Capítulo II – Modelos de Programação Distribuída (parte 2)

From: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
Edition 3, © Addison-Wesley

From: Cardoso, Jorge, "Programação de Sistemas Distribuídos em Java", FCA.

Paula Prata,
Departamento de Informática da UBI
http://www.di.ubi.pt/~pprata

1 – Modelos de comunicação por mensagens

2 – Exemplo: Comunicação por mensagens através de Sockets (em Java)

Modelos Arquitecturais

Cliente / Servidor

Múltiplos Servidores

Proxies

Peer processes

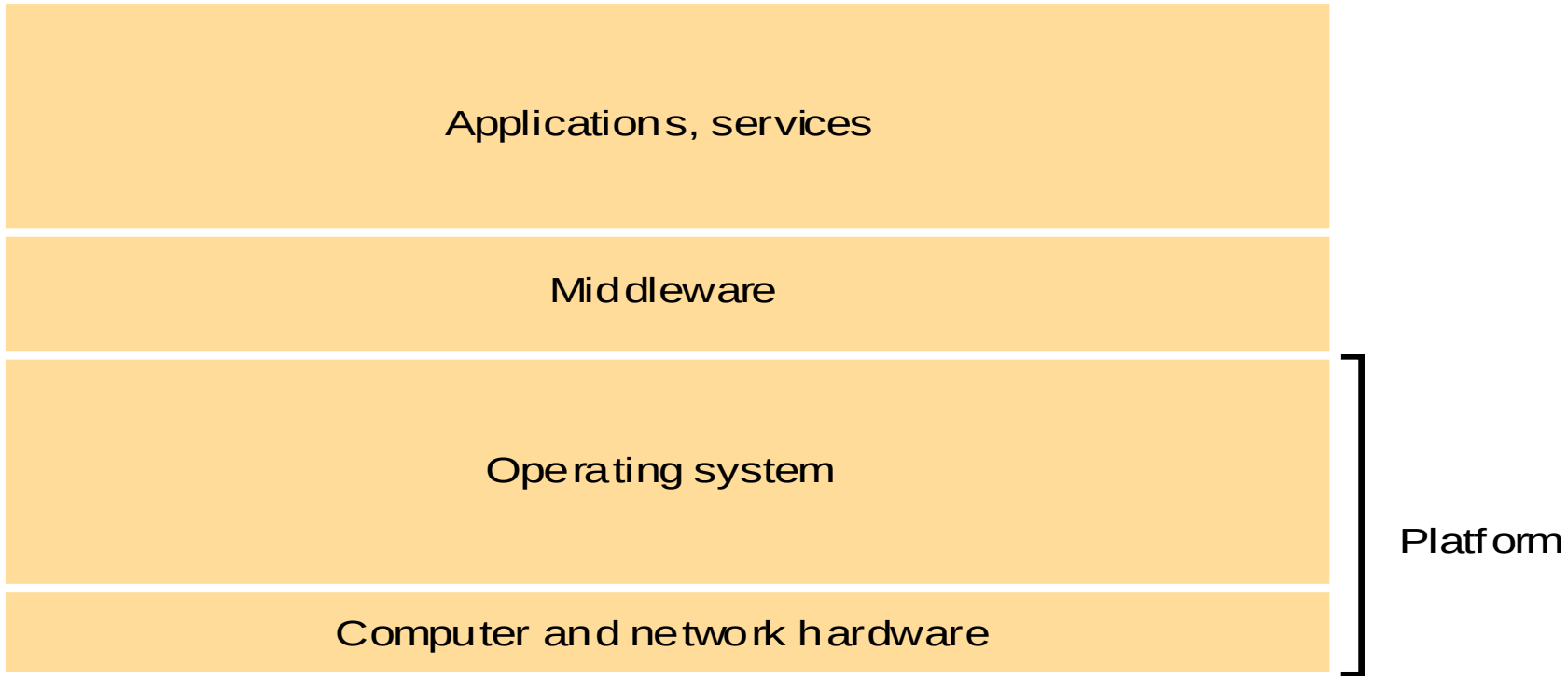
3 - Modelos arquiteturais

Um modelo arquitetural (ou a arquitetura) de um sistema distribuído é a estrutura do sistema em termos de localização das suas diferentes partes, do papel que cada parte desempenha e como se inter-relacionam.

A arquitetura tem implicações no desempenho, fiabilidade e segurança do sistema

3 - Modelos arquitecturais

Camadas de um sistema distribuído:



3 - Modelos arquitecturais

Ex.los de plataformas:

Intel I7/Windows

Intel x86/Linux

Power PC/Solaris

...

Middleware:

Camada de software que tem o objectivo de mascarar a heterogeneidade de um sistema distribuído fornecendo um modelo de programação uniforme.

Ex.los

Sun RPC

Java RMI

Corba

Microsoft .NET

...

3 - Modelos arquitecturais

Modelo (arquitetura) Cliente – Servidor

(Modelo independente do middleware utilizado)

Modelo assimétrico

Servidor *(back-end)* **:**
processo passivo que quando contactado por um cliente envia a resposta.

Cliente *(front-end)***:**
contacta o servidor com o objetivo de utilizar um serviço; envia um pedido (request/invocation) e fica à espera da resposta (reply/result).

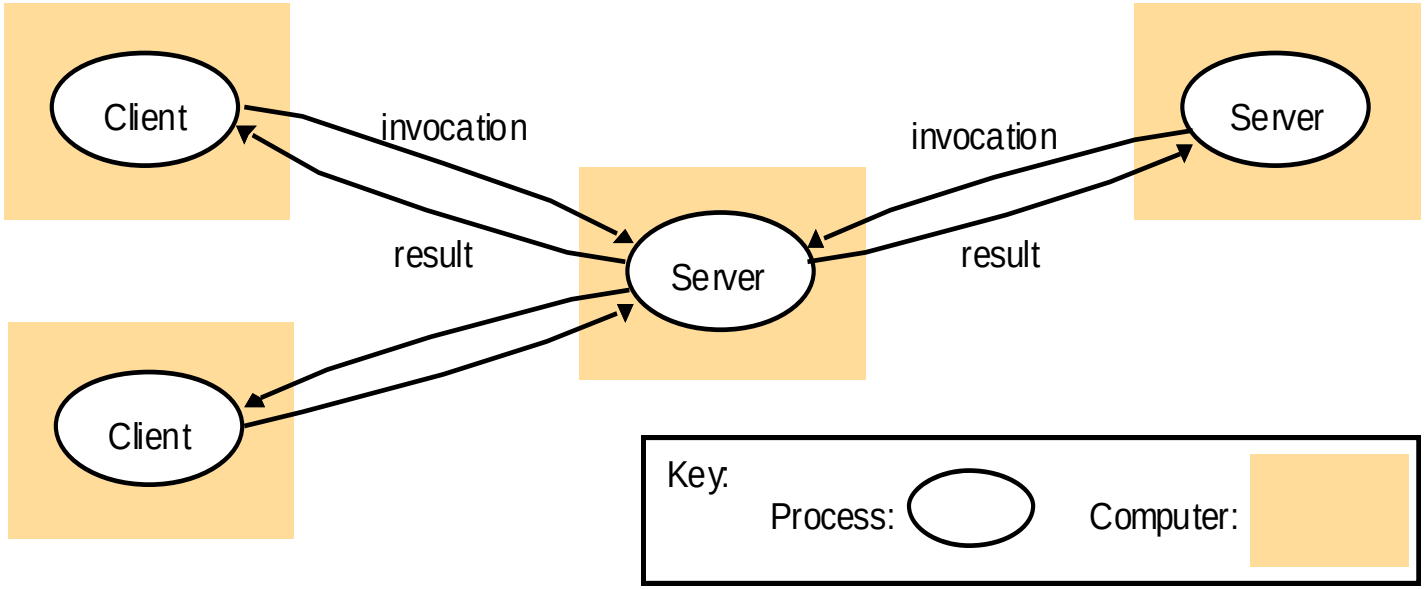
Cliente e Servidor são papéis que podem ser desempenhados.

Uma entidade pode simultaneamente ser cliente e servidor. Um processo para responder a um pedido, pode ter que recorrer a outro serviço, sendo cliente deste.

Ex.lo: um motor de pesquisa que usa "web crawlers" para pesquisar servidores web
...

3 - Modelos arquitecturais

Modelo Cliente – Servidor



- A máquina que suporta o processo servidor precisa de ter recursos mais poderosos de forma a suportar centenas de pedidos num curto intervalo de tempo.

3 - Modelos arquitecturais

Modelo Cliente – Servidor

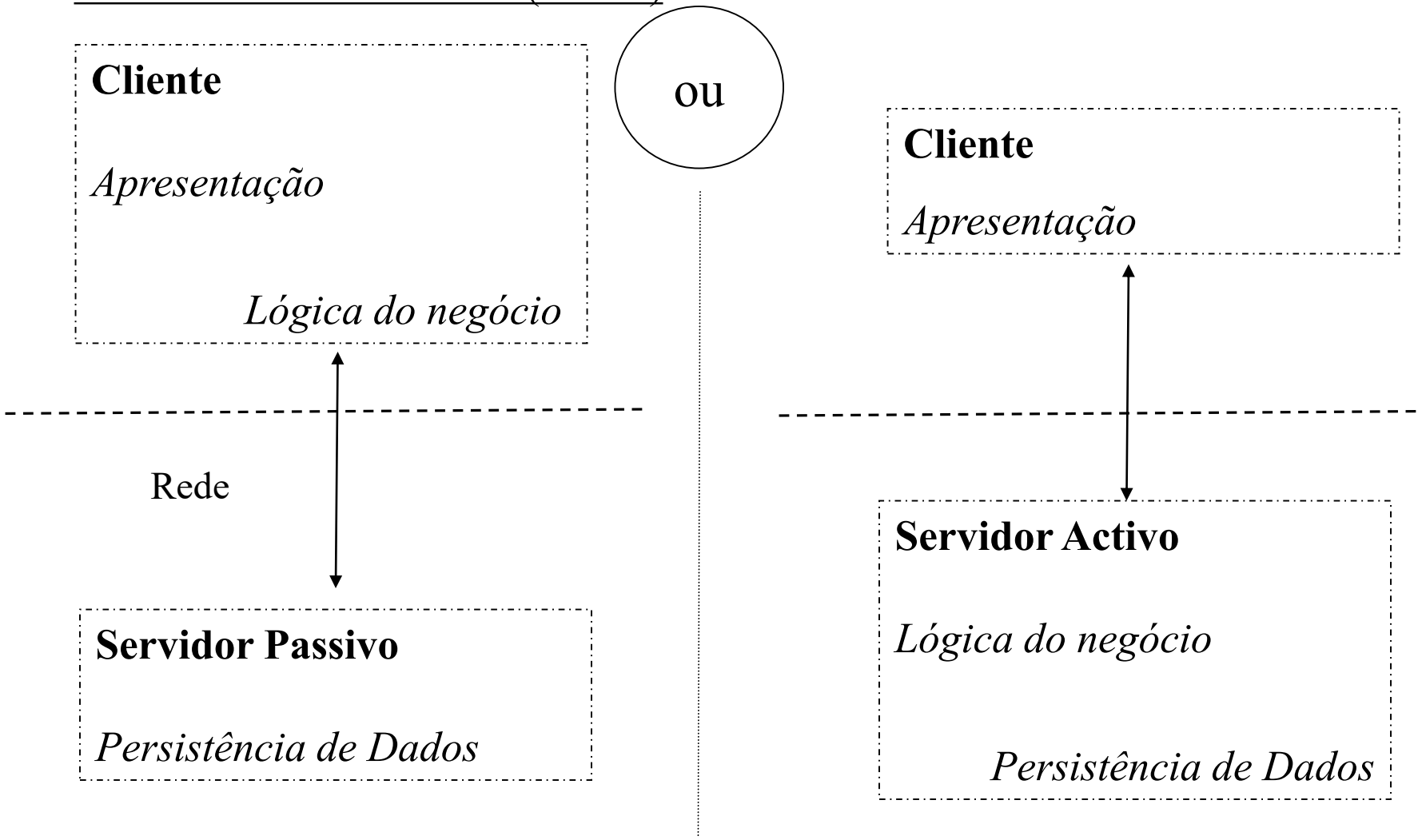
Num sistema de informação típico, existem três classes de funcionalidades:

- Camada de apresentação
  - Parte da aplicação responsável pela interface com o utilizador
- Camada de lógica de negócio
  - Regras de negócio que controlam o comportamento da aplicação
- Camada de persistência de dados
  - Parte que assegura o armazenamento e integridade dos dados

A primeira arquitetura cliente/servidor a ser desenvolvida foi a arquitetura de duas camadas (2-tiers)

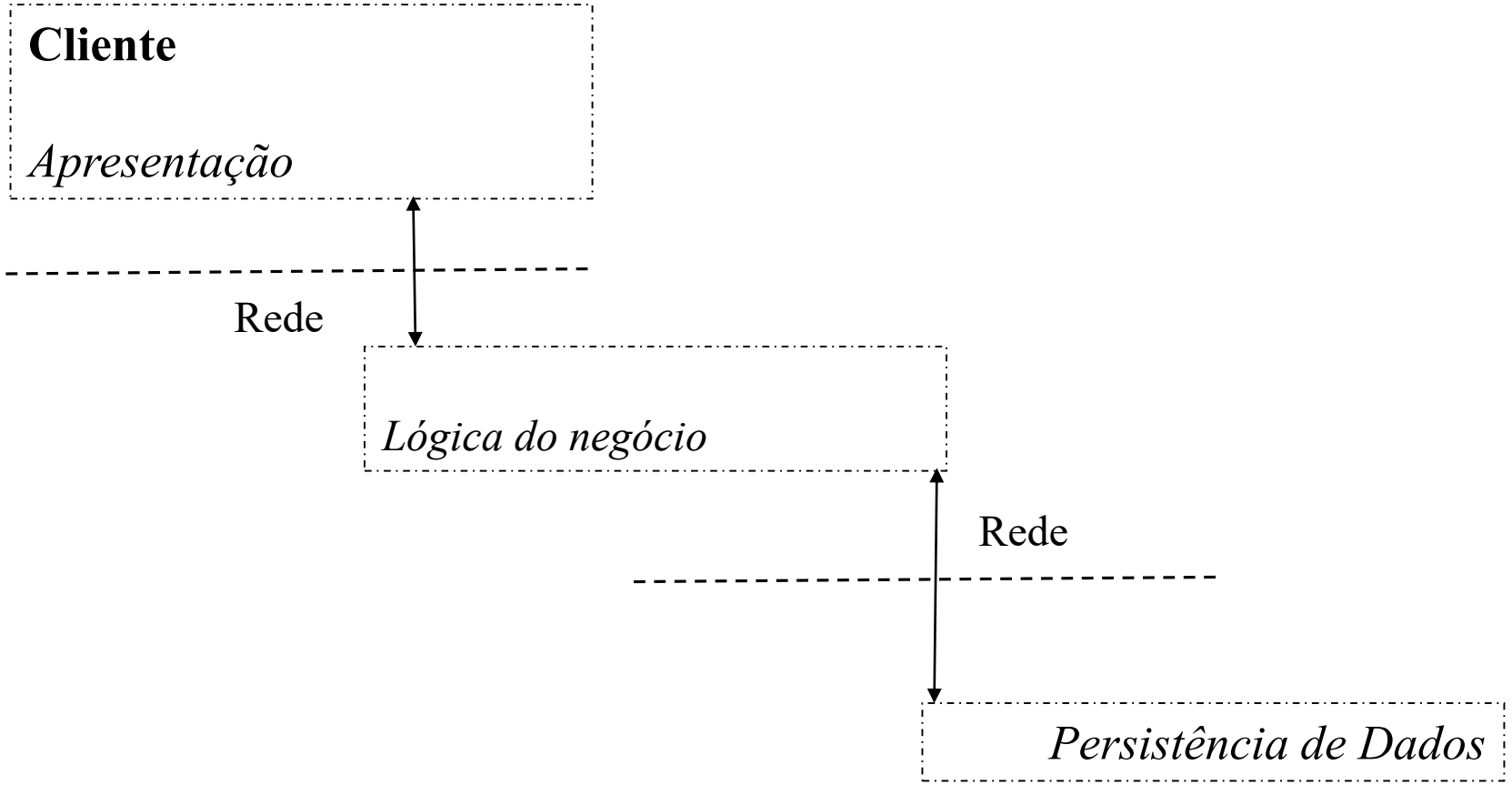
3 - Modelos arquitecturais

Modelo em duas camadas (2-tiers)



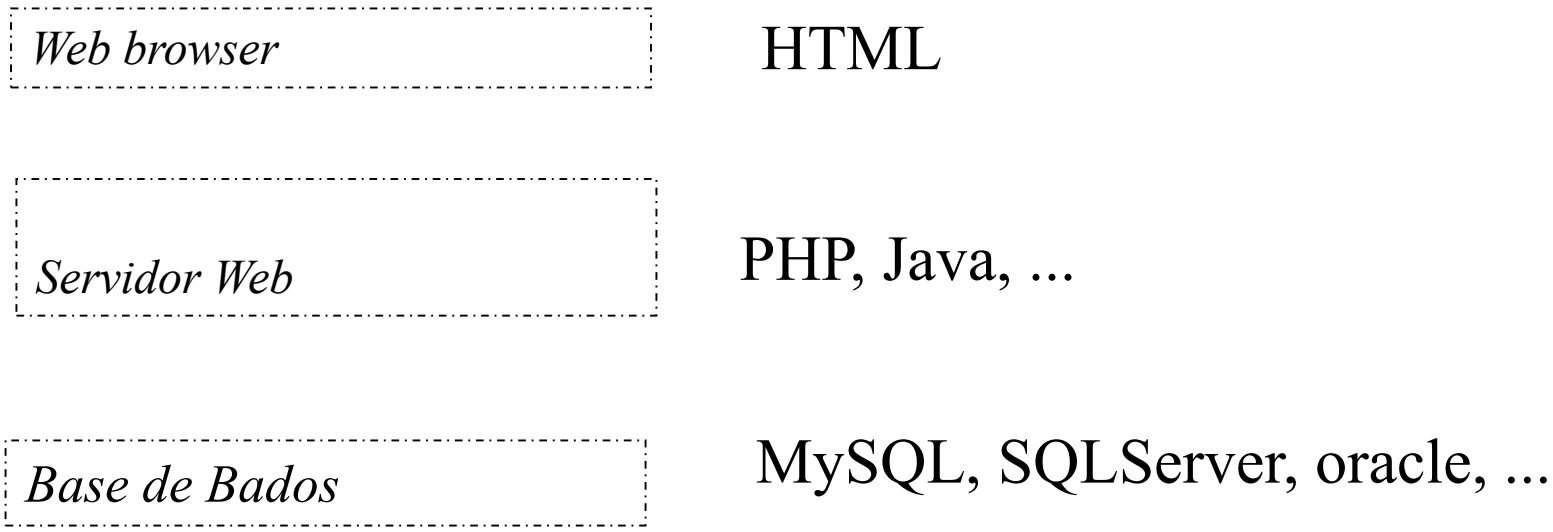
3 - Modelos arquitecturais

Modelo em três camadas (3-tiers)

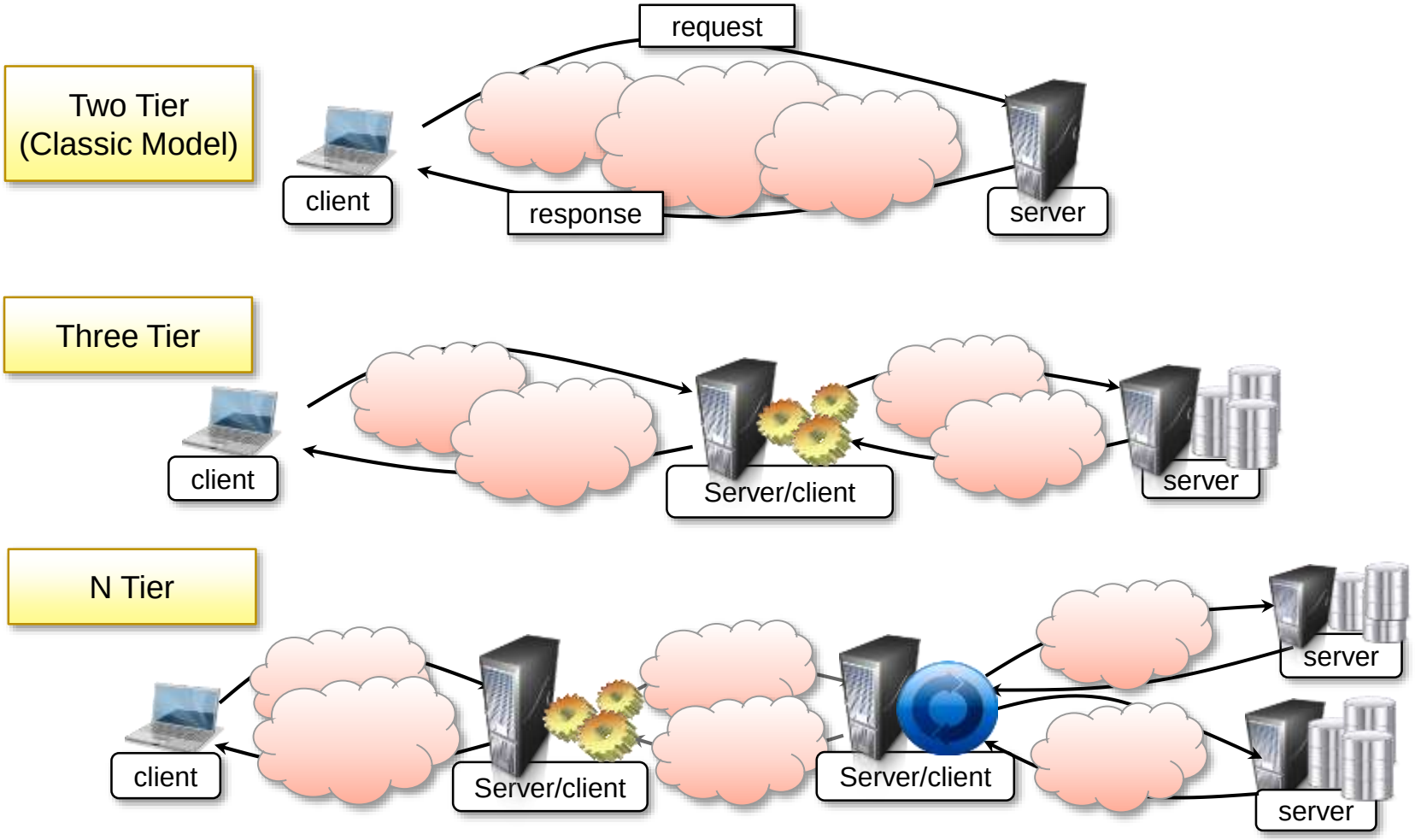


3 - Modelos arquitecturais

Exemplo de uma arquitectura em 3 camadas



3 - Modelos arquitecturais



3 - Modelos arquitecturais

Modelo cliente Servidor

- Modelo mais usado na prática

- Modelo de interação simples

- Segurança concentrada no servidor

- Servidor é um ponto de falha único

- Não é escalável para além de certos limites

Exemplos de grandes Sistemas Distribuídos com arquitetura cliente servidor:

Google, GoogleMaps, YouTube, Facebook, etc.

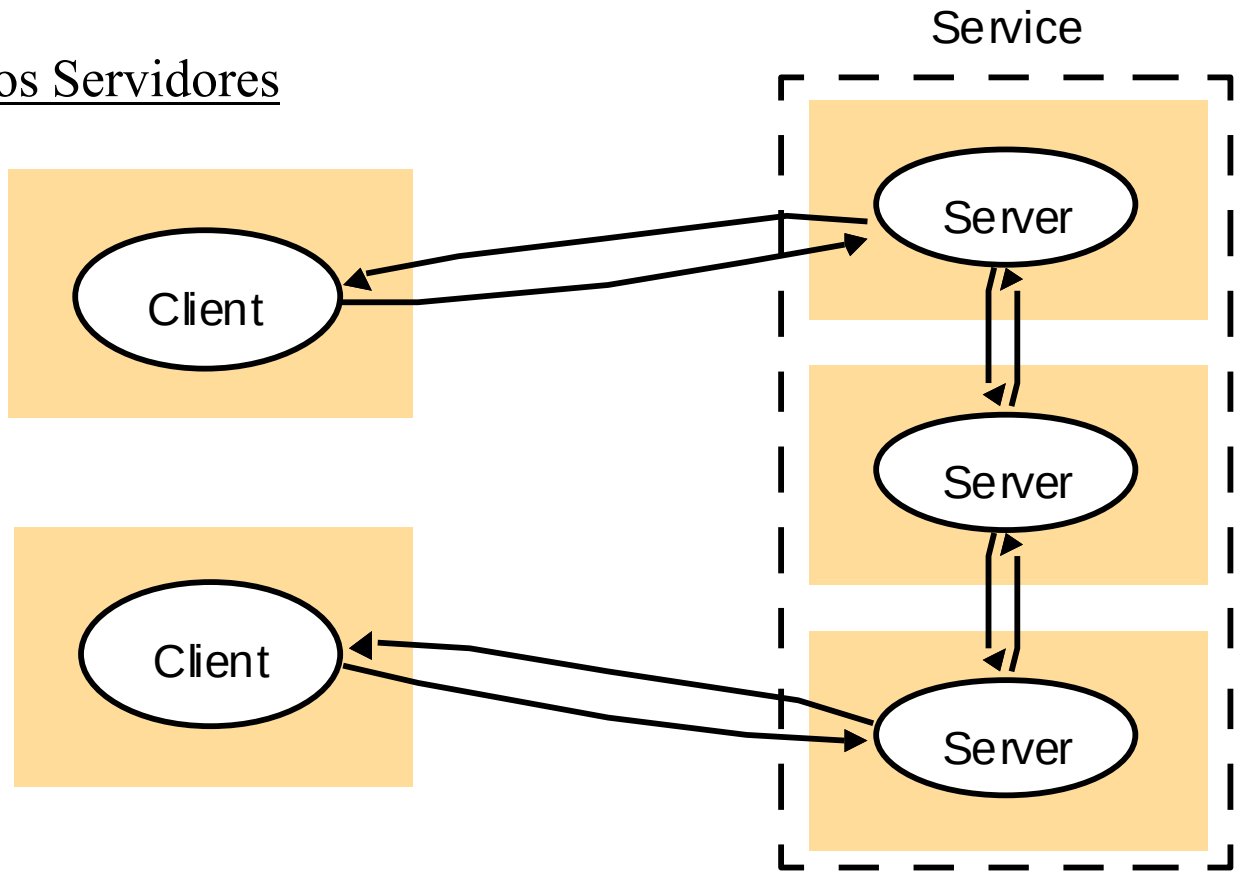
Variantes podem colmatar alguns problemas:
particionamento, replicação, hierarquia, etc...

Múltiplos Servidores

Um serviço pode ser implementado por vários processos servidores localizados em diferentes computadores.

3 - Modelos arquitecturais

Múltiplos Servidores



• Entidades que fornecem o serviço estão distribuídas por diferentes máquinas

ou

• Cópias replicadas pelas diferentes máquinas - > disponibilidade > tolerância a falhas

Cliente / Servidor **replicado:**

- Existem vários servidores, capazes de responder aos mesmos pedidos

- **Vantagens:**

- Permite distribuir a carga, melhorando o desempenho
- Não existe um ponto de falha único

- **Principal problema:**

- Manter estado do servidor coerente em todas as réplicas

Cliente / Servidor **particionado:**

- Existem vários servidores com a mesma interface, cada um capaz de responder **a uma parte dos pedidos (ex. DNS)**
- Servidor redirige o cliente para outro servidor (**iterativo)**
- Servidor invoca o pedido noutro servidor (**recursivo)**

Cliente / Servidor **particionado:**

• Vantagens →**Escalabilidade**

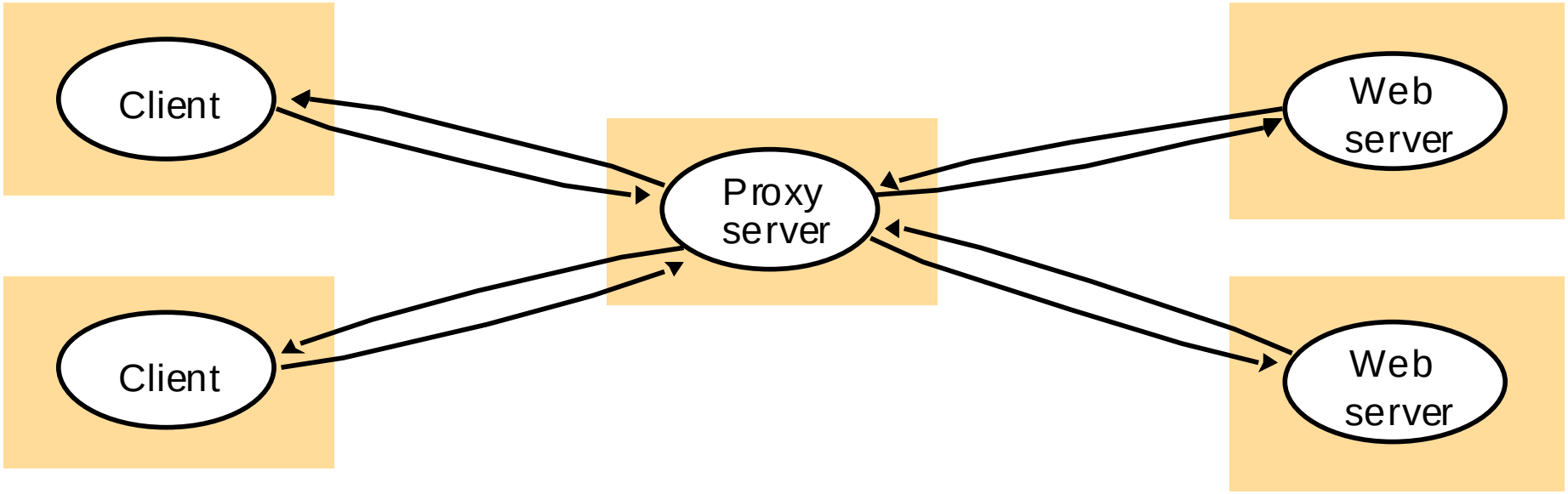
- Permite distribuir a carga, melhorando o desempenho
- Não existe um ponto de falha único

• Problemas

- Falha de um servidor impede acesso aos dados presentes nesse servidor

3 - Modelos arquitecturais

"Proxy servers and Caches"



• Uma "cache" permite o armazenamento, numa localização mais próxima, de dados/objectos recentemente usados

3 - Modelos arquitecturais

"Proxy servers and Caches"

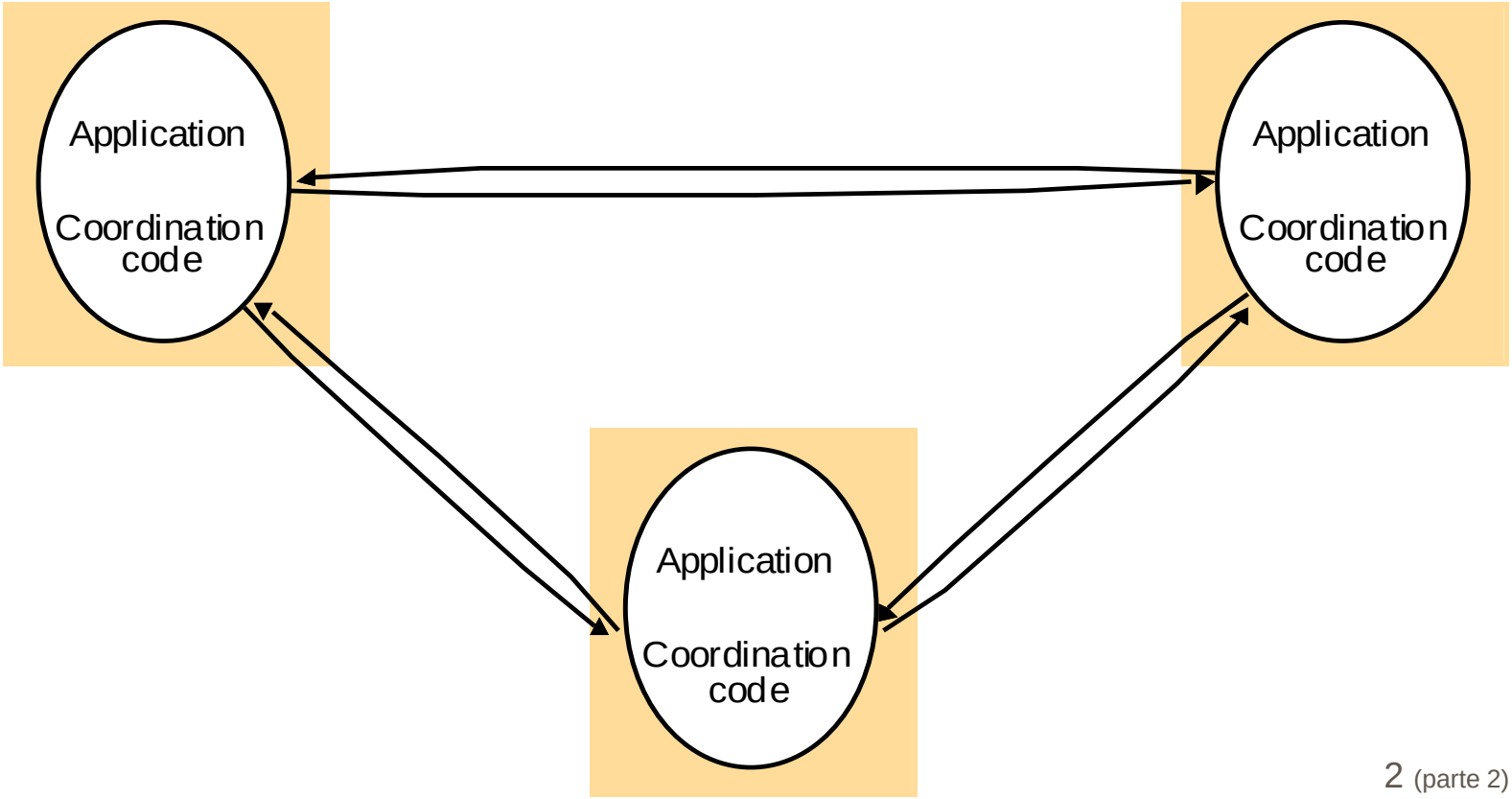
•Quando um cliente necessita de um objecto, o serviço de "caching" verifica se possui uma cópia actualizada do objecto, em caso afirmativo fornece essa cópia.

• Uma "cache" pode estar localizada no cliente ou em servidores "proxy" que são partilhados por vários clientes.

• Objectivo: aumentar a disponibilidade e a performance do serviço

"Processos pares" (peer processes)

Todos os processos desempenham papeis similares. Cada processo é responsável pela consistência dos seus dados (recursos) e pela sincronização das várias operações.



3 - Modelos arquitecturais

"Processos pares" (peer processes)

➔ Cada processo pode assumir (simultaneamente ou alternadamente) o papel de cliente e servidor do mesmo serviço

Paradigma de distribuição em que os serviços são suportados diretamente pelos seus clientes/utilizadores, sem recurso a uma infra-estrutura criada e mantida explicitamente para esse fim.

A ideia base é conseguir explorar os recursos disponíveis nas máquina ligadas em rede: cpu, disco, largura de banda...

"Processos pares" (peer processes)

• Modelos de interação e coordenação mais complexos (que em sistemas cliente/servidor)

• Algoritmos mais complexos

• Não existe ponto único de falha

• Grande potencial de escalabilidade

• Apropriado para ambientes em que todos os participantes querem cooperar para fornecer um dado serviço

"Processos pares" ...

Exemplos de grandes Sistemas Distribuídos com arquitetura peer-to-peer:

Napster, Kazaa, Gnutella, BitTorrent, ..., Skype, ...

Duas arquitecturas principais:

- Um servidor de diretório centralizado (Ex. – Napster)

  - Quando um novo processo se liga, informa o servidor central (do seu endereço, do seu conteúdo)

  - A partir daí pode comunicar com os outros pares

"Processos pares" ...

Serviço de diretório distribuído
(vários grupos de processos, ex. – Kazaa)

- Quando um novo processo se liga, liga-se a um grupo

- O líder do grupo regista o conteúdo de todos os elementos do grupo

- Cada processo acede ao líder do seu grupo para localizar o que pretende.

- Cada líder pode aceder aos outros líderes.

Outros modelos:

Variantes do modelo cliente servidor resultantes de:

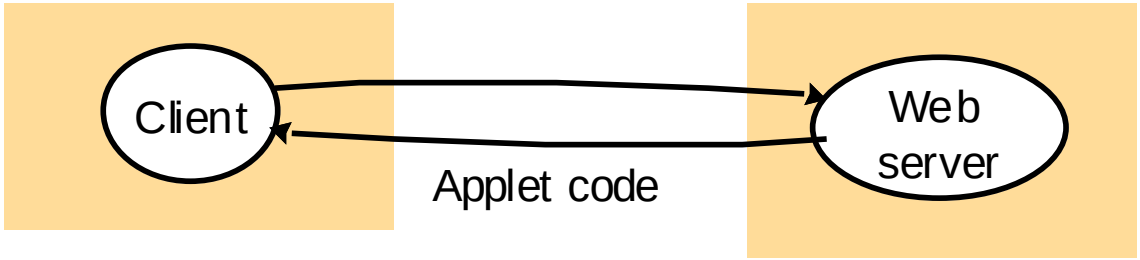
. Uso de código móvel;

. Uso de sistemas com hardware limitado;

. Requisitos de adicionar/remover ao/do sistema periféricos móveis;

Web Applets

a) client request results in the downloading of applet code



b) client interacts with the applet

3 - Modelos arquitecturais

Agentes móveis

- Um Agente é um programa executável que pode "mover-se" de uma máquina para outra.

Age em nome de um utilizador específico, e num dado computador,
para o qual se transfere, realizando algum serviço para o seu proprietário, podendo obter informações que mais tarde transmitirá ao local de origem.

Entidade capaz de interagir autonomamente com o ambiente que o rodeia, podendo apresentar características de adaptabilidade, mobilidade, cooperação ou competição.

. Problemas de segurança

. Dificuldades em realizar o trabalho devido a problemas impossíveis de prever

Network Computers

Computadores sem disco contando com o apoio da rede para fornecer os serviços ao utilizador.

-Aplicações **executam localmente**, os ficheiros são geridos por um servidor remoto.

Solução económica para centros de computação com poucos recursos

3 - Modelos arquitecturais

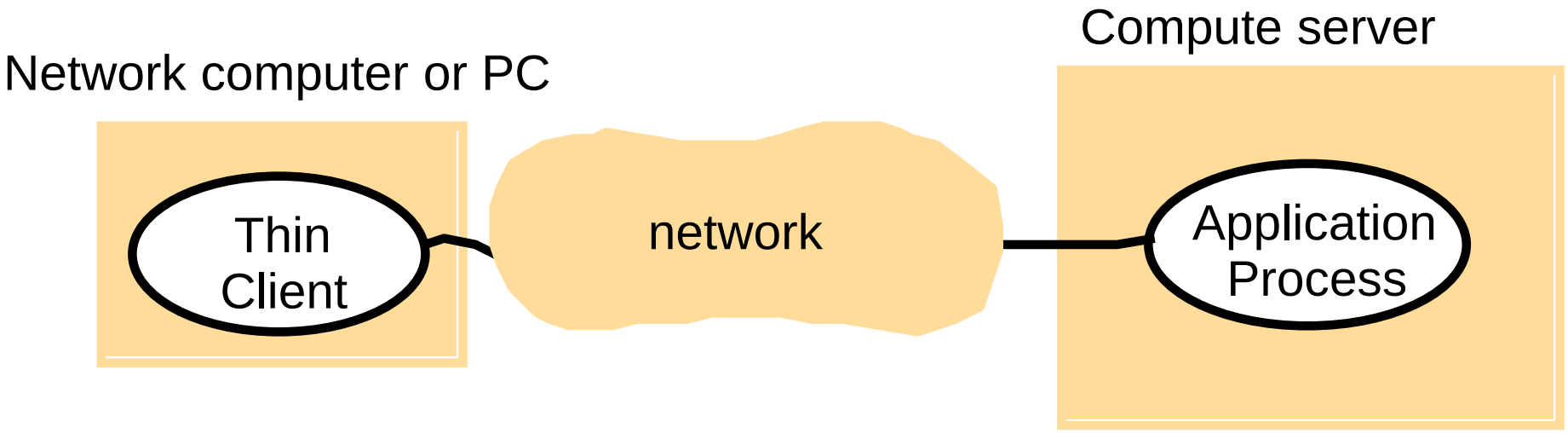
Thin Clients

- interface gráfica, baseada em windows, na máquina local ao utilizador

- as aplicações executam no servidor

. *problemas para aplicações gráficas interactivas*

Ex. Citrix WinFrame



3 - Modelos arquitecturais

Equipamentos móveis e redes espontâneas

- Laptops, PDA (Personal Digital Assistant), telemóveis, câmeras digitais, máquinas de lavar roupa, relógios, etc

- Protocolos Wireless: BlueTooth, Infrared, HomeRF

Principais características das redes sem fios:

- Configuração é feita automaticamente sem intervenção humana

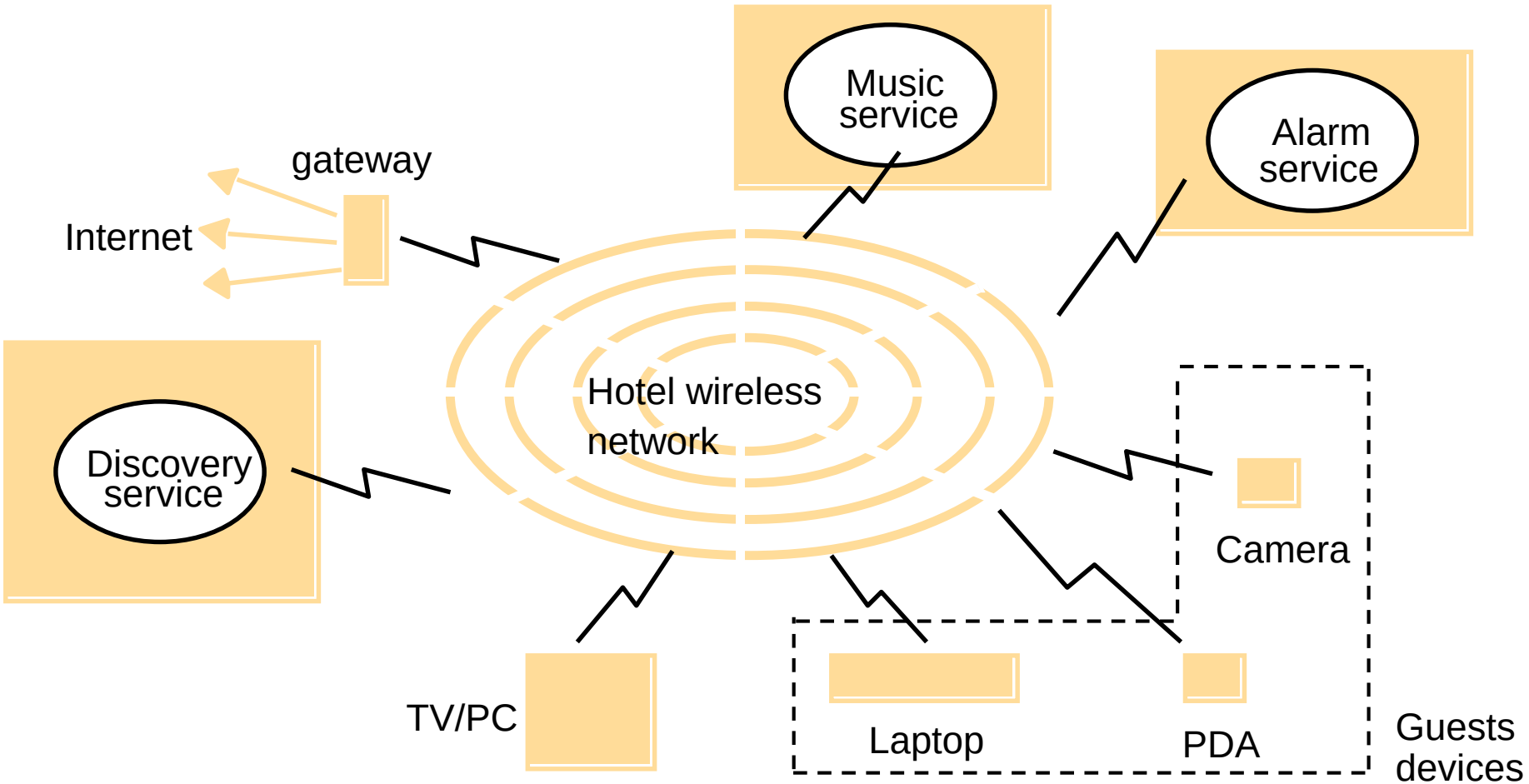
- Os equipamentos digitais móveis descobrem por si os serviços disponíveis

Problemas:

- Conexão limitada
(se os dispositivos se afastam demasiado do local de transmissão?)

- Segurança e privacidade

3 - Modelos arquitecturais



Redes espontâneas exigem:

Um meio de os clientes (equipamentos móveis) descobrirem que serviços estão disponíveis na rede a que se ligaram.

Um "discovery service" é um servidor (um ou mais processos) que mantém uma lista dos tipos e características dos serviços disponíveis dentro da rede local sem fios.

Oferecem dois tipos de serviços:

- Registo de serviços

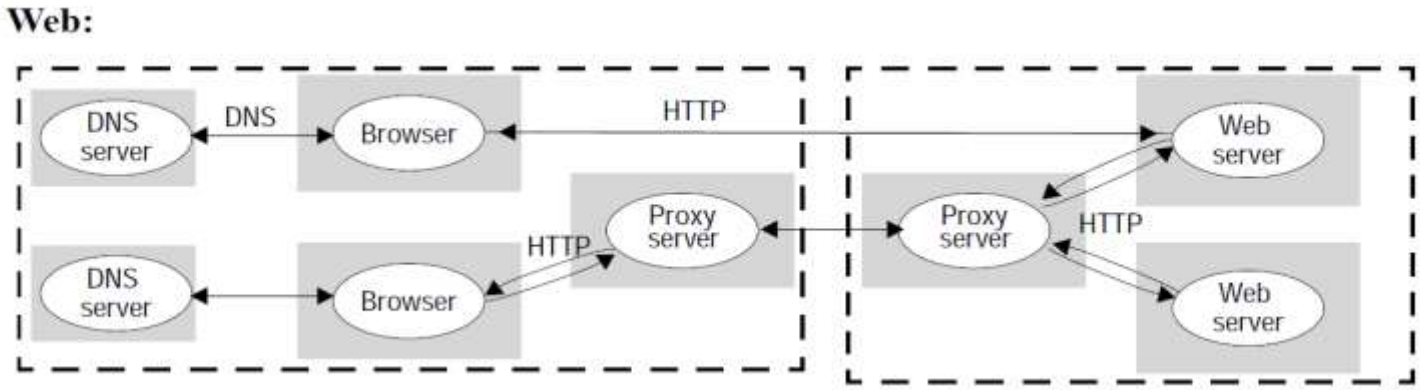
➔aceita pedidos para registar numa base de dados os detalhes de cada serviço disponível

- Lookup de serviços

➔ aceita “queries” aos serviços disponíveis, fornecendo detalhes suficientes para que o cliente se possa ligar ao serviço que escolher

3 - Modelos arquitecturais

Exemplo: Arquitetura de uma aplicação WEB



- Um browser é cliente do servidor de DNS e cliente do servidor Web;

- Algumas intranets, usam servidores Proxy.

Quando o proxy está localizado do lado do cliente, minimiza o trafego e atrasos da rede.

Quando o proxy está localizado do lado do servidor reduzem a carga do servidor.

O proxy e o servidor cooperam para fornecer o serviço. O proxy é responsável por manter a consistência, verificando com frequência a data em que as páginas foram modificadas no servidor.

Exercício:

Desenhe a arquitetura da aplicação do exercício 5, da FP02

i) Registar aluno; ii) Consultar quais os alunos registados; iii) Consultar o número de acessos ao servidor ...; iv) Dado um nome de aluno devolver o seu número e o seu contacto

Como cliente /servidor que comunica por sokets:

Exercício:

Desenhe a arquitetura da aplicação do exercício 3, da FP05

i) Registar aluno; ii) Consultar quais os alunos registados; iii) Consultar o número de acessos ao servidor ...; iv) Dado um nome de aluno devolver o seu número e o seu contacto

Servidor multi-treaded

Exercício:

Desenhe a arquitetura da aplicação do exercício 6, da FP06

Registar aluno; ii) Consultar quais os alunos registados; iii) Consultar o número de acessos ao servidor ...; iv) Dado um nome de aluno devolver o seu número e o seu contacto

Aplicação Java RMI

Exercício

– Suponha a classe AlunoSD, ilustrada abaixo, que representa os alunos de sistemas distribuídos com as suas notas finais. Foram definidos os getters e setters os métodos toString, equals e o construtor de omissão.

```
public class AlunoSD implements Serializable   {
        private int numero;
        private String nome;
        private int nota;
        …
```

Pretende-se uma aplicação cliente/servidor em Java RMI, em que o servidor possua um objeto remoto com uma lista de objetos do tipo AlunoSD, e que permita ao utilizador realizar as seguintes operações remotas:

Exercício

1 – Atribuir uma nota a um aluno, identificado pelo seu número;
2 – Consultar qual o número de alunos aprovados;
3 – Dado um nome obter todos os alunos com esse nome;
4 – Inserir um aluno na lista;

a) Construa todas as classes necessárias para a implementação de um protótipo que permita testar a aplicação descrita (Pode omitir o tratamento de exceções)

b) Modifique a classe servidor de forma a que, após a execução do método que corresponde à opção3 seja criada uma Thread que vai escrever para um ficheiro a lista de AlunoSD resultante. O ficheiro deve ter o nome AlunoSD, e para escrever a lista para um ficheiro pode invocar a função:

void writeFile (String fileName, ArrayList<AlunoSD> list);

socket do cliente:

```
        import java.net.*;
        import java.io.*;
Socket meuCliente = null;
        try { meuCliente = new Socket ("host", portNumber); }
        catch (IOException e){ … }
```

socket do servidor:

```
        ServerSocket  meuServidor = null;
        try { meuServidor = new ServerSocket (portNumber); }
        catch (IOException e){ …}
        Socket sServidor = null
        try { sServidor = meuServidor.accept(); }
       catch (IOException e){  }
```

Obter as Streams do socket e associar objectStrems:

```
ObjectOutputStream os = new ObjectOutputStream (
        meuCliente.getOutputStream());

ObjectInputStream is = new ObjectInputStream (
        meuCliente.getInputStream());

String s = "exemplo";

os.writeObject(s);
s  = (String) is.readObject();
```

RMI

Interface remota: java.rmi.Remote
Exceção remota: java.rmi.RemoteException
Objeto remoto : java.rmi.server.UnicastRemoteObject;

Sintaxe do nome que o objecto remoto tem no RMIregistry:
[rmi:] [//] [nomeMaquina] [:port] [/nomeObjecto]

Instalar um gestor de segurança:
System.setSecurityManager (new SecurityManager());
Iniciar o registry:
java.rmi.registry.LocateRegistry.createRegistry(1099);
Métodos da classe java.rmi.Naming:
void rebind (String nomeObjecto, Remote objecto);
Remote lookup (String nomeObjecto)