CS 432: Distributed Systems

Distributed Systems Architectures

Reading

- Tanenbaum (2nd Edition): 2.1, 2.2
- Coulouris (5th Edition): 2.3

Introduction

- Distributed systems:
  - are complex pieces of software
  - have components that are dispersed across multiple machines
- It is important to organize these systems to manage their complexity
- Organization of a distributed system
  - Hardware architecture (physical realization)
  - Software architecture: how software components are organized and interacting

System Models

- Physical models

  Types of computers and devices that constitute a system and their interconnectivity (sharing in local network, internet-scale, ubiquitous computing, cloud computing, IoT)

- Architecture Models

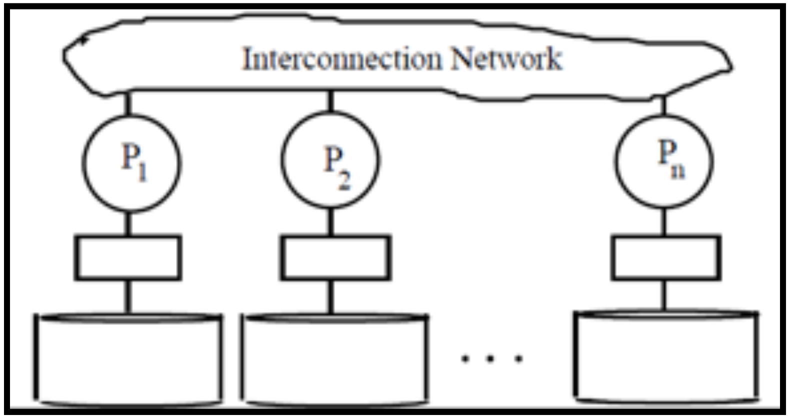
  The components of a distributed system and their interrelationships

- Fundamental Models
  - Interaction models
  - Failure models
  - Security models

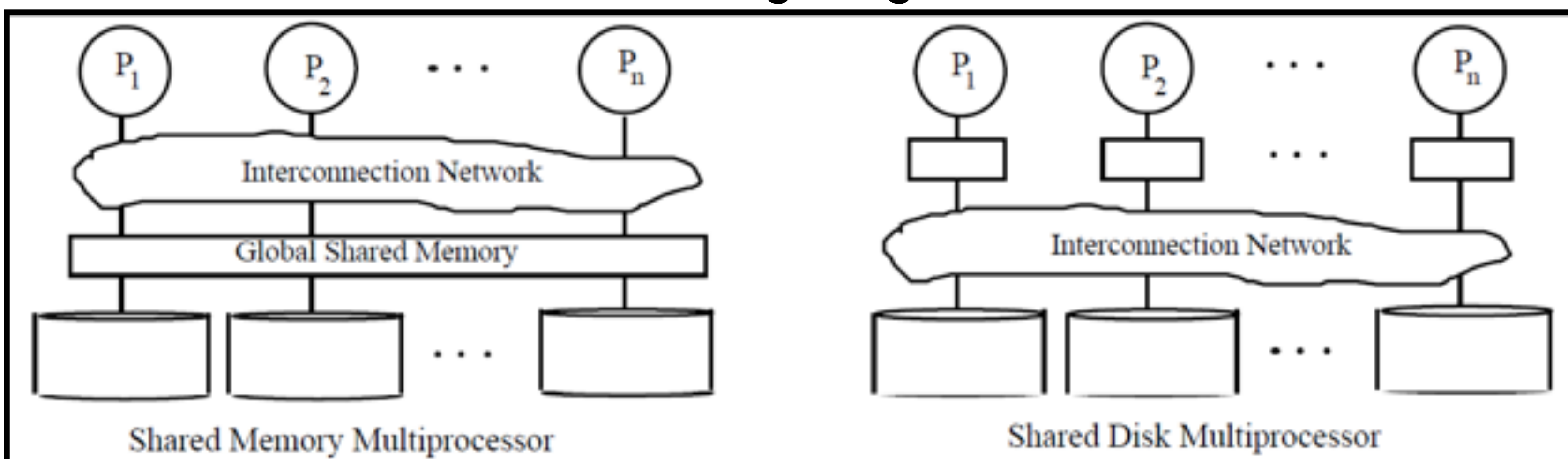
Outline

- Introduction
- **Hardware Architectures**
- Architecture Styles
- System Architectures
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - Decentralized Architecture: peer-to-peer
  - Hybrid Architecture

Hardware Architectures (Legacy)



Shared-nothing design



David J. DeWitt and Jim Gray. Parallel Database Systems: The Future of High Performance Database Systems. Communications of the ACM 35(6), 1992

Outline

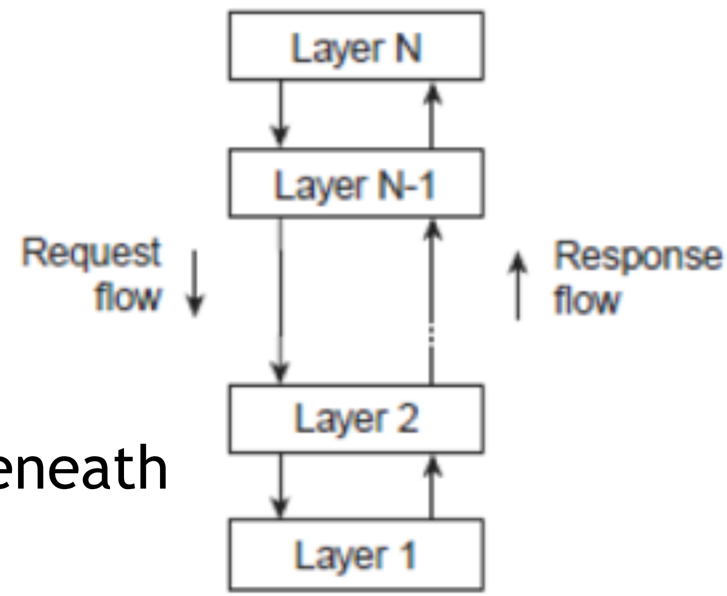
- Introduction
- Hardware Architectures
- **Architecture Styles**
- System Architectures
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - Decentralized Architecture: peer-to-peer
  - Hybrid Architecture

Architectural Styles

- Software architecture focuses on the software components and their interaction:
  - how components are connected to each other
  - data exchanged between components
  - how components are jointly configured into a system
- Mian objective: transparency
- Architecture Styles:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

Layered Architecture

- Layering == Abstraction
- Components are organized in layers
- Calls are only allowed in one direction
- Each layer offers a software abstraction
    - Higher layers are unaware of its implementation and the layers beneath it
- Layered architecture is used for client-server model

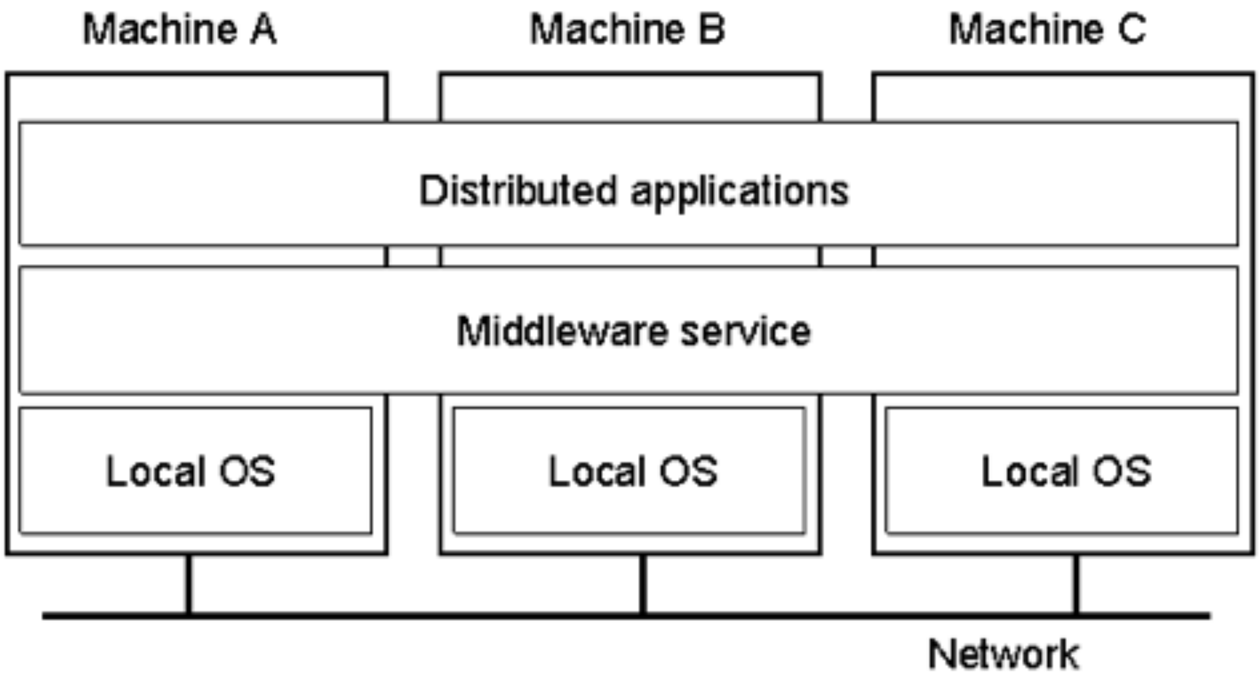


Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Distributed System Layers

Distributed applications, services,..
Hides lower layers and provides a communication platform

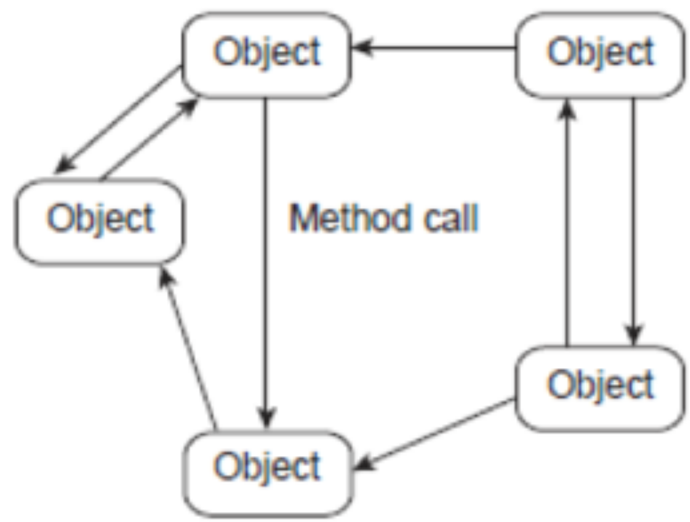
Communications and other hardware infrastructure



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Object-based Architecture

- Each object corresponds to a component
- Components are connected through remote procedure calls (RPC)
- Object-based style is used for distributed object systems



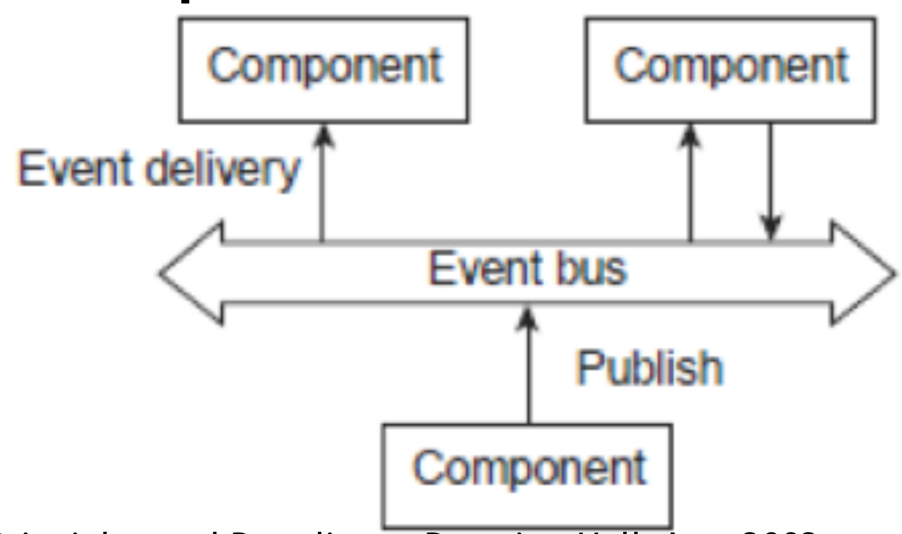
Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Data-centered Architecture

- Idea: Communication of processes is done through a common repository
- Examples:
  - Communication through files that are stored on a shared distributed file system
  - Shared Web-based data services

Event-based Architecture

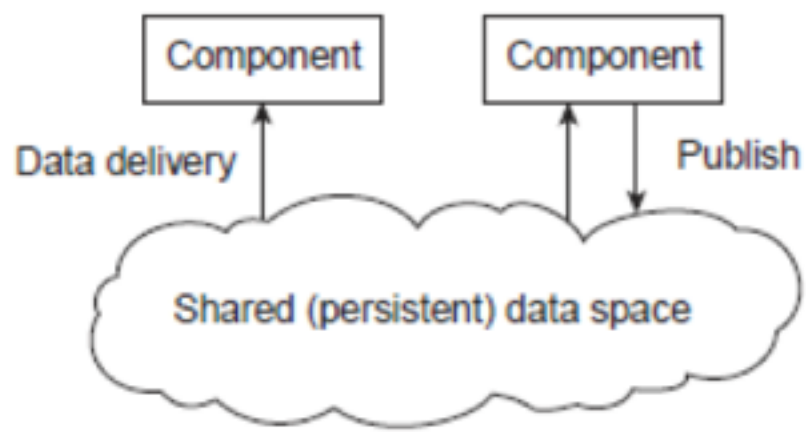
- Processes communicate through the propagation of events
- Example: publish/subscribe systems
- Processes are loosely coupled
- Decoupled in space



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Event-based + Data-centered Architecture

- A.K.A. shared data spaces
- Decoupled in space and time
  - Processes do not need to be active when communication takes place



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Outline

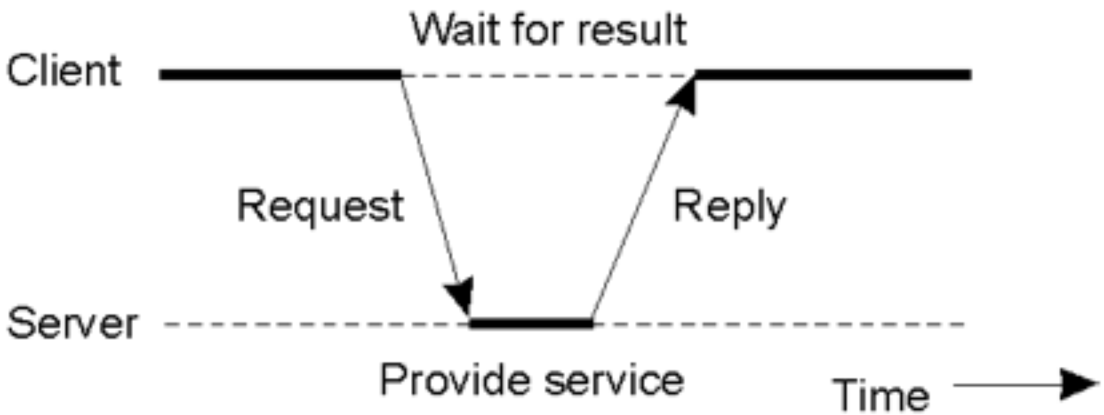
- Introduction
- Hardware Architectures
- Architecture Styles
- **System Architectures**
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - Decentralized Architecture: peer-to-peer
  - Hybrid Architecture

System Architectures

- Defines the structure of the system by identifying:
  - the components of the system
  - roles of each component
  - interrelationships and interactions between these components
  - how they map to the distributed infrastructure
- Goals: the architecture meets current and future demands
- Concerns: reliability, adaptability, manageability, cost-efficiency
- Types: Centralized, Decentralized, Hybrid

Client-Server Model

- Processes are divided into two groups:
  - Servers: processes offering services
  - Clients: processes requesting services
- Client-server interaction A.K.A. **request-reply**
- A connectionless protocol can be used (e.g. HTTP), however, mostly connection-oriented such as TCP/IP is used

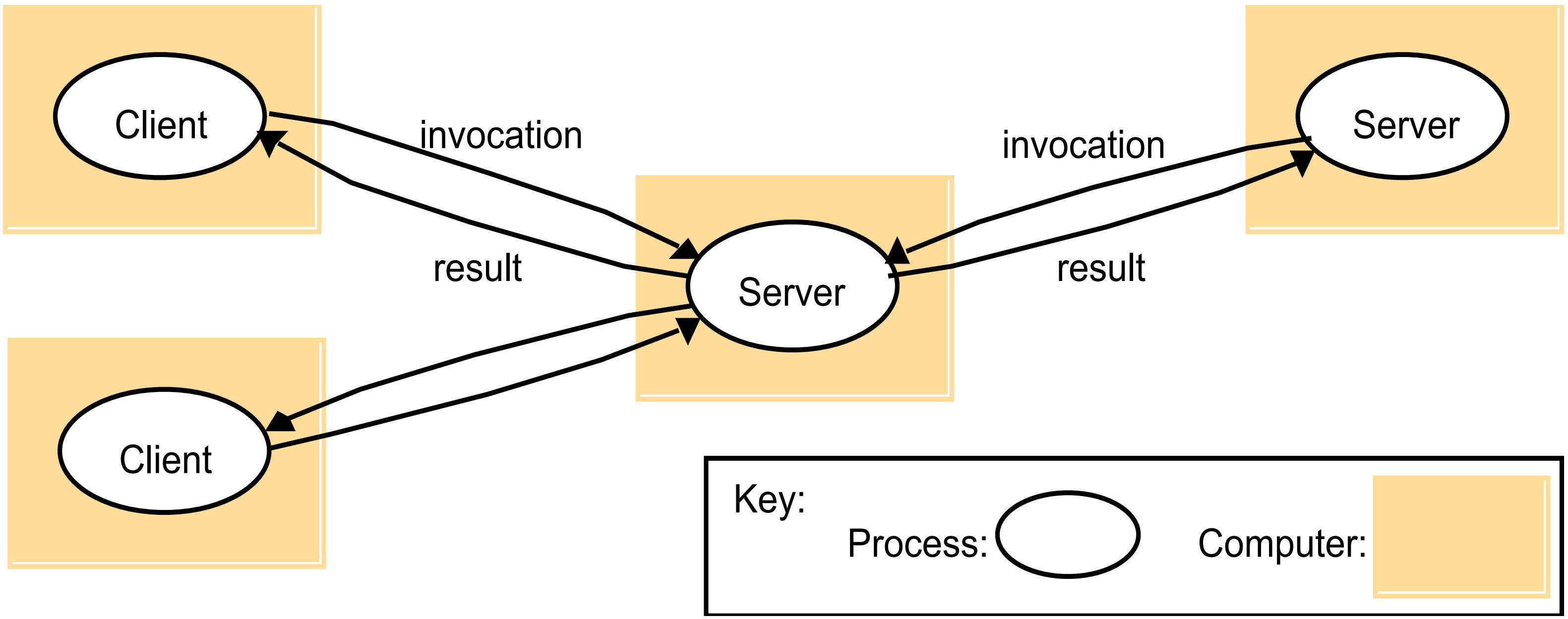


Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Client-Server Mechanism

- Client:
  - Clients is usually invoked by end users when they require service
  - A client usually blocks until server responds
- Server:
  - A process that provides service and usually with special privileges
  - A server waits for incoming requests
  - A server can have many clients making concurrent requests

Client-Server Communication



- Client processes interact with individual server processes in order to access shared resources
- Servers may be clients of other servers
  - Example: a web browser is a client of a web server, which can be a client of file server

An Example Client and Server (1)

- The *header.h* file used by the client and server

```
/* Definitions needed by clients and servers.              */
#define TRUE              1
#define MAX_PATH          255   /* maximum length of file name          */
#define BUF_SIZE          1024  /* how much data to transfer at once    */
#define FILE_SERVER       243   /* file server's network address        */

/* Definitions of the allowed operations */
#define CREATE            1     /* create a new file                    */
#define READ              2     /* read data from a file and return it  */
#define WRITE             3     /* write data to a file                 */
#define DELETE            4     /* delete an existing file              */

/* Error codes. */
#define OK                0     /* operation performed correctly        */
#define E_BAD_OPCODE      -1    /* unknown operation requested          */
#define E_BAD_PARAM       -2    /* error in a parameter                 */
#define E_IO              -3    /* disk error or other I/O error        */

/* Definition of the message format. */
struct message {
    long source;                /* sender's identity                    */
    long dest;                  /* receiver's identity                  */
    long opcode;                /* requested operation                  */
    long count;                 /* number of bytes to transfer          */
    long offset;                /* position in file to start I/O        */
    long result;                /* result of the operation              */
    char name[MAX_PATH];        /* name of file being operated on       */
    char data[BUF_SIZE];        /* data to be read or written           */
};
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

An Example Client and Server (2)

- A sample server

```
#include <header.h>
void main(void) {
      struct message ml, m2;                   /* incoming and outgoing messages          */
      int r;                                   /* result code                             */

      while(TRUE) {                            /* server runs forever                     */
            receive(FILE_SERVER, &ml);         /* block waiting for a message             */
            switch(ml.opcode) {                /* dispatch on type of request             */
                  case CREATE:   r = do_create(&ml, &m2); break;
                  case READ:     r = do_read(&ml, &m2); break;
                  case WRITE:    r = do_write(&ml, &m2); break;
                  case DELETE:   r = do_delete(&ml, &m2); break;
                  default:       r = E_BAD_OPCODE;
            }
            m2.result = r;                     /* return result to client                 */
            send(ml.source, &m2);              /* send reply                              */
      }
}
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

An Example Client and Server (3)

- A client using the server to copy a file

(a)

```
#include <header.h>
int copy(char *src, char *dst){              /* procedure to copy file using the server      */
    struct message ml;                       /* message buffer                               */
    long position;                           /* current file position                        */
    long client = 110;                       /* client's address                             */

    initialize( );                           /* prepare for execution                        */
    position = 0;
    do {
        ml.opcode = READ;                    /* operation is a read                          */
        ml.offset = position;                /* current position in the file                 */
        ml.count  = BUF_SIZE;                                                                /* how many bytes to read*/
        strcpy(&ml.name, src);               /* copy name of file to be read to message      */
        send(FILESERVER, &ml);               /* send the message to the file server          */
        receive(client, &ml);                /* block waiting for the reply                  */

        /* Write the data just received to the destination file.                             */
        ml.opcode = WRITE;                   /* operation is a write                         */
        ml.offset = position;                /* current position in the file                 */
        ml.count  = ml.result;               /* how many bytes to write                      */
        strcpy(&ml.name, dst);               /* copy name of file to be written to buf       */
        send(FILE_SERVER, &ml);              /* send the message to the file server          */
        receive(client, &ml);                /* block waiting for the reply                  */
        position += ml.result;               /* ml.result is number of bytes written         */
    } while( ml.result > 0 );                /* iterate until done                           */
    return(ml.result >= 0 ? OK : ml result); /* return OK or error code                      */
}
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

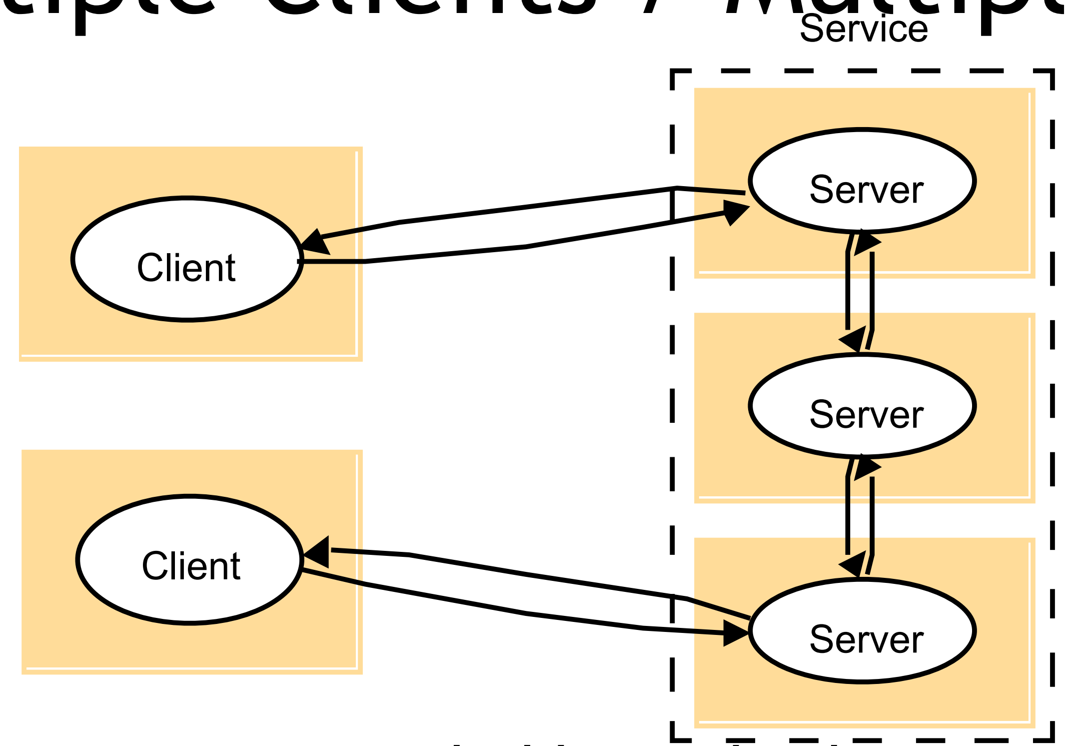
Advantages of the Client-Server Architecture

- Efficient division of labor
- Horizontal and vertical scaling of resources
- Better price/performance on client machines
- Ability to use familiar tools on client machines
- Client access to remote data (via standards)
- Full DBMS functionality provided to client workstations
- Overall better system price/performance

Problems with the Multiple Client / Single Server Architecture

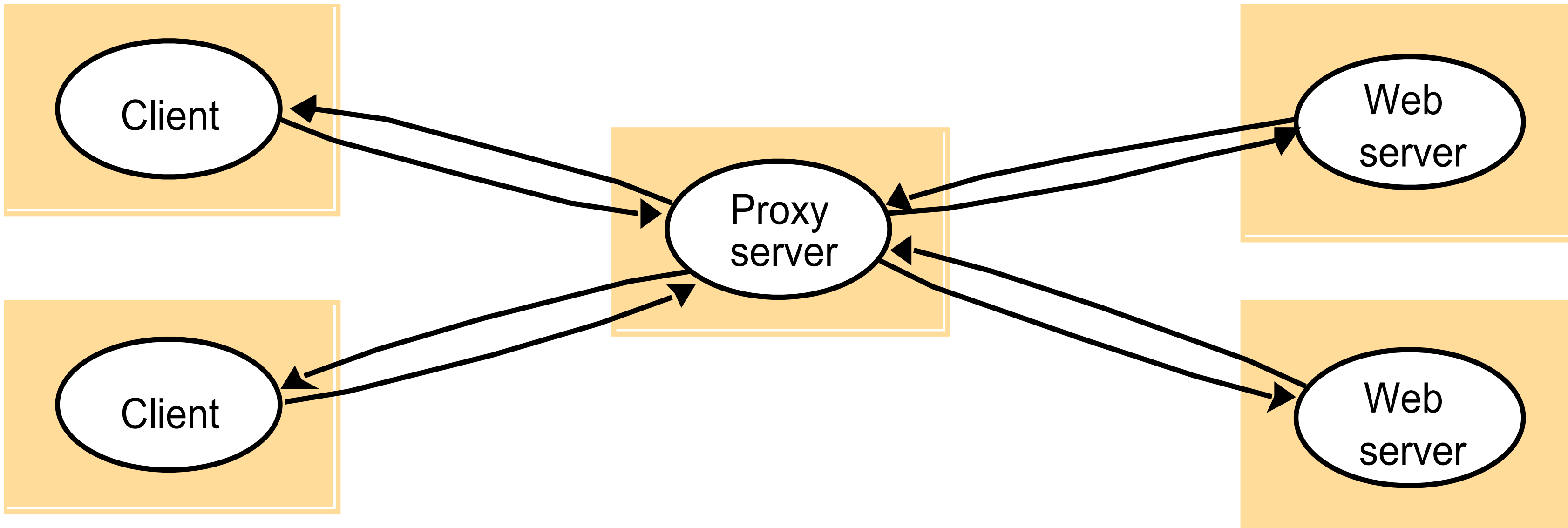
- Server forms a bottleneck
- Server forms a single point of failure
- System scaling is difficult

Multiple Clients / Multiple Servers



- Same service provided by multiple servers
- Data is partitioned (and/or replicated), each web server manages its own set of resources
- A user can employ a browser to access resources at any of the servers

Multiple Clients / Multiple Servers Using Proxy Server

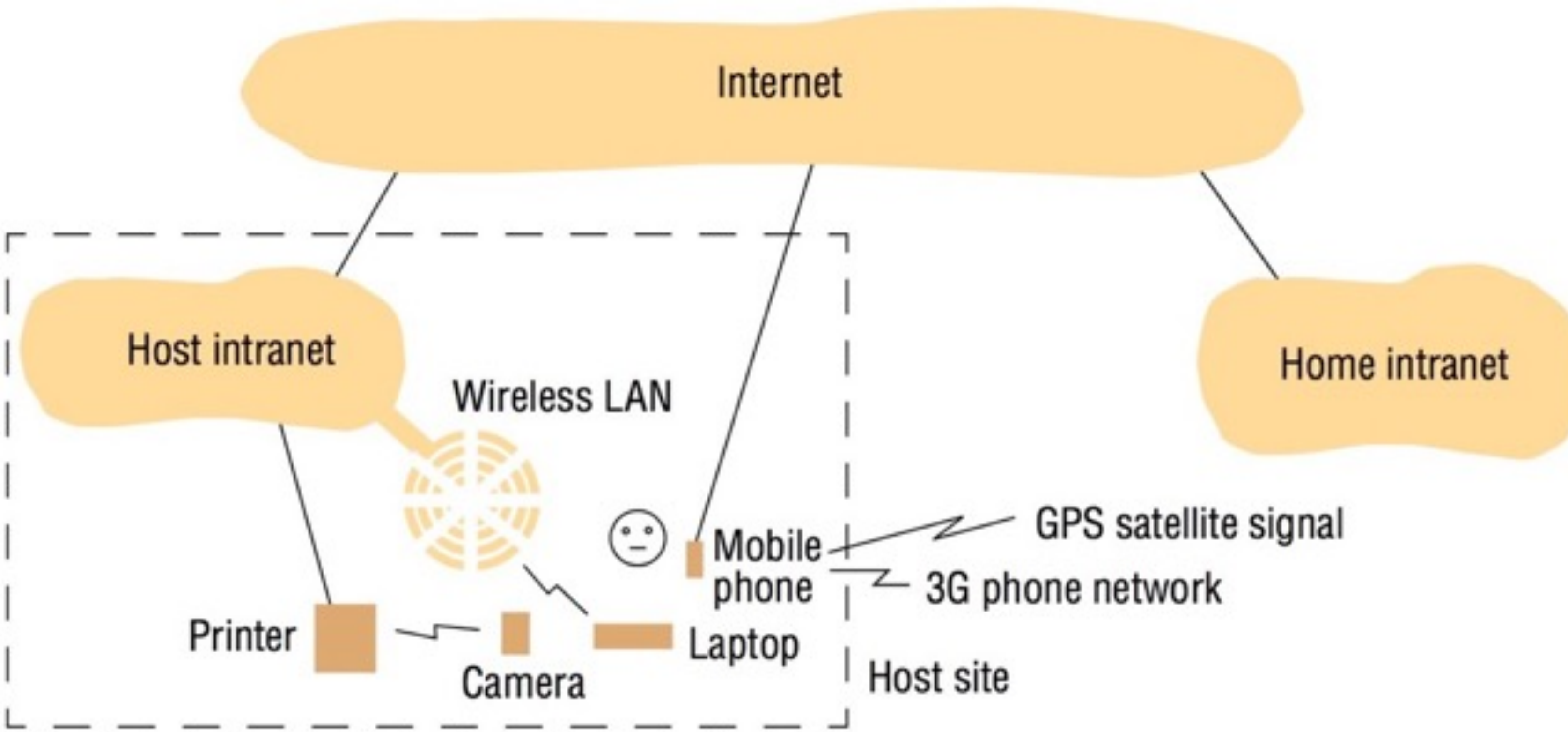


- A ***cache*** is a store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves
- Web proxy servers provide a shared cache of web resources

Outline

- Introduction
- System Layers
- Hardware Architectures
- Architecture Styles
- System Architectures
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - Decentralized Architecture: peer-to-peer
  - Hybrid Architecture

Mobile Computing



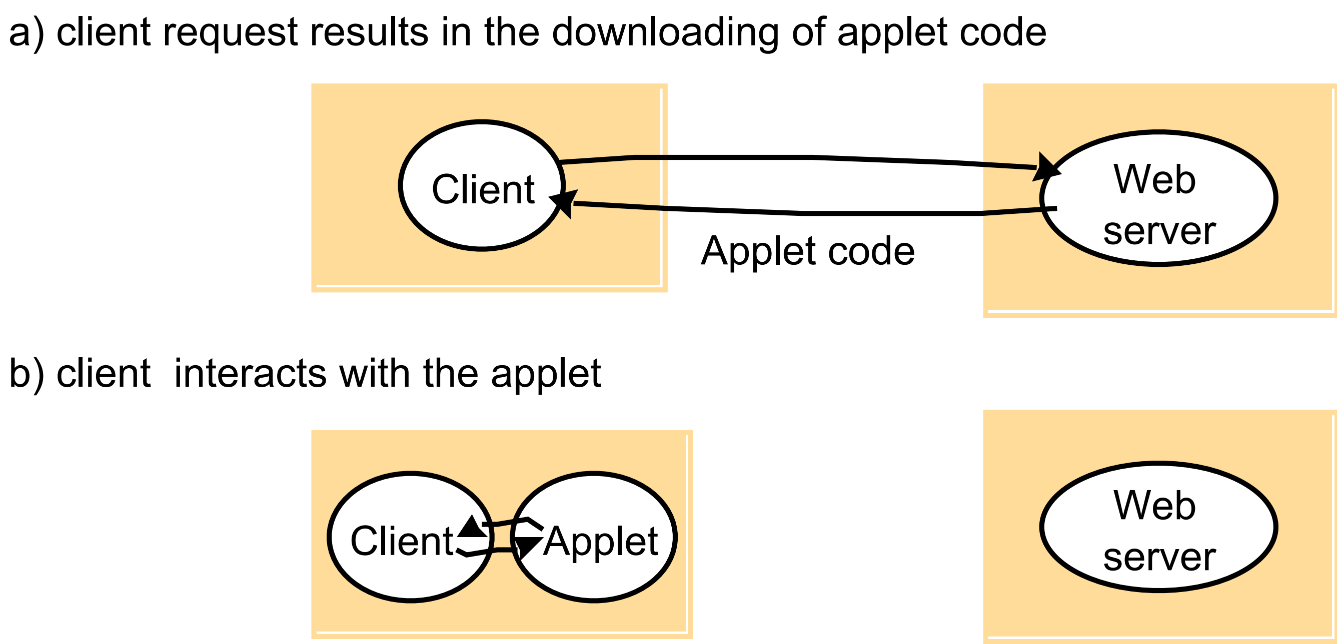
Mobile computing is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment

Characteristics of Mobile Computing

- Users who are away from their ‘home’ intranet can still access their ‘home’ intranet resources via the devices they carry with them
- Users can also access their ‘hosting’ intranet resources.

  (*location-aware*, *context-aware computing*)

Applets as an Example of Mobile Code



- Advantages: does not suffer from the delays or variability of the network bandwidth ➔ good interactive response
- Disadvantages: Mobile code is a potential security threat to the local resources in the destination computer

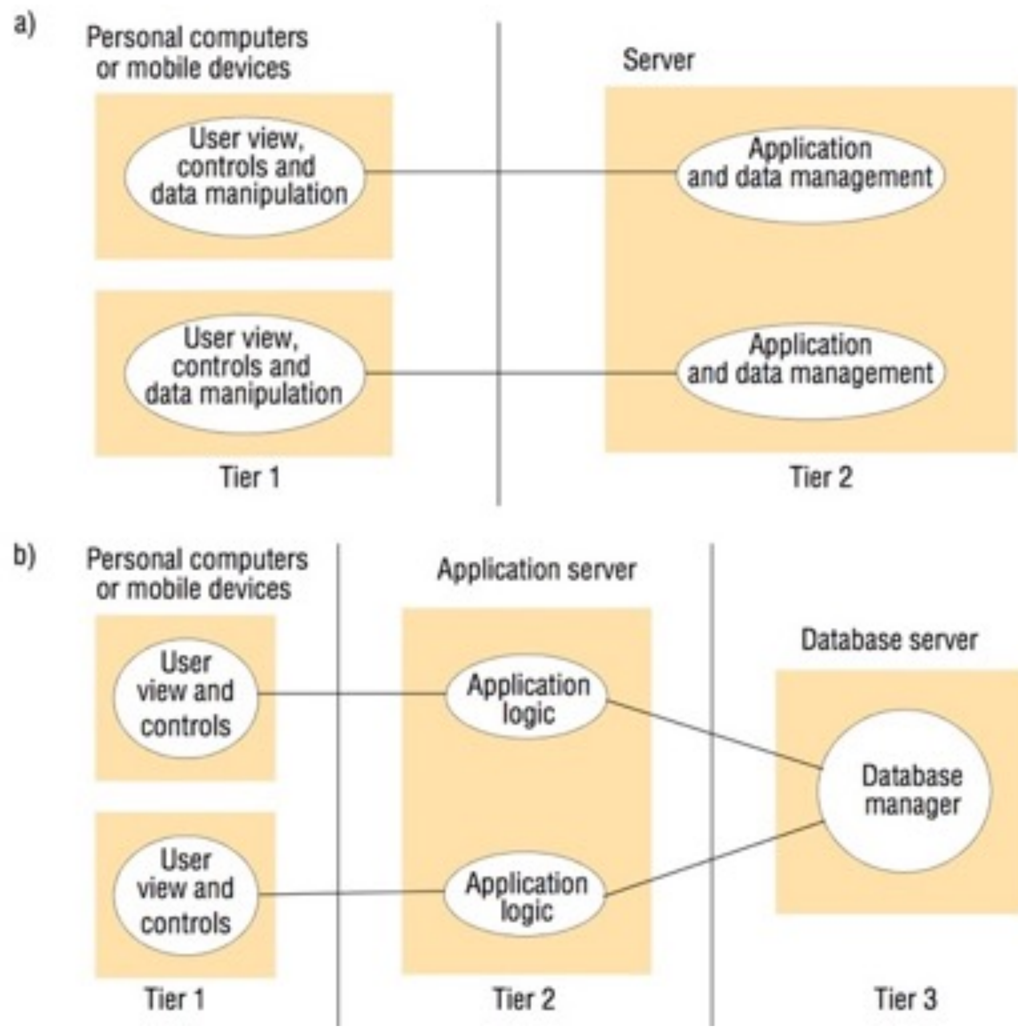
Mobile Agents

- A **mobile agent** is a running program (code and data) that travels from one computer to another in a network carrying out a task on someone's behalf, such as collecting information, and eventually returning with the results
- Remote invocations vs. mobile agents (data vs code)
- Advantages:
  - Reduction in communication cost and time
- Disadvantages:
  - Potential security threat to the resources in computers that they visit
  - Unable to complete their task if they are refused access to the information they need

Application Layering / Multitier systems

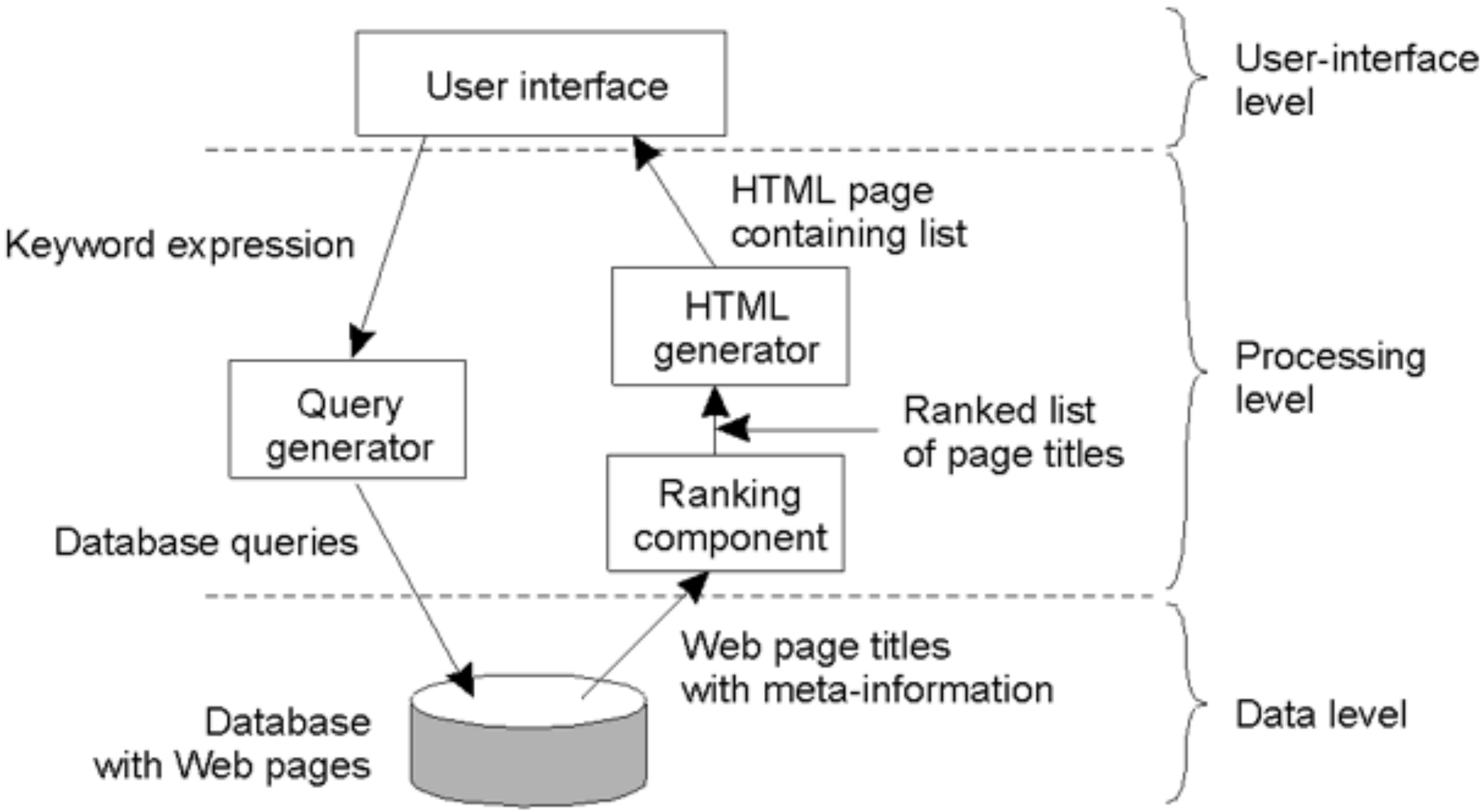
- Tiered architectures are complementary to layering:
  - **Layering** deals with the vertical organization of services into layers of abstraction
  - **Tiering** organizes functionality of a given layer and place this functionality into appropriate servers and on to physical nodes
- Layers of a three-tier architecture
  - User interface (presentation logic): user interaction and updating the view of the application as presented to the user
  - Application/ processing logic: application-specific processing
  - Data logic: the persistent storage of the application, e.g., DBMS

Two-tier and Three-tier Architectures



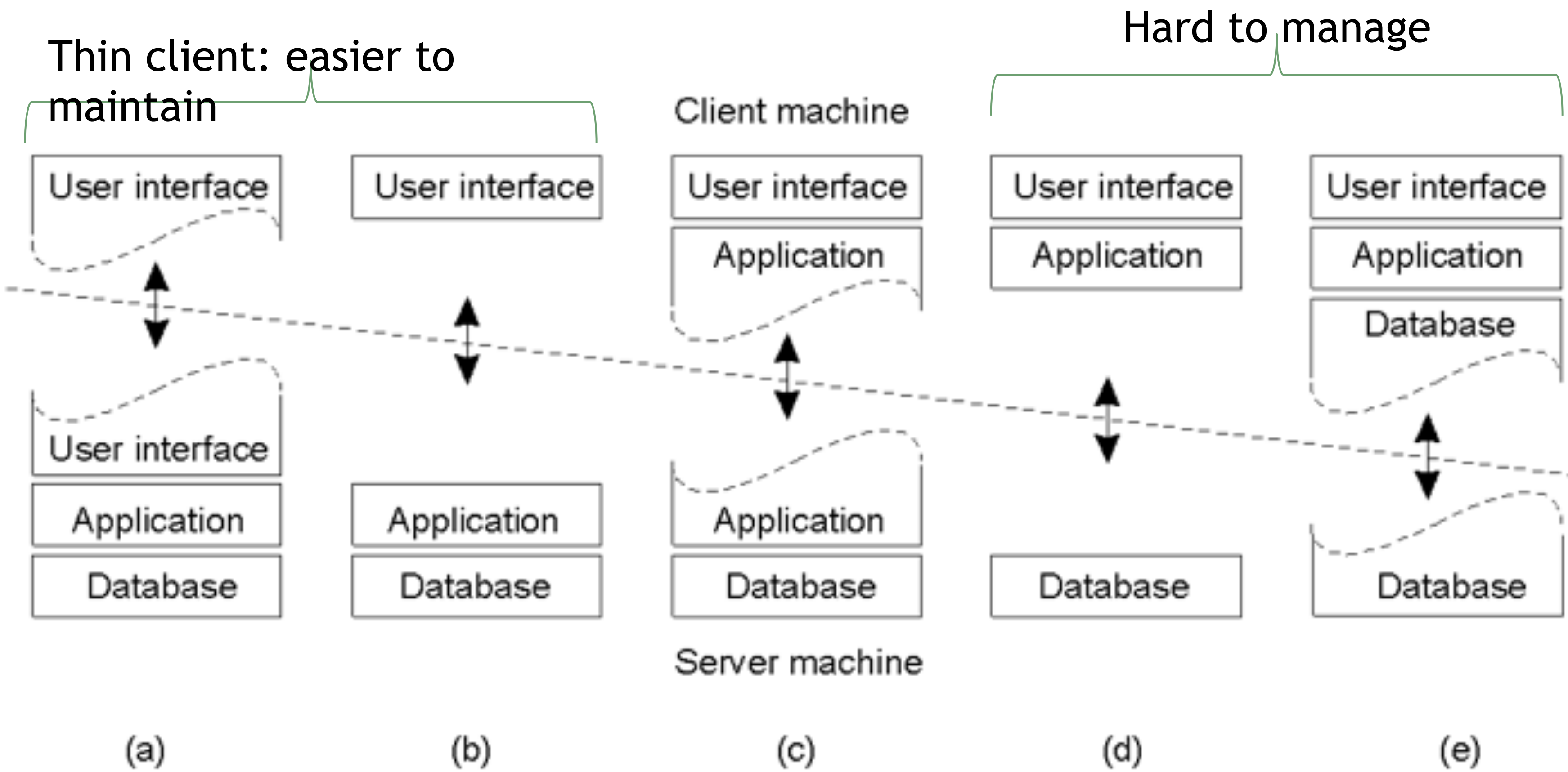
Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012

Example of Multitier Systems: Internet Search Engines



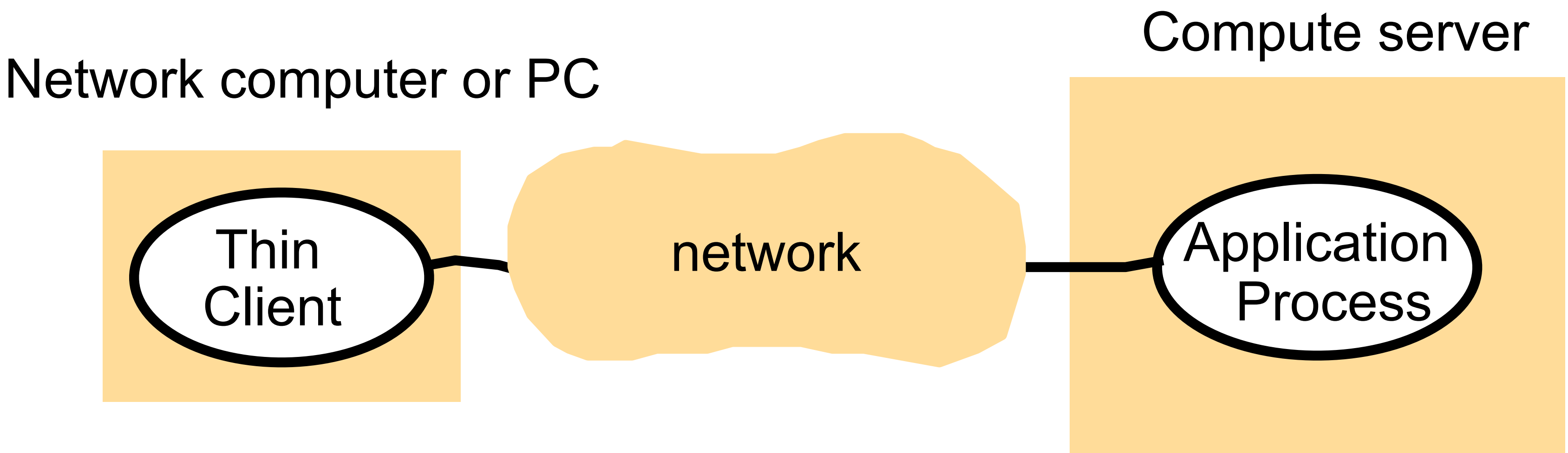
Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Alternatives of Multitier Systems



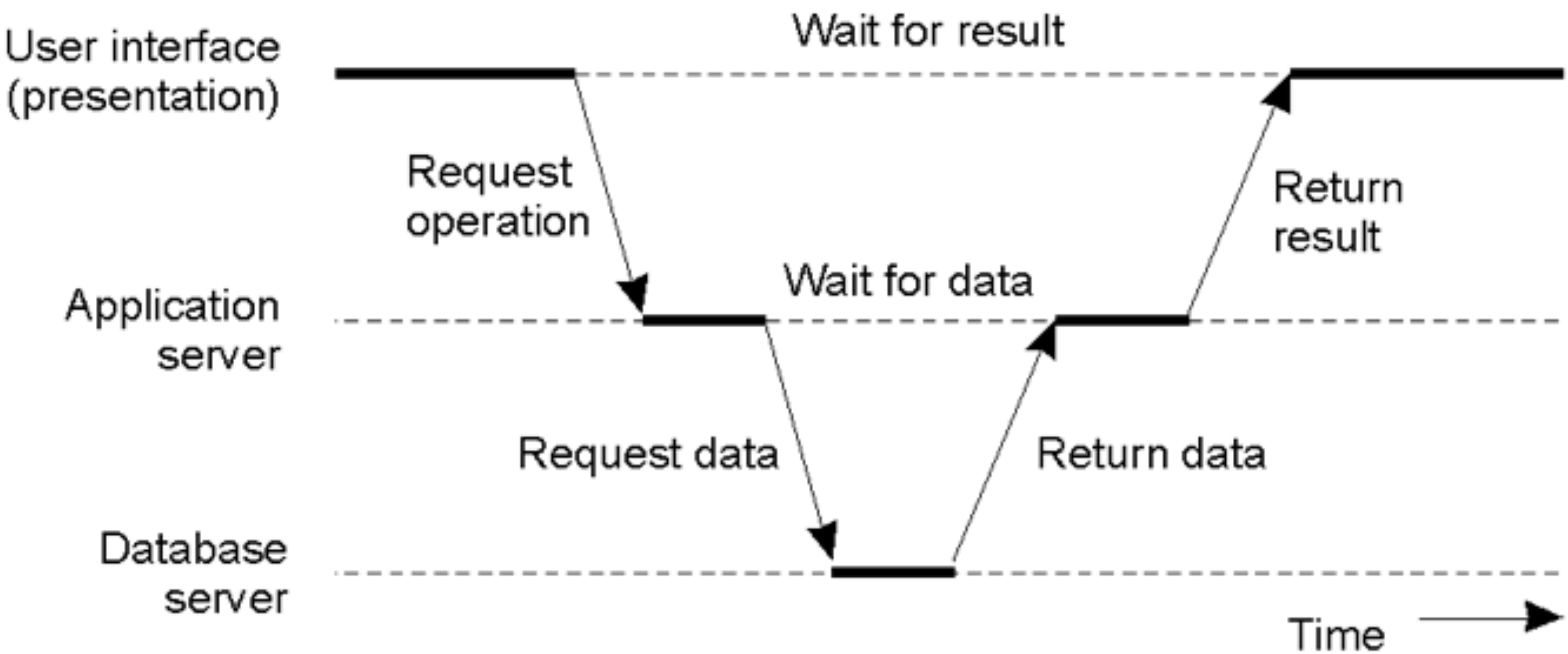
Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Thin Clients



- Moving complexity away from the end-user device towards services in the Internet
- Advantage: the user does not need high end computing machines
- Disadvantage: delays due to accessing remote data, graphics, ..

Communication in Multitier Systems

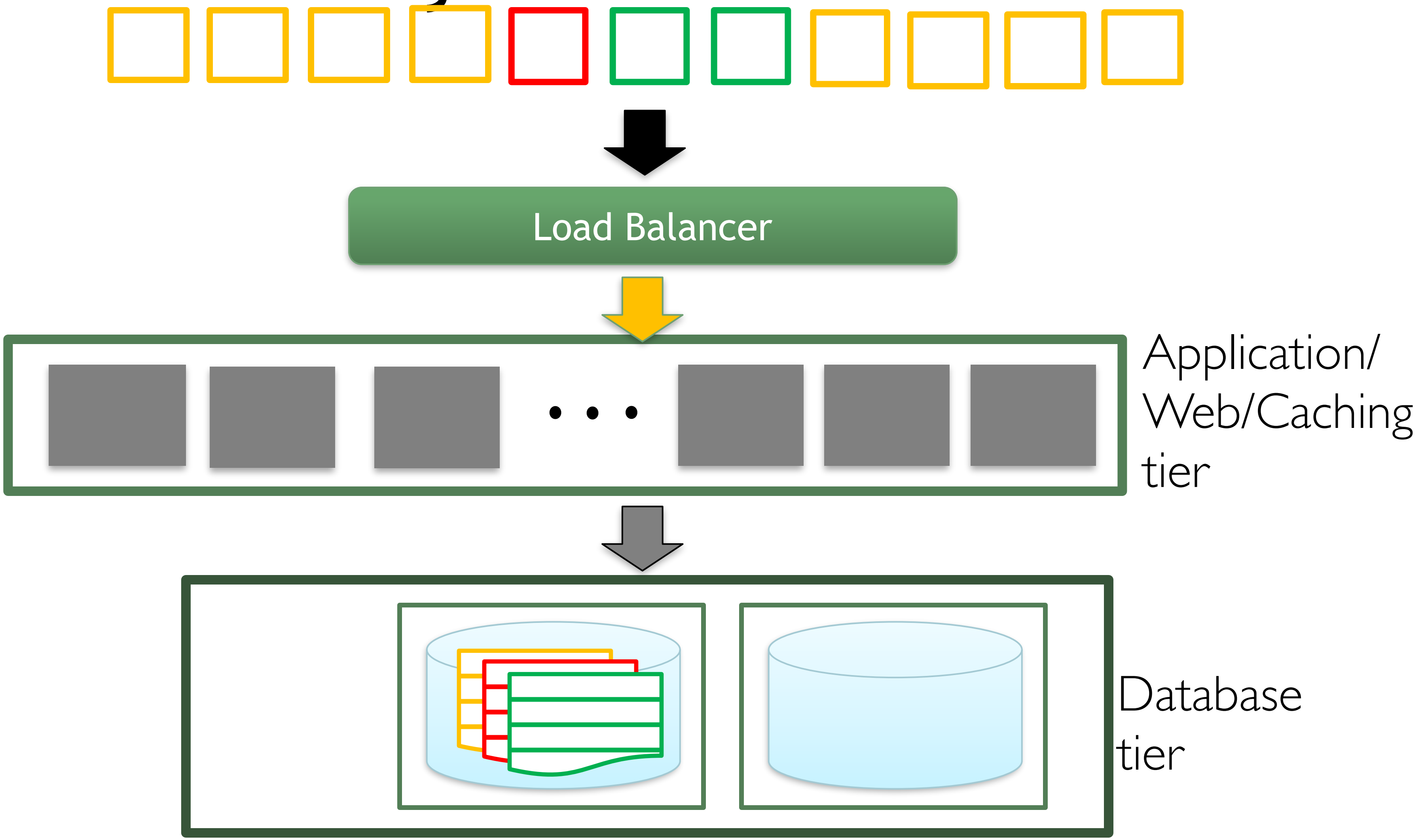


Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Advantages of Multitier Systems

- Frees clients from dependencies on the exact implementation of the database
- It allows “business logic” to be concentrated in one place
- Software updates are restricted to the middle layer
- Performance improvements are possible by batching requests from many clients to the database
- Database and business logic tiers could be implemented by multiple servers for scalability

Elasticity in a Multitenant DB



Sudipto Das, Shoji Nishimura, Divyakant Agrawal, Amr El Abbadi: Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud using Live Data Migration. VLDB 2011.

Outline

- Introduction
- Hardware Architectures
- Architecture Styles
- System Architectures
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - **Decentralized Architecture: peer-to-peer**
  - Hybrid Architecture

Decentralized Architectures

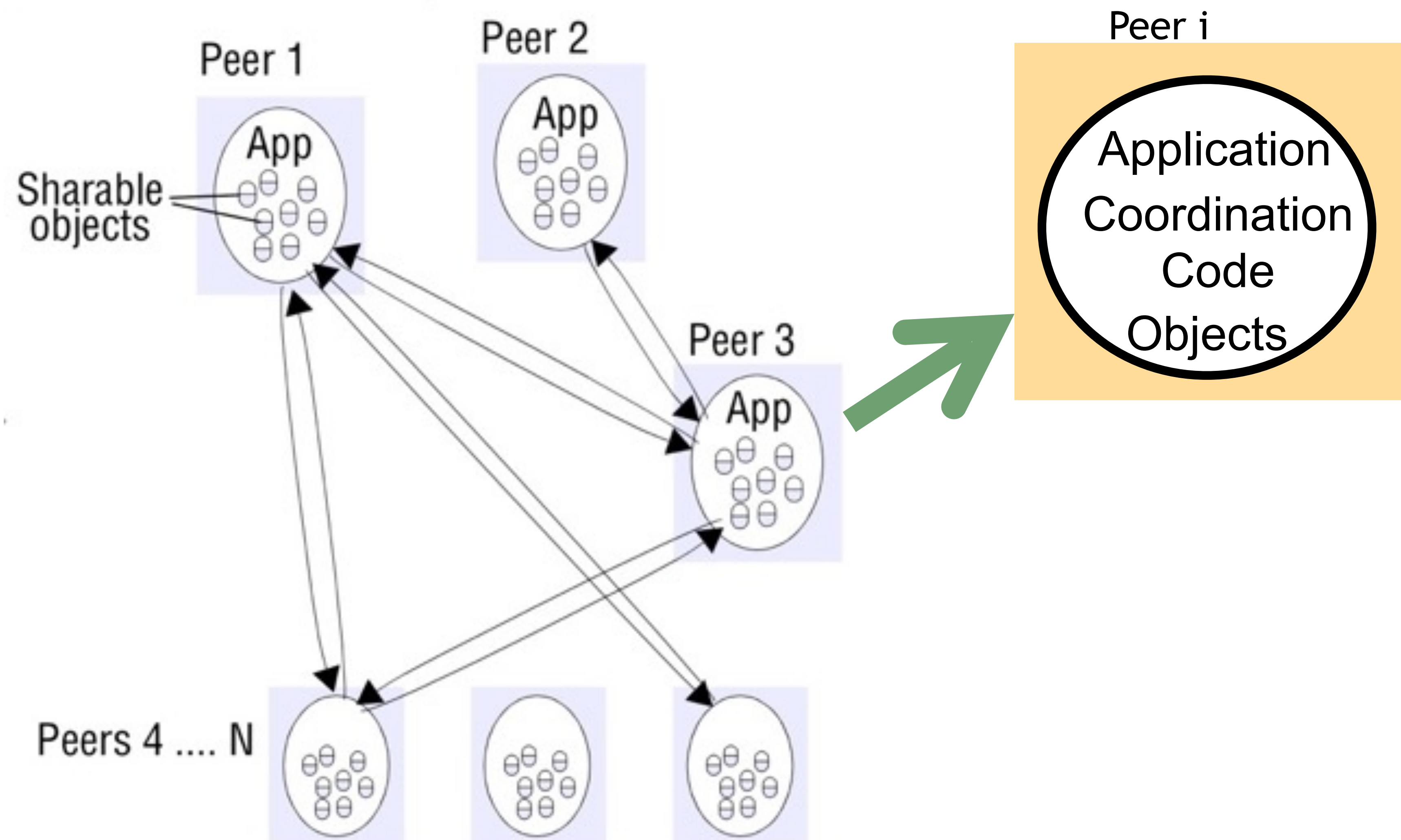
- Vertical distribution (multitier systems) vs horizontal distribution (peer-to-peer)
- The processes that constitute a peer-to-peer system are all equal
- The interaction between processes is symmetric: each process will act as a client and a server at the same time
- Peer-to-peer systems can be divided into:
  - **Structured P2P**: nodes are organized through a distributed data structure (e.g., DHT)
  - **Unstructured P2P**: nodes have randomly selected neighbors
  - **Hybrid P2P**: some nodes are appointed special functions

Decentralized Architectures

- Vertical distribution (multitier systems) vs horizontal distribution (peer-to-peer)
- The processes that constitute a peer-to-peer system are all equal.
- The interaction between processes is symmetric: each process will act as a client and a server at the same time.
- Peer-to-peer systems can be divided into:

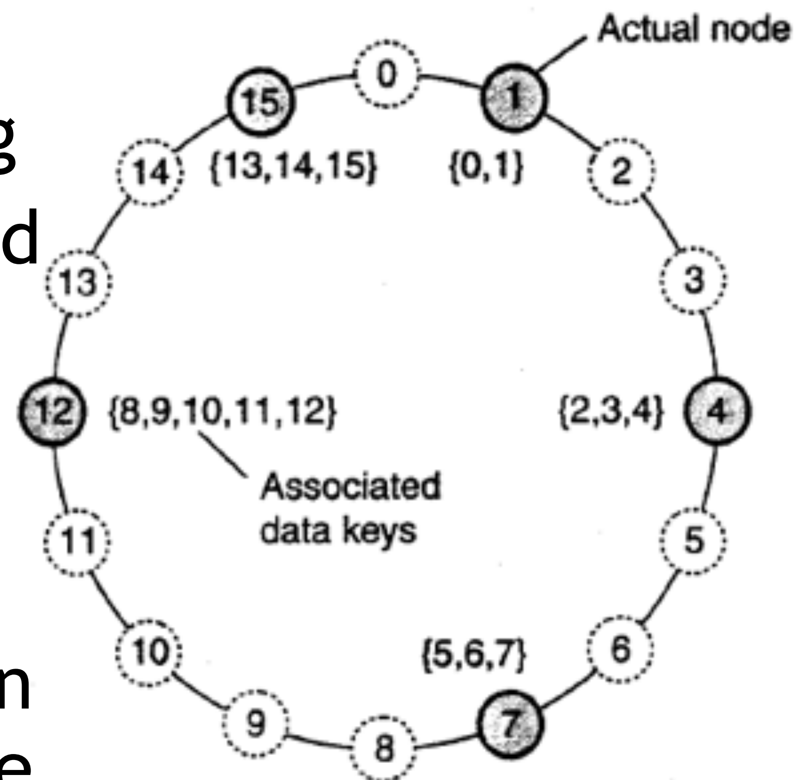
Organize the processes in an overlay network, that is, a network in which the nodes are formed by the processes and the links represent the possible communication channels (e.g., TCP connections ➔ application-level multicasting).

Peer-to-Peer Systems



Structured Peer-to-Peer

- Chord System: nodes are logically organized in a ring
- Mapping between nodes and the data they own is required
- Function lookup(k) returns the network address of the node owning k. Lookups can be done in O(log(N)), where N is the number of nodes



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Unstructured Peer-to-Peer

- Rely on **randomized algorithms** for constructing overlay networks that resembles a **random graph**
- Main idea:
  - Each node maintains a list of neighbors, but that this list is constructed in a more or less random way.
  - Data items are assumed to be randomly placed on nodes.
  - Goal is that each node constructs a partial view of the graph.

Examples of Peer-to-Peer Applications

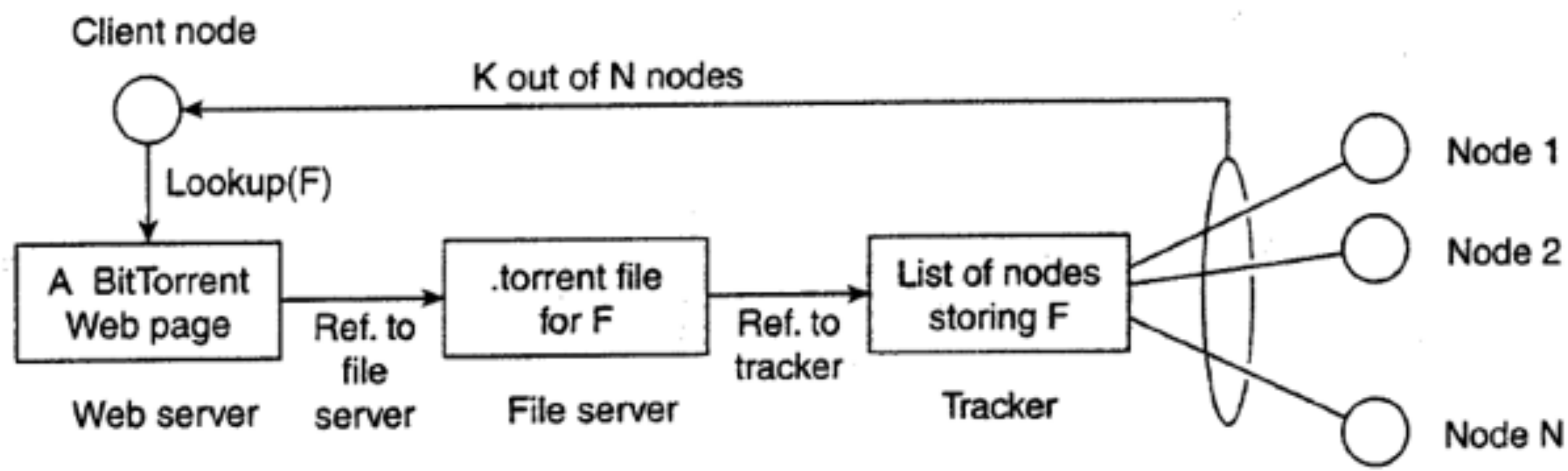
- File sharing
  - Napster, Gnutella, KaZaa
  - Second generation projects
    - Oceanstore, PAST, Freehaven, FreeNet
- Distributed Computation
  - SETI@home, Entropia, Parabon, United Devices, Popular Power
- Other Applications
  - Content Distribution (BitTorrent)
  - Instant Messaging (Jabber), Anonymous Email
  - Groupware (Groove)
  - P2P Databases

Outline

- Introduction
- Hardware Architectures
- Architecture Styles
- System Architectures
  - Centralized Architecture
    - Client-Server
    - Multiple-clients / multiple-servers
    - Mobile computing
    - Thin client
    - Application Layering
  - Decentralized Architecture: peer-to-peer
  - **Hybrid Architecture**

Hybrid Architecture

- Solution with client-server architectures are combined with decentralized architectures



- 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

Summary

- Hardware architectures: shared noting, shared memory, and shared disk architectures
- Software architectures: layered, object-based, data centred, and event-based architectures
- System architectures: centralized, decentralized, and hybrid architectures

Thank You