

Webmachine

a resource server for the Web

Justin Sheehy

Webmachine

a resource server for the Web

a toolkit for HTTP-based systems

Webmachine

a resource server for the Web

a toolkit for easily creating
well-behaved HTTP-based systems

Webmachine

a resource server for the Web

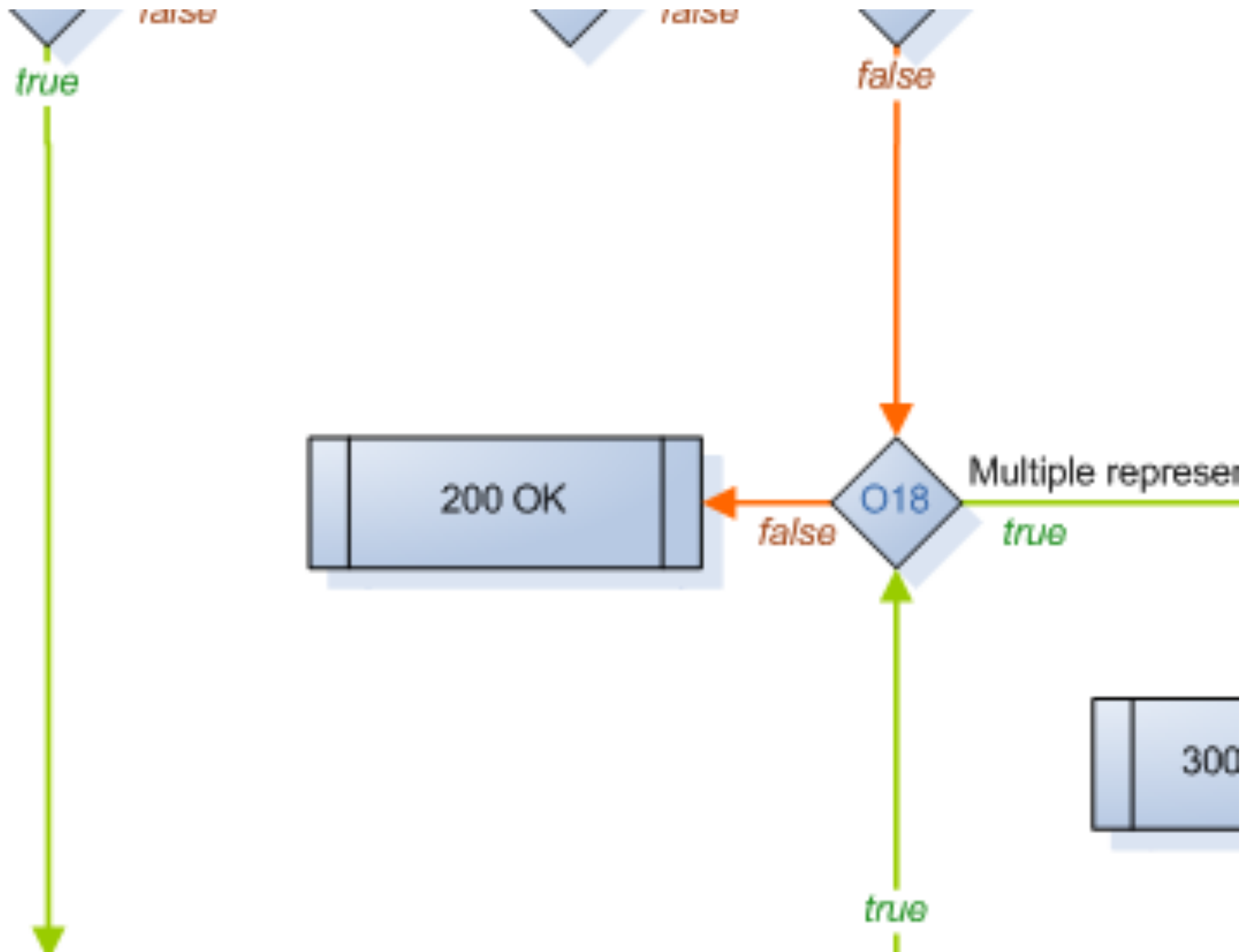
a toolkit for **easily creating?**
well-behaved HTTP-based systems

Webmachine

a resource server for the Web

a toolkit for easily creating
well-behaved? HTTP-based systems

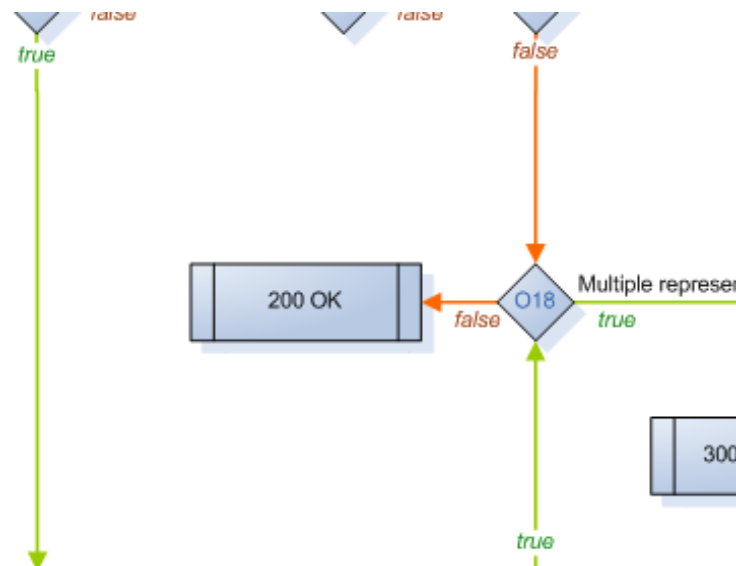
HTTP is complicated.



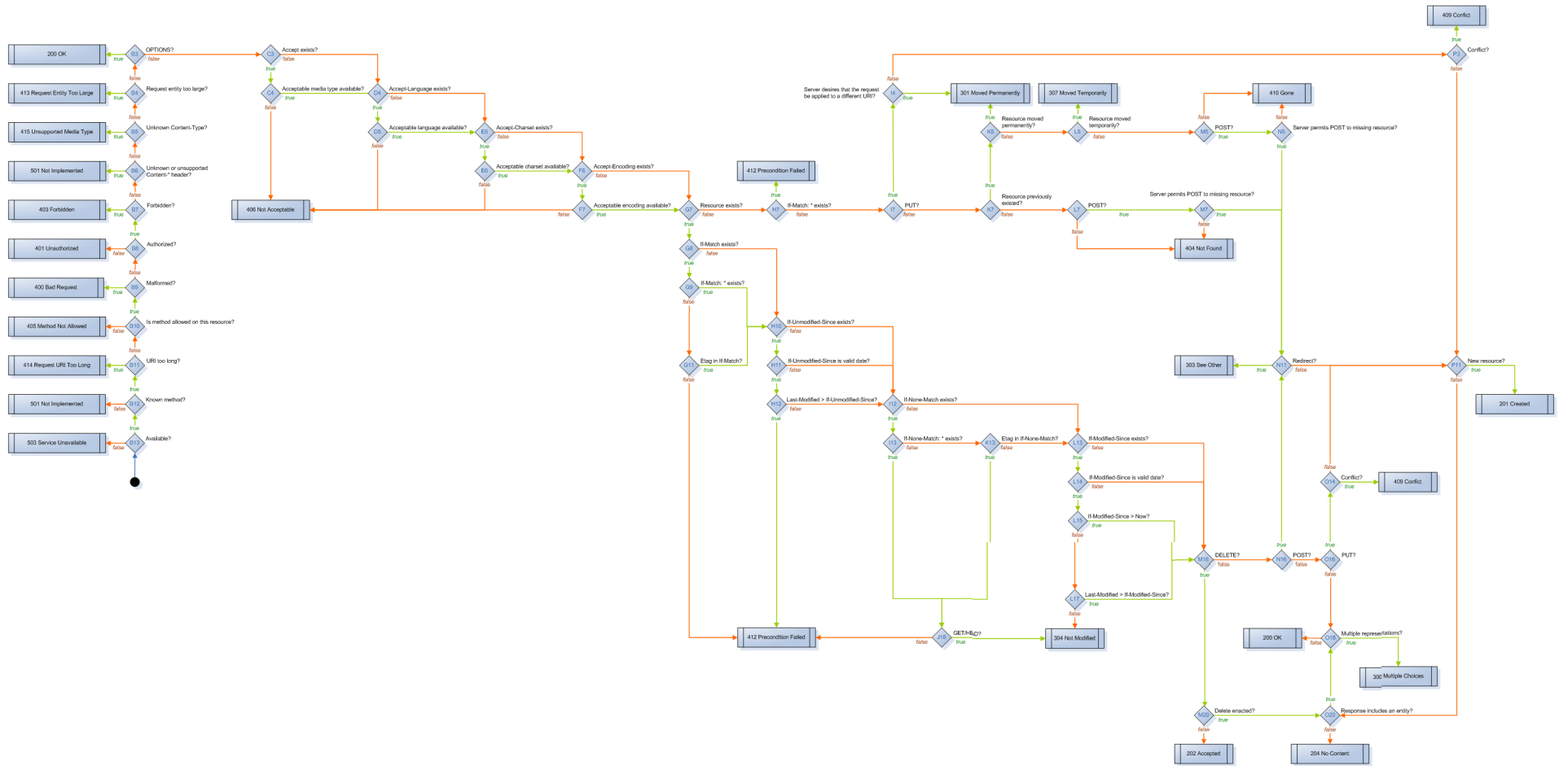
Webmachine makes HTTP easier.

```
-module(twohundred_resource).  
-export([init/1, to_html/2]).  
-include_lib("webmachine/include/webmachine.hrl").  
init([]) -> {ok, undefined}.
```

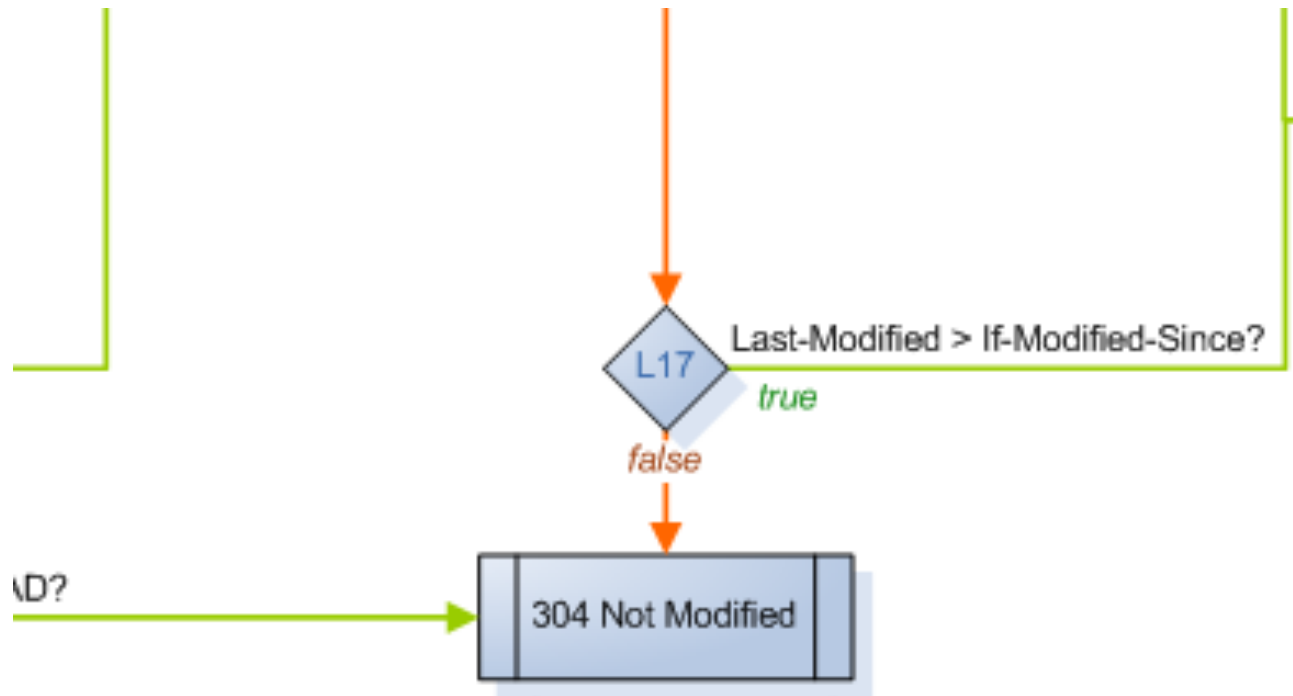
```
to_html(ReqData, State) ->  
    {"Hello, Webmachine world", ReqData, State}.
```



(that's it!)



Want to get more interesting?
 Just add generate_etag or last_modified...



Just add `generate_etag` or `last_modified`...
...and now you have conditional requests.

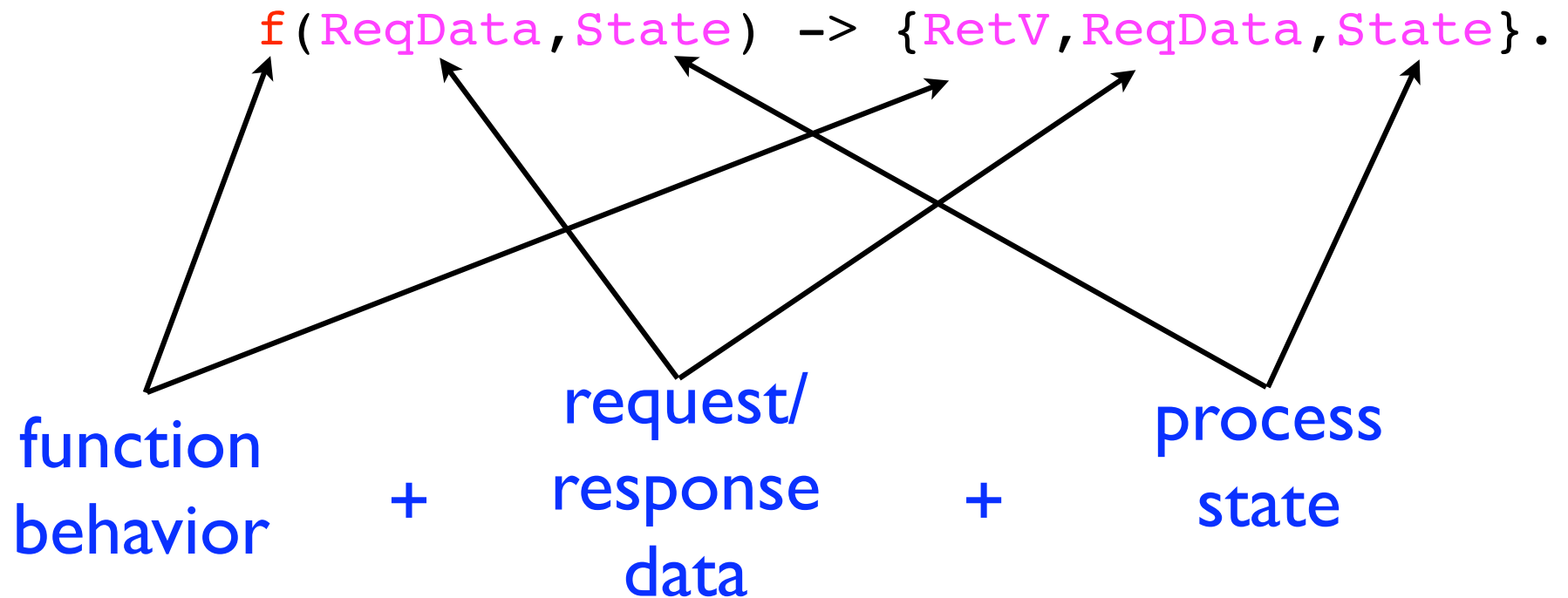
```
generate_etag(RD, State) ->  
  {mochihex:to_hex(erlang:phash2(State)), RD, State}.
```

```
last_modified(RD, State) ->  
  {filelib:last_modified(State#s.fpath), RD, State}.
```

A resource family is just a set of functions.

```
    to_html(ReqData, State) -> {Body, ReqData, State}.
generate_etag(ReqData, State) -> {ETag, ReqData, State}.
last_modified(ReqData, State) -> {Time, ReqData, State}.
resource_exists(ReqData, State) -> {bool, ReqData, State}.
is_authorized(ReqData, State) -> {bool, ReqData, State}.
    ... f(ReqData, State) -> {RetV, ReqData, State}.
```

A resource family is just a set of functions.



Resource functions are referentially transparent and have a uniform interface.

Manipulating Request/Response Data

`f (ReqData, State) -> {RetV, ReqData, State}.`

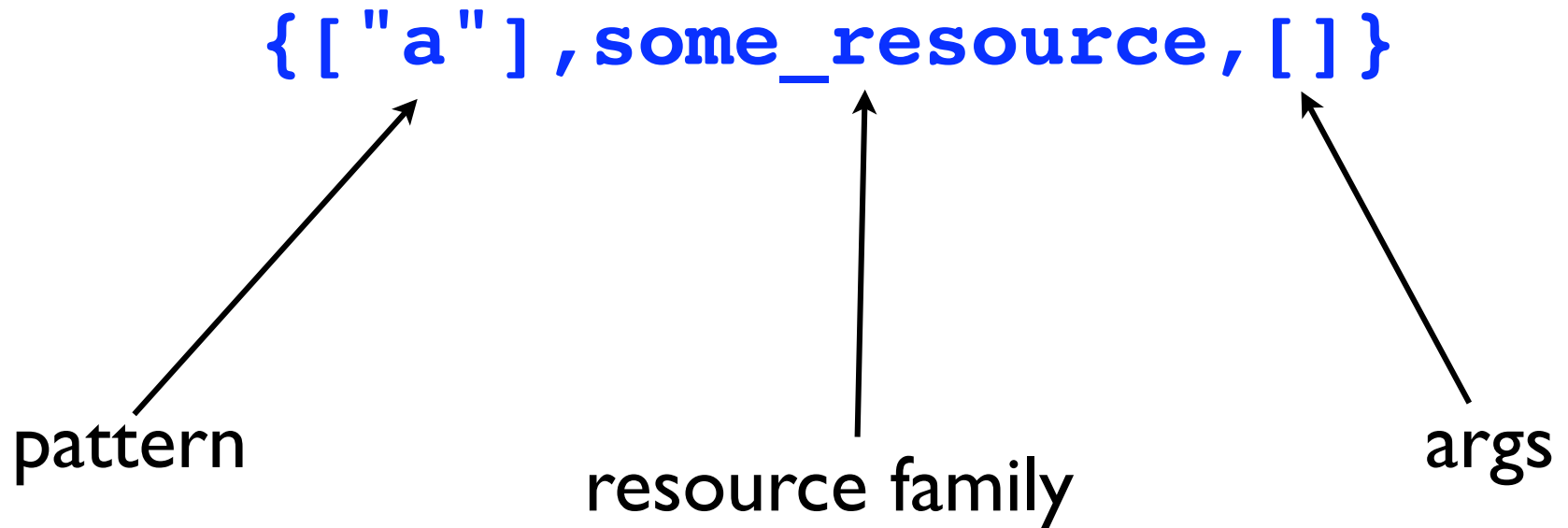
`wrq:get_req_header (HdrName, ReqData) -> 'undefined' | HdrVal`

`wrq:get_qs_value (Key, Default, ReqData) -> Value`

`wrq:set_resp_header (HdrName, HdrVal, ReqData) -> NewReqData`

The `wrq` module accesses and (nondestructively) modifies `ReqData`.

URL Dispatching = Pattern Matching



URL Dispatching = Pattern Matching

```
{ ["a"], some_resource, [] }
```

http://myhost/a → match!

any other URL → no match

If no patterns match, then 404 Not Found.

URL Dispatching = Pattern Matching

`{ ["a"], some_resource, [] }`

/a

`[]`

`"/a"`

`[]`

`[]`

`wrq:disp_path`

`wrq:path`

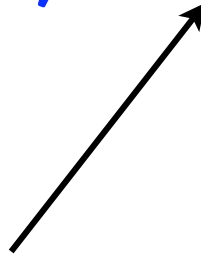
`wrq:path_info`

`wrq:path_tokens`

URL Dispatching = Pattern Matching

```
{["a", '*'], some_resource, []}
```

/a



(binds the remaining path)

```
[ ]  
"/a"  
[ ]  
[ ]
```

```
wrq:disp_path  
wrq:path  
wrq:path_info  
wrq:path_tokens
```

URL Dispatching = Pattern Matching

```
{["a", '*'], some_resource, []}
```

/a/b/c

"b/c"	wrq:disp_path
"/a/b/c"	wrq:path
[]	wrq:path_info
["b", "c"]	wrq:path_tokens

URL Dispatching = Pattern Matching

`{"a", foo}, some_resource, []`

`/a/b/c` → 404

(name-binds a path segment)

`["b/c"]` `wrq:disp_path`
`["/a/b/c"]` `wrq:path`
`[""]` `wrq:path_info`
`["b", "c"]` `wrq:path_tokens`

URL Dispatching = Pattern Matching

`{["a", foo], some_resource, []}`

`/a/b`

<code> []</code>	<code>wrq:disp_path</code>
<code> "/a/b"</code>	<code>wrq:path</code>
<code>[{foo, "b"}]</code>	<code>wrq:path_info</code>
<code> []</code>	<code>wrq:path_tokens</code>

URL Dispatching = Pattern Matching

```
{["a", foo, '*'], some_resource, []}
```

/a/b

[]	wrq:disp_path
"/a/b"	wrq:path
[{foo, "b"}]	wrq:path_info
[]	wrq:path_tokens

URL Dispatching = Pattern Matching

```
{["a", foo, '*'], some_resource, []}
```

/a/b/c/d

```
    "c/d"  
  "/a/b/c/d"  
[ {foo, "b"} ]  
  ["c", "d"]
```

```
wrq:disp_path  
wrq:path  
wrq:path_info  
wrq:path_tokens
```

URL Dispatching = Pattern Matching

```
{ ["a", foo, '*'], some_resource, [] }
```

/a/b/c/d?fee=ah&fie=ha

query strings are easy too

`wrq:get_qs_value("fie",ReqData) -> "ha"`

`"c/d"`

`"/a/b/c/d"`

`[{foo, "b"}]`

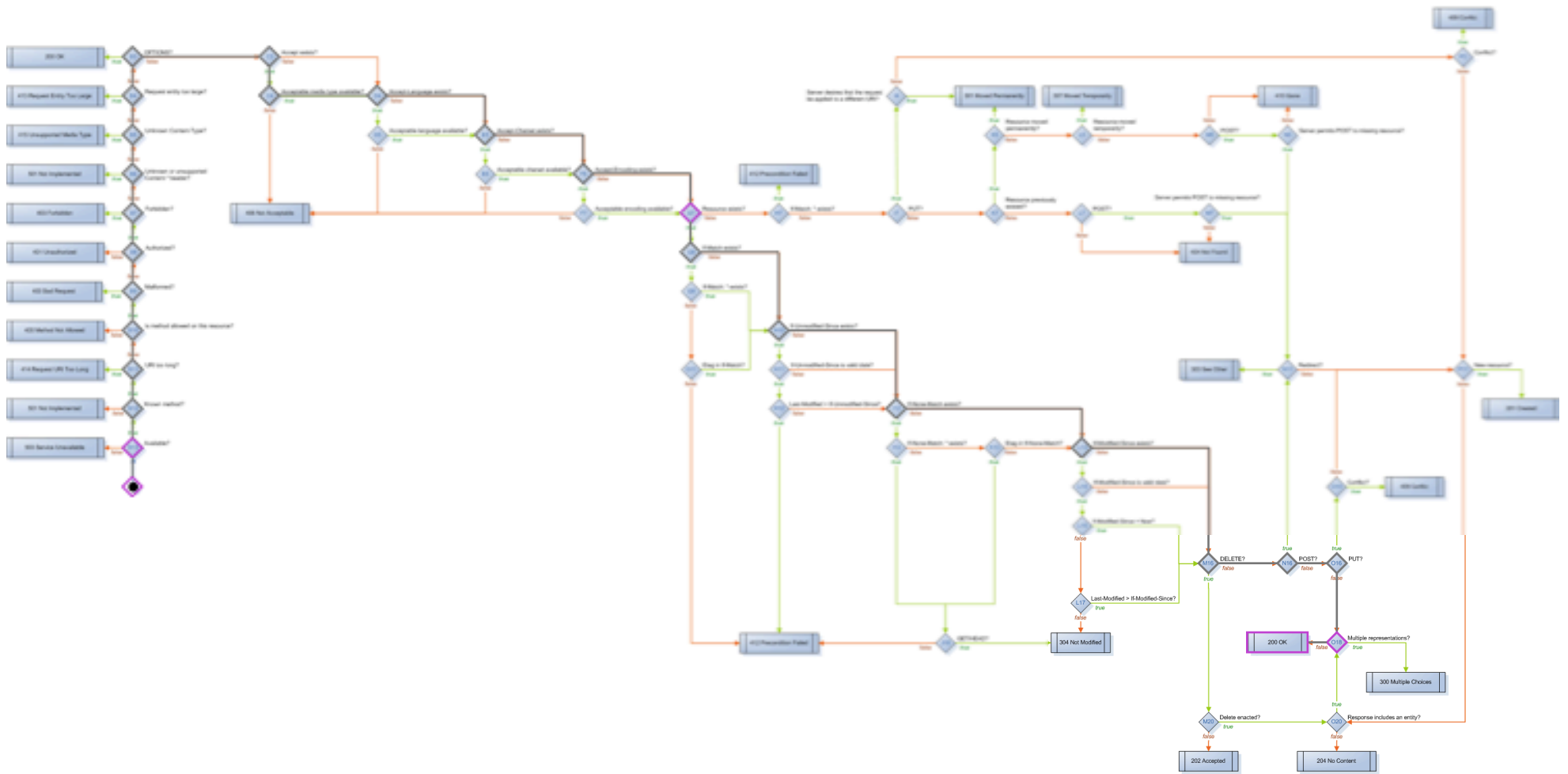
`["c", "d"]`

`wrq:disp_path`

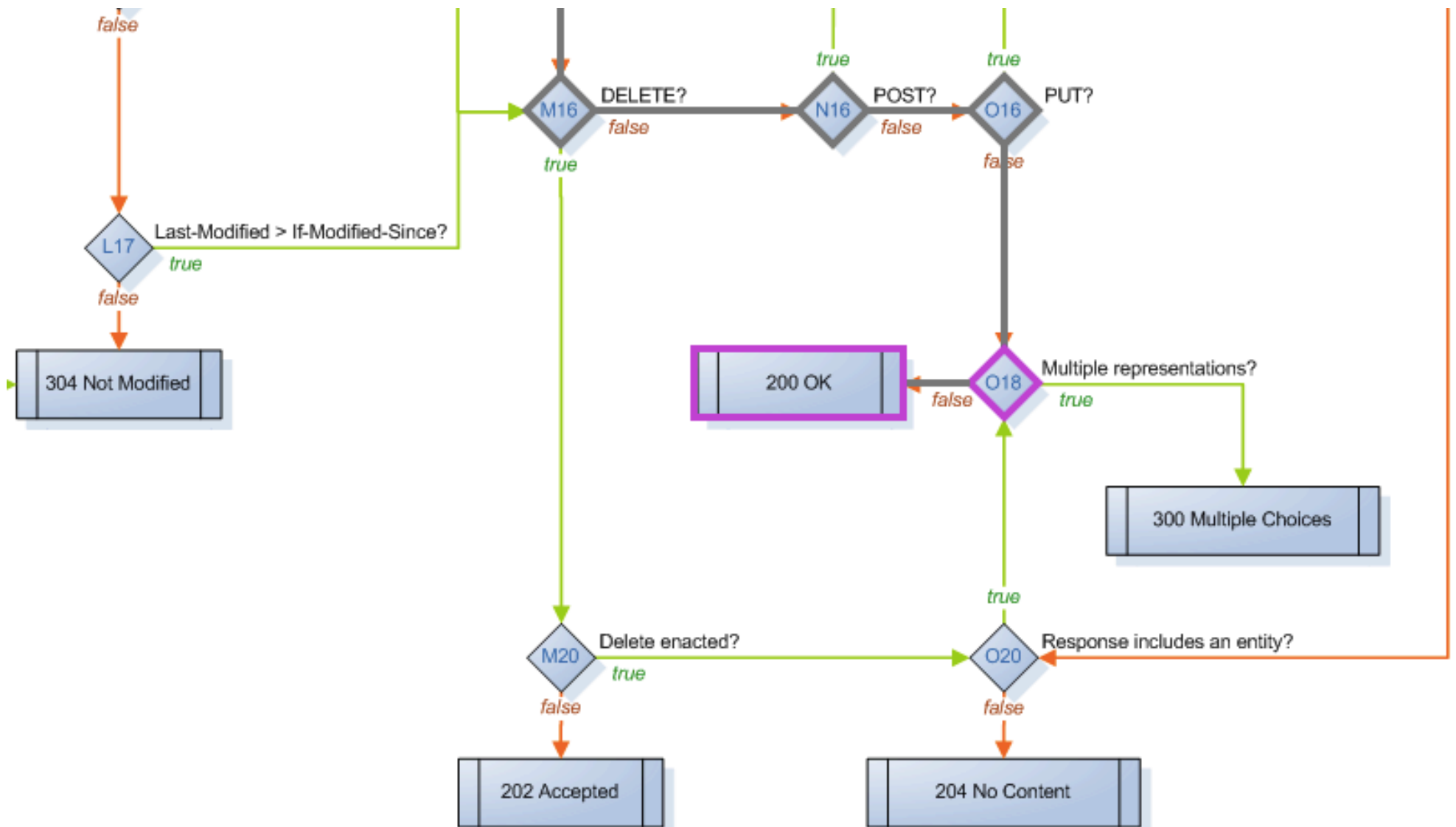
`wrq:path`

`wrq:path_info`

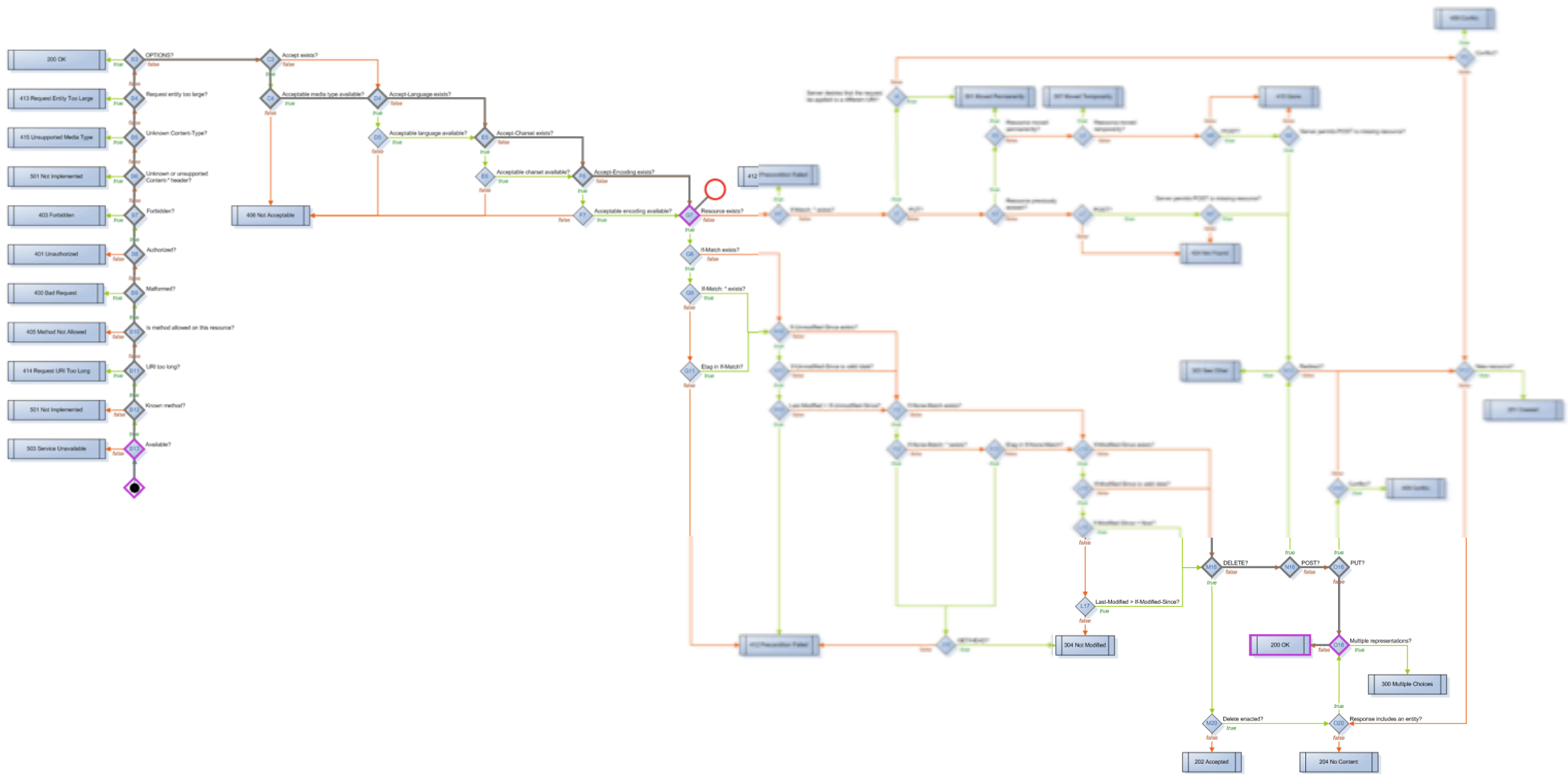
`wrq:path_tokens`



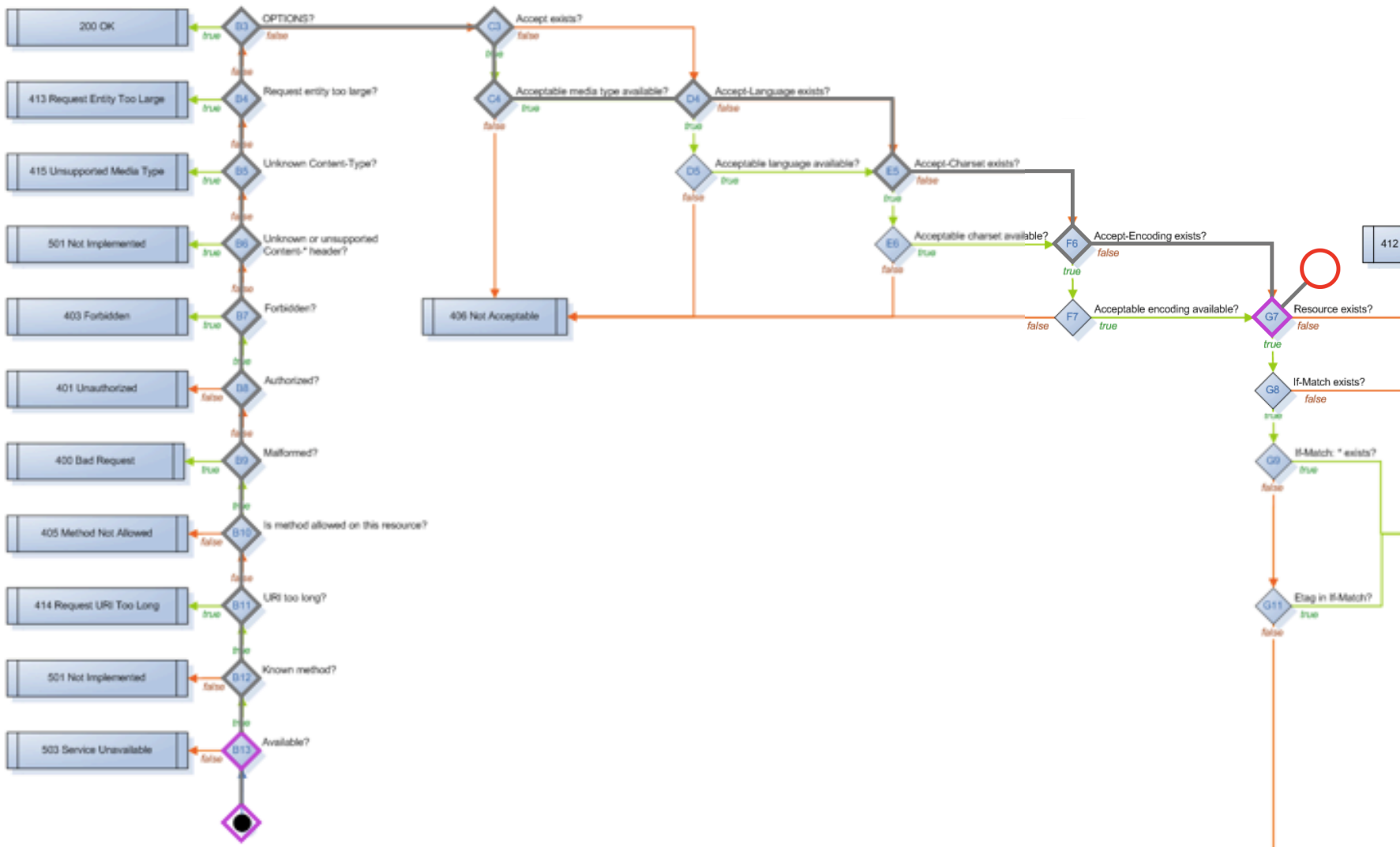
The Webmachine Visual Debugger



Hooray!

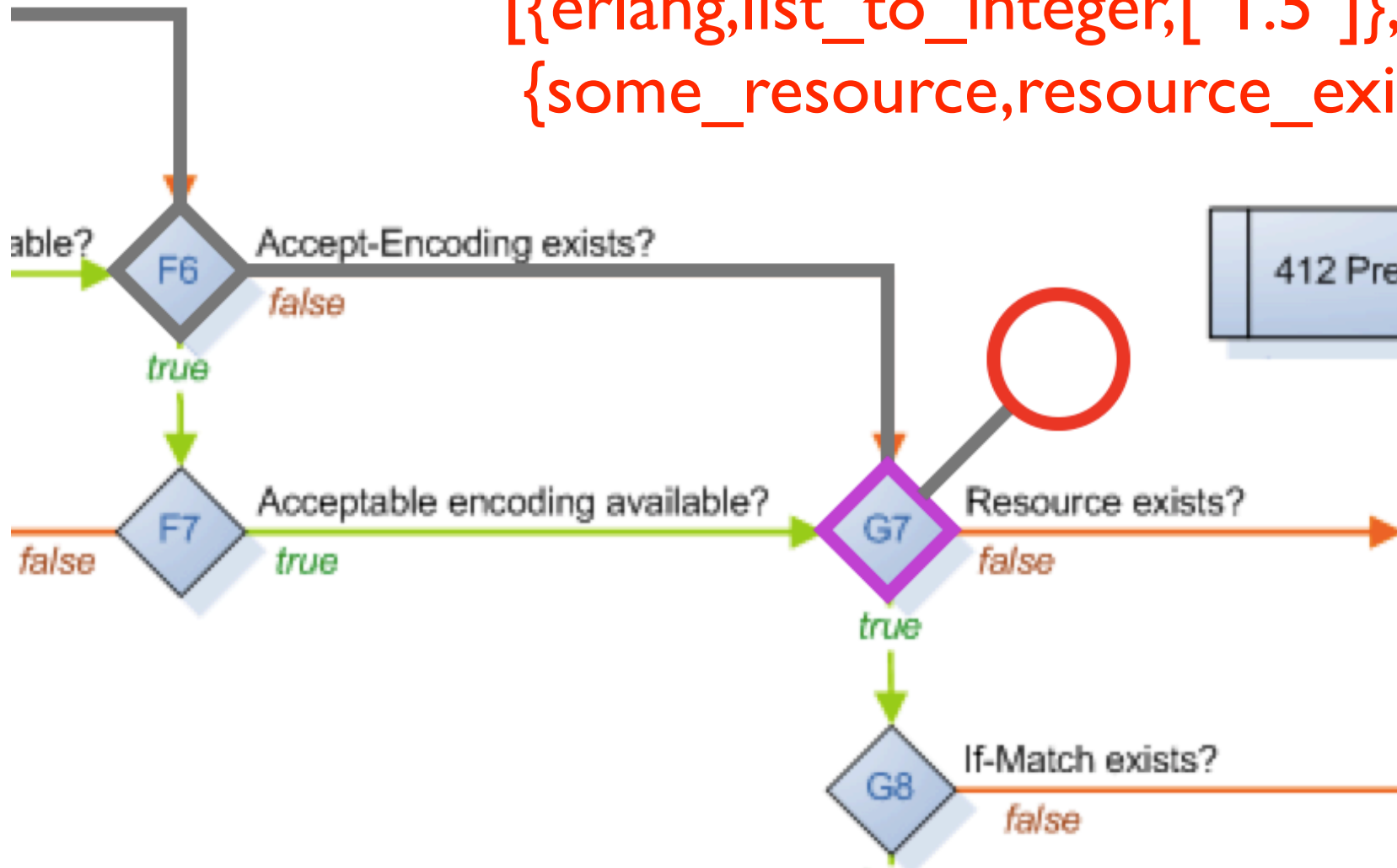


But sometimes things don't go as well.



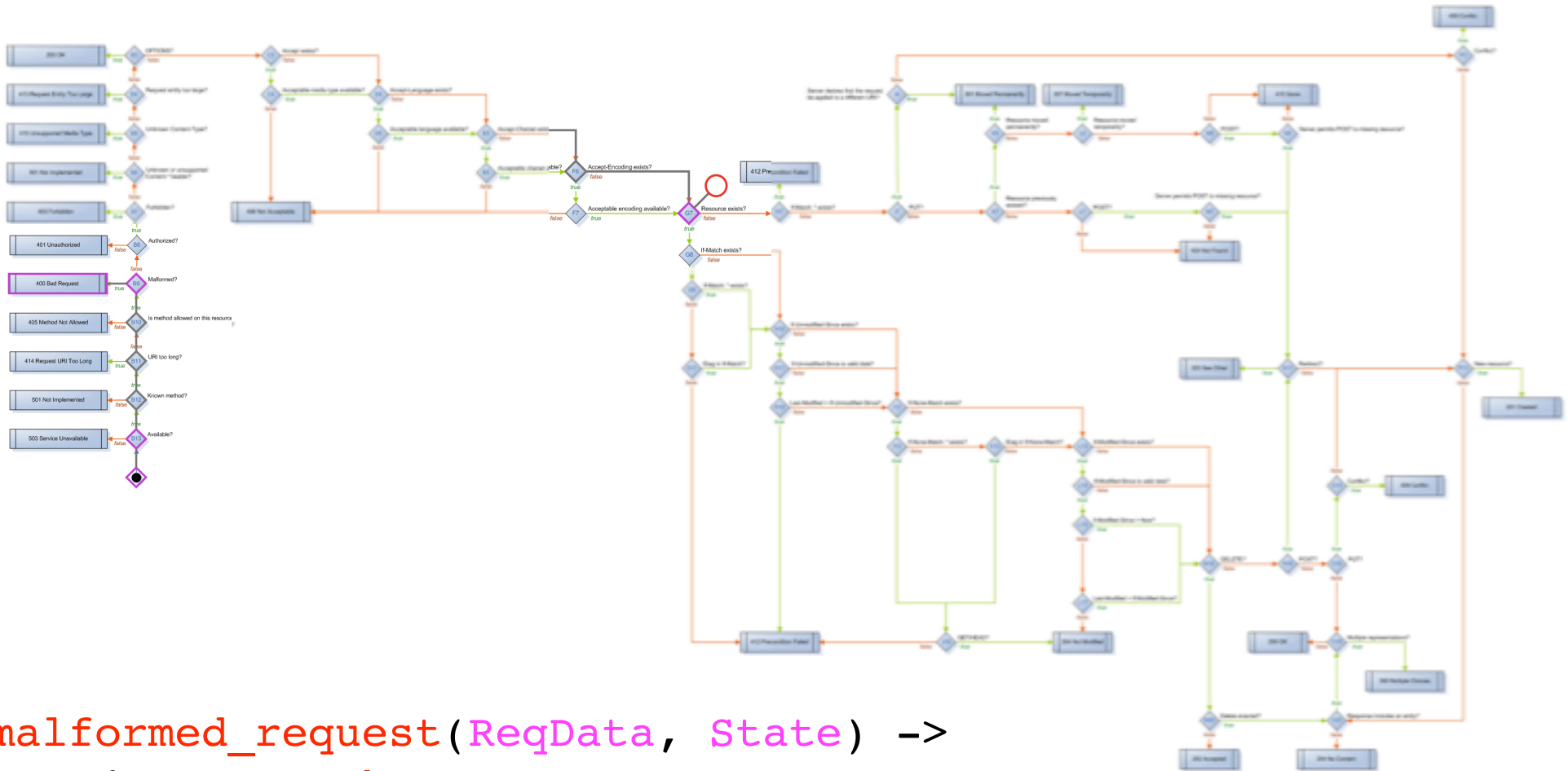
It's nice to know where your errors are.

`{{error,{error,badarg,`
`[{erlang,list_to_integer,["1.5"]},`
`{some_resource,resource_exists,2}...`

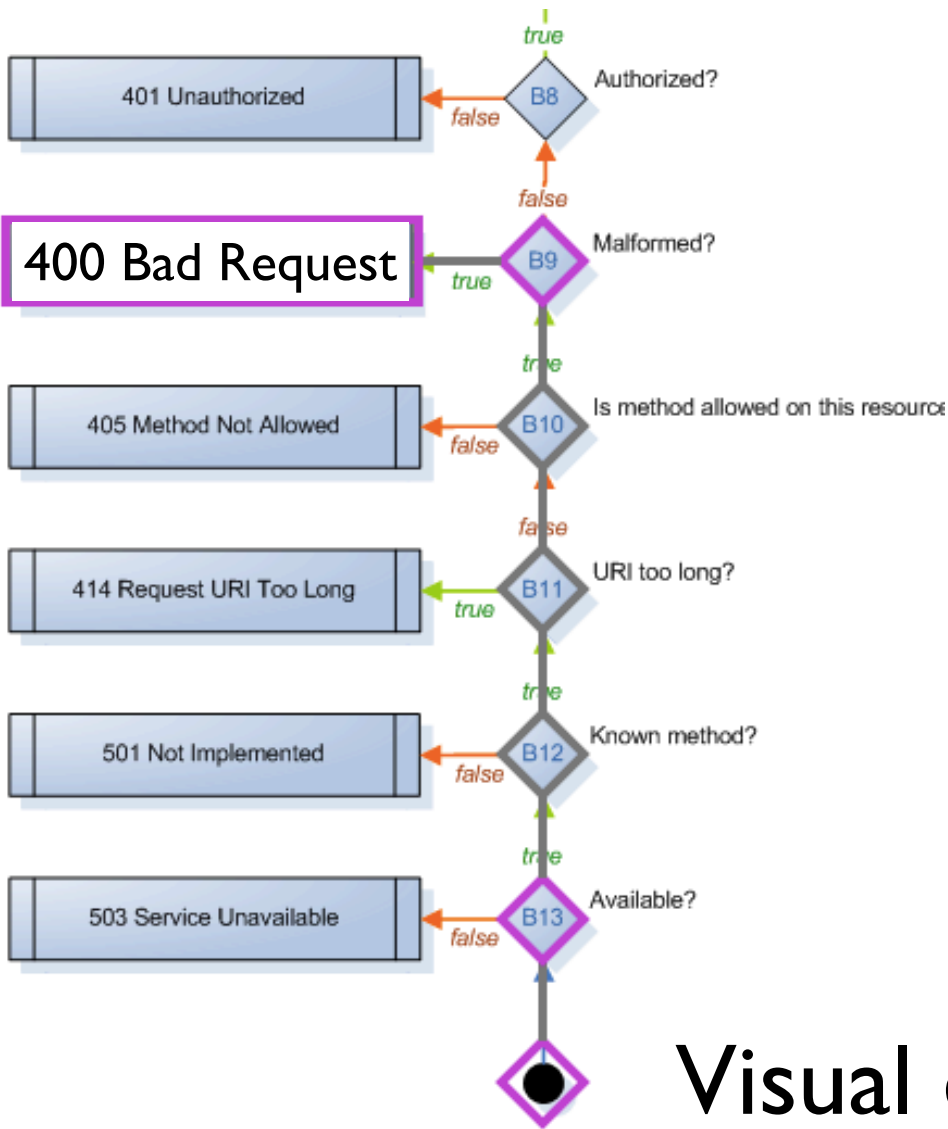


`wrq:path(RD) -> "/d/test?q=1.5"`

-export ([malformed_request/2]).

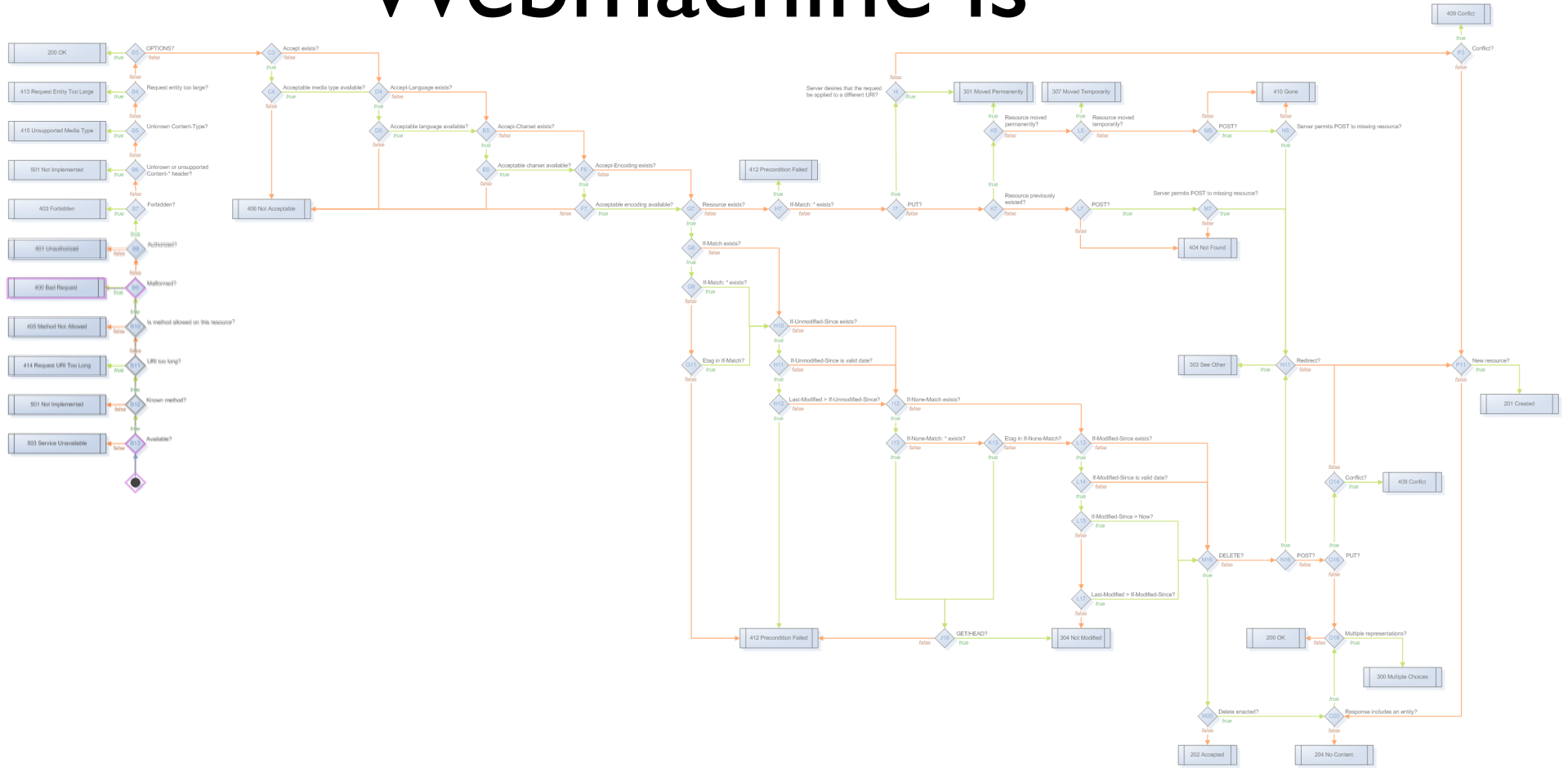


```
malformed_request(ReqData, State) ->
{case catch
  list_to_integer(wrq:get_qs_value("q", "0", ReqData)) of
    {'EXIT', _} -> true;
    _ -> false
end,
ReqData, State}.
```



Visual debugging helps you put the fixes in the right place.

Webmachine is



a higher-level abstraction for HTTP.

Webmachine is not a “framework.”

No built-in templating, no ORM or built-in storage.

Webmachine is a good component in a framework.

**Webmachine is not
an all-purpose network server.**

No support for continuous streaming, etc.

Webmachine is shaped like HTTP.

**Webmachine is
a resource server for the Web.**

**A toolkit for easily creating
well-behaved HTTP systems.**

**Webmachine is
a resource server for the Web.**

**A toolkit for easily creating
well-behaved HTTP systems.**

<http://bitbucket.org/justin/webmachine/>

<http://blog.thereastfulway.com/>