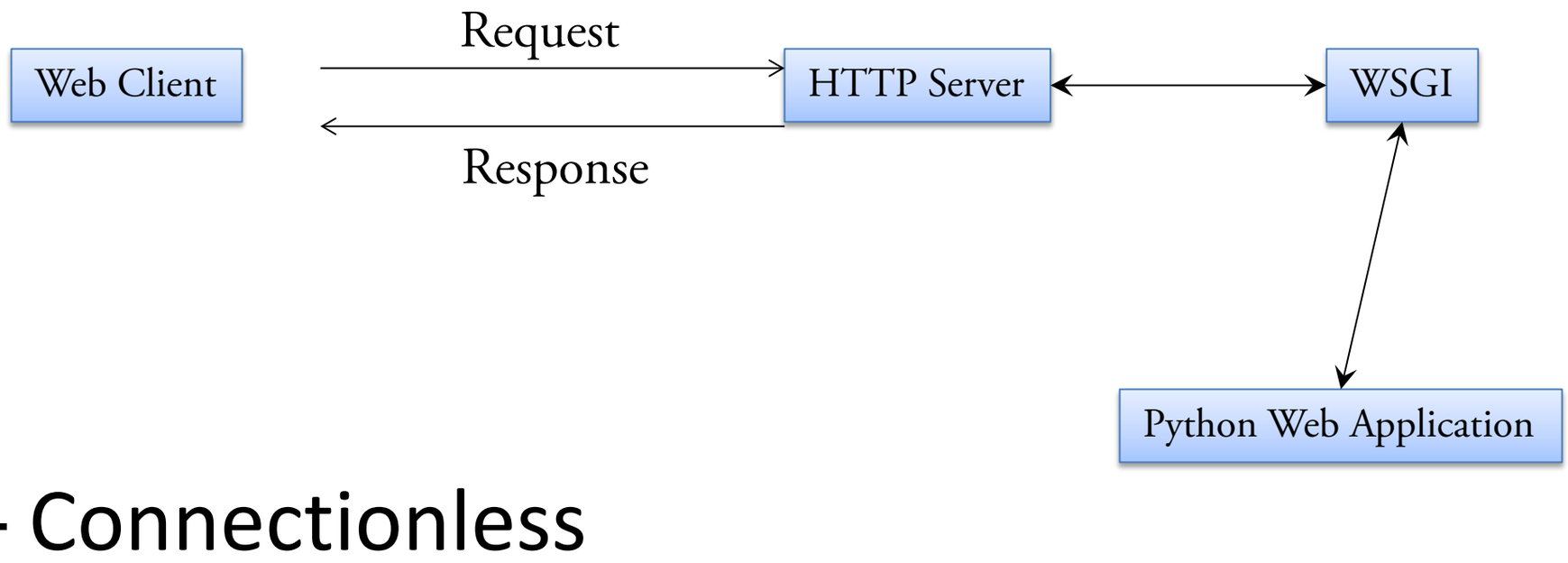
WWW

HTTP, Ajax, APIs, REST

HTTP

- Hypertext Transfer Protocol



  - Connectionless
  - Media Independent
  - Stateless

WSGI : Web Server Gateway Interface

HTTP Methods

- Most common: GET, POST, HEAD
  - GET: retrieve data from the server
    - Form submissions can be encapsulated in URLs
  - HEAD: like GET, but just get the headers from the server
  - POST: Used to send data to the server
    - Query Length can be unlimited (unlike in GET)
    - Can be used to send entire files
    - Form data is attached to the end of POST request

Other Methods: PUT and DELETE

GET Demo

- A simple example

```
$telnet matrix.cs.fsu.edu 80
Trying 192.168.122.23...
Connected to matrix.cs.fsu.edu (192.168.122.23).
Escape character is '^]'.
GET /~piyush/ HTTP/1.0
```

```
HTTP/1.1 200 OK
Date: Thu, 14 Apr 2011 13:34:39 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Sat, 10 Apr 2010 21:20:35 GMT
ETag: "6400cf-3fc-483e87b87eec0"
Accept-Ranges: bytes
Content-Length: 1020
Connection: close
Content-Type: text/html; charset=UTF-8
```

Response Headers

```
…
</html>
Connection closed by foreign host.
```

GET Authentication

- Basic Authentication (Another form: digest auth) – apache

```
$telnet matrix.cs.fsu.edu 80
Trying 192.168.122.23...
Connected to matrix.cs.fsu.edu (192.168.122.23).
Escape character is '^]'.
GET /~piyush/teach/py10/comments/ HTTP/1.0
```

```
HTTP/1.1 401 Authorization Required
Date: Thu, 14 Apr 2011 13:39:50 GMT
Server: Apache/2.2.3 (CentOS)
WWW-Authenticate: Basic realm="Restricted Files"
Content-Length: 483
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

…

GET Headers

- Specify *Request headers*: ☺

```
$telnet youruseragent.info 80
Trying 72.14.181.41...
Connected to youruseragent.info (72.14.181.41).
Escape character is '^]'.
GET /what-is-my-user-agent HTTP/1.0
Accept:application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;
Accept-Charset:ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding:gzip,deflate,sdch
Accept-Language:en-US,en;q=0.8
Connection:keep-alive
Host:youruseragent.info
User-Agent:Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/5
```

- HTTP client identifies itself using User-Agent

Ip2location.com can turn this into where you are?

Response

- Server Responds

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 14 Apr 2011 13:56:48 GMT
Content-Type: text/html;charset=utf-8
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
X-Content-Parsed-By: phpCMS 1.2.2
Content-Encoding: gzip
Last-Modified: Thu, 14 Apr 2011 13:56:48 GMT
Content-Length: 3203
```

Web Client

- urllib : You've already used it…
  - http, ftp, https, …

```
>>> f = urllib.urlopen("https://www.google.com")
>>> data = f.read()
```

A File like object

  - Other options: urllib2, mechanize, …
  - urllib2 provides urlopen as well + much more.

Rendering html

- A simple and quick html renderer for your html data:
- urllib2 can be used in place of urllib

```
from PyQt4.QtWebKit import QWebView
from PyQt4.QtGui import QApplication
import urllib2
import sys

url = "https://www.google.com"

app = QApplication(sys.argv)
web = QWebView()
web.setHtml(urllib2.urlopen(url).read())
web.show()
sys.exit(app.exec_())
```

urllib2

- Requests and Response are now objects
  - request = urllib2.Request('http://compgeom.com/~piyush/')
  - response = urllib2.urlopen(request)
  - data =response.read()

- Requests can have additional data
  - HTTP headers (helps emulating User-Agents)
  - Authentication
  - User data (POST)

- Automatically handles redirections.

Changing headers

```
>>> request = urllib2.Request('http://compgeom.com/~piyush/')
>>> request.add_header('If-Modified-Since', 'Mon, 11 Apr 2011 04:00:08 GMT')
>>> request.add_header('User-Agent', 'My supercool client')
>>> data = urllib2.urlopen(request).read()
```

- Apache Server Log:
  - 68.237.112.224 - - [14/Apr/2011:11:36:49 -0400] "GET /~piyush/ HTTP/1.1" 200 1020 "-" "My supercool client"

Handling exceptions

- Exception Classes: IOError → URLError → HTTPError
- Most errors raised by urllib2 will be caught in these classes
- Rarely, you might see other errors
- Catching urllib2 errors:

```
import sys
from urllib2 import Request, urlopen

req = Request("http://hello-world.compgeom.com")

try:
    response = urlopen(req)

except IOError as e:
    if hasattr(e, 'reason'):
        print 'Server Unreachable'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'Server did not fulfill the request.'
        print 'Error code: ', e.code
except:
    print "Unexpected Error!", sys.exc_info()[0]
    raise
else:
    print("So far so good.")
    print response.read()
```

```
Server Unreachable
Reason:  [Errno 11004] getaddrinfo failed
```


AJAX
Asynchronous Javascript And XML

What is AJAX ?

- Asynchronous Javascript and XML.
- Not a stand-alone language or technology.
- It is a technique that combines a set of known technologies in order to create faster and more user friendly web pages.
- It is a client side technology.

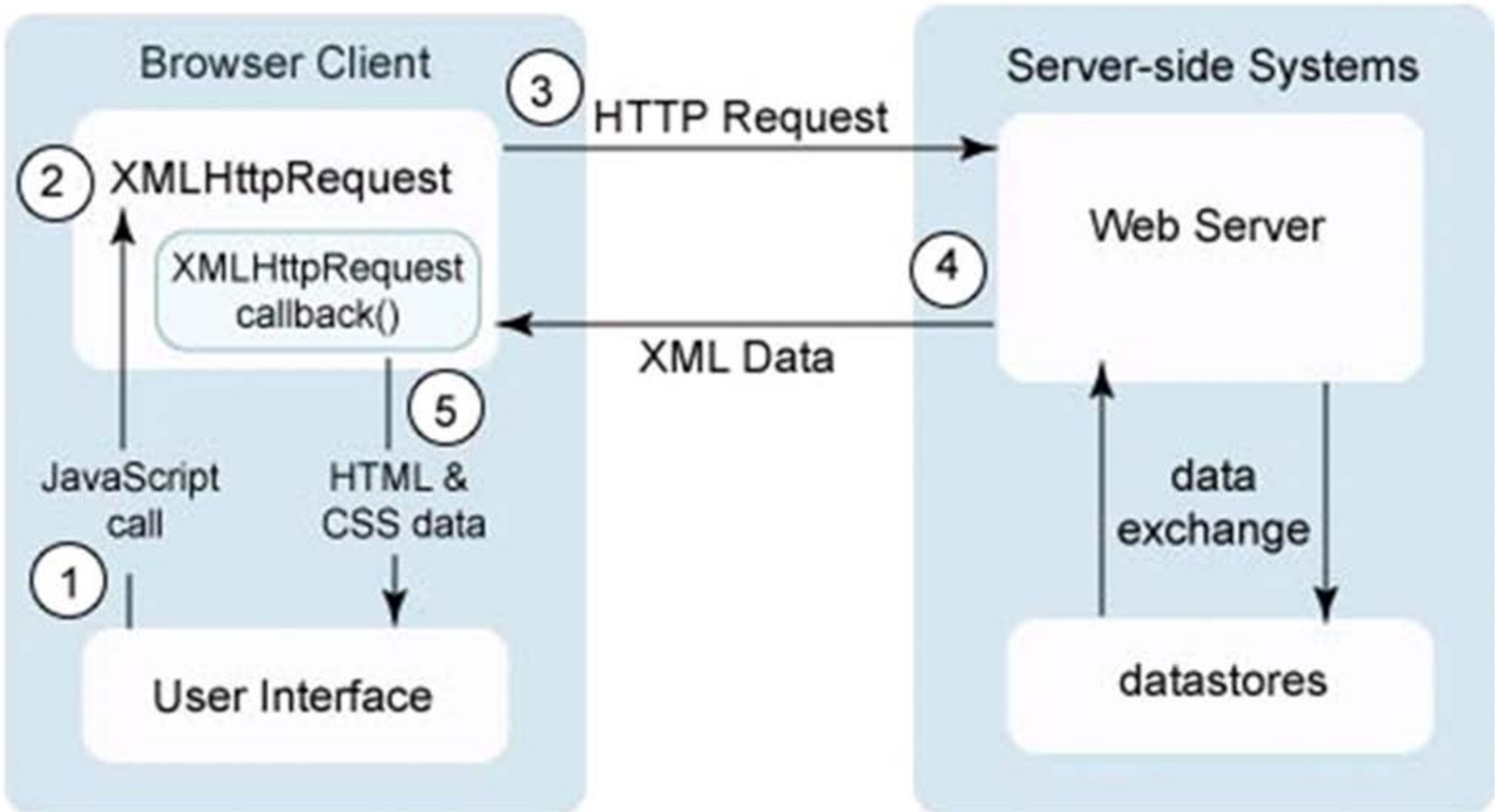
Purpose of AJAX

- Prevents unnecessary reloading of a page.
- When we submit a form, although most of the page remains the same, whole page is reloaded from the server.
- This causes very long waiting times and waste of bandwidth.
- AJAX aims at loading only the necessary information, and making only the necessary changes on the current page without reloading the whole page.

Purpose of AJAX

- Connection between client side script and server side script.
- Better user experience
- More flexibility
- More options

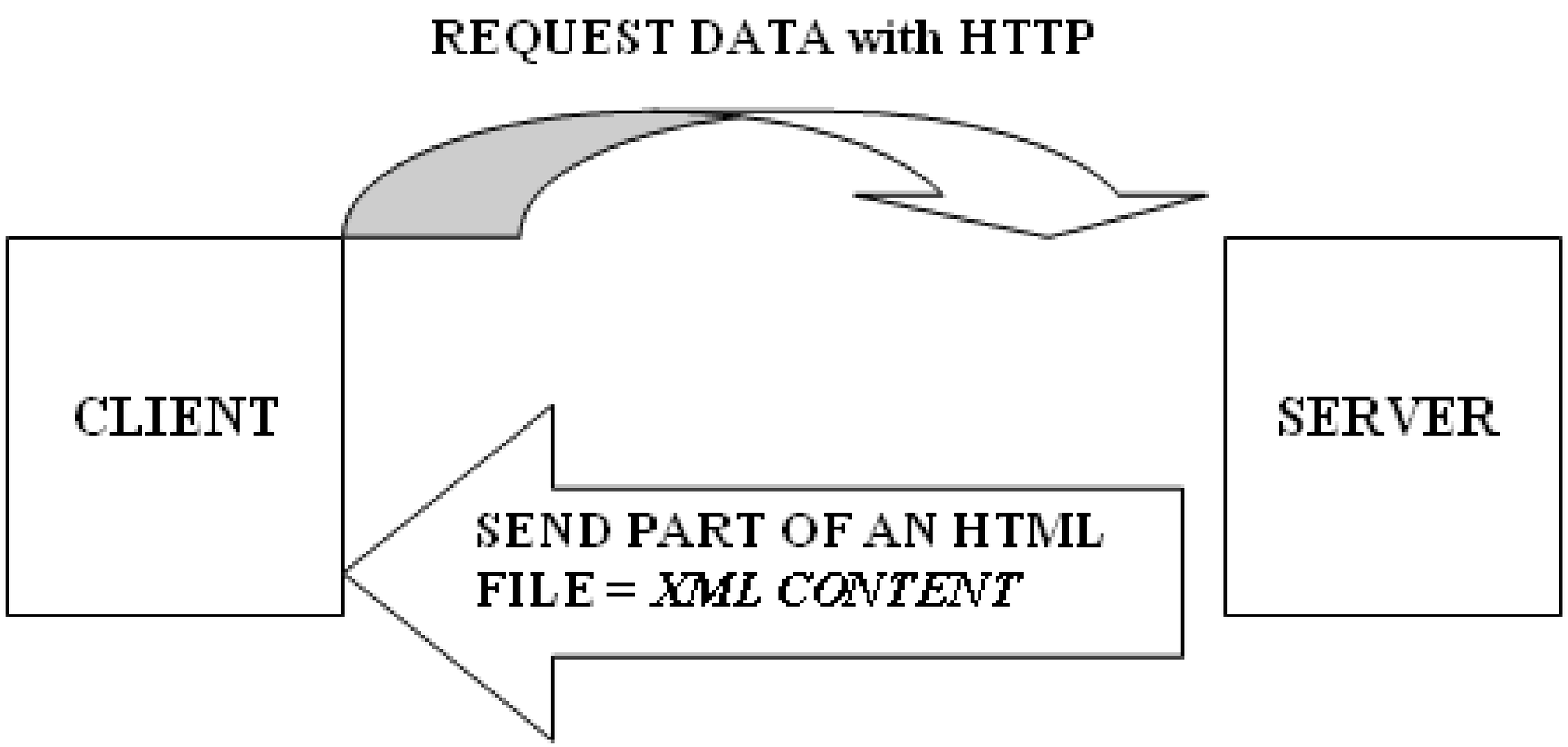
Big Picture


Browser Client
Server-side Systems
3
HTTP Request
2
XMLHttpRequest
XMLHttpRequest
callback()
4
XML Data
Web Server
5
JavaScript
call
HTML &
CSS data
data
exchange
1
User Interface
datastores

Simple Processing

- AJAX is based on Javascript, and the main functionality is to access the web server inside the Javascript code.
- We access to the server using special objects; we send data and retrieve data.
- When user initiates an event, a javascript function is called which accesses server using the objects.
- The received information is shown to the user by means of the Javascript's functions.

Data Exchange in AJAX



Data Exchange in AJAX

Examples

- Example 1

http://www.w3schools.com/ajax/ajax_example.asp

- Another example

http://www.w3schools.com/ajax/ajax_database.asp

- Therefore, by using AJAX, unnecessary exchange of data is prevented, web pages become:
  - More interactive
  - Faster
  - More user friendly

API

- Application Programming Interface
  - A protocol intended to be used as an interface by software components to communicate with each other.

- Source code interface
  - For library or OS
  - Provides services to a program

- At its base, like a header file
  - But, more complete

Why is API Important

- Company View
  - Can be asset – big user investment in learning and using
  - Bad design can be source of long-term support problems
- Once used, it’s tough to change
  - Especially if there are several users
- Public APIs – One chance to get it right

APIs are Everywhere

- Remote Procedure Calls (RPCs)
- File transfer
- Message delivery
- Java APIs

Characteristics of APIs

- Easy to learn
- Easy to use even without documentation
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to extend
- Appropriate to audience

REST

- Representational State Transfer
  - Web API design model
  - Software architecture for distributed systems
  - Rules for Clients/Servers

REST

- Constraints
  - Uniform interface separates Client / Server
  - Stateless
  - Cacheable
  - Layered System

RESTful web API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://example.com/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as `http://example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection.

to be continued...