Advanced Programming 2 - Architectural Patterns (2)

DR. ELIAHU KHALASTCHI

2016

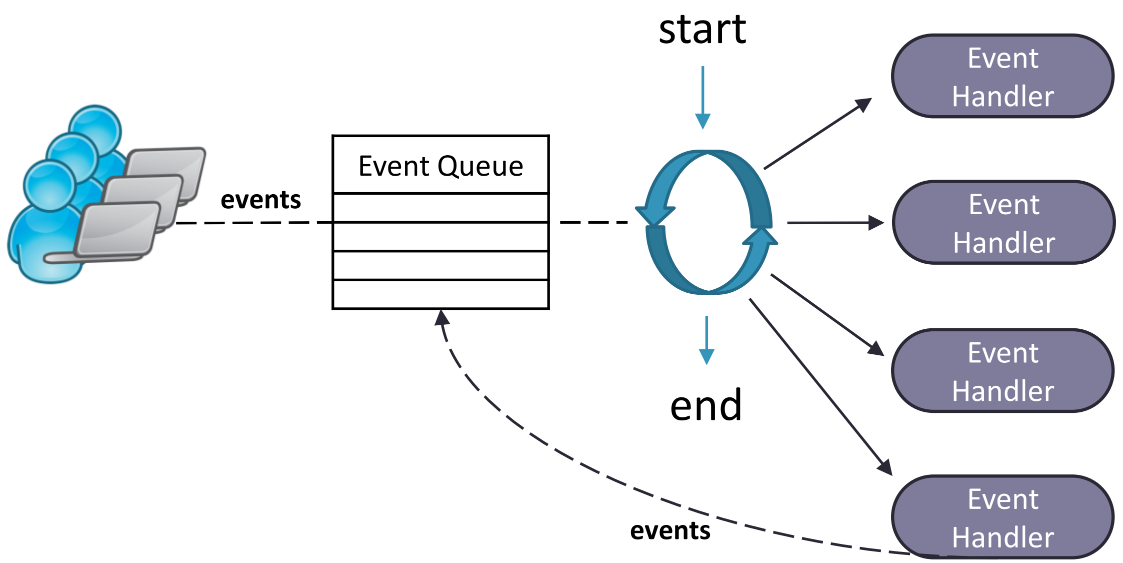
In the last lessons…

MVC, MVC + OBSERVER, MVVM

Event Driven - Architectural Pattern

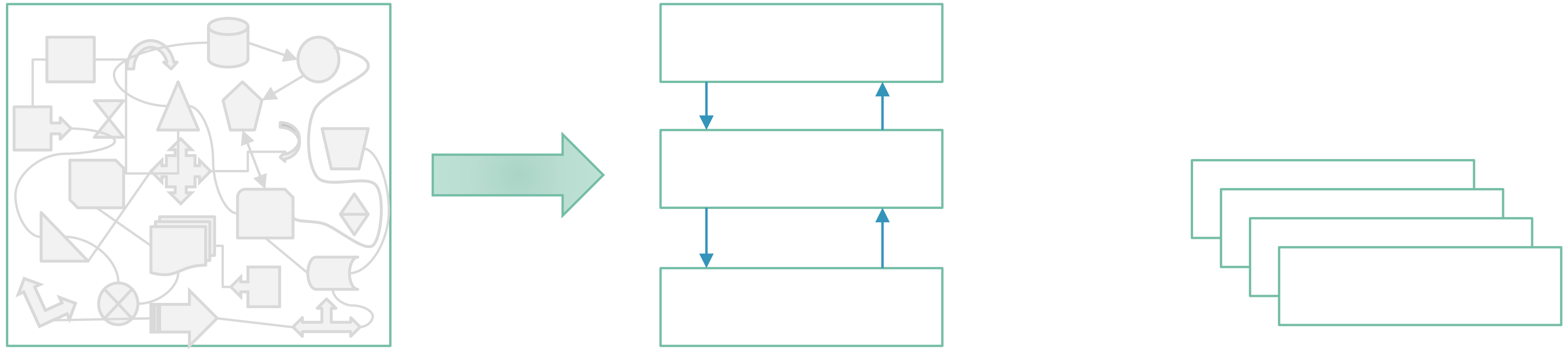
- The program continuously **listens** to defined **events** that **may occur** at **any given time**
- Upon the occurrence of an event, the program "fires" the appropriate **event handler**
  - This is the desired reaction for the event defined by the programmer
  - The event handler code **may trigger new events** as well

- Event driven programming includes:
  - The defined **events**
  - The **event queue** of created runtime events
  - The **event handlers** for the defined events
  - The **main event loop** that extracts events from the queue and triggers the event-handler's code

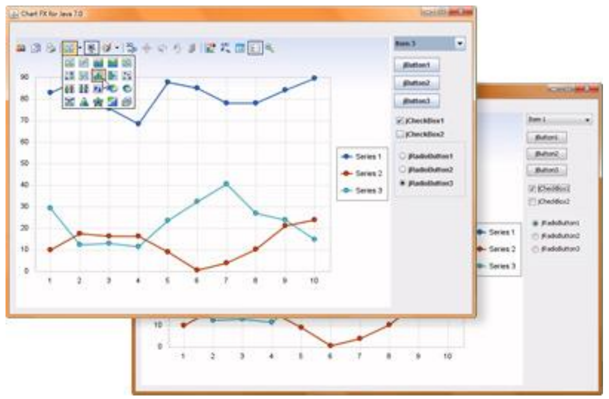


Dividing the code into layers

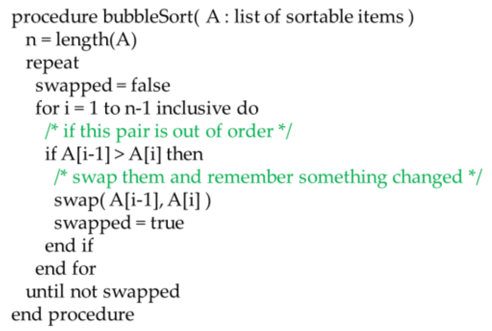
- We do not want to implement everything in 1 layer of code…
  - When something changes, everything has to be changed
- Instead, we want to divide the code into different layers
  - The code is modular
  - Different teams can work independently parallel to each other
  - Easier to trace and isolate bugs!



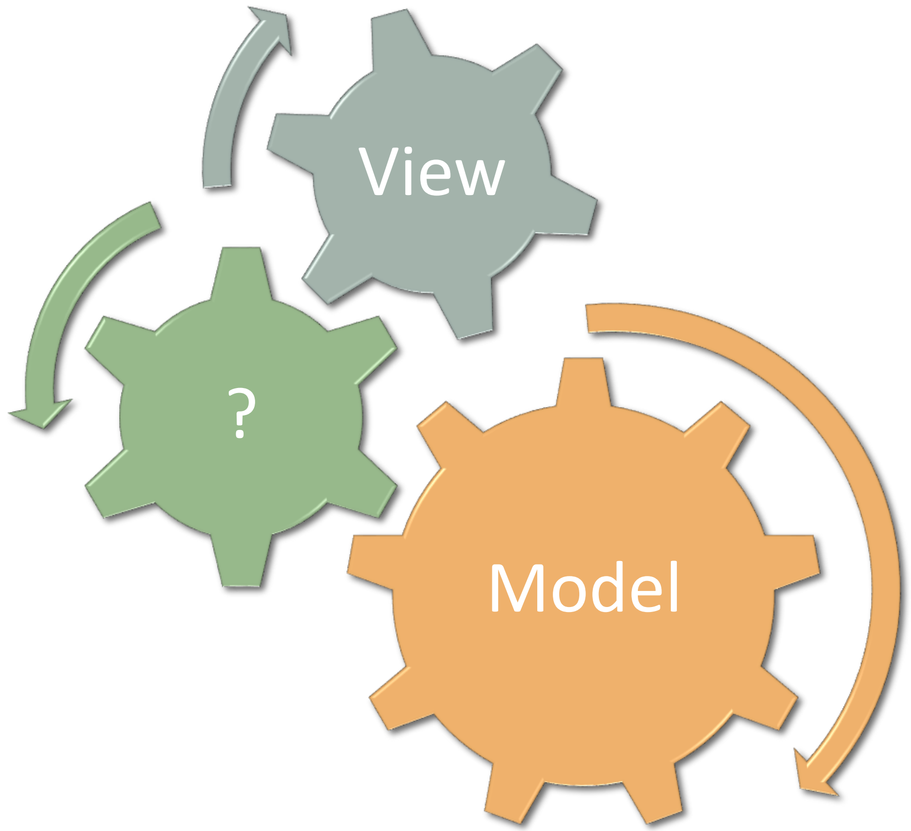
Separation of the **Model** and the **View**



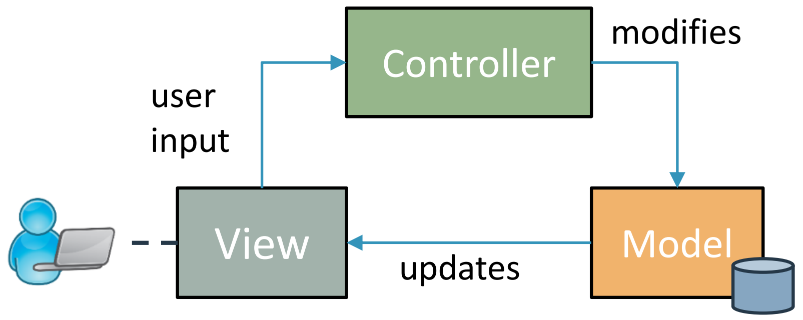
These layers should not “know” each other!

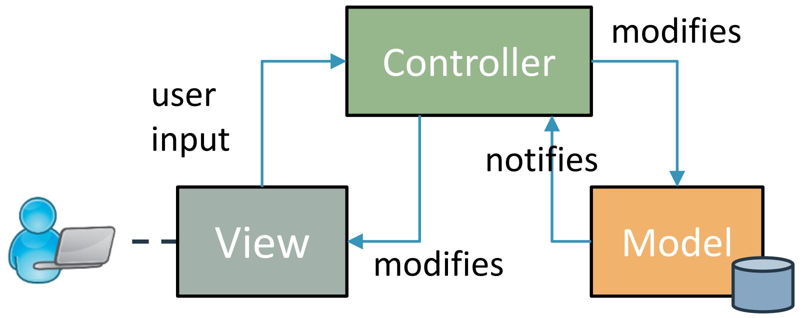


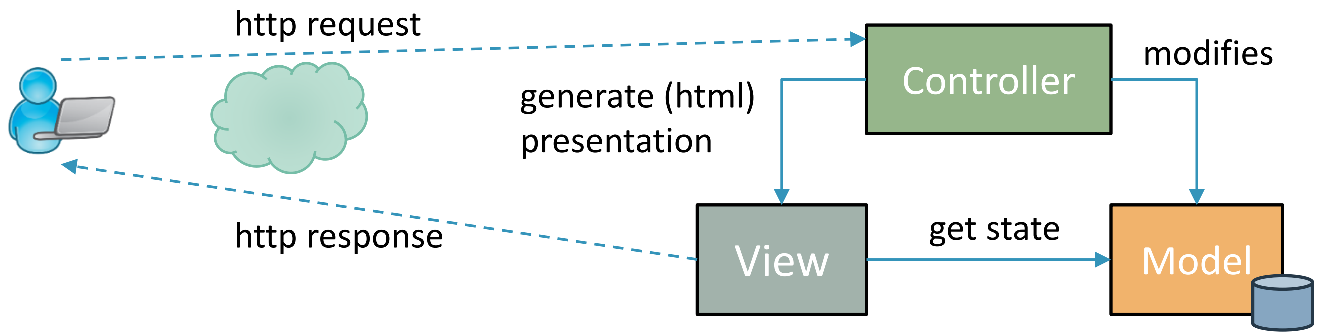
```
procedure bubbleSort( A : list of sortable items )
  n = length(A)
  repeat
    swapped = false
    for i = 1 to n-1 inclusive do
      /* if this pair is out of order */
      if A[i-1] > A[i] then
        /* swap them and remember something changed */
        swap( A[i-1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```



MVC variations

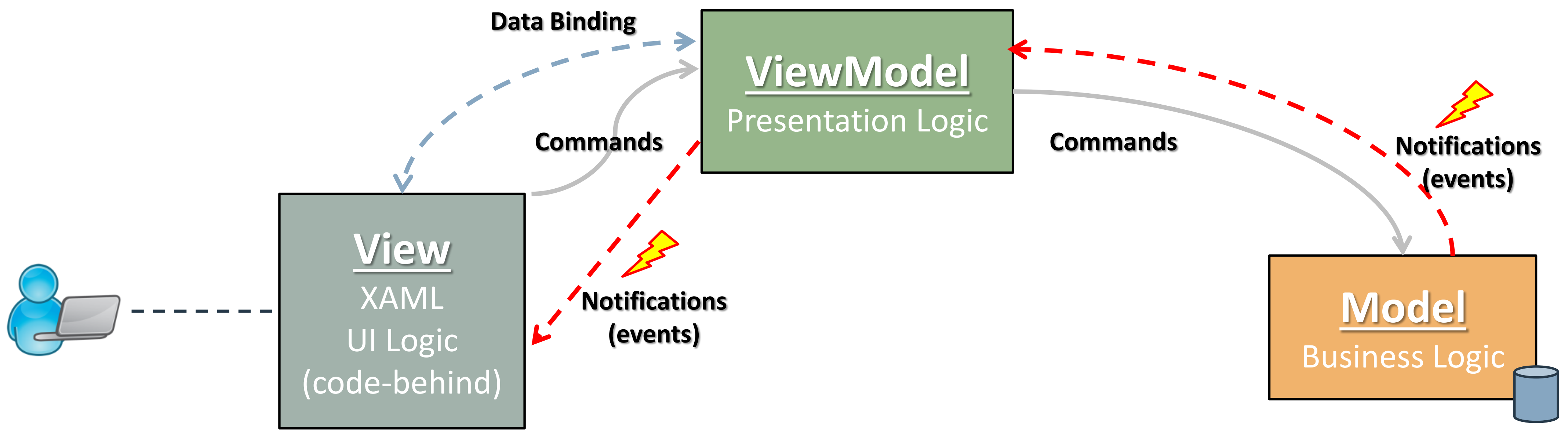






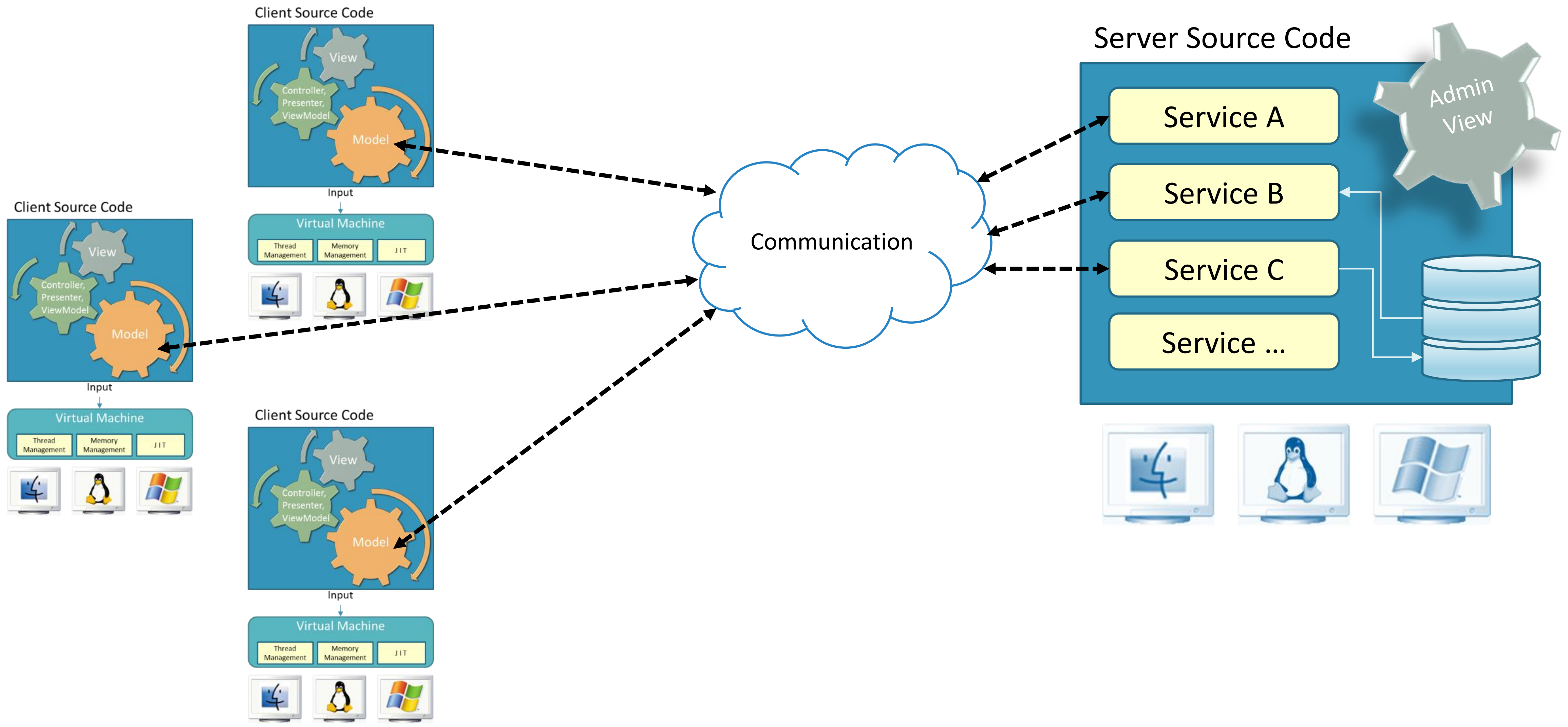
Web application

MVVM – Architectural Pattern



- The **ViewModel** is the “model of the view”
  - For the View, it is an abstraction of the Model
  - It passes commands from the view to the model
- The **ViewModel**
  - Converts model information into view information
  - Something the View can understand…

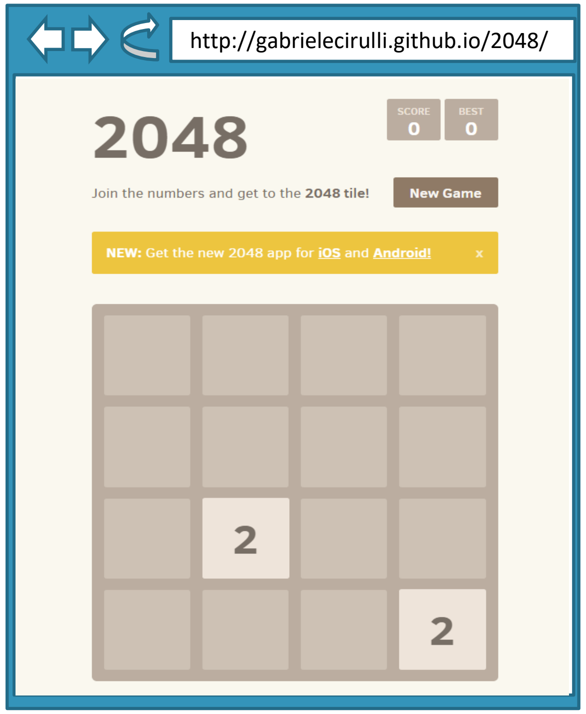
Client-Server Architectural Pattern

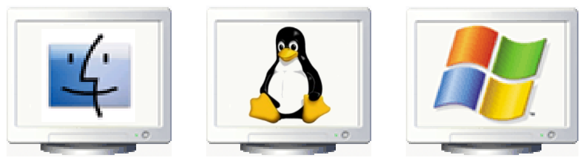


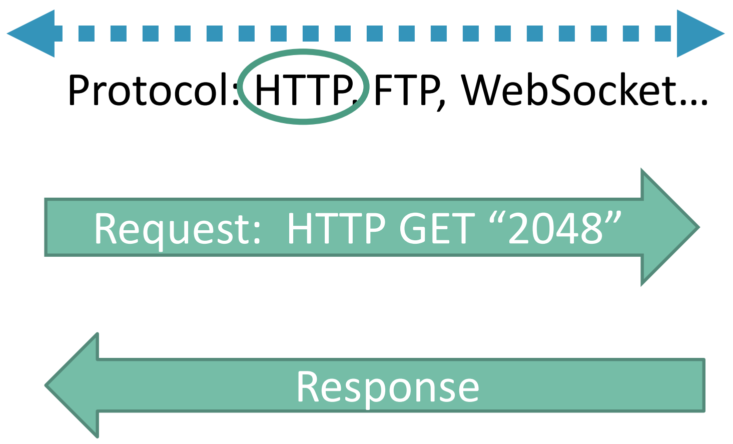
Web-Client-Server Architectural Pattern

Browser
Telnet client
…

Web Client

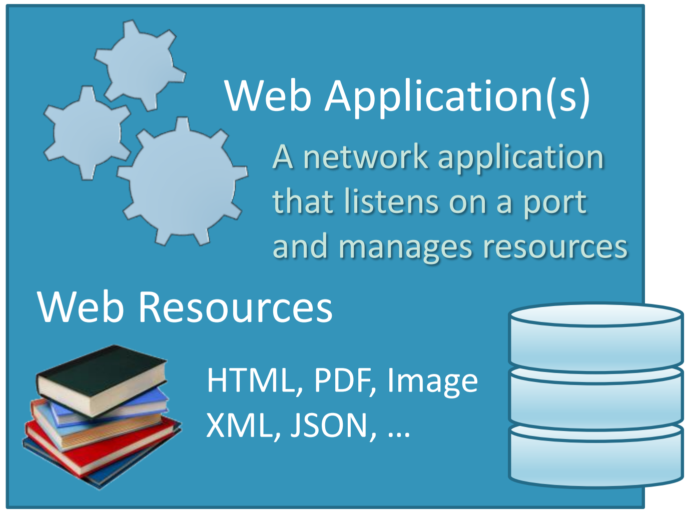

http://gabrielecirulli.github.io/2048/
2048
SCORE 0
BEST 0
Join the numbers and get to the 2048 tile!
New Game
NEW: Get the new 2048 app for iOS and Android!
2
2





```
<html>
…
  <body>
    <script src="js/tile.js"></script>
    …
  </body>
</html>
```

Web Server


Web Application(s)
A network application that listens on a port and manages resources
Web Resources
HTML, PDF, Image
XML, JSON, …

Static resource: does not change (true files)
Dynamic resource: generated on the fly

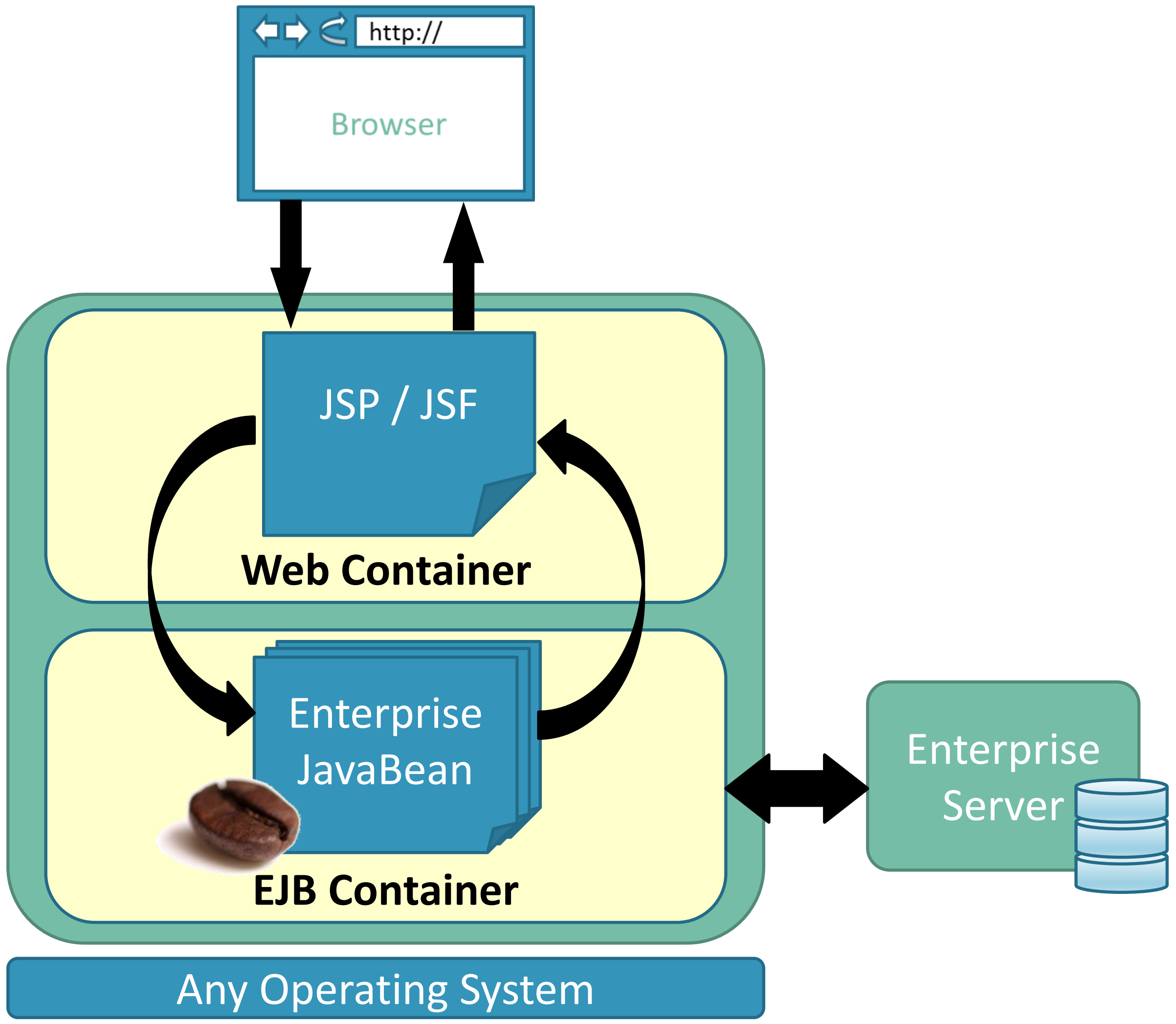
Each resource has a URI:
Uniform Resource Identifier
For example:
http://www.WebServer.com/path/resource.html

J2EE Architecture - model 1

Client Side:
(client tier)

Java2 EE
Application Server
(middle tier)

Enterprise
Information
System (EIS)
tier

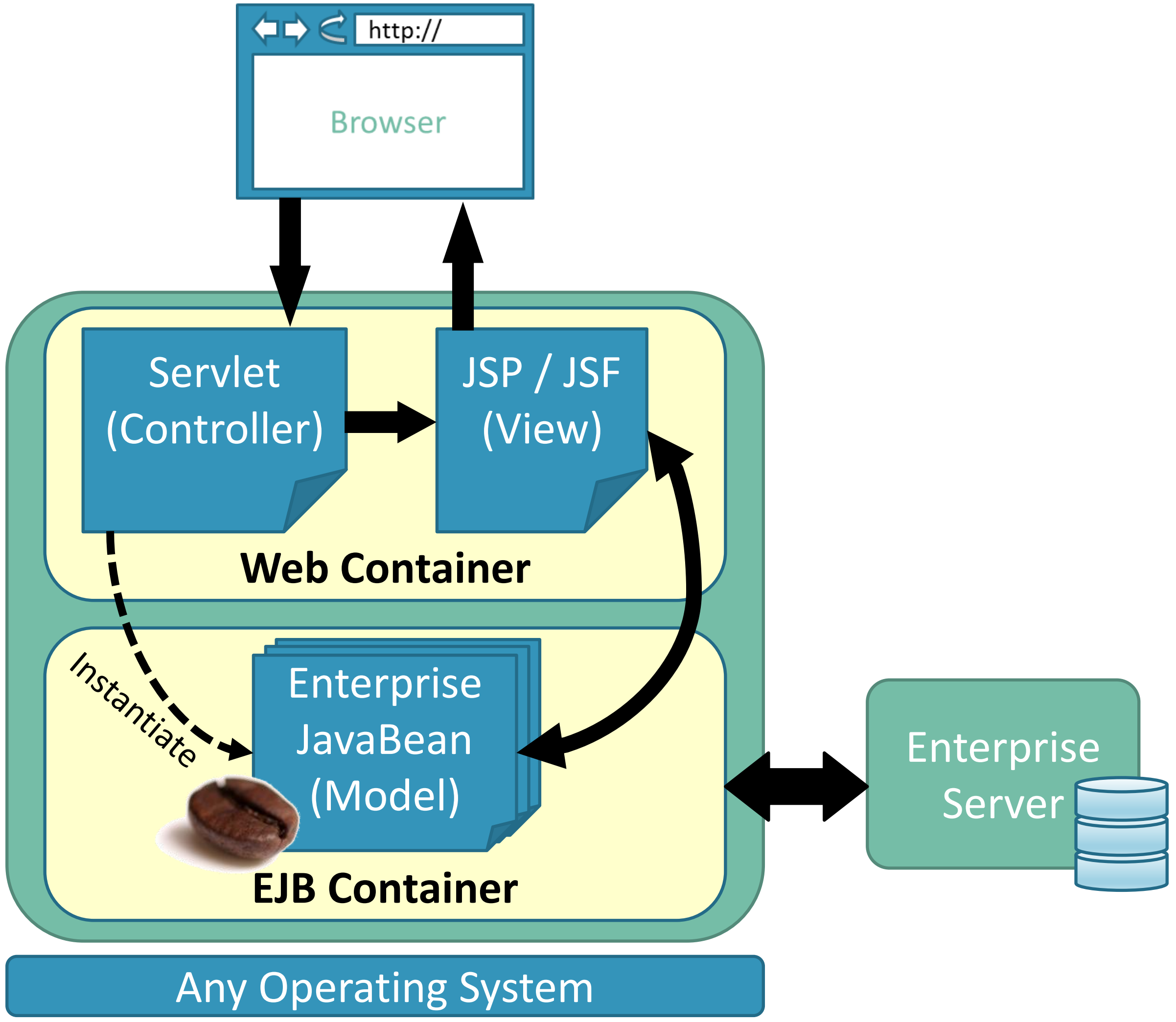


J2EE Architecture - model 2 (better)

Client Side:
(client tier)

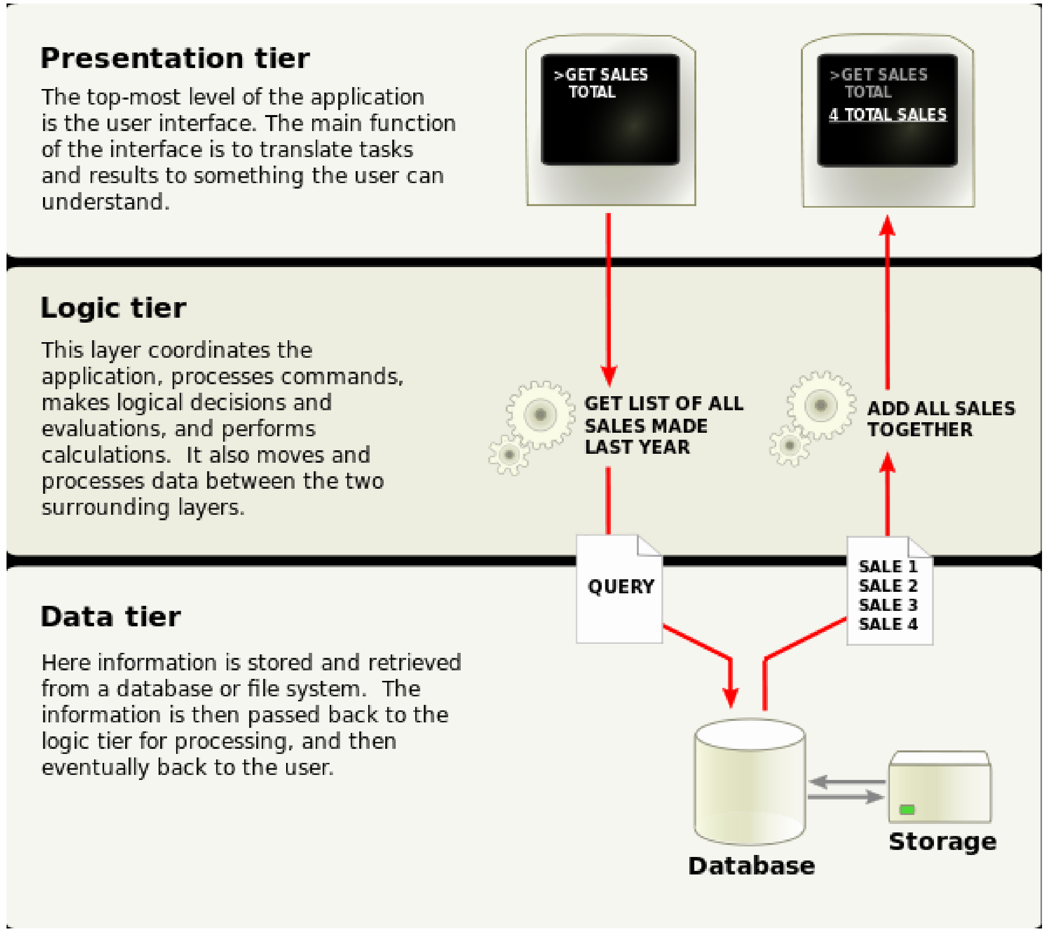
Java2 EE
Application Server
(middle tier)

Enterprise
Information
System (EIS)
tier



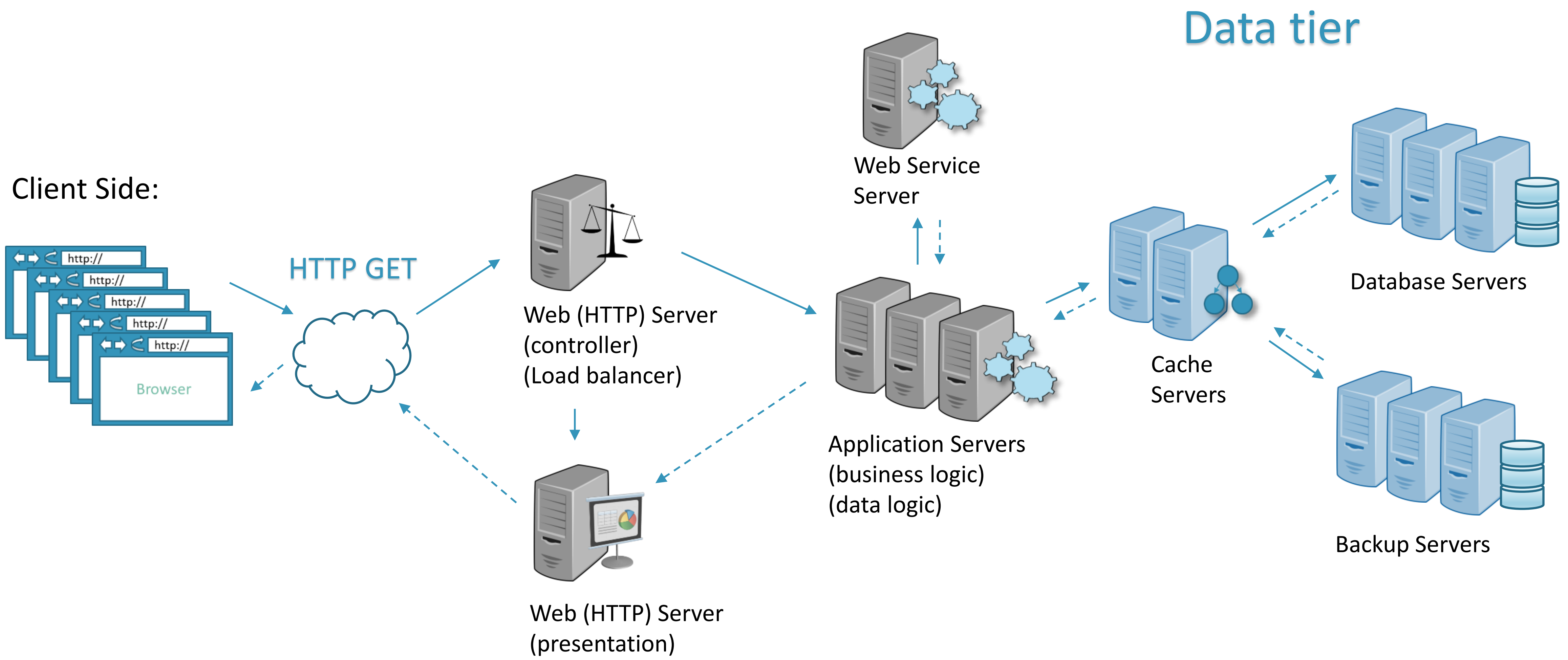
Multi-tier Architectural Pattern

- Commonly, 3-tier architecture
- A client-server architecture
- The main components are physically separated:
  - Presentation
  - application processing
  - data management
- Layer – logical separation
- tier – physical separation
- Any tier can be upgraded or replaced independently

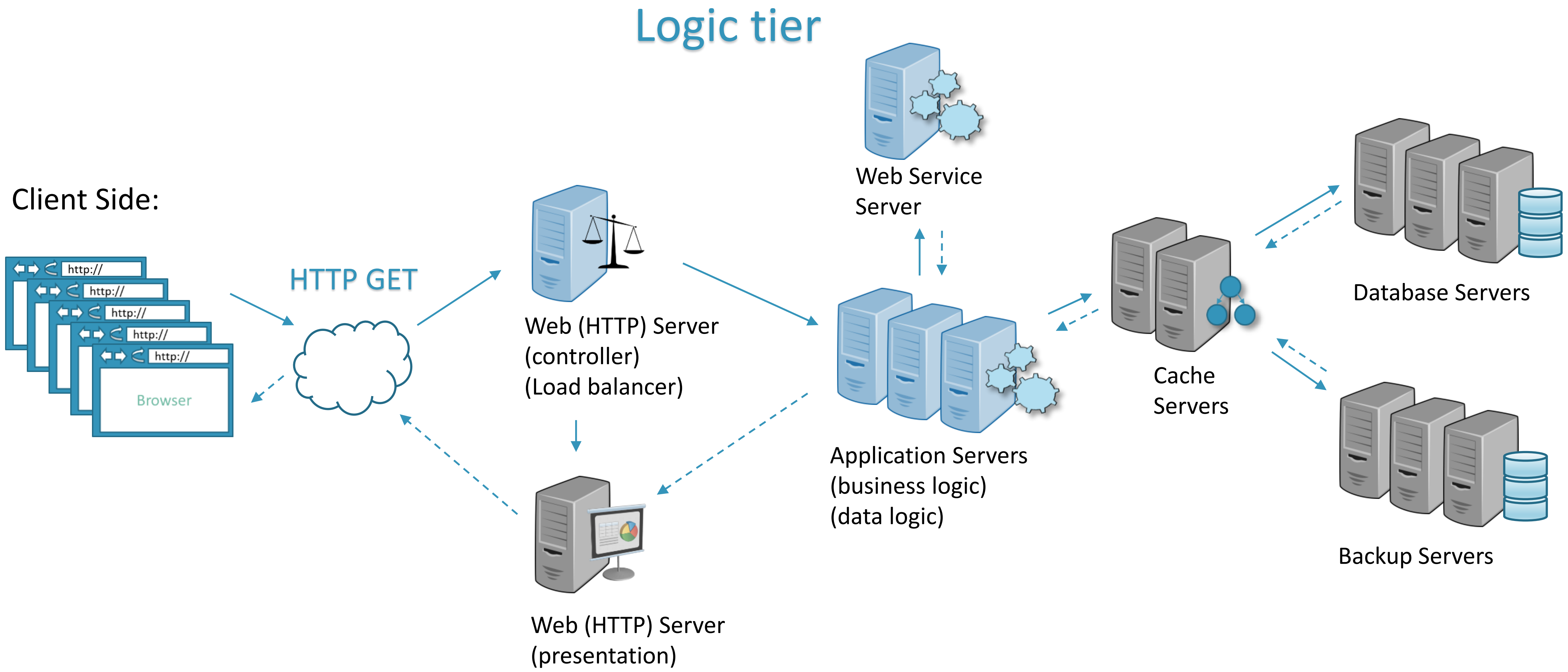


Presentation tier
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.
>GET SALES TOTAL
>GET SALES TOTAL
4 TOTAL SALES
Logic tier
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
GET LIST OF ALL SALES MADE LAST YEAR
ADD ALL SALES TOGETHER
QUERY
SALE 1
SALE 2
SALE 3
SALE 4
Data tier
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.
Database
Storage

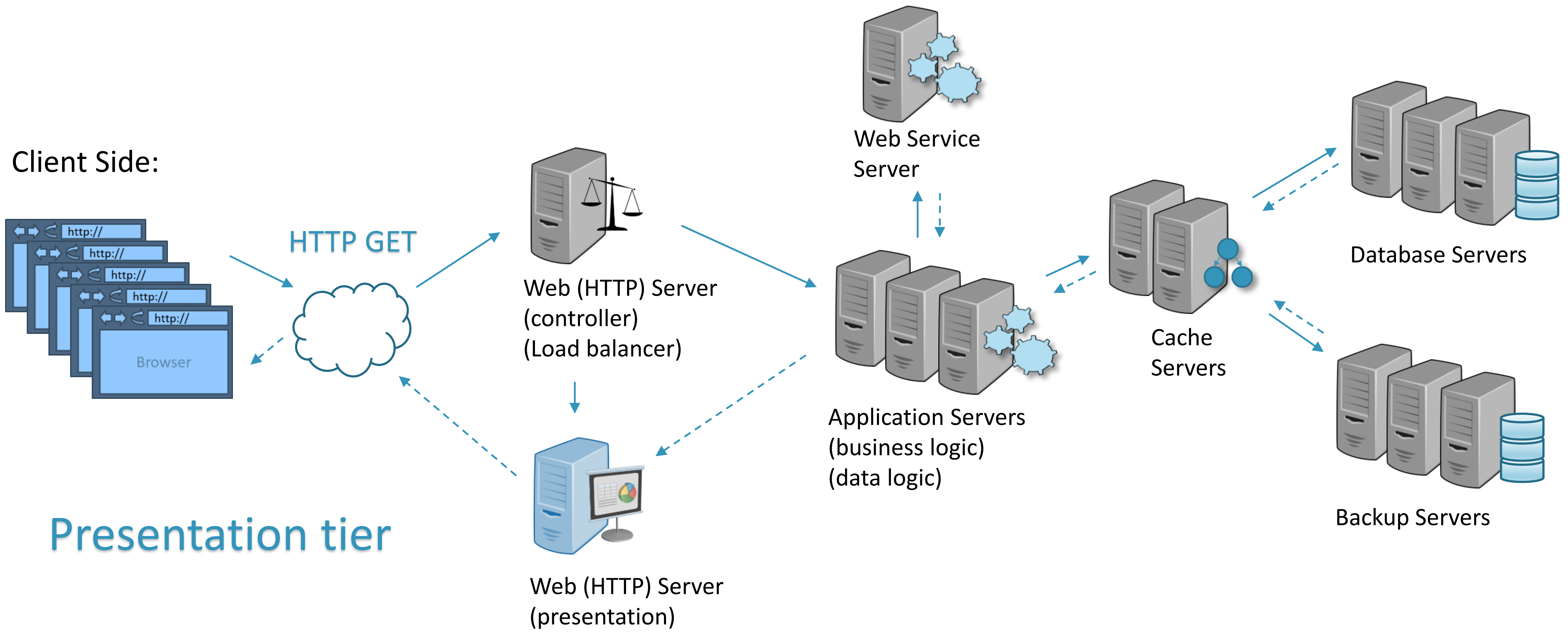
Our Enterprise


Data tier
Client Side:
http://
Browser
HTTP GET
Web Service Server
Web (HTTP) Server (controller) (Load balancer)
Application Servers (business logic) (data logic)
Cache Servers
Database Servers
Backup Servers
Web (HTTP) Server (presentation)

Our Enterprise

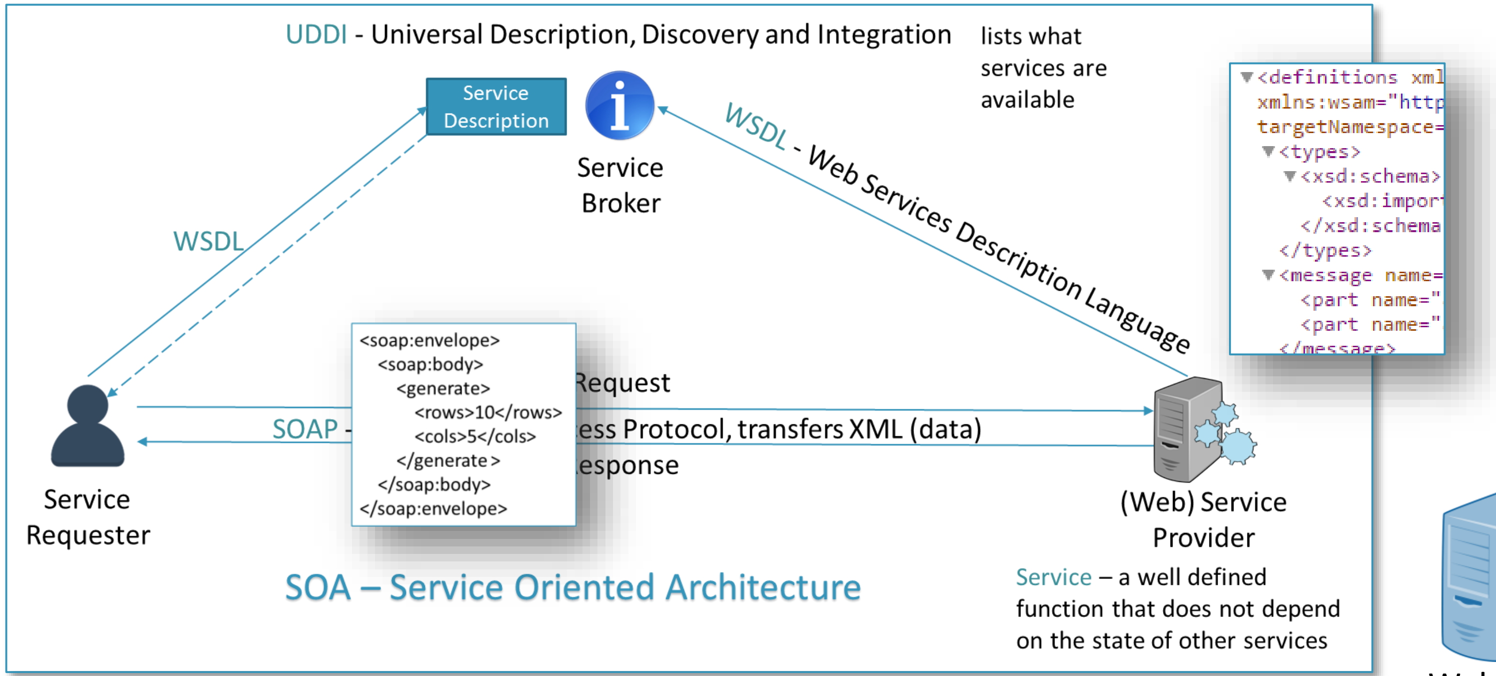


Our Enterprise


Client Side:
http://
Browser
HTTP GET
Web (HTTP) Server (controller) (Load balancer)
Web (HTTP) Server (presentation)
Web Service Server
Application Servers (business logic) (data logic)
Cache Servers
Database Servers
Backup Servers

Presentation tier

Service Oriented Architecture (SOA)

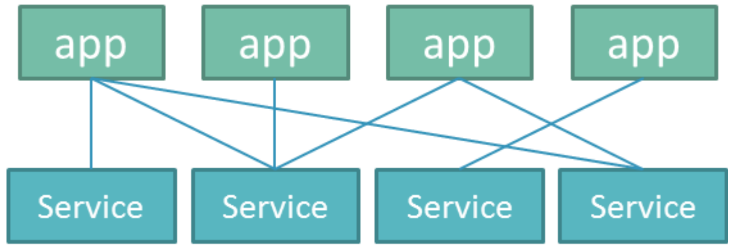

UDDI - Universal Description, Discovery and Integration
Service Description
Service Broker
WSDL
WSDL - Web Services Description Language
SOAP -
Request
ess Protocol, transfers XML (data)
esponse
Service Requester
(Web) Service Provider
SOA – Service Oriented Architecture

lists what services are available

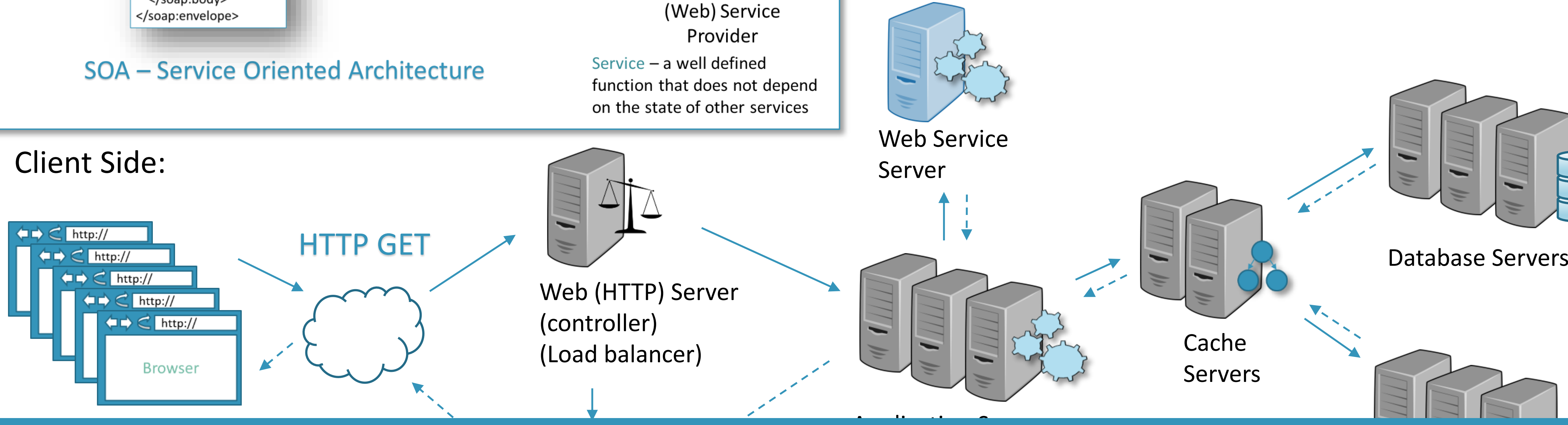
```
<soap:envelope>
  <soap:body>
    <generate>
      <rows>10</rows>
      <cols>5</cols>
    </generate>
  </soap:body>
</soap:envelope>
```

```
▼<definitions xml
 xmlns:wsam="http
 targetNamespace=
 ▼<types>
   ▼<xsd:schema>
       <xsd:impor
     </xsd:schema
   </types>
 ▼<message name=
     <part name="
     <part name="
   </message>
```

Service – a well defined function that does not depend on the state of other services



Client Side:



REST Architecture

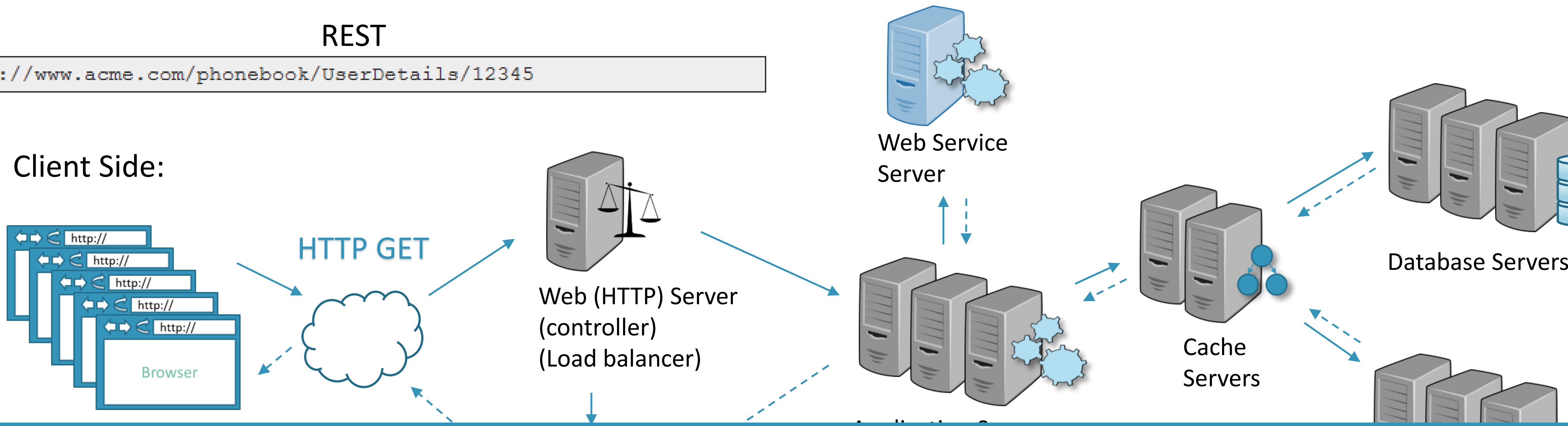
SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

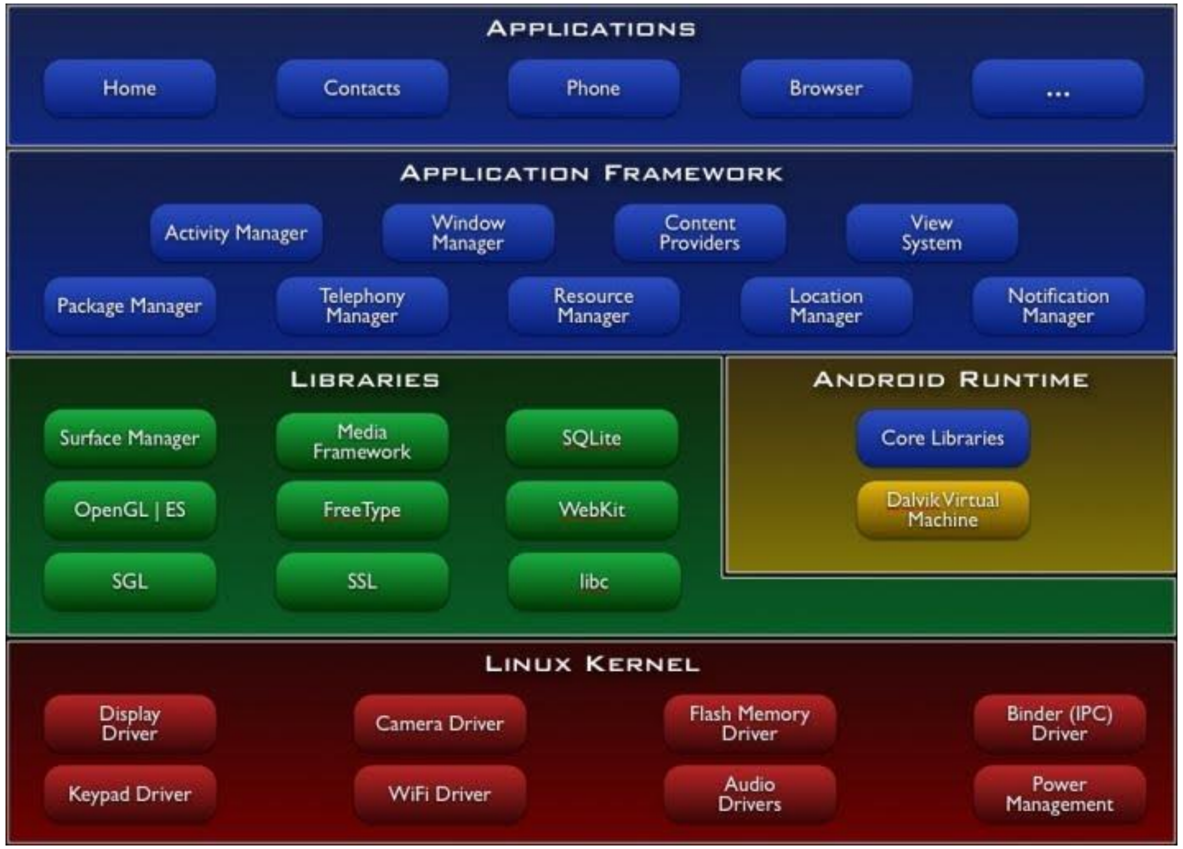
REST

```
http://www.acme.com/phonebook/UserDetails/12345
```

Client Side:



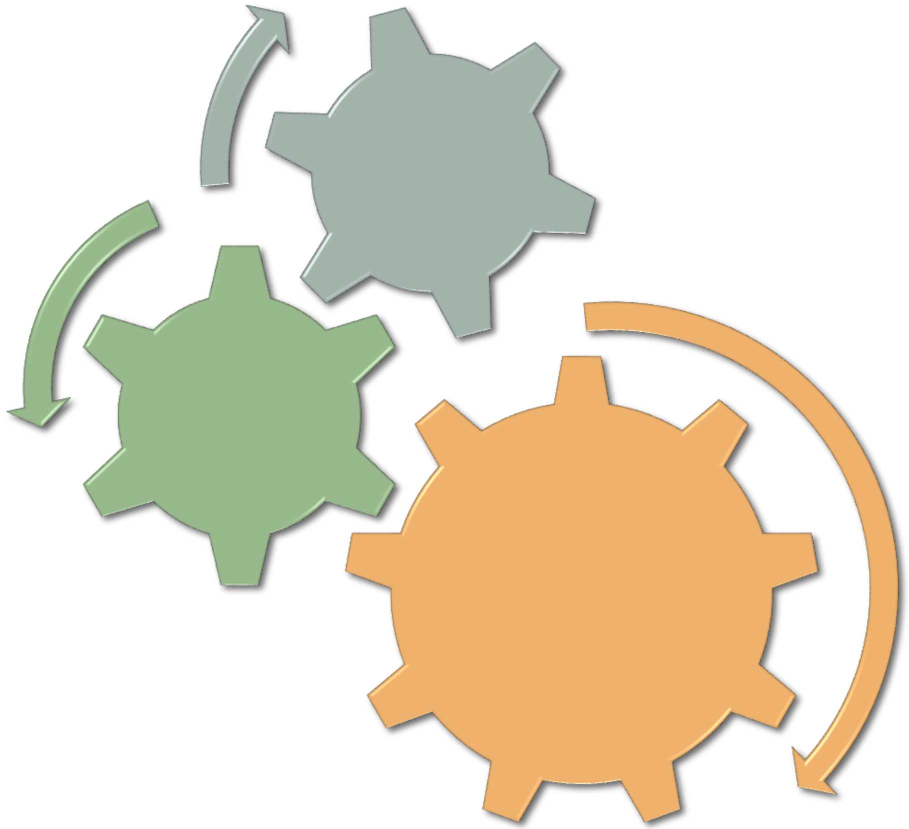
Layers Architecture (android example)



Today’s Lesson

OTHER ARCHITECTURAL PATTERNS!

Controllers

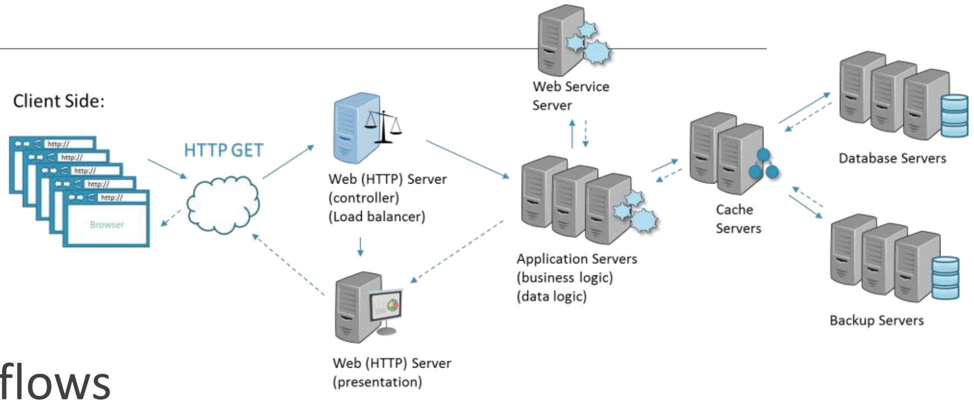


ARCHITECTURAL PATTERNS FOCUSED ON CONTROL

Front Controller

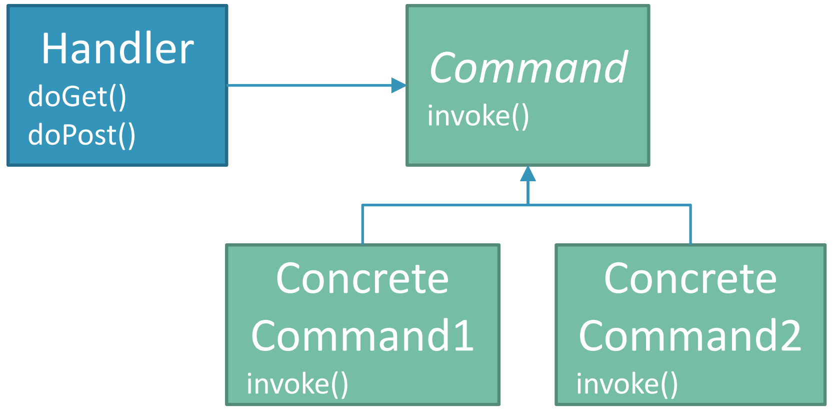
Front Controller

- Provides a **centralized** entry point for handling requests
  - All requests are handled by a single handler



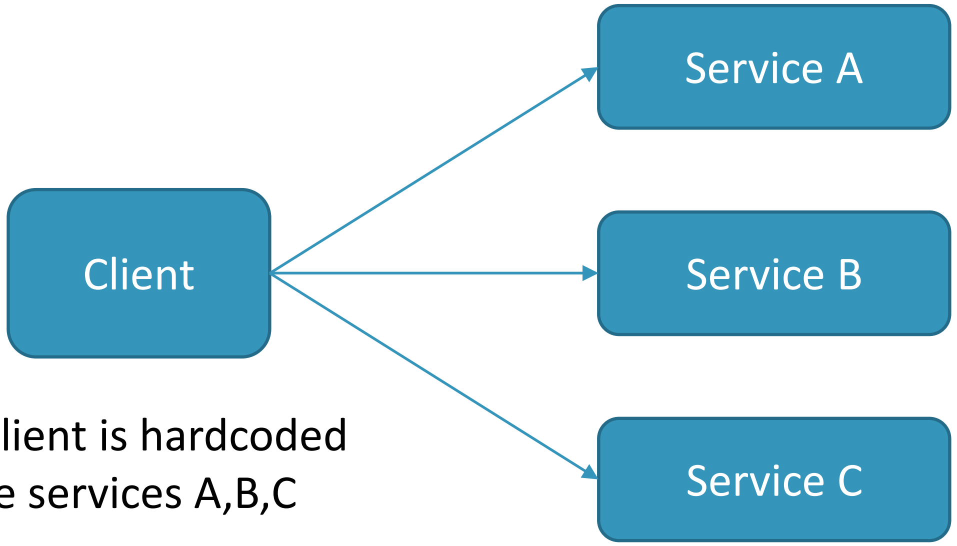
- Often used in web applications to implement common workflows
  - Authentication → Authorization → logging → tracking of requests
  - And then pass them to corresponding handlers
  - A decentralized approach might cause duplicated code

- The common behavior executed by the handler can be modified at runtime



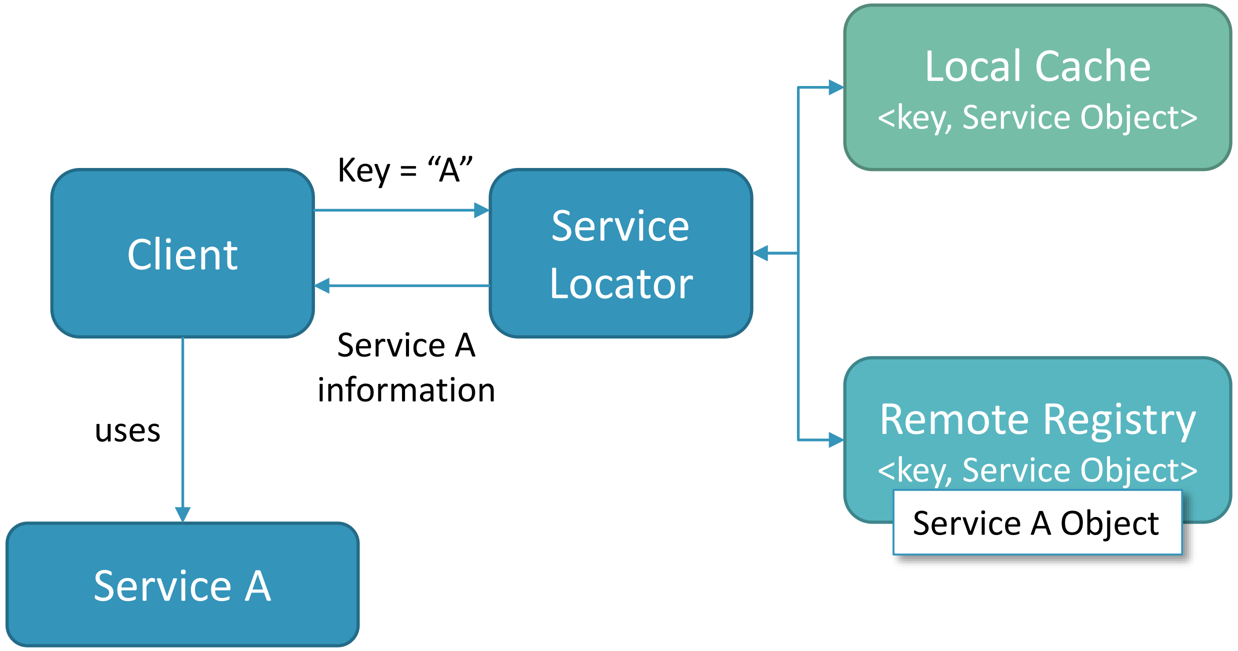
Service Locator

Without Service Locator

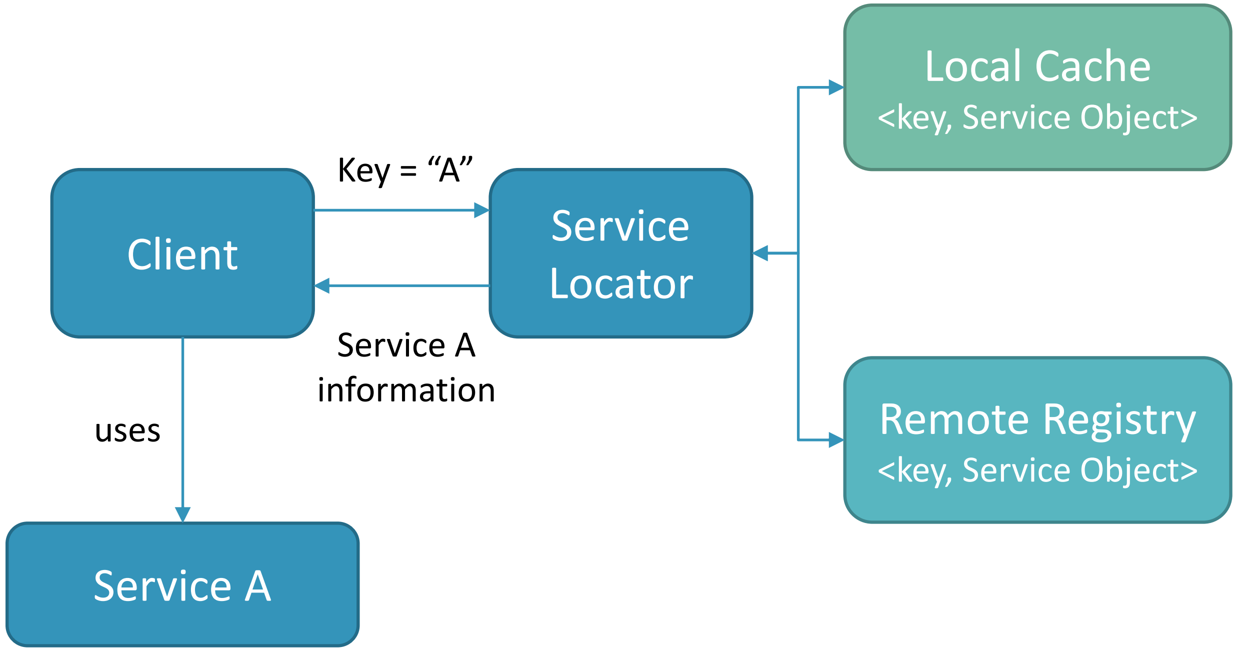


The client is hardcoded
to use services A,B,C

With Service Locator



With Service Locator



- Advantages
  - Runtime linker – code can be added at runtime
  - Apps can optimize to use the best services
  - Almost complete separation (the only link is the registry)
- Disadvantages
  - Services become black boxes
  - The service locator is a bottleneck for concurrency
  - Security risk – outsiders can inject code
  - Runtime errors instead of compile-time
  - Harder to maintain and test

Multiagent System

MAS

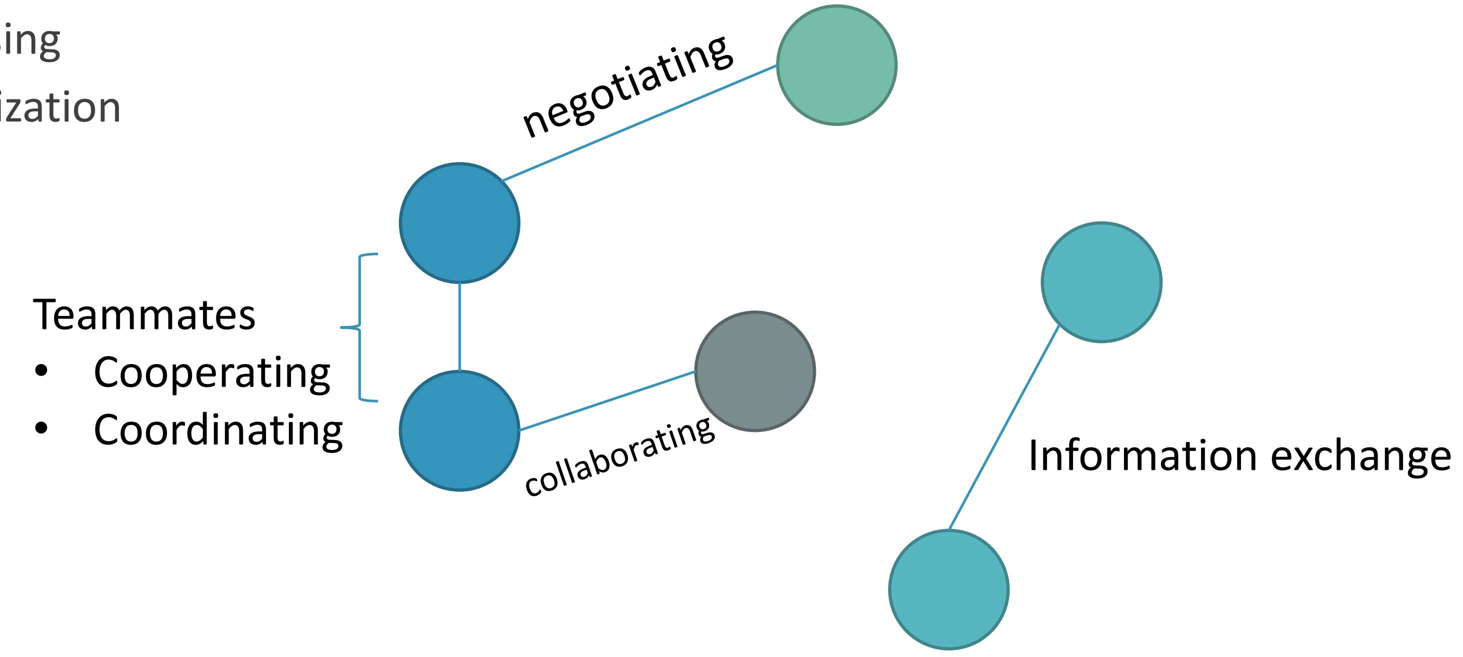
An agent

- A virtual (software agent) or physical (robot) entity that
- **Senses** its environment
  - Processes raw readings into abstract beliefs about the environment
- **Thinks**
  - May Apply learning, reasoning, decision making, global and local planning and derives tasks
- **Acts**
  - Applies actions that affect the agent and the environment
  - Interact with other agents to achieve common or selfish goals

- Examples of software agents
  - Shopping agent, network agent, UI agent, etc.

Multiagent System (MAS)

- A computerized system composed of multiple interacting intelligent agents within an environment
- Can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system
- Characteristics
  - Autonomy
  - Local sensing
  - Decentralization



Presentation Abstraction Control

(PAC)

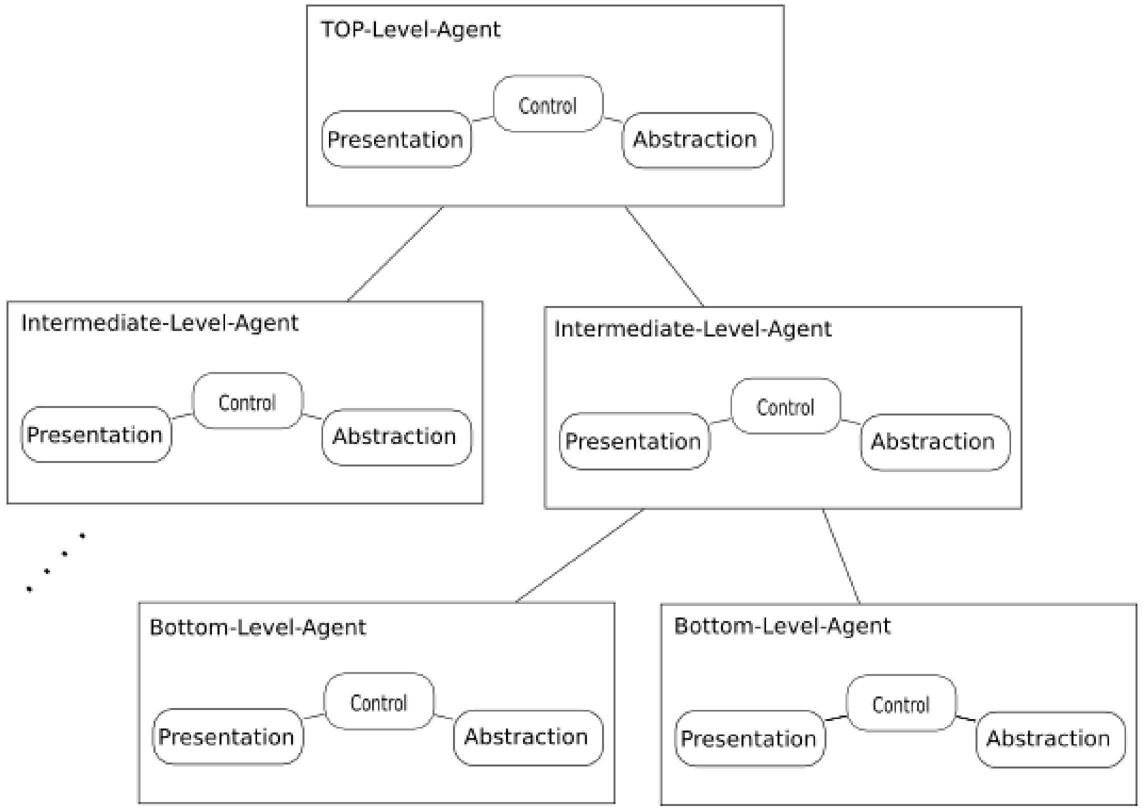
Presentation Abstraction Control (PAC)

- PAC 1987, HMVC (Hierarchal MVC) 2000, essentially the same architectural patterns
- Interaction-oriented software architecture
- Similar to MVC
  - **Abstraction** – retrieves and processes the data
  - **Presentation** – formats the visual (and audio) presentation of the data
  - **Control** – handles flow of control and communications between the two
- Different form MVC
  - A hierarchical structure of agents
  - Each is a PAC
  - Communicate through the control part

Presentation Abstraction Control (PAC)

- Modularity
  - Reduction of dependencies between unrelated parts
- Easier to program
- Changes to one agent does not affect the rest (agents are replaceable / extendable)
- Easier to multithread
  - Each agent can be an active object
- Suits multi-user applications
- Communication depended

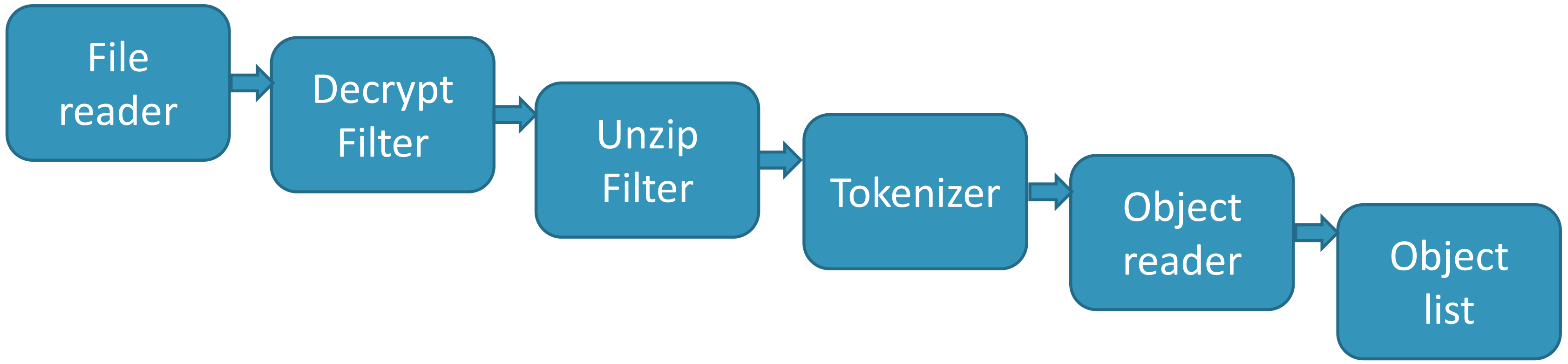
Examples:
- Strategy games
- Air traffic control system
- Web app (each "page" as an agent)



Pipes and Filters

Pipes and Filters

- The concept: a data stream is passed by pipes through filters
- Each filter manipulates the data stream and passes it on to the next filter
- Example:



Pipes and Filters

ADVANTAGES

No need for intermediate files

Filters are replaceable

Filters can be developed independently
- When standard formats are used

Filters can be executed in parallel
- If they are incremental in nature

DISADVANTAGES

Difficult to share global information

Parallelization is less useful here…

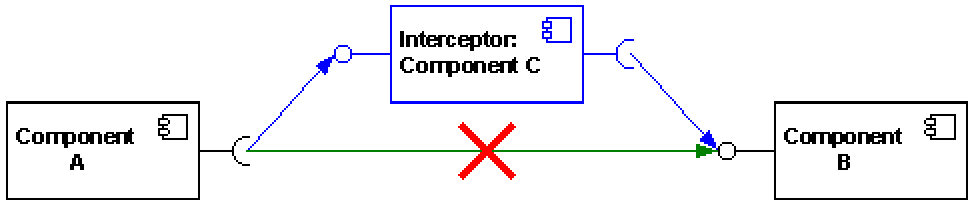
Might be expensive
- When many filters are used

What to do with errors?
- Especially with incremental filters

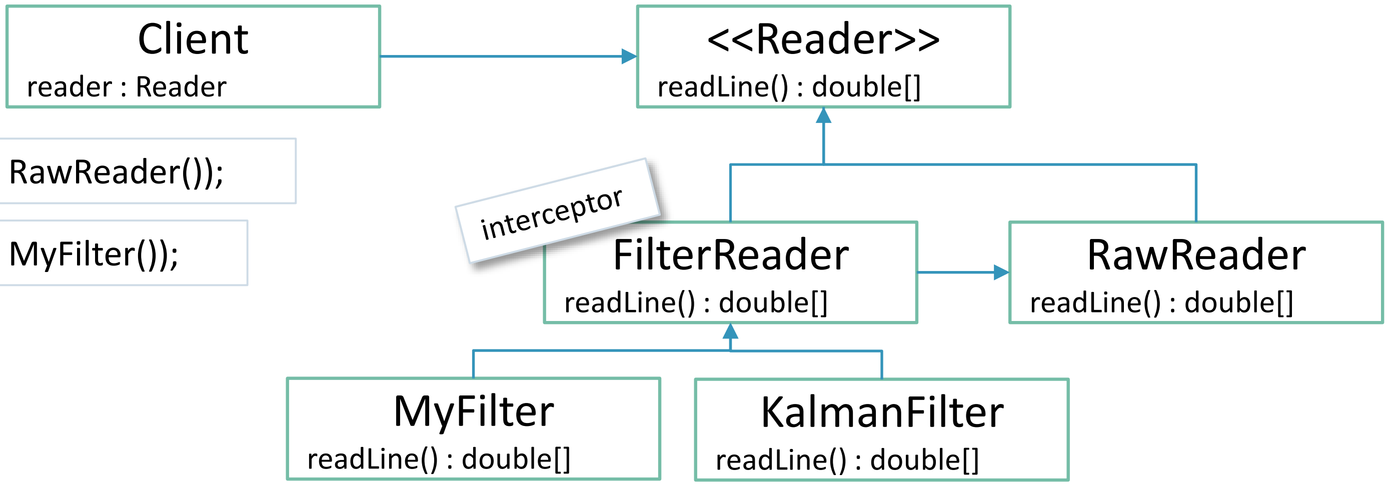
Interceptor pattern

Interceptor Pattern

- Used when software systems want to change their usual processing cycle



- For example:

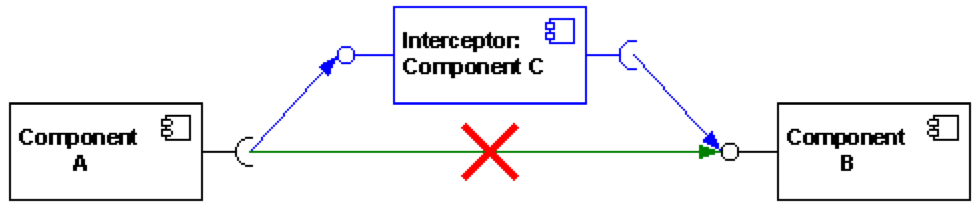


```
client.setReader(new RawReader());
```

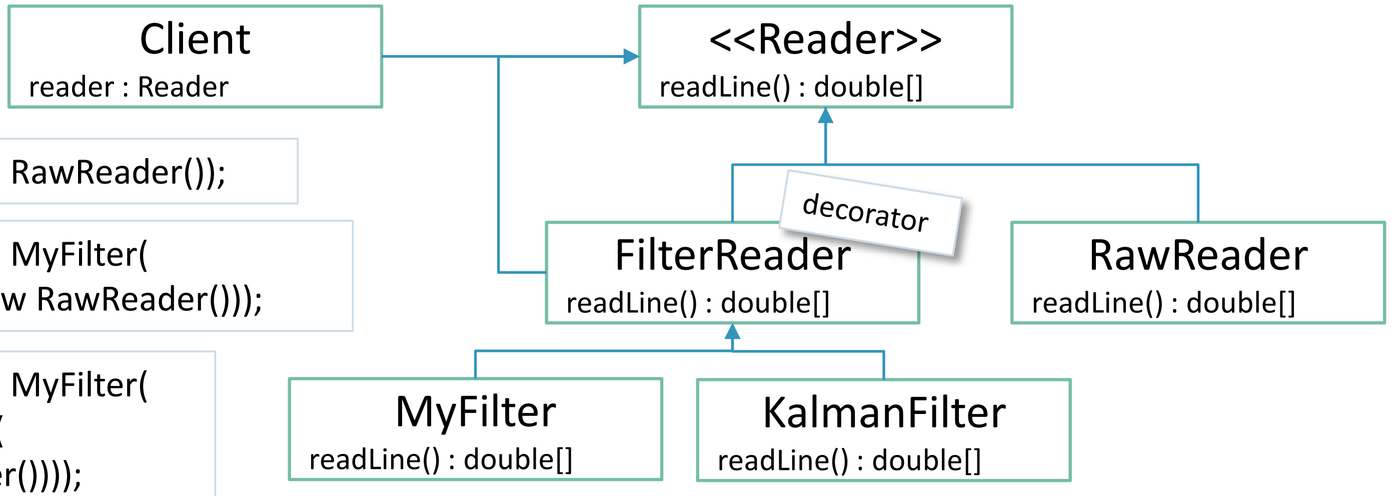
```
client.setReader(new MyFilter());
```

Interceptor Pattern

- Used when software systems want to change their usual processing cycle



- For example:



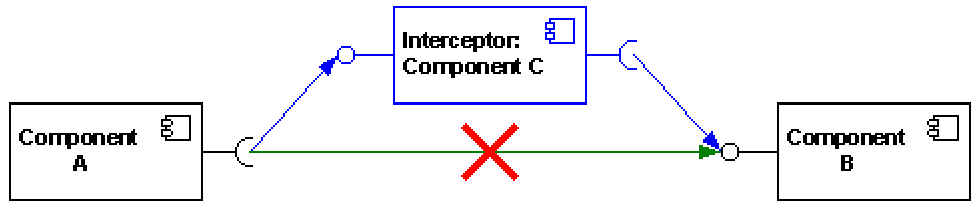
```
client.setReader(new RawReader());
```

```
client.setReader(new MyFilter(
                        new RawReader()));
```

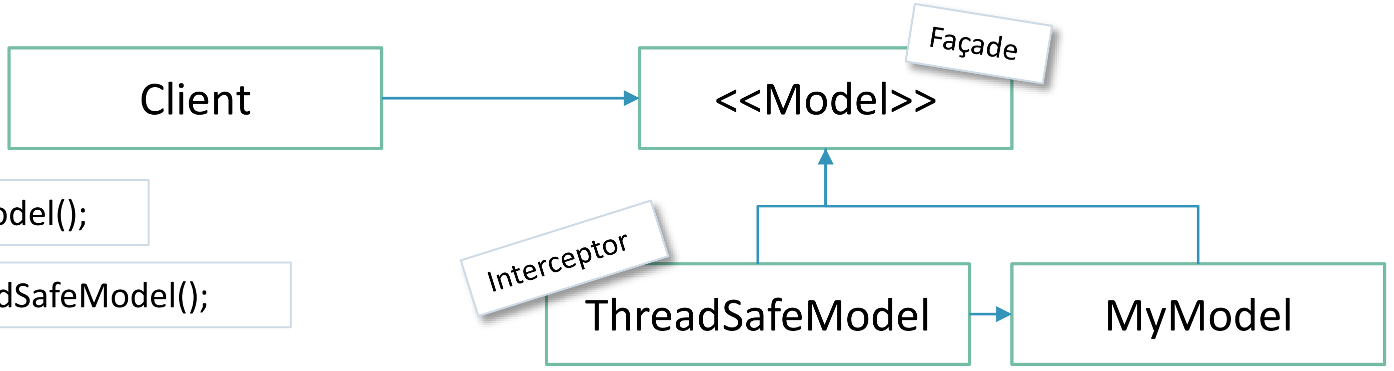
```
client.setReader(new MyFilter(
    new KalmanFilter(
        new RawReader())));
```

Interceptor Pattern

- Used when software systems want to change their usual processing cycle

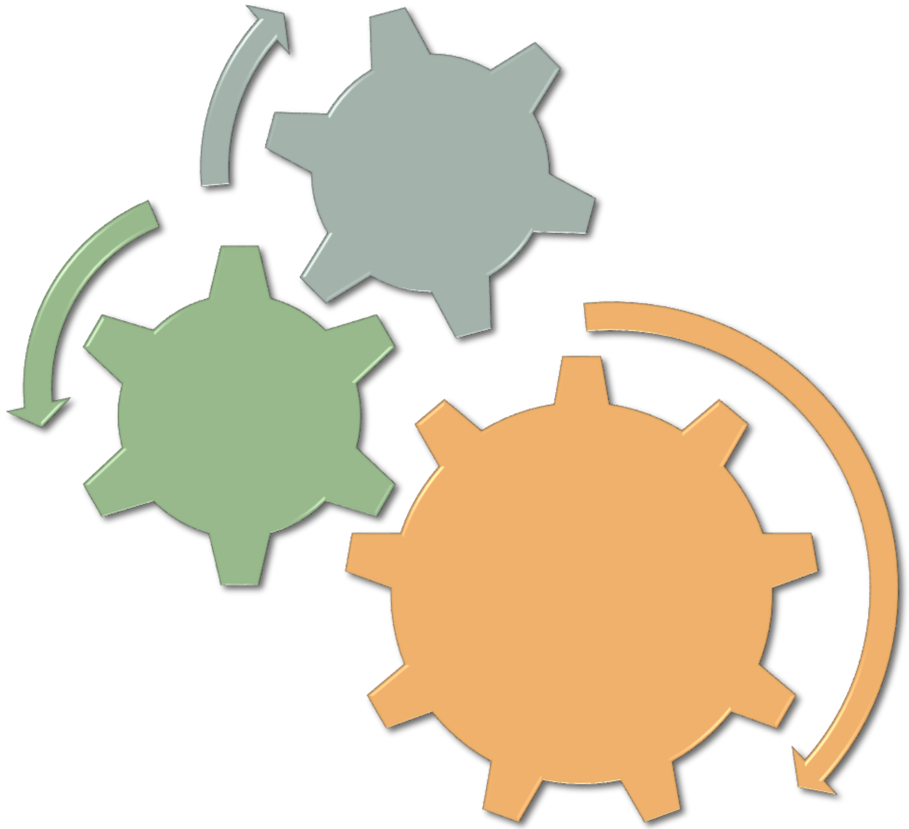


- For example:



```
Model m=new MyModel();
```

```
Model m=new ThreadSafeModel();
```



Viewers

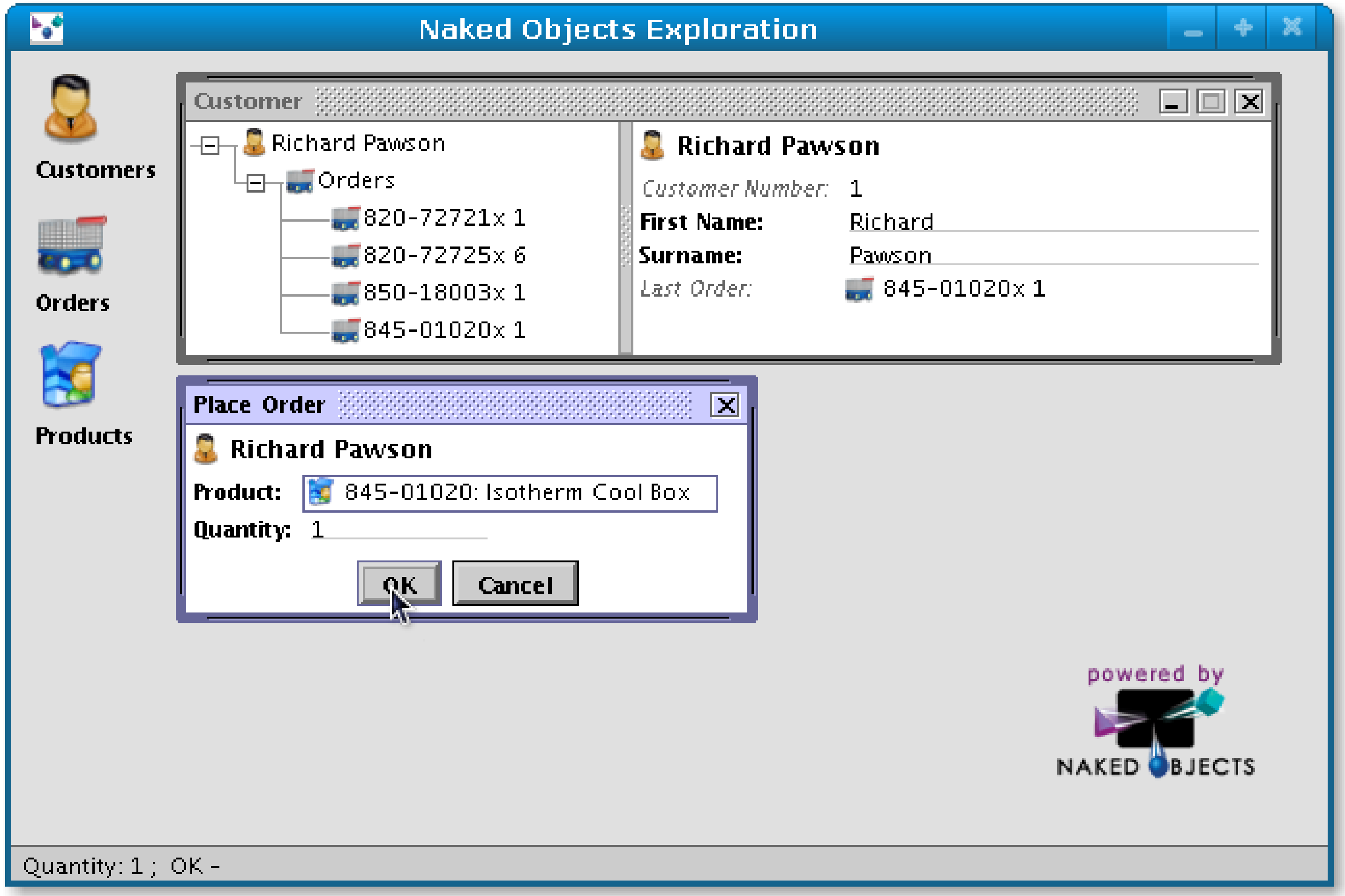
ARCHITECTURAL PATTERNS FOCUSED ON VIEW

Naked Objects

Naked Objects

- MVC
- The (generic) user interface should be 100% automatically created
  - from the definition of the business objects
- Naked objects frameworks to date favor the ***reflection*** technology
- Benefits:
  - A faster deployment cycle
  - Greater agility
  - Easier requirements analysis
  - Easier testing

Naked Objects Example



Richard Pawson
Orders
820-72721x 1
820-72725x 6
850-18003x 1
845-01020x 1

Richard Pawson
Customer Number: 1
First Name: Richard
Surname: Pawson
Last Order: 845-01020x 1

Richard Pawson
Product: 845-01020: Isotherm Cool Box
Quantity: 1

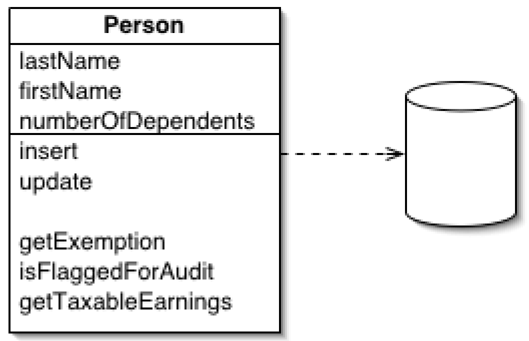
Quantity: 1 ; OK -

Data Access Layers

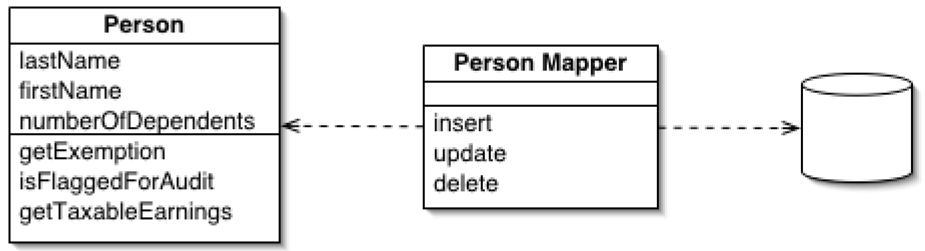
ACTIVE RECORDED VS. DATA MAPPER

Object Relational Mapping (ORM)

- ORM has two main patterns:
- Active Record (e.g., ActiveJDBC)



- Data Mapper (e.g., Hibernate)



http://martinfowler.com/

Active Record

- In memory objects are stored in a data-base
- A class relates to a table
- Each object is mapped to a row in the table

```
part = new Part()
part.name = "Sample part"
part.price = 123.45
part.save()
```

```sql
INSERT INTO parts (name, price) VALUES ('Sample part', 123.45);
```

```
b = Part.find_first("name", "gearbox")
```

```sql
SELECT * FROM parts WHERE name = 'gearbox' LIMIT 1;
```

- Commonly used by *object-relational mapping* (ORM)

Data Mapper (Hibernate example)

```
<?xml version="1.0"?>
  <!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

  <hibernate-mapping>
      <class name="User" table="USERS" >
          <id name="userId" type="java.lang.Long" column="user_id" >
                <generator class="increment" />
          </id>
          <property name="firstName" type="java.lang.String" column="first_name" length="20" />
          <property name="lastName" type="java.lang.String" column="last_name" length="20" />
          <property name="age" type="java.lang.Integer" column="age" length="-1" />
          <property name="email" type="java.lang.String" column="email" length="40" />
      </class>
  </hibernate-mapping>
```

Data Mapper (Hibernate example)

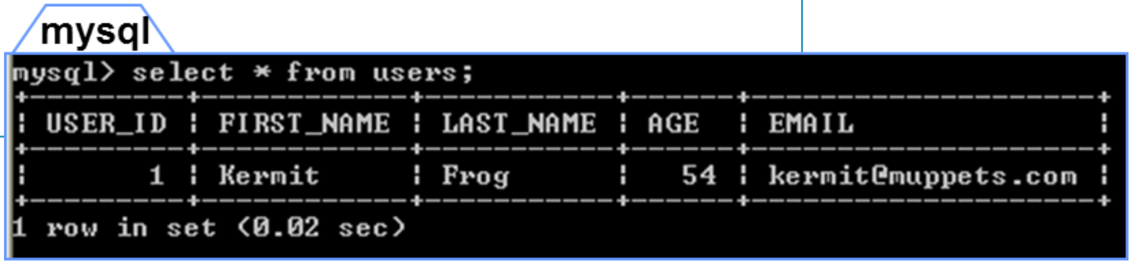
```java
import org.hibernate.Session;

public class UserManager {
 private Session session = null;
 public UserManager(Session session) {
    this.session = session;
 }
 public void saveUser(User user){
  session.save(user);
 }
 public void updateUser(User user){
  session.update(user);
 }
 public void deleteUser(User user) {
  session.delete(user);
 }
}
```

```java
public static void main(String[] args) {
 User user = new User();
 user.setFirstName("Kermit");
 user.setLastName("Frog");
 user.setAge(54);
 user.setEmail("kermit@muppets.com");

 SessionFactory sessionFactory = new
 Configuration().configure().buildSessionFactory();
 Session session = sessionFactory.openSession();
 UserManager manager = new UserManager(session);

 manager.saveUser(user);
 session.flush();
}
```



```
mysql> select * from users;
+---------+------------+-----------+-----+--------------------+
| USER_ID | FIRST_NAME | LAST_NAME | AGE | EMAIL              |
+---------+------------+-----------+-----+--------------------+
|       1 | Kermit     | Frog      |  54 | kermit@muppets.com |
+---------+------------+-----------+-----+--------------------+
1 row in set (0.02 sec)
```

ORM

ACTIVE RECORD

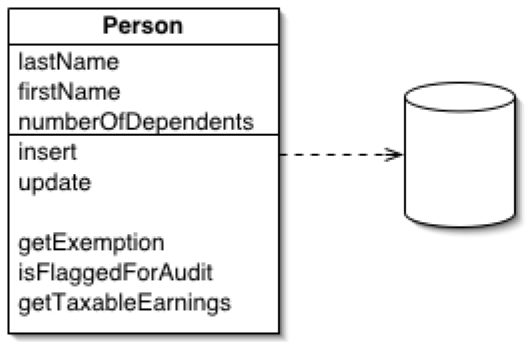
Suits CRUD based applications

- Create, Read, Update, Delete

When the code maps cleanly to a database

No single responsibility principle

Hard to test without a database

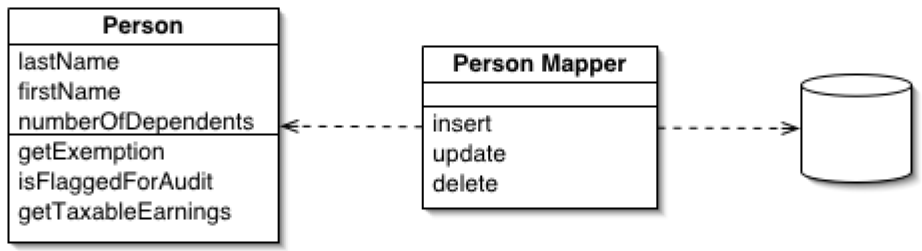


DATA MAPPER

Separates the in memory representation (objects) from the database

The objects are lighter

Single responsibility principle


Person
lastName
firstName
numberOfDependents
getExemption
isFlaggedForAudit
getTaxableEarnings
Person Mapper
insert
update
delete