

# Sistemas Distribuídos

**Prof. Fernando W Cruz**

# Introdução

- ↪ Desenvolvimento de microprocessadores poderosos a um baixo custo
- ↪ Desenvolvimento de redes de alta velocidade

**Surgem os Sistemas Distribuídos  
em contraste com os Sistemas  
Centralizados**

# Introdução

## ➤ Definição

- Coleção de computadores independentes que mostra-se aos usuários do sistema como um **único sistema** (Tanenbaum).
- Computadores interligados comunicam-se e coordenam suas ações através da troca de mensagens (Coulouris).

## ➤ Exemplo

- Internet, Web, Intranet, Computação Móvel

# Vantagens dos Sistemas Distribuídos

- ↪ Economia

- ↪ Velocidade

  - ↪ Um sistema distribuído pode ter um poder de processamento maior que o de qualquer mainframe

- ↪ Distribuição inerente

- ↪ Confiabilidade

- ↪ Escalabilidade

# Desvantagens dos Sistemas Distribuídos

- ↪ Maior complexidade
- ↪ Dependência da tecnologia de rede
- ↪ Segurança das informações

# Objetivos dos Sistemas Distribuídos

## ↗ Conectar usuários e recursos

- ↗ Economia

- ↗ Facilitar colaboração e troca de informação

## ↗ Transparência

- ↗ Quanto à localização

- ↗ Quanto à migração

- ↗ Quanto à relocação (Computação Móvel)

- ↗ Quanto à replicação

- ↗ Quanto à concorrência

- ↗ Quanto a falhas

# Objetivos dos Sistemas Distribuídos

- ↪ Ser um sistema aberto (*Openness*)
  - Usar IDL's (Linguagens de Definição de Interface)
  - Interoperabilidade
    - Componentes de diferentes fabricantes conversam entre si
  - Portabilidade
    - Aplicações desenvolvida para um SD A devem executar, sem modificação, num SD B
  - Flexibilidade
    - Fácil de configurar
    - Extensibilidade

# Objetivos dos Sistemas Distribuídos

## ↪ Escalabilidade

- Em termos de tamanho
- Em termos de distribuição geográfica
- Em termos de administração

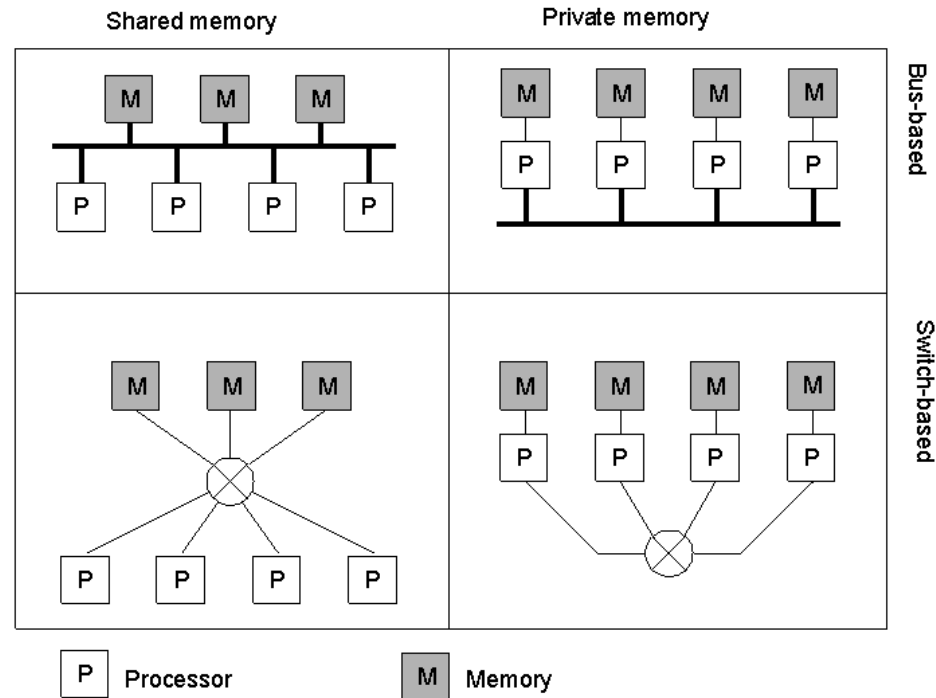
↪ Há benefícios e problemas



# Conceitos de Hardware

- Existem várias configurações possíveis de hardware em um SD
- Considerando uma divisão com base na utilização de memória
  - Multiprocessadores: compartilham memória
  - Multicomputadores: não compartilham memória
    - Comunicação entre processos é feita via troca de mensagens

# Conceitos de Hardware



# Conceitos de Hardware

- Sistemas multicomputadores ainda podem ser
  - Homogêneos
    - Clusters de workstations
  - Heterogêneos
    - Internet, Grids Computacionais

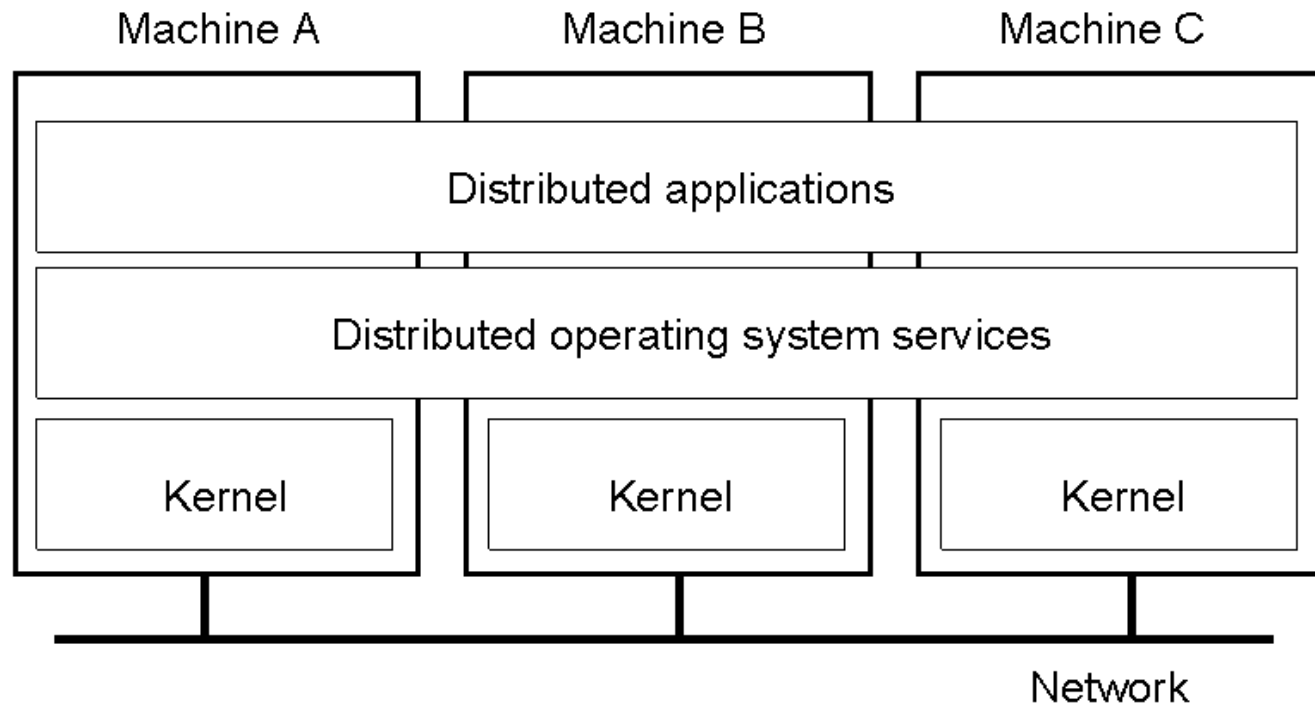
# Conceitos de Software

- Sistemas Distribuídos têm as mesmas funções de um SO tradicional
  - Gerente de recursos
  - Facilitar o uso dos recursos distribuídos
    - Máquina virtual

# Conceitos de Software

- Sistemas fortemente acoplados
  - Visão única do sistema
  - Sistemas distribuídos
  - Mesmo SO em todos os nodos
  - Usados em sistemas multiprocessadores e multicomputadores **homogêneos**

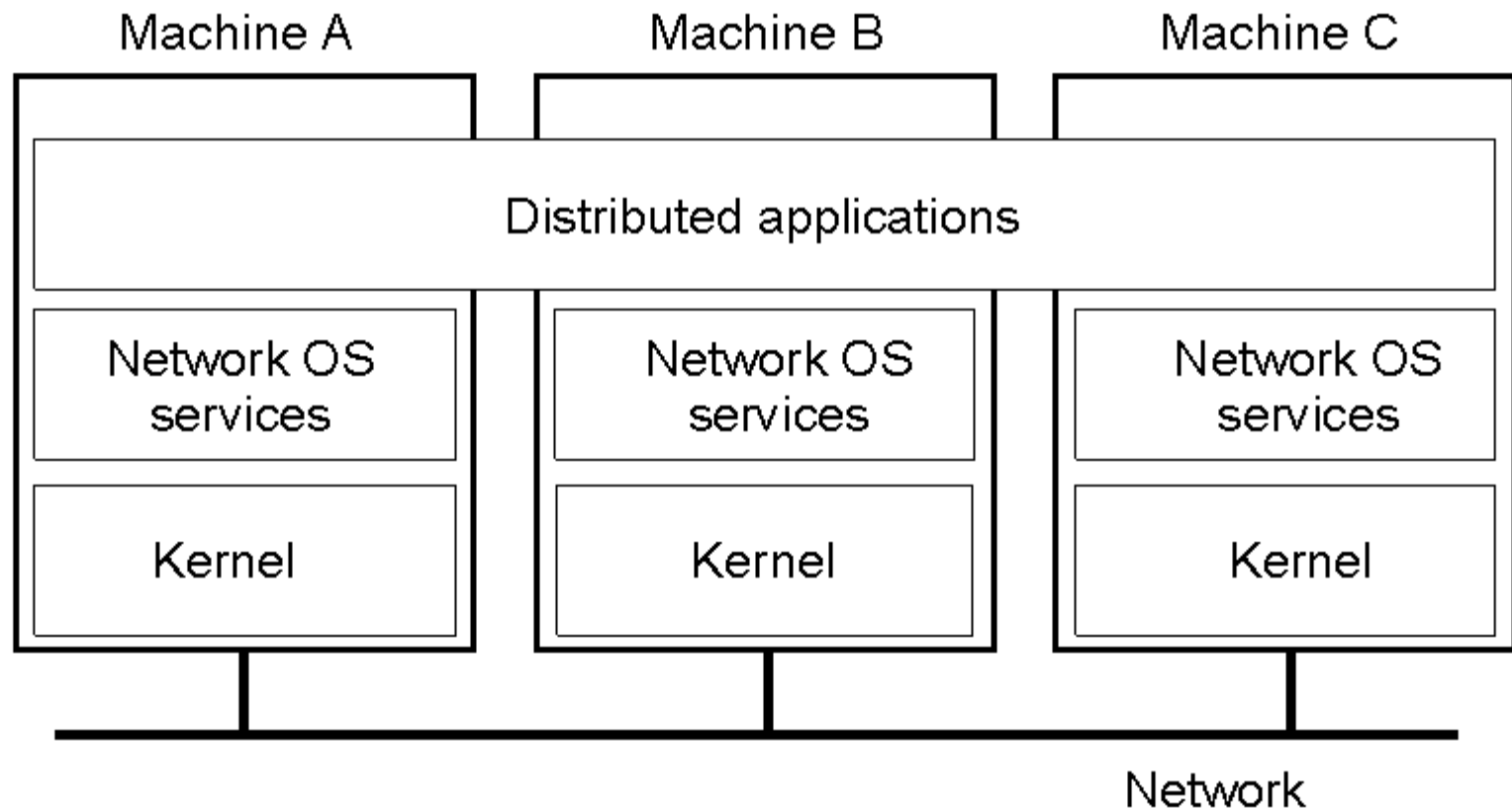
# Sistema Distribuído Multicomputador



# Conceitos de Software

- Sistemas fracamente acoplados
  - Sistemas Operacionais de Rede
  - Sistemas multicomputadores heterogêneos
  - Fornecem serviços locais para clientes remotos
  - Não há uma visão única do sistema

# Sistema Operacional de Rede

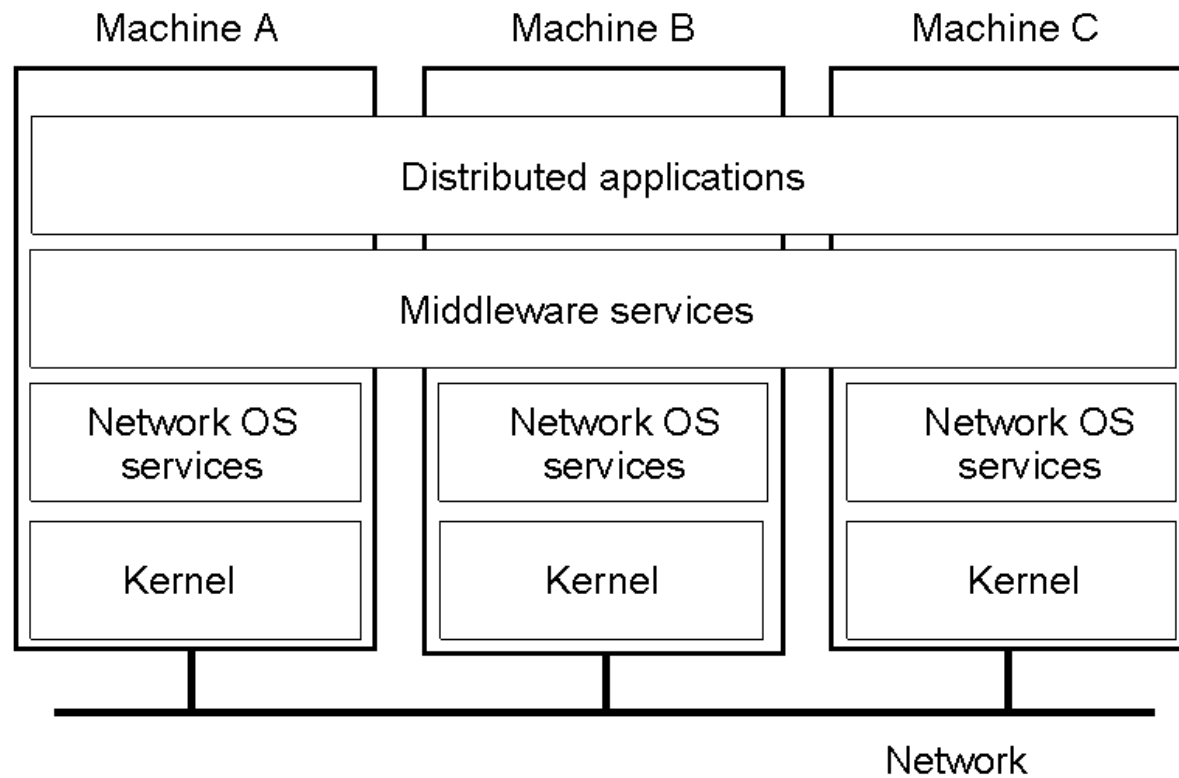




# Conceitos de Software

- Nem os sistemas distribuídos nem os sistemas operacionais de rede atendem à definição inicial
  - Coleção de computadores independentes que mostra-se aos usuários do sistema como um único sistema (Tanenbaum).
- SD são mais fáceis de usar
- SOR são mais escaláveis

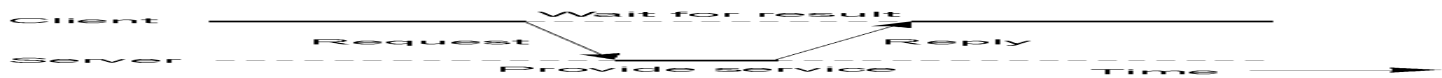
# Middleware



# Middleware

- O sistema operacional de rede gerencia os recursos locais
- O middleware esconde a heterogeneidade
- O middleware fornece serviços às aplicações
  - Facilidades de comunicação (RPC, RMI)
  - Segurança

# O Modelo Cliente Servidor



# Um exemplo de um servidor de arquivos

```
#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 */
    }
}
```

```

#include <header.h>
int copy(char *src, char *dst){
    struct message ml;
    long position;
    long client = 110;

    initialize( );
    position = 0;
    do {
        ml.opcode = READ;
        ml.offset = position;
        ml.count = BUF_SIZE;
        strcpy(&ml.name, src);
        send(FILESERVER, &ml);
        receive(client, &ml);

        /* Write the data just received to the destination file.
        ml.opcode = WRITE;
        ml.offset = position;
        ml.count = ml.result;
        strcpy(&ml.name, dst);
        send(FILE_SERVER, &ml);
        receive(client, &ml);
        position += ml.result;
    } while( ml.result > 0 );
    return(ml.result >= 0 ? OK : ml.result);
}

```

(a)

```

/* procedure to copy file using the server */
/* message buffer */
/* current file position */
/* client's address */

/* prepare for execution */

/* operation is a read */
/* current position in the file */
/* how many bytes to read*/
/* copy name of file to be read to message */
/* send the message to the file server */
/* block waiting for the reply */

/* operation is a write */
/* current position in the file */
/* how many bytes to write */
/* copy name of file to be written to buf */
/* send the message to the file server */
/* block waiting for the reply */
/* ml.result is number of bytes written */
/* iterate until done */
/* return OK or error code */

```

# Arquitetura em três camadas

