SOFTWARE ENGINEERING
(Week-4)

USAMA MUSHARAF

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

CONTENTS OF WEEK # 4

Architectural Styles

- Categories of Architectural Style
  - **Data Centered Software Architecture**
    - Black board
    - Shared Repository
  - **Component-Based Software Architecture**
  - **Distributed Software Architecture**
    - Client Server
    - Peer to Peer
    - REST

CATEGORIES OF ARCHITECTURAL STYLES

- **Hierarchical Software Architecture**
  - Layered
- **Data Flow Software Architecture**
  - Pipe and Filter
  - Batch Sequential
- **Data Centered Software Architecture**
  - Black board
  - Shared Repository
- **Component-Based Software Architecture**
- **Distributed Software Architecture**
  - Client Server
  - Peer to Peer
  - REST
  - SOA
  - Microservices
  - Cloud Architecture
- **Event Based Software Architecture**

DATA CENTERED / SHARED DATA SOFTWARE ARCHITECTURE

SHARED DATA SOFTWARE ARCHITECTURE

- Data-centered software architecture is characterized by a centralized data store that is shared by all surrounding software components.

- The software system is decomposed into two major partitions: data store and independent software component or agents.

SHARED DATA SOFTWARE ARCHITECTURE

- In pure data-centered software architecture, the software components don't communicate with each other directly; instead, all the communication is conducted via the data store.

- The shared data module provides all mechanisms for software components to access it, such as insertion, deletion, update, and retrieval.

SHARED DATA:

- Blackboard style
- Repository style

REPOSITORY ARCHITECTURE

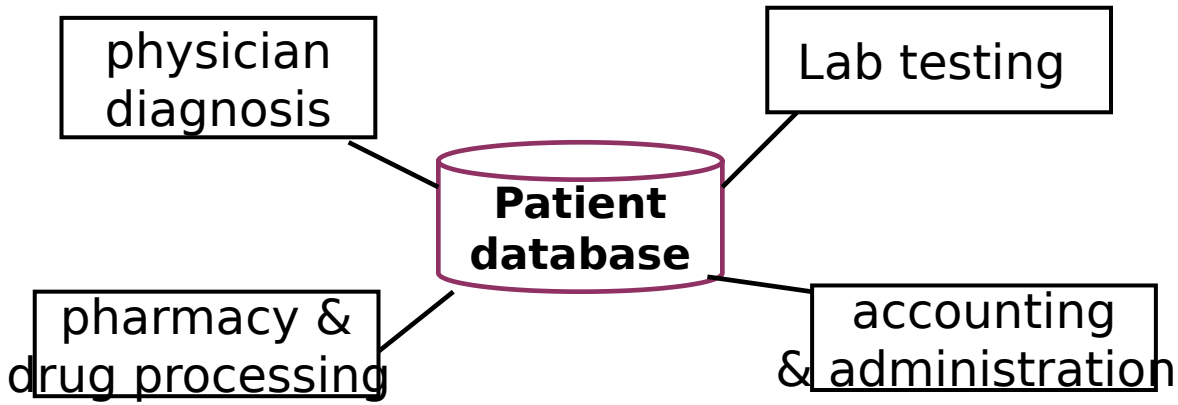
REPOSITORY ARCHITECTURE

- The repository architecture style is a data-centered architecture that supports user interaction for data processing.

- The software component agents of the data store control the computation and flow of logic of the system.

REPOSITORY ARCHITECTURE

- All data in a system is managed in a central repository that is accessible to all system components.
- Components do not interact directly, only through the repository.



REPOSITORY ARCHITECTURE

- Organizing tools around a repository is an efficient way to share large amounts of data.
  - There is no need to transmit data explicitly from one component to another.
- Although it is possible to distribute a logically centralized repository, there may be problems with data redundancy and inconsistency.
  - In practice, it may be difficult to distribute the repository over a number of machines.

EXAMPLE: INTEGRATED DEVELOPMENT ENVIRONMENT

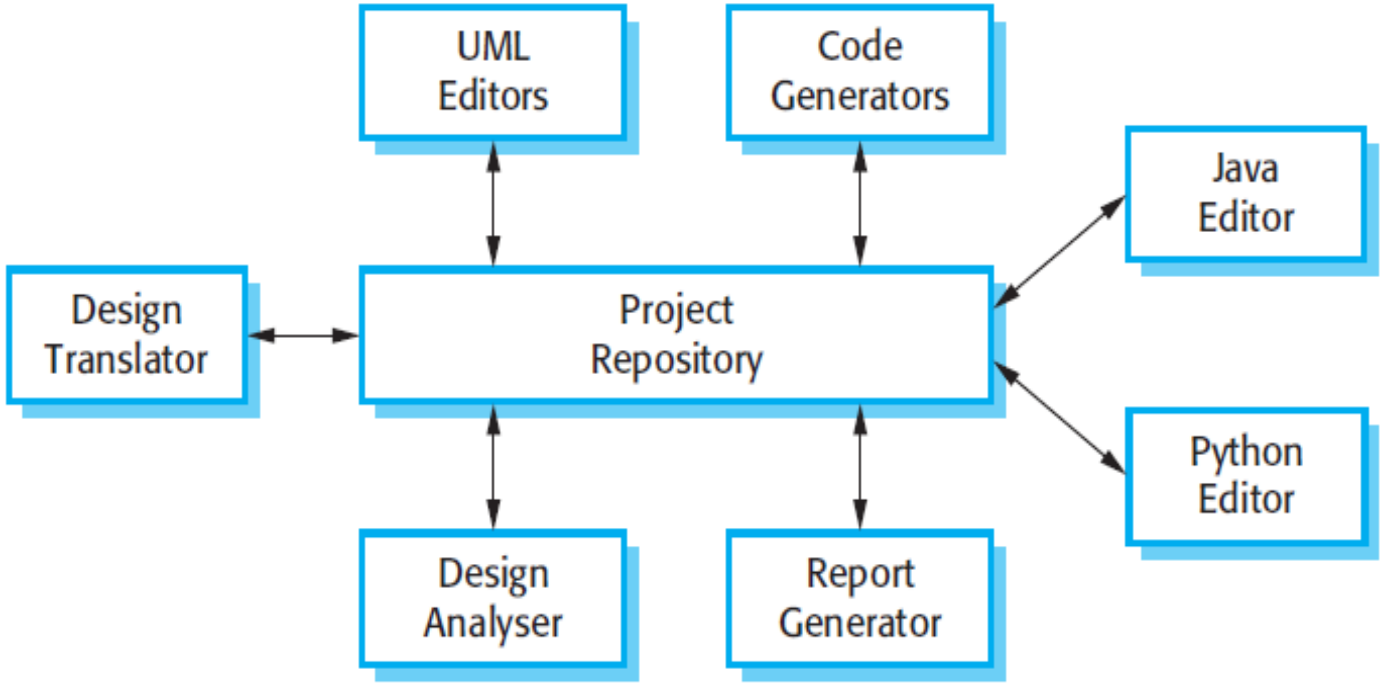


Figure 6.9 A repository architecture for an IDE

APPLICABLE DOMAINS OF REPOSITORY ARCHITECTURE:

- Suitable for large, complex information systems where many software component clients need to access them in different ways
- Requires data transactions to drive the control flow of computation

ADVANTAGES

- Components can be independent—they do not need to know of the existence of other components.
- Changes made by one component can be propagated to all components.
  - All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.

DISADVANTAGES

- The repository is a single point of failure so problems in the repository affect the whole system.
- May be inefficiencies in organizing all communication through the repository.
- Distributing the repository across several computers may be difficult.

BLACKBOARD ARCHITECTURE

BLACKBOARD ARCHITECTURE

- The blackboard architecture was developed for speech recognition applications in the 1970s.
- Other applications for this architecture are image pattern recognition and weather broadcast systems.

BLACKBOARD ARCHITECTURE

- The word blackboard comes from classroom teaching and learning.
- Teachers and students can share data in solving classroom problems via a blackboard.
- Students and teachers play the role of agents to contribute to the problem solving.
- They can all work in parallel, and independently, trying to find the best solution.

BLACKBOARD ARCHITECTURE

The entire system is decomposed into two major partitions.

- One partition, called the blackboard, is used to store data.
- while the other partition, called knowledge sources, stores domain specific knowledge.
- There also may be a third partition, called the controller, that is used to initiate the blackboard and knowledge sources and that takes a main role and overall supervision control.

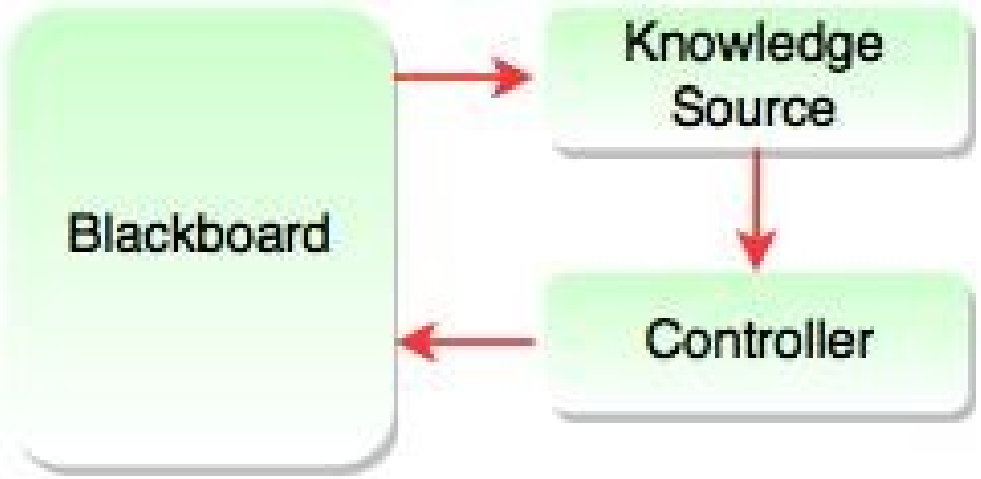
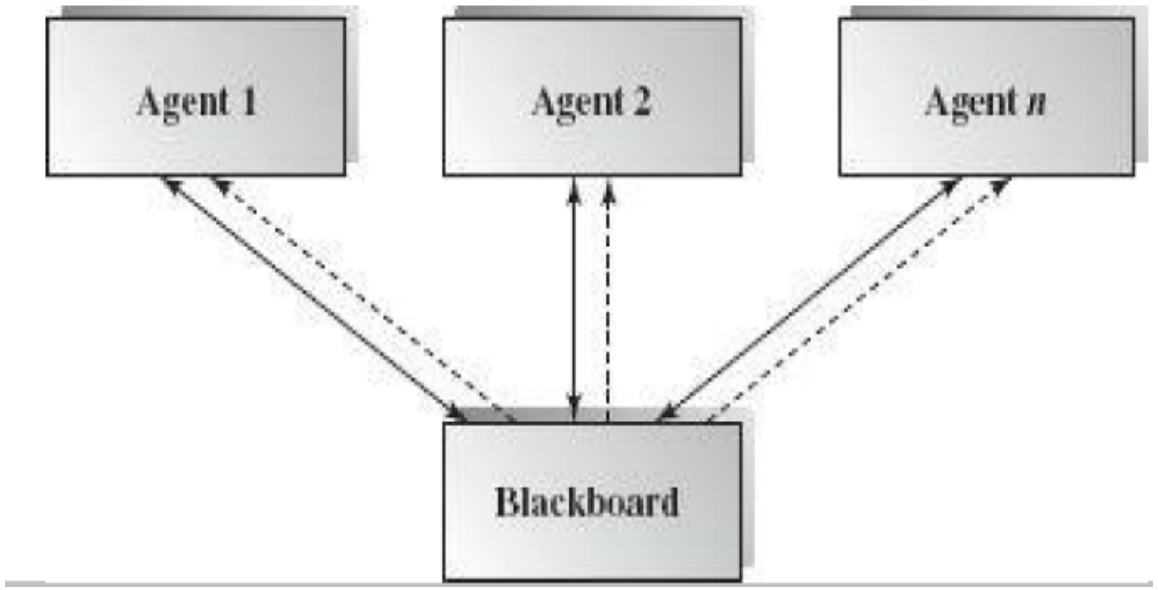


Fig. Blackboard Architectural Style

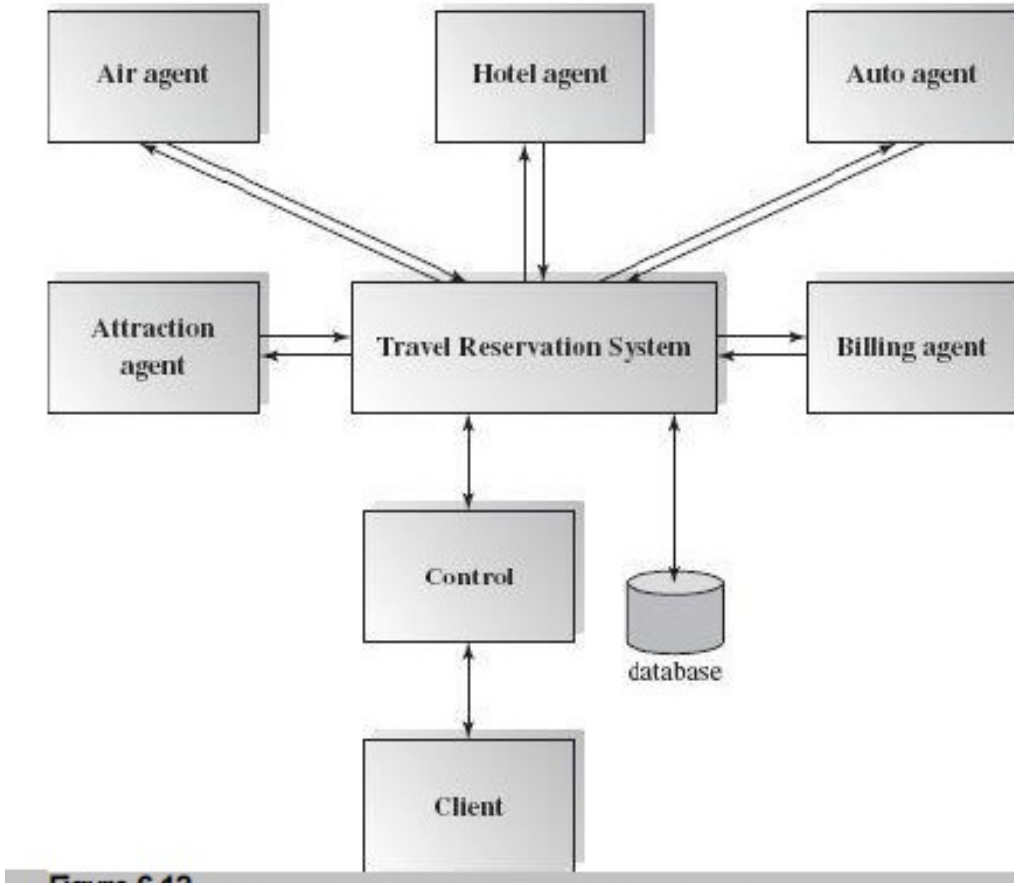
BLACKBOARD ARCHITECTURE - CONNECTIONS

- Data changes in the blackboard trigger one or more matched knowledge source to continue processing.
- This connection can be implemented in publish/subscribe mode.



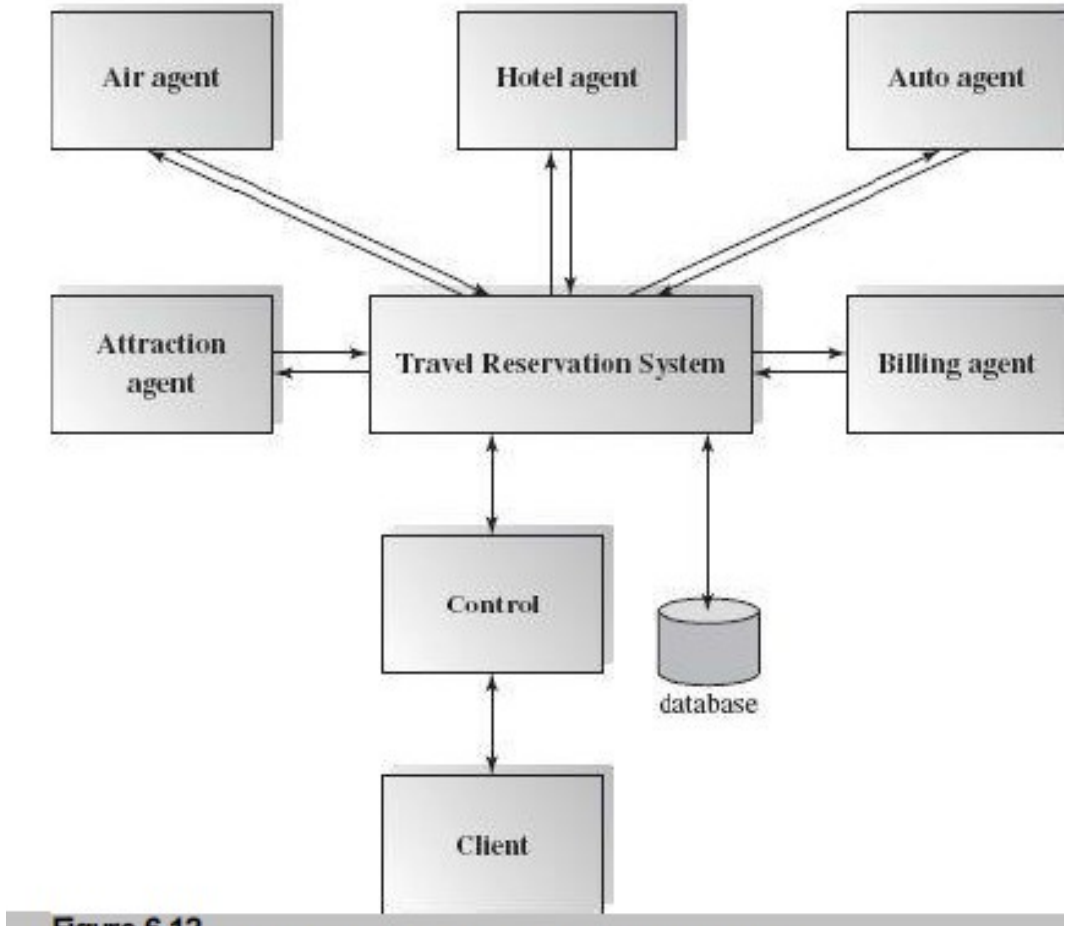
EXAMPLE - TRAVEL CONSULTING SYSTEM

There may be many air travel agencies, hotel reservation systems, car rental companies, or attraction reservation systems to subscribe to or register with through this travel planning system.



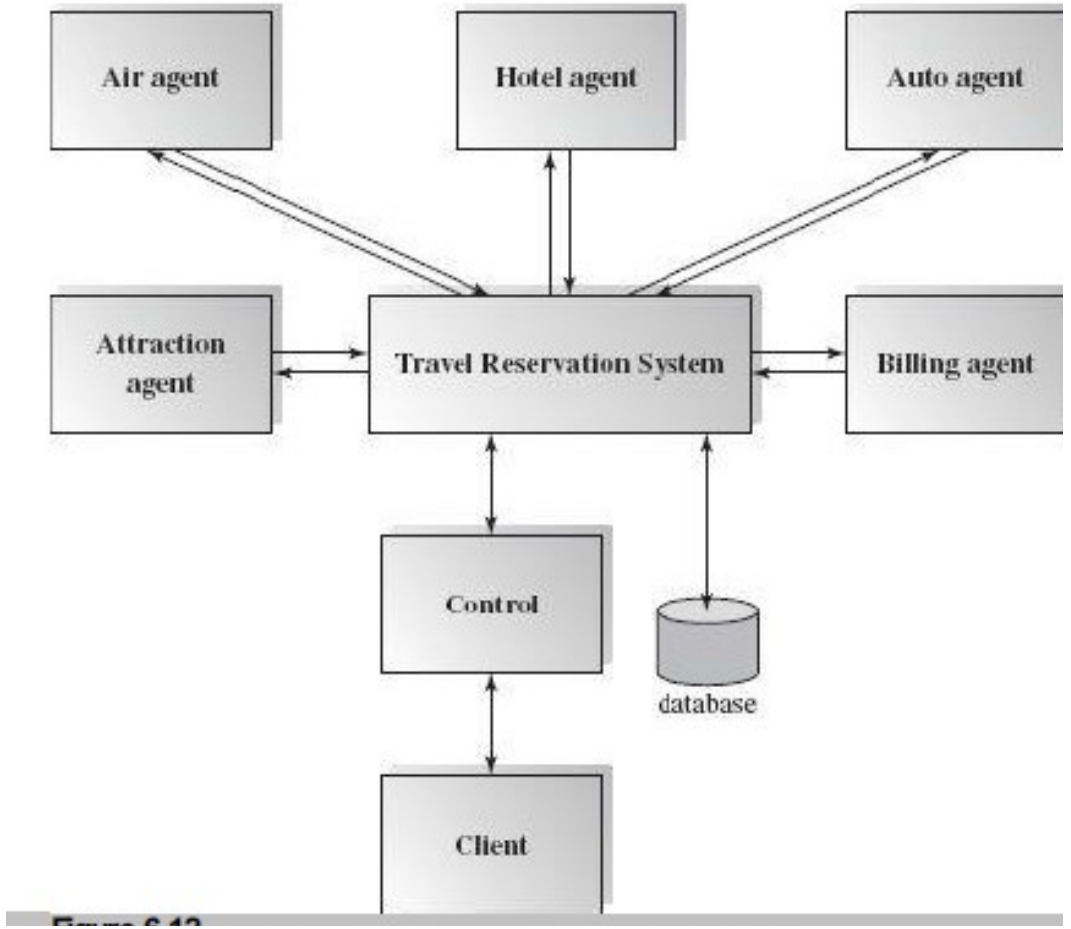
EXAMPLE - TRAVEL CONSULTING SYSTEM

- Once the system receives a client request, it publishes the request to all related agents and composes plan options for clients to choose from.



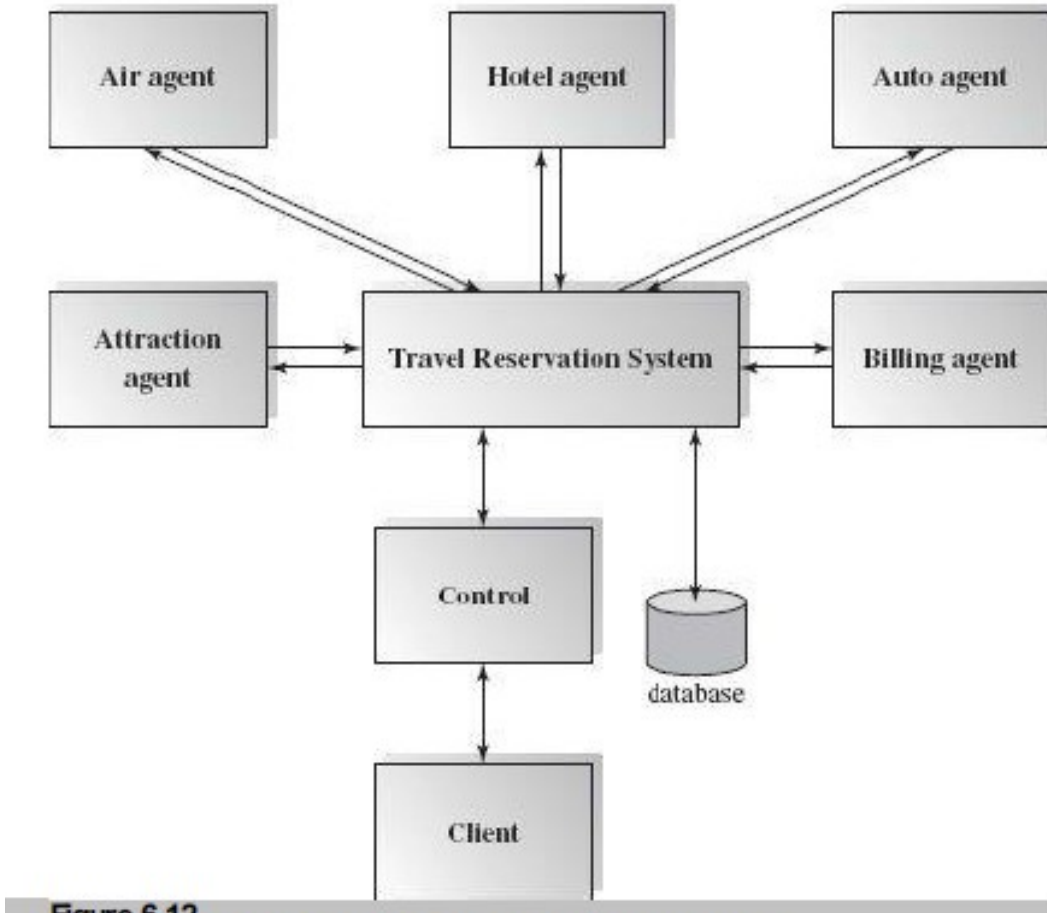
EXAMPLE - TRAVEL CONSULTING SYSTEM

- The system also stores all necessary data in the database.
- After the system receives a confirmation from the client, it invokes the financial billing system to verify credit background and to issue invoices.



EXAMPLE - TRAVEL CONSULTING SYSTEM

- The data in the data store plays an active role in this system.
- It does not require much user interaction after the system receives client requests since the request data will direct the computation and activate all related knowledge sources to solve the problem.



APPLICABLE DOMAIN:

- Suitable for solving complex problems such as artificial intelligence (AI) problems where no preset solutions exist.

BENEFITS:

- Scalability: easy to add or update knowledge source.
- Concurrency: all knowledge sources can work in parallel since they are independent of each other.
- Reusability of knowledge source agents.

COMPONENT BASED SOFTWARE ARCHITECTURE

COMPONENT-BASED SOFTWARE ENGINEERING

- CBSE is an approach to software development that relies on reuse
- CBSE emerged from the failure of object-oriented development to support reuse effectively
- Objects (classes) are too specific and too detailed to support design for reuse work

COMPONENT

- A software component is an independently deployable implementation of some functionality, to be reused as it is in a broad spectrum of applications.

- For a software component what it provides and requires should be clearly stated so that it can be used without any ambiguity.

COMPONENT BASED SOFTWARE ARCHITECTURE

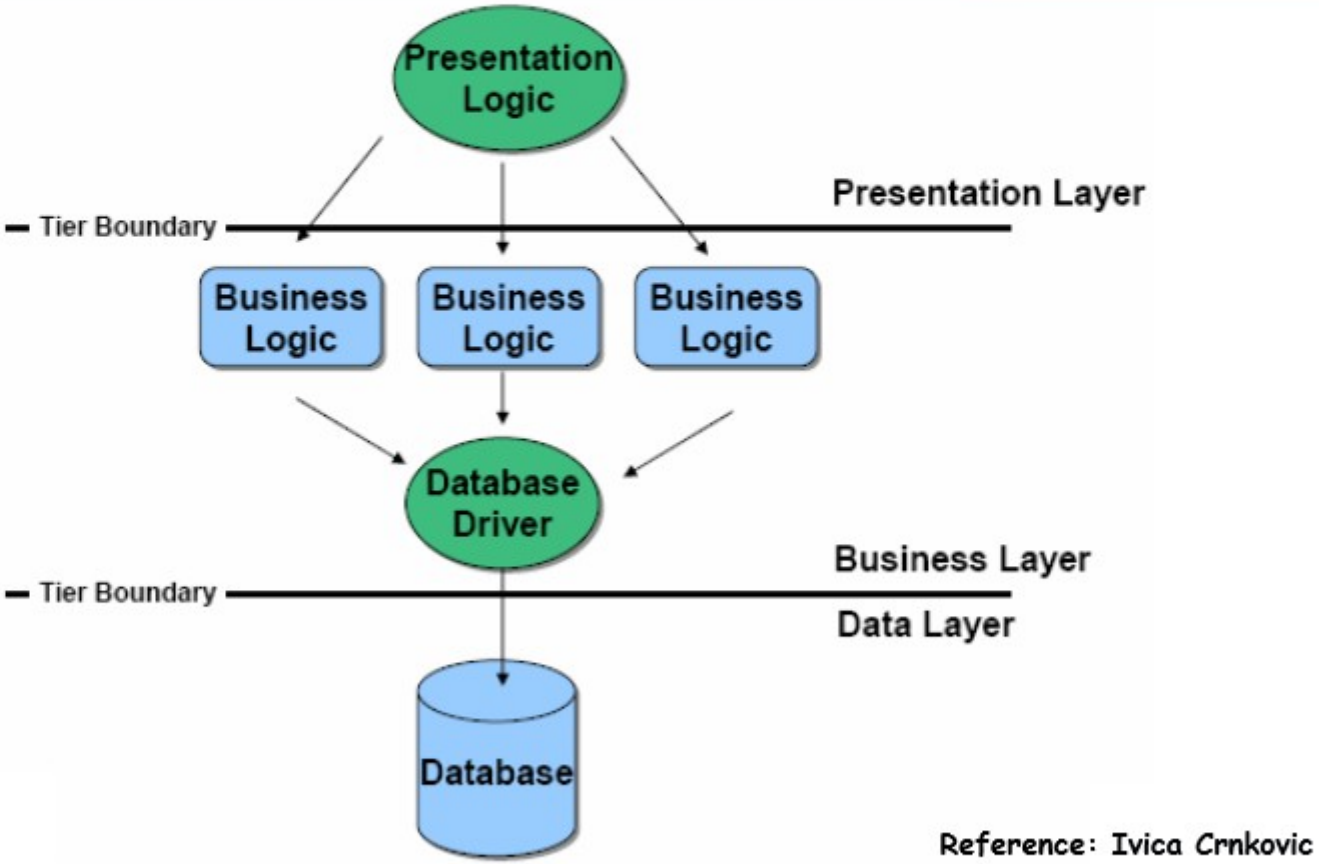
- The main motivation behind component-based design is component reusability.

- Designs can make use of existing reusable commercial off-the-shelf (COTS) components or ones developed in-house, and they may produce reusable components for future reuse.

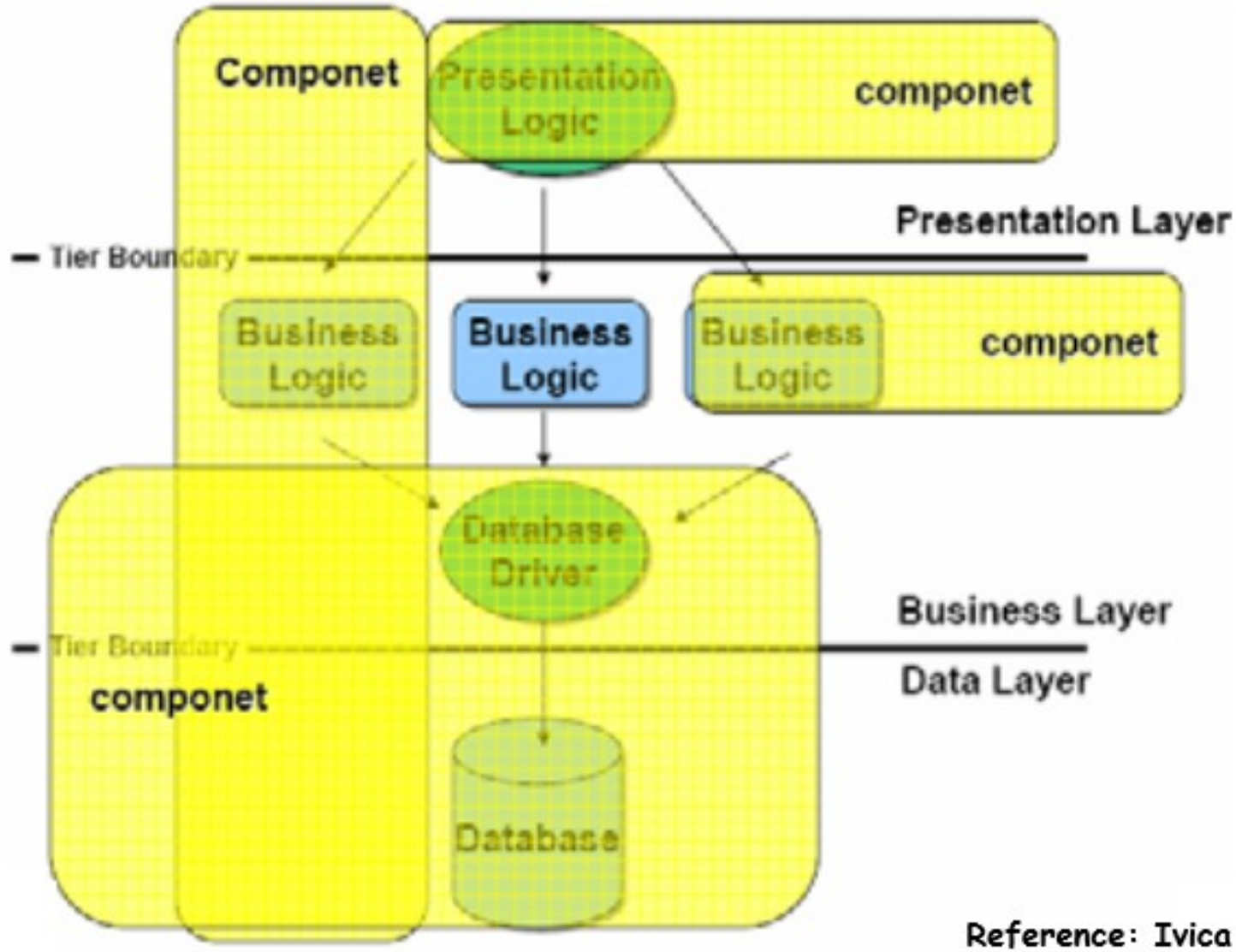
CATEGORIES OF COMPONENTS

- Commercial of-the-shelf components- complete application libraries readily available in the market.
- Qualified components—assessed by software engineers to ensure that not only functionality, but also performance, reliability, usability, and other quality factors conform to the requirements of the system/product to be built.
- Adapted components—adapted to modify (wrapping) unwanted or undesired characteristics.

N TIER ARCHITECTURE



Reference: Ivica Crnkovic



Reference: Ivica Crnkovic

CONNECTORS

Connectors connect components, specifying and ruling their interaction.

Component interaction can take the form of

- method invocations,
- asynchronous invocations such as
  - event listener and registrations,
  - broadcasting,

Requires interface
Defines the services that the component uses from the environment

Provides interface
Defines the services that are provided by the component to other components

COMPONENT DIAGRAM

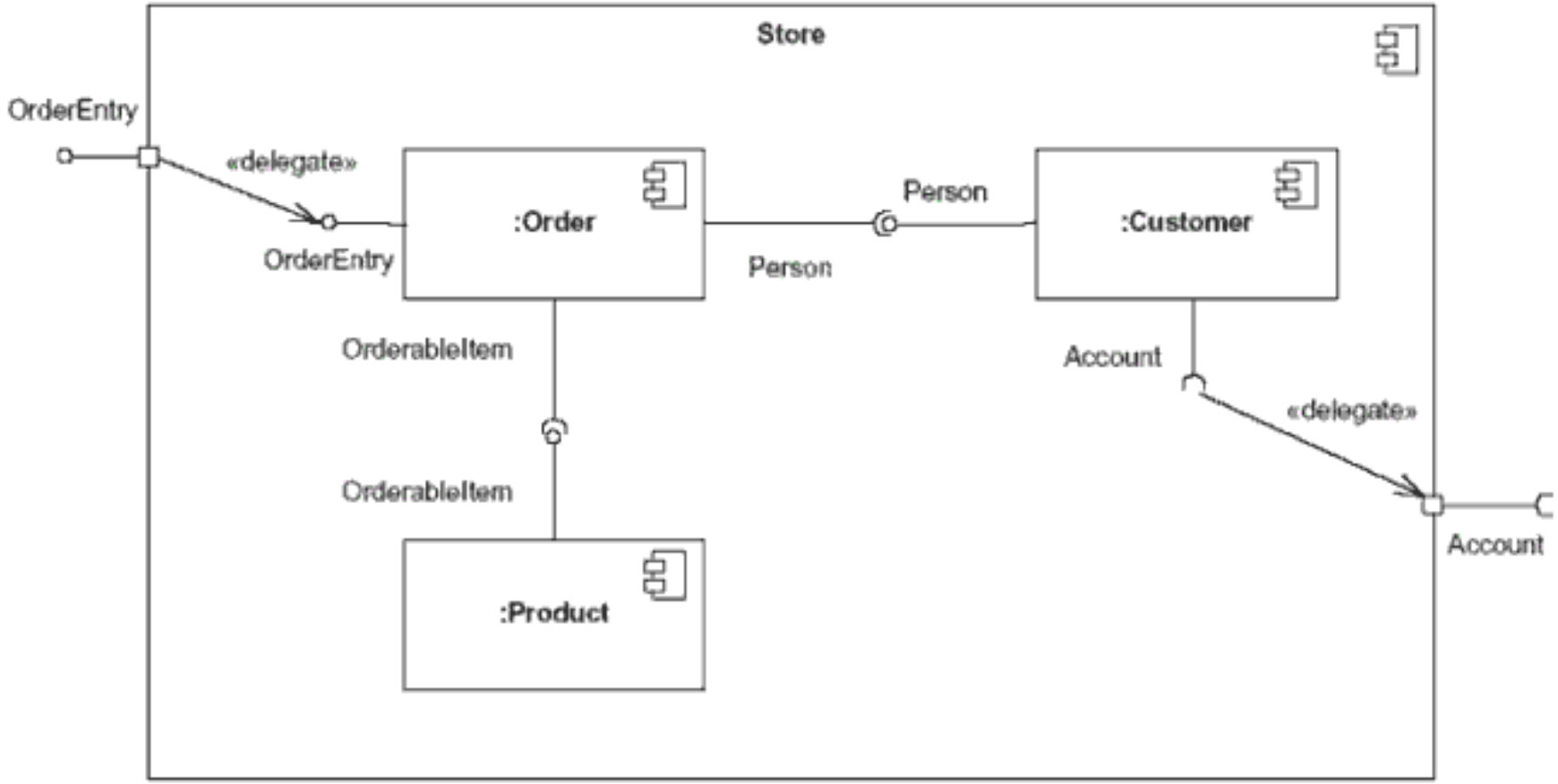
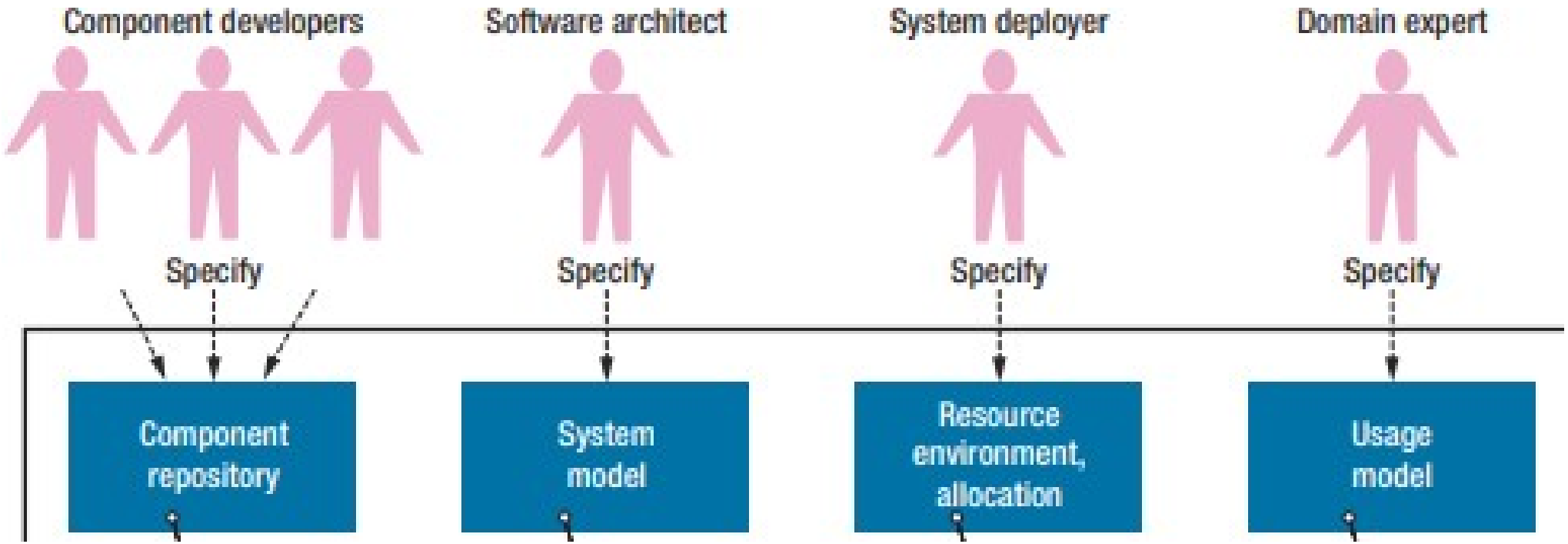
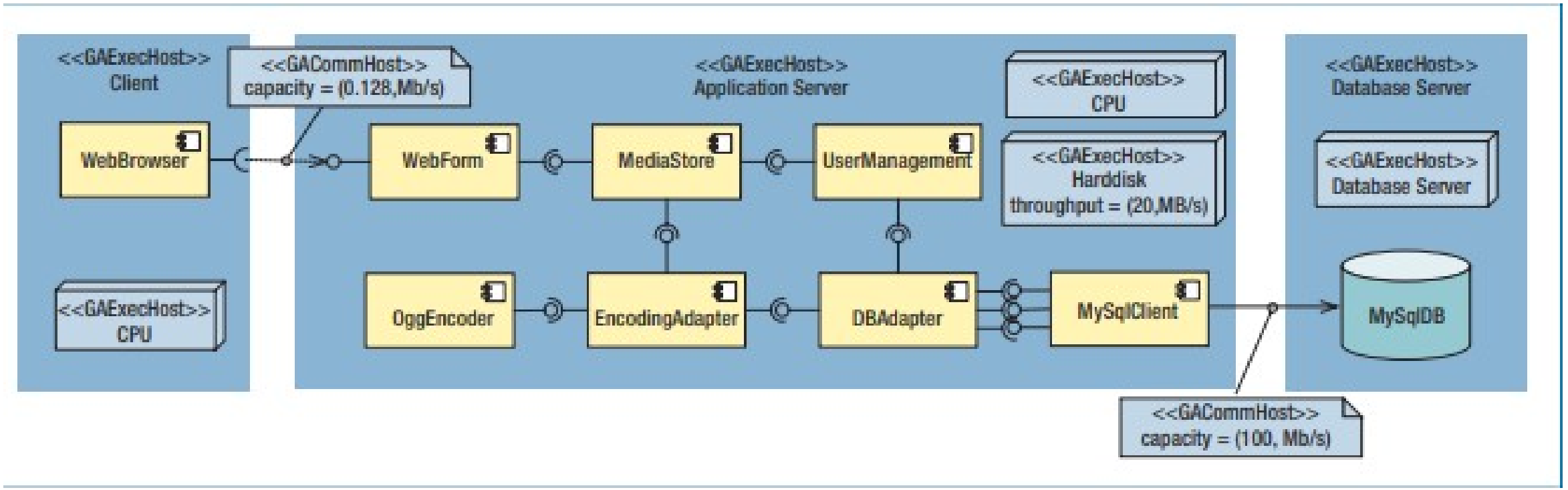


Figure: This component's inner structure is composed of other components


Component developers
Software architect
System deployer
Domain expert
Specify
Specify
Specify
Specify
Component repository
System model
Resource environment, allocation
Usage model


<<GAExecHost>>
Client
<<GACommHost>>
capacity = (0.128,Mb/s)
<<GAExecHost>>
Application Server
<<GAExecHost>>
CPU
<<GAExecHost>>
Database Server
WebBrowser
WebForm
MediaStore
UserManagement
<<GAExecHost>>
Harddisk
throughput = (20,MB/s)
<<GAExecHost>>
Database Server
<<GAExecHost>>
CPU
OggEncoder
EncodingAdapter
DBAdapter
MySqlClient
MySqlDB
<<GACommHost>>
capacity = (100, Mb/s)

COMPONENT BASED DEVELOPMENT

COMPONENT BASED DEVELOPMENT

This process basically consists of three main stages namely

- qualification,
- adaptation and
- composition

COMPONENT QUALIFICATION

Component qualification ensures that a candidate component

- will perform the function required,
- will properly fit into the architectural style specified for the system, and
- will exhibit the quality characteristics (e.g., performance, reliability, usability) required for the application.

COMPONENT ADAPTION

- Most of the times even after the component has been qualified for use in the architecture it exhibits some conflicts.
- To soothe these conflicts certain techniques such as component wrapping is used.

WRAPPING

- White-Box wrapping – this wrapping involves code level modifications in the components to avoid conflicts.

  This is not widely used because the COTS products are not provided with the source code.

WRAPPING

- Grey-Box wrapping- applied when the component library provides a component extension language or API that enables conflicts to be removed.

- Black-box wrapping - introduction of pre- and post-processing at the component interface to remove or mask conflicts.

COMPONENT COMPOSITION

- Architectural style depends the connection between various components and their relationships.

- The design of the software system should be In such a way that most of the components are replaceable or can be reused in other systems.

BENEFITS

- Ideally the components for reuse would be verified and defect-free.
- As the component is reused in many software systems any defect if there would be detected and corrected. So after a couple of reuses the component will have no defects.

BENEFITS

- Productivity is increased as every time the code need not be re-written from the scratch.
- The time taken to design and write the code also decreases with the use of software components.

LIMITATIONS

- It can be difficult to find suitable available components to reuse.
- Adaptation of components is an issue.

DISTRIBUTED SOFTWARE ARCHITECTURE

DISTRIBUTED SOFTWARE ARCHITECTURE

- A distributed system is a collection of computational and storage devices connected through a communications network.
- Data, software, and users are distributed.

CLIENT SERVER ARCHITECTURAL STYLE

CLIENT SERVER ARCHITECTURAL STYLE

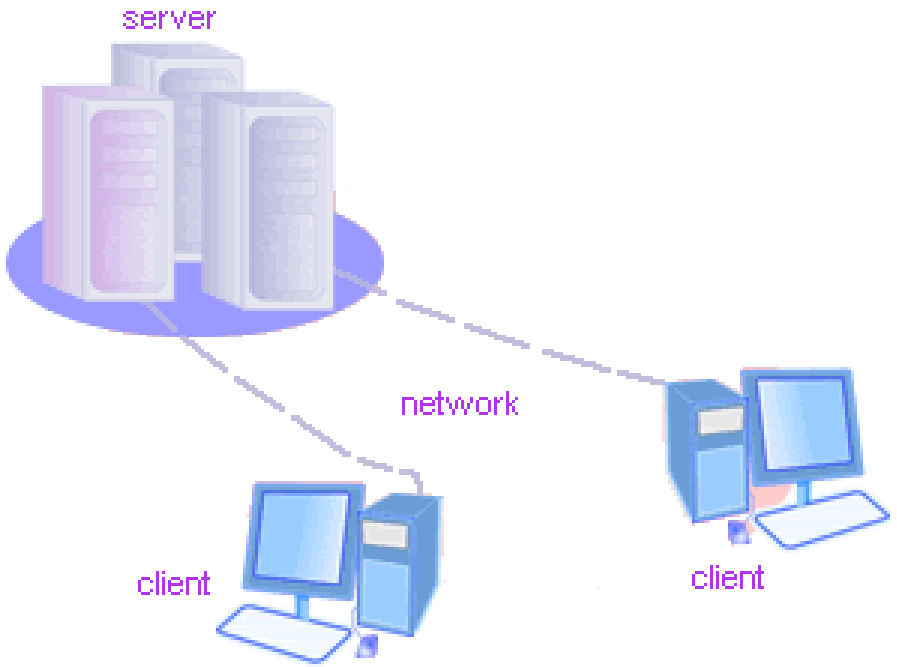
- Client/server architecture illustrates the relationship between two computer programs in which one program is a client, and the other is Server.

- **Client** makes a service request to server.
- **Server** provides service to the request.

CLIENT/SERVER

- Although the client/server architecture can be used within a single computer by programs, but it is a more important idea in a network.

- In a network, the client/server architecture allows efficient way to interconnect programs that are distributed efficiently across different locations.



CLIENT-SERVER STYLE

- Suitable for applications that involve distributed data and processing across a range of components.

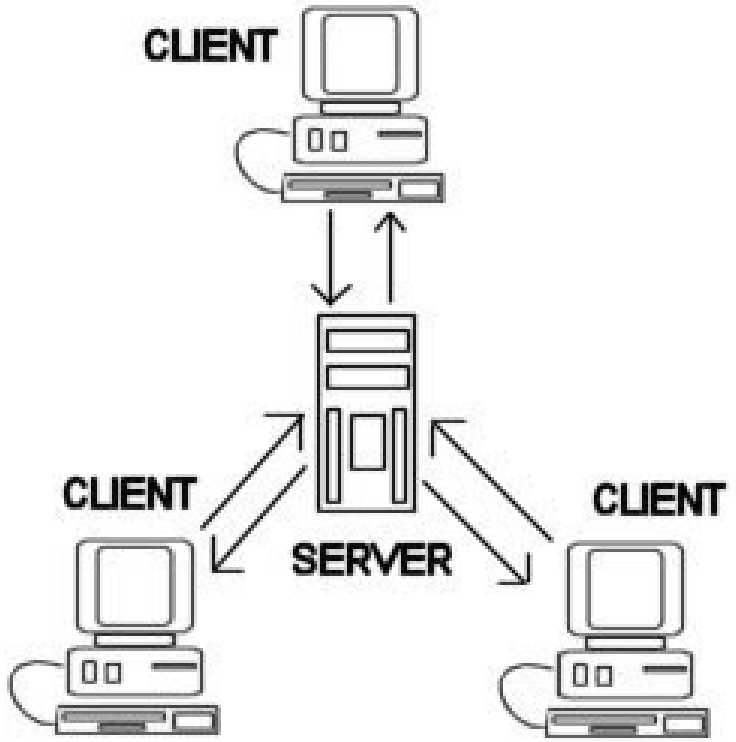
Components:

- **Servers:** Stand-alone components that provide specific services such as printing, data management, etc.
- **Clients:** Components that call on the services provided by servers.
- **Connector:** The network, which allows clients to access remote servers.

COMMON EXAMPLE

- The World Wide Web is an example of client-server architecture.

- Each computer that uses a Web browser is a client, and the data on the various Web pages that those clients access is stored on multiple servers.



ANOTHER EXAMPLE

- If you have to check a bank account from your computer, you have to send a request to a server program at the bank.
- That program processes the request and forwards the request to its own client program that sends a request to a database server at another bank computer to retrieve client balance information.
- The balance is sent back to the bank data client, which in turn serves it back to your personal computer, which displays the information of balance on your computer.

TYPES OF SERVERS

- **File Servers:**
  - Useful for sharing files across a network.
  - The client passes requests for files over the network to the file server.

- **Database Servers:**
  - Client passes SQL requests as messages to the DB server; results are returned over the network to the client.
  - Query processing done by the server.
  - No need for large data transfers.

MULTI-TIER CLIENT SERVER ARCHITECTURE

TYPES OF CLIENT SERVER

Two-tier client–server architecture,

- which is used for simple client–server systems, and in situations where it is important to centralize the system for security reasons.
- In such cases, communication between the client and server is normally encrypted.

- Multitier client–server architecture,
  - which is used when there is a high volume of transactions to be processed by the server.

A TWO-TIER CLIENT–SERVER ARCHITECTURE

The system is implemented as a single logical server plus an indefinite number of clients that use that server.

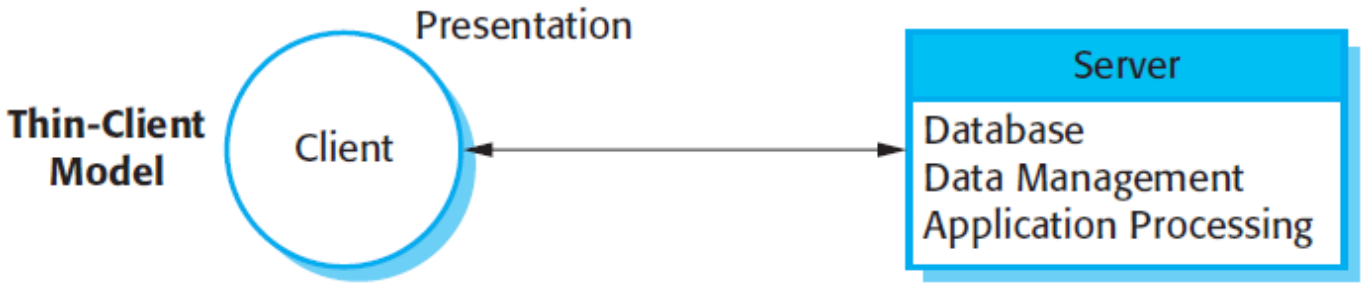
Two forms of this architectural model:

- A thin-client model,
- A fat-client model,

TWO-TIER CLIENT SERVER

A thin-client model,

- where the presentation layer is implemented on the client and all other layers (data management, application processing, and database) are implemented on a server.



ADVANTAGES

The advantage of the thin-client model is that it is simple to manage the clients.

- This is a major issue if there are a large number of clients, as it may be difficult and expensive to install new software on all of them. If a web browser is used as the client, there is no need to install any software.

DISADVANTAGES

The disadvantage of the thin-client approach, however is that it may place a heavy processing load on both the server and the network.

- The server is responsible for all computation and this may lead to the generation of significant network traffic between the client and the server.

TWO-TIER CLIENT SERVER

A fat-client model,

- where some or all of the application processing is carried out on the client.
- Data management and database functions are implemented on the server.

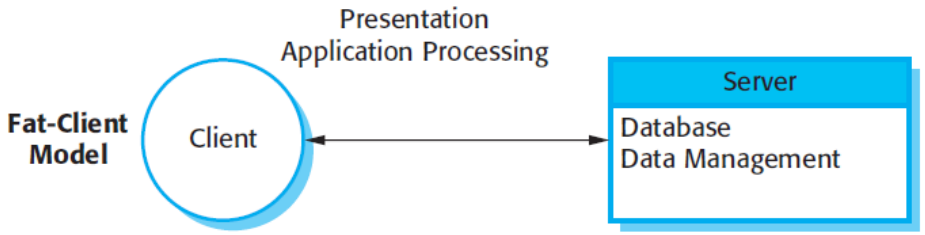
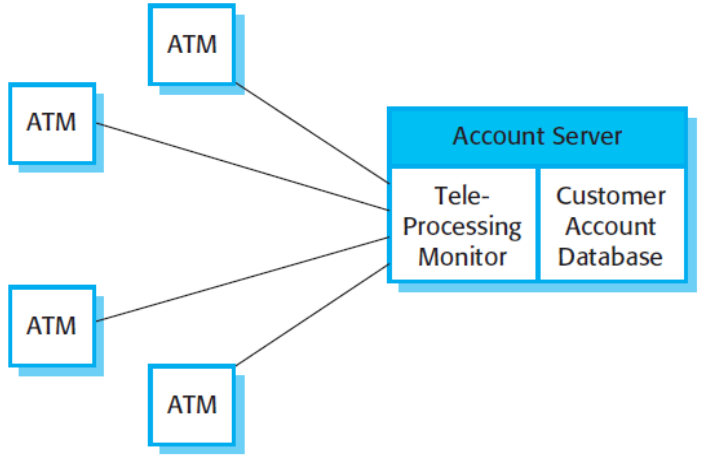


Figure 18.9 A fat-client architecture for an ATM system



MULTI-TIER CLIENT–SERVER ARCHITECTURES

- The fundamental problem with a two-tier client–server approach is that the logical layers in the system—presentation, application processing, data management, and database—must be mapped onto two computer systems: the client and the server.

- This may lead to problems with scalability and performance if the thin-client model is chosen, or problems of system management if the fat-client model is used.

MULTI-TIER CLIENT–SERVER ARCHITECTURES

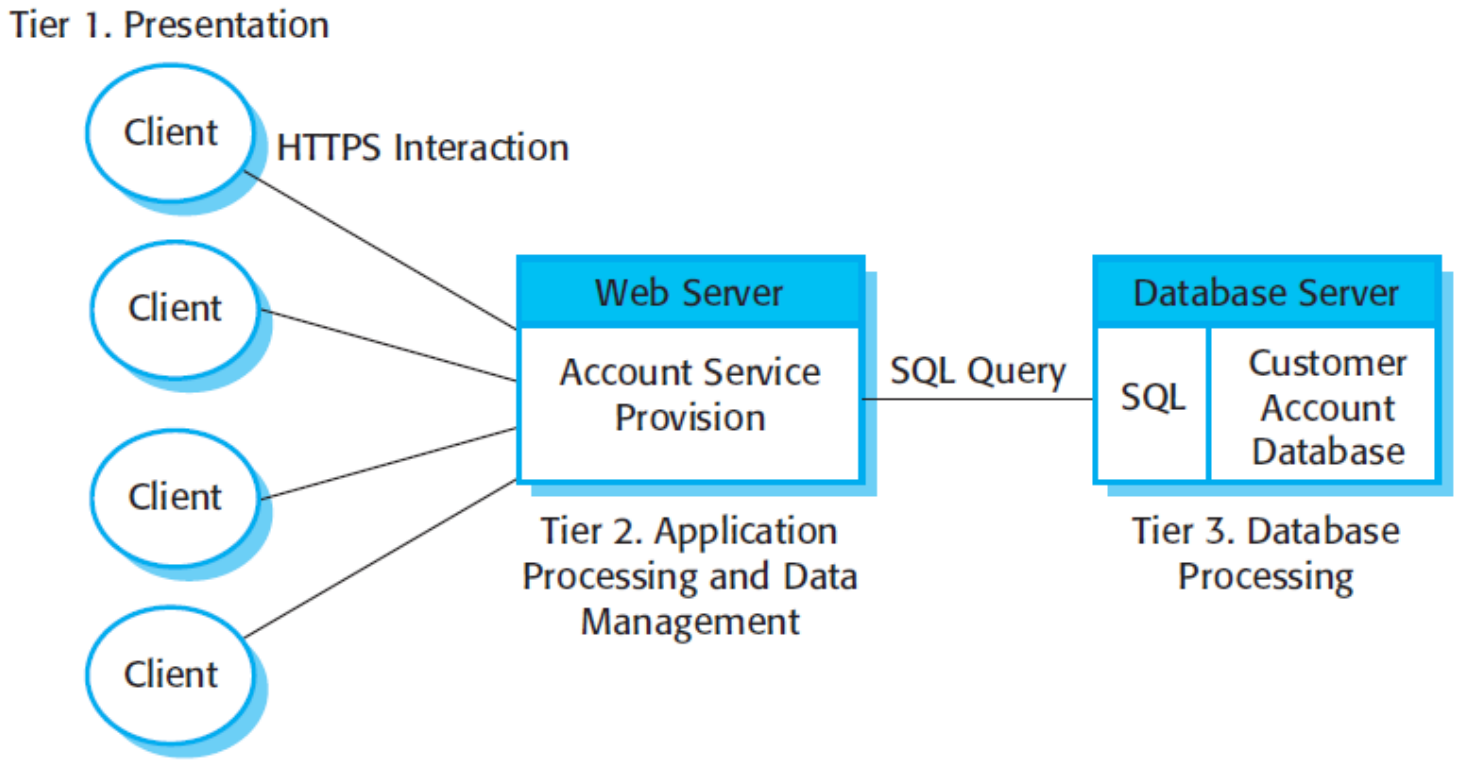


Figure 18.10 Three-tier architecture for an Internet banking system

MULTI-TIER CLIENT–SERVER ARCHITECTURES

- This system is scalable because it is relatively easy to add servers (scale out) as the number of customers increase.
- In this case, the use of a three-tier architecture allows the information transfer between the web server and the database server to be optimized.

REST ARCHITECTURE

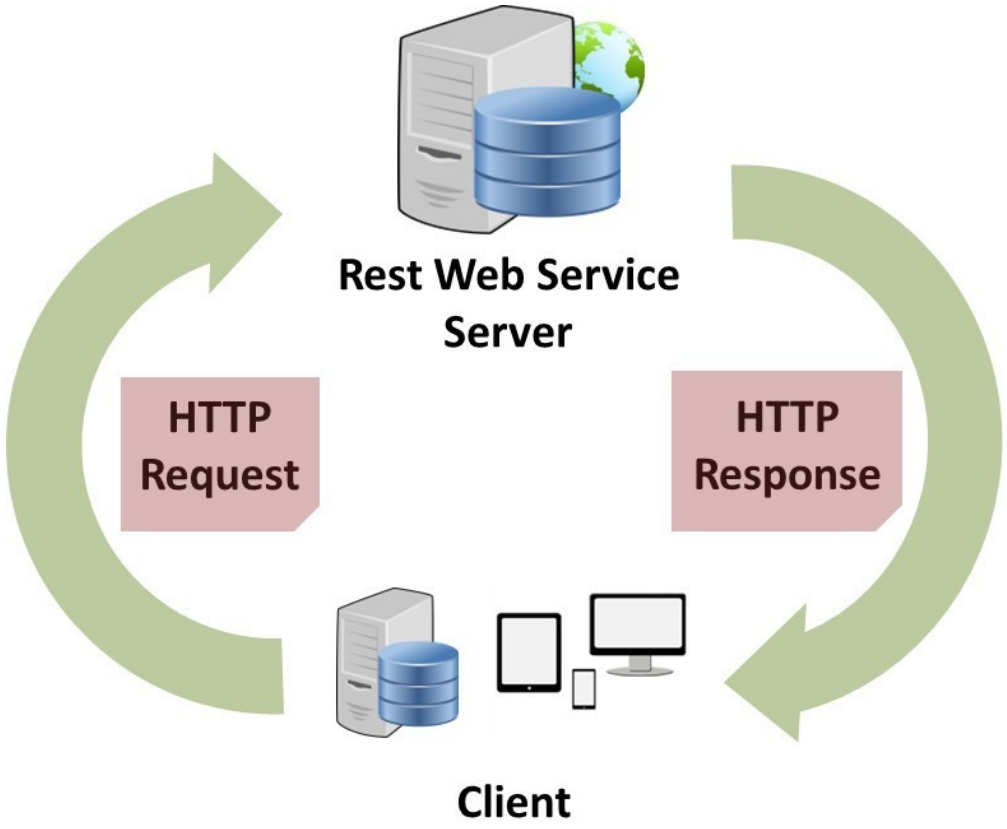
REPRESENTATIONAL STATE TRANSFER (REST)

- REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other.

- REST is a guideline for building performant and scalable applications.

REPRESENTATIONAL STATE TRANSFER (REST)

- Representational State Transfer (REST)
  - A style of software architecture for distributed systems such as the World Wide Web.
- REST is basically client/server architectural style
  - Requests and responses are built around the transfer of "representations" of "resources".
- HTTP is the main and the best example of a REST style implementation
  - But it should not be confused with REST
  - REST is not a protocol
  - REST is a guideline

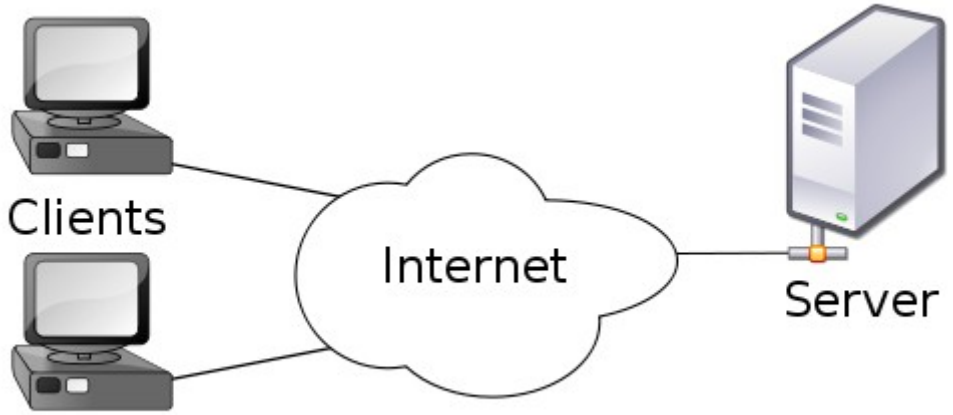


REST PRINCIPLES / ARCHITECTURAL CONSTRAINTS

- Client-server
- Stateless
- Cacheable
- Uniform interface
- Layered system
- Code on demand (optional)

1. CLIENT SERVER

- Separation of concerns is the principle behind the client-server constraints.
- By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- Client and server can evolve independently.

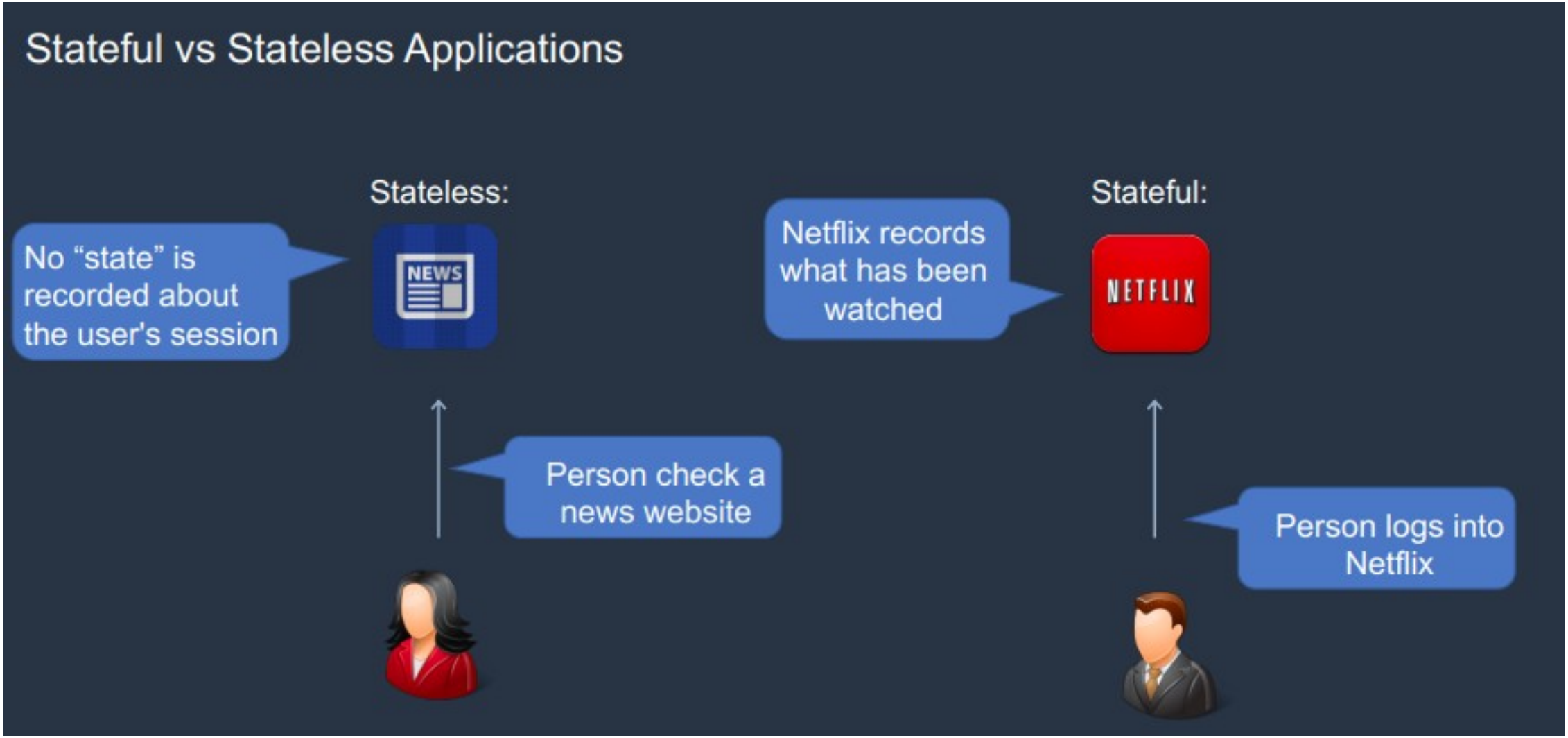


Client Server Model

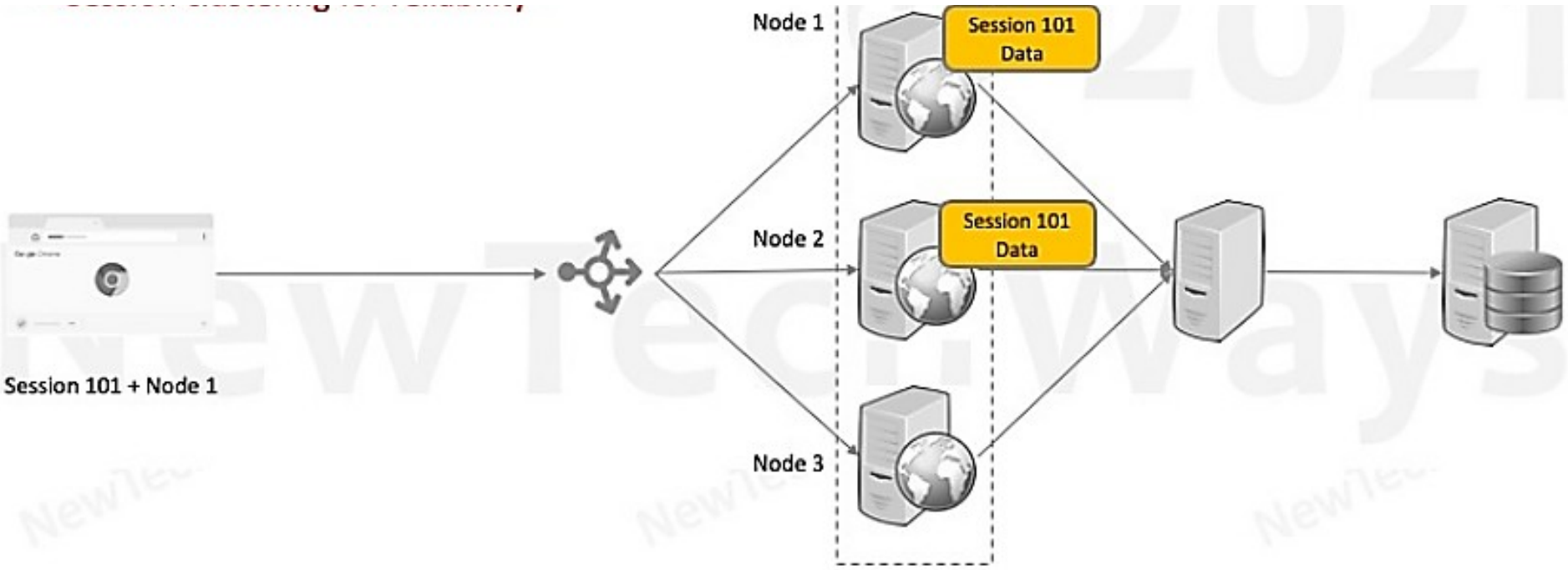
2. STATELESS

- Statelessness means communication must be stateless in nature as in the client stateless server style,

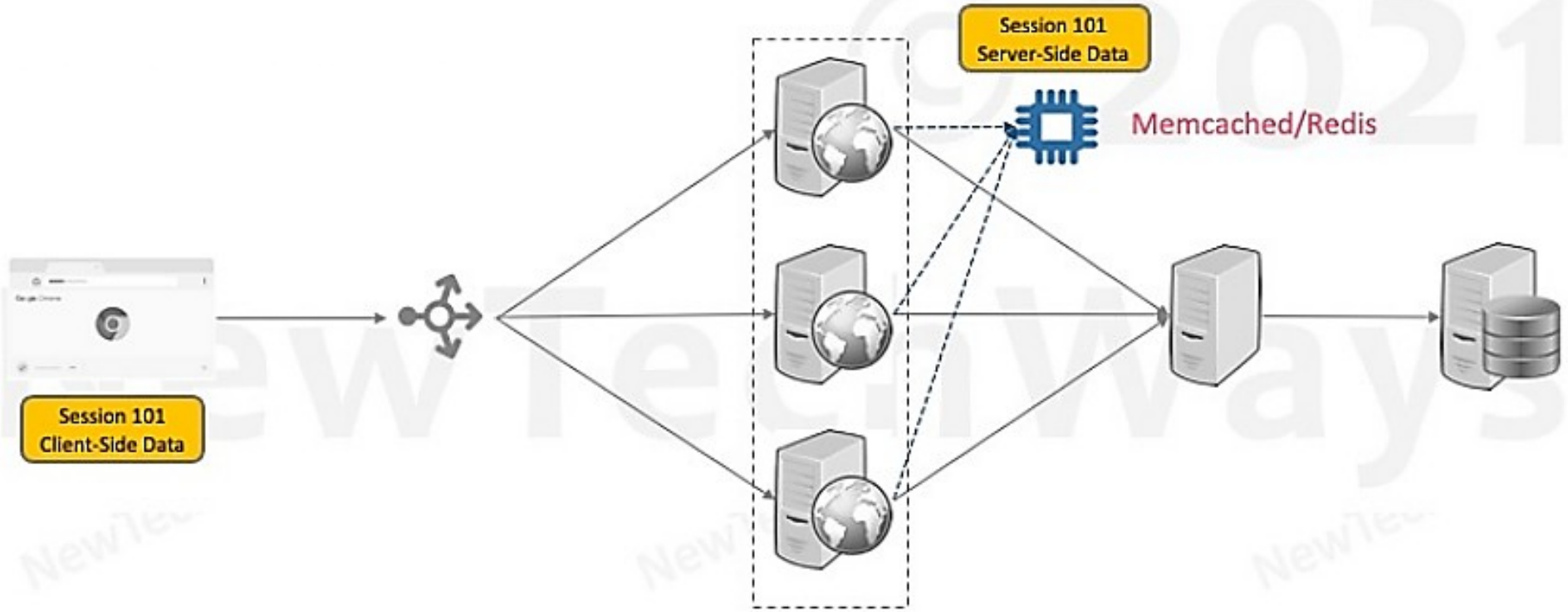
- Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.


Stateful vs Stateless Applications
Stateless:
No “state” is recorded about the user's session
NEWS
Person check a news website
Stateful:
Netflix records what has been watched
NETFLIX
Person logs into Netflix

STATEFUL


Node 1
Session 101
Data
Node 2
Session 101
Data
Node 3
Session 101 + Node 1

STATELESS

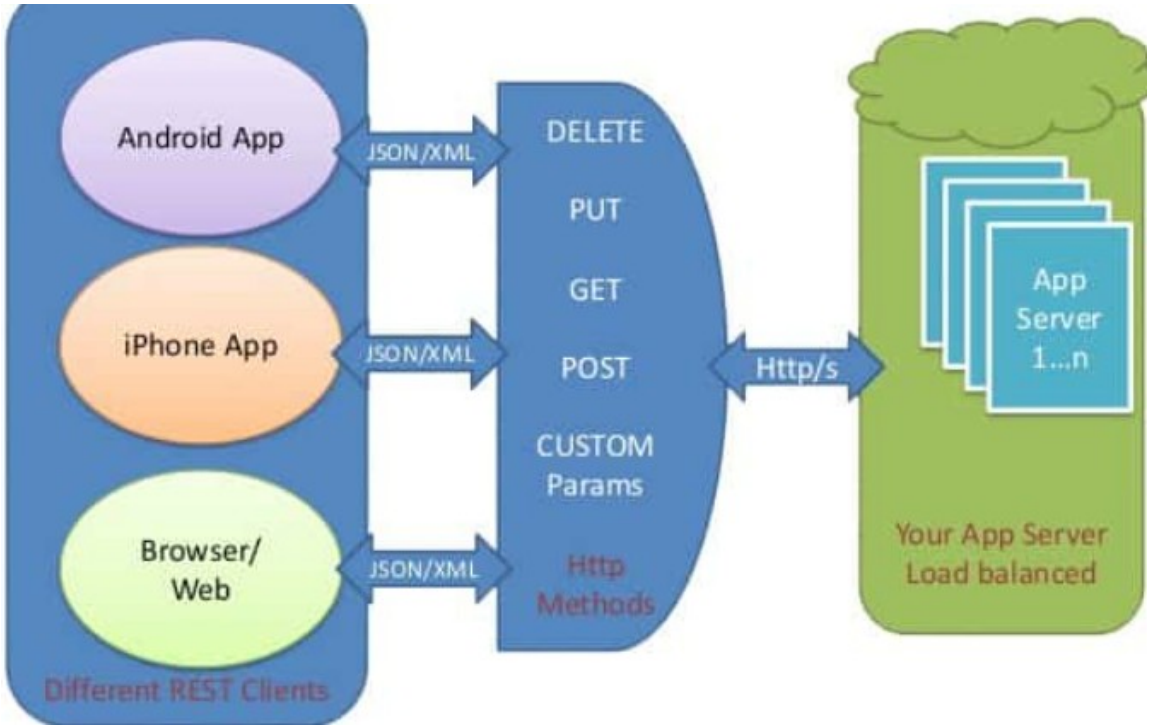

Session 101
Server-Side Data
Memcached/Redis
Session 101
Client-Side Data

3. CACHEABLE

- In order to improve network efficiency, cache constraints are added to the REST style.
- Cache constraints require that the data within a response to a request can be cacheable or not
- If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions.

4. UNIFORM INTERFACE

- Identification of resources (typically by URI).
- Manipulation of resources through representations.



5. LAYERED SYSTEM

- The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

6. CODE ON DEMAND (OPTIONAL)

- This states that the server can add more functionality to the REST client, by sending code that can be executable by that client. In the context of the web, one such example is JavaScript code that the server sends to the browser.

- For example, a web browser acts like a REST client and the server passes HTML content that the browser renders. At the server side, there is some sort of server-side language which is performing some logical work at the server side. But if we want to add some logic which will work in the browser then we (as server-side developers) will have to send some JavaScript code to the client side and the browser and then execute that JavaScript.

HAVE A GOOD DAY!