Ebb Ebb Ebb Ebb

ry@tinyclouds.org
/ry(a(n|h))? dahl/i

RuPy 2008

I wrote a web server called **Ebb**.

yawn

Almost as boring as another web framework, you think.

Many Ruby-HTTP interfaces exist already

FastCGI Thin Webrick Mongrel Evented Mongrel

Why write another?

Ebb is faster.

Ebb has cute features.

Ebb has different goals than the other servers.

Ebb is faster

because it's implemented in C it can perform 1.5× (on average) faster than the others.

Warning: Benchmarks follow.

Believe them at your own risk.

But I'll try to be honest about it...

Raw data and scripts can be provided upon request.

Created by averaging 40 trials of Apache Bench with 3 seconds of connections per trial.

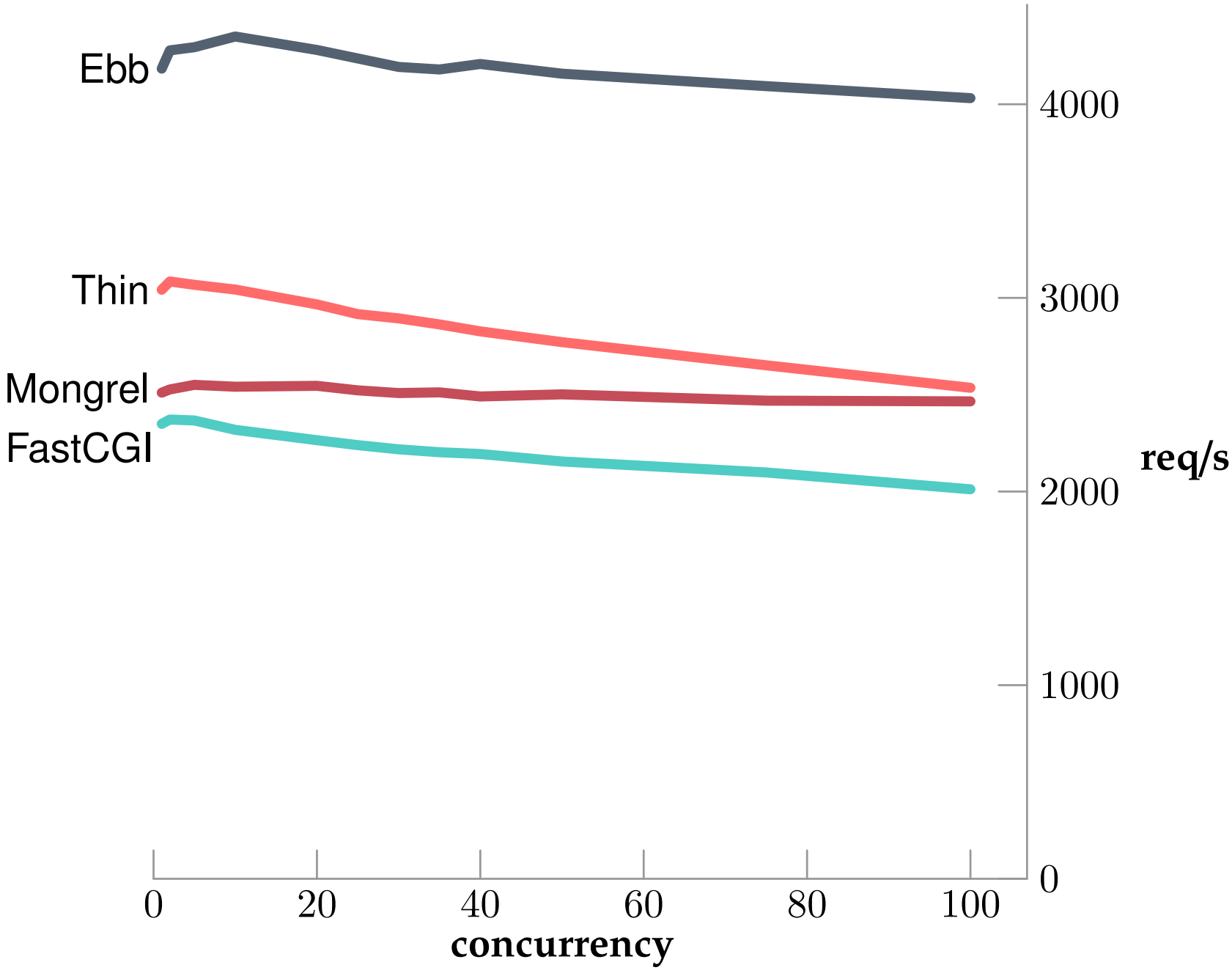
Standard deviation is < 250 req/sec.

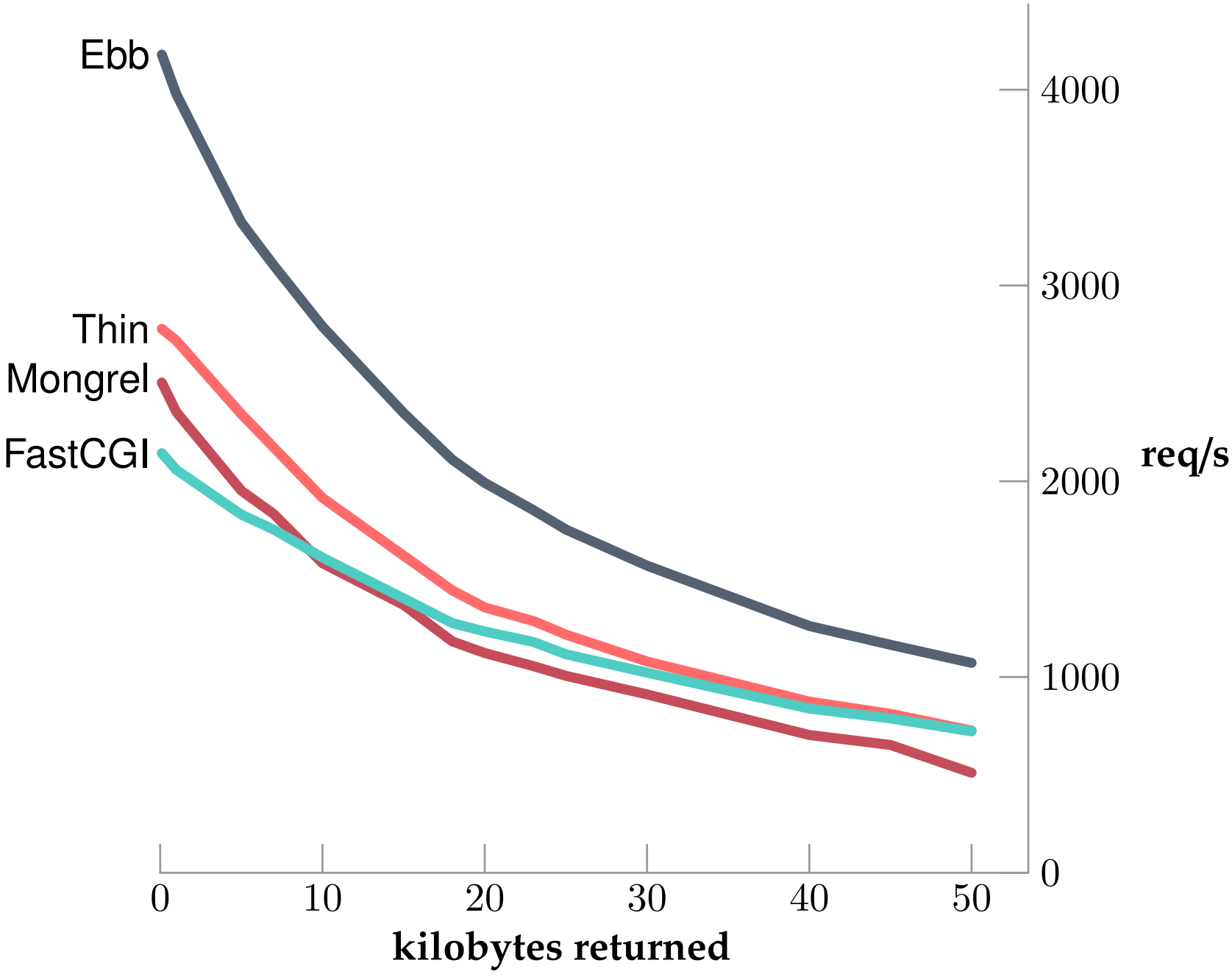
Mongrel version 1.0.1

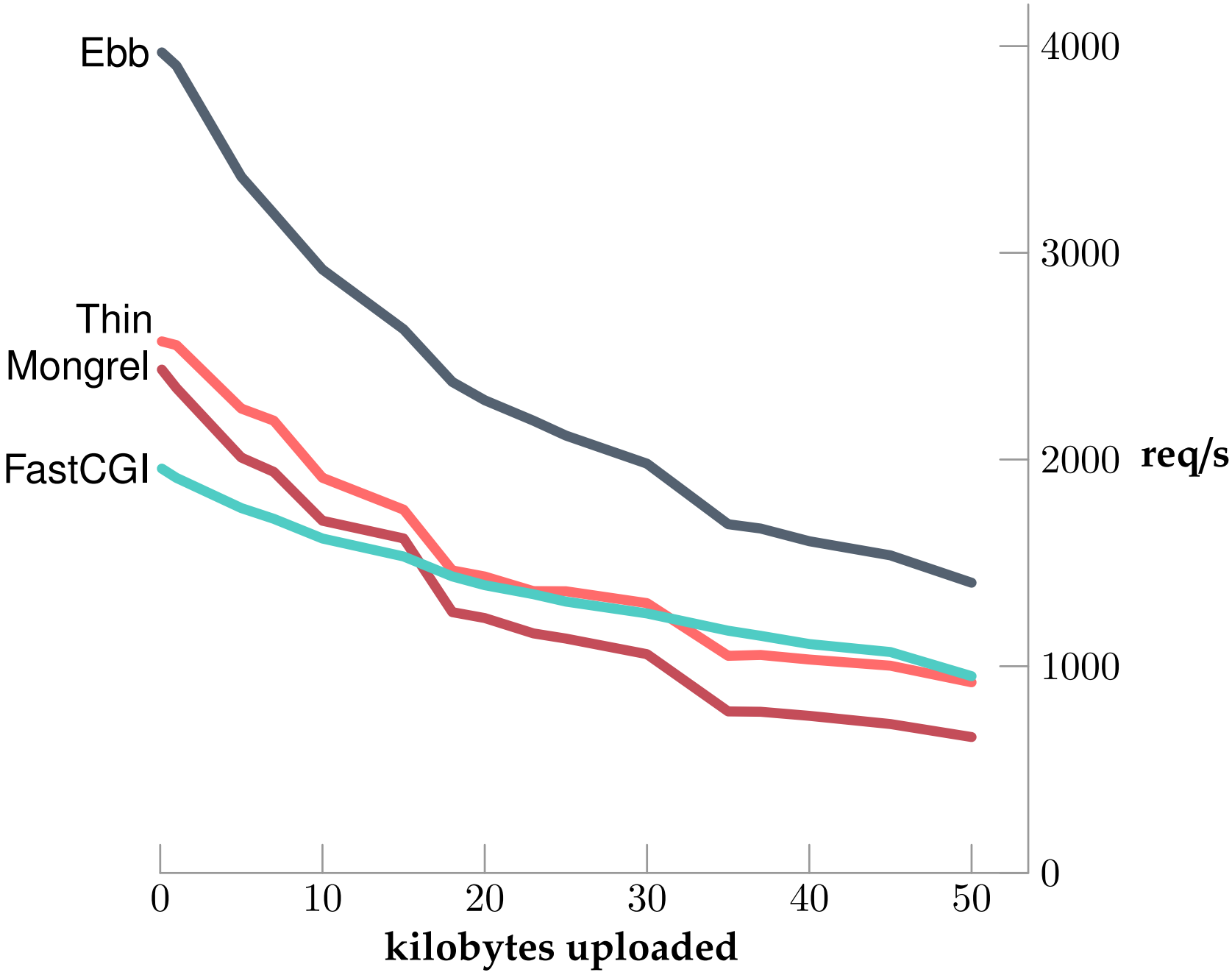
FastCGI is one instance of Rack's FastCGI handler running behind Lighttpd 1.4.

Thin version 0.7.0

Ebb version 0.1.0 (with one thread per request)


Ebb
Thin
Mongrel
FastCGI
4000
3000
2000
1000
0
req/s
0
20
40
60
80
100
concurrency


Ebb
Thin
Mongrel
FastCGI
4000
3000
2000
1000
0
req/s
0
10
20
30
40
50
kilobytes returned


Ebb
Thin
Mongrel
FastCGI
4000
3000
2000
1000
0
req/s
0
10
20
30
40
50
kilobytes uploaded

Ebb has cute features.

Cute Feature 1:

Ebb can handle requests asynchronously.

(more about this later)

Cute Feature 2:

Ebb supports `Keep-Alive`.

Cute Feature 3:

Ebb can listen on

- localhost TCP ports
- UNIX domain sockets
- arbitrary file descriptors

Ebb has different goals.

Ebb is written in C with a binding to Ruby.

The core of **Ebb** is independent of Ruby.

Ebb's goal is to be a little C library for creating little web servers.

Little because the core is quite simple: only ~ 1000 lines of C.

(For comparison **Mongrel** is ~ 2000 lines of Ruby)

The **Ebb** C library uses three libraries

- libev
- Mongrel's HTTP parser
- glib

libev is a new event loop library aimed at replacing libevent.

- less memory usage
- simpler API
- easy to including in projects
- trendy

Zed Shaw's **HTTP parser** is written in Ragel and C.

It's a state machine for the HTTP protocol.

It's fast, secure, and easy to use.

glib is only used for buffers.

I might replace it in the future.

Quickly I will sketch how to use the C library.

Initialize `ebb_server` with an `ev_loop` and a request callback

```
void request_cb(ebb_client*, void*)
```

The request callback is called on each request with a new `ebb_client`

`ebb_client` contains a `Hash` like structure containing the request parameters.

Then use

```
ebb_client_write_status()

ebb_client_write_header()

ebb_client_write_body()

ebb_client_release()
```

About the Ruby binding...

Ebb's Ruby binding implements a Rack interface.

A Rack interface just means that someone implements `YourApplication` which gets passed to **Ebb**.

```
app = YourApplication.new()
Ebb.start_server(app)
```

`app` has a method which gets called on each request called `:call`

```
app.call(env)
```

(Where `env` is the request parameter `Hash`)

`:call` must return a three element array.

The first element is the HTTP status code of the response.

The second is a Hash containing the response headers.

The third is the body of the response.

Ebb's Ruby binding checks for another method on your application called `:deferred?`

For each request, `:deferred?` is called with `env` and must return `true` or `false`.

If `true`, then **Ebb** will spawn a new thread for that request. If `false`, the request is processed immediately.

Normally you want to process requests immediately because thread creation and context switching are very slow in Ruby.

However if you have an action or two that responds slowly, you might want to handle it in a thread to allow **Ebb** to process other clients concurrently.

Example:

```
class YourApplication

  def call(env)
    sleep 10 if env["PATH_INFO"] == "/slow"

    [200, {"Content-Type"=>"text/plain"}, "Hey"]
  end

  def deferred?(env)
    env["PATH_INFO"] == "/slow"
  end
end
```

`:deferred?` allows for the benefits of both evented and threaded servers.

Aside

Mixing MRI's event loop and an external event loop while processing threads is not easy.

Aside

I've learned that `rb_thread_schedule()` is significantly faster than `rb_thread_select()`.

(Why? No one knows.)

Aside

YARV is much better because it has `rb_thread_blocking_region()` which makes these problems go away.

Aside

See **Ebb**'s `idle_cb()` for details.

The **Ebb** Python binding is 50% complete.

The **Ebb** Python binding implements both WSGI 1 and WSGI 2 interfaces.

(WSGI is the equivalent of Rack in the Python world)

Initial measurements show the Python binding being much faster. Perhaps twice as fast as the Ruby binding.

I don’t use Python–so if anyone wants to lend a hand in this effort, let me know :)

Also–there have been suggestions of Perl, Lua, and Erlang bindings!

I would like to see **Ebb**'s C library used

- in embedded devices (like wifi routers)
- or for daemons that need little HTTP servers (like Monit)
- or for a hardcore C-language web framework? :)

In the future, Ebb-C might collect a small set of often used utility functions:

- `multipart/form-data` parsing
- URL escaping
- URL Mapping (by reviving `Mongrel::URIClassifier`)

Version 0.2 of the Ruby binding will be the first “Beta” release. It will be out next week.

First version of the Python binding out in the next month.

Here are some links!

http://github.com/ry/ebb/tree/master

http://groups.google.com/group/ebbebb

http://ebb.rubyforge.org/