

CMPT 354: Database System I

Lecture 10. Application Programming

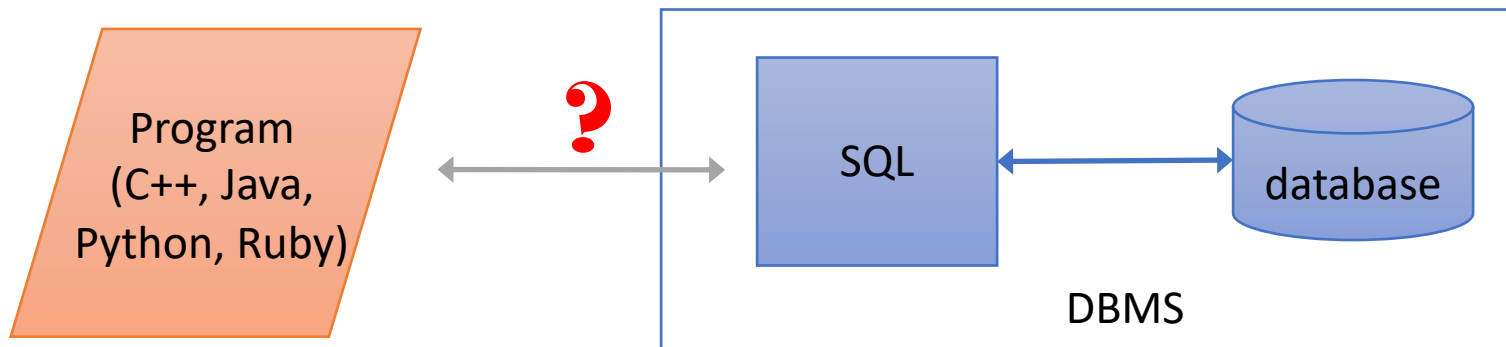
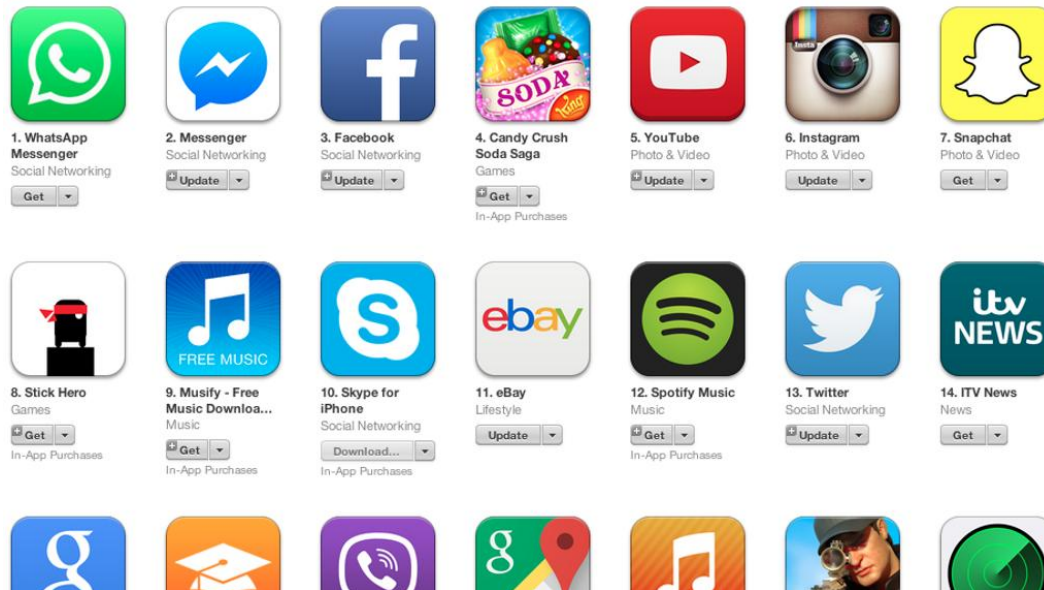
Why this lecture

- **DB designer:** establishes schema
- **DB administrator:** tunes systems and keeps whole things running
- **Data scientist:** manipulates data to extract insights
- **Data engineer:** builds a data-processing pipeline
- **DB application developer:** writes programs that query and modify a database

Outline

- Database Programming
- Application Architecture

Programming Environment



Many Database API options

- Fully embed into language (embedded SQL)
- Low-level library with core database calls (DB API)
- Object-relational mapping (ORM)
 - Ruby on rails, django, etc
 - define database-backed classes
 - magically maps between database rows & objects
 - magic is a double edged sword

Embedded SQL

- Extend host language (C/C++) with SQL syntax

```
4  int main() {
5      EXEC SQL INCLUDE SQLCA;
6      EXEC SQL BEGIN DECLARE SECTION;
7          int OrderID;      /* Employee ID (from user)      */
8          int CustID;       /* Retrieved customer ID      */
9          char SalesPerson[10] /* Retrieved salesperson name */
10         char Status[6]     /* Retrieved order status     */
11      EXEC SQL END DECLARE SECTION;
12
13      /* Prompt the user for order number */
14      printf ("Enter order number: ");
15      scanf_s("%d", &OrderID);
16
17      /* Execute the SQL query */
18      EXEC SQL SELECT CustID, SalesPerson, Status
19          FROM Orders
20          WHERE OrderID = :OrderID
21          INTO :CustID, :SalesPerson, :Status;
22
23      /* Display the results */
24      printf ("Customer number: %d\n", CustID);
25      printf ("Salesperson: %s\n", SalesPerson);
26      printf ("Status: %s\n", Status);
27      exit();
28 }
```

Declaring Variables

Embedded
SQL Query

Embedded SQL

CPP + embedded SQL



CPP + DB library calls



← DBMS library



Executable



Hard to maintain

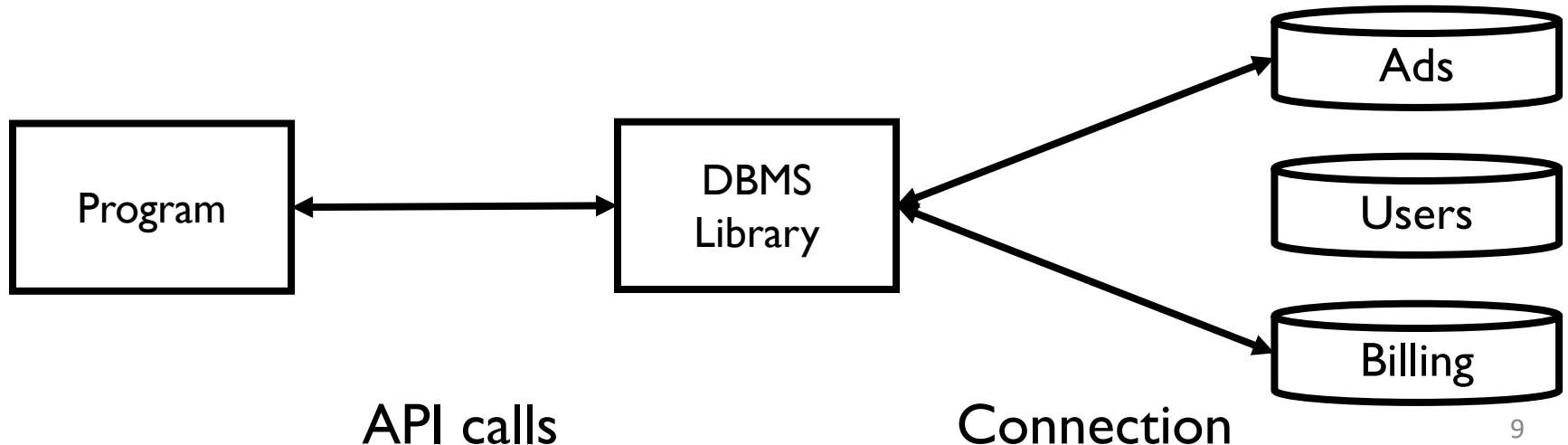
- What if SQL evolves?
- What if Compiler evolves?

Many Database API options

- Fully embed into language (embedded SQL)
- **Low-level library with core database calls (DB API)**
- Object-relational mapping (ORM)
 - Ruby on rails, django, Hibernate, sqlalchemy, etc
 - define database-backed classes
 - magically maps between database rows & objects
 - magic is a double edged sword

What does a library need to do?

- Single interface to possibly multiple DBMS engines
- Connect to a database
- Map objects between host language and DBMS
- Manage query results



ODBC and JDBC

- ODBC (Open DataBase Connectivity)



ODBC was originally developed by [Microsoft](#) and [Simba Technologies](#)

- JDBC (Java DataBase Connectivity)
 - Sun developed as set of Java interfaces
 - javax.sql.*

Connections

- Create a connection
 - Allocate resources for the connection
 - Relatively expensive to set up, libraries often cache connections for future use

```
conn = connect(sfu.db)
```

Should close connections when done! Otherwise resource leak.

Query Execution

```
foo = conn.execute("select * from student")
```

- Challenges
 - Type Mismatch
 - What is the return type of execute()?
 - How to pass data between DBMS and host language?

Type Mismatch

- SQL standard defines mappings between SQL and several languages

SQL types

CHAR(20)

INTEGER

SMALLINT

REAL

C types

char[20]

int

short

float

Python types

str

int

int

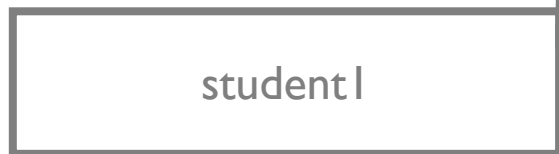
float

Cursor

- SQL relations and results are sets of records
- What is the type of foo?

```
foo = conn1.execute("select * from student")
```

- Cursor over the Result Set
 - similar to an iterator interface
 - Note: relations are unordered!
 - Cursors have no ordering guarantees
 - Use ORDER BY to ensure an ordering



cursor

Program



student 1
student 2
student 3
student 4
student 5
student 6
student 7
student 8
student 9
student 10

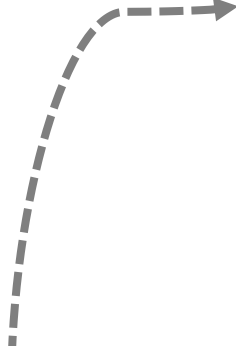
DBMS



student2

cursor

Program



DBMS

student1
student2
student3
student4
student5
student6
student7
student8
student9
student10

Cursor

- Cursor similar to an iterator
 - `cursor = conn.execute("SELECT * FROM student")`
- Cursor methods
 - `fetchone()`
 - `fetchall()`

Cursor

- Cursor similar to an iterator

```
1 conn = sqlite3.connect('sfu.db')
2 cursor = conn.execute("select * from student")
3
4 for record in cursor.fetchall():
5     print record
6
7 conn.close()
```

```
(u'Mary', 3.8)
(u'Tom', 3.6)
(u'Jack', 3.7)
```

SQL Injection!!

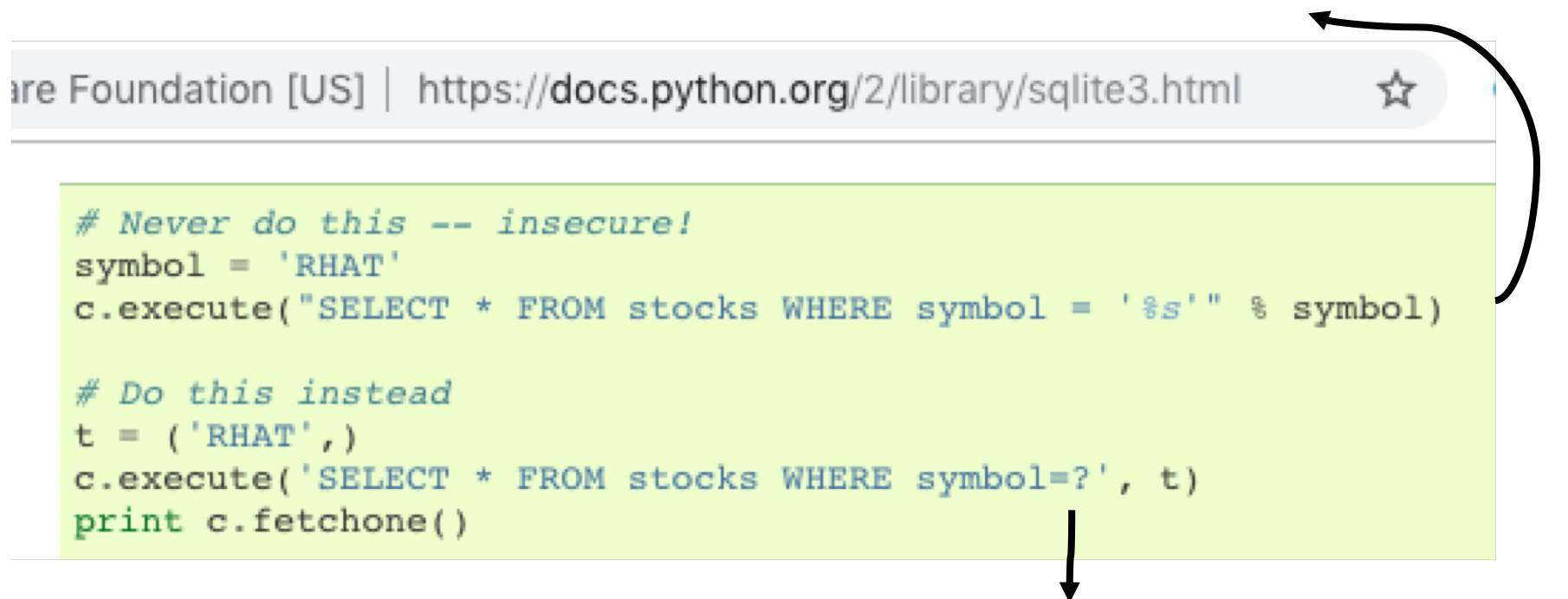
symbol = "RHAT' OR True -- "

SELECT * FROM stocks WHERE symbol = 'RHAT' OR True -- '

are Foundation [US] | <https://docs.python.org/2/library/sqlite3.html>

```
# Never do this -- insecure!
symbol = 'RHAT'
c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)

# Do this instead
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print c.fetchone()
```

A diagram illustrating the transition from an insecure SQL query to a secure one. A curved arrow points from the insecure query "SELECT * FROM stocks WHERE symbol = 'RHAT' OR True -- '" to a code block. Inside the code block, the insecure approach is shown as a comment, and the secure approach using parameterized queries is demonstrated. A straight arrow points from the secure code block to the secure SQL query "SELECT * FROM stocks WHERE symbol = 'RHAT' OR True -- '".

SELECT * FROM stocks WHERE symbol = 'RHAT' OR True -- '

Exercise

```
1 import sqlite3
2 conn = sqlite3.connect('sfu.db')
```

```
1 def search(name, conn):
2     """
3     Input:
4         @name: student name
5         @conn: database connection
6     Output:
7         @records: all the students whose name is @name
8     """
9     # REPLACE WITH YOUR CODE
```

```
1 name = 'Mary'
2 search(name, conn)
```

```
[(u'Mary', 3.8)]
```

Outline

- Database Programming
- **Application Architecture**

Architectures

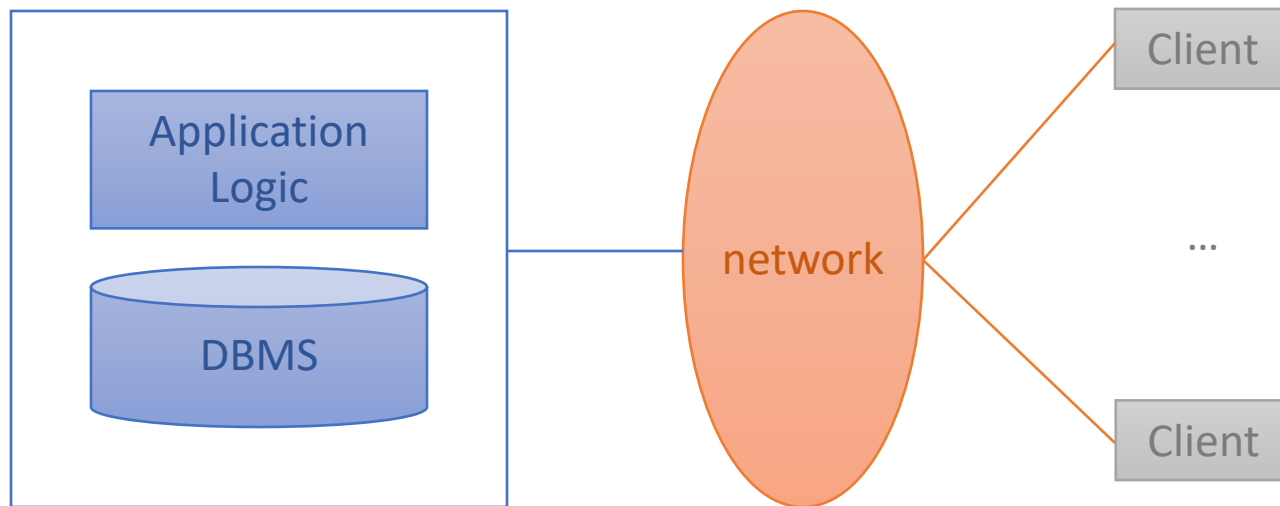
- Single tier
 - How things used to be ...
- Two tier
 - Client-server architecture
- Three tier (and multi-tier)
 - Used for many web systems
 - Very scalable

Single Tier Architecture

- Historically, data intensive applications ran on a single tier which contained
 - The DBMS,
 - Application logic and business rules, and
 - User interface

Two-Tier Architecture

- Client/ server architecture
 - The server implements the business logic and data management
- Separate presentation from the rest of the application

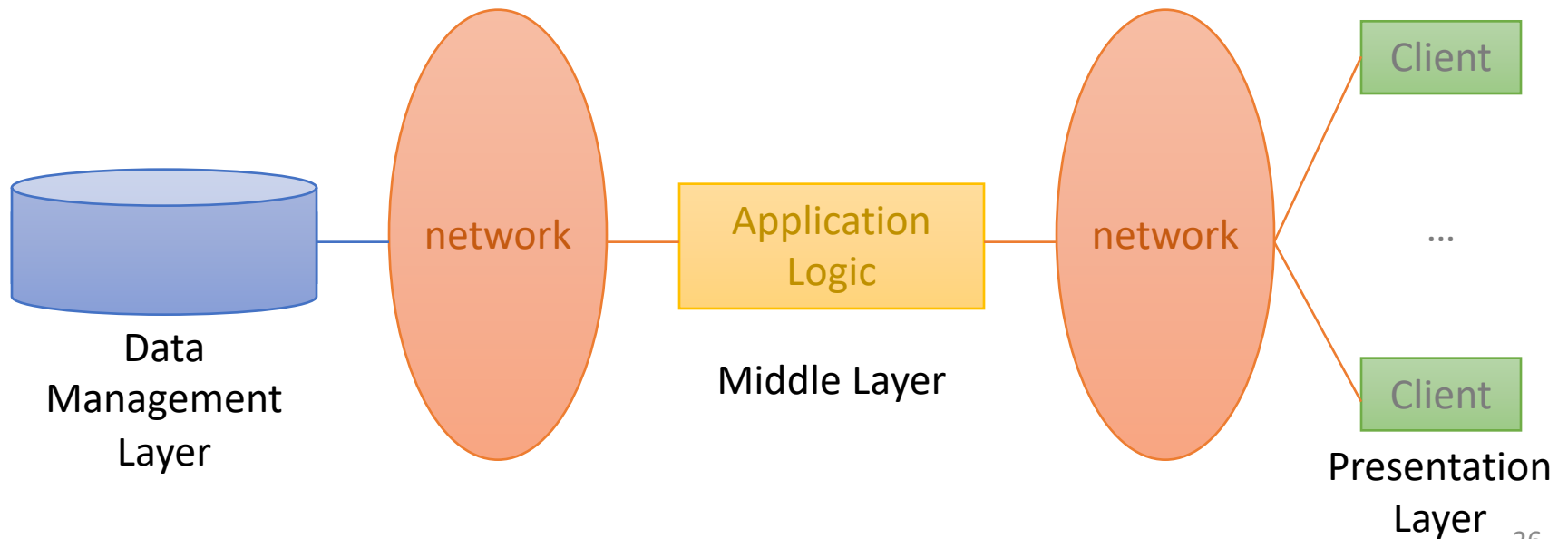


Presentation Layer

- Responsible for handling the user's interaction with the middle tier
- One application may have multiple versions that correspond to different interfaces
 - Web browsers, mobile phones, ...
 - Style sheets can assist in controlling versions

Three-Tier Architecture

- Separate presentation from the rest of the application
- Separate the application logic from the data management



Business logic Layer

- The middle layer is responsible for running the business logic of the application which controls
 - What data is required before an action is performed
 - The control flow of multi-stage actions
 - Access to the database layer
- Multi-stage actions performed by the middle tier may require database access
 - But will not usually make permanent changes until the end of the process
 - e.g. adding items to a shopping basket in an Internet shopping site

Data Management Layer

- The data management tier contains one, or more databases
 - Which may be running on different DBMSs
- Data needs to be exchanged between the middle tier and the database servers
 - This task is not required if a single data source is used but,
 - May be required if multiple data sources are to be integrated
 - *XML* is a language which can be used as a data exchange format between database servers and the middle tier

Example: Airline reservations

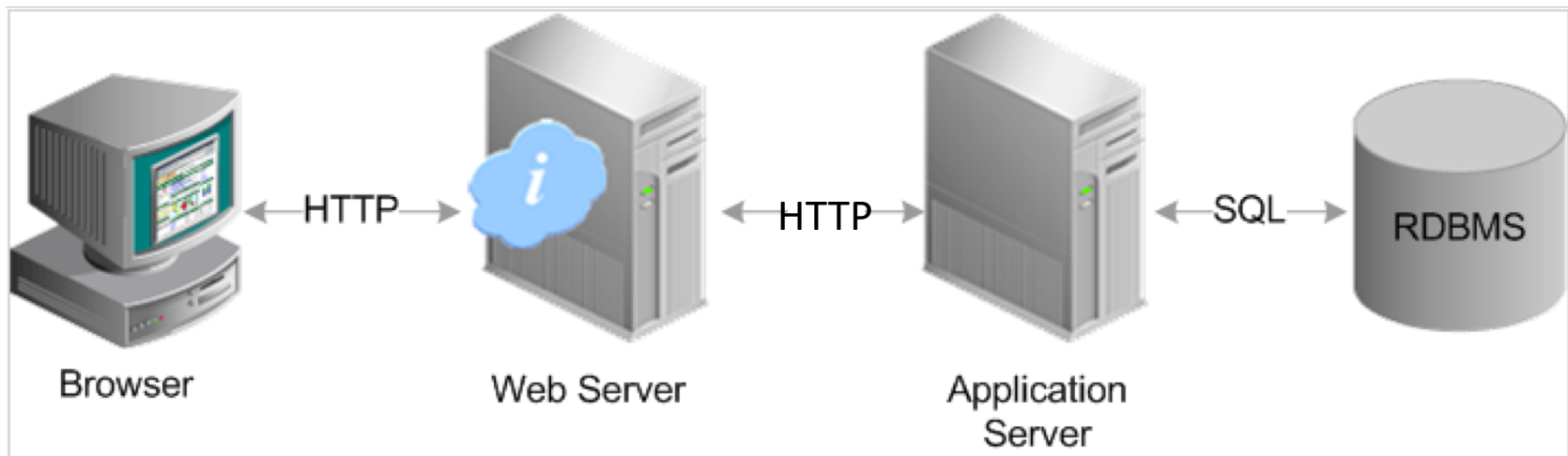
- Consider the three tiers in a system for airline reservations
- Database System
 - Airline info, available seats, customer info, etc.
- Application Server
 - Logic to make reservations, cancel reservations, add new airlines, etc.
- Client Program
 - Log in different users, display forms and human-readable output

Example: Course Enrollment

- Student enrollment system tiers
- Database System
 - Student information, course information, instructor information, course availability, pre-requisites, etc.
- Application Server
 - Logic to add a course, drop a course, create a new course, etc.
- Client Program
 - Log in different users (students, staff, faculty), display forms and human-readable output

3 Tier Architecture and the Web

- In the domain of web applications three tier architecture usually refers to
 - Web server
 - Application server
 - Database server



Summary

- Database Programming
 - Embedded SQL
 - DB API
- Application Architecture
 - Three Tier Architecture

Acknowledge

- Some lecture slides were copied from or inspired by the following course materials
 - “W4111: Introduction to databases” by Eugene Wu at Columbia University
 - “CSE344: Introduction to Data Management” by Dan Suciu at University of Washington
 - “CMPT354: Database System I” by John Edgar at Simon Fraser University
 - “CS186: Introduction to Database Systems” by Joe Hellerstein at UC Berkeley
 - “CS145: Introduction to Databases” by Peter Bailis at Stanford
 - “CS 348: Introduction to Database Management” by Grant Weddell at University of Waterloo