SE-2002
SOFTWARE DESIGN AND ARCHITECTURE
RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Architectural Patterns

TODAY'S OUTLINE

- **Architectural Patterns**

  - MVC

  - Layered Architecture

  - Repository Architecture

  - Client-server Architecture

  - Client-server Architecture

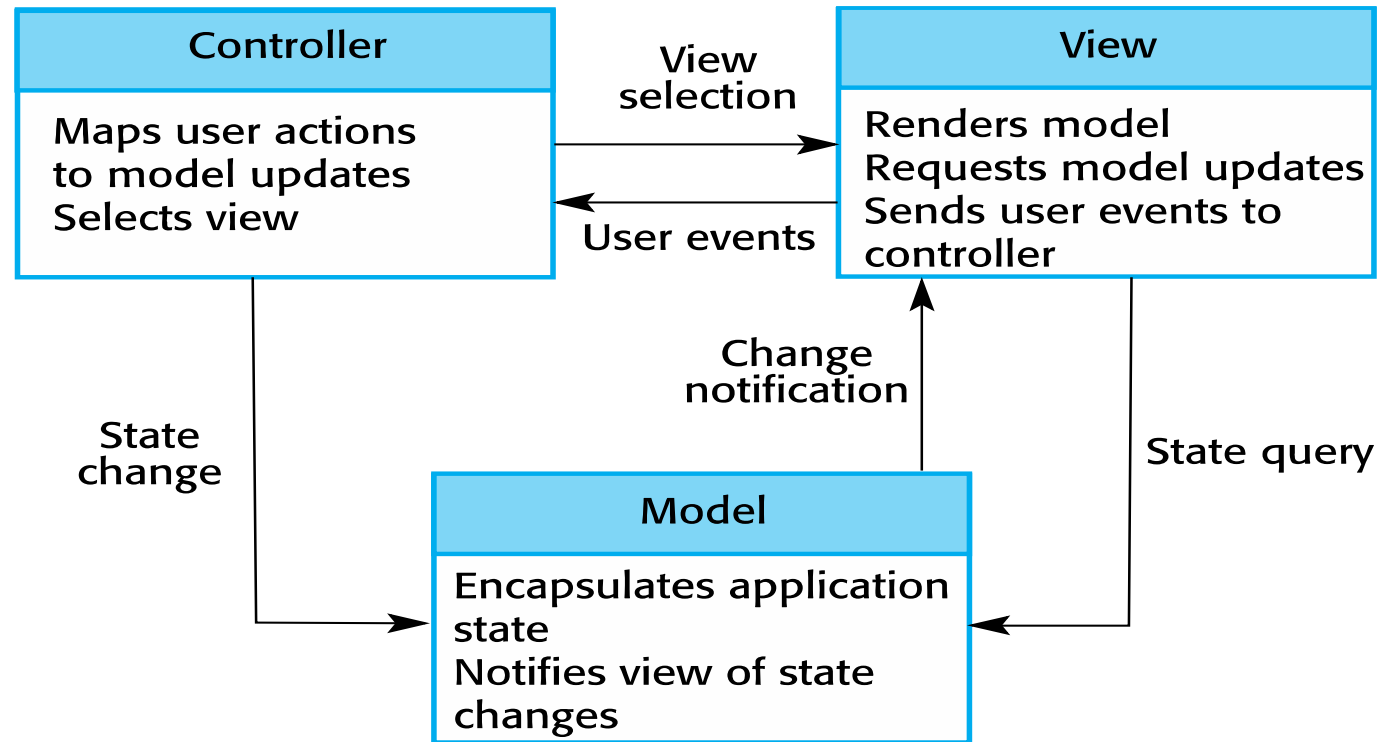- Application Architectures

ARCHITECTURAL PATTERNS

- Patterns are a means of representing, sharing and reusing knowledge.

- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.

- Patterns should include information about when they are and when they are not useful.

THE MODEL-VIEW-CONTROLLER (MVC) PATTERN

The goal of this pattern is to split a large application into specific section that has its own purpose.

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

THE ORGANIZATION OF THE MODEL-VIEW-CONTROLLER

WEB APPLICATION ARCHITECTURE USING THE MVC PATTERN
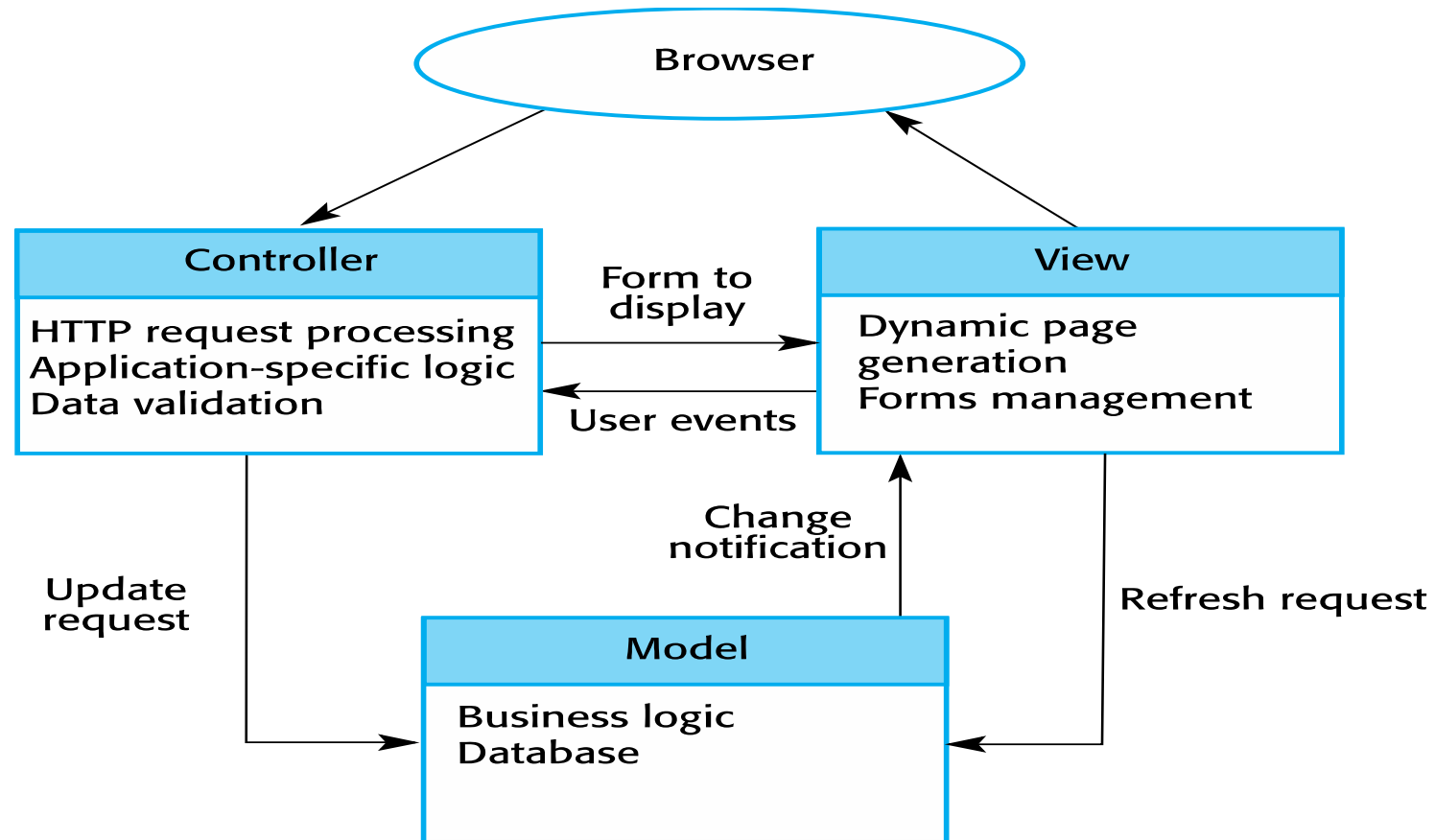
LAYERED ARCHITECTURE

- Used to model the interfacing of sub-systems.

- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.

- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

THE LAYERED ARCHITECTURE PATTERN

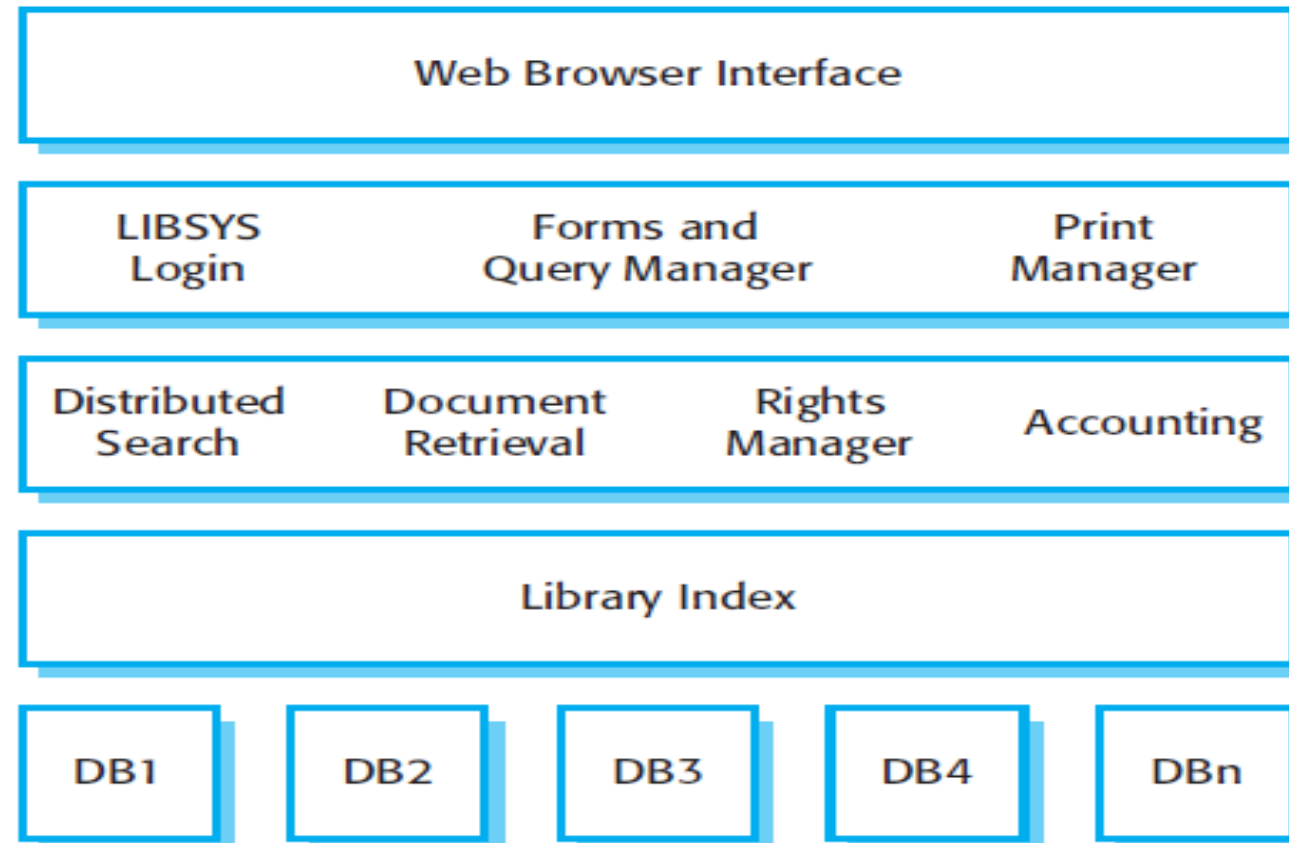| Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

A GENERIC LAYERED ARCHITECTURE

User interface

User interface management
Authentication and authorization

Core business logic/application functionality
System utilities

System support (OS, database etc.)

THE ARCHITECTURE OF THE LIBSYS SYSTEM

Web Browser Interface

LIBSYS Login	Forms and Query Manager	Print Manager

Distributed Search	Document Retrieval	Rights Manager	Accounting

Library Index

DB1	DB2	DB3	DB4	DBn

THE ARCHITECTURE OF THE ILEARN SYSTEM

Browser-based user interface   iLearn app

Configuration services

| Group management | Application management | Identity management |

Application services

Email   Messaging   Video conferencing   Newspaper archive

Word processing   Simulation   Video storage   Resource finder

Spreadsheet   Virtual learning environment   History archive

Utility services

Authentication   Logging and monitoring   Interfacing

User storage   Application storage   Search

REPOSITORY ARCHITECTURE

- Data is stored in a central shared repository.

- Components interact through the repository only.

- Suited to applications whose data is generated by one component and used by another.

- Data integrity, a database will ensure the data is accurate and consistent over its lifespan.

Advantages: -

- Components are independent/separate.

- Changes to data are automatically available to other components.

- Communication between components may be inefficient

REPOSITORY ARCHITECTURE

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used a this is an efficient data sharing mechanism.

THE REPOSITORY PATTERN

Name	Repository
Description	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.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
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.

A REPOSITORY ARCHITECTURE FOR AN IDE
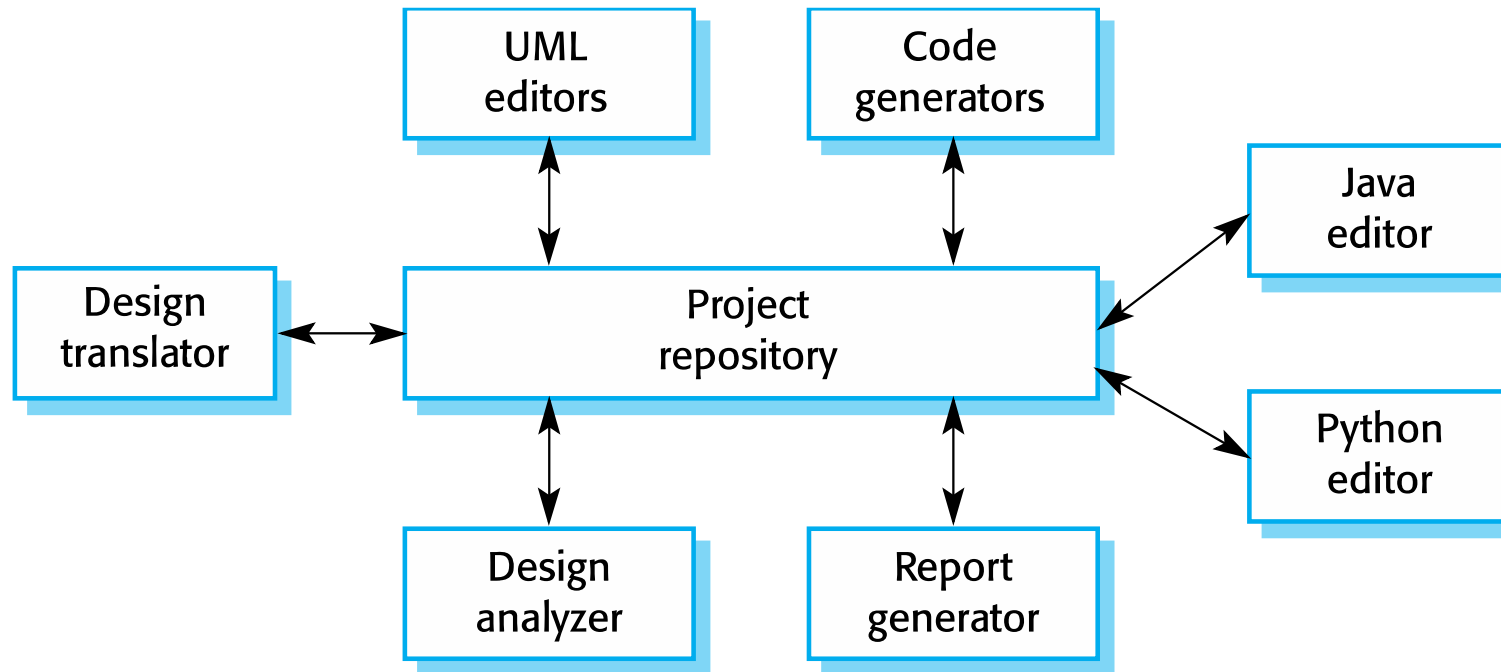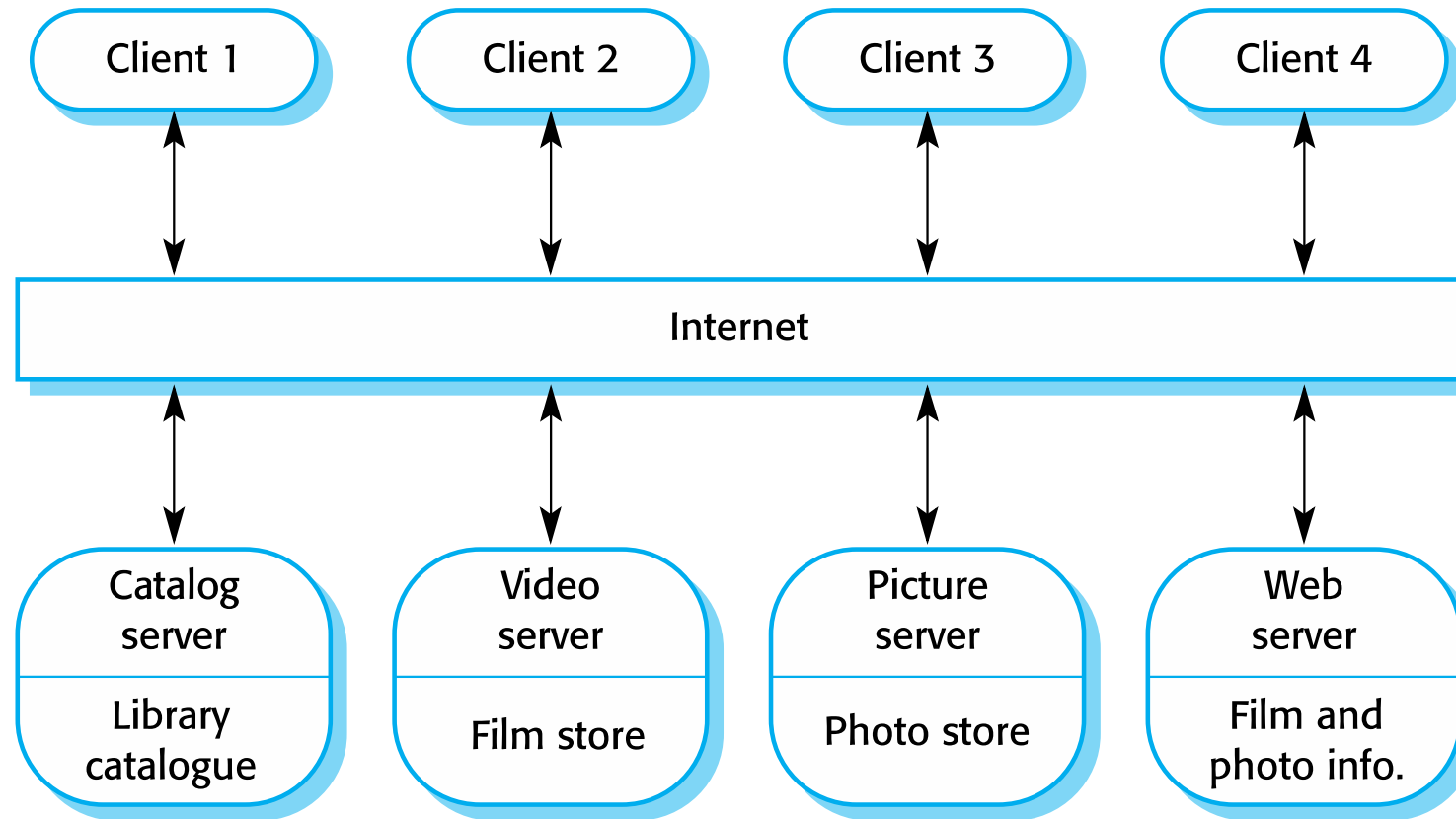
CLIENT-SERVER ARCHITECTURE

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

THE CLIENT–SERVER PATTERN

| Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

A CLIENT–SERVER ARCHITECTURE FOR A FILM LIBRARY

CLIENT-SERVER ARCHITECTURE:

- **Front-End:** This is the piece of software that interacts with users, even if they are on different platforms with different technologies. Any front-end module in a client-server architecture is designed to interact with all existing devices on the market. This level contains the login screens, menus, data screens.

- **Application server:** This is the server where the software modules of the application are installed. It connects to the database (called back-end) and interacts with users (called front-end). The application server is like the waiter on our restaurant example.

- **Database server:** This server contains the tables, indexes, and data managed by the application. Searches and insert/delete/update operations are executed here.
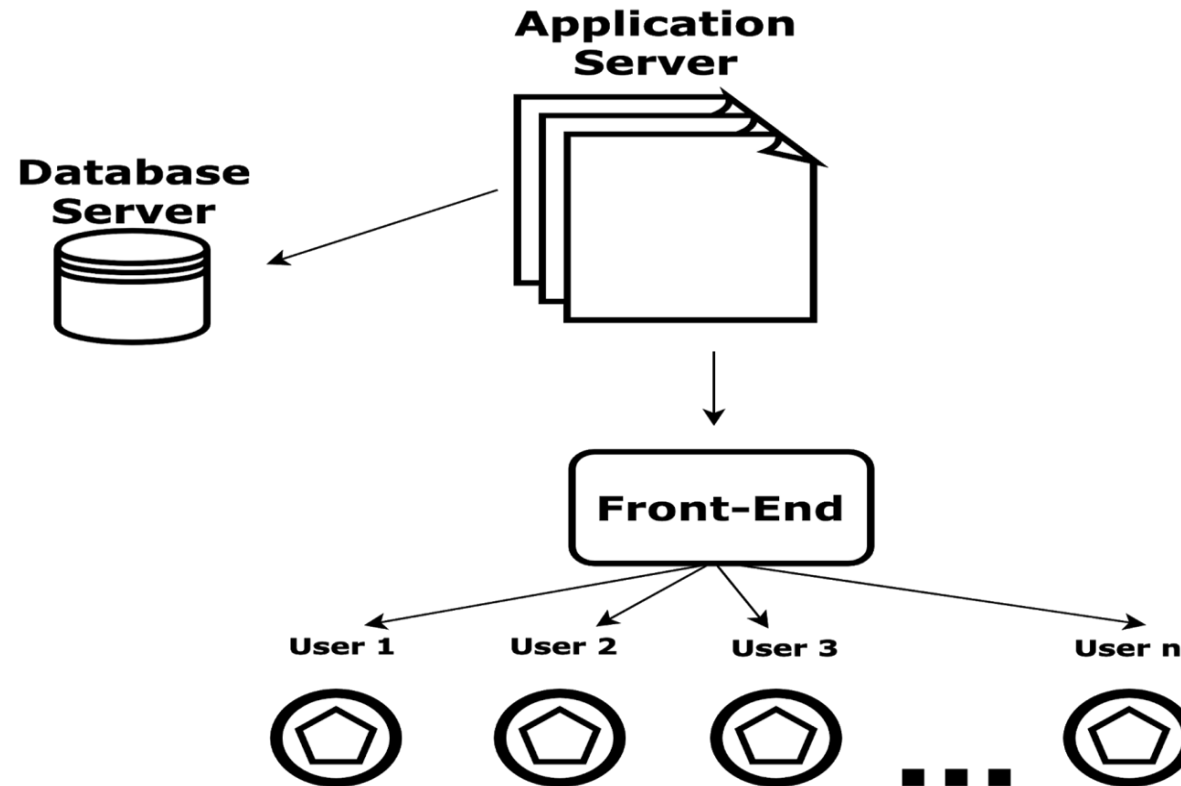
PIPE AND FILTER ARCHITECTURE

- Functional transformations process their inputs to produce outputs.

- May be referred to as a pipe and filter model (as in UNIX shell).

- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.

- Not really suitable for interactive systems.

THE PIPE AND FILTER PATTERN

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

AN EXAMPLE OF THE PIPE AND FILTER ARCHITECTURE USED IN A PAYMENTS SYSTEM
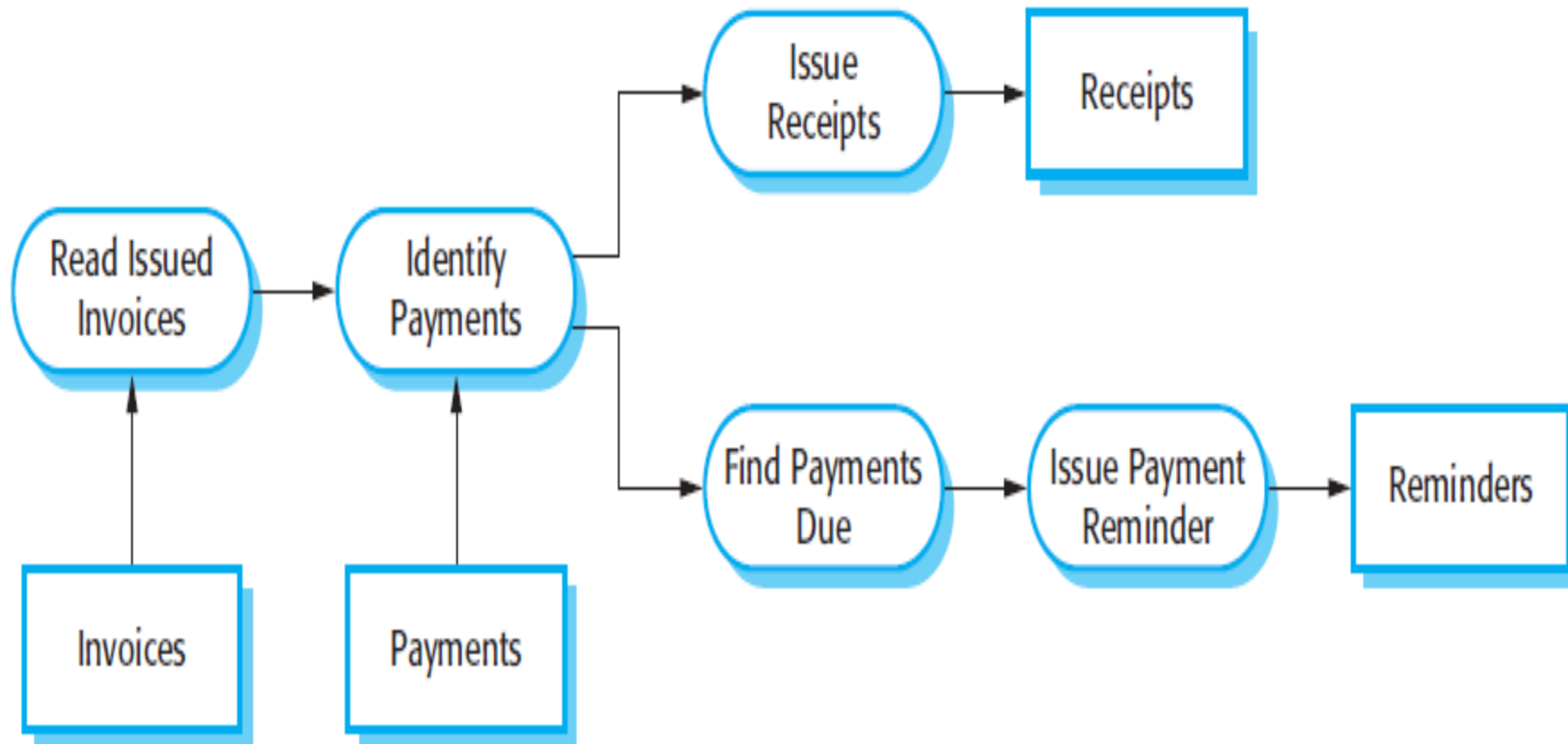
APPLICATION ARCHITECTURES

APPLICATION ARCHITECTURES

- Application systems are designed to meet an organisational need.

- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.

- A generic application architecture is an architecture for a type of software system that may be configured and adapted to create a system that meets specific requirements.

USE OF APPLICATION ARCHITECTURES

- As a starting point for architectural design.

- As a design checklist.

- As a way of organising the work of the development team.

- As a means of assessing components for reuse.

- As a vocabulary for talking about application types.

EXAMPLES OF APPLICATION TYPES

- Data processing applications

  - Data driven applications that process data in batches without explicit user intervention during the processing.

- Transaction processing applications

  - Data-centred applications that process user requests and update information in a system database.

- Event processing systems

  - Applications where system actions depend on interpreting events from the system's environment.

- Language processing systems

  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.
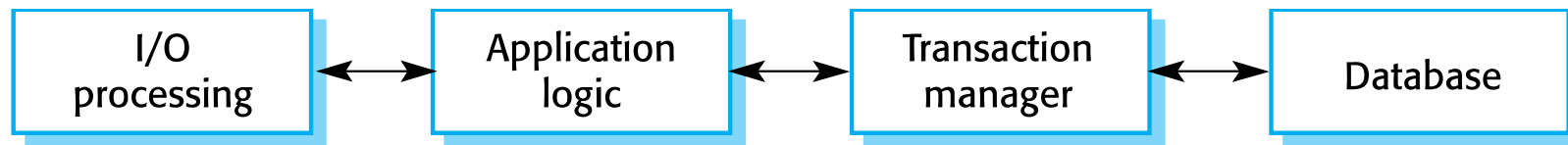
APPLICATION TYPE EXAMPLES

- Two very widely used generic application architectures are transaction processing systems and language processing systems.

- Transaction processing systems

  - E-commerce systems;

  - Reservation systems.

- Language processing systems
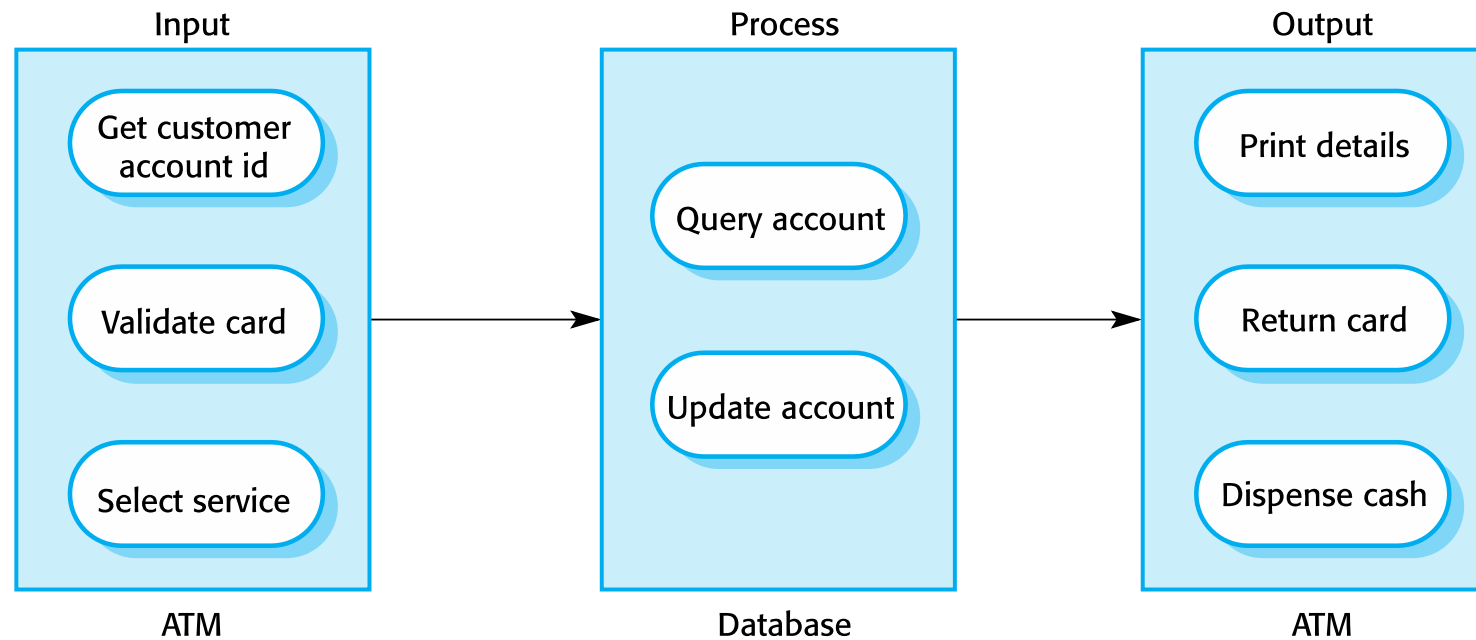
  - Compilers;

  - Command interpreters.

TRANSACTION PROCESSING SYSTEMS

- Process user requests for information from a database or requests to update the database.

- From a user perspective a transaction is:

  - Any coherent sequence of operations that satisfies a goal;

  - For example - find the times of flights from London to Paris.

- Users make asynchronous requests for service which are then processed by a transaction manager.

THE STRUCTURE OF TRANSACTION

```
I/O             Application          Transaction
processing  <->    logic      <->    manager     <->   Database
```

THE SOFTWARE ARCHITECTURE OF AN ATM SYSTEM

Input

- Get customer account id
- Validate card
- Select service

ATM

Process

- Query account
- Update account

Database

Output

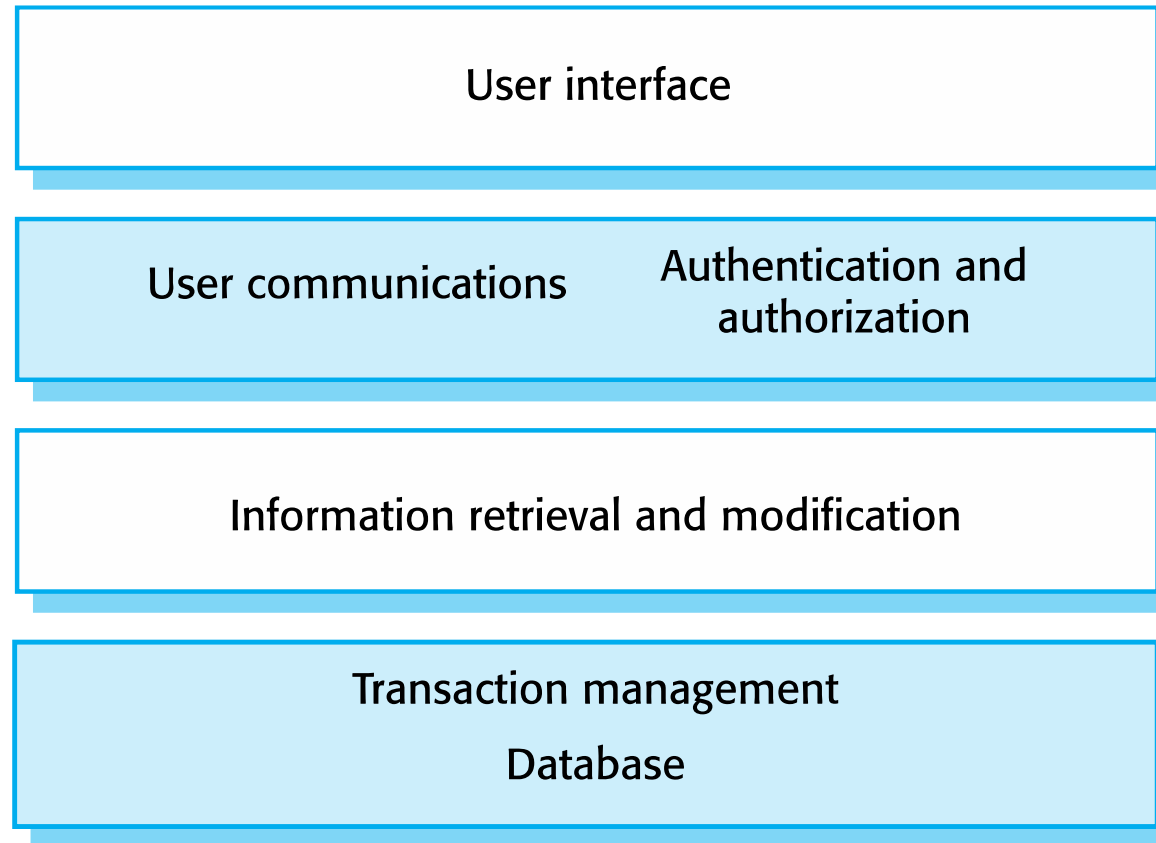- Print details
- Return card
- Dispense cash
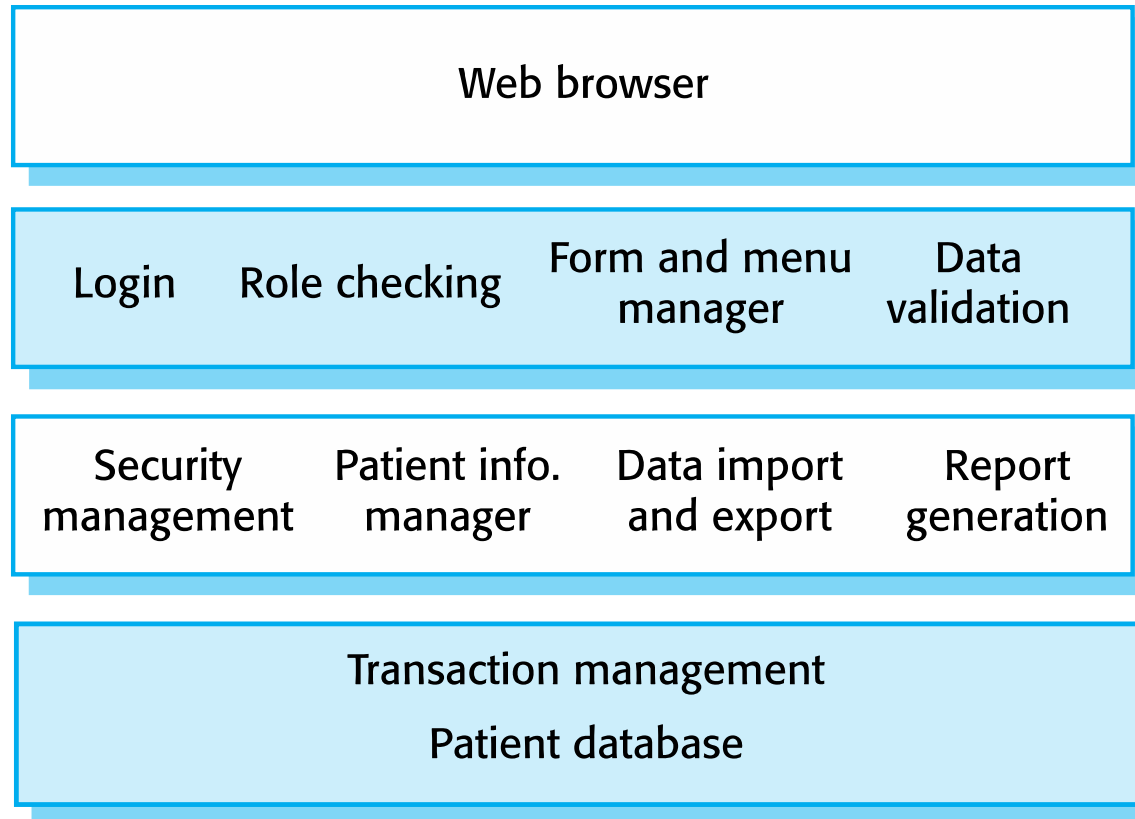
ATM

INFORMATION SYSTEMS ARCHITECTURE

- Information systems have a generic architecture that can be organised as a layered architecture.

- These are transaction-based systems as interaction with these systems generally involves database transactions.

- Layers include:

  - The user interface

  - User communications

  - Information retrieval

  - System database

LAYERED INFORMATION SYSTEM ARCHITECTURE

User interface

User communications | Authentication and authorization

Information retrieval and modification

Transaction management

Database

THE ARCHITECTURE OF THE MENTCARE SYSTEM

Web browser

Login Role checking Form and menu manager Data validation

Security management Patient info. manager Data import and export Report generation

Transaction management

Patient database

WEB-BASED INFORMATION SYSTEMS

- Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.

- For example, e-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.

- In an e-commerce system, the application-specific layer includes additional functionality supporting a 'shopping cart' in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

SERVER IMPLEMENTATION

- These systems are often implemented as multi-tier client server/architectures (discussed in Chapter 17)
  - The web server is responsible for all user communications, with the user interface implemented using a web browser;
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;
  - The database server moves information to and from the database and handles transaction management.

LANGUAGE PROCESSING SYSTEMS

- Accept a natural or artificial language as input and generate some other representation of that language.

- May include an interpreter to act on the instructions in the language that is being processed.

- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data

# THE ARCHITECTURE OF A LANGUAGE PROCESSING SYSTEM
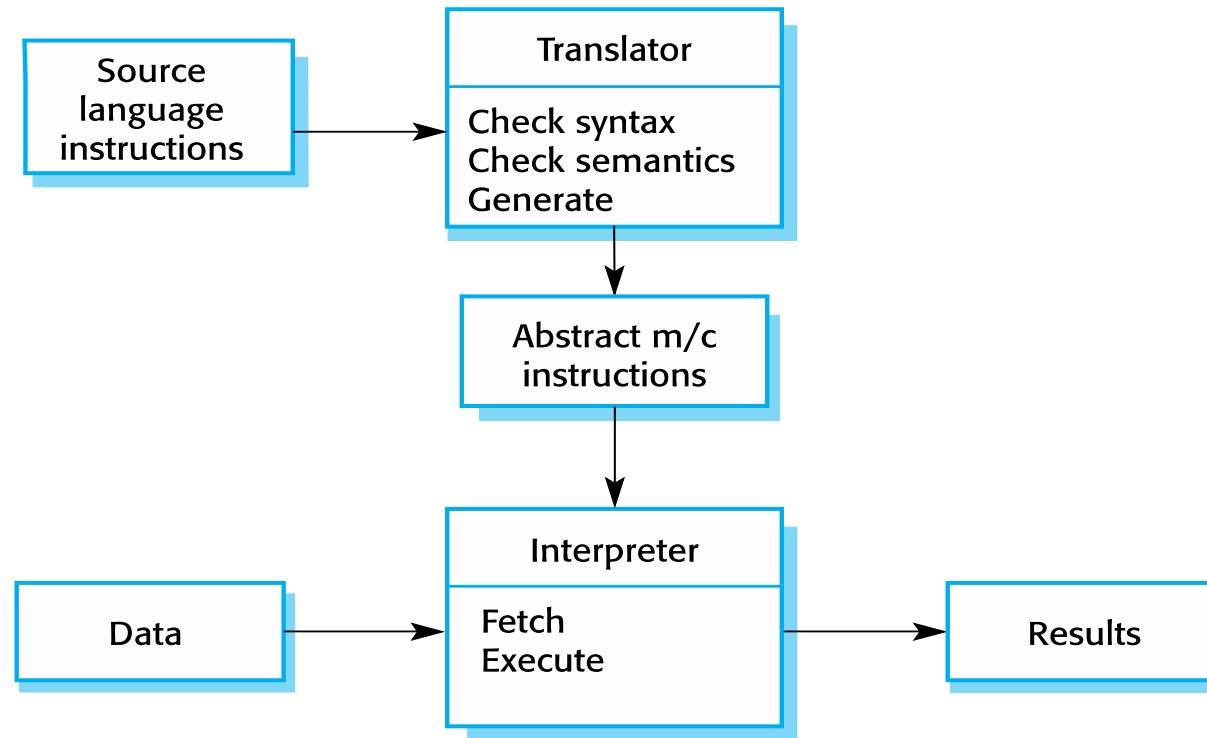
COMPILER COMPONENTS

- A lexical analyzer, which takes input language tokens and converts them to an internal form.

- A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.

- A syntax analyzer, which checks the syntax of the language being translated.

- A syntax tree, which is an internal structure representing the program being compiled.
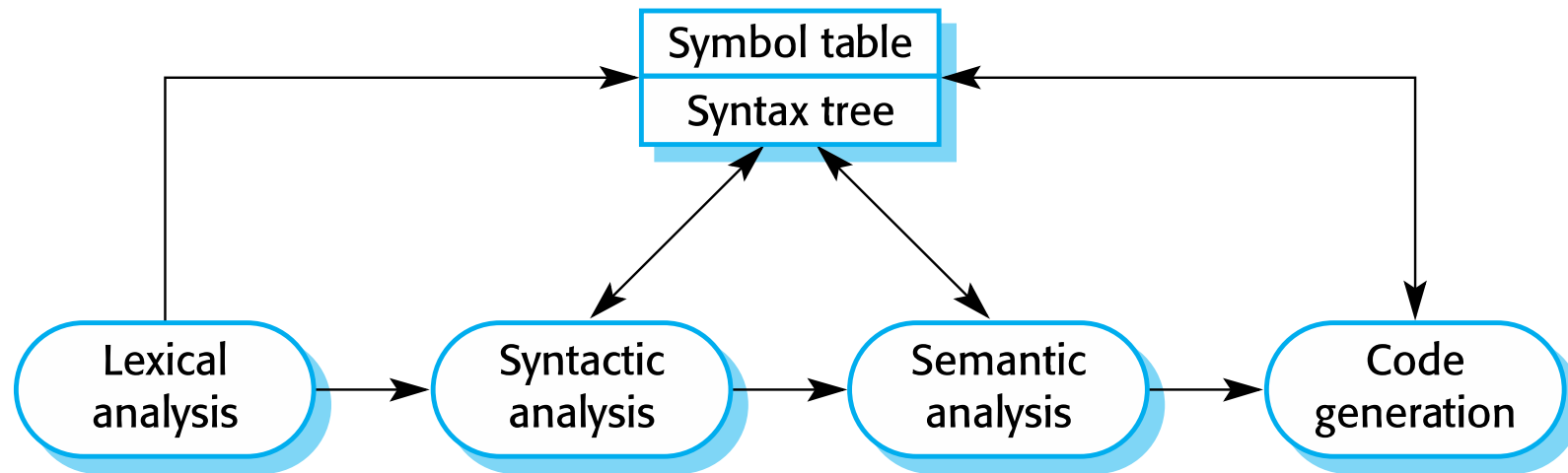
COMPILER COMPONENTS

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.

- A code generator that 'walks' the syntax tree and generates abstract machine code.

A REPOSITORY ARCHITECTURE FOR A LANGUAGE PROCESSING SYSTEM
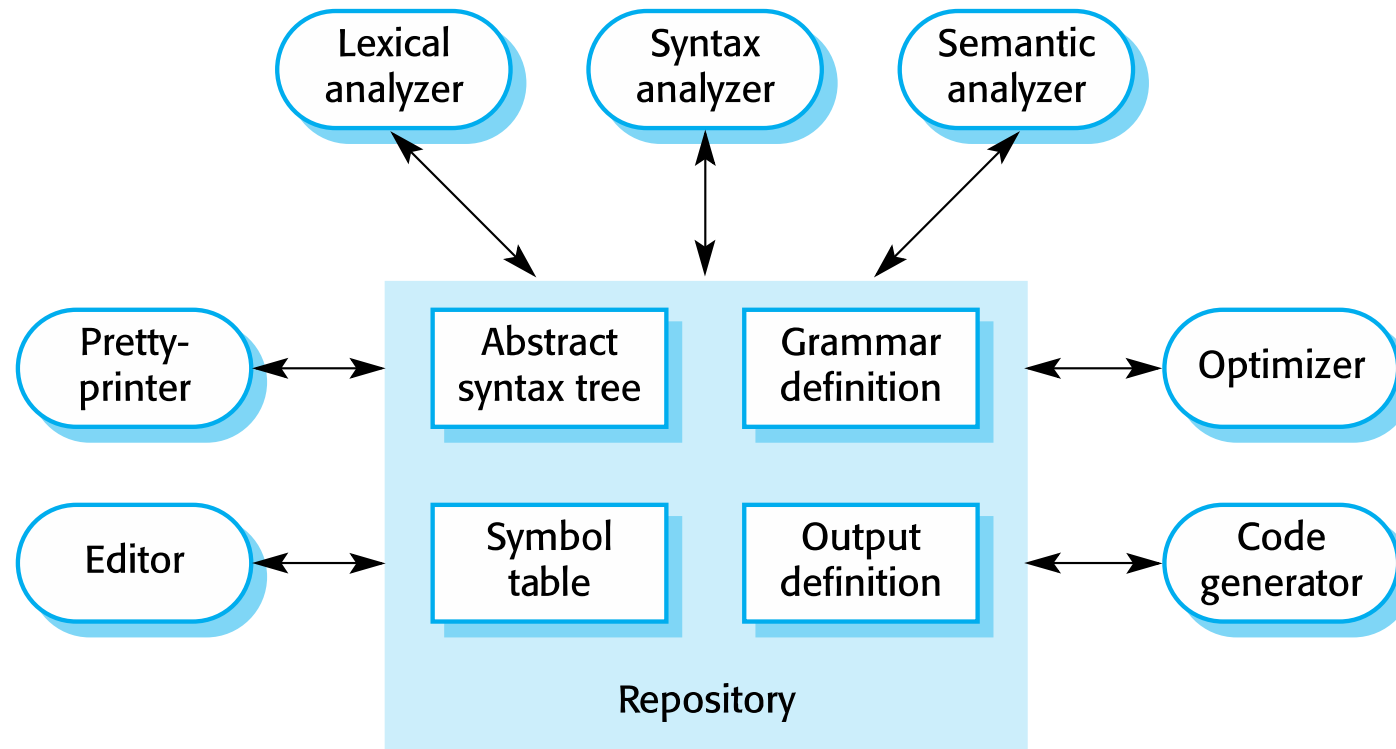
A PIPE AND FILTER COMPILER ARCHITECTURE

That is all