DS ARCHITECTURES

Architectural Styles

- It is the Logical organization of distributed systems into software components
- Such a style is formulated in terms of components:
  - How components are connected to each other,
  - The data exchanged between components
  - How these elements are jointly configured into a system.

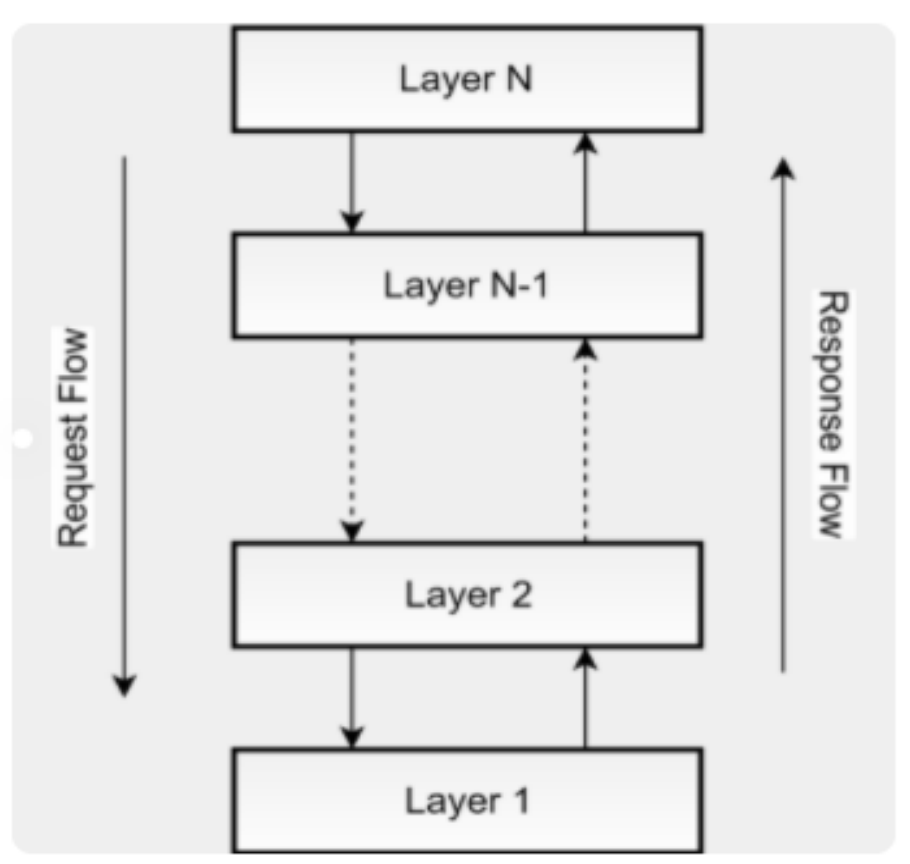
Layered architectures

- Components are organized in a layered fashion where a component at layer L; is allowed to call components at the underlying layer but not the other way around
- An key observation is that control generally flows from layer to layer: The layers on the bottom provide a service to the layers on the top. The request flows from top to bottom, whereas the response/results is sent from bottom to top

Layered architectures cont'd

- 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 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. The following image is the basic idea of a layered architecture style.

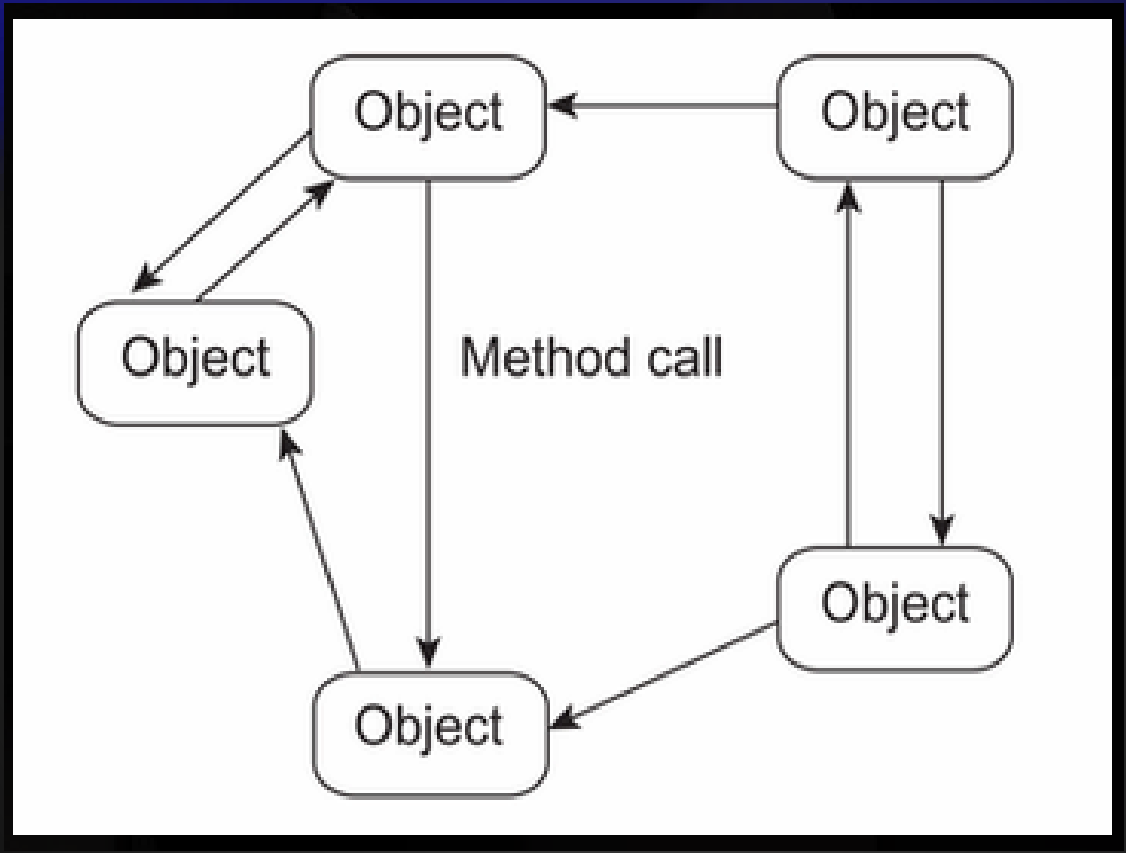
Layered architectures cont'd



Object-based architectures

- Each of the components are referred to as objects, and these components are connected through a (remote) procedure call mechanism/ interact with other objects through a given connector or interface.
- This architecture style is based on loosely coupled arrangement of objects. This has no specific architecture like layers.
- These are much more direct where all the different components can interact directly with other components through a direct method call.

Object-based architectures cont'd



Data-centered

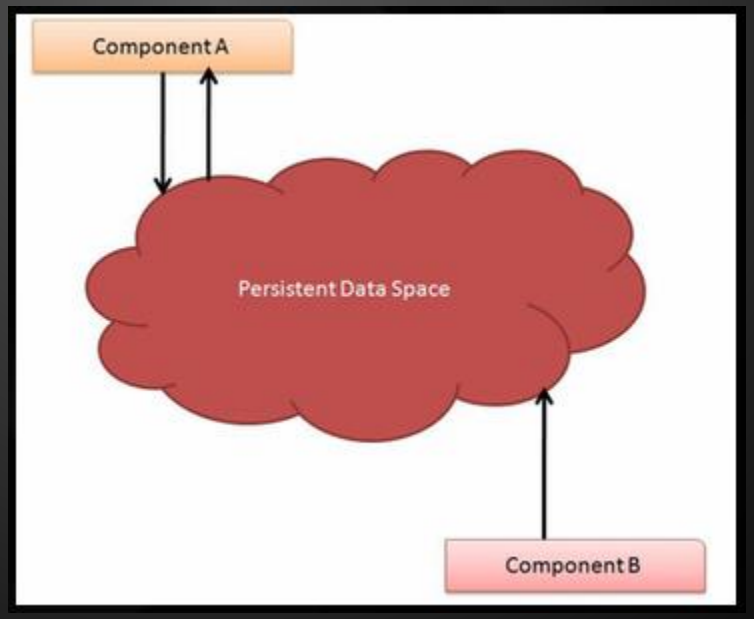
- Evolve around the idea that processes communicate through a common repository
- This common repository can be either active or passive
- This is more like a producer consumer problem. The producers produce items to a common data store, and the consumers can request data from it. This common repository, could even be a simple database

Data-centered cont'd

- For example, a wealth of networked applications have been developed that rely on a shared distributed file system in which virtually all communication takes place through files.
- Likewise, Web-based distributed systems are largely data-centric: processes communicate through the use of shared Web-based data services
- Communication between objects happens through shared common storage. This supports different components (or objects) by providing a persistent storage space for those components (such as a MySQL database).

Data-centered cont'd

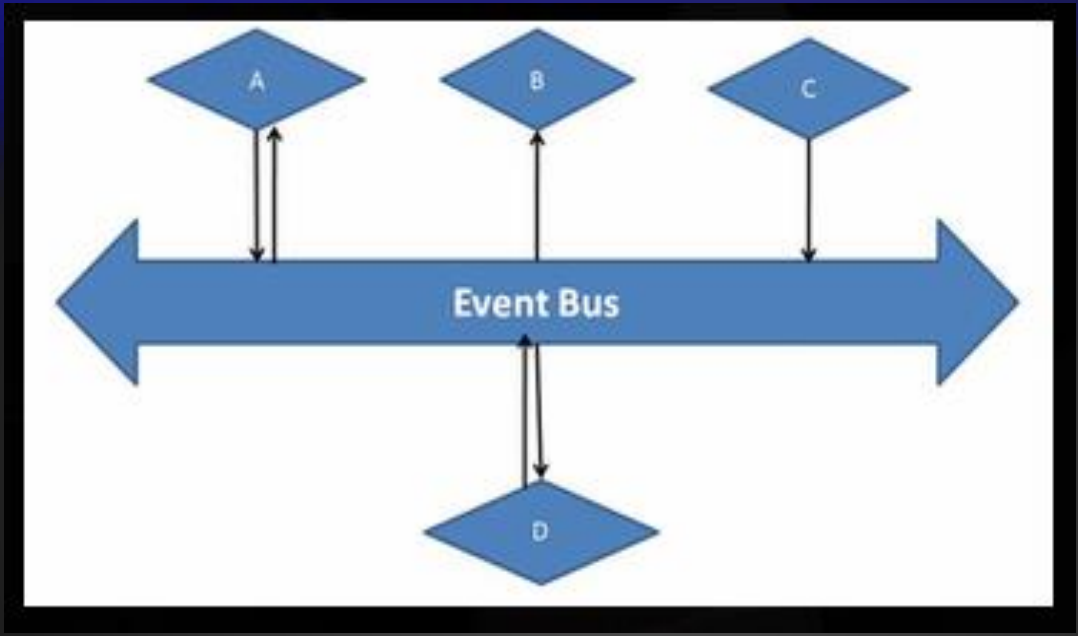
- All the information related to the nodes in the system are stored in this persistent storage.
- In event-based architectures, data is only sent and received by those components who have already subscribed.



Event-based

- Processes essentially communicate through the propagation of events, which optionally also carry data
- 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
- Processes are loosely coupled

Event-based



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.

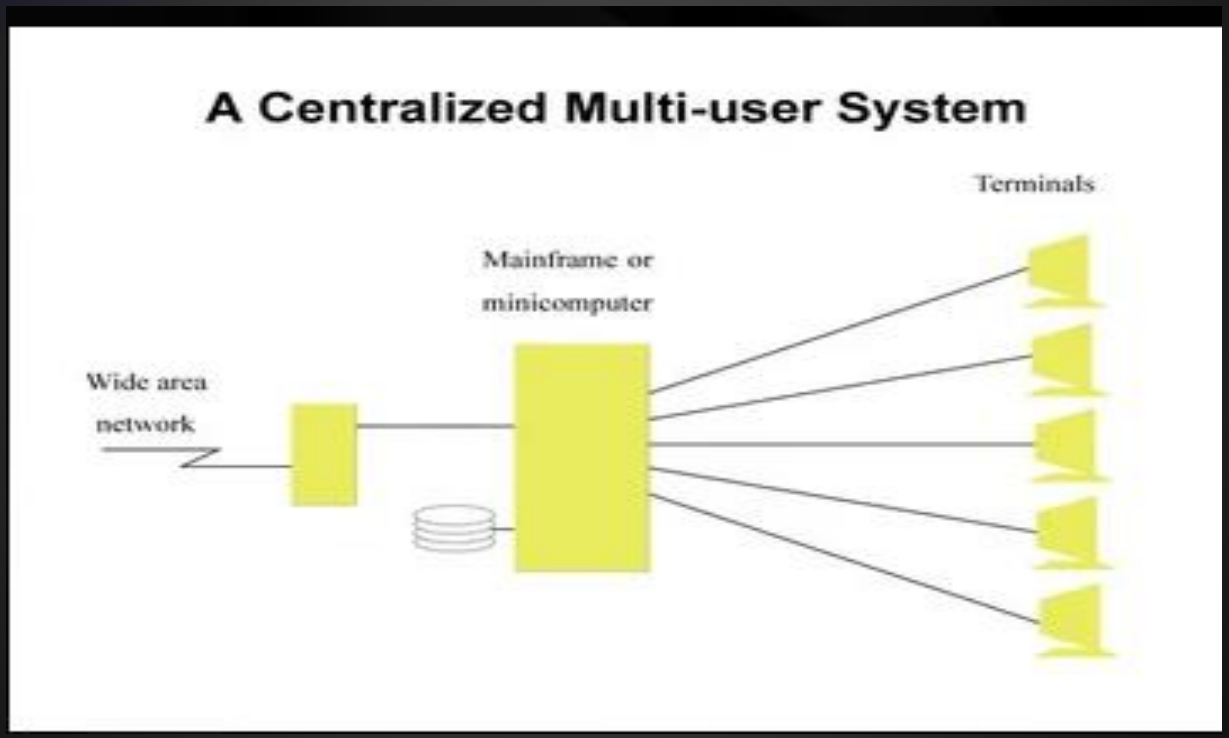
SYSTEM ARCHITECTURES

- Addresses the question of where software components are placed; their interaction, and their placement
  - Centralized
  - Decentralized

Centralized Architectures

- Client Server Model
- The two main structures within distributed system overlays are Centralized and Decentralized architectures.
- The centralized architecture can be explained by a simple client-server architecture where the server acts as a central unit

Centralized Architectures cont'd



Centralized Architectures cont'd

- Client Server Model e.g. search engine
  - The user-interface level
  - The processing level
  - The data level

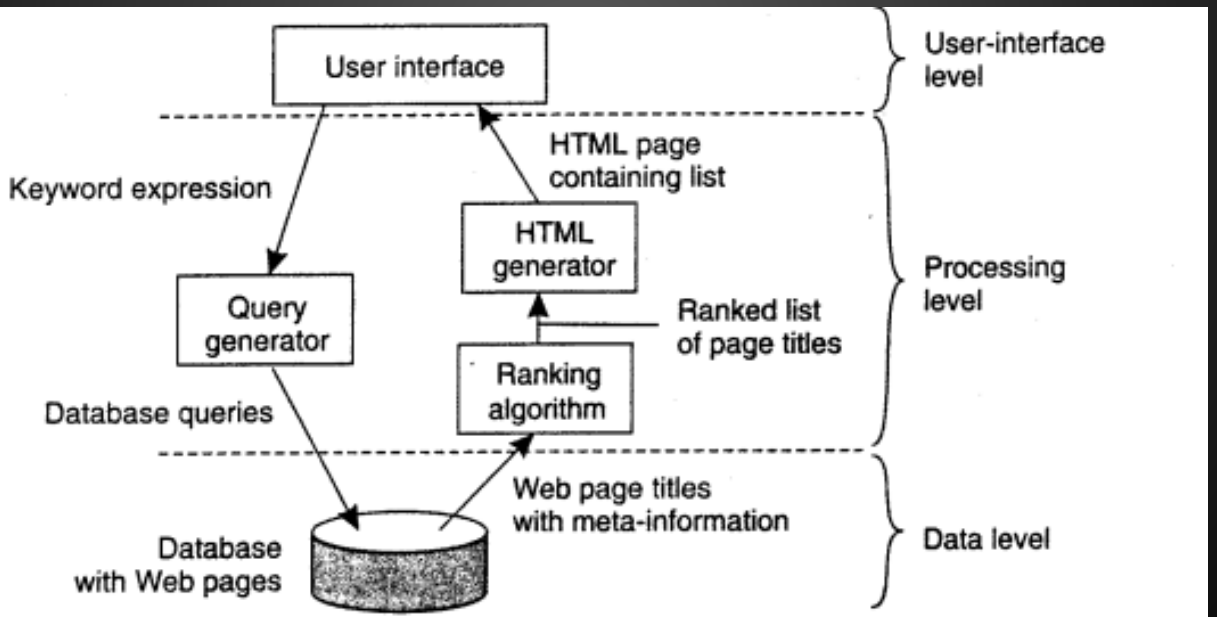


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

Decentralized Architectures

- Vertical Distribution
  - Distributed processing provided by using Client-Server architectures which are divided into multitiered architectures, a concept referred to vertical distribution.
  - The characteristic feature of vertical distribution is that it is achieved by placing logically different components on different machines
  - Functions are logically and physically split across multiple machines, where each machine is tailored to a specific group of functions

Decentralized Architectures

- Horizontal Distribution
  - A client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load

What is Middleware?

- Middleware is the software between the application programs and the operating System and base networking
- Integration Fabric that knits together applications, devices, systems software, data
- Middleware provides a comprehensive set of higher-level distributed computing capabilities and a set of interfaces to access the capabilities of the system.

More Views of Middleware

- Are Software technologies that help manage complexity and heterogeneity inherent to the development of distributed systems, distributed applications, and information systems
- Provides Higher-level programming abstraction for developing the distributed application

Middleware Systems – more views

- Aims at reducing the burden of developing distributed application for developer informally called "plumbing", i.e., like pipes that connect entities for communication often called "glue code", i.e., it glues independent systems together and makes them work together ; it masks the heterogeneity programmers of distributed applications have to deal with
  - network & hardware
  - operating system & programming language
  - different middleware platforms
  - location, access, failure, concurrency, mobility, ...
- often also referred to as transparencies, i.e., network transparency, location transparency

Middleware Systems Views

- An operating system is "*the software that makes the hardware usable*"
- Similarly, a middleware system makes the distributed system programmable and manageable
- A computer without OS could barely be programmed, so could the distributed application be developed without middleware
- Programs could be written in assembly, but higher-level languages are far more productive for this purpose

Middleware Architecture