

Pylons

Python web programming

Ben Bangert
Senior Web Architect
Stanford Law School IPLC



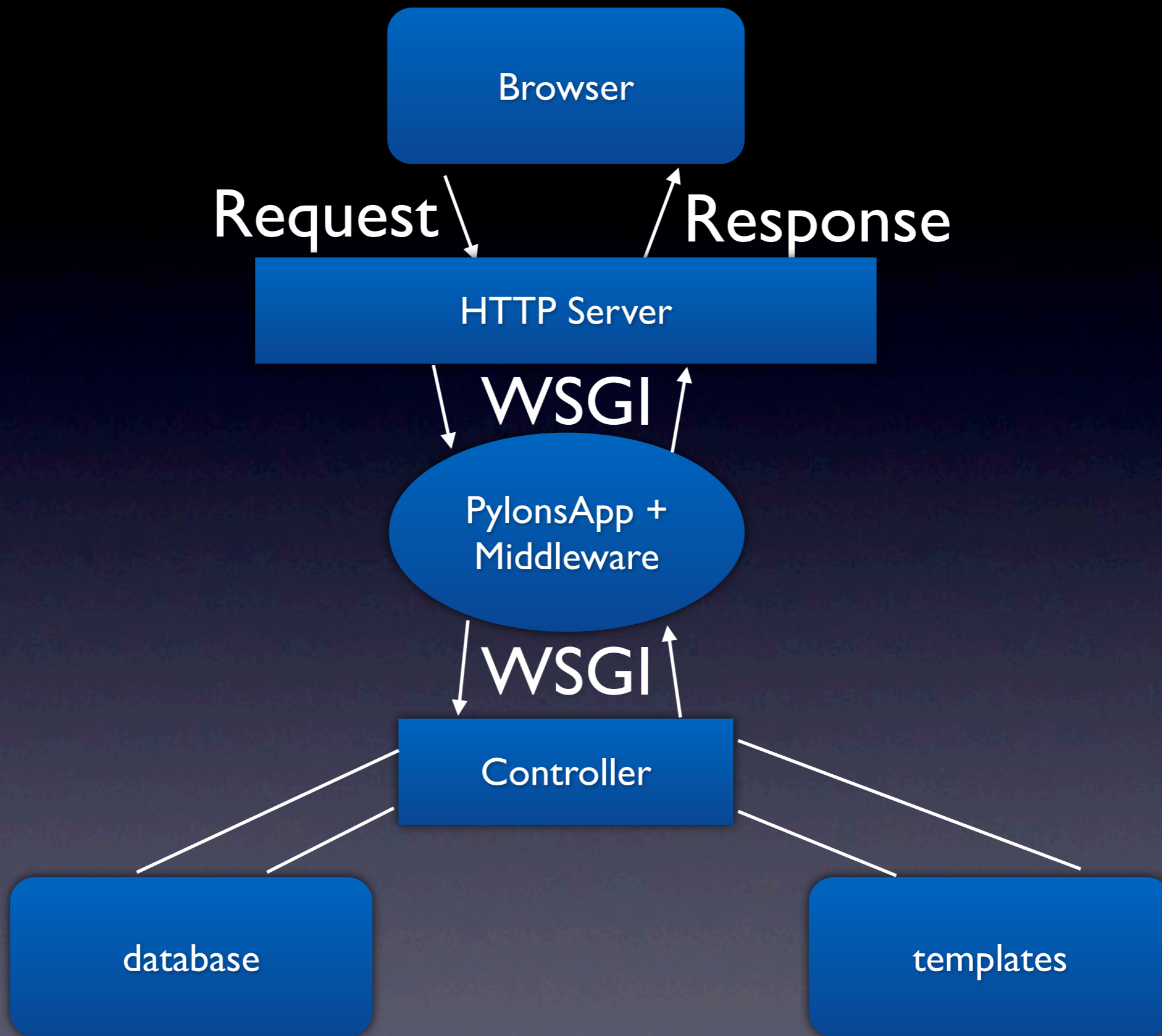
Background

- I create and maintain web applications
- Many frameworks optimize creation over maintaining
- Long lived applications are more unique and custom-tailored than frameworks usually account for
- Want to retain a toolset

Pylons History

- First version in 2004
- Second re-write / extraction in 2005
- First public release in late 2005
- 0.9 Series in 2006
- 0.9.7 in 2008
- 1.0 in 2009?

Internal Architecture



WSGI

- PEP 333
- Web Server Gateway Interface
- Brief description of how to handle a single request, and how the response should be provided
- Styled on CGI
- Originally designed to be a uniform interface for a Python web application

A simple WSGI app

```
def wsgiapp(environ, start_response):  
    start_response('200 OK',  
                  [('Content-Type', 'text/html')])  
    return ["Hello world!"]
```

Looking in environ

```
>>> print environ
{'HTTP_HOST': 'localhost',
 'PATH_INFO': '/',
 'QUERY_STRING': '',
 'REQUEST_METHOD': 'GET',
 'SCRIPT_NAME': '',
 'SERVER_NAME': 'localhost',
 'SERVER_PORT': '80',
 'SERVER_PROTOCOL': 'HTTP/1.0',
 ...}
```

Running it with wsgiref

```
from wsgiref.simple_server import make_server

def wsgiapp(environ, start_response):
    start_response('200 OK',
        [('Content-Type', 'text/html')])
    return ["Hello world!"]

httpd = make_server('', 8000, wsgiapp)
httpd.serve_forever()
```

Running it with wsgiref

```
from wsgiref.simple_server import make_server

def wsgiapp(environ, start_response):
    start_response('200 OK',
        [('Content-Type', 'text/html')])
    return ["Hello world!"]

httpd = make_server('', 8000, wsgiapp)
httpd.serve_forever()
```

Using a class for a WSGI app

```
class WSGIApp(object):
    def __init__(self, options):
        self.options = options

    def __call__(self, environ, start_response):
        # do something about self.options?
        start_response('200 OK',
            [('Content-Type', 'text/html')])
        return ['Bigger app!']
```

Demo of Pylons request dispatch w/ WSGI

Expert Tips

Exploit the tools

- Mako
 - Inheritance
 - Call with blocks (Ruby anonymous blocks?!?!)
 - Inheritance chaining

Inheritance

/article/show.html

Show the article...

```
<%def name="title()">${c.article.title}
</%def>
<%inherit file="/layout.html" />
```

/layout.html

```
<title>${self.title()}</title>
```

...

```
${self.body()}
```

...

```
<%def name="title()">Home</%def>
```

Inheritance Chains

```
/article/show.html
```

Show the article...

```
<%def name="title()">${parent.title  
()} - ${c.article.title}</%def>  
<%inherit file="layout.html" />
```

```
/article/layout.html
```

```
${next.body()}  
<%def name="title()">${parent.title  
()} - Article</%def>  
<%inherit file="/layout.html" />
```

```
/layout.html
```

```
<title>${self.title()}</title>  
...  
${next.body()}  
...  
<%def name="title()">Home</%def>
```

- SQLAlchemy
 - Keep table objects organized
 - Decide where mappers are based on imports (recursive imports are annoying!)
 - Utilize latest SA ORM queries with attribute queries

- Routes
 - Conditional functions for ugly legacy integration
 - Dict overloading for Pylons dispatch customization
 - Filter functions for generation altering
 - Always use named routes!