# OAuth and OpenID Connect

(IN PLAIN ENGLISH)

MICAH SILVERMAN
@AFITNERD
@OKTADEV

NATE BARBETTINI

@NBARBETTINI
@OKTADEV

# Identity use cases (circa 2007)

- Simple login – forms and cookies

- Single sign-on across sites – SAML

- Mobile app login – ???

- Delegated authorization – ???

# Don't do it this way!



Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service  ○ msn Hotmail  ○ YAHOO! MAIL  ○ AOL Mail  ● Gmail

Your Email Address  ima.testguy@gmail.com  (e.g. bob@gmail.com)

Your Gmail Password  ••••••••••  (The password you use to log into your Gmail email)

Skip this step  **Check Contacts**

# Don't do it this way!

# The delegated authorization problem

HOW CAN I LET A WEBSITE ACCESS MY DATA
(WITHOUT GIVING IT MY PASSWORD)?

# A lot of confusion around OAuth & OIDC

- ✗ Terminology and jargon
- ✗ Incorrect advice
- ✗ To understand OpenID Connect, you need to learn first about OAuth
- ✗ Hard to find a life-like example

# OAuth – A real world example: Hotel Analogy

Hotel Manager

Delegates Access to **1** →

Repair Person

Obtains Access (Hotel Key) **2** →

Hotel Receptionist

**3** → To fix specific rooms within the Hotel

Hotel Rooms

# OAuth: App Scenario



End User — Delegates Access to — **1** → Yelp — Obtains Access (Tokens) — **2** → Google

Yelp — **3** To access api → Contacts API {api}
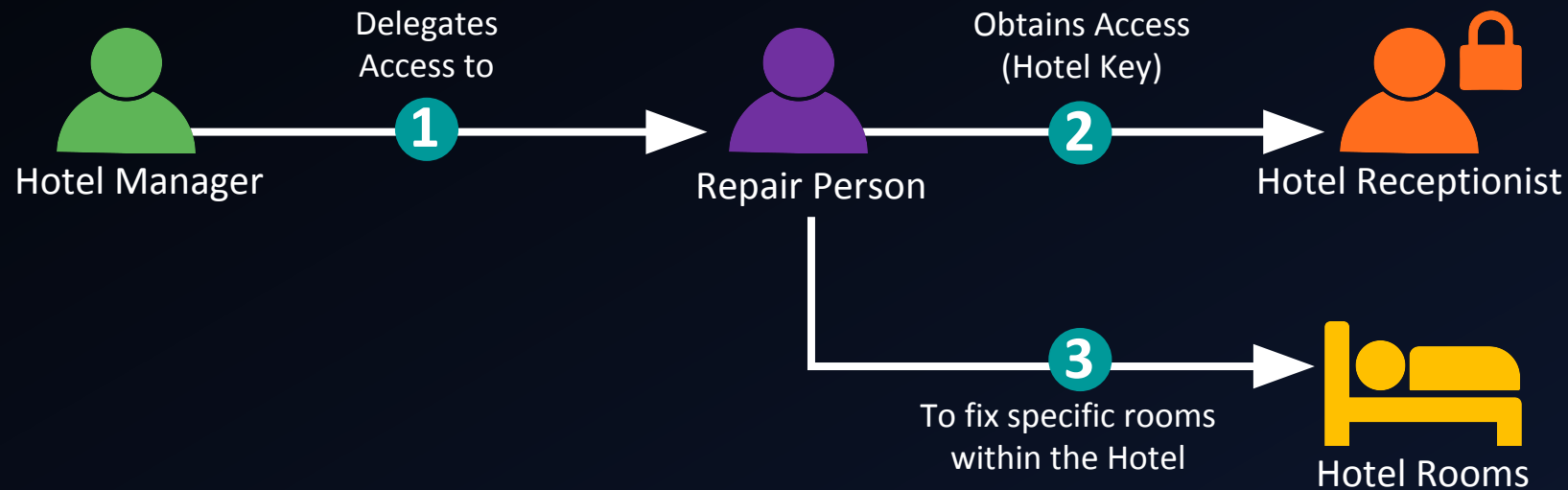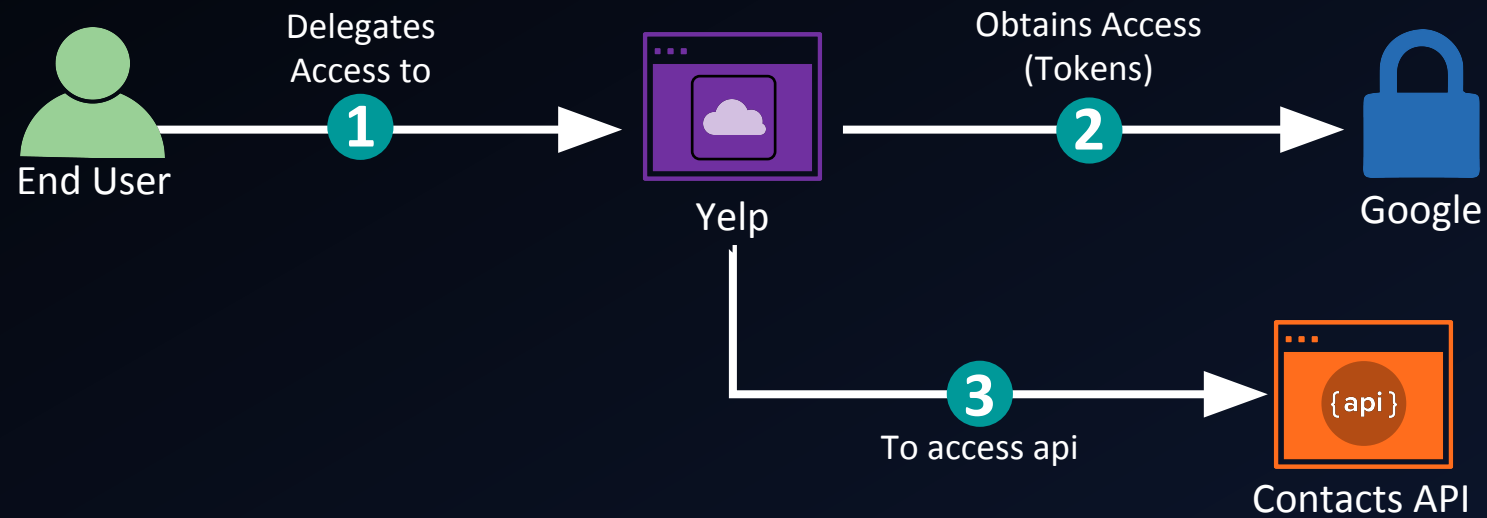
# OAuth 2.0 Terminology

| Hotel | App | OAuth | Description |
|---|---|---|---|
| Hotel Manager | End User | Resource Owner | Wants a Client App to do something on their behalf |
| Repair Person | Yelp | Client Application | Needs authorization to interact with an API on behalf of a user |
| Hotel Receptionist | Google | Authorization Server | Grants access (in the form of tokens) to an app |
| Hotel Rooms | Contacts API | Resource Server | Has an API that an app can use if presented with a token |

# Delegated authorization with OAuth 2.0

# OAuth 2.0 authorization code flow

**Client App**

**Authorization server**

yelp.com

[ Connect with Google ]

Resource owner

Go to authorization server

Redirect URI: `yelp.com/callback`
Response type: `code`

accounts.google.com

| Email |
| Password |

contacts.google.com

Talk to resource server
with access token

Exchange authorization code
for access token

yelp.com/callback

Loading…

Back to redirect URI
with authorization code

accounts.google.com

Allow Yelp to access your public
profile and contacts?

[ No ] [ Yes ]

# More OAuth 2.0 terminology

- Scope

- Consent

# OAuth 2.0 authorization code flow

**Client App**

yelp.com

[ Connect with Google ]

Resource owner

Go to authorization server

Redirect URI: `yelp.com/callback`
Response type: `code`
Scope: `profile contacts`

contacts.google.com

Talk to resource server
with access token

Exchange authorization code
for access token

yelp.com/callback

Loading…

Back to redirect URI
with authorization code

**Authorization server**

accounts.google.com

Email

Password

Request consent
from resource owner

accounts.google.com

Allow Yelp to access your public
profile and contacts?

[ No ]  [ Yes ]

# Even more OAuth 2.0 terminology

- Back channel (highly secure channel)
- Front channel (less secure channel)

# OAuth 2.0 authorization code flow

**Client App**

yelp.com

Connect with Google

Resource owner

Go to authorization server
(front channel)

Redirect URI: `yelp.com/callback`
Response type: `code`
Scope: `profile contacts`

contacts.google.com

Talk to resource server
with access token
(back channel)

Exchange authorization code
for access token (back channel)

yelp.com/callback

Loading…

**Authorization server**

accounts.google.com

Email
Password

Request consent
from resource owner

accounts.google.com

Allow Yelp to access your public
profile and contacts?

No    Yes

Back to redirect URI
with authorization code
(front channel)

# Starting the flow

```
https://accounts.google.com/o/oauth2/v2/auth?
  client_id=abc123&
  redirect_uri=https://yelp.com/callback&
  scope=profile&
  response_type=code&
  state=foobar
```

# Calling back

```
https://yelp.com/callback?
  error=access_denied&
  error_description=The user did not consent.
```

```
https://yelp.com/callback?
  code=oMsCeLvIaQm6bTrgtp7&
  state=foobar
```

# Exchange code for an access token

```
POST www.googleapis.com/oauth2/v4/token
Content-Type: application/x-www-form-urlencoded

code=oMsCeLvIaQm6bTrgtp7&
client_id=abc123&
client_secret=secret123&
grant_type=authorization_code
```

# Authorization server returns an access token

```json
{
  "access_token": "fFAGRNJru1FTz70BzhT3Zg",
  "expires_in": 3920,
  "token_type": "Bearer",
}
```

# Use the access token

```
GET api.google.com/some/endpoint
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg
```

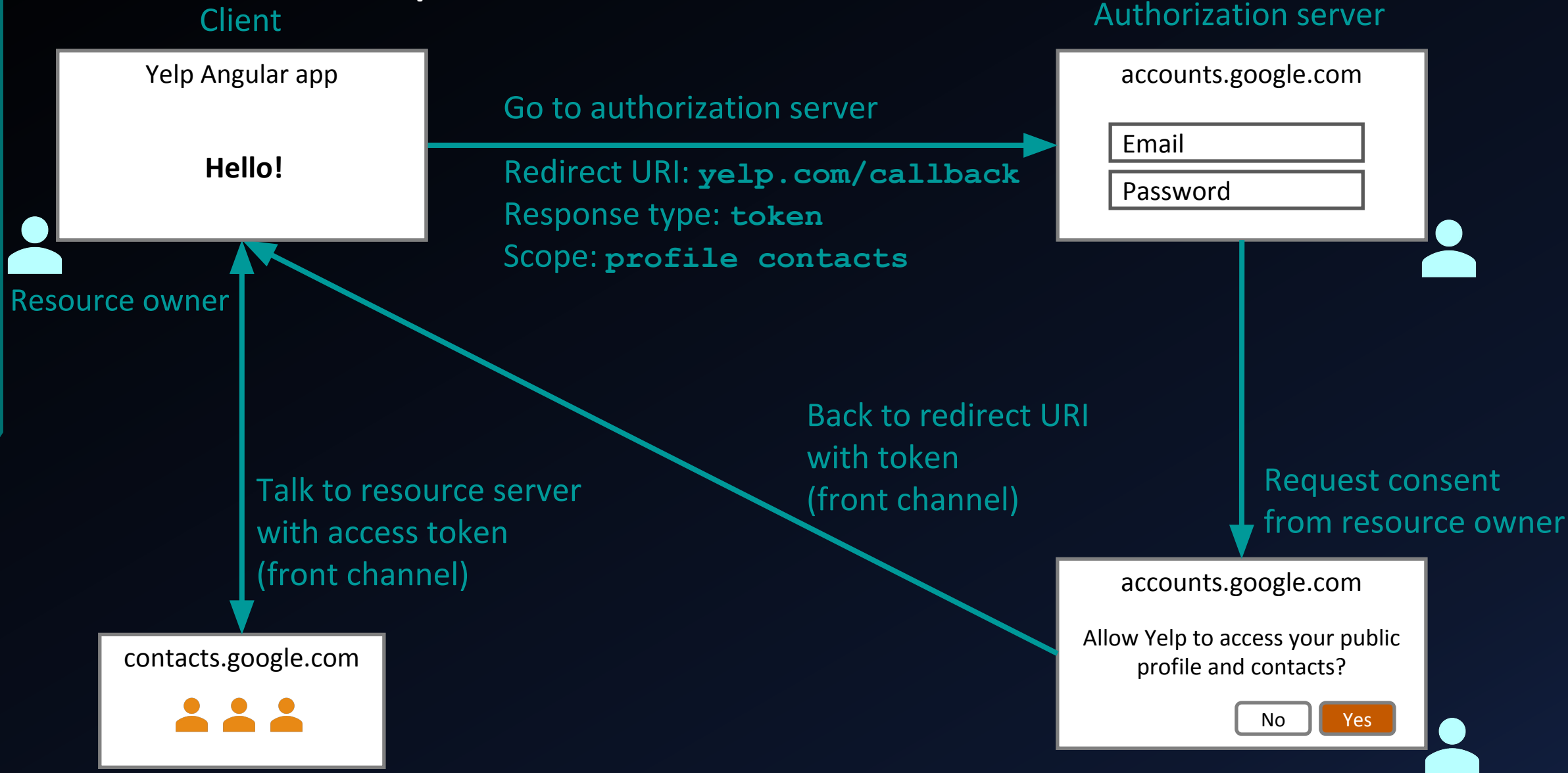# OAuth 2.0 flows

- Authorization code (front channel + back channel)

- Authorization code w/ PKCE (front channel + back channel)*

- Implicit (front channel only)

- Resource owner password (back channel only)

- Client credentials (back channel only)

# OAuth 2.0 implicit flow

**Client**

Yelp Angular app

**Hello!**

Resource owner

Go to authorization server

Redirect URI: `yelp.com/callback`
Response type: `token`
Scope: `profile contacts`

**Authorization server**

accounts.google.com

| Email |
| Password |

Talk to resource server
with access token
(front channel)

Back to redirect URI
with token
(front channel)

Request consent
from resource owner

contacts.google.com

accounts.google.com

Allow Yelp to access your public
profile and contacts?

No    Yes

# Identity use cases (circa 2012)

- Simple login – OAuth 2.0                        Authentication

- Single sign-on across sites – OAuth 2.0     Authentication

- Mobile app login – OAuth 2.0                Authentication
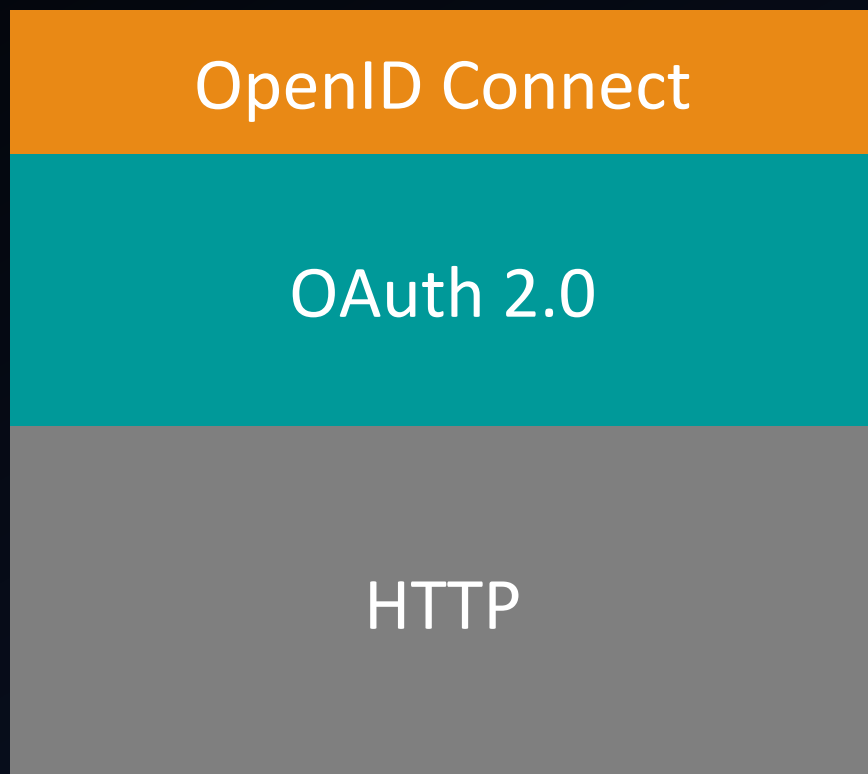
- Delegated authorization – OAuth 2.0      Authorization

# Problems with OAuth 2.0 for **authentication**

- No standard way to get the user's information

- Every implementation is a little different

- No common set of scopes

# OAuth 2.0 and OpenID Connect

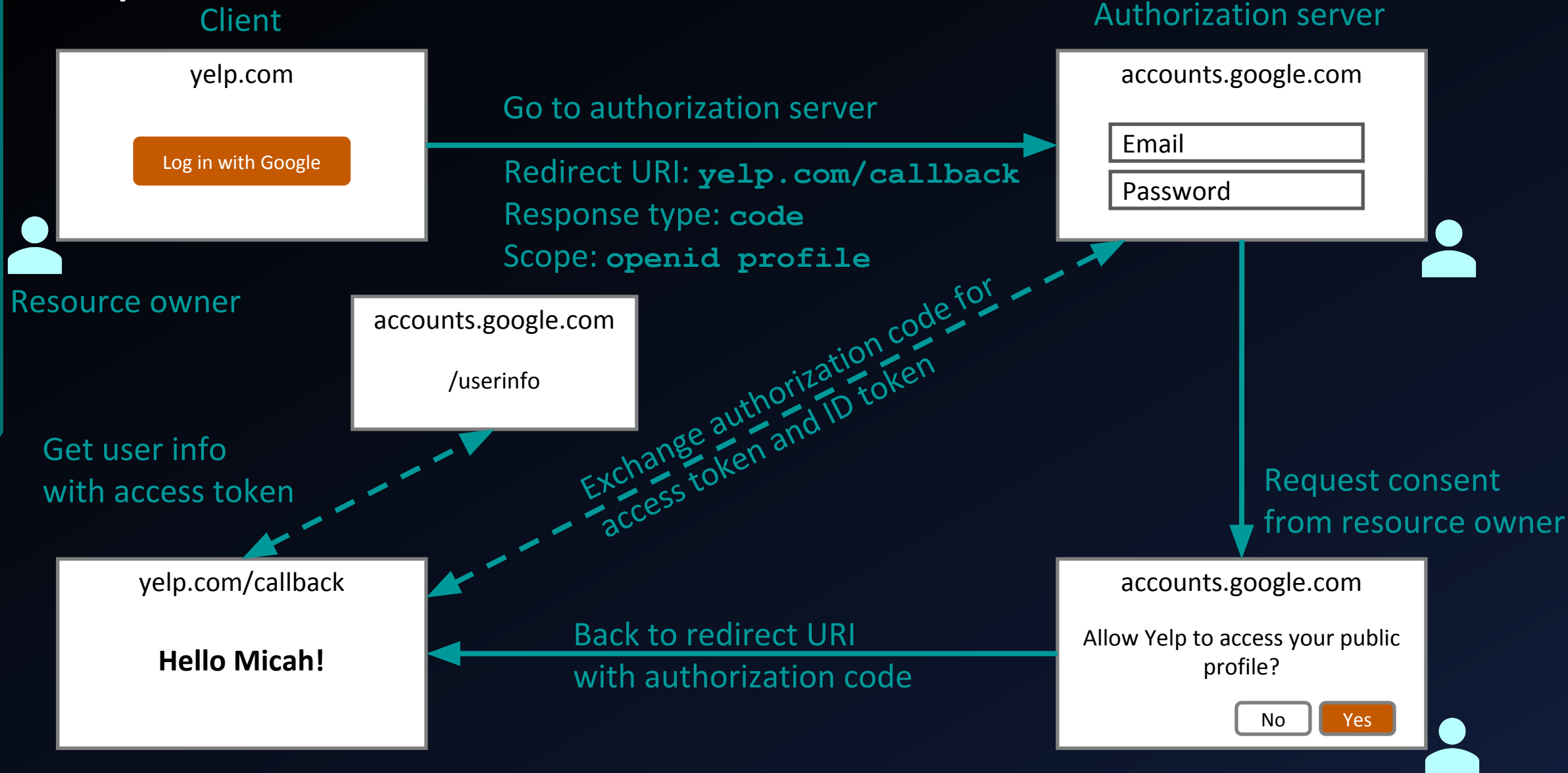| OpenID Connect |
|:---:|
| OAuth 2.0 |
| HTTP |

- OpenID Connect is for **authentication**

- OAuth 2.0 is for **authorization**

# What OpenID Connect adds

- ID token

- /userinfo endpoint for getting more user information

- Standard set of scopes

- Standardized implementation

# OpenID Connect authorization code flow

**Client**

**Authorization server**

yelp.com

Log in with Google

Resource owner

Go to authorization server

Redirect URI: `yelp.com/callback`
Response type: `code`
Scope: `openid profile`

accounts.google.com

Email

Password

accounts.google.com

/userinfo

Exchange authorization code for access token and ID token

Get user info
with access token

Request consent
from resource owner

yelp.com/callback

**Hello Micah!**

Back to redirect URI
with authorization code

accounts.google.com

Allow Yelp to access your public profile?

No    Yes

# Getting the discovery document

```
GET https://accounts.google.com/.well-known/openid-configuration

{
 "issuer": "https://accounts.google.com",
 "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
 "token_endpoint": "https://www.googleapis.com/oauth2/v4/token",
 "userinfo_endpoint": "https://www.googleapis.com/oauth2/v3/userinfo",
 "response_types_supported": [
    "code",
    "token",
    "id_token",
    "code token",
 ],
 ...
}
```

# Starting the flow

```
https://accounts.google.com/o/oauth2/v2/auth?
  client_id=abc123&
  redirect_uri=https://yelp.com/callback&
  scope=openid profile&
  response_type=code&
  state=foobar
```

# Calling back

```
https://yelp.com/callback?
  error=access_denied&
  error_description=The user did not consent.


https://yelp.com/callback?
  code=oMsCeLvIaQm6bTrgtp7&
  state=foobar
```

# Exchange code for access token **and** ID token

```
POST www.googleapis.com/oauth2/v4/token
Content-Type: application/x-www-form-urlencoded

code=oMsCeLvIaQm6bTrgtp7&
client_id=abc123&
client_secret=secret123&
grant_type=authorization_code
```

# Authorization server returns access and ID tokens

```json
{
  "access_token": "fFAGRNJru1FTz70BzhT3Zg",
  "id_token": "eyJraB03ds3F...",
  "expires_in": 3920,
  "token_type": "Bearer",
}
```

# ID token (JWT)

eyJhbGciOiJSUzI1NiIsImtpZCI6IkRNa3Itd0JqRU1EYnhOY25xaVJISVhu
YUxubWI3UUpfWF9rWmJyaEtBMGMifQ

.

eyJzdWIiOiIwMHU5bzFuaWtqdk9CZzVabzBoNyIsInZlciI6MSwiaXNzIjoi
aHR0cHM6Ly9kZXYtMzQxNjA3Lm9rdGFwcmV2aWV3LmNvbS9vYXV0aDIvYXVz
OW84d3ZraG9ja5VEwwaDciLCJhdWQiOiJsWFNlbkx4eFBpOGtRVmppKRTVz
NCIsImlhdCI6MTUwOTA0OTg5OCwiZXhwIjoxNTA5MDUzNDk4LCJqdGkiOiJJ
RC5oa2RXSXNBSXZTbnBGYVFHVSmhhMkgwcwS2c5Ykl3ZEVvVm1ZZHN3
IiwiYW1yIjpbImtiYSIsIm1mYSISInB3ZCJdLCJpZHAiOiIwMG85bzFuaWpr
aWpLeGNpjBoNyIsIm5vbmNlIjoidWpwwMmFzeHlqN2UiLCJhdXRoX3RpbWUi
OjE1MDkwNDk3MTl9

.

dv4Ek8B4BDee1PcQT_4zm7kxDEY1sRIGbLoNtlodZcSzHz-XU5GkKyl6sAVm
dXOIPUlAIrJAhNfQWQ-_XZLBVPjETiZE8CgNg5uqNmeXMUnYnQmvN5oWlXUZ
8Gcub-GAbJ8-NQuyBmyec1j3gmGzX3wemke8NkuI6SX2L4Wj1PyvkknBtbjf
iF9ud1-ERKbobaFbnjDFOFTzvL6g34SpMmZWy6uc_Hs--n4IC-ex-_Ps3FcM
wRggCW_-7o2FpH6rJTOGPZYrOx44n3ZwAu2dGm6axtPI-sqU8b6sw7DaHpog
D_hxsXgMIOzOBMbYsQEiczoGn71ZFz_1O7FiW4dH6g

Header

Payload
(claims)

Signature

# The ID token (JWT)

```
(Header)
.
{
    "iss": "https://accounts.google.com",
    "sub": "micah.silverman@okta.com",
    "name": "Micah Silverman"
    "aud": "s6BhdRkqt3",
    "exp": 1311281970,
    "iat": 1311280970,
    "auth_time": 1311280969,
}
.
(Signature)
```

# Calling the userinfo endpoint

```
GET www.googleapis.com/oauth2/v4/userinfo
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg


200 OK
Content-Type: application/json


{
    "sub": "you@gmail.com",
    "name": "Nate Barbettini"
    "profile_picture": "http://plus.g.co/123"
}
```

# Identity use cases (today)

- Simple login – OpenID Connect                          Authentication

- Single sign-on across sites – OpenID Connect           Authentication

- Mobile app login – OpenID Connect                      Authentication

- Delegated authorization – OAuth 2.0                    Authorization

# OAuth and OpenID Connect

Use OAuth 2.0 for:

- Granting access to your API

- Getting access to user data in other systems
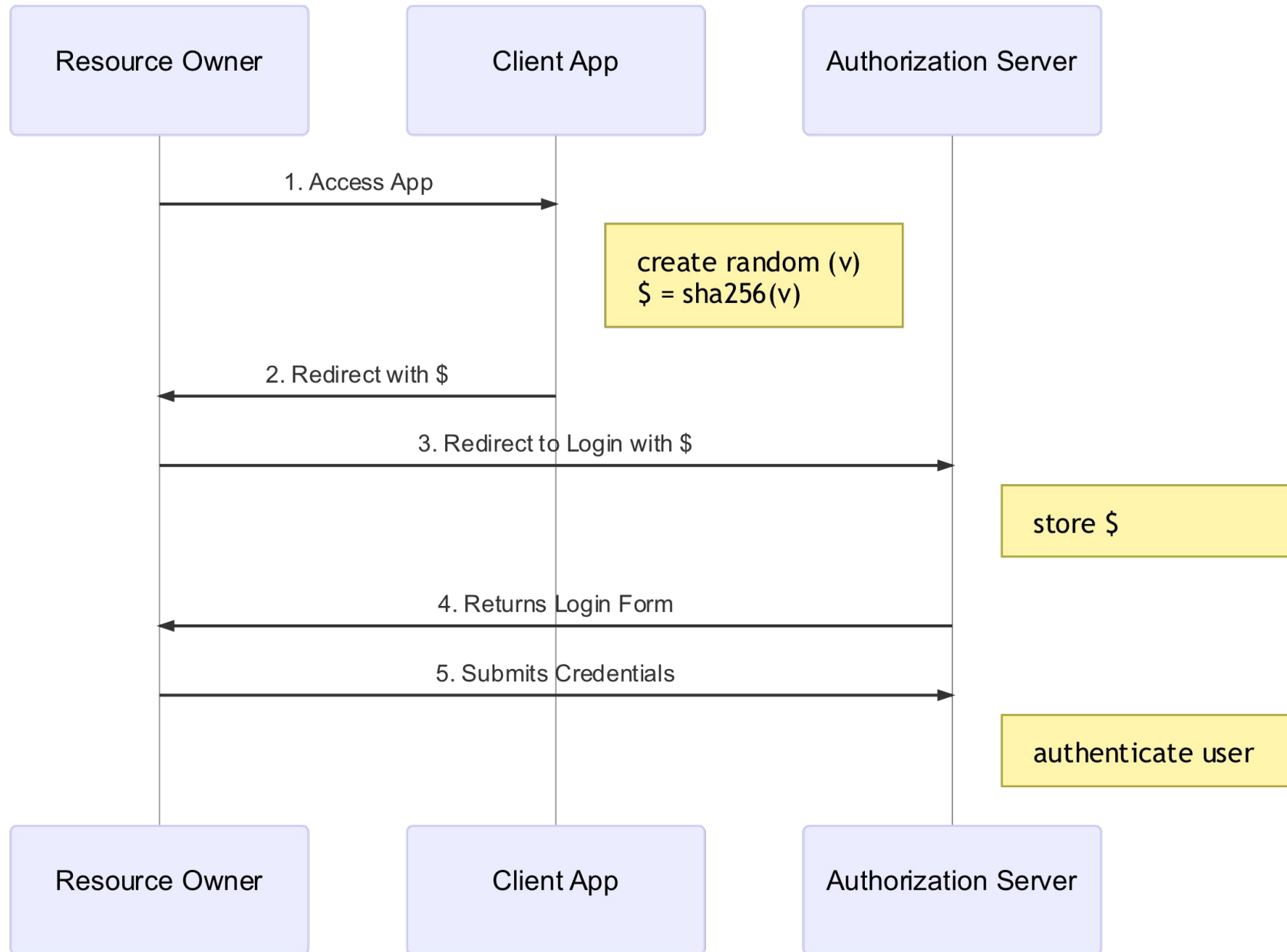
(Authorization)

Use OpenID Connect for:

- Logging the user in

- Making your accounts available in other systems
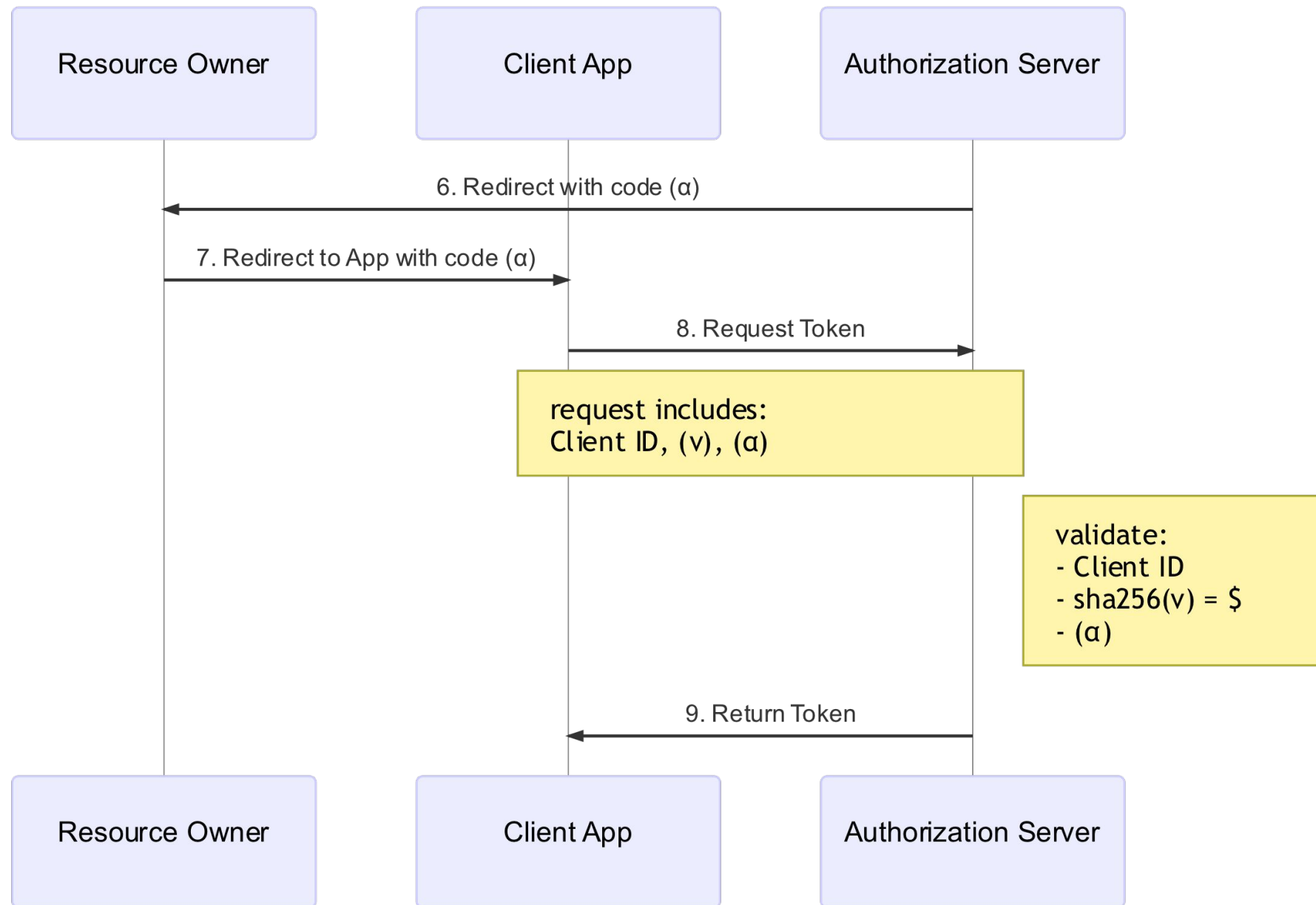
(Authentication)

# Which flow (grant type) do I use?

- Web application w/ server backend: **authorization code flow**

- Native or mobile app: **authorization code with PKCE flow**

- JavaScript app (SPA): **authorization code with PKCE flow**

- Microservices and APIs: **client credentials flow**

# Authorization Code with PKCE Flow (Part 1)

# Authorization Code with PKCE Flow (Part 2)

Resource Owner

Client App

Authorization Server

6. Redirect with code (α)

7. Redirect to App with code (α)

8. Request Token

request includes:
Client ID, (v), (α)

validate:
- Client ID
- sha256(v) = $
- (α)

9. Return Token

Resource Owner

Client App

Authorization Server

# Token validation

- The fast way: local validation
  - Validate cryptographic signature
  - Check expiration timestamp
- The strong way: introspection

# Thanks y'all!

github.com/dogeared/coder-cruise-2019-oauth-and-oidc

Micah Silverman

@afitnerd

oauth.com

@oktadev


Free hosted authorization server:

**developer.okta.com**