Fundamentals of Distributed Systems II

Instructor: Dmitri A. Gusev

Fall 2007

CS 502: Computers and Communications Technology

Lecture 15, October 29, 2007

Reading

- http://www.cs.vu.nl/~ast/books/ds1/01.pdf

Andrew S. Tanenbaum, Maarten van Steen, *Distributed Systems: Principles and Paradigms,* Section 1.4

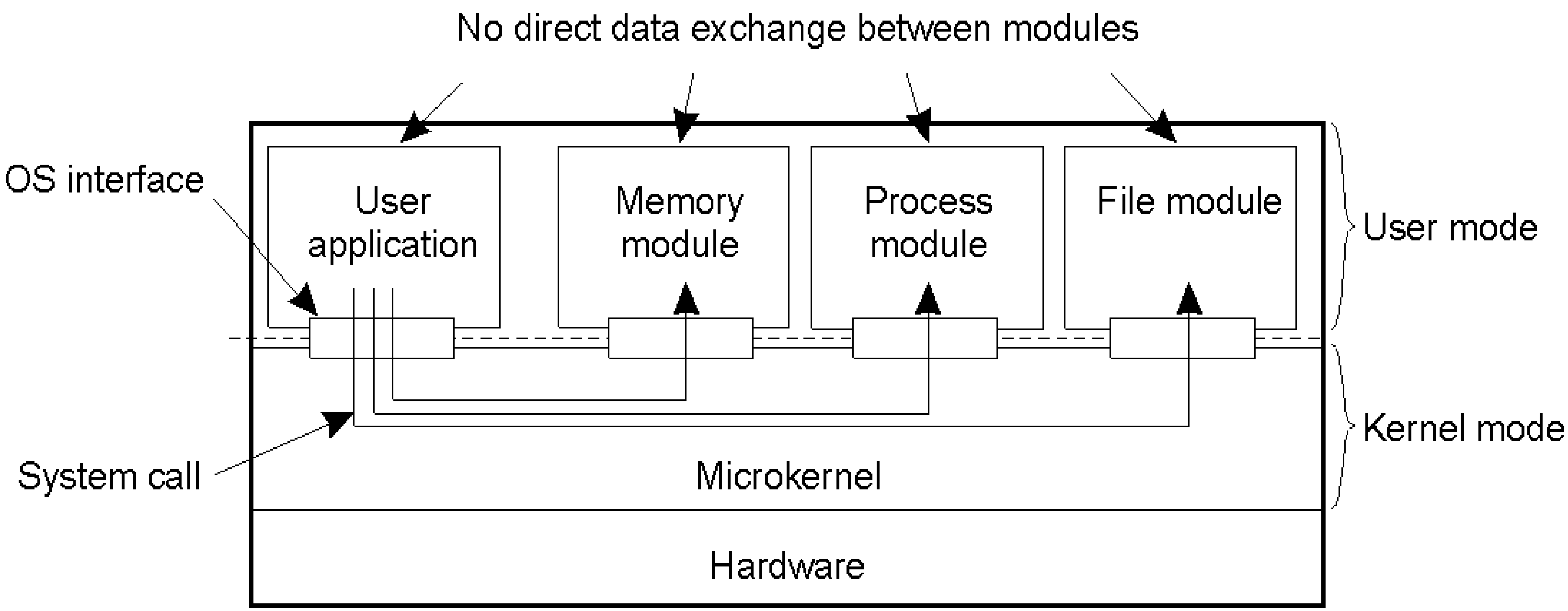
Software Concepts

System	**Description**	**Main Goal**
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

- An overview of
- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

Uniprocessor Operating Systems

- Separating applications from operating system code through a microkernel.



Multiprocessor Operating Systems (1)

- A monitor to protect an integer against concurrent access.

```
monitor Counter {
private:
 int count = 0;
public:
  int value() { return count;}
  void incr () { count = count + 1;}
  void decr() { count = count – 1;}
}
```

Multiprocessor Operating Systems (2)

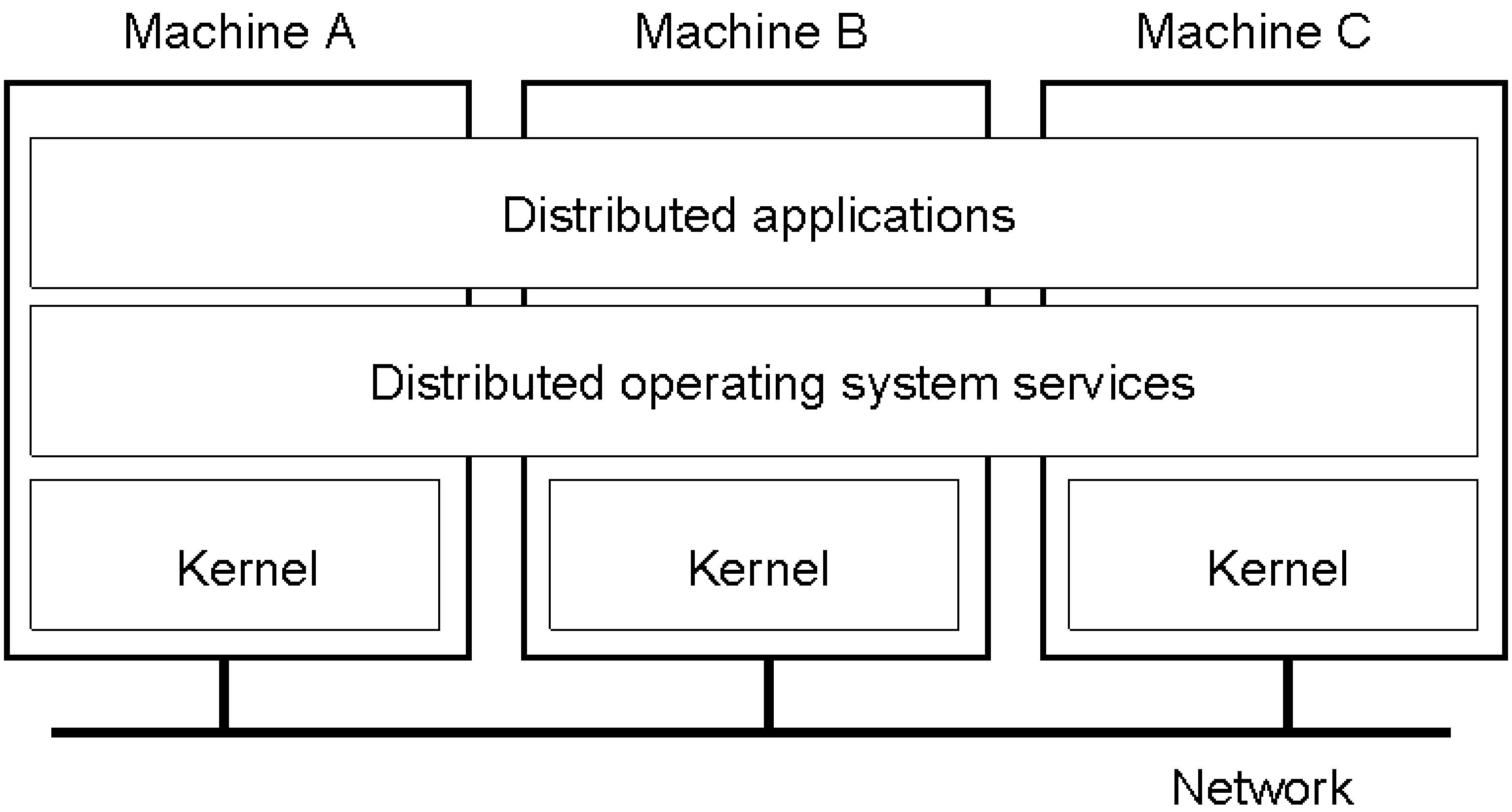
```
monitor Counter {
private:
  int count = 0;
  int blocked_procs = 0;
  condition unblocked;
public:
  int value () { return count;}
  void incr () {
       if (blocked_procs == 0)
         count = count + 1;
       else
         signal (unblocked);
}
```

```
void decr() {
if (count ==0) {
  blocked_procs = blocked_procs + 1;
  wait (unblocked);
  blocked_procs = blocked_procs – 1;
   }
  else
   count = count – 1;
 }
}
```

- A monitor to protect an integer against concurrent access by blocking a process.

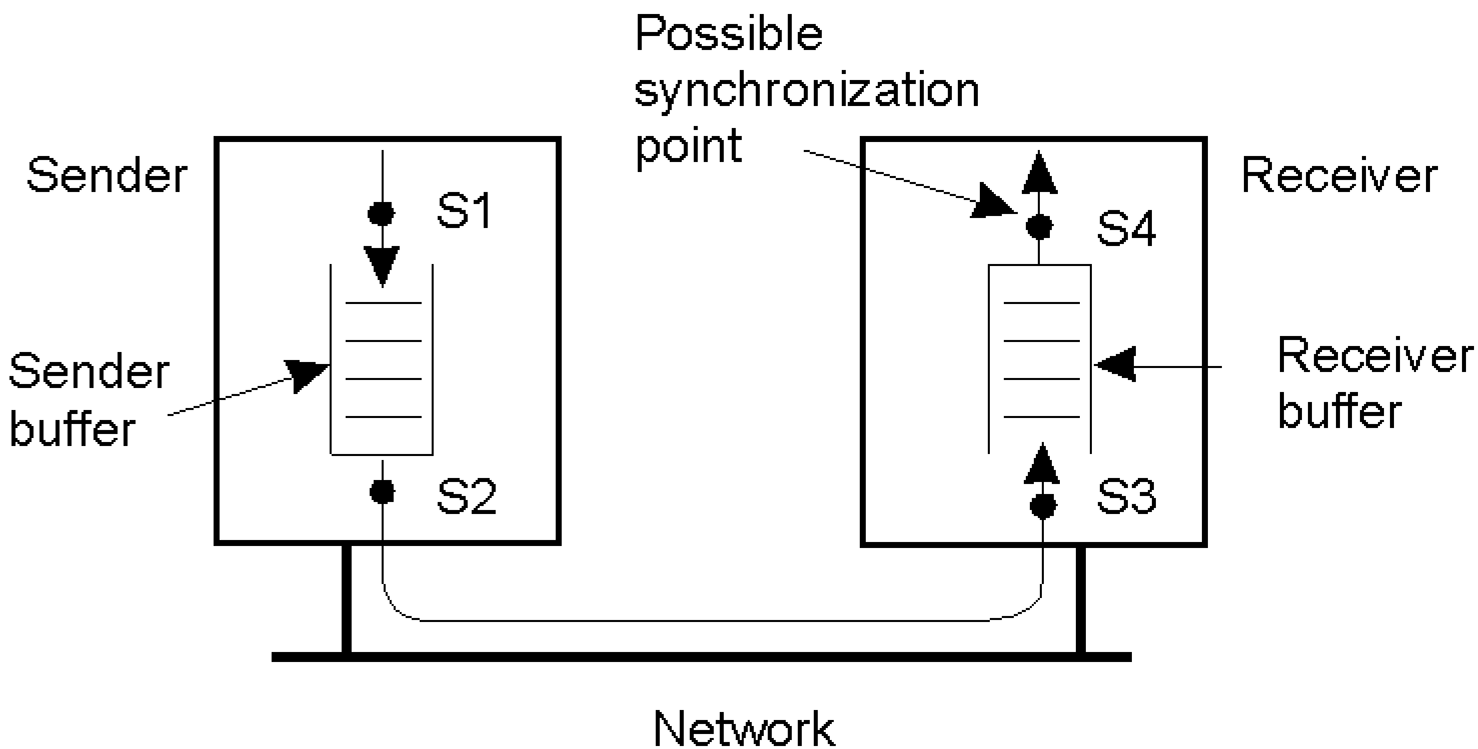
Multicomputer Operating Systems (1)

- General structure of a multicomputer operating system



Multicomputer Operating Systems (2)

- Alternatives for blocking and buffering in message passing.



Multicomputer Operating Systems (3)

Synchronization point	**Send buffer**	**Reliable comm. guaranteed?**
Block sender until buffer not full	Yes	Not necessary
Block sender until message sent	No	Not necessary
Block sender until message received	No	Necessary
Block sender until message delivered	No	Necessary

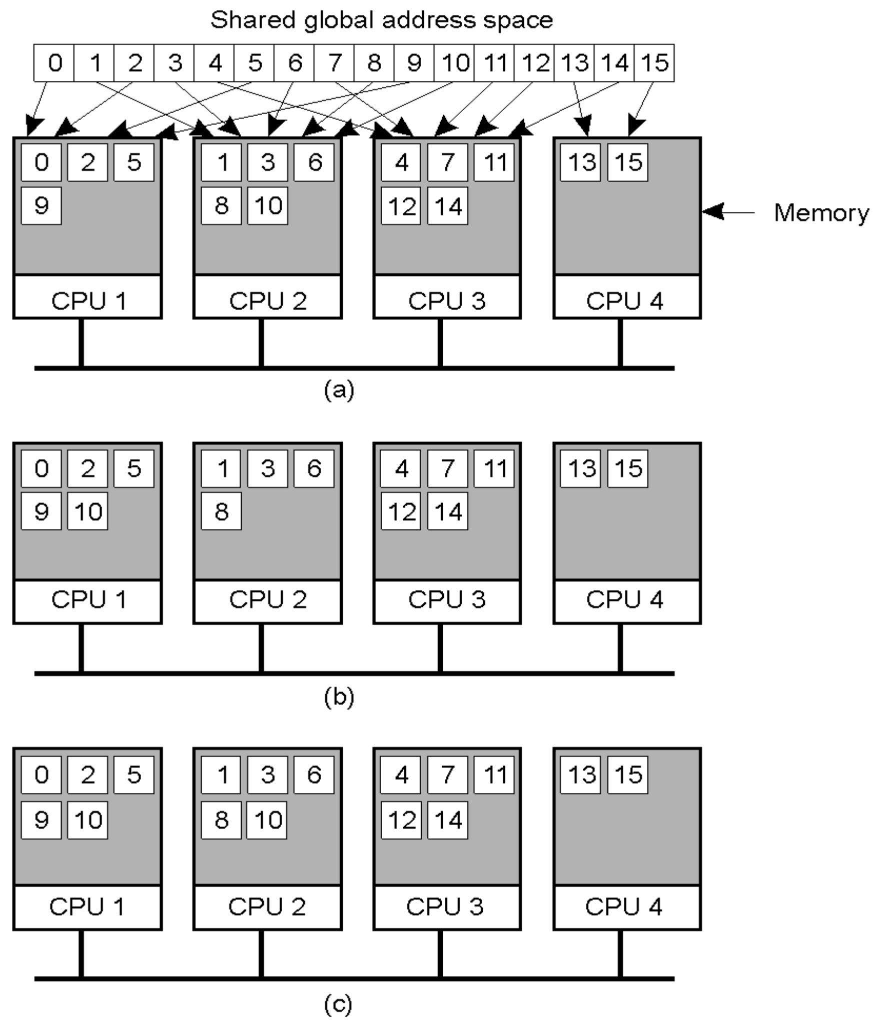
- Relation between blocking, buffering, and reliable communications.

Distributed Shared Memory Systems (1)

a) Pages of address space distributed among four machines

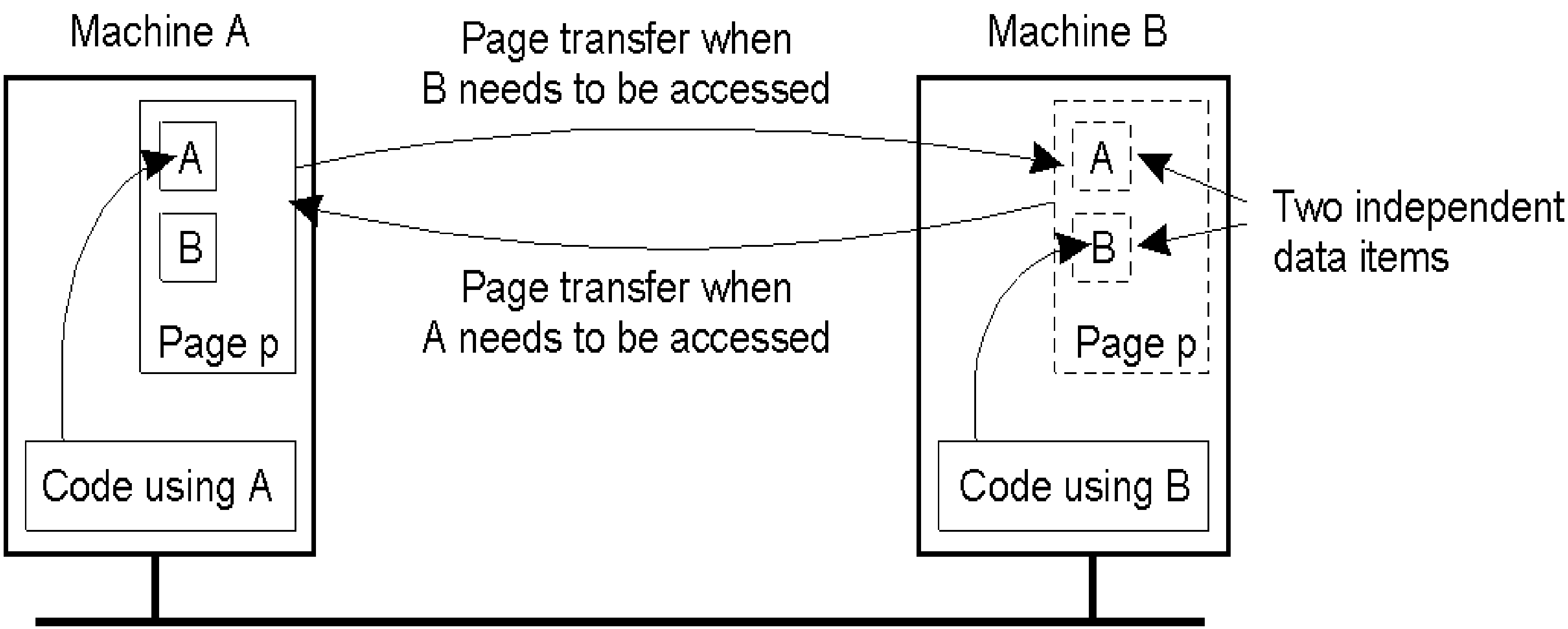
b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used



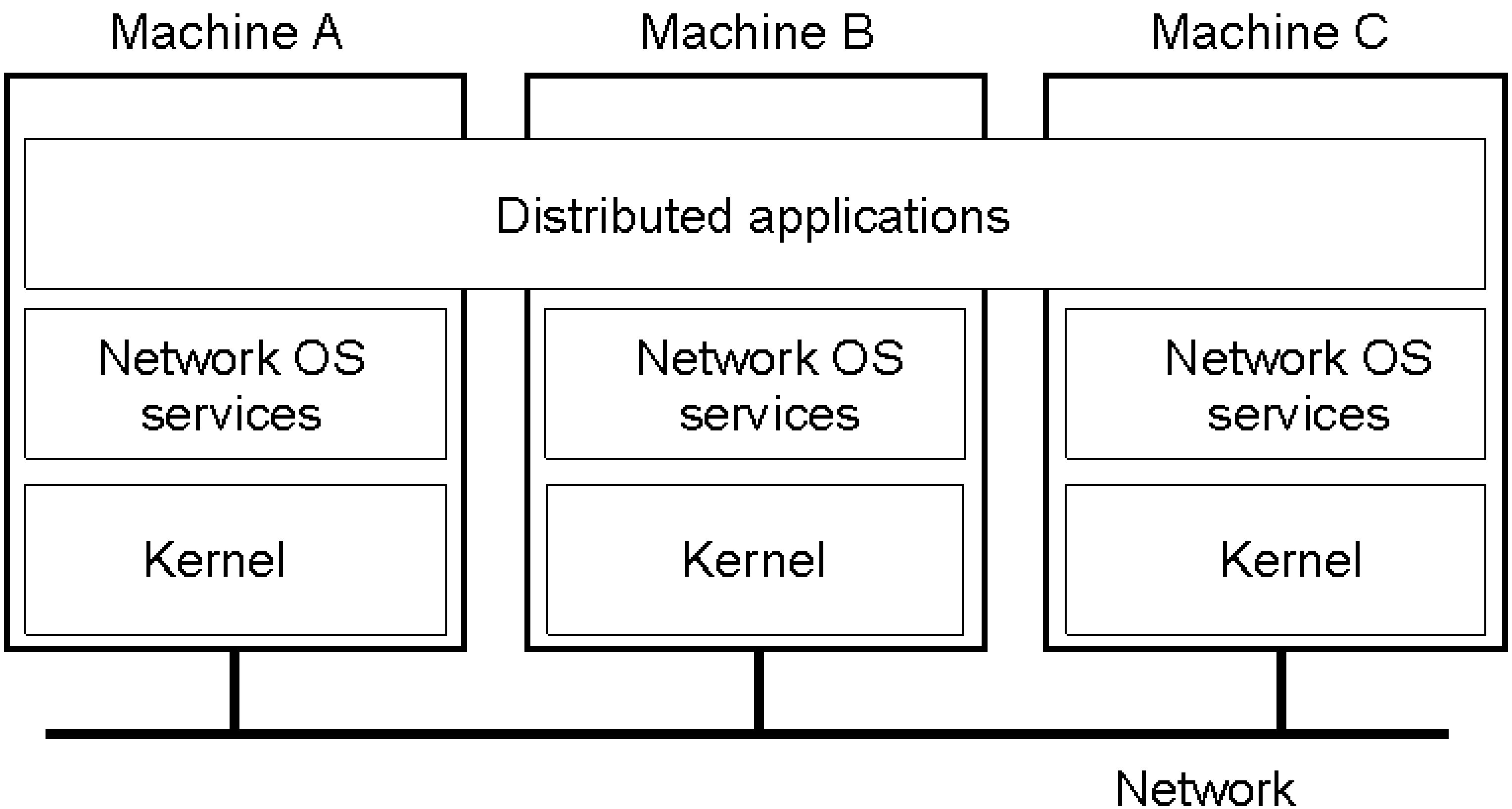
Distributed Shared Memory Systems (2)

- False sharing of a page between two independent processes.



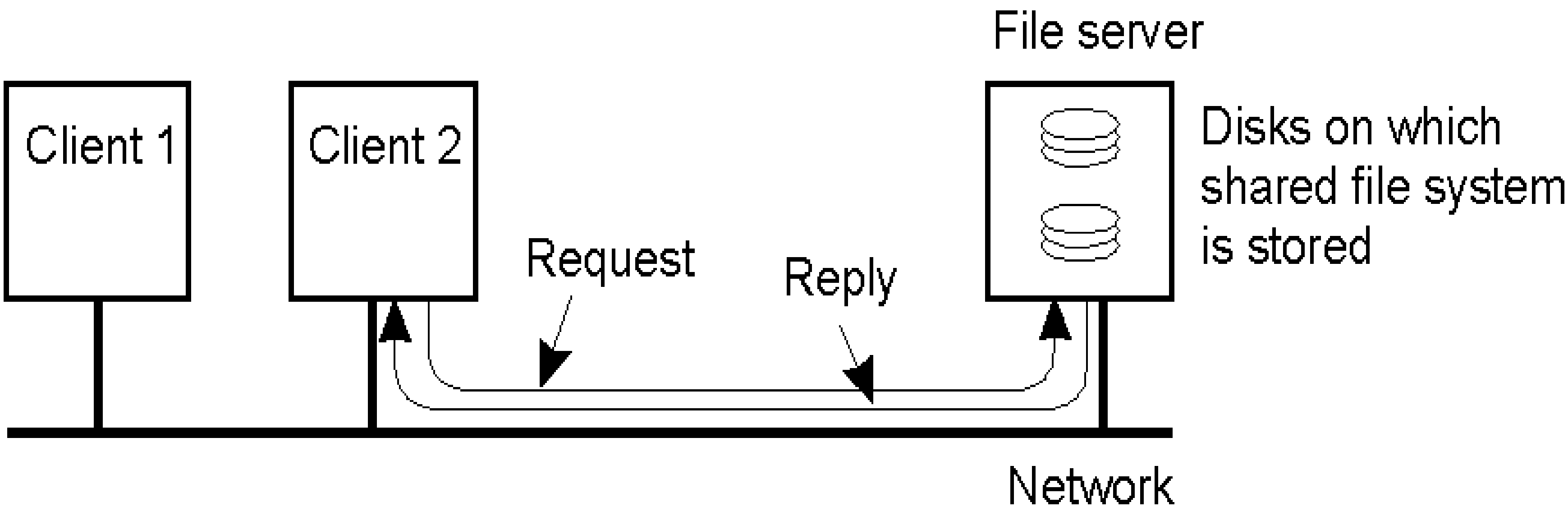
Network Operating System (1)

- General structure of a network operating system.



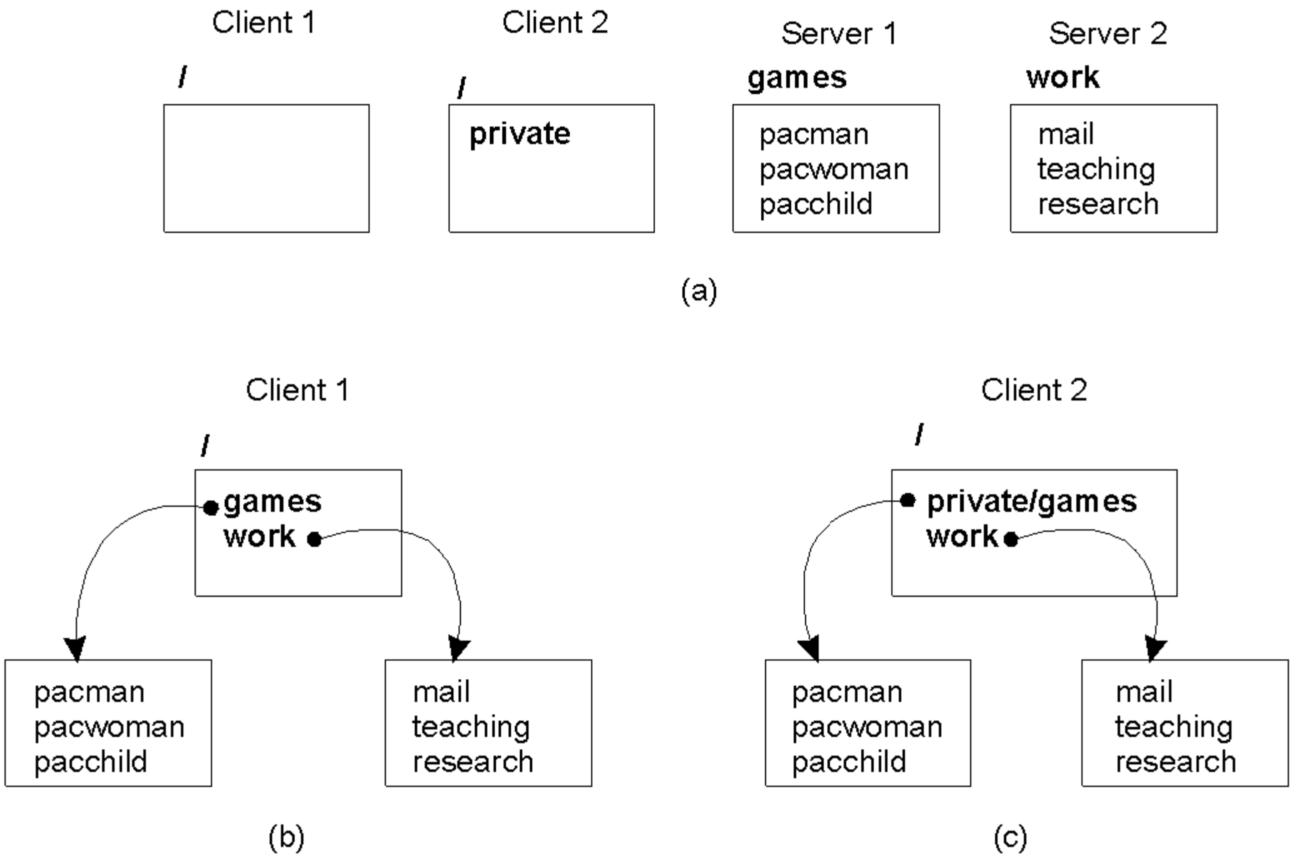
Network Operating System (2)

- Two clients and a server in a network operating system.



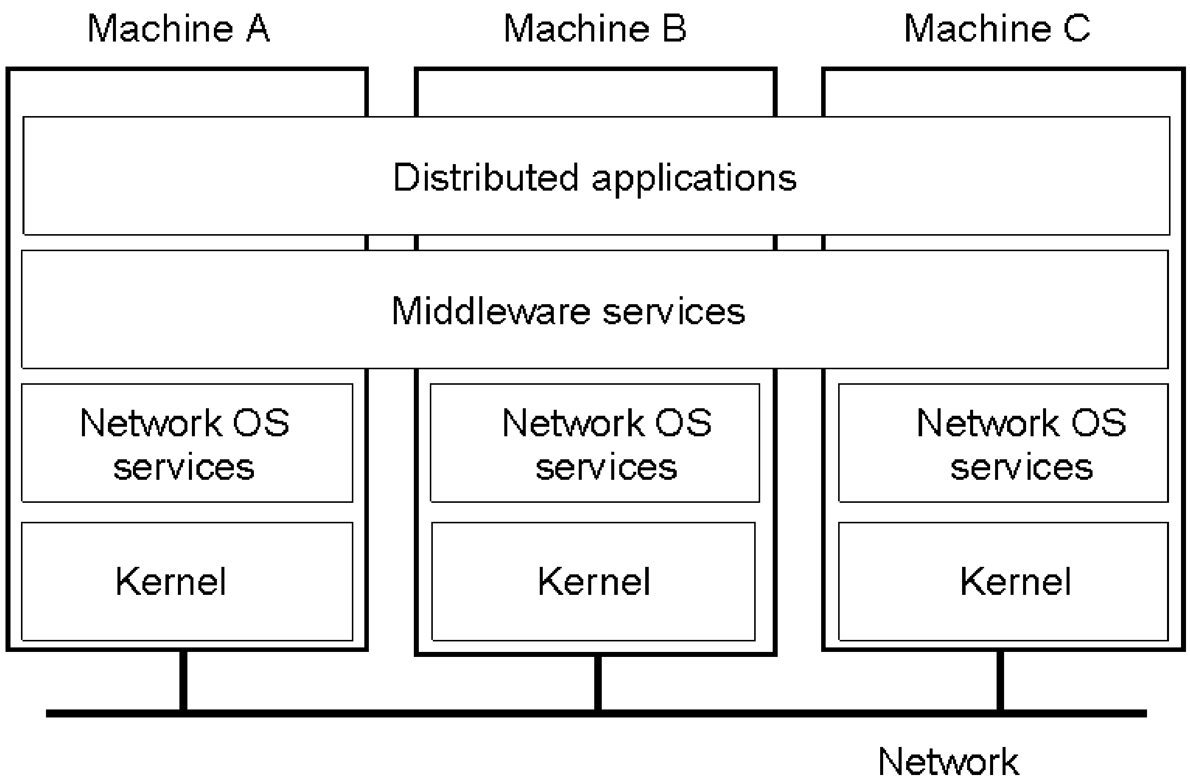
Network Operating System (3)

- Different clients may mount the servers in different places.

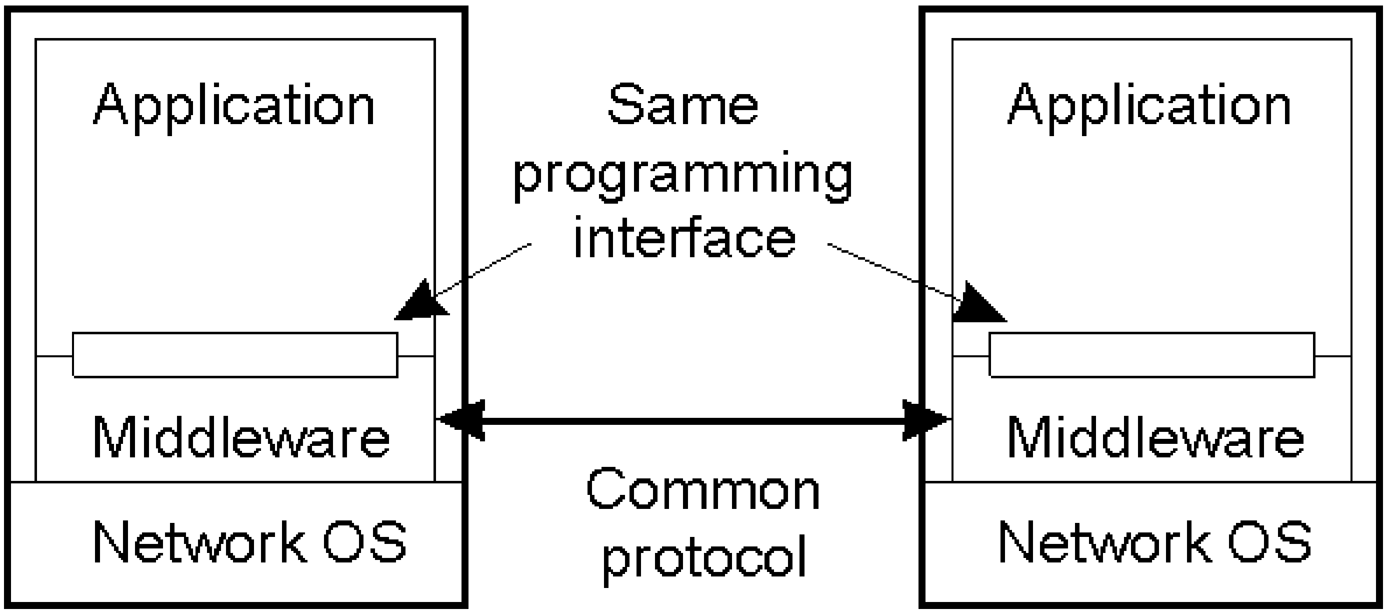


Positioning Middleware

- General structure of a distributed system as middleware.



Middleware and Openness



- In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.

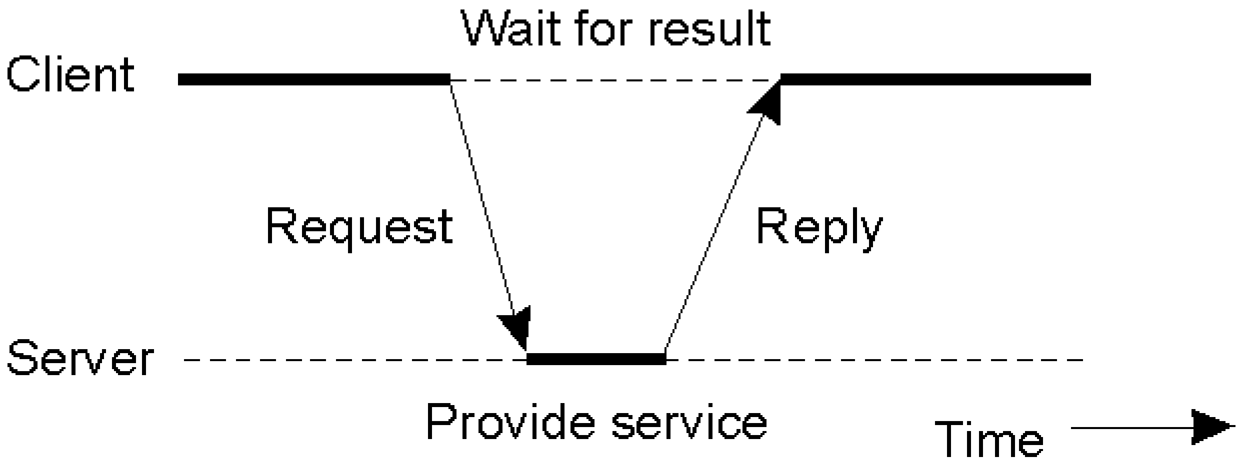
Comparison between Systems

- A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems.

Item	**Distributed OS**		**Network OS**	**Middleware-based OS**
	Multiproc.	**Multicomp.**		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Clients and Servers

- General interaction between a client and a 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         */
};
```

An Example Client and Server (2)

```
#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                         */
    }
}
```

- A sample server.

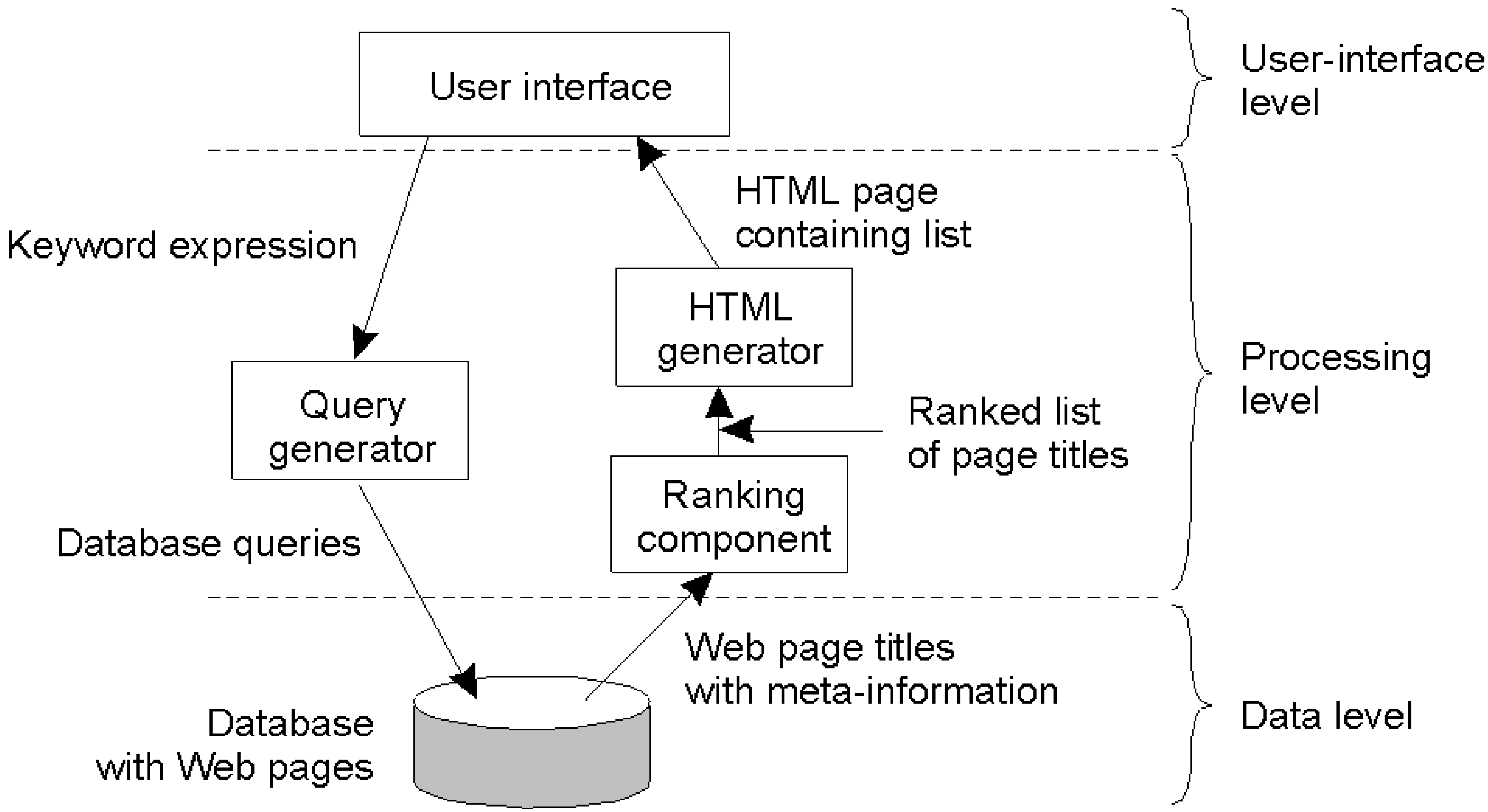
An Example Client and Server (3)

```
#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                    */
}
```

- A client using the server to copy a file.

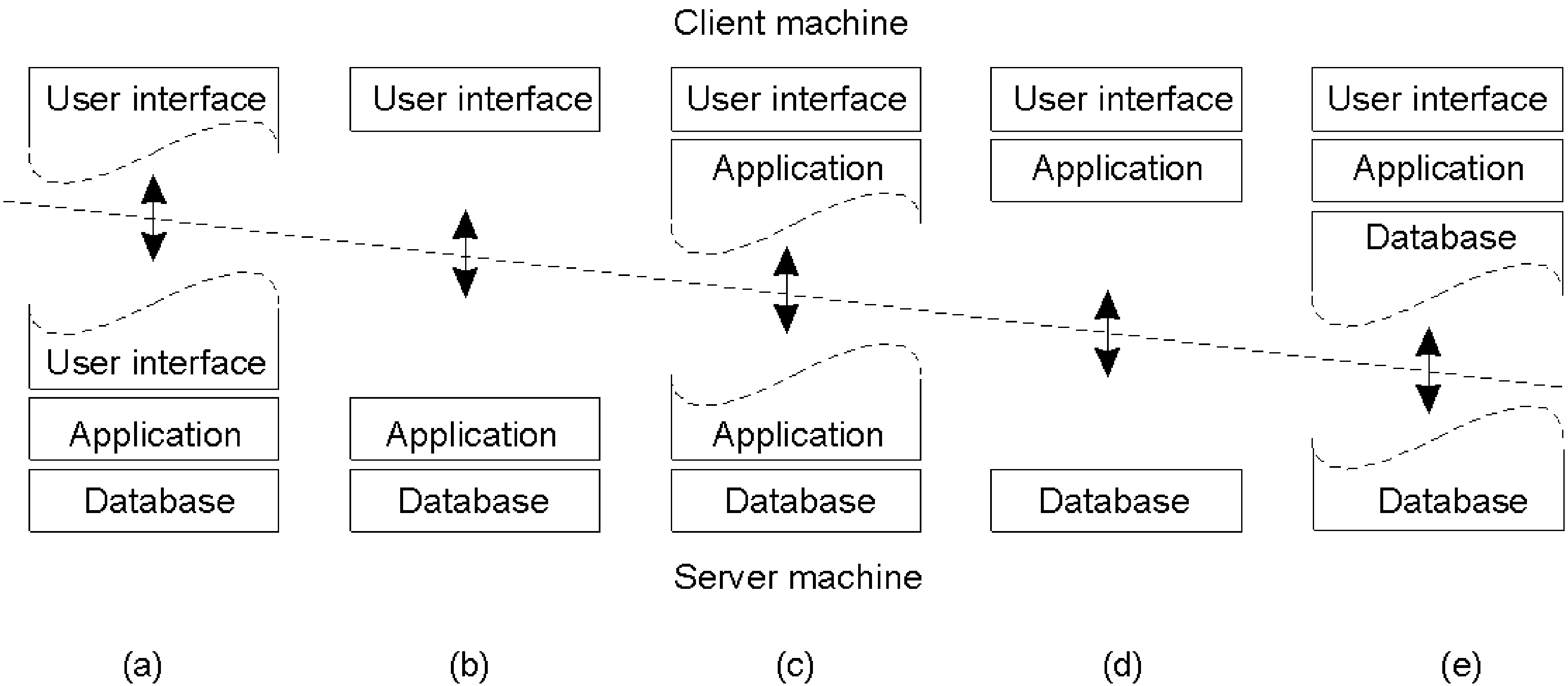
Processing Level

- The general organization of an Internet search engine into three different layers



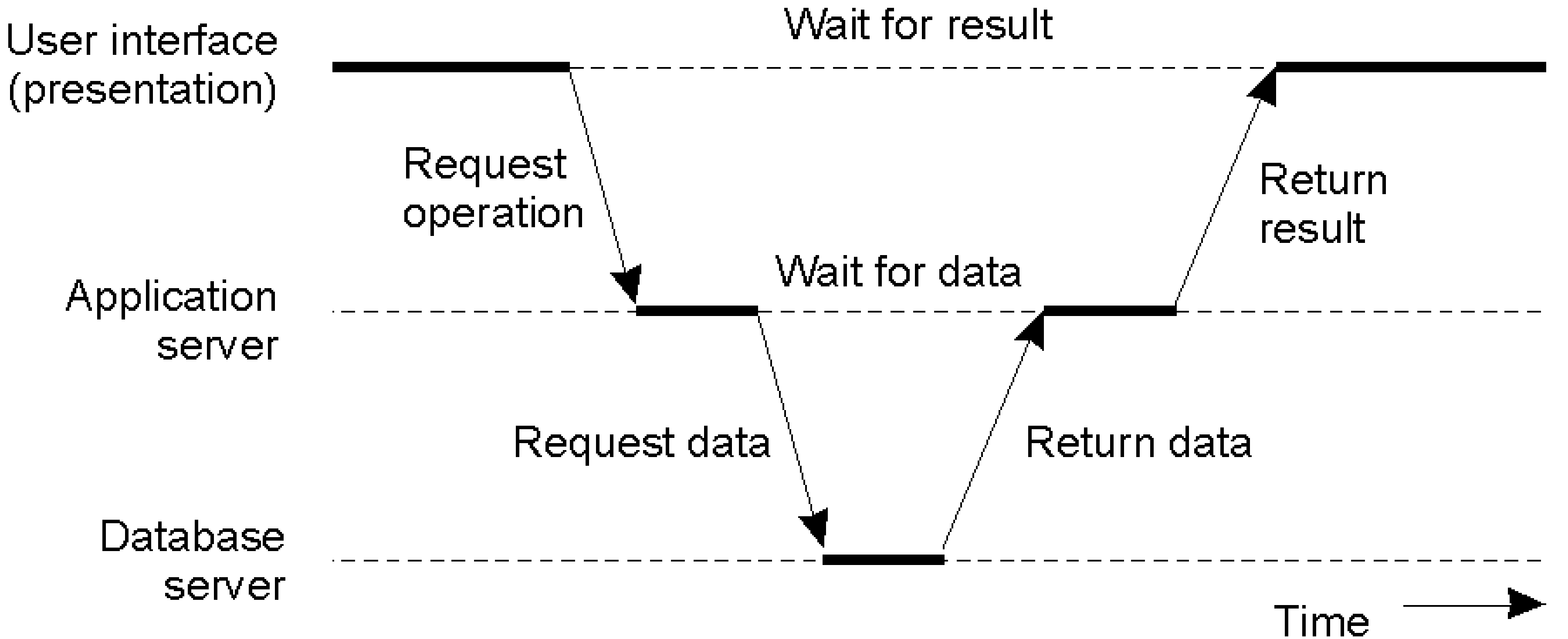
Multitiered Architectures (1)

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



Multitiered Architectures (2)

- An example of a server acting as a client.



Modern Architectures

- An example of horizontal distribution of a Web service.