SOFTWARE ARCHITECTURE VIEWS

ECC 811 – SOFTWARE ENGINEERING

Monday, July 28, 2025

DEFINITION OF SOFTWARE ARCHITECTURE VIEW

1. Software architecture descriptions are commonly organized into views.

Each view addresses a set of system concerns, following the conventions of its *viewpoint*.

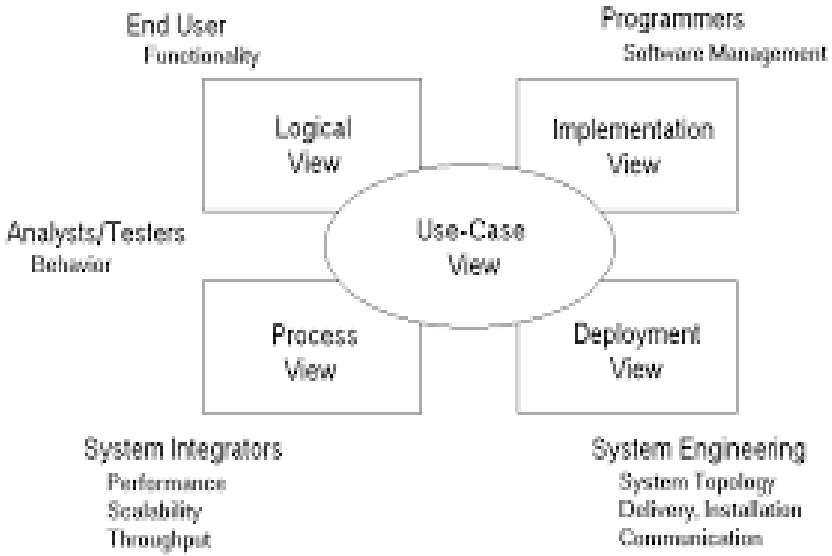
- Viewpoint - A position or direction from which something is observed or considered;
- View – Details or full specification considered from that viewpoint
  - (~ describes the notations, modeling and analysis techniques that express the architecture in question from the perspective of a given set of stakeholders)

2. Therefore, a software view is a representation of the system from the perspective of a viewpoint.

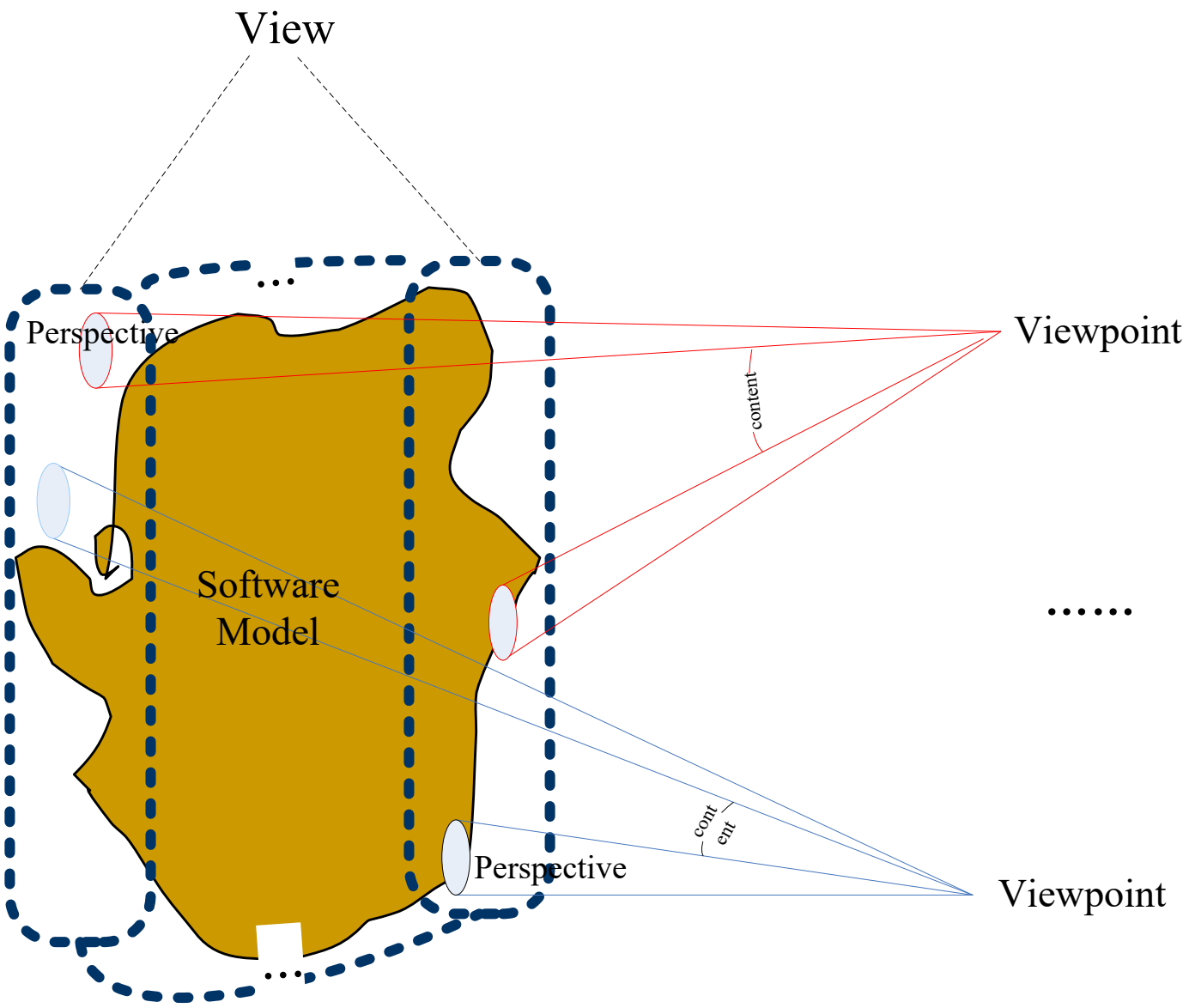
EXAMPLES OF SW ARCHITECTURE VIEWS

A view allows a user to examine a portion of a particular interest area.
Examples of software views include:

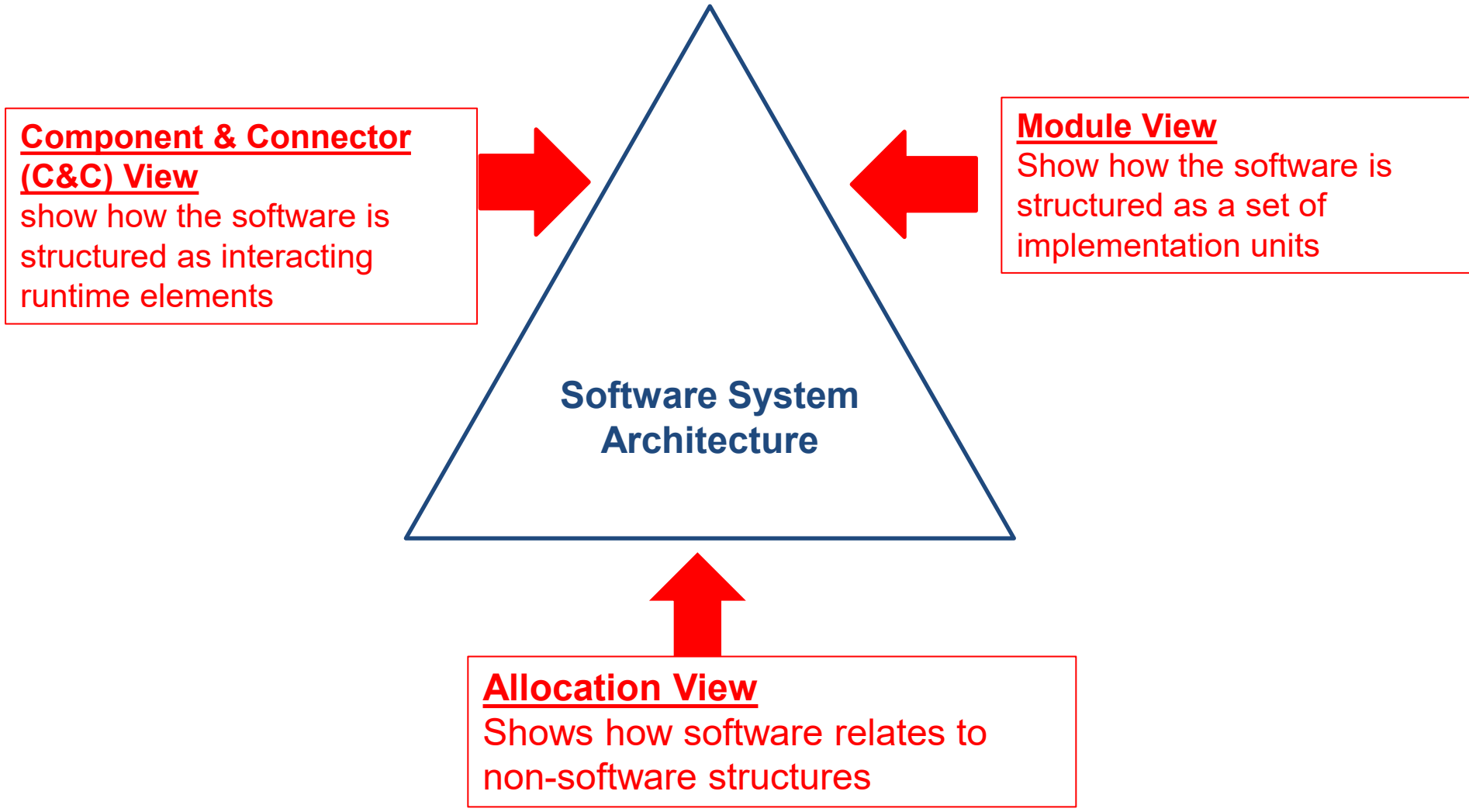
1. A Logical View (top / overall / bird's eye view)
    a) all functions,
    b) organizations,
2. Implementation view
    a) Technology (HW and networking)
    b) Module sequence
3. Developmental view
    a) Front end
    b) Backend
    c) Database connectivity
4. Process (Deployment) View
    a) Modules and its functions,
    b) Their interactions
    c) Control points
    d) Non Functional Requirements
5. Security View
    a) User ID / Password
    b) Graphical password
    c) Transactional password



SOFTWARE ARCHITECTURE VIEWS/VIEWPOINTS



VIEWS OF SOFTWARE ARCHITECTURE

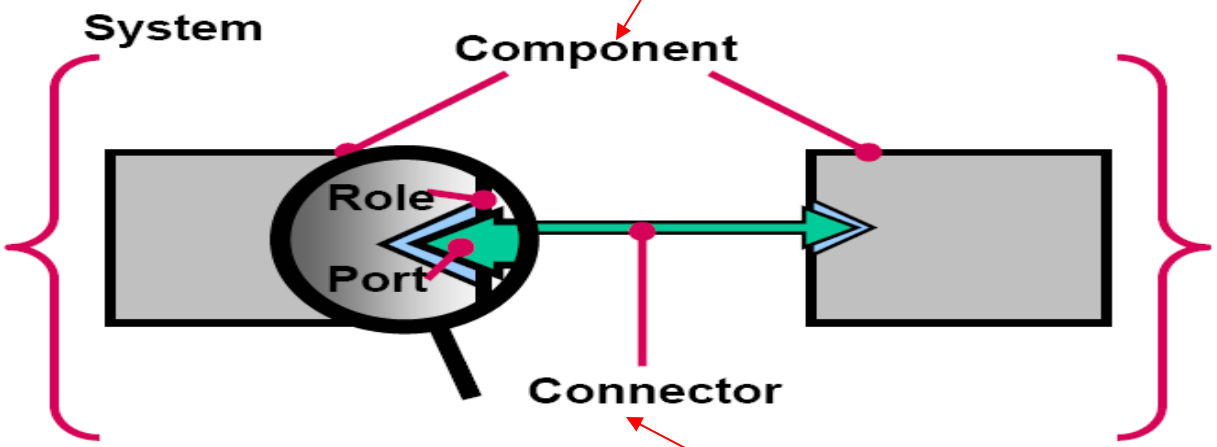


Software structures are categorized as (a) Module Structures, (b) Component and Connector Structures, (c) Allocation Structures

C & C ARCHITECTURE VIEW

1. **Component and Connector (C&C)** architecture view of a system has two main elements, i.e components and connectors.

Component
computational elements or data stores that have some presence during the system execution

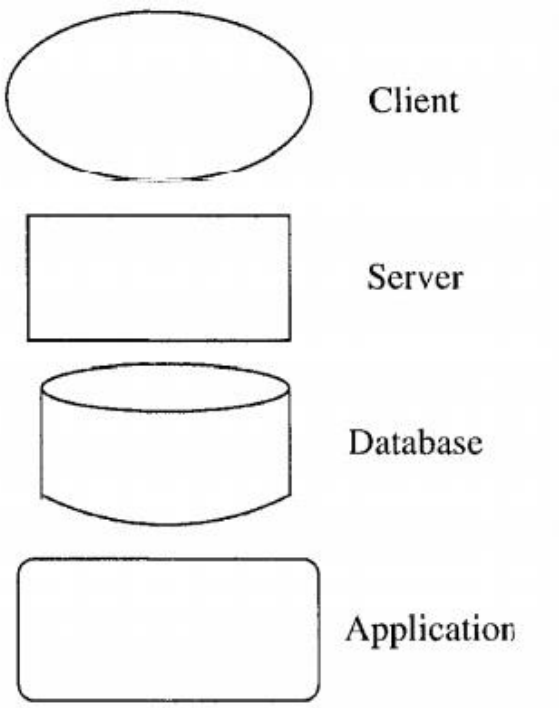


Connector
defines the means of interaction between components.

COMPONENTS

1. **Components** are generally units of computation or data stores in the system.

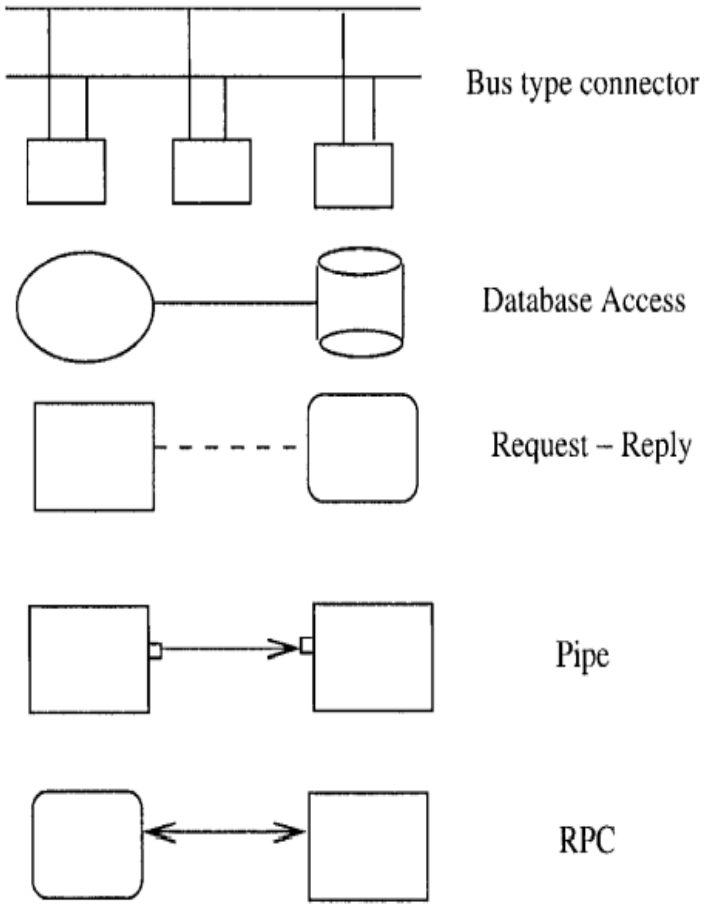
2. Each component has a name, which is generally chosen to represent the role of the component or the function it performs.



(a) Examples of components

WHAT IS A CONNECTOR?

1. Connectors is architectural element that models
   a) Interactions among components
   b) Rules that govern those interactions
2. Connector offers
   a) Simple interactions
      - Procedure calls
      - Shared variable access
   b) Complex & semantically rich interactions
      - Client-server protocols
      - Database access protocols
      - Asynchronous event multicast
3. Connector provides
   a) Interaction duct(s)
   b) Transfer of control and/or data



(a) Examples of software connectors

MODULES

1. Software elements are called modules
2. Modules are units of implementation
3. Each module is built using a separate Code
4. Each module has a specific assigned functional responsibility which includes:
   a) **Decomposition** – break large system to functionally working , understandable modules (Accounts, Registration, Appt. booking, Consulting)
   b) **Uses** – Each module is used by specific users following procedures (Dr Consultation, Registration etc..)
   c) **Layered** – Correct layered flow of use relations
   d) **Class** – generalisation allowing to reuse / inherit from other objects

MODULE STRUCTURES

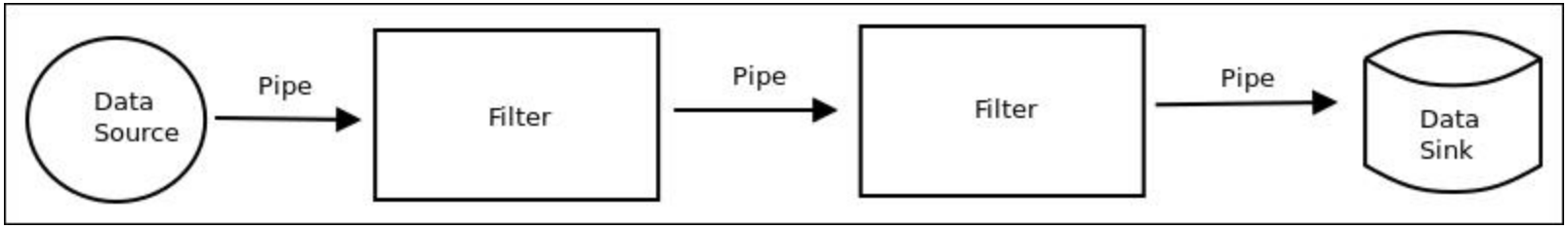
- Elements: modules (units of implementation). Modules are a code based way of considering the system
- Specifies:
  - Functional responsibility of modules
  - Other elements a module is allowed to use
  - Generalization and specialization relations
- Run-time operation of software is not a concern from this view (modularity, modifiability, development, data integrity, data hiding, reuse are consideed)

STYLES FOR C&C VIEW

- A style defines a family of architectures that satisfy the constraints of that style
- Module views: some of the common styles are decomposition, uses, generalization, and layered.
- Decomposition style: a module is decomposed into sub-modules, and the system becomes a hierarchy of modules.
- Uses style: modules are not parts of each other, but a module uses services of other modules (for example, a function call or a method invocation) to correctly do its own work.
- Generalization style: modules are often classes, and a child class inherits the properties of the parent class and specializes it.

PIPE & FILTER STYLE / 01

- **Pipe and filter style** of architecture's goal is to produce some output data by suitably transforming the input data.
- It consists of one or more data sources. The data source is connected to data filters via pipes. Filters process the data they receive, passing them to other filters in the pipeline. The final data is received at a Data Sink.
- A filter may have more than one inputs and more than one outputs.
- This style is suited for systems that primarily do data transformation some input data is received and processed.



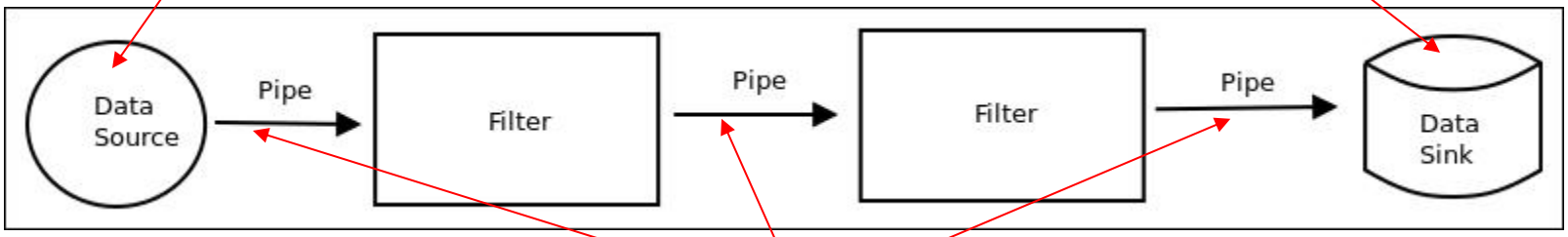
PIPE & FILTER STYLE / 02

Data source

- Provides a sequence of data values of the same structure or type.
- Can actively push the data values to the first processing stage, or passively provide data when the first filter pulls.

Data sink

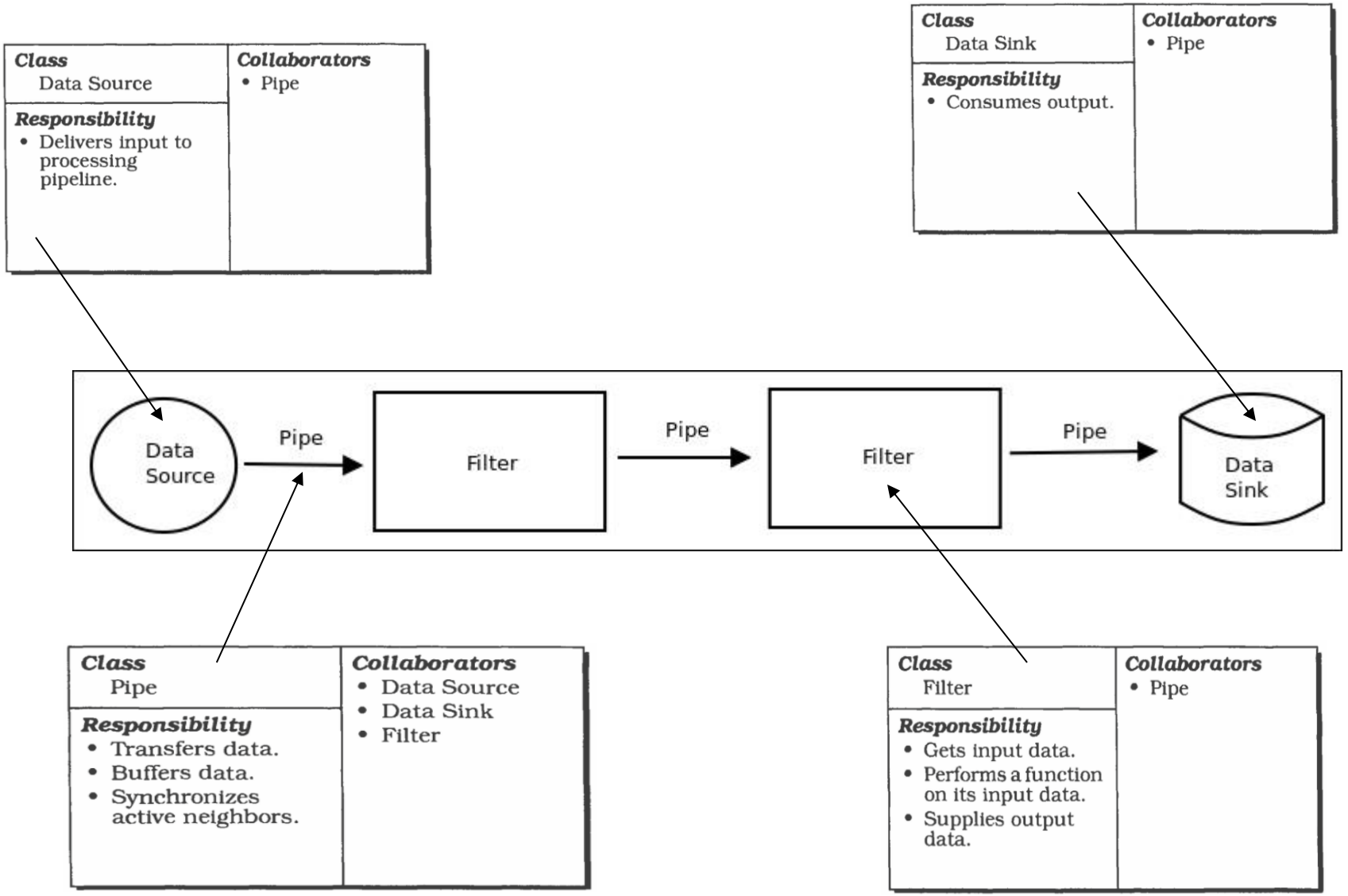
- collects the results from the end of the pipeline.
- Active data sink pulls results of the preceding processing stage
- Passive one allows the preceding filter to push or write the results into it.



Pipes

connections between filters, between the data source and the first filter, and between the last filter and the data sink.
If two active components are joined, the pipe synchronises them with a FIFO buffer.

PIPE & FILTER STYLE / 03



SHARED DATA STYLE /01

- **In data centred style** (also called data centred architecture), data is exchanged between components through shared storage.
- The computational components are coordinated, with subroutines to a main program sequences through them.
- Data is then communicated between the components through shared storage.
- Communication between the computational components and shared data is an unconstrained read-write protocol.

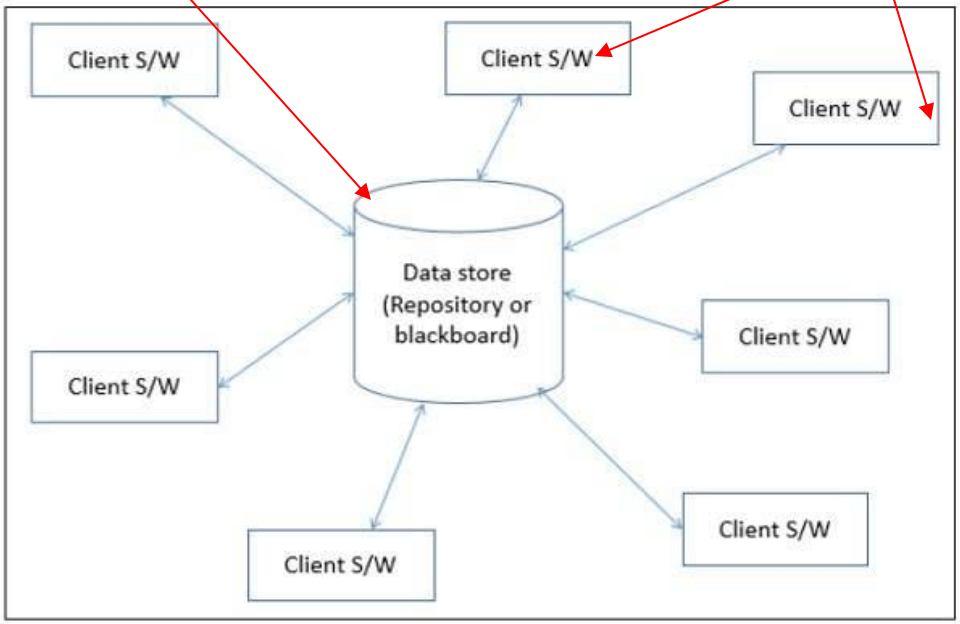
SHARED DATA STYLE /02

Data store:

- Data is centralized and accessed frequently by other components, which modify data.
- The main purpose of this style is to achieve integrality of data.

Shared Data Components

- Different components communicate through shared data repositories.
- Components access a shared data structure and are relatively independent, in that, they interact only through the data store.

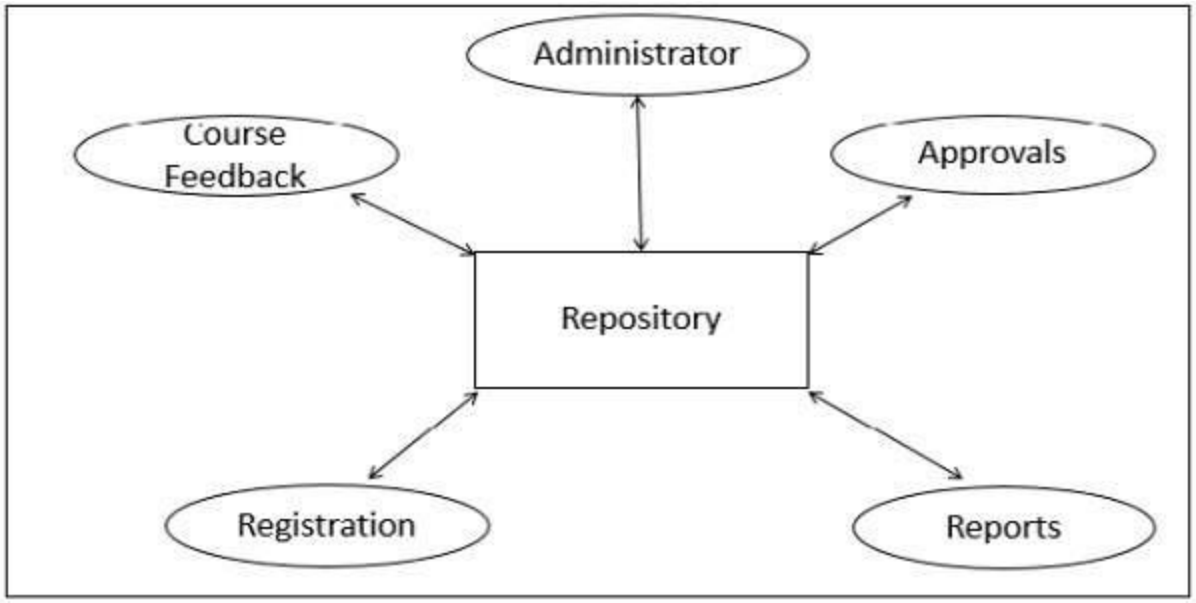


VARIANTS OF SHARED DATA VIEW

- There are two variations of this style possible, i.e repository and blackboard.
- **Repository Architecture**: the data store is passive and the clients (software components or agents) of the data store are active, which control the logic flow. The participating components check the data-store for changes.
- **Blackboard Architecture Style:** the data store is active and its clients are passive. Therefore the logical flow is determined by the current data status in data store. It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.

REPOSITORY ARCHITECTORAL VIEW

1. The computational processes are independent and triggered by incoming requests.
2. If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is traditional database or repository architecture, or passive repository.
3. This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers, and CASE (computer aided software engineering) environments.



ADVANTAGES OF REPOSITORY ARCHITECTORAL VIEW

Repository Architecture Style has following advantages −

1. Provides data integrity, backup and restore features.
2. Provides scalability and reusability of agents as they do not have direct communication with each other.
3. Reduces overhead of transient data between software components.

DISADVANTAGES OF REPOSITORY ARCHITECTORAL VIEW

Because of being more vulnerable to failure and data replication or duplication, Repository Architecture Style has following disadvantages:

1. High dependency between data structure of data store and its agents.
2. Changes in data structure highly affect the clients.
3. Evolution of data is difficult and expensive.
4. Cost of moving data on network for distributed data

BLACKBOARD ARCHITECTORAL VIEW

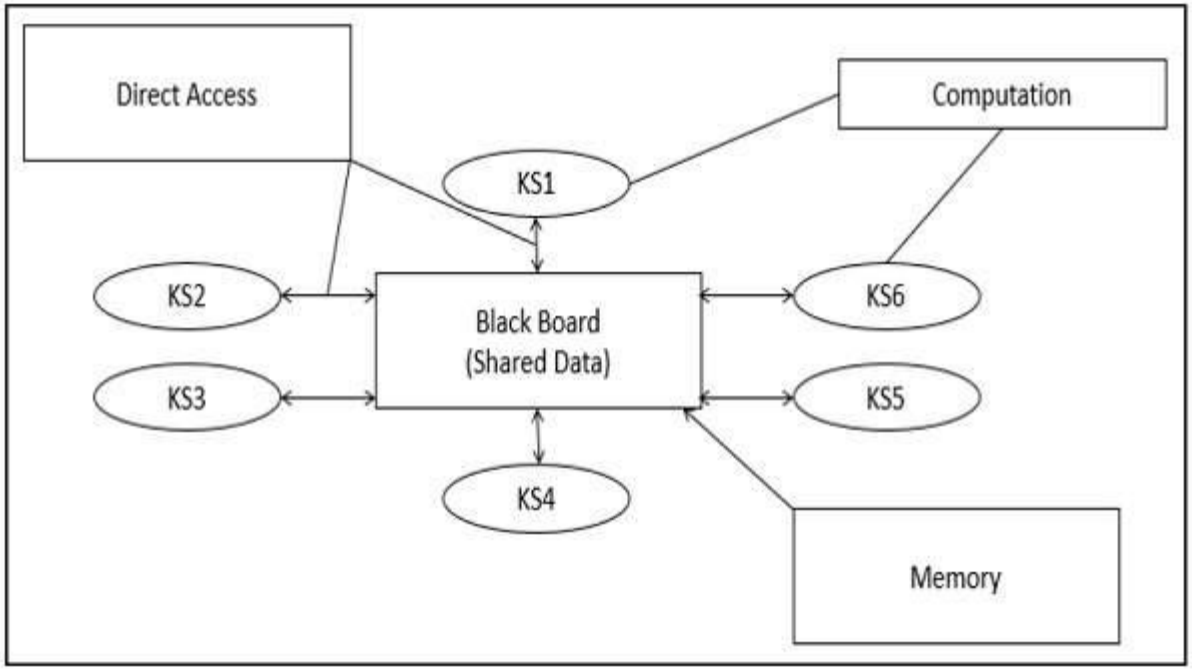
- In Blackboard Architecture Style, the data store is active and its clients are passive.
- The logical flow is determined by the current data status in data store.
- It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.
- The components interact only through the blackboard. The data-store alerts the clients whenever there is a data-store changes. The current state of the solution is stored in the blackboard and processing is triggered by the state of the blackboard.

PARTS OF A BLACKBOARD ARCHITECTURE MODEL

Knowledge Sources (KSn)

1. Also known as Listeners or Subscribers are distinct and independent units.
2. They solve parts of a problem and aggregate partial results.

Computation/control
manages tasks and checks the work state.



ADVANTAGES OF BLACKBOARD ARCHITECTURE

1. Blackboard Model provides concurrency that allows all knowledge sources to work in parallel as they independent of each other.
2. Its scalability feature facilitates easy steps to add or update knowledge source.
3. Further, it supports experimentation for hypotheses and reusability of knowledge source agents.

DISADVANTAGES OF BLACKBOARD ARCHITECTURE

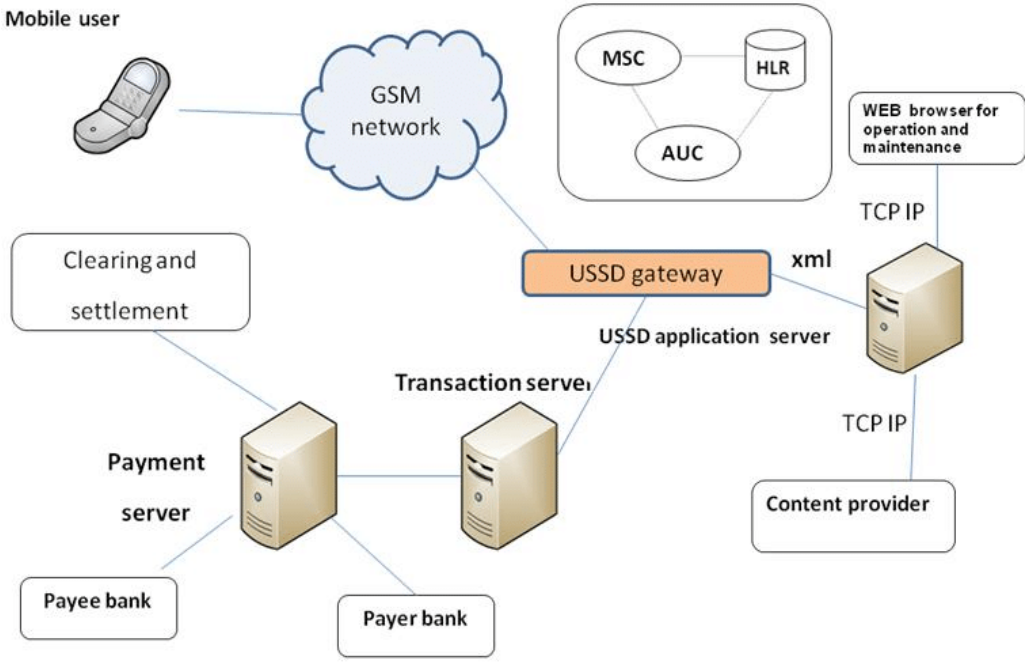
- The structural change of blackboard may have a significant impact on all of its agents, as close dependency exists between blackboard and knowledge source.
- Blackboard model is expected to produce approximate solution; however, sometimes, it becomes difficult to decide when to terminate the reasoning.
- Suffers some problems in synchronization of multiple agents, therefore, it faces challenge in designing and testing of the system.

CLIENT SERVER ARCHITECTURE

- In this style, there are two component types—clients and servers.
- A constraint of this style is that a client can only communicate with the server, and cannot communicate with other clients. The communication between a client component and a server component is initiated by the client the client sends a request for some service that the server supports.
- The server receives the request at its defined port, performs the service, and then returns the results of the computation to the client who requested the service.

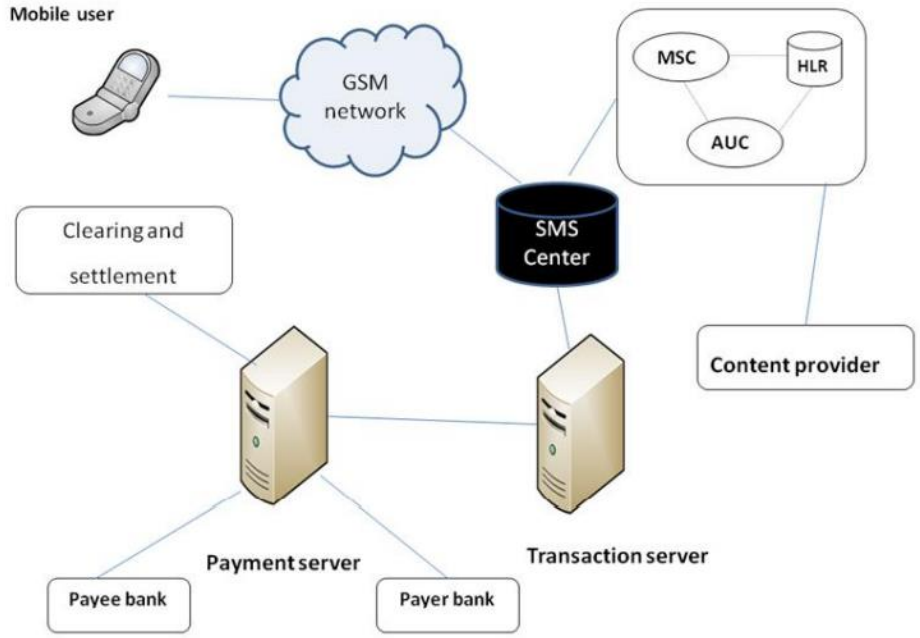
EXAMPLE: ARCHITECTURE OF USSD MOBILE PAYMENT

- USSD is a capability of GSM network used for transferring information between mobile phone and application.
- User requests a service by entering short code on mobile. Format of code is standardized and content is specified for each service, the content can be containing USSD code, from account, to account, amount, currency, target mobile number.
- USSD gateway service provider communicates with GSM network through SS7 protocol.



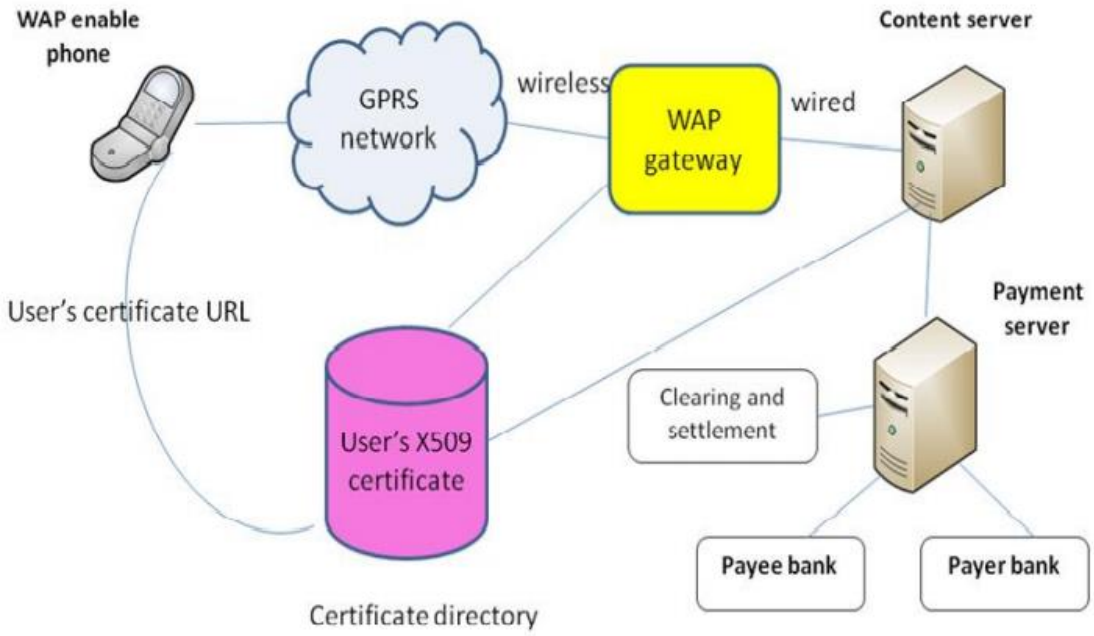
EXAMPLE: ARCHITECTURE OF SMS MOBILE PAYMENT SYSTEM

- No special software has been used in this platform.
- The communication channel between user and payment network is SMS.
- A standard format is used for sending messages such as timestamp, random number, from account, to account, amount, currency, and target mobile number. The payer authentication is based on payer mobile number and PIN.
- Because of security problem related to PIN a safer solution is achieved by one-time password.
- Some of mobile services which can be provided by this platform include bill payment, financial operation like account history and funds transfer



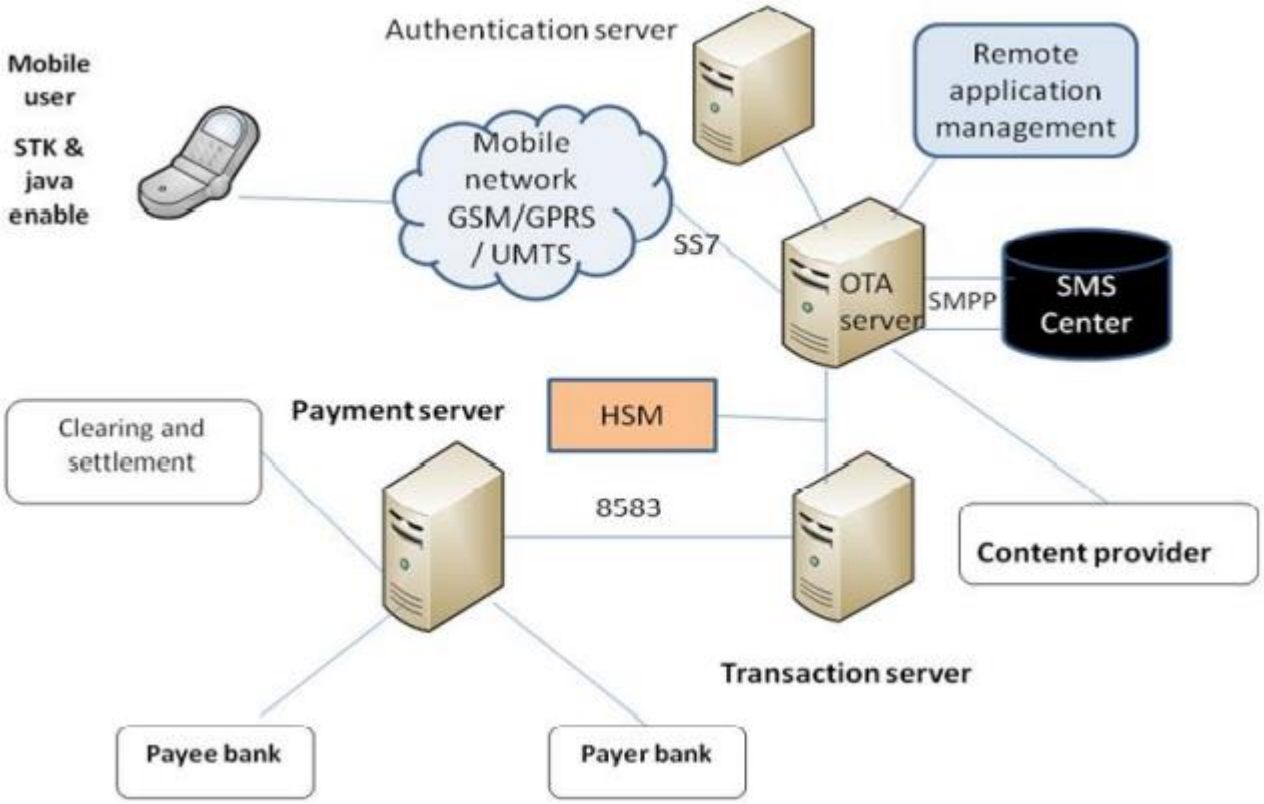
EXAMPLE: ARCHITECTURE OF WAP/GPRS MOBILE PAYMENT SYSTEM

- Authentication of the payer is done by digital certificate, mobile phone number and PIN.
- A URL link in mobile phone download associated certificate. Transferring of information routed by GPRS network and WAP enabled phone.
- WAP uses a special language WML for communication Between WAP Gateway and content on the Internet.
- The WAP Gateway converts between WML and HTML, allowing delivery of WAP based content to a WAP capable mobile device



EXAMPLE: ARCHITECTURE OF SIM-BASED APPLICATION MOBILE PAYMENT SYSTEM

- This platform is based on application installed on SIM.
- User receives payment software and other services directly through OTA server. When the software is successfully installed, user can send a request for supported services onto operator.
- This request is processed in OTA server and recorded on transaction server.



REVIEW QUESTIONS

1. Define View.
2. Define viewpoint.
3. What is the difference between view and viewpoint?
4. List the various views in software architecture
5. Why multiple views are required in defining SW Architecture ?
6. How SW architecture is connected to view and viewpoint ?
7. Explain logical view with an example. Who uses this view ?
8. Explain Process (Deployment) view with an example. Who uses this view ?
9. Explain Development (Implementation) view with an example. Who uses this view ?
10. Explain physical view with an example. Who uses this view ?
11. Explain usecase view (scenario). Why is this view important ?
12. Define Structure
13. How is structure different from views ?
14. List the categories of structure
15. Define module structure
16. Define component and connector structure
17. What do you understand by Allocation structure ?

16m Question Bank

1. Define and explain the importance of various views in a SWA
2. Define structure. List and explain various structures considered in SWA
3. Explain with a neat diagram Kruchten's 4+1 RUP view
4. Explain with a neat diagram Siemens 4 view
5. Explain with a neat diagram SEI view
6. Case study on following views
    1. RUP 4+1
    2. Siemens 4
    3. SEI