GBC074 – Sistemas Distribuídos

Revisão Redes

Arquiteturas

Baseado no material disponível pelo author em
https://www.distributed-systems.net

Redes de computadores

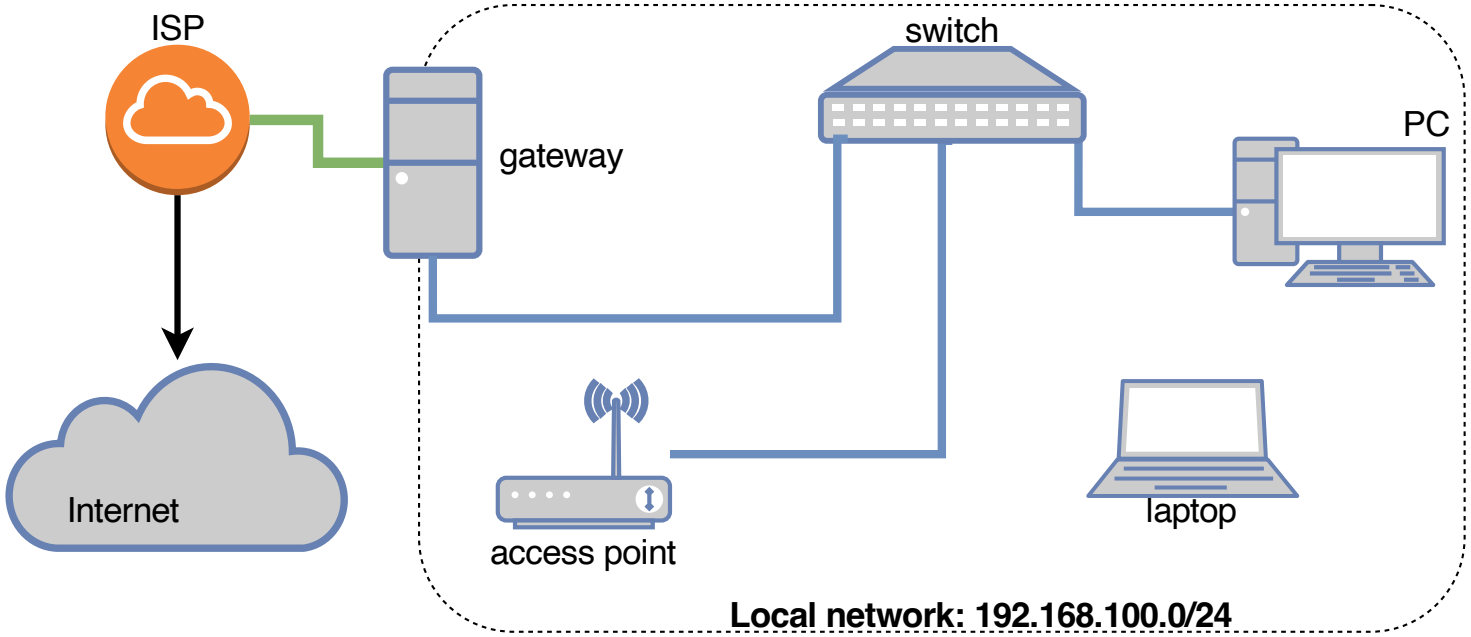
- Topologias

	Ponto-a-ponto	Barramento compartilhado	Token ring
Colisões	Sem	Com	Sem
Roteamento	Trivial	Complexo	Simples
# Conexões	Exponencial	Linear	Linear

- Ethernet (barramento compartilhado) é mais usado atualmente

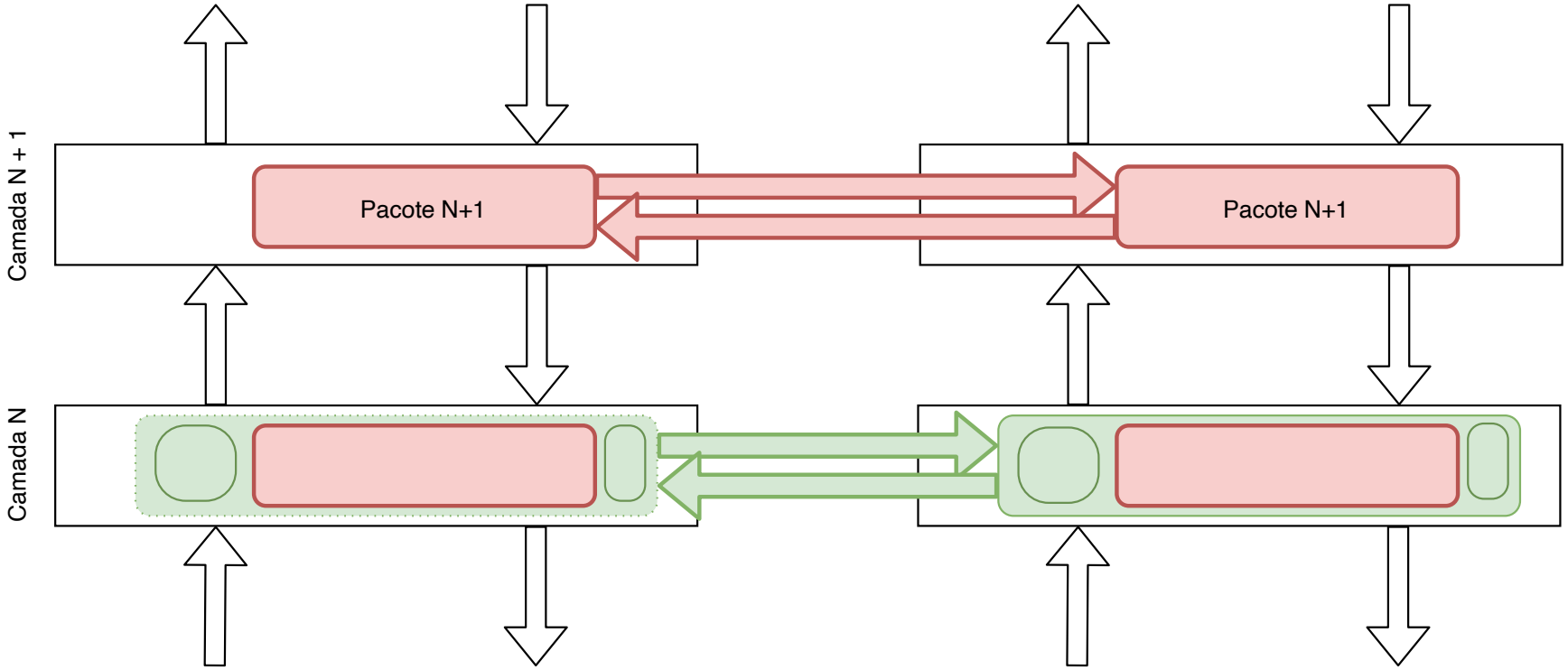
Redes de computadores

- Ethernet:
  - Controle de acesso ao meio
  - Transmissão de mensagens
  - Evitar e tratar colisões
- Exemplo de rede:



Redes de computadores

- Internet:
  - Implementada por meio de pilha de **protocolos**
  - Cada camada oferece **serviço** para camada acima por meio de **interfaces** bem definidas

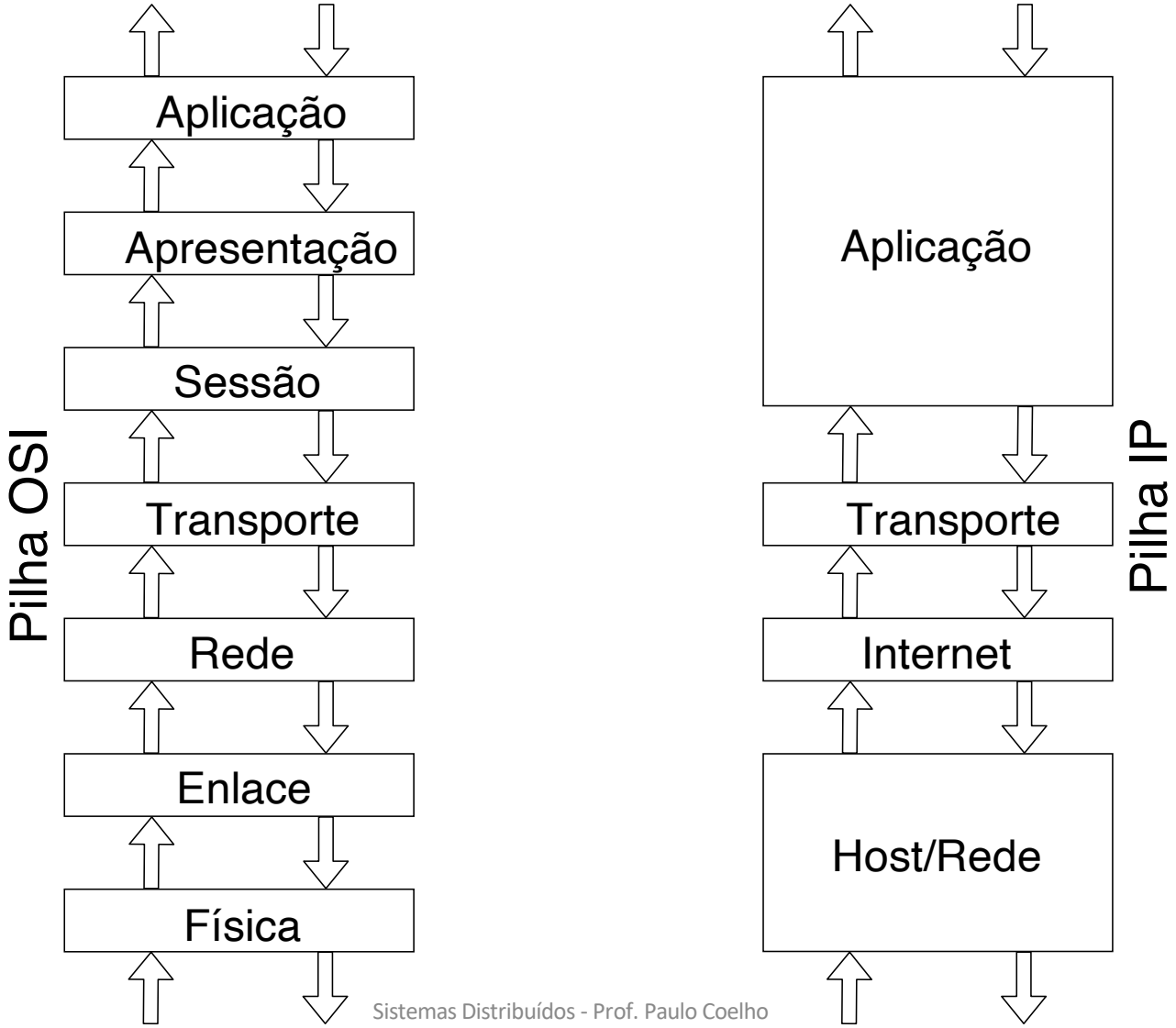


Redes de computadores

- Internet: modelo OSI vs. Arquitetura TCP/IP
  - Modelo OSI - 7 Camadas:
    - **Física**: Bits
    - **Enlace**: Frames/quadros; controle de fluxo; acesso ao meio.
    - **Rede**: Datagramas/pacotes; roteamento
    - **Transporte**: Controle de fluxo; fim a fim; confiabilidade; tcp e udp
    - **Sessão**: Streams/fluxos; conexões lógicas; restart; checkpoint; http, ssl
    - **Apresentação**: Objetos; json, xml; criptografia
    - **Aplicação**: Aplicações; http, pop, ftp
  - Arquitetura TCP/IP
    - Não apresenta as 7 camadas
    - Baseada em comutação de pacotes
    - Roteadores conectam redes diferentes
    - Utiliza “melhor esforço” para entrega da informação

Redes de computadores

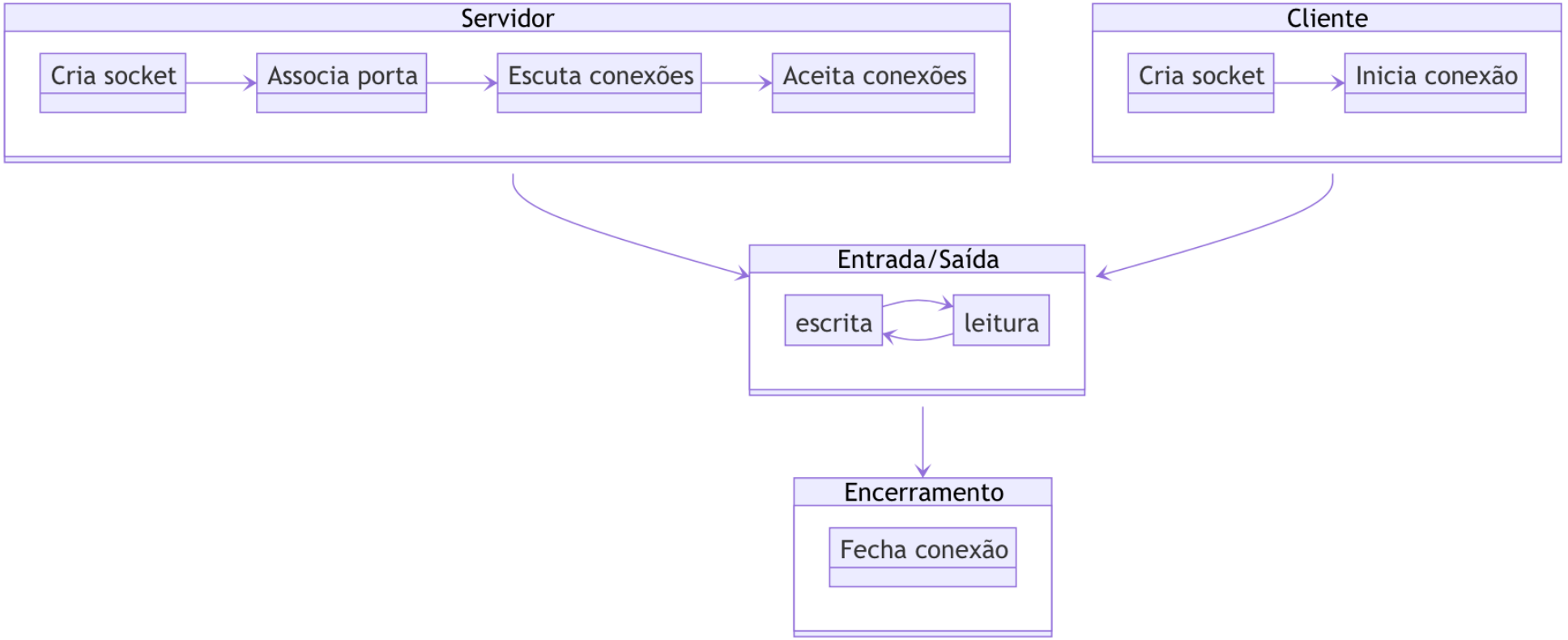
- Internet: modelo OSI vs. Arquitetura TCP/IP



Sockets

- Conexão direta entre duas *hosts*
- Definição do protocolo:
  - Camada 3: AF_INET ou PF_INET – Identifica unicamente o *host* a se conectar
  - Camada 4: SOCK_STREAM ou SOCK_DGRAM – Identifica o protocol de camada 4 a ser utilizado
- Porta: inteiro de 16 bits
  - Associadas a serviços pela Internet Assigned Numbers Authority, IANA.
    - Portas "Bem conhecidas": 0-1023
    - Portas Proprietárias: 49151
    - Portas Dinâmicas: 65535
- API define sequência de chamadas necessárias para partes iniciarem a “conversa”

Sockets – TCP


Servidor
Cria socket
Associa porta
Escuta conexões
Aceita conexões
Cliente
Cria socket
Inicia conexão
Entrada/Saída
escrita
leitura
Encerramento
Fecha conexão

Sockets – TCP

#server.py

```
#!/usr/bin/python                       # This is server.py file

import socket                           # Import socket module

s = socket.socket()                     # Create a socket object
host = socket.gethostname()             # Get local machine name
port = 12345                            # Reserve a port for your
service.
s.bind((host, port))                  # Bind to the port

s.listen(5)                             # Now wait for client
connections.
while True:
   c, addr = s.accept()                 # Accept client connection
   print('Got connection from', addr)
   c.send('Thank you'.encode())
   c.close()                            # Close the connection
```

Sockets – TCP

#client.py

```
#!/usr/bin/python                    # This is client.py

import socket                        # Import socket module

s = socket.socket()                  # Create a socket
host = socket.gethostname()          # Get local host name
port = 12345                         # Port for your service

s.connect((host, port))
data = s.recv(1024)                  # Receive data
print(data.decode())
s.close()                            # Close the socket
```

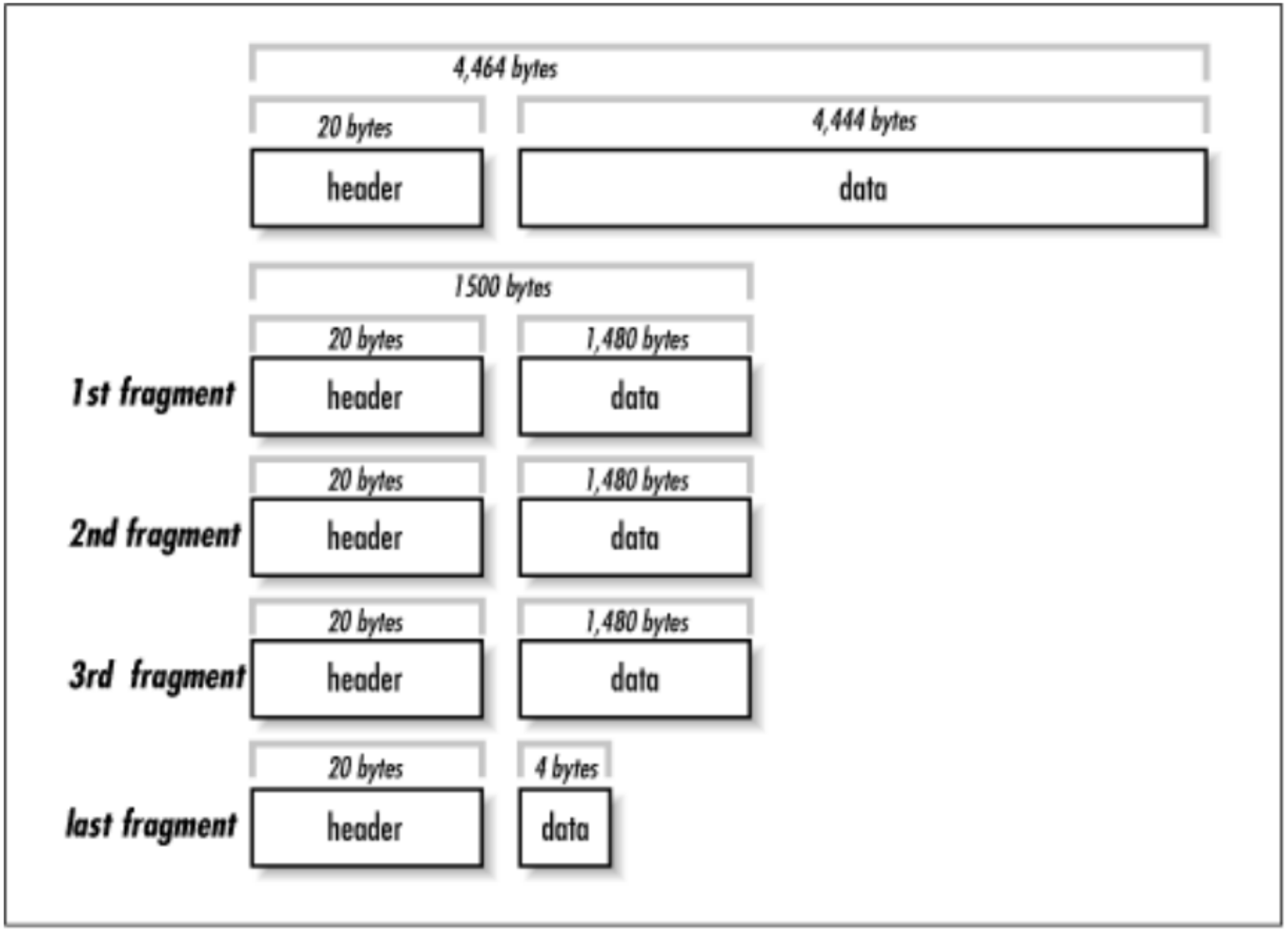
Sockets – UDP

- A criação do socket é feita explicitando-se o uso de **datagramas**:
  - `s =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)`
- Servidor UDP não executa listen ou accept
  - `data,addr = sock.recvfrom(4096)`
- Mesmo socket é usado para manter comunicação com múltiplos interlocutors:
  - `sent =sock.sendto(data, addr)`
- Outras características do UDP:
  - falta de ordem
  - falta de confiabilidade
  - menos dados lidos que enviados

Representação de Dados

- Fatores a serem considerados:
  - variações de definições de tipos, por exemplo, inteiro: 8: 16, 32, ou 64 bits?
  - variações na representação de dados complexos: classe x estrutura
  - conjunto de caracteres diferentes: ASCII x UTF
  - little endian, como x64 e IA-32, ou big endian como SPARC (< V9), Motorola e PowerPC? ou ainda, flexível como ARM, MIPS ou IA-64?
  - fim de linha com crlf (DOS) x lf (Unix)?
  - fragmentação de dados na rede

Representação de Dados



Representação de Dados

- Representação textual:
  - HTTP:

```
telnet www.google.com 80
Trying 187.72.192.217...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
host: www.google.com
                              <=== Linha vazia!
```

Representação de Dados

- Representação textual:
  - XML:

```
<person>
    <name>John Doe</name>
    <id>112234556</id>
    <email>jdoe@example.com</email>
    <telephones>
       <telephone type="mobile">123 321 123</telephone>
       <telephone type="home">321 123 321</telephone>
    </telephones>
</person>
```

Representação de Dados

- Representação textual:
  - JSON:

```
{
    "name": "John Doe",
    "id": 112234556,
    "email": "jdoe@example.com",
    "telephones": [
        { "type": "mobile", "number": "123 321 123"},
        { "type": "home", "number": "321 123 321"},
    ]
}
```

Representação de Dados

- Formatos binários:
  - Melhor uso do espaço
  - Usado em sistemas distribuídos na serialização
  - Exemplos:
    - Java serialization
    - Google Protocol Buffers
    - Thrift

Representação de Dados

- Formatos binários:
  - Protocol buffer:

```
message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;
    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }
    message PhoneNumber {
        required string number = 1;
        optional PhoneType type = 2 [default = HOME];
    }
    repeated PhoneNumber phone = 4;
}
```

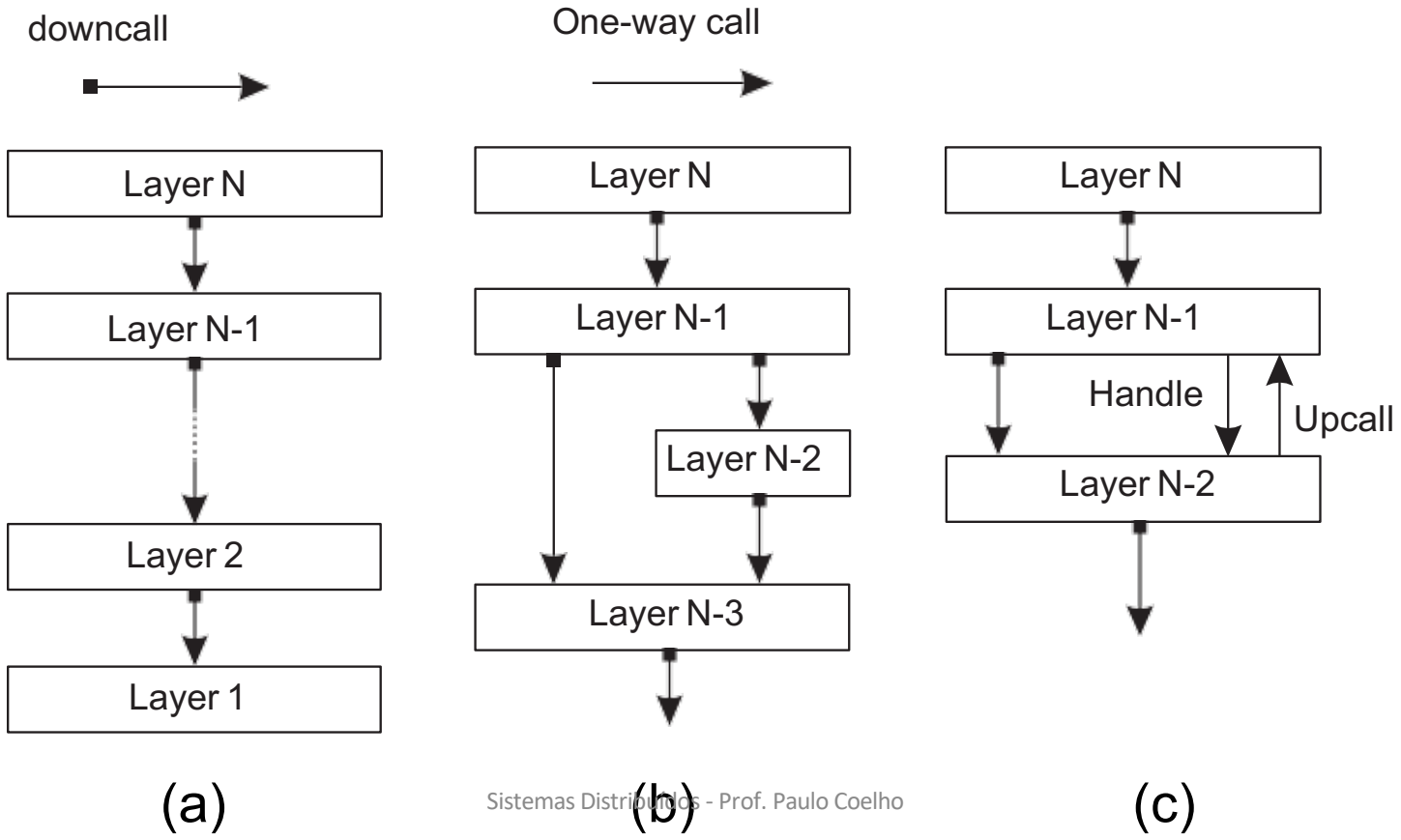
  - Observe a presença de campos de preenchimento opcional (**optional**), de enumerações (**enum**), e de coleções (**repeated**).

Estilos Arquiteturais

- Ideia básica:
  - Estilo definido em termos de
    - Componentes (substituíveis) com interfaces bem-definidas
    - Modo como componentes são interconectados
    - Dados trocados entre componentes
    - Como componentes e conectores são configurados para formar um sistema
- Conector
  - Mecanismo que reconcilia comunicação, coordenação ou cooperação entre componentes
  - Ex. Facilidades (remote) procedure call, troca de mensagens ou streaming

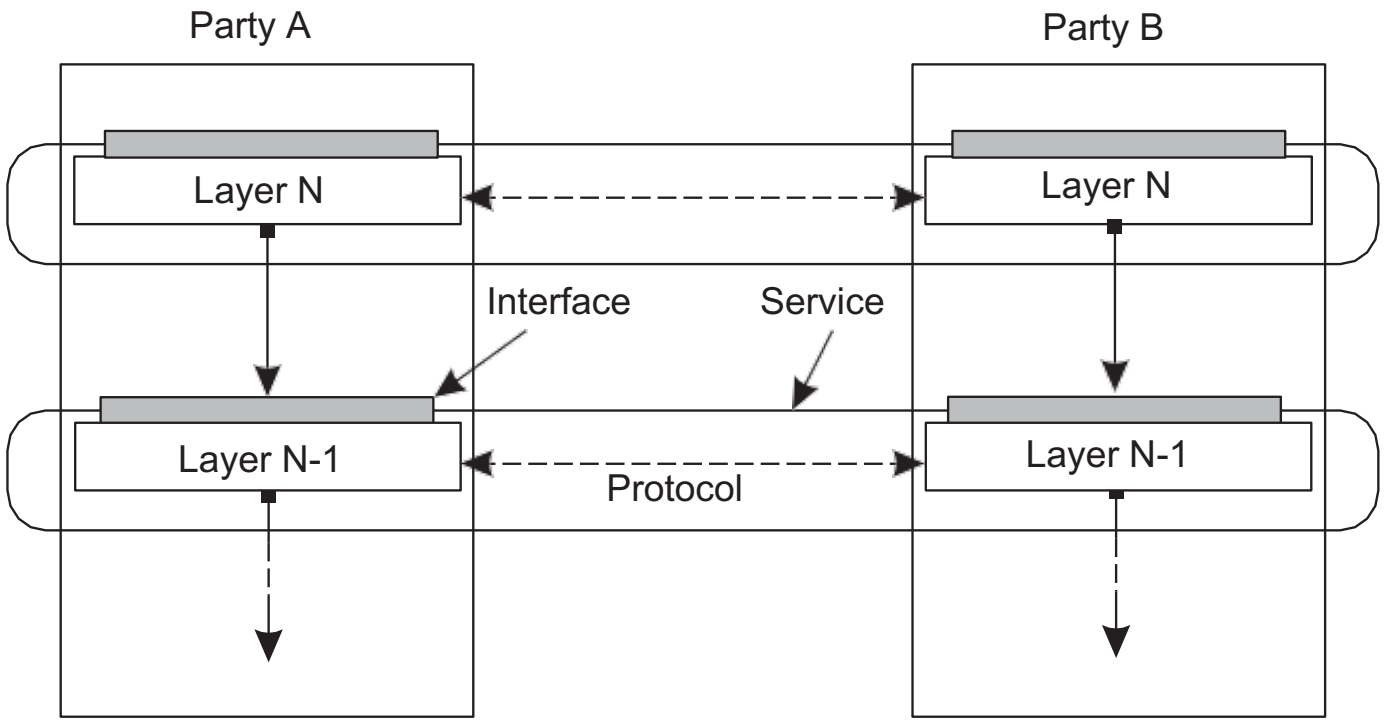
Arquitetura em camadas

- Diferentes modelos de organização
  - (a) Em camadas (tradicional)
  - (b) Organização mista de camadas
  - (c) Camadas com *upcall*



Exemplo: protocolos de comunicação

- Protocolo, serviço e interface



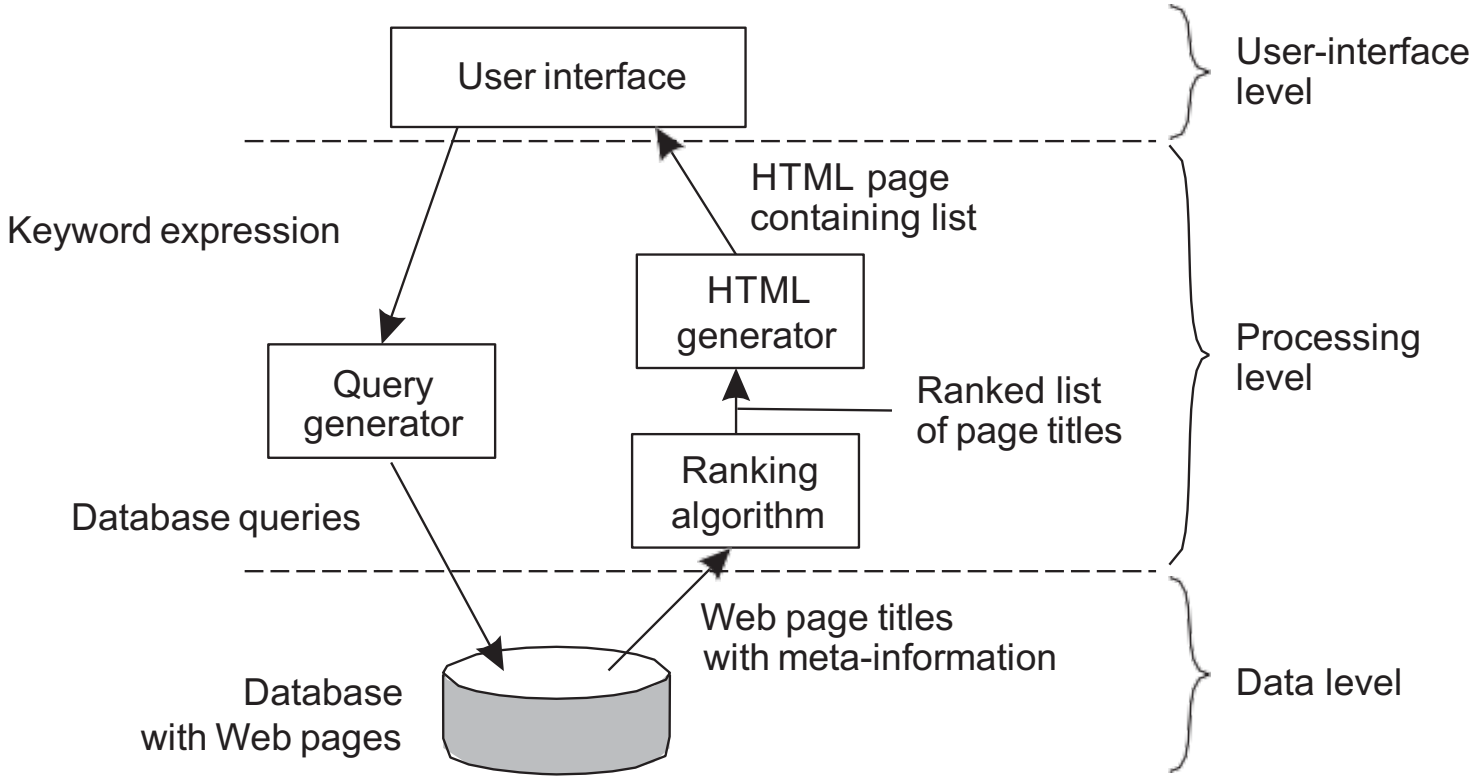
Exemplo: Camadas na Aplicação

- Visão tradicional em 3 **camadas**:
  - **Aplicação**: camada de interfaceamento com usuários ou aplicações externas
  - **Processamento**: contém as funções de uma aplicação (a lógica), sem os dados específicos
  - **Dados**: contém os dados que o cliente deseja manipular por meio dos componentes da aplicação

- Organização utilizada em diversos sistemas de informação distribuídos, usando bases de dados tradicionais e aplicações relacionadas

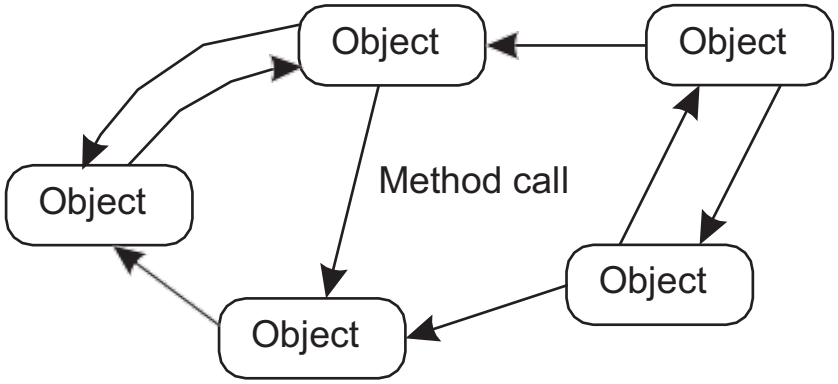
Exemplo: Camadas na Aplicação

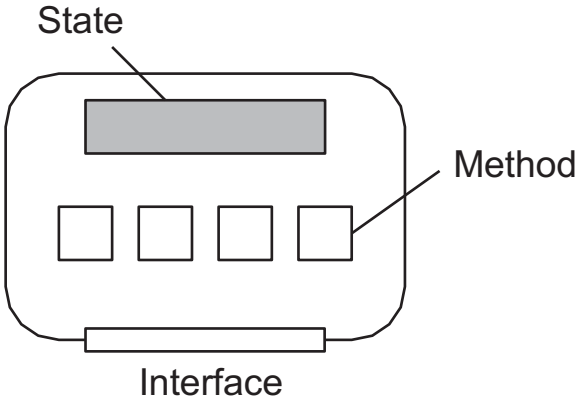
- Exemplo: sistema de busca simples



Arquitetura baseada em objetos

- Essência:
  - Componentes são **objetos**, conectados através de chamadas de procedimentos (métodos)
  - Objetos podem estar em máquinas diferentes, com chamadas executadas remotamente

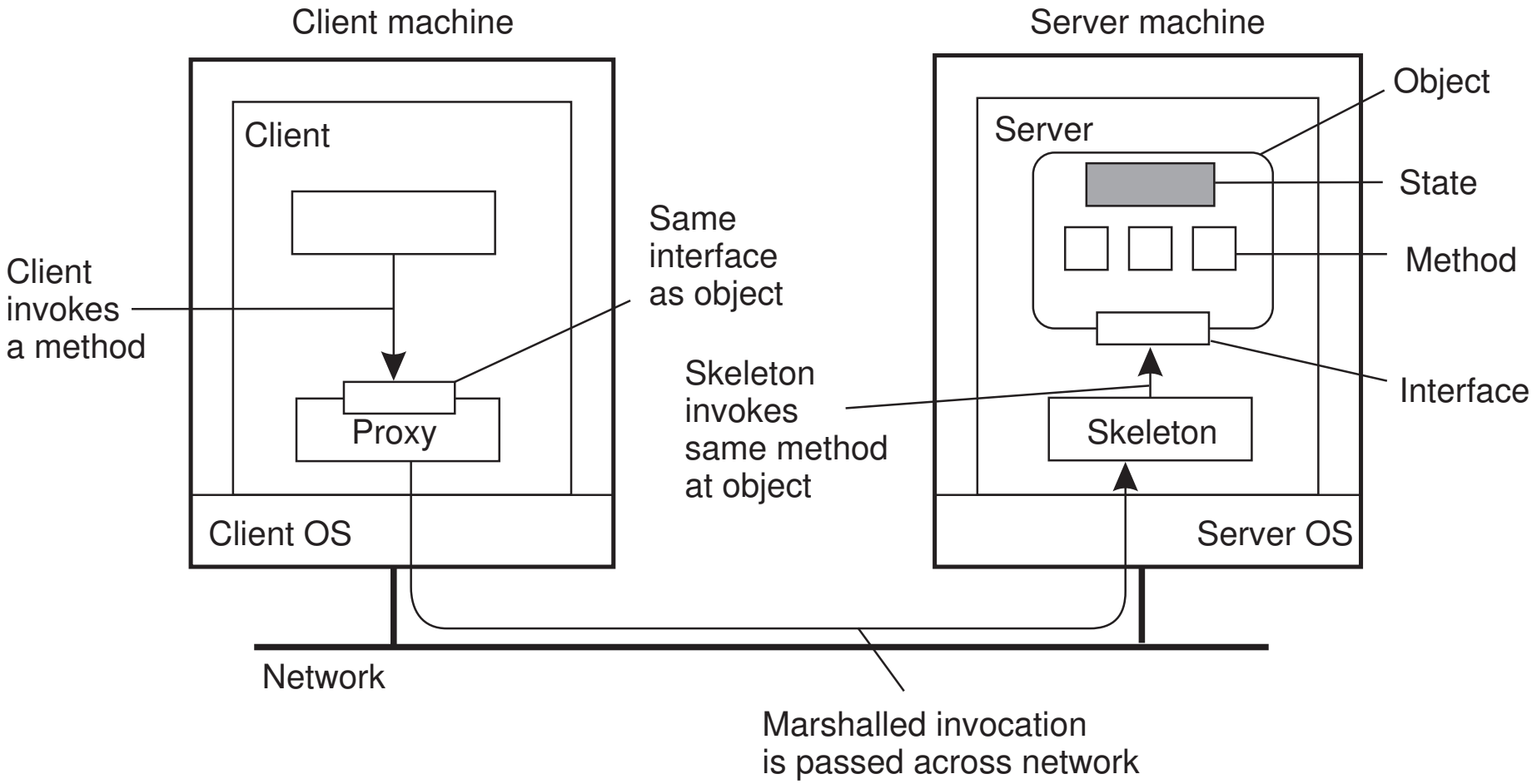




- Encapsulamento:
  - Objetos encapsulam os dados
  - Oferecem métodos sobre os dados, sem revelar implementação interna

Arquitetura baseada em objetos

- Invocação remota: *Stubs* e *skeletons*



Arquitetura baseada em objetos

- Invocação remota – pontos a considerar:
  - Transparência
  - Descoberta
  - Execução síncrona e assíncrona
  - Tratamento de falhas
  - Concorrência no acesso a objeto remoto
  - Reexecuções

Arquiteturas RESTful

- Essência:
  - Sistema distribuído como uma coleção de **recursos**
  - Recursos gerenciados individualmente por componentes
  - Recursos podem ser adicionados, removidos, recuperados e modificados por aplicações (remotas)
- Recursos identificados por mecanismo único de nomes
- Serviços com mesma interface
- Mensagens de/para o serviço são auto descritivas
- Após execução, componente "esquece" tudo a respeito do cliente
- Operações básicas:

.

Operação	Descrição
PUT	Cria novo recurso
GET	Recupera estado do recurso em alguma representação
DELETE	Remove um recurso
POST	Modifica recurso pela transferência de novo estado

Exemplo: Amazon's Simple Storage Service (S3)

- Essência:
  - Objetos (arquivos) armazenados em *buckets* (diretórios)
  - *Buckets* não podem ser colocados dentro de *buckets*
  - Operações em objeto *obj* no *bucket* bkt exigem o seguinte identificador:
    - http://bkt.s3.amazonaws.com/obj

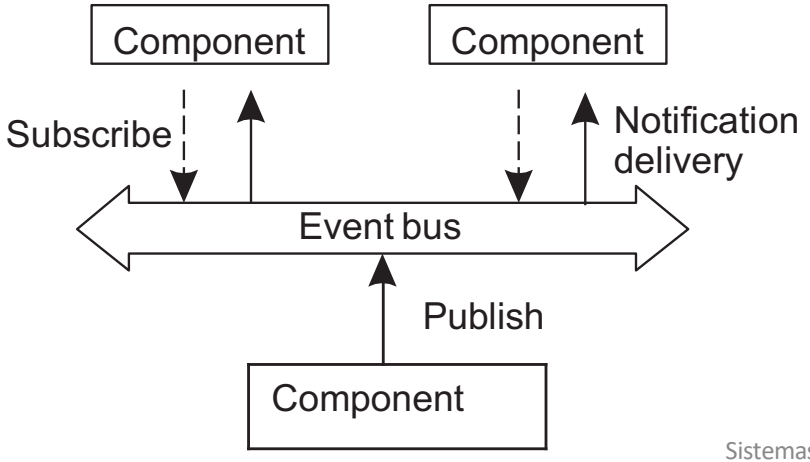
- Operações típicas: executadas pelo envio de requisições **HTTP**:
  - Criar *bucket* ou objeto: **PUT**, junto com a URI (*Uniform Resource Identifier*)
  - Listar objetos: **GET** sobre o nome do *bucket*
  - Ler um objeto: **GET** sobre a URI completa

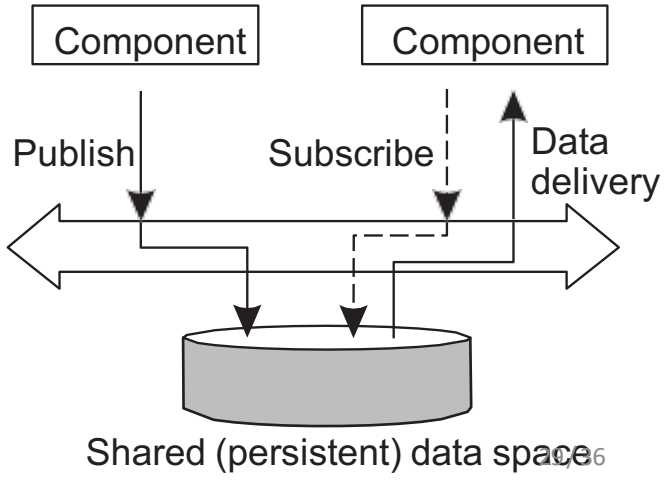
Arquiteturas baseadas em eventos (publish-subscribe)

- Coordenação
  - Acoplamento temporal e referencial
  - Baseado em eventos
  - Espaço de dados compartilhado

  - Relação entre comunicação e acoplamento:

	Acoplado temporalmente	**Desacoplado temporalmente**
Acoplado referencialmente	Direta	Mailbox
Descoplado referencialmente	Baseado em eventos	Espaço de dados compartilhado



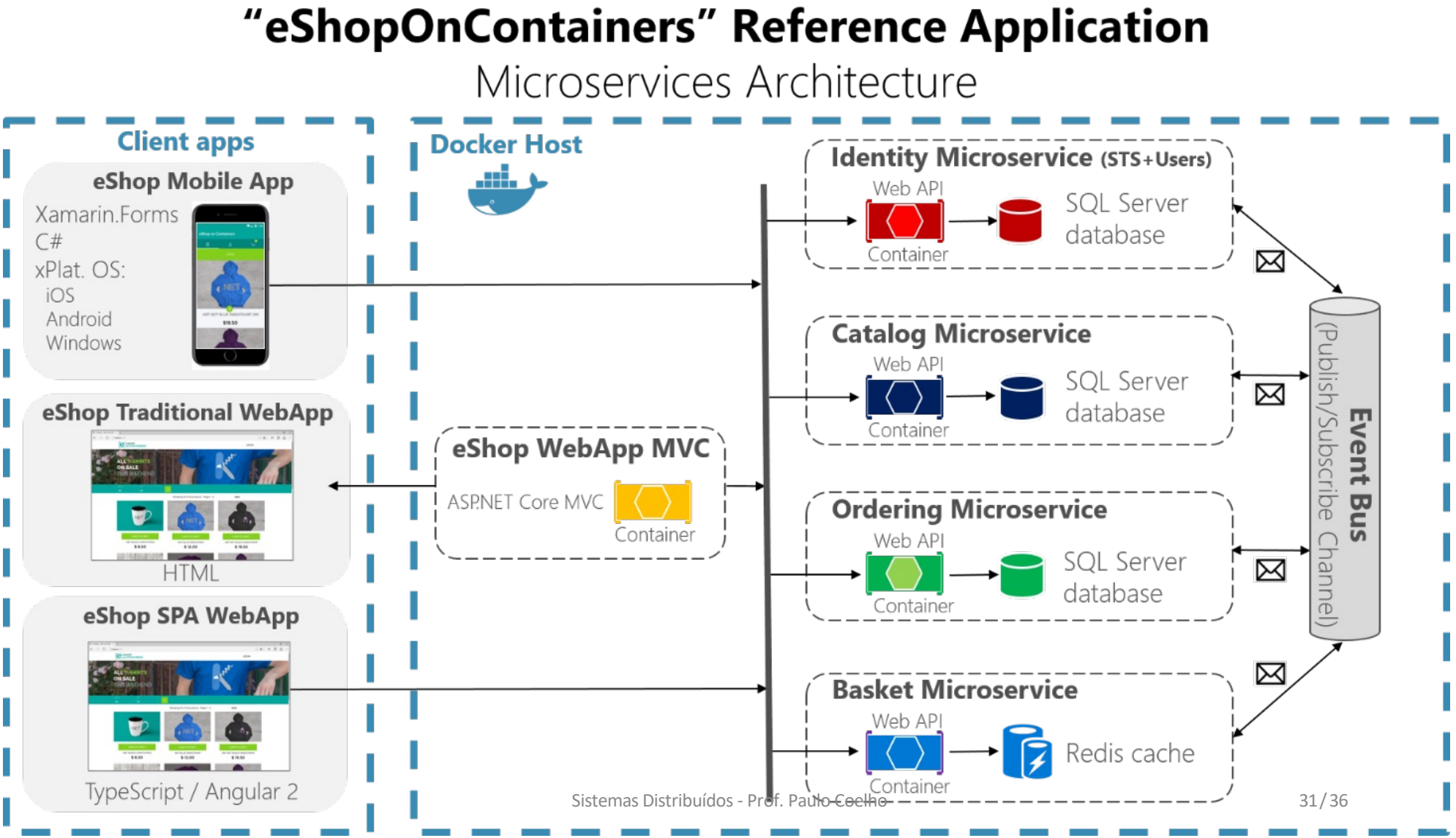


Arquiteturas baseadas micro serviços

- A moda da vez é a chamada arquitetura de **micro serviços**
- Objetivo é obter componentes mais simples para cada tarefa
- Componentes podem ser replicados, escalonados, desenvolvidos e mantidos independentemente
- Cada tarefa:
  - Diversos componentes
  - Organizados em camadas resolvendo um problema específico
- Tarefas contribuem para realização de uma tarefa maior comum.

Arquiteturas baseadas micro serviços

- Exemplo:



Organização do *Middleware*

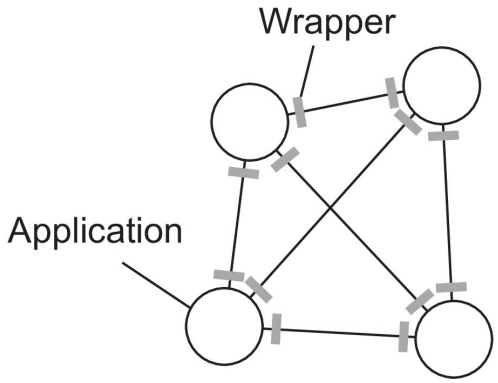
- Motivação
  - *Middleware* provê algum nível de **transparência** de distribuição
  - Camada independente de um sistema distribuído ou aplicação
  - 2 importantes padrões (*design patterns*):
    - *Wrappers* (“encapsuladores”)
    - *Interceptors* (“interceptadores”)

Organização do *Middleware*

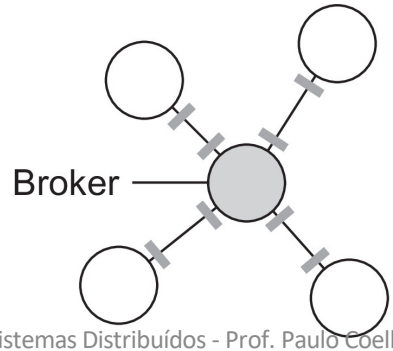
- *Wrappers* (ou adaptadores)
  - Interfaces oferecidas por aplicações legadas inadequadas para todas as aplicações
  - Bem mais do transformar interfaces:
    - Ex 1: Adaptadores de objeto: necessidade de invocar objetos remotos, mas a aplicação original é uma biblioteca atuando em tabelas de uma base de dados relacional
    - Ex2: *Amazon* S3: servidor web é um adaptador para o serviço real

Organização do *Middleware*

- *Wrappers*
  - Sistema com diversos componentes em colaboração
  - O(N^2) *wrappers* para N componentes

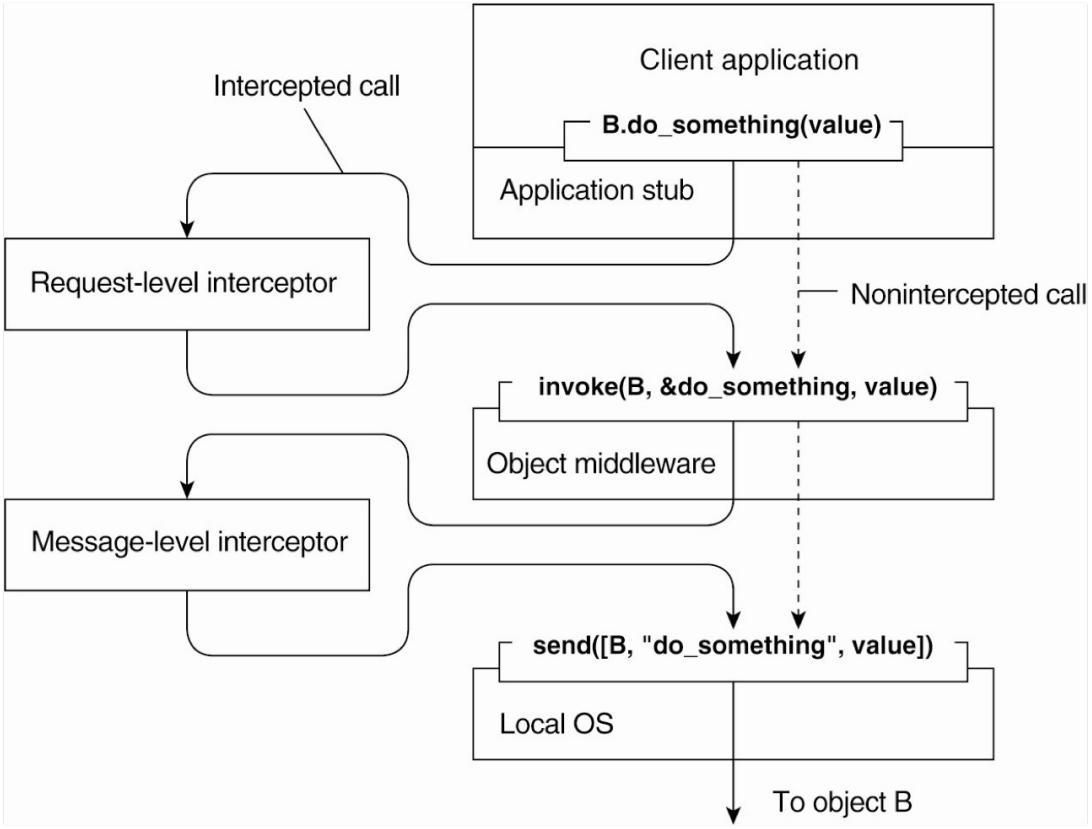


  - Alternativa: Brokers
    - Componente centralizado que manipula acesso entre componentes: O(N)



Organização do *Middleware*

- *Interceptors*
  - Quebra o fluxo de controle
  - Permite execução outro código (*application specific*)
  - Melhora gerência do software:
    - Transparência
    - Instrumentação
  - Ex:
    - Invocação de objeto remoto

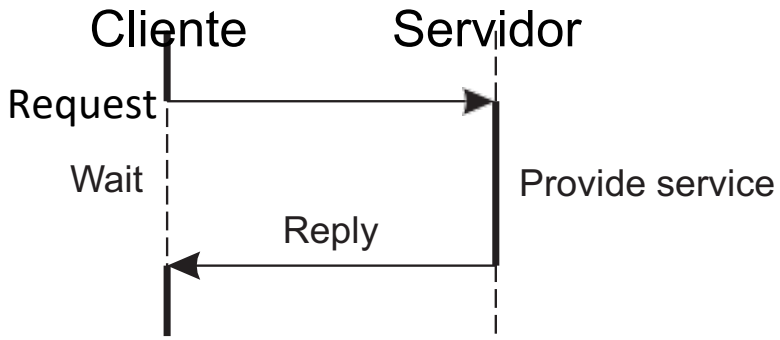


Organização do *Middleware*

- Software adaptativo - desafios
  - *Wrappers* e *interceptors*: adaptam-se a mudanças na mobilidade, QoS na rede, falhas, descarregamento de bateria, etc
  - Modificar o sistema on-the-fly
  - Abordagem possível: design baseado em componentes
    - Modificável via composição
    - Ex: módulos de um sistema operacional

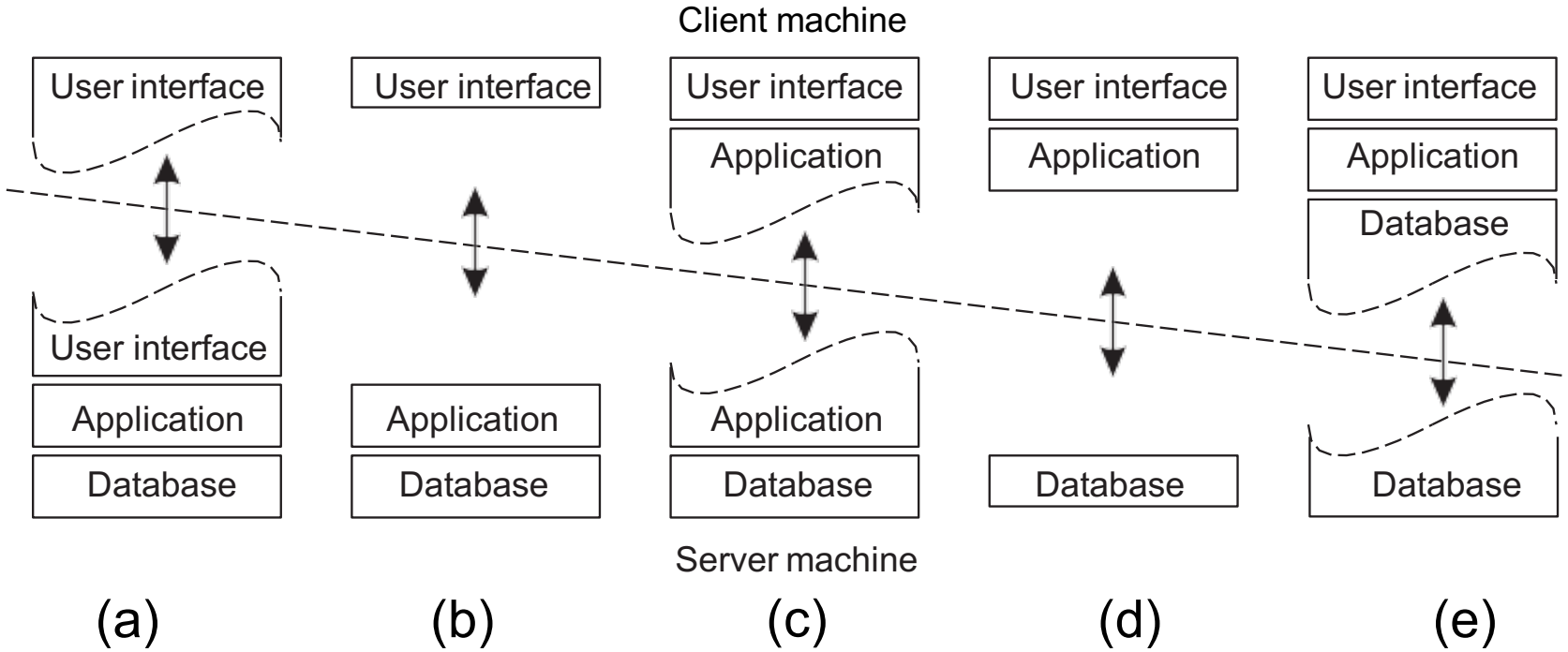
Arquitetura de Sistema Centralizada

- Modelo servidor:
  - Servidores: processos que oferecem serviços
  - Clientes: processos que utilizam serviços
- Clientes e servidores podem estar em máquinas distintas
- Modelo requisição/resposta (request/reply)



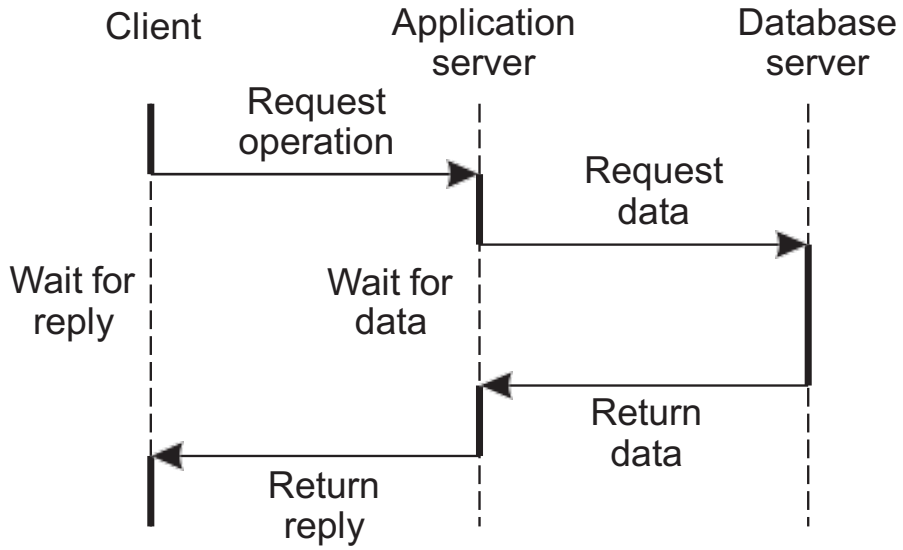
Arquiteturas centralizadas multi-camadas (multi-tiered)

- Configurações:
  - **Single-tiered**: mainframe e terminal "burro" (*dumb terminal*)
  - **Two-tiered**: configuração cliente/servidor (figura)
  - **Three-tiered**: cada camada em uma máquina diferente



Exemplo: arquitetura de 3 camadas

- *Application server:*
  - cliente e servidor ao mesmo tempo



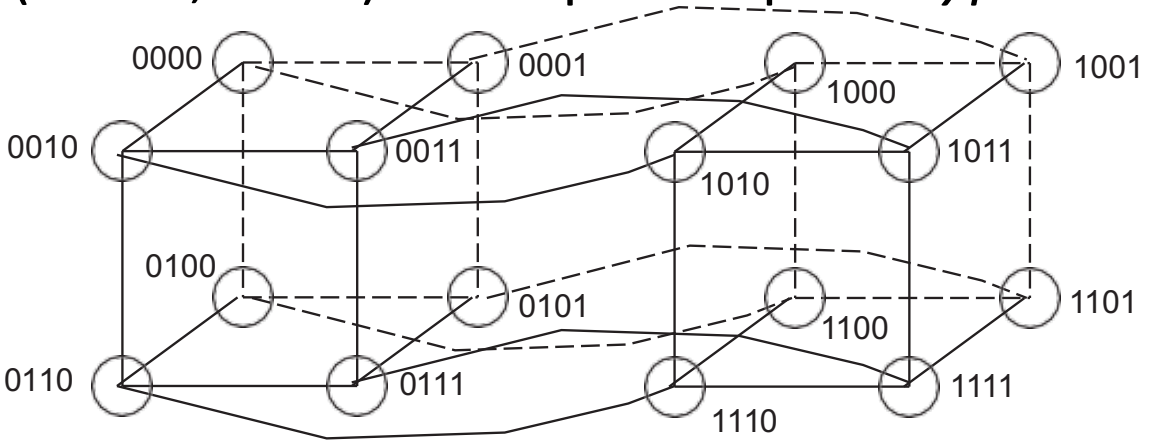
Organizações alternativas

- Distribuição vertical
  - Aplicação dividida em camadas lógicas e distribuídas em diferentes servidores
- Distribuição horizontal
  - Cliente ou servidor separados fisicamente em partes equivalentes logicamente
  - Cada parte opera em sua fatia do conjunto completo de dados
- Arquiteturas Peer-to-peer (P2P)
  - Processos são todos “iguais”:
    - Tarefas são executadas por todos os processos
    - Sem distinção entre cliente e servidor

P2P estruturado

- Indexação de informação
  - Cada item é associado a uma chave.
  - Chave é o índice para efeitos de localização
  - Geralmente utiliza-se uma função hash

    key(data item) = hash(data item's value).
  - Sistema P2P: responsável pelo armazenamento dos pares (chave, valor). Exemplo simples: *hypercube*



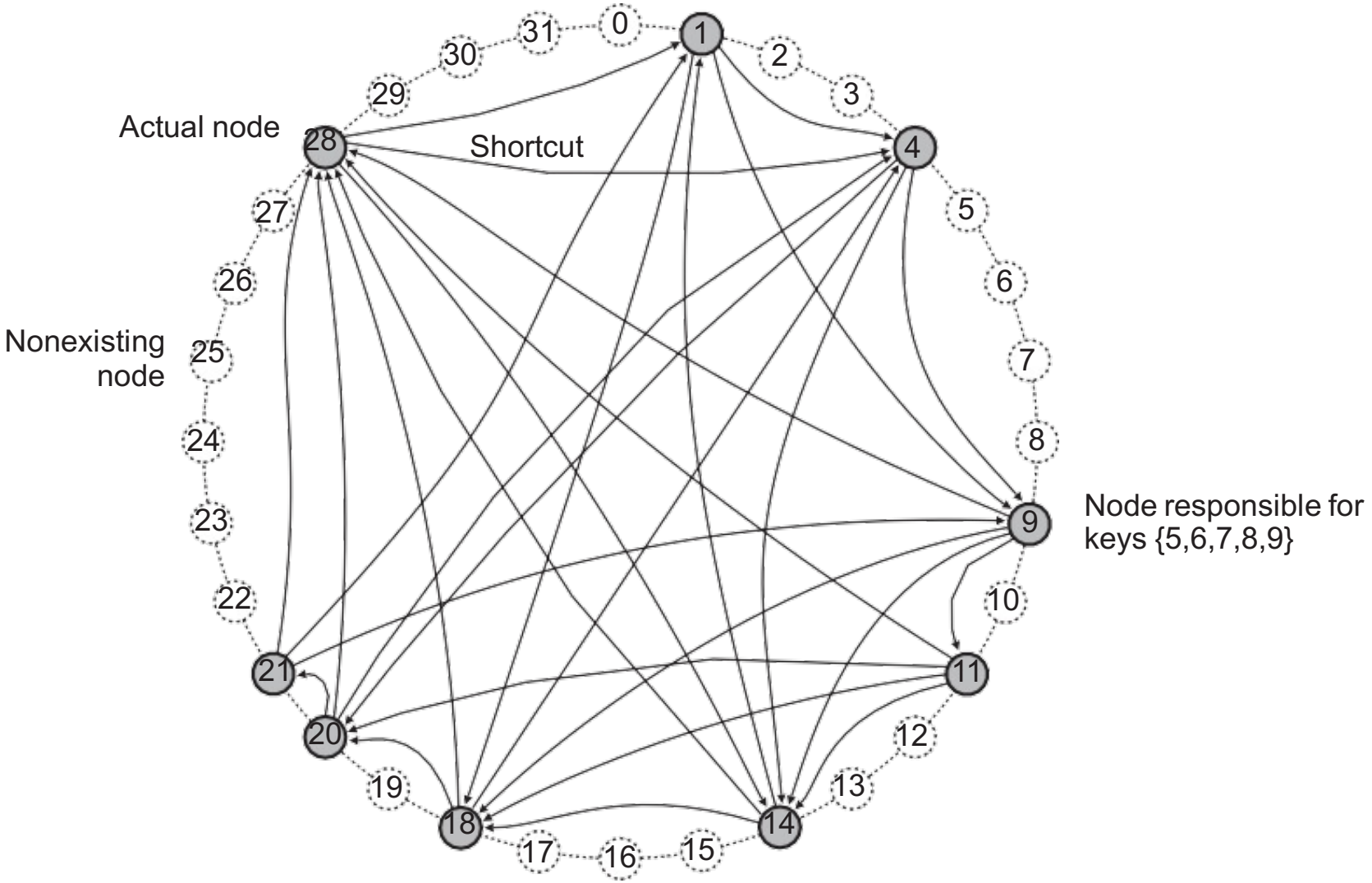
Pesquisar item *d* com chave $k \in \{0, 1, 2, \ldots, 2^4 - 1\}$ significa rotear requisação para nó com identificador *k*.

P2P estruturado: Chord

- Nós estruturados em um anel lógico
- Cada nó possui identificador (**id**) de **m** bits
- *Hash* de item gera chave **k** de tamanho **m**
- Item de dados com chave **k** é armazenado no nó com menor identificador **id** ≥ **k**
  - Nó é denominado sucessor da chave **k**
- Anel é estendido com links de atalho para outros nós

P2P estruturado: Chord

- Exemplo: buscar chave 3 a partir do nó 9



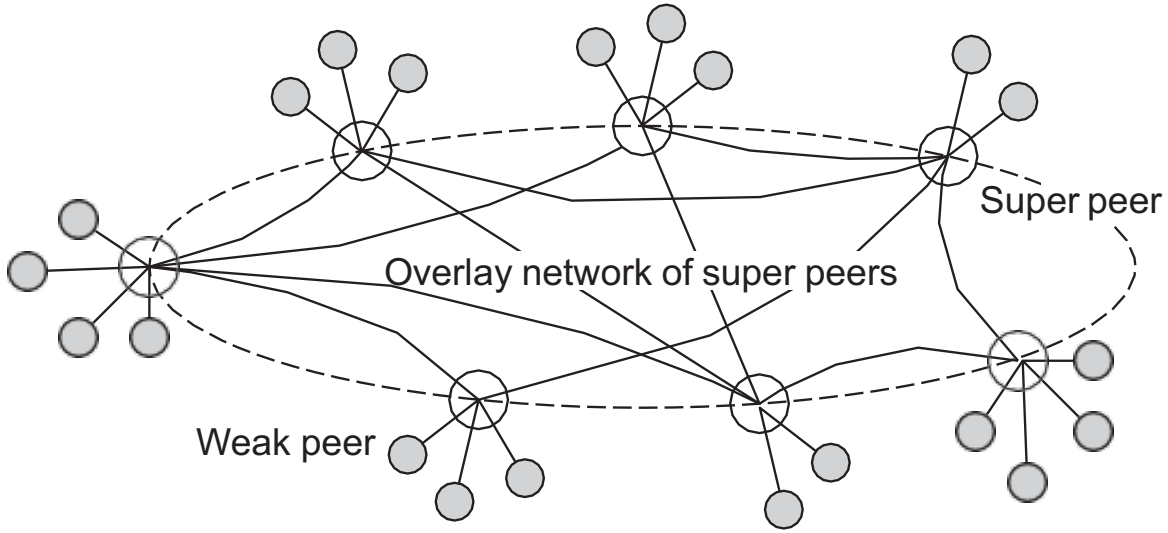
lookup(3)@9: 28 → 1 → 4

P2P não-estruturado

- Cada nó mantém lista *ad hoc* de vizinhos
- Rede overlay resultante semelhante a um grafo randômico:
  - borda (u, v) existe com probabilidade P[(u, v)]
- Busca
  - **Inundação** (*Flooding*):
    - Nó requisitante **u** envia requisição para **d** a cada vizinho **v**
    - Requisição recebida múltiplas vezes é ignorada
    - Nó **v** busca localmente por **d** (recursivamente)
    - Pode ser limitado por um *Time-To-Live*
  - **Caminhada aleatória** (Random Walk):
    - Nó requisitante **u** envia requisição para **d** a um vizinho **v** escolhido aleatoriamente
    - Se **v** não tem **d**, encaminha requisição ao outro vizinho **v'** escolhido aleatoriamente; e assim por diante.

Redes de super peers

- Necessidade de quebrar a simetria em redes P2P:
  - Ao pesquisar em uma rede P2P não-estruturada, a inclusão de servidores de índices melhora a performance
  - Decisão sobre armazenamento de dados pode ser mais eficiente se gerenciada por brokers



Redes de super peers: o caso do Skype

- Considere que o nó **A** deseja realizar chamada com nó **B**:
  - Situação 1: **A** e **B** na internet
    - Conexão TCP entre **A** e **B** para pacotes de controle
    - Chamada acontece com envio de pacotes UDP entre as portas negociadas
  - Situação 2: **A** opera atrás de um firewall, **B** está na internet
    - **A** estabelece conexão TCP com super peer **S** para pacotes de controle
    - **S** estabelece conexão TCP com **B** para encaminhar pacotes de controle
    - Chamada acontece diretamente entre **A** e **B** com envio de pacotes UDP
  - Situação 3: **A** e **B** operam atrás de um firewall
    - **A** estabelece conexão TCP com super peer **S**
    - **S** estabelece conexão TCP com **B**
    - Pacotes de controle encaminhados sobre as 2 conexões TCP
    - Chamada acontece por meio de um outro super peer **R**, que atua como relay
    - **A** e **B** estabelecem conexão com **R**

Arquitetura Edge-server

- Sistemas em que servidores são posicionados na **borda** da rede:
  - Fronteiras entre redes da empresa e a internet