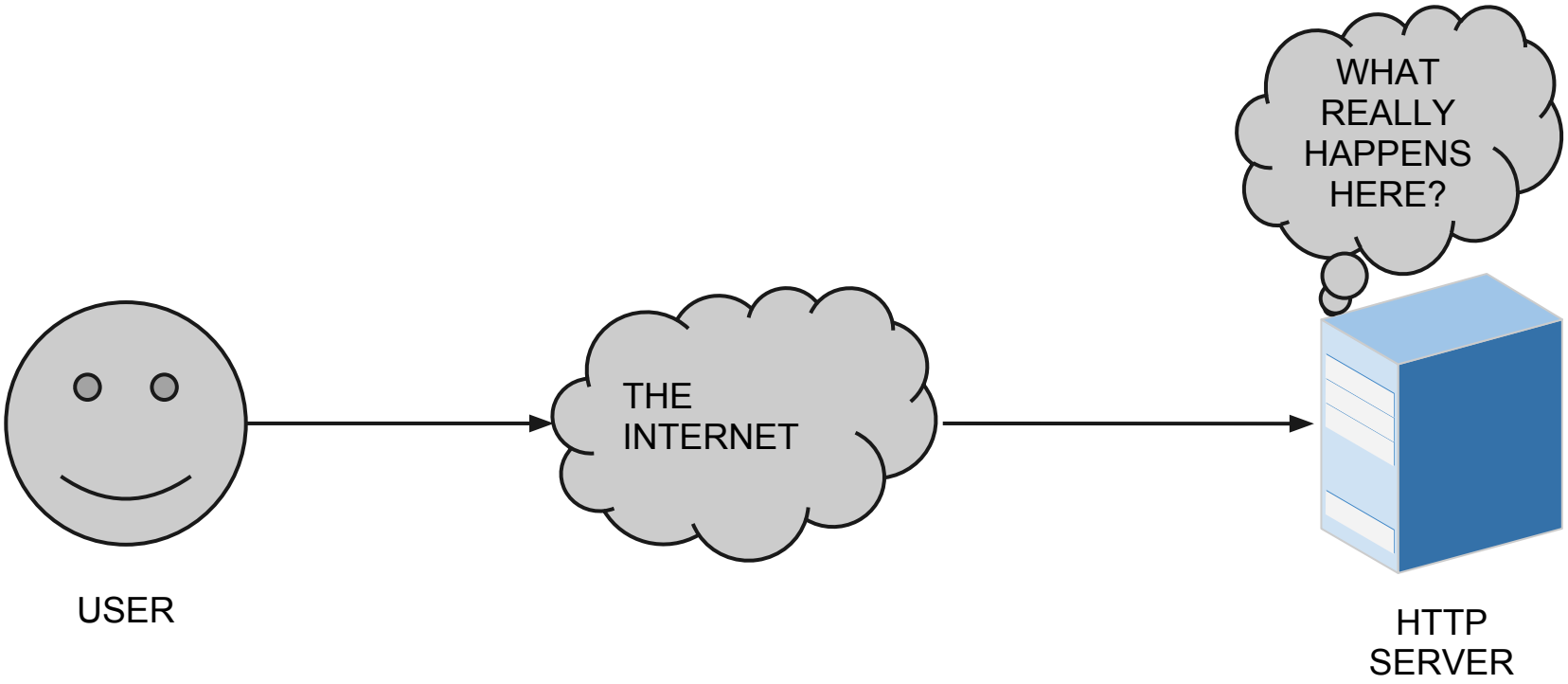
What happens when your browser knocks

Web Applications Explained with Python

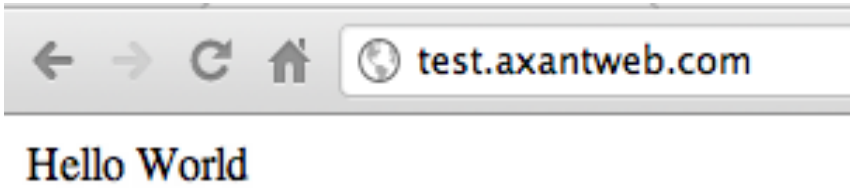
We all know the basics



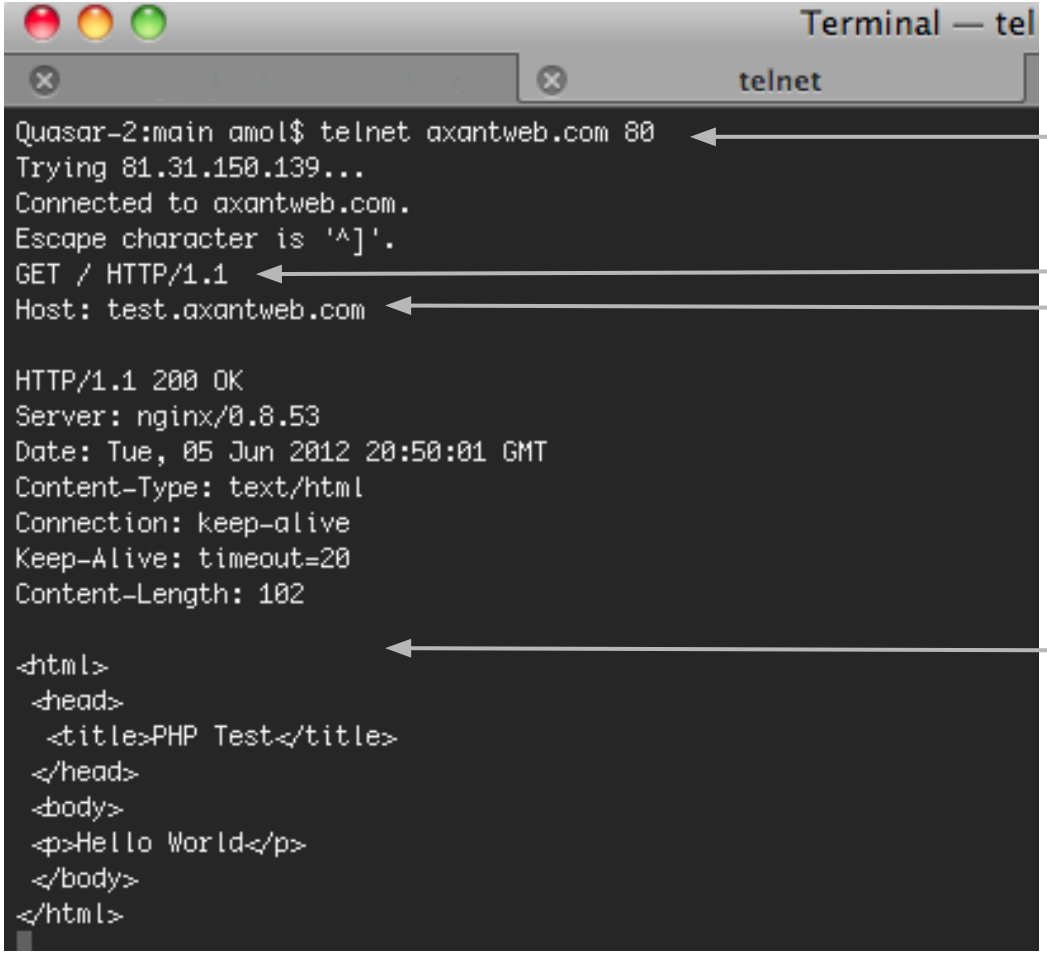
It's a PHP World

I just throw index.php inside my website directory and it works!

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```



What really happened?

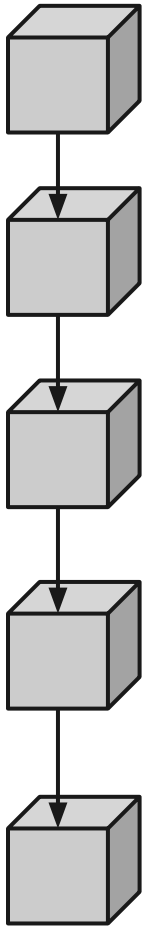


We get in touch with the webserver on port 80

We tell him that we want the path **"/"** for the website **test.axantweb.com**

We get back an answer that looks like "Hi! I'm nginx and here is the content you asked me for"

Somehow, what's behind



My Apache HTTP server received request "**GET /**"

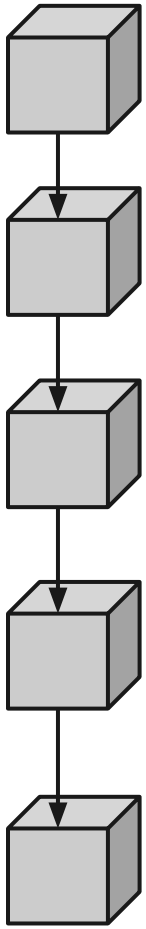
The virtual host manager of apache see the "**Host: test.axantweb.com**" line inside my request and looks for the configuration of that virtual host

Apache sees there is no path it can handle and decides to look for **index.html**, **index.php** or other index files. It finds my index.php!

Apache doesn't understand a single word of what it is inside of **index.php** so it looks for something registered to handle that kind of files.

Apache finds **mod_php** and asks him to do something

Not the end



mod_php reads the content and parses it to run php code inside **<?php** tags

URL parameters are parsed using the **urlencode** format and placed inside **$_GET**

In case of POST data, Body of the request is parsed using the related request content type. Usually **urlencode** or **multipart** and ends being the **$_POST**

output of our script is sent back to apache

apache sends it back to our user

Serving a Request

1. **Parse HTTP Request**
2. Route the Request to some code
3. Parse Request parameters
4. Parse Request Body
5. Generate an answer
6. **Send the Answer to the client**

mod_php does most of this for us, but the webserver will usually do only point 1 and 6. Understanding the other parts is great tool to write efficient and reliable applications.

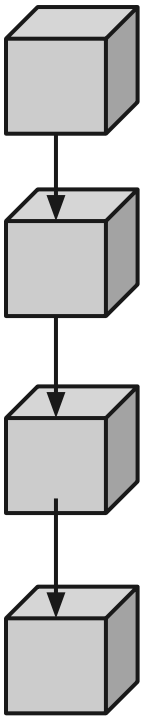
Web Apps Evolution

CGI: Each application was an executable file, the webserver would call it and get its output. Very Slow!

mod_php, mod_python and so on: Tightly coupled with web server, hard to implement custom dispatch

Modern Gateway Interfaces: HTTP server agnostic, optimized for performances, can easily do custom dispatch. **WSGI** is the Python one.

It's a Python World



My HTTP server receives the request "**GET /**"

The HTTP server forwards the request to the WSGI handler so that it can format it accordingly to the WSGI protocol

The WSGI handler runs our WSGI application and sends the output back to the HTTP server

the HTTP server sends the output back to our user

WSGI Applications

```
def hello_world(environ, start_response):
    start_response('200 OK', [('Content-type', 'text/plain')])
    return ['Hello world!\n']
```

We are in charge of:

- Routing request, all the paths will all end to our wsgi application
- Building the response
- Parsing request path
- Parsing query parameters (php $_GET)
- Parsing POST body and its parameters (php $_POST)

Parsing Query Args (urlencoded)

Most requests involve URL like: /index**?param1=v1&param2=v2**

Our web framework, being it mod_php or anything more complex has to parse the URL and extract the parameters.

```
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> myurl = '/index?param1=v1&param2=v2'
>>> url, params = myurl.split('?')
>>> params = params.split('&')
>>> params
['param1=v1', 'param2=v2']
>>> GET = {}
>>> for param in params:
...     name, value = param.split('=')
...     GET[name] = value
...
>>> GET
{'param2': 'v2', 'param1': 'v1'}
>>>
```

Multipart Encoding

Multipart form-data encoding is usually involved when sending files.

The boundaries are usually randomly generated and searched for inside the data that has be sent to make sure that they don't occur inside the data itself. In case of an occurrence a new boundary is generated.

I really don't want to!

You don't want to parse parameters, handle routing and so on by yourself. Those are all complex things that all the frameworks around will do for you.

But if you want to, you can get started at Python development without having to configure or install anything on your machine (apart python itself).

Serving Python

```
from wsgiref.simple_server import make_server

def hello_world_app(environ, start_response):
    status = '200 OK' # HTTP Status
    headers = [('Content-type', 'text/plain')] # HTTP Headers
    start_response(status, headers)
    return ["Hello World"]

httpd = make_server('', 8000, hello_world_app)
print "Serving on port 8000..."
httpd.serve_forever()
```

The Environ

The WSGI environ variable is a dictionary where our webserver will place all the data he knows about.

All the HTTP Request headers will be available in a key with the header name preceded by HTTP_. The **Host** header for example will be available as **HTTP_HOST**.

environ.py

Routing

How do I manage /index, /articles or even /article/5? This is what **routing** does!

Regular Expressions: All the routes that match a given regular expression will go to a function or class

Object Dispatch: Routes are splitted on / as the path of an object and its parents, last entry is the method of the object to call

Traversal: Like ObjectDispatch but uses dictionaries instead of objects.

Template Engines

```
${%def test(x):}
    <span>${x}</span>
${%end}
<html>
    <head>
        <title>${title}</title>
    </head>
    <body>
        ${%for i in range(5):}
            ${%test(i)}
        ${%end}
    </body>
</html>
```

HTML has often to be dinamically generated and appending strings is not the best way to do it.

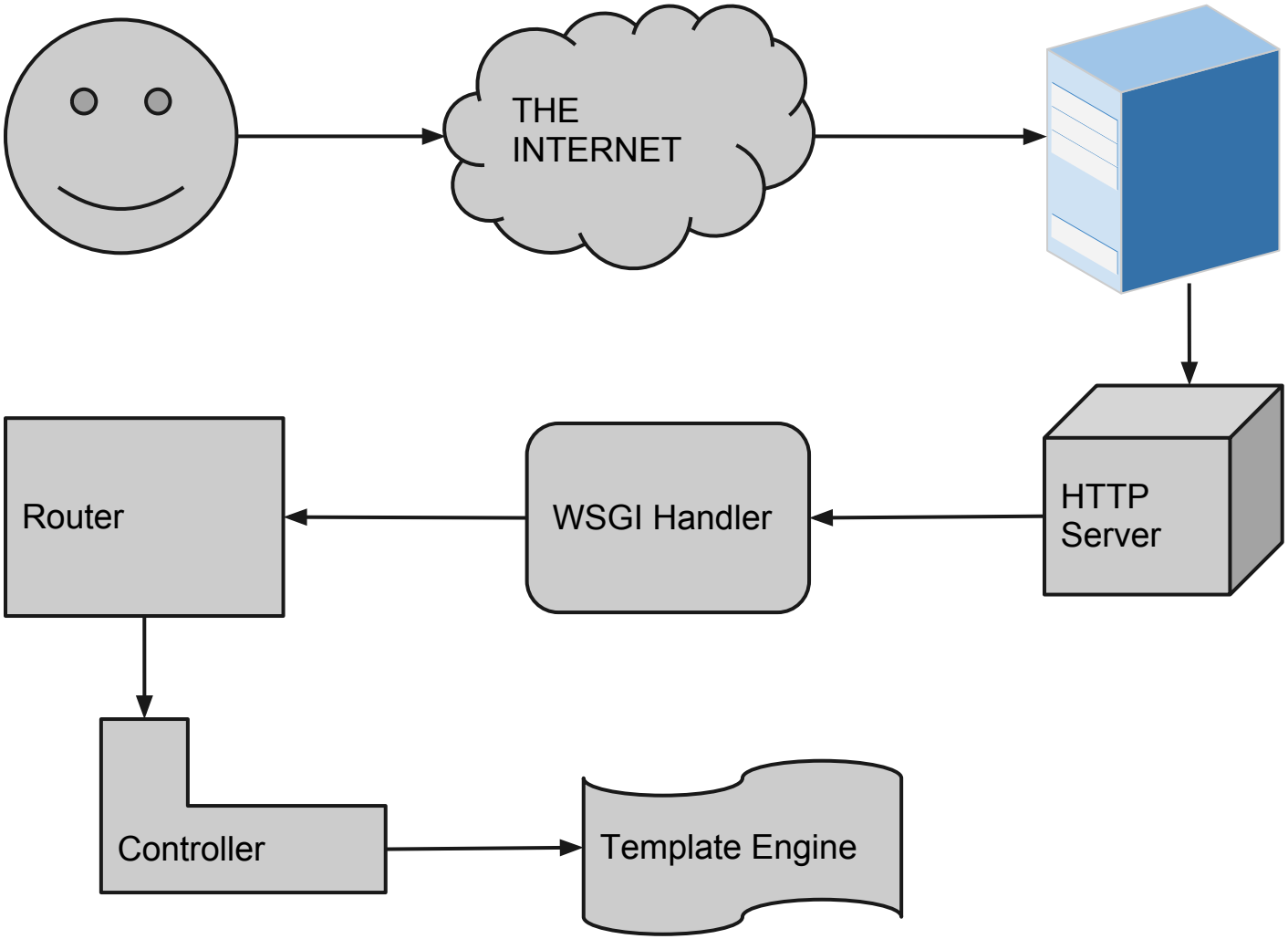
Template Engines permit to generate HTML without effort.

<?php tag of mod_php can be seen as a minimalistic template engine

How Template Engines Work

```
>>> print ArtichokeTemplate(txt)._source
def template():
        def test(x):
                __output__.append(u"<span>")
                __output__.append(escape(x))
                __output__.append(u"</span>")
        __output__.append(u"<html>")
        __output__.append(u"<head>")
        __output__.append(u"<title>")
        __output__.append(escape(title))
        __output__.append(u"</title>")
        __output__.append(u"</head>")
        __output__.append(u"<body>")
        for i in range(5):
                test(i)
        __output__.append(u"</body>")
        __output__.append(u"</html>")
template()
>>>
```

Everything Together



Storing Data

Most web frameworks provide for us a way to store data on server or on client and to match those.

Most commonly:

- Cookies
- Authentication
- Sessions

Cookies

Cookies are the foundation of Sessions and Authentication. Without them we wouldn't be able to track the user when he gets back

They are implemented using two HTTP headers: **Cookie** and **Set-Cookie**

mod_php **$_COOKIES** and **setcookie** are actually wrappers around those headers.

How Cookies are managed

```
def setcookie(headers, name, value):
    cookie = '%s=%s' % (name, value)
    headers.append(('Set-Cookie', cookie))

def getcookies(environ):
    cookies = {}
    for cookie in environ['HTTP_COOKIE'].split(';'):
        name, value = cookie.split('=')
        cookies[name.strip()] = value.strip()
    return cookies

def delcookie(headers, name):
    headers.append(('Set-Cookie', '%s=;max-age=0' % name))
```

When we want to set or update a cookie we send back a **Set-Cookie** header that will look like: **Set-Cookie name=value**

Cookies are passed inside multiple Cookie headers, WSGI will group them for us but we will have to parse them.

A quick way to delete a cookie is to set it with a maximum duration o 0 seconds.

Actually there is much more, like cookie Path, Expiration and so on.

Request & Response objects

Most Python frameworks have the concept of the so called Request and Response objects.

Those are objects that represent your browser request and the response you will send back.

Most common implementation is **WebOb** module which is used by Google AppEngine, TurboGears2, Pyramid, Pylons and many other frameworks.

Facing Truth

You don't want to manage this complexity by youself, always rely on a Web Framework