Unit 2
Architecture

Contents

- Architectural Styles (Software Architecture)
- System Architecture
- Middleware Organization
- Example Architectures

Introduction

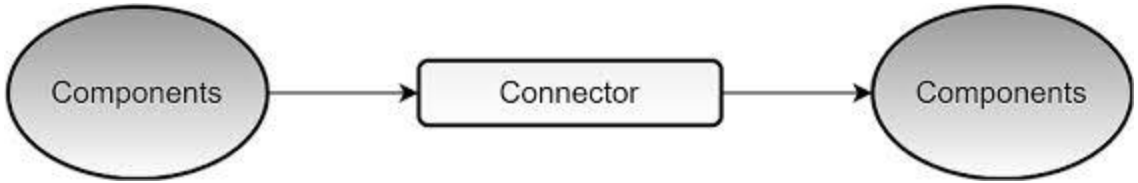
- Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines. To master their complexity, it is crucial that these systems are properly organized. There are different ways on how to view the organization of a distributed system, but an obvious one is to make a distinction between the logical organization of the collection of software components and on the other hand the actual physical realization.
- 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.
- The actual realization of a distributed system requires that we instantiate and place software components on real machines. There are many different choices that can be made in doing so. The final instantiation of a software architecture is also referred to as a system architecture
- An important goal of distributed systems is to separate applications from underlying platforms by providing a middleware layer. Adopting such a layer is an important architectural decision, and its main purpose is to provide distribution transparency.

Architectural Styles(Software Architecture)

- The logical organization of distributed systems into software components, also referred to as software architecture.
- The term software architecture referred originally to the structuring of software as layers or modules in a single computer and more recently in terms of services offered and requested between processes located in the same or different computers
- 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

Architectural Styles(Software Architecture)

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



Architectural Styles(Software Architecture)

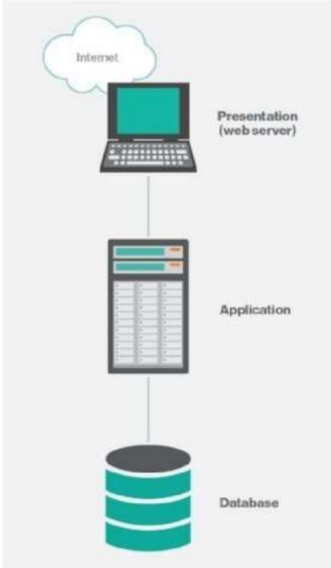
- There are four main architectural styles when it comes to distributed systems. The basic idea is to organize logically different components, and distribute those components over the various machines.
  1. Layered Architecture
  2. Object-based Architecture
  3. Data-Centered Architecture
  4. Event-based Architecture

Layered Architecture

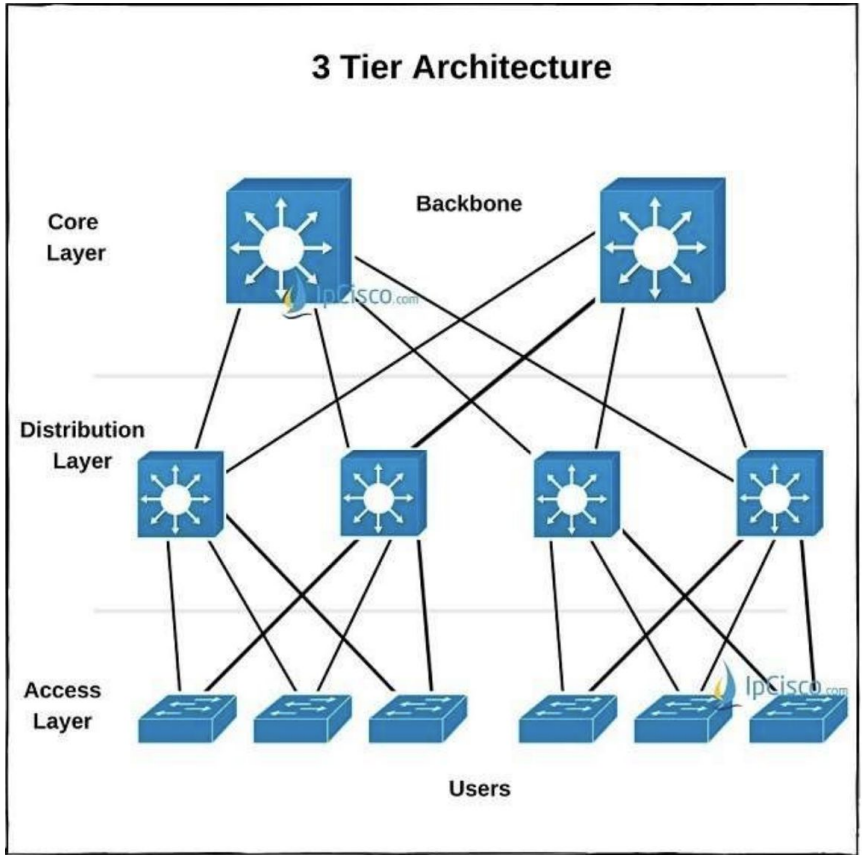
- The layered architecture separates layers of components from each other, giving it a much more modular approach. A well known example for this is the OSI model that incorporates a layered architecture when interacting with each of the components.
- Each interaction is sequential where a layer will contact the adjacent layer and this process continues, until the request is been catered to.
- 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, the calls always follow a predefined path, and that each layer can be easily replaced or modified without affecting the entire architecture.

Layered Architecture

- Layered Architecture Example: A web application with a user interface layer (presentation), an application server (business logic), and a database (data storage).



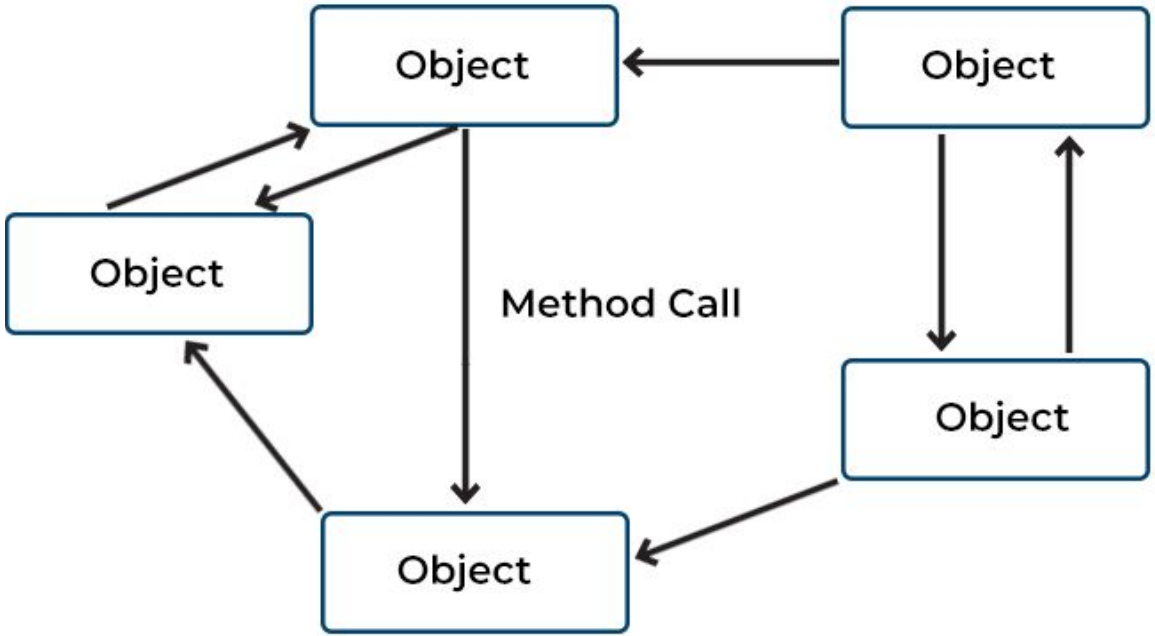
3-tier architecture



Object Based Architecture

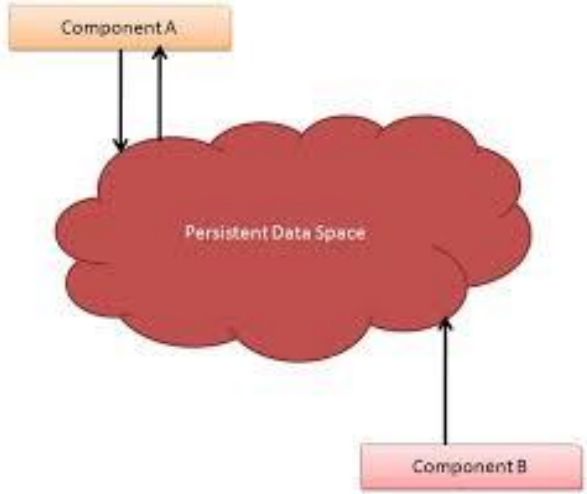
- This architecture style is based on loosely coupled arrangement of objects. This has no specific architecture like layers. Like in layers, this does not have a sequential set of steps that needs to be carried out for a given call.
- Each of the components are referred to as objects, where each object 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.
- Communication between object happen as method invocations. These are generally called Remote Procedure Calls (RPC).
- This architecture style is less structured.
- component = object
- connector = RPC or RMI

Object Based Architecture



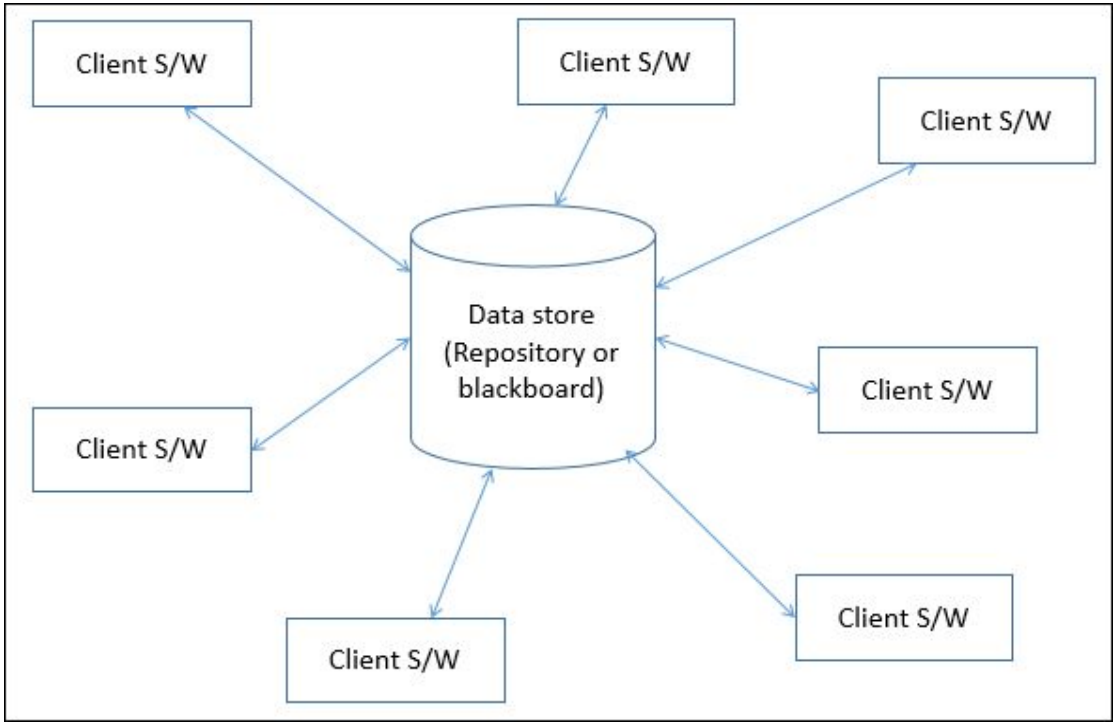
Data Centered Architecture

- It revolves around a central data repository that multiple components or services can access and manipulate.
- As the title suggests, this architecture is based on a data centre, where the primary communication happens via a central data repository.
- 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.



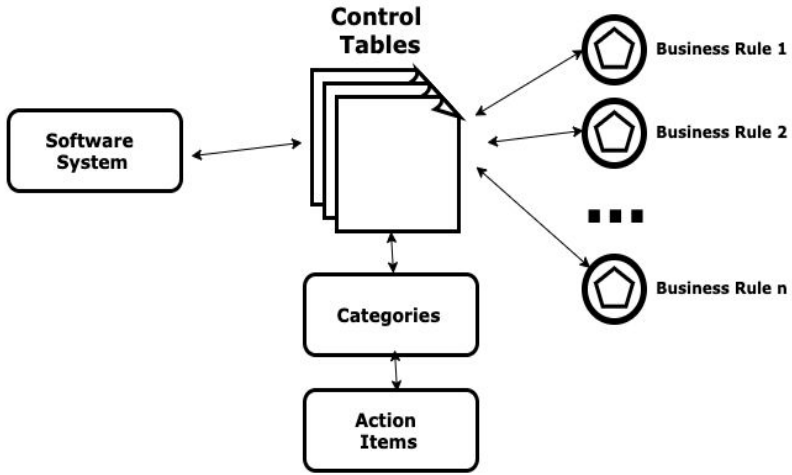
- This supports different components (or objects) by providing a persistent storage space for those components (such as a MySQL database). All the information related to the nodes in the system are stored in this persistent storage.

Data Centered Architecture



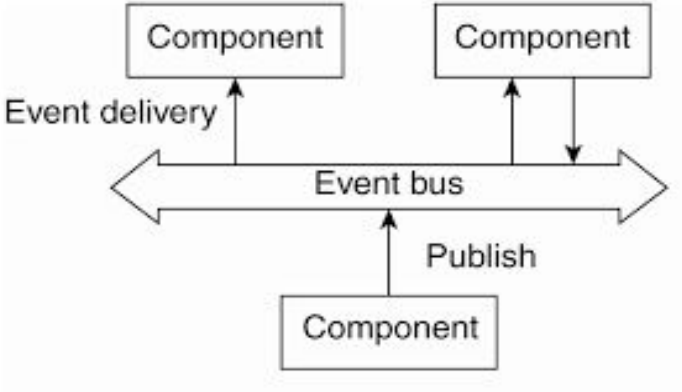
- Example1: A content management system (CMS) with a central database that stores all content, which is accessed and modified by various services like content delivery, editing, and indexing.
- Example2: CRM System

- It uses table-driven rules (as opposed to code-driven rules) to manage different categories and actions to come after a certain event. These tables are called control tables, and they allow programs to be simpler and more flexible.



Event-based Architecture

- 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.



- Processes communicate through the propagation of events. 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. This architectural style is based on 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.

Architecture

Architecture	Description
Layered	System organized into layers, each with specific responsibilities
Object-based	System as interacting objects with encapsulated data and behavior
Data-Centered	Central data repository accessed and manipulated by multiple components
Event-based	Components communicate by emitting and responding to events

System Architecture

- Software architecture refers to the logical organization of a distributed system into software components. Instead of one big monolithic application, distributed systems are broken down into multiple components. The way in which these components are broken down impacts everything from system performance to reliability to response latency.
- System architecture refers to the placement of these software components on physical machines. Two closely related components can be co-located or placed on different machines. The location of components will also impact performance and reliability.
    1. Centralized Architecture (Client Server Architecture)
    2. Decentralized Architecture (Peer to Peer)
    3. Hybrid Architecture

Centralized Architecture

- 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, for example, a file system service or a database 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.
- Communication between a client and a server can be implemented by means of a protocols.

Centralized Architecture

- This client-server interaction, also known as request-reply behavior is shown in figure below:

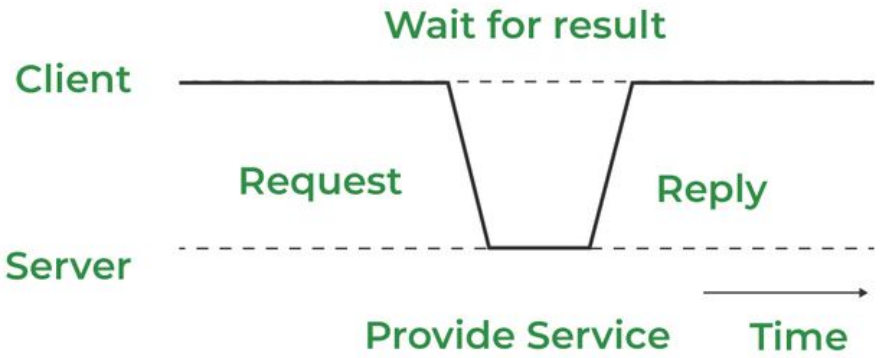


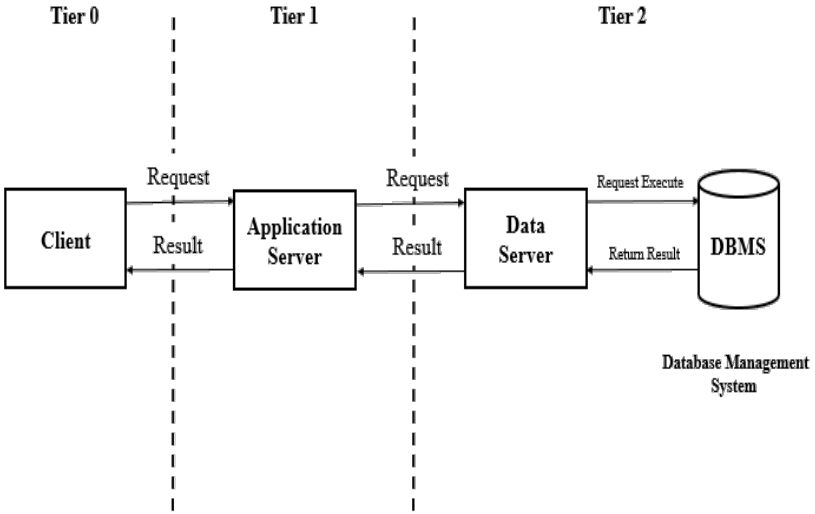
Fig: General Interaction between a client and a server

Application Layering

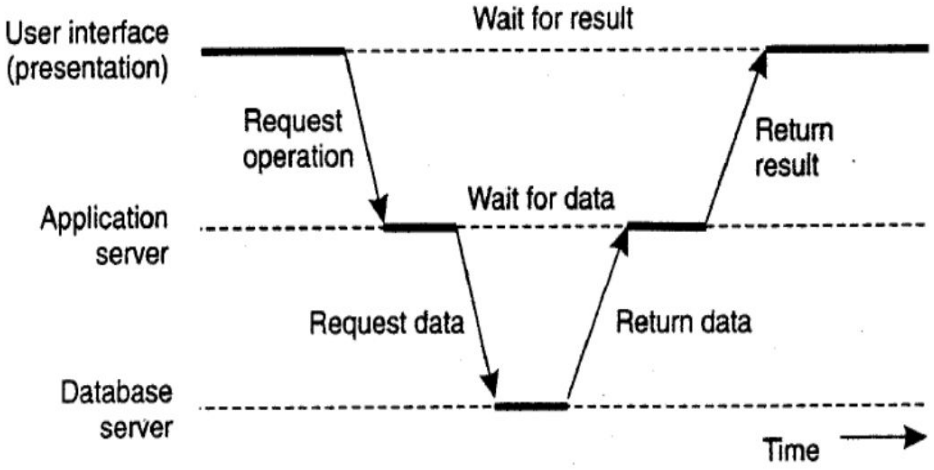
- Considering that many client-server applications are targeted toward supporting user access to databases, many people have advocated a distinction between the following three levels, essentially following the layered architectural style :
  1. The user-interface level
  2. The processing level
  3. The data level
- The user-interface level contains all that is necessary to directly interface with the user, such as display management. The processing level typically contains the applications. The data level manages the actual data that is being acted on.
- Clients typically implement the user-interface level. This level consists of the programs that allow end users to interact with applications.

Multi-tiered Architecture

3-tier client server architecture

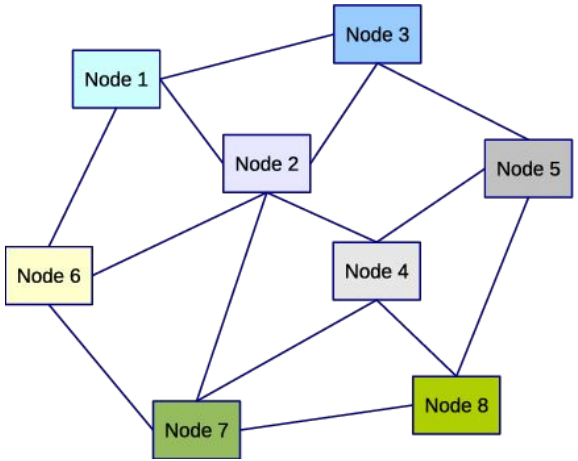


Example of server acting as a client



Decentralized Architecture

- The general idea behind peer to peer is where there is no central control in a distributed system. The basic idea is that, each node can either be a client or a server at a given time. If the node is requesting something, it can be known as a client, and if some node is providing something, it can be known as a server. In general, each node is referred to as a Peer.
- Due to the fact that there does not exist a central authority in the network, the solution to the single point of failure problem is inherent in the design of the architecture. As such, P2P architecture allows for the creation of a distributed networks where no one node is of critical importance to the network as whole.



Decentralized Architecture

- Unlike client-server architecture, P2P architecture becomes more powerful as the number of users increases. In client server architecture, the increase in requests from users will lower the availability of the server and may eventually cause the server to crash. However in P2P architecture, the amount of resources available to the network increases as more peers join. This makes P2P architecture extremely scalable.
- Two types of overlay networks exist: those that are structured and those that are not.

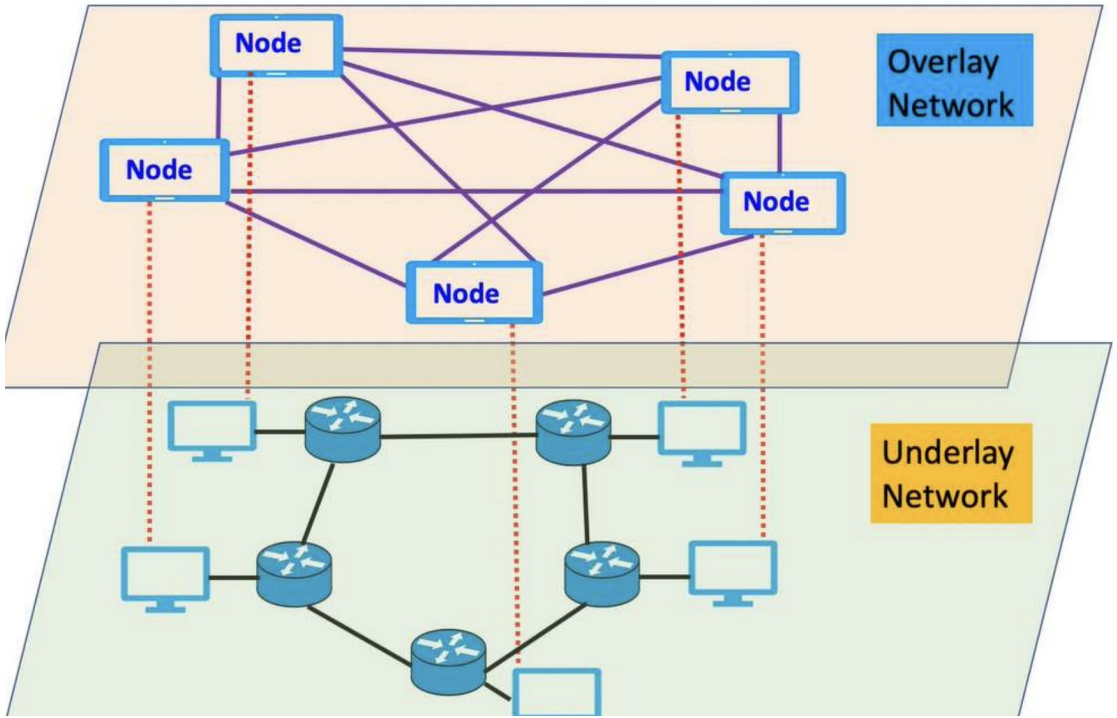
Overlay Networks

- An overlay network is a network that is built on top of another network (i.e. underlay network)
- Overlay network nodes are connected with each other through a logical or virtual link which corresponds to a path in the underlying network.
- Multiple overlay networks can exist on top of the same underlying network, where each implements its own specific services.

Examples

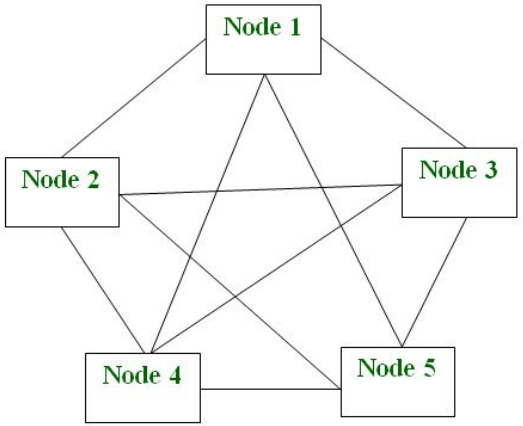
- Internet
- Peer-to-Peer (P2P) Networks
- Virtual Private Networks (VPNs)
- Voice over IP (VoIP) Services

Peer-to-Peer(P2P) Overlay Network



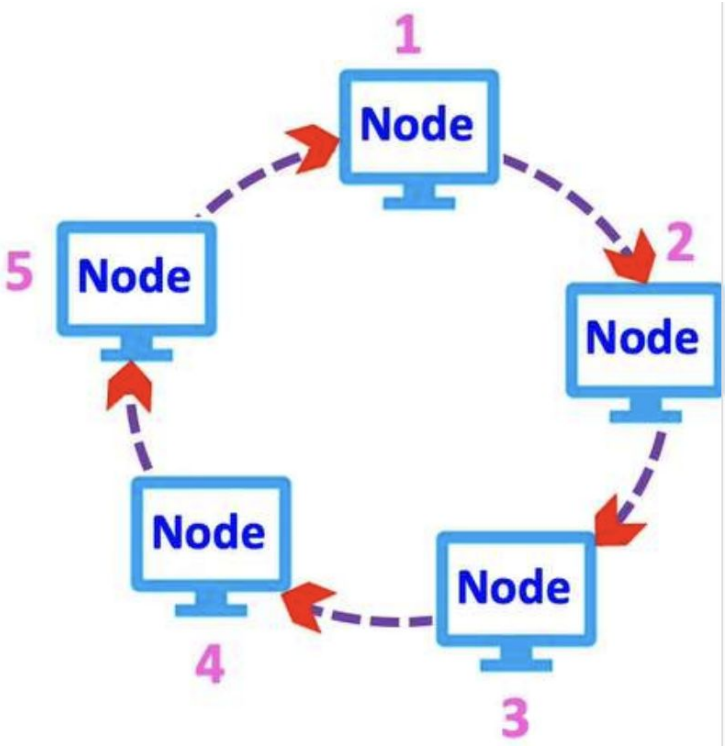
Unstructured Peer-to-Peer Network

- No specific structure
- Easy design
- Higher churn rate
- Inefficient searching
- Flooding-based searching method
- Examples: Napster, Gnutella, BitTorrent, Freenet, Gossip, Kazaa and Limewire



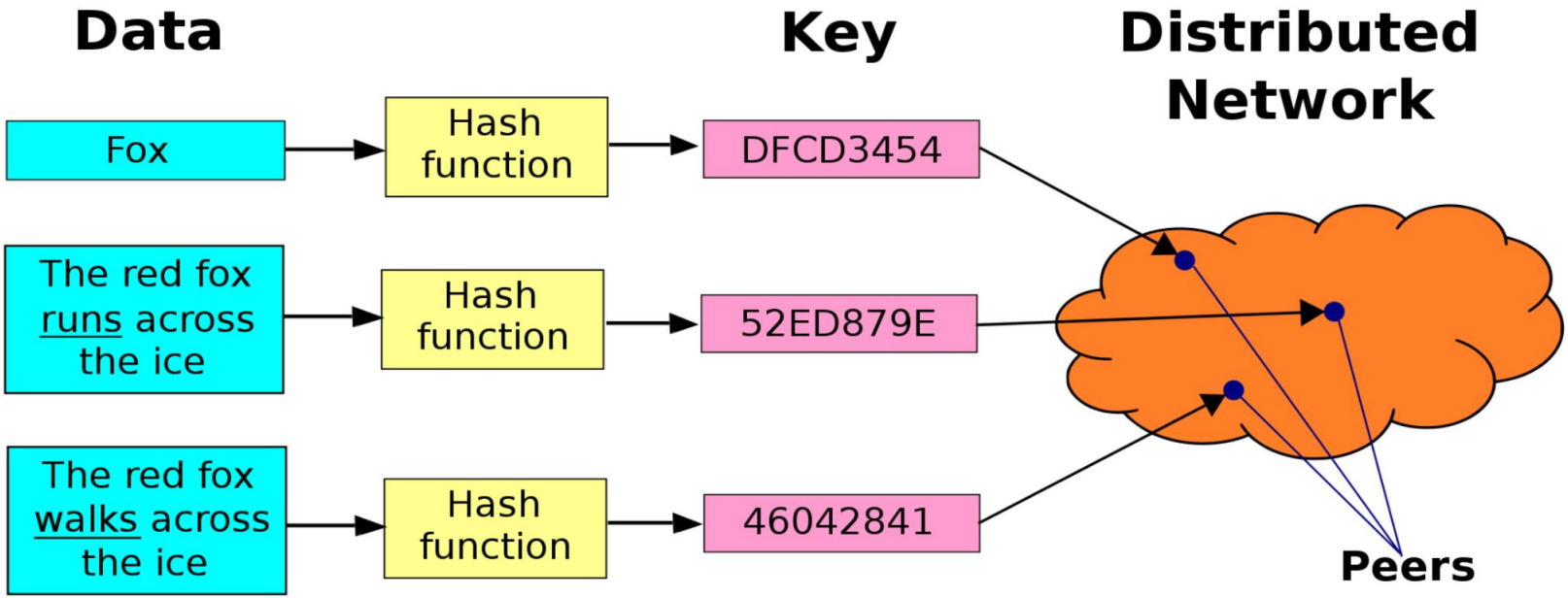
Structured Peer-to-Peer Network

- Specific structure
- Complex design
- Lower churn rate
- Efficient searching
- Hash-based searching method
- Examples: Chord, CAN, Tapestry, Viceroy, Kademila

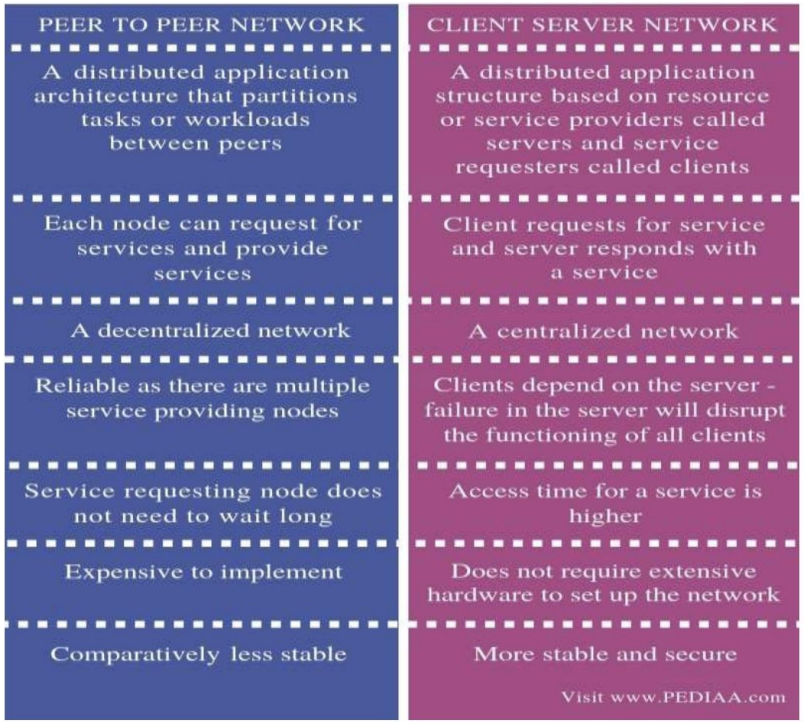


Structured Peer to Peer System

- In a structured peer-to-peer architecture, the overlay network is constructed using a deterministic procedure. By far the most-used procedure is to organize the processes through a distributed hash table (DHT). In a DHT -based system, data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier. Likewise, nodes in the system are also assigned a random number from the same identifier space.
- The crux of every DHT-based system is then to implement an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node.


Data
Key
Distributed Network
Fox
Hash function
DFCD3454
The red fox runs across the ice
Hash function
52ED879E
The red fox walks across the ice
Hash function
46042841
Peers

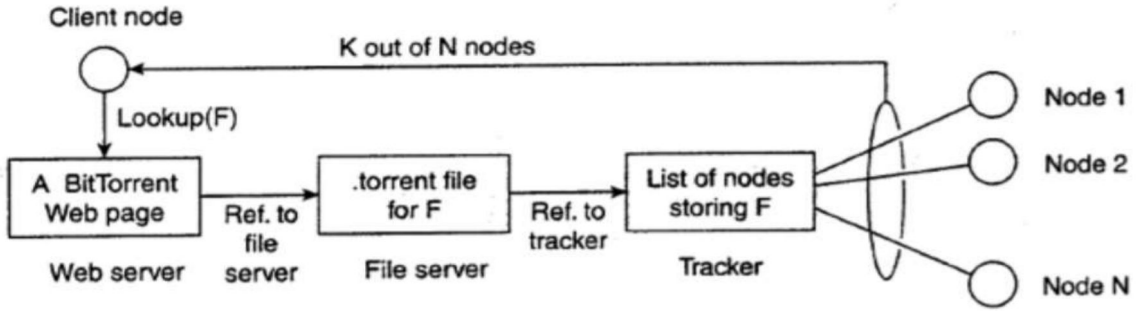
Peer-to-Peer network vs Client Server Network



PEER TO PEER NETWORK	CLIENT SERVER NETWORK
A distributed application architecture that partitions tasks or workloads between peers	A distributed application structure based on resource or service providers called servers and service requesters called clients
Each node can request for services and provide services	Client requests for service and server responds with a service
A decentralized network	A centralized network
Reliable as there are multiple service providing nodes	Clients depend on the server - failure in the server will disrupt the functioning of all clients
Service requesting node does not need to wait long	Access time for a service is higher
Expensive to implement	Does not require extensive hardware to set up the network
Comparatively less stable	More stable and secure

Hybrid Architecture

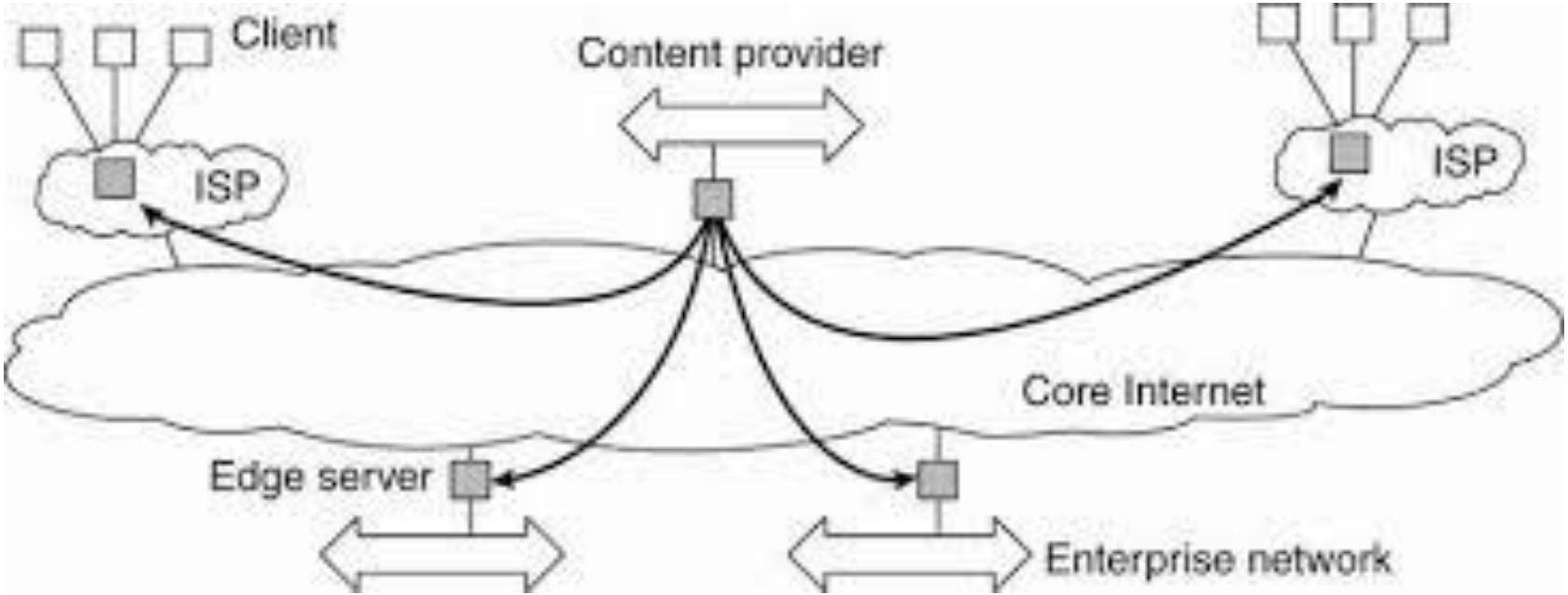
- Distributed systems in which client-server solutions are combined with decentralized architectures.
- Example: BitTorrent
- A centralized server is needed to let the client know about the nodes from which chunks of the file can be downloaded.
- Once the client joins the system as a node, a decentralized architecture will be used.



Hybrid Architecture Example: Super peers

- A super-peer is a node in a peer-to-peer network that operates both as a server to a set of clients, and as an equal in a network of super-peers.
- All communication from and to a regular peer proceeds through that peer's associated superpeer.

Hybrid Architecture Example: Edge Server System

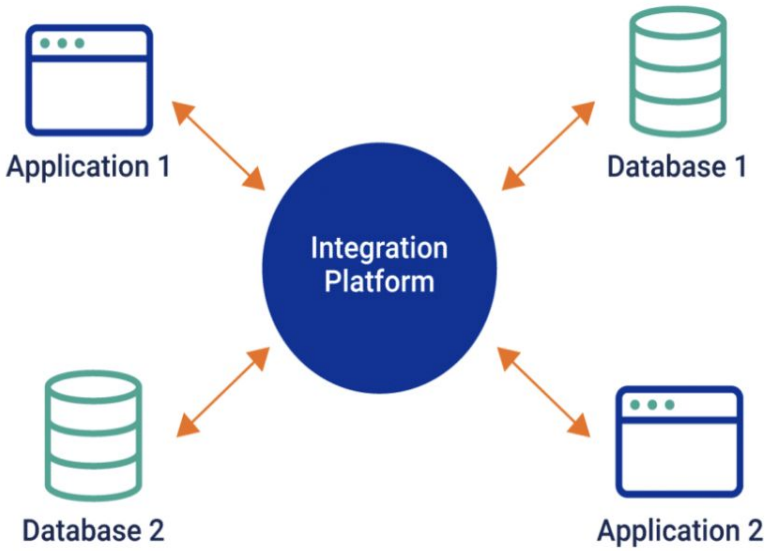


Hybrid Architecture Example: Edge Server System

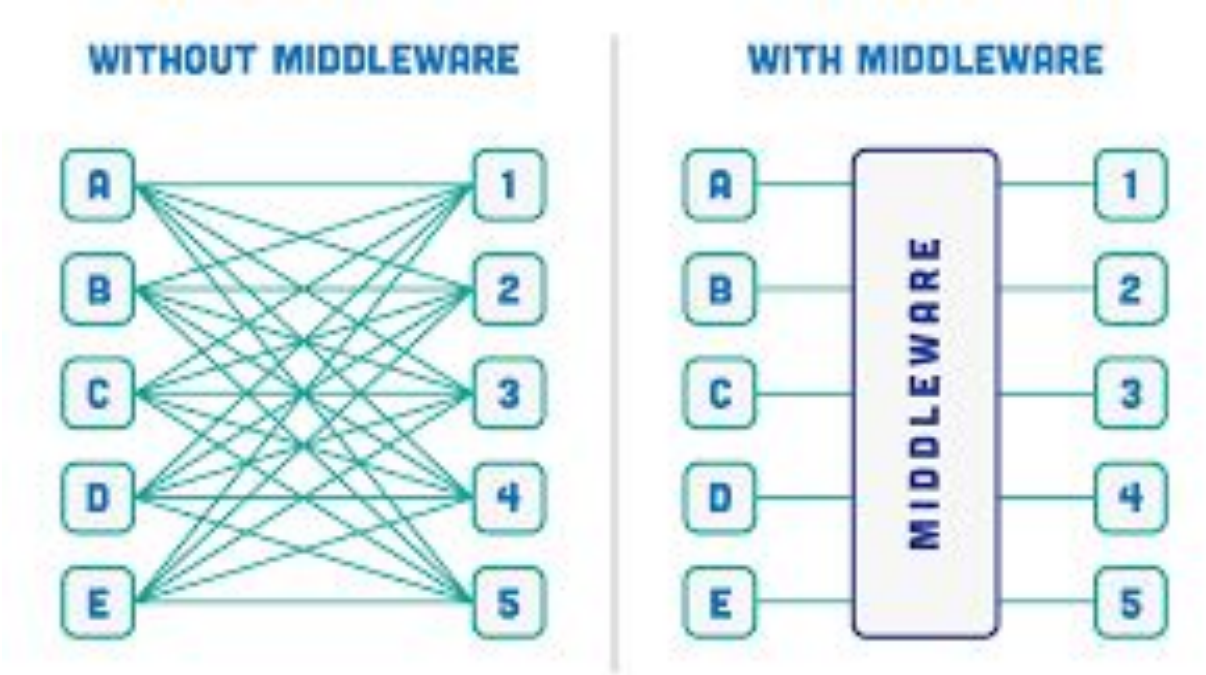
- Deployed on the Internet where servers are “at the edge” of the network (i.e. first entry to network)
- Each client connects to the Internet by means of an edge server. This edge is formed by the boundary between enterprise networks and the actual Internet. Example, as provided by an Internet Service Provider (ISP).
- The edge server's main purpose is to serve content as an origin server, after applying filtering and transcoding functions. This server can use other edge servers for replicating Web pages.
- A collection of edge servers can be used to optimize content and application distribution.

Middleware

- Middleware is the bridge that allows for information to be transported accurately between various applications, and it helps these separate applications run efficiently together. It’s very commonly referred to as “software glue”.



Middleware

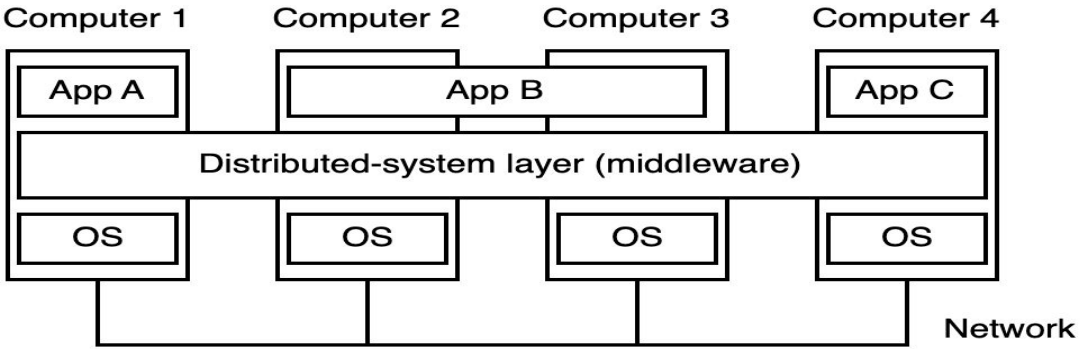

WITHOUT MIDDLEWARE
A
B
C
D
E
1
2
3
4
5
WITH MIDDLEWARE
A
B
C
D
E
MIDDLEWARE
1
2
3
4
5

Middleware

- Middleware in the context of distributed applications is software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data. Middleware supports and simplifies complex distributed applications.
- Middleware is a software layer situated between applications and operating systems. Middleware is typically used in distributed systems where it simplifies software development by doing the following:
  - Hides the intricacies 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 minimizes duplication of efforts and enhances collaboration between applications

Middleware

- Middleware is a software that acts as an intermediary between two applications or services to facilitate their communication.
- It’s sometimes called plumbing, as it connects two applications together so data and databases can be easily passed between the “pipe.”

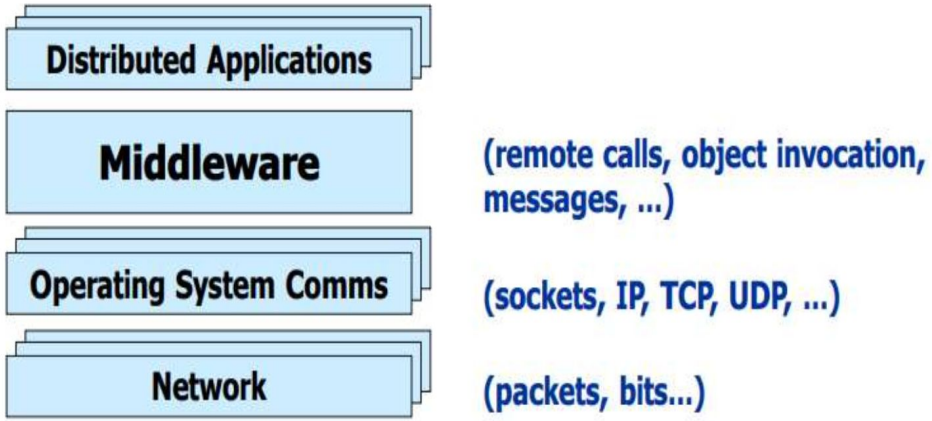


Middleware

- Most middleware follows object based and event based architectural style.

Middleware provides support for (some of):

- Naming, Location, Service discovery, Replication
- Protocol handling, Communication faults, QoS
- Synchronization, Concurrency, Transactions, Storage
- Access control, Authentication



Middleware

- Key Functions and Roles of Middleware in a Distributed System:
  - Communication Management: Facilitates communication between different applications or services running on various nodes in the network.
  - Resource Management: Manages resources such as memory, threads, and database connections across distributed components.
  - Data Management:Handles data consistency and synchronization across multiple nodes.
  - Security: Manages authentication, authorization, encryption, and other security services to protect data and communications in the distributed environment.
  - Interoperability: Allows different systems and applications to work together, even if they are built on different platforms or using different technologies.
  - Scalability and Fault Tolerance: Provides mechanisms for fault detection, recovery, and load balancing to ensure high availability and reliability.
  - Service Integration: Provides APIs, message brokers, and other tools for service discovery and integration.

Middleware

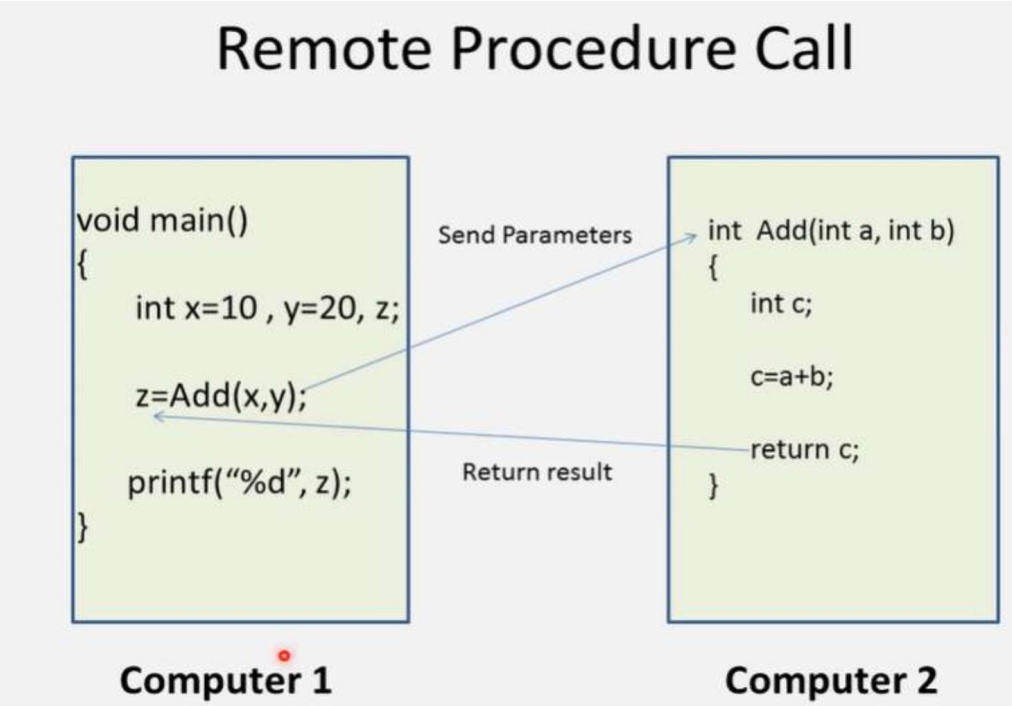
Different Types of Middleware:

- Remote Procedure Call (RPC): Client makes calls to procedures running on remote systems. Can be asynchronous or synchronous.
- Message-Oriented Middleware (MOM): enables application components using different messaging protocols to communicate to exchange messages. E.G. Java Message Service , IBM MQSeries ,Web Services, Microsoft Message Queue Server (MSMQ)
- Object-Oriented Middleware (OOM): This type of middleware makes it possible for applications to send objects and request services in an object-oriented system. E.g. Java RMI, CORBA
- Database Middleware: simplifies access to, and interaction with, back-end databases. E.g. Open Database Connectivity (ODBC), Java Database Connectivity (JDBC)

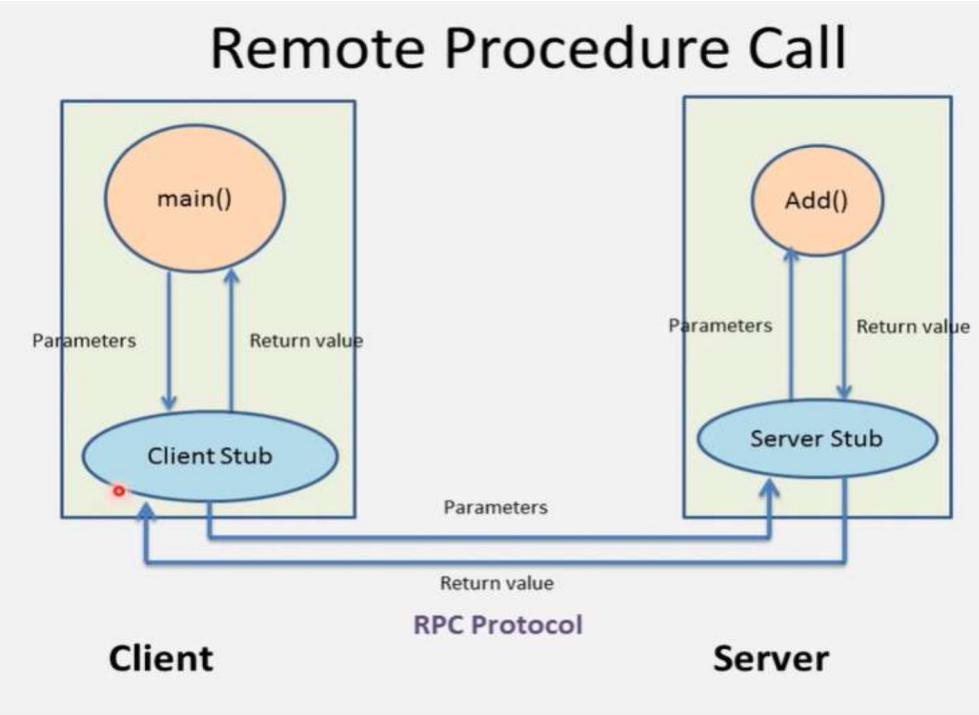
Remote Procedure Call (RPC)

- The concept of a remote procedure call (RPC), represents a major intellectual breakthrough in distributed computing.
- In RPC, procedures in processes on remote computers can be called as if they are procedures in the local address space. The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and the preserving of the required semantics for the procedure call.
- This approach directly and elegantly supports client-server computing with servers offering a set of operations through a service interface and clients calling these operations directly as if they were available locally.
- RPC systems therefore offer (at a minimum) access and location transparency.

Remote Procedure Call



Remote Procedure Call



Example Architectures

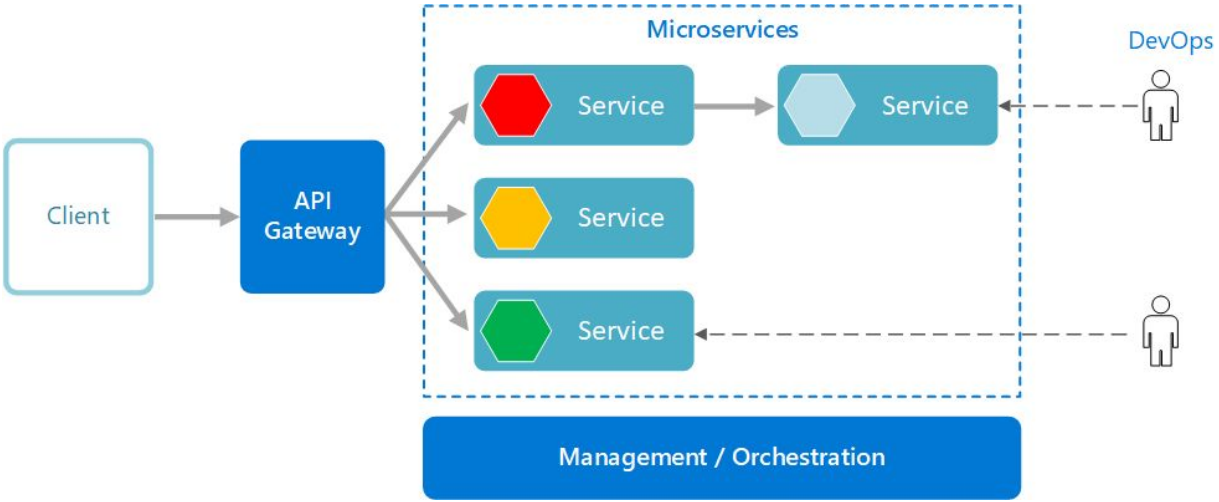
Some examples of distributed system architectures include:

- The three-tier architecture: in this architecture, the system is divided into three layers: the presentation layer, the application logic layer, and the data storage layer. The presentation layer is responsible for interacting with the user, the application logic layer contains the business logic of the system, and the data storage layer manages the storage and retrieval of data.

- The microservices architecture: in this architecture, the system is divided into small, independent components that communicate with each other through APIs. This allows for flexible and scalable development, but it can also increase the complexity of the system.

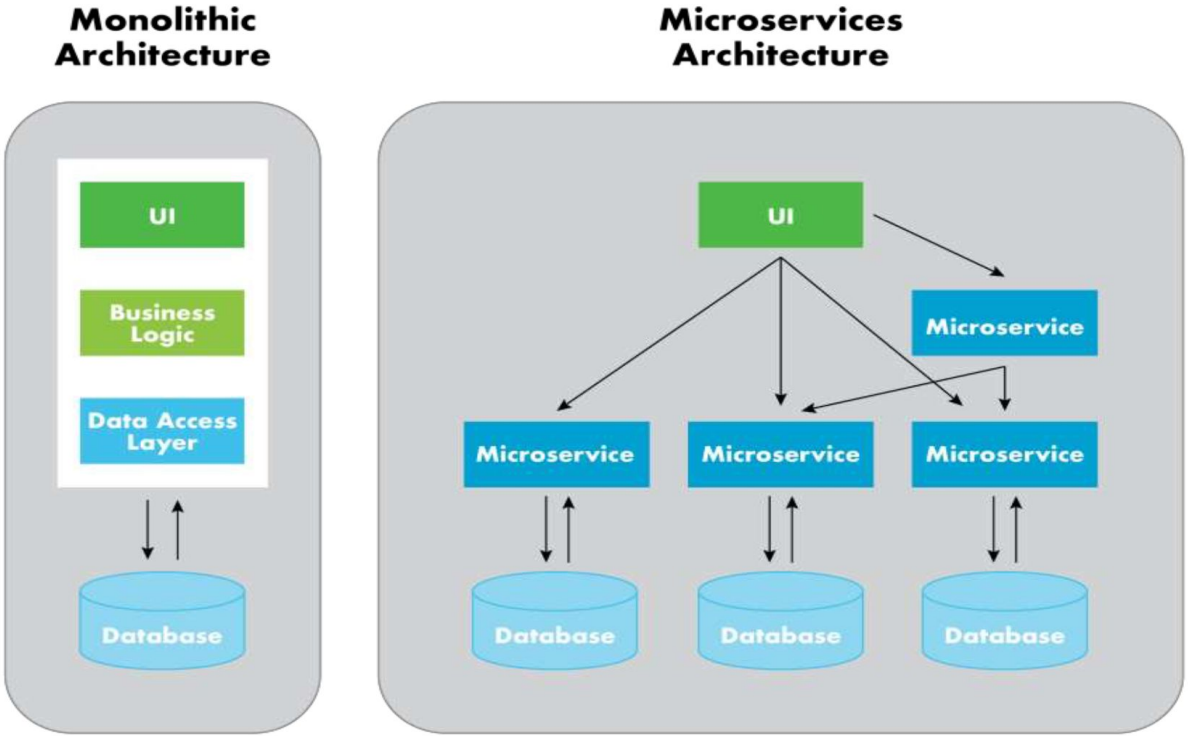
- The cloud computing architecture: in this architecture, users access services and resources over the internet. Cloud computing allows for on-demand access to computing resources and can improve scalability and reduce the need for hardware and software infrastructure

Microservices Architecture

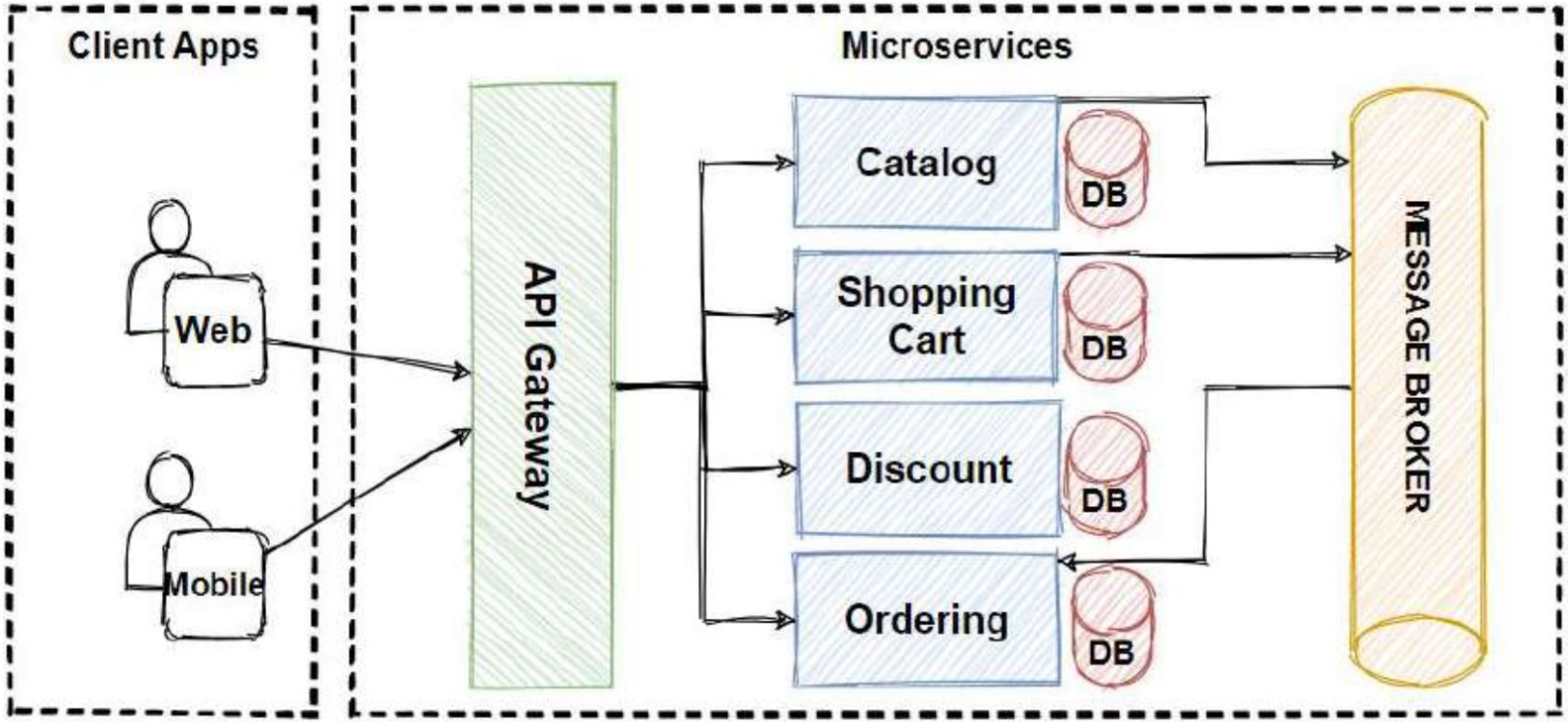


- Microservice are small business services that can work together and can be deployed autonomously / independently.
- These services communicate with each other by talking over the **network** and bring many advantages with them. One of the biggest advantages is that they can be d**eployed independently.**

Monolithic and Microservices architecture



Microservices Architecture



Microservices and Monoliths

	Microservices	Monoliths
Architecture	Collection of small services	Single build of a unified code
Scalability	Precise scaling and better usage of resources	Hard to scale
Time to Market	Easy to build and deploy	Complicated and time-consuming deployments
Reliability	Very reliable. If service fails, the application will not go down as a whole	If service fails, the entire application goes down

Microservices

Characteristics:

- Microservices are small, independent, and loosely coupled.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services communicate with each other by using well-defined APIs.