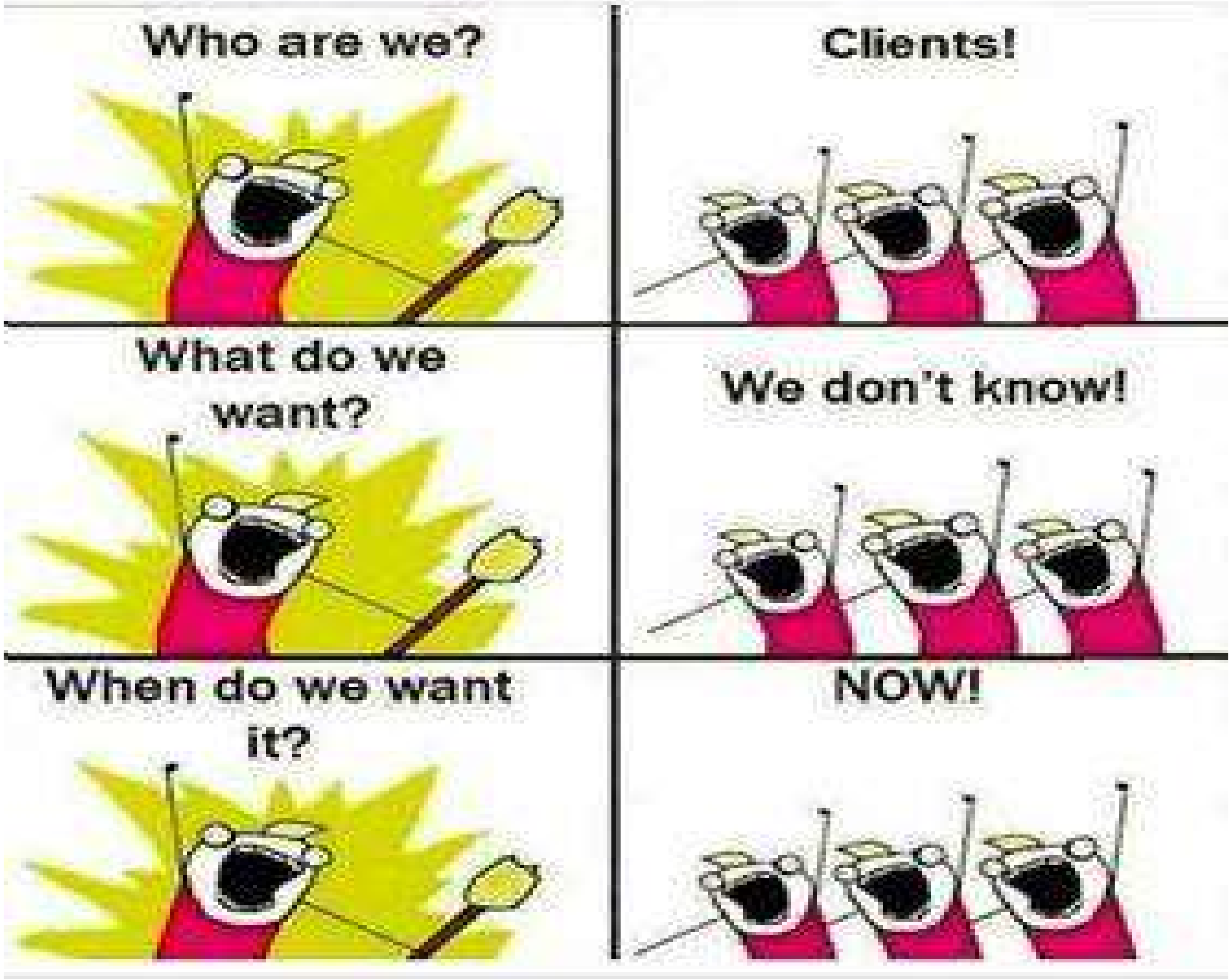
Lecture IX:

Architectural Design II

General Expectations



Topics Covered

- Repository Architecture
- Client Server Architecture
- Pipe and Filter Architecture
- Application Architectures

Repository Architecture

Repository Architecture

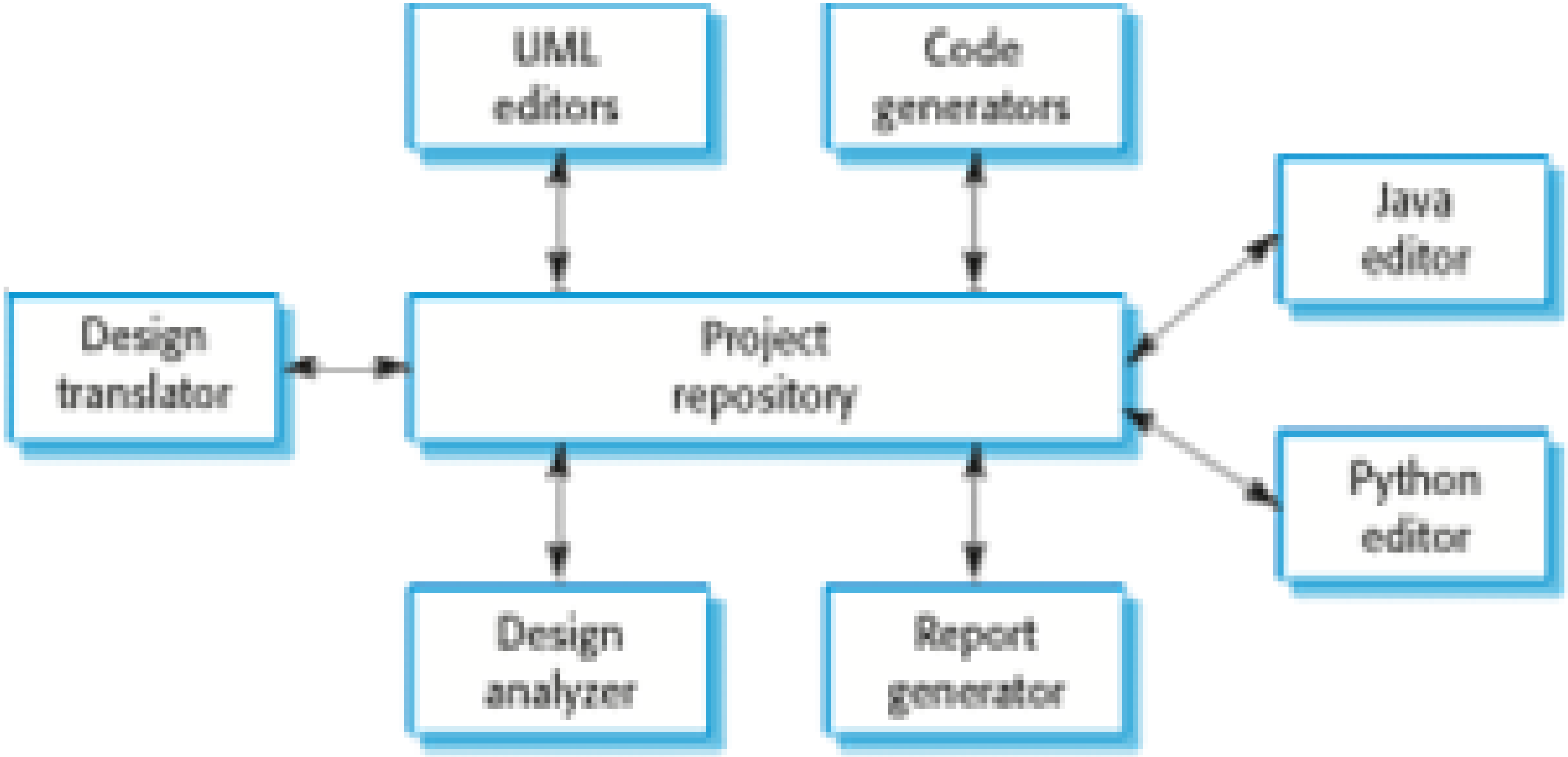
- **Sub-systems must exchange data**. This may be done in two ways:
  - **Shared data** is held in a **central database** or **repository** and may be ***accessed by all sub-systems***;
  - **Each sub-system** maintains its **own database** and passes data explicitly to other sub-systems.

- 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.

Repository Pattern

Name	Repository Pattern
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	• 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	• When you have a system in which large volumes of information are generated that has to be stored for a long time. • 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

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.

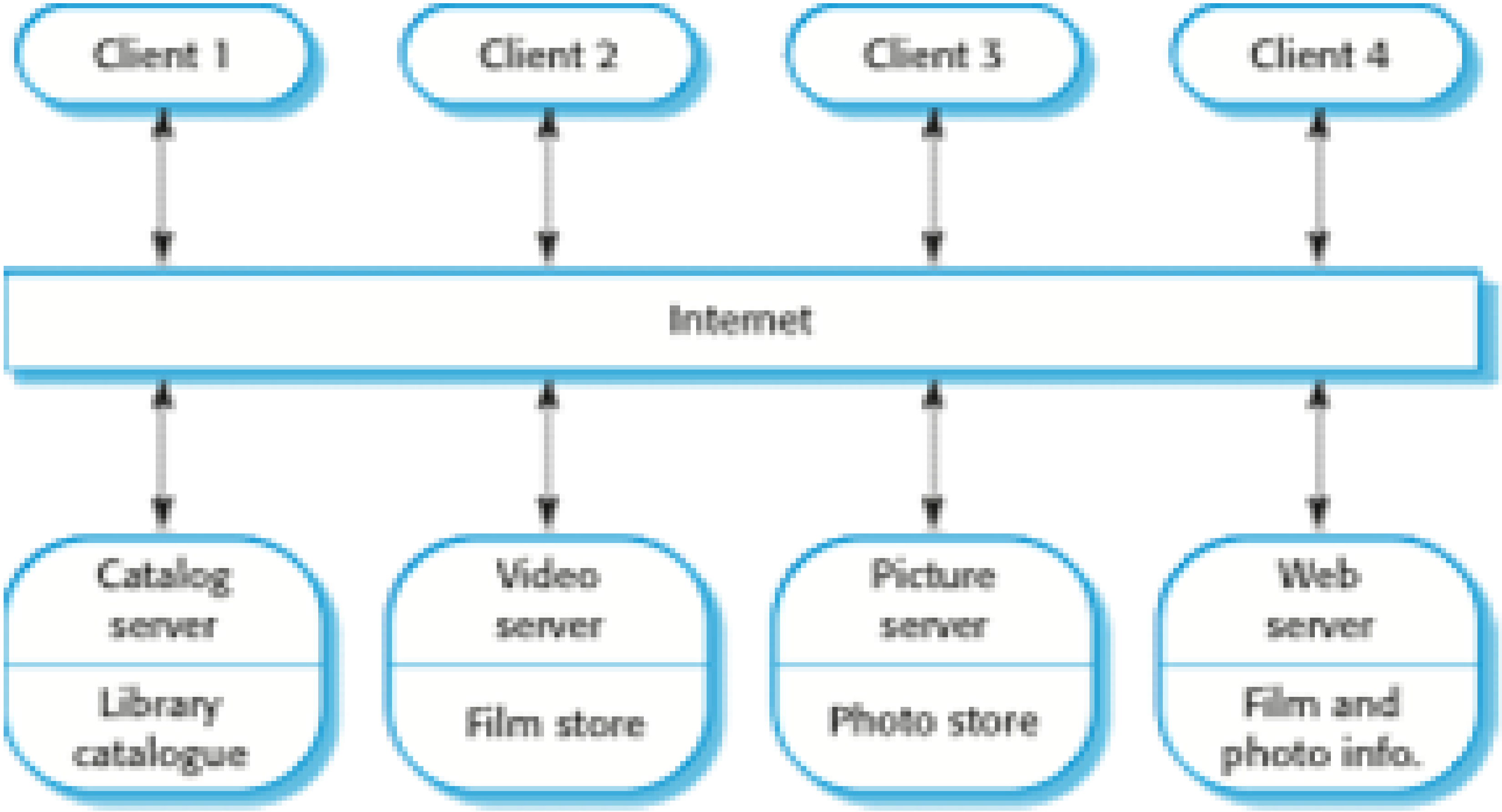
- **Set of clients** which **call** on these services.

- **Network** which **allows** clients to access servers.

Client-Server Pattern

Name	**Client-server Pattern**
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	*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



Pipe and Filter Architecture

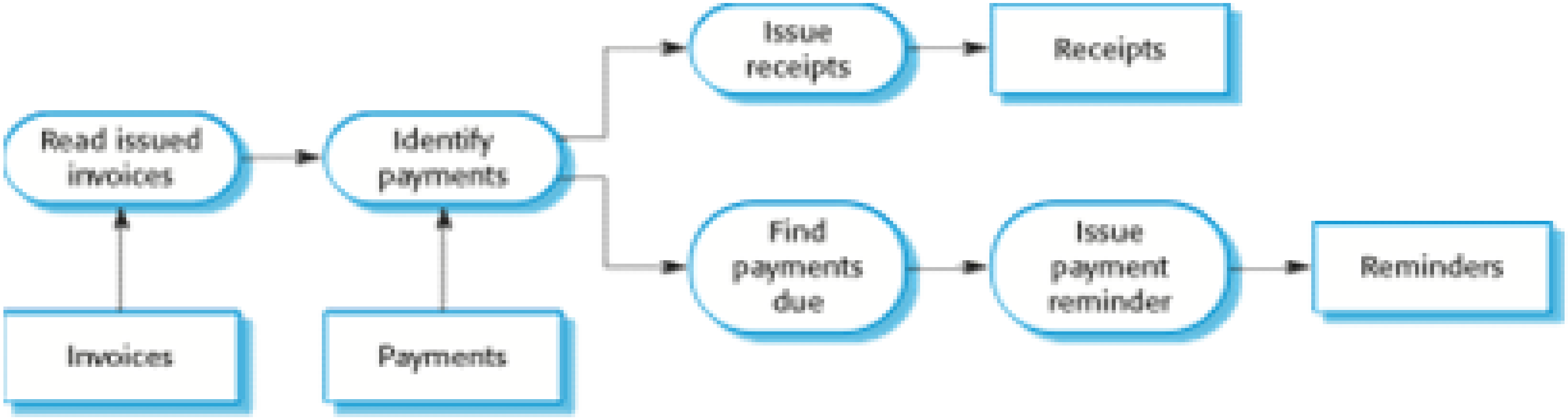
Pipe and Filter Architecture

- **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**.

Pipe and Filter Pattern

Name	Pipe and filter Pattern
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	*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



Application Architectures

Application Architectures

- **Application systems** are designed to **meet** an **organizational 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 **organizing** 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-centered 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.

Examples of Application Types

- We pick on transaction processing and language processing systems.

- And two types of each....

- **Transaction processing systems**
  - E-commerce systems;
  - Reservation systems.

- **Language processing systems**
  - Compilers;
  - Command interpreters.

Transaction Processing Systems

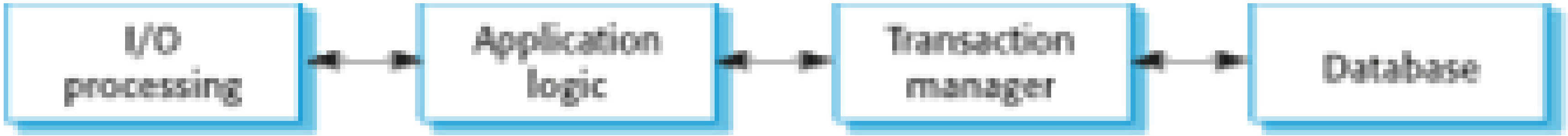
Transaction Processing Systems

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

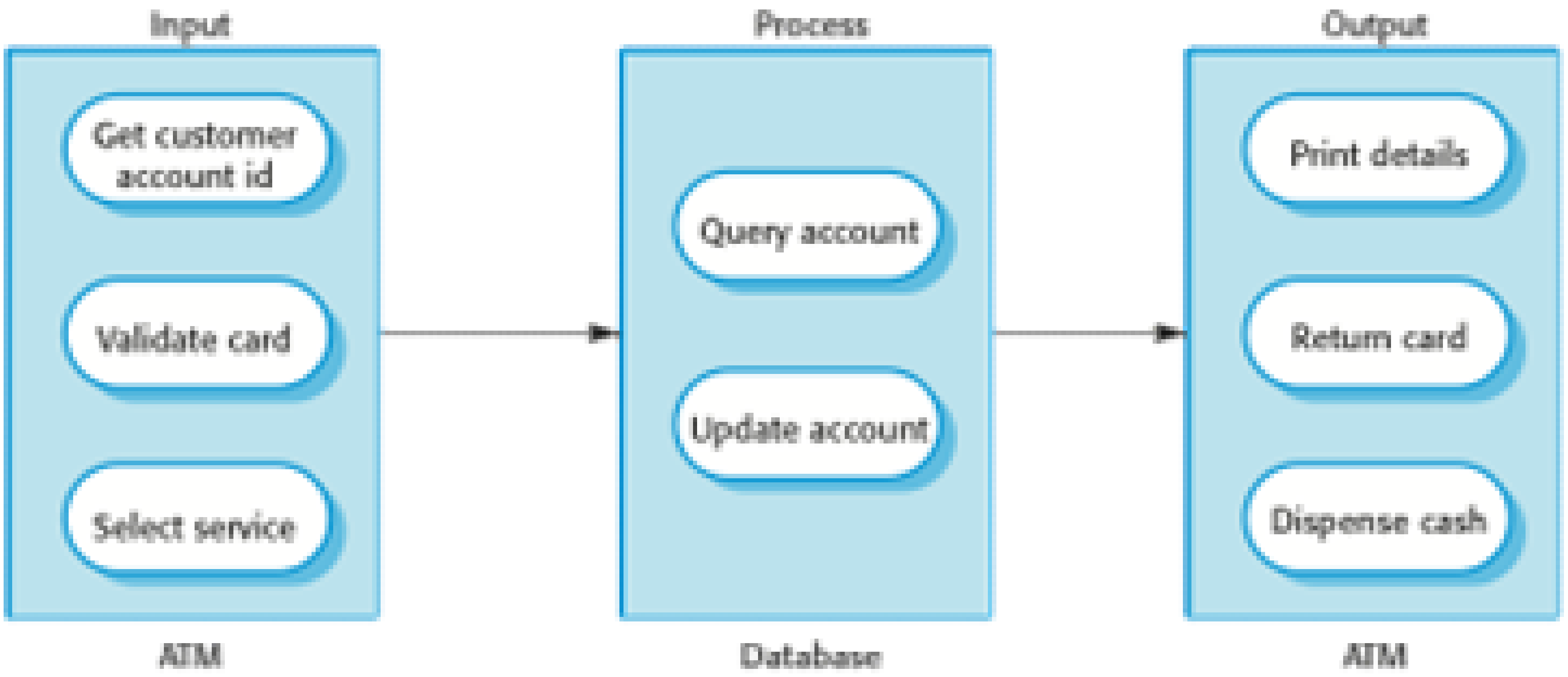
- From a **user perspective** a **transaction** is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from Nairobi to Paris.

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

The Structure of Transaction Processing Applications



The Software Architecture of an ATM System



Information Systems Architecture

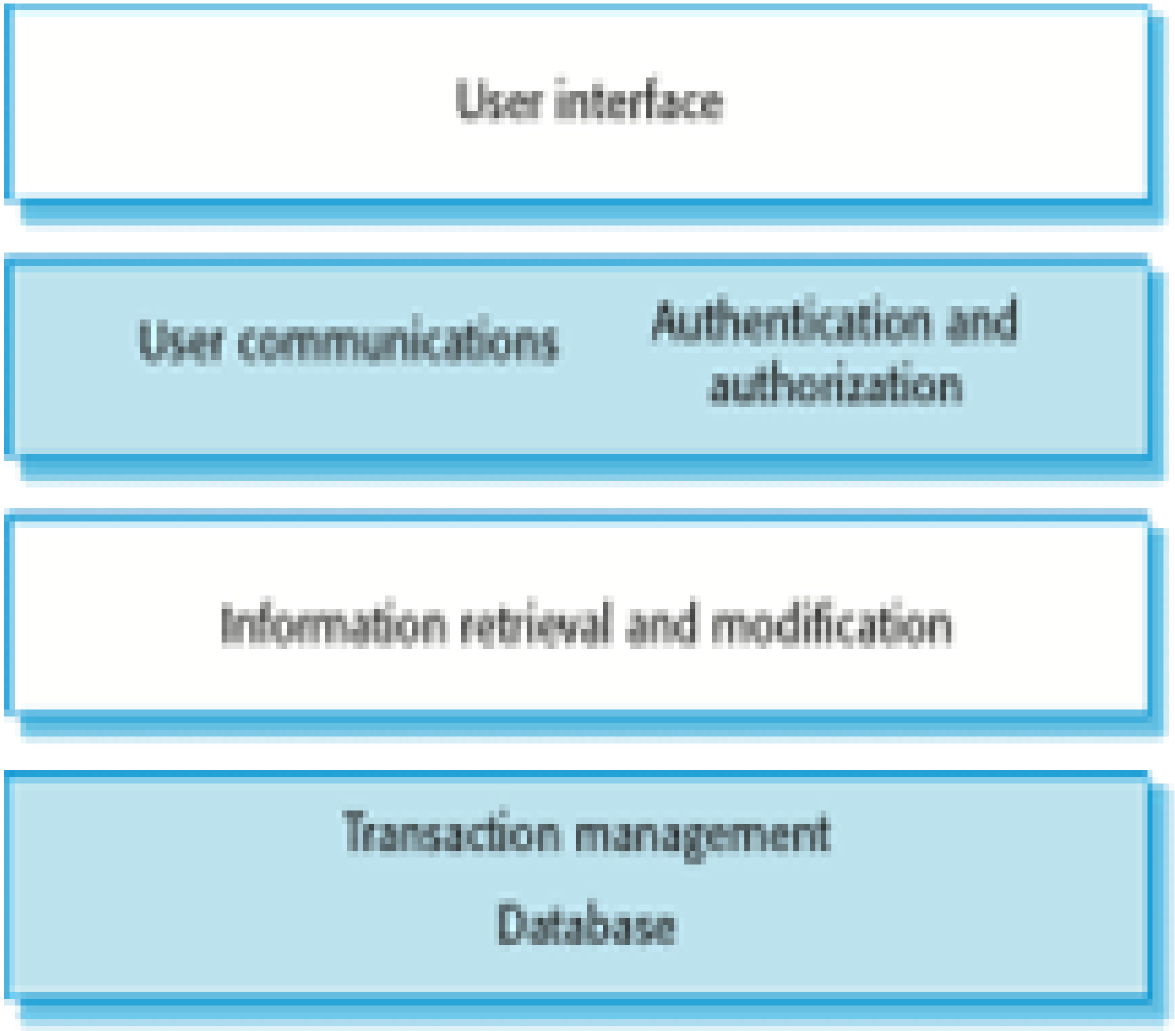
Information Systems Architecture

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

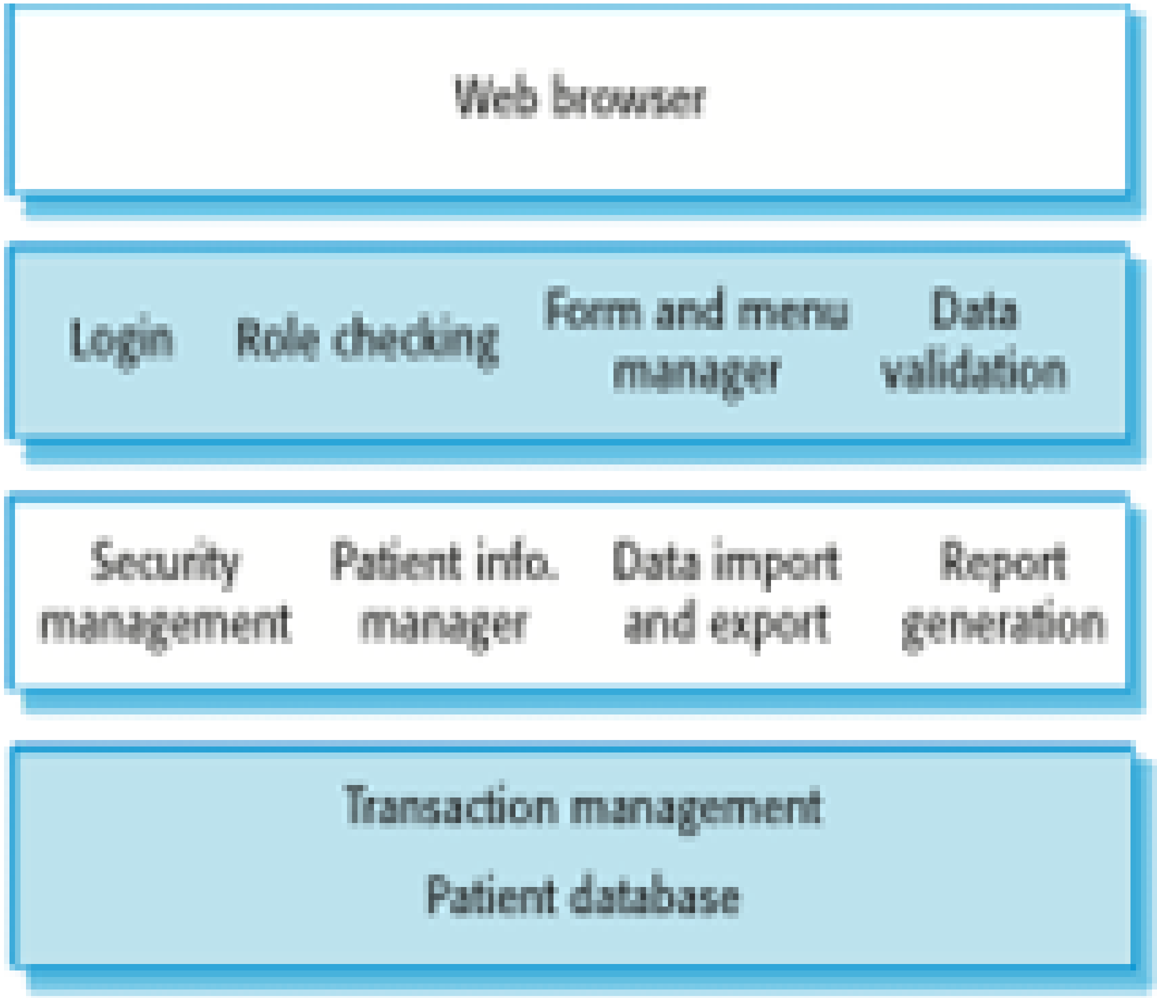
- These are **transaction-based systems** as **interaction** with these systems generally involves **database transactions**.

- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database

Layered Information Systems Architecture


User interface
User communications
Authentication and authorization
Information retrieval and modification
Transaction management
Database

The Architecture of the MoH-PMS



Language Processing Systems

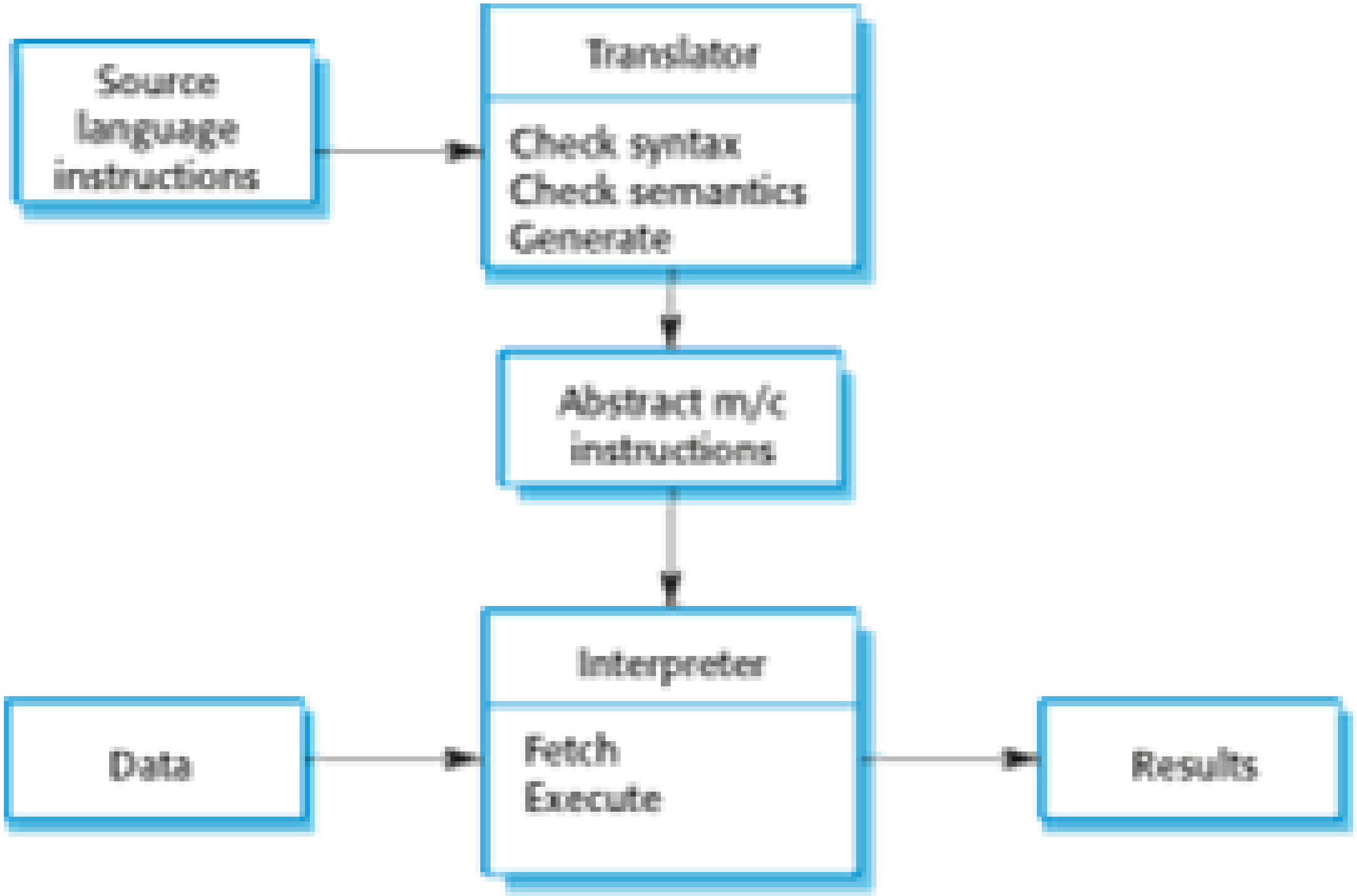
Language Processing Systems

- **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 the **system data**
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

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.

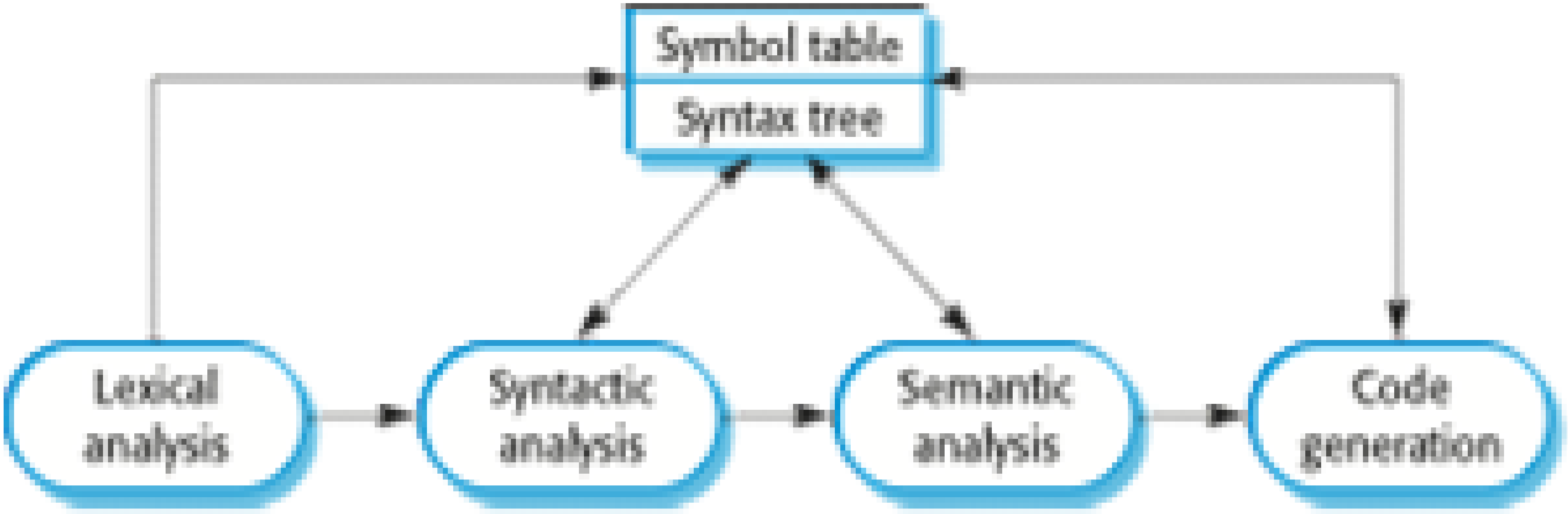
- A **syntax tree**, which is an **internal structure** representing the program being compiled.

Compiler Components (2)

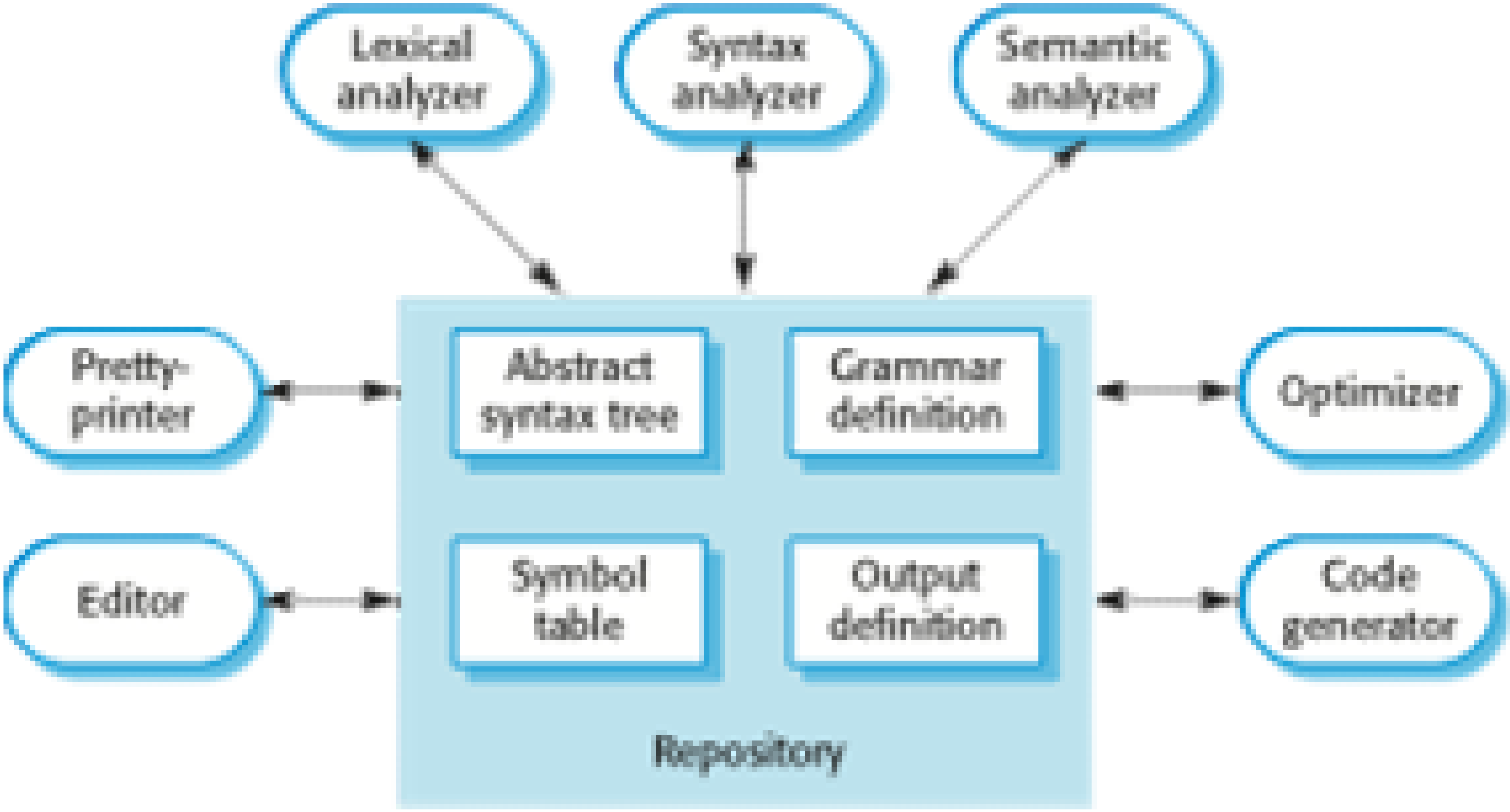
- A **semantic analyzer** that **uses** information from the **syntax tree** and **the symbol table** to **check** the **semantic correctness** of the **input language** text.

- A **code generator** that ‘**walks**’ the syntax tree and **generates abstract machine code**.

A Pipe and Filter Compiler Architecture



A Repository Architecture for a Language Processing System


Lexical analyzer
Syntax analyzer
Semantic analyzer
Pretty-printer
Abstract syntax tree
Grammar definition
Optimizer
Editor
Symbol table
Output definition
Code generator
Repository