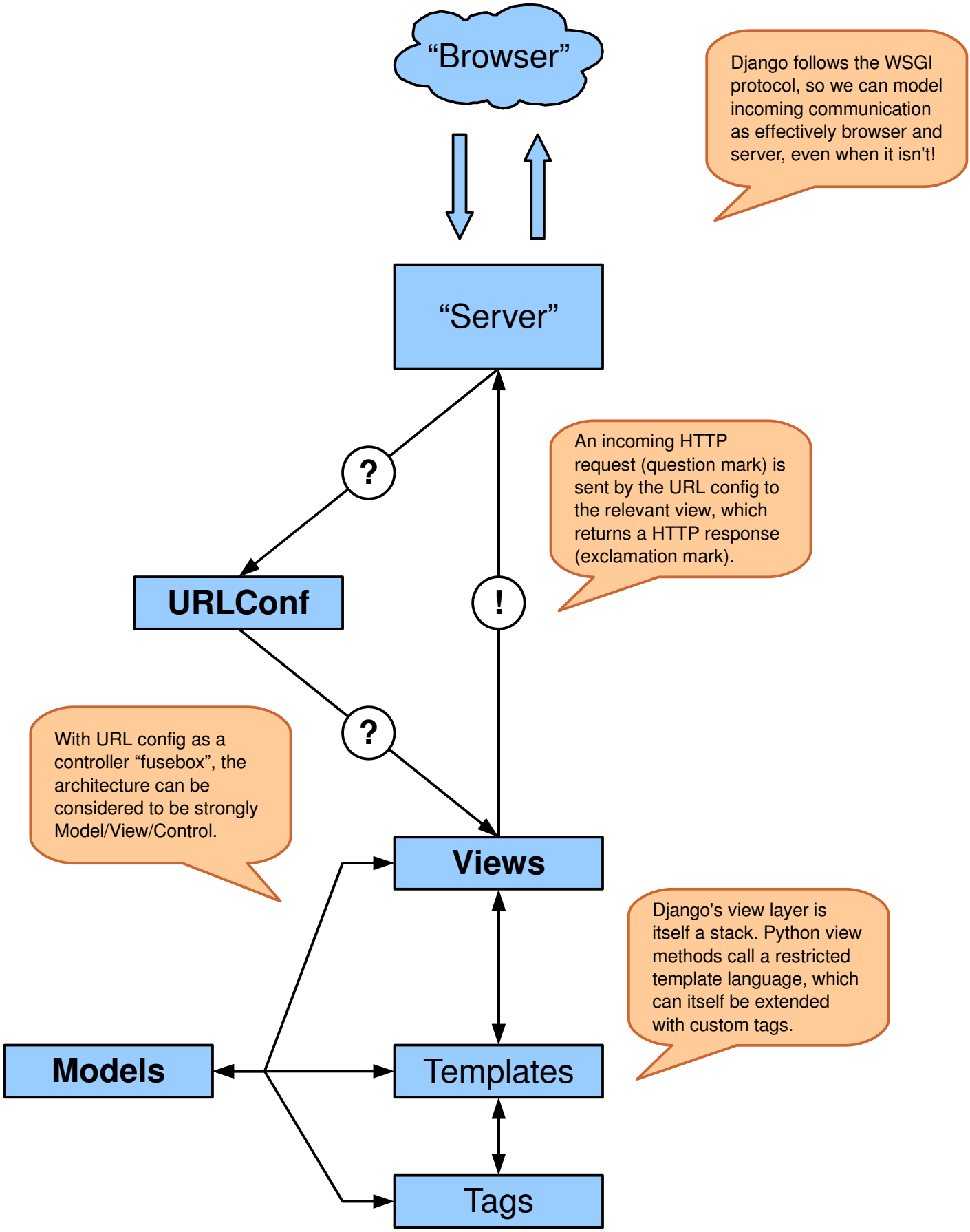
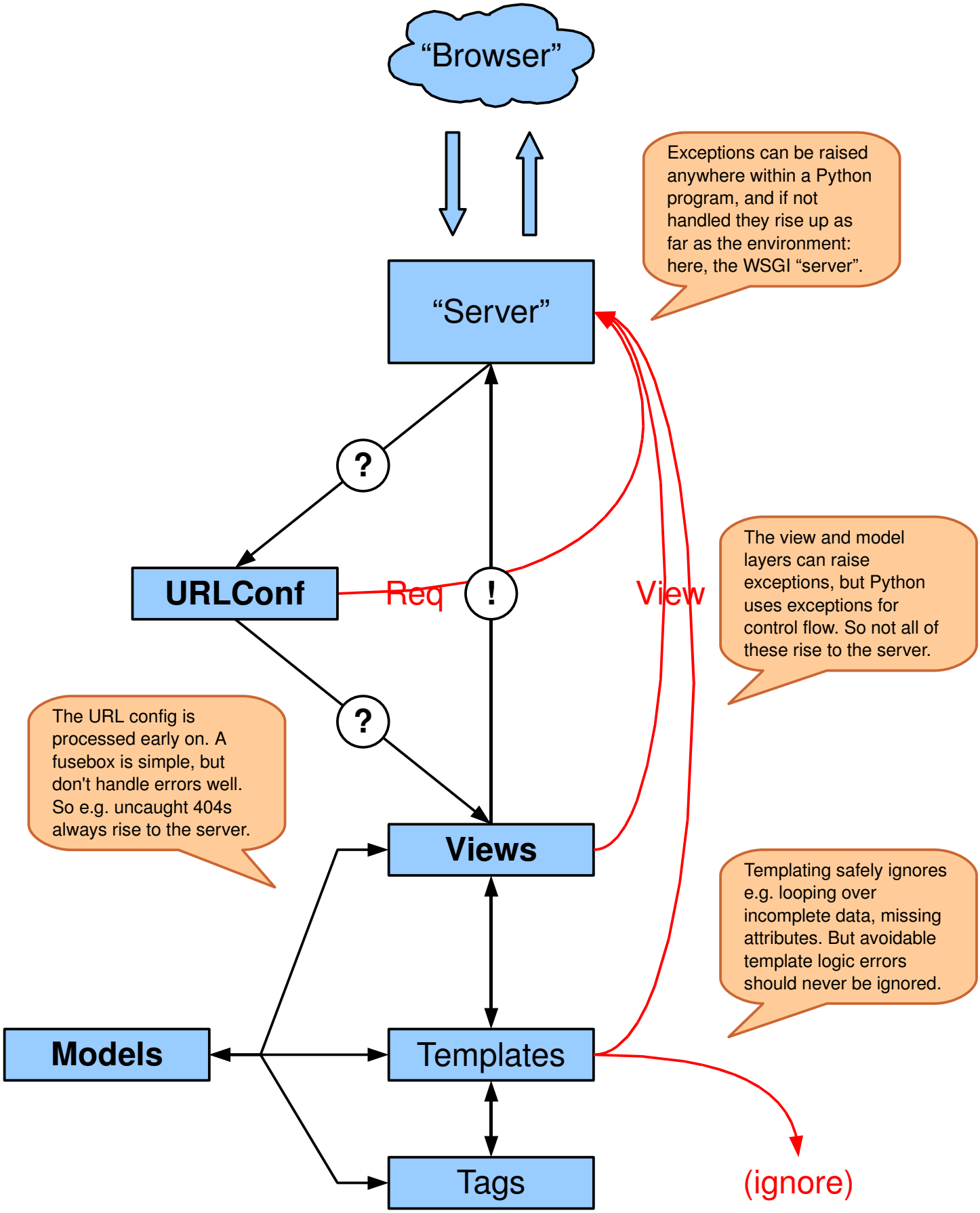
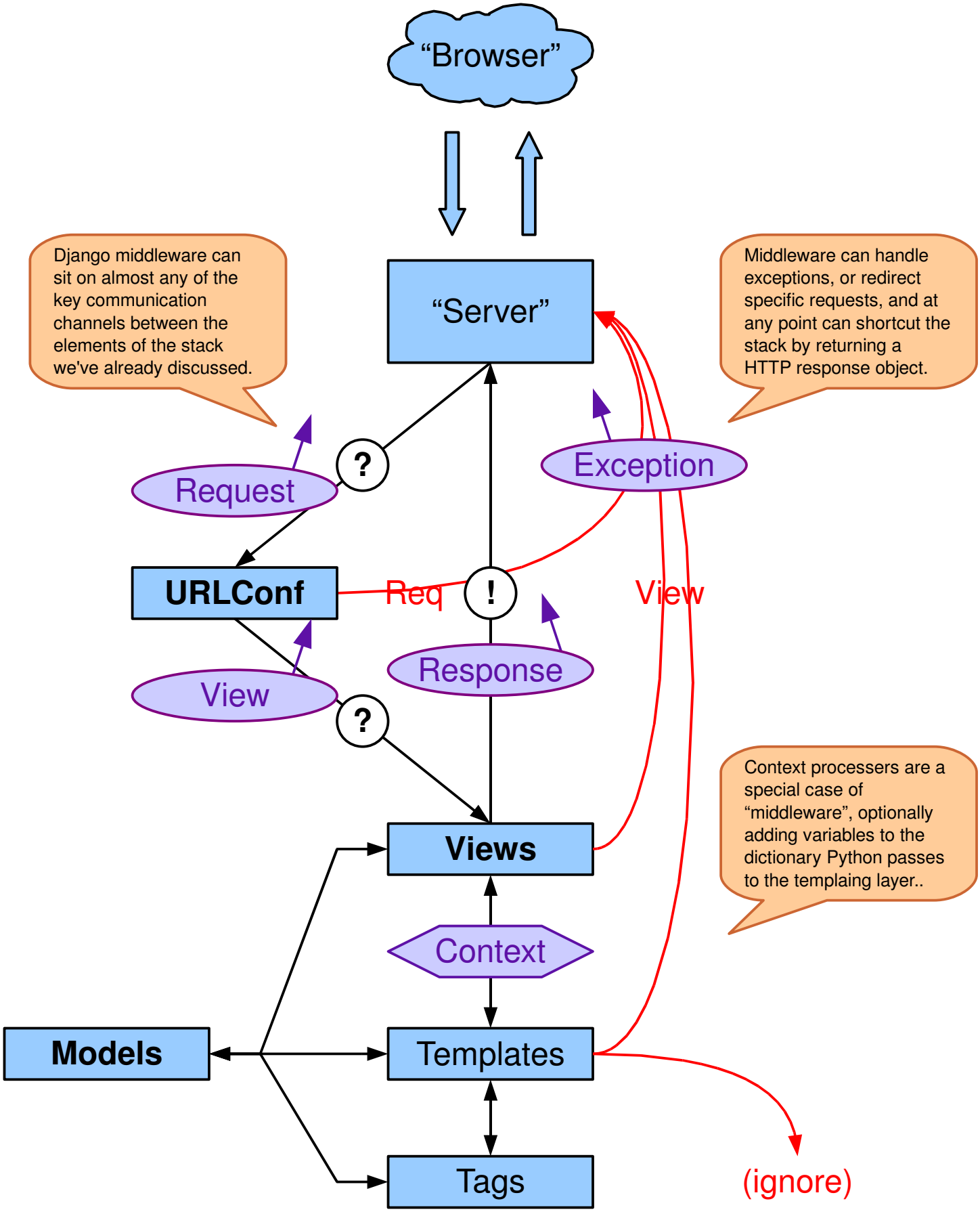

"Browser"
Django follows the WSGI protocol, so we can model incoming communication as effectively browser and server, even when it isn't!
"Server"
?
An incoming HTTP request (question mark) is sent by the URL config to the relevant view, which returns a HTTP response (exclamation mark).
URLConf
!
?
With URL config as a controller "fusebox", the architecture can be considered to be strongly Model/View/Control.
Views
Django's view layer is itself a stack. Python view methods call a restricted template language, which can itself be extended with custom tags.
Models
Templates
Tags


“Browser”
“Server”
Exceptions can be raised anywhere within a Python program, and if not handled they rise up as far as the environment: here, the WSGI “server”.
?
URLConf
Req
!
View
The view and model layers can raise exceptions, but Python uses exceptions for control flow. So not all of these rise to the server.
?
The URL config is processed early on. A fusebox is simple, but don't handle errors well. So e.g. uncaught 404s always rise to the server.
Views
Templating safely ignores e.g. looping over incomplete data, missing attributes. But avoidable template logic errors should never be ignored.
Models
Templates
Tags
(ignore)


"Browser"
"Server"
Django middleware can sit on almost any of the key communication channels between the elements of the stack we've already discussed.
Middleware can handle exceptions, or redirect specific requests, and at any point can shortcut the stack by returning a HTTP response object.
Request
?
Exception
URLConf
Req
!
View
Response
View
?
Context processers are a special case of "middleware", optionally adding variables to the dictionary Python passes to the templaing layer..
Views
Context
Models
Templates
Tags
(ignore)