Programação Distribuída

Aula 2

Índice

- Estilos Arquitetônicos
- Arquitetura de Sistemas
- Arquiteturas versus Middleware
- Autogerenciamento em Sistemas Distribuídos

Estilos Arquitetônicos

- Astrolabe
- Globule
- JADE

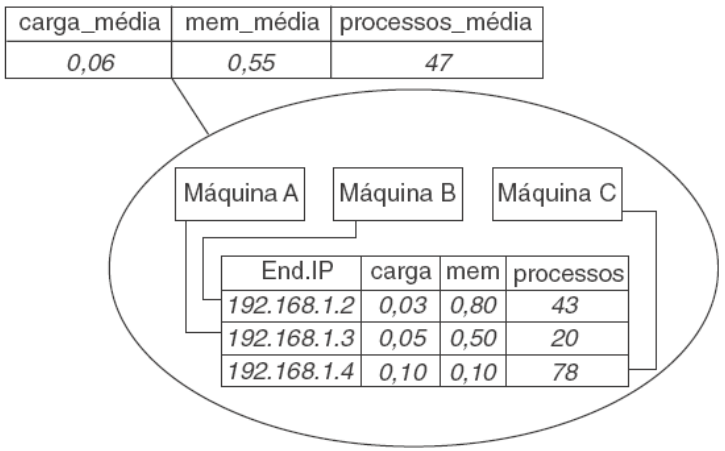


Figura 2.17 Coleta de dados e agregação de informações em Astrolabe.

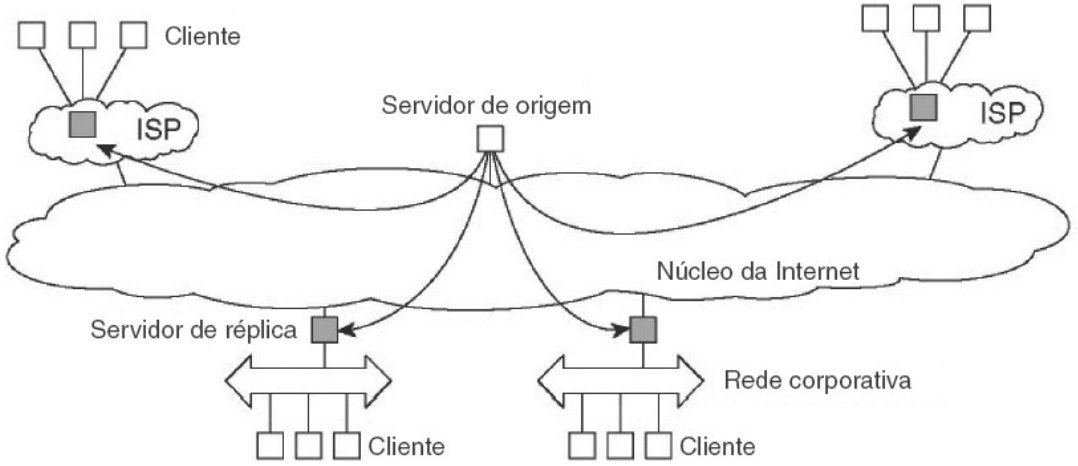


Figura 2.18 Modelo de servidor de borda adotado pela Globule.

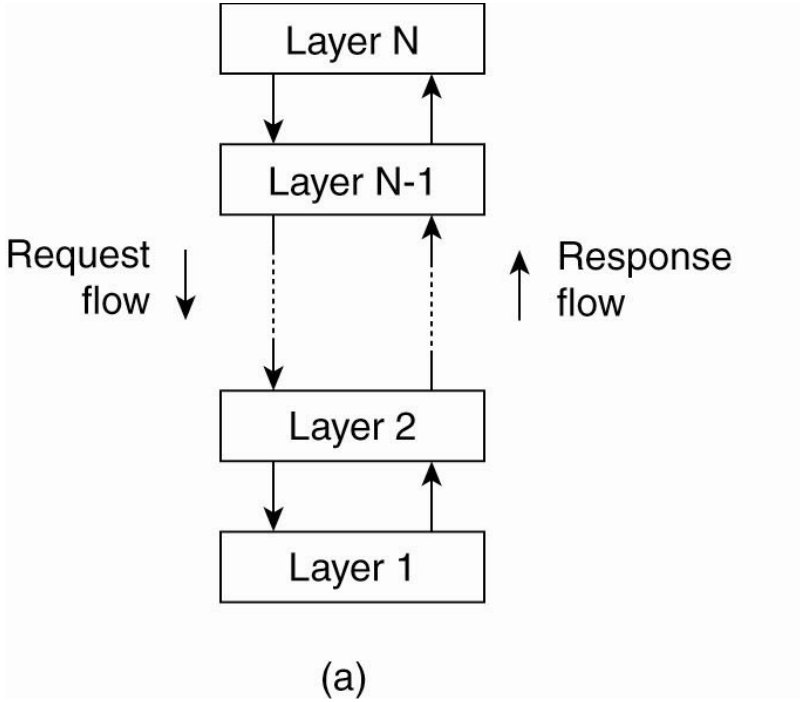
Estilos Arquiteturais

- Estilos arquitetônicos é formulado em termos de
  - Componentes: Unidade modular com interfaces requeridas e fornecidas bem definidas que é substituível dentro de seu ambiente
  - Conectores: Mecanismo que serve de mediador da comunicação ou da cooperação entre componentes

- Os principais estilos arquitetônicos são:
  - **Em camada**
  - **Baseadas em Objeto**
  - **Centradas em Dados**
  - **Baseadas em Eventos**

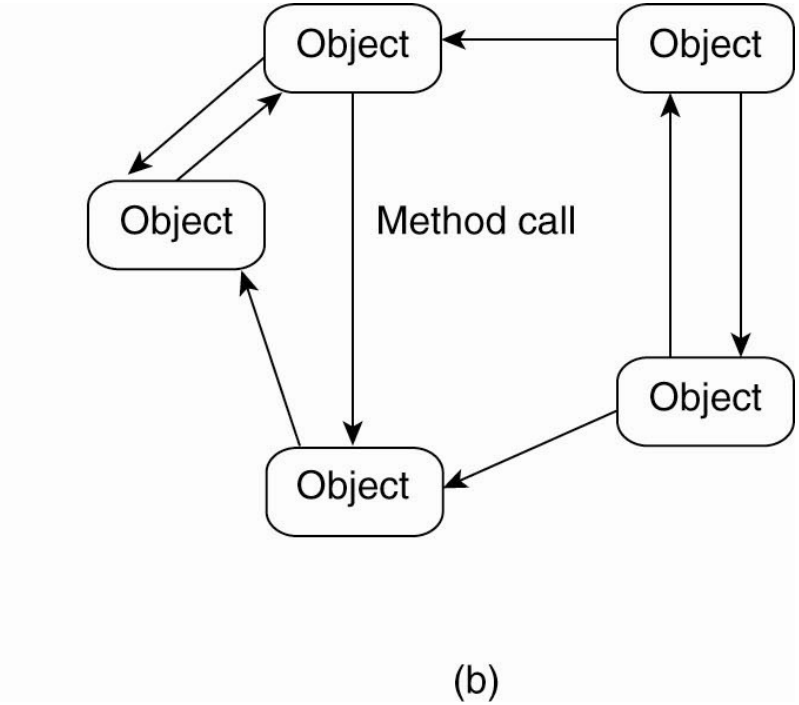
Estilos Arquiteturais

- **Em Camadas**
  - Componentes são organizados em **camadas**
  - Componente da camada N tem permissão de chamar componentes na camada N-1
  - Comum em redes de computadores



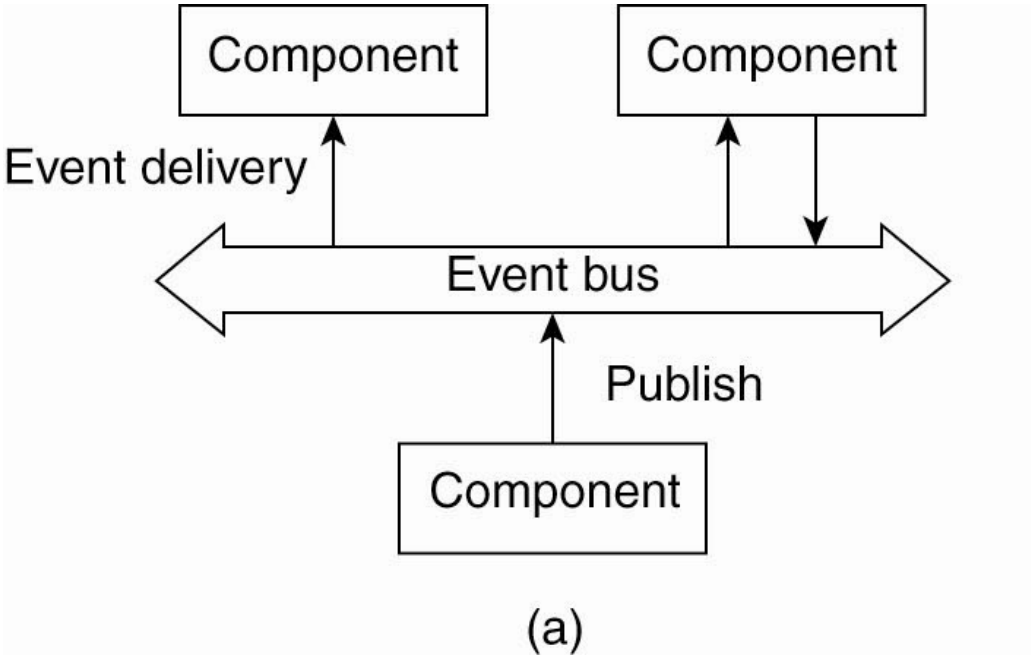
Estilos Arquiteturais

- **Baseadas em Objeto**
  - Objeto → Componente
  - Objetos são conectados por meio de uma chamada de procedimento (remota).
  - Amplamente utilizada para sistemas de software de grande porte.



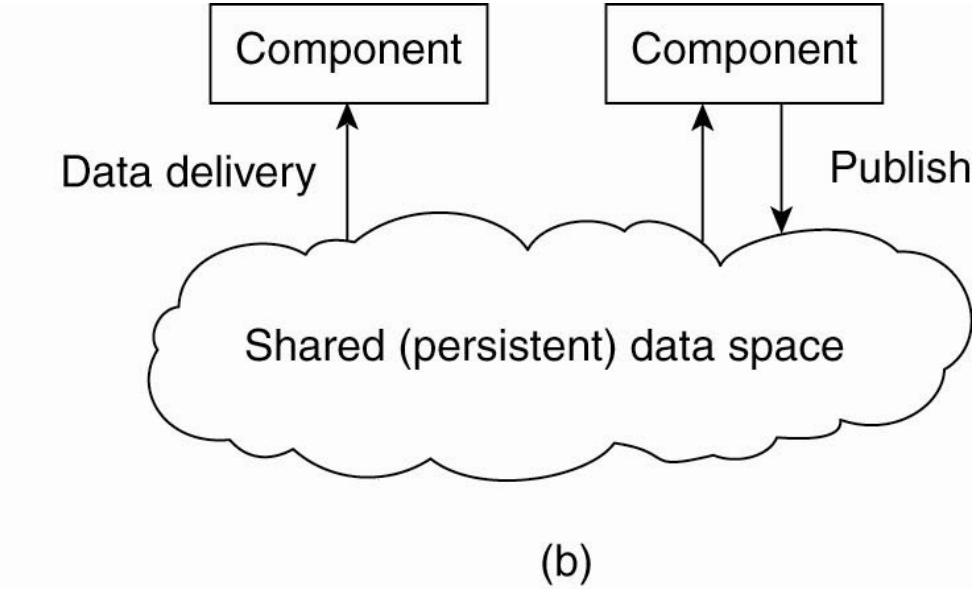
Estilos Arquiteturais

- **Centradas em Dados**
  - Processos se comunicam por meio de um repositório comum.
  - Sistemas distribuídos baseados na Web, em grande parte, são centrados em dados.



Estilos Arquiteturais

- **Baseadas em Eventos**
  - Sistemas publicar/subscrever
  - Processos publicam eventos e o **middleware** assegura que somente os processos que se subscreveram para esses eventos os receberão
  - Processos fracamente acoplados: processos não se referem explicitamente uns aos outros



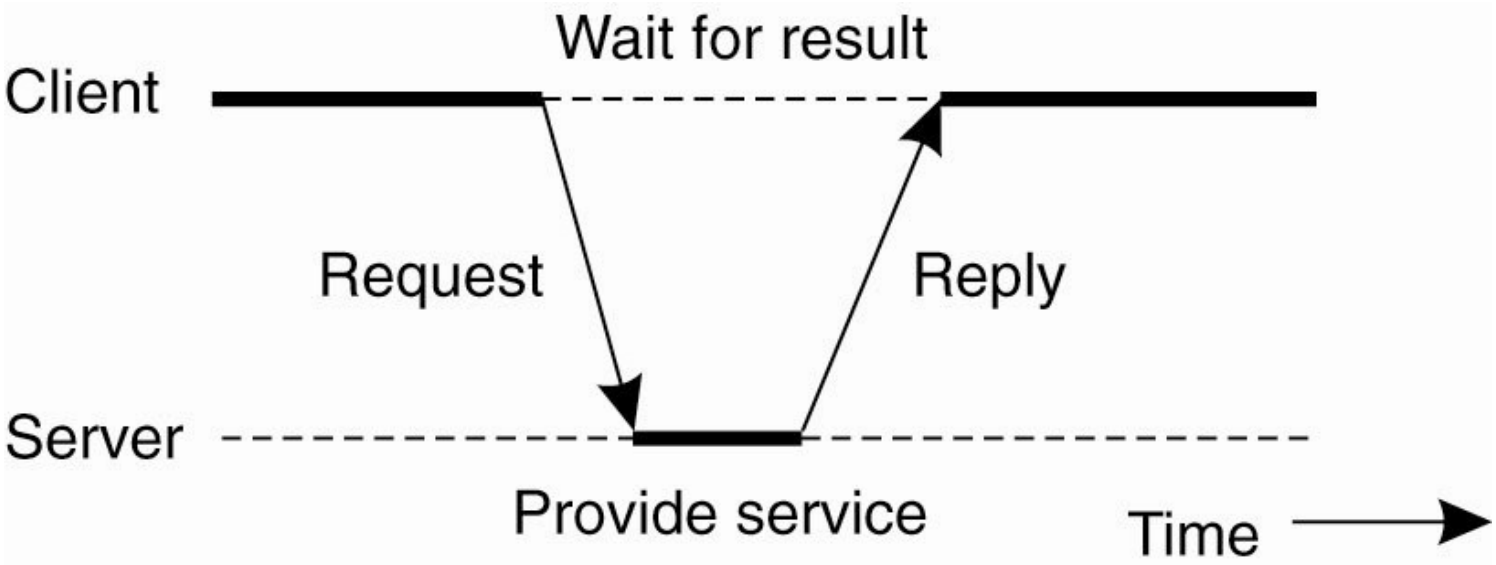
Arquitetura de Sistema

- Decisões a respeito de componentes de software, sua interação e sua colocação em máquinas reais.

- Três tipos:
  - Centralizadas
  - Descentralizadas
  - Hibridas

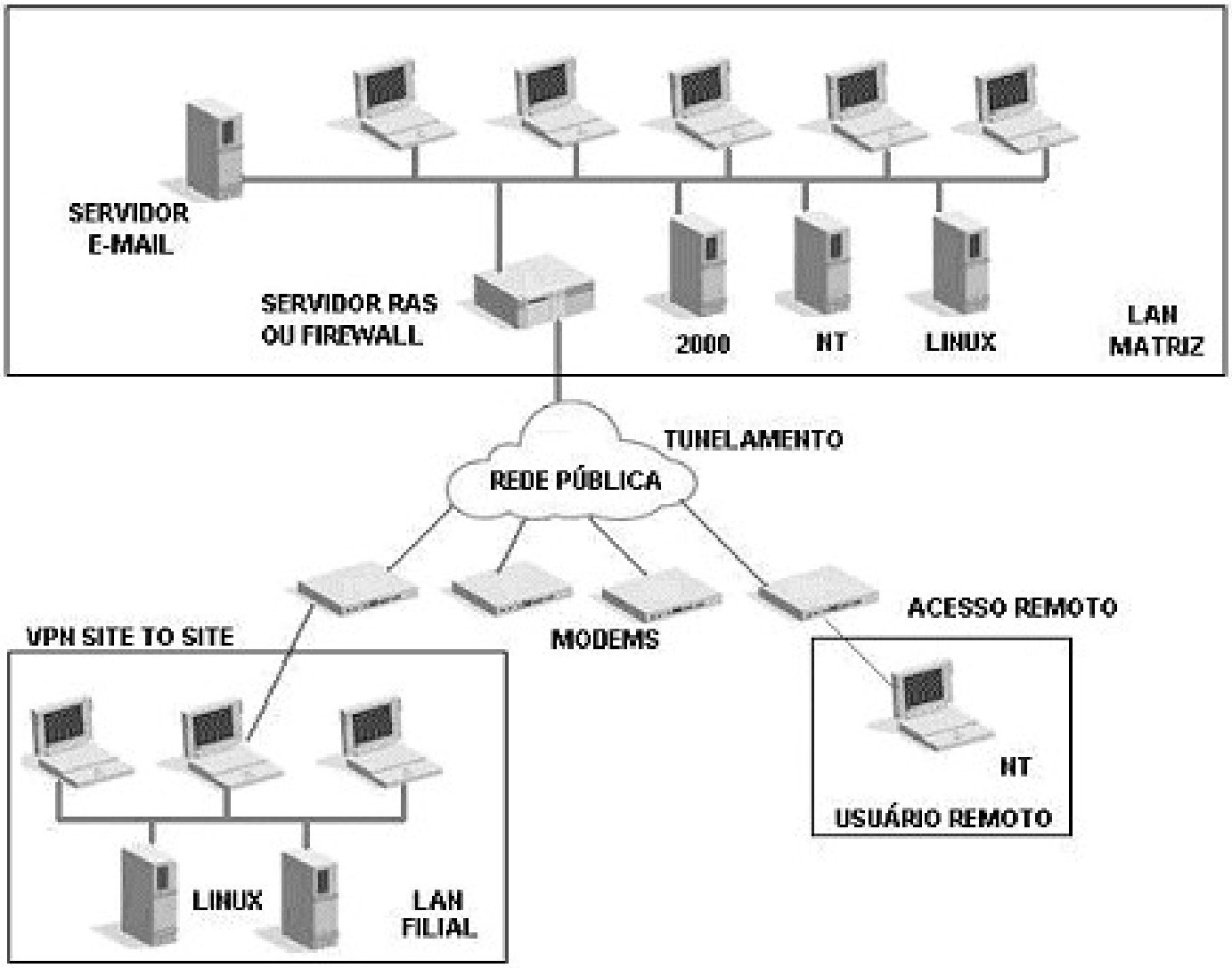
Arquitetura de Sistema

- **Centralizadas**
  - Modelo **cliente-servidor**
  - Comportamento de requisição-resposta



Arquitetura de Sistema

- **Centralizadas**



Arquitetura de Sistema

- **Centralizadas**
  - Como estabelecer a comunicação?
    1. **Protocolo sem conexão:**
       - Protocolo simples, que funciona bem em redes locais
       - Cliente empacota uma mensagem para o servidor diretamente
       - Eficiente se NÃO ocorrem problemas
       - Exemplo: Falhas → Transferências bancarias
       - Operações podem ser repetidas sem causar danos: idempotentes (múltiplos acessos ou pedidos a um recurso devem ter o mesmo efeito que um acesso somente)

- **Centralizadas**
  - Como estabelecer a comunicação?

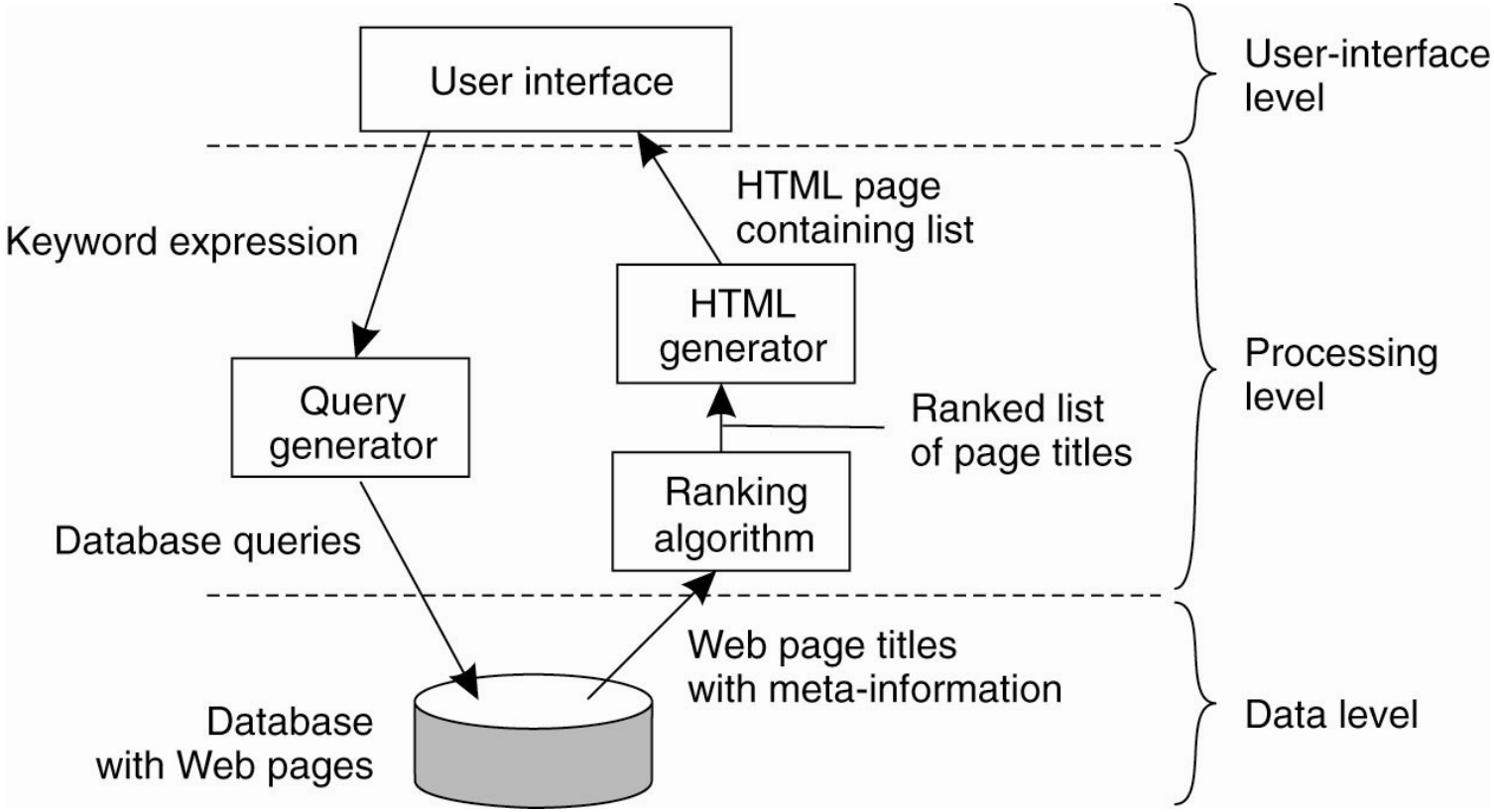
    2. Protocolo orientado a conexão
    - Solução funciona bem em sistemas de longa distância.
    - Sempre que um cliente requisita um serviço, primeiro se estabelece conexão com o servidor e depois se envia a requisição.

Arquitetura de Sistema

- **Centralizadas**
  - **Camadas de Aplicação**
    - » Como distinguir entre cliente e servidor?
      - Exemplo: Servidor de banco de dados distribuído → repassa requisições a servidores de arquivos. Assim, age como cliente continuamente.
    - » Como muitas aplicações cliente-servidor visam dar suporte ao acesso de usuários a banco de dados é conveniente que sejam divididas em três níveis distintos:
      - **Nível de interface de usuário**
      - **Nível de processamento**
      - **Nível de dados**

Arquitetura de Sistema

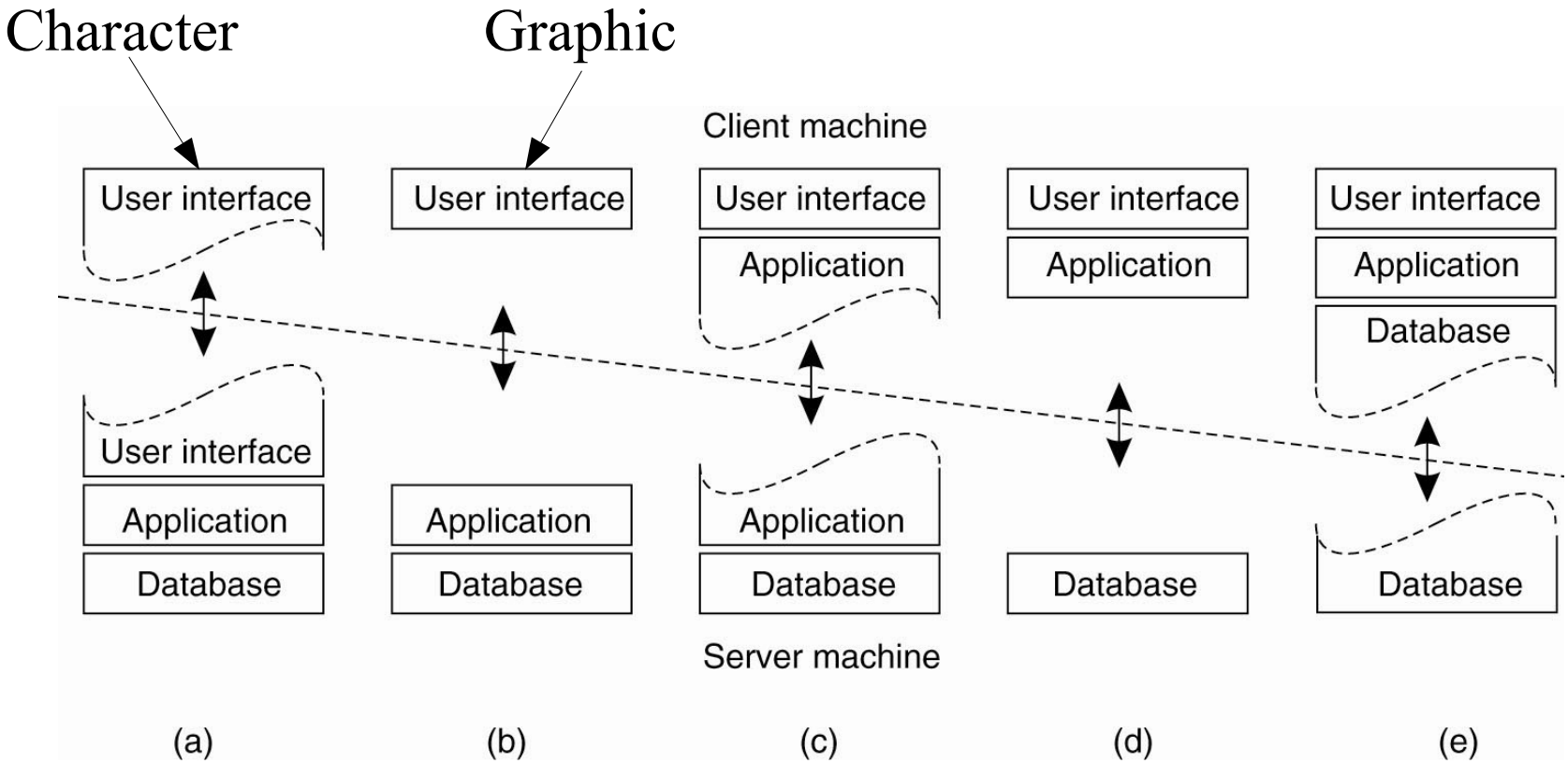
- **Centralizadas**
  - Exemplo:



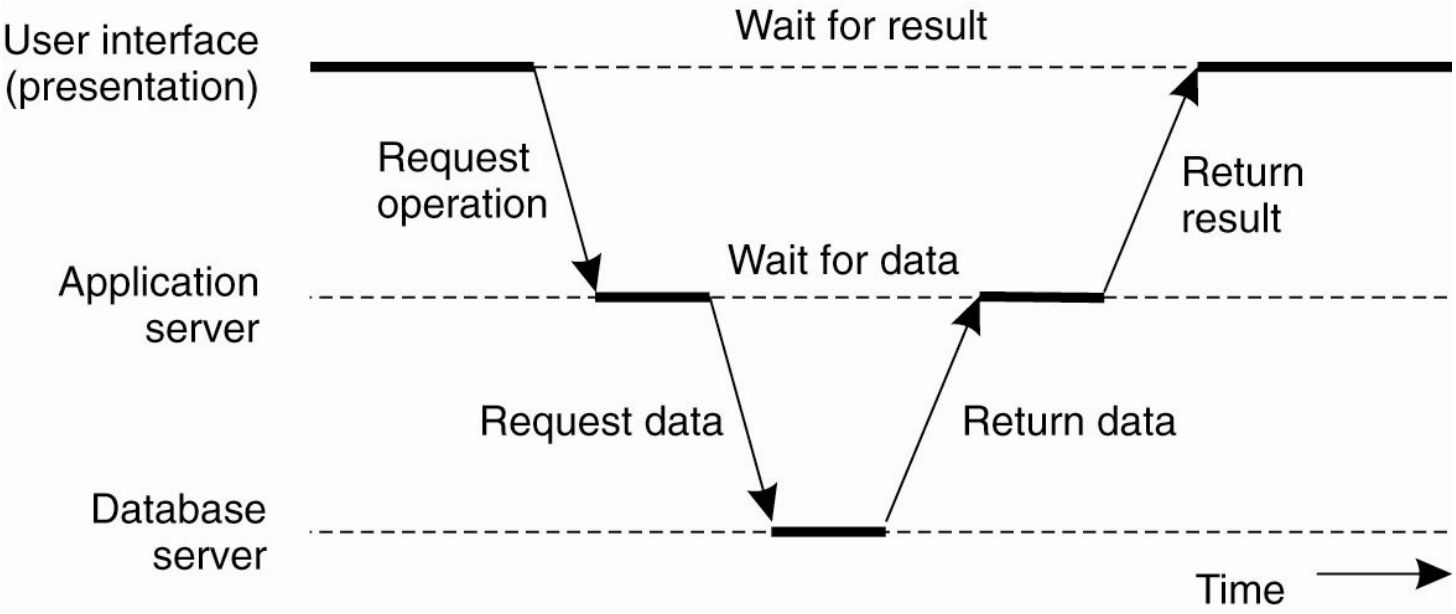
Arquitetura de Sistema

- **Centralizadas**
  - **Nível de interface de usuário.**
    - » Consiste em programas que permitam aos usuários finais interagir com aplicações.
    - » Diversos níveis de complexidade.
  - **Nível de processamento**
    - » Normalmente contem as aplicações
    - » Exemplo: Análise de dados financeiros que pode exigir métodos e técnicas sofisticados de estatística
  - **Nível de dados**
    - » Na sua forma mais simples, consiste em um sistema de arquivos.
    - » Mais comum utilizar um banco de dados.
    - » Normalmente implementado no lado servidor.
    - » Mantém os dados consistentes.
    - » Dados costumam ser persistentes.

- **Arquiteturas Multidivididas**
- Três níveis lógicos → várias possibilidades para a distribuição física de uma aplicação cliente-servidor por várias maquinas



- **Arquiteturas Multidivididas**
  - Gerenciamento de sistema:
    - » Clientes gordos (*fat clients*)
    - » Clientes magros (*thin clients*)
  - Servidor pode também agir como clientes: **arquitetura de três divisões**



- **Arquiteturas Descentralizadas**
  - Cliente-servidor possuem duas distribuições:
    - » **Distribuição vertical:**
      - Componentes logicamente diferentes em máquinas diferentes
      - Cada máquina é projetada para um grupo específico de funções
    - » **Distribuição horizontal:**
      - Cliente ou servidor pode ser fisicamente subdividido em partes logicamente equivalentes
      - Porção própria de dados (**peer-to-peer, processos iguais**)

Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer**
  - Processos são todos iguais.
  - Grande parte da interação entre processos é simétrica.
    - Cada processo age como cliente e servidor ao mesmo tempo (**servente**).
  - Formada por um conjunto de nós, organizados em um *overlay ou rede de sobreposição.*
    - » ***Overlay: rede na qual os nós são os processos e os enlaces representam os canais de comunicação possíveis.***
  - Comunicação não pode ser feita diretamente.
  - Arquiteturas estruturadas ou não-estruturadas.

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturada**

  - Rede de sobreposição e construída com a utilização de um procedimento determinístico.
  - Tabela de hash distribuida (**Distributed Hash Table - DHT**).
  - Ponto crucial: implementar um esquema eficiente e determinístico que mapeie a chave de um dado para o identificador de um nó.

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturada**

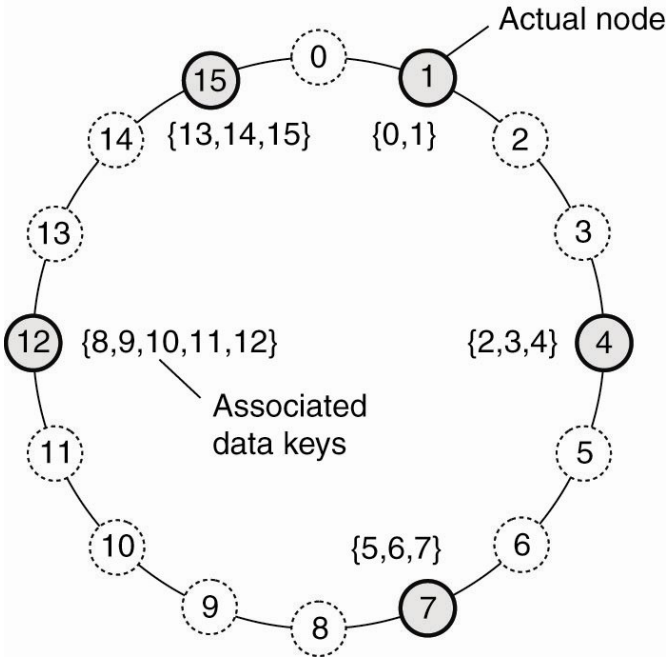
  - Ao consultar um determinado item de dado, o endereço de rede do nó com o conteúdo é retornado.
  - Requisição é roteada entre os nós até que o nó com o dado requisitado seja alcançado.
  - Dados e nós recebem uma chave aleatória.

Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturadas Gerais**
  - Quando o nó entra na rede, este recebe um espaço do conjunto dos índices dos arquivos, ao sair da rede a mesma deverá designar estes índices para outro nó. As buscas **não são** difundidas na rede sem direção como no **flooding**, ao invés disto são direcionadas para o nó correto.
  - Os protocolos que implementam este tipo de arquitetura são bem mais complexos, existem poucos estudos sobre utilização dos mesmos.

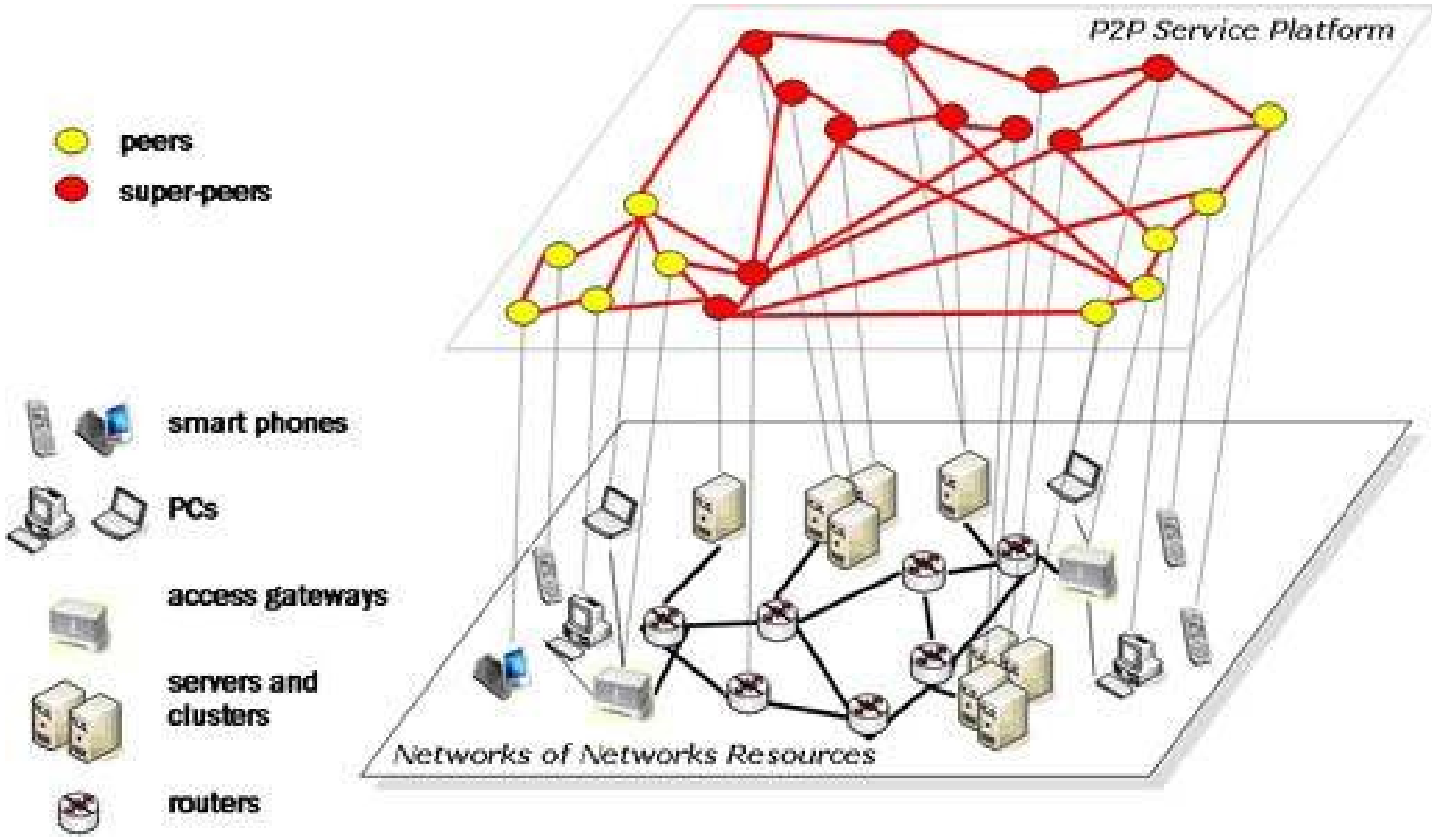
Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturadas Chord (protocolo)**
  - Nós estão logicamente organizados em um anel.
  - Item de dado com chave k e mapeado para o nó com o menor identificador id >= k → sucessor de k.
  - No é denominado sucessor da chave k.
  - Função LOOKUP(k), que retorna o endereço de rede succ(k)



Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturadas Chord (protocolo)**

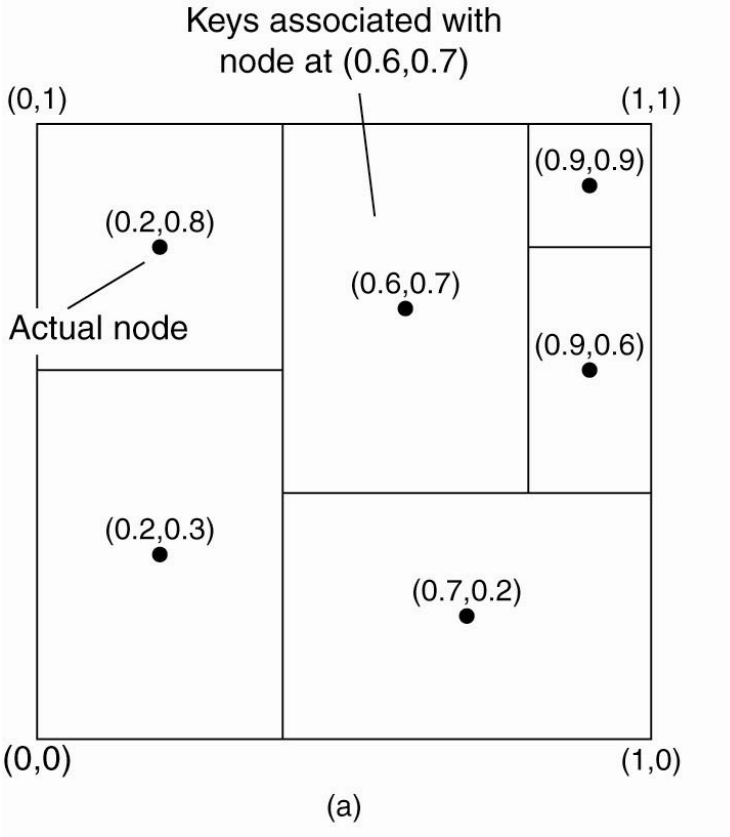


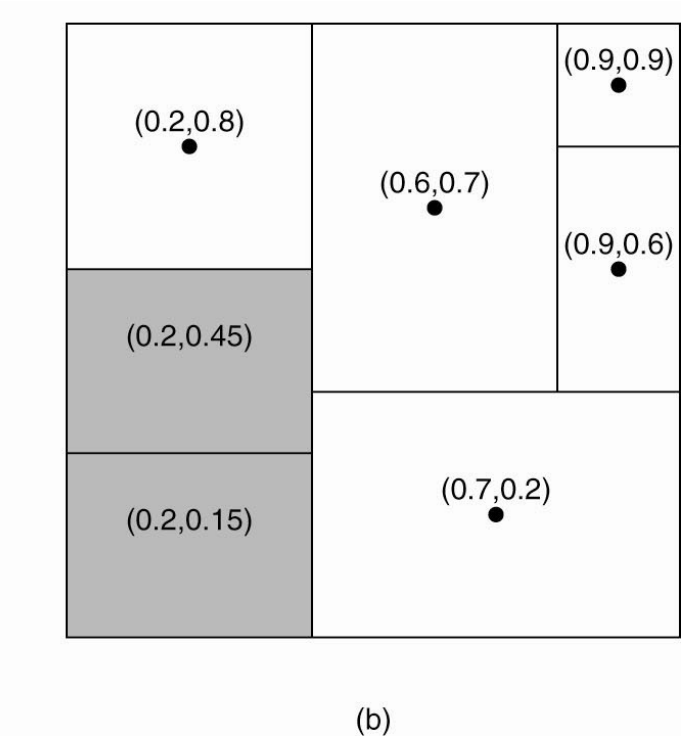
Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturadas Gerais**
  - Gerenciamento de associação ao grupo
    - » Ao entrar no sistema, o nó recebe um identificador aleatório *id.*
    - » Mas, como encontrar a posição no anel?
      - Pesquisa em *id* retorna o endereço de rede *succ*(id).
      - Novo no contata *succ*(id) e seu predecessor e se insere no anel.
      - Na partida, o no envia os dados para o *succ*(id).

Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Estruturadas**
  - Abordagens similares (rede de conteúdo endereçável)
  - CAN (Content Addressable Network)





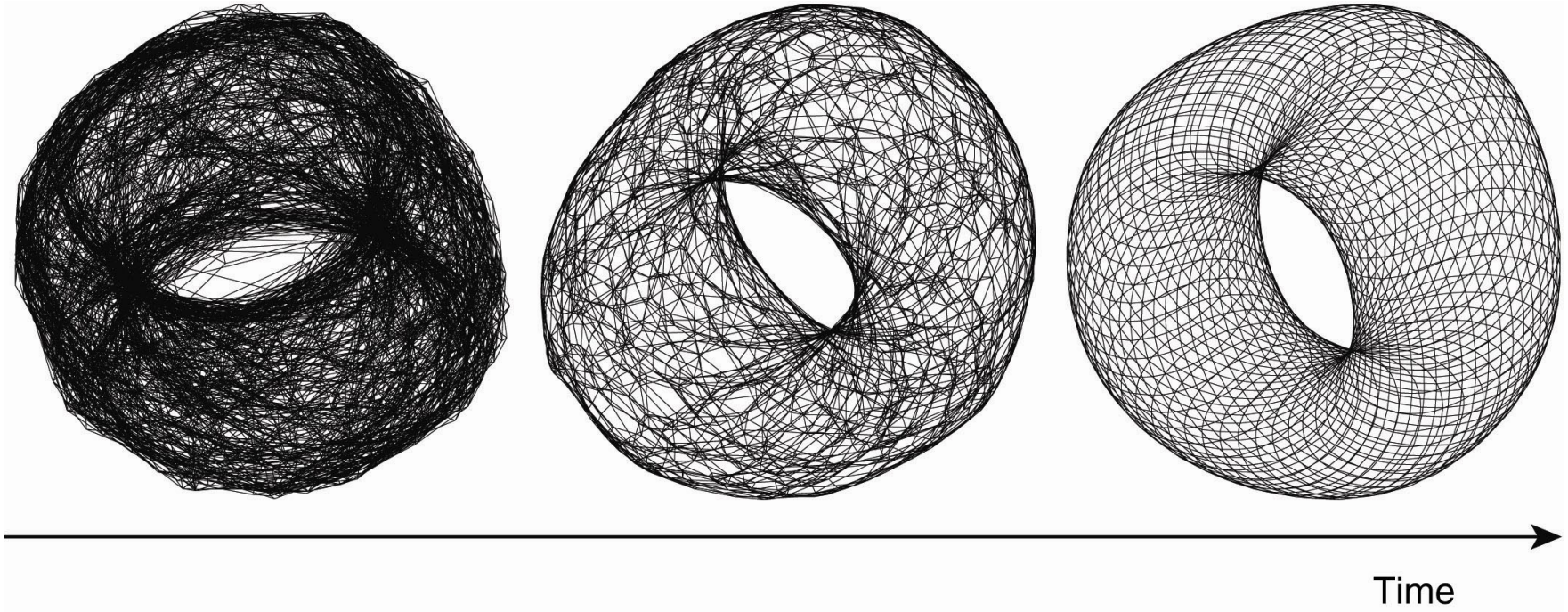
Arquitetura de Sistema

- **Arquiteturas Descentralizadas Peer-to-Peer Não Estruturadas**
  - Algoritmos aleatório são usados para construir a rede de sobreposição.
    - » Cada nó mantém uma lista de vizinhos.
    - » Dados também são espalhados aleatoriamente.
    - » Como encontrar os dados?
      - Inundar a rede com uma busca...

- **Arquiteturas Descentralizadas Peer-to-Peer Não Estruturadas**
  - Gerenciamento de associação ao grupo:
    - » Grafo aleatório.
    - » Cada no possui *n* vizinhos → **visão parcial.**
    - » Nós trocam entradas regularmente de sua visão parcial.
    - » Principal objetivo: atualizar saídas de nós, construir uma nova vizinhança de forma dinâmica para alcançar uma característica em específico.
    - » Nos trocam as listas de vizinhos em dois modos diferentes: *pull* (puxar) ou *push* (empurrar)
    - » Protocolos que usam somente *pull* ou *push* → grafos não conectados

Arquitetura de Sistema

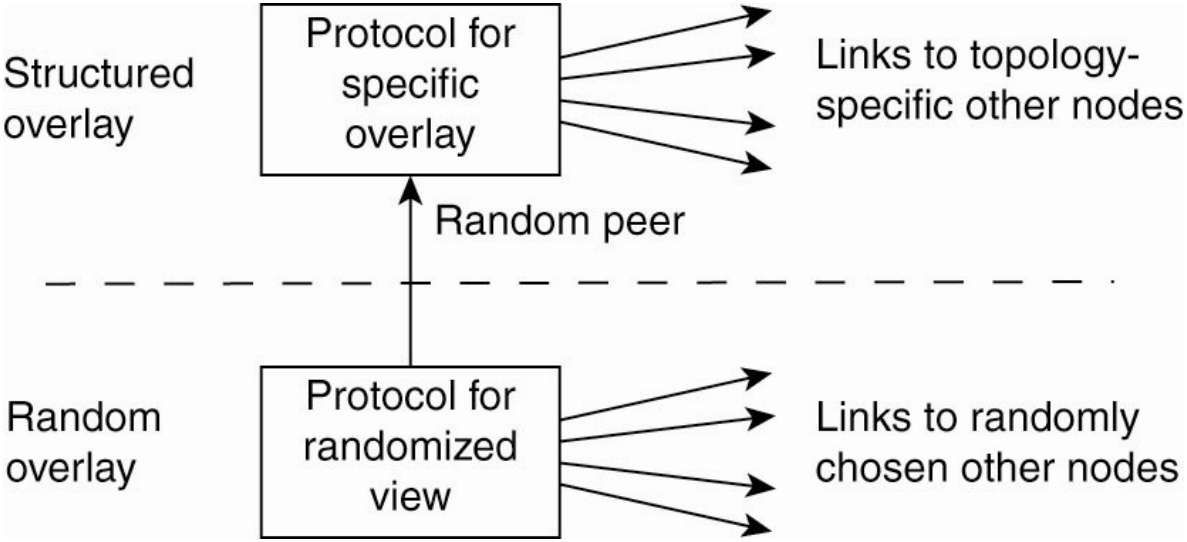
- **Arquiteturas Descentralizadas Peer-to-peer Não Estruturada**



Arquitetura de Sistema

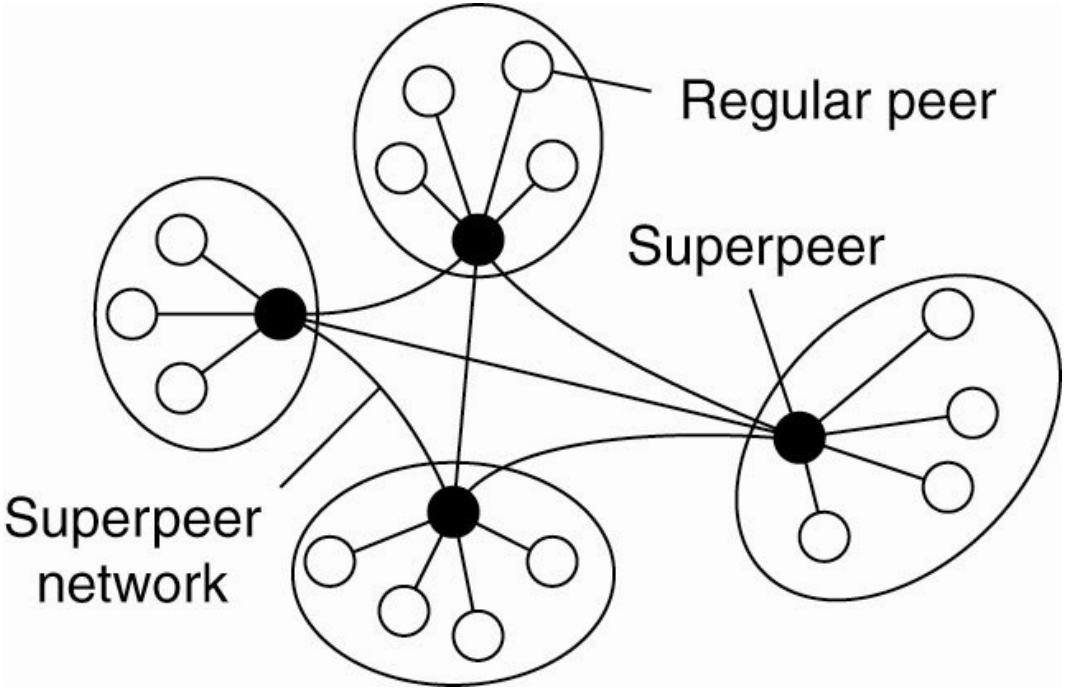
- **Arquiteturas Descentralizadas peer-to-peer: Gerenciamento de Redes de Sobreposição**

  - Abordagem de duas camadas
  - Camada mais baixa passa visão parcial para a mais alta, em que ocorre uma seleção adicional de entradas



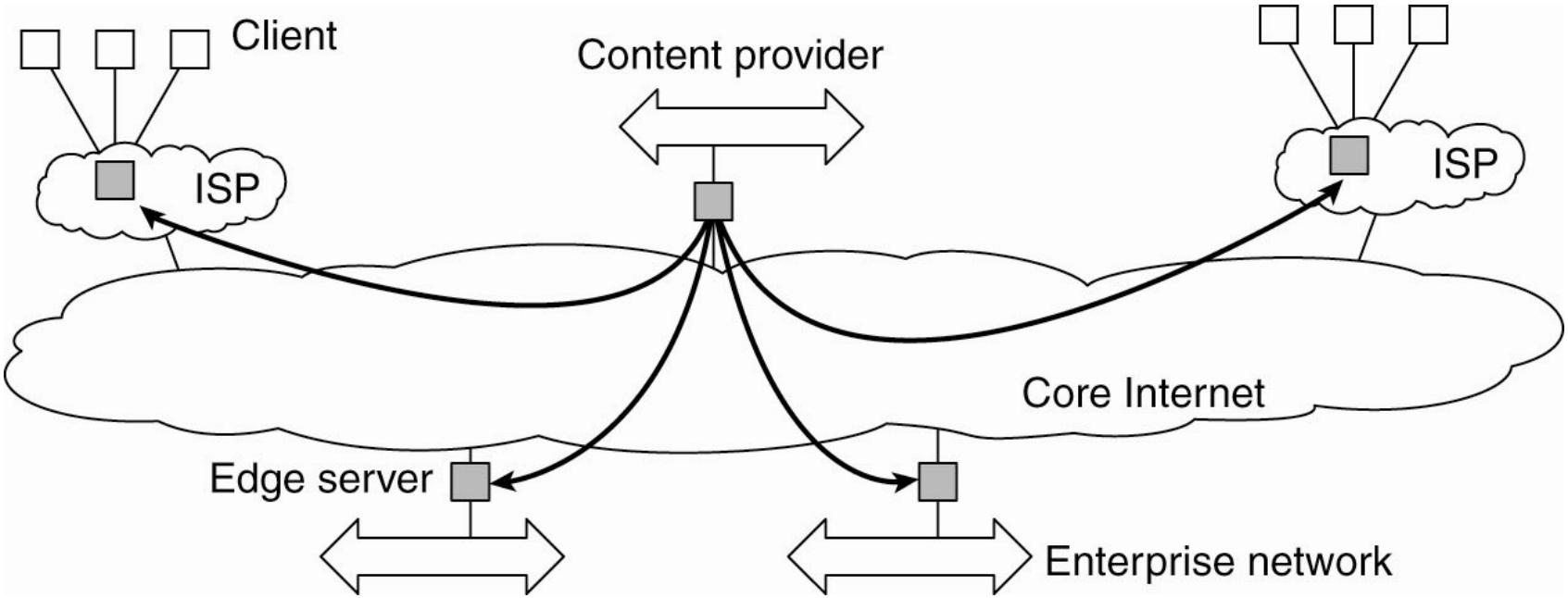
Arquitetura de Sistema

- **Arquiteturas Descentralizadas peer-to-peer: Superpares [superpeers]**
  - A medida que a rede cresce, localizar itens de dados em sistemas P2P não estruturados pode ser problemático.
  - Existe dificuldade nos nós que mantém o índice de dados ou que agem como nos intermediários que possuem dados para disponibilizar os recursos a nós vizinhos.
  - Sempre que um no comum se junta a rede, se liga a um dos superpares.



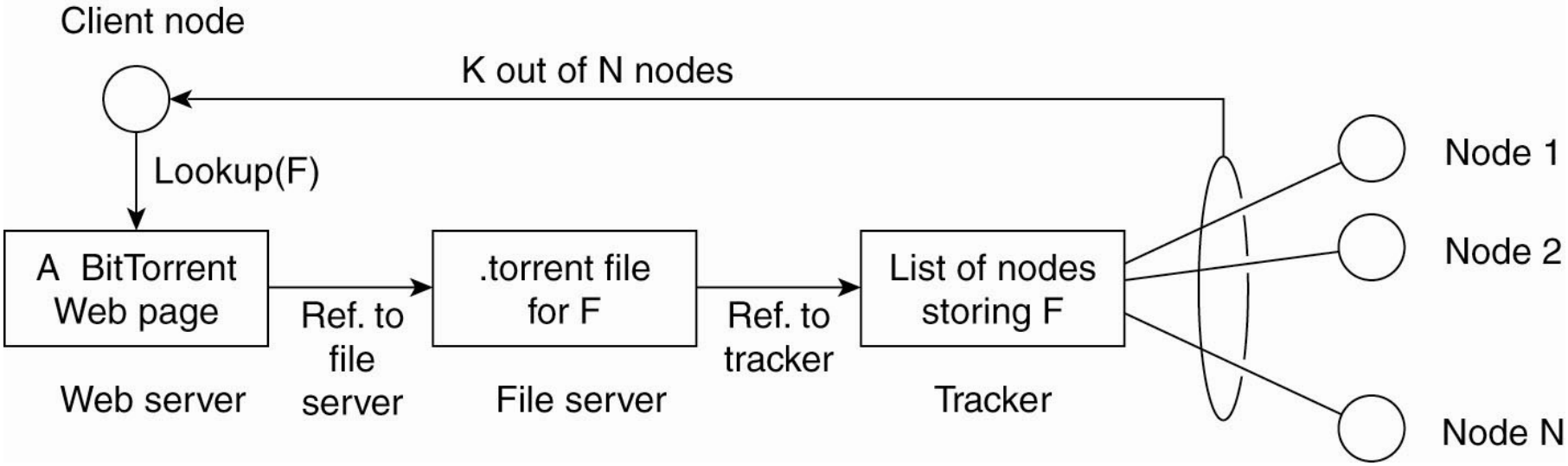
Arquitetura de Sistema

- **Arquiteturas Servidor de Borda**
  - Visão da Internet como uma rede composta por um conjunto de servidores de borda
  - ISP (Internet Service Provider)



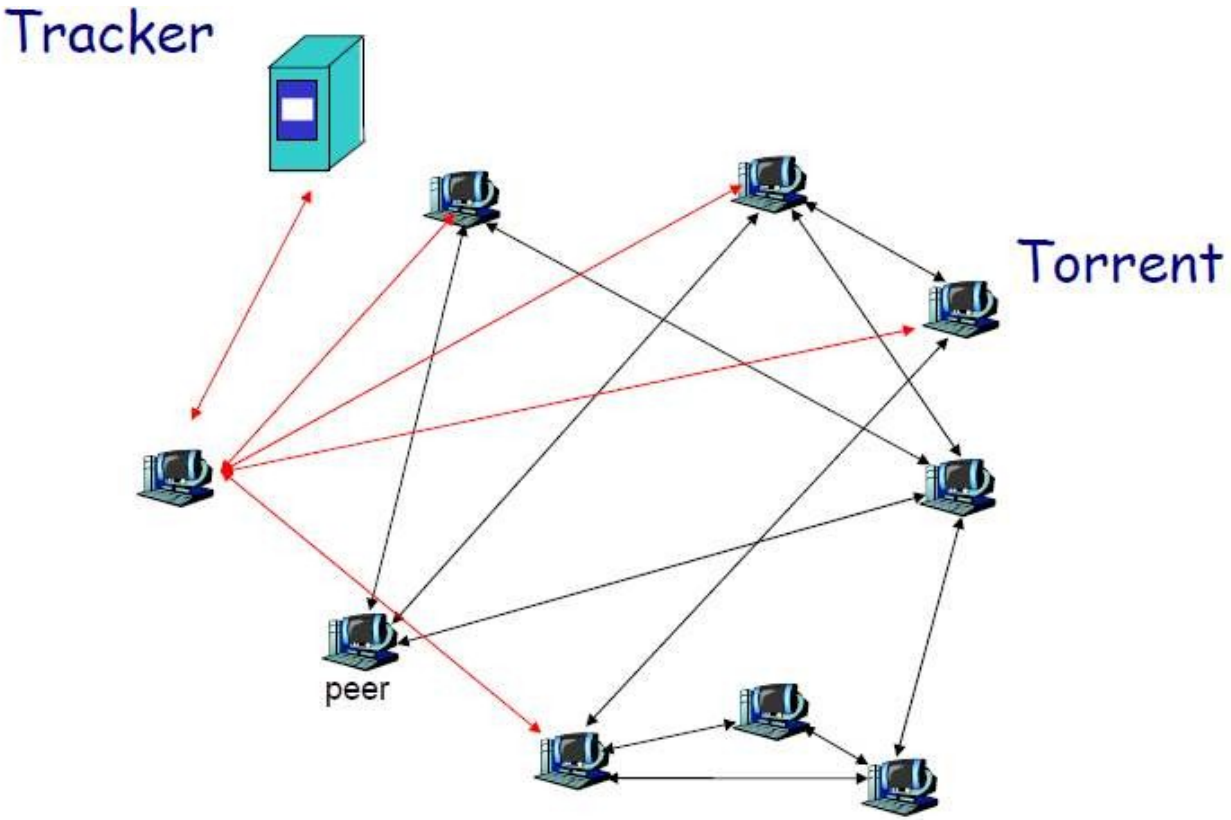
Arquitetura de Sistema

- **Arquiteturas Híbridas**
  - Sistemas distribuídos nas quais soluções clientes-servidor são combinadas com arquiteturas descentralizadas.
  - Exemplo: Sistemas distribuídos colaborativos.
    - » Principal objetivo e iniciar a troca de informações.
    - » Após adição do nó na rede, a distribuição dos dados é feita de forma descentralizada.
    - » BitTorrent
    - » Network Coding (Avalanche)



Arquitetura de Sistema

- **Arquiteturas Híbridas**

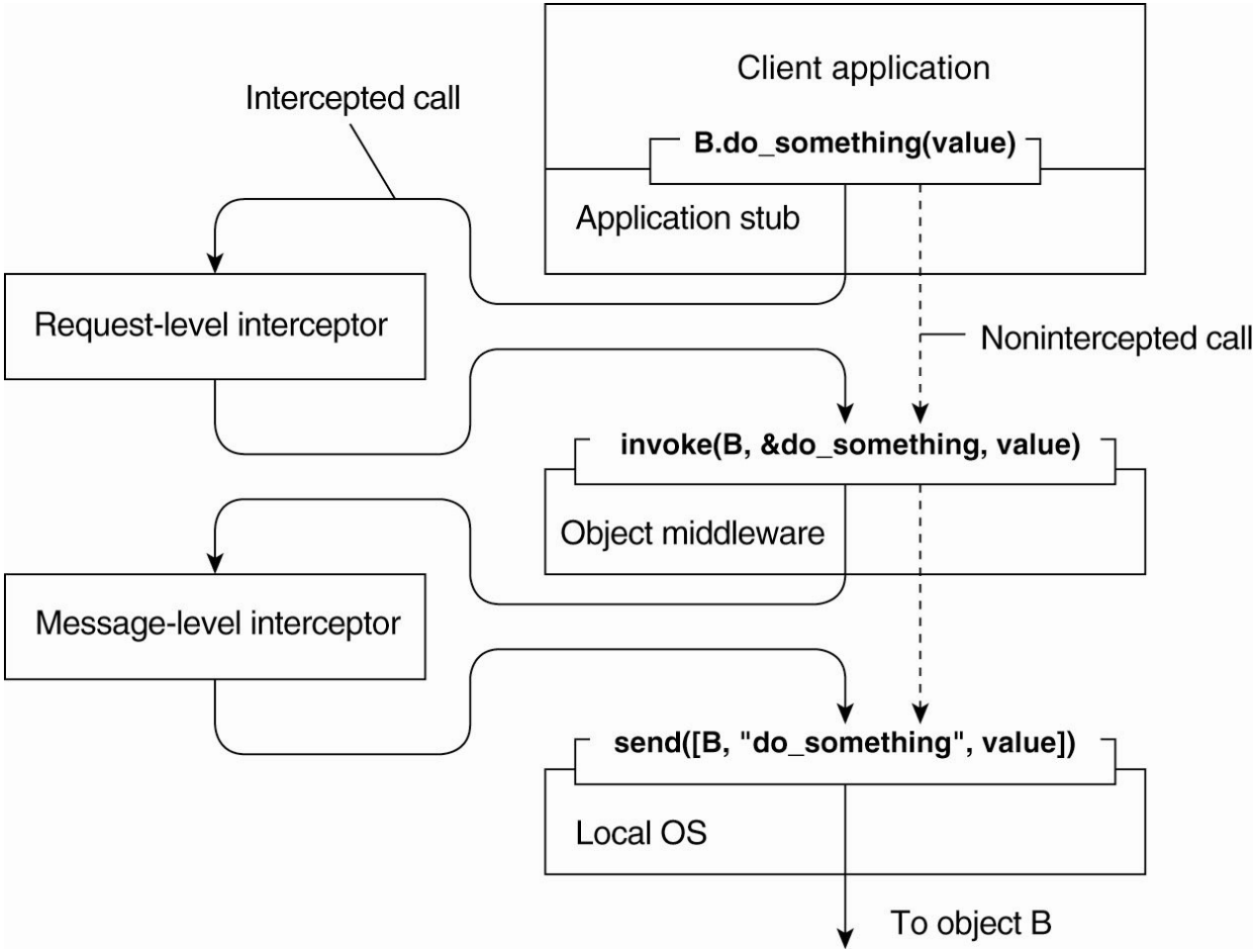


Arquiteturas *versus* Middleware

- Sistemas de middleware seguem um estilo arquitetônico especifico
  - Muitos middlewares adotam sistema arquitetônico baseado em objetos (CORBA,TIB/Rendezvous - www.tibco.com).
  - Ideia principal: desenvolver sistemas de middleware que sejam simples de configurar, adaptar e personalizar conforme necessidade da aplicação.
    - » Solução: Interceptadores (trata-se de um software que interromperá o fluxo de controle usual permitindo que seja executado um código específico da aplicação.

Arquiteturas *versus* Middleware

- **Interceptadores**



Arquiteturas *versus* Middleware

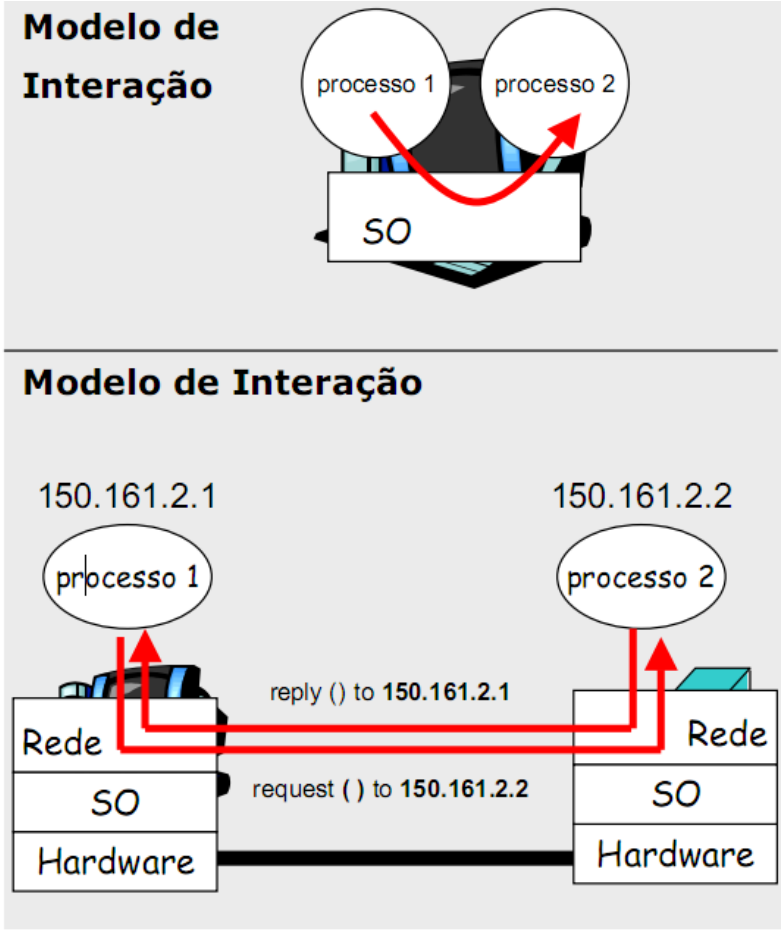
- Software que interrompera o fluxo de controle usual e permitira que seja executado um outro código.
  - Exemplo:
    - **A** simplesmente chama o método disponível na interface. Essa chamada original é transformada em uma chamada genérica, possibilitada por meio de uma interface geral de invocação de objeto oferecida pelo *middleware* na máquina onde **A** reside.
    - Finalmente, a invocação a objeto genérico é transformada em uma mensagem que é enviada por meio de uma interface de rede de nível de transporte como oferecida pelo sistema operacional local de **A**.

Arquiteturas *versus* Middleware

- **Aspectos sobre adaptação de software**
- Permitir que o software mude à medida que o ambiente muda, e questionar se a tal adaptação é uma boa adoção
- Razão que justifica o uso de softwares adaptativos é que um sistema distribuído não desliga (substituir ou atualizar componentes)
- Sistemas distribuídos devem ser capazes de reagir a mudanças em seu ambiente
  - Trocar dinamicamente políticas para alocação de recursos

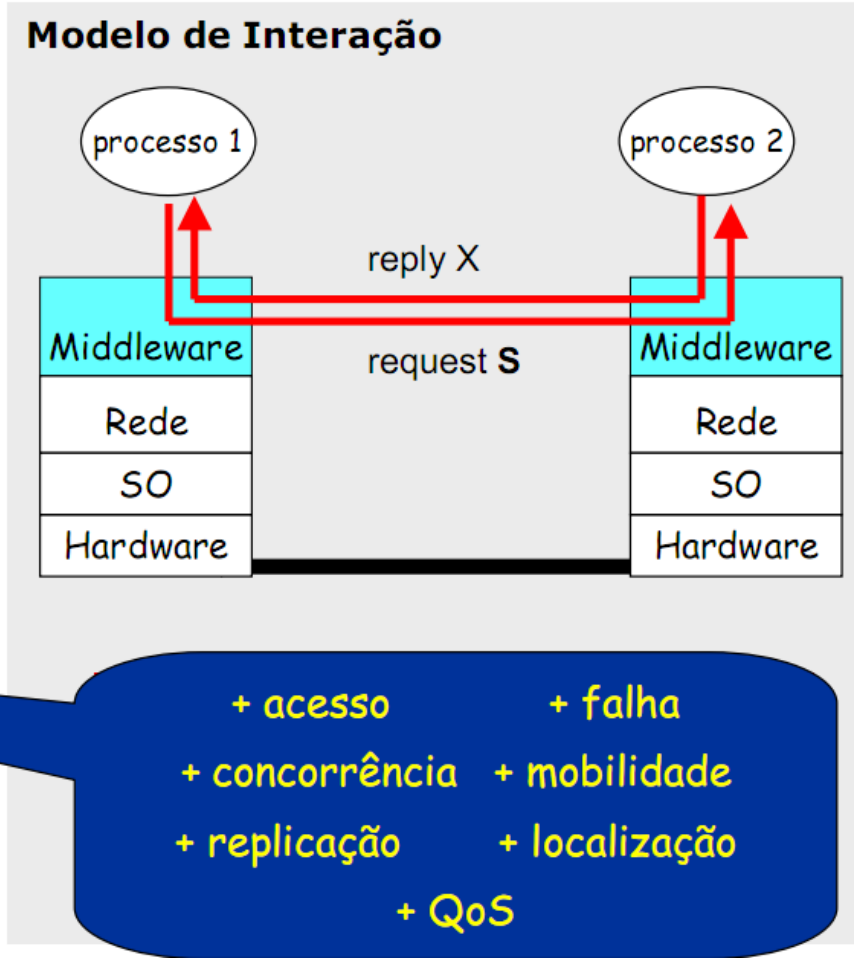
Arquiteturas *versus* Middleware

- Processos em uma mesma máquina
  - variável compartilhada, arquivo
  - o S.O "sabe" onde estão os processos
  - e.g., Word e Excel
- Processos em máquinas distintas
  - protocolos de comunicação
  - um processo "sabe" onde o outro está (endereço IP+porta)
  - e.g., aplicações em rede (correio eletrônico, web)



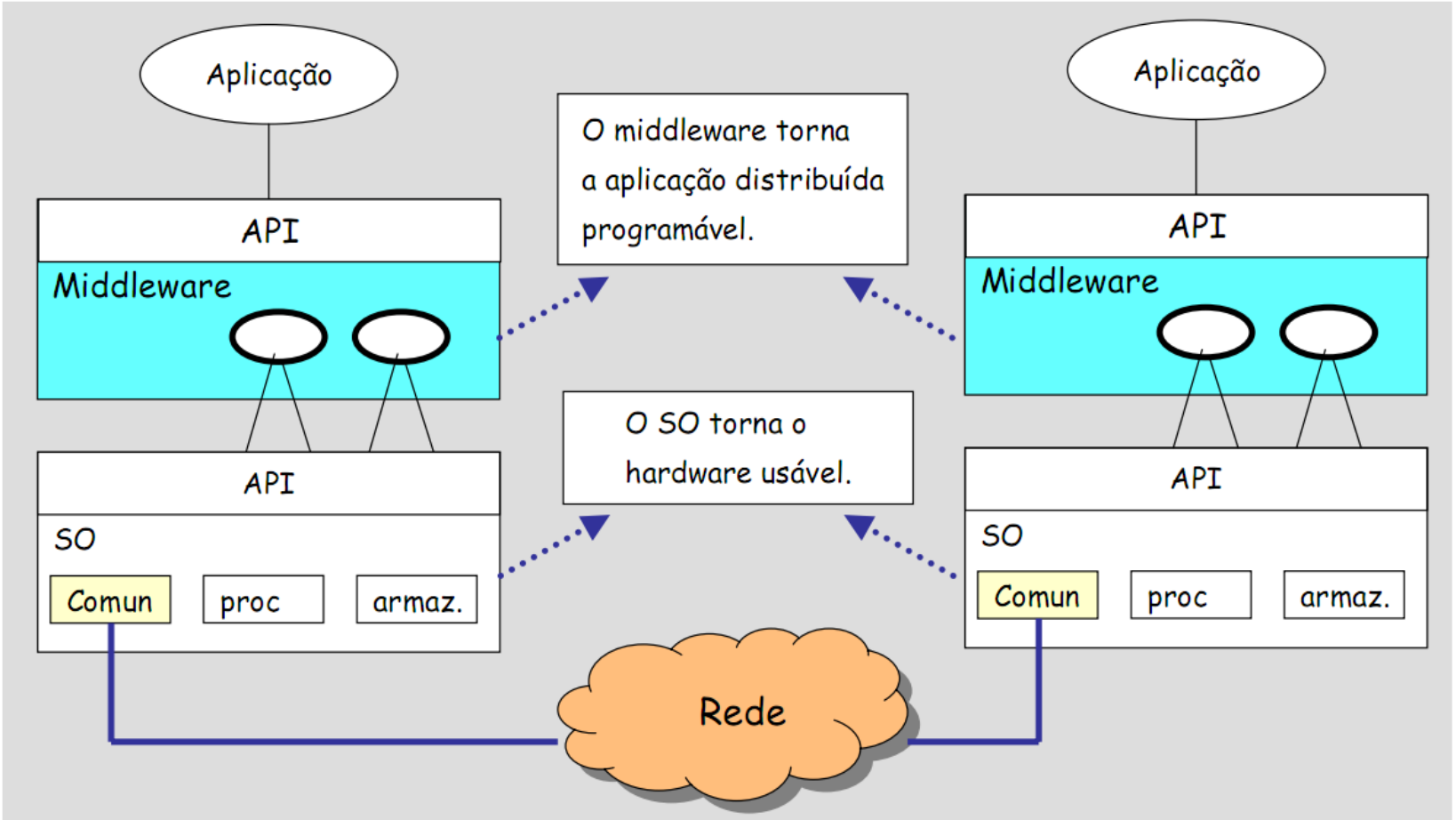
Arquiteturas *versus* Middleware

- camada de software que permite a comunicação entre aplicações (distribuídas)
- um conjunto de serviços que fornece comunicação e distribuição de forma transparente à aplicação
- componentes
  - ambiente de programação
  - ambiente de execução



Arquiteturas *versus* Middleware

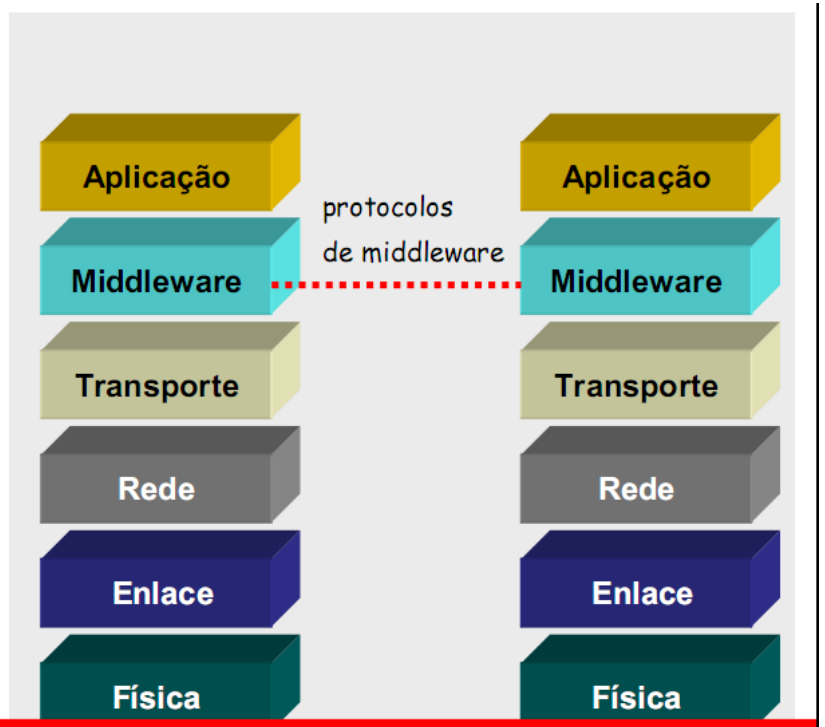
- **Contexto do middleware**



Arquiteturas *versus* Middleware

- **Contexto do middleware**

- o *middleware* está localizado entre as camadas de aplicação e transporte
- o *middleware* implementa seus próprios protocolos
  - protocolos que suportam os serviços fornecidos pelo *middleware*
    - e.g., protocolos de autenticação



ⓘ O middleware faz o papel da camada de apresentação no RM-OSI.

Arquiteturas *versus* Middleware

- **Contexto do middleware**

- comunicação
  - esconde os detalhes da rede
  - e.g., chamada de procedimento remoto, invocação de objeto
- serviço de nomes
  - e.g., páginas amarelas

- transações
  - e.g., atomicidade
- segurança
  - a camada de *middleware* deve implementar mecanismos de segurança
  - pervasiva (ubíqua)

ⓘ *O serviços fornecidos pelos middleware variam de ambiente para ambiente.*

Arquiteturas *versus* Middleware

- **Middleware para dispositivos móveis**

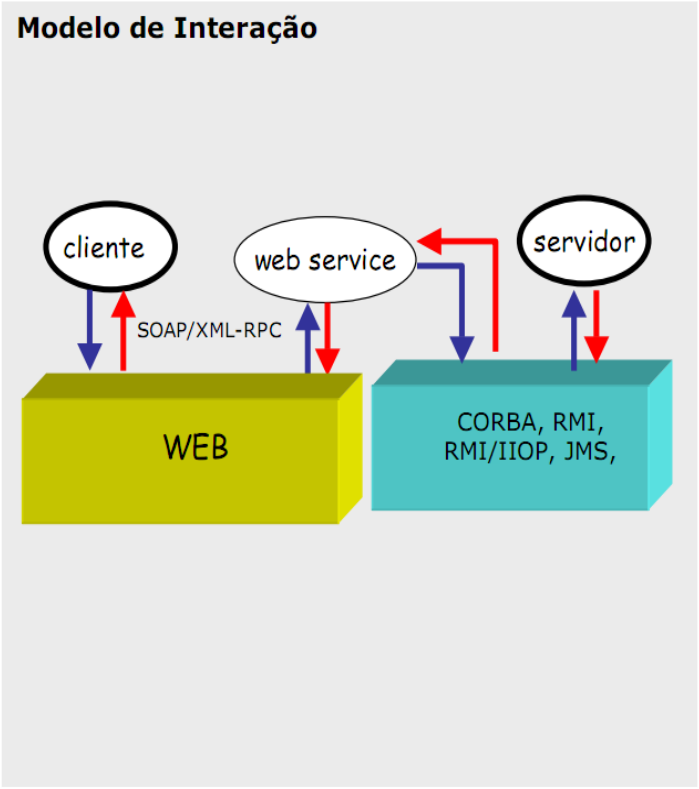
- carga computacional leve
  - os dispositivos móveis tem limitações de processamento, armazenamento
- comunicação assíncrona
  - clientes e servidores nem sempre estão conectados
- reconfiguração dinâmica
  - e.g., largura de banda, serviços disponíveis
- desenvolvedor ciente das mudanças (contexto)

- tipos de *middleware* móveis
  - reflectivos
    - reconfiguráveis (reflectivos)
    - *light-weight*
  - espaço de tuplas
    - comunicação assíncrona
  - *context-aware*
    - infomações contextuais são fornecidas a aplicação

Arquiteturas *versus* Middleware

- **Web Service e Middlewares**

- *Web Services* agem como uma "interface" para acessar os serviços providos por outros *middleware*



Autogerenciamento em Sistemas Distribuídos

- Sistemas Distribuídos precisam fornecer soluções gerais de blindagem contra aspectos indesejáveis inerentes a redes.
- Objetivo: suportar o maior numero possível de aplicações.
- Solução: Sistemas Distribuídos adaptativos.
- Ideia: Construir sistemas em que seja possível fazer monitoração e ajustes.