


SISTEMAS DISTRIBUÍDOS

Arquiteturas de SD


Prof. Guilherme C. Kurtz



Arquiteturas

- ▶ Sistemas distribuídos muitas vezes são **complexos**, sendo que um **único software** pode estar **dividido** em vários **pedaços** espalhados por **diversas máquinas**;
 - ▶ De forma a **controlar** tal **complexidade**, é crucial que estes **sistemas** estejam **organizados adequadamente**;
 - ▶ A organização se dá em diversas formas, sendo as principais em relação a **organização lógica** dos componentes de software e a **organização física** propriamente dita;
- 

Arquiteturas

- ▶ As **arquiteturas** de sistemas distribuídos irão tratar principalmente dos **componentes de software** que compõe tal sistema;
 - ▶ Tais arquiteturas irão nos dizer como estes **componentes** deverão estar **organizados** e como os mesmos irão **interagir**;
 - ▶ Há **diversas opções** de como os **componentes** de software estarão **distribuídos** em máquinas reais;
- 

Arquiteturas

- ▶ **Estilos Arquitetônicos:**
 - Os **estilos arquitetônicos** irão definir a **organização lógica** de um sistema distribuído;
- ▶ **Arquiteturas de sistemas:**
 - As **arquiteturas de sistemas** irão definir a **distribuição física dos componentes de software** de um sistema distribuído.

SISTEMAS DISTRIBUÍDOS

Estilos Arquitetônicos


Prof. Guilherme C. Kurtz



Estilos Arquitetônicos

- ▶ Os estilos arquitetônicos irão definir:
 - O modo como os **componentes** estão **conectados**
 - Os **dados** que serão **trocados** entre os **componentes**;
 - O modo como os **componentes** estarão **configurados** para formar o **sistema**;
- ▶ Primeiramente, é necessário definir alguns conceitos. Um **componente** é uma unidade modular com **interfaces bem definidas**, os quais são **substituíveis** dentro do ambiente;


Estilos Arquitetônicos

- ▶ Portanto, é importante ressaltar que um **componente pode ser substituído**, desde que seja **respeitado suas interfaces**;
 - ▶ Um outro conceito importante são os **conectores**. Estes são mecanismos que servirão de **mediadores na comunicação** e **cooperação entre componentes**;
 - Ex: chamadas de procedimentos remotos, troca de mensagens, fluxo de dados, etc.
- 

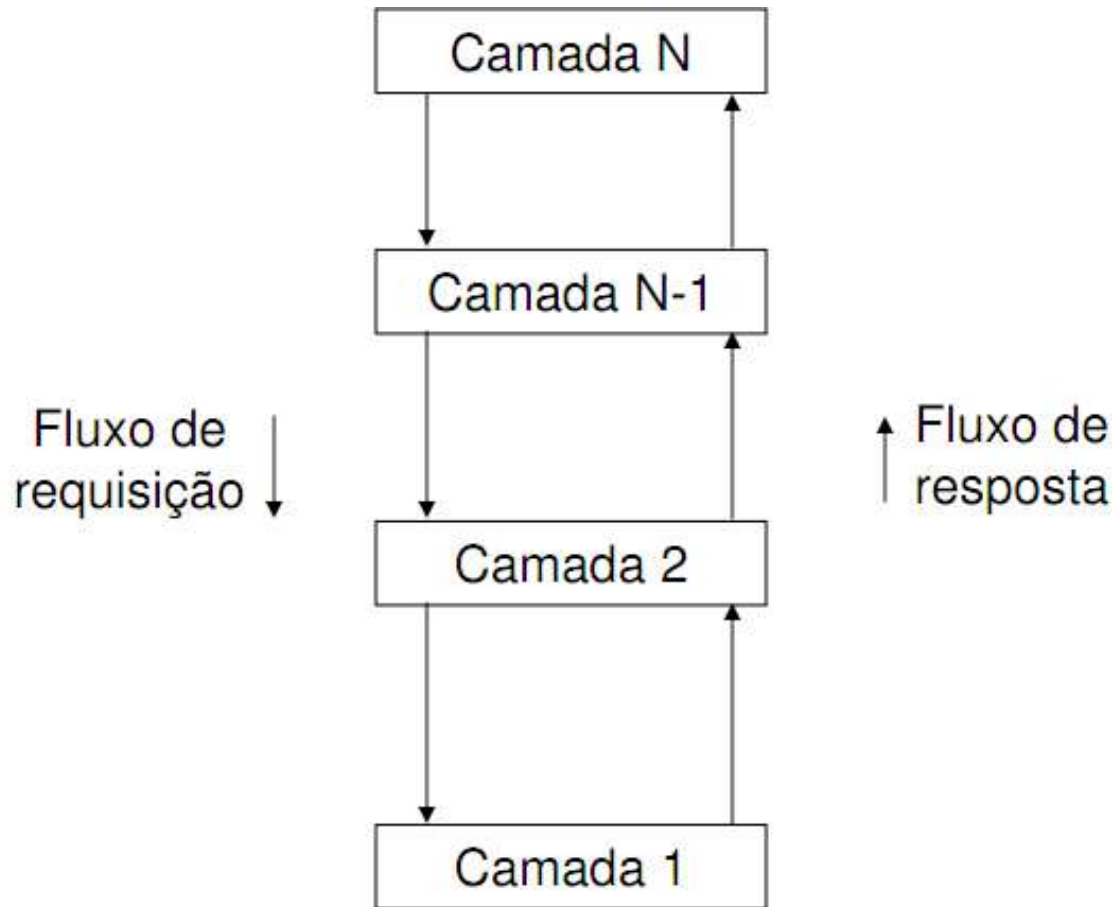
Estilos Arquitetônicos

- ▶ Com a utilização de **componentes** e **conectores** pode-se chegar a **diversas configurações lógicas de SDs**, sendo elas classificadas como **estilos arquitetônicos**;
- ▶ Dentre os estilos mais importantes, destacam-se:
 - Arquiteturas em camadas;
 - Arquiteturas baseadas em objetos;
 - Arquiteturas centradas em dados;
 - Arquiteturas baseadas em eventos.

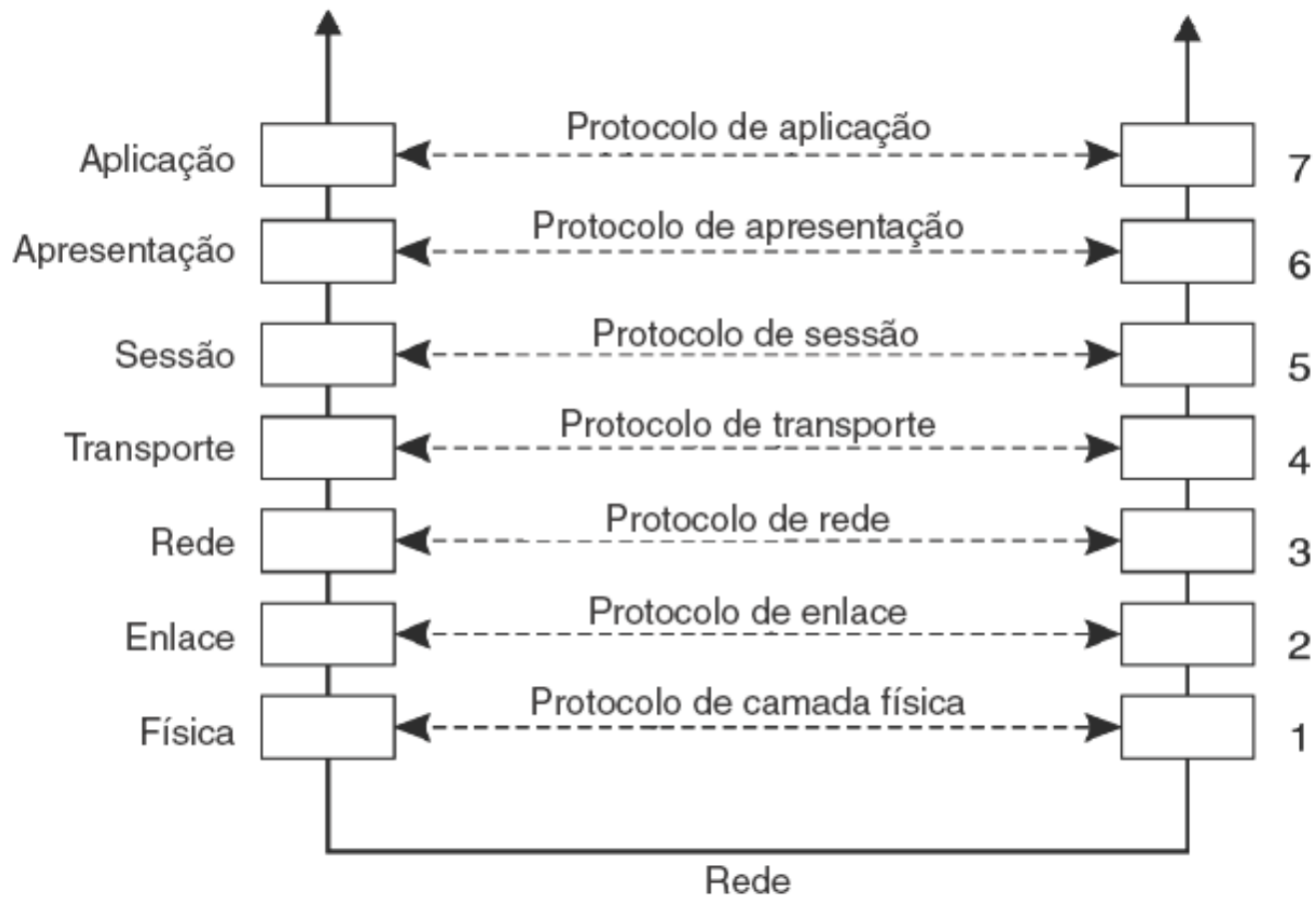
Arquiteturas em Camadas

- ▶ A idéia básica é: os **componentes** são **organizados em camadas**;
 - ▶ Um componente da **camada N** tem **permissão de chamar** componentes na **camada N-1**, mas não o contrário;
 - ▶ É um modelo bastante adotado **em redes de computadores**;
 - ▶ O controle flui de camada para camada: **requisições descem** na hierarquia e **resultados fluem para cima**;
- 

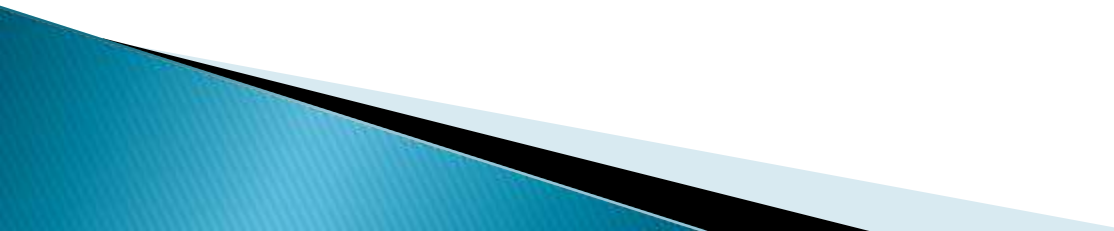
Arquiteturas em Camadas



Arquiteturas em Camadas



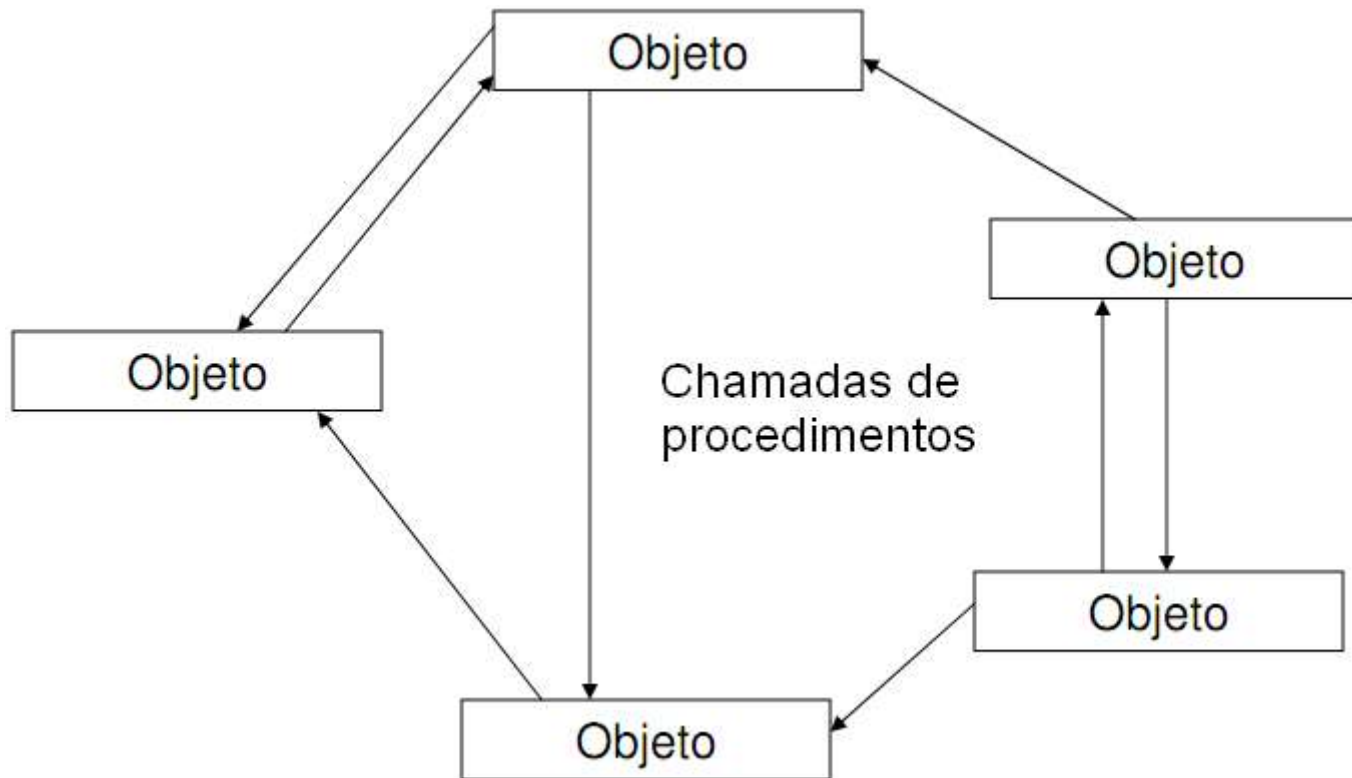
Arquiteturas baseadas em objetos

- ▶ Uma organização bem **mais solta** que a de camadas;
 - ▶ Neste estilo arquitetônico, um **objeto corresponde** ao que foi definido anteriormente como um **componente**;
 - ▶ Estes componentes são conectados uns aos outros através de um mecanismo de **chamadas de procedimentos remotos**;
- 

Arquiteturas baseadas em objetos

- ▶ Este estilo arquitetônico se ajusta à arquitetura de sistema **cliente-servidor**:
 - **Servidor** implementa e **cria o objeto**;
 - **Cliente** **acessa os métodos** deste **objeto** através de chamadas de procedimento remoto;
- ▶ **As arquiteturas em camadas e baseadas em objetos** formam os estilos **mais importantes** para sistemas computacionais de grande porte;

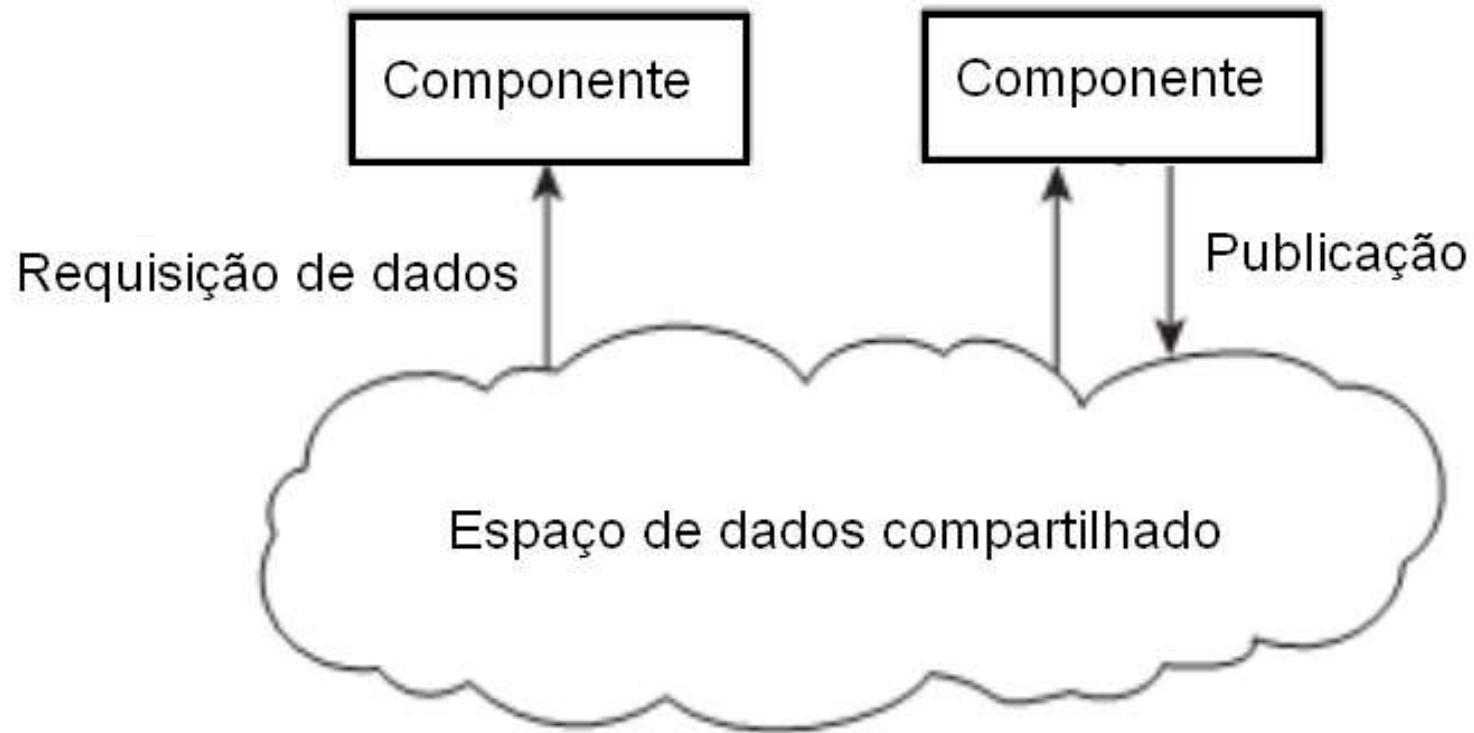
Arquiteturas baseadas em objetos



Arquiteturas centradas em dados

- ▶ Estilos centrados em dados são desenvolvidos na idéia de que os **processos se comunicam** por meio de um **repositório comum**:
 - Passivo ou ativo;
- ▶ Este tipo de arquitetura para sistemas distribuídos são tão **importantes** quanto as arquiteturas em **camadas** ou **baseadas em objetos**;
- ▶ **Sistemas distribuídos baseados na Web**, em grande parte, são **centrados em dados**, pois:
 - são um **grande conjunto de aplicações**;
 - dependem de um **sistema distribuído de arquivos compartilhados**
 - Quase toda a **comunicação** ocorre **através destes arquivos**;

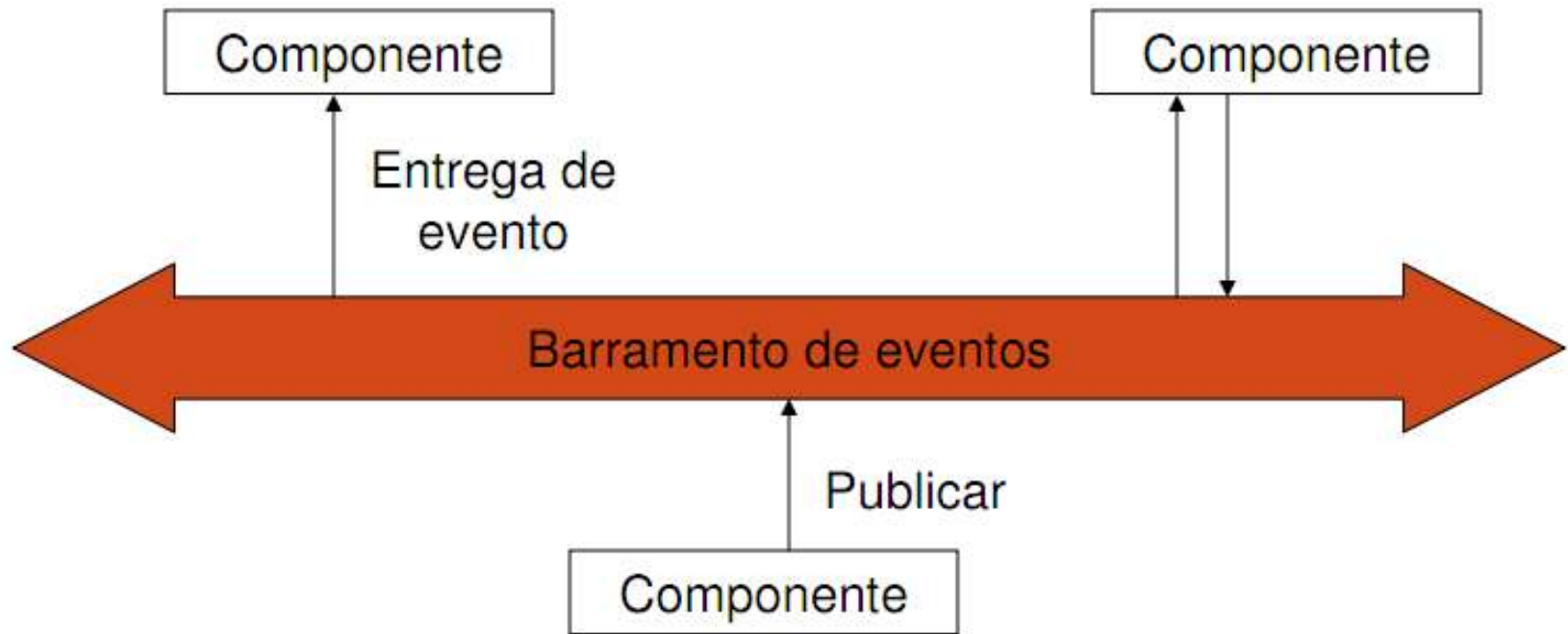
Arquiteturas centradas em dados



Arquiteturas baseadas em eventos

- ▶ **Comunicação entre os processos baseada na propagação de eventos;**
 - Que também pode transportar dados;
- ▶ **Sistemas publicar/subscrever:**
 - Certos processos **publicam** eventos;
 - Middleware assegura que **somente os processos** que se **subscreveram** para estes eventos o receberão;
 - Processos **fracamente acoplados**: processos não se **referem** explicitamente uns aos outros, mas sim aos **eventos** que lhe interessarem;

Arquiteturas baseadas em eventos



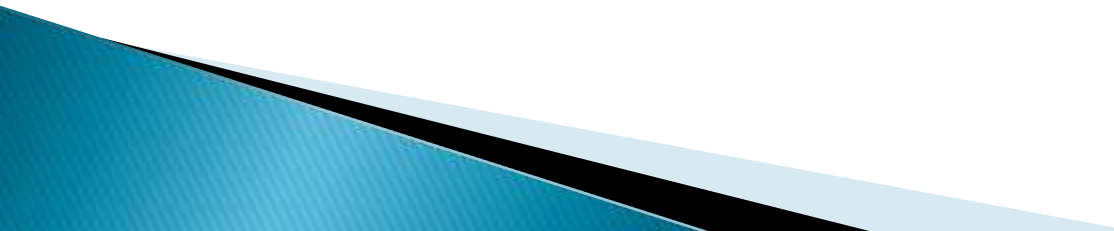
SISTEMAS DISTRIBUÍDOS

Arquiteturas de Sistemas

Prof. Guilherme C. Kurtz



Arquiteturas de sistemas

- ▶ As arquiteturas de sistemas irão definir questões a respeito dos **componentes de software** e sua **colocação em máquinas reais**;
 - ▶ Desta forma, as seguintes questões são tratadas:
 - Como os diversos **sistemas distribuídos** são realmente **organizados**;
 - Onde serão **colocados** os **componentes** de software;
 - Como é estabelecida a **interação** entre estes **componentes**;
- 

Arquiteturas de sistemas

- ▶ Dentre as principais arquiteturas, temos:
 - **Arquiteturas centralizadas:**
 - Cliente-servidor;
 - **Arquiteturas descentralizadas:**
 - Peer-to-peer;
 - **Arquiteturas híbridas;**

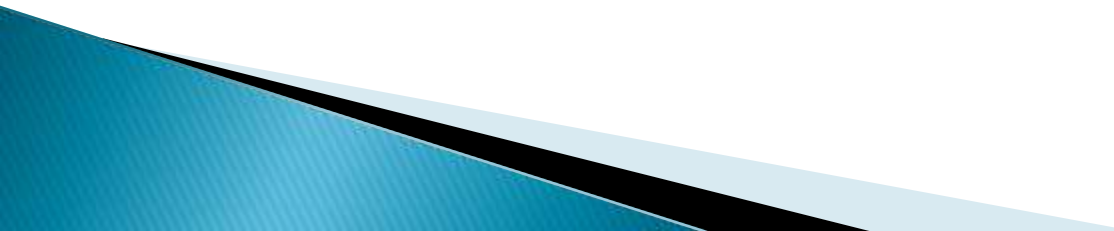
SISTEMAS DISTRIBUÍDOS

Arquiteturas Centralizadas

Prof. Guilherme C. Kurtz

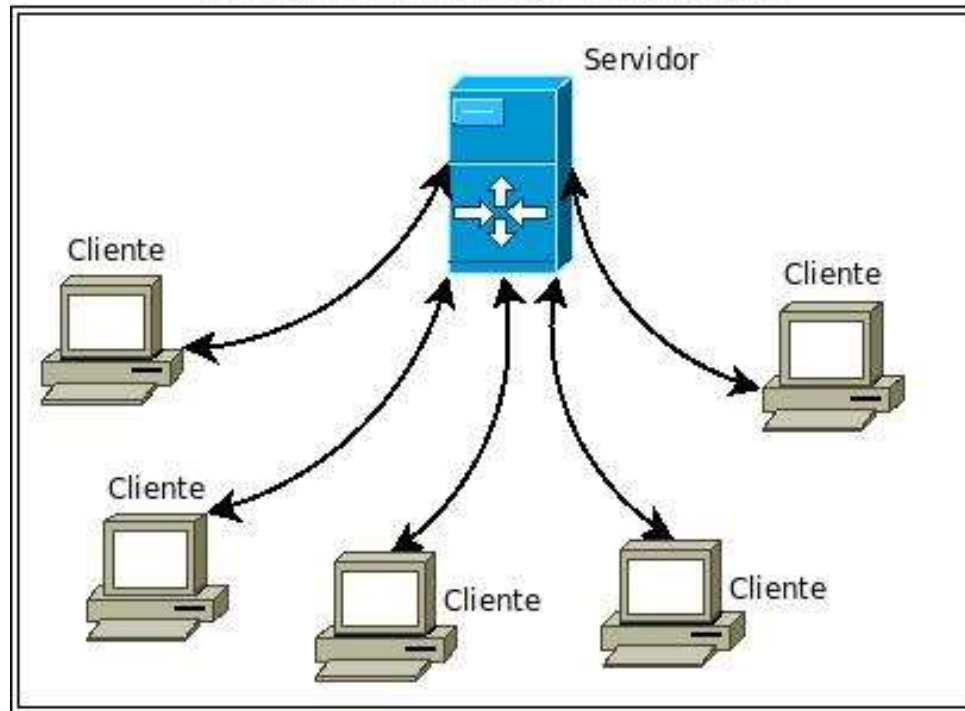


Arquiteturas centralizadas

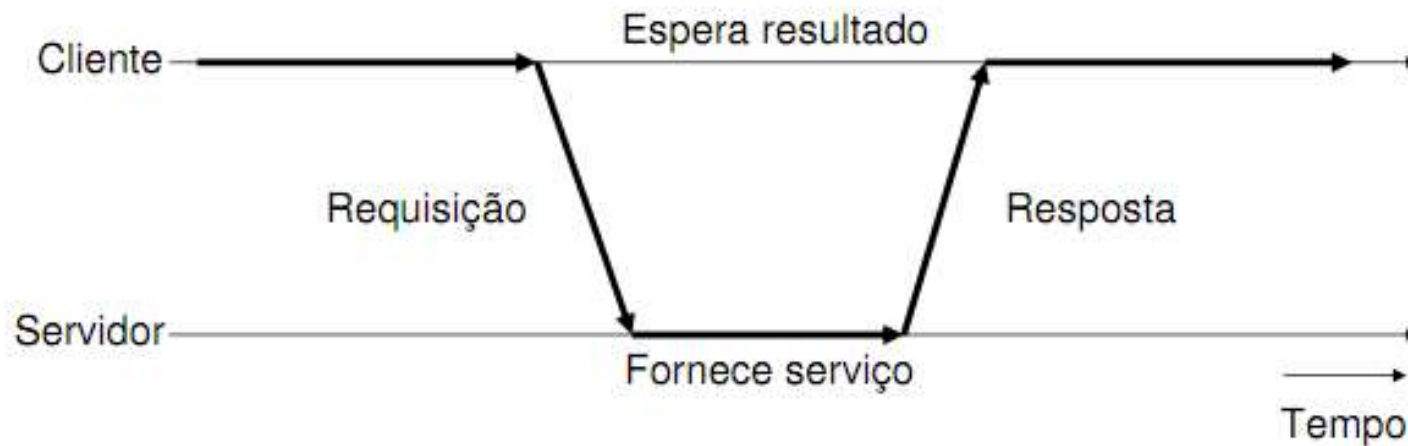
- ▶ **Modelo cliente-servidor;**
 - ▶ Neste modelo, os **processos** são **divididos** em **dois grupos**, com possível sobreposição:
 - **Servidor:** processo que irá **implementar** algum **serviço específico**, tal como um **banco de dados** ou um **sistema de arquivos**;
 - **Cliente:** processo que irá **requisitar** um **serviço** de um servidor através de uma **requisição**, e em seguida aguardando uma **resposta** do servidor.
- 

Arquitecturas centralizadas

Modelo Cliente-Servidor



Arquiteturas centralizadas

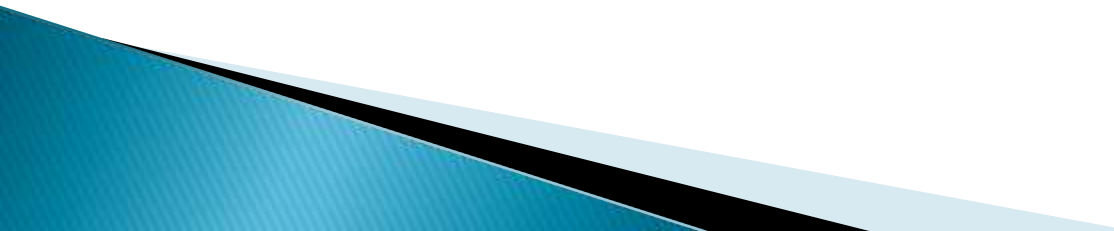


Arquiteturas centralizadas

► Comunicação:

- Protocolo sem conexão, não confiável (UDP):
- Protocolo com conexão, confiável (TCP):

Arquiteturas centralizadas

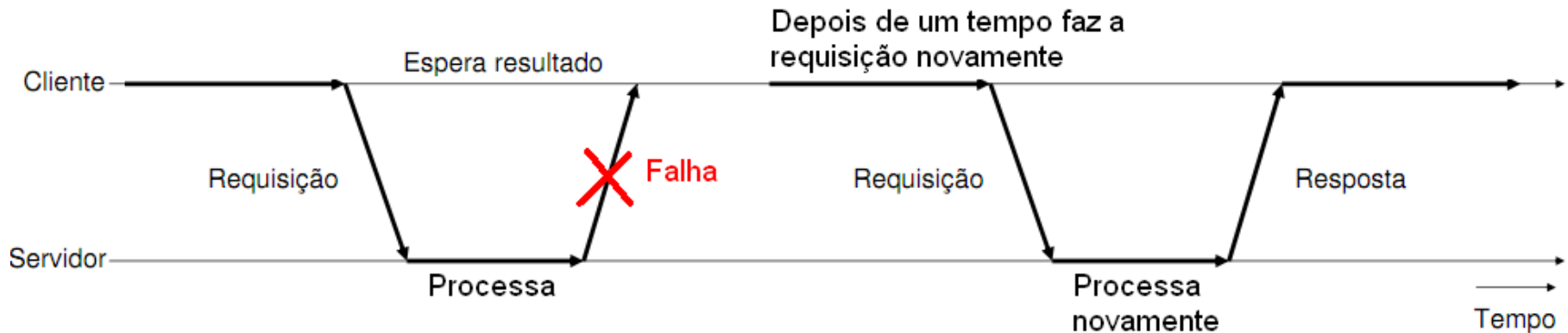
- ▶ Comunicação: protocolo sem conexão, não confiável (UDP):
 - Protocolo mais simples, com um bom funcionamento em redes locais;
 - Quando um cliente for requisitar um serviço, ele simplesmente empacota uma mensagem para o servidor, identificando o serviço desejado juntamente com os dados de entrada necessários;
 - A mensagem é enviada para o servidor;
 - O servidor, que está sempre aguardando alguma solicitação, recebe a mesma e realiza o processamento;
 - Por fim, o servidor empacota os resultados e envia para o cliente.
- 

Arquiteturas centralizadas

- ▶ Comunicação: protocolo sem conexão, não confiável (UDP):
 - É um protocolo **mais eficiente**, desde que se exista uma **garantia** de que não ocorram problemas de **mensagens perdidas ou corrompidas**;
 - Apesar disso, **tornar** o protocolo **livre** de possíveis **falhas** de transmissão **não é** uma tarefa **trivial**;
 - Solução possível e simples para uma falha na transmissão:
 - permitir que o cliente reenvie a transmissão quando ele não receber uma resposta do servidor...
 - Quais os possíveis cenários que surgem ao tomar esta atitude??

Arquiteturas centralizadas

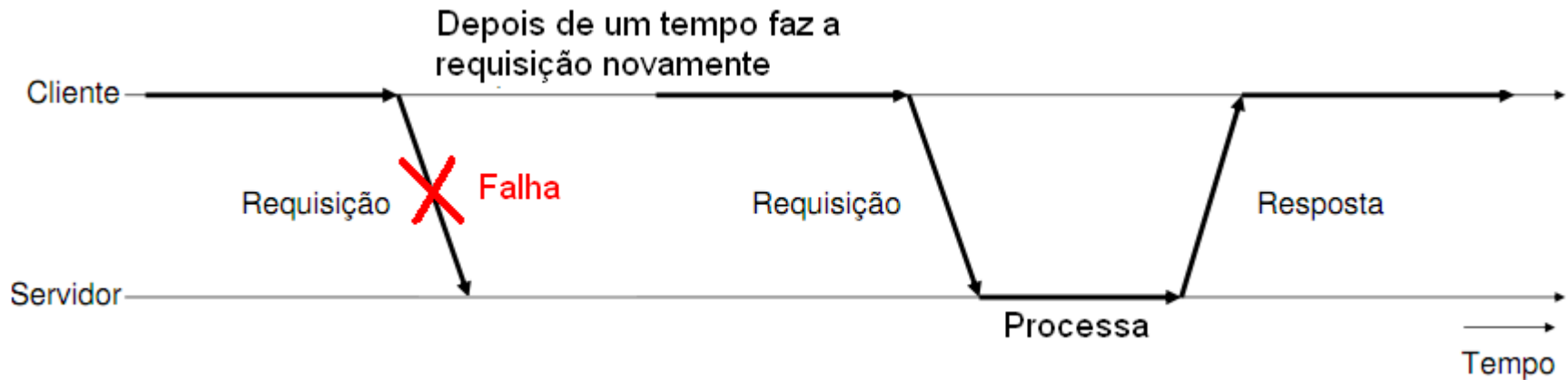
- ▶ Comunicação: protocolo sem conexão, não confiável (UDP):
 - **Caso 1:** A mensagem pode ter chegado ao servidor, e a falha foi na resposta.



- Problema: o servidor irá processar duas vezes!
 - “transfira R\$1.000,00 para a minha conta... Duas vezes!”

Arquiteturas centralizadas

- ▶ Comunicação: Protocolo sem conexão, não confiável (UDP):
 - **Caso 2:** A transmissão da mensagem pode realmente, de fato, ter falhado no seu **envio**.

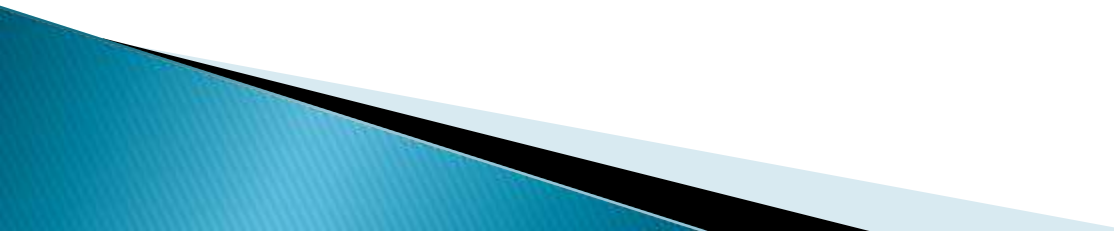


- Para este tipo de problema, a solução funcionaria.

Arquiteturas centralizadas

- ▶ Comunicação: Protocolo sem conexão, não confiável (UDP):
 - Quando uma **operação pode ser repetida diversas vezes** sem causar **nenhum dano** caso ocorra uma **falha**, ela é chamada **IDEMPOTENTE**.
 - Ex: consultar o **saldo de uma conta**, caso a resposta do servidor falhe, efetuar a requisição do saldo novamente não irá afetar em nada, logo, é uma **operação idempotente**.
 - Como algumas operações são idempotentes e outras não, fica claro que **não existe uma solução única** para tratar a perda de mensagens.

Arquiteturas centralizadas

- ▶ Comunicação: protocolo com conexão, confiável (TCP):
 - Não apropriada para redes locais devido a baixa performance;
 - Solução interessante em sistemas de longa distância, onde a comunicação é não confiável;
 - A maioria dos protocolos de aplicação da internet são baseados em TCP/IP confiável;
 - Neste caso, quando um cliente for requisitar um serviço, primeiramente ele irá estabelecer uma conexão com o servidor;
- 

Arquiteturas centralizadas

- ▶ Como distinguir entre cliente e servidor?

Arquiteturas centralizadas

- ▶ Como distinguir entre cliente e servidor?
 - Em alguns casos não há uma clara distinção;
 - Exemplo: Servidor de banco de dados distribuídos;
 - O servidor pode **atuar** também como um **cliente**, pois pode passar **requisições** para **diferentes servidores de arquivos**;
 - Considerando que a maior parte das **aplicações cliente-servidor** são voltadas ao **acesso de usuários a banco de dados**, é feita uma distinção em relação aos três seguintes níveis:
 - Nível de interface do usuário;
 - Nível de processamento;
 - Nível de dados.

Arquiteturas centralizadas

- ▶ **Nível de interface com o usuário:**
 - Contém tudo o que é necessário para **interface** direta com o usuário, de **interação** com a aplicação;
- ▶ **Nível de processamento:**
 - Contém o “**cérebro**” da aplicação;
 - Exemplo: aplicação financeira, que exige métodos e técnicas sofisticados de estatística;
- ▶ **Nível de dados:**
 - **Sistema de arquivos ou banco de dados**, contendo os dados propriamente ditos;
 - Normalmente são implementados no lado do servidor;

Arquiteturas centralizadas

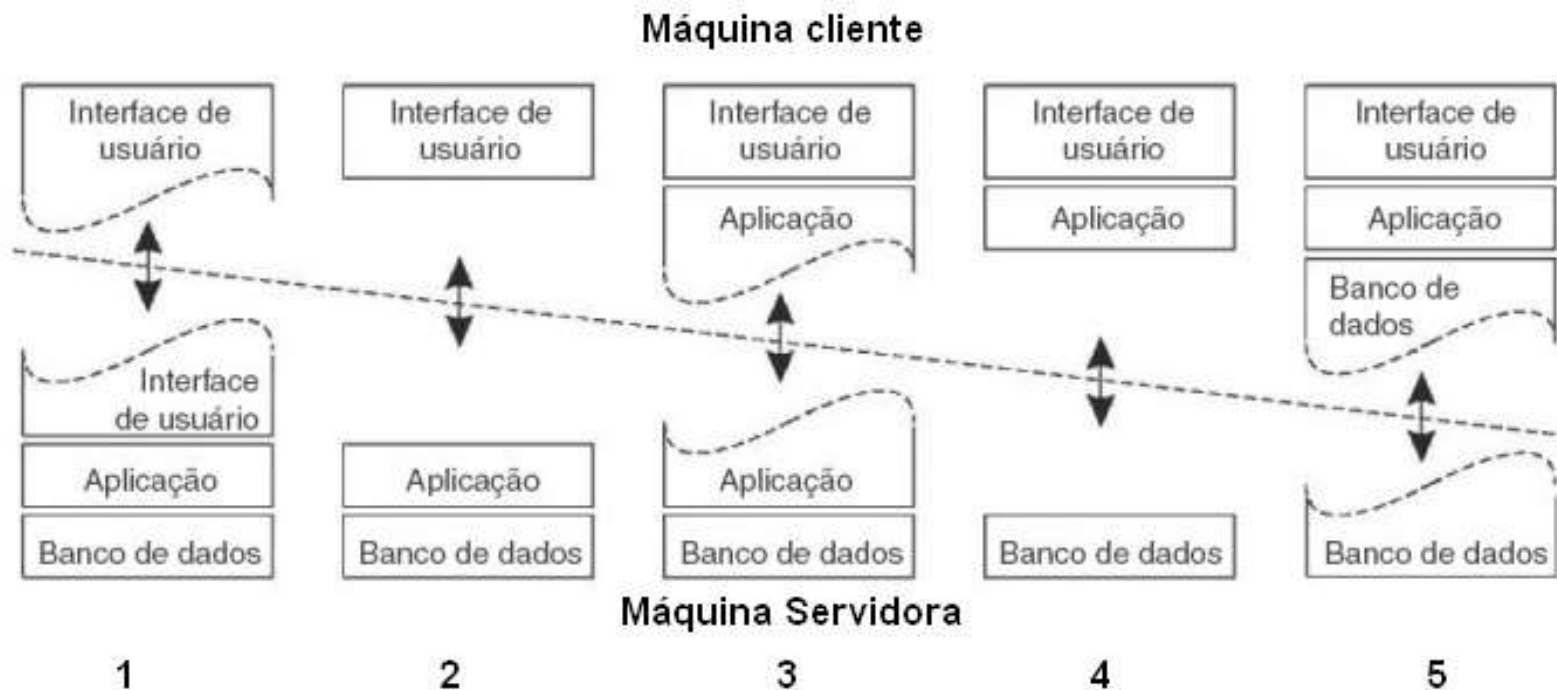
- ▶ Com a distinção entre os **3 níveis lógicos**, surgem **diversas maneiras** de se **distribuir** uma **aplicação** cliente-servidor por várias máquinas;
- ▶ Uma maneira simples de se realizar esta divisão é:
 - A **máquina cliente** contem somente os programas implementando a **interface do usuário**;
 - A **máquina servidora** contem o restante: os programas que implementam o **processamento** e os **dados**;
- ▶ Apesar disso, diversas outras possibilidades podem ser adotadas, tal como:
 - Uma máquina **cliente** que implementa a **interface**;
 - Uma máquina **intermediária** implementando o **processamento**;
 - Outra máquina **servidora** armazenando os **dados**;

Arquiteturas centralizadas

- ▶ Desta forma, através dos 3 níveis lógicos, percebe-se que uma aplicação cliente-servidor pode seguir alguma das seguintes arquiteturas:
 - Arquitetura com duas divisões físicas;
 - Arquitetura com três divisões físicas;

Arquiteturas centralizadas

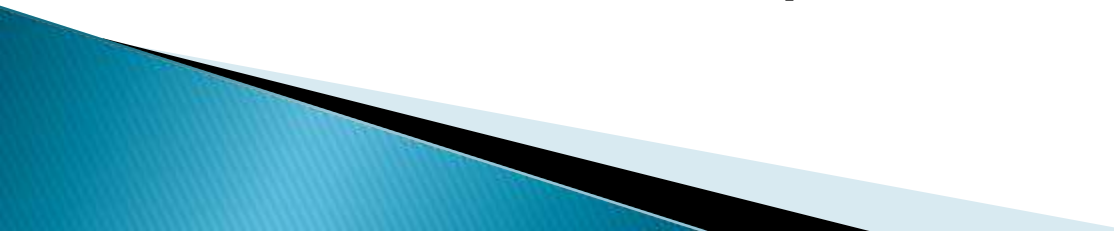
- ▶ Arquitetura com duas divisões físicas;
 - Várias possibilidades:



Arquiteturas centralizadas

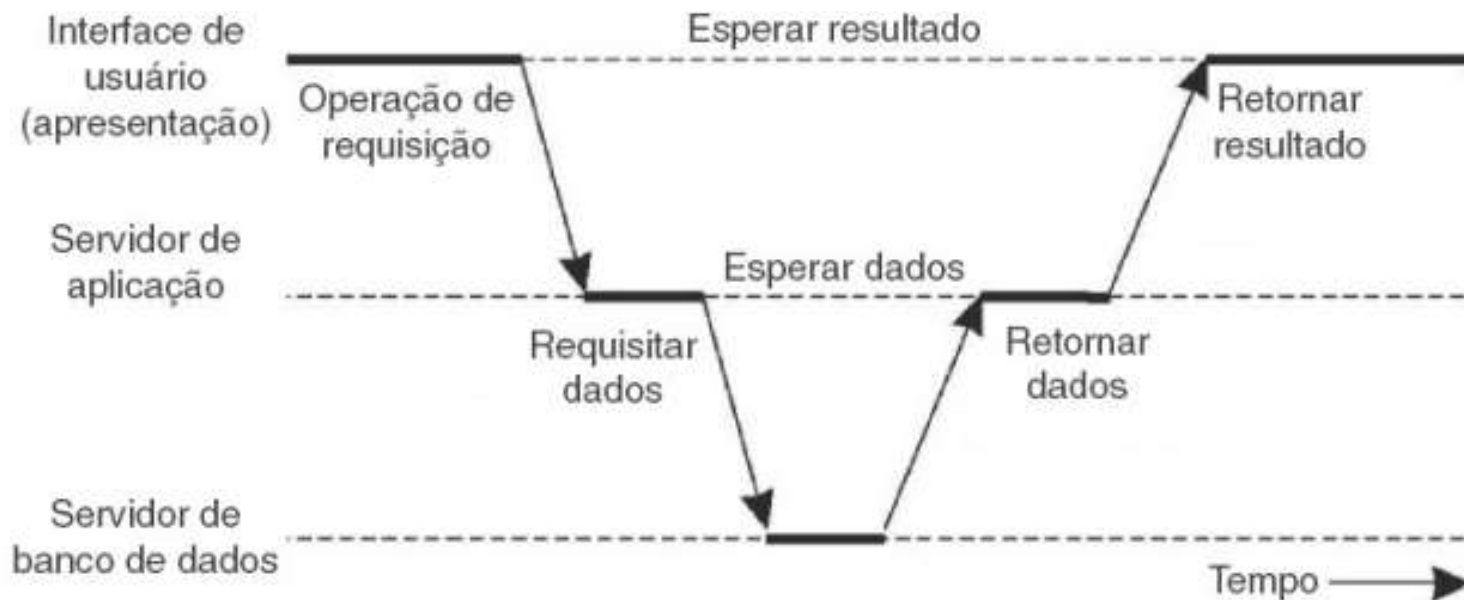
- ▶ Arquitetura com duas divisões físicas:
 - 1 – Uma máquina terminal sendo que o controle da interface é feita por uma máquina remota;
 - 2 – Toda a interface do usuário (inclusive o controle) está no lado cliente. Neste caso, o único processamento que a aplicação cliente realiza é a de apresentação da interface;
 - 3 – Parte da aplicação (processamento) também está no cliente. Ex: **validação de formulário**, que deve estar todo preenchido;
 - 4 – Maioria das aplicações, em que a interface e todo o processamento é feito pela máquina cliente, e o servidor simplesmente armazena os dados;
 - 5 – Representa os casos em que a máquina cliente armazena uma parte dos dados. Ex: cache dos navegadores.

Arquiteturas centralizadas

- ▶ Arquitetura com três divisões físicas:
 - Facilmente nota-se que as **aplicações distribuídas** cada vez **mais se distribuem** em diversas máquinas;
 - Muitas vezes um **servidor atua** também como uma máquina **cliente**, passando **requisições** para **outras máquinas**, e assim por diante;
 - Um exemplo disso são as arquiteturas com três divisões físicas, em que o **processamento** pode estar em uma **máquina** e os **dados** em outra;
- 

Arquiteturas centralizadas

- ▶ Arquitetura com três divisões físicas:



SISTEMAS DISTRIBUÍDOS

Arquiteturas Descentralizadas / P2P

Prof. Guilherme C. Kurtz

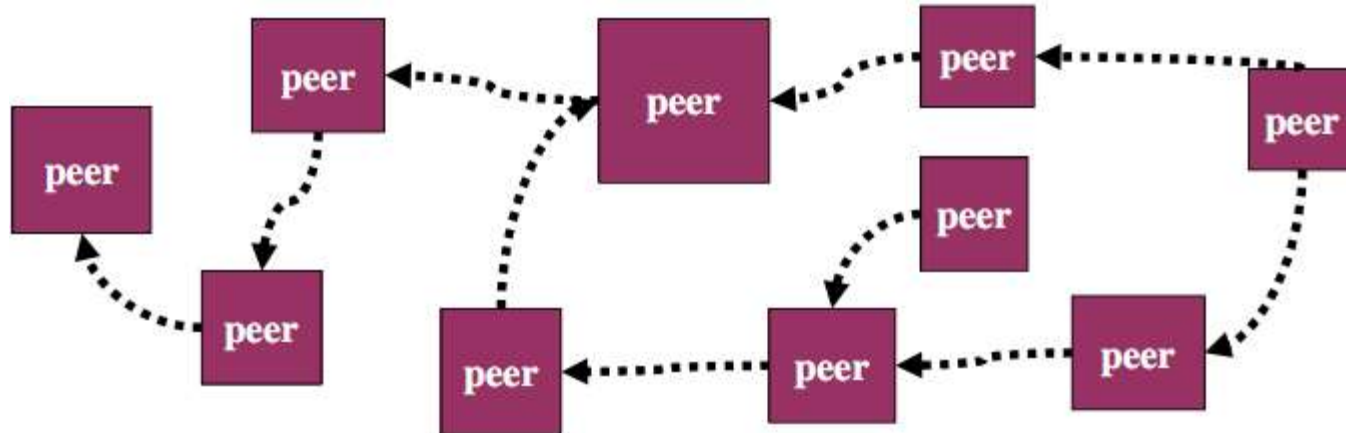


Arquiteturas descentralizadas

- ▶ Também conhecidas como **arquiteturas P2P** (*peer-to-peer*);
- ▶ Distribuição horizontal;
- ▶ **Clientes e servidores** são fisicamente subdivididos em partes logicamente equivalentes;
- ▶ **Carga equilibrada:**
 - Cada parte opera em sua própria porção do conjunto completo de dados;
- ▶ **Interação simétrica** entre os processos (todos iguais), onde cada máquina age como um **SER**vidor e **cliENTE** ao mesmo tempo → **SERVENTS**;

Arquiteturas descentralizadas

- ▶ **Todos os processos tem funcionalidades semelhantes**
 - Ex: compartilhamento de arquivos, edição colaborativa, etc.
- ▶ **Não possui nenhuma estrutura hierárquica ou controle centralizado** (na maioria dos casos);



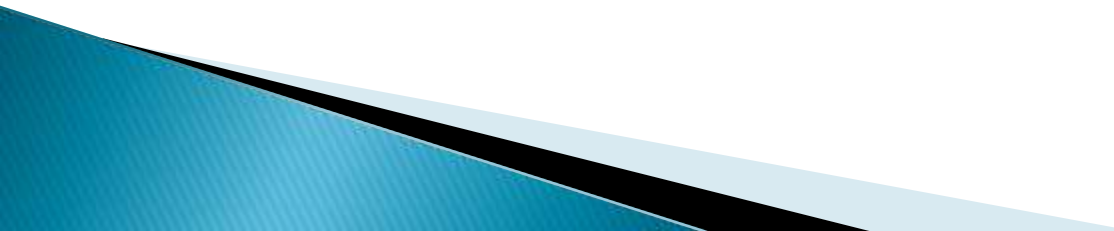
Características

- ▶ Os **peers constroem uma rede virtual** de sobreposição denominada **overlay**:
 - Rede lógica construída sobre a rede física já existente;
 - Os **nós** são os **processos** e os **enlaces** são os **canais de comunicação** existentes;
- ▶ De modo geral (podendo variar em alguns casos):
 - Não há uma **coordenação central**;
 - Não há um **banco de dados central**;
 - Nenhum **peer** tem uma **visão global** do sistema;
 - **Todo o dado** existente pode ser **acessado por qualquer peer**;

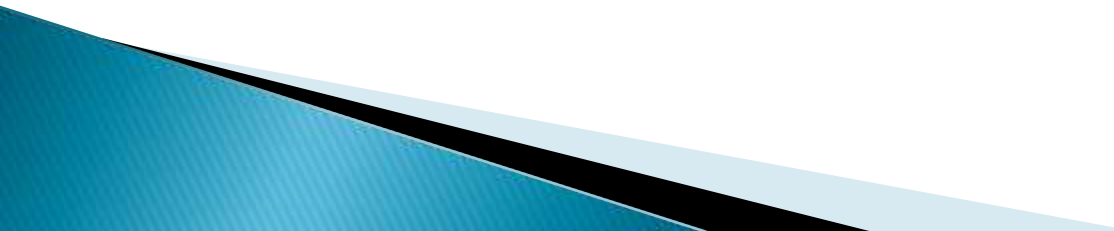
Propriedades

- ▶ **Interação mais complexa:**
 - Implementações mais complexas;
 - Operações de pesquisa são complexas;
- ▶ **Não há ponto único de falha;**
- ▶ **Grande potencial de escalabilidade;**
- ▶ **Redução de custos através do compartilhamento:**
 - Cliente/Servidor: servidor geralmente custa bem mais caro;
 - P2P: o custo é disseminado entre os vários peers;
- ▶ **Aplicado a ambientes em que todos os participantes cooperam para fornecer algum serviço onde:**
 - Capacidade total agregada > capacidade individual;

Propriedades

- ▶ **Anonimato / Privacidade:**
 - Difícil de garantir com um servidor central;
 - Muitos usuários não querem que um servidor saiba de sua participação no sistema;
 - ▶ **Dinamismo:**
 - Recursos (nodos computacionais) entram e saem do sistema continuamente;
 - ▶ **Comunicação Ad-hoc:**
 - Sistemas P2P não contam com uma infraestrutura estabelecida, eles mesmos constroem sua própria;
- 

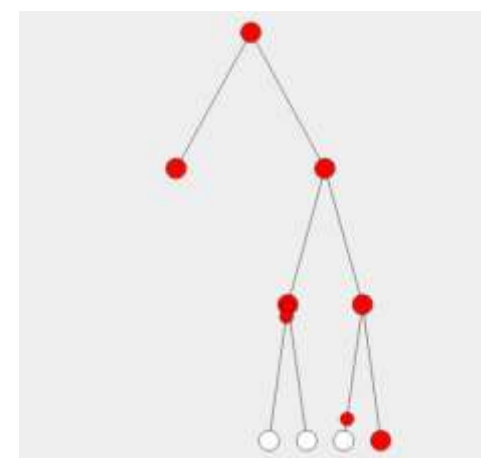
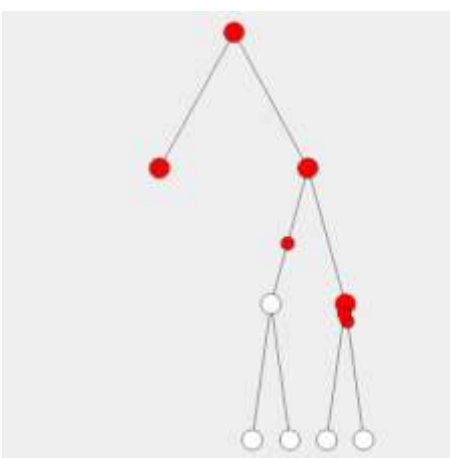
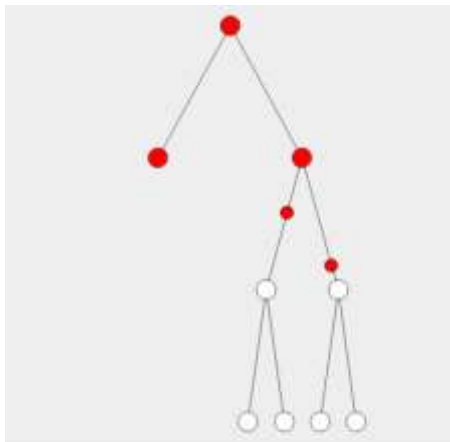
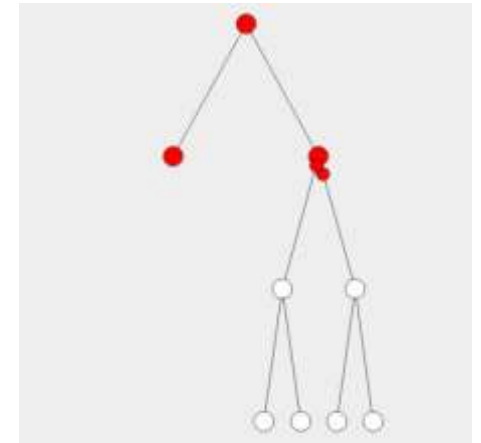
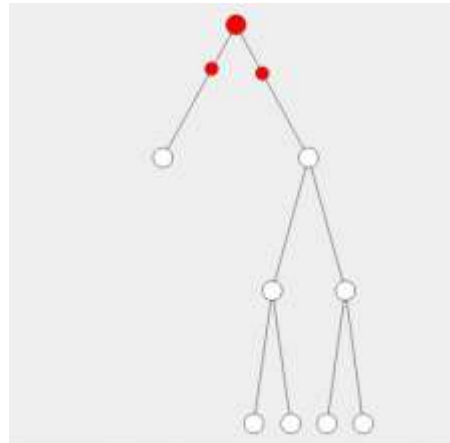
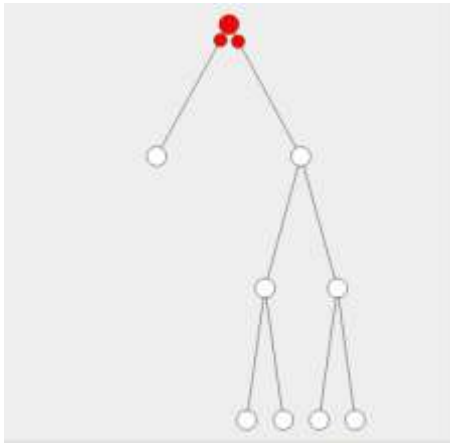
Desafios

- ▶ Como organizar os processos (peers) em uma **rede de sobreposição (overlay)**?
 - ▶ Como difundir o conteúdo?
 - ▶ Como incentivar os peers a **colaborarem**?
 - ▶ Maior quantidade de computadores envolvidos traz alguns problemas:
 - Heterogeneidade;
 - Segurança;
 - ▶ Arquiteturas P2P podem ser **estruturadas e não estruturadas**;
- 

P2P não estruturadas

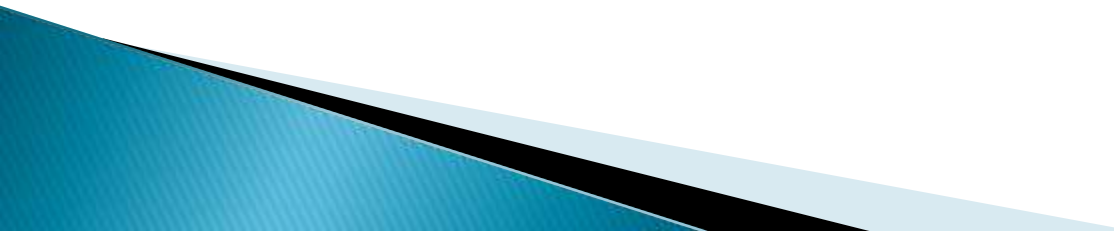
- ▶ **As ligações entre os membros são definidas de forma não-determinista:**
 - Ex: quando um **cliente entrar** em uma **rede**, ele irá escolher um **conjunto de contatos** (sendo que os mesmos podem variar durante a execução do sistema);
- ▶ Mais simples;
- ▶ Problemas:
 - **Pesquisa/Distribuição de informações é pesada**, geralmente através de algoritmos de inundação:
 - Cada nó age como receptor e transmissor, sendo que cada mensagem recebida é transmitida para todos os seus vizinhos;
 - A **latência** e a **escalabilidade** vão **depender** da qualidade do **grafo** formado entre os peers;

P2P não estruturadas



Algoritmo de inundação para distribuição de informação entre os nodos

P2P estruturadas

- ▶ Topologia de rede é rigidamente controlada e os arquivos são colocados precisamente nos locais especificados;
 - ▶ A rede **overlay** construída a partir de procedimentos determinísticos;
 - ▶ Provê um mapeamento entre o **identificador do arquivo** e sua **localização**;
- 

P2P estruturadas

- ▶ Tabela Hash distribuída (Distributed Hash Table – DHT):
 - **Itens de dados** recebem uma **chave aleatória**, como um identificador de 128 ou 160 bits;
 - Da mesma forma, os **nós** também recebem um **número aleatório de identificação**;
 - O sistema deve implementar um **esquema determinístico** eficiente que **mapeie** a **chave** de um **item de dado** para o **identificador** de um nó;
 - O mais importante é que, ao consultar um item de dado, o endereço de rede do nó responsável por aquele item de dado é retornado.
 - Consegue-se isso *roteando* uma requisição para um item de dado até o nó responsável.

Variações de P2P

- ▶ Sistemas P2P podem ter algumas variações, dentre elas:
 - P2P Centralizadas;
 - P2P Descentralizadas;
 - P2P Híbridas;

P2P Centralizadas

- ▶ Um servidor central (podendo ser virtual) controla a **interação entre os peers**;
 - Por existir tal interação que ainda é considerada P2P!
- ▶ O servidor controla a pesquisa e define os identificadores dos nós da rede;

Vantagens

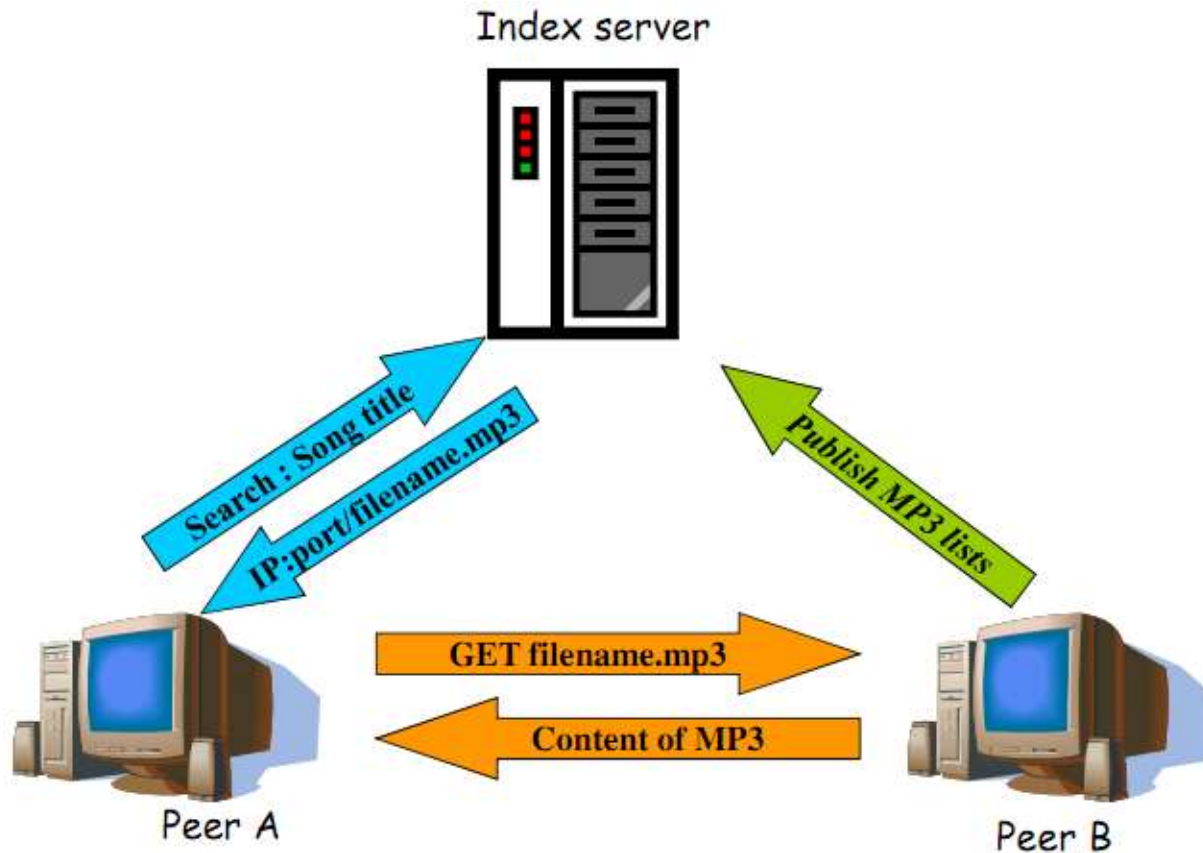
Mais simples de implementar e controlar;
Buscas são mais eficientes;

Desvantagens

Ponto único de falha;
Alto custo de centralização do serviço;
Escalabilidade limitada;
Mais vulnerável: ataques DoS
Gargalo na performance/escalabilidade

P2P Centralizadas

► Exemplo: Napster



P2P Descentralizadas

- ▶ P2P propriamente ditas, com nodos assumindo o papel de servidores e clientes (SERVENTS)
- ▶ Não há um servidor de coordenação central;

Vantagens

Não há necessidade de manutenção

Difícil de ser desligada

Privacidade

Escalabilidade

Robustez

Desvantagens

Consistência dos dados

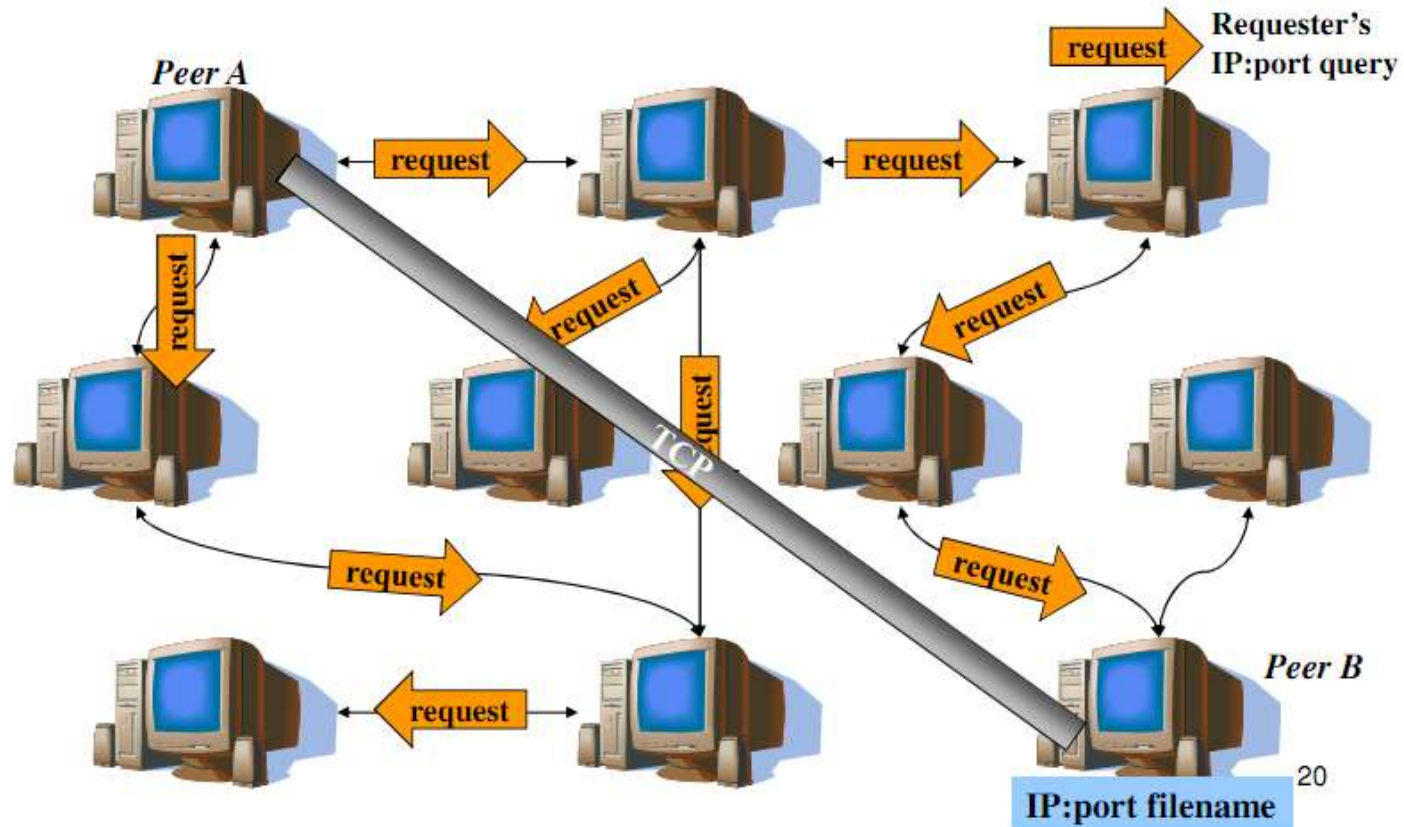
Busca ineficiente;

Gargalos espalhados (peers limitados)

Overhead de comunicação

P2P Descentralizadas

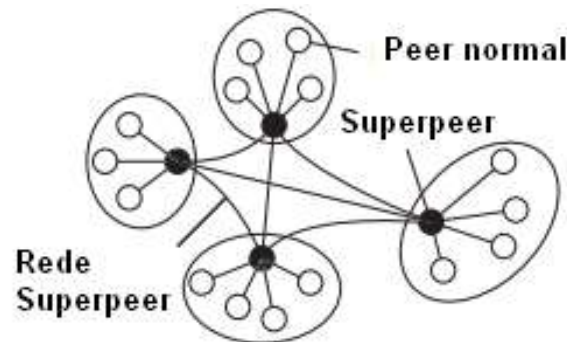
► Exemplo: Gnutella



P2P híbridas

► Utilização de Superpares (Superpeers):

- A medida que uma **rede não estruturada** cresce, pode se tornar **difícil a localização de itens de dados**, visto que não há um modo **determinístico** de se rotear uma mensagem;
- Desta forma, alguns **sistemas** mantêm **nós especiais** que armazenam **índices de dados**, denominados **superpares**;
- A utilização de superpares em uma rede P2P resulta em uma **organização hierárquica**;
- Ex: Kazaa, Emule, Bittorrent, etc.



Bittorrent

- ▶ BitTorrent é um sistema P2P para **transferência de arquivos**;
- ▶ A ideia básica de que quando um usuário estiver procurando por um arquivo, ele **“puxe” partes deste arquivos** de outros **usuários** até que todas as partes sejam **baixadas** e então **montadas**, resultando no **arquivo final** completo.
- ▶ Utiliza um protocolo para desencorajar os “caroneiros”:
 - Um arquivo poderá ser transferido se o cliente também estiver transferindo seu conteúdo com mais alguém;
 - Os nós que disponibilizarem uma alta velocidade de upload poderão obter uma alta velocidade de download;
 - A taxa de download de um nó será reduzida caso a velocidade de upload tiver sido limitada.

Bittorrent

► Funcionamento:

- Primeiramente, o cliente deve efetuar o download de um arquivo .torrent em algum website;
- Um arquivo .torrent contém:
 - Informações necessárias para sobre o arquivo:
 - Nome, tamanho, etc.;
 - Ele referencia um rastreador (*tracker*) que mantém uma contabilidade dos nós ativos que tem as porções do arquivo requisitado;
 - Nó ativo é aquele que está transferindo partes do arquivo em questão;
 - Quando um nó identifica as porções que ele deseja obter, ele torna-se ativo, e passa a auxiliar os outros.