# Sistemas Distribuídos

Os sistemas distribuídos há mais de uma década deixaram de ser uma promessa e hoje são a realidade tanto no meio acadêmico quanto no comércio e na indústria.

Uma coleção de computadores independentes que aparecem para os usuários como sendo um único computador.

Sistema computacional com diversas partes

- Diferentes programas executando em diferentes computadores
- Partes interagem para oferecer funcionalidades

# Introdução

O que é um Sistema Distribuído?

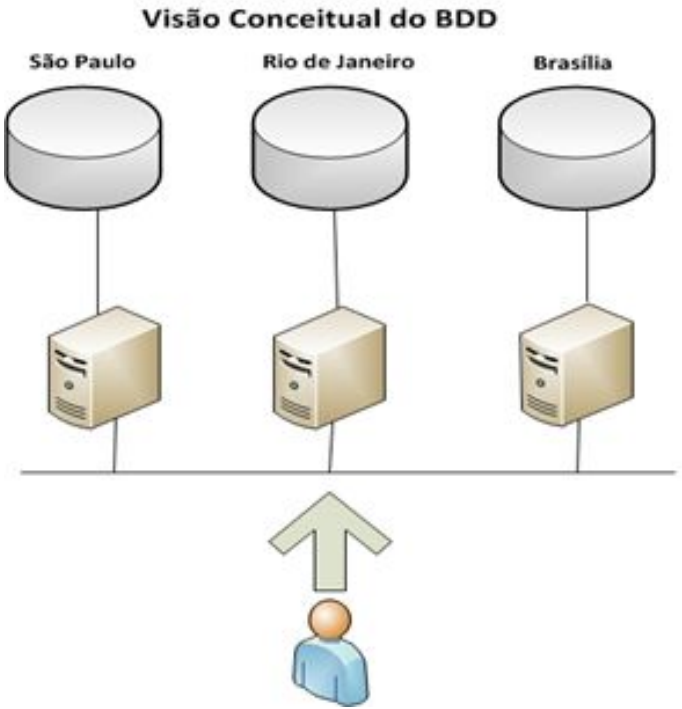
Define se um sistema distribuído aquele que se apresenta aos seus usuários como um sistema centralizado, mas que na verdade funciona em diversos CPU independentes.

Cujo não deve ter pontos críticos de falha, ou seja, se o componente quebrar isso não deve fazer com que o sistema como um todo falhe.

Essa característica de estabilidade é uma de suas principais vantagens em relação ao sistema centralizados de hardware e software.

Temos Hardware como máquina independente e autônoma em software como sistema que se apresenta ao usuário como uma única máquina

# Introdução


Visão Conceitual do BDD
São Paulo
Rio de Janeiro
Brasília


Visão do Usuário
São Paulo
Brasília
Rio de Janeiro

# Sistemas Distribuídos - Introdução

Característica importante

Lembrem-se que um sistema distribuído aquele que se apresenta aos seus usuários como um sistema centralizado, mas que na verdade funciona em diversos CPU independentes.

# Sistemas Distribuídos - Introdução

Um ponto muito importante desses sistemas distribuídos são as suas aplicações.

- Há vários tipos de aplicações que podem ser desenvolvidas utilizando os conceitos de sistemas distribuídos.
- Grande parte das aplicações web são exemplo sistemas distribuídos como por exemplo os sites de comércio eletrônico.

# Sistemas Distribuídos - Introdução

- Além deles podemos citar os seguintes exemplos mensagens instantâneas como Skype WhatsApp ou Telegram; e
- Compartilhamento de arquivos como por exemplo Netflix e Spotify; e
- Computação colaborativa como por exemplo o Waze, onde nós temos um computador coordenador que recebe as informações dos computadores escravos, e com isso consegue criar uma rede de informação; e
- Além, do bitcoin que é uma ferramenta hoje de criptomoeda que também utiliza o ambiente de sistemas distribuídos para realização de transações eletrônicas.

# Sistemas Distribuídos - Introdução

Todas as aplicações têm características em comum, tais características são muito específicas.

Veremos a seguir em detalhes cada uma delas.

# Sistemas Distribuídos - Introdução

## Vantagens

- Maior funcionalidade: Às vezes não é possível ser centralizado
- Maior escalabilidade: Atender a uma demanda muita alta
- Maior robustez: Operar mediante a falhas
- Maior transparência:  Esconder (do usuário) aspectos irrelevantes
- O compartilhamento de dados e dispositivo: são as bases de dados comuns e o acesso compartilhado a periféricos
- A confiabilidade si um nó falha, os demais podem continuar operando a velocidade da computação.

# Sistemas Distribuídos - Introdução

## Desafio: Escalabilidade

- Permanecer efetivo mesmo quando há um aumento significativo no número de recursos e usuários:
  - Controlar o custo dos recursos
  - Controlar a perda de desempenho
  - Prevenir que os recursos de software acabem
    - (ex., endereços IP)
  - Evitar gargalos de desempenho

# Sistemas Distribuídos - Introdução

## Desafio: Tratamento de Falhas

- Falha parcial
  - Detecção (crash de máquina ou atraso de rede?)
  - Mascaramento (retransmissão, replicação)
  - Cliente desiste após número pré-determinado de tentativas, timeout etc
  - Recuperação (checkpoint e rollback recovery)
  - Redundância

# Sistemas Distribuídos - Introdução

## Desafio: Concorrência

- É um problema quando dois ou mais usuários acessam o mesmo recurso ao mesmo tempo " Threads "
- Mecanismos de exclusão mútua (ex. semáforos)

# Sistemas Distribuídos - Introdução

Por que ter?

Para que?

Como funcionam?

# Sistemas Distribuídos - Introdução

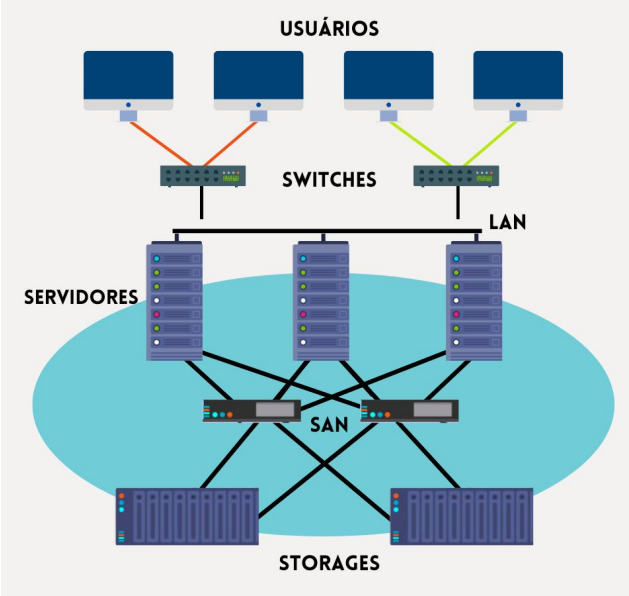
## Pra que?

Aproveitar recursos de software e hardware distribuídos

Disponibilidade

Tolerância a falha

Melhora do desempenho



# Sistemas Distribuídos - Introdução

## Para que ter?

- Escalabilidade: atendimento à demanda
- Desempenho: múltiplos processadores
- Segurança
- Disponibilidade: replicação; tolerância a falha

# Sistemas Distribuídos - Introdução

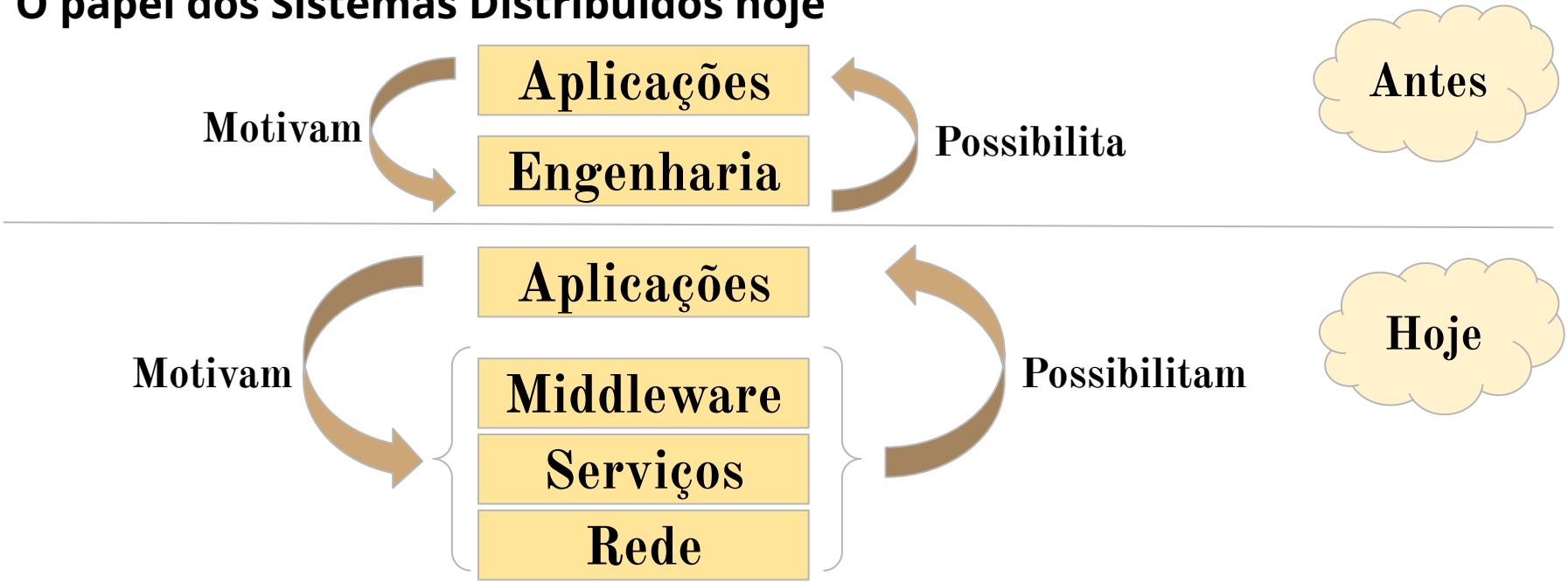
## Como funcionam?

- Suporte de comunicação
  - Redes velozes e confiáveis

- Suporte de software
  - Sistemas operacionais de rede
  - Sistemas operacionais distribuídos
  - Middleware

# Sistemas Distribuídos - Introdução

**O papel dos Sistemas Distribuídos hoje**



# Sistemas Distribuídos - Introdução

**Características Específicas:**

- Troca de Mensagens
- Falhas
- Sincronismo
- Segurança
- Heterogeneidade
- Desempenho
- Custo

- Distribuição Geográfica
  Compartilhamento de Recursos
  Capacidade de Expansão
- Disponibilidade
- Concorrência
- Transparência

# Sistemas Distribuídos - Introdução

Requisitos Iniciais

- **Dados de entrada:** Cidade + Data de início + data de fim
- **Dados de saída:** Transp. Aéreo + Transp. Terrestre + Acomodação

# Sistemas Distribuídos - Características

**Característica de troca de mensagens**

- Toda a comunicação é baseada em troca de mensagens.
- Quando se projeta um sistema distribuído deve-se preocupar com a troca de mensagens já que a comunicação ocorrerá dessa forma.
- Essa característica é um das responsáveis pelo tempo de execução de uma aplicação, sendo que o tempo de execução é a soma do tempo gasto em processamento e do tempo gasto em comunicação.
- Quanto melhor e mais eficiente for a troca de mensagens melhor será o tempo de execução utilizando troca de mensagens.
- A comunicação fica sujeita a um conjunto de fatores que pode afetar a sua confiabilidade como os protocolos e a interligação que existe entre as diversas redes.

# Sistemas Distribuídos - Características

## Troca de Mensagem

A troca de mensagens é um dos principais fatores que afetam o desempenho do sistema distribuído pois nesse tipo de ambiente tudo é baseado em comunicação.

# Sistemas Distribuídos - Características

## Sincronismo e Falhas

- Quando estamos em um sistema centralizado em um único computador o sistema operacional desse computador prevê recursos para efetuar a sincronização entre os processos.
- O principal recurso existente é o semáforo.
  - Ele consegue sincronizar a execução dos processos e o acesso aos recursos compartilhados.
- Porém em um sistema distribuído não há o recurso de semáforo disponível já que as informações estão divididas entre os diversos computadores.
- Esse tipo de ambiente para efetuar a sincronização dos processos é necessário que haja um algoritmo que efetue Esta sincronização.
  - O problema desses algoritmos é que eles necessitarão que mensagens sejam trocadas e isso aumentará o tempo de execução da aplicação.

# Sistemas Distribuídos - Características

**Sincronismo e Falhas**

- Devemos nos preocupar com a questão de sincronização para evitar que o sistema entre um estado inconsistente ou de falha.
- Quando estamos desenvolvendo um ambiente distribuído devemos nos preocupar com as falhas que podem ocorrer, já que em um ambiente centralizado há apenas um computador em um ambiente distribuído há n computadores.
    - Isso aumenta bastante a probabilidade de falhas.
    - Porém como estamos em um ambiente distribuído as falhas individuais não podem afetar o sistema como um todo.

# Sistemas Distribuídos - Características

## Sincronismo e Falhas

- Porém como estamos em um ambiente distribuído as falhas individuais não podem afetar o sistema como um todo.
- Para isso precisamos ter um algoritmo com:
  - Detecção e recuperação de falhas
  - Réplicas; e
  - Redundâncias.
  -
- Os principais fatores que levam às falhas são:
  - Elementos de interligação ou de comunicação
  - Interferências
  - Cabeamento mal estruturado; e
  - Problemas naturais
  - Falhas de software; e
  - Erros de programação; e
  - Falhas físicas nos equipamentos

# Sistemas Distribuídos - Características

## Sincronismo e Falhas

- Podemos tratar os problemas relacionados à sincronização e falha de duas maneiras distintas:
  - Utilizando a infra estrutura, como por exemplo, o uso de um balanceador entre a camada de apresentação e a camada de lógica
  - Lugar utilizando lógica de programação
    - Retornando exceções e logando os dados para tratativas e ações de acordo com a relevância do erro
- Importante destacar que dentro de um cenário distribuído muitas vezes não temos o domínio de uma aplicação.
  - Isso significa que caso um erro ocorra no lado do parceiro seremos avisados de alguma forma para que possamos analisar as alternativas.

# Sistemas Distribuídos - Características

## Segurança

- A questão da segurança é um fator que afeta bastante sistemas distribuídos.
- Além de ser o ponto que as pessoas sempre questionam e se demonstram preocupados nesse tipo de ambiente por ser distribuído temos a vulnerabilidade das redes que podem sofrer com observações de mensagens e ataques de usuários mal intencionados.
- Uma forma de minimizar esse fator, é fazer com que o usuário se identifique através de uma autenticação.
    - 1.Os usuários válidos recebem uma credencial para utilizar os recursos em um ambiente, que cria uma seção para esse usuário
    - 2. A exceção fica válida com o usuário no sistema enquanto ele estiver logado.
- 
- Outro ponto relacionado à segurança é que podemos ter sistemas heterogêneos e com políticas diferentes de segurança.
    - Por exemplo um computador em uma instituição Y tem uma política de segurança. Já um computador de instituição X possui outra política de segurança.

# Sistemas Distribuídos - Características

## Segurança

- Segurança tem três componentes:
  - Confidencialidade (proteção contra abertura para indivíduos não autorizados)
  - Integridade (proteção contra corrupção)
  - Disponibilidade (proteção contra interferência nos meios para acessar recursos)

- Dois difíceis desafios de segurança:
  - Ataques de Denial of Service (DOS)
  - Segurança de código móvel: pode acessar recursos locais?

# Sistemas Distribuídos - Características

## Heterogeneidade

- Heterogeneidade significa a capacidade de cada sistema ser diferente e diferente em sua arquitetura:
  - Sistema de comunicação; e
  - Sistema operacional; e
  - Infraestrutura; e
  - Dentre outras coisas...
  -
- Em um ambiente distribuído devemos nos preocupar em:
  - Variedade de arquiteturas a computadores que
    - são de 32 bits e outros que são de 64 bits.
    - Isso impacta no tamanho dos dados armazenados.
    - Arquiteturas vetoriais RISC CISC
      - Altera a forma como as instruções são executadas

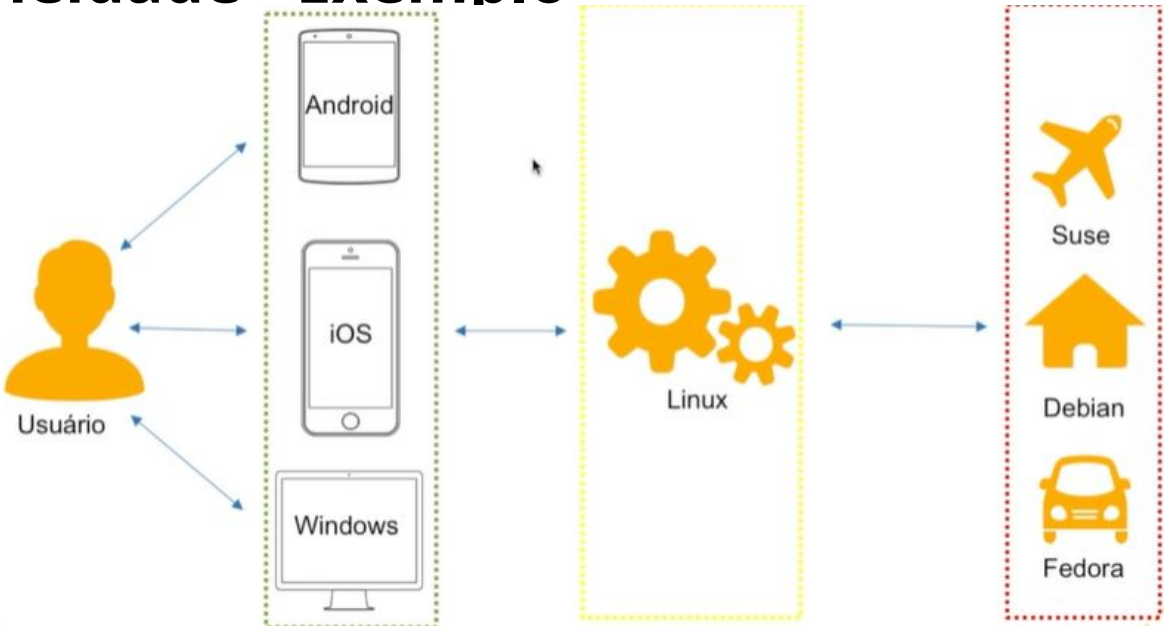
# Sistemas Distribuídos - Características

## Heterogeneidade

- Em um ambiente distribuído devemos nos preocupar em:
  - Variedade de sistemas operacionais.
    - Cada sistema operacional trabalha de uma forma diferente
    - basicamente eles possuem recursos similares, mas eles são diferenciados e utilizados de forma diferente.
    - Devemos nos preocupar com isso pois há aplicações que são desenvolvidas para um tipo específico de sistema operacional.
      - Caso isso ocorra precisa ter uma versão da aplicação para cada sistema operacional.
    - OBS: não existe um melhor sistema operacional ou aplicação.
      - Existe aquele que melhor se adequa às necessidades do consumidor ou do cliente;
      - Aquele que detém o melhor desempenho dentro de um certo cenário.

# Sistemas Distribuídos - Características

## Heterogeneidade - Exemplo



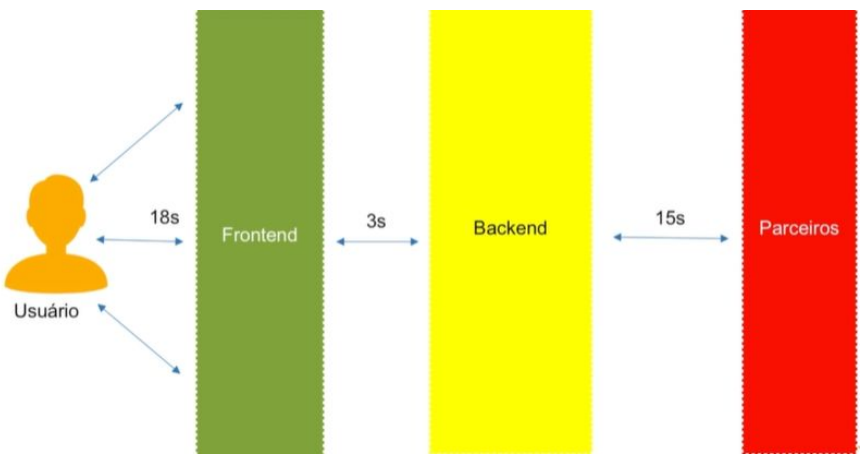
# Sistemas Distribuídos - Características

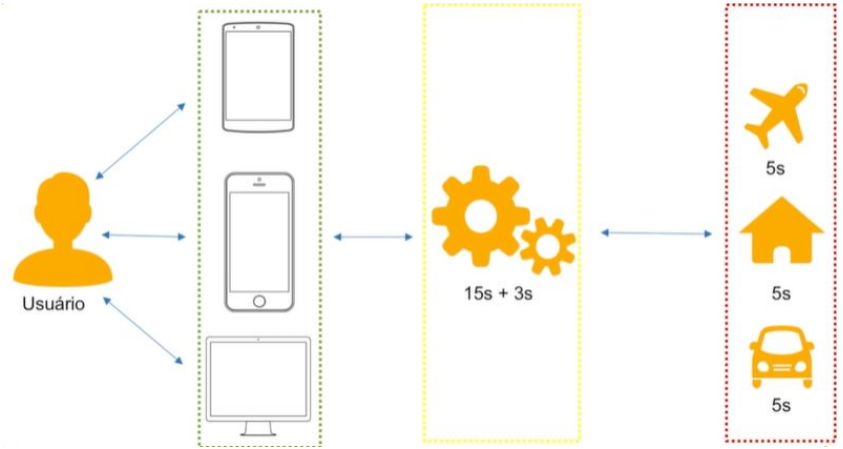
## Desempenho

- O desempenho é um dos critérios diferenciais de um sistema distribuído com relação a um sistema centralizado no ambiente distribuído.
  - Podemos desenvolver aplicações que utilizam diversos computadores para o processamento.
    - Isso faz com que o tempo de resposta de uma solicitação seja minimizado.
    - Em contrapartida há um custo para a utilização dos computadores nesse ambiente, a comunicação
      - É necessário que eles troquem mensagens para execução de tarefas, para troca de dados e para utilização das informações.
  - O desempenho de um sistema distribuído é devido à quantidade de computadores que participam da execução das aplicações, é melhor do que o desempenho do sistema centralizado.
  - Em ambiente de processamento de alto desempenho.
    - Essa característica é essencialmente fundamental, sendo o objetivo principal na modelagem de aplicações
    - Com essa finalidade em nosso cenário de caso de uso podemos analisar como será o tempo de resposta da nossa aplicação.

# Sistemas Distribuídos - Características

## Desempenho - Exemplo



# Sistemas Distribuídos - Características

## Custo

- Podemos obter um sistema distribuído com a mesma quantidade de processadores de um sistema centralizado porém com custo muito menor.
- Para ilustrar vamos dividir dois cenários.
  - Cenário 1: um computador multi processado com 32 núcleos de processamento.
  - Cenário 2:  32 computadores com um núcleo de processamento cada um.
  - OBS: Em ambos cenários há 32 núcleos de processamento, porém o custo de um computador com 32 núcleos de processamento, é muito maior do que o custo de 32 computadores com um núcleo de processamento, ou seja, com um custo muito menor.
  - Conseguimos criar um sistema distribuído que possua a mesma capacidade de processamento.

# Sistemas Distribuídos - Características

## Custo

- A questão é de que a execução de uma aplicação no cenário 1 é mais rápida do que a execução no cenário
  - Isso ocorre porque no cenário 1 os processadores compartilham a memória e não há necessidade de troca de mensagens.
  - Porém no cenário 1 há um limite físico na quantidade de núcleos de processamento.
  - Caso haja a necessidade de mais poder de processamento, esse computador não conseguirá atender a necessidades.
  -
- Caso ocorra a necessidade de aumentar o poder de processamento no cenário 2
  - Basta adicionar mais computadores no ambiente.
    - Isso torna o custo muito mais atrativo; e
    - Possibilita a redundância; e
    - Replicação das informações

# Sistemas Distribuídos - Características

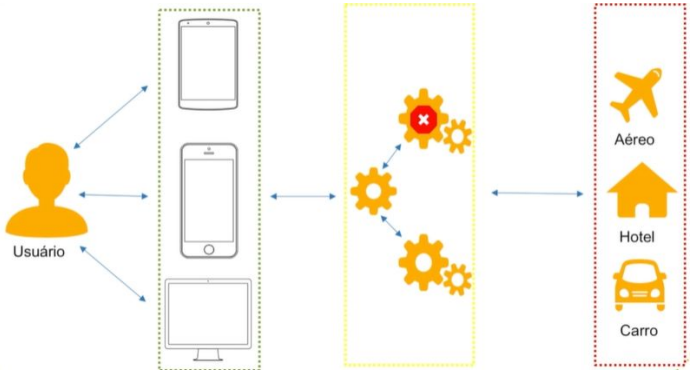
**Distribuição Geográfica (**Compartilhamento de Recursos
Capacidade de Expansão)

- No sistema distribuído, não há necessidade dos computadores estarem próximos geograficamente.
- O ambiente pode ser composto por computadores em qualquer parte do planeta.
- Basta que ter uma rede que possibilite a comunicação entre eles como por exemplo a internet.
- Compartilhamento de recursos/periféricos
    - O compartilhamento de periféricos ocorre com muita frequência em ambientes empresariais no quais periféricos de alto custo como por exemplo as impressoras a laser.
    - Compartilhamento de recursos como arquivos tabelas variáveis

# Sistemas Distribuídos - Características

## Disponibilidade

- A disponibilidade é uma característica muito importante para sistemas distribuídos que precisam estar sempre em funcionamento.
- Quando um componente falha em um SD, apenas a parte que usa este componente é afetada;
    - O componente pode ser reinicializado em outro computador



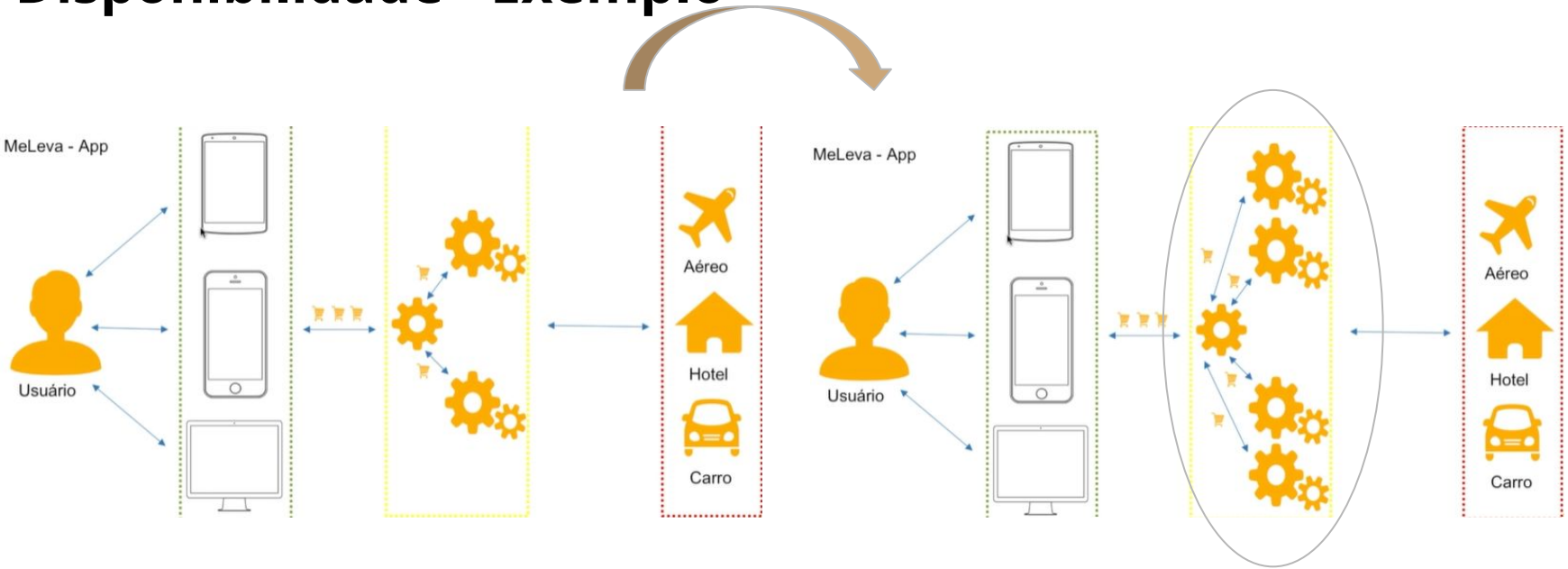
# Sistemas Distribuídos - Características

## Concorrência

- Em um ambiente distribuído a concorrência para o acesso aos recursos é mais complexa do que em um sistema centralizado, já que há mais processadores participando do ambiente.
- Isso faz com que tenhamos que nos preocupar em garantir ao processo o acesso exclusivo a um recurso compartilhado.
- Concorrência e execução paralela existem em um SD por causa de:
  - as atividades separadas de usuários; e
  - a independência de recursos;  e
  - a localização de processos em computadores separados

# Sistemas Distribuídos - Características

## Disponibilidade - Exemplo



# Sistemas Distribuídos - Características

## Transparência

- A transparência é outra característica importante
- Geralmente o sistema distribuído é transparente ao usuário.
- Porém este conceito de transparência envolve as seguintes áreas:
  - Localização: o usuário não precisa saber onde estão os recursos. Ele simplesmente os utiliza.
  - Replicação: não é necessário saber quantas cópias do recurso existem.
  - Migração: os recursos podem mudar de lugar sem a alteração dos nomes.
    - A migração pode ocorrer para ver o balanceamento de carga ou no caso de o computador falhar a concorrência.
  - Concorrência: Recurso pode ser disputado sem o conhecimento do usuário.
  - Paralelismo: várias atividades podem ocorrer simultaneamente sem o conhecimento dos usuários.

# Tipos de Sistemas Distribuídos

- **Categorias**:
  - Sistemas de computação distribuídos
  - Sistemas de informação distribuídos
  - Sistemas pervasivos

# Sistemas de Computação Distribuídos

- Utilizados para computação de alto desempenho
- Divididos em subgrupos:
  - Cluster Computing
  - Grid computing
  - Cloud computing

# Sistemas de Computação Distribuída

Os **sistemas de computação distribuída** são aqueles em que múltiplos computadores trabalham juntos para resolver problemas computacionais.

Esses sistemas compartilham recursos (como processamento, armazenamento e rede) para oferecer maior desempenho, escalabilidade e confiabilidade.

# Sistemas de Computação Distribuída

## Características Principais

**Transparência** – O sistema deve parecer um único ambiente para o usuário, mesmo sendo distribuído. Isso pode incluir:

- Transparência de acesso (o usuário não precisa saber onde os dados estão).
- Transparência de localização (os dados podem estar em qualquer máquina).
- Transparência de replicação (o sistema pode ter cópias de segurança sem que o usuário perceba).

**Escalabilidade** – O sistema pode crescer adicionando mais máquinas sem comprometer o desempenho.
**Tolerância a Falhas** – O sistema deve continuar funcionando mesmo se algumas máquinas falharem.
**Concorrência** – Múltiplos processos podem rodar ao mesmo tempo, aumentando a eficiência.
**Balanceamento de Carga** – A carga de trabalho deve ser distribuída entre os nós para evitar sobrecarga em alguns servidores.

# Sistemas de Computação Distribuída

Os sistemas de computação distribuída podem ser organizados de várias formas:

**Cliente-Servidor**

- O servidor fornece serviços e os clientes fazem requisições.
- Pode haver múltiplos servidores para diferentes funções (web, banco de dados, autenticação).
- **Exemplo:** Aplicações web, como um site de e-commerce onde um servidor processa as compras.

**Computação em Cluster**

- Vários computadores (nós) trabalham juntos como um único sistema.
- Normalmente usados para alto desempenho (HPC - High-Performance Computing).
- **Exemplo:** Supercomputadores, processamento de grandes volumes de dados (Big Data).

**Computação em Grade (Grid Computing)**

- Usa recursos computacionais geograficamente dispersos.
- Cada nó pode ter hardware e sistemas operacionais diferentes.

# Sistemas de Computação Distribuída

**Computação em Nuvem (Cloud Computing)**

- Recursos (como servidores, armazenamento e bancos de dados) são fornecidos via internet.
- Pode ser pública (AWS, Azure, Google Cloud), privada ou híbrida.
- **Exemplo:** Serviços de streaming (Netflix, Spotify), armazenamento em nuvem (Google Drive, OneDrive).

**Computação P2P (Peer-to-Peer)**

- Todos os nós podem atuar como cliente e servidor ao mesmo tempo.
- Funciona sem um servidor central.
- **Exemplo:** Torrent, redes blockchain.

# Sistemas de Computação Distribuída

## Vantagens da Computação Distribuída

**Melhor desempenho** – Trabalha de forma paralela para maior eficiência.
**Alta disponibilidade** – Se um nó falha, o sistema pode continuar funcionando.
**Escalabilidade** – Facilidade para adicionar novos nós conforme necessário.
**Redução de custos** – Pode usar máquinas comuns ao invés de supercomputadores caros.

# Sistemas de Computação Distribuída

## Exemplos de Uso da Computação Distribuída

**Google Search** – Processa bilhões de consultas usando clusters distribuídos.
**Amazon AWS** – Fornece serviços em nuvem distribuídos globalmente.
**Jogos Online** – Servidores distribuídos para jogos multiplayer (ex.: Fortnite, Minecraft).
**Simulações Científicas** – Modelagem climática, previsão de terremotos.
**Big Data e Machine Learning** – Sistemas como Hadoop e Spark processam grandes volumes de dados.

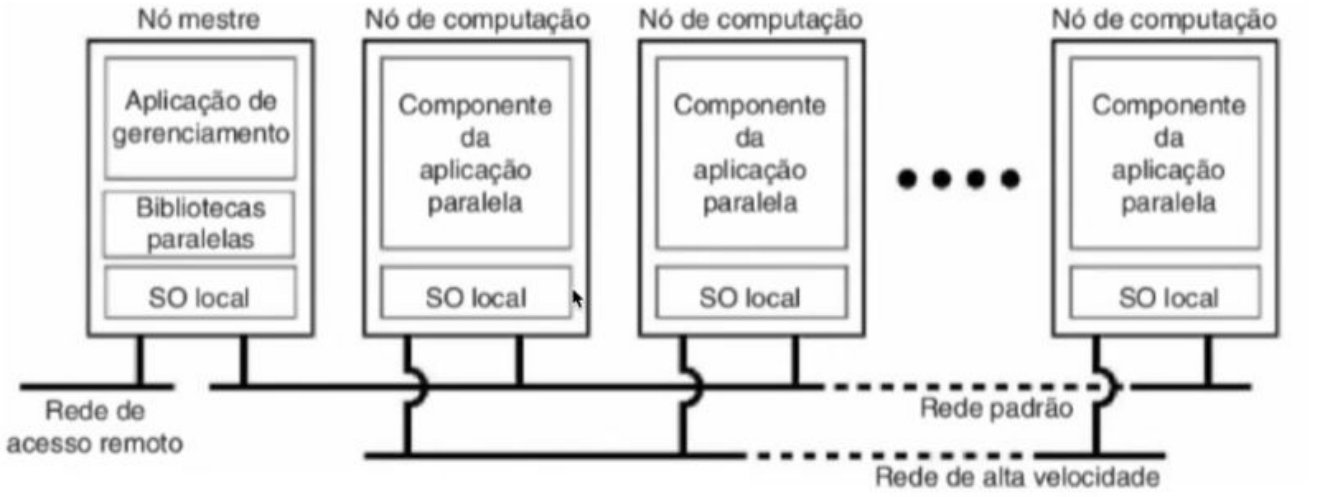
# Sistemas de Computação Distribuída

**O que podemos concluir da computação distribuída**

Os **sistemas de computação distribuída** são essenciais para aplicações modernas, desde serviços de nuvem até supercomputadores para pesquisa científica.

Eles oferecem desempenho e escalabilidade, mas também apresentam desafios como sincronização, segurança e gerenciamento de falhas.

# Cluster Computing

- Computadores similares conectados em rede de alta velocidade.
  - Muitos computadores comuns é mais barato que único supercomputador.
  - 

# Grid Computing

**Grid computing** (ou **computação em grade**) é um modelo de computação que usa **vários computadores conectados em rede**, trabalhando juntos para resolver **tarefas complexas** como se fossem **um supercomputador**

**Funcionamento:**

- Diversos **computadores independentes** (podem estar em lugares diferentes) são conectados por uma rede (geralmente a internet ou intranet).
- Cada computador da "grade" (grid) doa **tempo de processamento e/ou espaço de armazenamento**.
- Um **gerenciador de tarefas** divide o problema em partes menores e distribui entre os computadores.
- Os resultados são enviados de volta, reunidos e processados como um só.

# Grid Computing

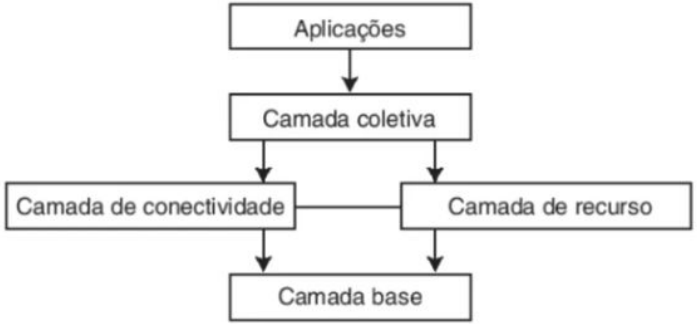
**Características**

| Descrição | Explicação |
| --- | --- |
| Distribuído | Usa recursos espalhados em diferentes locais |
| Colaborativo | Cada nó (computador) contribui com uma parte |
| Escalável | Pode crescer adicionando mais máquinas |
| Heterogêneo | Pode usar máquinas com sistemas diferentes |

# Grid Computing

- Federação de recursos de diferentes organizações
- Compartilhamento de servidores de computação armazenamento, BDs, dispositivos especiais
- Sistemas de computação em grad tem um alto grau de heterogeneidade, nenhuma premissa é adotada em relação ao Hardware, SO, redes, domínios administrativos,...
- Os recursos de diferentes organizações são reunidos para permitir a colaboração de um grupo de pessoas ou instituições.

# Grid Computing




**Coletiva**: Coordena ações entre múltiplos recursos.
Ex: balanceamento de carga, localização de serviços, colaboração entre diferentes partes da grid.

Ex: serviços de diretório, localização de recursos,

**Aplicação**: Onde os usuários interagem com a grid. São os programas e interfaces que usam os recursos da grid.

Ex: simulações científicas, análise de grandes dados (Big Data), modelagens climáticas.

**Camada base:** Também chamada de *Infraestrutura*.
É a base: conecta todos os dispositivos e fornece acesso aos recursos físicos, como CPU, memória, arquivos e sensores.

Ex: computadores, servidores, clusters, sensores.

**Conectividade**: Responsável pela **comunicação** entre os nós da grid. Define protocolos de segurança, autenticação e transferência de dados.

Ex: TCP/IP, SSL, protocolos de autenticação.

**Recursos**: Gerencia os recursos individuais de cada nó. Cuida de tarefas como:

- Descobrir recursos disponíveis
- Agendar tarefas
- Monitorar uso

NOTA

Normalmente as camadas coletiva, de conectividade e de recursos formam o cerne daquilo que podemos chamar de camada de middleware em grade.

# Grid Computing

| Camada | Função Principal |
| --- | --- |
| Base | Acesso direto aos recursos físicos |
| Conectividade | Comunicação segura entre os nós |
| Recursos | Gerenciamento de recursos individuais |
| Colaborativa | Coordenação entre vários recursos/nós |
| Aplicação | Interface de uso final (programas e usuários) |

# Grid Computing - Diferença entre Grid, Cluster e Cloud

| Característica | Grid Computing | Cloud Computing | Cluster Computing |
|---|---|---|---|
| **Conceito** | Computadores independentes trabalhando juntos pela rede | Fornecimento de recursos de TI via internet sob demanda | Conjunto de computadores interligados localmente para trabalhar como um só |
| **Localização** | Distribuído geograficamente (internet ou intranet) | Totalmente remoto (geralmente em data centers) | Local (mesmo local físico ou rede local) |
| **Conectividade** | Rede ampla (WAN, Internet) | Internet | Rede local (LAN) |
| **Gerenciamento** | Compartilhado entre organizações ou usuários | Gerenciado por provedores (ex: AWS, Azure, Google Cloud) | Centralizado e controlado localmente |

# Grid Computing - Diferença entre Grid, Cluster e Cloud

| Característica | Grid Computing | Cloud Computing | Cluster Computing |
|---|---|---|---|
| **Escalabilidade** | Alta (adiciona-se mais máquinas à grade facilmente) | Altíssima (on-demand e automática) | Limitada à infraestrutura física |
| **Custo** | Geralmente gratuito ou baseado em contribuição voluntária | Pago conforme uso | Alto custo inicial (compra e manutenção dos servidores) |
| **Tolerância a falhas** | Média – depende da estrutura da grade | Alta – com backups e redundância automática | Média – pode haver falhas se um nó principal cair |

# Grid Computing - Diferença entre Grid, Cluster e Cloud

| Característica | Grid Computing | Cloud Computing | Cluster Computing |
|---|---|---|---|
| **Integração entre nós** | Heterogênea (diferentes sistemas operacionais e hardware) | Transparente ao usuário | Homogênea (geralmente máquinas similares) |
| **Foco principal** | Compartilhar poder de processamento entre locais distintos | Flexibilidade, escalabilidade e serviços sob demanda | Desempenho alto e sincronização em tarefas específicas |

## Resumo do quadro comparativo

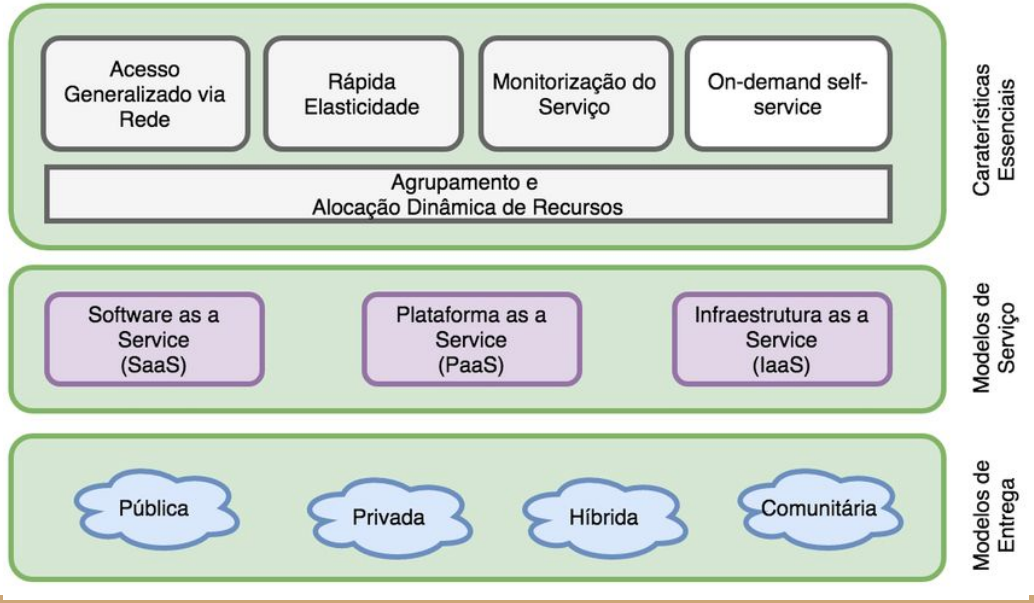
**Grid Computing** = Cooperação de computadores independentes espalhados.

**Cloud Computing** = Serviços e infraestrutura remota, sob demanda.

**Cluster Computing** = Conjunto de máquinas locais trabalhando como uma só.

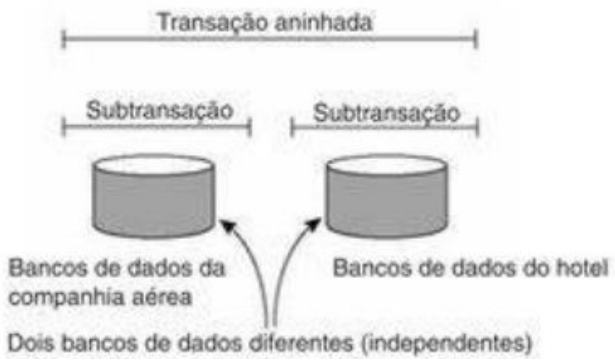
# Cloud Computing

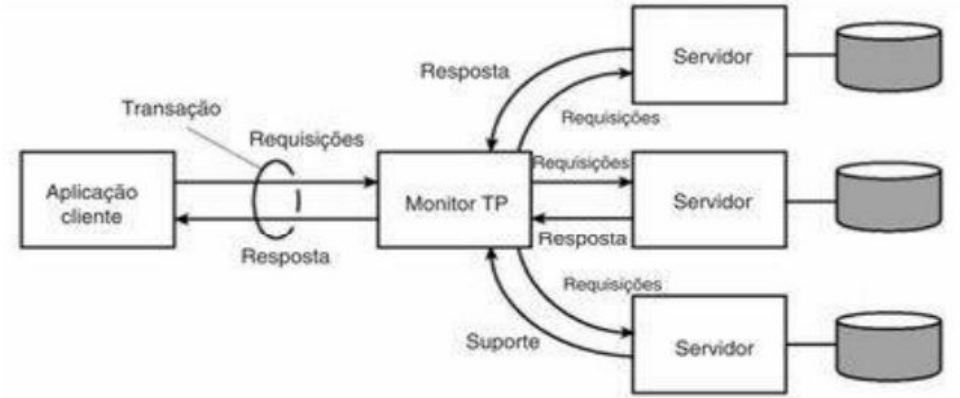
- Recursos computacionais virtualizados oferecidos por provedor
  - Elasticidade: Podem aumentar / diminuir recursos sob demanda
  - pay-per-use: Usuários só pagam o que consomem



# Sistemas Informação Distribuídos - Transações Distribuídas

- **Transação**: Conjunto de operações que executa tudo ou nada (Propriedades AICD);
- **Transações distribuídas:** Sub-transações podem rodar em servidores diferentes



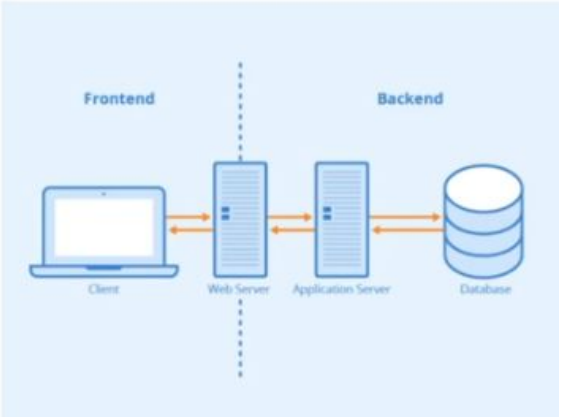



# Sistemas Informação Distribuídos - Integração de aplicações corporativas

- Aplicações remotas se comunicam para oferecer funcionalidades
  - EX: Web Services, Microservices
  -
- Diferentes formas de comunicação
  - Remote Procedure Call (RPC)
  - Remote Method Invocation (RMI)
  - Mensagens / publish-subscribe(pub-sub)

# Sistemas Informação Distribuídos

- Sistemas de informação integrados em rede
  - EX: Frontend/backend, BD, microsserviços
- Desafios de consistência dos dados e comunicaçã




- Dois tipos principais
  - Processamento de transações distribuídas
  - Integração de aplicações corporativas

# Sistemas de Informação Distribuída

**O que são Sistemas de Informação Distribuídos?**

São aplicações que usam uma infraestrutura de sistema distribuído para coletar, processar, armazenar e distribuir informações.

> Exemplos clássicos incluem sistemas de arquivos distribuídos, bancos de dados distribuídos e sistemas de recuperação de informações.

Esses sistemas são construídos sobre os princípios da computação distribuída, mas com o foco específico em **manipulação e gerenciamento de dados** de forma integrada e coesa, mesmo que os dados estejam fisicamente em locais diferentes.

# Sistemas de Informação Distribuída

Princípios Fundamentais

**Transparência**

A transparência é um dos pilares centrais e aparece em várias formas:

- **Transparência de Acesso**: o modo de acessar dados é o mesmo, independentemente de onde eles estão.
- **Transparência de Localização**: os usuários não precisam saber onde o dado está armazenado.
- **Transparência de Migração**: dados podem ser movidos sem afetar a aplicação.
- **Transparência de Replicação**: usuários não sabem se estão acessando uma cópia ou o original.
- **Transparência de Concorrência**: múltiplos usuários acessam os dados sem afetar a integridade.
- **Transparência de Falhas**: o sistema continua operando mesmo após falhas parciais.

# Sistemas de Informação Distribuída

**Componentes Clássicos de um Sistema de Informação Distribuído**

**Sistemas de Arquivos Distribuídos (DFS)**

Permitem que arquivos armazenados em diferentes servidores possam ser acessados como se estivessem em um único sistema.

- Exemplos: NFS (Network File System), AFS (Andrew File System).

- Abstrações como "montagem" e "caminhos globais" ajudam na transparência.

**Bancos de Dados Distribuídos (DDBMS)**

São sistemas de gerenciamento de banco de dados onde os dados estão fisicamente em mais de um local.

- Desafios: garantir **transações distribuídas**, **consistência dos dados**, **consultas eficientes**.

**Serviços de Diretório**

Como o LDAP, que armazenam informações sobre usuários, permissões, recursos de rede.

**Serviços Web e Sistemas Baseados em Middleware**

Como CORBA, SOAP, RESTful APIs — permitem comunicação entre diferentes sistemas distribuídos e heterogêneos.

# Sistemas de Informação Distribuída

## Segurança em Sistemas Distribuídos de Informação

A descentralização aumenta os riscos. Principais mecanismos de proteção:

- **Autenticação** (usuário é quem diz ser)
- **Autorização** (controle de acesso baseado em papéis)
- **Criptografia de dados em trânsito e em repouso**
- **Logs e auditorias**

# Sistemas de Informação Distribuída

## Transações Distribuídas e o Desafio da Confiabilidade

**Propriedades ACID em sistemas distribuídos:**

- **Atomicidade**: tudo ou nada.
- **Consistência**: o sistema permanece em estado válido.
- **Isolamento**: execuções concorrentes não interferem.
- **Durabilidade**: uma vez feito, não se desfaz.

# Sistemas de Informação Distribuída

**Escalabilidade**

Sistemas distribuídos precisam crescer sem queda significativa de desempenho. Estratégias:

- Divisão horizontal de dados (*sharding*)
- Caches distribuídos
- Balanceamento de carga
- Uso de redes P2P para sistemas de informação muito grandes (como sistemas de busca ou torrents)

# Sistemas de Informação Distribuída (Replicação e Consistência

**Replicação de Dados**

- Melhora desempenho (dados mais próximos dos usuários).
- Melhora disponibilidade (falhas em um servidor não impedem o acesso).
- Causa o desafio de **manter a consistência entre réplicas**.

**Modelos de Consistência**

- **Forte (strong)**: todas as réplicas mostram o mesmo valor após uma escrita.
- **Eventual**: após algum tempo, todas convergem para o mesmo estado.
- **Causal**: respeita a ordem causal das operações.

# Sistemas de Informação Distribuída (Replicação e Consistência)

**O que são Modelos de Consistência?**

Modelos de consistência definem **o comportamento esperado** de um sistema distribuído no que diz respeito à **visibilidade e atualização de dados replicados**. Em sistemas distribuídos, os dados geralmente são **replicados** em vários servidores por motivos de desempenho e disponibilidade. A consistência determina **quando e como essas réplicas refletem mudanças feitas por operações de escrita**.

# Sistemas de Informação Distribuída (Consistência)

## Consistência Forte (Strong Consistency)

Todos os nós veem as mesmas atualizações no mesmo instante.

> **Exemplo:** Imagine um sistema bancário. Você faz uma transferência e imediatamente em qualquer lugar do mundo o saldo é o mesmo.

**Vantagem:** previsibilidade.
**Desvantagem:** latência maior e menor disponibilidade.

# Sistemas de Informação Distribuída (Consistência)

## Consistência Causal (Causal Consistency)

Se uma operação *B* depende logicamente de uma operação *A*, então *todos os nós verão A antes de B*.

> **Exemplo:** Você posta uma foto no Instagram (A), depois alguém comenta (B). Outros usuários veem primeiro a foto, depois o comentário.

**Vantagem:** mantém a lógica das dependências.
**Desvantagem:** mais complexa que a eventual.

# Sistemas de Informação Distribuída (Consistência)

Quadro comparativo

| **Modelo** | **Ordem de Atualização** | **Disponibilidade** | **Uso Ideal** |
|---|---|---|---|
| Forte | Imediata e global | Menor | Banco, Sistemas críticos |
| Causal | Ordem lógica mantida | Moderada | Redes sociais, apps colaborativos |
| Eventual | Atualiza com o tempo | Alta | Sistemas distribuídos globais |

# Sistemas de Informação Distribuída (Consistência)

## Consistência Eventual (Eventual Consistency)

Todos os nós **eventualmente** terão os mesmos dados — **mas não imediatamente**.

> **Exemplo:** Você atualiza seu perfil em uma rede social. Alguém em outro país pode ver a versão antiga por alguns segundos antes da nova atualização aparecer.

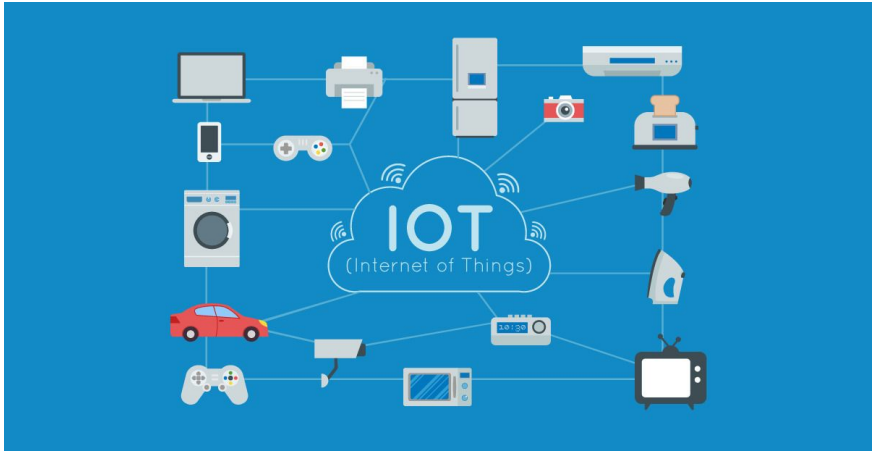
**Vantagem:** alta disponibilidade e baixo custo.
**Desvantagem:** não há garantia de ordem ou atualizações imediatas.

# Sistemas pervasivos

- Dispositivos móveis geralmente pequenos,

  instáveis, com sensores, conexão sem fio e uso de bateria

  - Formam a Internet das coisas
  -
- Divididos em:
  - Sistemas ubíquos
  - Sistemas móveis
  - Redes de sensores



# Sistemas pervasivos

Sistemas **pervasivos** são aqueles que **estão "em todo lugar"**, ou seja, **integram-se ao ambiente** de forma quase invisível e **funcionam automaticamente**, sem o usuário perceber.

### Ubíquo

- significa **"em todo lugar ao mesmo tempo"**.
- A ideia é que a tecnologia esteja **presente em todos os lugares**, **sempre disponível**.
- Exemplo: Um smartphone que você leva para todo lado é ubíquo — está sempre com você.

### **Pervasivo (**espalha ou  penetra)

- Refere-se a sistemas que **se integram ao ambiente** de forma **invisível**, **silenciosa** e **automática**.
- Exemplo: Sensores de movimento escondidos que controlam luzes sem que você veja ou perceba.

# Sistemas pervasivos

## Quadro comparativo

| Conceito | Ubíquo | Pervasivo |
|---|---|---|
| Presença | Está em todos os lugares | Está embutido no ambiente |
| Visibilidade | Pode ser visível | Geralmente é invisível ou discreto |
| Acesso | Sempre acessível | Sempre funcionando, quase sem interação |
| Foco | Disponibilidade | Integração e automação |

# Desafios - Exemplos

**Desafio:** Permitir a comunicação entre cliente e servidor

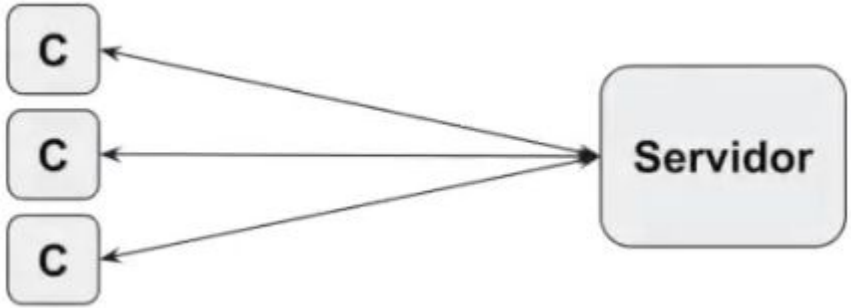
Como estabelecer a comunicação entre os clientes e o servidor

**Troca de Mensagem:** Chamada remota de procedimentos (RPC)

# Desafios - Exemplos

**Desafio:** Permitir o acesso de múltiplos usuários

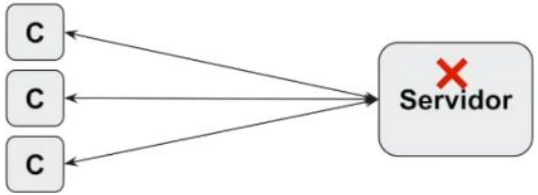
Multithreading, programação assíncrona



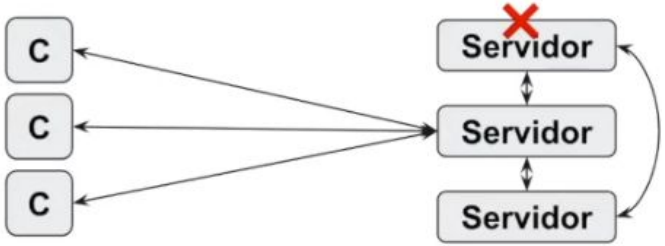
# Desafios - Exemplos

**Desafio**: Se o servidor falhar, o sistema fica indisponível

Se o disco falhar permanentemente , todos os dados serão perdidos



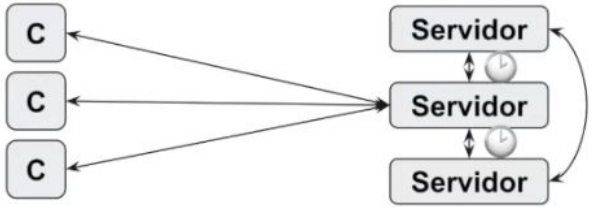

**Solução**: Replicação (réplicas secundárias ou backup)



# Desafios - Exemplos

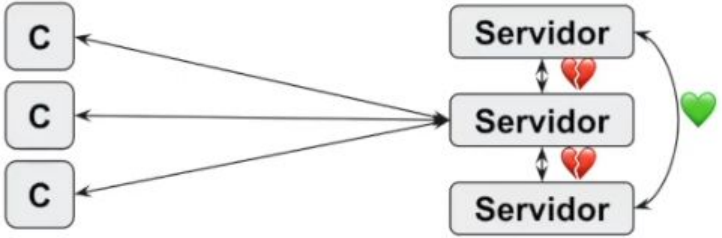
**Desafio**: Atrasos na comunicação entre servidores

A réplica primária falhou ou a conexão está lenta



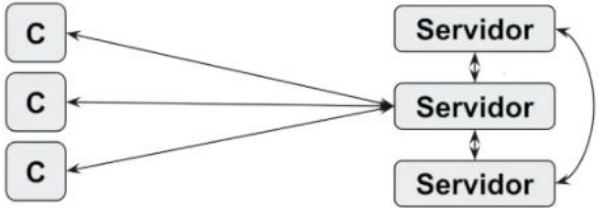

Solução: detector de falhas

Heartbeat periódico. Pode haver engano se houver atraso atípico

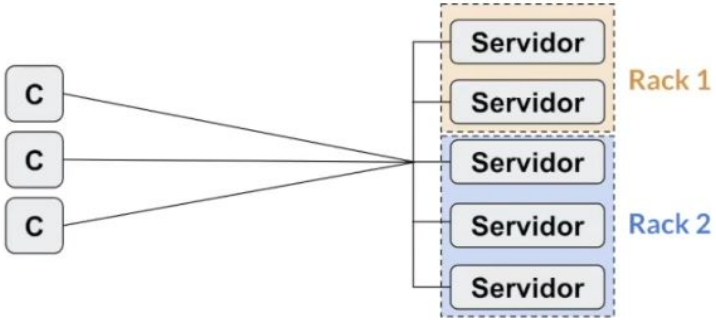


# Desafios - Exemplos

**Desafio**: Seu sistema popularizou e ficou sobrecarregado



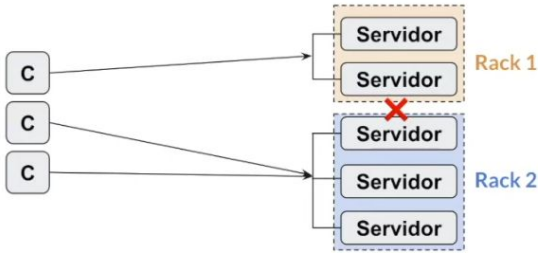

Solução: replicação ativa e balanceamento de carga



# Desafios - Exemplos

**Desafio**: Falha na comunicação entre grupos de servidores

**Split brain** (cérebro partido)**:** Pode gerar inconsistência nos dados:




Por exemplo, um problema na transação bancária

Solução:

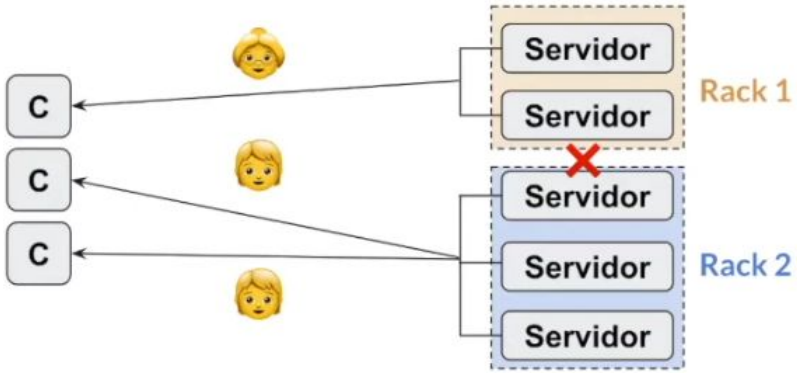
Só aplica ações se a maioria de servidores confirmarem

Apenas quem acessa o Rack 2 teria os serviços disponíveis

Precisa de 2f + 1 servidores para tolerar f falhas

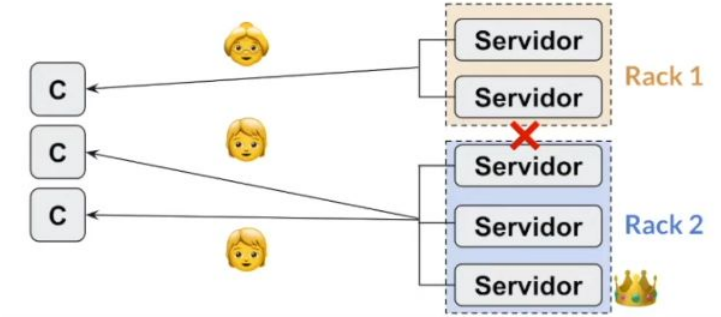
# Desafios - Exemplos

**Desafio**: Parte dos cliente pode ler dados desatualizados



# Desafios - Exemplos

**Desafio**: Parte dos cliente pode ler dados desatualizados




Solução: Eleição de um líder

Controlar e coordenar replicação nos servidores

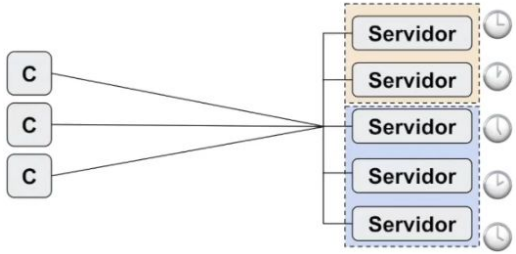
Só retorna valor ao cliente se o **líder** garantir que está consistente na maioria dos servidores

# Desafios - Exemplos

**Desafio**: Os relógios dos servidores marcam horas diferentes

Como ordenar os eventos que chegam nos diferentes servidores?

Latência da rede pode dificultar a sincronização dos relógios




Por exemplo, postagem e comentários, inversão de ordem devido ao problema de sincronismo de relógio

Solução 1: Sincronização de relógios

Ajustar relógios periodicamente (atrasos dificultam)

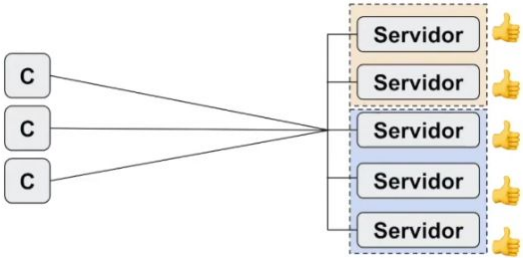
Solução 2: Relógios lógicos

Contador incremental de eventos indicando ordem

# Desafios - Exemplos

**Desafio**: Tolerar falhas e obter consenso dos servidores

Concordar em quais dados devem ser armazenados, na sua ordem e quando torná-los visíveis aos usuários




Solução: Aplicar algoritmos de consenso distribuídos

Juntar tudo que falamos e mais um pouco

Veremos melhor no decorrer da

# Estilos arquiteturais

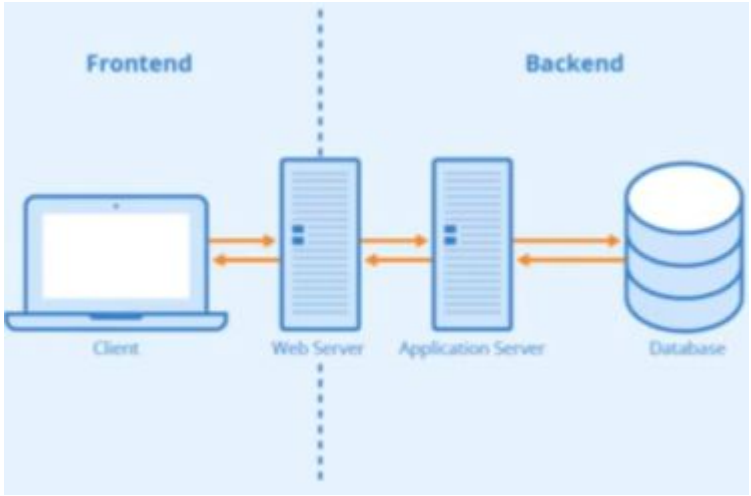
- Como o sistema é dividido em componentes?
- como os componentes estão conectados aos outros?
- Quais dados os componentes trocam entre eles?
- Como componentes são configurados conjuntamente no sistema?

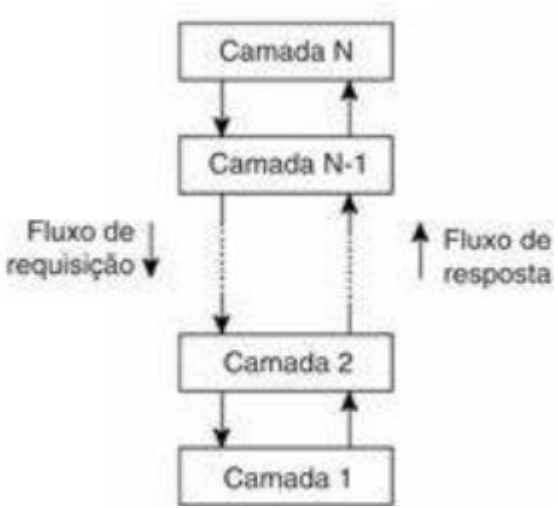
Estilos de arquitetura importantes para sistemas distribuídos:

- Arquitetura em Camadas
- Arquitetura orientada a objetos / Serviços
- Arquitetura baseada em recursos
- Arquitetura publish-subscribe

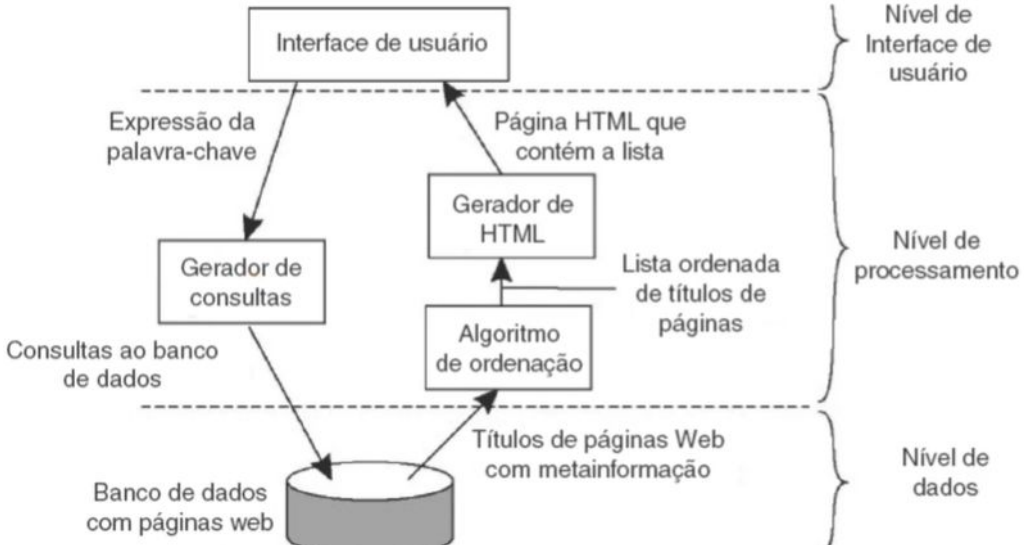
# Arquitetura em camadas

- O componente em uma camada só pode chamar a camada de abaixo e receber sua resposta.
    - Popular em sistemas Web (Fonteend/bakend/BD



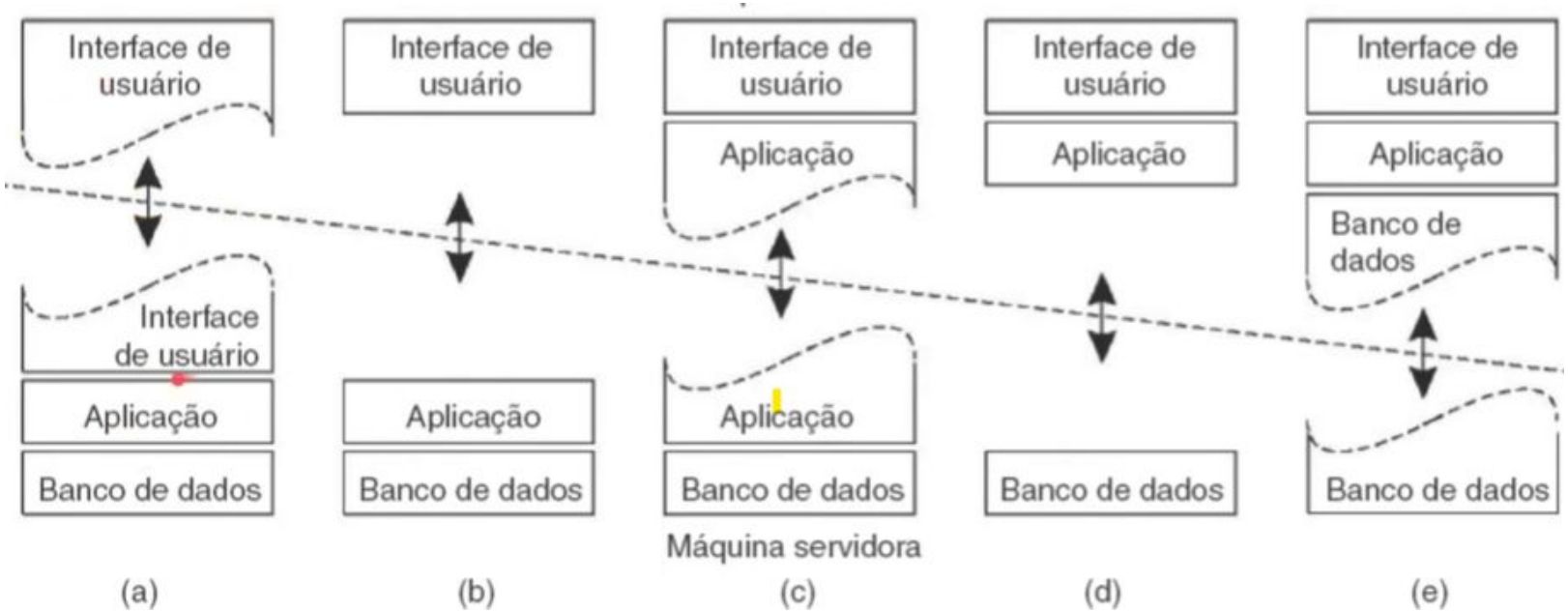



# Arquitetura em camadas




Organização simplificada de um mecanismo de busca da internet em três camadas diferentes.

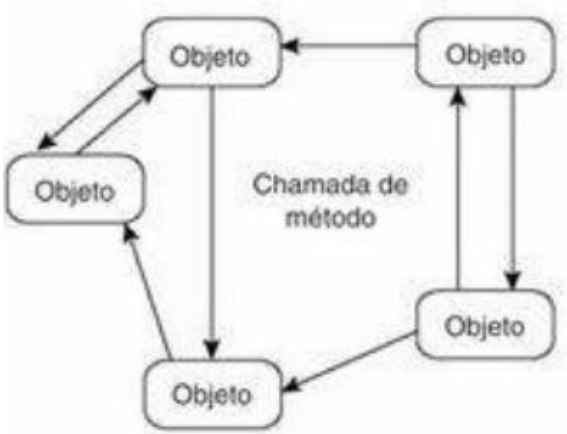
# Arquitetura em camadas

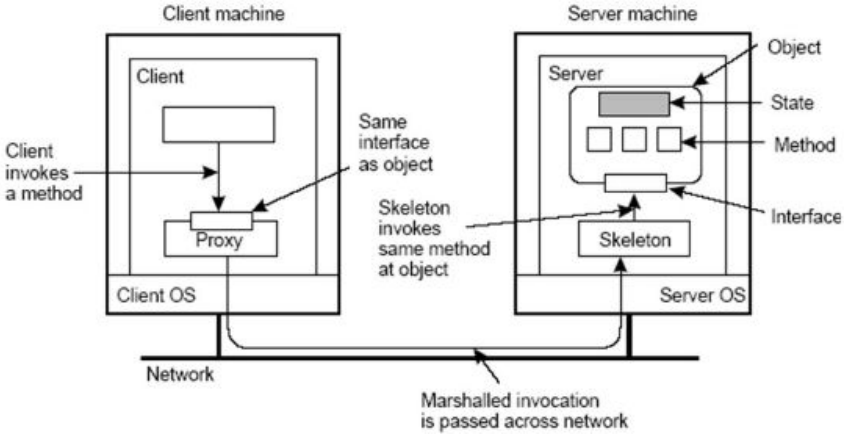



Alternativas de organizações cliente-servidor (a) e (e)

# Arquitetura Orientada a Objetos

- Objetos distribuídos interagem via chamada remota de métodos
  - Benefícios da orientação a objetos, rodando em máquinas diferentes.



# Arquitetura orientada a Serviços

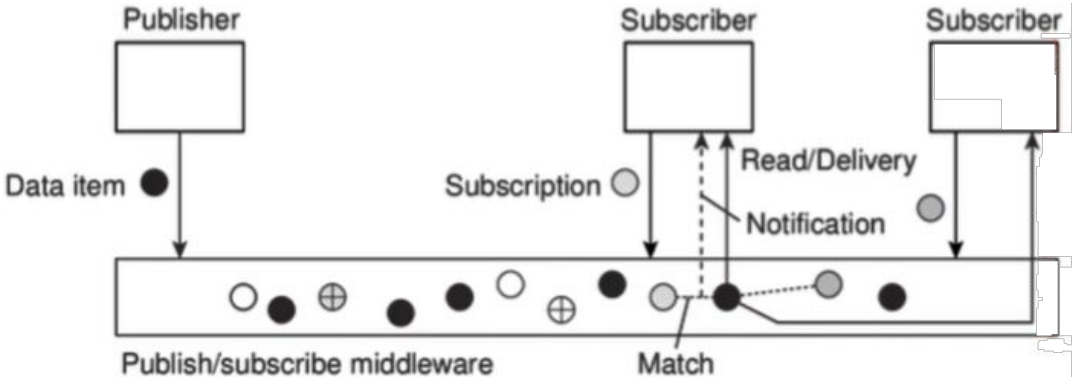
- Separação de serviços que operam de forma independente
  - Ideia similar aos objetos distribuídos
- Aplicação distribuída é composta por vários serviços diferentes
  - Importante ter interfaces bem definidas e padrão de comunicação

# Arquitetura Baseada em Recursos

- Sistema em uma coleção de recursos gerenciados individualmente
- Exemplo: Representational State Transfer (REST)
  - Recursos São identificados por esquema único de nomeação
  - Os serviços oferecem a mesma interface (GET, PUT, DELETE, POST)
  - Mensagem enviadas de / para serviços são-contidas
  - Os componentes não guardam estado das requisições(stateless)
  - Usa padrões populares (HTTP, JSON)
    - HTTP: Torna a comunicação mais simples, facilitando o uso pelos browsers
    - JSON: Organização dos dados
  - Caso precise guardar os estados anteriores, não é interessante utilizar o REST

# Arquitetura Publish-subscribe

- Um processo publica eventos para interessados (publish)
- Processos que se inscrevem e recebem os eventos (subscribe)
- Comunicação não exige referência direta. Pode ser:
  - **Baseado em eventos:** exige que ambos estejam conectados
  - **Espaço de dados compartilhados:** Comunicação offline



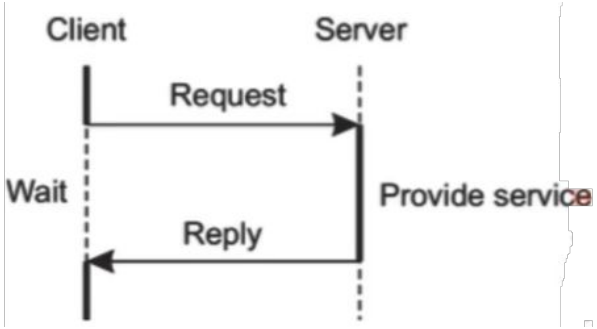

O middleware só permite acesso aos processos que se inscreveram para os eventos

# Arquitetura de Sistemas

- Quais componentes vão existir
  - Quais componentes deverão existir
  - Quais as suas responsabilidades
  - Como eles vão interagir?
  - Onde estarão localizados?
  -
- Formas de organização
  - Centralizada
  - Descentralizada
  - Híbrida

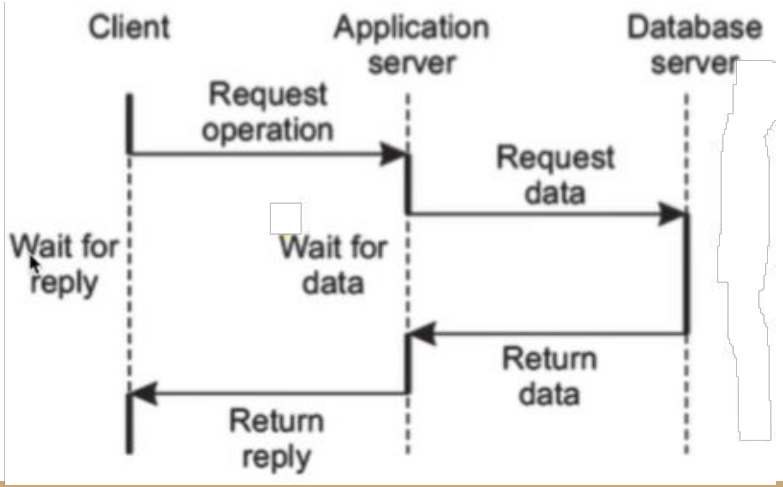
# Organização centralizada

- Arquitetura Cliente-Servidor
- Há processos oferecendo serviços (Servidor)
  - Servidor: Processo que implementa serviços específico
- Há processos que usam tais serviços oferecidos (clientes)
  - Cliente: Processo que requisita um serviço
    - Envia requisição (request) e guardar resposta do servidor (reply)

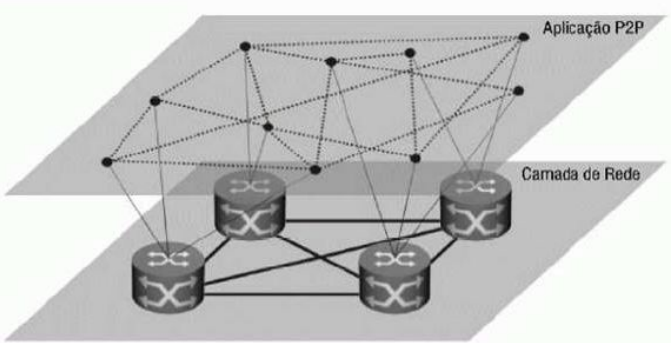


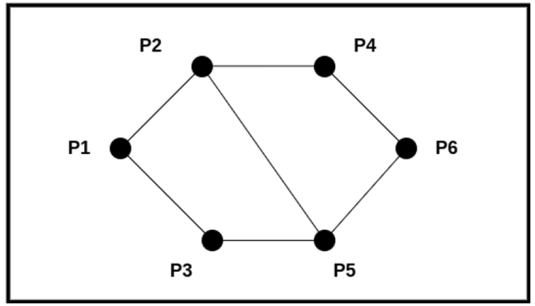
# Organização Centralizada

- **Arquitetura cliente-servidor multi-camadas**
- Componente em uma camada só acessa camada subsequente
  - Servidores podem agir como clientes, requisitando outros servidores
  - Exemplo: aplicações web em 3 camadas: front end, back end, database



# Organização descentralizada (p2p)

- **peer-to-peer:** Processos são clientes e servidores ao mesmo tempo
- **rede overlay (sobreposição):** rede de processos (nós) que podem se comunicar
  - Nem todos nós se comunicam diretamente
  - Não há um canal de comunicação direto entre todos os  pares de processos do sistema distribuído
    - Em alguns casos é necessário passar por um ou mais  intermediários
    - Envolve a questão do roteamento
  - 

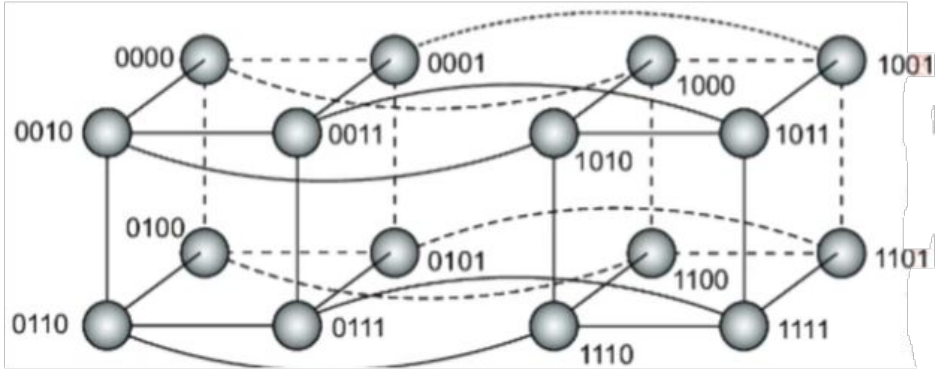
# Organização descentralizada (p2p)

- Tipos de organização de sistemas p2p:
  - Estruturada
  - Não estruturada
  - Hierárquica

# Organização descentralizada (p2p)

- **Sistemas p2p estruturados**
-
    - Topologia é bem determinada (anel, árvore, grade, etc)
    - EX: cada peer tem um id e se comunica no máximo com 4 peers;

      para buscar chave x (hash), repassa a requisição para vizinho mais próximo

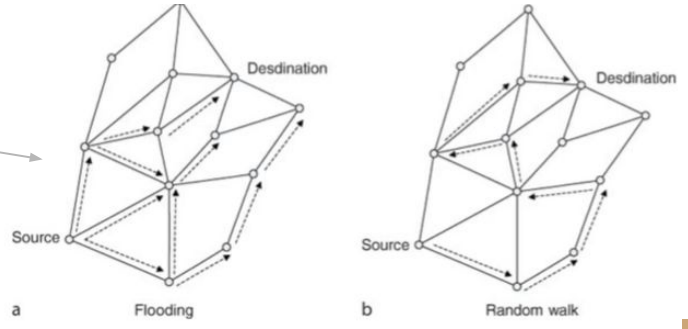


# Organização descentralizada (p2p)

## Sistemas p2p não estruturados

- A lista de vizinhos de cada peer é montada sem estrutura definida de forma ad hoc
  - Busca com Flooding (inundação): repassa requisição para todos os vizinhos
    - Pode alcançar o destino mais rápido
    - sobrecarrega a rede
  - Busca com Random walk (passeio aleatório): repassa para alguns vizinhos aleatórios
    - Reduz a quantidade de informações na rede
    - Pode demorar para alcançar o destino

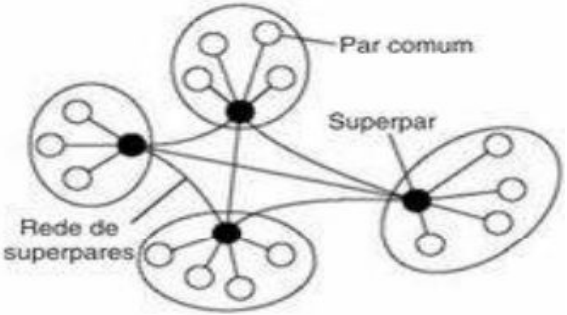
Time-to-Live



# Organização descentralizada (p2p)

## Sistemas p2p hierárquicos

- **Super peers** formam rede overlay para manter índice de dados
  - Auxilia os peers regulares a acharem dado / peer desejado
  - Coordena a comunicação entre os demais níveis
  - Exemplo Skype: super peers para buscar, login e lidar com firewall
  - Passam por uma eleição dentro do sistema (processamento, disponibilidade)
  - Um peer regular pode se associar em mais de um super peers

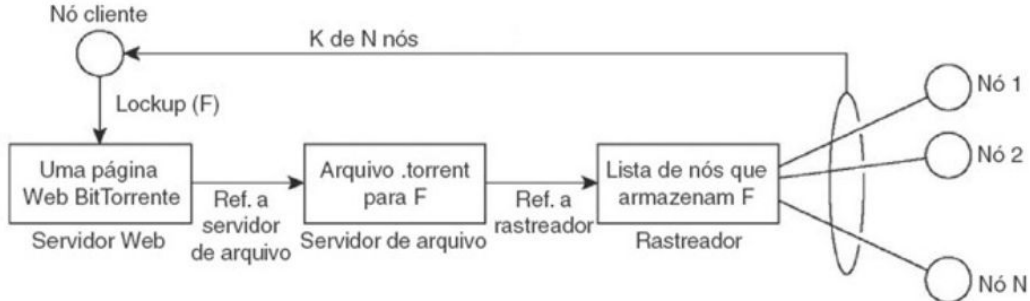


# Arquiteturas híbridas

**Sistemas distribuídos colaborativos (ex: BitTorrent)**

- Cliente-servidor para buscar arquivos + P2P para download/upload
- Trackers gerenciam peers ativos e sabem quem tem os dados
  - Os dados são segmentados e alocados em diversos peers
  - Estrutura centralizada ou distribuída em Distributed Hash Table (DHT) (pode usar os regulares)
  - Geralmente os Trackers são os super peers
- Peers fazem download/upload de blocos do arquivo com peers vizinhos

Cada F é dividido em blocos

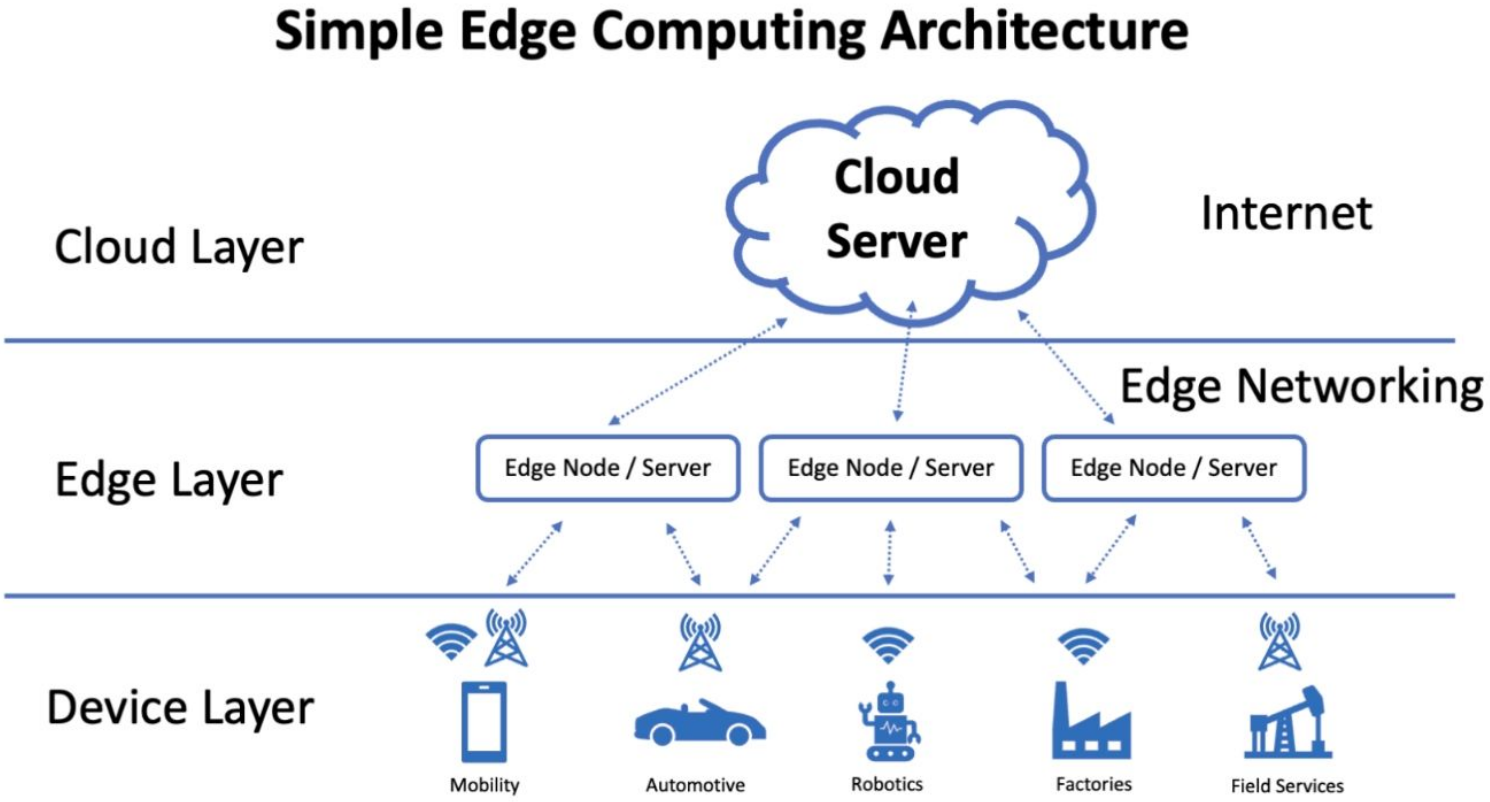


# Arquiteturas híbridas

- Edge computing (Computação de borda): servidores são posicionados na “borda” da rede
  - EX: limite entre a internet e a rede de um provedor de internet
  - Otimiza a distribuição de conteúdo e processamento
  - Dispositivos de clientes também podem fazer parte

- Na computação edge, dispositivos como sensores, câmeras, roteadores, ou pequenos servidores locais realizam o processamento dos dados ali mesmo, "na borda" (*edge*) da rede.
- Só informações relevantes ou processadas são enviadas para a nuvem ou datacenter, se necessário.

# Arquiteturas híbridas



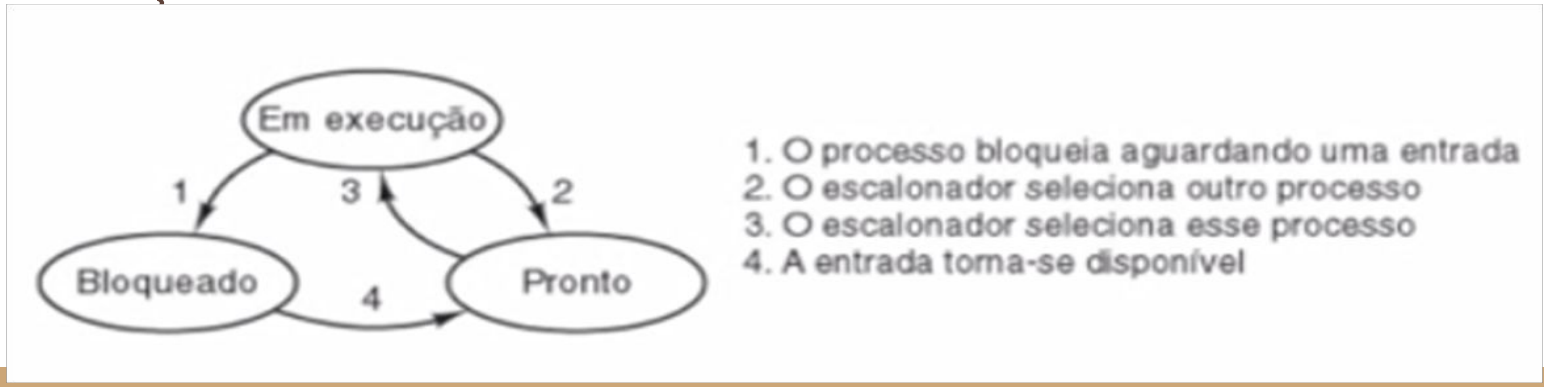
# Vantagem Edge Computer

**Vantagens:**

- **Baixa latência**: Respostas mais rápidas, ideais para aplicações em tempo real.
- **Menor uso de banda**: Menos dados precisam ser enviados para a nuvem.
- **Maior privacidade**: Dados sensíveis podem ser processados localmente.
- **Maior autonomia**: Funciona mesmo com conexão de rede limitada ou intermitente.

# Estados de um processo

- Processo: é um programa de computador em execução
- Estados Possíveis de Processo
  - Em Execução (Running) - Na CPU
  - Pronto (Ready) - Aguardando Acesso à CPU
  - Bloqueado (Blocked) - Evento Externo (I/O)
- 
- Transições entre Estados



# Comunicação entre processos

**Toda comunicação em sistemas distribuídos é baseada em troca de mensagem (baixo nível)**

No caso de sistemas distribuídos processos não compartilham memória

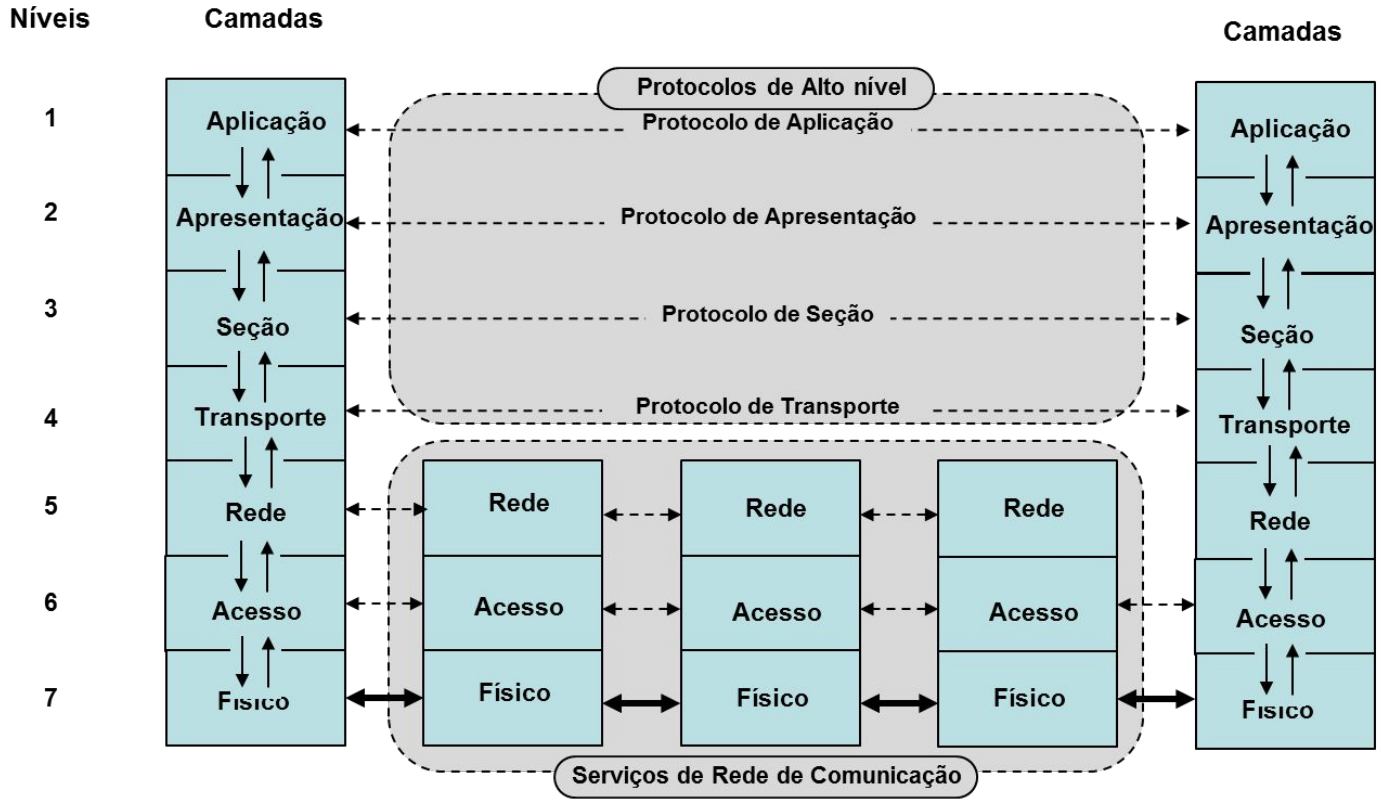
**Comunicação orientada a processos**

Lida diretamente com as primitivas da camada de transporte (TCP e UDP).

**Chamada remota de procedimento (RPC)**

Abstrai a troca de mensagens de baixo nível

# Protocolo de Comunicação



# Mensagens no protocolo TCP/IP

- **Comunicação de rede (IP)**
  - Comunicação best-effort de pacotes entre nós
    - Dados brutos sem garantias de recebimento, ordem, integridade
- **Camada de transporte (TCP / UDP)**
  - Comunicação entre processos
    - A comunicação IP é multiplexação em portas
  - Pode adicionar garantias

# Comunicação entre processos

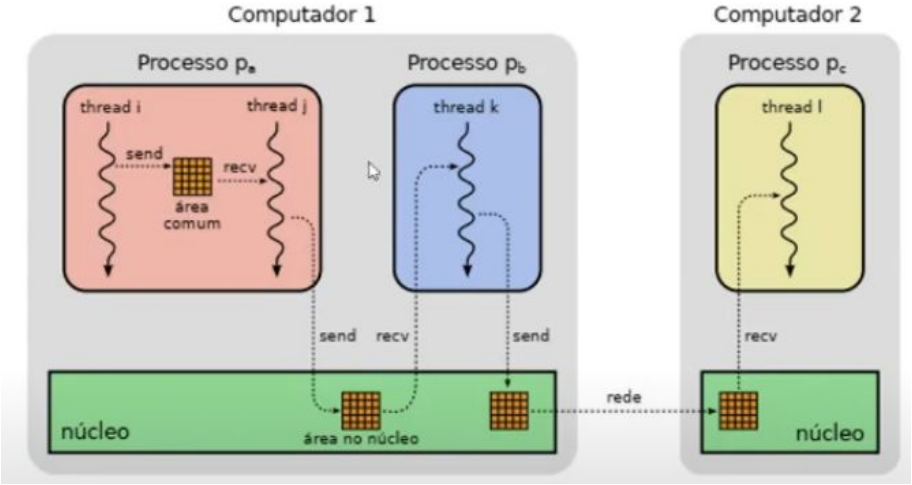
- Primitivas básicas:
  - **Send():** Envia mensagem
  - **Receive():** Recebe mensagem

Sockets

- Abstração de passagem de mensagem (datagrama) - UDP
- Abstração do fluxo TCP:
- Destino definido por <endereço ip>:<porta>

# Camada de transporte

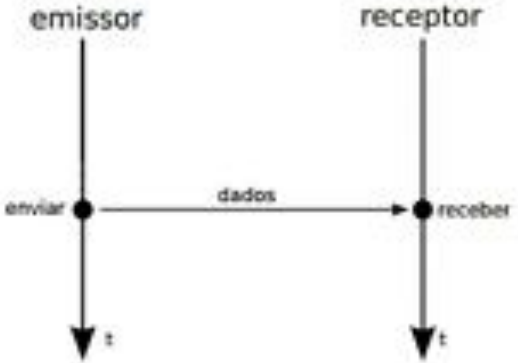
- Fornece os meios de comunicação atuais para a maioria dos sistemas distribuídos
- Protocolos de internet
  - **TCP**: Orientado a conexão, confiável, comunicação orientada por fluxo (pacotes sequenciados e não numerados)
  - **UDP**: Comunicação não confiável (melhor esforço) por datagramas



# Aspectos da Comunicação

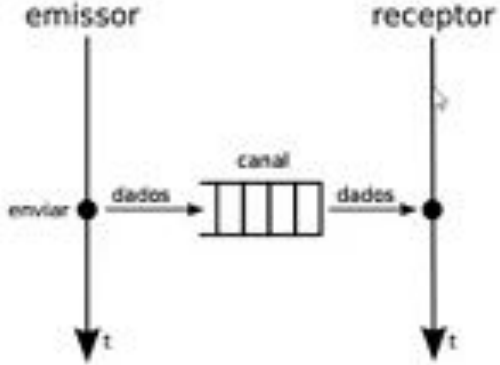
**Direta**

*enviar (dados, destino)*
*receber (dados, origem)*




**Indireta**

*enviar (dados, canal)*
*receber (dados, canal)*

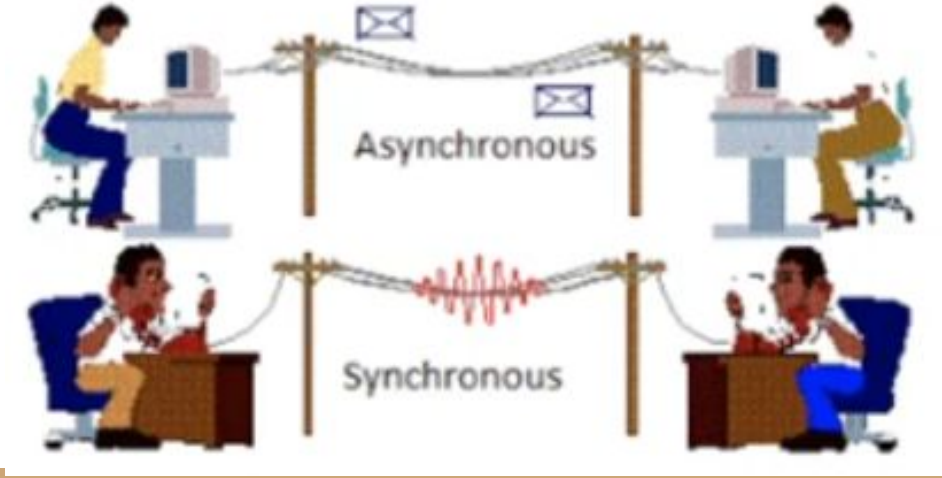



*Emissor e receptor não precisam se conhecer*

# Tipos de comunicação

Assíncrona (não bloqueante): o processo continua a executar após submeter a mensagem para transmissão:

Síncrona: (Bloqueante): Processo bloqueia até que a mensagem seja recebida



# Aspectos da Comunicação

- Capacidades dos canais (Buffers)
- Capacidade nula (n = 0)
  - Síncrona: Emissor fica bloqueado até que o emissor receba os dados e virce-versa.
  - Assíncrona: ñão viável
- Capacidade infinita n  = infinito
  - Não existe na prática
- Capacidade finita (0 < n < infinito)
  - Síncrona:Se o canal (buffer) estiver saturado, o emissor fica bloqueado até haver liberação de espaço
  - Assíncrona: Se o canal (buffer) estiver saturado, o emissor recebe um retorno indicando um erro

# Comunicação por UDP

- **UDP** é o **IP** no nível de transporte
  - Transmissão de datagramas convertidos em pacotes
  - checksum é opcional
  - Não há garantia de chegada ou ordenação
  - Não há retransmissão

# Comunicação por UDP

- Primitivas de comunicação
  - Send(): Envia datagramas, pode ser não-bloqueante
  - Receive(): Recebe datagramas, é bloqueante
    - Recebe de qualquer fonte (Canal “público”)
    - Pode ter  limite de espera (timeout) - (opção)
  - Mensagens são enviadas quando dadas ao IP e postas em um buffer de recebimento até o receive()
    - O que dar errado aqui?

# Comunicação por UDP

- O que pode dar errado?
  - Mensagens podem ser perdidas
    - EX: buffer de recebimento cheio
  - Mensagens podem sair de ordem
    - EX: seguindo caminhos diferentes de roteamento IP
- Essas informações nos orienta a que tipo de sistemas devemos construir com o protocolo UDP

# Comunicação por UDP

- UDP é util!
  - Exemplos de comunicação que podem ser desenvolvidos com UDP:
    - DNS, DHCP, SNMP, tracerouter, NTP
    - Transmissão de áudio
    - Transmissão de vídeo
    - etc.
  - Utilização eficiente de banda em alto desempenho

# Comunicação por TCP

- **TCP soma ao IP**
  - Confiabilidade (retransmissão se necessário)
  - Fluxo de dados (em conexões)
  - Segmentação automática dos fluxos de pacotes
  - Garantias com retransmissões, ordenação, checksums
  - Controle de fluxo
  - Buffering

# Primitivas do TCP

- accept() e connect(): estabelecem uma conexão
- send()  e receive(): Enviam / recebem bytes, não datagramas
- Abstraem:
  - Tamanho das mensagens (segmenta)
  - Perdas das mensagens (retransmissão)
  - Diferenças na velocidade de envio e recebimento (fluxo)
  - Duplicação e ordenação de mensagens
  - Destino das mensagens (outro lado da conexão)
    - Fonte conhecida e sendo a mesma até o fim da transmissão

# Comunicação por TCP

- **O que pode dar errado?**
  - Não há desordenação
  - Falhas são notificadas
    - Mas não são detectadas imediatamente (espera o ack não vir)
    - Difícil distinguir se a falha foi do processo ou da conexão
  - Conexão e comunicação é mais lenta que o UDP
    - Exige conexão e comunicações (acks)

# Comparativo resumido entre TCP X UDP

| TCP | UDP |
| --- | --- |
| Confiável | Não confiável |
| Orientado a conexão | Sem conexão |
| Retransmissão de segmentos e controle de fluxo | Sem retransmissão e sem controle de fluxo |
| Sequenciamento de segmentos | Sem sequenciamento |
| Confirmação de recebimento de segmentos (ACK) | Sem confirmação |