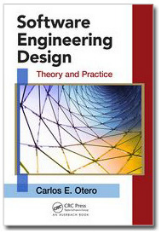
Chapter 4: Patterns and Styles in Software Architecture

Session II: Data-Centered, Data-Flow, and Distributed Systems

Software Engineering Design: Theory and Practice
by Carlos E. Otero



SESSION'S AGENDA

- ➢ Data-Centered Systems
  - ✓ Overview
  - ✓ Patterns
    - ▪ Blackboard

- ➢ Data Flow Systems
  - ✓ Overview
  - ✓ Patterns
    - ▪ Pipes-and-Filters

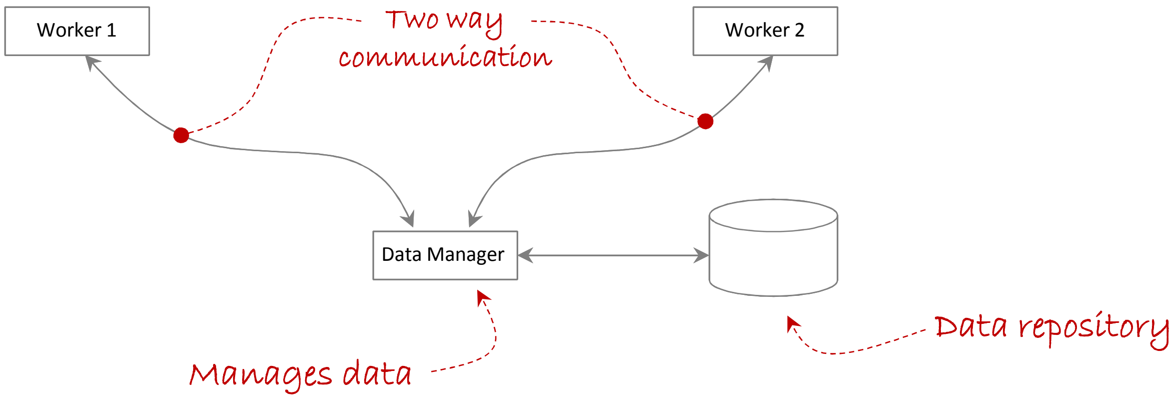
- ➢ Distributed Systems
  - ✓ Overview
  - ✓ Patterns
    - ▪ Client Server

- ➢ What's next…
  - ✓ Distributed systems – Broker Pattern
  - ✓ Interactive Systems
  - ✓ Hierarchical Systems

DATA-CENTERED SYSTEMS

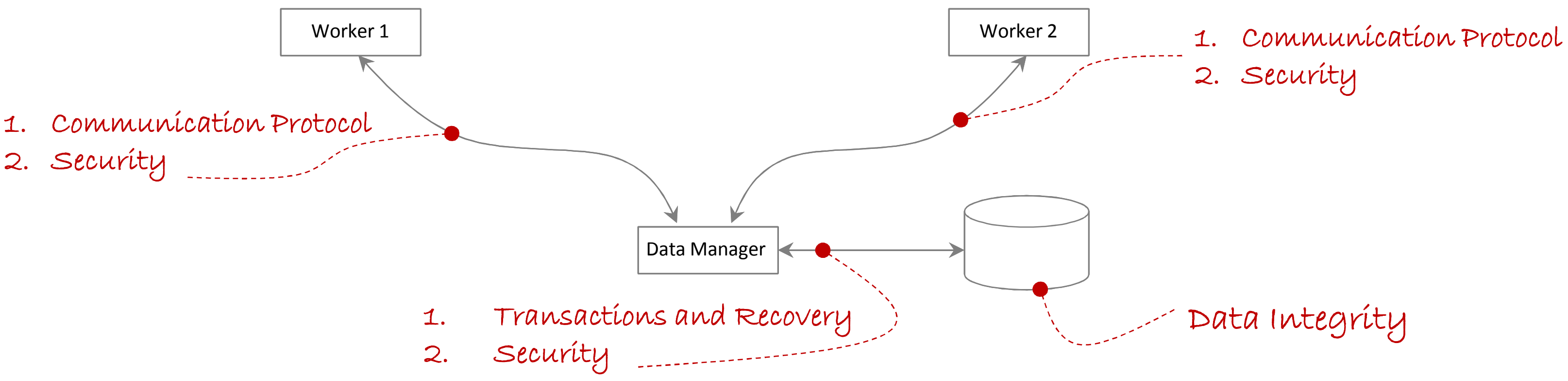
- Data-centered systems are systems primarily decomposed around a main central repository of data. These include:
  - ✓ Data management component
    - ▪ The data management component controls, provides, and manages access to the system's data.
  - ✓ Worker components
    - ▪ Worker components execute 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.



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


Worker 1
Worker 2
Data Manager
1. Communication Protocol
2. Security
1. Communication Protocol
2. Security
1. Transactions and Recovery
2. Security
Data Integrity

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

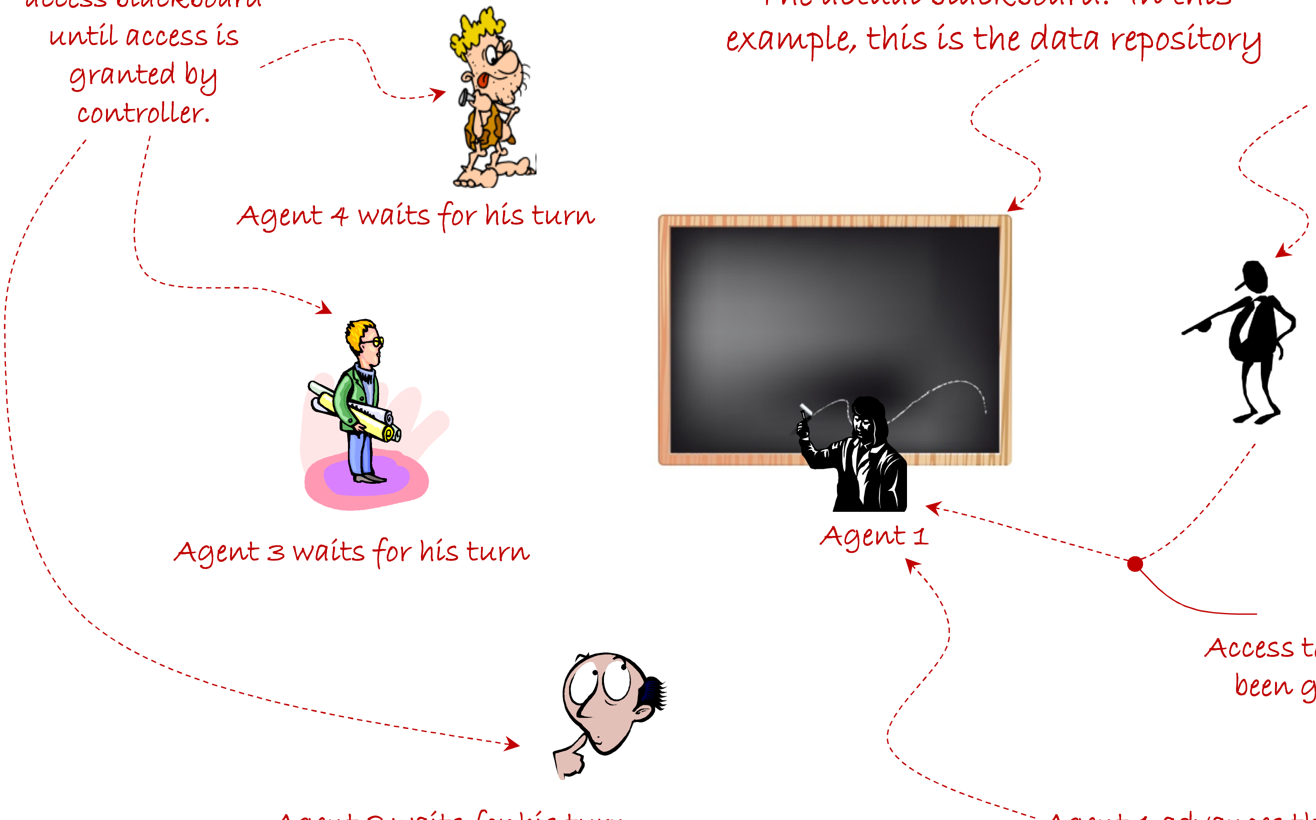
BLACKBOARD ARCHITECTURAL PATTERN

- 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



Agents cannot access blackboard until access is granted by controller.

The actual blackboard. In this example, this is the data repository

Controller

Agent 4 waits for his turn

Agent 1

Agent 3 waits for his turn

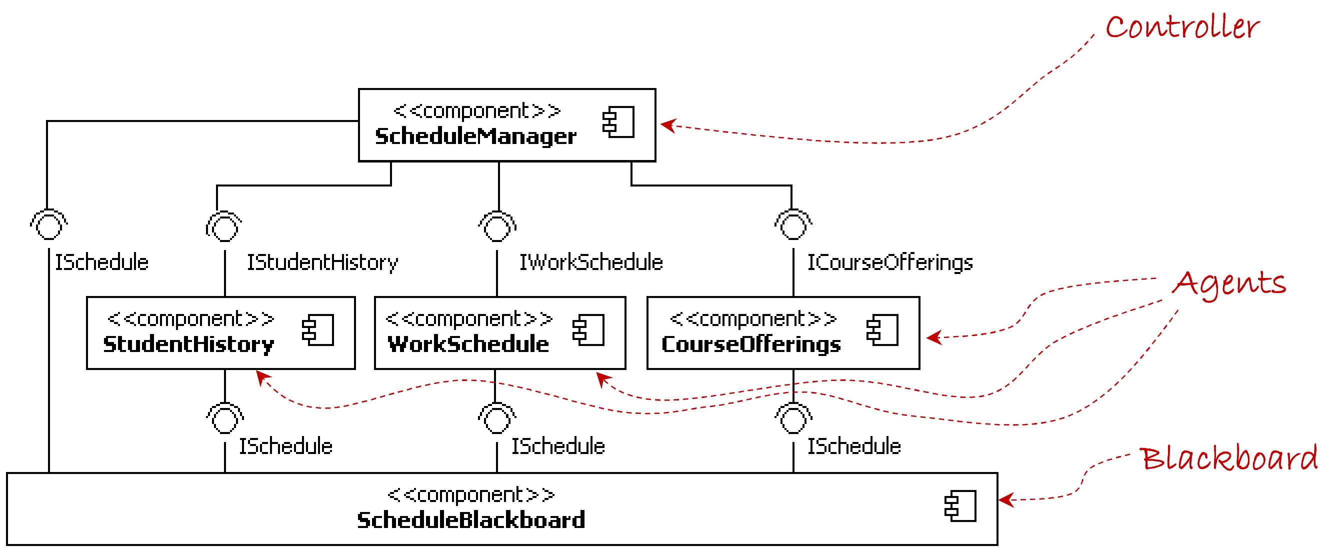
Access to the blackboard has been granted to Agent 1

Agent 2 waits for his turn

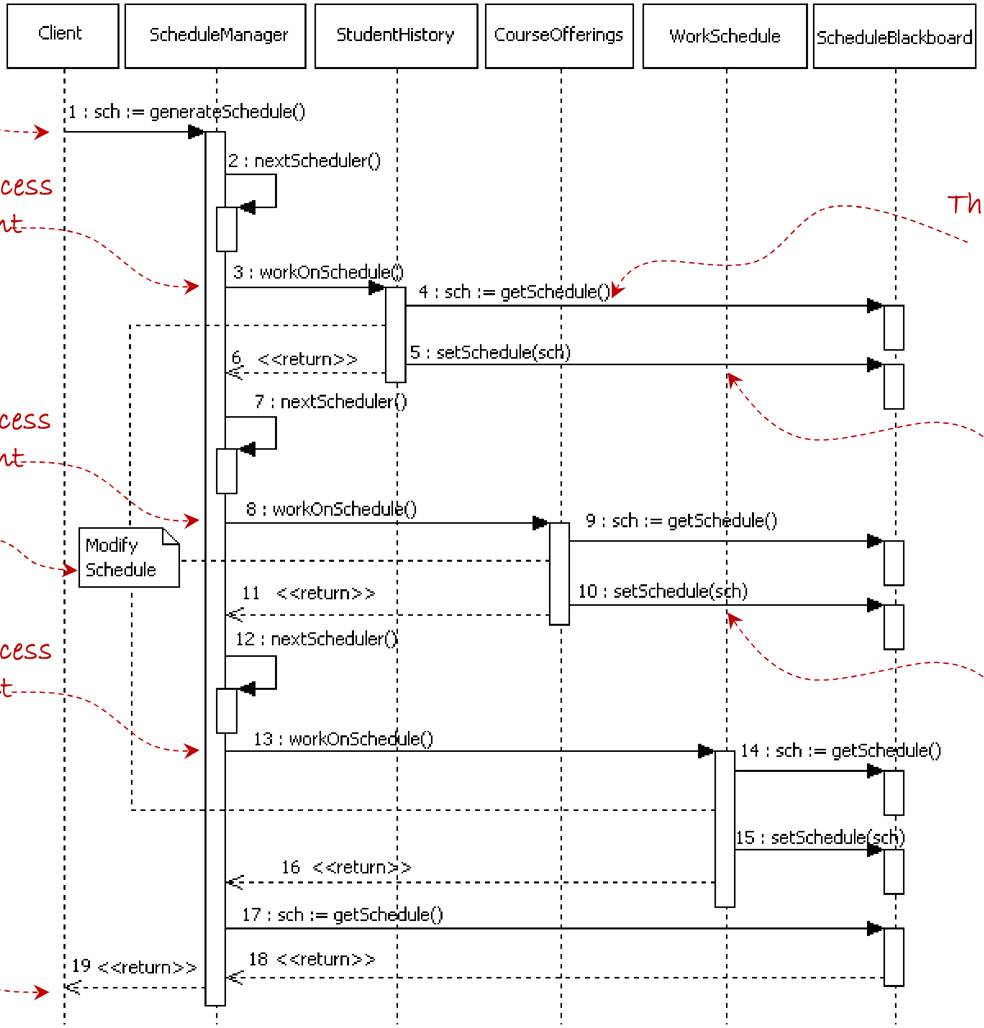
Agent 1 advances the solution!

BLACKBOARD ARCHITECTURAL PATTERN

- Consider the Students' Scheduling System from Chapter 4.



Blackboard Architectural Pattern



Client requests a schedule

Controller grants schedule access to the StudentHistory agent

Controller grants schedule access to the CourseOfferings agent

This note provides important information!

Controller grants schedule access to the WorkSchedule agent

Client receives an optimized schedule

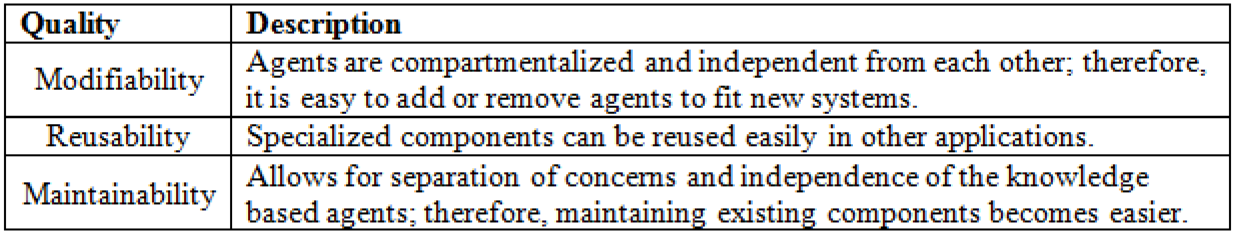
The StudentHistory agent retrieves the data

The StudentHistory agent modifies the schedule and stores the results back

The CourseOfferings agent retrieves, modifies, and stores the schedule back

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.

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:
    - Worker components, those that perform work on data
    - 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*.

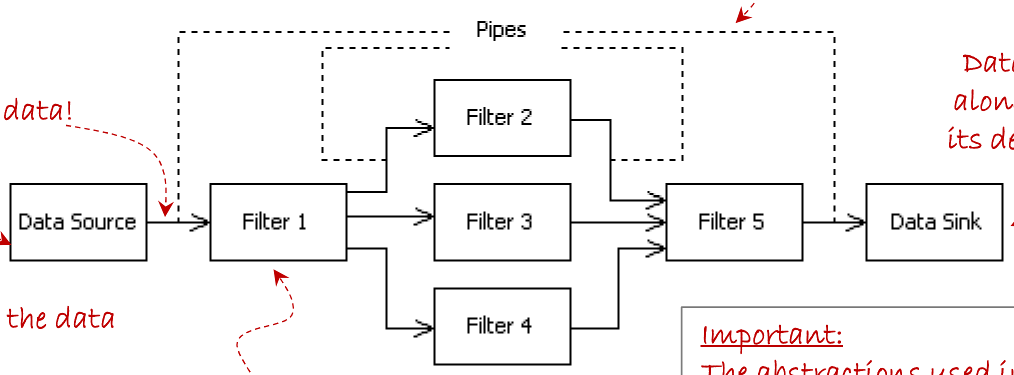
PIPES-AND-FILTERS ARCHITECTURAL PATTERN

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

Not an UML diagram!

Pipes used to move the data!

Data is moved and processed along the way until it reaches its destination, the Data Sink.



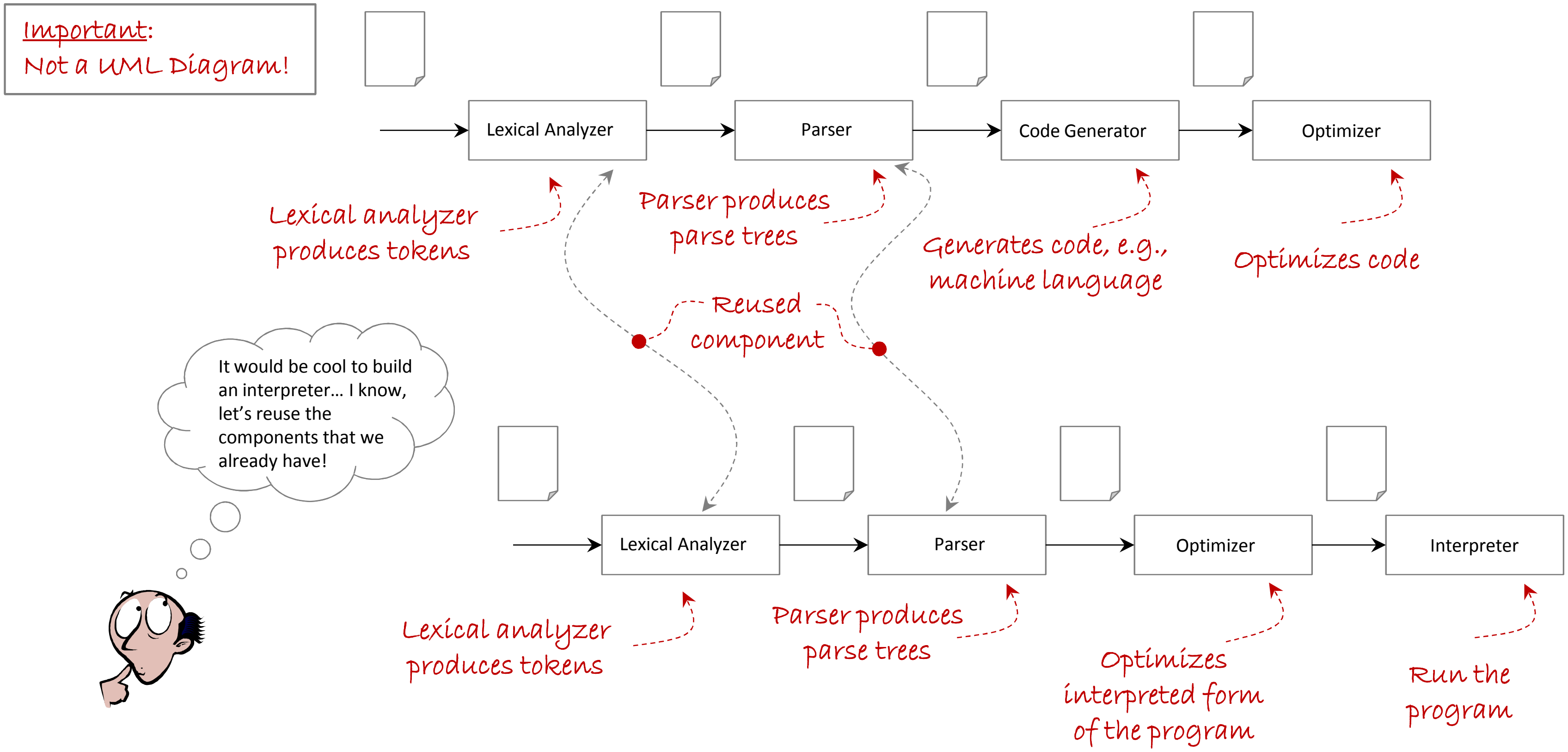
Data source produces the data

Filters work on the data!

Important:
The abstractions used in the box-and-line diagram above for Pipes can be deceiving, since they really do not convey the real identity of these mechanisms!

Pipes-and-Filters Architectural Pattern

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



Important:
Not a UML Diagram!

Lexical Analyzer → Parser → Code Generator → Optimizer

Lexical analyzer produces tokens

Parser produces parse trees

Generates code, e.g., machine language

Optimizes code

Reused component

It would be cool to build an interpreter… I know, let's reuse the components that we already have!

Lexical Analyzer → Parser → Optimizer → Interpreter

Lexical analyzer produces tokens

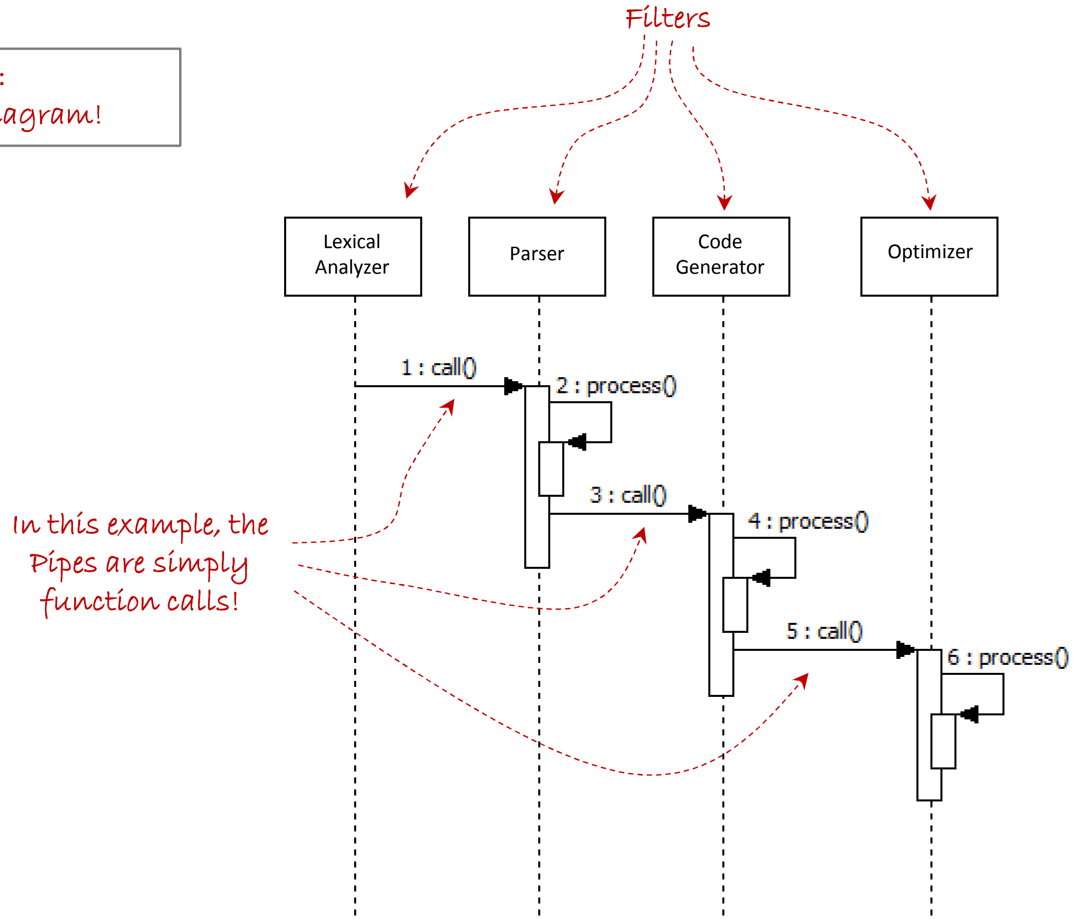
Parser produces parse trees

Optimizes interpreted form of the program

Run the program

PIPES-AND-FILTERS ARCHITECTURAL PATTERN

Important:
A UML Diagram!



PIPES-AND-FILTERS ARCHITECTURAL PATTERN

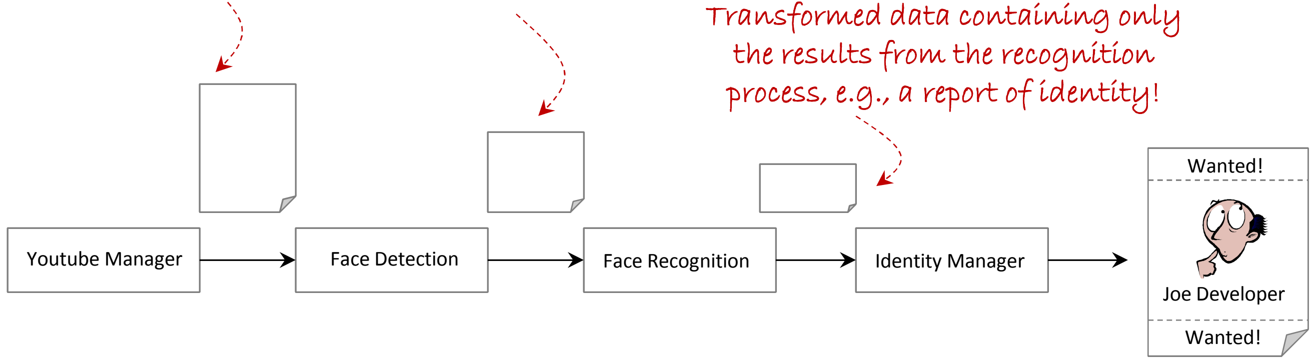
- Consider software that houses algorithms for automatically determining the identity of an individual:
  - ✓ The software access videos (with audio) from You Tube
  - ✓ The software detects faces of individuals in the video
    - Face detection is used to determine if a face is in the video
  - ✓ The 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:

Important:
Consider what would happen if a better algorithm for recognition is discovered?

Big video file!

Transformed data containing only the information from detected faces!

Transformed data containing only the results from the recognition process, e.g., a report of identity!



Youtube Manager

Face Detection

Face Recognition

Identity Manager

Wanted!

Joe Developer

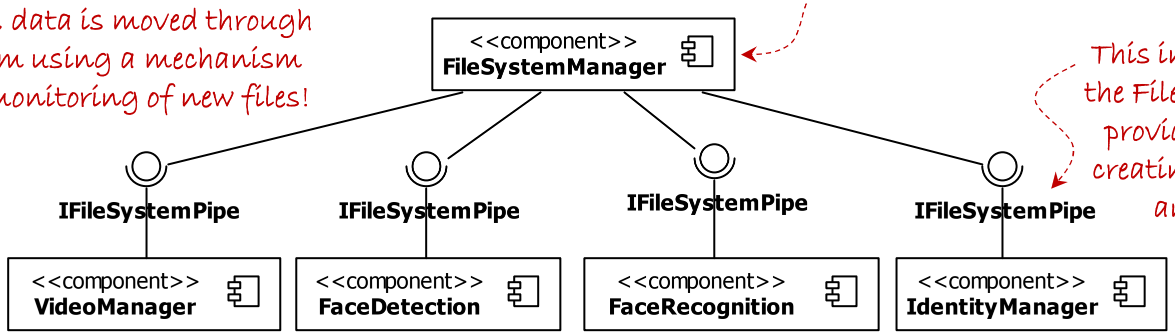
Wanted!

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

This component may reuse existing mechanisms to facilitate data movement, for example, the .NET FileSystemWatcher.

In this system, data is moved through the File System using a mechanism that relies on monitoring of new files!

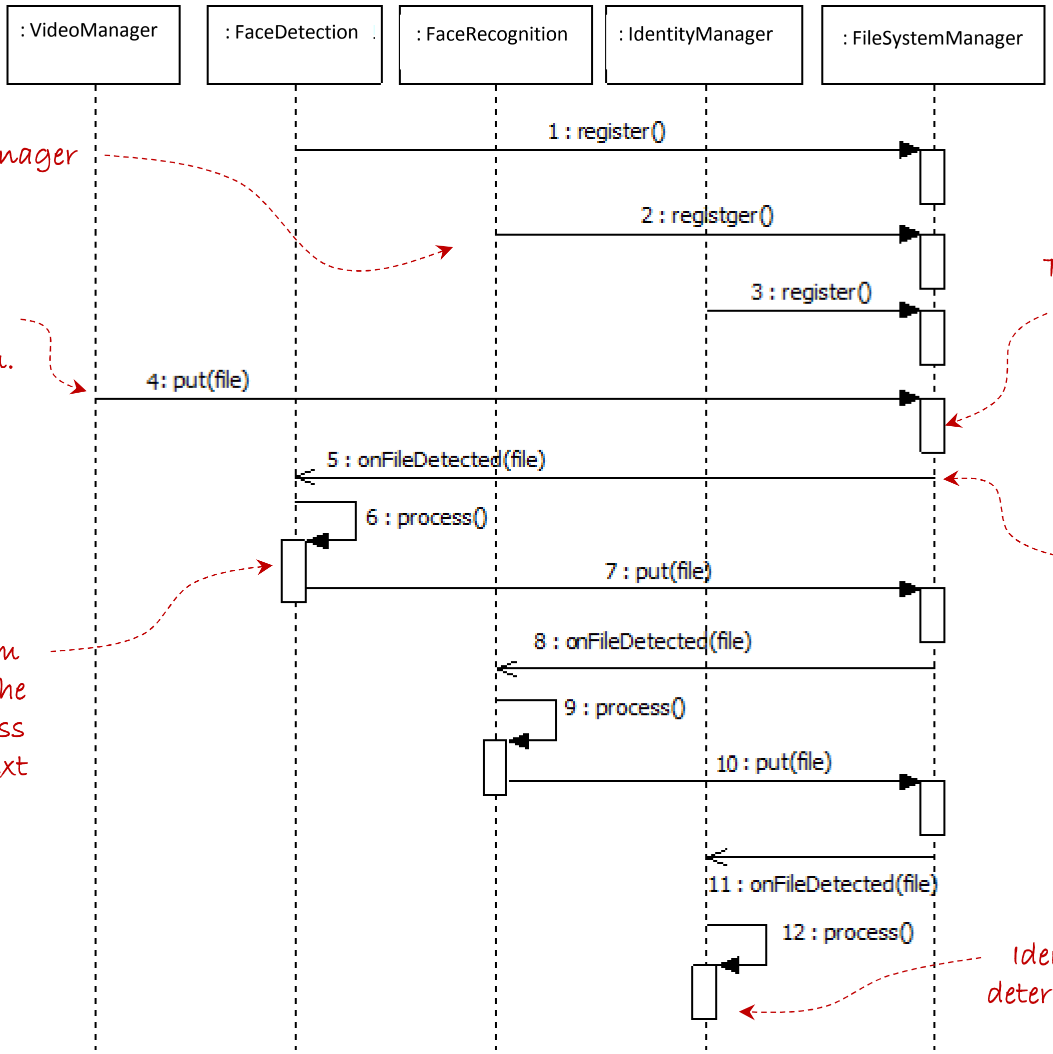


This interface encapsulates the FileSystemWatcher and provides other services for creating, deleting, reading and writing files.

Warning:
This is not the typical example that you would find for Pipes-and-Filters. However, it displays the inherent flexibility present when employing architectural patterns.

These components require monitoring of directories from the File System Manager. When a new file is detected, the File System Manger fires an event, indicating that a new file has been received, which triggers some processing by the Filter components.

A more detailed example of the
message exchanges in the example



Register with the file system manager

Youtube video file found!
Transport the file to the next
component via the file system.

This triggers
and event!

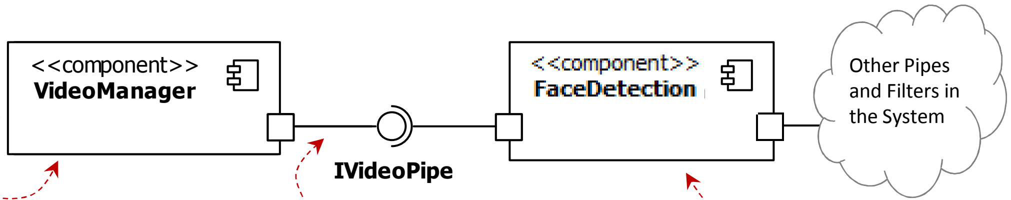
Notify anyone
registered for
monitoring of
this directory

Do work using the data from
the file, save the results on the
file system so that the process
can repeat again with the next
component / filter.

Identity
determined!

Consider the Pipes-and-Filters modeled this way

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



Filter Component

Pipe Interface

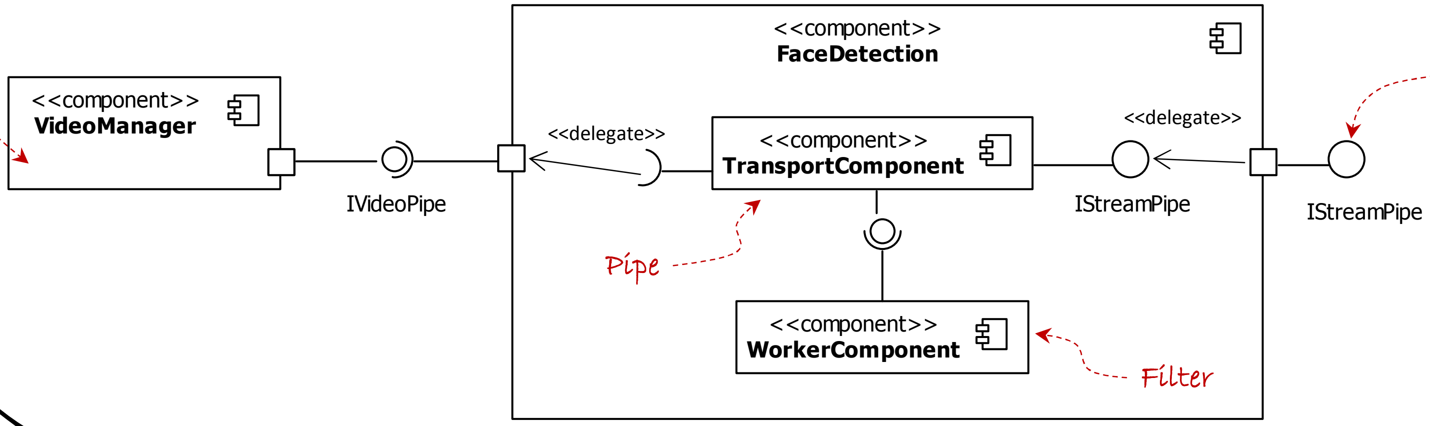
Filter Component

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

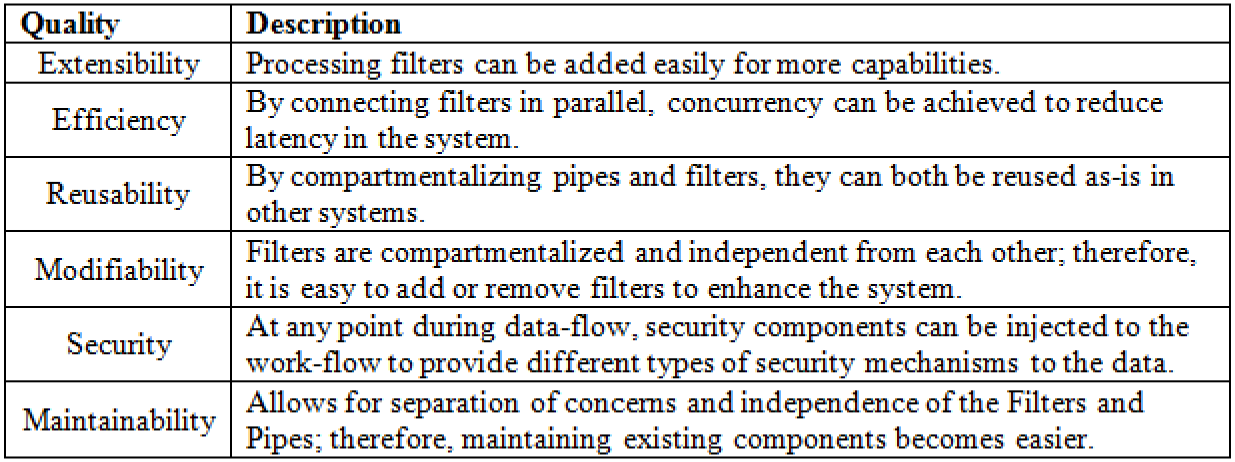


Pipe

Filter

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.

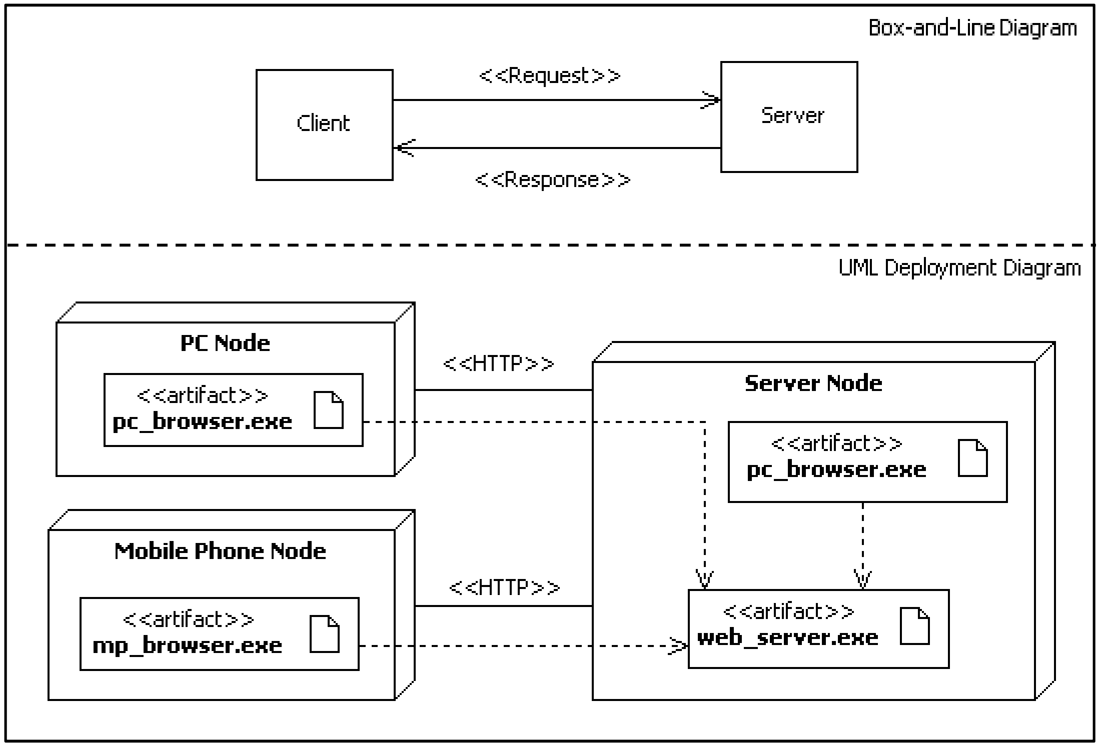
DISTRIBUTED SYSTEMS

- 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.

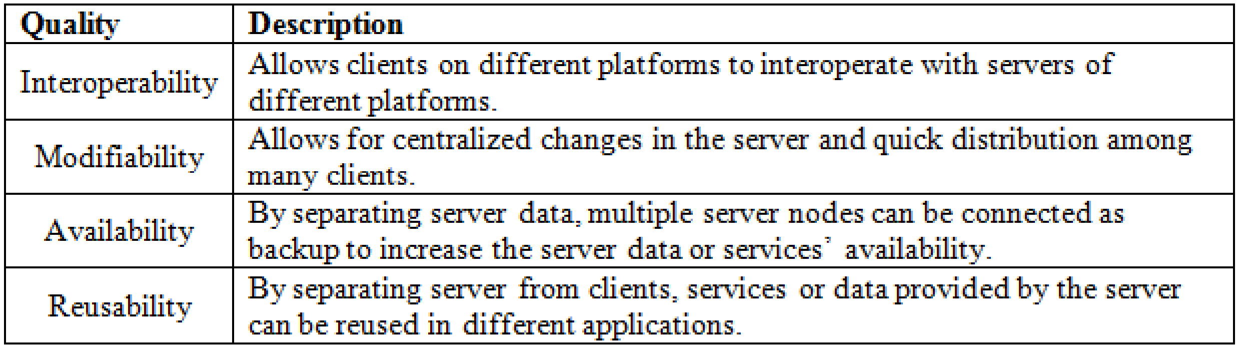
- Common architectural patterns for distributed systems include:
  - ✓ Client-Server Pattern
  - ✓ Broker Pattern

CLIENT-SERVER PATTERN



Client-Server Pattern

- Quality properties of the Blackboard architectural 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.

WHAT'S NEXT…

- 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

- In the next session, we will continue the discussion of distributed systems and present in depth two other types of systems (i.e., Interactive and Hierarchical) together with architectural patterns, including:
  - ✓ Model-View-Controller
  - ✓ Layered
  - ✓ Main program and subroutine