Dağıtık Sistemler

2. Ders
Mimari Yapılar
(ARCHITECTURES)

Genel Bakış (I)

Dağıtık bir sistemin mimari modeli, parçalarının yerleşimi ve aralarındaki bağlantı ile ilgilenir.

Örnek:

- İstemci-Sunucu (CS): dağıtık bir sistem için en yaygın dizayn
- peer process model (akran-eş görev modeli)

CS şu şekilde değiştirilebilir:

- Birlikte çalışan sunucularda veri ve kopyaların bölünmesi
- Verinin proxy sunucular veya istemcilerce ön belleğe alınması
- Mobil kod ve mobile temsilcilerin kullanımı
- Mobil aygıtların eklenmesi veya çıkarılması

Mimari Model

- Var olan ve ileride olabilecek talepleri karşılamak
- Sistemin emniyetli, yönetilebilir, uyumlu, ve fiyat dengeli hale getirilmesi
- Her bir bileşenin işlevlerini basitleştirip şeffaflaştırır.
- Bir bilgisayar ağı üzerinde bileşenlerin yerleşimi – veri ve iş yükü dağılımı için kalıp tanımlama
- Bileşenler arasındaki bağlantılar – yani, işlevsel roller ve aralarındaki iletişim kalıpları.
- **Öz olarak**, mimari model şunlarla formüle edilebilir:
  - Bileşenler neler
  - Bileşenler birbirine nasıl bağlanmış
  - Bileşenler arası al-ver yapılan veriler
  - Bu elemanlar birlikte nasıl sisteme dönüşüyor

Mimari Model(devam)

her bileşenin işlevini basitleştirip şeffaflaştırır:

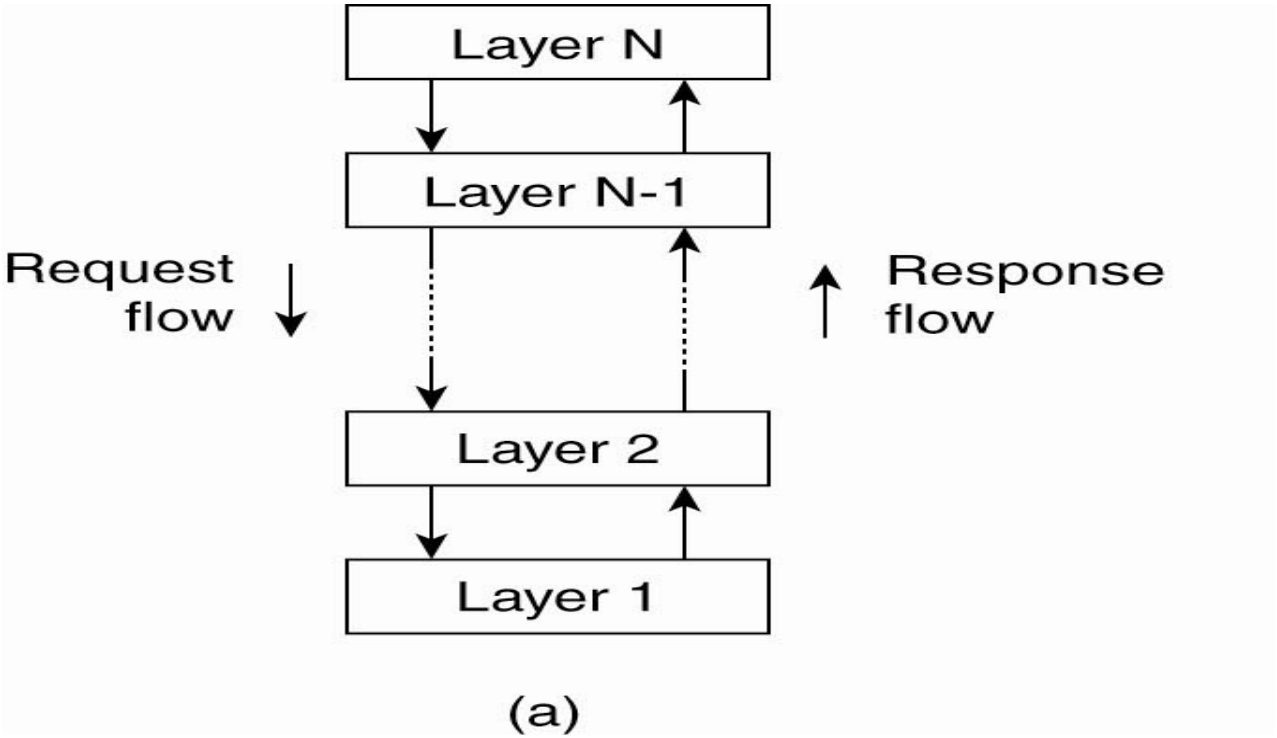
- İşlem-görev (process) ve nesne kavramları üzerinde bina edilmiştir
- İşlemlerin şu şekilde sınıflandırılmasıyla ilk basitleştirme elde edilir:
  - Server processes (Sunucu görevler)
  - Client processes (İstemci görevler)
  - Peer processes (Eş-Akran görevler)
    - Bir görevi yerine getirmek için simetrik şekilde işbirliği ve iletişim

Mimari (Architectural) Sitiller

Dağıtık Sistemler için önemli mimari sitiller şu biçimde listelenebilir:

- Layered architectures (Katmanlı mimariler)
- Object-based architectures (nesne-tabanlı mimariler)
- Data-centered architectures (Veri-merkezli mimariler)
- Event-based architectures (olay-tabanlı mimariler)

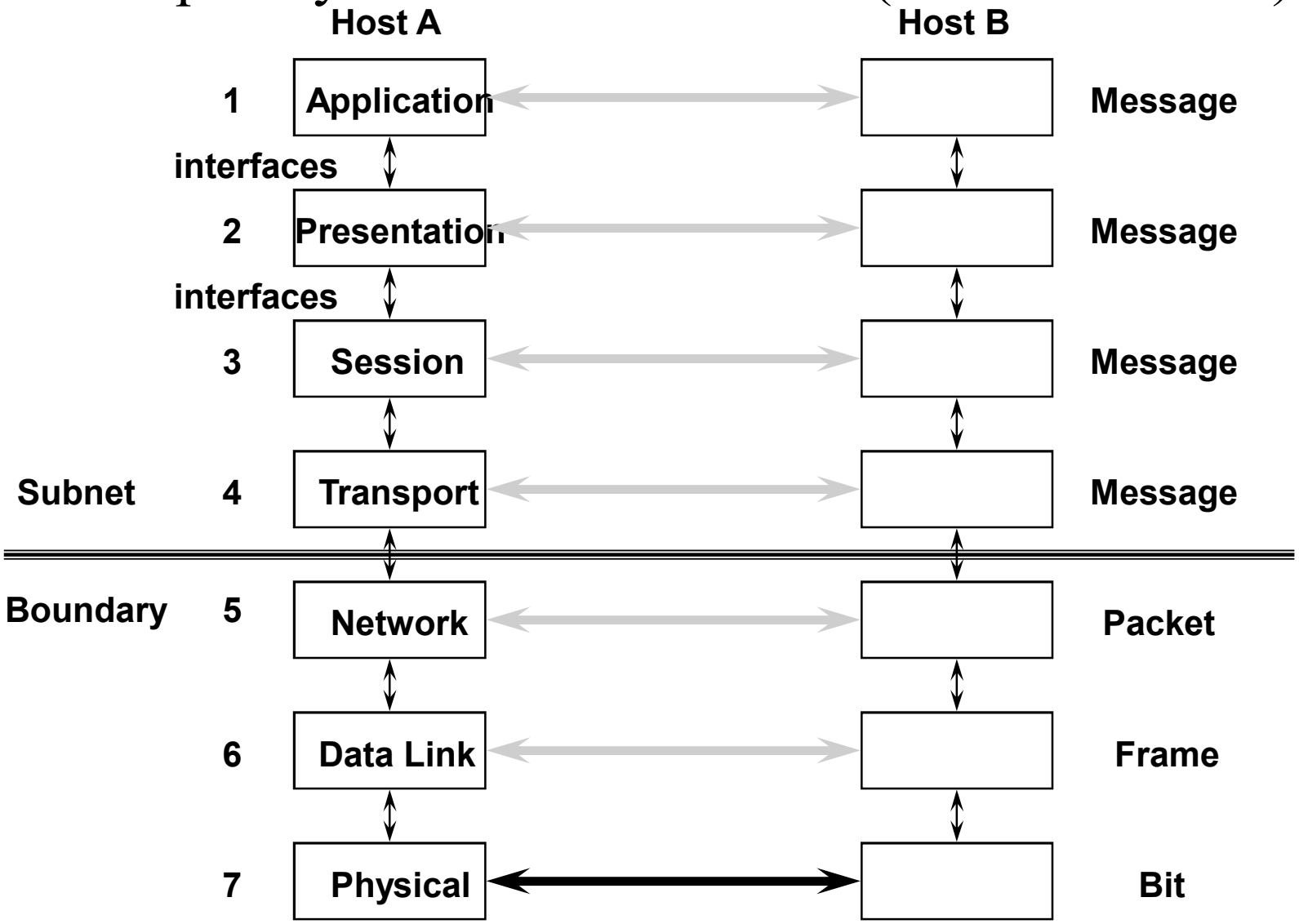
Layered Architecture



Sistemlerin katman ve servisler şeklinde dizayn edilmesiyle karmaşıklığın azaltılması

- Katman: Birbiriyle bağlantılı işlevsel bileşenler grubu
- Servis: sonraki (üst) katmana sunulan işlevsellik

The International Standards Organization Reference Model of Open System Interconnection (ISO OSI model)



ISO OSI Model (The 7 Layers)

Physical Layer

specifies physical signals (electrical, optical), cabling/wiring.

Data Link Layer

frame boundaries, error detection (noise), flow control

Network Layer

how packets are sent from sources to destinations: routing, congestion control, inter-networking.

Transport Layer

break messages into packets, multiplex/demultiplex messages,end-to-end reliability.

Session Layer

provides users interface: directory, access rights, synchronization.

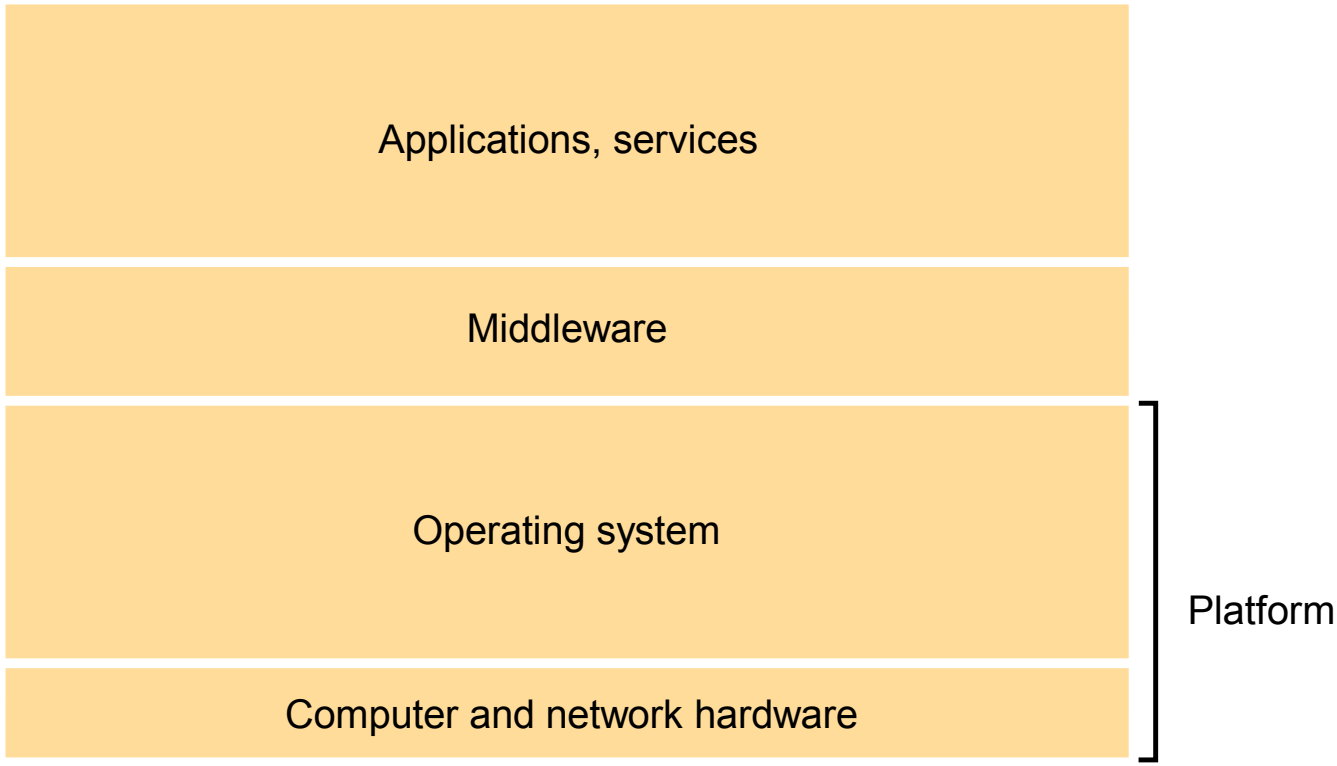
Presentation Layer

data encryption, data compression, code conversion.

Application Layer

virtual terminal, virtual file transfer

Dağıtık sistemlerde yazılım/donanım servis katmanları



Platform

- En alttaki donanım ve yazılım katmanlarına, genel olarak dağıtık sistemler/uygulamalar için **platform** denir.
- Bu alt-seviye katmanlar üstlerindeki katmana **servis** sunar. Ya da üst katman, alt katmanının sağladığı servisleri **kullanarak** işlevini yerine getirir.
- Bu katmanlar her bir bilgisayarda birbirinden bağımsız olarak hazırlanır.
- Bazı örnekler:
    - Intel x86/Windows
    - Intel x86/Linux
    - Intel x86/Solaris
    - SPARC/SunOS
    - PowePC/MacOS

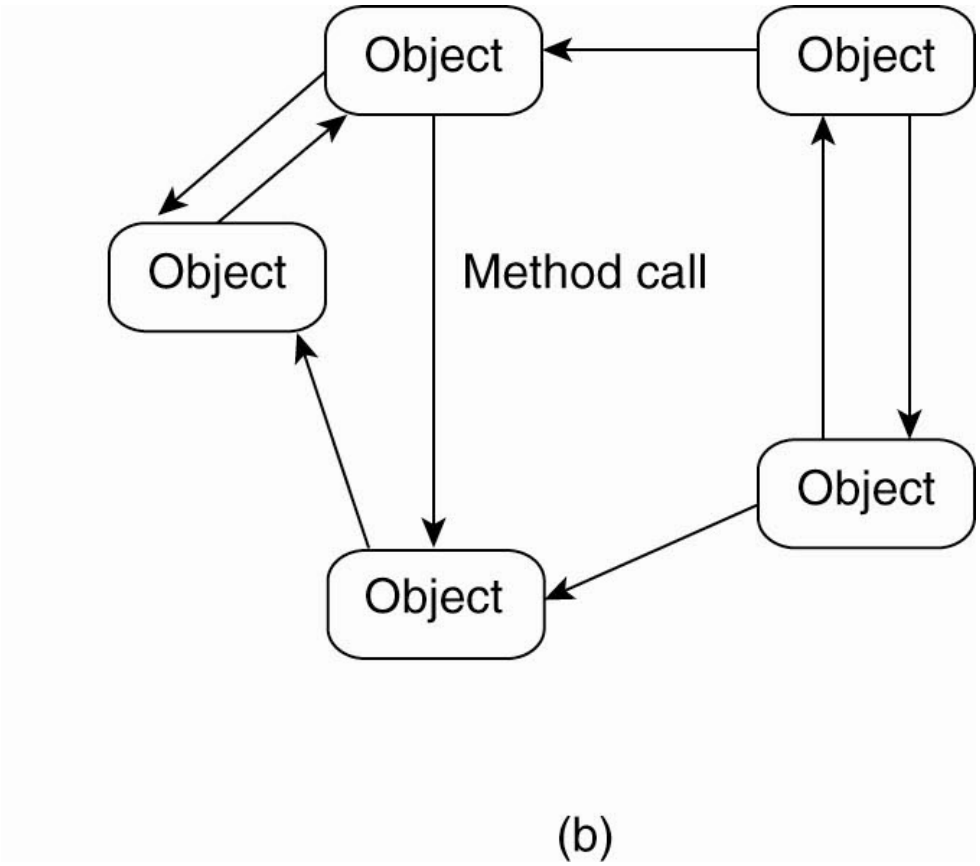
Middleware (Ara Katman)

- Amacı, dağıtık sistemlerde var olan heterojenliği maskelemek, ve, uygulama geliştiricilerine, elverişli bir programlama modeli sağlamak olan bir yazılım katmanı.
- Dağıtık uygulamalara, iletişim ve kaynak paylaşımı desteği sağlar.
- Birbirleriyle etkileşimde bulunan bir bilgisayarlar kümesindeki, işlemler ve nesnelerle gösterilir
  - Ana örnekler:
    - Sun RPC (Remote Procedure Calls) (Uzaktan Yordam Çağrıları)
    - Group communication system (Grup iletişim sistemi) (Isis)
    - OMG CORBA (Genel talep aracısı mimarisi)
    - Microsoft D-COM (Distributed Components Object Model) (Dağıtık Bileşenler Nesne Modeli)
    - Sun Java RMI
- Modern Middleware:
    - IBM WebSphere, Microsoft .NET, Sun J2EE

Applications, Services (Uygulamalar ve servisler)

- Dağıtık uygulamalar ve hizmetlerin olduğu katman. Orta katmanın sağladığı olanakları kullanır.
- Dağıtık servis: 1/1+ sunucu görev(process) tarafından sunulur, Eğer servis 1+ sunucu tarafından sağlanıyorsa, sunucular etkileşimde
- Örnek: herhangi bir istemciye şu anki zamanı iletmek için bir ağ zamanı servisi (**network time service**)

Nesne-tabanlı mimari



The object-based architectural style.

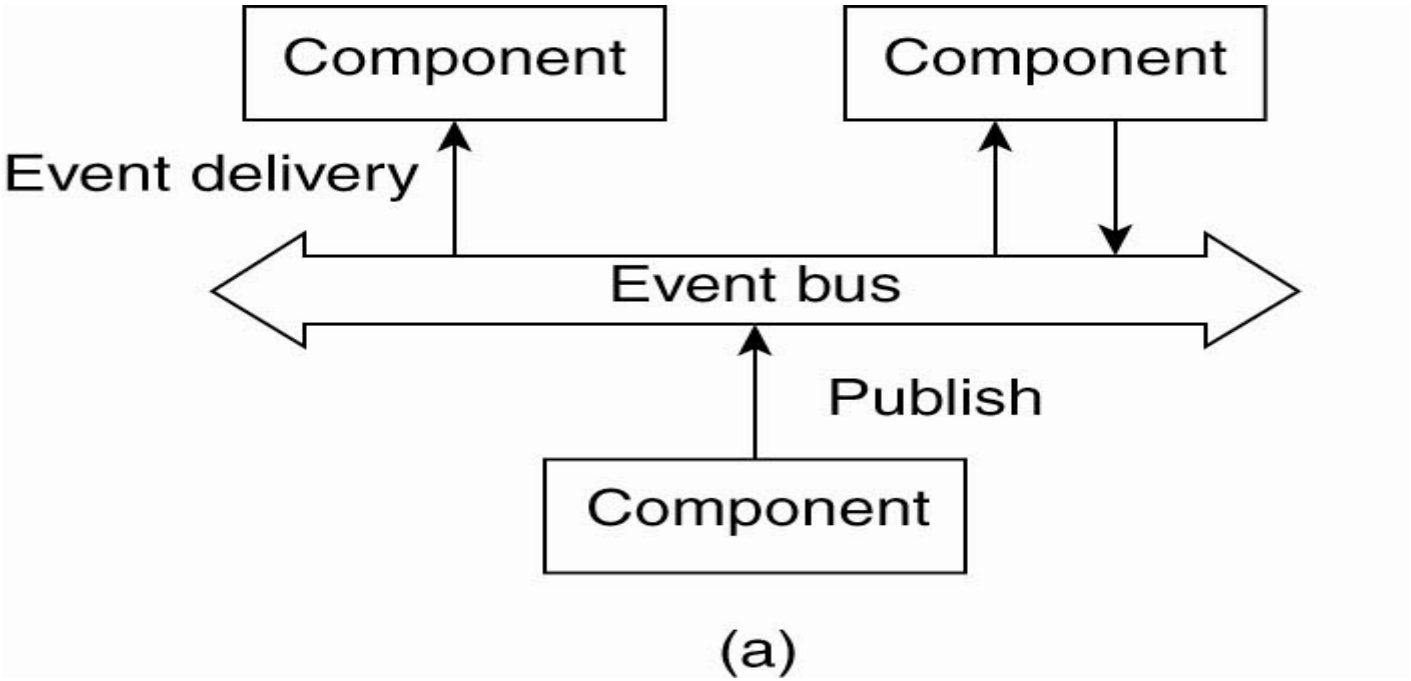
Nesne-tabanlı mimari

Bu modelde

- Nesne(object): Bir bileşene (component) karşılık gelir.
- **Component:** a modular unit with well-defined required and provided interfaces that is replaceable within its environment
- Components are connected through a (remote) call mechanism
- Bu mimari sitil client-server sistem ile eşleşiyor

- Genel olarak, **katmanlı ve nesne-tabanlı mimariler**, geniş hacimli yazılım sistemleri için en önemli mimari sitiller.

Olay-tabanlı mimari



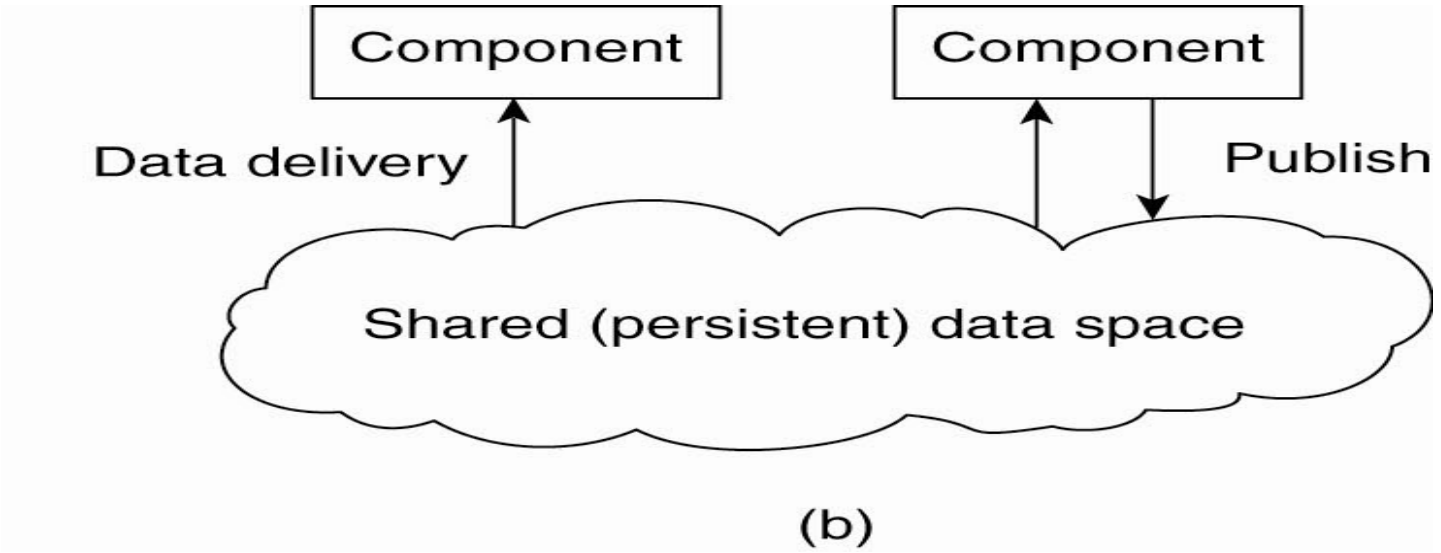
Processler olayların yayılımı (probagation of events) ile haberleşir

Olaylar, veri de taşıyabilir (opsiyonel)

Event propagation: publish/subscribe system ile eşlenik

Processes publish events after middleware ensures that only those processes that subscribed to these events will receive them

Veri Merkezli (data-centered) Mimari



Idea: processes communicate through a common (passive/active) repository

Example: a shared distributed file system, in which all communication takes place through files.

web-based distributed system: processes communicate through use of shared web-based data services

SQL-like interface to the shared repository

Sistem Mimarisi (Yazılım Mimarisi):

Consider where s/w components are placed

S/W architecture (System architecture)

- s/w components
- Their interaction
- Their placement

1. Centralized (Merkezi) Architectures
2. Decentralized (Merkezi Olmayan) architecture
3. Hybrid (Karışık) Architectures

The Client-Server Model

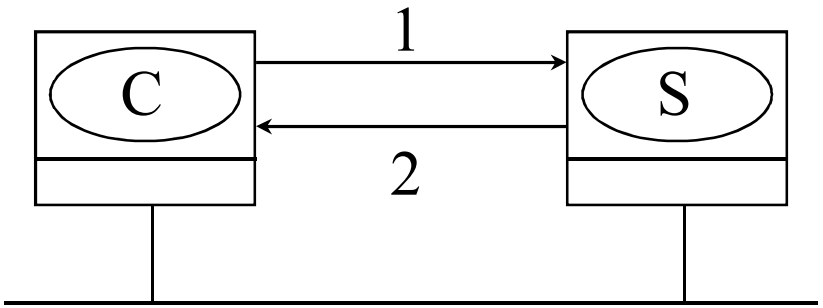
- A **server** is a program that provides a service.
  - It makes some resource available to other programs running somewhere on the network.
  - The **resource** could be anything: a database, a file system, a printer, a modem, a screen, or a scanner.
  - The server program runs on the machine which resource is attached to, and waits passively until the services are required.
  - Servers are often started up at boot-time, via commands in a start-up script and Servers usually spent most of their time asleep, waiting for work.
- A **client** is a program that uses a resource.
  - It may be running on the machine to which the resource is attached, or on a different machine.
  - A client actively makes a connection request across the network to the server it requires.

The Client-Server Model (continue)

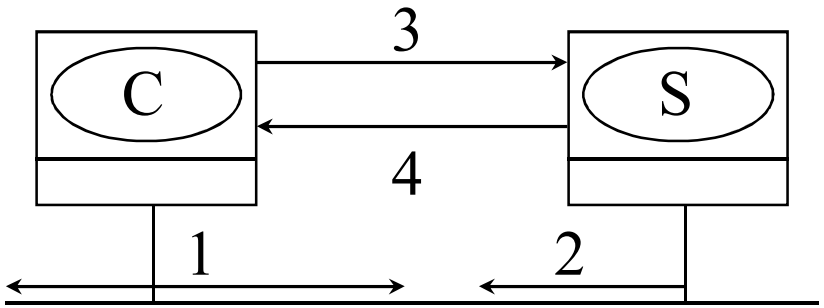
- The terms **client and server** are sometimes used to refer to **a machine**.
- We may point out a computer on our network which has a large disk attached and say “That is our **file server**, and these workstations here are its clients”.
- We may refer to a machine with a printer attached as a “**print server**”.
- The term client and server refer to the **relationship between individual programs**, not to machines.
- In general, machines **cannot** simply be labeled as clients and servers.
- One machine might be a server for a piece of the file system, but a client of a remote print server.
- Example, a program which is a server for a database might itself contact a time server to obtain an accurate timestamp to attach to a new record in the database
- In UNIX world, the term **daemon** is sometimes used to refer to a server.
- It means that the server is “permanently available”, which implies that it is started at boot-time.
- Daemon usually provide a “system-related” service, such as: rlogind, telnetd, ftpd, httpd, nfsd,… etc.

Client-Server Addressing

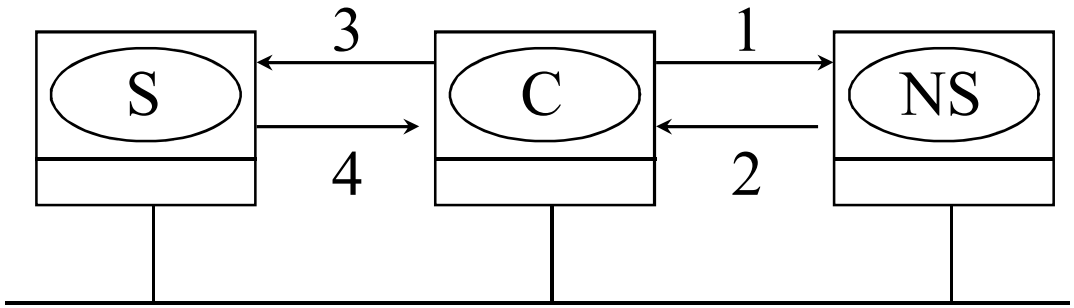
- Machine.process / Machine.local-id addressing
- Process addressing with broadcasting
- Address lookup via a name-server.



Hardwire machine.number into client code
1: Request to 234.0
2: Reply to 199.0

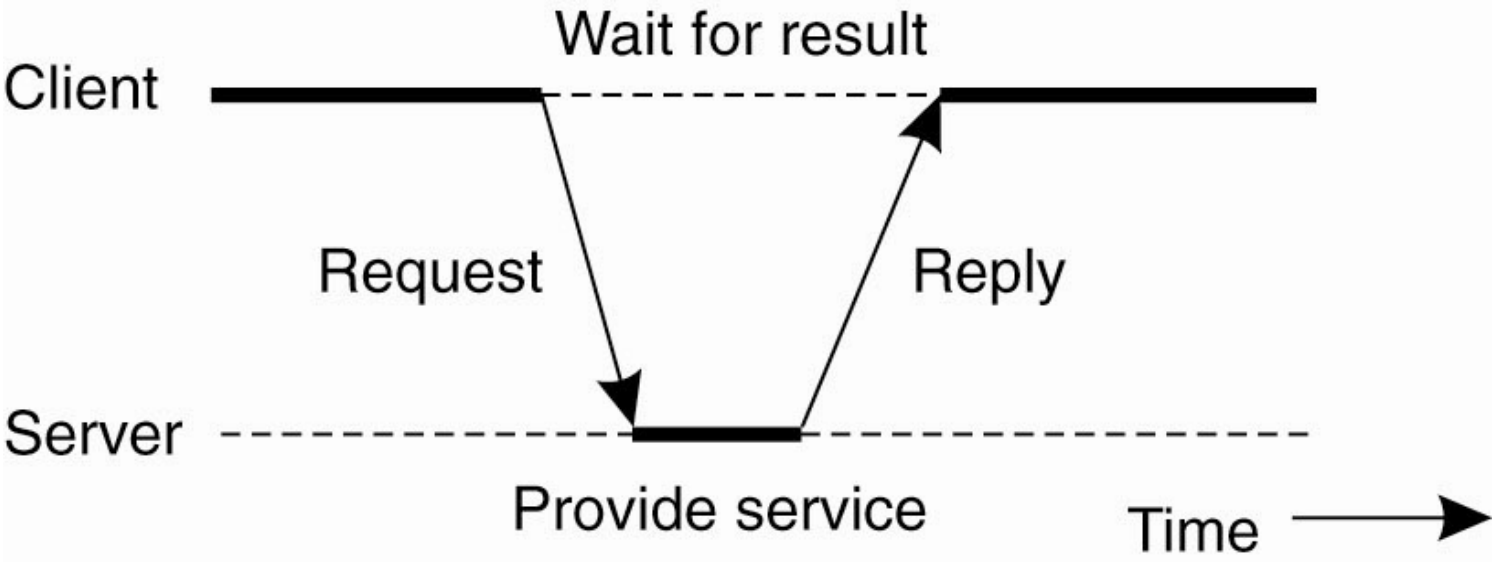


Processes pick a random #, locate them by broadcasting
1: Broadcast
2: Here I am
3: Request
4: Reply



Put ASCII server names in clients, look them up at run time
1: Look-up
2: NS reply
3: Request
4: Reply

Interaction between a client and a server



Client-server interaction aka request-reply behavior

Client rolündeki process server processe request (istek) gönderir ve beklemeye başlar. Server'ın cevabı (reply) gelinceye kadar beklemede kalır.

Örnekler:

– **Webserver**. Client (Web browser) web sayfası için bir request gönderir. Sonra da, webserver istenen sayfa ile reply eder.

– **SQL serverdan**, client işlemler veri isteğinde bulunur, veya verileri değiştirecek bir istekte bulunur.

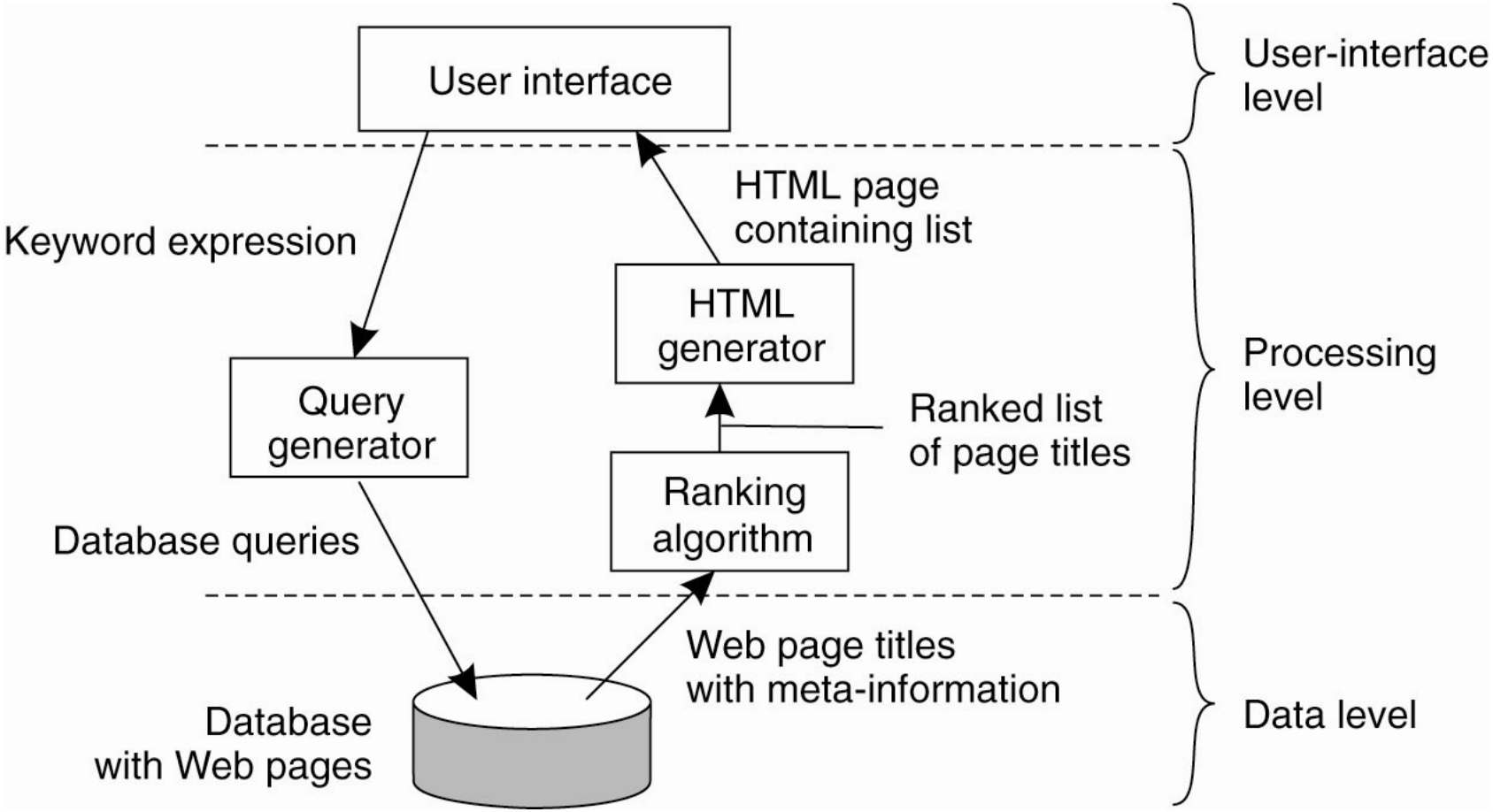
Sunucular istemci olabilirler. Örneğin, bir web sunucusu proxy olarak diğer sunuculara istek gönderebilir.

Uygulamayı Katmanlara Ayırma (Application Layering)

Recall previously mentioned layers of architectural style

- The user-interface level
  - Interface to users to interact with apps. Display mgmt.
- The processing level
  - Contains the apps
- The data level
  - Manage the actual data (a database)

Application Layering



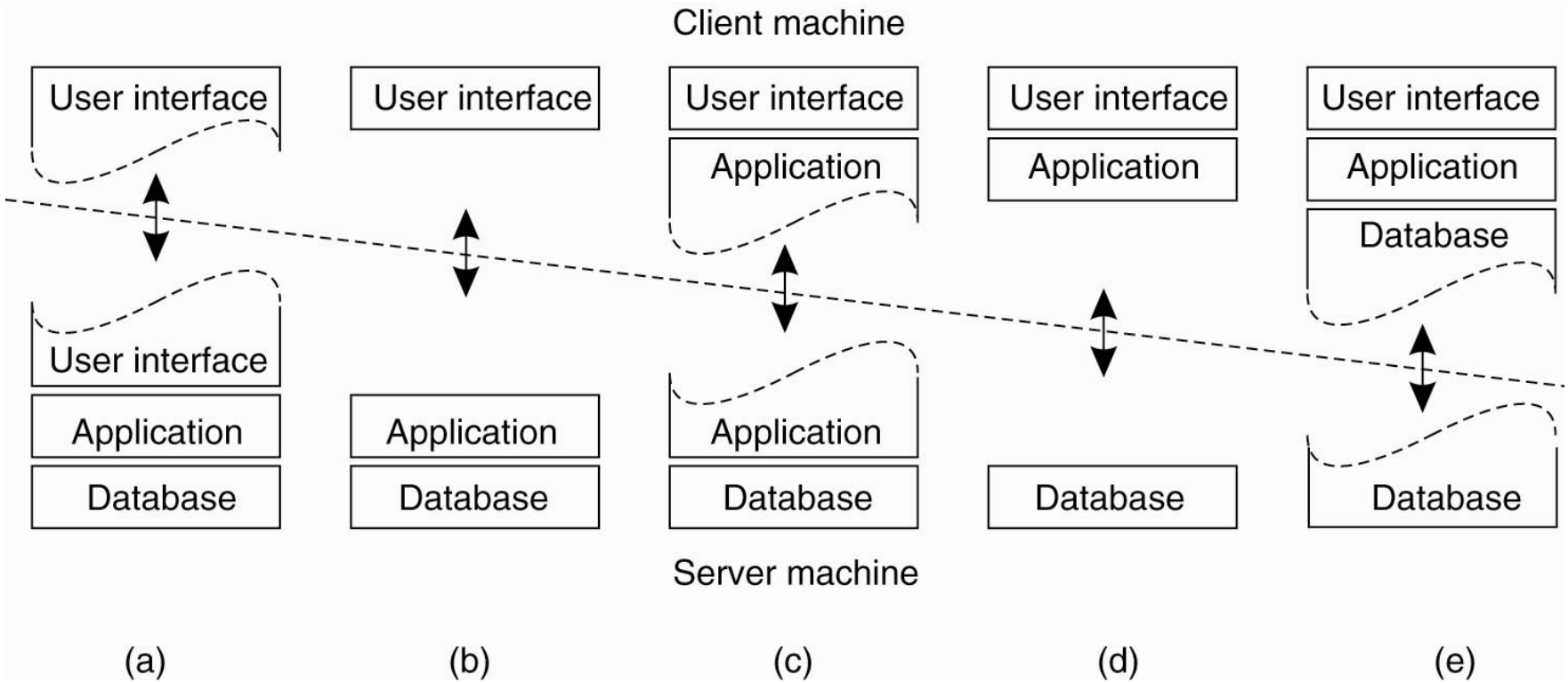
The simplified organization of an Internet search engine into three different layers.

Multitiered Architectures

The simplest organization is to have only two types of machines:

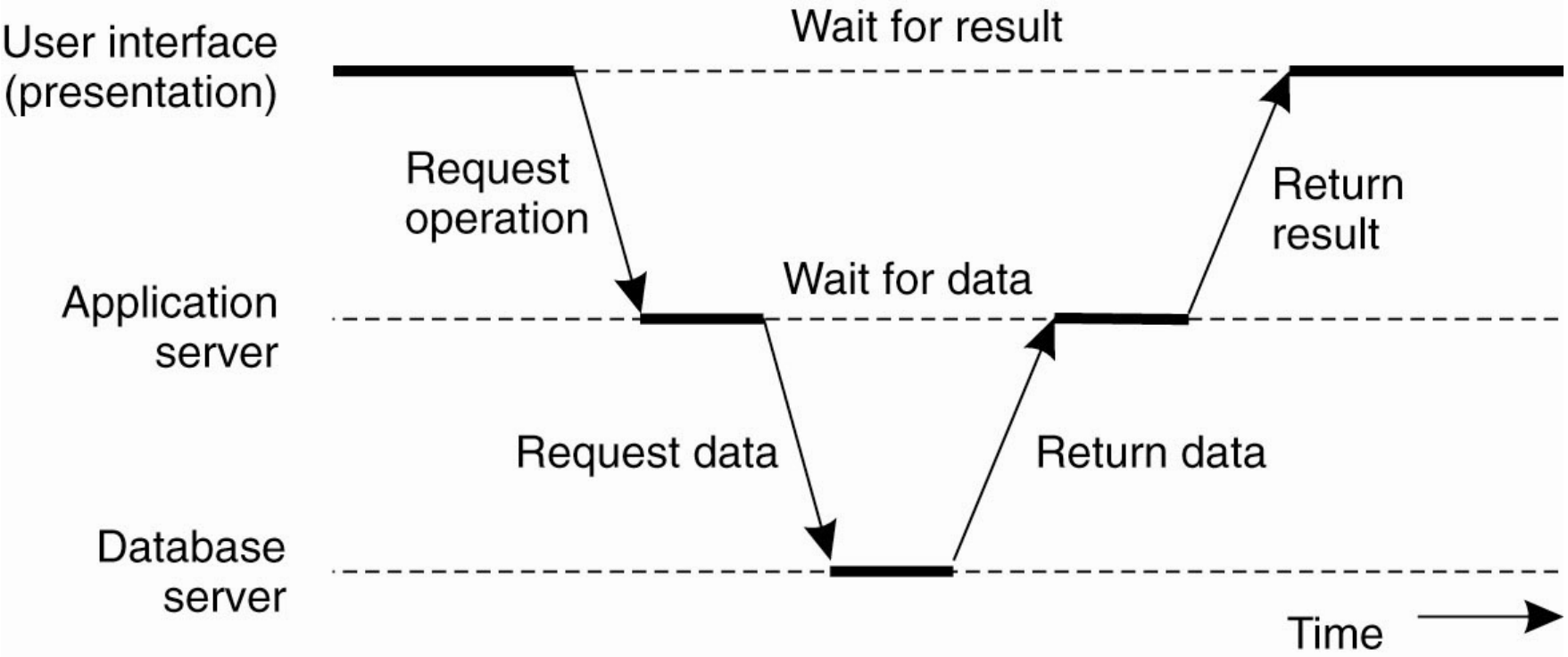
- A client machine containing only the programs implementing (part of) the user-interface level
- A server machine containing the rest,
  - the programs implementing the processing and data level

Multitiered Architectures (2)



Alternative client-server organizations (a)–(e).

Multitiered Architectures (3)



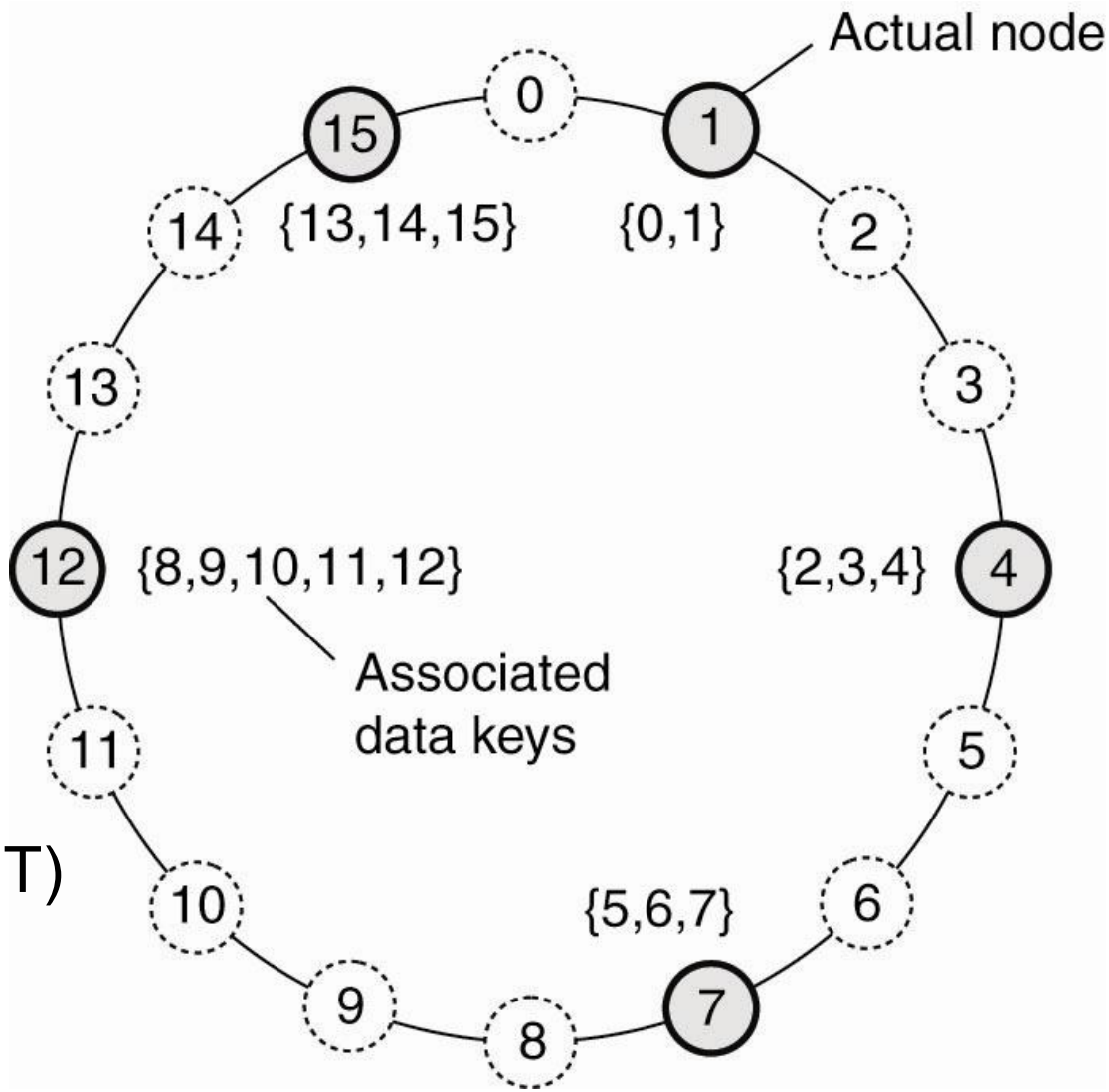
An example of a server acting as client.

Decentralized Architectures:

- Distributed processing (client-server way, multitiered): vertical distribution. Place logically diff components on diff machines
- Horizontal distribution: distribution of clients and servers.
  1. A client(server) may be physically split up into logically equivalent parts.
  2. Each part is operating on ist own share of complete data set (load balancing)
  3. Aka **peer-to-peer system**
- Tüm görevler benzer roller oynarlar. Dağıtık etkinlik veya hesaplamaları yürütmek için, client (veya server) processler, birlikte etkileşirler. Örn., müzik paylaşım sistemleri gibi, Distributed "**white board**" (Dağıtık "**beyaz tahta**") – farklı bilgisayarlarda bulunan kullanıcıların aynı resmi görüntülemeleri ve etkileşimli olarak üzerinde değişiklik yapmaları.

1. **Structured Peer-to-Peer Architectures**
2. **Unstructured Peer-to-Peer Architectures**

Structured Peer-to-Peer Architectures



Distribited Hash Table(DHT)

The mapping of data items onto nodes in Chord.

Unstructured Peer-to-Peer Architectures (1)

Actions by active thread (periodically repeated):

```
select a peer P from the current partial view;
if PUSH_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
} else {
    send trigger to P;
}
if PULL_MODE {
    receive P's buffer;
}
construct a new partial view from the current one and P's buffer;
increment the age of every entry in the new partial view;
```

(a)

The steps taken by the active thread.

Unstructured Peer-to-Peer Architectures (2)

Actions by passive thread:

```
receive buffer from any process Q;
if PULL_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
}
construct a new partial view from the current one and P's buffer;
increment the age of every entry in the new partial view;
```

(b)

The steps take by the passive thread

Superpeers

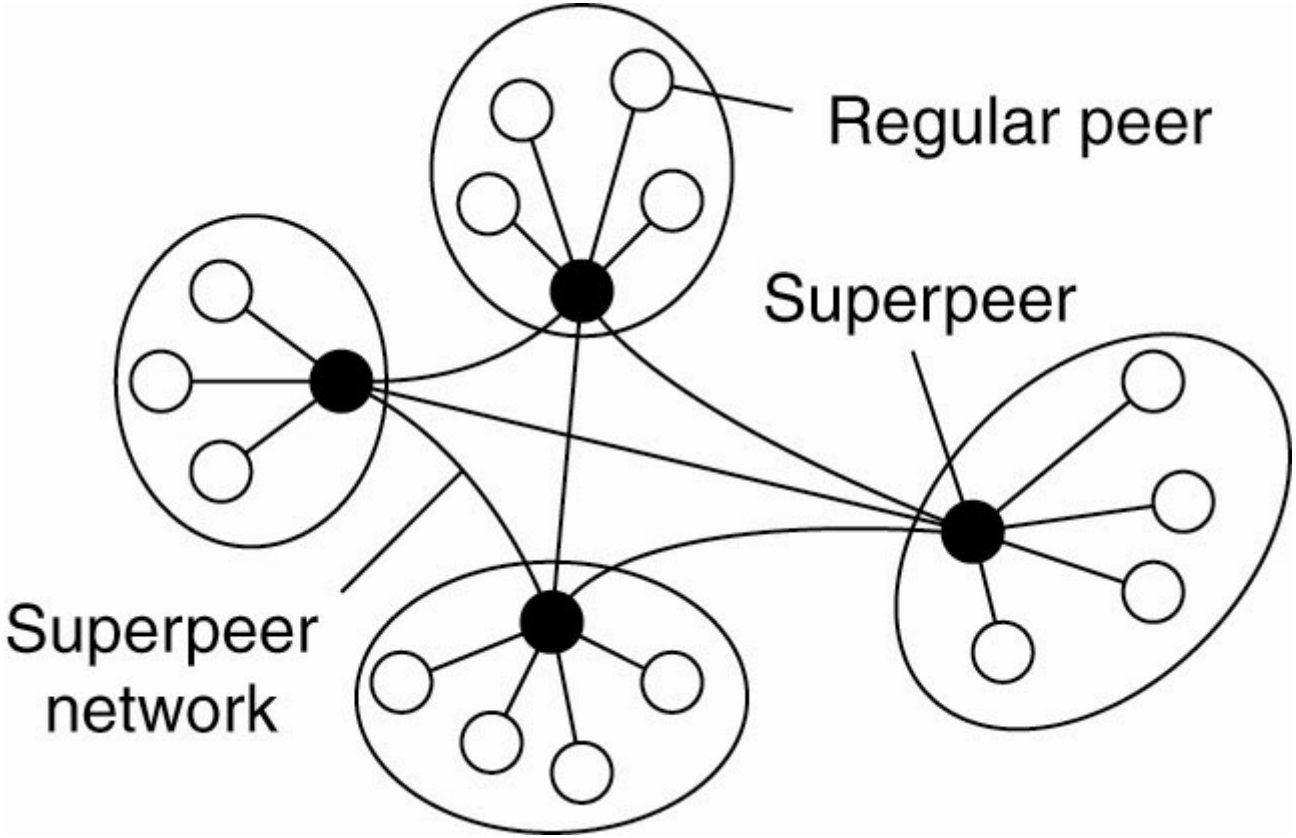


Figure 2-12. A hierarchical organization of nodes into a superpeer network.

Edge-Server Systems

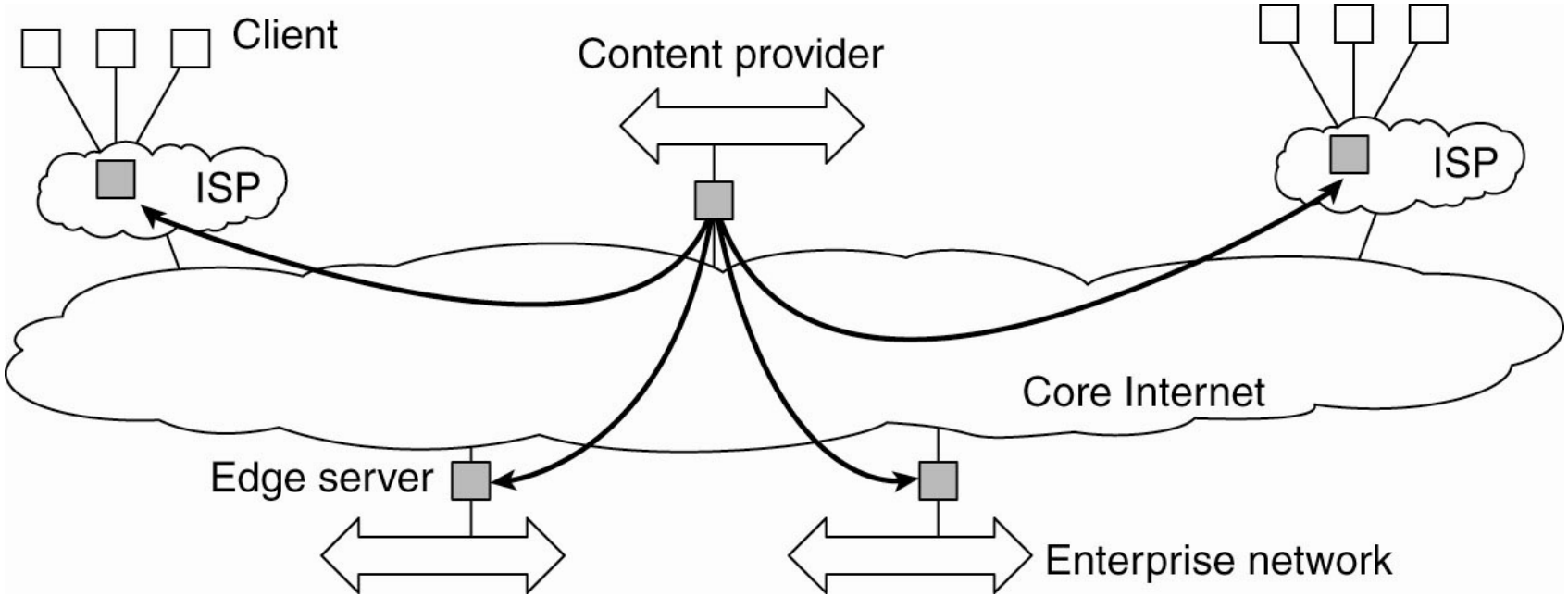


Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.