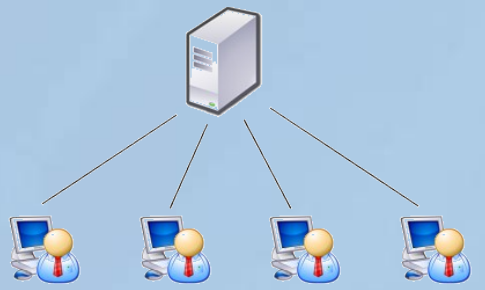
# Arquiteturas de Software
# Problemas e soluções

Marcos Monteiro, MBA, ITIL V3

http://www.marcosmonteiro.com.br
contato@marcosmonteiro.com.br

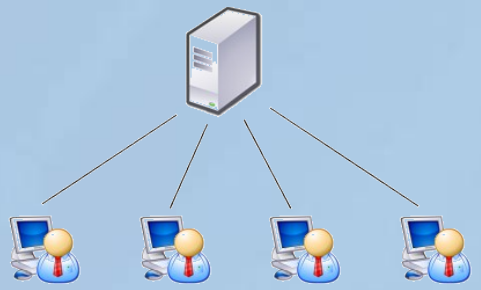
# Cliente - Servidor



- Cada instância de um cliente pode enviar requisições de dado para algum dos servidores conectados e esperar pela resposta.

- Por sua vez, algum dos servidores disponíveis pode aceitar tais requisições, processá-las e retornar o resultado para o cliente.

# Cliente - Servidor

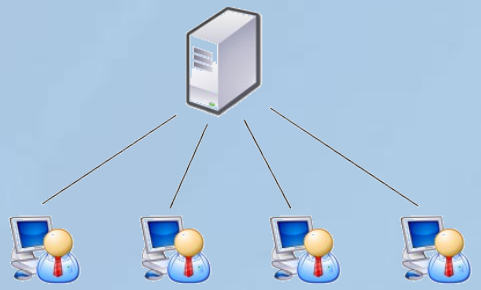


- Muitas vezes os clientes e servidores se comunicam através de uma rede de computador com hardwares separados, mas o cliente e servidor podem residir no mesmo sistema. A máquina servidor é um host que está executando um ou mais programas de servidor que partilham os seus recursos com os clientes.

- Um cliente não compartilha de seus recursos, mas solicita o conteúdo de um servidor ou função de serviço.

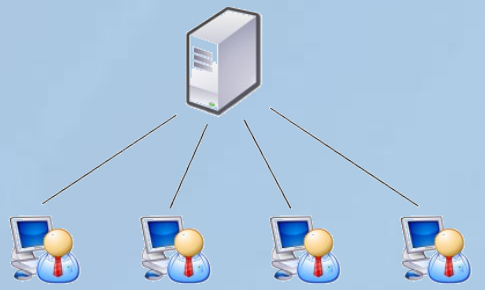
- Os clientes, portanto, iniciam sessões de comunicação com os servidores que esperam as solicitações de entrada.

# Descrição



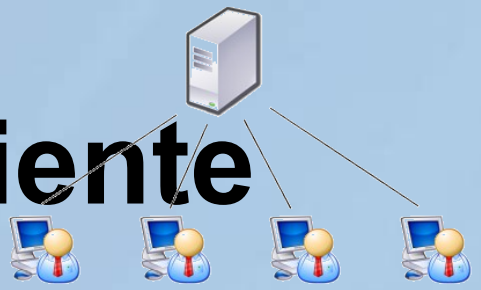
- Funções como a troca de e-mail, acesso à internet e acessar banco de dados, são construídos com base no modelo cliente-servidor.
  - Por exemplo, um navegador da web é um programa cliente em execução no computador de um usuário que pode acessar informações armazenadas em um servidor web na Internet.
  - Usuários de serviços bancários acessando do seu computador usam um cliente navegador da Web para enviar uma solicitação para um servidor web em um banco. Esse programa pode, por sua vez encaminhar o pedido para o seu próprio programa de banco de dados do cliente que envia uma solicitação para um servidor de banco de dados em outro computador do banco para recuperar as informações da conta. O saldo é devolvido ao cliente de banco de dados do banco, que por sua vez, serve-lhe de volta ao cliente navegador exibindo os resultados para o usuário.

# Descrição



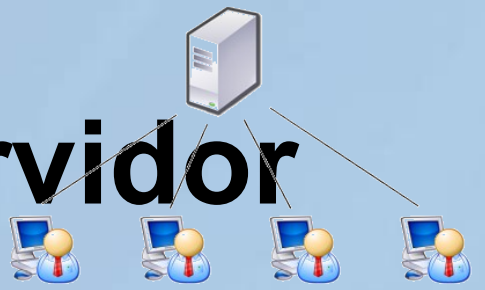
- Cada instância de software do cliente pode enviar requisições de dados a um ou mais servidores ligados.
- Por sua vez, os servidores podem aceitar esses pedidos, processá-los e retornar as informações solicitadas para o cliente.
- Embora este conceito possa ser aplicado para uma variedade de razões para diversos tipos de aplicações, a arquitetura permanece fundamentalmente a mesma.

# Características do Cliente



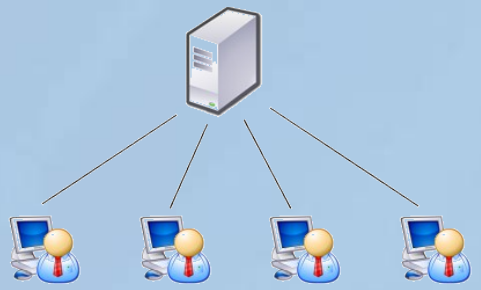
- Sempre inicia pedidos de servidores;
- Espera por respostas;
- Recebe respostas;
- Normalmente, se conecta a um pequeno número de servidores de uma só vez;
- Normalmente, interage diretamente com os usuários finais através de qualquer interface com o usuário , como interface gráfica do usuário.

# Características do Servidor



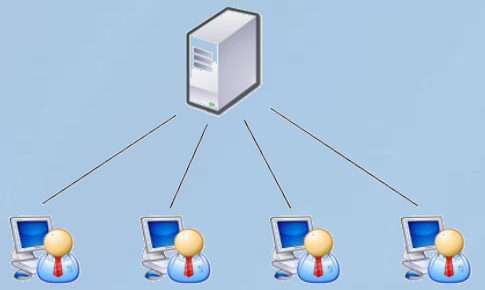
- Sempre esperar por um pedido de um dos clientes;
- Serve os clientes pedidos, em seguida, responde com os dados solicitados aos clientes;
- Um servidor pode se comunicar com outros servidores, a fim de atender uma solicitação do cliente.

# ☺ Vantagens



- Na maioria dos casos, a arquitetura cliente-servidor permite que os papéis e responsabilidades de um sistema de computação possa ser distribuído entre vários computadores independentes que são conhecidos por si só através de uma rede.
  - Isso cria uma vantagem adicional para essa arquitetura:
    - **Maior facilidade de manutenção**. Por exemplo, é possível substituir, reparar, atualizar ou mesmo realocar um servidor de seus clientes, enquanto continuam a ser a consciência e não afetado por essa mudança;

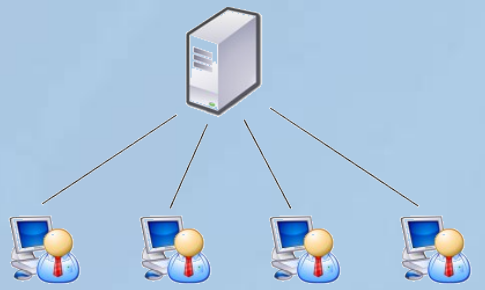
# ☺ Vantagens



- Todos os dados são armazenados nos servidores, que geralmente possuem controles de segurança muito maior do que a maioria dos clientes.

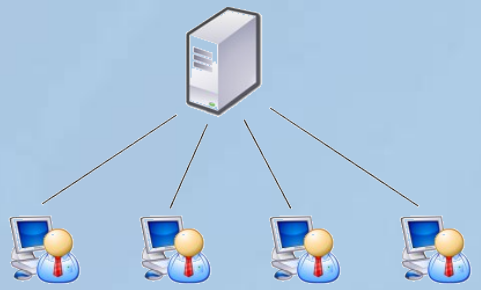
- Servidores podem controlar melhor o acesso e recursos, para garantir que apenas os clientes com as permissões adequadas podem acessar e alterar dados;

# ☺ Vantagens



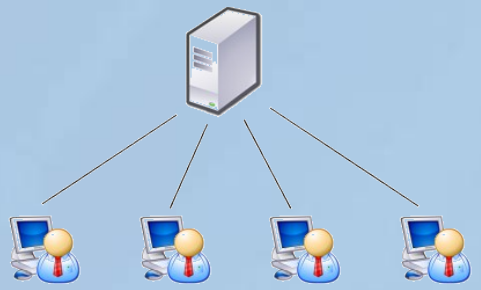
- Desde o armazenamento de dados é centralizada, as atualizações dos dados são muito mais fáceis de administrar, em comparação com o paradigma P2P, onde uma arquitetura P2P, atualizações de dados podem precisar ser distribuída e aplicada a cada ponto na rede, que é o time-consuming é passível de erro, como pode haver milhares ou mesmo milhões de pares;

# ☺ Vantagens



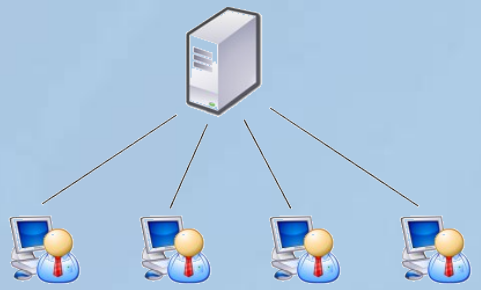
- Muitas tecnologias avançadas de cliente-servidor já estão disponíveis, que foram projetadas para garantir a segurança, facilidade de interface do usuário e facilidade de uso;

# ☺ Vantagens



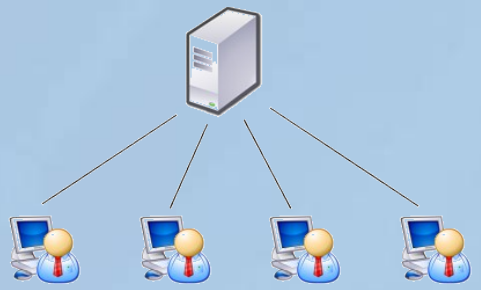
- Funciona com vários clientes diferentes de capacidades diferentes.

# ☹ Desvantagens



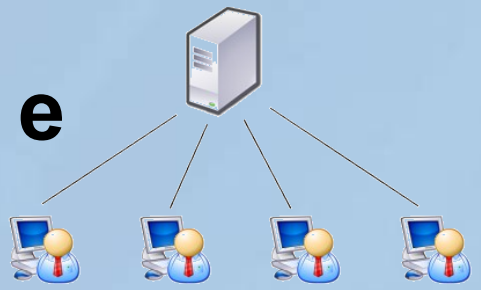
- Redes de tráfego de bloqueio é um dos problemas relacionados com o modelo cliente-servidor.
  - Como o número de solicitações simultâneas de cliente para um determinado servidor, o servidor pode ficar sobrecarregado;

# ☹ Desvantagens



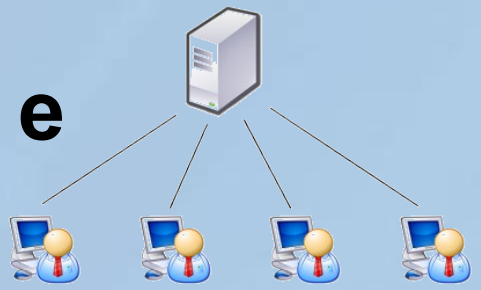
- O paradigma cliente-servidor não tem a robustez de uma rede P2P.
  - Sob cliente-servidor, se um servidor crítico falhar, os pedidos dos clientes não podem ser cumpridos.
  - Em redes P2P, os recursos são normalmente distribuídos entre vários nós. Mesmo se um ou mais nós partem e abandonam baixar um arquivo, por exemplo, os nós restantes ainda deve ter os dados necessários para completar o download.

# Protocolos de transporte e aplicações de rede



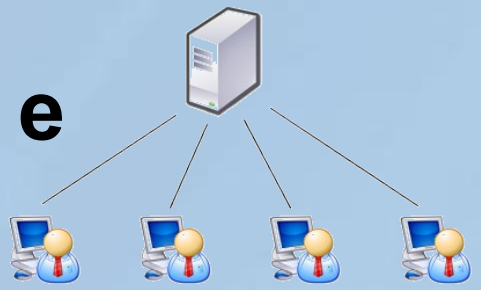
- Os protocolos do nível de transporte fornecem serviços que garantem uma transferência confiável de dados e aplicativos entre computadores (ou outros equipamentos) remotos. Os programas na camada de aplicação usam os protocolos de transporte para contactar outras aplicações. Para isso, a aplicação interage com o software do protocolo antes de ser feito o contacto. A aplicação que aguarda a conexão informa ao software do protocolo local que está pronta a aceitar mensagem. A aplicação que estabelece a conexão usa os protocolos de transporte e rede para contactar o sistema que aguarda. As mensagens entre as duas aplicações são trocadas através da conexão resultante.

# Protocolos de transporte e aplicações de rede



- Existem duas formas para que se estabeleça uma ligação cliente-servidor: enquanto uma delas é orientada à conexão, a outra não é.
  - O TCP, por exemplo, é um protocolo de transporte orientado à conexão em que o cliente estabelece uma conexão com o servidor e ambos trocam múltiplas mensagens de tamanhos variados, sendo a aplicação do cliente quem termina a sessão.
  - Já o protocolo UDP não é orientado à conexão, nele o cliente constrói uma mensagem e a envia num pacote UDP para o servidor, que responde sem estabelecer uma conexão permanente com o cliente.
    - Ver arquivo services:
      - Windows : \WINDOWS\system32\drivers\etc\services
      - Linux : /etc/services

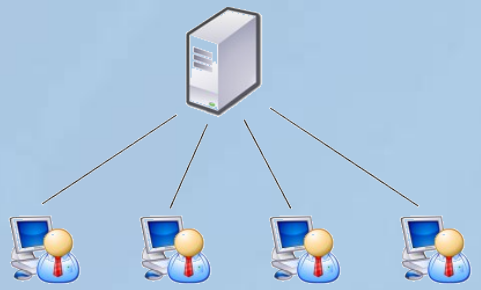
# Protocolos de transporte e aplicações de rede



- Comandos do Prompt de comando que auxiliaram em teste de conexão:
  - ping
    - Testar se o há rede com o servidor
      - ping <IP ou host>
        - » Ex: ping 192.168.0.1
        - » Espera-se por resposta, envia-se 4 pacotes e se recebe 4 pacotes.
  - telnet
    - Um otima forma de testar camada de transporte
      - telnet <IP>  <PORTA>
        - » Ex: telnet 192.168.0.1 8080
        - » Testando um servidor tomcat que esteja operando na porta 8080, qualquer mensagem ou ausência desta é indicador de conexão, o problema se dará caso apareça “falha de conexão”.

OBS: Um firewall do servidor por bloquear resposta ao ping e ao telnet.

# Referências




- MENDES, Antonio. Arquitetura de Software: desenvolvimento orientado para arquitetura. Editora Campus. Rio de Janeiro - RJ, 2002.

# Arquiteturas em n Camadas

Marcos Monteiro, MBA, ITIL V3

# Problema: Pressões do negócio

- Steve Jobs fala:
  - "Internet time" é um novo conceito (final dos anos 1990) e significa que as empresas devem implementar novos sistemas *muito* rapidamente
- Muitos sistemas de empresas devem migrar para Internet/Intranet/Extranet
  - Novos tipos de sistemas devem ser desenvolvidos para usar a tecnologia como *business advantage*, dando um diferencial nos negócios
- Muitos sistemas devem mudar devido a mudanças nos negócios
  - Fusões e aquisições

Resultado: tem *muito* mais software a fazer, *muito mais rapidamente*

- Mas há pressões para manter custos de IT (*Information Technology*) baixos

# Problema: Pressões Tecnológicas

- O desenvolvimento de software ficou muito mais complexo nos últimos anos
  - Pelos motivos acima
  - Porque usuários querem funcionalidade mais sofisticada
- Problemas de complexidade
  - O desenvolvimento de muitos grandes sistemas tem fracassado recentemente
  - A figura mais citada: 80% dos projetos são fracassos!
- Resumindo: fazer sistemas de produção customizados do zero in-house:
  - É muito caro
  - Demora muito tempo
  - Não produz boa qualidade

**O que grandes empresas, com grandes problemas de desenvolvimento de sistemas, querem, tecnologicamente?**

- Melhor flexibilidade
  - Possibilitando satisfazer novos requisitos do negócio (=funcionalidade) rapidamente
- Melhor adaptabilidade
  - Possibilitando customizar uma aplicação para vários usuários, usando várias alternativas de delivery dos serviços da aplicação com impacto mínimo ao núcleo da aplicação
- Melhor manutenabilidade
  - Possibilitando atualizar uma aplicação, mas minimizando o impacto da maioria das mudanças

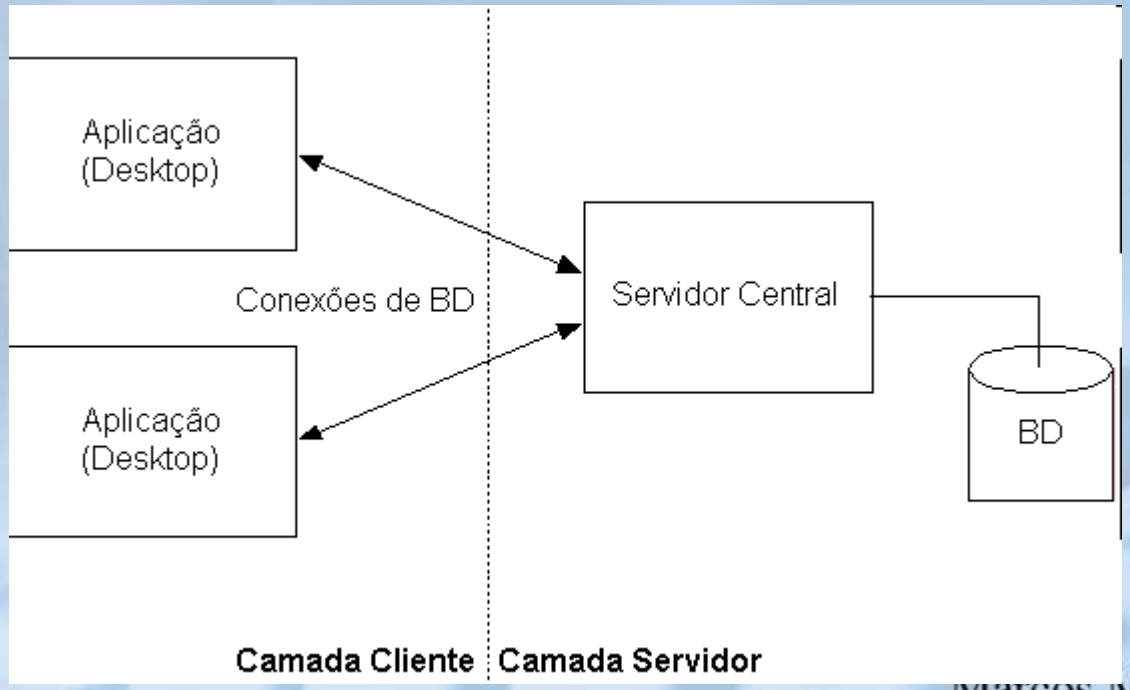
- Melhor reusabilidade
  - Possibilitando rapidamente montar aplicações únicas e dinâmicas
- Melhor aproveitamento do legado
  - Possibilitando reusar a funcionalidade de sistemas legados em novas aplicações
- Melhor interoperabilidade
  - Possibilitando integrar 2 aplicações executando em plataformas diferentes
- Melhor escalabilidade
  - Possibilitando distribuir e configurar a execução da aplicação para satisfazer vários volumes de transação

- Menor tempo de desenvolvimento
  - Equivalente a querer Produtividade do Programador
- Melhor robustez
  - Possibilitando ter soluções com menos defeitos
  - Possibilitando ter confiabilidade e disponibilidade
- Menor risco
  - Possibilitando tudo que falamos acima e ainda não se arriscar a ter projetos fracassados

Resumindo: tudo que ISO 9126 caracteriza como "qualidade de software"

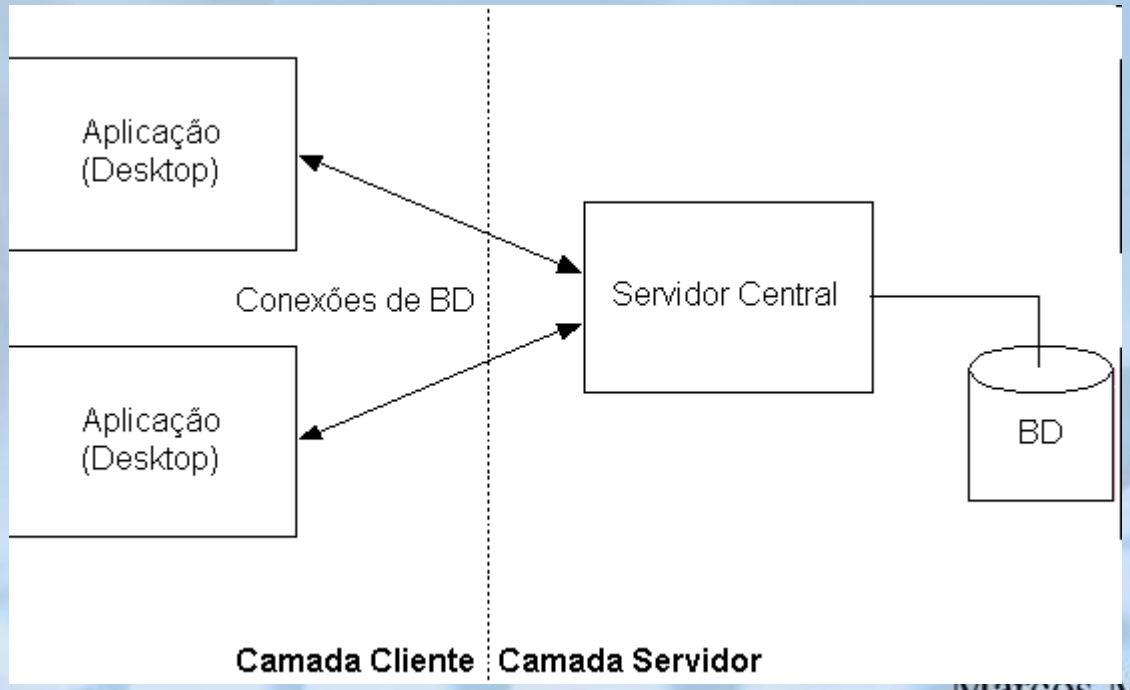
# Arquitetura em 2 camadas

- Os primeiros sistemas cliente-servidor eram de duas camadas
  - Camada cliente trata da lógica de negócio e da UI
  - Camada servidor trata dos dados (usando um SGBD)



# Arquitetura em 2 camadas

- Os primeiros sistemas cliente-servidor eram de duas camadas
  - Camada cliente trata da lógica de negócio e da UI
  - Camada servidor trata dos dados (usando um SGBD)

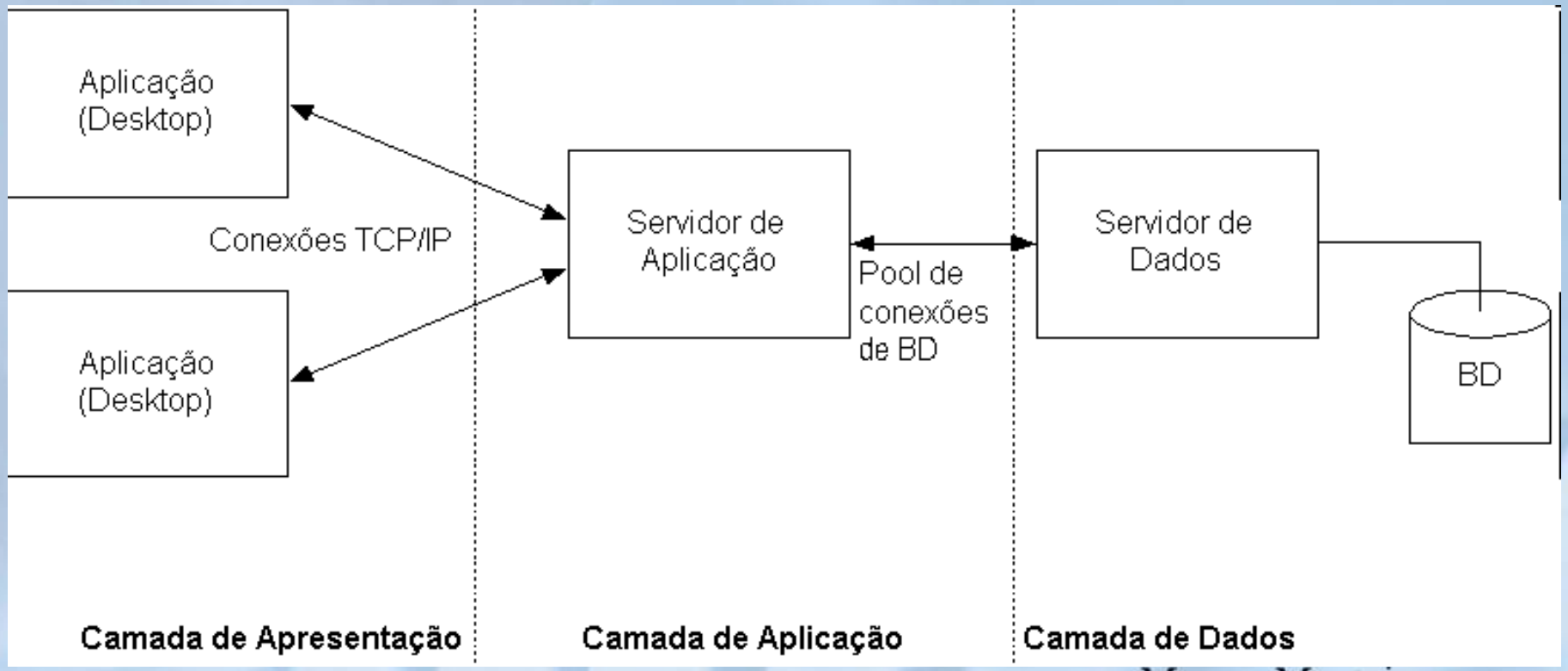


# PROBLEMAS!!!

- A arquitetura cliente/servidor em 2 camadas sofria de vários problemas:
  - Falta de escalabilidade (conexões a bancos de dados)
  - Enormes problemas de manutenção (mudanças na lógica de aplicação forçava instalações)
  - Dificuldade de acessar fontes heterogêneas

# Arquitetura em 3 camadas

- Camada de apresentação (UI)
- Camada de aplicação (business logic)
- Camada de dados



# Arquitetura em 3 camadas

- Problemas de manutenção foram reduzidos, pois mudanças às camadas de aplicação e de dados não necessitam de novas instalações no desktop

- Observe que as camadas são lógicas
  - Fisicamente, várias camadas podem executar na mesma máquina
  - Quase sempre, há separação física de máquinas

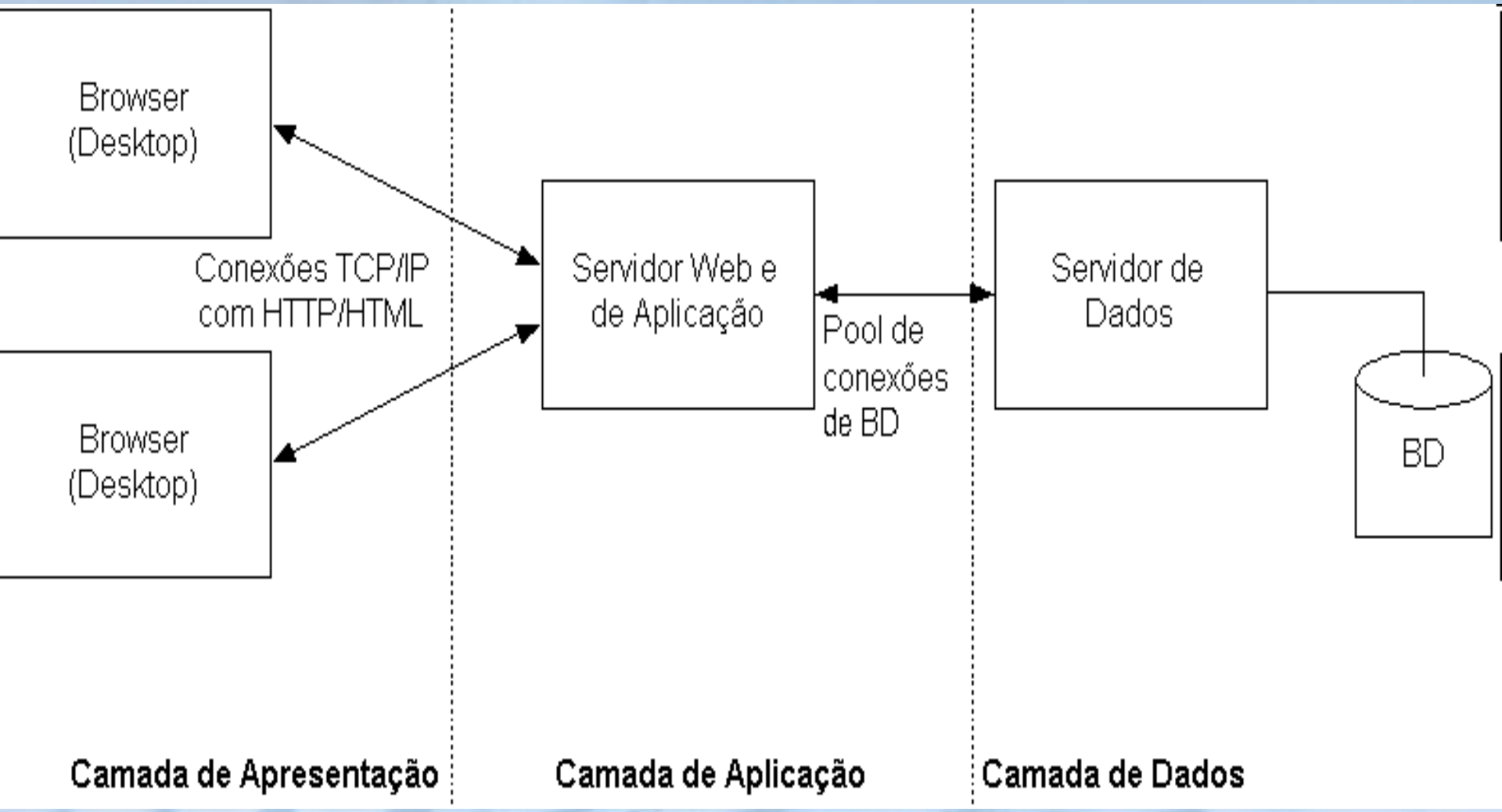
A arquitetura em 3 camadas original sofre de problemas:

- A instalação inicial dos programas no desktop é cara
- O problema de manutenção ainda persiste quando há mudanças à camada de apresentação
- Não se pode instalar software facilmente num desktop que não está sob seu controle administrativo
  - Em máquinas de parceiros
  - Em máquinas de fornecedores
  - Em máquinas de grandes clientes
  - Em máquinas na Internet

# Arquitetura em 3/4 camadas Web-Based

- Então, usamos o Browser como Cliente Universal
- Conceito de Intranet
- A camada de aplicação se quebra em duas: Web e Aplicação
- Evitamos instalar qualquer software no desktop e portanto, problemas de manutenção
- Evitar instalação em computadores de clientes, parceiros, fornecedores, etc.
- Às vezes, continua-se a chamar isso de 3 camadas porque as camadas Web e Aplicação freqüentemente rodam na mesma máquina (para pequenos volumes)

# Arquitetura em 3/4 camadas Web-Based



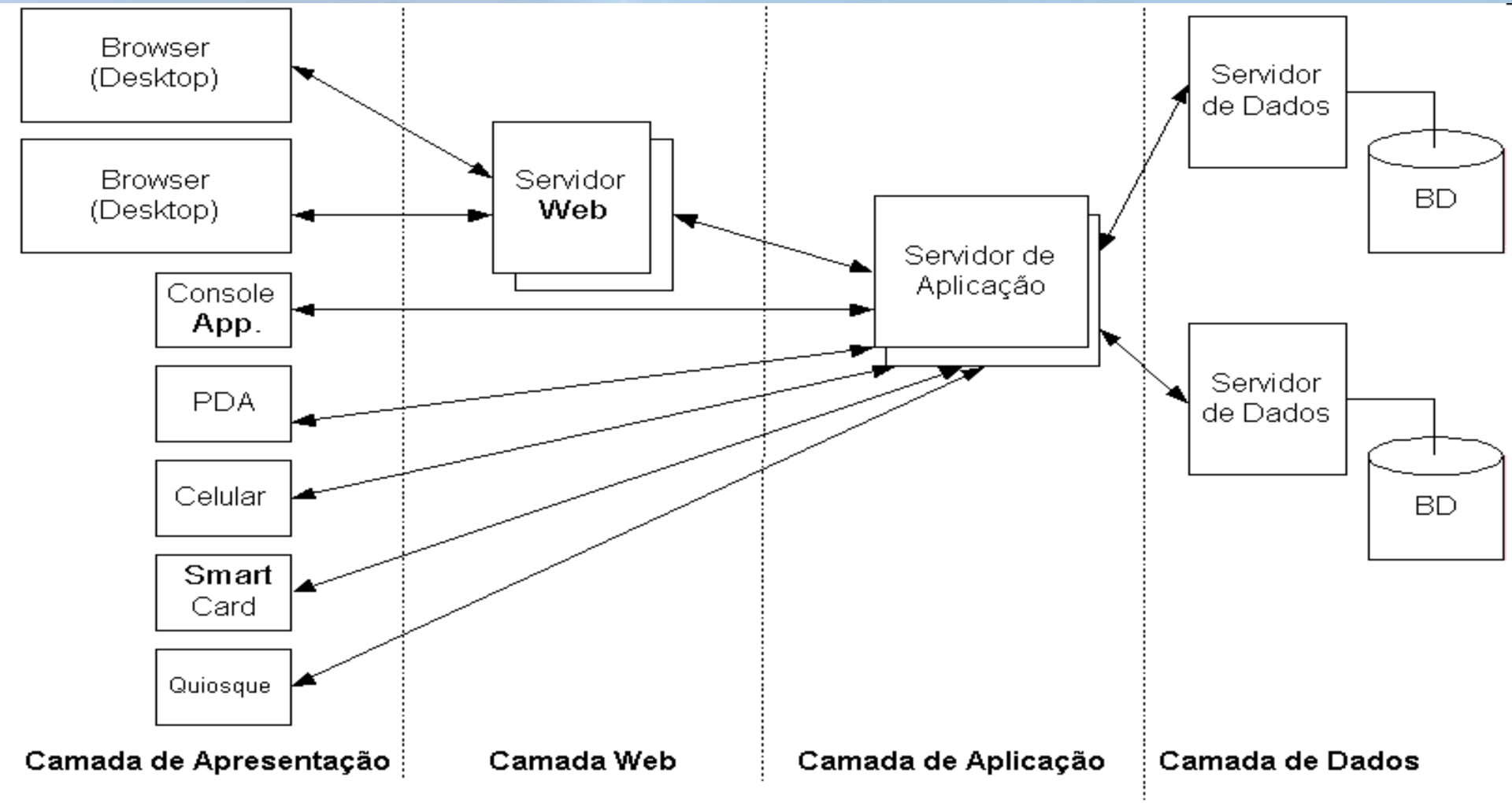
# os problemas remanescentes:

1. Não há suporte a Thin Clients (PDA, celulares, smart cards, quiosques, ...) pois preciso usar um browser (pesado) no cliente

- Dificuldade de criar software reutilizável: cadê a componentização?

- Fazer aplicações distribuídas multicamadas é difícil. Tem que:
  - Implementar persistência (*impedance mismatch* entre o mundo OO e o mundo dos BDs relacionais)
  - Implementar tolerância a falhas com fail-over
  - Implementar gerência de transações distribuídas
  - Implementar balanceamento de carga
  - Implementar *resource pooling*

- As camadas podem ter vários nomes:
  - Apresentação, interface, cliente
  - Web
  - Aplicação, Business
  - Dados, Enterprise Information System (EIS)



os problemas remanescentes:

O browser e o web server é Client-Server

- O truque é introduzir *middleware* num servidor de aplicação que ofereça esses serviços automaticamente
- Além do mais, as soluções oferecidas (J2EE, .Net) são baseadas em componentes
- Balanceamento de Cargas

# Computação distribuída

Marcos Monteiro, MBA, ITIL V3

A computação distribuída, ou sistema distribuído, é uma referência à computação paralela e descentralizada, realizada por dois ou mais computadores conectados através de uma rede, cujo objetivo é concluir uma tarefa em comum.

# Definição

- Um sistema distribuído é:
- "coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente";

Andrew Tanenbaum

- "coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados"

George Coulouris

# Definição

- O suporte completo de um sistema de banco de dados distribuídos implica que uma única aplicação seja capaz de operar de modo transparente sobre dados dispersos em uma variedade de banco de dados diferentes, gerenciados por vários SGBDs diferentes, em execução em uma variedade de máquinas diferentes que podem estar rodando em diversas plataformas diferentes e uma variedade de sistemas operacionais. Onde o modo transparente diz respeito à aplicação operar sob um ponto de vista lógico como se os dados fossem gerenciados por um único SGBD, funcionando em uma única máquina com apenas um sistema operacional.

# Definição

- Assim, a computação distribuída consiste em adicionar o poder computacional de diversos computadores interligados por uma rede de computadores ou mais de um processador trabalhando em conjunto no mesmo computador, para processar colaborativamente determinada tarefa de forma coerente e transparente, ou seja, como se apenas um único e centralizado computador estivesse executando a tarefa. A união desses diversos computadores com o objetivo de compartilhar a execução de tarefas, é conhecida como sistema distribuído.

# Organização

- Organizar a interação entre cada computador é primordial. Visando poder usar o maior número possível de máquinas e tipos de computadores, o protocolo ou canal de comunicação não pode conter ou usar nenhuma informação que possa não ser entendida por certas máquinas. Cuidados especiais também devem ser tomados para que as mensagens sejam entregues corretamente e que as mensagens inválidas sejam rejeitadas, caso contrário, levaria o sistema a cair ou até o resto da rede.

# Organização

- Outro fator de importância, é a habilidade de mandar softwares para outros computadores de uma maneira portável de tal forma que ele possa executar e interagir com a rede existente. Isso pode não ser possível ou prático quando usando hardware e recursos diferentes, onde cada caso deve ser tratado separadamente com cross-compiling ou reescrevendo software.

# Modelos de computação distribuída

- **Cliente/Servidor**
  - O *cliente* manda um pedido para o *servidor* e o *servidor* o retorna.
- **Peer-to-peer (P2P)**
  - É uma arquitetura de sistemas distribuídos caracterizada pela descentralização das funções na rede, onde cada nodo realiza tanto funções de servidor quanto de cliente.
- **Objetos Distribuídos**
  - Semelhante ao peer-to-peer, mas com um Middleware "mediador" intermediando o processo de comunicação.

# Software

- **Fracamente acoplados**
  - Permitem que máquinas e usuários de um sistema distribuído sejam fundamentalmente independentes e ainda interagir de forma limitada quando isto for necessário, compartilhando discos, impressoras e outros recursos.
- **Fortemente acoplados**
  - provê um nível de integração e compartilhamento de recursos mais intenso e transparente ao usuário caracterizando sistemas operacionais distribuídos.

# Cluster

Um cluster, ou aglomerado de computadores, é formado por um conjunto de computadores, que utiliza um tipo especial de sistema operacional classificado como sistema distribuído. Muitas vezes é construído a partir de computadores convencionais (personal computers), os quais são ligados em rede e comunicam-se através do sistema, trabalhando como se fossem uma única máquina de grande porte. Há diversos tipos de cluster. Um tipo famoso é o cluster da classe Beowulf, constituído por diversos nós escravos gerenciados por um só computador.

# Tipos de cluster

- Cluster de Alto Desempenho:
  - Também conhecido como cluster de alta performance, ele funciona permitindo que ocorra uma grande carga de processamento com um volume alto de gigaflops em computadores comuns e utilizando sistema operacional gratuito, o que diminui seu custo.

# Tipos de cluster

- Cluster de Alta Disponibilidade:
  - São clusters os quais seus sistemas conseguem permanecer ativos por um longo período de tempo e em plena condição de uso. Sendo assim, podemos dizer que eles nunca param seu funcionamento; além disso, conseguem detectar erros se protegendo de possíveis falhas..

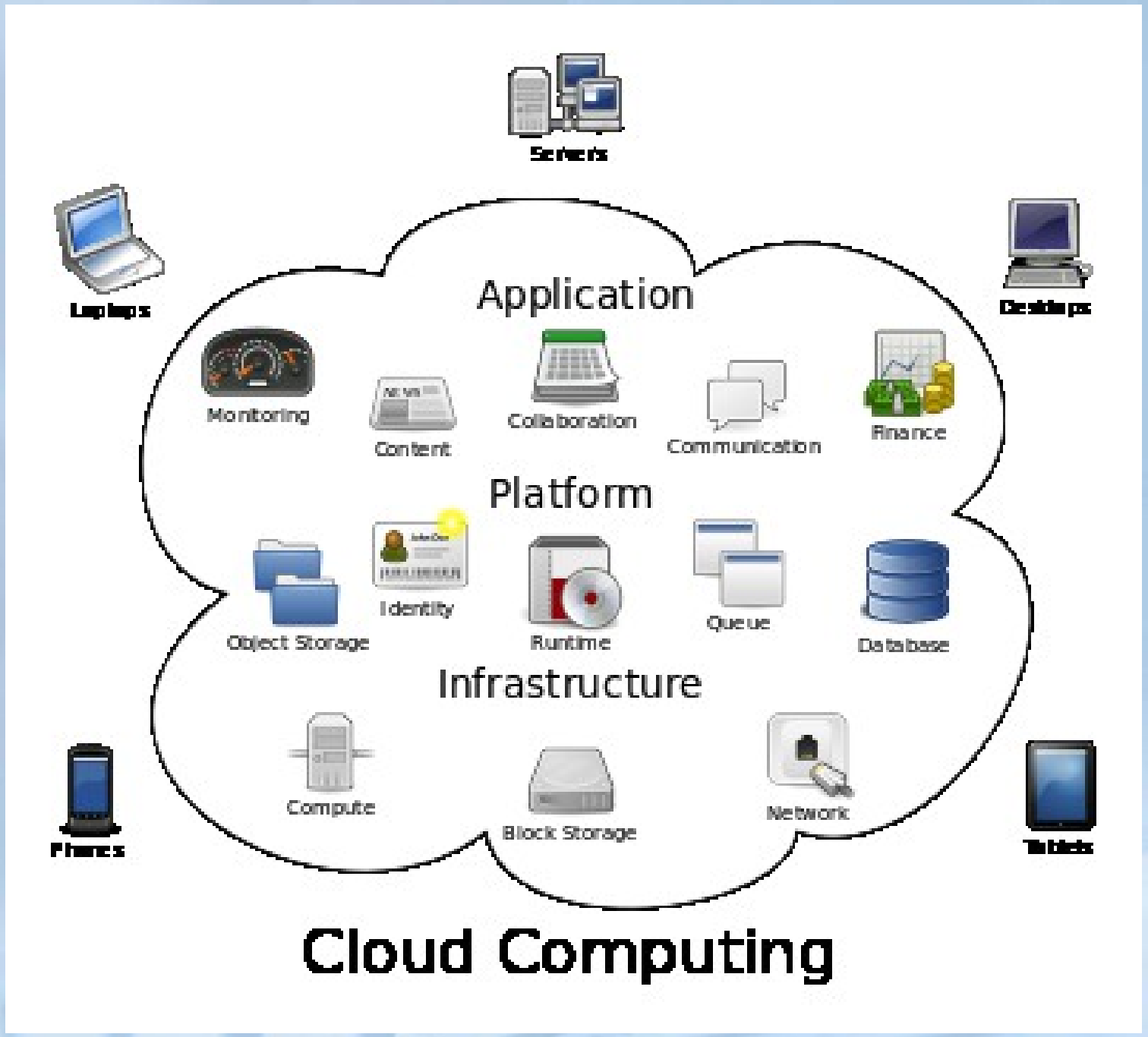
# Tipos de cluster

- Cluster para Balanceamento de Carga:
  - Esse tipo de cluster tem como função controlar a distribuição equilibrada do processamento. Requer um monitoramento constante na sua comunicação e em seus mecanismos de redundância, pois se ocorrer alguma falha, haverá uma interrupção no seu funcionamento.

O exemplo mais moderno desse paradigma é o BOINC, que é um framework de grade computacional no qual diversos projetos podem rodar suas aplicações, como fazem os projetos World Community Grid, SETI@Home, ClimatePrediction.net, Einstein@Home, PrimeGrid e OurGrid.

# Referências


- Tanenbaum, Andrew S., *Distributed Systems: Principles and Paradigms*, pg. 2


Servers
Laptops
Desktops
Application
Monitoring
Content
Collaboration
Communication
Finance
Platform
Identity
Object Storage
Runtime
Queue
Database
Infrastructure
Compute
Block Storage
Network
Phones
Tablets
Cloud Computing