Unit 2:
Architecture (4 Hrs.)

Unit 2: Architecture (4 Hrs.)

Background

1. Architectural Structure
2. Middleware organization
3. System Architecture
4. Example Architecture

Background

- Distributed systems are often complex pieces of software of which the components are dispersed across multiple machines.
- To master their complexity, it is crucial that these systems are properly organized.
- The software architectures tell us how the various software components are to be organized and how they should interact.

2.1 ARCHITECTURAL STYLE

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

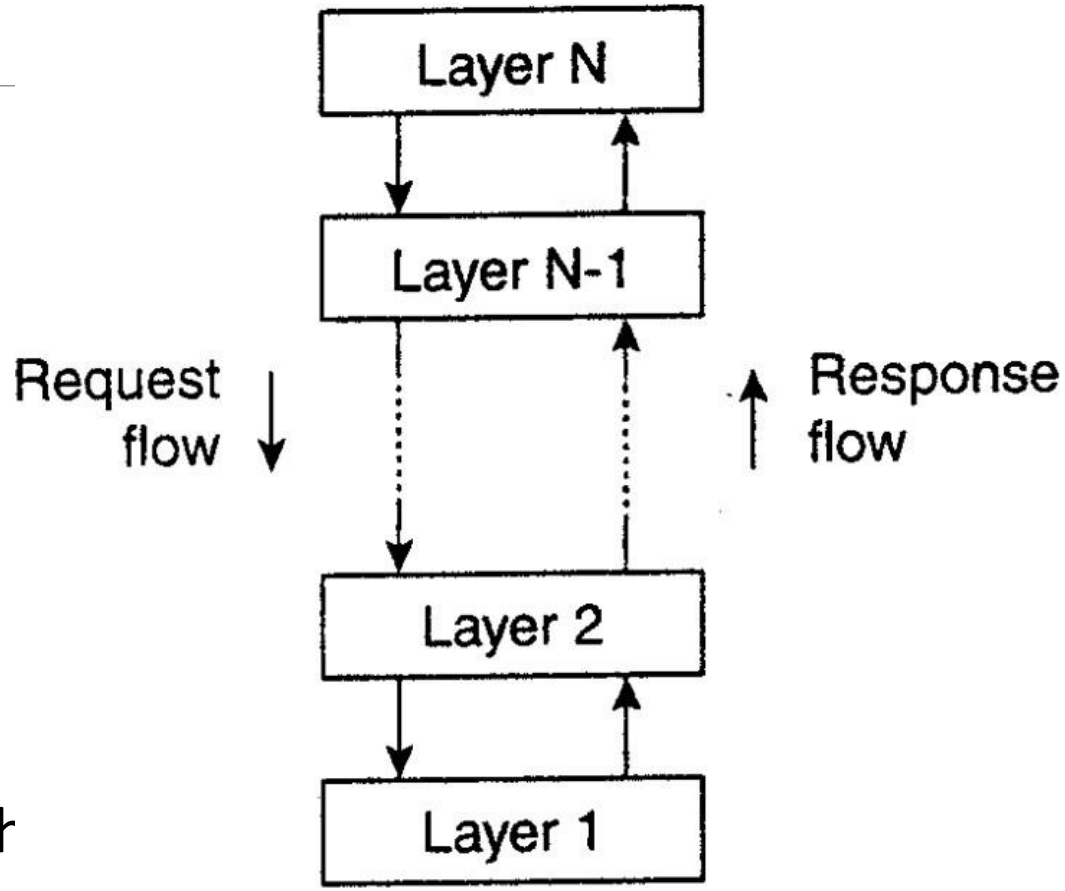
Layered architectures

- **Basic Idea:**

  --components are organized in a layered fashion where a component at layer L; is allowed to call components at Li, but not the other way around, as shown in Fig.

- Widely adopted by the networking community.

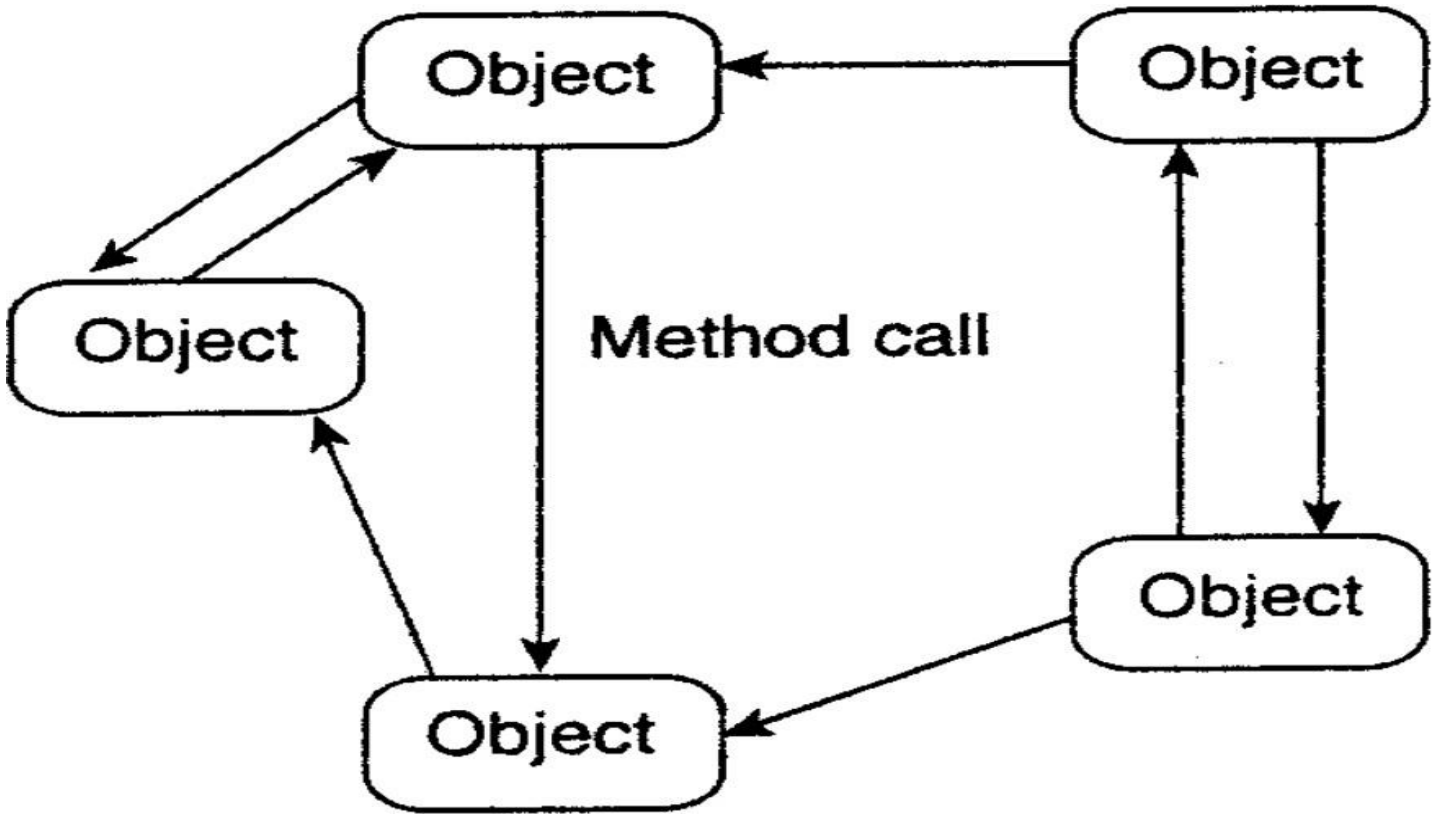
- Key observation is that control generally flows from layer to layer: requests go down the hierarch whereas the results flow upward.



Object-based architectures

- In this type of architecture, components are treated as objects which convey information to each other. Object-Oriented Architecture contains an arrangement of loosely coupled objects. Objects can interact with each other through method calls. Objects are connected to each other through the Remote Procedure Call (RPC) mechanism or Remote Method Invocation (RMI) mechanism.
- Web Services and REST API are examples of object-oriented architecture. Invocations of methods are how objects communicate with one another. Typically, these are referred to as Remote Procedure Calls (RPC). REST API Calls, Web Services, and Java RMI are a few well-known examples. These characteristics apply to this.

Object based cont..

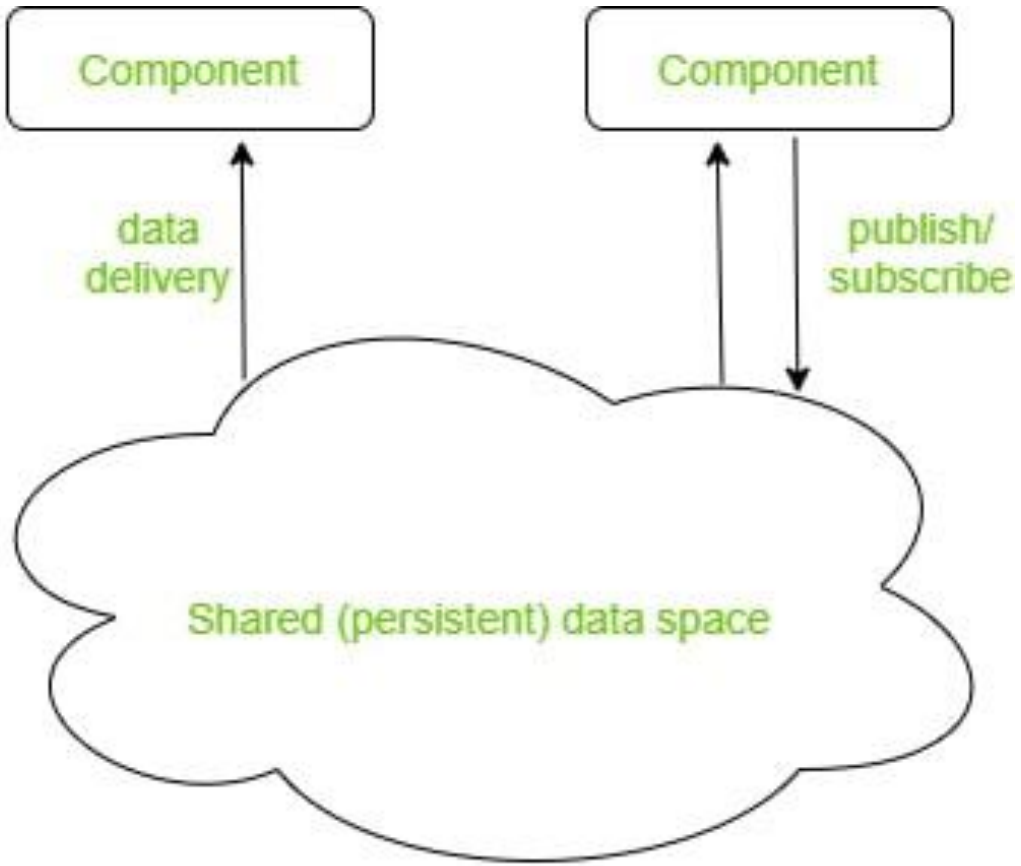


Data-centered architectures

- Data Centered Architecture is a type of architecture in which a common data space is present at the centre. It contains all the required data in one place a shared data space. All the components are connected to this data space and they follow publish/subscribe type of communication. It has a central data repository at the centre. Required data is then delivered to the components. Distributed file systems, producer-consumer systems, and web-based data services are a few well-known examples.

- For example Producer-Consumer system. The producer produces data in common data space and consumers request data.

Data centered architecture cont..

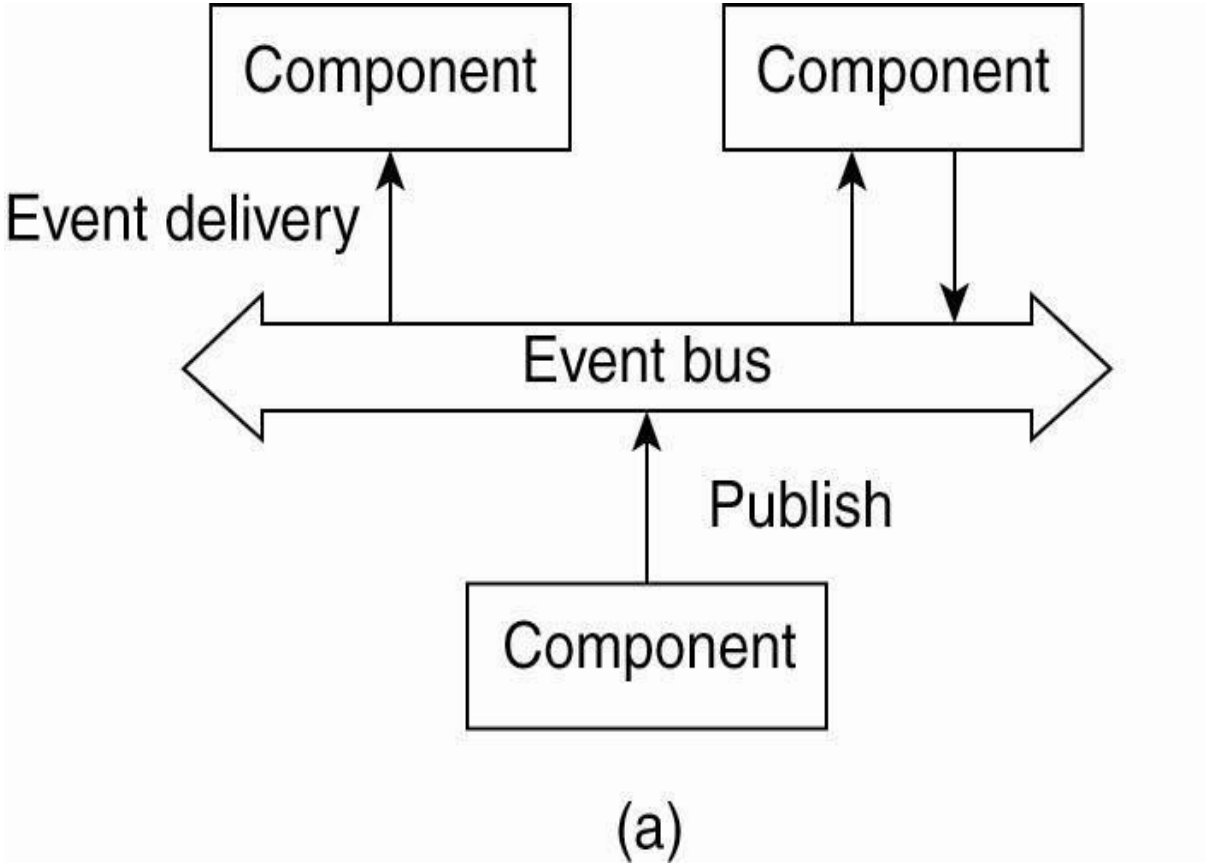


Event-based Architectural Style

- Event-Based Architecture is almost similar to Data centered architecture just the difference is that in this architecture events are present instead of data. Events are present at the centre in the Event bus and delivered to the required component whenever needed. In this architecture, the entire communication is done through events. When an event occurs, the system, as well as the receiver, get notified. Data, URLs etc are transmitted through events. The components of this system are loosely coupled that's why it is easy to add, remove and modify them. Heterogeneous components can communicate through the bus. One significant benefit is that these heterogeneous components can communicate with the bus using any protocol. However, a specific bus or an ESB has the ability to handle any kind of incoming request and respond appropriately.

Event-based architecture cont..

This architectural style is based in the publisher-subscriber architecture. Between each node there is no direct communication or coordination. Instead, objects which are subscribed to the service communicate through the event bus.



The Event based architecture supports, several communication styles:

- Publisher-subscriber
- Broadcast
- Point-to-point

The major advantage of this architecture is that the components are decoupled in space loosely coupled.

2.2 SYSTEM ARCHITECTURES

How many distributed systems are actually organized by considering where software components are placed.

- Centralized Architectures
- Decentralized Architectures
- Hybrid Architectures

Centralized Architectures

- The centralized architecture is defined as every node being connected to a central coordination system, and whatever information they desire to exchange will be shared by that system. A centralized architecture does not automatically require that all functions must be in a single place or circuit, but rather that most parts are grouped and none are repeated elsewhere as would be the case in a distributed architecture.

Characteristics

Single Point of Control: In a centralized system, there is a single point of control and authority. This central entity typically makes all decisions and manages all resources.

Centralized Data Management: All data and resources are stored and managed centrally. This means that all data processing, storage, and retrieval activities occur within the central system.

Centralized Architectures cont..

Characteristics

Dependent failure of components: Central node failure causes the entire system to fail. This make sense because when the server is down, no other entity is there to send/receive responses/requests.

Security:

Security measures can be more easily implemented and monitored at a single point.

However, if the central point is compromised, the entire system's security can be at risk.

Advantage

- Easy to Manage
- Consistent Data
- Better Security
- Simplified Backup and Recovery

Disadvantage

Highly dependent on the network connectivity – The system can fail if the nodes lose connectivity as there is only one central node.
Difficult server maintenance – There is only one server node and due to availability reasons, it is inefficient and unprofessional to take the server down for maintenance. So, updates have to be done on-the-fly(hot updates) which is difficult and the system could break.
Single point of failure: Centralized systems have a single point of failure, which can cause the entire system to fail if the central node goes down.
Limited scalability: Centralized systems have limited scalability as the central node can only h at a time.
Limited scalability: Centralized systems have limited scalability as the central node can only handle a limited number of clients at a time.

Note:Most common architecture of centralized system is Client server Architecture.

Decentralized Organization

A decentralized distributed system is a type of computing system where multiple independent entities (nodes or computers) work together to achieve a common goal, but there is no single central authority or control point. Each node in the system operates autonomously and makes its own decisions, communicating and coordinating with other nodes to perform tasks.

Characteristics:

Lack of a global clock:

Every node is independent of each other and hence, has different clocks that they run and follow.

Autonomy of Nodes:

Each node operates independently and has the ability to make its own decisions without relying on a central authority.

Redundancy and Fault Tolerance:

Because there is no single point of failure, the system can continue to function even if some nodes fail. This enhances reliability and fault tolerance.

Decentralized system cont..

Scalability:
These systems can easily scale by adding more nodes, as there is no central bottleneck to limit growth.
Peer-to-Peer Communication:
Nodes communicate directly with each other, often in a peer-to-peer manner, which helps distribute the load and reduce dependencies on any single node.
Resource Sharing:
Resources (such as data, computing power, or storage) are shared among nodes, improving overall system efficiency and utilization.
Decentralized Control:
Control and decision-making are distributed across all nodes, reducing the risk of central authority misuse or failure.

Advantage of Decentralized System

Minimal problem of performance bottlenecks occurring – The entire load gets balanced on all the nodes; leading to minimal to no bottleneck situations.
High availability – Some nodes(computers, mobiles, servers) are always available/online for work, leading to high availability.
More autonomy and control over resources – As each node controls its own behavior, it has better autonomy leading to more control over resources.
Improved fault tolerance: Decentralized systems are designed to be fault tolerant, meaning that if one or more nodes fail, the system can still continue to function. This is because the workload is distributed across multiple nodes, rather than relying on a single point of failure.
Greater security: Decentralized systems can be more secure than centralized systems because there is no single point of failure or vulnerability that can be exploited by attackers. Data is distributed across multiple nodes, making it more difficult to hack or compromise.

Disadvantage:

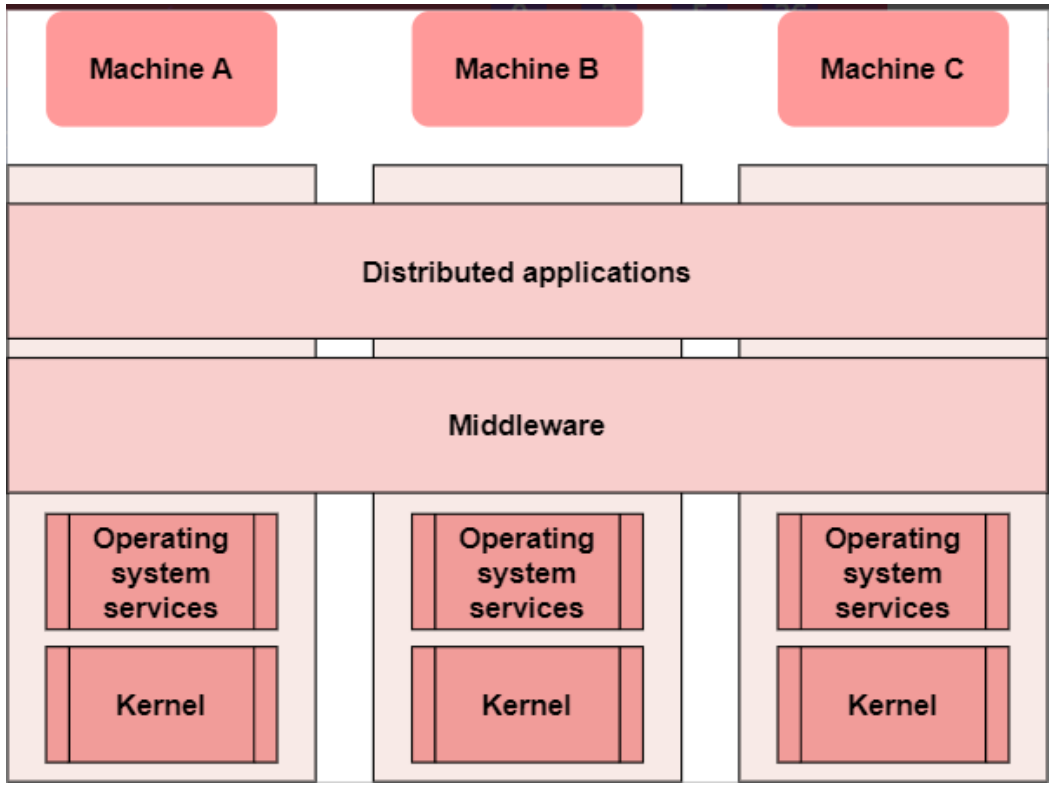
Difficult to achieve global big tasks – achieving large-scale tasks can be difficult because there's no central authority to give orders and coordinate everyone's actions.
No regulatory oversight- means there's no central authority or organization that monitors and enforces rules and standards.
Difficult to know which node failed – Each node must be pinged for availability checking and partitioning of work has to be done to actually find out which node failed by checking the expected output with what the node generated
Difficult to know which node responded – When a request is served by a decentralized system, the request is actually served by one of the nodes in the system but it is actually difficult to find out which node indeed served the request.
Slow transaction processing: Decentralized systems can be slower in processing transactions compared to centralized systems. This is because each transaction needs to be validated by multiple nodes, which can take time.

Note:Most common architecture of Decentralized system is peer-to-peer.

Middleware Organization

In a distributed system, middleware is a software component that serves between two or more applications.

Middleware usually resides between the operating system and the end user or end-user application. It provides essential features that the operating system doesn't offer. The term usually refers to large software products, such as database managers, transaction monitors, and web servers.



Middleware Organization cont..

The design and structure of middleware can be independent of the design and structure of the overall distributed system or application it supports. This means that middleware can be designed in a way that it works efficiently regardless of how the rest of the system is organized. To achieve this, two common design patterns are often used in organizing middleware: **Wrappers** and **Interceptors**. Here's what these terms mean in this context:

Wrappers
Definition: Wrappers, also known as adapters, are components that provide a compatible interface between a client application and a component that otherwise has an incompatible interface.
Purpose: They enable different parts of a system to communicate even if their native interfaces don't match.
Functionality: A wrapper can transform the requests from the client into a form that the component understands and vice versa.
Example: Suppose you have an old database that doesn't support modern web-based interfaces. A wrapper can be created to allow new web applications to interact with this old database by translating web requests into the database's native query language and then translating the responses back to a web-friendly format.

Middleware Organization cont..

Specific Wrapper for Each Pair:
A wrapper is created to make one application (A) communicate with another application (B) by converting the interface of A to be compatible with B.
If you have to do this for every pair of applications, each pair needs its own unique wrapper.
Number of Wrappers:
If you have N applications, each application needs a wrapper to communicate with every other application.
The formula N × (N − 1) represents the total number of wrappers needed because:

For each of the N applications, you need a wrapper to communicate with the remaining (N − 1) applications.

Example with 3 Applications:
If you have 3 applications (A, B, C):

A needs wrappers for B and C (2 wrappers).
B needs wrappers for A and C (2 wrappers).
C needs wrappers for A and B (2 wrappers).

Total wrappers = 2 (for A) + 2 (for B) + 2 (for C) = 6 wrappers.

Middleware Organization cont..

Problem with Many Wrappers:
Creating a specific wrapper for each pair of applications that need to communicate is impractical because it results in a large number of wrappers, which becomes hard to manage.
Solution with Middleware:
Middleware can simplify this by reducing the need for many wrappers. One effective way to do this is by using a "broker."
What is a Broker?:
A broker is a centralized component in the system that manages all communication between different applications. Instead of each application having its own wrapper for every other application, all applications communicate through the broker.
Example - Message Broker:
A message broker is a common type of broker.
Applications send their requests to the message broker, including what they need or what action they want to perform.
The message broker then handles the communication, ensuring the request reaches the right application or component.

Middleware Organization cont..

Simplified Example:
Imagine you have 5 different people who need to send letters to each other:
Without a broker: Each person writes a letter directly to every other person, resulting in many letters.
With a broker: Each person sends their letter to a central post office (the broker), which then forwards the letter to the right recipient.
In this way, the broker (like the post office) simplifies and centralizes communication, reducing the need for each person to handle multiple direct interactions.

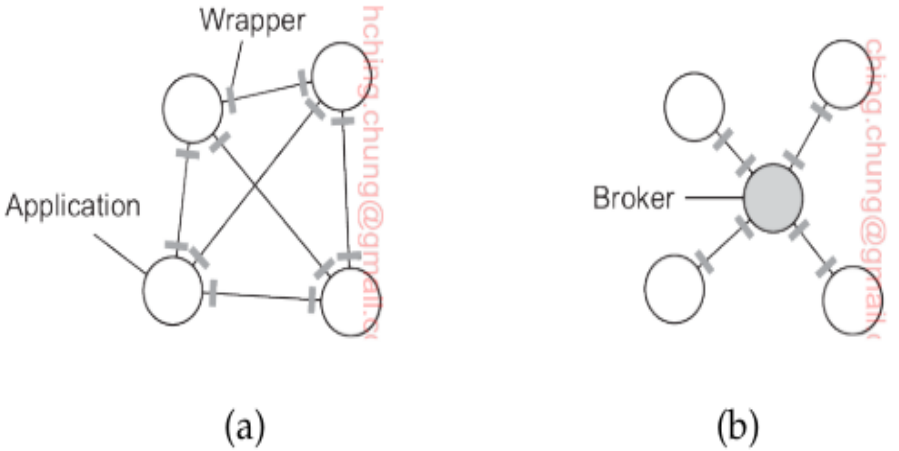


Figure 2.13: (a) Requiring each application to have a wrapper for each other application. (b) Reducing the number of wrappers by making use of a broker.

Middleware Organization cont..

Interceptors

Interceptors in middleware are components that intercept and process requests or responses as they travel between clients and services. They are used to implement cross-cutting concerns such as logging, authentication, monitoring, and more without modifying the core business logic of the application.

What Interceptors Do:

Intercept Requests:

When a request is made from a client to a service, the interceptor can catch this request before it reaches the service.

The interceptor can then perform various actions, such as logging the request details, checking if the user is authenticated, or modifying the request.

Intercept Responses:

After the service processes the request and generates a response, the interceptor can catch this response before it goes back to the client.

The interceptor can then perform actions like logging the response, adding additional headers, or modifying the response content.

Middleware Organization cont..

Simplified Example:
Imagine a security guard (interceptor) at the entrance of a building (service):
Intercepting Requests: The guard checks each person (request) entering the building to ensure they have an ID badge (authentication). If they don't, they are not allowed to enter.
Intercepting Responses: Before someone leaves the building (response), the guard may check if they are carrying anything suspicious (logging or modification).