HOSTING PYTHON WEB APPLICATIONS

Graham Dumpleton

PyCon Australia

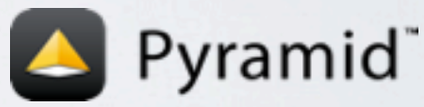
Sydney 2011

WEB APPLICATIONS

Plone®

Only a few well known Python web applications.

WEB FRAMEWORKS



Many Python web frameworks for building your own however.

FRAMEWORK SERVERS

- BaseHttpServer.HttpServer (Django, Werkzeug, Flask)
- wsgiref.simple_server (Bottle)
- paste.httpserver (Pylons, Pyramid, TurboGears)
- Rocket (Web2Py)
- cherrypy.wsgiserver (CherryPy, web.py)

Sunday, 21 August 2011

Most web frameworks bundle in there own web server to make development easier. They often have the ability to perform automatic detection of code changes and will restart automatically. These servers fall into two categories based on how capable they are.

DEVELOPMENT ONLY

- BaseHttpServer.HttpServer (Django, Werkzeug, Flask)
- wsgiref.simple_server (Bottle)

Sunday, 21 August 2011

The simplest of these are usually based on basic web server functionality provided in the Python standard library. These are definitely only suitable for development and definitely not adequate for production use.

SHORT COMINGS

- Non existent or inadequate concurrency support.
- SSL support non existent or not of production quality.
- Lack of good logging support.
- Inadequate process management.
- No direct static file serving ability.

Sunday, 21 August 2011

Just not up to handling the rigours of a high load production web site, have not been security audited and could just well result in your site blowing up as soon as any one looks at it.

PRODUCTION GRADE?

- paste.httpserver (Pylons, Pyramid, TurboGears)
- Rocket (Web2Py)
- cherrypy.wsgiserver (CherryPy, web.py)

Claimed by the developers to be suitable for production usage and do see a lot of use as a result. Reasonably safe option but mileage may vary.

GETTING BETTER

- Don't provide a full process management solution.
- Still usually require supervisord/monit to manage them.
- No direct static file serving ability.
- Still use Apache/nginx/lighttpd/Cherokee for static files.
- Usually recommended they sit behind traditional server.

Sunday, 21 August 2011

Still not a complete solution however. Generally need separate process management to ensure they stay running and definitely not good solutions for static file serving. Will perform better as a result when placed behind a front end web server.

INDEPENDENT SERVERS

- gunicorn
- meinheld
- fapws
- bjoern
- diesel
- gevent
- eventlet
- tornado
- twisted
- pants

Lots of other options. Rather than build new web frameworks, these days the cool thing to do seems to be building new WSGI servers.

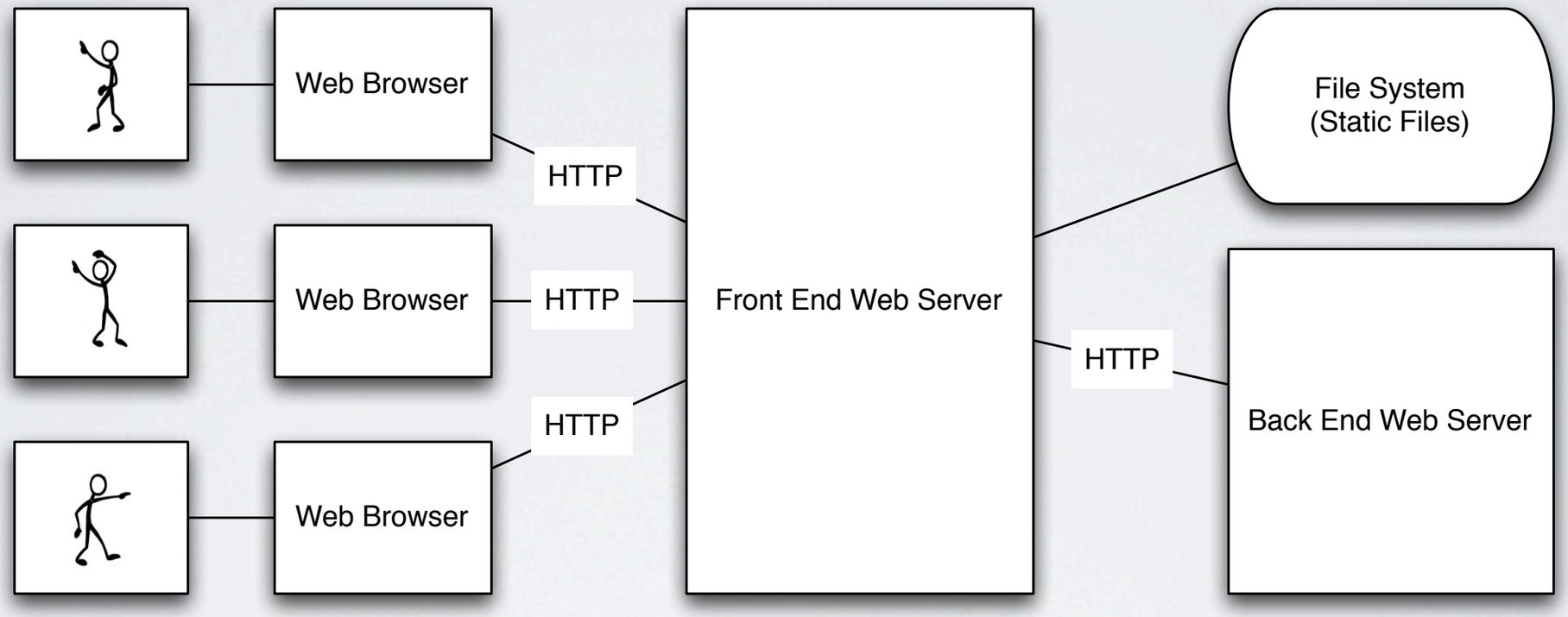
MIXED BAG

- The gunicorn server best of this crowd.
- Still usually require supervisord/monit to manage it.
- No direct static file serving ability.
- Still use Apache/nginx/lighttpd/Cherokee for static files.
- Usually recommended they sit behind traditional server.

Sunday, 21 August 2011

Gunicorn probably the best standalone WSGI server and is the flavour of the month. Like paste.httpserver, Rocket and cherrypy.wsgiserver still benefits by being placed behind a front end web server.

FRONT END WEB PROXY



Front end web server handles SSL, virtual hosts and static file serving. All requests for dynamically generated content are proxied through to the back end web server. In this case our Python WGSI server. The idea is here is that no one web server is going to be the best at everything so we start specialising and use a server in front designed to handle static content.

PERFORMANCE BENEFITS

- Static file handling speed far superior.
- Can implement caching in the front end web server.
- Isolates dynamic web application from slow clients.
- Removes need for application server to handle keep alive.

Sunday, 21 August 2011

In addition to the the benefits we get from improved static file handling, can implement caching and avoid requests getting to Python WSGI server. The architecture of the solution also helps as it isolates the Python WSGI server from slow clients and requests only get to if needed and when ready.

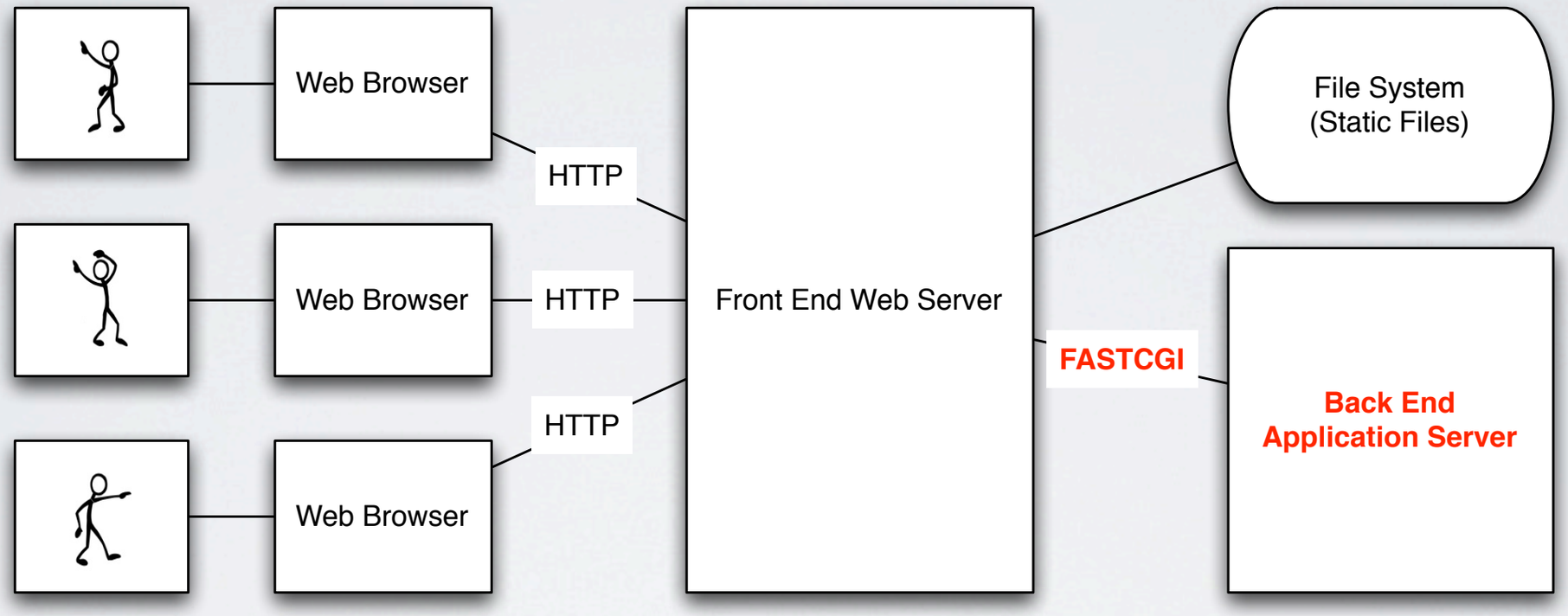
GOOD OPTIONS

- nginx
- Cherokee
- lighttpd
- ~~Apache~~

Sunday, 21 August 2011

Best options for a front end web server are any which are implemented using an asynchronous or event driven system. Use of a threading module is just too heavy weight when dealing with high levels of concurrency. Apache is therefore not a good option for a high volume site, with nginx being arguably the best.

ALTERNATIVES TO HTTP



Rather than proxy HTTP, then can use FASTCGI. Back end is no longer strictly a web server but an application server using a custom protocol. The WSGI adapter in this case bridges between the FASTCGI wire protocol and the WSGI API rather than the HTTP protocol. If there is little difference with how things work with HTTP however, some argue what is the point.

GOOD POINTS

- Language agnostic, can support more than Python.
- Web server may optionally be able to handle processes.
- If so, no longer need to use supervisord/monit.
- Run as distinct user rather than web server user.
- Request timeouts and controls on rampant processes.

Sunday, 21 August 2011

Provides a reasonably robust model when implemented properly. For shared web hosting companies, that it can be used to support multiple languages is very convenient. Web hosting companies don't want to have to deal with distinct managed processes listening on separate ports. Want a seamless solution that doesn't need manual setup.

BAD POINTS

- Different languages don't have same resource requirements.
- Generally configured for PHP, which sucks for Python.
- Deployment for Python is more complicated than necessary.

Sunday, 21 August 2011

Problem with shared web hosting is that the setups are PHP biased and suck for Python. Python needs more memory and processes really need to be persistent. For PHP you can just drop in a .php file, but for Python you can't just drop in a .wsgi script file and have it work. Deployment experience could be made better quite easily, but no one seems to care.

RECENT ALTERNATIVES

- uWSGI
- Phusion Passenger
- Mongrel

uWSGI started in Python now branching into other languages. Phusion Passenger and Mongrel started in Ruby and now moving into Python. uWSGI has similar architecture to FASTCGI but uses a slight variation on SCGI for wire protocol. Process management is distinct from the web server.

HOSTING ADOPTION

- FASTCGI solutions still the incumbent with web hosting.
- Phusion Passenger used because of Ruby popularity.
- uWSGI slowly making headway with multi language support.
- Can't see the existing PHP hosting world changing at all.
- Better solutions will only come with new hosting providers.

Sunday, 21 August 2011

Reality is that web hosting companies specialising in PHP will probably never provide a good solution for Python. Some will try to cater for what is popular, but deployment never seamless. Hope comes in the form of new hosting companies with no legacy. DotCloud for multi language and possibly others. For Python, gondor.io, ep.io and djangozoom.

ELEPHANT IN THE ROOM

- Apache is still the most popular web server in use.
- System administrators more likely to know Apache.
- Apache is more a platform than just a web server.
- Support for Python in form of mod_python and mod_wsgi.

Sunday, 21 August 2011

Apache is still used in over 60% of web sites. In contrast nginx is used in less than 10%. More collective experience with using Apache than other servers, but it has its detractors. Accusations of being bloated and hard to configure. Python has especially got a raw deal with this because of servers not being set up properly for Python use.

MOD_PYTHON

- Runs Python embedded in Apache child worker processes.
- Officially declared dead by Apache Software Foundation.
- No longer supported and no new releases.
- Will not be ported to Python 3.0+.
- Still suffers memory leaks.
- Bugs if you deviate off the path.

Sunday, 21 August 2011

It has prime real estate in the name mod_python and is what a lot of people will logically look for when first thinking of Python and Apache. Just don't go there. The project has been declared dead and best left that way.

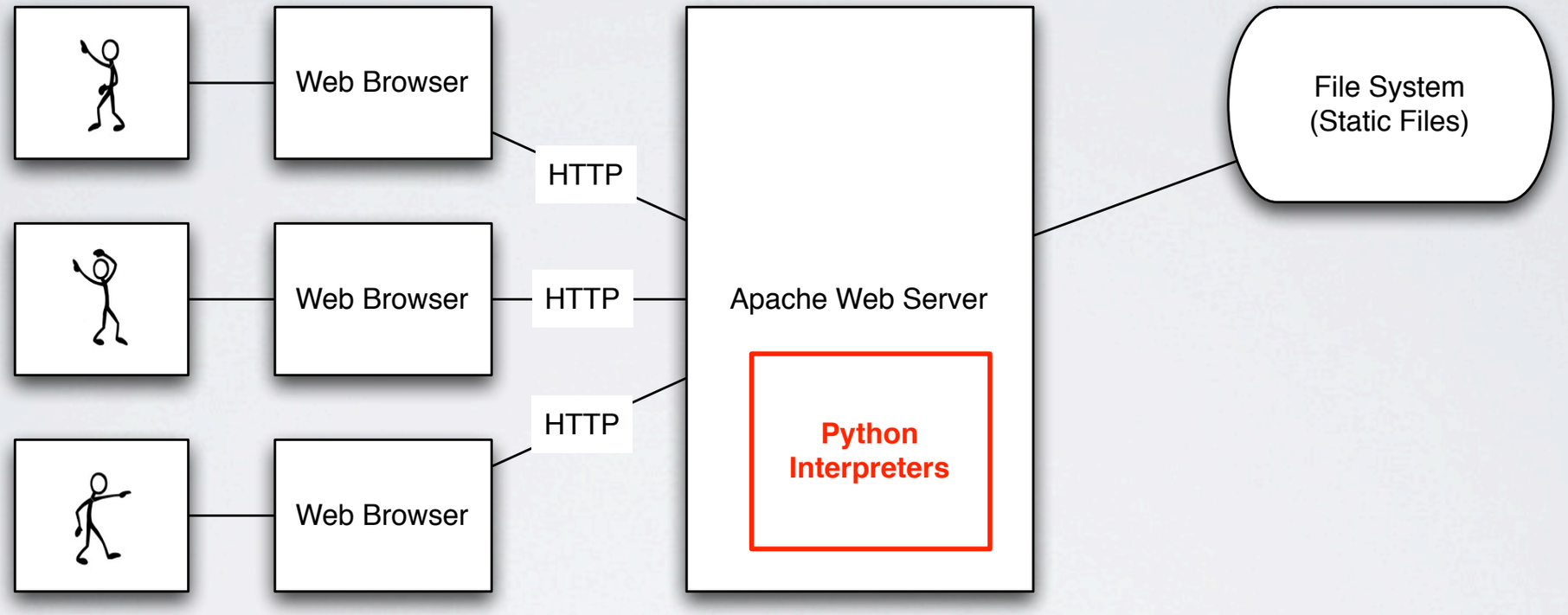
MOD_WSGI

- Intended to be a better mod_python.
- Focus on providing a WSGI compatible API.
- Limited abilities for Apache authnz providers.
- Good robust general purpose solution.
- Multiple modes of operation.

Sunday, 21 August 2011

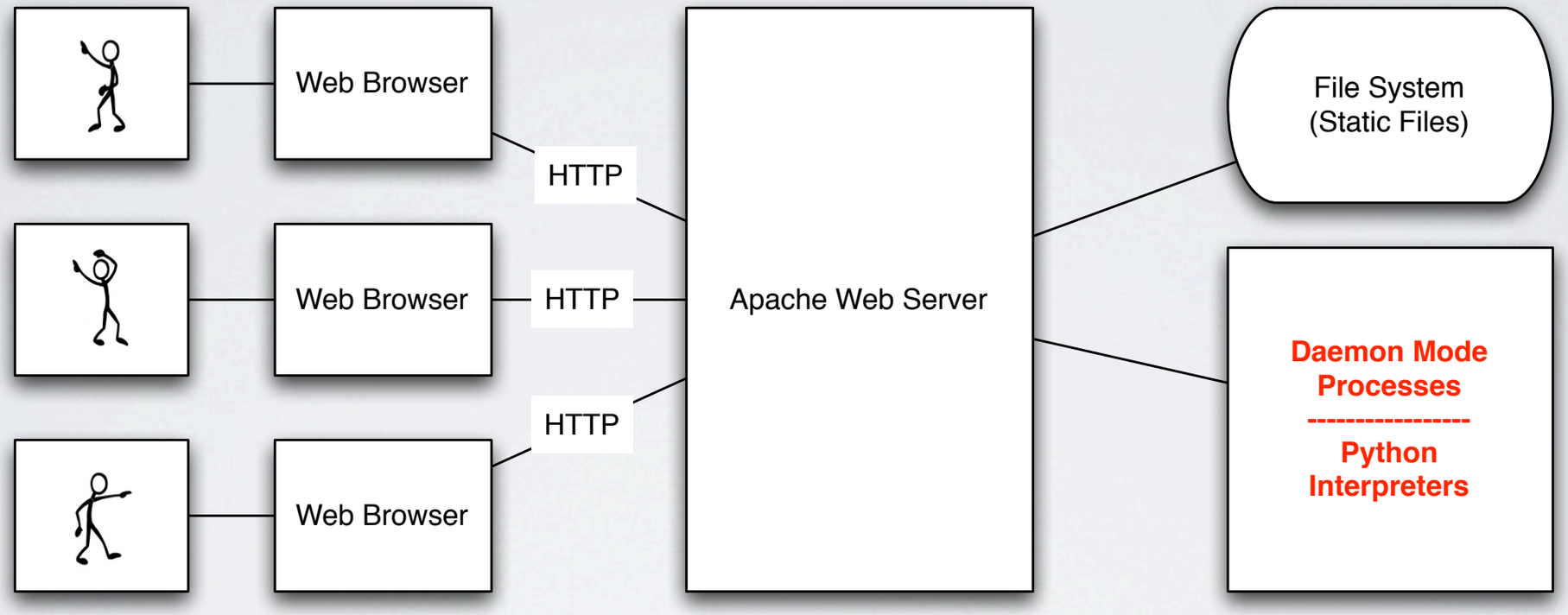
Learnt from the mistakes of mod_python. Doesn't try to be a framework in itself but just a bridge to any WSGI application/framework. Can be used in embedded mode or daemon mode.

EMBEDDED MODE



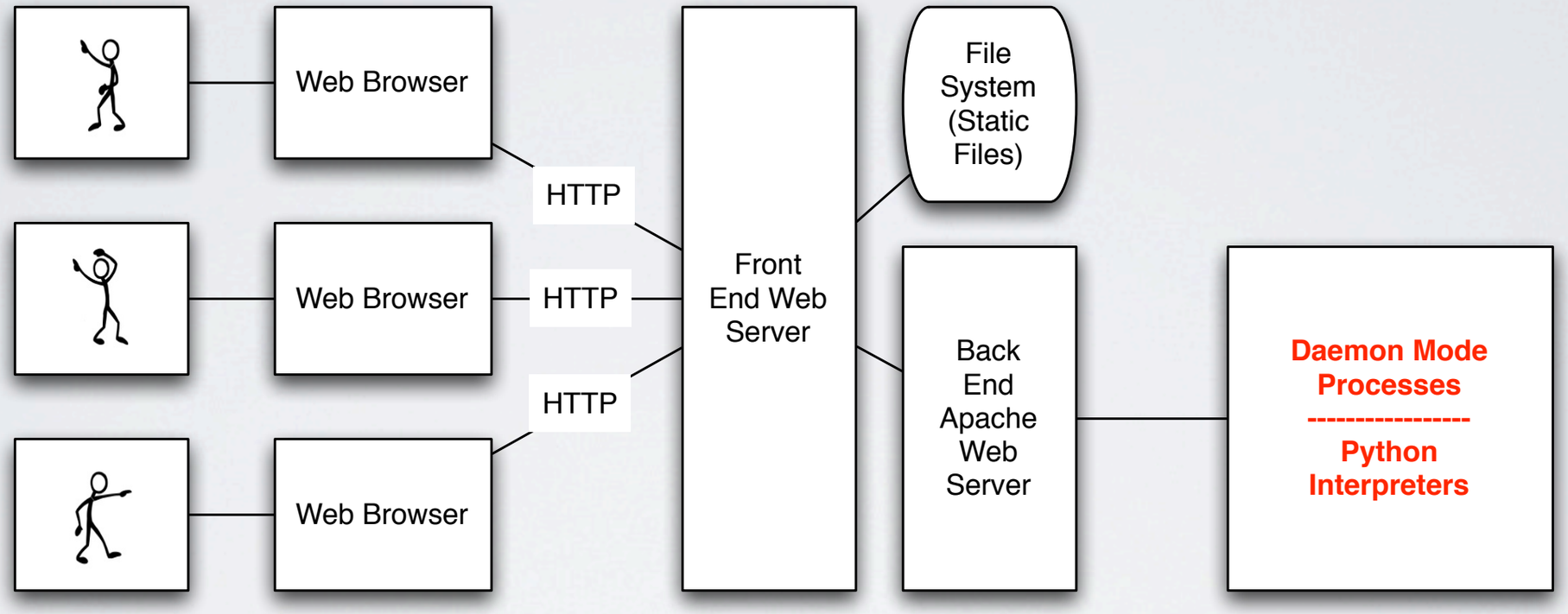
For embedded mode the Python interpreter running the WSGI application is inside of the web server process which are handling static file requests. Not recommended for single threaded prefork MPM where large number of processes as memory requirements much larger. Multi threaded worker MPM preferred. Adequate for most if worker MPM used even though static files handled as well.

DAEMON MODE



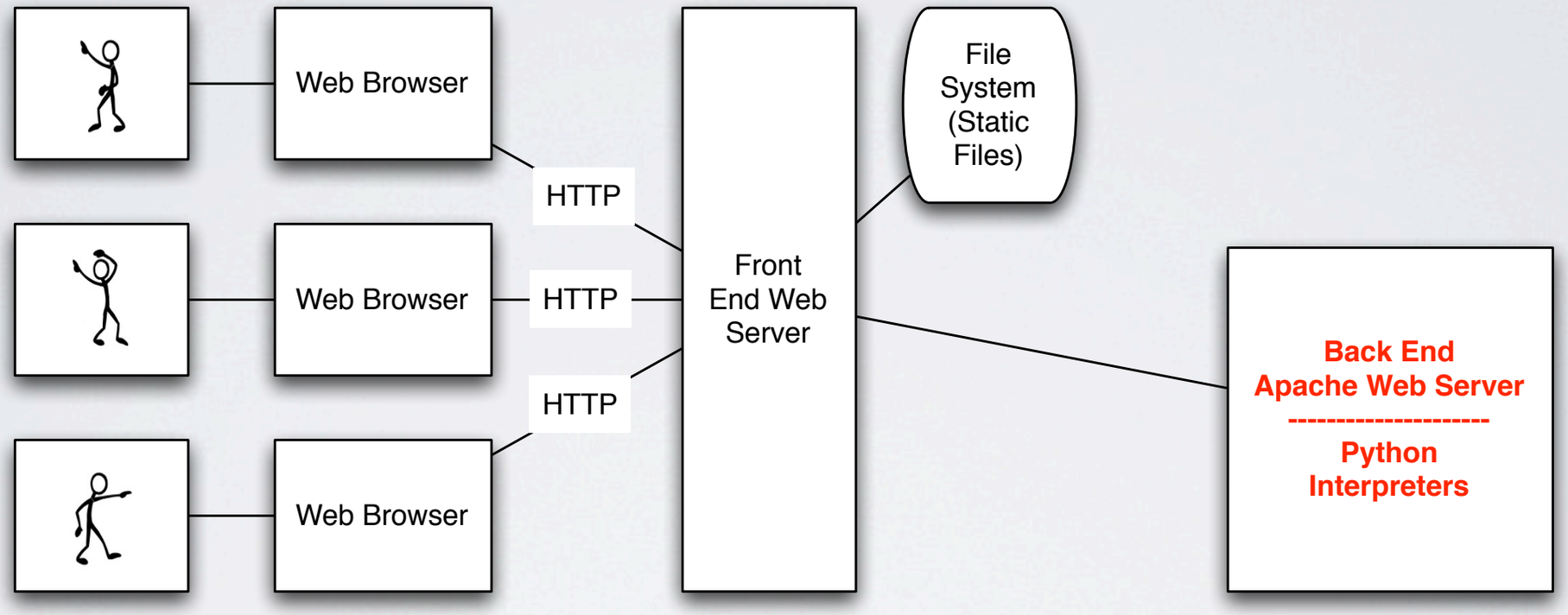
Effectively the same model as FASTCGI and uWSGI. Being Apache however, you don't get some of the benefits of using async front end web server. Still the best general purpose configuration for Apache/mod_wsgi when multithreading used as most peoples sites do not see enough traffic to warrant a front end async web server for static files anyway.

FRONT END + DAEMON



Popular upgrade path is to put a front end web server in front of Apache/mod_wsgi. Good solution if also using Apache for other purposes such as access control, authentication, authorisation or other languages such as PHP. If Apache only being used for the single Python web application, could instead use embedded mode with worker MPM.

FRONT END + EMBEDDED



Need to ensure that Apache is configured properly for running fat Python web applications. Start all processes up front and don't allow them to be killed off when idle. Preload WSGI application, don't allow lazy loading. Should only be used if you know how to configure Apache/mod_wsgi properly. Will be best performing solution when using Apache/mod_wsgi.

SO MANY OPTIONS

- HTTP proxy to gunicorn, paste server, CherryPy WSGI server.
- FASTCGI/SCGI/AJP
- uWSGI, Phusion Passenger, Mongrel
- Apache/mod_wsgi embedded mode.
- Apache/mod_wsgi daemon mode.
- HTTP proxy to Apache/mod_wsgi.

Bewildering array of options to choose from.

WHAT SHOULD YOU USE?

- The hosting mechanism is usually never the bottleneck.
- Don't believe hello world benchmarks.
- Don't believe what people tell you on IRC channels.
- Use what you have the skills to setup and configure.
- Use what you find has the qualities you need.
- Test the configuration you choose with real applications.

Sunday, 21 August 2011

Nearly any solution will work for most people because the hosting mechanism is not the bottleneck. Use what you feel comfortable being able to administer. Identify that different servers have different strengths and no one server may be the perfect solution. Custom Python async web server (non WSGI) may be the answer for parts of your application.

MOST POPULAR

- Apache/mod_wsgi (daemon mode).
- nginx HTTP proxy to Apache/mod_wsgi.
- nginx HTTP proxy to gunicorn.
- nginx to uWSGI.

Sunday, 21 August 2011

FASTCGI usually only gets used where people have no choice. Tendency is for people to use Apache/mod_wsgi, gunicorn and uWSGI. Apache/mod_wsgi is acknowledged as very stable and predictable, even if boring, but you may find Apache can be a pain to setup. Still hear occasional questions marks over reliability of uWSGI and to lesser extent gunicorn but they are improving all the time.

DON'T WASTE YOUR TIME

- Concentrate on removing bottlenecks.
- Look at improving your database performance.
- Look at implementing/improving page/data caching.
- Look at improving browser page rendering times.
- Look at monitoring tools such as New Relic and Munin.
- The hosting mechanism is only a small part of the problem.

Sunday, 21 August 2011

The hosting mechanism is only a small part of the problem. Don't prematurely optimise by trying to find what you think is the fastest hosting mechanism. Your application/database overhead will dwarf any benefits one server may have over another. Tuning your application gives benefits for any solution. Use production monitoring to know what is going on.

QUESTIONS?