Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Arquiteturas de Sistemas Distribuídos

Componentes e seus relacionamentos

- Componentes em computação distribuída:
  - Processos, recursos compartilhados (objetos, componentes)

- Relação entre os componentes
  - Invocação de serviço, coordenação entre componentes, compartilhamento, replicação

- Requisitos comuns em SD:
  - Desempenho, escalabilidade, concorrência, extensibilidade, heterogeneidade, etc.

- **Paradigmas de comunicação**
  - Comunicação entre processos
  - Invocação remota
  - Comunicação indireta

Paradigmas de comunicação

- Comunicação entre processos
  - Suporte de baixo nível de comunicação: API oferecida por protocolos de Internet (ex. programação com *sockets*)

- Invocação remota
  - Baseada na invocação e resposta: protocolos *request-reply*
    - *Remote Procedure Calls* *(Birrell e Nelson, 1984)*
    - *Remote Method Invocation*

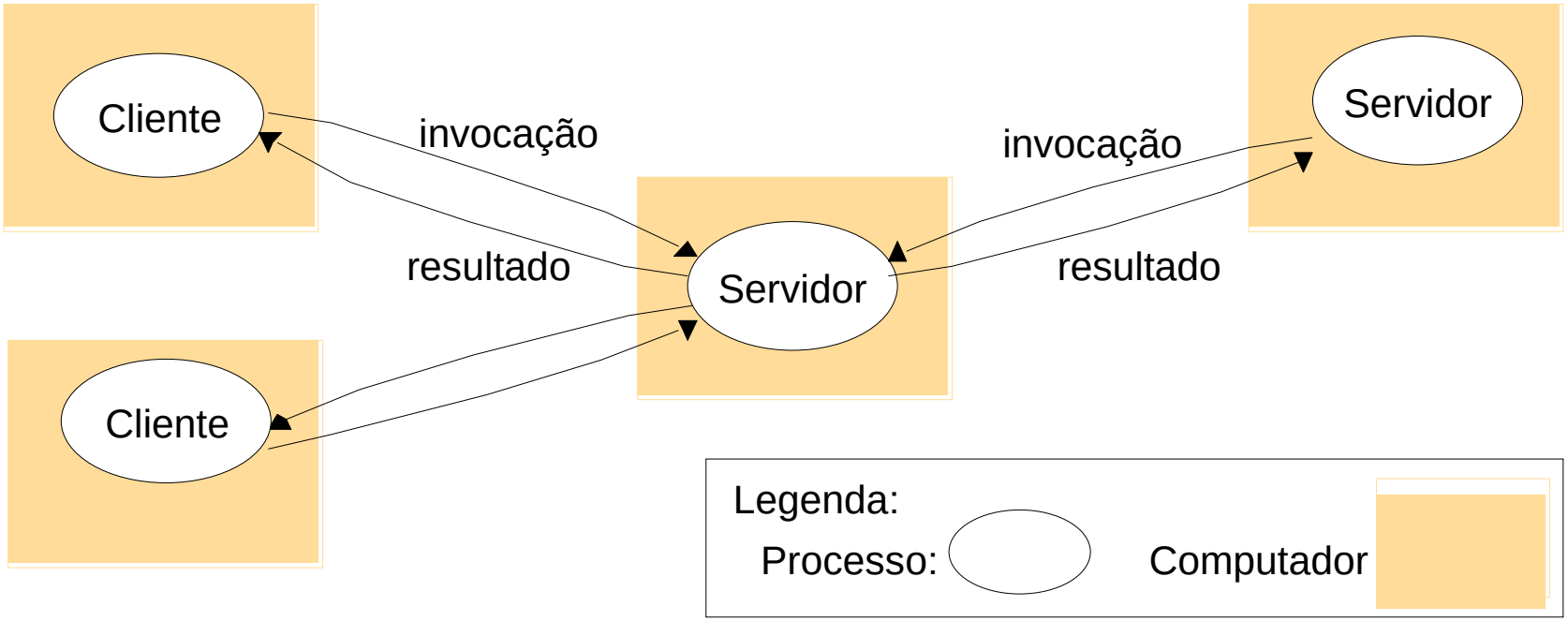
- Comunicação indireta
  - Maior grau de abstração e garantias extras são oferecidas
    - Comunicação em grupo
    - Sistema *publish-subscribe*
    - Filas de mensagens
    - Espaço de tuplas
    - Memória compartilhada distribuída

Arquitetura Cliente-Servidor

- Computadores executam processos servidores que oferecem algum serviço

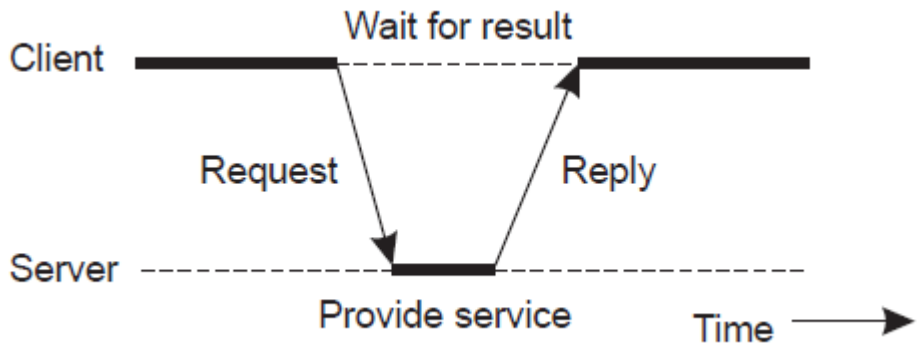
- Outros computadores executam processos clientes que podem enviar requisições aos servidores

- Servidores podem enviar requisições a outros servidores (atuando como clientes)



Fluxo de Execução em modelo Cliente-Servidor

- Cliente envia requisição para o servidor
- Servidor atende a requisição, processando a mensagem através de um serviço específico
- Servidor envia o retorno para a requisição ao cliente

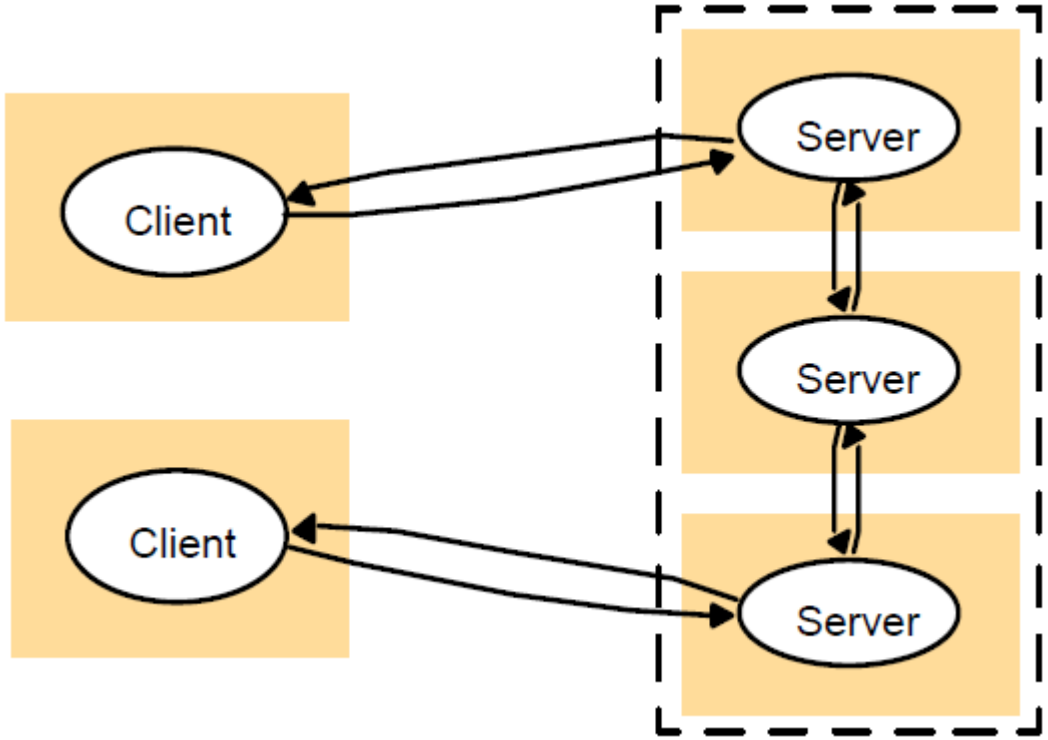


Exemplo de serviço:

- Cliente solicita o hora correta no fuso horário UTC -03:00 (*timezone)*
- Servidor recebe requisição e faz conversão necessária para atender a requisição
- Servidor envia o horário correto para o cliente

Arquitetura Cliente-Servidor com Múltiplos Servidores

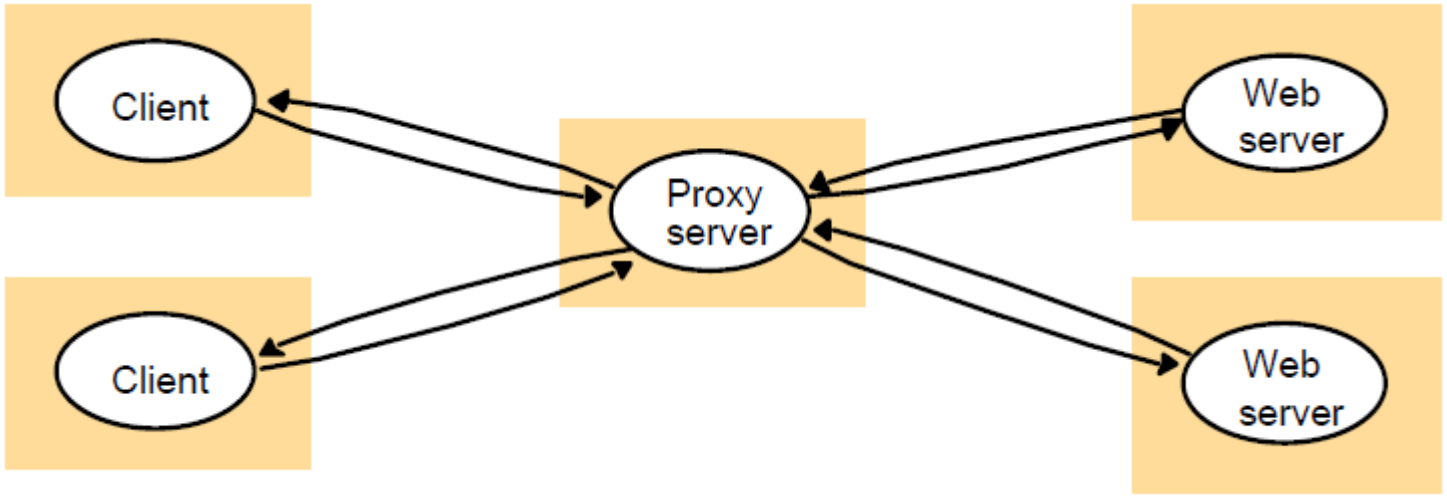
- Múltiplos servidores podem implementar as mesmas funções ou funções diferentes
  - Se implementam mesmas funções, podem estar presentes na arquitetura por questões de desempenho ou redundância
  - Se implementam funções diferentes, a solução é proposta de forma modular e servidores podem inclusive prover serviços para outros servidores



Servidores Proxy e Cache

- Servidores podem atuar como *proxies* entre clientes e servidores
  - Proxy implementa um serviço intermediário (entre o remetente e destinatário)
  - Exemplo: os *proxies* podem aplicar políticas de segurança, fazer cache de conteúdo, etc.

- Servidores ou dispositivos responsáveis por balanceamento de carga podem ser posicionados entre os clientes e servidores
  - Objetivo é equilibrar o número de requisições entre os servidores disponíveis, de modo a não sobrecarregar nenhum deles



Código Móvel e migração

- O código migra de uma máquina para outra:
  - Código é transferido de A para B e é executado em B
  - Após migração, código pode estabelecer contato com servidor de origem
  - Exemplos: *Applets* e ActiveX (depreciados), Javascript, Ajax, *Worms*, máquinas virtuais e contêineres

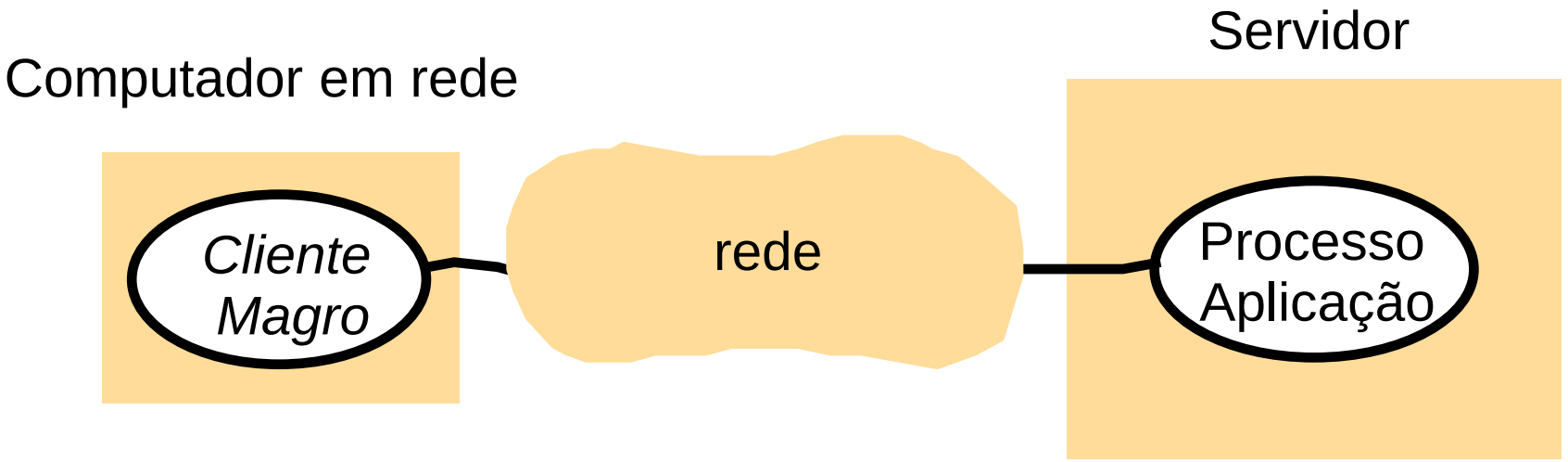
- Agentes móveis mantêm dados e código móveis, sendo comum sucessivas migrações entre nodos distribuídos
  - Agentes monitores/inspetores em sistemas de manufatura ou redes de sensores

- Migração de máquinas virtuais em computação ambientes elásticos (ex. computação em nuvem)

- **Desafios**:
  - Segurança
  - Portabilidade e independência de plataforma (é importante que o código não necessite ser recompilado após a migração)
  - Complexidade de desenvolvimento perante os desafios dos ambientes distribuídos (no caso de agentes móveis e migração de VMs)

Clientes Magros (*Thin Client*)

Estação de trabalho em rede que segue o modelo cliente servidor. Esta estação tem poucos ou nenhum aplicativo instalado, sendo que aplicações executam sempre no servidor.



Vantagens:

- Baixo custo (HW e SW)
- Fácil manutenção
- Proteção

Desvantagens:

- Altamente dependente do servidor
- Exige boa largura de banda

Servidor Multicamadas

1 Camada (*Single-Tiered*)
Terminal burro – a máquina do cliente apenas faz *login* em um servidor remoto.
Toda a aplicação reside no servidor

2 Camadas (*Two-Tiered*)
Configuração cliente/servidor com servidor único (uma camada de serviço)

Multi-camadas (Three-Tiered)
Cada camada em servidores separados

Arquitetura *Single-Tier*

Vantagens

Segurança, controle, gerenciamento

Desvantagens

- Dificuldade de reutilização dos componentes de aplicação
- Servidor é ponto único de falha

Arquitetura *Two-Tier*

Vantagens

- Clientes são independentes, podendo apresentar camadas de aplicação diferentes.
- Aproveitamento do poder computacional da máquina cliente

Desvantagens

- Servidor precisa tratar conexões de vários clientes
- Lógica da aplicação “amarrada” aos dados

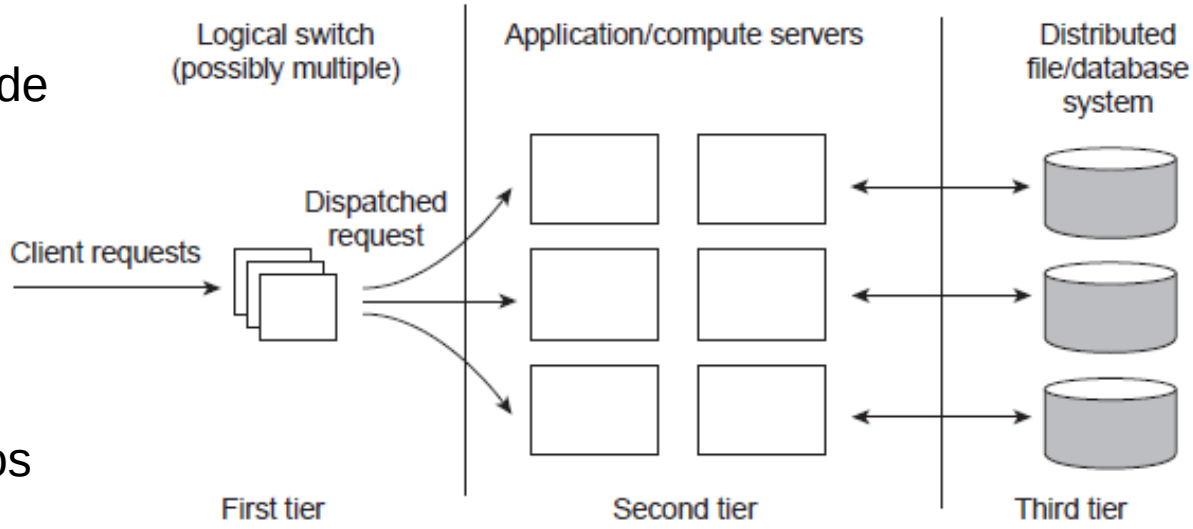
Servidor com Múltiplas Camadas

Primeira Camada

Interface com o usuário, esta camada é a camada de apresentação para o usuário, possivelmente replicada com vários servidores oferecendo balanceamento de carga.

Segunda Camada

Camada de processamento, possui as funções (componentes) da aplicação, porém não mantém os dados persistidos nesta camada.



Terceira Camada

Camada de dados, onde os dados são armazenados e manipulados através dos componentes da segunda camada.

Exemplo de Servidor com 3 Camadas (*Three-Tiered*)

Primeira Camada

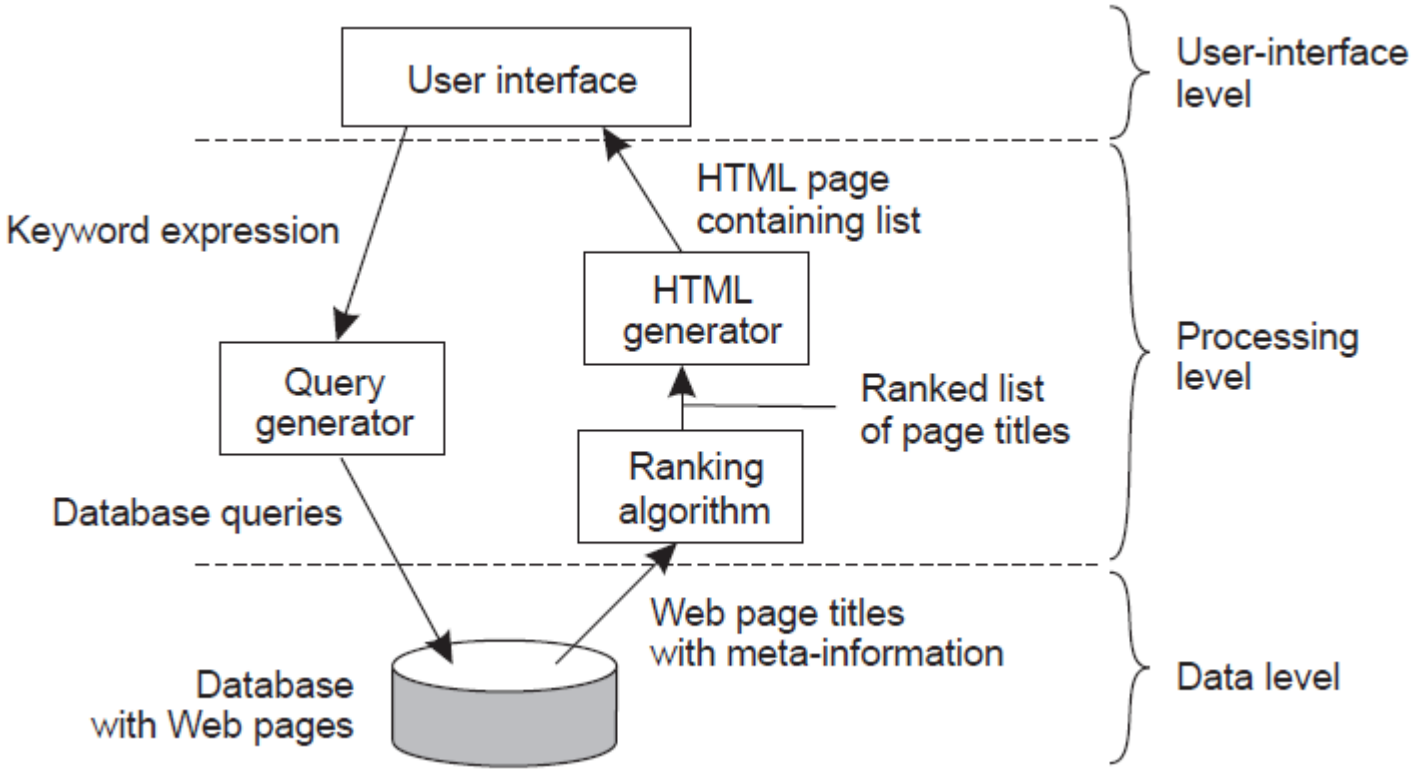
Oferece a interface de busca para o usuário que acessa a aplicação.

Segunda Camada

Contém métodos (ou componentes) para a geração das consultas e implementa funções de classificação dos resultados. Há também a implementação do mecanismo de geração de HTML dinâmico, baseado no resultado das buscas.

Terceira Camada

Mantém conteúdo das páginas Web da aplicação e mantém metadados com conteúdo usado nas pesquisas.



Exemplo de Servidor com 3 Camadas (*Three-Tiered*)

Primeira Camada

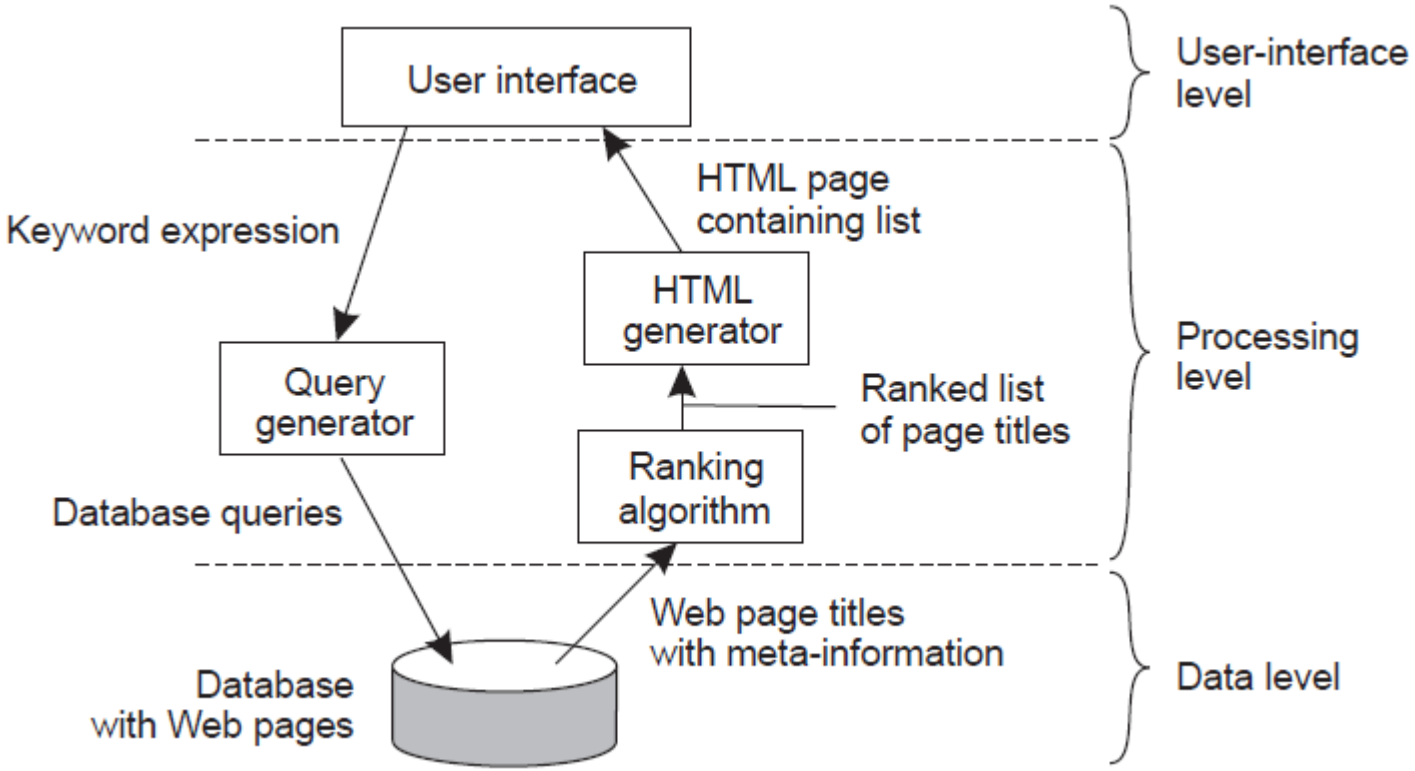
Oferece a interface de busca para o usuário que acessa a aplicação.

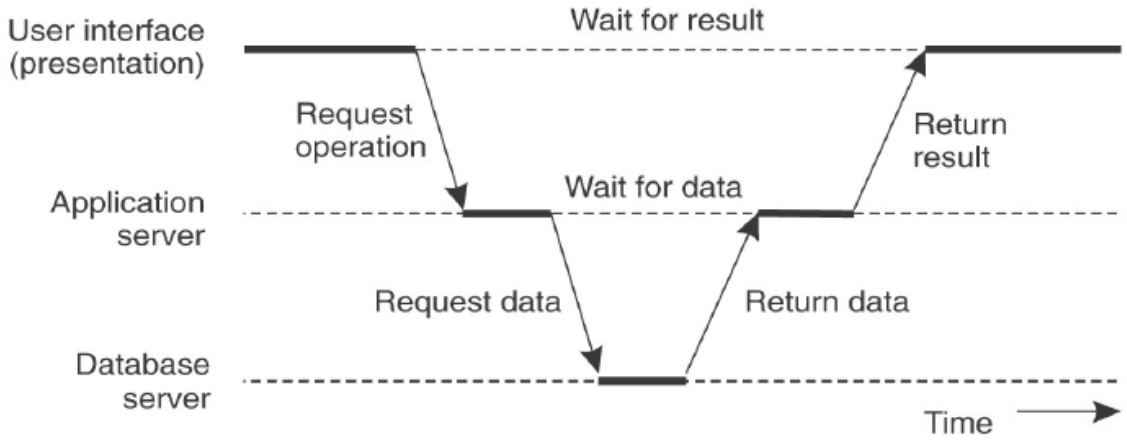
Segunda Camada

Contém métodos (ou componentes) para a geração das consultas e implementa funções de classificação dos resultados. Há também a implementação do mecanismo de geração de HTML dinâmico, baseado no resultado das buscas.

Terceira Camada

Mantém conteúdo das páginas Web da aplicação e mantém metadados com conteúdo usado nas pesquisas.





Arquiteturas Par a Par (P2P – *Peer-to-Peer*)

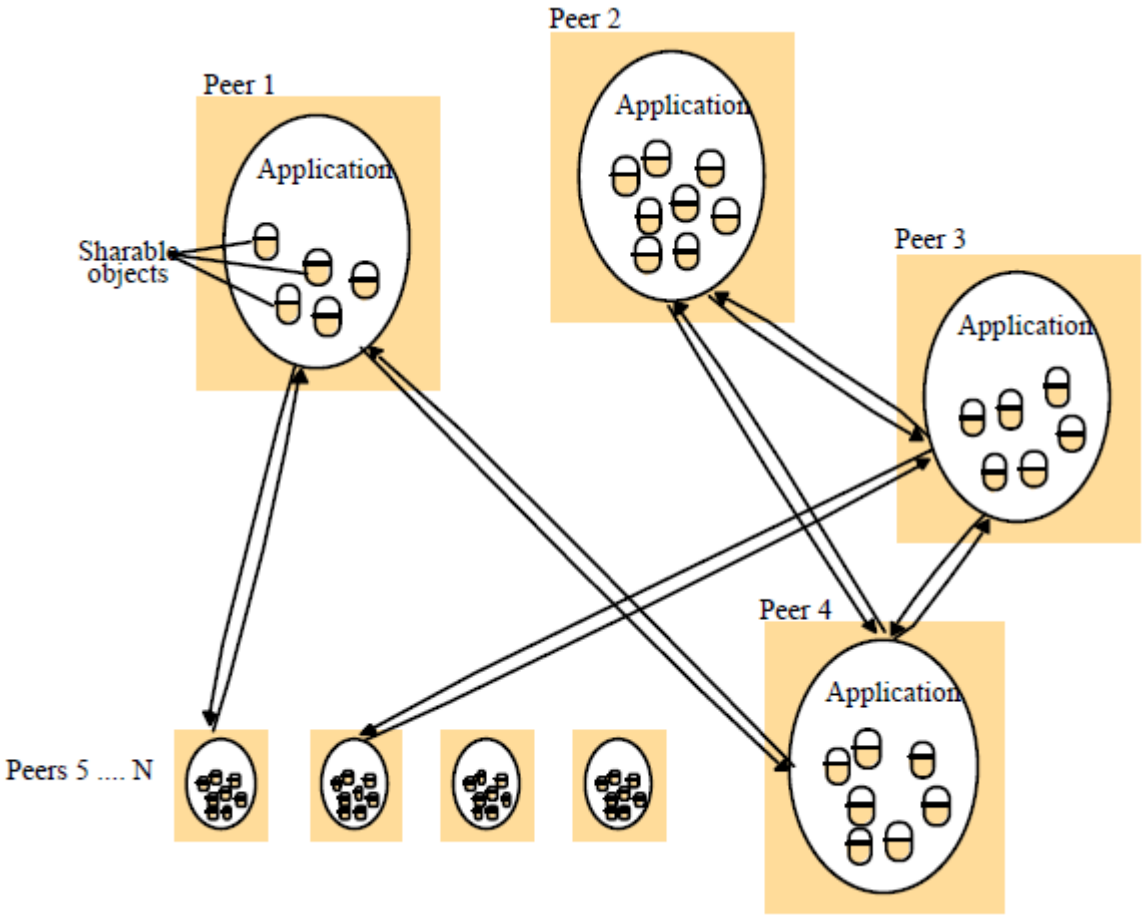
- Arquiteturas descentralizadas compostas de vários nodos capazes de compartilhar recursos

- Não há diferenciação entre clientes e servidores. Neste modelo de arquitetura, os nodos atuam como ambos

Primeiros sistemas distribuídos com enfoque em escalabilidade em larga escala em termos no dós participantes

Técnicas de p2p são empregadas em diversos outros sistemas, como key-value stores:

- Cassandra, Riak, Voldermort usam DHT (*Distributed Hash Table*)

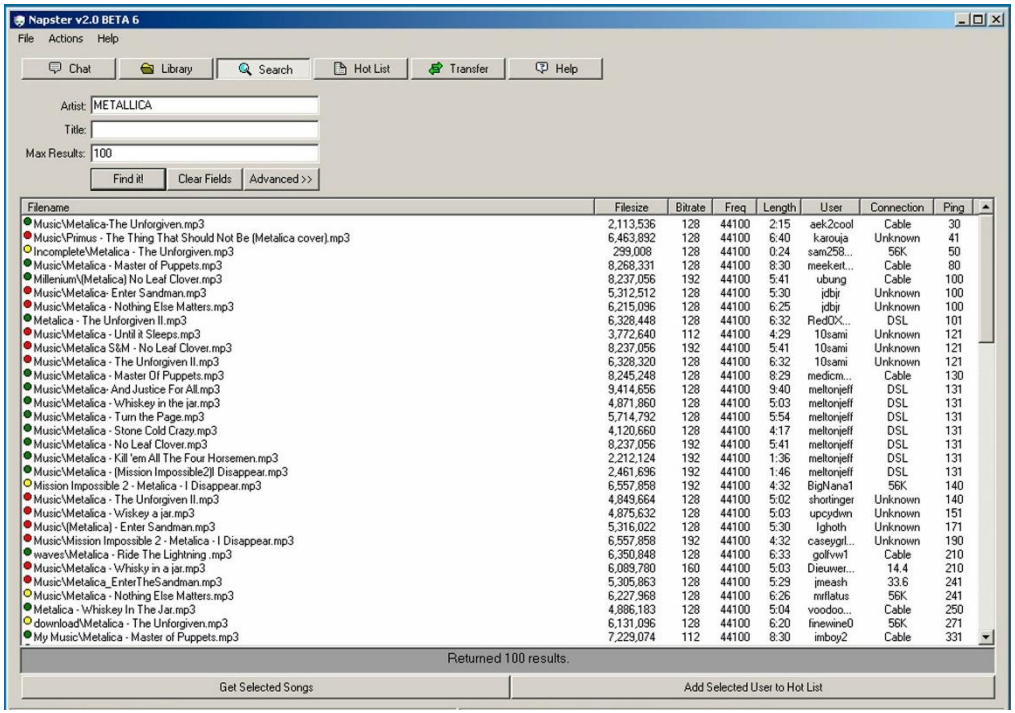


Arquiteturas Par a Par (P2P – *Peer-to-Peer*)

Breve histórico

Napster

- 1999: Shawn Fanning lança o Napster
- 2000: Grandes volumes de tráfego com transferências do Napster (ex. Univ. de Wiscosin, 25% do tráfego era Napster)
- 2001: Processos de gravadoras contra violação de direitos autorais
- 2001: Napster decide cobrar pelo serviço e pagar percentual as gravadoras



Outros usos:

- Em 2005, 3,91 milhões de PCs participaram do projeto SETI@home, com o processamento de 221 milhões de unidades de trabalho (~27,36 teraflops)

- Surgimento de outros sistemas P2P
  - 1ª geração: Napster (1999)
  - 2ª geração: Freenet, Gnutella, Kazaa (2000)
  - 3ª geração: Bit Torrent (2001)

Opennap:
Protocolo do Napster é aberto e livre para uso:
https://opennap.sourceforge.net/

Gnutella (depreciado):
https://www.gnutellaforums.com/

Arquiteturas P2P – Características

- Auto-organização: não há um coordenador central
  - Cada nodo pode adicionar ou remover recursos no sistema
  - Os nodos têm as mesmas responsabilidades funcionais

- Adaptabilidade e escalabilidade: rede se ajusta ao ambiente
  - Arquiteturas amplamente distribuídas com recursos voláteis
  - Recursos podem deixar o sistema voluntariamente ou involuntariamente

- Comunicação direta entre os pares:
  - Alternativa ao modelo cliente servidor. Apenas os pares envolvidos no compartilhamento de recursos comunicam

- Por se tratar de um ambiente aberto, é necessário prover certo grau de segurança e anonimato para os provedores e usuários de serviços

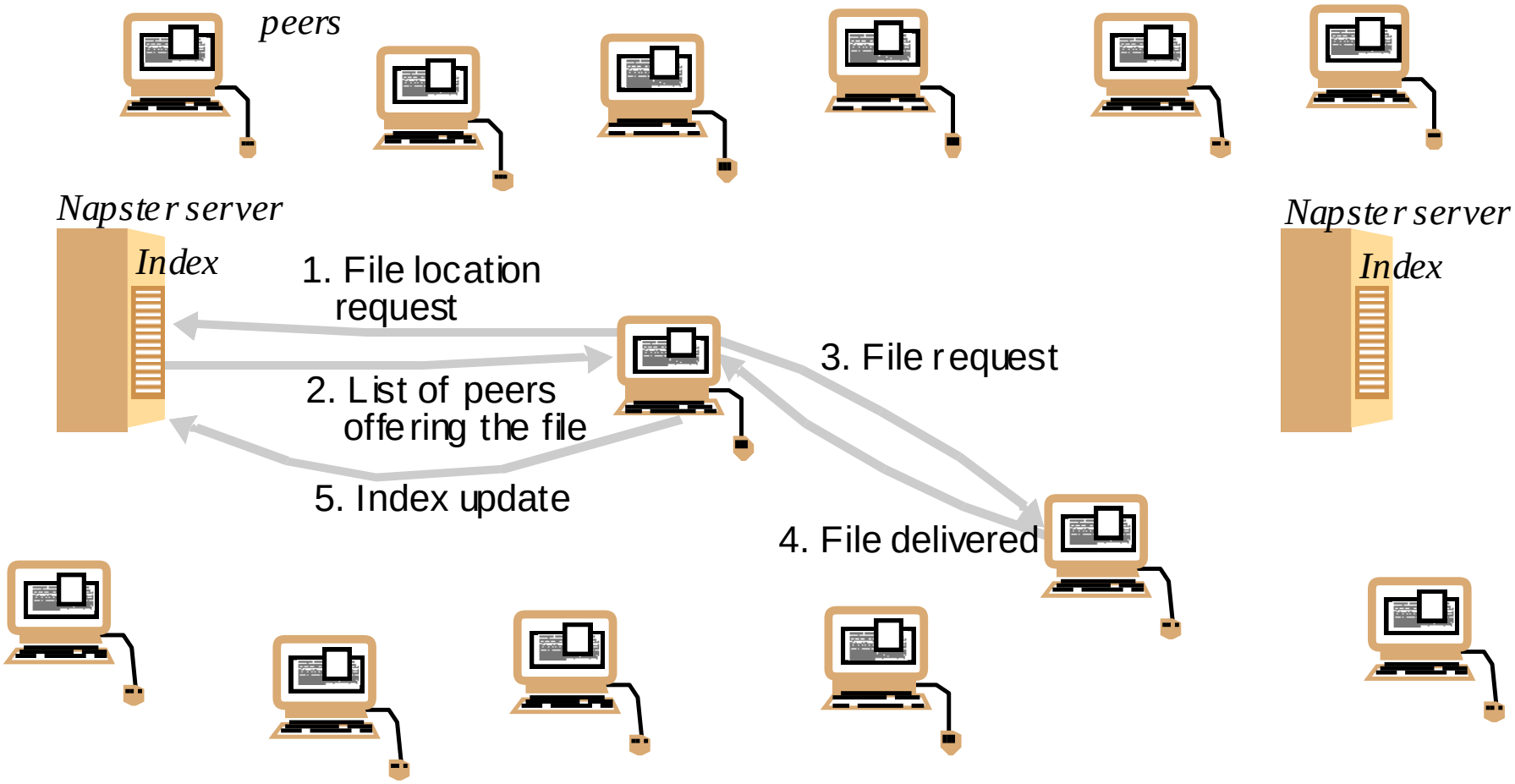
Arquiteturas P2P Estruturadas

Com infraestrutura centralizada

Utiliza serviços centralizados para controle de acesso, publicação e pesquisa

- Servidor central controla entradas e saídas na rede
- Pares registram no serviço os recursos que irão compartilhar
- Pesquisas por recursos disponíveis são efetuadas pelo servidor central
- Acesso (transferência) aos recursos é realizada pelos pares envolvidos
- Ex. Napster, eMule

Indexação e busca – Exemplo Napster



Servidor de Indexação centralizado, replicado

Arquiteturas P2P – Napster Limitações

- Servidor centralizado
  - Fonte para congestionamento
  - Ponto único de falhas
- Vulnerabilidades e falhas de segurança
  - Mensagens de texto planas com conteúdo e senhas

- Declarado judicialmente como responsável por violação de direitos autorais:
  - Indiretamente permitia as infrações

- Sistemas da 2ª geração trataram estes problemas:
  - Gnutella, Kazaa, etc.
  - Elimina servidores centrais (clientes atuam como servidores /serventes)

Arquiteturas P2P Não-Estruturadas

Não há uma infraestrutura de controle

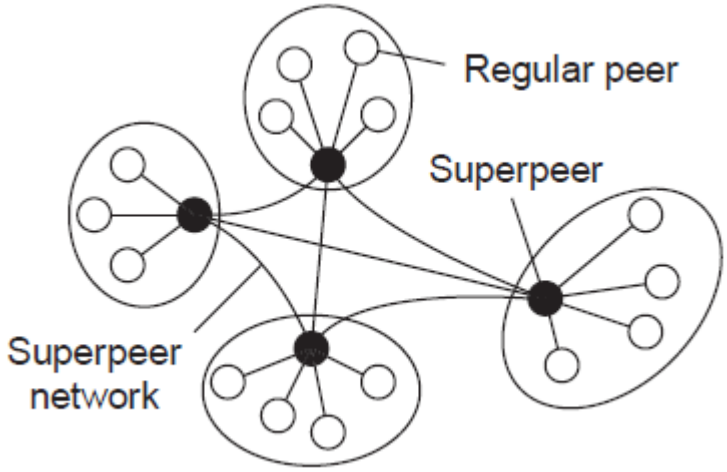
- Todos os pares possuem papel equivalente
- Pesquisas por recursos compartilhados são feitas baseadas em algoritmos de inundação (*flooding*)
- Gera um alto tráfego na rede
- Desempenho das pesquisas é ruim devido à necessidade de contactar muitos nós e aguardar a resposta

- Ex.: Gnutella1, JXTA

Arquiteturas P2P Estruturadas

Com infraestrutura distribuída

Alguns nodos para desempenharem algum trabalho específico, os ***superpeers (superpares)***



Funcionalidades dos *superpeers*

- Controlam o ingresso de nodos na rede
- Podem indexar e buscar recursos compartilhados pelos pares
- No caso de falhas, novos *superpeers* podem ser adicionados ao sistema

- Ex. Gnutella2, Skype

Indexação e busca – Exemplo Chord (e outros)

Infraestrutura distribuída

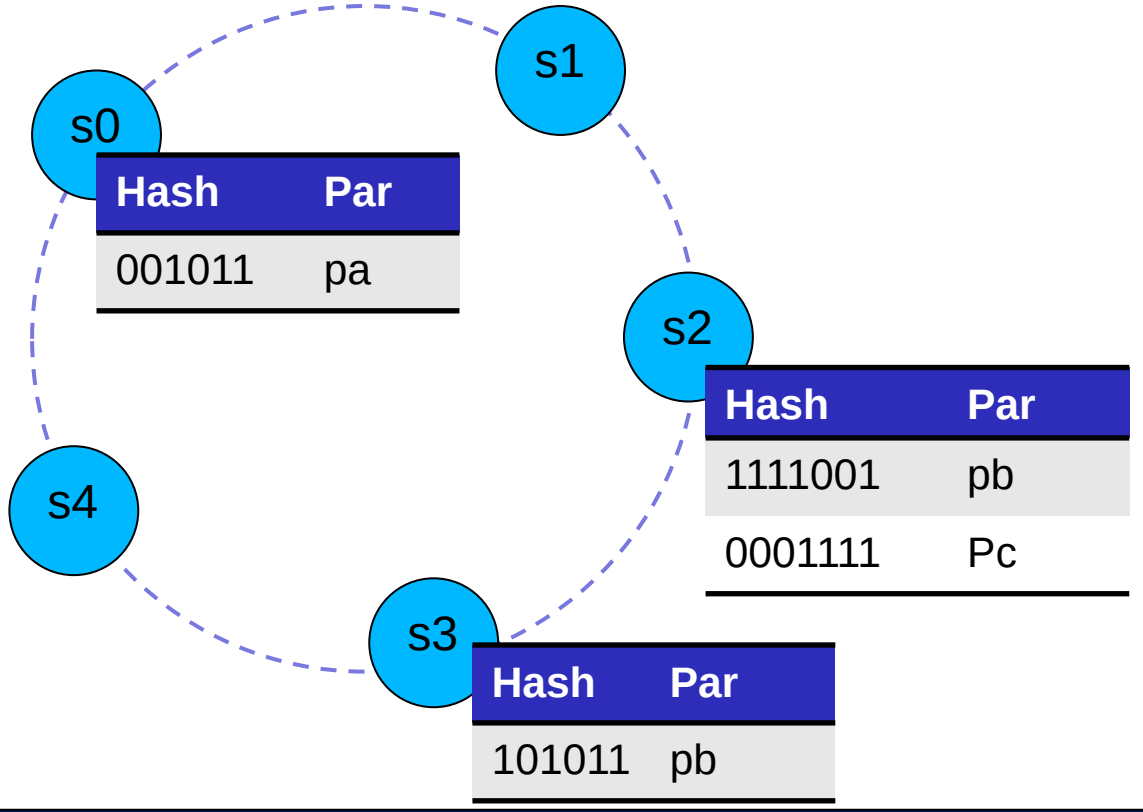
Superpares mantêm uma DHT (*Distributed Hash Table*)

- DHT mantém resumos (*hashes*) dos nomes dos arquivos compartilhados e identificação dos pares que contém o arquivo
- Cada *superpar* contém uma parte da tabela (não precisam ter a informação de todos os recursos da rede)
- *Superpares* são organizados em um anel lógico

Indexação e busca – Exemplo Chord (e outros)

Infraestrutura distribuída

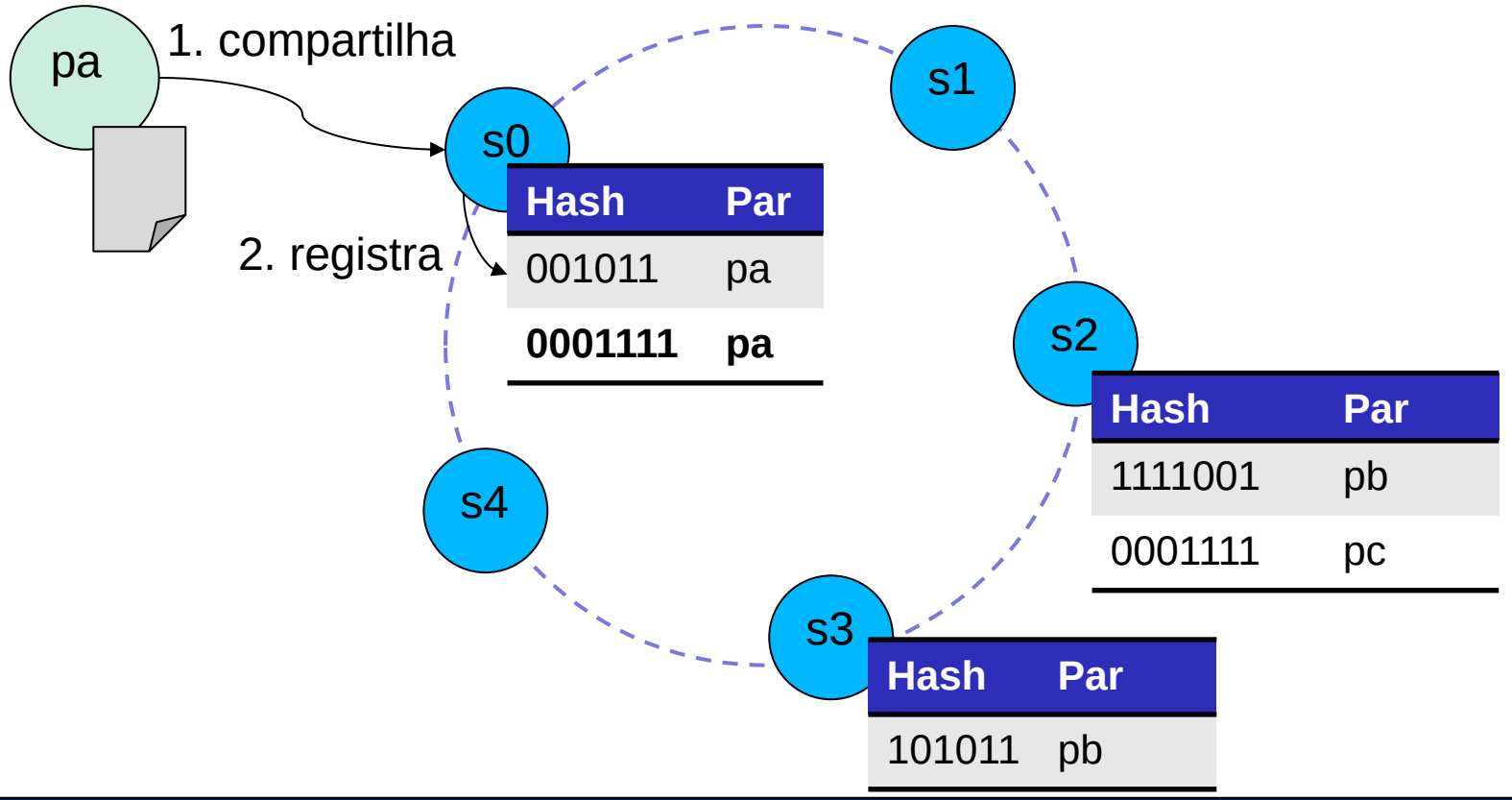
Superpares mantém uma DHT (*Distributed Hash Table*)



Indexação e busca – Exemplo Chord (e outros)

Infraestrutura distribuída

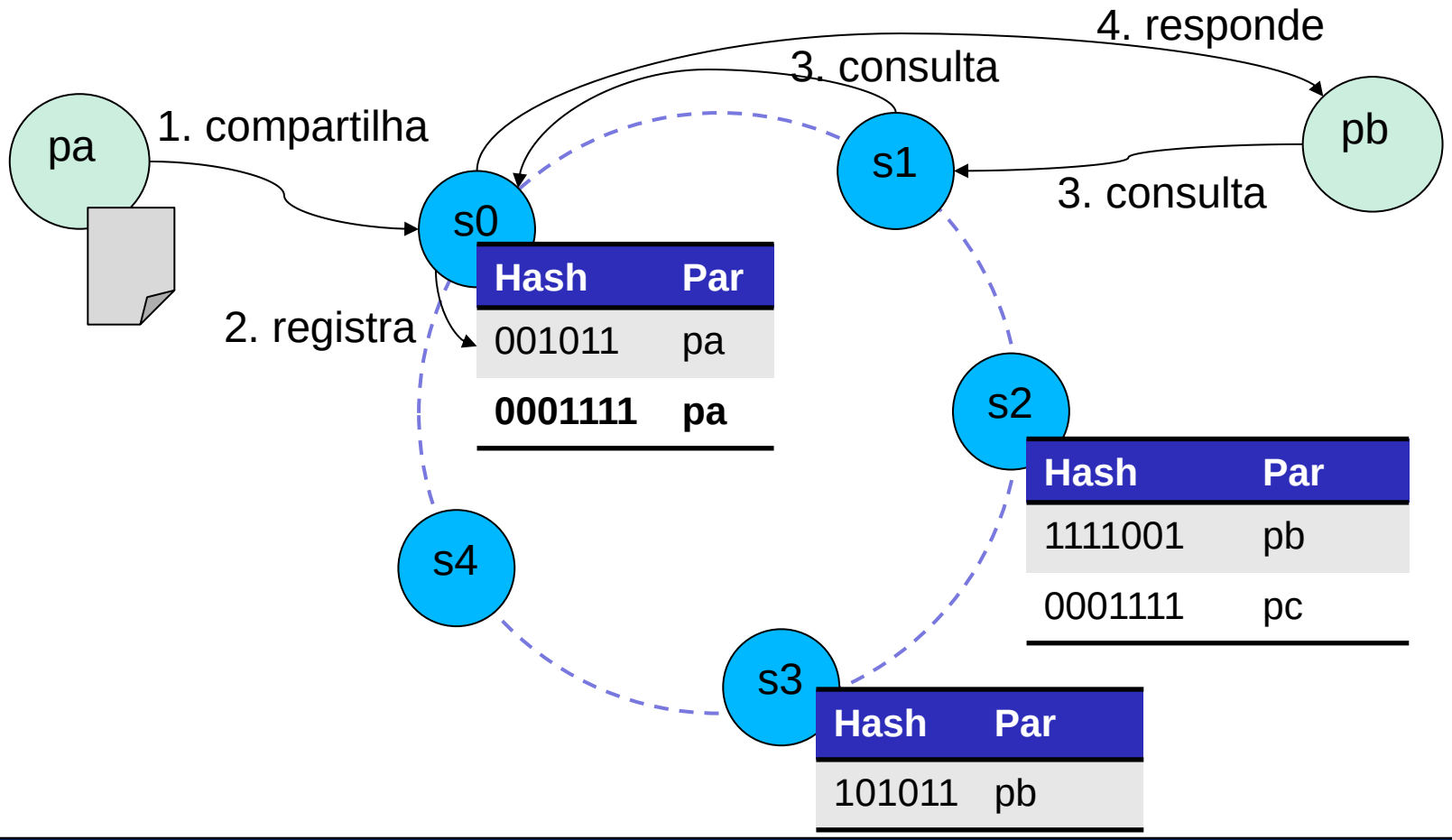
Superpares mantém uma DHT (*Distributed Hash Table*)



Indexação e busca – Exemplo Chord (e outros)

Infraestrutura distribuída

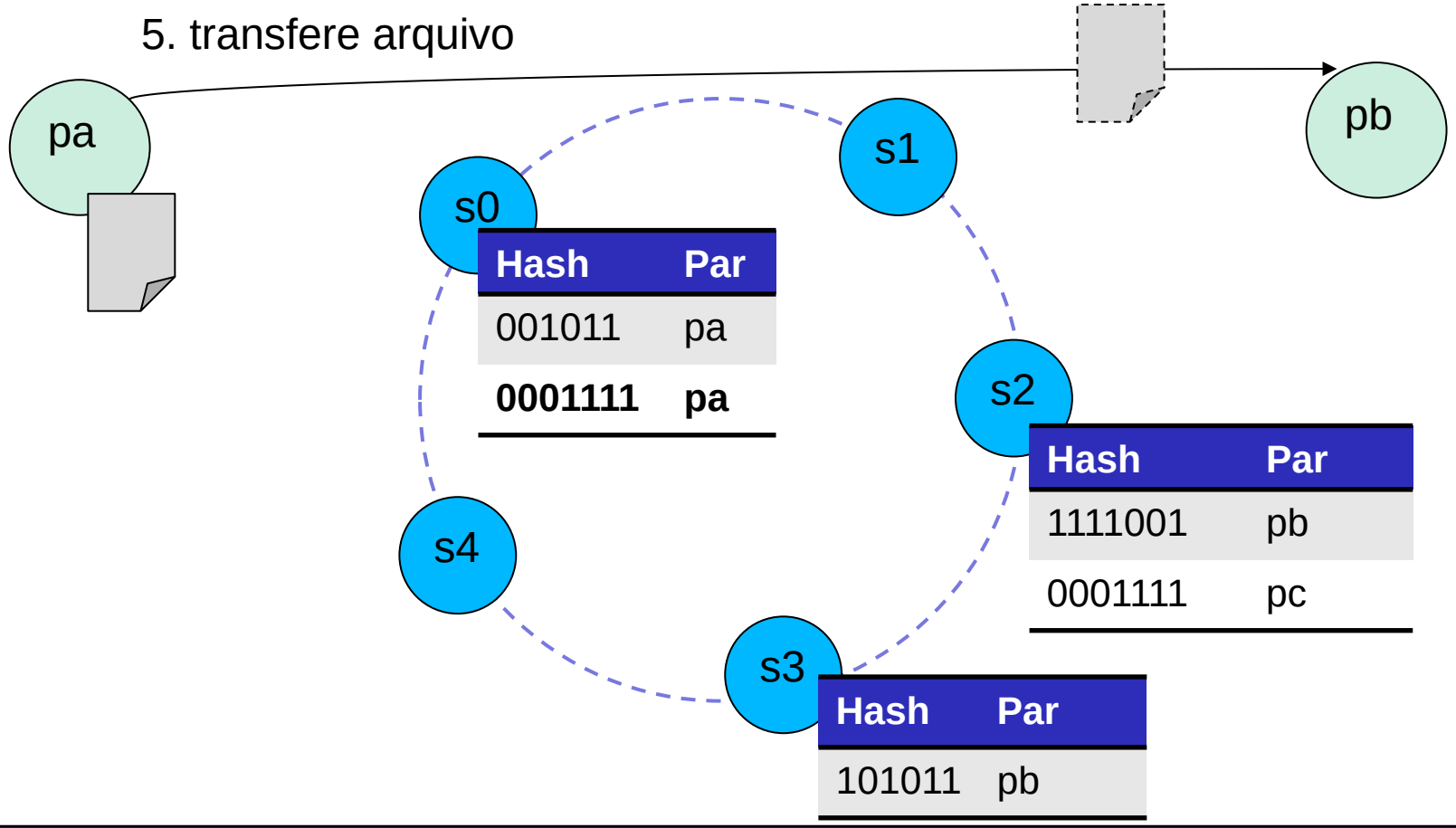
Superpares mantém uma DHT (*Distributed Hash Table*)



Indexação e busca – Exemplo Chord (e outros)

Infraestrutura distribuída

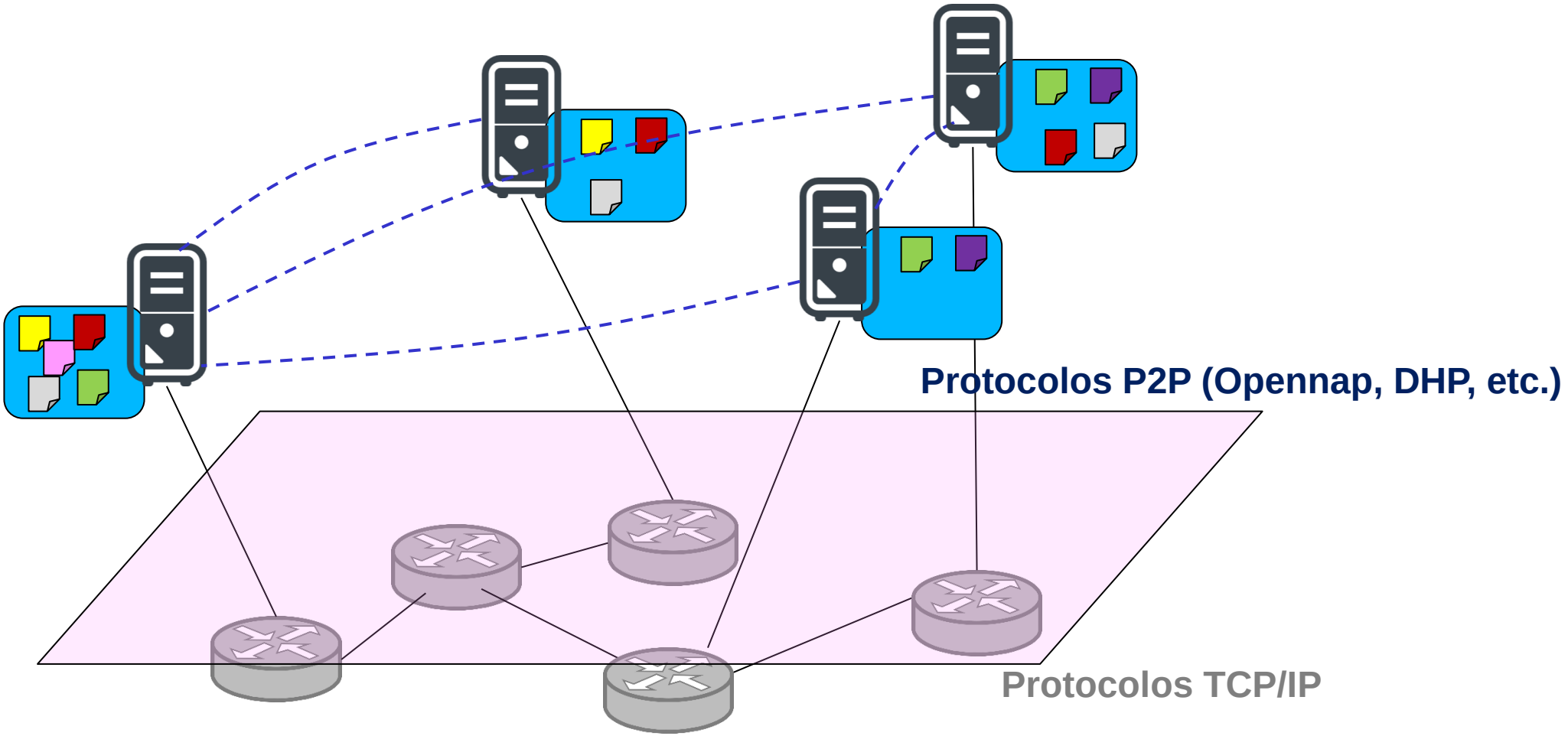
Superpares mantém uma DHT (*Distributed Hash Table*)



Super Peers – Sobreposição de Roteamento

Overlay networks: Protocolos de roteamento sobrepostos aos protocolos de roteamento da rede subjacente

Composição de um *overlay graph*



Sobreposição de Roteamento

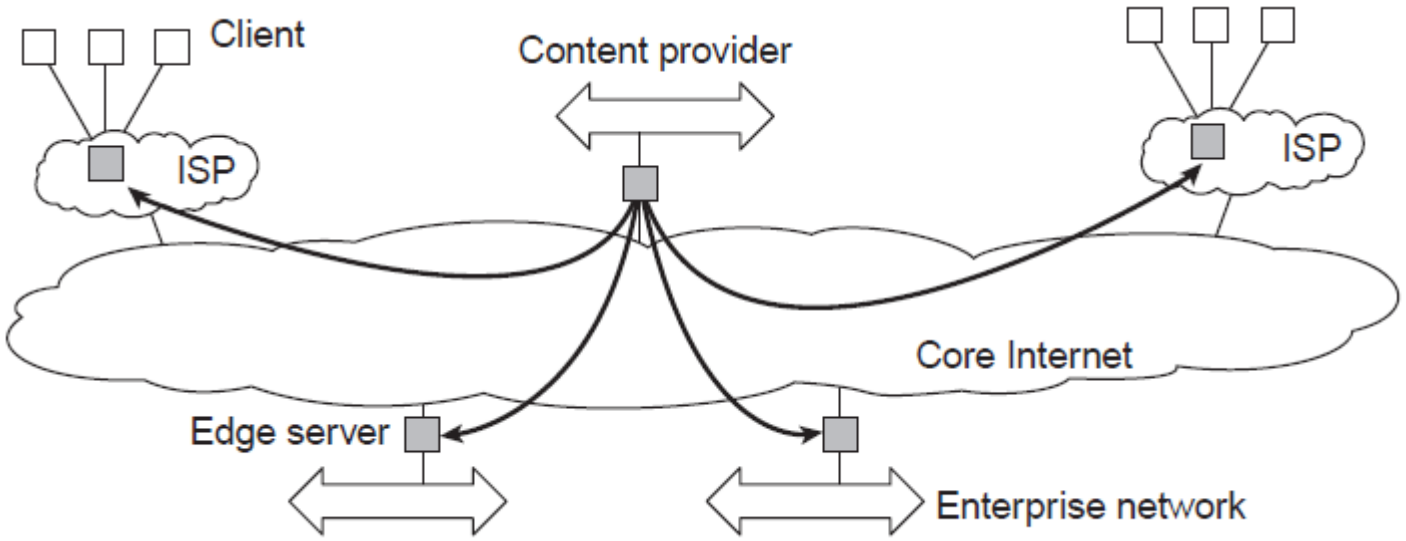
Comparação entre roteamento IP e roteamento em nível de aplicação

	IP	*Roteamento em nível de aplicação*
Escala	IPv4 limitado à 2^{32} nós endereçáveis. IPv6 amplia para 2^{128} nós. Ainda assim há uma política administrativa para a reserva dos endereços	Recursos endereçáveis por identificadores globais únicos (GUID – tamanho definido pela aplicação)
Balanceamento de Carga	Cargas sobre roteadores definidas pela topologia da rede e tráfego	Objetos podem posicionar-se aleatoriamente, tornando os padrões de tráfego independentes da topologia
Identificação do destinatário	Cada endereço IP é mapeado em exatamente 1 nodo destinatário	Mensagens direcionadas para a réplica mais próxima do objeto destino

Arquiteturas Híbridas – Cliente/Servidor + P2P

Redes de Entrega de Conteúdo (*Content Delivery Networks*)

- Provedores de conteúdo ficam geograficamente distribuídos
- Cada provedor mantém o mesmo conteúdo dos demais replicado
- Cliente acessam aplicação através de uma mesma URL
- ISP (*Internet Service Provider*) irá estabelecer a conexão do cliente com o servidor mais próximo



Arquiteturas Híbridas – Cliente/Servidor + P2P

Exemplo: Servidor de CDN da Netflix

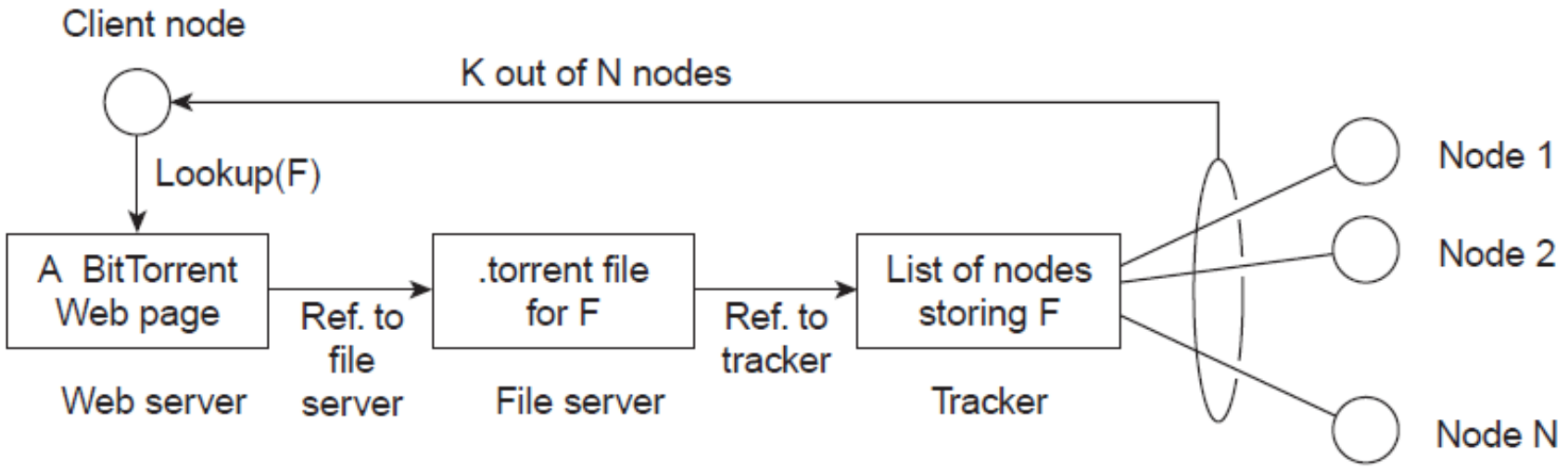
Arquiteturas Híbridas – Cliente/Servidor + P2P

BitTorrent

- Uma vez que um nó identifica onde encontra um arquivo para baixar, ele une-se a um conjunto de nós interessados no mesmo arquivo

- Cada nó baixa e mantém disponível para *download* partes do arquivo

- Dessa forma os nós cooperam entre si para obter o máximo ou todas as partes de um arquivo.



Arquiteturas Híbridas – Cliente/Servidor + P2P

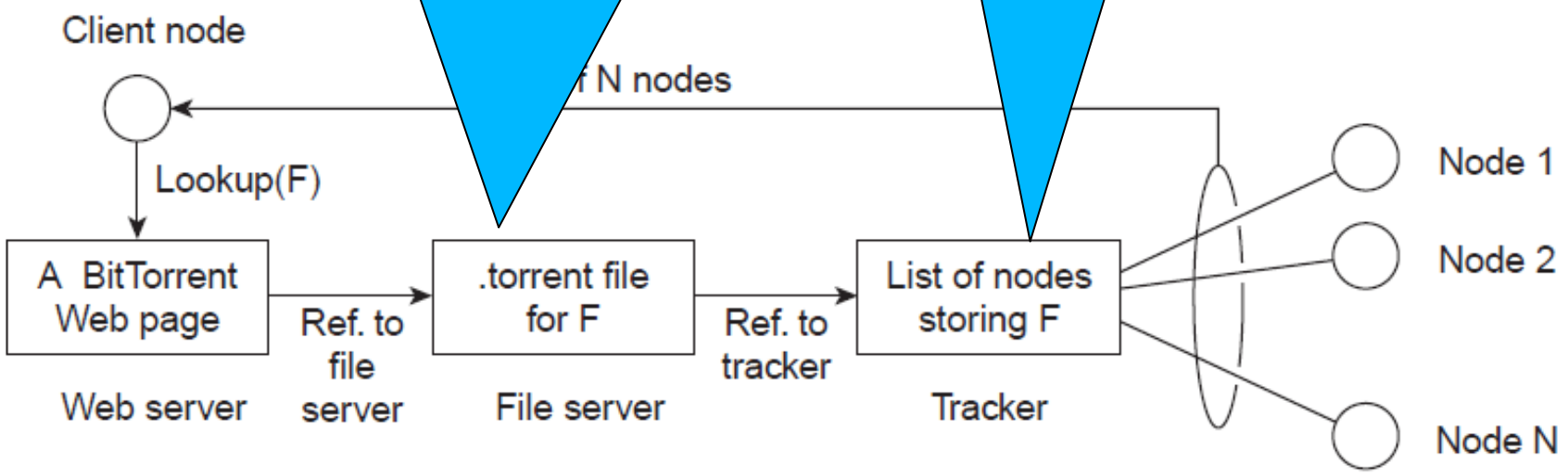
BitTorrent

Arquivo .torrent:
- Criado por nó *seed*, que compartilha arquivo
- Contém metadados que descrevem o arquivo e permitem verificar sua integridade
- Indica servidores *tracker*

Servidor tracker:
- Coordenam a distribuição de arquivos
- Indicam os nós para *download*
- Controlam a velocidade do *download*
- Se o nó não colabora (*leech*), perde banda

downl do arquivo

a obter no ou todas as partes



Referências

Parte destes slides são baseadas em material de aula dos livros:

- *Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. Sistemas Distribuídos: Conceitos e Projetos. Bookman; 5ª edição. 2013. ISBN: 8582600534*

- *Tanenbaum, Andrew S.; Van Steen, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2007. Pearson Universidades; 2ª edição. ISBN: 8576051427*