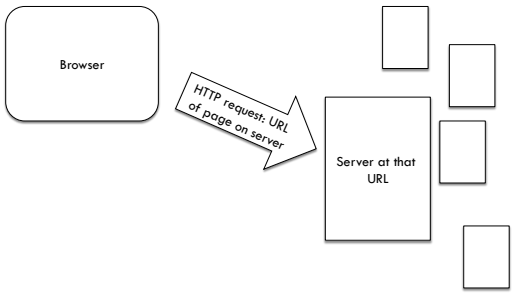
ECS 189
Web Programming

5/19

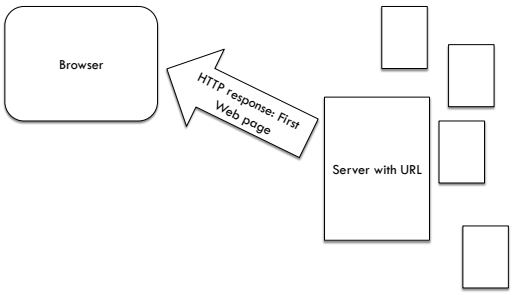
RESTful APIs

- We're implementing what is called a RESTful API
- ReST stands for "representational state transfer"
- The term was coined in 2000 by Roy Fielding, who at the time was PhD student at UCI.
- Some basic REST ideas:
  - Client (browser here) needs to know only a single URL to access the resource (photoBooth app); further interactions are learned as it goes along
  - Server does not need to know or remember anything about state of client

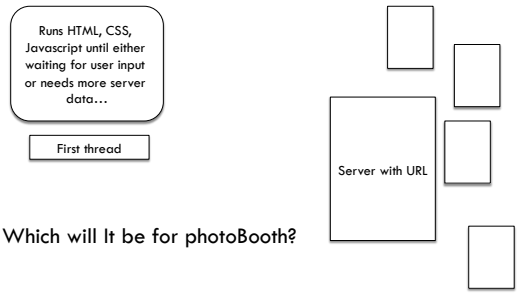
Browser's view



Browser's view

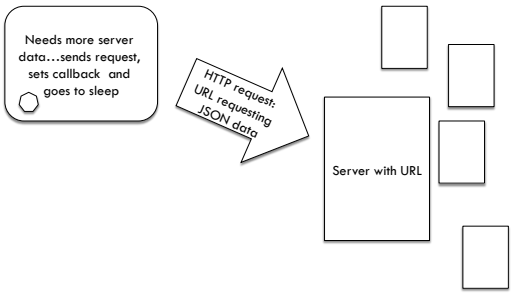


Browser's view

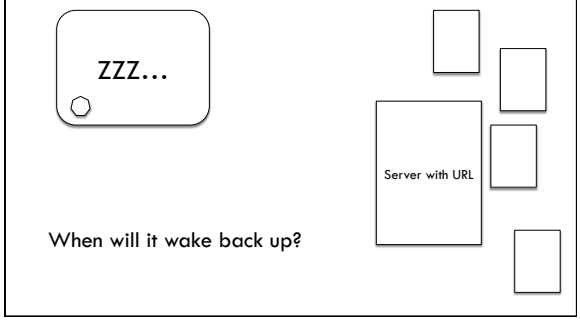


Which will It be for photoBooth?

Browser's view

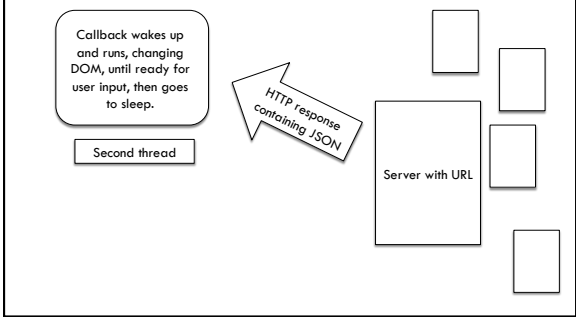


Browser's view

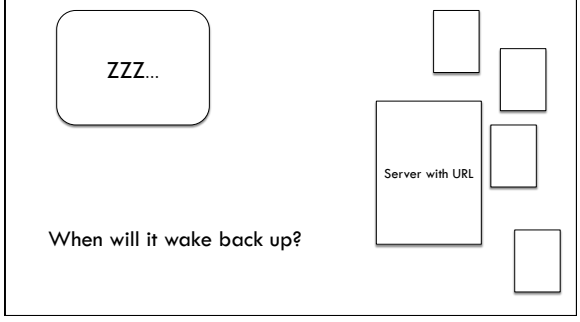


When will it wake back up?

Browser's view

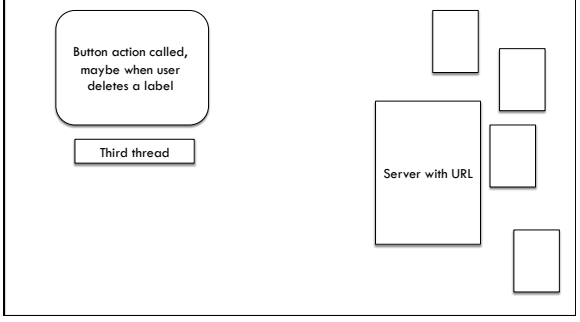


Browser's view

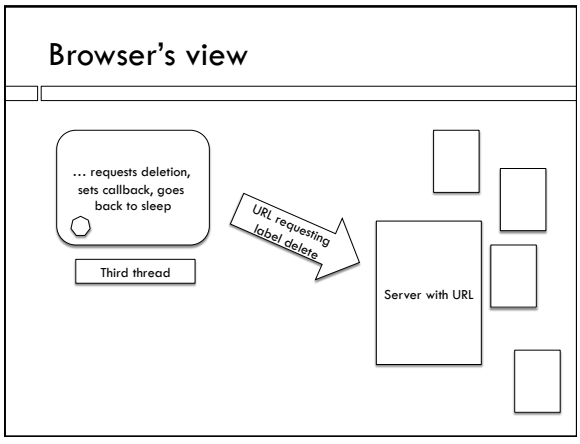


When will it wake back up?

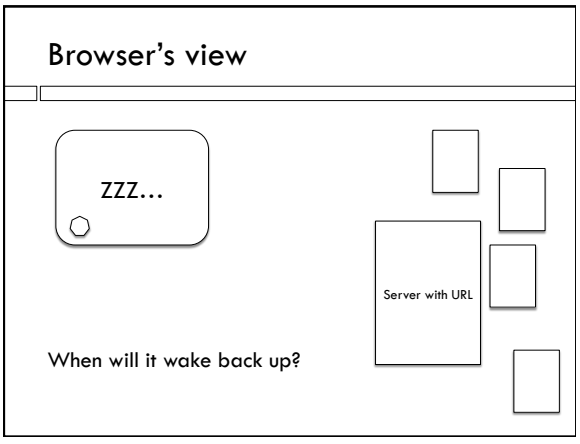
Browser's view



Browser's view

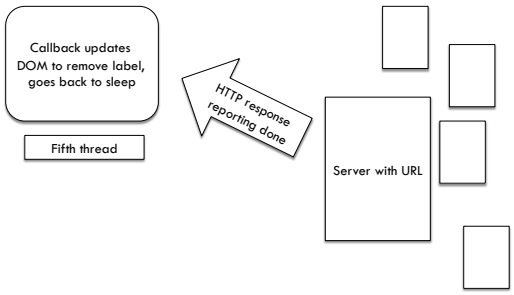


Browser's view

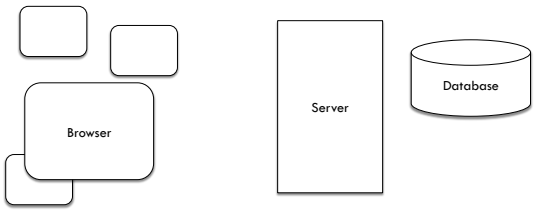


When will it wake back up?

Browser's view

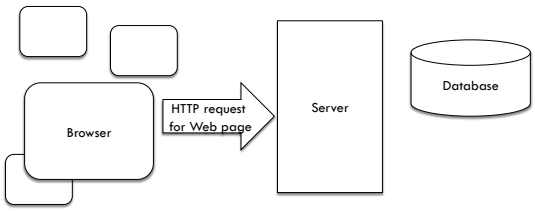


Server's view

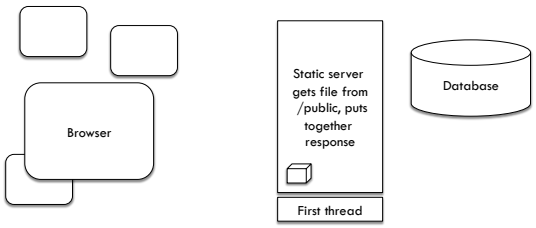


- The world is full of potential clients.

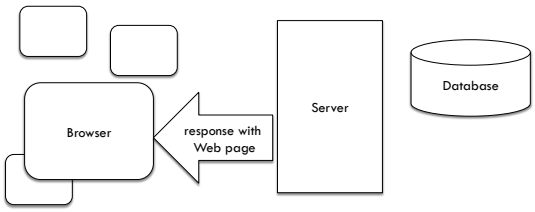
Server's view



Server's view

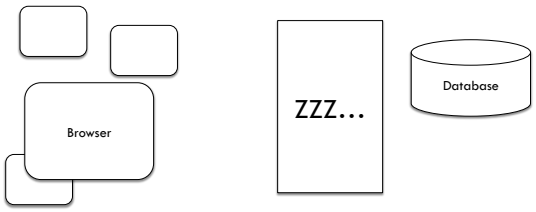


Server's view



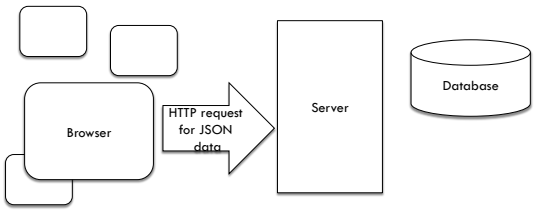
- Server sends back response object that is self-addressed to browser.

Server's view

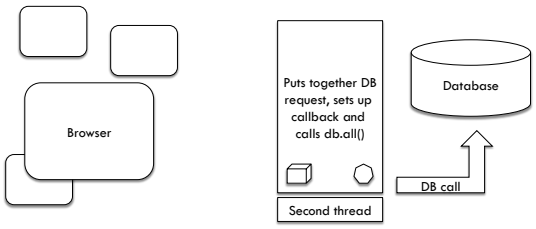


- The server remembers nothing about the transaction.

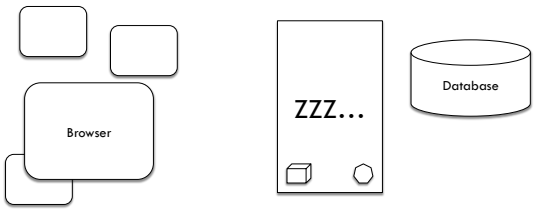
Server's view



Server's view

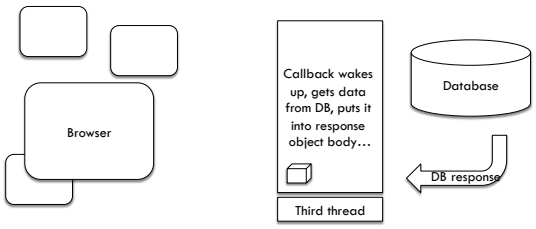


Server's view

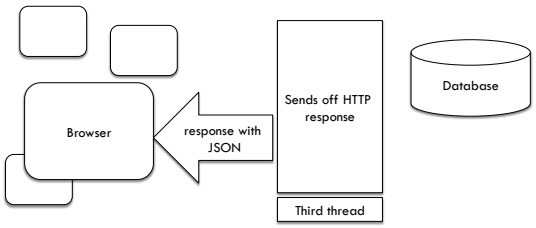


- The server callback remembers response object

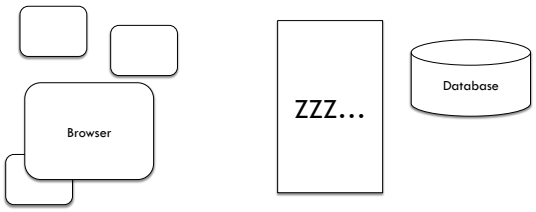
Server's view



Server's view



Server's view



- Server remembers nothing about transaction

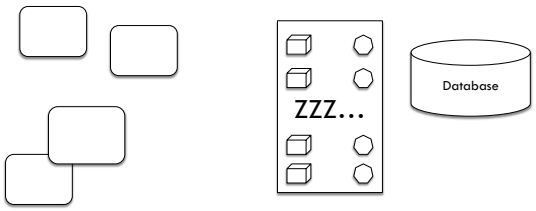
Last time

- DB operations using SQL
- One server thread might fire off multiple DB operations
- Order in which DB operations complete is not necessarily the order in which they are issued.
- When calling two DB operations in the same thread you can serialize them using db.serialize().
- We'll see a more typical approach in a bit.

DB operations in different threads

- Can't be serialized.
- On a production server, many HTTP requests might be in process at once.

Production server



- Many callback-HTTP response object pairs waiting for their DB responses

Database state

- No way to order these operations.
- Best we can hope for is to ensure that database is always in some meaningful valid state, eg. we don't have half a label written when it is read by someone else.
- Database systems work hard to ensure this.

Debugging – see database

```
amenta@cs189h:$ squlite3
 …Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open photos.db
sqlite> select * from PhotoLabels;
  hula.jpg|Dance, Event, Hula, Folk Dance|0
  eagle.jpg||0
  redwoods.jpg||0
sqlite> .quit
```

- You can also change items using "update", etc.

Debugging

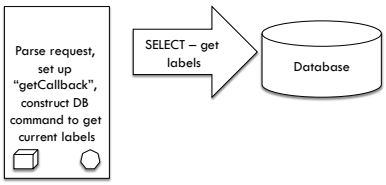
- SQLite3 is nasty to debug, run-time errors just crash, do not tell you where it failed
- Always handle errors and print something on the error callbacks, gives you a fighting chance at figuring out what is going on!

Editing the labels string

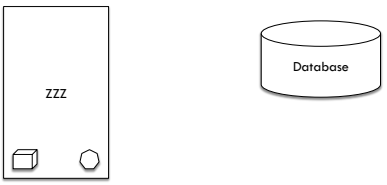
- Need to read it, then write it.
- Chain together the two operations, using their callbacks to guarantee sequence

Editing the labels string

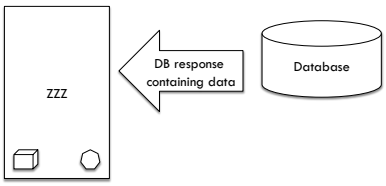
Editing the labels string



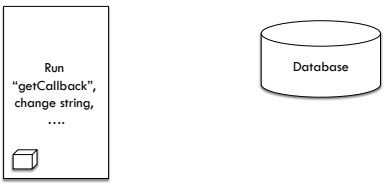
Editing the labels string



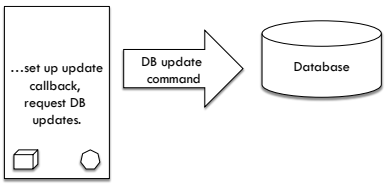
Editing the labels string



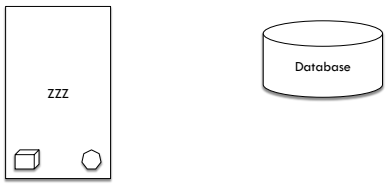
Editing the labels string



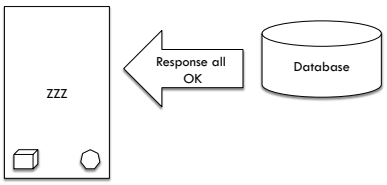
Editing the labels string



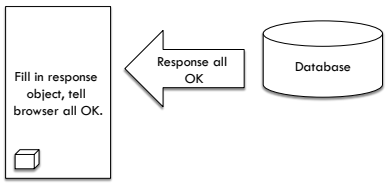
Editing the labels string



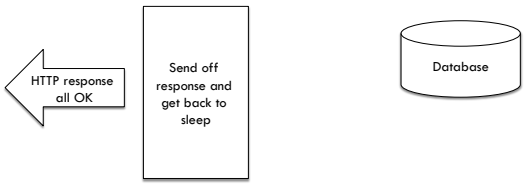
Editing the labels string



Editing the labels string



Editing the labels string



Parse the query

```
var querystring = require('querystring');
function answer(query, response) {
   queryObj = querystring.parse(query);
   if (queryObj.op == "add") {
        var newLabel = queryObj.label;
        var imageFile = queryObj.img;
         if (newLabel && imageFile) {
                …
```

Call DB to get current labels

```
db.get( 'SELECT labels FROM photoLabels WHERE
  fileName = ?',
  [imageFile],
  getCallback);
```

- Using "fill in the blanks" Node.js SQLite3 syntax

First callback

- Defined inside "answer" so it has access to all local variables – because it's in the closure of "answer"!

```
function getCallback(err,data {
  …handle error…
  db.run('UPDATE photoLabels SET labels = ? WHERE
  fileName = ?',
  [data.labels+", "+newLabel, imageFile],
  updateCallback);      }
```

Second callback

- Also defined inside "answer"

```
function updateCallback(err) {
  …handle error…
      response.status(200);
      response.type("text/plain");
      response.send("added label "+newLabel+" to
      "+imageFile);  }
```

- Uses Express syntax, could also have used Node

Module for queries

- Putting all the server code in one file will get messy
- The code for answering queries will get big
- Let's put it in it's own module
- To make a file into a module add lines at the end to tell it to export stuff

```
// function answer visible from outside as "answer"
exports.answer = answer;
```

Using the module in server file

- Put pathname to file containing module as name of module

```
var queries = require("./queries");
```

- Function to answer queries is now visible in tripleThreatServer as:

```
queries.answer(request,response);
```