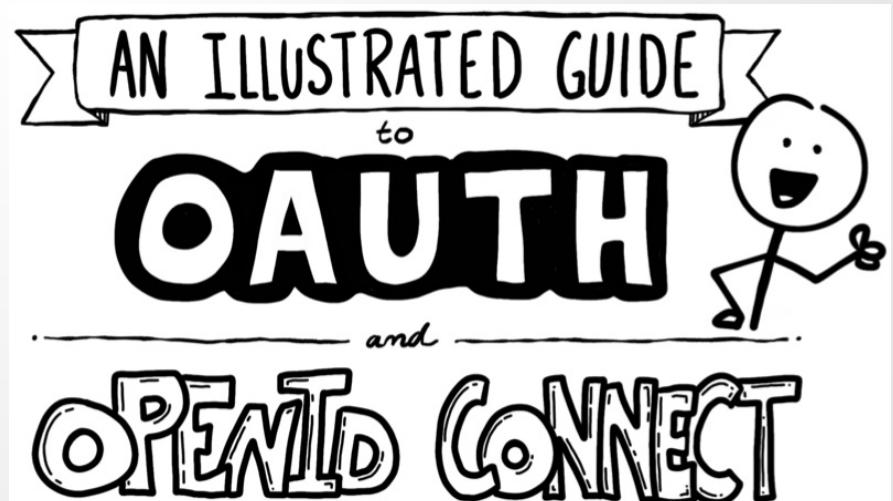


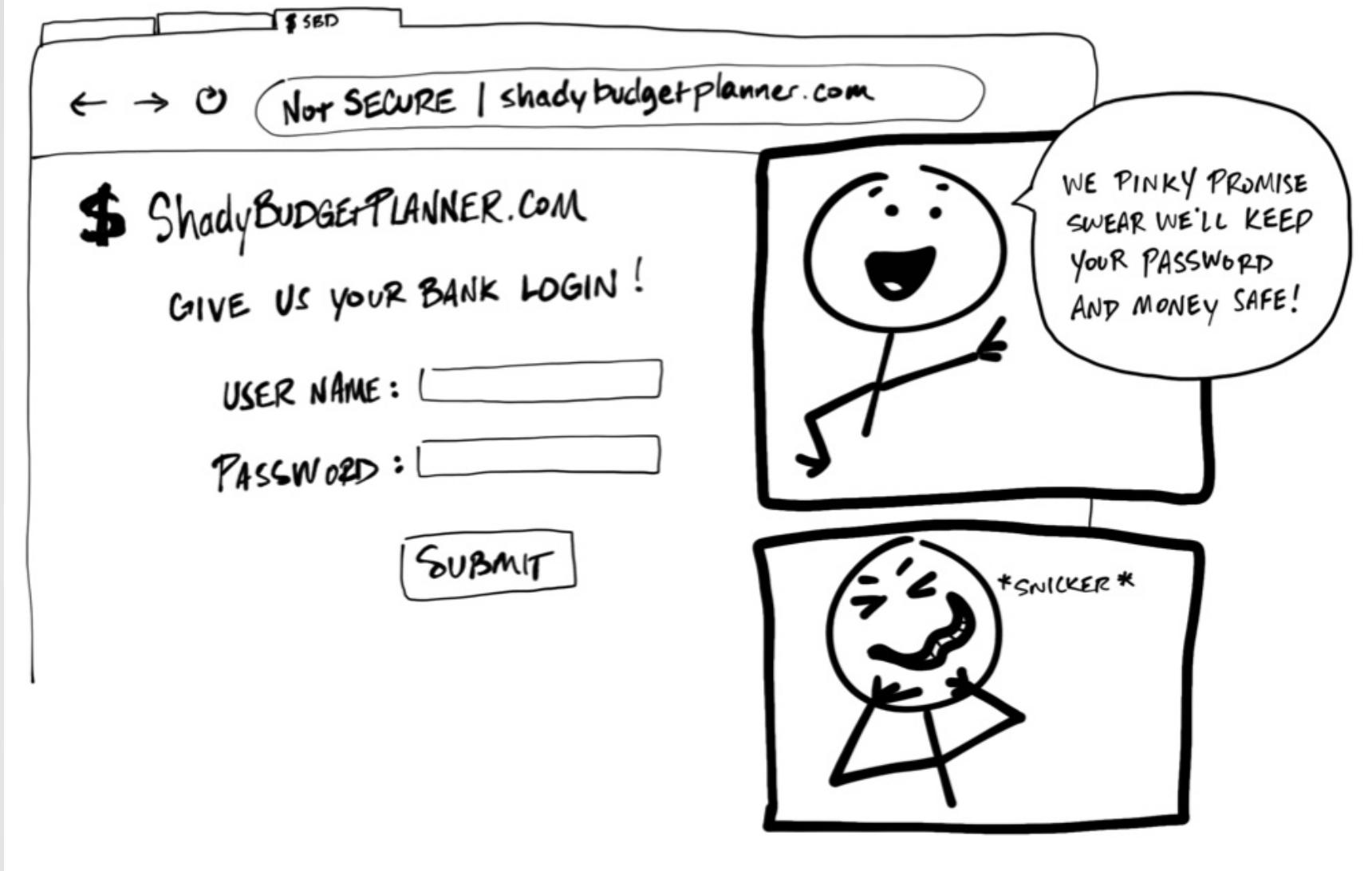
# OAuth 2.0

& OpenID Connect



<https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc>

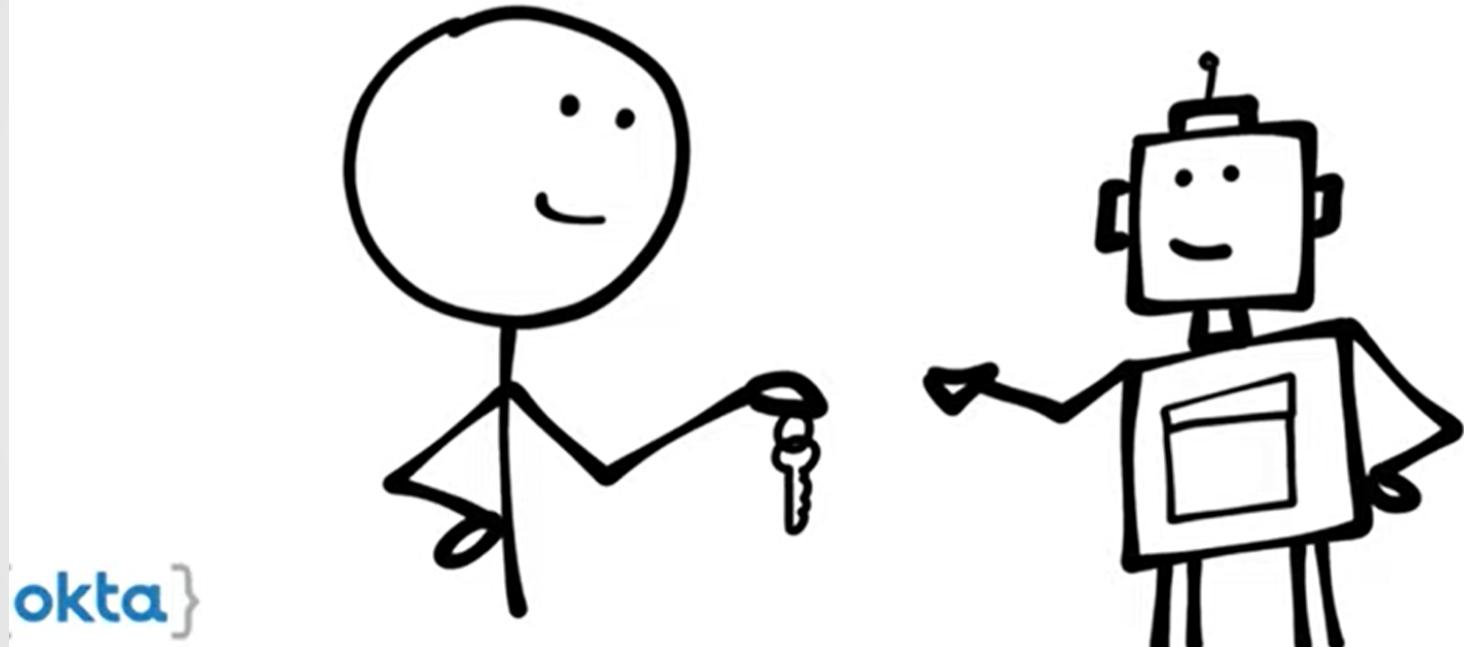
The Old  
Times



Anmeldeinformationen wurden einfach geteilt  
Setzt großes Vertrauen voraus

# OAuth 2.0

# OAuth 2.0



okta}

Statt der Anmeldedaten lieber einen Schlüssel austauschen  
Mit dem kann ich dann Sachen in der anderen Applikation machen  
Die Anmeldung an dem Dienst heißt 'Authorization' oder 'Delegated Authorization'

Ich autorisiere einen Applikation Dinge für mich zu tun, ohne dieser Applikation mein Passwort zu geben.

Ich melde mich am Authorization-Server an und sage dem, dass die Applikation das darf.

Die Applikation bekommt nur einen Schlüssel und sieht nie meine Daten.

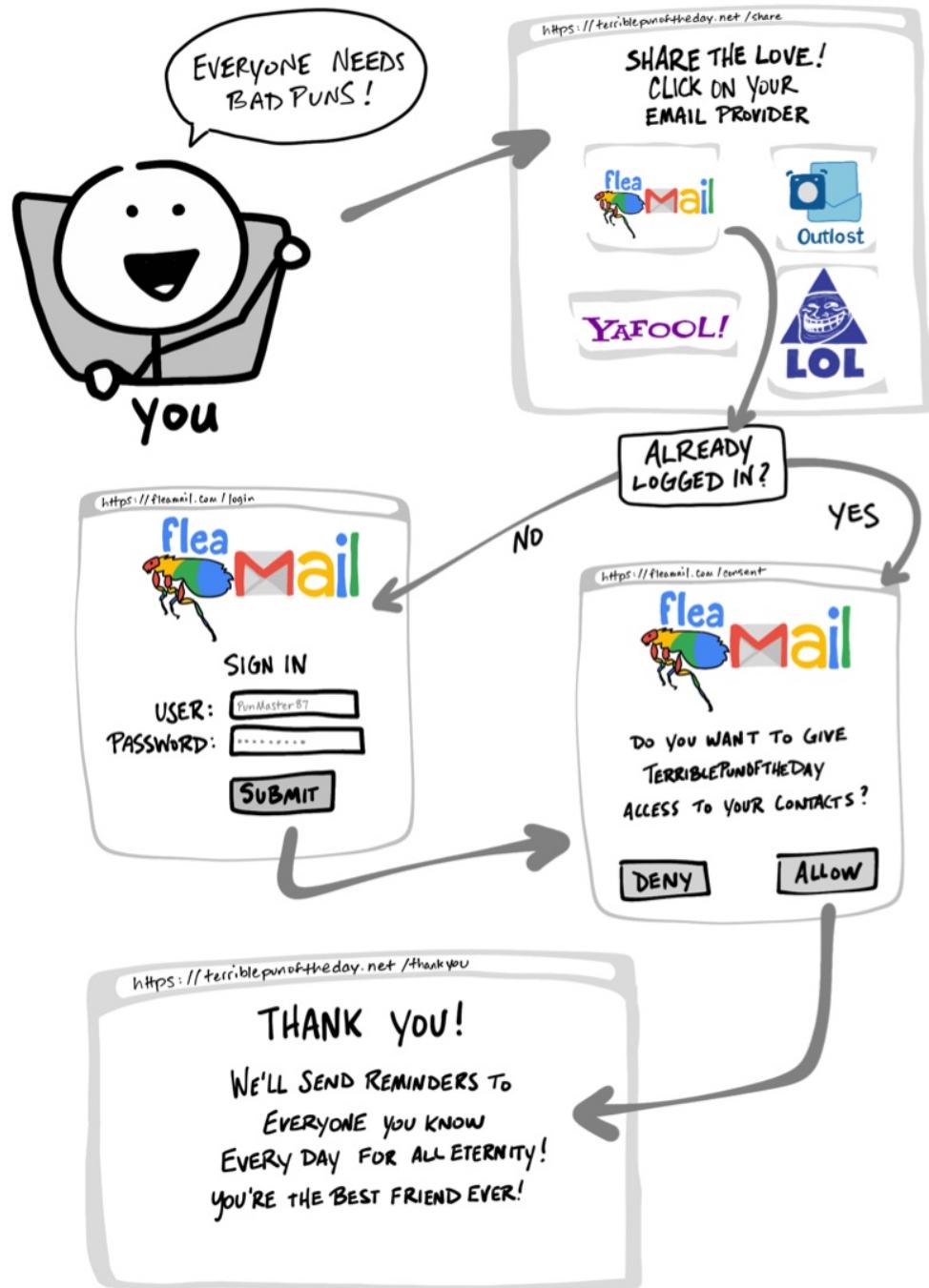
Dies funktioniert über Redirects.

Ich kann jederzeit am Authorization-Server den Key zurückziehen.

# OAuth 2.0

## Authorization Code Flow

1. E-Mail Provider auswählen
2. Redirect zum E-Mail Provider und Login, falls notwendig
3. Erlaubnis erteilen, die Kontakte zu browsen
4. Redirect zurück zur aufrufenden Seite



# Terminologie

## Resource Owner



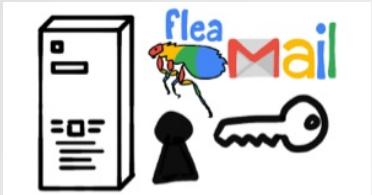
Der Benutzer. Hat Gewalt über seine Daten und kann bestimmen, was damit passieren soll und was nicht.

## Client



Die Applikation, die im Namen des Benutzers etwas machen, oder auf Daten zugreifen will.

## Authorization Server



Der Server, der den Benutzer bereits kennt.  
Der hat dort einen Account und kann sich dort  
auch anmelden.

## Resource Server



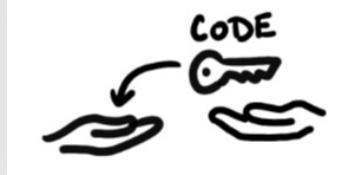
API oder Service, die der Client mit dem  
Account des Benutzers benutzen will.  
(Oft am Authorization Server selbst)

## Redirect URI

`https://terriblepunoftheday.net/callback`

Nach dem Autorisieren wird der Benutzer an  
diese URI weitergeleitet. (Callback URL)

## Response Type



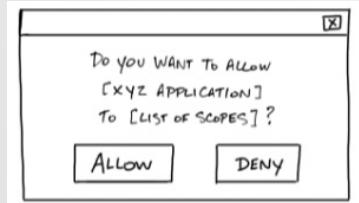
Der Response-Type, den der Client empfangen will, wenn die Autorisierung abgeschlossen ist. (meistens 'code')

## Scope

- READ CONTACTS
- CREATE CONTACT
- DELETE CONTACT
- READ PROFILE

Granulare Permissions, die der Client verlangt.  
Zugang zu Daten, Aktionen...

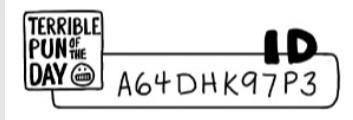
## Consent



Der Benutzer wird gefragt, ob er für den Client die Liste von Scopes absegnet.

Die Antwort ist dann die Einwilligung (consent).

## Client ID



Der Authorization-Server weist jedem Client eine eindeutige ID zu.

## Client Secret



Secret Password zwischen Client und Authorization-Server. Damit können die beiden Daten im Hintergrund austauschen.

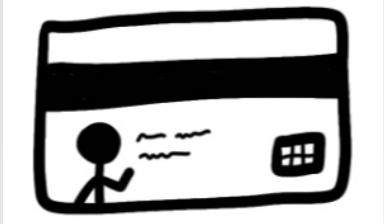
## Authorization Code



Nur kurze Zeit gültiger Code, den der Authorization-Server an den Client schickt.

Der Client schickt dann den Code + Client Secret zum Server zurück und bekommt einen Access Token.

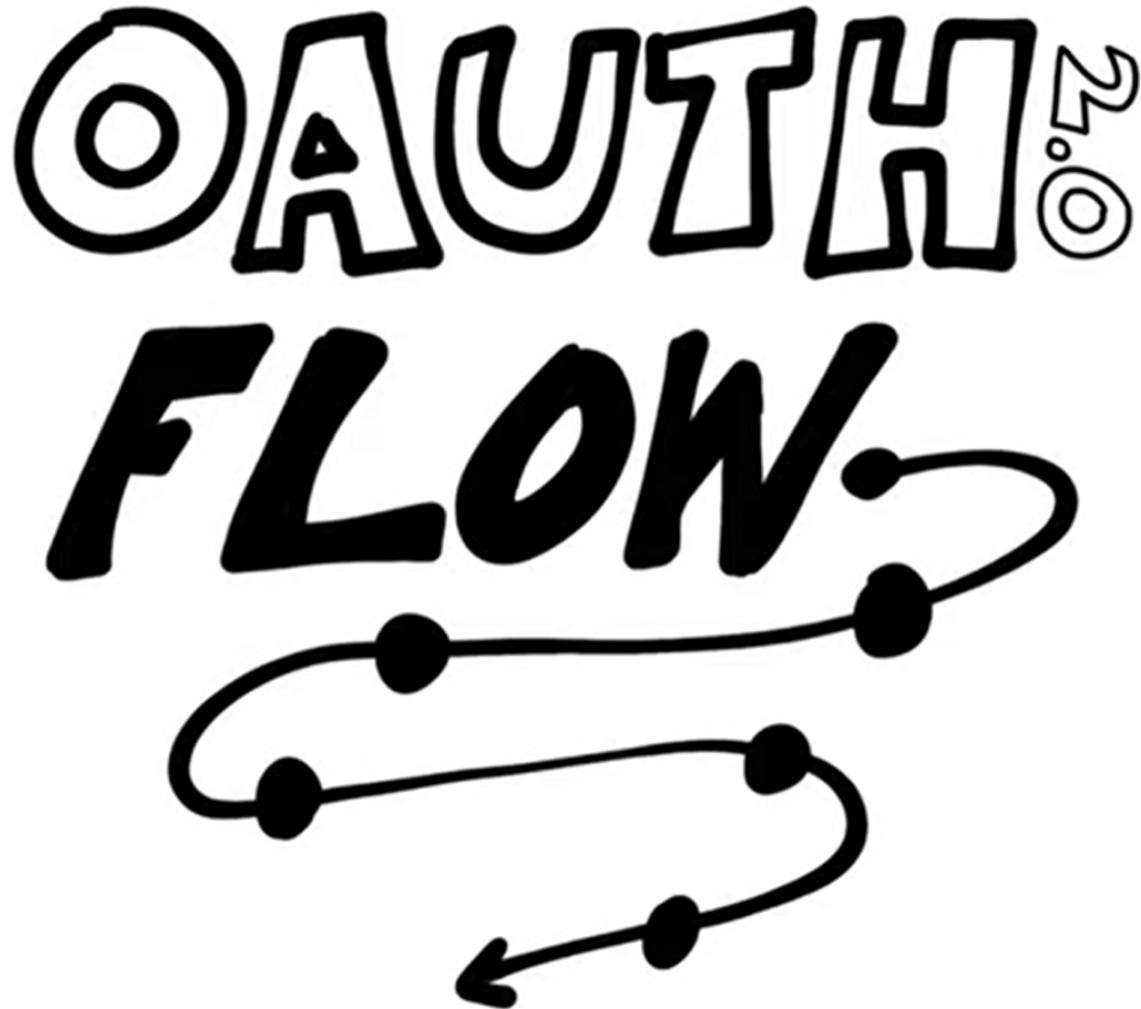
## Access Token



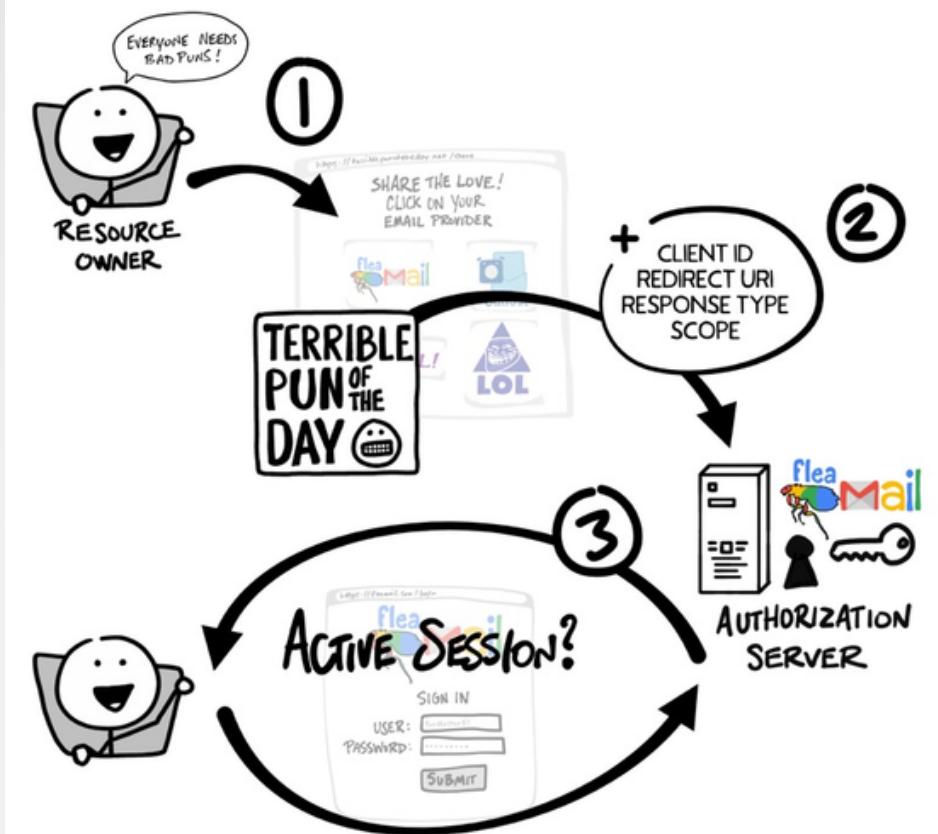
Der Schlüssel, den der Client ab jetzt dazu verwenden wird, mit dem Resource-Server zu kommunizieren.

Solange der Client einen Access Token hat, der nicht abgelaufen ist, darf er auf den Resource-Server zugreifen.

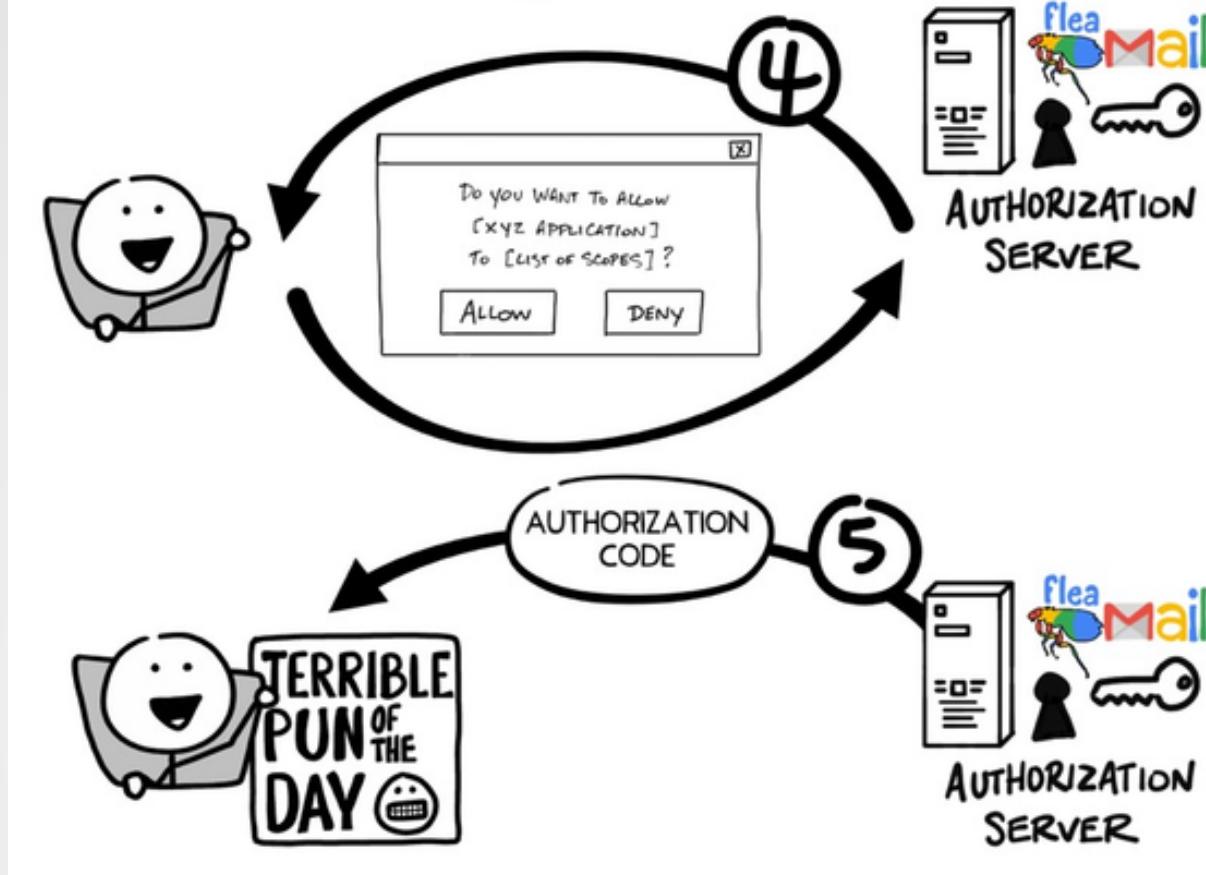
Lebenszeit eines Access Token ist meistens kurz (z.B. 3 Minuten), da er nicht invalidiert werden kann und nur beim Refresh eine Überprüfung des Zustimmung erfolgen kann.



Am vorherigen Beispiel

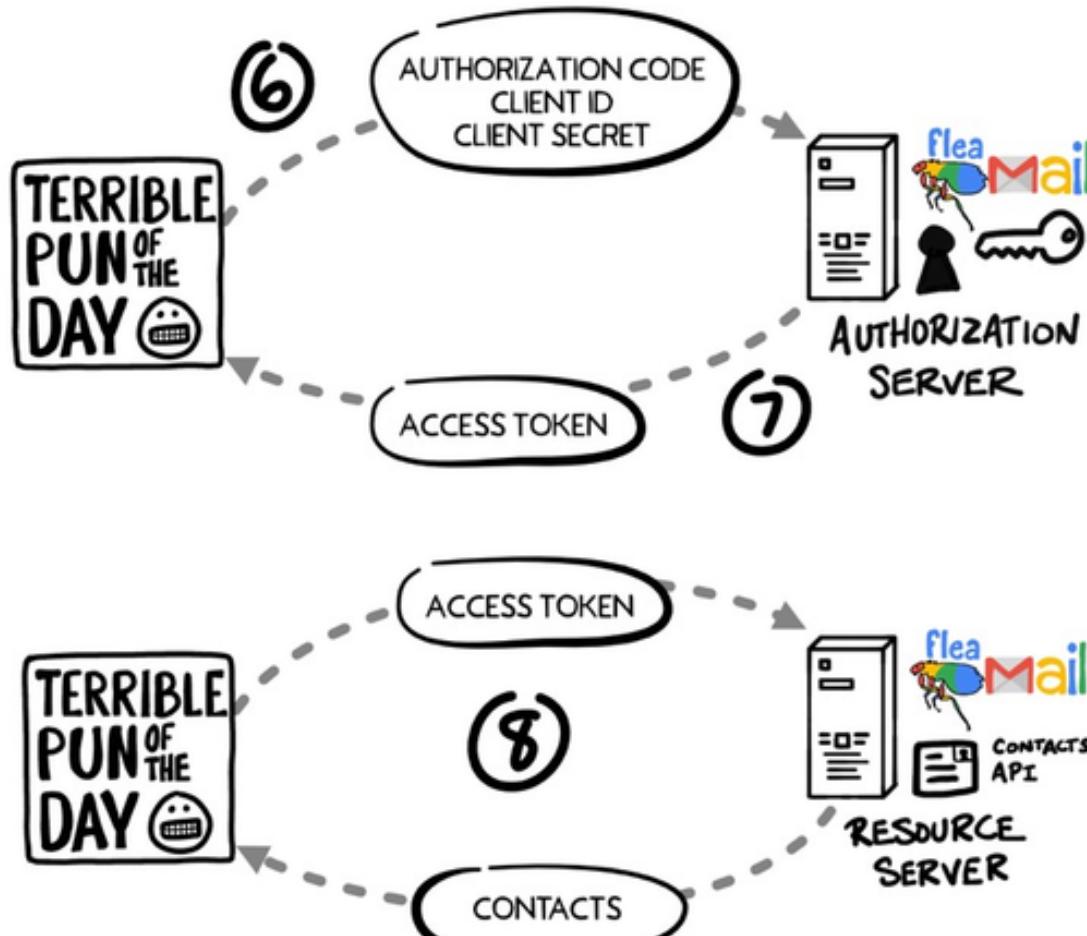


- 1- Du, der Resource Owner, will dem Client "Terrible Pun of the Day" dem Client, Zugriff auf die Kontakte geben, damit er Einladungen an alle verschicken kann.
- 2- Der Client leitet Deinen Browser auf den Authorization-Server um und inkludiert im Request die Client-ID, Redirect URI, Response Type und die Scopes, die er benötigt.
- 3 - Der Authorization-Server verifiziert Deine Identität und braucht eventuell dafür ein Login.



4 - Der Authorization-Server will von Dir eine Zustimmung für den Client und die Scopes, die er verlangt. Du kannst die Zustimmung geben, oder auch nicht.

5 - Der Authorization-Server leitet zurück zur Redirect URI und hängt den Authorization-Code an.



6- The Client meldet sich direkt beim Authorization-Server und schickt die Client-ID, das Client-Secret und den eben empfangenen Authorization Code.

7 - Der Authorization Server verifiziert die Daten und antwortet mit einem zeitlich begrenzten Access Token.

8 - Der Client kann nun diesen Access Token verwenden, um einen Request an den Resource-Server zu stellen, damit er die Kontakte bekommt.

# OpenID

# Connect

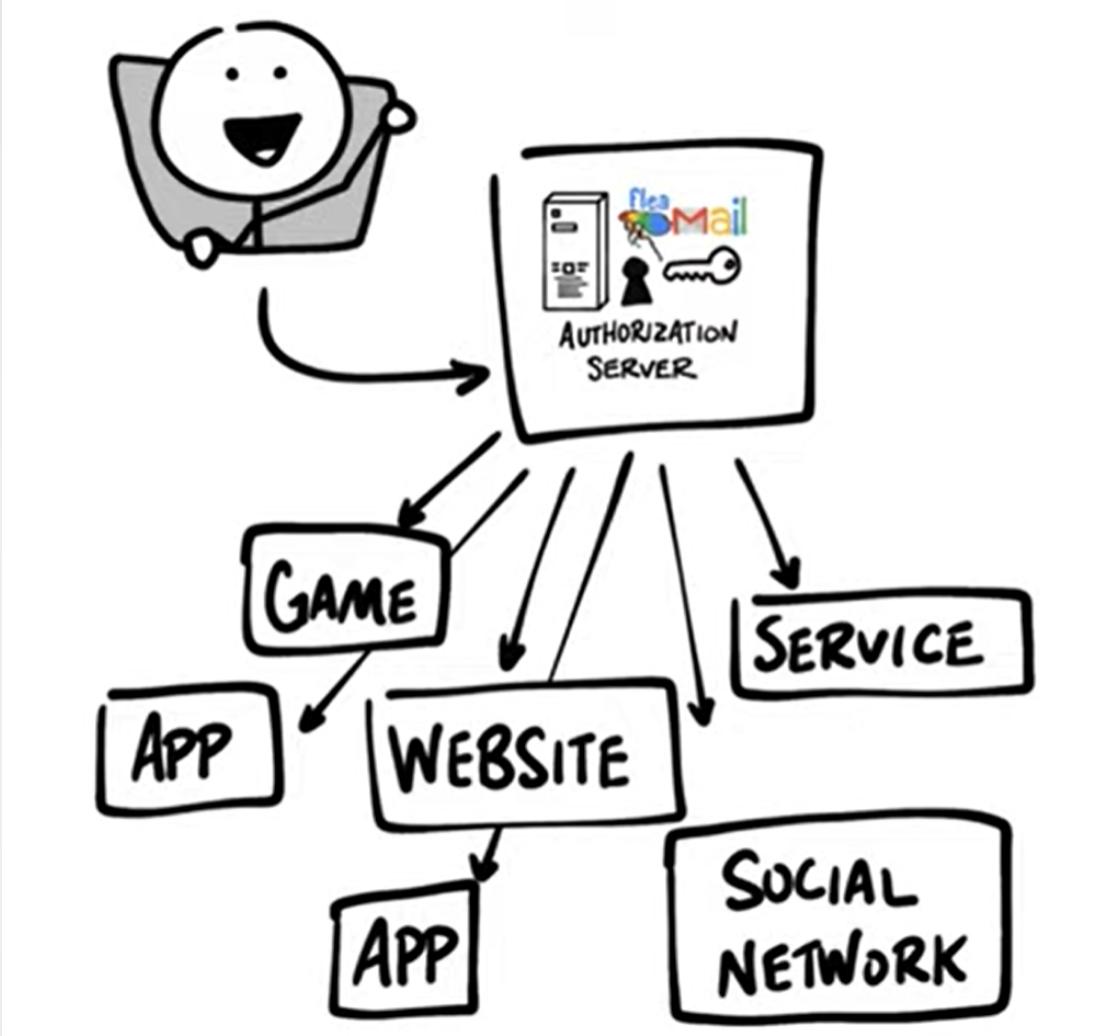
# (OIDC)



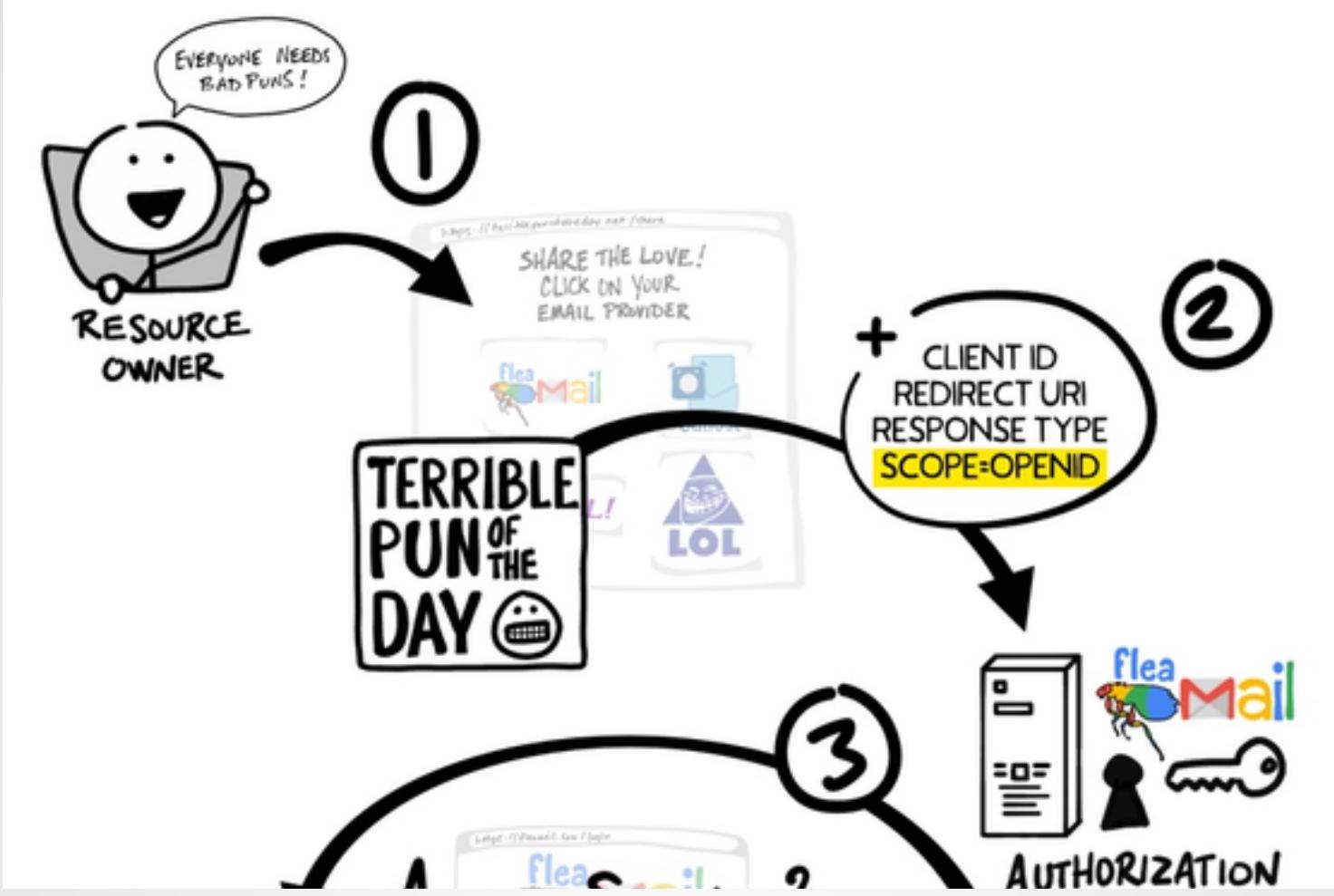
OAuth 2.0 - Macht Autorisierung von einer  
Applikation zu einer anderen - Ist wie  
einen Schlüssel zu übergeben.

OIDC - Erweitert OAuth 2.0 im  
Informationen über den Benutzer - Ist wie  
statt dem Schlüssel einen Ausweis  
zugeben, in dem noch Informationen  
drinnen stehen

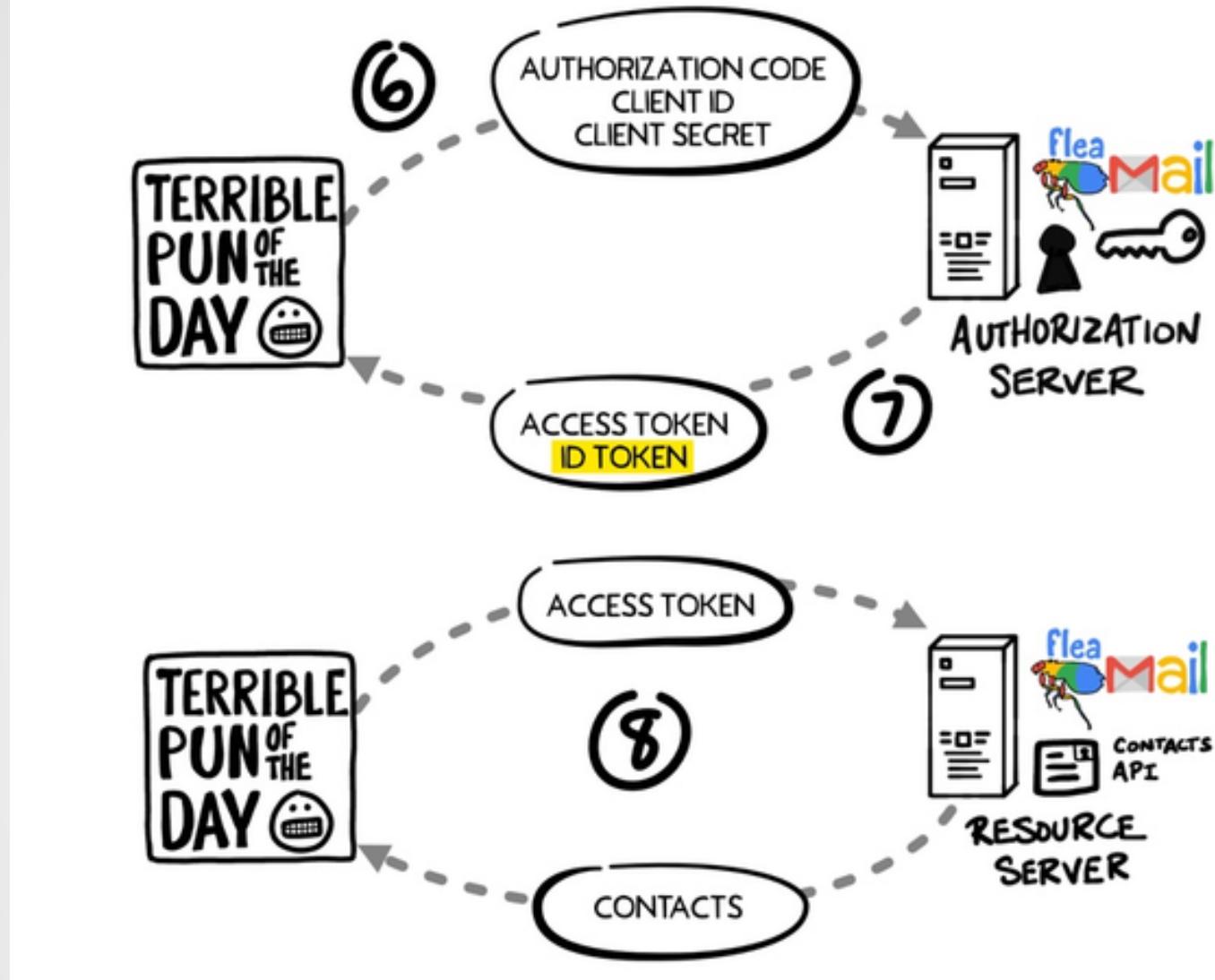
Authorization-Server, der OIDC  
unterstützt, wird deswegen oft auch  
Identity-Provider genannt



Ein Login kann für mehrere Applikationen  
benutzt werden (Single Sign On - SSO)



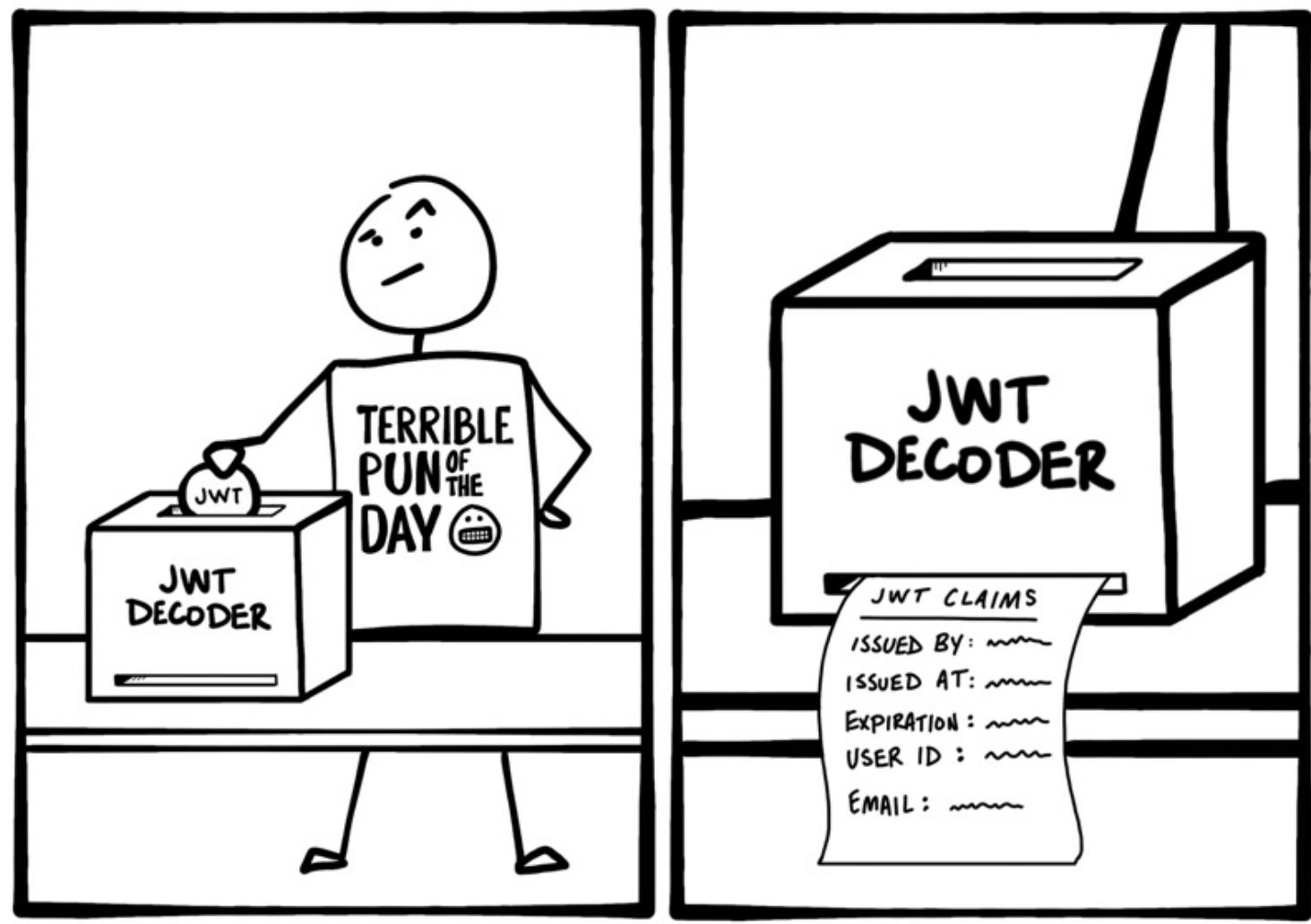
Gleich wie vorher, aber SCOPE=OPENID



Alles wie vorher - nur kommt jetzt ein ID-Token zusätzlich zum Access Token zurück

# JWT

## Json Web Token



Enthält verschlüsselte Informationen  
(Claims), die mit einem Decoder gelesen  
werden können.

# JWT Beispiel (<https://jwt.io>)

## Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

## Decoded

EDIT THE PAYLOAD AND SECRET

### HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

### PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

### VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

Base64 encoded Header and Payload, gefolgt von einer Checksum  
im angegebenen Verfahren + Secret

# Keycloak

(RedHat)

<https://www.keycloak.org/>



