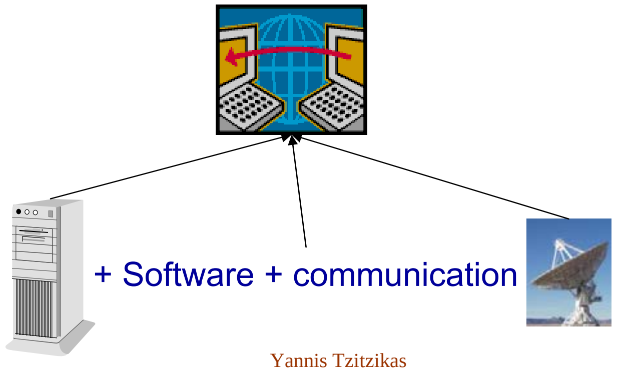
ΗΥ351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design

Σχεδίαση Φυσικής Αρχιτεκτονικής
(Physical Architecture Design)



Γιάννης Τζίτζικας

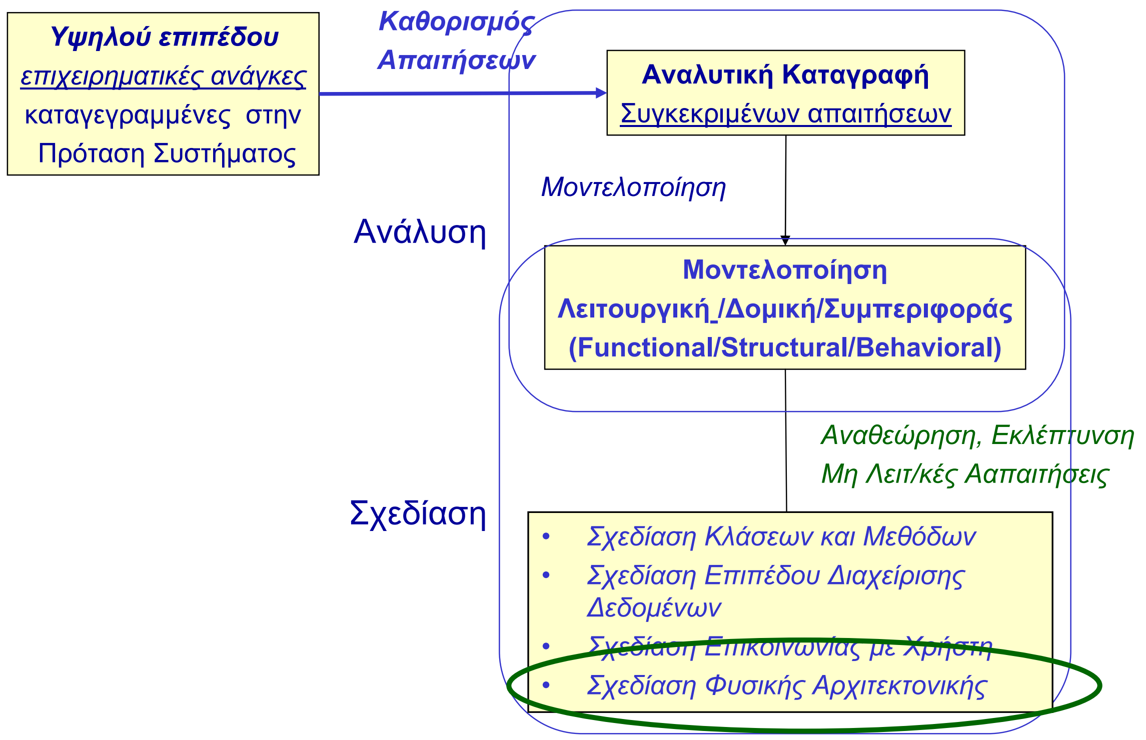
Διάλεξη :
Ημερομηνία : 2008
Θέμα :

Διάρθρωση

- Τι είναι η Σχεδίαση Φυσικής Αρχιτεκτονικής
- Οι 4 βασικές λειτουργίες ενός Πληροφοριακού Συστήματος
- Διαστρωματωμένες Αρχιτεκτονικές Λογισμικού
  - Layered Software Architectures
- Αρχιτεκτονικές Λογισμικού
  - Client-server, N-tier architectures,Virtual machine
  - Service-oriented computing, P2P
- Πρωτόκολλα Επικοινωνίας
- Το σχεδιαστικό μοτίβο MVC

2ο Μέρος (Επόμενο μάθημα):
- Σχετικά Διαγράμματα της UML
  - **Component** and **Deployment** Diagrams

Από τα Μοντέλα Ανάλυσης στα Μοντέλα Σχεδίασης



Τι είναι η Σχεδίαση Φυσικής Αρχιτεκτονικής (ή της αρχιτεκτονικής του συστήματος); System (or Physical) Architecture Design ?

System Architecture Design comprises plans for

(a) the **hardware**,

(b) the **software**,

(c) the **communications**

for the new application.

The 4 primary software components of a system

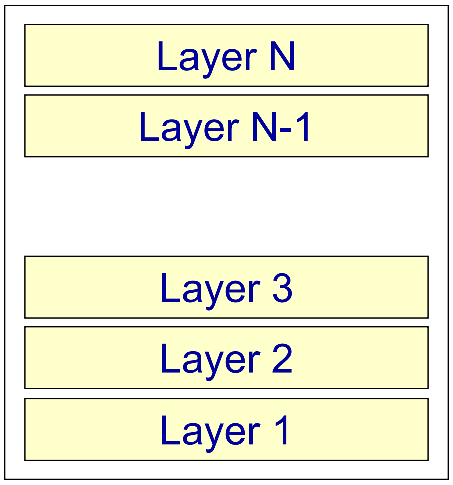
All software systems could be divided into 4 basic functions

- *Data storage*
- *Data access logic*
- *Application logic*
- *Presentation logic*

Διαστρωματωμένα Συστήματα
Layered Systems

- The functionality of the application is partitioned to a set of layers
- Each layer uses the services of the lower layers and offers services to the upper layers

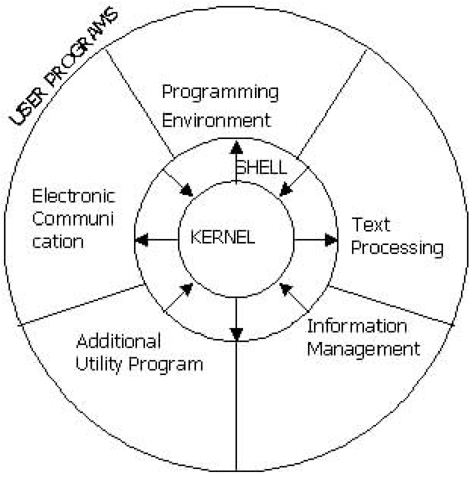
- Advantages
  - Abstraction during design
  - Allow reuse
  - Can define standard layer interfaces
- Disadvantages
  - Sometimes it is difficult to identify with clarity the layers.
  - Sometimes this architecture is not very efficient (redundant)

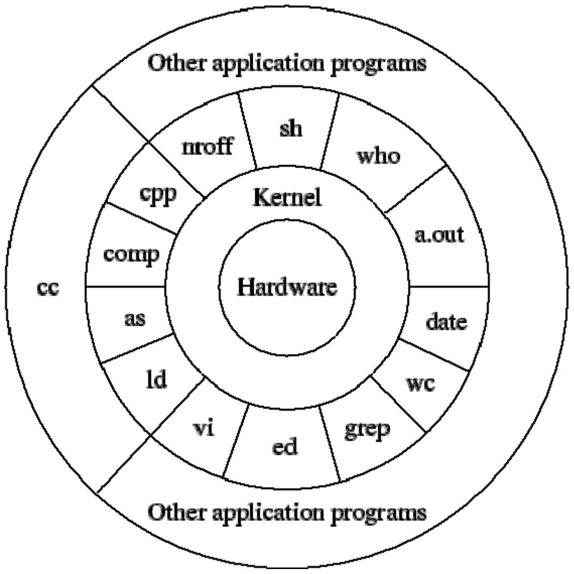


Layering: Διαστρωμάτωση

Διαστρωματωμένα Συστήματα: Παραδείγματα
Examples of Layered Systems

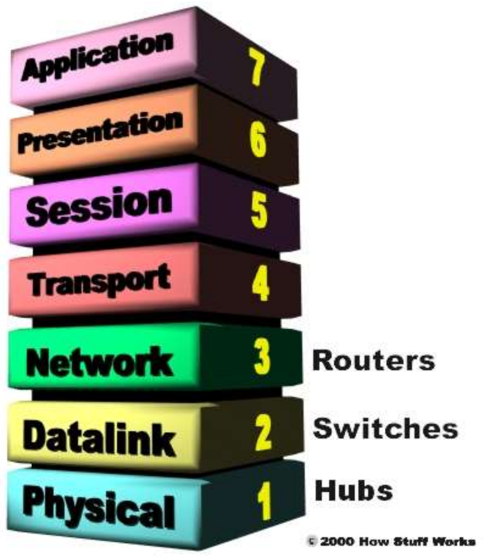
The Unix Operating System

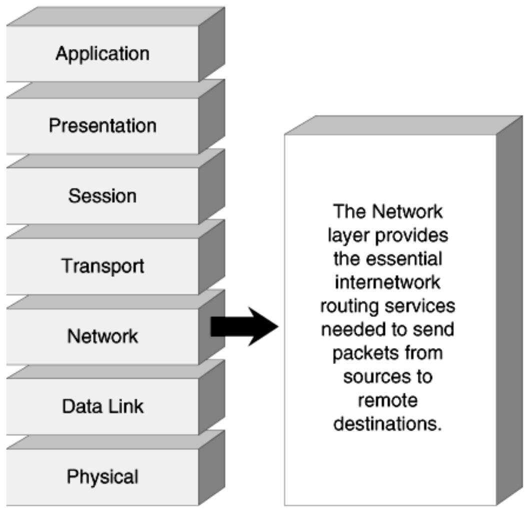




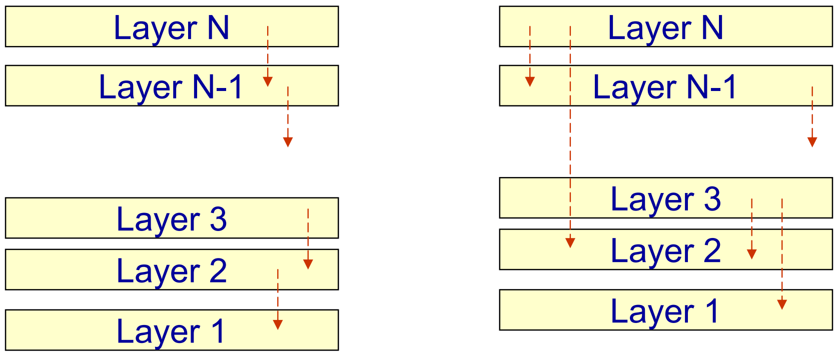
Διαστρωματωμένα Συστήματα: Παραδείγματα
Examples of Layered Systems

OSI Network Protocol




Application
Presentation
Session
Transport
Network
Data Link
Physical
The Network layer provides the essential internetwork routing services needed to send packets from sources to remote destinations.

Διαστρωματωμένη Αρχιτεκονική: **Κλειστή** vs **Ανοικτή**
Layered Architectures: Closed vs Open



Closed

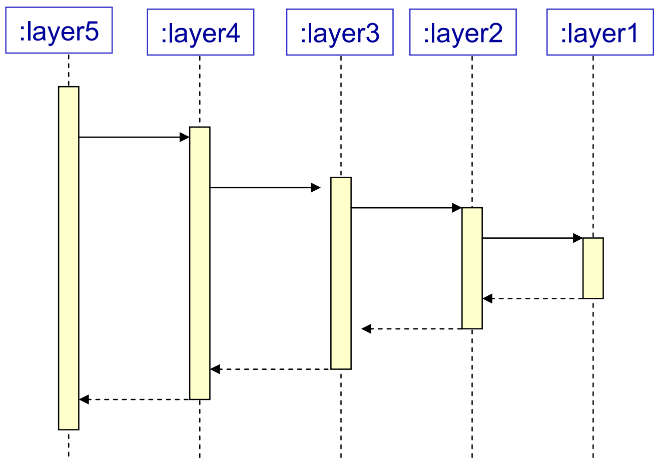
- each layer can use services of the immediately lower layer
- minimizes dependencies

Open

- each layer can use services of any lower layer
- increased dependencies however the code can be more compact

Recall the <u>trade-off between understandability and efficiency</u>: increasing the understandability of a design usually results in inefficiencies, while focusing only on efficiency usually results in design that is difficult to understand by someone else

Ενδεικτικό Διάγραμμα Ακολουθίας μιας κλειστής διαστρωματωμένης αρχιτεκτονικής
The form of sequence diagrams in a closed layered architecture



Παράδειγμα υλοποίησης μιας Κλειστής Διαστρωματωμένης Αρχιτεκτονικής
Example of implementing a Closed Layered Architecture

```
public abstract class L1Provider {
    public abstract void L1Service();
}

public abstract class L2Provider {
    protected L1Provider level1;

    public abstract void L2Service();
    public void setLowerLayer(L1Provider l1)
        {
        level1 = l1;
    }
}

public abstract class L3Provider {
    protected L2Provider level2;

    public abstract void L3Service();
    public void setLowerLayer(L2Provider l2)
        {
        level2 = l2;
    }
}
```

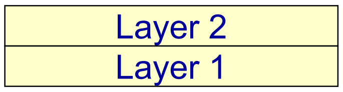
```
public class DataLink extends L1Provider {
    public void L1Service() {
        println("L1Service doing its job");
    }
}
public class Transport extends L2Provider
        {
    public void L2Service() {
        println("L2Service starting its job");
        level1.L1Service();
        println("L2Service finishing its job");
    }
}
public class Session extends L3Provider{
    public void L3Service() {
        println("L3Service starting its job");
        level2.L2Service();
        println("L3Service finishing its job");

    }
}
```

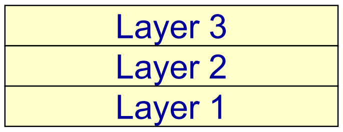
```
public class Network {
    public static void main(String args[]) {
        DataLink dataLink = new DataLink();
        Transport transport = new
        Transport();
        Session session = new Session();

        transport.setLowerLayer(dataLink);
        session.setLowerLayer(transport);

        session.L3Service();
    }
}
```

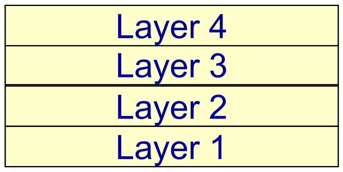
EXECUTION RESULT:

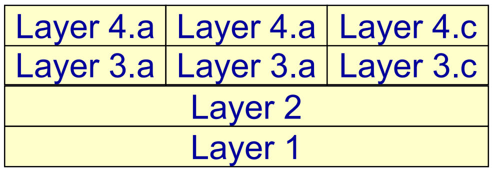
```
L3Service starting its job
L2Service starting its job
L1Service doing its job
L2Service finishing its job
L3Service finishing its job
```

Πλήθος στρωμάτων
Number of Layers



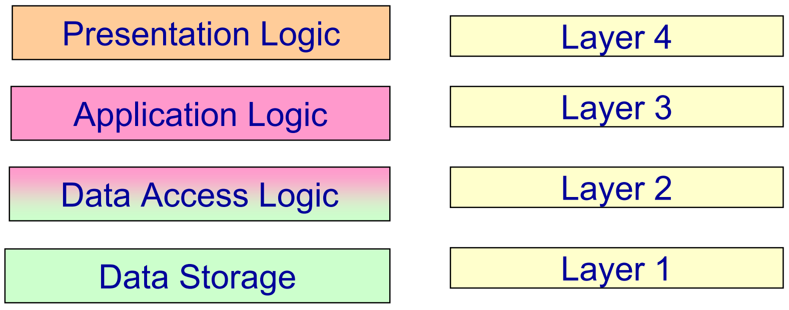




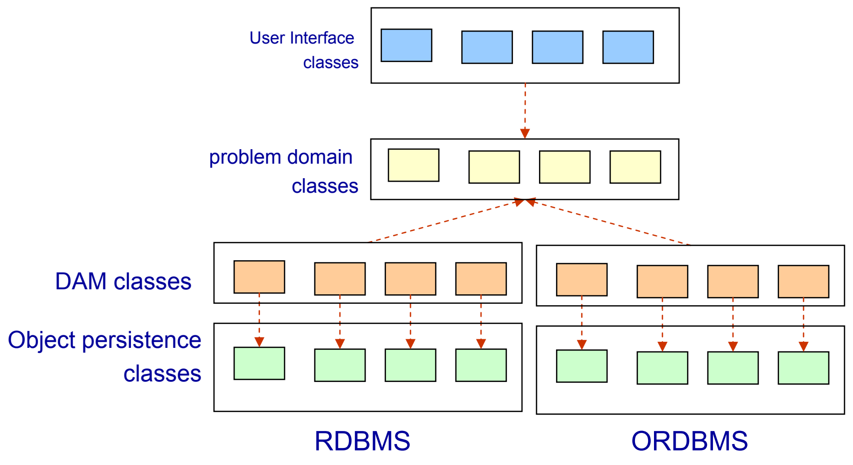


Θεωρώντας τα 4 βασικά συστατικά λογισμικού ενός ΠΣ ως στρώματα
Considering the 4 primary software components of an IS as Layers

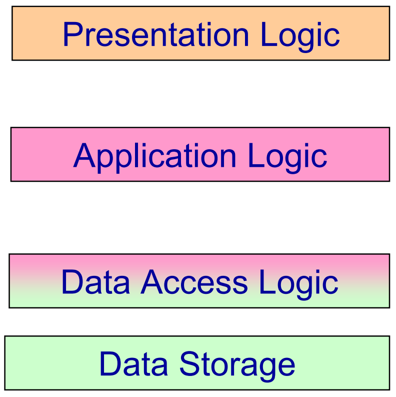
- Presentation logic
- Application logic
- Data access logic
- Data storage



Υπενθυμιστικό: Σχεδίαση Στρώσης Διαχείρισης Δεδομένων
Refresher: Data Mgmt Layer Design



- *Notice that in this way the problem domain classes remain unchanged*
- *We have kept them independent from the underlying database management system.*
- *Changing DBMS requires changing only the DAM classes*



Τα 3 κυριότερα εξαρτήματα υλικού ενός συστήματος
The 3 primary hardware components of a system

- Client computers
- Servers
- Network

Τύποι Αρχιτεκτονικών
Kinds of Architectures

Primary **Hardware** components

- Client computers
- Servers
- Network

X

Primary **Software** components

Presentation Logic

Application Logic

Data Access Logic

Data Storage

= *architectures*

According to the distribution of the 4 basic layers to hardware nodes we can distinguish the following architectures:

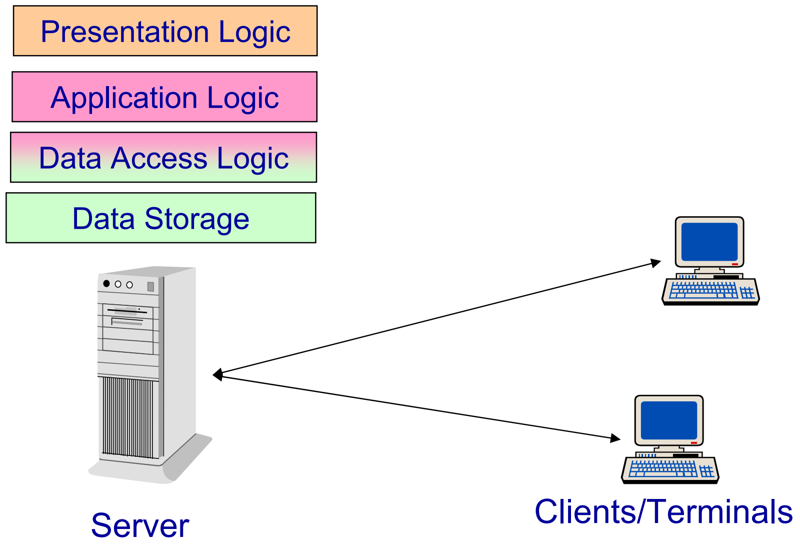
(a) **Server-based** computing

(b) **Client-based** computing

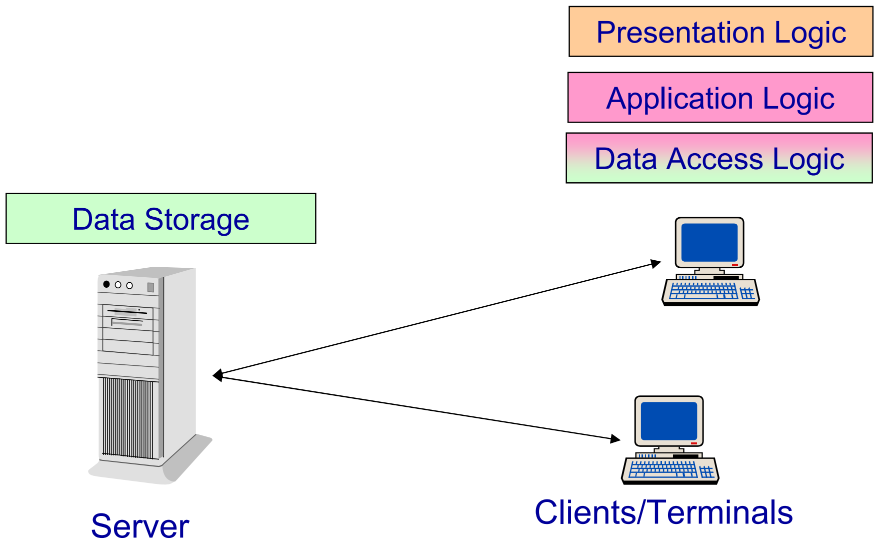
(c) **Client-server-**based computing

(d) **3/4/N tiers** computing

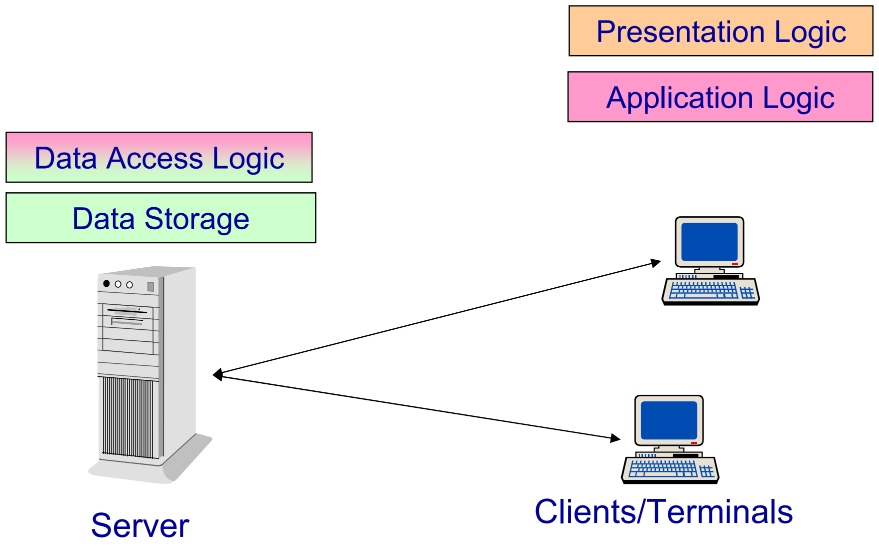
(a) Server-based Computing



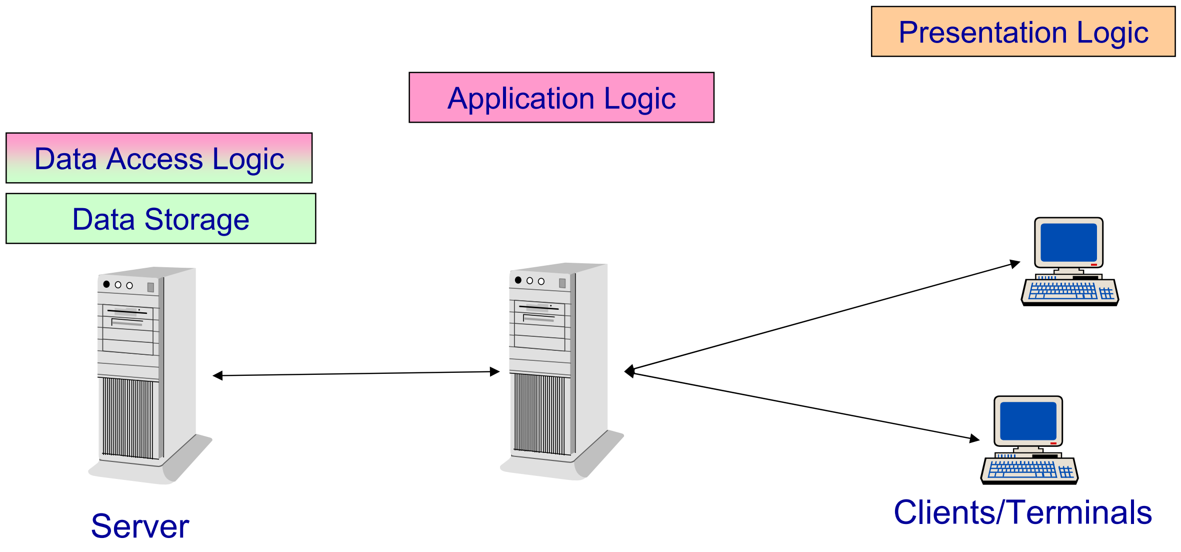
(b) Client-based Computing



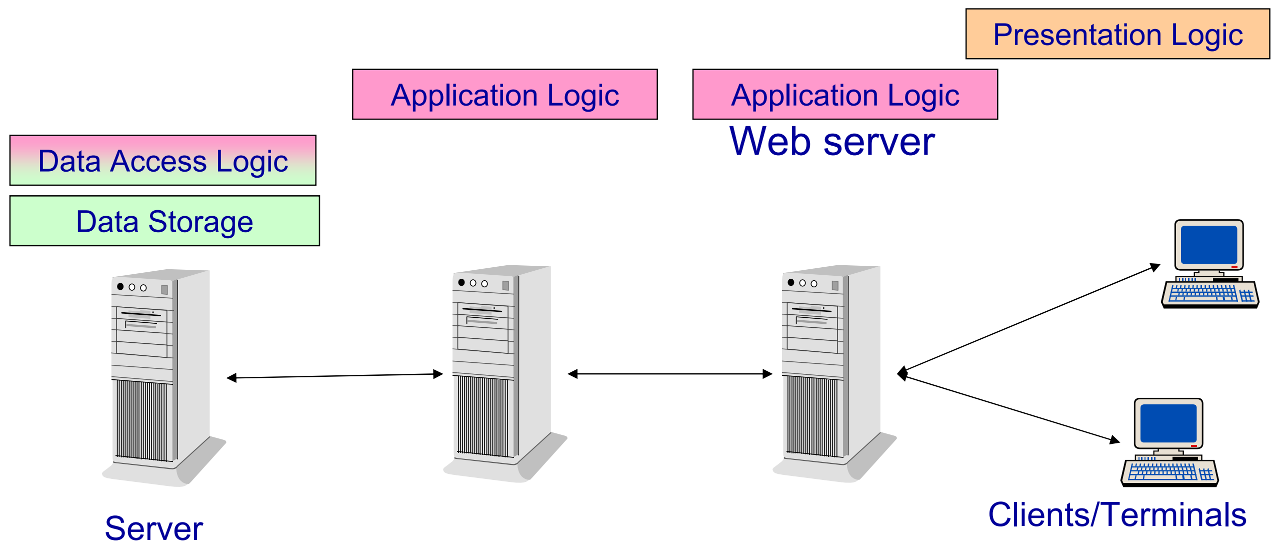
(c) Client-Server-based Computing (2 Tiers)



(d) 3 tiers based computing

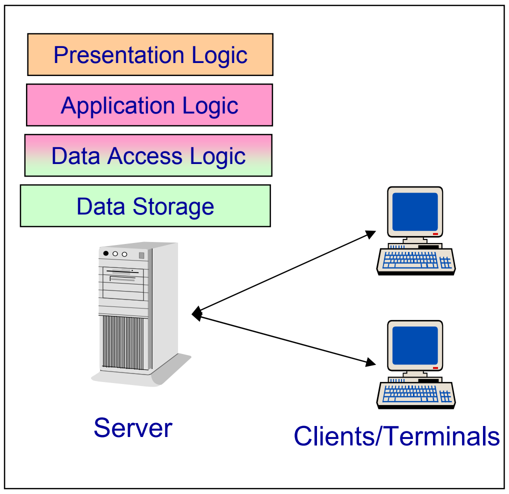


(d') 4 tiers based computing



Some more details about the previous architectures

(a) Server-based Computing



Characteristics

• The server does almost everything. The client is actually a very thin client

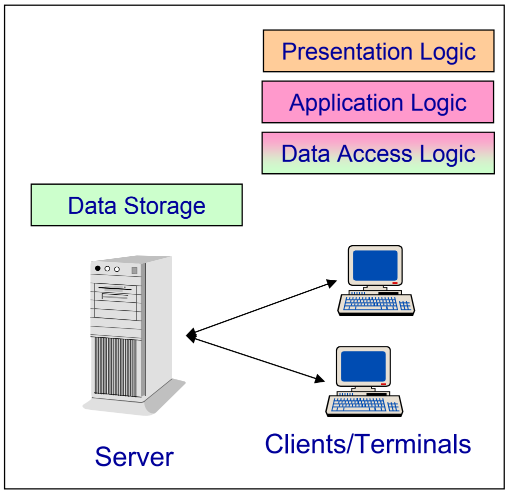
[-]: The Server has very high load

•the clients do not contribute to the computation

[+]: Not so difficult to implement

[+] If platform changes (e.g. OS) we have to rewrite only the thin client

(b) Client-based Computing



Characteristics

[+] The server has less load

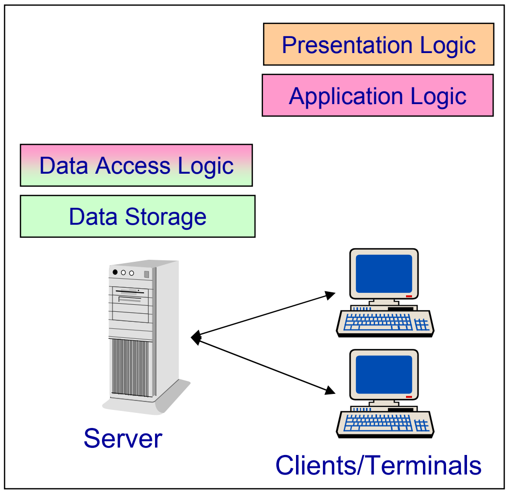
[-] The clients are very heavy (they should be computationally powerful machines)

[-] sometimes a lot of data have to be communicated through the network

[-] If we the OS changes then we have to rewrite the 3 layers of the client

•(in server-based computing we could keep the server running in the old OS) and we would need to change only the thin client so that to run in the new OS

(c) Client-Server-based Computing (2 Tiers)

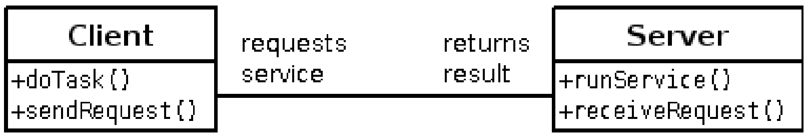


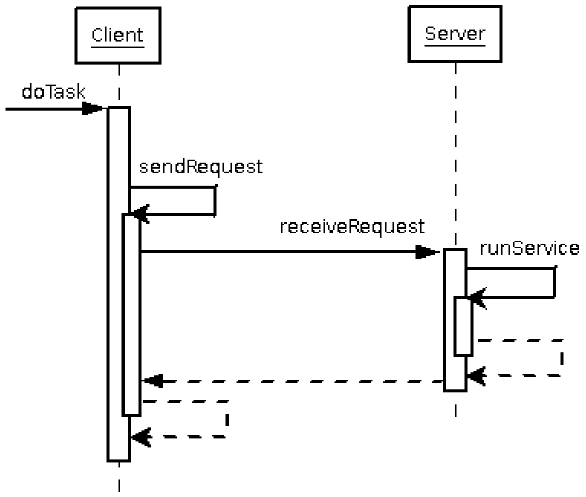
Characteristics

This is like having a **thick client** (thin client: if responsible only for the UI)

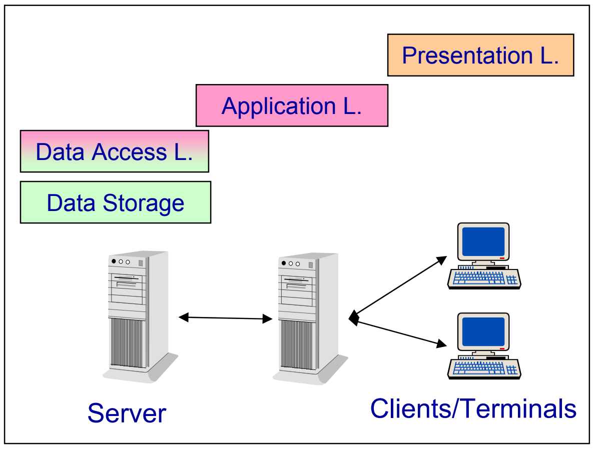
- **[+] The client has less load (comparing to client-based computing)**
- **[+] The server has less load comparing to server-based computing**
- **[-] We have to rewrite the application logic if platform changes**
- **[-] Sometimes a lot of data have to be communicated through the network**
- **[+] Good overall performance**

Client Server: Class and Interaction Diagrams





(d) 3 tiers based computing



Characteristics

[+] Good load balancing

the server and the client have less load

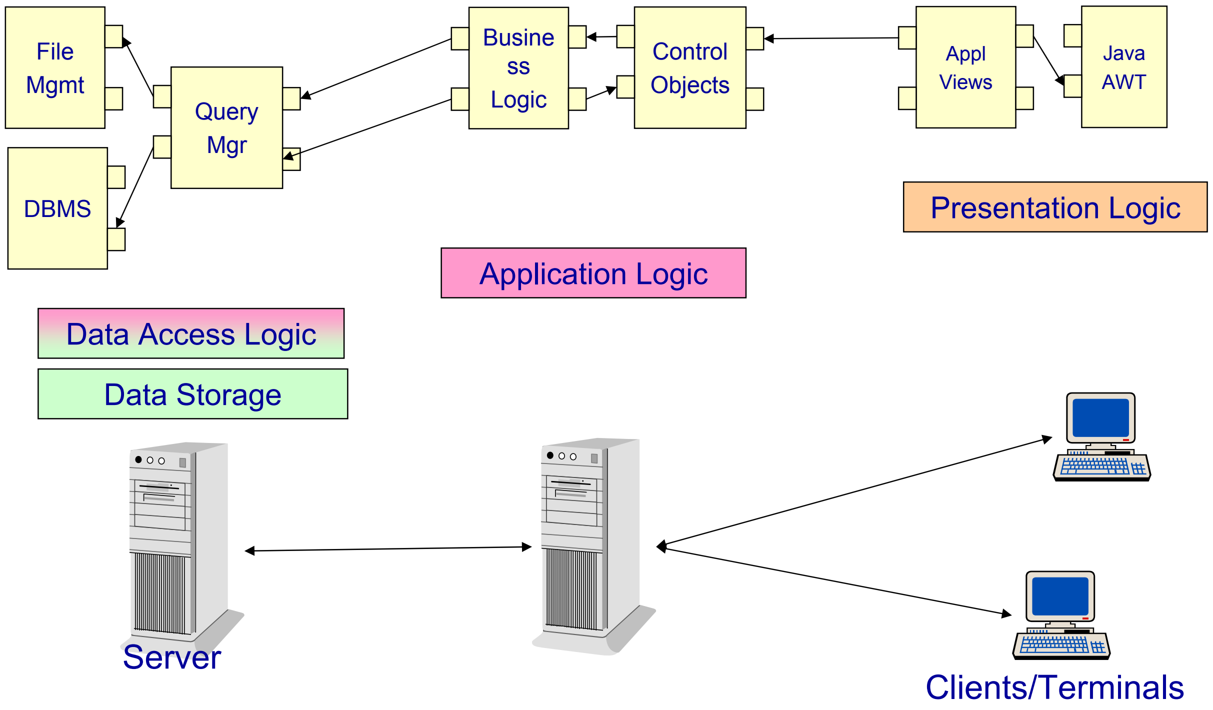
[+] the UI component is independent of the rest system

•server-based computing has also this property but in that case the server has excessive load

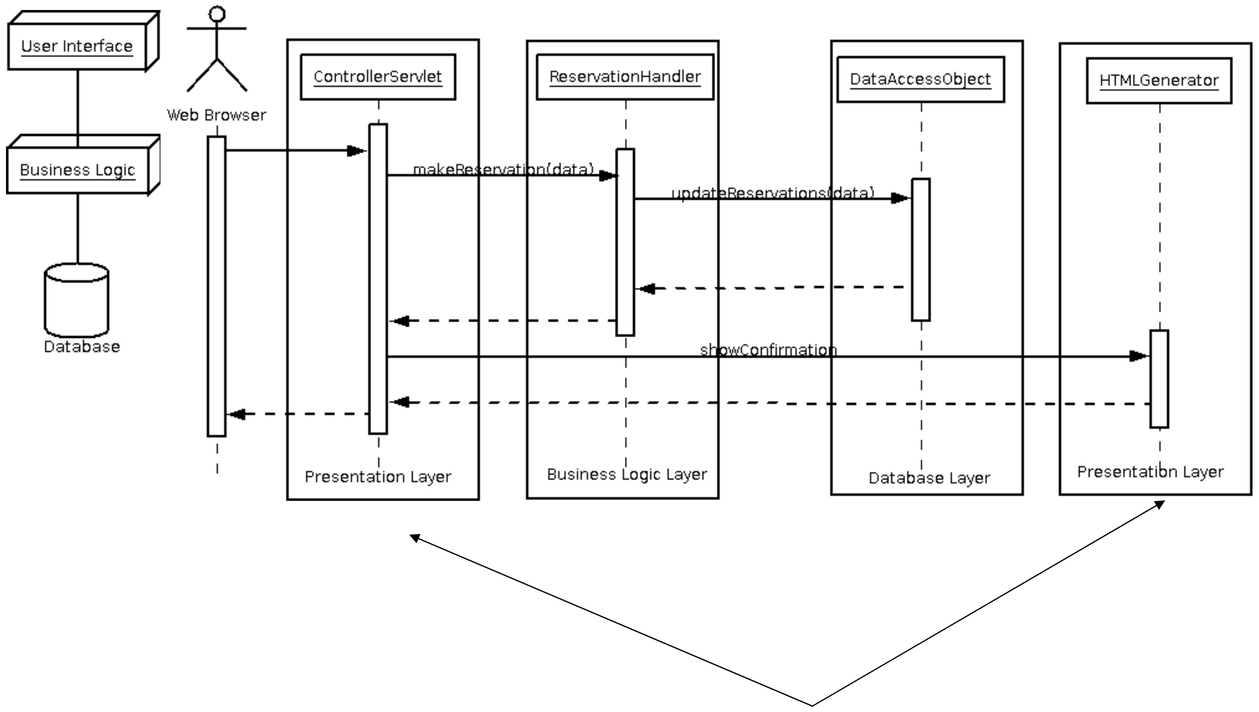
• This architecture is suited for heterogeneous environments

[-] more complex implementation - more data are transferred through the network

(d) 3 tiers: Example



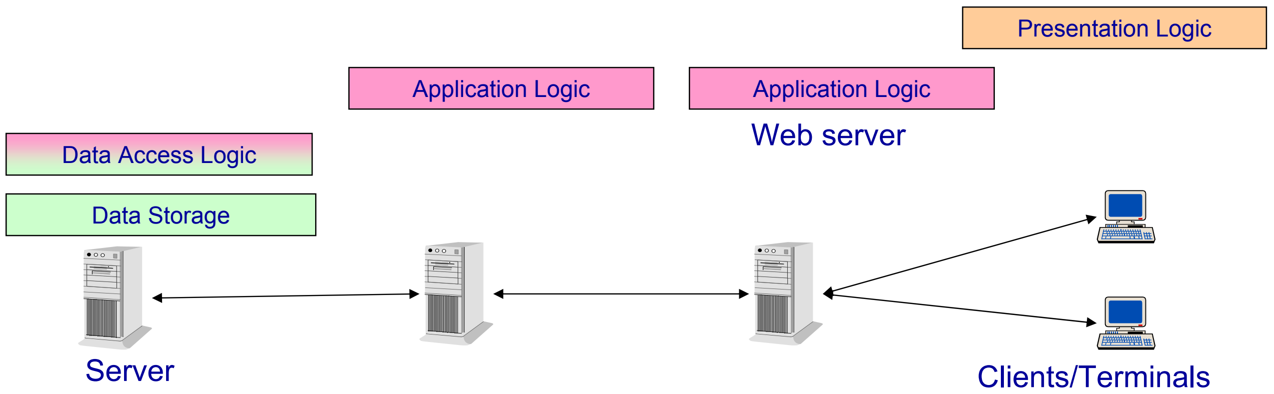
3-tier



(d) N-Tiered Client-Server architectures

General Remarks

- Advantages
  - Separates processing to better balance load
  - The system is more scalable
- Disadvantages
  - Higher load on the network
  - More difficult to implement and test



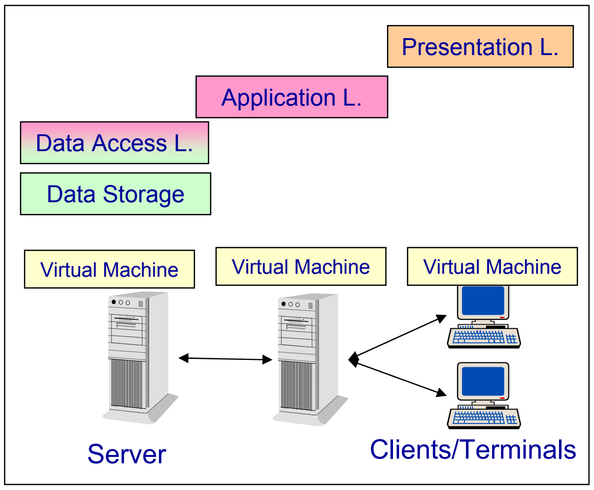
Επιλέγοντας Αρχιτεκτονική Λογισμικού
Selecting a Computing Architecture

	Server-Based	Client-based	Client-server
Cost of infrastructure	Very high	Medium	Low
Cost of development	Medium	Low	High
Ease of development	Low	High	Low-medium
Interface capabilities	Low	High	High
Control and security	High	Low	Medium
Scalability	Low	Medium	High

Virtual Machine

Εικονική Μηχανή
Virtual Machine

- It is a form of layered architecture
- It allows using the same API independently of the underlying OS/hardware
- The compiler produces intermediate code (bytecodes in Java) which can be handled by the virtual machine

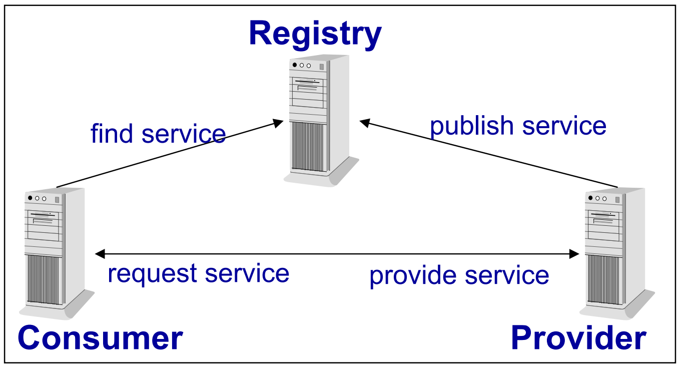


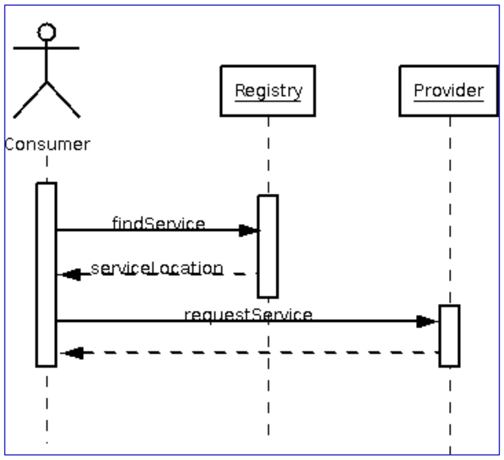
Service-Oriented Computing
Υπηρεσιοστρεφής Υπολογισμός

Υπηρεσιοστρεφής Υπολογισμός
Service-oriented Computing

SOA: Service Oriented Architecture

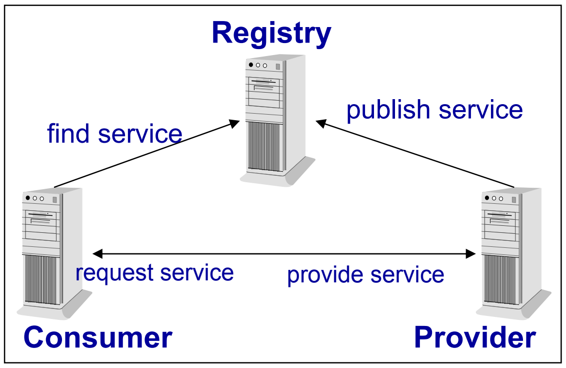
Software is considered as a set of services
We can have
- **service providers**
- **consumers**
- **registries** (catalogs of available services)





Υπηρεσιοστρεφής Υπολογισμός (2)
Service-oriented Computing (2)

- Based on open standards (SOAP, REST, WSDL, UDDI)
- Data is exchanged using XML



Characteristics

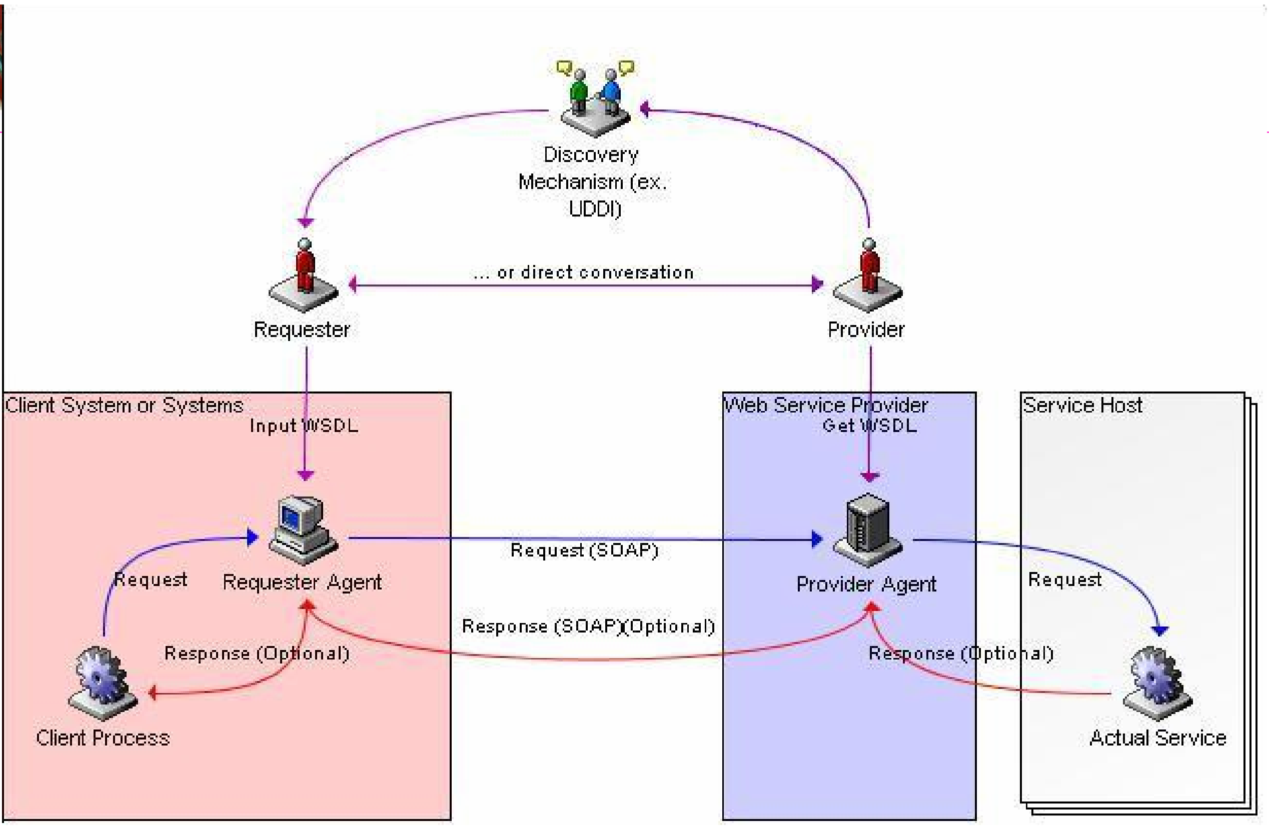
[+] complete separation between providers & consumers

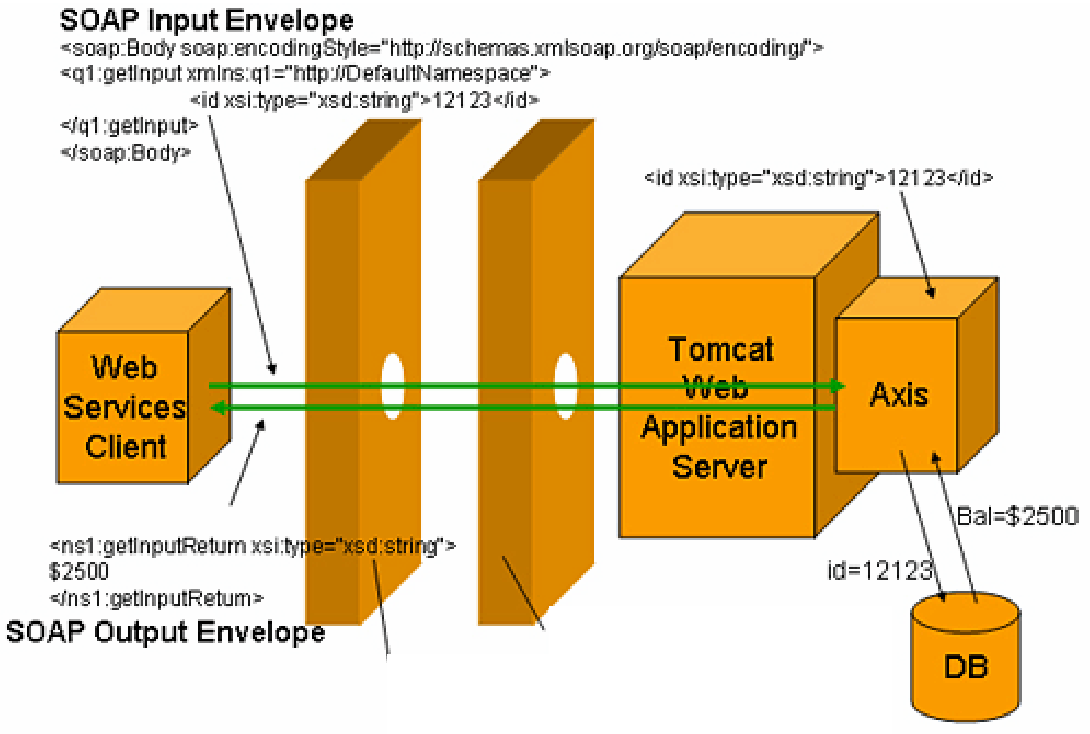
[+] the same service can be provided with different characteristics (quality, price, speed, etc) from different providers => competitiveness

[+] open standards

[-] not mature technology, no registries for business services

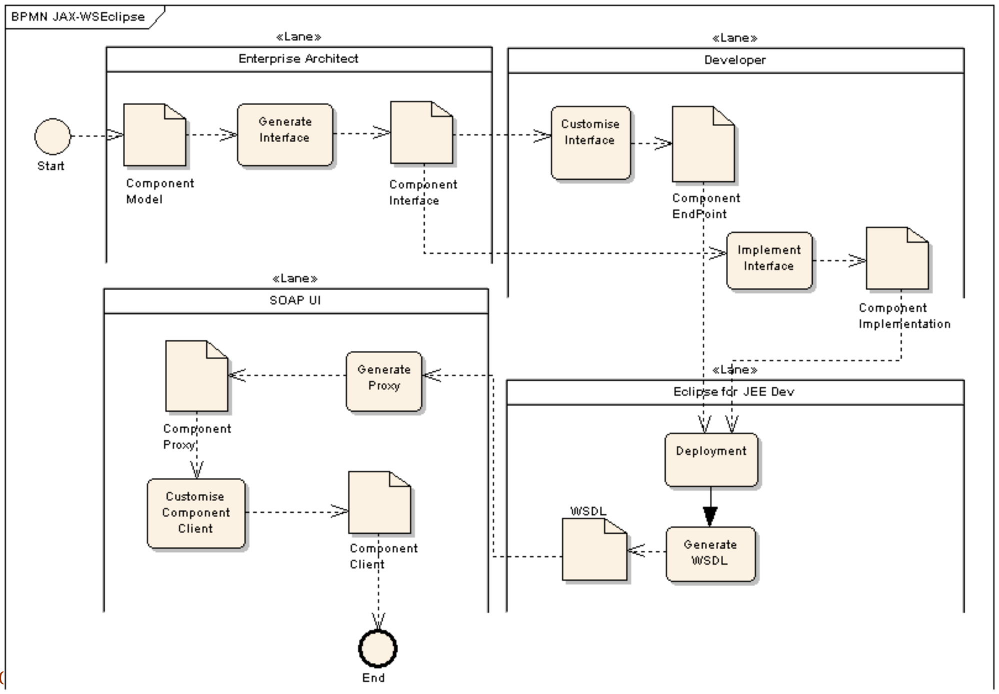
- Web Services
  - Data are exchanged in XML (SOAP, REST)
  - Data are transferred using HTTP
  - The “interface” provider-consumer is described in XML (WSDL)


Discovery Mechanism (ex. UDDI)
... or direct conversation
Requester
Provider
Client System or Systems
Input WSDL
Web Service Provider
Get WSDL
Service Host
Request (SOAP)
Request
Requester Agent
Provider Agent
Request
Response (SOAP)(Optional)
Response (Optional)
Response (Optional)
Client Process
Actual Service

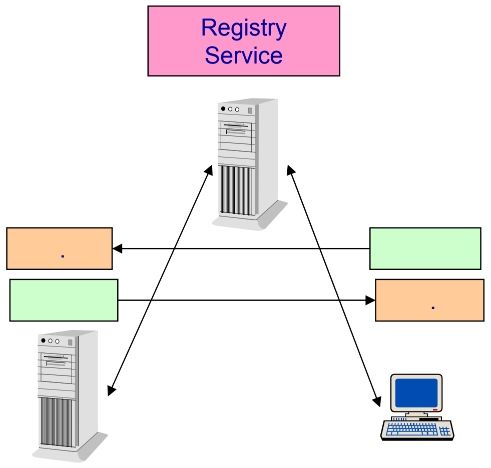

SOAP Input Envelope
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<q1:getInput xmlns:q1="http://DefaultNamespace">
<id xsi:type="xsd:string">12123</id>
</q1:getInput>
</soap:Body>
<id xsi:type="xsd:string">12123</id>
Web Services Client
Tomcat Web Application Server
Axis
Bal=$2500
id=12123
DB
<ns1:getInputReturn xsi:type="xsd:string">
$2500
</ns1:getInputReturn>
SOAP Output Envelope

Exampe:
Developing web services using JAX-WS

- JAX-WS: standard to support JAVA Web Services (evolution of the well known and widely adopted JAX-RPC)



Υπηρεσιοστρεφής Υπολογισμός (3)
Service-oriented computing (3)



Extra Reading

- [1] “Model-driven Web Services Development“
  – http://folk.uio.no/roygr/EEE-2004.pdf
- [2] “From UML to BPEL (*Model Driven Architecture in a Web services world)”*
  – http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/

Service-oriented computing and UML (Based on [1])

- With UML we can express the contents and behavior of web services in a more understandable way than WSDL

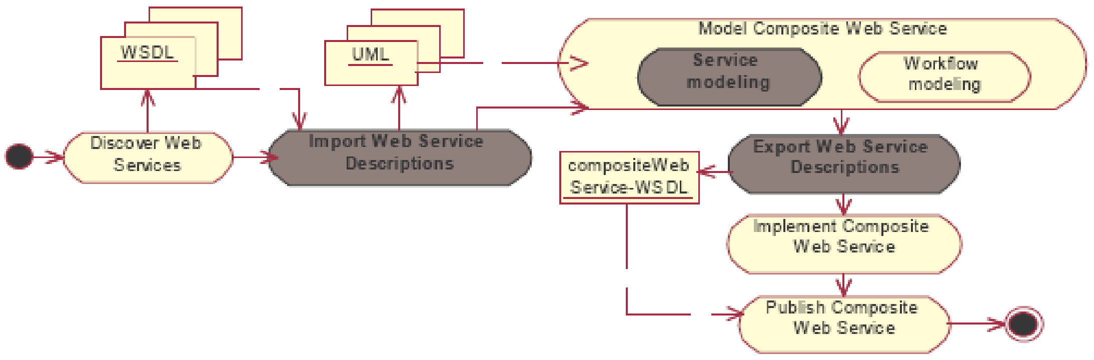
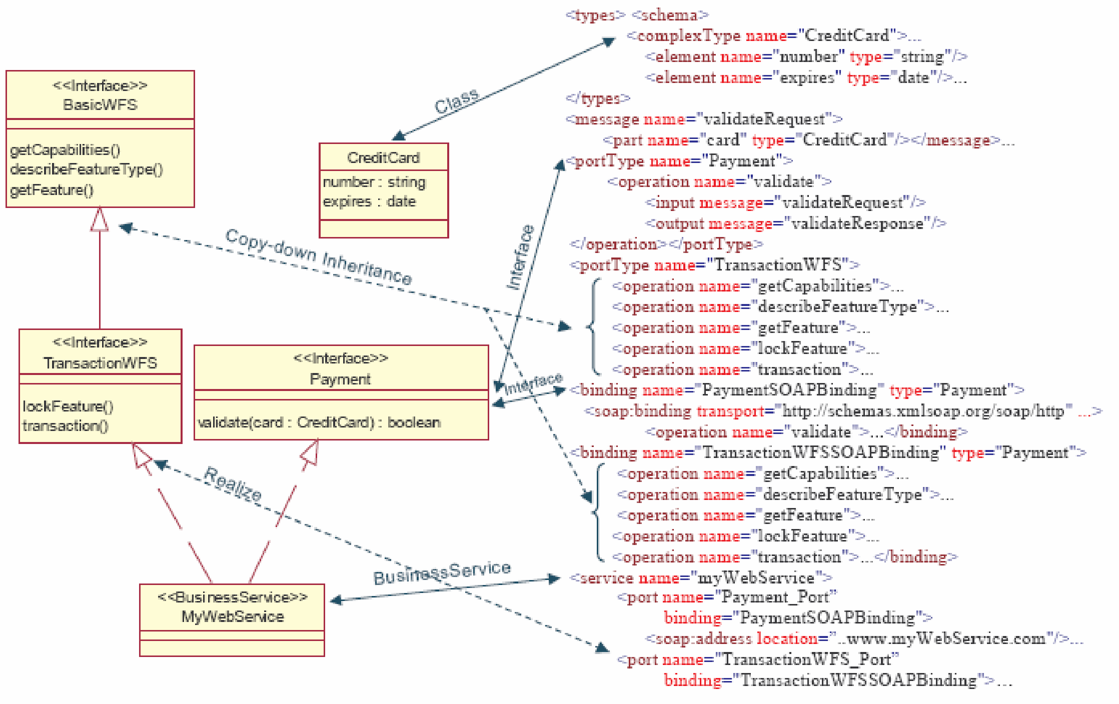

WSDL
UML
Model Composite Web Service
Service modeling
Workflow modeling
Discover Web Services
Import Web Service Descriptions
compositeWeb Service-WSDL
Export Web Service Descriptions
Implement Composite Web Service
Publish Composite Web Service

Figure 1: Steps of model-driven web services development

Service-oriented computing and UML (Based on [1])



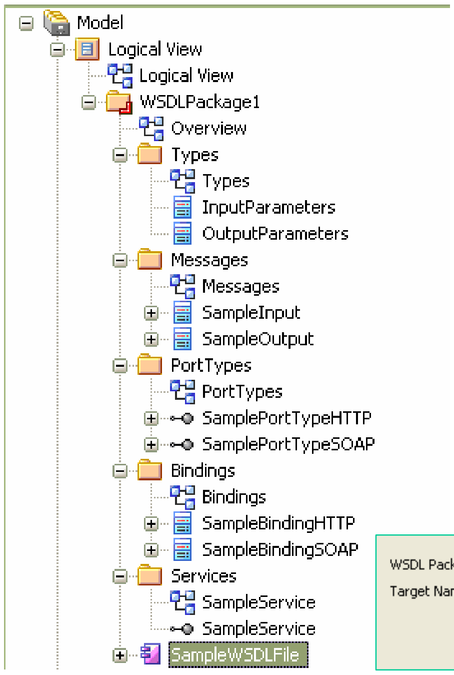
```
<types> <schema>
    <complexType name="CreditCard">...
        <element name="number" type="string"/>
        <element name="expires" type="date"/>...
</types>
<message name="validateRequest">
    <part name="card" type="CreditCard"/></message>...
<portType name="Payment">
    <operation name="validate">
        <input message="validateRequest"/>
        <output message="validateResponse"/>
</operation></portType>
<portType name="TransactionWFS">
    <operation name="getCapabilities">...
    <operation name="describeFeatureType">...
    <operation name="getFeature">...
    <operation name="lockFeature">...
    <operation name="transaction">...
<binding name="PaymentSOAPBinding" type="Payment">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" ...>
        <operation name="validate">...</binding>
<binding name="TransactionWFSSOAPBinding" type="Payment">
    <operation name="getCapabilities">...
    <operation name="describeFeatureType">...
    <operation name="getFeature">...
    <operation name="lockFeature">...
    <operation name="transaction">...</binding>
<service name="myWebService">
    <port name="Payment_Port"
          binding="PaymentSOAPBinding">
        <soap:address location="..www.myWebService.com"/>...
    <port name="TransactionWFS_Port"
          binding="TransactionWFSSOAPBinding">...
```

Figure 2: Conversion between a UML model and a WSDL document

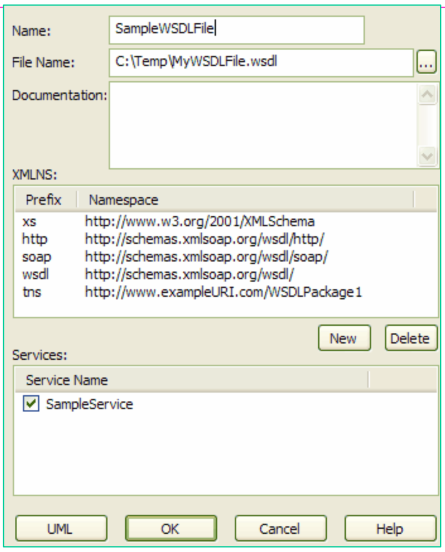
From UML to BPEL (Based on [2])

- The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) is an XML-based standard for defining how you can combine Web services to implement business processes. It builds upon the Web Services Definition Language (WSDL) and XML Schema Definition (XSD).
- [2] describes a tool which takes processes defined in the Unified Modeling Language (UML) and generates the corresponding BPEL and WSDL files to implement that process.
- A UML Profile is used for defining stereotypes relating to Business Process Execution Language for Web Services. A Mapping is provided for automatically generating Web services artifacts (BPEL, WSDL, XSD) from a UML model meeting the profile.

Web Services Support in Enterprise Architect: It supports modeling, importing, and generation



```
Model
  Logical View
    Logical View
    WSDLPackage1
      Overview
      Types
        Types
        InputParameters
        OutputParameters
      Messages
        Messages
        SampleInput
        SampleOutput
      PortTypes
        PortTypes
        SamplePortTypeHTTP
        SamplePortTypeSOAP
      Bindings
        Bindings
        SampleBindingHTTP
        SampleBindingSOAP
      Services
        SampleService
        SampleService
      SampleWSDLFile
```



Name: SampleWSDLFile

File Name: C:\Temp\MyWSDLFile.wsdl

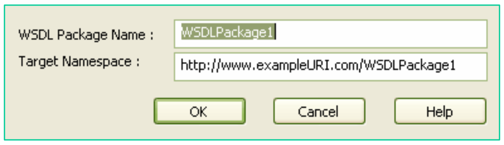
Documentation:

XMLNS:

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
http	http://schemas.xmlsoap.org/wsdl/http/
soap	http://schemas.xmlsoap.org/wsdl/soap/
wsdl	http://schemas.xmlsoap.org/wsdl/
tns	http://www.exampleURI.com/WSDLPackage1

Services:

Service Name
✓ SampleService



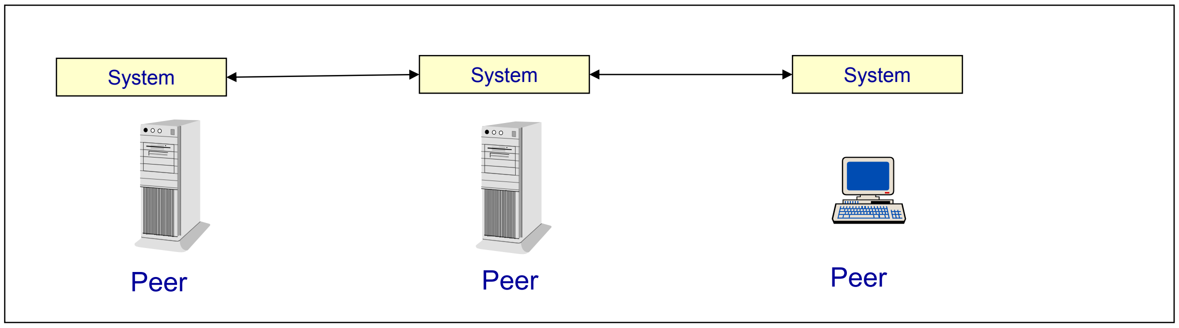
WSDL Package Name : WSDLPackage1

Target Namespace : http://www.exampleURI.com/WSDLPackage1

Διομότιμες Αρχιτεκτονικές
Peer-to-Peer (P2P) architectures

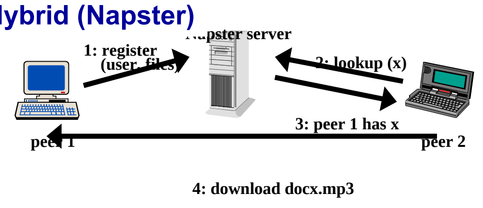
Peer to Peer

- Pure
  - all are equal. No layering. Each peer depends on the others

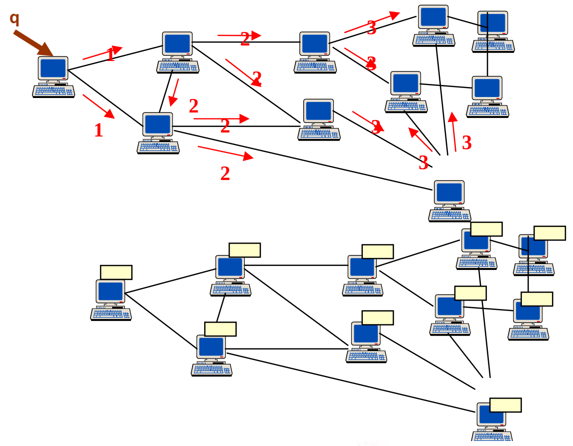


Peer-to-Peer Architectures

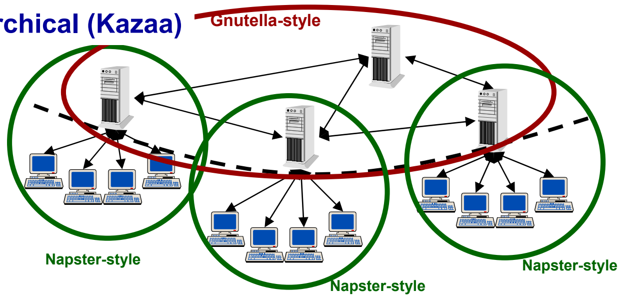
Hybrid (Napster)



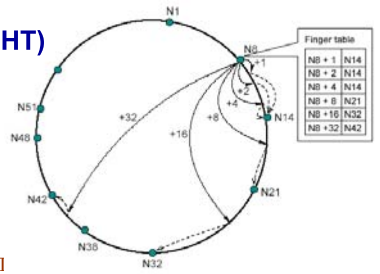
Decentralized (Gnutella)



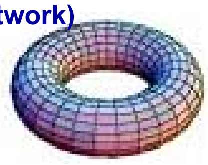
Hierarchical (Kazaa)



Chord (DHT)



CAN (Content Addressable Network)



Πρωτόκολλα Επικοινωνίας
Communication Protocols

Πρωτόκολλα Επικοινωνίας
Communication Protocols

How objects of different layers at different machines can communicate ?

- RPC (Remote Procedure Call):
  - can invoke a remote procedure, send results, (RPC is widely supported in languages such as C, C++)
- RMI (Remote Method Invocation)
  - in java (recall *www.csd.uoc.gr/~hy252*)
- DCOM
  - Microsoft's Distributed Component Object Model
- CORBA (Common Object Request Broker Architecture)
  - The object-oriented industry standard by OMG (1995)
- SOAP (Simple Object Access Protocol)
  - uses XML to encapsulate messages and data that can be sent from one process to another

Communication Protocols
Platform dependent vs Open Standards

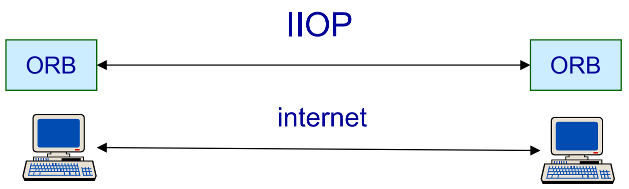
- RMI or DCOM are language/operating system specific protocols
  - they restrict the design to implementation on certain platforms
- CORBA or SOAP are open standards
  - they allow building component-based systems that are not tied to particular platforms

Case: CORBA

- CORBA separates the interface of a class (the operations it can carry out) from the implementation of that class.
- The interface can be compiled into a program running on one computer.
- An object instance can be created or accessed by name.
- To the client program it appears to be in memory on the same machine, however, it may actually be running on another computer.
- When the client program sends it a message to invoke one of its operations, the message and its parameters are converted into a format that can be sent over the network (known as *marshalling*). At the other end the server unmarshals the data back into a message and parameters and passes it to the implementation of the target object.
- This object then carries out the operation and, if it returns a value, that value is marshalled on the server, unmarshalled on the client and finally provided as a return value to the client program

CORBA (2)

- CORBA achieves this by means of programs known as ORBs (Object Request Brokers) that run on each machine.
- The ORBs communicate with each other by means of an Inter-ORB Protocol (IOP).
- Over the Internet, the protocol used is IIOP (Internet IOP).



CORBA (3)

- To use this facility, the developer must specify the interface (public attributes and operations) of each class in an **Interface Definition Language (IDL).**
- The IDL file is then processed by a program that converts the interface to a series of files in the target language or languages.

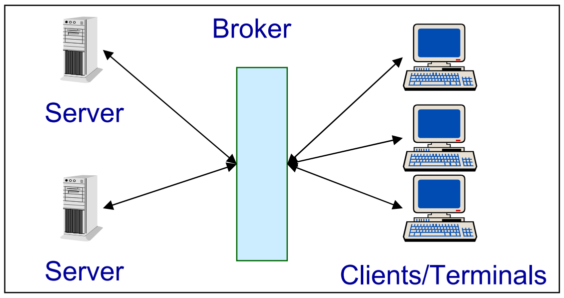
- In Java, this program is called IDL2JAVA and produces
  - a file that defines the interface in Java,
  - a stub file that provides the link between the client program and the ORB,
    - it implements the interface on the client and is compiled into the client program
  - a file that provides a skeleton for the implementation of the server
    - it implements the interface on the server; the developer updates this file (provides the implementation) and it is compiled on the host

The IDL file for a class Location

```
Module CretanTourismApplication
{ interface Location
   {attribute string locationCode;
    attribute string locationName;
    void addHotel(in Hotel hotel);
    void removeHotel(in string hotelCode);
    int numberOfHotels(); };
};
```

CORBA (4)
Supporting different PLs, Wrapping legacy systems

- CORBA is known as **middleware**, as it acts as an intermediary between clients and servers.As such it enables the implementation of a 3 or 4 tier architecture that isolates the UI and client programs from the implementation of classes on one or more servers.



- CORBA also provides **interoperability between different languages**: a Java client program can invoke operations on a C++ object that exist on a separate machine.
- CORBA also makes it possible to encapsulate pre-existing programs (legacy systems) written in non-object oriented languages by **wrapping them in an interface**. To the client it looks like an object, but internally it may be implemented in a language like COBOL.

CORBA (5)
More advanced features

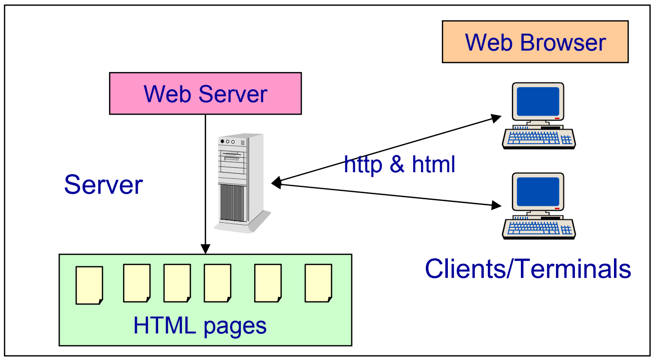
Systems developed using CORBA can be set up so that the remote objects are located on a named machine and accessed by name. This is what we need in the majority of applications.

CORBA also provides a number of more advanced services:

- Services for locating objects by name when it is not known where they are running.
- Services for locating objects that implement a certain interface and for interrogating an object to determine its interface (operations, parameter types and return types) in order to dynamically invoke its operations.

Ιστο-βασισμένες εφαρμογές
Web-based applications

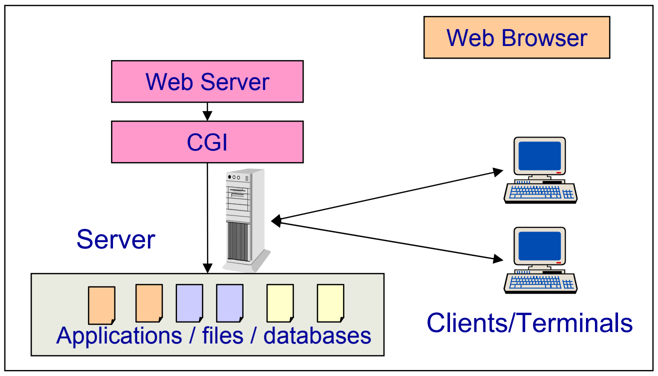
HTTP (HyperText Transfer Protocol): transfers hypertext documents over the internet
- HTML (HyperText Markup Language): defines hypertext documents

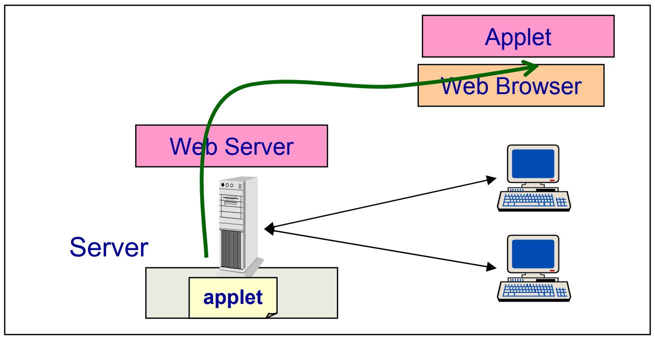


Very static architecture

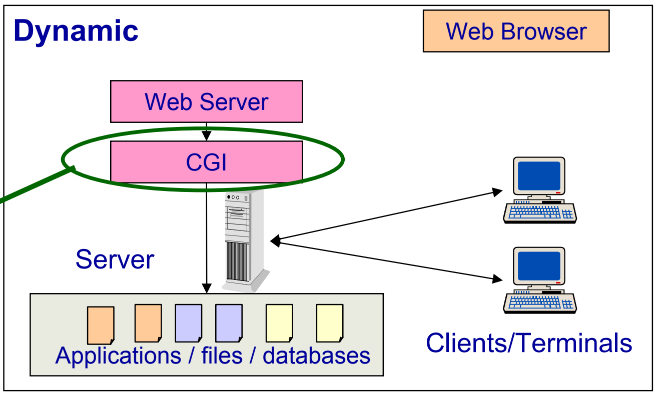
Web-based applications: Adding .. "dynamism"

CGI (Common Gateway Interface):
CGI scripts are programs (e.g. a unix shell script or a perl script) that reside on the web server and can be invoked by elements of the web pages





Web-based applications

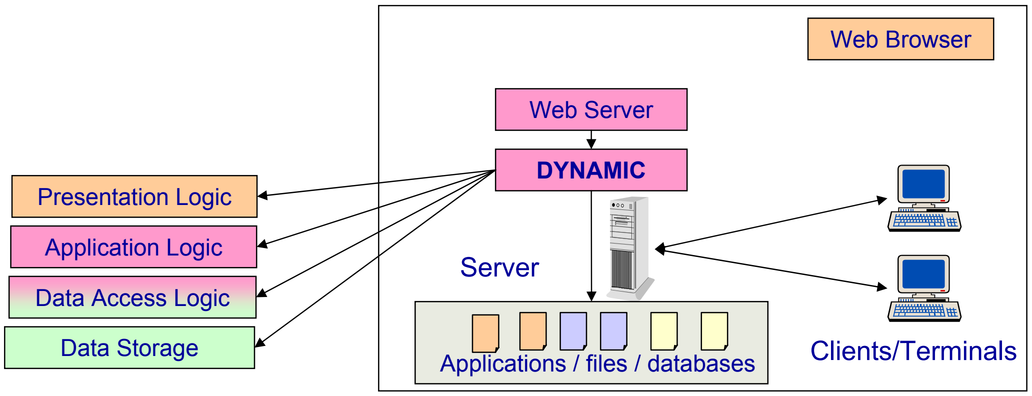


alternatives

- ASP (Active Server Pages)
  - limited to Microsoft Platform
- JSP (Java Server Pages)
  - JSP is designed to be platform and server independent, created from a broader community of tool, server, and database vendors

Web-based applications

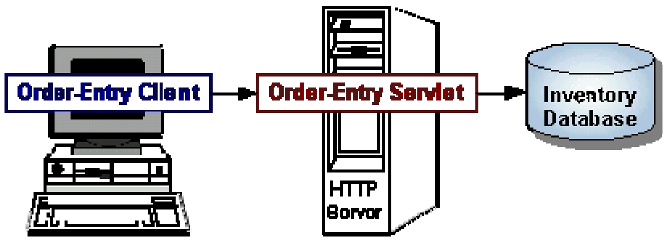
- Here we have to design our layers assuming the Web platform



- So Web Servers and the Web Browsers become parts of our information system.

Servlets

- **<u>Servlets are to servers what applets are to browsers</u>**.
- Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers.
  - A servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.
- Servlets can be embedded in many different servers because the servlet API, which you use to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers; many web servers support Java Servlet technology.



Servlets (II)

Servlets are an effective replacement for CGI scripts.
- They are easier to write and run faster

So we can use servlets to handle HTTP client requests.
- We can have servlets to process data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet like this could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

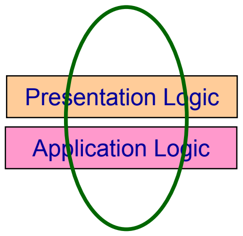
Other Uses for Servlets
- A servlet can handle multiple requests concurrently, and can synchronize requests. This allows servlets to support systems such as on-line conferencing.
- Servlets can forward requests to other servers and servlets. Thus servlets can be used to balance load among several servers that mirror the same content, and to partition a single logical service over several servers, according to task type or organizational boundaries.

A Simple Servlet (Hello World)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse
      response)
  throws IOException, ServletException
  {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
  }
}
```

Σχεδιαστικό Μοτίβο
Model-View-Controller (MVC)

Pattern
Model-View-Controller (MVC)

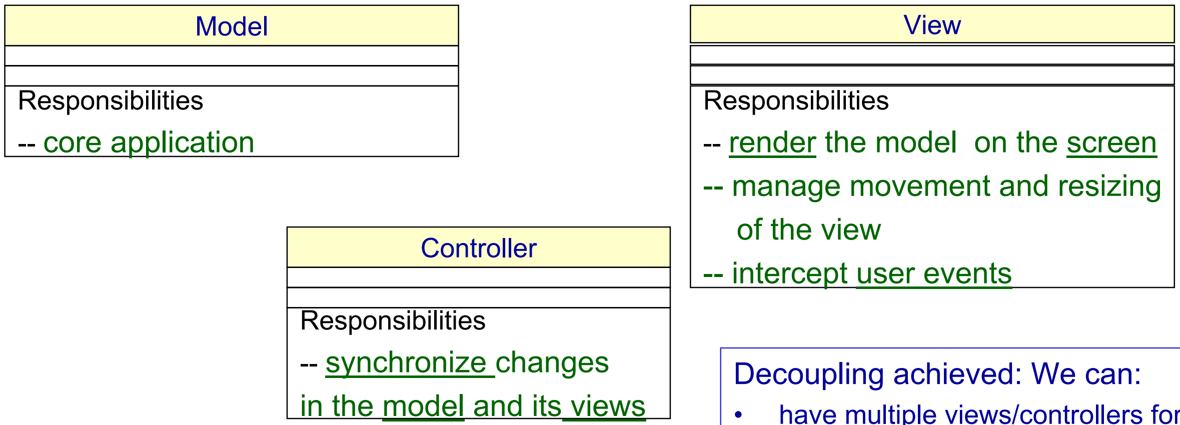


Model-View-Controller (MVC)

- This pattern is used in applications where the UI is very important
- Motivation
  - same data may be displayed differently
  - display and application must reflect data changes immediately
  - UI changes should be easy and even possible at runtime
  - Porting the UI to another platform should not affect core application code
- Solution
  - Divide application into 3 parts
    - Model
    - View
    - Controller

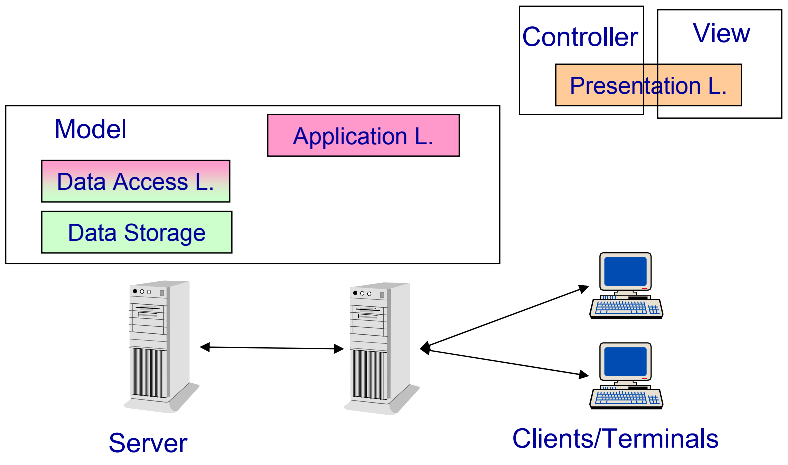
Model-View-Controller

Model: provides the essential functionality of the application (application logic)
View: supports a particular style of interaction with the user (display output)
Controller: accepts user input in the form of events and synchronizes changes between the model and its views (user input)

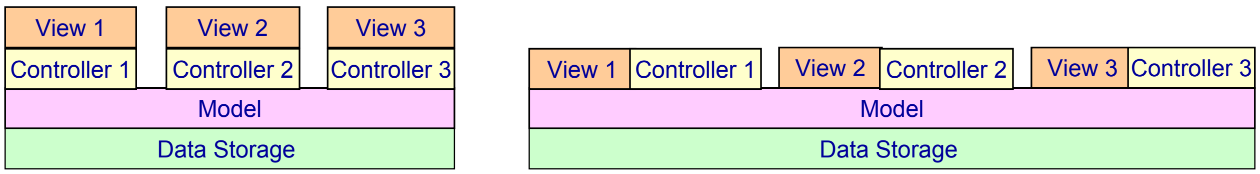

Model
Responsibilities
-- core application
View
Responsibilities
-- render the model on the screen
-- manage movement and resizing of the view
-- intercept user events
Controller
Responsibilities
-- synchronize changes in the model and its views

Decoupling achieved: We can:
- have multiple views/controllers for the same model
- reuse views/controllers for other models

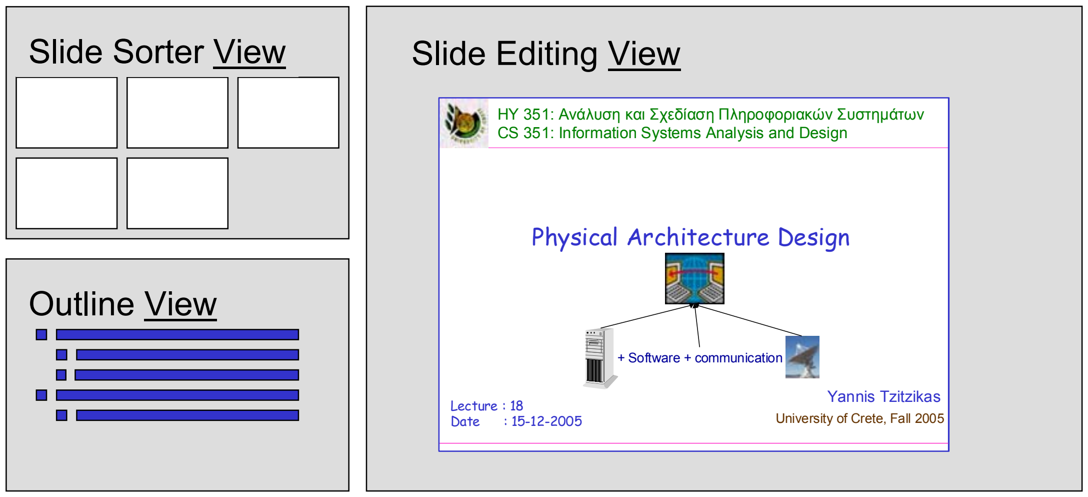
MVC: connection with the previous discussion



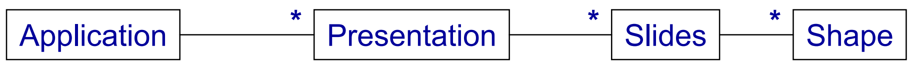
Keypoints
- One central model, many views (viewers)
- Each view has an associated controller
- The controller handles updates from the user of the view
- Changes to the model are propagated to all the views



Example: The Views of Powerpoint


Slide Sorter View
Outline View
Slide Editing View
HY 351: Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
CS 351: Information Systems Analysis and Design
Physical Architecture Design
+ Software + communication
Lecture : 18
Date : 15-12-2005
Yannis Tzitzikas
University of Crete, Fall 2005

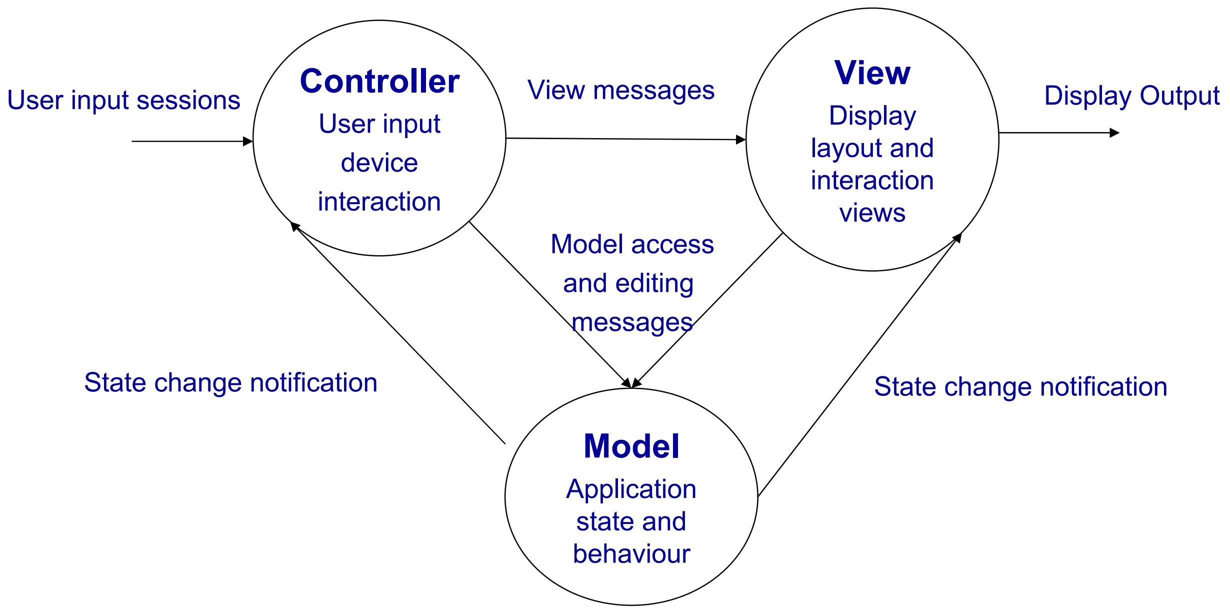
The structure of the model of Powerpoint



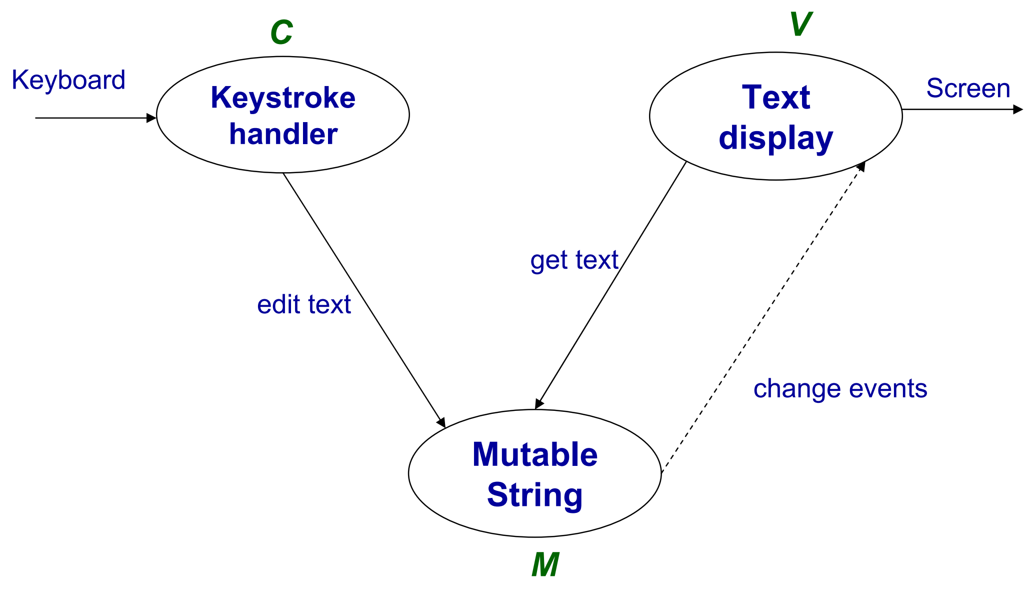
MVC

- Model
  - Core application code
    - maintains application state
  - Contains a list of observers (view or controller)
  - Has a broadcast mechanism to inform views of a change
- View
  - displays information to user
  - obtains data from model
  - each view has a controller
- Controller
  - handles input from user as events (keystrokes, mouse clicks and movements)
  - maps each event to proper action on model and/or view

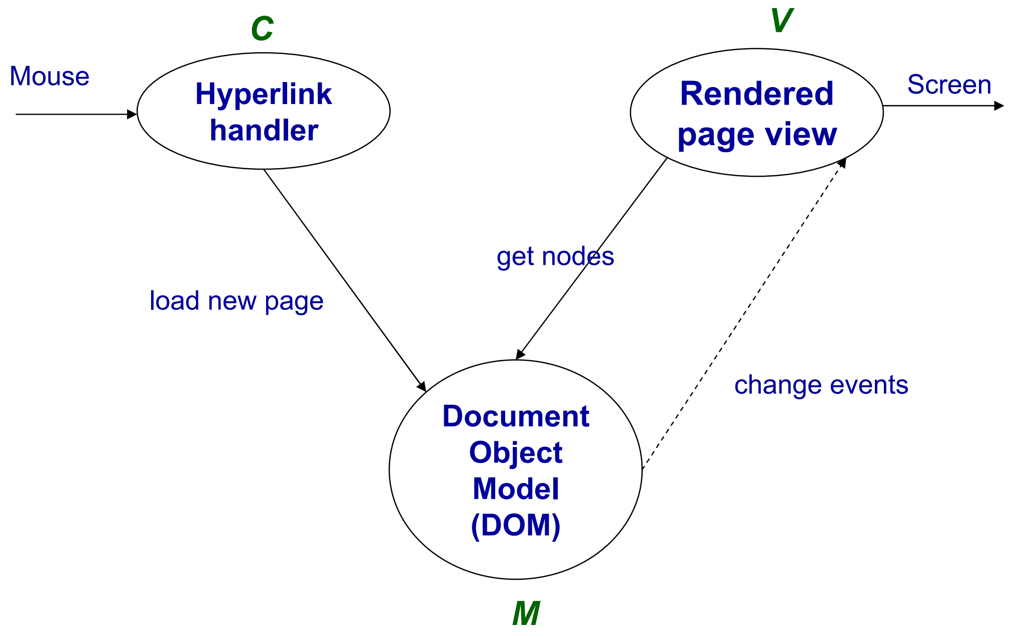
MVC


User input sessions
Controller
User input device interaction
View messages
View
Display layout and interaction views
Display Output
Model access and editing messages
State change notification
State change notification
Model
Application state and behaviour

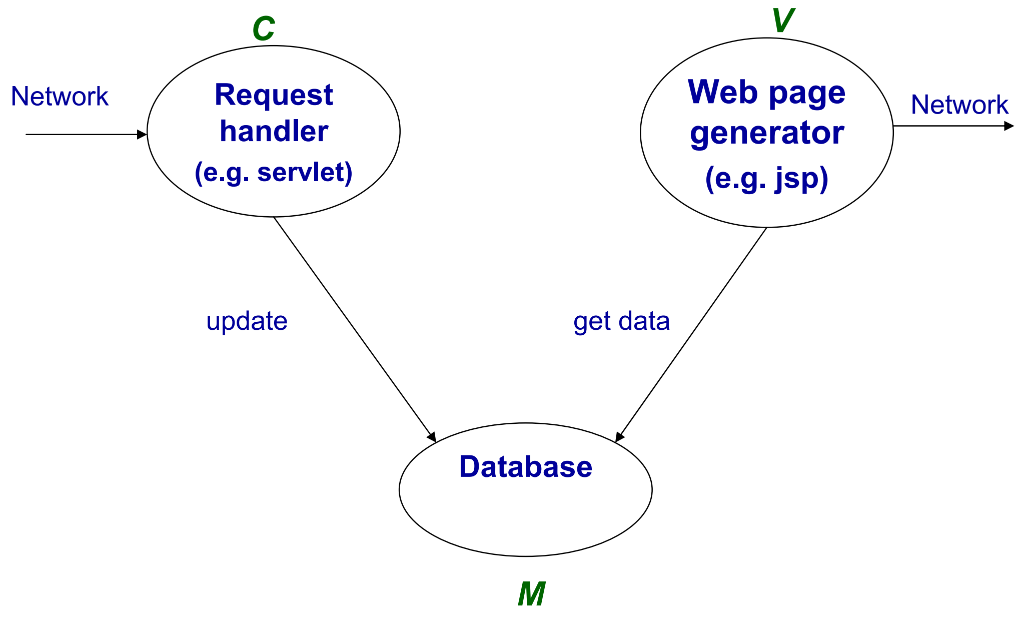
MVC Example: *Text Field*



MVC Example: *Web Browser*



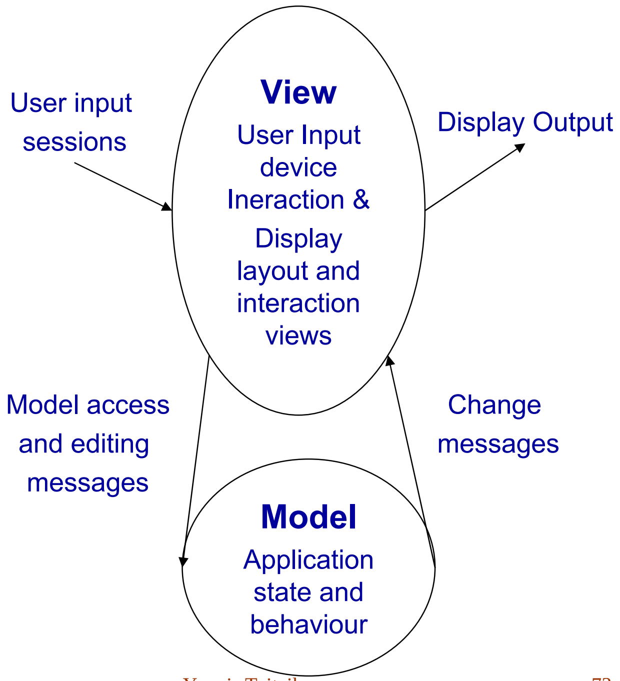
MVC Example: *Database-backed web server*



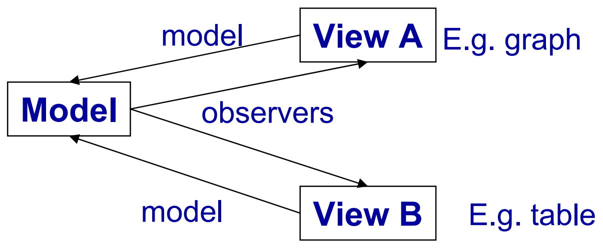
MVC and MV

In many cases, view and controller are very tightly coupled.

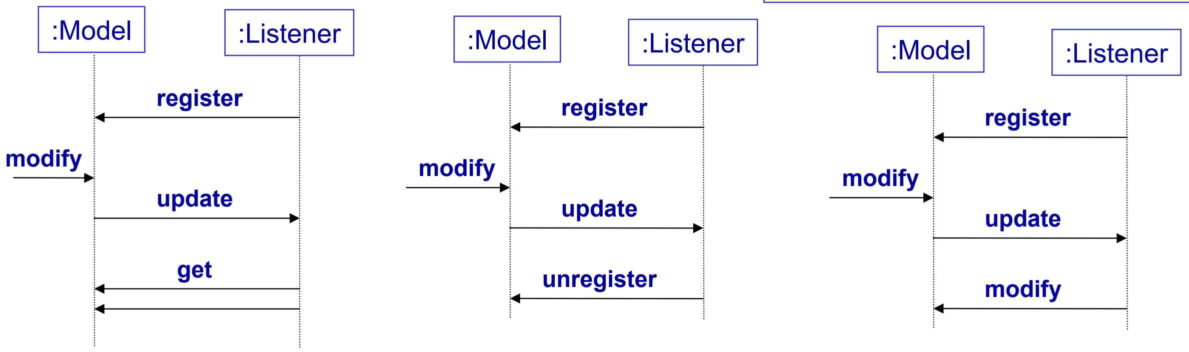
- so instead of MVC we have MV (Model-View)
- a reusable view manages both output and input
  - also called widgets, components, …
    - e.g. scrollbars, buttons, ...


User input sessions
View
User Input device Ineraction & Display layout and interaction views
Display Output
Model access and editing messages
Change messages
Model
Application state and behaviour

Observer pattern is used to decouple model from views



```
interface Model {
       void register(Observer)
       void unregister(Observer)
       Object get()
       void modify()
}
interface Observer {
       void update(Event)
}
```



- *How we can depict the Physical Architecture of a System?*
- *Is there any standard diagrammatic notation?*

=> UML Components and Deployment diagrams

- – (next lecture)