

# Cookies, sessions and third-party authentication

**Claudia Hauff**

TI1506: Web and Database Technology

[ti1506-ewi@tudelft.nl](mailto:ti1506-ewi@tudelft.nl)

# Learning objectives

- **Decide** for a given usage scenario whether cookies or sessions are suitable
- **Explain and implement** cookie usage
- **Explain and implement** session usage
- **Implement** third-party authentication

# Introduction to cookies and sessions

# Recall: HTTP

- HTTP is a **stateless** protocol
- Every HTTP request contains **all information** needed to serve a response
- The server is not required to keep track of the requests issued
- Advantage: simplifies the server architecture
- Disadvantage: clients have to resend **the same information** in every request

We do a lot of things requiring a known state ...

# We do a lot of things requiring a known state ...

- **bol.com**  keeps your *Winkelwagentje* full, even when you leave the website

# We do a lot of things requiring a known state ...

- **bol.com**  keeps your *Winkelwagentje* full, even when you leave the website
- **StatCounter** (tracking users' visits) can exclude a particular visitor from being tracked

# We do a lot of things requiring a known state ...

- **bol.com**  keeps your *Winkelwagentje* full, even when you leave the website
- **StatCounter** (tracking users' visits) can exclude a particular visitor from being tracked
- JavaScript games can keep track of the game's status when you re-visit the game (website)

# We do a lot of things requiring a known state ...

- **bol.com**  keeps your *Winkelwagentje* full, even when you leave the website
- **StatCounter** (tracking users' visits) can exclude a particular visitor from being tracked
- JavaScript games can keep track of the game's status when you re-visit the game (website)
- Websites can tell you how many times you have visited

0:29



How big can a cookie be?

0:29



How big can a cookie be?

0:29



Can users change their cookies?

0:29



Can users change their cookies?

0:29



Can cookies execute programs?

0:29



Can cookies execute programs?

# Cookies cannot ...

- **Execute** programs
- Access information from a **user's hard drive**
- Generate spam
- Be **read by arbitrary parties**
  - Only the server setting the cookie can access it
  - But: beware of **third-party cookies**

# Cookies

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

# Cookies

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

**Cookie:** a short amount of text **(key/value)** sent by the server and **stored by the client** for some amount of time.

# Cookies

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

**Cookie:** a short amount of text (**key/value**) **sent by the server** and **stored by the client** for some amount of time.

Minimum client storage requirements (RFC6265 from 2011)

- Store at least 4096 bytes per cookie
- Store at least 50 cookies per domain
- Store at least 3000 cookies total.

# Cookies

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

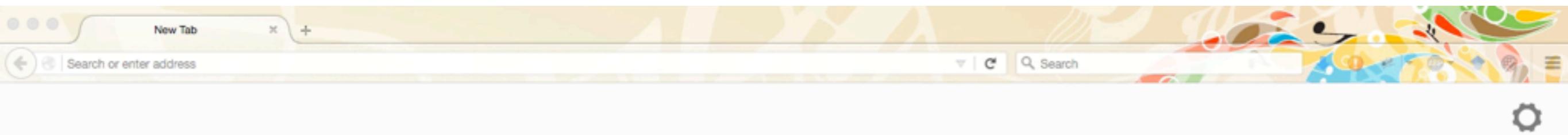
**Cookie:** a short amount of text (**key/value**) **sent by the server** and **stored by the client** for some amount of time.

Minimum client storage requirements (RFC6265 from 2011)

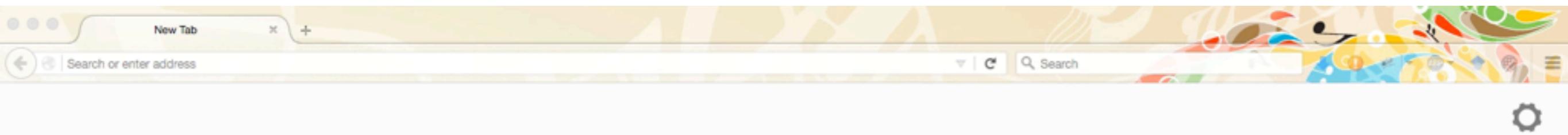
- Store at least 4096 bytes per cookie
- Store at least 50 cookies per domain
- Store at least 3000 cookies total.

“Servers SHOULD use **as few** and **as small** cookies as possible to avoid reaching these implementation limits and minimise network bandwidth”

# Where can I find the cookies?



# Where can I find the cookies?



# Why am I seeing the same cookies everywhere?

Cookie Name	Default Expiration Time	Description
__utma	2 years from set/update	Used to distinguish users and sessions. The cookie is created when the javascript library executes and no existing __utma cookies exists. The cookie is updated every time data is sent to Google Analytics.
__utmt	10 minutes	Used to throttle request rate.
__utmb	30 mins from set/update	Used to determine new sessions/visits. The cookie is created when the javascript library executes and no existing __utmb cookies exists. The cookie is updated every time data is sent to Google Analytics.
__utmc	End of browser session	Not used in ga.js. Set for interoperability with urchin.js. Historically, this cookie operated in conjunction with the __utmb cookie to determine whether the user was in a new session/visit.
__utmz	6 months from set/update	Stores the traffic source or campaign that explains how the user reached your site. The cookie is created when the javascript library executes and is updated every time data is sent to Google Analytics.
__utmv	2 years from set/update	Used to store visitor-level custom variable data. This cookie is created when a developer uses the <code>_setCustomVar</code> method with a visitor level custom variable. This cookie was also used for the deprecated <code>_setVar</code> method. The cookie is updated every time data is sent to Google Analytics.

# Cookie & session basics

A very **old** piece  
of Web technology!  
Developed in **1994**.

# Cookie & session basics

A very **old** piece  
of Web technology!  
Developed in **1994**.

- Cookies are **visible** to the users (who make the effort)
  - By default, stored in the clear

# Cookie & session basics

A very **old** piece  
of Web technology!  
Developed in **1994**.

- Cookies are **visible** to the users (who make the effort)
  - By default, stored in the clear
- Clients (users, i.e. you!) can **delete/disallow** cookies
- Cookies can be **altered by the client**
  - Opens up a line of attack: **servers** should not send sensitive information in simple cookies

# Cookie & session basics

A very **old** piece  
of Web technology!  
Developed in **1994**.

- Cookies are **visible** to the users (who make the effort)
  - By default, stored in the clear
- Clients (users, i.e. you!) can **delete/disallow** cookies
- Cookies can be **altered by the client**
  - Opens up a line of attack: **servers** should not send sensitive information in simple cookies
- **Sessions** are preferable to cookies
  - Sessions themselves make use of cookies
  - Cookie usually contains a single value (session ID), the rest is stored on the server

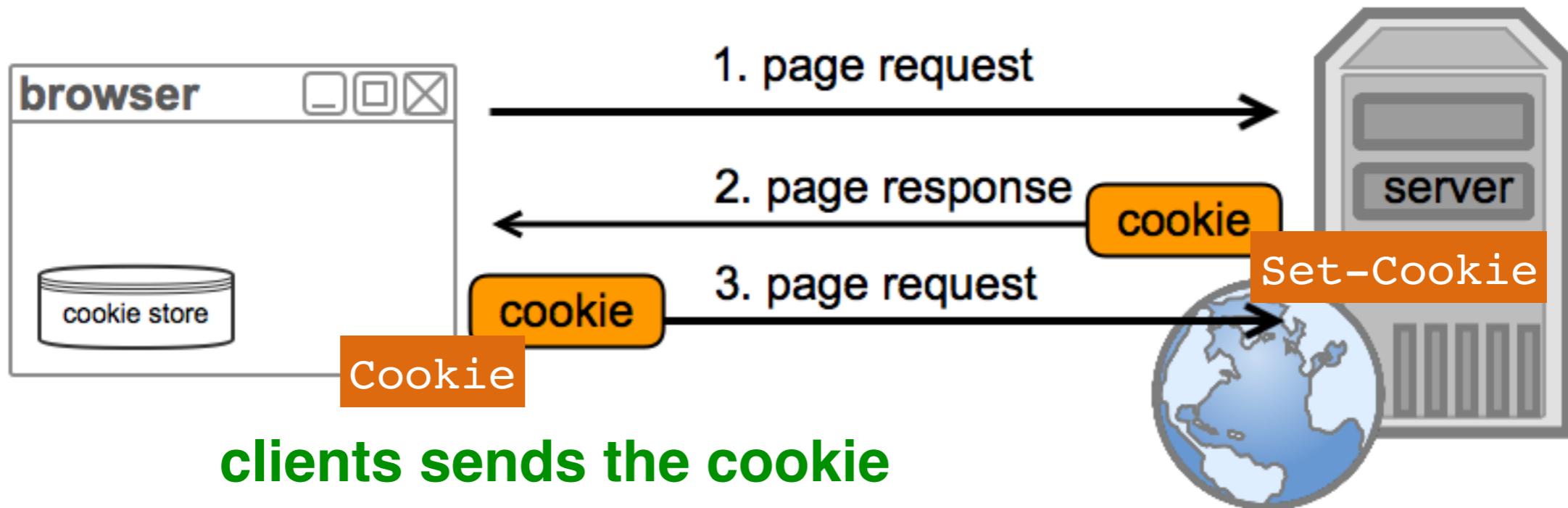
# A word of warning: RFC6265

“This document defines the HTTP **Cookie** and **Set-Cookie** header fields. These header fields can be used by HTTP servers **to store state** (called cookies) at HTTP user agents, letting the servers maintain a **stateful session** over the mostly stateless HTTP protocol.

**Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet. ”**

# Cookie basics

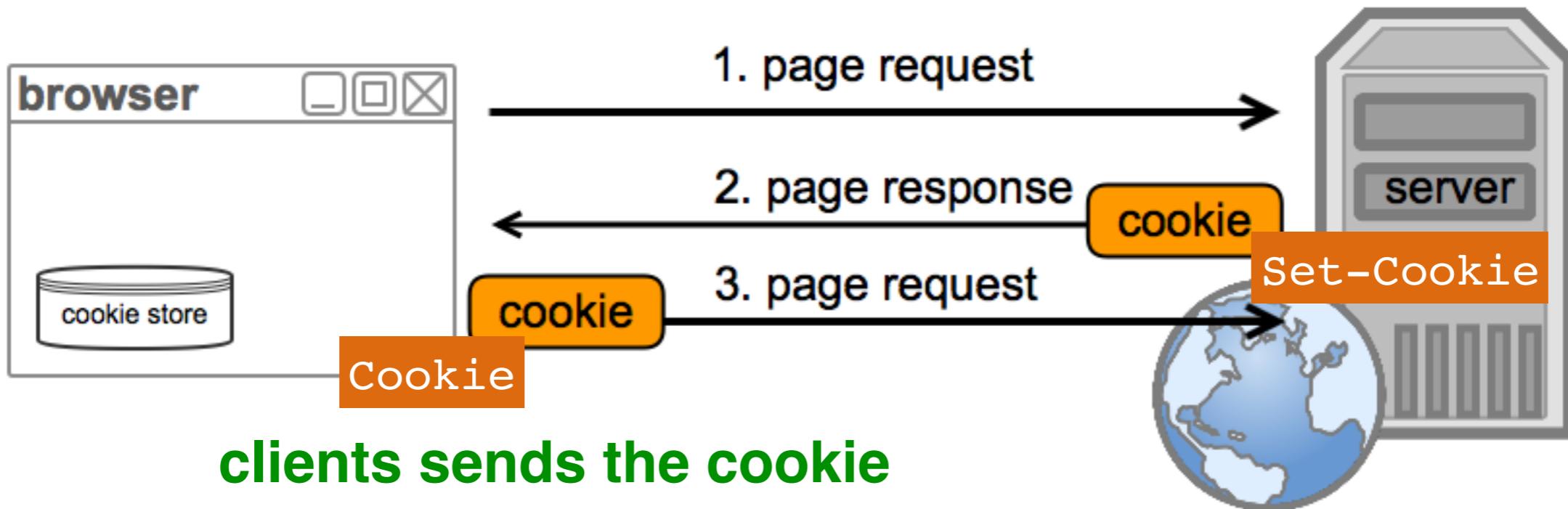
**server sends a cookie once;  
resends when key/value changes**



**clients sends the cookie  
back in every request**

# Cookie basics

server sends a cookie once;  
resends when key/value changes



clients sends the cookie  
back in every request

- Encoded in the **HTTP header**
- Web frameworks have designated methods to work with cookies
- Cookies are **bound** to a **site domain name**, are only sent back on requests to this specific site

# What can be stored in cookies?

- Cookies are the **server's short term memory**
- Information in a cookie is decided by the server
- **Examples:**
  - History of page views
  - Settings of form elements (can also be fully client-side)
  - Tracking of user's UI preferences
  - ...

# Session vs. persistent cookies

# Session vs. persistent cookies

- **Session** (or transient) cookies:
  - Exist in memory only, are deleted when the browser is closed
  - Cookies are session cookies if **no expiration date** is defined.

Not the **tab** or **window**!

# Session vs. persistent cookies

- **Session** (or transient) cookies:  
  - Exist in memory only, are deleted when the browser is closed
  - Cookies are session cookies if **no expiration date** is defined.
- **Persistent** cookies:  
  - Cookies remain intact after the browser is closed
  - Have a **maximum age**
  - Are **send back to the server** as long as they are **valid**

Not the **tab** or **window**!

# Cookie fields

- **Name=value**
- **Expiration** date (UNIX timestamp) or **max age**
- **Domain** the cookie is associated with; cookies can only be assigned to the **same domain** the server is running on
- **Path** the cookie is applied to (automatic wildcarding):  
  / matches all pages, /todos all pages within todos, etc.
- **Secure** flag
- **httpOnly** flag
- **Signed** flag

# Cookie fields

- **Name=value** the only **required** field
- **Expiration** date (UNIX timestamp) or **max age**
- **Domain** the cookie is associated with; cookies can only be assigned to the **same domain** the server is running on
- **Path** the cookie is applied to (automatic wildcarding):  
  / matches all pages, /todos all pages within todos, etc.
- **Secure** flag
- **httpOnly** flag
- **Signed** flag

# Making cookies more robust

# Making cookies more robust

- **Secure** cookies:
  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)

# Making cookies more robust

- **Secure** cookies:
  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)

Secure setting via **HTTP**: the cookie will **not** be **sent**

# Making cookies more robust

- **Secure** cookies:
  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)
- **HttpOnly** cookies:
  - Cookies are **not accessible** to non-HTTP entities (e.g. **JavaScript**)
  - Minimises the threat of cookie theft
  - Applies to session management cookies, not browser cookies

Secure setting via **HTTP**: the cookie will **not** be **sent**

# Making cookies more robust

- **Secure** cookies:
  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)
- **HttpOnly** cookies:
  - Cookies are **not accessible** to non-HTTP entities (e.g. **JavaScript**)
  - Minimises the threat of cookie theft
  - Applies to session management cookies, not browser cookies
- **Signed** cookies (appended **HMAC[value]**):
  - Ensures that the **value** has not been **tampered** with by the client (offers **no privacy**)

Secure setting via **HTTP**: the cookie will **not** be **sent**

**Hash Message Authentication Code**

# Making cookies more robust

- **Secure** cookies:
  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)
- **HttpOnly** cookies:
  - Cookies are **not accessible** to non-HTTP entities (e.g. **JavaScript**)
  - Minimises the threat of cookie theft
  - Applies to session management cookies, not browser cookies
- **Signed** cookies (appended **HMAC[ value ]**):
  - Ensures that the **value** has not been **tampered** with by the client (offers **no privacy**)

Secure setting via **HTTP**: the cookie will **not** be **sent**

**Hash Message Authentication Code**

# Cookie domain

# Cookie domain

- **Origin: request domain** of the cookie (a cookie is always applicable to its origin server)

GET `http://www.my_site.nl/todos` → `www.my_site.nl`

- Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`

# Cookie domain

- **Origin: request domain** of the cookie (a cookie is always applicable to its origin server)  
GET `http://www.my_site.nl/todos` → `www.my_site.nl`
  - Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`
- **Domain attribute**: a cookie's **Domain** attribute has to cover the origin domain
  - If not set, a cookie is only applicable to its origin domain (a cookie from `www.my_site.nl` is not applicable to `my_site.nl`)
  - If set, a cookie is applicable to the domain listed in the attribute and all its **subdomains**

GET `http://www.my_site.nl/todos`

`Set-Cookie: name=value; Path=/; Domain=my_site.nl`

# Cookie domain

- **Origin:** **request domain** of the cookie (a cookie is always applicable to its origin server)  
GET `http://www.my_site.nl/todos`  `www.my_site.nl`
  - Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`
- **Domain attribute:** a cookie's **Domain** attribute has to cover the origin domain
  - If not set, a cookie is only applicable to its origin domain (a cookie from `www.my_site.nl` is not applicable to `my_site.nl`)
  - If set, a cookie is applicable to the domain listed in the attribute and all its **subdomains**

GET `http://www.my_site.nl/todos`

`Set-Cookie: name=value; Path=/; Domain=my_site.nl`

applicable to `www.my_site.nl todos.my_site.nl`  
`serverA.admin.todos.my_site.nl`

# Cookie domain

- **Origin:** **request domain** of the cookie (a cookie is always applicable to its origin server)  
GET `http://www.my_site.nl/todos` → `www.my_site.nl`
  - Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`
- **Domain attribute:** a cookie's **Domain** attribute has to cover the origin domain
  - If not set, a cookie is only applicable to its origin domain (a cookie from `www.my_site.nl` is not applicable to `my_site.nl`)
  - If set, a cookie is applicable to the domain listed in the attribute and all its **subdomains**

GET `http://www.my_site.nl/todos`

`Set-Cookie: name=value; Path=/; Domain=my_site.nl`

applicable to `www.my_site.nl todos.my_site.nl`  
`serverA.admin.todos.my_site.nl`

**Domain attribute  
cannot be a public  
suffix  
(.com, .nl, ...)**

# Example: (1) send cookies to a client that requests them; (2) list all cookies sent by the client.

The screenshot shows a developer environment with a code editor window titled "cookieTester.js - Example9". The code editor displays the following JavaScript code:

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require("./credentials");
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(3000);
9
10
```

The code uses Express.js and cookie-parser to handle HTTP requests and responses. It sets up a server on port 3000 and applies the cookie-parser middleware to handle cookies sent by clients. The code editor interface includes a toolbar, a sidebar with tabs for VARIABLES, WATCH, CALL STACK, and BREAKPOINTS, and a status bar at the bottom indicating the current line (Ln 10, Col 1), character encoding (UTF-8), line separator (LF), and language (JavaScript).

# Example: (1) send cookies to a client that requests them; (2) list all cookies sent by the client.

The screenshot shows a developer environment with a code editor window titled "cookieTester.js - Example9". The code editor displays the following JavaScript code:

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require("./credentials");
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(3000);
9
10
```

The code uses Express.js and cookie-parser to handle HTTP requests and responses. It sets up a server on port 3000 and applies the cookie-parser middleware to handle cookies sent by clients. The code editor interface includes a toolbar, a sidebar with tabs for VARIABLES, WATCH, CALL STACK, and BREAKPOINTS, and a status bar at the bottom indicating the current line (Ln 10, Col 1), character set (UTF-8), line separator (LF), and language (JavaScript).

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

cookie-parser middleware

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

cookie-parser middleware

creating cookies

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

cookie-parser middleware

creating cookies

signing a cookie

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

cookie-parser middleware

creating cookies

signing a cookie

reading cookies

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11     console.log("Handing out cookies");
12     res.cookie("chocolate", "kruemel");
13     res.cookie("signed_choco", "monster", { signed: true});
14     res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18     console.log("++++ unsigned ++++");
19     console.log(req.cookies);
20     console.log("++++ signed ++++");
21     console.log(req.signedCookies);
22     res.send();
23 });
```

---

# How does `credentials.js` look like?

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");
5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

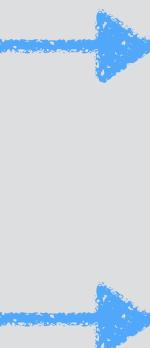
```
npm install cookie-parser
```

# Cookies in express

```
1 var express = require("express");
2 var http = require("http");
3 var credentials = require('./credentials.js');
4 var cookies = require("cookie-parser");

5
6 var app = express();
7 app.use(cookies(credentials.cookieSecret));
8 http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11   console.log("Handing out cookies");
12   res.cookie("chocolate", "kruemel");
13   res.cookie("signed_choco", "monster", { signed: true});
14   res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18   console.log("++++ unsigned ++++");
19   console.log(req.cookies);
20   console.log("++++ signed ++++");
21   console.log(req.signedCookies);
22   res.send();
23 });
```

```
module.exports = {
  cookieSecret: 'abc'
};
```



# Accessing and deleting cookies in express

- **Accessing the value** of a particular key/value pair:

```
var val = req.signedCookies.signed_choco;
```

- **Deleting** a cookie:

```
res.clearCookie('chocolate');
```

# Accessing and deleting cookies in express

- **Accessing the value** of a particular key/value pair:

```
var val = req.signedCookies.signed_choco;
```

cookie key

- **Deleting** a cookie:

```
res.clearCookie('chocolate');
```

# Accessing and deleting cookies in express

- **Accessing the value** of a particular key/value pair:

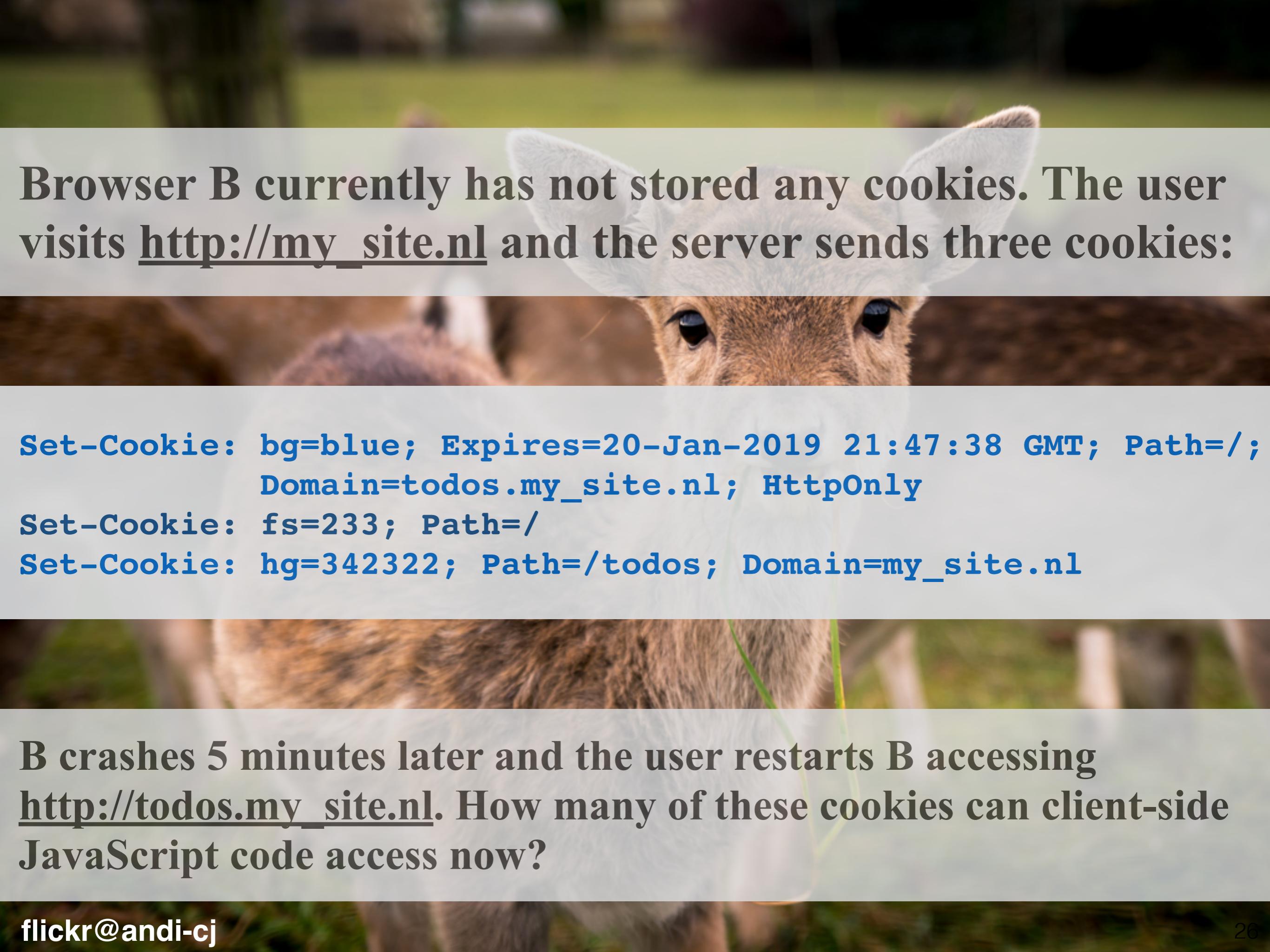
```
var val = req.signedCookies.signed_choco;
```

cookie key

- **Deleting** a cookie:

```
res.clearCookie('chocolate');
```

delete in the **response!**



Browser B currently has not stored any cookies. The user visits [http://my\\_site.nl](http://my_site.nl) and the server sends three cookies:

**Set-Cookie: bg=blue; Expires=20-Jan-2019 21:47:38 GMT; Path=/;**  
**Domain=todos.my\_site.nl; HttpOnly**

**Set-Cookie: fs=233; Path=/**

**Set-Cookie: hg=342322; Path=/todos; Domain=my\_site.nl**

B crashes 5 minutes later and the user restarts B accessing [http://todos.my\\_site.nl](http://todos.my_site.nl). How many of these cookies can client-side JavaScript code access now?

After accessing <https://nl.login.meebo.com> for the first time, the server sent the following cookies to B:

**Set-Cookie: ID1=32sfs32; Domain=login.meebo.com;  
Path=/todos; Expires=Fri, 30 Jan 2019 01:01:01 GMT;  
Secure; HttpOnly**

**Set-Cookie: ID2=532aaaa; Domain=meebo.com; Path=/;  
Expires=Fri, 30 Jan 2019 01:01:01 GMT; HttpOnly**

**Set-Cookie: ID3=ssd33dd; Path=/; Expires=Fri, 30 Jan 2018  
01:01:01 GMT; HttpOnly**

Next, B tries to access <https://www.meebo.com/todos>. How many of the 3 cookies are sent back to the server?

0:29



In how many different ways can cookie information be stored in the browser?

0:29



In how many different ways can cookie information be stored in the browser?

A more pessimistic view  
on cookies

# Evercookie

“evercookie is a javascript API available that produces **extremely persistent cookies** in a browser. Its goal is to **identify a client** even **after they've removed standard cookies** [...] evercookie accomplishes this by storing the cookie data in **several types of storage mechanisms** that are available on the local browser. Additionally, if evercookie has found the user has removed any of the types of cookies in question, it **recreates** them using each mechanism available.”

Source: <http://www.samy.pl/evercookie/>

# Evercookie

## Browser Storage Mechanisms

---

Client browsers must support as many of the following storage mechanisms as possible in order for Evercookie to be effective.

- Standard [HTTP Cookies](#)
- Flash [Local Shared Objects](#)
- Silverlight [Isolated Storage](#)
- CSS [History Knocking](#)
- Storing cookies in [HTTP ETags](#) ([Backend server required](#))
- Storing cookies in [Web cache](#) ([Backend server required](#))
- [HTTP Strict Transport Security \(HSTS\)](#) Pinning (works in Incognito mode)
- [window.name caching](#)
- Internet Explorer [userData storage](#)
- HTML5 [Session Storage](#)
- HTML5 [Local Storage](#)
- HTML5 [Global Storage](#)
- HTML5 [Database Storage via SQLite](#)
- HTML5 Canvas - Cookie values stored in RGB data of auto-generated, force-cached PNG images ([Backend server required](#))
- HTML5 [IndexedDB](#)
- Java [JNLP PersistenceService](#)
- Java exploit [CVE-2013-0422](#) - Attempts to escape the applet sandbox and write cookie data directly to the user's hard drive.

# Evercookie

## Browser Storage Mechanisms

---

Client browsers must support as many of the following storage mechanisms as possible in order for Evercookie to be effective.

- Standard [HTTP Cookies](#)
- Flash [Local Shared Objects](#)
- Silverlight [Isolated Storage](#)
- CSS [History Knocking](#)
- Storing cookies in [HTTP ETags](#) ([Backend server required](#))
- Storing cookies in [Web cache](#) ([Backend server required](#))
- [HTTP Strict Transport Security \(HSTS\)](#) Pinning (works in Incognito mode)
- [window.name](#) caching
- Internet Explorer [userData](#) storage
- [HTML5 Session Storage](#)
- [HTML5 Local Storage](#)
- [HTML5 Global Storage](#)
- [HTML5 Database Storage via SQLite](#)
- HTML5 Canvas - Cookie values stored in RGB data of auto-generated, force-cached PNG images ([Backend server required](#))
- [HTML5 IndexedDB](#)
- Java [JNLP PersistenceService](#)
- Java exploit [CVE-2013-0422](#) - Attempts to escape the applet sandbox and write cookie data directly to the user's hard drive.

# Often though, we are tracked without our knowledge

The screenshot shows the homepage of [www.volkskrant.nl](http://www.volkskrant.nl). At the top, there's a banner for 'ZieZo van Zilveren Kruis' with a price of €69,75 per month. Below the banner, the main navigation menu includes links for BANEN, ABONNEER, DIGITALE KRANT, SERVICE, WINKEL, VOORDEEL, and social media links for Twitter and Facebook. The main content area features a large photo of two men in suits walking through a crowd at night. A blue arrow points from the top right towards a small icon of a person with a speech bubble in the bottom right corner of the page. On the far right, a sidebar lists various tracking companies: BidSwitch, ChartBeat, Crazy Egg, DoubleClick, Ghostery Privacy Notice, Google Adsense, Google Analytics, Google Tag Manager, Platform161, Rubicon, Usabilla, and Weborama.

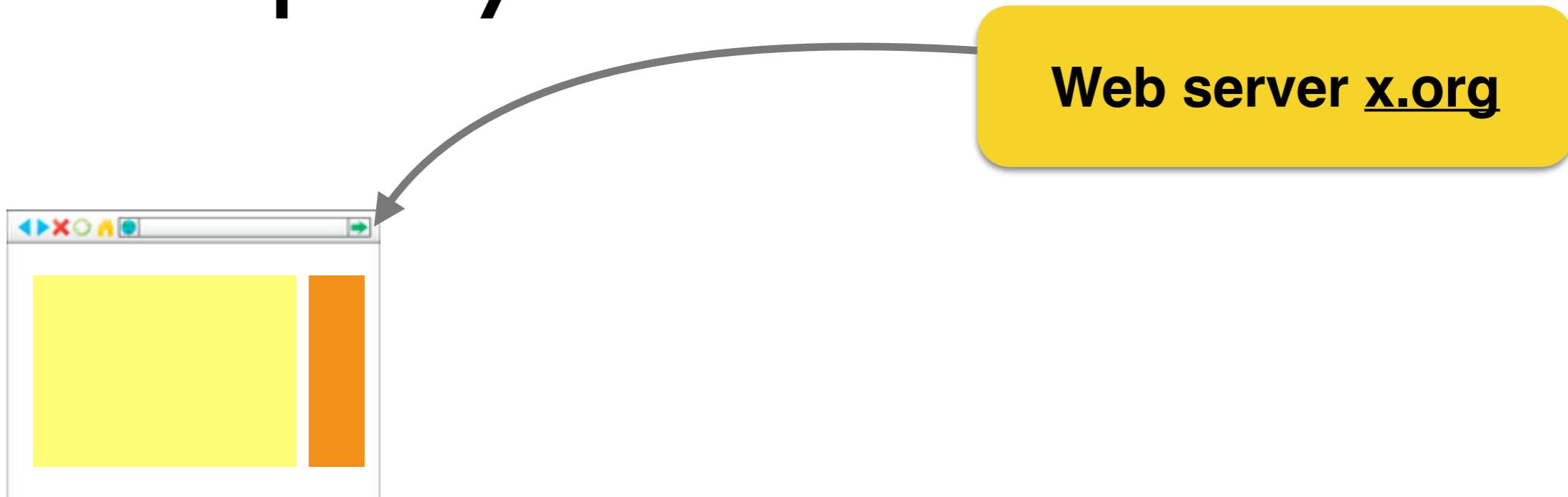


<https://www.ghostery.com/>

# Third-party cookies

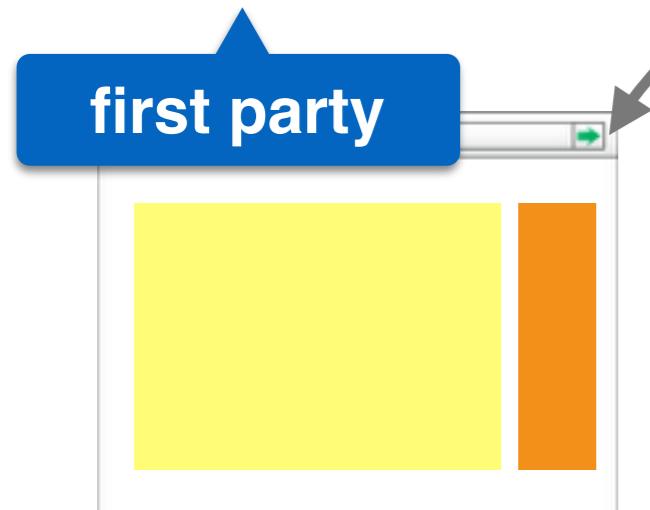


# Third-party cookies



# Third-party cookies

Set-Cookie: x.org



**Web server x.org**

# Third-party cookies

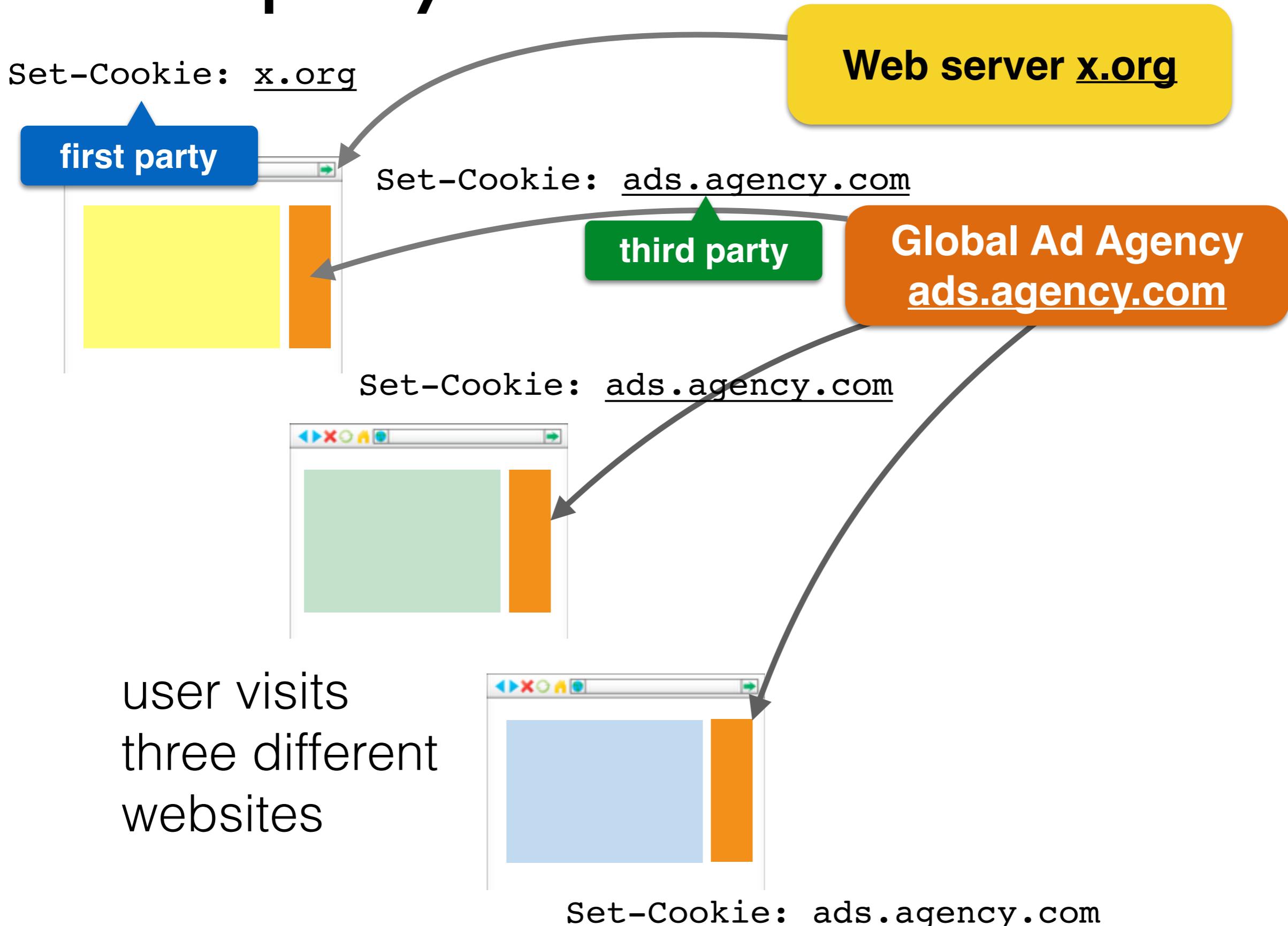
Set-Cookie: x.org



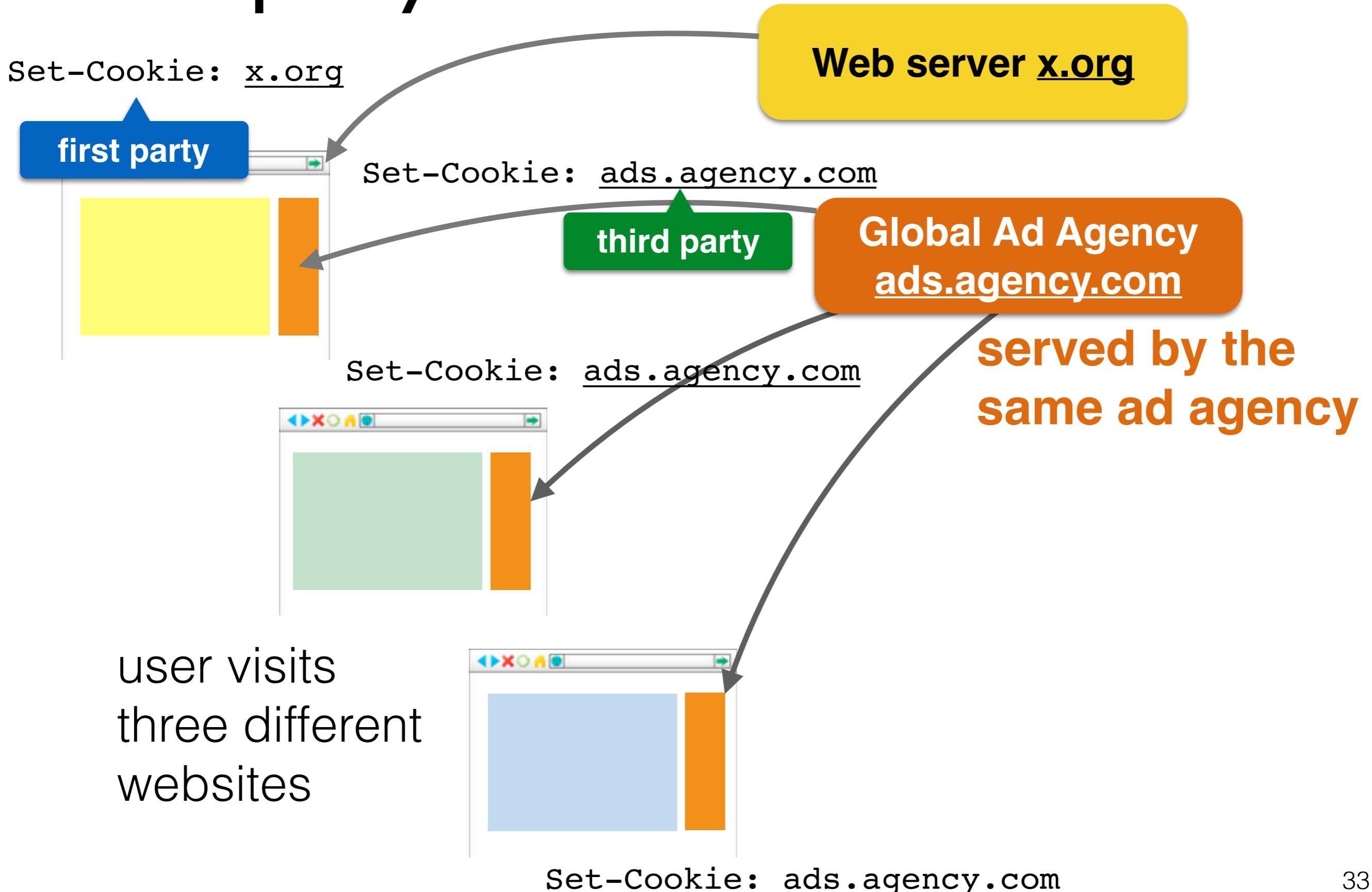
# Third-party cookies



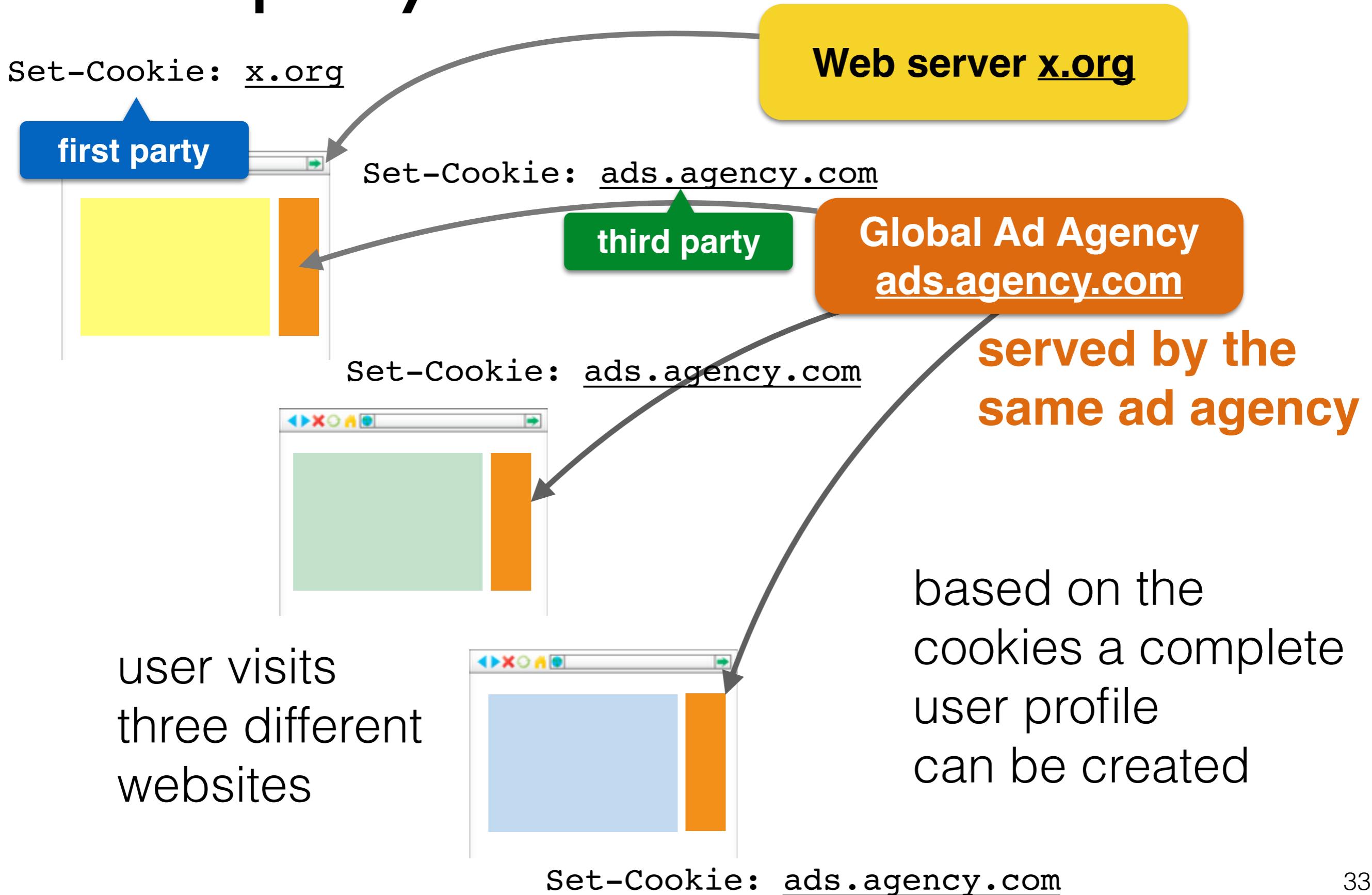
# Third-party cookies



# Third-party cookies



# Third-party cookies



# Client-side cookies

# Cookies in JavaScript

- Not always necessary to receive cookies from a server
- Cookies can be **set in the browser**
- Standard use case: remember form input

# Cookies in JavaScript

- Not always necessary to receive cookies from a server
- Cookies can be **set in the browser**
- Standard use case: remember form input

```
1 //set a cookie
2 document.cookie = "name=value";
3 document.cookie = "name=value; expires=Fri,
                    24-Jan-2017 12:45:00 GMT";
4
5 //delete a cookie
6 document.cookie = "name=value; expires=Fri,
                    24-Jan-1970 12:45:00 GMT";
```

# document.cookie is unlike any other

```
1 //adding three cookies
2 document.cookie = "couponnum=123";
3 document.cookie = "couponval=20%";
4 document.cookie = "expires=60";
5
6 //delete a cookie
7 //document.cookie=null or document.cookie="" has no effect
8 document.cookie = "name=value; expires=Thu,
                    01-Jan-1970 00:45:00 GMT";
```

Add a cookie :

Delete a cookie:

**Modify cookies**

**Show cookies**

# document.cookie is unlike any other

```
1 var toadd = document.getElementById('addCookie')
2             .value;
3 if( toadd.length > 0) {
4     document.cookie = toadd;
5 }
6 var todel = document.getElementById(
7             'deleteCookie').value;
8 if( todel.length > 0) {
9     document.cookie =
10        todel+'; expires=Thu, 01-Jan-1970 00:00:01 GMT';
11 }
```

The screenshot shows a dark-themed user interface for managing cookies. At the top left, there is an input field labeled "Add a cookie : name=value". Below it, another input field is labeled "Delete a cookie: name=value". At the bottom left, there are two buttons: a blue button labeled "Modify cookies" and an orange button labeled "Show cookies". The right side of the interface is a large, empty dark area.

# Reading cookies in JavaScript

- `document.cookie["firstname"]` does **not work**
- **String** returned by `document.cookie` needs to be parsed  
`couponnum=123; couponval=20%; expires=60`

```
1 var cookiesArray = document.cookie.split("; ");
2 var cookies=[ ];
3
4 for(var i=0; i < cookiesArray.length; i++) {
5   var cookie = cookiesArray[i].split("=");
6   cookies[cookie[0]]=cookie[1];
7 }
```

- **Alternative:** jQuery cookie plugin (118 lines of code)

# Sessions

# Establishing a session

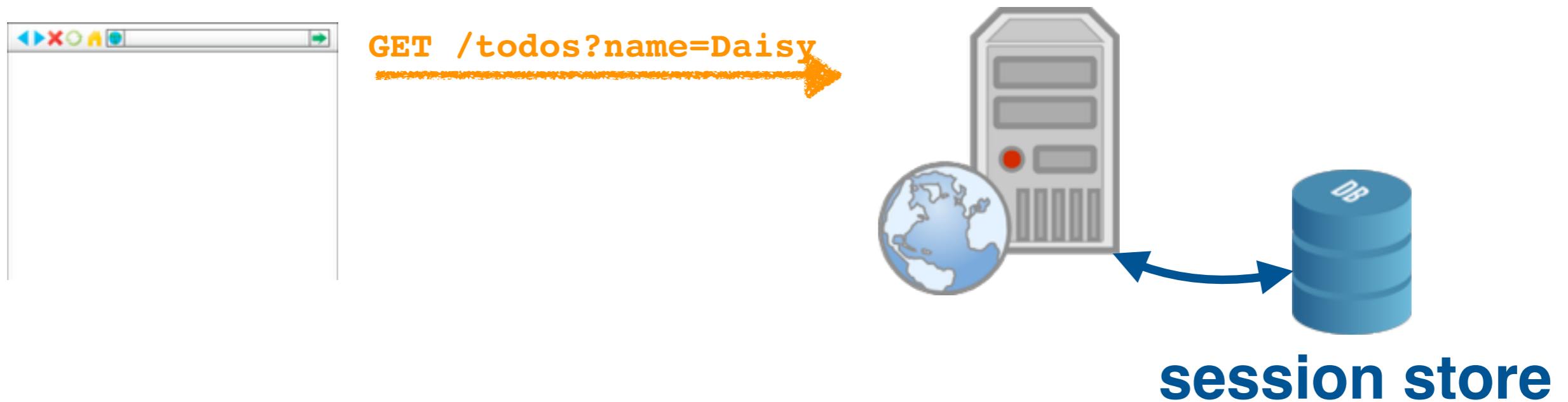
- **Common scenario:** short period of time that users interact with a web site (a session)
- **Goals:**
  - Track the user without relying (too much) on unreliable cookies
  - Allow larger amounts of data to be stored
- **Problem:** without cookies the server cannot tell clients apart
- **Solution:** **hybrid approach** between cookies and server-side saved data

# Sessions in one slide



- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side** in memory or in a database

# Sessions in one slide



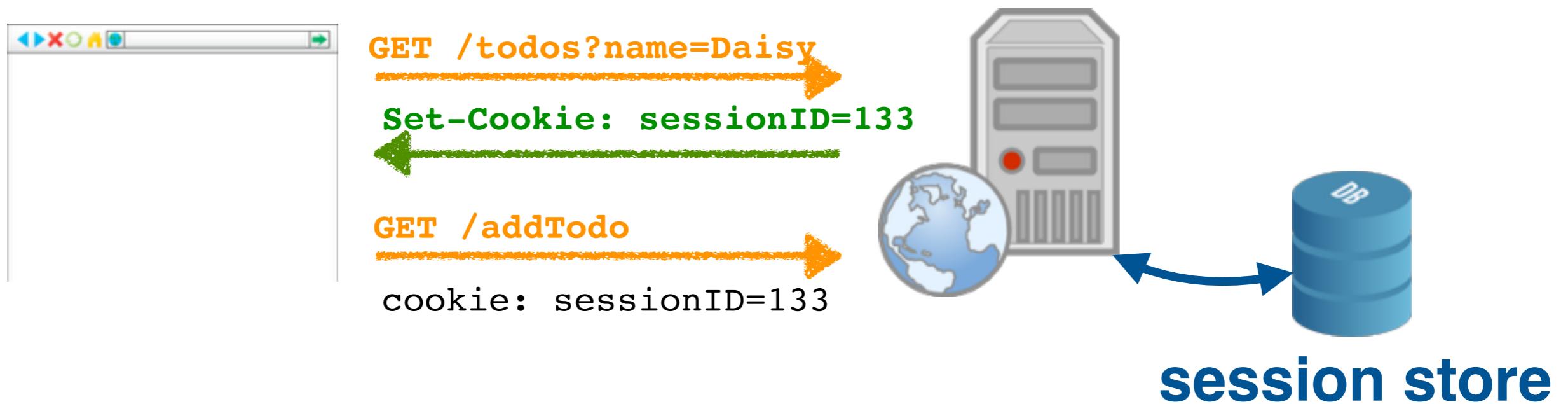
- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side** in memory or in a database

# Sessions in one slide



- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side** in memory or in a database

# Sessions in one slide



- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side** in memory or in a database

# Establishing a session

# Establishing a session

1. **Client** requests a first page from the server

# Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client

# Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID

# Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID
4. From now on, the client sends **page requests together with the cookie**

# Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID
4. From now on, the client sends **page requests together with the cookie**
5. Server can use the **ID to personalise** the response

# Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID
4. From now on, the client sends **page requests together with the cookie**
5. Server can use the **ID to personalise** the response
6. A **session ends** when no further requests with that session ID come in (timeout)

# Sessions in express with memory stores

# Sessions in express with memory stores

- Easy to set up in express

# Sessions in express with memory stores

- Easy to set up in express
- Same drawback as any in-memory storage: not **persistent** across machine failure

# Sessions in express with memory stores

- Easy to set up in express
- Same drawback as any in-memory storage: not **persistent** across machine failure
- A middleware component is helping out:  
**express-session**: <https://github.com/expressjs/session>

# Sessions in express with memory stores

- Easy to set up in express
- Same drawback as any in-memory storage: not **persistent** across machine failure
- A middleware component is helping out:  
**express-session**: <https://github.com/expressjs/session>
- Most common use case: **authentication**

# Sessions in express with memory stores

- Easy to set up in express
- Same drawback as any in-memory storage: not **persistent** across machine failure
- A middleware component is helping out:  
**express-session**: <https://github.com/expressjs/session>
- Most common use case: **authentication**

**Authentication**: verifying a user's identity

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");  
var http = require("http");  
var credentials = require("./credentials");  
var cookies = require("cookie-parser");  
var sessions = require("express-session");  
  
var app = express();  
app.use(cookies(credentials.cookieSecret));  
app.use(sessions(credentials.cookieSecret));  
http.createServer(app).listen(3001);  
  
app.get("/countMe", function (req, res) {  
    var session = req.session;  
    if (session.views) {  
        session.views++;  
        res.send("You have been here " +  
            session.views + " times (last visit: " + session.lastVisit + ")");  
        session.lastVisit = new Date().toLocaleDateString();  
    }  
    else {  
        session.views = 1;  
        session.lastVisit = new Date().toLocaleDateString();  
        res.send("This is your first visit!");  
    }  
});
```

```

var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
  var session = req.session;
  if (session.views) {
    session.views++;
    res.send("You have been here " +
            session.views + " times (last visit: " + session.lastVisit + ")");
    session.lastVisit = new Date().toLocaleDateString();
  }
  else {
    session.views = 1;
    session.lastVisit = new Date().toLocaleDateString();
    res.send("This is your first visit!");
  }
});

```

# How many middleware components have we set up?

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
  var session = req.session;
  if (session.views) {
    session.views++;
    res.send("You have been here " +
      session.views + " times (last visit: " + session.lastVisit + ")");
    session.lastVisit = new Date().toLocaleDateString();
  }
  else {
    session.views = 1;
    session.lastVisit = new Date().toLocaleDateString();
    res.send("This is your first visit!");
  }
});
```

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
  var session = req.session;
  if (session.views) {
    session.views++;
    res.send("You have been here " +
      session.views + " times (last visit: " + session.lastVisit + ")");
    session.lastVisit = new Date().toLocaleDateString();
  }
  else {
    session.views = 1;
    session.lastVisit = new Date().toLocaleDateString();
    res.send("This is your first visit!");
  }
});
```

cookie & session setup

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");  
var http = require("http");  
var credentials = require("./credentials");  
var cookies = require("cookie-parser");  
var sessions = require("express-session");  
  
var app = express();  
app.use(cookies(credentials.cookieSecret));  
app.use(sessions(credentials.cookieSecret));  
http.createServer(app).listen(3001);
```

cookie & session  
setup

```
app.get("/countMe", function (req, res) {  
    var session = req.session;  
    if (session.views) {  
        session.views++;  
        res.send("You have been here " +  
            session.views + " times (last visit: " + session.lastVisit + ")");  
        session.lastVisit = new Date().toLocaleDateString();  
    }  
    else {  
        session.views = 1;  
        session.lastVisit = new Date().toLocaleDateString();  
        res.send("This is your first visit!");  
    }  
});
```

client's session object

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");  
var http = require("http");  
var credentials = require("./credentials");  
var cookies = require("cookie-parser");  
var sessions = require("express-session");  
  
var app = express();  
app.use(cookies(credentials.cookieSecret));  
app.use(sessions(credentials.cookieSecret));  
http.createServer(app).listen(3001);
```

cookie & session setup

```
app.get("/countMe", function (req, res) {  
    var session = req.session;  
    if (session.views) {  
        session.views++;  
        res.send("You have been here " +  
            session.views + " times (last visit: " + session.lastVisit + ")");  
        session.lastVisit = new Date().toLocaleDateString();  
    }  
    else {  
        session.views = 1;  
        session.lastVisit = new Date().toLocaleDateString();  
        res.send("This is your first visit!");  
    }  
});
```

client's session object

session exists!

# Sessions in express with memory stores

```
npm install cookie-parser  
npm install express-session
```

```
var express = require("express");  
var http = require("http");  
var credentials = require("./credentials");  
var cookies = require("cookie-parser");  
var sessions = require("express-session");  
  
var app = express();  
app.use(cookies(credentials.cookieSecret));  
app.use(sessions(credentials.cookieSecret));  
http.createServer(app).listen(3001);
```

cookie & session setup

```
app.get("/countMe", function (req, res) {  
  var session = req.session;  
  if (session.views) {  
    session.views++;  
    res.send("You have been here " +  
            session.views + " times (last visit: " + session.lastVisit + ")");  
    session.lastVisit = new Date().toLocaleDateString();  
  }  
  else {  
    session.views = 1;  
    session.lastVisit = new Date().toLocaleDateString();  
    res.send("This is your first visit!");  
  }  
});
```

client's session object

session exists!

session does not yet exist

0:29

A photograph of two deer in a field of tall, golden-yellow grass. One deer is in the foreground, facing right, while another is partially visible behind it. The background is a soft-focus view of more deer and the landscape.

How do you protect the session cookie in your Web application?

0:29

A photograph of two deer in a field of tall grass. One deer is in the foreground, facing left, and the other is slightly behind it, also facing left. The background is a dense field of tall, golden-yellow grass.

How do you protect the session cookie in your Web application?

# Third-party authentication

# Twitter

# Google

# Facebook

# Yahoo

# LinkedIn

...

The image shows the LinkedIn sign-up form. It features four input fields for First name, Last name, Email, and Password. Below these is a terms and conditions section with a link to User Agreement, Privacy Policy, and Cookie Policy. A large blue "Join now" button is at the bottom. Further down, there's a "Continue with Facebook" button with a red arrow pointing to it from the left.

First name  
Last name  
Email  
Password (6 or more characters)

By clicking Join now, you agree to LinkedIn's User Agreement, Privacy Policy, and Cookie Policy

**Join now**

— or —

Continue with Facebook

# Overview

- **Weakest link** in an authenticated application is the **user's password**
- **Application-based decision**
  - Does the application need authentication?
  - Are cookies/sessions enough?
  - If authentication is needed, should third-party authentication be used? (low cognitive burden for the user)

# Third-party authentication

- Authenticating users through popular social Web services (Twitter, Facebook, Google, LinkedIn, etc.)
- **Easy** to develop for popular platforms
- **Trusted** social Web platforms **provide authentication**, no need to store passwords or employ particular security measures
- **However**: some users may not use social Web platforms or do not like to hand over their data

# OAuth 2.0 Authorization Framework

“The OAuth 2.0 authorization framework enables a **third-party application** to obtain **limited access** to an HTTP service, either **on behalf** of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.”

# OAuth 2.0 roles

# OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource

# OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting the protected resources, capable of accepting and responding to protected resource requests using **access tokens**.

# OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting the protected resources, capable of accepting and responding to protected resource requests using **access tokens**.

a string denoting a specific scope, lifetime and other access attributes

# OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting the protected resources, capable of accepting and responding to protected resource requests using **access tokens**.  

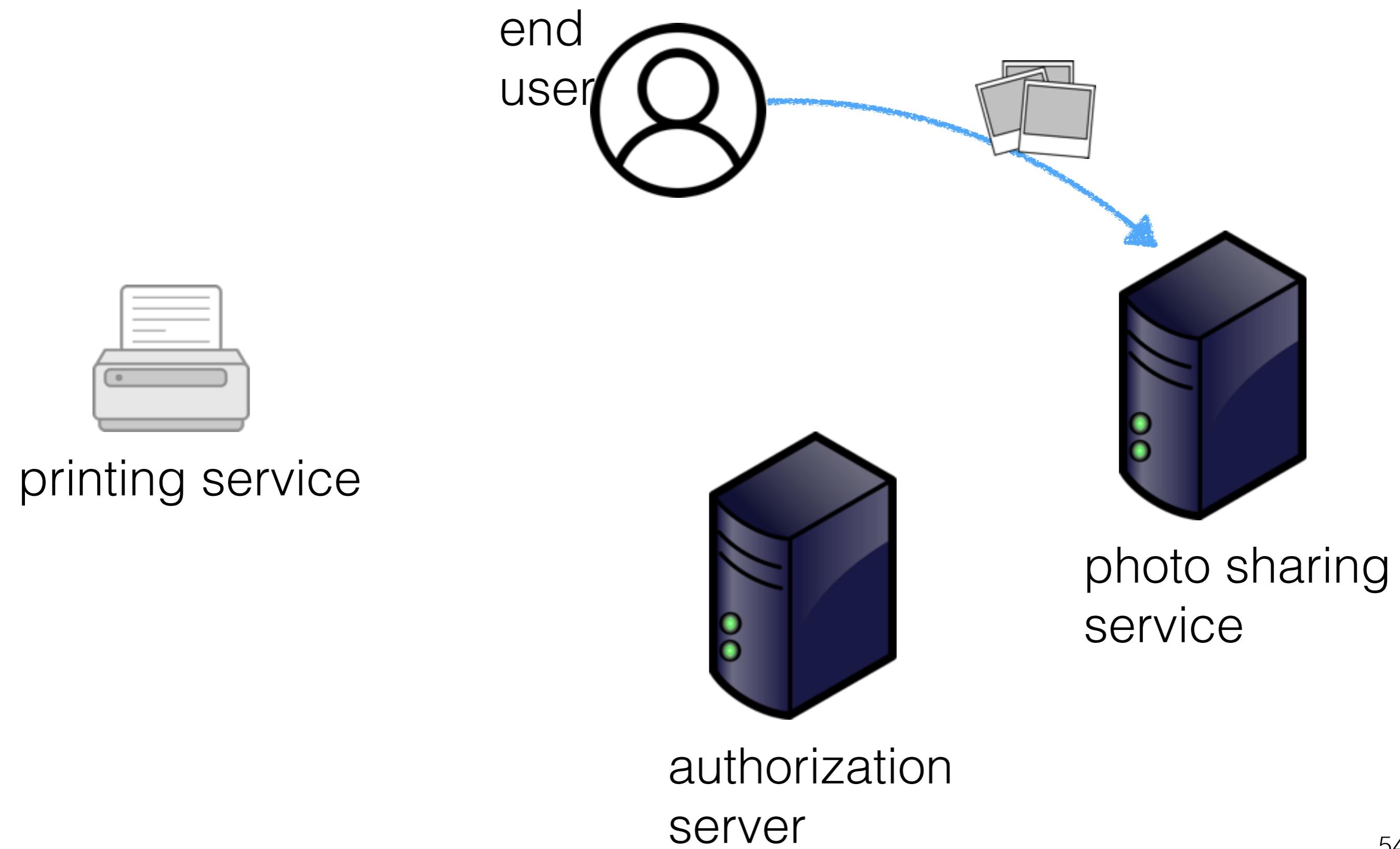
a string denoting a specific scope, lifetime and other access attributes
- **Client**: an application making protected resource requests on behalf of the resource owner **and with its authorisation**

# OAuth 2.0 roles

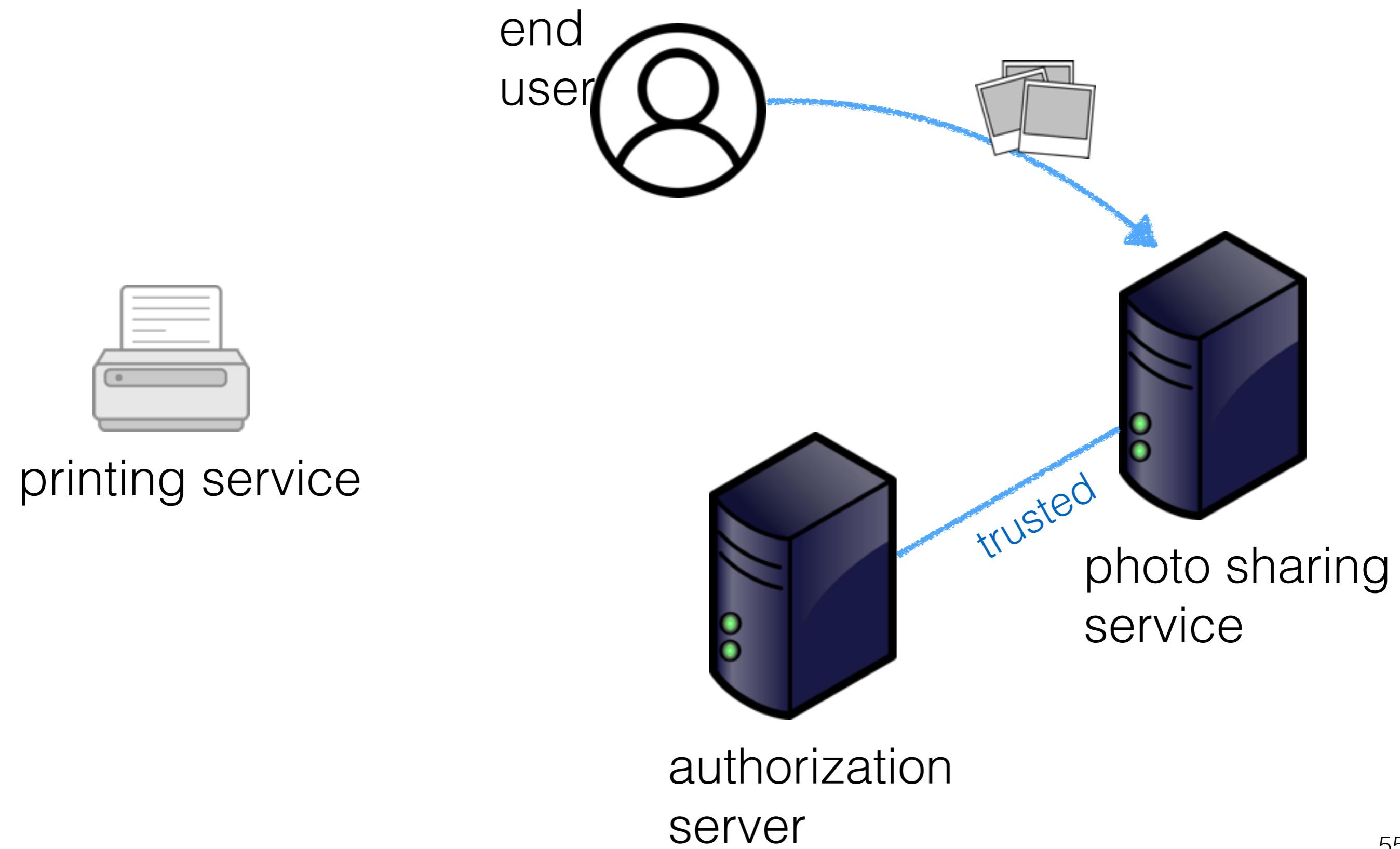
- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting the protected resources, capable of accepting and responding to protected resource requests using **access tokens**.  

a string denoting a specific scope, lifetime and other access attributes
- **Client**: an application making protected resource requests on behalf of the resource owner **and with its authorisation**
- **Authorization server**: server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

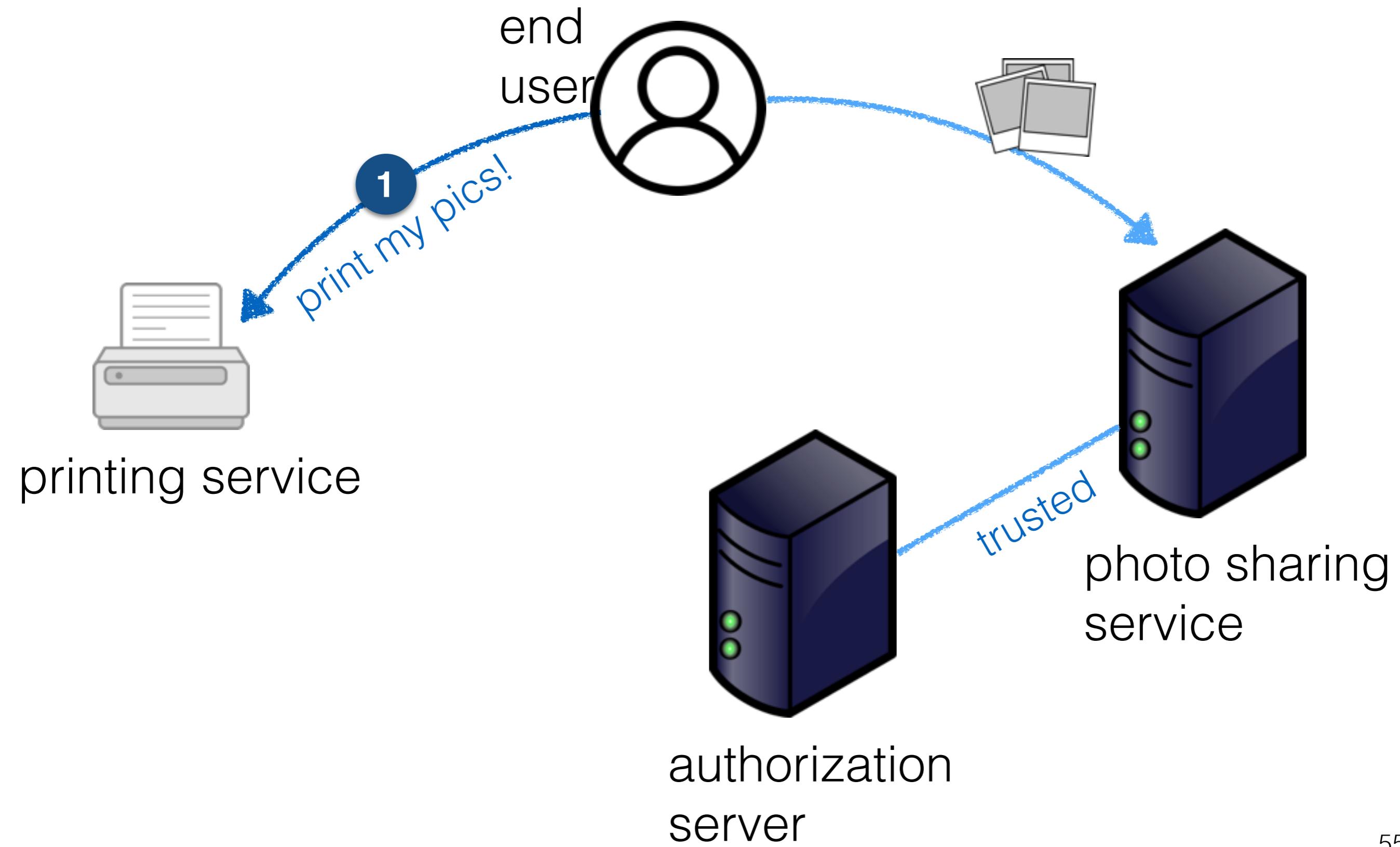
# OAuth 2.0 roles exemplified



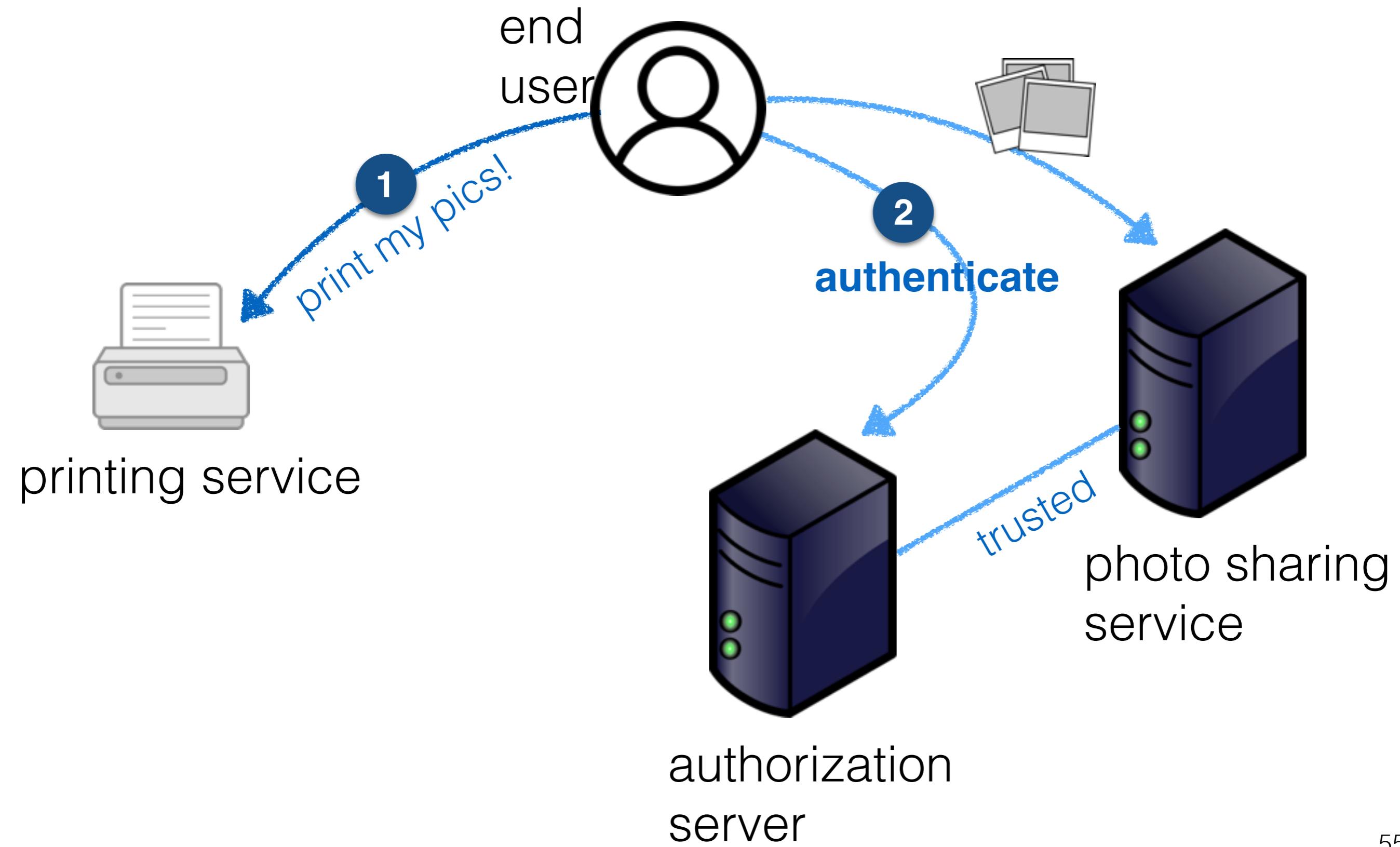
# OAuth 2.0 roles exemplified



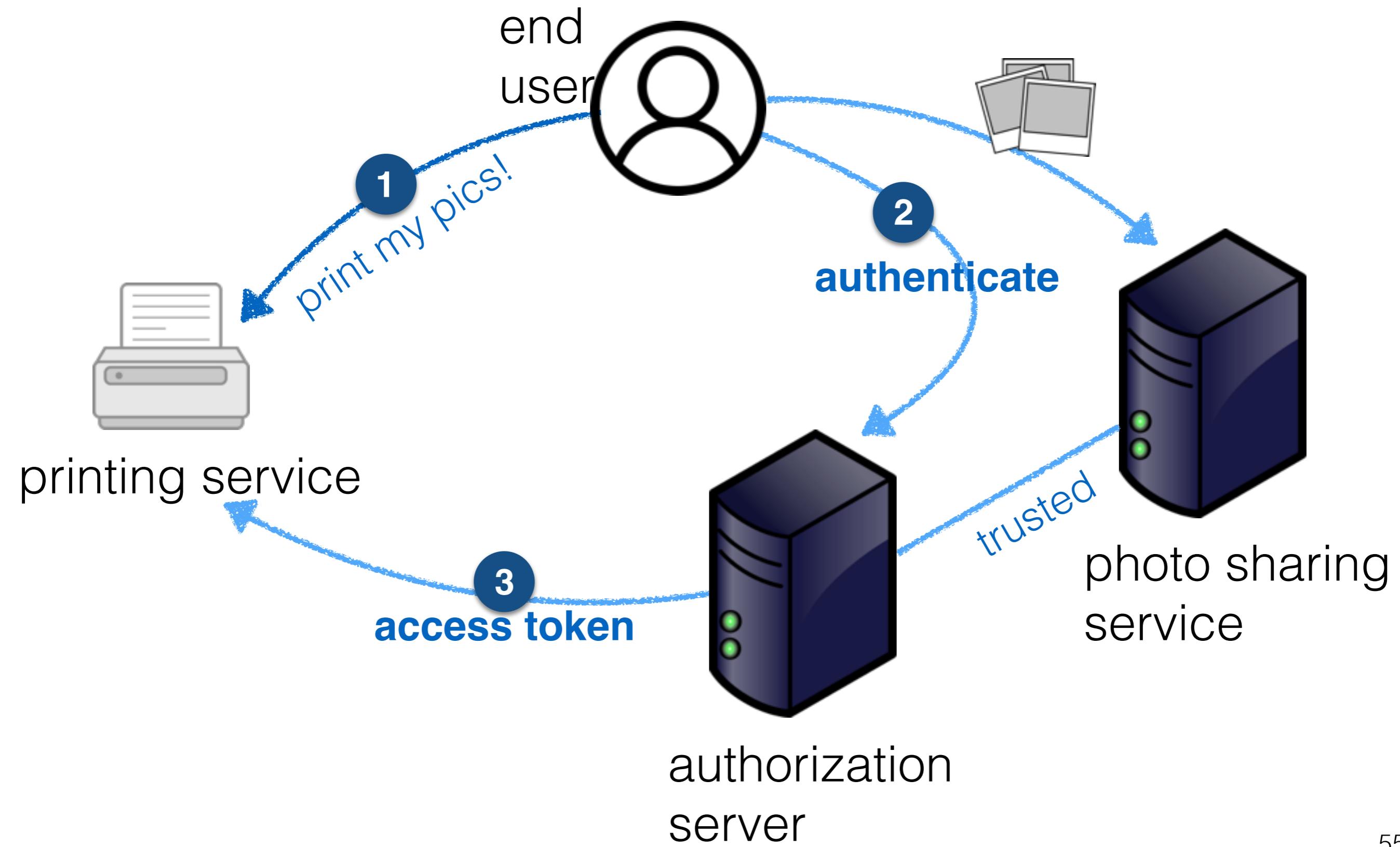
# OAuth 2.0 roles exemplified



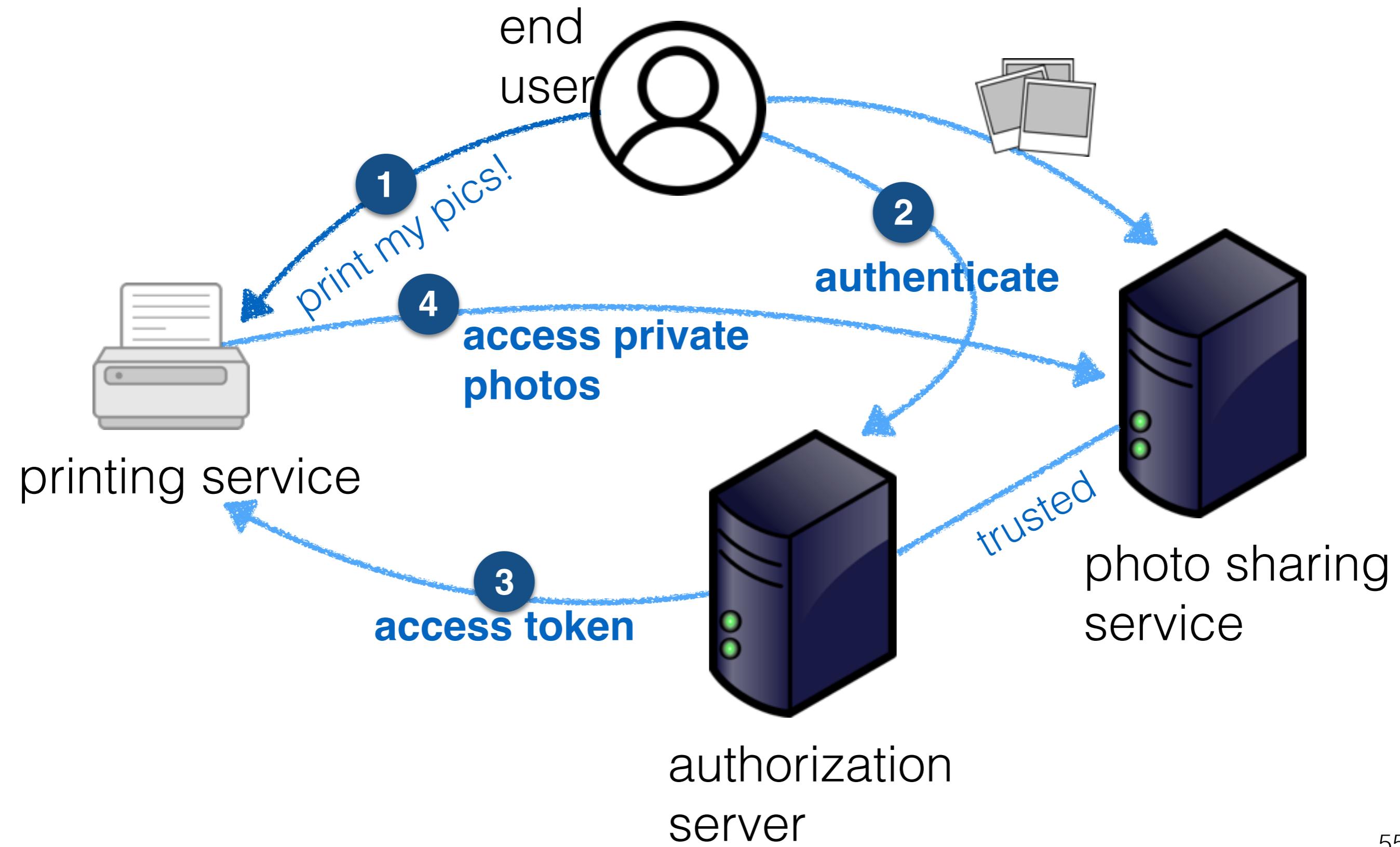
# OAuth 2.0 roles exemplified



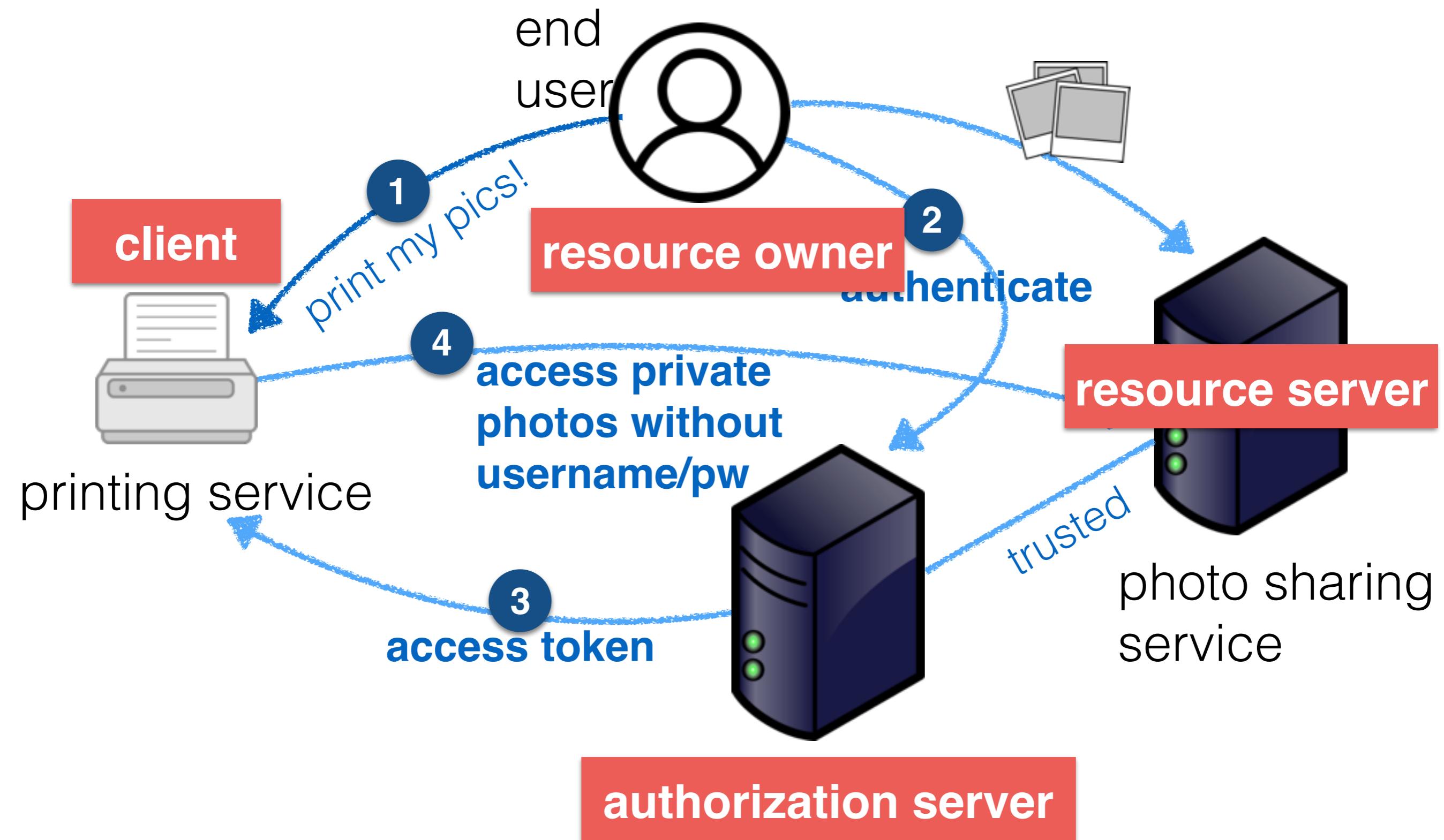
# OAuth 2.0 roles exemplified



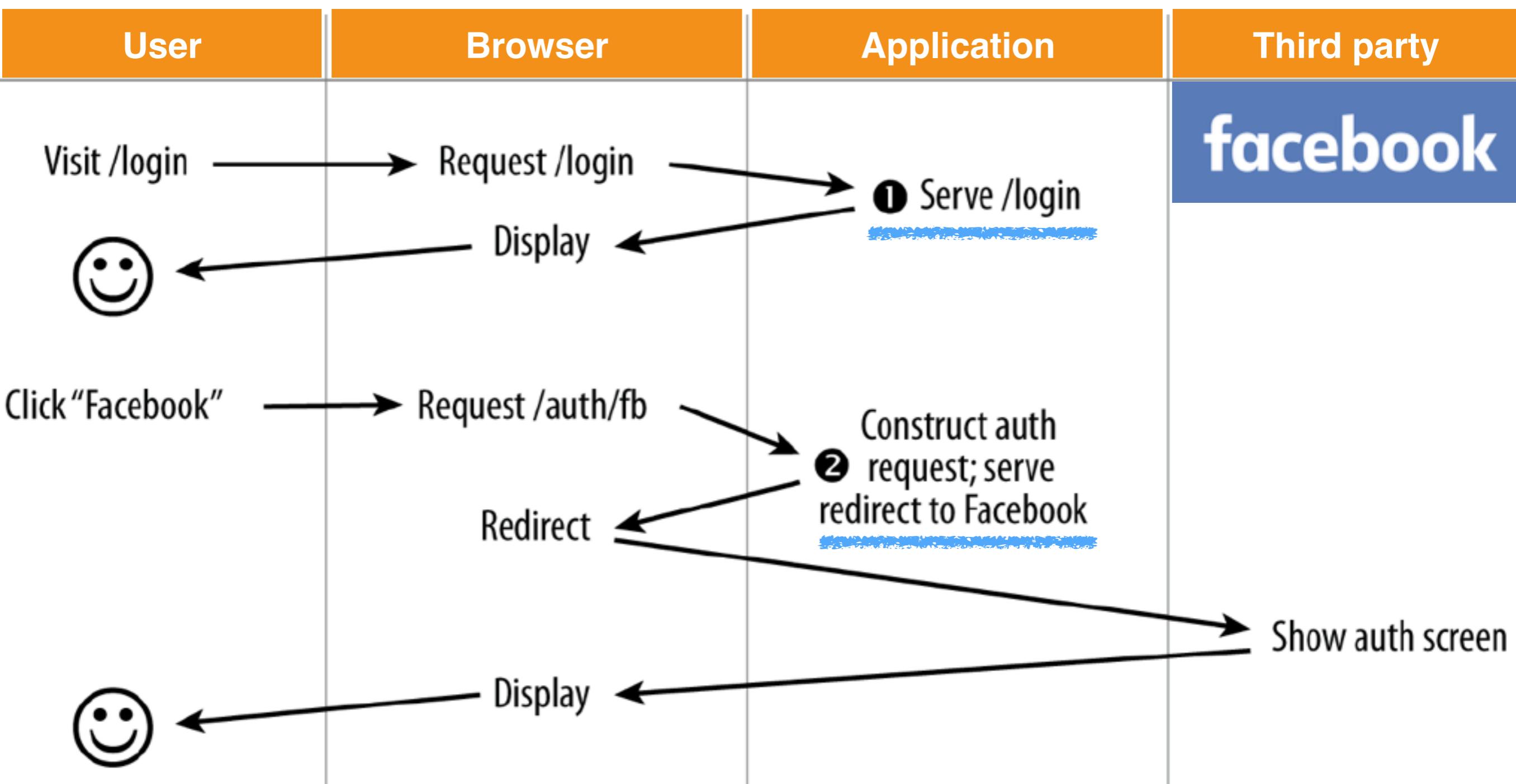
# OAuth 2.0 roles exemplified



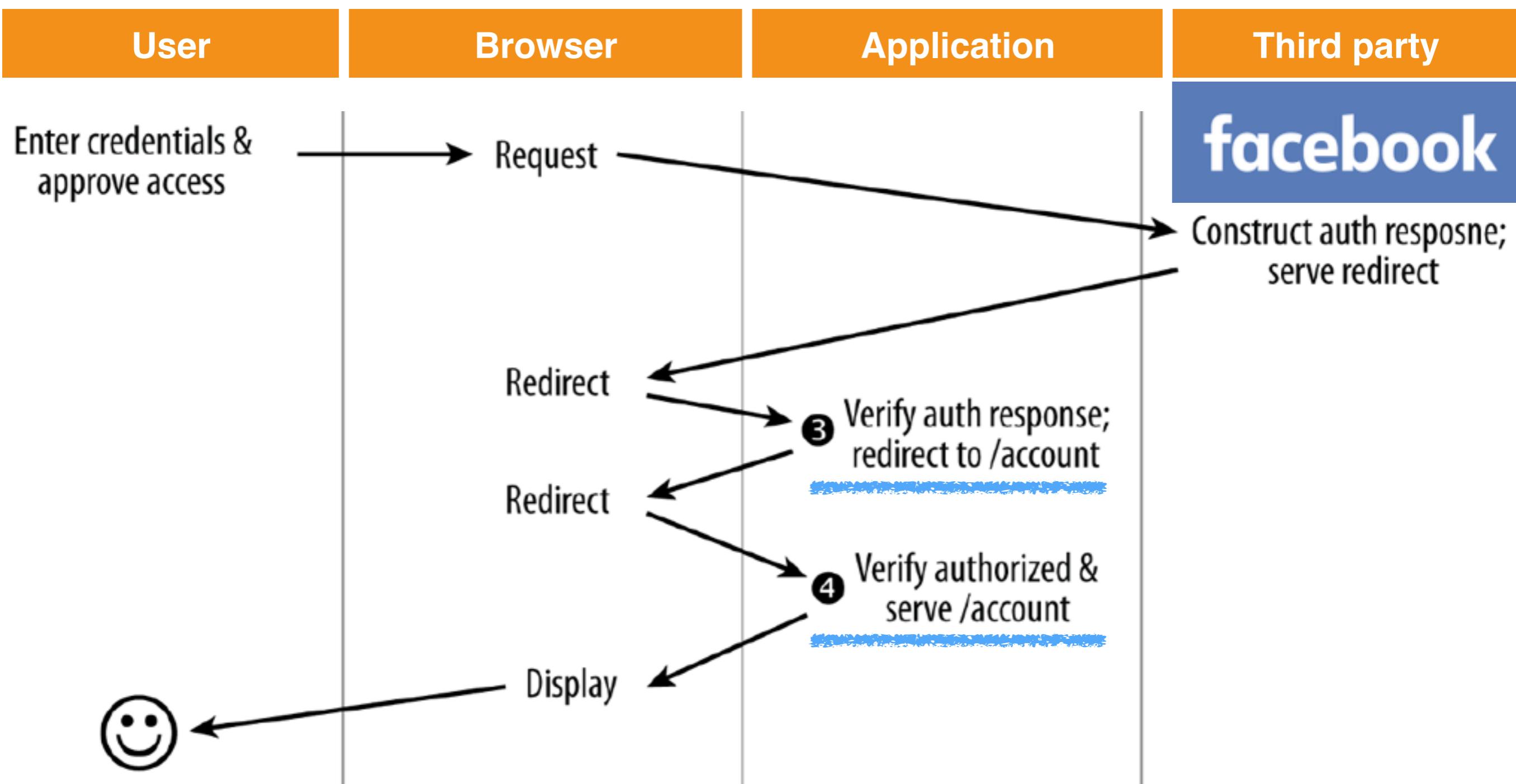
# OAuth 2.0 roles exemplified



# Third-party authentication in practice (steps 1 & 2)



# Third-party authentication in practice (steps 3 & 4)



# Third-party authentication stepwise

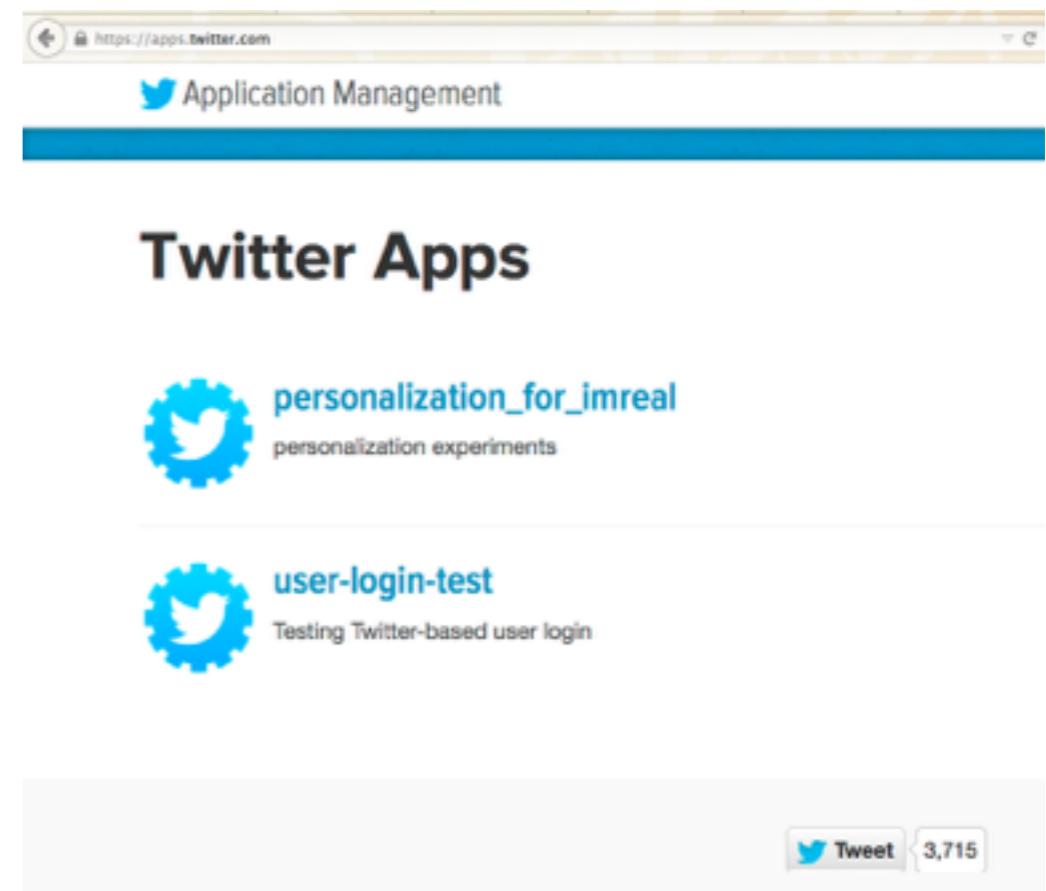
- **Login method**: browser displays options and the user selects one
- **Authentication request**: construct request to be send to the third party
  - You can ask for more (name, email, etc.)
  - Requests differ considerably between services
- **Verify the authentication response**: distinguish between authorised (i.e. valid auth response) and unauthorised access
  - Authorised users should be given a session
- **Verify the authorisation**: is verified user  $U$  allowed to access the resource  $R$ ?
  - Store access rights in a database

# Third-party authentication

## Twitter example

Goal: “Sign in with your Twitter account”

- Works similarly (but not in exactly the same way) across different services
- Starting point: **create** an “app” (Twitter app, Facebook app, etc.)



<https://apps.twitter.com/>

# Third-party authentication

## Twitter example

### Create an application

**Application Details**

**Name \***  
  
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

**Description \***  
  
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

**Website \***  
  
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.  
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

**Callback URL**

<https://apps.twitter.com/>

# Third-party authentication

## Twitter example

### Create an application

Application Details

**Name \***

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

**Description \***

Your app's description. This is used in user-facing authorization screens. Between 10 and 200 characters max.

**Website \***

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.  
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

**Callback URL**

<https://apps.twitter.com/>

# Third-party authentication

## Twitter example

### Create an application

Application Details

**Name \***

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

**Description \***

Your app's description. This is used in user-facing authorization screens. Between 10 and 200 characters max.

**Website \***

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.  
(If you don't have a URL)

**Callback URL**

URL to callback from third party with result

<https://apps.twitter.com/>

# Third-party authentication

## Twitter example cont.

- In application settings, check “Allow this application to be used to Sign in with Twitter”
- Read out the **access tokens**

**user-login-test** Test OAuth

[Details](#) [Settings](#) [Keys and Access Tokens](#) [Permissions](#)

---

### Application Settings

*Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.*

Consumer Key (API Key)	usqAEPbH7I0gJV3oxqyf04KZW
Consumer Secret (API Secret)	5IxThWZZEYgqUhFTu0IPfWPIZTNUtTIEUU2mTLyAMrefHrbfx
Access Level	Read and write ( <a href="#">modify app permissions</a> )
Owner	CharlotteHase
Owner ID	305474242

# Third-party authentication

## Twitter example cont.

- In application settings, check “Allow this application to be used to Sign in with Twitter”
- Read out the **access tokens**

**user-login-test** Test OAuth

[Details](#) [Settings](#) [Keys and Access Tokens](#) [Permissions](#)

**Application Settings** we need this key and secret

*Keep the "Consumer Secret" a secret.*

Consumer Key (API Key)	usqAEPbH7I0gJV3oxqyf04KZW
Consumer Secret (API Secret)	5IxThWZZEYgqUhFTu0IPfWPIZTNUtTIEUU2mTLyAMrefHrbfx
Access Level	Read and write ( <a href="#">modify app permissions</a> )
Owner	CharlotteHase
Owner ID	305474242

# Third-party authentication

## Twitter example cont.

- Express can make use of `passport`, one of the most popular **authentication middleware components**
  - 300+ authentication **strategies**
  - Supports OpenID and OAuth
- `Passport` hides a lot of complexity from you

Installing a strategy

```
$ npm install passport-twitter
```

<http://passportjs.org/>

# Example: authenticating through Twitter

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files: .vscode, static, credentials.js (selected), jsconfig.json, and twitterLogin.js.
- Code Editor (Center):** Displays the contents of the credentials.js file:

```
1 module.exports = {
2     cookieSecret: 'my_secret_abc_123',
3     secret: 'my_secret_abc_123',
4     cookie : {maxAge : 6000},
5     twitter : {
6         consumerKey: "U8A4b3divWaTMQoZqJKNe2ld3",
7         consumerSecret: "hA80EmCboIyieggH41fDqI2t0XfPIYNadzxX808caN16cAbGmW",
8         callbackURL: "http://127.0.0.1:3005/test-login"
9     }
10};
11
12
```
- Status Bar (Bottom):** Shows file statistics (1 file, 0 changes), encoding (UTF-8), line endings (LF), language (JavaScript), and a smiley face icon.



# Example: authenticating through Twitter

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files: .vscode, static, credentials.js (selected), jsconfig.json, and twitterLogin.js.
- Code Editor (Center):** Displays the contents of the credentials.js file:

```
1 module.exports = {
2     cookieSecret: 'my_secret_abc_123',
3     secret: 'my_secret_abc_123',
4     cookie : {maxAge : 6000},
5     twitter : {
6         consumerKey: "U8A4b3divWaTMQoZqJKNe2ld3",
7         consumerSecret: "hA80EmCboIyieggH41fDqI2t0XfPIYNadzxX808caN16cAbGmW",
8         callbackURL: "http://127.0.0.1:3005/test-login"
9     }
10};
11
12
```
- Status Bar (Bottom):** Shows file statistics (1 file, 0 changes), encoding (UTF-8), line/character information (Ln 11, Col 1), and file type (JavaScript).



# Third-party authentication

## Twitter example cont.

```
1 // Redirect the user to Twitter for authentication.  
2 app.get('/auth/twitter', passport.authenticate('twitter'));  
3  
4 // Twitter will redirect the user to this URL after approval.  
5 app.get('/test-login',  
6   passport.authenticate('twitter', { failureRedirect: '/failure' }),  
7   function(req, res) {  
8     res.redirect('/success');  
9   });  
10  
11 app.get("/success", function (req, res) {  
12   console.log("Success!");  
13   res.send("User login via Twitter successful!");  
14 });  
15  
16 app.get("/failure", function (req, res) {  
17   console.log("Failure!");  
18   res.send("User login via Twitter was unsuccessful!");  
19 });
```

# Third-party authentication

## Twitter example cont.

```
1 <!doctype html>
2   <head>
3   </head>
4   <body>
5     <a href="/auth/twitter">Sign in with Twitter</a>
6   </body>
7 </html>
```

# Summary

- Cookies
- Sessions
- Third-party authentication

# **End of Lecture**