chapter 2

- **Architectures of distributed systems**
- **Fundamental Models**

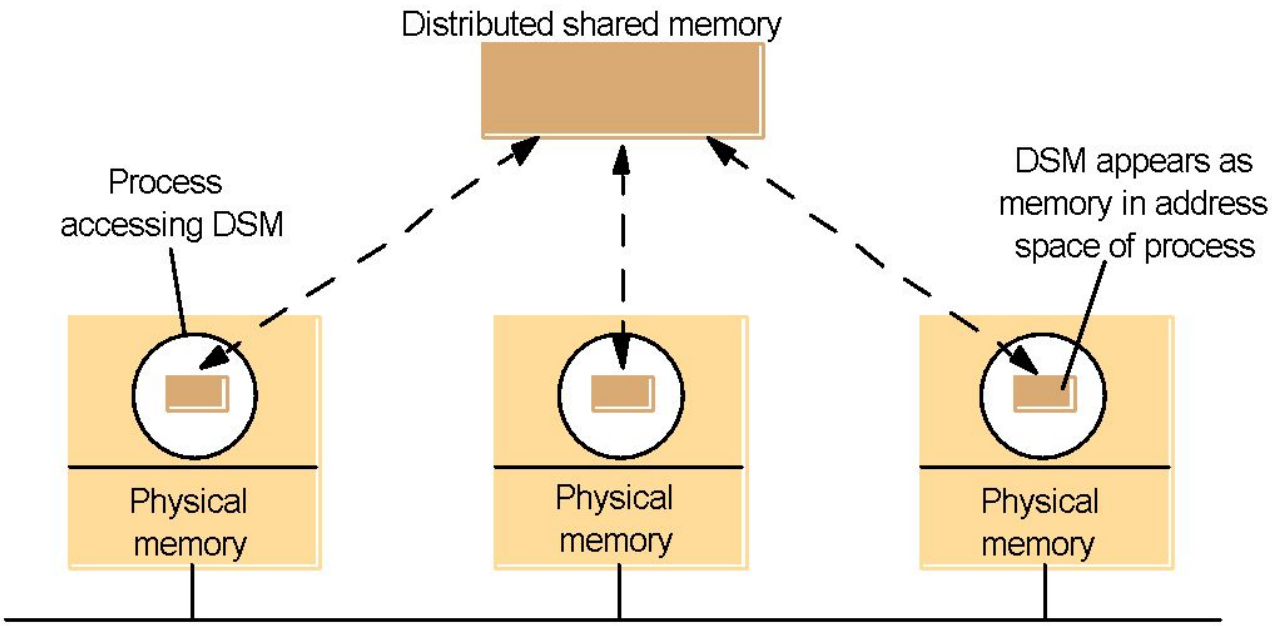
Architectures of Distributed Systems

- **Tightly coupled**
  - Highly integrated machines that may look as a single computer.
- **Loosely coupled** (share nothing)
  - Client-Server
  - Peer-to-Peer
  - A Key differentiation based on the Programming Interface:
    - Distributed Objects
    - Web Services

Tightly coupled systems

- **Distributed shared memory** (DSM)provides the illusion of a single shared memory: it spares the programmer the concerns of message passing.

Figure 18.1
The distributed shared memory abstraction



Architectural styles

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

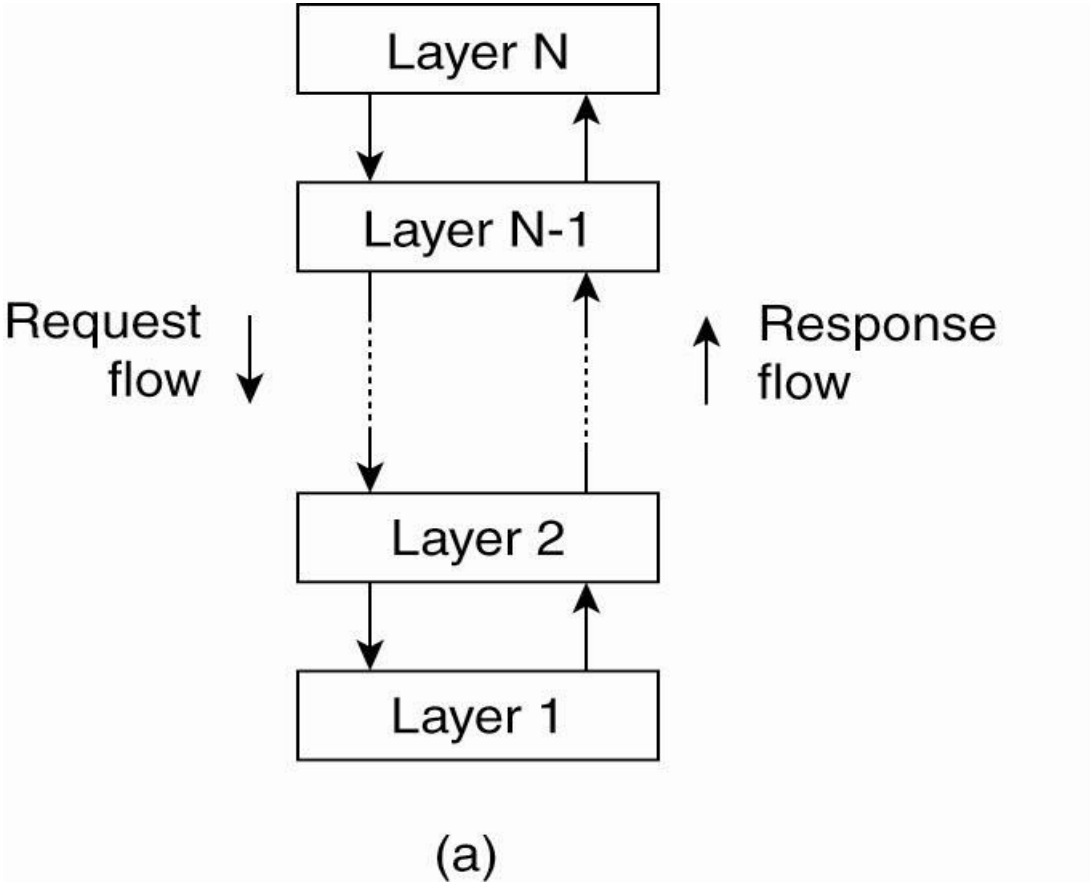
- Components: A modular unit with well-defined interfaces;replaceable ;reusable
- Connectors : A communication link between mediates coordination or cooperation among components.

Architectural Styles
(talking about loosely-coupled systems)

Based on the logical organization of the components of distributed systems:

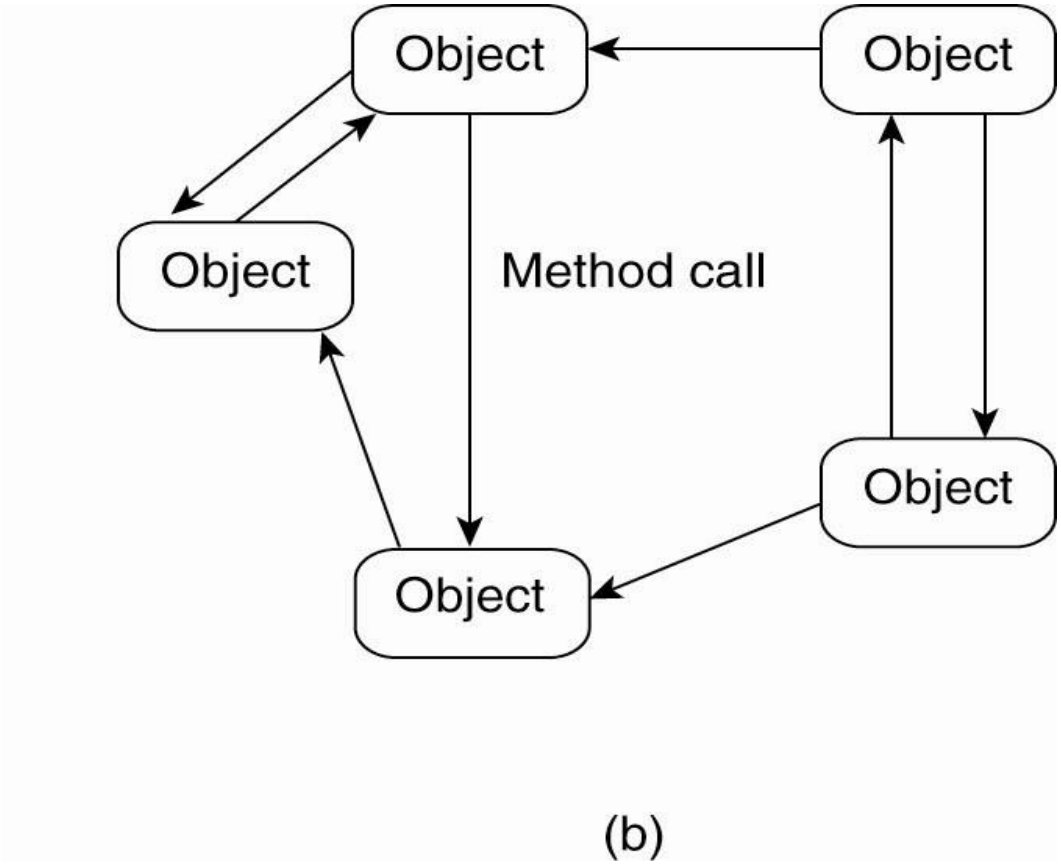
- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

The layered architectural style



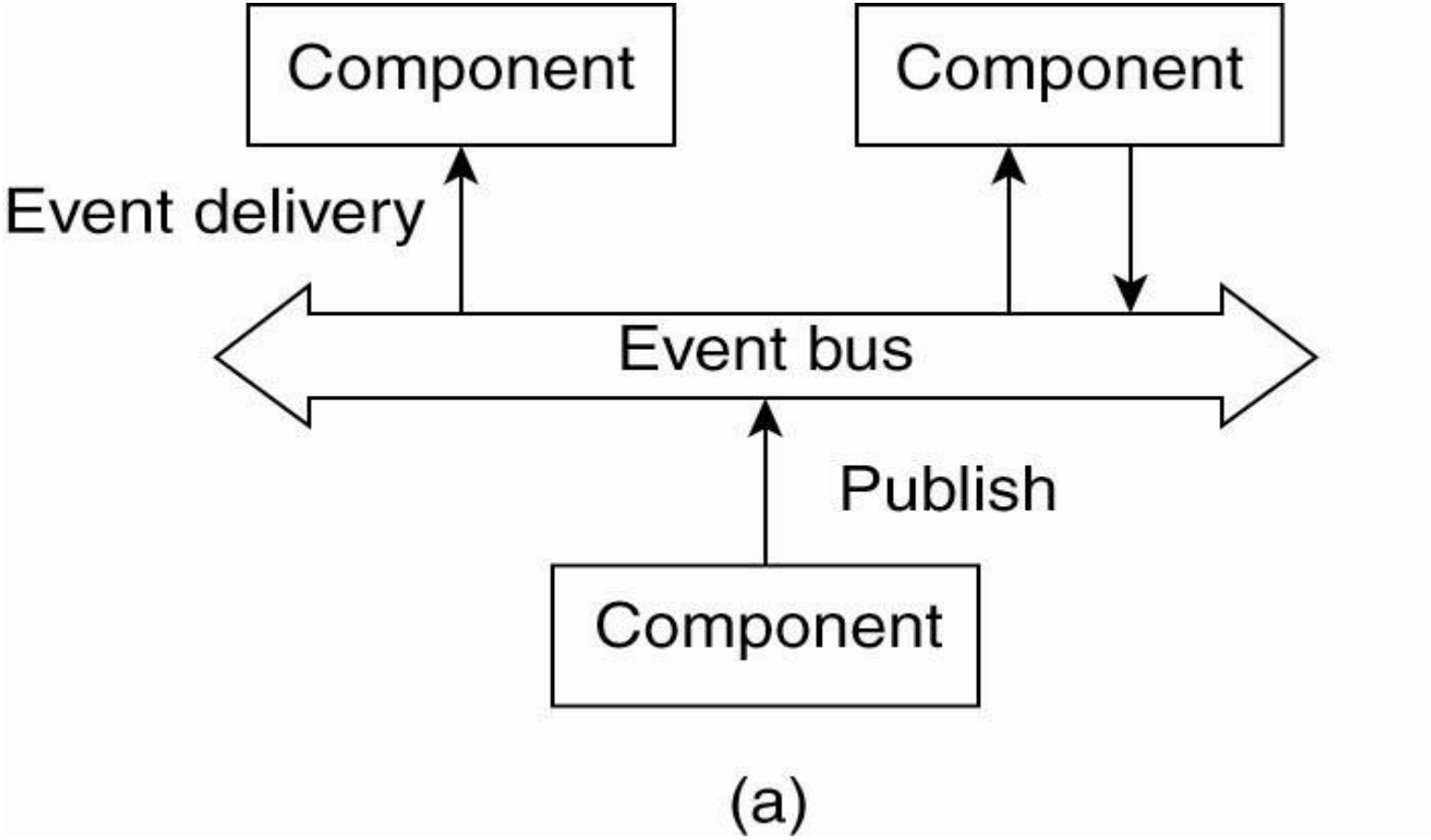
See Figure 2-1 in Tanenbaum and Van Steen book

The object-based architectural style



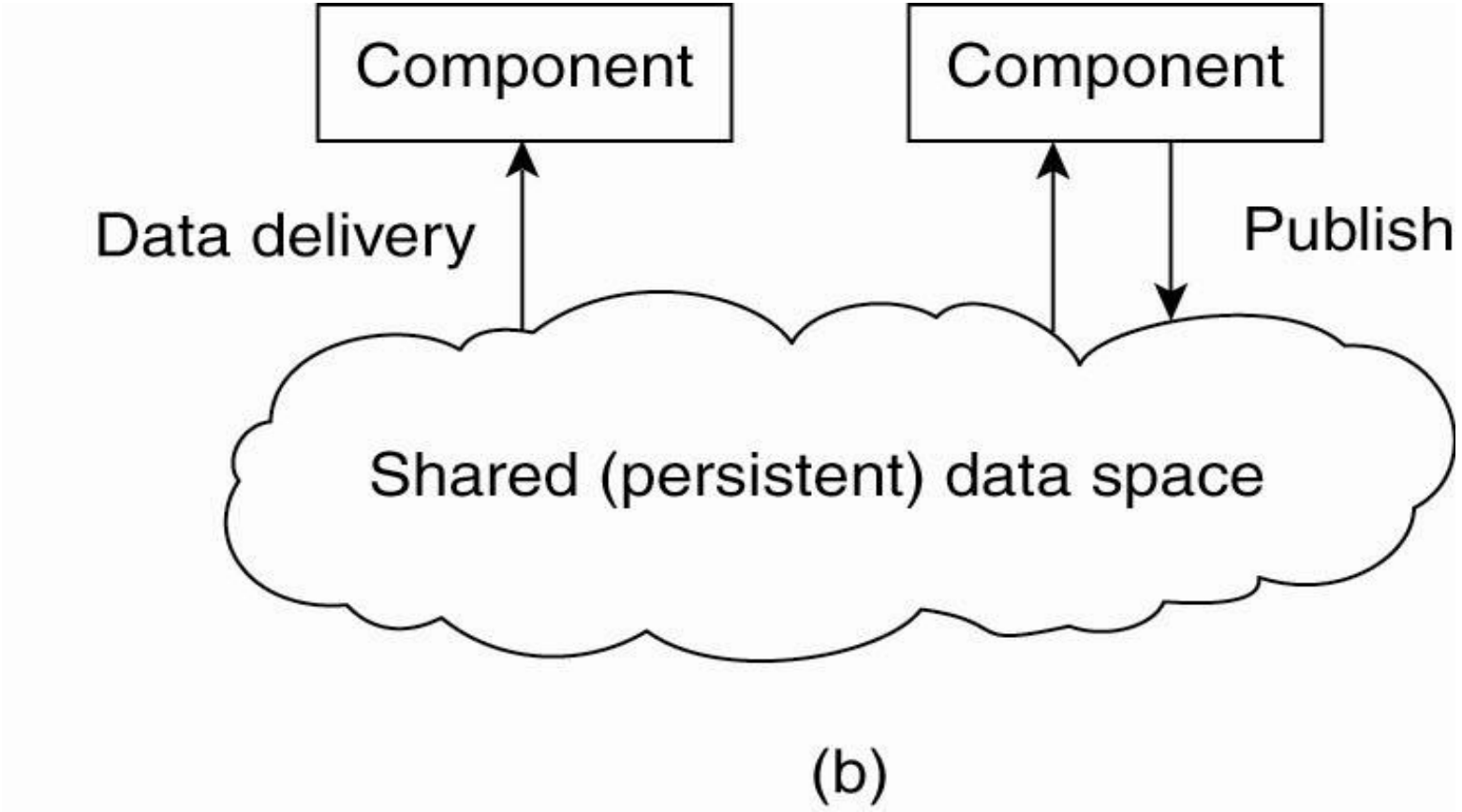
See Figure 2-1 in Tanenbaum and Van Steen book

The Event-Based architectural style



- See Figure 2-2 in Tanenbaum and Van Steen book

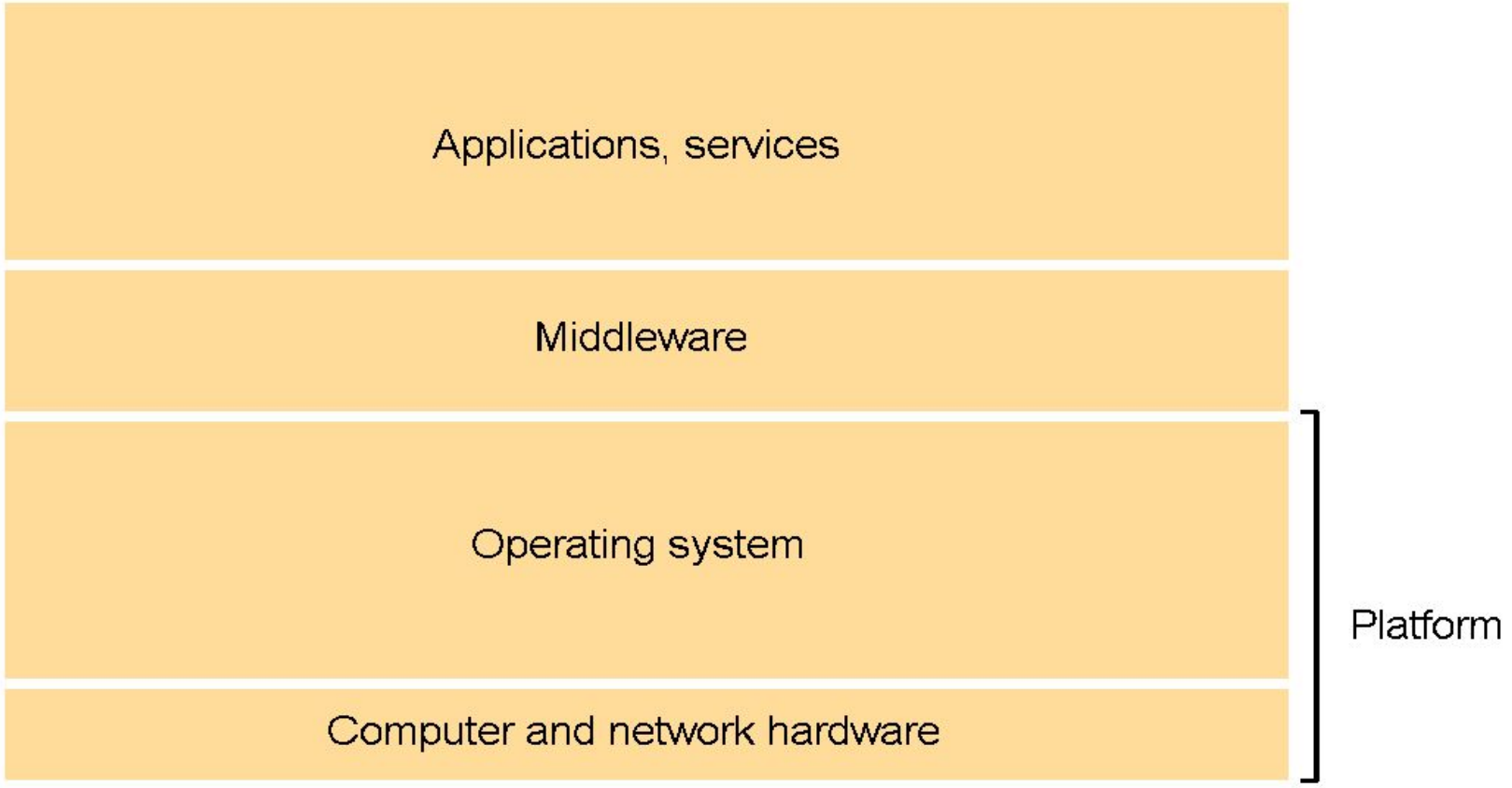
Shared-Data Space Architectural Style



- See Figure 2-2 in Tanenbaum and Van Steen book

System Architectures of Distributed Systems

Figure 2.1
Software and hardware service layers in distributed systems

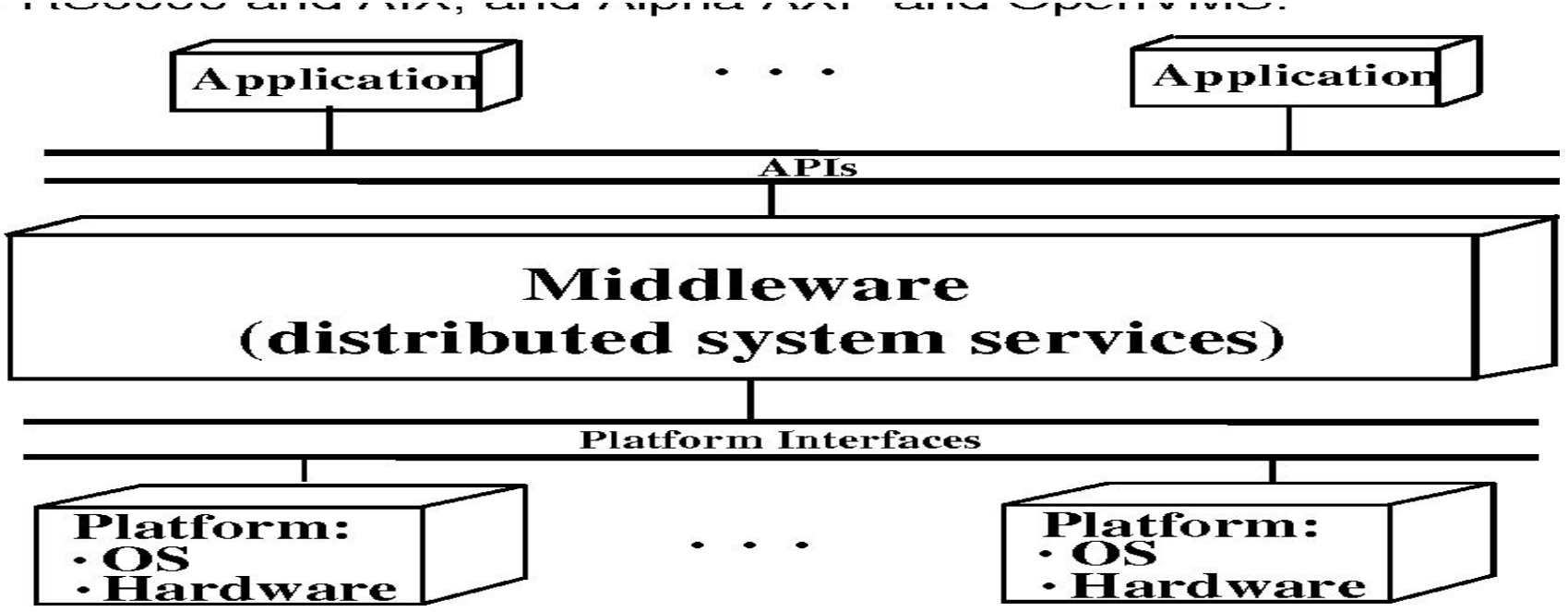


Middleware

- **Middleware**: a software layer that provides a programming abstraction to mask the heterogeneity of the underlying platforms (networks, languages, hardware, …)
  - E.g., CORBA, Java RMI

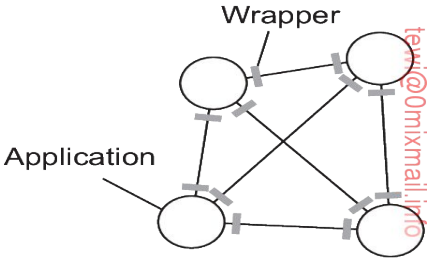
Middleware organization

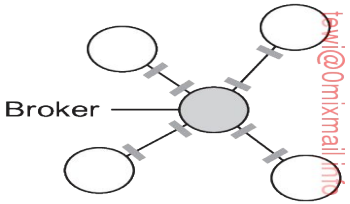
- In the previous section we discussed a number of architectural styles that are often used as general guidelines to build and organize distributed systems. Let us now zoom into the actual organization of middleware, that is, independent of the overall organization of a distributed system or application



Wrappers

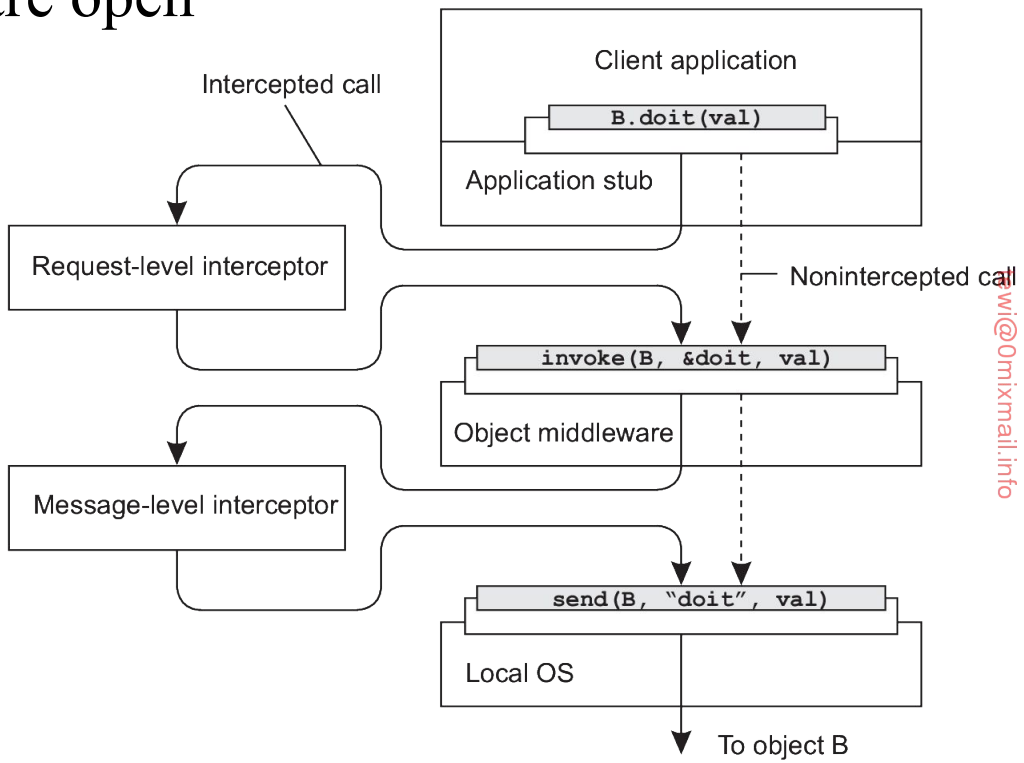
- A **wrapper** or **adapter** is a special component that offers an interface acceptable to a client application, of which the functions are transformed into those available at the component. In essence, it solves the problem of incompatible interfaces
- Wrappers have always played an important role in extending systems with existing components. Extensibility, which is crucial for achieving openness, used to be addressed by adding wrappers as needed
- need to develop $N \quad (N - 1) = \mathrm{O}\,(N^2)$ wrappers. For broker 2N connectors required





Interceptors

- Conceptually, an **interceptor** is nothing but a software construct that will break the usual flow of control and allow other (application specific) code to be executed. Interceptors are a primary means for adapting middleware to the specific needs of an application. As such, they play an important role in making middleware open



System architecture

- take a look at how many distributed systems are actually organized by considering where software components are placed
- Deciding on software components, their interaction, and their placement leads to an instance of a software architecture, also known as a **system architecture**
- We will discuss centralized and decentralized organizations, as well as various hybrid forms.

Centralized organizations

- It uses client/server architecture where one or more client nodes are directly connected to a central server.This is the mostly used type of system in many organizations where a client sends a request to a company server and receives the response.

Characteristics of centralized system

- **Presence of global clock :**all client nodes are sync up with the global clock (the clock of central node).
- **One single central unit :** one single central unit which serves/coordinates all the other nodes in the system.
- **Dependent failure of components:** Central node failure causes the entire system failure .If server is down ,the entire system is failed .

Advantages of centralized system

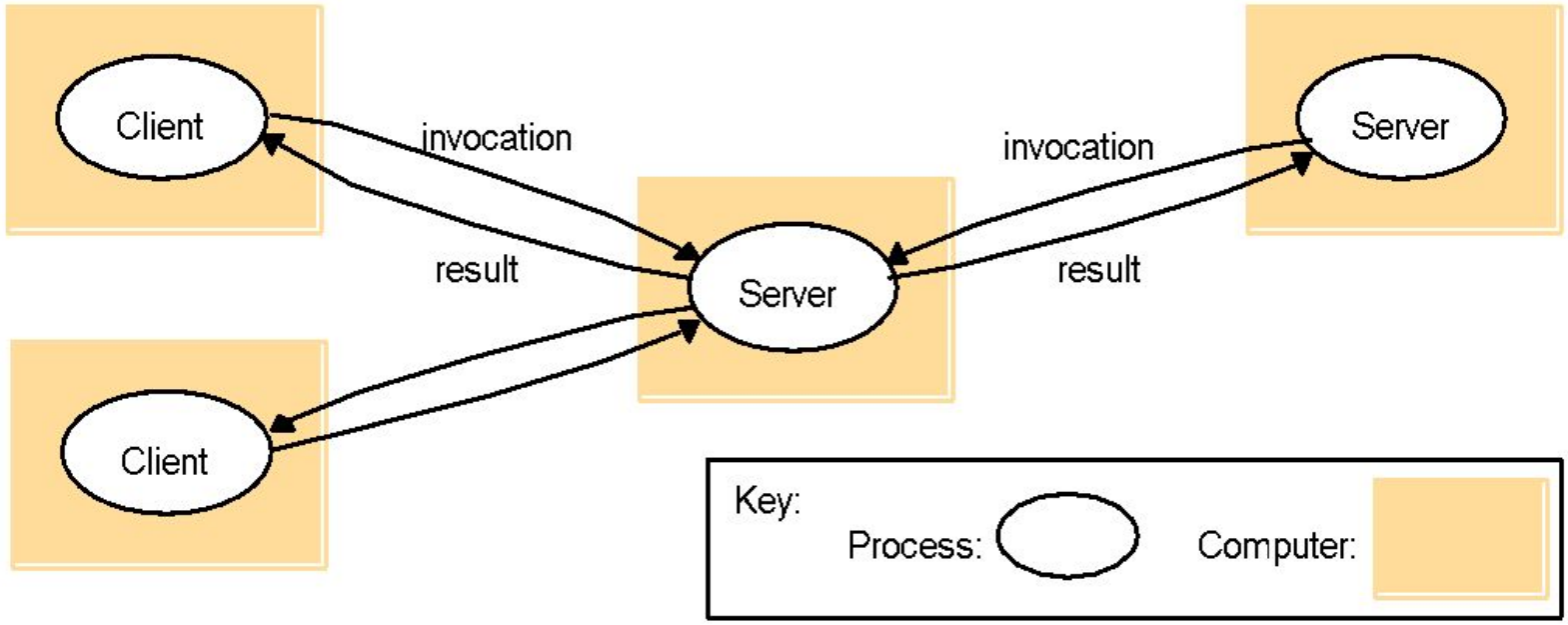
- easy to physical secure
- Smooth and elegant personal experience
- More cost efficient
- Quick updates are possible.

Disadvantages of centralized system

- Highly dependent on the network connectivity
- Less possibility of data backup.
- Difficult server maintenance
- Less reliable

Client-Server Model

Figure 2.2
Clients invoke individual servers



Client-Server Model

- Characteristics of a server:
  - Passive (slave)
  - Waits for requests
  - Upon receipts of requests, it processes them and sends replies
  - Can be stateless (does not keep any information between requests) or stateful (remembers information between requests)
- Characteristics of a client:
  - Active (master)
  - Sends requests
  - Waits for and receives server replies

Decentralized organization

- Multitiered client-server architectures are a direct consequence of dividing distributed applications into a user interface, processing components, and data-management components
- The characteristic feature of distribution is that it is achieved by placing logically different components on different machines.

Characteristics of decentralized system

- Lack of global clock
- Multiple central units

Advantages 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

Decentralized organization

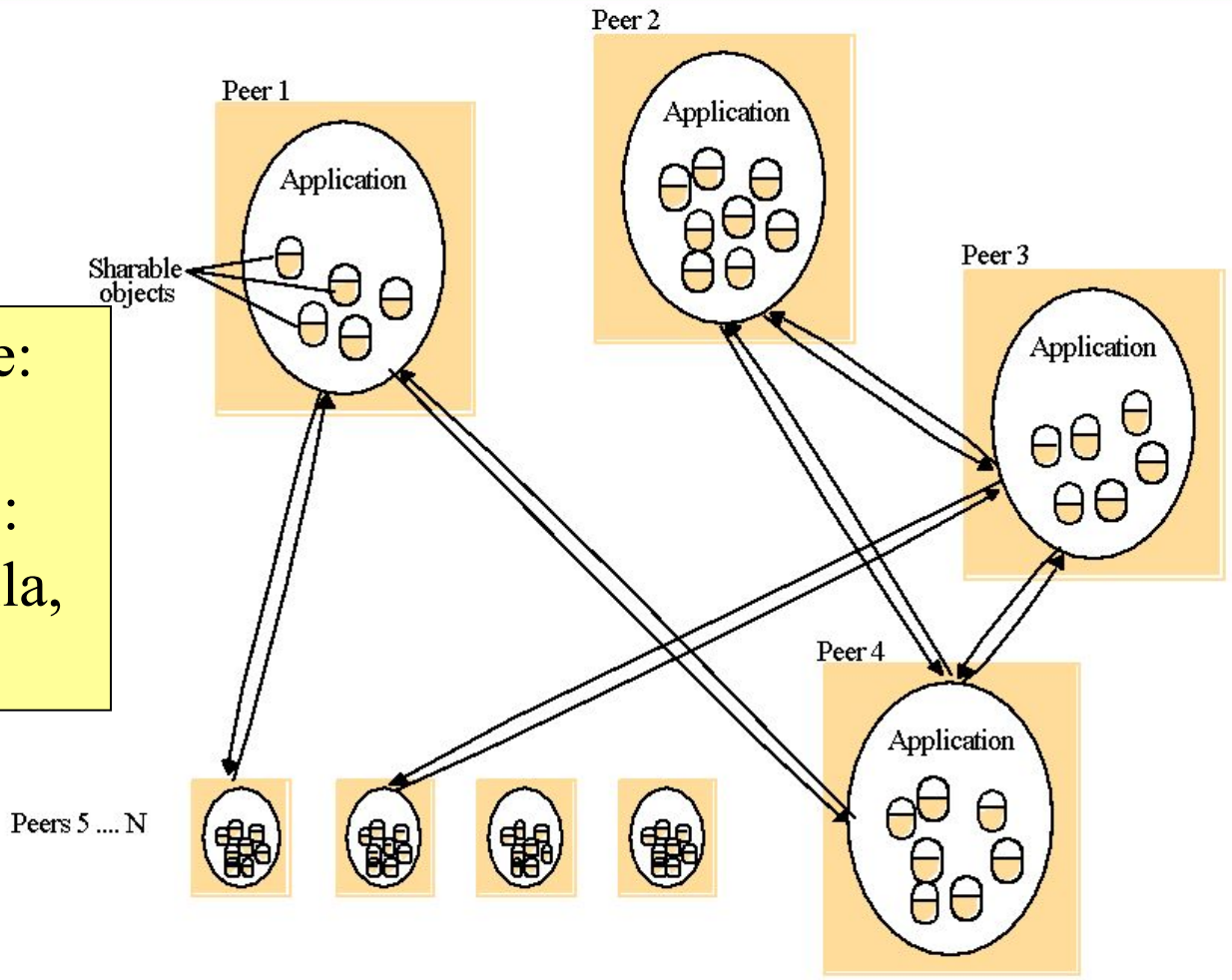
- More autonomy and control over resources – As each node controls its own behavior, it has better autonomy leading to more control over resources

Disadvantages of Decentralized System

- Difficult to achieve global big tasks – No chain of command to command others to perform certain tasks
- 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

Peer-to-Peer (P2P)

Figure 2.3
A distributed application based on peer processes



Earlier Example:
- Usenet News

Other examples:
- Napster, Gnutella, Freenet, …

Client-Server vs P2P

- Client-Server
  - Widely Used
  - Functional Specialisation
  - Asymmetrical
  - Tends to be centralised
  - Tends to scale poorly
- P2P
  - Symmetrical, computers have same “rights”
  - Truly Distributed
  - Share / exploit resources with a large number of participants
  - Resource discovery is a challenge

Variations on a theme…

- Sometimes we need to consider:
  - The use of multiple servers to increase performance and resilience
  - The use of mobile code
  - Users' need for low-cost computers
  - Requirements to add and remove mobile devices