



CHAPTER 2

ARCHITECTURES

BY ANKU JAISWAL




Contents

- 2.1 ARCHITECTURAL STYLES
 - 2.2. MIDDLEWARE ORGANIZATION
 - 2.3. SYSTEM ARCHITECTURES
 - 2.4. EXAMPLE ARCHITECTURE
- 

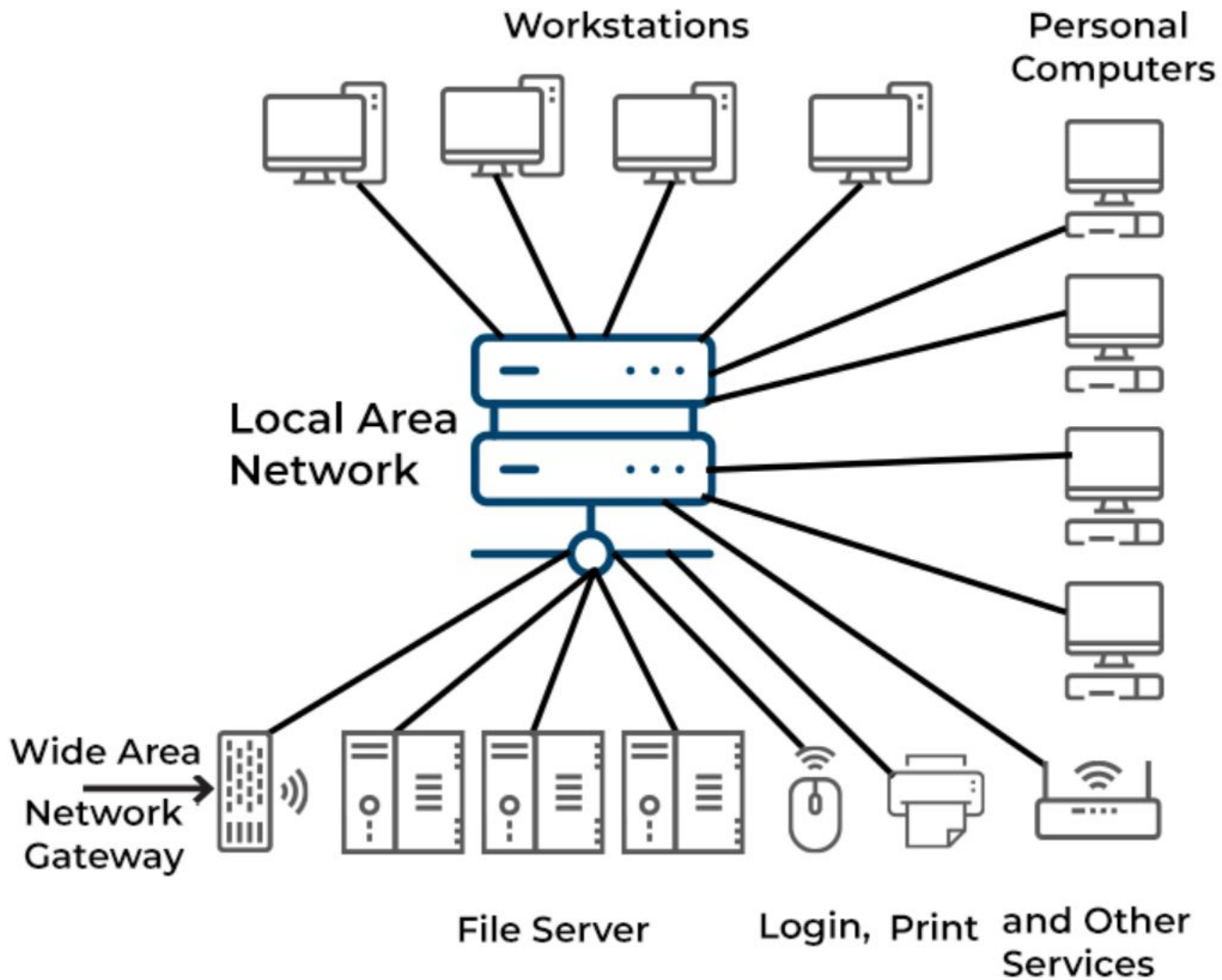




OVERVIEW

- Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines.
 - To master their complexity, it is crucial that these systems are properly organized.
- 

Distributed System Architecture


- Distributed system architectures are bundled up with components and connectors.
- Components can be individual nodes or important components in the architecture
- whereas connectors are the ones that connect each of these components.



- 
- 
- Component: A modular unit with well-defined interfaces; replaceable; reusable
 - Connector: A communication link between modules which mediates coordination or cooperation among components






So the idea behind distributed architectures is to:

- have these components presented on different platforms,
 - where components can communicate with each other over a communication network in order to achieve specific objectives.
- 



ARCHITECTURES

- There are different ways on how to view the organization of a distributed system:
 - Software architectures
 - System architecture
- 


- 
- 
- The organization of distributed systems is mostly about the software components that constitute the system
 - These software architectures tell us how the various software components are to be organized and how they should interact




2.1 ARCHITECTURAL STYLES


- Designing or adopting an architecture is crucial for the successful development of large systems

Architectural style

- It is formulated in terms of components
 - the way that components are connected to each other,
 - the data exchanged between components and finally how these elements are jointly configured into a system
- 






Using components and connectors, we can come to various configurations, which, in turn have been classified into architectural styles

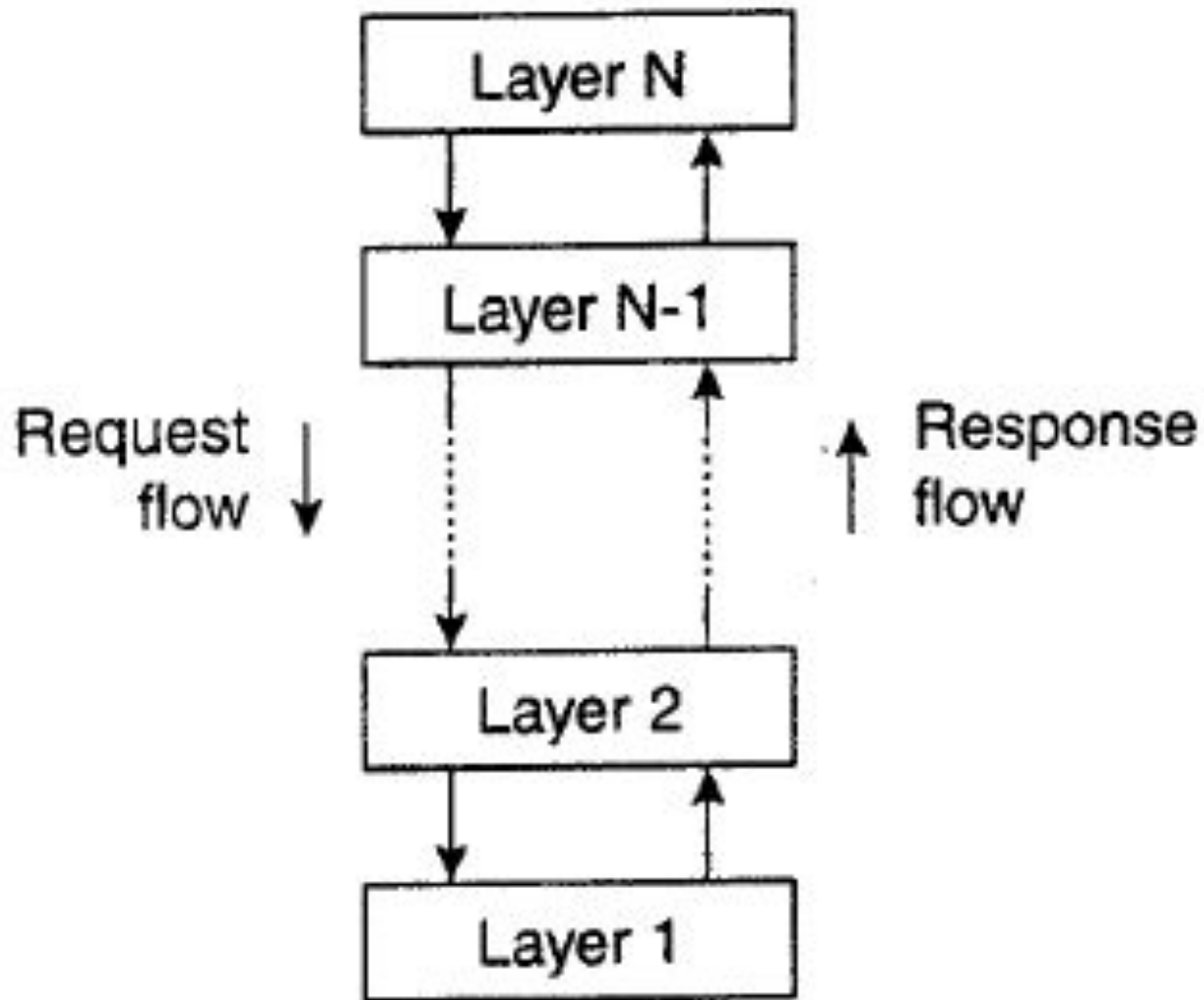
- Layered architectures
 - Object-based architectures
 - Data-centered architectures
 - Event-based architectures
- 





1. Layered Architecture


- The layered architecture separates layers of components from each other, giving it a much more modular approach.
 - A well known example for this is the OSI model that incorporates a layered architecture when interacting with each of the components.
 - Each interaction is sequential where a layer will contact the adjacent layer and this process continues.
- 

- 
- 
- The layers on the bottom provide a service to the layers on the top.
 - The request flows from top to bottom, whereas the response is sent from bottom to top.
 - The advantage of using this approach is that, each layer can be easily replaced or modified without affecting the entire architecture.



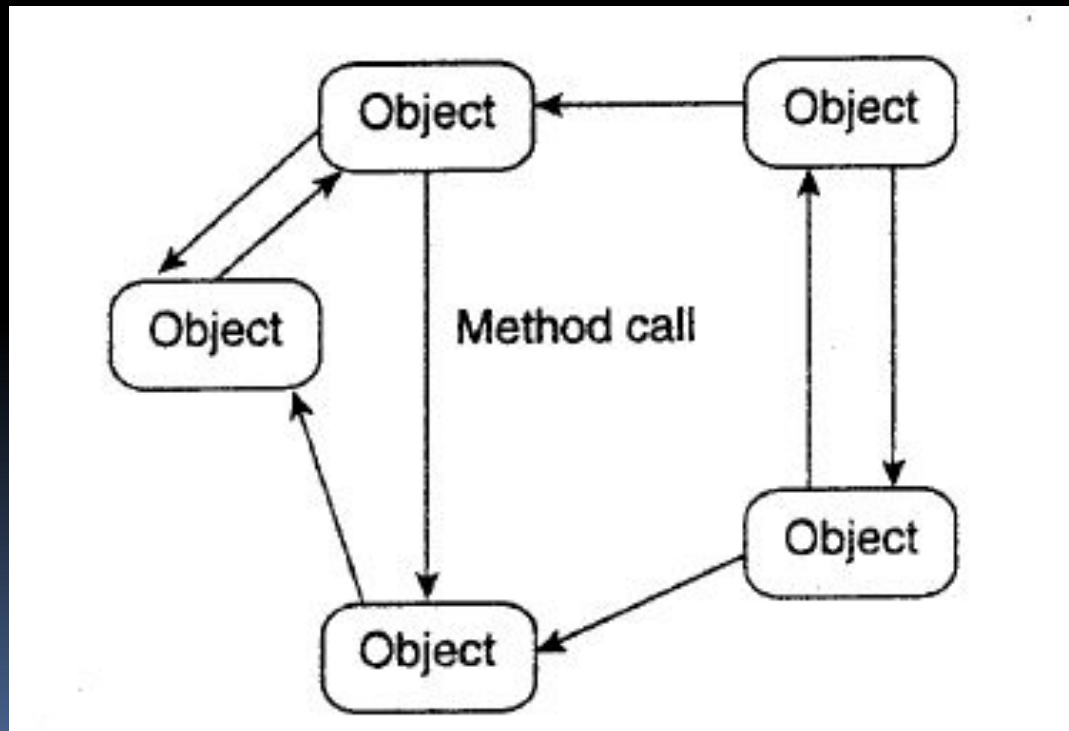
(a)



- 
- 
- Components are organized in a layered fashion where a component at layer L is allowed to call components at the underlying layer $L-1$
 - Control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward



- 
- This model has been widely adopted by the networking community;
 - An key observation is that control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward.

2. Object-based Architectures

- Each object corresponds a component, which are connected through a (remote) procedure call mechanism

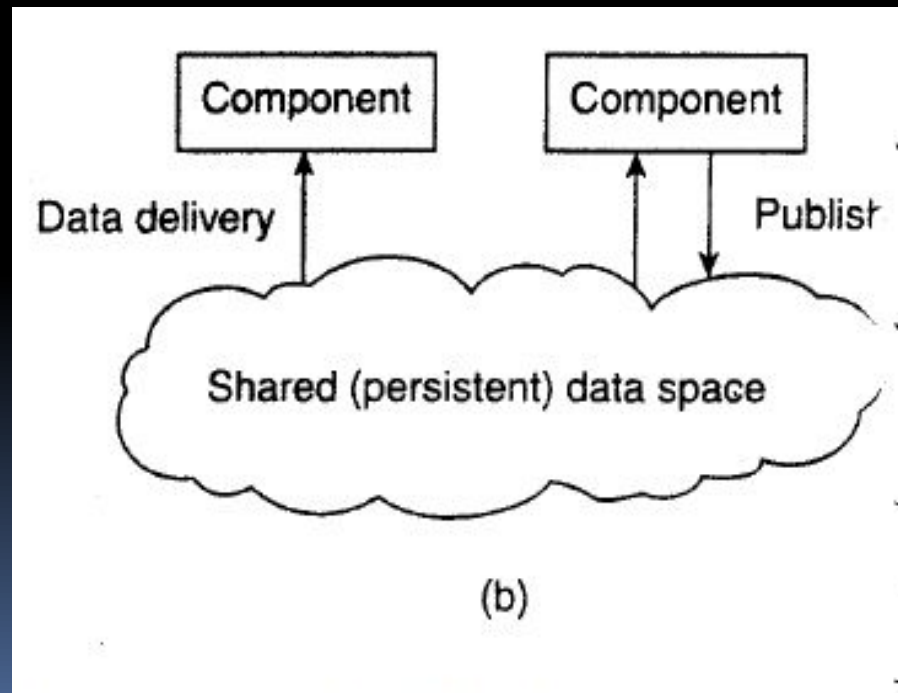




- 
- 
- This architecture style is based on loosely coupled arrangement of objects.
 - This has no specific architecture like layers. Like in layers, this does not have a sequential set of steps that needs to be carried out for a given call.
 - Each of the components are referred to as objects, where each object can interact with other objects through a given connector or interface.
 - These are much more direct where all the different components can interact directly with other components through a direct method call.



- 
- 
- These are generally called Remote Procedure Calls (RPC). Some popular examples are Java RMI, Web Services and REST API Calls.
 - This has the following properties.
 - This architecture style is less structured.
 - component = object
 - connector = RPC

3. Data-centered architectures

- Evolve around the idea that processes communicate through a common (passive or active) repository




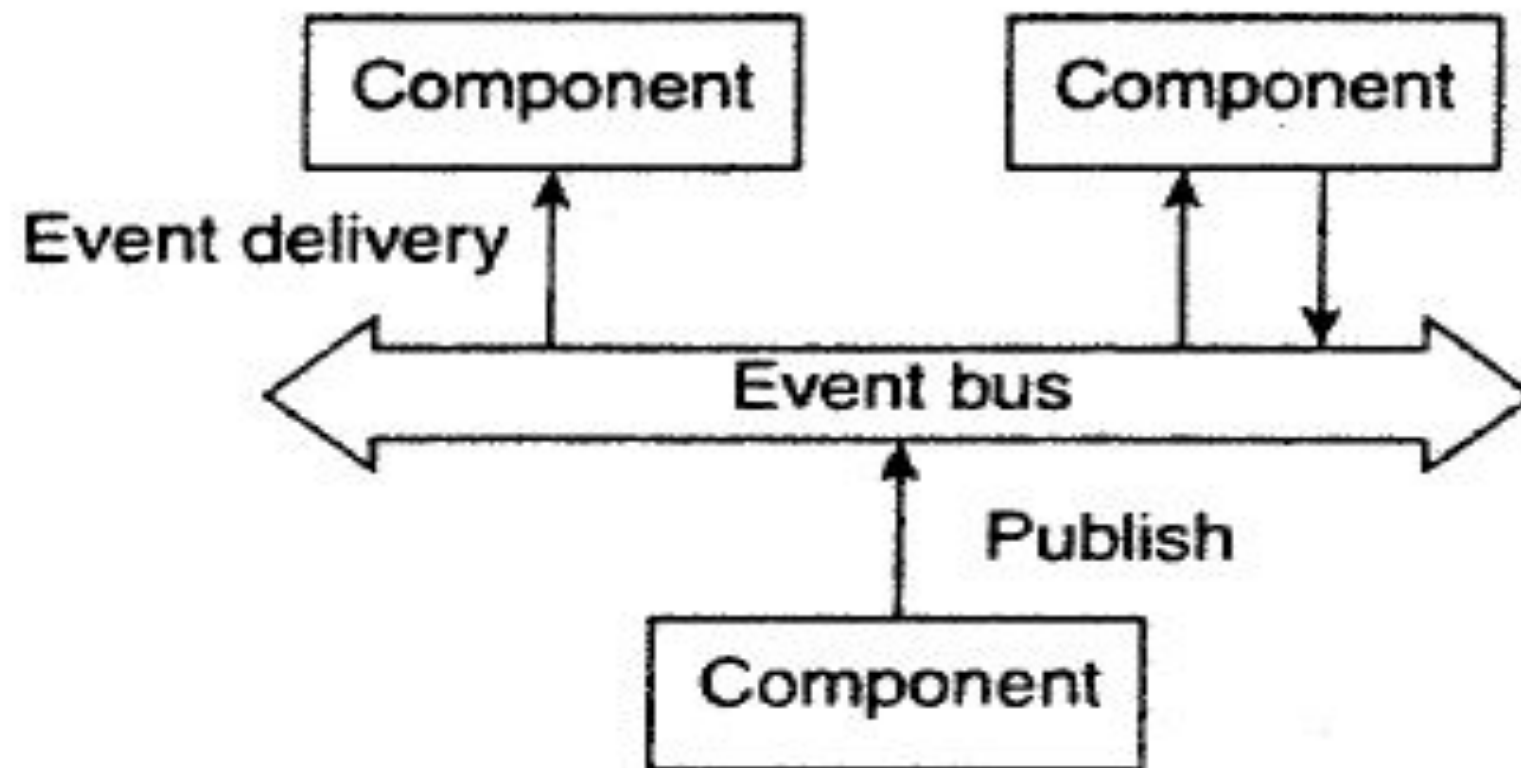
- 
- 
- As the title suggests, this architecture is based on a data center, where the primary communication happens via a central data repository.
 - This common repository can be either active (actual data store) or passive(mirror).
 - This is more like a producer consumer problem.



- 
- 
- The producers produce items to a common data store, and the consumers can request data from it.
 - This common repository, could even be a simple database.
 - But the idea is that, the communication between objects happening through this shared common storage.






Event-based Architectures

- In event-based architectures, processes essentially communicate through the propagation of events, which optionally also carry data, as shown in Fig.
 - For distributed systems, event propagation has generally been associated with what are known as publish/subscribe systems.
 - The basic idea is that processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.
- 




- 
- 
- The entire communication in this kind of a system happens through events.
 - When an event is generated, it will be sent to the bus system.
 - With this, everyone else will be notified telling that such an event has occurred.

- 
- 
- So, if anyone is interested, that node can pull the event from the bus and use it.
 - Sometimes these events could be data, or even URLs to resources.
 - So the receiver can access whatever the information is given in the event and process accordingly.

- 
- These events occasionally carry data. An advantage in this architectural style is that, components are loosely coupled.
 - So it is easy to add, remove and modify components in the system.
 - Some examples are, publisher - subscriber system, Enterprise Services Bus (ESB) and akka.io.




Data space

- Event-based architectures can be combined with data-centered architectures, yielding what is also known as shared data spaces
- 




2.2 SYSTEM ARCHITECTURES

Deciding on software components, their interaction, and their placement leads to an instance of a software architecture, also called a system architecture





2.2 SYSTEM ARCHITECTURES

- 2.2.1 Centralized Architectures
 - 2.2.2 Decentralized Architectures
 - 2.2.3 Hybrid Architectures
- 



Centralized architectures

- one server implements the software components
- remote clients can access that server

Decentralized architectures

- machines more or less play equal roles, as well as hybrid organizations
- 



CENTRALIZED NETWORK

WHAT IS CENTRALIZATION?

In a centralized network, there is a central authority that governs and handles the network.

ADVANTAGES

- Command chain
- Reduced costs
- Consistent output



DISADVANTAGES

- Not 100% Trustable
- Single point of failure
- Scalability limitation



DECENTRALIZED NETWORK

WHAT IS DECENTRALIZATION?

In a decentralized network, there is no central authority that governs and handles the network.

ADVANTAGES

- Full control
- Immutable data
- High security




DISADVANTAGES

- Costly
- Misuse of authority
- Volatility






CENTRALIZED VS. DECENTRALIZED

	CENTRALIZED	DECENTRALIZED	
Third-Party Involvement	Yes	No	
Control	Full control stays with the central authority	Control stays with the user itself	
Hackable	More prone to hacks and data leaks	Less prone to hacks and data leaks	
Single Point of Failure	Yes	No	
Ease of Use	Intuitive and easy to use	Not easy to use	
Exchange Fees	Higher fees	Less fees	



2.2.1 Centralized architecture

- 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 together and none are repeated elsewhere as would be the case in a distributed architecture.
- 

- 
- It consists following types of architecture:
 - Client-server
 - Application Layering
- 



■ **Client Server**

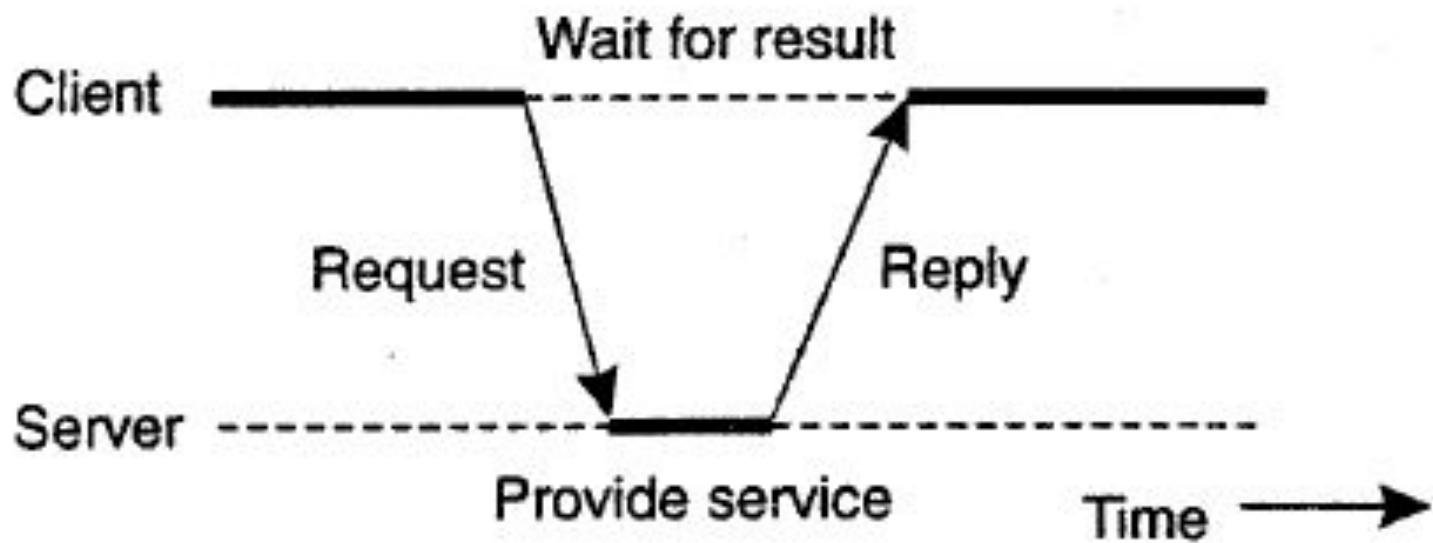
- Processes in a distributed system are split into two (potentially overlapping) groups in the fundamental client-server architecture.
- A server is a program that provides a particular service, such as a database service or a file system service.
- A client is a process that sends a request to a server and then waits for the server to respond before requesting a service from it.
- This client-server interaction, also known as request-reply behavior is shown in the figure below:

General interaction between a client and a server

In the basic client-server model, processes in a distributed system are divided into two (possibly overlapping) groups.

A server is a process implementing a specific service.

A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply






Reliable connection-oriented protocol


- reliable connection-oriented protocol
- sets up a connection to the server
- sending the request
- Uses that same connection, to send the reply message
- The connection is torn down
- setting up and tearing down a connection is relatively costly,





Application Layering

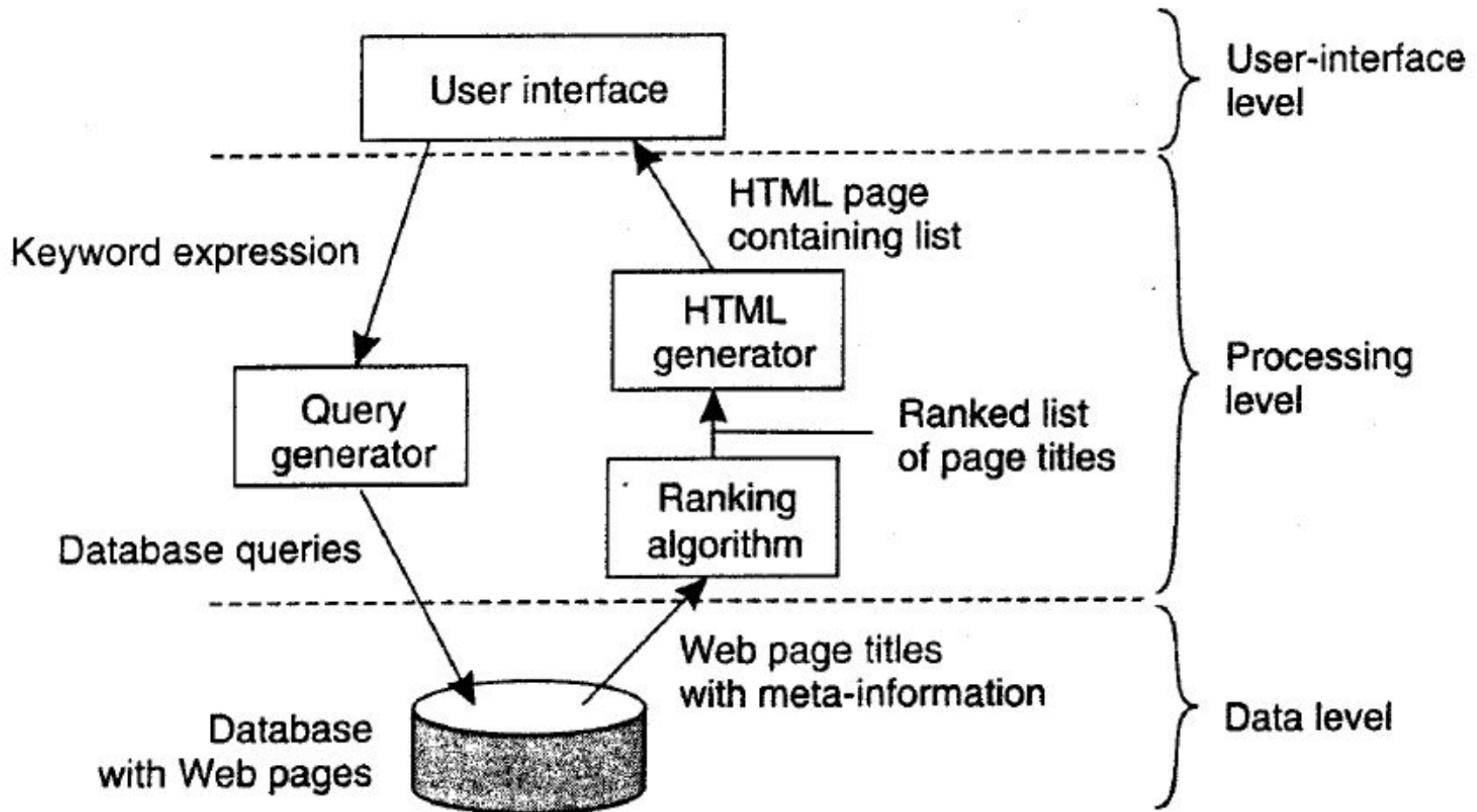
- The user-interface level
 - The processing level
 - The data level
- 


- 
- **The user interface level:** The user-interface level is often implemented by clients.
 - Programs that let users interact with applications make up this level.
- 

- 
- **The Processing level:** This is the middle part of the architecture.
 - This is a logical part of the system where all the processing actions are performed on the user interaction and the data level.
 - It processes the requests from the user interface and performs certain operations.

- 
- **The Data level:** The data level in the client-server model contains the programs that maintain the actual data on which the applications operate.
- 

Internet search engine






2.2.2 Decentralized Architectures

Decentralized Architecture refers to a system design in which control, decision-making, and management are distributed across multiple nodes, rather than being concentrated in a central authority or server.

This structure contrasts with centralized systems, where a single point of control governs the entire system.



In a decentralized architecture, there is no single failure point, which provides higher resilience, scalability, and fault tolerance.



Features of Decentralized Architecture

Distributed Control

Scalability

Redundancy and Fault Tolerance

Security



Transparency

Autonomy of Nodes:

Types of Decentralized Architecture

Peer-to-Peer (P2P)

Networks:

In a P2P network, all participating nodes are equal, and they share resources and services directly with each other. Examples of this type of architecture include file-sharing systems (e.g., BitTorrent) and cryptocurrency networks (e.g., Bitcoin).



Blockchain:

Blockchain is a decentralized, distributed ledger technology where data is stored across multiple nodes, each of which has a copy of the entire ledger. It provides an immutable record of transactions and enables decentralized applications (dApps) to run without needing a central server. Popular examples include Ethereum and Bitcoin.

Overlay Network


- An overlay network is a computer network which is built on the top of another network
- Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network

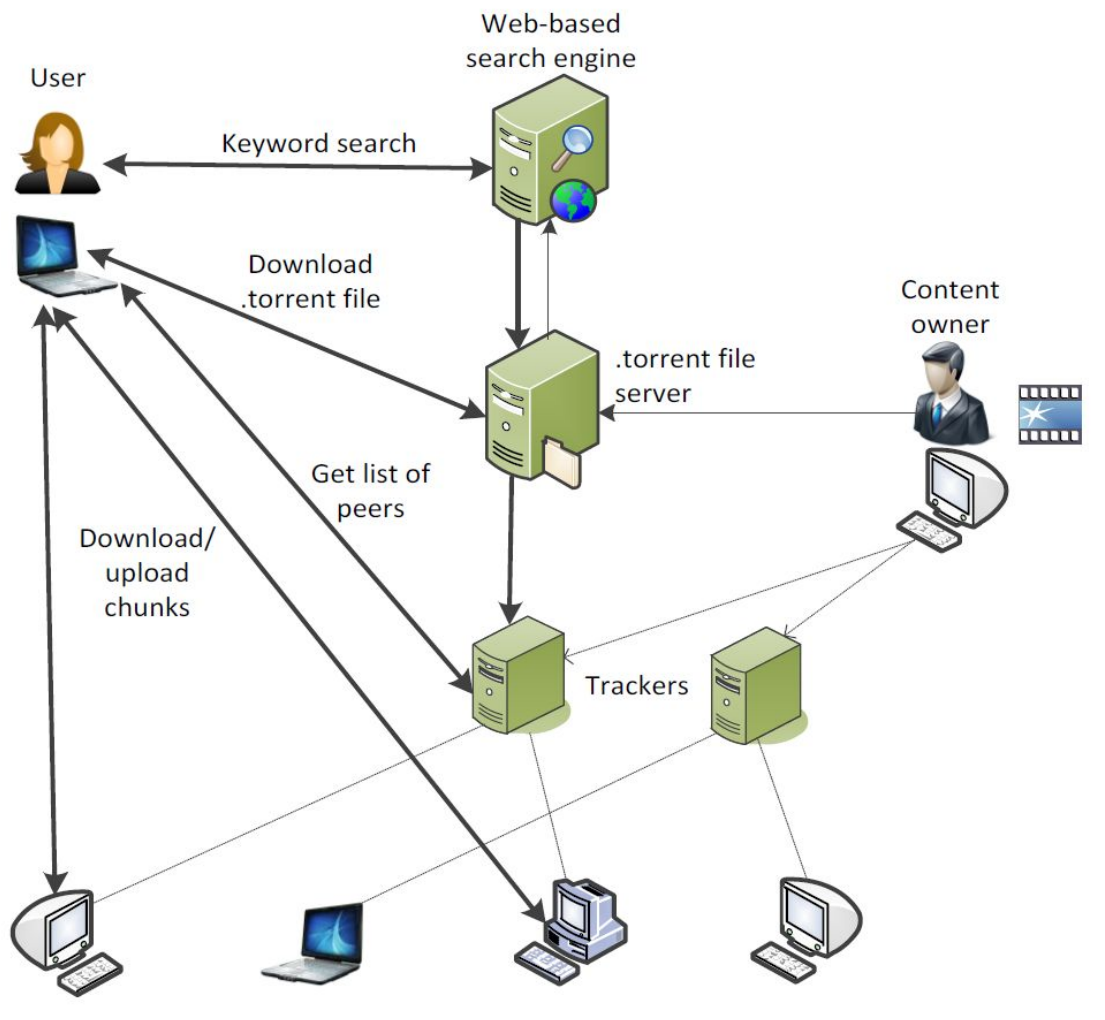
-

- 
- 
- For example, distributed systems such as cloud computing, peer-to-peer networks, and client-server applications are overlay networks because their nodes run on top of the Internet
 - The Internet was originally built as an overlay upon the telephone network while today the telephone network is increasingly turning into an overlay network built on top of the Internet



HYBRID ARCHITECTURE

- Hybrid systems are often based on both client server architectures and p2p networks.
 - A famous example is Bittorrent, which we use everyday.
- 





2.2.3 Hybrid Architectures

- Edge-Server
 - Collaborative Distributed Systems
- 

Edge-Server Systems

- servers are placed "at the edge" of the network
- This edge is formed by the boundary between enterprise networks and the actual Internet
 - ? For example, Internet Service Provider (ISP)
- end users at home connect to the Internet through their ISP
- the ISP can be considered as residing at the edge of the Internet

Edge-Server Systems

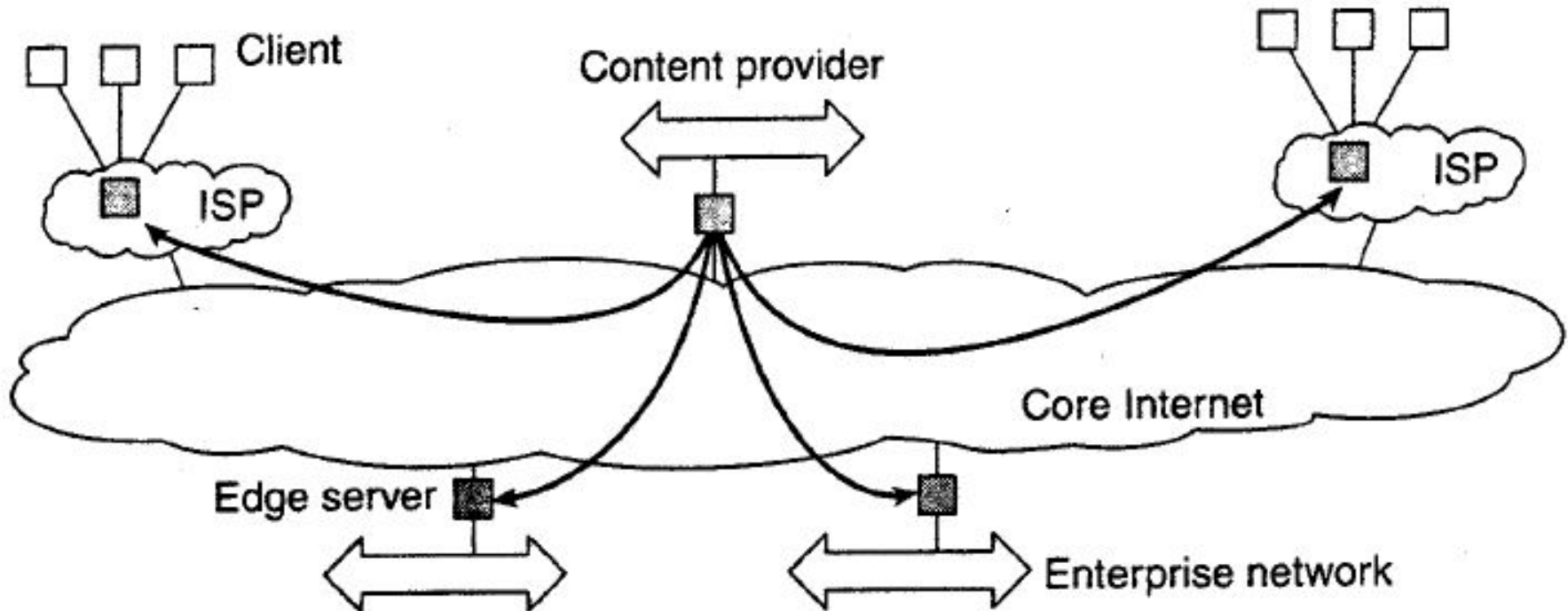



Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.




Edge-Server

- An edge server, in a system administration context, is any server that resides on the "edge" between two networks, typically a private network and the Internet.
- 



COLLABORATIVE DISTRIBUTED SYSTEM

A **collaborative distributed system** refers to a network of independent computing nodes (such as computers, servers, or devices) that work together in a coordinated manner to achieve a common goal, while being geographically spread out or distributed.



In such systems, collaboration refers to the way these nodes interact, share resources, and solve problems collectively, despite being physically separated.

These systems are typically designed to be fault-tolerant, scalable, and flexible.



Examples of CDS

Cloud computing platforms (like AWS, Google Cloud, and Microsoft Azure) provide collaborative distributed environments where multiple servers and services cooperate to handle large-scale applications.


Peer-to-peer networks (such as BitTorrent) where users share resources directly with each other.

Distributed databases that maintain copies of data on multiple nodes and ensure consistency and availability.

Blockchain systems like Bitcoin and Ethereum, where the system's nodes collaborate to validate and store transactions securely.





SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS

- reason, distributed systems should be adaptive
full distribution transparency is not what most applications actually want
 - autonomic computing
organizing distributed systems as high-level feedback-control systems allowing automatic adaptations to changes
 - self-managing
self-healing,
self-configuring
self-optimizing
- 




MIDDLEWARE ORGANIZATION

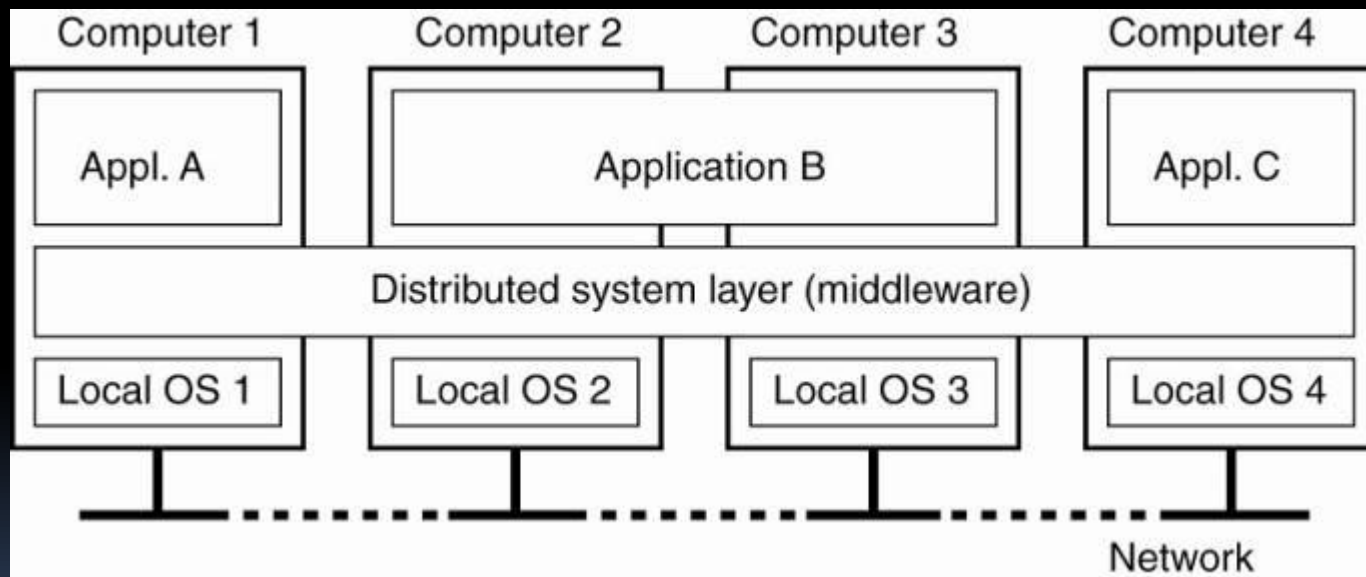
- In distributed system various heterogeneous devices are spread all over world.
 - Distributed system can achieve this consistency through a common layer to support the underlying hardware and OS.
 - This common layer is called Middleware.
- 



Middleware is software that acts as a bridge or intermediary between different software applications or components within a distributed computing system.






It enables communication and data management between disparate systems, helping them work together seamlessly. Middleware typically operates "in the middle" between the operating system and the applications, providing a common platform for communication, authentication, data processing, and other services.






Function of Middleware

- Hides the details of distributed applications
 - Hides the heterogeneity of hardware, operating systems and protocols
 - Provides uniform and high-level interfaces used to make interoperable, reusable and portable applications
 - Provides a set of common services that enhances collaboration between applications
- 

- 
- 
- Middleware is used to interconnect various kinds of nodes together.
 - Two important types of design pattern often applied to organization of middleware are:
 - wrapper
 - interceptors



Interceptors

- an **interceptor** is a software construct that will break the usual flow of control and allow other (application specific) code to be executed.
- 



Wrappers

a wrapper is a function that is intended to call one or more other functions, sometimes purely for convenience, and sometimes adapting them to do a slightly different task in the process. For example, SDK Libraries for AWS are examples of wrappers.

