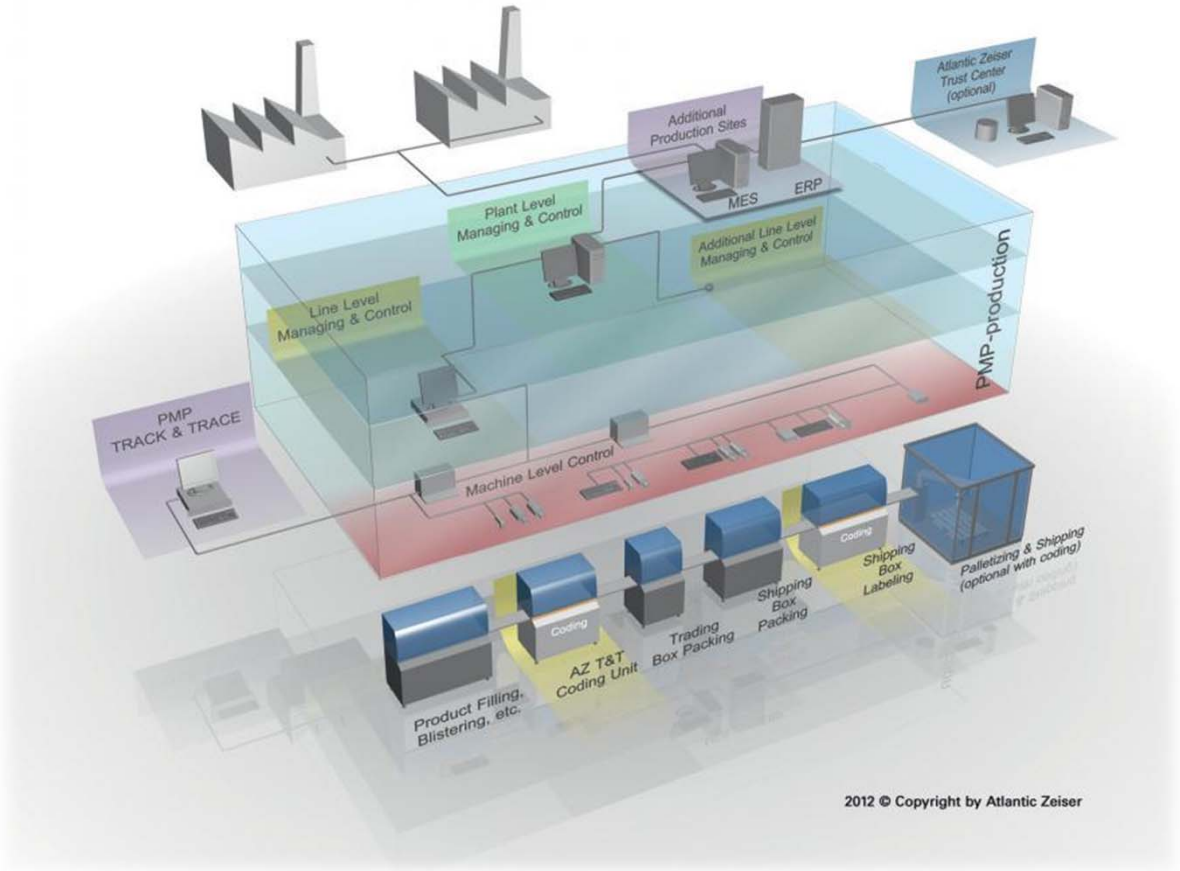




# Chapter 6- Architectural Design

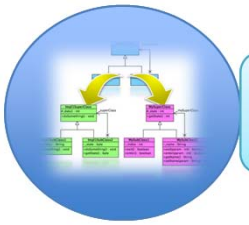



Lecture 6

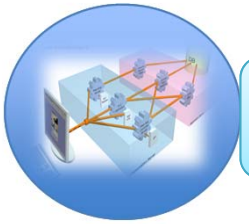
# Topics covered


Architectural Design Decision

Architectural Views

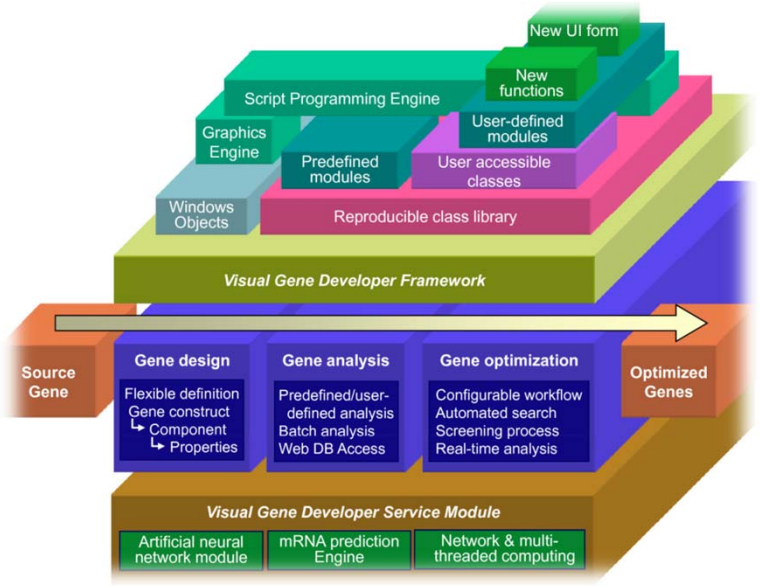


Architectural Patterns

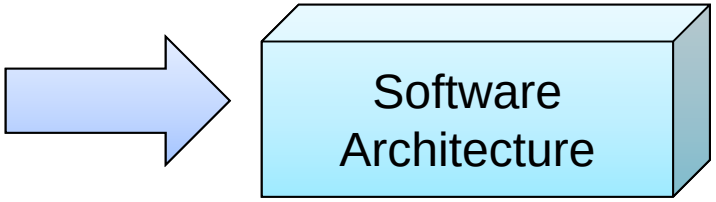


Application Architectures

# Software Architecture


Design Process




Output

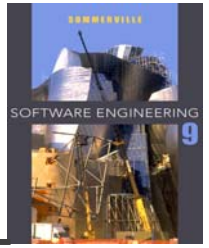
## Architectural Design

The design process for identifying sub-systems to make a system and framework for sub-systems control and communication

## Output

The output of this design process is the description of the software architecture

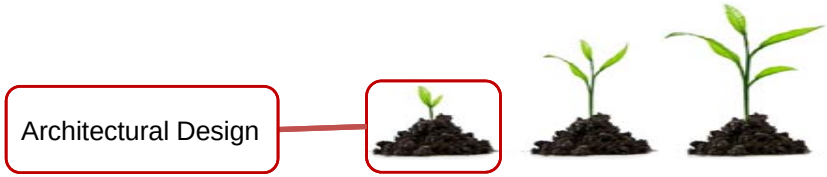
# Architectural Design




1

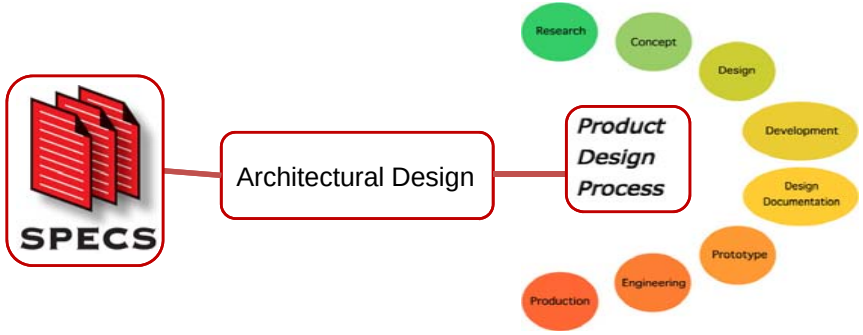
Architectural Design is the early stage of the system design process

System Design Process




2

Architectural Design represents the link between specification and design process



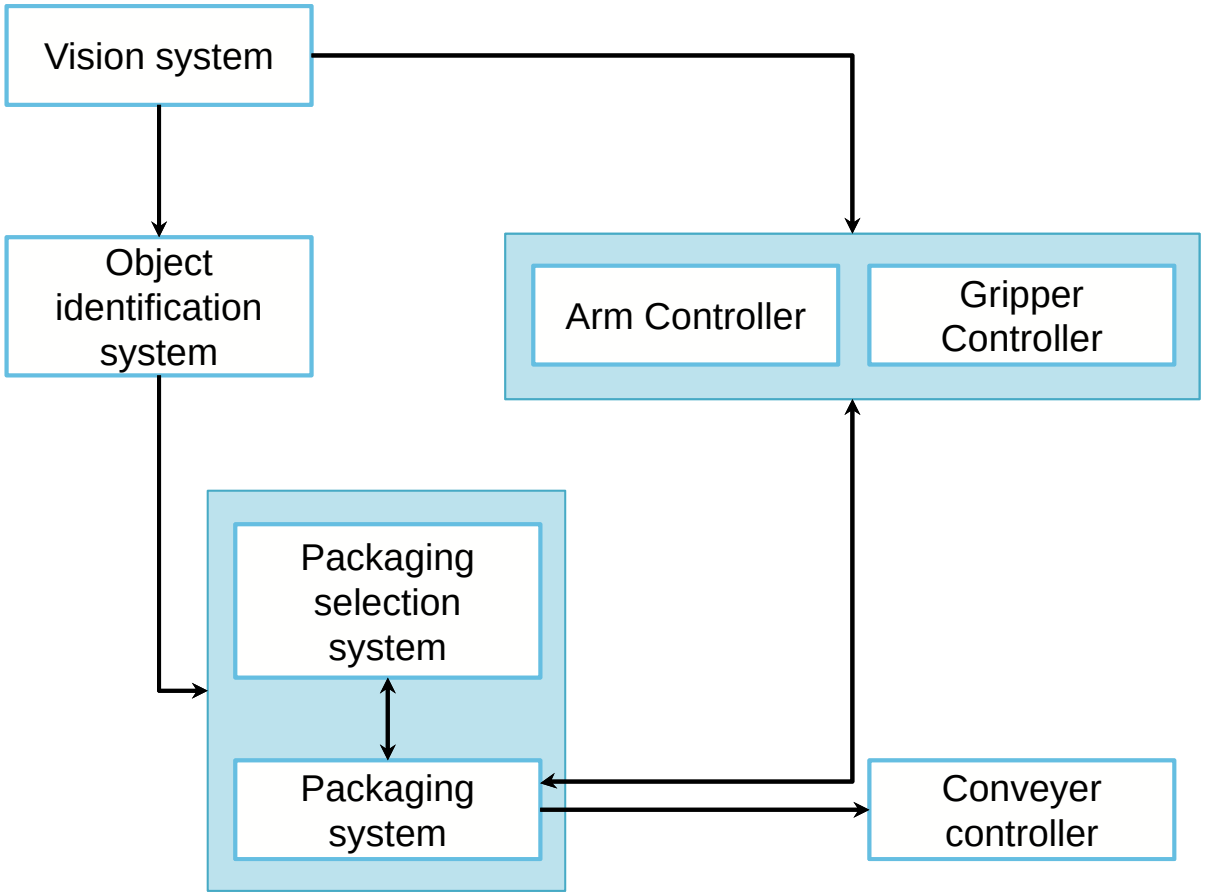

3

Often carried out in parallel with some specification activities

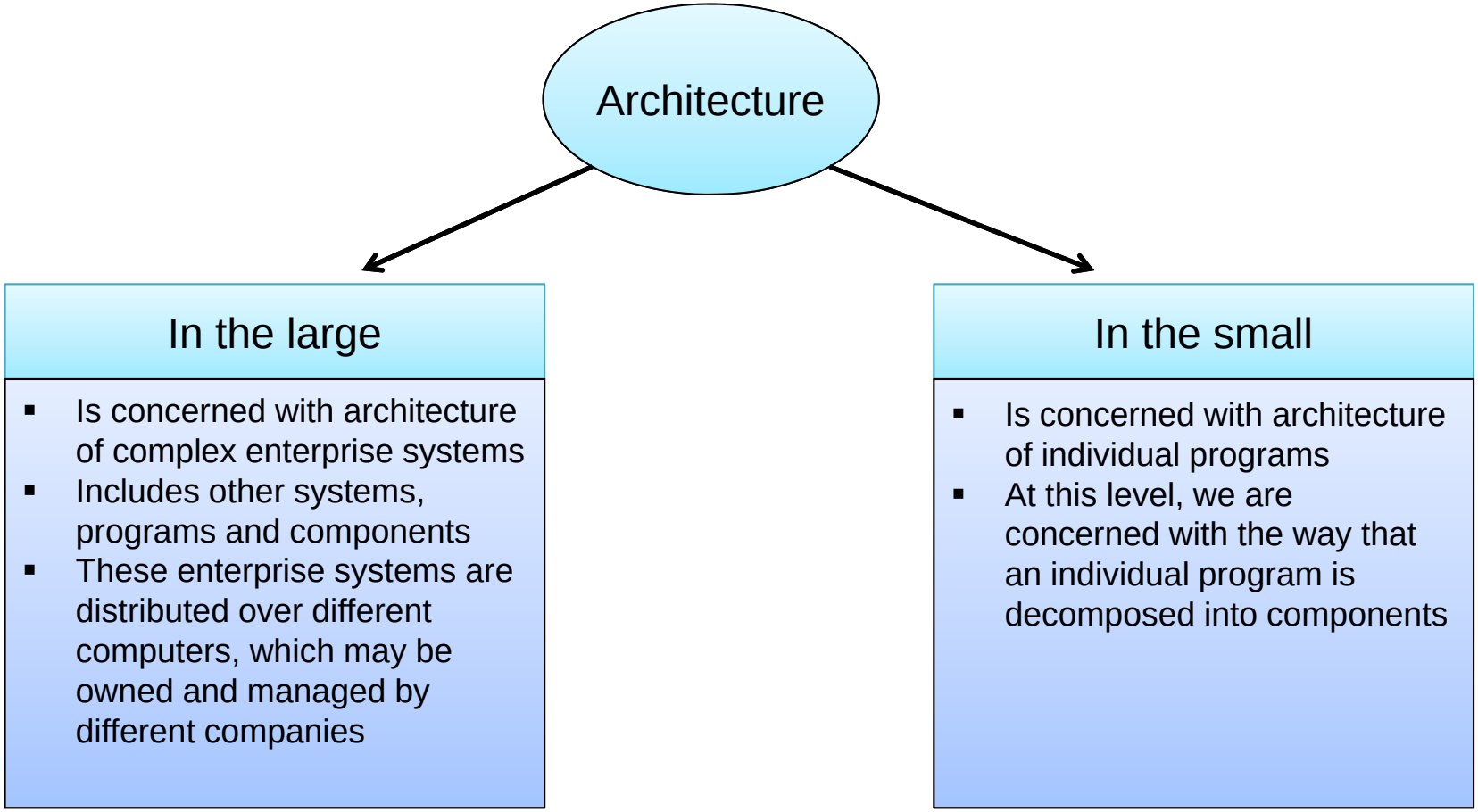
4

It involves Identifying major system components and their communications

# The architecture of a packing robot control system

# Architectural abstraction

# Advantages of explicit architecture


## Stakeholder Communication

- Architecture may be used as a focus of discussion by system stakeholders

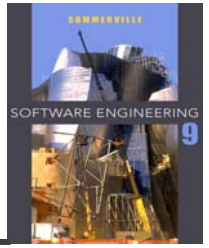
## System Analysis

- Means that analysis of whether the system can meet its non-functional requirements is possible

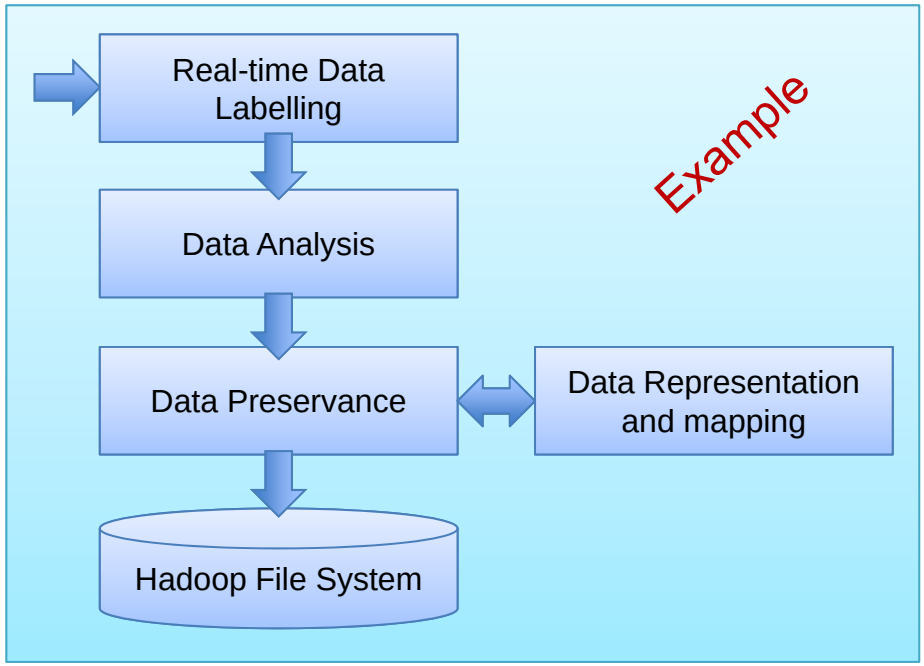

## Large-scale reuse

- The architecture may be reusable across a range of systems
- Product-line architectures may be developed

# Architectural representations




**Block Diagram for Data curation layer**




## Advantages

- It is simple to understand
- Informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures

## Limitations

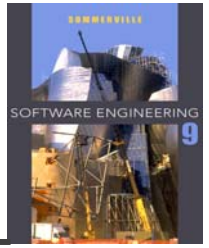
- But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture

## Dependency

- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used

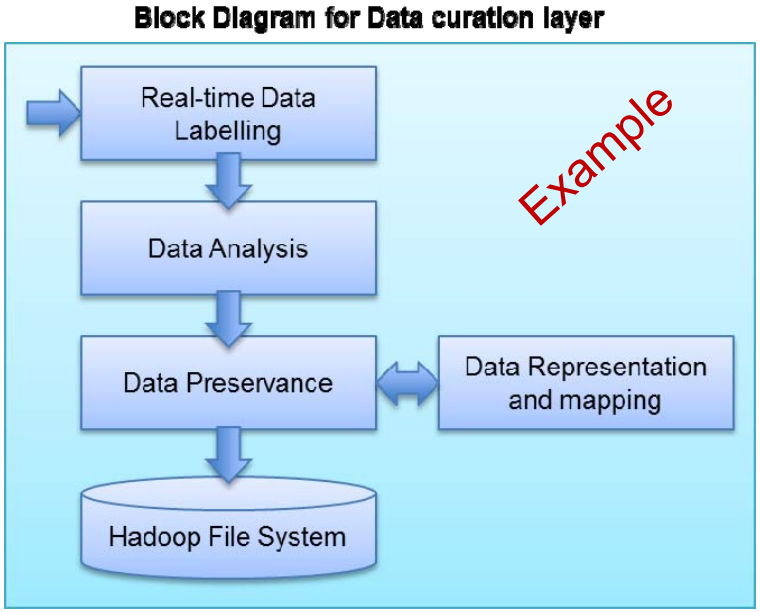
Box and line diagram is same as block diagram

# Box and line diagrams




## Definition and Abstract

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems
- However, useful for communication with stakeholders and for project planning.



# Use of architectural models




## Facilitating discussion

- As a way of facilitating discussion about the system design
  - A high-level Architectural view of a system is useful for communication with stakeholders and project planning because it is not cluttered with detail
  - Stakeholders can relate to it and understand an abstract view of the system. They can then discuss the system as a whole without being confused by detail.

## Facilitating documentation

- As a way of documenting an architecture that has been designed
  - The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections.

# Architectural design decisions

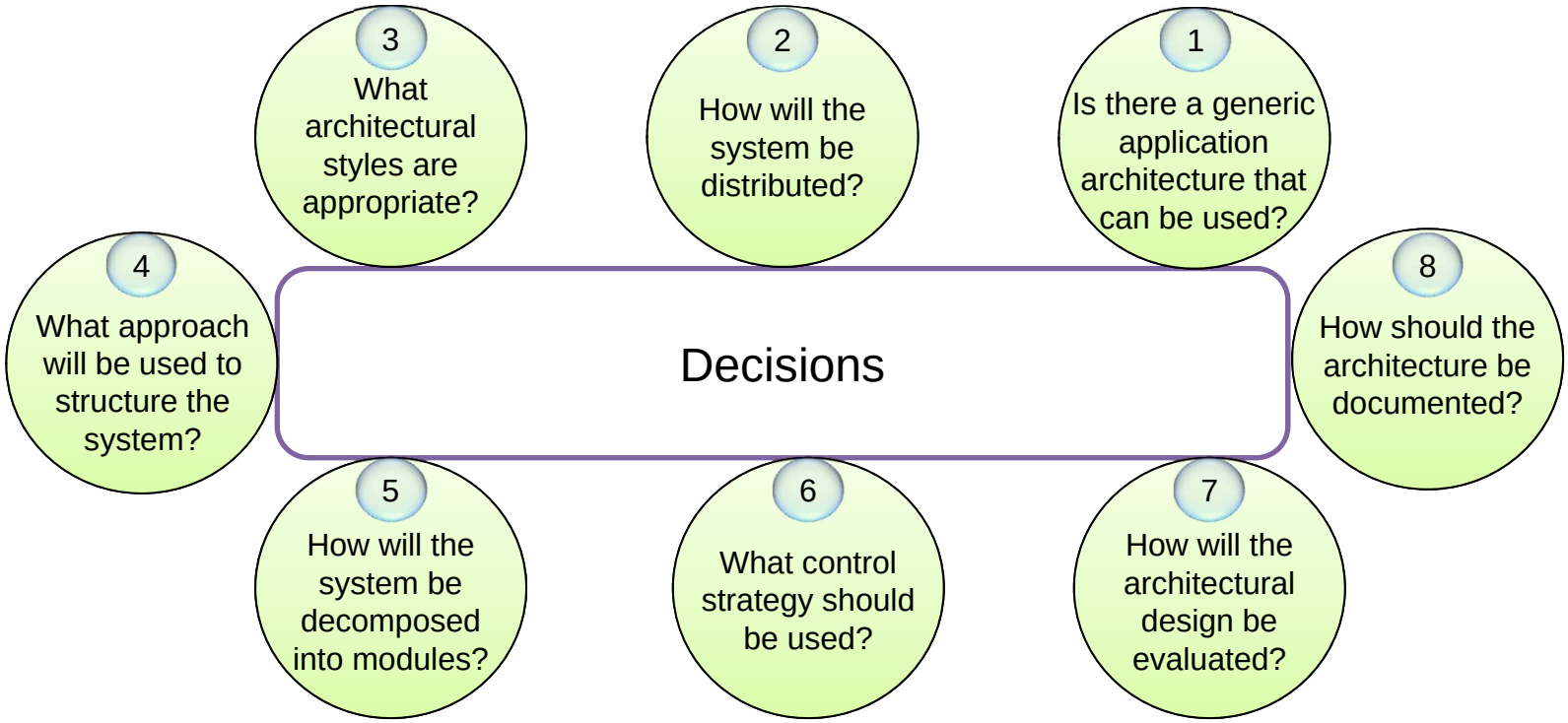

## Creative Process

- Architectural design is a creative process so the process differs depending on the type of system being developed

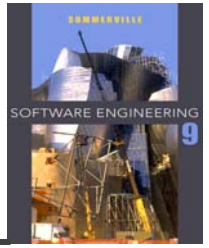
## Design Decision

- A number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system

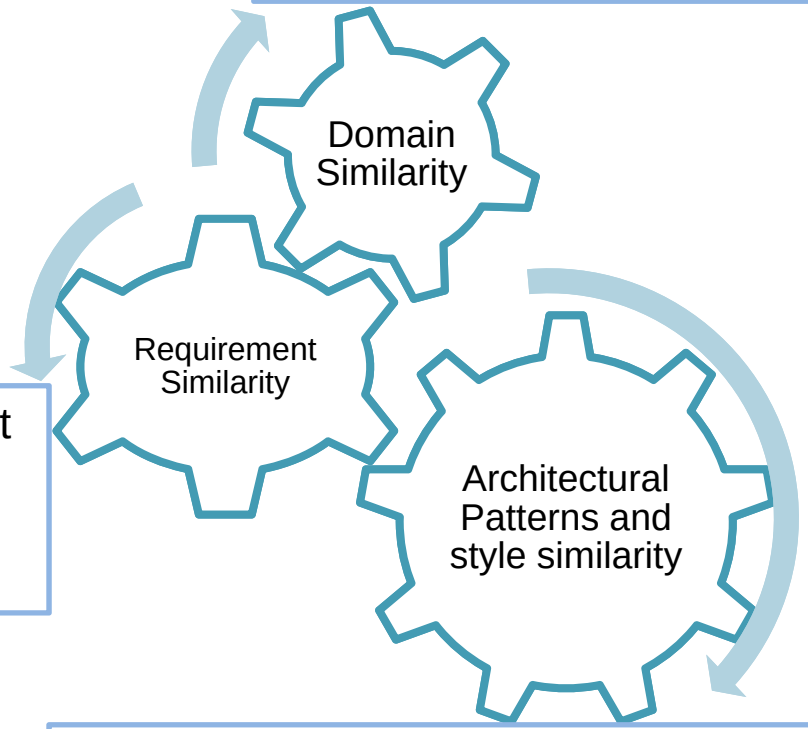
# Architectural design decisions

# Architecture reuse




Systems in same domain have similar architectures that reflect domain concepts.




Application product lines are built around a core architecture with variants that satisfy particular customer requirements

- The architecture of a system may be designed around one of more architectural patterns or 'styles'
  - These capture the essence of an architecture and can be instantiated in different ways

# Architecture and system characteristics


System Characteristics

**Performance**

- Localize critical operations and minimize communications. Use large rather than fine-grain components


**Security**

- Use a layered architecture with critical assets in the inner layers


**Safety**

- Localize Safety-critical features in a small number of sub-systems


**Availability**

- Include redundant components and mechanisms for fault tolerance

**Maintainability**

- Use fine-grain, replaceable components

# Architectural views

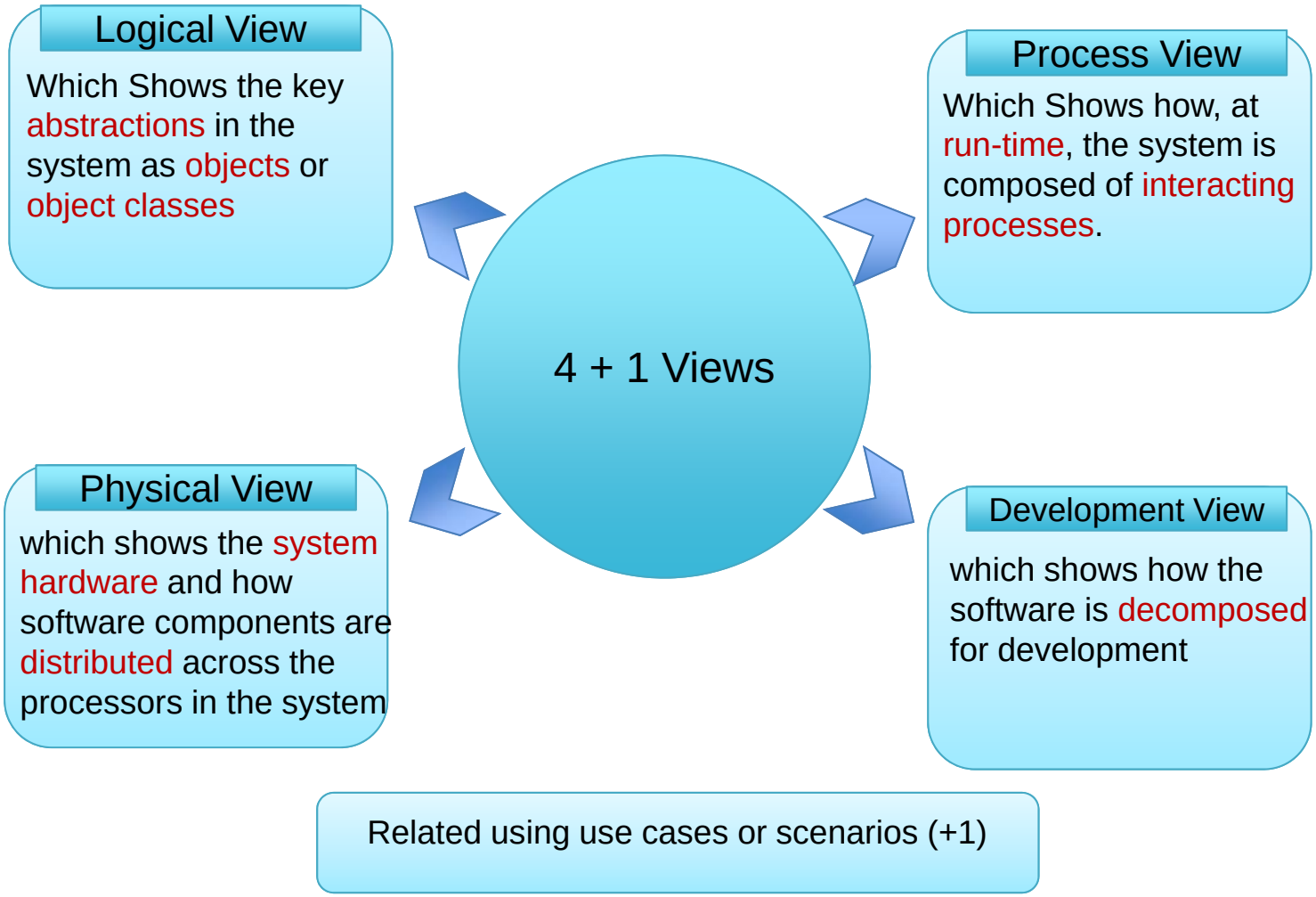

**Views and Prospective**

- What views or perspectives are useful when designing and documenting a system's architecture.
- What notations should be used for describing architectural models?

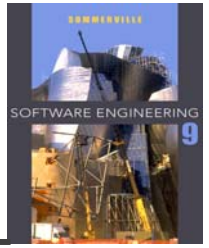
**Single View**

- Each architectural model only shows a single view or prospective of the system
  - It might show how a system is decomposed into modules,
  - how the run-time processes interact or the different ways in which system components are distributed across a network.
  - For both design and documentation, you usually need to present multiple views of the software architecture

# 4 + 1 view model of software architecture

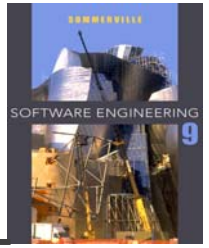



# Architectural patterns




| Architectural Patterns | | | |
|---|---|---|---|
| Definition | Stylized Description | Include Information | Representation |
| Patterns are a means of<br>▪ representing<br>▪ sharing<br>▪ reusing<br>of knowledge | 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 | Patterns may be represented using tabular and graphical descriptions. |

# The Model-View-Controller (MVC) pattern

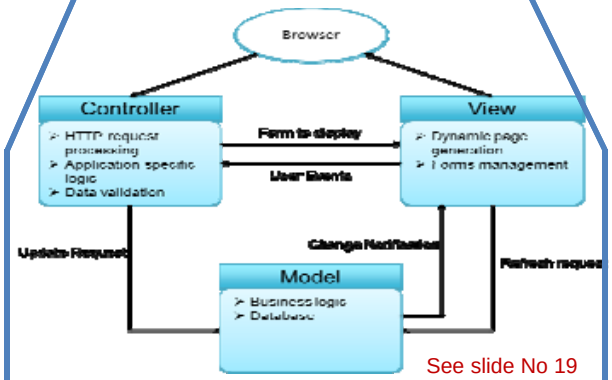



## Description

Separates presentation and interaction from system data.

***Model Component:*** Manages system data and associated operation on data.
***View Component:*** defines and manages how the data is presented to the user.
***Controller Component***: manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model

## Example




See slide No 19

## When Used?

When: There are multiple ways to view and interact with data.

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.

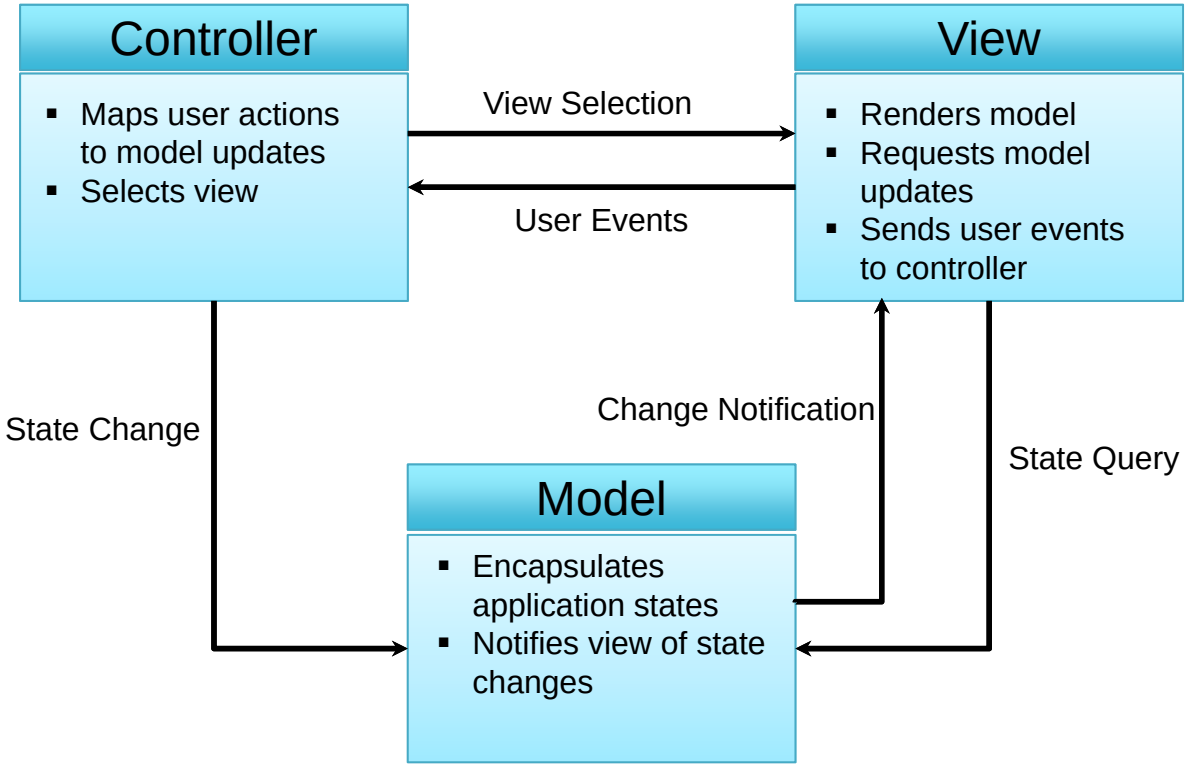
## Disadvantages

- Can involve additional code
- Code complexity increase when the data model and interactions are simple

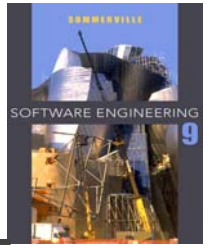
# MVC Architecture overview (video)

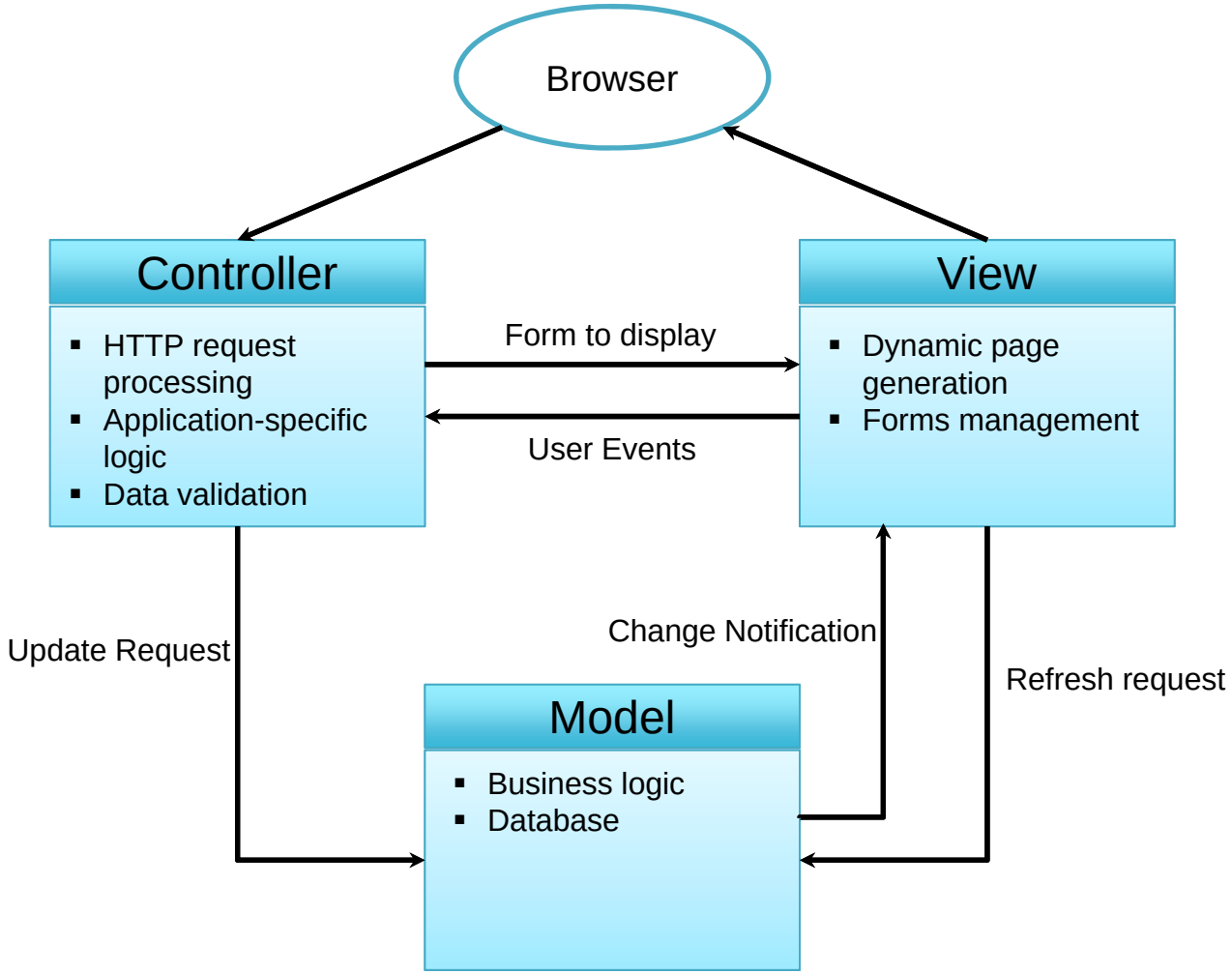

http://www.youtube.com/watch?v=3mQjtk2YDkM

# The organization of the Model-View-Controller

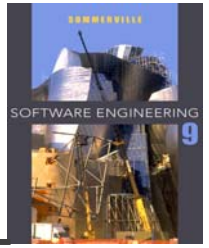



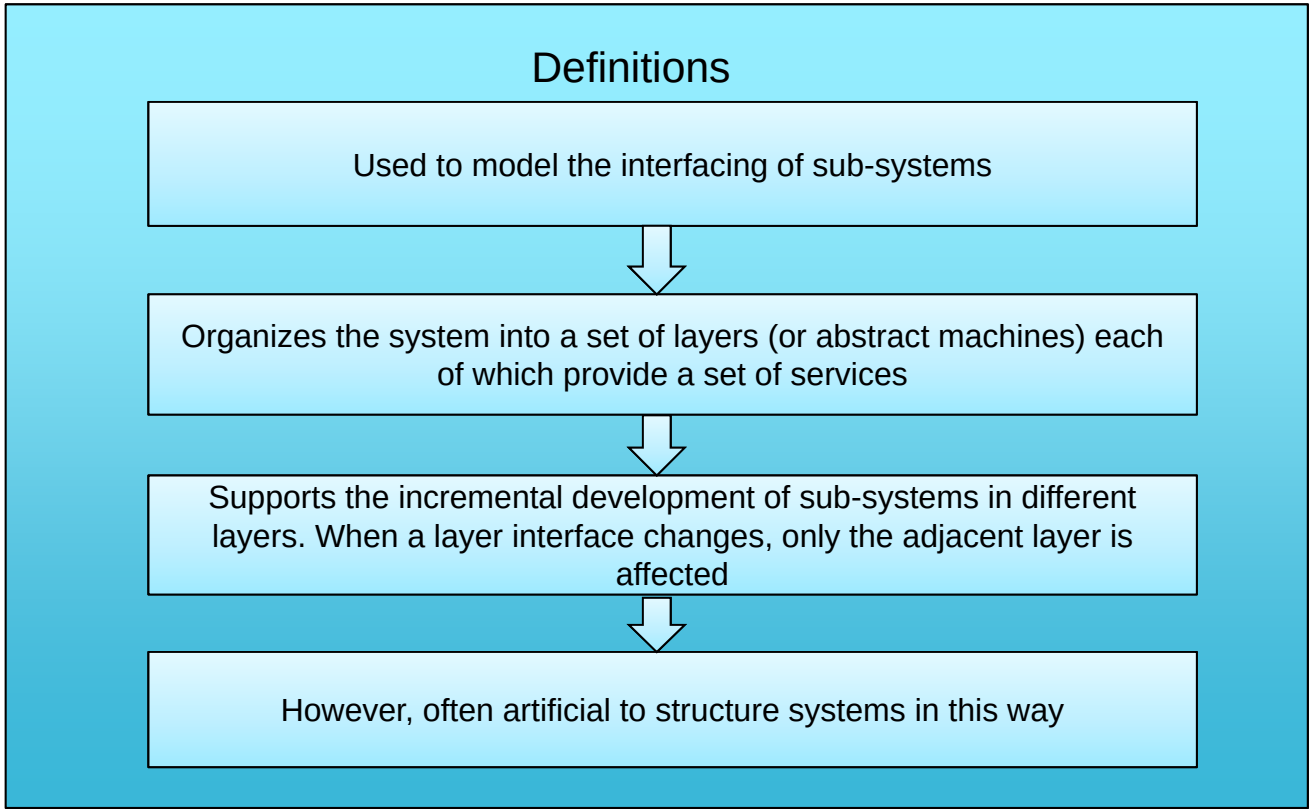
# Web application architecture using the MVC pattern



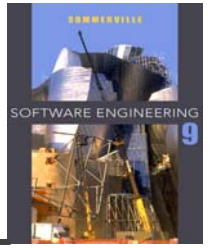



# Layered architecture



# The Layered architecture pattern



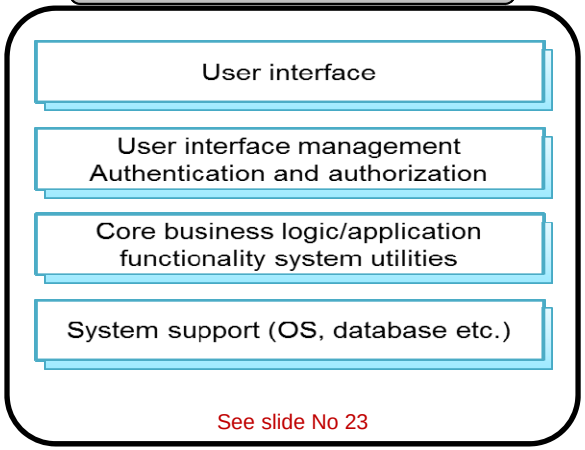

## Description

- Organizes the system into layers with related functionality
- Each layer provides services to layer above it.
- The services provided by the lowest layer are considered as core services
- The core services are likely to be used through out the system

## When Used?

When: Building the new facilities on top of existing systems.
When: Development spread across several teams, each team is responsible for a layer of functionality
When: There is requirement for multi-level security

## Example




See slide No 23

## Advantages

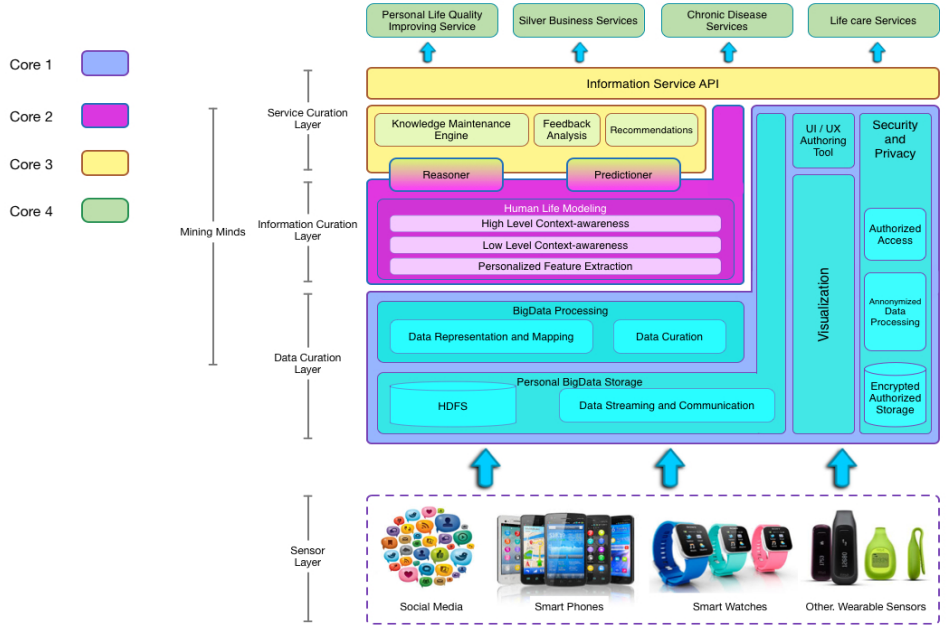
- Allows replacement of entire layers so long as the interfaces are maintained.
- Redundant facilities (e.g. Authentication) can be provided in each layer to increase the dependency of the system

## 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.
- Multiple level of interpretation of a service request can decrease the performance

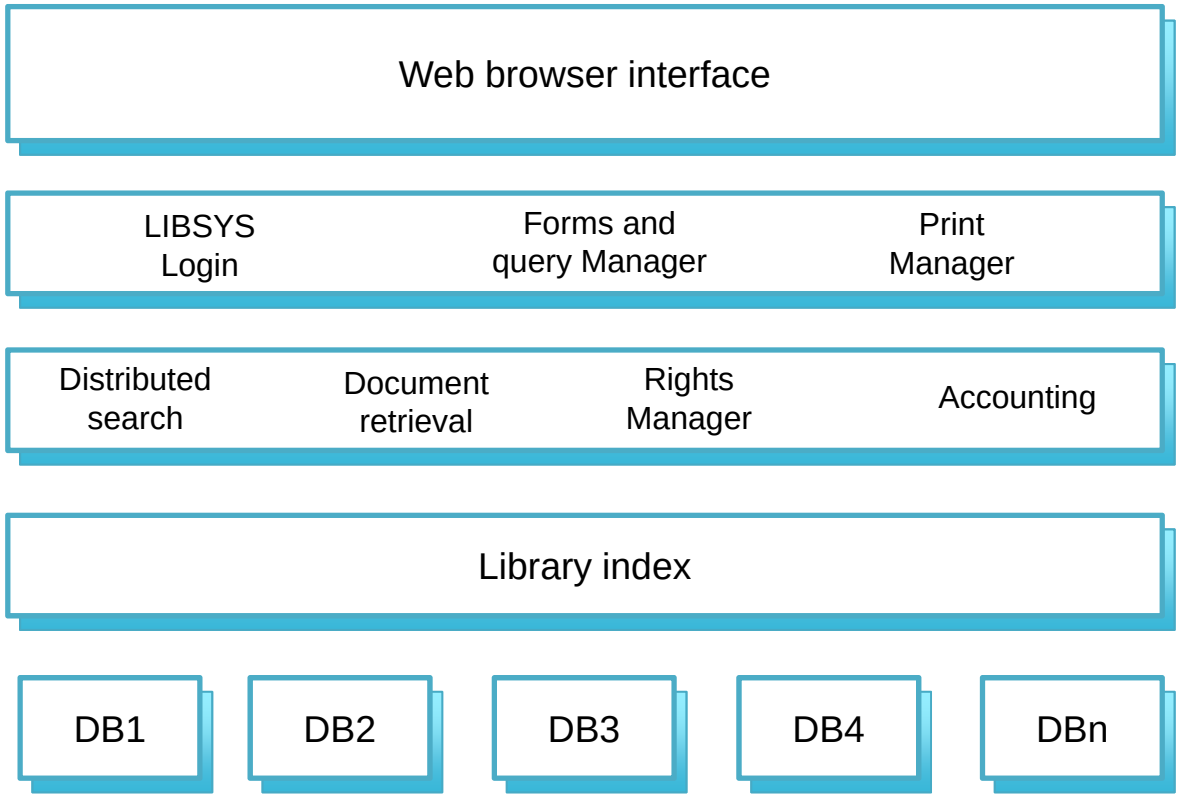
# The generic layered architecture and Mining Minds architecture


- User interface
- User interface management
  Authentication and authorization
- Core business logic/application functionality system utilities
- System support (OS, database etc.)




http://www.miningminds.re.kr/technical-report/presentations/

# The architecture of the LIBSYS system

# Key points



- A Software architecture is a description of how a software system is organized

Architectural design decisions include decisions on
- Type of application
- The distribution of the system
- The architectural styles to be used

Architectures may be documented from several different perspectives or views such as
- A conceptual view
- A logical view
- A process view
- A development view

Architectural patterns are means of reusing knowledge about generic system architectures
- They describe the architecture
- Explanation when it may be used
- Describe advantages and disadvantages

# Software Architecture Conclusion (video)


## What is Software Architecture?

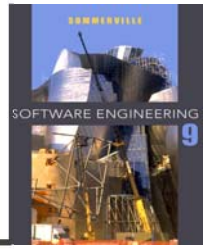
**George Fairbanks**

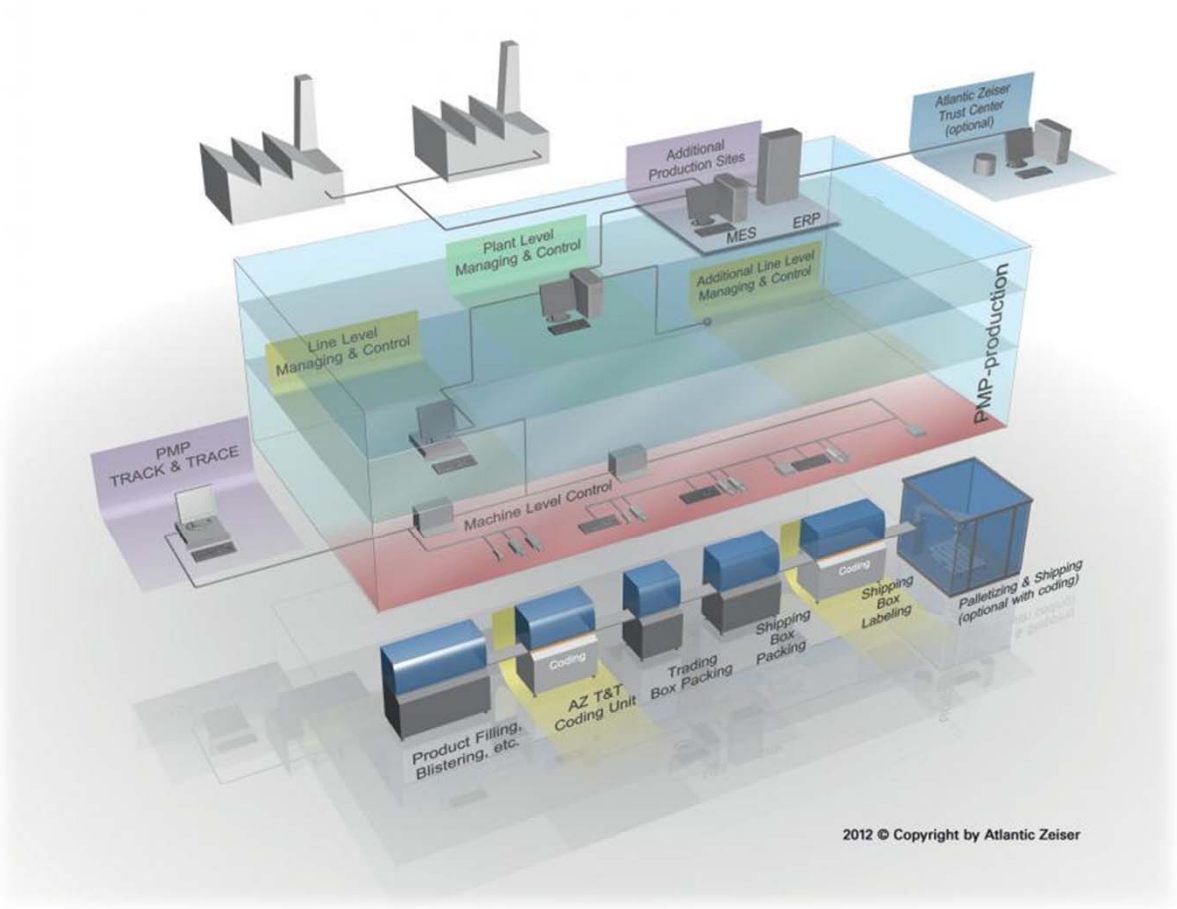
1 January 2013

**Rhino Research**
Software Architecture Consulting and Training
http://RhinoResearch.com

http://www.youtube.com/watch?v=Rn1g6V-vlHw

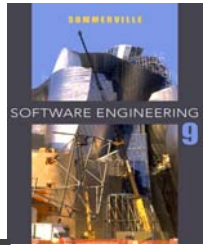

# Chapter 6 – Architectural Design




Lecture 2

# Repository architecture

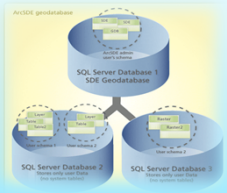



Sub-systems must exchange data (Two Ways)

## Central Repository

- Shared data is held in central database or repository
- May be accessed by all sub-systems

## Multiple Repositories

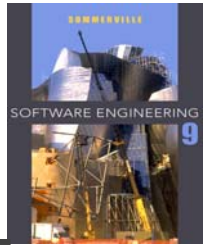


- Each sub-system maintains its own database
- Passes data explicitly to others sub-systems

## Repository Model


- When large amount of data are to be shared, a Repository Model is most commonly used
- This is an efficient data sharing mechanism

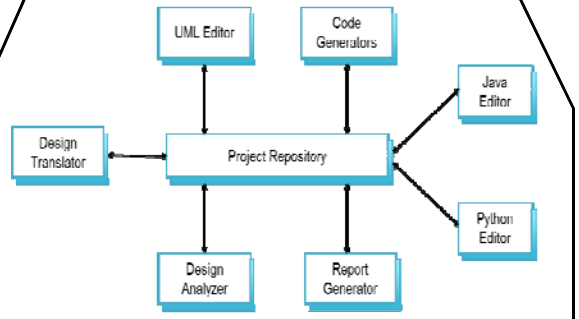
# The Repository pattern




## Description

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

## Example




See slide No 30

## When Used?

When: You have a system with large volume of information are generated

When: You want to store data for long time.

When: there is need some data-driven systems where the inclusion of data in the repository triggers an action or tool

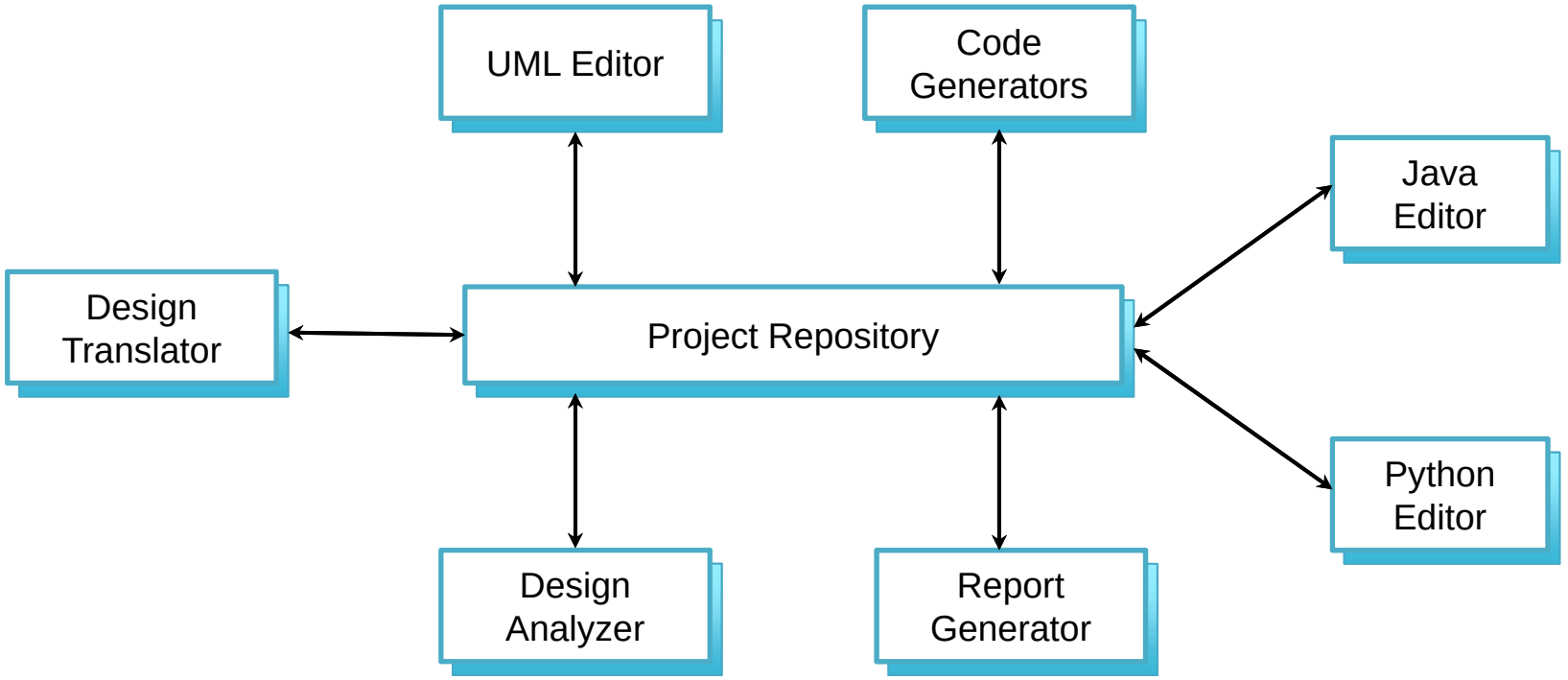
## Advantages

- Components can be independent.
- No 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

## 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.
- Distributing repository through several systems is difficult

# A repository architecture for an IDE

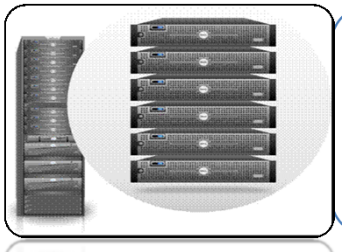



# Client-server architecture



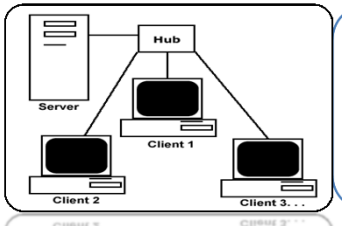
- Distributed System model which shows how data and processing is distributed across a range of components.
- Can be implemented on a single computer

=



- Set of stand-alone **servers** which provide specific services
  - Printing
  - Data management

+




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

+

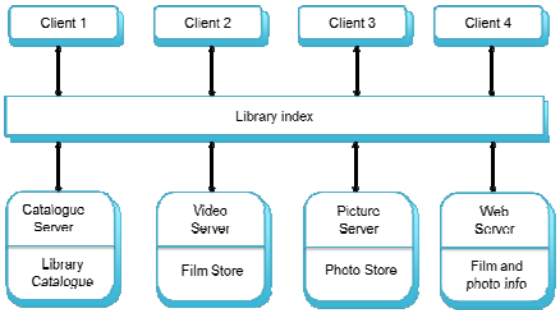
- **Network** which allows clients to access servers

# The Client–server pattern


## Description

- In a client-server architecture, the functionality of the system organized into services
- Each service delivered from a separate server
- Clients are users of these services and access servers to make use of them

## Example




See slide No 33

## When Used?

When: Data in a shared database has to be accessed from a range of locations.
When: Servers want to replicated the data
When: Load on the system is variable

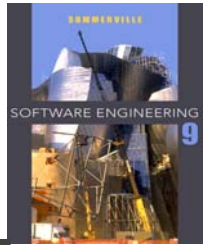
## Advantages

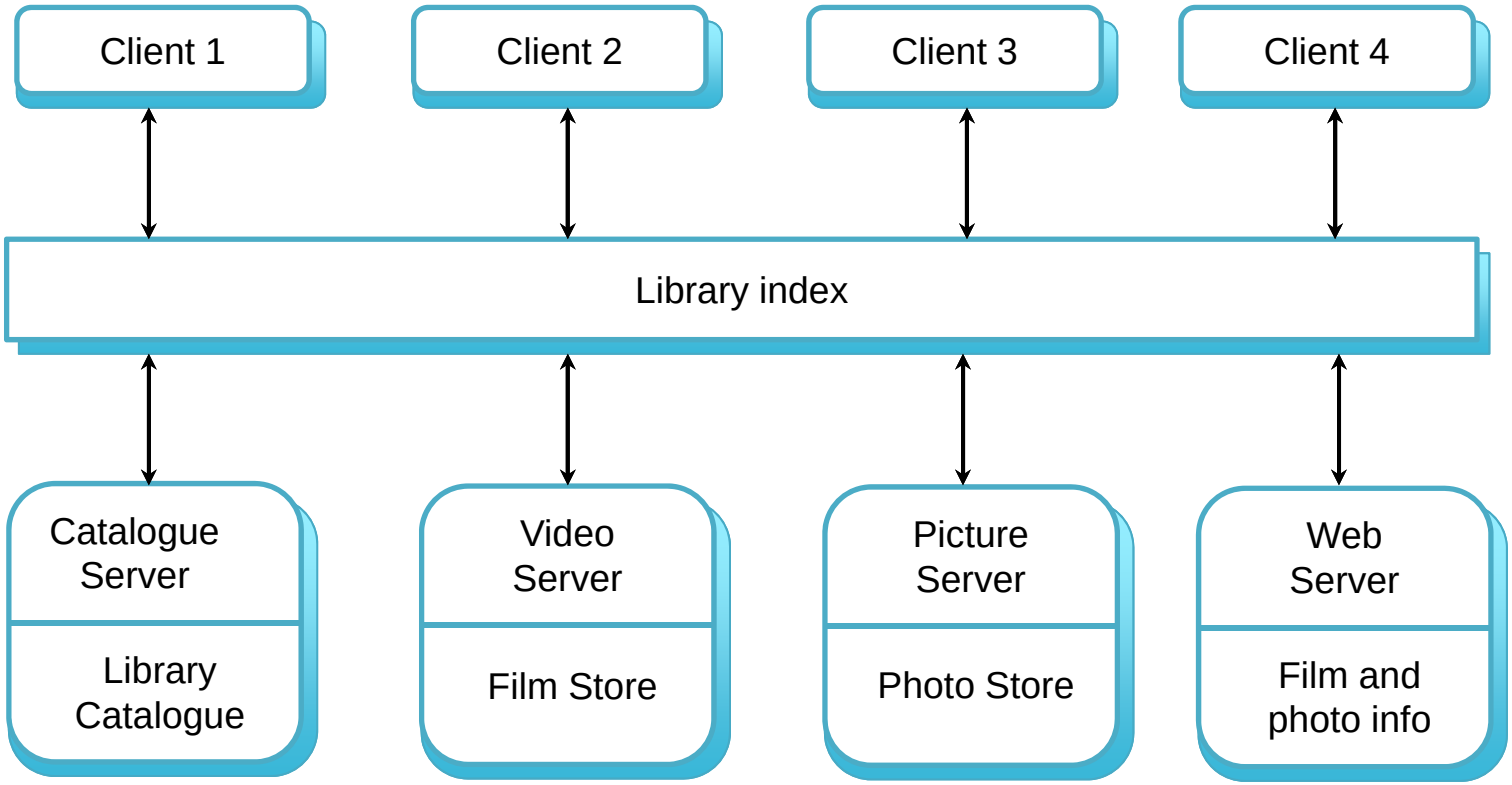
- The principal advantage of this model is that servers can be distributed across a network.
- General functionality (i.e. printing service) can be available to all clients.
- Does not need to be implemented by all services.

## Disadvantages

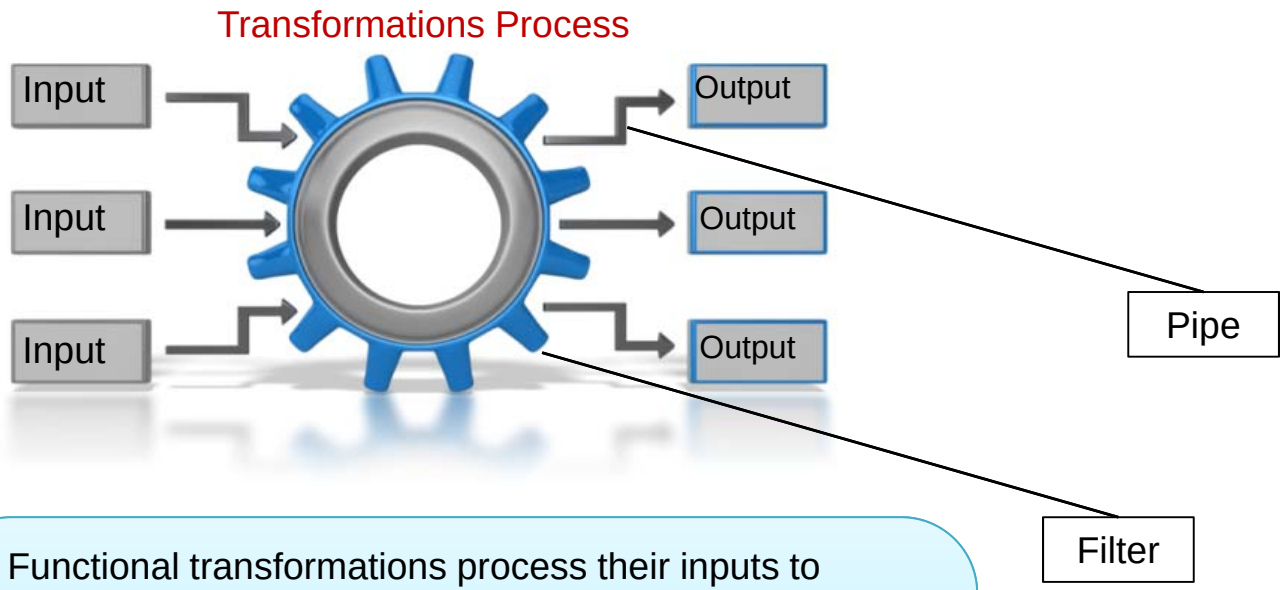
- Each service is a single point of failure so susceptible to denial of service attacks or server failure.
- Performance may be unpredictable because it depends on the network as well as the system
- May be management problems if servers are owned by different organization.

# A client–server architecture for a film library



# 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

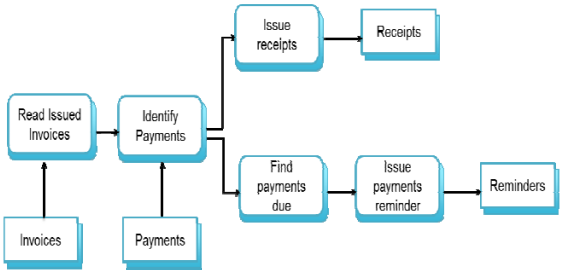

## Description

- The processing of the data in a system is organized in a way that processing component (filter) is discrete and carries out one type of data transformation
- The data flows (pipe) from one component to another for processing

## When Used?

- Commonly used in data processing applications
- Used in both batch-based and transaction-based applications
- Inputs are processed in separate stages to generate related output.

## Example




See slide No 36

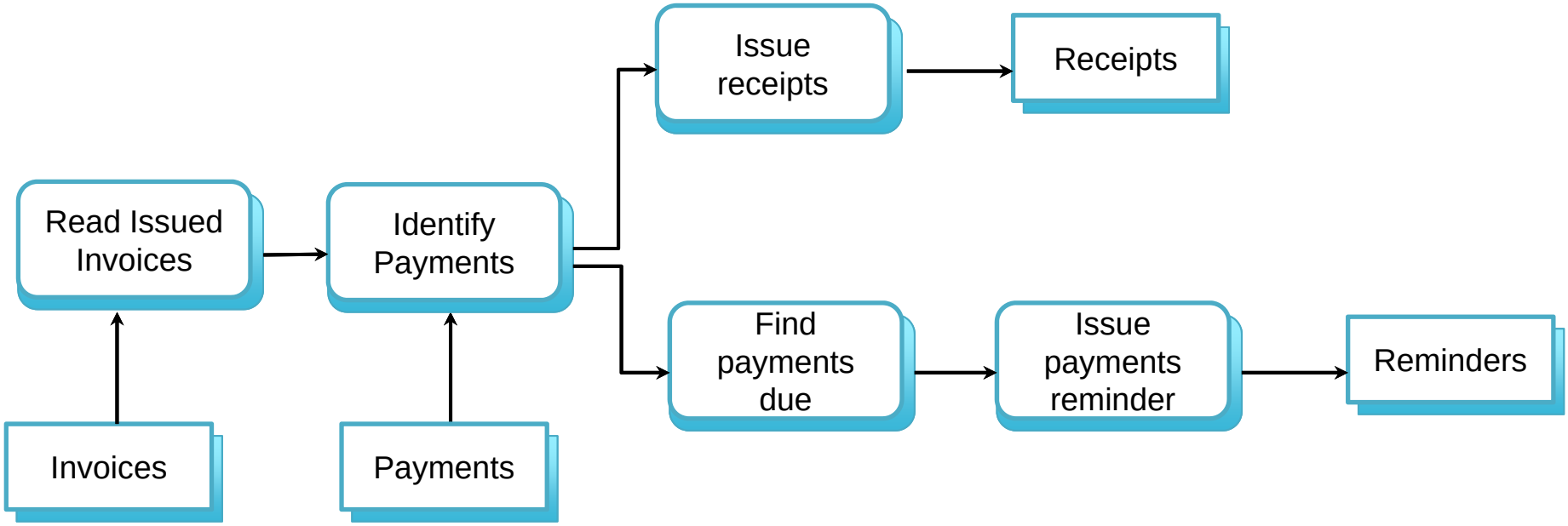
## Advantages

- Easy to understand and supports transformation reuse
- Workflow style matches the structure of many business process
- Evolution by adding transformations is straightforward.
- Can be implemented as either a sequential or concurrent system.

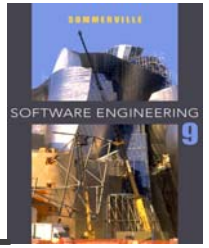
## Disadvantages

- The format for data transfer has to be agreed upon between communicating transformations.
- Each transformation must parse its input and un-parse its output to the agreed form.
- It increase system overhead
- It is impossible to reuse transformations that use incompatible data structures

# An example of the pipe and filter architecture

# Application architectures




Application System are designed → To meet → Organizational Needs

- 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

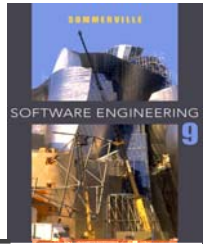
As a design checklist

As a way of organizing the work of the development teams

As a means of assessing components for reuse

As a vocabulary for talking about application types

# Examples of application types



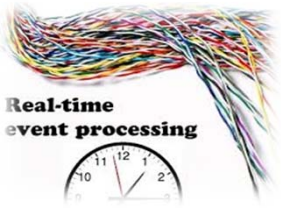


Data Processing Applications

- 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



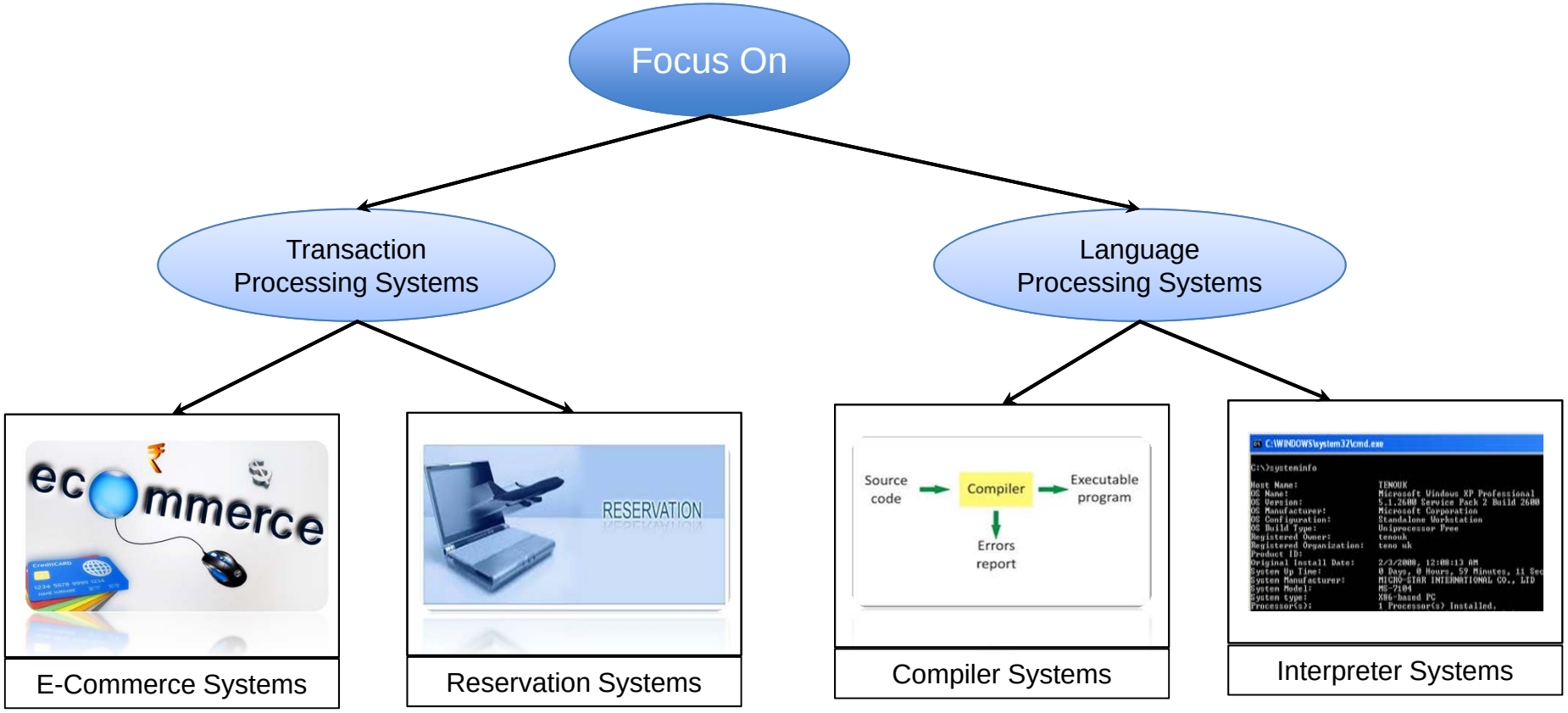

Event Processing Applications

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

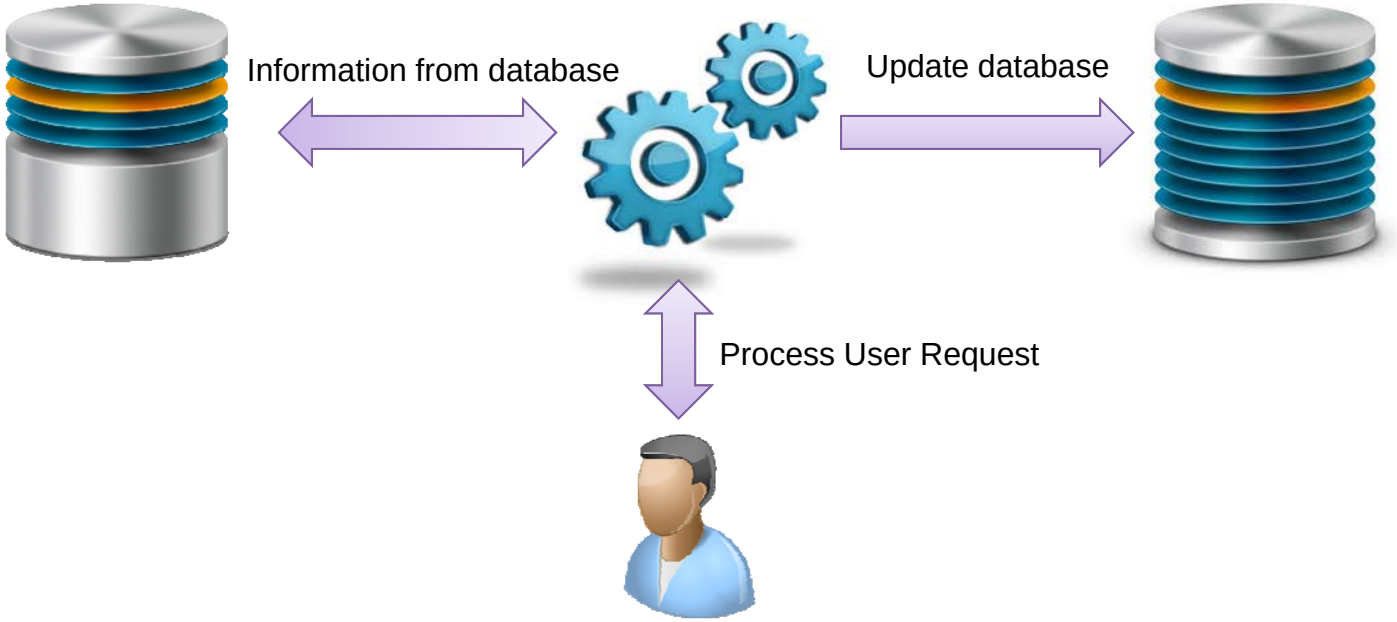
Language Processing Applications

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

# Application type examples

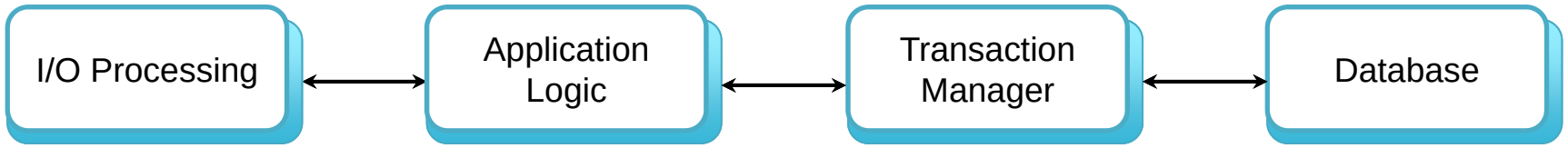



# Transaction processing systems

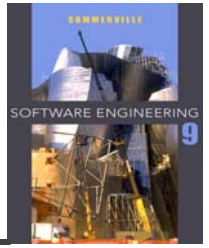



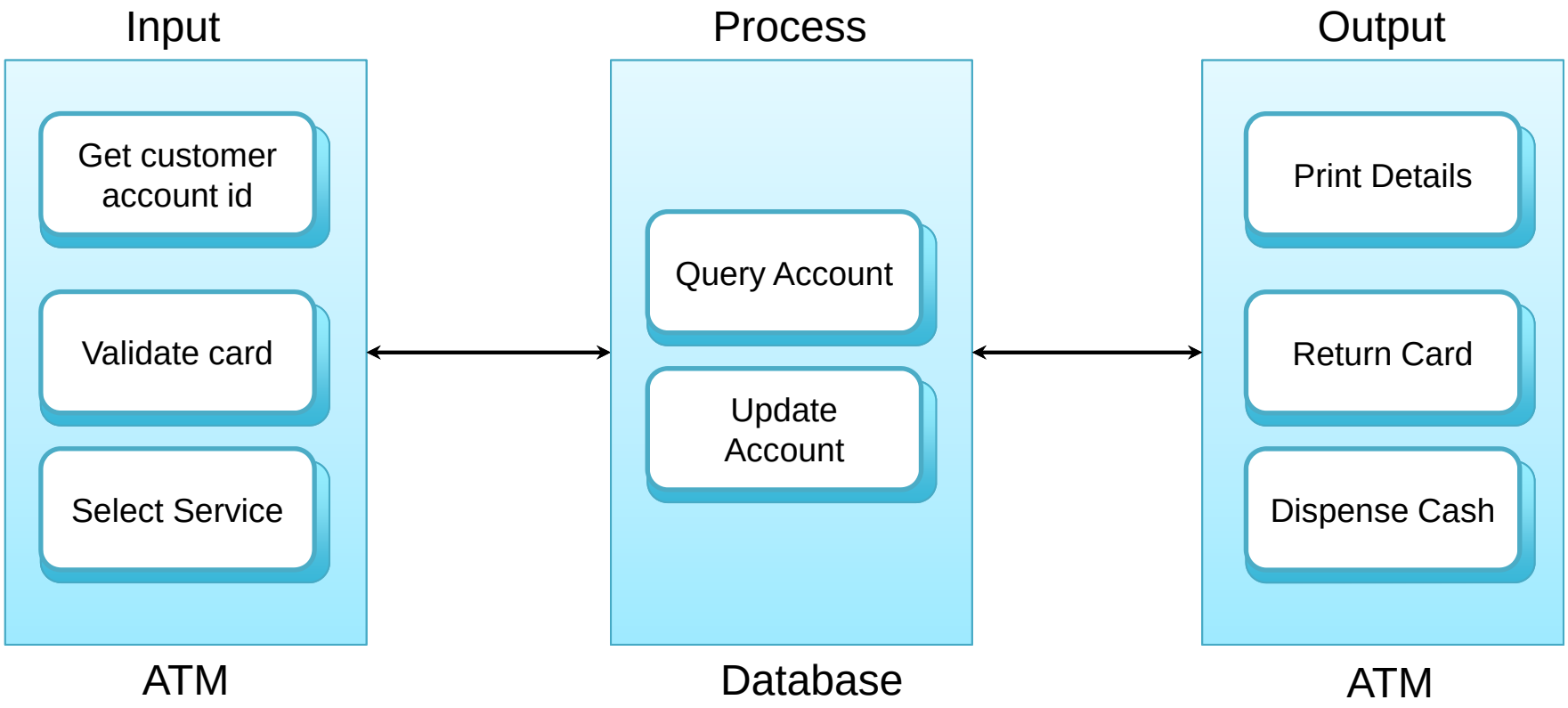

- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager

# The structure of transaction processing applications

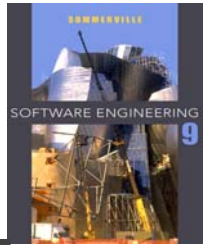



# The software architecture of an ATM system



# Information systems architecture

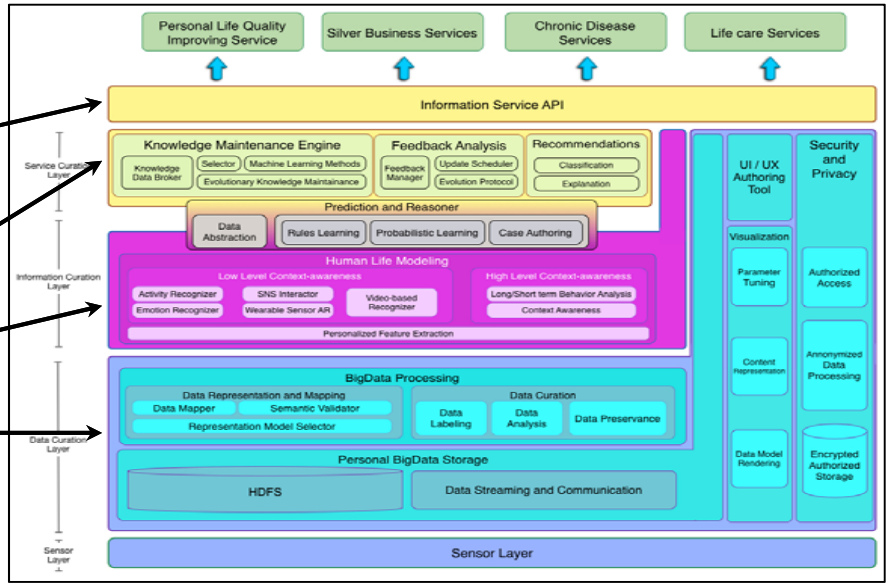



Information systems have generic architecture that can be organized as a layered architecture

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

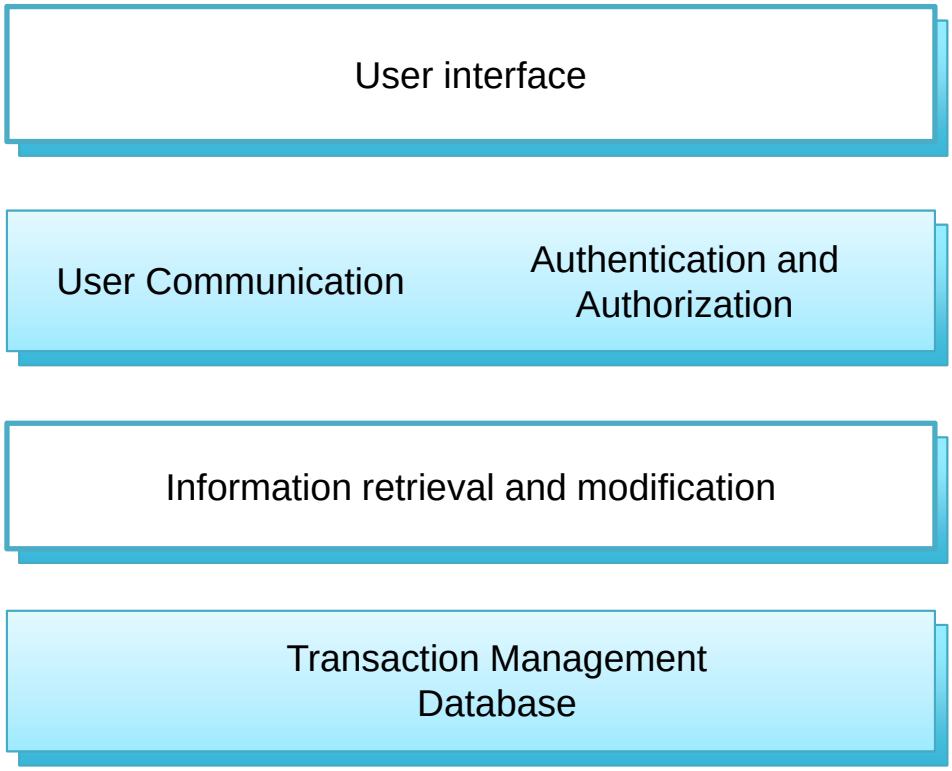
Layers Include:

- User Interface
- User Communication
- Information Retrieval
- System Database

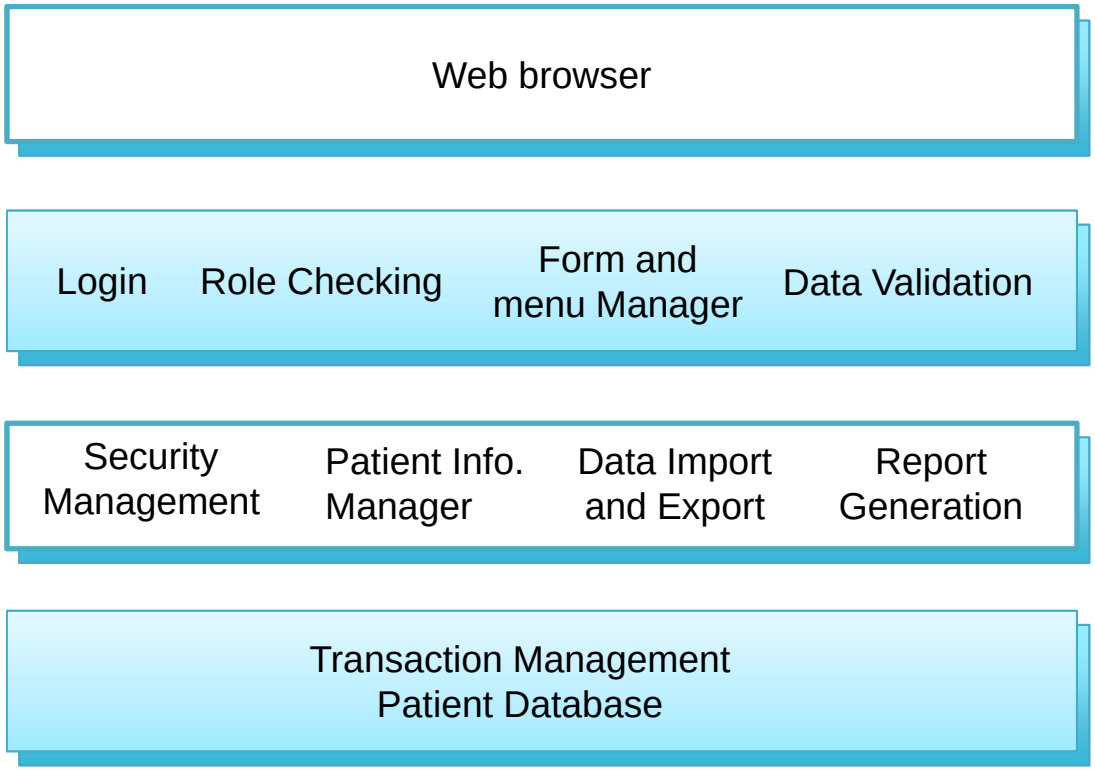



http://www.miningminds.re.kr/technical-report/presentations/

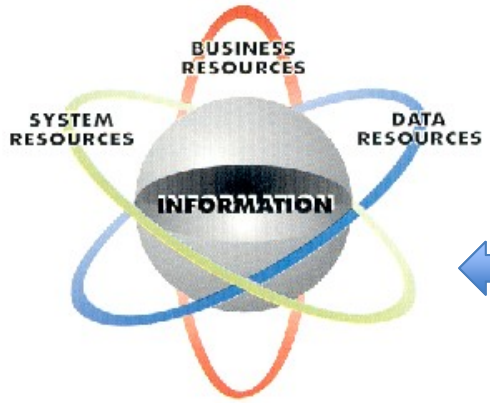
# Layered information system architecture

# The architecture of the MHC-PMS

# Web-based information systems


Information Systems

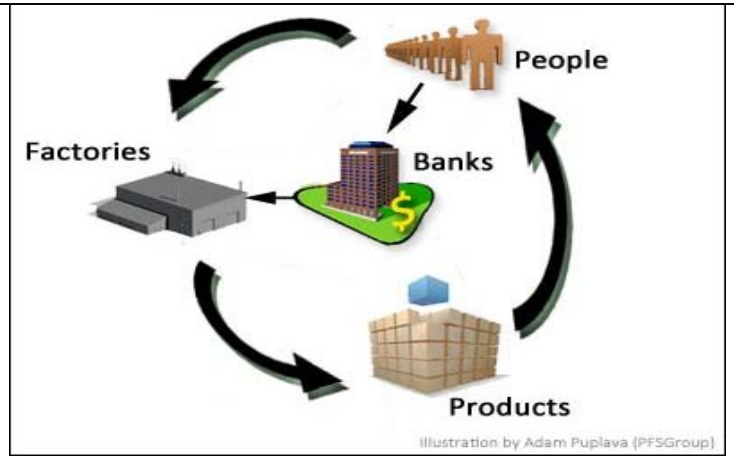
In form of

Web Applications

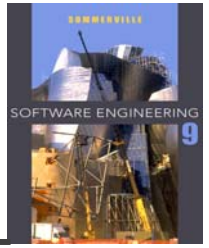
Shows on

Browsers

Example of E-Commerce Application

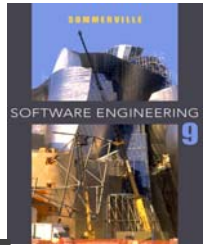


## Web-based information systems




- 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

The web server is responsible for all user communications, with the user interface implemented using a web browser

Web Server

The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests

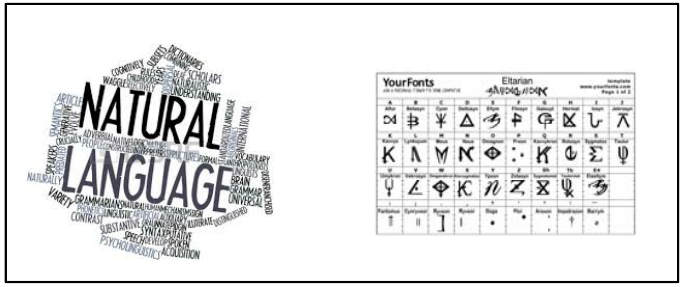
Application Server

The database server moves information to and from the database and handles transaction management

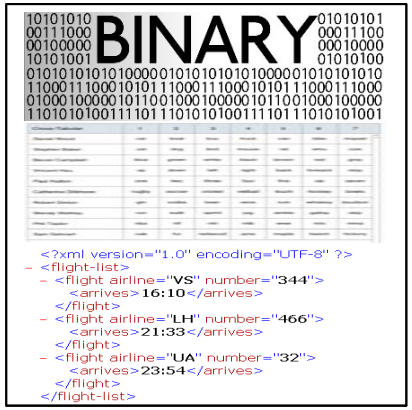
Database Server

# Language processing systems


Natural Or Artificial language

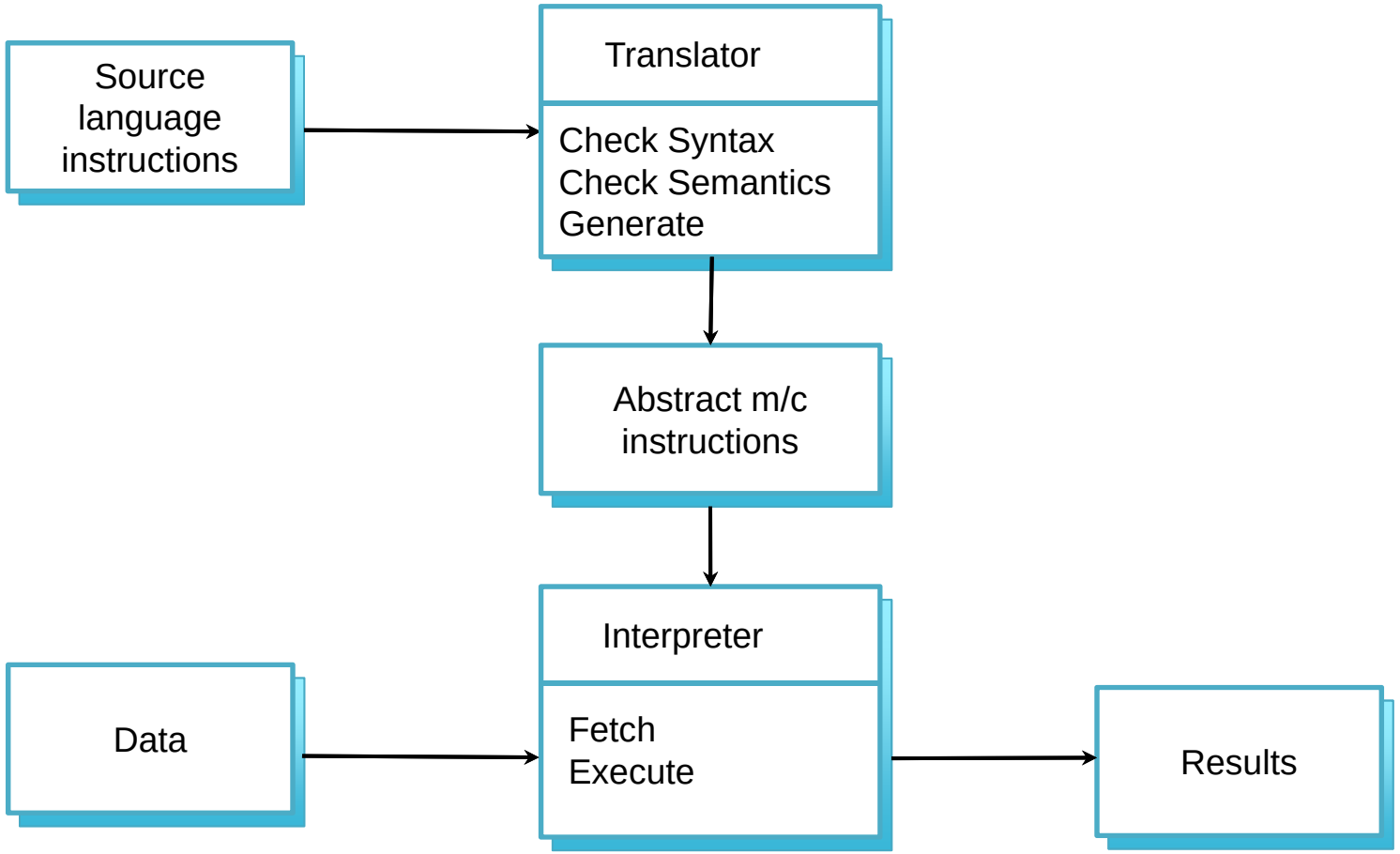



Conversion

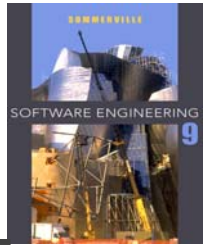



- Accept a natural or artificial language as input and generate some other representation of that language
- May include an interpreter to act on the instructions in the language that is being processed
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
- Meta-case tools process tool descriptions, method rules, etc and generate tools

# The architecture of a language processing system

# Compiler components




**Lexical Analyzer**

It takes input language tokens and converts them into an internal form

**Symbol Table**

It holds information about the names of entities (variables, classes, objects etc.)

**Syntax Analyzer**

It checks the syntax of the language being translated

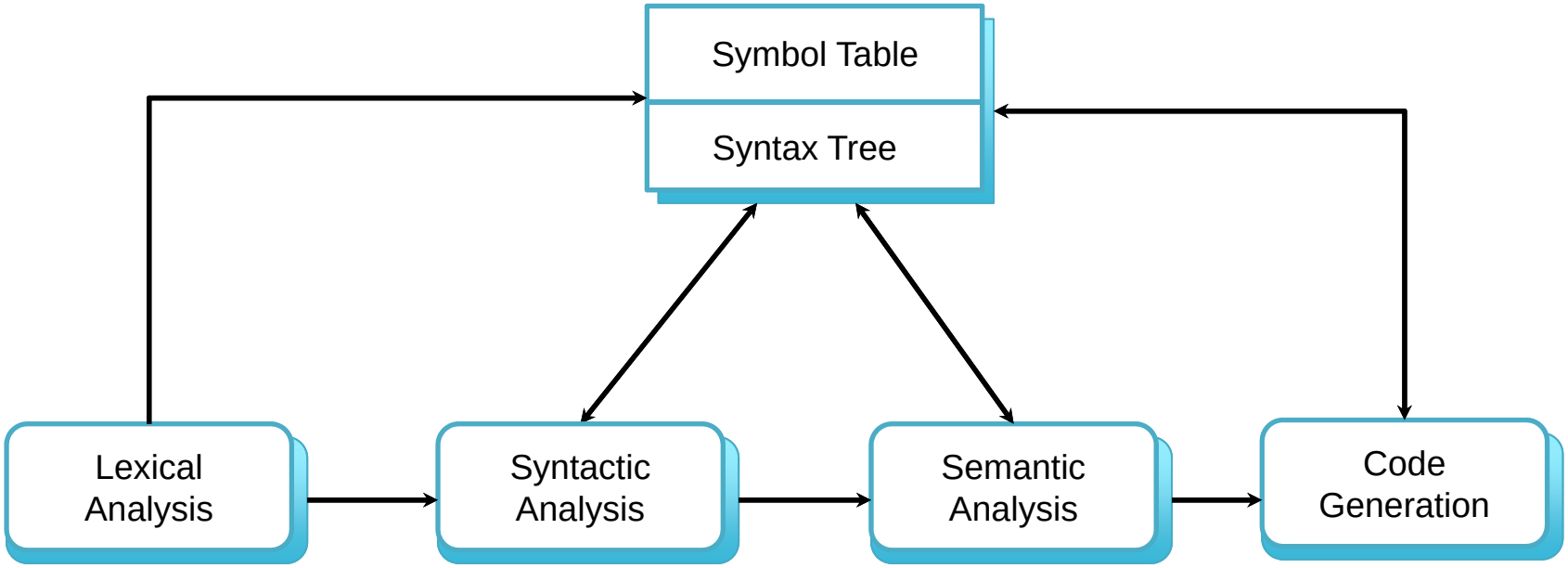
**Syntax Tree**

It is the internal structure representing the program being compiled

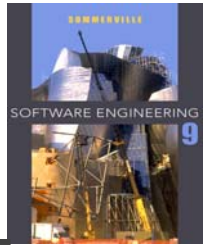
## Compiler components

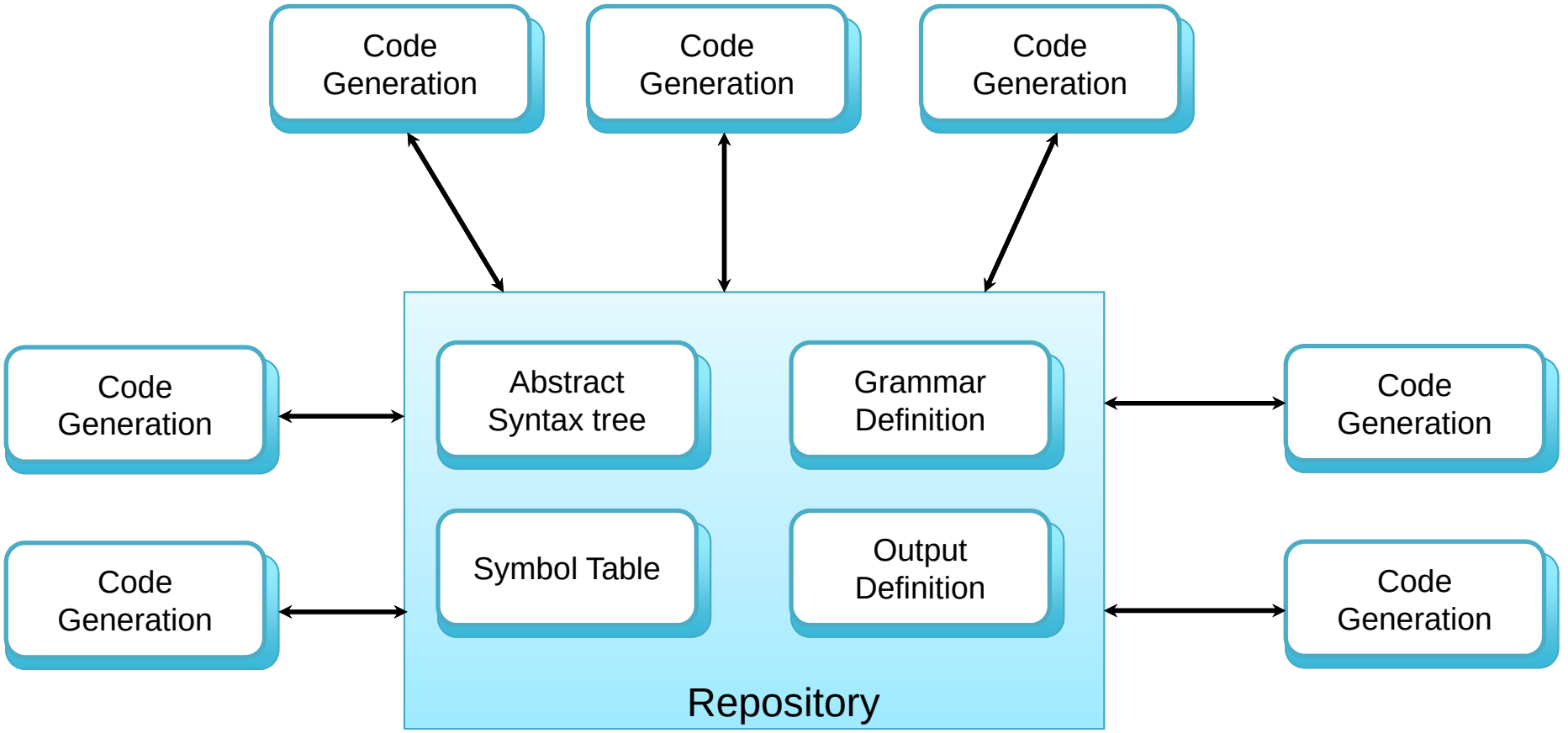

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- A code generator that ‘walks’ the syntax tree and generates abstract machine code.

# A pipe and filter compiler architecture

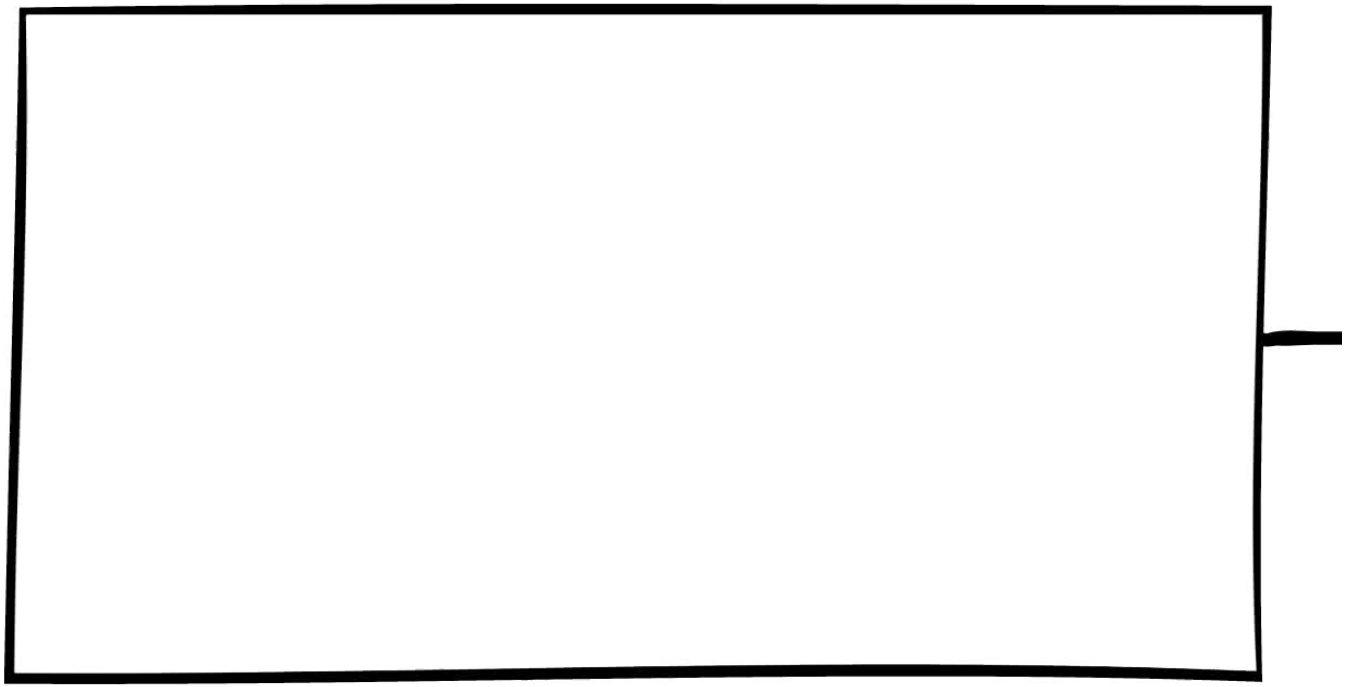



# A repository architecture for a language processing system



# Why Enterprise/Software Architect? (Video)

http://www.youtube.com/watch?v=qDI2oF1bASk

# Key points



Models of application system architectures help us

- understand and compare application
- Validate application system designs
- Assess large scale components for reuse

Transaction processing systems are interactive systems

- Allow information in a database to be remotely access
- Modified by a number of users

- Language processing systems are used to translate texts from one language to another and to carry out the instructions specified in the input language
- They include a translator and abstract machine that executes the generated language


Thank you