

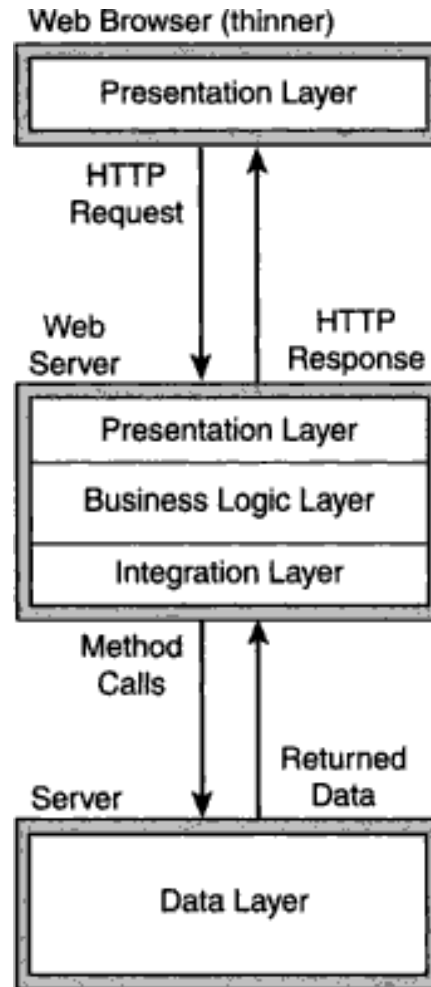
*ECOM 5416 — Parallel and
Distributed Systems*

Cloud Computing

**Web Application Development Server
Side**

Lec: 03

Architecture





Flask is a microframework for Python.

<http://flask.pocoo.org/>

“Easy” to learn

Plug-in architecture provides rich functionality

Hello World

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run()
```

Client-Handler Interaction

- Clients request the execution of a handler program by means of a:
 - A request method (e.g., GET, POST)
 - Universal Resource Identifier (URI)
 - Identifies handler
 - Extra arguments

URI Syntax

<protocol>://<host><port>/<path-info><script>"?"<query-string>

<http://finance.yahoo.com/add?op1=10&op2=20>

<protocol>	http
<host>	finance.yahoo.com
<port>	80
<path-info>	null
<script>	add
<query-string>	op1=10, op2=20

RFC 2396 Encoding

- All arguments

- Single string of ampersand (&) separated name=value pairs

name_1=value_1&name_2=value_2&...

- Spaces

- Replaced by a plus (+) sign

- Other characters (ie, =, &, +)

- Replaced by a percent sign (%) followed by the two-digit hexadecimal ASCII equivalent

Method

- GET
 - Arguments appear in the URL after the ?
 - Can be expressed as a URL
 - Limited amount of information can be passed this way
 - URL may have a length restriction on the server
 - Arguments appear on server log
- POST
 - Arguments are sent in the HTTP message body
 - Cannot be expressed as URL
 - Arbitrarily long form data can be communicated (some browsers may have limits (i.e. 7KB)).
 - Arguments usually does not appear in server logs.

Flask request object

- Provides access to http request arguments and environment attributes

form parsed form data from POST or PUT requests

args parsed contents of the query string

values contents of both form and args

cookies contents of all cookies transmitted with request

environ the underlying WSGI environment

method current request method

__init__.py

```
from flask import Flask
```

```
webapp = Flask(__name__)
```

```
from app import hello_v2
```

```
from app import example1
```

```
from app import example2
```

❏

run.py

```
#!/venv/bin/python  
from app import webapp  
webapp.run(debug=True)
```



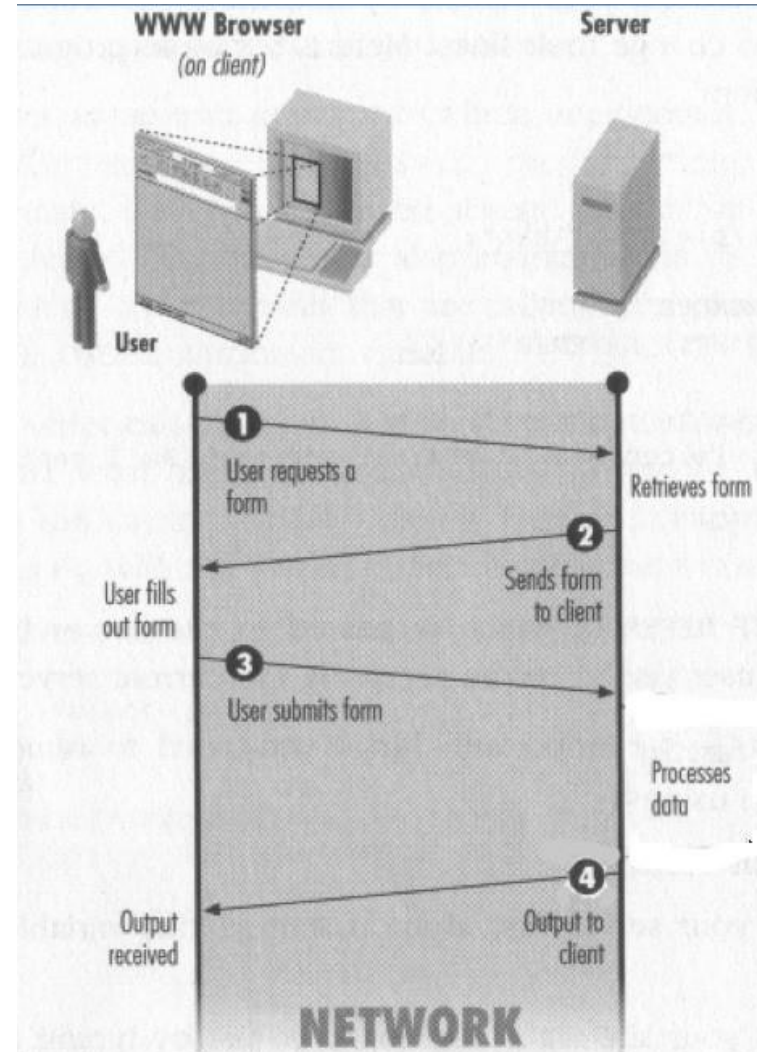
hello_v2.py

```
from app import webapp

@webapp.route('/')
def hello_world():
    return 'Hello, World!'
```

Forms

- Specify request method in “method” attribute
- Automatically encodes all **named** field values



Session Maintenance

- HTTP is stateless by default
 - No memory of browsers state
- Motivations for preserving state?
 - Enabling multi-step interactions with users
- Examples:
 - access control (i.e., login)
 - shopping cart
 - web site analytics

Flask Sessions

Data stored on client as encrypted cookie

Initialize session

```
app.secret_key = '\x8\xbeHJ:\x9f\x0!\xb1a\xaa\x0f\xee'
```

Set session attribute

```
session["loggedIn"] = True
```

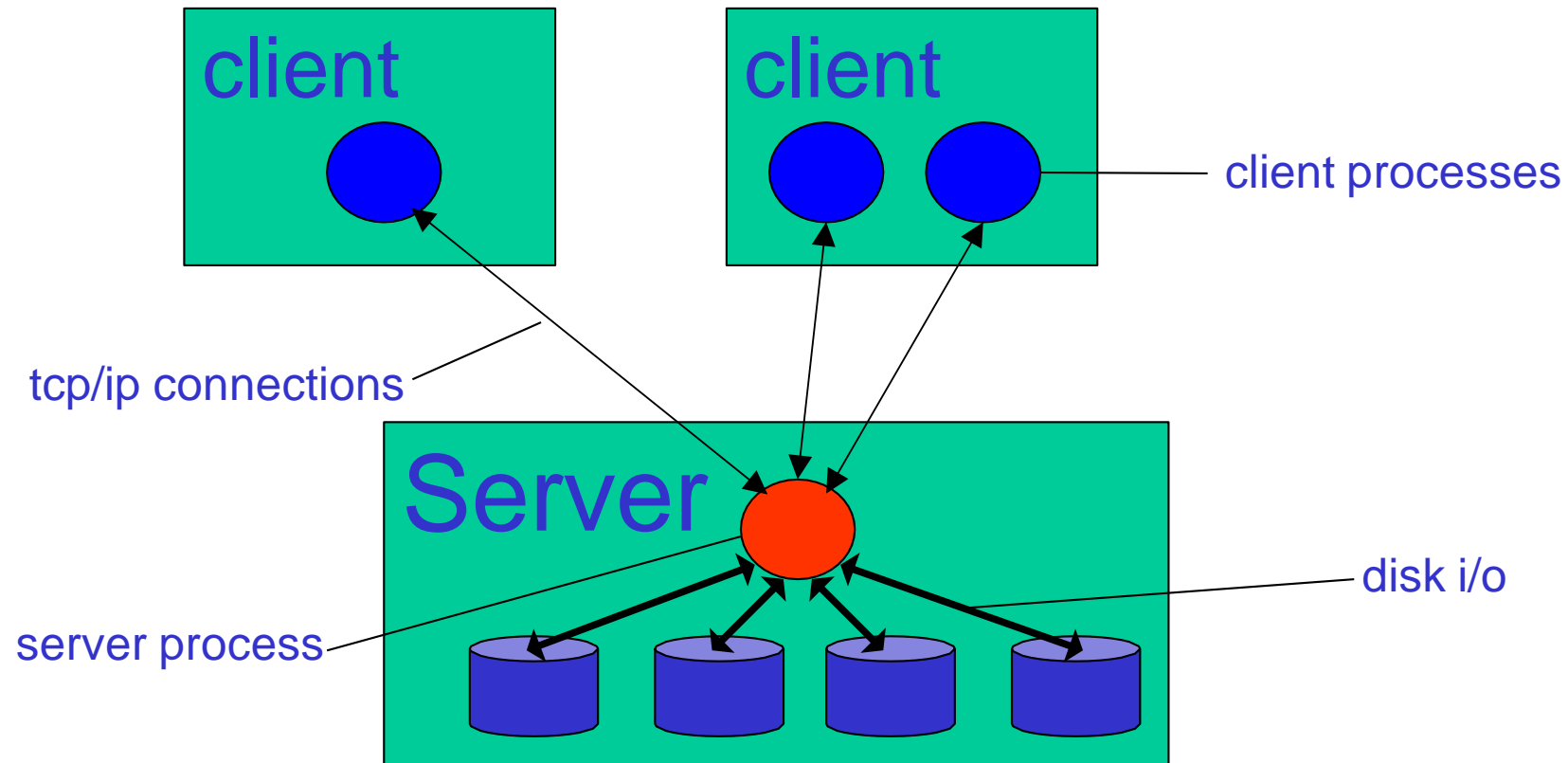
Retrieve session attribute

```
count = session["count"]
```

Invalidate session

```
session.clear()  
session.pop("count")
```

Client/Server Databases



MySQL Python Connector

- Standard SQL database access interface.
- Allows a Python program to issue SQL statements and process the results.
- Defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata.

API: Connection

```
import mysql.connector
```

```
db_config = {      'user': 'XXX',  
                   'password': 'some_password',  
                   'host': '127.0.0.1',  
                   'database': 'estore'}
```

```
db = mysql.connector.connect(user=db_config['user'],  
                             password=db_config['password'],  
                             host=db_config['host'],  
                             database=db_config['database'])
```

```
db.close()
```

API: Executing Queries

- A query can return many rows, each with many attributes
- Steps are

- 1 Send query to the database

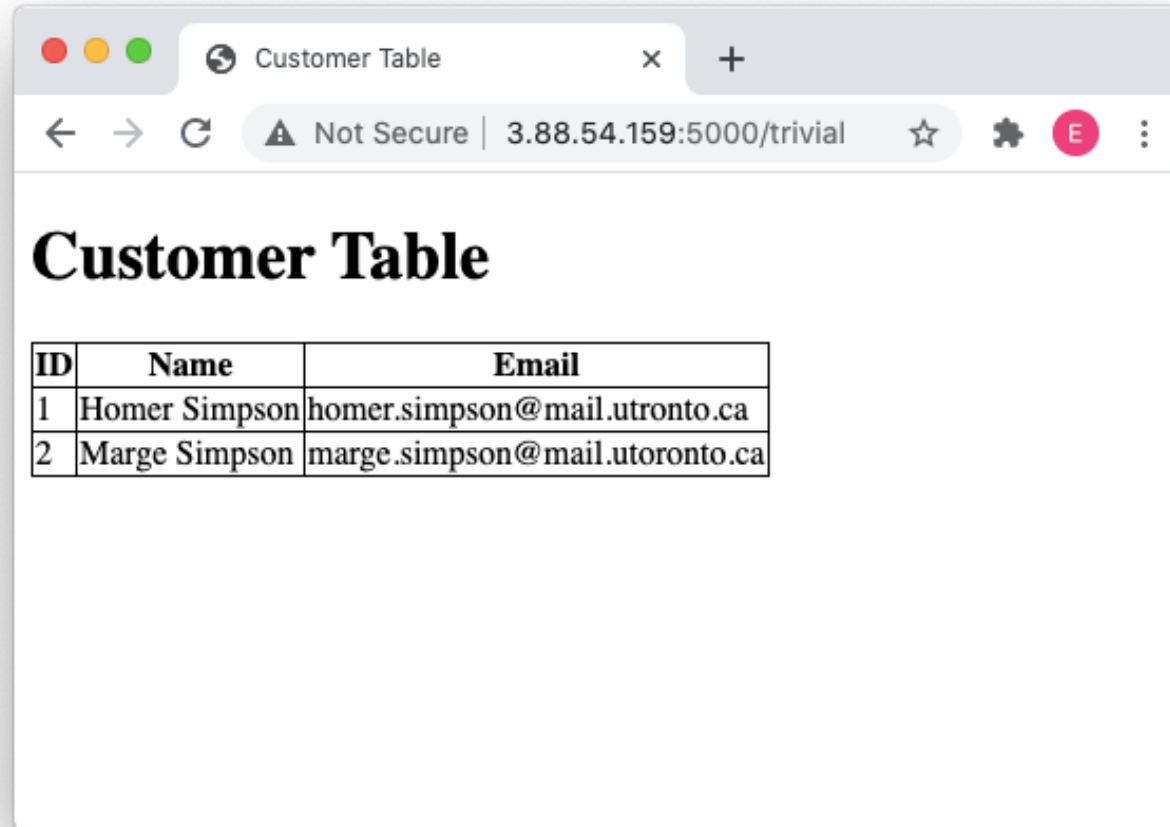
```
cursor = cnx.cursor()  
query = 'SELECT * FROM customer'  
cursor.execute(query)
```

- 2 Retrieve one row at a time

- 3 For each row, retrieve attributes

```
for row in cursor:  
    print(row[0])
```

Trivial Example



A screenshot of a web browser window. The title bar shows 'Customer Table' and a close button. The address bar shows 'Not Secure | 3.88.54.159:5000/trivial' with navigation icons and a star. The main content area has the heading 'Customer Table' and a table with three columns: ID, Name, and Email. The table contains two rows of data.

ID	Name	Email
1	Homer Simpson	homer.simpson@mail.utronto.ca
2	Marge Simpson	marge.simpson@mail.utoronto.ca

trivial.py

```
1 from flask import render_template
2 from app import webapp
3
4 import mysql.connector
5
6
7
8 @webapp.route('/trivial', methods=['GET'])
9 # Display an HTML list of all product.
10 def trivial():
11
12     cnx = mysql.connector.connect(user='XXX',
13                                   password='secret',
14                                   host='127.0.0.1',
15                                   database='estore')
16
17     cursor = cnx.cursor()
18     query = "SELECT * FROM customer"
19     cursor.execute(query)
20     view = render_template("trivial.html", title="Customer Table", cursor=cursor)
21     cnx.close()
22     return view
```

trivial.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>{{title}}</title>
5      <style>
6        table, tr, td, th {
7          border: 1px solid black;
8          border-collapse: collapse;
9        }
10     </style>
11  </head>
12  <body>
13    <h1>{{title}}</h2>
14    <table>
15      <thead>
16        <th>ID</th><th>Name</th><th>Email</th>
17      </thead>
18      {% for row in cursor %}
19        <tr>
20          <td>{{ row[0]}}</td><td>{{ row[1]}}</td><td>{{ row[2]}}</td>
21        </tr>
22      {% endfor %}
23    </table>
24  </body>
25 </html>
```

Transactions

- Definition: A transaction is a collection of DB modifications, which is treated as an atomic DB operation.
 - Transactions ensure that a collection of updates leaves the database in a consistent state (as defined by the application program); all updates take place or none do.
 - A sequence of **read** and **write** operations, terminated by a **commit** or **abort**
- Definition: Committed
 - A transaction that has completed successfully; once committed, a transaction cannot be undone
- Definition: Aborted
 - A transaction that did not complete normally
- Python Connector: By default in transactional mode: auto commit has been disabled, the method commit must be called explicitly; otherwise, database changes will not be saved.

Questions?