Data Mining - Preliminaries

Bruno Gonçalves
www.bgoncalves.com

Python Modules

- urllib2
- posixpath
- requests
- BeautifulSoup
- gzip
- json
- OAuth

urllib2

http://docs.python.org/2/library/urllib2.html

- Extensible library for opening and manipulating URLs
- https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw
  - https <- protocol
  - foursquare.com <- server
  - /tyayayayaa/checkin/5304b652498e734439d8711f <- resource within server
  - s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw <- Query string

```
import urllib2

url = "https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw"

parsed = urllib2.urlparse.urlparse(url)
query = parsed.query
query_dict = urllib2.urlparse.parse_qs(query)
```

urllib2

http://docs.python.org/2/library/urllib2.html

- urllib2.urlopen(url) opens a url for reading and returns a "file handle"-like object
- Information about the webpage can be obtained with the .info() method in the form of an HTTPMessage
- The .geturl() method returns the final location of the webpage.
  .urlopen() follows redirects until it connects with the final content.
- .getcode() returns the status code of the call
  - 200 OK
  - 404 File Not Found
  - 500 Internal Server Error
- webserver returned headers can be found in the dict field inside the HTTPMessage object

Challenge - urllib2

- Find the final location of the shortened url:

http://bit.ly/GoogleScholar

- and access the headers and info of the request

```
import urllib2

url = "http://bit.ly/GoogleScholar"

webpage = urllib2.urlopen(url)
code = webpage.getcode()
info = webpage.info()

headers  = info.dict

new_url = webpage.geturl()
```

posixpath

http://docs.python.org/2/library/os.path.html

- Manipulate paths in a POSIX operating system
- Also useful to extract information from remote resource paths
- Aliased to os.path if your operating systems is POSIX
- https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f -> Path in remote filesystem
- .basename(path) -> returns the file name (if there is one) 5304b652498e734439d8711f
- .dirname(path) -> return the directory portion /tyayayayaa/checkin

posixpath

http://docs.python.org/2/library/os.path.html

```
import urllib2
import posixpath

url = "https://foursquare.com/tyayayayaa/checkin/5304b652498e734439d8711f?s=ScMqmpSLg1buhGXQicDJS4A_FVY&ref=tw"

parsed = urllib2.urlparse.urlparse(url)
filename = posixpath.basename(parsed.path)
directory = posixpath.dirname(parsed.path)
```

requests

http://requests.readthedocs.org/en/latest/

- "HTTP for Humans" - Simplified HTTP requests:
    - authentication (basic authentication, OAuth1, OAuth2, etc)
    - header manipulation
    - error handling
    - etc...

requests

http://requests.readthedocs.org/en/latest/

- .get(url) open the given url for reading (like .urlopen()) and returns a Response
- .get(url, auth=("user", "pass")) open the given url and authenticate with username="user" and password="pass" (Basic Authentication)
- Response.status_code is a field that contains the calls status code
- Response.headers is a dict containing all the returned headers
- Response.text is a field that contains the content of the returned page
- Response.url contains the final url after all redirections
- Response.json() parses a JSON response (throws a JSONDecodeError exception if response is not valid JSON). Check "content-type" header field.

requests

http://requests.readthedocs.org/en/latest/

```
import requests
import sys

url = "http://httpbin.org/basic-auth/user/passwd"

request = requests.get(url, auth=("user", "passwd"))

if request.status_code != 200:
    print >> sys.stderr, "Error found", request.get_code()

content_type = request.headers[“content-type"]

response = request.json()

if response["authenticated"]:
    print "Authentication Successful"
```

requests

http://requests.readthedocs.org/en/latest/

```
import requests
import sys

url = "http://httpbin.org/basic-auth/user/passwd"

request = requests.get(url, auth=("user", "passwd"))

if request.status_code != 200:
    print >> sys.stderr, "Error found", request.get_code()

content_type = request.headers[“content-type"]

response = request.json()

if response["authenticated"]:
    print "Authentication Successful"
```

- .get(url, headers=headers) open the url for reading but pass “headers” contents to server when establishing a connection
    - Useful to override default values of headers or add custom values to help server decide how to handle our request
    - Most commonly used to spoof the user-agent

requests

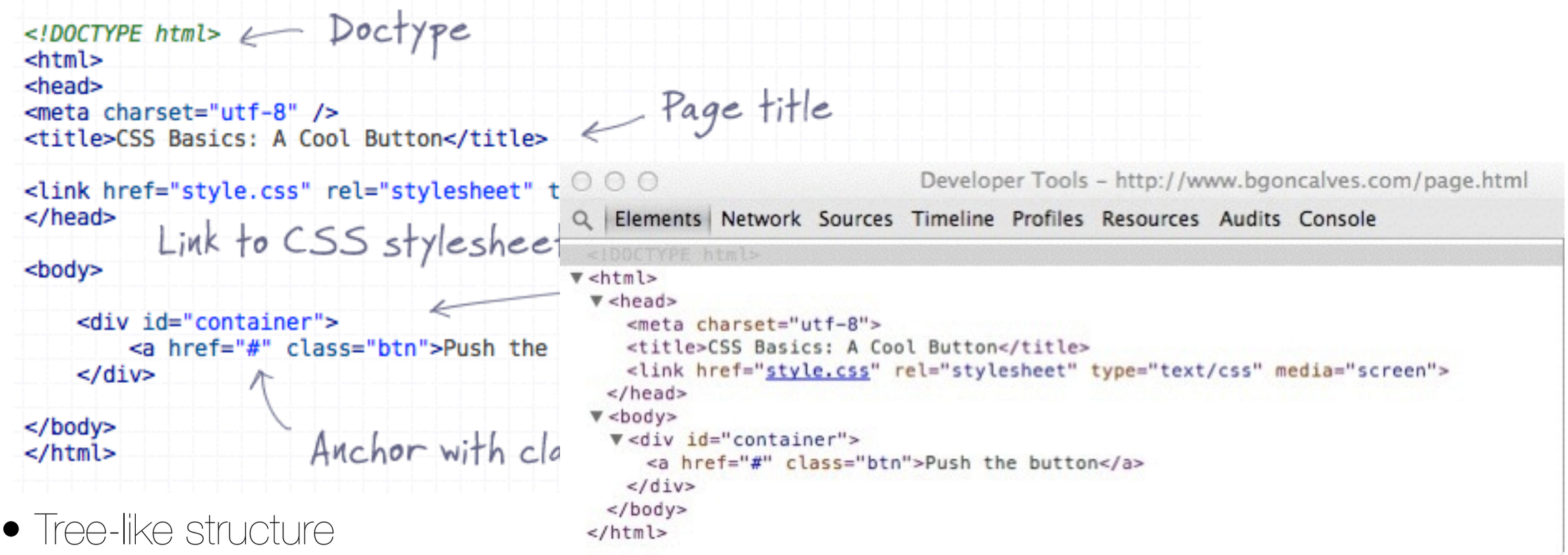
http://requests.readthedocs.org/en/latest/

```
import requests
from BeautifulSoup import BeautifulSoup

url = "http://whatsmyuseragent.com/"

headers = {"User-agent" : "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:25.0) Gecko/20100101 Firefox/25.0"}

request_default = requests.get(url)
request_spoofed = requests.get(url, headers=headers)

soup_default = BeautifulSoup(request_default.text)
soup_spoofed = BeautifulSoup(request_spoofed.text)

print "Default:", soup_default.h2.text
print "Spoofed:", soup_spoofed.h2.text
```

- Some servers use the User-agent string to decide how to format the output
  - Correctly handle specific versions of web browsers
  - Provide lighter/simplified versions to users on their mobiles
  - Refusing access to automated tools, etc

Basic Structure of a web page

http://www.w3schools.com/tags/



- Tree-like structure
- Nested <tags> with attributes and content
- Two main sections under <html>:
  - <head> - meta data and resource location
  - <body> - page contents

BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

- Parses html and xml files into a tree.
- **BeautifulSoup(page)** where page is a string or a "file handle"-like object
- BeautifulSoup parses the contents of the page and returns a BeautifulSoup object, corresponding to the root of the document tree.
- Each leaf of the tree is a Tag object:
  - can be used as dicts to access tag attributes,
  - contains pointers to children (**.findChildren()**), siblings (**.findSiblings()**) and parent (**.findParent()**)
  - can be accessed recursively by name (**head.title.content**)
  - modifying the contents of a tag modifies the contents of the document

BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
import requests
from BeautifulSoup import BeautifulSoup

url = "http://www.bgoncalves.com/page.html"

request = requests.get(url)

soup = BeautifulSoup(request.text)

print "The title tag is", soup.title
print "The id of the div is", soup.div["id"]

soup.div["id"] = "new_id"

print "And now it's", soup.body.div["id"]
```

- .findAll() returns a list of all tags matching a certain criteria
  - .findAll(name="a") find all "a" tags (links)
  - .findAll(name=["a", "div"]) find all "a" and "div" tags
  - .findAll(attrs = {"class": "btn"}) find all tags with class "btn", regardless of tag name
  - .findAll(name="a", attrs = {"class": "btn"}, limit=2) find the first two "a" tags with class "btn"

Challenge - Feynman's top 100 papers

- Extract the title of Feynman's 100 most cited papers from Google Scholar

Richard Feynman

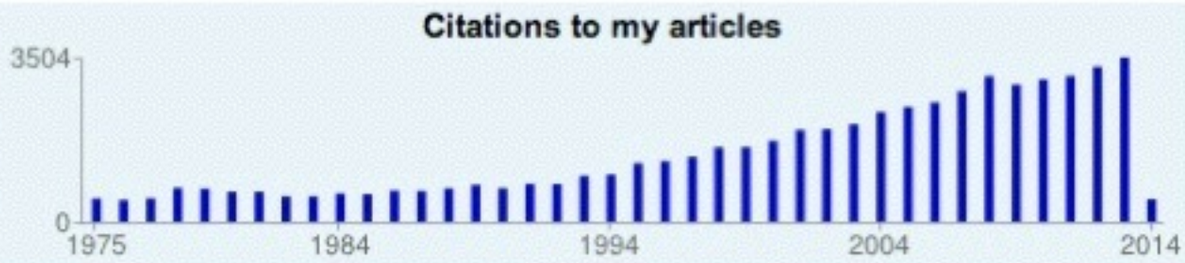
California Institute of Technology

quantum mechanics - quantum electrodynamics

No verified email

Citation indices

	All	Since 2009
Citations	64274	16335
h-index	56	40
i10-index	87	67



BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
import requests
from BeautifulSoup import BeautifulSoup

url = "http://scholar.google.com/citations?hl=en&user=B7vSqZsAAAAJ&view_op=list_works&pagesize=100"

request = requests.get(url)
soup = BeautifulSoup(request.text)

table = soup.find("table", attrs={"class" : "cit-table"})

for paper in table.findAll("td", attrs={"id": "col-title"}):
    print paper.a.string
```

gzip

https://docs.python.org/2/library/gzip.html

- Module to support compressed files
- Syntax similar to normal file handlers
- .open(filename, mode)
  - Returns a file handler that works the same as a normal file.
  - Mode defaults to "r"-ead, can also be "w"-rite and "b"-inary
- .close()

json

- JavaScript Object Notation - Serialization format originally developed for Javascript
- Currently widely accepted format for data dissemination
- Most languages have excellent libraries to handle it
- .loads(obj_str) - load JSON data from a string - returns native Python object
- .load(fp) - load JSON data from a file handle - returns native Python object
- .dumps(obj) - convert JSON data to a string
- .dump(obj, fp) - write the string version of obj to the file handle fp

Challenge

- Download the compressed JSON file:

http://data.githubarchive.org/2015-01-01-15.json.gz

- and write a script that reads the file and extracts all the actor id and repository id pairs

```
import gzip
import sys
import json

for line in gzip.open(sys.argv[1]):
    data = json.loads(line.strip())

    print data["actor"]["id"], data["repo"]["id"]
```

OAuth

OAuth 1

http://hueniverse.com/oauth/

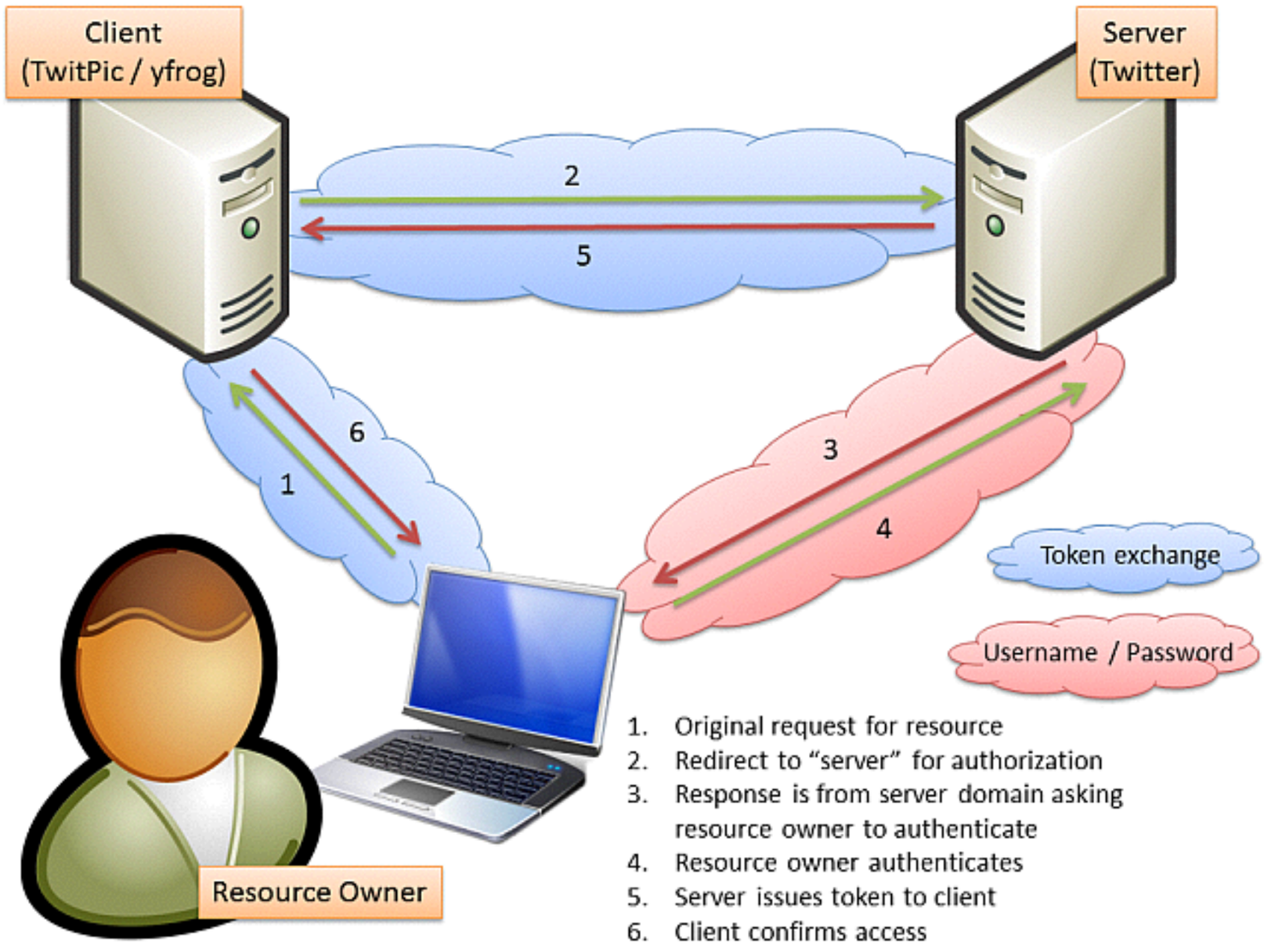
- "An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications."

- The idea is to allow for a safe way to share privileges without divulging private credentials

  - Give XPTO Application permission to post to your Twitter account without having to trust the developers of XPTO with your username/password and while being able to unilaterally revoke privileges.

OAuth 1

http://hueniverse.com/oauth/



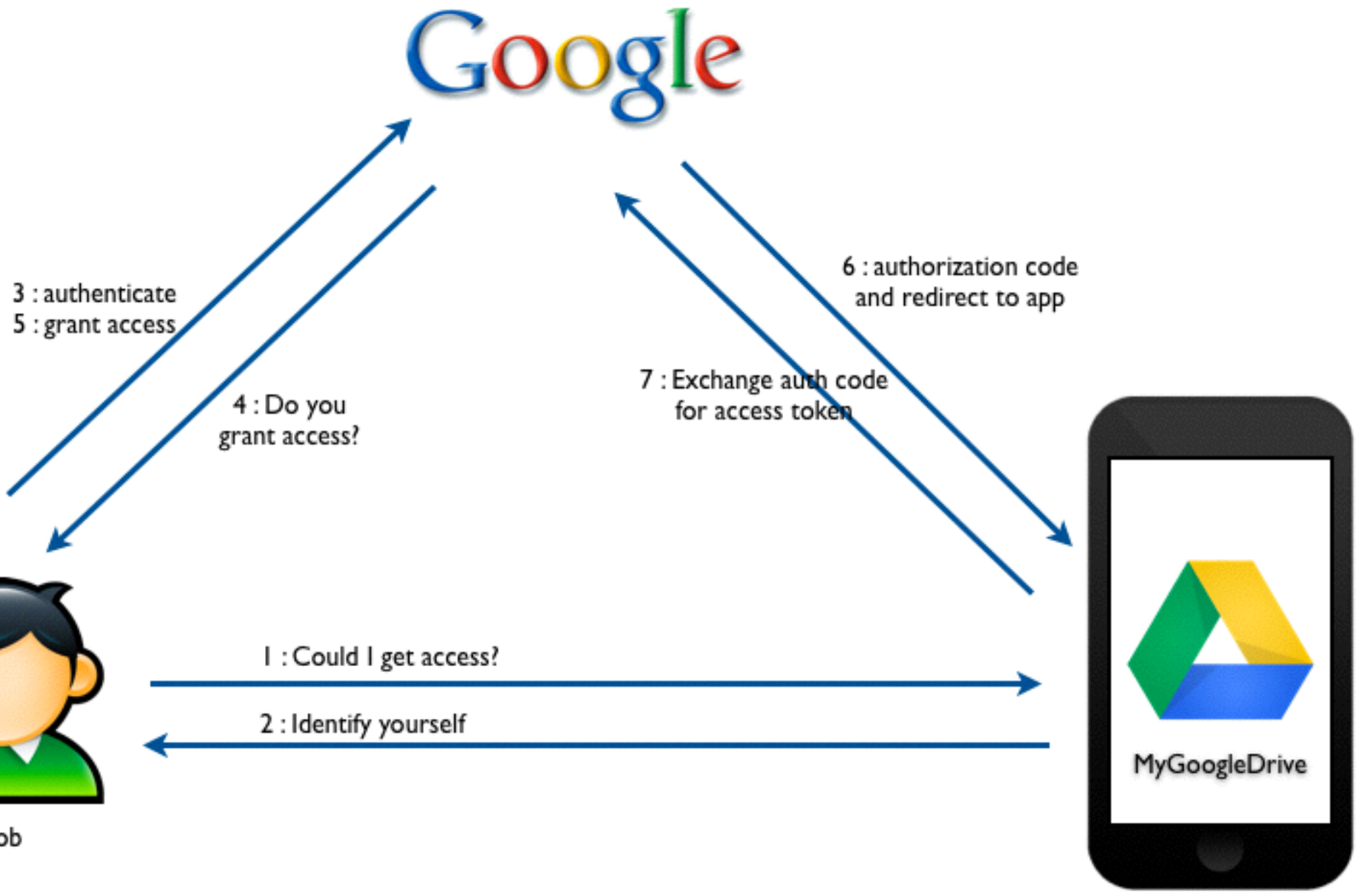
1. Original request for resource
2. Redirect to “server” for authorization
3. Response is from server domain asking resource owner to authenticate
4. Resource owner authenticates
5. Server issues token to client
6. Client confirms access

OAuth 1

http://hueniverse.com/oauth/

- After the "OAuth dance" is concluded, client application has two sets of keys:
  - one that uses to identify itself as a valid application (api_key, api_secret)
  - one that uses to identify the user it wants to access (token, token_secret)
- You can revoke access at any time by letting the token provider that a given app is no long authorized (invalidating token and token_secret).

OAuth 2


Google
3 : authenticate
5 : grant access
4 : Do you
grant access?
6 : authorization code
and redirect to app
7 : Exchange auth code
for access token
1 : Could I get access?
2 : Identify yourself
Bob
MyGoogleDrive

OAuth 2

- Latest version of OAuth protocol
    - Similar "dance" required
    - Allows for "bearer tokens" - access is given to anyone able to provide a valid token without any further restrictions or authentication
        - access tokens are provided along with the request for the resource through a secure connection
        - tokens can expire automatically
- We will use both OAuth and OAuth2 over the next few days