Webmachine

a practical executable model for HTTP

Steve Vinoski

Architect, Basho Technologies

QCon SF 2011

16 Nov 2011

@stevevinoski

http://steve.vinoski.net/

vinoski@ieee.org

Webmachine

a practical executable model for HTTP

a toolkit for HTTP-based systems

Webmachine

a practical executable model for HTTP

a toolkit for

HTTP-based systems

Webmachine

a practical executable model for HTTP

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

Webmachine

a practical executable model for HTTP

a toolkit for easily creating?

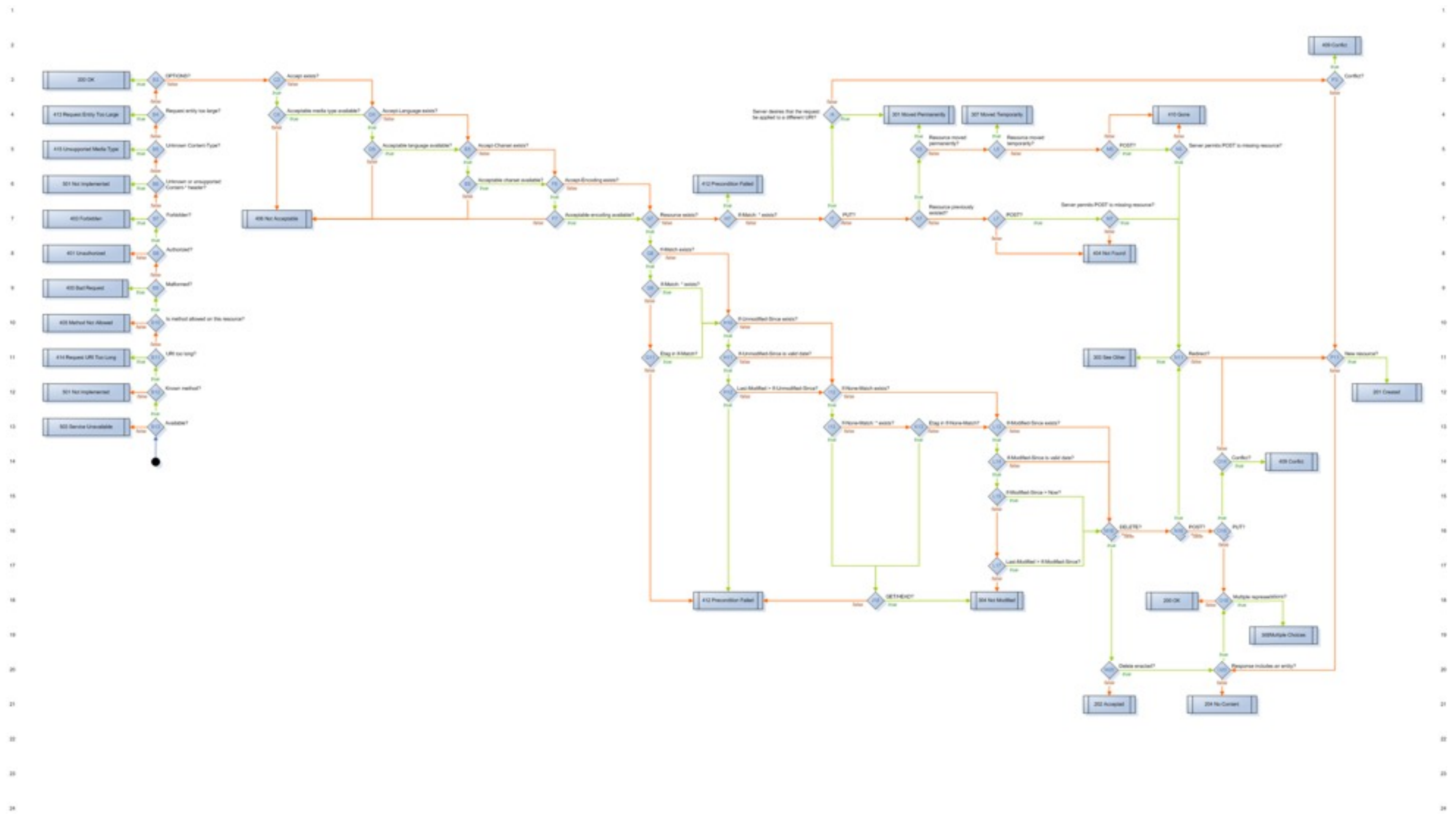
well-behaved HTTP-based systems

Webmachine

a practical executable model for HTTP

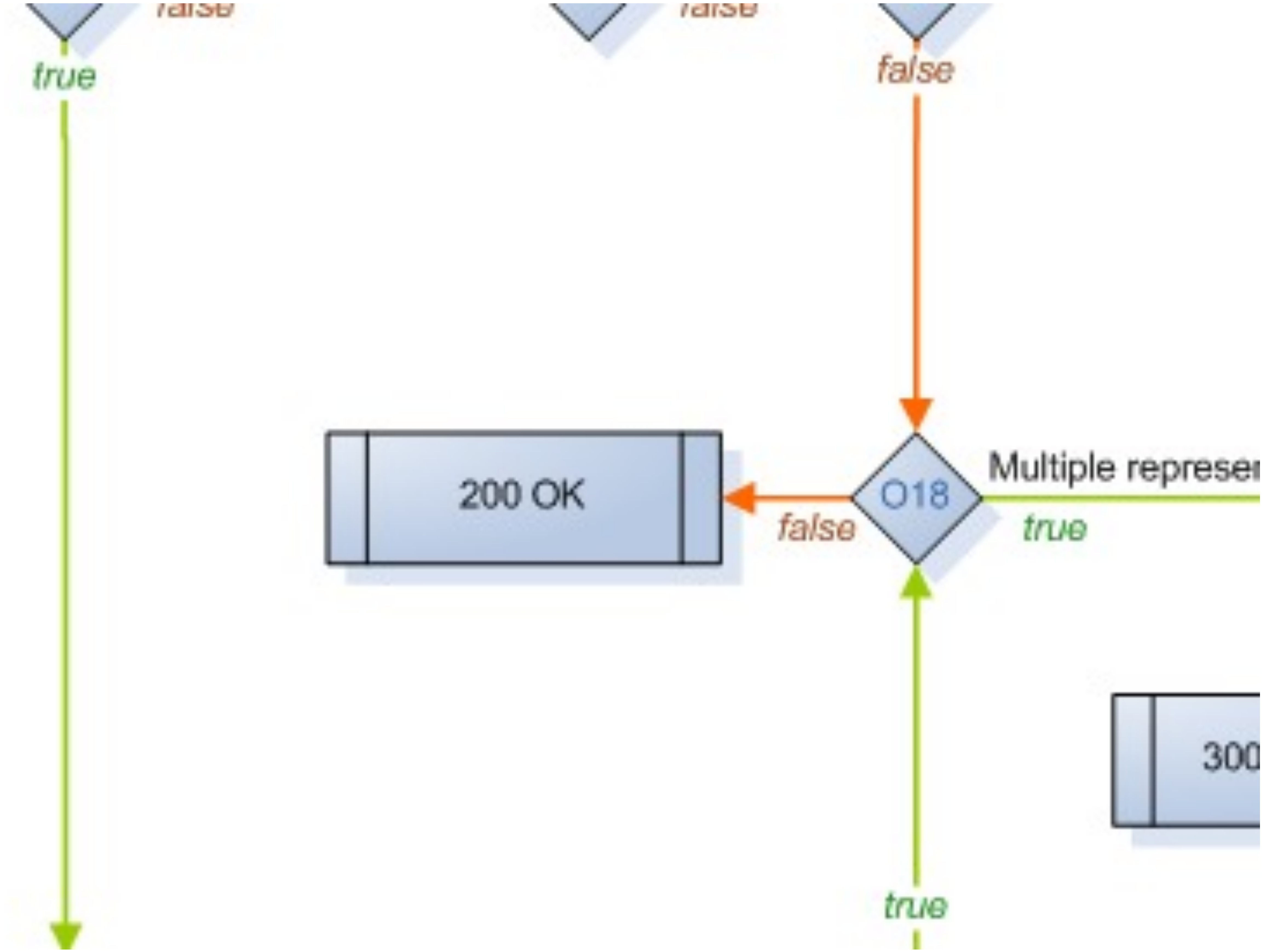
a toolkit for easily creating

well-behaved?HTTP-based systems



HTTP is complicated.

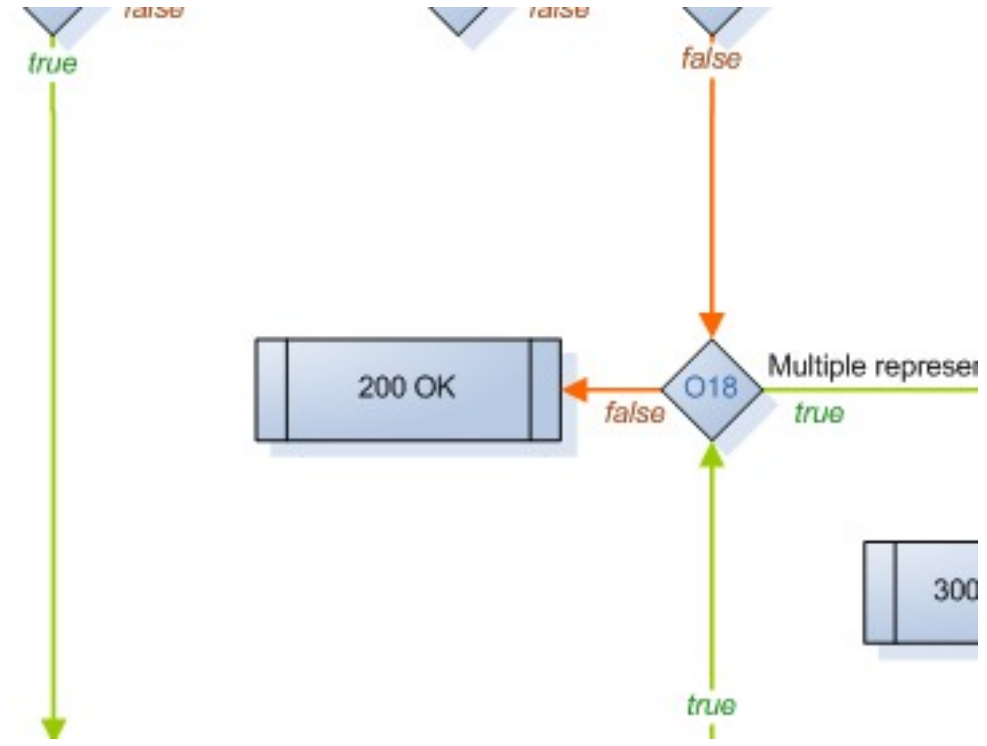
(see http://webmachine.basho.com/)



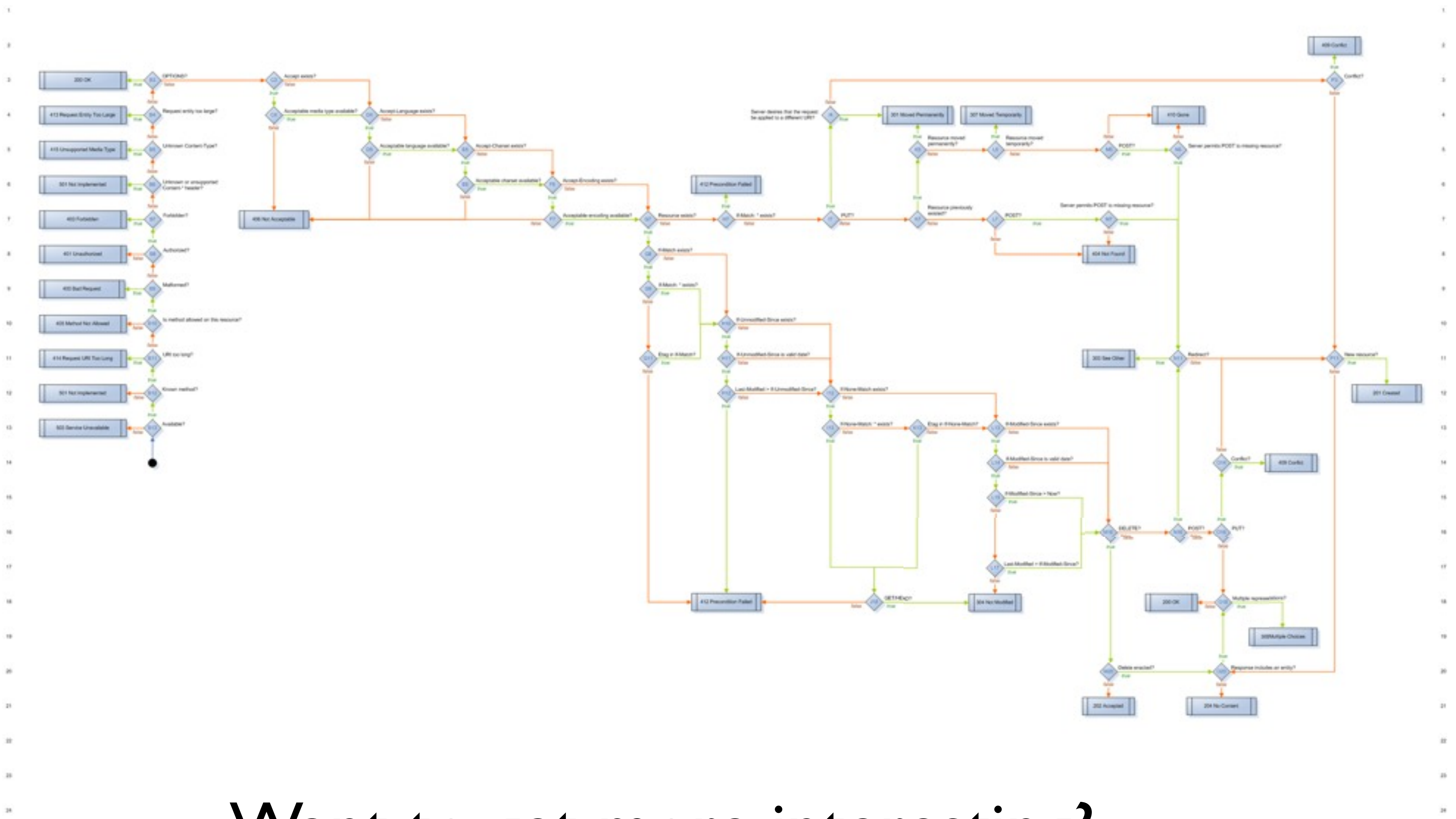
Webmachine makes HTTP easier.

```
-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}.
```

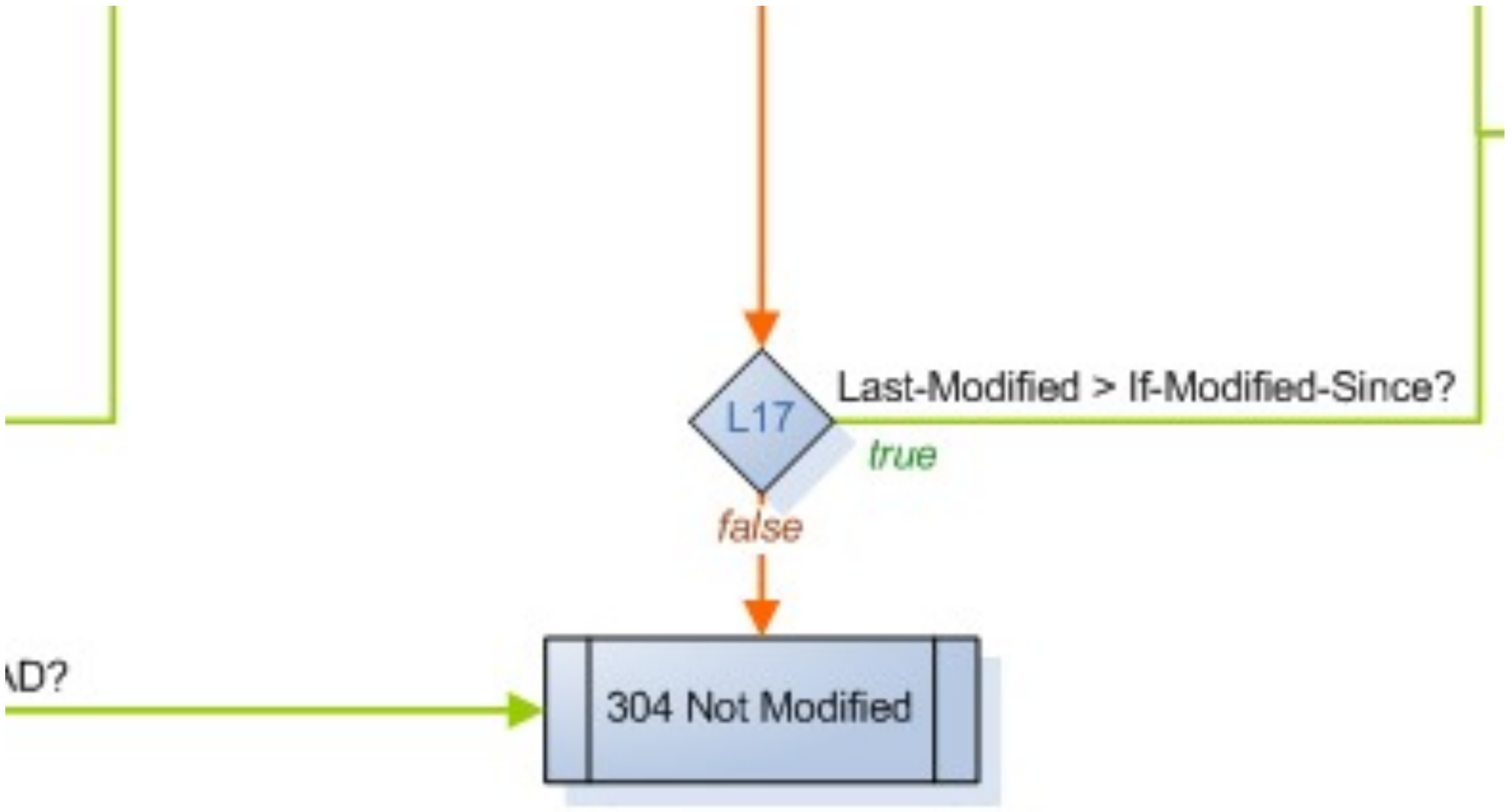


(that's it!)



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.

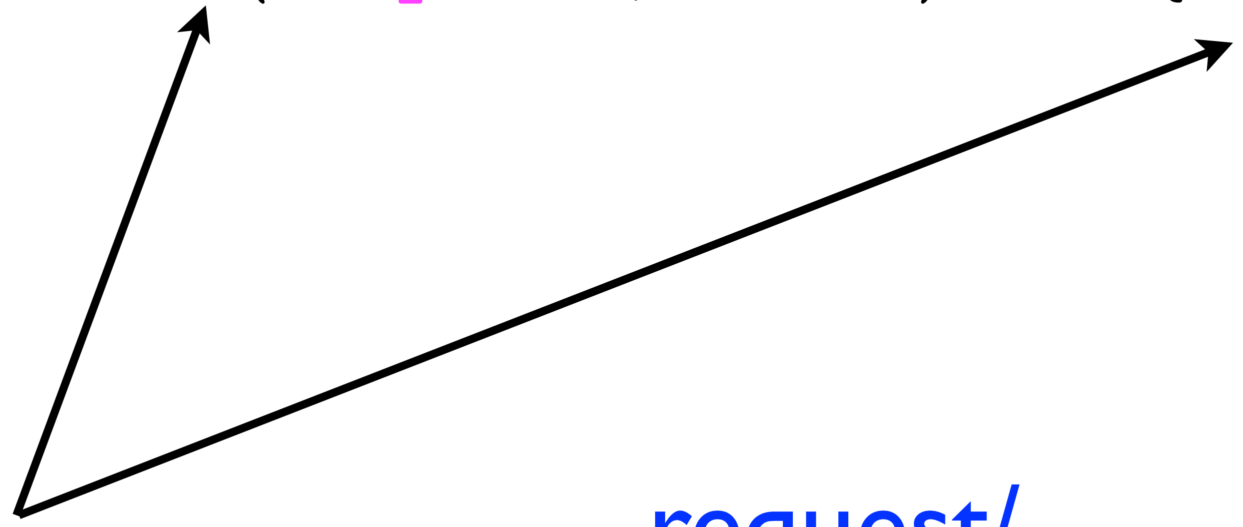
```
f(ReqData,State) -> {RetV,ReqData,State}.
```

function behavior + request/response data + process state

Resource functions are referentially transparent and have a uniform interface.

A resource family is just a set of functions.

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



function behavior + request/response data + process state

Resource functions are referentially transparent and have a uniform interface.

A resource family is just a set of functions.

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

function behavior + request/response data + process state

Resource functions are referentially transparent and have a uniform interface.

A resource family is just a set of functions.

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

function behavior + request/ response data + process state

Resource functions are referentially transparent and have a uniform interface.

A resource family is just a set of functions.

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

function behavior + request/response data + process state

Resource functions are referentially transparent and have a uniform interface.

Manipulating Request/Response Data

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

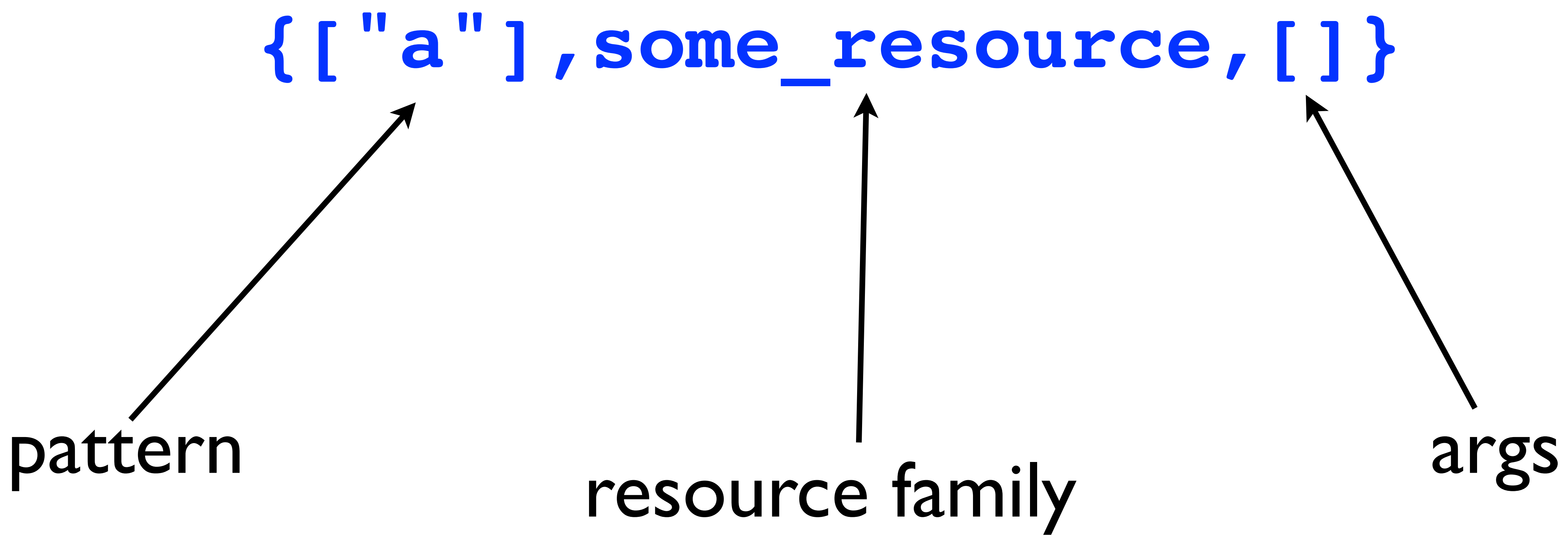
```
wrq:get_req_header(HdrName,ReqData) -> 'undefined' | HdrVal

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



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

URL Dispatching = Pattern Matching

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

$$\texttt{http://myhost/a} \xrightarrow{\qquad\qquad} \text{match!}$$

$$\texttt{any other URL} \xrightarrow{\qquad\qquad} \text{no match}$$

If no patterns match, then `404 Not Found`.

URL Dispatching = Pattern Matching

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

/a

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

URL Dispatching = Pattern Matching

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

/a

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

URL Dispatching = Pattern Matching

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

```
/a
```

[ ]	wrq:disp_path
"/a"	wrq:path
[ ]	wrq:path_info
[ ]	wrq:path_tokens

URL Dispatching = Pattern Matching

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

```
/a
```

(binds the remaining path)

[ ]	wrq:disp_path
"/a"	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
```

`[]`	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
```

`[]`	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

`[]`	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”`	wrq:disp_path
`"/a/b/c/d"`	wrq:path
`[{foo, “b”}]`	wrq:path_info
`[“c”,”d”]`	wrq:path_tokens

URL Dispatching = Pattern Matching

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

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

`“c/d”`	wrq:disp_path
`"/a/b/c/d"`	wrq:path
`[{foo, “b”}]`	wrq:path_info
`[“c”,”d”]`	wrq:path_tokens

URL Dispatching = Pattern Matching

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

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

`"c/d"`	wrq:disp_path
`"/a/b/c/d"`	wrq:path
`[{foo, "b"}]`	wrq:path_info
`["c","d"]`	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”	wrq:disp_path
"/a/b/c/d"	wrq:path
[{foo, “b”}]	wrq:path_info
[“c”,”d”]	wrq:path_tokens

An Example: Wriaki

- A wiki built on Webmachine and Riak
- Written by Bryan Fink of Basho as a sample application for Riak
- Wriaki is simple and elegant
- https://github.com/basho/wriaki

Wriaki Web Resources

- *User resources* represent wiki users
- *Articles* represent wiki pages
- *Archives* represent individual page versions
- *History* represents a page's history
- *Sessions* track user logins

Wriaki Dispatch Map

```
{["wiki"],      redirect_resource, "/wiki/Welcome"}.
{["wiki",'*'], wiki_resource,      []}.
{[],            redirect_resource, "/wiki/Welcome"}.

{["user"],                 login_form_resource, []}.
{["user",name],            user_resource,       []}.
{["user",name,session], session_resource,    []}.

{["static",'*'], static_resource, "www"}.
```

Wriaki Dispatch Map

```
{["wiki"], redirect_resource, "/wiki/Welcome"}.
```

- The *pathspec* declares the path we want to match

Wriaki Dispatch Map

```
{["wiki"], redirect_resource, "/wiki/..."}.
```

- The *resource module* declares which Erlang module implements the resource

Wriaki Dispatch Map

```
{[...],   redirect_resource, "/wiki/Welcome"}.
```

- *Args* is a list that Webmachine provides as the argument to the resource module's init function upon dispatching
- (In Erlang, a string is a list)

Redirect Resource

```
{["wiki"],      redirect_resource, "/wiki/Welcome"}.
{[],            redirect_resource, "/wiki/Welcome"}.
```

- Dispatch target for paths `/` and `/wiki`
- Aliases those paths to `/wiki/Welcome`

Redirect Init

```
init(Target) ->
    {ok, Target}.
```

- Called whenever a request is dispatched to the redirect resource
- Returns its argument as state for the request handling process
- Argument comes from dispatch map

Redirection

```
moved_permanently(RD, Target) ->
    {{true, Target}, RD, Target}.
```

- Effects redirection (HTTP status 301)
- Returns true with redirected path
- Redirect path is the process state returned from init
- Location K5 on Webmachine flow diagram

Wiki Pages

- Implemented by `wiki_resource` module
- Pages must be readable (of course)
- Must also accept POSTs for editing

Wiki Page Init

```
init([]) ->
    {ok, Client} = wrc:connect(),
    {ok, #ctx{client = Client}}.
```

- Called whenever a request is dispatched to a wiki page
- Returns a `#ctx` record as process state
- Record holds connection to Riak database where page data, user info, etc. are stored

Allowed Methods

```
allowed_methods(RD, Ctx) ->
    {['HEAD','GET','POST','PUT'],
       RD, Ctx}.
```

- Tells Webmachine what methods a wiki page allows
- `POST` and `PUT` allow writes
- Webmachine default allows only `HEAD` and `GET`

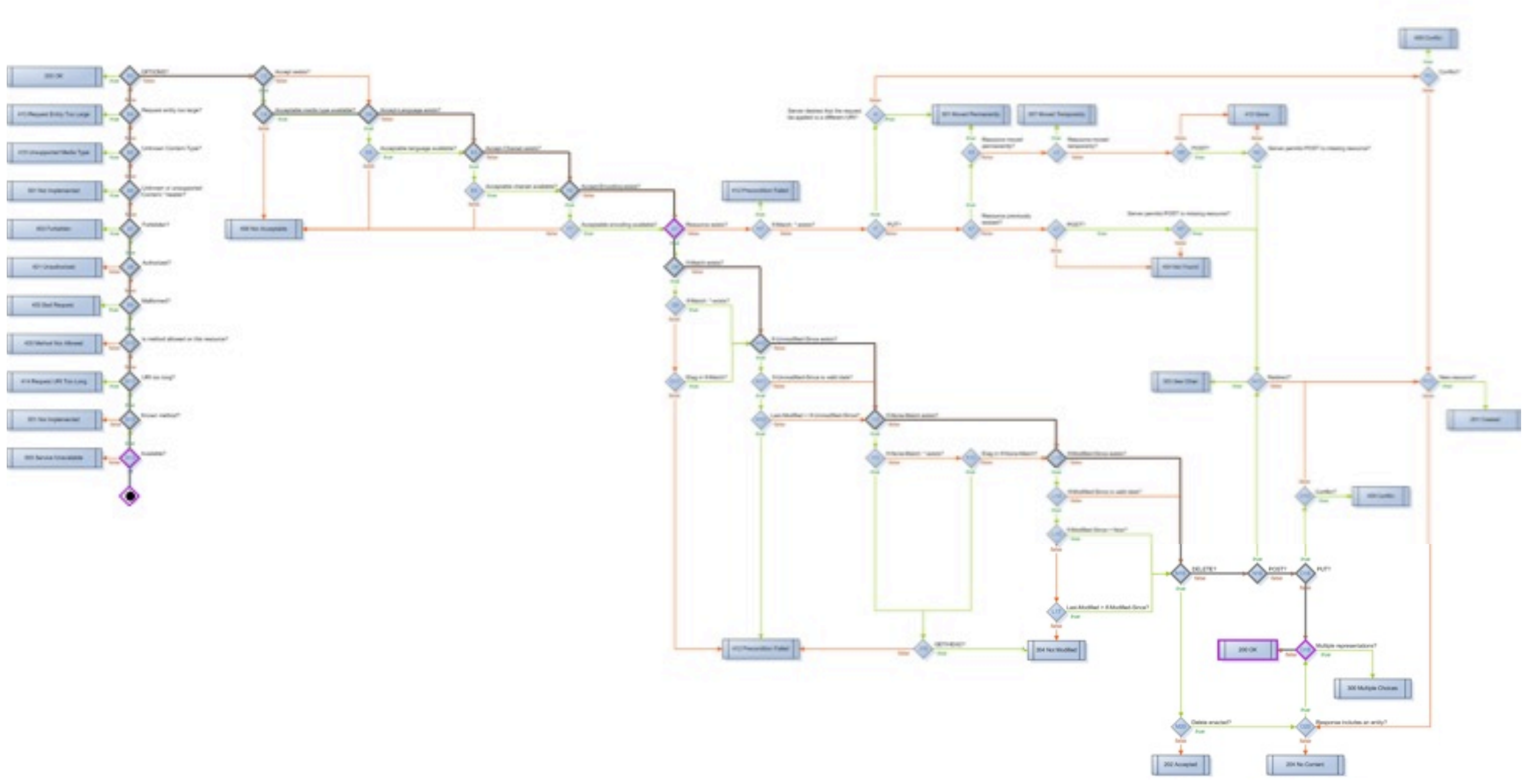
Accepted Content

```
content_types_accepted(RD, Ctx) ->
 MT =
  "application/x-www-form-urlencoded",
 {[{MT, accept_form}], RD, Ctx}.
```

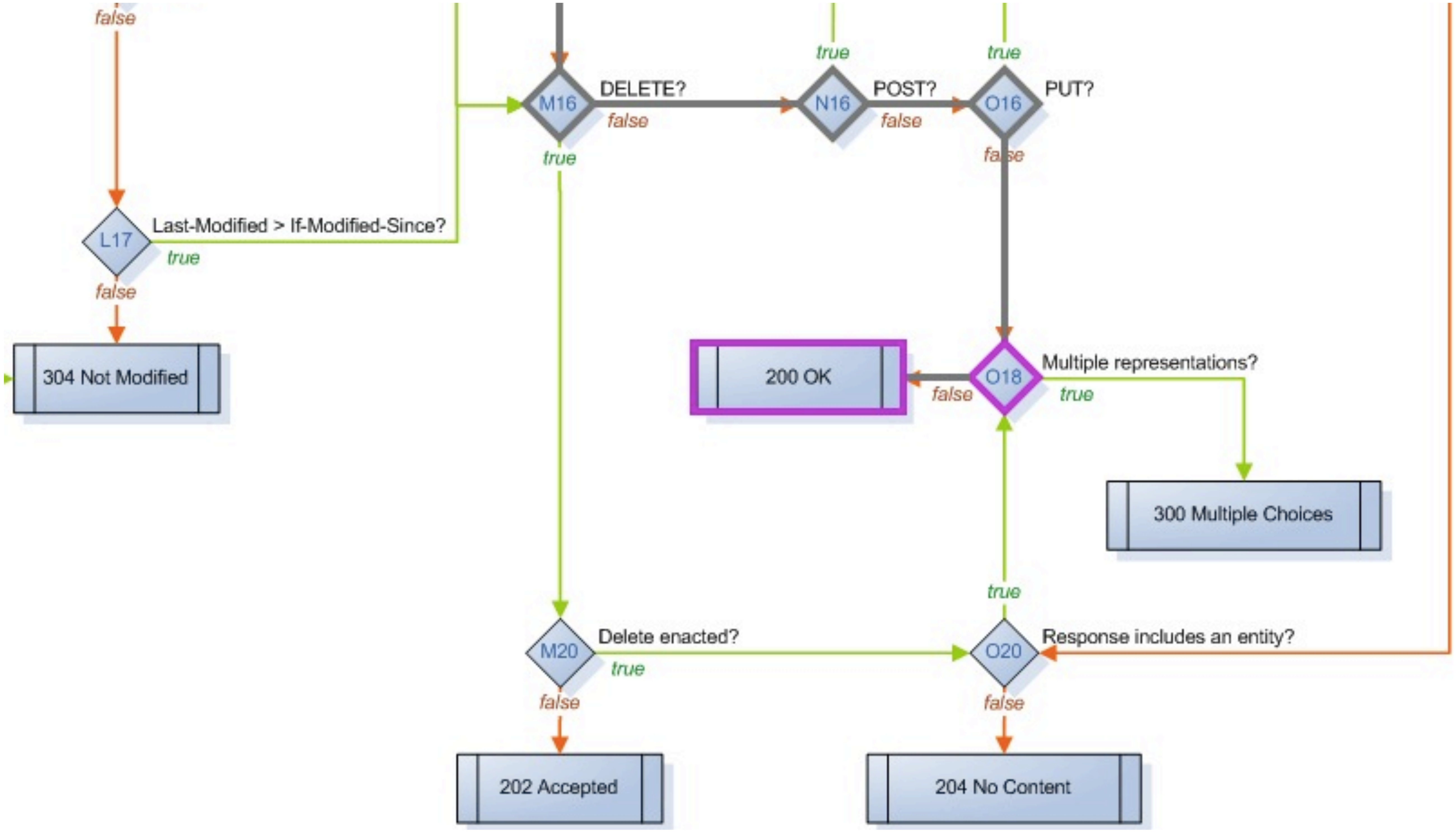
- Tells Webmachine what content (MIME types) a wiki page allows
- Also what function to call to process each MIME type
- Call `accept_form` to handle URL-encoded data

Simplicity

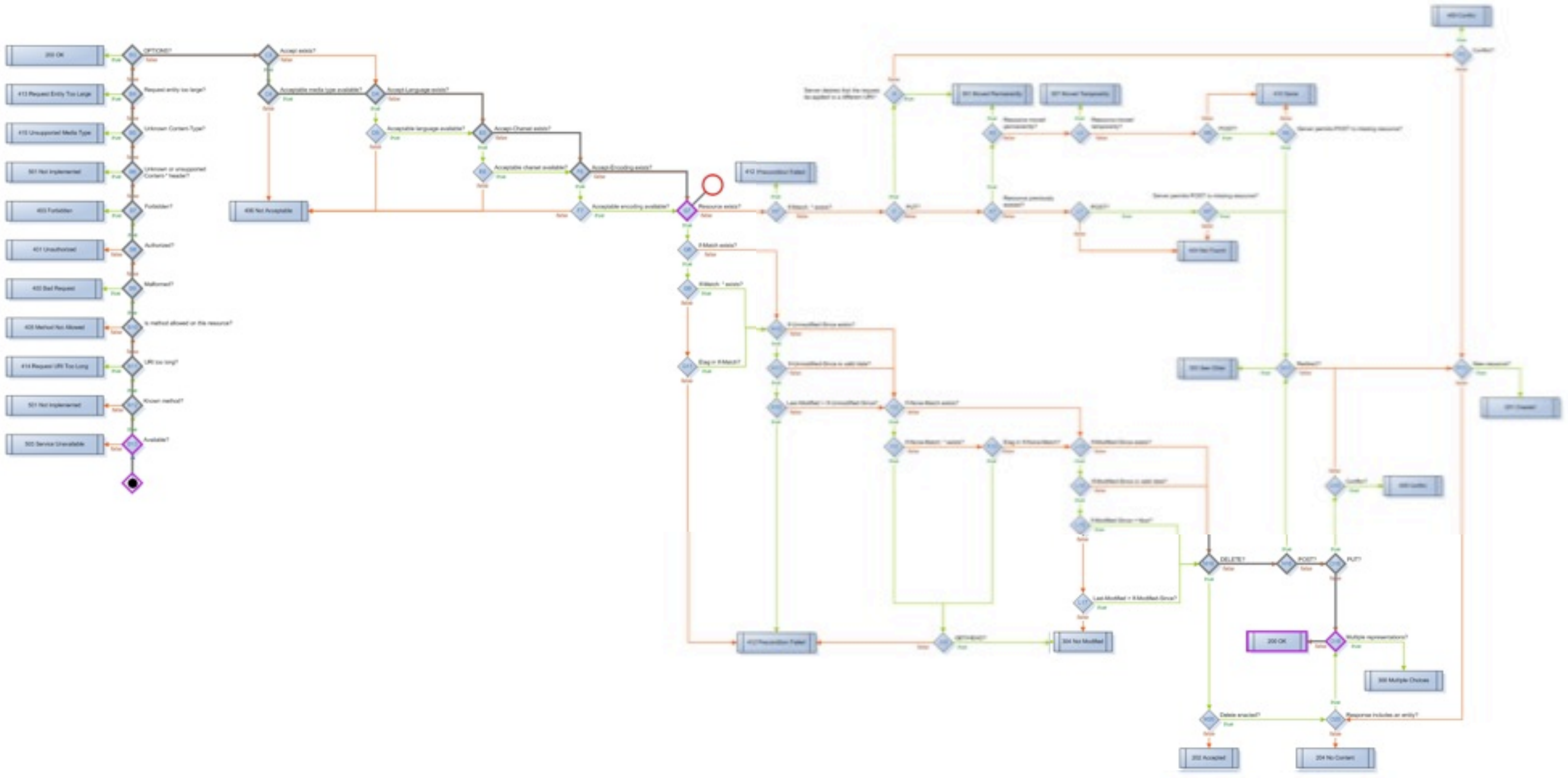
- Identify the resource functions specific to each resource module
- Webmachine provides reasonable defaults for all resource functions
- Take advantage of Erlang's "let it crash" philosophy
  - avoid reams of (buggy, incomplete) error-handling code



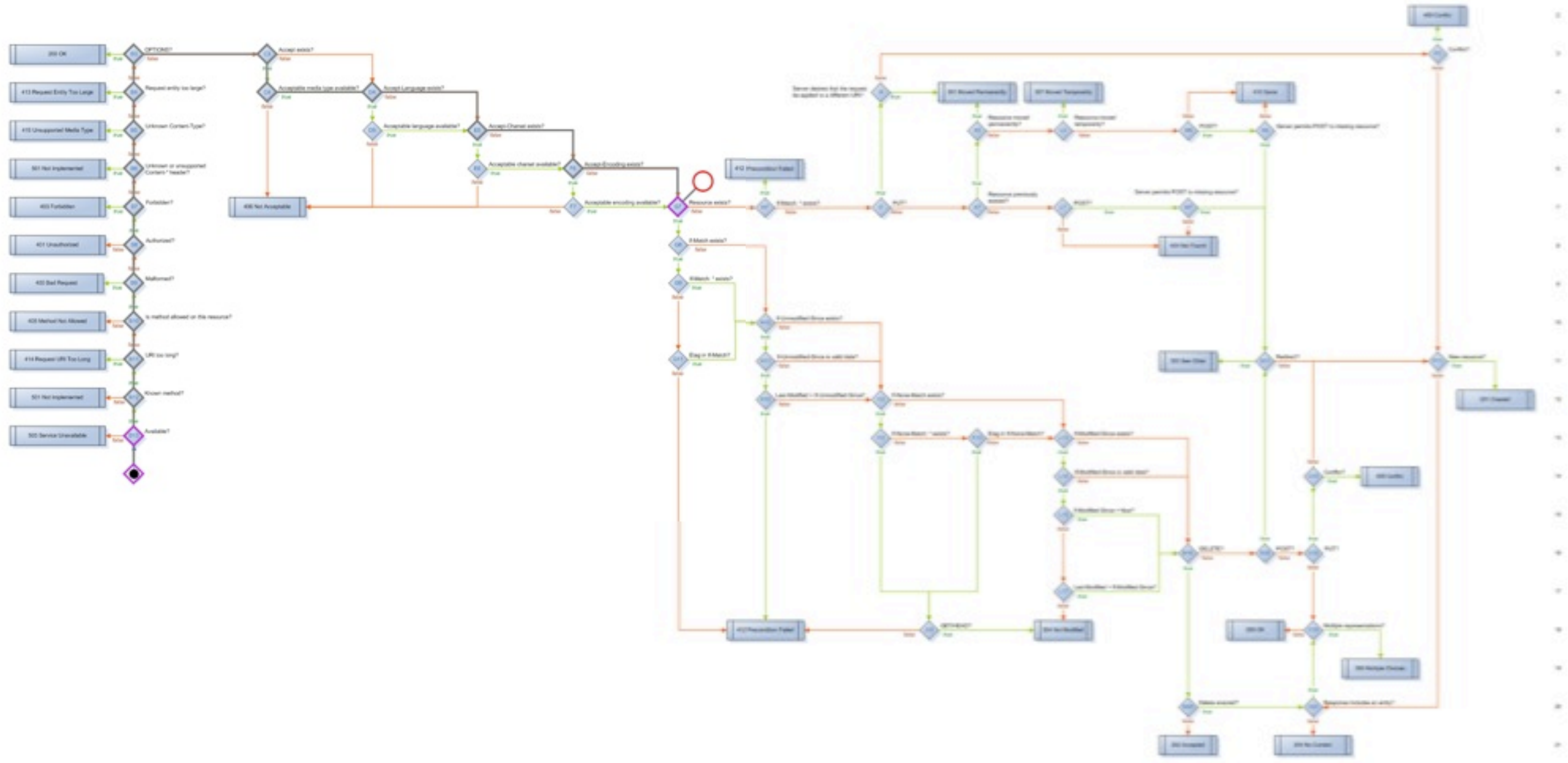
The Webmachine Visual Debugger


false
M16
DELETE?
false
true
true
N16
POST?
false
true
O16
PUT?
false
L17
Last-Modified > If-Modified-Since?
true
false
304 Not Modified
200 OK
O18
Multiple representations?
false
true
300 Multiple Choices
true
M20
Delete enacted?
true
false
O20
Response includes an entity?
false
202 Accepted
204 No Content

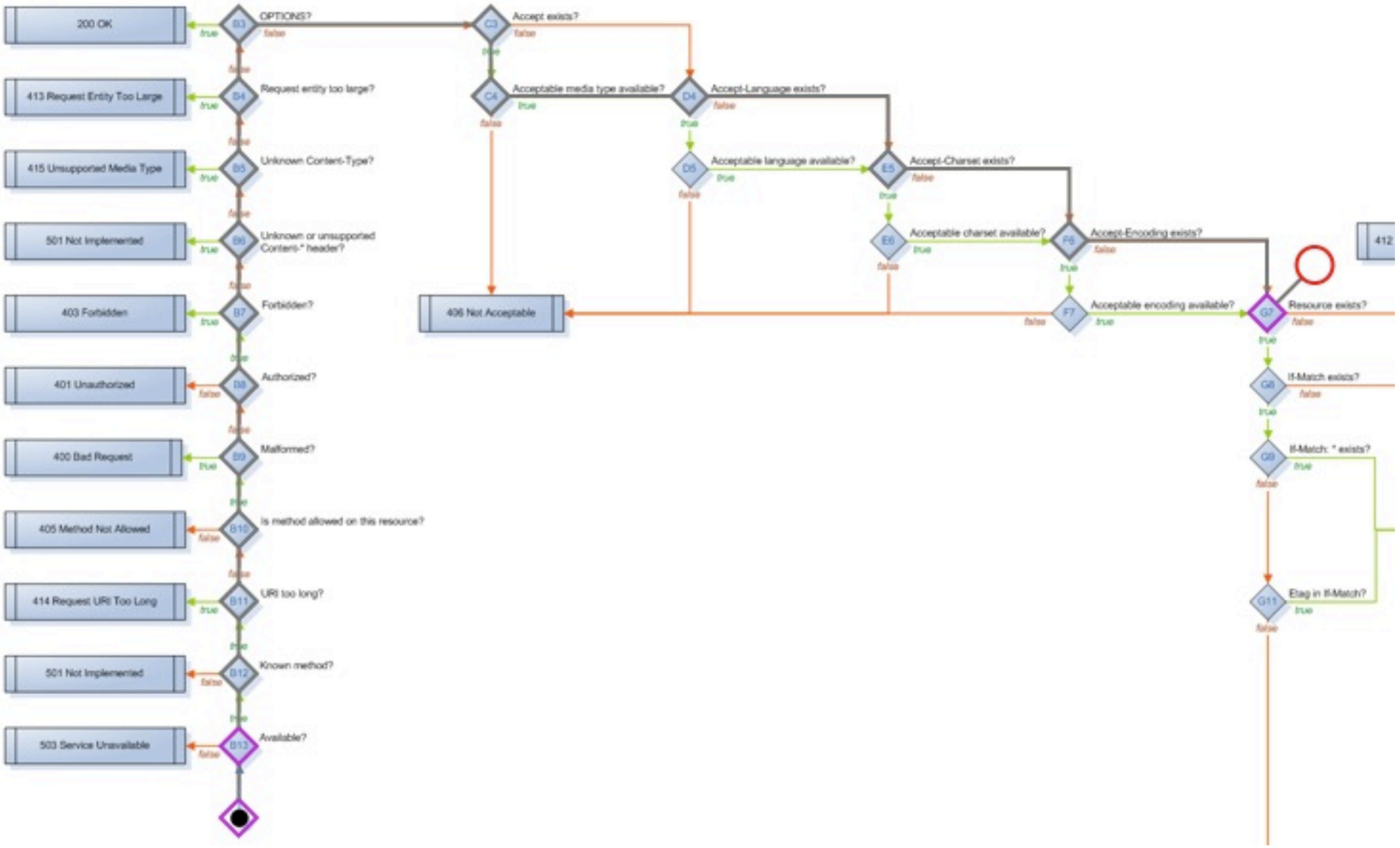
Hooray!



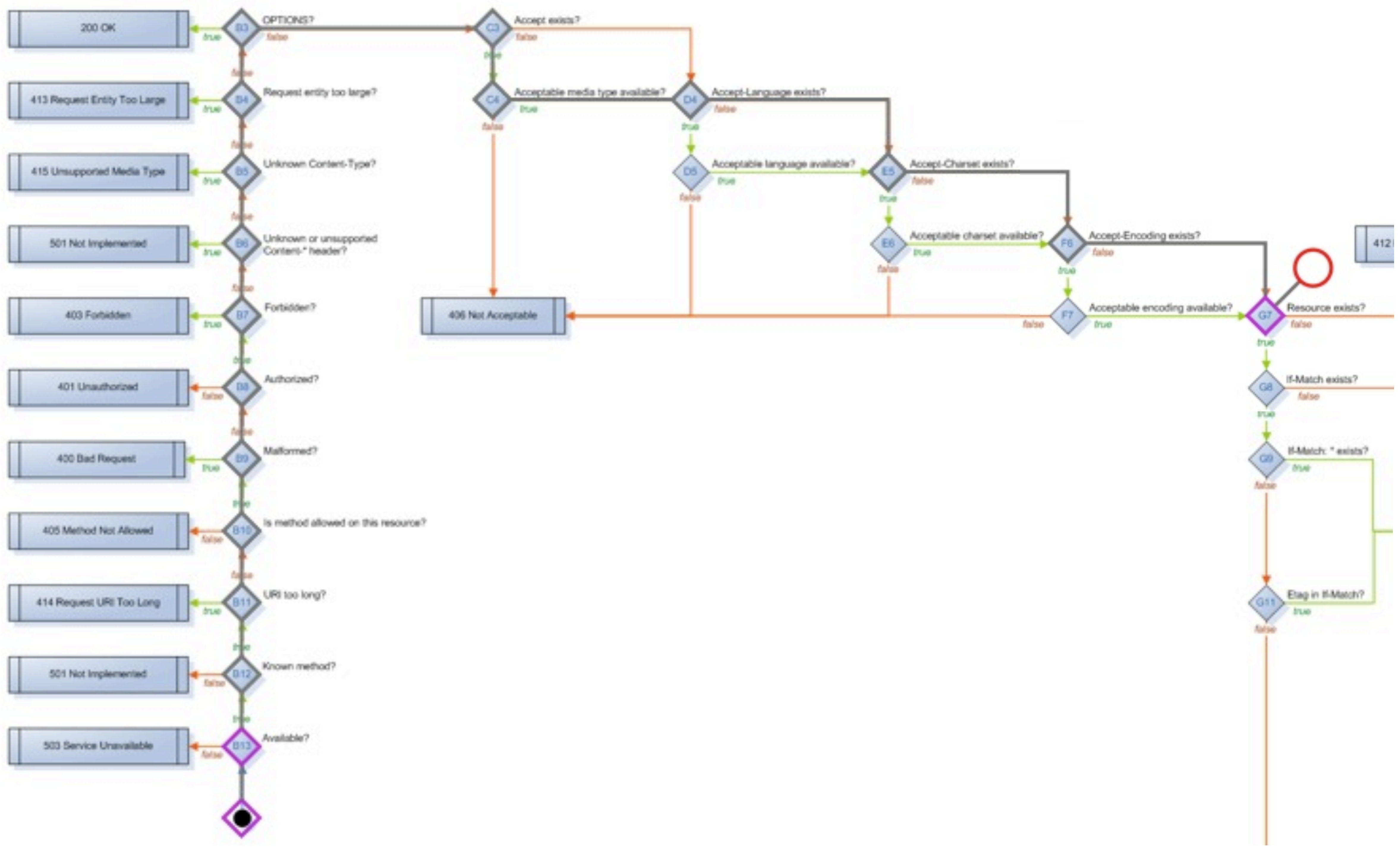
But sometimes things don’t go as well.



But sometimes things don’t go as well.

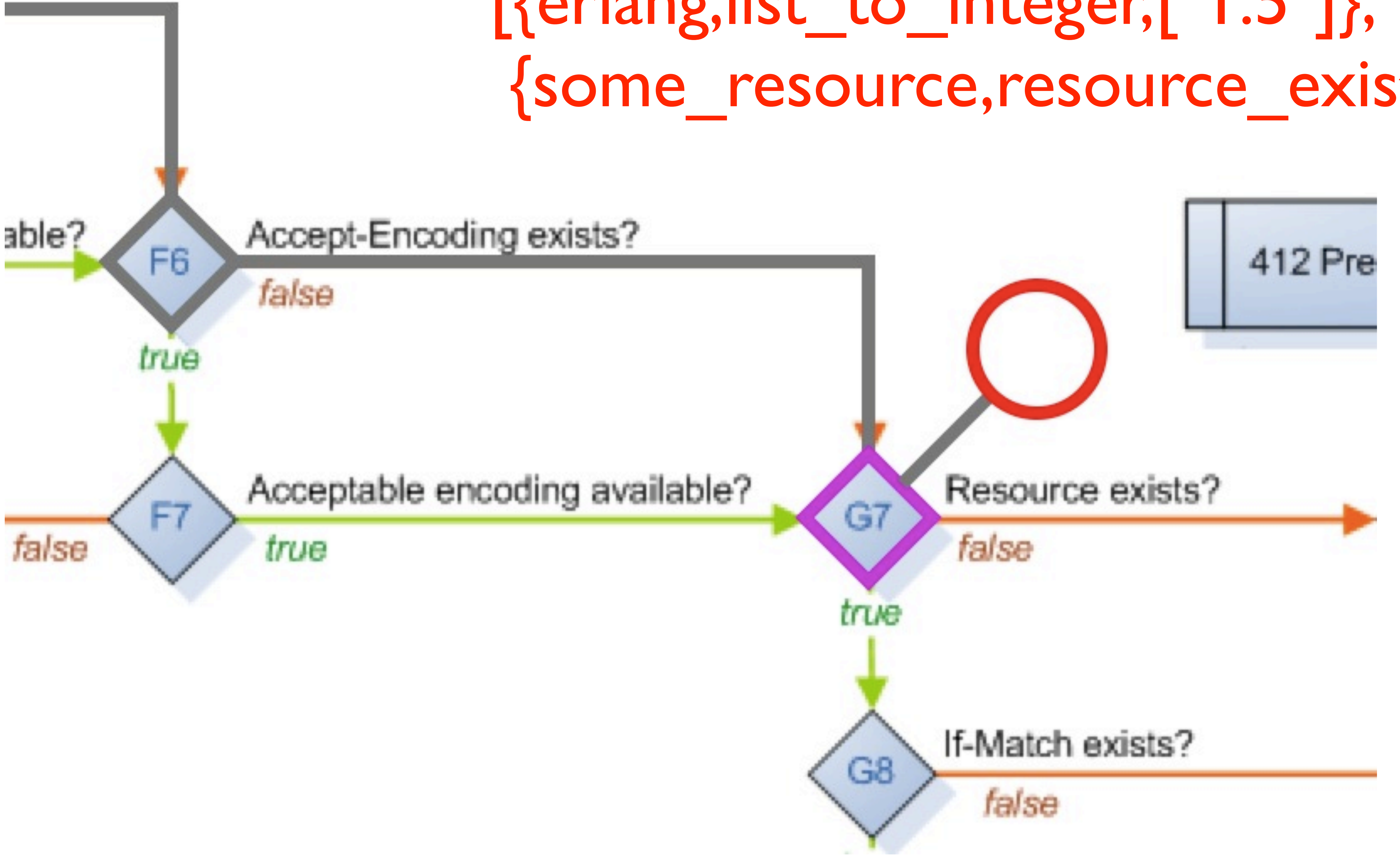


It’s nice to know where your errors are.



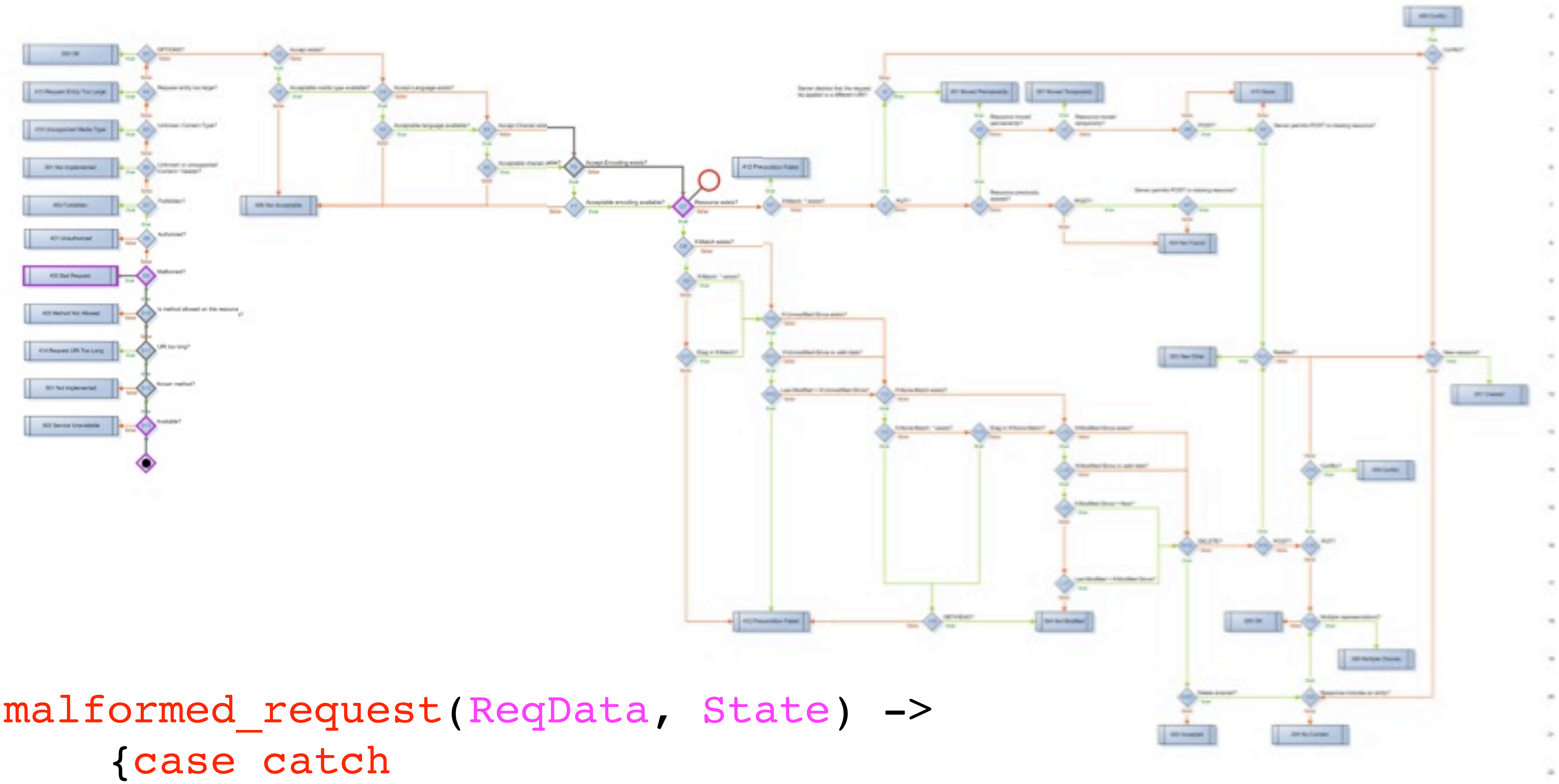
It’s nice to know where your errors are.

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



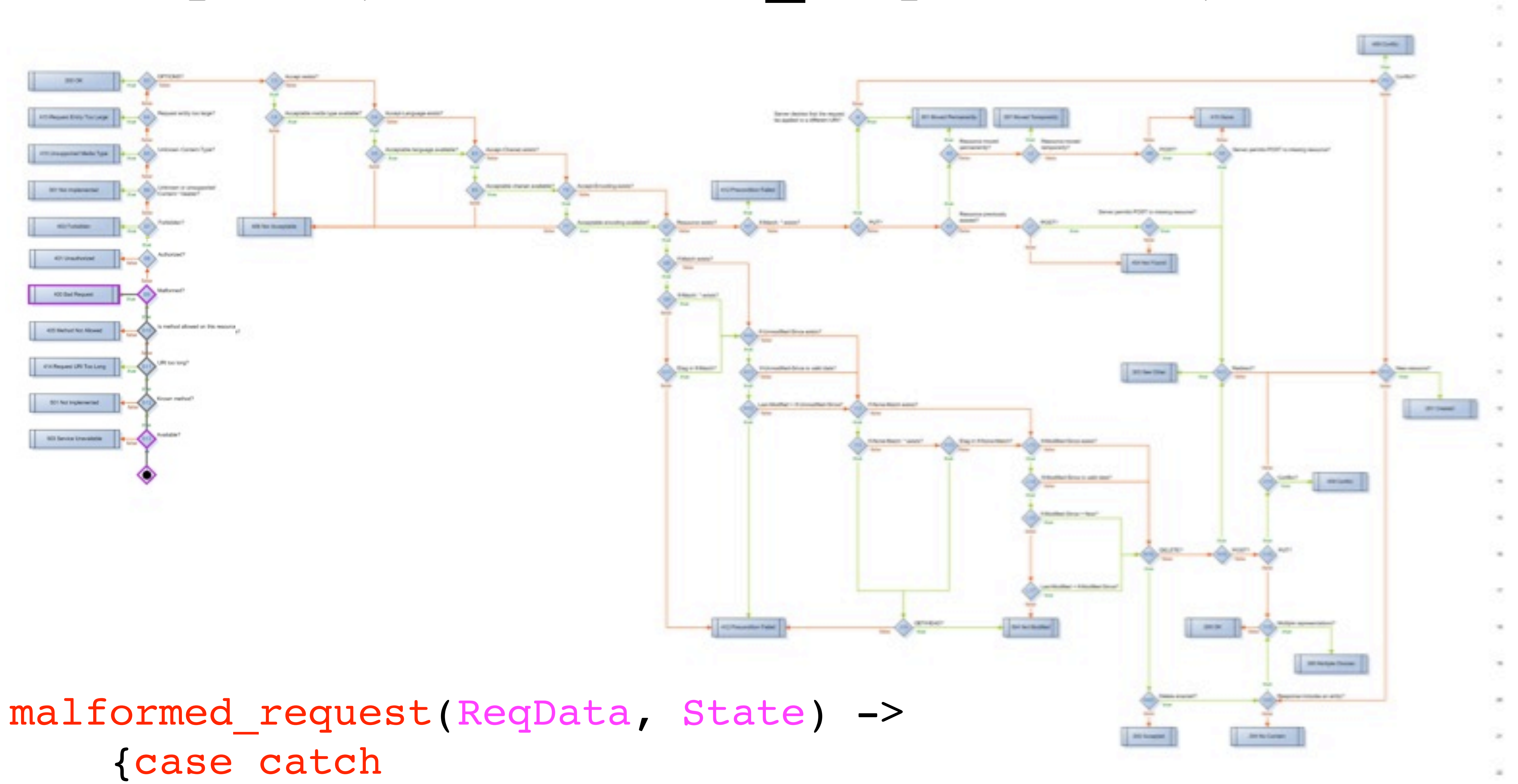
```
wrq:path(RD) -> "/d/test?q=1.5"
```

```
-export([malformed_request/2]).
```

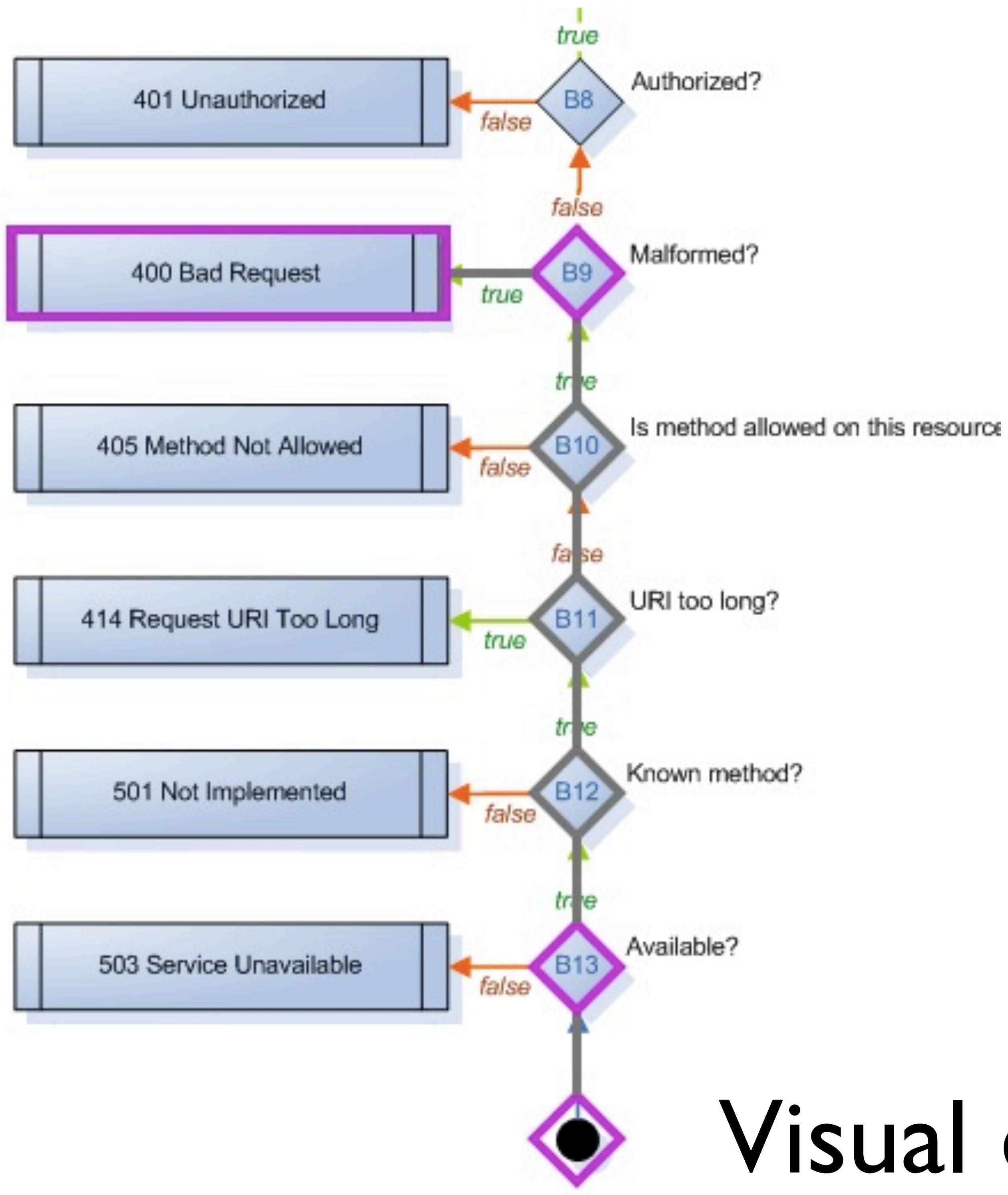


```
malformed_request(ReqData, State) ->
    {case catch
       list_to_integer(wrq:get_qs_value("q","0",ReqData)) of
         {'EXIT', _} -> true;
         _ -> false
    end,
    ReqData, State}.
```

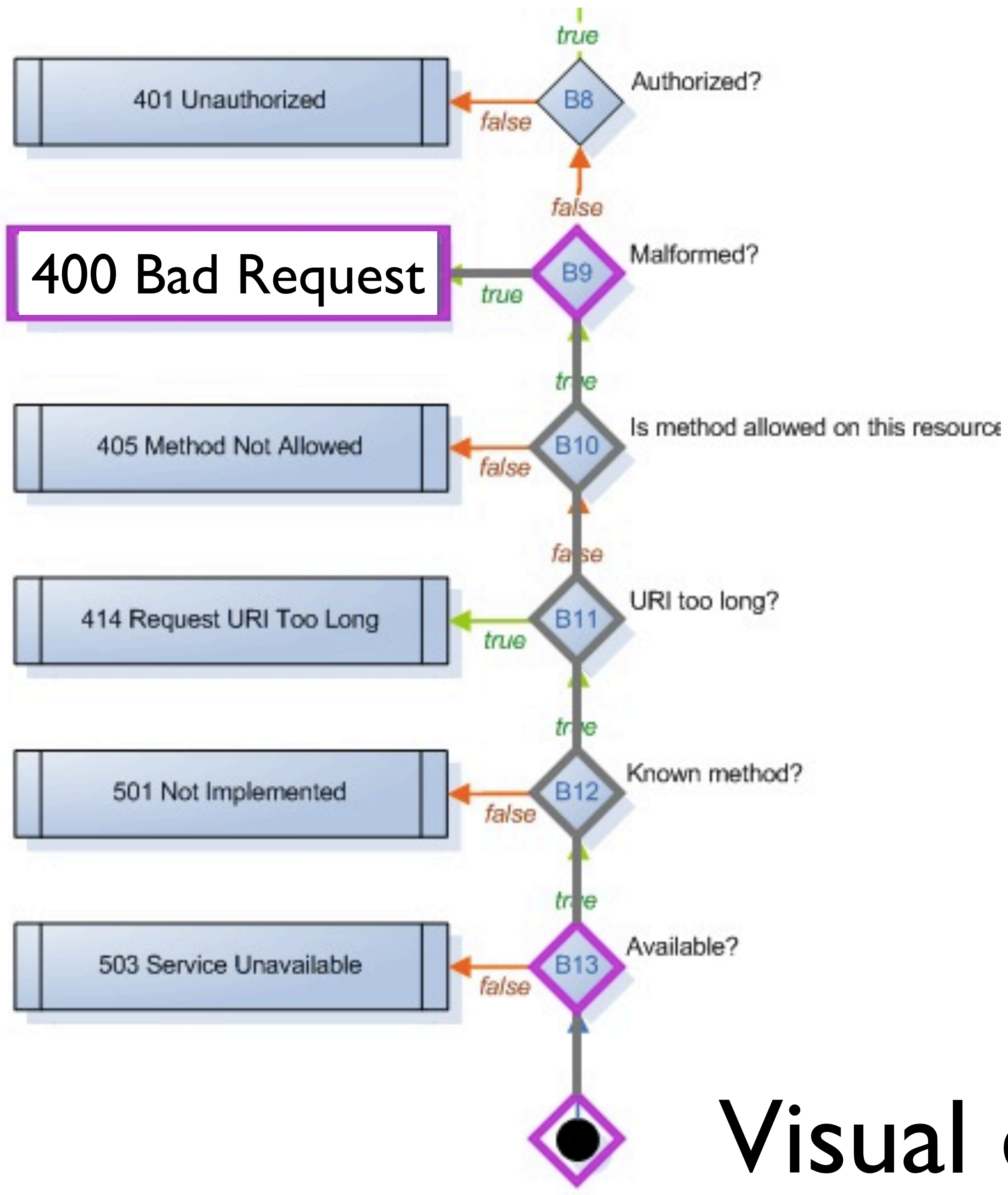
```erlang
-export([malformed_request/2]).
```



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

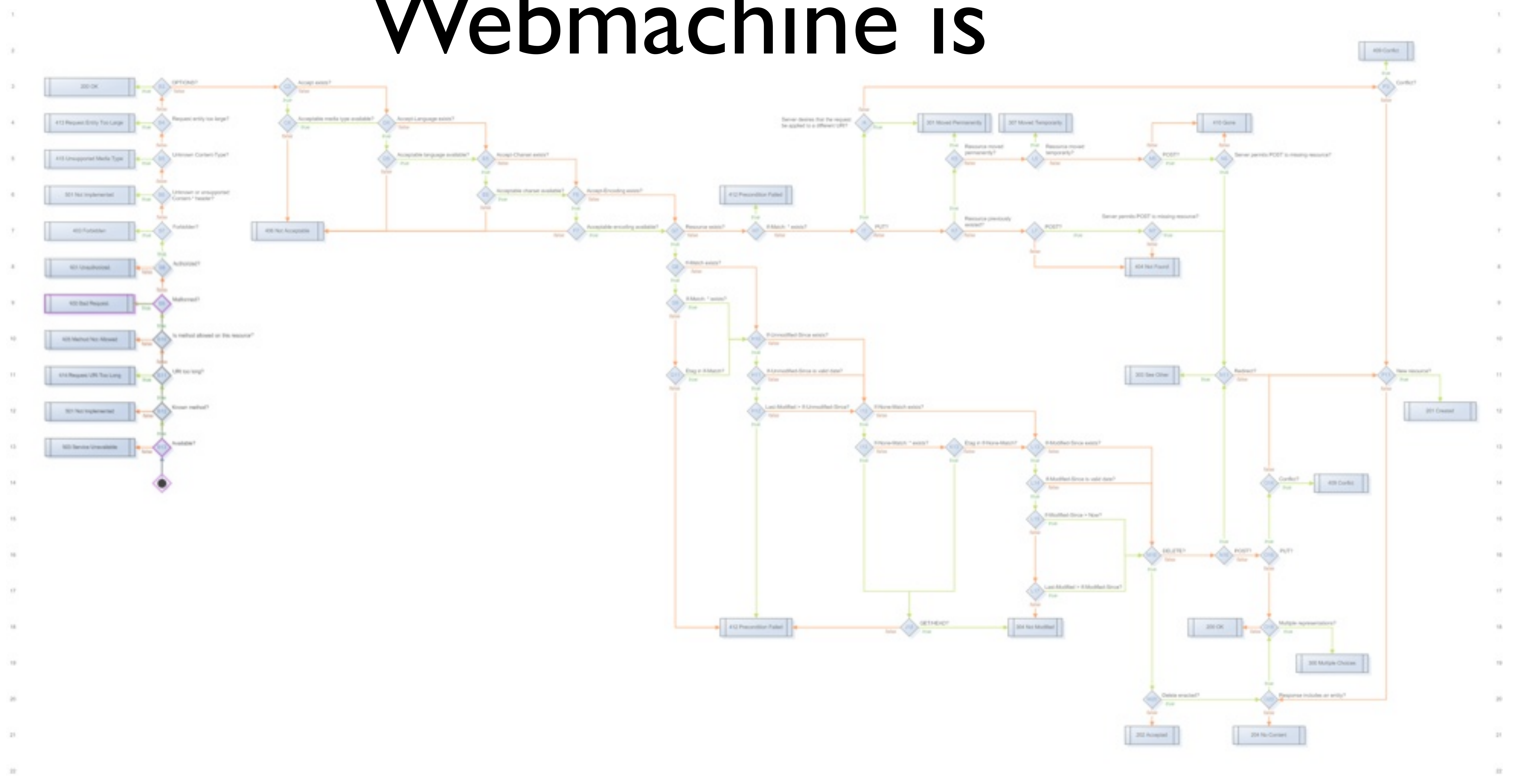


Visual debugging helps you put the fixes in the right place.



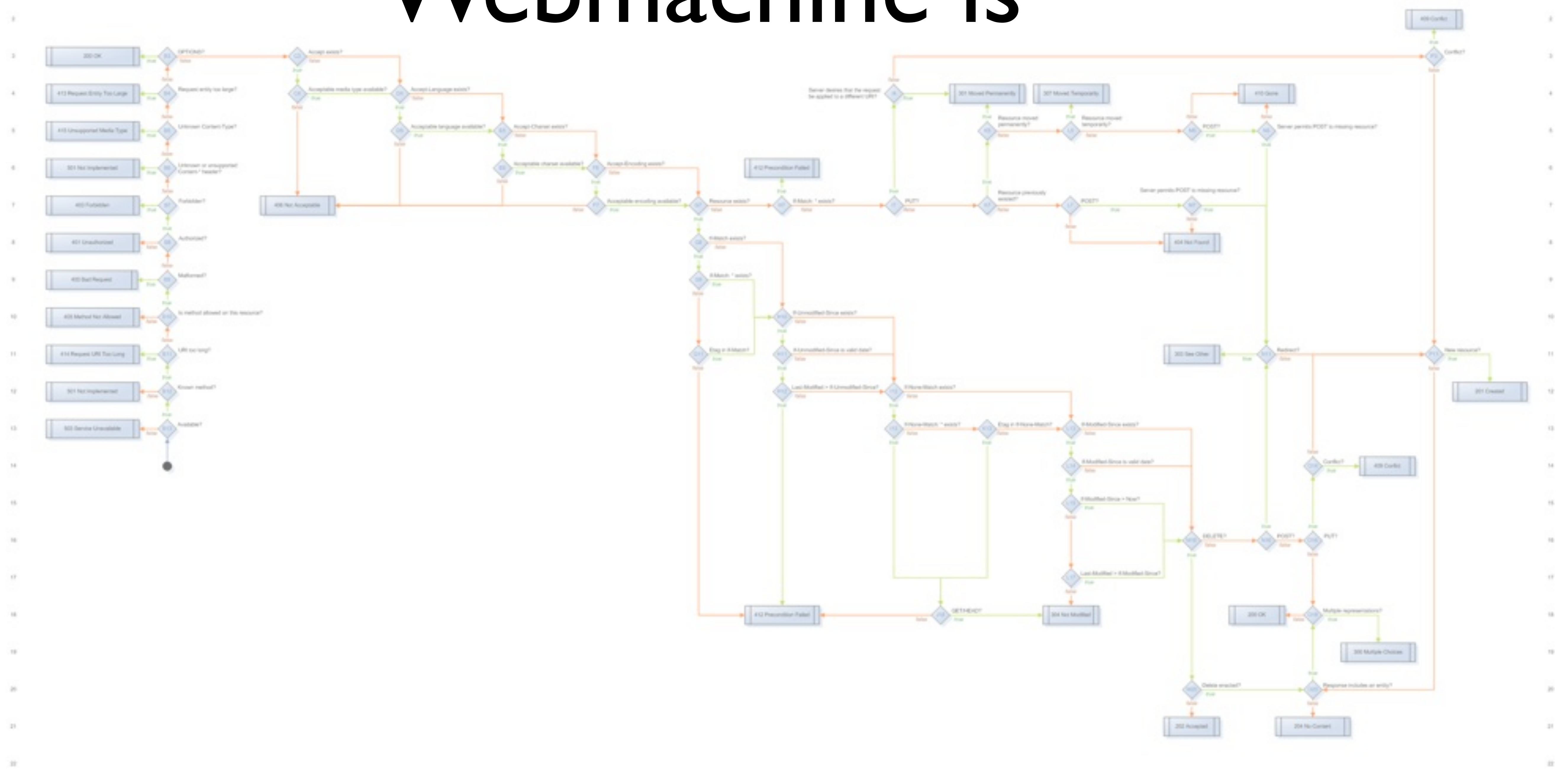
Visual debugging helps you put the fixes in the right place.

Webmachine is



a higher-level abstraction for HTTP.

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 arbitrary sockets, etc.

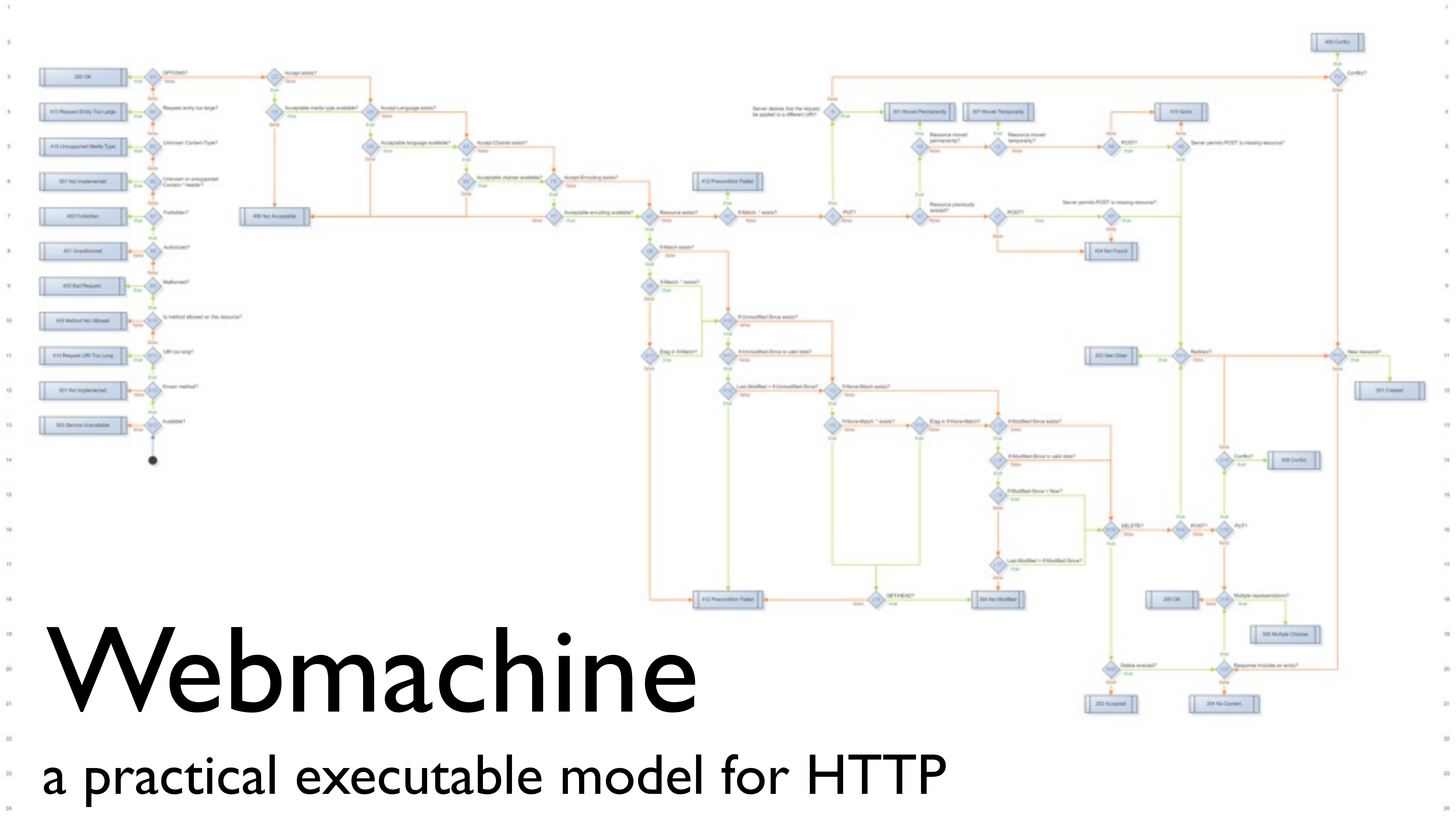
Webmachine is shaped like HTTP.

Webmachine is
a resource server for the Web.

A toolkit for easily creating
well-behaved HTTP systems.

Webmachine is sincerely flattered.

dj-webmachine:	Django/Python
clothesline:	Clojure
lemmachine:	Agda
nodemachine:	JavaScript
webmachine-ruby:	Ruby



Webmachine

a practical executable model for HTTP

http://webmachine.basho.com/

Steve Vinoski
vinoski@ieee.org