# Redes de Computadores e Sistemas Distribuídos


*Prof. Me. Wallace Rodrigues de Santana*

*www.neutronica.com.br*

# Atribuição-NãoComercial-CompartilhaIgual 3.0 Brasil (CC BY-NC-SA 3.0)

## Você tem a liberdade de:

**Compartilhar** — copiar, distribuir e transmitir a obra.

**Remixar** — criar obras derivadas.

## Sob as seguintes condições:

**Atribuição** — Você deve creditar a obra da forma especificada pelo autor ou licenciante (mas não de maneira que sugira que estes concedem qualquer aval a você ou ao seu uso da obra).

**Uso não comercial** — Você não pode usar esta obra para fins comerciais.

**Compartilhamento pela mesma licença** — Se você alterar, transformar ou criar em cima desta obra, você poderá distribuir a obra resultante apenas sob a mesma licença, ou sob uma licença similar à presente.

## Ficando claro que:

**Renúncia** — Qualquer das condições acima pode ser **renunciada** se você obtiver permissão do titular dos direitos autorais.

**Domínio Público** — Onde a obra ou qualquer de seus elementos estiver em **domínio público** sob o direito aplicável, esta condição não é, de maneira alguma, afetada pela licença.

**Outros Direitos** — Os seguintes direitos não são, de maneira alguma, afetados pela licença:

- Limitações e exceções aos direitos autorais ou quaisquer **usos livres** aplicáveis;
- Os **direitos morais** do autor;
- Direitos que outras pessoas podem ter sobre a obra ou sobre a utilização da obra, tais como **direitos de imagem** ou privacidade.

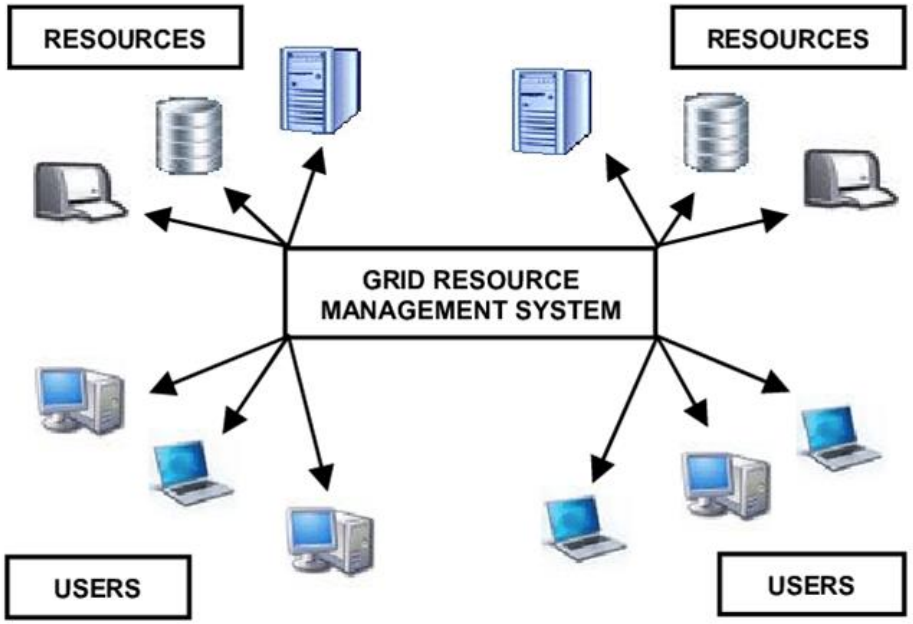
**Aviso** — Para qualquer reutilização ou distribuição, você deve deixar claro a terceiros os termos da licença a que se encontra submetida esta obra. A melhor maneira de fazer isso é com um link para esta página.

# Módulo 13

Introdução a Sistemas Distribuídos

# Definição

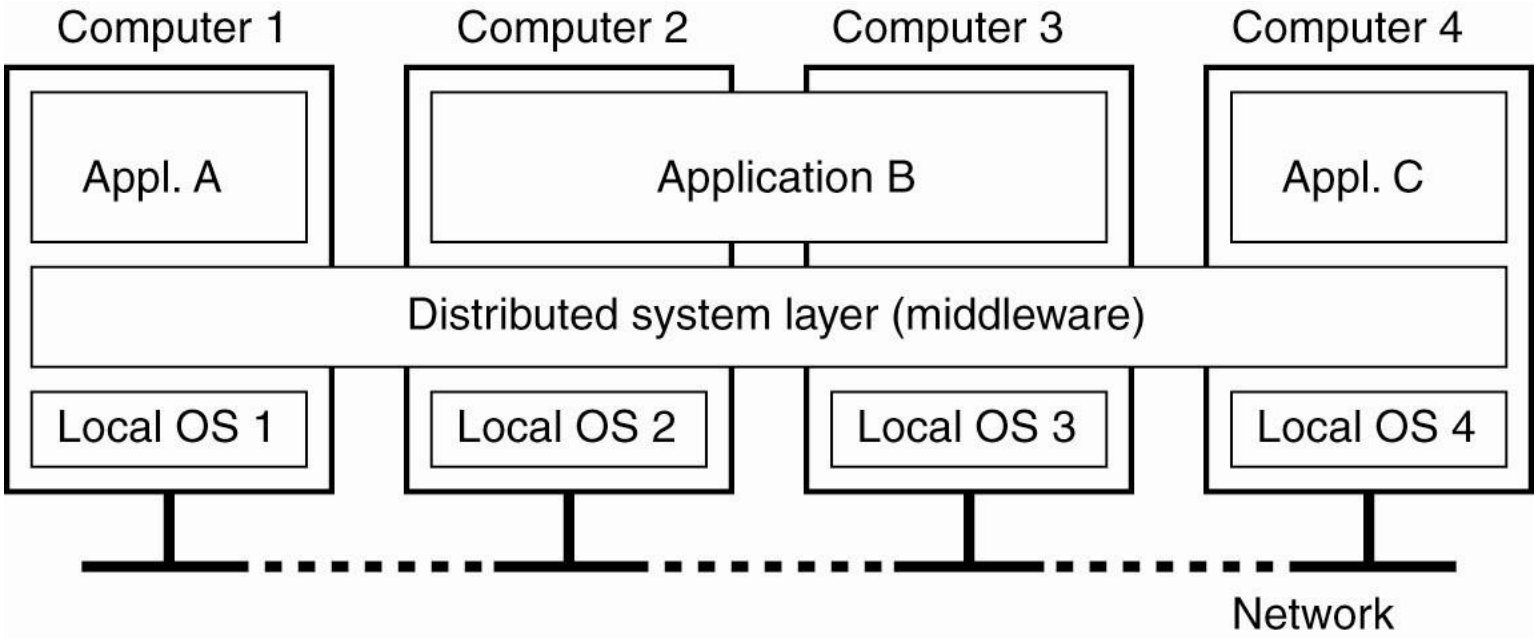
*"Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e consistente."*




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Organização

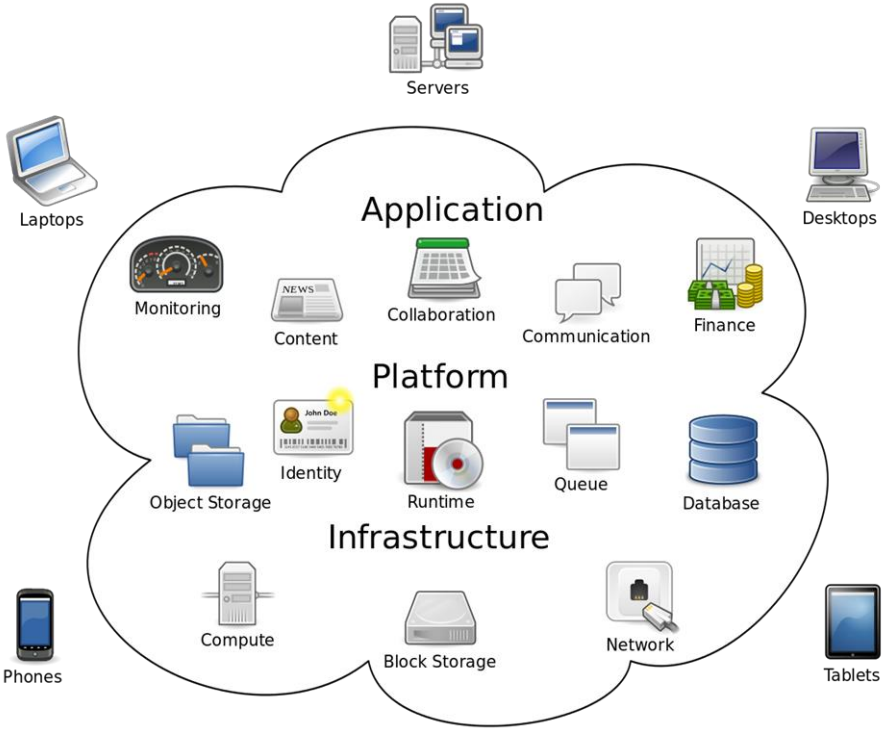
Um sistema distribuído é composto por uma camada de usuários e aplicações, uma segunda camada de middleware e uma terceira camada composta por sistema operacional e rede.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – acesso a recursos

Facilitar aos usuários e às aplicações o acesso a recursos remotos e seu compartilhamento de maneira controlada e eficiente.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – transparência

Quando um sistema distribuído oculta o fato de que seus processos e recursos estão fisicamente distribuídos por vários computadores.

| | |
|---|---|
| Acesso | •Oculta diferenças na representação de dados e no modo de acesso a um recurso. |
| Localização | •Oculta o lugar em que um recurso está localizado. |
| Migração | •Oculta que um recurso pode ser movido para outra localização. |
| Relocação | •Oculta que um recurso pode ser movido para uma outra localização enquanto em uso. |
| Replicação | •Oculta que um recuso é replicado. |
| Concorrência | •Oculta que um recurso pode ser compartilhado por diversos usuários concorrentes. |
| Falha | •Oculta a falha e a recuperação de um recurso. |

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – abertura

Um sistema distribuído aberto é aquele que oferece serviços de acordo com regras padronizadas que descrevem a sintaxe e a semântica desses serviços.

| | |
|---|---|
| Interoperabilidade | • Caracteriza até que ponto duas implementações de sistemas ou componentes de fornecedores diferentes podem coexistir e trabalhar em conjunto. |
| Portabilidade | • Caracteriza até que ponto uma aplicação desenvolvida para um dado sistema distribuído pode ser executada, sem modificação, em um sistema distribuído diverso que implementa as mesmas interfaces do primeiro. |

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – escalabilidade

É a capacidade de um sistema atender e gerenciar uma demanda crescente de processamento.

Pode ser medida em três dimensões:

| | |
|---|---|
| Tamanho | • Caracteriza um sistema distribuído que pode facilmente adicionar mais usuários ou recursos. |
| Geográfica | • Caracteriza um sistema distribuído no qual usuários e recursos podem estar dispersos geograficamente. |
| Administrativa | • Caracteriza um sistema distribuído que continua sendo fácil de gerenciar. |

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – escalabilidade

A escalabilidade pode encontrar os seguintes problemas ao ser implementada:

- Dimensão tamanho: serviços centralizados são difíceis de serem distribuídos em diversas máquinas; dados centralizados que precisem ser distribuídos em máquinas diferentes podem ter problemas de replicação e consistência; e por último e não menos importante é a dificuldade em manter o sincronismo dos relógios;
- Dimensão geográfica: a comunicação de dados em redes de longa distância tem problemas de latência;
- Dimensão administrativa: ao abranger muitas unidades organizacionais diferentes, talvez seja necessário delegar tarefas de administração, o que pode acarretar problemas de segurança.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Objetivos – escalabilidade

A escalabilidade pode ser alcançada usando-se as seguintes técnicas:

- Comunicação assíncrona: em redes de longa distância a latência pode ser um problema. Aplicações cliente/servidor em geral usam comunicação síncrona, onde o cliente envia uma requisição e aguarda a resposta do servidor para continuar o processamento. Usar comunicação assíncrona significa construir uma aplicação que possa continuar o processamento sem precisar ficar esperando respostas às suas requisições, como por exemplo em sistemas de processamento em lote e aplicações paralelas;
- Distribuição: envolve tomar um componente, subdividi-lo em partes menores e distribuí-la. Um exemplo é o Sistema de Nomes de Domínio DNS, que é organizado em uma árvore de domínios dividida em zonas sem sobreposição;

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

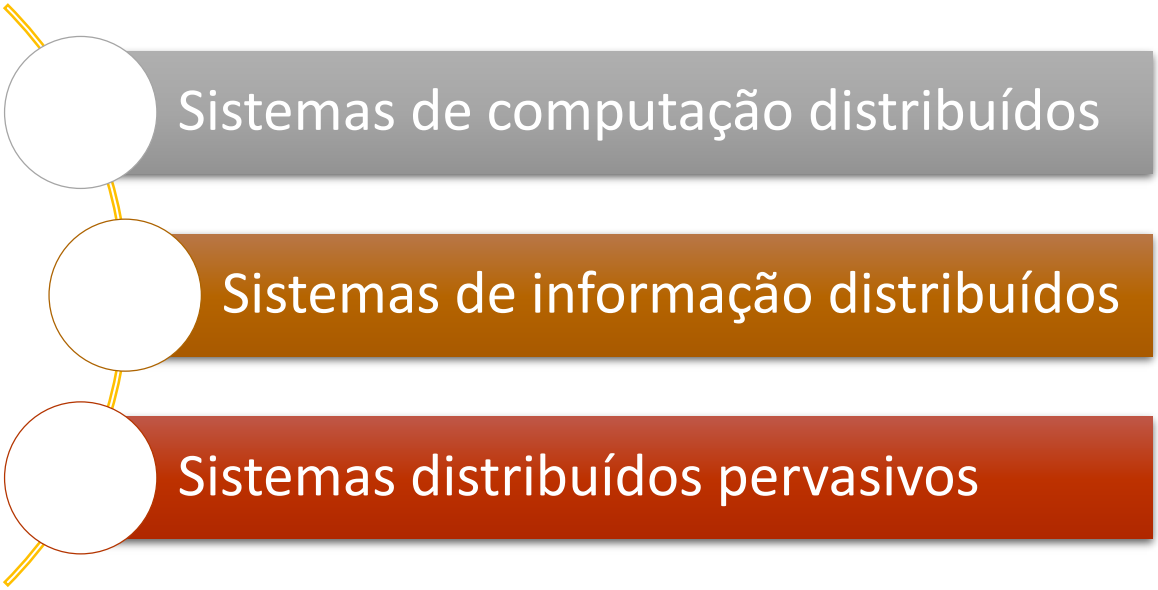
# Objetivos – escalabilidade

- Replicação: a cópia e distribuição de recursos aumenta a disponibilidade e ajuda a equilibrar a carga entre os componentes. A replicação pode se apresentar na forma de cache, que é a cópia de um recurso na proximidade do cliente.
- Consistência: é o que garante que as cópias de recursos em esquemas de replicação ou cache estejam consistentes, ou seja, se um componente for alterado ou atualizado, as novas informações devem ser imediatamente propagadas pela rede.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos

Os sistemas distribuídos pode ser do tipo:

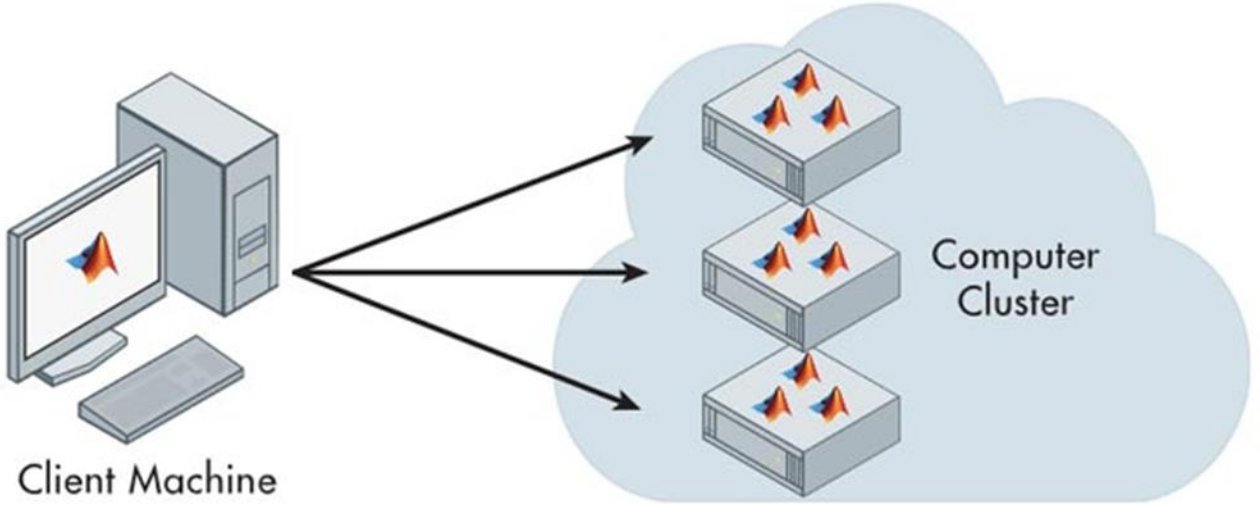



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de computação distribuídos

Os sistemas de computação distribuídos podem ser do tipo:

- Cluster: composto de servidores de configuração semelhante, que executam o mesmo sistema operacional e que estão conectados por meio de uma rede local de alta velocidade. É um sistema homogêneo;




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de computação distribuídos

- Grid: uma federação de computadores geograficamente distribuídos e em domínios administrativos diferentes, que podem ser compostos de servidores ou máquinas de diferentes tipos e sistemas operacionais diversos. É um sistema heterogêneo.


Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos

Os sistemas de informação distribuídos caracterizam-se por fornecerem às organizações um ambiente cliente/servidor onde a integração das aplicações e a interoperabilidade são um desafio.

O tipo mais comum de aplicações neste tipo de sistemas distribuídos são aquelas que lidam com o processamento de transações, em especial os sistemas baseados em banco de dados.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos

Sistemas de processamento de transações possuem quatro propriedades características, denotadas pela sigla ACID.


Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos


| | |
|---|---|
| Atomicidade | • Para o mundo exterior, a transação acontece como se fosse indivisível. |
| Consistência | • A transação não viola invariantes do sistema. |
| Isolamento | • Transações concorrentes não interferem umas com as outras. |
| Durabilidade | • Uma vez comprometida uma transação, as alterações são permanentes. |

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos

Exemplo de processamento de transações com MySQL:

```
mysql> CREATE TABLE Cliente (iD INT, Nome CHAR (20), INDEX (iD));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO Cliente VALUES (10, 'Ana');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Fonte: dev.mysql.com

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos

```
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO Cliente VALUES (15, 'Joao');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO Cliente VALUES (20, 'Paulo');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM Cliente WHERE Nome = 'Ana';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

Fonte: dev.mysql.com

# Tipos de sistemas distribuídos
# Sistemas de informação distribuídos

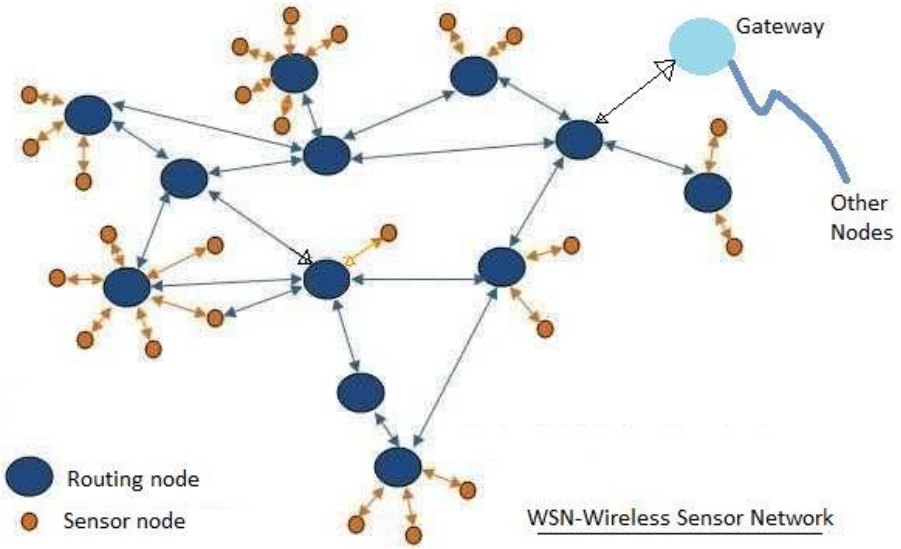
```
mysql> SELECT * FROM Cliente;
+------+--------+
| iD   | Nome   |
+------+--------+
|   10 | Ana    |
+------+--------+
1 row in set (0.00 sec)
```

Fonte: dev.mysql.com

# Tipos de sistemas distribuídos
# Sistemas distribuídos pervasivos

Sistemas distribuídos pervasivos são aqueles formados por dispositivos de computação móveis e embutidos.

Um exemplo de sistema distribuído pervasivo são as redes de sensores, compostas por dispositivos de tamanho reduzido, alimentados por bateria e que usam comunicação sem fio para propagar as informações coletadas.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Para saber mais...

… leia o Capítulo 1 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Módulo 14

Arquitetura de Sistemas Distribuídos

# Estilos de arquitetura

Um sistema distribuído é composto por complexas peças de software espalhadas por várias máquinas. Para controlar e administrar esta complexidade, se faz necessário que estes sistemas sejam organizados adequadamente.

Estilos de arquitetura referem-se ao modo como os componentes de hardware e software estão conectados uns aos outros, dos dados trocados entre os componentes e a maneira como esses elementos são configurados em conjunto para formar um sistema.

Um **componente** pode ser entendido como uma unidade modular com interfaces requeridas e fornecidas bem definidas e que é substituível dentro do seu ambiente.

Já um **conector** é descrito como um mecanismo que serve de mediador da comunicação ou cooperação entre os componentes.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Estilos de arquitetura

Usando componentes e conectores, pode-se implementar um dos seguintes estilos de arquitetura de sistemas distribuídos:

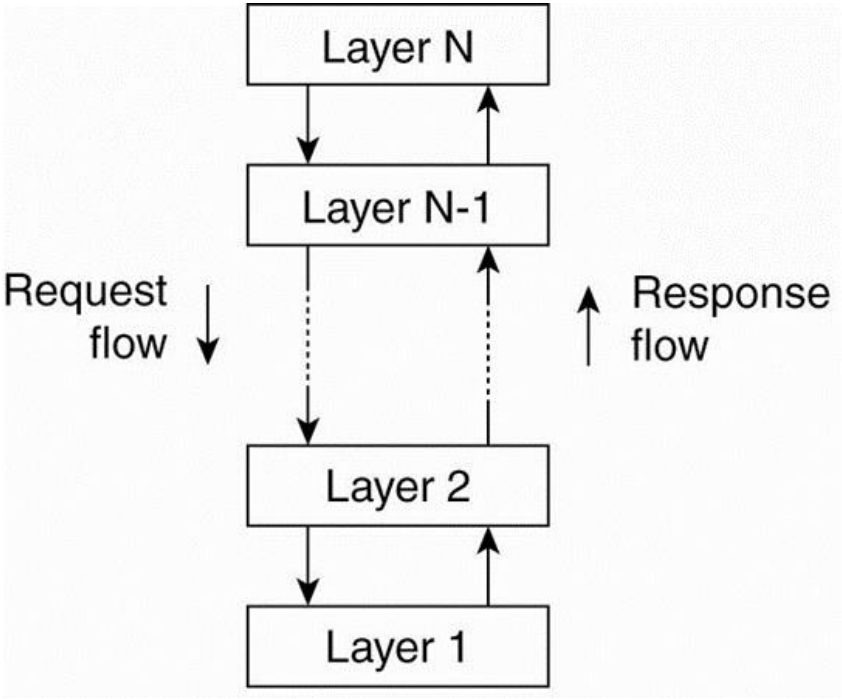
- Em camadas;
- Baseada em objetos;
- Centrada em dados;
- Baseada em eventos.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Estilos de arquitetura
# Em camadas

O estilo de arquitetura em camadas é um tipo de organização de onde um componente da camada Li tem permissão de chamar componentes na camada subjacente Li-1, mas não o contrário.

Em geral, o controle flui de camada para camada, onde as requisições descem pela hierarquia e as respostas fluem para cima.

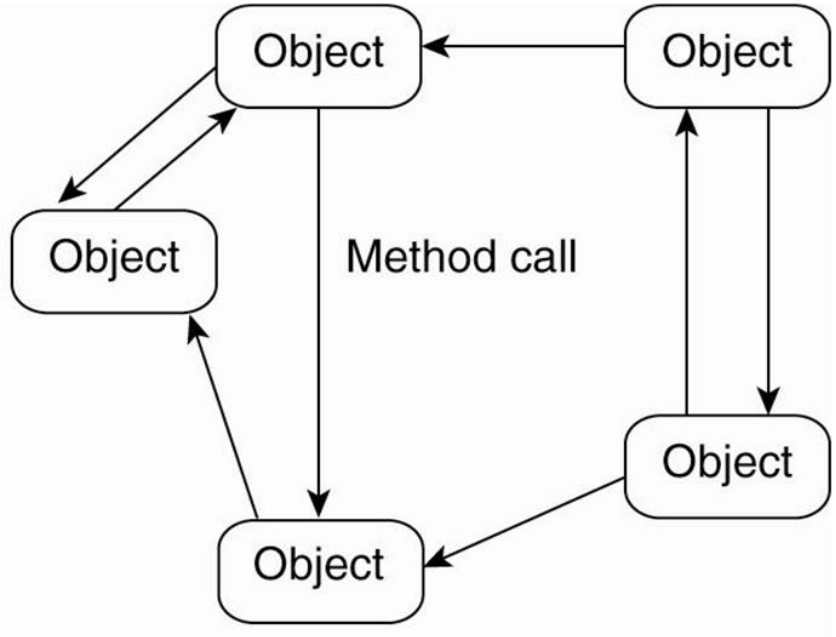



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Estilos de arquitetura Baseada em objetos

O estilo de arquitetura baseado em objetos é formado por componentes que são conectados por meio de uma chamada de procedimento remota.

Em geral, é o tipo de arquitetura que melhor se ajusta a sistemas cliente/servidor.



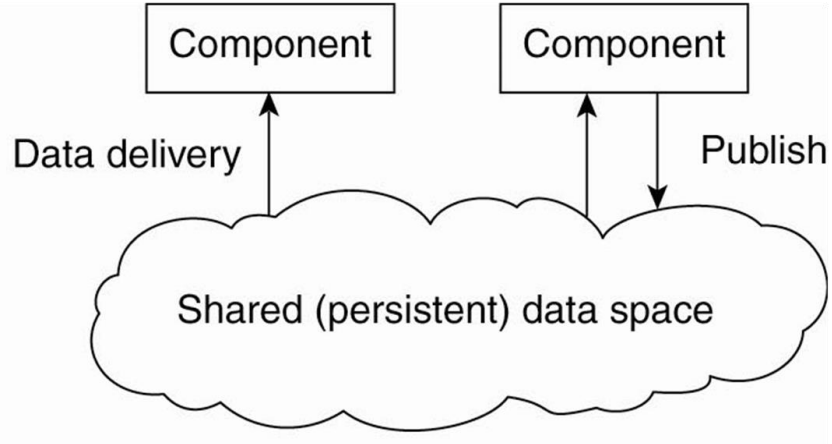

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Estilos de arquitetura Centrada em dados

O estilo de arquitetura centrado em dados é baseado em processos que se comunicam por meio de um repositório comum.

Este repositório comum, que pode ser um banco de dados, por exemplo, elimina a necessidade de movimentar informações entre sistemas legados e novos.

Essa abordagem permite também que as aplicações possam ser construídas e alteradas enquanto os dados são mantidos intactos.

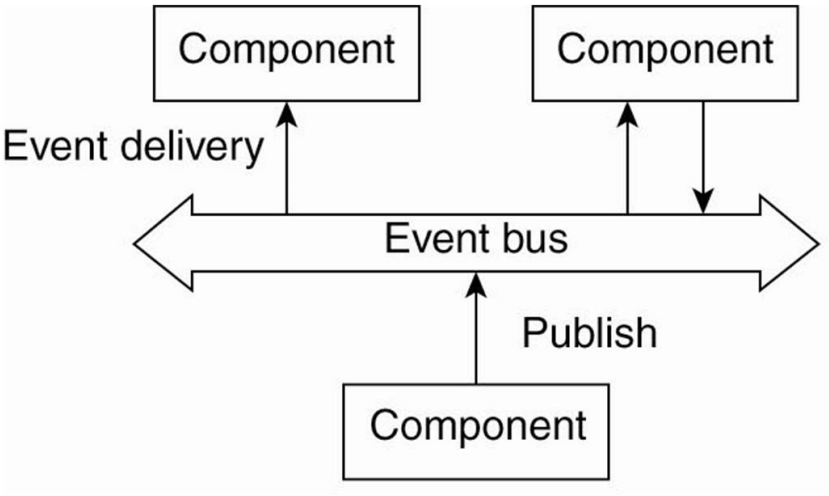



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Estilos de arquitetura Baseada em eventos

O estilo de arquitetura baseado em eventos permite que os processos se comuniquem por meio da propagação de eventos, também conhecido como sistemas publicar/subscrever, e que opcionalmente podem transportar dados também.

Quando um componente publica um evento, o middleware assegura que somente os demais componentes que se subscreveram para receber este evento específico possam acessá-lo, fazendo com que neste estilo de arquitetura os processos sejam fracamente acoplados.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas

Arquitetura de sistemas refere-se a como os sistemas distribuídos são realmente organizados, considerando-se onde são colocados os elementos de hardware e software e demais componentes e como eles interagem entre si.

As arquiteturas de sistemas podem ser do tipo:

- Centralizadas;
- Descentralizadas;
- Híbridas.

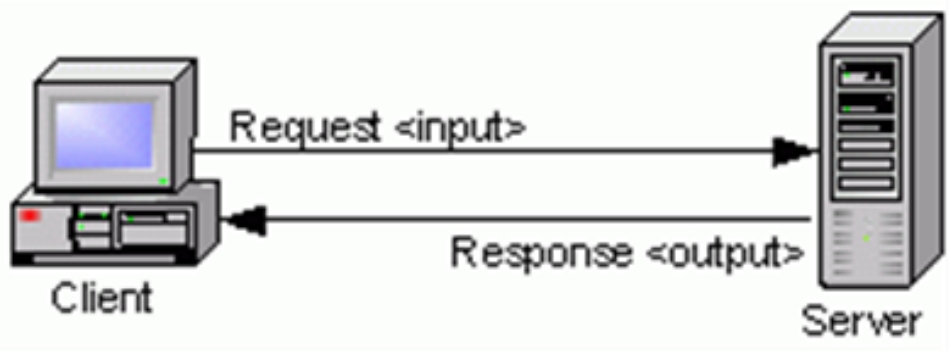
Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

Arquitetura de sistemas do tipo centralizada é aquela em que os clientes requisitam serviços a um ou mais servidores.

O servidor é um processo que implementa um serviço específico. Já o cliente é um processo que requisita um serviço de um servidor enviando-lhe uma requisição e em seguida aguarda por uma resposta do servidor.

Esse tipo de interação é conhecido como comportamento requisição-resposta.



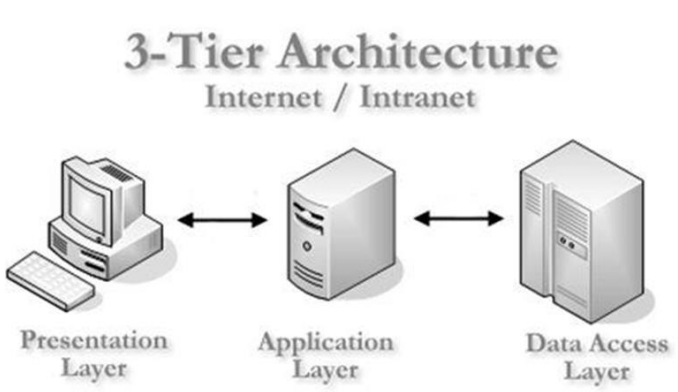

Arquitetura de sistemas do tipo centralizada também é conhecida como arquitetura de distribuição vertical.

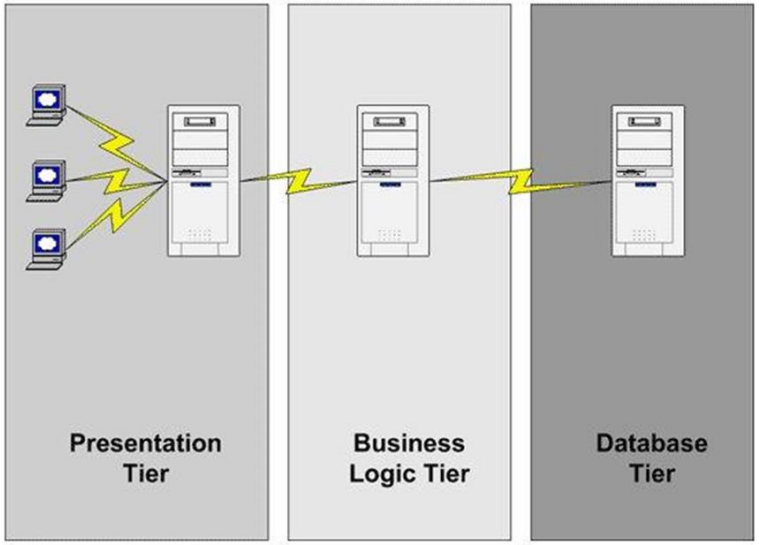
Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

Eventualmente, um servidor que atende requisições de clientes também pode ser cliente de um outro servidor.

Em ambientes multicamadas, por exemplo, um servidor de aplicação também pode ser cliente de um servidor de banco de dados.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

Em um sistema multicamadas, aplicações cliente/servidor procuram dar suporte ao acesso de usuários a banco de dados, por exemplo, ou a qualquer outro tipo de repositório de dados.

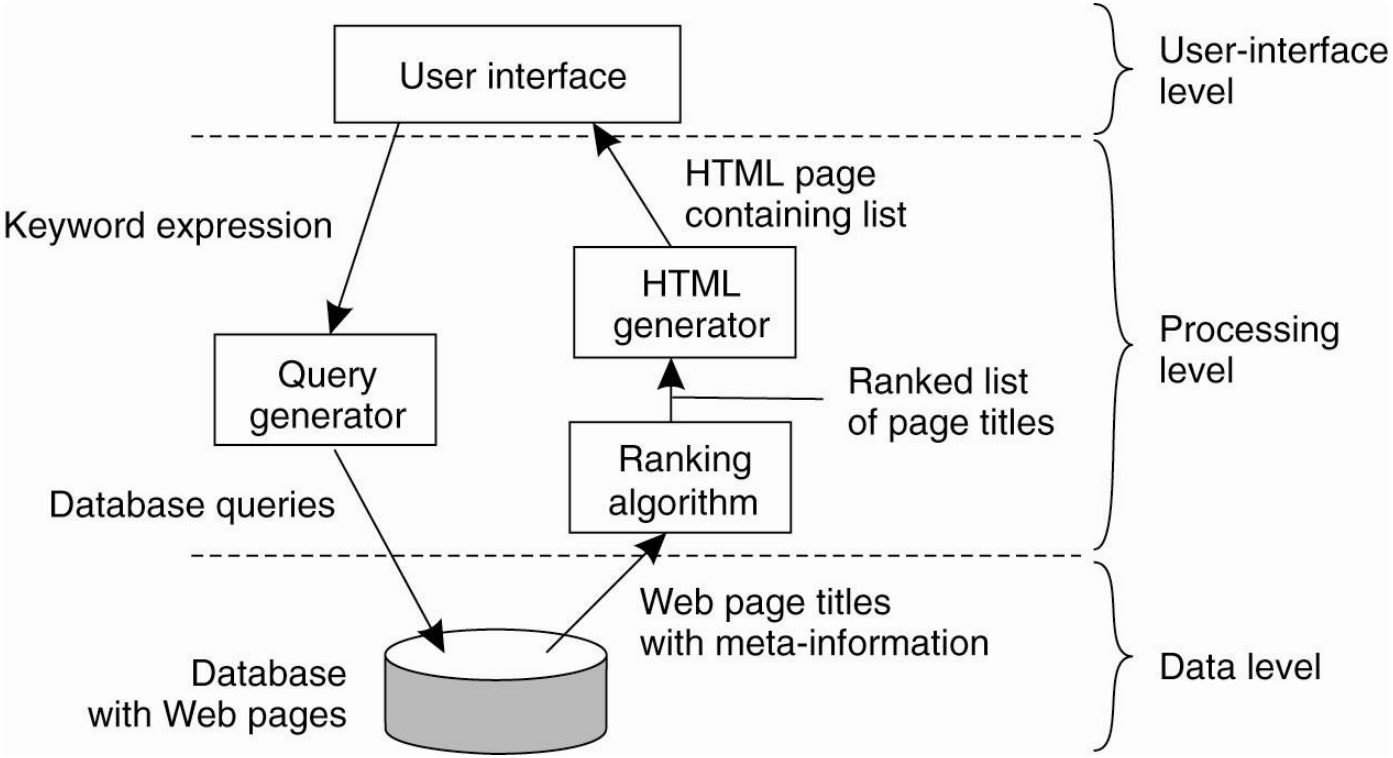
Considerando isso, tais sistemas podem implementar os seguintes níveis:

- Nível de interface de usuário;
- Nível de processamento;
- Nível de dados;
- Nível de segurança;
- Etc.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

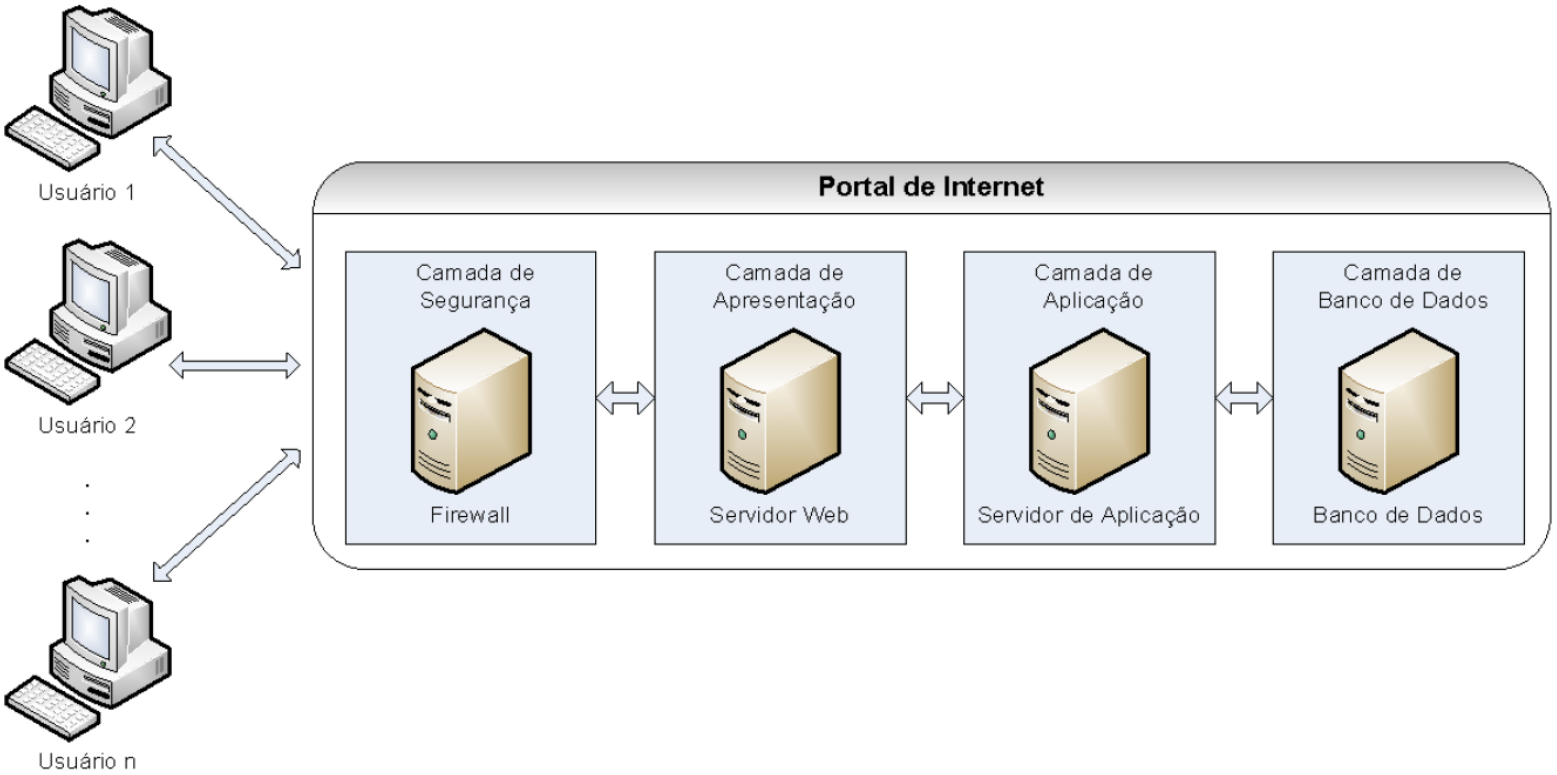
Exemplo de sistemas multicamadas:




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

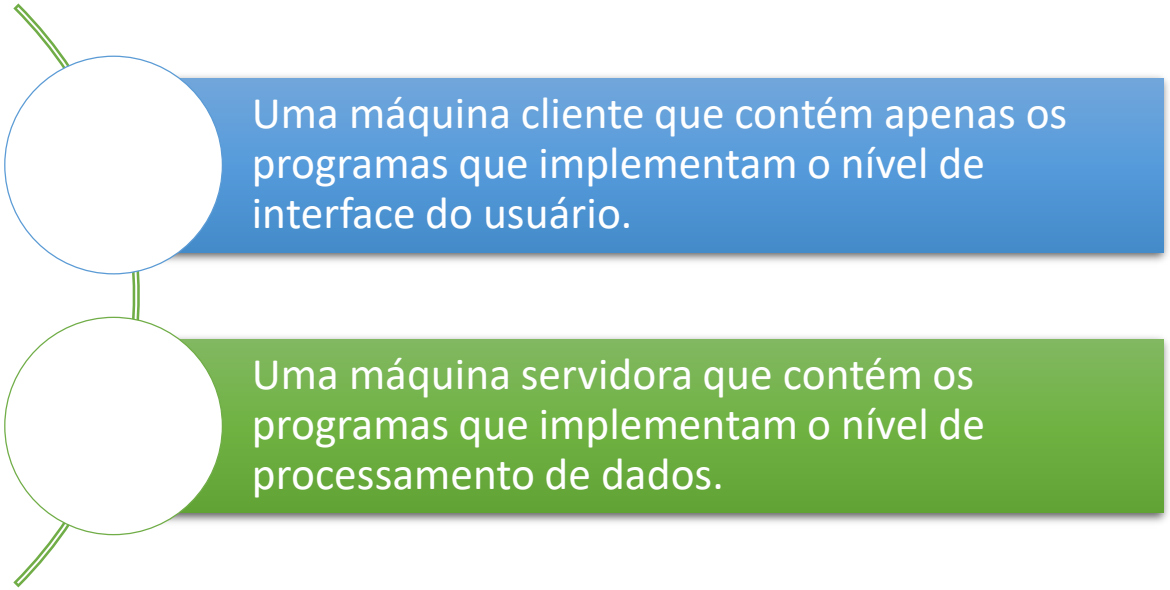
Exemplo de sistemas multicamadas:



# Arquiteturas de sistemas Centralizadas

A abordagem de sistemas multicamadas abre muitas possibilidades quanto a distribuição física de uma aplicação cliente/servidor em várias máquinas.

A organização mais simples é ter somente dois tipos de máquinas:

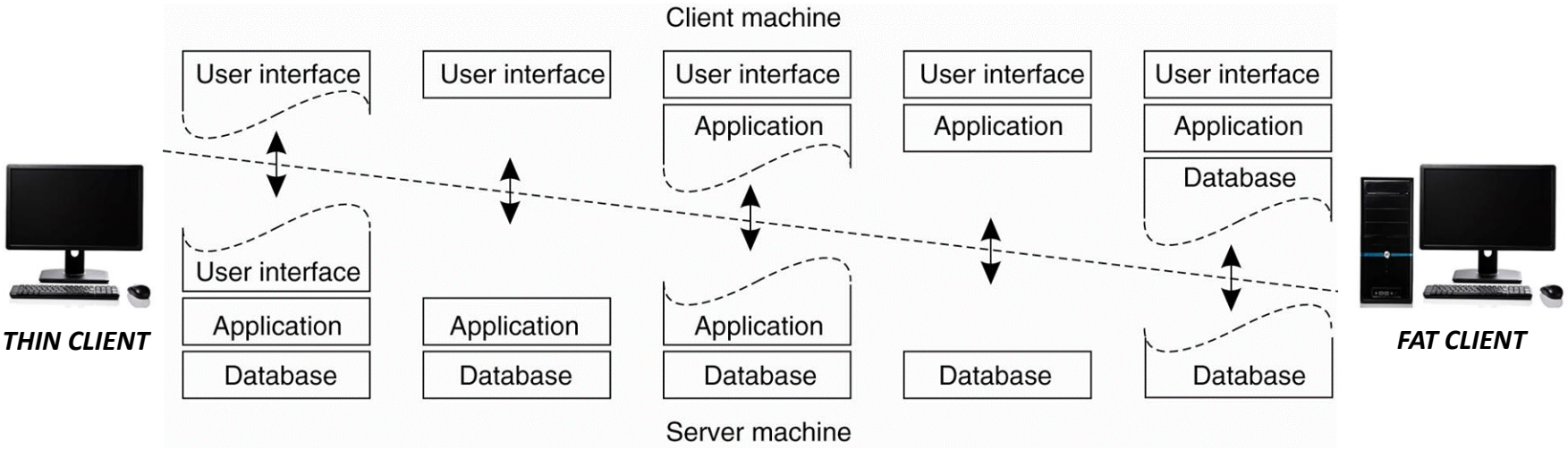



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

Uma outra forma de organização é distribuir os diferentes componentes da aplicação entre o cliente e o servidor.

Clientes que executam uma quantidade mínima de processos é denominada de Thin Client, enquanto que clientes que executam uma grande carga de processamento são chamados de Fat Client.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Centralizadas

Exemplo de validação do CPF no lado do cliente:

```
<script>
function ValidaCPF() {
   var strCPF = document.getElementById('CPF').value;
   var Soma;
   var Resto;
   Soma=0;
   for (i=1; i<=9; i++) Soma=Soma+parseInt(strCPF.substring(i-1,i))*(11-i);
   Resto=(Soma*10)%11;
   if ((Resto==10) || (Resto==11)) Resto=0;
   if (Resto!=parseInt(strCPF.substring(9,10))) {
      document.getElementById("ResultadoTesteCPF").innerHTML="CPF Inválido!";
      return; //return false
   };
   Soma=0;
   for (i=1; i<=10; i++) Soma=Soma+parseInt(strCPF.substring(i-1,i))*(12-i);
   Resto=(Soma*10)%11;
   if ((Resto==10) || (Resto==11)) Resto=0;
   if (Resto!=parseInt(strCPF.substring(10,11))) {
      document.getElementById("ResultadoTesteCPF").innerHTML="CPF Inválido!";
            return; //return false
   };
   document.getElementById("ResultadoTesteCPF").innerHTML="CPF Válido!"; //return true
}</script>
```

Fonte: www.geradorcpf.com

# Arquiteturas de sistemas Centralizadas

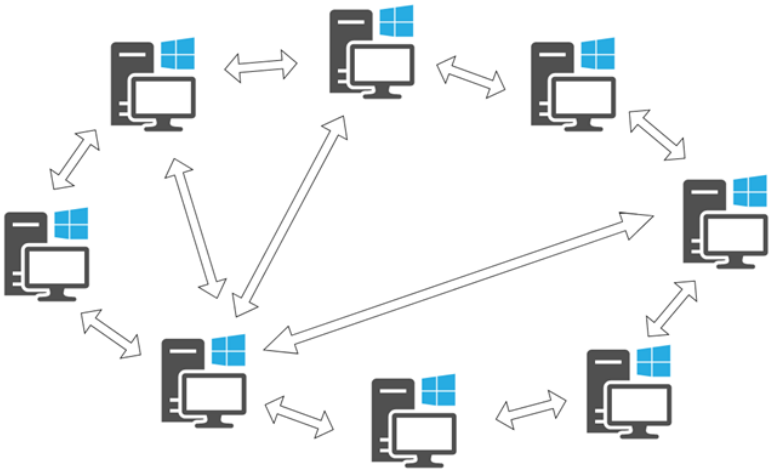
Exemplo de validação do CPF no lado do servidor:

```
<%
dim strCPF
strCPF=request.querystring("CPF")
dim Soma
dim Resto
dim teste
Soma=0
for i=1 to 9
   Soma=Soma+Cint(mid(strCPF,i,1))*(11-i)
next
Resto=(Soma*10) MOD 11
if ((Resto=10) OR (Resto=11)) then
   Resto=0
end if
if (Resto<>Cint(mid(strCPF,10,1))) then
   response.write("CPF Inválido!")
   response.end
end if
Soma=0
for i=1 to 10
   Soma=Soma+Cint(mid(strCPF,i,1))*(12-i)
next
Resto=(Soma*10) MOD 11
if ((Resto=10) OR (Resto=11)) then
   Resto=0
end if
if (Resto<>Cint(mid(strCPF,11,1))) then
   response.write("CPF Inválido!")
   response.end
end if
response.write("CPF Válido!")
%>
```

Fonte: www.geradorcpf.com

# Arquiteturas de sistemas Descentralizadas

Arquitetura de sistemas do tipo descentralizada é aquela em que os clientes e/ou servidores podem ser fisicamente subdivididos em partes logicamente equivalentes, mas onde cada parte opera na sua própria porção do conjunto completo de dados, de modo a equilibrar a carga.



Arquitetura de sistemas do tipo descentralizada também é conhecida como arquitetura de distribuição horizontal.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Descentralizadas

O exemplo mais comum de redes descentralizadas são as redes peer-to-peer (P2P) ou redes par-a-par.

Em sistemas peer-to-peer, os processos são todos iguais, o que implica na simetria da interação entre os processos. Assim, cada processo acaba agindo como cliente e servidor ao mesmo tempo.

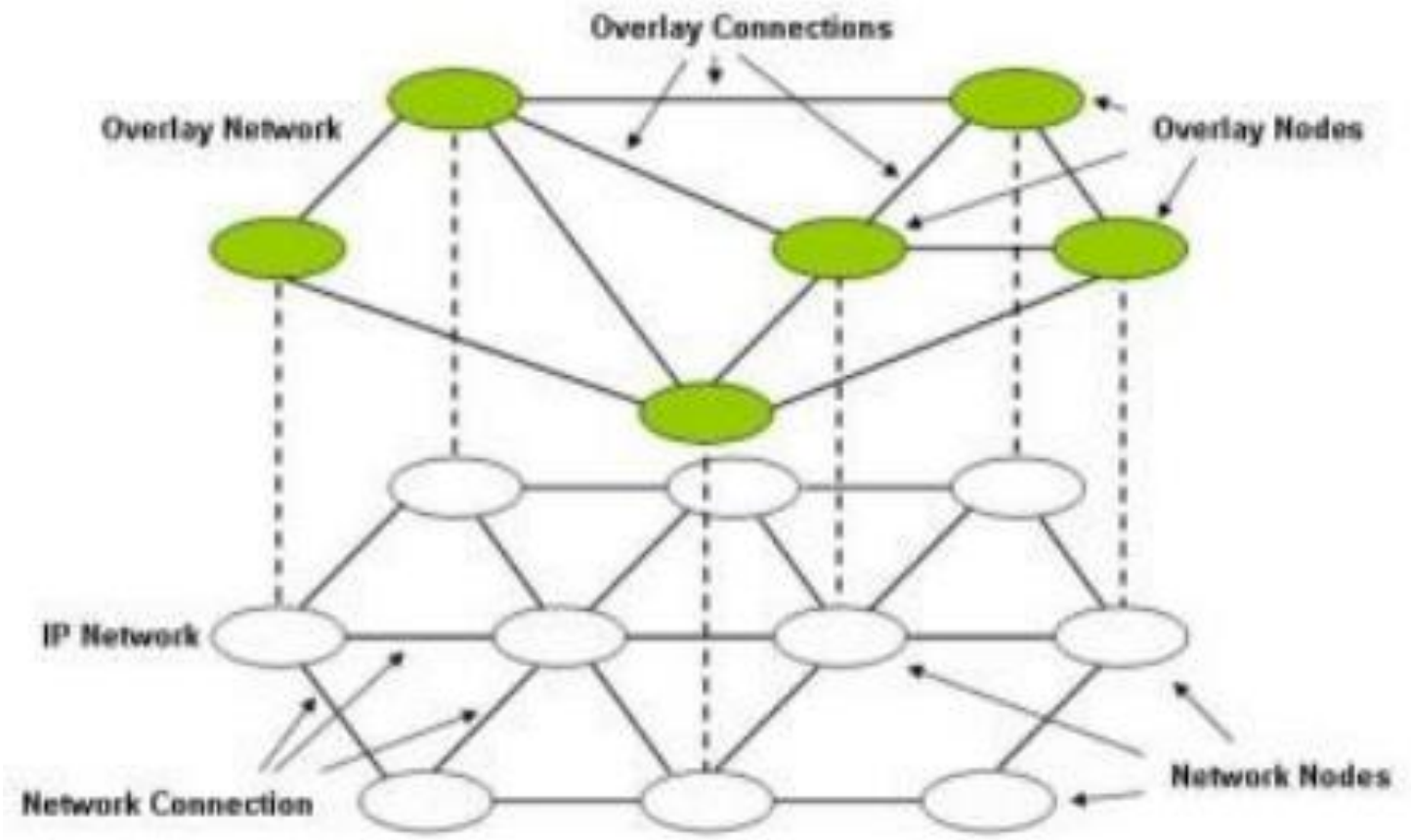
Sistemas peer-to-peer são por natureza uma rede de sobreposição, onde os nós são formados pelos processos e os enlaces de rede representam os canais de comunicação.

São dois os tipos de rede de sobreposição:

- Redes de sobreposição estruturadas;
- Redes de sobreposição não estruturadas.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Descentralizadas – Rede de sobreposição

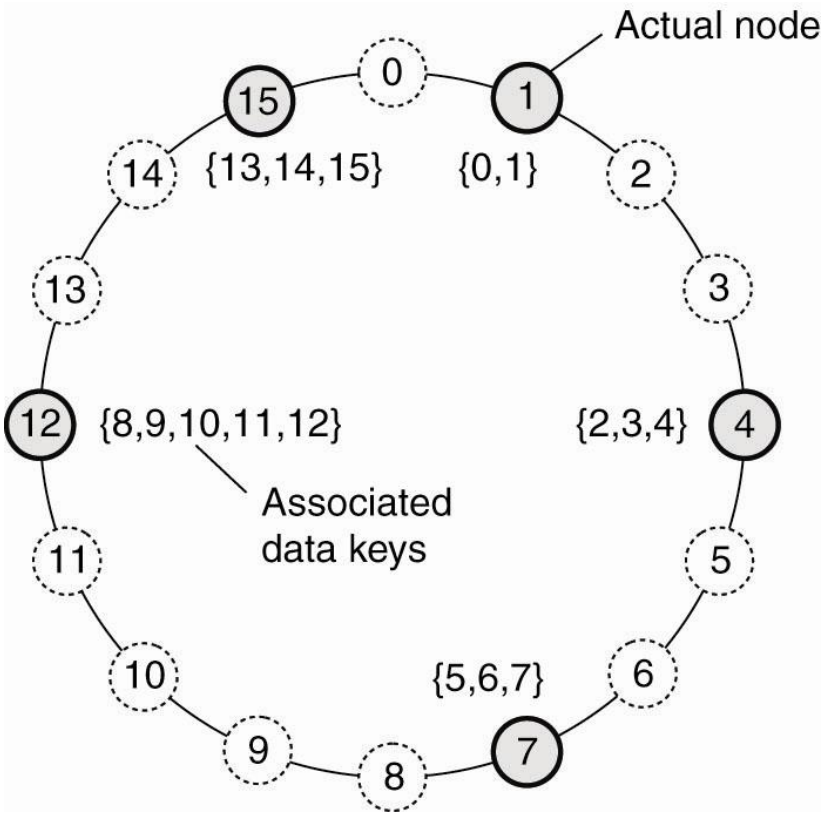


# Arquiteturas de sistemas Descentralizadas

**Redes de sobreposição estruturadas**

Nesta arquitetura, a rede de sobreposição é construída com a utilização de um procedimento determinístico.

O algoritmo mais usado é o Distributed Hash Table, ou DHT.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Descentralizadas

**Redes de sobreposição não estruturadas**

Nesta arquitetura, a rede de sobreposição depende de algoritmos aleatórios para ser construída.

Cada nó mantem uma lista de vizinhos construída de modo mais ou menos aleatório.

Como consequência, quando um nó precisa localizar um item de dado específico, é necessário inundar a rede com uma consulta de busca, o que diminui a sua eficiência.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Híbridas


Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de sistemas Híbridas

O BitTorrent é um sistema híbrido que permite o compartilhamento de arquivos pela internet, sendo utilizado principalmente para a distribuição de vídeos, músicas e softwares.

O BitTorrent é bastante popular e ao mesmo tempo muito eficiente. Isso se deve, basicamente, a dois motivos:

- quando um arquivo está sendo baixado para um computador, “pedacinhos” deste são obtidos de várias outras máquinas simultaneamente, não apenas de uma;
- ao mesmo tempo em que um computador obtém um arquivo, os dados que já foram baixados também são compartilhados, ou seja, para receber é preciso também fornecer.

O BitTorrent em si, na verdade, é um protocolo de compartilhamento de dados e não um sistema centralizado. Não há um servidor provendo os dados, mas sim um padrão de comunicação entre vários computadores que permite que arquivos sejam localizados, distribuídos e obtidos por todos.

Fonte: www.infowester.com

# Arquiteturas de sistemas Híbridas

Para que seja possível fazer download e upload pelo BitTorrent, é necessário que cada arquivo compartilhado esteja associado a um torrent. Trata-se de um arquivo pequeno e simples que contém as informações necessárias ao compartilhamento, como tamanho em bytes do conteúdo a ser compartilhado, dados que confirmam a integridade deste, endereços de trackers (servidor que orienta a comunicação do compartilhamento, conceito abordado mais abaixo), entre outros.

Alguns dos campos encontrados em um arquivo torrente são os seguintes:

- announce: informa qual o tracker que trata da distribuição do arquivo;
- announce-list: informa eventuais trackers auxiliares;
- comment: um comentário qualquer inserido pelo criador do torrent;
- created by: informa com qual software o torrent foi criado;
- info: contém todos os dados referentes ao arquivo, como nome, tamanho, código de verificação de integridade (hash), etc.

Fonte: www.infowester.com

# Arquiteturas de sistemas Híbridas

Para que se consiga encontrar um determinado arquivo para baixar, é necessário primeiro achar um torrent associado, que deve ser aberto em um cliente apropriado capaz de iniciar o download do material.

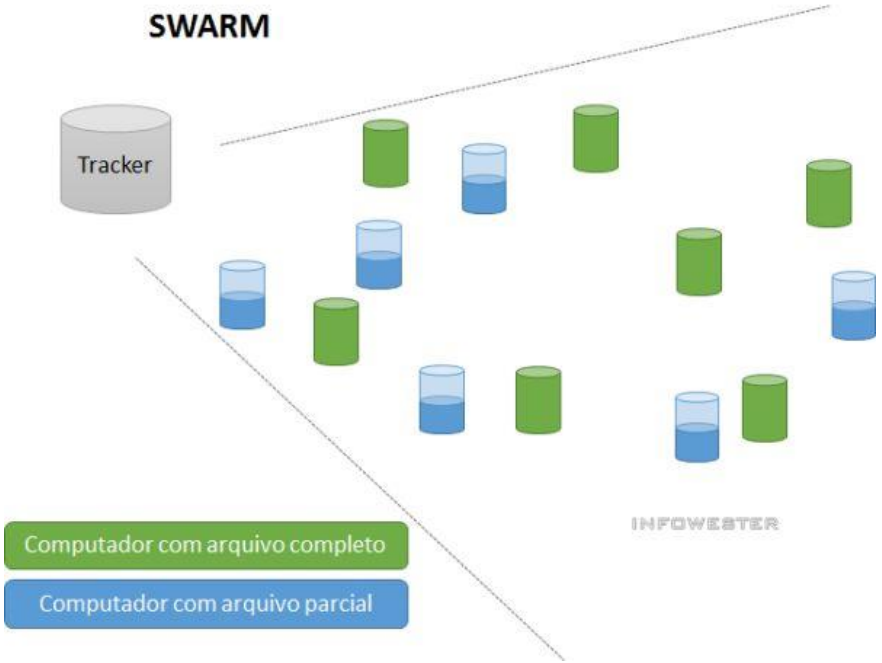
Abaixo estão descritas cinco denominações básicas que estão relacionadas com o compartilhamento de arquivos:

- Seed (semeador): é o nome dado a cada máquina que possui o arquivo completo que está sendo compartilhado. É necessário que haja pelo menos um seed para que o compartilhamento ocorra integralmente;
- Peer (nó): termo que indica cada computador que compartilha arquivos. Quando se está baixando algo pelo BitTorrent, o computador automaticamente assume o papel de um peer, ou seja, de um ponto ou nó na rede;
- Leecher (sugador): o termo faz referência aos computadores que ainda estão baixando arquivos ou que já o baixaram completamente, mas por alguma razão não o estão compartilhando;
- Tracker (rastreador): faz referência a um servidor que mantém o controle de comunicação entre todos os seeds e peers, de forma que os computadores envolvidos no processo possam saber a quais máquinas se conectar. No entanto, o tracker não tem cópia do arquivo, muito menos interfere diretamente no compartilhamento;

Fonte: www.infowester.com

# Arquiteturas de sistemas Híbridas

- Swarm (enxame): nome dado ao conjunto de computadores que está compartilhando o mesmo arquivo. Se determinado arquivo estiver sendo compartilhado por 8 seeds e 34 peers, o swarm do arquivo contém ao todo 42 computadores.




Fonte: www.infowester.com

# Arquiteturas vs Middleware

Interceptadores

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Autogerenciamento

Interceptadores

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Para saber mais...

… leia o Capítulo 2 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Módulo 15

Processos

# Introdução

Para executar um programa, um sistema operacional cria vários processadores virtuais, cada um para executar um programa diferente.

Para monitorar esses processadores virtuais, o sistema operacional tem uma tabela de processos que contem entradas para armazenar valores de registradores de CPU, mapas de memórias, arquivos abertos, informações de contabilidade, privilégios e assim por diante.

Um processo é definido como um programa em execução, isto é, um programa que está sendo executado em um dos processadores virtuais do sistema operacional no momento em questão.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução

O sistema operacional se assegura que processos independentes não possam afetar, de modo intencional, malicioso ou por acidente a correção do comportamento um do outro.

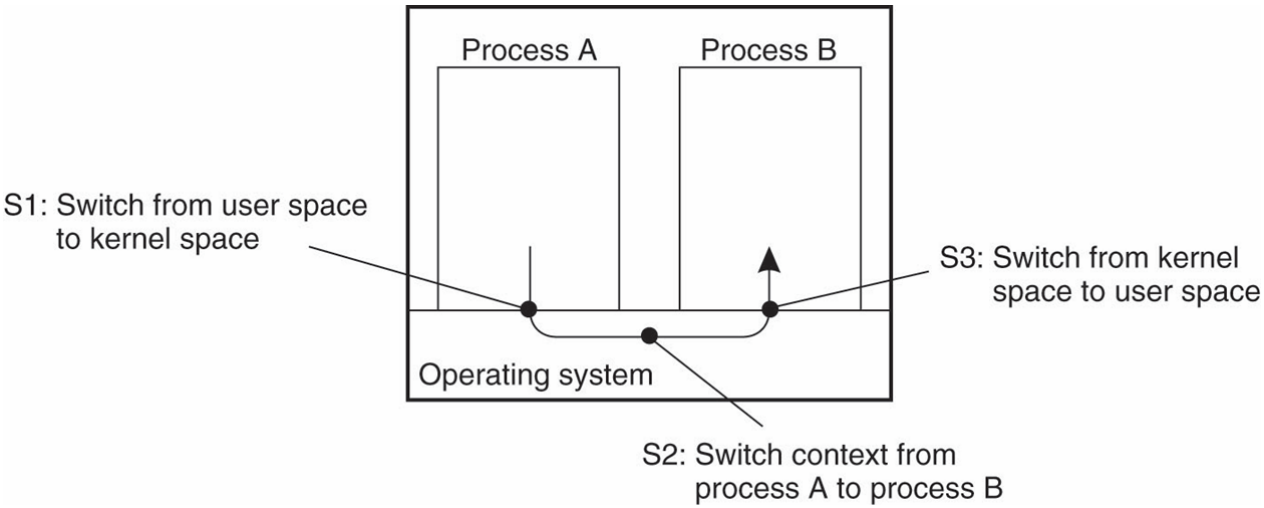
Cada vez que um processo é criado, o sistema operacional cria um espaço de memória independente.

No caso do Windows 32 bits, o espaço de endereços reservados a cada processo é de 4GB. Para Windows 64 bits, o espaço de endereços reservador é de 16TB!

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução

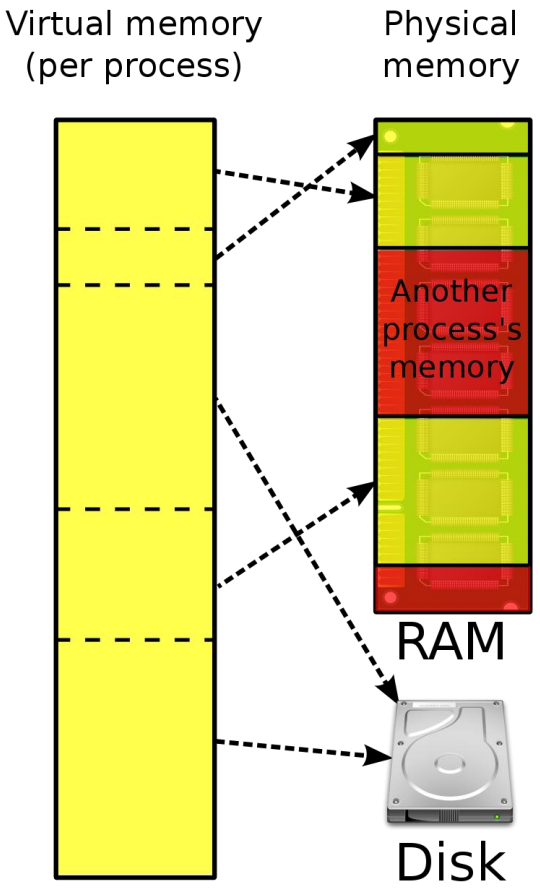
Para o sistema operacional chavear a CPU entre um processo e outro é necessário salvar o contexto da CPU (registradores, contador de programas, ponteiro de pilha, etc.).




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução

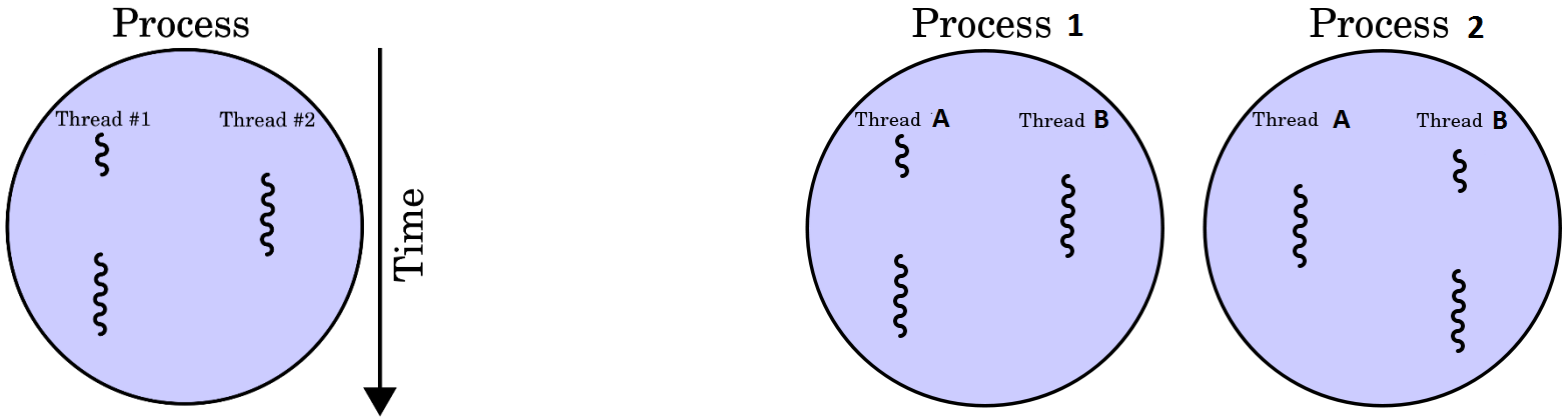
Se o sistema operacional suportar mais processos do que pode conter simultaneamente a memória RAM, pode ser necessária a troca dinâmica de processos entre a RAM e o disco (memória virtual).




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Thread

Um thread é uma parte de um processo. Processos são por natureza monothread, mas podem ser programados usando-se técnicas de multithreading, o que acarreta em melhora do desempenho. Os threads de um mesmo processo compartilham o mesmo espaço de memória. O chaveamento entre threads também é mais simples, pois em geral apenas o contexto de registradores precisa ser salvo.

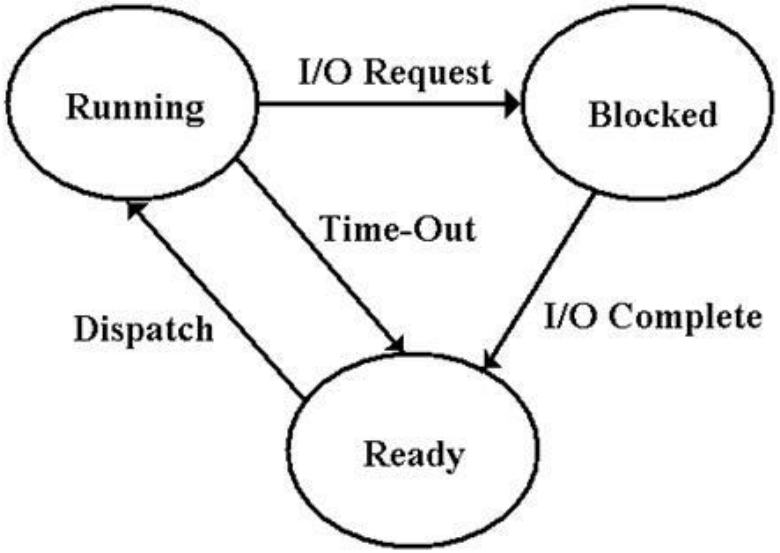



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Thread

No processo, quando for executada uma chamada bloqueadora de sistema, o processo é bloqueado como um todo.

Em processos multithreading, quando um thread executa uma chamada bloqueadora de sistema, em geral apenas a thread é bloqueada, e as demais continuam sua execução. O uso de threads favorece o paralelismo.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Threads em sistemas distribuídos

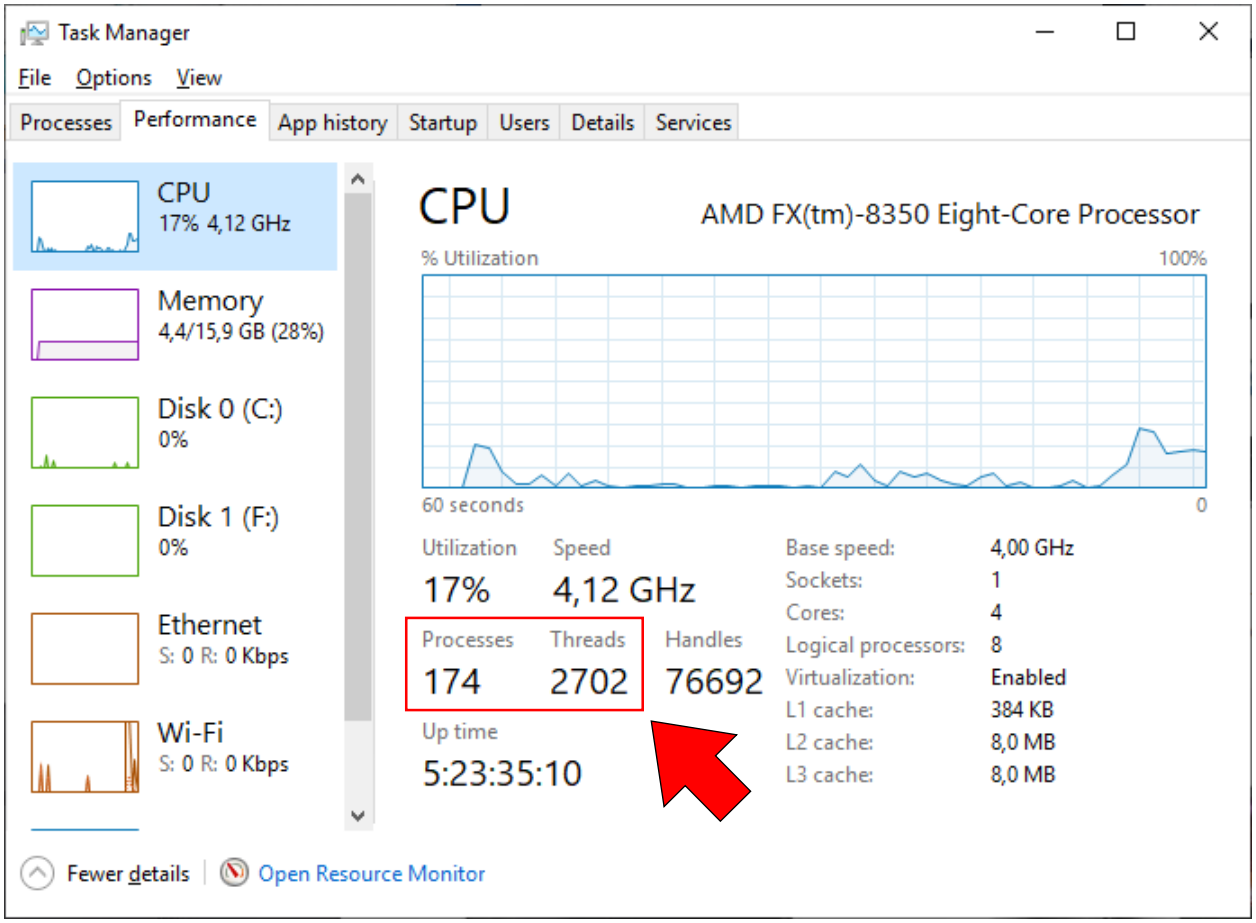
Em sistemas distribuídos, o uso de threads pode ser notado tanto em clientes quanto em servidores.

Do lado do cliente, um navegador de internet ao acessar uma página web pode executar vários threads, uma para cada elemento da página, por exemplo.

Do lado do servidor, o sistema de arquivos poderia ficar bloqueado caso um usuário solicitasse a leitura de um arquivo (chamada bloqueadora), de modo que demais usuários que quisessem realizar operações no disco teriam que esperar. Ao usar threads, cada operação no disco poderia ser executada de forma independente.

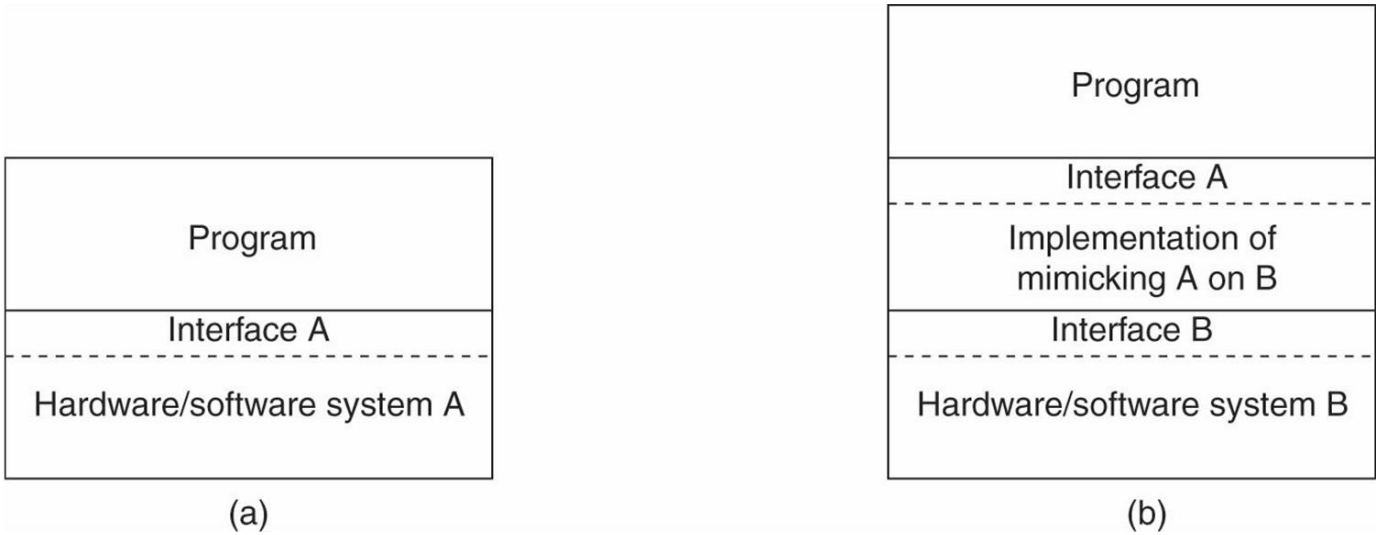
Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Exemplo


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
CPU
17% 4,12 GHz
Memory
4,4/15,9 GB (28%)
Disk 0 (C:)
0%
Disk 1 (F:)
0%
Ethernet
S: 0 R: 0 Kbps
Wi-Fi
S: 0 R: 0 Kbps
CPU
AMD FX(tm)-8350 Eight-Core Processor
% Utilization
100%
60 seconds
0
Utilization
17%
Speed
4,12 GHz
Processes
174
Threads
2702
Handles
76692
Up time
5:23:35:10
Base speed: 4,00 GHz
Sockets: 1
Cores: 4
Logical processors: 8
Virtualization: Enabled
L1 cache: 384 KB
L2 cache: 8,0 MB
L3 cache: 8,0 MB
Fewer details
Open Resource Monitor

# Virtualização em sistemas distribuídos

Do ponto de vista de sistemas distribuídos, a virtualização trata de estender ou substituir uma interface existente de modo a imitar o comportamento de um outro sistema.

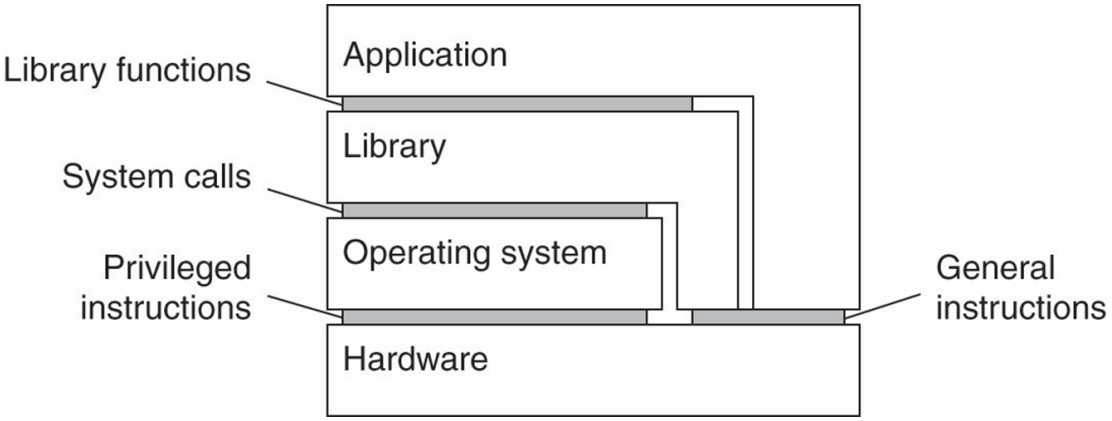



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de máquinas virtuais

Sistemas de computadores oferecem quatro tipos diferentes de interfaces em quatro níveis diferentes:

- Instruções de máquina invocadas por qualquer programa;
- Instruções de máquina privilegiadas;
- Chamadas de sistema via sistema operacional;
- Interface de aplicação de programas (API).

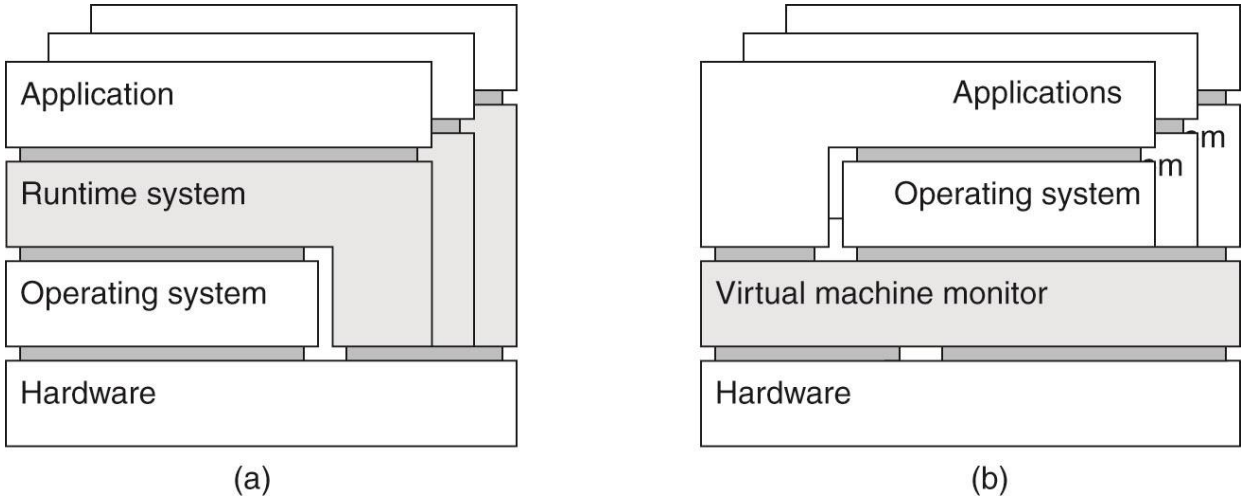



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Arquiteturas de máquinas virtuais

A virtualização pode ocorrer de dois modos:

- Sistema de execução que forneça um conjunto de instruções abstratas que deve ser utilizado para executar aplicações, como por exemplo Java e Wine. Também chamado de máquina virtual de processo;
- Sistema que isola o hardware oferecendo como interface um conjunto completo de instruções. Também chamado de monitor de máquina virtual.



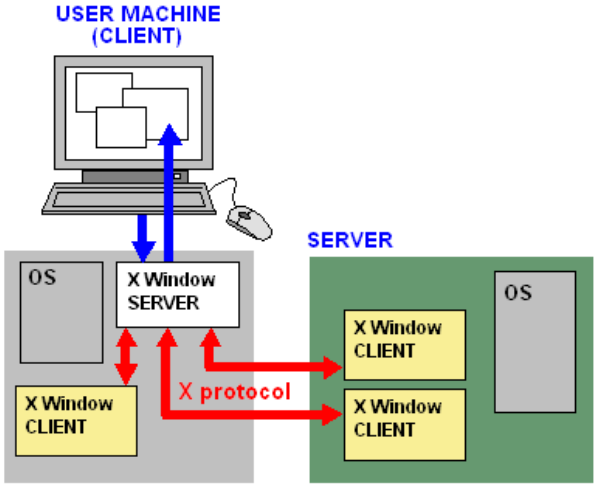

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Clientes

Uma tarefa importante de máquinas clientes é proporcionar aos usuários meios de interagir com servidores remotos.

Um primeiro modo seria cada serviço remoto ter uma contraparte separada que pode contatar o serviço pela rede.

Um segundo modo é fornecer acesso direto a serviços remotos usando apenas uma interface de usuário, como Telnet ou X-Window.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Servidores

Um servidor é um processo que implementa um serviço específico em nome de um conjunto de clientes, ou seja, ele espera por uma requisição que vem de um cliente e em seguida assegura que ela será atendida, após o que espera pela próxima requisição.

São as seguintes as formas de se organizar servidores:

- Servidor iterativo: é o próprio servidor que manipula a requisição e retorna uma resposta ao cliente;
- Servidor concorrente: não manipula por si próprio a requisição, mas a passa para um thread separado ou para outro processo, após o que imediatamente espera pela próxima requisição.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Para saber mais...

… leia o Capítulo 3 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Módulo 16

Comunicação

# Introdução

A comunicação entre processos é um dos principais aspectos de um sistema distribuído, pois os processos que são executados em máquinas diferentes necessitam trocar informações.

Uma forma de processos em máquinas diferentes se comunicarem é por meio da troca de mensagens de baixo nível oferecida pela rede subjacente.

Outras formas de comunicação são:

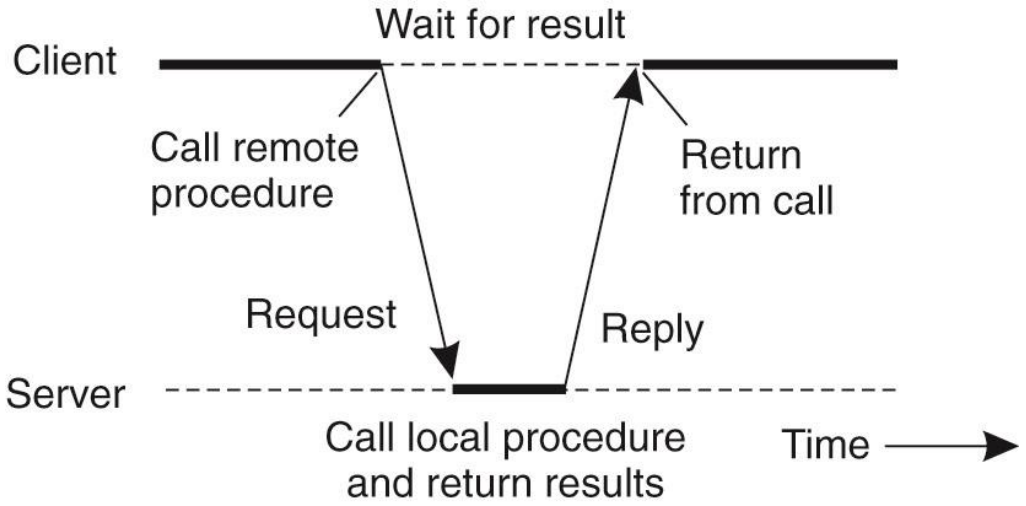
- Chamada de procedimento remoto;
- Middleware orientado a mensagem;
- Fluxo de dados.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto

A Chamada de Procedimento Remoto ou RPC (Remote Procedure Call) ocorre quando um processo na máquina A (cliente) chama um procedimento na máquina B (servidor).

O processo chamador em A é suspenso e a execução do procedimento chamado ocorre em B. Informações podem ser transportadas do chamador para quem foi chamado nos parâmetros e podem voltar no resultado do procedimento.

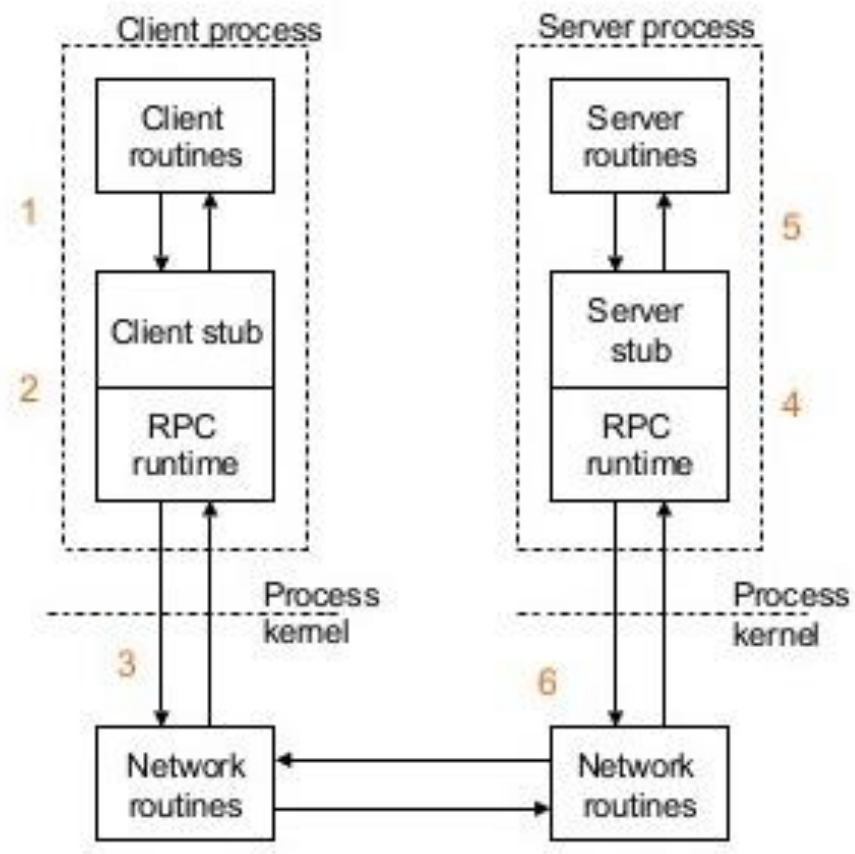



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto

Uma chamada de procedimento remoto ocorre nas seguintes etapas:

1. O procedimento de cliente chama o apêndice de cliente (client stub) do modo normal;
2. O apêndice de cliente (client stub) constrói uma mensagem e chama o sistema operacional local;
3. O sistema operacional do cliente envia a mensagem para o sistema operacional remoto;
4. O apêndice do servidor (server stub) desempacota os parâmetros e chama o servidor;
5. O servidor faz o serviço e retorna o resultado para o apêndice do servidor (server stub), que em seguida empacota o resultado em uma mensagem e chama o sistema operacional local;
6. O sistema operacional do servidor envia a mensagem para o sistema operacional do cliente.

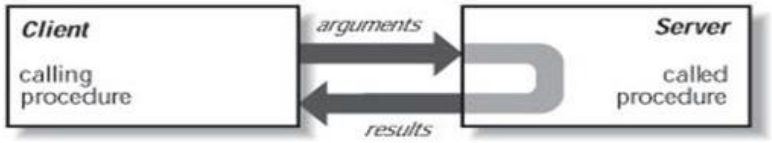



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento local

A Chamada de Procedimento Local ou LPC (Local Procedure Call) é um recurso de comunicação entre processos para transmissão de mensagens em alta velocidade disponível na plataforma Microsoft Windows.

Quando uma aplicação precisa chamar uma função de API em um subsistema como o Win32, por exemplo, o cliente usa um espaço reservado chamado de apêndice (stub), que está localizado em uma DLL.

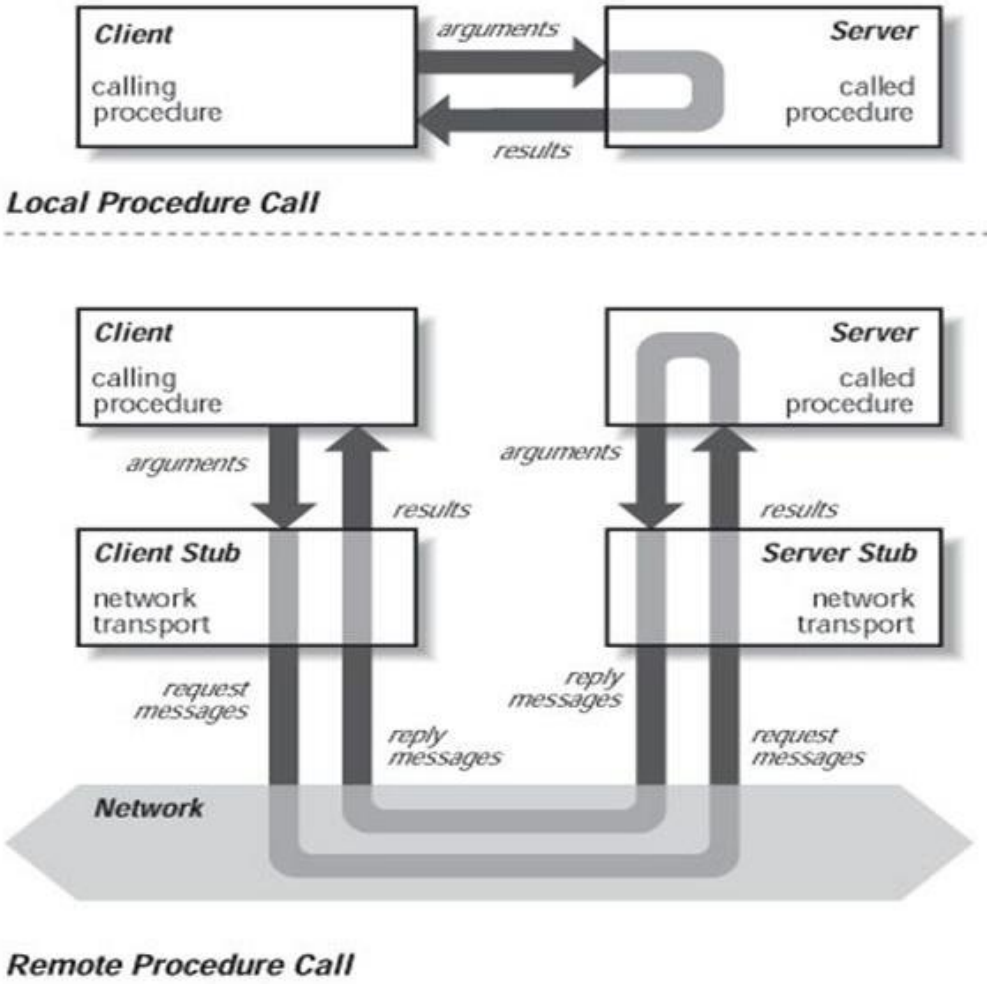



O stub é usado para empacotar e enviar os parâmetros que estão sendo transmitidos para o processo do subsistema do servidor que implementa a chamada, que os descompacta e executa a função chamada. O LPC espera então que uma resposta seja emitida de volta.

Do ponto de vista da aplicação, todo o processo parece ocorrer localmente dentro da DLL. O aplicativo não sabe que a DLL encaminhou a chamada para outra API usando o recurso de LPC.

Fonte: www.thenetworkencyclopedia.com

# LPC vs. RPC


Client
calling procedure
arguments
Server
called procedure
results
Local Procedure Call
Client
calling procedure
Server
called procedure
arguments
arguments
results
results
Client Stub
network transport
Server Stub
network transport
request messages
reply messages
reply messages
request messages
Network
Remote Procedure Call

# Chamada de procedimento remoto
# Passagem de parâmetros

A função do apêndice do cliente (client stub) é pegar os parâmetros passados pela aplicação, empacotá-los em uma mensagem e enviá-los ao apêndice do servidor (server stub).

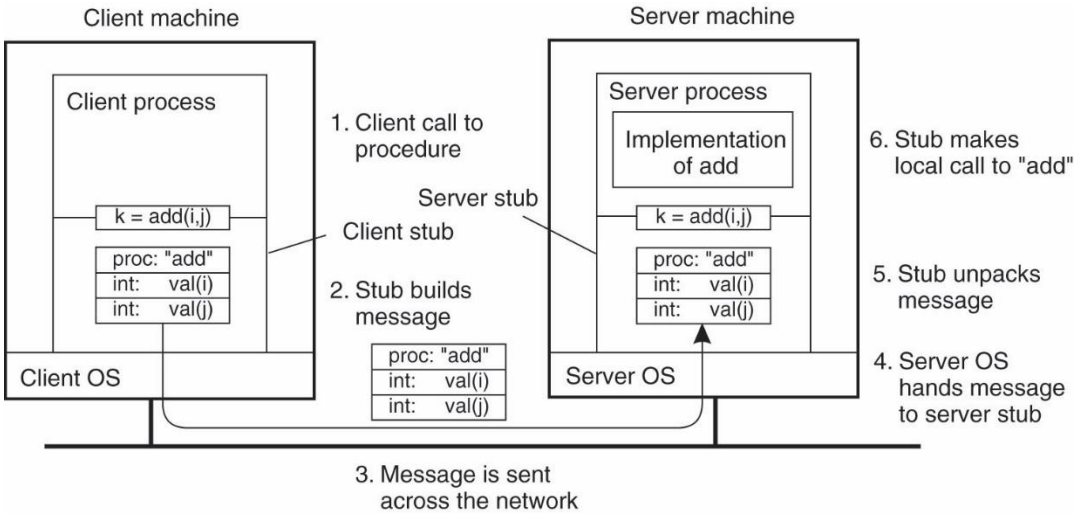
A passagem de parâmetros pode ser realizada de uma das seguintes formas:

- Passagem de parâmetros de valor;
- Passagem de parâmetros por referência.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto
# Passagem de parâmetros

Na **passagem de parâmetros de valor**, quando um processo chama um procedimento (função ou subrotina) e precisa passar parâmetros, uma cópia dos mesmos é enviada junto com a passagem de valor.




Na passagem por valor, é passada uma cópia do valor da variável.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto
# Passagem de parâmetros

Na **passagem de parâmetros por referência**, quando um processo chama um procedimento (função ou subrotina) e precisa passar parâmetros, o endereço da área de memória que contem os valor dos parâmetros é enviada junto com a passagem por referência.

No entanto, quando os processos chamador e chamado encontram-se em máquinas separadas, esta abordagem não é possível de ser implementada pois as máquinas não compartilham o mesmo espaço de endereços de memória.

Assim, em chamadas de procedimento remoto, a passagem de parâmetros por referência deve ser evitada.

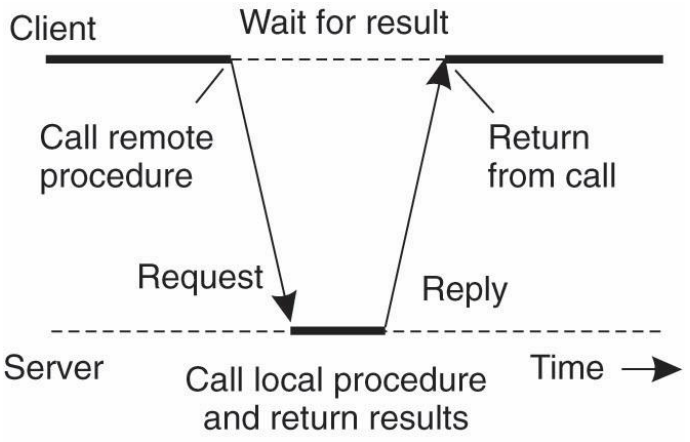
Na passagem por referência, é passada uma referência à própria variável.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto Comunicação assíncrona

Nas chamadas de procedimento convencionais, quando um cliente chama um procedimento remoto, o mesmo fica bloqueado até que uma resposta seja retornada.

Esse comportamento é desnecessário quando não há nenhum resultado a retornar, pois só leva ao bloqueio do cliente enquanto ele poderia continuar a realizar outras tarefas.

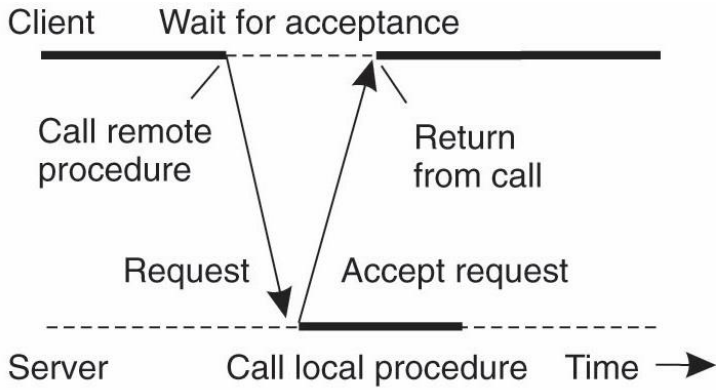



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Chamada de procedimento remoto Comunicação assíncrona

Já nas chamadas de procedimento com suporte a comunicação assíncrona, o servidor envia imediatamente uma resposta de volta ao cliente no momento em que a requisição é recebida e em seguida chama o procedimento requisitado.

Neste caso, a resposta do servidor age como um reconhecimento para o cliente de que o servidor irá processar a chamada. O cliente continuará assim sem mais bloqueio, tão logo tenha recebido o reconhecimento do servidor.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Middleware orientado a mensagem

O Middleware Orientado a Mensagem ou Message-Oriented Middleware (MOM) é usado quando não há como garantir que o receptor de uma chamada de procedimento remoto estará em execução no momento em que a requisição é emitida.

O MOM pode usar duas abordagens:

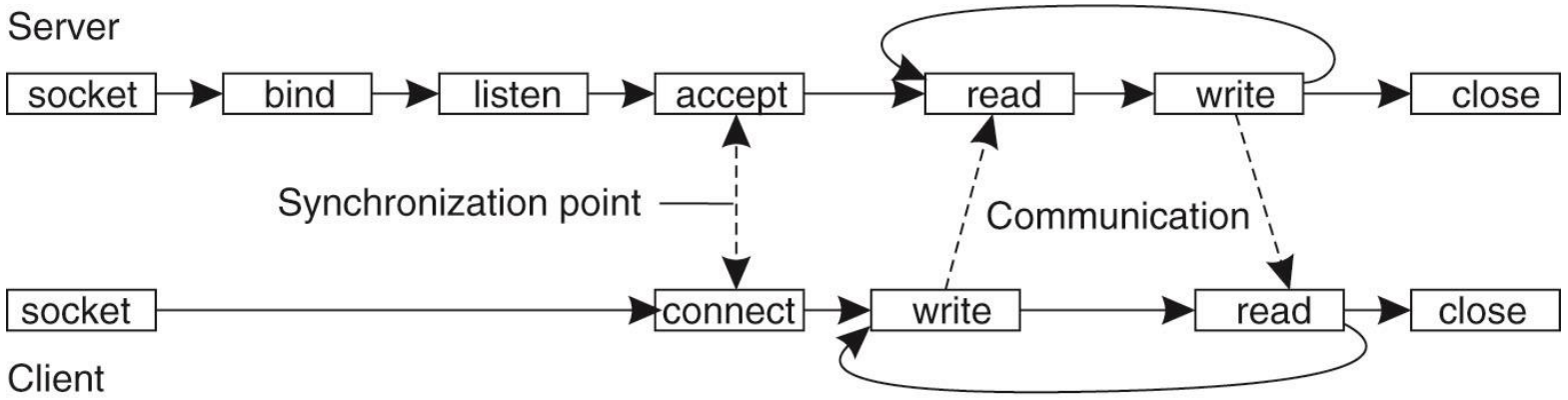
- Comunicação transiente orientada a mensagem;
- Comunicação persistente orientada a mensagem.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Middleware orientado a mensagem
# Comunicação transiente orientada a mensagem

Na comunicação transiente orientada a mensagem, tanto o cliente quanto o servidor precisam estar em execução no momento da troca de mensagens em uma chamada de procedimento remota.

Para isso pode ser usado a camada de comunicação de rede por meio de sockets TCP/IP.

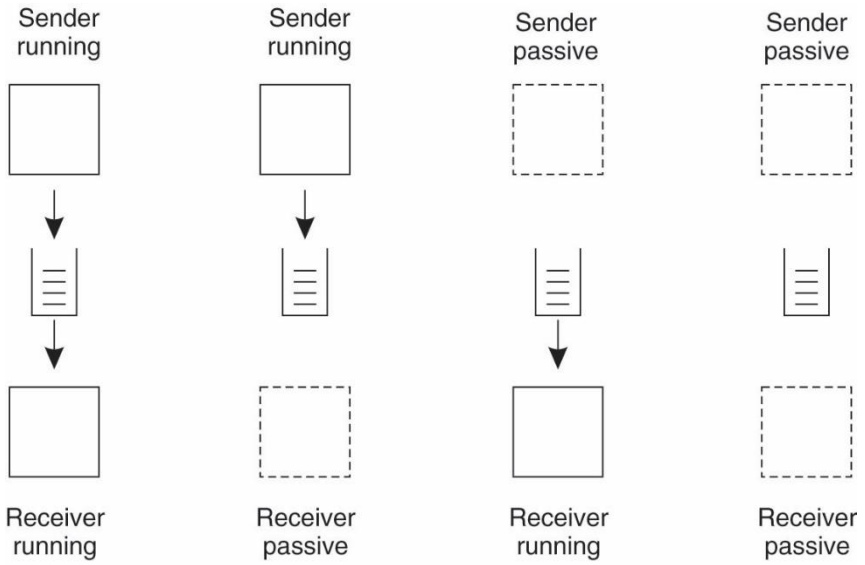



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Middleware orientado a mensagem
# Comunicação persistente orientada a mensagem

Na comunicação persistente orientada a mensagem, uma mensagem apresentada para transmissão é armazenada pelo sistema de comunicação pelo tempo que for necessário para entrega-lo.

Para isso é usado um mecanismo de enfileiramento de mensagens.

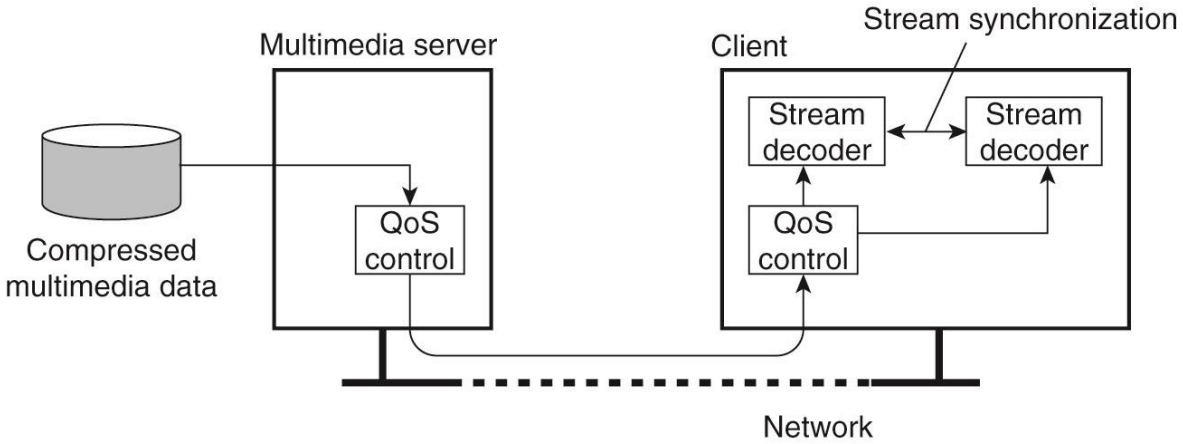



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Comunicação orientada a fluxo

Na comunicação orientada a fluxo, a questão central é se duas ou mais mensagens consecutivas tem ou não uma relação temporal.

Quando estas mensagens tem uma relação temporal, o fluxo de dados passa a ser contínuo, como nas transmissões de áudio e vídeo, por exemplo.



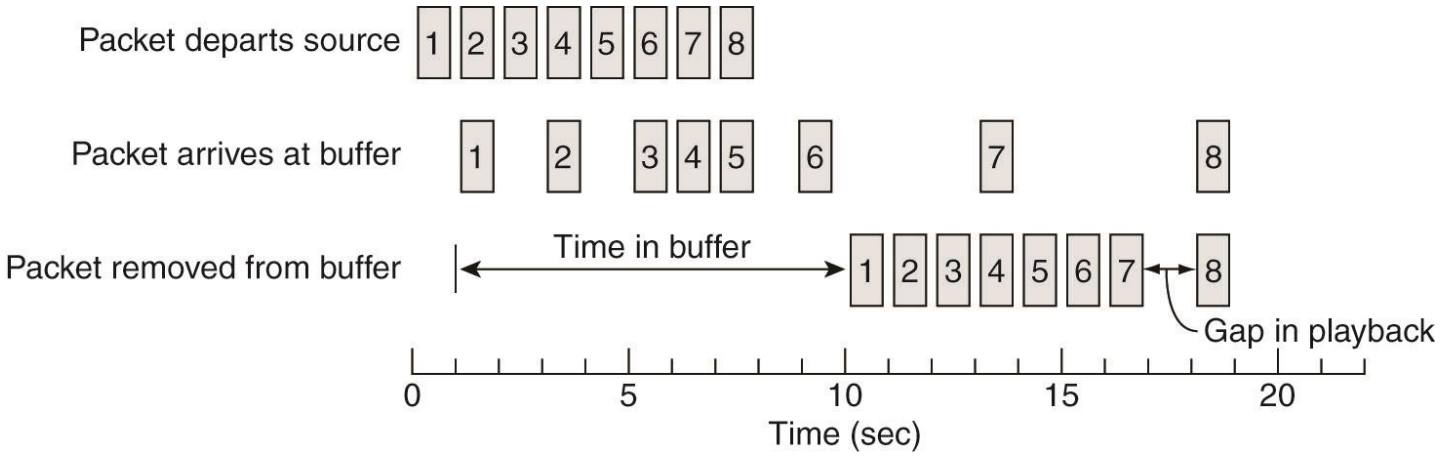

Nestes casos um atraso fim-a-fim máximo deve ser especificado para cada mensagem, a fim de oferecer tanto quanto possível uma melhor experiência para o usuário.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Comunicação orientada a fluxo

No entanto, a variância no atraso na entrega nas mensagens pode dificultar a constância do fluxo da transmissão.

Neste caso pode-se usar um buffer de armazenamento.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Para saber mais...

… leia o Capítulo 4 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Módulo 17

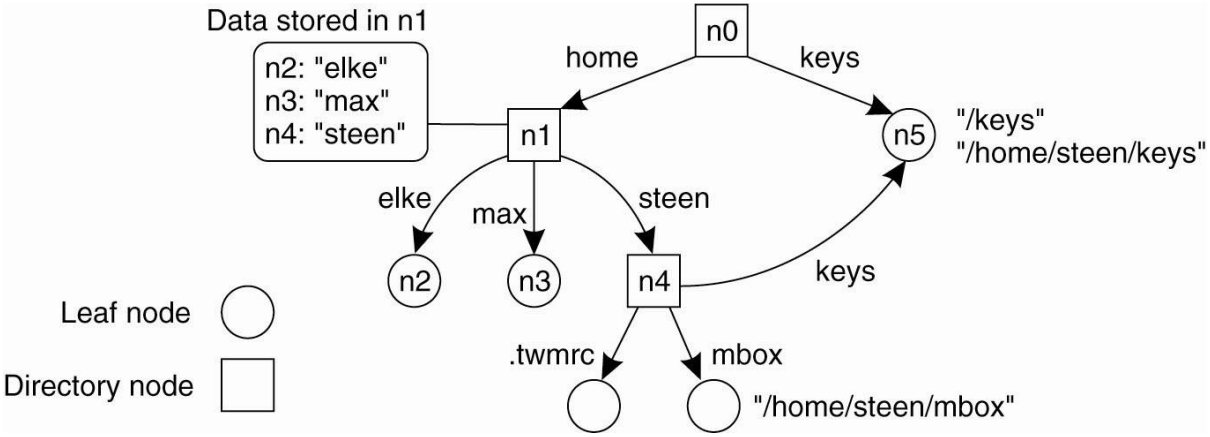
Nomeação

# Introdução

Nomes desempenham um papel muito importante em todos os sistemas de computadores, pois são usados para compartilhar recursos, identificar entidades de maneira única, fazer referência a localizações e outras funções.

Um nome pode ser resolvido para a entidade à qual se refere, ou seja, a resolução de nomes permite que um processo acesse a entidade nomeada.

A implementação de um sistema de nomeação costuma ser distribuída por várias máquinas, o que faz com que este sistema seja mais eficiente e escalável.




Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução
# Nomes, identificadores e endereços

Um **nome** em um sistema distribuído é uma cadeia de bits ou caracteres usada para referenciar uma entidade.

Uma **entidade** em um sistema distribuído pode ser praticamente qualquer coisa, como por exemplo impressoras, discos e arquivos, bem como processos, usuários, caixas postais, páginas Web, janelas gráficas, mensagens, conexões de rede, etc.

Entidades são ativas, pois devem oferecer uma interface que permita interagir com elas. E para agir sobre uma entidade, é necessário um ponto de acesso.

Um ponto de acesso é um tipo de especial de entidade. O nome de um ponto de acesso é denominado **endereço**.

Uma entidade pode oferecer mais de um ponto de acesso.

O endereço de um ponto de acesso de uma entidade também é denominado simplesmente endereço daquela entidade.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução
# Nomes, identificadores e endereços

Um **identificador** é um outro tipo de nome usado para identificar exclusivamente uma entidade.

Um identificador possui três propriedades:

- Um identificador referencia somente uma entidade;
- Cada entidade é referenciada por exatamente um identificador;
- Um identificador nunca é atribuído a uma outra entidade.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# DNS

O Domain Name System é um banco de dados hierárquico que oferece o serviço de resolução de nomes URL (Uniform Resource Locator) usados para identificar um domínio.

Toda comunicação na Internet é feita por meio dos endereços IP, mas é muito mais fácil memorizar URL´s do que endereços IP.

Assim, o que o serviço de DNS faz é converter as URL´s em endereços IP:

**`www.brasil.gov.br → 170.246.252.243`**

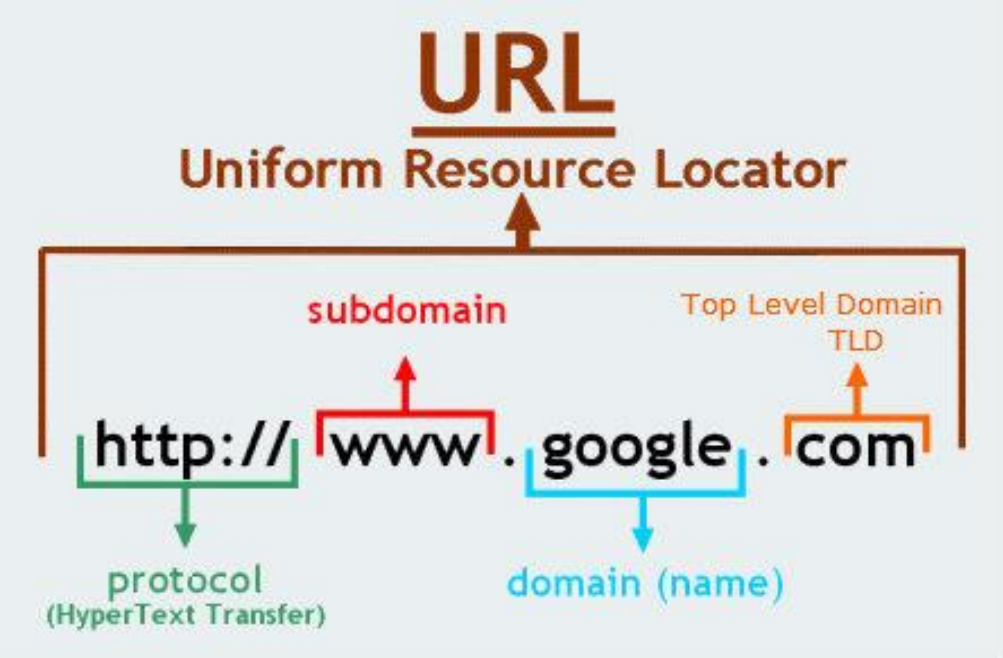
**`www.tj.sp.gov.br → 200.142.86.230`**

**`www.google.com → 172.217.162.100`**

# URL – Uniform Resource Locator

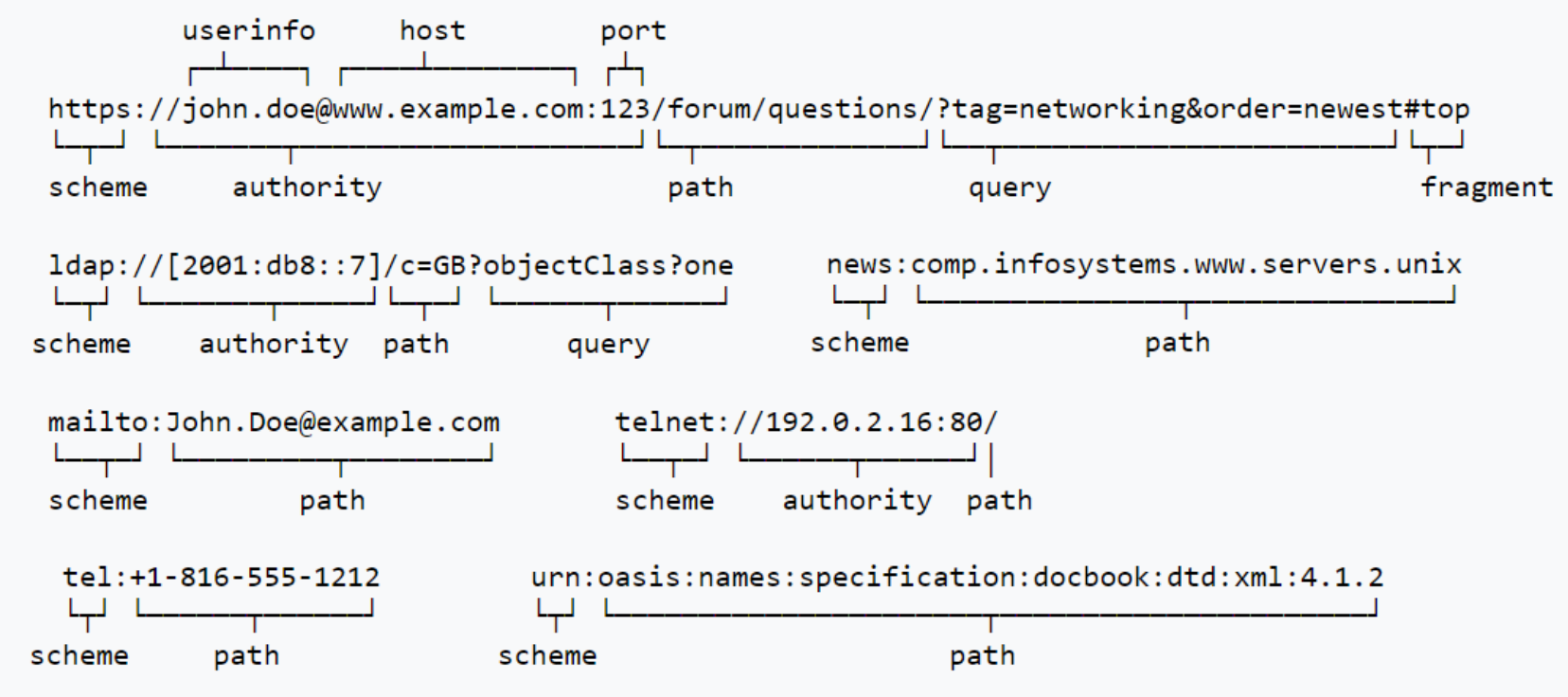
Um URL (Uniform Resource Locator), é uma referência a um recurso da web que especifica sua localização em uma rede de computadores e um mecanismo para recuperá-lo.

Um URL é um tipo específico de URI (Uniform Resource Identifier).



# URI – Uniform Resource Identifier

Um URI (Uniform Resource Identifier) é uma cadeia de caracteres que identifica inequivocamente um recurso específico.



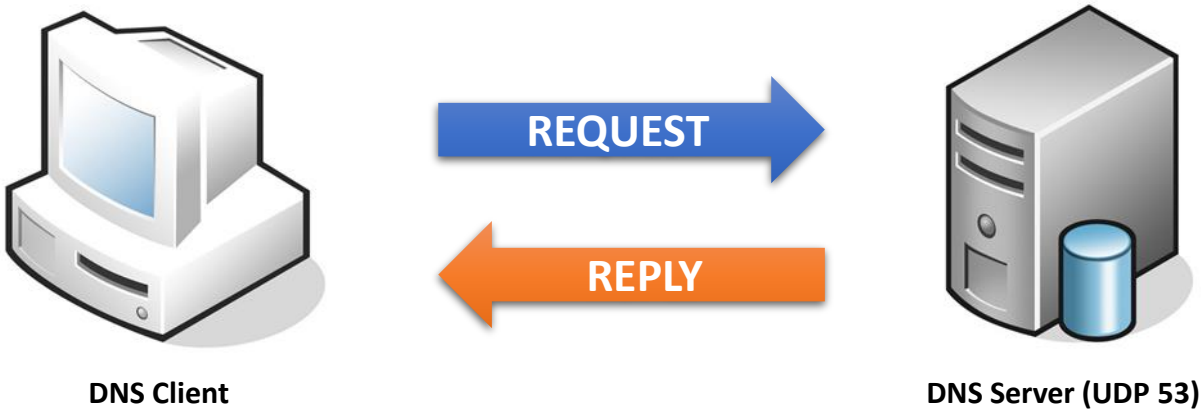

Fonte: wikipedia.org

# DNS

Um cliente DNS é todo aquele que requisita respostas a uma determina consulta feita a um servidor DNS.

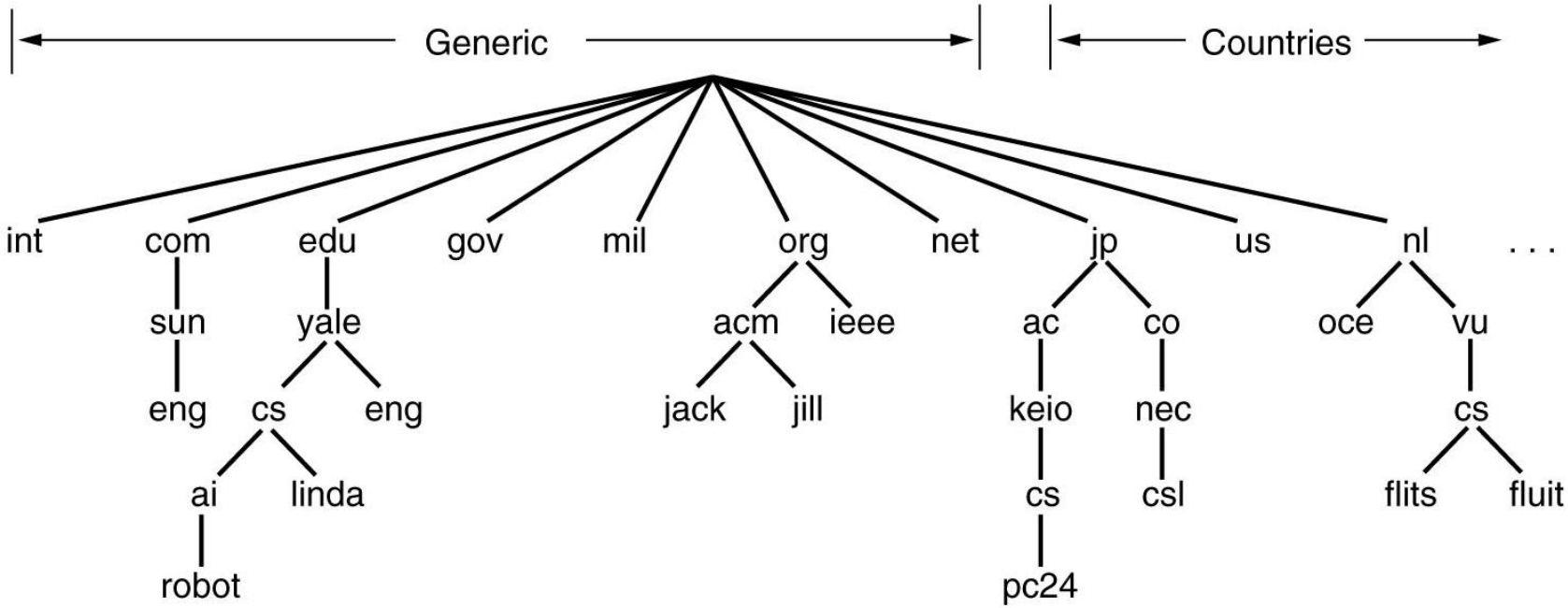
Um servidor DNS é todo aquele que responde às consultas feitas por um cliente.

O servidor DNS opera na porta UDP 53.



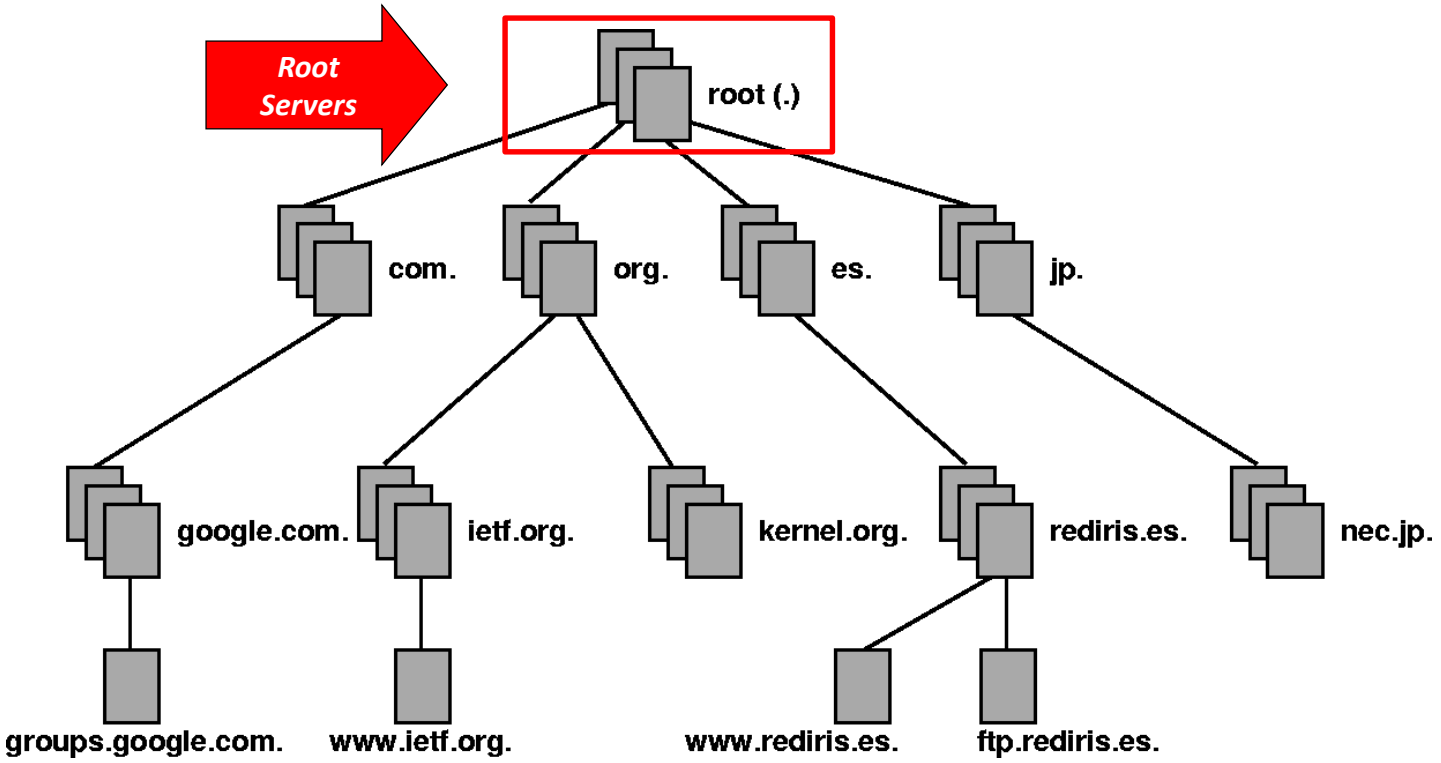
# DNS

Os nomes de domínio servem para identificar uma rede ou grupo de computadores. Estão dispostos de forma hierárquica e geralmente possuem um ou mais servidores DNS responsáveis por mapear todos os nomes abaixo daquele domínio (ou subdomínio) em endereços IP.



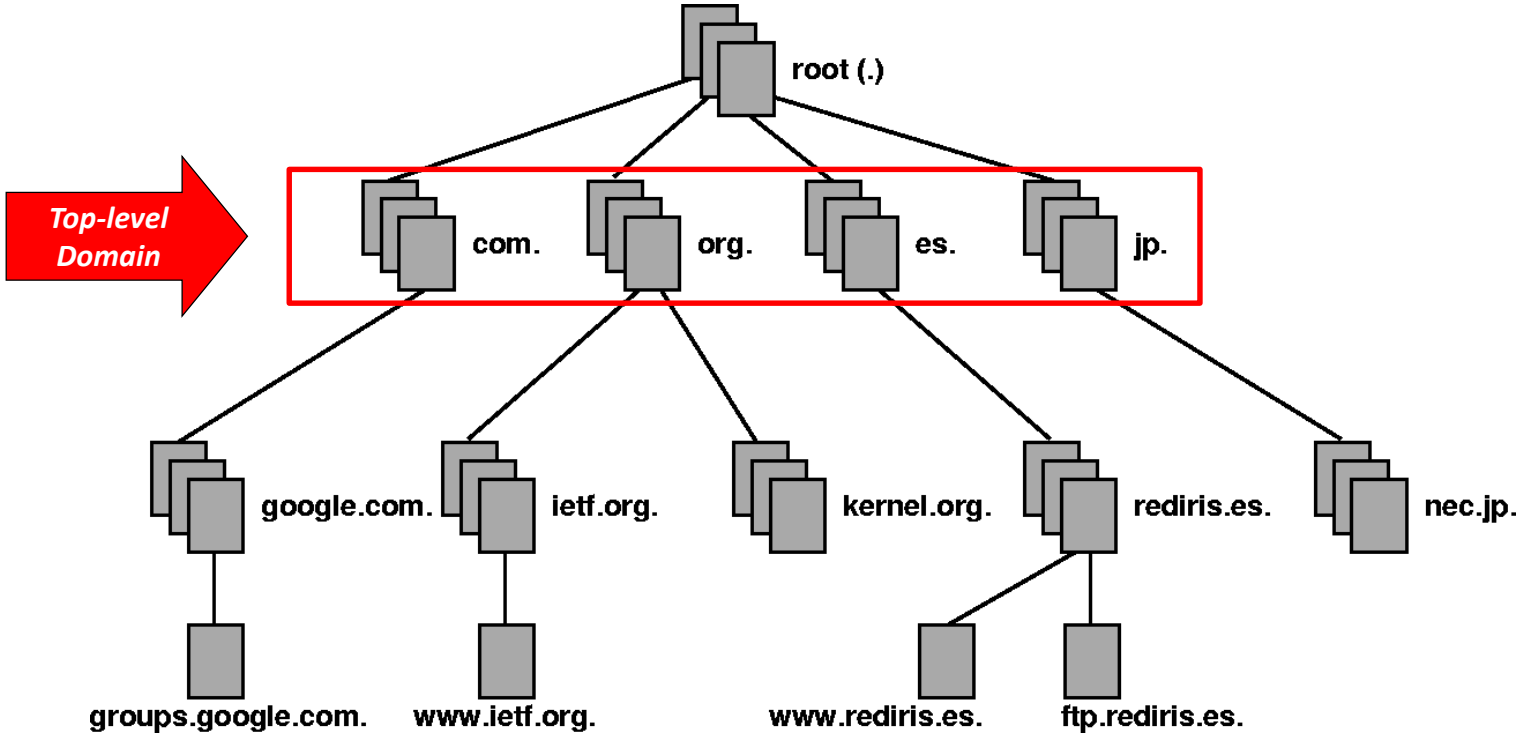
# DNS

O ponto mais alto da cadeia é denominado *root*. O servidor DNS responsável por este ponto é o *root server*. Este servidor possui todas as entradas para os servidores imediatamente abaixo dele.



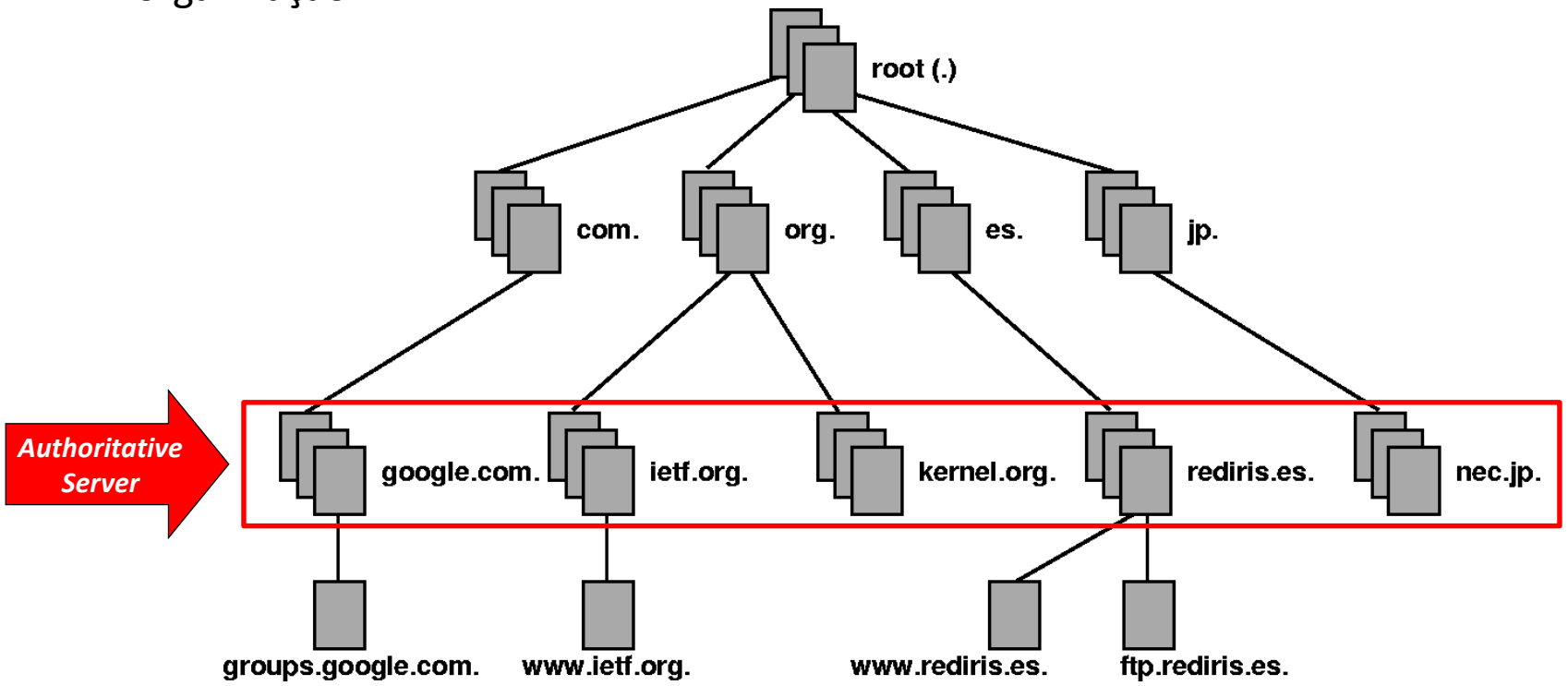
# DNS

Os *Top-level Domain* identificam domínios genéricos, como .com ou .gov, e domínios de países, como .br, .jp, .it, etc. O servidor DNS responsável por este ponto é o *TLD server*. Este servidor possui todas as entradas para os servidores imediatamente abaixo dele.

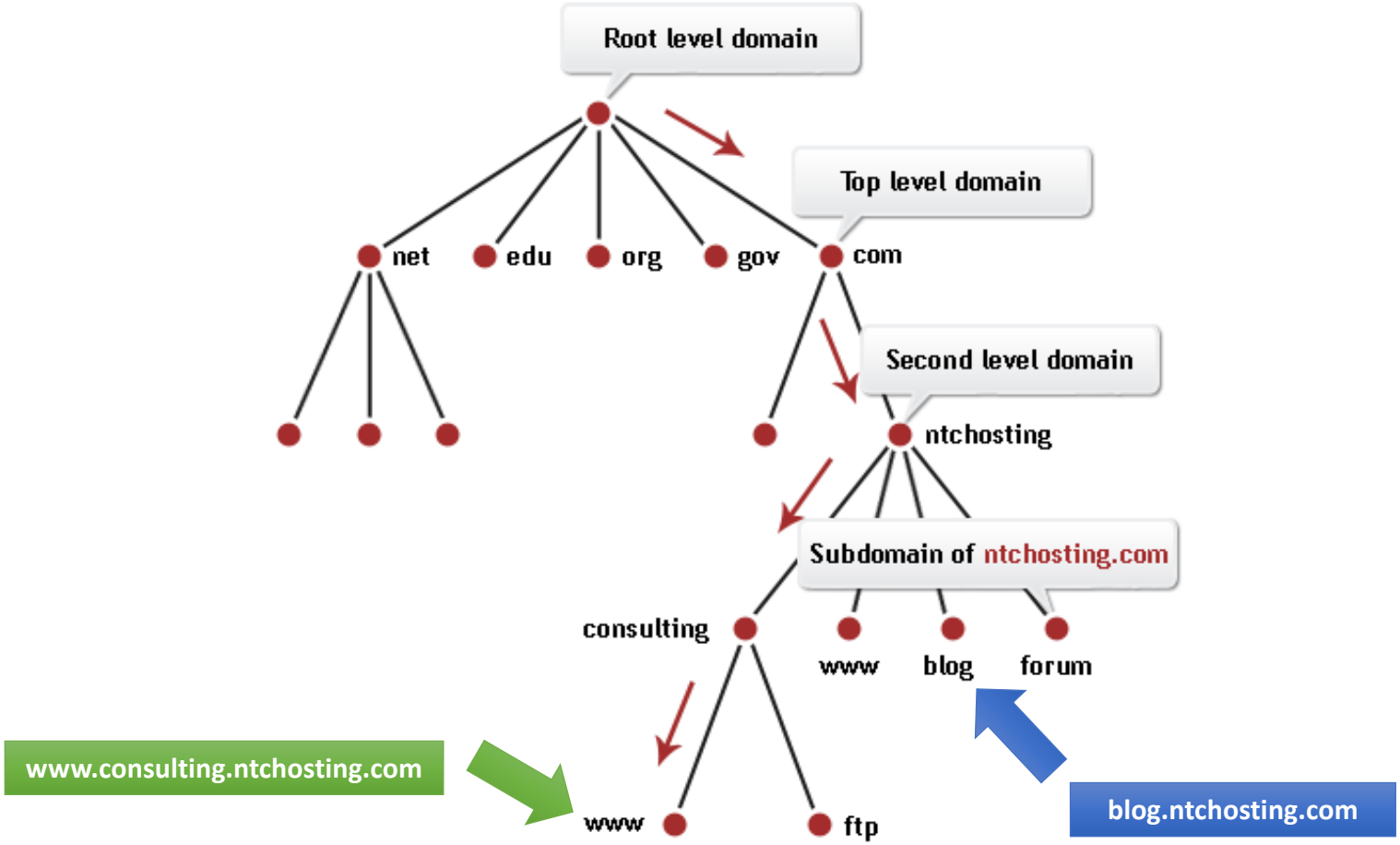


# DNS

Os servidores autoritativos são responsáveis pelas empresas ou organizações que representam. O servidor DNS responsável por este ponto é o *authoritative server*. Este servidor possui todas as entradas para os servidores e demais *hosts* dentro da organização.



# DNS – exemplo


Root level domain
Top level domain
net
edu
org
gov
com
Second level domain
ntchosting
Subdomain of ntchosting.com
consulting
www
blog
forum
www
ftp
www.consulting.ntchosting.com
blog.ntchosting.com

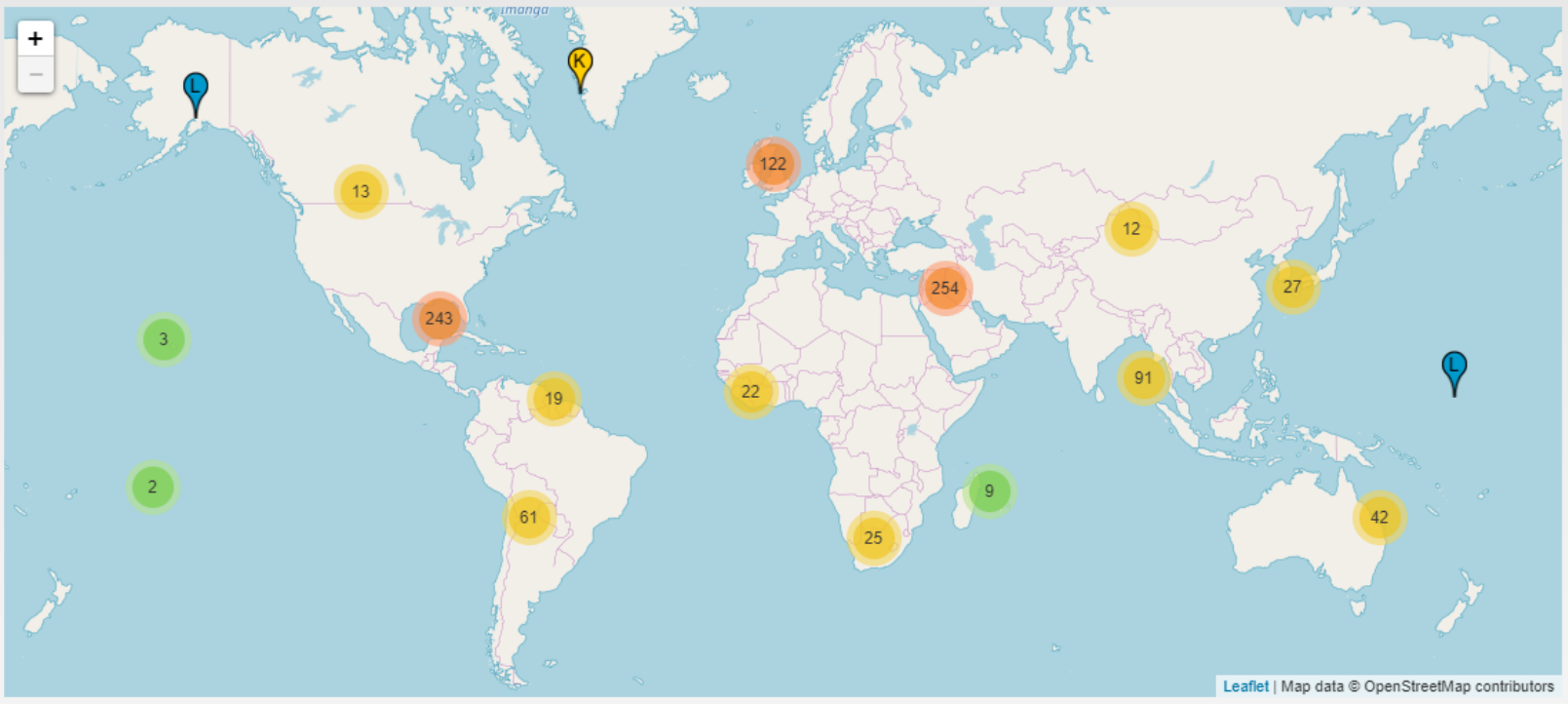
# DNS – Root servers

Os *root servers* são servidores DNS que possuem informações sobre os servidores *top-level domain*. São os primeiros a serem consultados. Ao todo são treze servidores.

| NOME | IP | OPERADOR |
|---|---|---|
| a.root-servers.net | 198.41.0.4 | Verisign |
| b.root-servers.net | 192.228.79.201 | USC-ISI |
| c.root-servers.net | 192.33.4.12 | Cogent Communications |
| d.root-servers.net | 199.7.91.13 | University of Maryland |
| e.root-servers.net | 192.203.230.10 | NASA |
| f.root-servers.net | 192.5.5.241 | Internet Systems Consortium |
| g.root-servers.net | 192.112.36.4 | Defense Information Systems Agency |
| h.root-servers.net | 128.63.2.53 | U.S. Army Research Lab |
| i.root-servers.net | 192.36.148.17 | Netnod |
| j.root-servers.net | 192.58.128.30 | Verisign |
| k.root-servers.net | 193.0.14.129 | RIPE NCC |
| l.root-servers.net | 199.7.83.42 | ICANN |
| m.root-servers.net | 202.12.27.33 | WIDE Project |

# DNS – Root servers

Os *root servers* são formados por 948 instâncias mantidas por doze operadores diferentes, distribuídas geograficamente conforme o mapa.

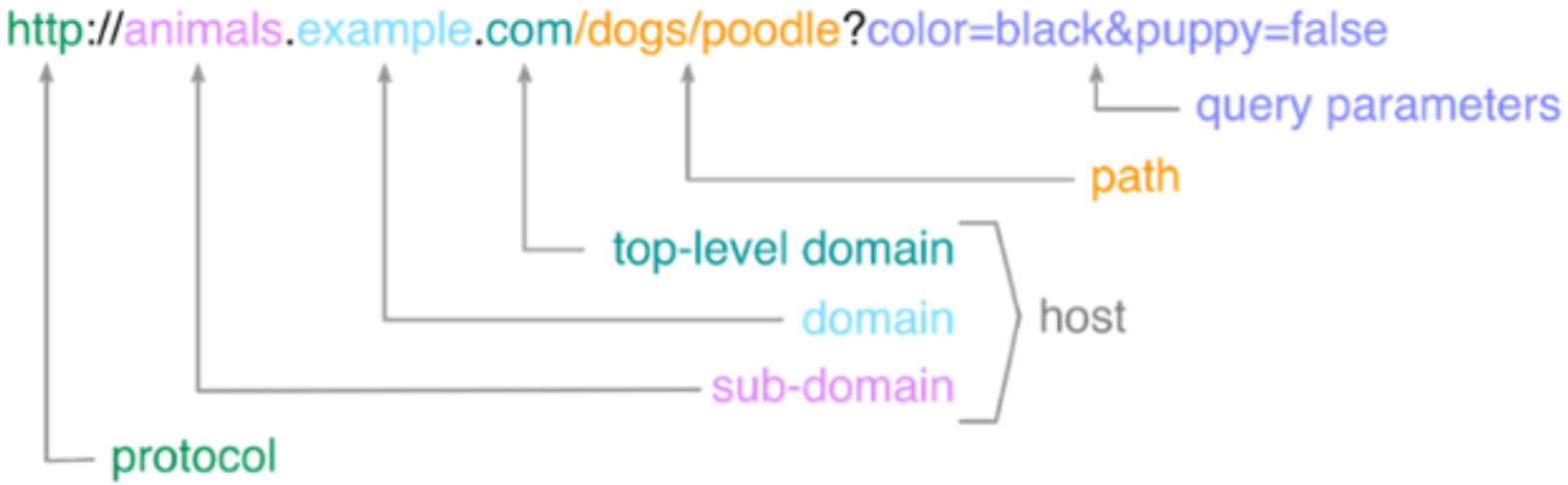



Fonte: root-servers.org

# DNS – Top-level domain

Um Top-level domain (TLD) é um dos domínios no nível mais alto da hierarquia de nomes de domínio da Internet. Os top-level domain são instalados a partir dos root server.

Para todos os domínios em níveis inferiores, é a última parte do nome do domínio, ou seja, o último rótulo de um nome de domínio totalmente qualificado, também conhecido como FQDN (Fully Qualified Domain Name).




Fonte: wikipedia.org

# DNS – Authoritative server

Um Authoritative server é um servidor que fornece respostas a perguntas sobre nomes em uma zona. Um authoritative server somente retorna respostas para consultas sobre nomes de domínio que foram especificamente configurados pelo administrador. Os servidores de nomes também podem ser configurados para fornecer respostas autoritativas a consultas em algumas zonas, enquanto atuam como um servidor de nomes em cache para todas as outras zonas.

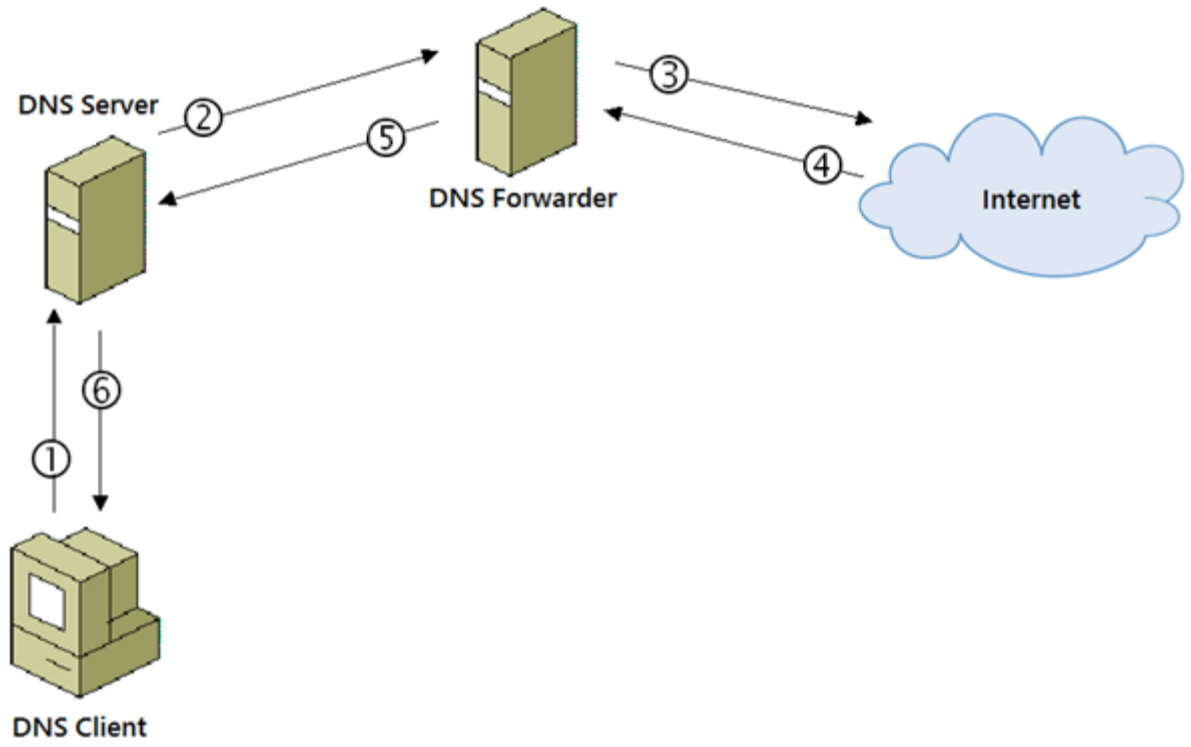
Um authoritative server pode ser do tipo principal (master) ou um servidor secundário (slave). Um servidor primário para uma zona é o servidor que armazena as versões definitivas de todos os registros nessa zona.

Um authoritative server master é identificado pelo registro SOA (Start of Authority). Um servidor secundário para uma zona usa um mecanismo de atualização automática para manter uma cópia idêntica do banco de dados do servidor primário para uma zona.

Fonte: wikipedia.org

# DNS - Forwarder

Um DNS forwarder ou encaminhador é um servidor DNS que encaminha consultas DNS para outros servidores e armazena localmente um cache de pesquisas já realizadas.



# Arquivo hosts

O arquivo *hosts.txt* é um arquivo texto que nos primórdios da Internet era mantido manualmente e disponibilizado via compartilhamento de arquivos pelo Stanford Research Institute para a associação ARPANET, contendo os nomes de host e seus endereços.

Nos sistemas operacionais modernos, este arquivo permanece como um mecanismo de resolução de nome alternativo.

No GNU/Linux, este arquivo se encontra em:

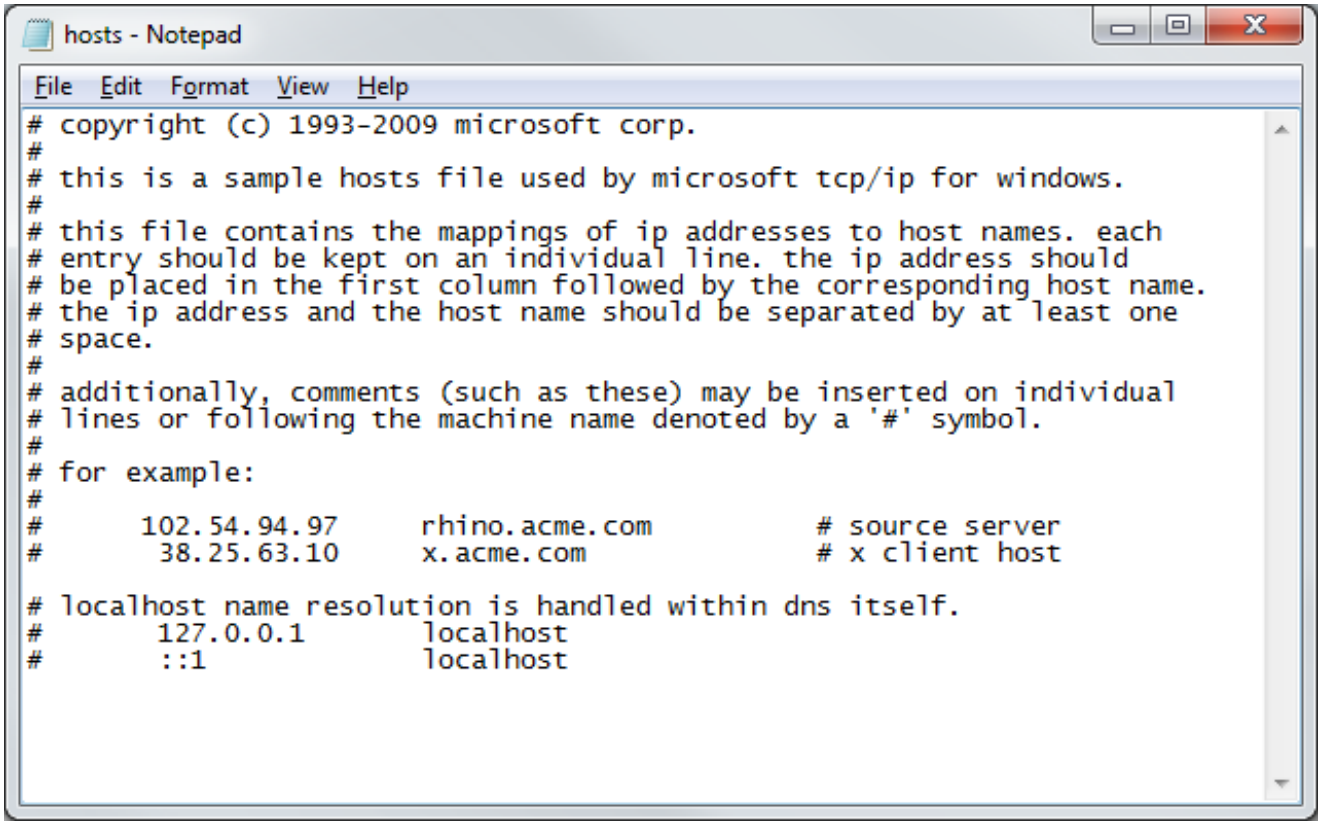
**\etc\hosts**

No Windows, este arquivo se encontra em:

**c:\windows\system32\drivers\etc\hosts.txt**

# Arquivo hosts

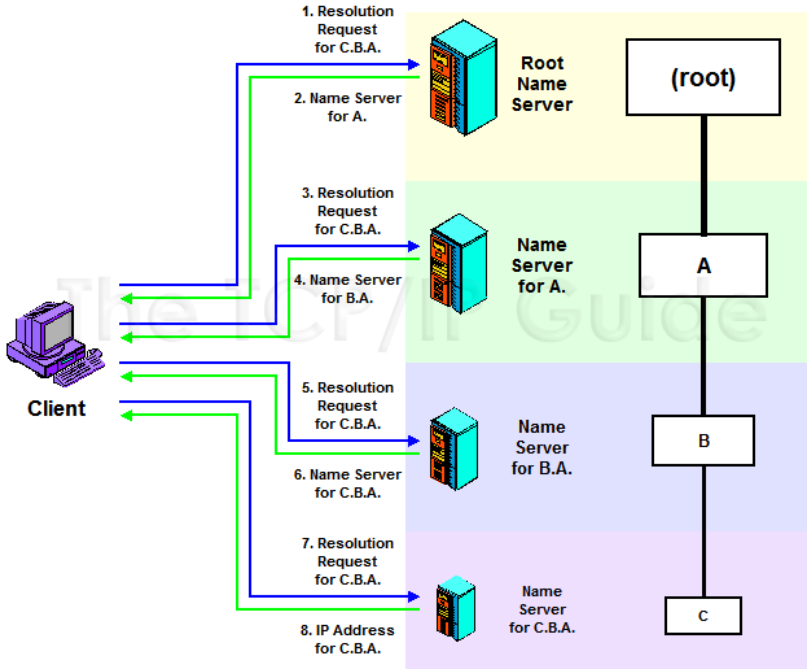
Arquivo *hosts.txt* no Windows:



# DNS – resolução de nomes
# Técnica iterativa

Quando um cliente envia uma solicitação iterativa para um servidor DNS, o servidor responde com a resposta à solicitação, ou seja, o endereço IP correspondente, ou então com o nome de outro servidor que tenha as informações.

O cliente original deve então iterar com o novo servidor, enviando uma nova solicitação para este possa responder ou fornecer outro nome de servidor.

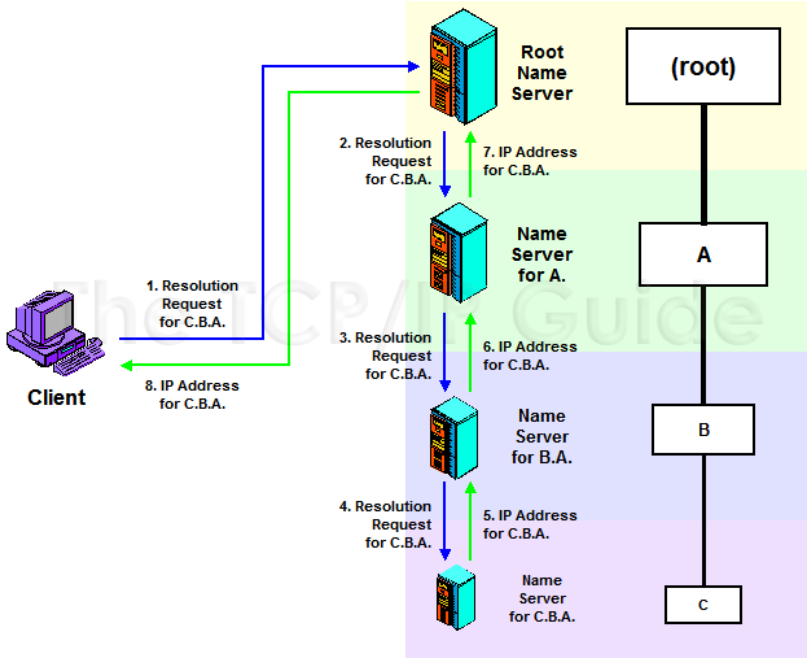



Fonte: tcpipguide.com

# DNS – resolução de nomes
# Técnica recursiva

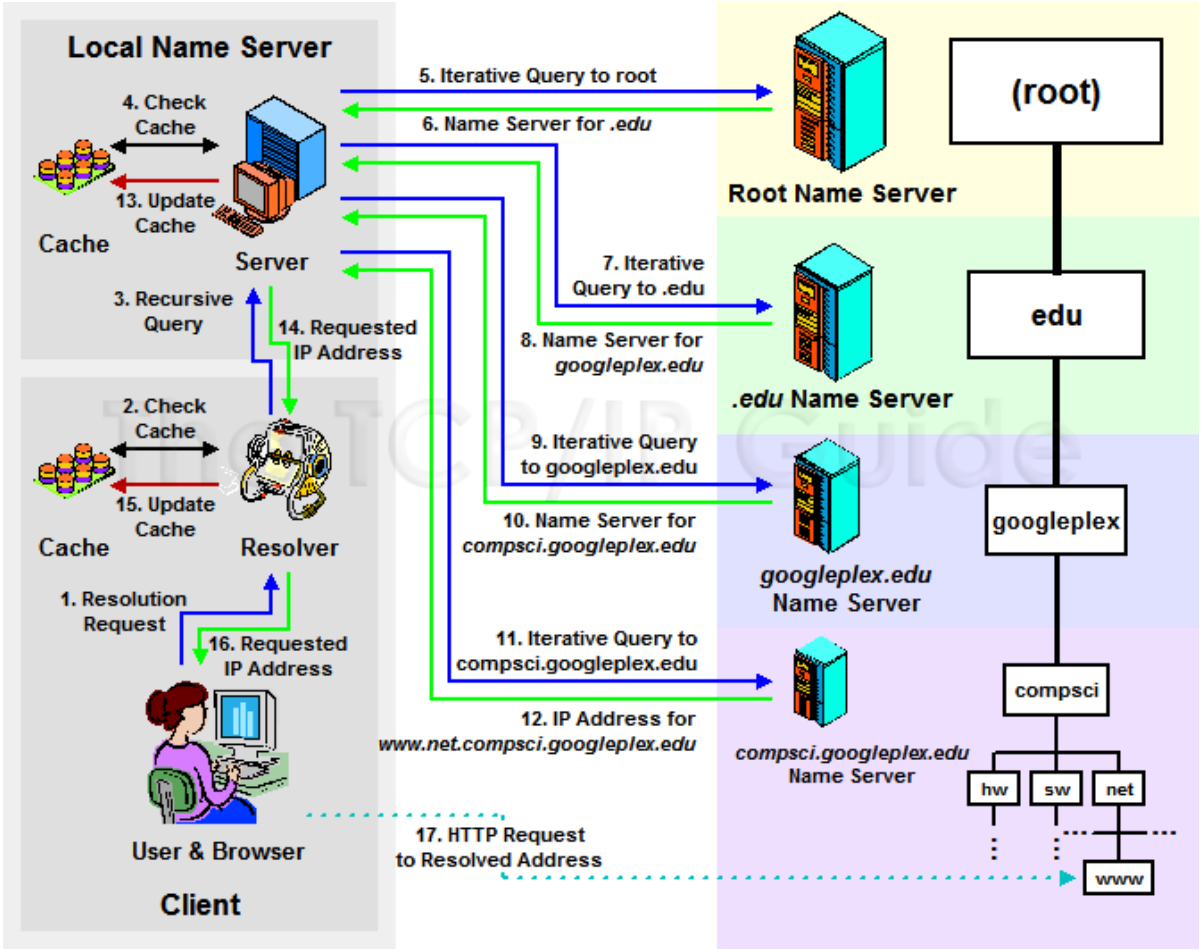
Quando um cliente envia uma solicitação recursiva para um servidor DNS, o servidor responde com a resposta se tiver a informação solicitada. Caso contrário, o servidor assumirá a responsabilidade de encontrar a resposta, tornando-se um cliente em nome do cliente original e enviando novas solicitações para outros servidores.

O cliente original envia apenas uma solicitação e, eventualmente, obtém as informações desejadas (ou uma mensagem de erro, se não estiver disponível).




Fonte: tcpipguide.com

# DNS – resolução de nomes
# Resumo



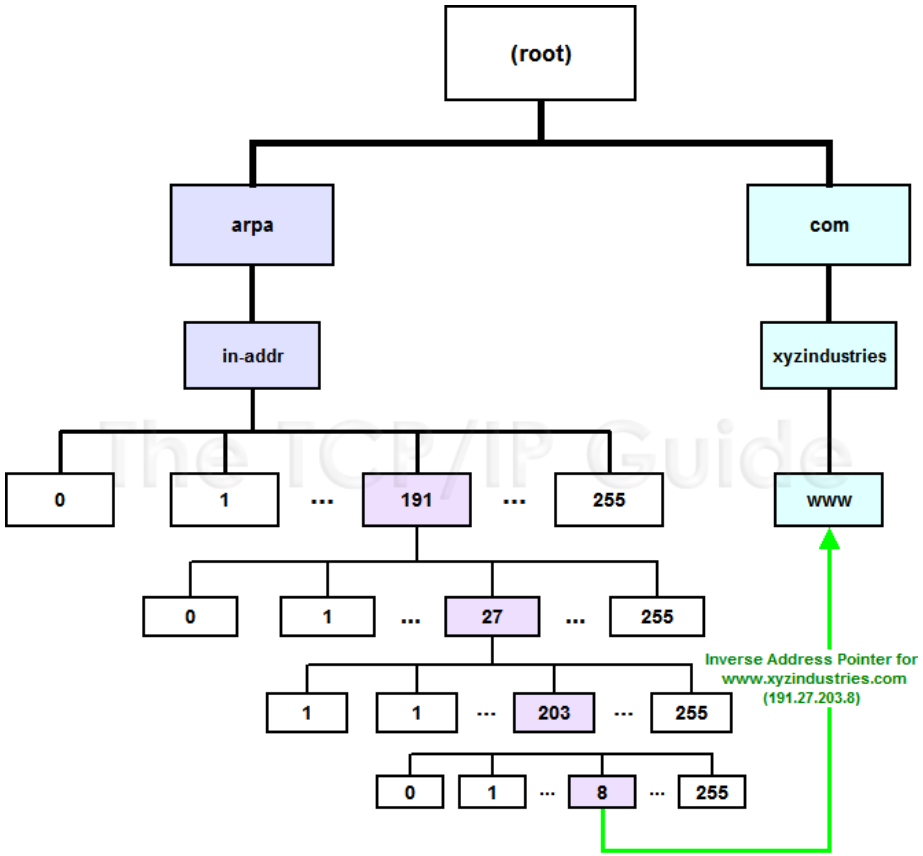

Fonte: tcpipguide.com

# DNS – resolução de nomes
# Pesquisa reversa

A hierarquia especial "IN-ADDR.ARPA" foi criada para permitir pesquisas reversas fáceis de nomes DNS.

"IN-ADDR.ARPA" contém 256 subdomínios numerados de 0 a 255, cada um dos quais tem 256 subdomínios numerados de 0 a 255 e assim por diante, abaixo de quatro níveis. Assim, cada endereço IP é representado na hierarquia.

No diagrama ao lado, o nome de domínio DNS “www.xyzindustries.com” tem um registro de recurso convencional apontando para seu endereço IP 191.27.203.8, bem como um registro de resolução reversa em 8.203.27.191.IN-ADDR.ARPA, apontando para o nome de domínio “www.xyzindustries.com”.




Fonte: tcpipguide.com

# Serviço de diretório

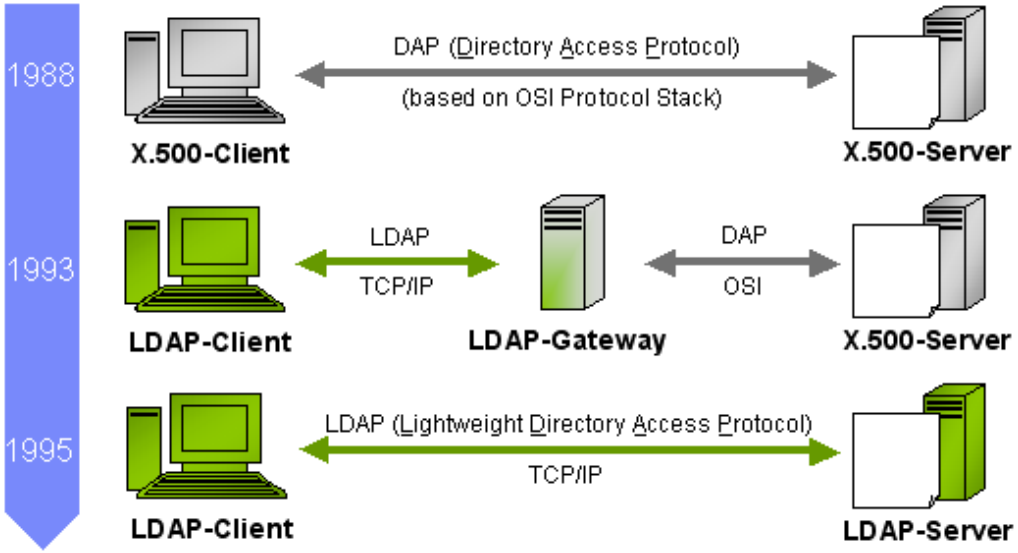
Serviço de diretório é um sistema que armazena e organiza informações sobre usuários, computadores e recursos compartilhados em uma rede de computadores.

Um serviço de diretório é útil para administrar e gerenciar usuários e recursos em uma rede de forma organizada e centralizada.

# LDAP

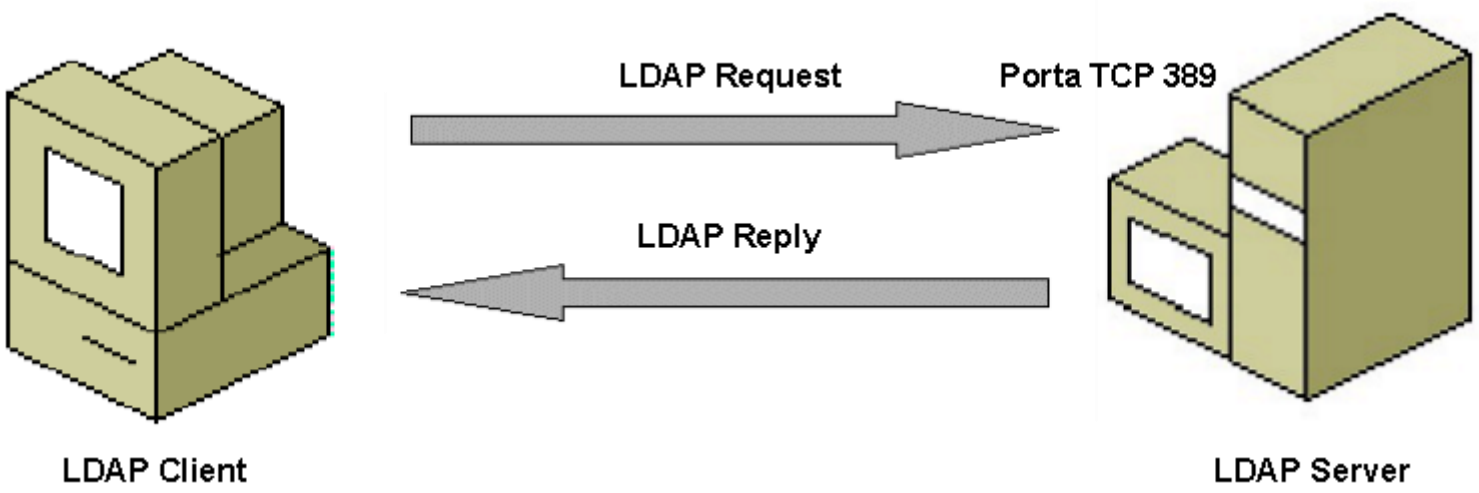
LDAP (Lightweight Directory Access Protocol) ou Protocolo Leve de Acesso à Diretórios tem a função de definir como as informações sobre usuários, computadores e recursos são armazenadas no banco de dados do repositório central do serviço de diretório.

O LDAP é um protocolo baseado no modelo cliente/servidor e foi desenvolvido como alternativa ao protocolo X.500, desenvolvido pela ITU-T e pela ISO.



# LDAP

O LDAP é um protocolo que segue o modelo cliente/servidor. O cliente LDAP conecta-se ao servidor LDAP por meio da porta TCP 389.



# LDAP

Dentre as diversas implementações do protocolo LDAP, podemos destacar o eDirectory da Novell, o OpenLDAP da comunidade GNU/Linux e o Active Directory da Microsoft.

Novell eDirectory™

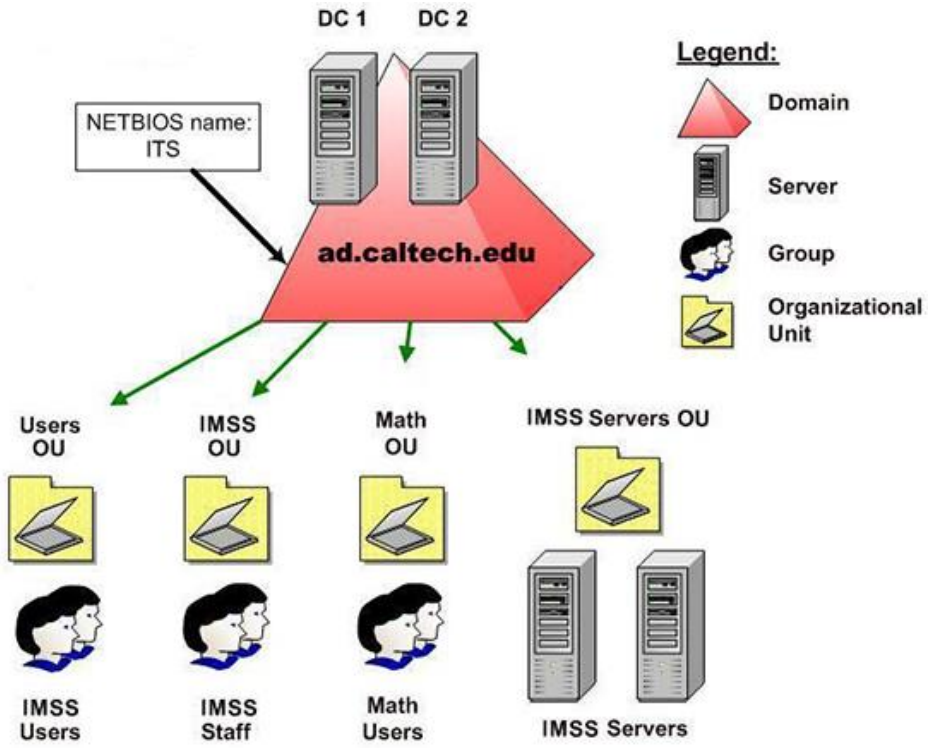

Windows Server®
Active Directory

# Active Directory

O Active Directory é a implementação da Microsoft para o serviço de diretório baseado no protocolo LDAP.

Também conhecido como AD, foi introduzido a partir do Windows Server 2000.

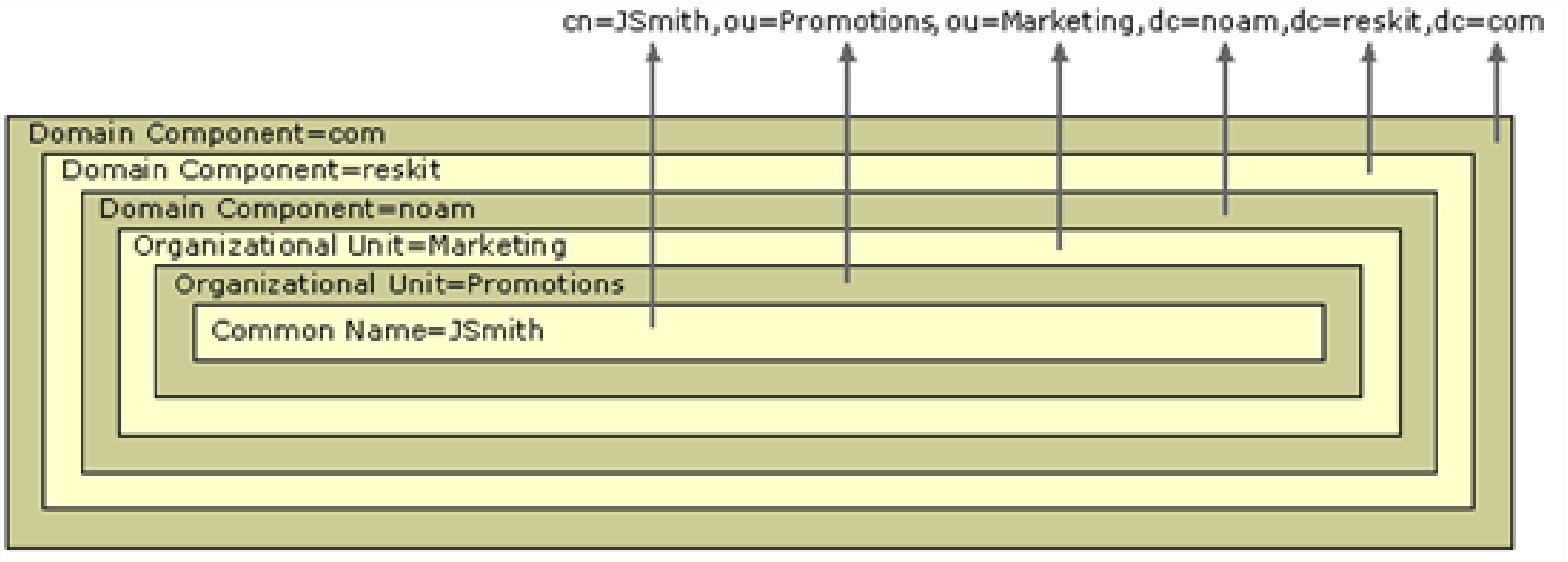
Na nomenclatura da Microsoft, uma unidade administrativa denomina-se Domínio, e é representada por um triângulo. Todo domínio deve ter, no mínimo, um controlador de domínio, também conhecido como DC, que é responsável por conter o banco de dados do serviço de diretório. Os demais servidores do domínio são conhecidos como servidores membros ou members servers.



# Active Directory – objetos

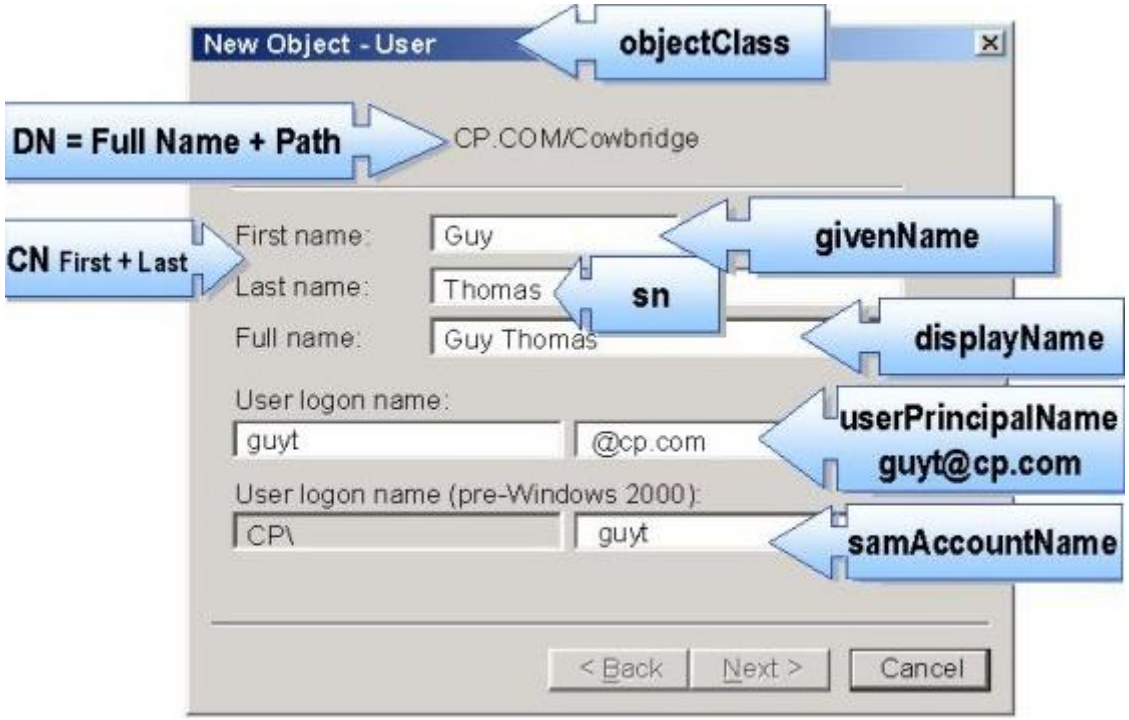
Todo objeto no Active Directory deve possuir um Distinguished Name (nome distinto), que deve ser único e exclusivo.

Isso é possível usando-se o caminho completo do objeto, incluindo o nome do objeto e todos os objetos pai para até a raiz do domínio. Desta forma o cliente LDAP consegue recuperar as informações do objeto do diretório.



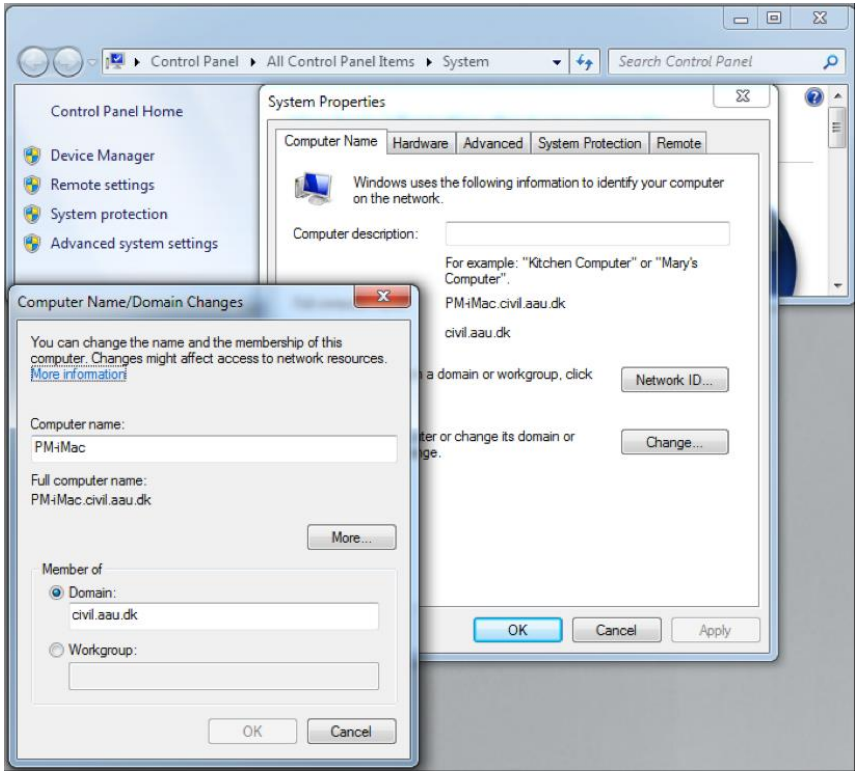
# Active Directory – atributos

Atributos são informações sobre um usuário, organização, grupo ou qualquer outro tipo de objeto. Cada atributo é associado a um tipo que fornece diversas propriedades sobre como os clientes e o servidor de diretórios devem interagir com esse atributo.



# Active Directory – logon

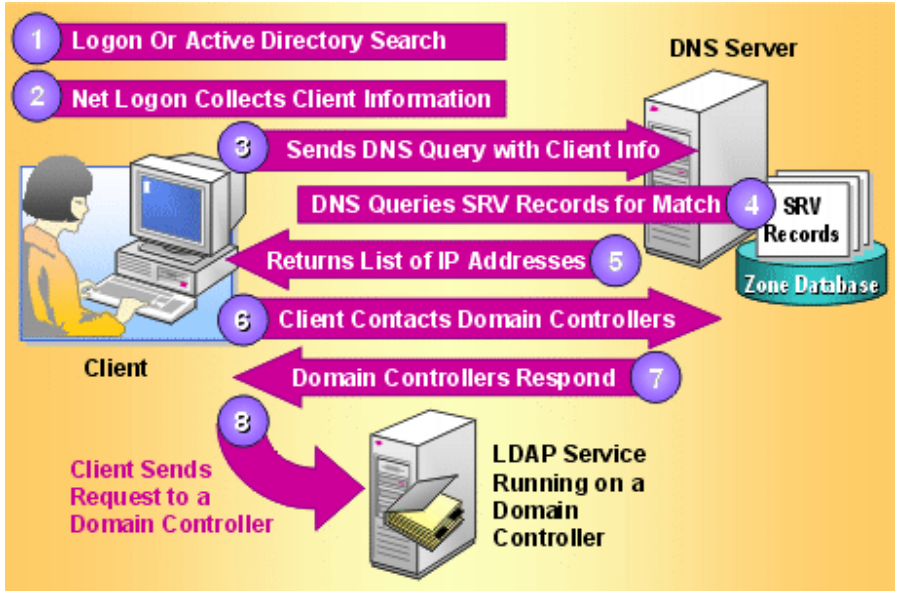
Para que os usuários possam conectar-se no domínio, as estações de trabalho devem estar registradas no domínio e os usuários devem possuir contas cadastradas no serviço de diretório.

# Active Directory – DNS

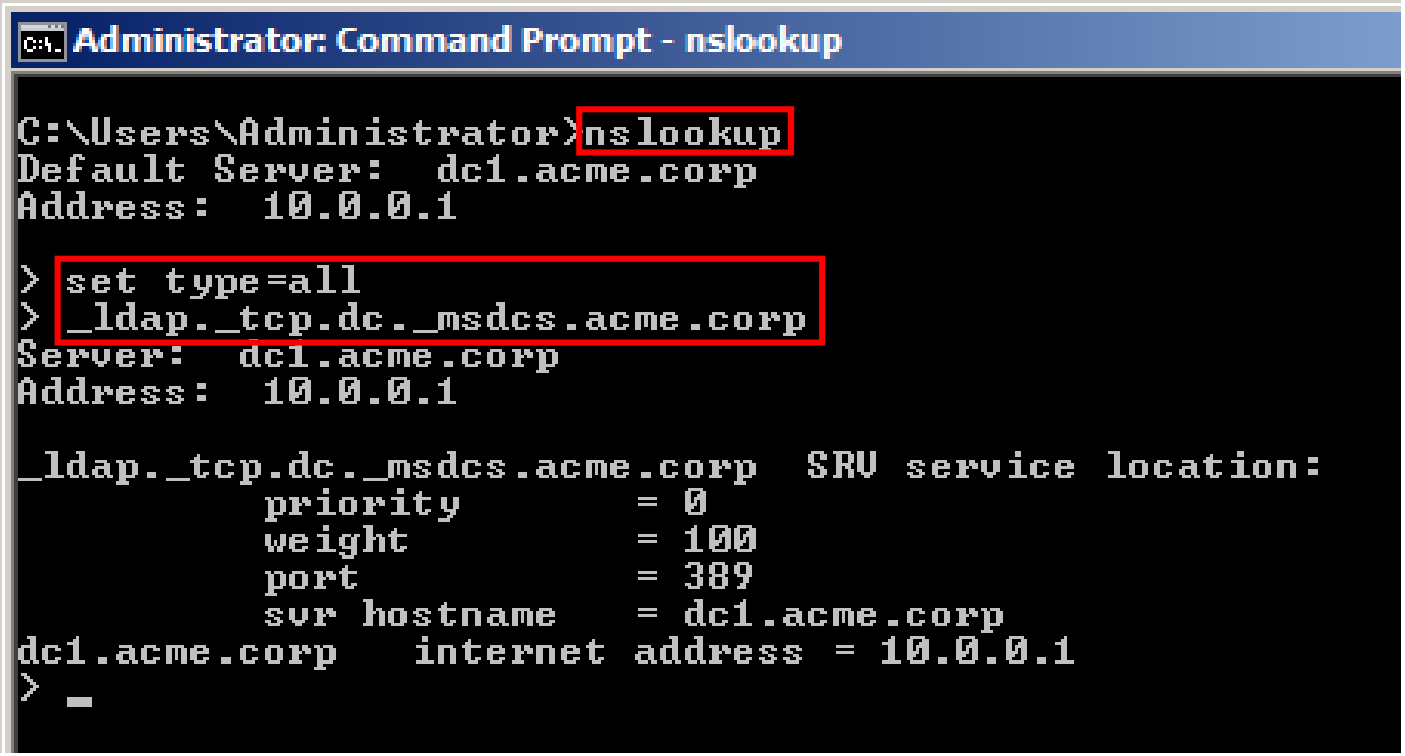
Para que a estação de trabalho possa encontrar o controlador de domínio da rede quando o usuário insere suas credencias, a estação faz uma consulta ao servidor DNS.

O Active Directory é dependente do serviço de DNS, pois sem ele a estação de trabalho não tem como encontrar o controlador de domínio responsável por autenticar aquele usuário.



# Active Directory – consulta DNS

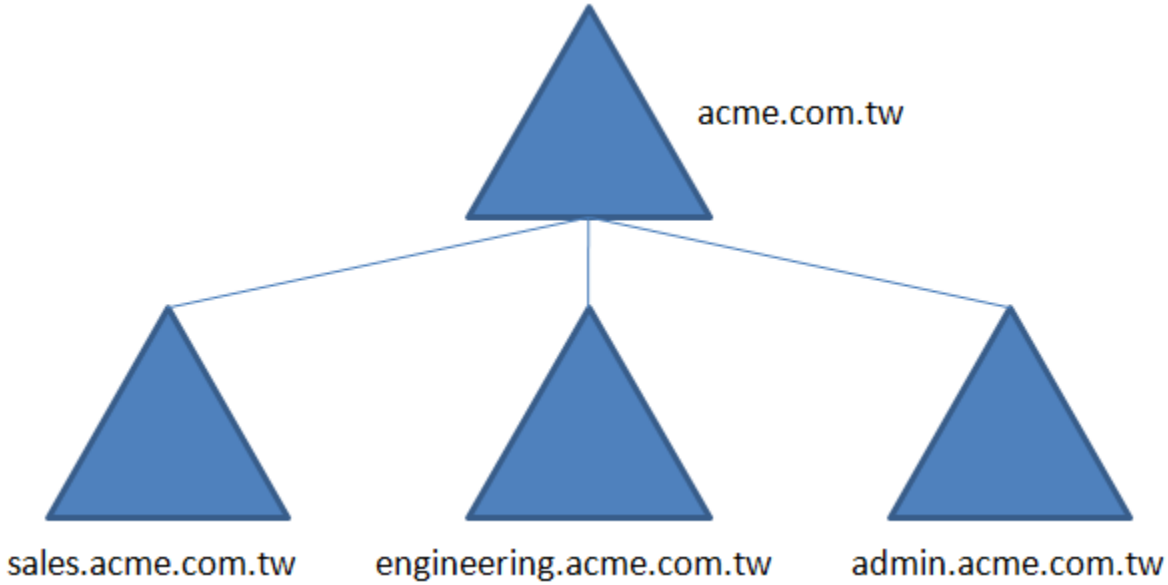
Para consultar o controlador de domínio que atende uma determinada rede, neste exemplo a rede ACME.CORP, pode-se usar o comando `nslookup`:



# Active Directory – árvore

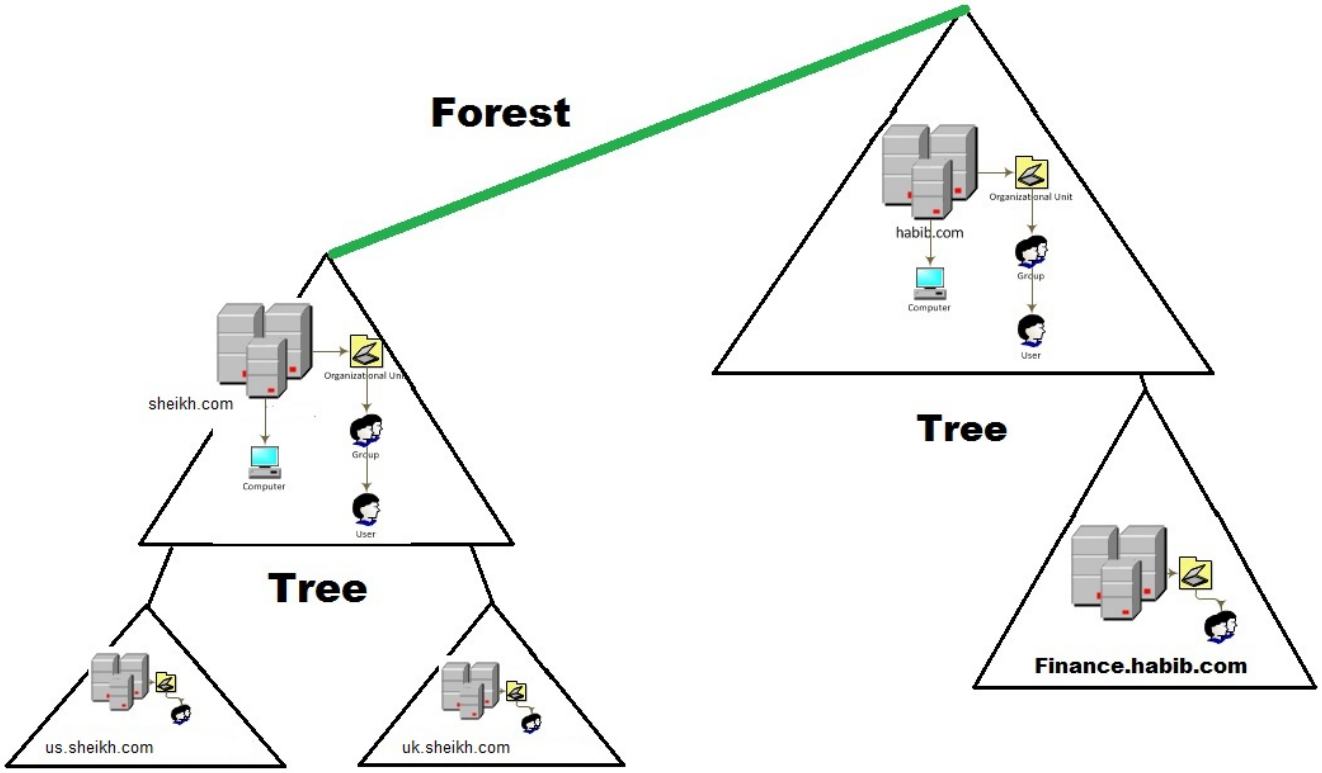
Várias unidades administrativas, ou domínios, podem ser combinados desde que compartilhem o mesmo espaço de nomes, de modo que tenhamos um domínio pai ou raiz e domínios filhos ou subdomínios. A este conjunto de domínios dá-se o nome de Árvore.

No exemplo abaixo, os domínios sales, engineering e admin são subdomínios do domínio raiz acme.com.tw.



# Active Directory – floresta

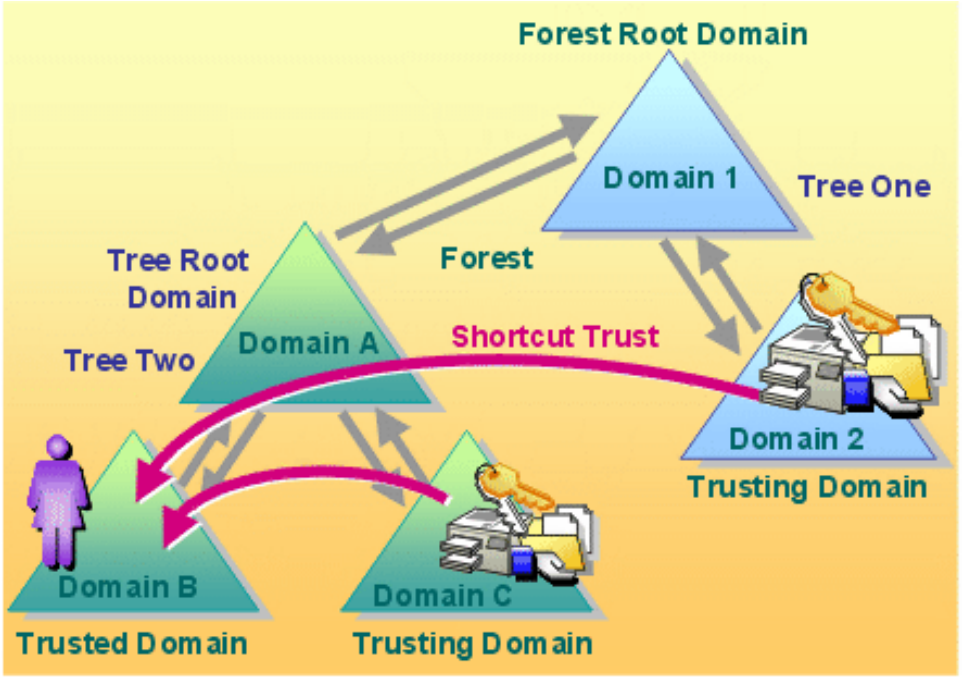
Unidades administrativas ou domínios, que não compartilham o mesmo espaço de nomes, também podem ser combinados. Neste caso teremos uma Floresta.



# Active Directory – trust

O trust ou relação de confiança é um canal de autenticação que permite que usuários de um domínio possam acessar recursos em outro domínio.

Pode ser do tipo direta, quando um domínio é subdomínio de outro; ou transitiva, quando dois domínios são subdomínios de uma mesma raiz.



# Para saber mais...

… leia o Capítulo 5 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

… leia o material online sobre Domain Name System, de Júlio Battisti.

... leia a apostila Domain Name Service Configuração e Administração, de Rubens Queiroz de Almeida.

… leia o documento sobre Arquitetura do Active Directory, da Microsoft.

# Módulo 18

Sincronização

# Introdução

Uma questão importante a ser considerada na comunicação entre processos diz respeito a como estes são sincronizados. Em um sistema centralizado, quando um processo quer saber a hora, este faz uma chamada de sistema e o sistema operacional responde com a hora exata do computador.

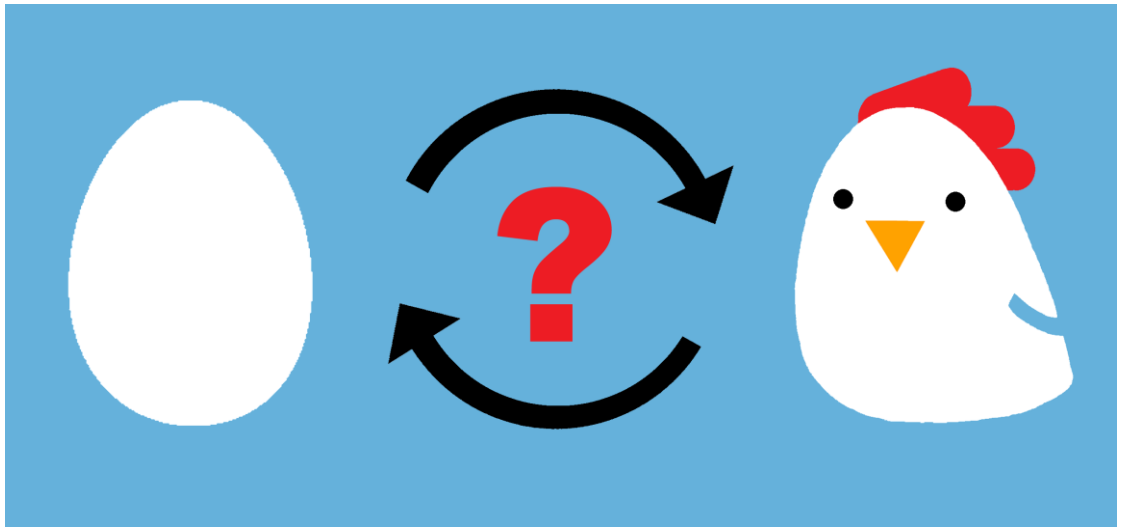
No entanto, em sistemas distribuídos a sincronização de relógios não é algo trivial, pois é muito difícil implementar um relógio único que seja globalmente compartilhado entre os diversos processos espalhados geograficamente.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Introdução

**O que é o tempo?**

Se considerarmos dois eventos, um ocorrendo depois do outro, podemos definir o tempo em termos de causalidade, ou seja, se o primeiro evento provocar o segundo, então podemos dizer certamente que o segundo evento ocorre depois do primeiro.




Fonte: ntp.br

# Introdução

**O que é o tempo?**

Mas quanto o segundo evento ocorre depois do primeiro? A resposta é a quantidade que costuma-se chamar de tempo, ou, mais precisamente, de intervalo de tempo.

Essa quantidade pode ser medida por um dispositivo chamado relógio, que trabalha de forma contínua fornecendo indicações instantâneas, que podem ser chamados de momentos.

Então, se o primeiro evento ocorre em um momento $m_1$ e o segundo evento ocorre em outro momento $m_2$, o intervalo de tempo entre os dois eventos é t, onde:

$$t = m_2 - m_1$$

Fonte: ntp.br

# Introdução

**O que é o tempo?**

Assim, podemos considerar que o tempo é o intervalo entre dois eventos, ou o momento indicado pelo relógio. O tempo é medido em segundos, que é uma unidade do SI (Sistema Internacional de Unidades).

Historicamente o segundo era medido com base no dia solar médio (1/86400 do dia solar médio), mas a rotação da Terra é bastante imprecisa. Então, em 1954, definiu-se o segundo com base na rotação da Terra em torno do Sol (1/31.556.925,9747 do tempo que levou a Terra a girar em torno do Sol à partir das 12h de 04/01/1900). Contudo, a rotação da Terra em torno do Sol também é imprecisa.

Desde 1967 o segundo é definido com base na medição de relógios atômicos, como:

"O segundo é a duração de 9.192.631.770 períodos da radiação correspondente à transição entre dois níveis hiperfinos do estado fundamental do átomo de césio 133."

Fonte: ntp.br

# Introdução

**As escalas de tempo**

As escalas de tempo podem ser definidas como sistemas de ordenamento de eventos. Pode-se entendê-las também como convenções sobre a forma de medir e representar o tempo. As escalas não podem ser ambíguas, devem ser estáveis e homogêneas.

Existem diversas escalas de tempo. As mais importantes no contexto do NTP são:

- **TAI (Tempo Atômico Internacional ou Temps Atomique International):** É calculada pelo Bureau International des Poids et Mesures (BIPM) a partir da leitura de mais de 260 relógios atômicos localizados em institutos e observatórios de metrologia ao redor do mundo. No Brasil o Observatório Nacional participa da geração do TAI. Estima-se que o erro do TAI em relação a um relógio imaginário perfeito esteja em torno de 100 ns por ano;

Fonte: ntp.br

# Introdução

**As escalas de tempo**

- **TUC (Tempo Universal Coordenado) ou UTC (Universal Time Coordinated)**: É a base para o tempo legal no mundo todo, inclusive no Brasil. O UTC acompanha o TAI, mas é disciplinado pelo período solar. Para ajustar o UTC em relação ao período solar é acrescentado ou removido um segundo sempre que necessário. Isso é chamado de **segundo intercalado**, ou ***leap second***, e a necessidade de aplicar e em que data fazê-lo é determinada pela International Earth Rotation & Reference Systems Service (IERS). Assim, assegura-se que o Sol esteja exatamente sobre o meridiano de Greenwich as 12 horas, com um erro máximo de 0,9 s. O UTC é então o sucessor do GMT (*Greenwich Mean Time*), que era a escala de tempo utilizada quando a definição de segundo era baseada no dia solar;

Fonte: ntp.br

# Introdução

**As escalas de tempo**

- **TA(k)**: Tempo Atômico. É a designação dada às escalas de tempo materializadas por um relógio atômico específico. Por exemplo, a designação TA(ONRJ) indica a escala mantida pelo Observatório Nacional no Rio de Janeiro, e que contribui na geração do TAI. TA(NIST) indica a escala mantida pelo US National Institute of Standards and Technology;
- **GPS Time**: Os satélites GPS adotaram uma escala sincronizada com o UTC em 1980, mas desde então não sofreram as correções dos segundos intercalados. O GPS está adiantado em relação ao UTC em 14 s. Atenção aqui porque os receptores GPS normalmente apresentam o tempo em UTC, fazendo a "correção" internamente. Desconsiderando-se a questão dos segundos intercalados, o GPS (por definição) não diverge do UTC mais do que 1 µs. Na prática, o erro não passa de algumas dezenas de nanosegundos;

Fonte: ntp.br

# Introdução

**As escalas de tempo**

- **Tempo Local**: Diferentes regiões do mundo adotam fusos horários distintos. O tempo local é uma escala de tempo baseada numa diferença em relação ao UTC, de forma a adequá-lo ao tempo solar local. No Brasil, a legislação determina os fusos em seus respectivos Estados que o adotam. A Lei n° 11.662 de 24/04/2008 e o Decreto n° 6.558 de 08/09/2008 definem:

| Diferença em relação ao UTC | Sem horário de Verão | No horário de Verão |
|---|---|---|
| UTC-2 | Ilhas de Fernando de Noronha, Trindade, Martin Vaz, Penedos de São Pedro e São Paulo e o Atol das Rocas. | Ilhas de Fernando de Noronha, Trindade, Martin Vaz, Penedos de São Pedro e São Paulo e o Atol das Rocas. Estados da região Sudeste e Sul, Goiás e o Distrito Federal. |
| UTC-3 | Estados da região Nordeste, Sudeste, Sul, além do Distrito Federal, Goiás, Tocantins, Amapá e Pará. | Estados da região Nordeste, Tocantins, Amapá, Pará, Mato Grosso e Mato Grosso do Sul. |
| UTC-4 | Estados de Roraima, Rondônia, Mato Grosso, Mato Grosso do Sul, Amazonas e Acre. | Estados de Roraima, Rondônia, Amazonas e Acre. |

Fonte: ntp.br

# Introdução

**A importância da sincronização**

Uma importante propriedade do tempo é sua monotonicidade, que significa que o tempo sempre avança. Essa parece ser uma propriedade simples, óbvia e fácil de ser mantida, mas de fato não é. Relógios implementados em software, como é o caso dos relógios utilizados pelos diversos Sistemas Operacionais, podem ser facilmente ajustados, intencionalmente ou não, para representar um tempo no passado.

Diferentes softwares e aplicações podem ser sensíveis a problemas relativos à sincronização do tempo de formas diversas. Dentre os possíveis problemas com a sincronização, podemos considerar:

- um computador, ou grupo de computadores, **com o tempo diferente da hora legal**;
- um computador cujo **tempo foi ajustado para o passado**;
- um grupo de computadores **discordando entre si quanto ao tempo correto** (isso implica que ao menos *n-1* integrantes desse grupo terão o tempo diferente da hora legal, implica também que haverá computadores para os quais, tendo como referência seu relógio local, o relógio de um ou mais de seus companheiros de grupo, estará no passado).

Fonte: ntp.br

# Introdução

**A importância da sincronização**

Quando se trata de um computador isolado a exatidão em relação à uma referência de tempo como a hora legal brasileira não é tão importante. Nesse caso o mais importante é manter a monotonicidade do tempo. Além disso, eventuais ajustes no relógio devem ser, sempre que possível, graduais. Saltos no tempo, mesmo de alguns poucos segundos, para o futuro podem ser ruins, e para o passado, desastrosos.

Como exemplos de aplicações afetadas pelo tempo pode-se citar:

- **Sistemas de distribuição de conteúdo** (*www, usenet news, etc*): Utilizam estampas de tempo para controlar a expiração dos documentos e o cache. Servidores com o tempo errado podem causar perda de informações ou impedir o acesso às mesmas;
- **Sistemas de arquivos** (*filesystems*): Alguns eventos importantes como a criação e modificação de arquivos são marcados por estampas de tempo. Algumas aplicações lêem essas informações e delas dependem. Se alguma dessas datas estiver no futuro, as aplicações podem agir de forma indevida, ou mesmo deixar de funcionar por completo. Como exemplos de aplicações sensíveis a essa situação pode-se citar os sistemas de controle de versão (como o *cvs*), sistemas de compilação automática (*make*), sistemas de backup de dados e sistemas de banco de dados;
- **Agendadores de eventos**: Aplicações como o *cron* e o *at* dos sistemas Unix dependem do tempo correto para funcionarem;

Fonte: ntp.br

# Introdução

**A importância da sincronização**

- **Criptografia**: Muitas técnicas criptográficas fazem uso de estampas de tempo para os eventos e chaves para prevenir alguns tipos de ataques. Se os computadores envolvidos não estiverem sincronizados entre si, a autenticação e comunicação criptografada podem falhar;
- **Protocolos de comunicação e aplicações de tempo real**: Essas aplicações, que incluem as **Interfaces Gráficas**, fazem uso de filas de eventos, *timeouts*, *timers*, e outros recursos de software ligados ao tempo. Para seu correto funcionamento é necessário garantir a monotonicidade, uma boa resolução, e a continuidade (ausência de saltos) no tempo;
- **Sistemas transacionais e bancos de dados distribuídos**: Dependem de relógios exatos e muitas vezes, de sua sincronia com a hora legal. Como exemplo dessas aplicações pode-se citar o *Home Banking*, o *Home Broker*, os sistemas *EDI*, etc. As bolsas de valores, por exemplo, tem horários bem definidos de início e término do pregão. A Receita Federal aceita as declarações de Imposto de Renda geralmente até a meia noite da data limite para a entrega.

Fonte: ntp.br

# Introdução

**A importância da sincronização**

É importante também do ponto de vista de **segurança de redes** que os relógios dos computadores estejam sincronizados. Investigações relacionadas a incidentes de segurança tornam-se impossíveis caso os servidores envolvidos e os diversos arquivos de *log* discordem entre si em relação às estampas de tempo dos eventos.

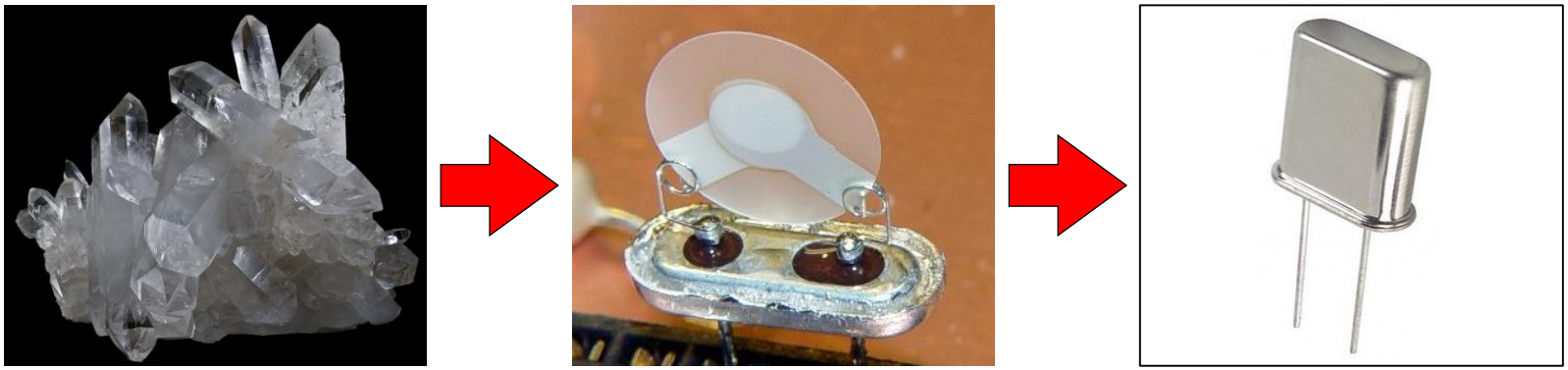
Fonte: ntp.br

# Relógios físicos

Todos os computadores possuem um circuito temporizador que monitora a passagem do tempo.

Este circuito é baseado em um cristal de quartzo lapidado e usinado com precisão, que quando submetido a uma determinada tensão elétrica, oscila a uma frequência bem definida.

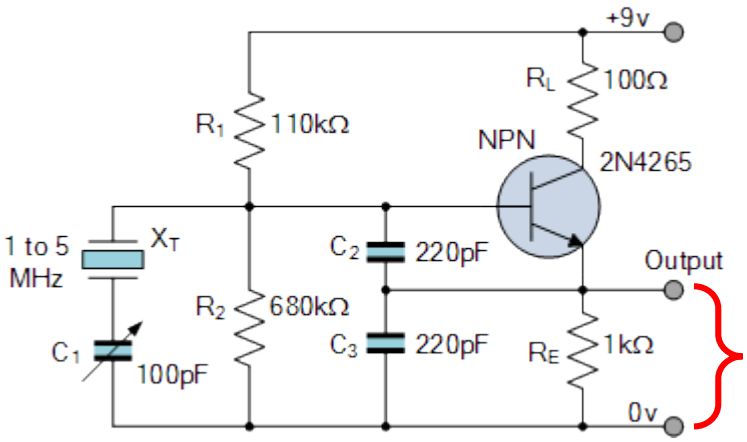
Esta característica do material é denominada de efeito piezoelétrico.

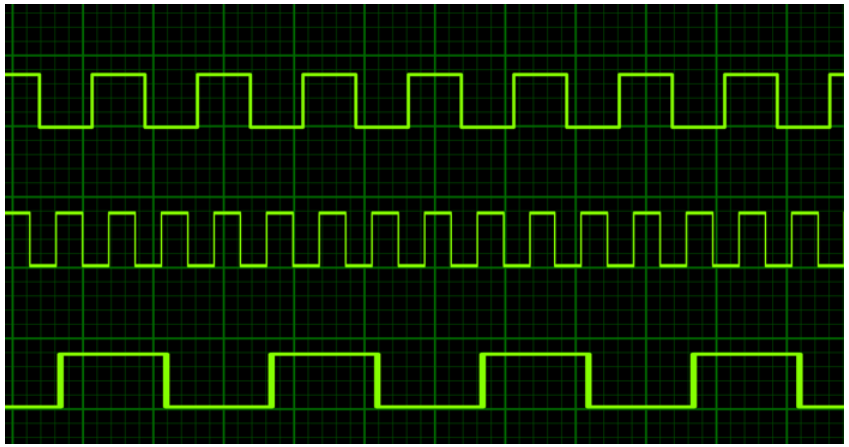


Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Relógios físicos

A partir de um circuito que usa um cristal de quartzo, é possível gerar uma saída de onda quadrada com uma frequência bem conhecida e estável.



# Relógios físicos

**Propriedades importantes dos relógios**

- **Exatidão (*Accuracy*):** É quanto o relógio está próximo à referência, ou seja, indica se o relógio está “certo” ou “errado” ou ainda quanto o relógio está “certo” ou “errado”.

Se um relógio de computador funciona de forma isolada, sua exatidão tende a piorar com o tempo, por conta de erros sistemáticos na frequência. O relógio atrasa ou adianta conforme o tempo passa. Um relógio típico de computador, se funcionando livremente, pode adiantar-se ou atrasar-se desde alguns segundos até cerca de 1 minuto por semana.

Para garantir uma melhor exatidão, a alternativa é **disciplinar o relógio** em relação à fontes de tempo e frequência mais confiáveis. Disciplinar o relógio significa que ele é sincronizado (fase) e sintonizado (frequência) com uma fonte mais estável de tempo.

Fonte: ntp.br

# Relógios físicos

**Propriedades importantes dos relógios**

- **Precisão (*Precision*), Resolução (*Resolution*) e Granularidade (*Granularity*):** Por resolução, se entende o valor do menor incremento possível do contador do relógio. A resolução de um relógio de computador é determinada pela frequência das interrupções de *hardware* que fazem funcionar o contador. Os valores normalmente variam entre 100 Hz e 1 kHz, o que resulta em resoluções de 10 ms a 1 ms.

Em alguns casos é possível utilizar outras fontes de frequência (maiores), como por exemplo o relógio da CPU, para interpolar os valores obtidos pelo relógio, conseguindo assim uma resolução melhor. Isso geralmente é chamado de granularidade do relógio. Com isso, resoluções de aproximadamente 1 µs são comuns hoje. Implementações de *software* como o *nanokernel*, que hoje é parte integrante do FreeBSD e do Linux, permitem resoluções da ordem de nanosegundos.

Por precisão, entende-se geralmente o menor incremento de tempo que pode ser lido pelo computador. Pode ser um valor maior do que a resolução, já que ler o relógio é uma tarefa realizada por *software* e há um certo tempo e incerteza envolvidos nela. Ou pode ser menor, quando o *software* consegue ler o relógio mais rápido do que esse pode contar.

Fonte: ntp.br

# Relógios físicos

**Propriedades importantes dos relógios**

- **Dispersão (*Dispersion*):** É o desvio ou erro estimado nas leituras do relógio. Pode ser causado por flutuações de curta duração na frequência do oscilador, por erros de medida ocasionados por excesso de utilização do processador, latência causada por interrupções, latência na rede, etc.
- **Variação (*Jitter*):** É o desvio ou erro nas leituras de relógio. Pode ser causado por flutuações de curta duração na frequência do oscilador, por erros de medida ocasionados por excesso de utilização do processador, latência causada por interrupções, latência na rede, etc.
- **Deslocamento (*Offset* ou *time offset*):** É a diferença de tempo entre dois relógios.

Fonte: ntp.br

# Relógios físicos

**Propriedades importantes dos relógios**

- **Envelhecimento (*Aging* ou *Ageing*):** É a instabilidade na frequência do oscilador causada por fatores internos. Ou seja, quanto a frequência do relógio varia com o tempo quando os fatores externos como radiação, pressão, temperatura e umidade são mantidos constantes.

O envelhecimento (*ageing*) de um relógio atômico de rubídio é da ordem de $5x10^{-11}$ ano, enquanto os relógios atômicos de césio modernos não apresentam envelhecimento.

- **Escorregamento (*Drift*):** É a instabilidade na frequência do oscilador. Ou seja, quanto a frequência do relógio varia com o tempo. Essa instabilidade pode ser causada por fatores externos, como variações na radiação, pressão, temperatura ou umidade e pelo envelhecimento (*ageing*).

Ao sintonizar os osciladores de dois relógios, se as frequências permanecerem constantes dali em diante, as diferenças de deslocamento também permanecerão.

Fonte: ntp.br

# Sistema de posicionamento global

O Sistema de Posicionamento Global ou GPS (Global Positioning System), é um sistema distribuído baseado em satélites lançado em 1978 para aplicações militares.

O sistema de GPS tornou-se público a partir de 1983, quando passou a usado principalmente na aviação civil.

O sistema atual de GPS utiliza 29 satélites que circulam em uma órbita a uma altura aproximada de 20.000 km.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

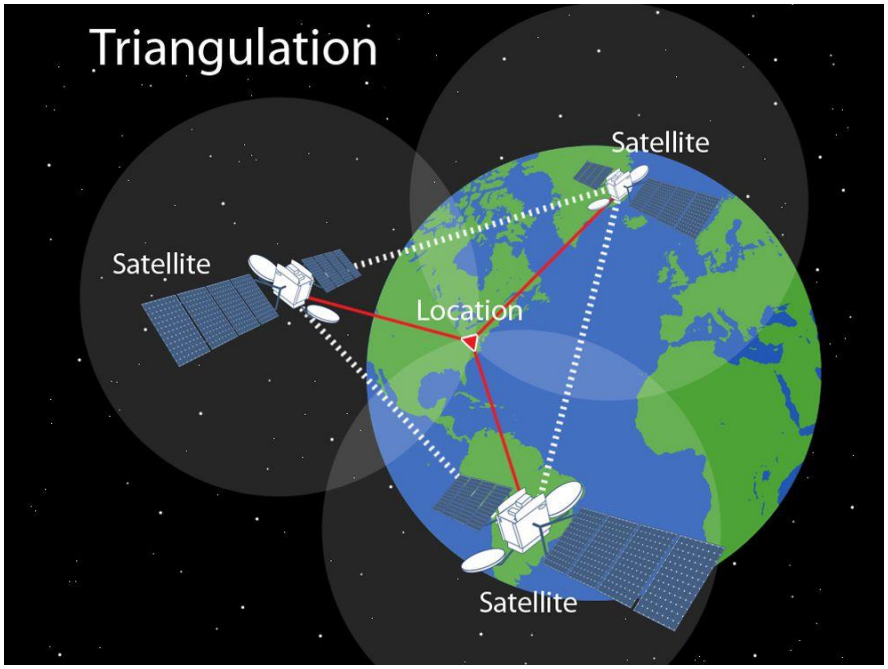
# Sistema de posicionamento global

Cada satélite tem até quatro relógios atômicos que são calibrados periodicamente por estações especiais na Terra.

Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Sistema de posicionamento global

Um satélite transmite continuamente sua posição por meio de difusão e anexa marcas de tempo a cada mensagem, informando sua hora local. Essa transmissão permite que todo receptor na Terra calcule com precisão sua própria posição usando, em princípio, somente três satélites.

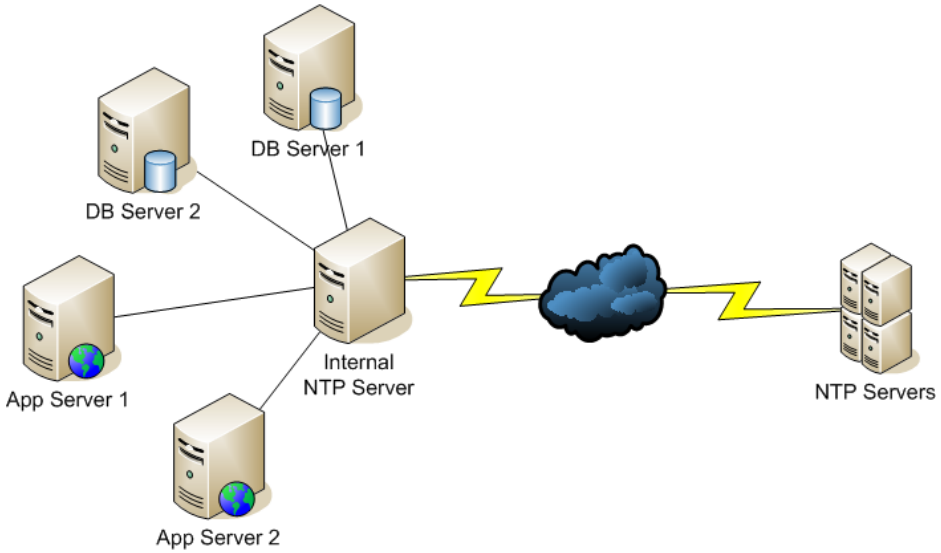



Fonte: Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

# Network time protocol

O Protocolo de Tempo para Redes ou NTP (Network Time Protocol), é um protocolo que permite a sincronização dos relógios dos dispositivos de uma rede como servidores, estações de trabalho, roteadores e outros equipamentos à partir de referências de tempo confiáveis.

O NTP usa o método de transporte UDP na porta 123.

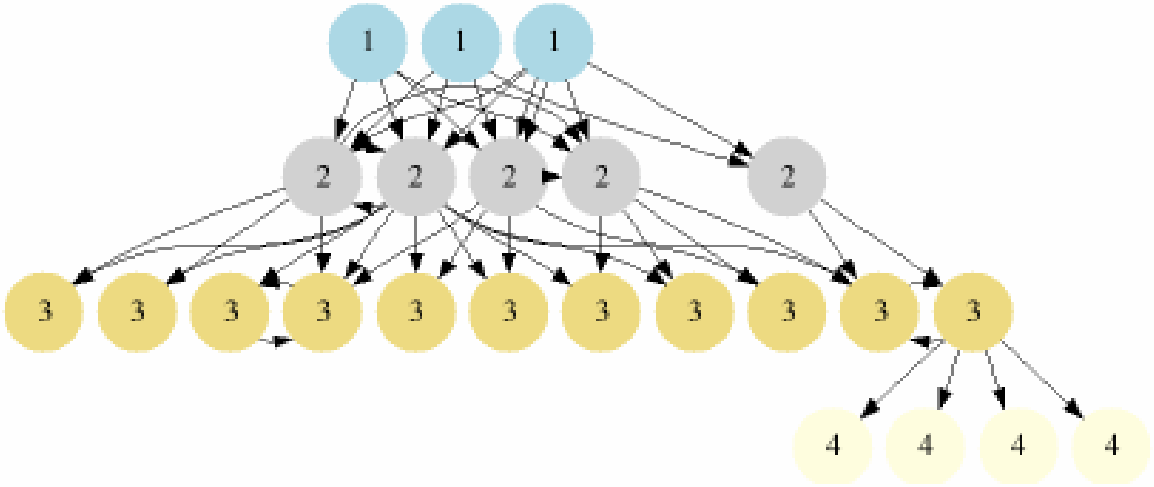



Fonte: ntp.br

# Arquitetura do NTP

Os servidores NTP formam uma topologia hierárquica, dividida em camadas ou estratos (do inglês *strata*) numerados de 0 (zero) a 16 (dezesseis).

O estrato 0 (*stratum 0*) na verdade não faz parte da rede de servidores NTP, mas representa a referência primária de tempo, que é geralmente um receptor do Sistema de Posicionamento Global (GPS) ou um relógio atômico. O estrato 16 indica que um determinado servidor está inoperante.




Fonte: ntp.br

# Arquitetura do NTP

O estrato 0, ou relógio de referência, fornece o tempo correto para o estrato 1, que por sua vez fornece o tempo para o estrato 2 e assim por diante. O NTP é então, simultaneamente, servidor (fornece o tempo) e cliente (consulta o tempo).

De forma geral, quanto mais perto da raiz, ou seja, do estrato 0, maior a exatidão do tempo. O estrato ao qual o servidor pertence é a principal métrica utilizada pelo NTP para escolher dentre vários, qual o melhor servidor para fornecer o tempo.

Por outro lado, normalmente as diferenças de exatidão entre os estratos não são expressivas e há, no momento de escolher-se um conjunto de servidores de tempo para configurar um determinado cliente, outros fatores a se considerar, como por exemplo a carga à qual os servidores estão submetidos, e o atraso de rede.

O estrato de um servidor não é uma característica estática. Se houver perda de conexão com as fontes de tempo, ou outros fatores que modifiquem a topologia da rede, o estrato pode variar.

Fonte: ntp.br

# Funcionamento do NTP

O protocolo NTP possui diversos componentes que colaboram para:

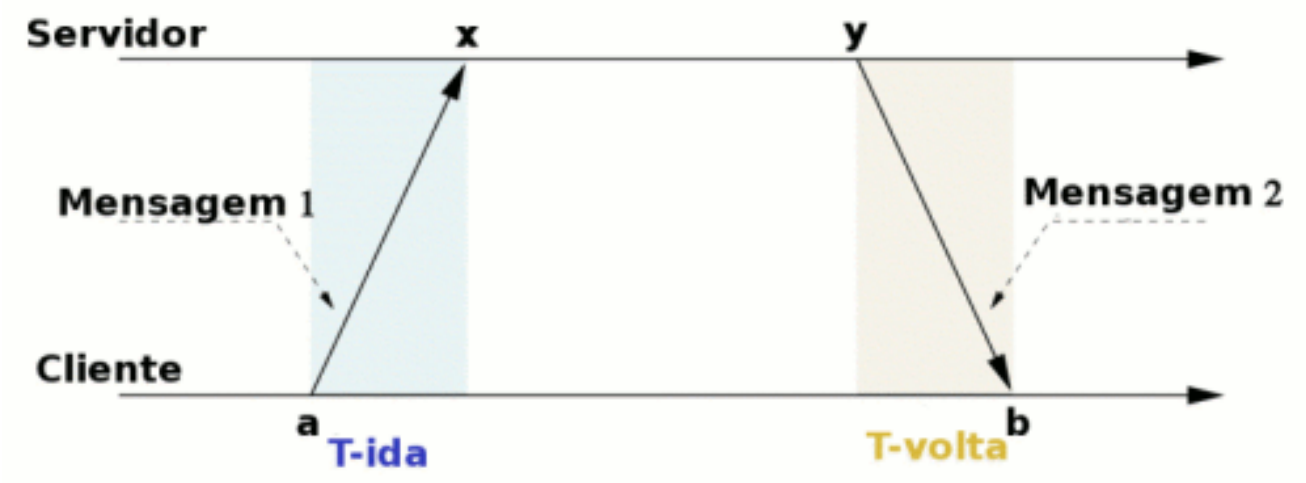
- Obter, à partir de diversas amostras, informações de tempo de um determinado servidor, como o deslocamento, dispersão e variação;
- Discernir, dentre um conjunto de servidores, quais fornecem o tempo correto e quais estão mentindo;
- Escolher, dentre os servidores que fornecem o tempo correto, qual é a melhor referência;
- Disciplinar o relógio local, descobrindo seus principais parâmetros de funcionamento, como precisão, estabilidade e escorregamento e ajustando-o de forma contínua e gradual, mesmo na ausência temporária de referências de tempo confiáveis, para que tenha e melhor exatidão possível;
- Garantir a monotonicidade do tempo;
- Identificar, à partir de métodos criptográficos, servidores de tempo conhecidos e confiáveis, evitando possíveis ataques;
- Formar, em conjunto com outros servidores NTP, uma topologia simples, confiável, robusta e escalável para a sincronização de tempo.

Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento

As mensagens do NTP são baseadas no protocolo UDP, que é um protocolo não confiável e não orientado à conexão.

A troca de mensagens entre cliente e servidor permite que o cliente descubra qual seu deslocamento (*offset*) em relação ao servidor, ou seja, o quanto seu relógio local difere do relógio do servidor.

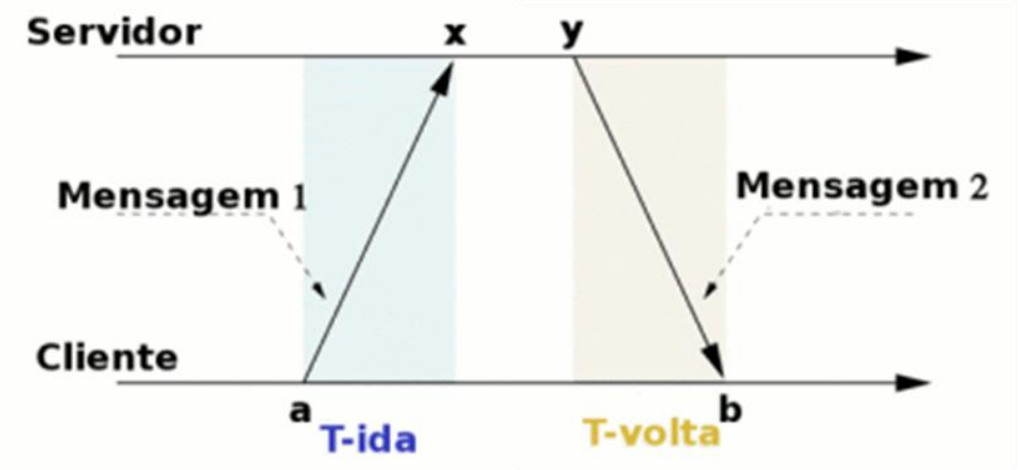



Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento

Considerando que servidor e cliente possuem relógios não sincronizados, a troca de mensagens seria da seguinte forma:

- O **Cliente** lê seu relógio, que fornece o tempo **a**.
- O **Cliente** envia a **Mensagem 1** com a informação de tempo **a** para o servidor.
- O **Servidor** recebe a **Mensagem 1** e nesse instante lê seu relógio, que fornece o instante **x**. O **Servidor** mantém **a** e **x** em variáveis.
- O **Servidor** após algum tempo lê novamente seu relógio, que fornece o instante **y**.
- O **Servidor** envia a **Mensagem 2** com **a**, **x** e **y** para o cliente.
- O **Cliente** recebe a **Mensagem 2** e nesse instante lê seu relógio, que fornece o instante **b**.



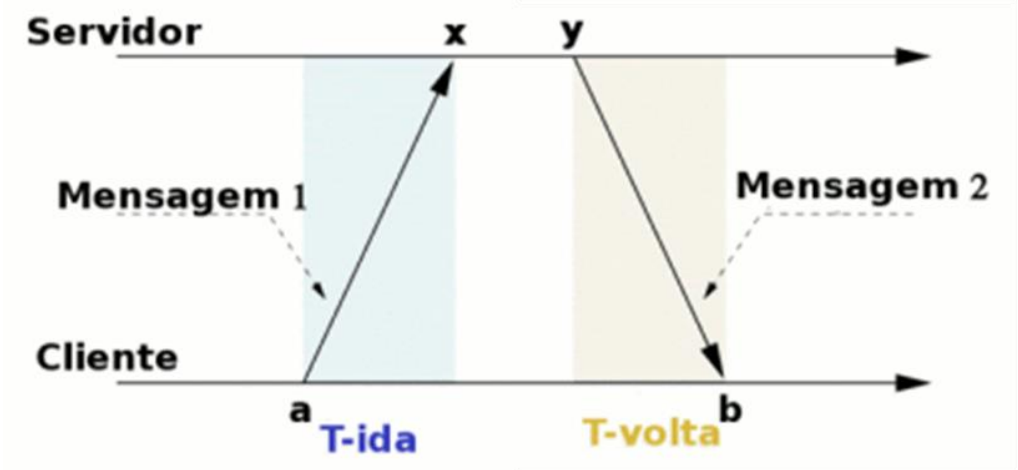

Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento

Ao receber a **Mensagem 2**, o **Cliente** passa a conhecer os instantes **a**, **x**, **y** e **b**. Mas **a** e **b** estão numa escala de tempo, enquanto **x** e **y** em outra. O valor do incremento dessas escalas é o mesmo, mas os relógios não estão sincronizados.

Não é possível, então, calcular o tempo que a **Mensagem 1** levou para ser transmitida (**T-ida**), nem o tempo que a **Mensagem 2** gastou na rede (**T-volta**).

Contudo, o **tempo total** de ida e volta, ou **atraso** (também conhecido por ***Round Trip Time*** ou ***RTT***) que é a soma **T-ida + T-volta** pode ser calculado como:




**atraso (delay) = (b-a)-(y-x)**

Considerando-se que o tempo de ida é igual ao tempo de volta, pode-se calcular o deslocamento entre o servidor e o relógio local como como:

**deslocamento (offset) = x - (a + atraso/2)**

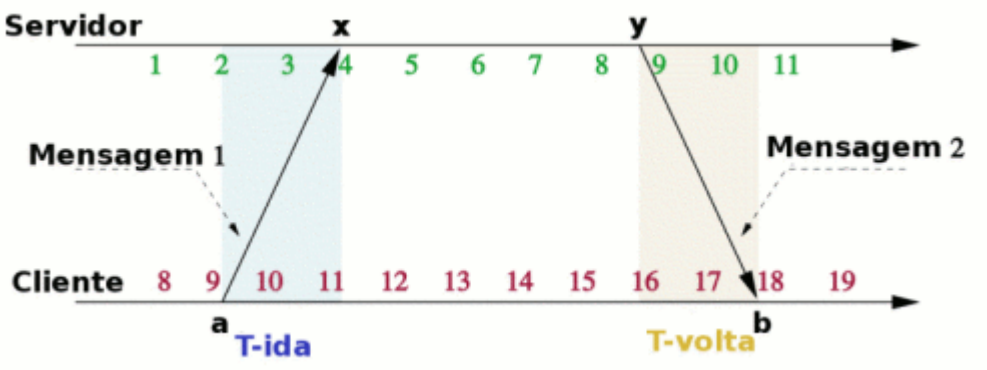
**deslocamento (offset) = (x-a+y-b)/2.**

Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento Exemplo

Para facilitar a compreensão, considere-se o seguinte exemplo numérico.

- O **Cliente** lê o relógio: **a=9**.
- O **Cliente** envia a **Mensagem 1 (a=9)**.
- O **Servidor** recebe a **Mensagem 1 (a=9)** e lê seu relógio: **x=4**.
- O **Servidor** algum tempo depois lê seu relógio novamente: **y=9**.
- O **Servidor** envia a **Mensagem 2 (a=9, x=4, y=9)**.
- O **Cliente** recebe a **Mensagem 2 (a=9, x=4, y=9)** e lê seu relógio: **b=18**.



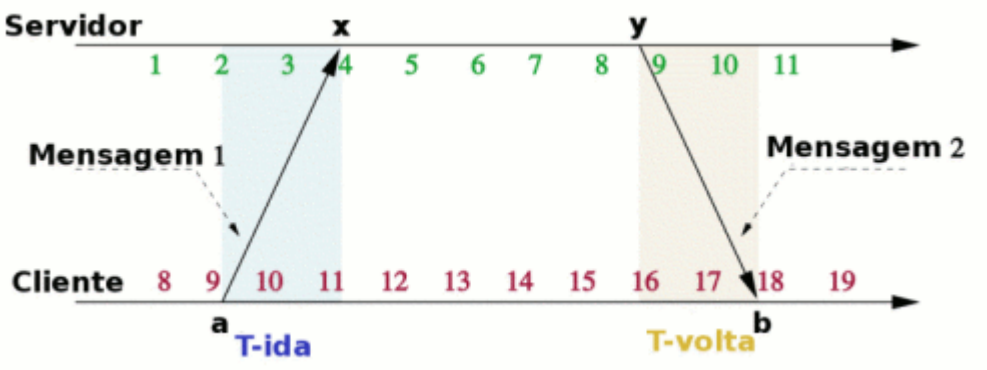

Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento Exemplo

Ao olhar a figura, pode-se observar que **T-ida=2** e **T-volta=2**. Contudo, nem o **Cliente** nem o **Servidor** têm essa visão. O **Servidor** ao final da troca de mensagens descarta todas as informações sobre a mesma. O **Cliente** conhece as variáveis **a=9**, **x=4**, **y=9** e **b=18**, mas à partir delas é impossível calcular **T-ida** ou **T-volta**. Contudo, é possível calcular o atraso e o deslocamento:

**atraso = (b-a)-(y-x) = (18-9)-(9-4) = 9 - 5 = 4**

**deslocamento = (x-a+y-b)/2 = (4-9+9-18)/2 = -14/2 = -7**

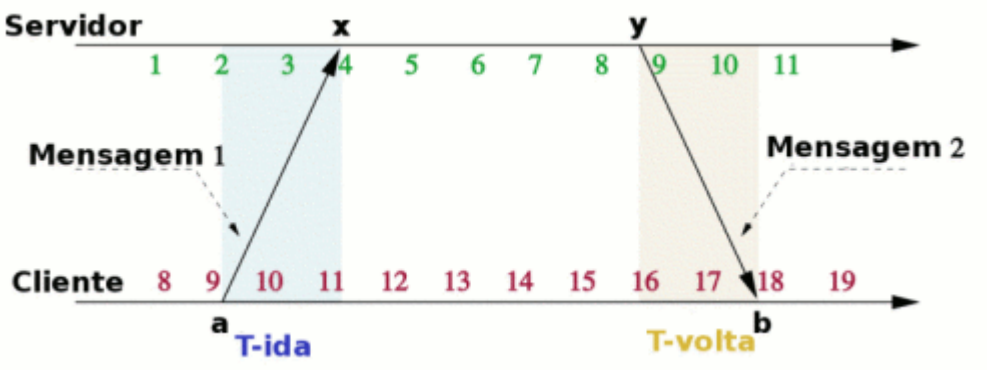



Fonte: ntp.br

# Troca de mensagens e cálculo do deslocamento Exemplo

Um deslocamento de **-7** significa que o relógio local do **Cliente** deve ser **atrasado 7 unidades de tempo** para se igualar ao do **Servidor**.

No entanto, este cálculo é mais aproximado do que exato, pois existem atrasos estocásticos nas redes devido às filas dos roteadores e switchs. Numa WAN ou na Internet enlaces de diferentes velocidades e roteamento assimétrico, além de outros fatores, também causam diferenças entre **T-ida** e **T-volta**.




Fonte: ntp.br

# Para saber mais...

… leia o Capítulo 6 do livro Sistemas Distribuídos: Princípios e Paradigmas, de Andrew S. Tanenbaum e Maarten Van Steen.

… acesse o site ntp.br.

FIM