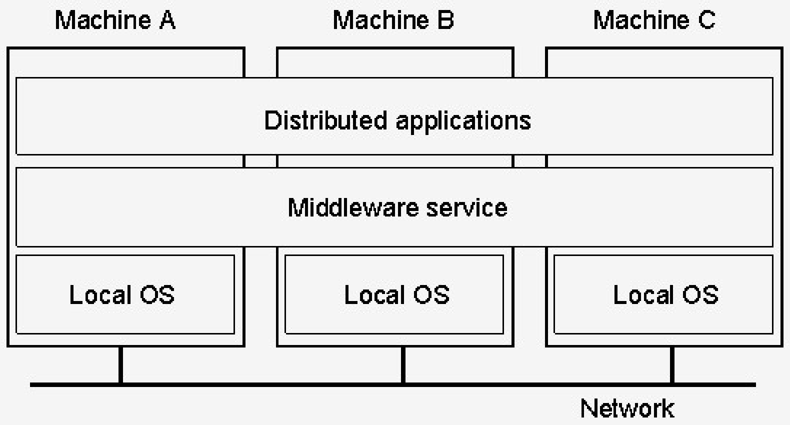
Distributed Computer Systems

CHAITANYA
WOMEN'S COLLEGE
DEGREE & PG
AFFILIATED TO ANDHRA UNIVERSITY

The Rise of Distributed Systems

- Computer hardware prices falling, power increasing
  - If cars the same, Rolls Royce would cost 1 dollar and get 1 billion miles per gallon (with 200 page manual to open the door)
- Network connectivity increasing
  - Everyone is connected with fat pipes
- It is *easy* to connect hardware together
- Definition: a *distributed system* is
  - A collection of independent computers that appears to its users as a single coherent

Definition of a Distributed System



Examples:
- The Web
- Processor Pool
- Airline Reservation

A distributed system organized as middleware.

Users can interact with the system in a consistent way, regardless of where the interaction takes place

Transparency in a Distributed System

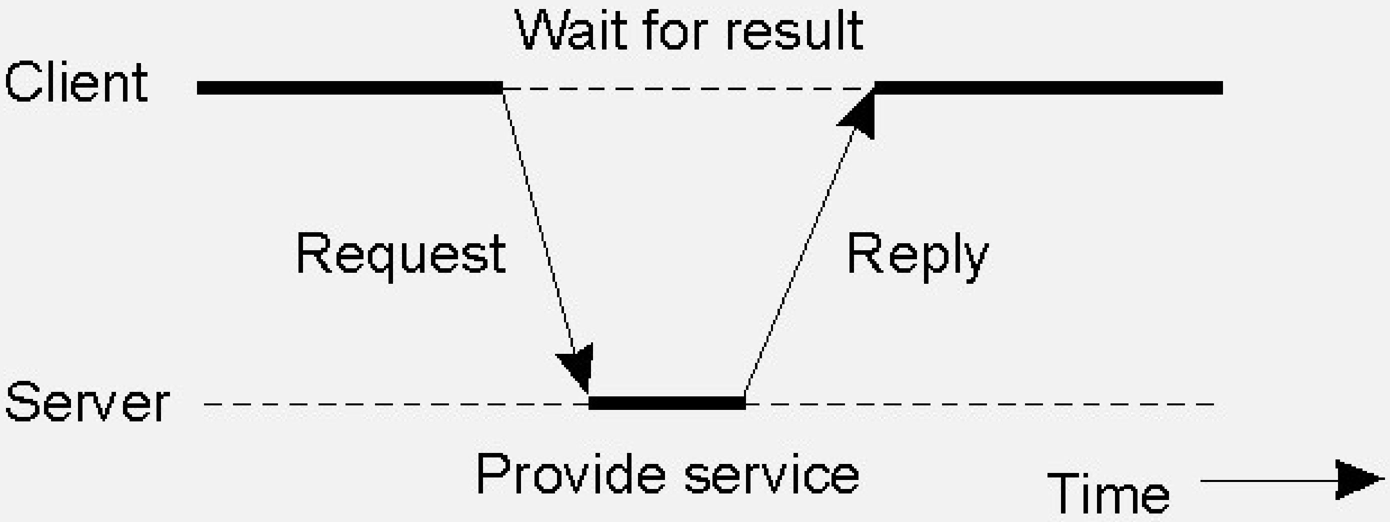
Transparency	**Description**
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Comparison between Systems

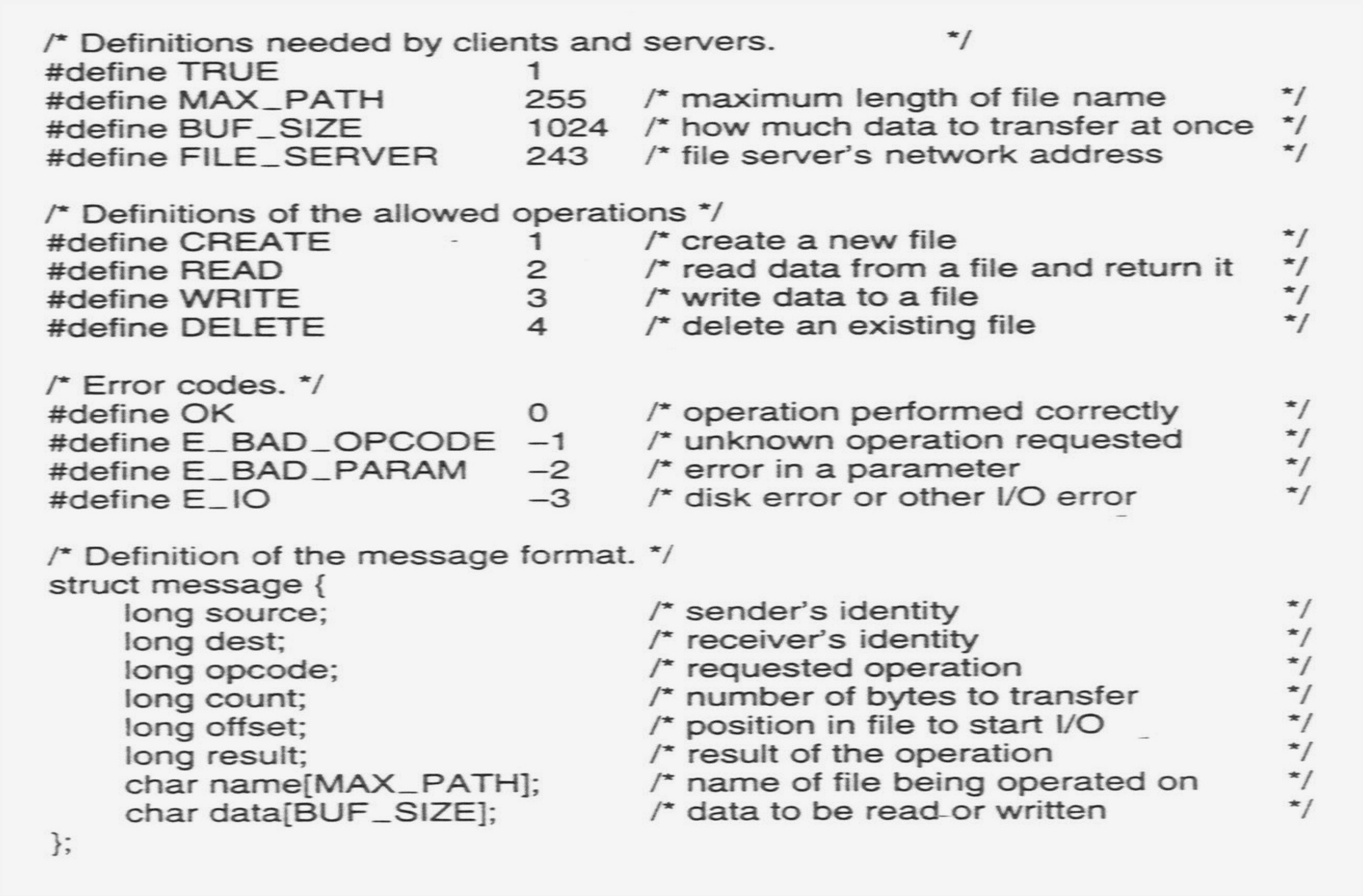
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global,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

Clients and Servers

- Thus far, have not talked about organization of *processes*
  - Again, many choices but most agree upon client-server
- If can do so without connection, quite simple
  - If underlying connection is unreliable, not trivial
  - Resend? What if receive twice
- Use TCP for reliable connection (apps on Internet)
  - Not always appropriate for high-speed LAN connection (4513)

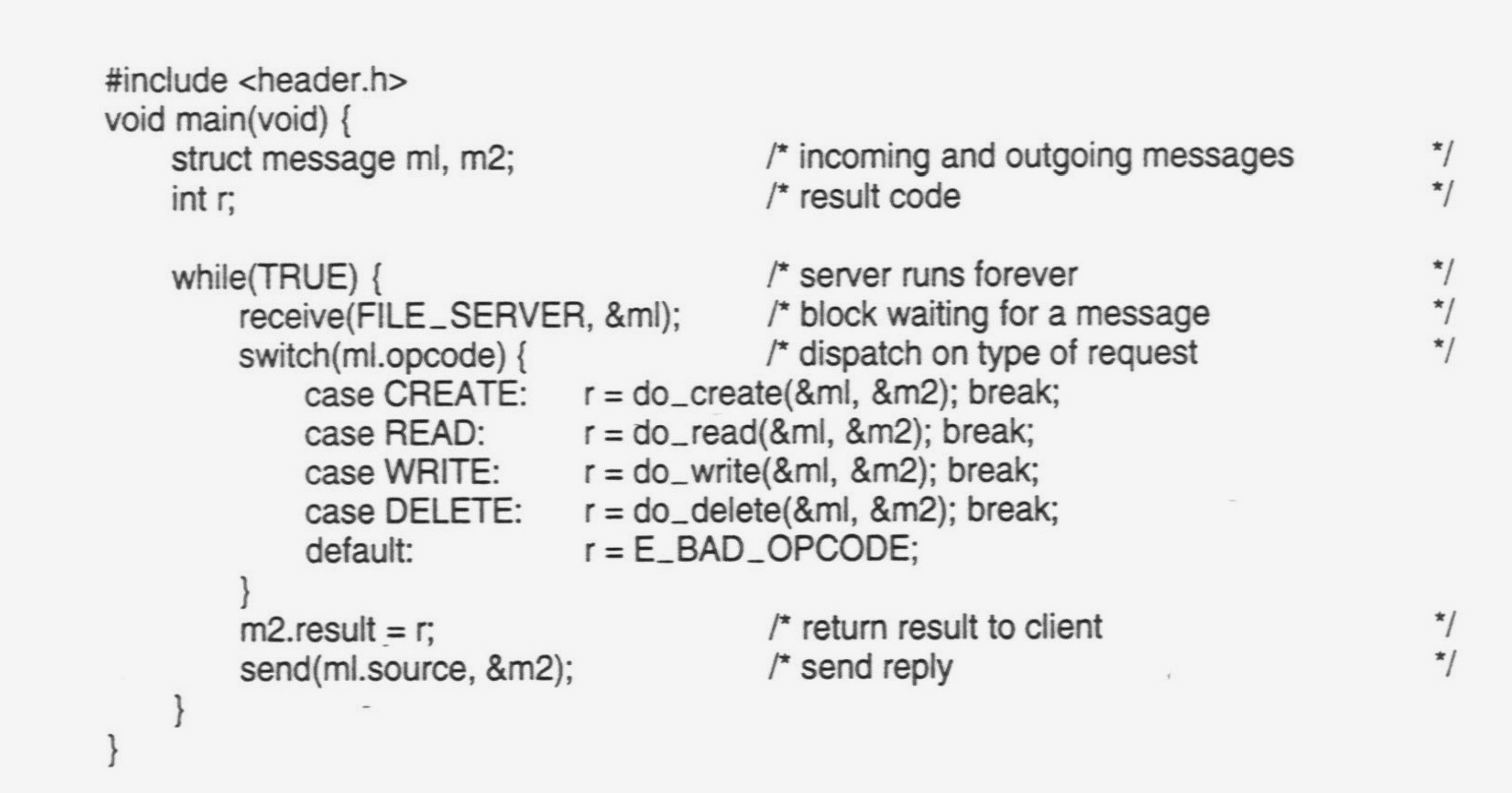

Wait for result
Client
Request
Reply
Server
Provide service
Time

Example Client and Server: Header



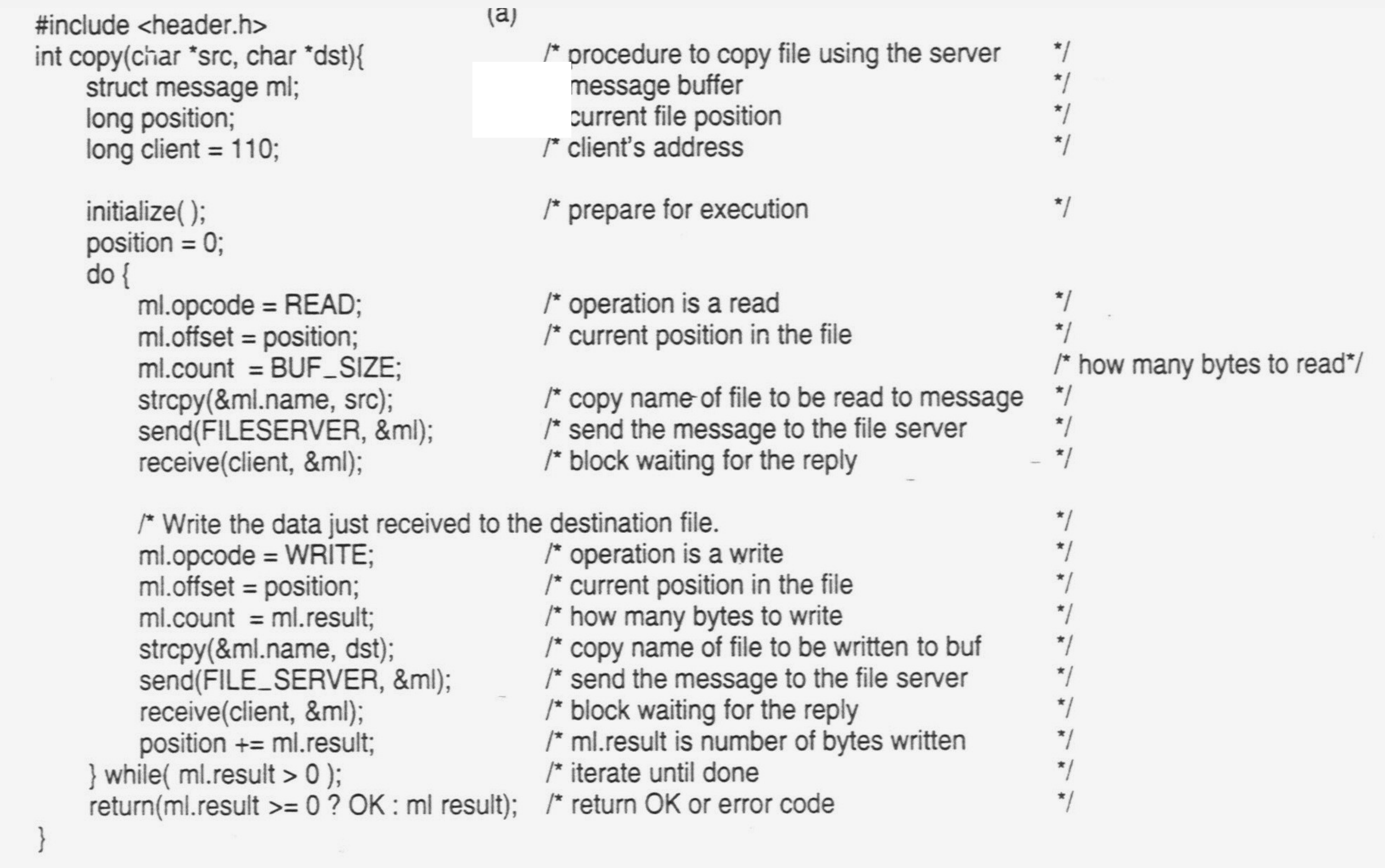
```
/* Definitions needed by clients and servers.                    */
#define TRUE                  1
#define MAX_PATH              255    /* maximum length of file name          */
#define BUF_SIZE              1024   /* how much data to transfer at once    */
#define FILE_SERVER           243   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

Example Client and Server: 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                         */
    }
}
```

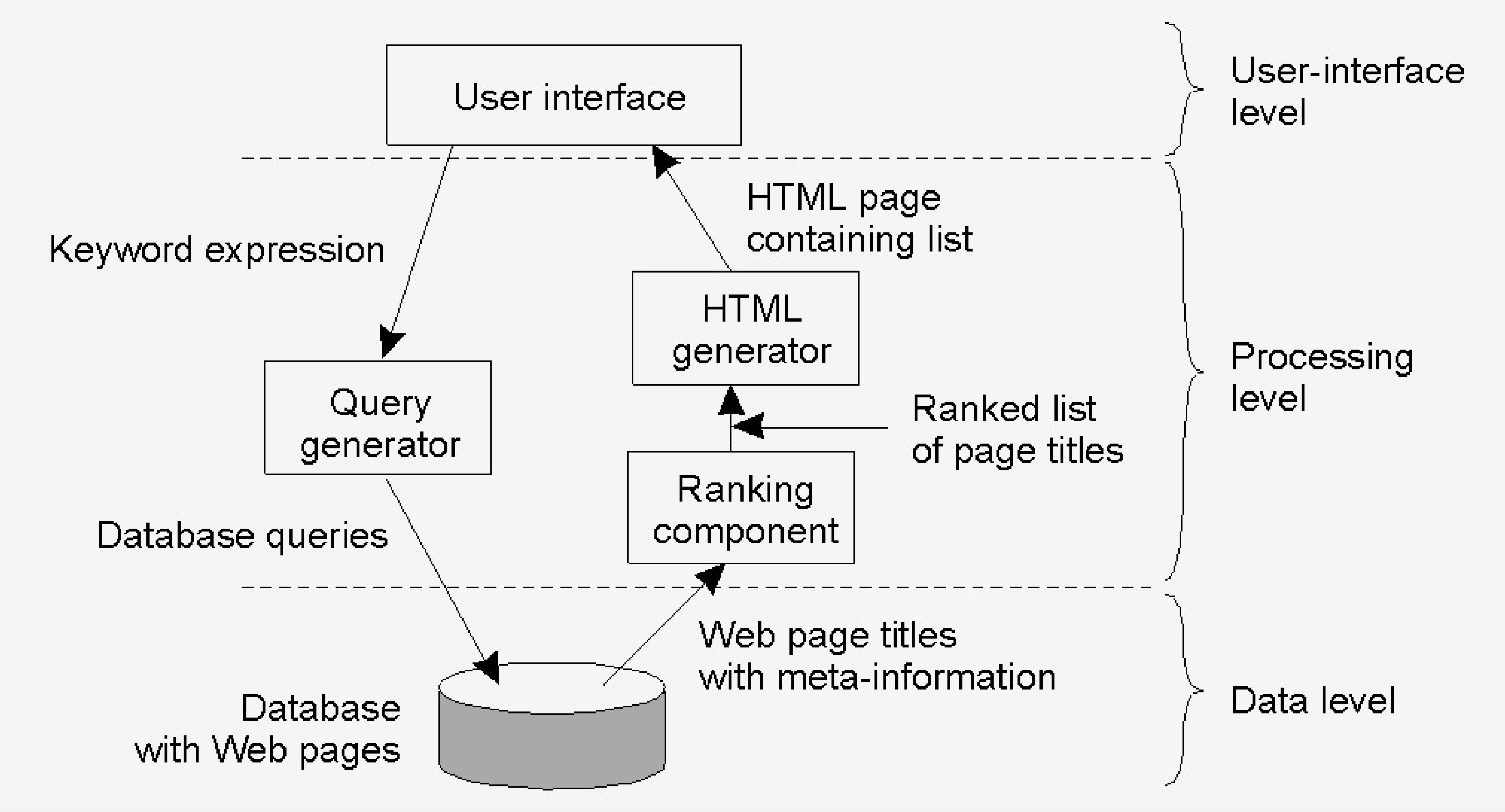
Example Client and Server: Client



(a)

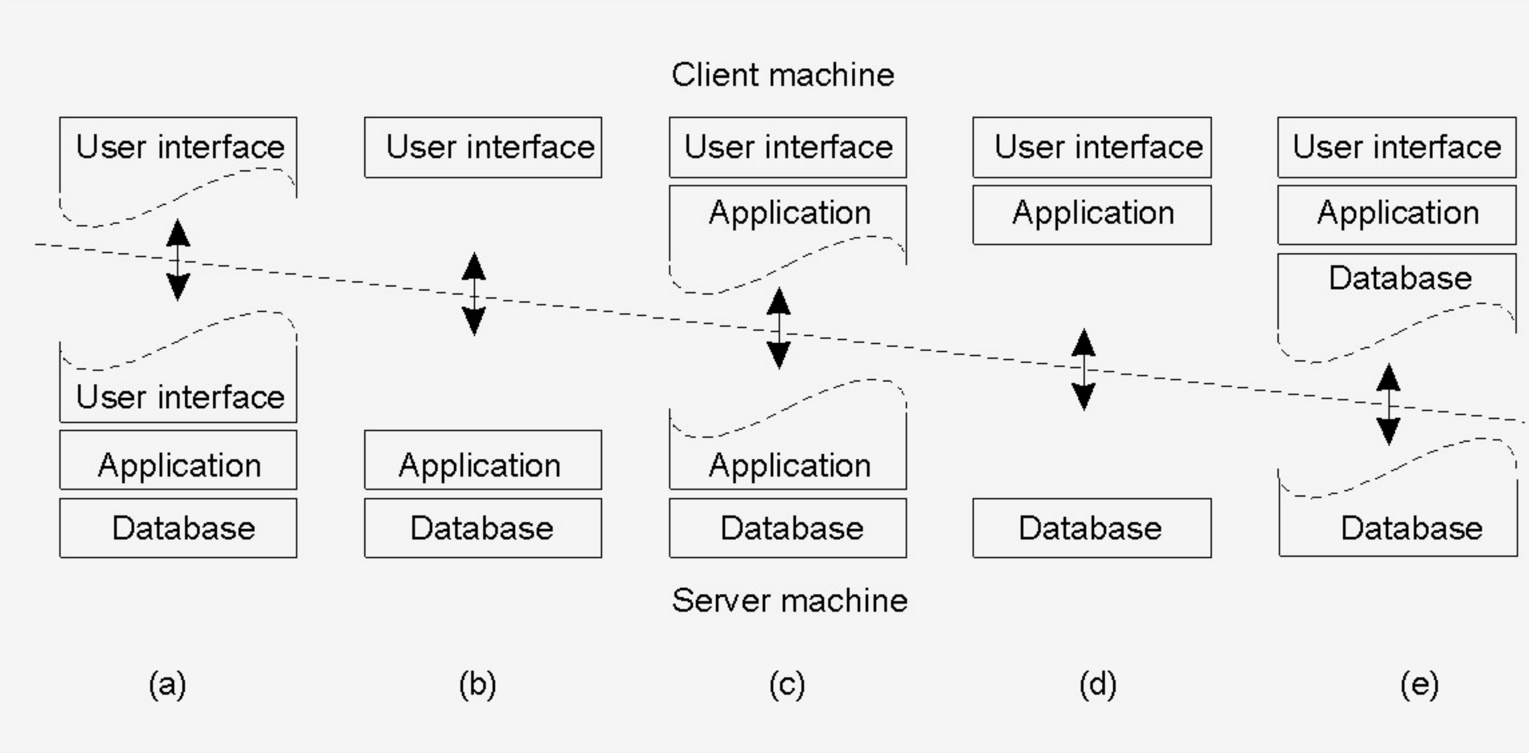
```
#include <header.h>
int copy(char *src, char *dst){                 /* procedure to copy file using the server       */
      struct message ml;                           message buffer                                 */
      long position;                               current file position   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

Client-Server Implementation Levels

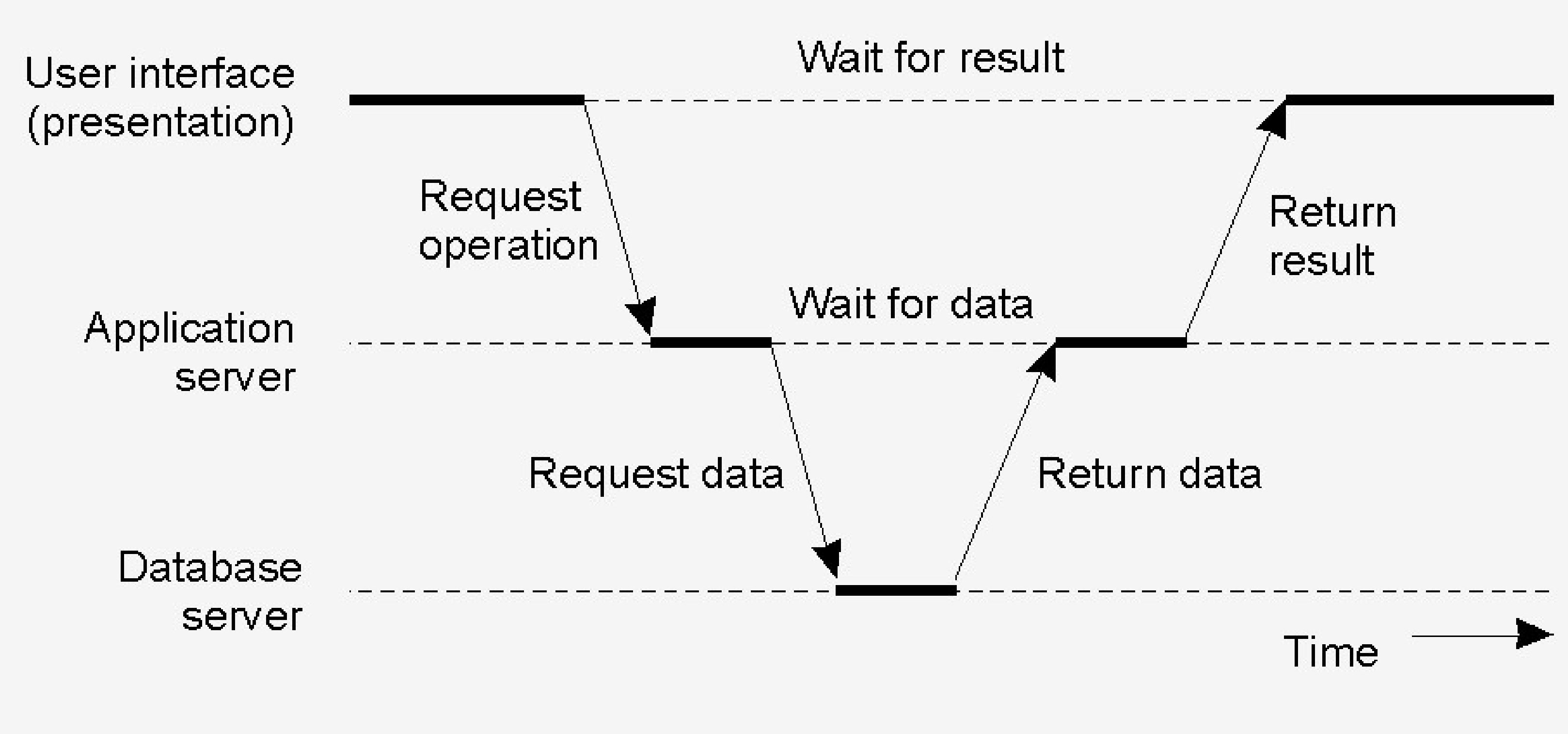


- Example of an Internet search engine
  - UI on client
  - Processing can be on client or server
  - Data level is server, keeps consistency

Multitiered Architectures



Multitiered Architectures: 3 tiers


Wait for result
User interface
(presentation)
Request
operation
Return
result
Wait for data
Application
server
Request data
Return data
Database
server
Time