CMPS 356

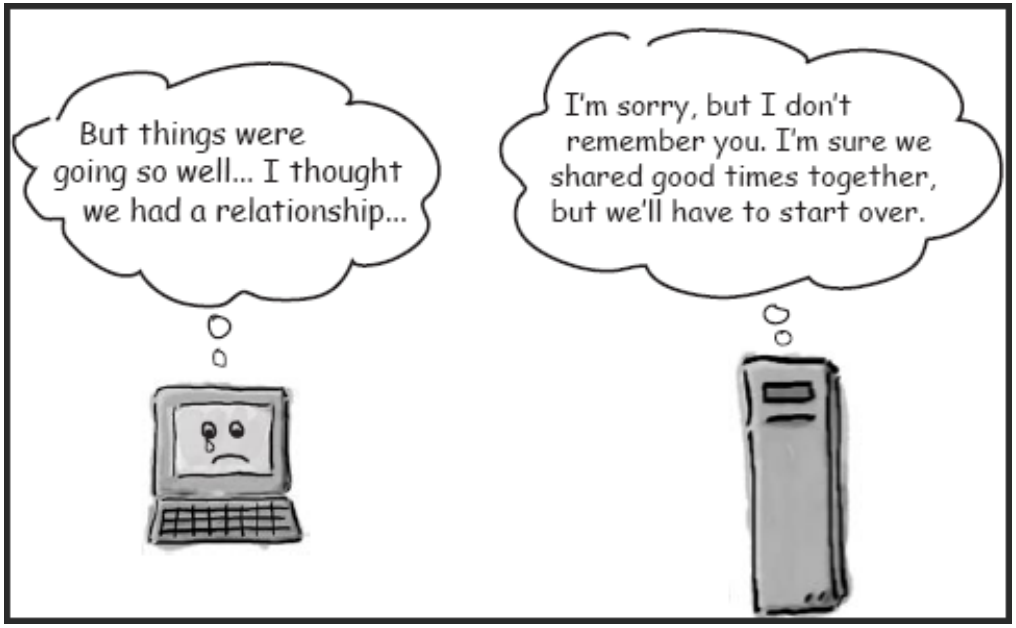
Session Management

Dr. Abdelkarim Erradi
Dept. Computer Science & Engineering
QU

Outline

1. Session tracking
2. Cookies
3. Data Sharing
4. HTML5 Local Storage

Session Tracking



Session is a mechanism used by Web Apps to **maintain state about a series of requests from the same user** (that is, requests originating from the same browser) within a period of time

Back

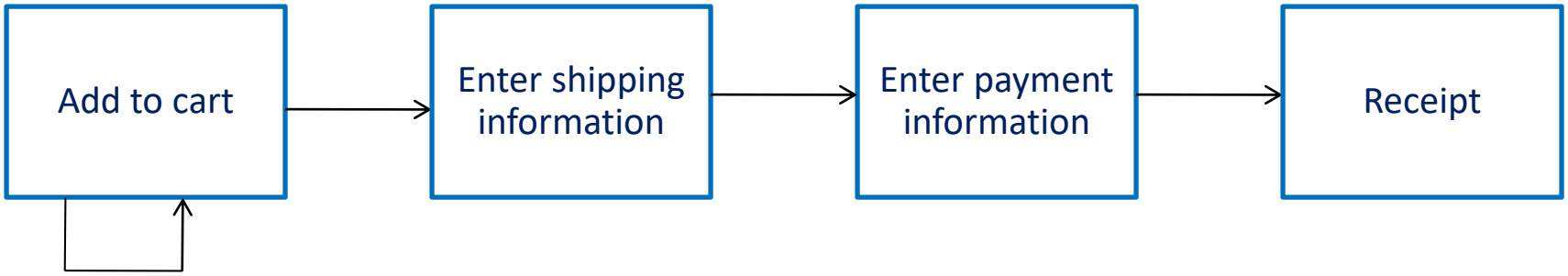
Need for Session Tracking

- HTTP is a "**stateless**" protocol
  - **Does not support conversations**
  - Has No easy way to distinguish between clients
  - This is good for scalability … but keeping state is needed for some scenarios

Issue

Solution

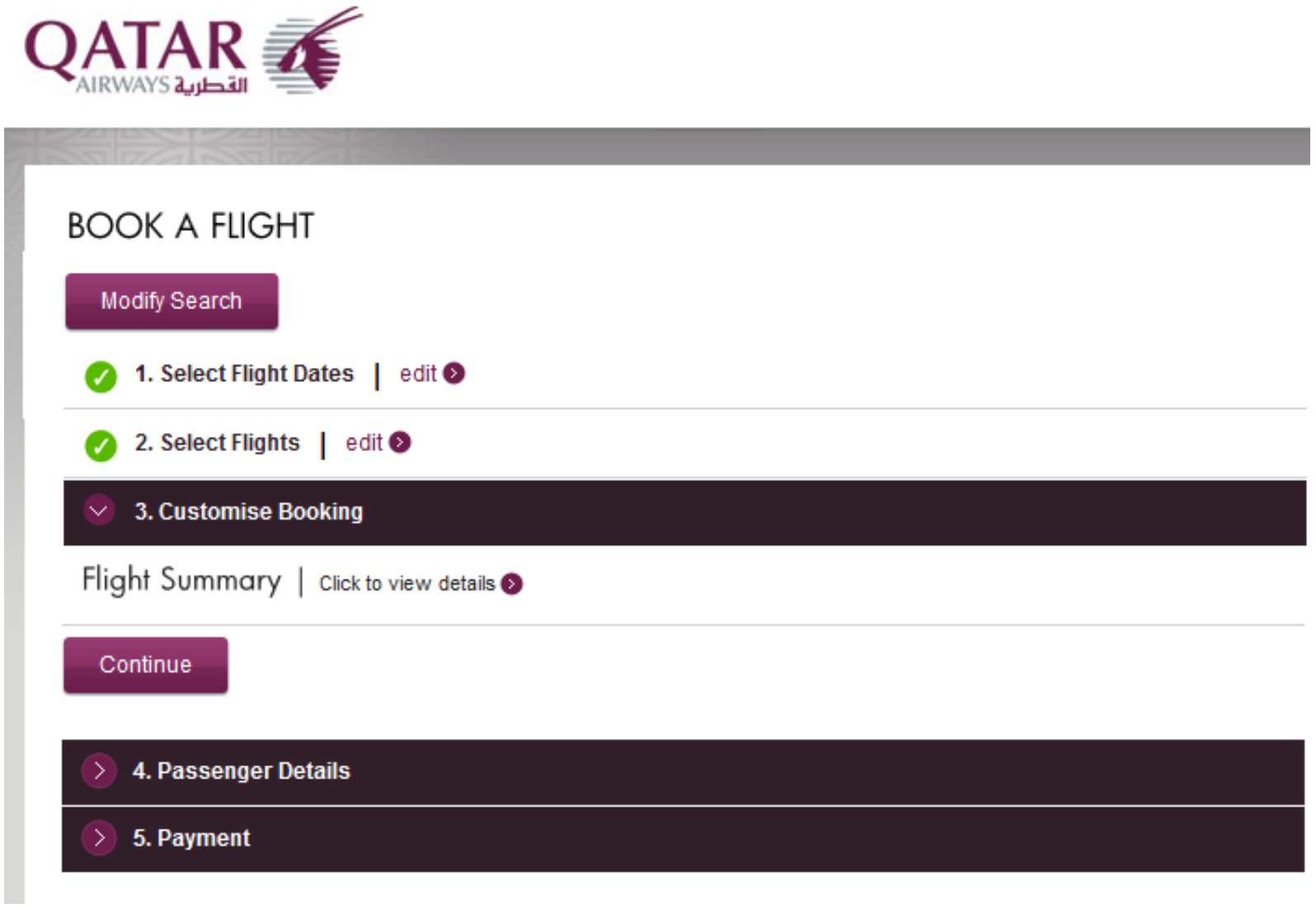
- Session: maintain **state** between set of interactions with a user to accomplish goal
  - E.g., shopping cart in online store
  - Server may have to simultaneously manage thousands of sessions



An example where State Management is needed

- Checkout Process

Stateful design use cases - Wizards & conversation-oriented web apps are good examples



Session Tracking Basics

- Sessions are data objects that can be associated with a user
  - The objects exist only on the server (memory or database)

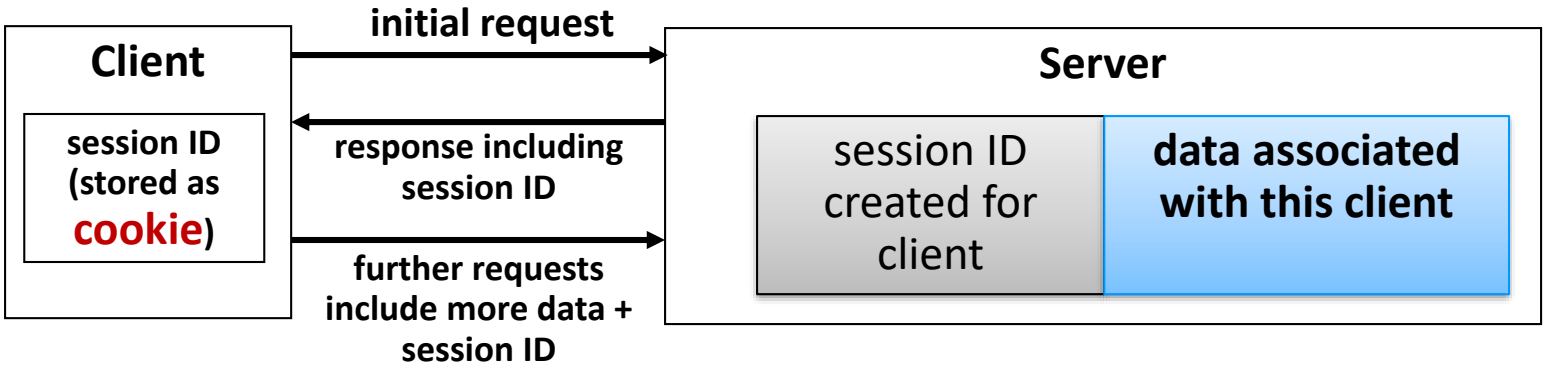
- Access the session object
  - Call req.session to get the Session object then read and write session data

- ```
  req.session.destroy((err) => {
      res.redirect('/login');
  });
  ```
  ```

Session Handling

Session does not travel across network. Only its unique identifier does

- Server assigns each new client a **unique Session ID** at start of the session. This is one automatically first time a **req.session** is called
- Server **Passes** the **session ID** as a cookie to client as part of the response
- Client **Stores** the Session ID as a **cookie**
- Client **Passes** the Session ID back to server with subsequent requests
- Server stores client data in a server session as a **table** indexed by Session ID



Cookies

"Set-Cookie" is just another header sent in the response.

Here's your cookie with the session ID inside...

HTTP Response

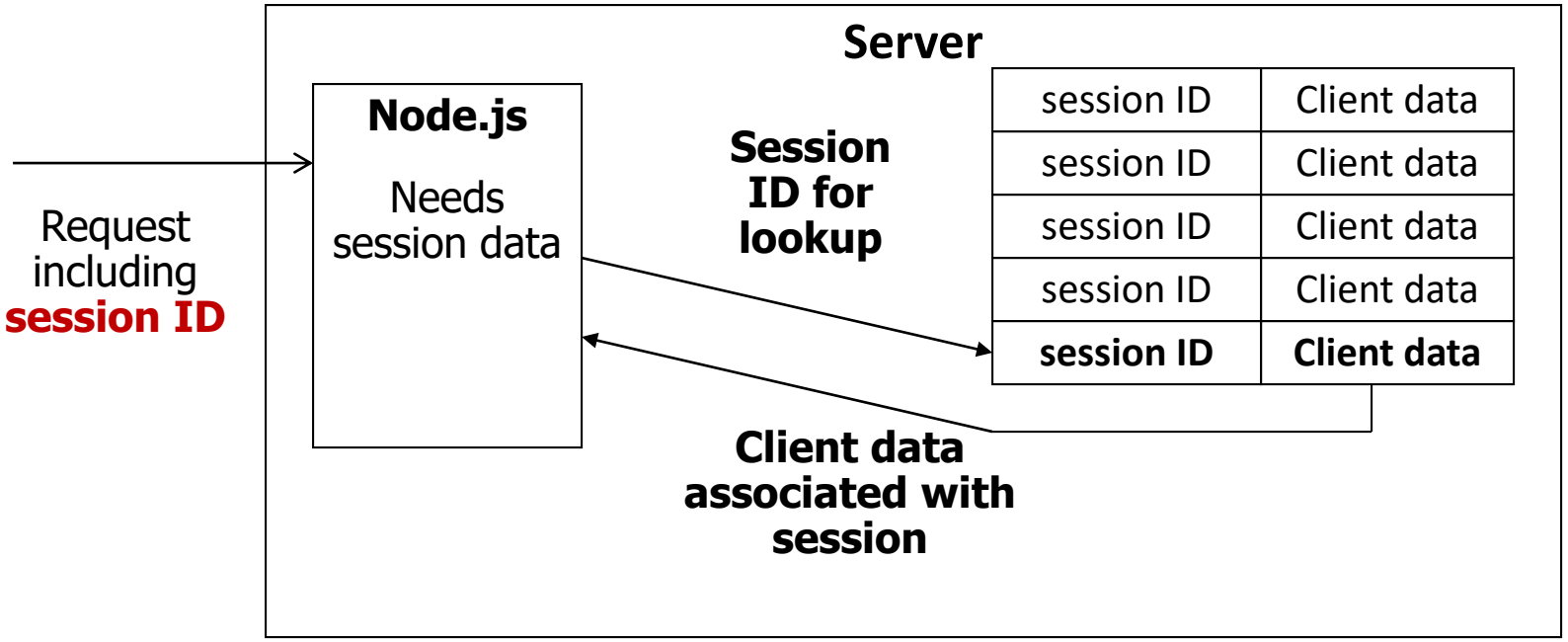
OK, here's the cookie with my request

"Cookie" is another header sent in the request.

HTTP Request

Associating Session Data

- Node.js can store data associated with session ID
- Node.js can look up that data in future when passed the session ID in the request



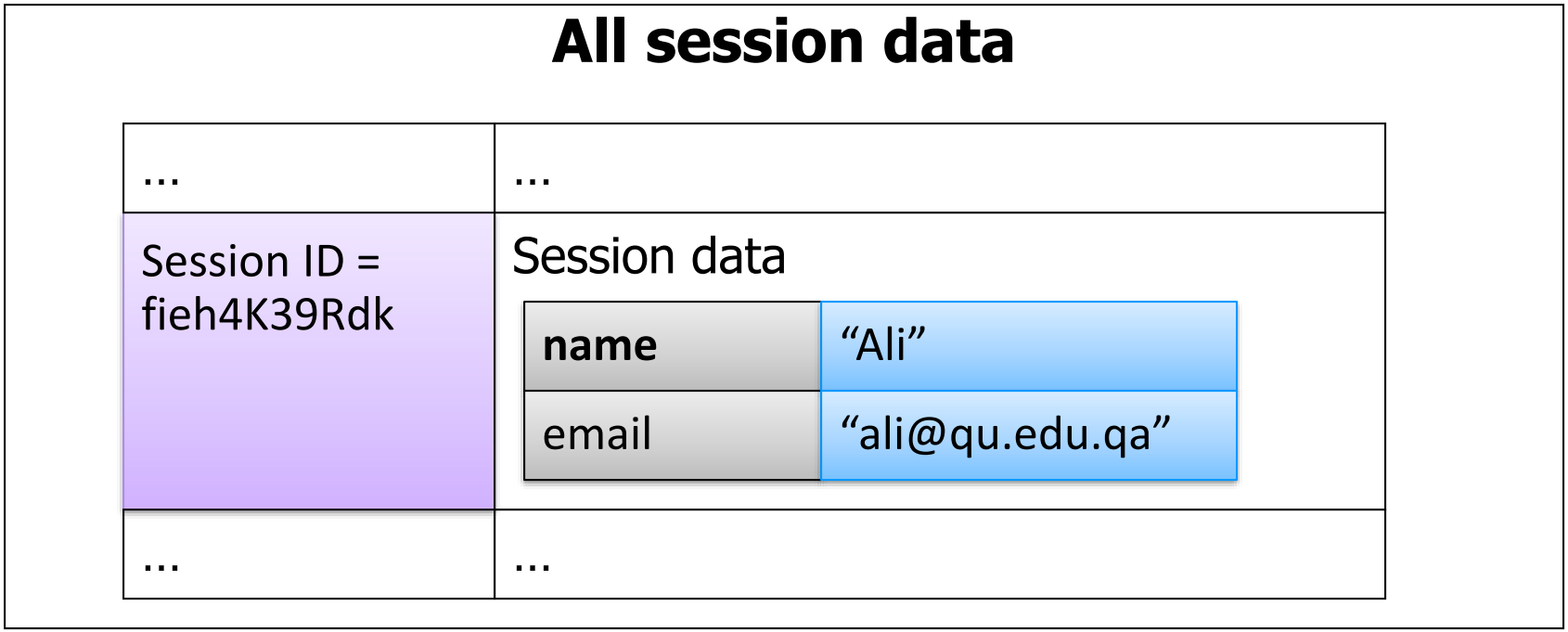
Storing Session Data

- **Syntax:**

```
req.session.name = value;

let myVar = session.name;
```

- Session data stored as name/value pairs



Session Expiration

- Can set time until session expiration using maxAge parameter

```
// ...
app.use(express.cookieParser());
/* only the encrypted session ID is stored in the client side cookie.
The expired date is stored as req.session.cookie.expires in the server side
*/
app.use(express.session({secret:'yoursecret', cookie:{maxAge:6000}}));

//This custom middleware will check whether current session is expired.
app.use(function(req, res, next) {
    // if now() is after `req.session.cookie.expires` then redirect to login
    next();
});

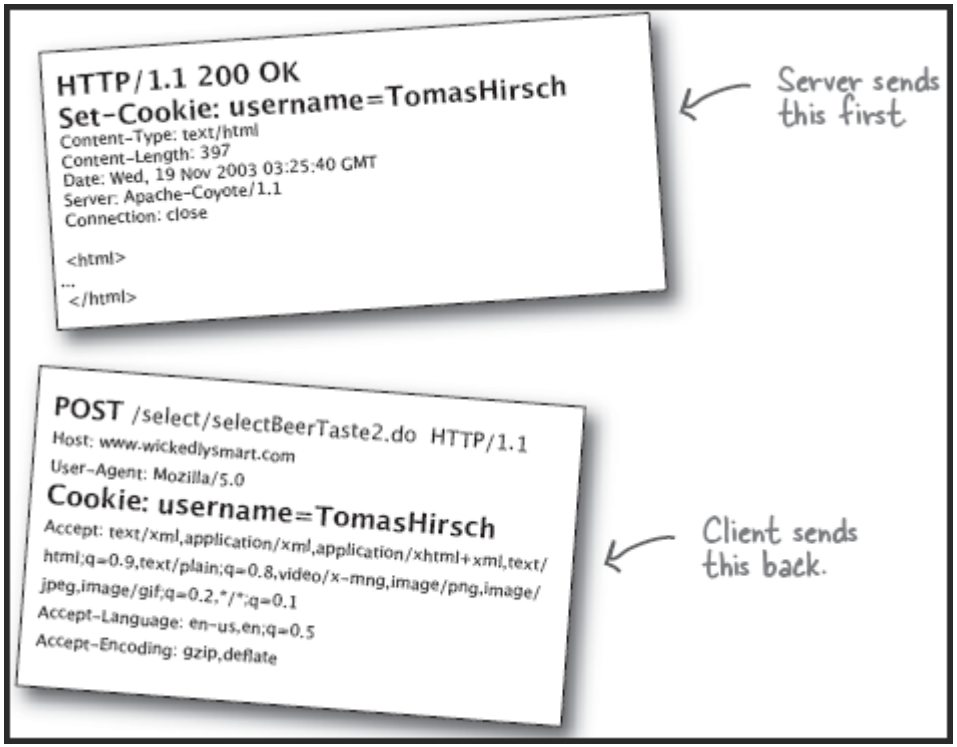
// ...
```

- Session expires if no request within time limit
  - Session inactive
  - Session id and all attributes destroyed

Cookies

Watch This Video!

https://www.youtube.com/watch?v=I01XMRo2ESg



Back

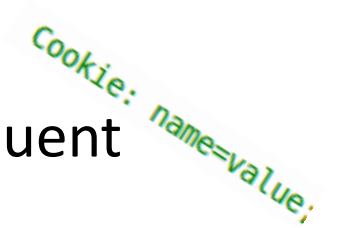
Cookies

- Idea
  - Server sends a simple name and value pair to client.
  - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings).
- Value limited to 4KB
- Has expiration date, and a server name (returned to same host and not to others)
- Cookie is sent in HTTP header of the response

```
res.cookie('varName', 'varValue'); // to send a cookie
```

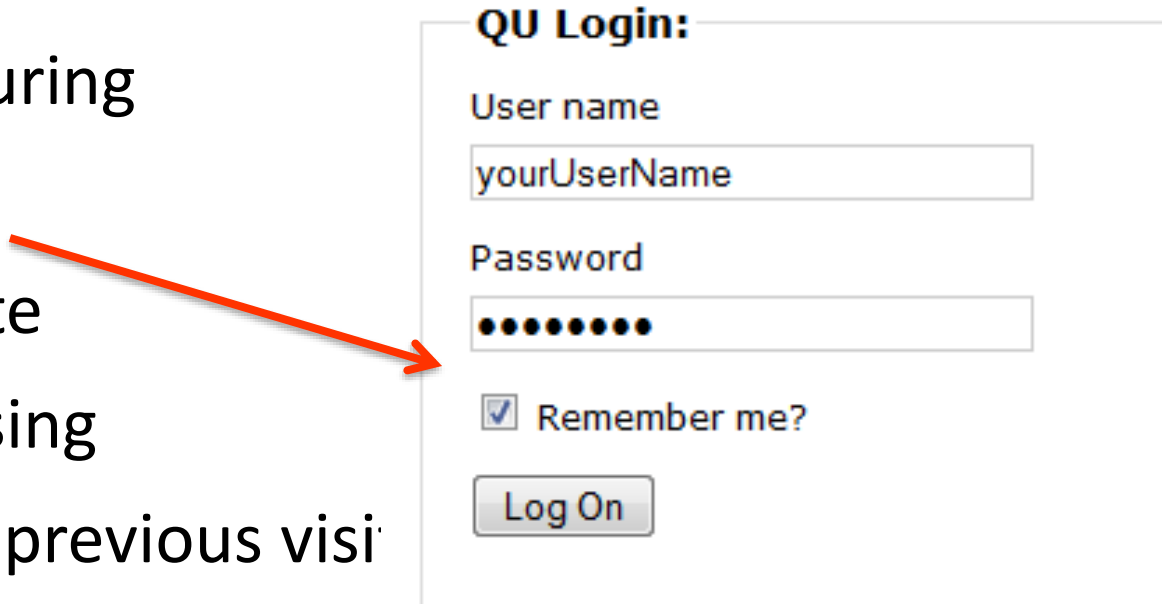
- Cookie is returned to server in HTTP header of subsequent request

```
cookies = req.cookies;  //to get cookies
```



The Potential of Cookies

- Typical Uses of Cookies
  - Identifying a user during an e-commerce session
  - Implement famous
  - ‘Remember me’ during
  - Login
  - Customizing a site
  - Focused advertising
  - Store info about previous visi
    the visitor did



Cookies and Focused Advertising

Amazon.com home page for new visitor or visitor with cookies disabled.

Amazon.com home page for repeat visitor. Books shown are based on prior history.

While not just use the client IP address to identify the client

- There is no guarantee that the same IP address means the same user
- … or that the same user always accesses the site with the same computer
- And importantly, most machines' IP addresses actually change over time...

Some Problems with Cookies

- The problem is privacy and security risks
  - Servers can remember your previous actions
  - Servers can share cookie information through use of a cooperating third party like doubleclick.net
  - Poorly designed sites store sensitive information like credit card numbers directly in cookie
  - Hacker can steel your cookies and hijack your session or get access to sites under your name, and essentially be logged in as the user associated with it!
    - It is frightening thing if a malicious individual finds out the value of your cookie!

=> Don't put sensitive info in cookies

Summary - Cookies

- Basic functionality
  - Cookies are name/value pairs sent from server to browser and *automatically* returned when the same page (or possibly same site or domain) is visited later
- Cookies let you
  - Track sessions
  - Permit users to avoid logging in (when rememberMe is ticked)
  - Customize sites for different users
  - Focus content or advertising
- Setting cookies
  - res.cookie('varName', 'varValue'); // to send a cookie
- Reading cookies
  - cookies = req.cookies; //to get cookies

Data Sharing

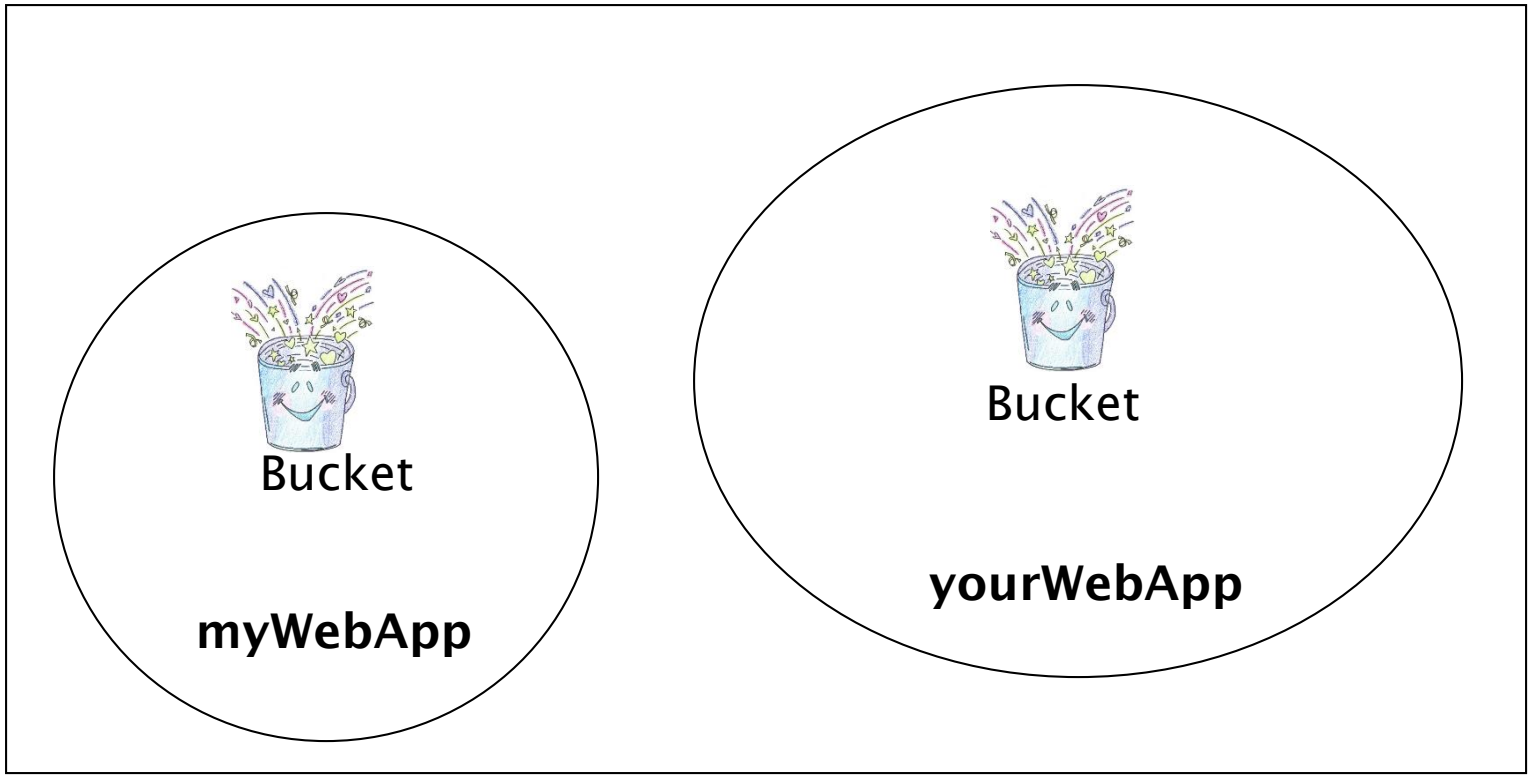
Back

Web container 3 built-in data sharing ‘buckets’

- **Application scope -** 1 per web-application
  - Accessible by the entire web application
  - Attributes are stored in the global object
  - Available **for the lifetime of the application**
- **Session scope**
  - **1 per client per web-application**
    - used for storing session data
  - Attributes are stored in the **req.session** object
  - **Available for the life of the session**
- **Request scope**
  - 1 per request
  - Attributes are stored in the **req** object
  - Available for the life of the request (until the request handling **completes**)

Application scope ‘Bucket’

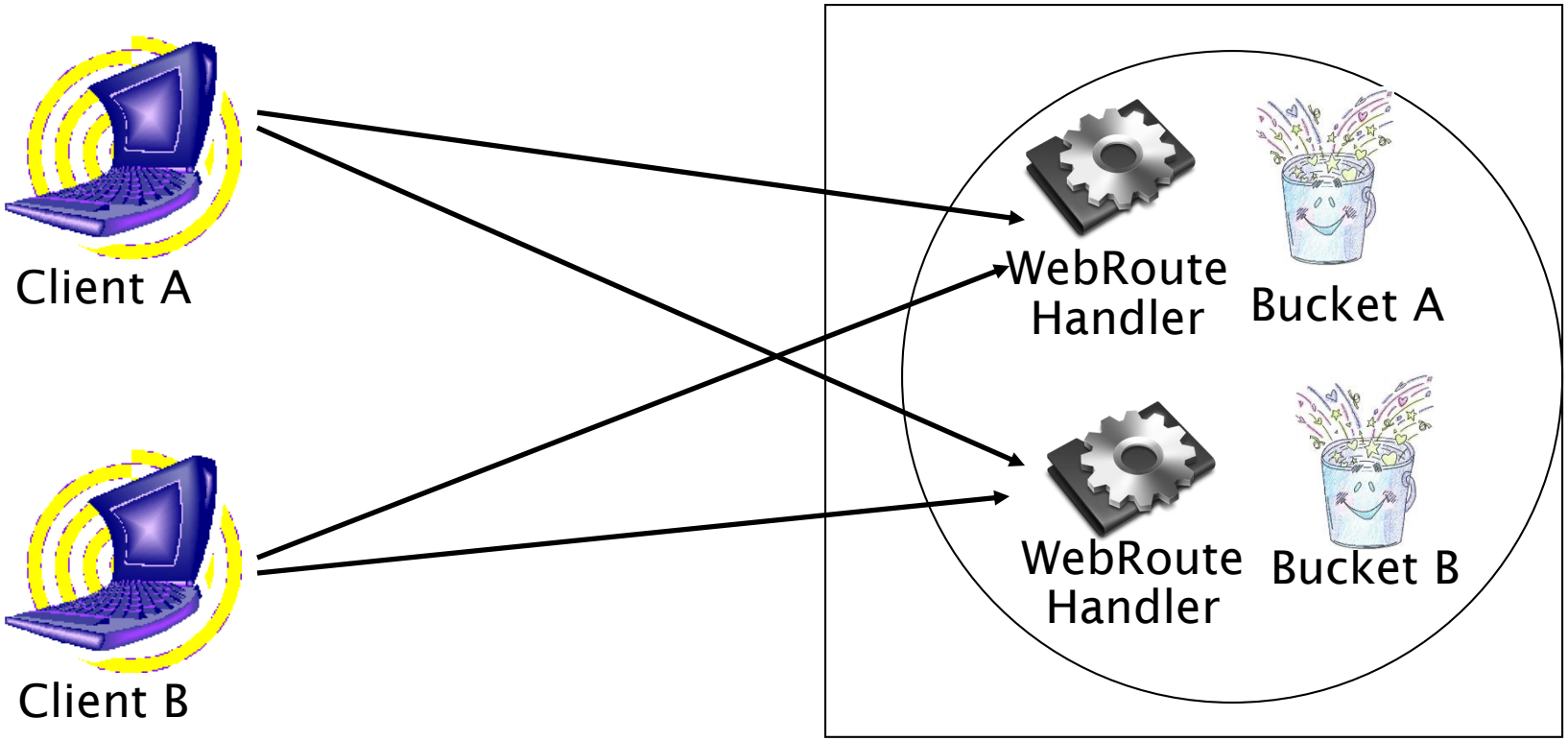
1 per web-application




```
global.companyName = ‘QU’;
Console.log(global.companyName);
```

Session scope ‘Bucket’

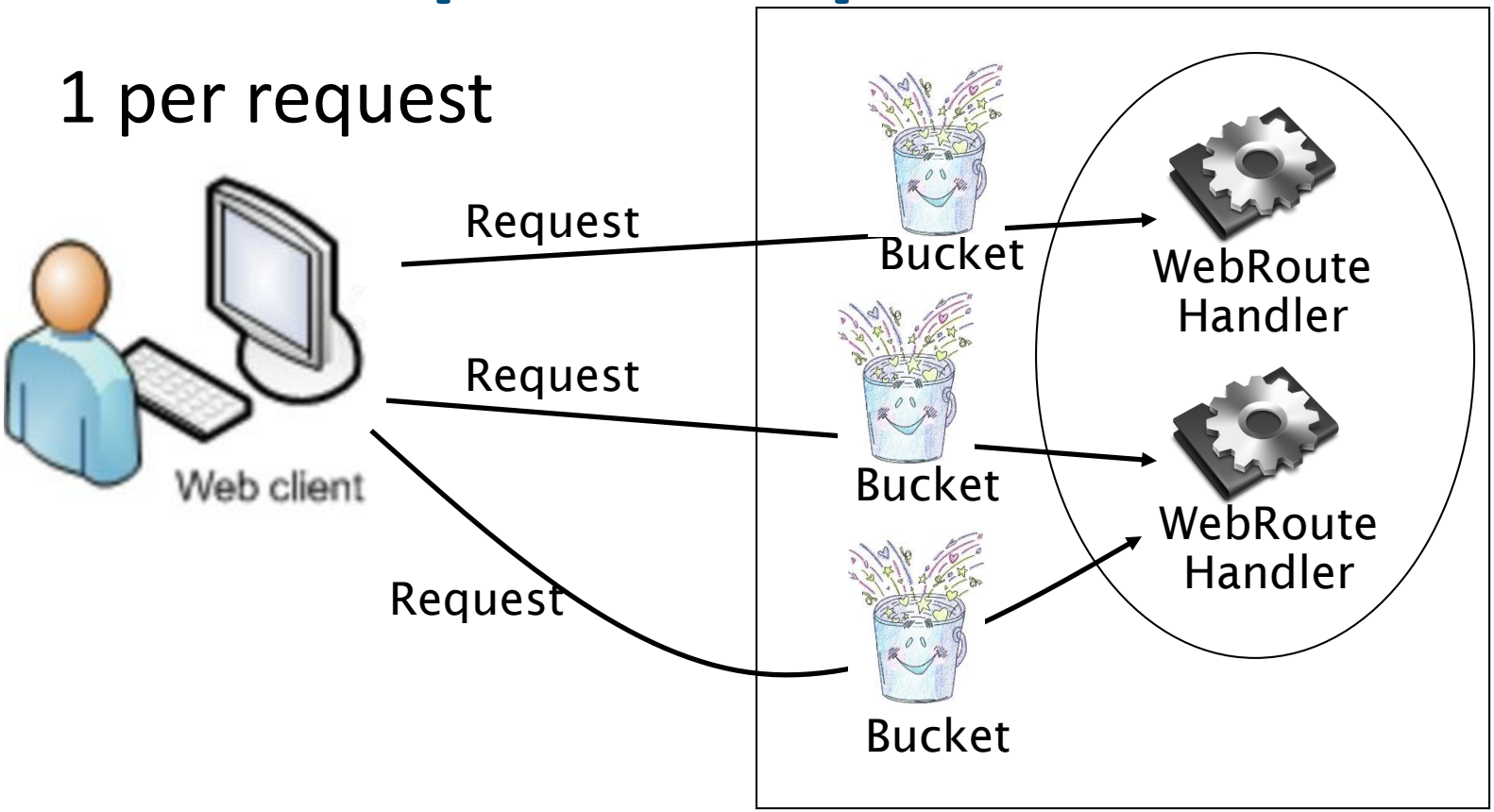
1 per Web client




```
req.session.name = value;
let name = session.name;
```

Request scope ‘bucket’

1 per request




```
req.name = value;
let name = req.value;
```

HTML5 Local Storage

HTML5 Local Storage

- Cookies are no longer the only way to store data on the client machine.
- HTML5 introduces local storage to store set of name value pairs directly accessible with **client-side** JavaScript
- Data placed in **local storage** is per origin (the combination of protocol, hostname, and port number) and persists after the browser is closed
  - the data is available to all scripts loaded from pages from the same origin that previously stored the data.
- **Session storage** is per-origin-per-tab and data are available until the user closes the tab/browser

Simple API

- Store

```
localStorage.lastname = "Smith";
```

- Retrieve

```
Console.log(localStorage.lastname)
```

- Remove

```
localStorage.removeItem("lastname");
```

- Remove all saved data

```
localStorage.clear();
```

Cookies vs. Local Storage

- Cookies are auto-included with every HTTP request
- Cookies are limited to about 4 KB

- Data in local/session storage are NOT auto-included with every HTTP request
- Storage limited to about 5 MB

- Both cookies and browser storage can be cleared by the user and should not be completely relied upon for storage