Distributed Systems Architectures

Moumita Asad
IIT, DU

Overview

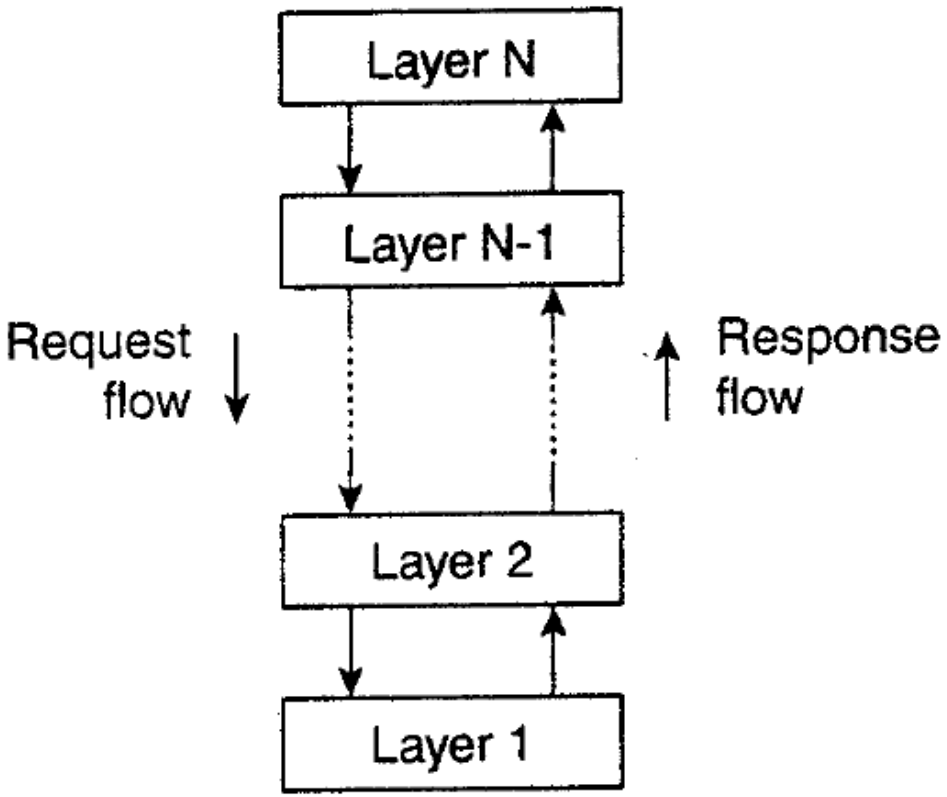
- Distributed systems are often complex pieces of software
- To master this complexity, systems must be properly organized
- Two key organization:
  1. **Software architecture:** the logical organization of a distributed system into software components
  2. **System architecture:** the placement of software components on physical machines

Software Architecture

- Important styles of architecture for distributed systems:
  - Layered Architectures
  - Object-Oriented, Service-Oriented Architectures, Microservices
  - Publish-Subscribe Architectures

Layered Architectural Style

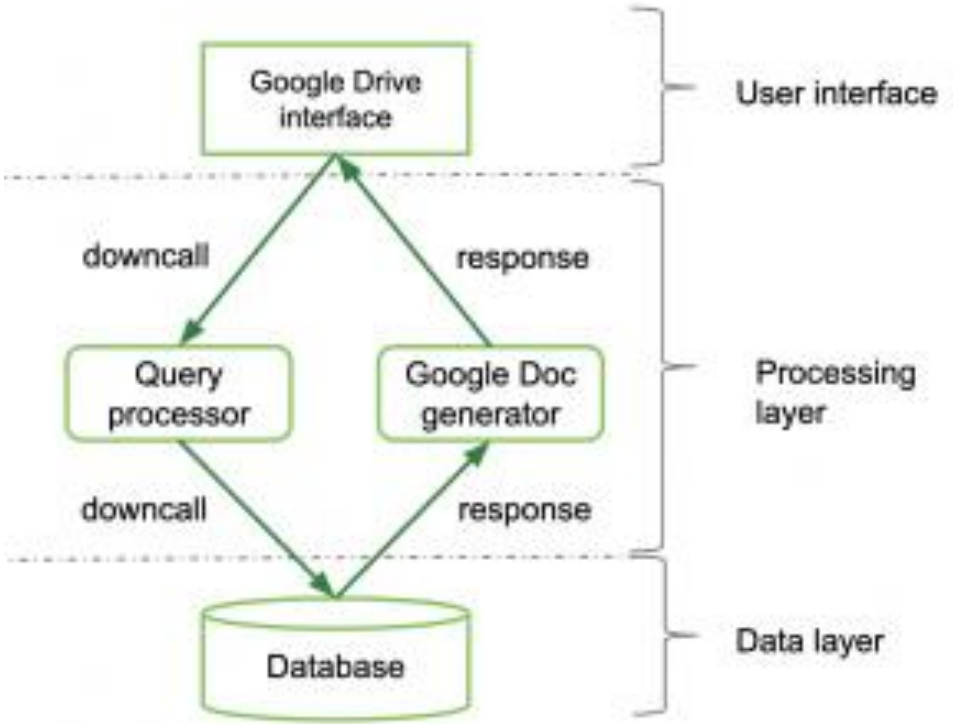
- Components are organized in layers
- Components on a higher layer make downcalls (send requests to a lower layer)
- Lower layer components respond to higher layer requests



Layered Architectural Style

- Google Docs consists of 3 layers:
  1. Interface layer: you request to see the latest doc from your drive.
  2. Processing layer: processes your request and asks for the information from the data layer.
  3. Data layer: stores persistent data (your file) and provides access to higher-level layers.
- The data layer returns the information to the processing layer which in turn sends it to the interface where you can view and edit it.

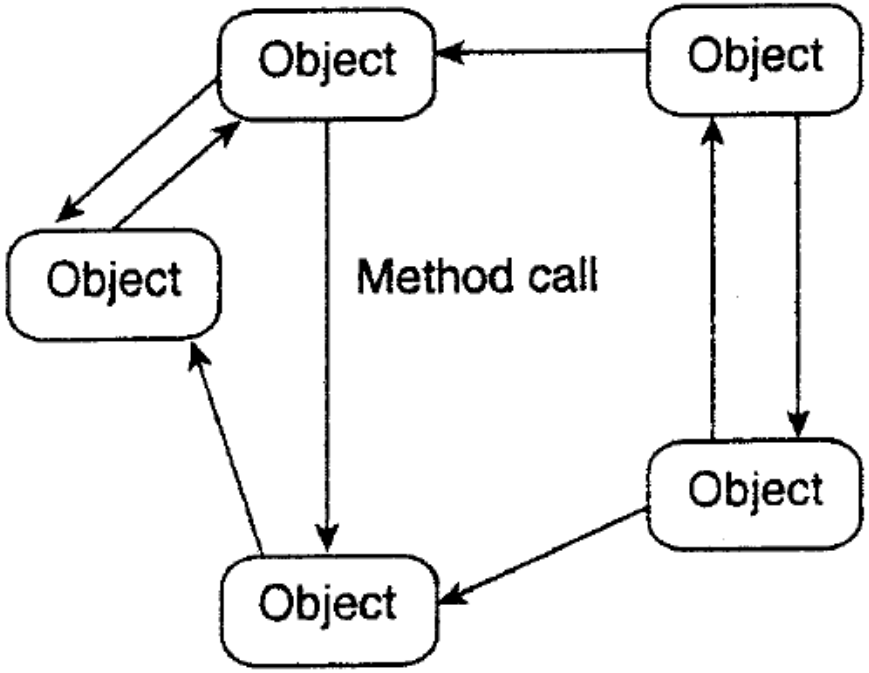
Layered Architectural Style



Object-based Architectural Styles

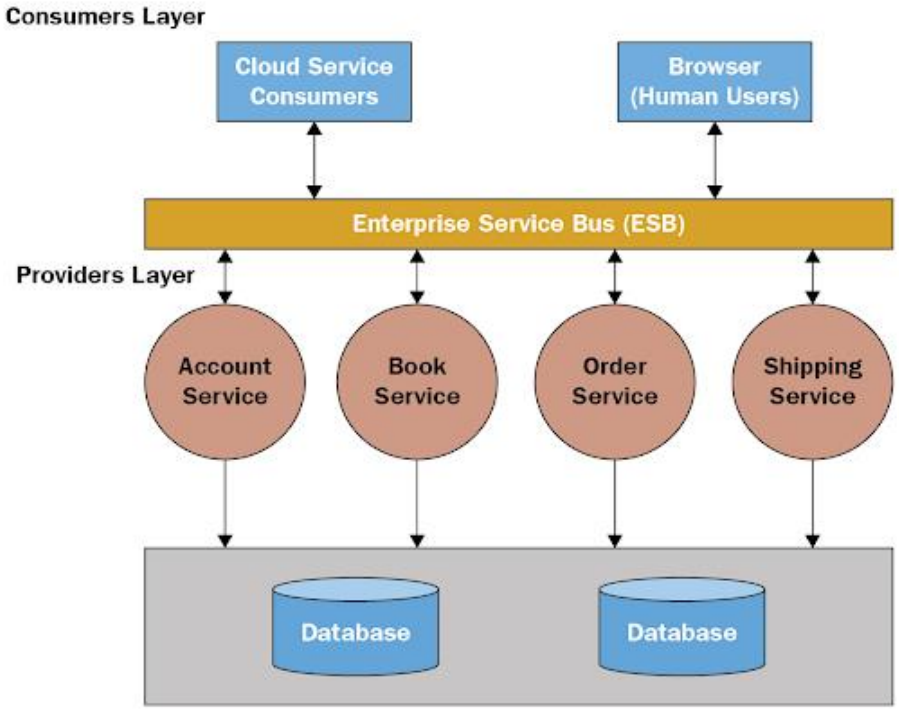
- A programming methodology
- Logical components are grouped together as objects
- Each object has its own encapsulated data set, referred to as the **object's state.**
- An **object's method** is the operations performed on that data.
- Objects are connected through **procedure call mechanisms** (an object "calls" on another object for specific requests)

Object-based Architectural Styles



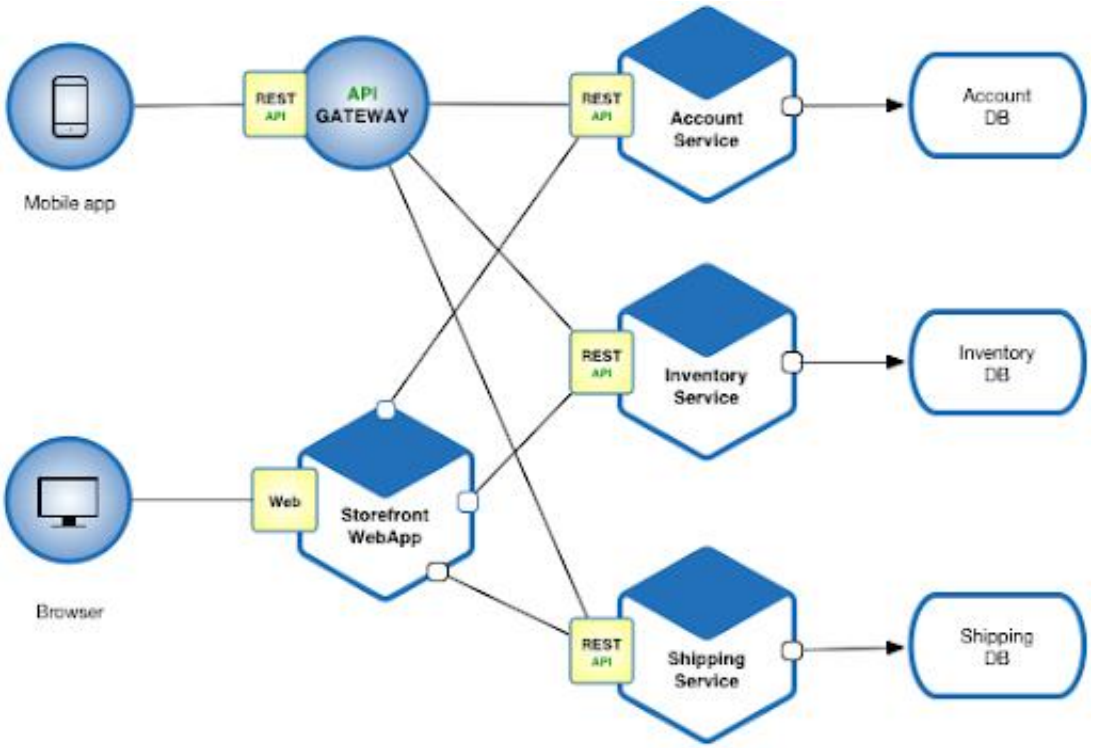
Service-Oriented Architecture

- An interface (the Enterprise Service Bus unifies all services together and exposes APIs for the frontend clients to communicate with the Providers layer



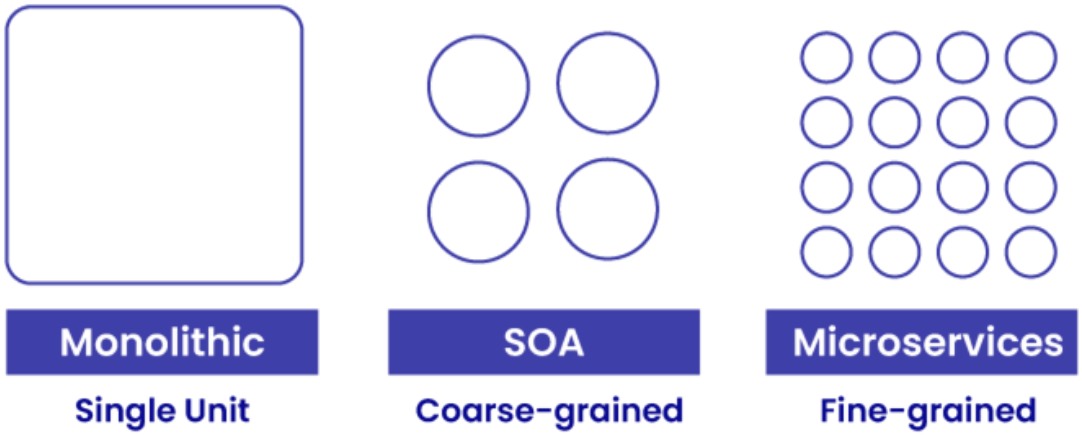
Microservices

- Microservices are smaller than services in an SOA, less tightly coupled, and more lightweight



Difference between SOA and Microservices

- SOA is “coarse grained”, meaning it focuses on large, business-domain functionalities
- Microservices is much “finer grained”, creating a mesh of functionalities that each has a single focus called a bounded context



Publish-Subscribe Architectures

- A loosely coupled architecture that allows processes to easily join or leave.
- The key difference here is how services communicate.
  - Instead of calling and getting a response, services send one-way, usually asynchronous messages, generally not to a specific receiver.
  - They rely on a configurator, administrator, or developer to configure who'll receive what message.
  - In some cases, the receivers themselves can sign up to receive messages.

Publish-Subscribe Architectures

- Example: how you get your breaking news push notifications. The Washington Post, for instance, publishes a news item categorized as “breaking news” and whoever subscribes to these updates will receive it

Remarks

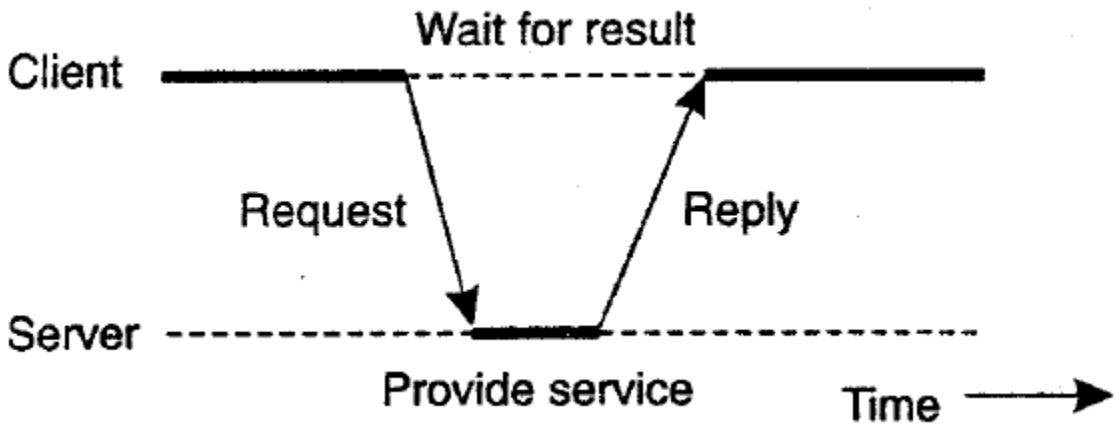
- Software architectures aim at achieving distribution transparency (at a reasonable level)
- However, it requires making trade-offs between performance, fault tolerance, ease-of-programming, and so on
- There is no single solution that will meet the requirements for all possible distributed applications

System Architecture

- Encompasses decisions as to
  - Where to place specific software components
  - Should certain components be placed on the same server or on different machines?
- Broadly 2 categories
  1. Centralized architecture
  2. Decentralized architecture

Centralized Architecture: Client-Server System

- **Server:** a process implementing a specific service (e.g. database service)
- **Client:** a process requesting that service from a server
- The client sends the request and waits for the reply (request-reply behavior)



Communication between Client-Server

1. **Connectionless protocol**
   - Usage: when the underlying network is fairly reliable (e.g., local-area networks)
   - Advantage: efficient
   - What if a message gets lost?

What if a Message Gets Lost?

- let the client resend the request when no reply message comes in
- Problem: the client cannot detect whether the original request message was lost or the transmission of the reply failed
- If the reply was lost, then resending a request may result in performing the operation twice

What if a Message Gets Lost?

- If the operation was "transfer $10,000 from my bank account," it would be better that we simply reported an error instead
- If the operation was "tell me how much money I have left," it would be perfectly acceptable to resend the request
- When an operation can be repeated multiple times without harm, it is said to be idempotent
- Since some requests are idempotent and others are not it should be clear that there is no single solution for dealing with lost messages

Communication between Client-Server

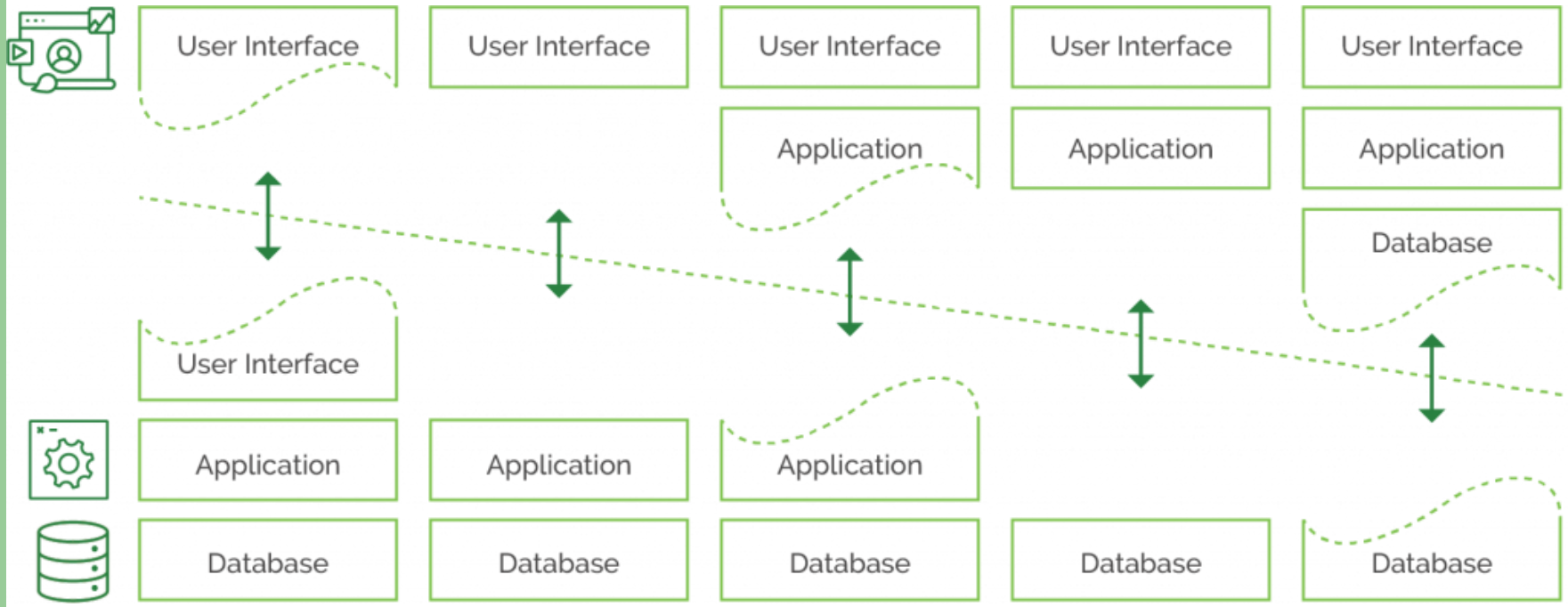
2. Connection-oriented protocol

- whenever a client requests a service, it first sets up a connection to the server before sending the request. The server generally uses that same connection to send the reply message, after which the connection is torn down
- Usage: good for wide-area systems in which communication is inherently unreliable
- Disadvantage: relatively low performance

Distribution of Client-Server Application

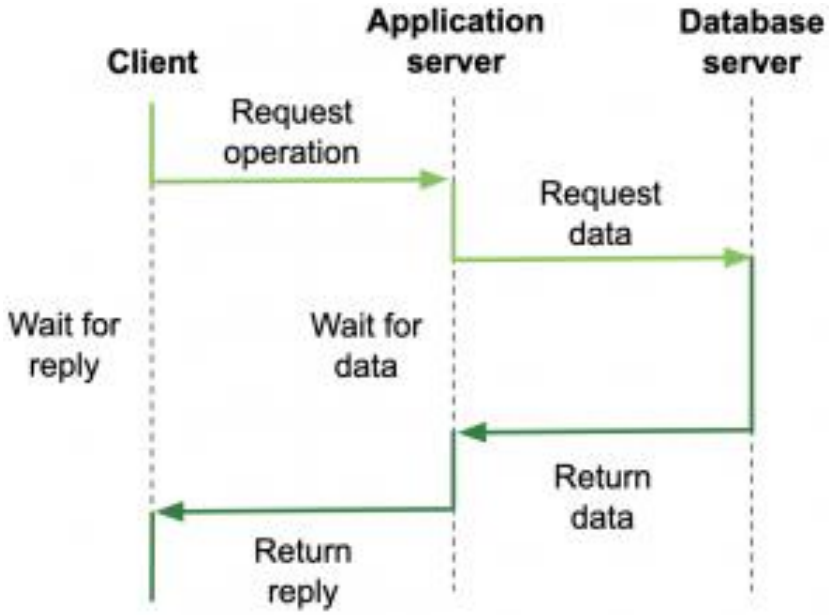
- **(Physically) two-tiered architecture**
  - Two types of machines:
    1. a client machine with the user interface
    2. the server interface containing the program implementing the process and the data

Variations of Physically Two-Tiered Architecture



Distribution of Client-Server Application

- **(Physically) three-tiered architectures**
  - The application is spread across three machines- one client and two servers
  - One of the servers also has to act as a client
    - One of the servers may need input from the other server to process the client request, acting as a client



Decentralized Architectures

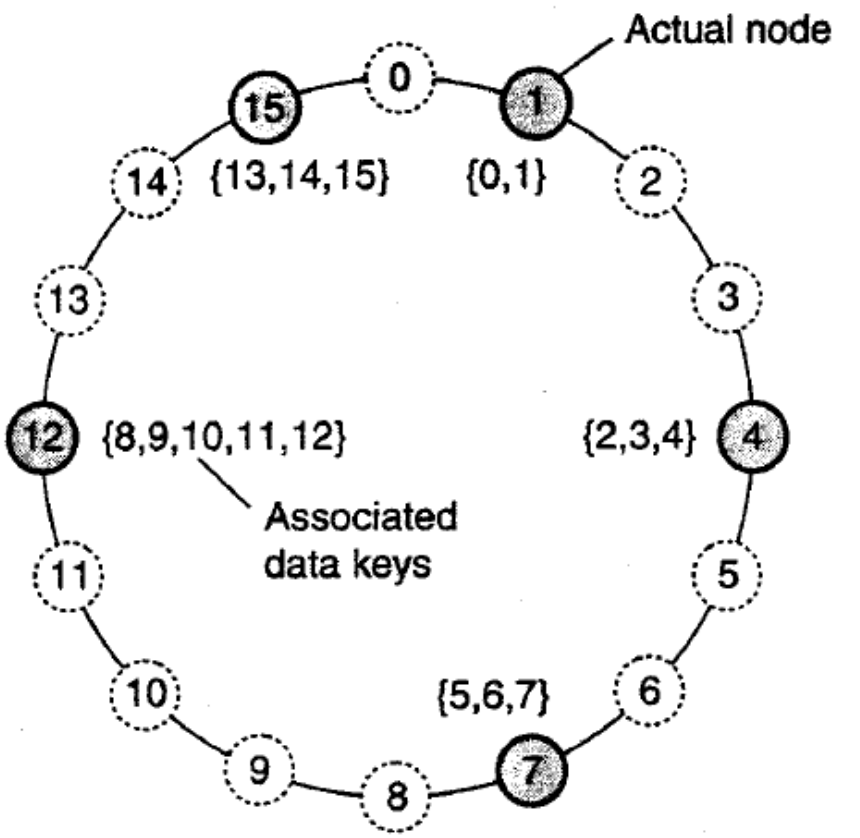
- Processes that constitute a peer-to-peer system are all equal
- Each process will act as a client and a server at the same time (which is also referred to as acting as a servant)
- Two types of overlay (logical) networks exist
  1. Structured
  2. Unstructured

Structured Peer-to-Peer Architectures

- 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 key of a data item is mapped to the identifier of a node
- When looking up a data item, the network address of the node responsible for that data item is returned

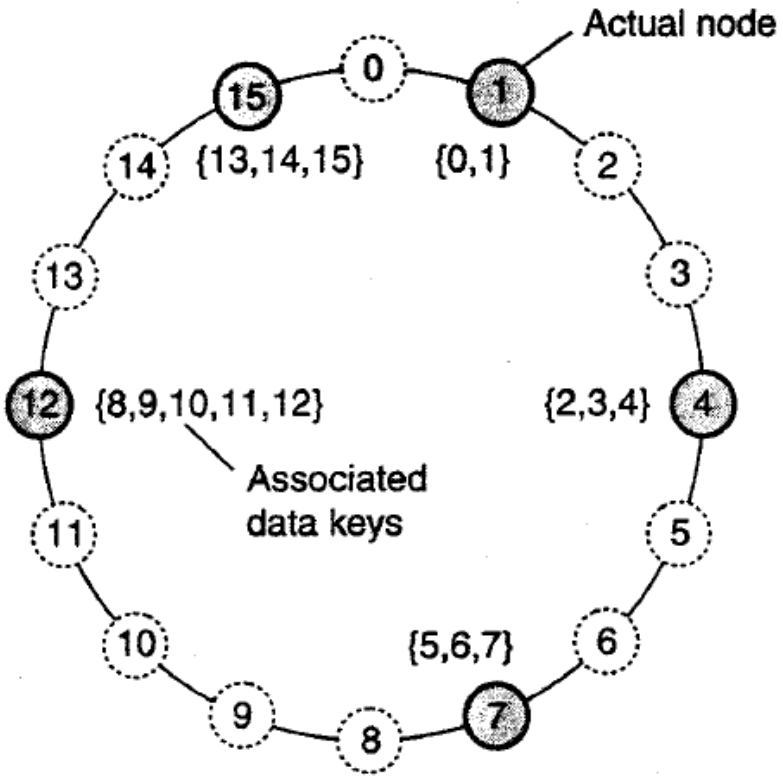
Structured Peer-to-Peer Architectures

- In the Chord system, the nodes are logically organized in a ring such that a data item with key *k* is mapped to the node with the smallest identifier *id* ~ *k*. This node is referred to as the *successor* of key *k* and denoted as *succ(k)*

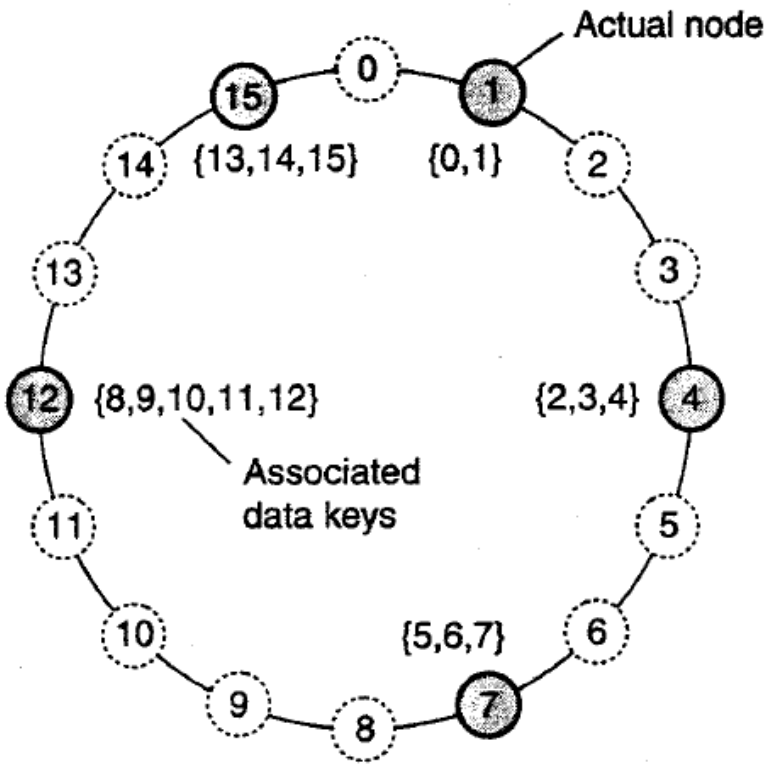


Structured Peer-to-Peer Architectures

- To actually look up the data item, an application running on an arbitrary node would then call the *lookup(k)* function which would subsequently return the network address of *succ(k)*
- Next, the application can contact the node to obtain a copy of the data item



- When a node wants to join the system, it generates a random identifier *id*
  - if the identifier space is large enough and the random number generator is of good quality, the probability of generating an identifier that is already assigned to an actual node is close to zero

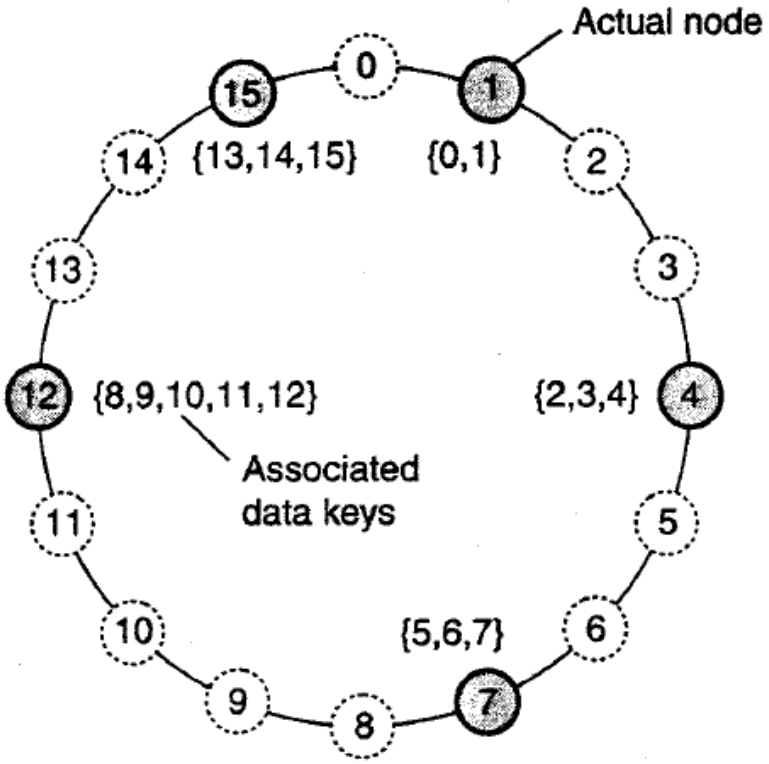


Structured Peer-to-Peer Architectures

- Then, the node can simply do a lookup on *id*, which will return the network address of *succ(id)*
- Next, the joining node can simply contact *succ(id)* and its predecessor and insert itself in the ring
- This scheme requires that each node also stores information on its predecessor
- Insertion also yields that each data item whose key is now associated with node *id*, is transferred from *succ(id)*

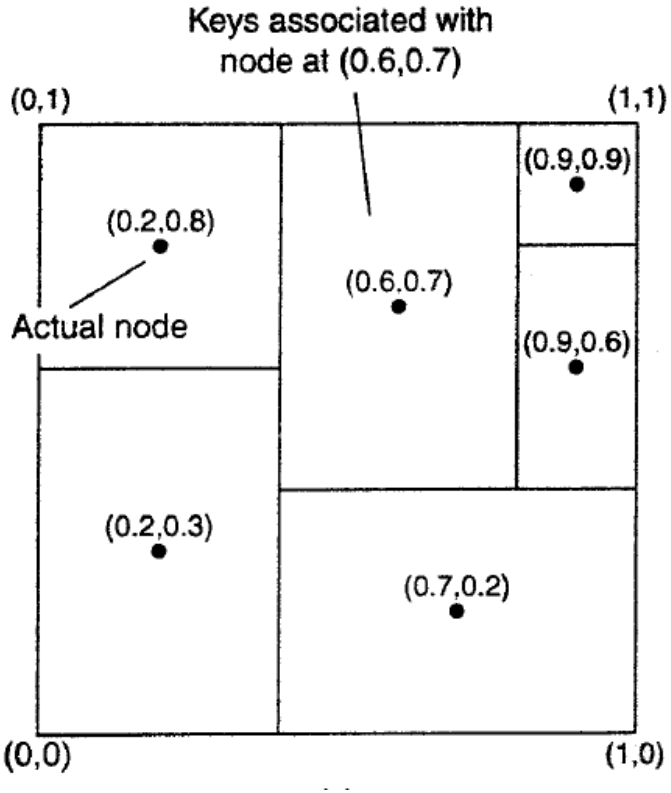
Structured Peer-to-Peer Architectures

- To leave the system, node *id* informs its departure to its predecessor and successor, and transfers its data items to *succ(id)*



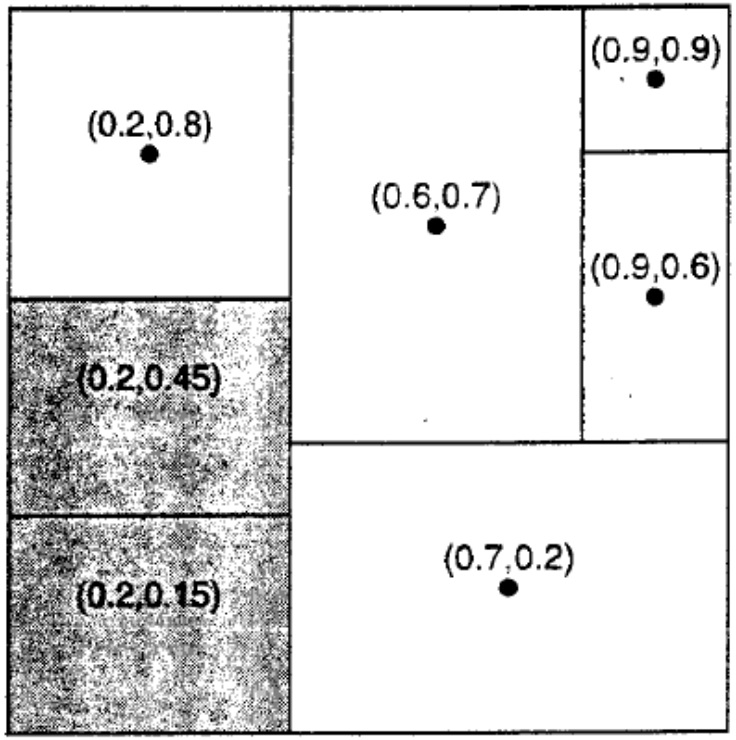
Structured Peer-to-Peer Architectures

- Content Addressable Network (CAN) deploys a d-dimensional Cartesian coordinate space, which is completely partitioned among all the nodes that participate in the system
- Every data item in CAN is assigned a unique point in this space, after which it is also clear which node is responsible for that data



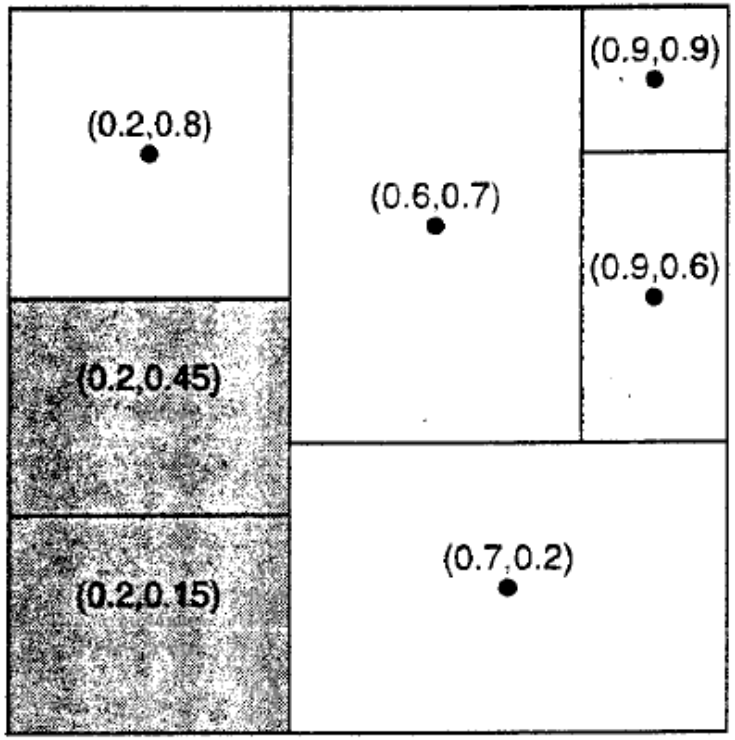
Structured Peer-to-Peer Architectures

- When a node *P* wants to join a CAN system, it picks an arbitrary point from the coordinate space and subsequently looks up the node Q in whose region that point falls
- Node Q then splits its region into two halves and one half is assigned to the node *P*

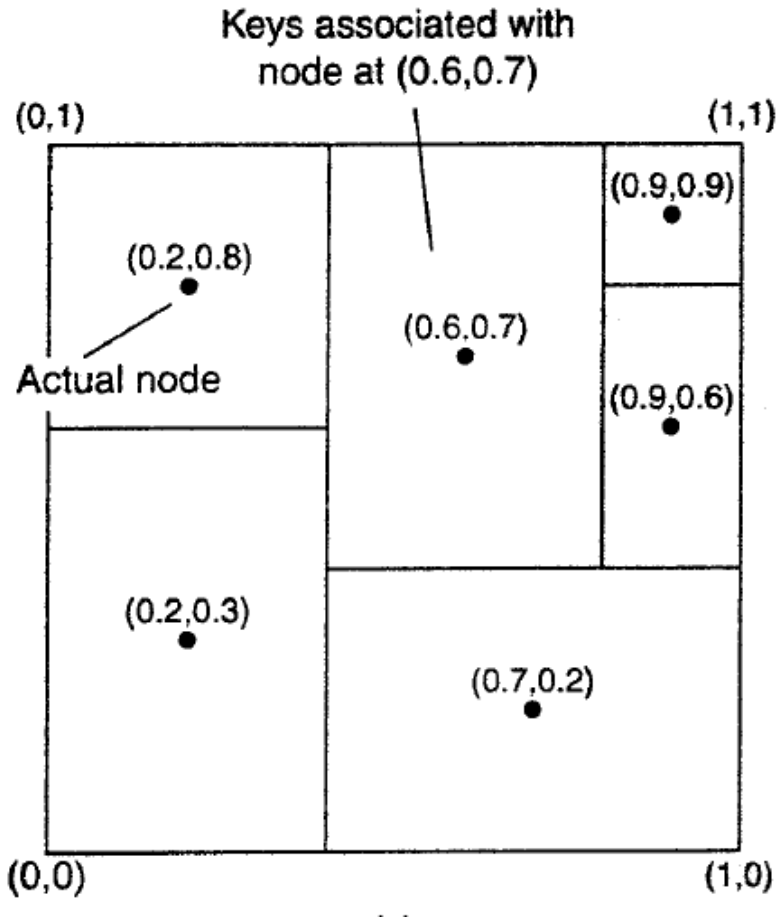


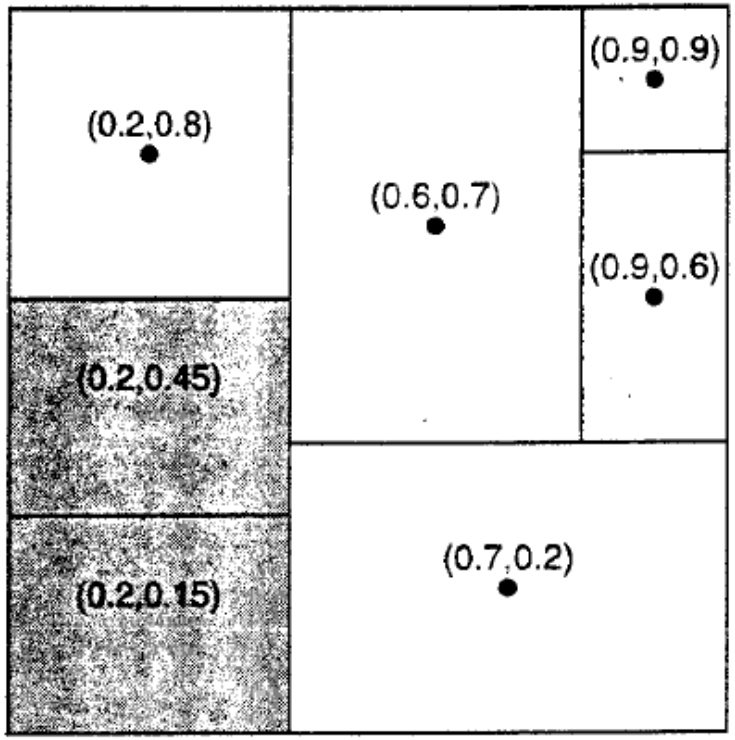
Structured Peer-to-Peer Architectures

- Nodes keep track of their neighbors (nodes responsible for adjacent region)
- When splitting a region, the joining node *P* can easily come to know who its new neighbors are by asking node *Q*
- The data items for which node *P* is now responsible are transferred from node *Q*



Structured Peer-to-Peer Architectures





Resources

- https://thenewstack.io/primer-understanding-software-and-system-architecture/
- https://scoutapm.com/blog/soa-vs-microservices