Erlang Web Development with Yaws

Steve Vinoski
Basho Technologies
Cambridge, MA USA
http://basho.com
vinoski@ieee.org
http://steve.vinoski.net/
@stevevinoski

Claes "Klacke" Wikström

Erlang bit syntax

Erlang bit syntax

Distributed Erlang

Erlang bit syntax

eprof

Distributed Erlang

Erlang bit syntax

eprof

ASN.1 compiler

Distributed Erlang

Erlang bit syntax

ets

eprof

ASN.1 compiler

Distributed Erlang

Erlang bit syntax

ets

dets

eprof

ASN.1 compiler

Distributed Erlang

Erlang bit syntax

ets

dets

Mnesia

eprof

ASN.1 compiler

Distributed Erlang

Erlang bit syntax

ets

dets

eprof

Mnesia

ASN.1 compiler

Distributed Erlang

2001

2001

“I was absolutely struck with horror when I finally realized how horrible the LAMP stack was, and in particular the ugliness of the PHP language.”

—Klacke

(recounting web development tools available in 2001)

Yet another web server

Yaws

Yaws Features

Embedded or Stand-alone

Reliability & Stability

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file uploads
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compile .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compile .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs
- file upload
- multipart/mime support

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compile .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods

- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs

- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- SSL support
- cookie/session support

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats

- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs

- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support

- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- CGI and FCGI
- PHP handling via FCGI

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs
- forward and reverse proxies

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- **munin stats**
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- **heart integration**
- both autoconf and rebar builds
- **logging in Apache common format**
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- logging in Apache common format
- munin stats
- heart integration
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- both autoconf and rebar builds
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- **munin stats**
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- **heart integration**
- both autoconf and rebar builds
- **logging in Apache common format**
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- precompressed static files
- JIT-compiled .yaws pages
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP scripting via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- virtual directories
- virtual servers
- configurable MIME types
- configurable deflate
- ACLs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

- HTTP 1.1
- URL/#arg rewriting
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- virtual servers
- yapps
- GET/POST chunked transfer
- multipart/mime support
- PHP handling via FCGI
- server side includes
- heart integration
- pool and conf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

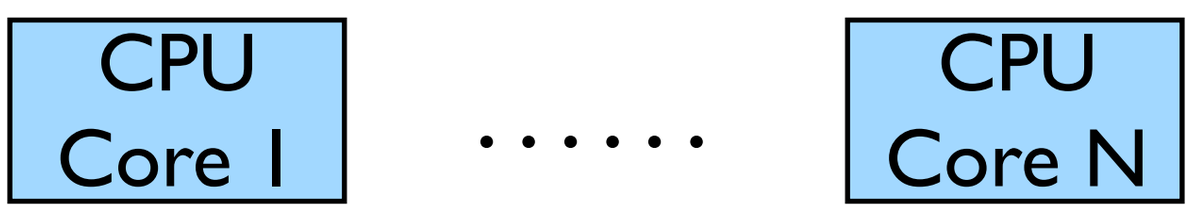
- JIT-compiled .yaws pages
- ehtml and exhtml
- JSON and JSON-RPC 2.0
- appmods
- streaming
- Server-Sent Events
- WebSocket support

- HTTP 1.1
- URL/#arg rewriting
- appmods
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- forward and reverse proxies
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- ehtml and exhtml
- virtual directories
- configurable deflate
- ACLs
- precompressed static files
- configurable MIME types
- JIT-compiled .yaws pages
- virtual servers
- yapps
- JSON and JSON-RPC 2.0
- Server-Sent Events
- WebSocket support
- GET/POST chunked transfer
- streaming
- multipart/mime support
- PHP handling via FCGI
- server-side includes
- heart integration
- both autoconf and rebar builds
- logging in Apache common format
- man pages and LaTex/PDF docs

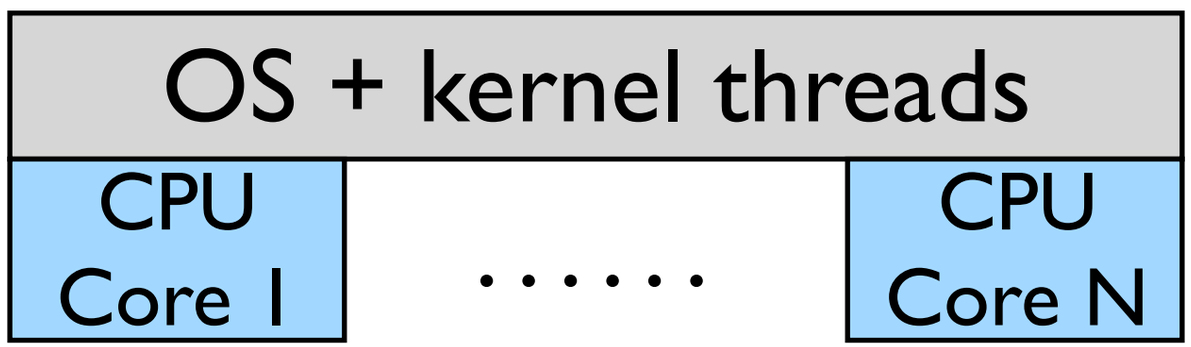
A Simple Erlang Web Server

Erlang Process Architecture

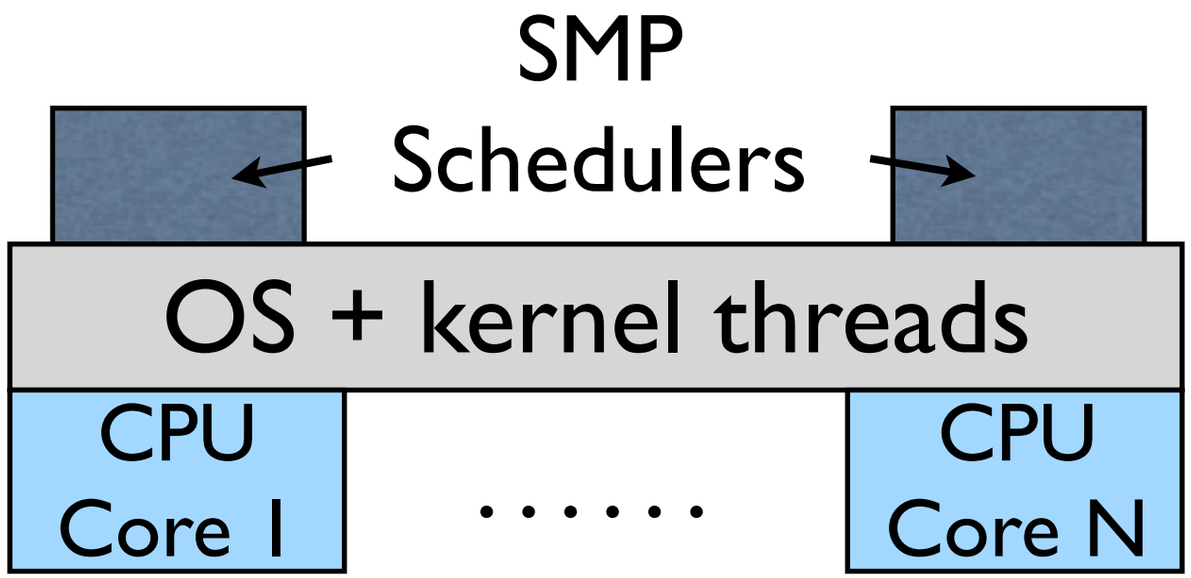
Erlang Process Architecture



Erlang Process Architecture

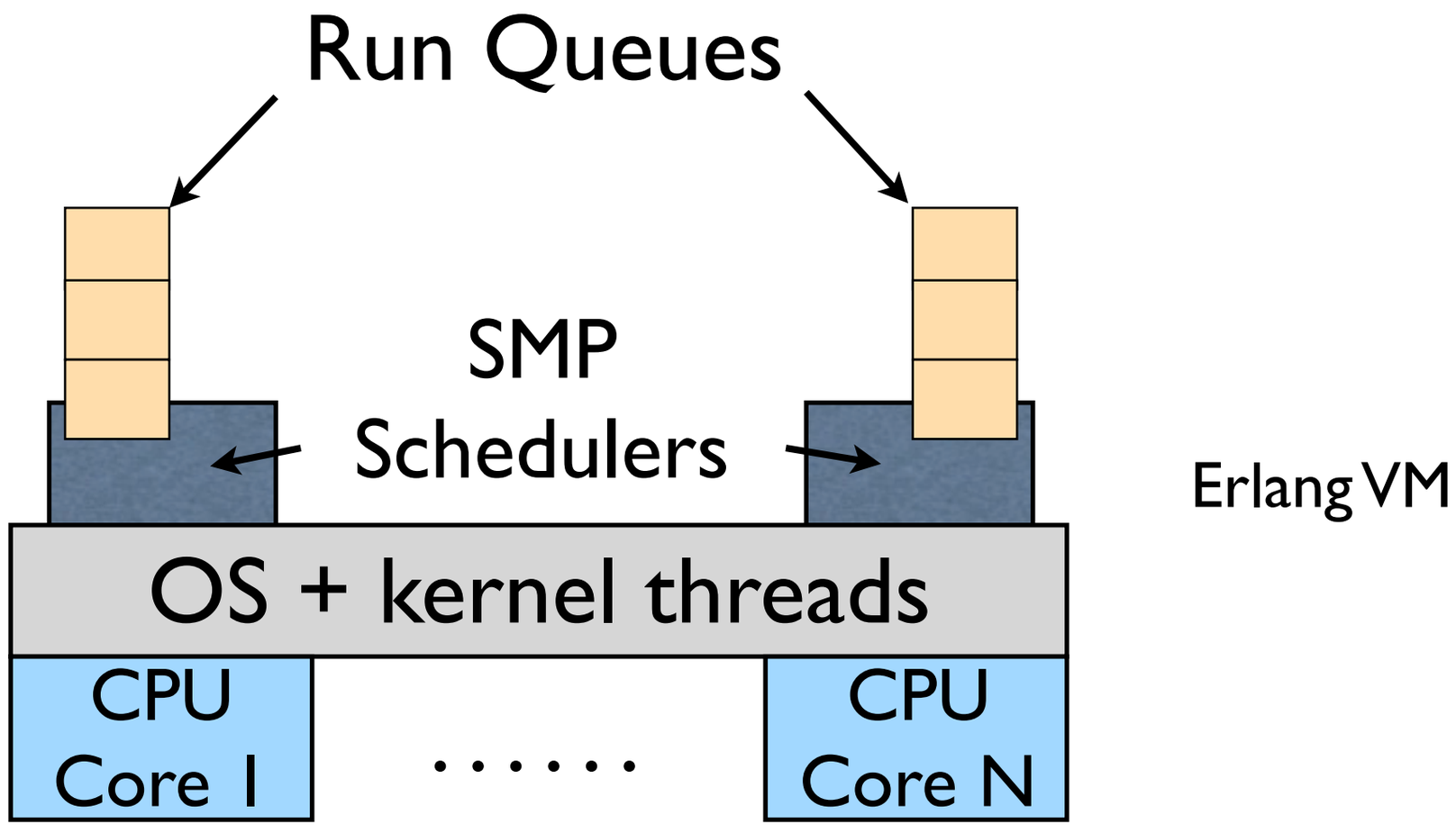


Erlang Process Architecture

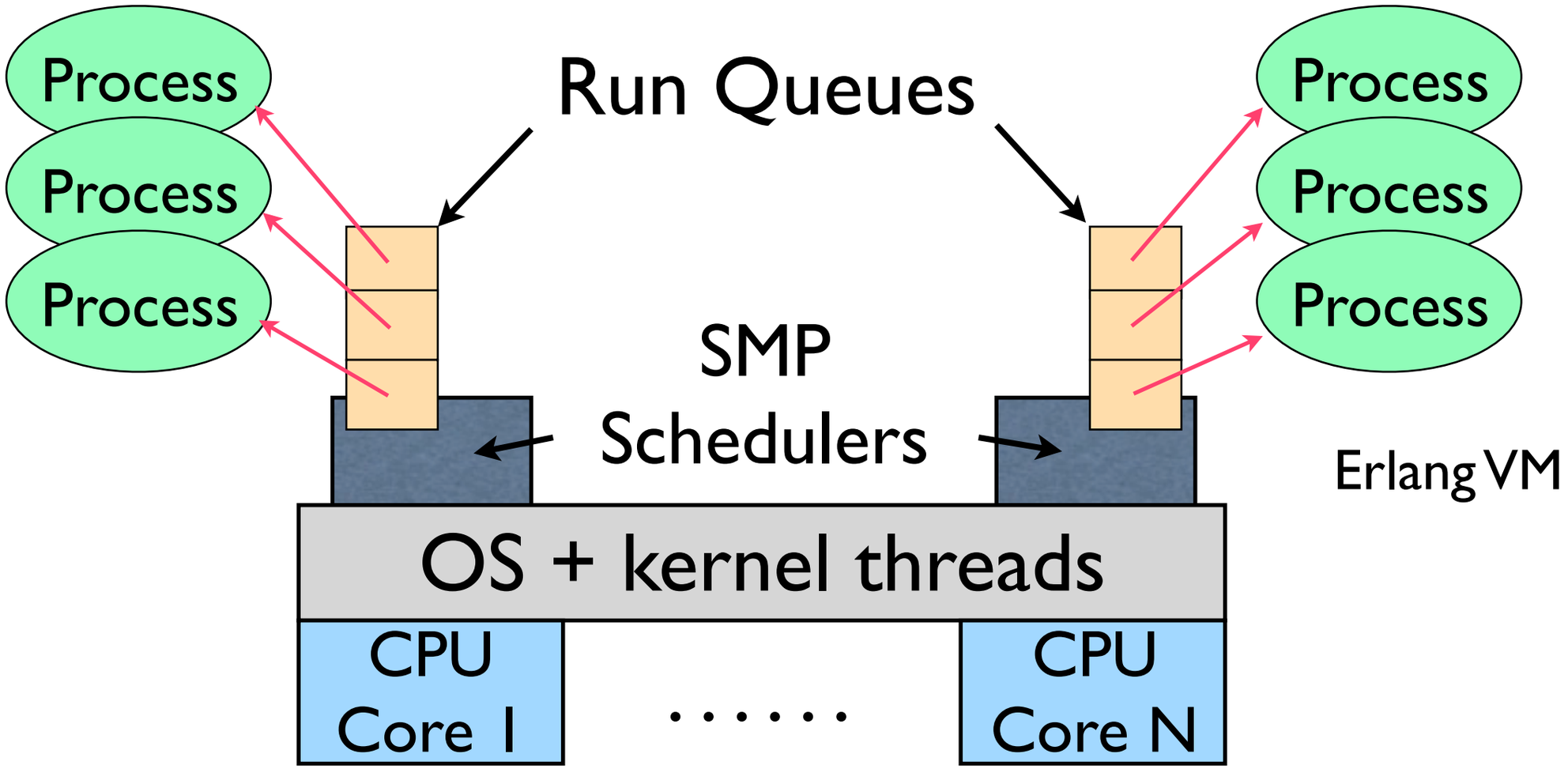


Erlang VM

Erlang Process Architecture



Erlang Process Architecture



A Simple Erlang Web Server

A Simple Erlang Web Server

1. Open a listen socket

A Simple Erlang Web Server

1. Open a listen socket
2. Spawn a new Erlang process to accept connections

A Simple Erlang Web Server

1. Open a listen socket
2. Spawn a new Erlang process to accept connections

1. Accept connection

A Simple Erlang Web Server

1. Open a listen socket
2. Spawn a new Erlang process to accept connections

1. Accept connection
2. Spawn a new acceptor process

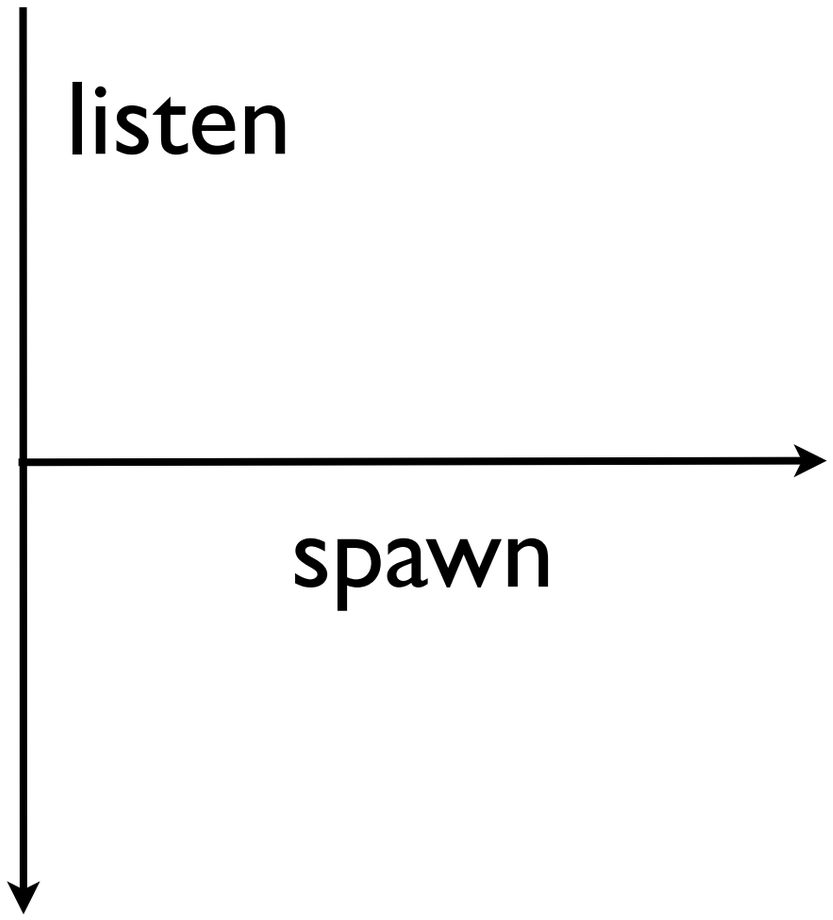
A Simple Erlang Web Server

1. Open a listen socket
2. Spawn a new Erlang process to accept connections

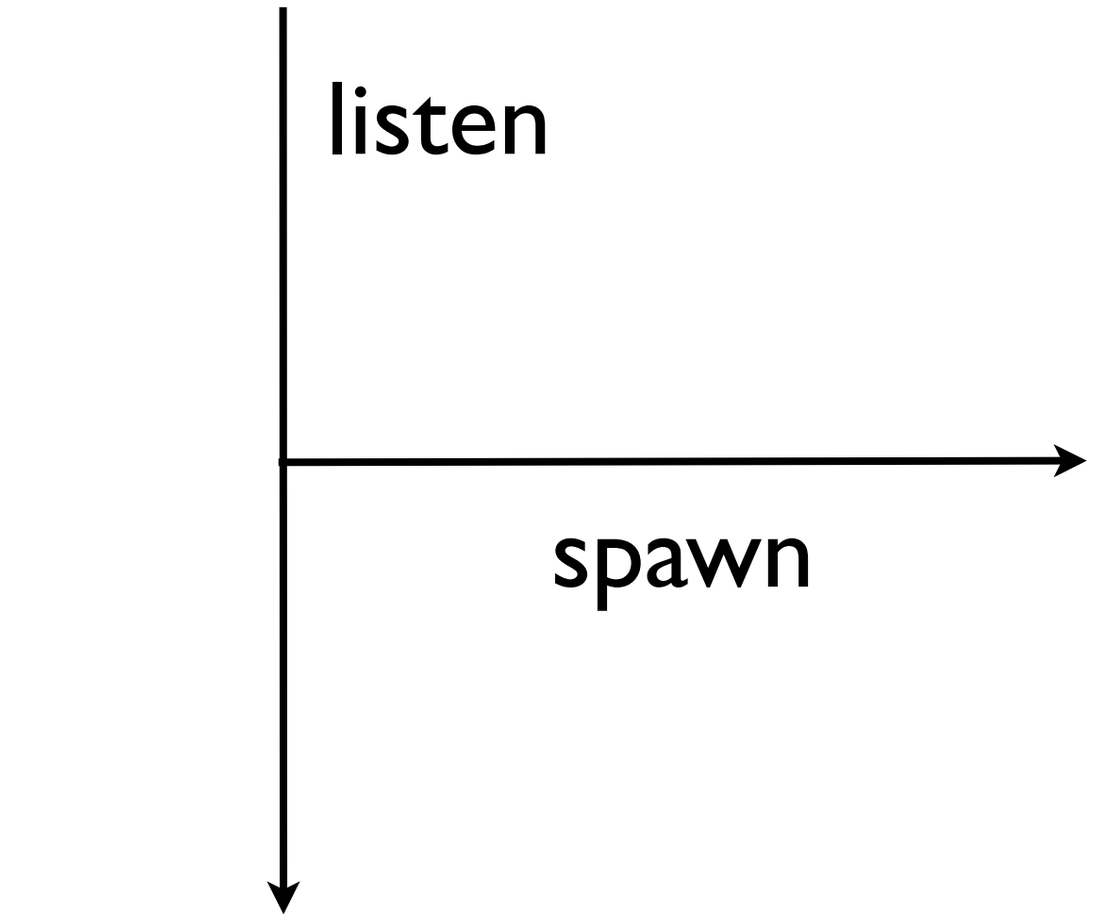
1. Accept connection
2. Spawn a new acceptor process
3. Handle HTTP request

A Simple Erlang Web Server

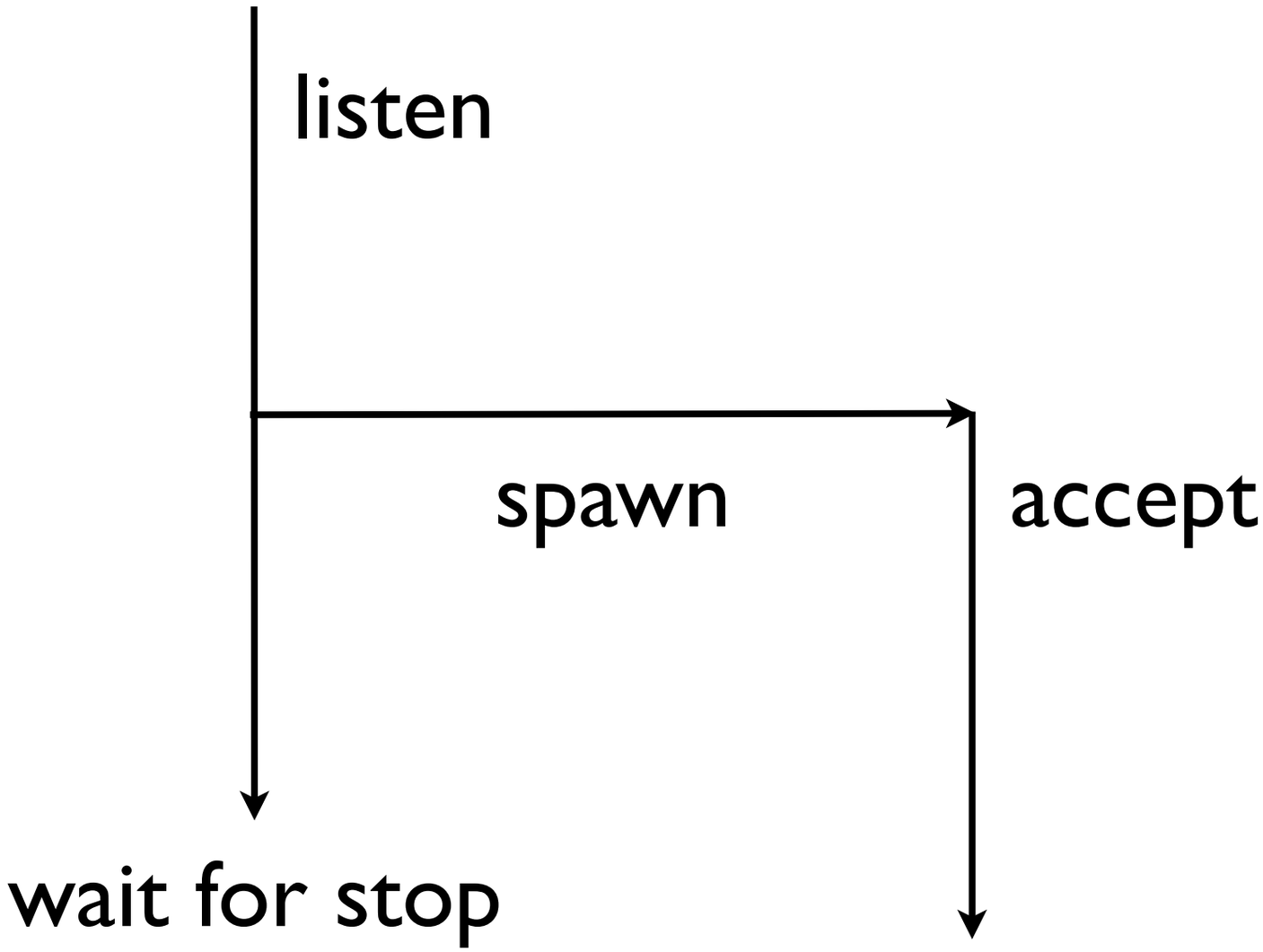
A Simple Erlang Web Server



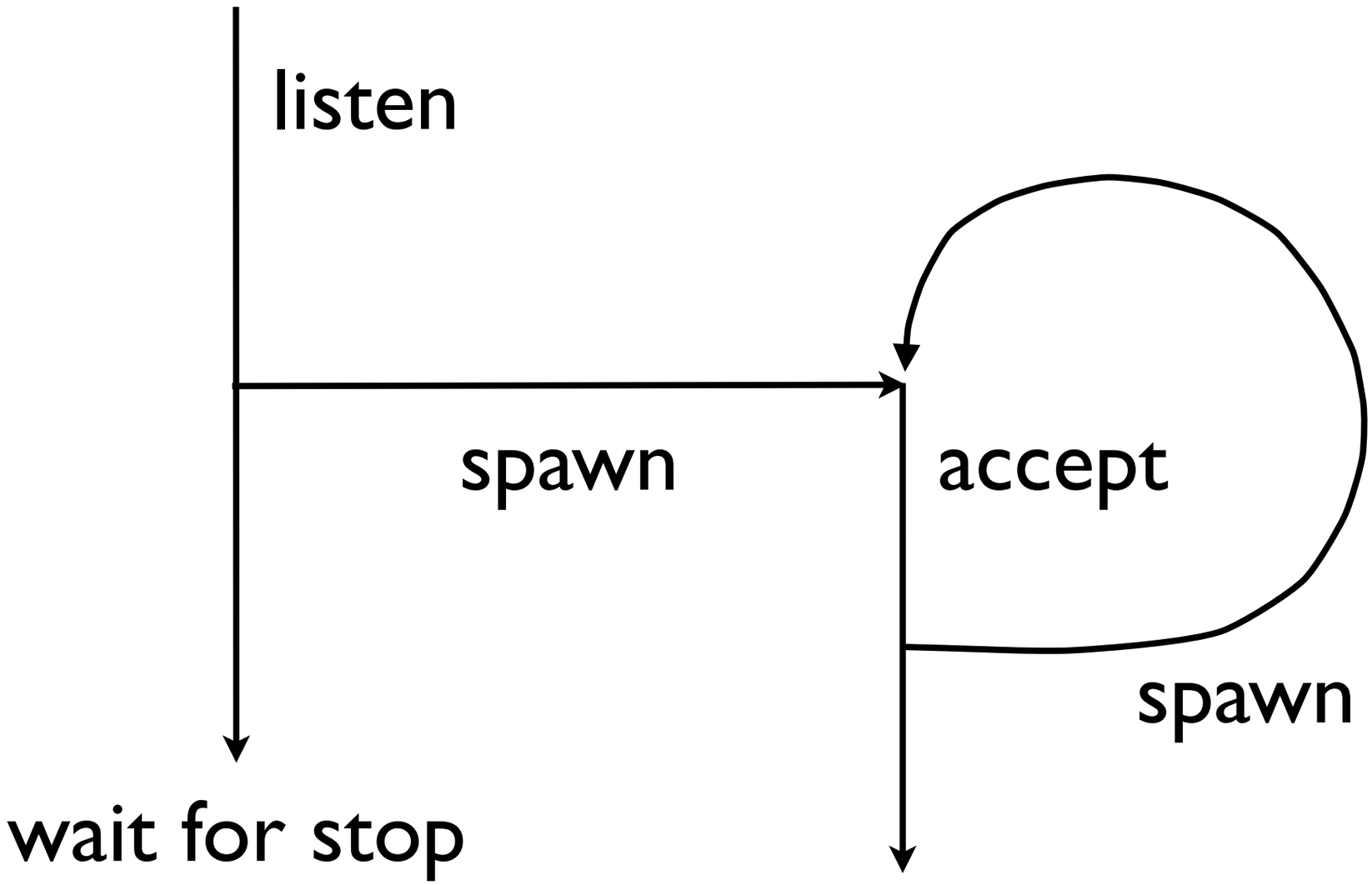
A Simple Erlang Web Server



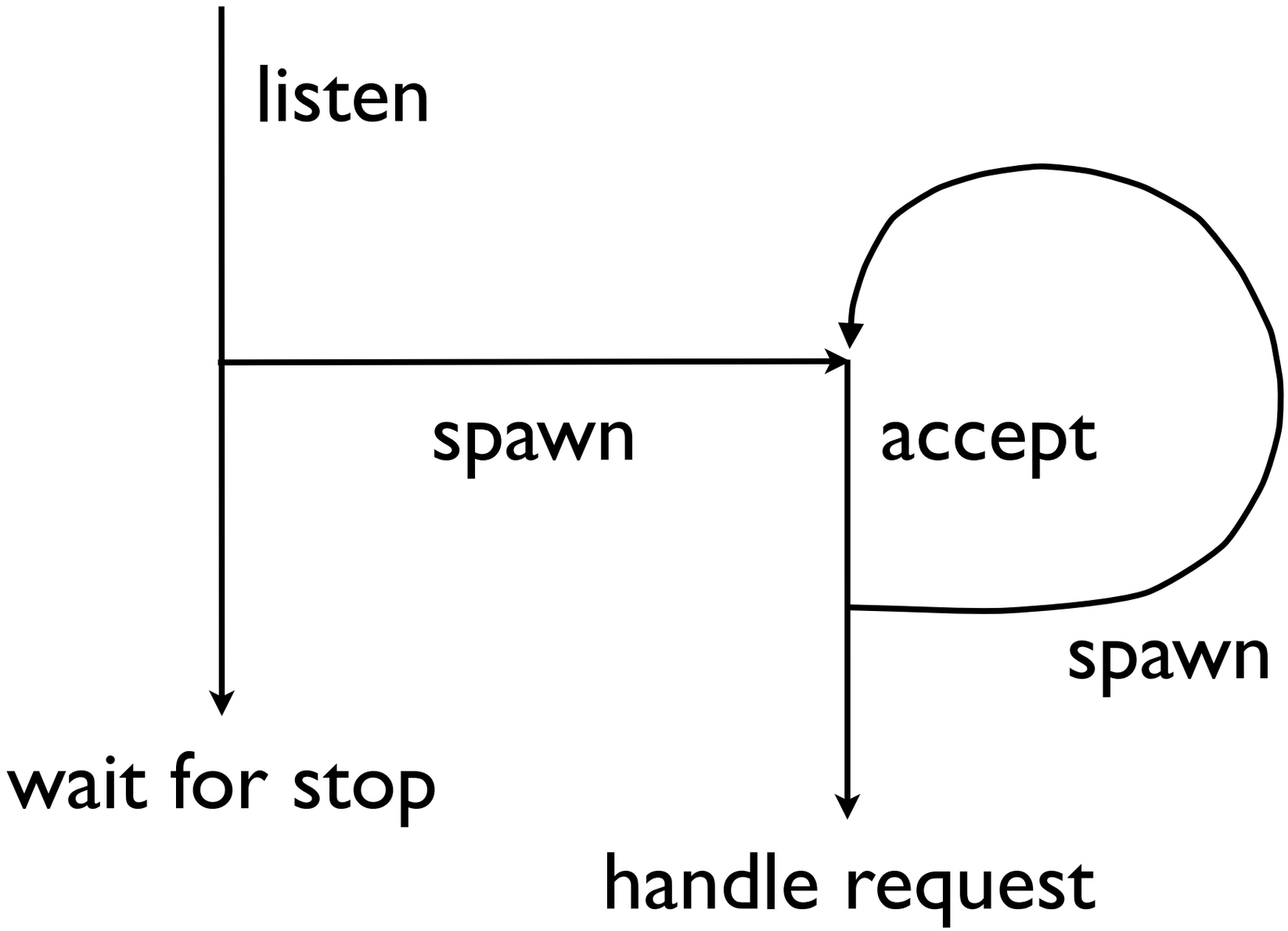
A Simple Erlang Web Server



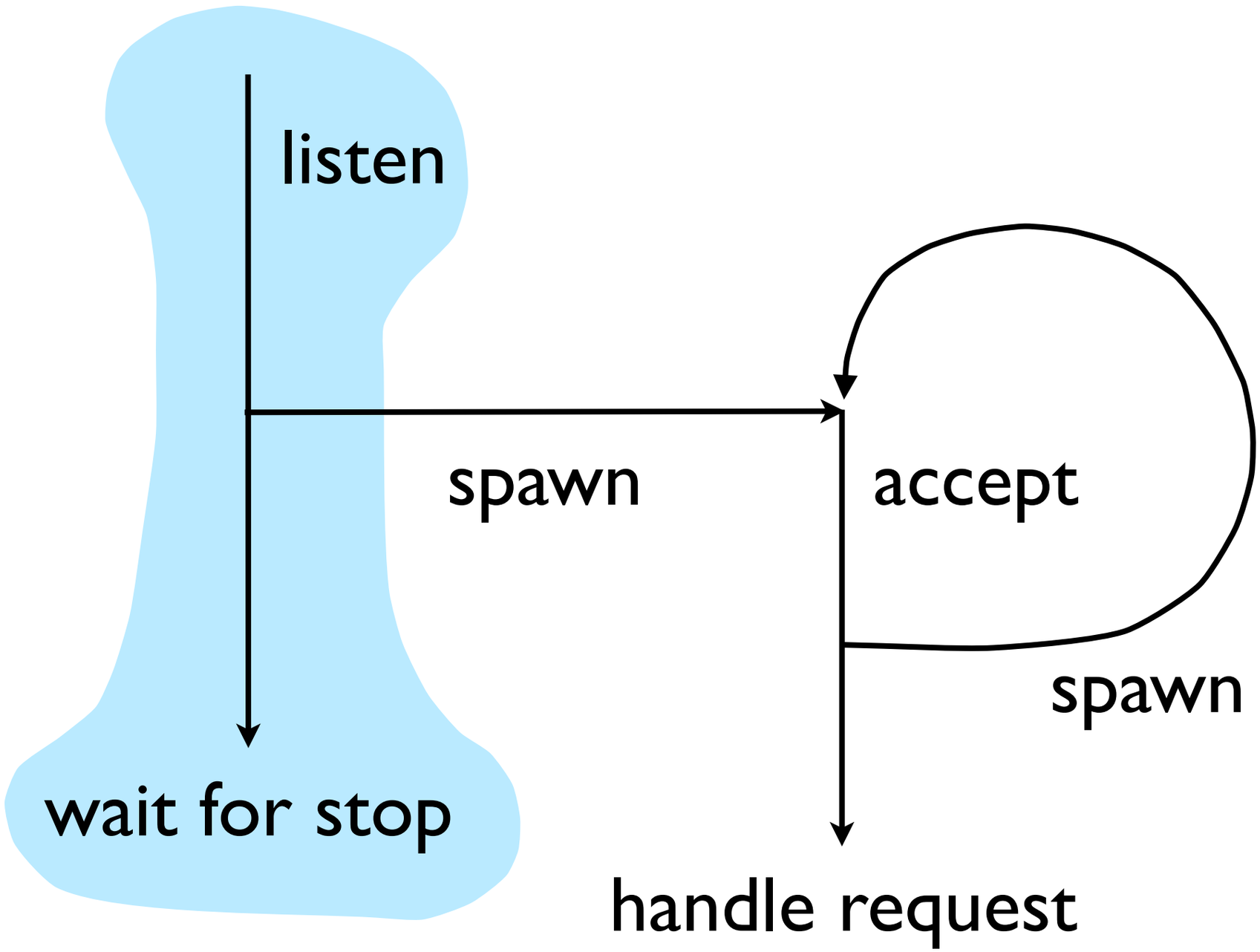
A Simple Erlang Web Server



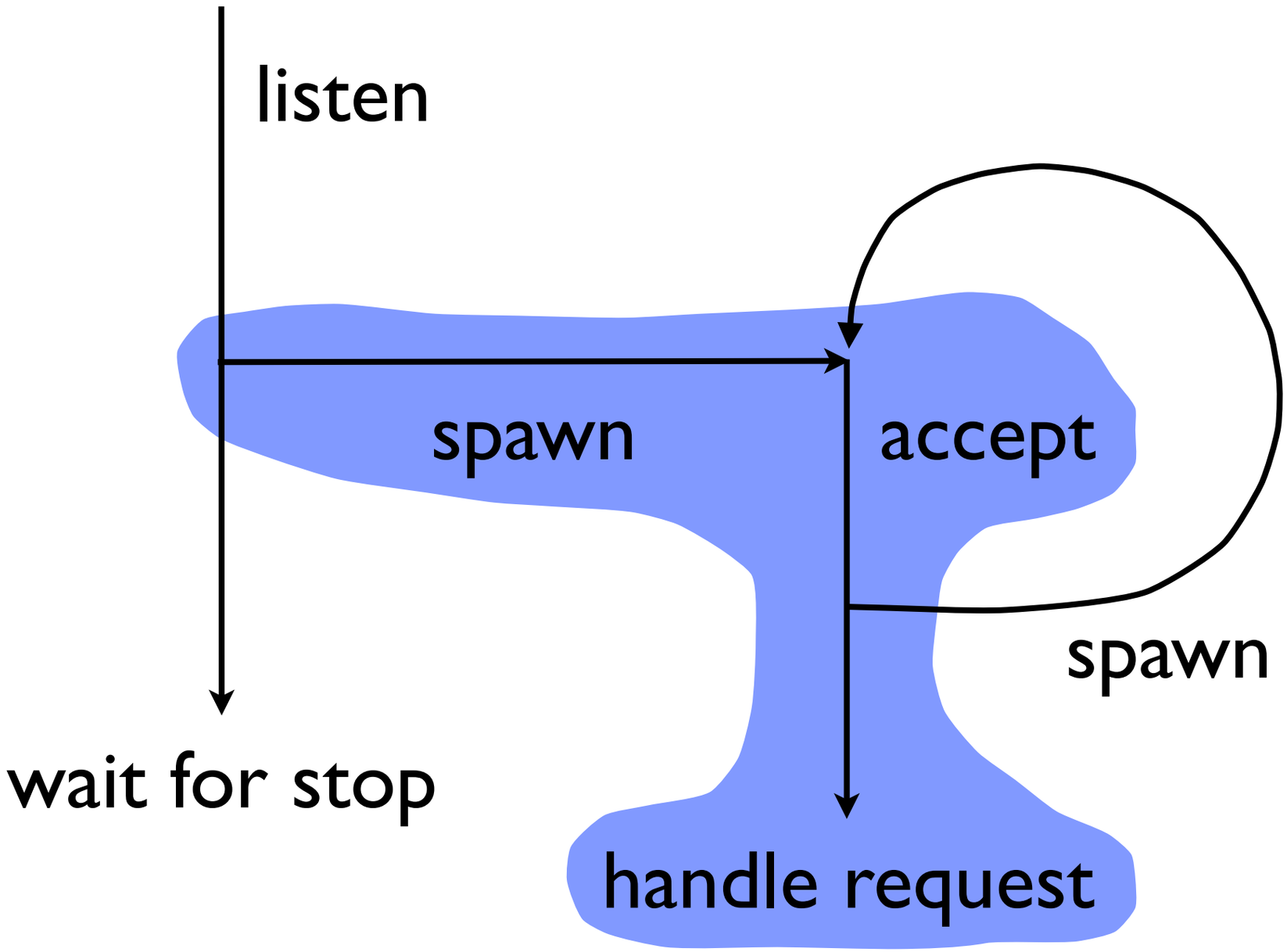
A Simple Erlang Web Server



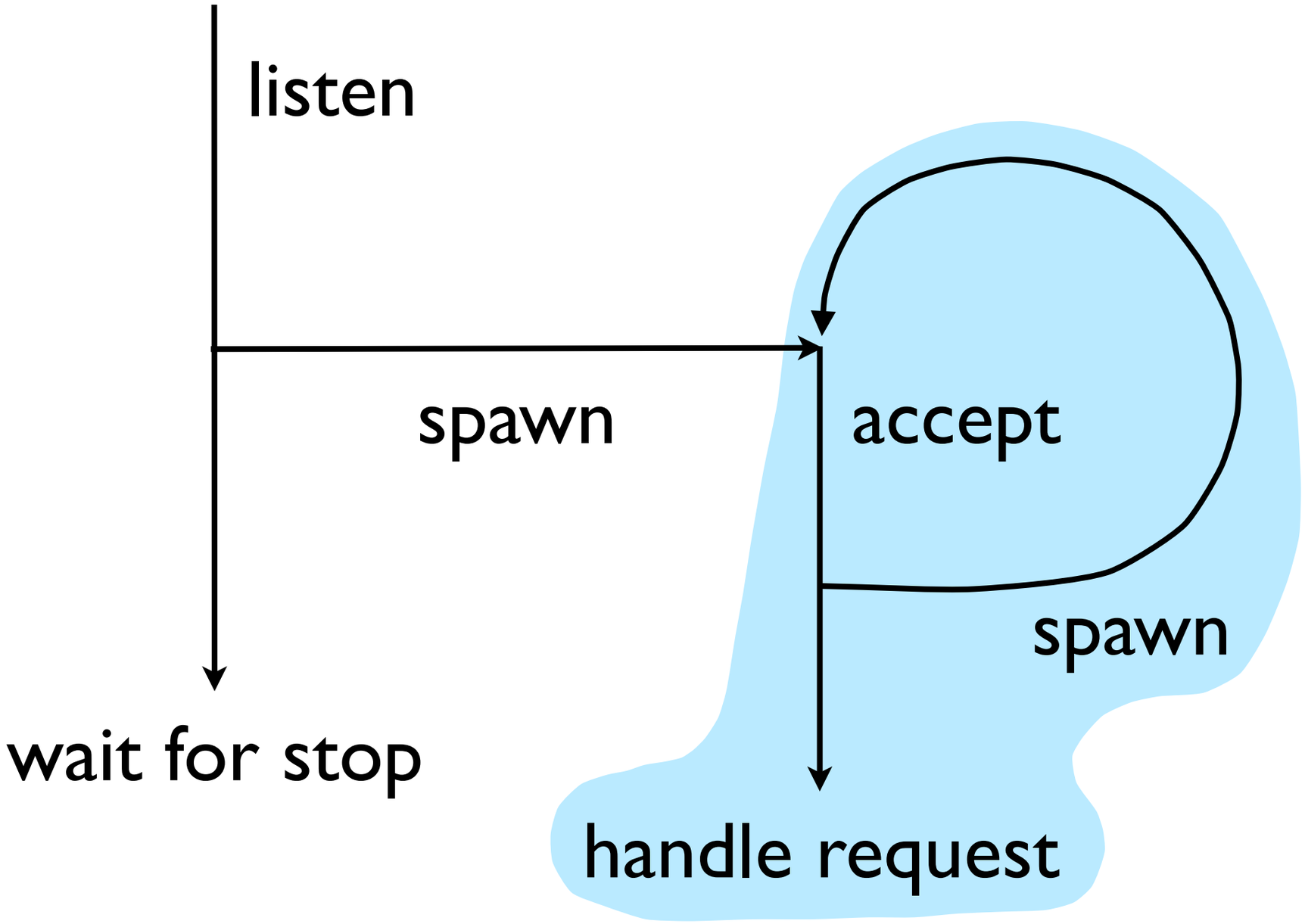
A Simple Erlang Web Server



A Simple Erlang Web Server



A Simple Erlang Web Server



A Simple Erlang Web Server

- Caller supplies a request handler function
- Handler takes:
  - a socket, and
  - a key-value list supplying request details
- Handler returns a 3-tuple:

  {StatusCode, ResponseHeaders, ResponseBody}

```
start(Handler) ->
    start(Handler, 12345).
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
    spawn(fun() -> accept(LS, Handler) end),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
    spawn(fun() -> accept(LS, Handler) end),
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.

accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
    spawn(fun() -> accept(LS, Handler) end),
    serve(S, Handler, [{headers, []}]).
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.

accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
    spawn(fun() -> accept(LS, Handler) end),
    serve(S, Handler, [{headers, []}]).
```

```
start(Handler) ->
    start(Handler, 12345).
start(Handler, Port) ->
    Opts = [{reuseaddr,true},binary,{backlog,1024}],
    {ok, LS} = gen_tcp:listen(Port, Opts),
    spawn(fun() -> accept(LS, Handler) end),
    receive stop -> gen_tcp:close(LS) end.
```

```
accept(LS, Handler) ->
    {ok, S} = gen_tcp:accept(LS),
    ok = inet:setopts(S, [{packet,http_bin}]),
    spawn(fun() -> accept(LS, Handler) end),
    serve(S, Handler, [{headers, []}]).
```

```
serve(S, Handler, Req) ->
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
{headers, Hdrs} = lists:keyfind(headers, 1, Req),
serve(S, Handler,
      lists:keystore(headers, 1, Req,
                     {headers, [{Hdr,Val}|Hdrs]}));
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
{headers, Hdrs} = lists:keyfind(headers, 1, Req),
serve(S, Handler,
      lists:keystore(headers, 1, Req,
                     {headers, [{Hdr,Val}|Hdrs]}));
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
            {Status, Hdrs, Resp} =
                try Handler(S, Req)
                catch _:_ -> {500, [], <<>>} end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
            {Status, Hdrs, Resp} =
                try Handler(S, Req)
                catch _:_ -> {500, [], <<>>} end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
            {Status, Hdrs, Resp} =
                try Handler(S, Req)
                catch _:_ -> {500, [], <<>>} end,
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
    http_eoh ->
        ok = inet:setopts(S, [{packet, raw}]),
        {Status, Hdrs, Resp} =
            try Handler(S, Req)
            catch _:_ -> {500, [], <<>>} end,
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
{http_error, Error} ->
    exit(Error);
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
{http_error, Error} ->
    exit(Error);
ok ->
    ok
end.
```

```
{http_request, M, {abs_path, Uri}, Vsn} ->
    NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
    serve(S, Handler, NReq);
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
        ok = gen_tcp:send(
               S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
                   [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
                   "\r\n", Resp]),
        gen_tcp:close(S);
    {http_error, Error} ->
        exit(Error);
    ok ->
        ok
end.
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
    http_eoh ->
        ok = inet:setopts(S, [{packet, raw}]),
        {Status, Hdrs, Resp} =
            try Handler(S, Req)
            catch _:_ -> {500, [], <<>>} end,
        ok = gen_tcp:send(
               S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
                   [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
                   "\r\n", Resp])
```

```
serve(S, Handler, Req) ->
    ok = inet:setopts(S, [{active, once}]),
    HttpMsg = receive
                  {http, S, Msg} -> Msg;
                  _ -> gen_tcp:close(S), ok
              end,
    case HttpMsg of
        {http_request, M, {abs_path, Uri}, Vsn} ->
            NReq = [{method,M},{uri,Uri},{version,Vsn}|Req],
            serve(S, Handler, NReq);
        {http_header, _, Hdr, _, Val} ->
            {headers, Hdrs} = lists:keyfind(headers, 1, Req),
            serve(S, Handler,
                  lists:keystore(headers, 1, Req,
                                 {headers, [{Hdr,Val}|Hdrs]}));
        http_eoh ->
            ok = inet:setopts(S, [{packet, raw}]),
            {Status, Hdrs, Resp} =
                try Handler(S, Req)
                catch _:_ -> {500, [], <<>>} end,
            ok = gen_tcp:send(
                   S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
                       [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
                       "\r\n", Resp]),
            gen_tcp:close(S);
        {http_error, Error} ->
            exit(Error);
        ok ->
            ok
    end.
```

40 Lines of Code

"*Erlang is the DSL for writing (web) servers.*"

—@pavlobaron

Feature Details

`out/1` Function

`out/1` Function

- Yaws application callback function
- Used in a number of features
- One argument: an #arg record
- The #arg contains everything Yaws knows about an HTTP request
  - HTTP headers, URL path, client socket, HTTP method, etc.

`out/1` Function

- `out/1` returns a directive or list of directives telling Yaws how to form the response
- Directives look like this:

```
{directive, Details}
```

- Directives can set HTTP status, response headers, response content, redirect, etc.

".yaws" Pages

- Mix Erlang and HTML
- Erlang is JIT-compiled on first page access
- Erlang results inserted into HTML

html.yaws

```
<html>


</html>
```

html.yaws

```
<html>
  <erl>

  </erl>
</html>
```

html.yaws

```
<html>
  <erl>
    out(_) ->
      {html, ["<p>", <<"Hello from html">>, "</p>"]}.
  </erl>
</html>
```

html.yaws

```
<html>
  <erl>
    out(_) ->
      {html, ["<p>", <<"Hello from html">>, "</p>"]}.
  </erl>
</html>
```

html.yaws

```
<html>
  <erl>
    out(_) ->
      {html, ["<p>", <<"Hello from html">>, "</p>"]}.
  </erl>
</html>
```

html.yaws

```
<html>
    <p>Hello from html</p>

</html>
```

".yaws" Pages

- Zero or more `<erl> ... </erl>` blocks per page

ehtml and exhtml

ehtml and exhtml

- Embedding HTML in Erlang strings or binaries is painful

ehtml and exhtml

- Embedding HTML in Erlang strings or binaries is painful
- Instead, write HTML using Erlang terms

ehtml.yaws

```
<html>
  <erl>
    out(_) ->
      {ehtml, [{p, [], "Hello from Yaws"}]}.
  </erl>
</html>
```

ehtml.yaws

```
<html>
  <erl>
    out(_) ->
      {ehtml, [{p, [], "Hello from Yaws"}]}.
  </erl>
</html>
```

ehtml.yaws

```
<html>
  <erl>
    out(_) ->
      {ehtml, [{p, [], "Hello from Yaws"}]}.
  </erl>
</html>
```

HTML element

ehtml.yaws

```
<html>
  <erl>
    out(_) ->
      {ehtml, [{p, [], "Hello from Yaws"}]}.
  </erl>
</html>
```

HTML element

HTML attributes

ehtml.yaws

```
<html>
  <erl>
    out(_) ->
      {ehtml, [{p, [], "Hello from Yaws"}]}.
  </erl>
</html>
```

HTML element

HTML attributes

child elements

Application Modules (appmods)

Application Modules (appmods)

- An "appmod" is an app-specific module that

Application Modules (appmods)

- An "appmod" is an app-specific module that
  - exports an `out/1` function

Application Modules (appmods)

- An "appmod" is an app-specific module that
  - exports an `out/1` function
  - is registered in config to serve a particular URL path

Application Modules (appmods)

- An "appmod" is an app-specific module that
  - exports an `out/1` function
  - is registered in config to serve a particular URL path
  - can be registered to serve "/" to get all requests for that virtual server

```
-module(appmod).
-export([out/1]).
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
```

```erlang
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
    H = yaws_api:reformat_header(A#arg.headers),
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
    H = yaws_api:reformat_header(A#arg.headers),
    {ehtml,
     [{hr,[],[]},
      {h3,[], "The headers passed to us were:"},
      {ol, [], [{li,[], {p,[],Hdr}} || Hdr <- H]},
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
    H = yaws_api:reformat_header(A#arg.headers),
    {ehtml,
     [{hr,[],[]},
      {h3,[], "The headers passed to us were:"},
      {ol, [], [{li,[], {p,[],Hdr}} || Hdr <- H]},
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
    H = yaws_api:reformat_header(A#arg.headers),
    {ehtml,
     [{hr,[],[]},
      {h3,[], "The headers passed to us were:"},
      {ol, [], [{li,[], {p,[],Hdr}} || Hdr <- H]},

      {hr,[],[]},
      {h3, [], "The request:"},
      {ul,[],
       [{li,[], f("method: ~s", [Method])},
        {li,[], f("path: ~p", [Path])},
        {li,[], f("version: ~p", [Vsn])}]}]}.
```

```
-module(appmod).
-export([out/1]).
-include_lib("yaws_api.hrl").
-import(yaws_api, [f/2]).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    Path = Req#http_request.path,
    Vsn = Req#http_request.version,
    H = yaws_api:reformat_header(A#arg.headers),
    {ehtml,
     [{hr,[],[]},
      {h3,[], "The headers passed to us were:"},
      {ol, [], [{li,[], {p,[],Hdr}} || Hdr <- H]},

      {hr,[],[]},
      {h3, [], "The request:"},
      {ul,[],
       [{li,[], f("method: ~s", [Method])},
        {li,[], f("path: ~p", [Path])},
        {li,[], f("version: ~p", [Vsn])}]}]}.
```

appmod Results

The headers passed to us were:

1. Connection: keep-alive
2. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
3. Host: localhost:8000
4. User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/536.26.14 (KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
5. Accept-Encoding: gzip, deflate
6. Accept-Language: en-us
7. Dnt: 1

The request:

- method: GET
- path: {abs_path,"/appmod?a=1&b=4"}
- version: {1,1}

JSON Example

```
fl(F,P) ->
    lists:flatten(f(F,P)).
```

```
fl(F,P) ->
    lists:flatten(f(F,P)).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    {abs_path, Path} = Req#http_request.path,
    {Mjr,Mnr} = Req#http_request.version,
    Hdrs = yaws_api:reformat_header(A#arg.headers,
                                    fun(H,V) -> {H,V} end),
```

```erlang
fl(F,P) ->
    lists:flatten(f(F,P)).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    {abs_path, Path} = Req#http_request.path,
    {Mjr,Mnr} = Req#http_request.version,
    Hdrs = yaws_api:reformat_header(A#arg.headers,
                                    fun(H,V) -> {H,V} end),
    Json = {struct, [{"headers", {struct, Hdrs}},
                     {"method", fl("~s", [Method])},
                     {"path", fl("~p", [Path])},
                     {"version", fl("~p.~p", [Mjr,Mnr])}]},
```

```
fl(F,P) ->
    lists:flatten(f(F,P)).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    {abs_path, Path} = Req#http_request.path,
    {Mjr,Mnr} = Req#http_request.version,
    Hdrs = yaws_api:reformat_header(A#arg.headers,
                                    fun(H,V) -> {H,V} end),
    Json = {struct, [{"headers", {struct, Hdrs}},
                     {"method", fl("~s", [Method])},
                     {"path", fl("~p", [Path])},
                     {"version", fl("~p.~p", [Mjr,Mnr])}]},
    JsonStr = json2:encode(Json),
```

```
fl(F,P) ->
    lists:flatten(f(F,P)).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    {abs_path, Path} = Req#http_request.path,
    {Mjr,Mnr} = Req#http_request.version,
    Hdrs = yaws_api:reformat_header(A#arg.headers,
                                    fun(H,V) -> {H,V} end),
    Json = {struct, [{"headers", {struct, Hdrs}},
                     {"method", fl("~s", [Method])},
                     {"path", fl("~p", [Path])},
                     {"version", fl("~p.~p", [Mjr,Mnr])}]},
    JsonStr = json2:encode(Json),
    [{status, 200},
     {content, "application/json", JsonStr}].
```

```erlang
fl(F,P) ->
    lists:flatten(f(F,P)).

out(A) ->
    Req = A#arg.req,
    Method = Req#http_request.method,
    {abs_path, Path} = Req#http_request.path,
    {Mjr,Mnr} = Req#http_request.version,
    Hdrs = yaws_api:reformat_header(A#arg.headers,
                                    fun(H,V) -> {H,V} end),
    Json = {struct, [{"headers", {struct, Hdrs}},
                     {"method", fl("~s", [Method])},
                     {"path", fl("~p", [Path])},
                     {"version", fl("~p.~p", [Mjr,Mnr])}]},
    JsonStr = json2:encode(Json),
    [{status, 200},
     {content, "application/json", JsonStr}].
```

JSON Results

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 18:49:49 GMT

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 18:49:49 GMT
Content-Length: 201

JSON Results

```
$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 18:49:49 GMT
Content-Length: 201
Content-Type: application/json
```

JSON Results

```
$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 18:49:49 GMT
Content-Length: 201
Content-Type: application/json
```

JSON Results

$ curl -D /dev/tty http://localhost:8000/appmod
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 18:49:49 GMT
Content-Length: 201
Content-Type: application/json

{"headers":{"Accept":"*/*","Host":"localhost:
8000","User-Agent":"curl/7.21.4 (universal-apple-
darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8r zlib/
1.2.5"},"method":"GET","path":"\"/appmod
\"","version":"1.1"}

Streaming

Streaming

- Sometimes you can't deliver responses all at once
- Maybe you don't have all the content yet, e.g. live video
- Maybe you're using long polling to simulate server events

Streaming

Streaming

- Return stream directive from `out/1`

Streaming

- Return stream directive from `out/1`
- For chunked transfer:

```
{streamcontent, MimeType, FirstChunk}
```

Streaming

- Return stream directive from `out/1`
- For chunked transfer:

```
{streamcontent, MimeType, FirstChunk}
```

- For non-chunked:

```
[{header, {transfer_encoding, erase}},
{streamcontent_from_pid, MimeType, Streamer}]
```

Chunked Streaming

```
out(A) ->
```

```
out(A) ->
    YawsPid = A#arg.pid,
```

```
out(A) ->
    YawsPid = A#arg.pid,
    spawn(fun() ->
                B = <<"01234567890">>,
                Bin = list_to_binary([B,B,B,B]),
                loop(YawsPid, Bin, 18)
        end),
```

```
out(A) ->
    YawsPid = A#arg.pid,
    spawn(fun() ->
                B = <<"01234567890">>,
                Bin = list_to_binary([B,B,B,B]),
                loop(YawsPid, Bin, 18)
        end),
{streamcontent, "application/octet-stream", <<"ABCDEF">>}.
```

```
out(A) ->
    YawsPid = A#arg.pid,
    spawn(fun() ->
                B = <<"01234567890">>,
                Bin = list_to_binary([B,B,B,B]),
                loop(YawsPid, Bin, 18)
        end),
{streamcontent, "application/octet-stream", <<"ABCDEF">>}.
```

```
loop(YawsPid, <<>>, _) ->
```

```
loop(YawsPid, <<>>, _) ->
    yaws_api:stream_chunk_end(YawsPid);
```

```
loop(YawsPid, <<>>, _) ->
    yaws_api:stream_chunk_end(YawsPid);
loop(YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
```

```
loop(YawsPid, <<>>, _) ->
    yaws_api:stream_chunk_end(YawsPid);
loop(YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_chunk_deliver(YawsPid, Chunk),
            loop(YawsPid, Rest, ChunkSize);
```

```
loop(YawsPid, <<>>, _) ->
    yaws_api:stream_chunk_end(YawsPid);
loop(YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_chunk_deliver(YawsPid, Chunk),
            loop(YawsPid, Rest, ChunkSize);
        _ ->
            yaws_api:stream_chunk_deliver(YawsPid, Bin),
            loop(YawsPid, <<>>, ChunkSize)
    end.
```

```
loop(YawsPid, <<>>, _) ->
    yaws_api:stream_chunk_end(YawsPid);
loop(YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_chunk_deliver(YawsPid, Chunk),
            loop(YawsPid, Rest, ChunkSize);
        _ ->
            yaws_api:stream_chunk_deliver(YawsPid, Bin),
            loop(YawsPid, <<>>, ChunkSize)
    end.
```

Chunked Results

$ nc localhost 8000 < stream1.get
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 19:54:37 GMT
Content-Type: application/octet-stream
Transfer-Encoding: chunked

6
ABCDEF
12
012345678900123456
12
789001234567890012
8
34567890
0

Chunked Results

```
$ nc localhost 8000 < stream1.get
HTTP/1.1 200 OK
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 19:54:37 GMT
Content-Type: application/octet-stream
Transfer-Encoding: chunked

6
ABCDEF
12
012345678900123456
12
789001234567890012
8
34567890
0
```

Non-chunked Streaming

```
out(A) ->
    Sock = A#arg.clisock,
```

```
out(A) ->
    Sock = A#arg.clisock,
    Pid = spawn(fun() ->
                receive
                    {ok,YawsPid} ->
```

```
out(A) ->
    Sock = A#arg.clisock,
    Pid = spawn(fun() ->
                receive
                    {ok,YawsPid} ->
                        First = <<"ABCDEF">>,
                        B = <<"01234567890">>,
                        Bin = list_to_binary([First,B,B,B,B]),
                        loop(Sock, YawsPid, Bin, 18);
```

```
out(A) ->
    Sock = A#arg.clisock,
    Pid = spawn(fun() ->
                receive
                    {ok,YawsPid} ->
                        First = <<"ABCDEF">>,
                        B = <<"01234567890">>,
                        Bin = list_to_binary([First,B,B,B,B]),
                        loop(Sock, YawsPid, Bin, 18);
                {discard,YawsPid} ->
                    yaws_api:stream_process_end(Sock,YawsPid)
            end
        end),
```

```
out(A) ->
    Sock = A#arg.clisock,
    Pid = spawn(fun() ->
                receive
                    {ok,YawsPid} ->
                        First = <<"ABCDEF">>,
                        B = <<"01234567890">>,
                        Bin = list_to_binary([First,B,B,B,B]),
                        loop(Sock, YawsPid, Bin, 18);
                {discard,YawsPid} ->
                    yaws_api:stream_process_end(Sock,YawsPid)
            end
        end),
[{header, {transfer_encoding, erase}},
 {streamcontent_from_pid, "application/octet-stream", Pid}].
```

```
out(A) ->
    Sock = A#arg.clisock,
    Pid = spawn(fun() ->
                receive
                    {ok,YawsPid} ->
                        First = <<"ABCDEF">>,
                        B = <<"01234567890">>,
                        Bin = list_to_binary([First,B,B,B,B]),
                        loop(Sock, YawsPid, Bin, 18);
                {discard,YawsPid} ->
                    yaws_api:stream_process_end(Sock,YawsPid)
            end
        end),
[{header, {transfer_encoding, erase}},
 {streamcontent_from_pid, "application/octet-stream", Pid}].
```

```
loop(Sock, YawsPid, <<>>, _) ->
    yaws_api:stream_process_end(Sock, YawsPid);
```

```
loop(Sock, YawsPid, <<>>, _) ->
    yaws_api:stream_process_end(Sock, YawsPid);
loop(Sock, YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
```

```
loop(Sock, YawsPid, <<>>, _) ->
    yaws_api:stream_process_end(Sock, YawsPid);
loop(Sock, YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_process_deliver(Sock, Chunk),
            loop(Sock, YawsPid, Rest, ChunkSize);
```

```
loop(Sock, YawsPid, <<>>, _) ->
    yaws_api:stream_process_end(Sock, YawsPid);
loop(Sock, YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_process_deliver(Sock, Chunk),
            loop(Sock, YawsPid, Rest, ChunkSize);
        _ ->
            yaws_api:stream_process_deliver(Sock, Bin),
            loop(Sock, YawsPid, <<>>, ChunkSize)
    end.
```

```
loop(Sock, YawsPid, <<>>, _) ->
    yaws_api:stream_process_end(Sock, YawsPid);
loop(Sock, YawsPid, Bin, ChunkSize) ->
    case Bin of
        <<Chunk:ChunkSize/binary, Rest/binary>> ->
            yaws_api:stream_process_deliver(Sock, Chunk),
            loop(Sock, YawsPid, Rest, ChunkSize);
        _ ->
            yaws_api:stream_process_deliver(Sock, Bin),
            loop(Sock, YawsPid, <<>>, ChunkSize)
    end.
```

Non-chunked Results

```
$ nc localhost 8000 < stream2.get
HTTP/1.1 200 OK
Connection: close
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 20:09:04 GMT
Content-Type: application/octet-stream

ABCDEF01234567890012345678900123456789001234567890
```

Non-chunked Results

```
$ nc localhost 8000 < stream2.get
HTTP/1.1 200 OK
Connection: close
Server: Yaws 1.94
Date: Sun, 23 Sep 2012 20:09:04 GMT
Content-Type: application/octet-stream

ABCDEF01234567890012345678900123456789001234567890
```

Server-Sent Events

Server-Sent Events

- Server-Sent Events (SSE) is a W3C working draft for standardizing how servers send events to clients
- See http://www.w3.org/TR/eventsource/
- Supported by Chrome, Safari, Firefox, Opera

SSE in Yaws

SSE in Yaws

- Supported by the `yaws_sse` module (see https://github.com/klacke/yaws/blob/master/src/yaws_sse.erl)

SSE in Yaws

- Supported by the `yaws_sse` module (see https://github.com/klacke/yaws/blob/master/src/yaws_sse.erl)
- Built over non-chunked streaming

SSE in Yaws

- Supported by the `yaws_sse` module (see https://github.com/klacke/yaws/blob/master/src/yaws_sse.erl)
- Built over non-chunked streaming
- `yaws_sse:headers/1` returns HTTP headers and Yaws directives to set up an event streaming Erlang process

SSE in Yaws

- Supported by the `yaws_sse` module (see https://github.com/klacke/yaws/blob/master/src/yaws_sse.erl)
- Built over non-chunked streaming
- `yaws_sse:headers/1` returns HTTP headers and Yaws directives to set up an event streaming Erlang process
- `yaws_sse:data` functions deliver event data to client

SSE in Yaws

- Supported by the `yaws_sse` module (see https://github.com/klacke/yaws/blob/master/src/yaws_sse.erl)
- Built over non-chunked streaming
- `yaws_sse:headers/1` returns HTTP headers and Yaws directives to set up an event streaming Erlang process
- `yaws_sse:data` functions deliver event data to client
- And more, see `yaws_sse` for more details

yaws_sse:headers/1

```
headers(StreamPid) ->
    [{status, 200},
     {header, {"Cache-Control", "no-cache"}},
     {header, {connection, "close"}},
     {header, {transfer_encoding, erase}},
     {streamcontent_from_pid, "text/event-stream", StreamPid}].
```

SSE in Yaws

- To see a running example, visit

  http://yaws.hyber.org/server_sent_events.yaws

- Also see this Internet Computing column:

  http://steve.vinoski.net/pdf/IC-Yaws_SSE.pdf

WebSocket

WebSocket

- WebSocket Protocol, RFC 6455
- The HTTP connection is "upgraded" to WebSocket using the Upgrade HTTP header
- WebSocket has a bit of framing but is essentially a bidirectional TCP connection

Yaws WebSocket Support

Yaws WebSocket Support

- An `out/1` function indicates it wants WebSocket by returning

```
{websocket, CallbackModule, Options}
```

Yaws WebSocket Support

- An `out/1` function indicates it wants WebSocket by returning

```
{websocket, CallbackModule, Options}
```

- Once the connection is upgraded, Yaws calls `handle_message/1` or `handle_message/2` on the callback module

Yaws WebSocket Support

- An `out/1` function indicates it wants WebSocket by returning

```
{websocket, CallbackModule, Options}
```

- Once the connection is upgraded, Yaws calls `handle_message/1` or `handle_message/2` on the callback module
- App can also call `yaws_api:websocket_send` to send data outside of a callback

Yaws WebSocket Example

- Visit http://yaws.hyber.org/websockets.yaws for a working example
- Let's go through it

Basic Echo Example

- Sending "bye" closes the connection.
- Sending "say hi later" sends "hi there!" asynchronously.
- Sending "something" does nothing.

client-connected
echo this
I'll say hi in a bit...
hi there!

Say Something: bye Send

out/1 Function

```
out(A) ->
    CallbackMod = basic_echo_callback,
    Opts = [{origin, "http://" ++ (A#arg.headers)#headers.host}],
    {websocket, CallbackMod, Opts}.
```

Callback Module

```
handle_message({text, <<"bye">>}) ->
    {close, normal};
```

Callback Module

```
handle_message({text, <<"bye">>}) ->
    {close, normal};
```

Callback Module

```
handle_message({text, <<"bye">>}) ->
    {close, normal};
```

```
handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;
```

```
handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;
```

```
handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;
```

```
handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;

handle_message({text, <<"say hi later">>}) ->
    timer:apply_after(3000, ?MODULE, say_hi, [self()]),
    {reply, {text, <<"I'll say hi in a bit...">>}};
```

```
handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;

handle_message({text, <<"say hi later">>}) ->
    timer:apply_after(3000, ?MODULE, say_hi, [self()]),
    {reply, {text, <<"I'll say hi in a bit...">>}};
```

```
say_hi(Pid) ->
    yaws_api:websocket_send(Pid, {text, <<"hi there!">>}).

handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;

handle_message({text, <<"say hi later">>}) ->
    timer:apply_after(3000, ?MODULE, say_hi, [self()]),
    {reply, {text, <<"I'll say hi in a bit...">>}};
```

```
say_hi(Pid) ->
    yaws_api:websocket_send(Pid, {text, <<"hi there!">>}).

handle_message({text, <<"bye">>}) ->
    {close, normal};

handle_message({text, <<"something">>}) ->
    noreply;

handle_message({text, <<"say hi later">>}) ->
    timer:apply_after(3000, ?MODULE, say_hi, [self()]),
    {reply, {text, <<"I'll say hi in a bit...">>}};
```

```
handle_message({text, Message}) ->
    {reply, {text, <<Message/binary>>}};

handle_message({binary, Message}) ->
    {reply, {binary, Message}};

handle_message({close, _Status, _Reason}) ->
    {close, normal}.
```

Performance

Simple Web Server

Simple Web Server

$ ab -c 500 -n 50000 http://localhost:12345/

...

Simple Web Server

$ ab -c 500 -n 50000 http://localhost:12345/

...

Requests per second: **20923.17** [#/sec] (mean)

Simple Web Server

$ ab -c 500 -n 50000 http://localhost:12345/

...

Requests per second: **20923.17** [#/sec] (mean)

- One response header ("Server: sws")

Simple Web Server

$ ab -c 500 -n 50000 http://localhost:12345/

...

Requests per second: **20923.17** [#/sec] (mean)

- One response header ("Server: sws")
- No response body

Simple Web Server

$ ab -c 500 -n 50000 http://localhost:12345/

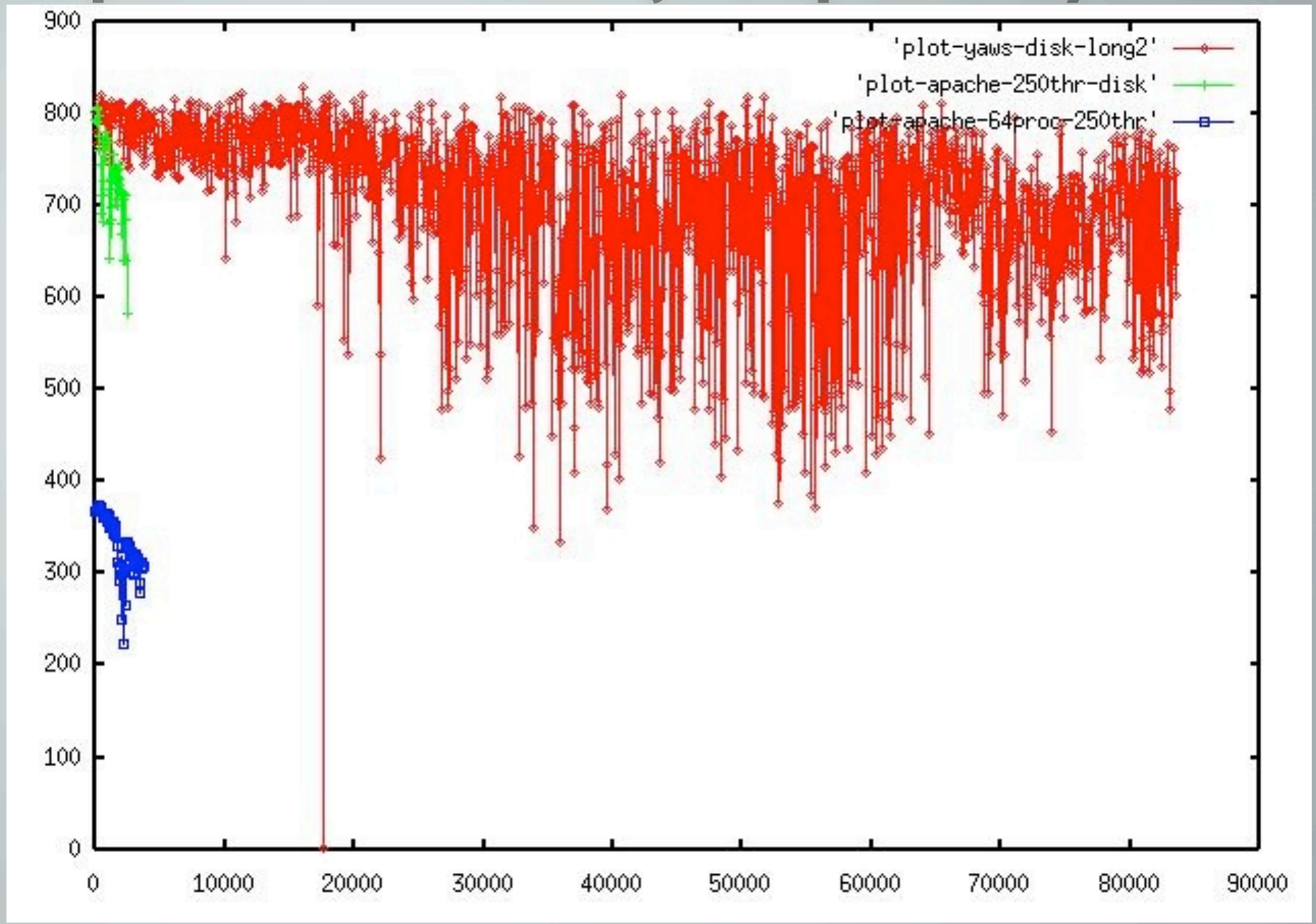
...

Requests per second: **20923.17** [#/sec] (mean)

- One response header ("Server: sws")
- No response body
- Server and client on same host (Ubuntu 12.04, 16GB RAM, 3.4GHz Intel i7-2600K)

Apache vs. Yaws

http://www.sics.se/~joe/apachevsyaws.html



$ curl -I http://www.sics.se/~joe/apachevsyaws.html

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes
Content-Length: 4286

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes
Content-Length: 4286
Connection: close

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes
Content-Length: 4286
Connection: close
Content-Type: text/html

$ curl -I http://www.sics.se/~joe/apachevsyaws.html
HTTP/1.1 200 OK
Date: Sat, 22 Sep 2012 23:26:25 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes
Content-Length: 4286
Connection: close
Content-Type: text/html

The Performance Presumption

steve.vinoski.net/pdf/IEEE-The_Performance_Presumption.pdf

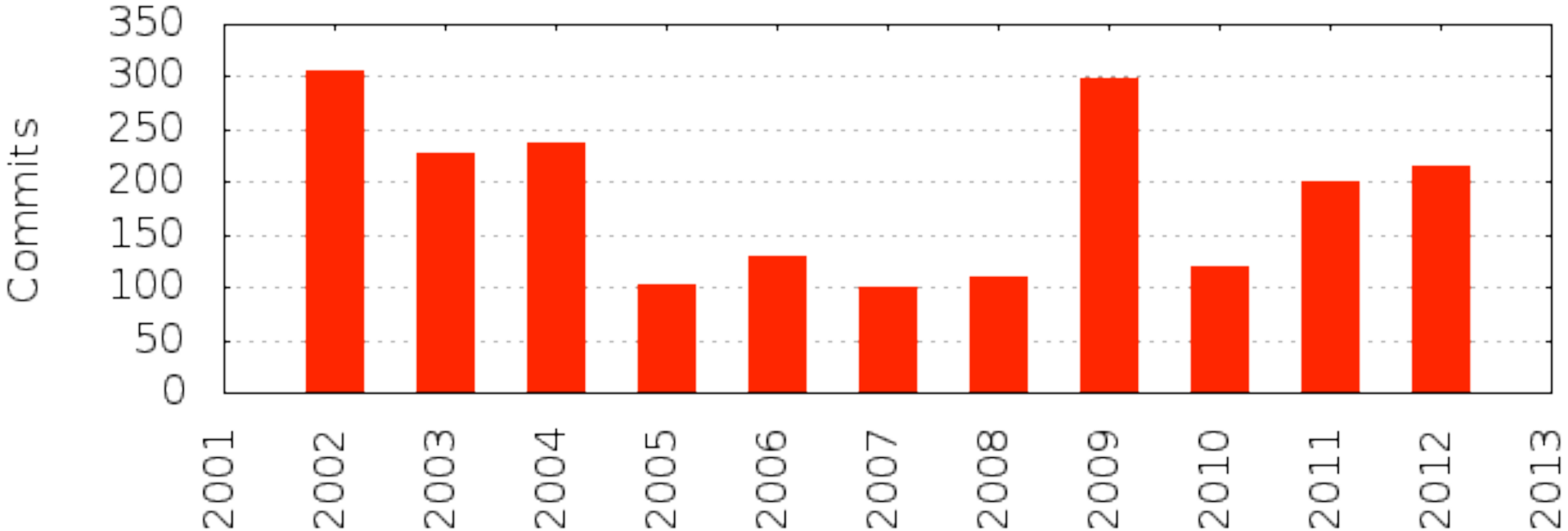
Do your own benchmarks!

http://steve.vinoski.net/blog/2011/05/09/erlang-web-server-benchmarking/

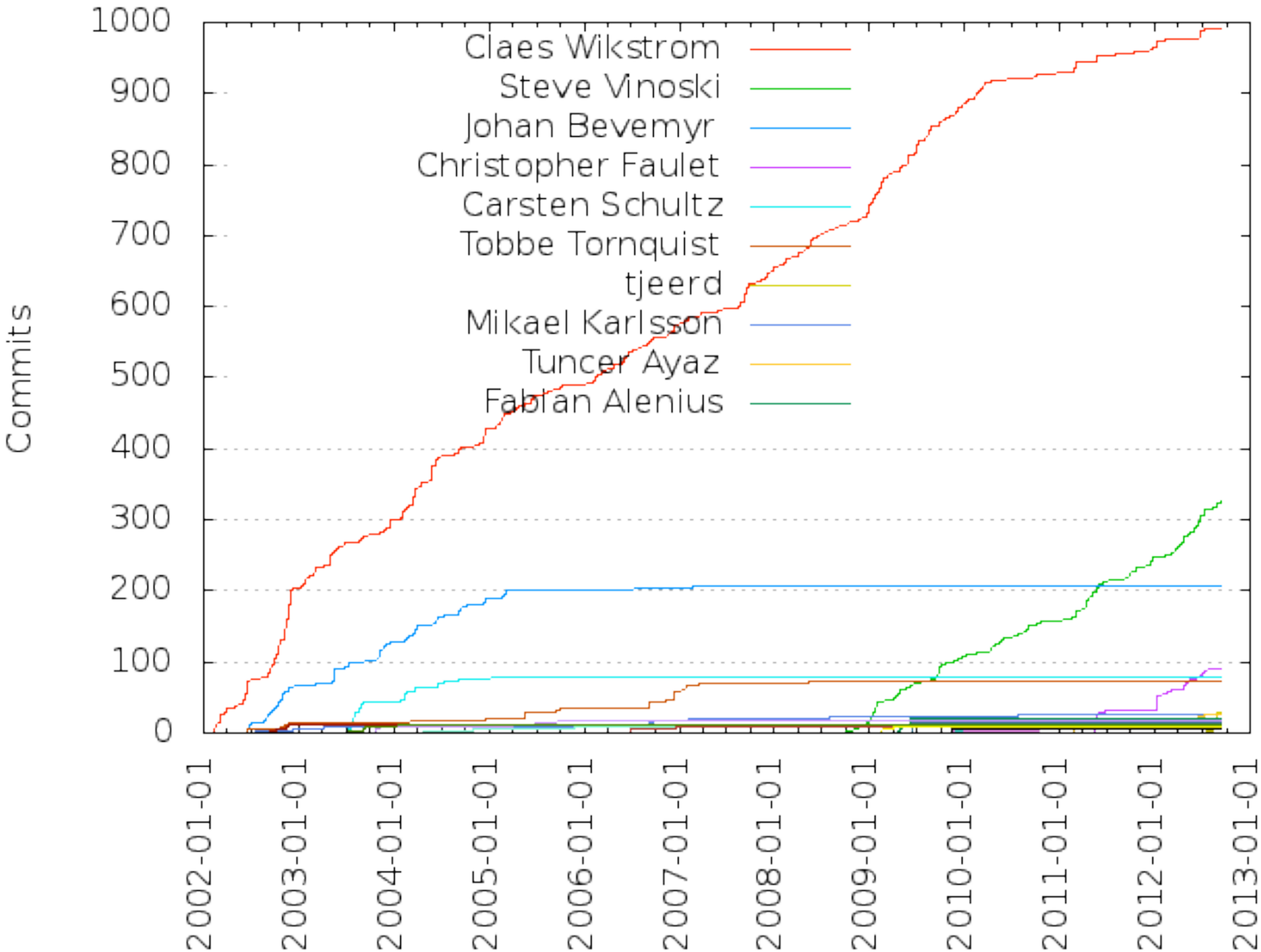
http://www.mnot.net/blog/2011/05/18/http_benchmark_rules

Yaws Community

Commits Per Year



Commits Per Author



Top Contributors

Author of Year

Year	Author	Commits (%)	Next top 5	Number of authors
2012	Steve Vinoski	78 (36.11% of 216)	Christopher Faulet, tjeerd, Claes Wikstrom, Tuncer Ayaz, Sergei Golovan	16
2011	Steve Vinoski	90 (45.00% of 200)	Christopher Faulet, Claes Wikstrom, Tuncer Ayaz, Torbjorn Tornkvist, Garret Smith	14
2010	Steve Vinoski	50 (41.32% of 121)	Claes Wikstrom, Mikael Karlsson, Torbjorn Tornkvist, Per Andersson, Dominique Boucher	16
2009	Claes Wikstrom	141 (47.32% of 298)	Steve Vinoski, Fabian Alenius, Olivier Girondel, Hans Ulrich Niedermann, davide	12
2008	Claes Wikstrom	88 (79.28% of 111)	Steve Vinoski, Tobbe Tornquist, Mikael Karlsson	4
2007	Claes Wikstrom	80 (80.00% of 100)	Tobbe Tornquist, Julian Noble, Mikael Karlsson, Johan Bevemyr	5
2006	Claes Wikstrom	84 (65.12% of 129)	Tobbe Tornquist, Yariv Sadan, Mikael Karlsson, Johan Bevemyr, Sebastian Stroll	7
2005	Claes Wikstrom	62 (60.19% of 103)	Tobbe Tornquist, Johan Bevemyr, Mickael Remond, Martin Bjorklund, Carsten Schultz	7
2004	Claes Wikstrom	128 (53.78% of 238)	Johan Bevemyr, Carsten Schultz, Martin Bjorklund, Tobbe Tornquist, Leon Smith	8
2003	Claes Wikstrom	98 (43.17% of 227)	Johan Bevemyr, Carsten Schultz, Leon Smith, Mickael Remond, Tobbe Tornquist	8
2002	Claes Wikstrom	203 (66.12% of 307)	Johan Bevemyr, Tobbe Tornquist, Luke Gorrie, Mikael Karlsson, Seah Hinde	7

115 Total Contributors

We welcome your contributions

(i.e., please don't write yet another Erlang web server)

Yaws Future

- Current release: Yaws 1.94
- Experimenting with Webmachine on Yaws
- Ideas for more flexible dispatching
- What about Yaws 2.0? SPDY? HTTP 2.0?

Yaws evolution will continue to be user-driven

Yaws Summary

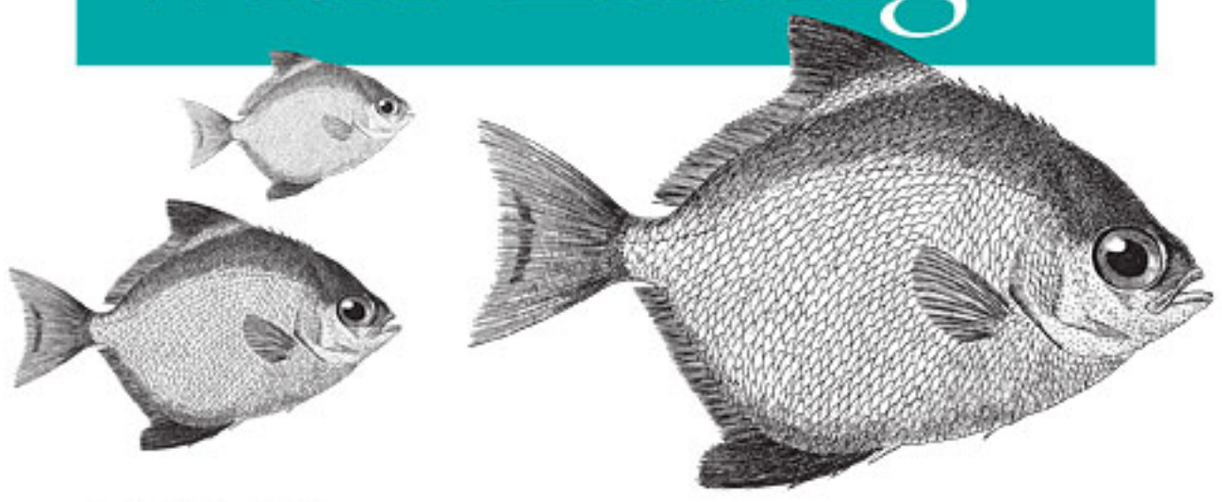
- Yaws supplies a lot of features
- Known for great reliability, stability, and performance
- All features driven by actual production usage
- Even though it's 11 years old, we continue to actively develop and maintain Yaws

For More Info

- http://yaws.hyber.org
- http://github.com/klacke/yaws
- Yaws mailing list: https://lists.sourceforge.net/lists/listinfo/erlyaws-list
- http://steve.vinoski.net/blog/internet-computing-columns/#2011-4
- http://www.infoq.com/articles/vinoski-erlang-rest

Working with REST and WebSockets on Yaws

Building Web Applications with Erlang



O'REILLY®

Zachary Kessin

Steve Vinoski
Basho Technologies
Cambridge, MA USA
http://basho.com
vinoski@ieee.org
http://steve.vinoski.net/
@stevevinoski