ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

ARQUITETURAS DE SISTEMAS DISTRIBUÍDOS

Arquiteturas

- Estilos arquiteturais
- Arquiteturas de software
- Arquiteturas versus middleware
- Sistemas distribuídos autogerenciáveis

Estilos Arquiteturais

Ideia básica
Um estilo é definido em termos de:

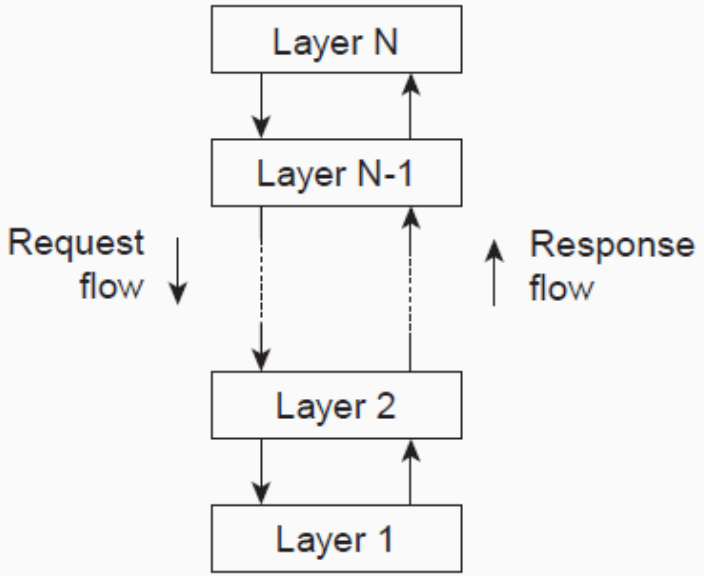
- componentes (substituíveis) com interfaces bem definidas
- o modo como os componentes são conectados entre si
- como os dados são trocados entre componentes
- como esses componentes e conectores são configurados conjuntamente em um sistema

Conector
Um mecanismo que intermedeia comunicação, coordenação ou cooperação entre componentes. Exemplo: recursos para chamadas de procedimento (remotos), mensagens ou *streaming*.

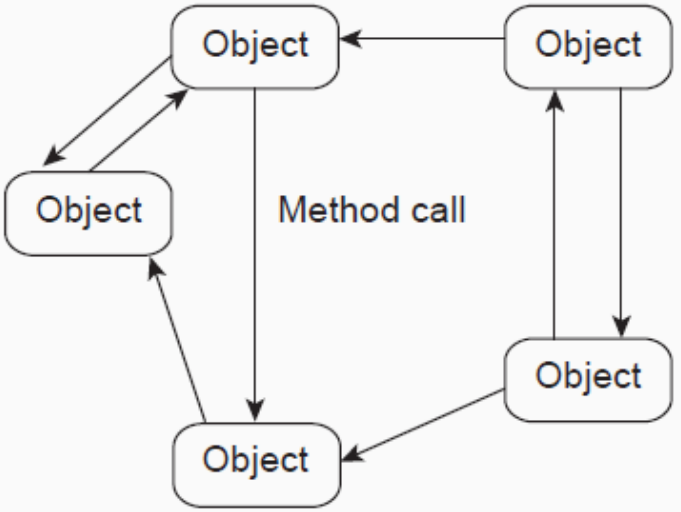
Estilos Arquiteturais

Ideia básica

Organize em componentes logicamente diferentes e os distribua entre as máquinas disponíveis.



(a)



(b)

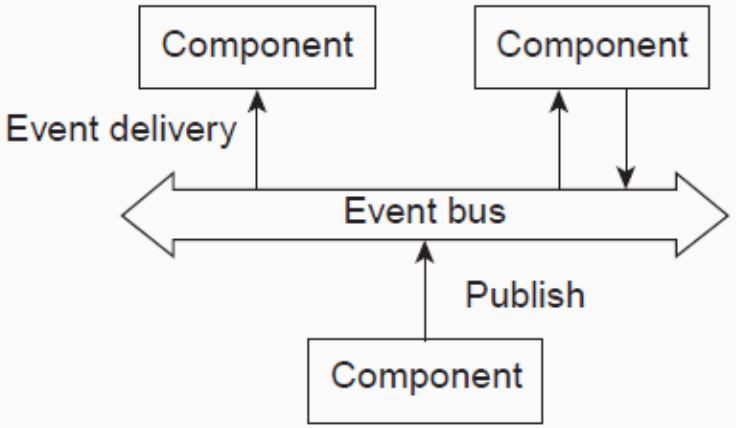
(a) Estilo em camadas é usado em sistemas cliente-servidor

(b) Estilo orientado a objetos usado em sistemas de objetos distribuídos.

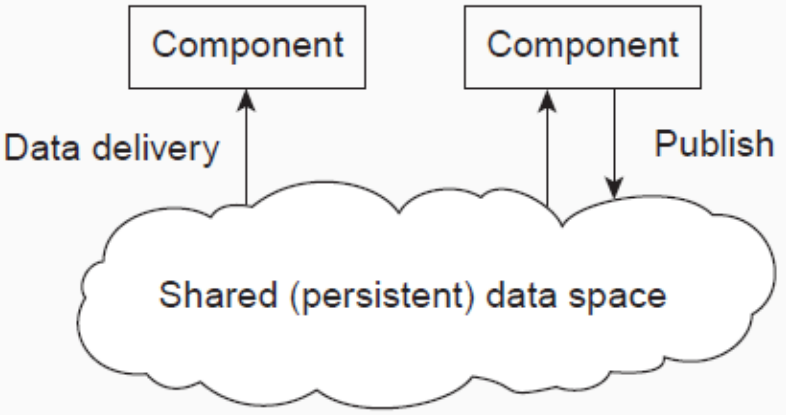
Estilos Arquiteturais

Observação

Desacoplar processos no espaço (anônimos) e tempo (assíncronos) pode levar a estilos diferentes.



(a)



(b)

(a) Publish/subscribe [desaclopado no espaço]

(b) Espaço de dados compartilhados [desacoplado no espaço e tempo]

Estilo Multicamada

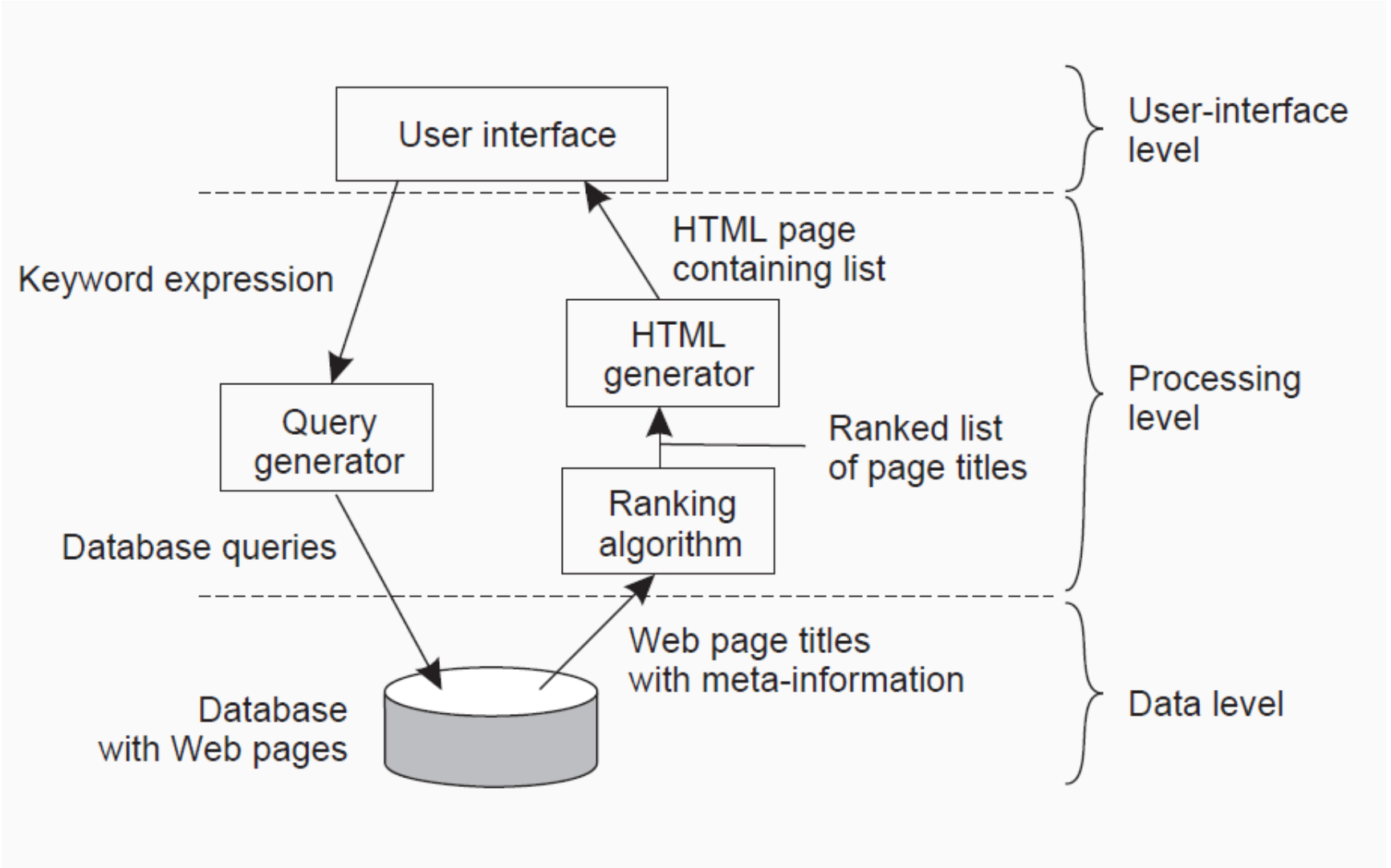
Visão tradicional em três camadas

- A camada de apresentação contém o necessário para a aplicação poder interagir com o usuário
- A camada de negócio contém as funções de uma aplicação
- A camada de dados contém os dados que o cliente quer manipular através dos componentes da aplicação

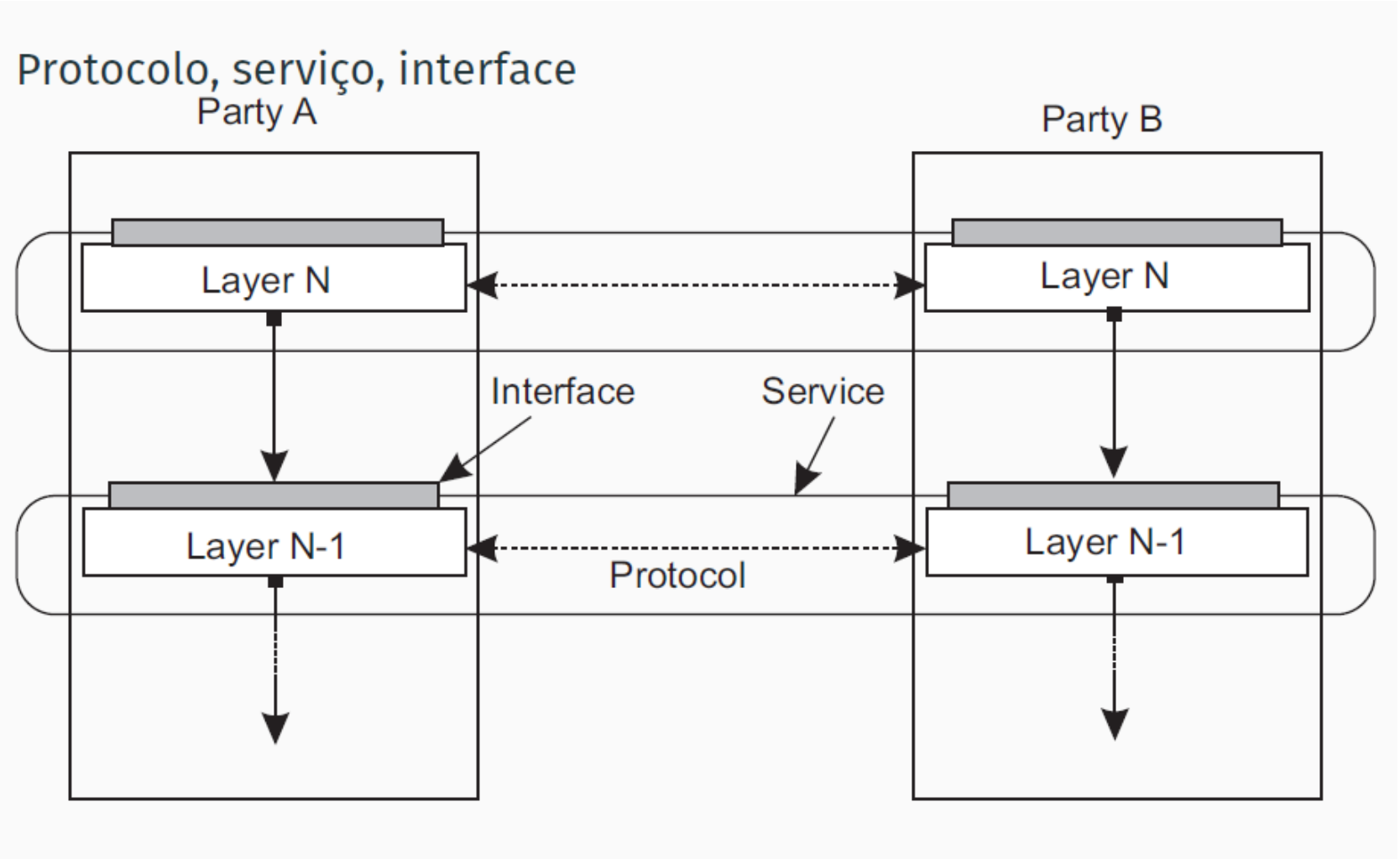
Observação

Estas camadas são encontradas em muitos sistemas de informação distribuídos, que usam tecnologias de bancos de dados tradicionais e suas aplicações auxiliares.

Estilo Multicamada



Ex: Protocolos de Comunicação



Comunicação entre as partes

Servidor

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
(conn, addr) = s.accept()  # returns new socket and addr. client
while True:                # forever
  data = conn.recv(1024)   # receive data from client
  if not data: break       # stop if client stopped
  conn.send(str(data)+"*") # return sent data plus an "*"
conn.close()               # close the connection
```

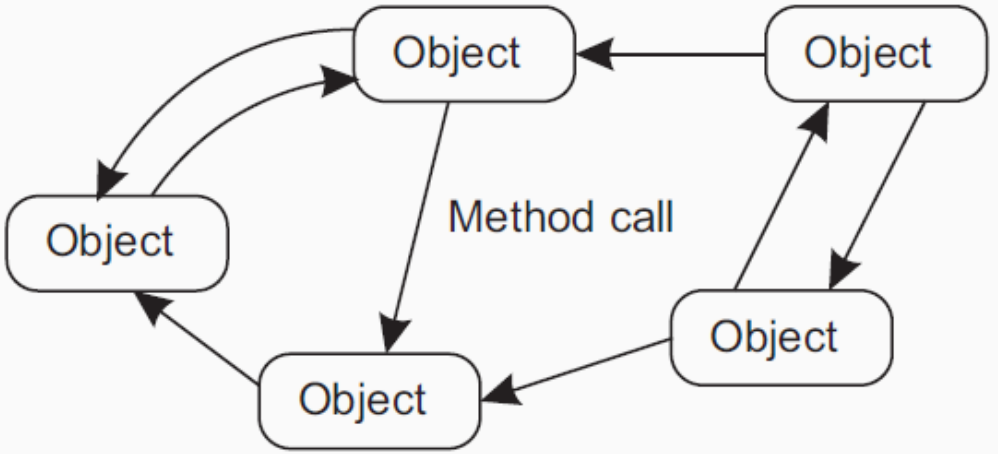
Cliente

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT)) # connect to server (block until accepted)
s.send('Hello, world')  # send some data
data = s.recv(1024)     # receive the response
print data              # print the result
s.close()               # close the connection
```

Estilo Arq. Baseado em Objetos

Essência

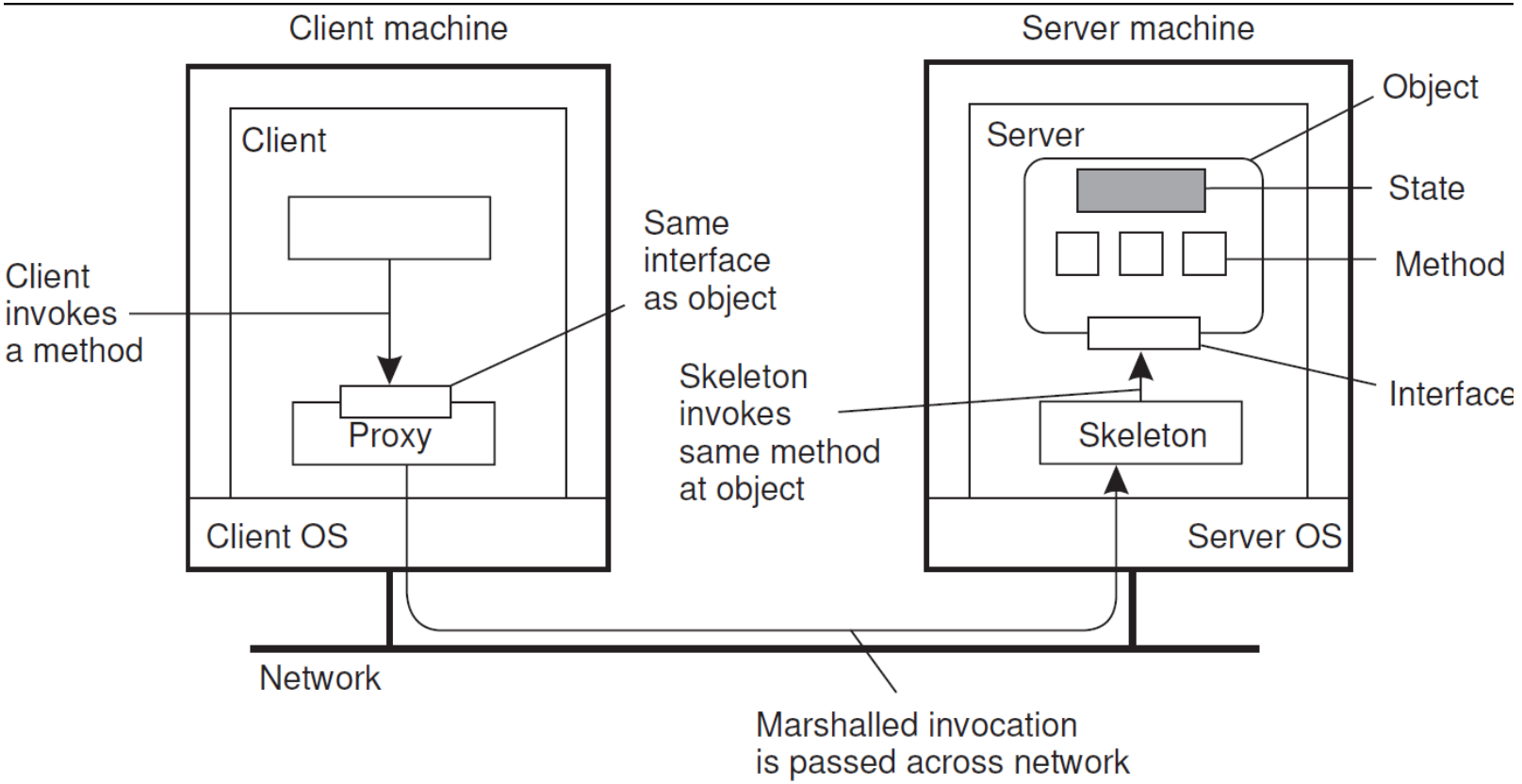
Componentes são objetos, conectados entre si usando chamadas de procedimentos. Objetos podem ser colocados em máquinas diferentes; chamadas, por tanto, devem executar usando a rede.



Encapsulamento

Dizemos que um objeto *encapsula dados* e oferece *métodos para os dados* sem revelar sua implementação.

Objetos Distribuídos



Arquiteturas RESTFUL

Estilo centrado em recurso

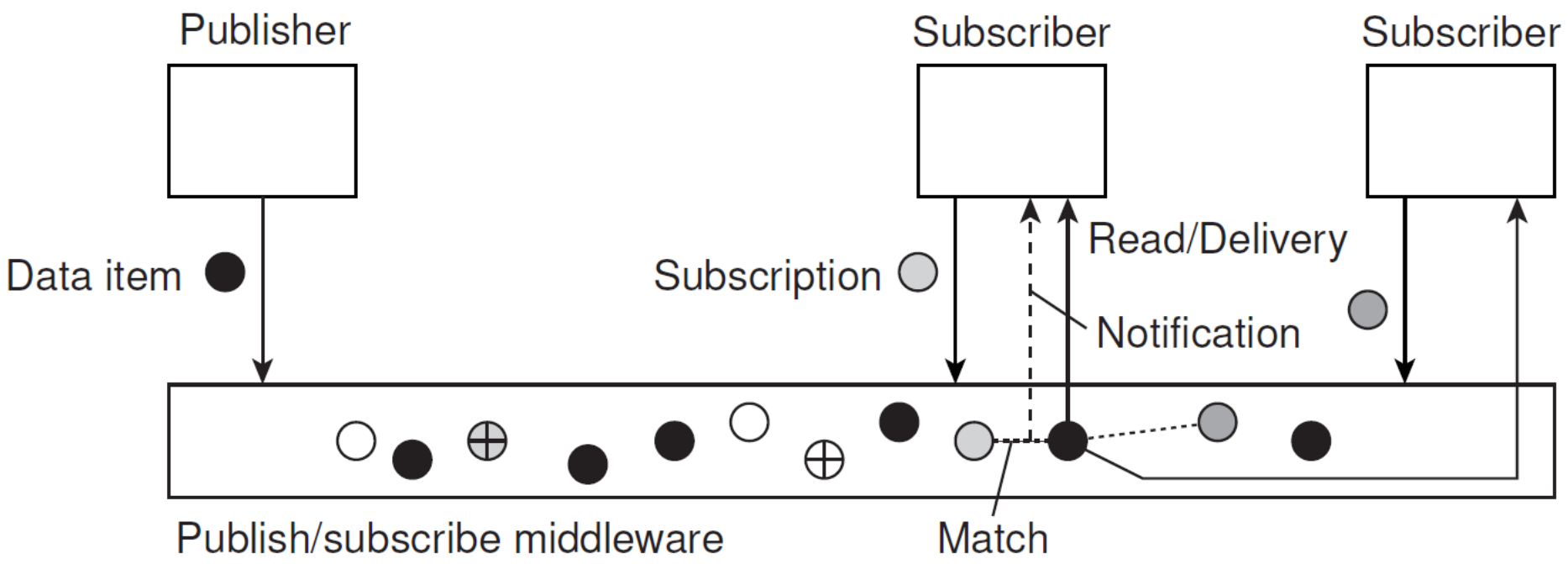
Vê um sistema distribuído como uma coleção de recursos que são gerenciados individualmente por componentes. Recursos podem ser adicionados, removidos, recuperados e modificador por aplicações (remotas).

1. Recursos são identificados usando um único esquema de nomeação
2. Todos os serviços oferecem a mesma interface
3. Mensagens enviadas de ou para um serviço são auto-descritivas
4. Após a execução de uma operação em um serviço, o componente esquece tudo sobre quem chamou a operação

Operações básicas

Operação	Descrição
PUT	Cria um novo recurso
GET	Recupera o estado de um recurso usando um tipo de representação
DELETE	Apaga um recurso
POST	Modifica um recurso ao transferir um novo estado

Arquiteturas Baseadas em Eventos



Estilos Arquiteturais Coordenação

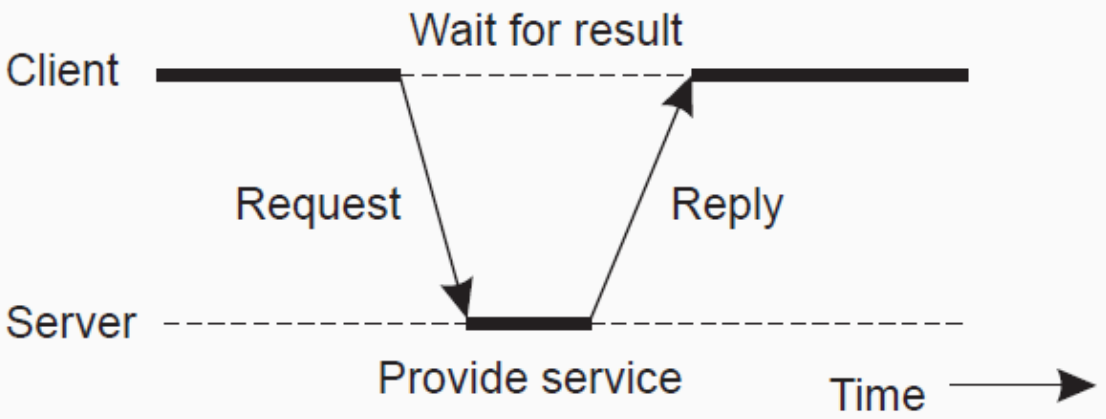
	Temporalmente Acoplado	Temporalmente Desacoplado
Referencialmente Acoplado	Direto	Mailbox
Referencialmente Desacoplado	Evento	Espaço de dados compartilhados

Arquiteturas de Sistemas

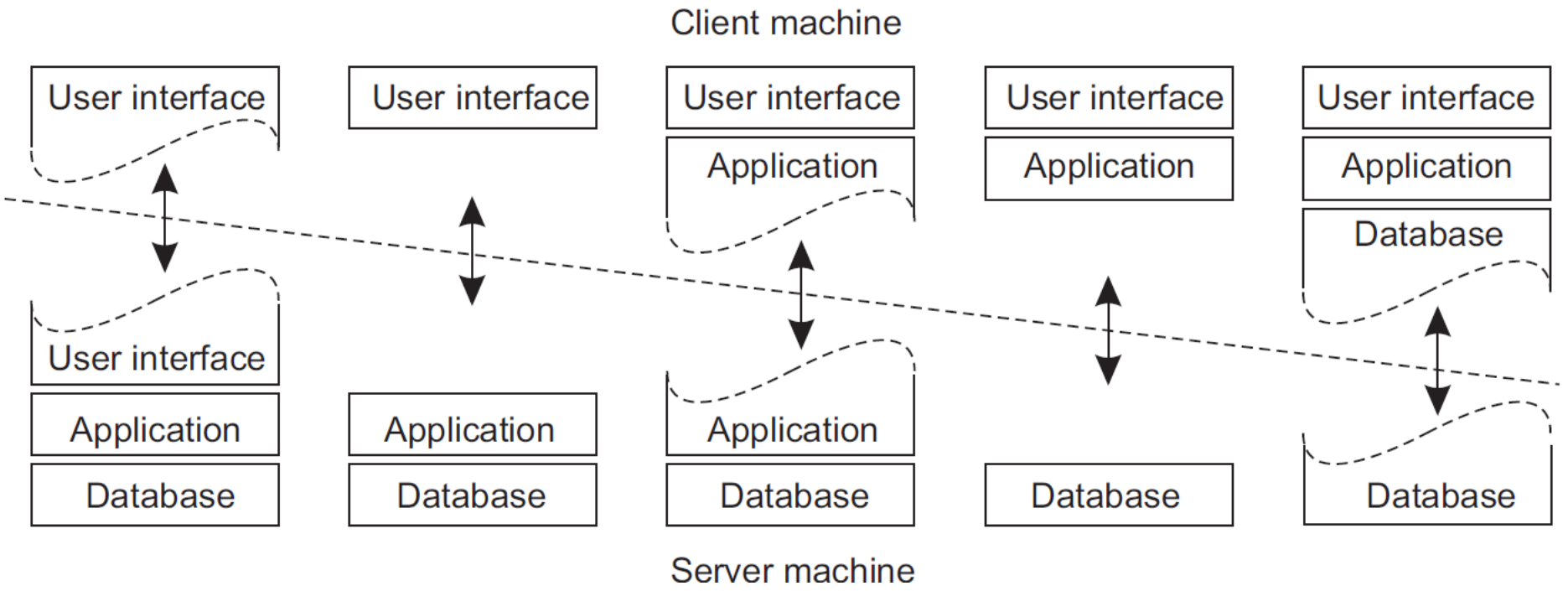
Organização centralizada

Características do modelo Cliente–Servidor

- Existem processos que oferecem serviços (servidores)
- Existem processos que usam esses serviços (clientes)
- Clientes e servidores podem estar em máquinas diferentes
- Clientes seguem um modelo requisição/resposta ao usar os serviços



Cliente-Servidor Multinível



Organizações Descentralizadas

P2P estruturado os nós são organizados seguindo uma estrutura de dados distribuída específica

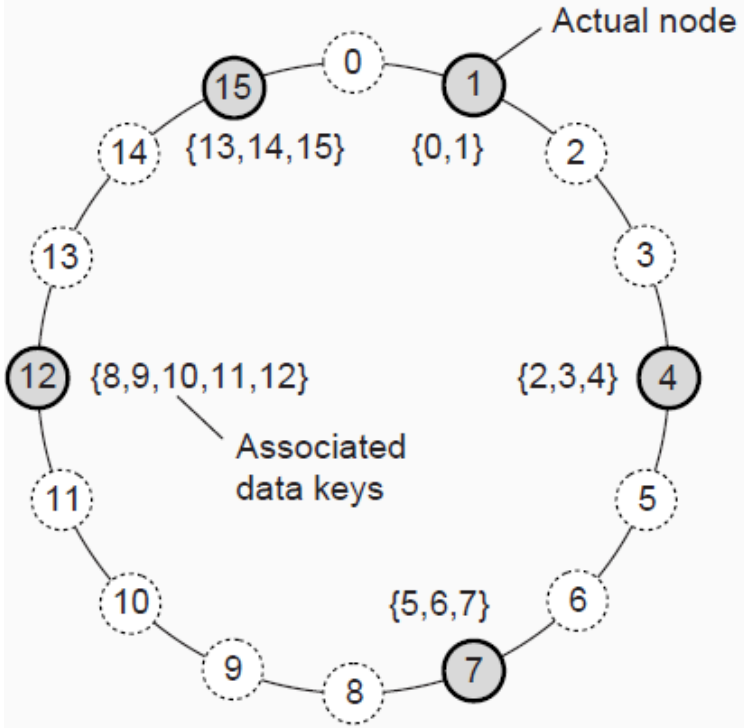
P2P não-estruturado os nós selecionam aleatoriamente seus vizinhos

P2P híbrido alguns nós são designados, de forma organizada, a executar funções especiais

P2P estruturados

Ideia básica

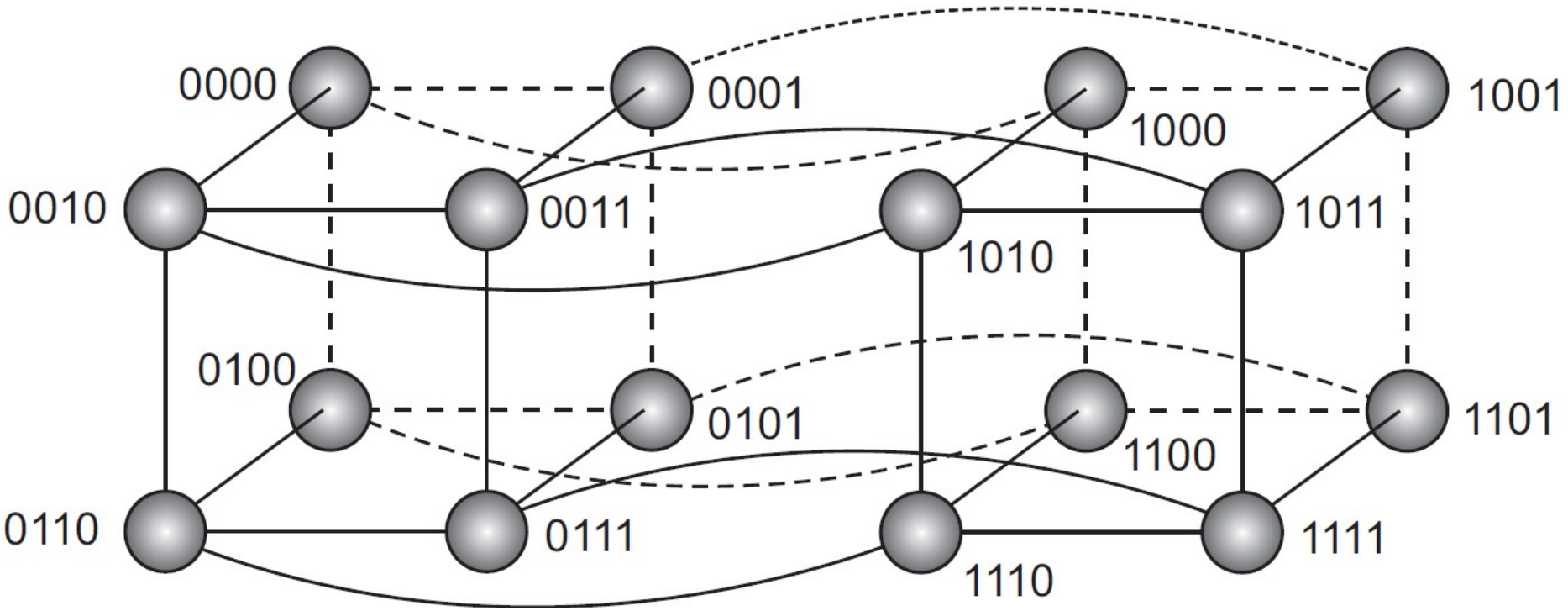
Organizar os nós em uma rede overlay estruturada, tal como um anel lógico, e fazer com que alguns nós se tornem responsáveis por alguns serviços baseado unicamente em seus IDs.



Nota

O sistema provê uma operação `LOOKUP(key)` que irá fazer o roteamento de uma requisição até o nó correspondente.

P2P estruturados



P2P não-estruturados

Observação
Muitos sistemas P2P não-estruturados tentam manter um grafo aleatório.

Princípio básico
Cada nó deve contactar um outro nó selecionado aleatoriamente:

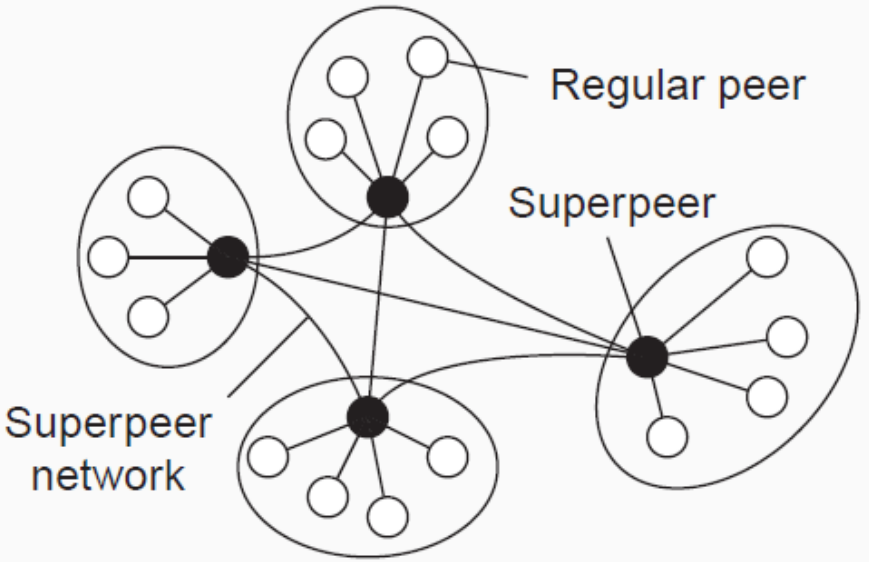
- Cada participante mantém uma visão parcial da rede, consistindo de *c* outros nós
- Cada nó *P* seleciona periodicamente um nó *Q* de sua visão parcial
- *P* e *Q* trocam informação && trocam membros de suas respectivas visões parciais

Nota
Dependendo de como as trocas são feitas, não só a aleatoriedade mas também a robustez da rede pode ser garantida.

Superpeers

Observação

Às vezes, selecionar alguns nós para realizar algum trabalho específico pode ser útil.



Exemplos:

- Peers para manter um índice (para buscas)
- Peers para monitorar o estado da rede
- Peers capazes de configurar conexões

Princípio de operação do Skype

Tanto A quanto B estão na Internet pública

- Uma conexão TCP é estabelecida entre A e B para envio de pacotes de controle
- A chamada real usa pacotes UDP entre as portas negociadas

A está atrás de um firewall, B está na Internet pública

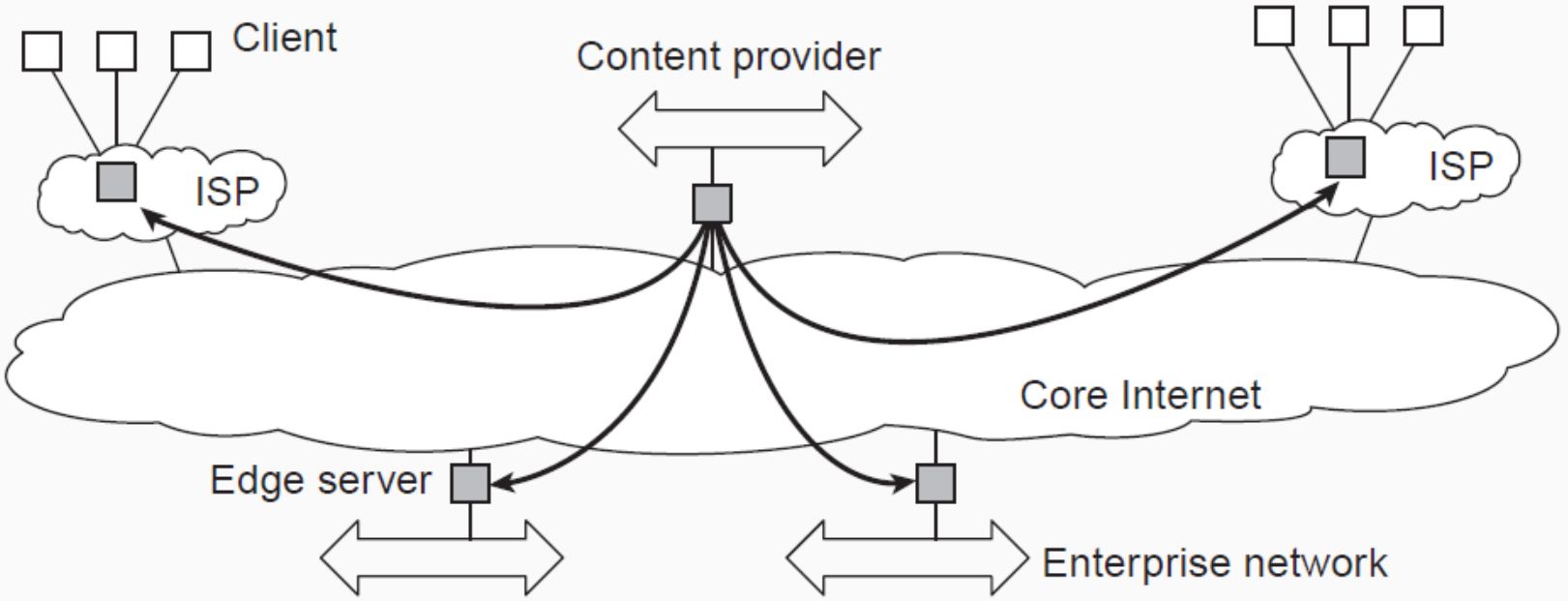
- A configura uma conexão TCP (para os pacotes de controle) com um superpeer S
- S configura uma conexão TCP (para redirecionar os pacotes de controle) com B
- A chamada real usa pacotes UDP diretamente entre A e B

Tanto A quanto B estão atrás de um firewall

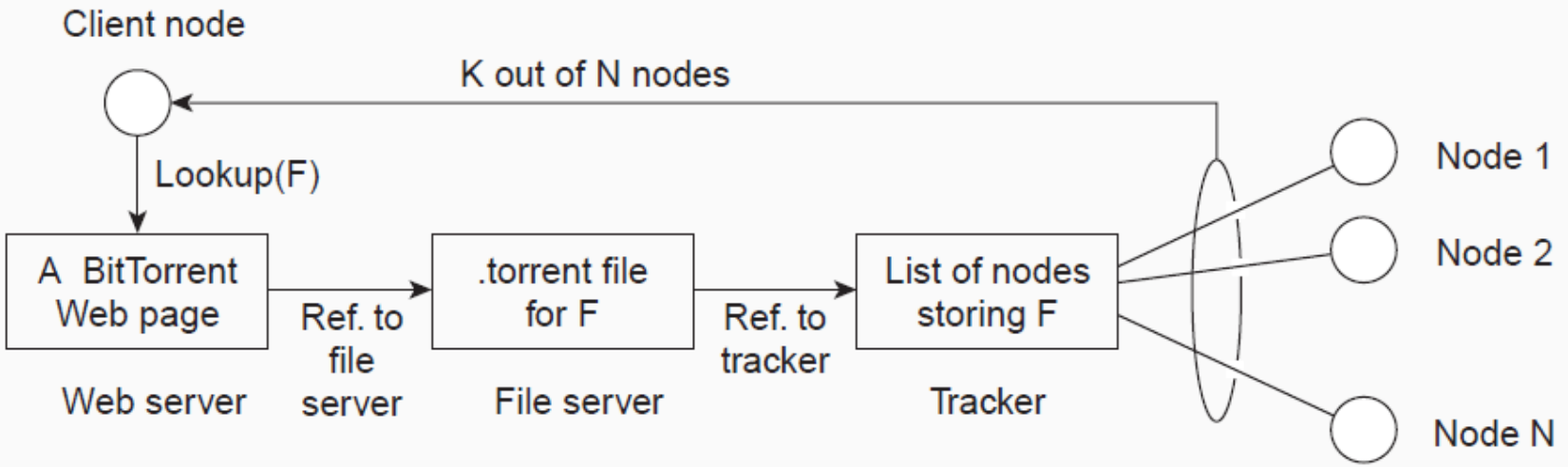
- A conecta com um superpeer S via TCP
- S configura uma conexão TCP com B
- Para a chamada real, outro superpeer é usado para funcionar como retransmissor (relay): A (e B) configura a conexão com R
- A chamada é encaminhada usando duas conexões TCP, usando R como intermediário

Arquiteturas Híbridas

Exemplo:
Arquiteturas de servidores de borda (*edge-server*), utilizados com frequência como Content Delivery Networks (redes de distribuição de conteúdo).



Arquiteturas Híbridas - Bittorrent



Ideia básica

Assim que um nó identifica de onde o arquivo será baixado, ele se junta a uma swarm (multidão) de pessoas que, em paralelo, receberão pedaços do arquivo da fonte e redistribuirão esses pedaços entre os outros.

Arquiteturas versus Middleware

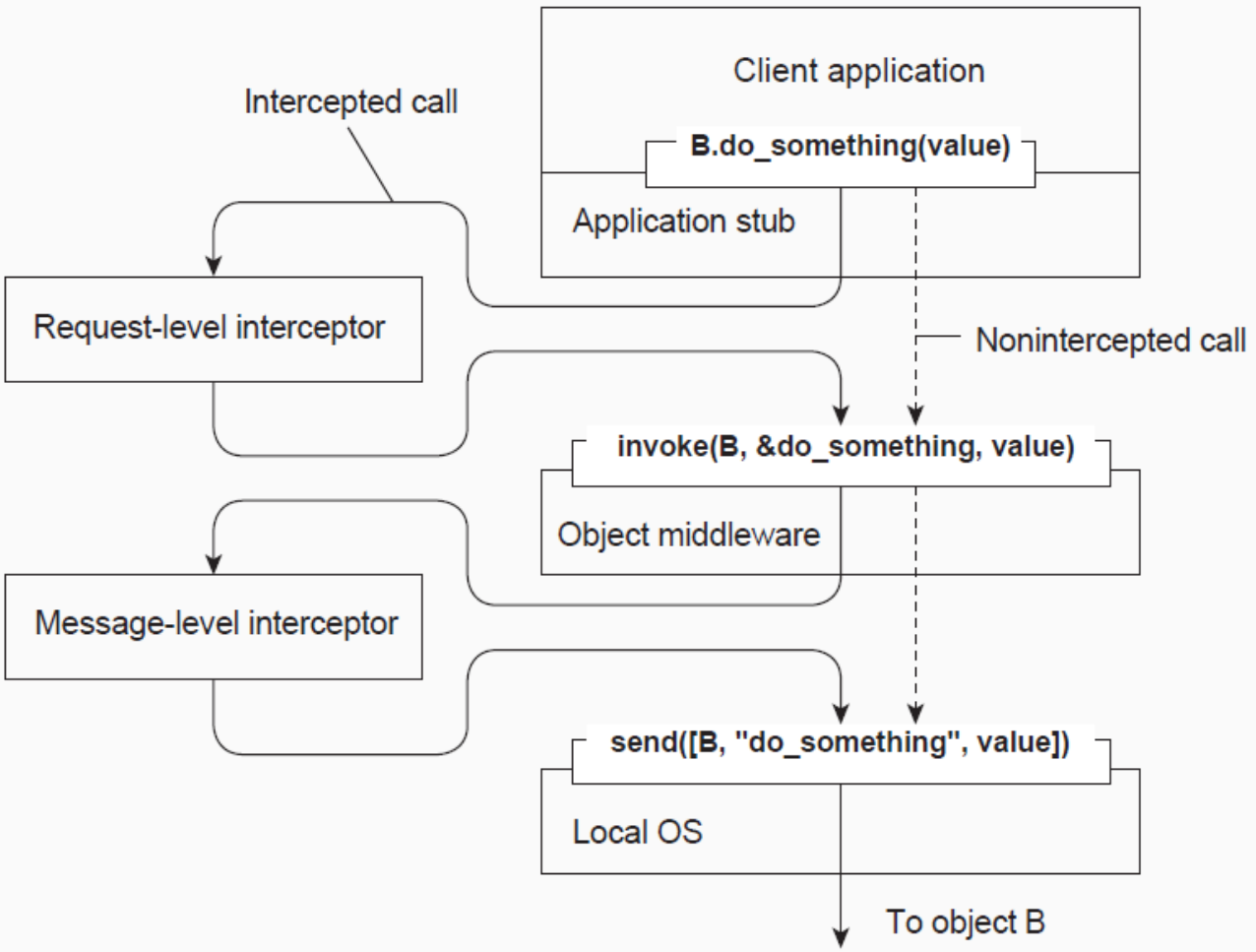
Problema

Em muitos casos, arquiteturas/sistemas distribuídos são desenvolvidos de acordo com um estilo arquitetural específico. O estilo escolhido pode não ser o melhor em todos os casos ⇒ é necessário adaptar o comportamento do middleware (dinamicamente).

Interceptors

Interceptam o fluxo de controle normal quando um objeto remoto for invocado.

Interceptors



Middleware Adaptativo

- Separação de interesses: tente separar as funcionalidades extras e depois costurá-las em uma única implementação ⇒ aplicabilidade restrita (*toy examples*)
- Reflexão computacional: deixe o programa inspecionar-se em tempo de execução e adaptar/mudar suas configurações dinamicamente, se necessário ⇒ ocorre principalmente no nível da linguagem, aplicabilidade não é muito clara.
- Projeto baseado em componentes: organize uma aplicação distribuída em componentes que podem ser substituídos dinamicamente quando necessário ⇒ causa muitas e complexas interdependências entre componentes.

Leituras Obrigatórias

Exemplos: NFS e Web