

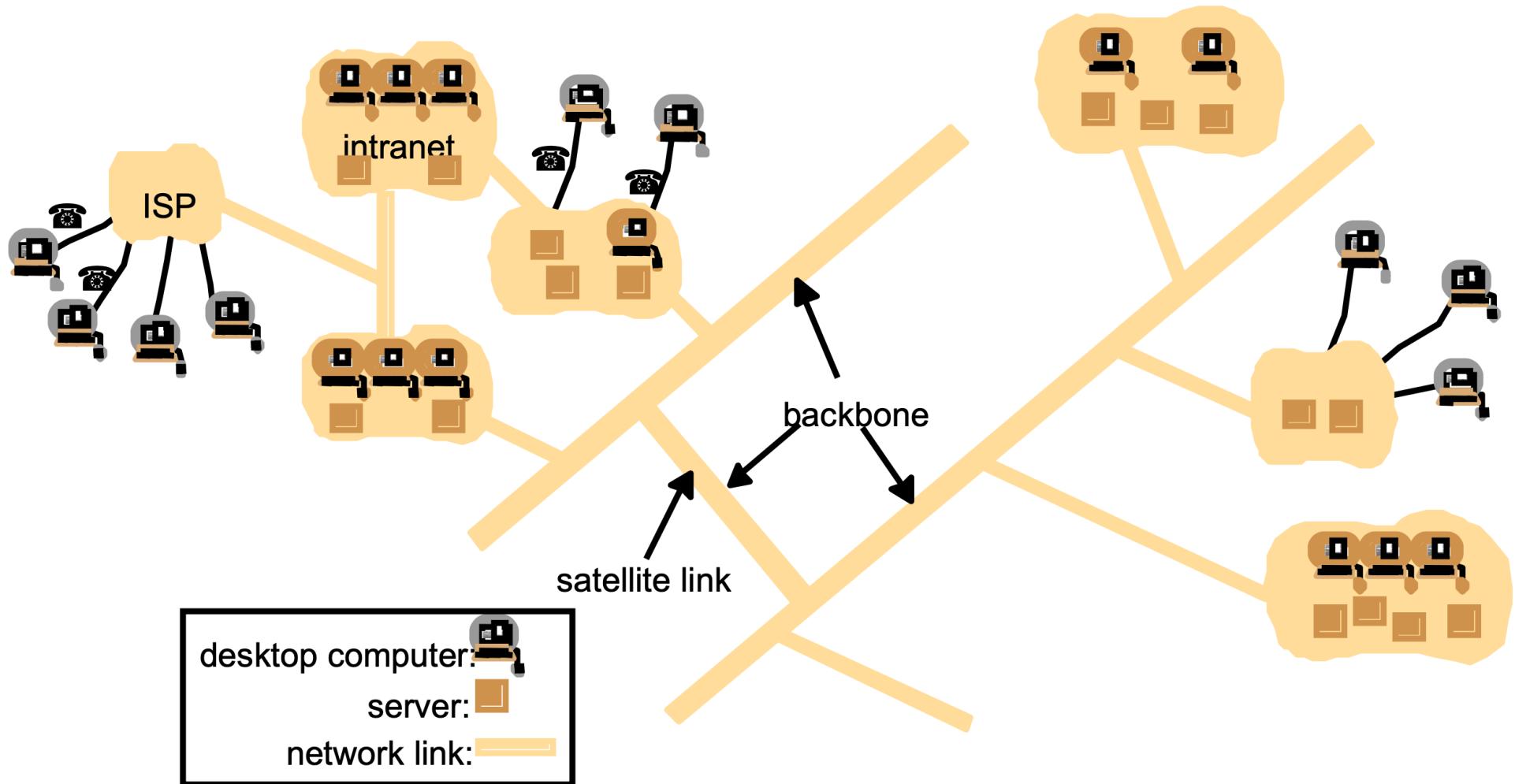
"How does he ever expect to be an architect if he can't invent a new roof?"

Introdução/Arquitecturas

Computação Distribuída 2023/24

dgomes@ua.pt

1 / 31



Sistemas Distribuídos

Aplicação ≠ Serviço
↓ ↓
tem dono vou pagando ...

Definição

Sistema em que os componentes de hardware e de software, localizados numa rede de computadores, comunicam e coordenam as suas acções através da passagem de mensagens
(DS Coulouris)

Um conjunto de computadores independentes que são apresentados ao utilizador como um único sistema integrado
(DS Tanenbaum)

"Um sistema distribuído é uma coleção de elementos de computação autónomos que são apresentados ao utilizador como sendo um único sistema integrado e concorrente"

Características

- Colecção de elementos de computação autónomos.
 - Todos os peças não independentes!,, →
 - ↑ servidores de abstracção
 - / \ \ \
 - vários servidores especializados
- Relógio global
 - Eles têm de estar sincronizados!,,
 - Não podemos levantar o dinheirinho 2 vezes em caixas diferentes
- Gestão de membros
 - Aberto → Permite entrada e saída (escalável !!)
 - Fechado → Não muda! (estável !!)
- Sistema único coerente.
 - Igual para cada utilizador!

Middleware

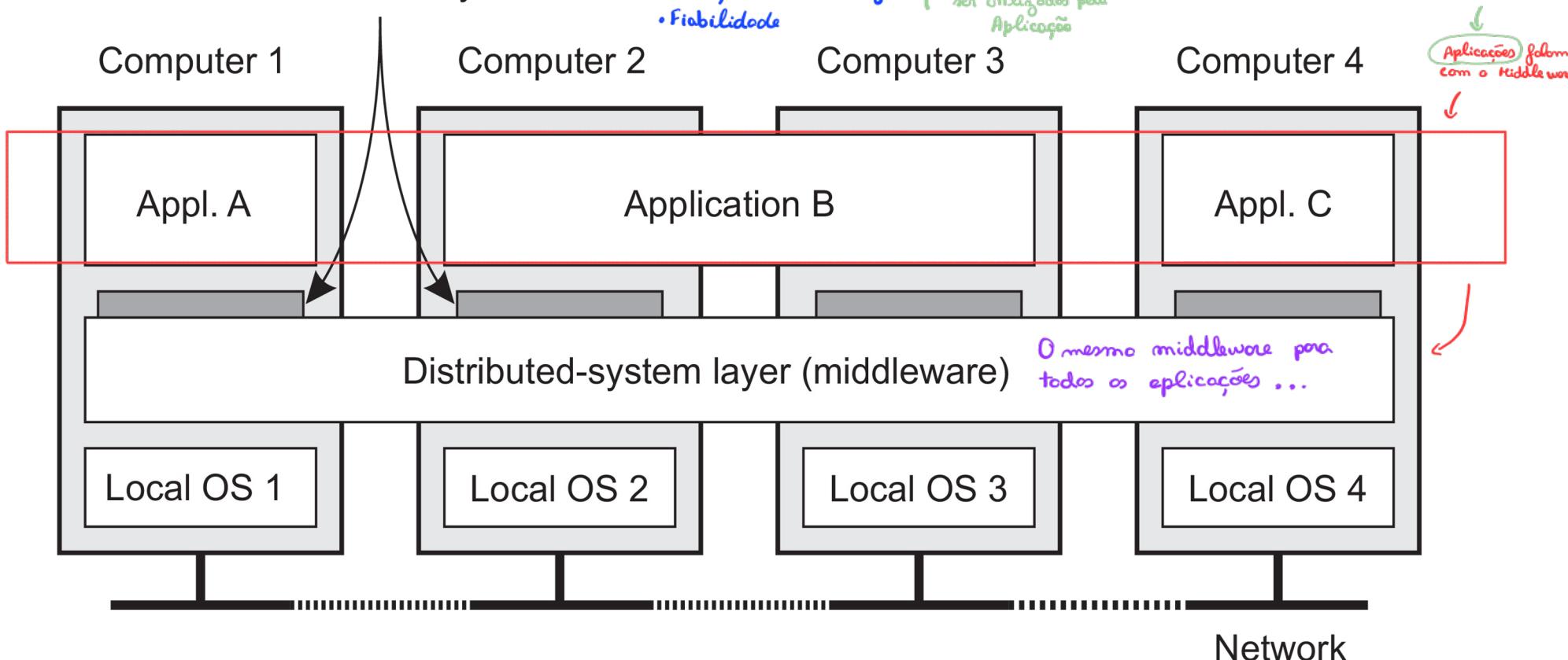
(Abstracção) Aplicações - SO

Assim, as aplicações não têm
de lidar com sistemas
distribuídos

Dimens é muitos vezes
um Middleware

→ Camada de abstracção usada para ajudar no desenvolvimento
de SO, logicamente colocado por cima do SO

Same interface everywhere



Middleware

Serviços disponibilizados:

- Comunicação (ex. RPC)
enviaçao uma função num PC remoto ?
- Transacções
Mais de que uma operação
mas tudo de uma vez
Todos ou Nenhuma
(Operação atómica)
- Composição de Serviços
- Fiabilidade
~ Confiança no Sistema (DropBox)
acredita-se que não se perde
ficheiros ...)
Isto é um grande
problema ...
(veremos ...)

Qual o objectivo?

- Suportar a partilha de recursos
- **Distribuição transparente (aplicação e utilizador)**
- Abertura *→ Especificações públicas (interfaces)
"Como a aplicação usa ??"*
- Escalabilidade

1 utilizador / 10 utilizadores / 1000 ...

Acesso aos websites, temos um Servidor
que envia o web para o Client.

↑
Tudo na internet é
código aberto!,,

• Em Java, é feita muita abstracção...
→ corre em qualquer computador!

• JVM - Java Virtual Machine é uma Middleware ☺

Distribuição Transparente

→ Vários níveis...

Transparência	Descrição
Acesso → Fibra & Satélite, mas para nós é igual ...	Esconde diferenças na representação dos dados e na forma como os mesmos são acedidos
Localização	Esconde onde um objecto está localizado
Relocalização → chamada num carro...	Esconde que um objecto pode ser movido para outra localização enquanto usado
Migração → Backups...	Esconde que um objecto pode ser movido para outra localização
Replicação ↳ esconde os ficheiros...	Esconde que um objecto pode ser replicado
Concorrência "várias pessoas a escreverem no mesmo ficheiro"	Esconde que um objecto pode ser partilhado entre diversos utilizadores concorrentes
⚠ Falhas	Esconde as falhas e recuperações de um objecto

Abertura

"openness"

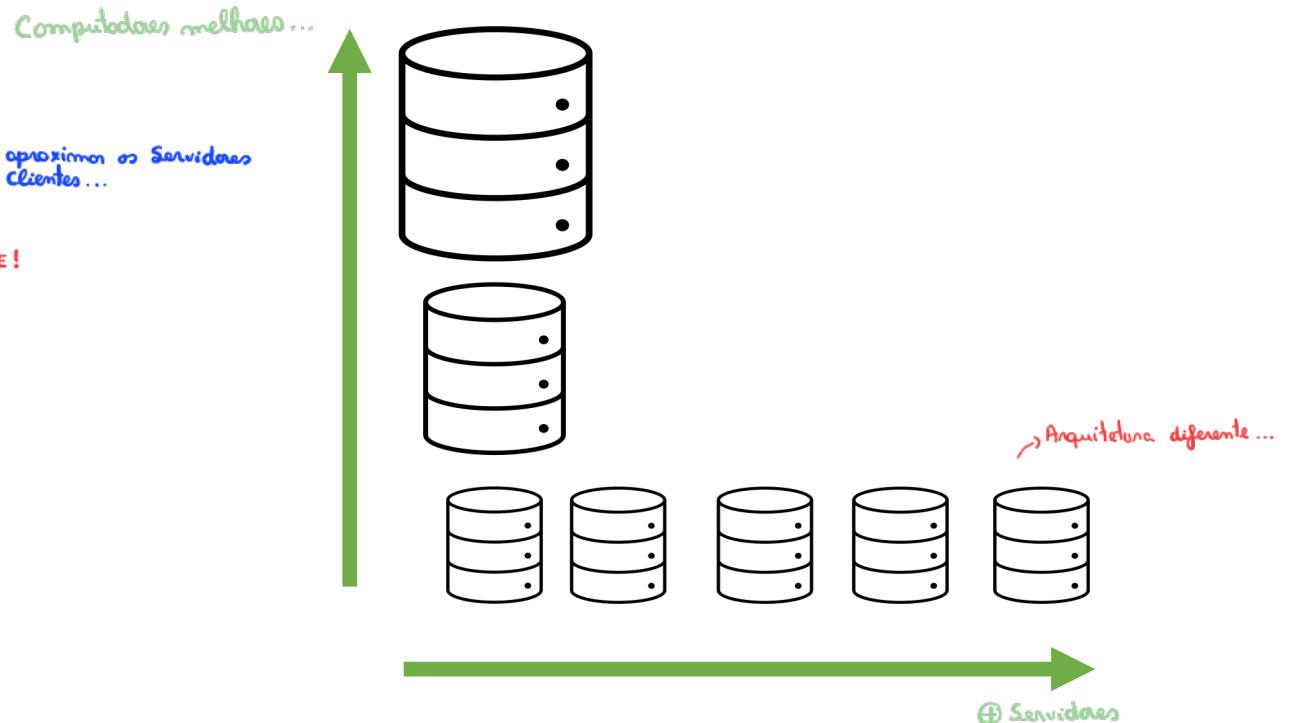
- “Being Open”
- Utilizar componentes que podem ser usados por, ou integrados, em outros sistemas
- Inter-operação
- Composição
- Extensibilidade

Escalabilidade

Dimensões

- Tamanho
- Geográfica *→ Jogos tentam aproximar os Servidores dos clientes...*
- Administrativa
→ Dados europeus ... Apenas em Servidores da UE!

Vertical vs Horizontal



- Coisas que não consideramos na computação normal

Armadilhas (Pitfalls)

Erros Comuns ...

Temos de contar com a falha...

A rede é fiável

As ordens nos jogos de futebol têm de ser amplificadas (e.g.: 100 pessoas \Rightarrow 100.000 pessoas)

A topologia
não se altera

Dúvidas dos ambientes em que estamos..

A rede é
segura

tempo entre o Cliente - Servidor

Latência é
zero

"Vídeos todos em 8K"

\rightarrow mas os clientes não vão todos conseguir
"Percorrer o contexto do cliente"

A rede é
homogénea

Largura de
banda é
infinita

Europa \neq USA \neq China

Custo de
transporte é
zero

Valor com os clientes custa
dinheiro... (aficiente)

Existe um
único
administrador

Arquitecturas de Sistemas Distribuidos

Tipos:

→ Cluster/Grid/Cloud Computer

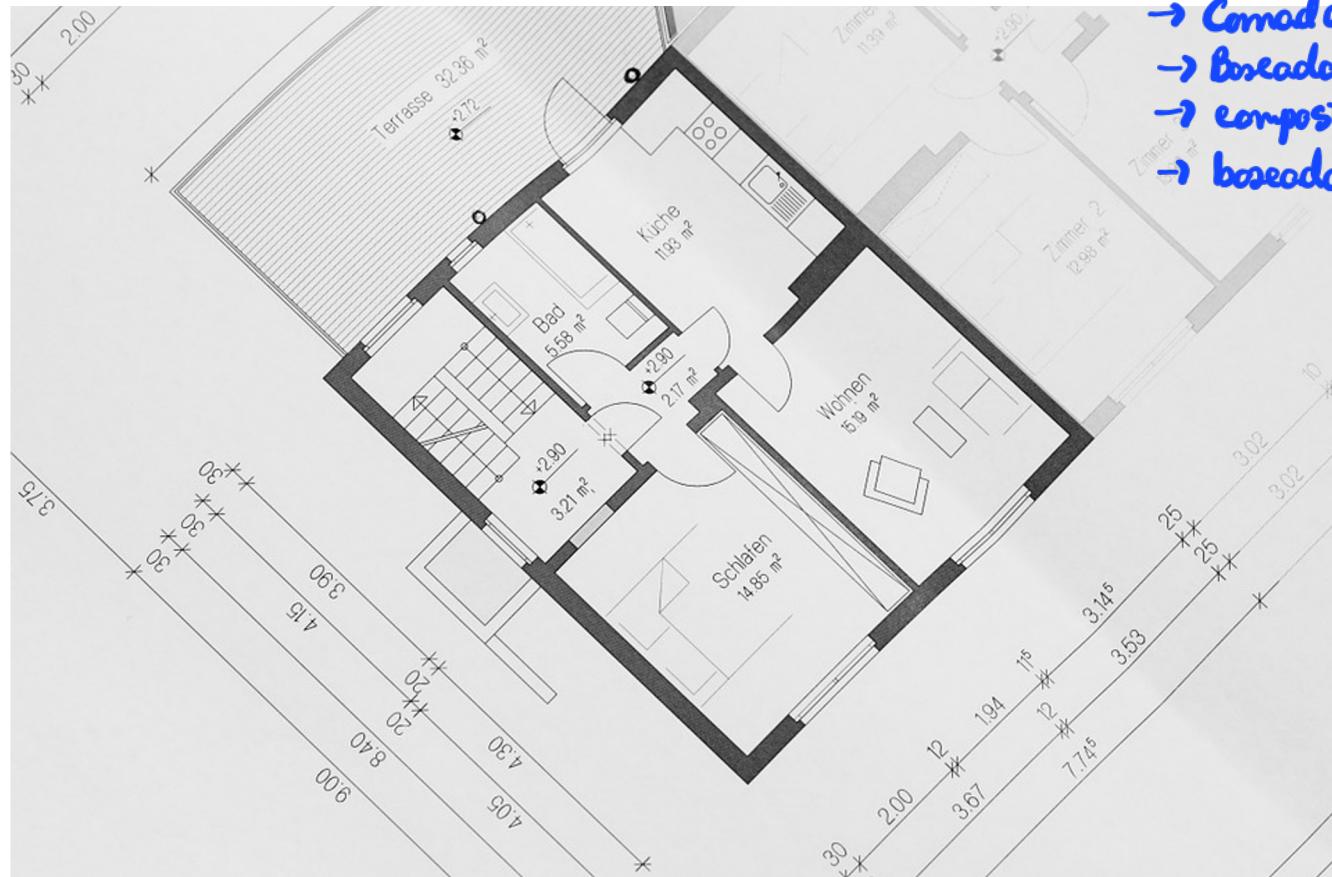
Estilos:

→ Comandos

→ baseada em Objetos

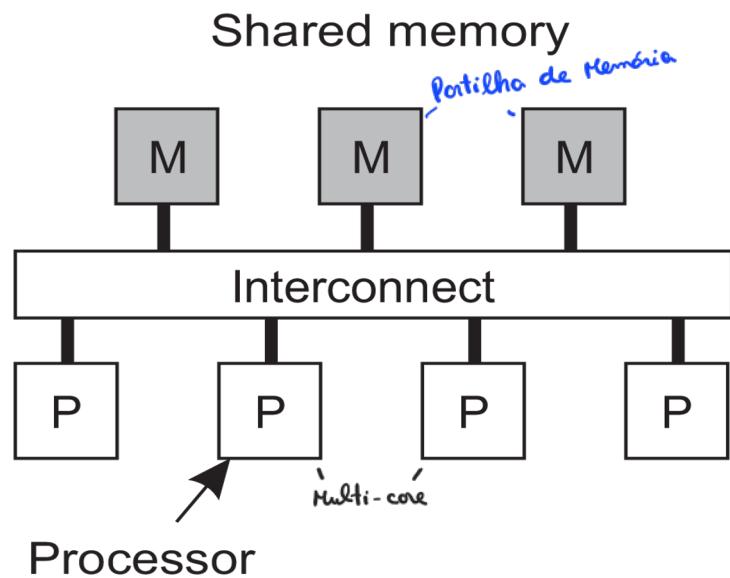
→ composta por recursos

→ baseada em Pub-Sub

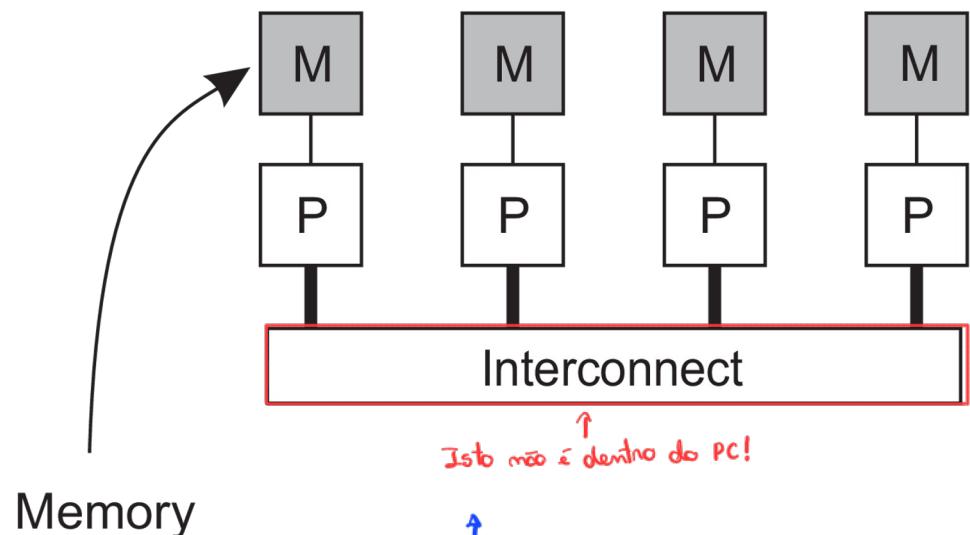


Arquitecturas de Hardware

Multiprocessador e multicore VS multicomputador



Private memory



- Não têm uma boa escalabilidade
(existe limites na ligação à Shared Memory)

Solução

dgomes@ua.pt

- Outro problema: comunicação entre os nós tem de ser extremamente eficiente

Arquitecturas

Cluster Computing

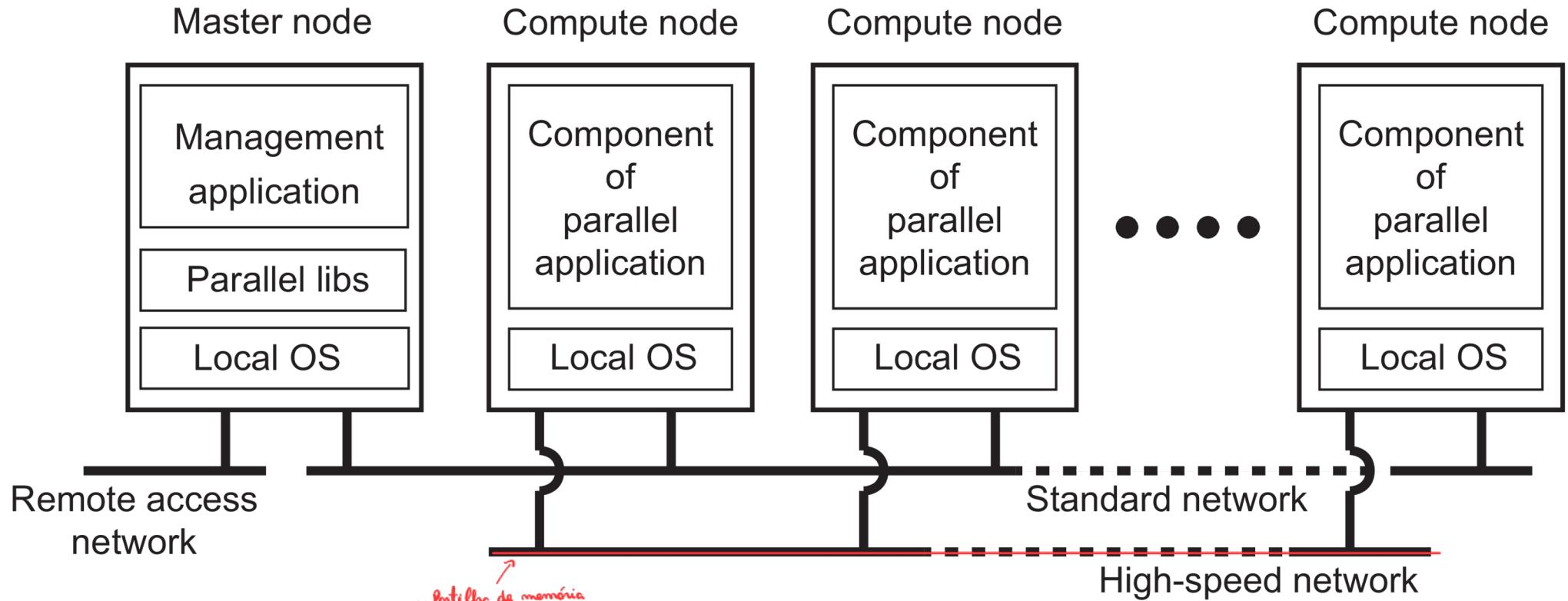
Exemplo de Super Computadores que
s o utilizados para simular algo ...

Super Cool !

(em paralelo)
→ 1 aplica o a funcionar
em m computadores em
simultâneo

⇒ Aqui, o Middleware n o ´
muito abstrato ...

- N os similares conectados num "high-speed network"



Arquitecturas

- São muito dinâmicas ...
→ Nos temos muitos computadores parados!,,

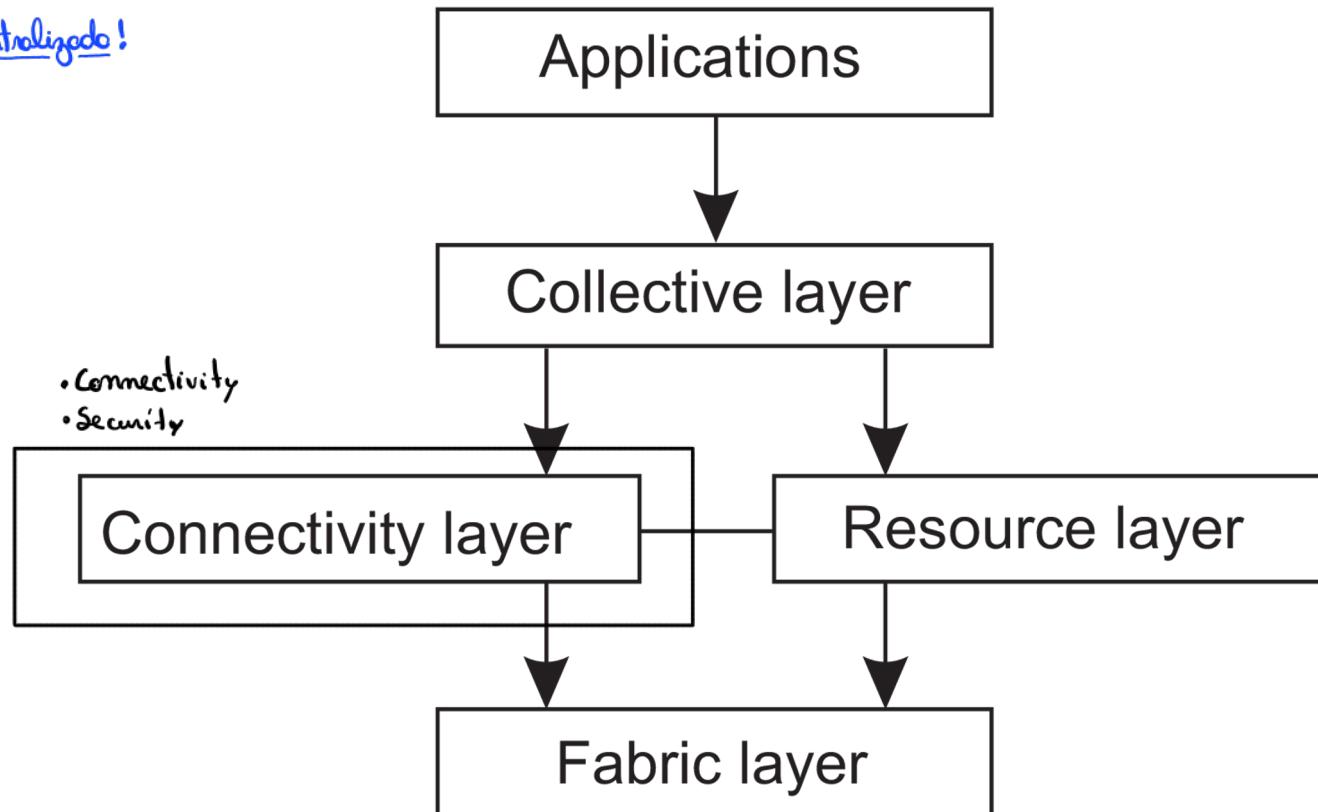
Grid Computing

↑
em rede

• Descentralizado!

(Sem abstrações, mas está mais perto ...)

→ Não necessitam de ser computadores iguais!



Arquitecturas

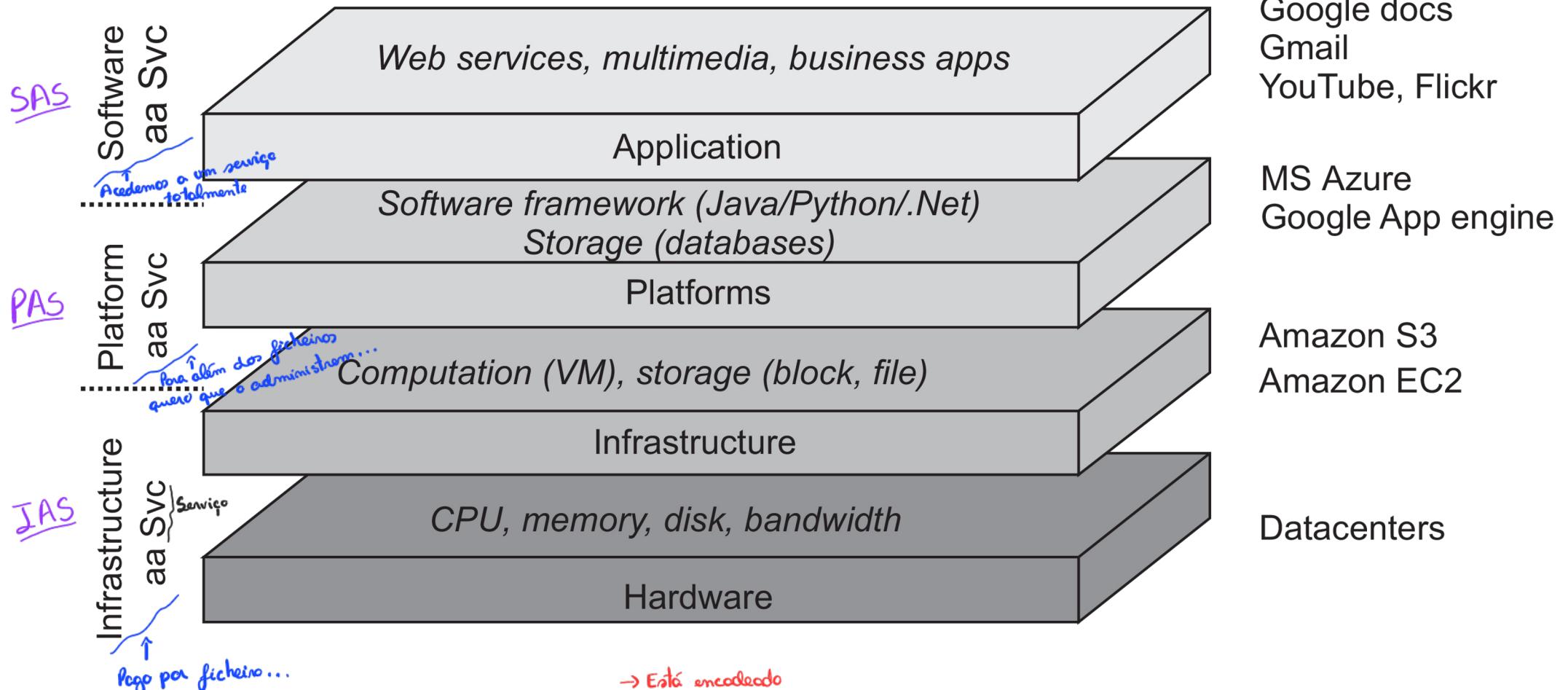
Muito Importante !!!

- Pago para utilização de um computador ...
→ Permite expandir a nossa computação a qualquer momento ...

• Uma boa solução !

Cloud Computing

→ Ex. de um DataCenter

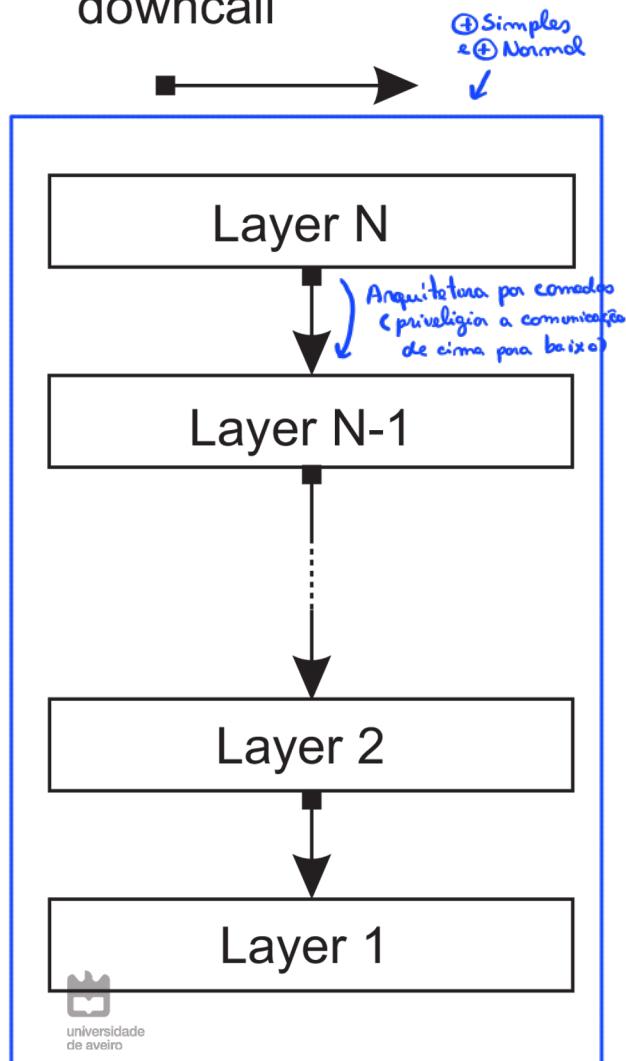


dgomes@ua.pt

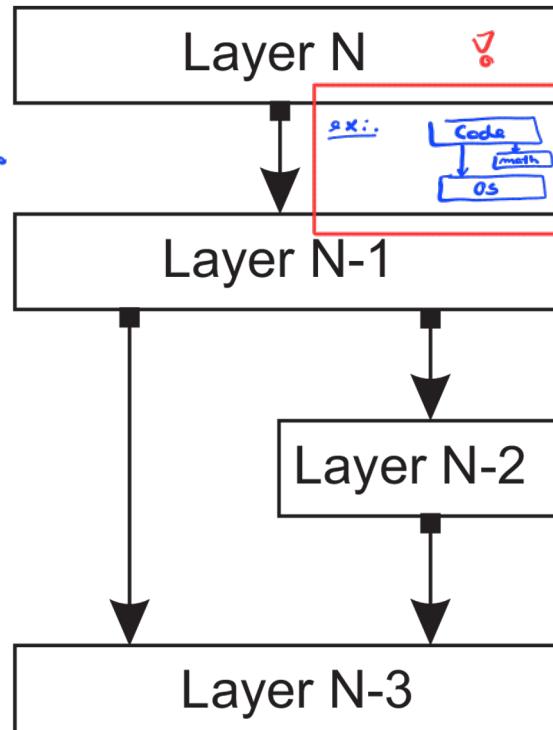
16 /

Arquitectura por camadas

Request/Response
downcall



One-way call



• chamados Sincronos ou Assimetricos

- Pedido
- Resposta
- Pedido
- (...)

• Pode saltar camadas !!!

→ Cada camada só comunica (normalmente) com a camada de cima e a da baixa

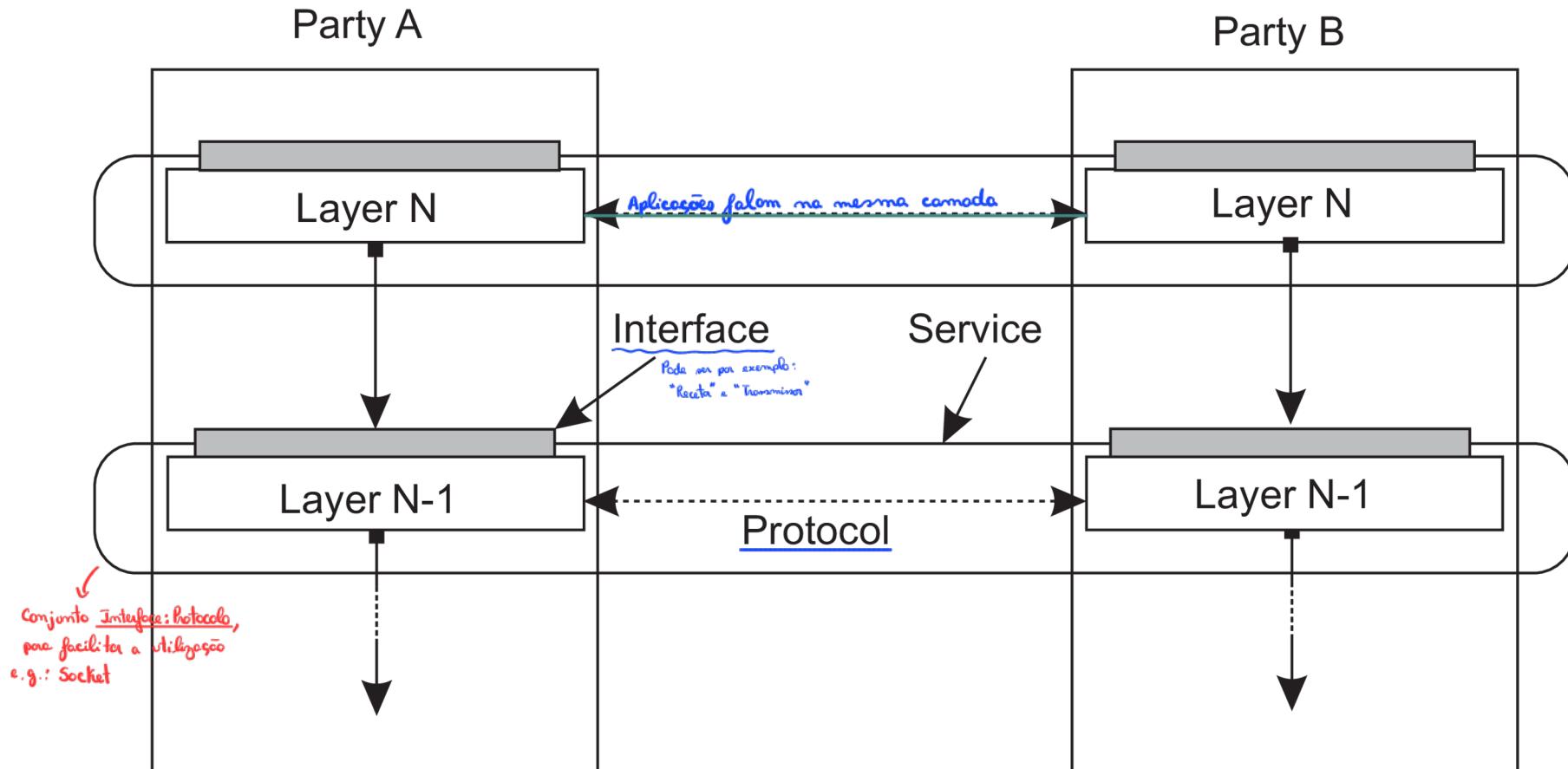
dgames@ua.pt

17 /

• Ainda por comodos...

Protocolo, serviço, interface

Ex: usando TCP

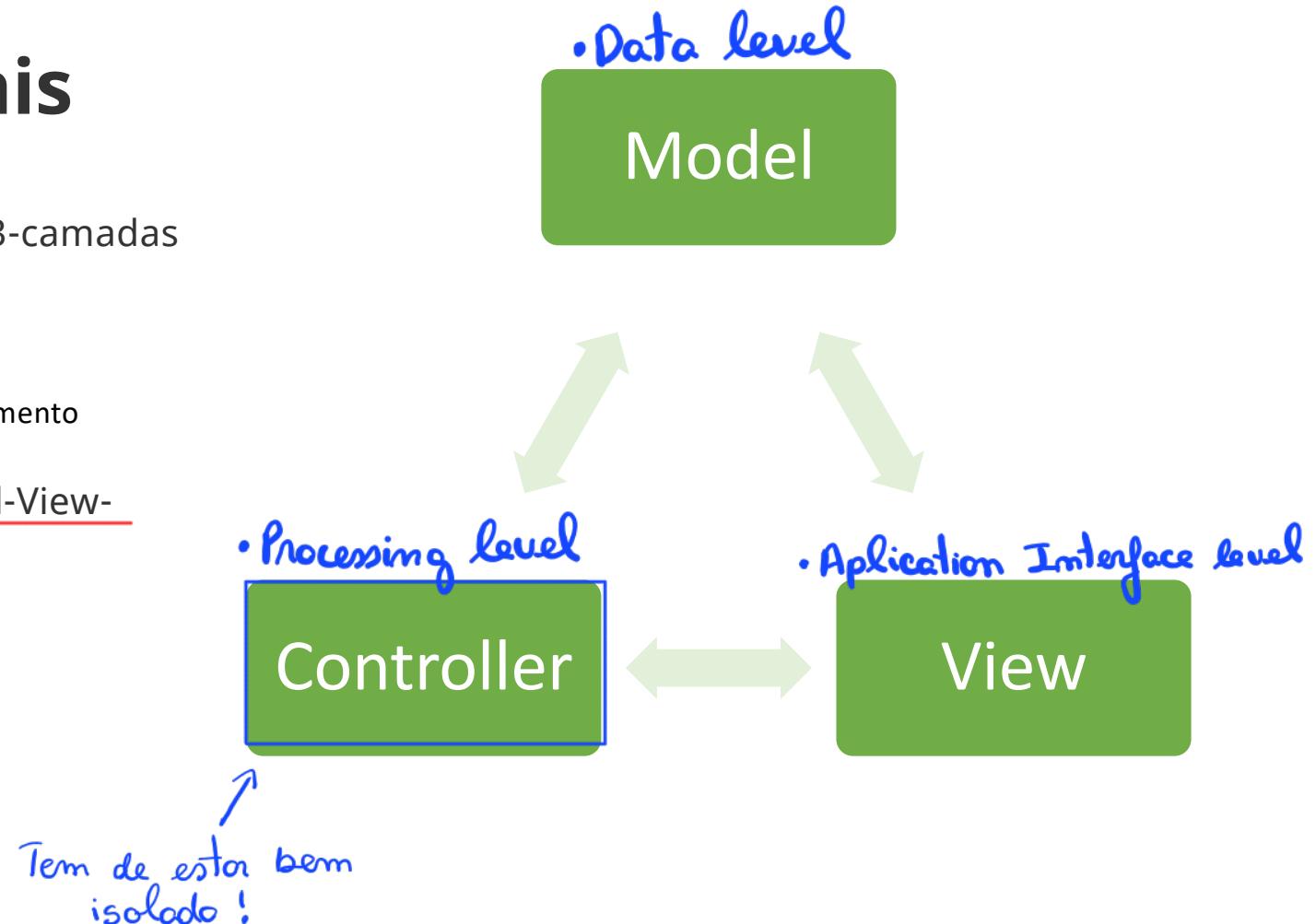


Camadas Aplicacionais

- Visão tradicional de 3-camadas
 - Camada de Dados
 - Camada de Interface
 - Camada de Processamento

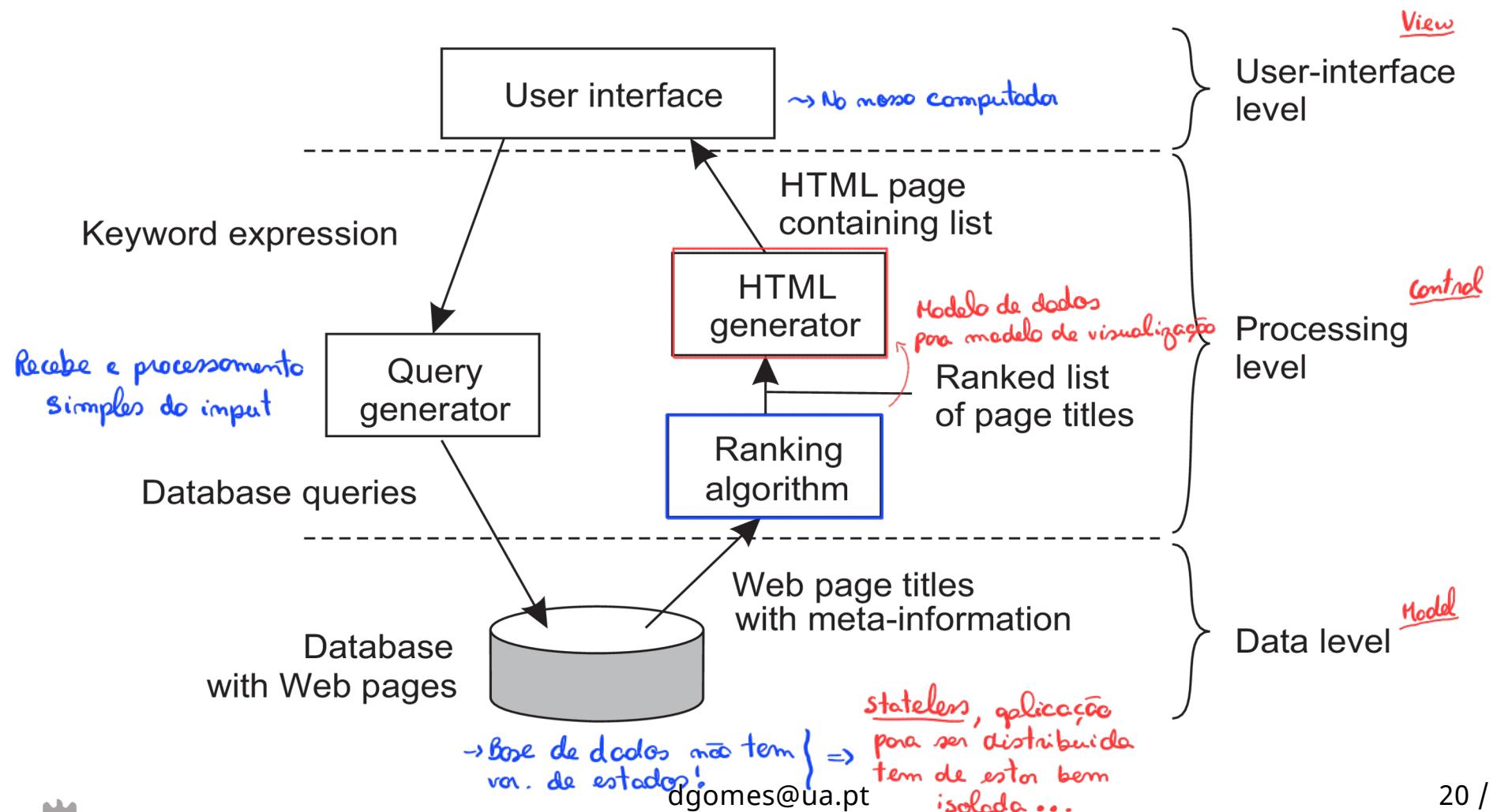
Em inglês: MVC - Model-View-Controller

- Podemos e devemos separar os coisões...
→ Controller é o com mais diferenças



MVC

Exemplo: Motor de Pesquisa



Arquitecturas baseadas em objectos e orientadas a serviço

~ Dos primeiros ...

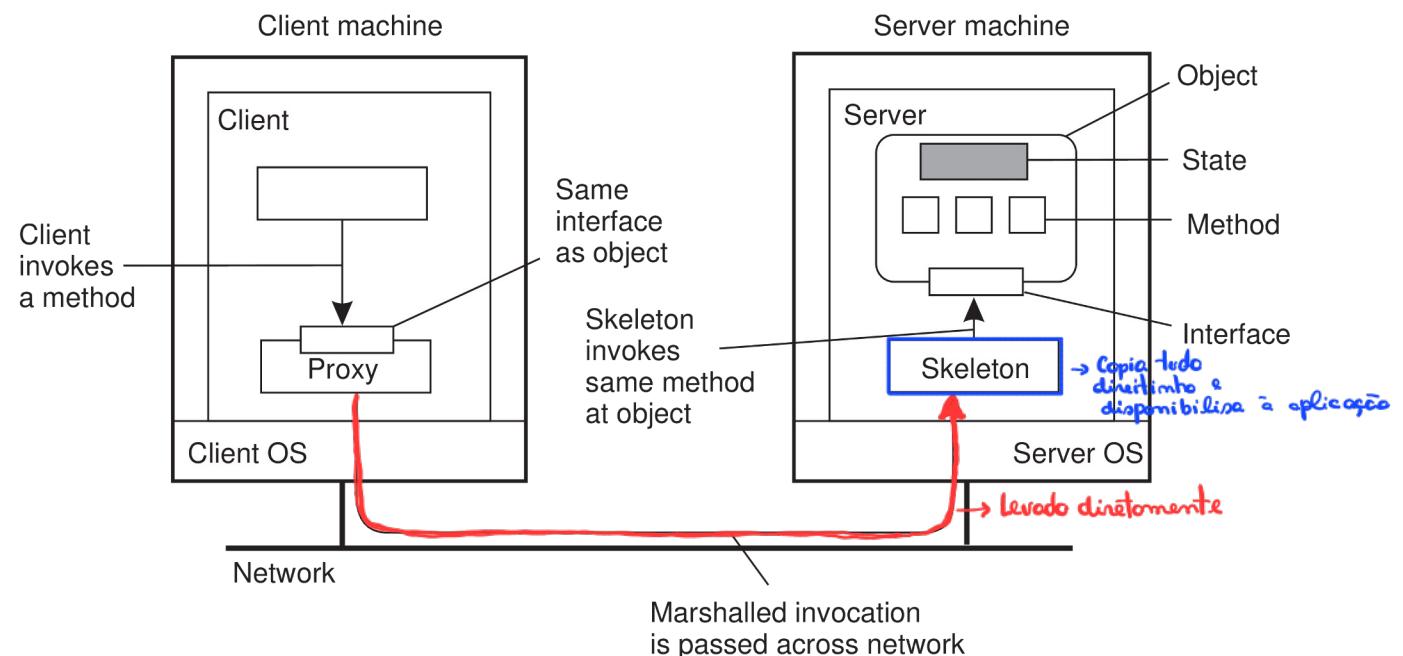
Toda a gente tem de estar na mesma versão...
(Invíavel)

Arquitecturas baseadas em objectos:

- encapsulamento natural dos dados
- Interface do objecto esconde detalhes de implementação
- Quando cliente faz bind a um objecto distribuído, é carregada uma implementação da interface do objecto
 - proxy
- Do lado do servidor surge uma entidade complementar o skeleton

Pode variar com o SO

Referência para o objecto...
Basta associar o endereço!



Isto no mundo ideal é bom, mas...
→ entre diferentes computadores é muito complicado (versões incompatíveis)

Arquitecturas baseadas em Recursos

Características de uma Arquitetura REST:

- Recursos são identificados por um esquema de identificadores único. (*mapping scheme*)
- Todos serviços oferecem mesma interface (*operações*)
- Todas mensagens são autocontidas {Têm toda a informação necessária para serem identificados}
- Após executar uma operação o componente esquece tudo sobre quem o chamou

↳ stateless → não tem estado associado
"stateless execution"

Permite crescimento horizontal!
(vários máquinas)

• Nasceu do problema de integrar muitos PCs na web...

Operação	Descrição
POST	Cria um recurso
GET	Acede ao estado de um recurso
DELETE	Destroi um recurso
PUT	Altera um recurso, substituindo o estado

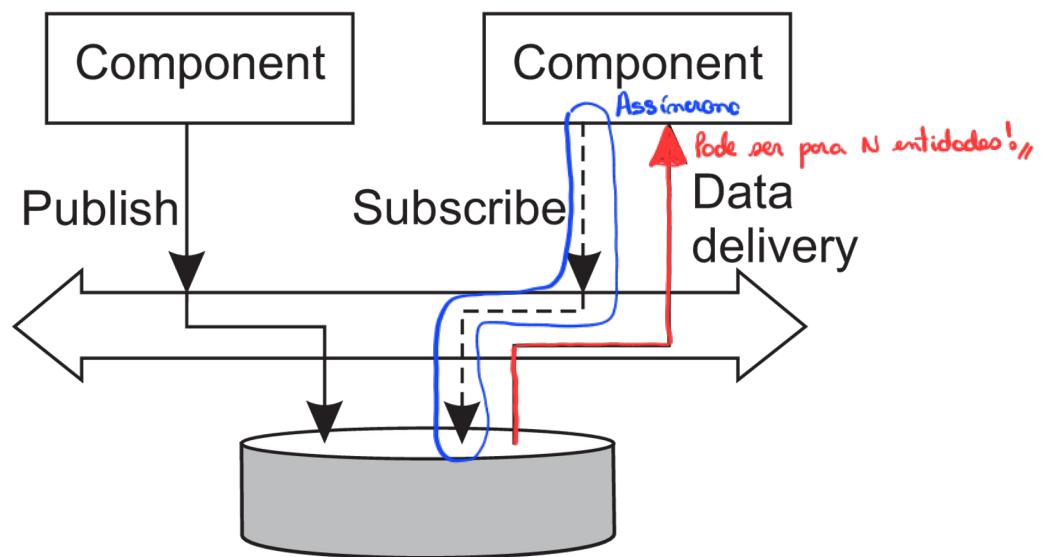
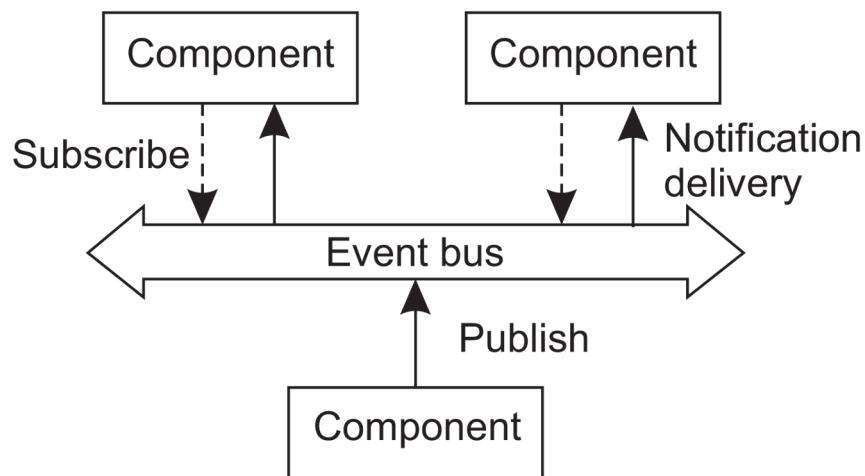
• Podem ser adicionados/removidos por aplicações remotas

Arquitecturas Publish-Subscribe

- Queremos que os processos sejam o mais "soltos" possível !!!
→ Separação entre processos e coordenação

→ Padrão assíncrono

- Um problema aqui é gerir o número de pedidos e notificações que conseguimos suportar



- Fazemos Subscribe do evento que queremos (momming entra em ação)

É ativo quando houver um Publish

- Quando esse evento for ativo enviamos com base dgomes@ua.pt no momming é enviado ...

Shared (persistent) data space

"câilindro"

→ Responsável por entregar a informação!

(Aqui não é Pedido - Resposta)

→ Pedido de subscrição

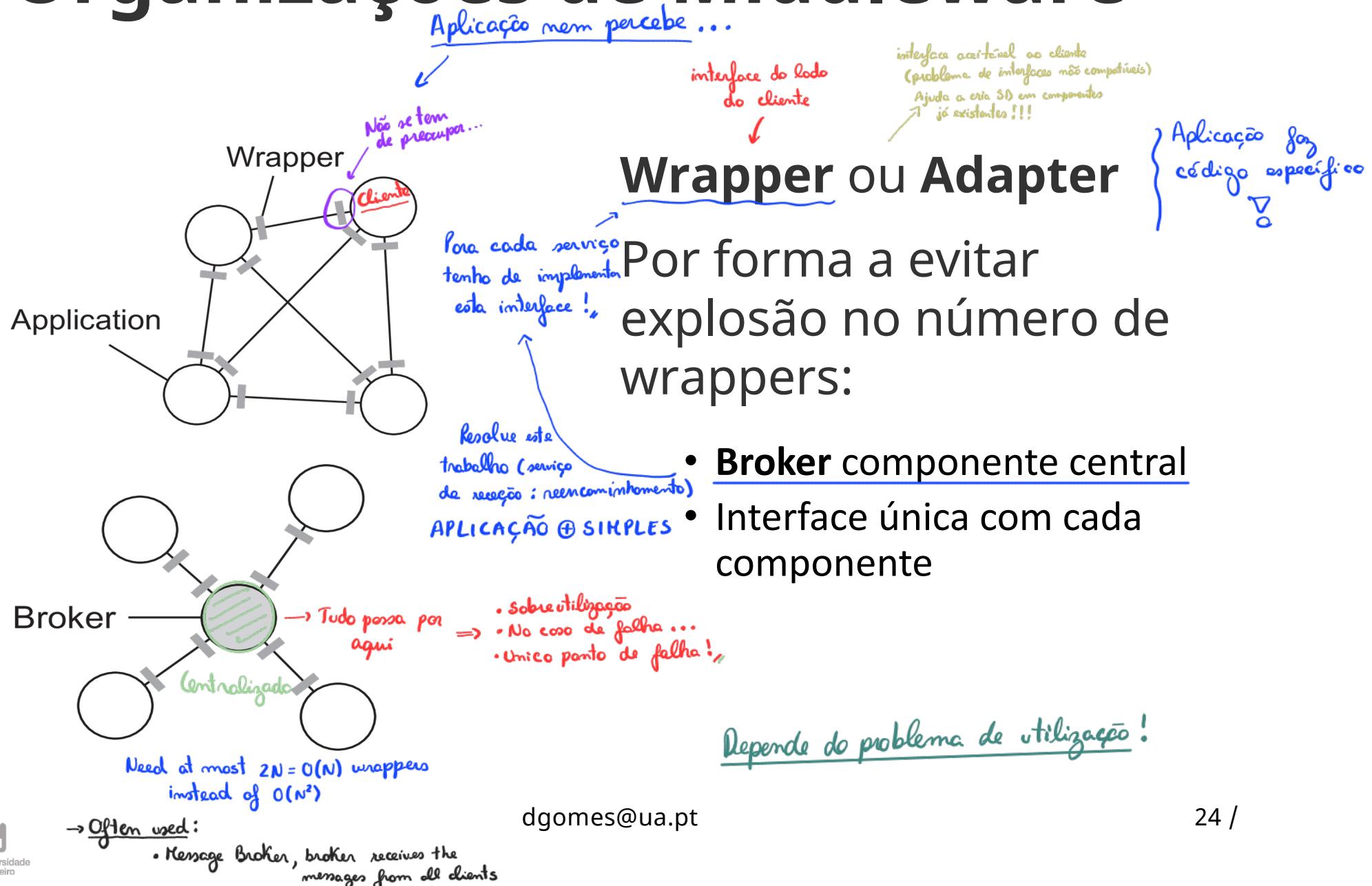
→ Sempre que é uma resposta ele procura alguém subscrito!,

23 /

Exemplo o SG adapta-se a qualquer sistema de ficheiros (FAT, NFS), as aplicações não querem saber... { O SG aqui é um middleware!

Organizações de Middleware

→ Atingir a abertura!



Organizações de Middleware

- **Interceptors**
- Ao objecto A é oferecido um interface local
- A chamada de A é transformada num objecto genérico pelo middleware disponível na máquina A
- Finalmente o objecto genérico é transformado numa mensagem enviada pela rede

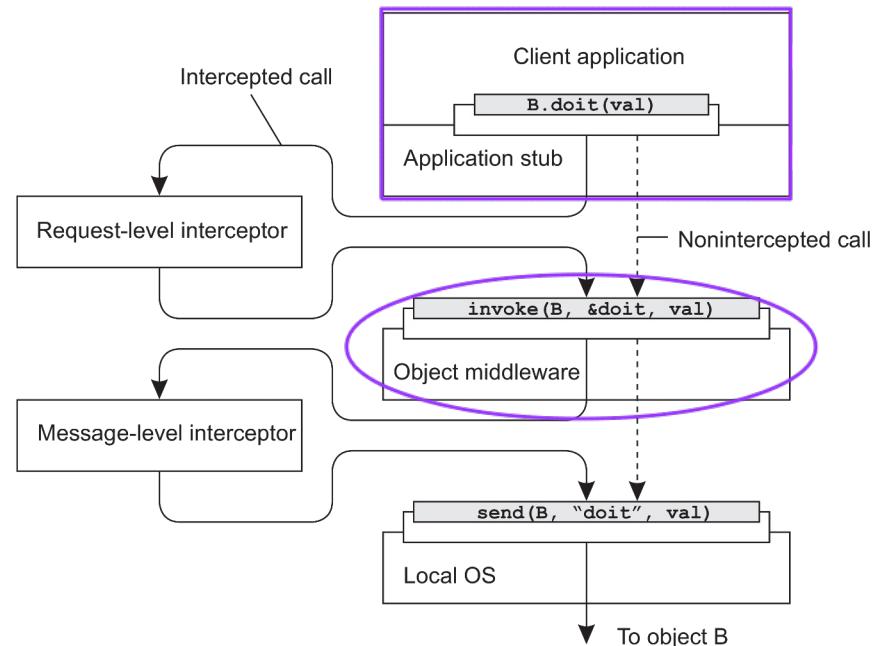
↳ Abstração ao nível dos objetos

Pode ser posto como uma operação que afeta diretamente o código do cliente, não está apenas na saída... Por exemplo, fazer o marshalling da informação a enviar... Isto também pode ser visto como abstração ao nível dos objectos em (Object Oriented Architecture)

→ Pode se excluir a uma biblioteca exterior ...

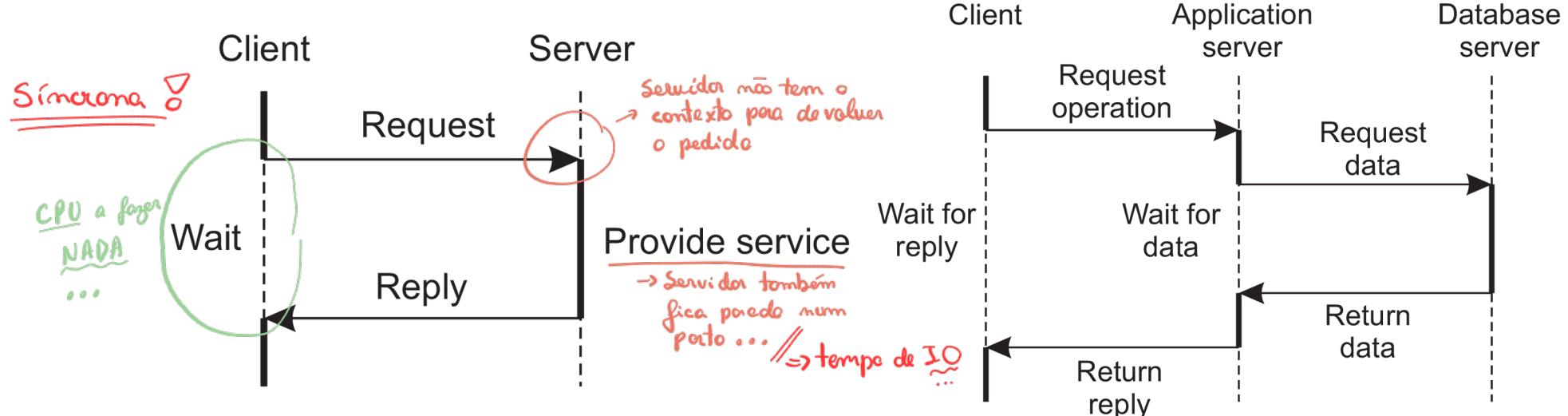
Objeto → JVM → string de Bytes

Em sistemas distribuídos usa-se muito o JVM mas depois à problemas de dependências pois usamos arq. de Objectos



Arquitecturas de Sistema (Organizações centralizadas)

Anarquetaura Cliente - Servidor!



Cliente/Servidor



```
# server.py

from socket import *
from socket import * # Não quero saber a fragmentação de pacotes
                    # (TCP) (Antes que seja UDP)
s = socket(AF_INET, SOCK_STREAM) # Pode viajar pela Internet
                                # Informo SO que quero uma Socket
                                # com protocolo IP
s.bind(('', 8888)) # bind to port on this machine → Se alguém estiver a usar ele
                    # → Assim não precisamos de permissão (0 a 1024 precisavam) avisar! ""
s.listen() # Escuta...
          # Aqui criou
          # um Socket

(conn, addr) = s.accept() # returns new socket and addr.
                          # Nova Socket para cada cliente [Estrutura (conn, addr)]
while True: # forever

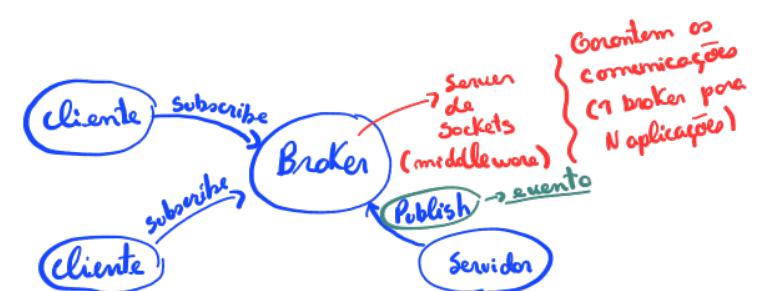
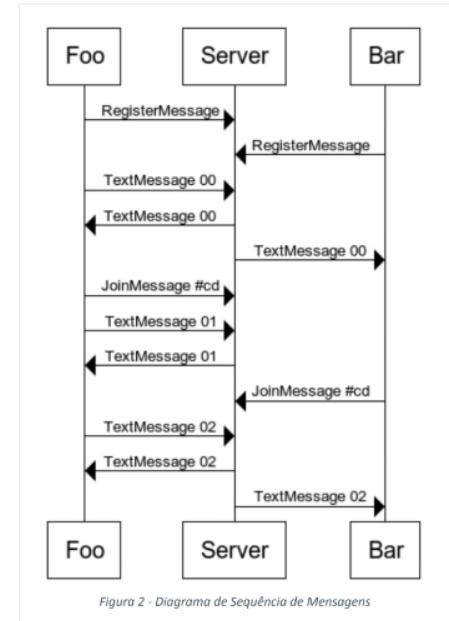
    data = conn.recv(1024) # receive data from client

    if not data: break # stop if client stopped

    conn.send(b"*" + data) # return sent data plus and "*"

conn.close() # Em byte, pois em
            # python as strings são
            # de 2 bytes... Temos
            # de fazer Encode e Decode
```

Server reúne os dois clientes através de Sockets!
Nossa Middleware para abstração...



Cliente/Servidor

```
# cliente.py

from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect(('127.0.0.1', 8888)) # connect to server (block until accepted)
s.send(b"Hello, world") # send some data
data = s.recv(1024) # receive the response
print(data) # print the result
s.close() # close the connection
```

Para usar ip temos de estar na MakerLab
Nat da UA não permite (Firewall)

→ Dentro do mesmo PC (localhost)

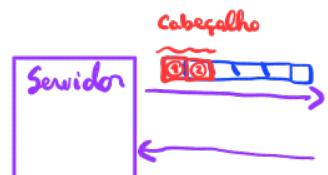
array de bytes

→ Não devia ter nem menos nem mais... (quanto menor melhor)

/// Se o Server morrer
2 rapidamente ele
pode associar os 2
numa ...

Utilizar no Projeto (eficiente)

① Projeto: tomonto?,,



① tomonto do conteúdo

→ Outra solução ...

Usar um buffer circular e
uma string de bits (bytes) que
demarcam o fim de cada pacote!

→ ...

Todos os mensagens têm
o mesmo tomonto!
(pode haver desperdício)

Transmissão é responsável
pela fragmentação ...

Não é o Middleware ...

dgomes@ua.pt

→ HTTP ...

Cabeçalho com content length!,,
(cabeçalho tem 2 terminações de
fim) Usamos um buffer circular...

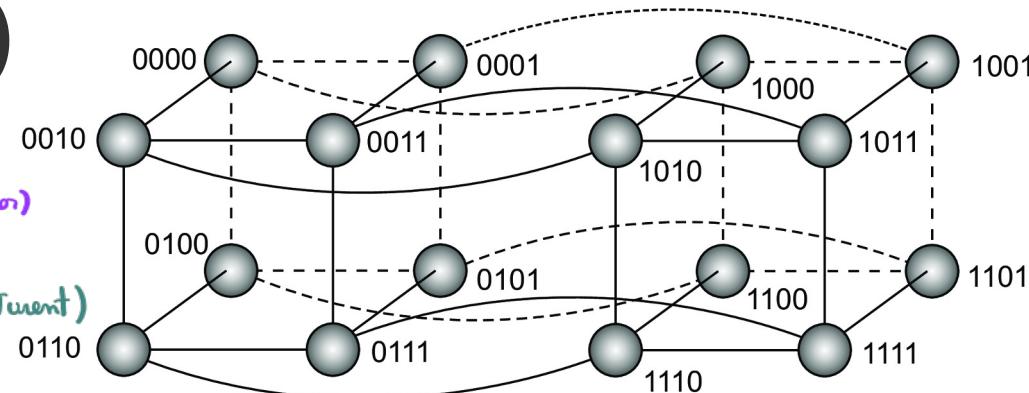
→ Lemos o conteúdo com o CL!

Arquitecturas de Sistema (Organizações descentralizadas)

→ Vou me ligando até haver uma resposta (utilizados pela World Wide Web)
(não estruturado, sem servidores)
⇒ inúmeras computadoras

- Peer-to-peer (e.g.: Torrent)

P2P → tem vários arquiteturas



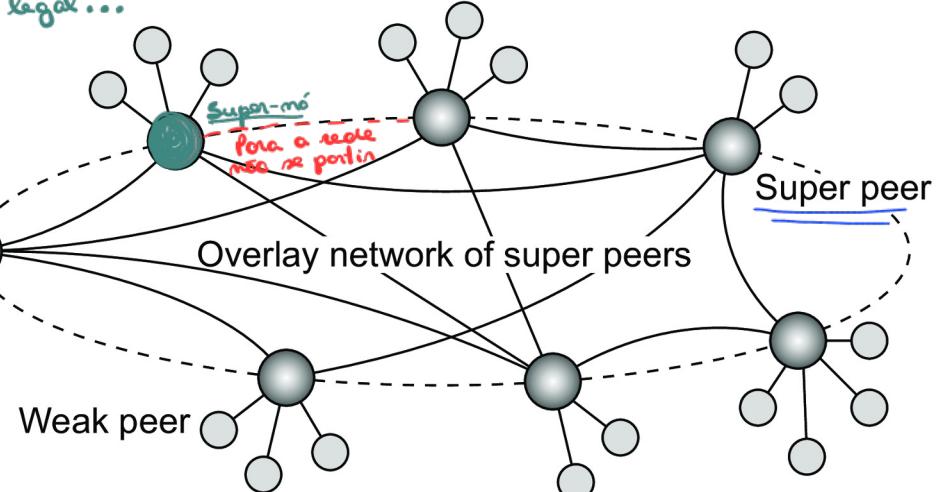
- Estruturado → Nós ponta a ponta (todos ligados) mas existem Super-Nós! (Gnutella, os Super-Nós guardam info. de todos os nós) ⇒ esta rede é legal...

- Não Estruturado

- Flooding → Para encontrar, e utilizamos o Time To Live ...

- Random walks → Encomenda só para alguns (aleatório)

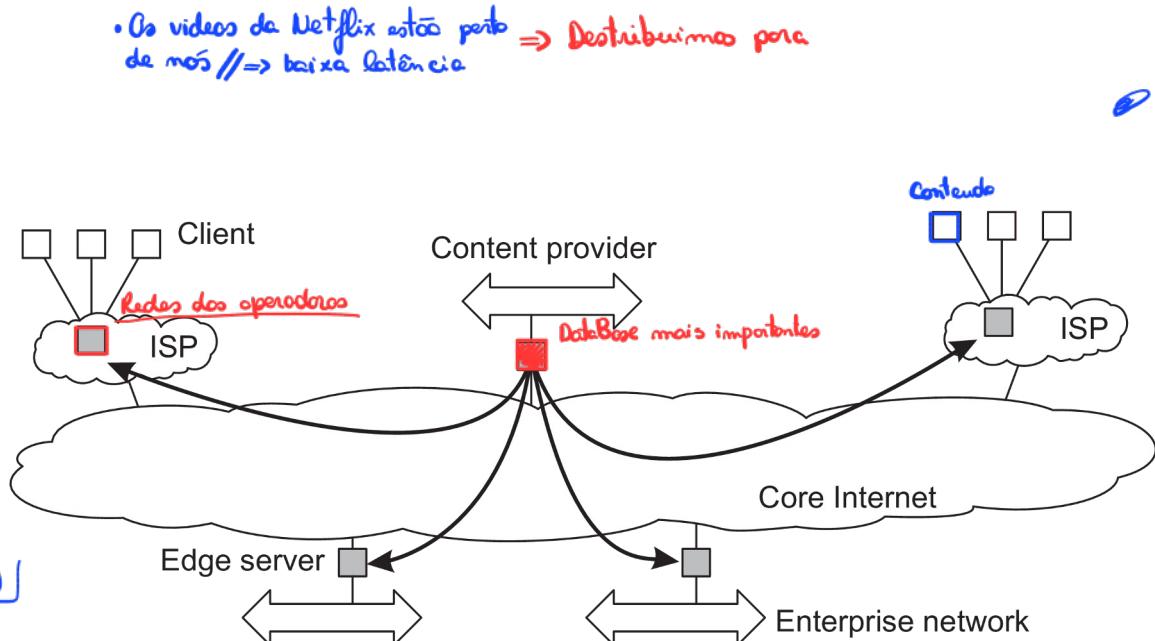
- Hierárquico → Mais lento, mas nunca falha por causa do TTL



Arquitecturas de Sistema (Organizações hibridas)

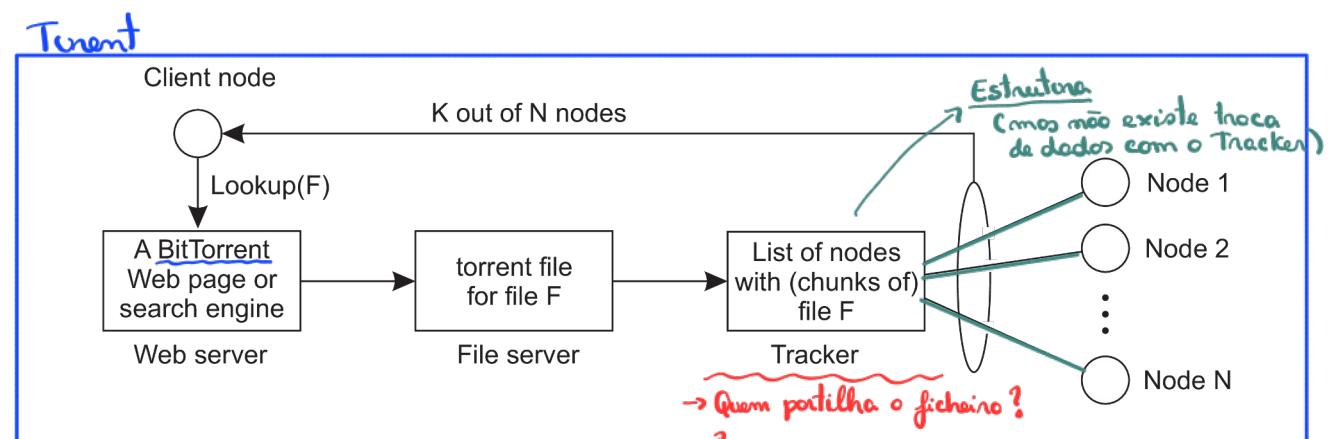
- Sistemas Edge-server

(e.g.: AKAMAI) → • CDN's
Rede de distribuição de conteúdo ...



- Sistemas distribuição colaborativa

• BitTorrent
(ilegal)





Imagens retiradas de
<http://distributed-systems.net>