Sistemas Distribuídos

Aula 2 – Arquiteturas

DCC/IM/UFRRJ
Marcel William Rocha da Silva

Objetivos da aula

- **Aula anterior**
  - Introdução e conceitos básicos
    - *Middleware*
  - Metas de um sistema distribuído
  - Tipos de sistemas distribuídos

- **Aula de hoje**
  - Arquiteturas de sistemas
    - Estilos arquitetônicos
    - Arquiteturas centralizadas, descentralizadas e híbridas
  - Arquiteturas vs. middleware

Arquiteturas de sistemas distribuídos

- O que é uma arquitetura?
  - Organização lógica e física dos componentes
    - Como estão interconectados?
    - Como se comunicam?
    - Como se relacionam para formar um sistema?

- **Componentes**...
  - Unidade modular com interfaces requeridas e fornecidas bem definidas e que é substituível dentro do seu ambiente

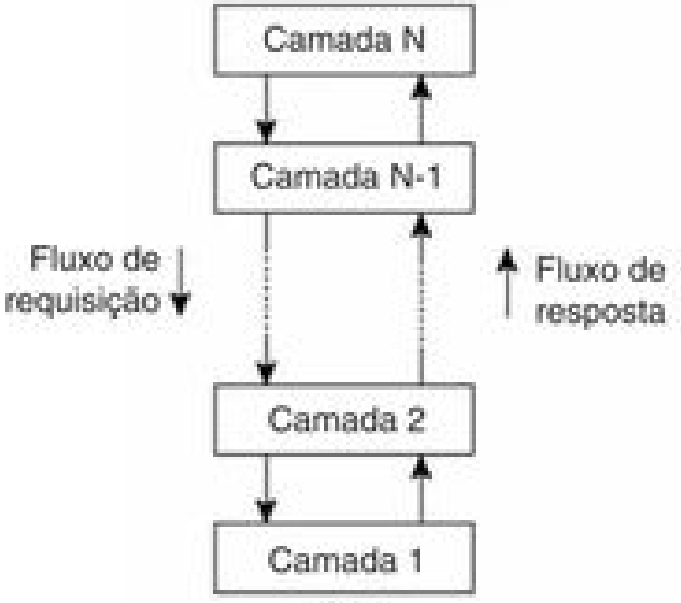
Estilos arquitetônicos

- Modelos comumente adotados por SDs

- Estilos de arquitetura:
  - **Em camadas**
  - **Baseadas em objetos**
  - **Centradas em dados**
  - **Baseadas em eventos**

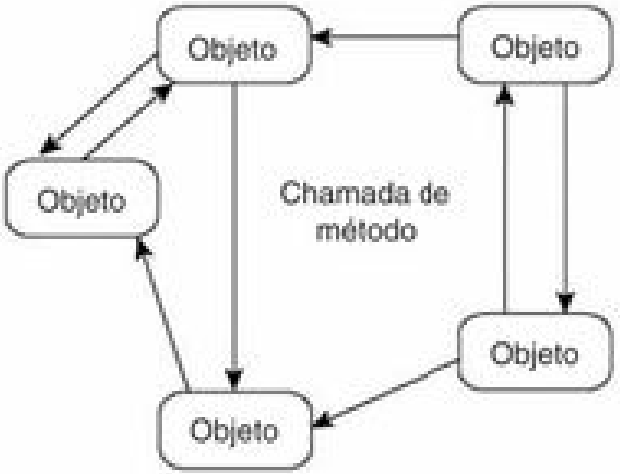
Estilos: Arquitetura em camadas

- Componentes da camada L_i utilizam os serviços de componentes da camada L_{i-1}
  - Ex.: protocolos do modelo TCP/IP



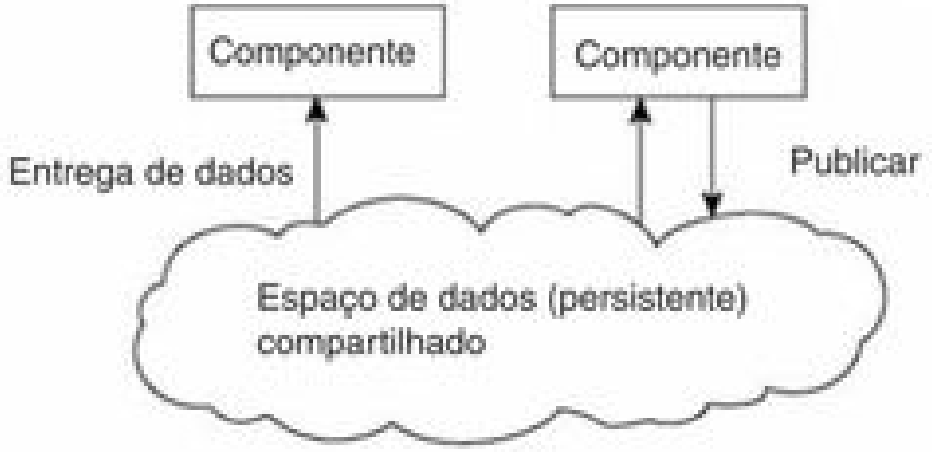
Estilos: Arq. baseada em objetos

- Componentes mapeados em objetos
- Comunicação através de chamadas de procedimento remotos
  - Ex.: Java RMI



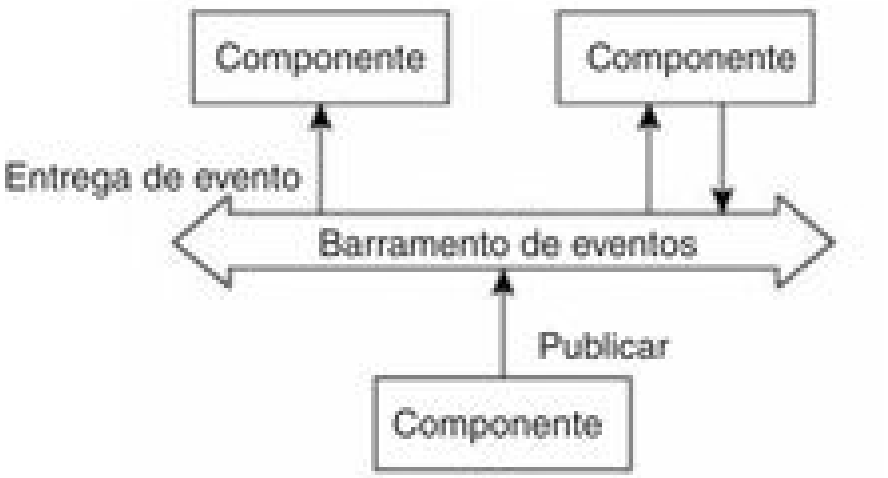
Estilos: Arq. centrada em dados

- Comunicação acontece por um repositório de dados comum
  - Ex.: arquivos ou base de dados



Estilos: Arq. baseada em eventos

- Componentes se comunicam através de eventos
  - Publicar/subscrever
  - Podem carregar também dados
- Componentes fracamente acoplados
  - Não precisam fazer referência direta uns aos outros



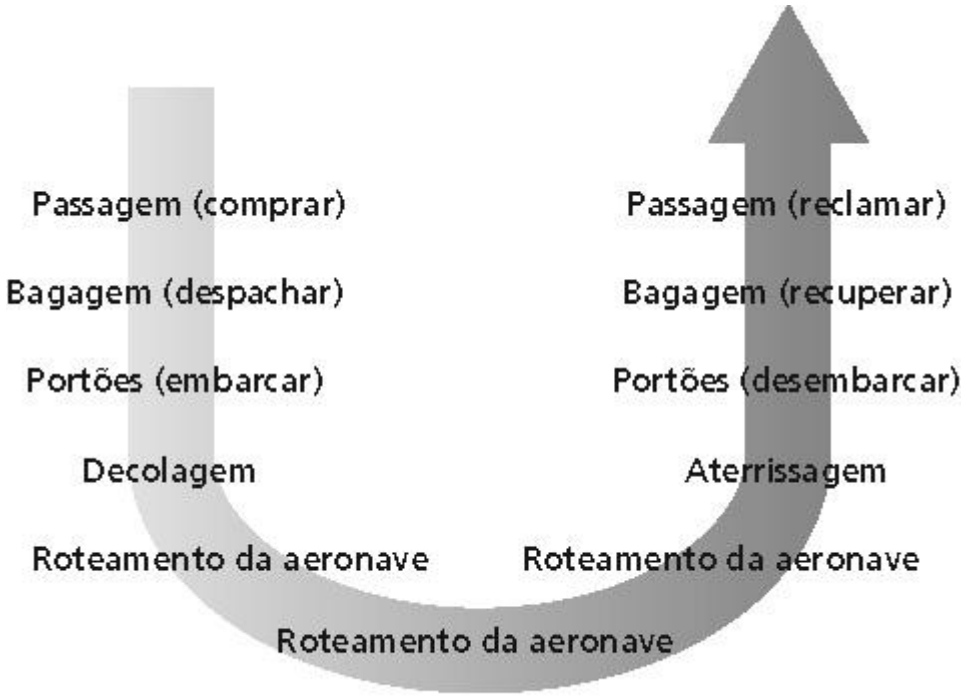
Pode ser combinado também com o estilo centrado em dados → **desacoplamento temporal**

Antes, um pouco sobre redes...

- Redes são complexas!
  - Hosts, roteadores, enlaces variados...
  - Aplicações, protocolos e serviços variados...
  - Hardware e software
- Como organizar isso tudo?!
- Modelo em camadas!

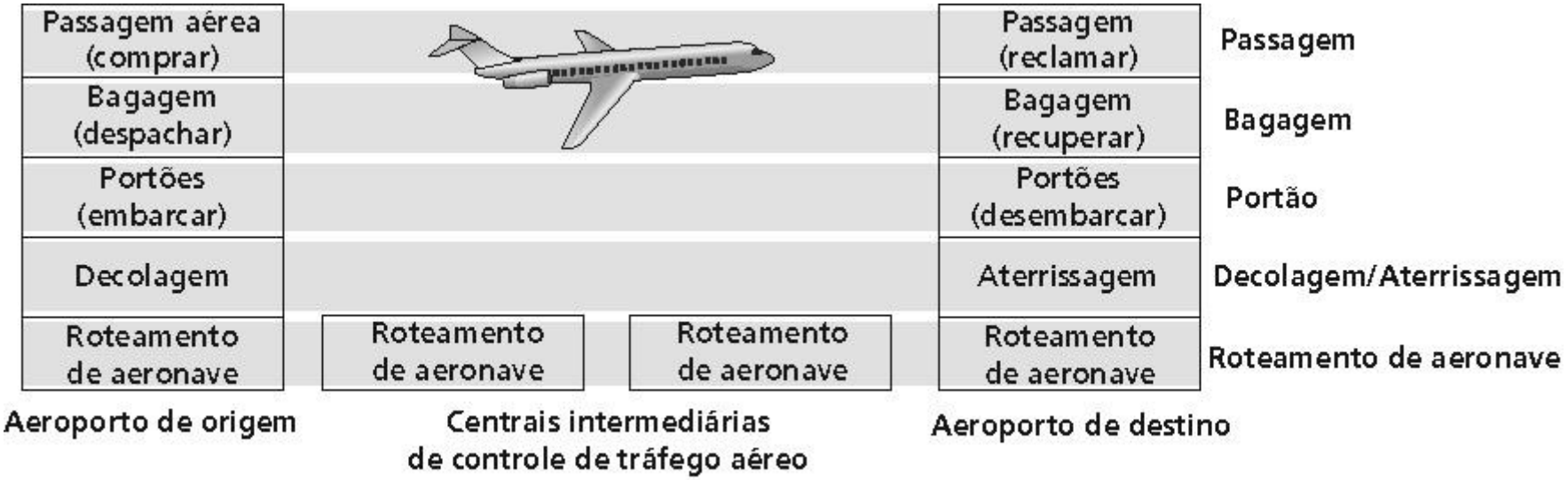
Exemplo: Viagem de avião

- Série de etapas



Exemplo: Viagem de avião

- **Camadas**: cada uma implementa um serviço
  - Usando suas ações internas
  - Usando serviços prestados pela camada inferior

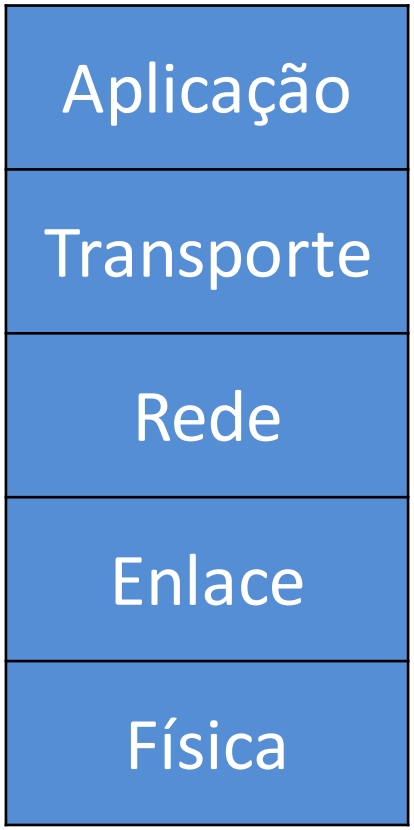


Modelo em camadas

- Grande valor na análise de sistemas complexos
  - Facilita a análise e o estudo das partes do sistema e suas inter-relações
- Subdivisão do “problema” em módulos menores
  - Facilita a implementação e atualização
  - Mudanças em uma camada são transparentes para o restante do sistema
    - Basta que implementem o mesmo serviço!
    - Ex.: Troca do serviço de portão

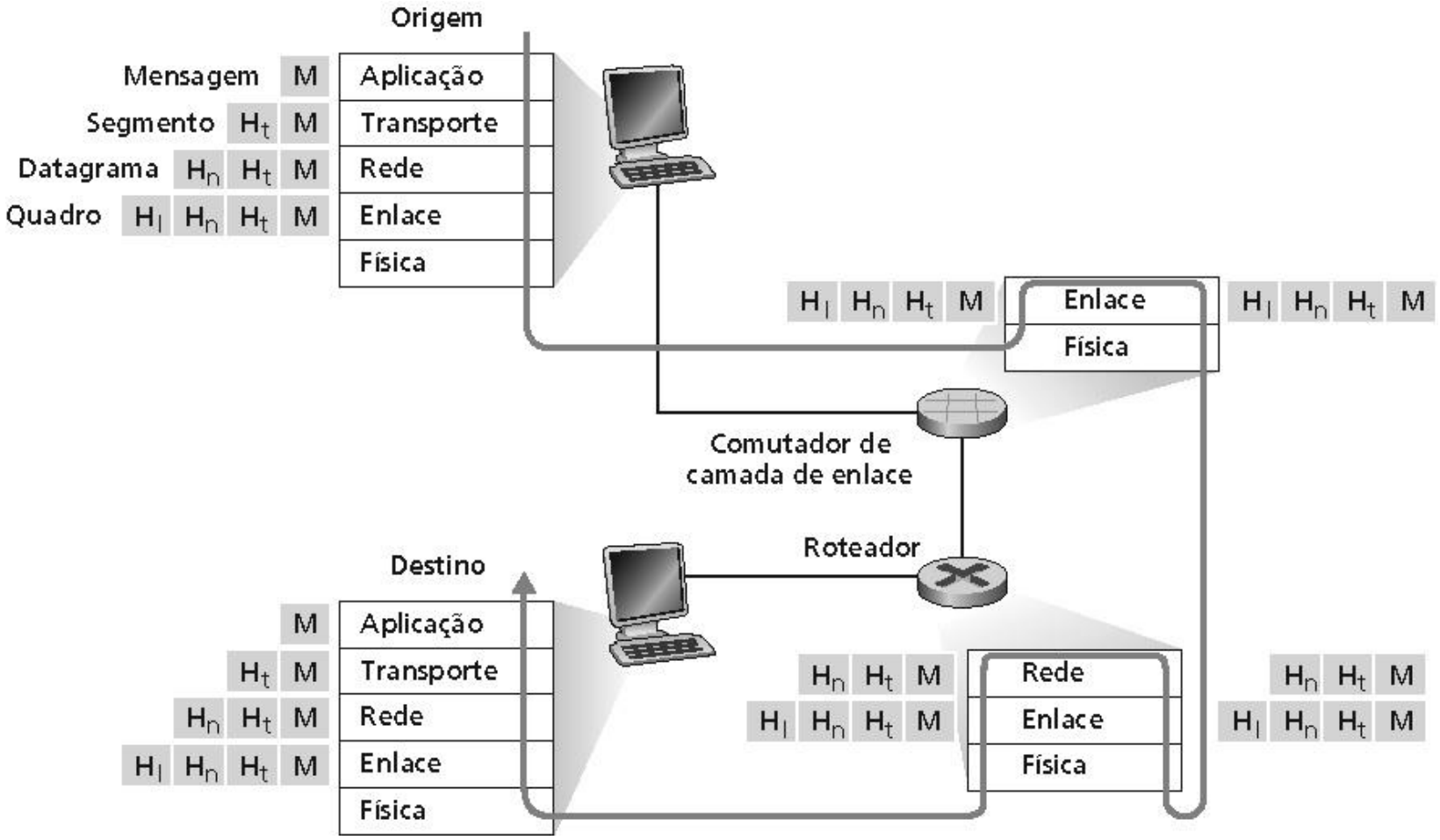
Modelo de camadas na Internet

- **Aplicação**: programas dos usuários
  - Web, email, torrent, ...
- **Transporte**: transferência de dados fim-a-fim
  - TCP e UDP
- **Rede**: roteamento de pacotes da origem ao destino
  - IP, protocolos de roteamento
- **Enlace**: transferência de dados entre elementos vizinhos
  - PPP, Ethernet, 802.11, ...
- **Física**: transferir bits pelo meio físico



Pilha de protocolos TCP/IP

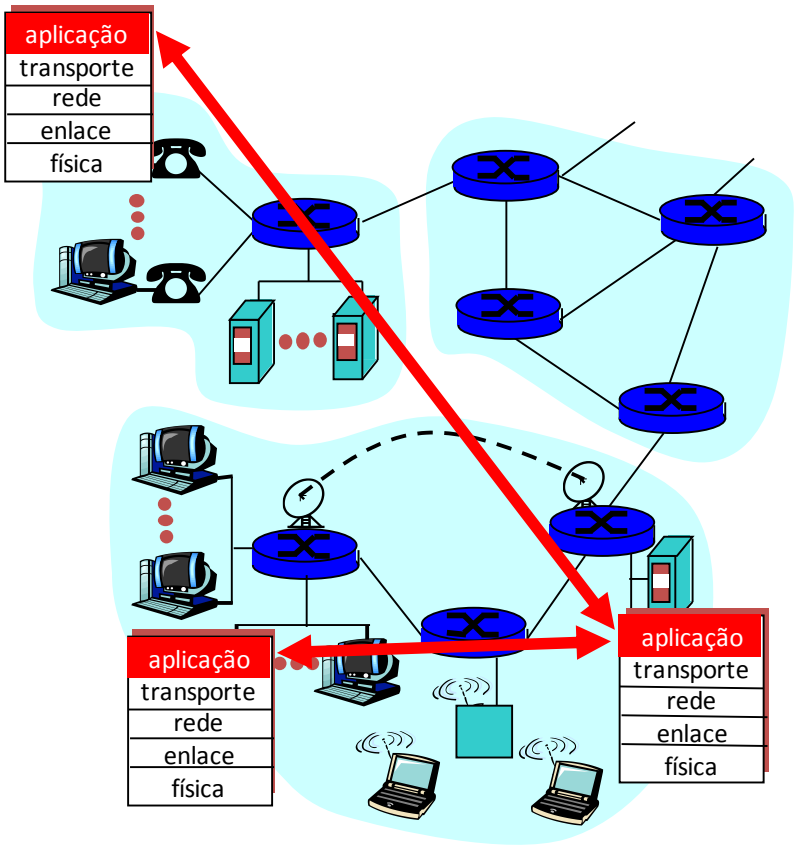
Encapsulamento



Como criar uma aplicação?

- Escrever um programa!
  - Será executado nos hosts
  - Comunicação via rede
  - Ex.: servidor Web

- Não é possível (nem necessário) desenvolver software para rodar no núcleo da rede
  - Não implementam camada de aplicação
  - Vantagem → facilita o desenvolvimento!



Comunicação entre processos

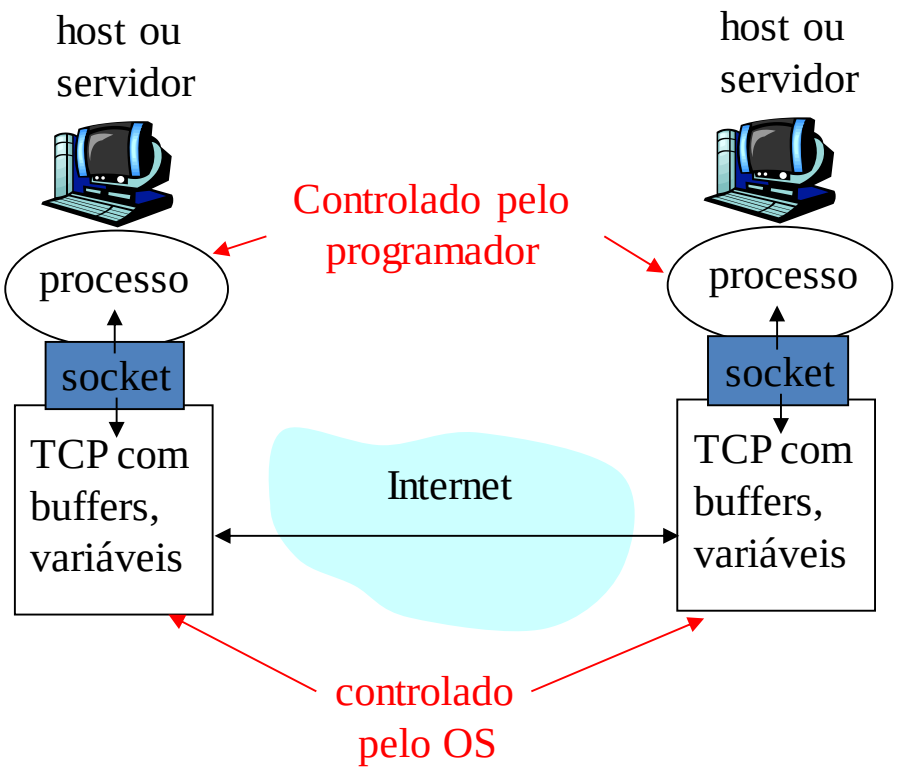
- **Processo** → programa em execução num sistema final
  - Processo de um mesmo sistema se comunicam por funcionalidades do SO
  - Processos em sistemas diferentes se comunicam através da troca de mensagens pela rede

- Tipos de processo
  - Processo cliente: é aquele que inicia a comunicação, que envia a primeira mensagem para outro processo
  - Processo servidor: fica aguardando novas mensagens de um processo cliente

- Em arquiteturas P2P, os peers utilizam ambos os tipos de processo

Sockets

- Interface usada pelos processos para o envio de mensagens
  - Interface entre as camadas de aplicação e transporte
  - Analogia da porta da casa
    - Processo “empurra” (envia) mensagens pela porta (socket)
    - Mensagens entram (são recebidas) pela porta
- Sockets possuem uma API
  - *Application Programming Interface*
  - Determina como o programador pode criar e configurar um socket
    - Tipos: TCP ou UDP
    - Outros parâmetros



Endereço dos processos

- Como identificar o destino das mensagens?
  - Endereço IP: identifica o outro sistema final
- Vários processos podem existir no mesmo host!
  - Número de porta: identifica qual processo no outro sistema final
  - Portas específicas são reservadas para aplicações populares
    - Servidor Web: 80 (http)
    - Servidor de envio de email: 25 (smtp)

Protocolos de camada de aplicação

- Define como os processos da camada de aplicação trocam mensagens entre si
  - Tipos de mensagens
  - Sintaxe (informações) das mensagens
  - Semântica (significado) das informações
  - Quando e como enviar as mensagens

- RFCs definem vários protocolos (ex.: HTTP [RFC 2616])
  - Facilita a interoperabilidade
  - Mas também existem protocolos proprietários! (ex.: Skype)

Requisitos de aplicações

- O que será feito com as mensagens depois que passam pela porta?
  - Diferentes tipos de serviço de transporte
  - Escolha do serviço adequado depende da aplicação
- Parâmetros importantes a serem avaliados na escolha
  - Transferência de dados confiável (perda de pacotes)
  - “Largura de banda” (vazão de dados)
  - Restrições de temporização (atraso)

Requisitos de aplicações mais comuns

Aplicação	Perdas	Vazão de dados	Sensível ao atraso
transf. de arquivos	sem perdas	elástica	não
e-mail	sem perdas	elástica	não
documentos Web	tolerante	elástica	não
áudio/vídeo em tempo real	tolerante	áudio: 5 kbps - 1 Mbps vídeo:10 kbps - 5 Mbps	sim, 100’s mseg
áudio/vídeo armaz.	tolerante	igual à anterior	sim, segundos
jogos interativos	tolerante	alguns kbps - 10 Mbps	sim, 100’s mseg
msg. instantênea	sem perda	elástica	sim e não

Serviços dos protocolos de transporte na Internet

- **Serviço TCP**
  - Orientado à conexão: apresentação inicial entre cliente e servidor
  - Transferência confiável
  - Controle de fluxo: não sobrecarrega o receptor lento
  - Controle de congestionamento: reduz a vazão quando a rede está sobrecarregada
  - Não fornece: garantia de vazão mínima e atraso

- **Serviço UDP**
  - Transferência não confiável
  - Não fornece: conexão, confiabilidade, controles de fluxo e congestionamento, garantia de vazão e atraso

- Vale a pena usar UDP?

Aplicações da Internet: protocolos de aplicação e transporte

Aplicação	Protocolo	Transporte
transf. de arquivos	FTP/HTTP	TCP
e-mail	SMTP/IMAP/POP3	TCP
documentos Web	HTTP	TCP
áudio/vídeo em tempo real	RTP (aberto) Skype (proprietário)	UDP
áudio/vídeo armaz.	Youtube* (HTTP)	UDP ou TCP
msg. instantênea	XMPP (aberto) Skype (proprietário)	TCP

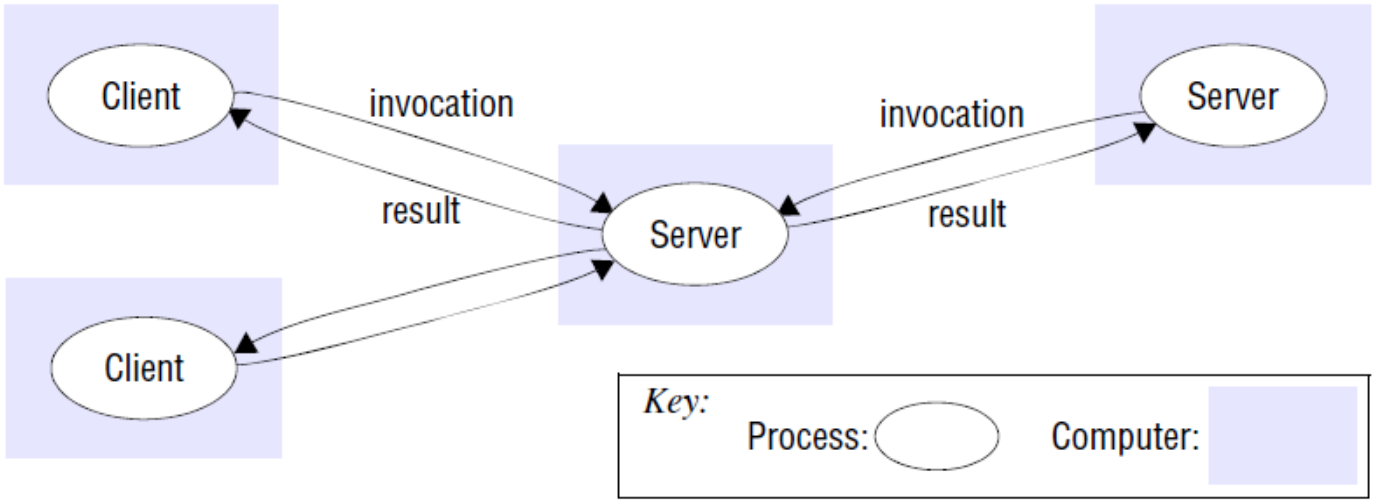
* Depende de outra aplicação para ser executada (Web)

Tipos de arquiteturas

- **Arquiteturas centralizadas**
  - Cliente-servidor
  - Ex.: vídeo sob demanda, terminais bancários
- **Arquiteturas descentralizadas**
  - Peer-to-peer
  - Ex.: Chord
- **Arquiteturas híbridas**
  - Peer-to-peer
  - Ex.: BitTorrent

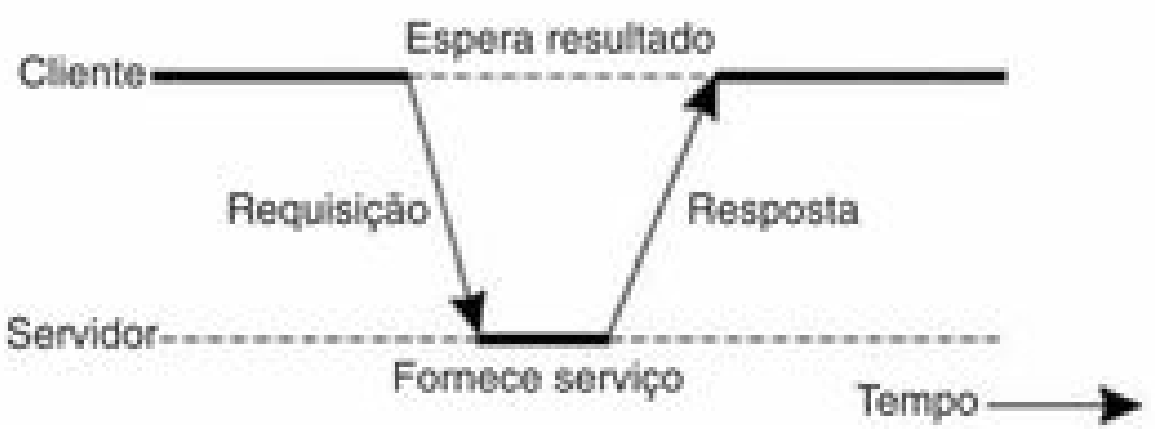
Arquiteturas centralizadas

- Dois tipos de componentes (com possível sobreposição)
  - **Servidor** → implementa e fornece um serviço
  - **Cliente** → requisita um serviço ao servidor
    - Comportamento **requisição-resposta**



Arquiteturas centralizadas

- Requisição-resposta:



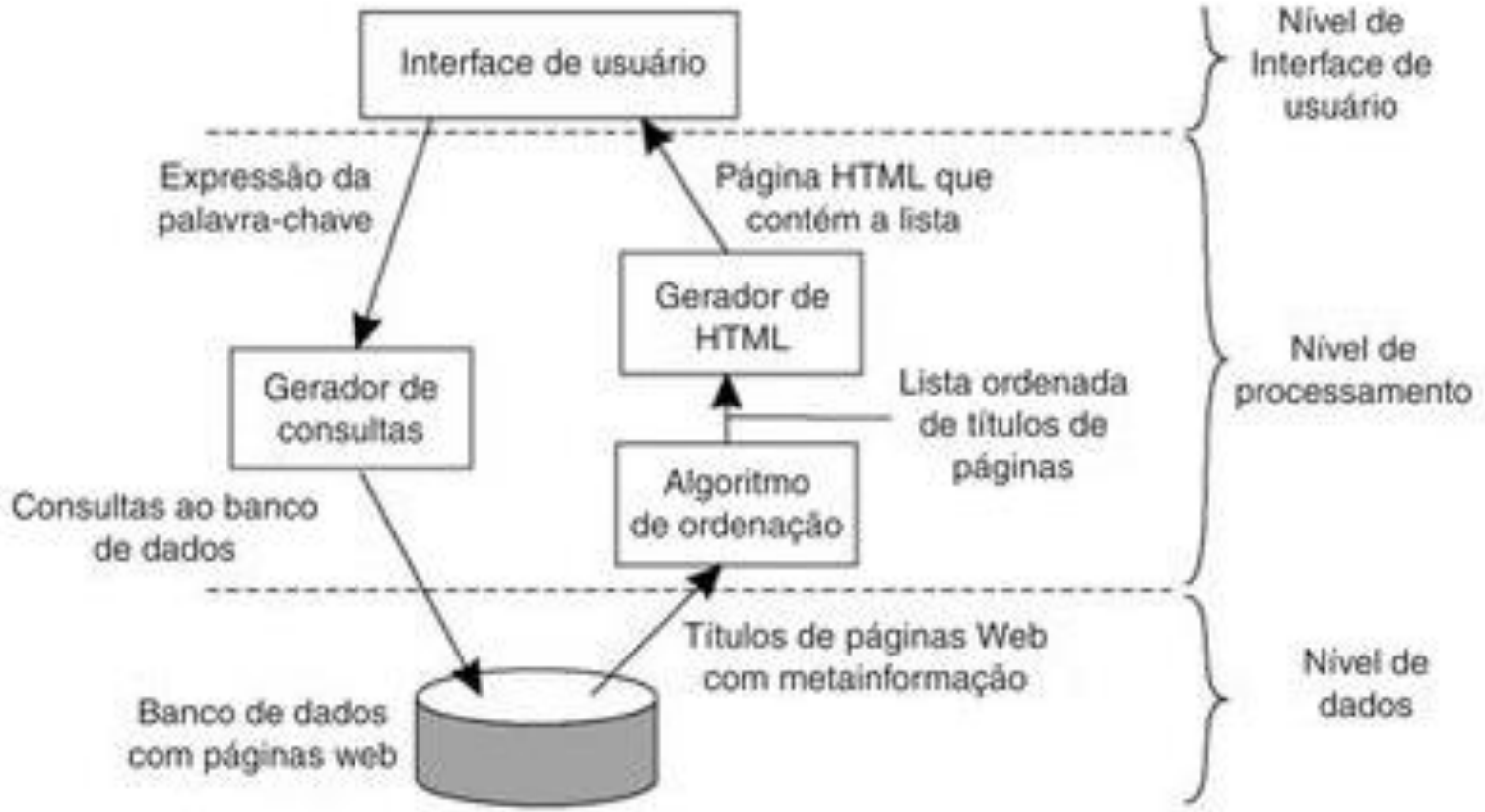
Arquiteturas centralizadas

- Muitas aplicações cliente-servidor visam dar suporte ao acesso de usuários a banco de dados
  - Ex.: Web

- Partes de uma aplicação cliente-servidor
  - Nível de interface do usuário
  - Nível de processamento
  - Nível de dados

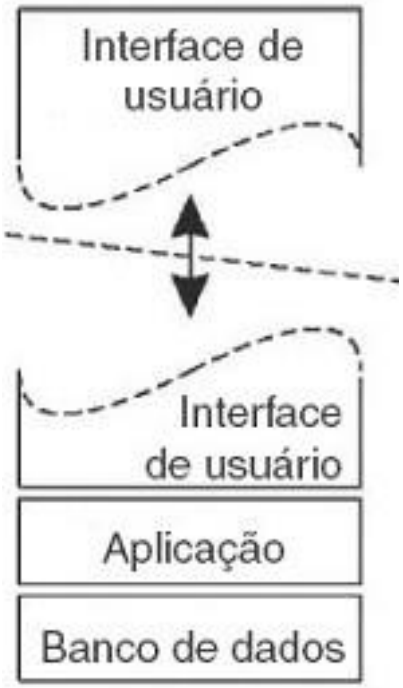
Arquiteturas centralizadas

- Exemplo: busca na Web



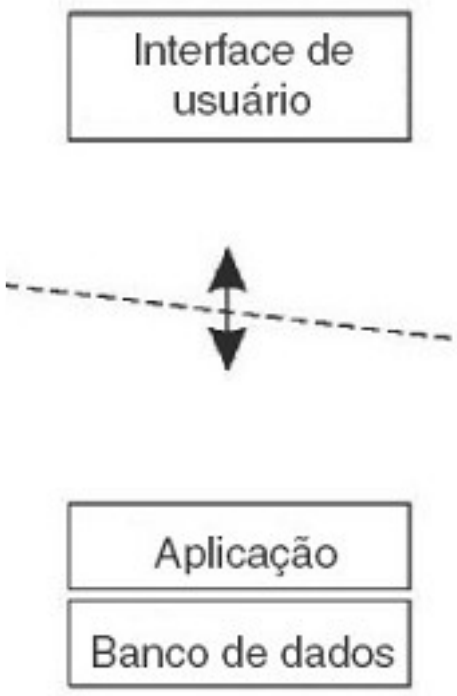
Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Duas divisões físicas (Caso 1)**
    - Cliente apenas apresenta dos dados
    - Aplicações controlam remotamente a apresentação dos dados
    - Ex.: terminal gráfico remoto (VNC)



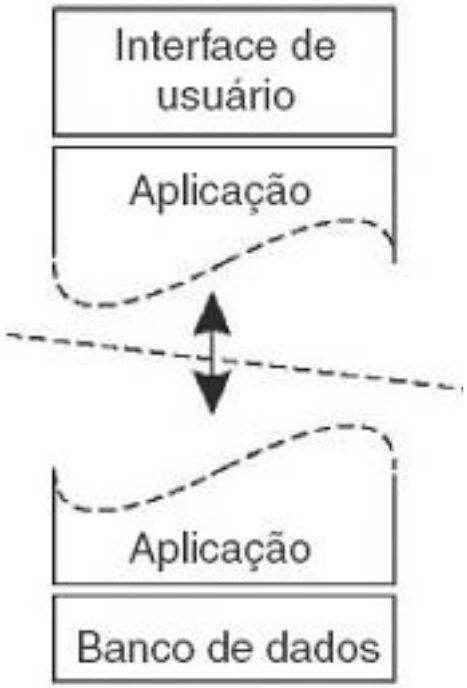
Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Duas divisões físicas (Caso 2)**
    - Cliente faz apenas o processamento necessário para apresentar a interface da aplicação
    - Lado servidor independe do cliente



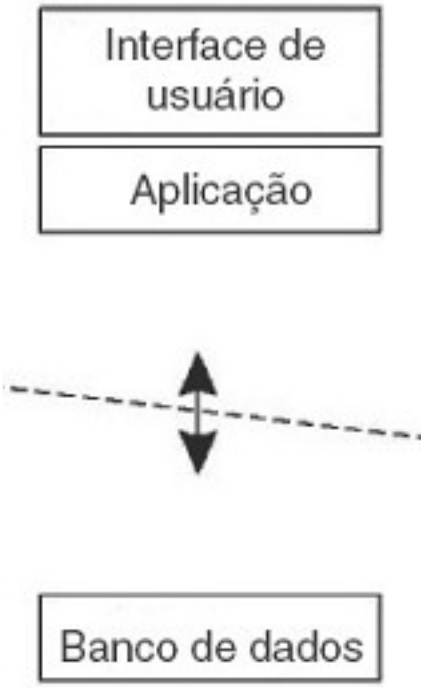
Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Duas divisões físicas (Caso 3)**
    - Parte do processamento fica do lado cliente
    - Ex.: formulário Web



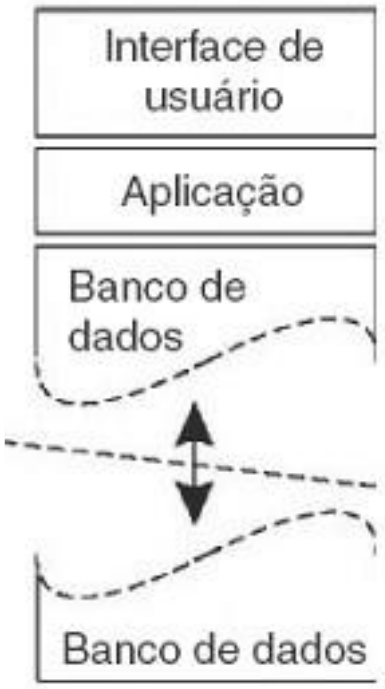
Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Duas divisões físicas (Caso 4)**
    - Processamento do lado cliente
    - Apenas acesso remoto aos dados



Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Duas divisões físicas (Caso 5)**
    - Parte dos dados armazenada no lado cliente
    - Ex.: cache de um navegador Web

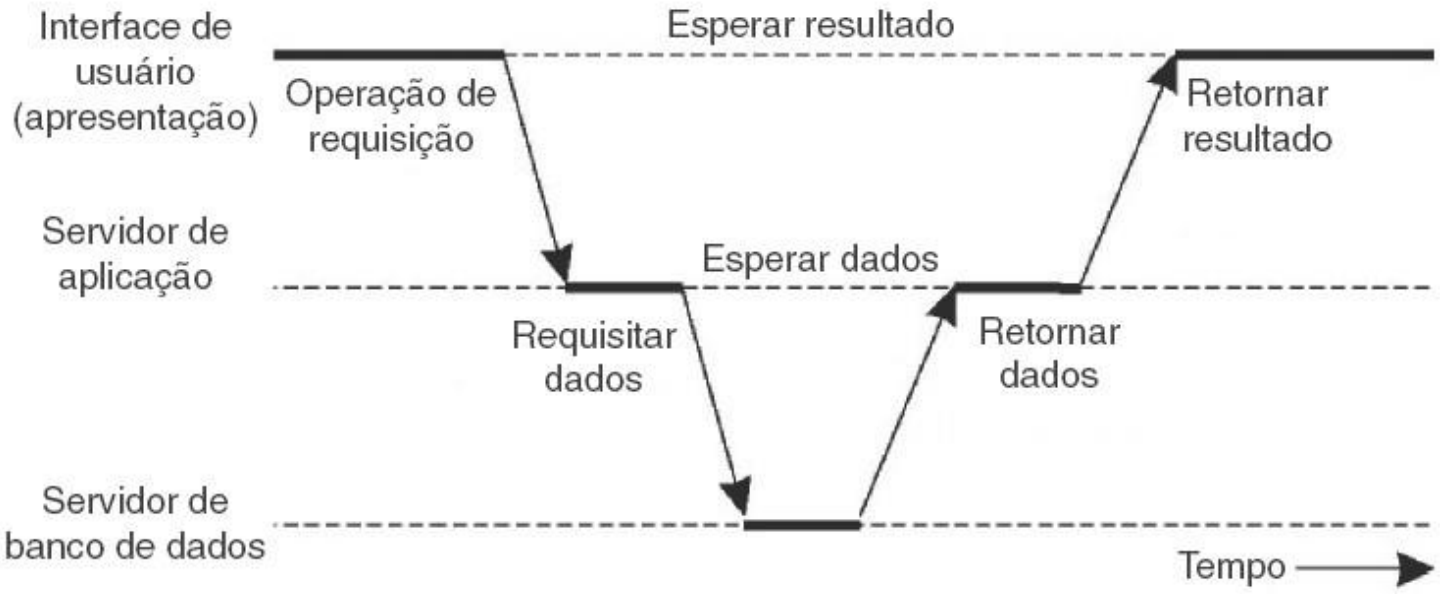


Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - Tendência é adotar modelos dos casos iniciais (casos 1 a 3)
    - Clientes “magros” (***thin clients***) → pouco, ou nada, de processamento nos clientes
  - Clientes “gordos” (***fat clients***) são mais propensos a erros e difíceis de gerenciar
    - Grande parte do processamento do lado cliente

Arquiteturas centralizadas

- Como distribuir fisicamente os três níveis lógicos?
  - **Três divisões físicas** →servidor também pode operar como cliente



Arquiteturas descentralizadas

- **Distribuição vertical** → divisão do sistema em componentes com funções diferentes que são colocados em máquinas diferentes
  - Ex.: Cliente-servidor multinível

- **Distribuição horizontal** → divisão do sistema em componentes logicamente equivalentes que operam sobre diferentes porções do conjunto completo de dados
  - Ex.: Peer-to-peer

Arquiteturas descentralizadas

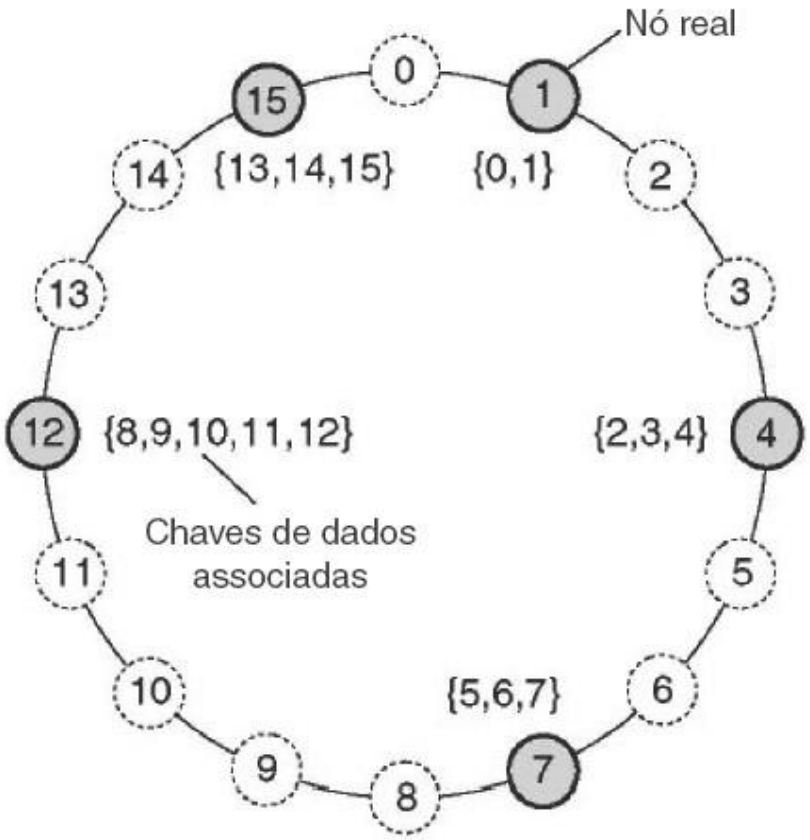
- **Sistema peer-to-peer**
  - Todos os processos são iguais
  - Interação entre processos é simétrica
    - São ao mesmo tempo clientes e servidores
  - **Rede de sobreposição** (*overlay*)
    - Nós = processos
    - Enlaces = canais de comunicação entre processos (geralmente, conexões TCP)
  - Comunicação entre processos utiliza os enlaces da rede de sobreposição

Arquiteturas descentralizadas

- Tipos de rede de sobreposição:
  - **Estruturadas** → procedimento determinístico para definição do overlay
    - Ex.: tabela hash distribuída (DHT)
  - **Não-estruturadas** → algoritmos aleatórios para a construção da rede overlay, gerando um grafo aleatório

Arquiteturas descentralizadas

- **Sistema Chord (rede overlay estruturada)**



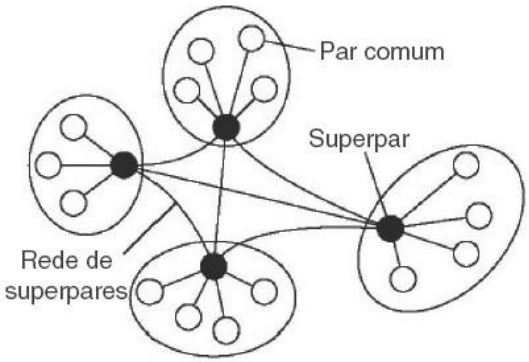
- *Stoica et al, 2003*
- Organização lógica na forma de um anel
- Dado com a chave *k* é mapeado para o nó com *id*>=*k*
- Função de **LOOKUP(*k*)**
  - Retorna **succ(*k*)**
- Entrada e saída de nós da rede é uma tarefa determinística

Arquiteturas descentralizadas

- **Rede overlay não-estruturada**
  - Cada par possui uma lista dos vizinhos (**visão parcial**)
    - Mantida através da troca de visões parciais entre vizinhos
  - Mecanismos para atualizar vizinhança periodicamente
  - Entrada e saída de nós é trivial

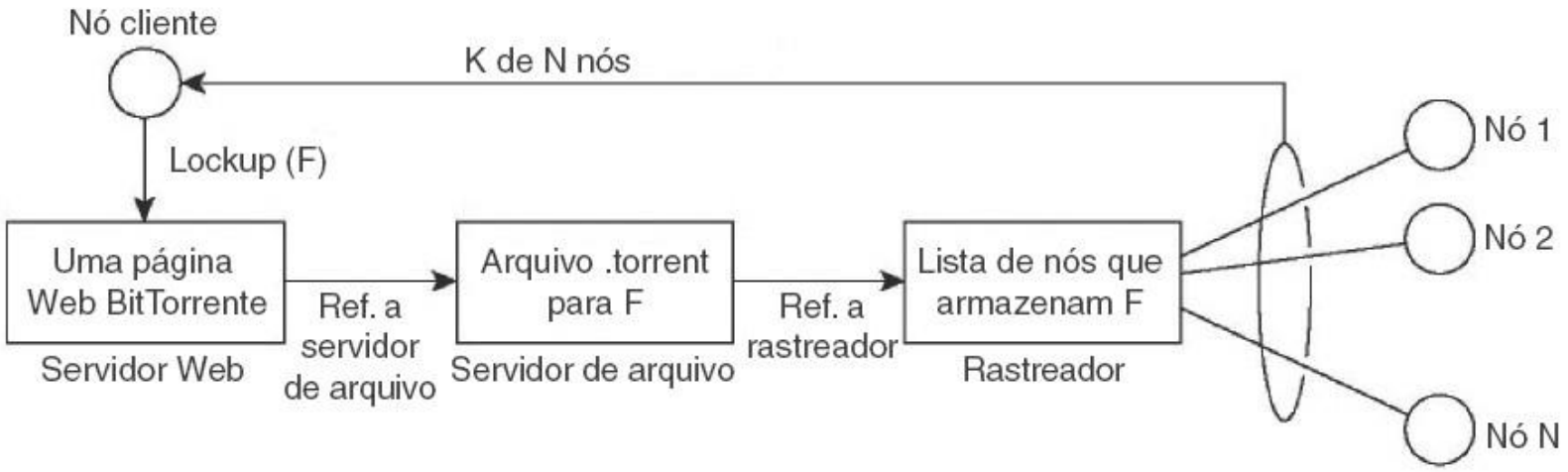
Arquiteturas descentralizadas

- **Rede overlay não-estruturada**
  - Como encontrar dados?
    - **Inundação na rede overlay**
      - Envio de mensagem de busca para todos os outros nós
      - Diversas técnicas para o controle de inundação
    - **Superpares** (*superpeers*)
      - Mantém índice para a localização dos dados
      - Mensagem de busca enviada apenas para o superpar
      - Organização hierárquica → nós comuns conectados à superpares
      - Problemas:
        - » Como eleger o superpar?
        - » Como garantir sua disponibilidade?



Arquiteturas híbridas

- **BitTorrent** (Cohenm, 2003)



Arquiteturas vs. *middleware*

- Middlewares possuem estilos arquitetônicos específicos
  - Vantagem → Facilitam o projeto de aplicações
  - Problema → Podem não atender todas as necessidades do projeto
- Ideal seria um *middleware* genérico...
  - Mas dar suporte à diversos estilos arquitetônicos faz com que o middleware fique “inchado”
- Compromisso entre facilidade de projeto e flexibilidade de uso do middleware

Arquiteturas vs. *middleware*

- **Interceptadores**
  - Permite tornar middleware genérico sem muito “inchaço”