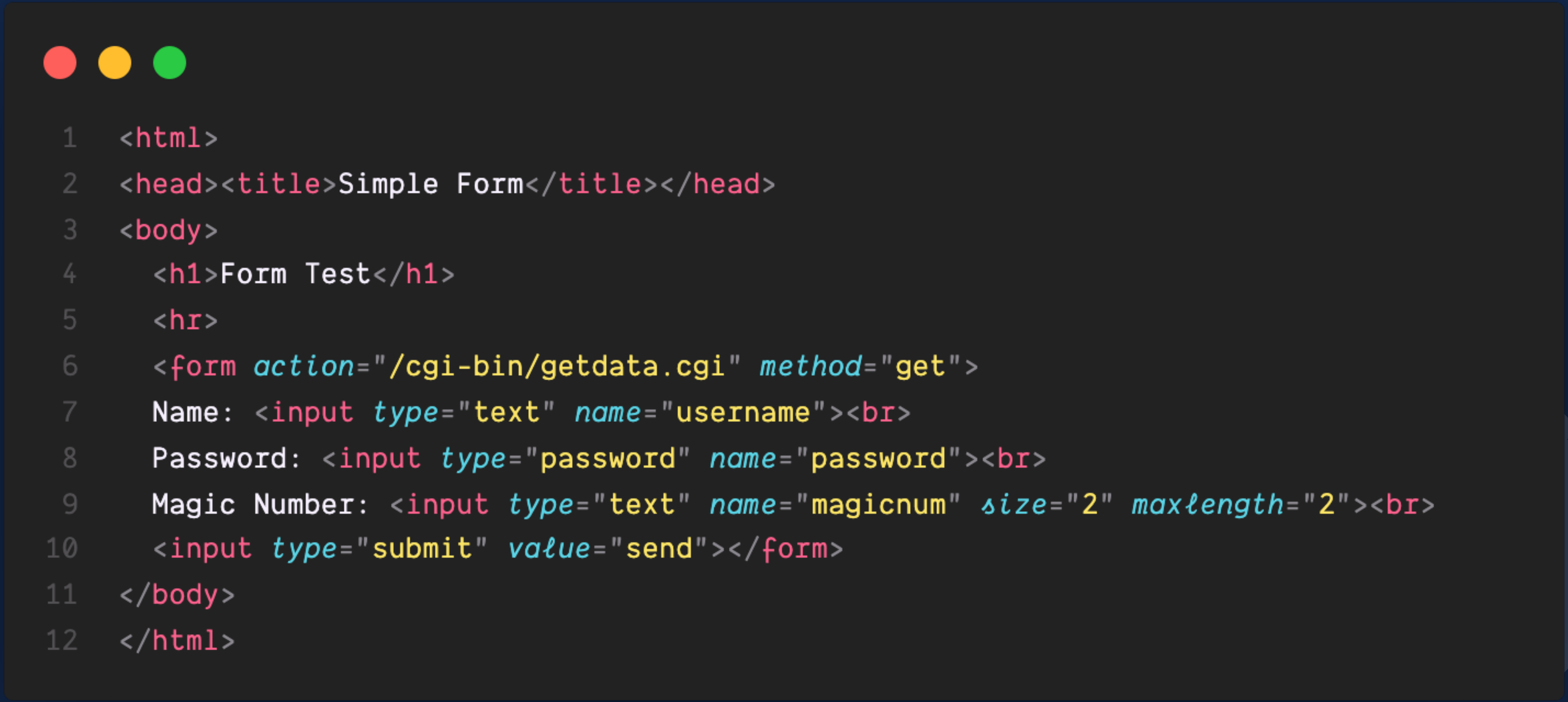
```
POST /doathing.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Chrome 121)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate, br, super-krazy-compress
Referer: http://example.com/simplehttp
Content-Type: application/x-www-form-urlencoded
Content-Length: 31

login=tpowell&password=ZomgInClear
```

Response was same so omitted

Reading Data from Users

- Libraries (ex. CGI.pm)make it easy to read data in, they put it in a hash for us (no matter the method) by name/value pairs.

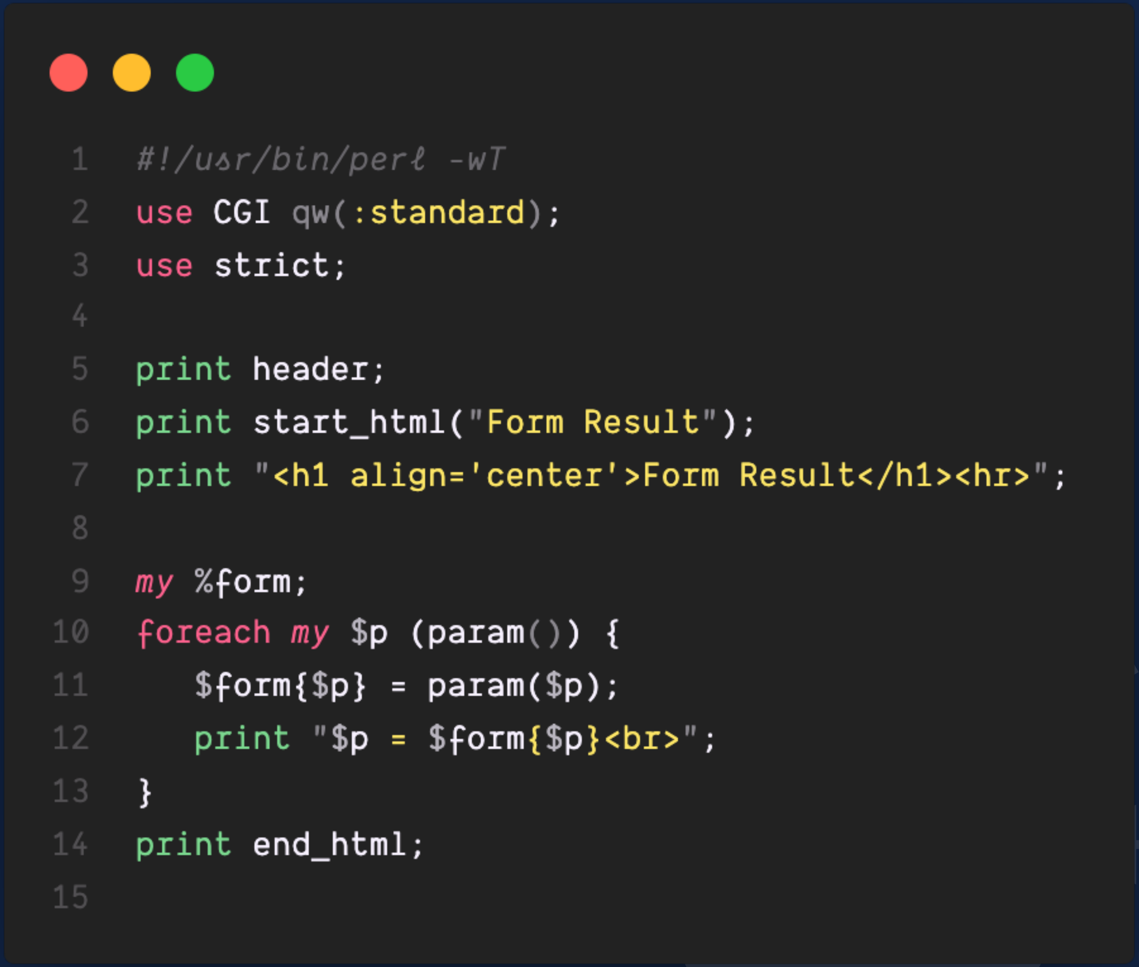
- We’ll see in scripting languages like PHP it gets even easier!
- So given



```html
<html>
<head><title>Simple Form</title></head>
<body>
  <h1>Form Test</h1>
  <hr>
  <form action="/cgi-bin/getdata.cgi" method="get">
  Name: <input type="text" name="username"><br>
  Password: <input type="password" name="password"><br>
  Magic Number: <input type="text" name="magicnum" size="2" maxlength="2"><br>
  <input type="submit" value="send"></form>
</body>
</html>
```

Reading Data from Users Contd.

- We parse the data like so



```perl
#!/usr/bin/perl -wT
use CGI qw(:standard);
use strict;

print header;
print start_html("Form Result");
print "<h1 align='center'>Form Result</h1><hr>";

my %form;
foreach my $p (param()) {
    $form{$p} = param($p);
    print "$p = $form{$p}<br>";
}
print end_html;

```

Reading Data Tips

- **`PRO TIP #1:`** It is easier to think about what you need to do and then figure out the syntax needed. If you know you want the query string, message body or headers it is now just a matter of looking up syntax.

- **`PRO TIP #2:`** Query for what you know and expect. Never just trust a whole packet submitted.

- **`PRO TIP #3:`** If you receive unexpected data or data type - just reject it!

CGI Troubles

- Since we are on the topic of security, why do people think CGI is insecure?
  - Often CGI programs run at higher permissions than they should be
  - Sometimes people use the shell
    - imagine form data being executed at a shell!
  - Again devs trust input way too much
    - Cross site scripting, code injection, etc.
    - This isn’t unique to CGI, but the architecture particularly if they use shell scripts seems to make things worse

CGI Troubles

- Speed Problems
  - Every run of a CGI program server sets up environment, loads script/program, and runs it and then rips it down when done
  - Multiple users will equal many CGIs running
  - Often programmed in an interpreted language like Perl.
- Solutions
  - Keep the script loaded in memory and running as a co-process (e.g. FastCGI - www.fastcgi.com)
  - Use an embedded script interpreter (mod_perl)
  - Move to a server API program (e.g. ISAPI extension or Apache module)

CGI Troubles

- Problem with apparent complexity
    - A lot of details are still exposed to the programmer
    - Using a library much can be hidden but you still have to understand headers, environment variables, etc. more than you might see in other environments
    - Some areas like session management are not abstracted as well in CGI programming as in more modern frameworks
    - Solution: Move to a scripting environment like PHP that hides these details better - but is their a trade-off there?
- Yet despite being very archaic CGI it is still heavily used by certain types of Web developers and it is useful as it demonstrates the details of server-side programming which is always there regardless of framework in play – dress it up, call it what you want but it all comes down to the same inputs, outputs, and network characteristics – better to know that than not!

Summary

- CGI demonstrates the basics of ALL server-side dev environments
  - Code Trigger (path)
  - Reading data from requests
    - Query String, Headers, Message Body
  - Returning results
    - Headers (esp. Content-Type) and content (esp. HTML)
- Already we see the ease provided, but we should never focus too much on syntax as all the systems we will explore will be roughly equivalent