ECS 162
WEB PROGRAMMING

RESTful Web Services

- Name for the interaction model we have been using
- REST stands for “representational state transfer”
- Resource identification through a URI
  - E.g., https://twitter.com/stk
- Uniform interface via HTTP request-response
  - HTTP verbs: GET, POST, maybe PUT and DELETE

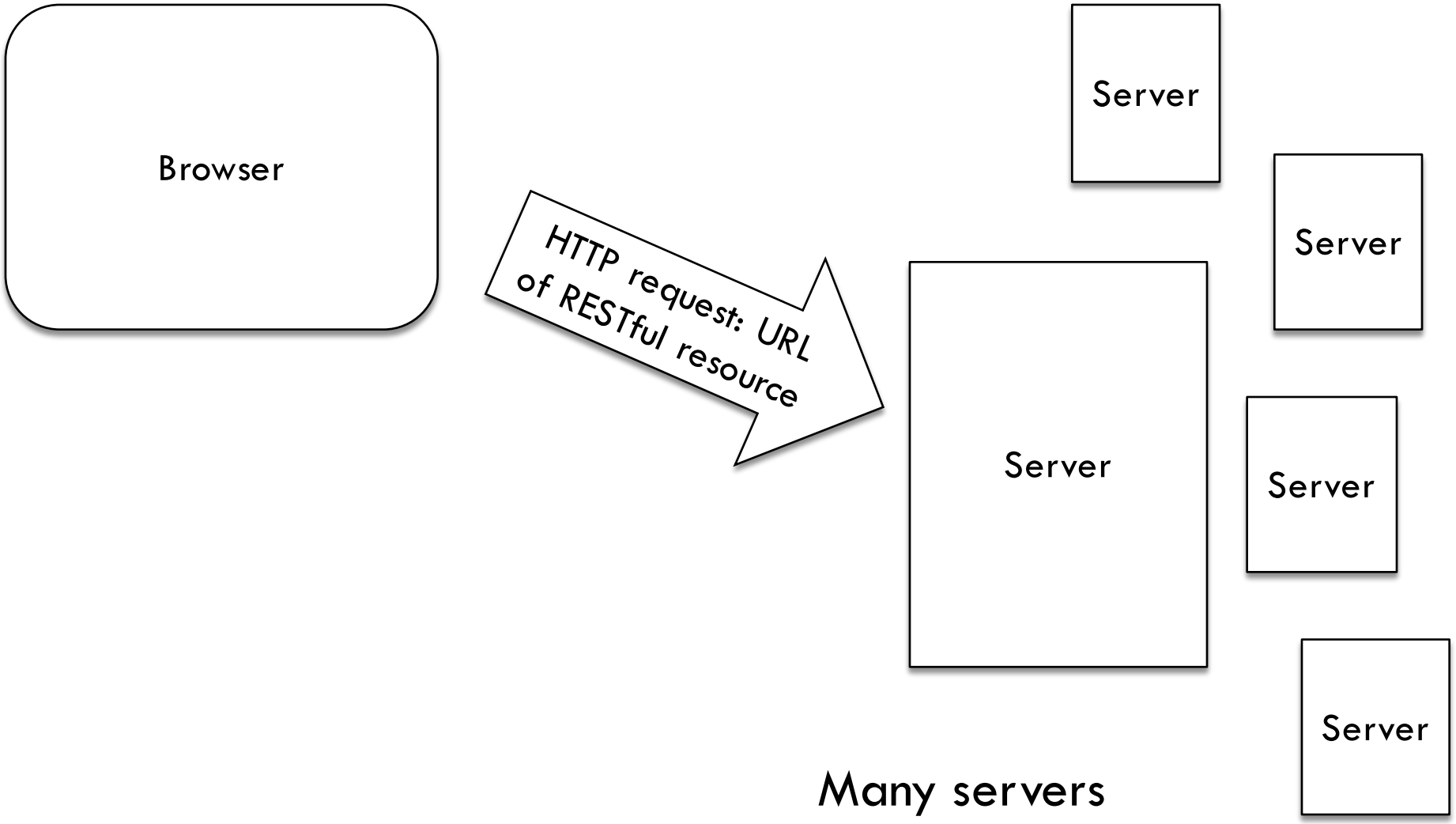
Efficient approach to stateful interactions

- Client (browser) needs to know only a single URL to access the resource; further interactions are learned as it goes along
- Server does not need to know or remember anything about state of client

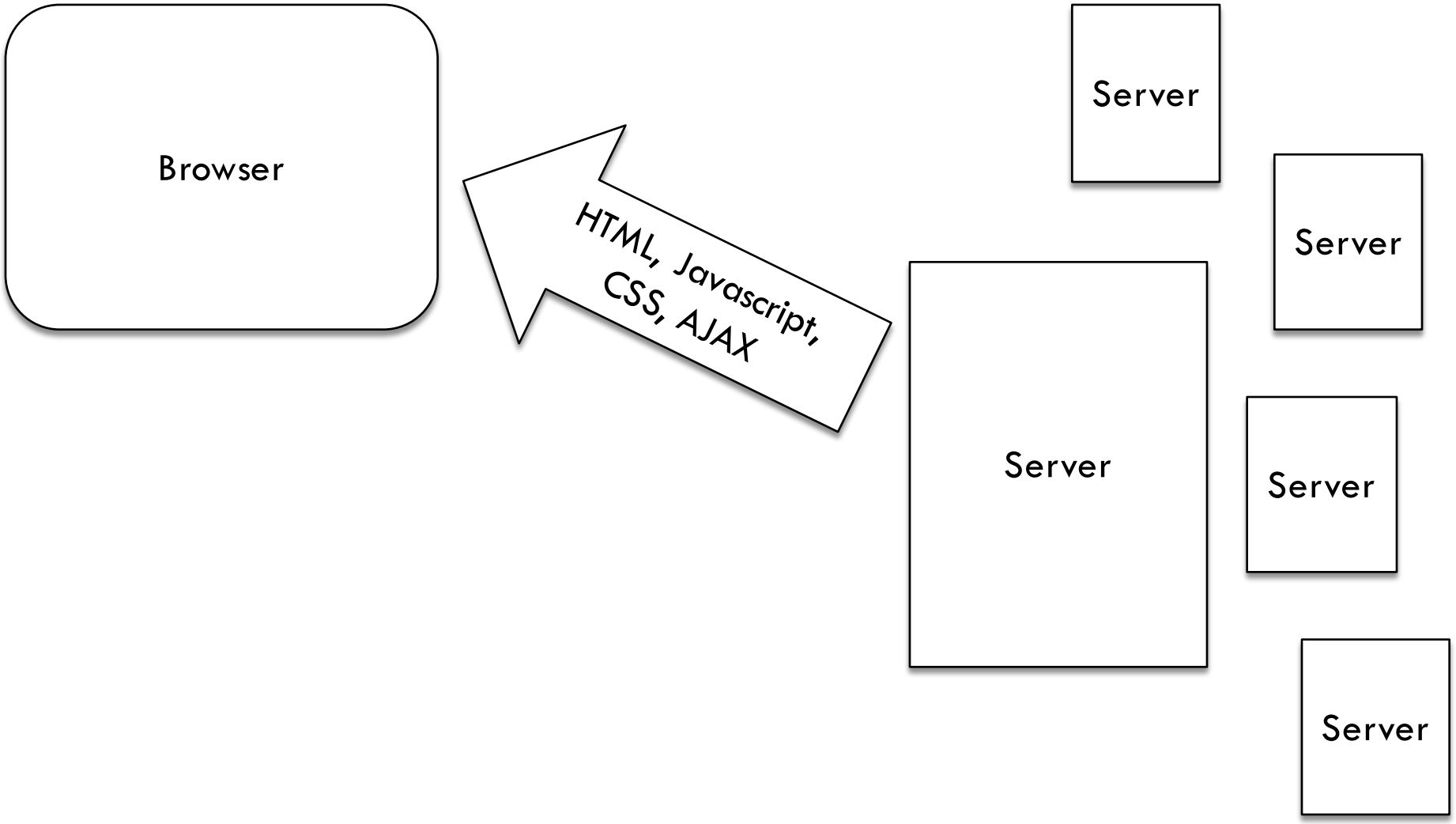
RESTFul applied

- Key part, identify your objects
  - Twitter: users, tweets, DMs
  - Lyft: drivers, passengers, rides, payment methods
  - Google drive: users, folders, and documents
  - HTTP requests to access

Browser's view of the world



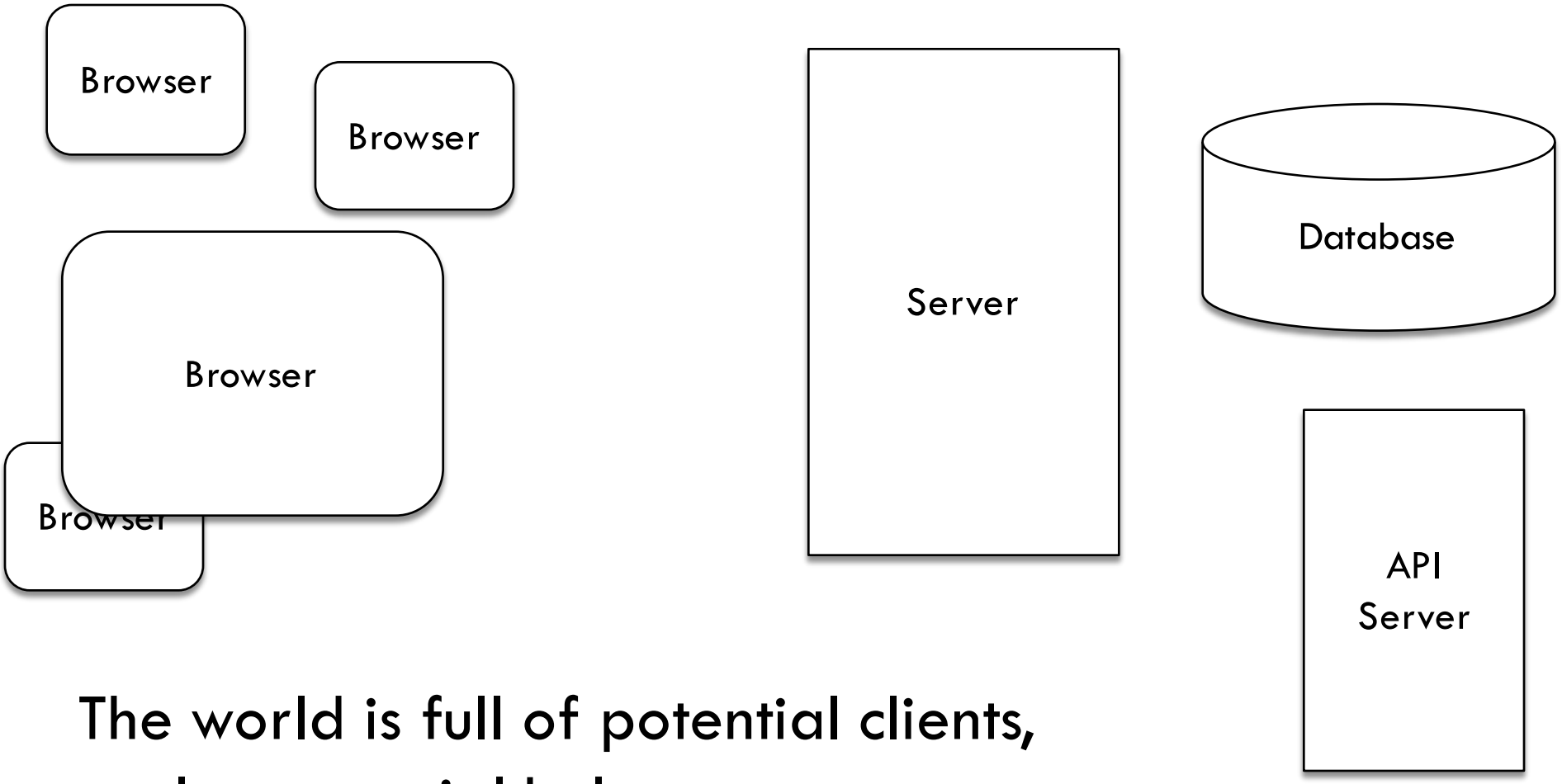
Browser's view



Browser's view

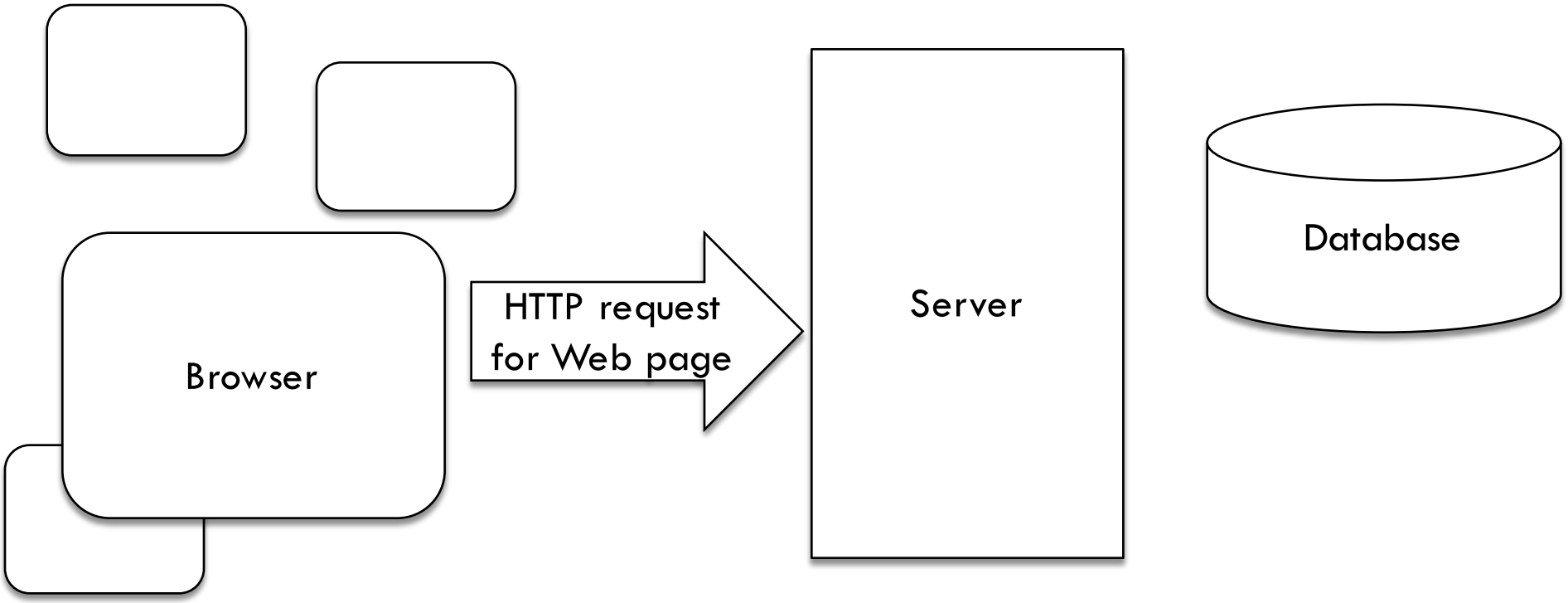
- Browser presents coherent narrative to the user.
- It stops and starts, but there is a logical sequence of operations, one following another.
- All interactions are simple request-response sequences
- Server's view is very different.
- It processes HTTP request-response pairs, without trying to connect them together in a coherent narrative.
- But interactions might be more complex

Server's view of the world

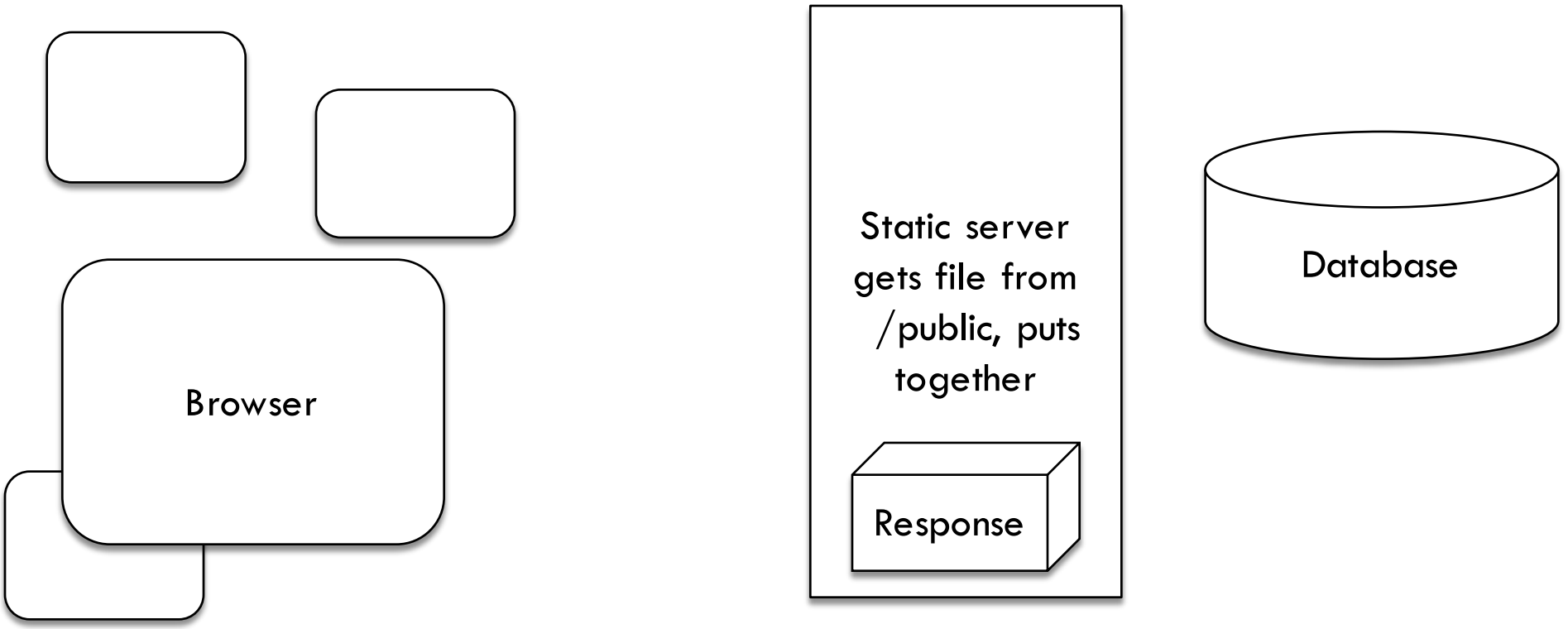


The world is full of potential clients, also potential helpers.

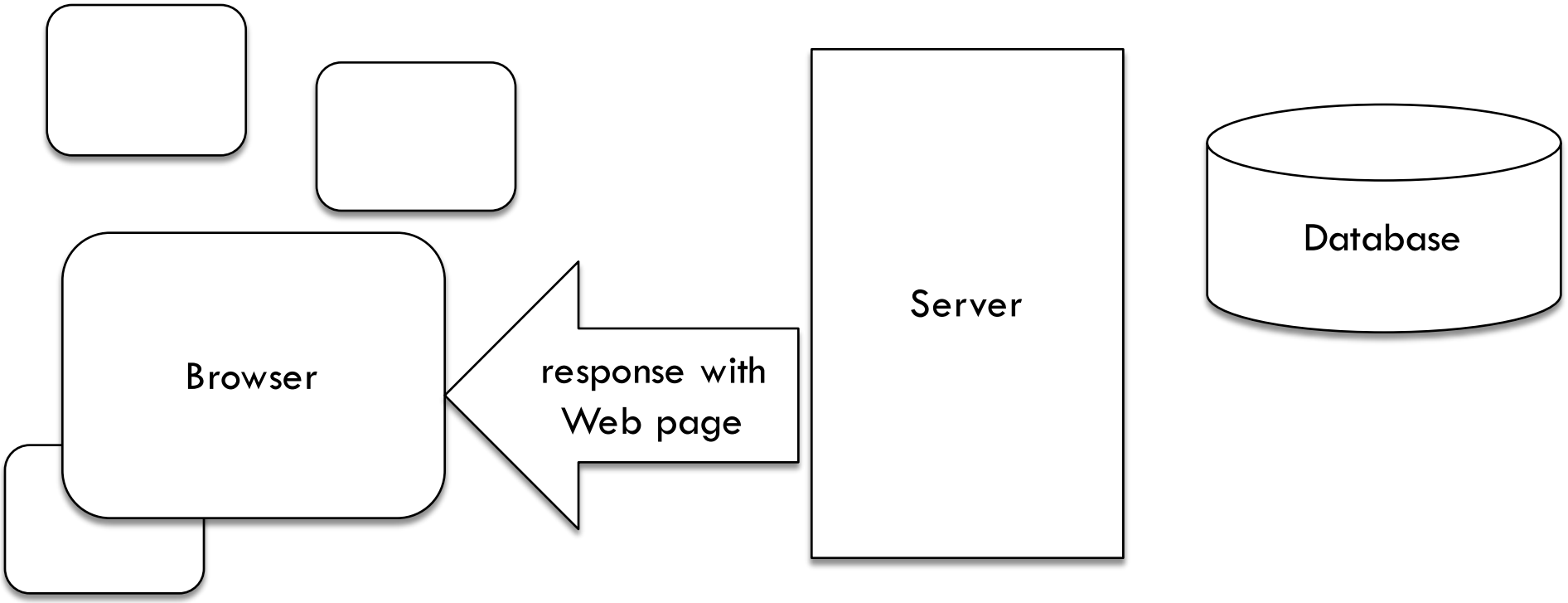
Server's view



Server's view

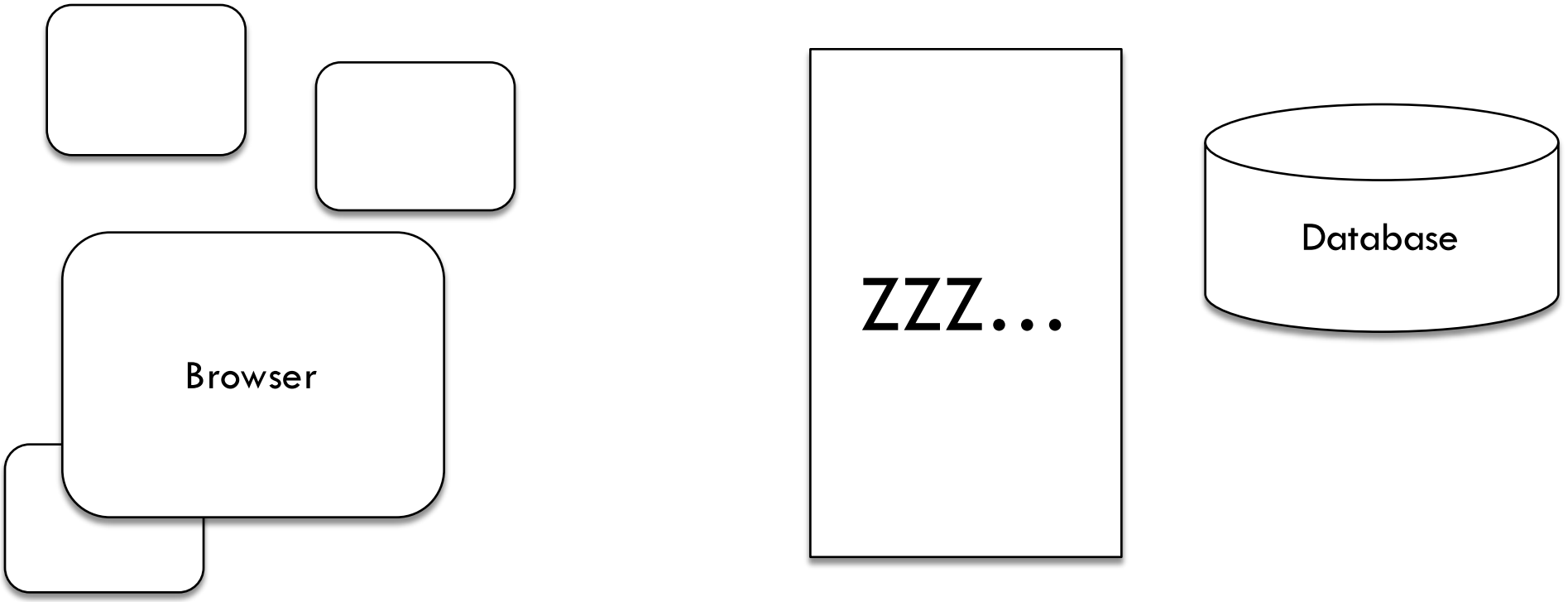


Server's view



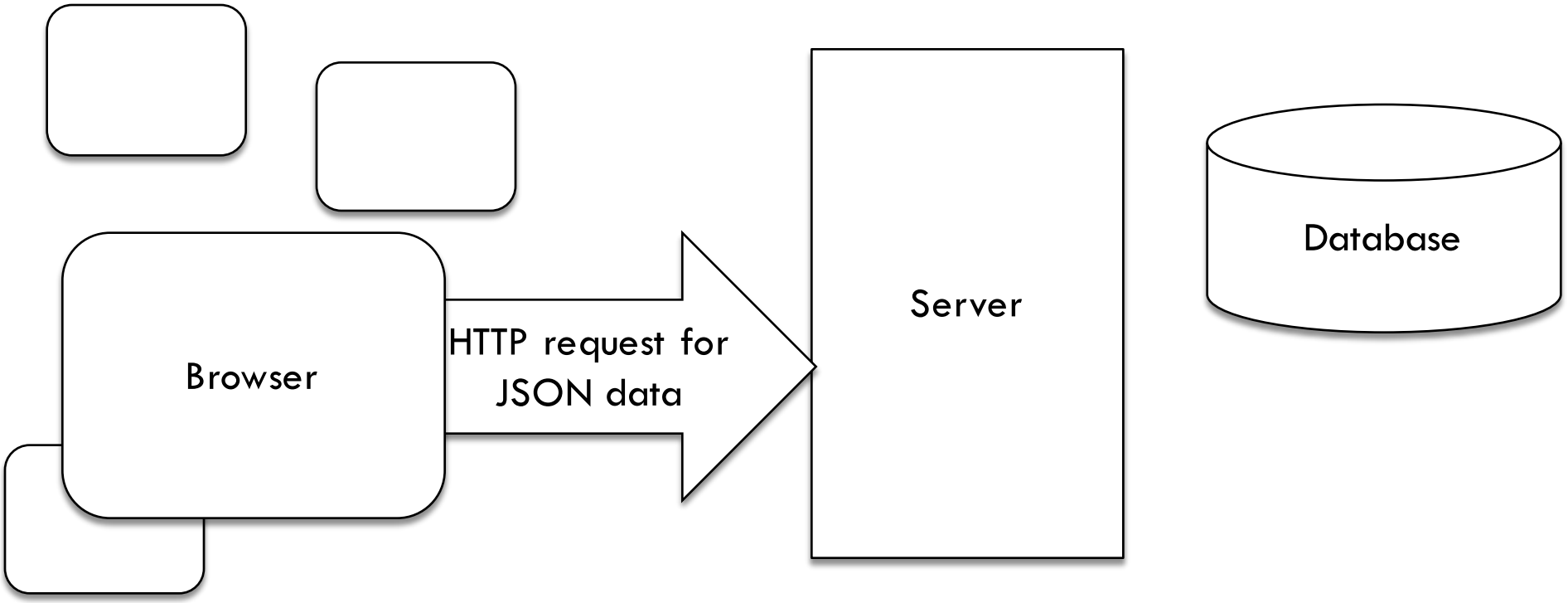
Server sends back response object that is self-addressed to browser.

Server's view

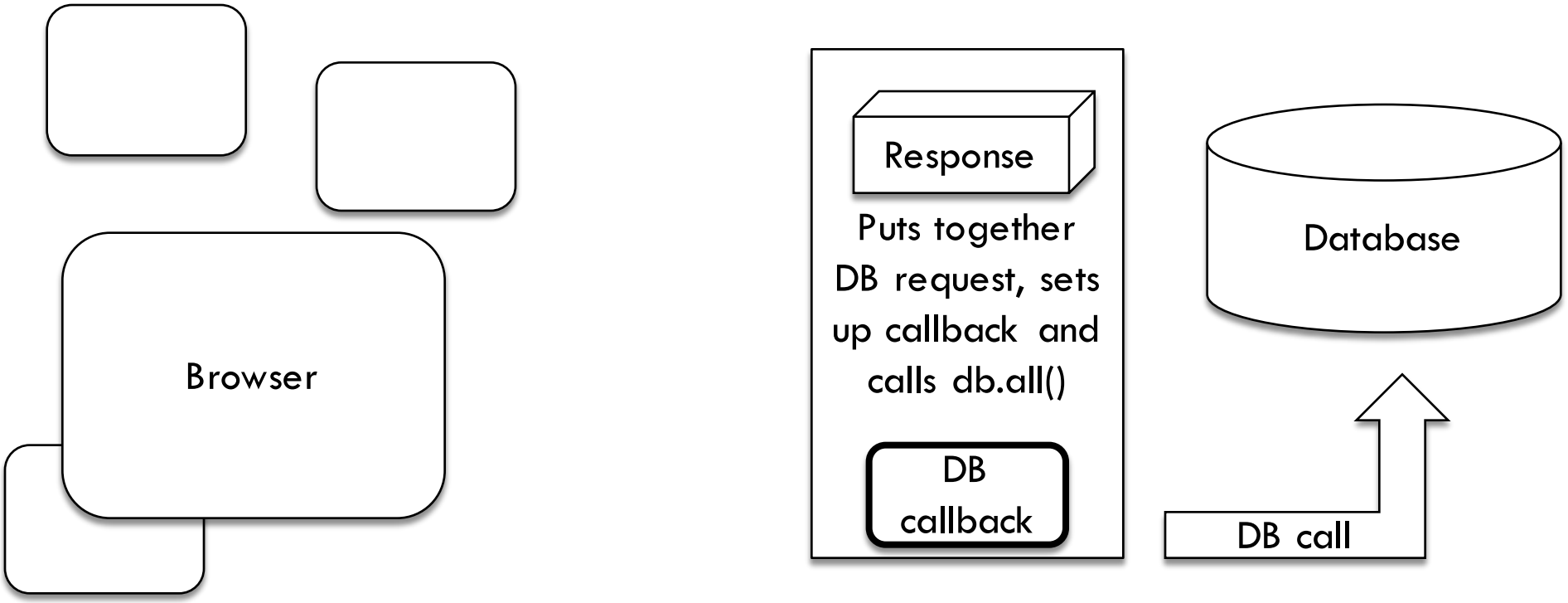


The server remembers nothing about the transaction.

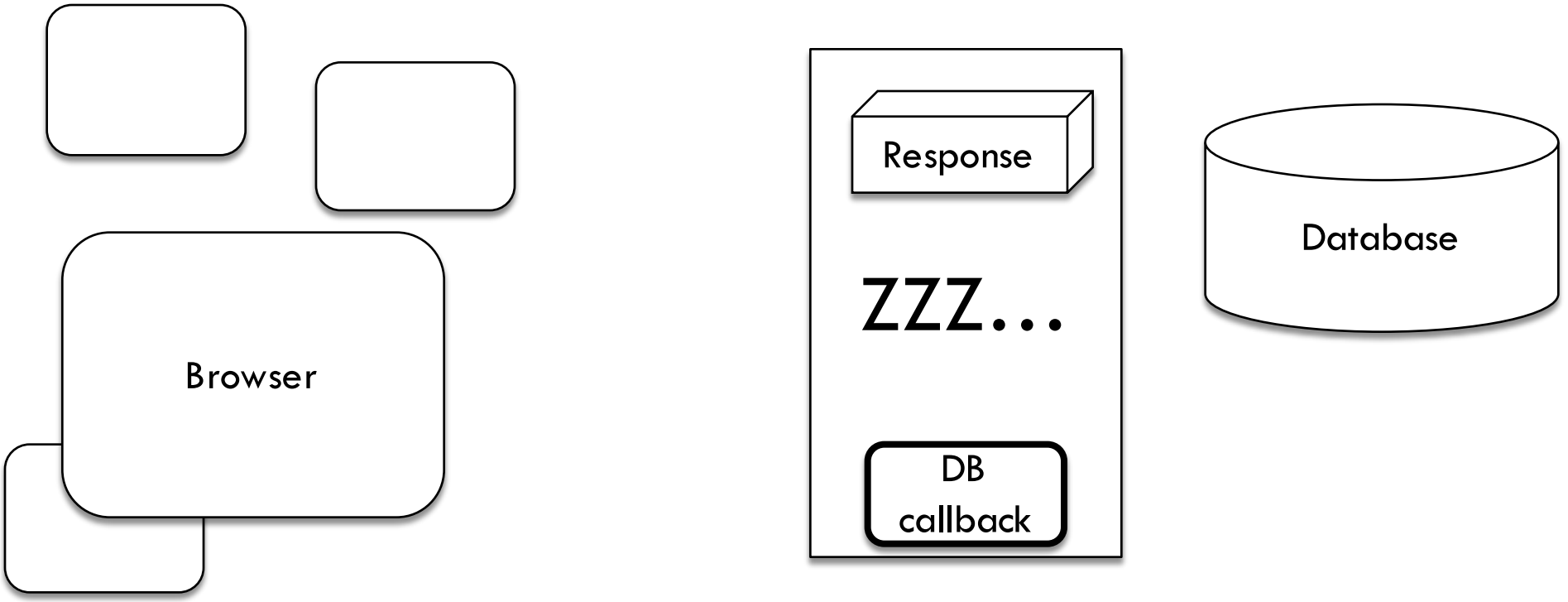
Server's view



Server's view

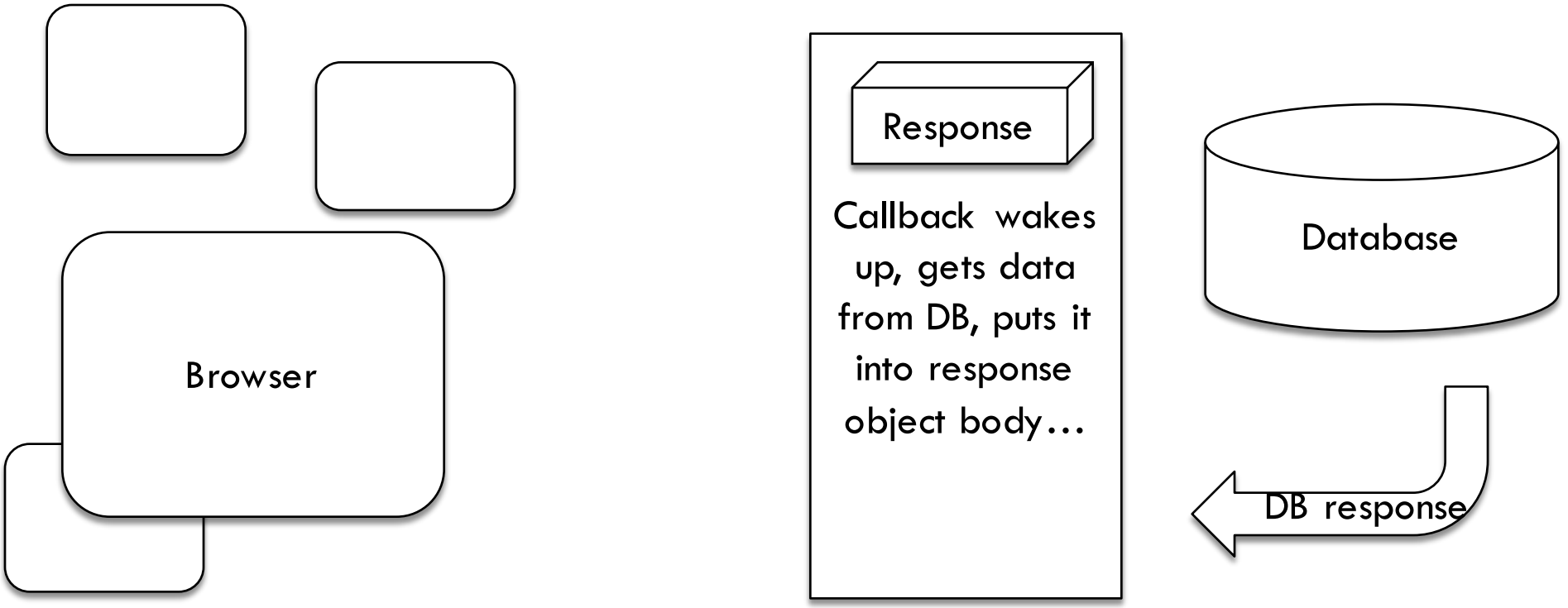


Server's view

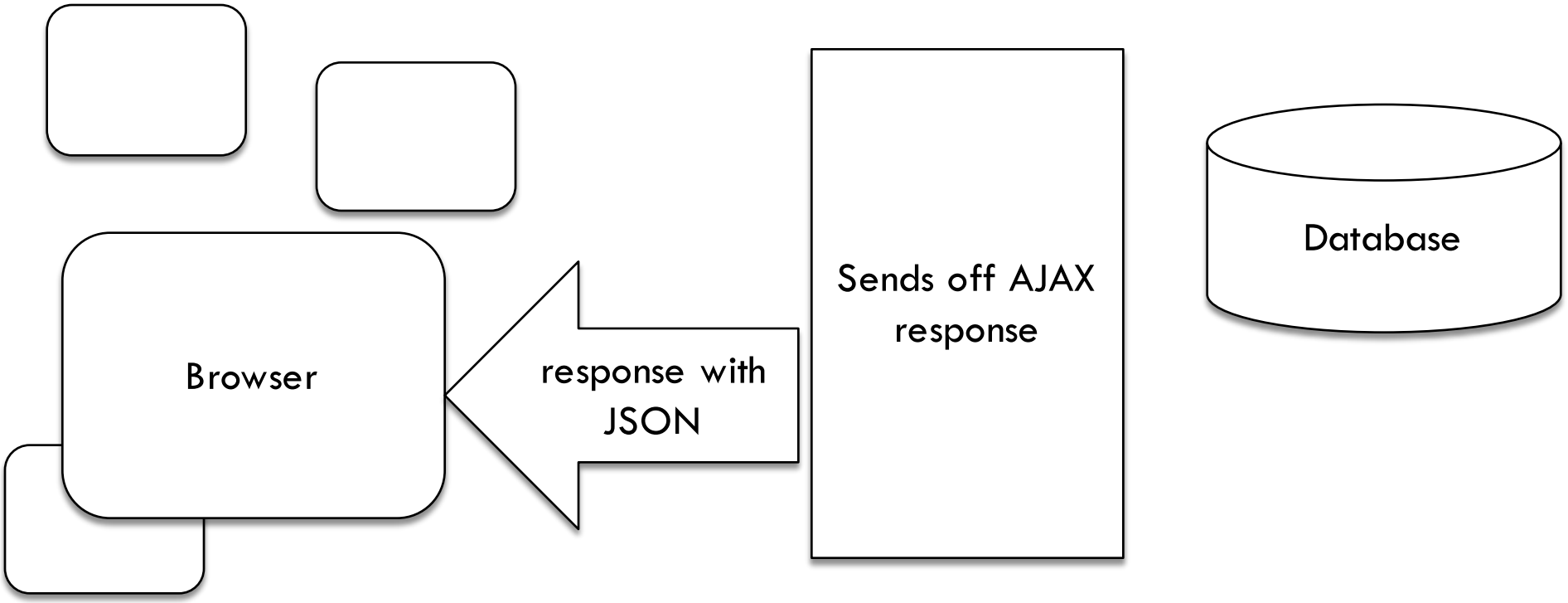


The server callback remembers response object. Many new HTTP requests may come in before this database request returns.

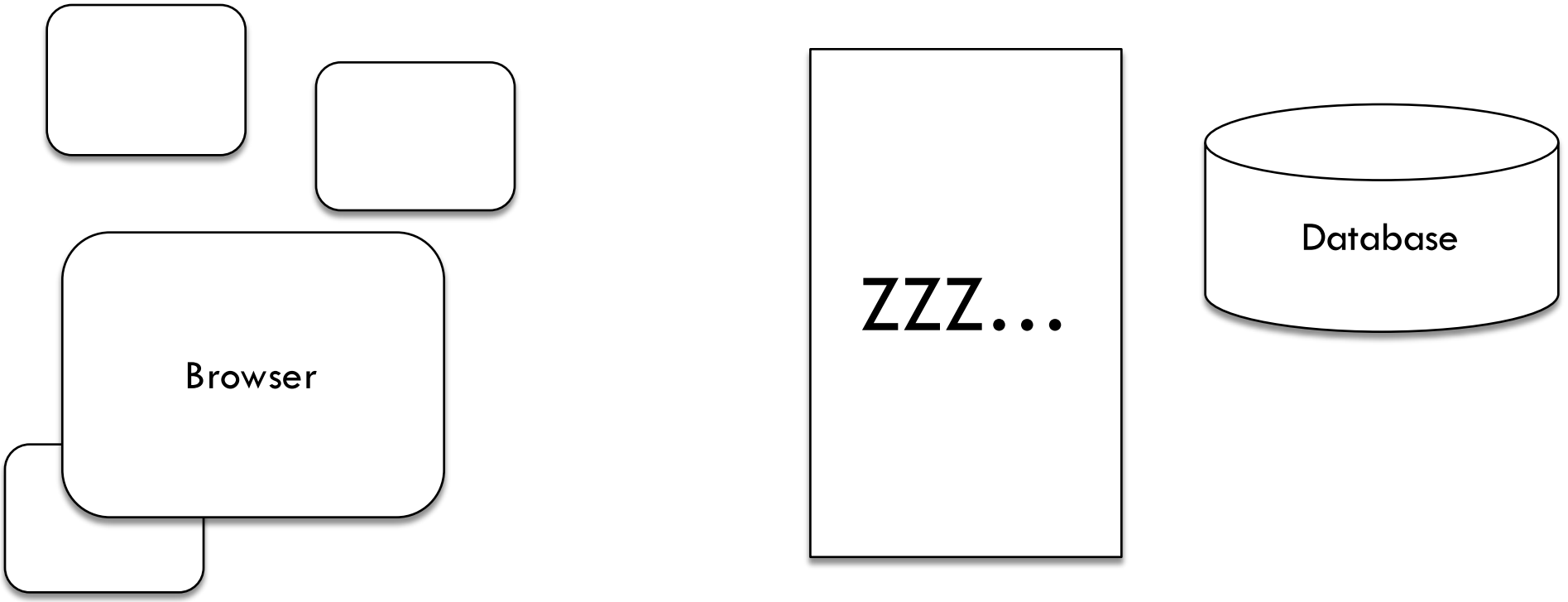
Server's view



Server's view

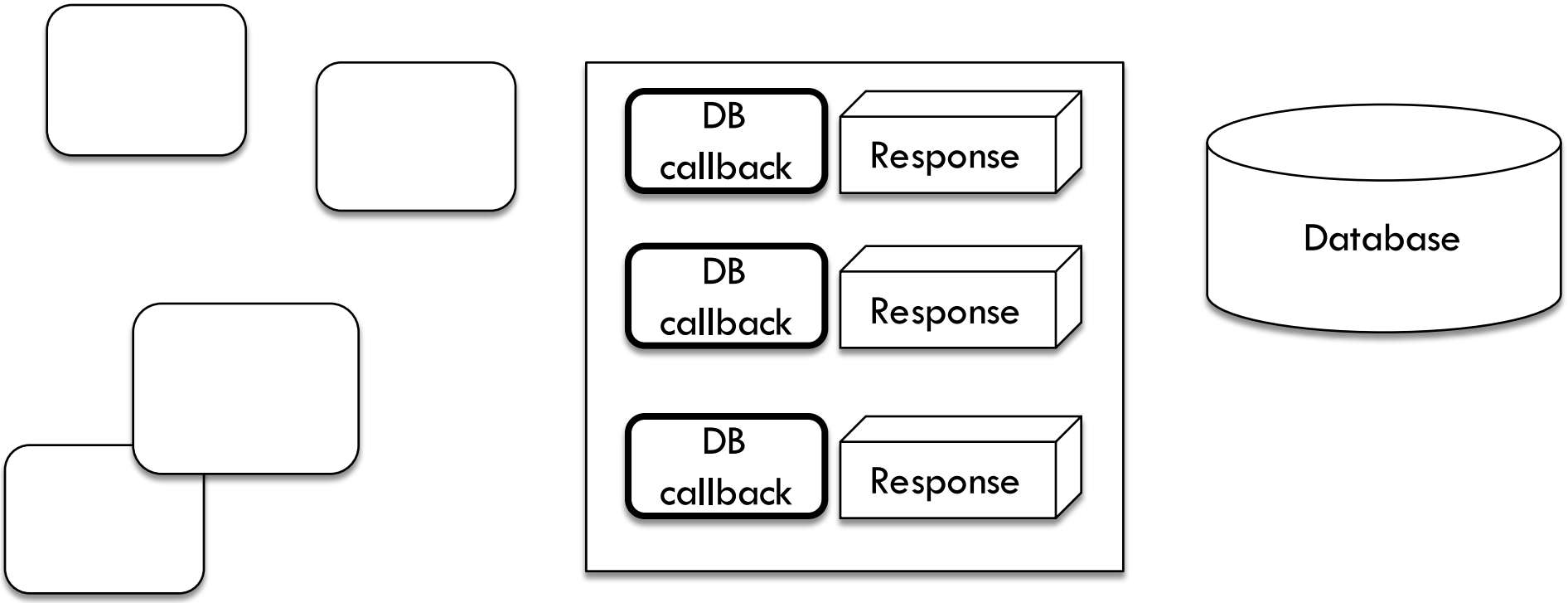


Server's view



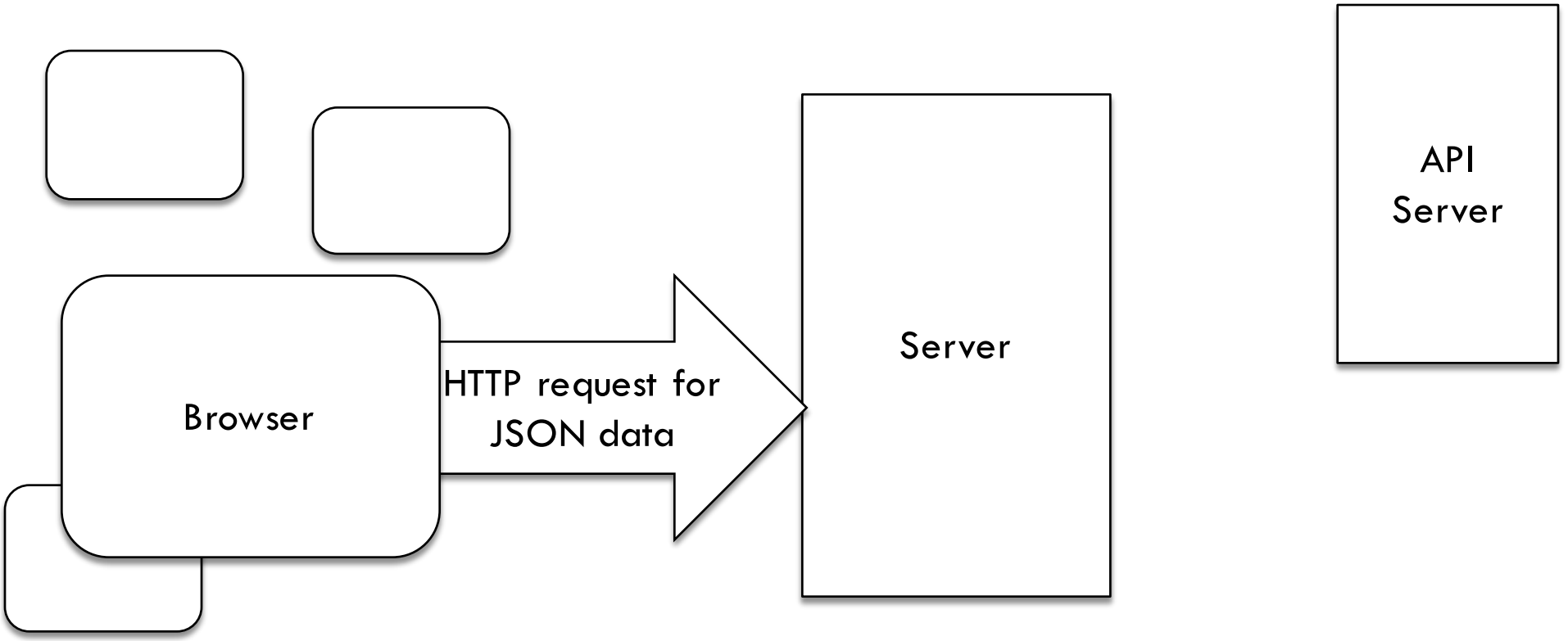
Server remembers nothing about transaction

Production server

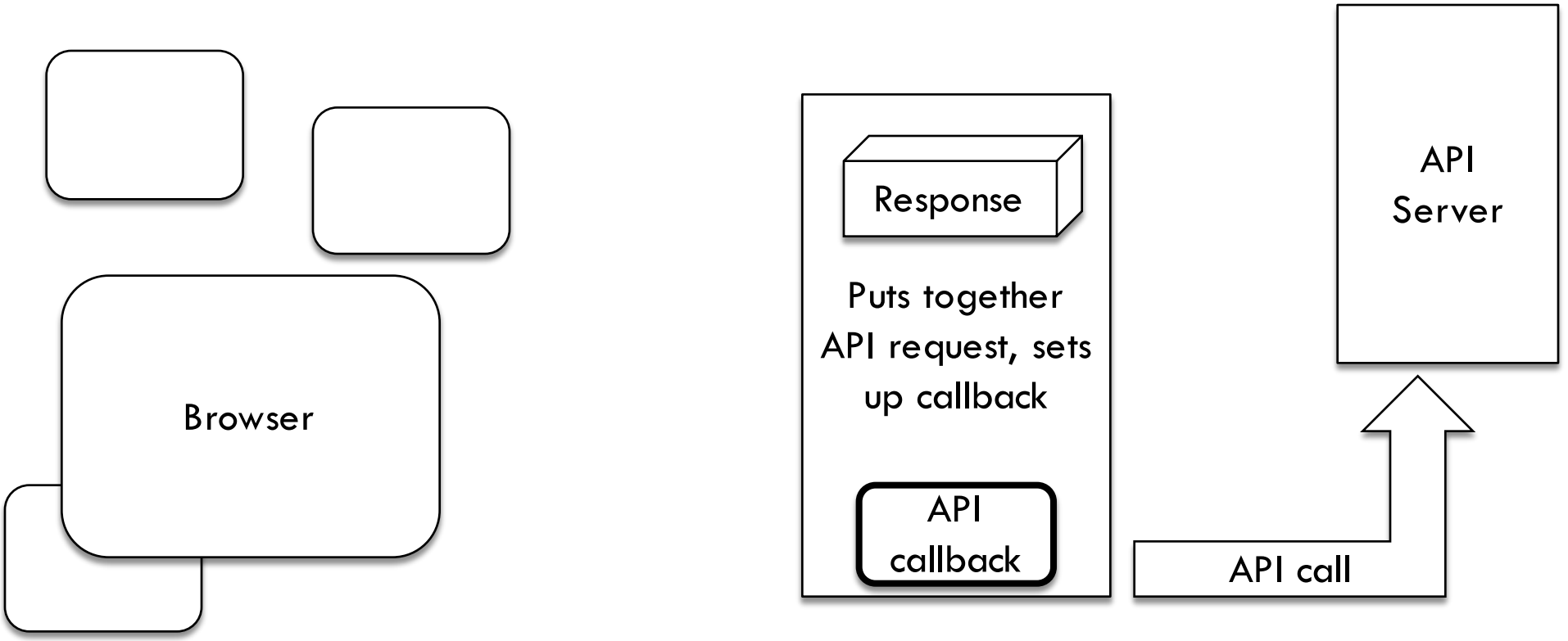


Many requests from different browsers, possibly many (response objects, callback) pairs waiting for DB responses at the same time.

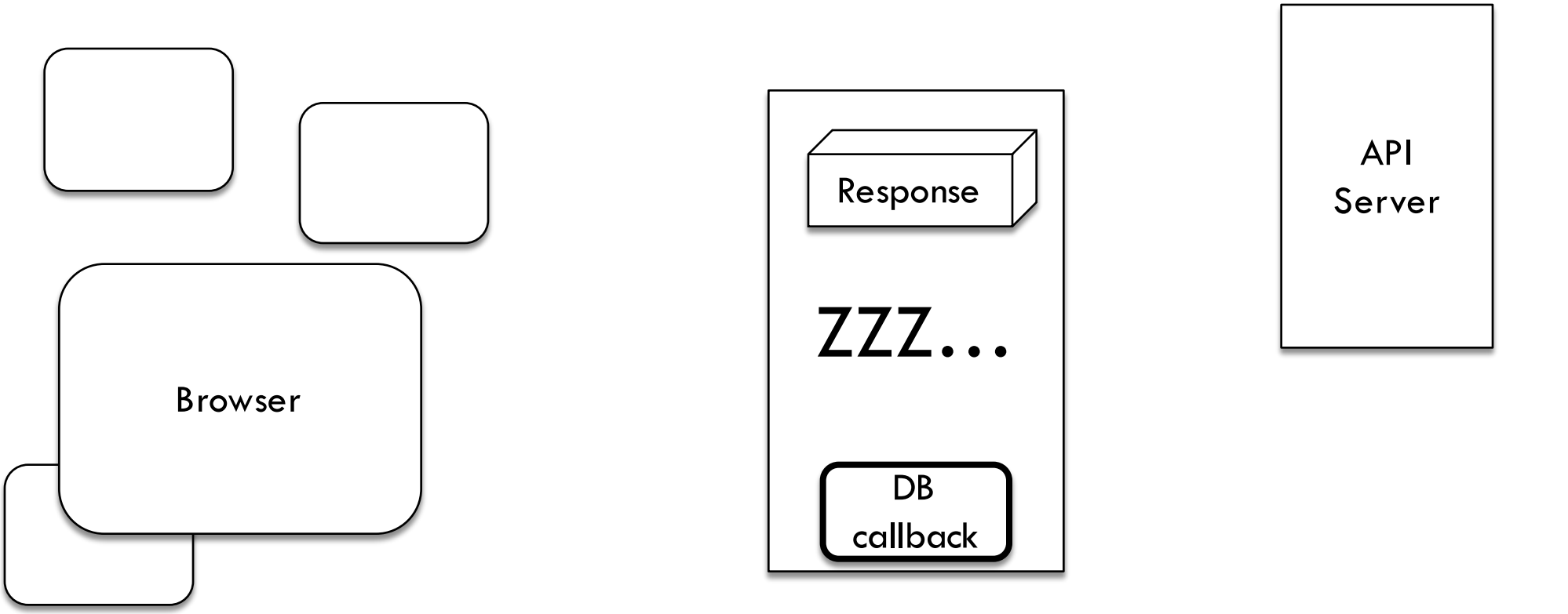
Server's view



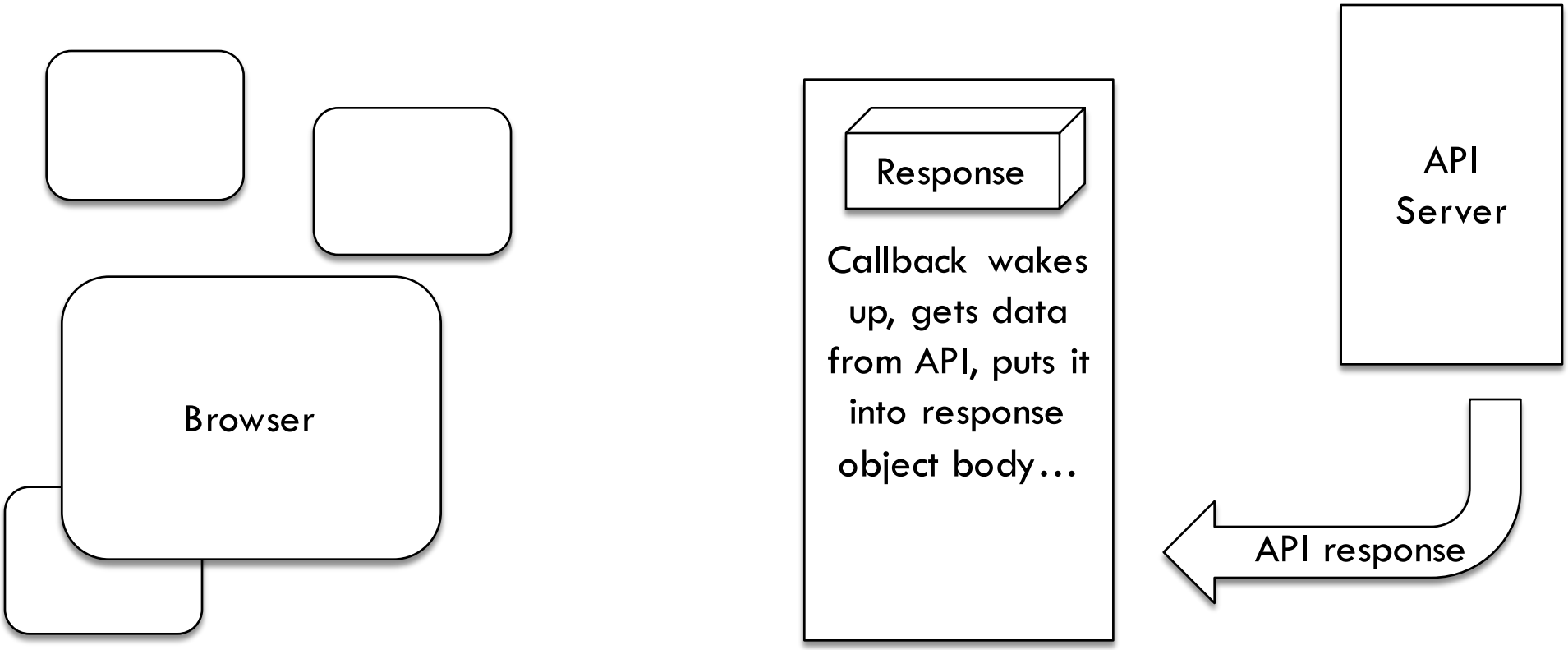
Server's view



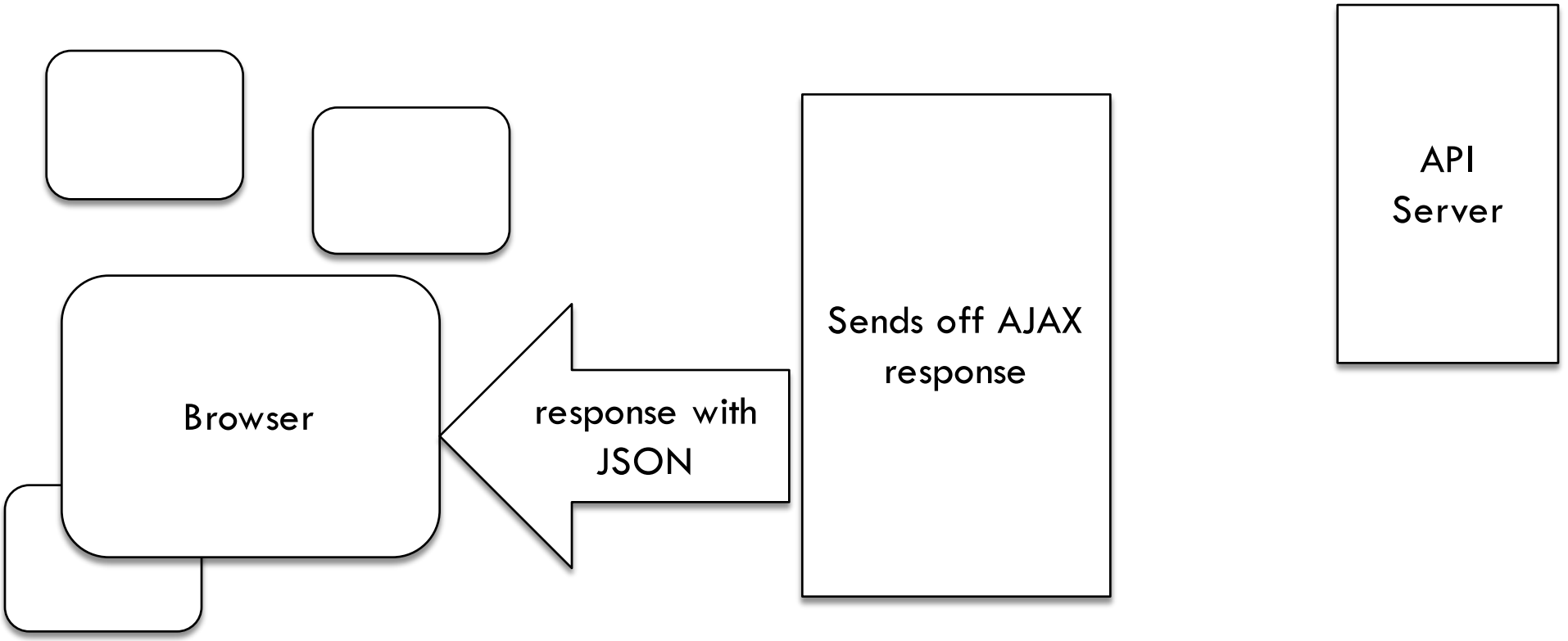
Server's view



Server's view



Server's view



Google Translate API

- Send English text, get translation back.
- How does it do the translation? "Deep learning", or perhaps "Magic"….
- To use Google Translate, you need a developer account on Google, and you'll need to set that up using Jason's directions.
- Instructions on Web site

Lots of ways to use GT API

- We'll use the very simplest
- Google issues you an API key (a code that identifies us)
- You include the API key in the URL of every GET or POST request you make:

```
const url =
   "https://translation.googleapis.com/language/translate/v2?key="+ APIkey;
```

Using the API

- Use it from Server, not Browser!

- Follows the usual 4-step plan
  - Make up request
  - Set up callback
  - Send off request
  - Handle result in callback

- But as usual has it's quirks

The HTTP request

- Use a POST request
- query in JSON in body
- Allows API user to send big hunks of text

HTTP POST
content-type: application/json
url:
https://translation.googleapis.com/language/translate/v2?key=???

```
{
        "source": "en",
        "target": "ko",
        "q": [
            "example phrase"
        ]
   }
```

Node request function

- To build a server HTTP request using Node, the usual way is to use the node request module

  npm install request

- This gives us the request function
- The functionality here is exactly the same as using the XMLHttpRequest object in the browser, but (because this is the Web) everything looks different…

Where are the four parts?

```
request(
    // first operand is object describing request
  {url: url,
  method: "POST",
  headers: {"content-type": "application/json”}
  json: requestObject },
      // second operand is callback
  APIcallback );
```

Callback function

```
function APIcallback(err, head, body) {
    if ((err) || (head.statusCode != 200)) {
        console.log("Got API error");
    } else {
        var newJSON = body.responses[0];
        // do stuff with response here
}}
```

- Expecting a callback function with 3 arguments
- Check for error!

Example response JSON


```
{ "data": {
    "translations": [
      {
        "translatedText": "예시 문구"
      } ]
  }
}
```