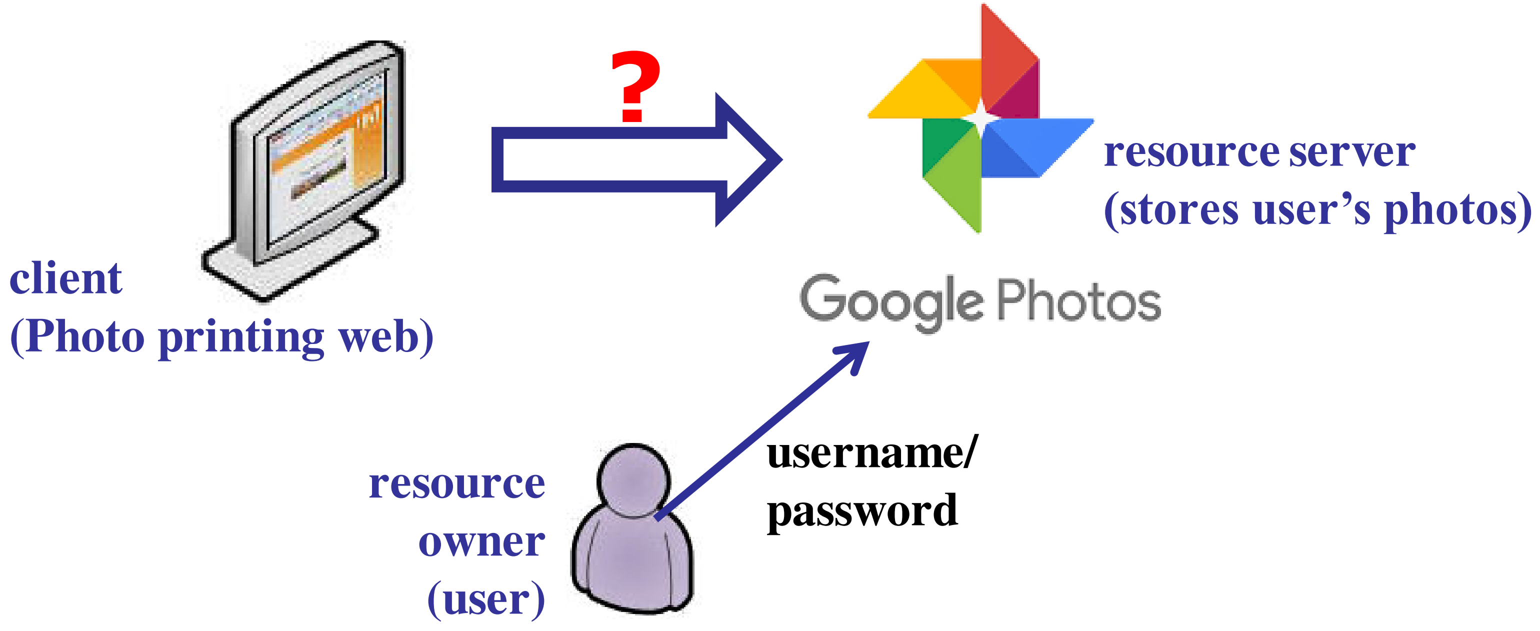
6. OAuth (an open standard to authorization) + OpenID Connect (OIDC)

some pictures from Ping Identity,
IETF RFC 6749,
N. Barbettini@OCTADEV

Motivating example

- Interaction in web
  - A photo printing service website prints user's photos stored at another website.



  - User authorizes the photo printing website to access her photos

APIs for web services

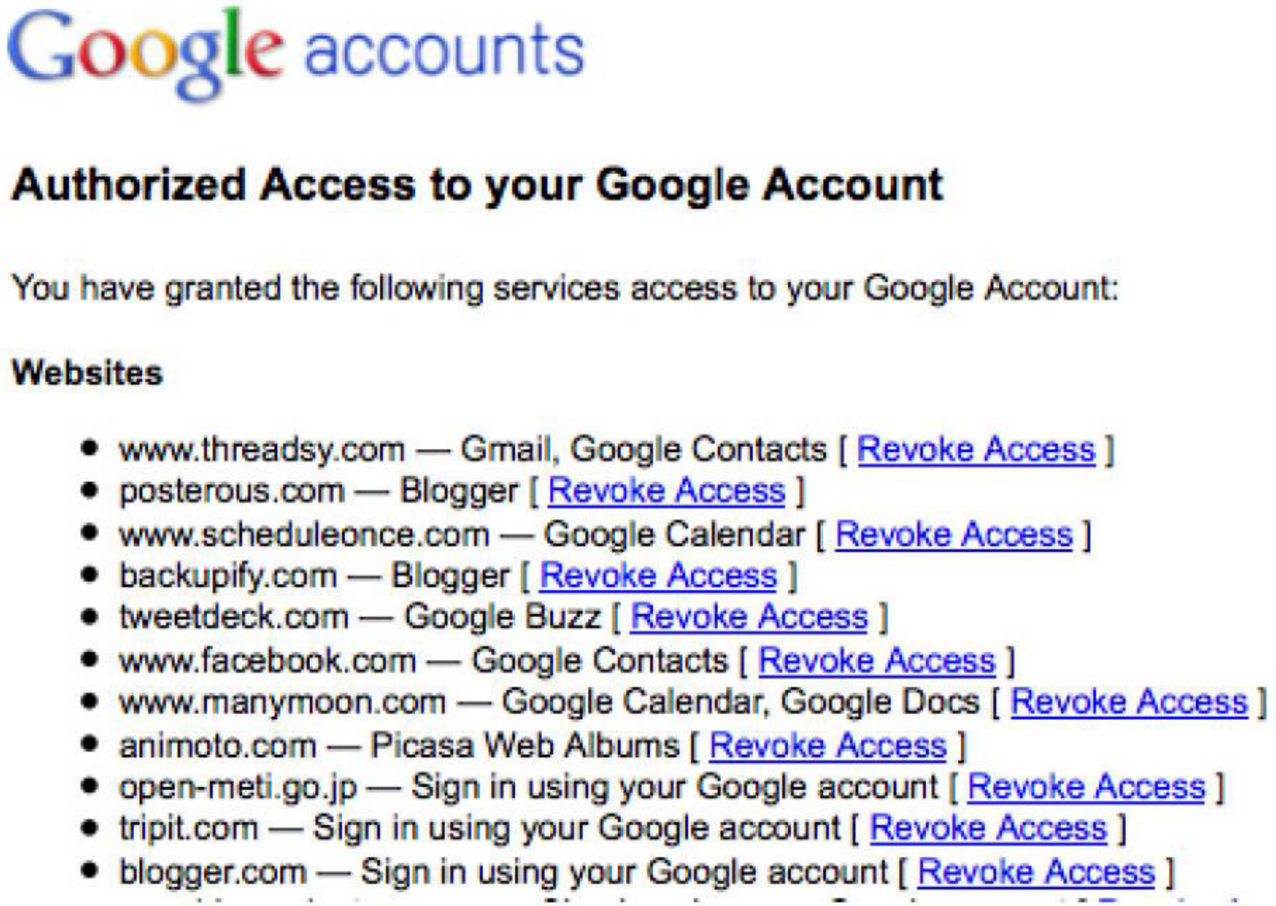
- Three party authentication
  - resource server: photo storage (google photos)
  - Client: photo printing service website
  - User: resource owner
- A method for the user to grant client access to the data stored at resource server
- Through the API defined by resource server
  - e.g. Amazon Web Services API
- OAuth 2.0 authorization framework enables a third-party application to obtain limited access to owner’s resources by the resource server’s APIs

Sharing user credential?

- if users are indiscriminate with distributing their passwords
- resource server is not involved in authorization; it cannot know which client the user has approved
- Sharing password with client presents a risk of breach
- It doesn’t support granular permissions
- It doesn’t support (easy) revocation

OAuth status: example

- Allows the user to delegate to requesting site (or client) the desired permissions



Google accounts

Authorized Access to your Google Account

You have granted the following services access to your Google Account:

Websites

- www.threadsy.com — Gmail, Google Contacts [Revoke Access]
- posterous.com — Blogger [Revoke Access]
- www.scheduleonce.com — Google Calendar [Revoke Access]
- backupify.com — Blogger [Revoke Access]
- tweetdeck.com — Google Buzz [Revoke Access]
- www.facebook.com — Google Contacts [Revoke Access]
- www.manymoon.com — Google Calendar, Google Docs [Revoke Access]
- animoto.com — Picasa Web Albums [Revoke Access]
- open-meti.go.jp — Sign in using your Google account [Revoke Access]
- tripit.com — Sign in using your Google account [Revoke Access]
- blogger.com — Sign in using your Google account [Revoke Access]

Terminology (1/2)

- Resource Owner
  - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user
- Client
  - An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices
- Resource Server (RS)
  - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens
- authorization server
  - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Terminology (2/2)

- Access token
  - Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. Tokens represent specific scopes and durations of access.
- Refresh Token
  - Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner).

Basic OAuth Flow

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: Abstract Protocol Flow

(A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary.

(B) The client receives an authorization grant, which is a credential representing the resource owner's authorization. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.

(C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.

(F) The resource server validates the access token, and if valid, serves the request

Authorization grant

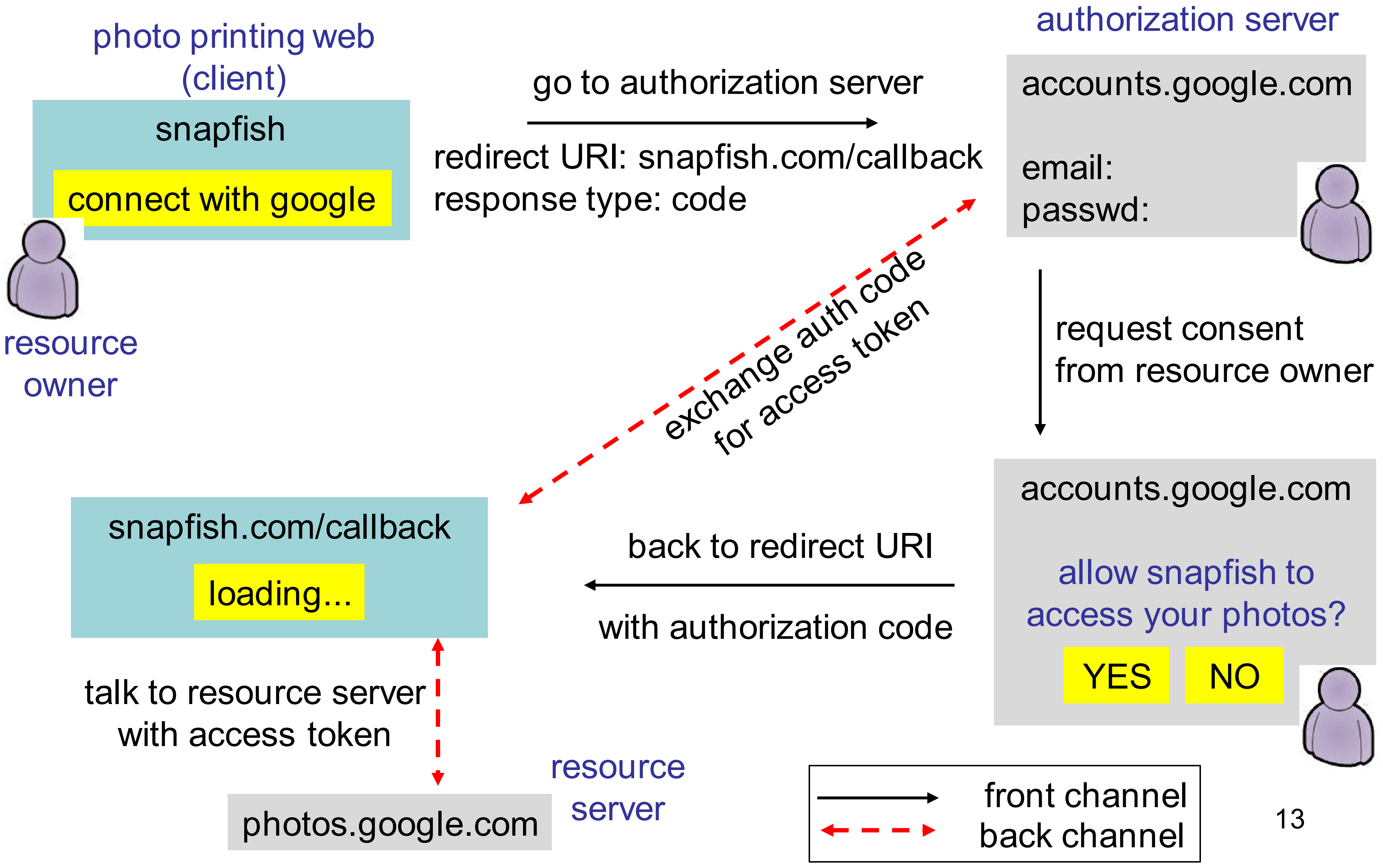
- a credential representing the resource owner’s authorization (to access its protected resources)
- 4 types
  1) authorization code (front + back channels)
  2) implicit (front channel)
  3) resource owner password credentials (back channel)
  4) client credentials (back channel)

* front channel: may be vulnerable
* back channel; secure like TLS

(1) Authorization code

- the client directs the resource owner to an authorization server (via its user-agent, e.g. browser), which in turn directs the resource owner back to the client with the authorization code
- Because the resource owner only authenticates with the authorization server, the resource owner's credentials are never shared with the client.
- The authorization code provides a few important security benefits, such as the ability to authenticate the client, as well as the transmission of the access token directly to the client without passing it through the resource owner's user-agent and potentially exposing it to others, including the resource owner.

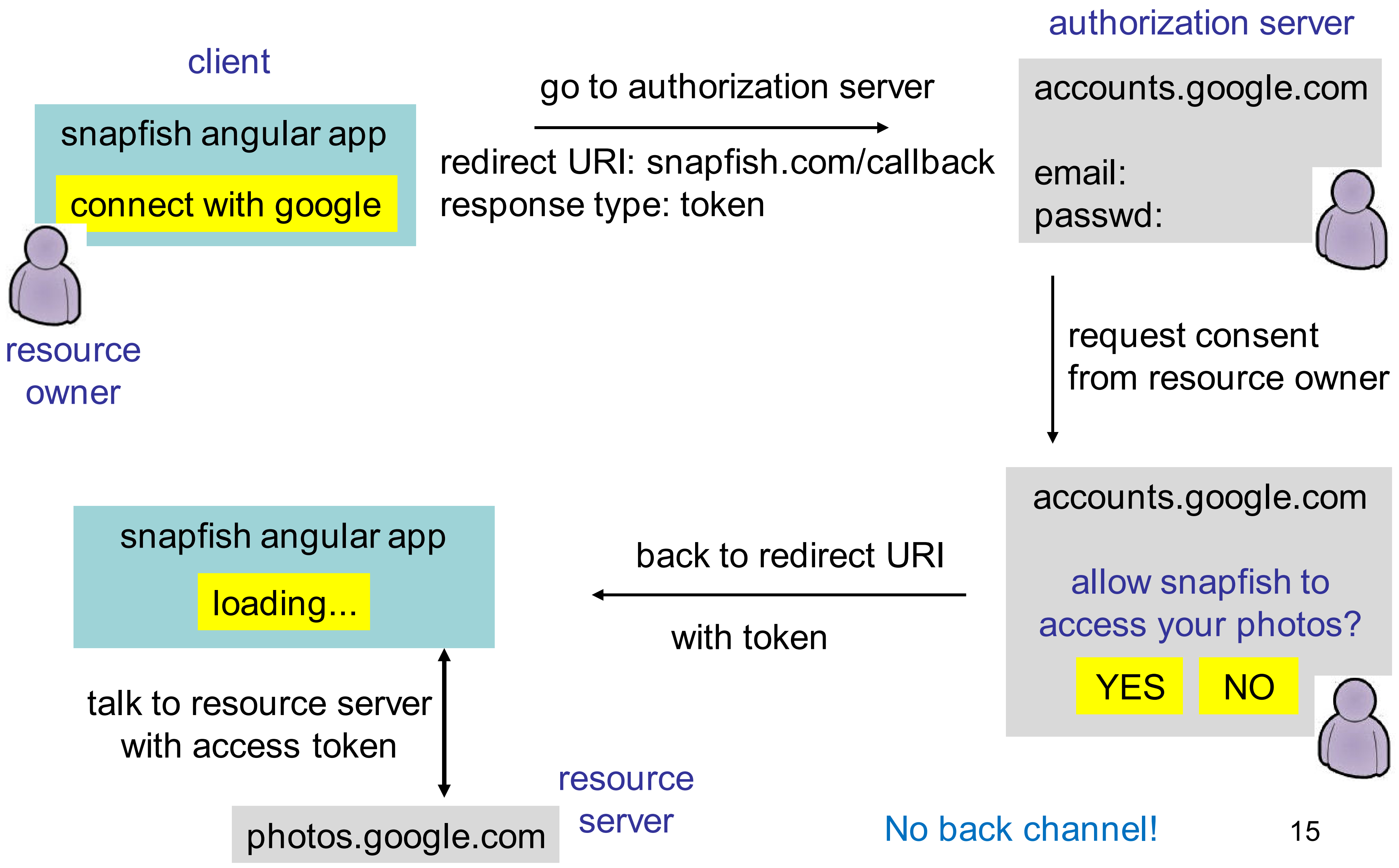
(1) Auth code to access token



(2) implicit

- The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly
- no intermediate credentials (such as an authorization code) are issued
- The authorization server does not authenticate the client. In some cases, the client identity can be verified via the redirection URI used to deliver the access token to the client. The access token may be exposed to the resource owner.

(2) Implicit flow



No back channel!

(3) Resource owner pwd credentials

- The resource owner password credentials (i.e., username and password) can be used directly as an authorization grant. The credentials should only be used when there is a high degree of trust between the resource owner and the client (e.g., the client is part of the device operating system)
- This grant type can eliminate the need for the client to store the resource owner credentials for future use, by exchanging the credentials with a long-lived access token or refresh token.

(4) Client credentials

- The client credentials (or other forms of client authentication) can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server.
- Client credentials are used as an authorization grant typically when the client is acting on its own behalf (the client is also the resource owner) or is requesting access to protected resources based on an authorization previously arranged with the authorization server.

Access token

- a string representing an authorization issued to the client
  - usually opaque to the client
- an identifier used to retrieve the authorization information or may self-contain the authorization information in a verifiable manner
  - E.g. a token string consisting of some data and a signature
- It removes the resource server’s need to understand a wide range of authentication methods

Refresh token (optional)

- issued to the client by the authorization server
- used to obtain a new access token when the current access token becomes invalid or expires
- If the authorization server issues a refresh token, it is included when issuing an access token, i.e., step (D)
- a string representing the authorization granted to the client by the resource owner
- refresh tokens are intended for use only with authorization servers and are never sent to resource servers

Refreshing tokens

```
+--------+                                           +---------------+
|        |--(A)------- Authorization Grant --------->|               |
|        |                                           |               |
|        |<-(B)----------- Access Token -------------|               |
|        |               & Refresh Token             |               |
|        |                                           |               |
|        |                            +----------+   |               |
|        |--(C)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(D)- Protected Resource --| Resource |   | Authorization |
| Client |                            |  Server  |   |     Server    |
|        |--(E)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(F)- Invalid Token Error -|          |   |               |
|        |                            +----------+   |               |
|        |                                           |               |
|        |--(G)----------- Refresh Token ----------->|               |
|        |                                           |               |
|        |<-(H)----------- Access Token -------------|               |
+--------+           & Optional Refresh Token        +---------------+

             Figure 2: Refreshing an Expired Access Token
```

Client registration

- Before initiating OAuth, the client registers with the authorization server
  - Client identifier
- In registration, the client should
  - Specify redirection URI, client type,…
- client may set up a secret with authorization server in advance

client types

- Confidential
  - Clients can maintain the confidentiality of their credentials
    - E.g. running on a secure server
- Public
  - Clients cannot maintain the confidentiality of their credentials
    - E.g. clients running on the device by the resource owner

Client profiles

- Web application
  - a confidential client running on a web server
  - Client credentials are not accessibly by the resource owner
- user-agent based application
  - A public client which is running in a browser of the resource owner
  - Credentials are accessible by resource owner
- Native applications
  - a public client installed and executed on the device used by the resource owner
  - Dynamically issued credentials (access/refresh tokens) are protected from other servers

Another Drawing of case (1): Authorization code

```
     +----------+
     | Resource |
     |   Owner  |
     |          |
     +----------+
          ^
          |
         (B)
     +----|-----+          Client Identifier      +---------------+
     |         -+----(A)-- & Redirection URI ---->|               |
     |  User-   |                                 | Authorization |
     |  Agent  -+----(B)-- User authenticates --->|     Server    |
     |          |                                 |               |
     |         -+----(C)-- Authorization Code ---<|               |
     +-|----|---+                                 +---------------+
       |    |                                         ^      v
      (A)  (C)                                        |      |
       |    |                                         |      |
       ^    v                                         |      |
     +---------+                                      |      |
     |         |>---(D)-- Authorization Code ---------'      |
     |  Client |          & Redirection URI                  |
     |         |                                             |
     |         |<---(E)----- Access Token -------------------'
     +---------+       (w/ Optional Refresh Token)
```

Note: The lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent.

```
(A) GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
        &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
    Host: server.example.com
```

```
(C) HTTP/1.1 302 Found
        Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA
        &state=xyz
```

```
(D) POST /token HTTP/1.1
    Host: server.example.com
    Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
    Content-Type: application/x-www-form-urlencoded
        grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
        &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

```
(E) HTTP/1.1 200 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache
    {"access_token":"2YotnFZFEjr1zCsicMWpAA",
     "token_type":"example",
     "expires_in":3600,
     "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
     "example_parameter":"example_value“}
```

OpenID Connect (OIDC)

identity use cases

- single sign-on across sites (Oauth 2.0)
- mobile app login (Oauth 2.0)
- delegated authorization (Oauth 2.0)

identity use cases

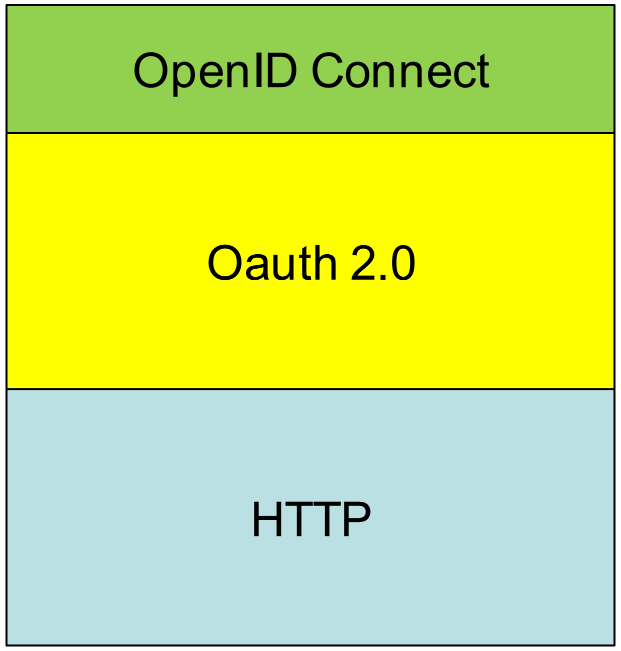
- single sign-on across sites (Oauth 2.0): authentication
- mobile app login (Oauth 2.0): authentication
- delegated authorization (Oauth 2.0): authorization

problems with Oauth 2.0 for authentication

- no standard way to get the user’s info
- every implementation is a little different
- no common set of scopes

OpenID Connect (OIDC)

- add a thin layer on top of Oauth 2.0

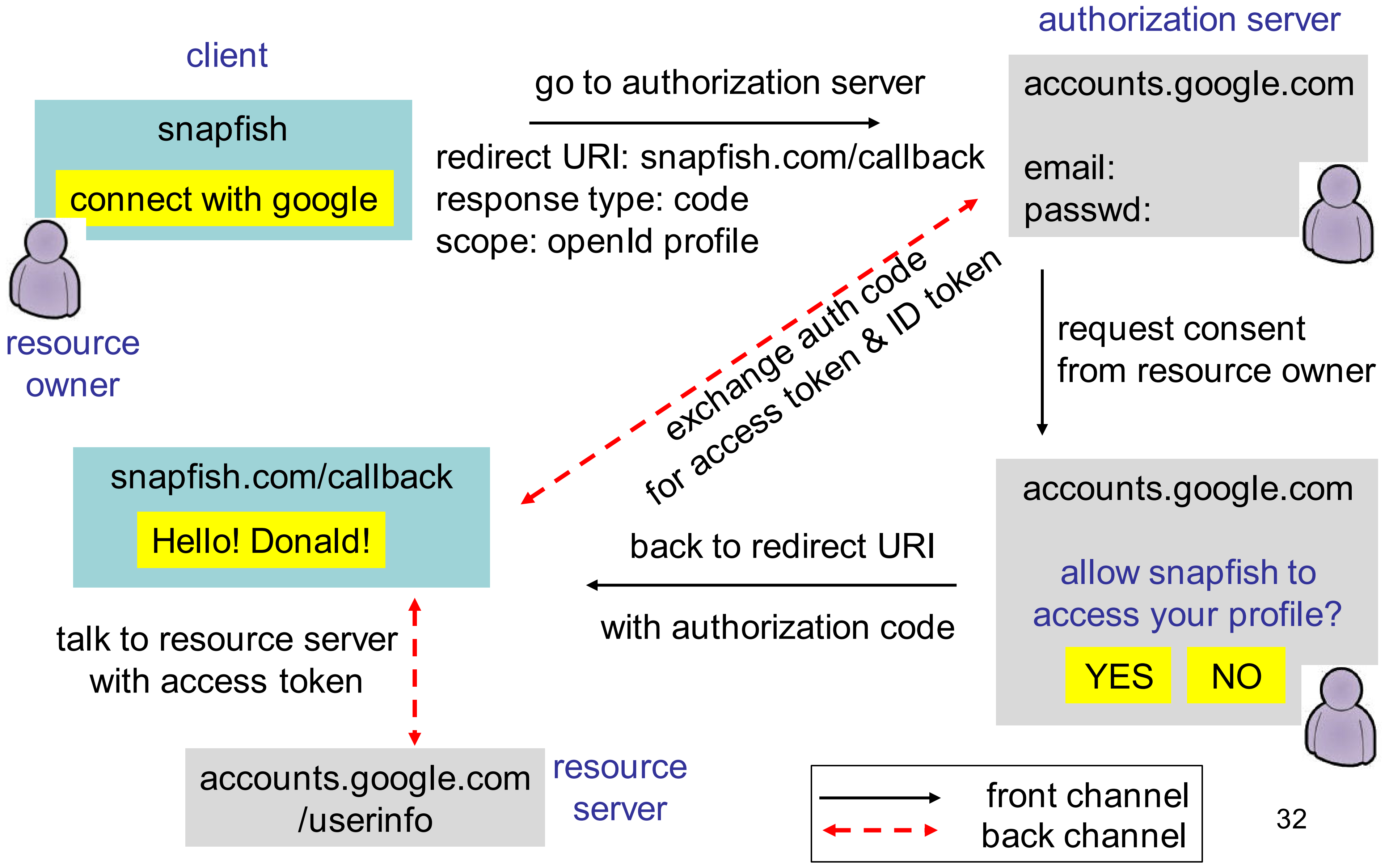


- OIDC for authentication
- Oauth 2.0 for authorization

what OIDC adds are

- ID token
  - issuer, subject, time, expiration,...
  - name, email, DoB,...
  - signature is added
- UserInfo Endpoint
  - URL that, when presented with an Access Token by the Client, returns authorized information about the End-User
  - over https
- standard set of scopes
  - profile, email, addrss, phone,...
- standardized implementation

OIDC authorization code flow



Oauth and OpenID Connect

use Oauth 2.0 for

- granting access to your API
- getting access to user data in other systems

(authorization)

use OIDC for

- logging the user in
- making your account available in other systems

(authentication)