ICA0002: IT Infrastructure Services

Web Servers

Roman Kuchin
Juri Hudolejev
2024

Basic terms

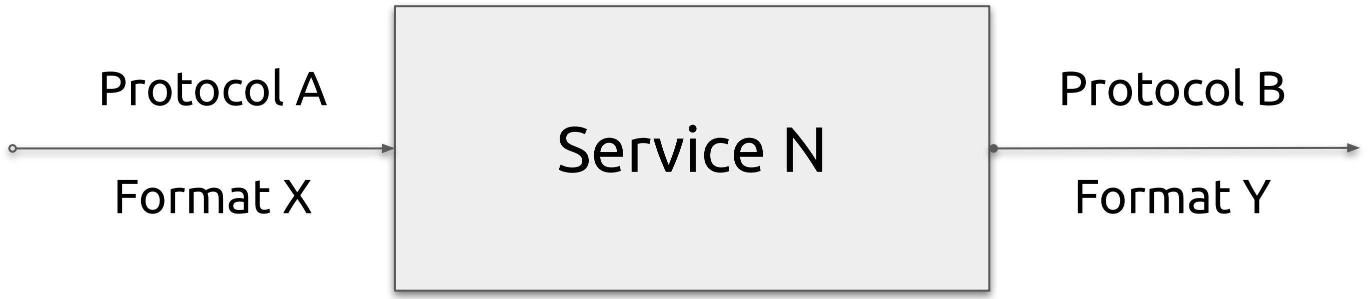
WWW: World Wide Web, the Web

URL: Uniform Resource Locator

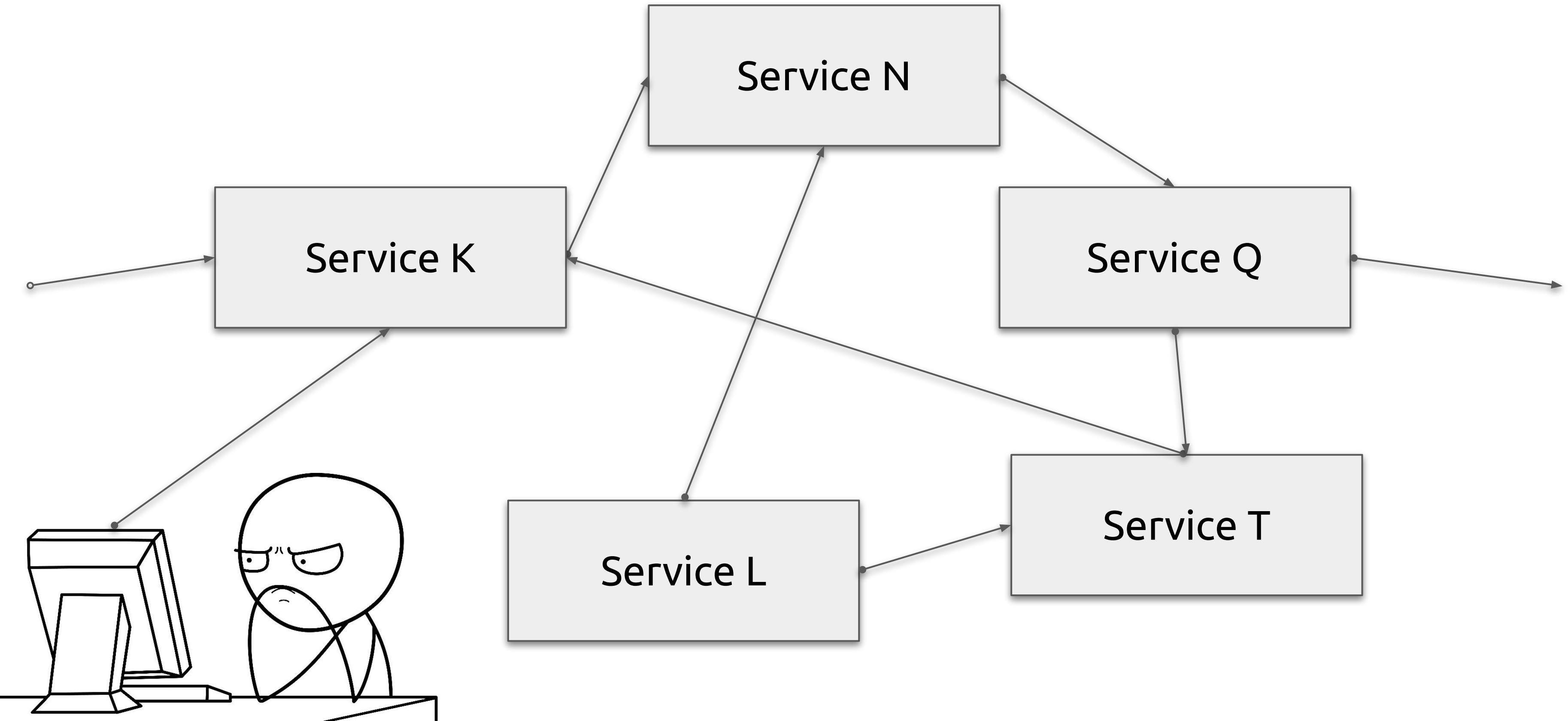
Web server

Web client (web browser, user agent)

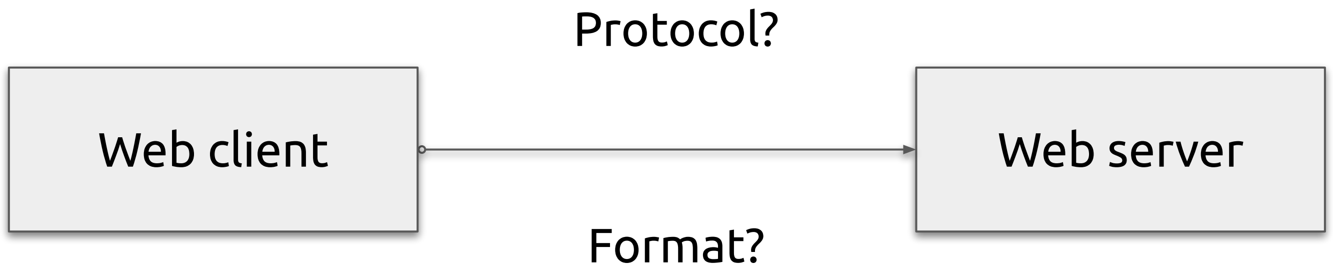
(very) Generic IT service



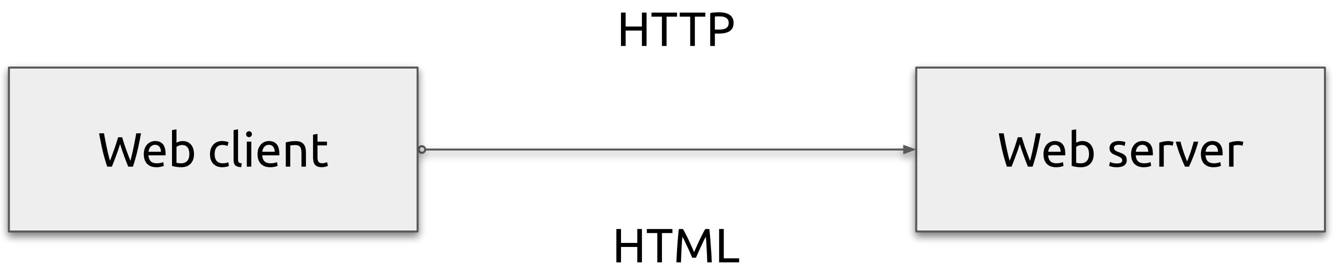
Service communication



Web client and web server

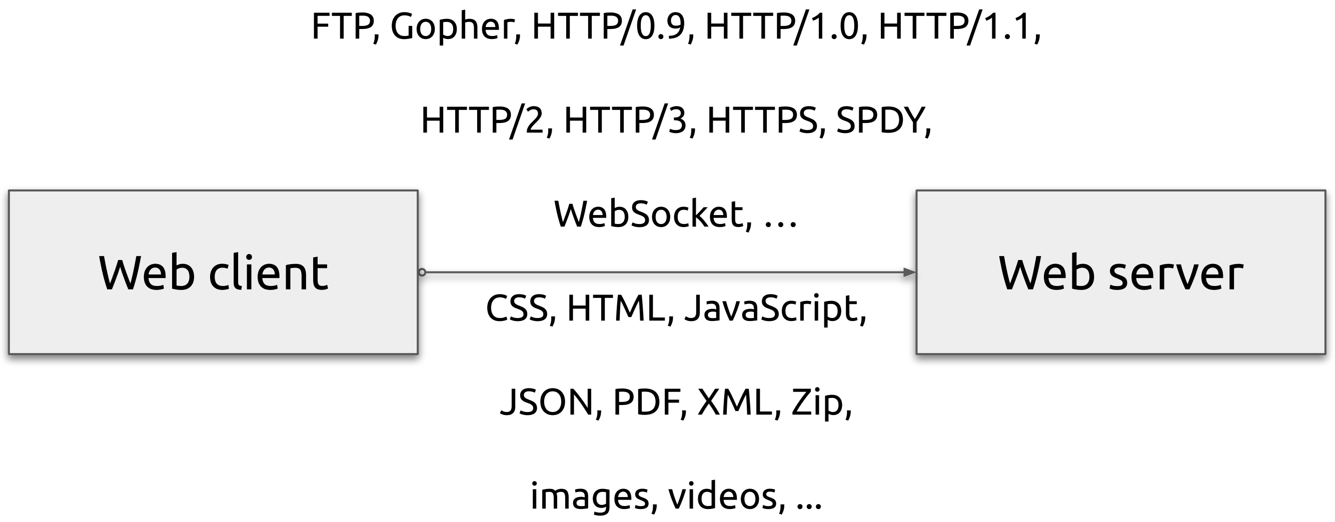


Web client and web server

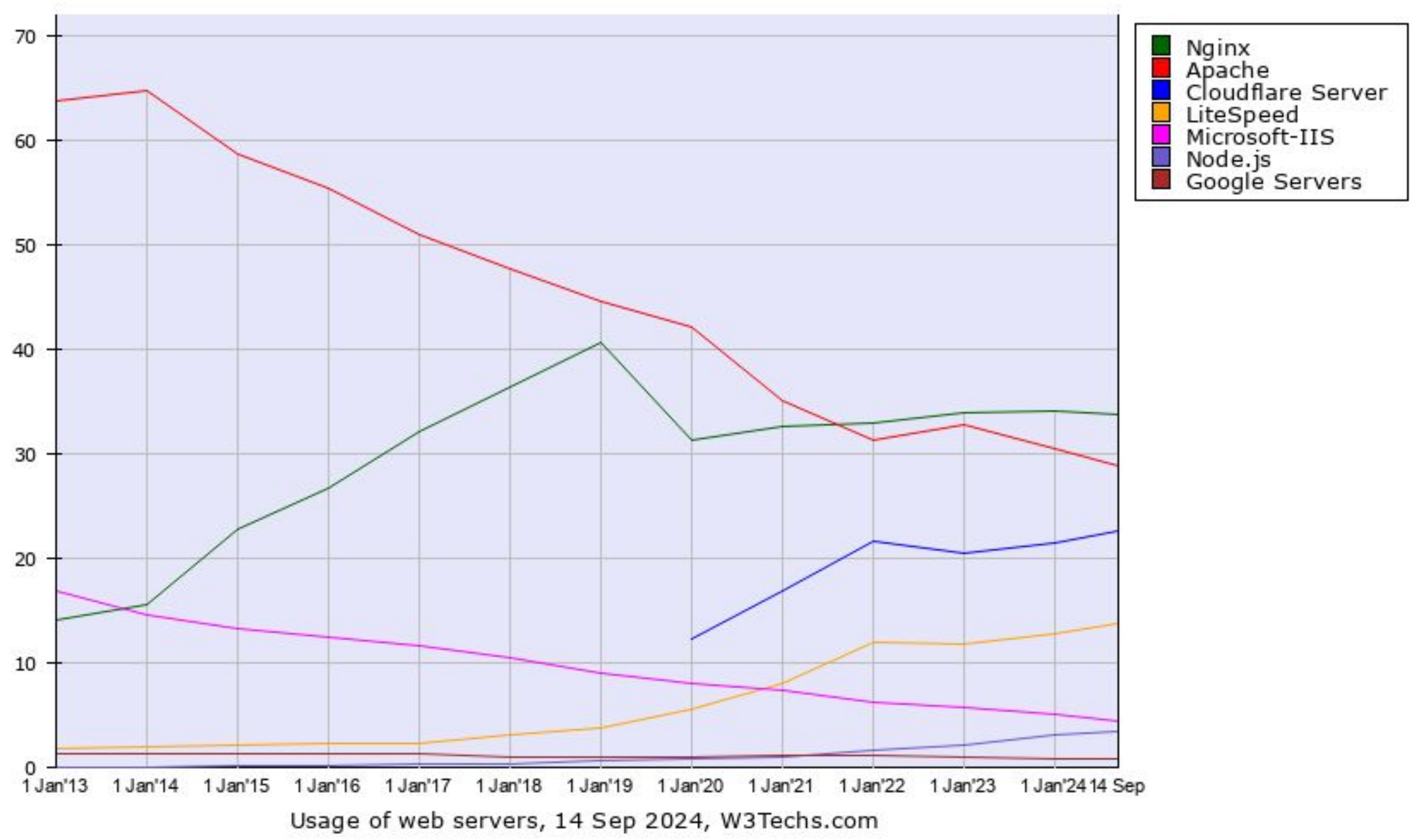


Demo time!

Web client and web server



Web server market share



Apache HTTPd

The oldest of the existing mainstream web servers, and still widely used

Free and open-source, maintained by Apache Software Foundation

First release in 1995, current stable version: 2.4

Modules for TLS, server-side scripts, authentication, proxying, etc.

- List of modules: https://en.wikipedia.org/wiki/List_of_Apache_modules

Web site: https://httpd.apache.org

Nginx

Rather 'new' web server: first public release in 2004, current stable version: 1.26

- Nginx: free and open-source (BSD license)
- Nginx Plus: proprietary

The most widely used web server today, closely followed by Apache HTTPd

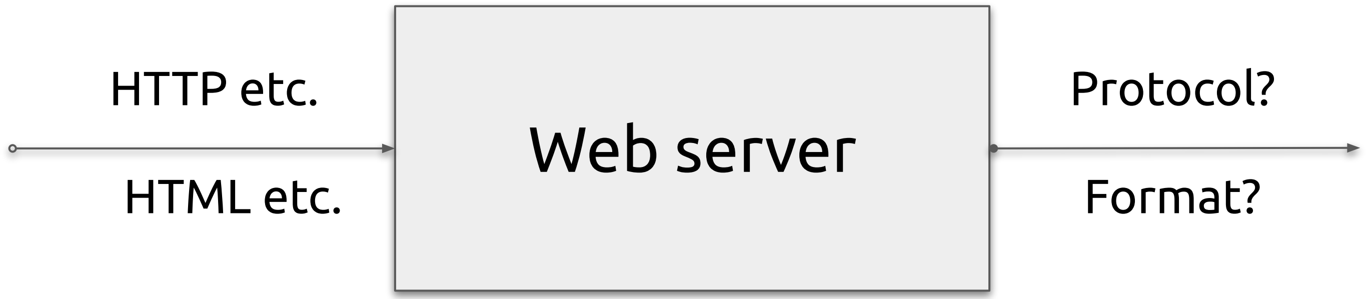
Web server, HTTP proxy, load balancer

- List of modules: https://nginx.org/en/docs

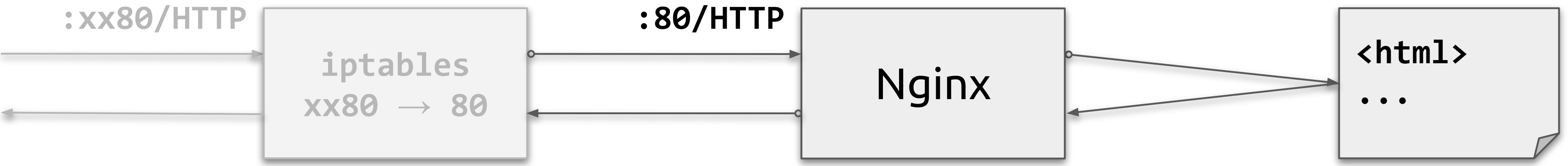
Web site: https://nginx.org

Questions?

Behind the web server



Previous lab



Web server operation modes

Static documents:

- web server sends files from local filesystem as is

Dynamic documents:

- web server interacts with other programs (scripts) to generate the resource on the fly (dynamically), and sends that generated resource to the client

Proxy mode:

- web server forwards request to other services

Web server operation modes

Static documents:

- web server sends files from local filesystem as is

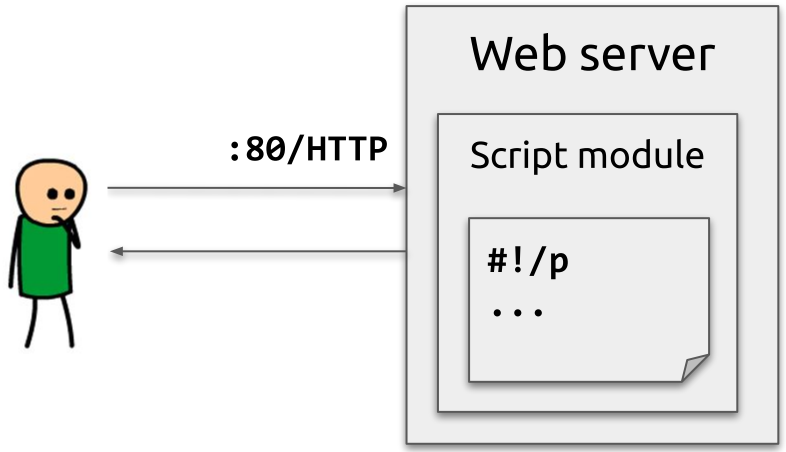
Dynamic documents:

- web server interacts with other programs (scripts) to generate the resource on the fly (dynamically), and sends that generated resource to the client

Proxy mode:

- web server forwards request to other services

Web server script modules



Server runs the script inside the main process using the extension module

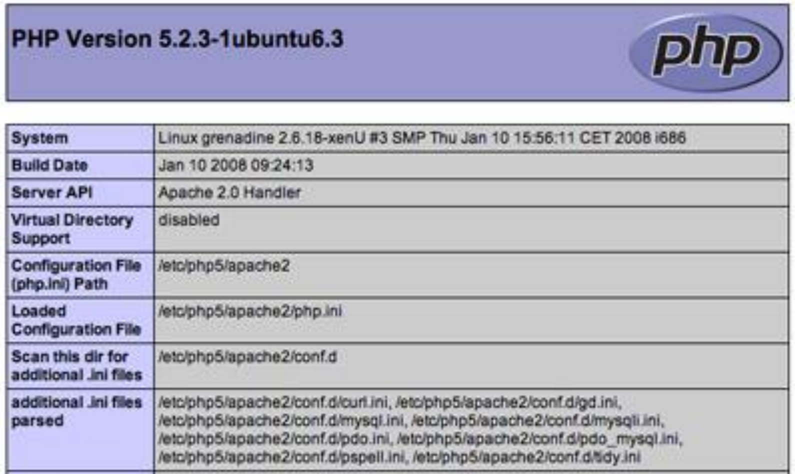
- Apache HTTPd: Perl module, PHP module etc.
- Nginx: Lua module, JavaScript module etc.

Dynamic resource example

```php
<?php

echo '<h1>It works!</h1>';
```

It works!

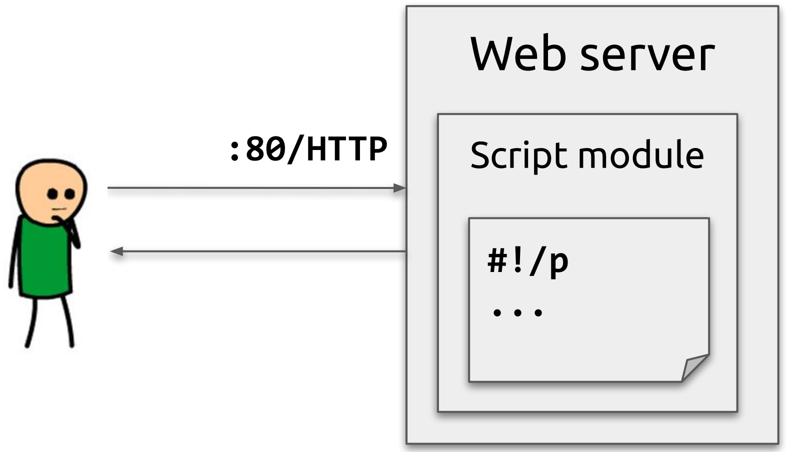
```php
<?php

phpinfo();
```



PHP Version 5.2.3-1ubuntu6.3

System	Linux grenadine 2.6.18-xenU #3 SMP Thu Jan 10 15:56:11 CET 2008 i686
Build Date	Jan 10 2008 09:24:13
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini, /etc/php5/apache2/conf.d/pspell.ini, /etc/php5/apache2/conf.d/tidy.ini

Web server script modules

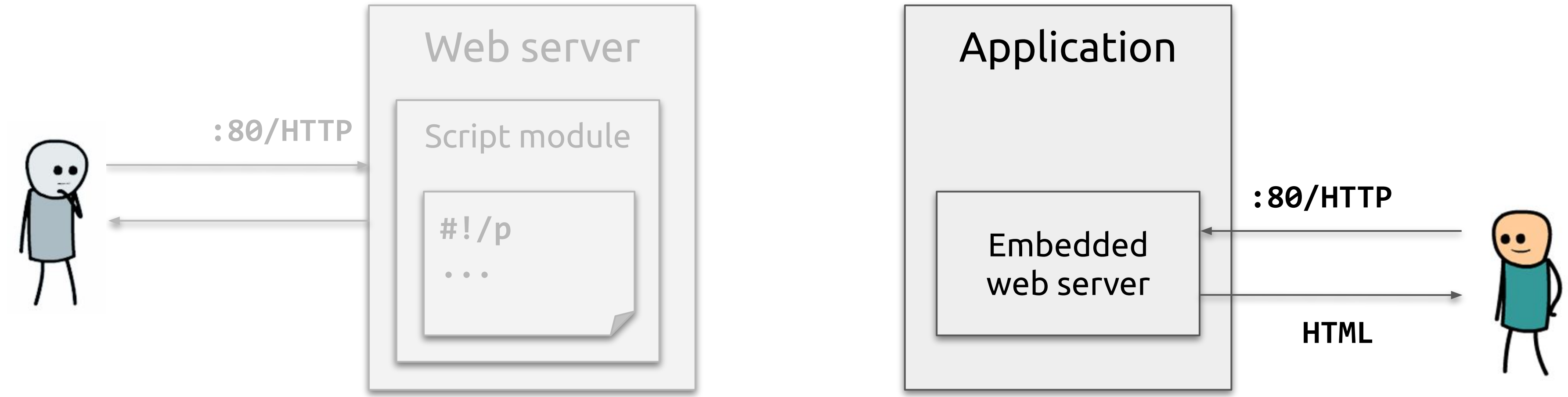


Probably the fastest method if configured correctly

Web server needs a custom module

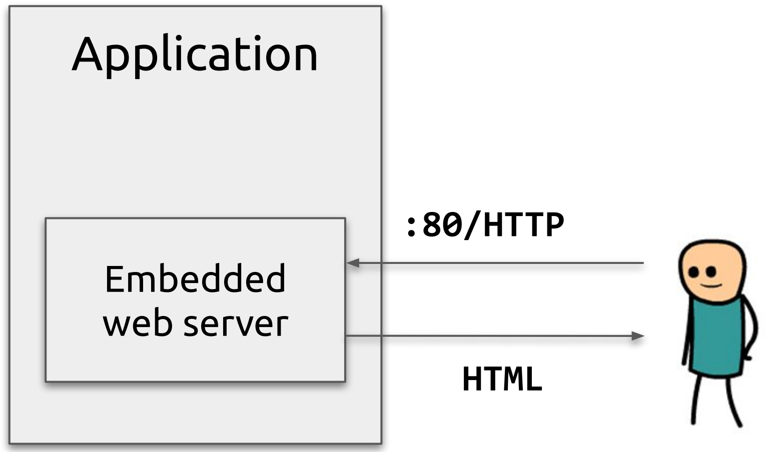
Script runs inside web server -- security issues

Embedded web servers



Instead of web server running an app (script) -- app could run a web server!

Embedded web servers



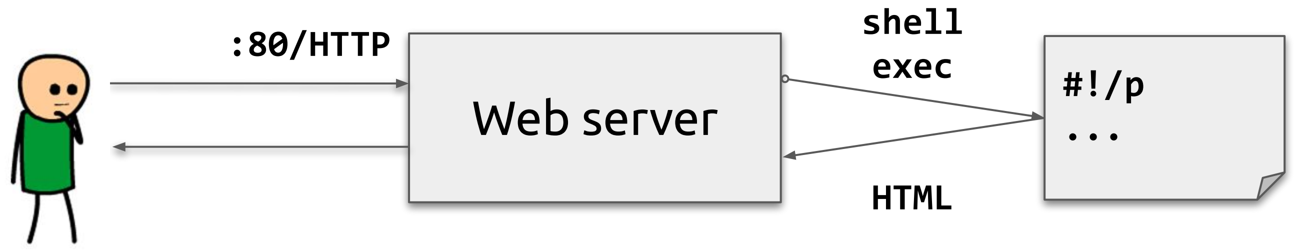
Upgrades are pain

Lack of features as compared to standalone web servers

Reimplementing the web server on every programming language

Performance issues: works for Java etc. (sort of) but not for scripting languages

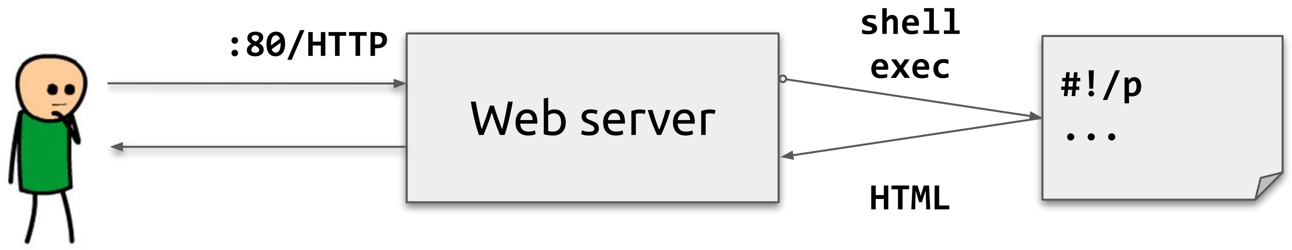
External scripts



Script is executed by Web server as a separate process

The simplest and the earliest known method

External scripts



Slow and very inefficient

Script runs in the context of web server -- security issues

No standard interface for servers to communicate with scripts

Gateway interfaces

1993: Common Gateway Interface (CGI)

Gateway interfaces

1993: Common Gateway Interface (CGI)

1996: FastCGI (binary protocol) -- scripts are run by a separate process

Gateway interfaces

1993: Common Gateway Interface (CGI)

1996: FastCGI (binary protocol) -- scripts are run by a separate process

2001: Simple Common Gateway Interface (SCGI)

Gateway interfaces

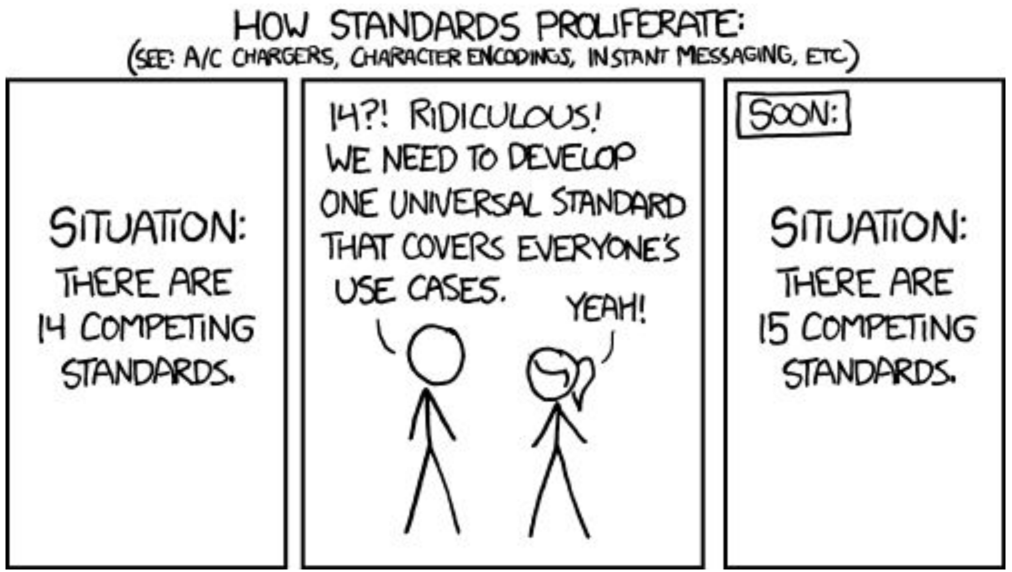
1993: Common Gateway Interface (CGI)

1996: FastCGI (binary protocol) -- scripts are run by a separate process

2001: Simple Common Gateway Interface (SCGI)

Netscape, Microsoft, Apache etc. developed their own protocols

Web server modules to run scripts are still there


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

https://xkcd.com/927

Gateway interfaces

1993: Common Gateway Interface (CGI)

1996: FastCGI (binary protocol) -- scripts are run in a separate process

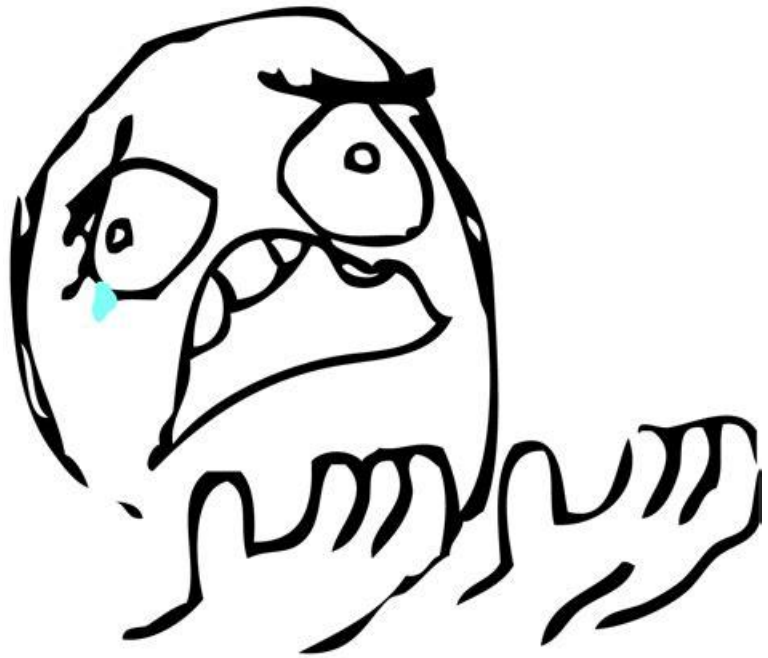
2001: Simple Common Gateway Interface (SCGI)

2003: Web Server Gateway Interface (WSGI) for Python

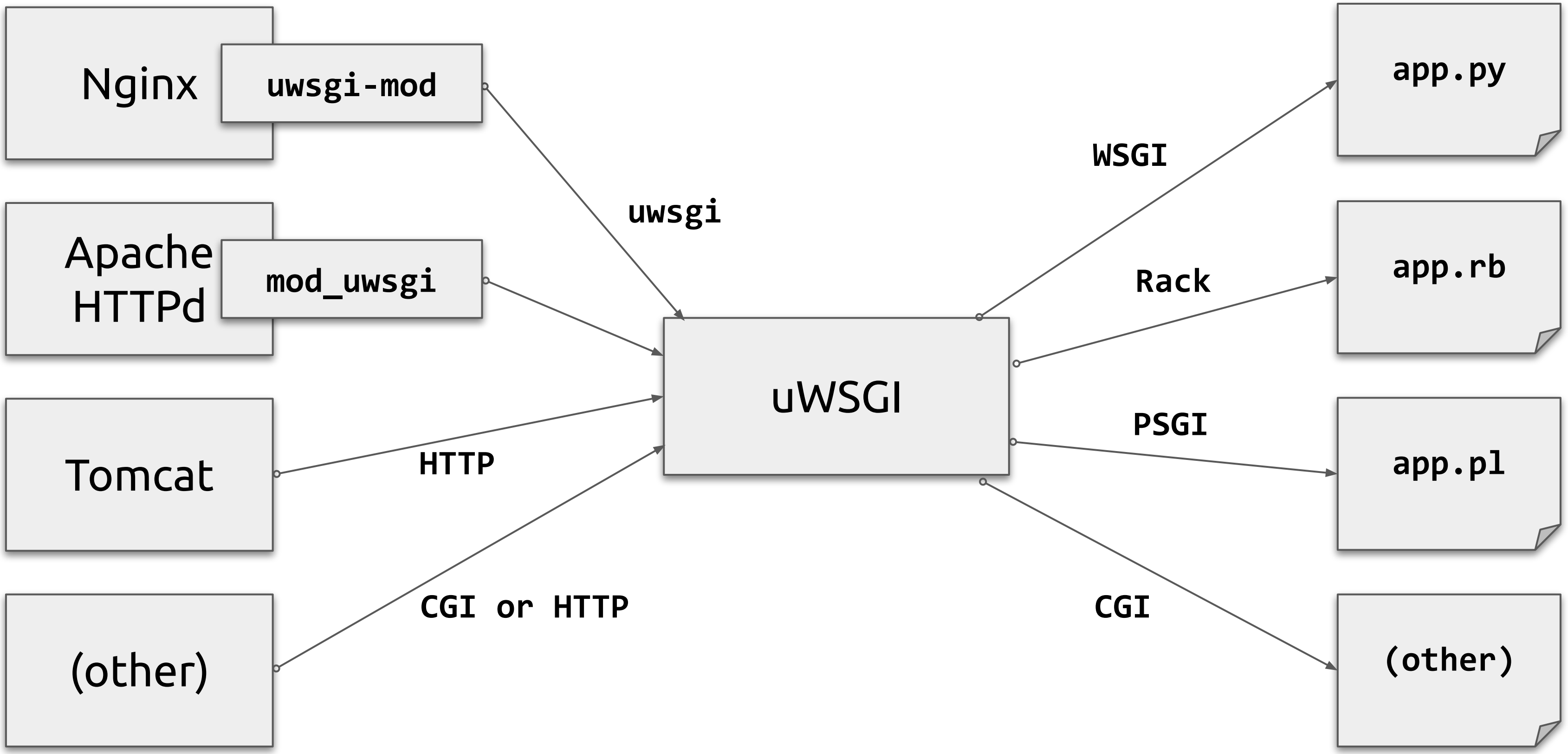
Followed by JSGI for JavaScript, PSGI for Perl, Rack for Ruby etc.

Good read: https://docs.python.org/3.4/howto/webservers.html

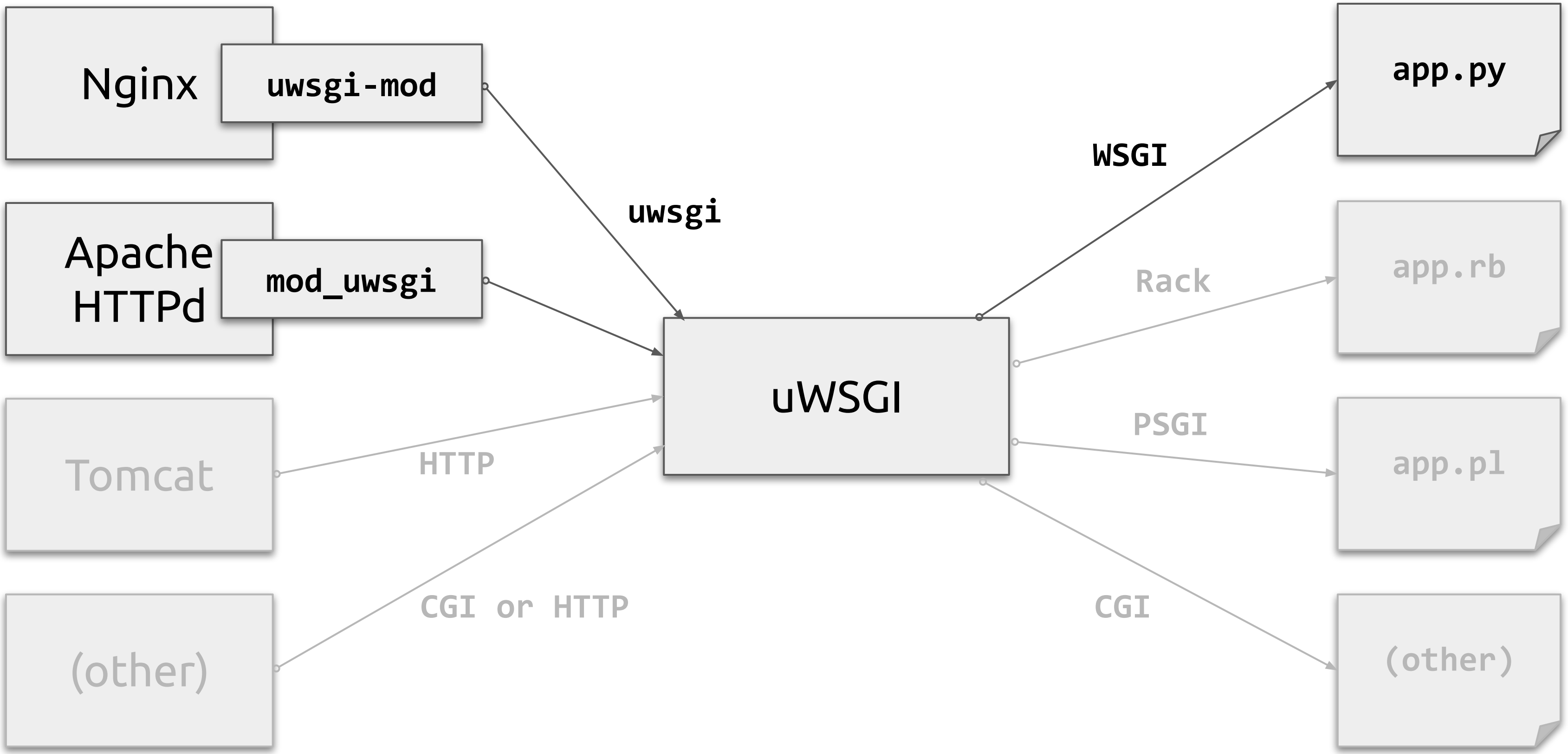
CGI SCGI FastCGI PSGI WSGI and all of these GI-s...



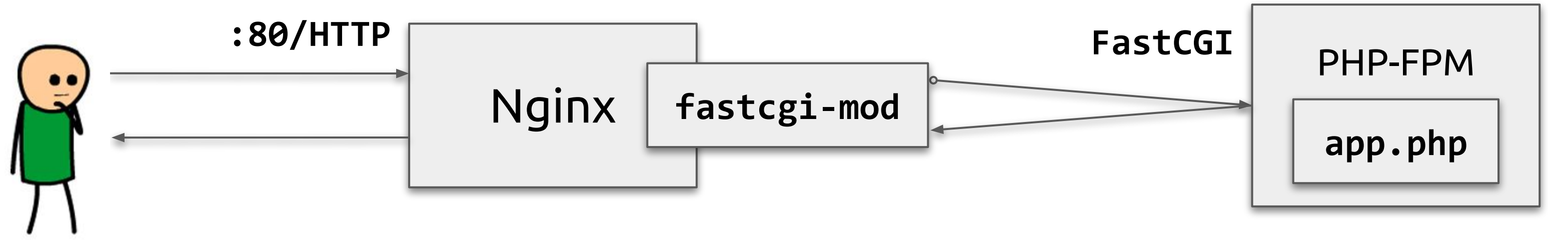
uWSGI mission

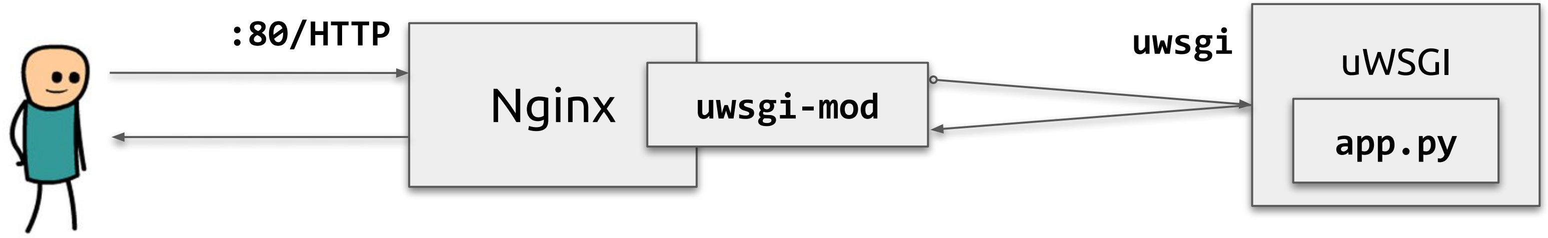


uWSGI mission



FastCGI and uwsgi examples





Script is executed by **application server** (FPM, uWSGI, Unicorn etc.)

Nginx FastCGI configuration example

```
server {
    listen 80;

    location / {
        fastcgi_pass 127.0.0.1:9000;  # may be remote host as well
        include fastcgi_params;       # found in /etc/nginx/
    }
}
```

uWSGI example is almost identical (**`fastcgi_pass`** → **`uwsgi_pass`**)

Dynamic web resources

1. Web server runs the script (app) "inside" to generate the resource

   Easier to set up but not very resource efficient and has security issues

2. App generates the resource and runs the embedded web server to serve it

   Language-specific solution, lack of features

3. Web server communicates with app server that generates the resource

   More complex to set up but is usually preferred for larger deployments

Principle 1 of Unix philosophy:

Write programs that do **one** thing and do it well.

Other web server topics

Covered later in this course:

- Proxying
- High availability

Out of scope of this course:

- HTTPS, SSL/TLS
- WebSockets, HTTP/2, HTTP/3
- Caching

Questions?