Module 1 - Distributed System Architectures & Models

Architecture

- Distributed systems tend to be very complex.
- It is critical to properly organize these systems to manage the complexity.
- The organization of a distributed system is primarily about defining the software components that constitute the system.
  - A component is a modular unit with well-defined required and provided interfaces.

Architecture (2)

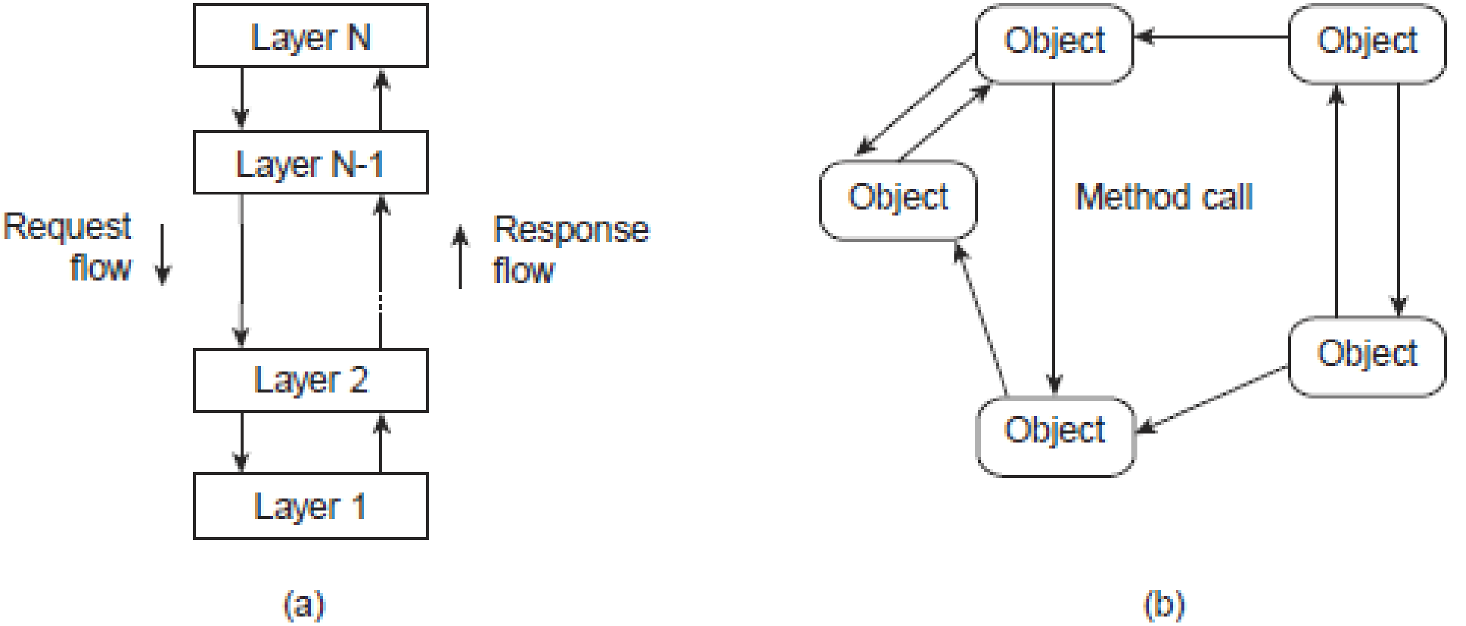
- Software Architecture:
  - Tells us how software components should be organized and how they should interact.

- System Architecture:
  - Instantiation of a software architecture on real machines.
  - Functions of each component are defined
  - Interrelationships and interactions between components are defined

Architectural styles

Basic idea

Organize into logically different components, and distribute those components over the various machines.



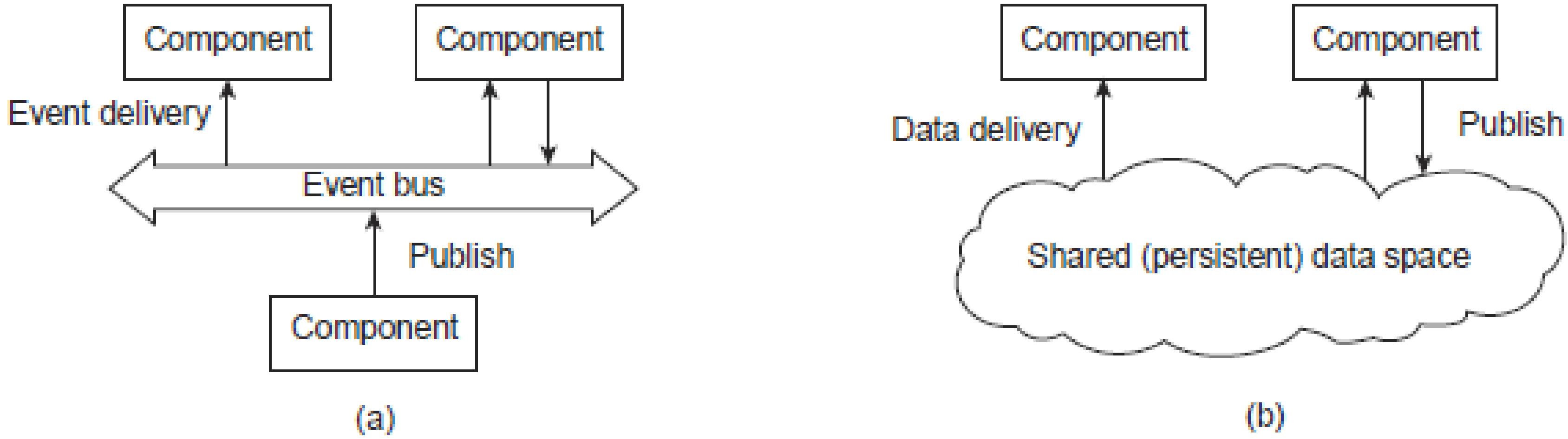
(a) (b)

(a) Layered style is used for client-server system
(b) Object-based style for distributed object systems.

Architectural Styles

Observation

Decoupling processes in space ("anonymous") and also time ("asynchronous") has led to alternative styles.



(a) Publish/subscribe [decoupled in space]
(b) Shared dataspace [decoupled in space and time]

System Architectures

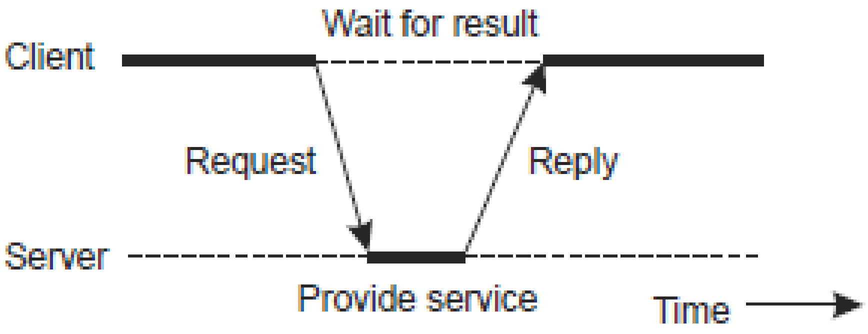
- Centralized architectures
  - Client-server
    - Multiple-client/single-server
    - Multiple-client/multiple-servers
  - Multitier systems
- Decentralized architectures

Centralized Architectures

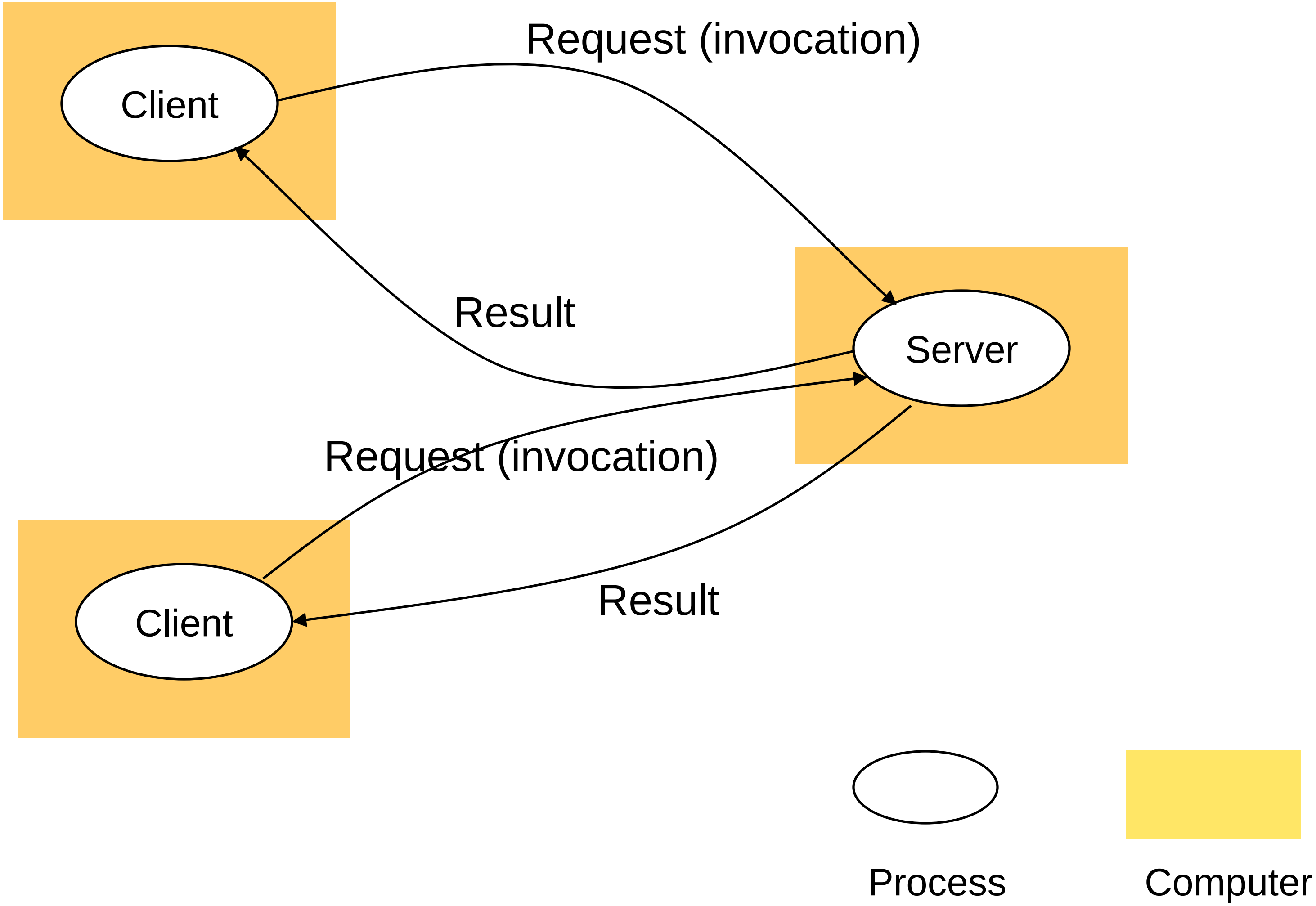
Basic Client–Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model wrt to using services



Client-Server Communication

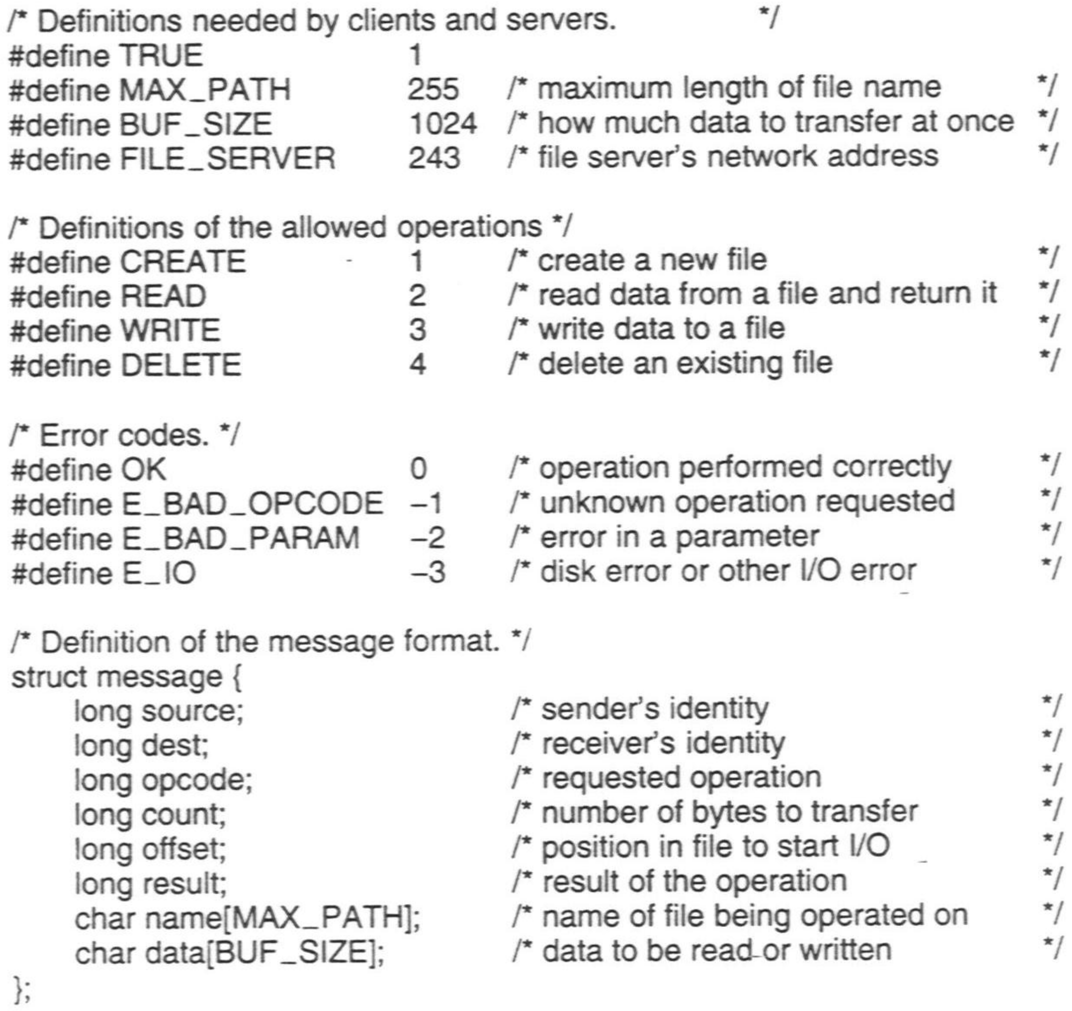


Advantages of Client/Server Computing

- More 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

An Example Client and Server (1)

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

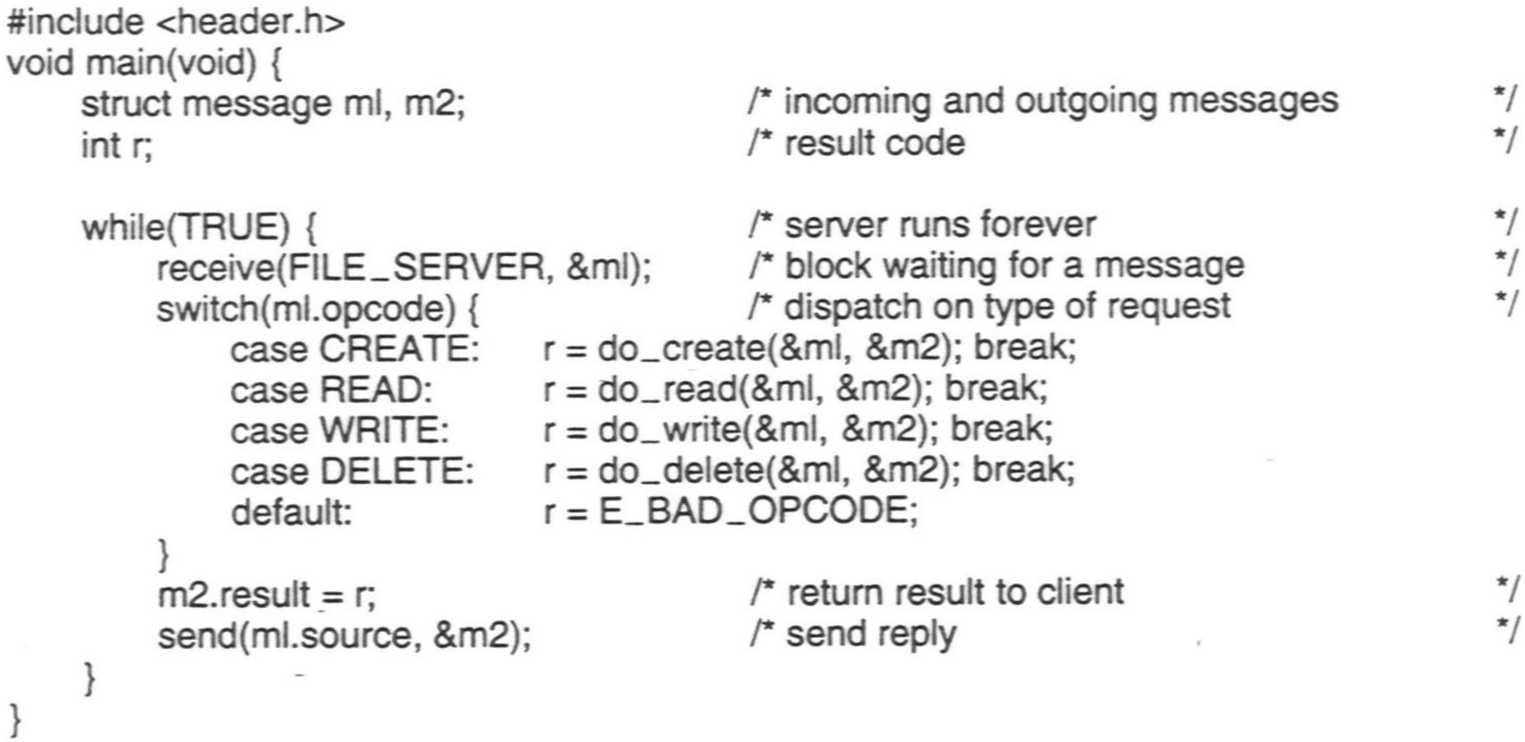


```
/* Definitions needed by clients and servers.              */
#define TRUE                 1
#define MAX_PATH             255    /* maximum length of file name         */
#define BUF_SIZE             1024   /* how much data to transfer at once   */
#define FILE_SERVER          243    /* file server's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

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

An Example Client and Server (2)

- A sample server.

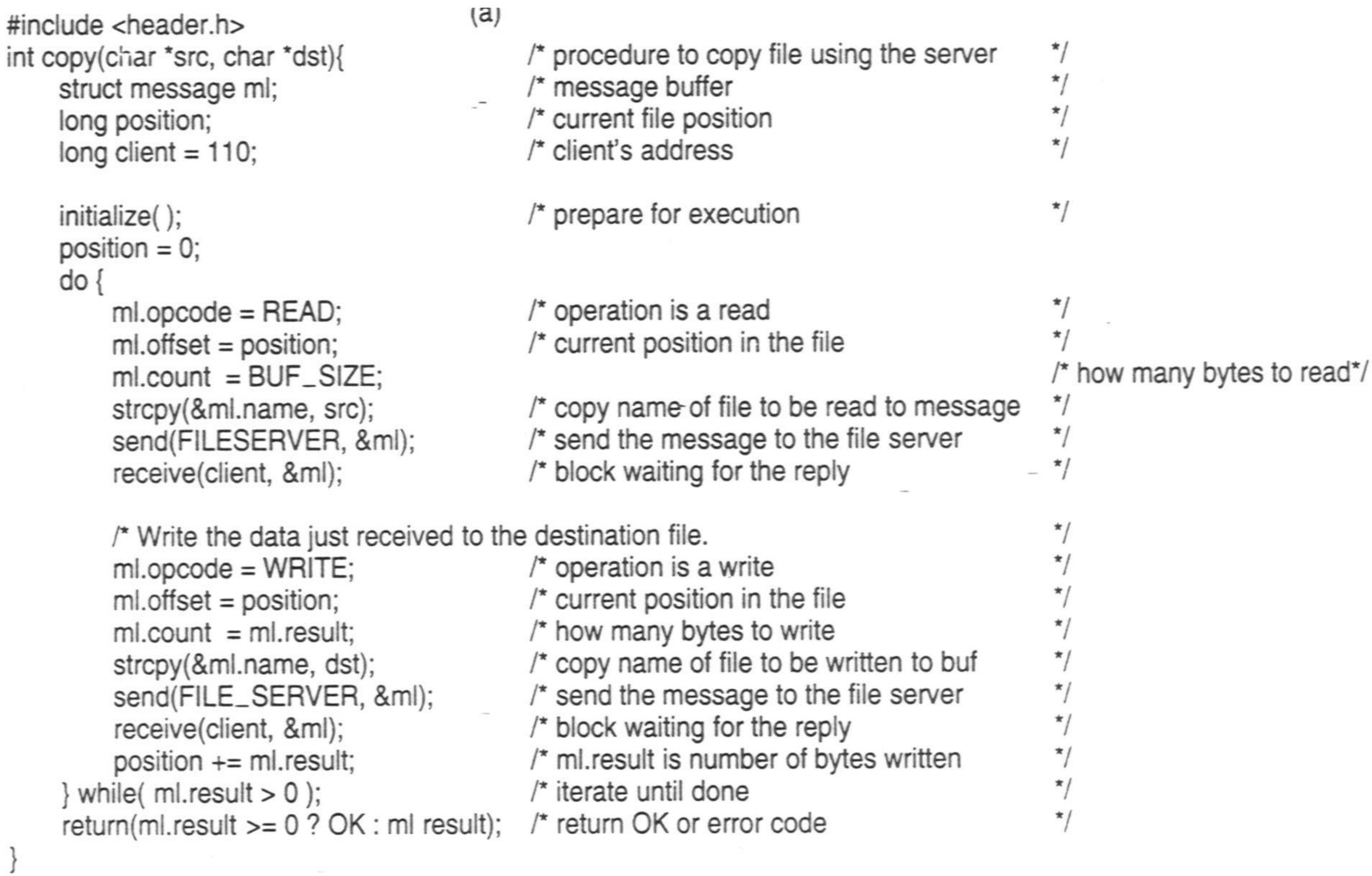


```
#include <header.h>
void main(void) {
    struct message ml, m2;                  /* incoming and outgoing messages      */
    int r;                                  /* result code                         */

    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                          */
    }
}
```

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

An Example Client and Server (3)

- A client using the server to copy a file.



(a)

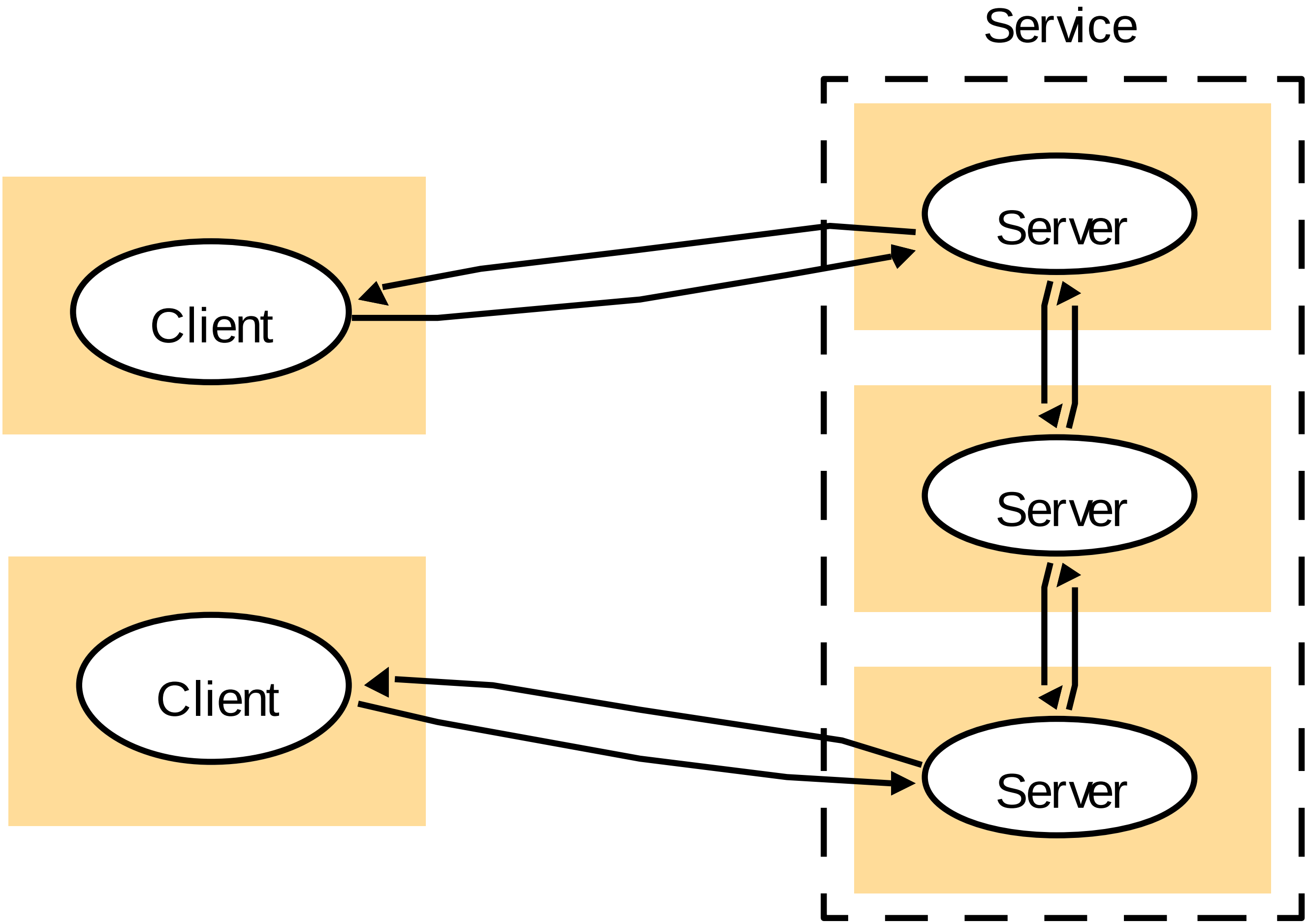
```
#include <header.h>
int copy(char *src, char *dst){              /* procedure to copy file using the server     */
     struct message ml;                       /* message buffer                              */
     long position;                           /* current file position               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

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

Problems With Multiple-Client/Single Server

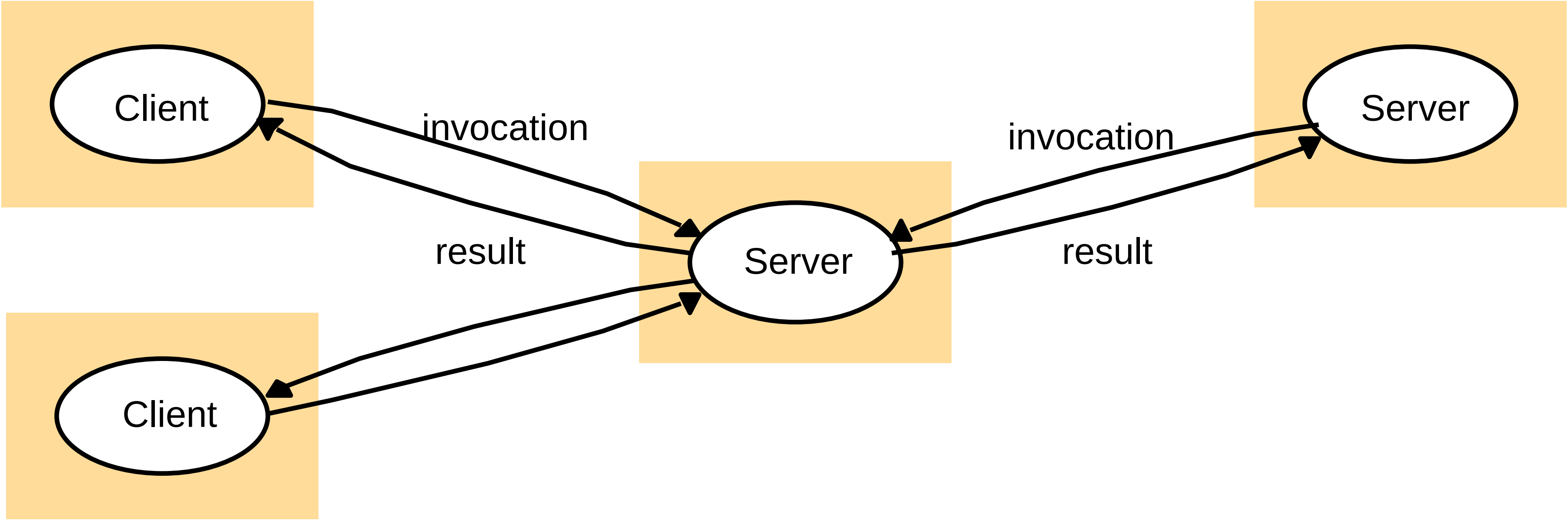
- Server forms bottleneck
- Server forms single point of failure
- System scaling difficult

Service Across Multiple Servers



From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Multiple-Client/Multiple-Server Communication



From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Application Layering

Traditional three-layered view

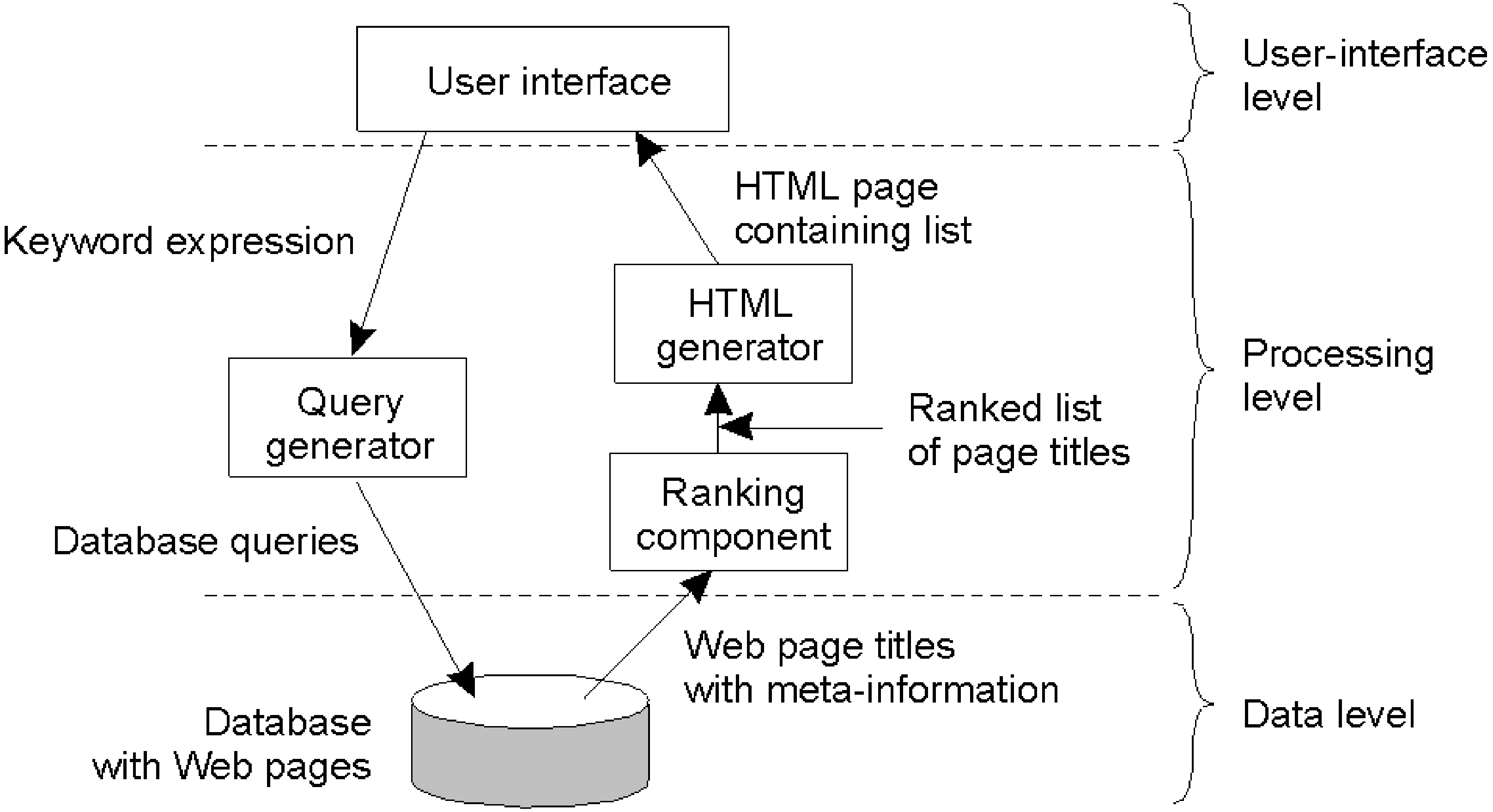
- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

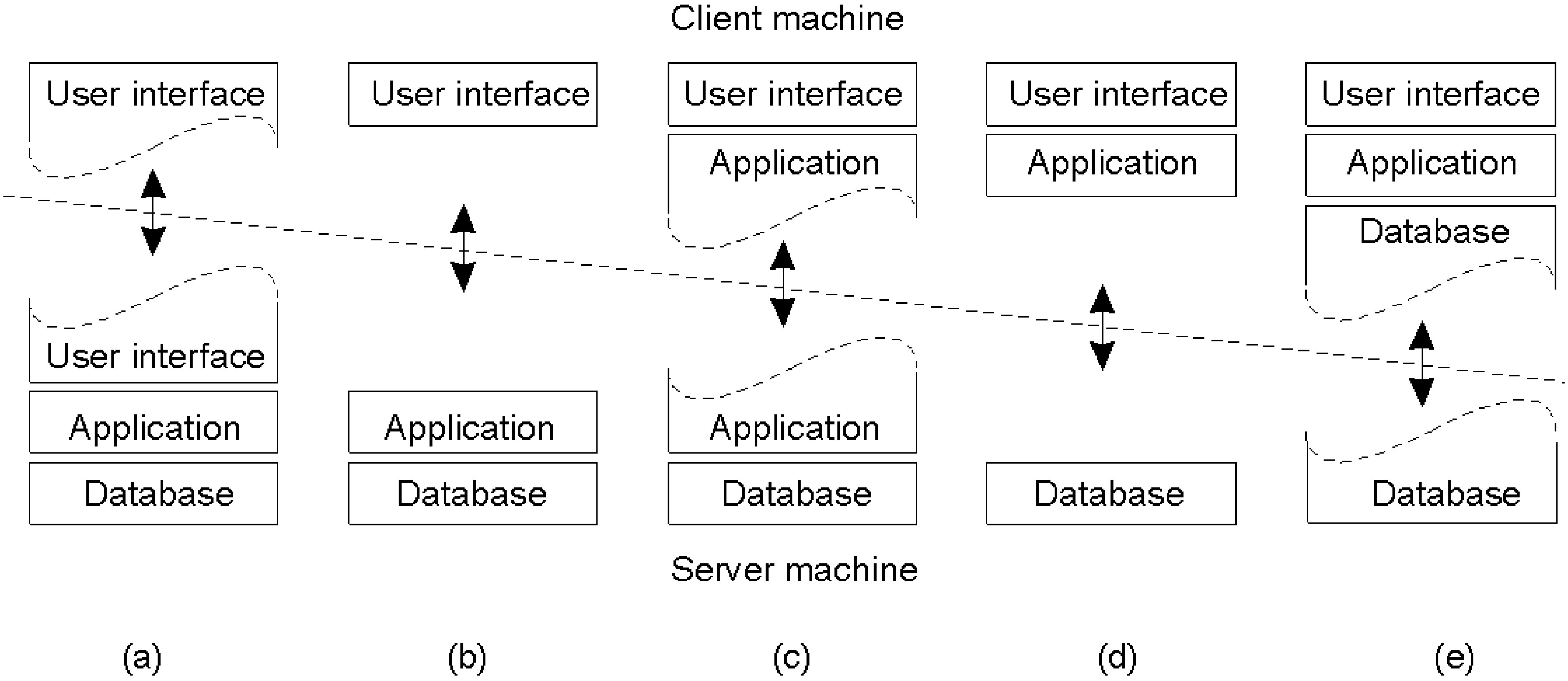
Multitier Systems

- Example: Internet Search Engines



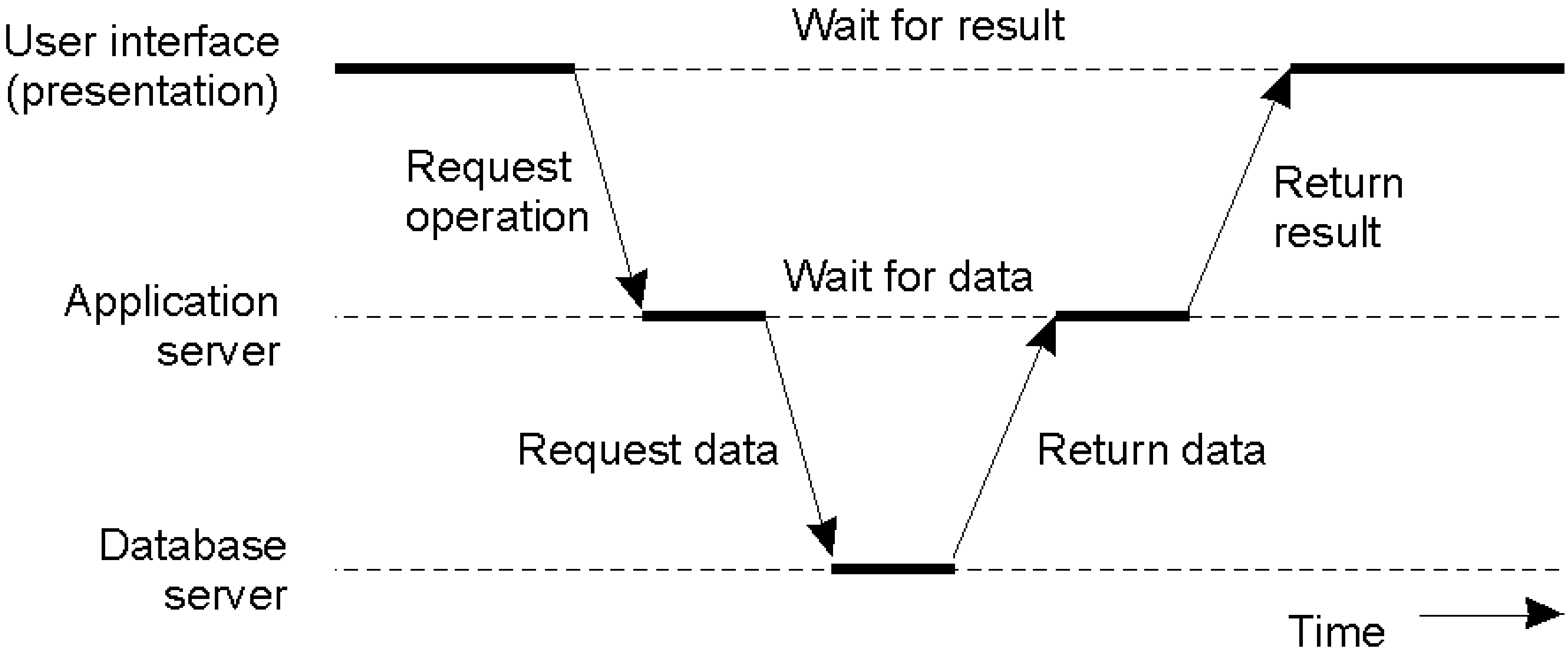
From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

Multitier System Alternatives



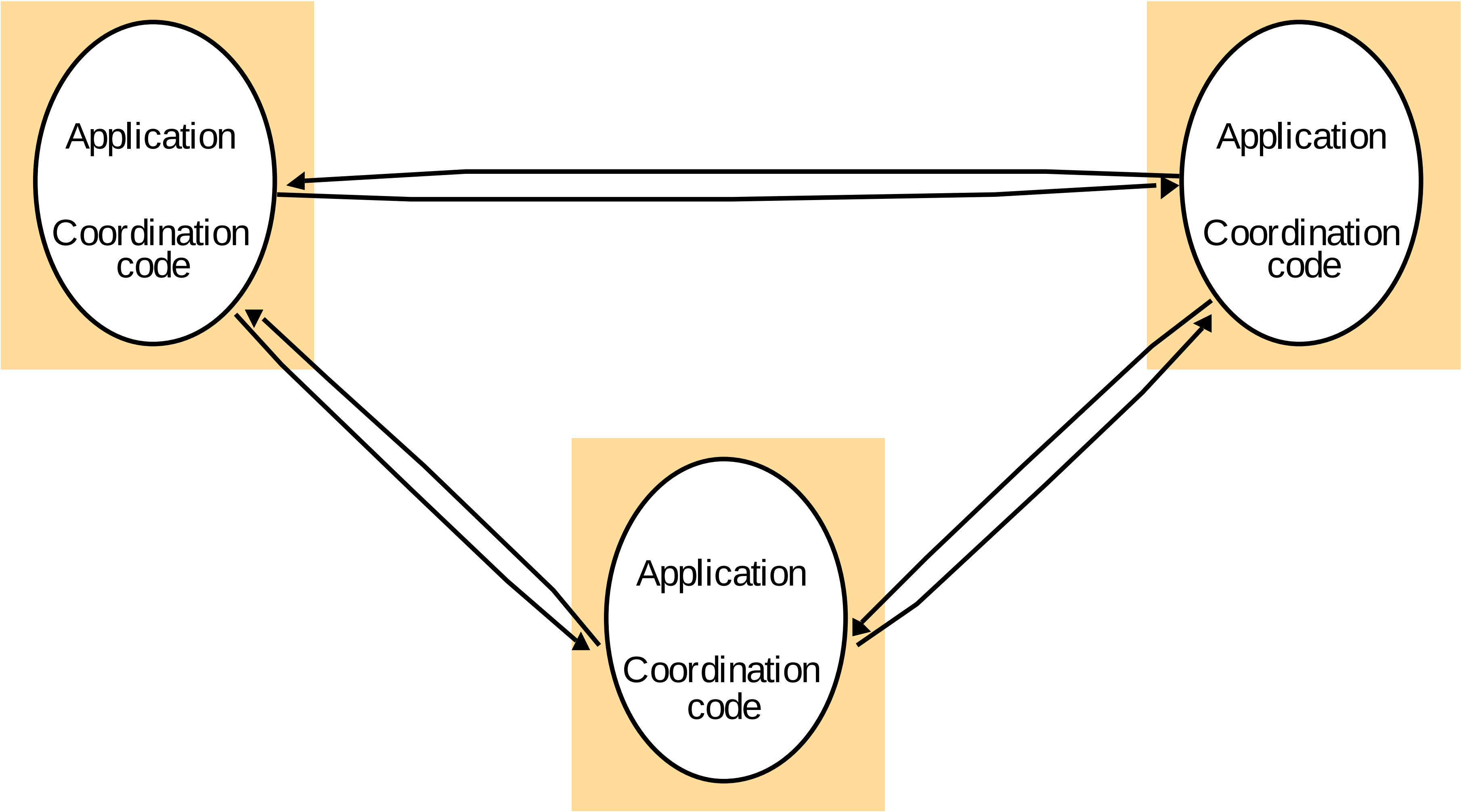
From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms
© Prentice-Hall, Inc. 2002

Communication in Multitier Systems



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

Peer-to-Peer Systems



From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Decentralized Architectures

Observation

In the last couple of years we have been seeing a tremendous growth in peer-to-peer systems.

- Structured P2P: nodes are organized following a specific distributed data structure
- Unstructured P2P: nodes have randomly selected neighbors
- Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

Note

In virtually all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes (cf. application-level multicasting)

Unstructured P2P Systems

Observation

Many unstructured P2P systems attempt to maintain a random graph.

Basic principle

Each node is required to contact a randomly selected other node:

- Let each peer maintain a partial view of the network, consisting of c other nodes
- Each node P periodically selects a node Q from its partial view
- P and Q exchange information **and** exchange members from their respective partial views

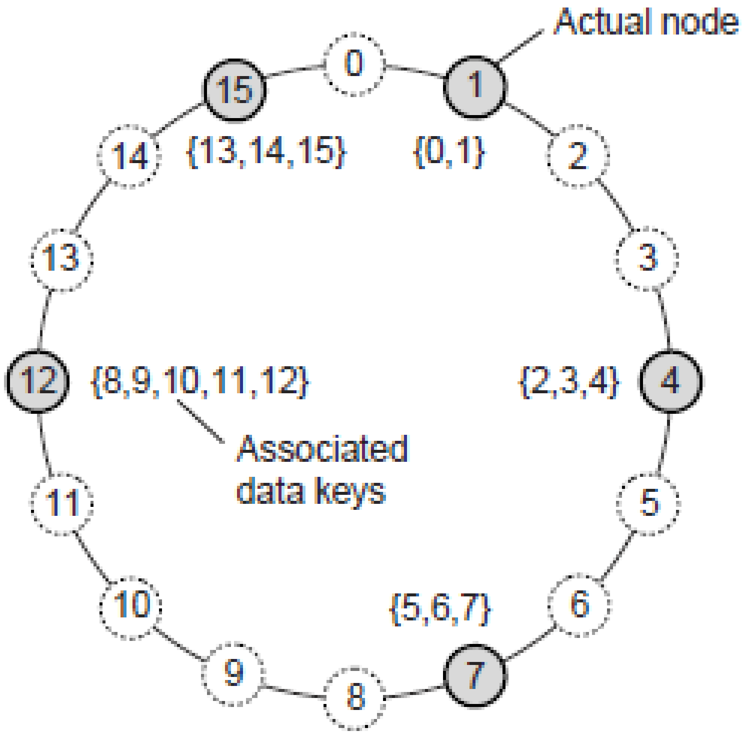
Note

It turns out that, depending on the exchange, randomness, but also robustness of the network can be maintained.

Structured P2P Systems

Basic idea

Organize the nodes in a structured overlay network such as a logical ring, and make specific nodes responsible for services based only on their ID.



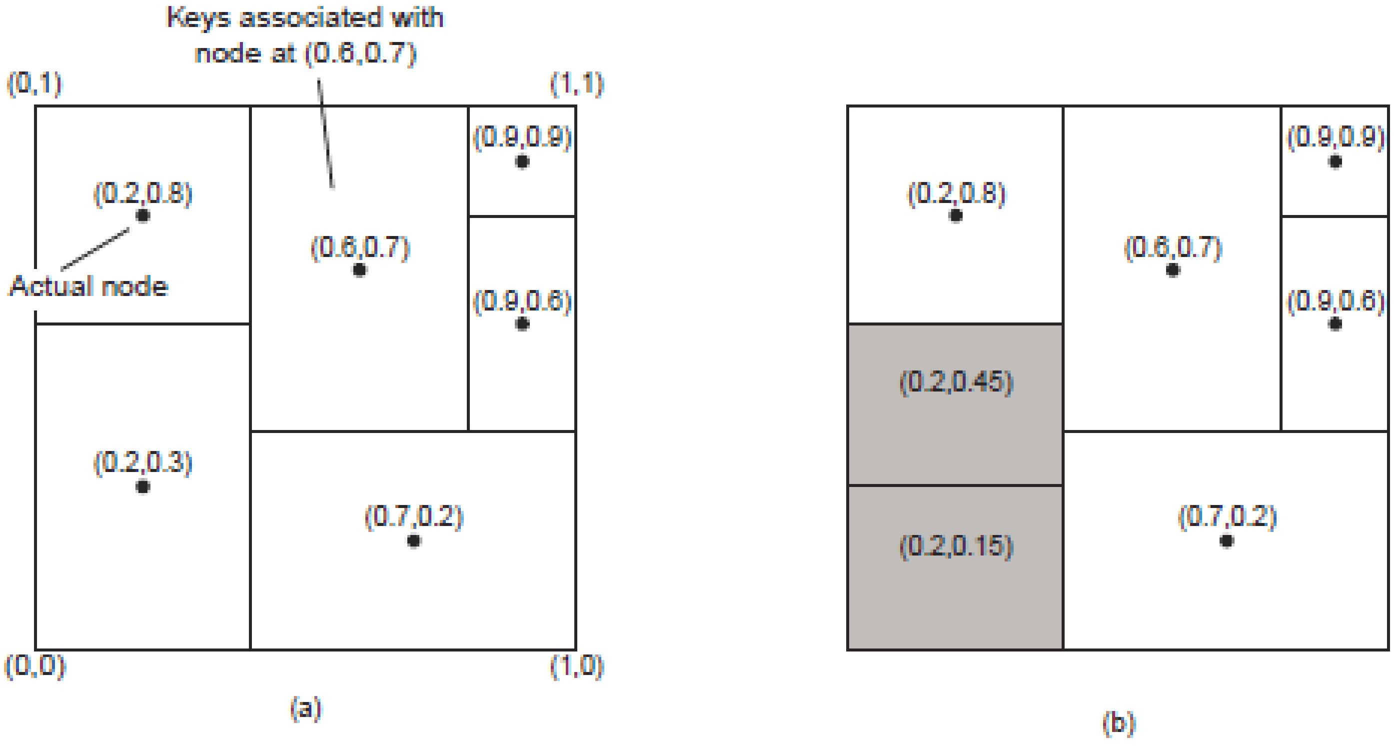
Note

The system provides an operation *LOOKUP(key)* that will efficiently route the lookup request to the associated node.

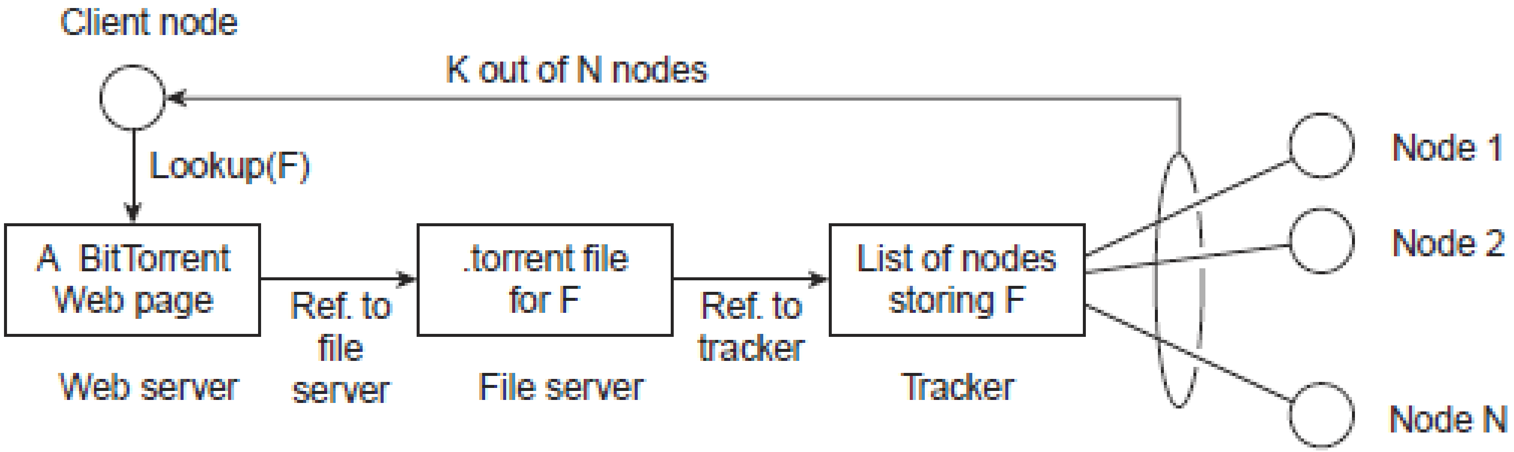
Structured P2P Systems

Other example

Organize nodes in a *d*-dimensional space and let every node take the responsibility for data in a specific region. When a node joins ⇒ split a region.



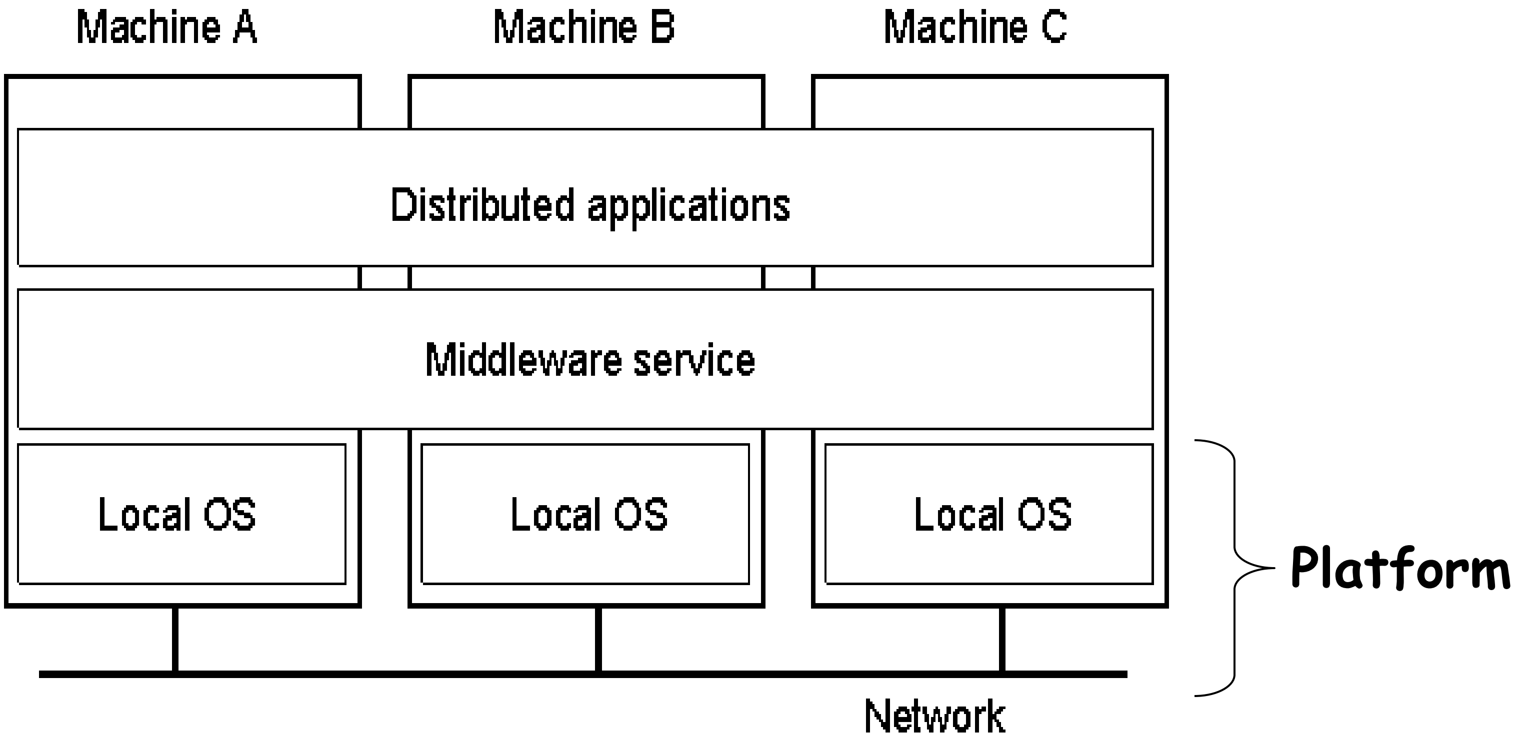
Hybrid Architectures: C/S with P2P – BitTorrent



Basic idea

Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.

Software Layers



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

Layers

- Platform
  - Fundamental communication and resource management services
  - We won’t be worried about these
- Middleware
  - Provides a service layer that hides the details and heterogeneity of the underlying platform
  - Provides an “easier” API for the applications and services
  - RPC, RMI, CORBA, etc.
- Applications
  - Distributed applications, services
  - Examples: e-mail, ftp, etc

Example Client/Server Middleware

- Remote Procedure Call (RPC)
  - Uses the well-known procedure call semantics.
  - The caller makes a procedure call and then waits. If it is a local procedure call, then it is handled normally; if it is a remote procedure, then it is handled as a remote procedure call.
  - Caller semantics is blocked send; callee semantics is blocked receive to get the parameters and a nonblocked send at the end to transmit results.