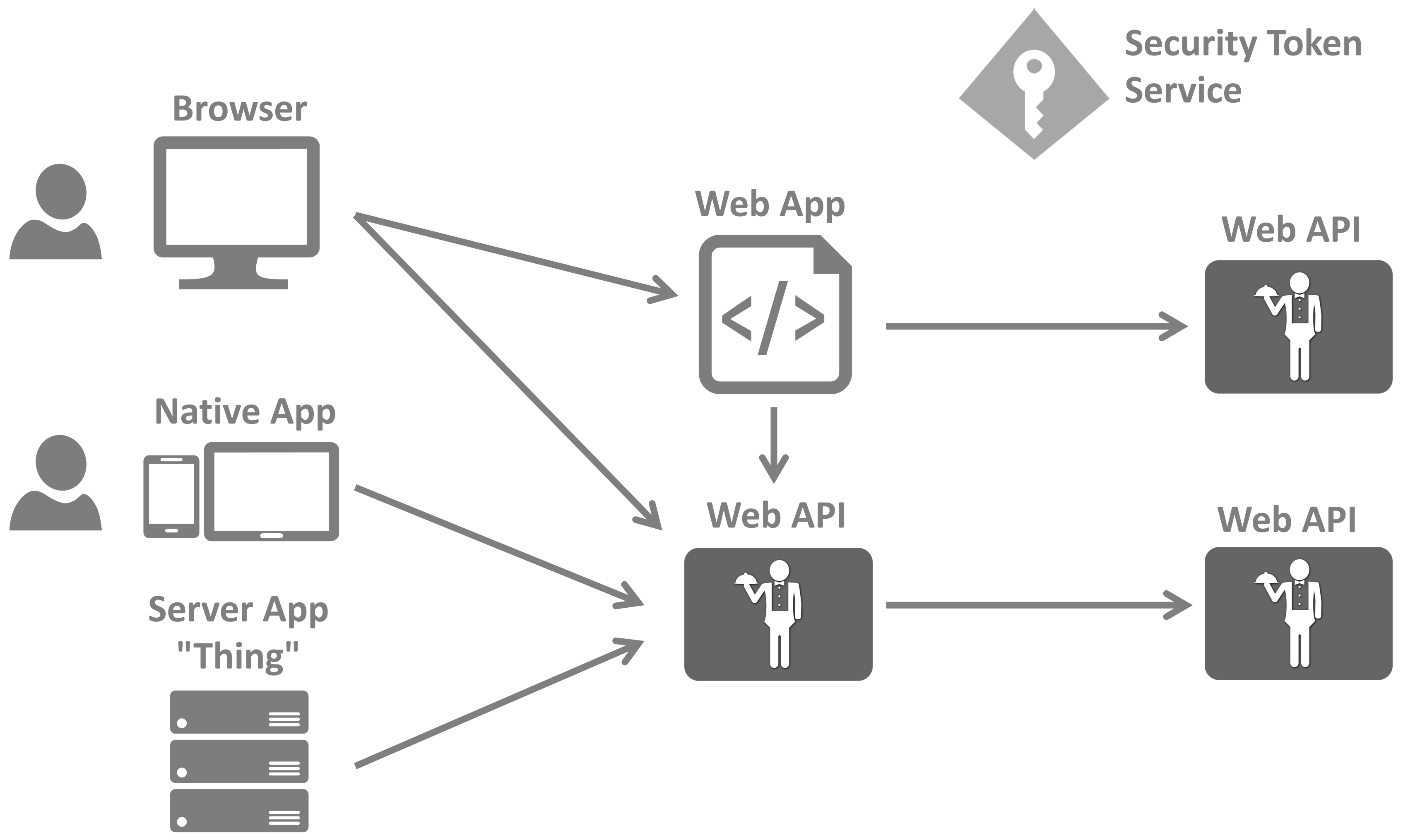
Securing Web APIs

Dominick Baier
http://leastprivilege.com
@leastprivilege

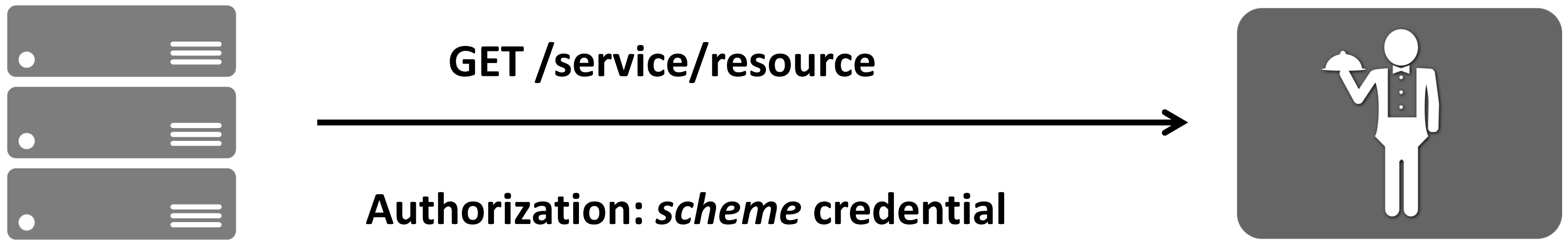
Brock Allen
http://brockallen.com
@brocklallen

The Big Picture

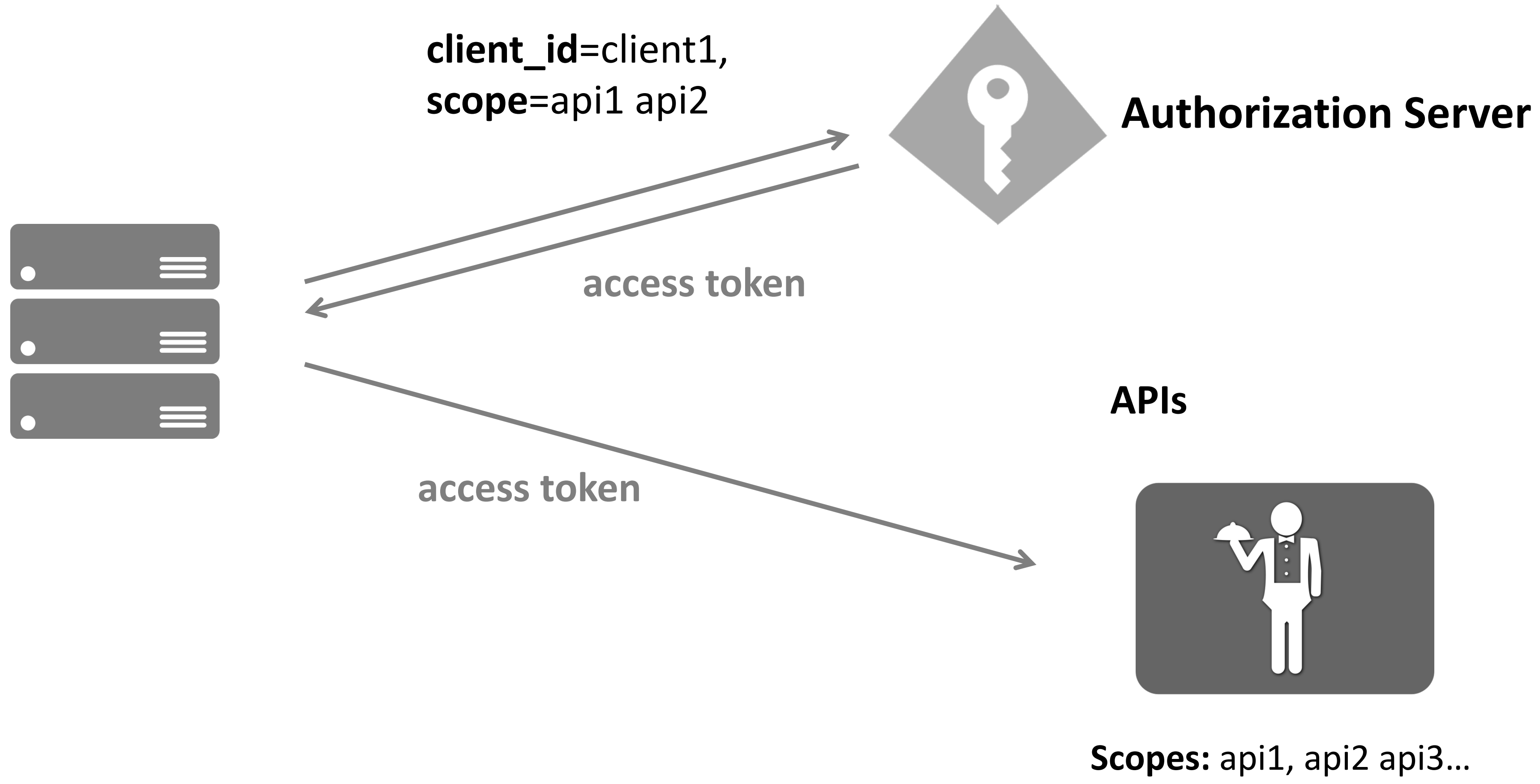


Server to Server Communication

- **Credentials transmitted (typically) via *Authorization* header**
  - e.g. shared secrets, signatures, access tokens…


GET /service/resource
Authorization: *scheme* credential

OAuth 2.0


client_id=client1,
scope=api1 api2
Authorization Server
access token
access token
APIs
Scopes: api1, api2 api3…

Access Tokens

Header

```
{
  "typ": "JWT",
  "alg": "RS256"
  "kid": "1"
}
```

Payload

```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",

  "client_id": "client1",
  "scope": ["api1", "api2"]
}
```

401 vs 403

RFC 7235: HTTP 1.1 Authentication

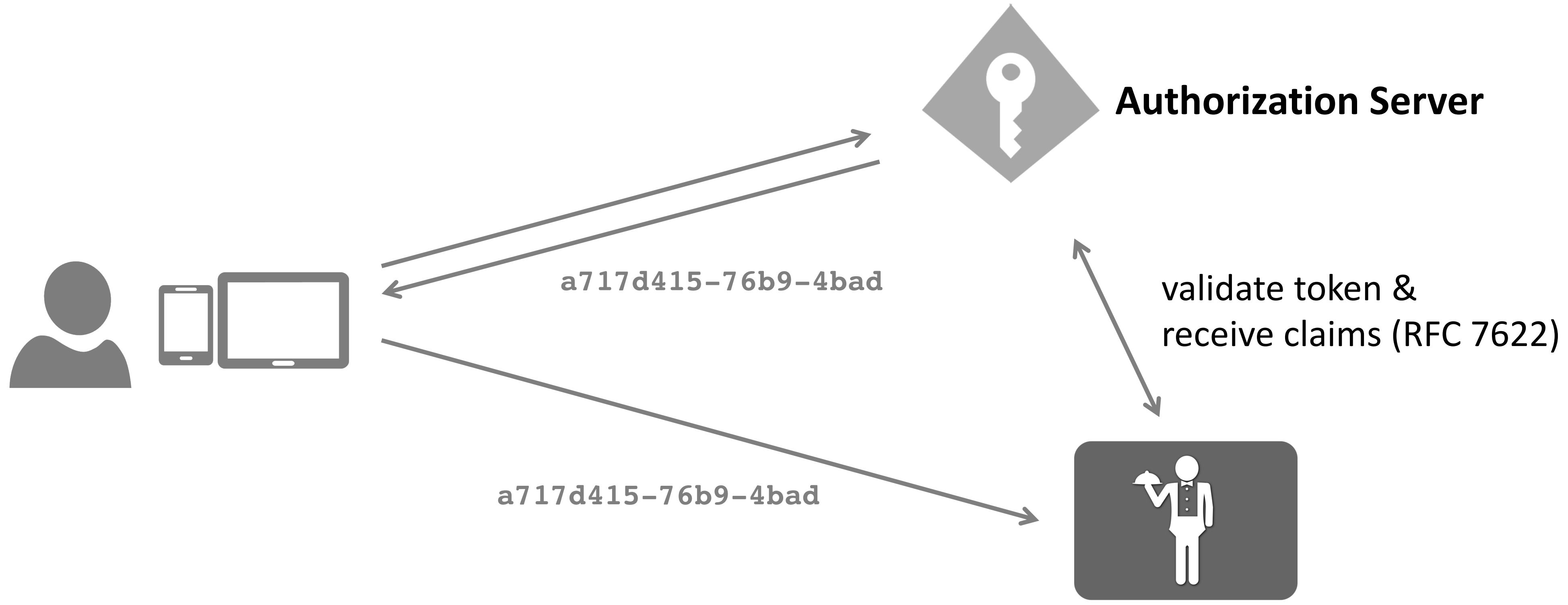
```
The 401 (Unauthorized) status code indicates that the request has
    not been applied because it lacks valid authentication
    credentials for the target resource.  The server generating a
    401 response MUST send a WWW-Authenticate header field
    (Section 4.1) containing at least one challenge applicable to
    the target resource.


A server that receives valid credentials that are not adequate to
    gain access ought to respond with the 403 (Forbidden) status
    code
```

Access Token Validation

- **JWT bearer token authentication handler**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication("Bearer")
        .AddJwtBearer("Bearer", options =>
        {
            options.Authority = "https://your_oidc_provider";
            options.Audience = "your_api_identifier";
        });
}
```

Reference Tokens



Reference Token Validation

- **Using OAuth 2.0 Introspection**
  - *IdentityModel.AspNetCore.OAuth2Introspection* nuget

```
services.AddAuthentication(OAuth2IntrospectionDefaults.AuthenticationScheme)
    .AddOAuth2Introspection(options =>
    {
        options.Authority = "https://demo.identityserver.io";

        options.ClientId = "api1";
        options.ClientSecret = "secret";
    });
```

IdentityServer Token Validation

- **Combines JWT bearer & introspection**
  - *IdentityServer4.AccessTokenValidation* nuget

```
services.AddAuthentication(IdentityServerAuthenticationDefaults.AuthenticationScheme)
    .AddIdentityServerAuthentication(options =>
    {
        options.Authority = "https://demo.identityserver.io";

        options.ApiName = "api1";
        options.ApiSecret = "secret";
    });
```

User-Centric Clients

- **Typical Pattern**
  - authenticate user
  - make API calls **on behalf** of the user

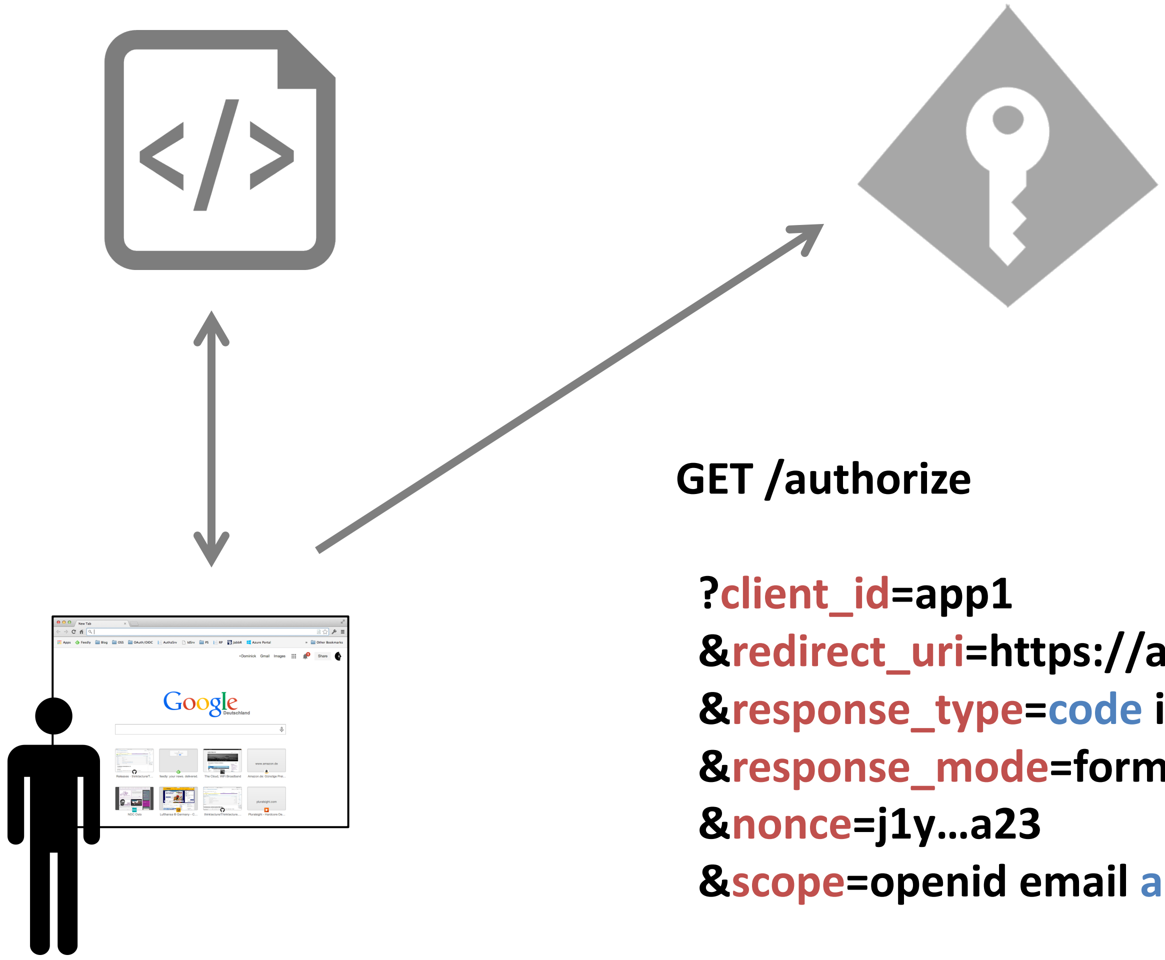
- **Server-side Web Applications**
- **Client-side Web Apps/SPAs**
- **Native/Mobile Applications**

Web Applications

- **OpenID Connect Hybrid Flow combines**
  - user authentication (identity token)
  - access to APIs (access token)

- **Additional Security Features**
  - access tokens not exposed to the browser
  - (optional) long-lived API access

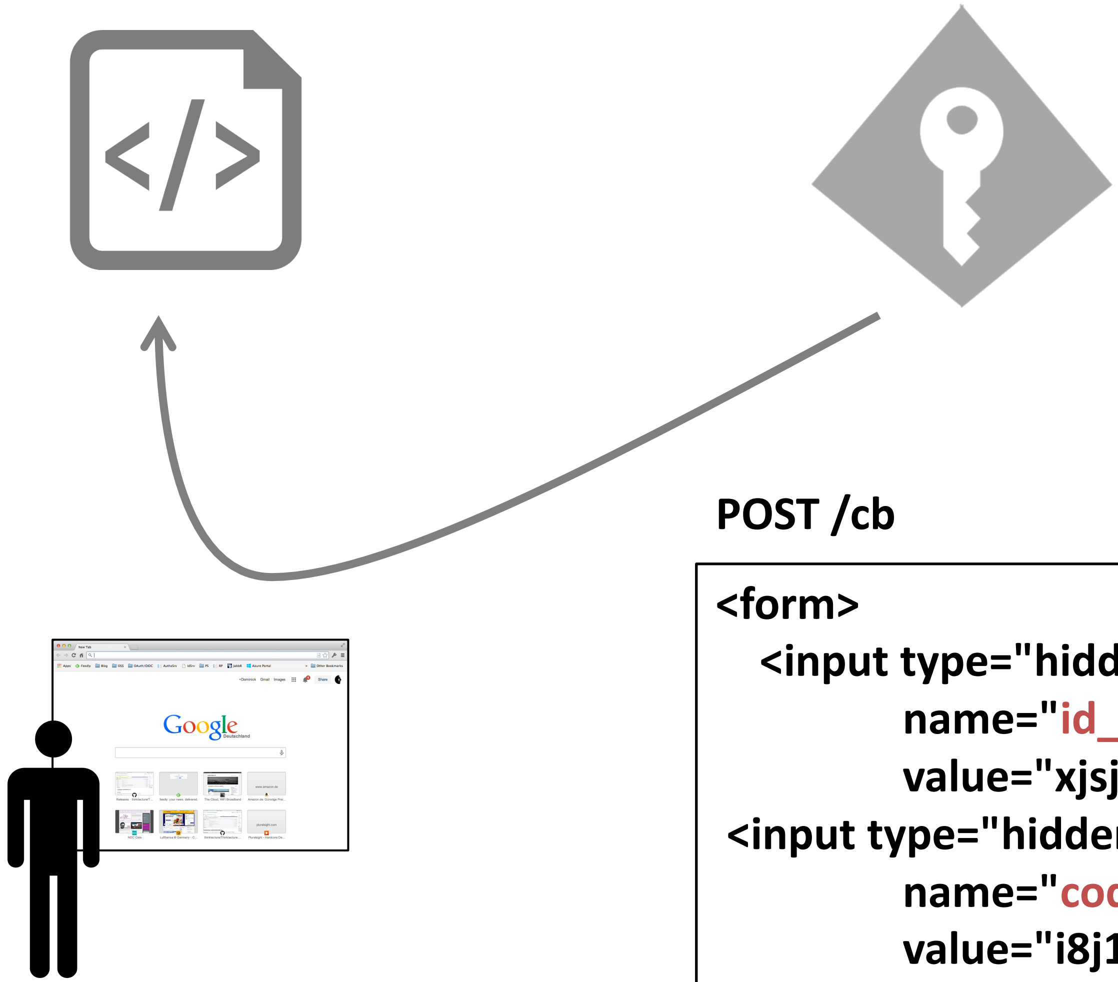
Hybrid Flow Request



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/cb
&response_type=code id_token
&response_mode=form_post
&nonce=j1y…a23
&scope=openid email api1 api2

Hybrid Flow Response

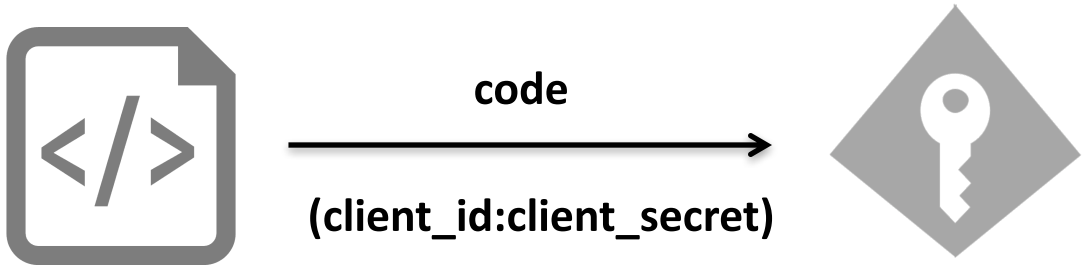


POST /cb

```
<form>
   <input type="hidden"
              name="id_token"
              value="xjsj…aas" />
 <input type="hidden"
              name="code"
              value="i8j1…jj19" />
</form>
```

Retrieving the Access Token

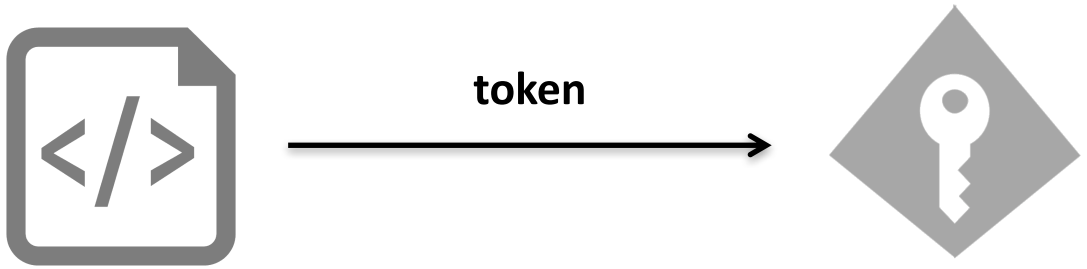
- **Exchange code for access token**
  - using client id and secret



```
{
  access_token: "xyz…123",
  expires_in: 3600,
  token_type: "Bearer"
}
```

UserInfo Endpoint

- **Access token allows to retrieve user claims via a back-channel call**
  - keeps identity token small

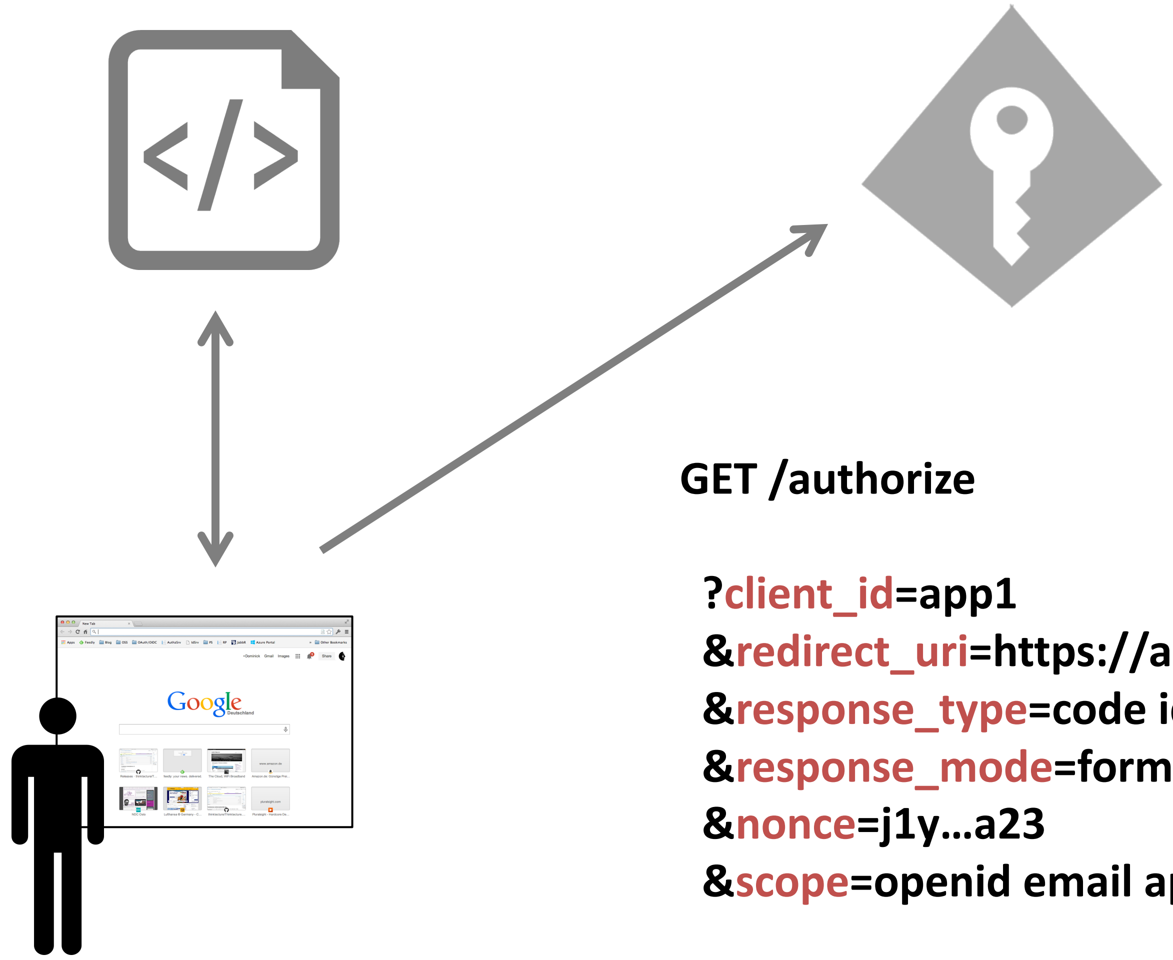


```
{
  sub: "91jj21",
  given_name: "Bob",
  profile_picture: "ksjjj…jdj_"
}
```

Access Token Lifetime Management

- **Access tokens have finite lifetimes**
  - requesting a new token requires browser round trip to authorization server
  - should be as short lived as possible
- **Refresh tokens allow renewal semantics**
  - no user interaction required
  - typically combined with a revocation feature

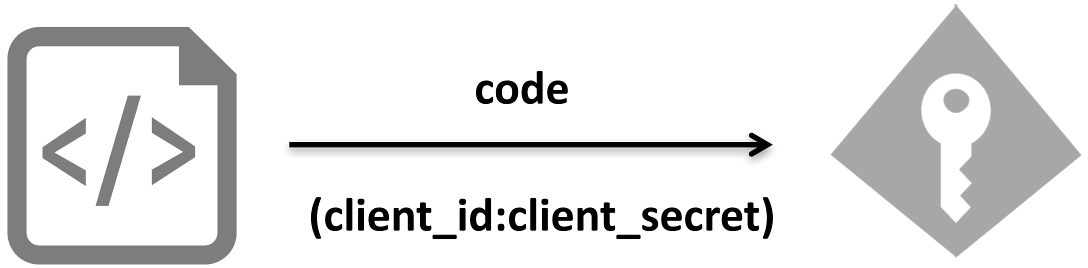
Requesting a Refresh Token



GET /authorize

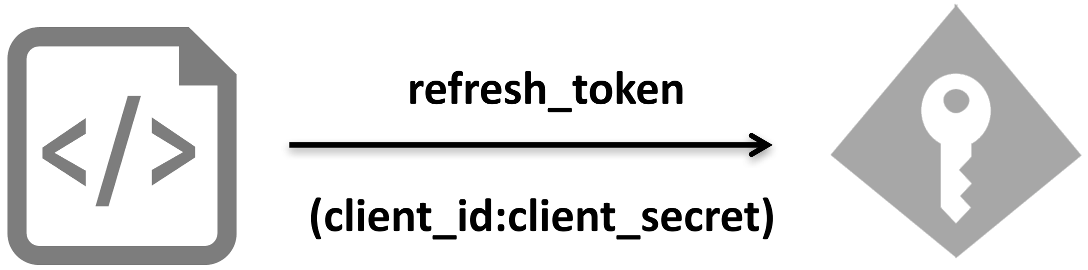
?client_id=app1
&redirect_uri=https://app.com/cb
&response_type=code id_token
&response_mode=form_post
&nonce=j1y…a23
&scope=openid email api1 offline_access

Retrieving the Access Token (w/ Refresh Token)



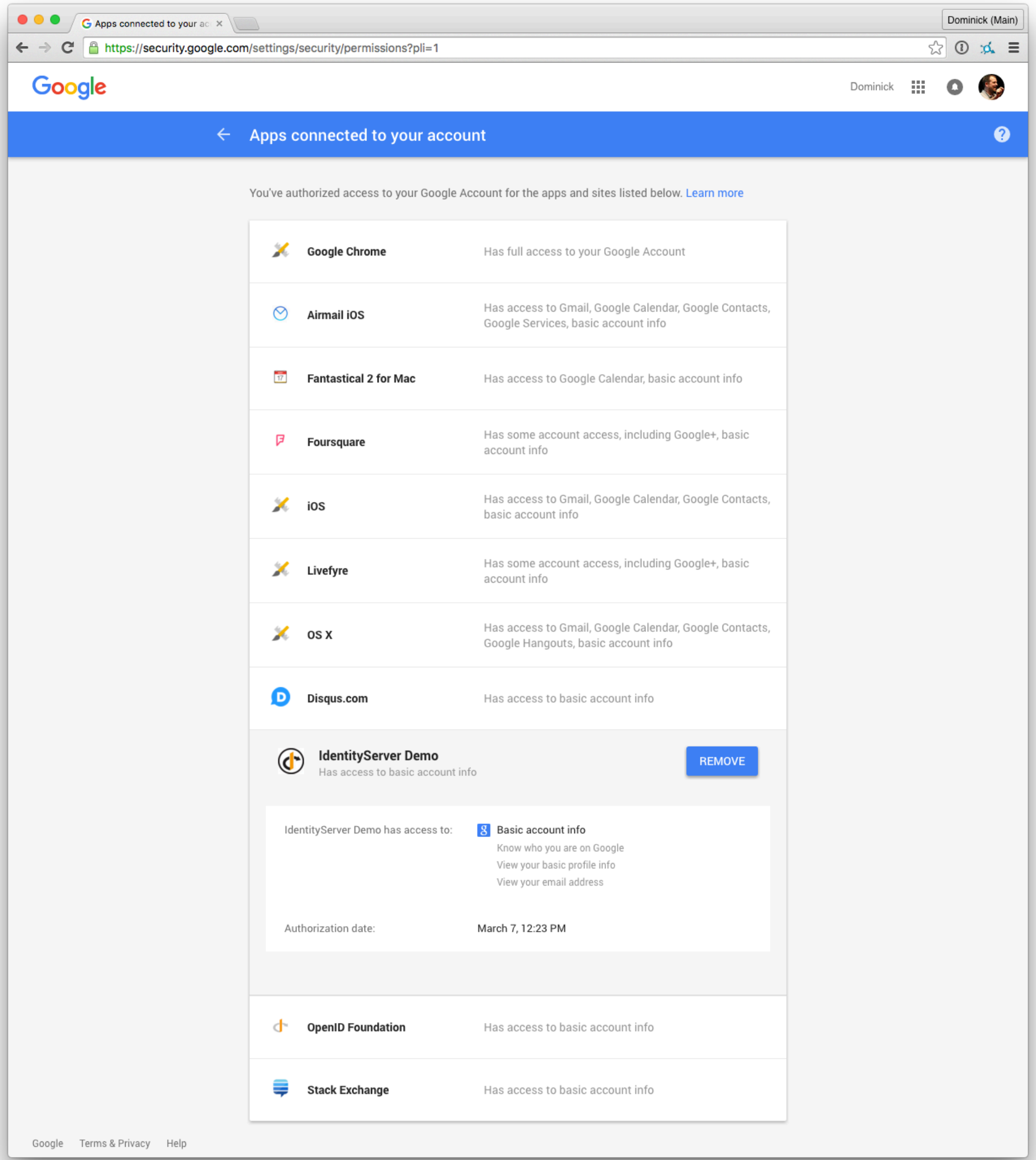
```
{
  access_token: "xyz…123",
  refresh_token: "jdj9…192j",
  expires_in: 3600,
  token_type: "Bearer"
}
```

Refreshing an Access Token



```
{
  access_token: "xyz…123",
  refresh_token: "jdj9…192j",
  expires_in: 3600,
  token_type: "Bearer"
}
```

Revocation


Apps connected to your ac
Dominick (Main)
https://security.google.com/settings/security/permissions?pli=1
Google
Dominick
Apps connected to your account

You've authorized access to your Google Account for the apps and sites listed below. Learn more

Google Chrome	Has full access to your Google Account
Airmail iOS	Has access to Gmail, Google Calendar, Google Contacts, Google Services, basic account info
Fantastical 2 for Mac	Has access to Google Calendar, basic account info
Foursquare	Has some account access, including Google+, basic account info
iOS	Has access to Gmail, Google Calendar, Google Contacts, basic account info
Livefyre	Has some account access, including Google+, basic account info
OS X	Has access to Gmail, Google Calendar, Google Contacts, Google Hangouts, basic account info
Disqus.com	Has access to basic account info

IdentityServer Demo
Has access to basic account info

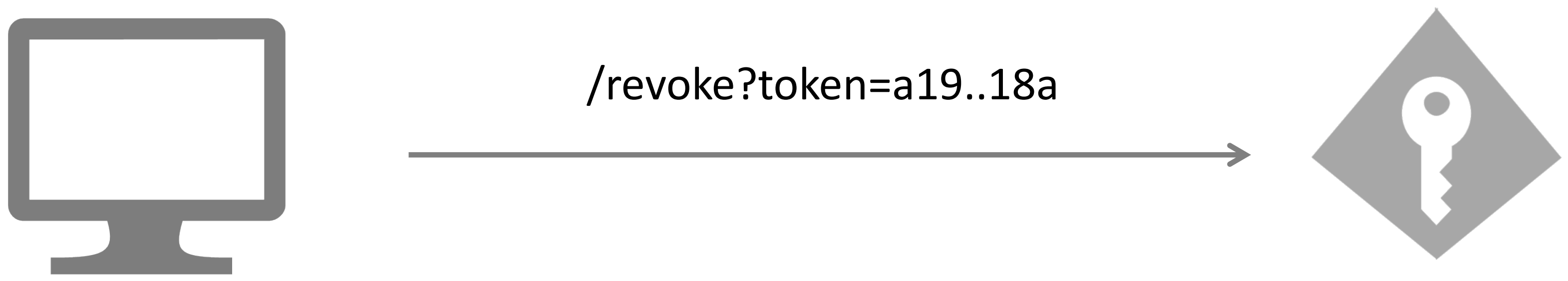
REMOVE

IdentityServer Demo has access to:	Basic account info Know who you are on Google View your basic profile info View your email address
Authorization date:	March 7, 12:23 PM

OpenID Foundation	Has access to basic account info
Stack Exchange	Has access to basic account info

Token Revocation

- **Endpoint to programmatically revoke tokens (RFC 7009)**
  - reference tokens
  - refresh tokens



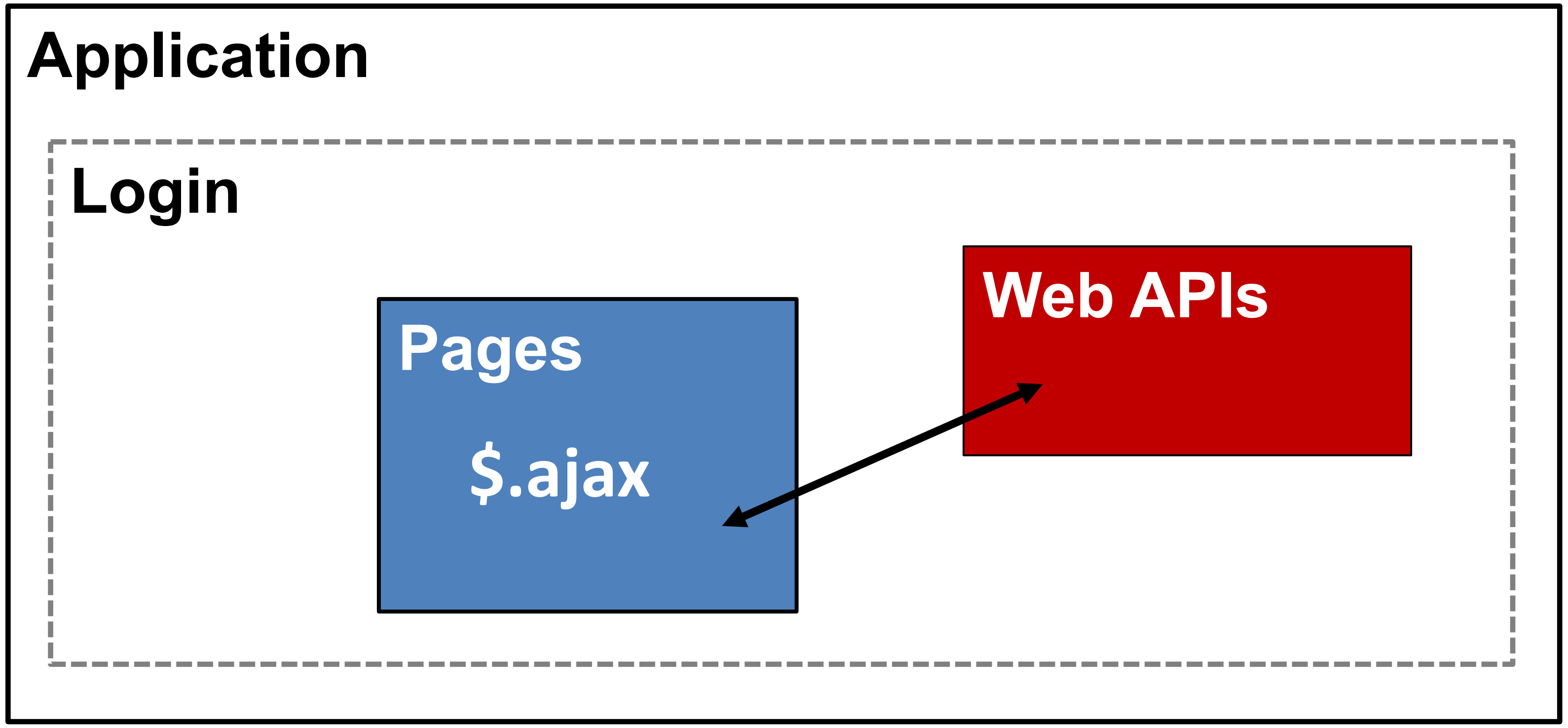
JavaScript Applications - Common Approaches

- **"Legacy" Applications**
  - mixture of server UI and client scripts
  - APIs part of same application
  - often cookies used for session management
  - often CSRF problems

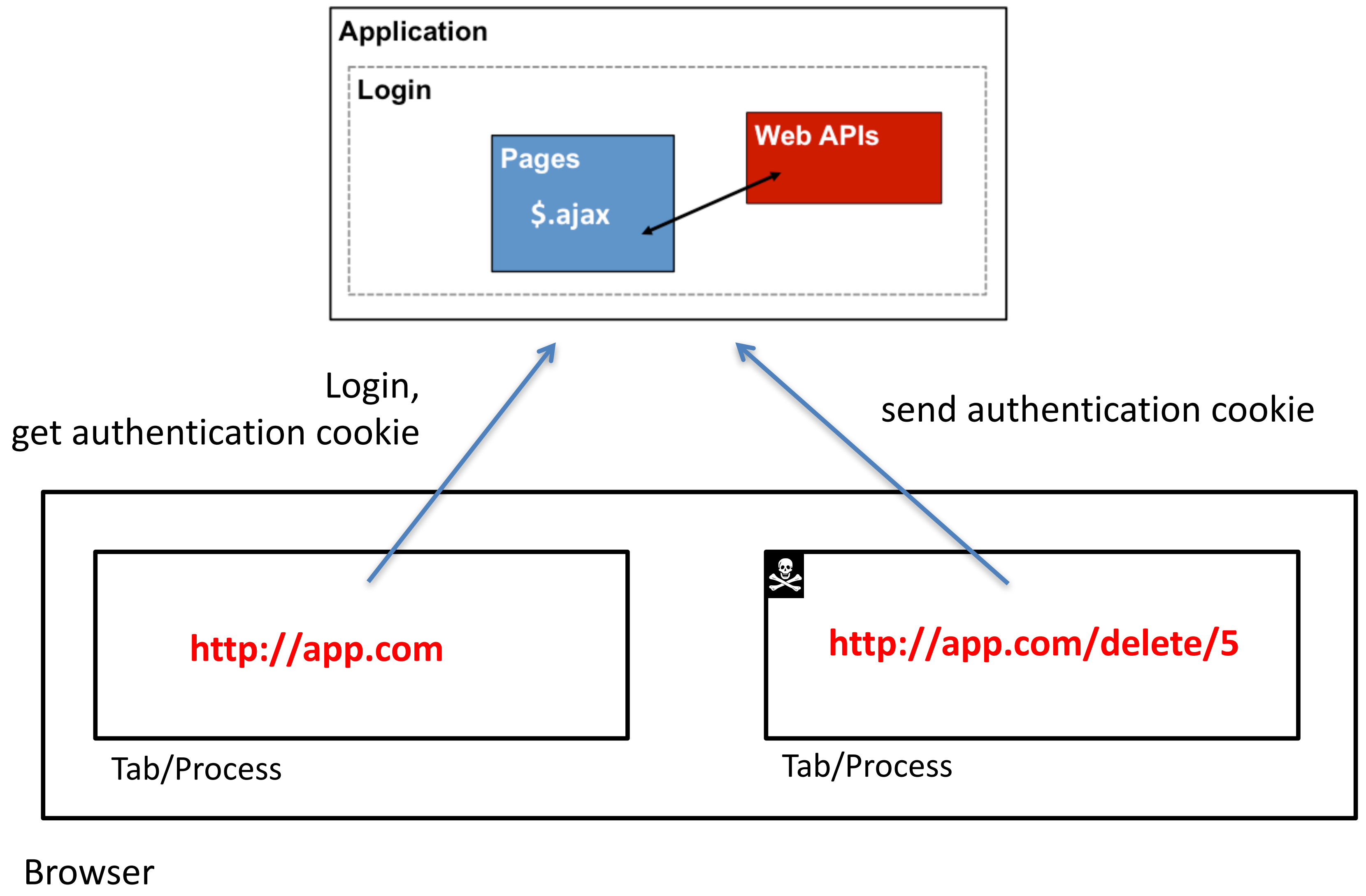
- **"Pure" SPAs**
  - no UI back-end (e.g. served from a CDN)
  - APIs designed to be stand-alone and shareable
  - token-based authentication

"Legacy"

- **Implicit Authentication**
  - e.g. cookies, Windows authentication, client certs...

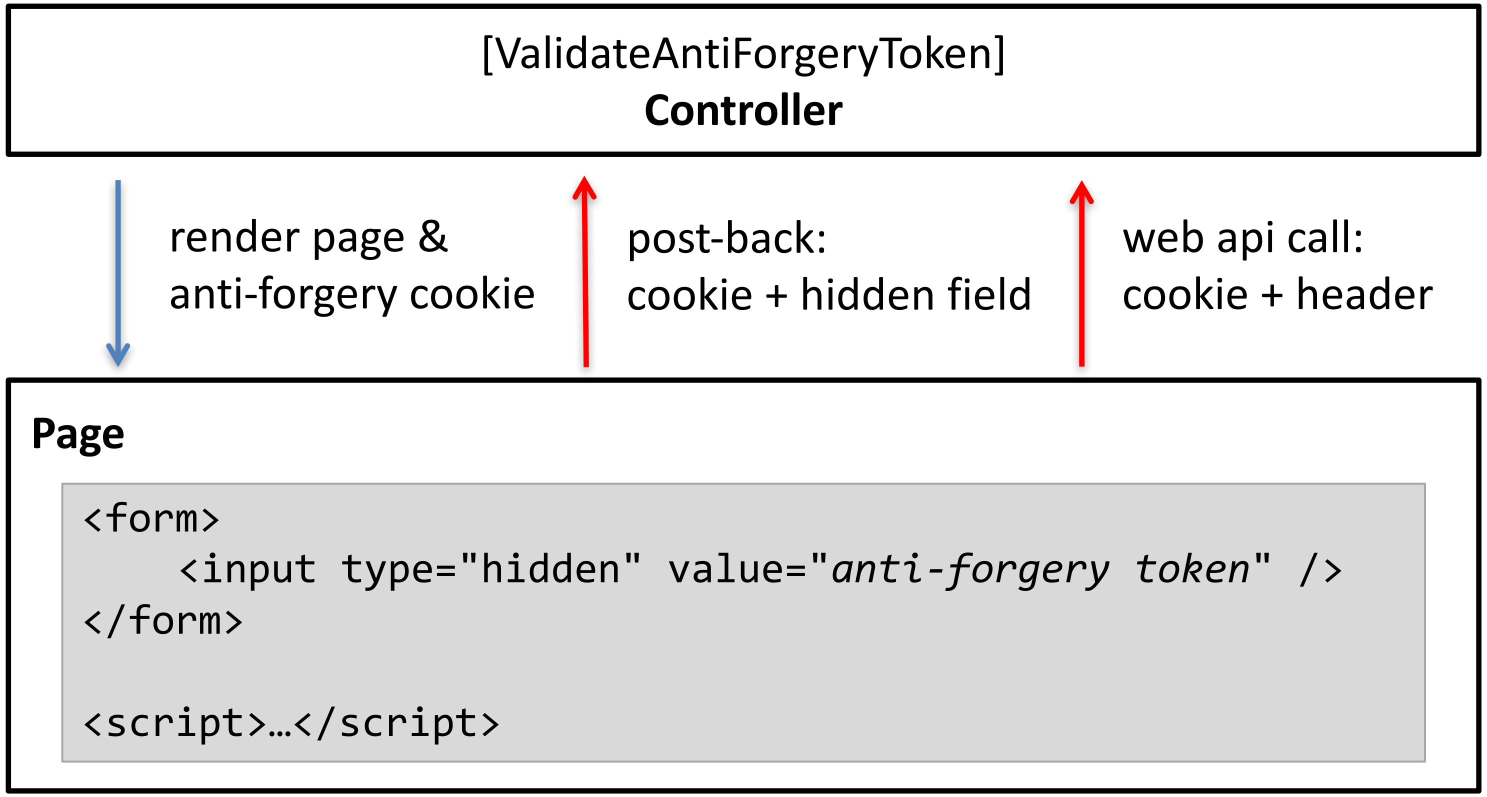


CSRF – The Problem



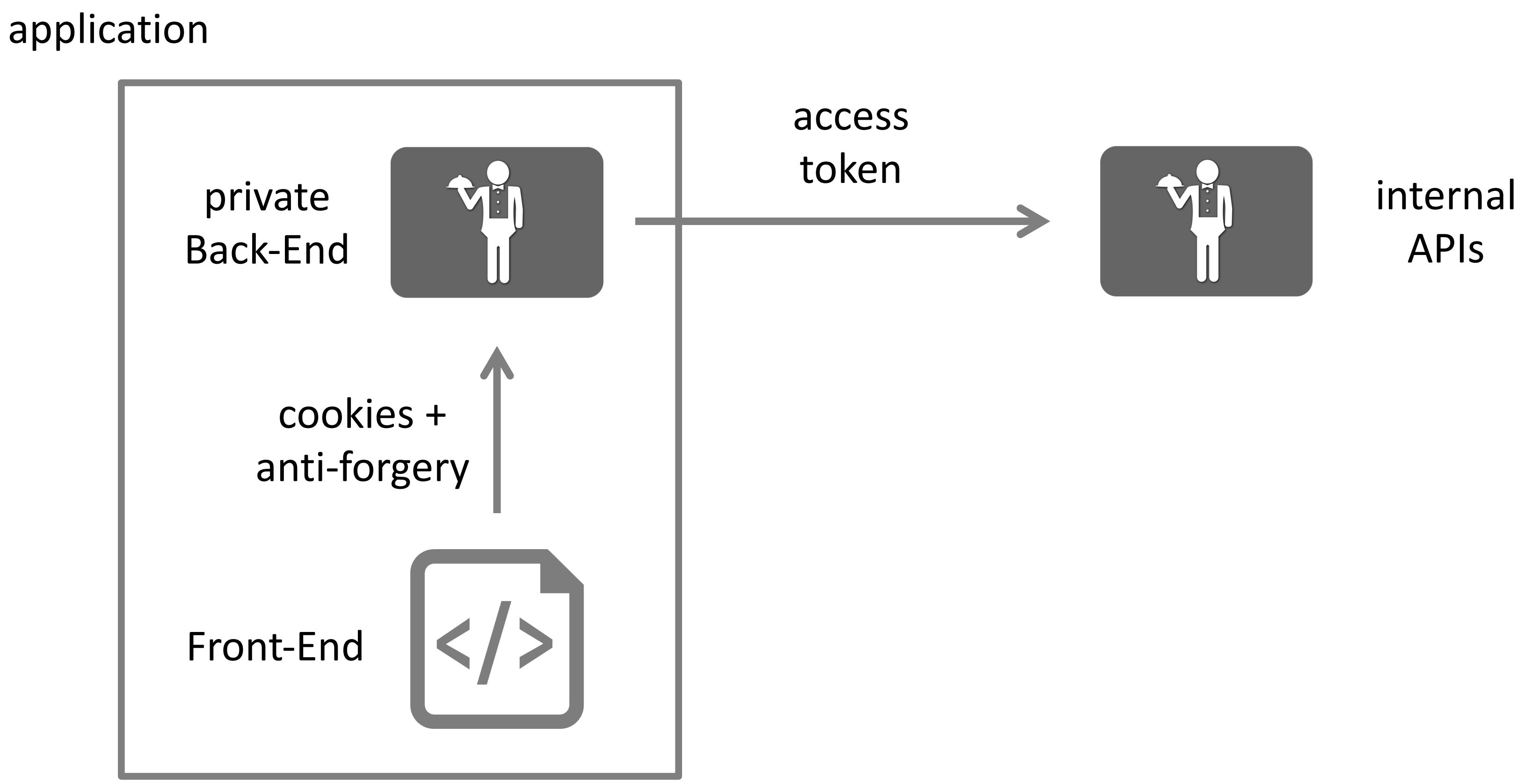
Example: Anti-Forgery Tokens

- **Add explicit "credential"**
  – makes API private to application

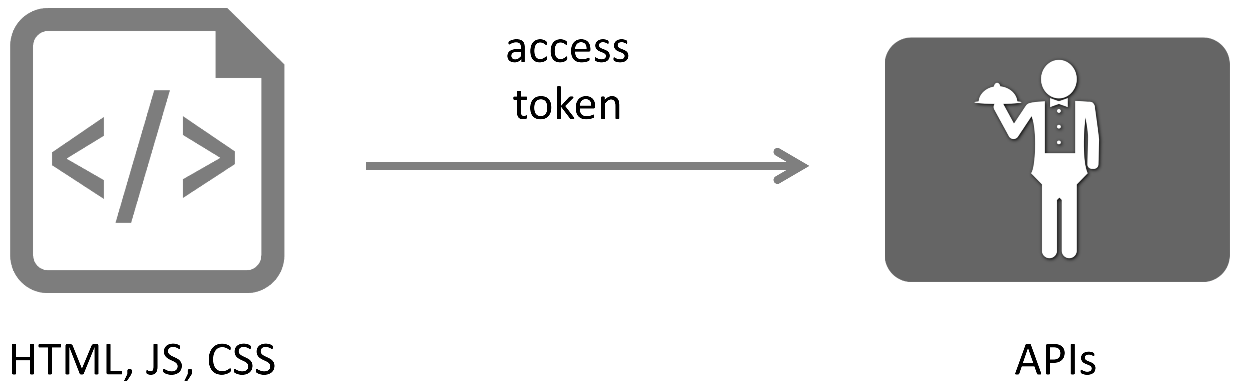

[ValidateAntiForgeryToken]
Controller
render page & anti-forgery cookie
post-back: cookie + hidden field
web api call: cookie + header
Page

```
<form>
    <input type="hidden" value="anti-forgery token" />
</form>

<script>…</script>
```

"BFF" Architecture



"SPA" Architecture

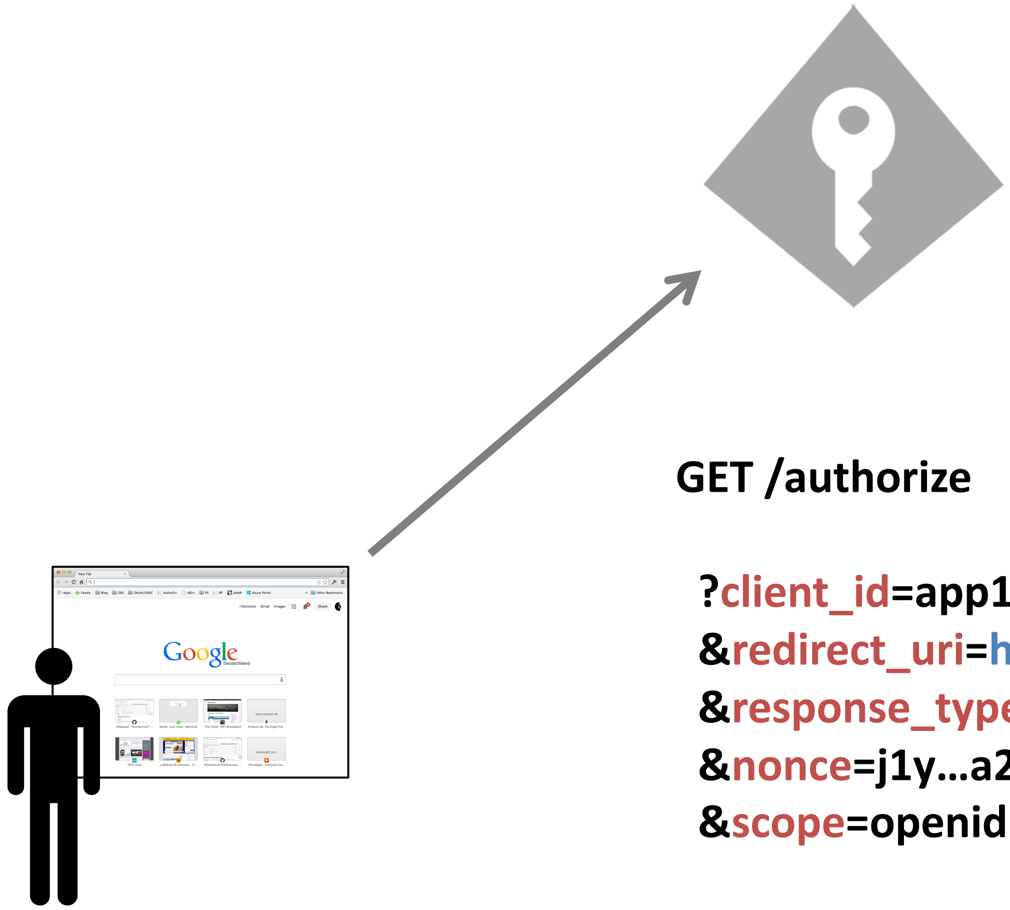


Token-based Authentication

- **OpenID Connect Implicit Flow designed for JS/Browser-based Applications**
  - simplified flow
  - no secret required
  - limited features

- **Tokens always passed explicitly to the API**

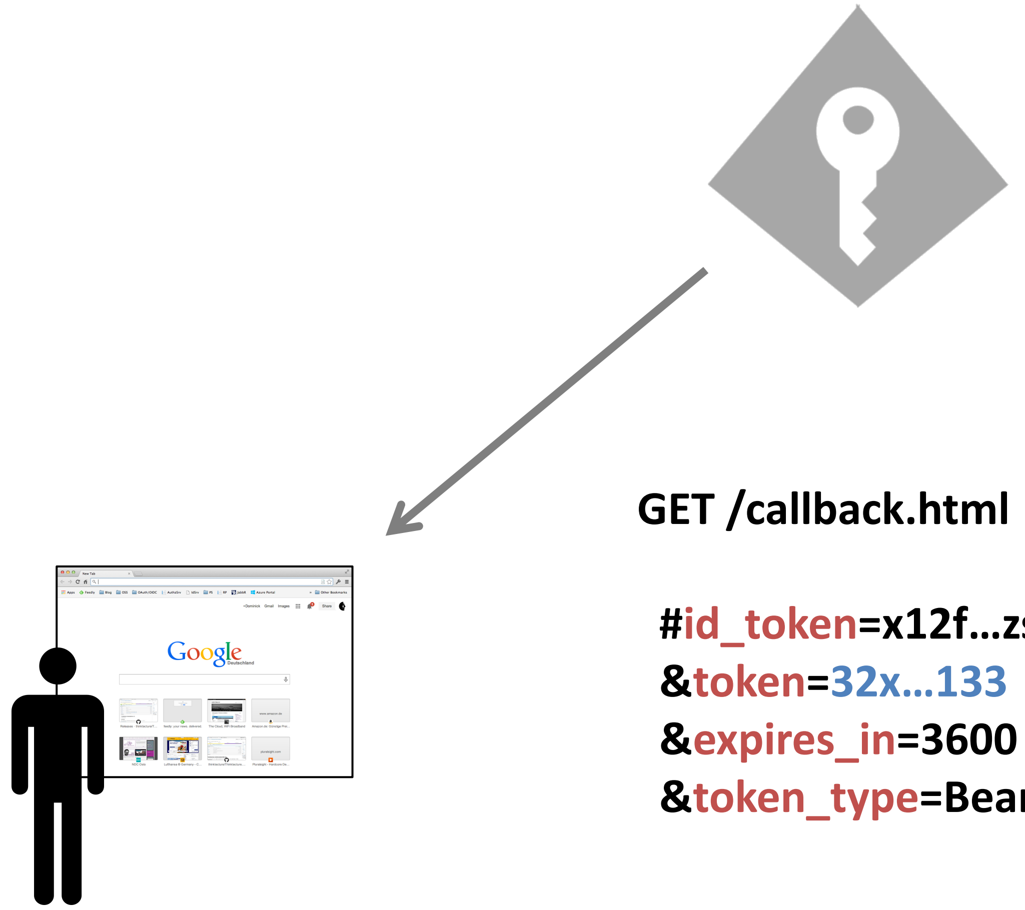
Implicit Flow Request



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/cb.html
&response_type=id_token token
&nonce=j1y…a23
&scope=openid email api1 api2

Response



GET /callback.html

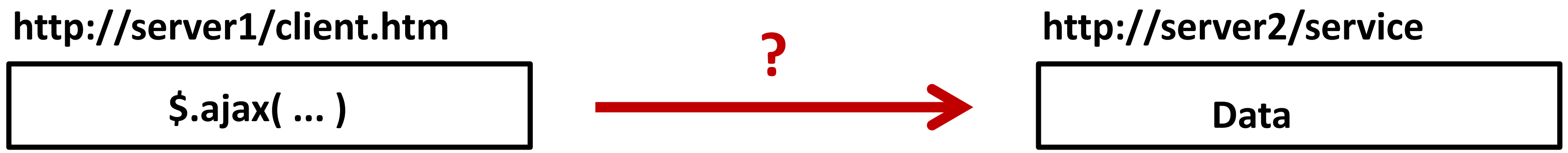
#id_token=x12f…zsz
&token=32x…133
&expires_in=3600
&token_type=Bearer

Java Script Client Library

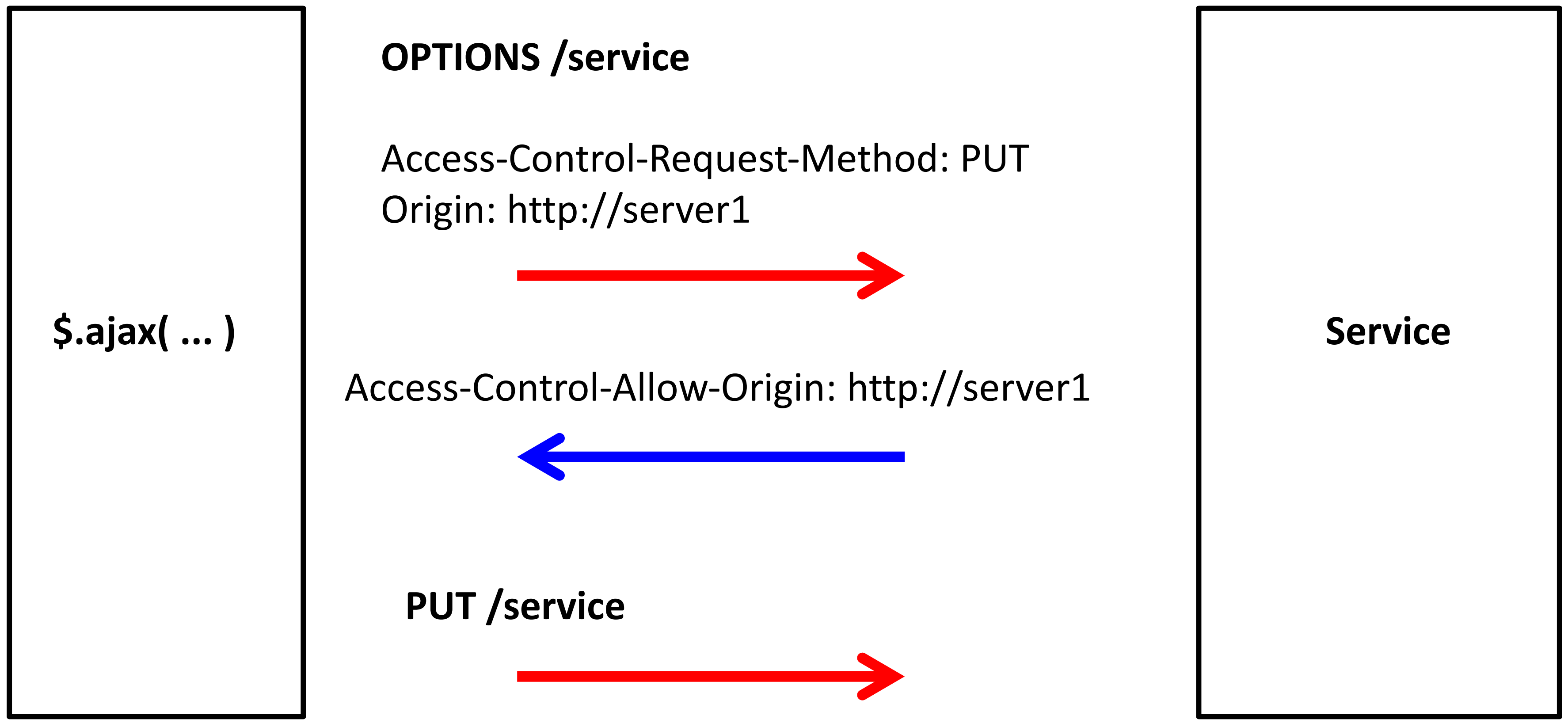
- **https://github.com/IdentityModel/oidc-client-js**

```
var settings = {
    authority: 'http://localhost:5152/',
    client_id: 'spa',
    redirect_uri: 'http://localhost:5152/callback.html',
    response_type: 'id_token token',
    scope: 'openid profile api',
};

var mgr = new Oidc.UserManager(settings);

mgr.getUser().then(function (user) {
    if (user) {
        log("logged in", user);
    }
    else {
        mgr.signinRedirect();
    }
});
```

Excursion: CORS (Cross Origin Resource Sharing)



CORS Sample


$.ajax(...)
OPTIONS /service
Access-Control-Request-Method: PUT
Origin: http://server1
Access-Control-Allow-Origin: http://server1
PUT /service
Service

CORS for ASP.NET Core

- **Available as middleware**

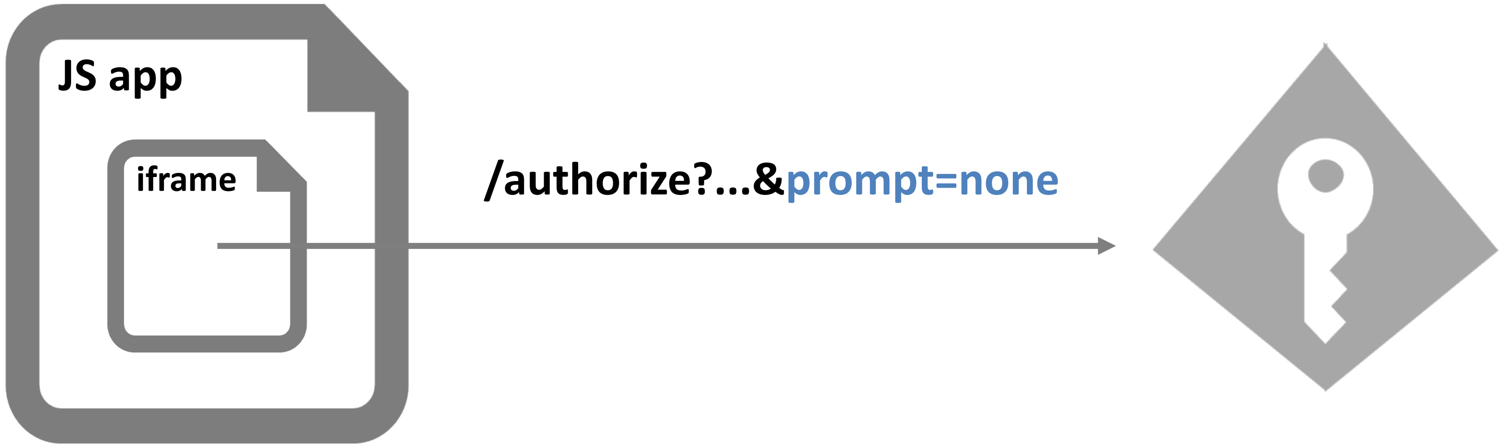
```
public void Configure(IApplicationBuilder app)
{
    app.UseCors(policy =>
    {
        policy.WithOrigins(
            "http://localhost:28895",
            "http://localhost:7017");

        policy.AllowAnyHeader();
        policy.AllowAnyMethod();
    });
}
```

Token Lifetime for JS Apps

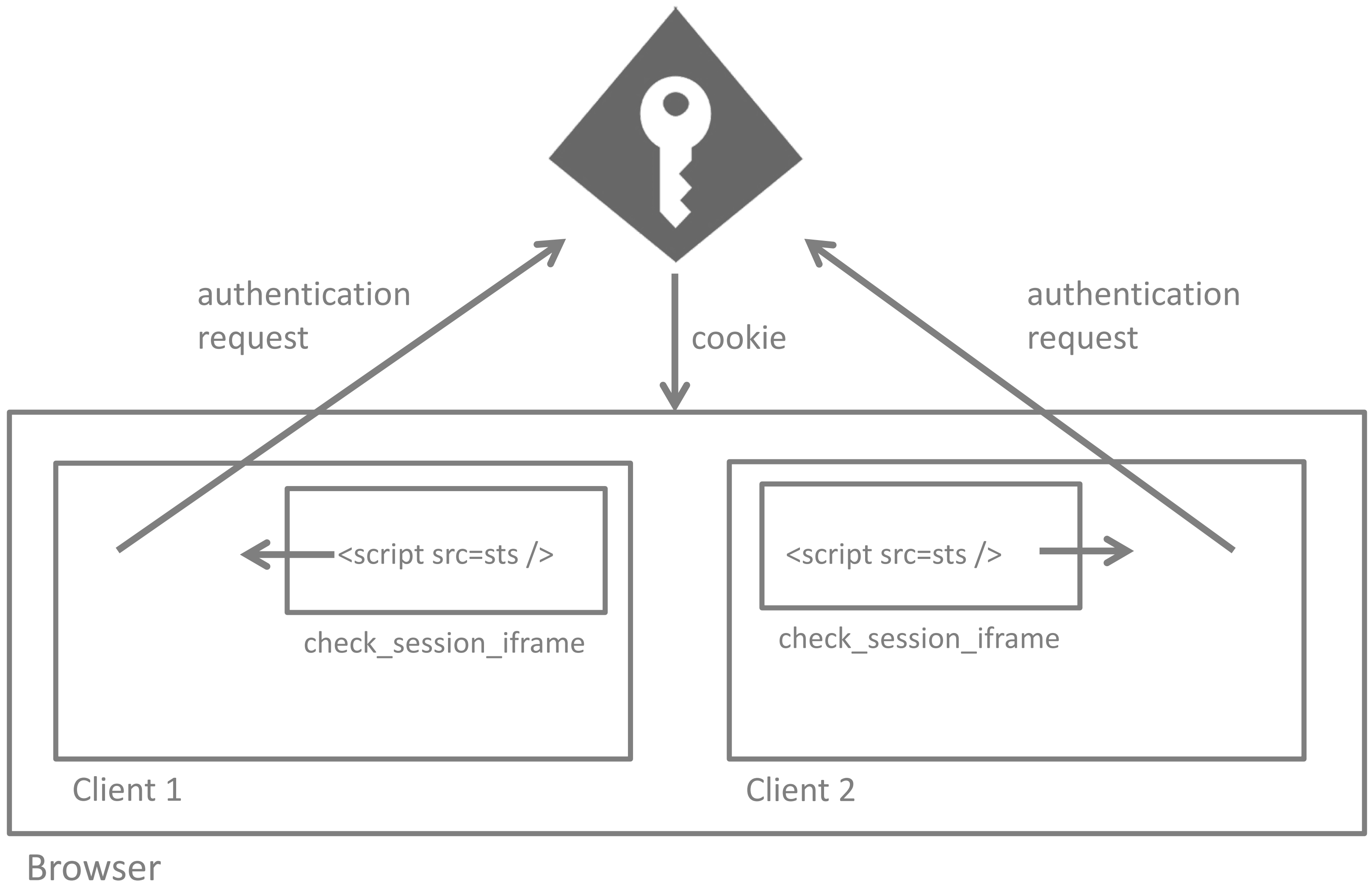
- **Implicit flow does not allow refresh tokens**
  - browser is not a fully trusted environment
- **"Silent renew" technique re-uses browser session**

Silent Renew

- **Request new token in a hidden iframe**
  - only possible if no user interaction is required



OIDC Session Status Change Notifications

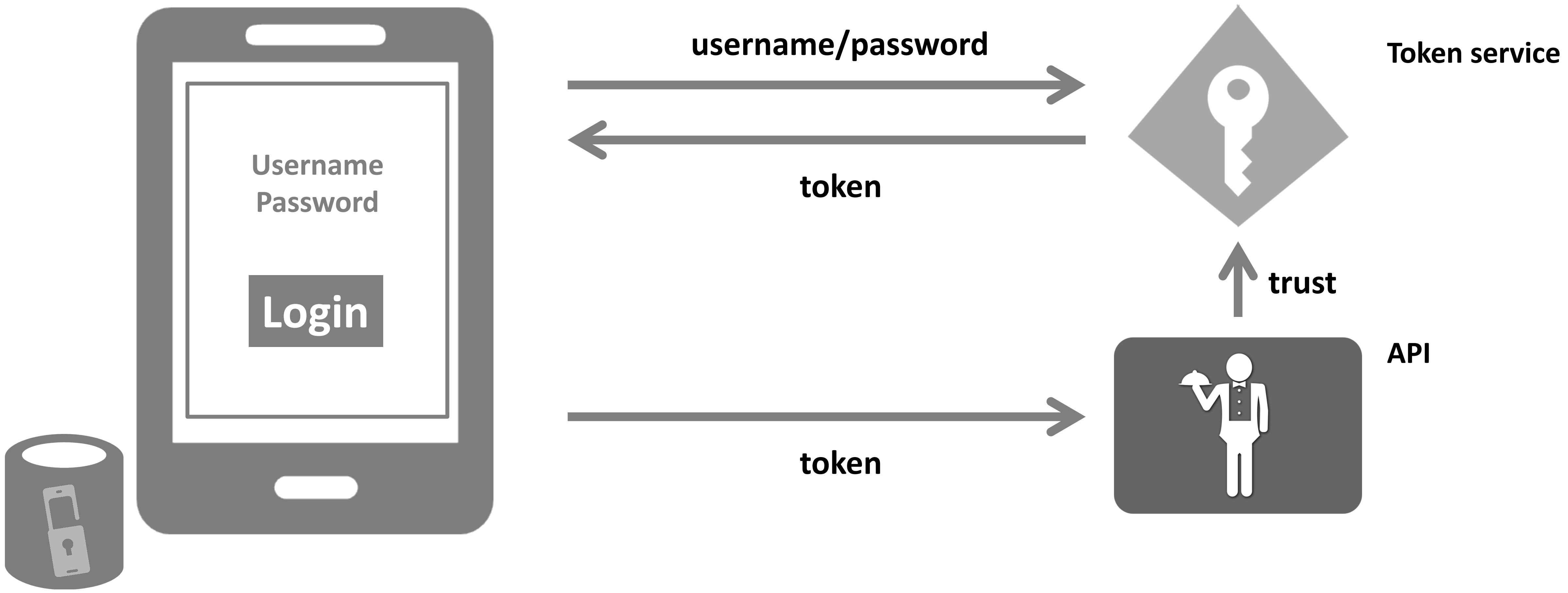


Native/Mobile Applications

- **IOW applications that have access to native platform APIs**
  - desktop or mobile

- **"OAuth 2.0 for native Applications"**
  - https://tools.ietf.org/wg/oauth/draft-ietf-oauth-native-apps/

Native login dialogs

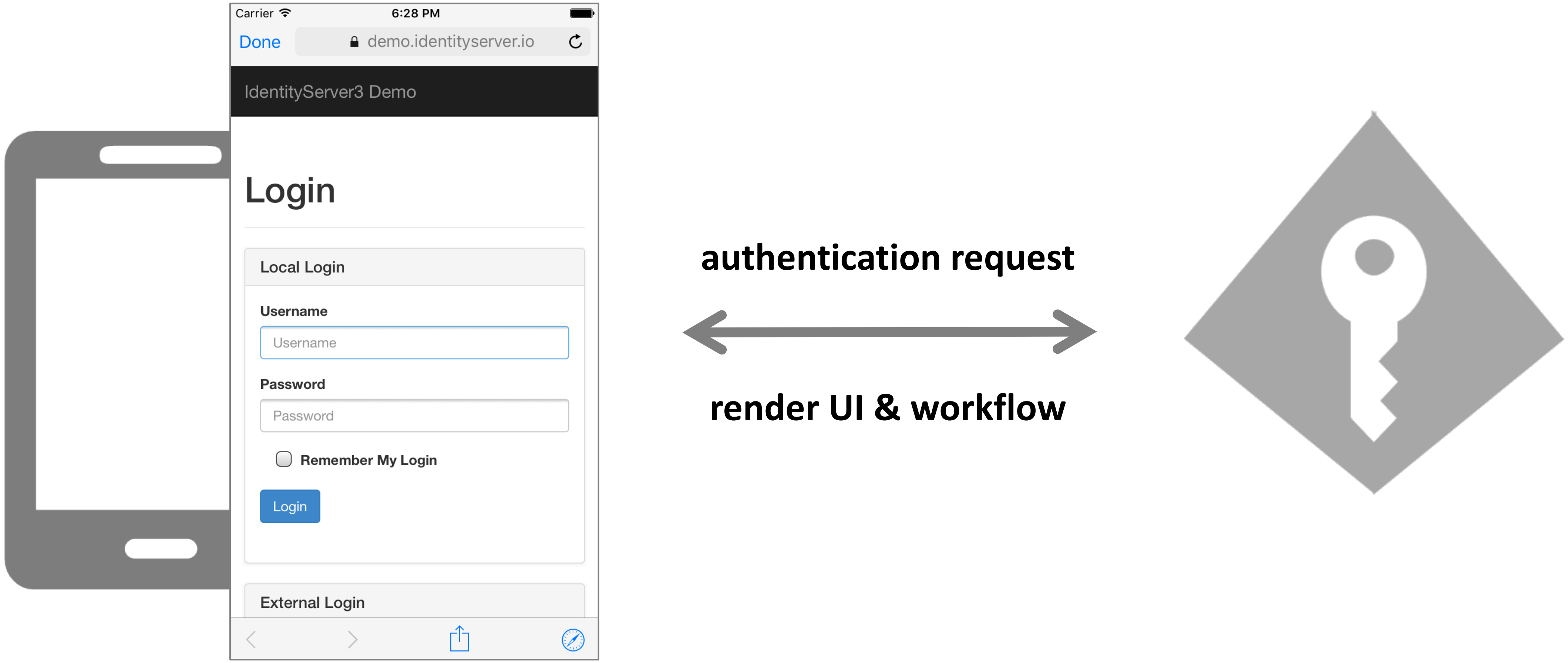


OAuth 2.0 Resource Owner Password Flow

- **Pros**
  - client app has full control over login UI
  - support for long lived API access without having to store a password

- **Cons**
  - user is encouraged to type in his master secret into "external" applications
    - especially problematic once applications also come from 3rd parties
  - no cross application single sign-on or shared logon sessions
  - no federation with external identity providers/business partners
  - every change in logon workflow requires versioning the application

Using a browser for driving the authentication workflow



Using a browser for driving the authentication workflow

- **Centralize authentication logic**
  - consistent look and feel
  - implement once, all applications get it for free
  - allows changing the workflow without having to update the applications
    - e.g. consent, updated EULA, 2FA

- **Enable external identity providers and federation**
  - federation protocols are browser based only

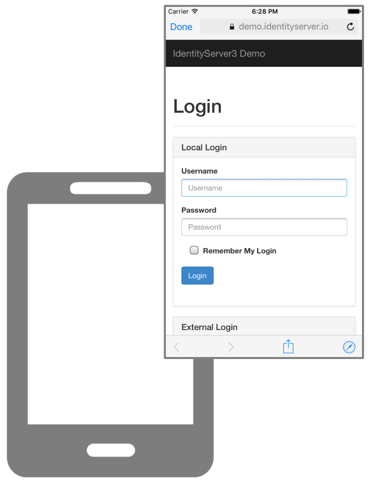
- **Depending on browser, authentication sessions can be shared between apps and OS**

Browser types

- **Embedded web view**
  - private browser & private cookie container
  - e.g. WinForms or WPF browser control
- **System browser**
  - e.g. SFAuthenticatedSession, Chrome Custom Tabs or desktop browser
  - full featured including address bar & add-ins
  - shared cookie container

Starting the authentication request

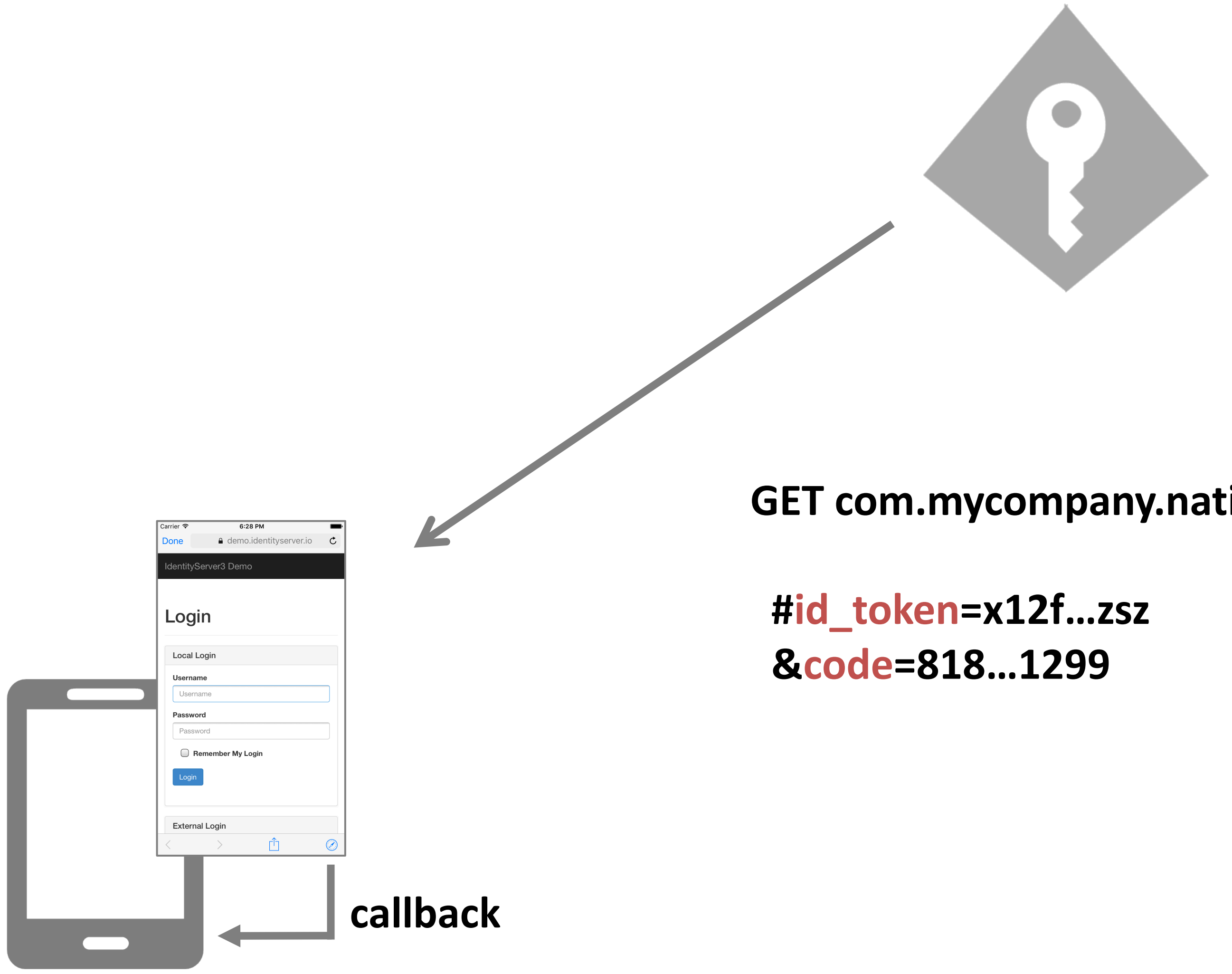
nonce = random_number
code_verifier = random_number
code_challenge = hash(code_verifier)



GET /authorize

?client_id=nativeapp
&scope=openid profile api1 api2 *offline_access*
&redirect_uri=com.mycompany.nativeapp://cb
&response_type=*code id_token*
&nonce=j1y…a23
&code_challenge=x929..1921

Receiving the response

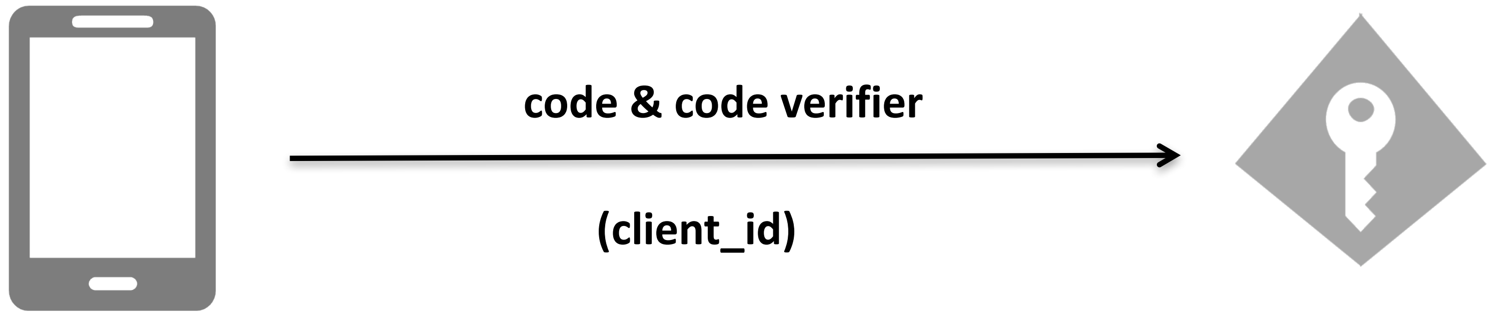


GET com.mycompany.nativeapp://cb

#id_token=x12f…zsz
&code=818…1299

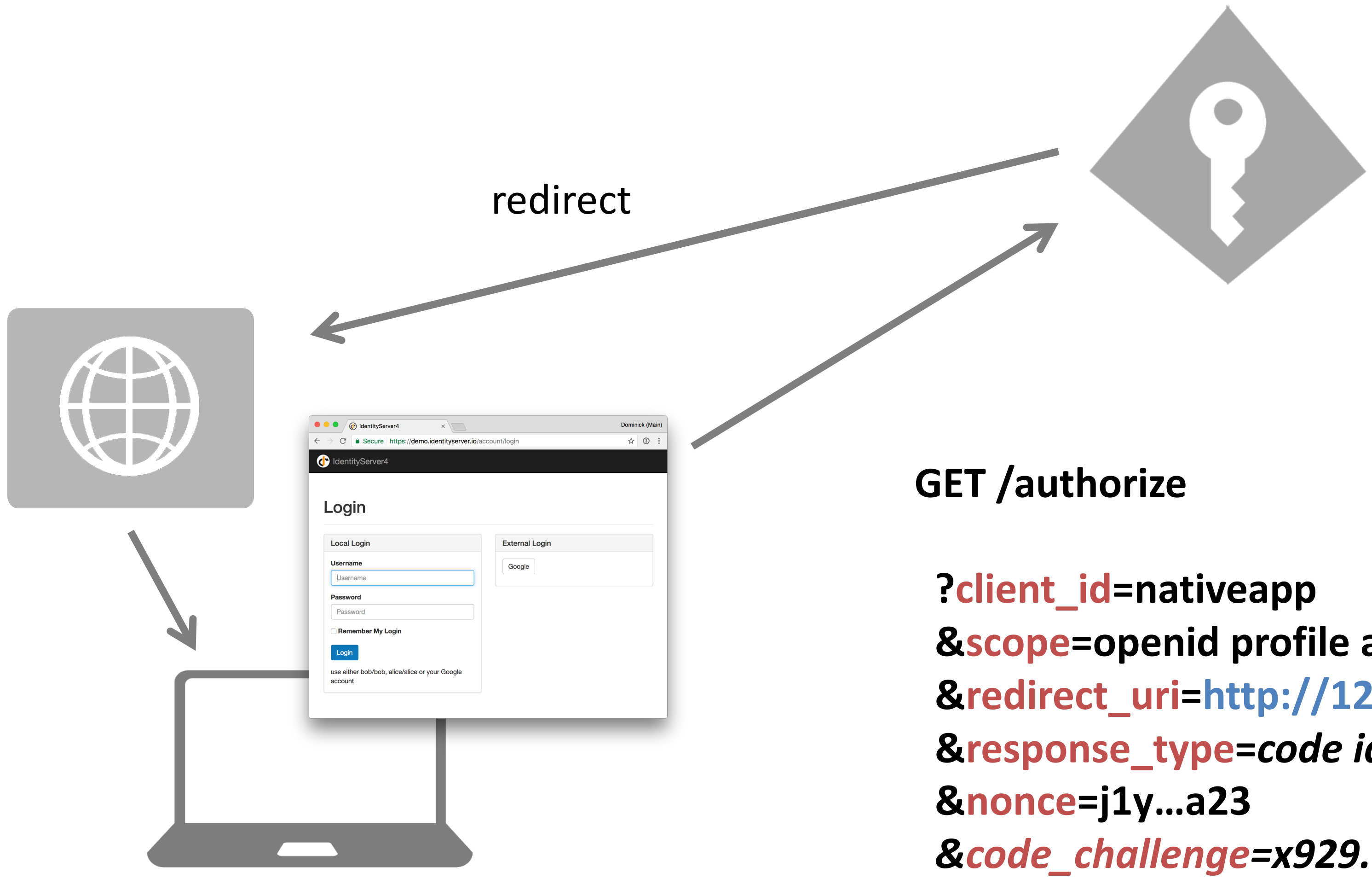
Requesting the access token

- **Exchange code for access token**
  - using client id and secret (and code verifier)



```
{
  access_token: "xyz…123",
  refresh_token: "dxy…103"
  expires_in: 3600,
  token_type: "Bearer"
}
```

Pattern: desktop browser and local callback



GET /authorize

?client_id=nativeapp
&scope=openid profile api1 api2 *offline_access*
&redirect_uri=http://127.0.0.1:some_port
&response_type=*code id_token*
&nonce=j1y…a23
&code_challenge=x929..1921

That's a lot of work!

- **Native libraries**
  - https://github.com/openid/AppAuth-iOS
  - https://github.com/openid/AppAuth-Android

- **C# .NET standard library (desktop .NET, UWP, mobile, iOS, Android)**
  - https://github.com/IdentityModel/IdentityModel.OidcClient2
  - https://github.com/IdentityModel/IdentityModel.OidcClient.Samples

identity MODEL

Summary

- **Client Credentials Flow**
  - server to server communication
  - no user identity in access token
- **Hybrid Flow**
  - web applications
  - access token contains user identity
  - access token not exposed to browser
  - refresh token
- **Hybrid Flow + PKCE**
  - native applications
  - hardened for system browser IPC
- **Implicit Flow**
  - JavaScript applications
  - no refresh tokens (silent renew as alternative)