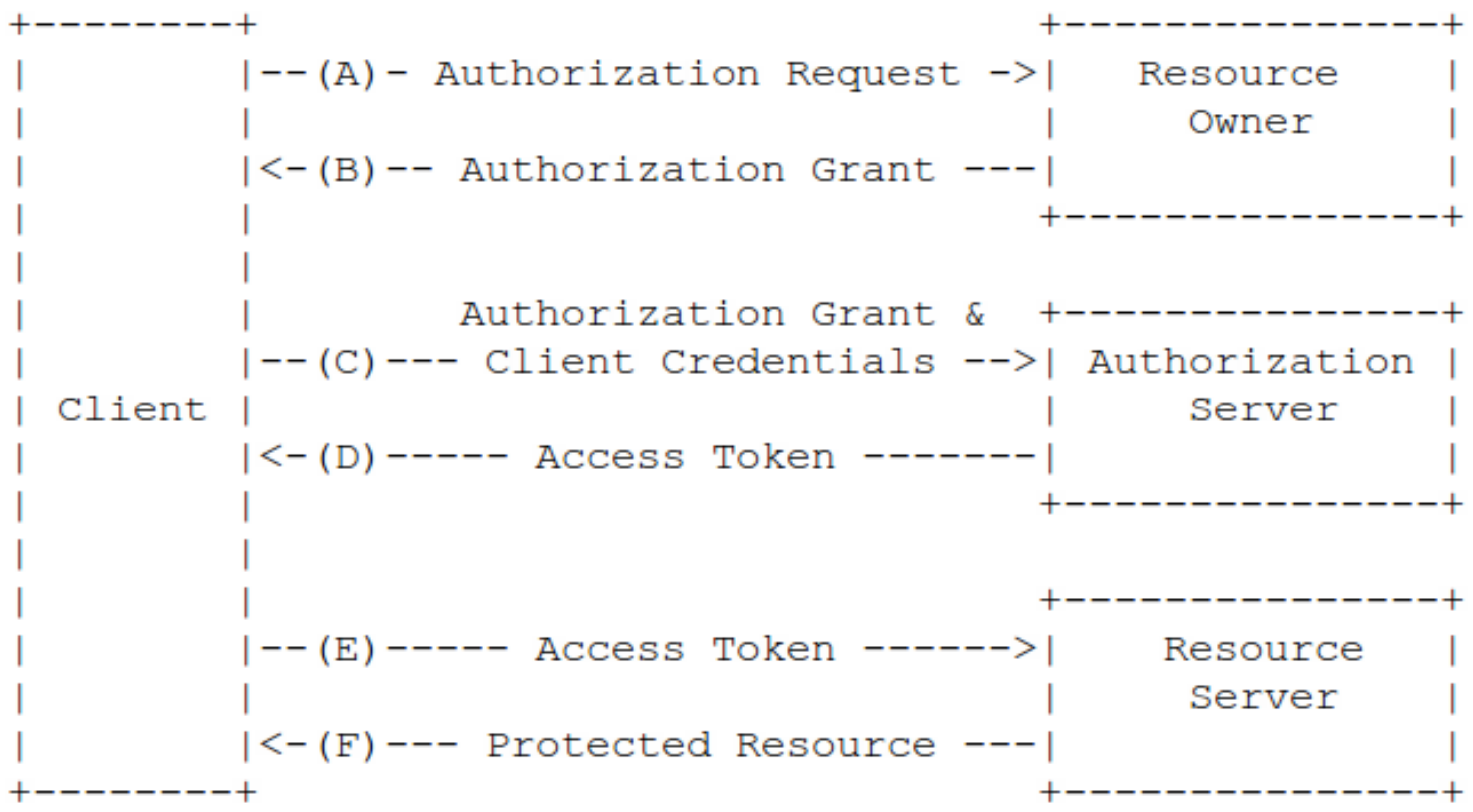
OAuth 2.0 and the Google API Client for Python

What is OAuth 2.0?



OAuth 2.0 the protocol can be a little complex.

What is OAuth 2.0?

API

User

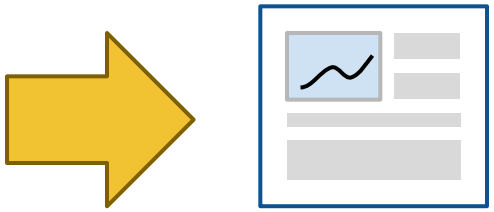
App

It is trying to solve a tricky problem.

What is OAuth 2.0?

API

User

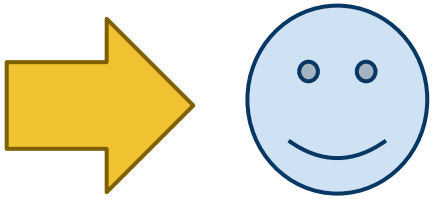


App

If you, the developer, are building an application.

What is OAuth 2.0?

API



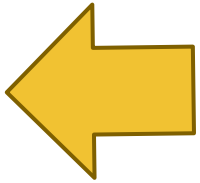
User

App

And your users

What is OAuth 2.0?

API



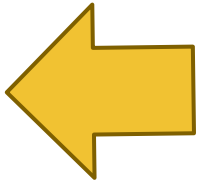
User

App

have data in another service that your application needs to function

What is OAuth 2.0?

API



User

App

such as their tasks list, or their photos

What is OAuth 2.0?

API

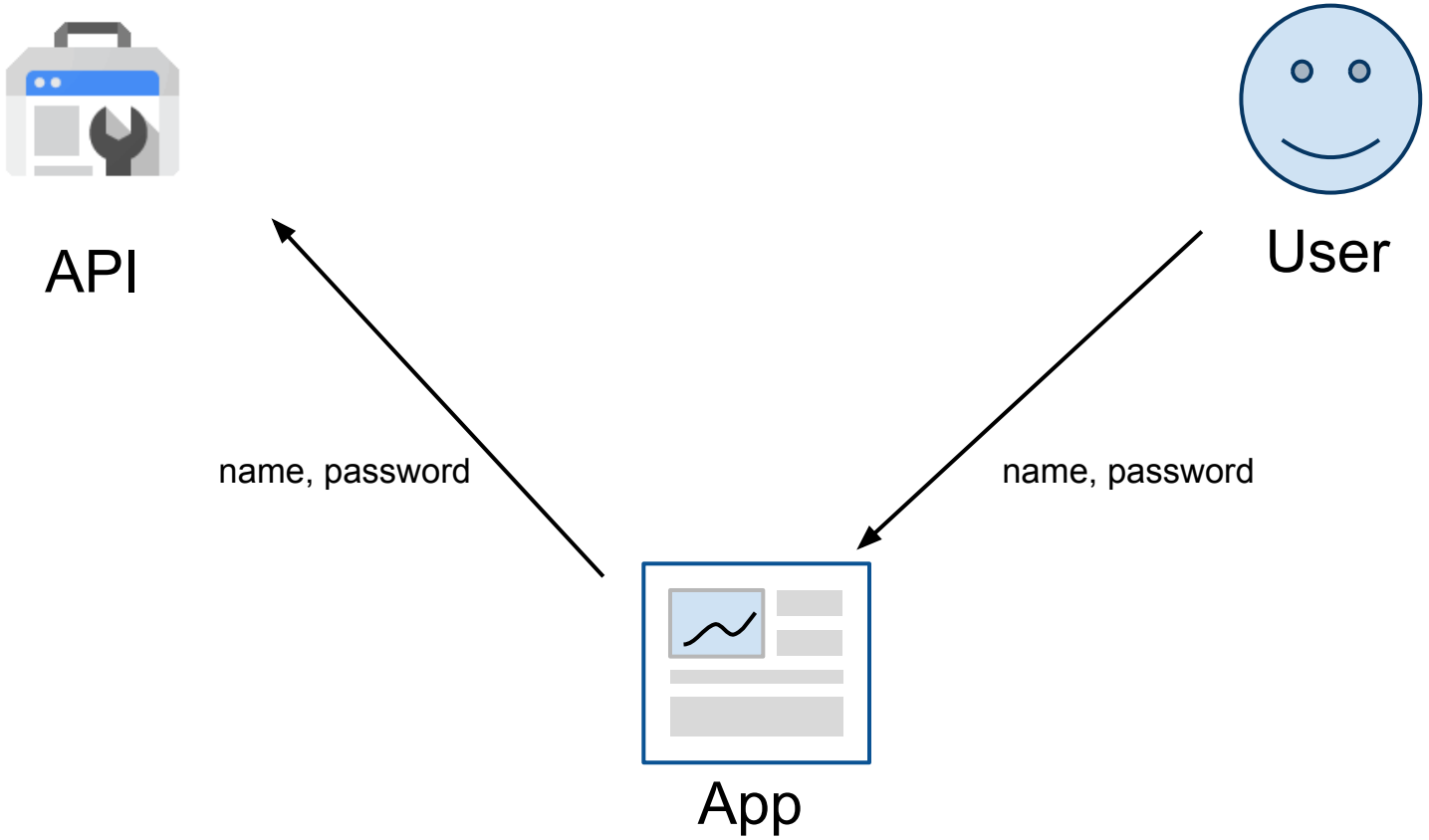
???

User

App

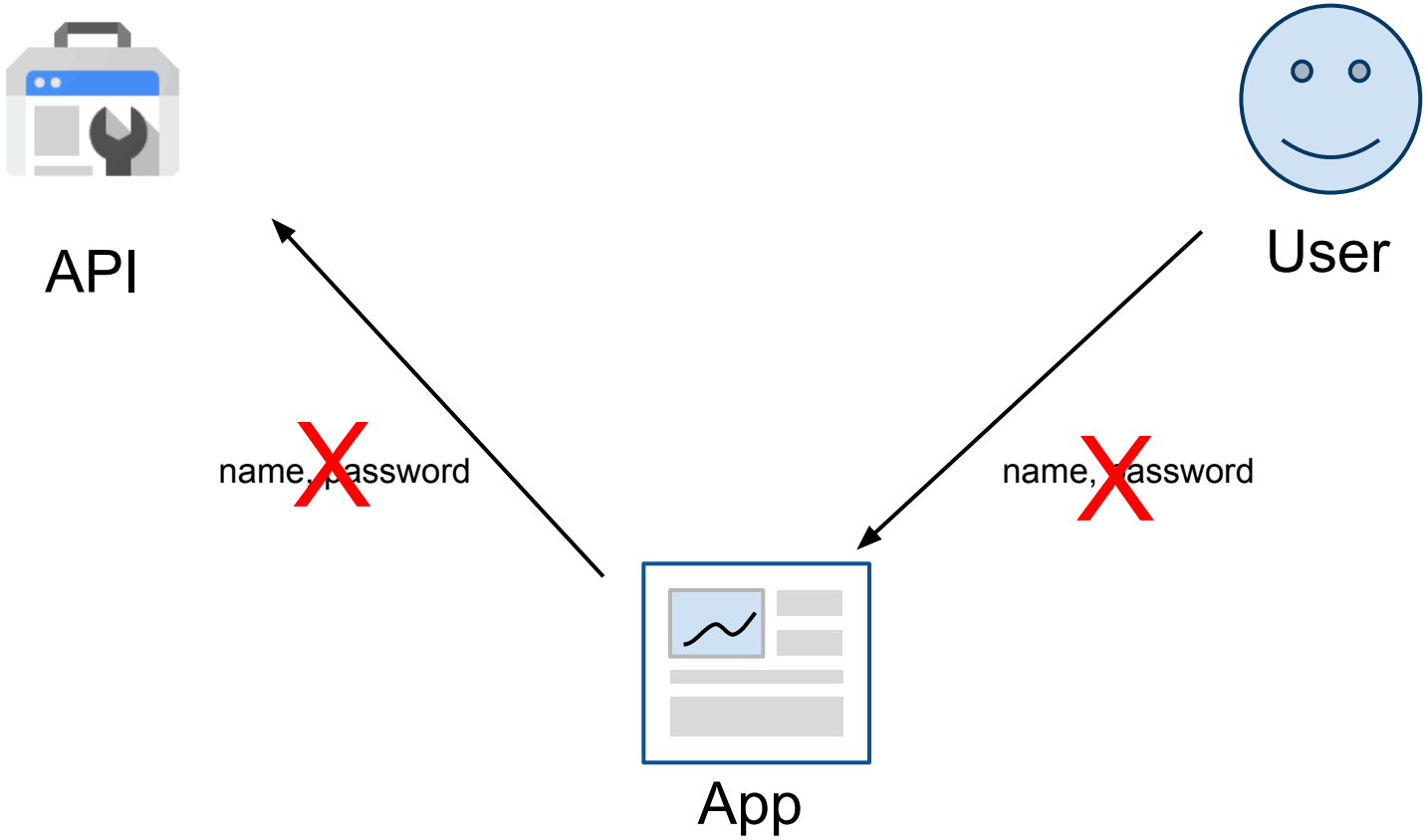
how do you go about getting it?

Noooooooo



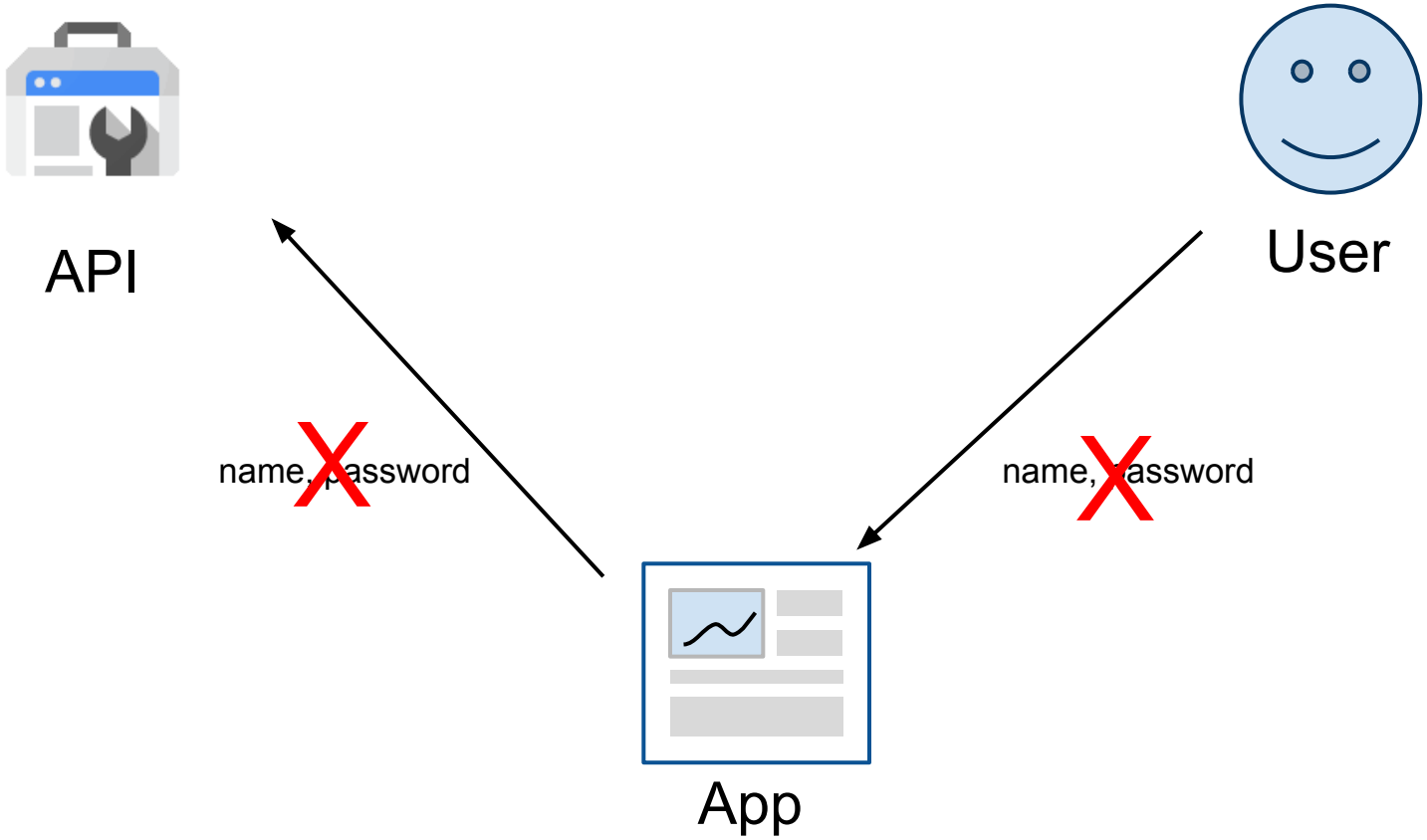
You could ask the user for their name and password.

Noooooooo



But then the user has given your application access to all their data on that service. That's not safe. Don't do that.

Noooooooo



The user's name and password are like keys to their digital kingdom, you should never ask for them.

Better

API

User

App

What we really want is a special key, one that only allows access to a limited set of data in the API.

Better

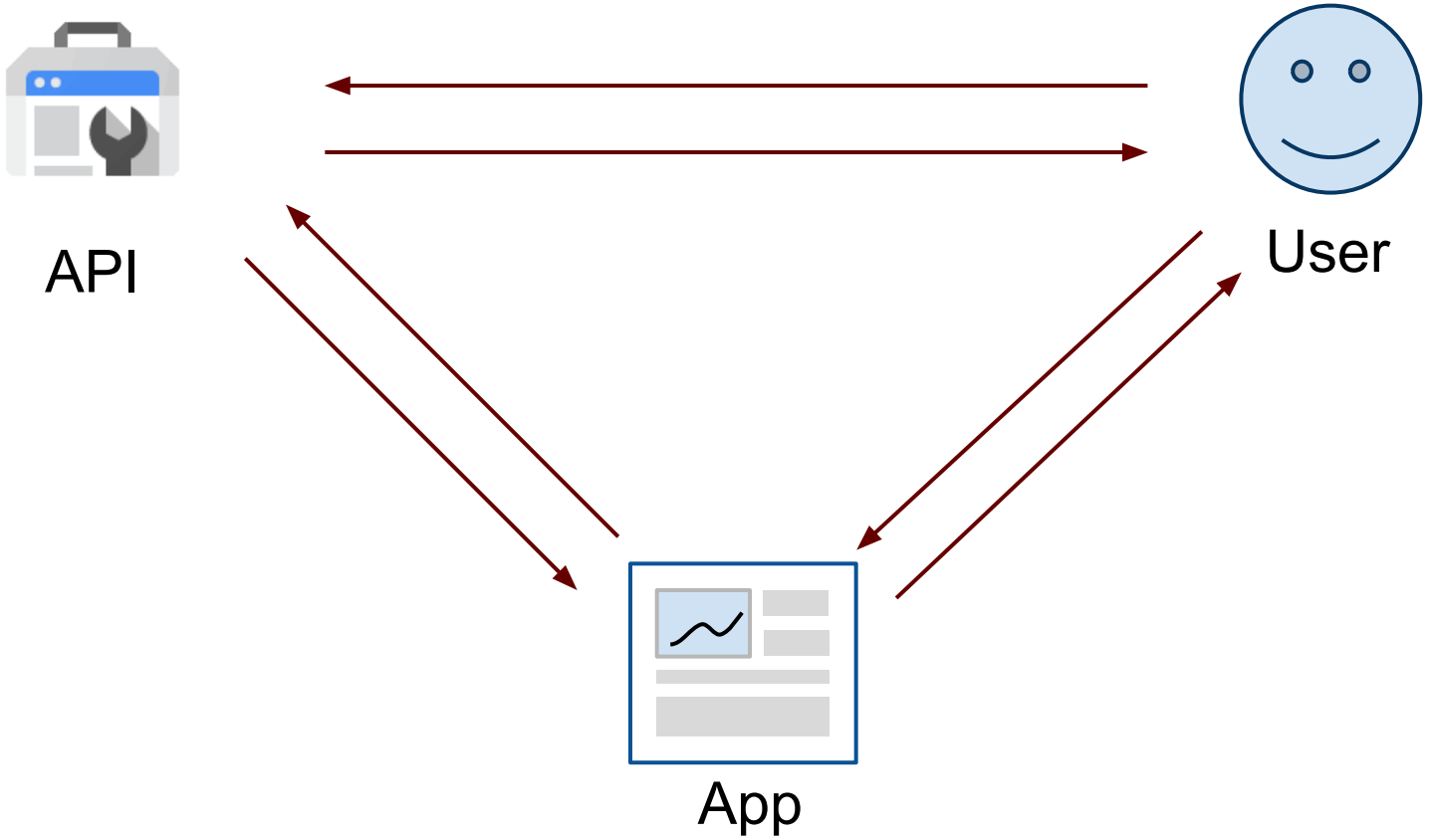
API

User

App

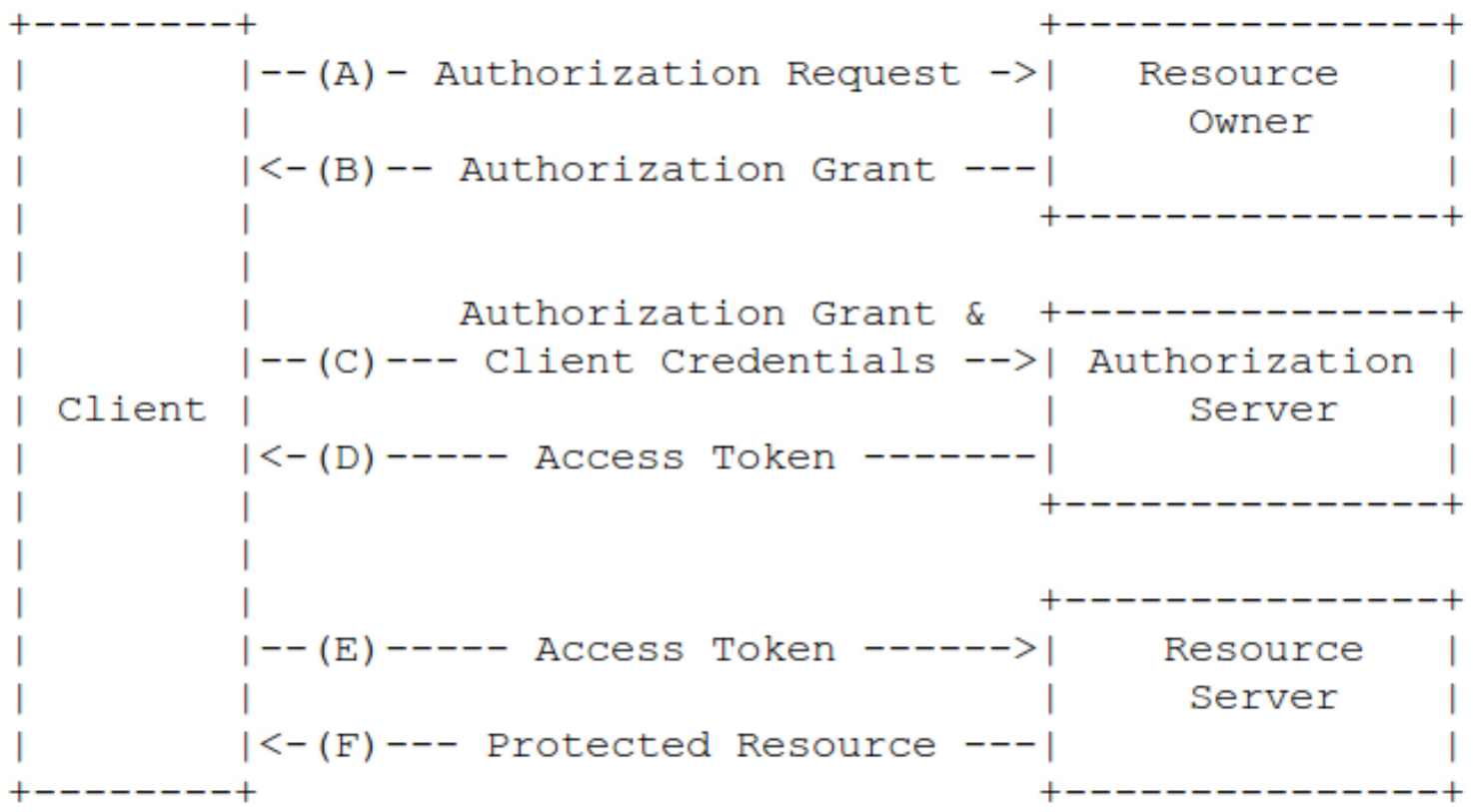
A special key that the User can let the App acquire and use without the use of their name and password.

This is OAuth 2.0



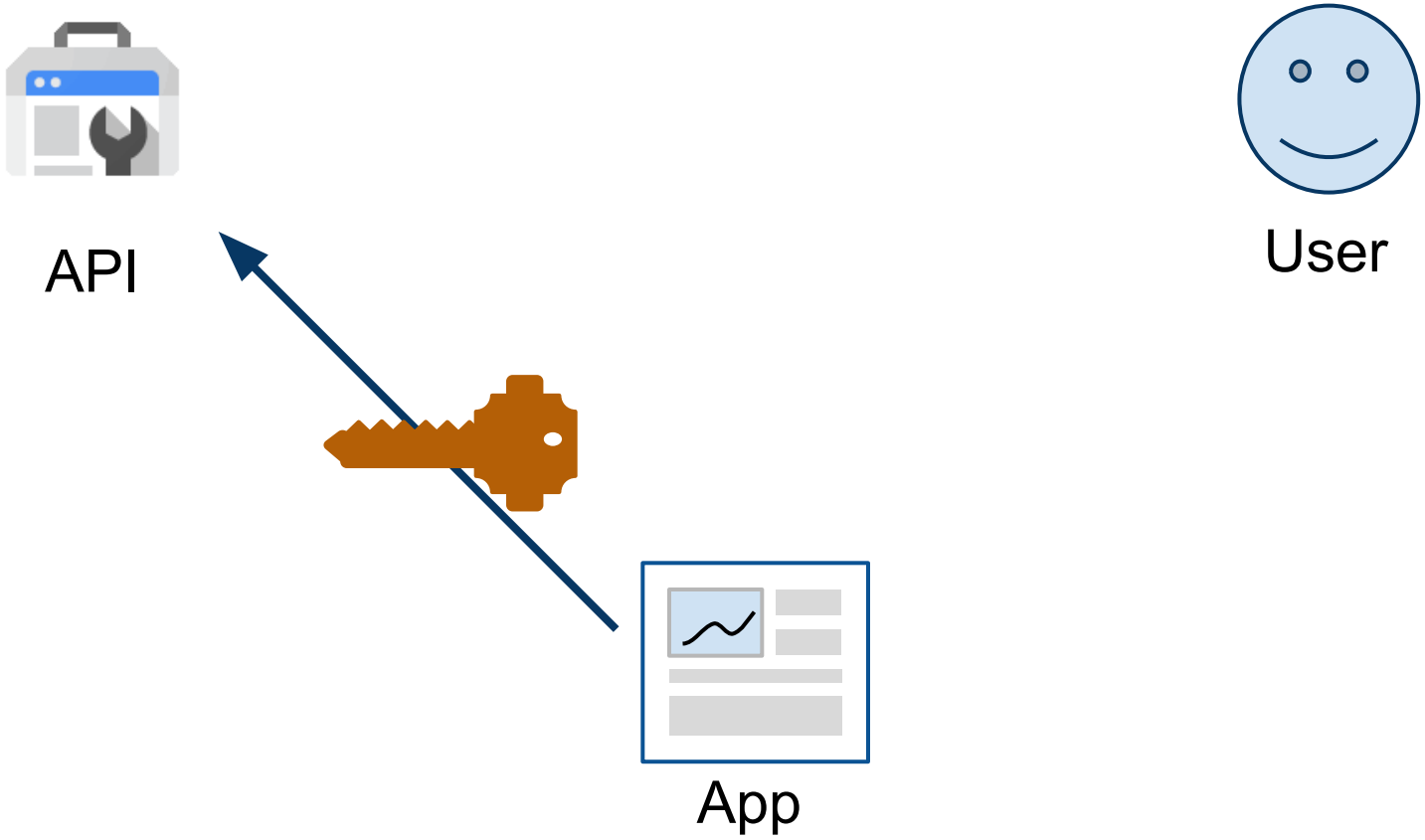
But for that to work, everyone has to confirm that everyone else is who they say they are.

This is OAuth 2.0



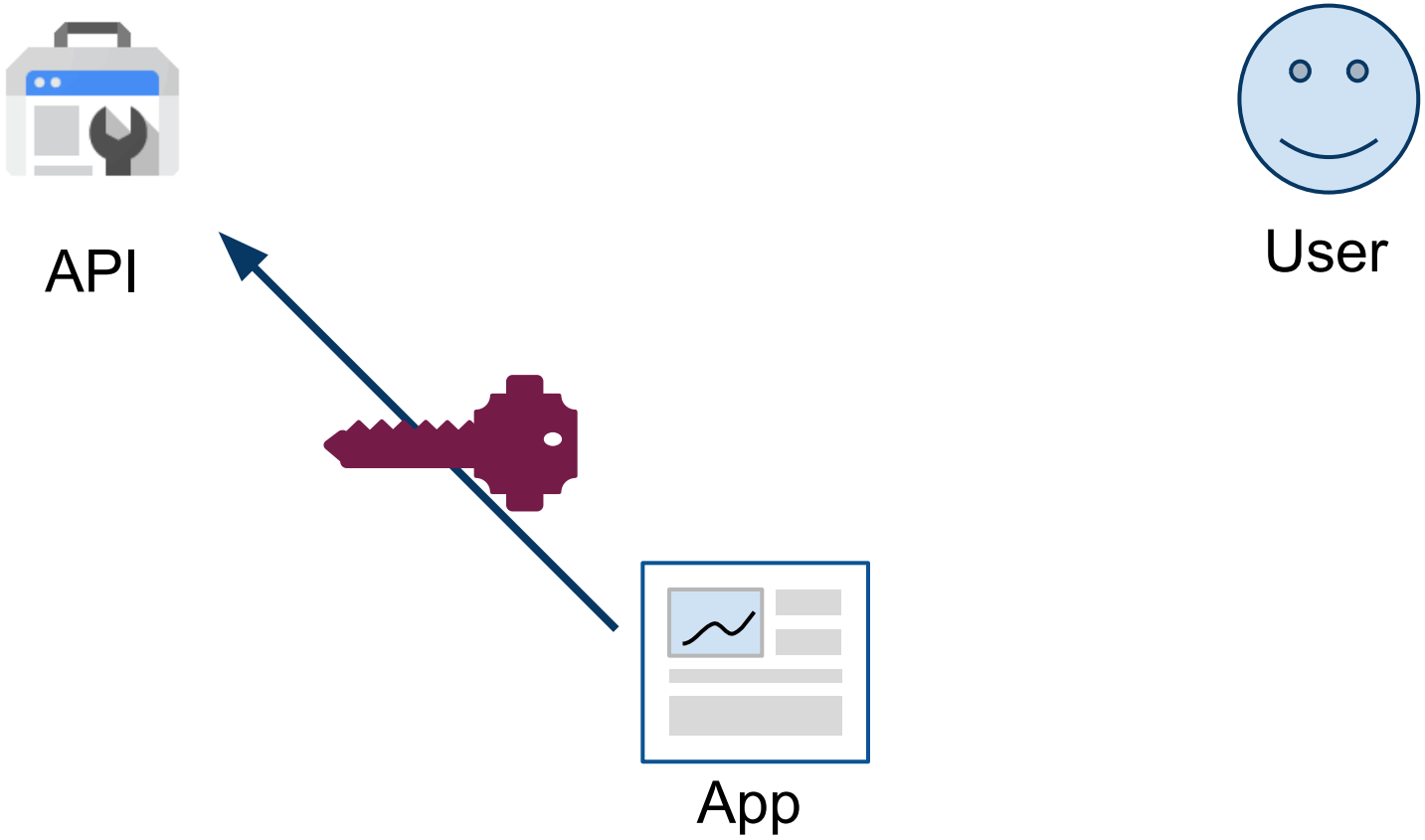
Which is where the complexity comes from.

Keys



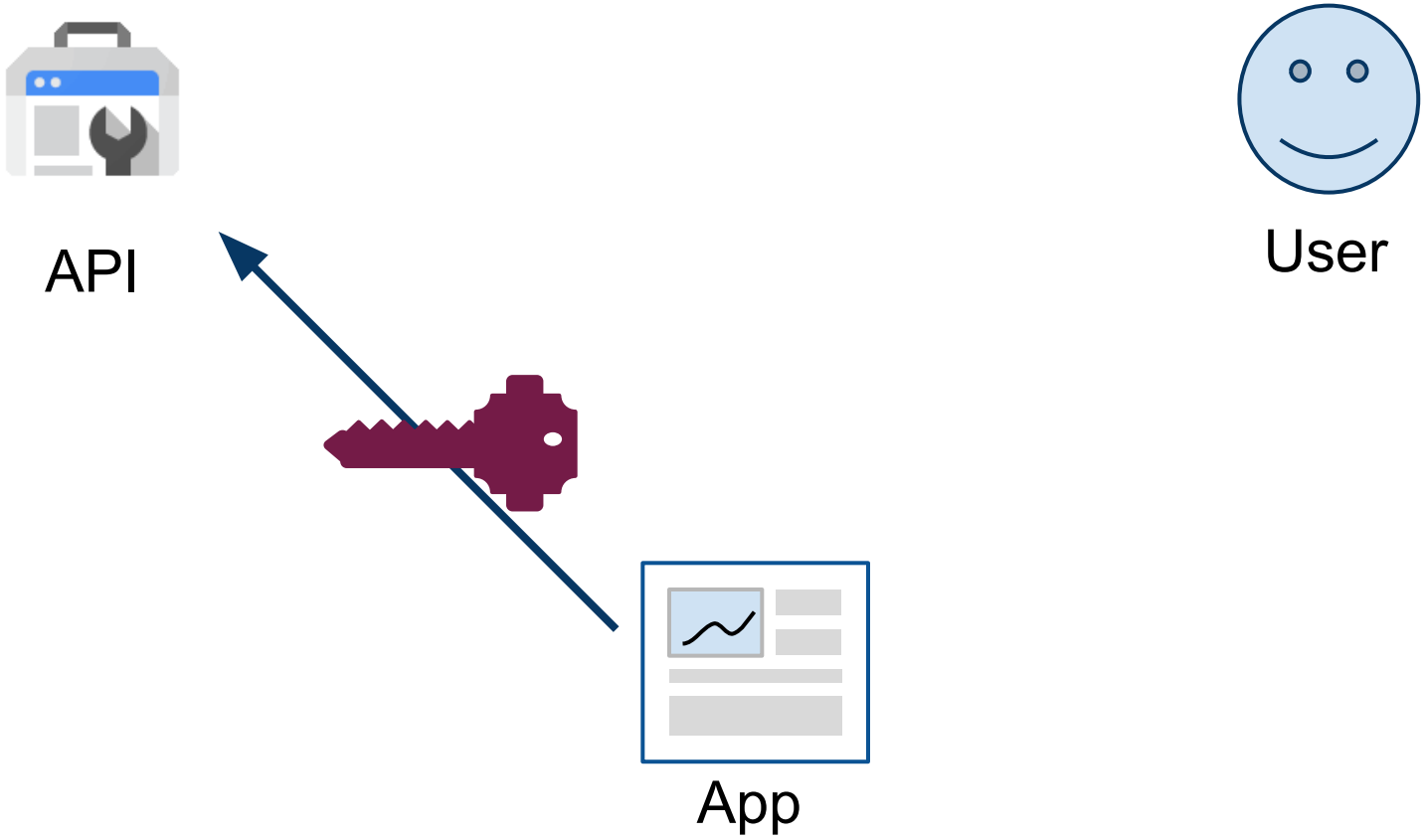
It's actually a little more complicated than even that, because that special key

Keys



can change over time to keep things secure.

Keys



Now that we know what OAuth 2.0 looks like, how does it work in the Google API Client for Python?

Credentials

API

Credentials

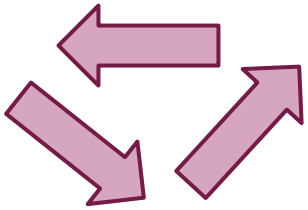
User

App

The key is held in a Credentials object.

Flow

API



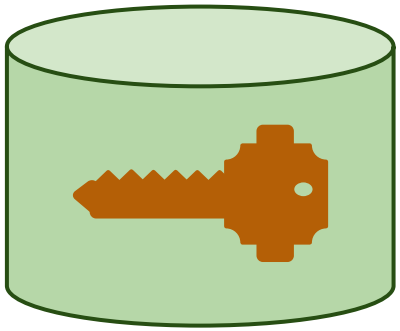
Flow

User

App

All the steps needed to go through getting Credentials is in a Flow object.

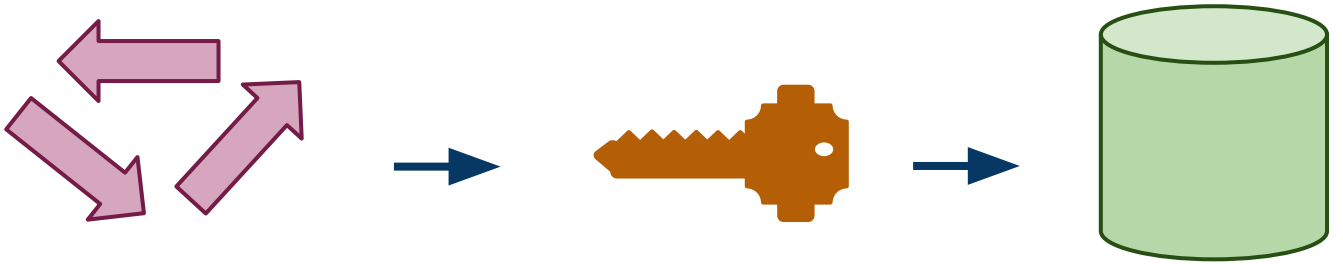
Storage



Storage

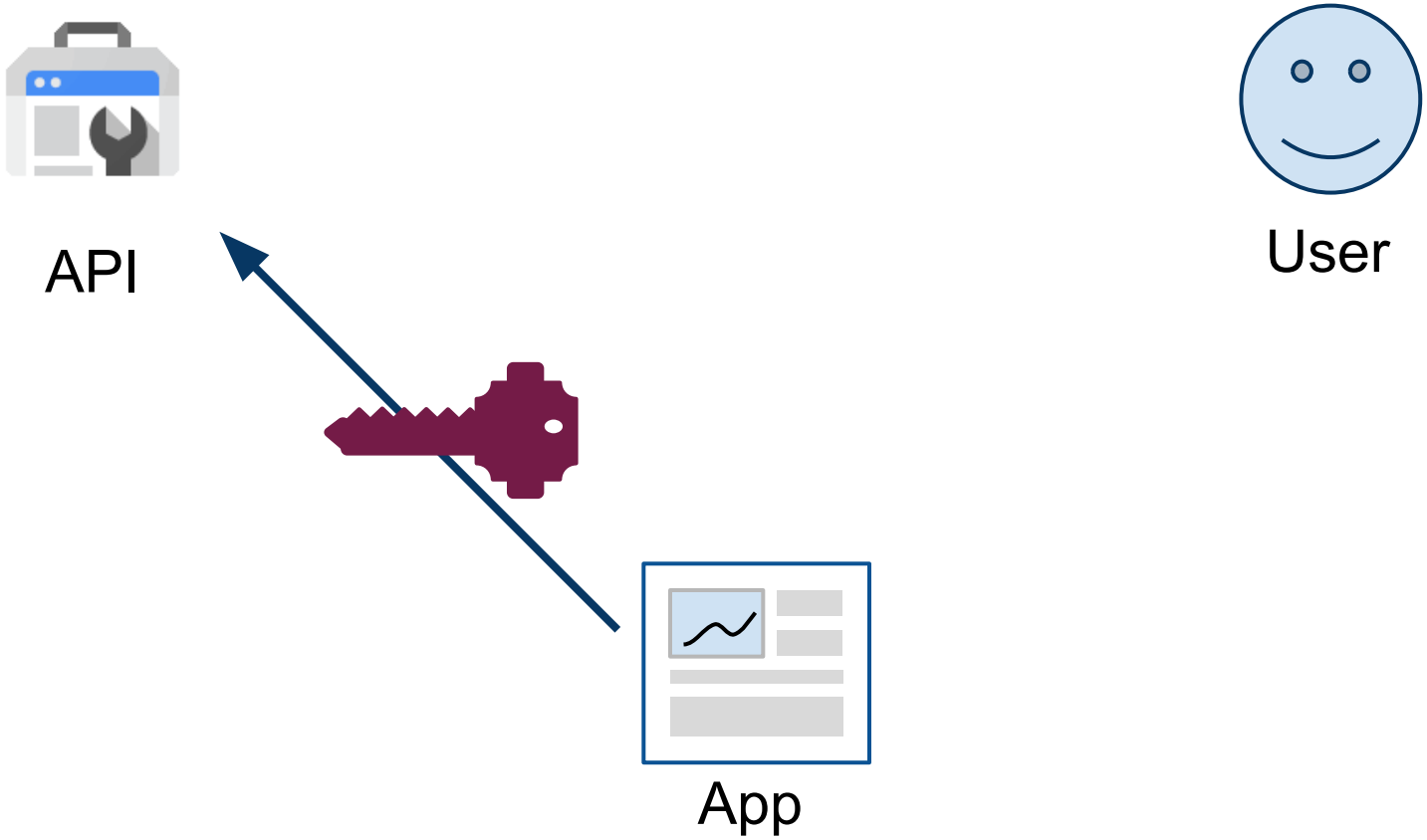
And finally, because keys can change over time there is a Storage object for storing and retrieving keys.

The model



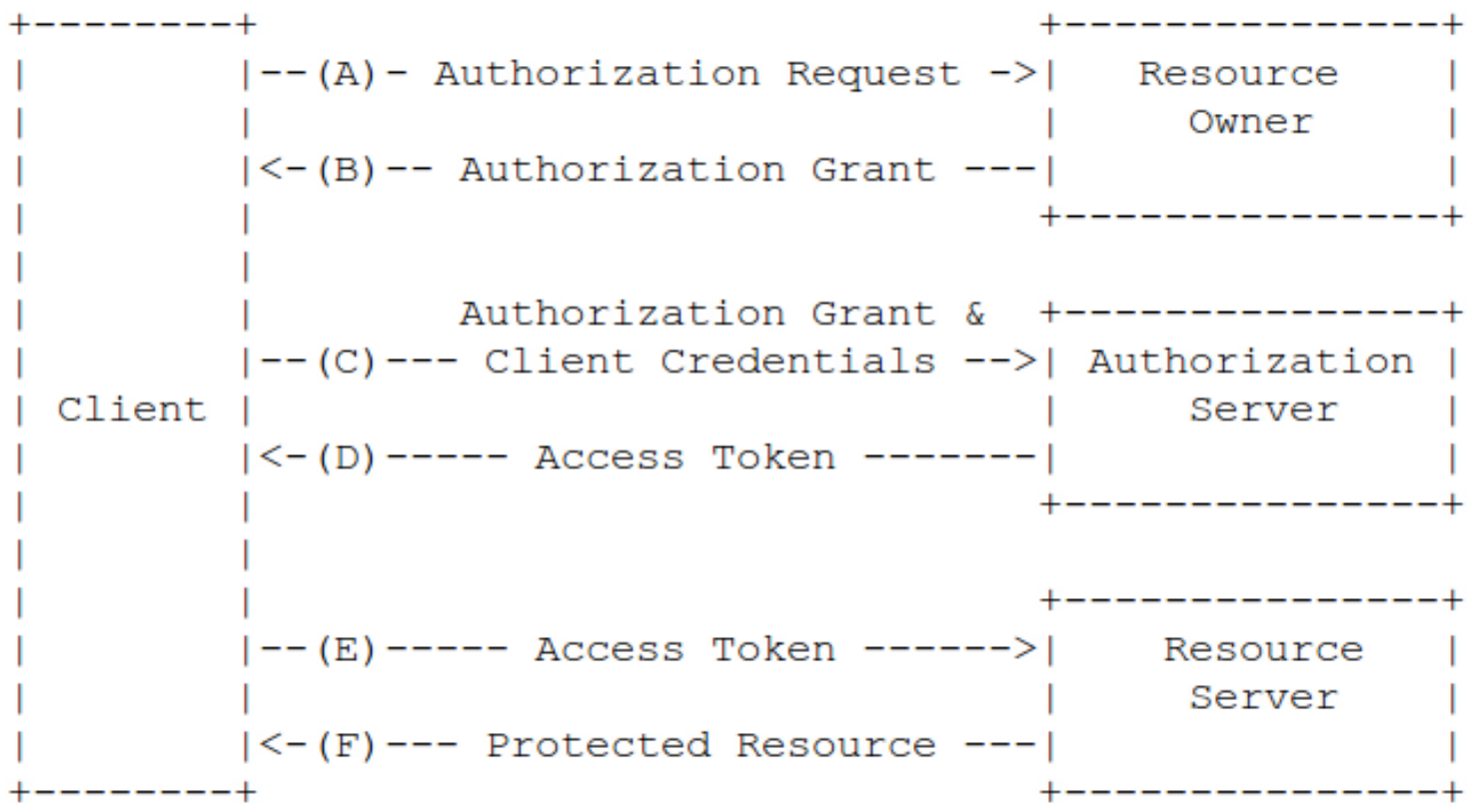
You set up and run a Flow, which in the end produces Credentials, which you store in a Storage.

From Python



Later, when you need the key, you take it out of Storage and use it.

This is OAuth 2.0



Which I hope you agree is simpler than this.

Step by step

So let's look at actual code.

Step by step

```
FLOW = OAuth2WebServerFlow(
  client_id='<CLIENT ID HERE>',
  client_secret='<CLIENT SECRET HERE>',
  redirect_uri='https://.../oauth2callback',
  scope='https://.../tasks',
  user_agent='my-sample/1.0')
```

First, create a Flow.

Step by step

```
FLOW = OAuth2WebServerFlow(
  client_id='<CLIENT ID HERE>',
  client_secret='<CLIENT SECRET HERE>',
  redirect_uri='https://.../oauth2callback',
  scope='https://.../tasks',
  user_agent='my-sample/1.0')
```

For Google APIs you visit http://code.google.com/apis/console to create a client id and secret for your application.

Step by step

```
authorize_url = FLOW.step1_get_authorize_url()
self.redirect(authorize_url)
```

Then we kick off the Flow.

Step by step

```
credentials = flow.step2_exchange(self.request.params)
storage = StorageByKeyName(
    Credentials, user.user_id(), 'credentials'
  )
storage.put(credentials)
```

We get Credentials when the Flow finishes, which we save in a Storage.

Step by step

```
user = users.get_current_user()
storage = StorageByKeyName(
        Credentials, user.user_id(), 'credentials'
    )
credentials = storage.get()

http = httplib2.Http()
http = credentials.authorize(http)
```

To use Credentials we retrieve them from the Storage and apply them to an httplib2.Http() object.

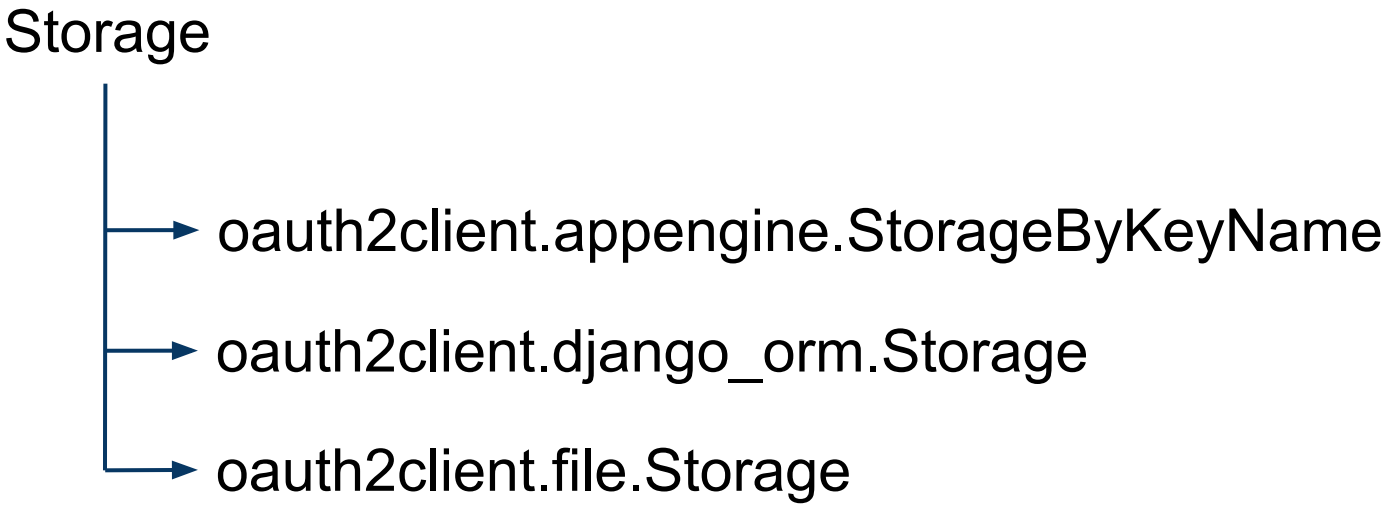
Step by step

```
user = users.get_current_user()
storage = StorageByKeyName(
        Credentials, user.user_id(), 'credentials'
    )
credentials = storage.get()

http = httplib2.Http()
http = credentials.authorize(http)
```

Now any HTTP requests made with http will be authorized with those Credentials.

Specializations



There are Storage classes for different platforms

Details

```
decorator = OAuth2Decorator(
    client_id='<YOUR CLIENT ID>',
    client_secret='<YOUR CLIENT SECRET>',
    scope='htt.../tasks',
    user_agent='my-sample-app/1.0')

http = httplib2.Http(memcache)
service = discovery.build('tasks', 'v1', http=http)

class MainHandler(webapp.RequestHandler):
  @decorator.oauth_required
  def get(self):
    http = decorator.http()
    followers = service.people().list(
```

And there are helpers for App Engine that make things even easier.

Details

```
decorator = OAuth2Decorator(
    client_id='<YOUR CLIENT ID>',
    client_secret='<YOUR CLIENT SECRET>',
    scope='htt.../tasks',
    user_agent='my-sample-app/1.0')

http = httplib2.Http(memcache)
service = discovery.build('tasks', 'v1', http=http)

class MainHandler(webapp.RequestHandler):
  @decorator.oauth_required
  def get(self):
    http = decorator.http()
    followers = service.people().list(
```

The decorator uses Flows, Storage and Credentials, but does so under the covers.

Details

```
decorator = OAuth2Decorator(
    client_id='<YOUR CLIENT ID>',
    client_secret='<YOUR CLIENT SECRET>',
    scope='htt.../tasks',
    user_agent='my-sample-app/1.0')

http = httplib2.Http(memcache)
service = discovery.build('tasks', 'v1', http=http)

class MainHandler(webapp.RequestHandler):
  @decorator.oauth_aware
  def get(self):
    if decorator.has_credentials():
        http = decorator.http()
        followers = service.people().list(
```

While oauth_required is the simplest interface to use, the suggested interface is oauth_aware.

Details

```
decorator = OAuth2Decorator(
    client_id='<YOUR CLIENT ID>',
    client_secret='<YOUR CLIENT SECRET>',
    scope='htt.../tasks',
    user_agent='my-sample-app/1.0')

http = httplib2.Http(memcache)
service = discovery.build('tasks', 'v1', http=http)

class MainHandler(webapp.RequestHandler):
  @decorator.oauth_aware
  def get(self):
    if decorator.has_credentials():
      http = decorator.http()
      followers = service.people().list(
      ...
    else:
      link = decorator.authorize_url()
```

So you can put a link for the user to follow along next to an explanation of why you are requesting access to their data.

The End

That's it for this overview. Here's some further reading.

There's more info on the wiki:
https://developers.google.com/api-client-library/python/guide/aaa_oauth

And PyDoc for all the classes we've talked about:
- OAuth2WebServerFlow
- Credentials
- StorageByKeyName
- OAuth2Decorator