Cookies and Sessions

- Recall that HTTP is stateless: it simply allows a browser to request a single document from a web server and doesn't remember or keep track of anything between invocations (short-lived).

- Every resource that is accessed via HTTP is a single, one-off request with no threaded connection between them.

- Being stateless makes a lot of sense when sharing static information (like html, pdf, images. But what if you want a stateful setup for your web application?

- You need:

  - A way to remember/track states.

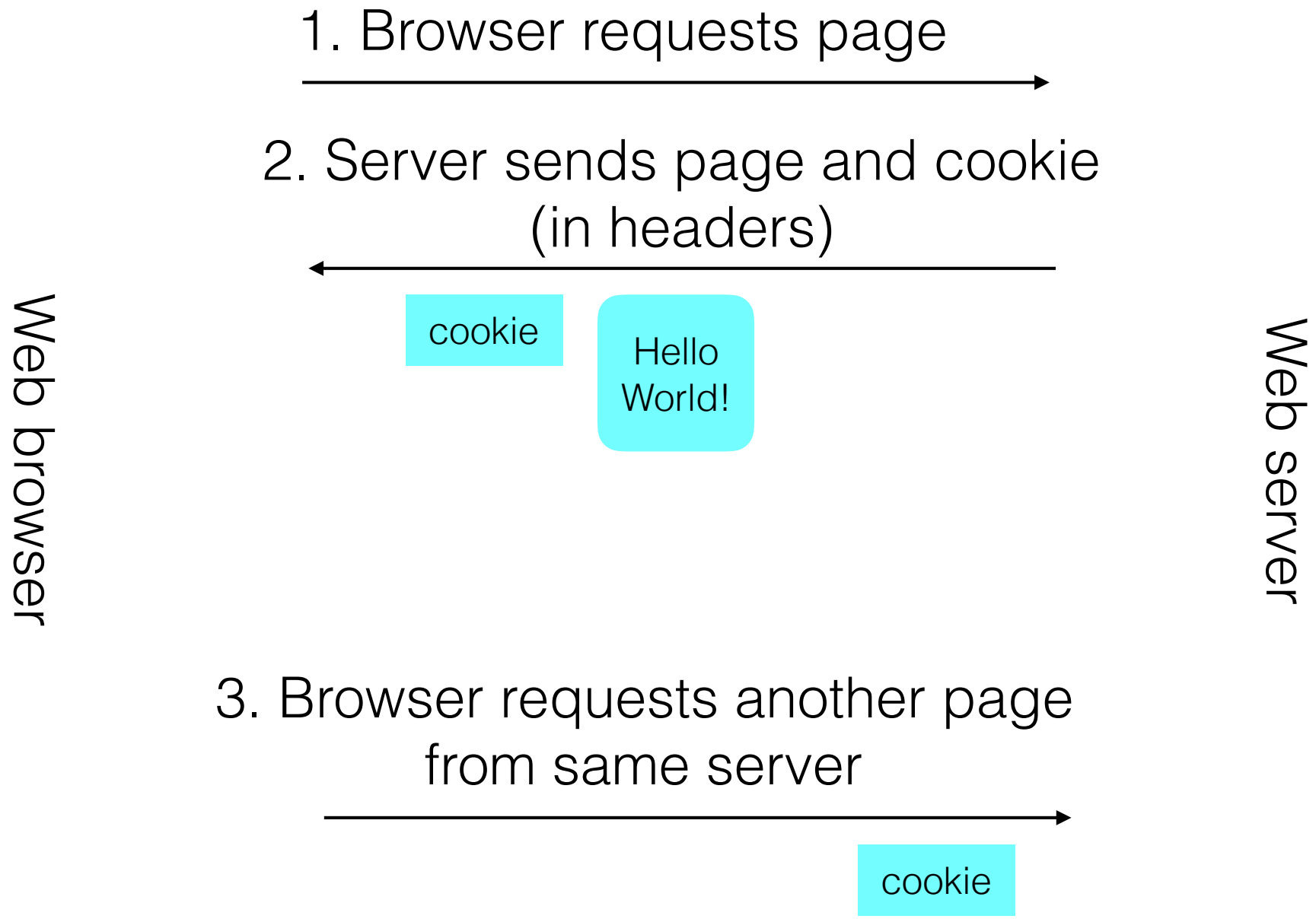
  - Uniquely identify each client for the server.

  - A way of providing custom content for each client.

- You have several options to add and maintain states on top of HTTP:
- Client mechanisms:
  - Cookies
  - Hidden variables
  - URL rewriting
  - HTML5 local storage (for HTTP 1.1)
- Server mechanisms:
  - Sessions

Cookies

- Cookies are a small piece of data sent by a server to a browser, and then sent back by the browser on future page requests.
    - Authentication
    - User tracking
    - Personalization: Maintaining user preferences, shopping carts, etc.


1. Browser requests page
2. Server sends page and cookie (in headers)
cookie
Hello World!
Web browser
Web server
3. Browser requests another page from same server
cookie

Basic idea

- Web server add Set-Cookie: to HTTP response header

```
Set-Cookie: cookie_name1=cookie_value1
Set-Cookie: cookie_name2=cookie_value2; expires=Sun, 16 Jul
2016 06:23:41 GMT
```

  - Each cookie is just a name-value pair.

- Future requests from browser to same server should include the Cookie:header

```
Cookie: cookie_name1=cookie_value1; cookie_name2=cookie_value2
```

Browser side

- You can use document.cookie to manually set/get cookie data

```
document.cookie = "username=smith;
```

or

```
Cookies.set("username", "smith"); ...
alert(Cookies.get("username")); // smith
```

- Keep track of state on the client (e.g. which checkboxes selected)

Cookies

- Mostly only visible to the application (not others on the same browser)
- **`HttpOnly`**: don't allow JavaScript to manipulate cookies, only send back to server
  - I.e. can't use **`Document.cookie`** API
- Caveat: doesn't seem to be definitive best practices yet

Lifetime

- A **session** cookie, the default type, is a temporary cookie that is stored only in the browser's memory, so that when the browser is closed, it’s erased.
  - Can’t be used for long-term tracking.
  - Safer, because only the browser has access.

- Persistent cookies are stored in a a file on the browser's computer and can track long-term information
  - Potentially less secure because users (or the programs that they run) can open cookie files, see/change the cookie values, etc.
  - But…

Problems and Limitations

- Browsers can disable cookies. Users can disable and delete cookies.
- Size limit / expiration policy:

  http://browsercookielimits.squawky.net/
- Security issues: because they're stored in plain text, can be tampered with.
- Privacy issues: identification of the user
- Not handling cookie expiry well (expectations).

Hidden Variables

- An alternative to cookies is hidden variables that store state information in web pages rather than in the browser/client.
  - Cross-browser support.
  - For form-based web apps only.
    - Changing the URL, such as by clicking a hyperlink, loses the state.
    - Current submitted page is the current state, irrespective of what was submitted previously.

```
<input type="hidden" name="secret"
value="Don't tell anyone!!">
```

URLs

- You can also store the state in the URL such that the URL becomes a GET request.
  - Supported by all browsers.
  - Requires all URLs contain all state information (leading to long, unwieldly URLs).
  - Current submitted page represents current state.
  - Independent of what was done previously.

Example

https://www.google.ca/maps/place/
40+St+George+St,+Toronto,+ON+M5S+2E4/
@43.6596449,-79.3989808,17z/data=!3m1!4b1!4m5!
3m4!1s0x882b34c75165c957:0x1447336578b012b6!
8m2!3d43.659641!4d-79.3967921

Sessions

- A server-side option is sessions, which store the current state on the server (i.e., in a file, database). Each request includes a **token** identifying the browser's session.
  - Tokens can be passed via cookies, hidden variables, URL rewriting).
  - At each request, the executing script uses the token to fetch the session state.
  - Beware session hijacking! Avoid by adding a unique value and signature.

Options for storing session state

Web server's memory

- fastest access
- may be too large
- makes load balancing across servers hard

Storage system

- easily shared
- may be overkilll (don't need the reliability)
- may be too much load for the storage system

Specialized storage system

- support fast fetching of small, short-lived data
- Example: memcached, redis - in memory key-value stores

Cookie replacement: Web Storage

- sessionStorage: Per origin storage when page is open
- localStorage: Per origin storage with longer lifetime
- Standard key-value interface
- Limited space (~10MB) and similar reliability issues to cookies.

local storage

`window.localStorage` - stores data with no expiration date

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML
= localStorage.getItem("lastname");
```

- http://html5demos.com/storage

session storage

- Stores data for one session (data is lost when the browser tab is closed)

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount =
            Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML =
      "You have clicked the button " +
       sessionStorage.clickcount +
      " time(s) in this session.";
```