1. Styles and Patterns Overview
2. Data-Centered, Data-Flow, Distributed Systems
3. Interactive, Hierarchical Systems

Session I: Overview and History of Styles and Patterns

SESSION'S AGENDA

- Overview of **Architecture Styles and Patterns**
- History of Architectural Styles and Patterns
  - ✓ Origin of styles and patterns
- **Classification of Architectural Styles and Patterns**
  - ✓ **Data-Centered**
  - ✓ **Data flow**
  - ✓ **Distributed**
  - ✓ **Interactive**
  - ✓ **Hierarchical**

OVERVIEW OF ARCHITECTURE STYLES AND PATTERNS

- In the previous module, it was established that software systems need to be carefully architected and evaluated from different perspectives.
  - ✓ This is necessary to address multiple concerns that shape the quality of the system from different stakeholders with different backgrounds.

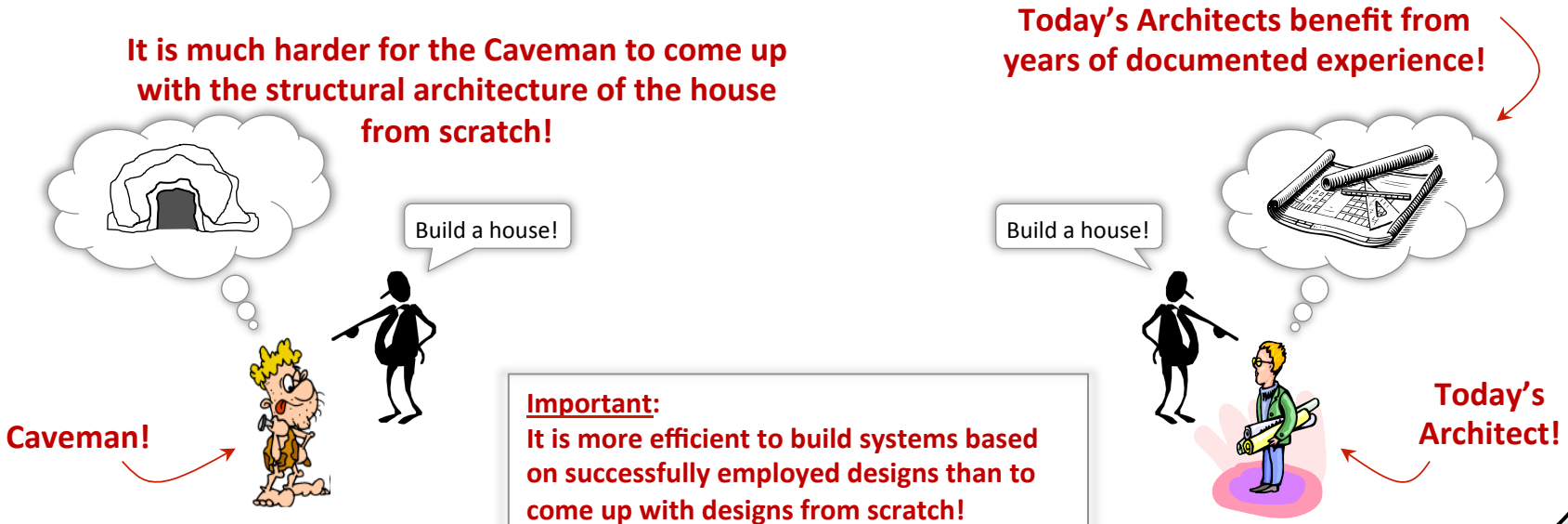
- In this module, we pay special attention (mostly) to the ***logical view*** **of software architecture.**
  - ✓ That is, the perspective that deals with **decomposing the software system** into **logical components** that represent the **structural integrity** that supports **functional** and **non-functional (quality) requirements.**

- When designing logical architectures, it is important to use past experience to discover overall strategies that have been used successfully in the development of software systems.
  - ✓ To this end, the concepts of *architectural styles* and *architectural patterns* have emerged as mainstream approach for achieving reuse at the architectural level.

Overview of Architecture Styles and Patterns

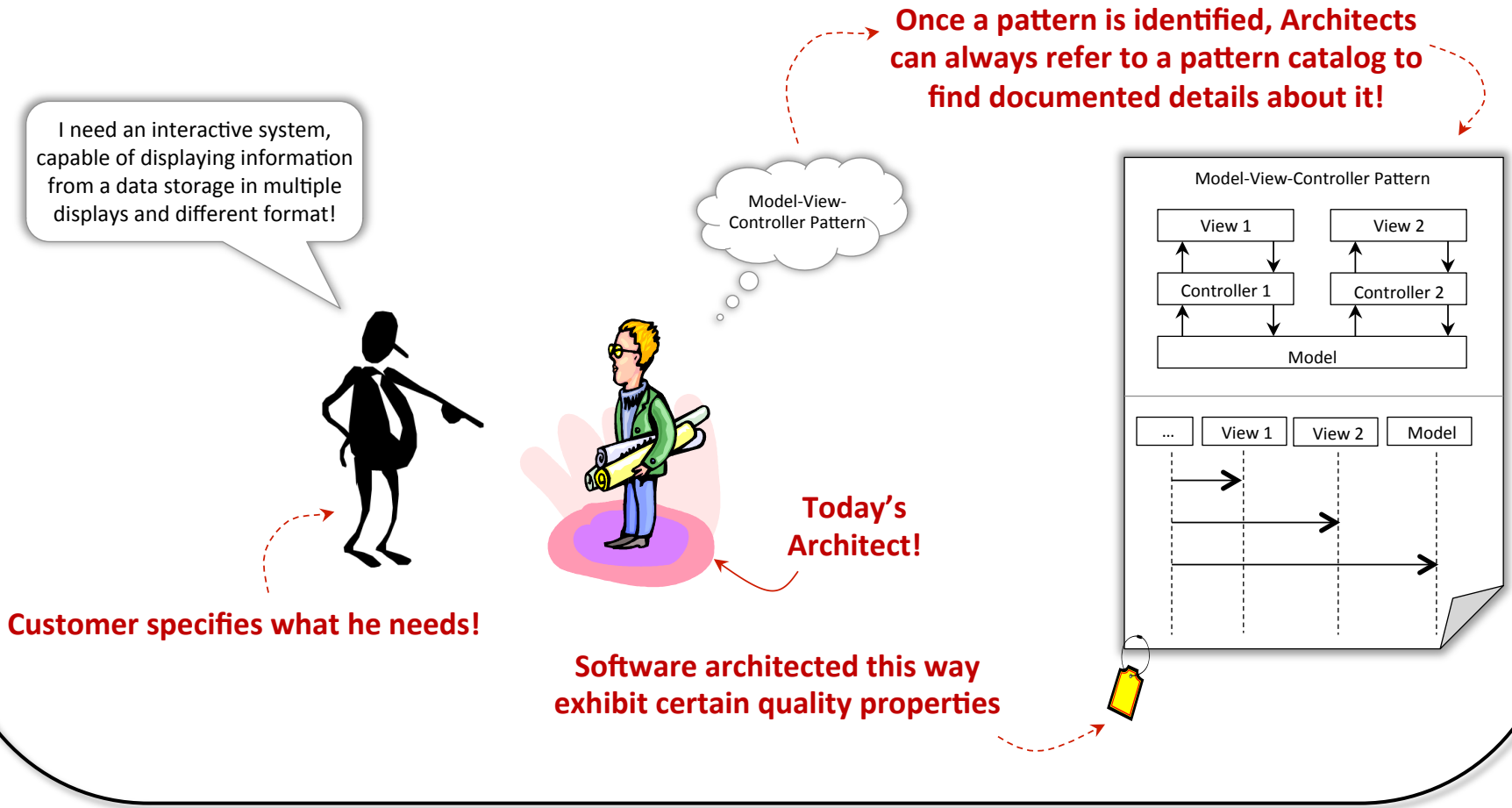
- Architectural styles and architectural patterns provide **generic, reusable solutions that can be easily understood.**
  - ✓ These can be easily applied to new problems requiring similar architectural features.

- Decisions based on architectural styles and patterns benefit from years of documented experience that highlights
  - ✓ The **solution** approach to a given problem.
  - ✓ The **benefits of these approaches**.
  - ✓ The **consequences** of employing these approaches.

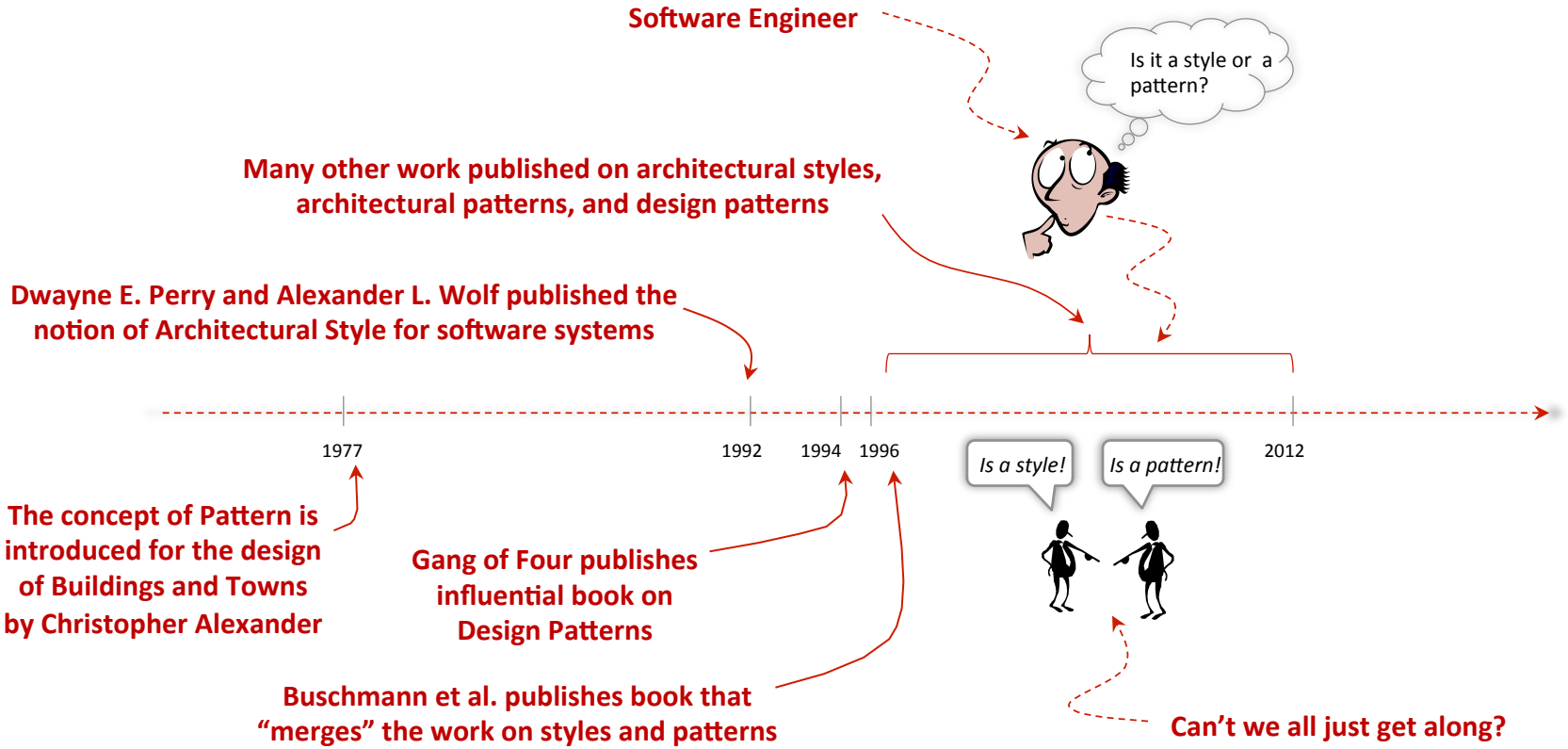


Overview of Architecture Styles and Patterns

- Similar to the previous example, today's software architect can benefit from numerous documented styles and patterns for software architecture.



History of Architecture Styles and Patterns



History of Architecture Styles and Patterns

- Before we move on, it is important to discuss the history of styles and patterns. This will help us eliminate some of the confusion between the terms *styles* and *patterns*.

- In 1977, Christopher Alexander presented a language intended to help individuals, or teams, design quality structures of different sizes, shapes, and complexities [1]. According to Alexander et al.:
  - ✓ "*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*"

- Alexander's work resulted in a catalogue of 253 patterns, each describing in detail the essential information required for documenting the patterns, including [1]:
  - ✓ Picture of the pattern
  - ✓ Context of the pattern
  - ✓ Problem that the pattern attempts to solve
  - ✓ Evidence for its validity
  - ✓ Related patterns

History of Architecture Styles and Patterns

- Although Alexander's work on patterns appears relevant to the software engineering profession, it actually referred to patterns found in the design of buildings and towns.
  - ✓ This work, however, significantly impacted the field of software engineering.

- In the 1990s, the software engineering community began researching and finding recurring high-level problem solutions in terms of specific elements and their relationships; these were originally referred to as **architectural styles** [2].
  - ✓ Similar to Alexander's work, Architectural Styles provided the means for software architects to reuse design solutions in different projects; that is, to use a "solution a million times over, without ever doing it the same way twice."[1]

- In 1994, Gamma, Helm, Johnson, and Vlissides—better known as the Gang of Four (GoF)—published their influential work that focused on a finer-grained set of object-oriented detailed design solutions that could be used in different problems "a million times over, without ever doing it the same way twice."
  - ✓ Influenced by Alexander's work, they called these **Design Patterns**.
  - ✓ Their work resulted in the creation of a catalogue of 23 (detailed design) patterns.
  - ✓ Each pattern was described in detail, using a specific pattern specification format.

History of Architecture Styles and Patterns

- In 1996, the work of Buschmann, Meunier, Rohnert, Sommerland, and Stal [3], integrated the work of architectural styles and design patterns by providing a set of well-known architectural styles using a pattern-like approach [2].
  - ✓ They referred to these as **Architectural Patterns**.

- In their work, Buschmann et al. provided their views about architectural patterns vs. design patterns[3]:
  - ✓ "*Architectural patterns … specify the system-wide structural properties of an application. They specify the system-wide structural properties of an application, and have an impact on the architecture of its subsystems.*"
  - ✓ "Design patterns are medium-scale patterns."
  - ✓ "*The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem.*"

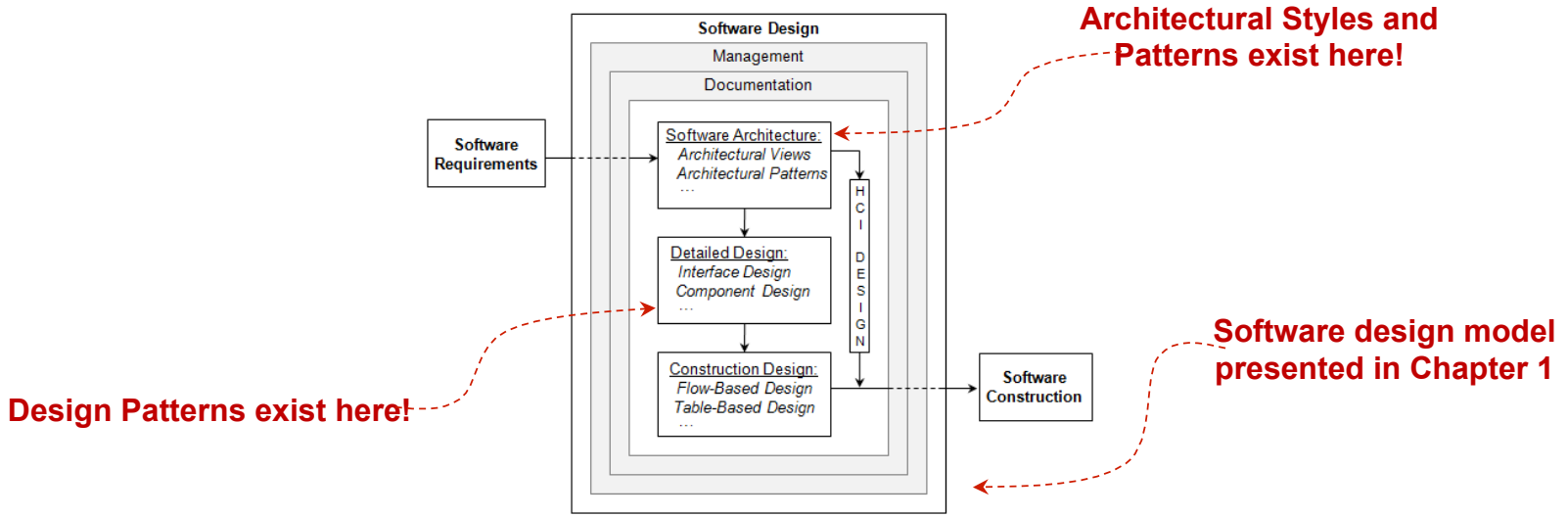
- Hopefully, this helps clear the air between the concepts of *architectural patterns* vs. *design patterns*, but what about *Architectural Styles* vs. *Architectural Patterns*?
  - ✓ There **are** documented differences between architectural styles and patterns!
  - ✓ However, perhaps Buschmann et al. [3] put it best by stating that "*Architectural styles are very similar to our architectural patterns. In fact every architectural style can be described as an architectural pattern.*"

History of Architecture Styles and Patterns

- Today, the terms *architectural styles* and *architectural patterns* are used to convey **fundamental structural and architectural organization for software systems.**
  - ✓ Throughout the rest of the course, these terms are used interchangeably to denote reusable design solutions that occur during the *software architecture activity* of the design process.

History of Architecture Styles and Patterns

- The term *Design Patterns*, as seen later on in the course, is used to denote reusable design solutions that occur during the *detailed design activity* of the design process.
- Architectural styles and architectural patterns do not describe the detailed design of systems
  - ✓ They are used as basis for system decomposition and for analyzing the structure of systems in principled manner.



ARCHITECTURAL PATTERN CLASSIFICATION

- The choice of applying architectural patterns depend on the **type of system, requirements, and desired quality attributes.**
  - ✓ These characteristics help guide the choice of selecting one particular pattern over another.

- In some cases, more than one architectural pattern can be used in combination to collectively provide the appropriate architectural solution.

- Architectural patterns can be classified depending on the type of system as shown below:

Type	Description
Data-Centered	Systems that serve as a centralized repository for data, while allowing clients to access and perform work on the data.
Data Flow	Systems oriented around the transport and transformation of a stream of data.
Distributed	Systems primarily involve interaction between several independent processing units connected via a network.
Interactive	Systems that serve users or user-centric systems.
Hierarchical	Systems where components can be structured as a hierarchy (vertically and horizontally) to reflect different levels of abstraction and responsibility.

Recap On Important Quality Attributes Of Software Systems (Chapter 3- Architecture Principles : Key Tasks)

Quality	Description
Usability	The degree of complexity involved when learning or using the system.
Modifiability	The degree of complexity involved when changing the system to fit current or future needs.
Security	The system's ability to protect and defend its information or information system.
Performance	The system's capacity to accomplish useful work under time and resource constraints.
Reliability	The system's failure rate.
Portability	The degree of complexity involved when adapting the system to other software or hardware environments.
Testability	The degree of complexity involved when verifying and validating the system's required functions.
Availability	The system's uptime.
Interoperability	The system's ability to collaborate with other software or hardware systems.

Back

SUMMARY…

- In this session, we presented fundamentals concepts of architectural styles and patterns, including:
  - ✓ Overview of Architectural Styles and Patterns
  - ✓ History of Architectural Styles and Patterns
    - Origin of styles and patterns
  - ✓ Classification of styles and patterns
  - ✓ Types of systems for classifying styles and patterns:
    - Data-Centered
    - Data flow
    - Interactive
    - Hierarchical

Session II:
System : Data-Centered, Data-Flow, Distributed

SESSION'S AGENDA

1. **Data-Centered Systems**
   - ✓ Overview
   - ✓ Patterns
     - **Blackboard**

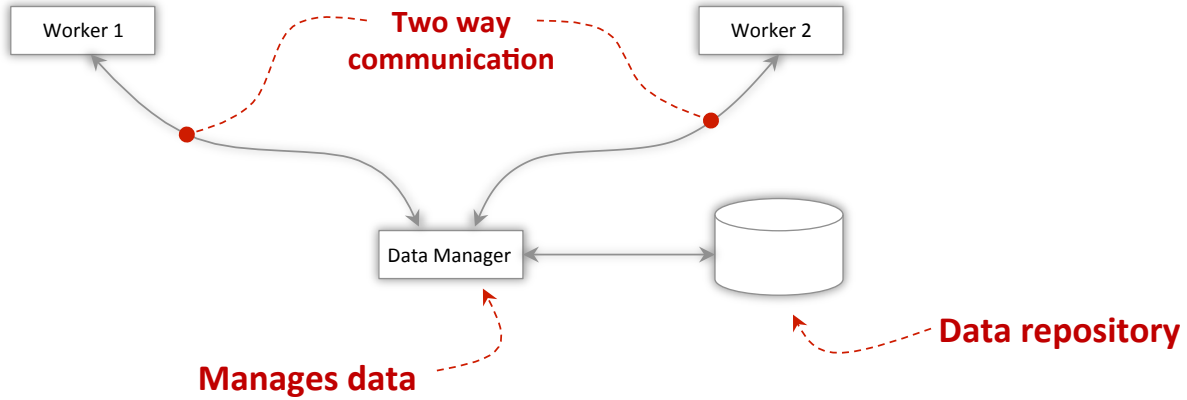
2. **Data Flow Systems**
   - ✓ Overview
   - ✓ Patterns
     - **Pipes-and-Filters**

3. **Distributed Systems**
   - ✓ Overview
   - ✓ Patterns
     - **Client Server**
     - **Broker**

1. Data-Centered Systems

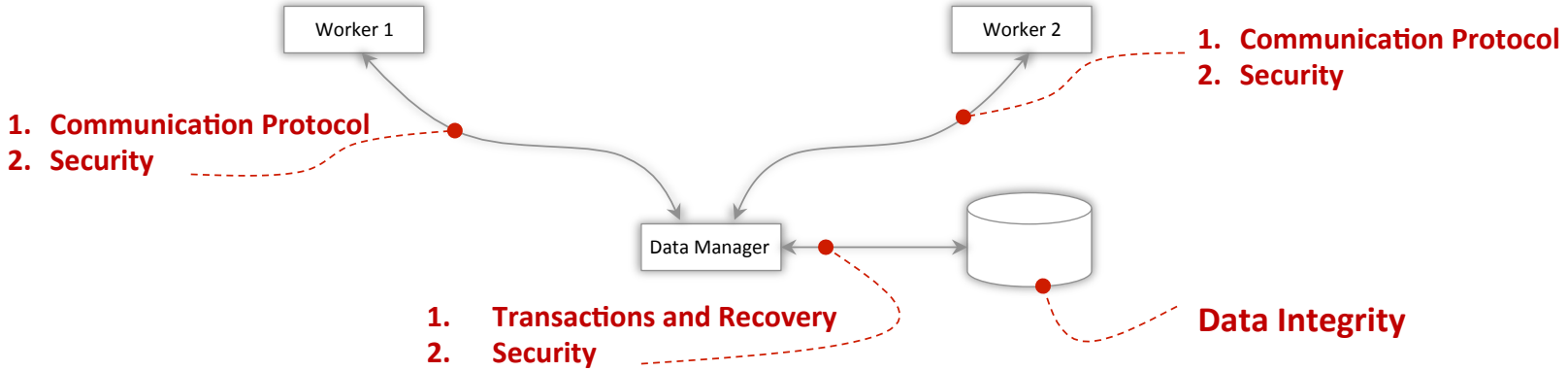
- Data-centered systems are s***ystems primarily decomposed around a main central repository of data***. These include:
  1. **Data management component -** controls, provides, and manages access to the system's data.
  2. **Worker components -** execute operations and perform work based on the data.

- Communication in data-centered systems is characterized by a one-to-one bidirectional communication between a worker component and the data management component.
  - ✓ Worker components do not interact with each other directly; all communication goes through the data management component.



1. Data-Centered Systems

- Because of the architecture of these systems, they must consider issues with:
  - ✓ **Data integrity**
  - ✓ **Communication protocols** between worker and data management
  - ✓ **Transactions and recovery** (also known as roll-back)
  - ✓ **Security**



- A common architectural pattern for data-centered systems is the ***Blackboard Pattern*.**

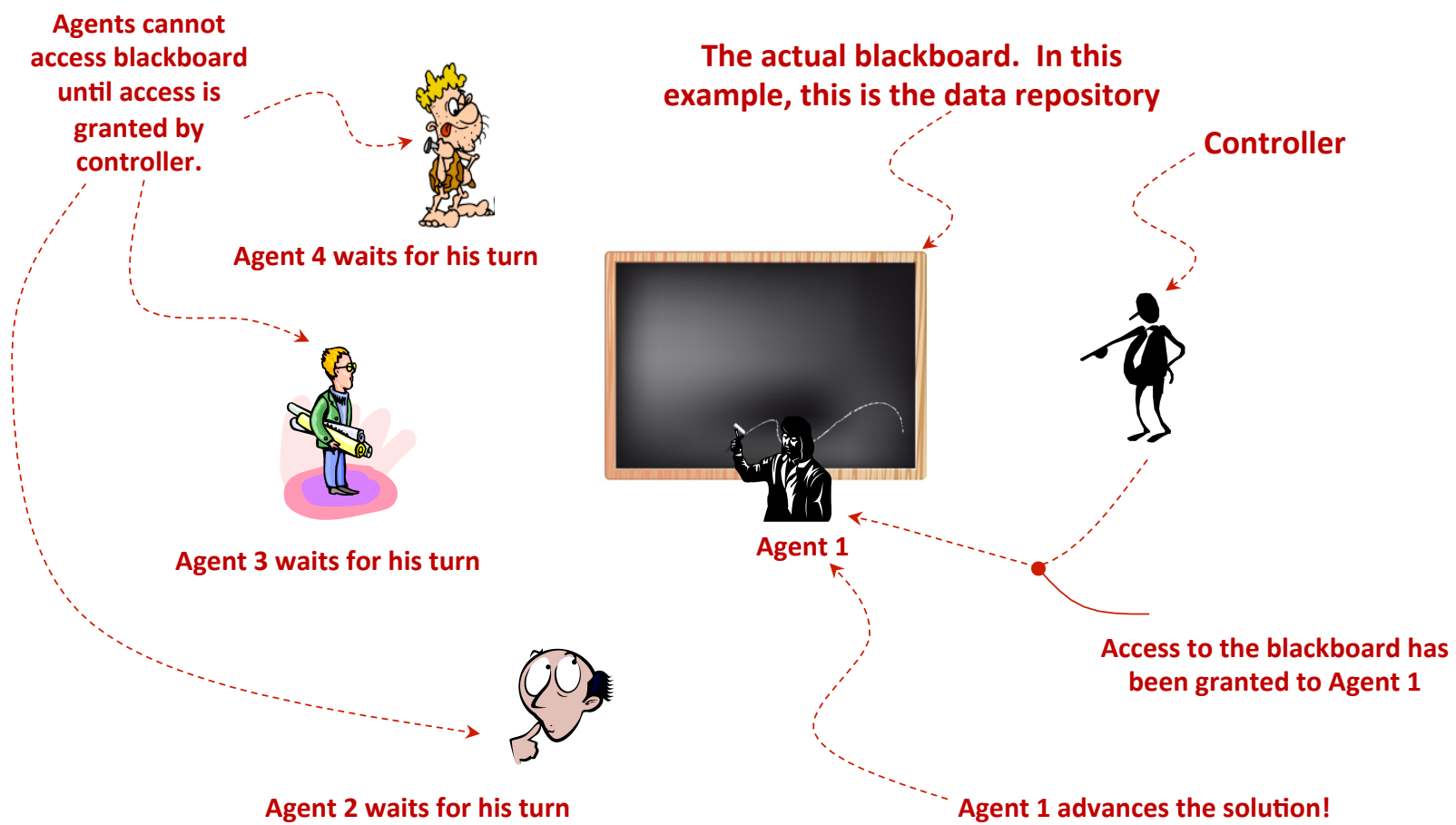
BLACKBOARD ARCHITECTURAL PATTERN

- Blackboard decomposes systems into components that work around a central data component to provide solutions to complex problems.
  - ✓ These components work independently from each other to provide partial solutions to problems using an opportunistic problem-solving approach.
  - ✓ That is, there are no predetermined, or correct, sequences of operations for reaching the problem's solution.

- The Blackboard architectural pattern resembles the approach a group of scientists would employ to solve a complex problem.
  - ✓ Consider a group of scientists at one location using a blackboard (chalkboard, whiteboard, or electronic blackboard) to solve a complex problem.
  - ✓ Assume that to manage the problem-solving process, a mediator controls access to the blackboard.
  - ✓ Once the mediator (or controller) assigns control to the blackboard, a scientist evaluates the current state of the problem and if possible, advances its solution before releasing control of the blackboard.
  - ✓ With new knowledge obtained from the previous solution attempt, control is assigned to the next scientist who can further improve the problems' state.
  - ✓ This process continues until no more progress can be made, at which point the blackboard system reaches a solution.

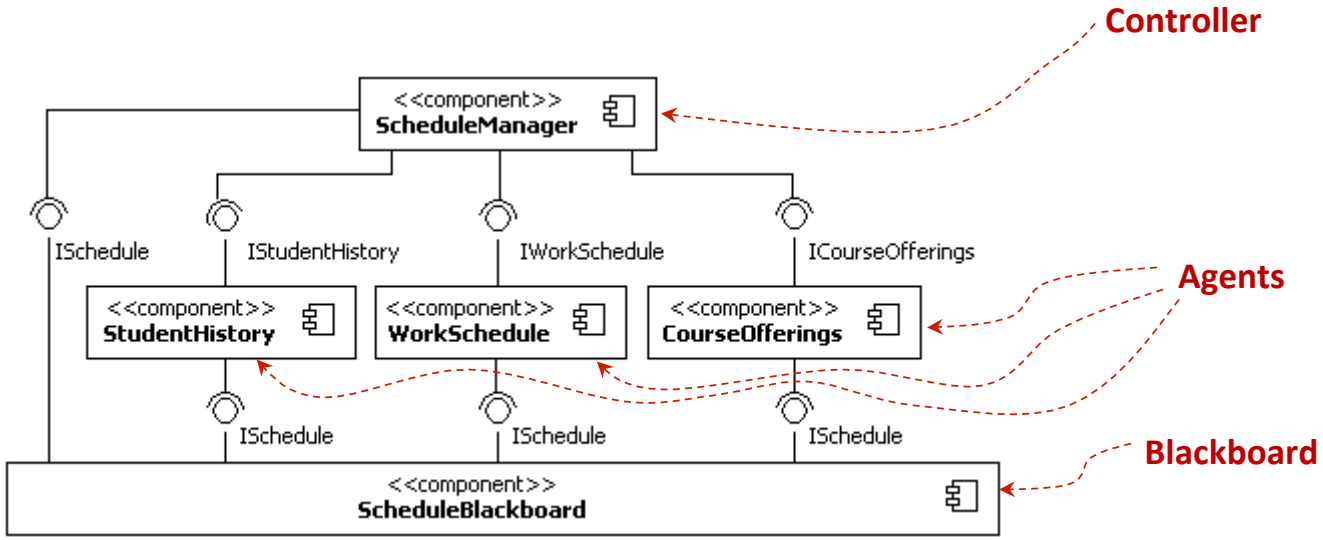
- This behavior is prevalent in expert systems, therefore, the Blackboard architectural pattern is a good choice for depicting the logical architecture of **expert systems**.

Blackboard Architectural Pattern

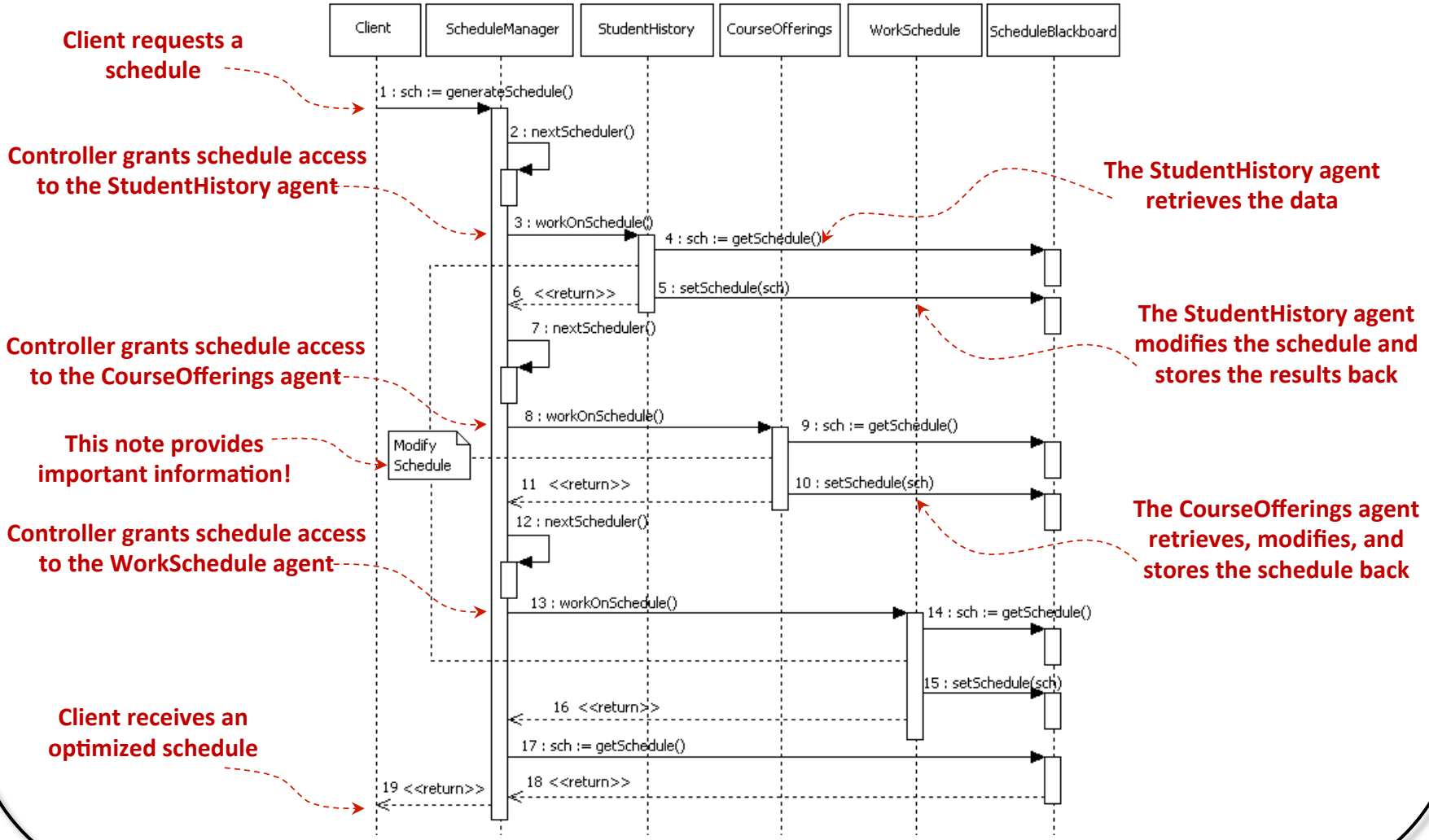


BLACKBOARD ARCHITECTURAL PATTERN

- Consider the **Students' Scheduling System** from Chapter 4.



Blackboard Architectural Pattern



BLACKBOARD ARCHITECTURAL PATTERN

- Quality properties of the Blackboard architectural pattern include the ones specified below.

Quality	Description
Modifiability	Agents are compartmentalized and independent from each other; therefore, it is easy to add or remove agents to fit new systems.
Reusability	Specialized components can be reused easily in other applications.
Maintainability	Allows for separation of concerns and independence of the knowledge based agents; therefore, maintaining existing components becomes easier.

- An important aspect of the Blackboard and any other architectural pattern is their **deployment aspect** (i.e., the deployment view). For example, It is not easily determined from the logical view where each agent or blackboard component reside.
  - ✓ Depending on their location, Blackboard can have increased complexity when managing communication between agents, controller, and blackboard.

2. DATA FLOW SYSTEMS

- Data flow systems are decomposed around the central theme of **transporting data (or data streams)** and **transforming the data** along the way to meet application-specific requirements.
  - ✓ Typical responsibilities found in components of data-flow systems include:
    1. **Worker** components, those that **perform work on data**
    2. **Transport** components, those that **transporting data**

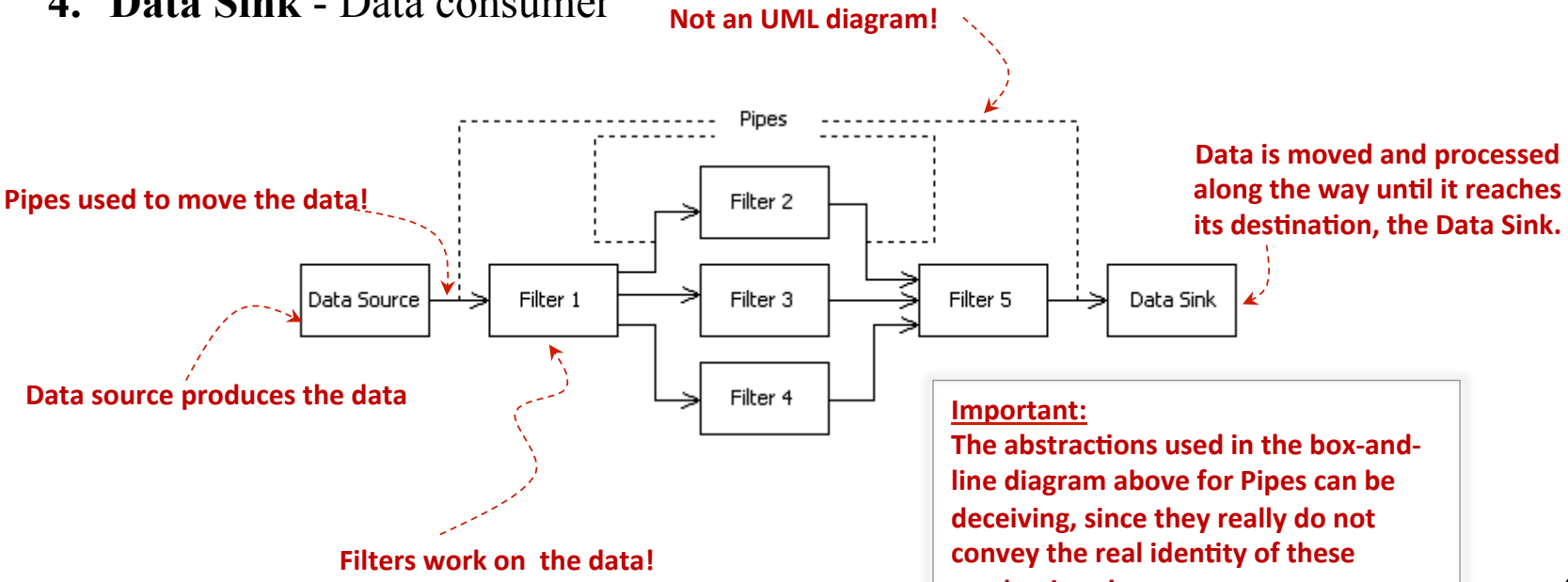
- Worker components abstract data transformations and processing that need to take place before forwarding data streams in the system, e.g.,
  - ✓ Encryption and decryption
  - ✓ Compression and decompression
  - ✓ Changing data format, e.g. ,from binary to XML, from raw data to information, etc.
  - ✓ Enhancing, modifying, storing, etc. of the data

- Transport components abstract the management and control of the data transport mechanisms, which could include:
  - ✓ Inter-process communication
    - Sockets, serial, pipes, etc.
  - ✓ Intra-process communication
    - Direct function call, etc.

- An example of an architectural pattern for data flow systems is the ***Pipes-and-Filters***.

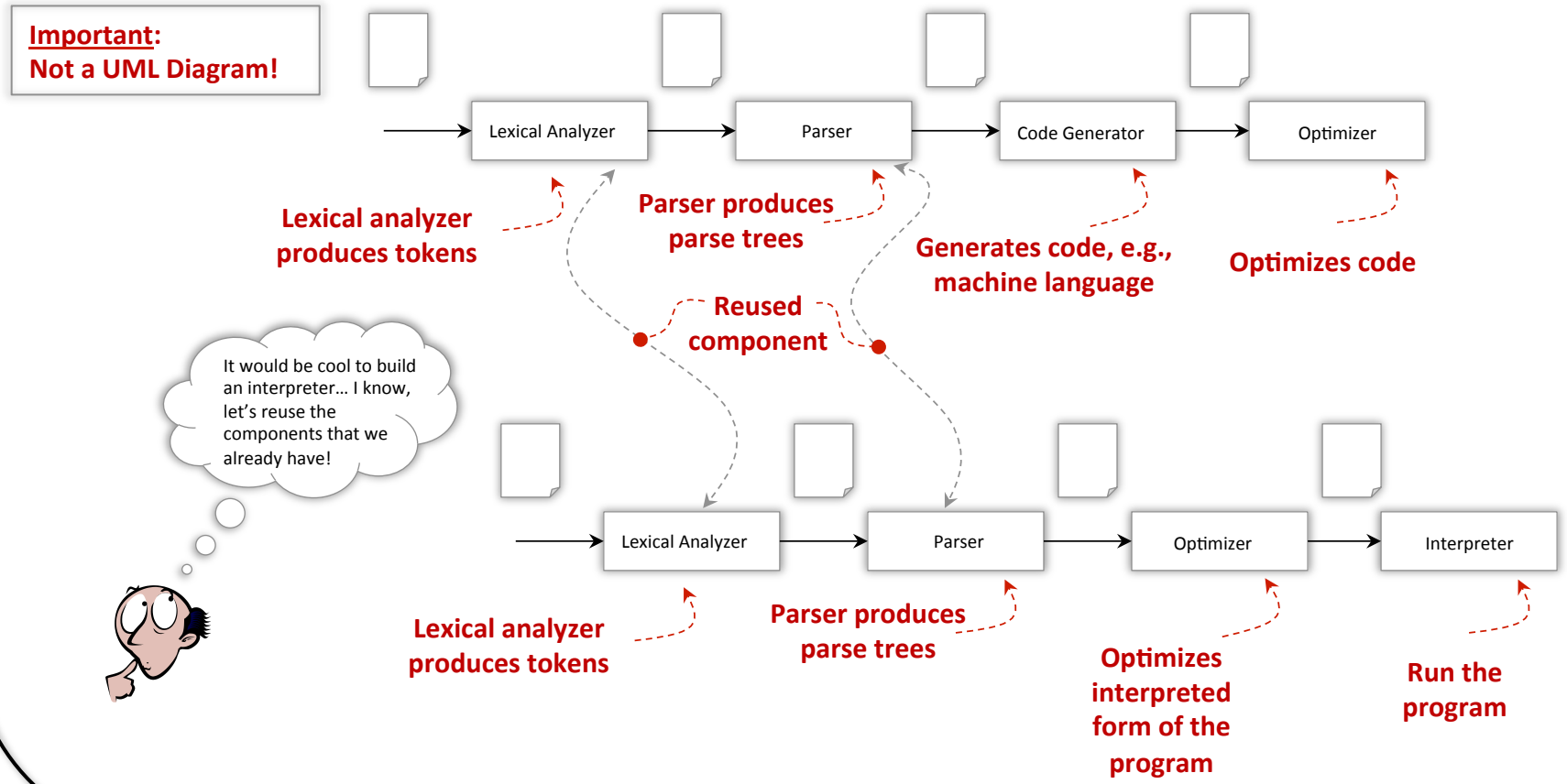
PIPES-AND-FILTERS ARCHITECTURAL PATTERN

- Pipes-and-Filters is composed of the following components:
  1. **Data source -** Produces the data
  2. **Filter** - Processes, enhances, modifies, etc. the data
  3. **Pipes** - Provide connections between data source and filter, filter to filter, and filter to data sink.
  4. **Data Sink** - Data consumer



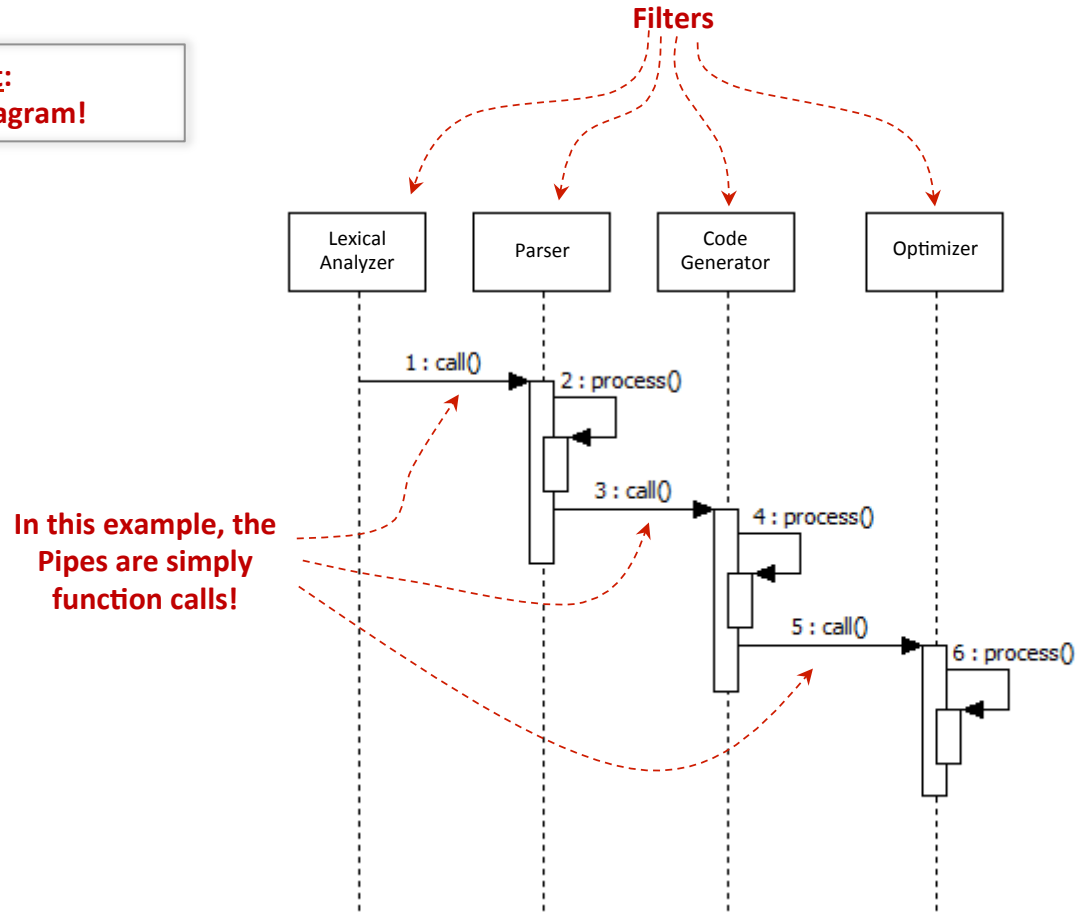
PIPES-AND-FILTERS ARCHITECTURAL PATTERN

- A common example for the Pipes-and-Filters pattern:
  - ✓ **Architecture of a Language Processor (e.g., compiler, interpreter)**



Pipes-and-Filters Architectural Pattern

Important:
A UML Diagram!

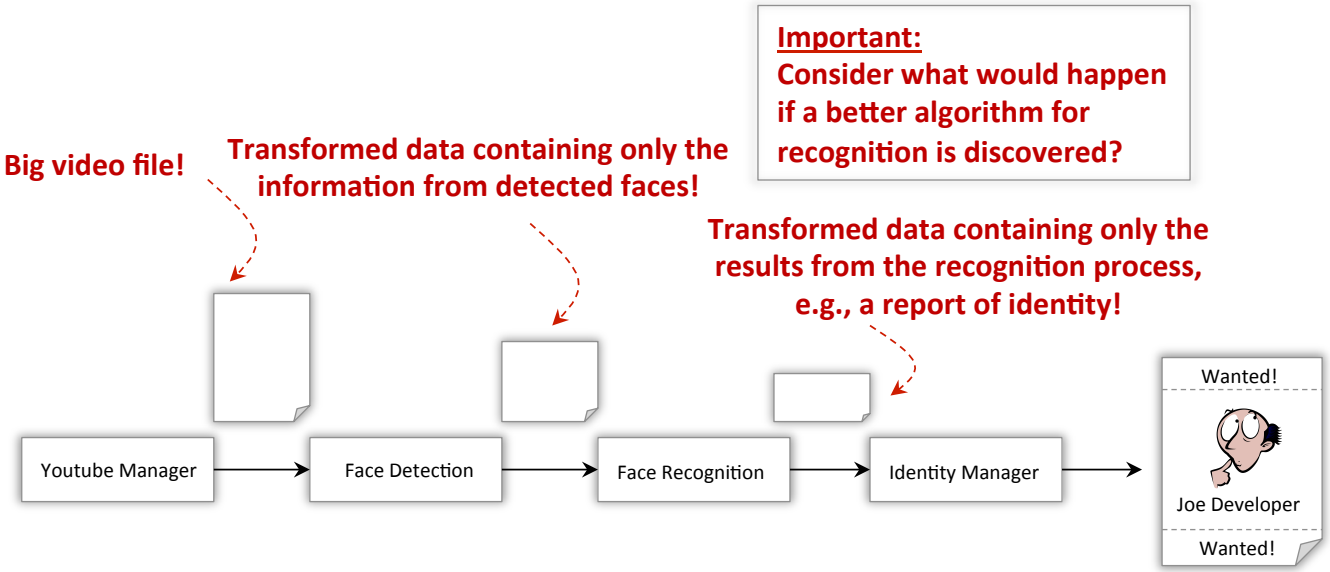


In this example, the Pipes are simply function calls!

PIPES-AND-FILTERS ARCHITECTURAL PATTERN

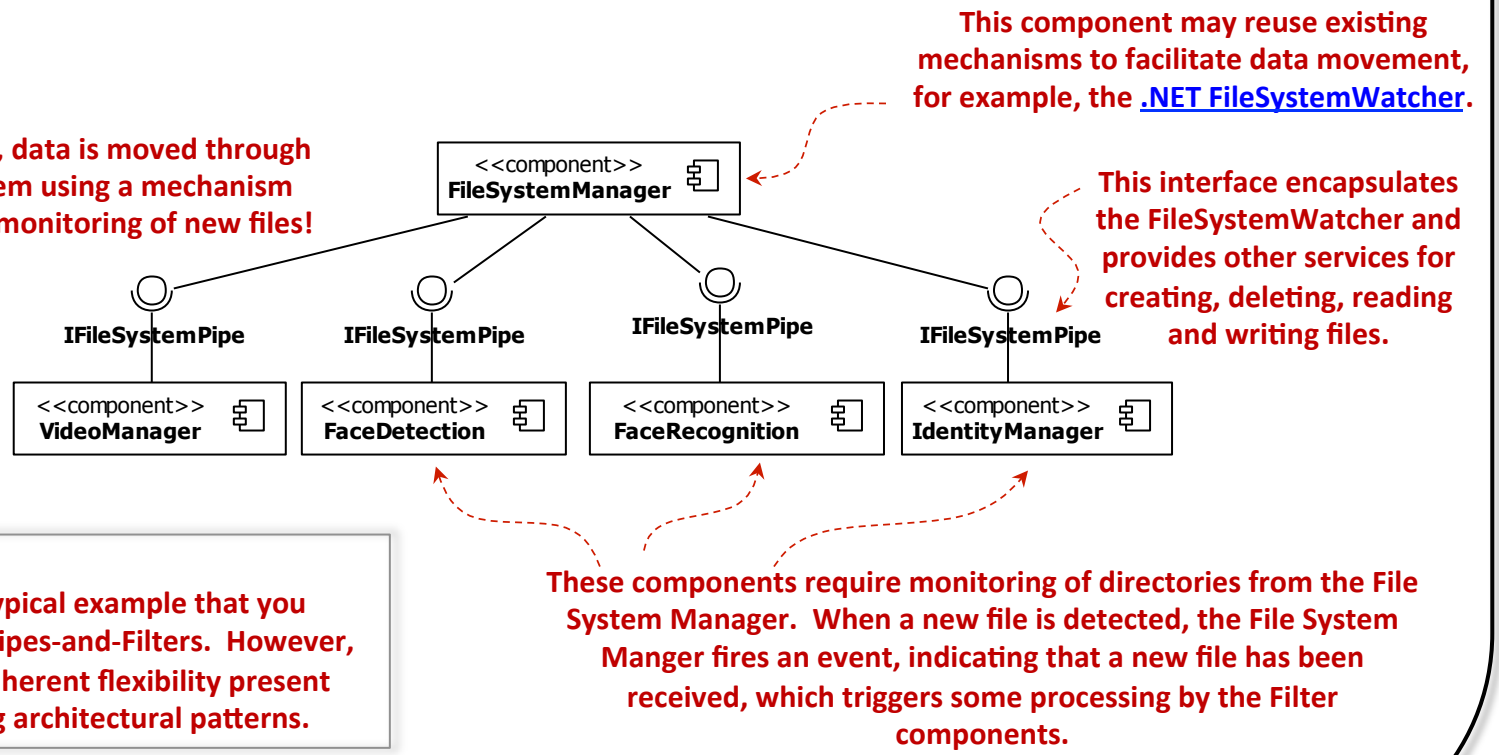
- Consider software that houses algorithms for **automatically determining the identity of an individual:**
  - ✓ The software access videos (with audio) from You Tube
  - ✓ The software detects faces of individuals in the video
    - Face detection is used to determine if a face is in the video
  - ✓ The software recognizes faces speech from the video
    - Face recognition is used to determine the identity of the person from the detected face.
  - ✓ Based on detection and recognition, the software predicts the identity of individuals in the video

- Using the pipes and filters architecture, the logical structure of the system can be modeled as follows:

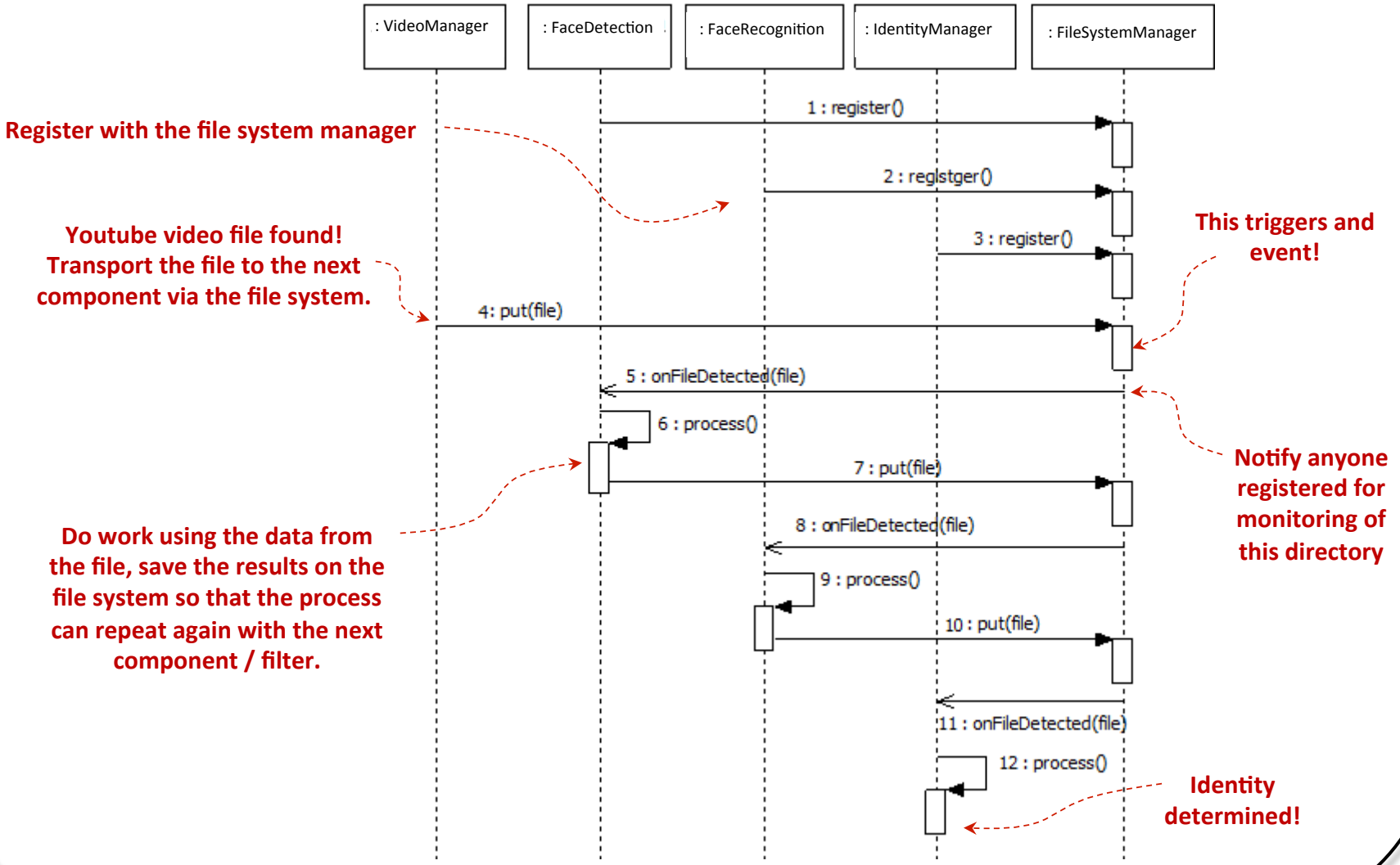


PIPES-AND-FILTERS ARCHITECTURAL PATTERN

- In the previous example, the box-and-line diagram was useful for visualizing the components in the system, however, it conveyed nothing about how data is transported from one Filter to the next, i.e., the Pipes.
  - ✓ Consider the following UML component for the same system

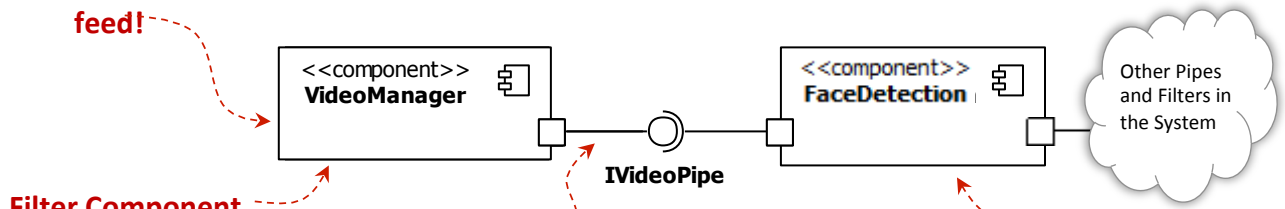


A more detailed example of the message exchanges in the example



Assume now that unlike the previous example, the Video component now interfaces with a camera for real-time video feed!

Consider the Pipes-and-Filters modeled this way



Filter Component

Pipe Interface

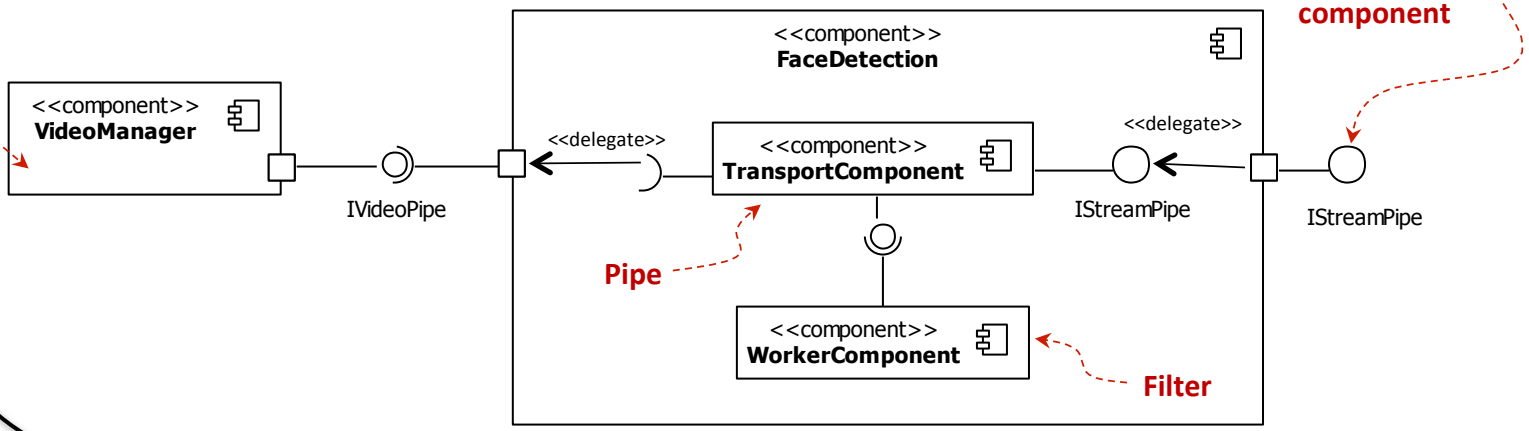
Filter Component

When modeled this way, there are implications about the internal structure of these components!

For example, see below

Similarly, since Pipes-and-Filters specify the separation between pipes and filters, there is an implication about the existence of both pipe and Filter component inside the Video Manager

Provided interface to transport the data stream to the next component



PIPES-AND-FILTERS ARCHITECTURAL PATTERN

- Quality properties of the Pipes-and-Filters architectural pattern include the ones specified below.

Quality	Description
Extensibility	Processing filters can be added easily for more capabilities.
Efficiency	By connecting filters in parallel, concurrency can be achieved to reduce latency in the system.
Reusability	By compartmentalizing pipes and filters, they can both be reused as-is in other systems.
Modifiability	Filters are compartmentalized and independent from each other; therefore, it is easy to add or remove filters to enhance the system.
Security	At any point during data-flow, security components can be injected to the work-flow to provide different types of security mechanisms to the data.
Maintainability	Allows for separation of concerns and independence of the Filters and Pipes; therefore, maintaining existing components becomes easier.

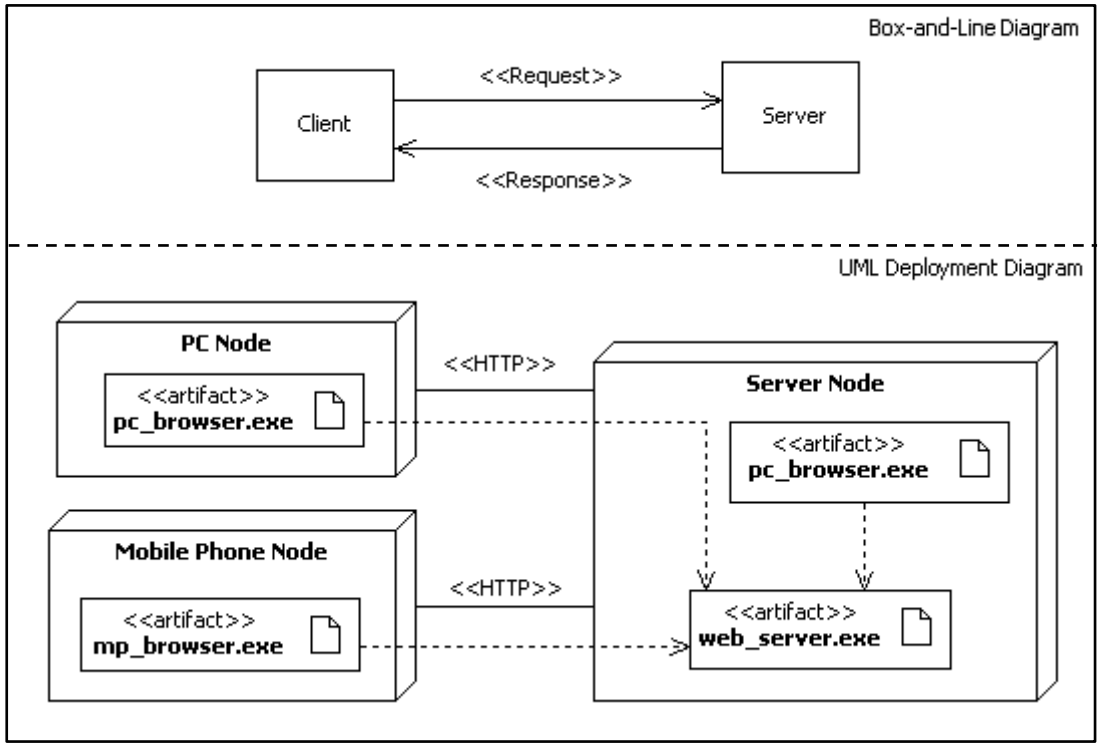
3. DISTRIBUTED SYSTEMS

- Distributed systems are decomposed into **multiple processes** that (typically) **collaborate through the network.**
  - ✓ These systems are ubiquitous in today's modern systems thanks to wireless, mobile, and internet technology.
    - In some distributed systems, one or more distributed processes perform work on behalf of client users and provide a bridge to some server computer, typically located remotely and performing work delegated to it by the client part of the system.
    - Other distributed systems may be composed of peer nodes, each with similar capabilities and collaborating together to provide enhanced services, such as music-sharing distributed applications.
  - ✓ These types of distributed systems are easy to spot, since their deployment architecture entails **multiple physical nodes.**
  - ✓ However, with the advent of multi-core processors, distributed architectures are also relevant to **software that executes on a single node with multiprocessor capability**.

- Some examples of distributed systems include:
  - ✓ **Internet systems, web services, file- or music-sharing systems, high-performance systems,** etc.

- Common architectural patterns for distributed systems include:
  1. **Client-Server Pattern**
  2. **Broker Pattern**

3.1 CLIENT-SERVER PATTERN



3.1 CLIENT-SERVER PATTERN

- Quality properties of the client-server pattern include the ones specified below.

Quality	Description
Interoperability	Allows clients on different platforms to interoperate with servers of different platforms.
Modifiability	Allows for centralized changes in the server and quick distribution among many clients.
Availability	By separating server data, multiple server nodes can be connected as backup to increase the server data or services' availability.
Reusability	By separating server from clients, services or data provided by the server can be reused in different applications.

3.2 BROKER ARCHITECTURAL PATTERN

- The Broker architectural pattern provides mechanisms for **achieving better flexibility** between clients and servers in a distributed environment.
  - ✓ In a typical client-server system, **clients are tightly coupled with servers**.
  - ✓ This leads to complexity for systems that need to provide services from multiple servers hosted at different locations.

- In some software systems, clients (in a distributed environment) need to be able to access services from multiple servers without known their actual locations or particular details of communication.
  - ✓ When these concerns are separated, it leads to systems that are flexible and interoperable.

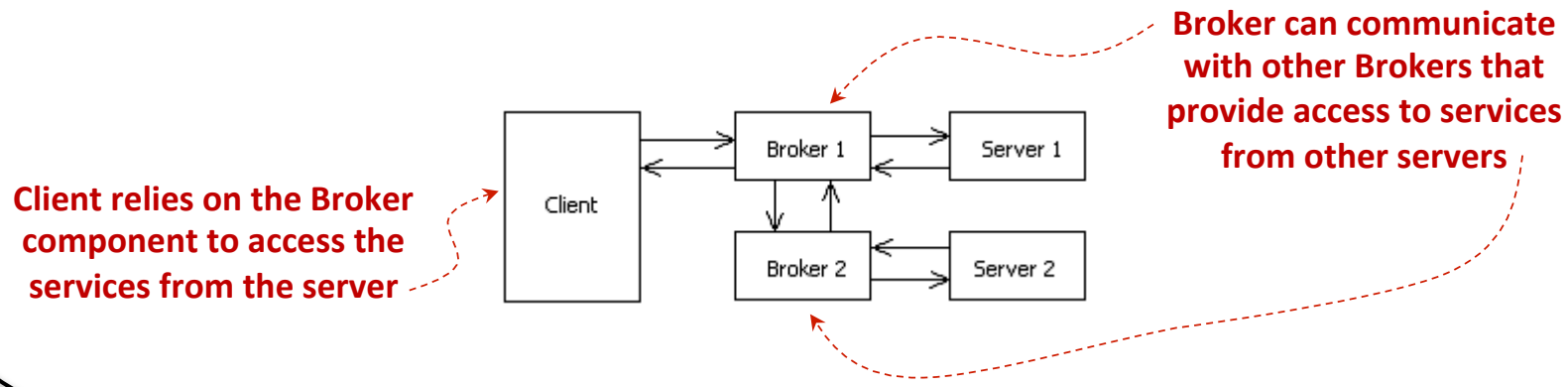
- The broker pattern **decreases coupling** between **client and servers** by **mediating** between them so that **one client can transparently access the services of multiple servers**

3.2 Broker Architectural Pattern

- The Broker architectural pattern includes the following components:

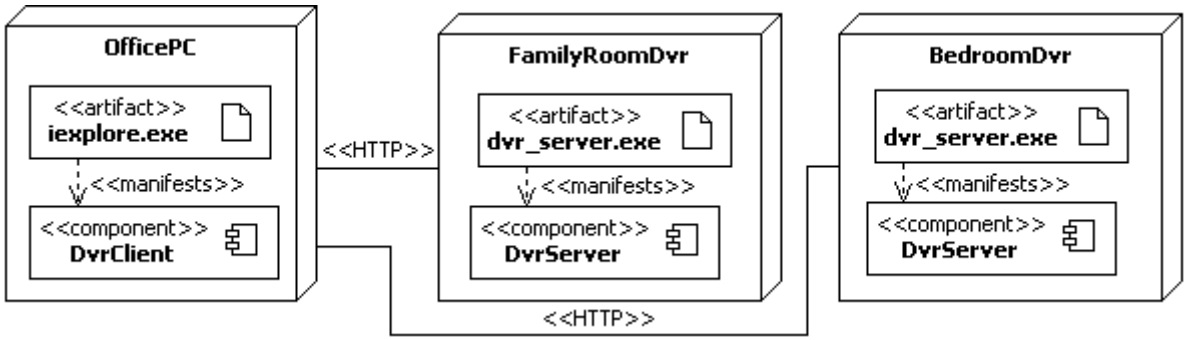
Component	Description
Client	Applications that use the services provided by one or more servers.
ClientProxy	Component that provides transparency (at client) between remote and local components so that remote components appear as local ones.
Broker	Component that mediates between client and server components.
ServerProxy	Component that provides transparency (at server) between remote and local components so that remote components appear as local ones.
Server	Provide services to clients. May also act as client to the Broker.
Bridge	Optional component for encapsulating interoperation among Brokers.

- An box-and-line example of the broker architecture is presented below.



3.2 Broker Architectural Pattern

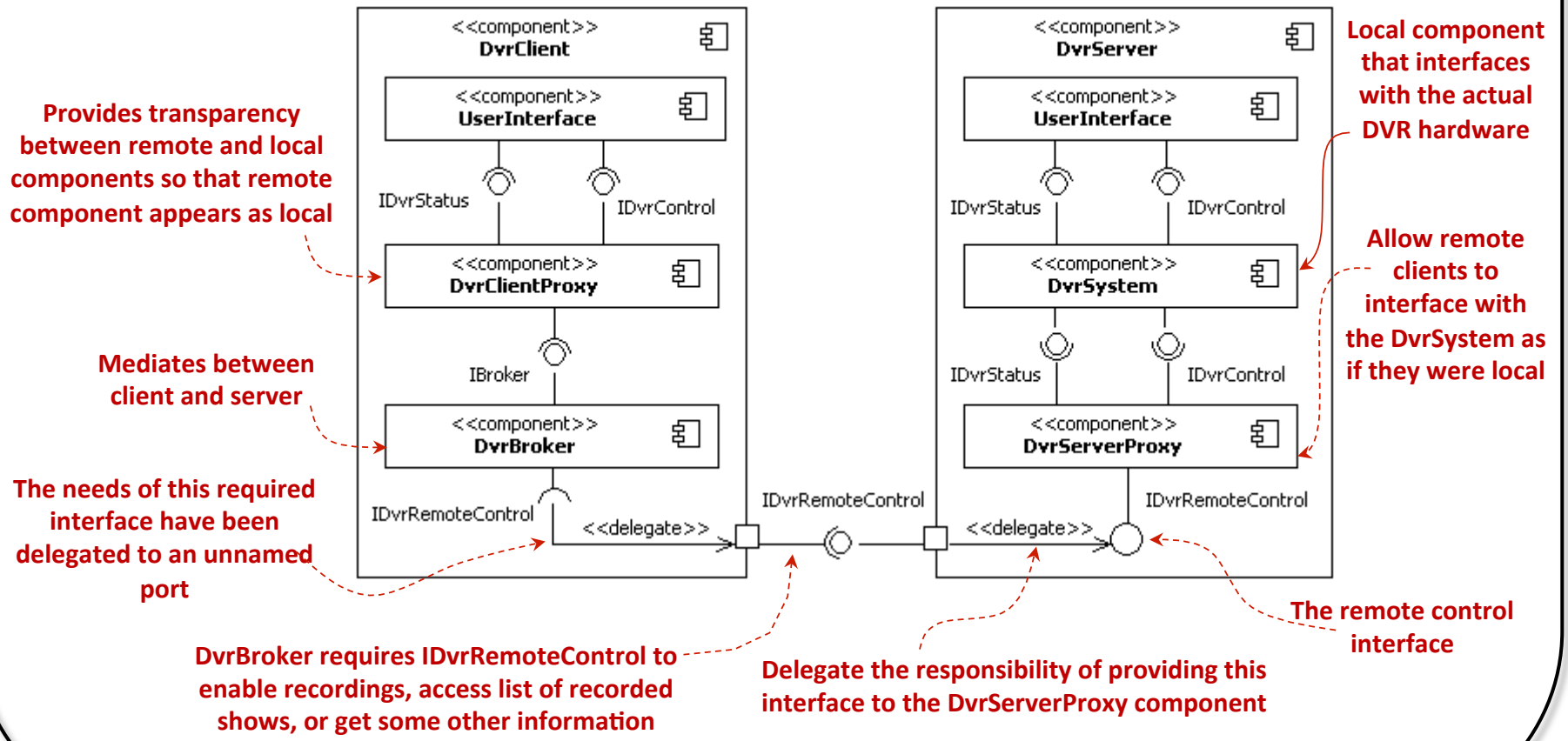
- Consider the typical **Digital Video Recording (DVR) system** found in many houses today.
  - ✓ In this example, there are two DVRs
    - One in the family room
    - The other in a bedroom
  - ✓ DVR services can be accessed from inside the house, using the home office PC.
    - This could be done in many ways, but for simplicity, assume that access is obtained via the browser, using a Java Applet, Microsoft Active X, etc.
    - To gain some insight into the physical architecture of the system, take a look at the following deployment diagram.



3.2 Broker Architectural Pattern

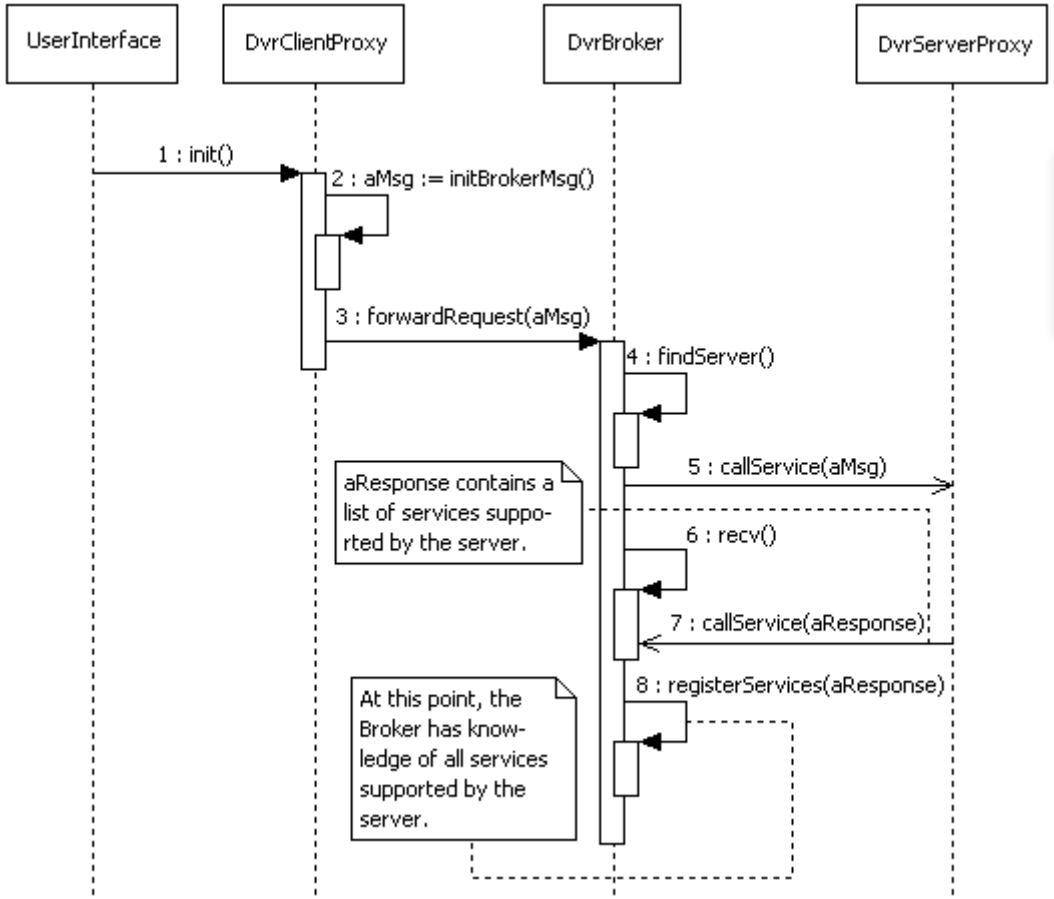
Important:
Remember, this is NOT the physical view of the system!!

- The logical architecture can be designed as seen below:



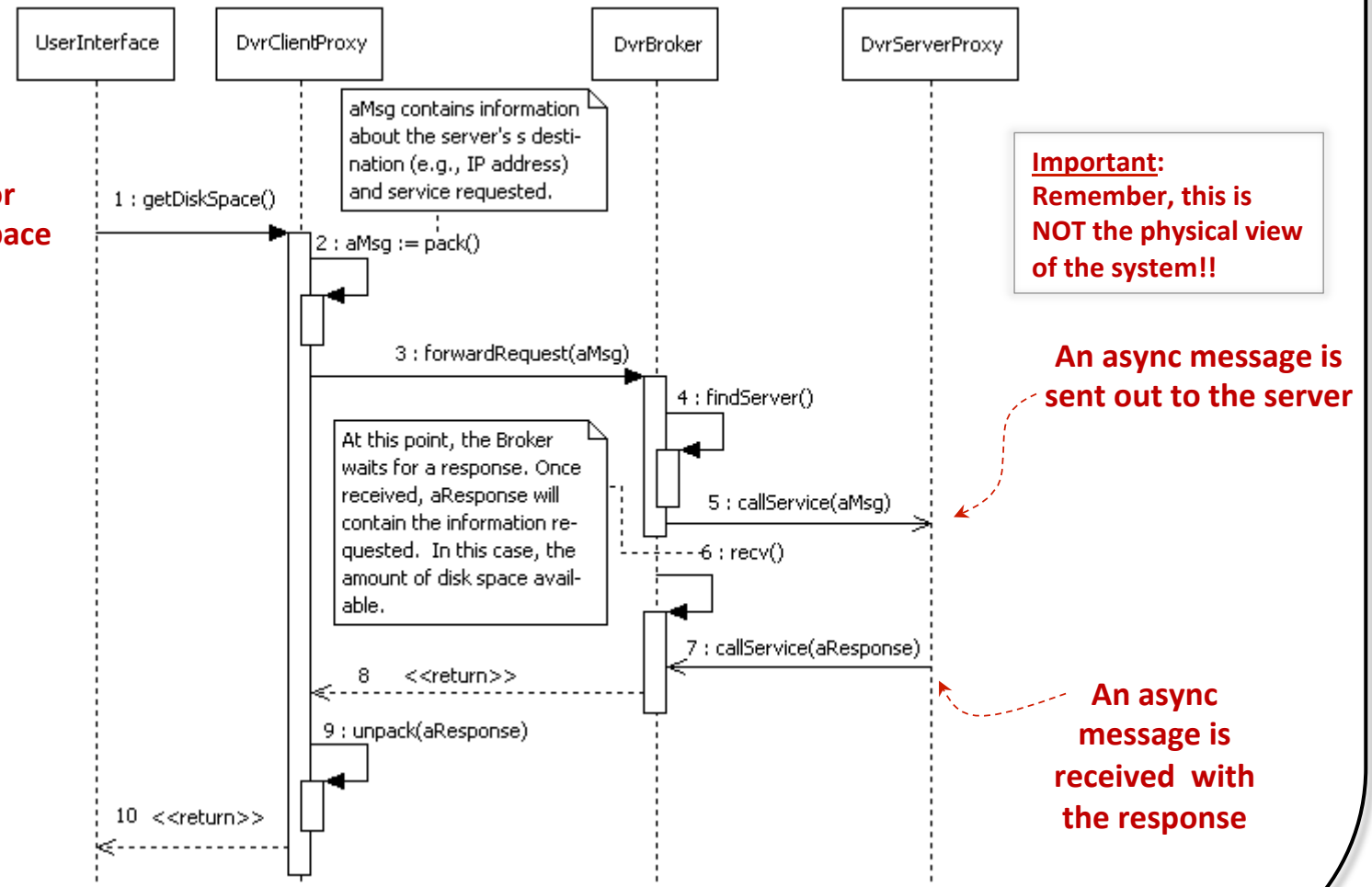
3.2 Broker Architectural Pattern

Initializing the Broker System



Important:
Remember, this is NOT the physical view of the system!!

3.2 Broker Architectural Pattern



Requesting for available disk space

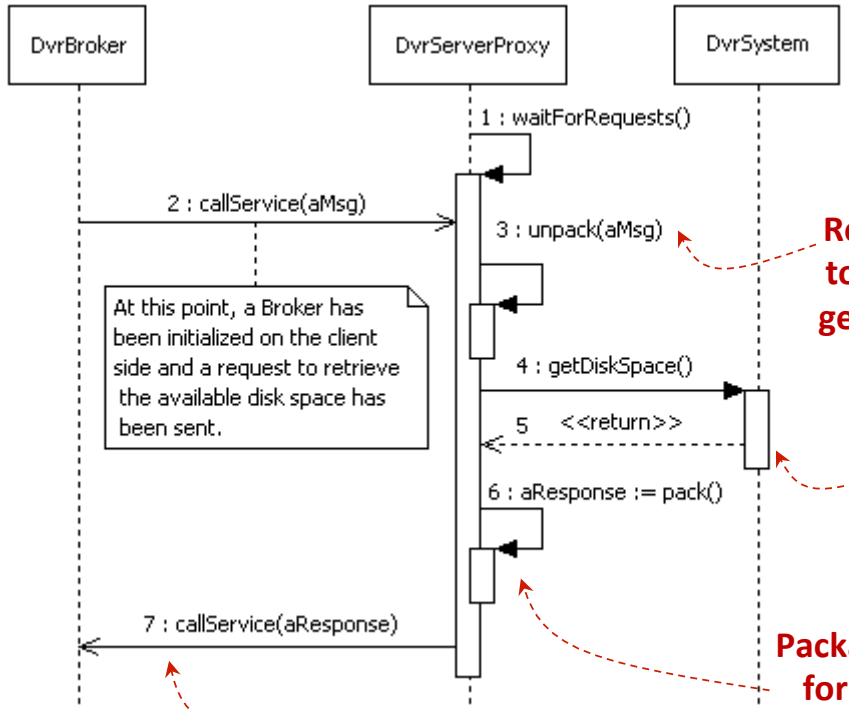
Important:
Remember, this is NOT the physical view of the system!!

An async message is sent out to the server

An async message is received with the response

3.2 Broker Architectural Pattern

Meanwhile, on the server side…



Retrieve the message id to determine that it is a get free space command

Requests the free space data from the actual DVR system

Package the response in whatever format the system requires for transmission over the network

Send the response to the broker

3.2 Broker Architectural Pattern

- Quality properties of the Broker architectural pattern include the ones specified below.

Quality	Description
Interoperability	Allows clients on different platforms to interoperate with servers of different platforms. Also, allows clients to interoperate (transparently) with multiple servers.
Modifiability	Allows for centralized changes in the server and quick distribution among many clients.
Portability	By porting the broker to different platforms, services provided by the system can be easily acquired by new clients in different platforms.
Reusability	Brokers abstract many system calls required for providing communication between nodes. When using brokers, many complex services can be reused in other applications that require similar distributed operations.

SUMMARY…

- In this session, we presented fundamentals concepts of data-centered , data flow, and distributed systems, together with essential architectural patterns for these systems, including:
  - ✓ Blackboard
  - ✓ Pipes-and-Filters
  - ✓ Client-server
  - ✓ Broker

Session III:
System : Interactive and Hierarchical

SESSION'S AGENDA

4. Interactive Systems

- ✓ Overview
- ✓ Patterns
  - **Model-View-Controller**

5. Hierarchical Systems

- ✓ **Main program and Subroutine**
- ✓ **Layered**

4. INTERACTIVE SYSTEMS

- Interactive systems support **user interactions, typically through user interfaces**.
  - ✓ When designing these systems, two main quality attributes are of interest:
    - **Usability**
    - **Modifiability**

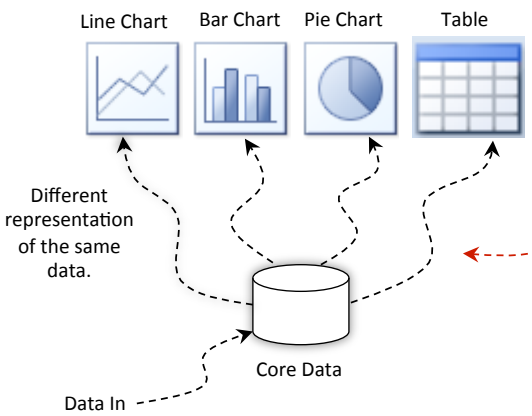
- The mainstream architectural pattern employed in most interactive systems is the **Model-View-Controller (MVC).**

- The MVC pattern is used in interactive applications that require **flexible incorporation of human-computer interfaces.** With the MVC, systems are decomposed into three main types of components:

Component	Description
Model	Component that represents the system's core, including its major processing capabilities and data.
View	Component that represents the output representation of the system (e.g., graphical output or console-based).
Controller	Component (associated with a view) that handles user inputs.

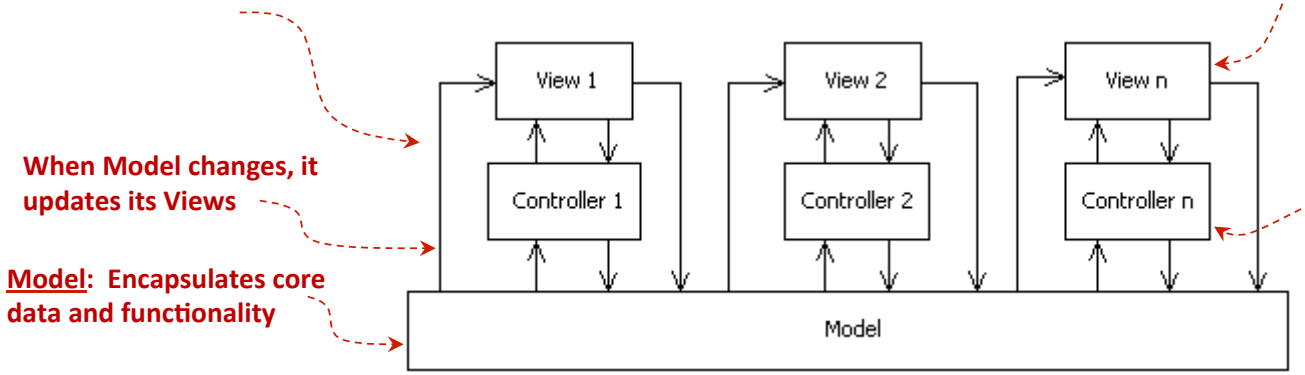
Model-View-Controller Architectural Pattern

Consider the popular example where data needs to be represented in different formats



When data changes, all views are updated to reflect the changes.

Box-and-line diagram of the MVC architectural pattern

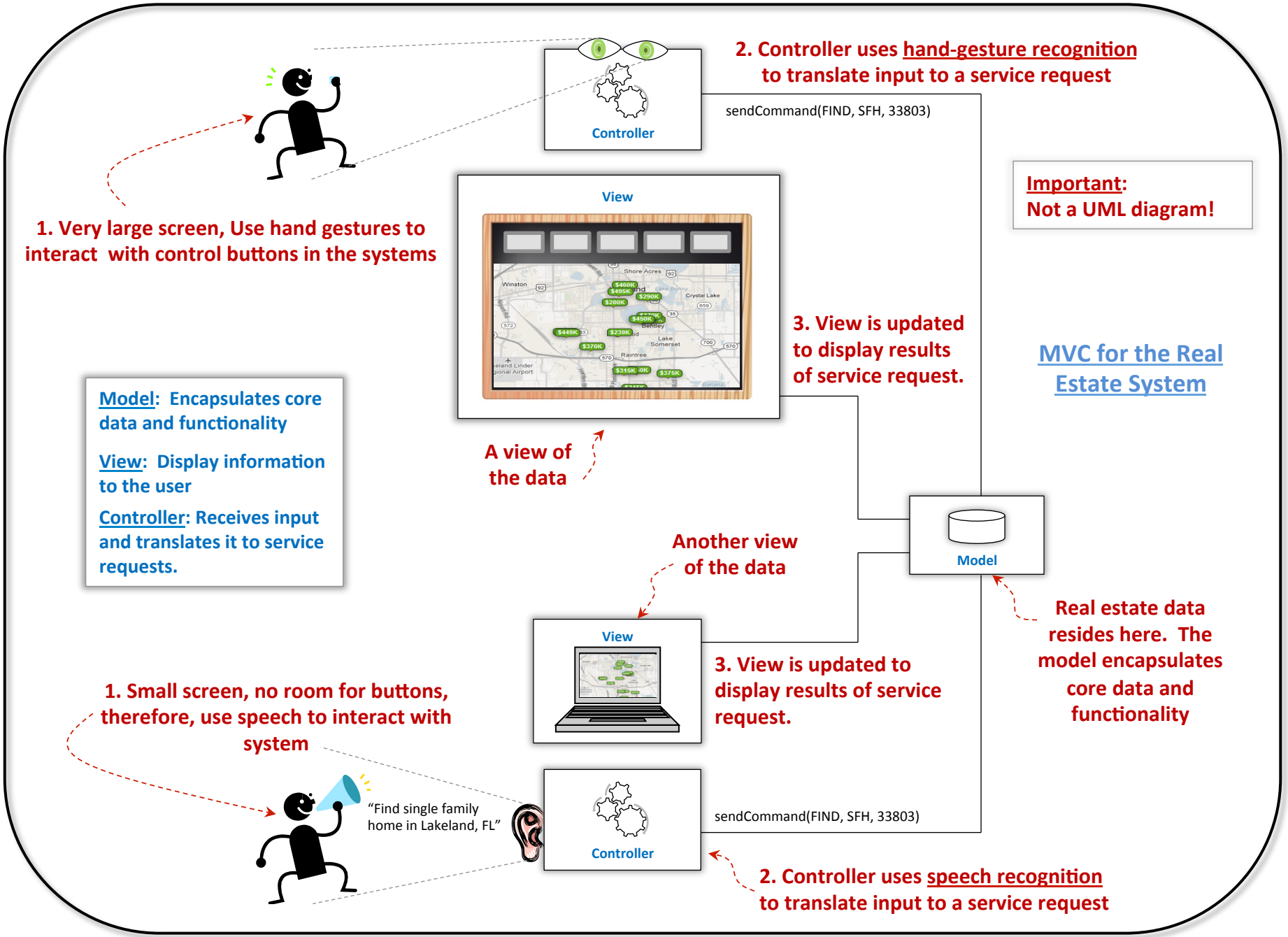


When Model changes, it updates its Views

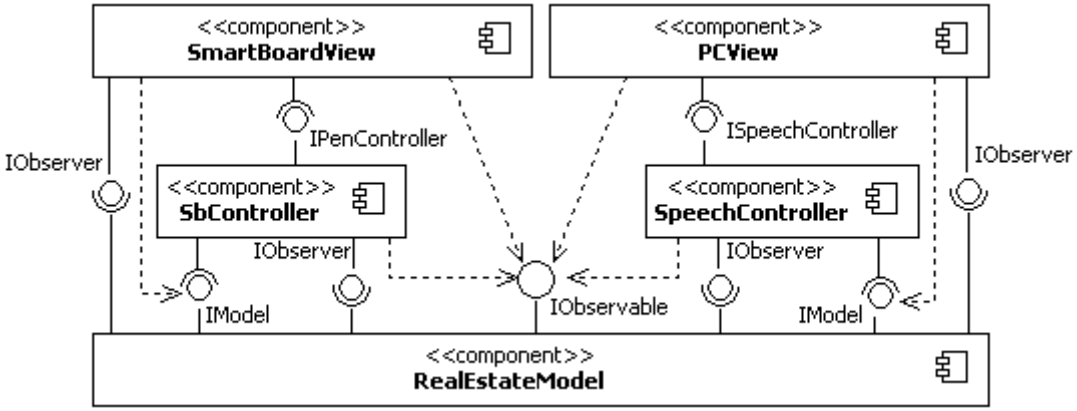
Model: Encapsulates core data and functionality

View: Display information to the user. Every View is associated with a Controller.

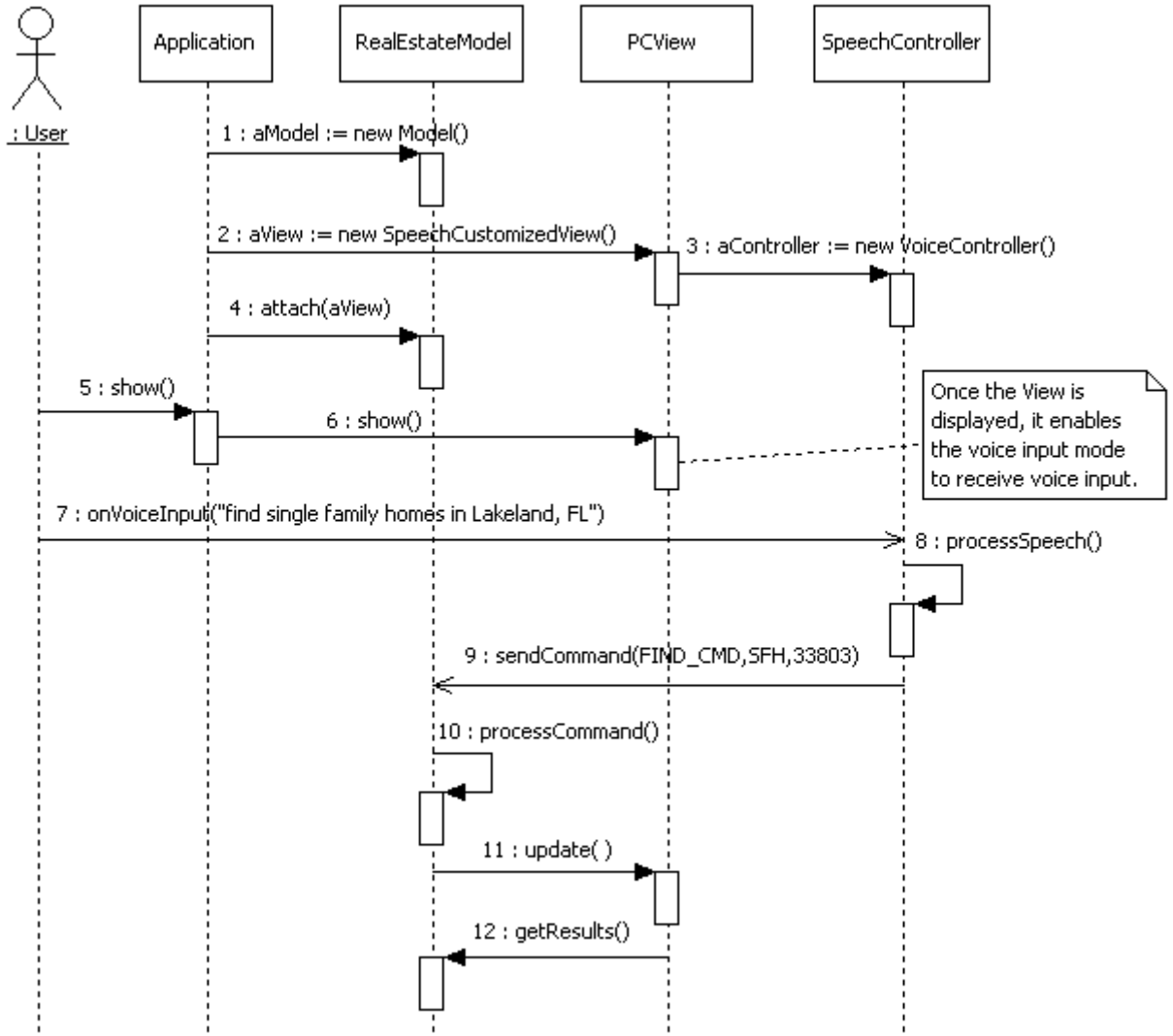
Controller: Receives input and translates it to service requests.


1. Very large screen, Use hand gestures to interact with control buttons in the systems
Controller
2. Controller uses hand-gesture recognition to translate input to a service request
sendCommand(FIND, SFH, 33803)
Important:
Not a UML diagram!
View
3. View is updated to display results of service request.
A view of the data
MVC for the Real Estate System
Model: Encapsulates core data and functionality
View: Display information to the user
Controller: Receives input and translates it to service requests.
Another view of the data
Model
Real estate data resides here. The model encapsulates core data and functionality
View
3. View is updated to display results of service request.
1. Small screen, no room for buttons, therefore, use speech to interact with system
"Find single family home in Lakeland, FL"
Controller
sendCommand(FIND, SFH, 33803)
2. Controller uses speech recognition to translate input to a service request

MODEL-VIEW-CONTROLLER ARCHITECTURAL PATTERN



MODEL-VIEW-CONTROLLER ARCHITECTURAL PATTERN



MODEL-VIEW-CONTROLLER ARCHITECTURAL PATTERN

- Quality properties of the MVC architectural pattern include the ones specified below.

Quality	Description
Modifiability	Easy to exchange, enhance, or add additional user interfaces.
Usability	By allowing easy exchangeability of user interfaces, systems can be configured with different user interfaces to meet different usability needs of particular groups of customers.
Reusability	Be separating the concerns of the model, view, and controller components, the can all be reused in other systems.

- There are many variations of the MVC architectural pattern.
  - ✓ One popular variation includes the fusion of views and controller components, as made famous in the 1990s by Microsoft's Document-View architecture
  - ✓ Other more extensive variations include the process-abstraction-controller pattern.

- MVC has been successfully adapted as an architecture for the World Wide Web, e.g., see:
  - ✓ JavaScriptMVC
  - ✓ Backbone

5. HIERARCHICAL SYSTEMS

- Hierarchical systems can be **decomposed and structured in hierarchical fashion**. Two common architectural patterns for hierarchical systems are:
  1. **Main program and subroutine**
  2. **Layered**

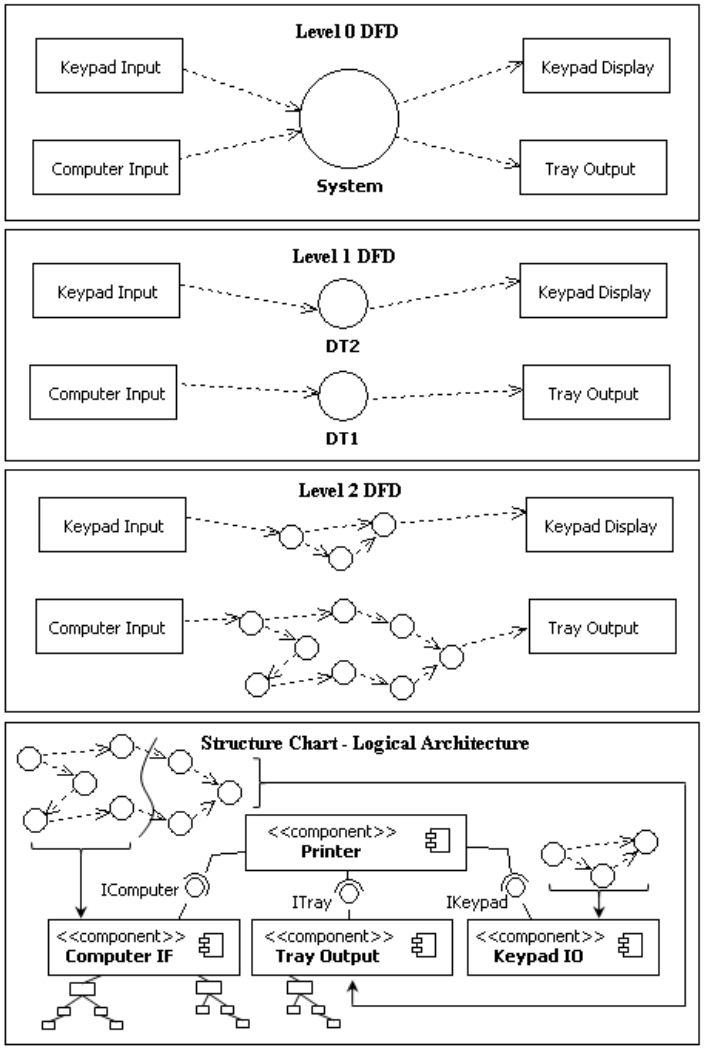
- Quality properties of the **Main Program and Subroutine** architectural pattern include:

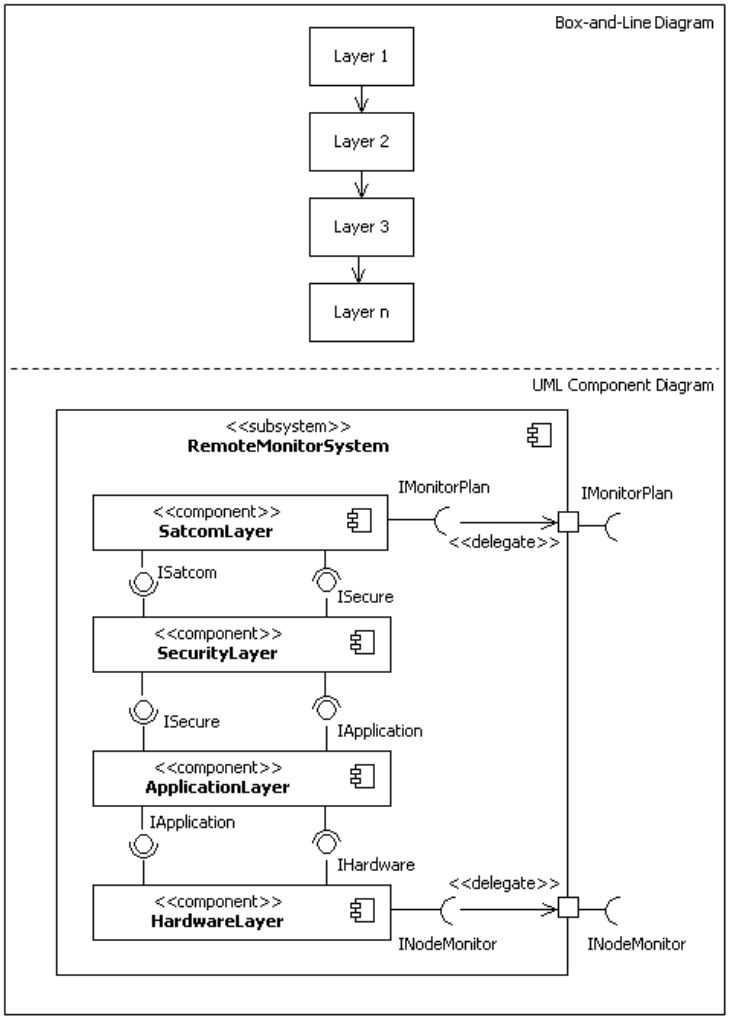
Quality	Description
Modifiability	By decomposing the system into independent, single purpose components, each component becomes easier to understand and manage.
Reusability	Independent, finer-grained components can be reused in other systems.

- Quality properties of the **Layered** architectural pattern include the ones specified below.

Quality	Description
Modifiability	Dependencies are kept local within layer components. Since components can only access other components through a well-defined and unified interface, the system can be modified easily by swapping layer components with other enhanced or new layer components.
Portability	Services that deal directly with platform's API's can be encapsulated using a system layer component. Higher level layers rely on this component for providing system services to the application, therefore, by porting the system's API layer to other platforms systems become more portable.
Security	The controlled hierarchical structure of layered systems allow for easy incorporation of security components to encrypt/decrypt incoming/outgoing data.
Reusability	By compartmentalizing each layer's services, they become easier to reuse.

Main Program and Subroutine and Layered Patterns





SUMMARY…

- In this session, we continued the discussion on distributed systems and presented fundamentals concepts of interactive and hierarchical systems, together with essential architectural patterns for these systems, including:
  - ✓ MVC
  - ✓ Layered
  - ✓ Main Program and Subroutine

REFERENCES

- [1] Alexander, Christopher, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, 1977.
- [2] Clements, Paul, Rick Kazman, and Mark Klein. *Evaluating Software Architectures*. Santa Clara, CA: Addison Wesley, 2001.