Sistemas Distribuídos
– COM233 –

Prof. Tiago Garcia de Senna Carneiro

DECOM – UFOP

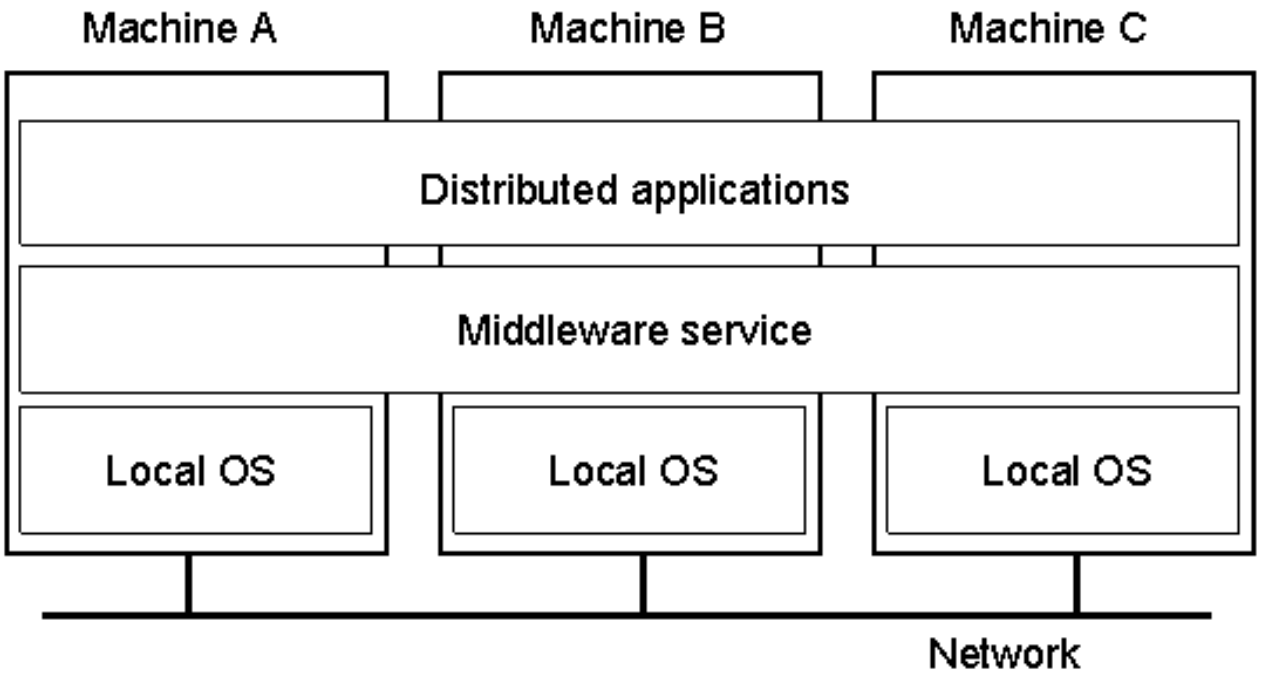
1º semestre de 2008

Definição de Sistema Distribuído

Um sistema distribuído é:

Um conjunto de computadores autônomos conectados por uma rede de comunicação que se apresentam para o usuário como se fossem um único computador.

Definição de Sistema Distribuído



Um sistema distribuído organizado como um *middleware*.

Note que o *middleware* se estende sobre múltiplas máquinas.

Transparência em Sistemas Distribuídos

Transparência	**Descrição**
Acesso	Esconde diferenças na **representação dos dados** e em como um **recurso**
Localização	Esconde **onde** um recurso está localizado
Migração	Esconde que um **recurso pode se mover** para outro local
Realocação	Esconde que um **recurso pode ser movido** para outro local mesmo quando em uso
Replicação	Esconde que um **recurso pode ser compartilhado** por vários usuários concorrentes
Concorrência	Esconde que um recurso pode ser compartilhado **por vários usuários concorrentes**
Falha	Esconde a **falha** e a **recuperação** de um recurso
Persistência	Esconde se um recurso (programa) está na memória ou em disco

Diferentes formas de transparência em sistemas distribuídos

Problemas de Escalabilidade

Conceitos	**Exemplos**
Servidores Centralizados	Um único servidor para vários usuários
Dados Centralizados	Um único catálogo de telefone on-line
Algoritmos Centralizados	Calcular rota com base na informação completa sobre o estado do sistema

Exemplos de limitações relacionadas à escalabilidade do sistema.

Objetivo

- Apresentar as técnicas básicas de escalonamento, comunicação e sincronização de processos distribuídos.
- Apresentar ferramentas para análise de desempenho de sistema distribuídos.
- Apresentar os principais conceitos envolvidos na resolução paralela de problemas computacionais.
- Apresentar as principais plataformas de distribuição atualmente disponíveis.

Programa da Disciplina (1)

- Introdução
  - Conceitos Fundamentais
  - Arquiteturas de Sistemas Distribuídos
  - Aspectos de Projeto
- Comunicação entre Processos
  - Troca de Mensagens
  - Chamada Remota a Procedimentos
  - Memória Compartilhada Distribuída
  - Comunicação Grupal
- Sincronização em Sistemas Distribuídos
  - Sincronização Através do *Clock*
  - Algoritmos de Exclusão Mútua
  - Algoritmos de Votação
  - Algoritmos para *Broadcast/Commit*
  - Algoritmos para Prevenção de *Deadlocks*

Programa da Disciplina (2)

- Escalonamento de Processos Distribuídos
  - Políticas de Escalonamento
  - Balanceamento de Carga
  - Resposta e Estabilidade no Escalonamento
- Análise de Desempenho
  - Caracterização da Carga de Trabalho
  - Métricas para Avaliar o Desempenho do Sistema
  - Relatando Desempenho
  - Escalabilidade e *Speedup*
- Sistemas de Arquivo Distribuídos
  - Aspectos de Projeto
  - Implementação
  - Novas Tendências

Programa da Disciplina (3)

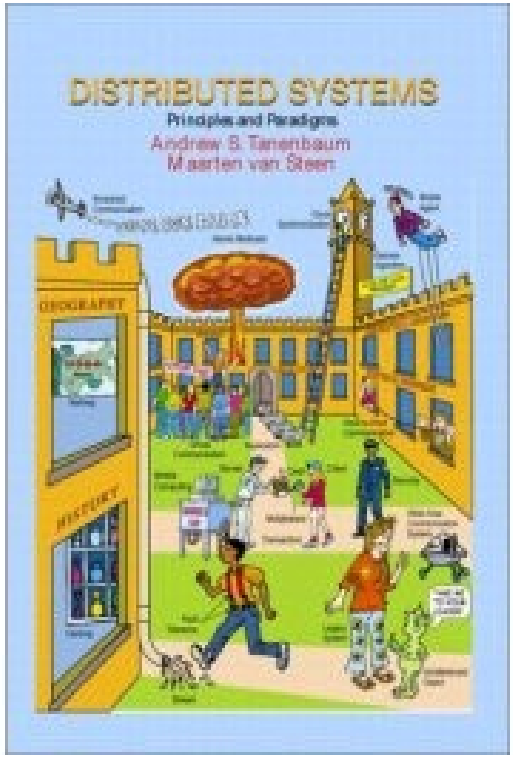
- Projeto de Algoritmos Paralelos
  - Particionamento do Problema
  - Comunicação
  - Algomeração
  - Alocação de Processadores

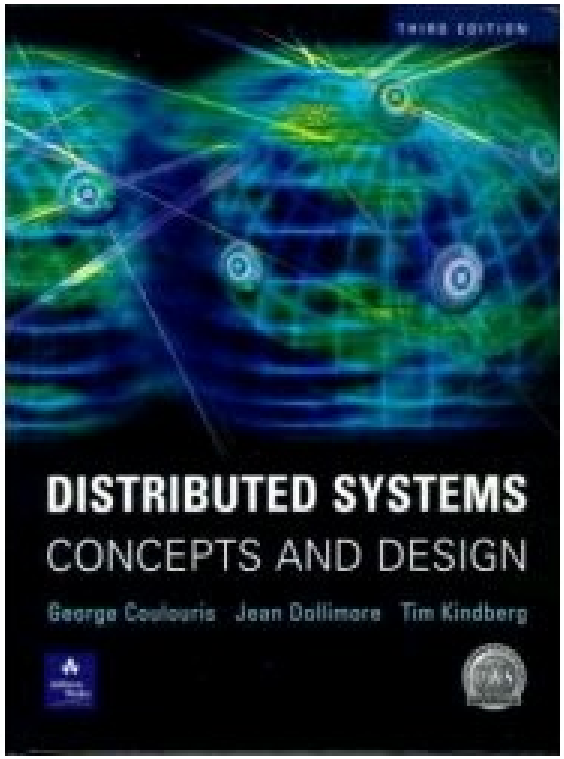
- Plataformas Distribuídas da Atualidade
  - Java *Threads* & *TCP/IP sockets*
  - Java RMI – *Remote Method Invocation*
  - Java J2EE – *Java 2 Enterprise Edition*
  - PVM - *Parallel Virtual Machine*
  - MPI - *Message Passing Interface*
  - Treadmarks – Distributed Method Invocation
  - CORBA - *Common Object Request Broker*

Livros Texto

Distributed Systems: Principles and Paradigms (Hardcover)

Andrew S. Tanenbaum, Maarten van Steen





Distributed Systems: Concepts and Design (3rd Edition)

by George Coulouris, Jean Dollimore, Tim Kindberg

Método de Avaliação

- somente serão aprovados os alunos que atingirem **60%** de aproveitamento;

- **30 %** dos pontos serão distribuídos através de **listas de exercícios** que deverão ser feitas pelos alunos e entregues na data estipulada pelo professor. Aqueles alunos que não entregarem as listas de exercícios nas datas corretas serão penalizados;

- **70%** dos pontos serão distribuídos através de **3 trabalhos práticos** que deverão ser feitos pelos alunos e entregues na data estipulada pelo professor. Aqueles alunos que não entregarem os trabalhos práticos nas datas corretas serão penalizados; **(não haverá prova)**

- para aqueles **alunos que não atingirem 60%** de aproveitamento (somando-se os pontos distribuídos nas listas de exercícios e trabalhos práticos) e que tiverem **75% ou mais da frequência** durante o período letivo, será concedido um único exame especial cuja **nota deverá ser somada aos pontos obtidos pelo aluno em todas atividades acadêmicas realizadas durante o período e a soma deverá ser dividida por 2**. O valor resultante deverá ser superior a 6.0 para que o aluno seja aprovado.Este exame constitui-se de uma **prova escrita englobando toda a matéria** lecionada durante o período letivo; e

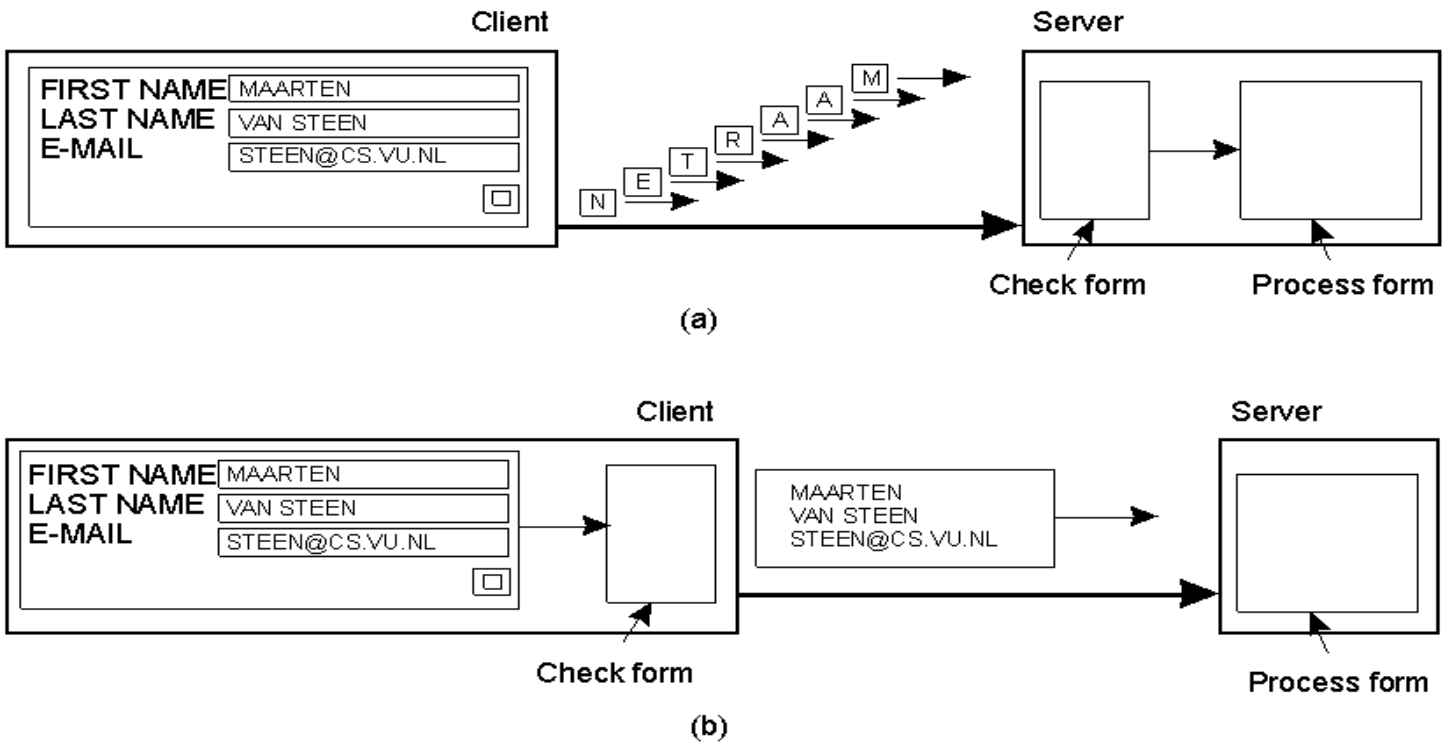
- **forma de penalização:** aqueles **alunos que atrasarem a entrega** dos trabalhos práticos ou listas de exercícios terão as notas desses trabalhos ou listas reduzidas em **20%**.

Informações sobre Curso

- Material didático
  - www.decom.ufop.br/prof/tiago
- Dúvidas
  - tiago@decom.ufop.br

Escalabilidade

Técnicas para Escalabilidade (1)

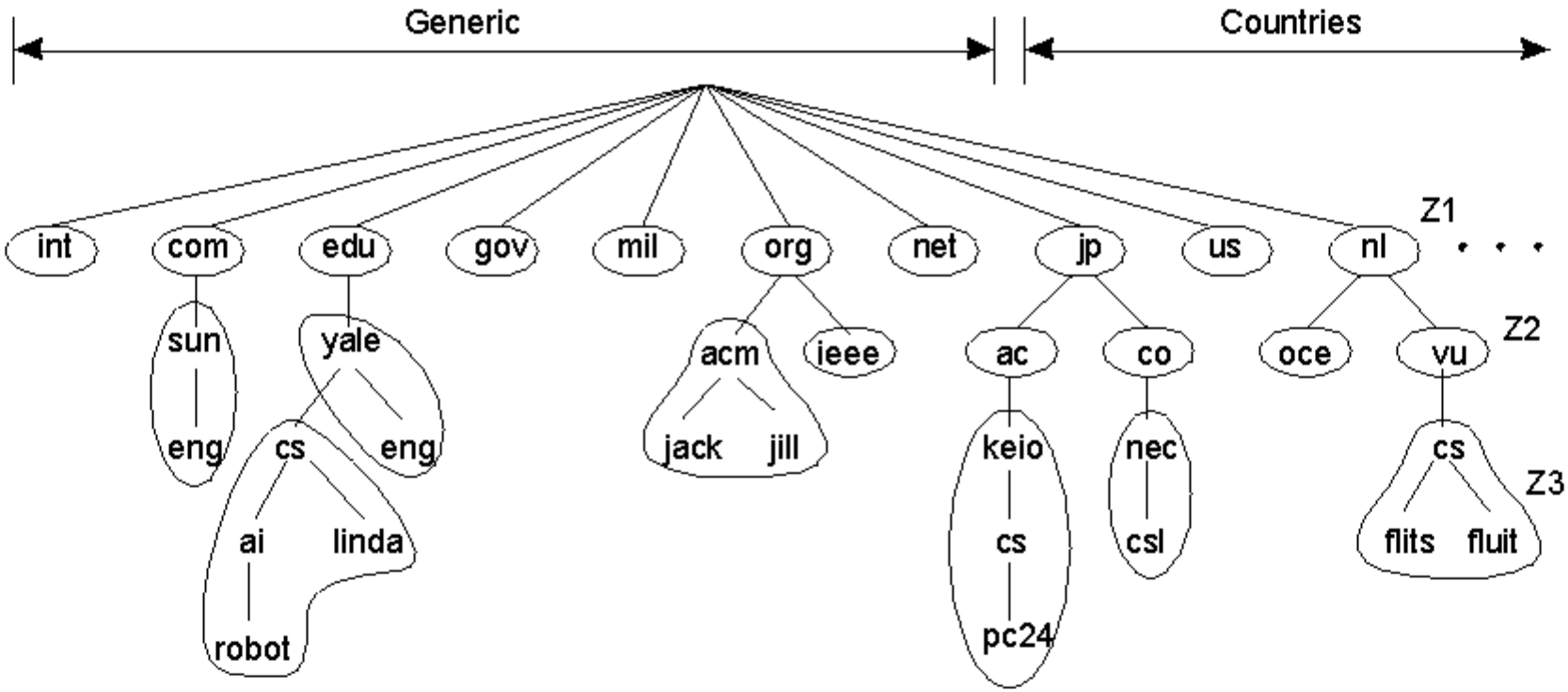


A diferença entre:

b) o servidor todos os formulários

c) o cliente checa o formulário durante seu preenchimento

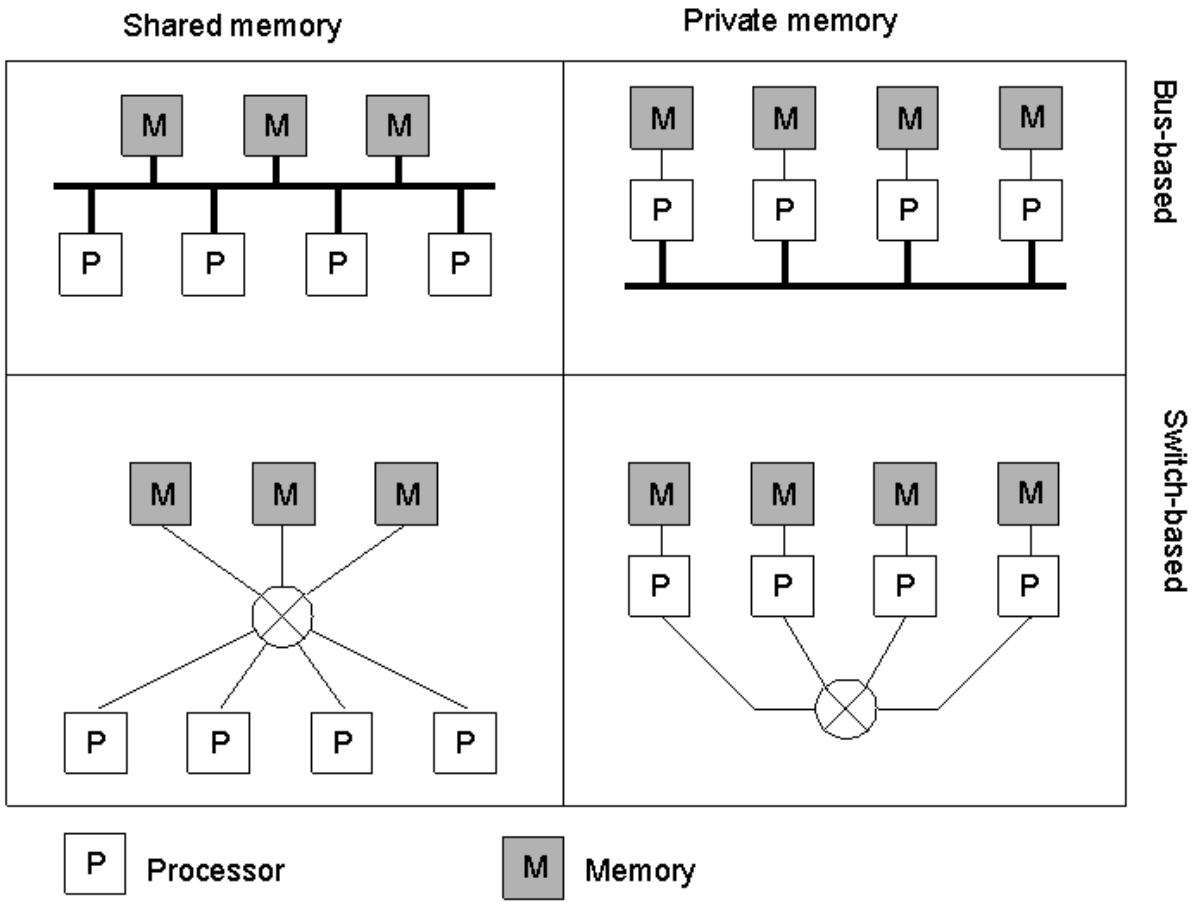
Técnica para Escalabilidade (2)



Na Internet, os nomes de domínios (DNS - *Domain Name Service)* são hierarquicamente organizados em zonas.

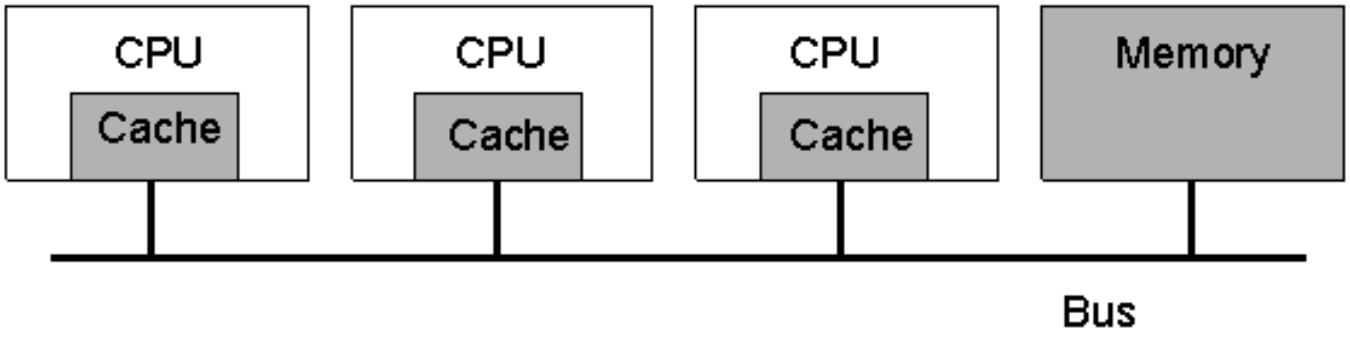
Conceitos de *Hardware*

Arquiteturas de *Hardware*



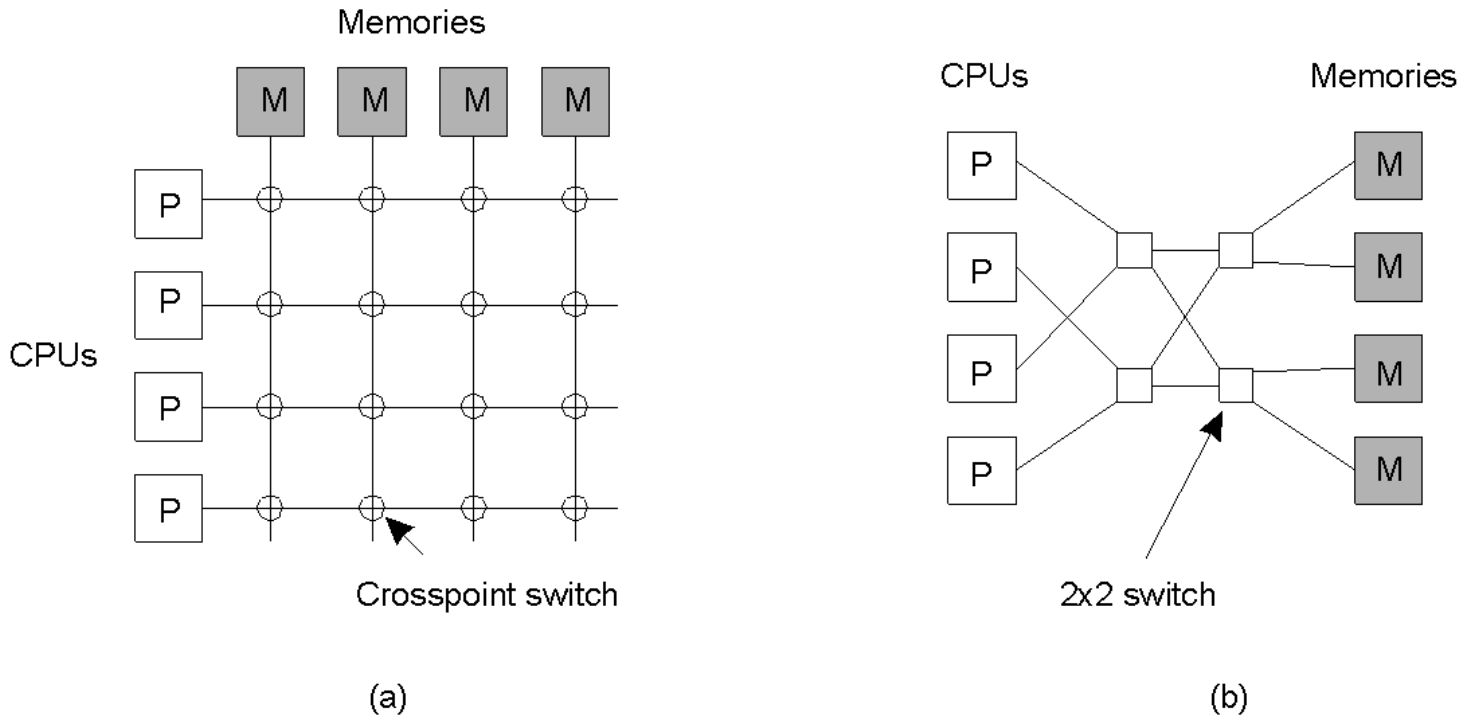
Diferentes arquiteturas de sistemas distribuídos

Multiprocessadores (1)



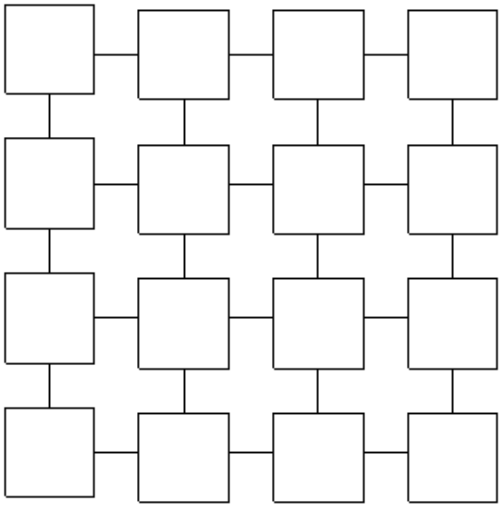
- A bus-based multiprocessor.

Multiprocessadores (2)

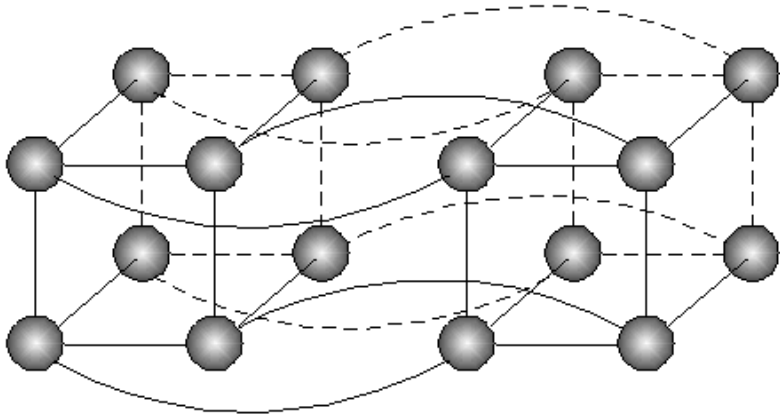


a) A crossbar switch
b) An omega switching network

Multi-computadores Homogêneos



(a)



(b)

a) Grid
b) Hypercube

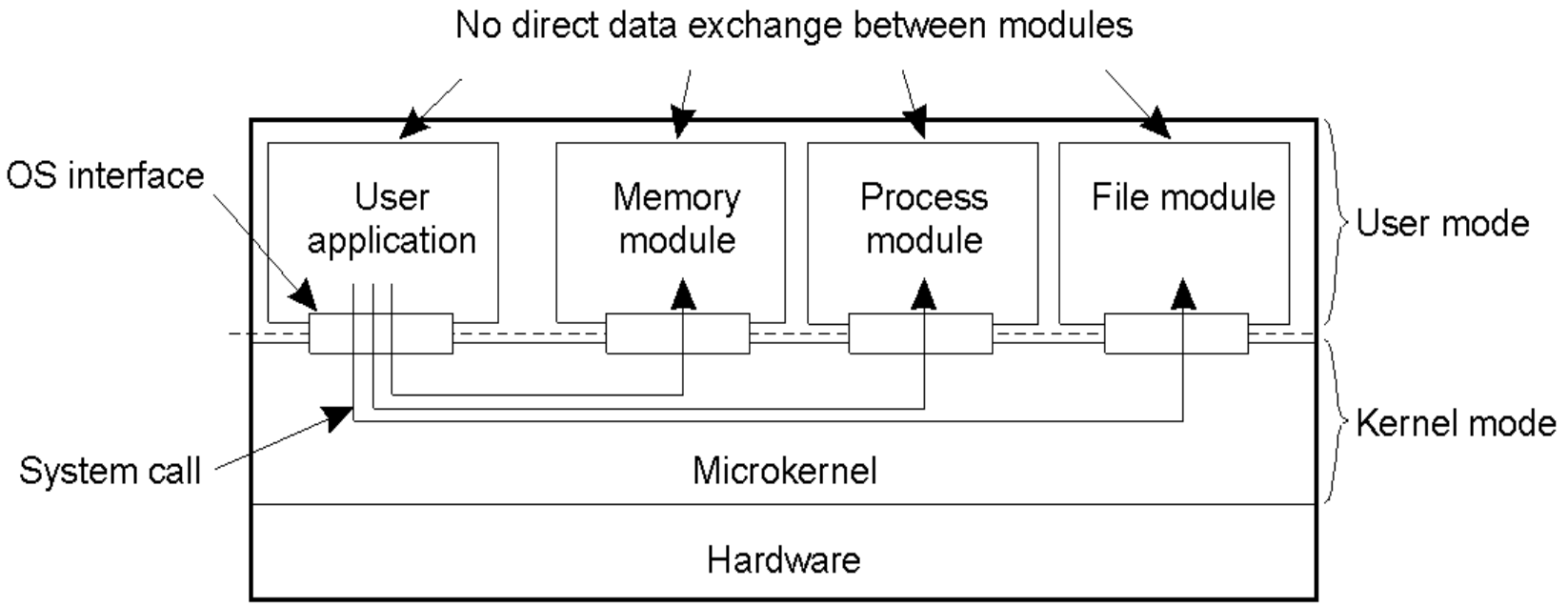
Conceitos de Software

Sistemas Operacionais

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)

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

- A monitor to protect an integer against concurrent access.

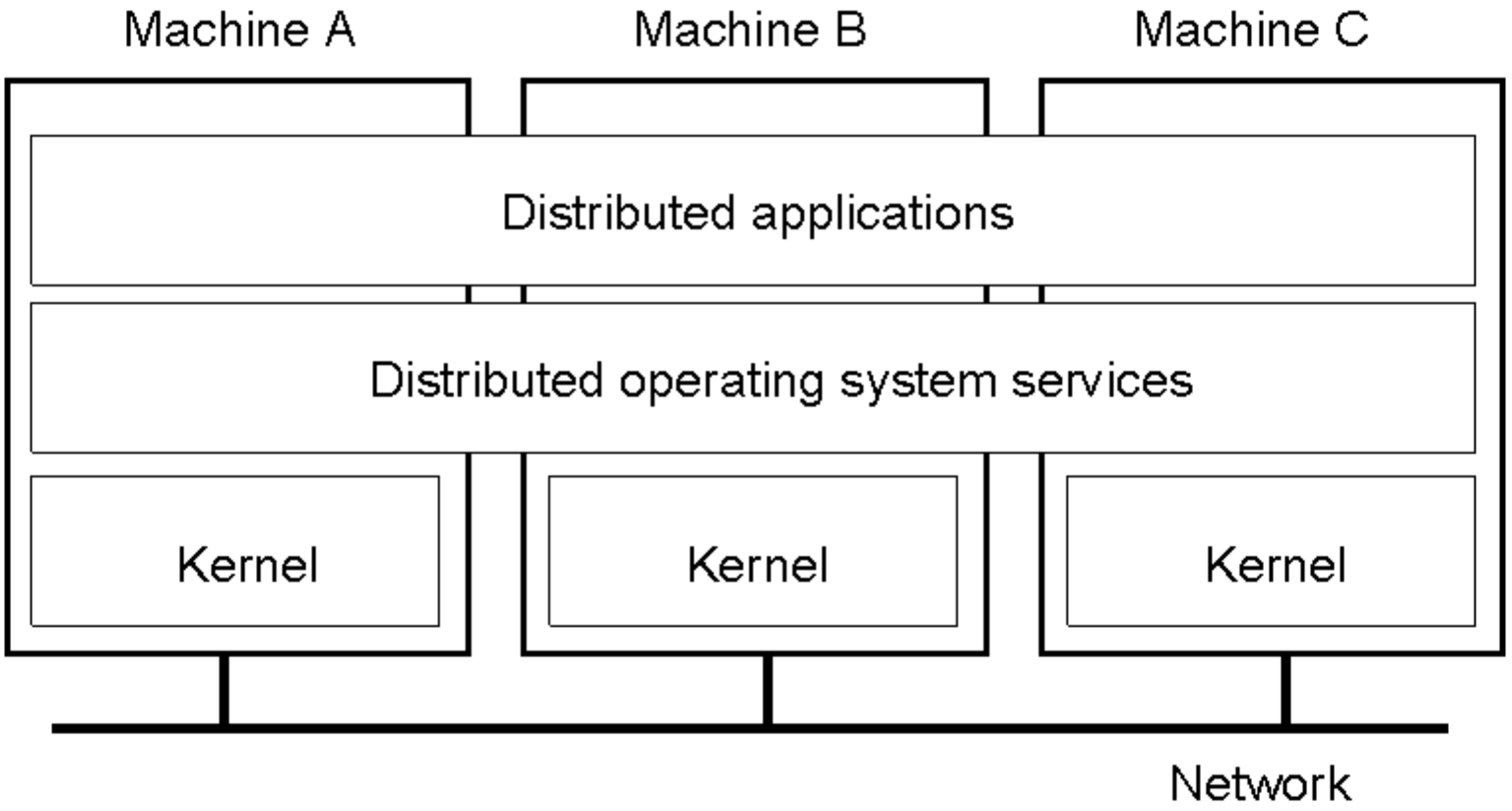
Multiprocessor Operating Systems (2)

```
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;
 }
}
```

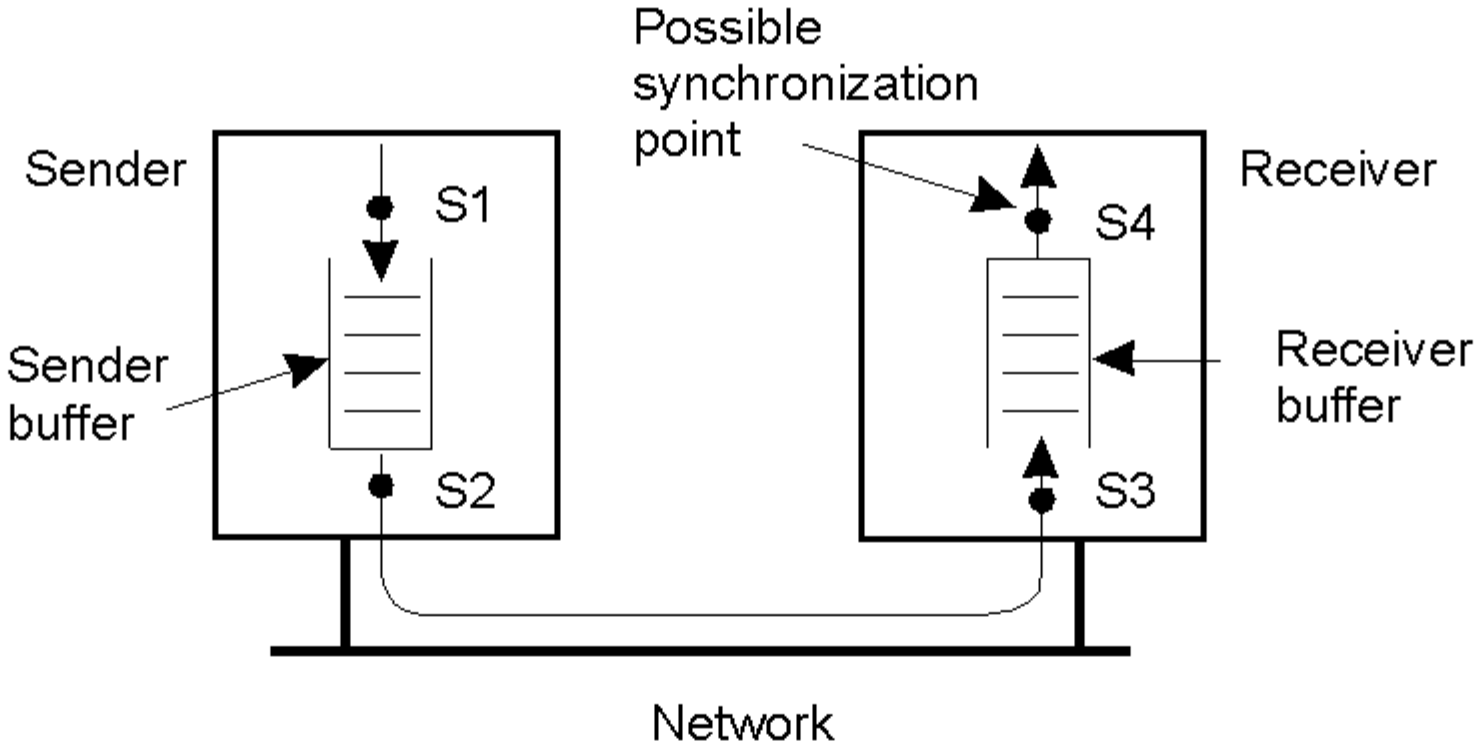
A monitor to protect an integer against concurrent access, but blocking a process.

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

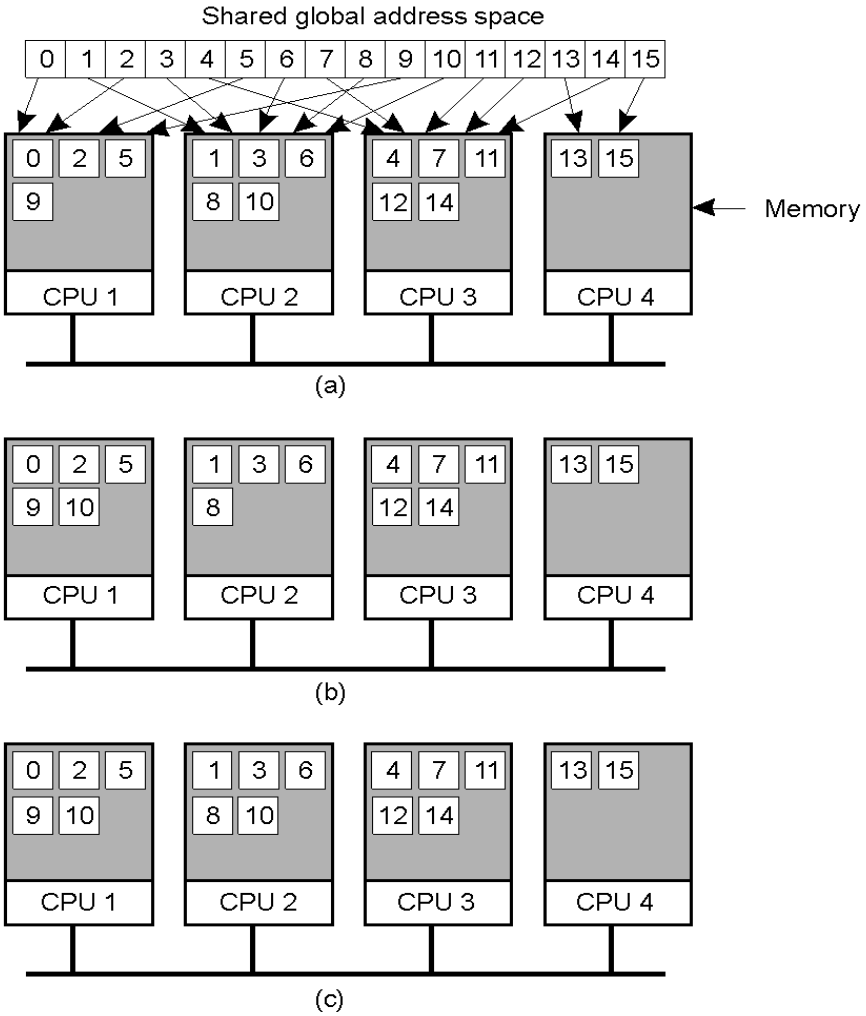
Relation between blocking, buffering, and reliable communications.

Distributed Shared Memory Systems (1)

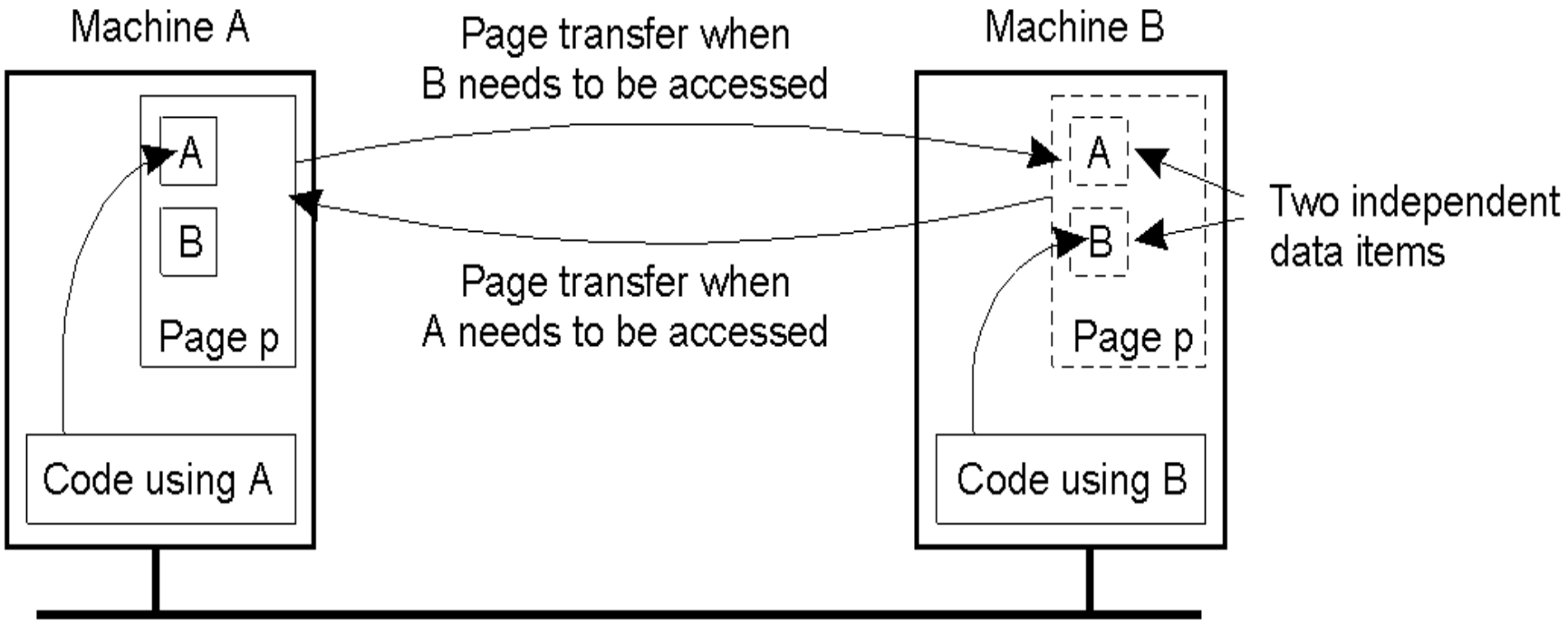
a) Pages of address space distributed among four machines

c) Situation after CPU 1 references page 10

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

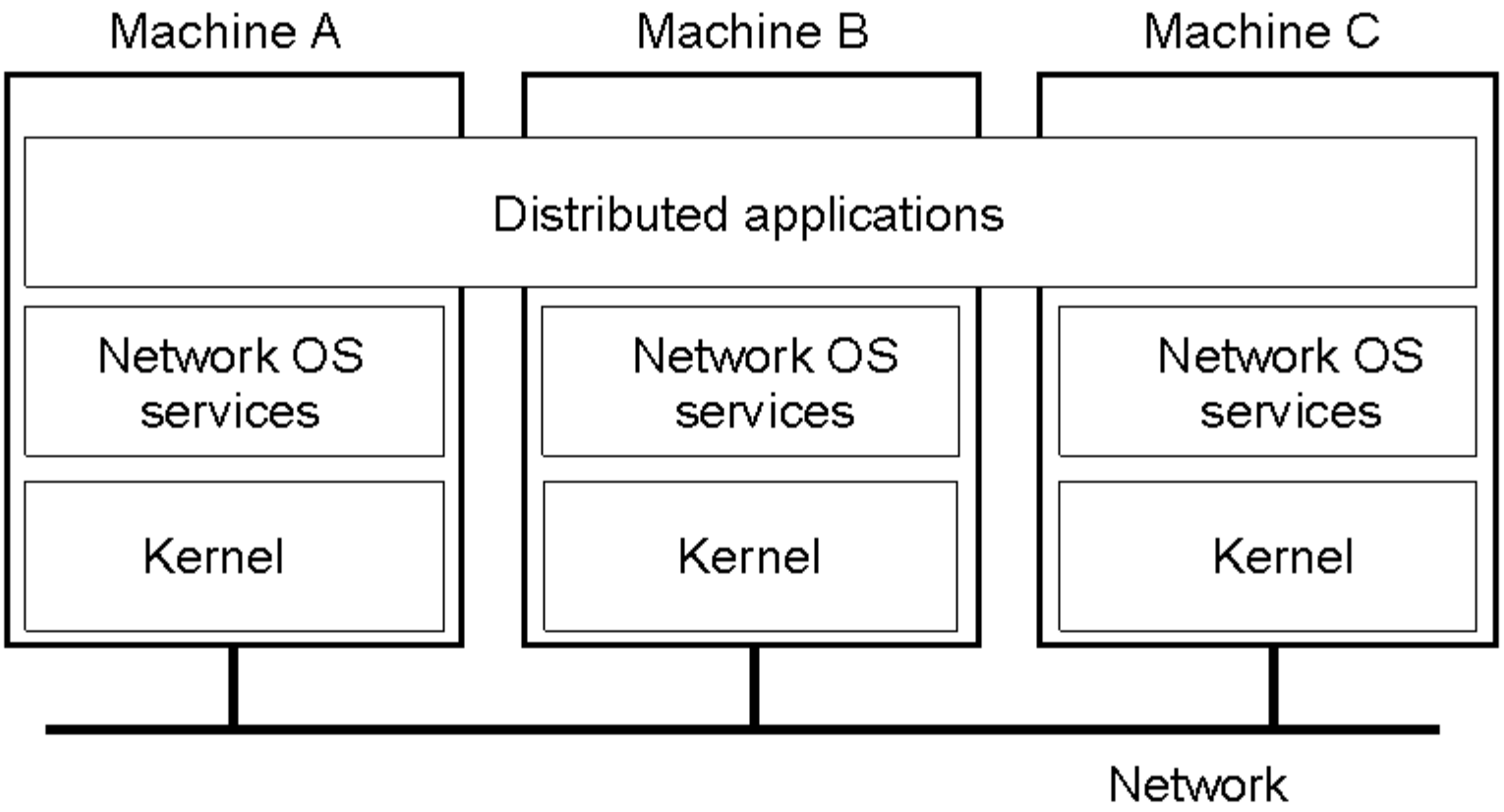


Distributed Shared Memory Systems (2)



Distributed Shared Memory Systems (2)

Network Operating System (1)

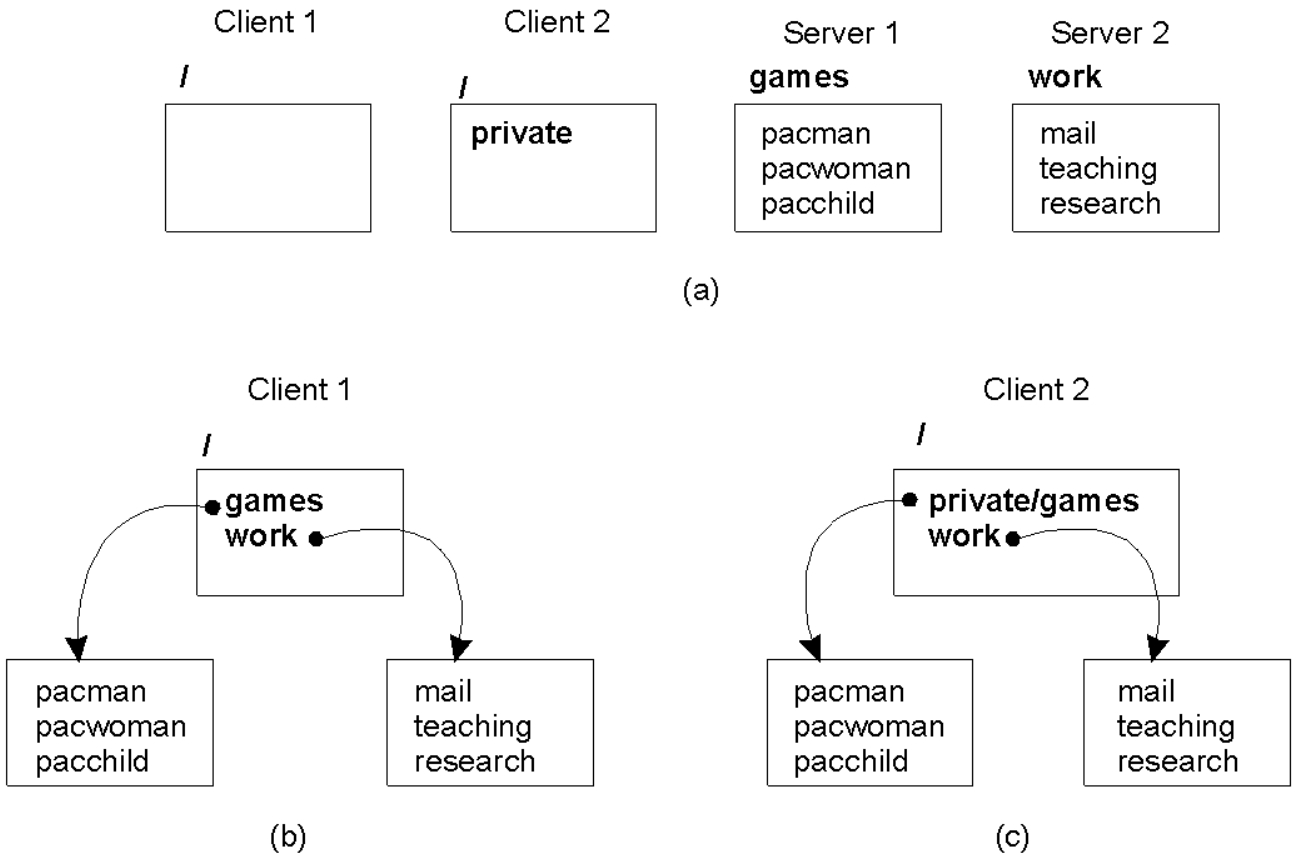


General structure of a network operating system.

Network Operating System (2)

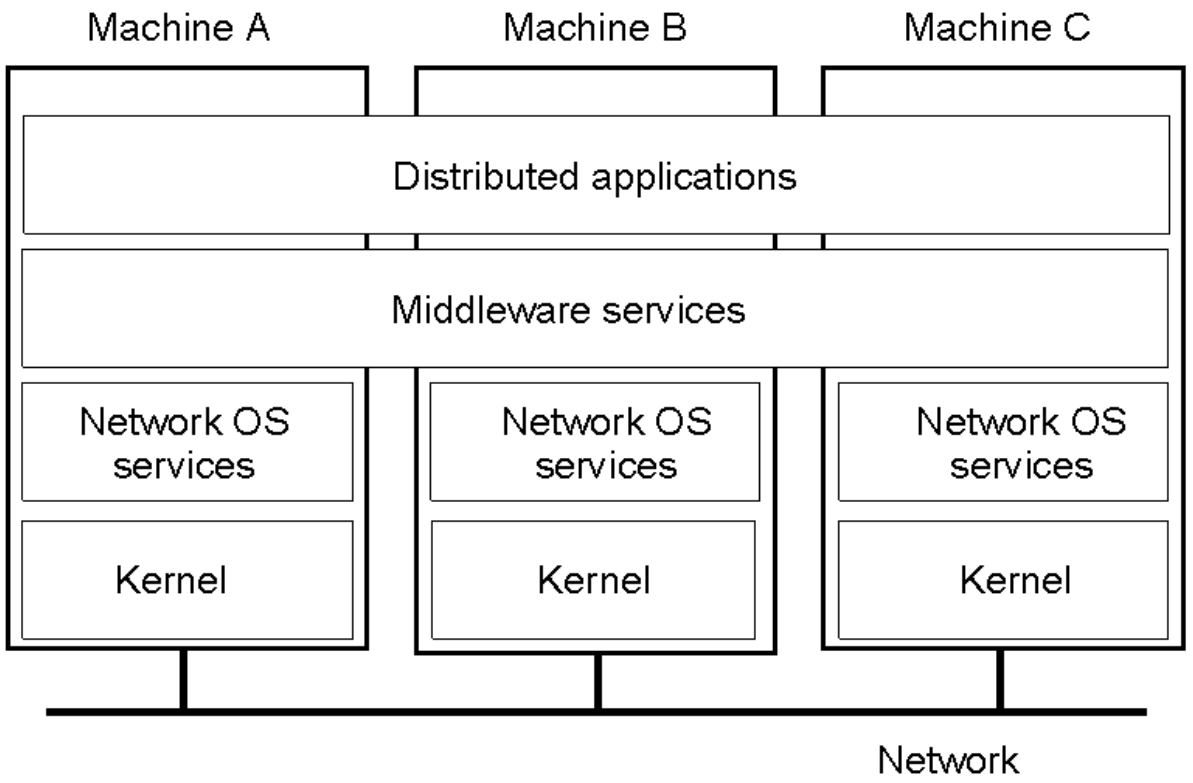
Two clients and a server in a network operating system.

Network Operating System (3)



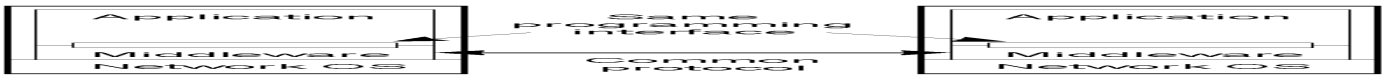
Different clients may mount the servers in different places.

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

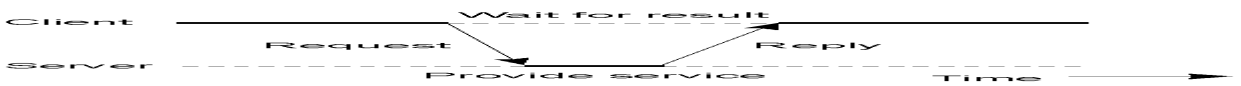
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global,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

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

Aplicações Distribuidas

Clients and Servers

- General interaction between a client and a server.



An Example Client and Server (1)

```
/* 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           */
};
```

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

An Example Client and Server (2)

```
#include <header.h>
void main(void) {
     struct message m1, m2;                    /* incoming and outgoing messages      */
     int r;                                    /* result code                         */

     while(TRUE) {                             /* server runs forever                 */
          receive(FILE_SERVER, &m1);           /* block waiting for a message         */
          switch(m1.opcode) {                  /* dispatch on type of request         */
               case CREATE:    r = do_create(&m1, &m2); break;
               case READ:      r = do_read(&m1, &m2); break;
               case WRITE:     r = do_write(&m1, &m2); break;
               case DELETE:    r = do_delete(&m1, &m2); break;
               default:        r = E_BAD_OPCODE;
          }
          m2.result = r;                       /* return result to client             */
          send(m1.source, &m2);                /* send reply                          */
     }
}
```

- A sample server.

An Example Client and Server (3)

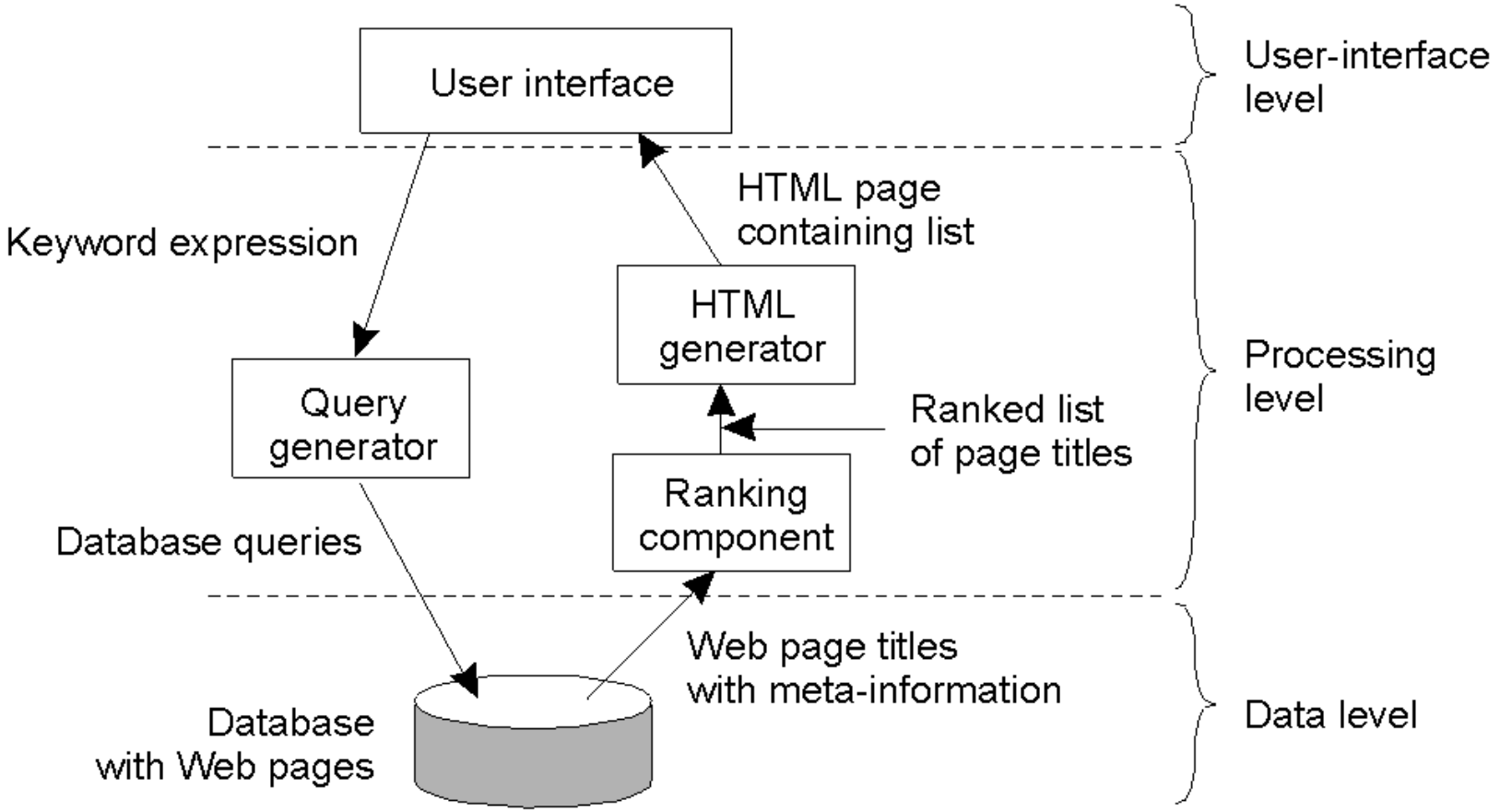
```
#include <header.h>
int copy(char *src, char *dst){                 /* procedure to copy file using the server     */
     struct message ml;                         /* message buffer                              */
     long position;                             /* current file position        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

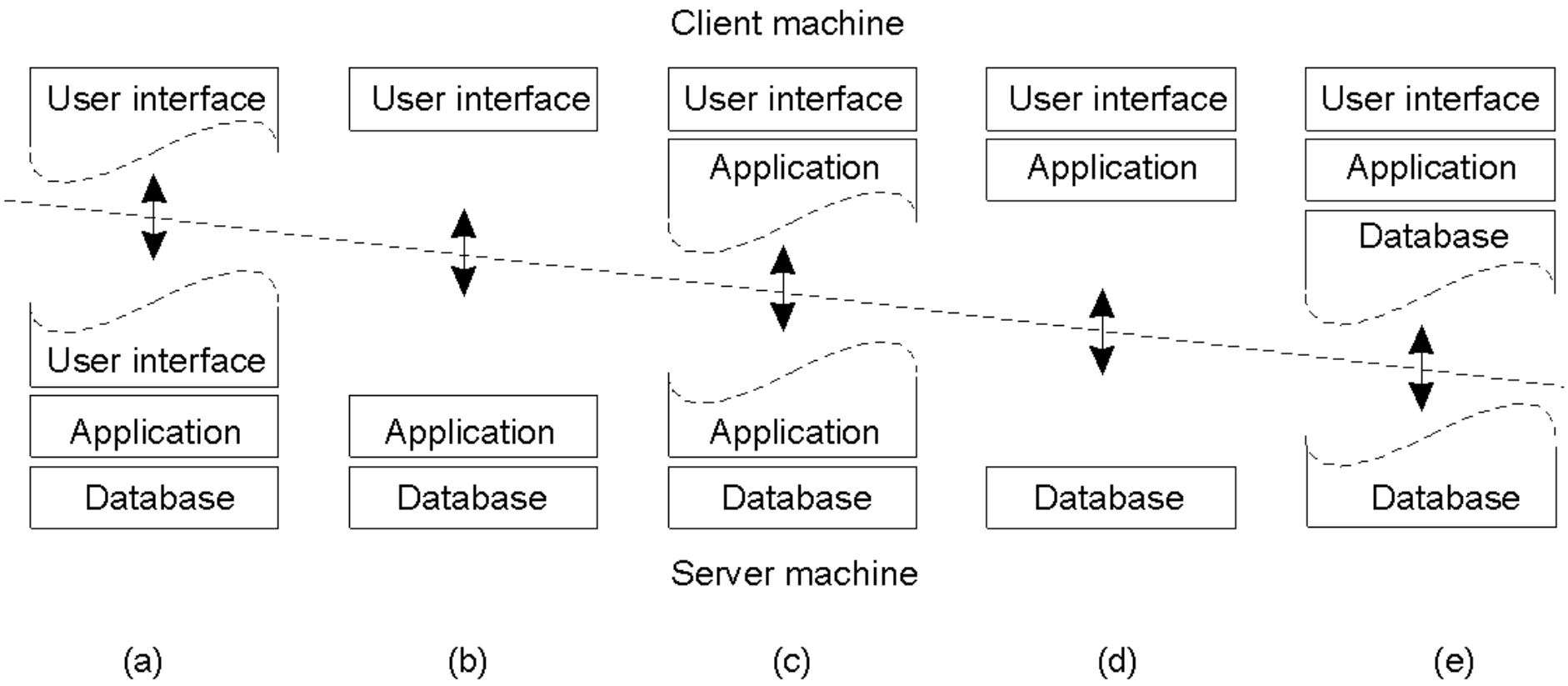
Multitiered Architectures (1)



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

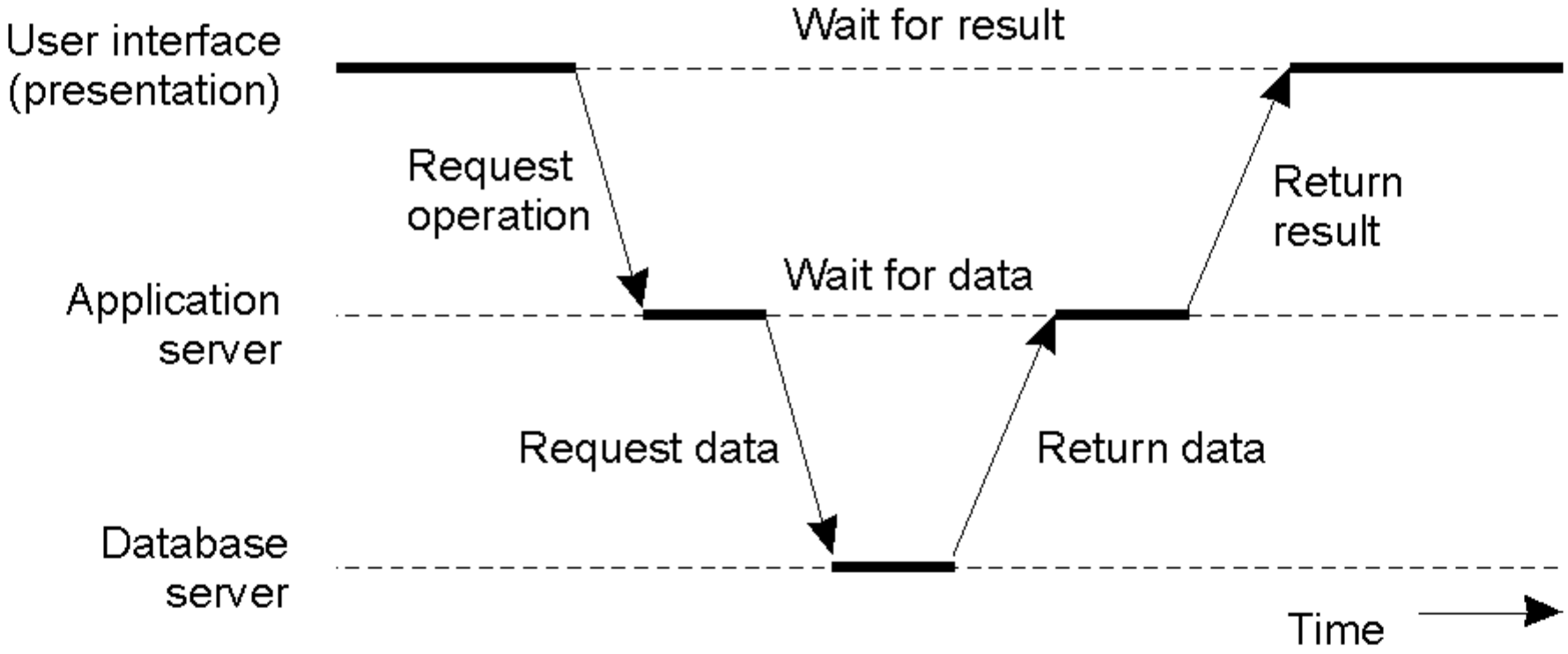
Multitiered Architectures (2)

- 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.