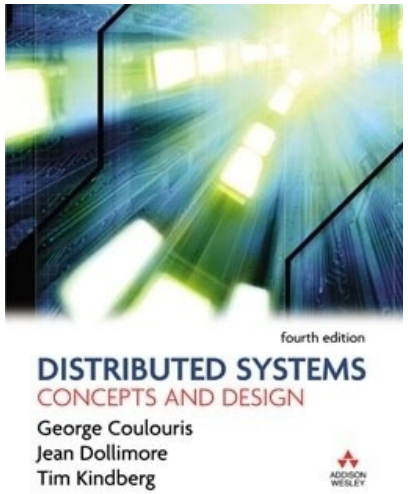
Slides for Chapter 2:
Modelos de Sistema



From Coulouris, Dollimore and Kindberg
Distributed Systems:
Concepts and Design

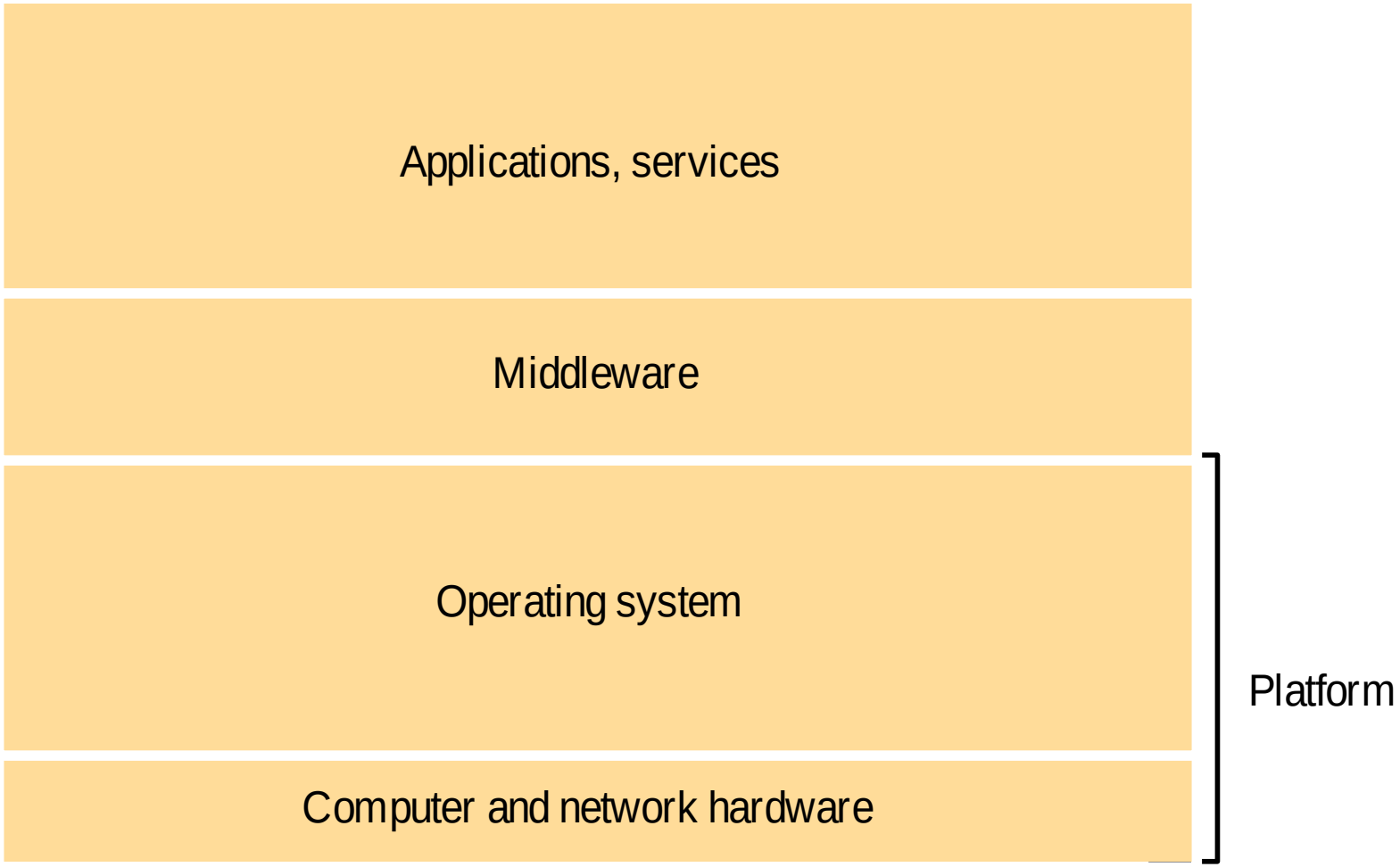
Para que Servem os Modelos?

- Capturam a “essência” de um sistema, permitindo que aspectos importantes do seu comportamento possam ser mais facilmente estudados e analisados pelos seus projetistas
  - Quais são os principais elementos do sistema?
  - Como eles se relacionam?
  - Que características afetam o seu comportamento (individual e colaborativo)?
- Ajudam a:
  - Tornar explícitas todas as pré-suposições sobre o comportamento esperado do sistema
  - Fazer generalizações sobre o que deve e não deve acontecer com o sistema, dadas essas pré-suposições

Tipos de Modelos

- Modelos arquitetônicos descrevem a estrutura organizacional dos componentes do sistema
  - Como interagem uns com os outros
  - Como são mapeados para a infra-estrutura física (rede) subjacente
- Modelos fundamentais descrevem problemas e características chaves comuns ao projeto de todos os tipos de sistemas distribuídos
  - Mecanismos de interação e comunicação
  - Tratamento de falhas
  - Segurança

Modelos arquitetônicos - Modelo em camadas



Modelos arquitetônicos - Modelo em camadas

- Plataforma:
  - Camadas de *hardware* e *software*
  - Fornecem serviços para as camadas que estão acima
- Objetivo: levar a interface de programação do sistema a um nível que facilita a comunicação e a coordenação entre os processos
- Exemplos:
  - Intel x86/Windows,
  - Intel x86/Linux,
  - PowerPC/Mac OS

Modelo em camadas

- *Middleware*
  - Camada de software que tem por objetivo **mascarar a heterogeneidade** e fornecer um modelo de programação conveniente para os programadores dos aplicativos
  - Composto por um conjunto de processos ou objetos, em um grupo de computadores, que interagem entre si de forma a **implementar a comunicação** e oferecer suporte para **compartilhamento de recursos** a aplicativos distribuídos

Middleware

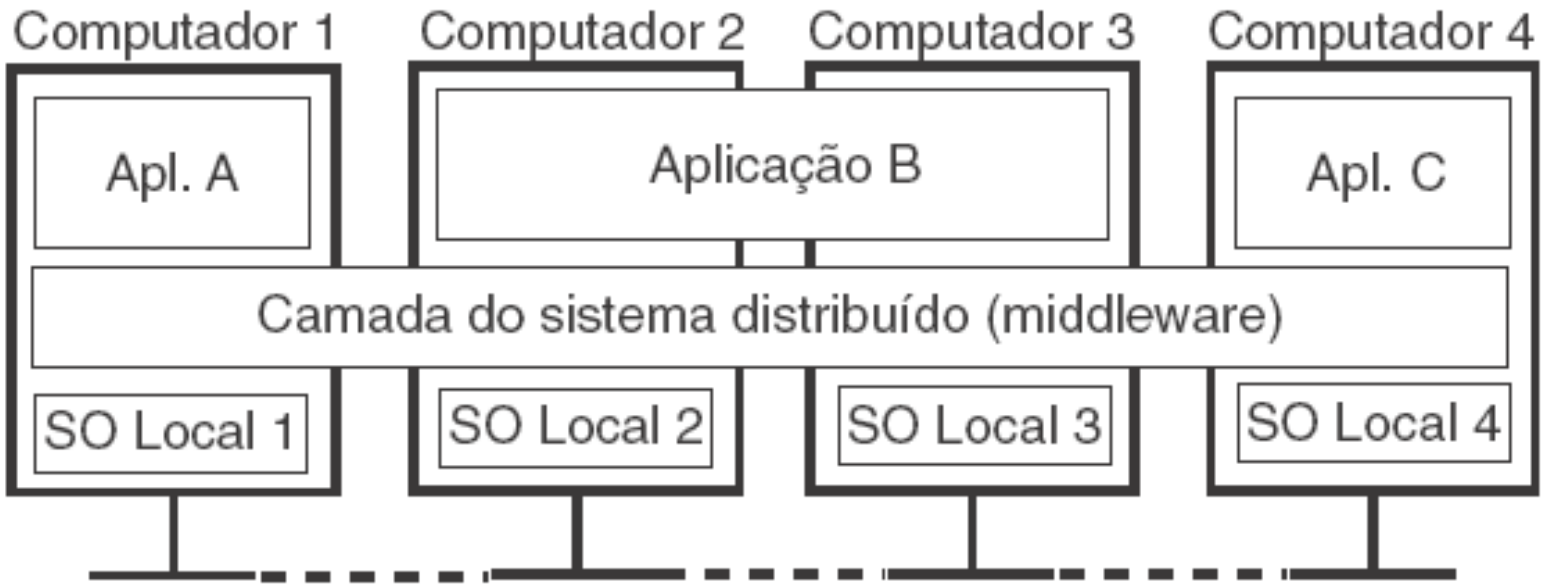


Figura 1.1 Sistema distribuído organizado como middleware. A camada de middleware se estende por várias máquinas e oferece a mesma interface a cada aplicação.

Modelo em camadas

- *Middleware*
  - Serviços
    - Invocação a métodos remotos
    - Comunicação entre um grupo de processos
    - Notificação de eventos
    - Replicação
    - Transações
  - Ex. de *Middlewares* OO
    - CORBA
    - Java RMI
    - Web Services
    - DCOM (*Disctributed Component Object Model*) da Microsoft

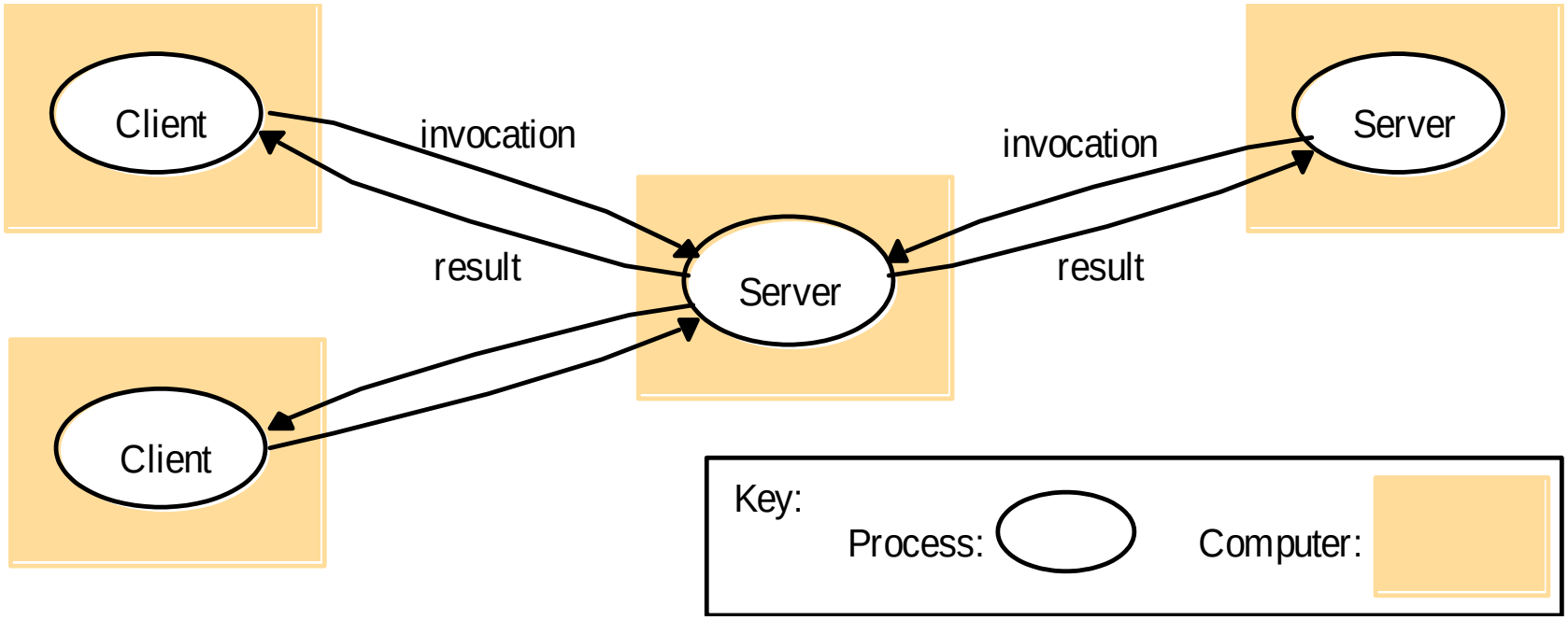
Modelos Arquitetônicos – Arquiteturas de Sistemas

- Foco na arquitetura – estrutura de alto nível do sistema, descrita em termos de **componentes** e seus **relacionamentos**
  - Base para garantir que a estrutura do sistema atenderá sua atual e provável futura demanda em termos de atributos de qualidade como **confiabilidade**, **adaptabilidade**, **desempenho**, **gerência**, etc.
- Descrição simplificada e abstrata dos componentes do sistema:
  - Funcionalidades (ou responsabilidades)
    - Ex.: servidor, cliente, peer
  - Distribuição física (recursos e carga de trabalho)
    - Ex.: regras de particionamento e/ou replicação
  - Padrões de interação e comunicação
    - Ex.: cliente-servidor, ponto-a-ponto

Estilo Cliente-Servidor

- Divisão das responsabilidades entre os componentes do sistema de acordo com dois papéis bem definidos:
  - Clientes
  - Servidores
- **Servidores** são responsáveis por gerenciar e controlar o acesso aos recursos mantidos no sistema
- **Clientes** interagem com servidores de modo a terem acesso aos recursos que estes gerenciam
- Alguns servidores podem assumir o papel de clientes de outros servidores
  - Ex.: Servidor web no papel de cliente de um servidor de nomes
- Continua sendo o modelo de sistema distribuído **mais estudado e utilizado na prática**!

Figure 2.2
Clients invoke individual servers



Camadas de Aplicação

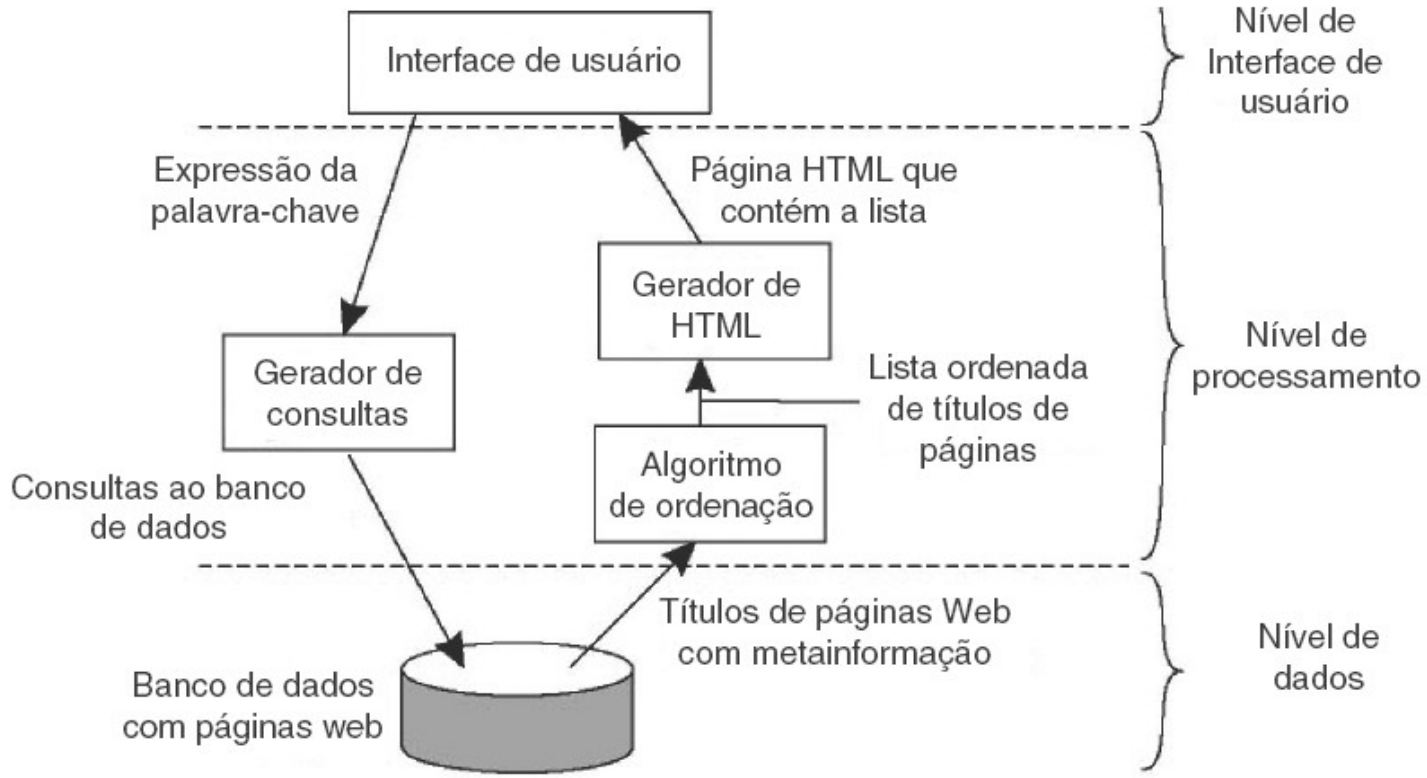
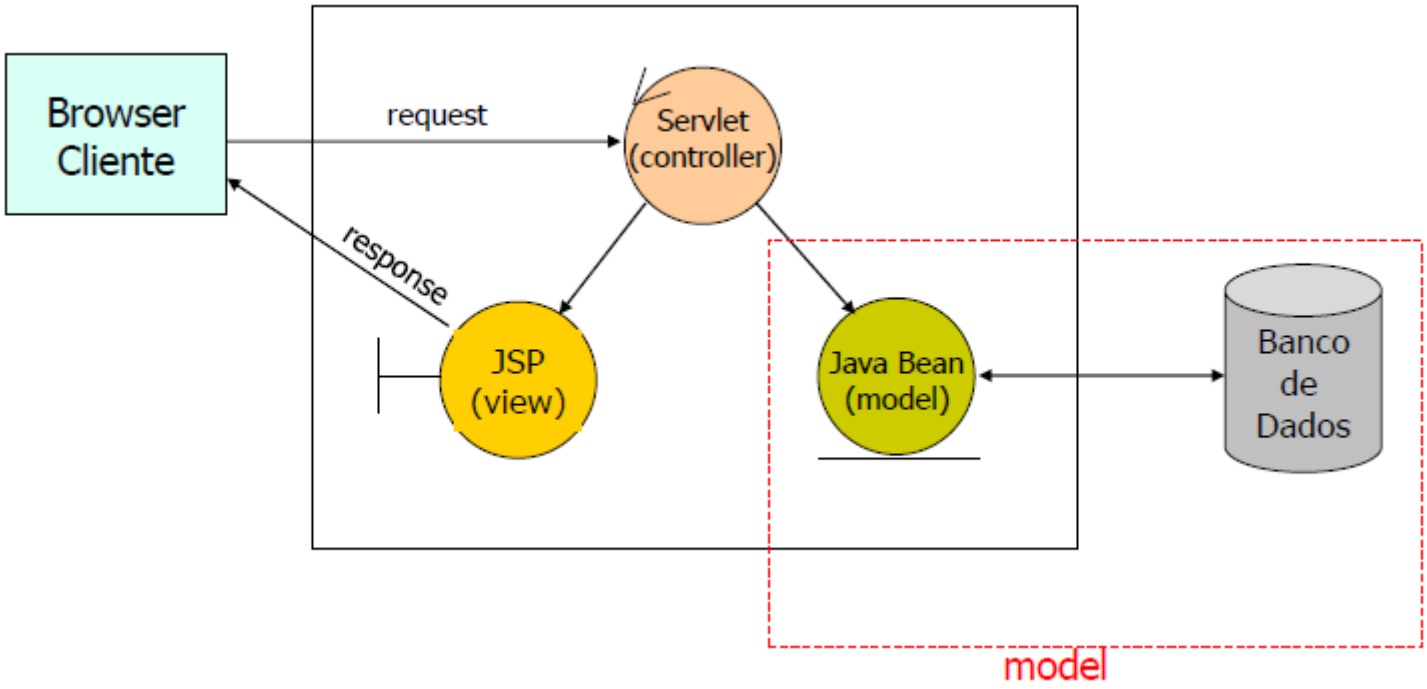


Figura 2.4 Organização simplificada de um mecanismo de busca da Internet em três camadas diferentes.

Arquitetura - MVC



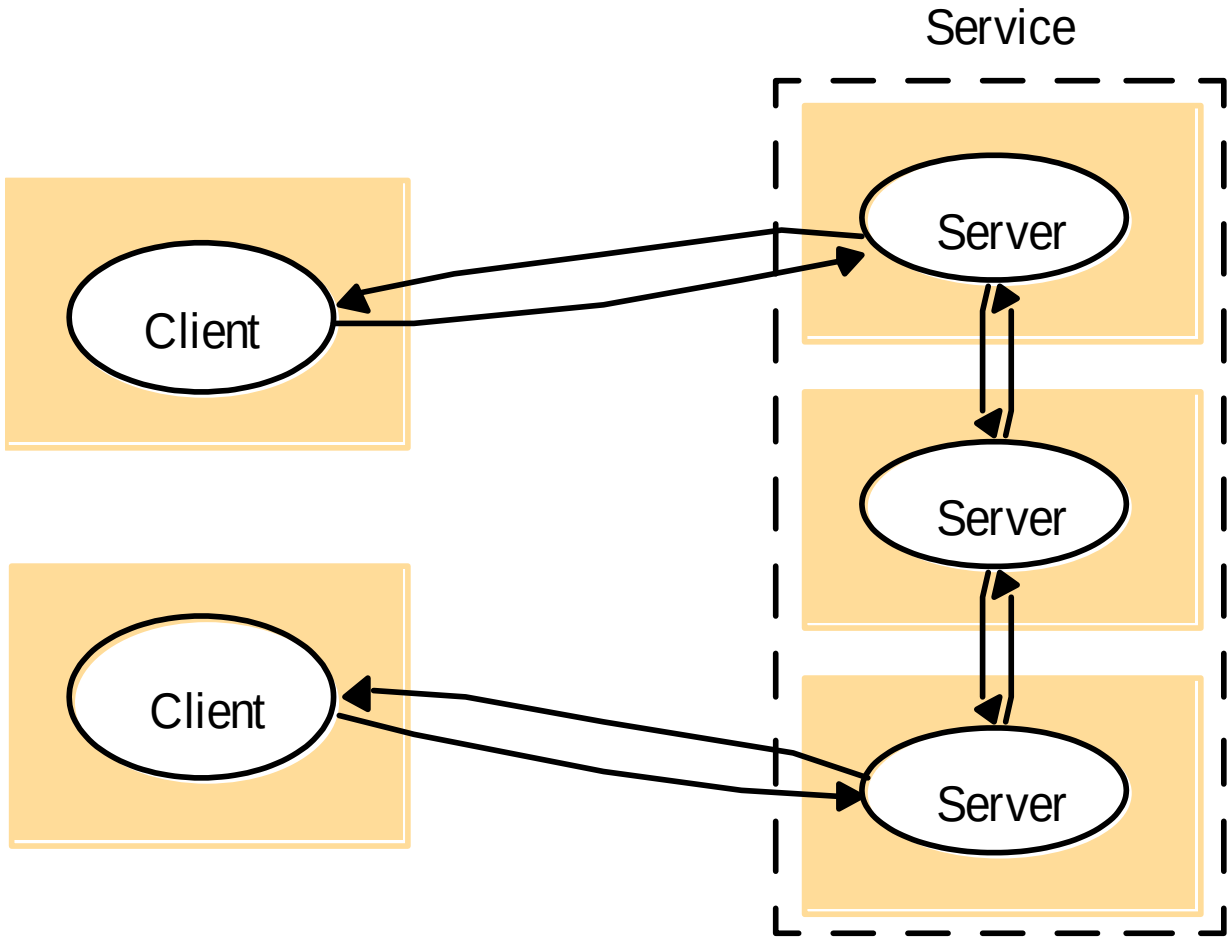
Estilo Cliente-Servidor

- Sete variações do estilo Cliente-Servidor mais estudadas:
  - Múltiplos servidores por serviço
  - Cache e servidores proxy
  - Clientes magros
  - Código móvel
  - Agentes móveis
  - Objetos distribuídos
  - Dispositivos móveis

CS com Múltiplos Servidores

- Cada serviço é implementado por um conjunto de servidores, possivelmente localizados em diferentes pontos da rede
- Servidores podem interagir entre si para oferecer uma visão global consistente do serviço para os clientes
- Técnicas mais utilizadas:
  - Particionamento – distribuição física dos recursos entre os vários servidores
    - Maior facilidade de gerência e maior escalabilidade
    - Ex.: Clusters de servidores do portal UOL
  - Replicação – manutenção de cópias do mesmo recurso lógico em dois ou mais servidores
    - Maior desempenho e disponibilidade
    - Ex.: Base de dados do Google

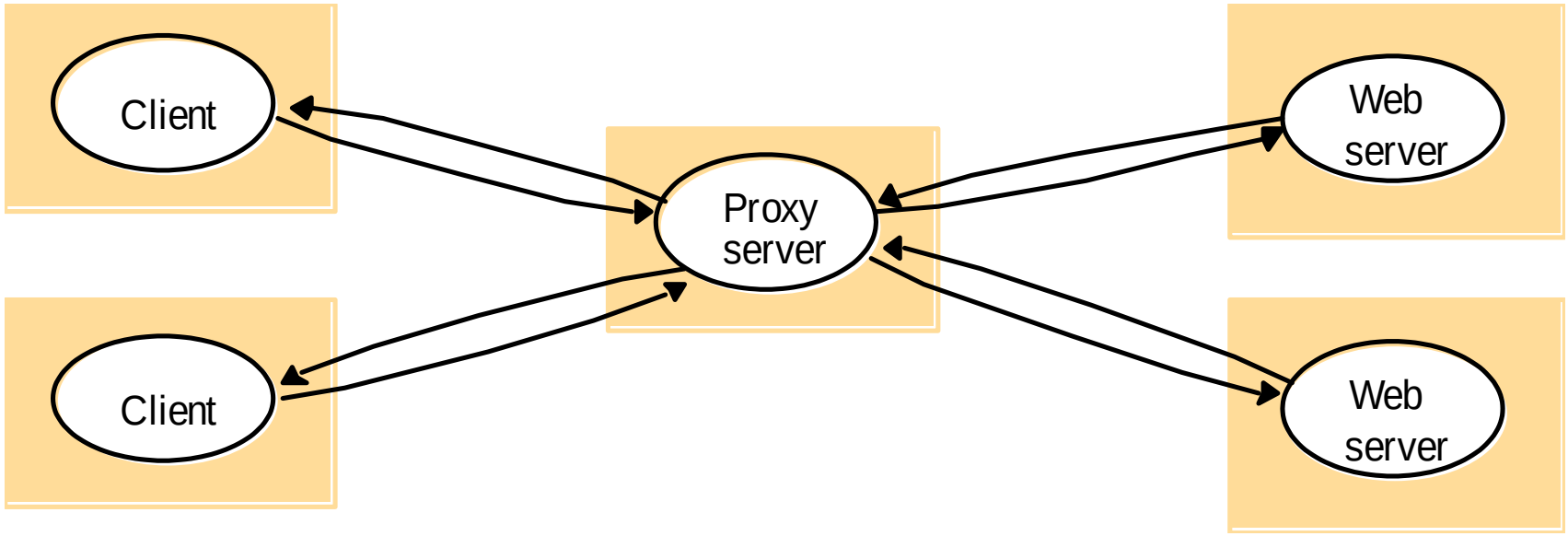
Figure 2.4
A service provided by multiple servers



CS com Cache e Servidor Proxy

- Cache
  - Repositório de cópias de objetos recentemente utilizados que está fisicamente mais próximo do que os objetos originais
  - Principais desafios:
    - Política de atualização (controla a entrada e saída de objetos no cache)
  - Localização física (nos clientes ou em um ou mais servidores proxy)
- Servidor proxy
  - Processo compartilhado por vários clientes que serve como cache para os recursos disponibilizados por outros servidores remotos
  - Principais funções:
    - Reduzir o tempo de acesso
    - Aumentar a disponibilidade
    - Também utilizado para proteção, filtragem, adaptação, etc.

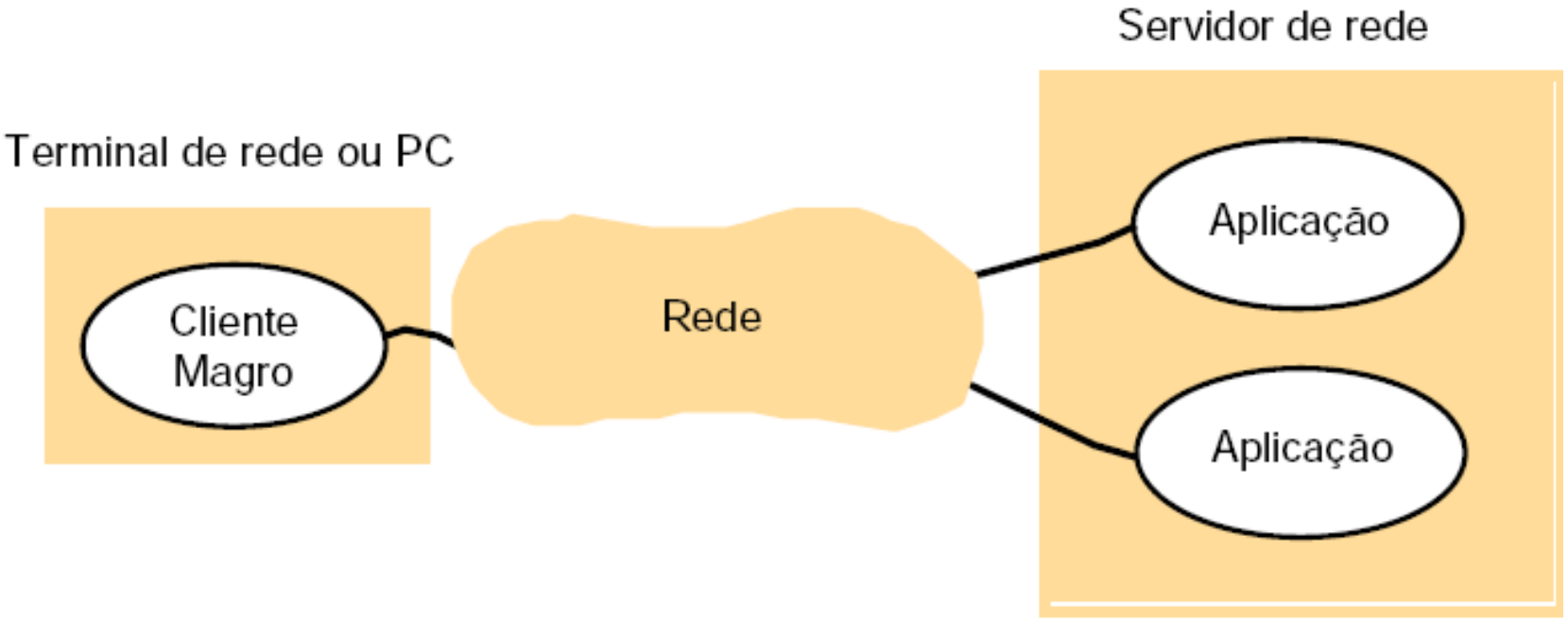
Figure 2.5
Web proxy server



CS com Clientes Magros

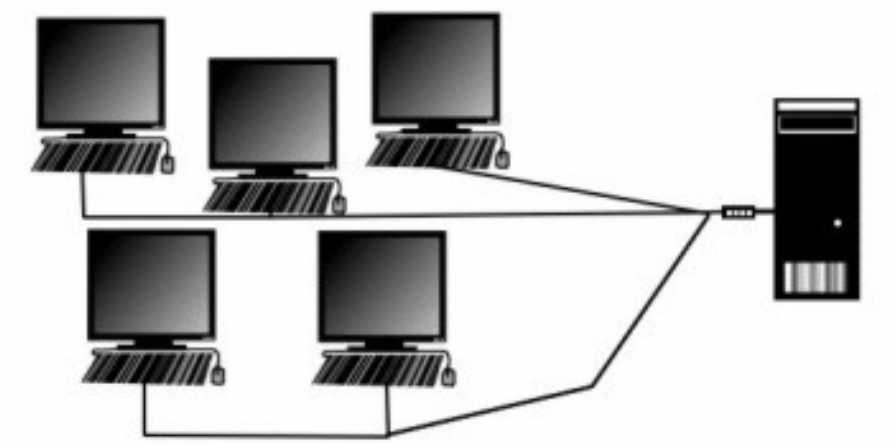
- Cliente magro
  - Camada de software com suporte para interação local com o usuário, e que executa aplicações e solicita serviços exclusivamente a partir de servidores remotos
    - VNC (*Virtual Network Computing*),
    - LTSP (*Linux Terminal Server Project*)
  - A favor:
    - Baixo custo de hardware e software para os clientes
    - Maior facilidade de gerência e manutenção das aplicações
  - Contra:
    - Alto custo de hardware e software para os servidores
    - Centralização da carga de trabalho e do tráfego de mensagens
    - Risco de sobrecarga dos servidores e/ou da rede
    - Baixo desempenho para aplicações altamente interativas

CS com Clientes Magros

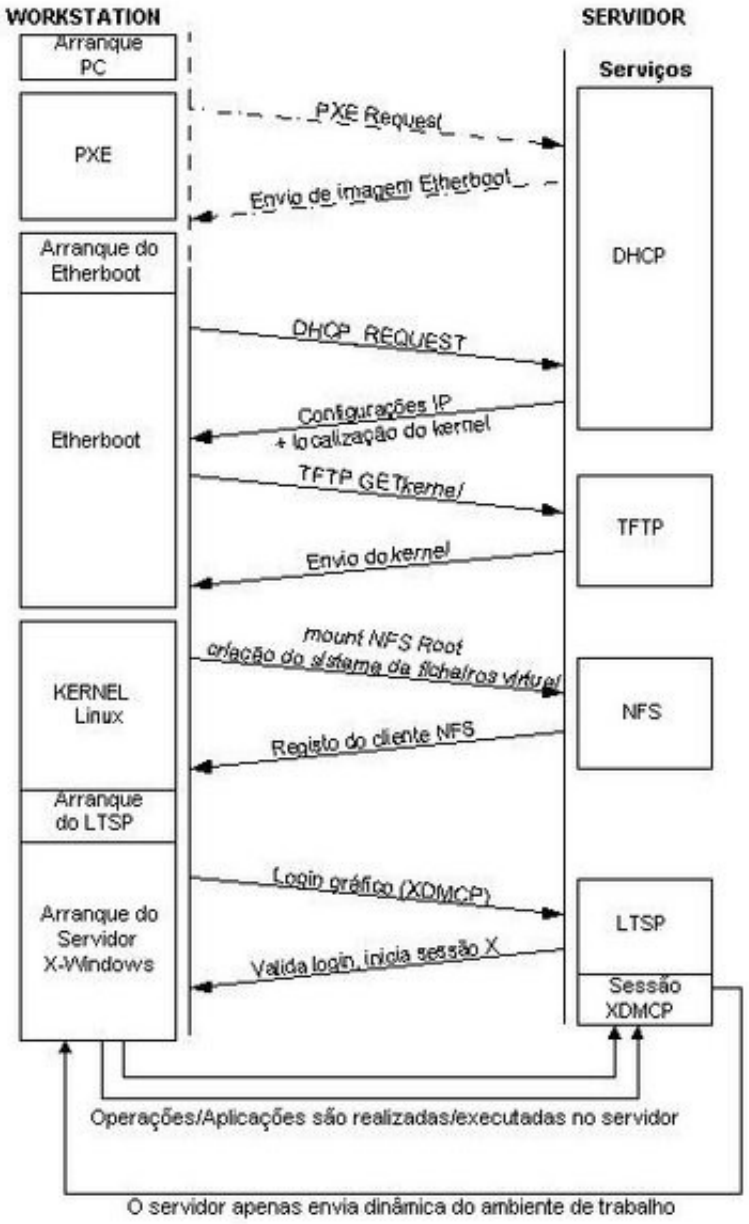


LTSP (*Linux Terminal Server Project*)

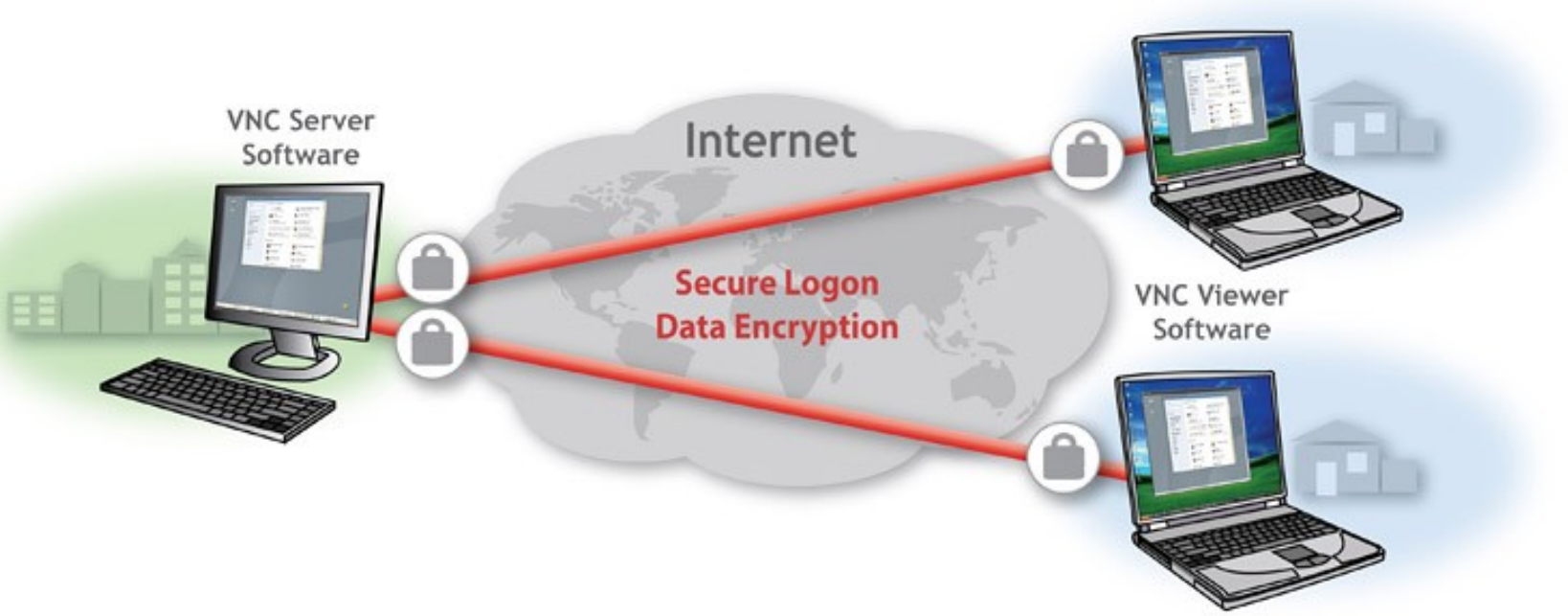
- Utiliza uma combinação de DHCP, TFTP e NFS para permitir que as estações não apenas rodem aplicativos instados no servidor, mas realmente dêem boot via rede, baixando todos os softwares de que precisam diretamente do servidor.



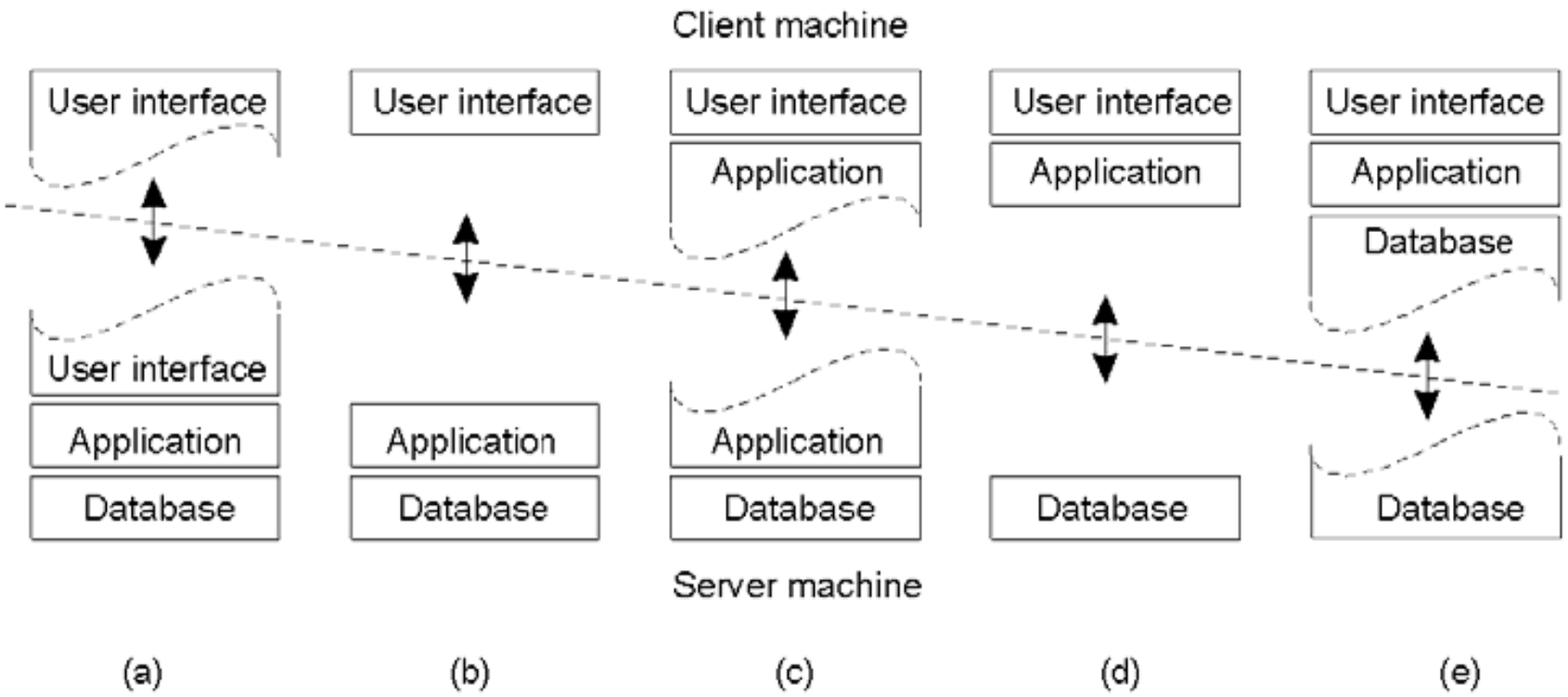
LTSP-BOOT



VNC *(Virtual Network Computing)*



Clientes Magros e Seus Níveis



CS com Código Móvel

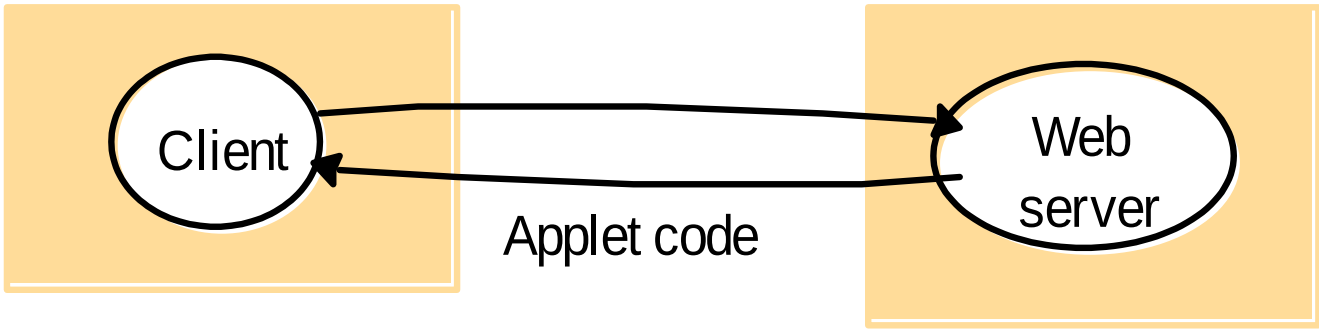
- Serviços oferecidos na forma de um código (programa) específico que deve ser descarregado do servidor
  - Aplicações clientes executam e interagem localmente com o código móvel recebido
  - Dependendo do serviço, código móvel pode interagir com um ou mais servidores em nome da aplicação cliente
  - Ex.: Java applets
- Principais benefícios:
  - Redução do tempo de resposta para aplicações interativas
  - Maior facilidade de customização e atualização da interface de acesso ao serviço
  - Possibilidade de estender dinamicamente as funcionalidades das aplicações clientes

CS com Código Móvel

- Navegadores impedem que o Código Móvel:
  - Acesse arquivos locais
  - Impressoras
  - Sockets de Rede
- JVM
  - Classes carregadas por download são armazenadas separadamente das classes locais, impedindo sobrescrita por classes maliciosas
  - É verificada a validade dos *bytecodes.*
    - Composto de instruções de um conjunto específico?
    - Acessa endereços de memória inválidos?

CS com Código Móvel

a) client request results in the downloading of applet code



b) client interacts with the applet

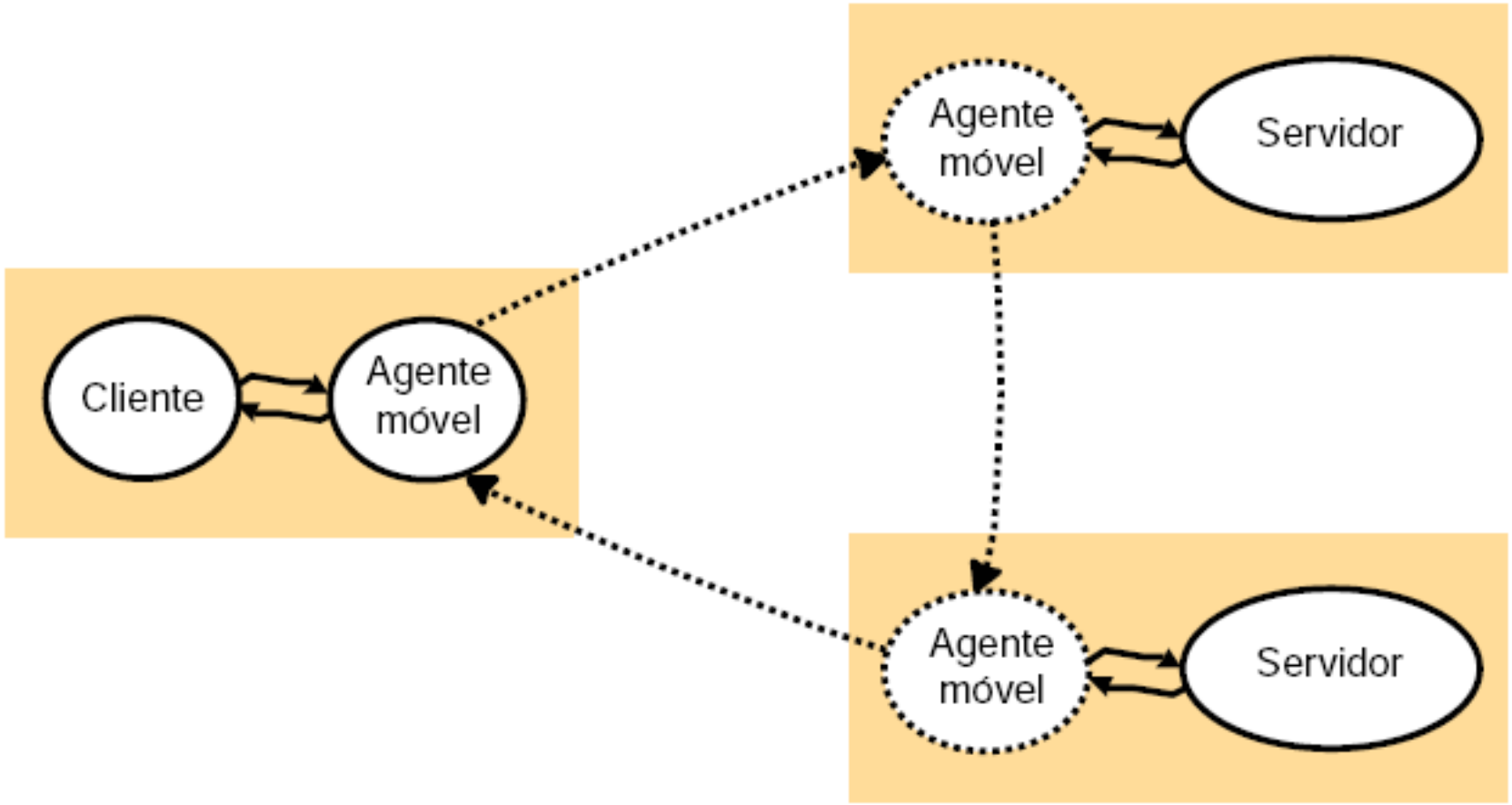
CS com Código Móvel

- Desafios de projeto:
  - Heterogeneidade do código móvel e da arquitetura de execução das aplicações clientes
    - Solução: máquinas virtuais padronizadas embutidas nas aplicações clientes
  - Riscos de segurança na execução do código móvel
    - Solução: limitar as ações do código móvel ou executá-lo em um ambiente isolado do restante da rede
  - Atrasos causados pelo tempo de transferência do código móvel e pelo tempo de inicialização do seu ambiente de execução
    - Solução: transferência do código em formato compactado; cache de códigos recentemente utilizados; pré-inicialização do ambiente de execução

CS com Agentes Móveis

- Agente móvel
  - Programas em execução (código + dados) que circula pela rede solicitando serviços em nome de um usuário ou de uma aplicação cliente
    - Ex.: agente para coleta de dados, busca e comparação de preços de produtos, instalação de software, etc
  - O acesso aos serviços é feito localmente pelo agente, ou de locais fisicamente próximos (da mesma rede local) aos servidores
- Benefícios:
  - Redução dos custos e do tempo de acesso
    - Acesso antes remoto agora passa a ser local
  - Maior tolerância a falhas de comunicação
    - Conexão necessária apenas durante a transferência do agente
  - Melhor distribuição do tráfego de mensagens na rede

CS com Agentes Móveis



CS com Agentes Móveis

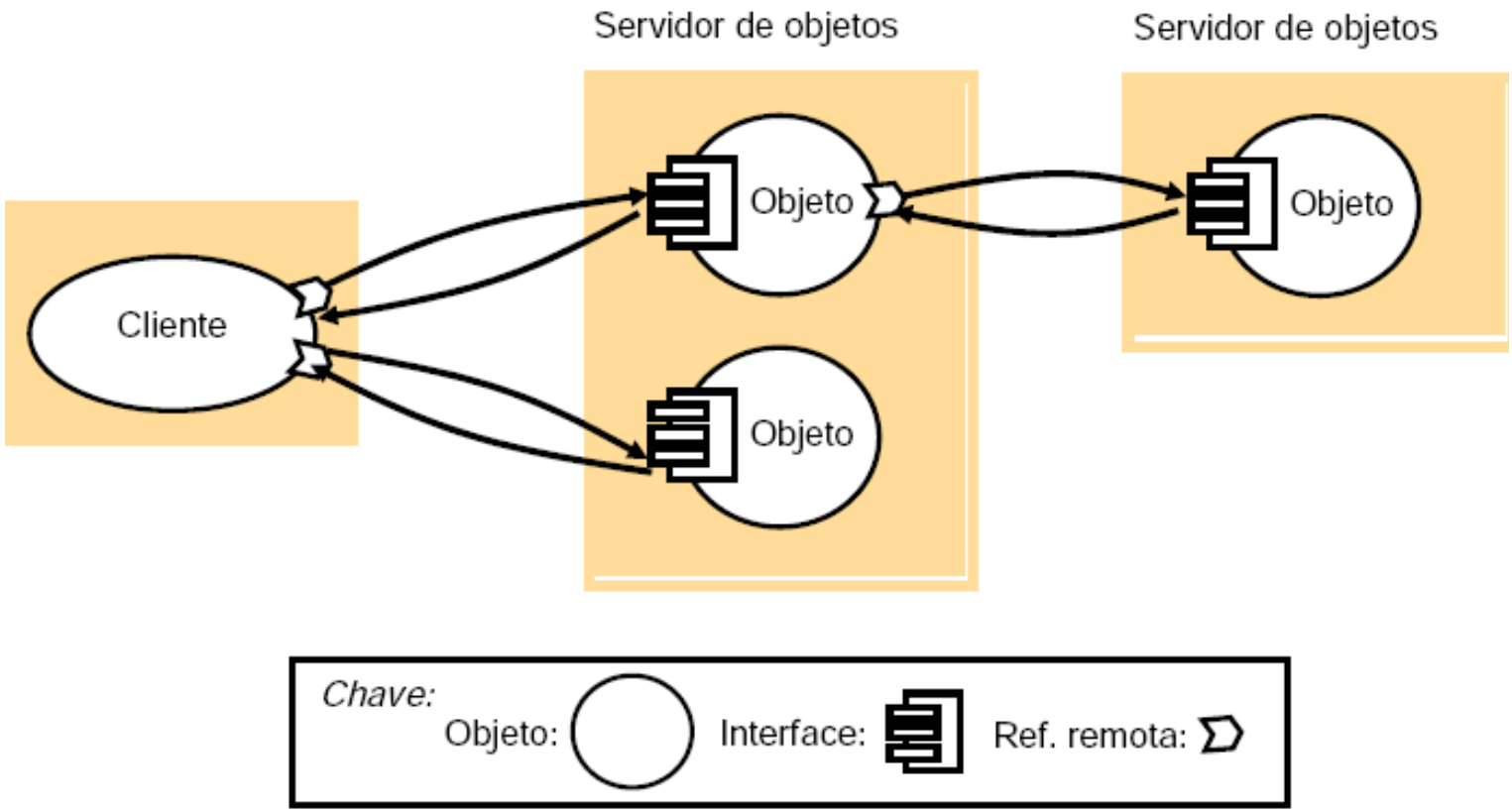
- Desafios de projeto:
  - Heterogeneidade do código e da arquitetura de execução dos agentes
    - Solução: ambientes de execução padronizados em cada ponto do sistema
  - Riscos de segurança na execução dos agentes
    - Solução: restringir a entrada a agentes certificados e executá-los em um ambiente isolado ou com acesso aos recursos locais rigorosamente controlado
  - Riscos de interrupção dos agentes devido à negação de acesso por parte dos servidores
    - Solução: utilizar agentes certificados e com permissão de acesso aos recursos requisitados
  - Atrasos causados pela tempo de transferência dos agentes

Solução: transferência dos agentes em formato compactado

CS com Objetos Distribuídos

- Objetos encapsulados em processos servidores
  - Objetos acessados por outros processos (clientes) através de referências remotas para uma ou mais de suas interfaces
  - Referência remota permite invocar remotamente os métodos disponíveis na interface do objeto referenciado
- Implementação na forma de middleware orientada a objetos (ex.: CORBA, Java-RMI, EJB, .NET)
  - Diferentes mecanismos para **criar**, **executar**, **publicar**, **localizar**, e **invocar** objetos remotos
  - Diferentes serviços de suporte
    - Transação, persistência, replicação, segurança, etc

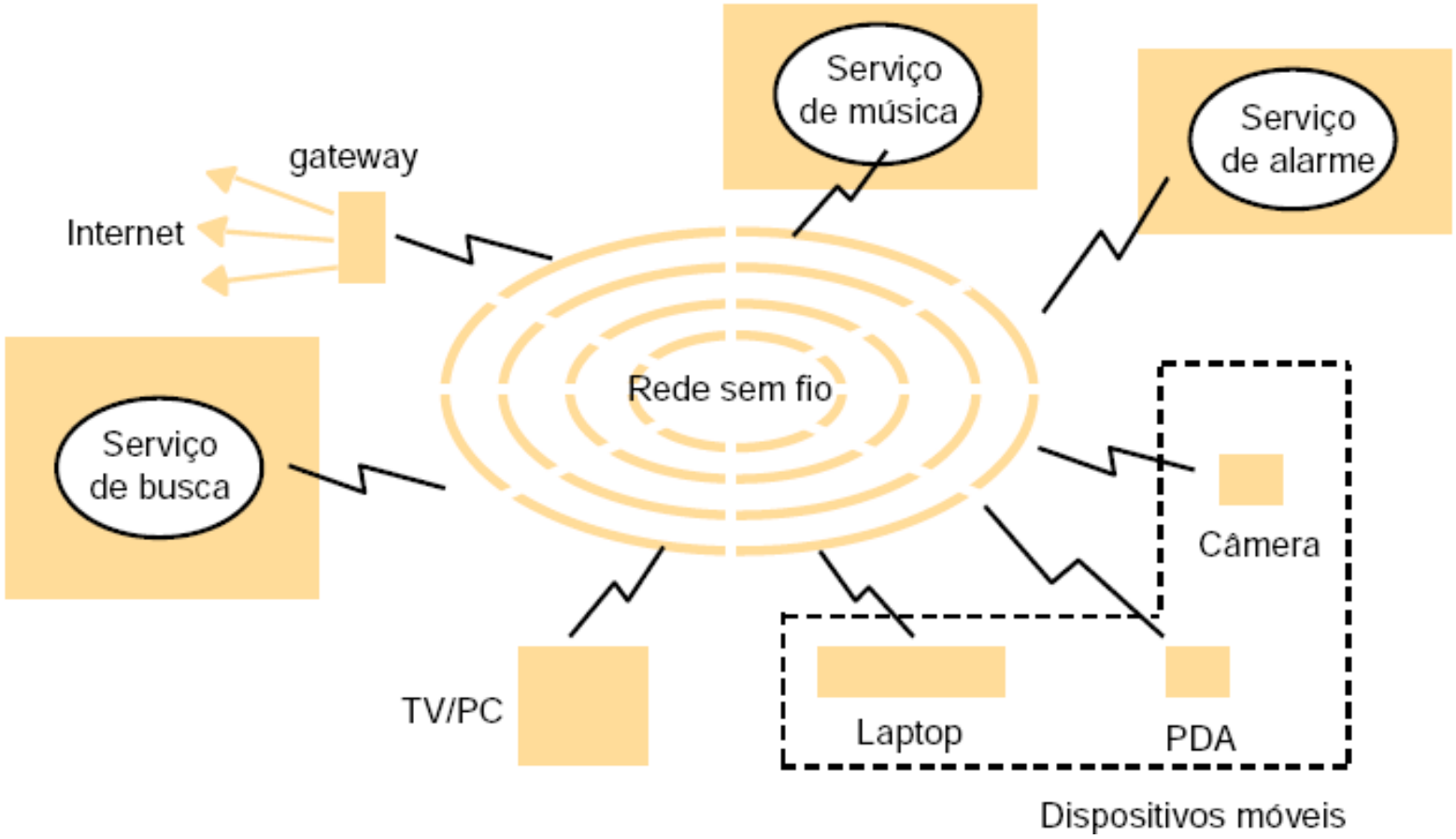
CS com Objetos Distribuídos



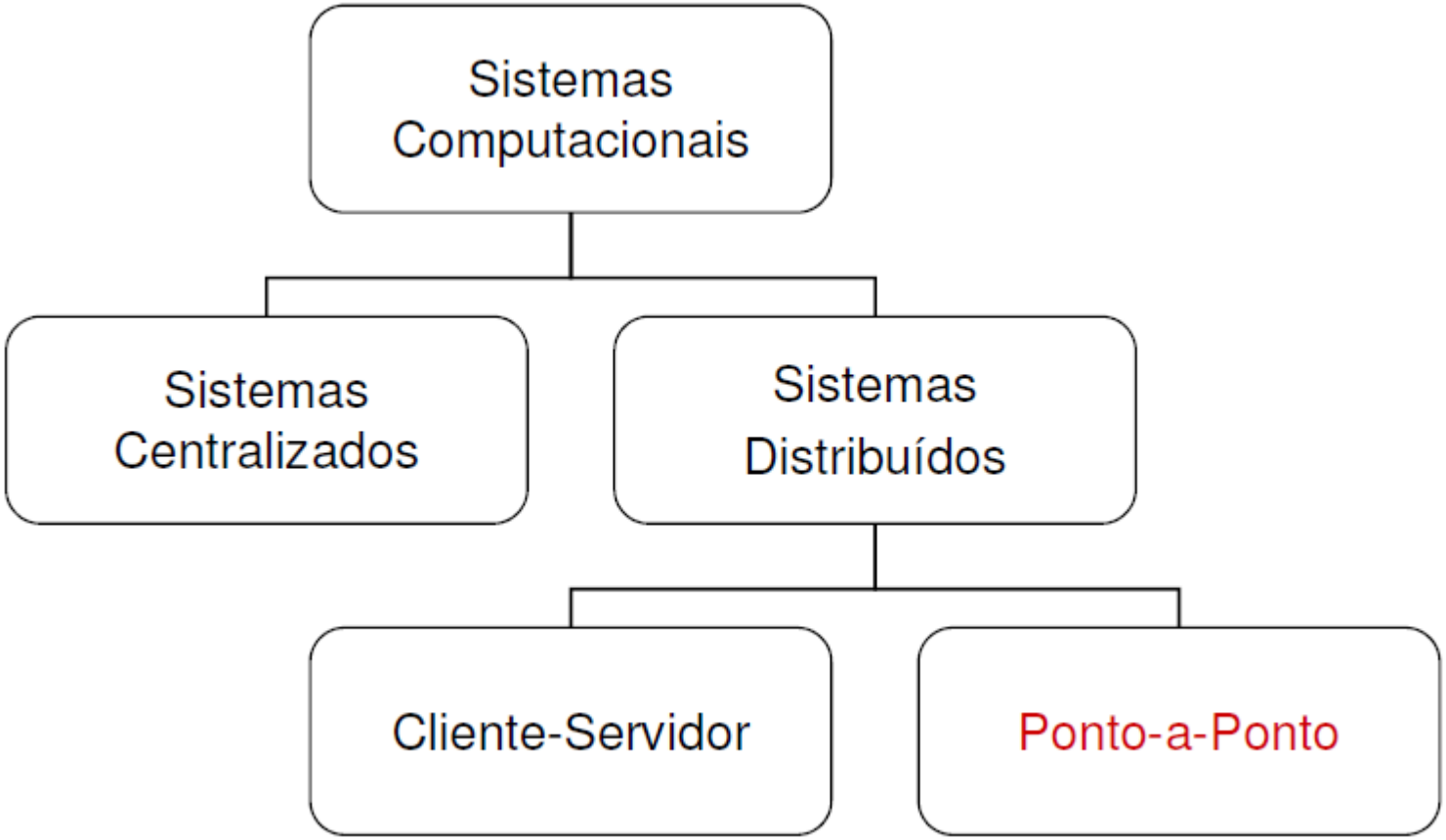
CS com Dispositivos Móveis

- Formado por aplicações clientes que executam em dispositivos móveis (PDAs, laptops, celulares, etc) e acessam servidores da rede fixa através de uma infraestrutura de comunicação sem fio
  - Diferença para as variações com código móvel e agentes móveis?
- Principais benefícios:
  - Fácil conexão dos dispositivos a uma nova rede local
    - Inclusão de novos clientes sem a necessidade de configuração explícita
  - Fácil integração dos clientes aos serviços locais
    - Descoberta automática de novos serviços (sem intervenção do usuário)
- Desafios de projeto:
  - Identificação de recursos independente de sua localização física
  - Limitações de processamento, tempo de conexão e largura de banda
  - Privacidade e segurança

CS com Dispositivos Móveis



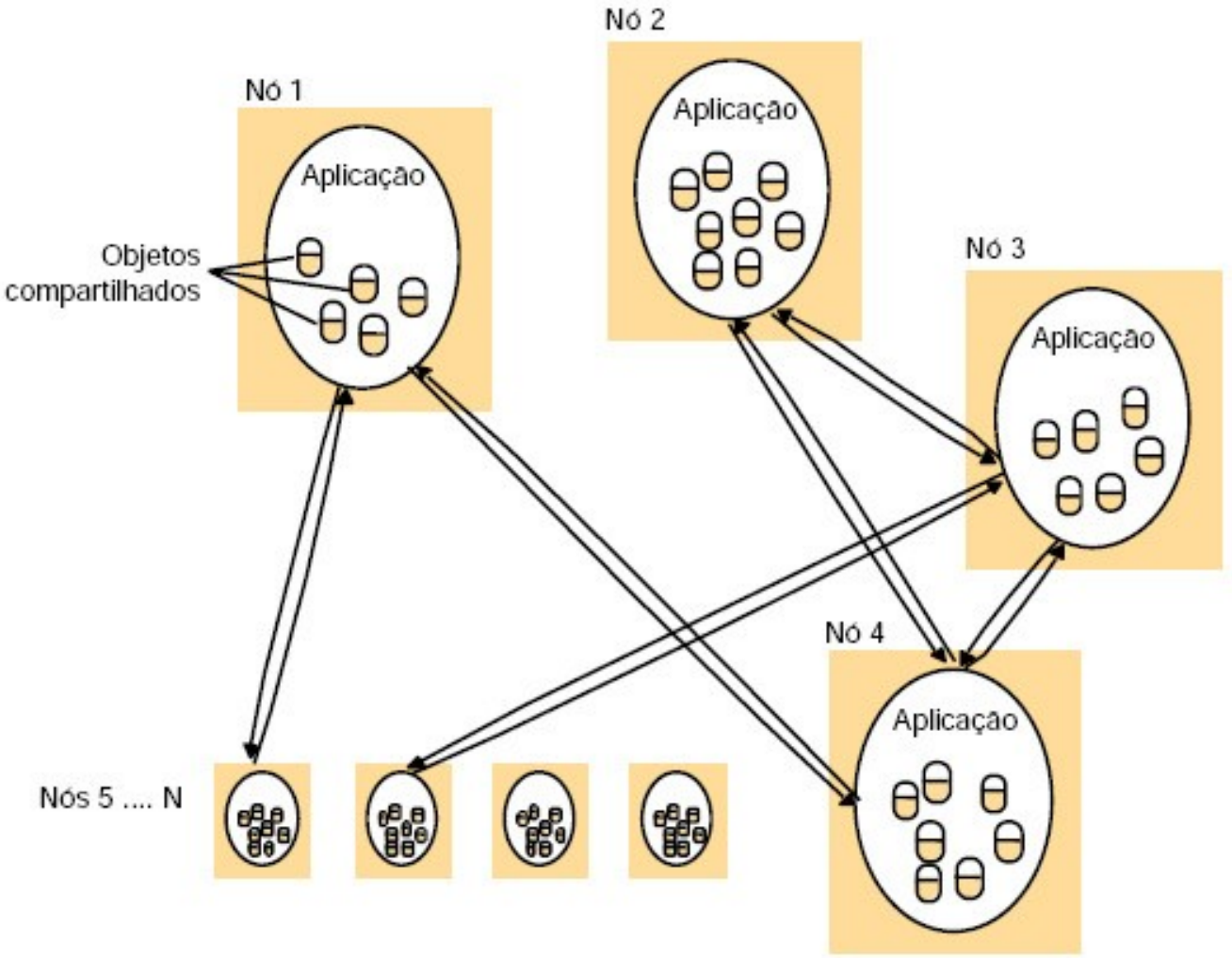
Taxonomia – Sistemas Computacionais



Estilo Ponto-a-Ponto

- Todos os processos (**nós**) envolvidos em uma mesma tarefa ou atividade exercem papéis similares, **interagindo cooperativamente** como “parceiros” (**peers**) uns dos outros
  - Criado para suprir as conhecidas deficiências de escalabilidade do modelo CS tradicional
- O objetivo principal é explorar os recursos (hardware & software) de um grande número de máquinas/usuários interessados em realizar uma determinada tarefa ou atividade
  - Uso pioneiro no compartilhamento de arquivos de áudio (Napster)
  - Sucesso do Napster abriu caminho para vários outros sistemas e middleware P2P de propósito geral (KaZaA, Gnutella, Emule, JXTA, etc)

Estilo Ponto-a-Ponto



Estilo Ponto-a-Ponto

- Características dos sistemas P2P:
  - Arquitetura totalmente **distribuída** (sem qualquer controle centralizado)
  - **Sem distinção** entre clientes e servidores (cada nó é cliente e servidor ao mesmo tempo)
  - Nós podem **trocar recursos** diretamente entre si
  - Nós são **autônomos** para se juntarem ao sistema ou deixá-lo quando quiserem
  - Interação entre nós pode ser feita utilizando mecanismos apropriados para **comunicação em grupo** (difusão seletiva, notificação de eventos)

Estilo Ponto-a-Ponto

- Características dos sistemas P2P:
  - Seu projeto garante que cada usuário contribua com recursos para o sistema
  - Seu correto funcionamento não depende da existência de quaisquer sistemas administrados de forma centralizada
  - Podem oferecer um grau limitado de anonimato para provedores e usuários de recursos
  - Um problema importante dos sistemas p2p é a distribuição de objetos de dados em muitos hosts e subsequente acesso a eles de uma maneira que equilibre a carga de trabalho e garanta a disponibilidade sem adicionar cargas indevidas.

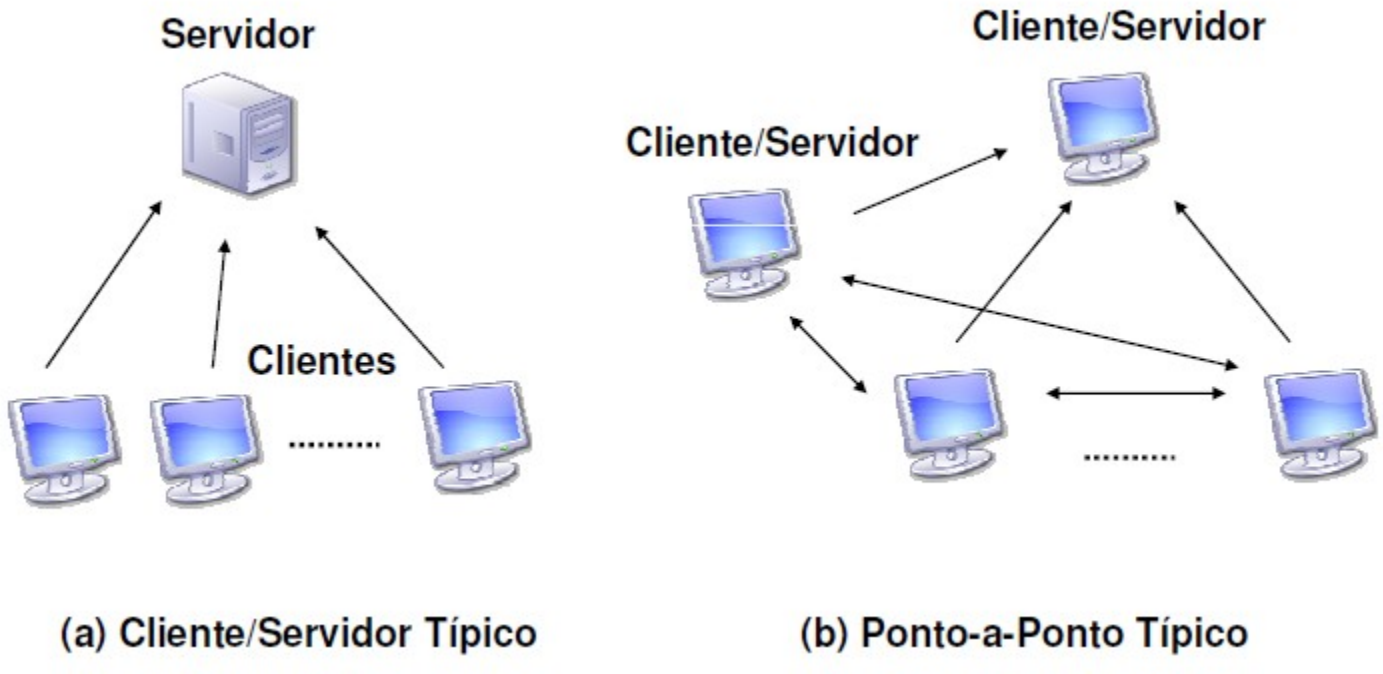
Estilo Ponto-a-Ponto

- Exemplos
  - Primeira Geração
    - Napster
  - Segunda Geração
    - Freenet
    - Gnutella
    - Kazaa
    - Bit-Torrent
  - Terceira Geração – *Middleware*
    - Pastry
    - CAN
    - Chord
    - Tapestry

Classificação de aplicações p2p

- Compartilhamento de arquivos
  - Napster, Gnutella, Freenet, Oceanstore, PAST, Freehaven, KaZaA
- Computação distribuída
  - Seti@home, Entropia, Parabon, Popular Power, …
- Trabalho colaborativo
  - Mensagens instantâneas (Jabber, MSN, AIM, YahooMessager!, …)
  - Groupware(Groove, Netmeetingetc)

Estilo Ponto-a-Ponto vs Clente-Servidor



(a) Cliente/Servidor Típico

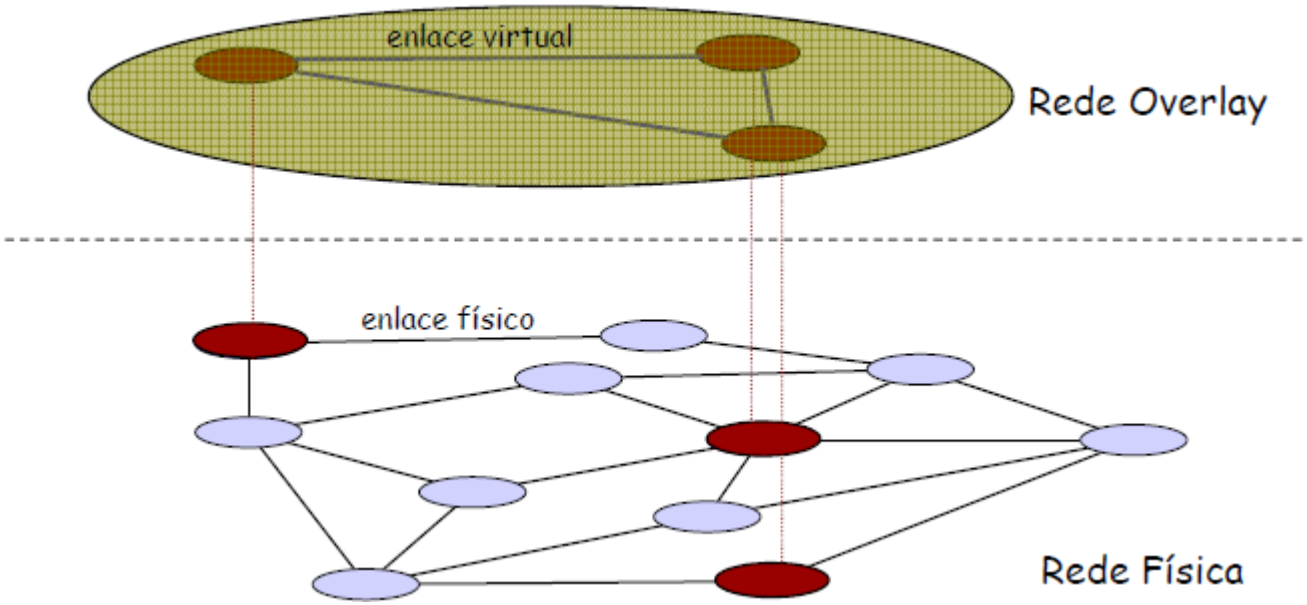
(b) Ponto-a-Ponto Típico

Recursos compartilhados

- Recursos computacionais
  - Espaço em disco
  - Processamento

Rede P2P

- Rede overlay (Nível de Aplicação)
  - Rede virtual criada sobre uma rede já existente
  - Não é exatamente igual à rede física



Tipos de Sistemas

- Não Estruturados (Aleatórios)
  - Sem restrição de localização dos dados
  - Principal aplicação: compartilhamento de arquivos
  - Busca por palavra-chave
  - Alta disponibilidade de arquivos (réplicas nos pontos)
- Estruturados (Determinísticos)
  - Referenciados como *Distributed Hash Tables (DHT)*
  - Alta escalabilidade
  - Boa cobertura e alta precisão

Arquitetura P2P Estruturada

- Tabela *Hash* Distribuída
  - Os dados e os nós recebem um identificador gerado via função hash
  - Arquitetura Chord (Anel)
    - Um item de dado K é mapeado para um nó com id >= K. ***succ(k)***
    - Entradas e saídas geram atualizações no mapeamento

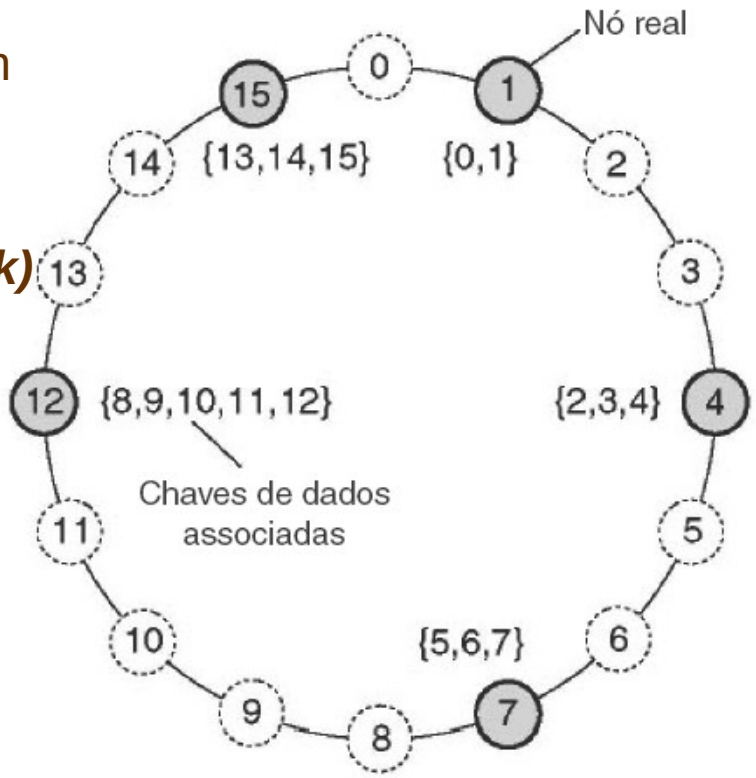
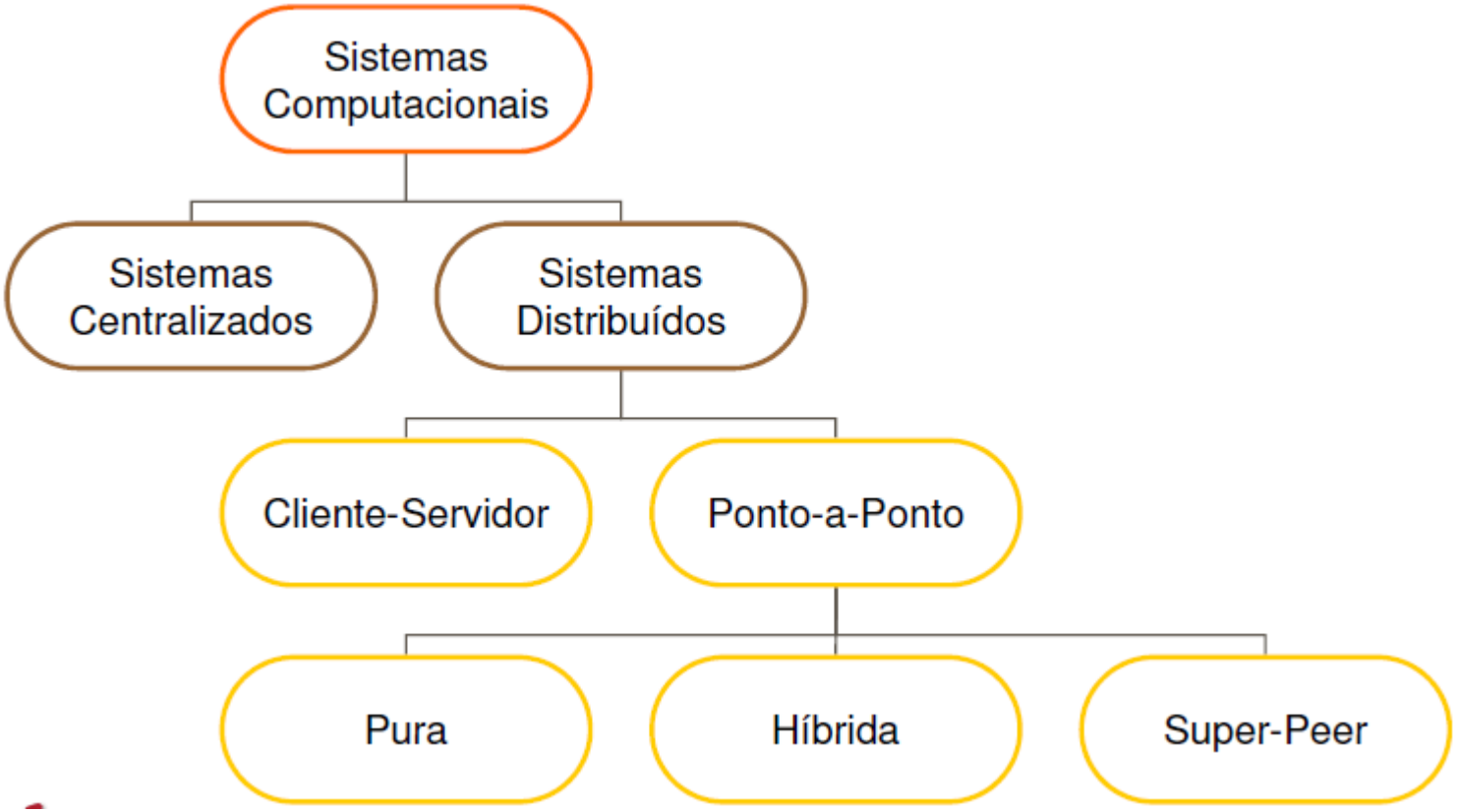


Figura 2.7 Mapeamento de itens de dados para nós em Chord.

Topologias de Redes P2P

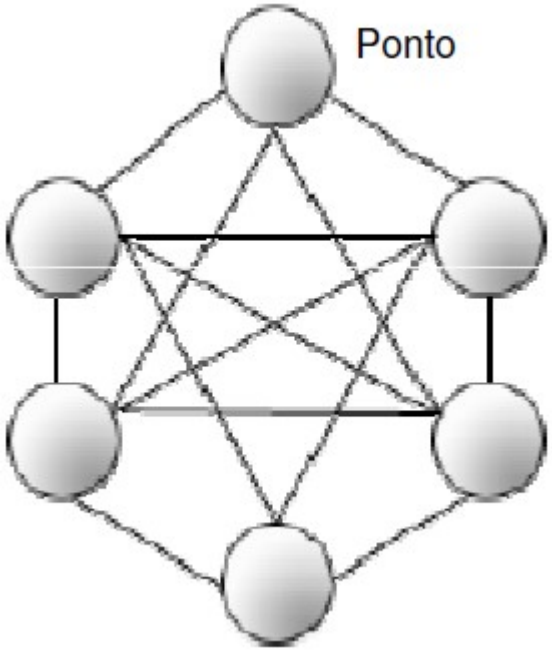
- Topologia
  - Define a organização lógica dos pontos na rede
- Tipos
  - Pura
  - Híbrida
  - *Super-Peer*

Topologias de Redes P2P



Topologia Pura

- Inexistência de um servidor ou repositório centralizado
- Todos os pontos são “iguais” e conectados entre si
- Cada nó conhece apenas os seus vizinhos
  - Determinados quando o nó se junta ao sistema (busca na rede pelos vizinhos mais próximos)
  - Conjunto de vizinhos pode mudar ao longo do tempo
- Busca
  - Não-Estruturada
    - *Flooding*
    - TTL (*time-to-live)*
  - Estruturada: DHT
- Sistemas: Gnutella, Freenet

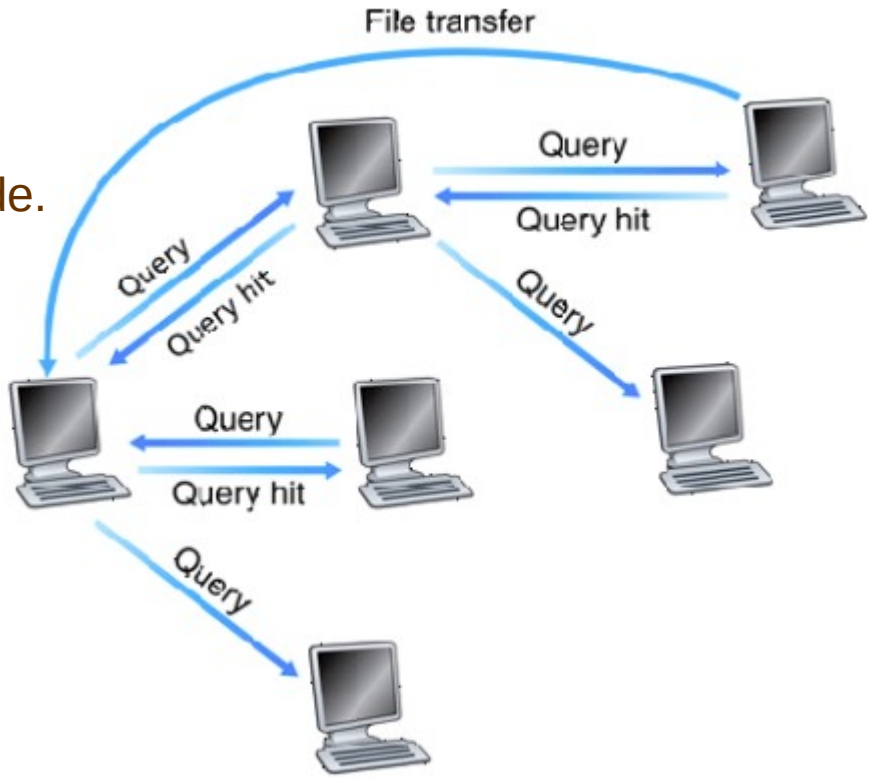


Topologia Pura

- Responsabilidades do ponto, como cliente:
  - Enviar pedidos de serviço a outros pontos
  - Receber as respostas dos pedidos feitos
- Responsabilidades do ponto, como servidor:
  - Receber pedidos de serviço de outros pontos
  - Processar os pedidos e executar um serviço requerido
  - Enviar a resposta com os resultados do serviço requerido
  - Propagar os pedidos de serviço a outros pontos

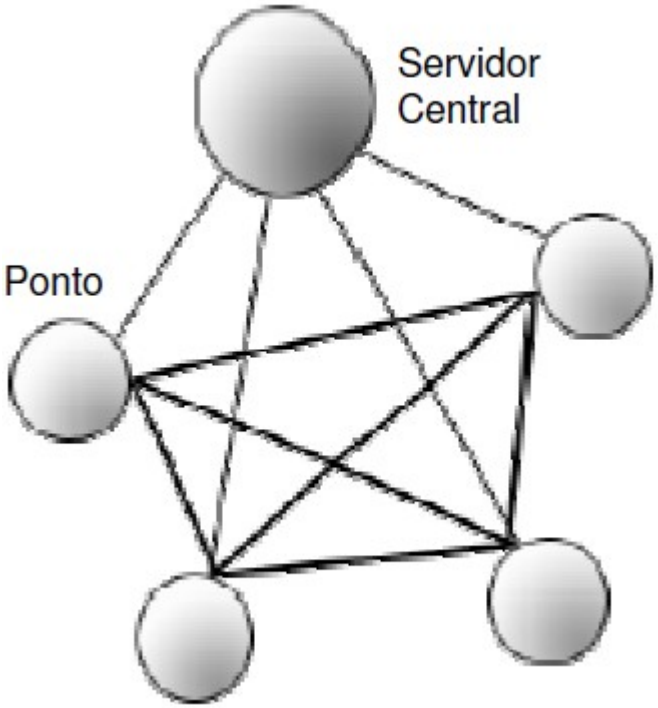
Topologia Pura

- Gnutella
  - ping
    - Descobre servidores na rede.
  - pong
    - Resposta ao ping.
  - query
    - Procura por um arquivo.
  - query hit
    - Resposta ao query.
  - push
    - Solicitação de download para um nó protegido por firewall.

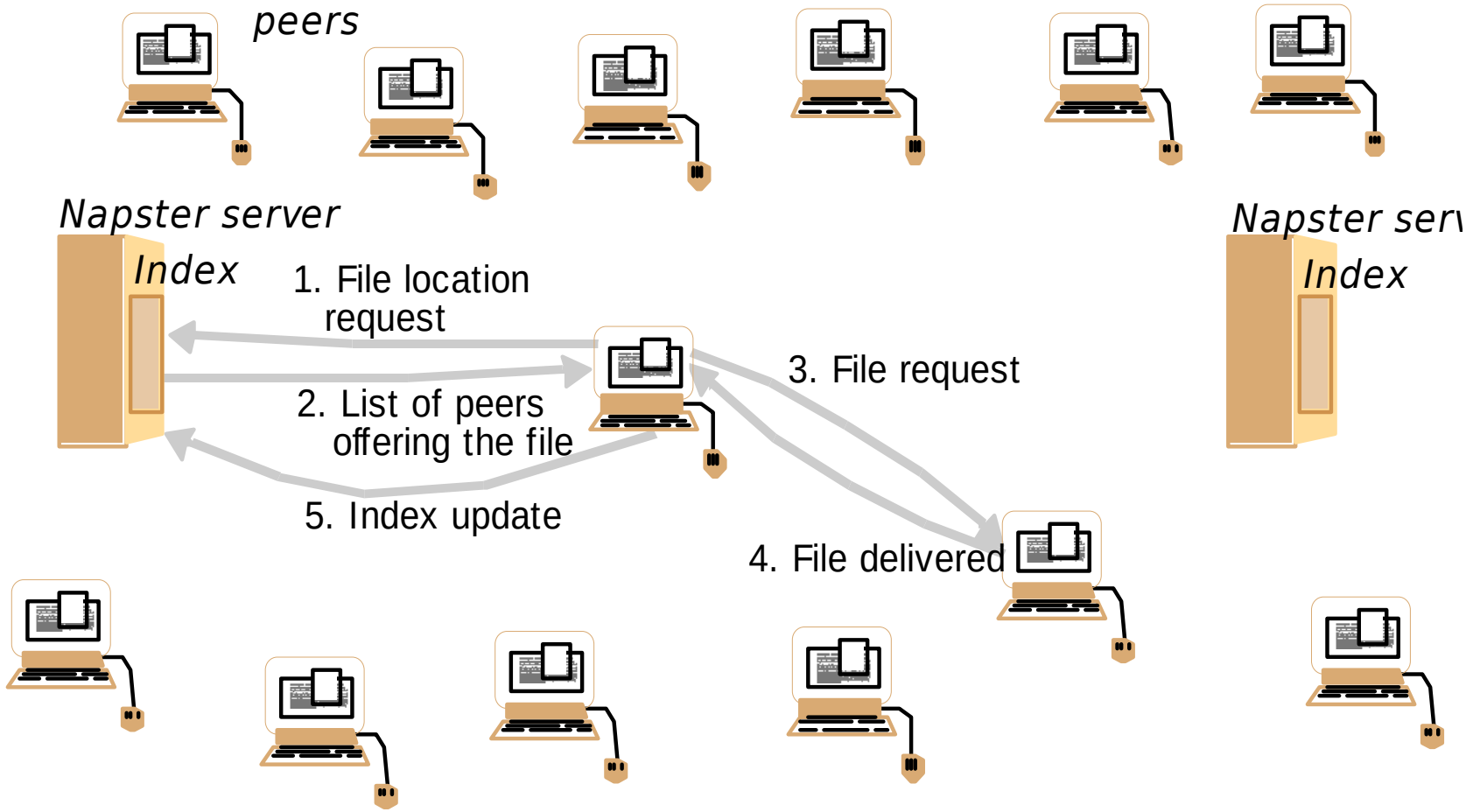


Topologia Híbrida

- Existência de um ou mais servidores centrais
- Informações de controle são armazenadas e fornecidas por um servidor central
  - Ex: descoberta da localização de recursos
- Gerência facilitada
- Servidor central representa um ponto único de falha
- Sistema: Napster



Topologia Híbrida - Napster



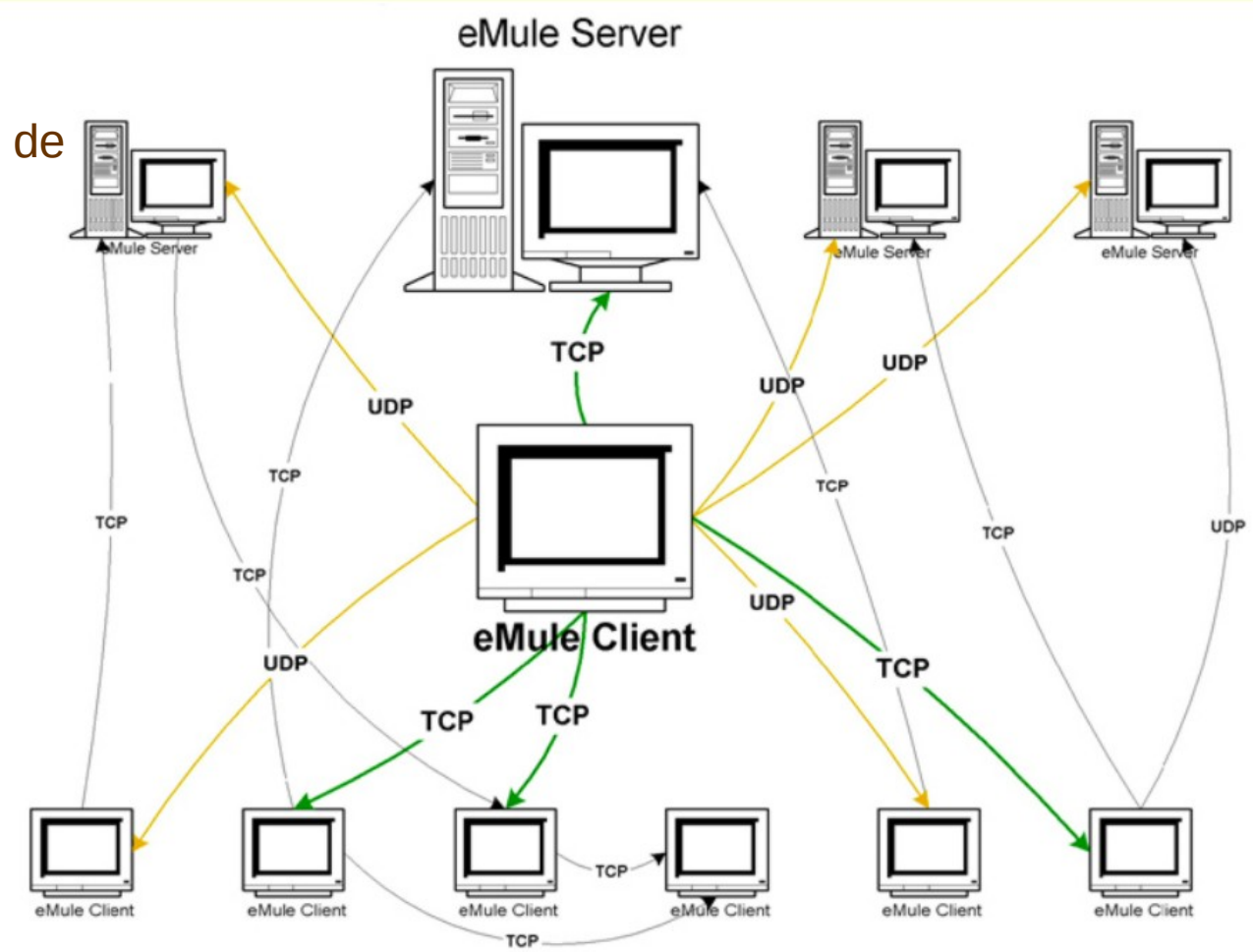
Topologia Híbrida - eMule

TCP

- Envio/recebimento de informações referentes aos arquivos compartilhados
- Download dos arquivos

UDP

- *Keepalive*
- Busca nos demais servidores
- Posição nas filas

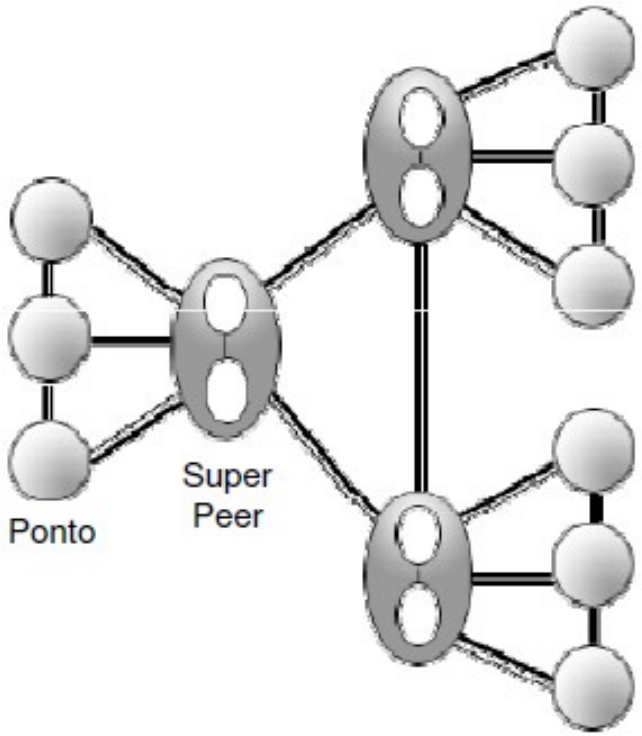


Topologia Híbrida

- Responsabilidades do ponto, como cliente:
  - **Registrar** no servidor seus serviços disponíveis
  - **Enviar ao servidor** pedidos de busca por serviços e receber respostas contendo listas de pontos com os serviços desejados
  - **Enviar a outros pontos** pedidos de serviço e receber as respostas destes pedidos
  - **Processar e executar** os serviços requeridos e enviar repostas a quem fez o pedido
- Responsabilidades do ponto, como servidor:
  - **Registrar** serviços disponíveis nos pontos
  - **Receber pedidos** de busca por serviços disponíveis, buscar por esses serviços e enviar respostas com as localizações dos serviços desejados

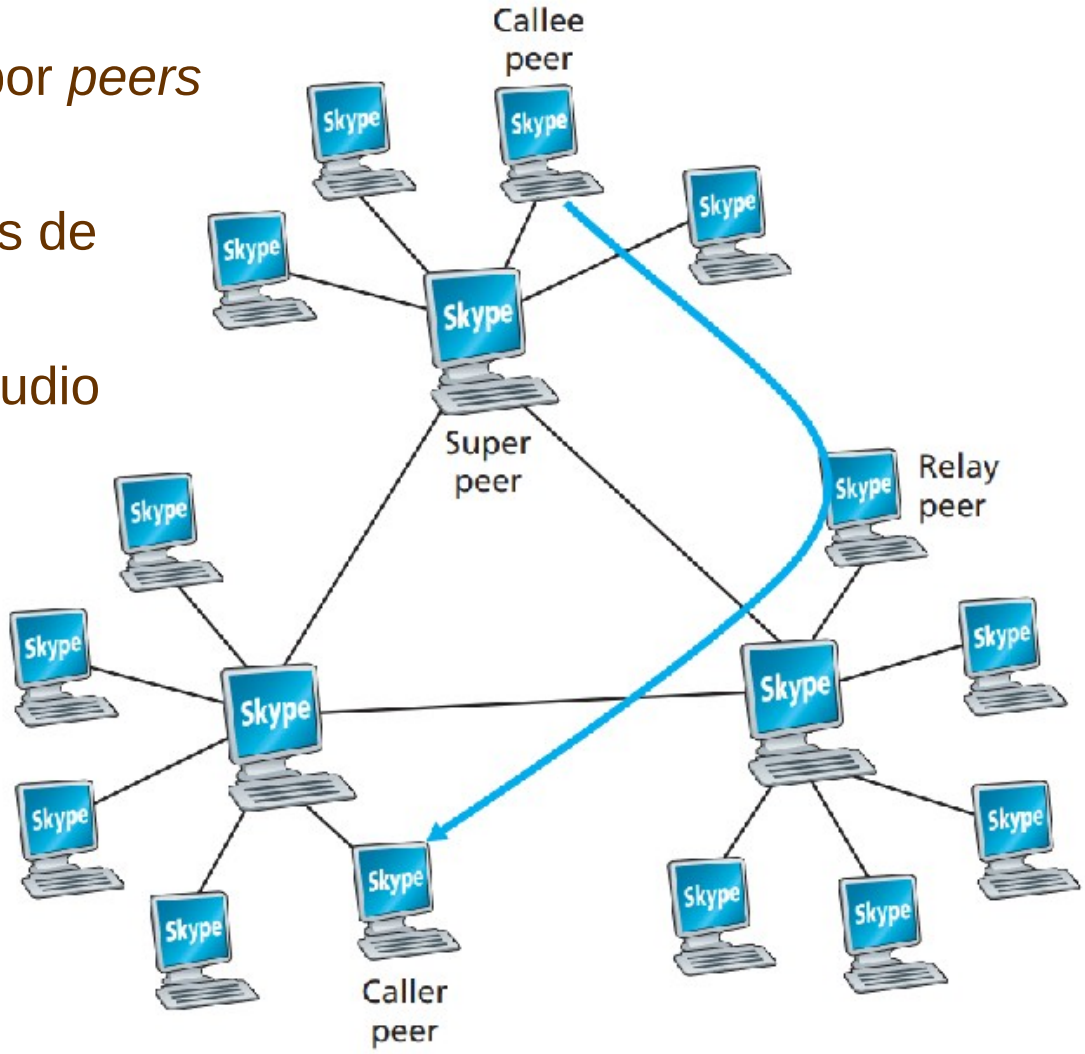
Topologia *Super-Peer*

- Considera:
  - **Heterogeneidade** dos pontos
  - Muitos pontos em conexões de **baixa capacidade** e **alta instabilidade**
  - Poucos pontos em conexões de **alta capacidade** e **baixa instabilidade**
  - Pontos heterogêneos
  - Organização hierárquica
  - Grupos de pontos comunicam-se com outros grupos através de *superpeers*
  - Cada *super-peer indexa as* informações armazenadas no seu conjunto de pontos
- Sistemas: KaZaA, Morpheus



Topologia *Super-Peer - Skype*

- Hierarquia formada por *peers* e *super peers*
- TCP para mensagens de controle
- UDP para envio de áudio e vídeo



Arquiteturas P2P

- Arquitetura básica (ou “pura”)
  - Nenhum nó é especial
  - Cada nó conhece apenas os seus vizinhos
    - Determinados quando o nó se junta ao sistema (busca na rede pelos vizinhos mais próximos)
    - Conjunto de vizinhos pode mudar ao longo do tempo
  - Topologia estruturada (ex.: Chord) ou não-estruturada (ex.: Gnutella)
- Arquitetura híbrida
  - Algumas tarefas (com descoberta da localização de recursos) realizadas através de um ou mais componentes centralizados
  - Demais tarefas realizadas de forma descentralizada através da interação direta entre os nós interessados
  - Exs.: Napster, KaZaA

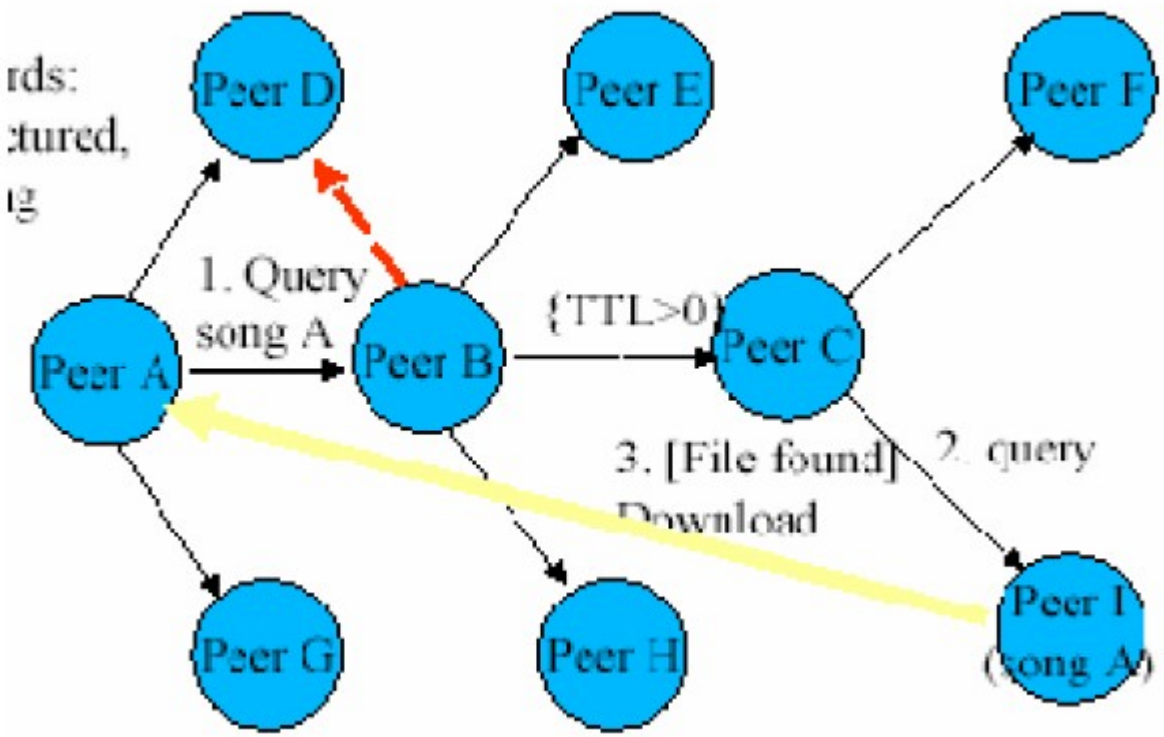
Descoberta de Recursos

- Mecanismos de busca na arquitetura básica
  - Topologia não estruturada
    - Busca por inundação
    - Busca cega
    - Busca informada
    - Busca informada com replicação
  - Topologia estruturada
    - Busca via mapeamento de IDs
  - Mecanismos de busca na arquitetura híbrida
    - Busca em catálogo centralizado
    - Busca via super nós

Busca por Inundação

- O nó que inicia uma consulta a envia para todos os seus vizinhos
- Ao receber uma consulta:
  - Se o nó possui o recurso, ele notifica o iniciador da consulta; o iniciador da consulta pode então obter o recurso diretamente do nó que o notificou
  - Se o nó não possui o recurso, ele decrementa o valor do tempo de vida (***Time To Live – TTL***) da consulta e, caso TTL > 0, repassa a consulta para todos os outros vizinhos

Busca por Inundação (cont.)



Busca por Inundação (cont.)

- Observações:
  - O iniciador da consulta pode obter **resultados redundantes** ou até não obter **nenhum resultado** mesmo que o recurso exista em algum nó da rede **(Por quê?)**
  - Um nó pode ser visitado **mais de uma vez** (nó “D” na figura anterior)
  - ID do iniciador pode ser transmitido junto com a consulta ou ID do provedor do recurso pode ser informado ao iniciador seguindo o caminho inverso da consulta

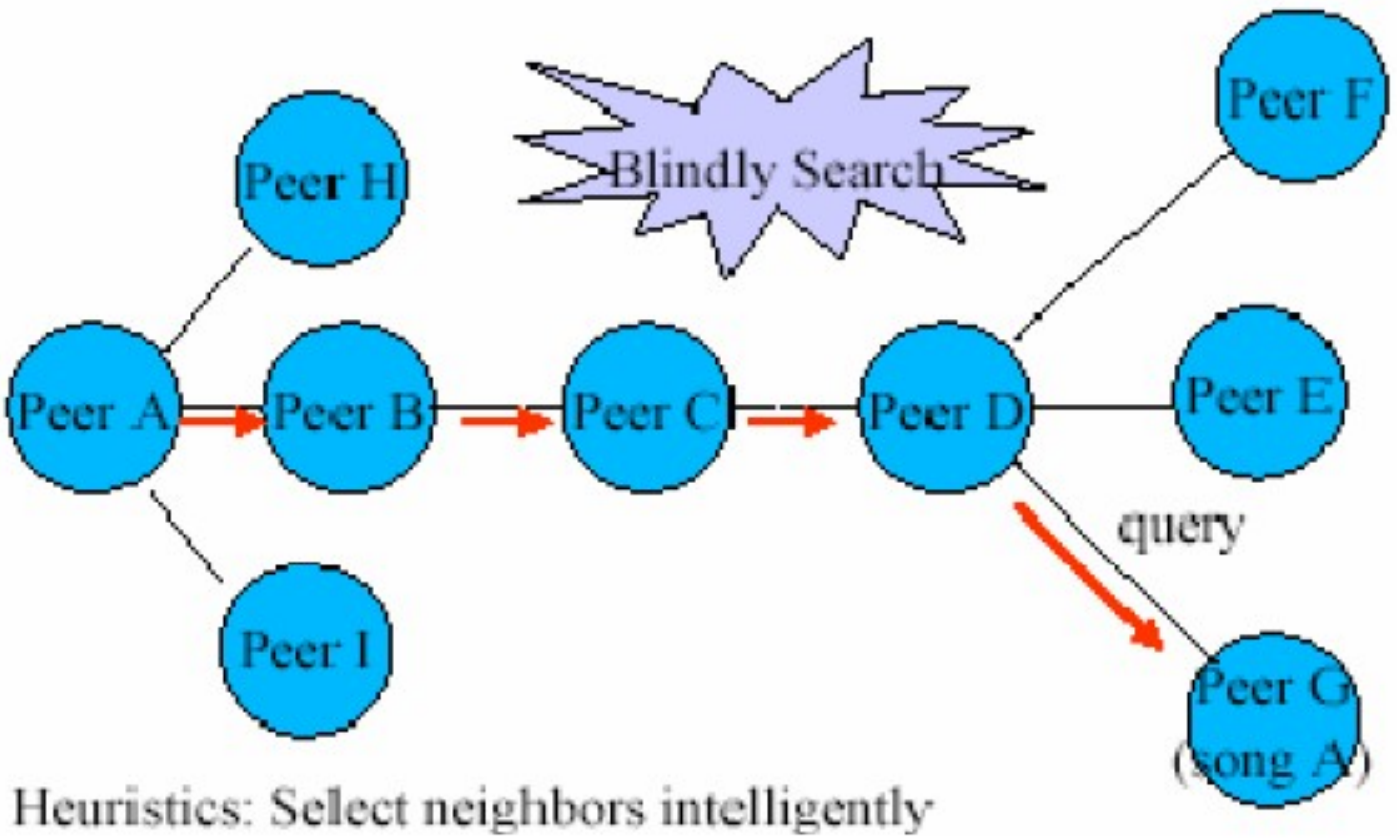
Desvantagens da Busca por Inundação

- Número grande de mensagens trocadas entre os nós
- Consultas enviadas de forma duplicada
- Difícil escolher o valor do tempo de vida das consultas
- TTL alto demais pode sobrecarregar a rede
- TTL baixo demais pode encerrar a busca antes de chegar ao provedor do recurso

Busca Cega (Busca Aleatória)

- Iniciador da consulta seleciona um único vizinho para enviar a consulta
- A seleção é baseada em algumas heurísticas:
  - Por exemplo, se um vizinho sempre retorna um resultado satisfatório, ele deve ser selecionado com mais freqüência
    - Deve ser possível saber quando cada vizinho foi capaz ou não de encontrar um determinado recurso
- Se o vizinho não possui o recurso, ele seleciona um dos seus vizinhos para repassar a consulta
  - Seleção também baseada em heurísticas
- O processo se repete até que o recurso seja encontrado ou o tempo de vida consulta expire (TTL = 0)

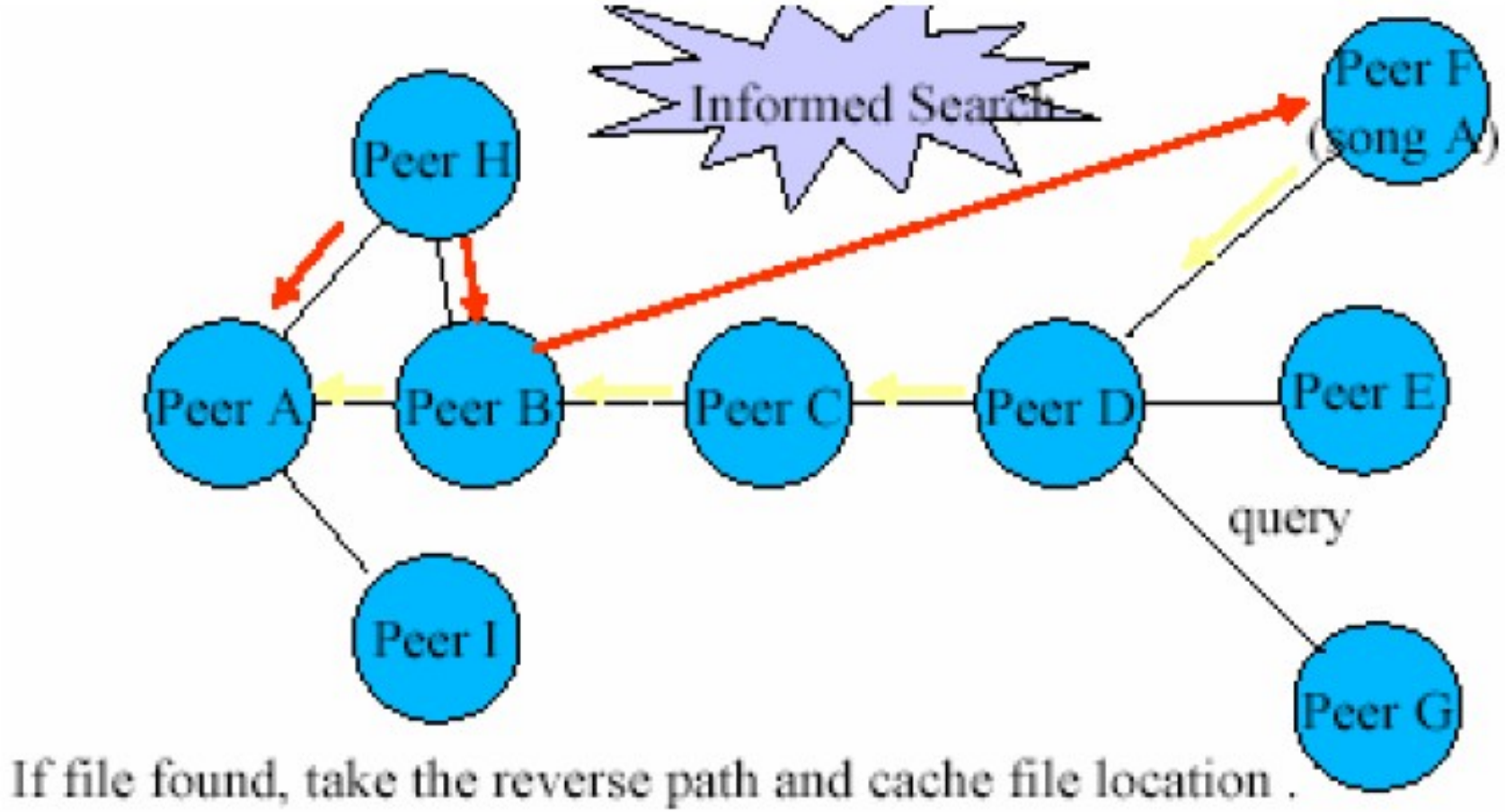
Busca Cega (Busca Aleatória)



Busca Informada

- Cada nó possui um *cache para armazenar a* localização de recursos que tenham sido consultados previamente
- Ao receber uma consulta:
  - Se o nó encontra a localização do recurso no seu *cache, ele a informa ao nó de quem recebeu a consulta*
  - Do contrário, ele tenta descobrir a localização do recurso fazendo busca por inundação
- Uma vez que o recurso é encontrado, o caminho inverso da consulta é usado para informar a localização ao iniciador da consulta
  - Dessa forma, os nós que fazem parte do caminho percorrido pela consulta podem atualizar seus *caches,* acelerando assim as próximas consultas

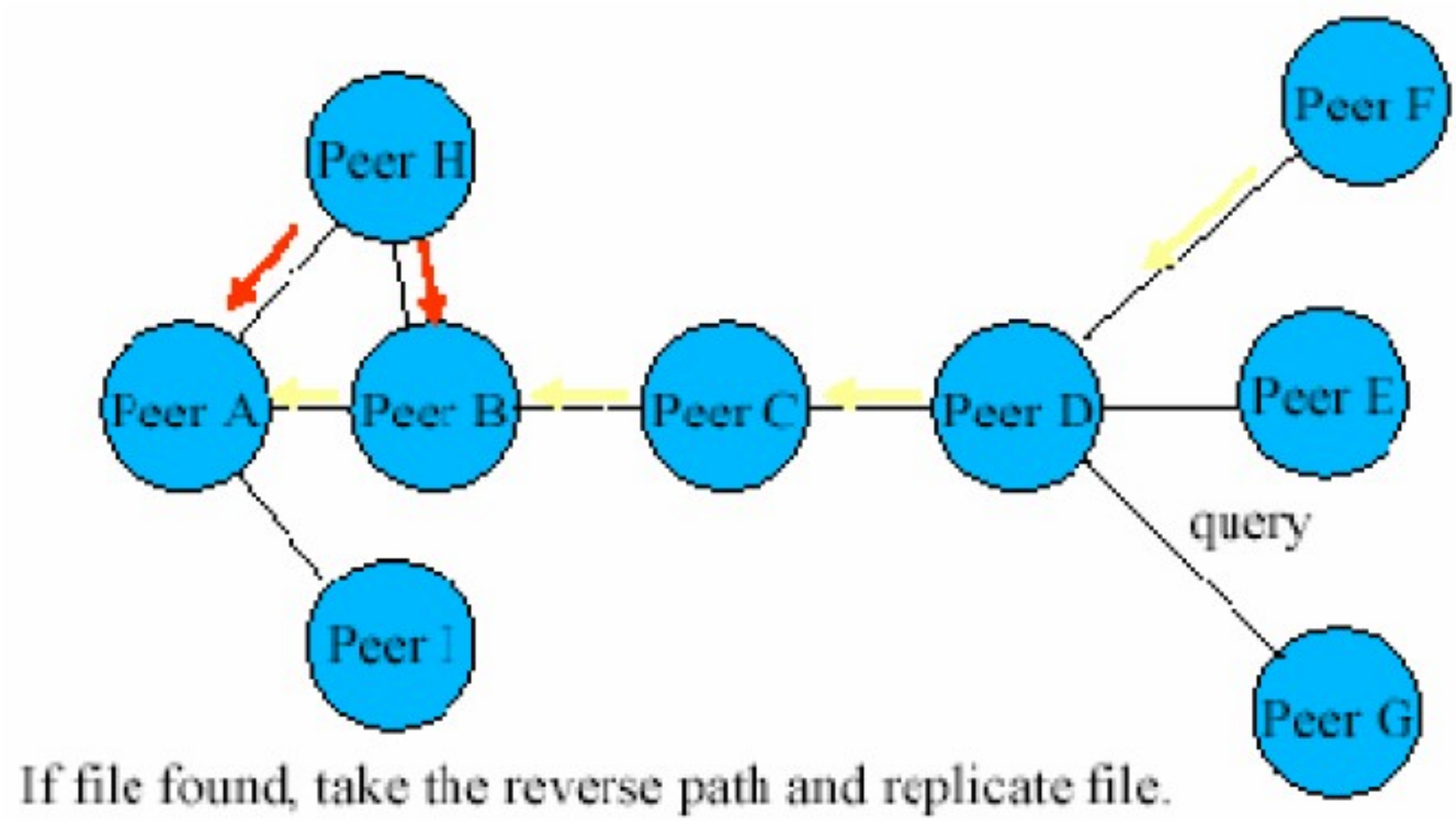
Busca Informada



Busca Informada com Replicação

- A diferença para a busca informada simples (sem replicação) é que os *caches dos nós que compõem o* caminho inverso da consulta serão usados para **armazenar o próprio recurso**, ao invés de apenas a sua localização
- Dessa forma, os nós que fazem parte do caminho da consulta acelerarão não apenas a descoberta mas também o próprio acesso ao recurso nas próximas consultas

Busca Informada com Replicação



Busca Via Mapeamento de IDs

- Tentativa de melhorar os algoritmos de roteamento dos sistemas P2P não-estruturados
- Os recursos são distribuídos através dos nós de acordo com um **algoritmo de mapeamento**
  - Os nós não escolhem seus recursos por “vontade própria”
  - Necessidade de um **mecanismo de replicação adicional** para oferecer maior disponibilidade
- Algoritmo mais usado
  - Funções Hash

Busca Via Mapeamento de IDs

- Mecanismo de mapeamento
  - Mapeia um ponto em um identificador único
    - h('172.17.166.99') -> 8400
  - Mapeia um item (arquivo) em um identificador único
    - h('aula-sd-2.ppt') -> 8045
  - Qualquer função aleatória de *hash "boa" é suficiente*
    - Padrão SHA-1 (dificilmente ocorre colisão)
  - Cada nó é responsável por armazenar recursos que tenham chaves de busca "similares" ao (ou seja, que possam ser facilmente mapeadas para o) identificador do nó
  - Dada uma consulta com uma chave de busca, qualquer nó é capaz de repassá-la rapidamente ao nó cujo identificador mais se assemelha à chave

Busca Via Mapeamento de IDs

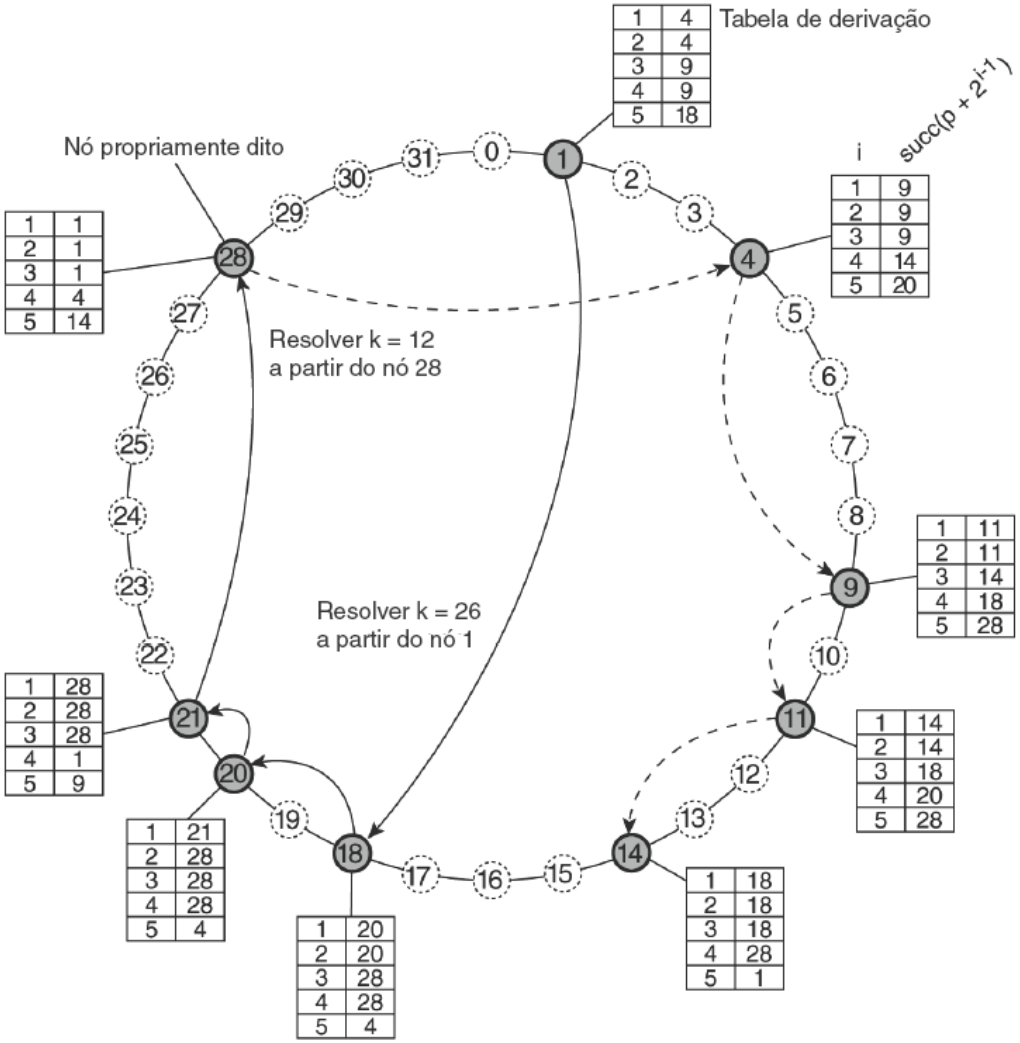
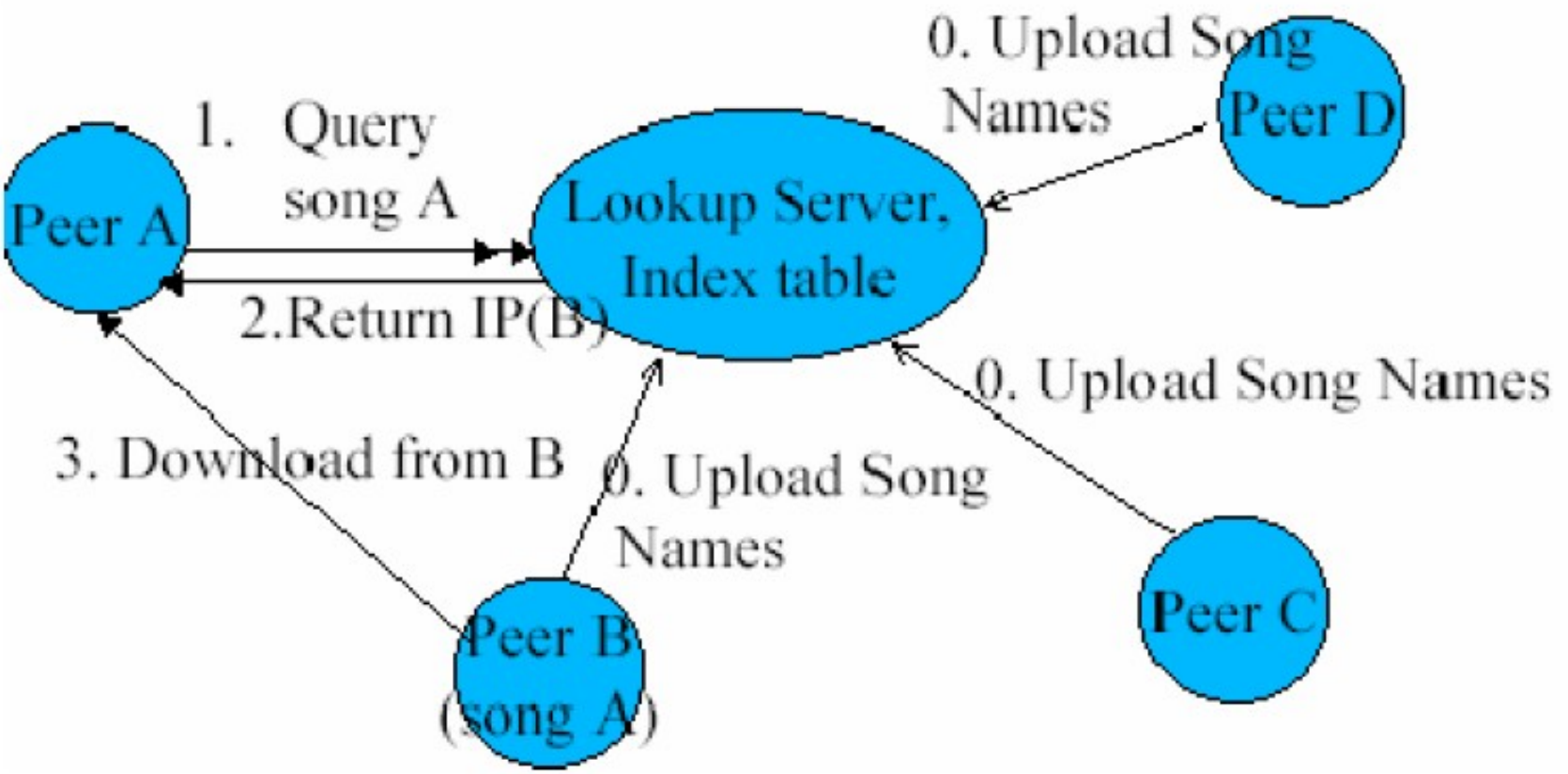


Figura 5.4 Resolução da chave 26 a partir do nó 1 e da chave 12 a partir do nó 28 em um sistema Chord.

Busca em Catálogo Centralizado

- Descoberta da localização dos recursos feita através de consulta a um único nó central (**servidor de *lookup)***
- Cada nó fornece ao servidor de *lookup* **meta-informação descrevendo os** recursos que provê
- Acesso aos dados dos recursos feito **diretamente entre os nós clientes e o** nós provedores

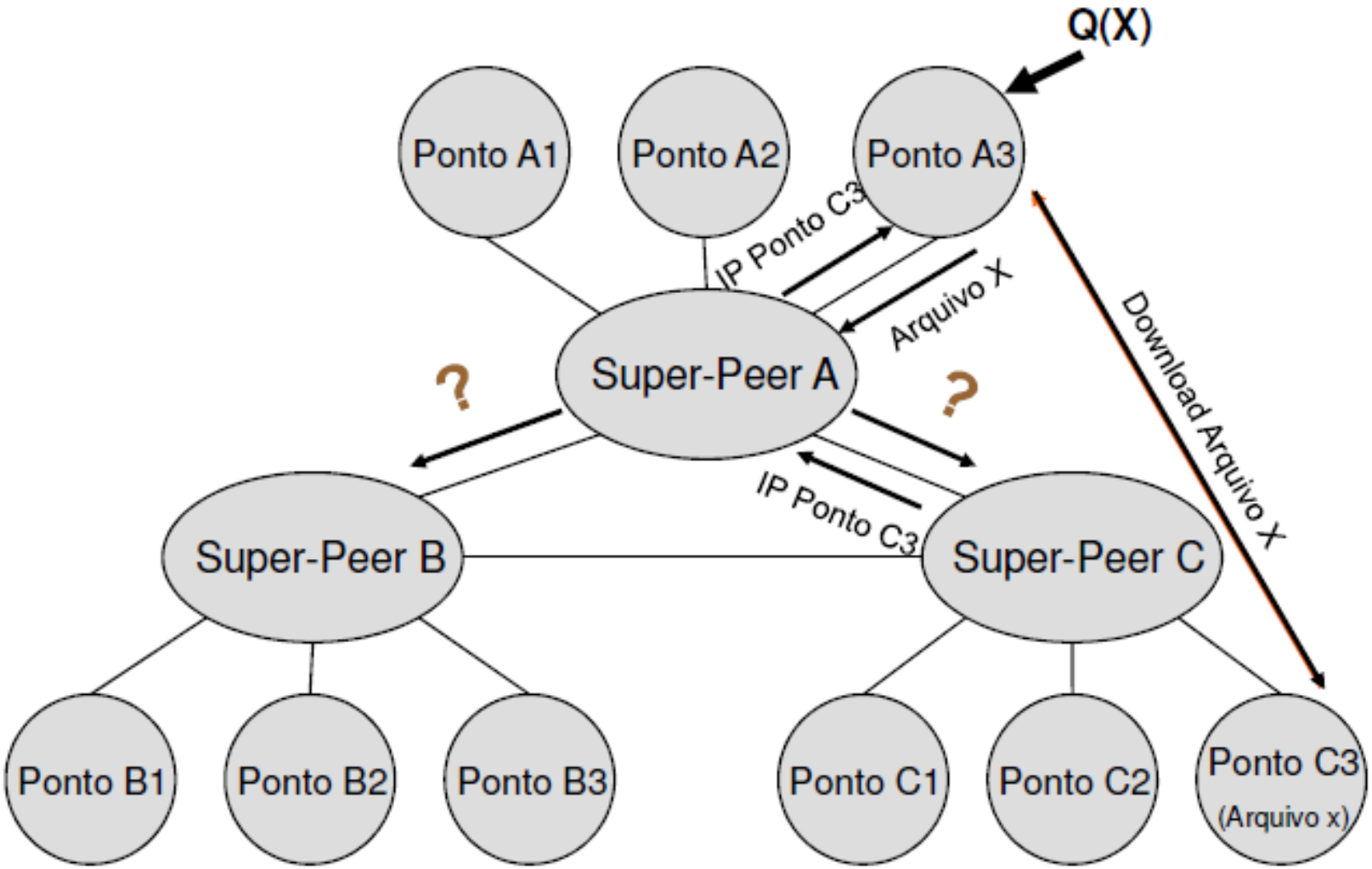
Busca em Catálogo Centralizado



Busca Via Super Nós

- Nós organizados numa **estrutura hierárquica** composta por nós "**filhos" e "super" nós**
- Cada super nó mantém um **catálogo** descrevendo os recursos mantidos por cada um de seus nós filhos
- Um nó filho inicia uma consulta a **enviando** para o seu super nó
- Um super nó processa consultas recebidas de **outros super nós em nome de seus nós filhos**
- Acesso aos dados dos recursos feito **diretamente entre os nós clientes e os nós** provedores

Busca Via Super Nós



Vantagens

- Poder computacional (recursos dos demais pontos)
- Pontos com diferentes papéis (cliente ou servidor)
- Compartilhamento de recursos
- Melhor desempenho, tolerância a falhas (replicação)
- Autonomia dos pontos participantes
- Ausência de administração
- Escalabilidade (e.g. KaZaA com ~3-4 milhões de usuários)

Desvantagens

- Problemas com disponibilidade e consistência
- Pode prejudicar o desempenho de pontos
- Ausência de administração centralizada
- Usuários responsáveis por gerenciar seus próprios recursos
- Segurança

Requisitos de Projeto para Arquiteturas Distribuídas

- Questões chave para o projeto (arquitetura) de um sistema distribuído
  - Desempenho
  - Qualidade de serviço (QoS)
  - Cache e replicação
  - Dependabilidade
- Geralmente consideradas na implementação de aplicações distribuídas que compartilham recursos em larga escala

Desempenho

- Capacidade do sistema para reagir de **forma rápida e consistente** às requisições dos usuários
  - Sujeita às **limitações de processamento e comunicação** dos computadores e da **infra-estrutura** de rede
- Principais fatores envolvidos:
  - Tempo de resposta (***responsiveness - reatividade)***
    - Afetado pelo **número de camadas** de software necessário para a invocação dos serviços remotos e pelo **volume de dados** transferidos através da rede
  - Taxa de trabalho (***throughput)***
    - Medida do desempenho do sistema considerando todos os usuários
  - Balanceamento de carga (***load balance)***
    - Utilizado para explorar de forma mais **eficiente os recursos** computacionais disponíveis

QoS

- Capacidade do sistema para oferecer **serviços com garantias** suficientes para atender de forma satisfatória as necessidades específicas de seus usuários
- Principais fatores:
  - Confiabilidade
  - Segurança
  - Desempenho
  - Adaptabilidade
    - para atender mudanças de configuração e disponibilidade de recursos
  - Disponibilidade
- Aspectos de **confiabilidade, segurança e desempenho** serão abordados no contexto dos modelos fundamentais de **falha, segurança e interação,** respectivamente

QoS (cont.)

- No contexto de QoS, o **desempenho também é** definido em termos da capacidade do sistema de atender restrições de **tempo crítico**
- Em geral, essas restrições devem ser **mantidas durante todo o tempo**, e sob todas as circunstâncias, em que os recursos são utilizados, especialmente sob alta demanda
  - **Recursos críticos** devem ser reservados a priori junto aos seus respectivos servidores (solicitações não atendidas são rejeitadas)
- Garantias negociadas entre as partes através de acordos em nível de serviço (**SLAs)**

Cache e Replicação

- Capacidade do sistema para manter múltiplas cópias de um mesmo recurso lógico fisicamente distribuídas, de modo a reduzir o seu tempo de acesso
  - Ex.: protocolo de cache da web
- Principais questões envolvidas:
  - **Alocação e distribuição das réplicas**
  - Políticas de **acesso e atualização**
  - Mecanismo de **validação**
  - Compromisso entre a **consistência e a qualidade do** serviço
    - Freqüência de **atualização X desempenho**
    - Suporte para operações “desconectadas” (***off-line)***

Dependabilidade

- Capacidade do sistema continuar operando efetivamente, mesmo diante da **ocorrência de falhas** e da ameaça de **acessos indevidos** aos recursos compartilhados
- Principais questões:
  - **Tolerância a falhas**
    - Obtida através da **redundância (replicação)** de recursos lógicos e físicos
    - Implica em **maiores custo e complexidade**
  - **Segurança**
    - Obtida através de mecanismos de **criptografia,** garantia da **integridade** dos dados**, assinatura** digitais, políticas de controle de acesso, etc

Modelos Fundamentais

Modelos Fundamentais

- Independente de ser cliente/servidor ou *p2p, todos os modelos possuem características* comuns:
  - São constituídos de processos
  - Esses processos se comunicam através do envio de mensagens em uma rede de comunicação
  - Requisitos de projeto semelhantes
    - Desempenho e confiabilidade das redes e processos
    - Segurança dos recursos compartilhados

Modelos Fundamentais

- Foco em três importantes aspectos de projeto:
  - Mecanismo de interação
  - Tratamento de falhas
  - Segurança
- Utilizados para ajudar a planejar, entender e analisar o comportamento esperado do sistema
- Principais benefícios:
  - **Correção** antecipada de erros
  - Investigação, avaliação e reuso de diferentes **alternativas de projeto**
  - **Menor custo** de desenvolvimento, manutenção e evolução

Modelos Fundamentais

- Modelos de Interação:
  - Como os processos se comunicam?
  - Deve refletir o fato de que a comunicação ocorre com atrasos
- Modelos de Falha:
  - Define e classifica as falhas
- Modelos de Segurança:
  - Define e classifica as formas que ataques de agentes externos ou internos podem assumir

Modelo de Interação

- Descreve as formas de **interação e coordenação** entre os componentes do sistema
  - Programa Simples vs Sistema Distribuído
- Influenciado pela:
  - Capacidade de **comunicação da rede**
    - **Latência (transmissão, acesso à rede, S.O.)**
    - Largura de banda
    - **Instabilidade (*jitter)***
  - Ausência de estado global
    - Impossibilidade de acordo sobre a mesma **noção de tempo**
    - Dificuldade para **sincronizar relógios através da rede**
- Duas variantes em relação ao tempo:
  - Modelo **síncrono**
  - Modelo **assíncrono**

Modelo de Interação

- Descreve as formas de **interação e coordenação** entre os componentes do sistema
  - Processos interagem entre si através da **troca de mensagens** para coordenar suas atividades
  - Um algoritmo **distribuído** define os passos necessários para essa coordenação
  - Cada processo possui seu próprio **estado**
    - Ausência de estado global
  - O desempenho do canal de comunicação subjacente não é previsível (atraso, banda, *jitter)*
  - Não é possível manter uma noção global de tempo única/precisa

Modelo de Interação

- Desempenho do canal de comunicação
  - Latência: Atraso entre o início da transmissão da mensagem para um processo e o início da recepção dessa mensagem pelo outro processo
    - soma de várias componentes:
      - transmissão + acesso à rede + propagação
  - Largura de banda
    - Quantidade total de informação que pode ser transmitida em uma unidade de tempo
  - *Jitter*
    - Variação do atraso
    - Muito importante para dados de tempo real

Relógios e temporização de eventos

- Cada computador possui seu próprio relógio
  - Pode ser usado para marcar o tempo de eventos locais
- Mas sem significância global
  - Cada relógio marca um tempo diferente
  - Defasagem entre os relógios
- Técnicas para sincronização de relógios podem ser aplicadas
  - GPS, algoritmos de sincronização

Modelo Síncrono

- Características:
  - O tempo para **executar cada passo de um processo tem limites inferior e superior conhecidos**
  - Cada **mensagem transmitida** por um canal de comunicação é recebida dentro de um **limite conhecido** de tempo
  - Cada processo tem um relógio local cuja **taxa de desvio** do tempo real tem um **limite conhecido**
- Vantagens:
  - **Mais fácil** para **programar e analisar** o comportamento dos processos
    - ex.: *timeouts* para detectar falhas
- Desvantagens:
  - Dificuldade de definir e garantir **valores realistas para os** limites de tempo (***timeout)***
  - Resultados de análise e simulação podem não ser confiáveis

Modelo Assíncrono

- Características:
  - **Sem limites** conhecidos para
    - **Velocidade** de execução dos processo
    - **Atraso na transmissão** das mensagens
    - Taxa de **desvio dos relógios**
- Vantagens:
  - Mais realista
  - Soluções assíncronas também são válidas para o modelo síncrono
- Desvantagens:
  - Mais difícil de implementar e analisar
- Exemplos?

Ordenação de Eventos

- Em muitos casos, a execução de um sistema distribuído pode ser descrita em **termos dos eventos** ocorridos no sistema e da ordem em que eles ocorreram
  - Problema:
    - como saber se um determinado evento (ex.: envio ou recebimento de uma mensagem) de um determinado processo ocorreu **antes, após ou concorrentemente** a um outro evento de um outro processo (possivelmente remoto)?
- Como exemplo, considere o seguinte cenário de troca de mensagens entre um grupo de quatro usuários (X, Y, Z e A) de correio eletrônico
  1. O usuário X envia uma mensagem *m1 com o assunto* “Encontro”;
  2. Os usuários Y e Z respondem enviando as mensagens *m2* e *m3, respectivamente, com o assunto “Re: Encontro”.*

Ordenação de Eventos

- (Continuação do exemplo)
  - Em tempo “real”, X envia *m1 primeiro; em seguida, Y recebe* e lê *m1, e então responde enviando m2; por fim, Z recebe e* lê tanto *m1 quanto m2, e responde enviando m3*
  - Porém, devido a atrasos na rede, as três mensagens podem ser entregues a alguns usuários na ordem errada. Por exemplo, o usuário A poderia receber as mensagens na seguinte ordem:

Caixa de Entrada de A	
De	**Assunto**
Z	Re:Encontro
X	Encontro
Y	Re:Encontro

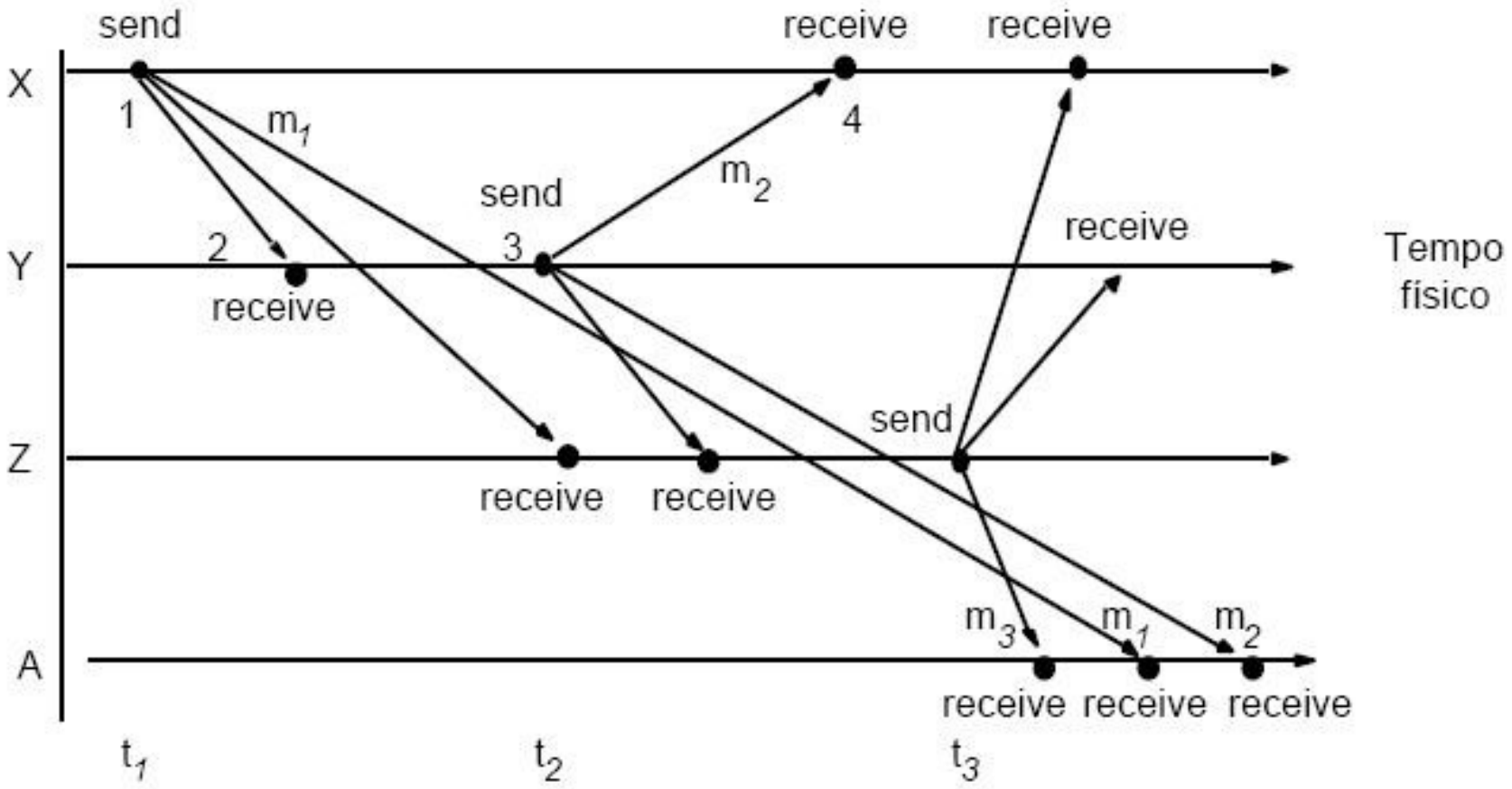
Ordenação de Eventos

- **Tentativa de solução:**
  - **Embutir a hora local do** processo remetente em cada mensagem enviada. Assim, as mensagens recebidas poderiam ser ordenadas pelo processo de destino, com **base no valor da hora embutido** em cada uma
  - **Problemas?**

Ordenação de Eventos

- Como é impossível sincronizar relógios perfeitamente em um sistema distribuído
  - *Lamport* propôs a noção de **tempo lógico** como mecanismo de ordenação de eventos em sistemas distribuídos
  - O conceito de tempo lógico permite estabelecer uma ordem parcial entre eventos ocorridos em processos executando em diferentes computadores, **sem a necessidade de recorrer a relógios "reais"**
  - Com base na relação de **causa-e-efeito** entre eventos

Ordenação de Eventos



Modelo de Falha

- Descreve as maneiras através das quais podem ocorrer **falhas nos processos e canais de comunicação,** de modo a facilitar o entendimento dos seus efeitos no sistema
- ***Falhas por Omissão***
  - Processo: processo termina inesperadamente
    - ***Fail-stop:*** *outros processos podem detectar a falha com certeza (através de timeouts): requer sistema síncrono*
  - Canal: mensagens enviadas não são recebidas

Modelo de Falha

- ***Falhas Arbitrárias***
  - *Um processo ou canal arbitrariamente omite* passos de processamento que deveriam ser executados, ou executa passos não intencionados (mais grave tipo de falha, também conhecido como **falha bizantina)**
  - *Qualquer tipo de erro pode acontecer:*
    - *Canal: mensagens podem ser omitidas, alteradas, duplicadas, etc*
    - *Processo: passos em um algoritmo podem ser omitidos...*
      - ***Não é possível detectar a falha***
- ***Falhas de Temporização*** *(apenas para o modelo síncrono)*
  - *Um* processo ou canal não atende os limites de tempo que lhes são estabelecidos (geralmente causada pela sobrecarga dos processos ou da rede)
  - Violação das suposições sobre a temporização de eventos em sistemas síncronos

Falhas por Omissão e Arbitrárias

Classe	*Afeta*	*Descrição*
Falha-e-pára	Processo	Processo interrompe sua execução em definitivo. Outros processos podem vir a detectar este estado.
Pane	Processo	Processo interrompe sua execução em definitivo. Outros processos podem não ser capazes de detectar este estado.
Omissão	Canal	Uma mensagem inserida no *buffer* de saída do remetente nunca chega ao *buffer* de entrada do destinatário.
Omissão-envio	Processo	Um processo completa um *envio*, mas a mensagem não é inserida no seu *buffer* de saída.
Omissão-receb.	Processo	Uma mensagem é inserida no *buffer* de entrada de um processo, mas o processo não a recebe.
Arbitrária (Bizantina)	Processo ou canal	Processo/canal exibe um comportamento arbitrário: envio ou recebimento de mensagens arbitrárias em momentos arbitrários; omissões; interrupção ou ação incorreta de um processo.

Falhas de Temporização

Classe	*Afeta*	*Descrição*
Relógio	Processo	Relógio local do processo excede os limites de sua taxa de desvio do tempo real.
Desempenho	Processo	Processo excede os limites do intervalo esperado entre dois passos.
Desempenho	Canal	A transmissão de uma mensagem atrasa além do limite estabelecido.

Lidando com falhas

- Mascaramento de falhas
  - Por exemplo, utilizando múltiplos servidores replicados para ocultar a falha de uma das réplicas
- Confiabilidade da comunicação
  - Detecção de mensagens com erro (CRCs)
  - Detecção de mensagens perdidas e retransmissão
  - Detecção de mensagens duplicadas

Modelo de Segurança

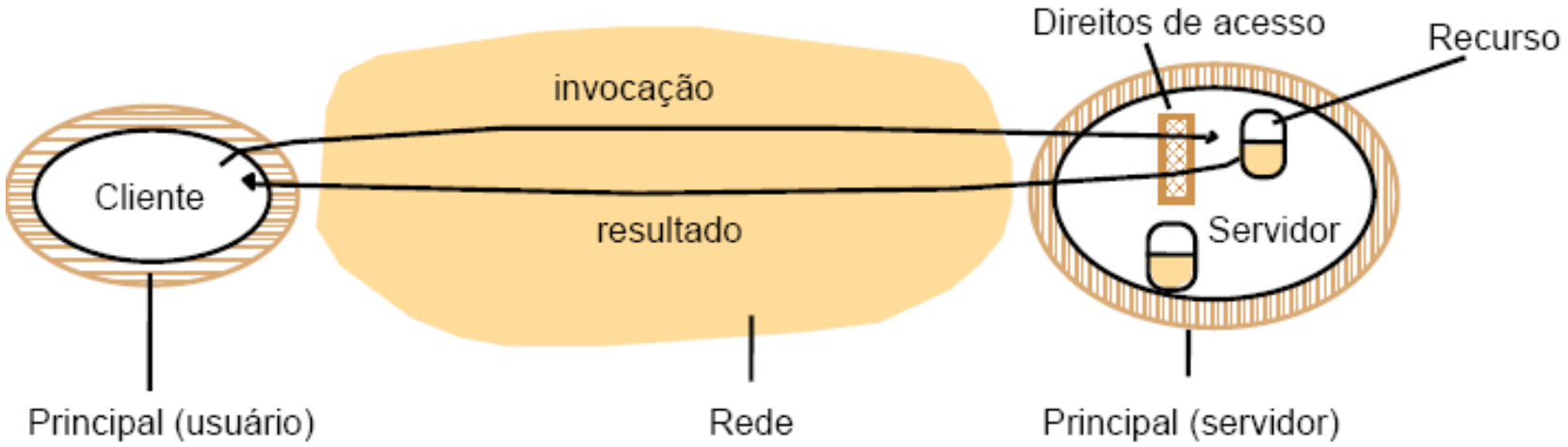
- Descreve os mecanismos utilizados para garantir a **segurança dos processos e de seus canais de comunicação, e para proteger os recursos que os** processos encapsulam contra acessos não autorizados
- Principais questões:
  - Proteção dos recursos
  - Segurança dos processos e de suas interações
- Desafios:
  - Uso de técnicas de segurança implica em **custos substanciais de processamento e de gerência**
- Necessidade de uma **análise** cuidadosa das possíveis fontes de ameaças, incluindo **ambientes externos** ao sistema (rede, físico, humano, etc)

"Encryption software is expensive...so we just rearranged all the letters on your keyboard."

Proteção de Recursos

- Conceitos envolvidos:
  - ***Direitos de acesso –*** *especificação de quem pode* realizar as operações disponíveis para um determinado recurso
  - ***Principal –*** *entidade (usuário ou processo) autorizada* para solicitar uma operação, ou para enviar os resultados de uma operação para o principal solicitante
- Responsabilidade compartilhada entre clientes e servidores
  - Servidor **verifica a identidade** do principal por trás de cada invocação e **checa** se ele tem direitos de acesso suficientes para realizar a operação solicitada
  - Cliente **verifica a identidade** do principal por trás do servidor para **garantir** que os resultados vêm do servidor requisitado

Proteção de Recursos



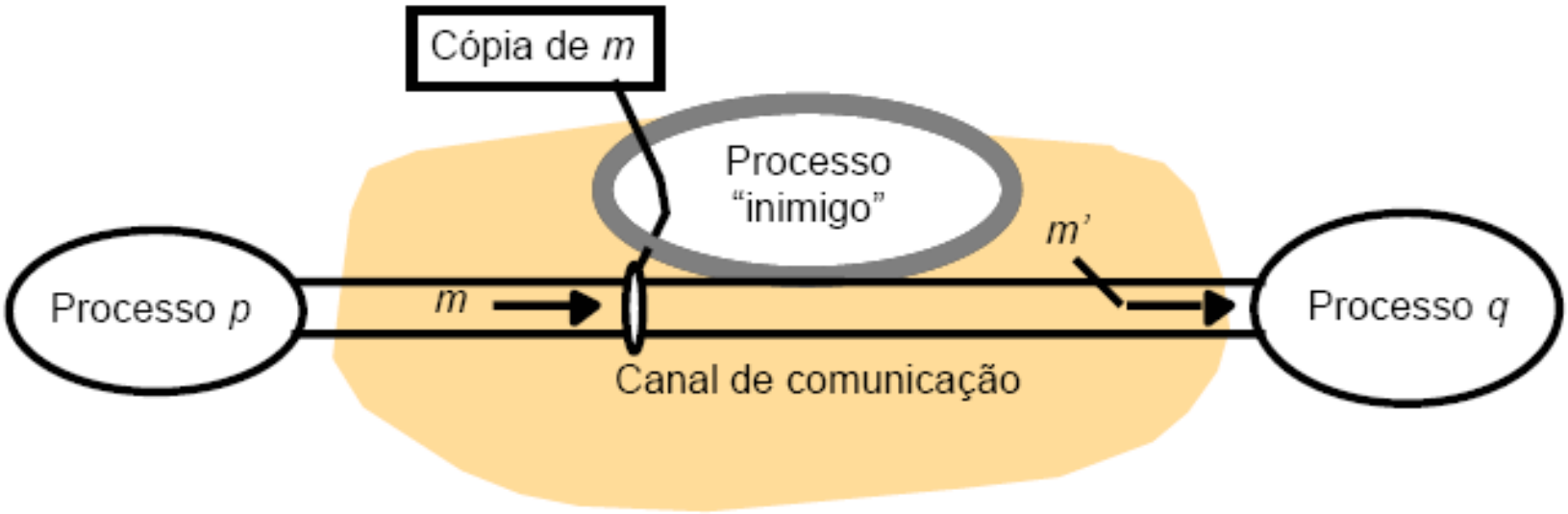
Modelo de Segurança

- Medidas de proteção aplicadas a Objetos (ou processos)
  - Direitos de acesso às interface dos objetos
  - Identidade dos objetos (*principals) e autenticação*
- Canais de comunicação
  - Encriptação dos dados transmitidos
  - Modelo de canal seguro
    - Autenticação dos objetos que se comunicam através do canal
    - Privacidade e integridade dos dados
    - Marcas de tempo nas mensagens para impedir *replay*

Segurança dos Processos e Interações

- **Natureza** aberta da rede e dos serviços de comunicação **expõe** os processos a ameaças e ataques "**inimigos"**
- Principais tipos de ameaça:
  - ***Aos processos***
    - *cliente e servidores podem não* ***ser*** capazes de determinar a identidade dos processos com os quais se comunicam
  - ***Aos canais de comunicação***
    - *mensagens podem ser* indevidamente copiadas, alteradas, forjadas ou removidas enquanto transitam pela rede
  - ***Negação de serviço***
    - *envios excessivos de mensagens ou* invocações de serviços através da rede, resultando na sobrecarga dos recursos físicos do sistema e prejudicando seus usuários
  - ***Mobilidade de código***
    - *ameaças disfarçadas na forma de* código móvel que deve ser executado localmente pelos clientes ("Cavalo de Tróia")

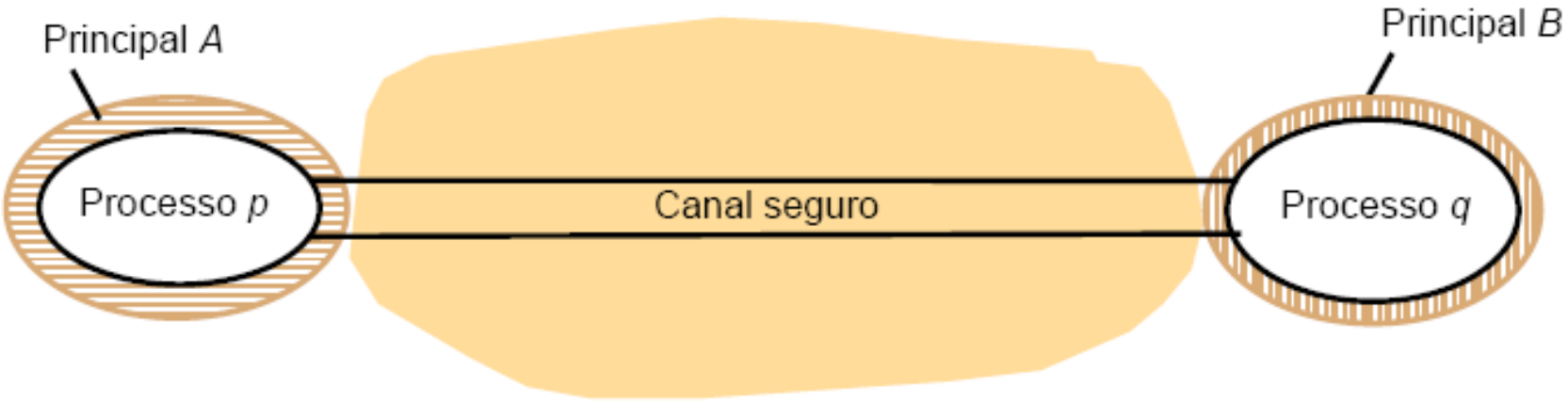
Segurança dos Processos e Interações



Comunicação Segura

- Principais mecanismos:
  - **Criptografia –** processo de **"embaralhamento"** de uma mensagem de modo a esconder seu conteúdo de usuários não autorizados
  - **Autenticação –** utiliza **chaves secretas e criptografia** para garantir a identidade dos processos clientes e servidores
  - **Canal seguro –** *canal de comunicação conectando um par* **de processos**, onde cada processo conhece e confia na identidade do principal em nome do qual o outro processo está executando
    - Geralmente implementado como uma **camada extra de serviço** sobre os serviços de comunicação existentes
    - Utiliza mecanismos de **autenticação e criptografia** para garantir a privacidade e a integridade das mensagens transmitidas através do canal
    - Também pode **garantir a entrega e ordem** de envio das mensagens
    - Ex.: VPN, SSL

Comunicação Segura



Exercícios

1) Considere um servidor simples que responde a requisições dos clientes sem acessar outros servidores

- Explique porque normalmente não é possível estabelecer um limite do tempo que se leva para que tal servidor envie a resposta ao cliente
- O que precisaria ser feito para fazer com que o servidor seja capaz de executar as requisições em um tempo limite? Isso é uma opção prática?

Exercícios

2) Considere dois serviços de comunicação para uso em um sistema distribuído assíncrono. No serviço A, as mensagens podem ser perdidas, duplicadas ou chegar atrasadas e os *checksums são aplicados* somente ao cabeçalho. No serviço B, as mensagens podem ser perdidas, atrasadas ou entregues tão rapidamente que o receptor não consegue manipulá-las, mas elas são entregues em ordem e com o conteúdo correto.

- Descreva as classes de falhas de cada um dos serviços.
- O serviço B pode ser descrito como um serviço de comunicação confiável?