

Who am I?



- Ezra Zygmuntowicz
- Rubyist for 4 years
- Engine Yard Founder and Architect
- Blog: <http://brainspl.at>

Mongrel

Learning how to walk the dog



What is Mongrel?

Mongrel is an HTTP Server Library written by Zed Shaw

- Fast HTTP Parser written in Ragel + C
- Fast URI Classifier written in C
- Stackable Request Handlers
- Flexible Configuration
- Secure and RFC Compliant HTTP Parser

Ragel State Machine Defined HTTP Parser

```
#### HTTP PROTOCOL GRAMMAR
# line endings
CRLF = "\r\n";

# character types
CTL = (cntrl | 127);
safe = (" $" | "-" | "_" | ".");
extra = ("!" | "*" | "'" | "(" | ")" | ",");
reserved = (";" | "/" | "?" | ":" | "@" | "&" | "=" | "+");
unsafe = (CTL | " " | "\" | "#" | "%" | "<" | ">");
national = any -- (alpha | digit | reserved | extra | safe | unsafe);
unreserved = (alpha | digit | safe | extra | national);
escape = ("% " xdigit xdigit);
uchar = (unreserved | escape);
pchar = (uchar | ":" | "@" | "&" | "=" | "+");
tspecials = ("(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | \" | "/" | "[" | "]" | "?" | "=" | "{" | "}" | " " | "\t");

# elements
token = (ascii -- (CTL | tspecials));

# URI schemes and absolute paths
scheme = ( alpha | digit | "+" | "-" | "." ) * ;
absolute_uri = (scheme ":" (uchar | reserved) *);

path = (pchar+ ( "/" pchar* ) * ) ;
query = ( uchar | reserved ) * %query_string ;
param = ( pchar | "/" ) * ;
params = (param ( ";" param ) * ) ;
rel_path = (path? %request_path ( ";" params ) ? ) ( "?" %start_query query ) ? ;
absolute_path = ( "/" + rel_path );

Request_URI = ( "*" | absolute_uri | absolute_path ) >mark %request_uri;
Method = (upper | digit | safe){1,20} >mark %request_method;

http_number = (digit+ "." digit+ ) ;
HTTP_Version = ("HTTP/" http_number) >mark %http_version ;
Request_Line = (Method " " Request_URI " " HTTP_Version CRLF) ;
```

Why?

FastCGI is Poop



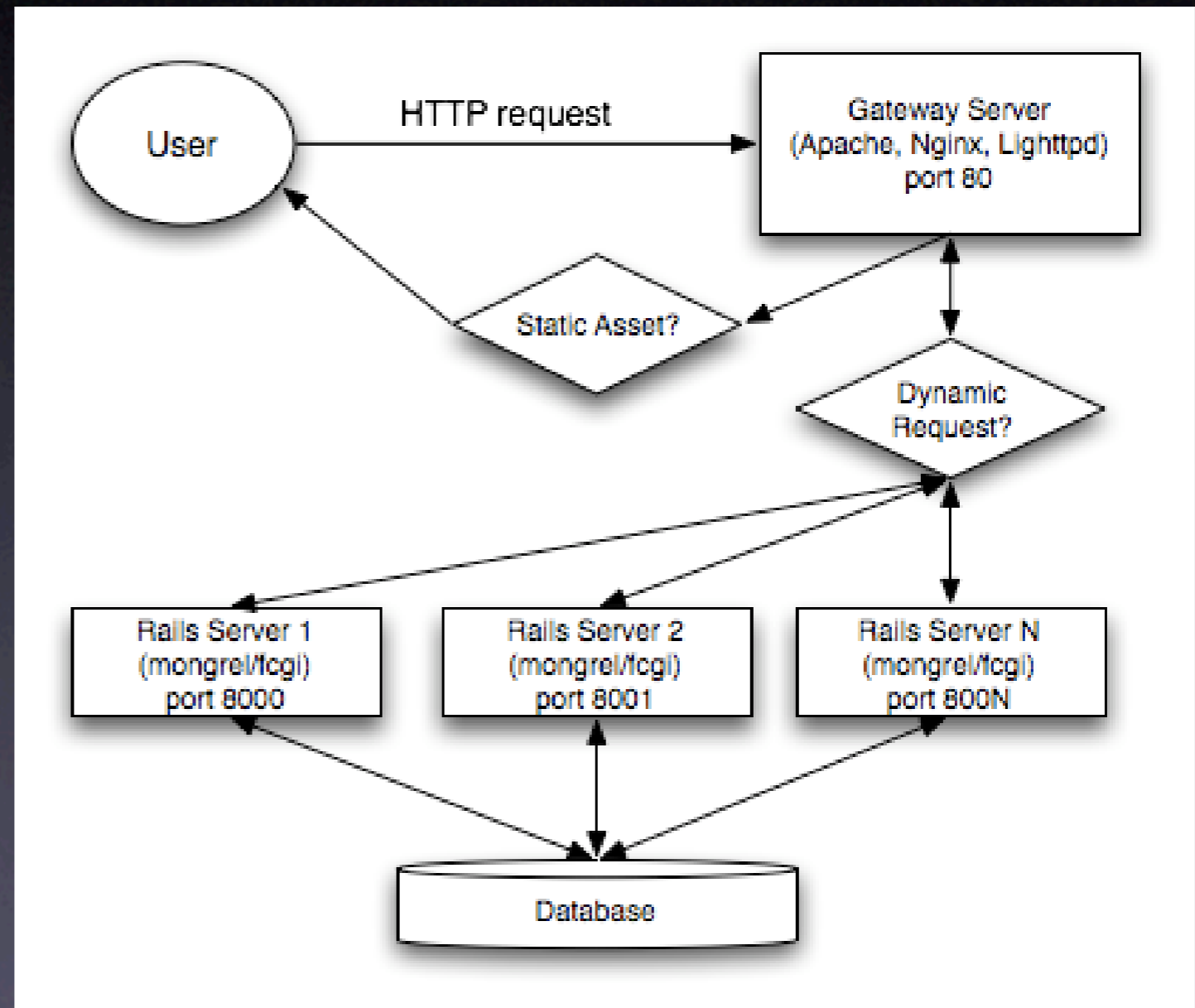
- HTTP is a well known and well tooled protocol
- Mongrel is way easier to setup and use
- Transparent wire protocol

But Rails isn't Thread Safe!

- Giant Mutex Lock around Rails Dispatch
- Only one request served at a time by one mongrel
- Use `mongrel_cluster` to scale with multiple processes

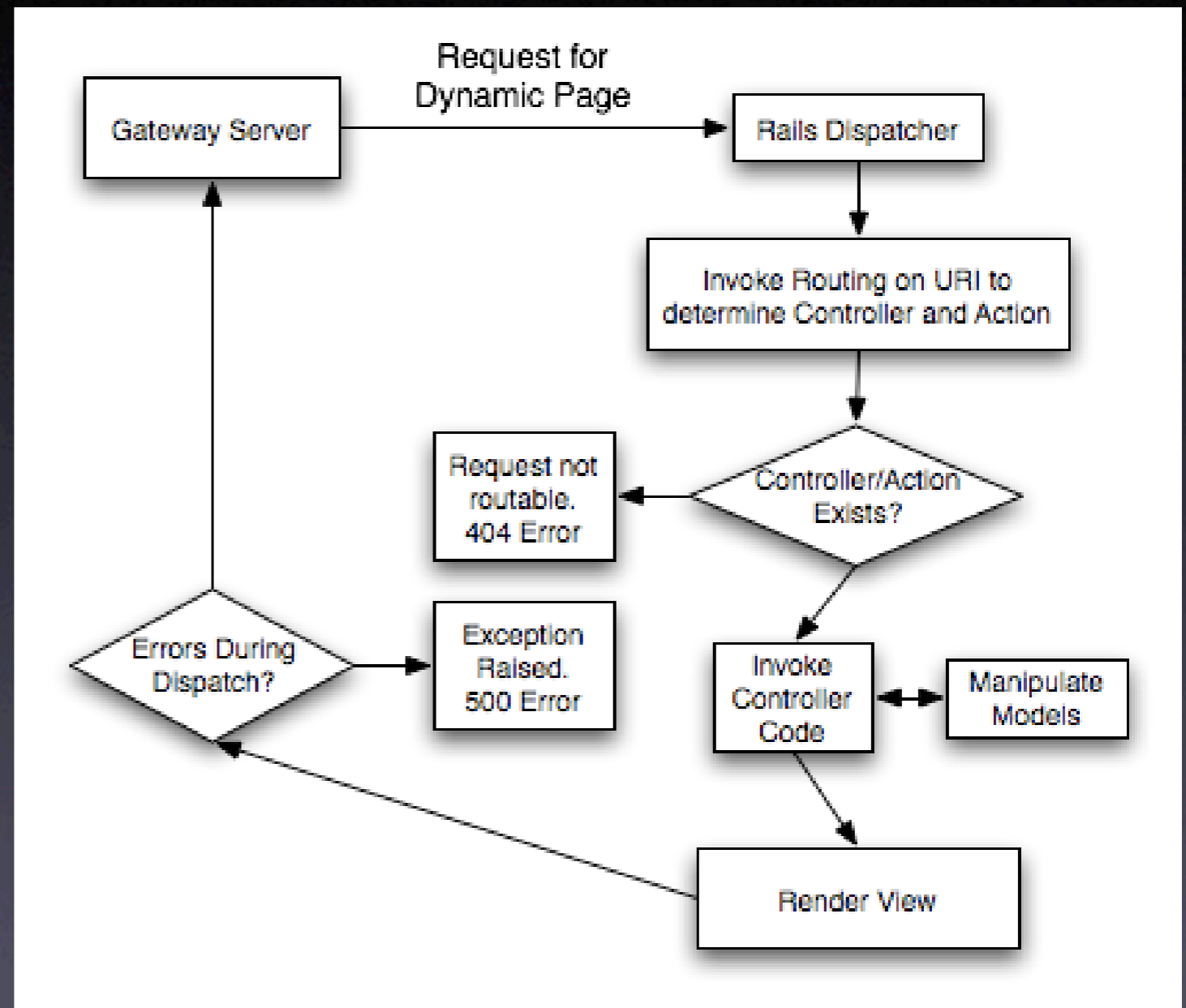
Full Stack Request/Response Life-Cycle

- Request comes into gateway server
- Rewrite rules are evaluated and request gets served directly if it's a static asset
- Dynamic requests are proxied to one Mongrel in the Mongrel Cluster
- Mongrel dispatches request through Rails and returns response to client



Rails Internal Request/Response Life-Cycle

- Mongrel Locks Mutex
- Rails Dispatcher is invoked with request/response objects
- Routing is invoked and returns the proper Controller object or 404 if no route found
- Filter chain is invoked
- Controller's Action is called, manipulates Models
- View is rendered and any after filters are called
- Mongrel Unlocks Mutex
- Final response or error page returned to client



Mongrel != Rails

- Mongrel **is** Thread Safe
- Mongrel is capable of much more than just running Rails
- Rails is beautiful for the 80% part of the 80/20 rule
- What to do when your app needs the other 20%?

Handle It!

- Building Mongrel Handlers is easier than you think
- Mongrel is **very** high performance
- Ruby not so slow after all? Maybe it's just Rails that is slow?

Rails vs Mongrel Handler in a Hello World Battle

Rails:

```
class HelloController < ApplicationController

  def world
    render :text => "Hello World!"
  end

end
```

Mongrel
Handler:

```
class HelloHandler < Mongrel::HttpHandler

  def process(request, response)
    response.start(200) do |head, out|
      head["Content-Type"] = "text/html"
      out.write "Hello World!"
    end
  end

end
```

Naive Benchmarks

Mongrel
Handler:
7Mb RAM

```
Concurrency Level:      100
Time taken for tests:   1.136 seconds
Complete requests:     1000
Failed requests:       0
Broken pipe errors:    0
Total transferred:     132156 bytes
HTML transferred:      12000 bytes
Requests per second:   880.28 [#/sec] (mean)
Time per request:      113.60 [ms] (mean)
Time per request:      1.14 [ms] (mean, across all concurrent requests)
Transfer rate:         116.33 [Kbytes/sec] received
```

```
... 0.073s
-- add_index(:sessi
--> 0.2173s
-- add_index(:sessi
--> 0.0603s
-- AddSessions: mig
ez railsapp & mongre
** Starting Mongrel
** Starting Rails v
** Loading any Rail
** Signal's ready
```

Rails:
35Mb RAM

```
Concurrency Level:      1
Time taken for tests:   8.105 seconds
Complete requests:     1000
Failed requests:       0
Broken pipe errors:    0
Total transferred:     286000 bytes
HTML transferred:      12000 bytes
Requests per second:   123.38 [#/sec] (mean)
Time per request:      8.10 [ms] (mean)
Time per request:      8.10 [ms] (mean, across all concurrent requests)
Transfer rate:         35.29 [Kbytes/sec] received
```

```
from /usr/lo
from /usr/lo
from /usr/lo
from /usr/lo
from /usr/lo
from /usr/lo
... 40 leve
from /usr/lo
from /usr/lo
from /usr/lo
ez railsapp & mongre
** Starting Mongrel
```

Why the Huge Difference?

- Of course Rails provides much more out of the box
- Our HelloHandler runs in a multi-threaded way, hence the 100 concurrent users in our benchmark
- Rails has to run requests in serial, hence the 1 concurrent user.

```
@guard.synchronize {  
  @active_request_path = request.params["PATH_INFO"]  
  Dispatcher.dispatch(cgi, ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, response.body)  
  @active_request_path = nil  
}
```

What can Custom Mongrel Handlers do for me?

- More concurrent users on fewer processes
- Higher throughput with less resources
- Less convenience for developers means you don't need it until you **need** it

Standalone or Integrated with Rails?

- Mongrel Handlers can 'Stack'
- You can use `:in_front => true` to put a custom Mongrel Handler in process with your Rails app and have it intercept and serve certain urls
- Access to your Rails models and other loaded classes

Integrating HelloHandler into our Rails app

```
---  
port: 3000  
pid_file: log/mongrel.pid  
servers: 1  
config_script: /Users/ez/mongrel/railsapp/config/hello.rb  
environment: production
```

```
class HelloHandler < Mongrel::HttpHandler  
  def process(request, response)  
    response.start(200) do |head, out|  
      head["Content-Type"] = "text/html"  
      out.write "Hello World! from HelloHandler"  
    end  
  end  
end  
end  
uri "/hellohandler", :handler => HelloHandler.new, :in_front => true
```

Another Useless Benchmark

Real World Example

SecureFile

The next logical step?

- Gee, Mongrel without Rails is hella fast
- I'll start writing more and more of my apps as Handlers
- I'll start implementing the parts of Rails I need in my handlers...

Merb

Started as a hack, Merb == Mongrel + Erb

- No CGI.rb !!
- Clean room implementation of ActionPack
- Thread Safe with configurable Mutex Locks
- Rails compatible REST routing
- No Magic(well less anyway ;)
- It's what you will end up with if you keep writing custom Mongrel Handlers

Dispatching Rails vs Merb

Rails

Request comes in

****Mutex gets locked****

Parse CGI + Mime(expensive)

Route recognition(expensive)

Before Filter Chain

Call Controller Action

Render Template

****Mutex Unlocked****

Results returned to client

Merb(with ActiveRecord)

Request comes in

Parse CGI + Mime

Route recognition

****Mutex gets locked****

Before Filter Chain

Call Controller Action

Render Template

****Mutex Unlocked****

Results returned to client

Merb Hello World

```
class Hello < Application
  def world
    "Hello World!"
  end
end
```

```
Concurrency Level: 1
Time taken for tests: 2.205 seconds
Complete requests: 1000
Failed requests: 0
Broken pipe errors: 0
Total transferred: 211000 bytes
HTML transferred: 12000 bytes
Requests per second: 453.51 [#/sec] (mean)
Time per request: 2.20 [ms] (mean)
Time per request: 2.20 [ms] (mean, across all concurrent requests)
Transfer rate: 95.69 [Kbytes/sec] received
```

Routing

- Rails routing is 1800 lines of **very complex** non thread-safe code
- Merb's router is 200 lines of complex but **thread safe** code and much more efficient for route matching

```
puts "Compiling routes.."

Merb::RouteMatcher.prepare do |r|
  # restfull routes
  r.resources :posts
  # default route, usually you don't want to change this
  r.default_routes
  # change this for your home page to be avaiable at /
  r.add '/', :controller => 'hello', :action => 'world'
end

puts Merb::RouteMatcher.compiled_regexen|
puts Merb::RouteMatcher.compiled_statement
```

Why should you care?

- Rails has gotten 10-20% slower with each recent release
- Resident RAM usage per process gets larger as well
- Premature Optimization is blah, blah
- At this point it is not premature anymore ;)
- There is a point where optimization Matters

Merb's Mongrel Handler

Questions?