INE 5418

Computação Distribuída

Professor:

Frank Siqueira

INE – UFSC

frank@inf.ufsc.br

Conteúdo Programático

1. Fundamentos de Computação Distribuída

   Arquitetura de Sistemas Distribuídos; Paradigmas de Computação Distribuída; Suporte Computacional para Aplicações Distribuídas; Comunicação entre Processos; Sistemas de Arquivos Distribuídos

2. Tecnologias para Computação Distribuída

   Objetos Distribuídos; Web Services; Redes Peer-to-Peer; Middleware Orientado a Mensagens; Memória Compartilhada Distribuída; Grids Computacionais

3. Algortimos para Computação Distribuída

   Segurança de Funcionamento; Comunicação em Grupo; Coordenação e Acordo; Sincronização

Recursos Computacionais

- Página da disciplina
  http://www.inf.ufsc.br/~frank/INE5418
- Lista de e-mails
  ine5418-05208@inf.ufsc.br
- Linguagem de Programação
  Java SE (opcional: C ou C++)

Unidade 1

Fundamentos de Computação Distribuída

- Arquitetura de Sistemas Distribuídos
- Paradigmas de Computação Distribuída
- Suporte Comp. para Aplic. Distribuídas
- Comunicação entre Processos
- Sistemas de Arquivos Distribuídos

Panorama Atual

Sistemas computacionais estão cada vez mais elaborados e complexos

Grande parte das máquinas interligada por redes de computadores

Computação Distribuída

Sistemas Distribuídos

- Maior poder de processamento
- Maior carga, maior número de usuários, ...
- Melhor tempo de resposta, Maior confiabilidade ...

Fundamentos de Computação Distribuída

- Definição de Computação Distribuída
  - Consiste em executar aplicações cooperantes em máquinas diferentes
  - Tornou-se possível a partir da popularização das redes de computadores
- Aplicações são executadas em máquinas diferentes interligadas por uma rede
  - Intranets
  - Internet
  - Outras redes públicas ou privadas

Fundamentos de Computação Distribuída

- Acoplamento
  - Sistemas distribuídos são fracamente acoplados, ou seja, trocam dados através de um meio relativamente lento e pouco confiável
- Previsibilidade
  - O comportamento de sistemas distribuídos é imprevisível devido ao uso da rede e à possibilidade de ocorrerem falhas em máquinas e na rede

Fundamentos de Computação Distribuída

- Influência do Tempo
  - Sistemas distribuídos são bastante influenciados pelo tempo de comunicação pela rede; em geral não há uma referência de tempo global
- Controle
  - Sistemas distribuídos não têm total controle sobre todos os recursos computacionais utilizados, pois empregam também recursos de terceiros

Fundamentos de Computação Distribuída

- Vantagens
  - Usam melhor o poder de processamento
  - Apresentam um melhor desempenho
  - Permitem compartilhar dados e recursos
  - Podem apresentar maior confiabilidade
  - Permitem reutilizar serviços já disponíveis
  - Atendem um maior número de usuários
  - ...

Fundamentos de Computação Distribuída

- Dificuldades
  - Desenvolver, gerenciar e manter o sistema
  - Controlar o acesso concorrente a dados e a recursos compartilhados
  - Evitar que falhas de máquinas ou da rede comprometam o funcionamento do sistema
  - Garantir a segurança do sistema e o sigilo dos dados trocados entre máquinas
  - Lidar com a heterogeneidade do ambiente
  - ...

Arquitetura de Sistemas Distribuídos

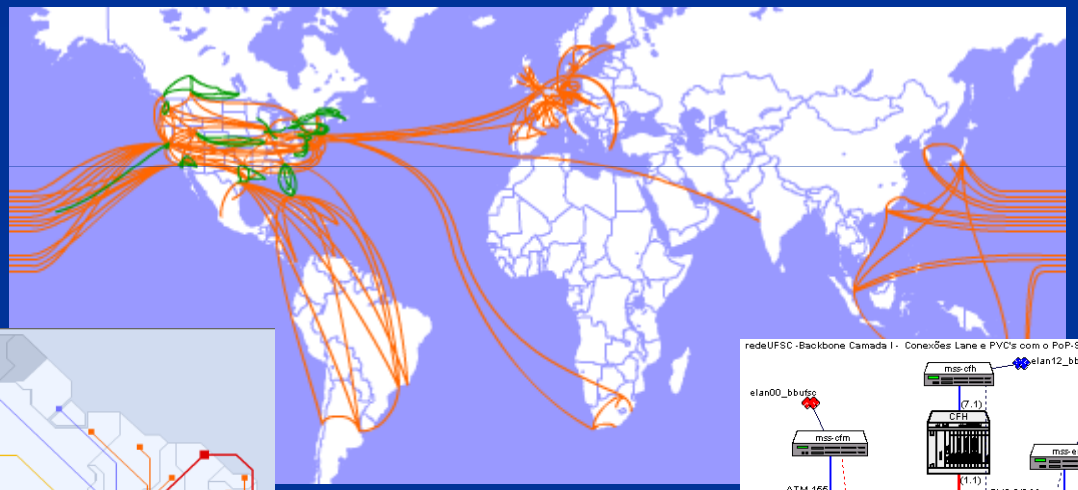
- Diversos dispositivos podem ser usados para execução de aplicações distribuídas
  - Computadores
  - Celulares
  - Sensores
  - Tablets
  - Dispositivos Embarcados
  - etc.

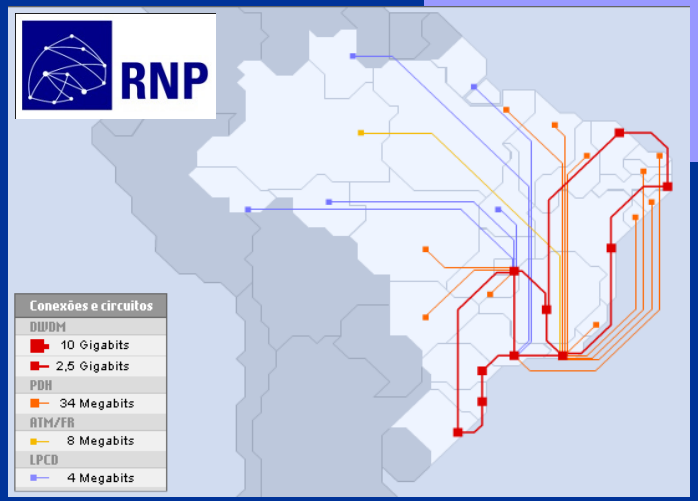
Arquitetura de Sistemas Distribuídos

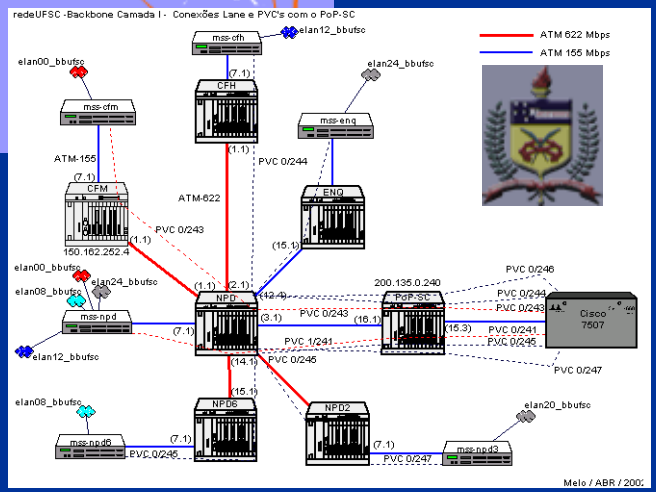
- Recursos computacionais podem ser reunidos em:
  - *Cluster* (agregado): processadores interligados por barramento ou rede de alta velocidade
  - *Grid* (grade): reúnem recursos heterogêneos dispersos geograficamente (distribuídos) interligados por uma rede de longa distância
  - *Cloud* (nuvem): conjunto compartilhado de recursos (servidores, armazenamento, rede, aplicações, etc.), distribuídos ou não, que são alocados temporariamente para os usuários

Arquitetura de Sistemas Distribuídos

- Computação Distribuída requer o uso de redes, como uma Intranet ou a Internet







Paradigmas de Computação Distribuída

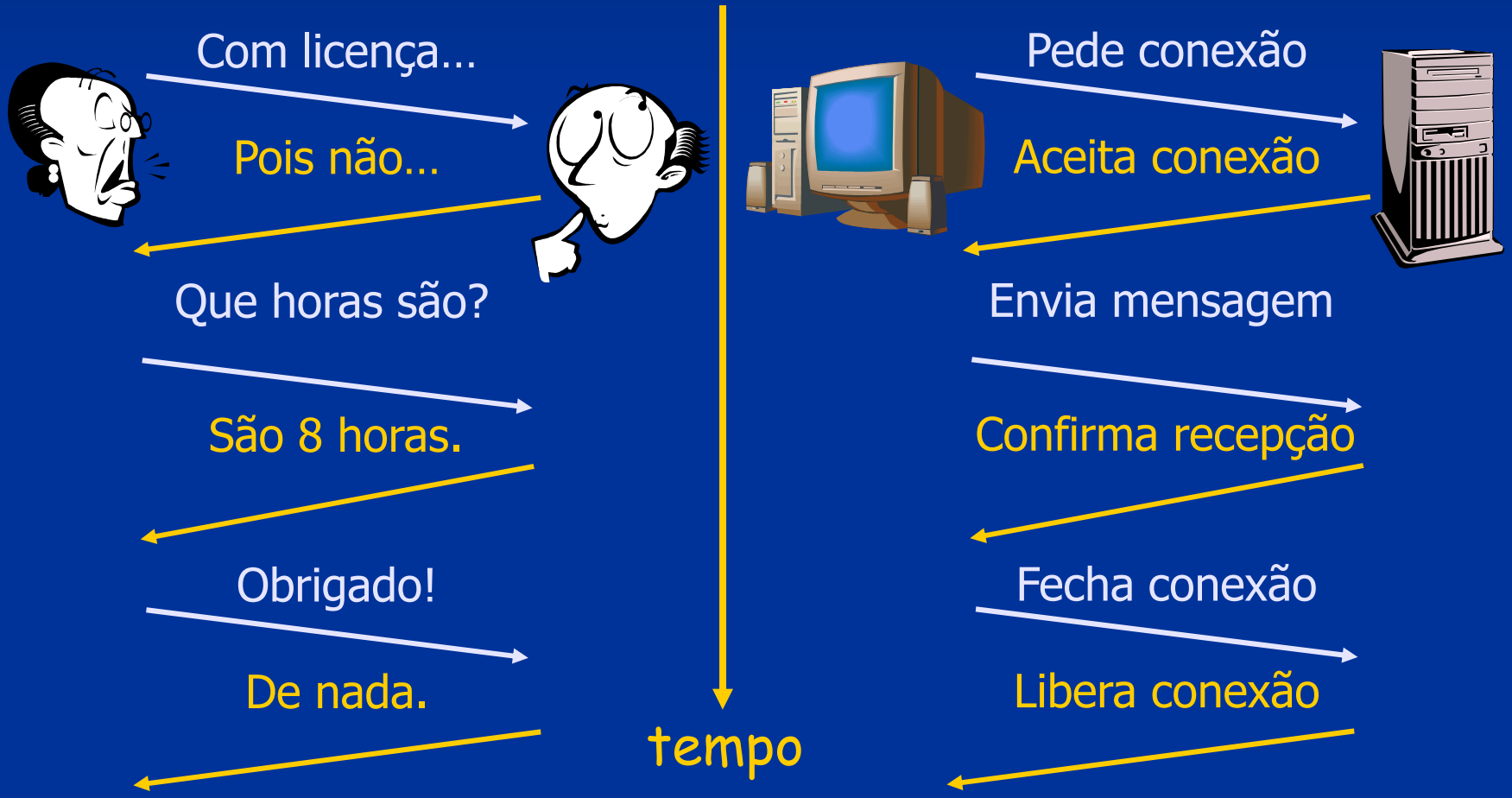
- As partes que compõem um sistema distribuído interagem através da rede para trabalhar de forma cooperativa
  - Trocam dados / mensagens
  - Utilizam os serviços de comunicação fornecidos pelo sistema hospedeiro
  - Adotam protocolos de comunicação para que possam entender uns aos outros

Paradigmas de Computação Distribuída

- Protocolos
  - Estabelecem caminhos virtuais de comunicação
  - Duas entidades precisam usar os mesmos protocolos para trocar informações

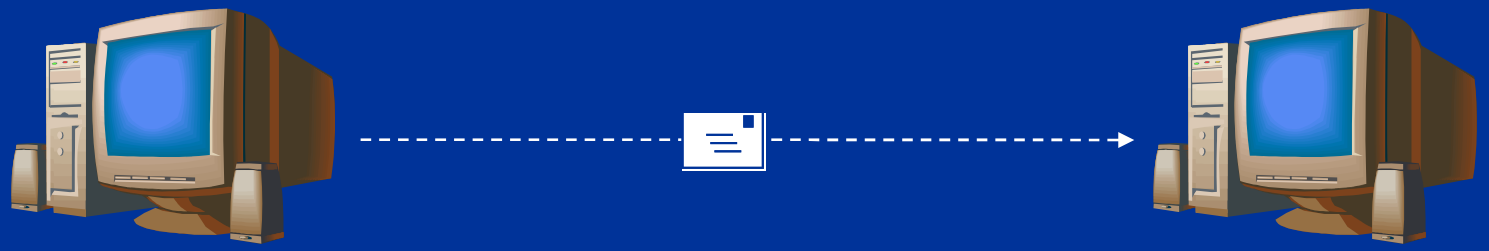
Paradigmas de Computação Distribuída

- Protocolos



Paradigmas de Computação Distribuída

- Serviços de comunicação
  - Serviço sem Conexão: cada unidade de dados é enviada independentemente das demais



  - Serviço com Conexão: dados são enviados através de um canal de comunicação

Paradigmas de Computação Distribuída

- Formas de Interação
  - Troca de mensagens: um processo envia uma mensagem pela rede destinada a um ou mais processos
  - Cliente-servidor: o servidor incorpora toda a lógica e os dados necessários para executar um serviço, que é executado por solicitação de um ou mais clientes remotos
  - Memória compartilhada: área de memória pode ser acessada por diversos processos rodando em diferentes máquinas

Paradigmas de Computação Distribuída

- Características dos serviços de comunicação:
  - Abrangência: local ou remota
  - Participantes: 1 → 1, 1 → N ou M → N
  - Tamanho das mensagens: fixo ou variável; limitado ou não
  - Sincronismo: comunicação síncrona, assíncrona ou semi-síncrona

Suporte Computacional para Aplicações Distribuídas

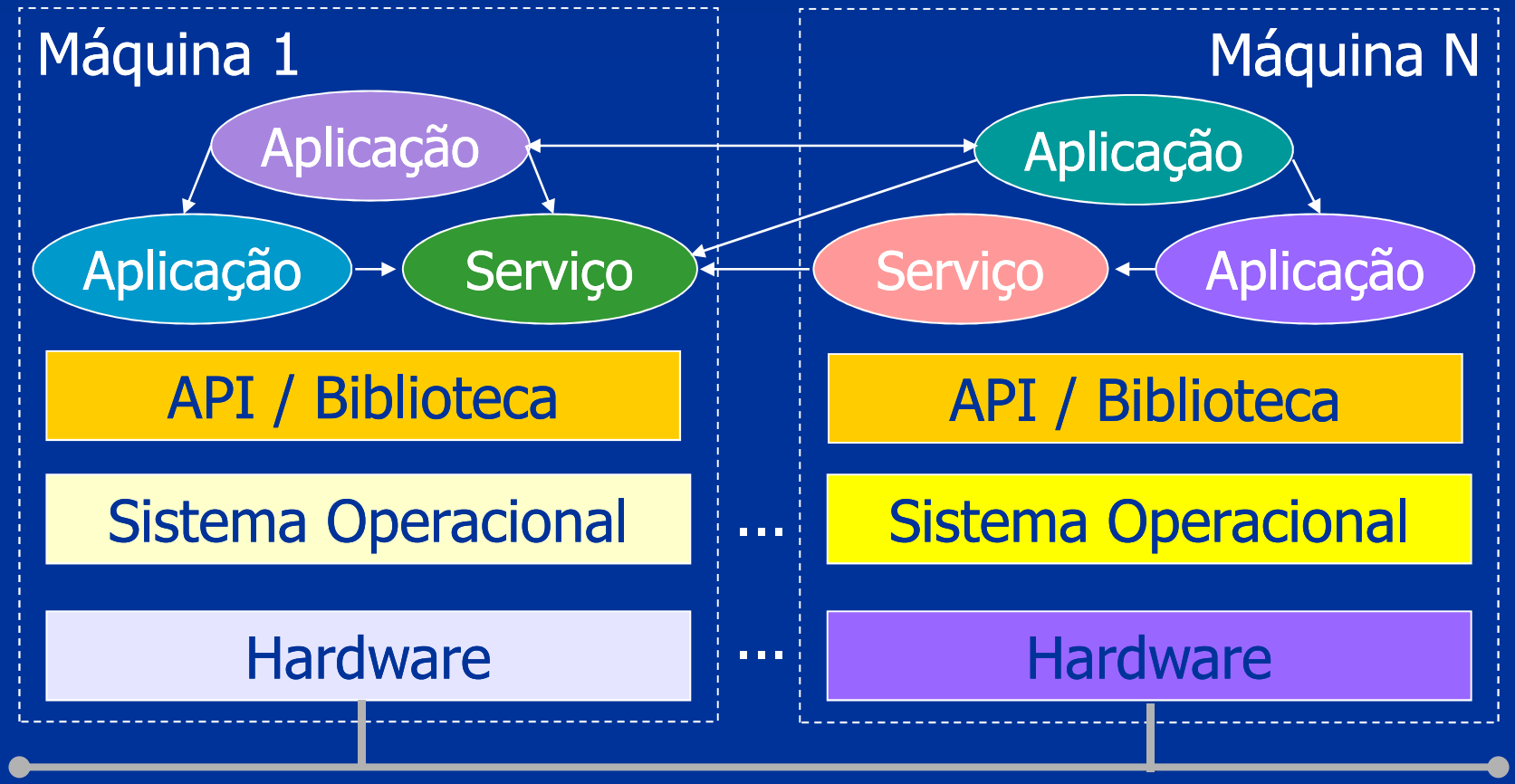
- Suportes para Computação Distribuída devem fornecer:
  - Mecanismos para execução distribuída de programas
  - Mecanismos para controle de concorrência
  - Mecanismos para comunicação entre processos
  - Ferramentas e mecanismos para desenvolvimento, testes, gerenciamento, controle, segurança, tolerância a faltas, etc.

Suporte Computacional para Aplicações Distribuídas

- Suporte para Computação Distribuída
  - APIs e Bibliotecas: fornecem rotinas para comunicação entre processos
    Ex.: UNIX Sockets, WinSock, java.net, etc.
  - Middleware para Programação Distribuída: fornece suporte para criar / executar programas distribuídos. Ex.: CORBA, COM, etc.
  - Servidores de Aplicação: permitem o acesso a aplicações via rede. Ex.: Tomcat, JBoss, etc.
  - Linguagens e sistemas operacionais distribuídos caíram em desuso por não suportarem heterogeneidade de ambiente

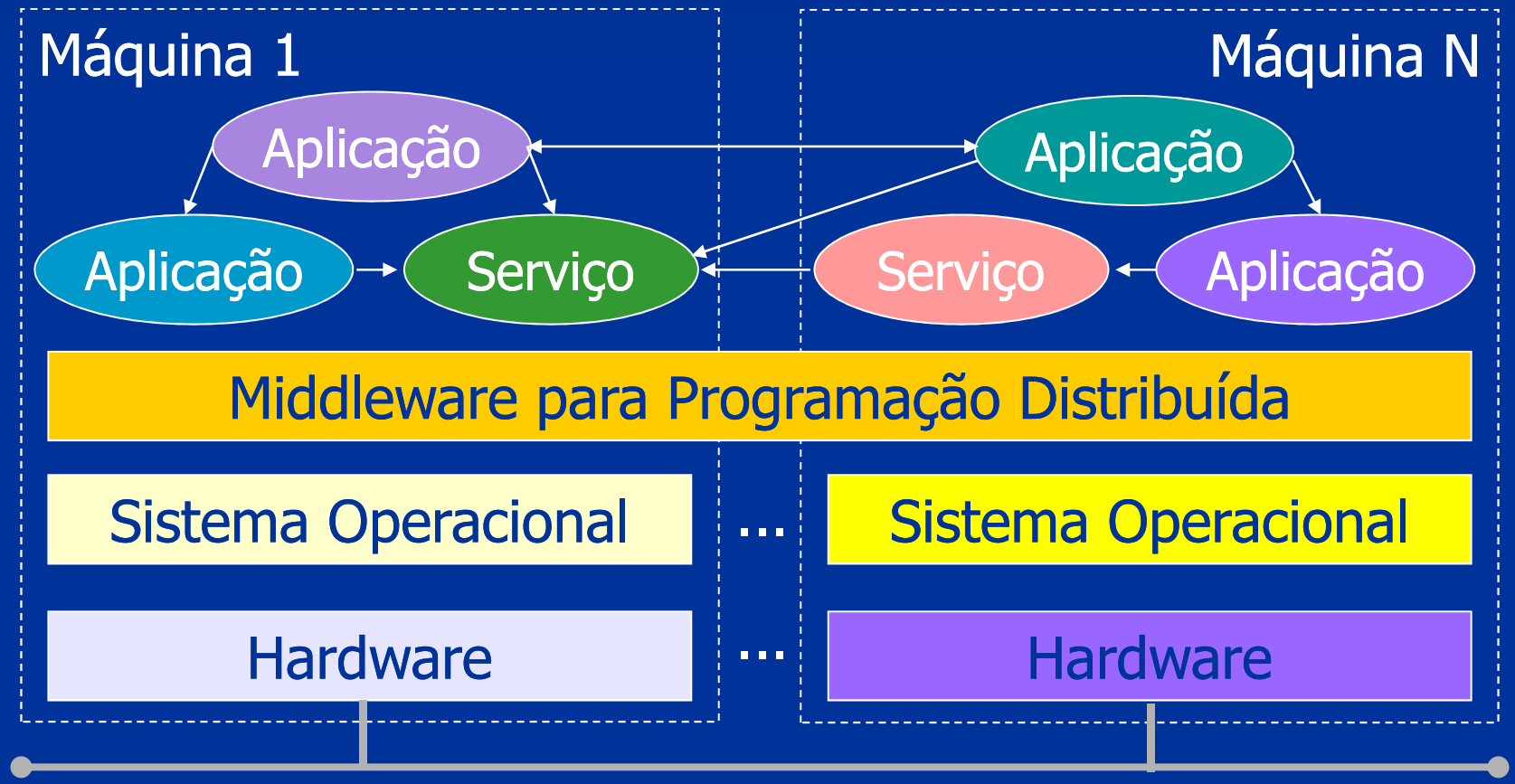
Suporte Computacional para Aplicações Distribuídas

- APIs e Bibliotecas para Comp. Distribuída



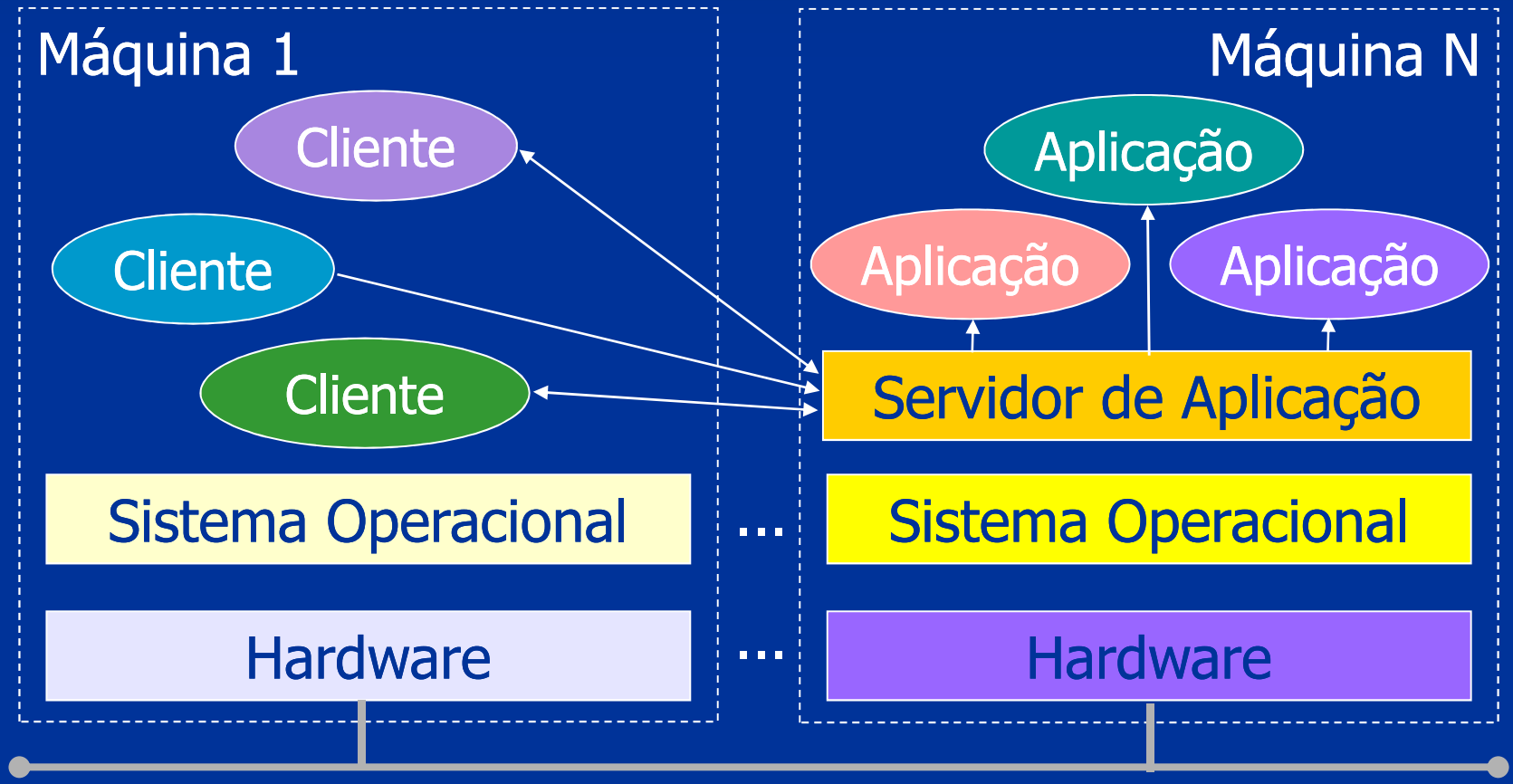
Suporte Computacional para Aplicações Distribuídas

- Middleware para Programação Distribuída



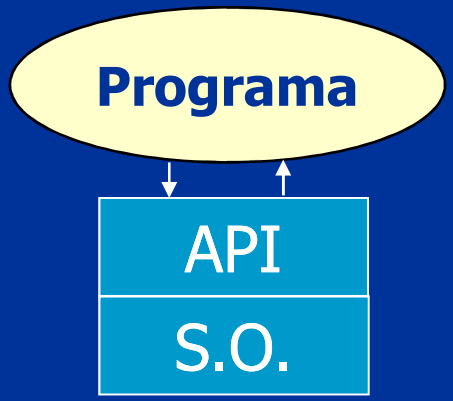
Suporte Computacional para Aplicações Distribuídas

- Servidor de Aplicação



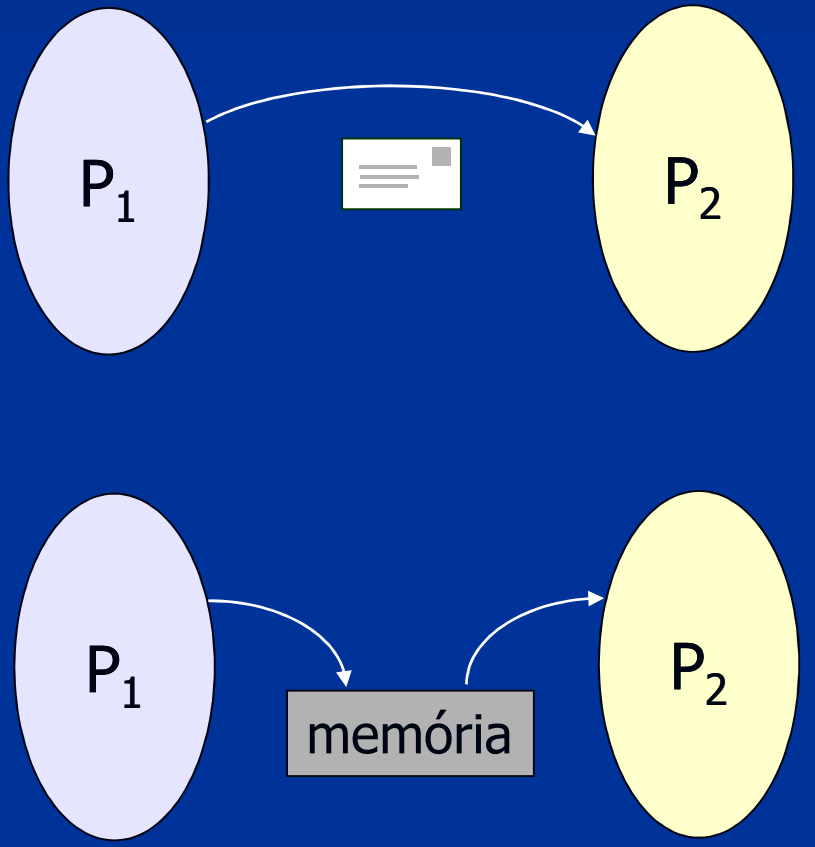
Suporte Computacional para Aplicações Distribuídas

- APIs de comunicação
  - Permitem que aplicações troquem dados
  - Fornecem primitivas de comunicação que podem ser chamadas a partir do código
  - Provêem acesso aos serviços de comunicação, que podem assim ser usados pelas aplicações



Suporte Computacional para Aplicações Distribuídas

- APIs de Comunicação de Sist. Operacionais
  - Mecanismos fornecidos pelos S.O.'s permitem enviar mensagens (trechos de memória) de um processo a outro
  - Alguns S.O.'s permitem que sejam criadas áreas de memória compartilhadas entre dois ou mais processos



Suporte Computacional para Aplicações Distribuídas

- Exemplos de APIs de comunicação:
  - Pipes: canais de comunicação locais
  - Sockets: portas de comunicação locais ou de rede (versão segura: SSL)
  - Suportes de RPC (*Remote Procedure Call*): permitem chamar procedimentos/métodos remotamente (ex.: RMI, CORBA, COM, ...)
  - Canais de eventos: permitem notificar threads e processos dos eventos ocorridos no sistema (Ex.: JMS, CORBA Notification Service, ...)
  - ...

Suporte Computacional para Aplicações Distribuídas

- Mecanismos de comunicação do UNIX
  - *Signals* e *Pipes* (comunicação local)
  - *Sockets*
- Mecanismos de comunicação do Windows
  - *Pipes* e *Mailslots* (comunicação local)
  - WinSock
  - Microsoft RPC
  - Memória compartilhada: *File Mapping* e *Dynamic Data Exchange* (DDE)
  - *Object Linking and Embedding* (OLE)
  - *Component Object Model* (COM)

Comunicação entre Processos

- Socket
  - Abstração que representa uma porta de comunicação bidirecional associada a um processo
- Principais Tipos de Socket
  - Socket Datagrama: envia/recebe datagramas sem criar conexão; usa protocolo UDP
  - Socket Multicast: recebe as mensagens endereçadas a um grupo; usa UDP multicast
  - Socket Stream: estabelece uma conexão com outro socket; usa protocolo TCP

Comunicação entre Processos

- Operações com Sockets Datagrama
  - Criar um socket datagrama:
    DatagramSocket s = new DatagramSocket(porta);
  - Criar pacotes de dados para envio:
    DatagramPacket pack = new DatagramPacket(msg, tamanho, destino, porta);
  - Enviar dados: s.send(pack);
  - Criar pacotes de dados para recepção:
    DatagramPacket pack = new DatagramPacket(msg,tam);
  - Receber dados: s.receive(pack);
  - Ler dados do pacote: pack.getData()

Comunicação entre Processos

- Sockets Datagrama – Envio

```
try {
   DatagramSocket socket = new DatagramSocket();
   InetAddress destino = InetAddress.getByName(
      "127.0.0.1");
   String mensagem = "Hello";
   byte[] dados = mensagem.getBytes();
   int porta = 5000;
   DatagramPacket pacote = new DatagramPacket(
      dados, dados.length, destino, porta);
   socket.send(pacote);
} catch (SocketException e) { e.printStackTrace();
} catch (IOException e) {e.printStackTrace(); }
```

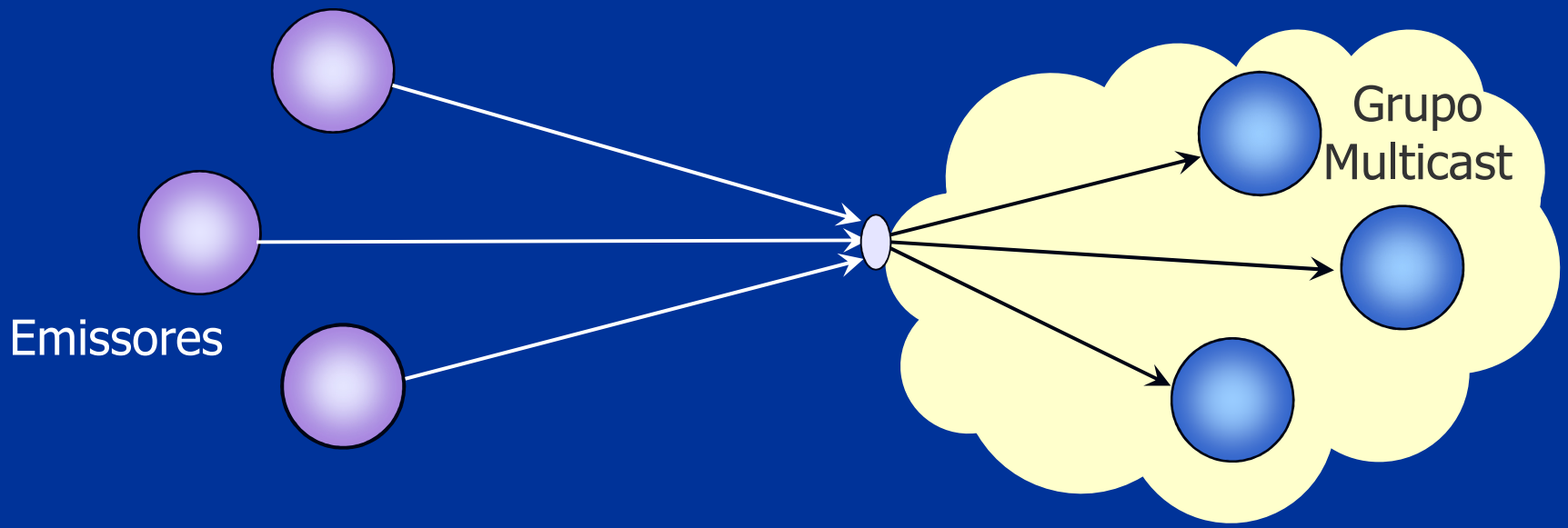
Comunicação entre Processos

- Sockets Datagrama – Recepção

```
try {
   int porta = 5000;
   DatagramSocket socket = new DatagramSocket(porta);
   byte[] dados = new byte[100];
   DatagramPacket pacote = new DatagramPacket(
      dados, dados.length);
   socket.receive(pacote);
   String mensagem = new String(pacote.getData(), 0,
      pacote.getLength() );
   System.out.println("Mensagem: " + mensagem);
} catch (SocketException e) { e.printStackTrace();
} catch (IOException e) {e.printStackTrace(); }
```

Comunicação entre Processos

- Sockets Multicast
  - Permitem o envio simultâneo de datagramas a grupos de destinatários
  - Grupos multicast são identificados por endereços IP de 224.0.0.0 a 239.255.255.255



Comunicação entre Processos

- Sockets Multicast
  - Vários emissores podem mandar mensagens para o grupo (ou seja, mensagens vão de X emissores $\rightarrow$ Y receptores)
  - Emissor não precisa fazer parte do grupo para enviar mensagens ao grupo, e nem precisa saber quem são os seus membros; basta conhecer o endereço IP do grupo
  - O receptor entra em um grupo (se torna um membro do grupo) e passa a receber as mensagens destinadas ao grupo

Comunicação entre Processos

- Sockets Multicast – Envio

```
try {
   DatagramSocket socket = new DatagramSocket();
   InetAddress grupo = InetAddress.getByName(
      "230.1.2.3");
   String mensagem = "Hello";
   byte[] dados = mensagem.getBytes();
   int porta = 5000;
   DatagramPacket pacote = new DatagramPacket(
      dados, dados.length, grupo, porta);
   socket.send(pacote);
} catch (SocketException e) { e.printStackTrace();
} catch (IOException e) { e.printStackTrace(); }
```

Comunicação entre Processos

- Sockets Multicast – Recepção

```
try {
   int porta = 5000;
   MulticastSocket msocket = new MulticastSocket(porta);
   InetAddress grupo = InetAddress.getByName(
      "230.1.2.3");
   msocket.joinGroup(grupo);
   byte[] dados = new byte[100];
   DatagramPacket pacote = new DatagramPacket(
      dados, dados.length);
   msocket.receive(pacote);
   System.out.println("Mensagem: " +
      new String(pacote.getData(), 0, pacote.getLength()));
} catch (Exception e) {e.printStackTrace(); }
```

Comunicação entre Processos

- Sockets Stream
  - Estabelecem canais de comunicação entre aplicações, permitindo troca de dados pela rede
  - Adotam o paradigma cliente-servidor
    - Cliente: pede para conectar ao servidor
    - Servidor: aguarda conexões dos clientes

Comunicação entre Processos

- Operações com Sockets Stream no Servidor
  - Criar um socket servidor:
    ServerSocket s = new ServerSocket(porta, maxClientes);
  - Aguardar conexão: Socket c = s.accept();
  - Obter nome do host conectado:
    String host = c.getInetAddress().getHostName();
  - Criar fluxos de comunicação:
    ObjectInputStream in = new
    ObjectInputStream(c.getInputStream());
    ObjectOutputStream out = new
    ObjectOutputStream(c.getOutputStream());
  - Fechar conexão: in.close(); out.close(); c.close();

Comunicação entre Processos

- Operações com Sockets Stream no Cliente
  - Criar um socket cliente:
    Socket c = new Socket(InetAddress.
    getByName("*servidor.com* "), porta);
  - Criar fluxo, enviar e receber dados, e fechar: idem ao servidor
- Exceções geradas
  - SocketException
  - UnknownHostException
  - IOException

Comunicação entre Processos

- Sockets Stream – Servidor

```
try {
   ServerSocket s = new ServerSocket(5000, 10);
   Socket c = s.accept();
   ObjectOutputStream out = new ObjectOutputStream(
      c.getOutputStream());
   output.flush();
   ObjectInputStream input = new ObjectInputStream(
      c.getInputStream() );
   String mensagem = (String) input.readObject();
   String resposta = "Olá Cliente";
   output.writeObject(resposta);
   output.flush();
} catch (Exception e) { e.printStackTrace(); }
```

Comunicação entre Processos

- Sockets Stream – Cliente

```
try {
   Socket socket = new Socket(InetAddress.getByName(
      "127.0.0.1"), 5000);
   ObjectOutputStream out = new ObjectOutputStream(
      socket.getOutputStream());
   output.flush();
   ObjectInputStream input = new ObjectInputStream(
      socket.getInputStream() );
   String mensagem = "Olá Servidor";
   output.writeObject(mensagem);
   output.flush();
   String resposta = (String) input.readObject();
} catch (Exception e) { e.printStackTrace(); }
```

Sistemas de Arquivos Distribuídos

- Definições
  - Um sistema de arquivos distribuído permite que arquivos de computadores remotos sejam acessados como se estivessem armazenados no computador local
  - Um computador da rede compartilha um diretório (pasta) ou volume, que pode ser montado em outro computador da rede, passando a ser visto como uma unidade de disco (Windows) ou como um diretório do sistema de arquivos (UNIX/Linux)

Sistemas de Arquivos Distribuídos

- Vantagens
  - Permite que o usuário acesse seus arquivos em qualquer computador de uma rede
  - Facilita o compartilhamento de arquivos entre os usuários da rede
  - Simplifica a administração dos computadores da rede, provendo um ponto central para armazenamento de dados e programas
  - Pode ser usado para efetuar *backup* remoto de arquivos locais

Sistemas de Arquivos Distribuídos

- Requisitos desejáveis
  - Transparência de acesso e de localização
  - Funcionalidades e desempenho comparáveis a um sistema de arquivos convencional
  - Controle de acesso
  - Controle de concorrência
  - Heterogeneidade de hardware e SO
  - Consistência
  - Escalabilidade
  - Tolerância a falhas

Sistemas de Arquivos Distribuídos

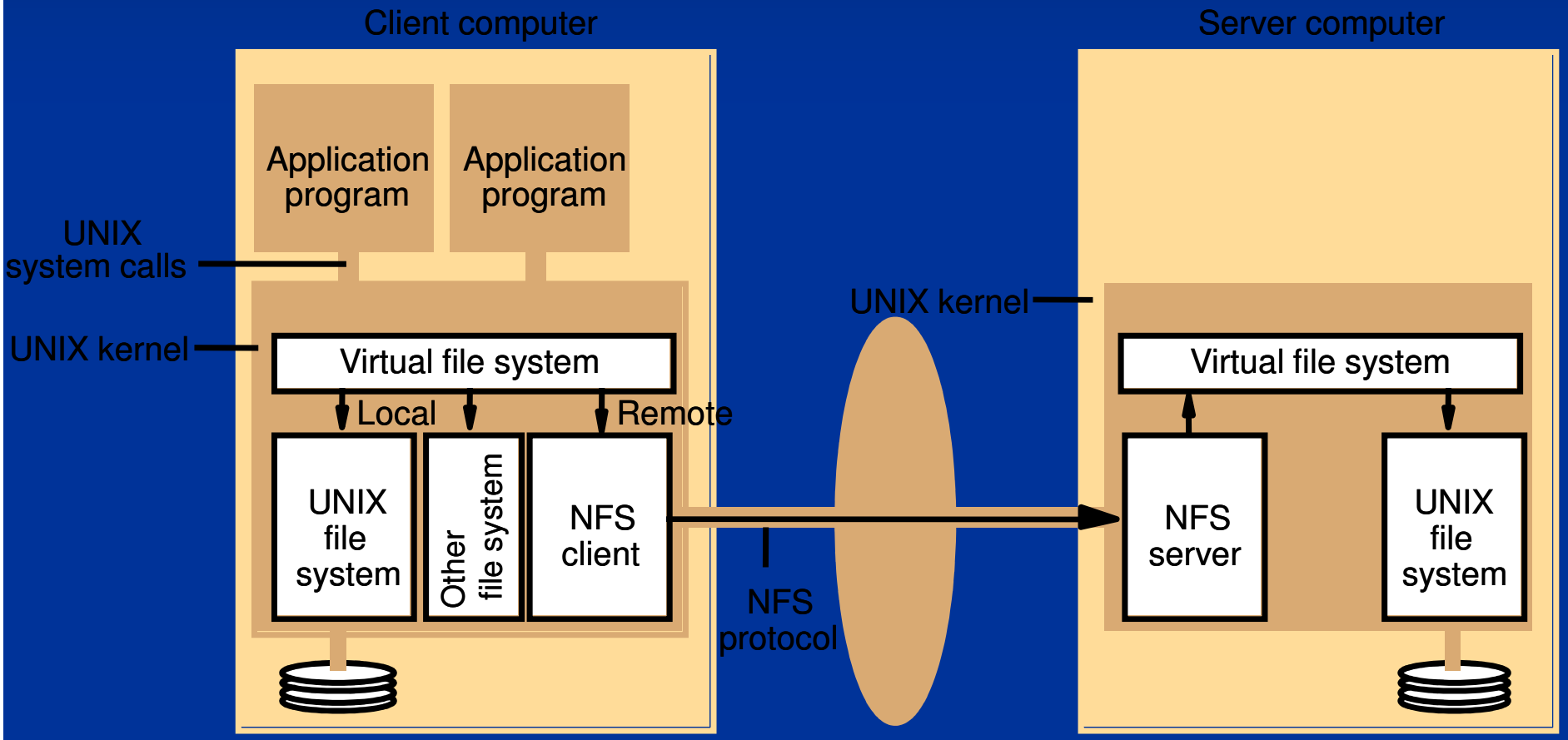
- Acesso a arquivos
  - As mesmas chamadas de sistema ou classes da API usadas para acesso a arquivos locais são utilizadas no acesso a arquivos remotos
  - No UNIX/Linux: chamadas de sistema *open*, *close*, *read*, *write*, etc.
  - No Windows: chamadas de sistema *CreateFile*, *ReadFile*, *WriteFile*, *CloseHandle*, etc.
  - Em Java: classes da API *FileInputStream*, *FileOutputStream*, *FileReader*, *FileWriter*, etc.

Sistemas de Arquivos Distribuídos

Módulo de Diretório	• Relaciona nomes com IDs de arquivos
Módulo de Arquivo	• Relaciona IDs de arquivos com arquivos físicos
Módulo de Controle de Acesso	• Verifica se o usuário possui permissão para efetuar a operação solicitada
Módulo de Acesso a Arquivo	• Lê ou escreve dados ou atributos de um arquivo
Módulo de Bloco	• Acessa e aloca blocos de disco
Módulo de Dispositivo	• Efetua operações de entrada e saída no disco • Gerencia o uso de *buffers*

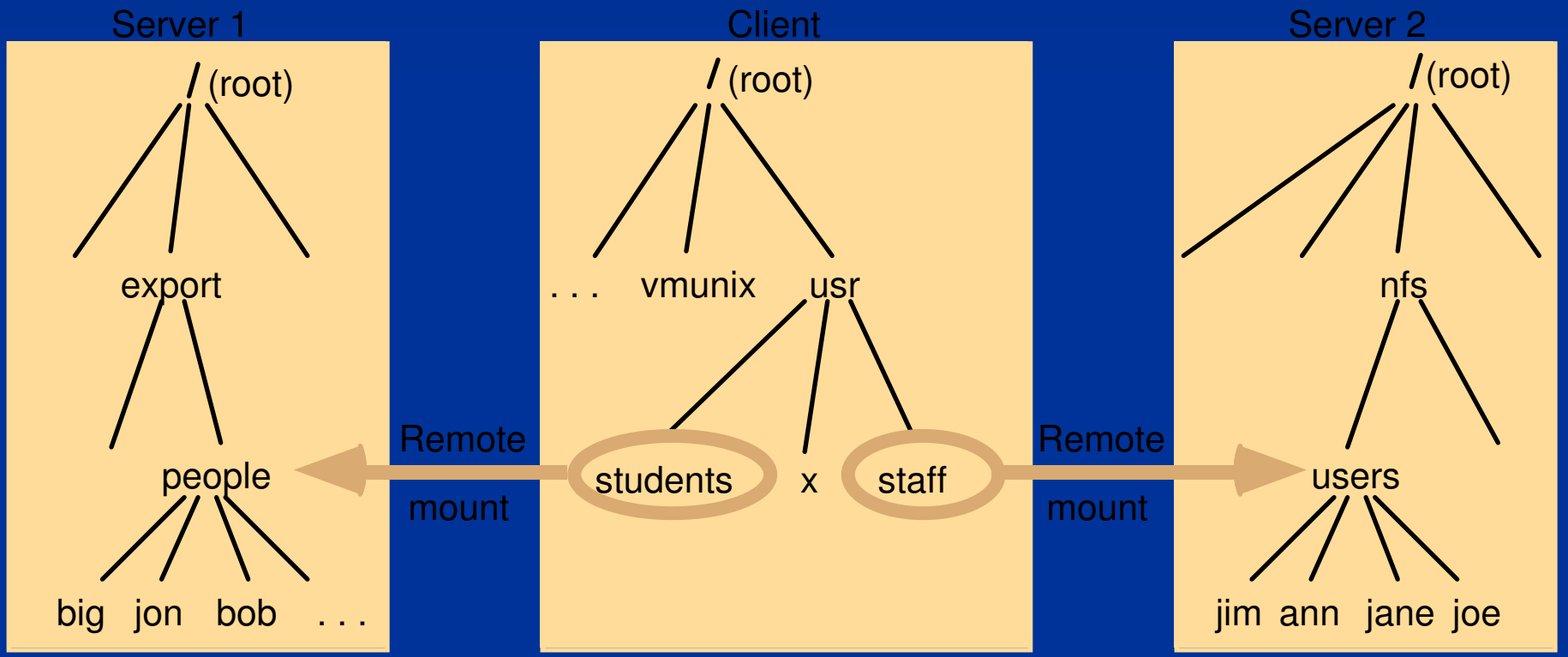
Sistemas de Arquivos Distribuídos

- *Sun Network File System* (NFS)



Sistemas de Arquivos Distribuídos

- Exportação e montagem de diretórios NFS



Sistemas de Arquivos Distribuídos

- Configuração do NFS
  - */etc/exports* : contém os diretórios exportados pela máquina local para as demais máquinas da rede
  - */etc/fstab* : lista os diretórios remotos que devem ser montados na máquina local
- Comandos do NFS
  - *exportfs*
  - *mount*

Sistemas de Arquivos Distribuídos

- Evolução do NFS
  - NFSv1: versão de desenvolvimento, usada apenas internamente na Sun
  - NFSv2: 1º release; comunicação entre cliente e servidor através de UDP (não confiável)
  - NFSv3: adicionou suporte para comunicação através de TCP (confiável), para uso em WANs
  - NFSv4: atual; desenvolvida no âmbito da IETF; suporta a replicação de volumes de dados

Sistemas de Arquivos Distribuídos

- *Andrew File System* (AFS)
  - Desenvolvido com o intuito de permitir a replicação de volumes de dados
  - Criado na Univ. de Carnegie Mellon, USA
  - Tornou-se um produto da Transarc, que posteriormente foi adquirida pela IBM
  - IBM deixou de dar suporte ao produto e o disponibilizou como software livre (OpenAFS)

Sistemas de Arquivos Distribuídos

- *Server Message Block* (SMB) / *Common Internet File System* (CIFS)
  - Suportado nativamente no Windows para compartilhamento de arquivos, periféricos, etc.
  - Implementado também no UNIX/Linux por meio do serviço Samba
  - Pode ser construído sobre diversos protocolos de comunicação, como NetBIOS, TCP/IP, etc.