SISTEMAS DISTRIBUÍDOS

João Leitão, Sérgio Duarte, Pedro Camponês

(baseado nos slides de Nuno Preguiça)

Aula 9: Capítulo 2

Arquiteturas e Modelos de Sistemas Distribuídos

NOTA PRÉVIA

A apresentação utiliza algumas das figuras do livro de base do curso

G. Coulouris, J. Dollimore and T. Kindberg,
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition, 2005

ORGANIZAÇÃO DO CAPÍTULO

Modelos arquiteturais

- Arquitetura/camadas de software
- Cliente/servidor, peer-to-peer, variantes

Modelos fundamentais – usados para descrever propriedades parciais, comuns a todas as arquiteturas

- Modelo de interação
- Modelo de falhas
- Modelo de segurança

CONTEXTOS - ARQUITETURA

Camadas de software

- Reparte a complexidade de um sistema, em várias camadas, com interfaces bem definidas entre si. Cada camada pode usar os serviços da camada abaixo, sem conhecimento dos detalhes de implementação.

Arquitetura (distribuída) multinível/camada

- as camadas do sistema são atribuídas a processos/máquinas diferentes

Arquitetura distribuída

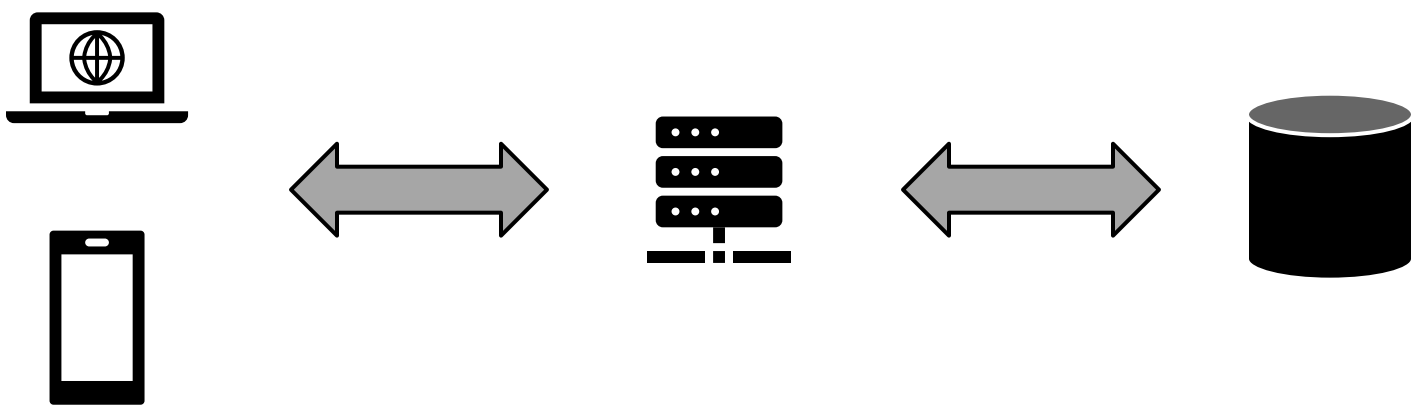
- Especifica como se organizam e quais as interações entre os vários **componentes** de **um sistema distribuído**

- Em todos os casos há implicações no desempenho, fiabilidade e segurança do sistema

ARQUITETURA EM CAMADAS: MODELO THREE-TIER

As aplicações Web e móveis são frequentemente implementadas usando uma arquitetura de três níveis:

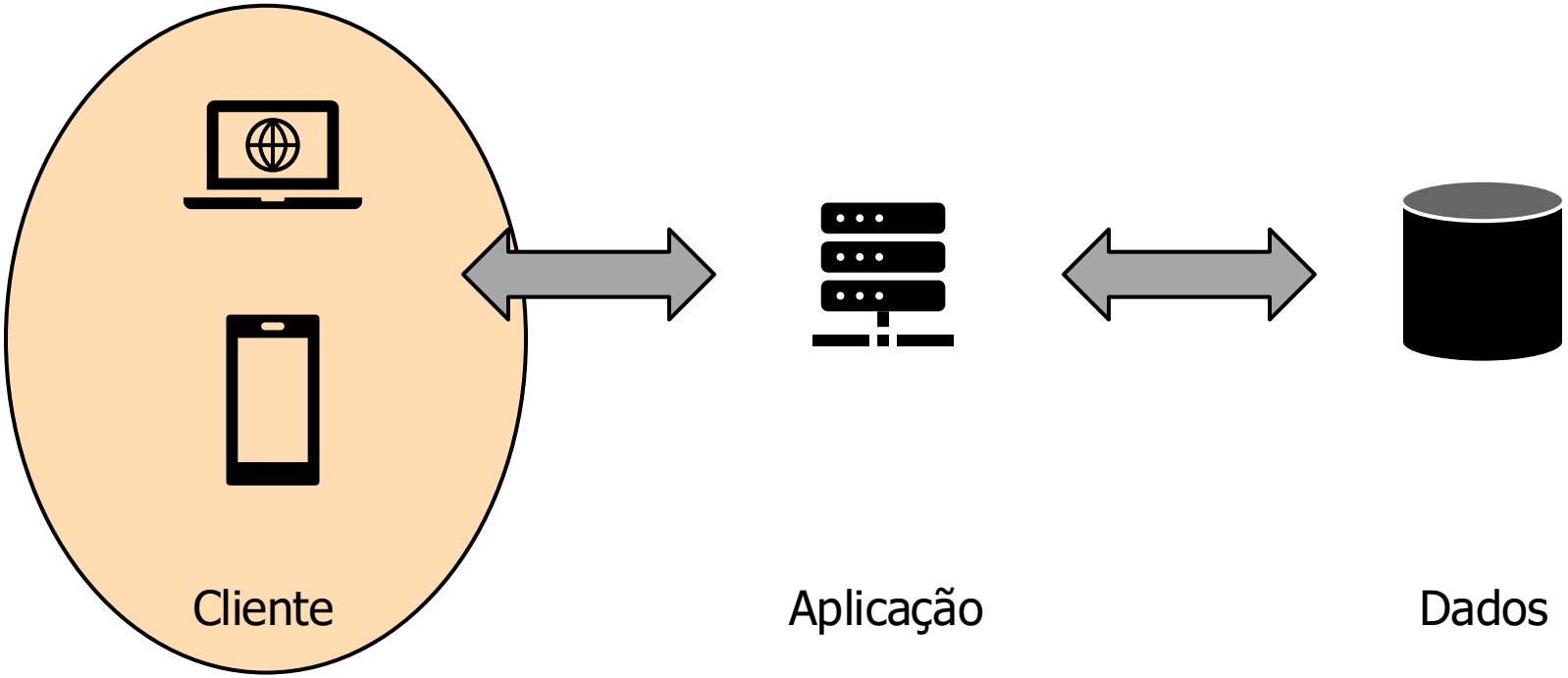
- Cliente (ou apresentação)
- Aplicação (ou lógica)
- Dados (ou armazenamento)



ARQUITETURA EM CAMADAS: MODELO THREE-TIER

O nível do cliente é composto:

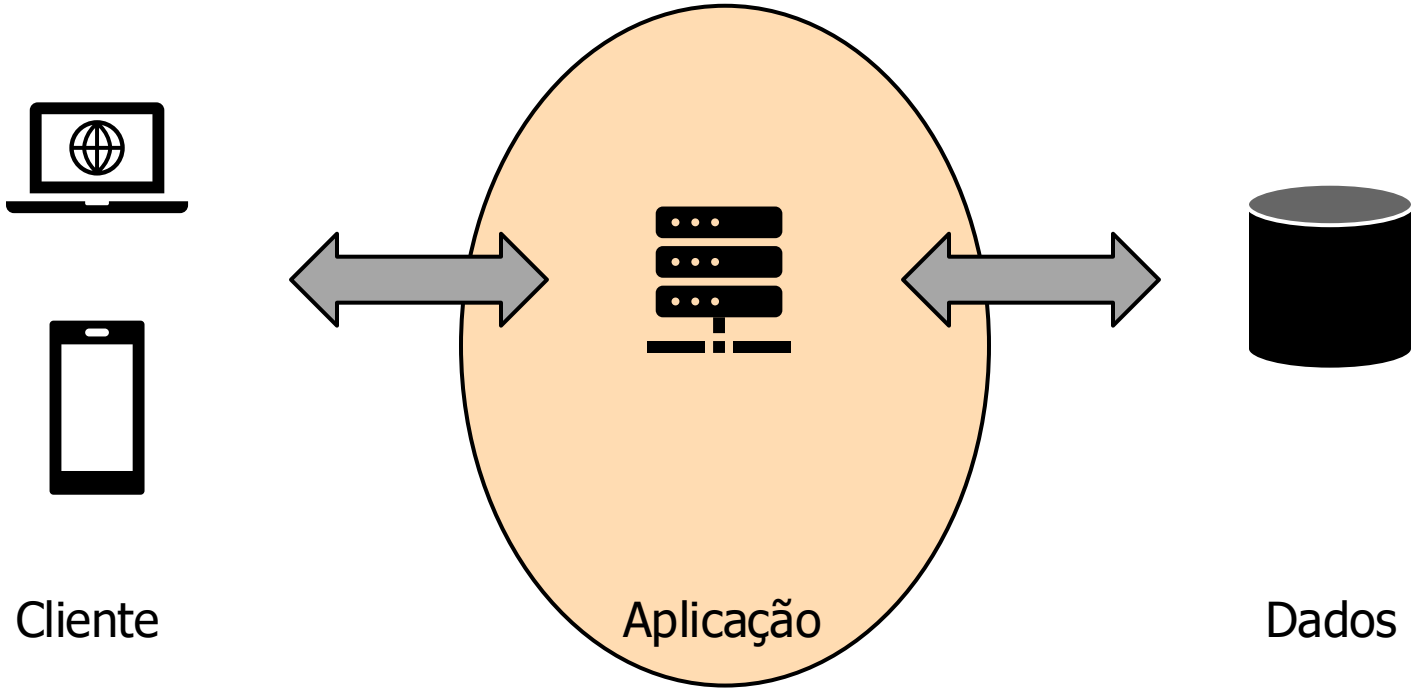
- Clientes Web: HTML + Javascript
- Aplicações móveis: Android, iOS, etc.



ARQUITETURA EM CAMADAS: MODELO THREE-TIER

Nível da aplicação é composto por:

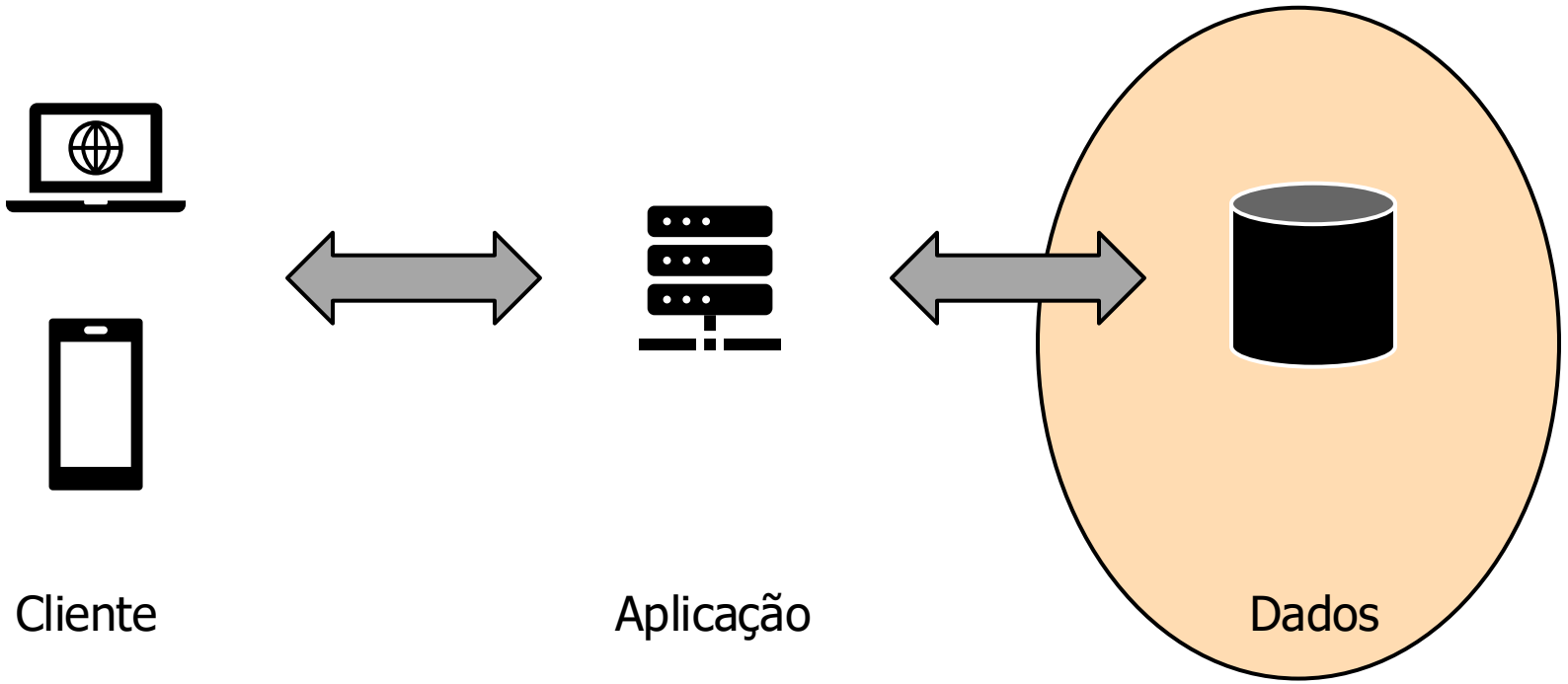
- Servidores REST / SOAP
- Páginas web dinâmicas e estáticas
- Implementado usando servidores aplicacionais
  - E.g.: Tomcat, Wildfly, ASP.NET, etc.



ARQUITETURA EM CAMADAS: MODELO THREE-TIER

O nível dos dados é composto por:

- Sistemas de ficheiros, BLOB stores, key-value stores, bases de dados SQL, etc.
- Outros serviços: caches, filas de mensagens, etc.



ARQUITETURA DISTRIBUÍDA

Arquitetura do sistema

- Organização de um sistema (complexo) em componentes **mais simples** com funcionalidades/responsabilidades próprias

Arquitetura do sistema distribuído

- Define os componentes, o que fazem, onde estão e como interagem entre si.
- Terá implicações em diversas propriedades do sistema: desempenho, fiabilidade e segurança do sistema

ARQUITETURA DISTRIBUÍDA

Arquitetura de um sistema distribuído pode (e deve) ser determinada por diversos fatores:

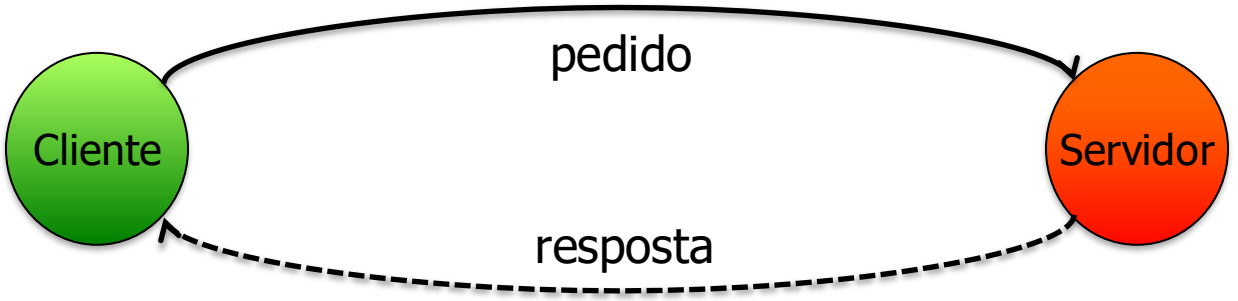
- Requisitos funcionais:
- "tudo relacionado com a função do sistema"
- e.g. Lógica de negócio

Requisitos não funcionais:

- Desempenho (escalabilidade, latência), disponibilidade
- Custo (desenvolvimento, operação, manutenção)
- Segurança, confiabilidade

Arquitetura distribuída mais simples e muito comum?

CLIENTE/SERVIDOR



Sistema em que os processos podem ser divididos em dois tipos, de acordo com o seu modo de operação:

Cliente: programa que solicita pedidos a um processo servidor

Servidor: programa que executa operações solicitadas pelos clientes, enviando-lhes o respectivo resultado

CLIENTE/SERVIDOR: PROPRIEDADES

Arquitetura mais simples, muito comum e usada na prática...

Positivo

- Interação simples facilita implementação
- Segurança apenas tem de se concentrar no servidor

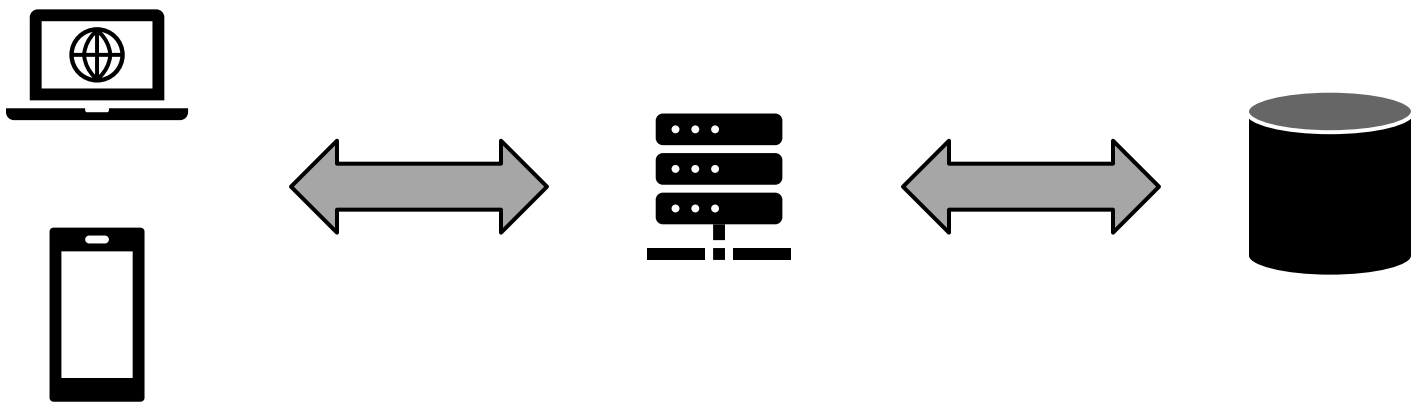
Negativo

- Servidor é um ponto de falha único
- Não escala para além dum dado limite (servidor pode tornar-se ponto de contenção - *bottleneck*)

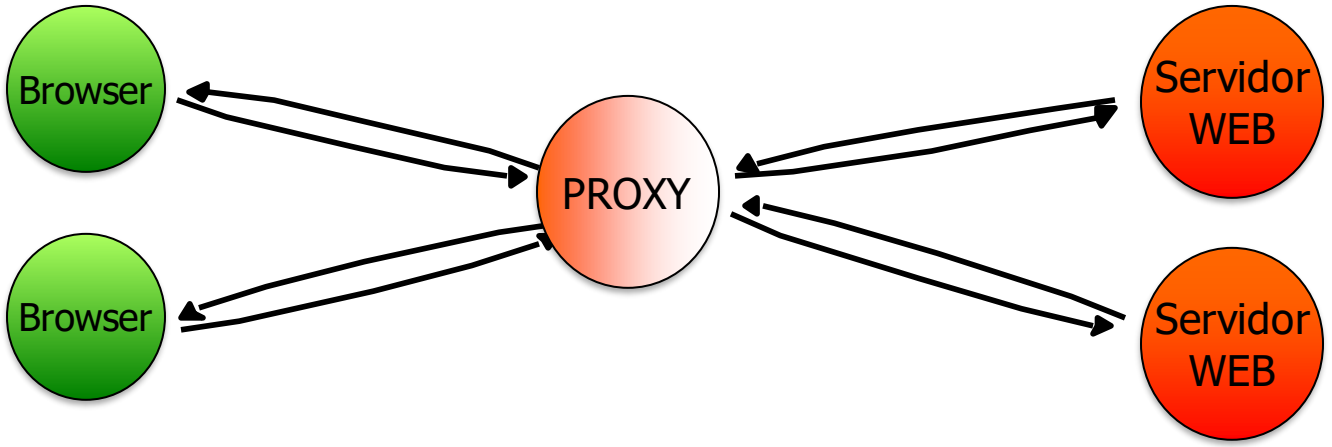
Arquitetura em Camadas e modelo cliente/servidor

As aplicações que usam uma arquitetura em três camadas tipicamente usam interação cliente/servidor entre as várias camadas

- Cliente <-> Aplicação
- Aplicação <-> Dados



NOÇÃO DE PROXY DE UM SERVIÇO



Proxy de um serviço

Componente que fornece um serviço recorrendo a um servidor (do serviço) para executar o serviço

Utilizações possíveis

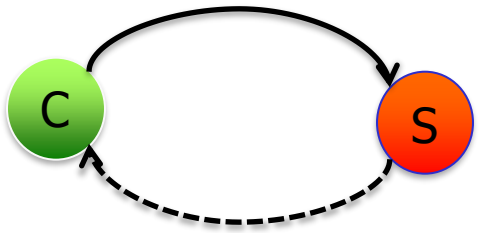
- Intermediário simples (apenas encaminha pedidos e respostas)
- Intermediário complexo (*gateway*)
  - Transformação dos pedidos
- Serviço adicional através do *caching* das respostas
  - Diminuição do tempo de resposta (latência inferior para o proxy)
  - Diminuição da carga do servidor
  - Mascarar falhas do servidor / desconexão

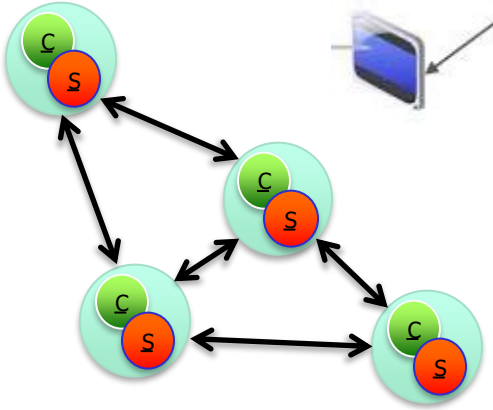
CLIENTE/SERVIDOR: COMPOSIÇÃO

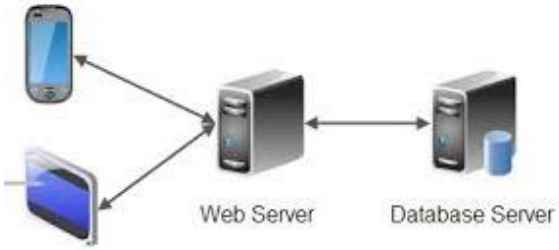
Arquiteturas mais sofisticadas, com novas propriedades, podem ser obtidas por composição do modelo C/S base

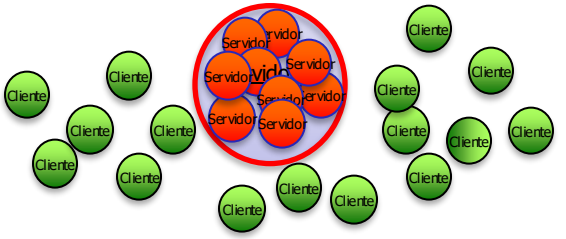
Servidor [particionado, replicado, geo-replicado]

P2P, 3-Tier, etc.



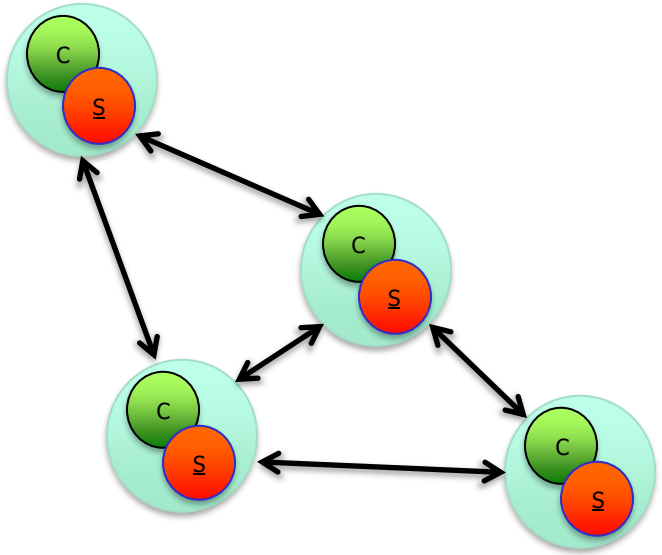






Modelo alternativo para lidar com limitações do modelo cliente/servidor

MODELO *PEER-TO-PEER (P2P)*

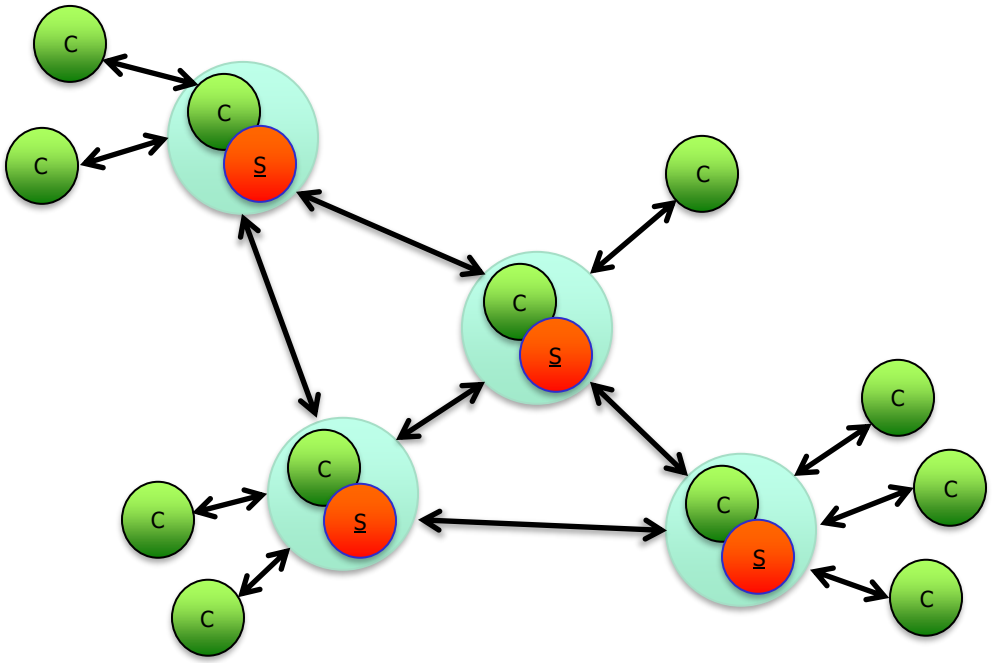


Todos os processos têm funcionalidades semelhantes

- Durante a sua operação podem assumir o papel de clientes e servidores do mesmo serviço em diferentes momentos

Exemplos: partilha de ficheiros, VoIP, edição colaborativa

MODELO *PEER-TO-PEER (P2P)*



Frequentemente o modelo peer-to-peer é usado para implementar um serviço, existindo clientes que usam este serviço contactando um nó do serviço usando o modelo cliente/servidor.

MODELO *PEER-TO-PEER*: PROPRIEDADES

Positivo

- Não existe ponto único de falha
- Melhor potencial de escalabilidade
- Baixo custo de operação

Negativo

- Interação mais complexa (do que num sistema cliente/servidor) leva a implementações mais complexas
  - Operações de pesquisa podem ser complexas
- Maior número de computadores envolvidos pode colocar questões relativas a heterogeneidade e segurança

MODELO *PEER-TO-PEER*: PROPRIEDADES

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

Razões possíveis para cooperação:

- Aumentar a capacidade do sistema
  - Capacidade agregada >> capacidade individual
  - E.g. Sistemas P2P de partilha de ficheiros.
- Manter controlo sobre os dados
  - Cada servidor controla parte dos dados
  - E.g. Sistema de email.

Variantes do modelo cliente/servidor para lidar com limitações de escalabilidade e tolerância a falhas

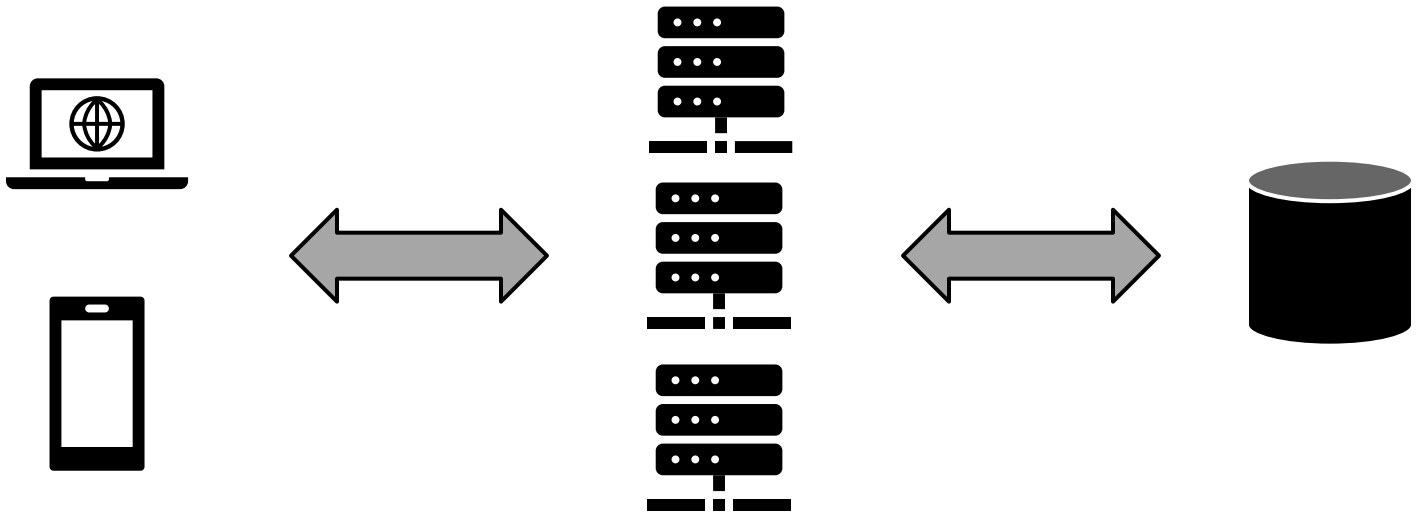
Soluções do lado do servidor

NOTA PRÉVIA

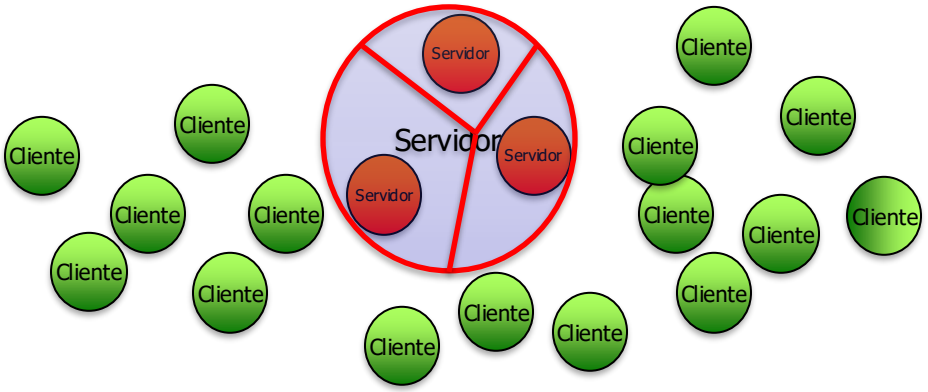
Qual a dificuldade de replicar / particionar um servidor?

Necessário lidar com o estado armazenado pelo servidor.

Se o servidor não tiver estado, basta criar novas cópias do servidor. E.g. replicar o o servidor de aplicação *stateless* numa arquitetura de três níveis.



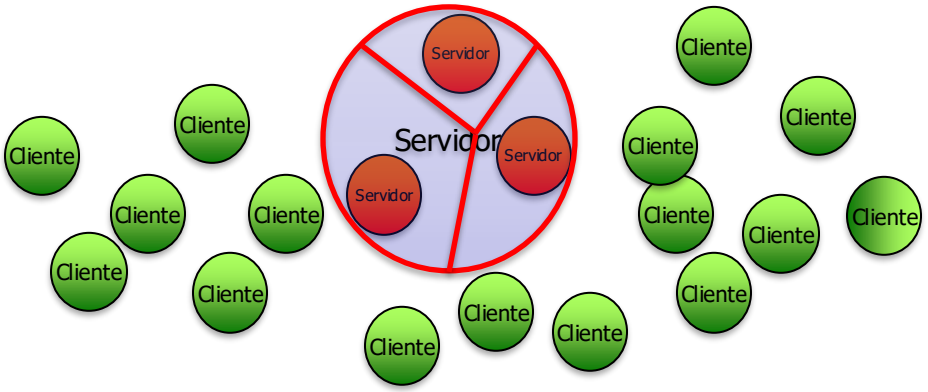
CLIENTE/SERVIDOR PARTICIONADO



Existem vários servidores com a mesma interface, cada um capaz de responder **a uma parte** dos pedidos

Exemplo: DNS.

CLIENTE/SERVIDOR PARTICIONADO



Como é que o pedido do cliente é enviado para o servidor correto?

- Servidor redirige cliente para outro servidor (iterativo)
- Servidor invoca pedido noutro servidor (recursivo)

CLIENTE/SERVIDOR PARTICIONADO

Positivo

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

Negativo

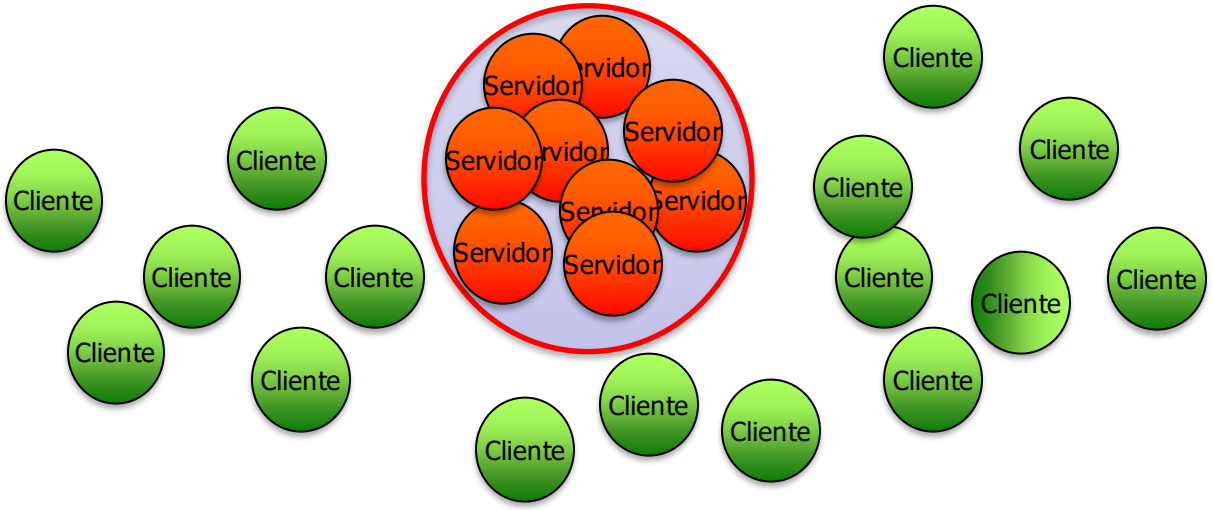
- Falha de um servidor impede acesso aos dados presentes nesse servidor
- Difícil de aplicar em alguns modelos de dados

Sendo: *w* : nº de escritas; *r* : nº de leituras; n : nº de partições

Cada partição recebe, em média: w / n + r / n pedidos

CLIENTE/SERVIDOR REPLICADO

Existem vários servidores idênticos (i.e. capazes de responder aos mesmo pedidos)



CLIENTE/SERVIDOR REPLICADO

Positivo

- Redundância - não existe um ponto de falha único
- Permite distribuir a carga, melhorando o desempenho (potencialmente)

Negativo

- Coordenação - manter estado do servidor coerente em todas as réplicas
- Recuperar da falha parcial de um servidor

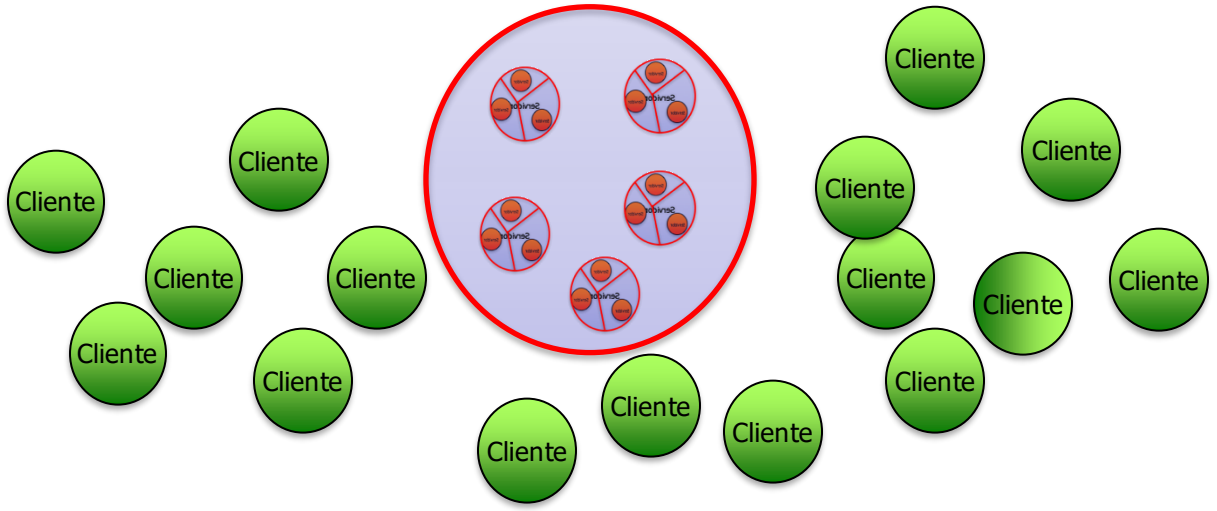
Sendo: *w* : nº de escritas; *r* : nº de leituras; n : nº de réplicas

Se todas as réplicas receberem todos os pedidos de escrita, cada réplica recebe: w + r / n pedidos

CLIENTE/SERVIDOR REPLICADO + PARTICIONADO

Na prática, as implementações frequentemente combinam o particionamento com a replicação.

Cada partição é replicada em várias máquinas.



CLIENTE/SERVIDOR REPLICADO + PARTICIONADO

Positivo

- Redundância - não existe um ponto de falha único
- Permite distribuir a carga, melhorando o desempenho (potencialmente)

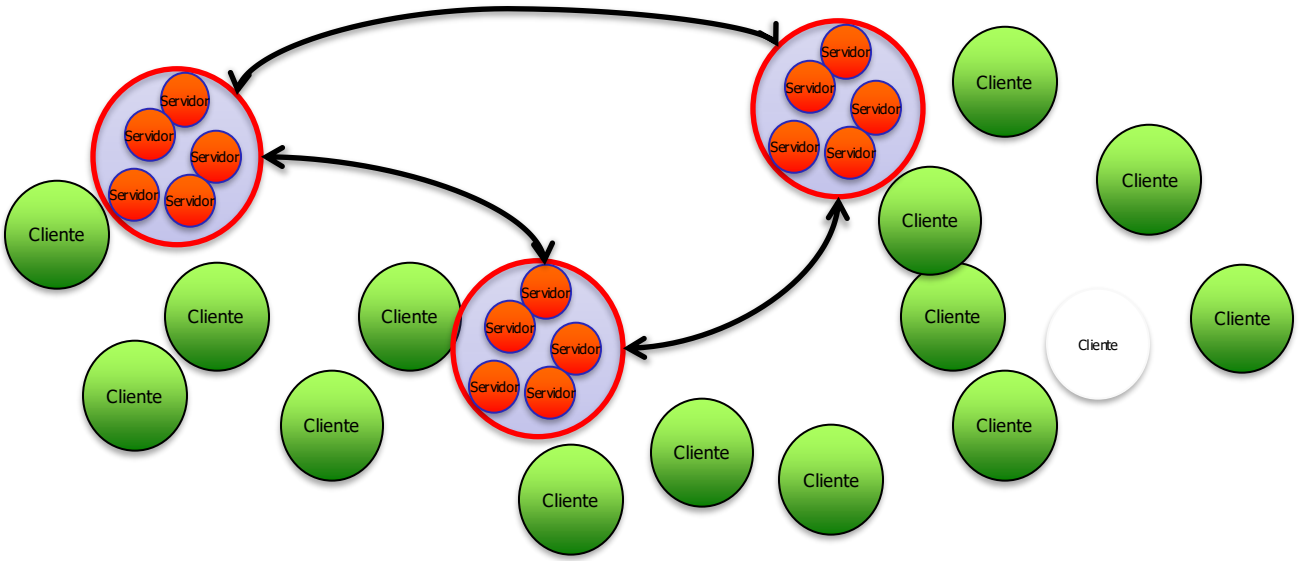
Negativo

- Coordenação - manter estado do servidor coerente em todas as réplicas

CLIENTE/SERVIDOR GEO-REPLICADO

Servidor replicado, com réplicas distribuídas geograficamente

Tipicamente, dentro de cada localização geográfica, os dados são particionados+replicados



CLIENTE/SERVIDOR GEO-REPLICADO

Positivo

- Proximidade aos clientes melhora a qualidade de serviço (latência)
- Redundância acrescida – as falhas das réplicas são (ainda) mais independentes

Negativo

- Coordenação mais dispendiosa – maior separação física das réplicas traduz-se na utilização de canais com latência significativa

Variantes do modelo cliente/servidor para lidar com limitações de escalabilidade e tolerância a falhas

Soluções do lado do cliente

CLIENTE LEVE (*THIN CLIENT*)/SERVIDOR

O cliente apenas inclui uma interface (gráfica) para executar operações no servidor (ex.: browser)

Positivo:

- Cliente pode ser muito simples

Negativo

- Maior peso no servidor
- Impacto na interatividade (latência)

CLIENTE COMPLETO (ESTENDIDO)/SERVIDOR

O cliente executa localmente algumas operações que seriam executadas pelo servidor

Usado na Web em vários níveis: browsers fazem cache de páginas, imagens, etc; HTML 5.0 permite às aplicações Web guardar dados localmente – e.g. suporte offline no Google Docs, Gmail

CLIENTE COMPLETO (ESTENDIDO)/SERVIDOR

Positivo:

- Permite funcionar *offline,* quando não é possível contactar o servidor (recorrendo a *caching*)
- Permite diminuir a carga do servidor e melhorar o desempenho e a interatividade

Negativo:

- Implementação do cliente mais complexa
- Necessário tratar da coerência dos dados entre o cliente e o servidor

PARA SABER MAIS

George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair,
Distributed Systems – Concepts and Design,
Addison-Wesley, 5th Edition, 2011

Capítulo 2.