SISTEMAS DISTRIBUÍDOS

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

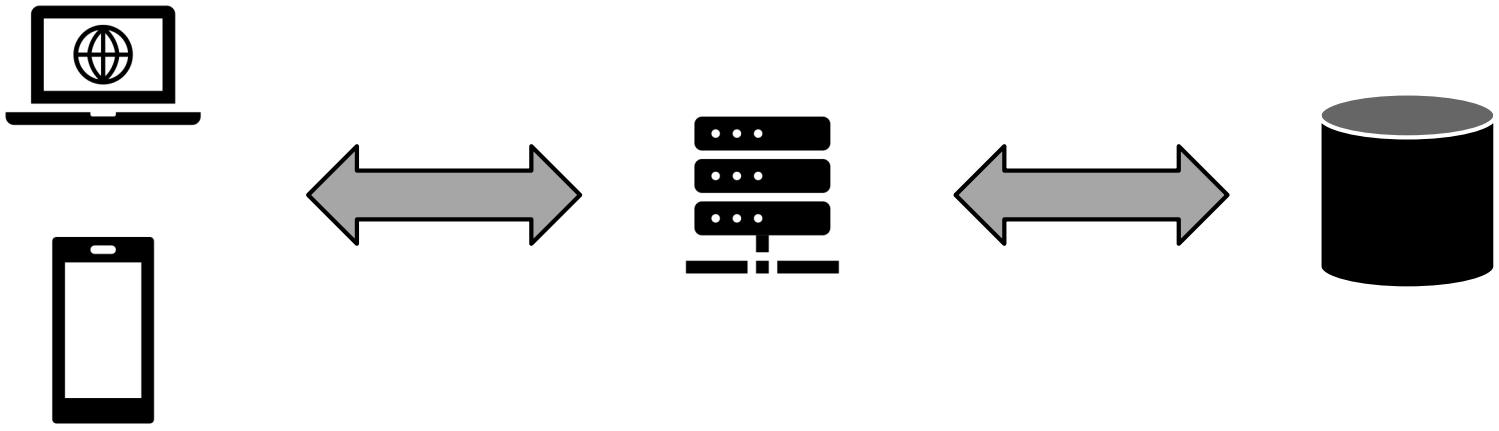
- Especifica como se organizam e quais as interações entre os vários **componentes** de **um sistema distribuído**

- 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:

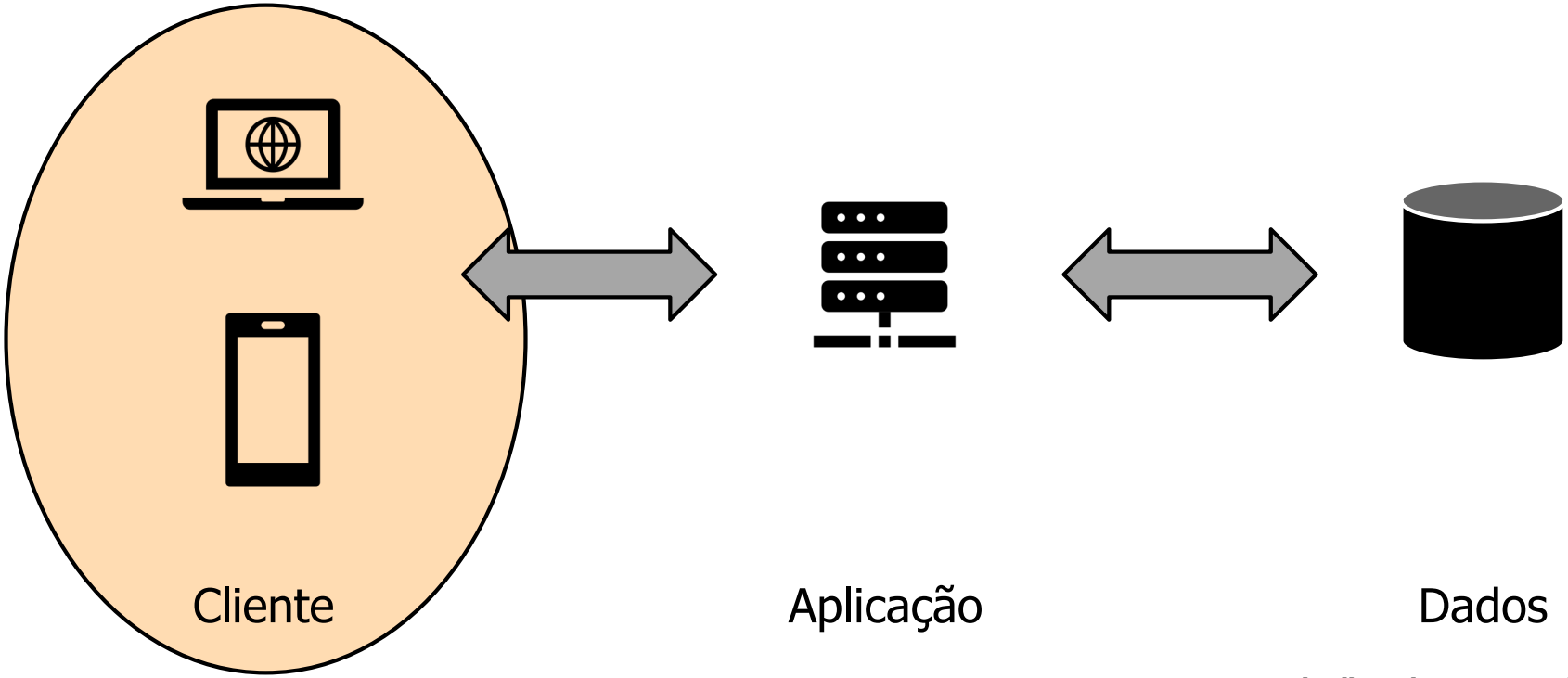
- Cliente (ou apresentação)
- Aplicação (ou lógica)
- Dados (ou armazenamento)



ARQUITETURA EM CAMADAS: MODELO THREE-TIER

O nível do cliente é composto:

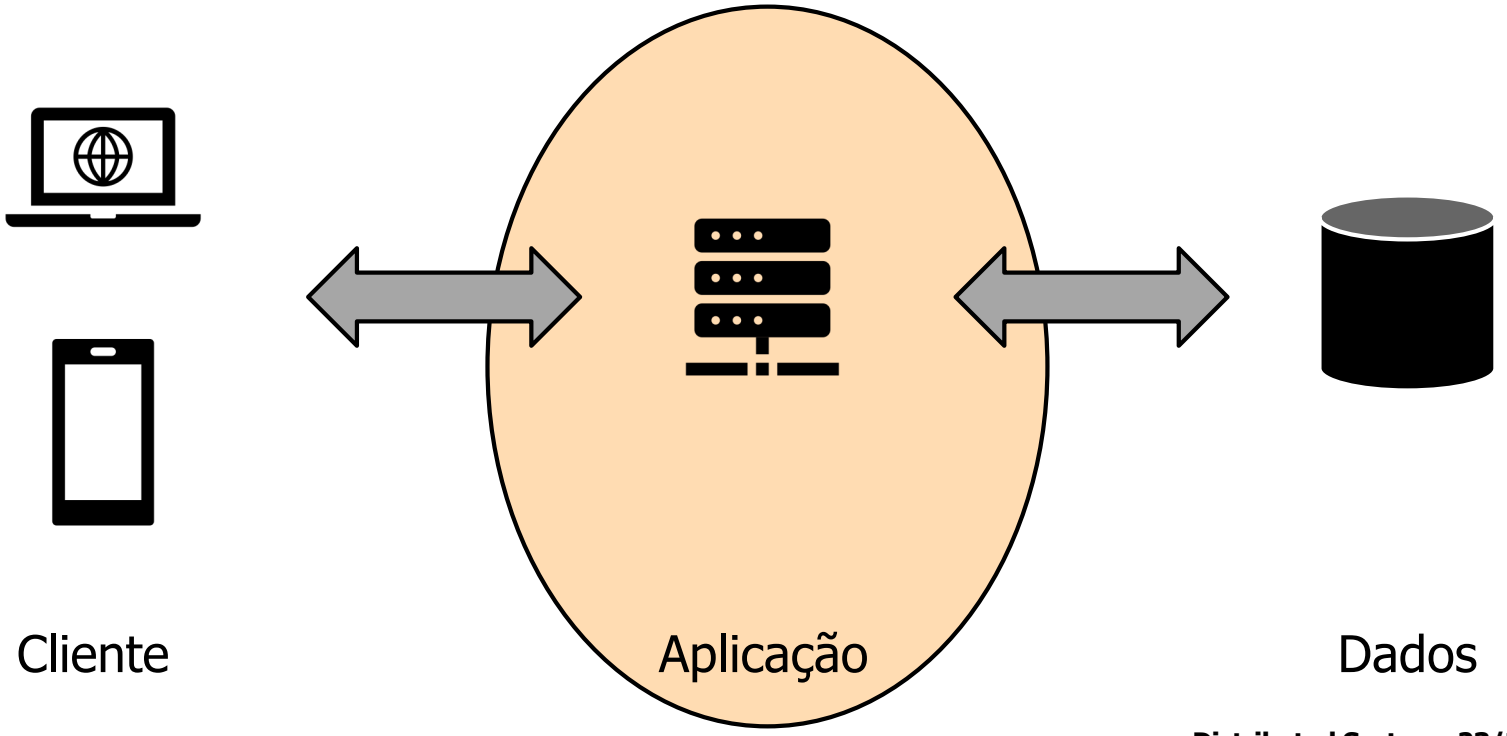
- Clientes Web: HTML + Javascript
- Aplicações móveis: Android, iOS, etc.



ARQUITETURA EM CAMADAS: MODELO THREE-TIER

Nível da aplicação é composto por:

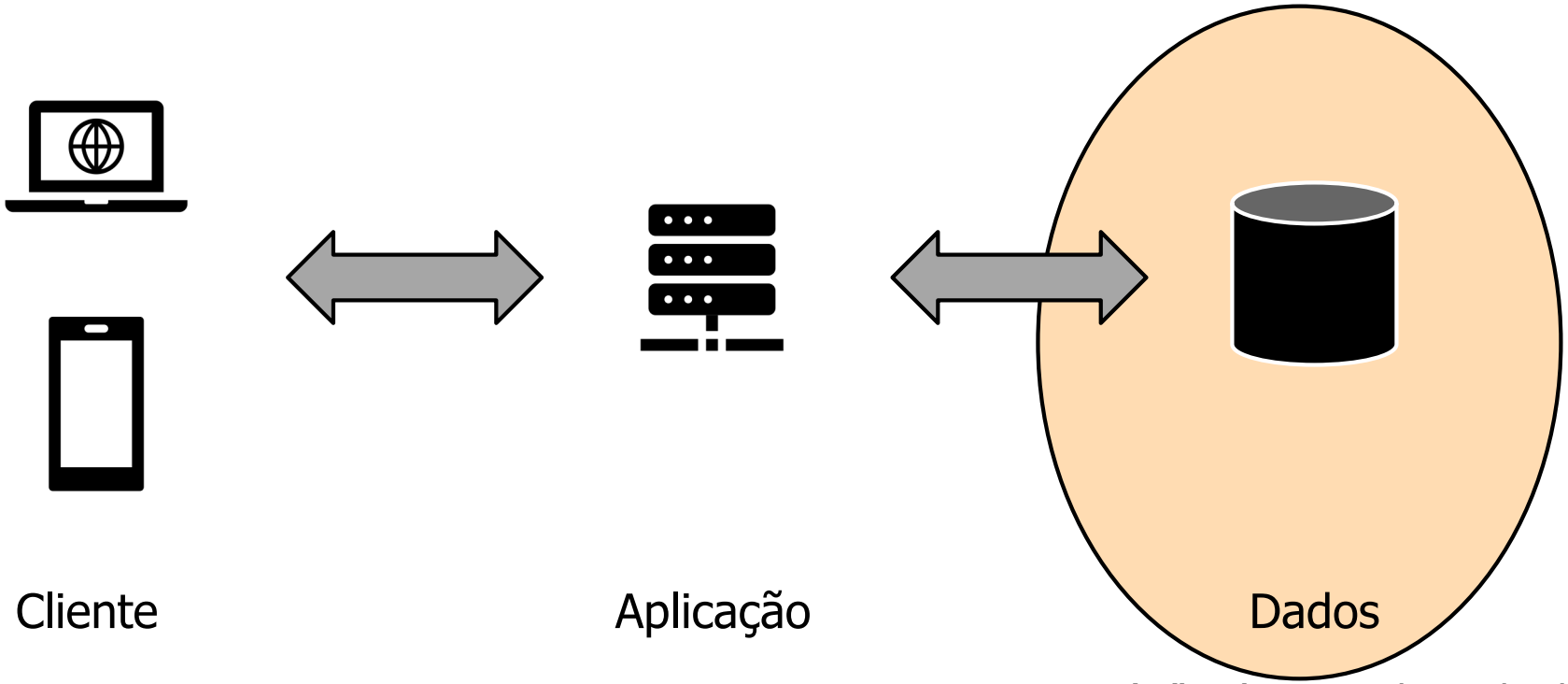
- Servidores REST / SOAP
- Páginas web dinâmicas e estáticas
- Implementado usando servidores aplicacionais
  - E.g.: Tomcat, Wildfly, ASP.NET, etc.



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:

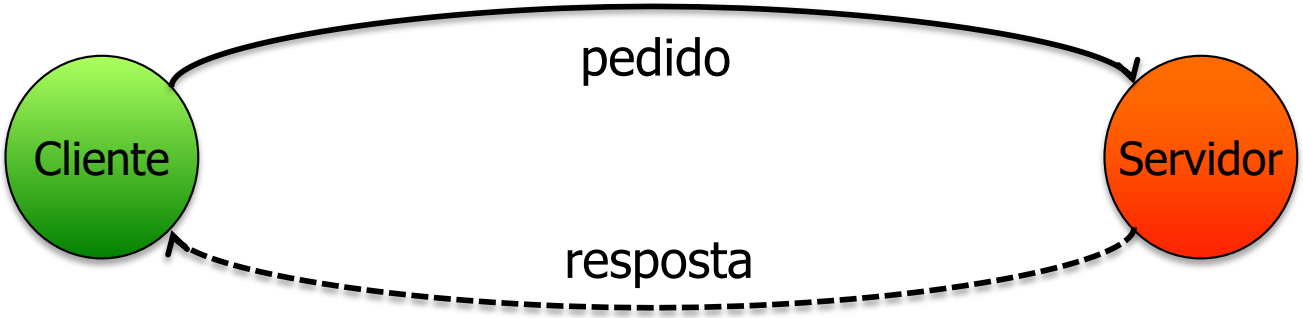
- Requisitos funcionais:
- "tudo relacionado com a função do sistema"
- e.g. Lógica de negócio

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

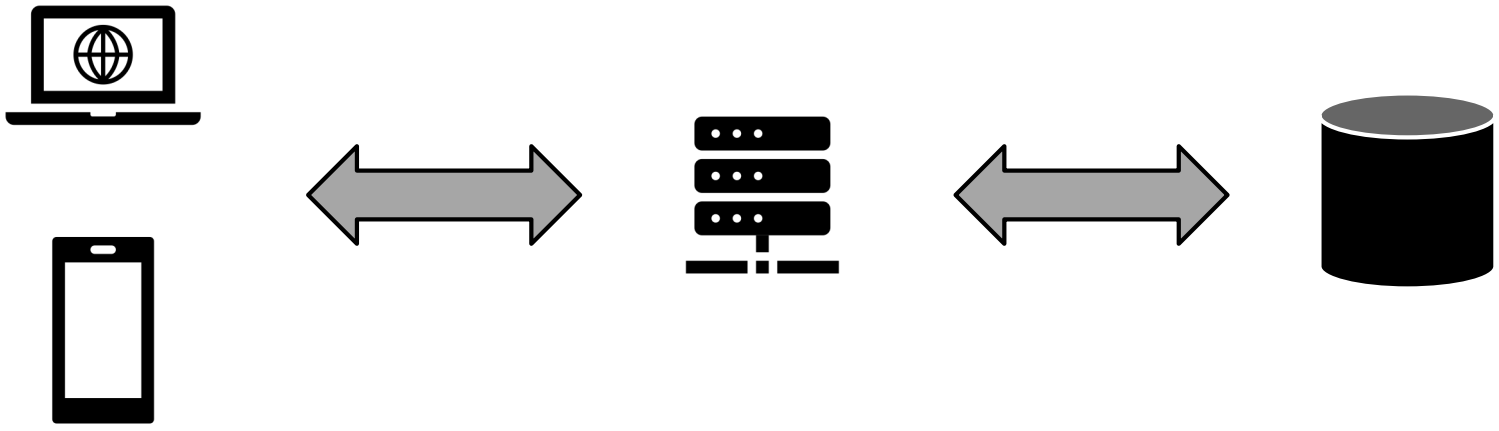
Negativo

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

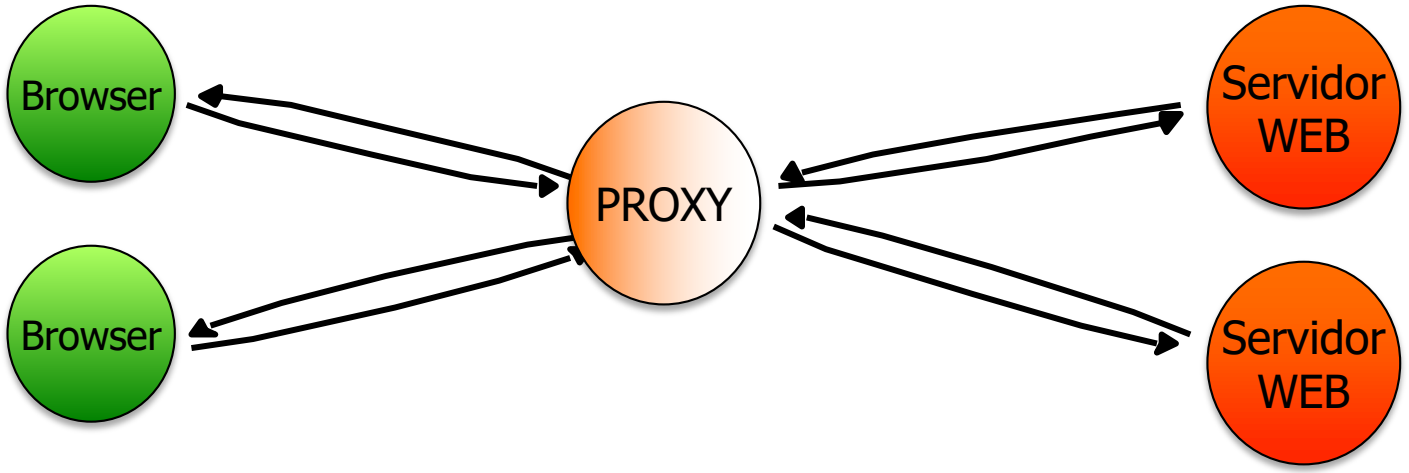
Arquitetura em Camadas e modelo Cliente/Servidor

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

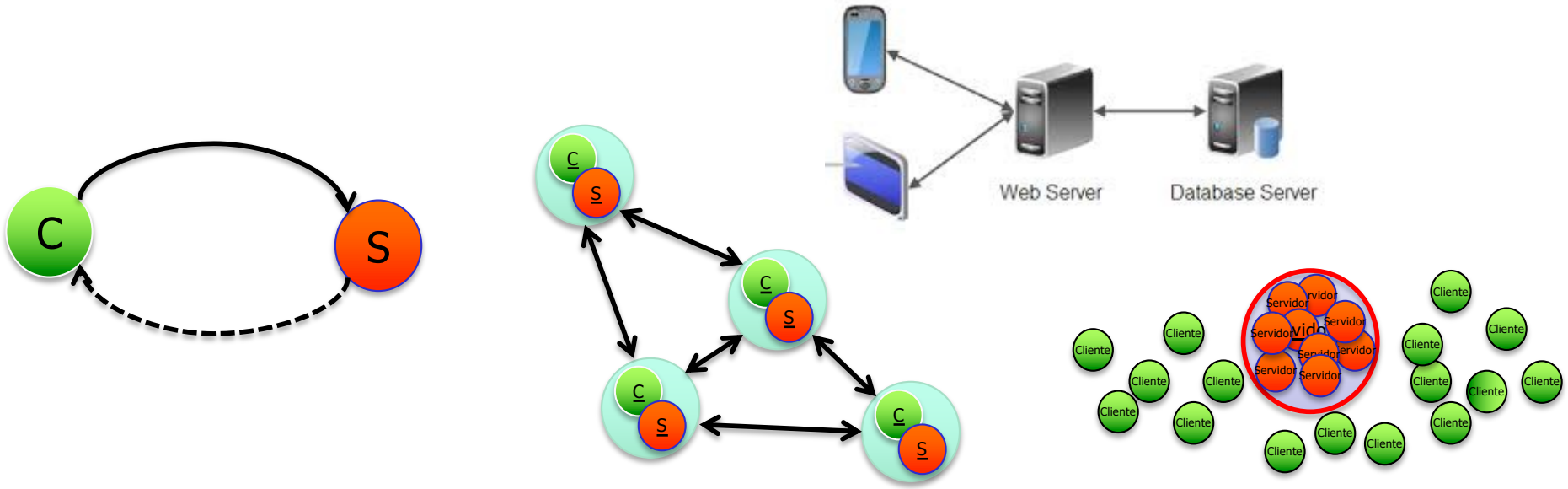
Mascarar falhas do servidor / desconexão

CLIENTE/SERVIDOR: COMPOSIÇÃO

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

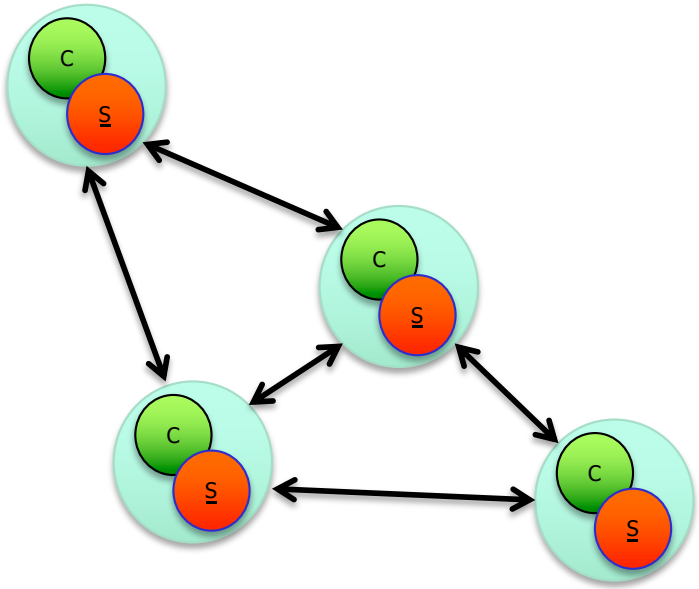
Servidor [particionado, replicado, geo-replicado]

P2P, 3-Tier, etc.



Modelo alternativo para lidar com limitações do modelo cliente/servidor

MODELO *PEER-TO-PEER (P2P)*

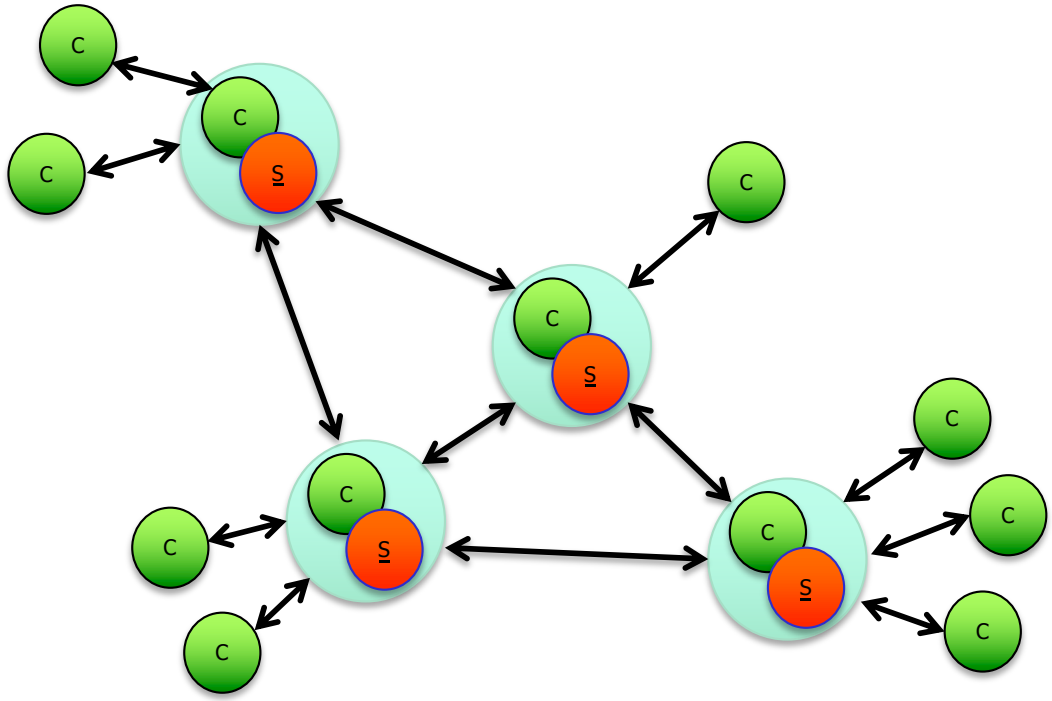


Todos os processos têm funcionalidades semelhantes

- 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

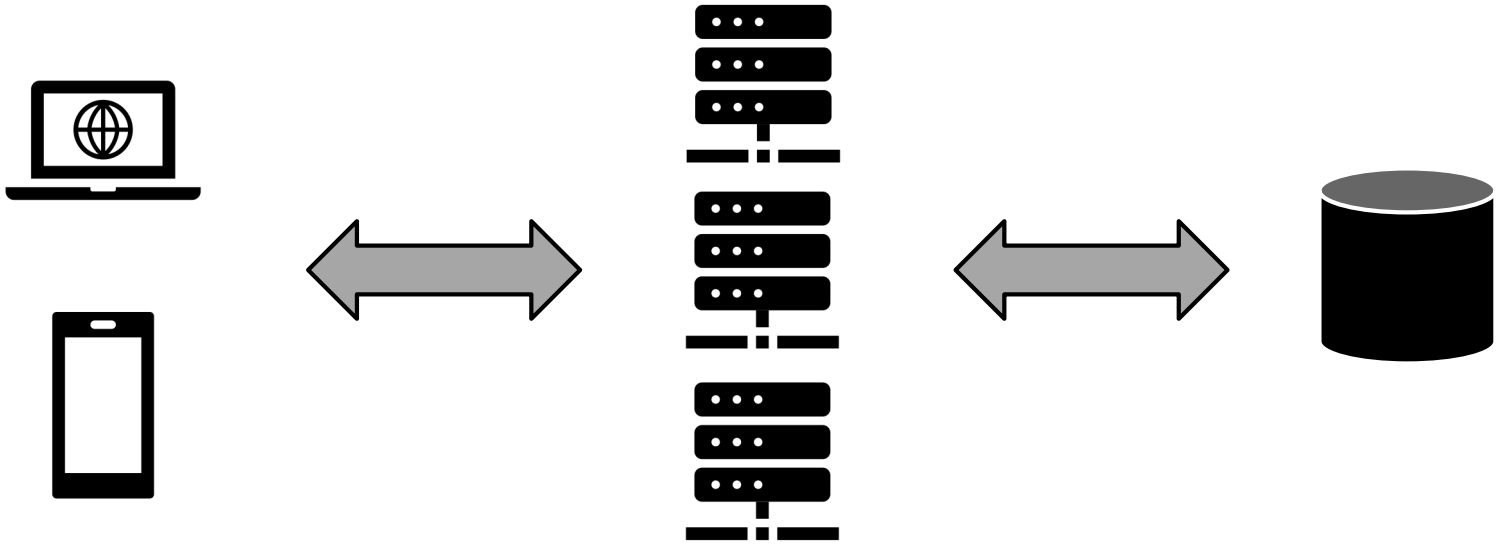
Soluções do lado do servidor

NOTA PRÉVIA

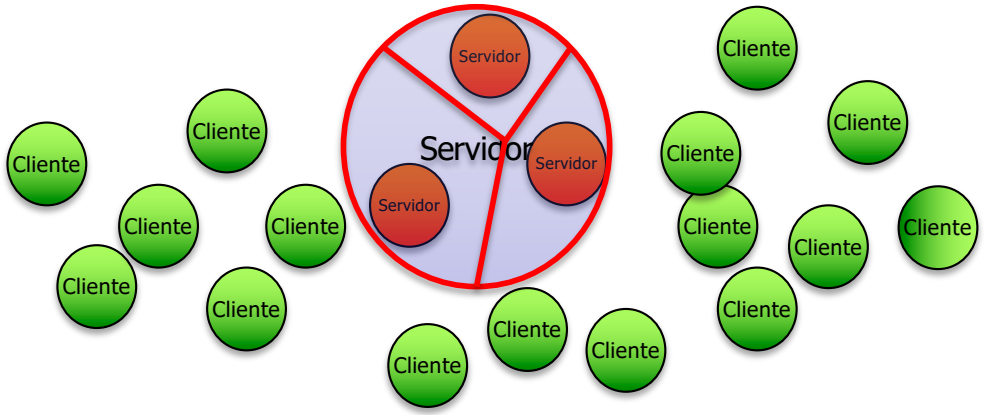
Qual a dificuldade de replicar / particionar um servidor?

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.



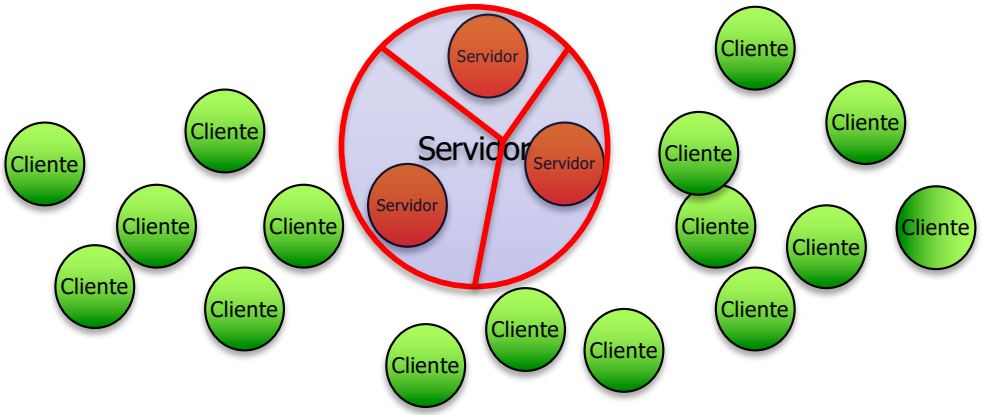
CLIENTE/SERVIDOR PARTICIONADO



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

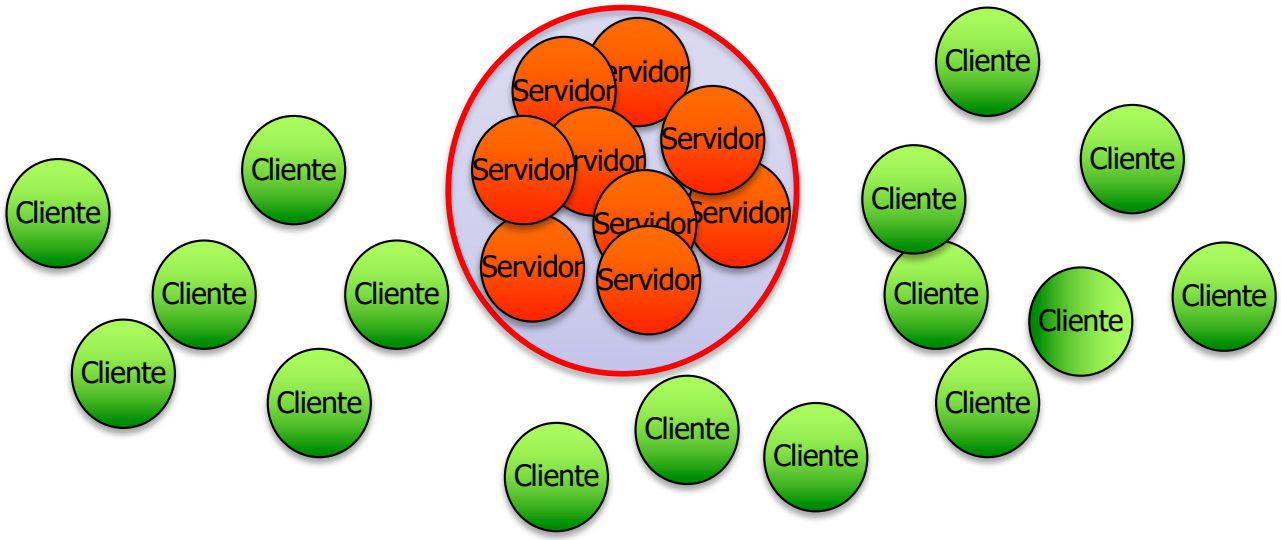
- Falha de um servidor impede acesso aos dados presentes nesse servidor
- Difícil de aplicar em alguns modelos de dados

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

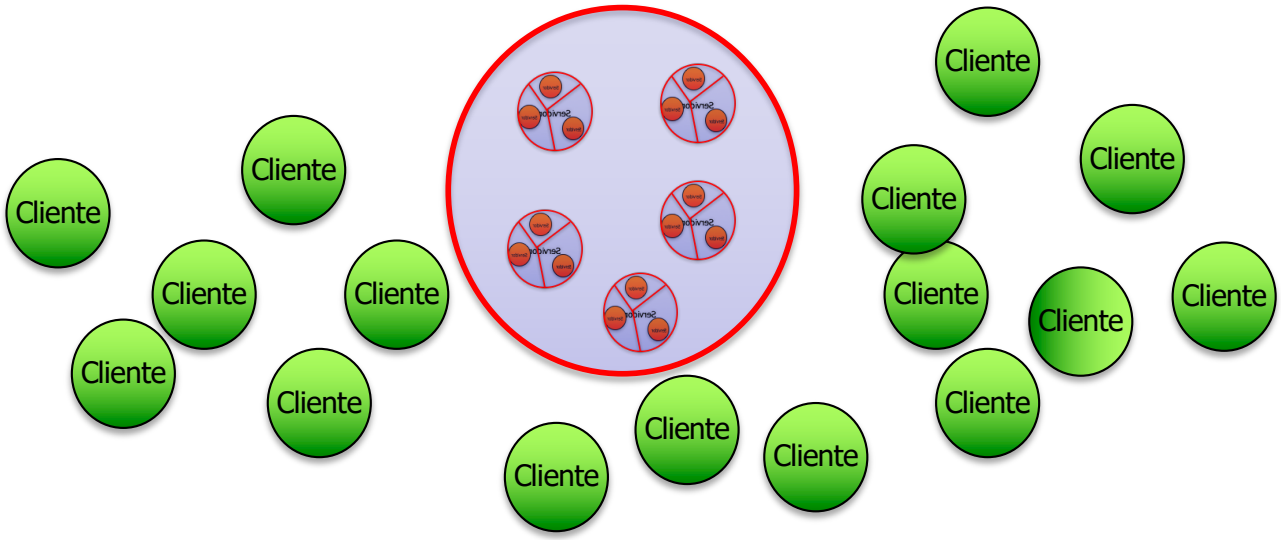
Sendo: *w* : nº de escritas; *r* : nº de leituras; n : nº de réplicas

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)

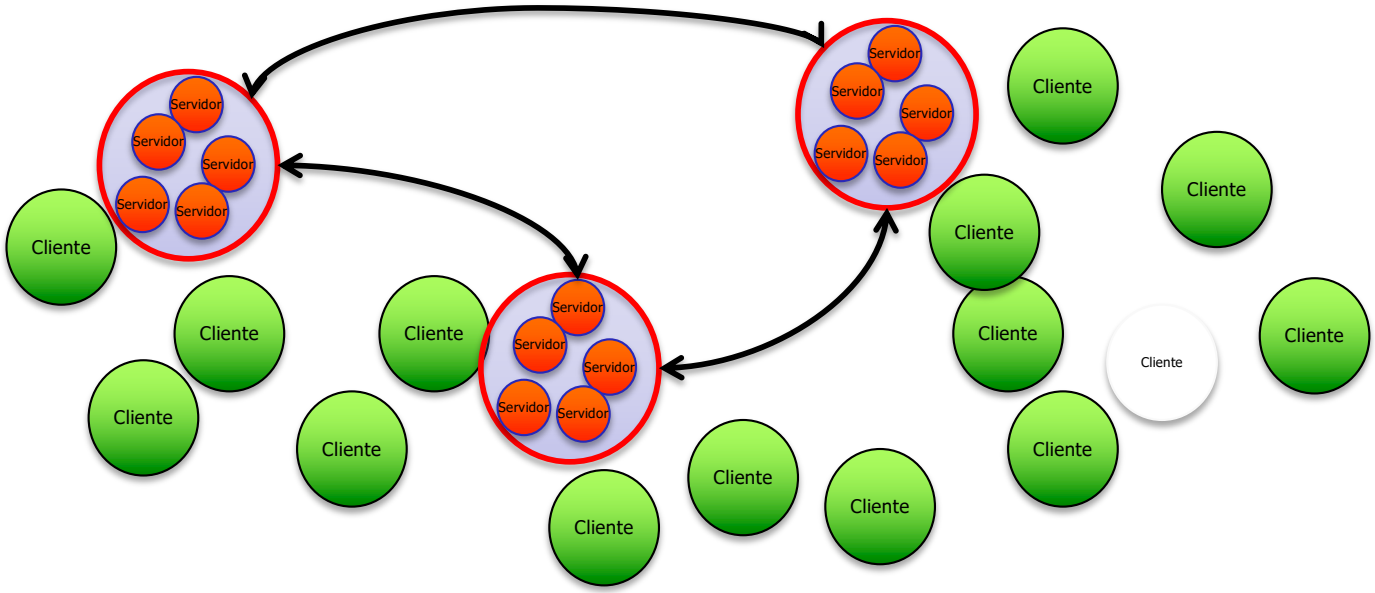
Negativo

- 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

SISTEMAS DISTRIBUÍDOS

Capítulo 2

Arquiteturas e Modelos de Sistemas Distribuídos

... NA AULA ANTERIOR

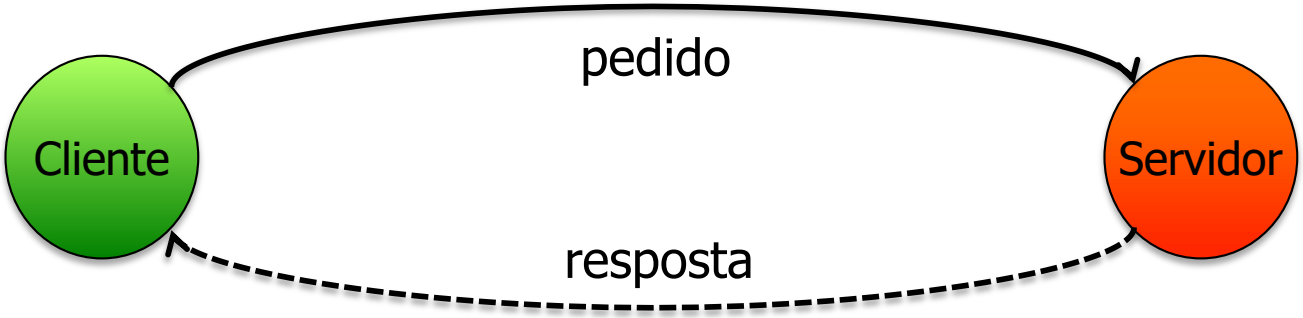
Arquitetura de um sistema distribuído

Estabelece a sua organização em componentes mais simples com funcionalidades/responsabilidades próprias

Define: quais os componentes, o que fazem, onde estão e como interagem entre si

Desenhada: com base num conjunto de requisitos (funcionais e não funcionais)

CLIENTE/SERVIDOR



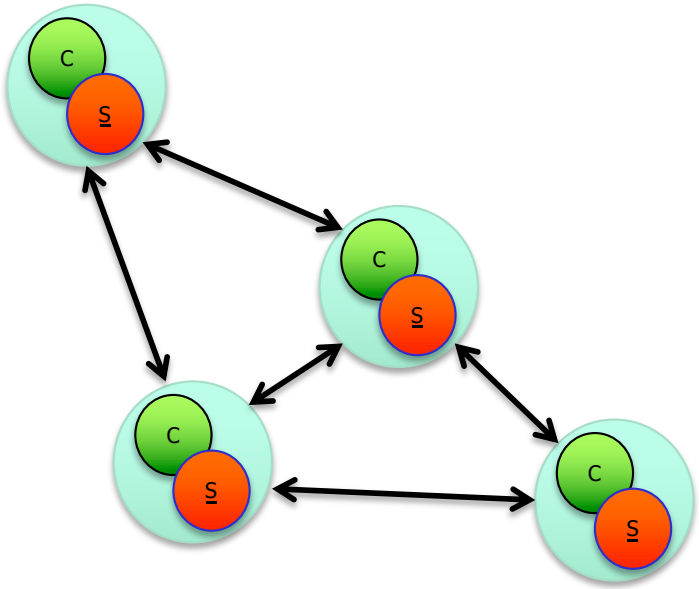
Positivo

- 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*)

MODELO *PEER-TO-PEER (P2P)*



Positivo

- Não existe ponto único de falha
- Melhor potencial de escalabilidade

Negativo

- Interação mais complexa
- Maior número de computadores envolvidos pode colocar questões relativas a heterogeneidade e segurança

... NA AULA ANTERIOR

Variantes do modelo cliente/servidor: servidor

- Cliente/servidor particionado
- Cliente/servidor replicado
- Cliente/servidor particionado + replicado
- Cliente/servidor geo-replicado

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:

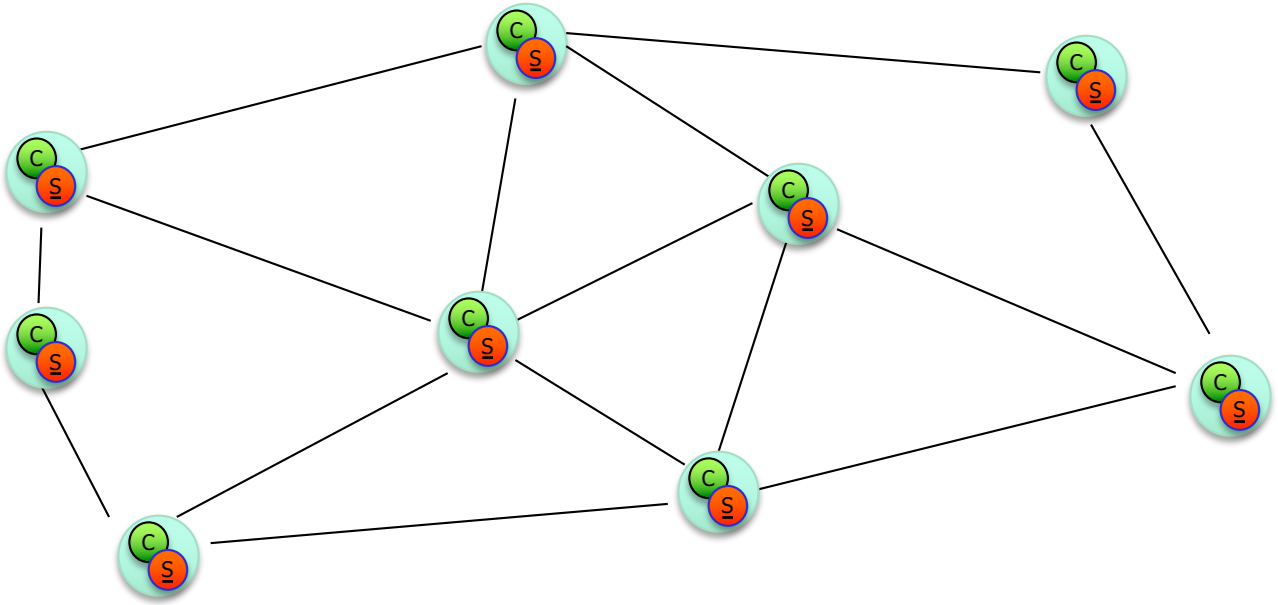
- Permite funcionar *offline,* quando não é possível contactar o servidor (recorrendo a *caching*)
- Permite diminuir a carga do servidor e melhorar o desempenho e a interatividade

Negativo:

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

Variantes do modelo P2P com diferentes propriedades

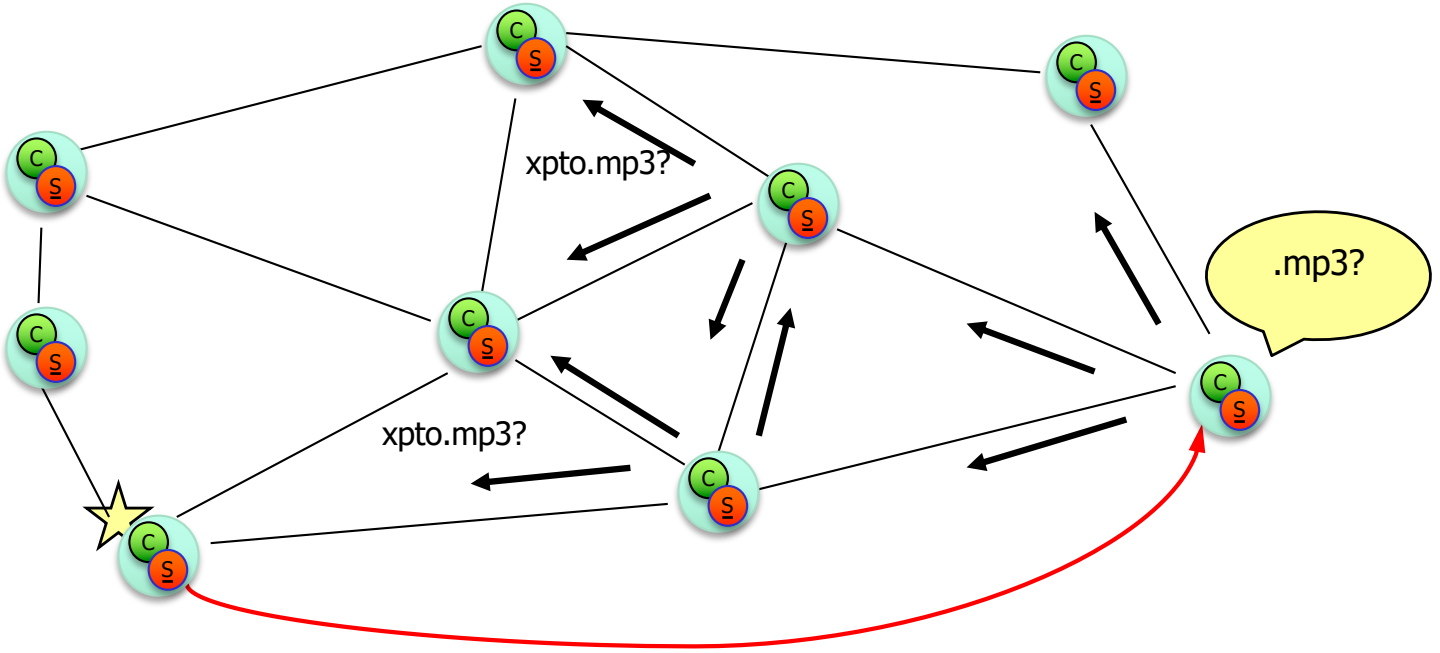
SISTEMA P2P NÃO ESTRUTURADO



As ligações entre os membros são formadas de forma **não-determinista**

E.g. quando se junta à rede, um membro escolhe para vizinhos um pequeno conjunto de contactos (os contactos podem variar durante a execução do sistema e de execução para execução)

SISTEMA P2P NÃO ESTRUTURADO



Positivo:

Simplicidade de construir, robusto ao dinamismo da rede (churn – entrada e saída de nós participantes)

Negativo:

Produz topologias que podem ser difíceis de explorar de forma eficiente

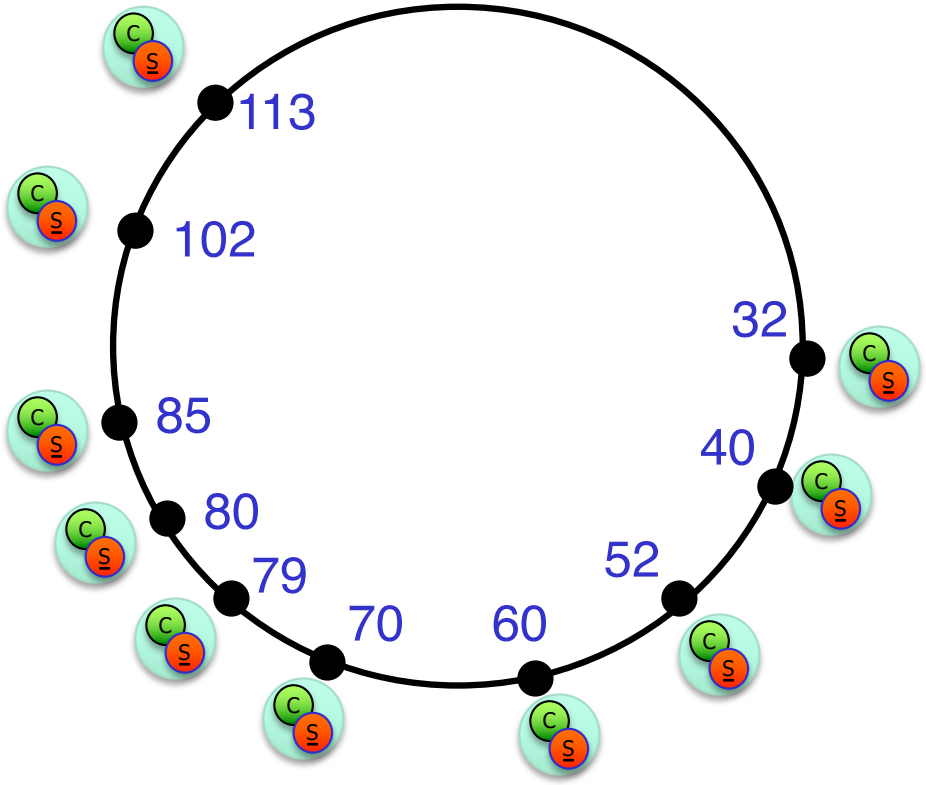
eg., dificuldade em indexar informação / pesquisa pesada (frequentemente por inundação)

SISTEMAS *P2P* ESTRUTURADOS

Os membros do sistema comunicam de acordo com uma organização definida de forma determinista com base num **endereço lógico**

A topologia é induzida por uma relação (matemática) entre os endereços lógicos.

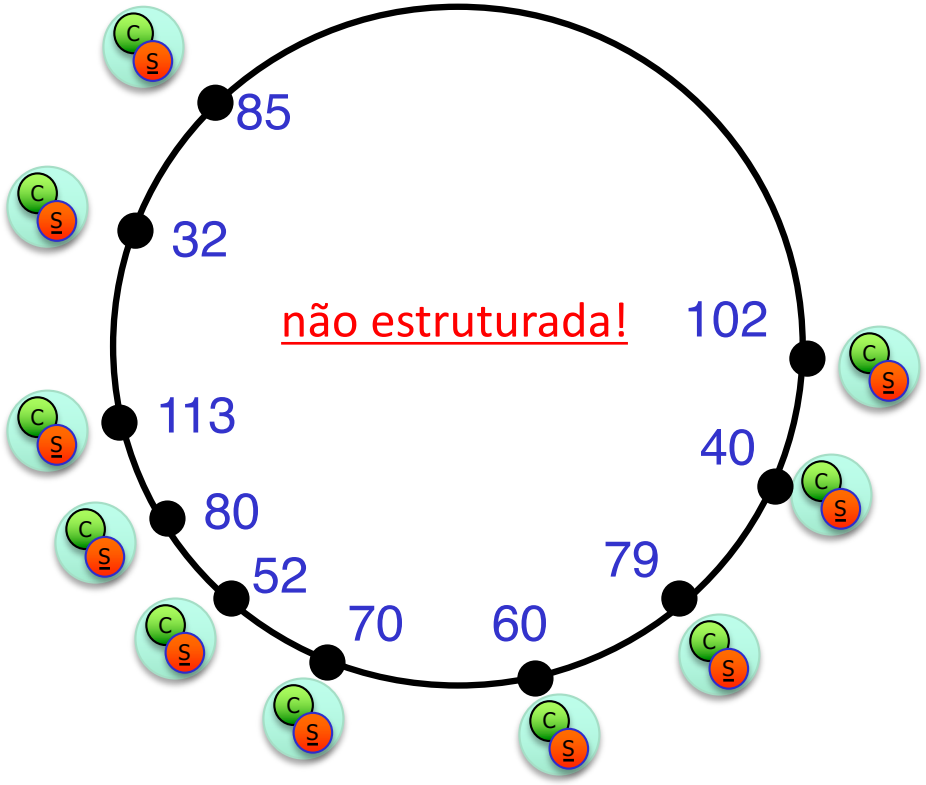
Existem topologias para todos os gostos...



SISTEMAS *P2P* ESTRUTURADOS

Importante: A topologia (as relações de vizinhança) dos nós têm que ser induzidas pelos identificadores lógicos

Os nós podem estar organizados num anel, por exemplo, e não formar um sistema P2P estruturado...

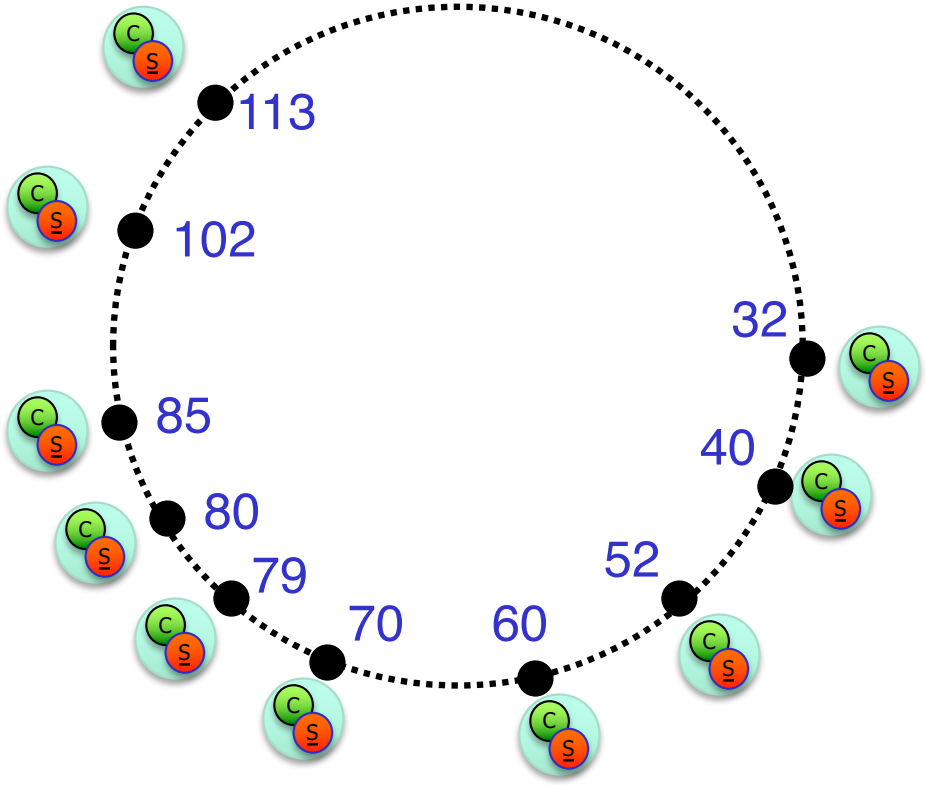


Analogia: árvore binária vs. árvore binária de pesquisa

quanto custa pesquisar um valor no primeiro caso vs. no segundo?

SISTEMAS *P2P* ESTRUTURADOS

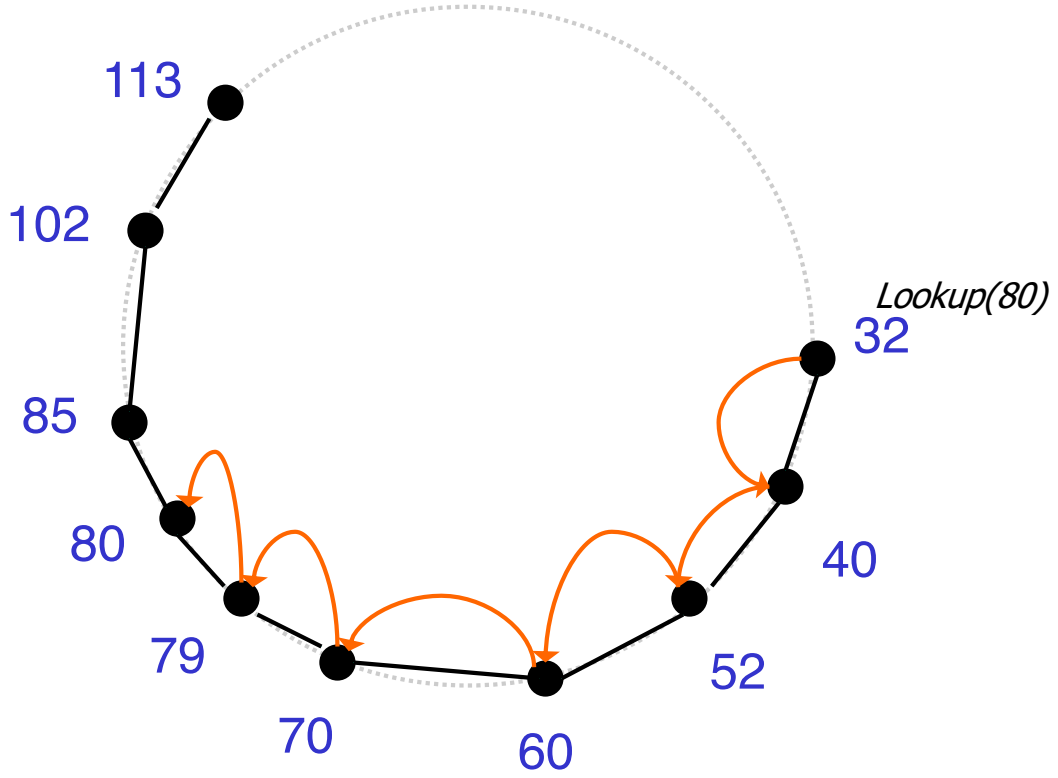
Encaminhamento e pesquisas por identificador O(log N)?



SISTEMAS *P2P* ESTRUTURADOS

Encaminhamento e pesquisas por identificador O(log N)?

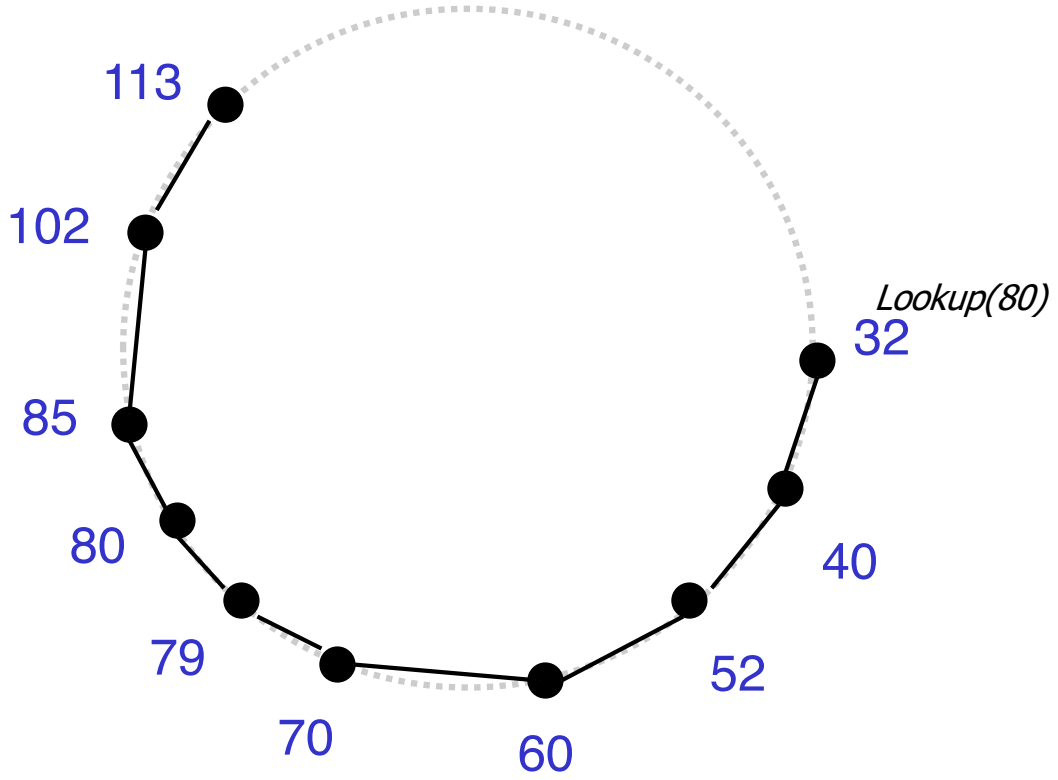
Seguir o sucessor não é solução; tem custo O(N)



SISTEMAS *P2P* ESTRUTURADOS

Encaminhamento e pesquisas por identificador O(log N)?

Em cada passo é necessário reduzir o espaço de pesquisa para metade...

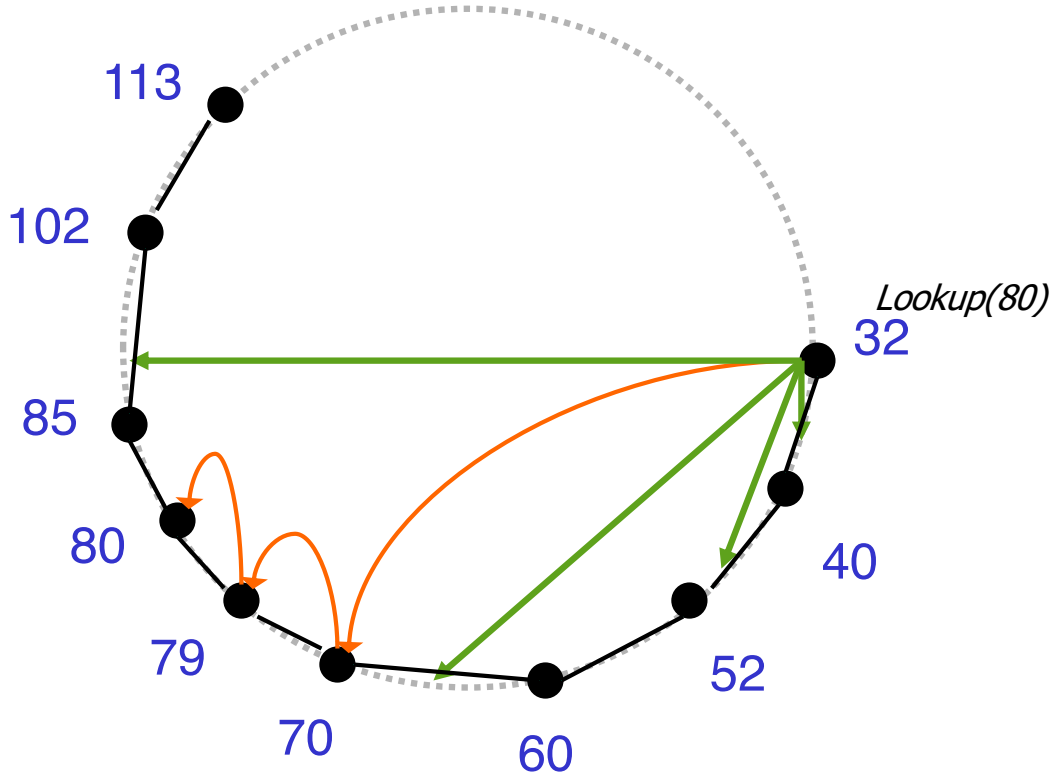


SISTEMAS *P2P* ESTRUTURADOS

Encaminhamento e pesquisas por identificador O(log N)?

Em cada passo é necessário reduzir o espaço de pesquisa para metade...

Ideia: cada nó liga-se a nós noutros pontos da topologia P2P e usa essas ligações como atalhos: ex: O(log N) vizinhos



SISTEMAS *P2P* ESTRUTURADOS

<Simulação Sistema Chord, circa 2001>

Características:

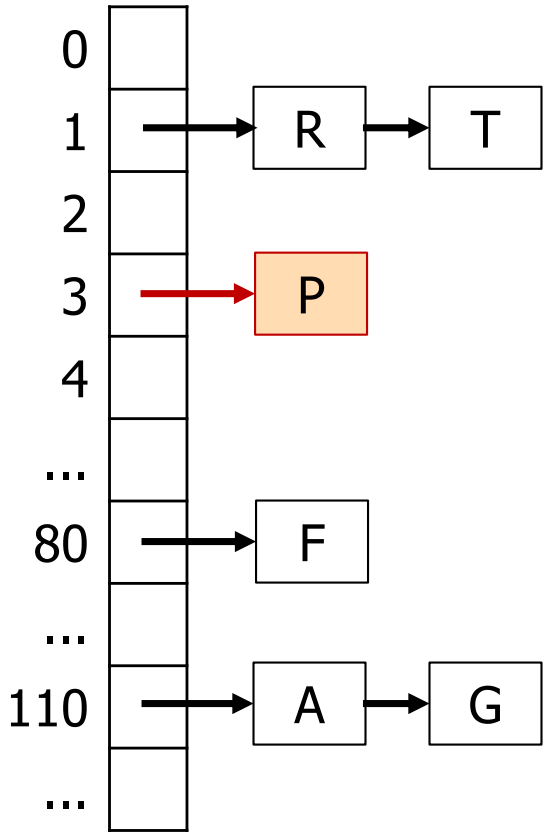
- Identificadores de 128 bits
- Tabelas de vizinhança com O(log N) peers
- Encaminhamento e Pesquisa por identificador em O(log N) passos

SISTEMAS *P2P* ESTRUTURADOS E DADOS

Um sistema P2P estruturado permite-nos descobrir um nó eficientemente (em log(n)).

Como é que podemos usar esta funcionalidade para guardar informação?

DISTRIBUTED HASH TABLE (DHT)

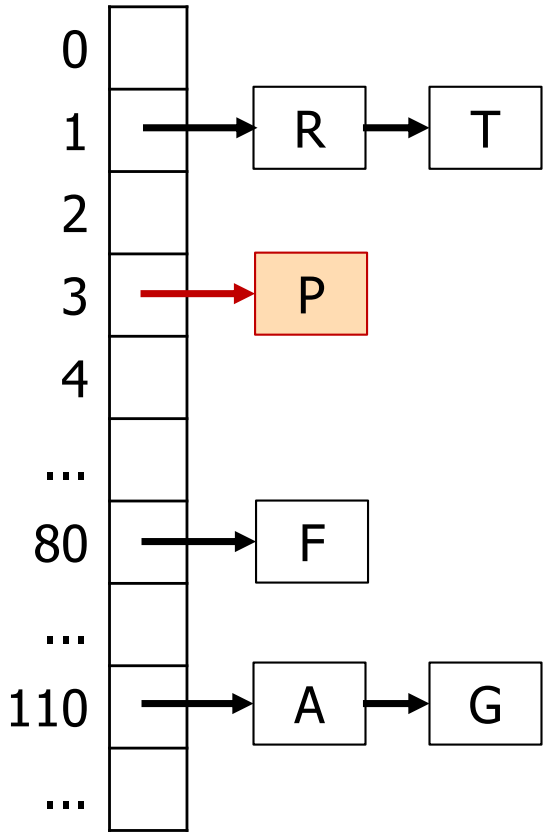


Numa **tabela de dispersão**, quando se quer guardar um valor, v, calcula-se o hash(v) e guarda-se o valor nessa posição.

E.g. para inserir P, calcula-se hash(P) = 3, e insere-se o valor na posição respetiva.

Nota: Em geral P é um par (chave,valor), sendo que neste caso só se faz hash(chave).

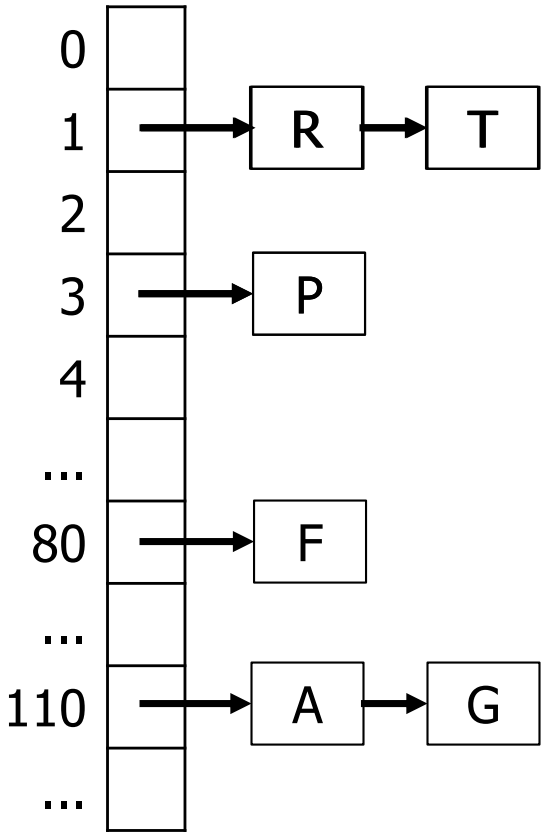
DISTRIBUTED HASH TABLE (DHT)

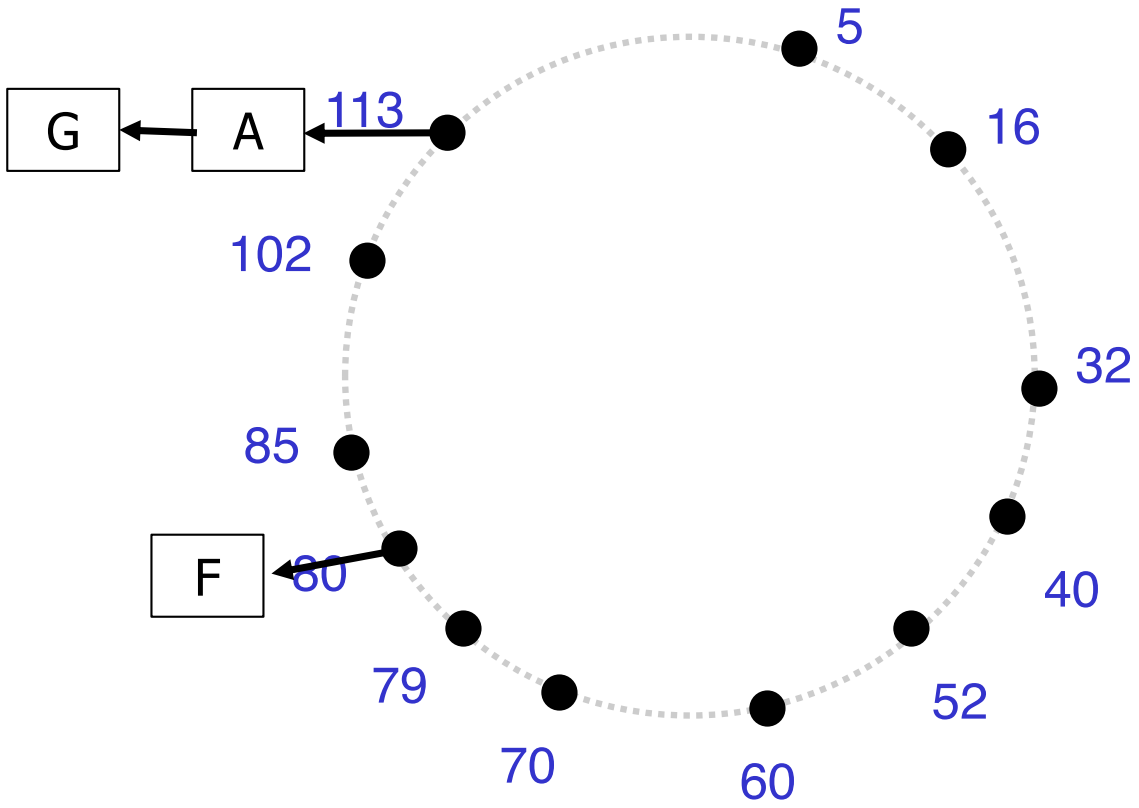


Numa **tabela de dispersão distribuída** (distributed hash table, DHT), vai-se usar a mesma ideia: vamos guardar ficheiros/blocos no servidor que tiver o identificador igual ao hash(chave).

DISTRIBUTED HASH TABLE (DHT)

Os dados que numa tabela de dispersão ficaria na posição n, na DHT ficam no nó com o identificador igual ou superior a n.



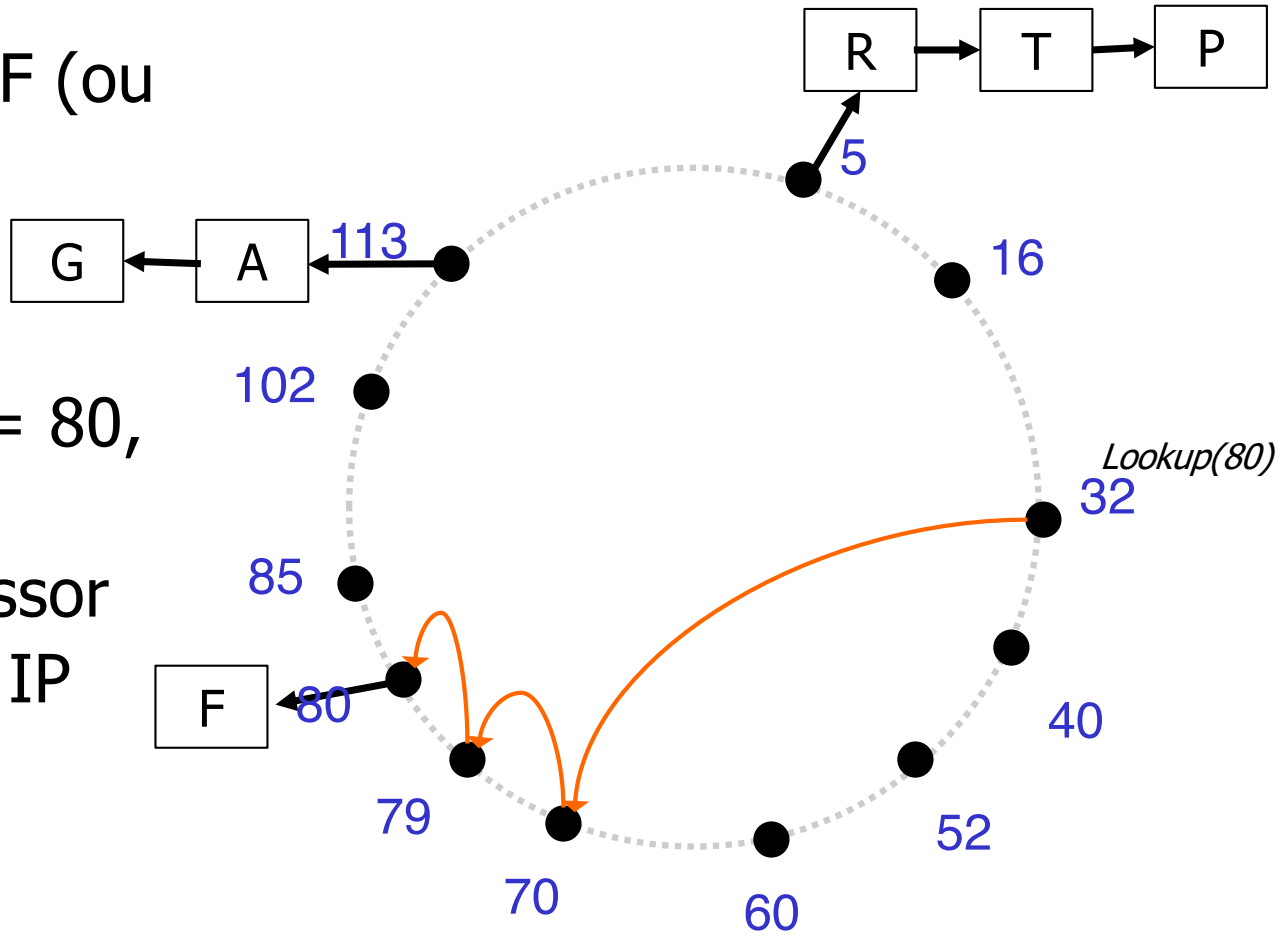


DISTRIBUTED HASH TABLE (DHT)

Para ler o ficheiro/bloco F (ou com chave F):

1. **lookup(F)->IP_{80}**

Lookup calcula hash(F) = 80, e usa sistema P2P para procurar nó 80 (ou sucessor de 80). Sistema devolve IP do nó 80.



DISTRIBUTED HASH TABLE (DHT)

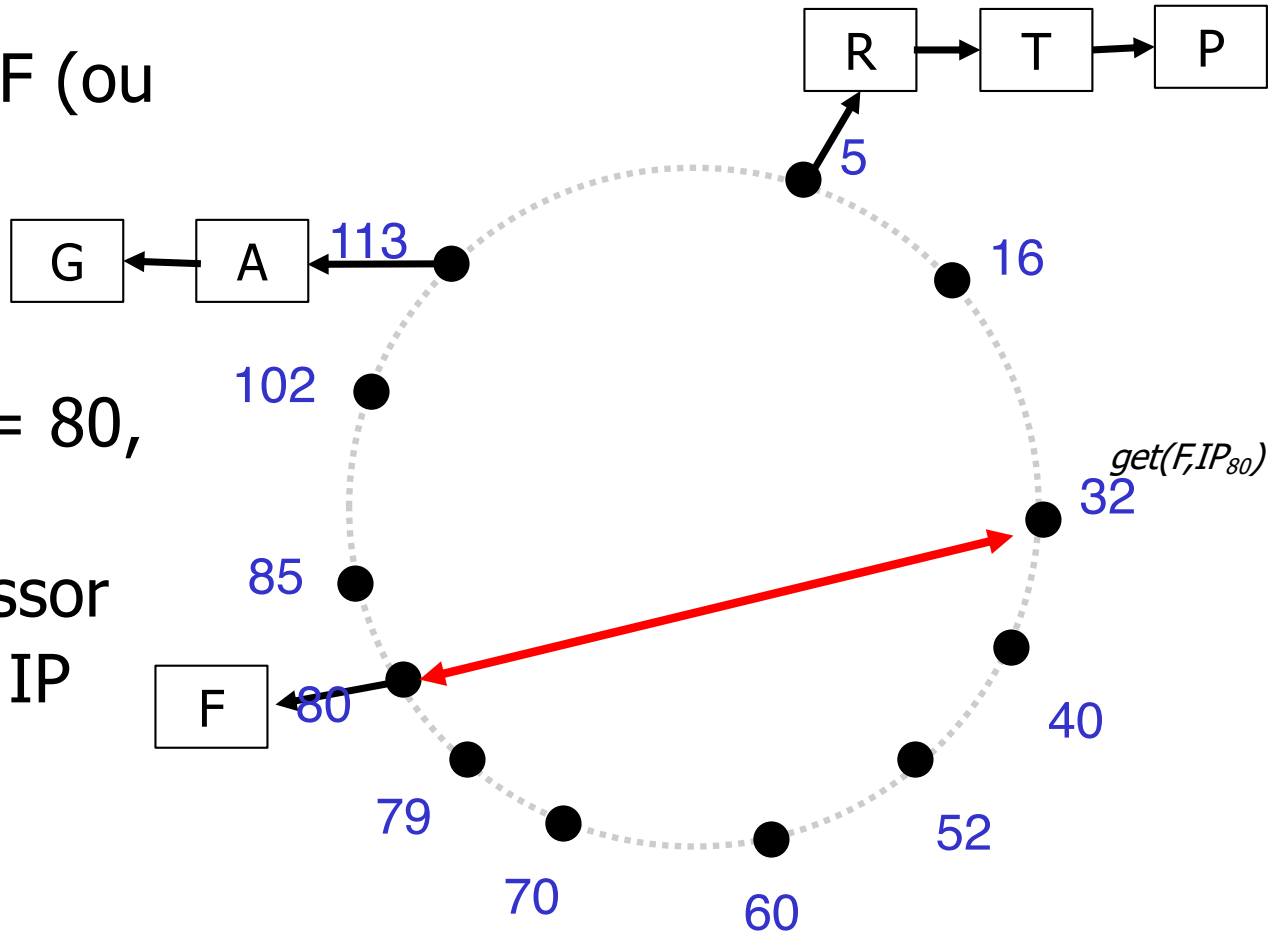
Para ler o ficheiro/bloco F (ou com chave F):

1. **lookup(F)->IP_{80}**

Lookup calcula hash(F) = 80, e usa sistema P2P para procurar nó 80 (ou sucessor de 80). Sistema devolve IP do nó 80.

2. **get(F,IP_{80})->valor**

Contacta-se diretamente o nó para obter os dados.

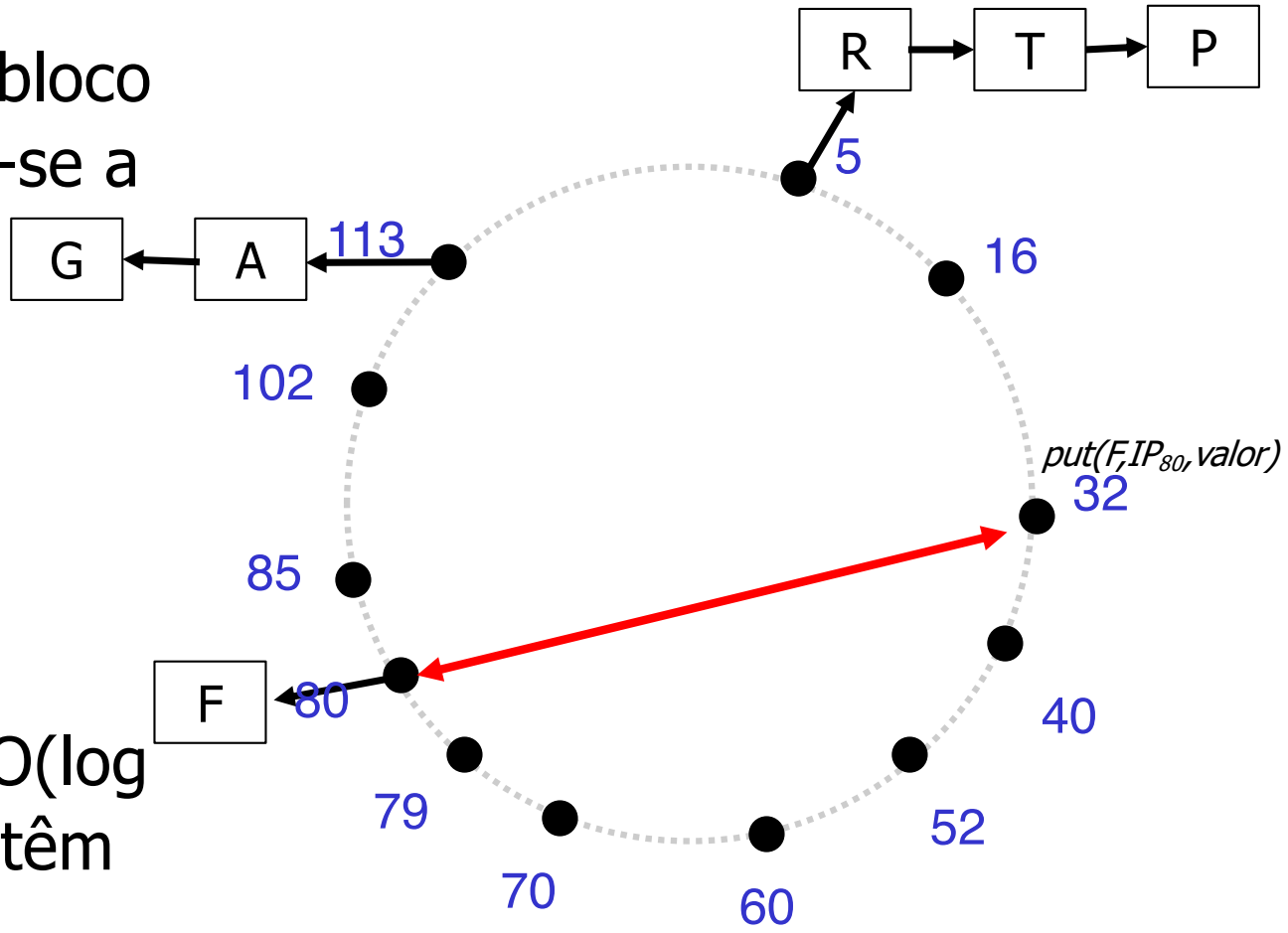


DISTRIBUTED HASH TABLE (DHT)

Para escrever o ficheiro/bloco F (ou com chave F), usa-se a mesma aproximação.

1. **lookup(F)->IP_{80}**
2. **put(F,IP_{80}, valor)**

Lookup(key) com custo O(log N), as outras operações têm custo constante (independente de N)



DHTs: CARACTERÍSTICAS

Identifica-se a informação usando uma função de *hash*

identificador = hash(info)

Cada nó fica responsável por um conjunto de identificadores (de forma determinista)

- e.g. cada nó usa um identificador único, gerado aleatoriamente, ficando responsável por manter a informação com os identificadores mais próximos do seu

Aspetos importantes...

- Pesquisa?
- Distribuição da informação?
- Replicação da informação?

DHTs: CARACTERÍSTICAS

Identifica-se a informação usando uma função de *hash*

identificador = hash(info)

Cada nó fica responsável por um conjunto de identificadores (de forma determinista)

- e.g. cada nó usa um identificador único, gerado aleatoriamente, ficando responsável por manter a informação com os identificadores mais próximos do seu

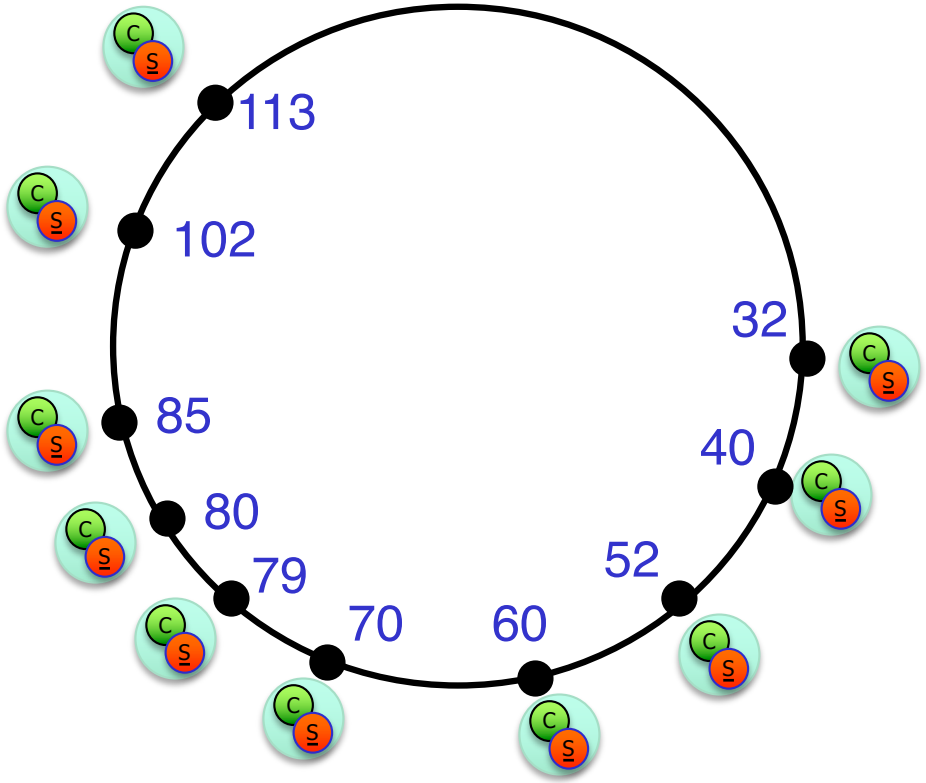
Aspetos importantes...

- Pesquisa? – pesquisa por identificador deve ser eficiente e suportada por todos os nós.
- Distribuição da informação? – deve ser uniforme por todos os nós
- Replicação da informação? – a entrada e saída contínua de nós obriga a manter a informação replicada nos nós certos e em número adequado.

SISTEMAS *P2P* ESTRUTURADOS

Positivo

Boa latência/escalabilidade - Conhecem-se topologias com encaminhamento/pesquisa com custo ***O(log n)***, com ***n*** nós no sistema, por exemplo.



Negativo

Maior complexidade

É necessário gastar recursos para manter a topologia correta face às entradas e saídas dos nós (*churn*)

SISTEMAS DISTRIBUÍDOS

Capítulo 2

Arquiteturas e Modelos de Sistemas Distribuídos

NA ÚLTIMA AULA...

Variantes do modelo cliente/servidor: cliente

Variantes dos modelos peer-to-peer

NA AULA DE HOJE....

Combinações modelo cliente/servidor e modelos peer-to-peer

- E.g. BitTorrent, P2p hierárquico

Modelos fundamentais de um sistema distribuído

Modelo de interação

Modelo de falhas

Modelo de segurança

Combinando os dois modelos (P2P e C/S) pode-se ter o melhor dos dois

COMBINAÇÃO DOS MODELOS C/S E P2P

Cliente + (Serviço) P2P

- Um sistema P2P pode disponibilizar um serviço a outros processos (clientes) que não pertencem ao sistema P2P

Propriedades:

- Permite limitar o número de processos que fazem parte do sistema P2P

COMBINAÇÃO DOS MODELOS C/S E P2P

Cliente/servidor + P2P

- Serviço disponibilizado por um sistema pode ser dividido em várias funcionalidades, sendo umas fornecidas por um sistema cliente/servidor e outras por um sistema P2P.

- O sistema cliente/servidor pode, por exemplo, servir como serviço de diretório, suportar pesquisas eficientes, etc.
  - e.g. BitTorrent

Propriedades:

- Permite combinar as vantagens de ambos os sistemas

- Terá contras?
- Ter também as desvantagens de ambos...

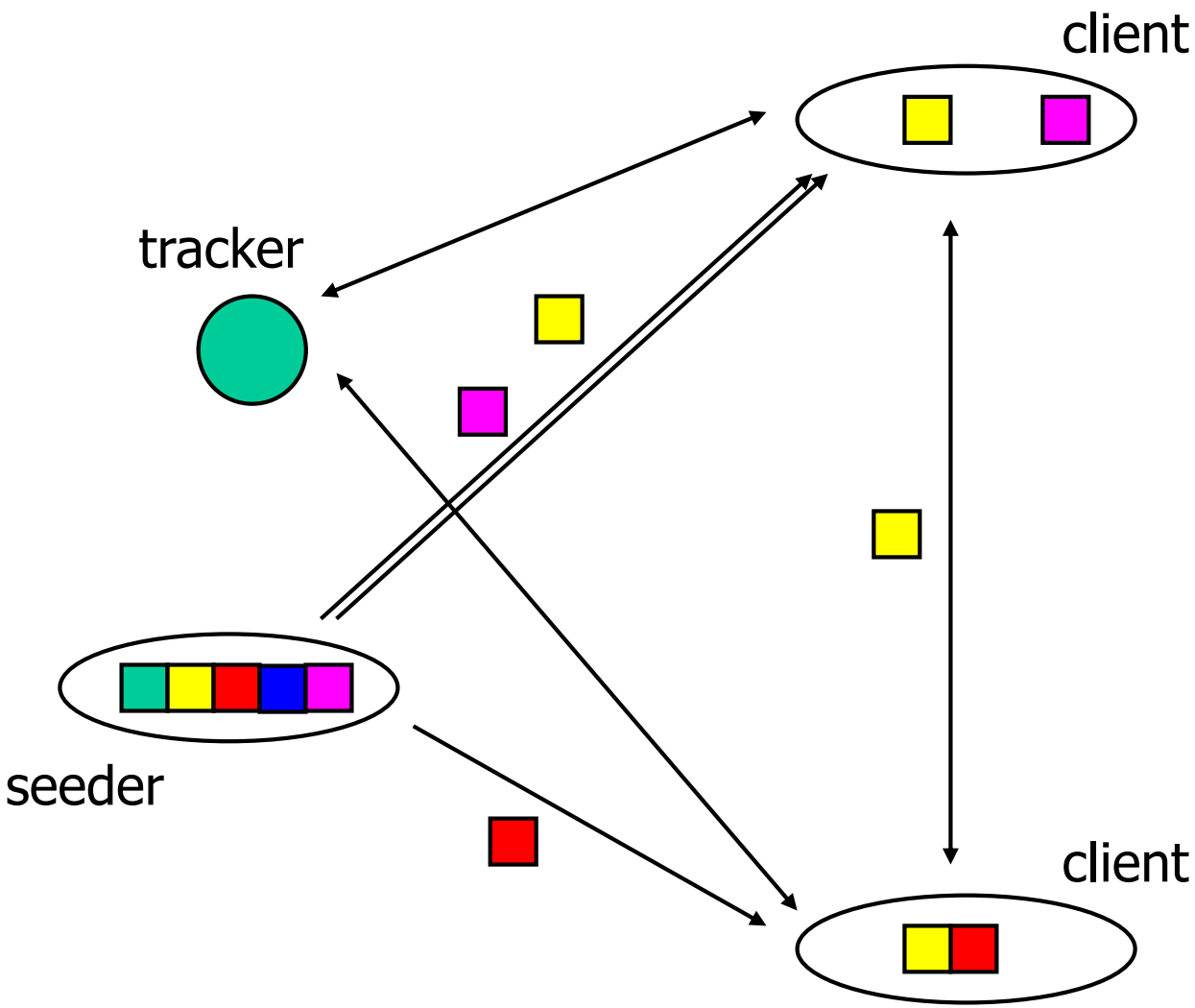
BitTorrent-versão original (simplificado)

.torrent contém informação sobre ficheiro a ser partilhado, incluindo: url do tracker, hash seguro de cada bloco do ficheiro (SHA-1)

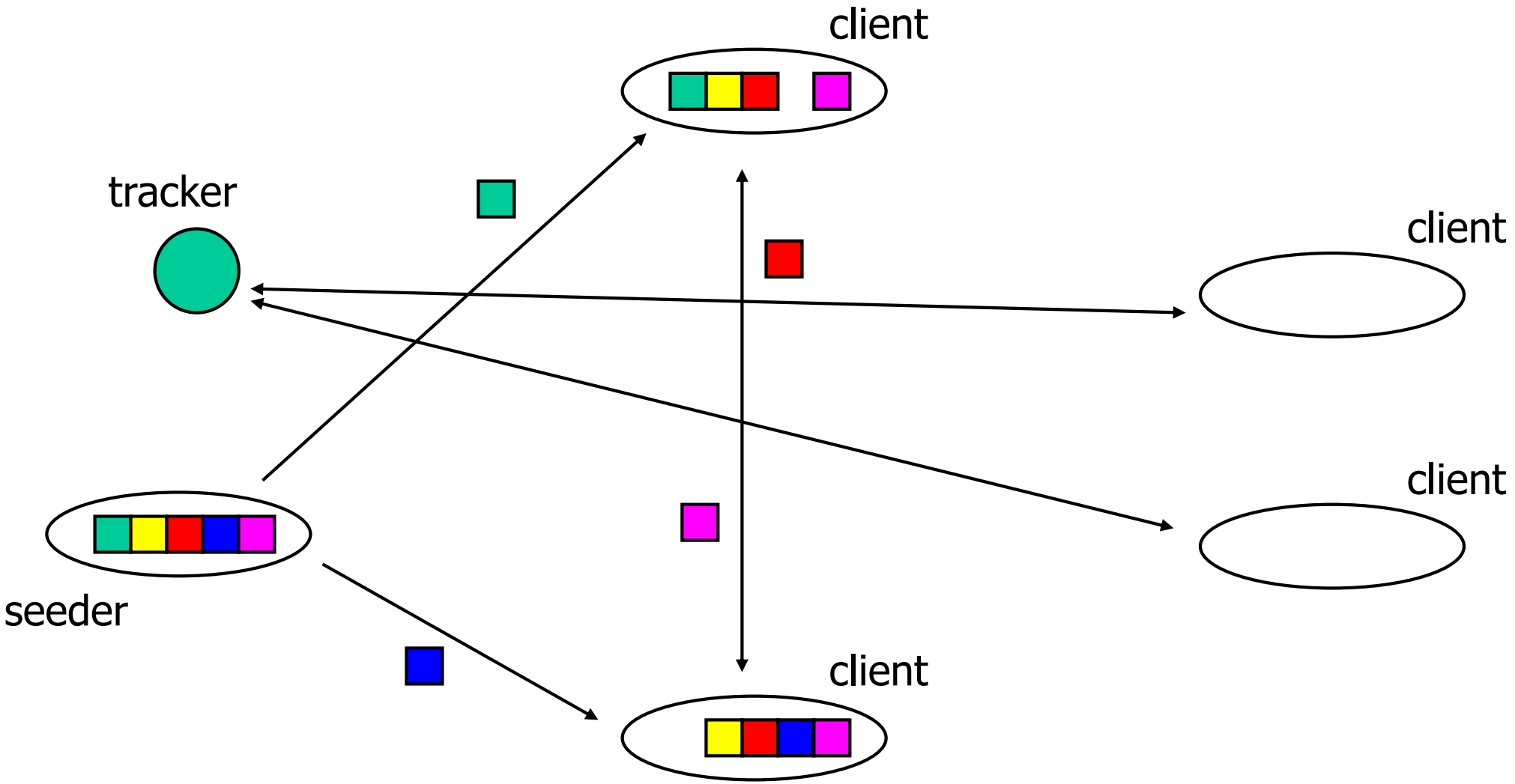
Arquitetura BitTorrent:

- C/S
  - **Tracker**: servidor centralizado HTTP que serve o ficheiro .torrent e a lista dos *peers* que estão a descarregar o ficheiro
  - ***Peers***: como clientes, acedem ao *tracker* para obter a lista de outros peers a descarregar o ficheiro
- P2P
  - ***Peers*** comunicam entre si para trocar blocos do ficheiro
    - Toma-lá dá-cá (*tit-for-tat*): peer só envia bloco em troca de outro bloco
    - *Peer* envia alguns blocos a outros peers (sem contrapartida - aleatoriamente)
    - *Peer* descarrega blocos aleatoriamente
    - (Qual a motivação de cada uma destas propriedades?)

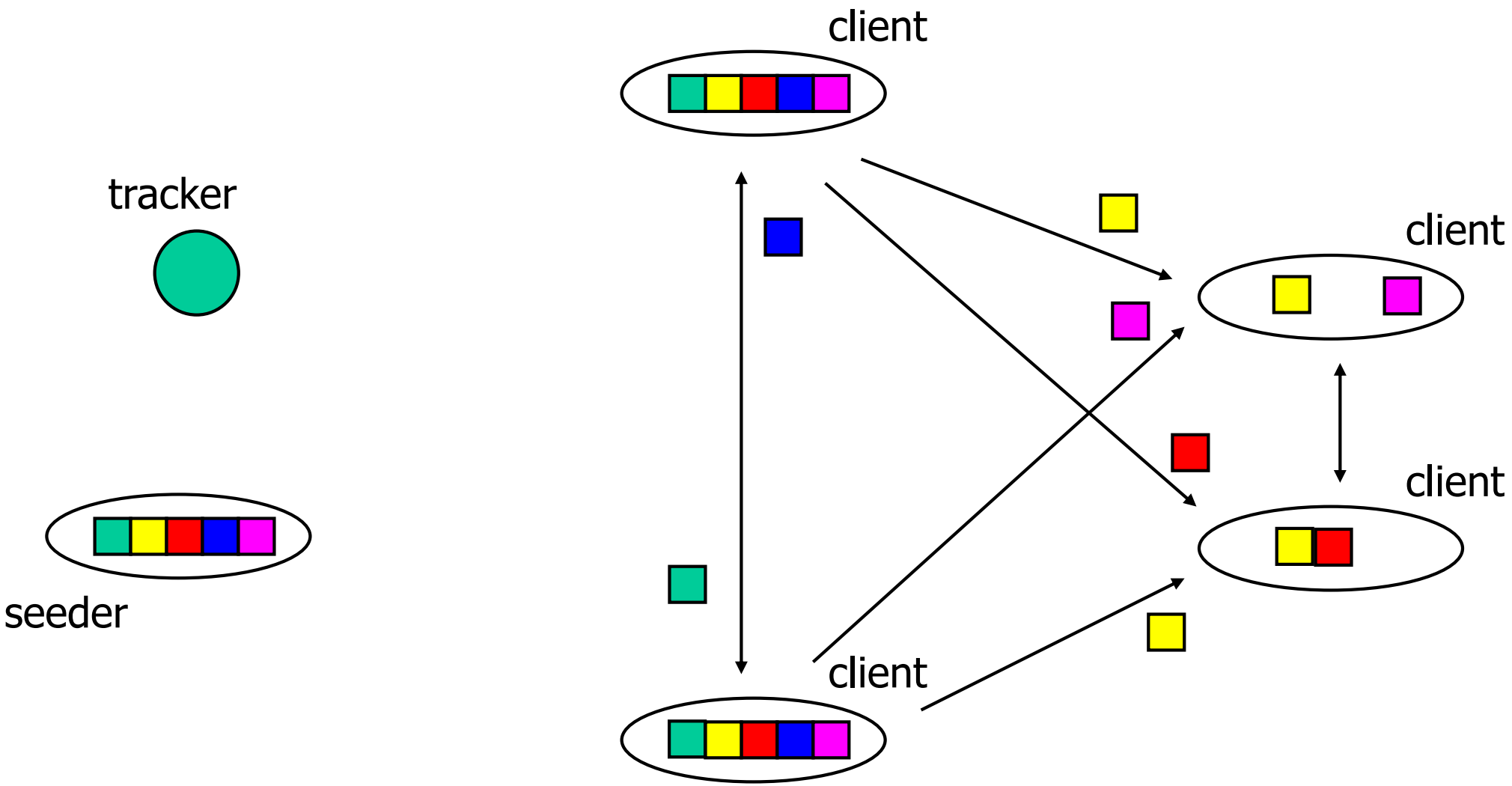
BITTORRENT



BITTORRENT



BITTORRENT



BitTorrent DHT

BitTorrent tem a possibilidade de funcionar sem trackers

Neste caso, os peers formam uma DHT

- Cada nó gera um identificador único
- Cada *torrent* tem um identificador único

- Informação sobre quais os nós que estão a partilhar/descarregar um ficheiro/torrent é armazenada nos nós com identificadores mais próximos ao identificador do torrent

- Nota: baseado no sistema Kademlia (circa 2002)

AINDA OUTROS EXEMPLOS...

Sistemas peer-to-peer hierárquicos

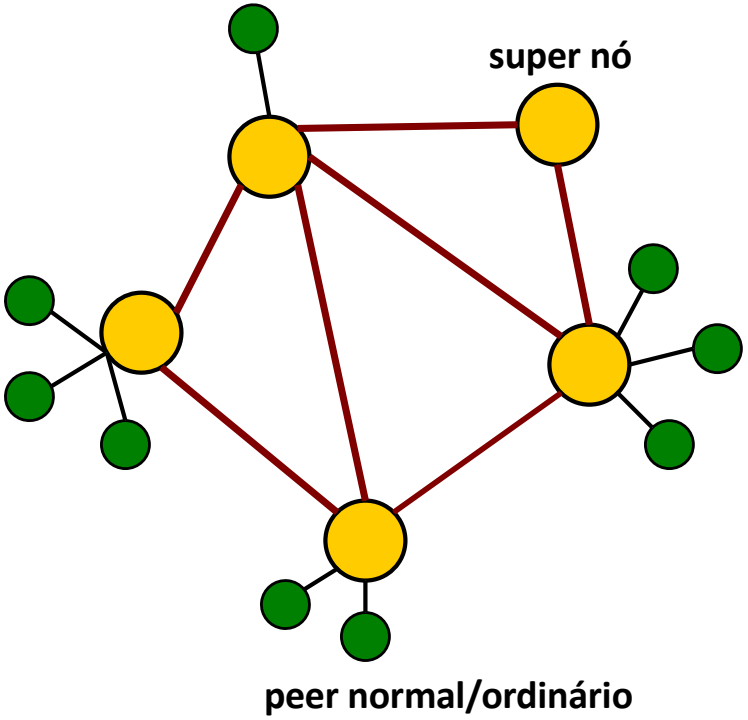
- Subconjunto de super-nós que se agrupam como num sistema peer-to-peer
- Nós ligam-se a um super-nó

...

KaZaa / Skype (original)

Arquitetura:

- **Super-nós** (SN): nós com maior capacidade tornam-se super-nós
  - Cada SN tem 60-150 nós ligados
  - Cada SN liga-se a outros 30-50 SN
- **Peers** (ordinary node – ON)
  - ON liga-se a um dos SNs que conhece (após se ter ligado, atualiza lista de SNs)



Pesquisa:

- ON contacta o seu SN
- SNs propagam pedidos entre si

Transferência de ficheiros:

- Directamente entre **peers**

Na prática, tendem a existir mais componentes num sistema distribuído

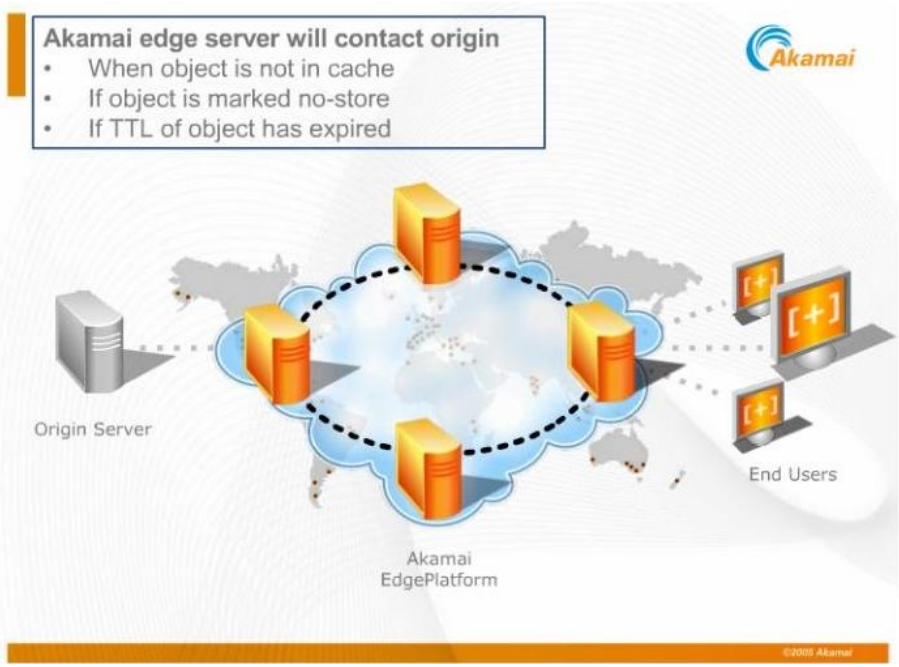
EDGE-SERVER

Sistemas edge-server

- Existem servidores colocados nos ISP para responder a pedidos
  - e.g., content-distribution networks (CDNs)
- Propriedades
  - Menor latência, filtragem, distribuição de carga, etc.

Suportam **distribuição de conteúdos estáticos** de grande volume e/ou popularidade:

eg., streaming: Youtube, Netflix



src: https://clickmotive.files.wordpress.com/2009/11/akamaiedgeplatform.jpg

MODELOS FUNDAMENTAIS

Modelos fundamentais de um sistema distribuído

Permitem estabelecer quais as premissas que existentes a respeito de aspetos chave.

Permitem avaliar de forma objetiva as propriedades e garantias do sistema resultante.

Modelo de interação

Modelo de falhas

Modelo de segurança

MODELO DE INTERACÇÃO (1)

Modelo de interação

- Num sistema distribuído, a coordenação que permite as várias componentes operar como um todo, requer comunicação por troca de mensagens.

- Modelo de interação, carateriza as interações que ocorrem entre os componentes do sistema distribuído

  - define os padrões de comunicação: no tempo e espaço
  - o grau de acoplamento entre componentes: no tempo e espaço
  - as características e garantias oferecidas pelos canais de comunicação

MODELO DE INTERACÇÃO (1)

Tipo de interação

- Ativa
  - Processo solicita execução de operação noutro processo (de forma explícita)
- Reativa
  - Evento no sistema desencadeia ação num processo
- Indireta
  - Processos comunicam através de um espaço partilhado

Modelo reactivo (publish/subscribe – event-based system): motivação

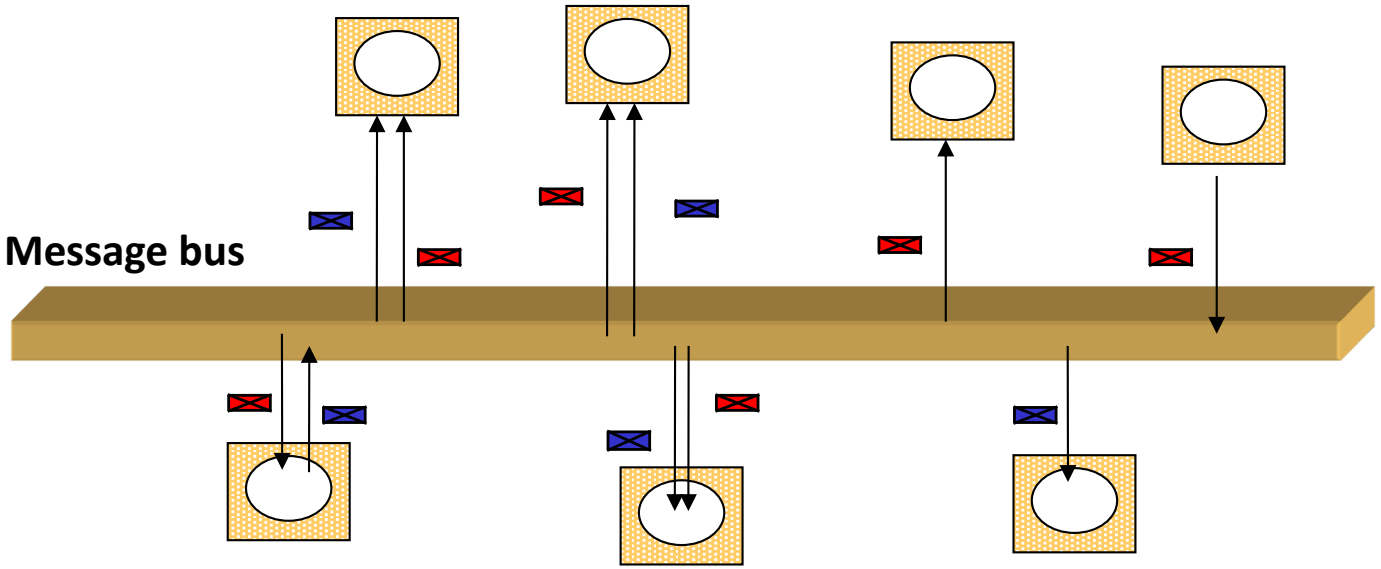
Como difundir informação sobre:

- Eventos dum jogo de futebol
- Cotações da bolsa

Propriedades

- Permitir desacoplar o emissor, dos receptores (no espaço e no tempo)
- Eventos são produzidos de forma independente dos consumidores
- Todos os consumidores consomem todos os eventos
- Consumidores podem variar no tempo

MODELO REACTIVO (PUBLISH/SUBSCRIBE – EVENT-BASED SYSTEM)

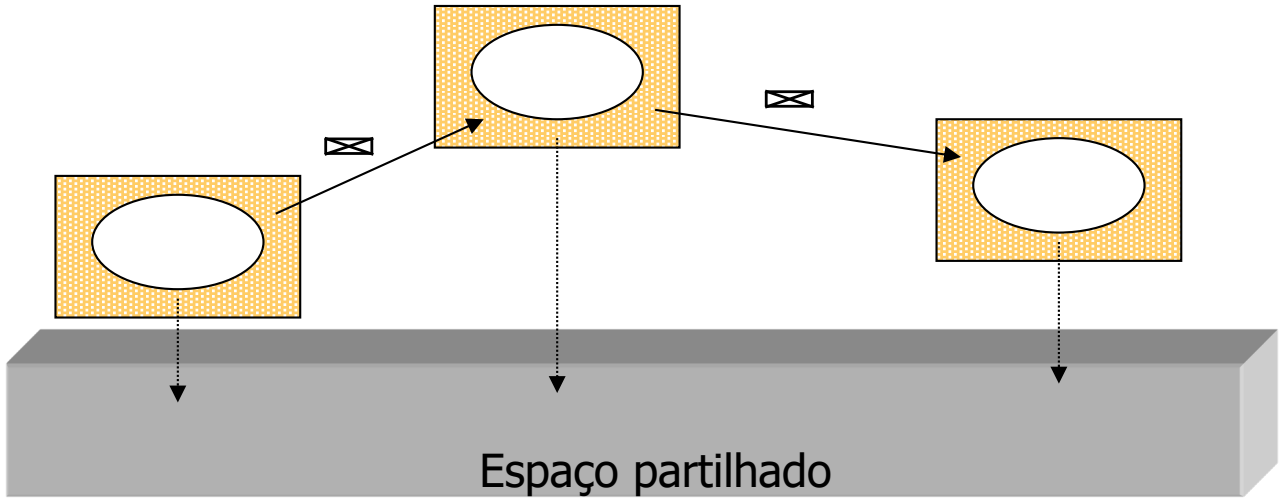


Modelo editor/assinante / "publish/subscribe" / pub/sub

- Processo subscritor/consumidor manifesta o interesse num conjunto de eventos (tópicos ou com base num filtro sobre o conteúdo)
- Processo produtor produz eventos

Sistema encarrega-se de propagar eventos produzidos para subscritores interessados

MODELO DE INTERAÇÃO INDIRETA



Processos comunicam e coordenam-se através dum espaço de "memória partilhada distribuída"

- Processos escrevem e leem dados no espaço de dados partilhada
- Espaço de dados partilhado pode ser base de dados, espaço de tuplos, DHT, etc.

MODELO DE INTERACÇÃO (2)

Tipo de interação

- Ativa
  - Processo solicita execução de operação noutro processo
- Reativa
  - Evento no sistema desencadeia ação num processo
- Indireta
  - Processos comunicam através de um espaço partilhado

Conteúdo da interação

- Informação/dados
- Código

CONTEÚDO DA INTERACÇÃO: DADOS

Processos trocam dados

- Pedido (operação a invocar + parâmetros + inf. utilizador + ...)
- Resposta (resultado de operação + ...)
- Evento (num sistema de eventos)

Propriedades

- Parceiros devem conhecer formato das mensagens
- Parceiros devem saber processar mensagens (operações)

CONTEÚDO DA INTERACÇÃO: CÓDIGO MÓVEL – CLIENTE/SERVIDOR

a) client request results in the downloading of javascript code

b) client interacts with the web application

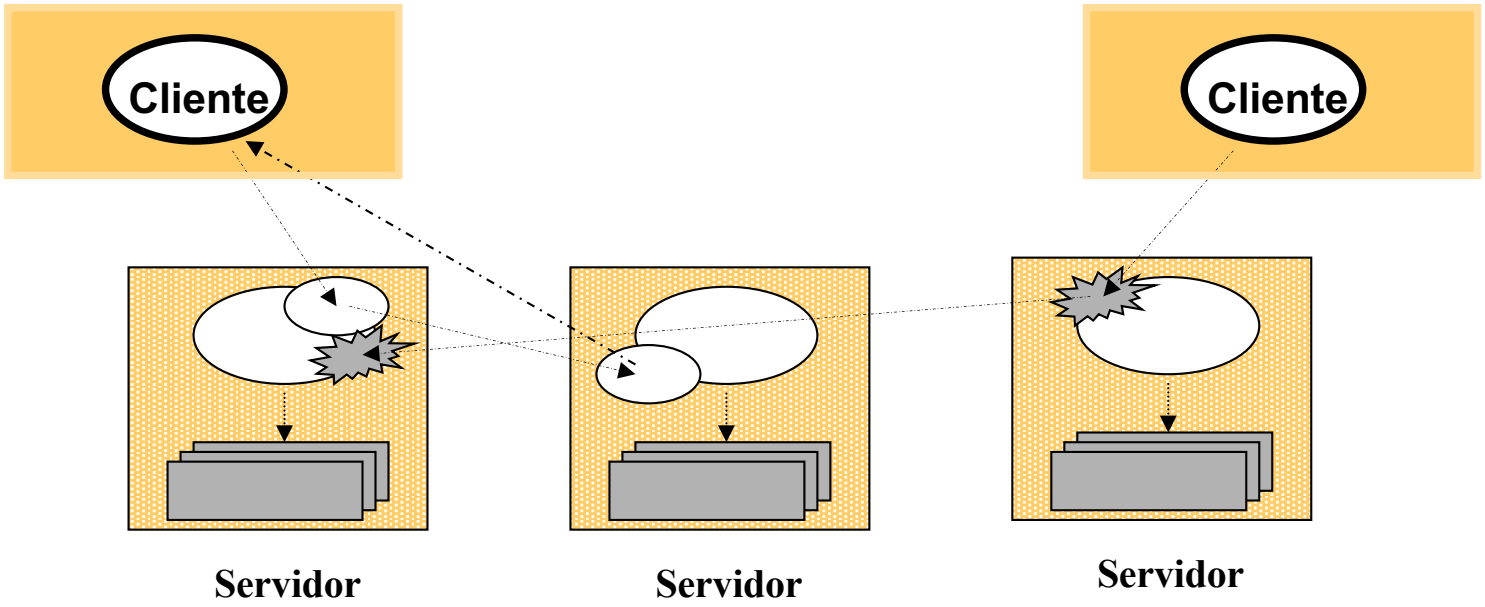
A execução do código no cliente pode:

- Melhorar o desempenho
- Ser usado para implementar funcionalidade adicional

Segurança

- Necessário proteger o cliente do código executado (sandboxing)

CONTEÚDO DA INTERACÇÃO: CÓDIGO MÓVEL – AGENTES



Ideia: agentes navegam entre servidores, executando cada parte no servidor em que é mais apropriado

- Exemplo: ?

Problemas

- Proteger informação do agente do ambiente de execução (impossível?)
- Desenhar sistema de forma que recursos usados sejam menores do que outra arquitetura

SISTEMAS DISTRIBUÍDOS

Capítulo 2

Arquiteturas e Modelos de Sistemas Distribuídos

NA ÚLTIMA AULA...

Combinações modelo cliente/servidor e modelos peer-to-peer

- E.g. BitTorrent, P2p hierárquico

Modelos fundamentais de um sistema distribuído

Modelo de interação

NA AULA DE HOJE....

Modelos fundamentais de um sistema distribuído

...

Modelo de falhas

Modelo de segurança

MODELO DE FALHAS

Num sistema distribuído, tanto os processos (e computadores) como os canais de comunicação podem falhar

- Não é possível conceber componentes sem falhas, apenas se pode diminuir a probabilidade de as mesmas ocorrerem

- modelo de falhas consiste na definição rigorosa de quais os erros ou avarias, assim como das falhas que podem ter lugar nas diferentes componentes.

- o modelo de falhas abrange ainda a indicação rigorosa do comportamento global do sistema na presença dos diferentes tipos de falhas.

ALGUMAS DEFINIÇÕES

Fault tolerance - tolerância a falhas. Propriedade de um sistema distribuído que lhe permite **recuperar** da existência de falhas **sem introduzir comportamentos incorretos**.

Um sistema deste tipo pode **mascarar as falhas e continuar a operar**, ou **parar** e voltar a operar mais tarde, de forma coerente, após reparação da falha.

ALGUMAS DEFINIÇÕES

Availability – disponibilidade. Mede a fracção de tempo em que um serviço está a operar correctamente, isto é, de acordo com a sua especificação.

- Para um sistema ser altamente disponível (highly available) deve combinar

um reduzido número de falhas com um curto período de recuperação das falhas (durante o qual não está disponível).

Reliability - fiabilidade. Mede o tempo desde um instante inicial até à primeira falha, i.e., o tempo que um sistema funciona correctamente sem falhas.

- Um sistema que falha com grande frequência e recupere rapidamente tem

baixa fiabilidade, mas alta disponibilidade.

CLASSIFICAÇÃO DOS SISTEMAS

Classe	Disponibilidade	Indisponibilidade (por ano – máximo)	Exemplos
1	90,0%	36d 12h	Personal clients, experimental systems
2	99,0%	87h 36min	entry-level business systems
3	99,9%	8h 46min	top Internet Service Providers, mainstream business systems
4	99,99%	52min 33s	high-end business systems, data centers
5	99,999% (alta disponibilidade)	5min 15s	carrier-grade telephony; health systems; banking
6	99,9999%	31,5 seg	military defense systems

TIPOS DE FALHAS DOS COMPONENTES

Uma falha por omissão dá-se quando um processo ou um canal de comunicação falha a execução de uma acção que devia executar

- Exemplo: uma mensagem que devia chegar não chegou, processo falha (crash)

Uma falha temporal dá-se quando um evento que se devia produzir num determinado período de tempo ocorreu mais tarde – normalmente em sistemas de tempo real

- Exemplo: limite temporal para a propagação de uma mensagem, execução dum passo de computação, etc.

Uma falha arbitrária ou bizantina dá-se quando se produziu algo não previsto

- Exemplo: chegou uma mensagem corrompida, um atacante produziu uma mensagem não esperada.
- Para lidar com estas falhas é necessário garantir que elas não levam a que outros componentes passem a estados incorretos

TIPOS DE ERRO/FALHA: DURAÇÃO

Permanentes: mantêm-se enquanto não forem reparadas (ex: computador avaria)

- Mais fáceis de detectar
- Mais difíceis de reparar

Temporárias: ocorrem durante um intervalo de tempo limitado, geralmente por influência externa

- Mais difíceis de reproduzir, detectar
- Mais fáceis de reparar

- Erros transientes: ocorrem instantaneamente, ficam reparados imediatamente após terem ocorrido (ex.: perda de mensagem)

EXEMPLO: TRABALHO PRÁTICO

Qual o modelo de falhas.

Falhas dos nós?

Não, no primeiro trabalho assume-se que os nós não falham.

Falhas de comunicação?

Sim, falhas por **omissão, temporárias**, i.e., as mensagens podem-se perder durante um período de tempo limitado.

SEGURANÇA NUM SISTEMA DISTRIBUÍDO (MODELO DE SEGURANÇA)

A informação gerida por um sistema distribuído tem valor para os utilizadores

A segurança de um sistema **não é** uma garantia absoluta.

Tratam-se de medidas para **gerir o risco** de o sistema ser comprometido.

Como? **Aumentando o custo do ataque** face ao valor associado ao sistema.

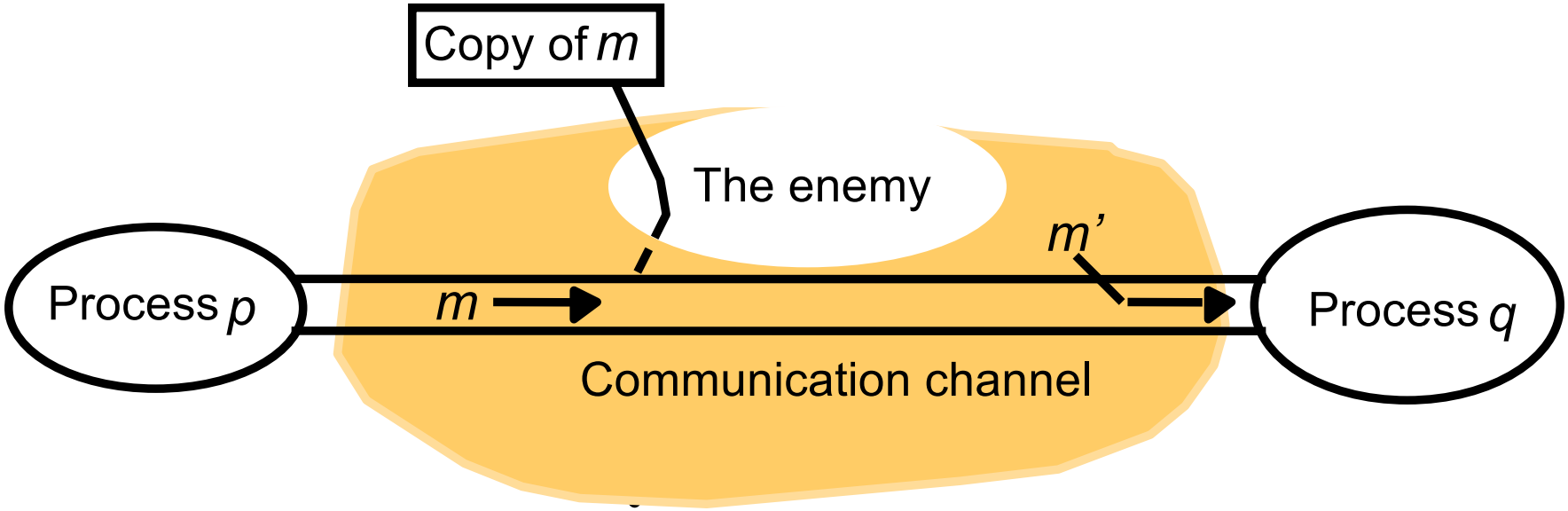
SEGURANÇA NUM SISTEMA DISTRIBUÍDO (MODELO DE SEGURANÇA)

O modelo de segurança consiste em definir quais as ameaças das quais um sistema se consegue defender

Por outra palavras, consiste em **definir o modelo do atacante**.

Enumerando o que o atacante pode ou não fazer, sem que o sistema resulte comprometido.

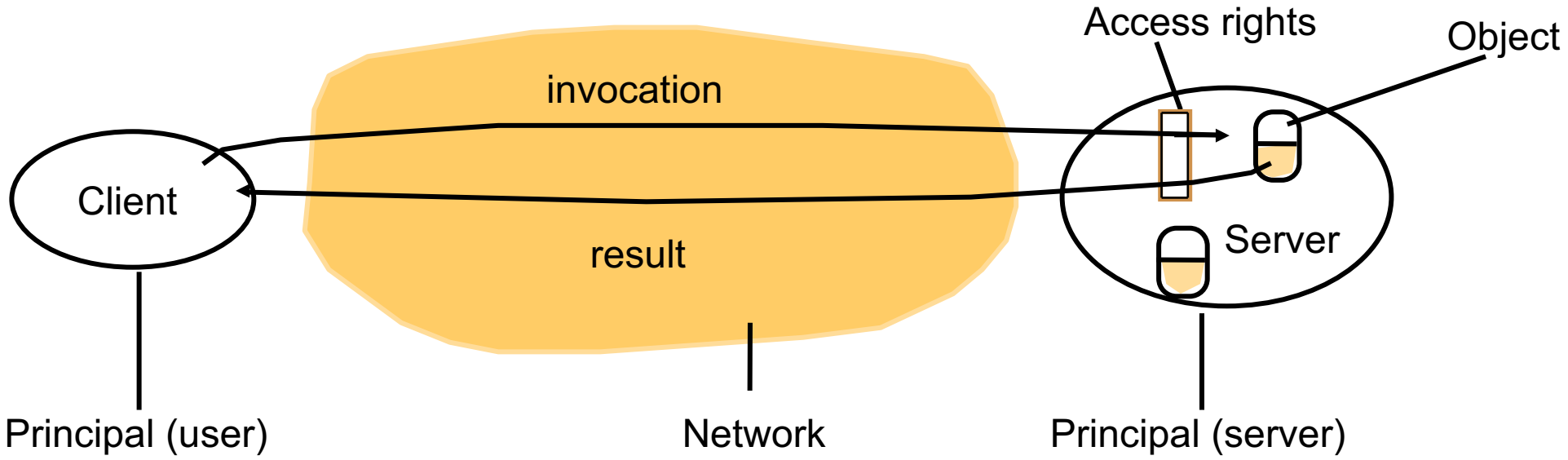
AMEAÇAS BÁSICAS AOS CANAIS DE COMUNICAÇÃO (E CONSEQUENTEMENTE AOS PROCESSOS)



Para modelar ameaças de segurança, assume-se que o inimigo tem grande capacidade e pode:

- Enviar mensagens para qualquer processo
  - Forjar endereço das mensagens, fazendo-se passar por outro processo
- Ler, copiar, remover e reenviar mensagens que passam no canal
  - Impedir interacção, repetir interacção, etc.

PRINCIPAIS, OBJECTOS, CONTROLO DE ACESSO E CANAIS



A segurança num sistema distribuído passa por:

- Autenticar os principais
- Verificar direitos de acesso aos objectos – controlo de acessos
- Utilizar canais seguros

OUTRAS AMEAÇAS

Denial of service: ataque em que o inimigo interfere (impede) atividade dos utilizadores autorizados

Distributed Denial of service: versão em os agente que desencadeiam o ataque de negação de serviço é desencadeado de forma descentralizada por vários agentes.

Muito difícil de prevenir e muito dispendioso

e.g., Akamai DDoS Protection

PARA SABER MAIS

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

Capítulo 2.