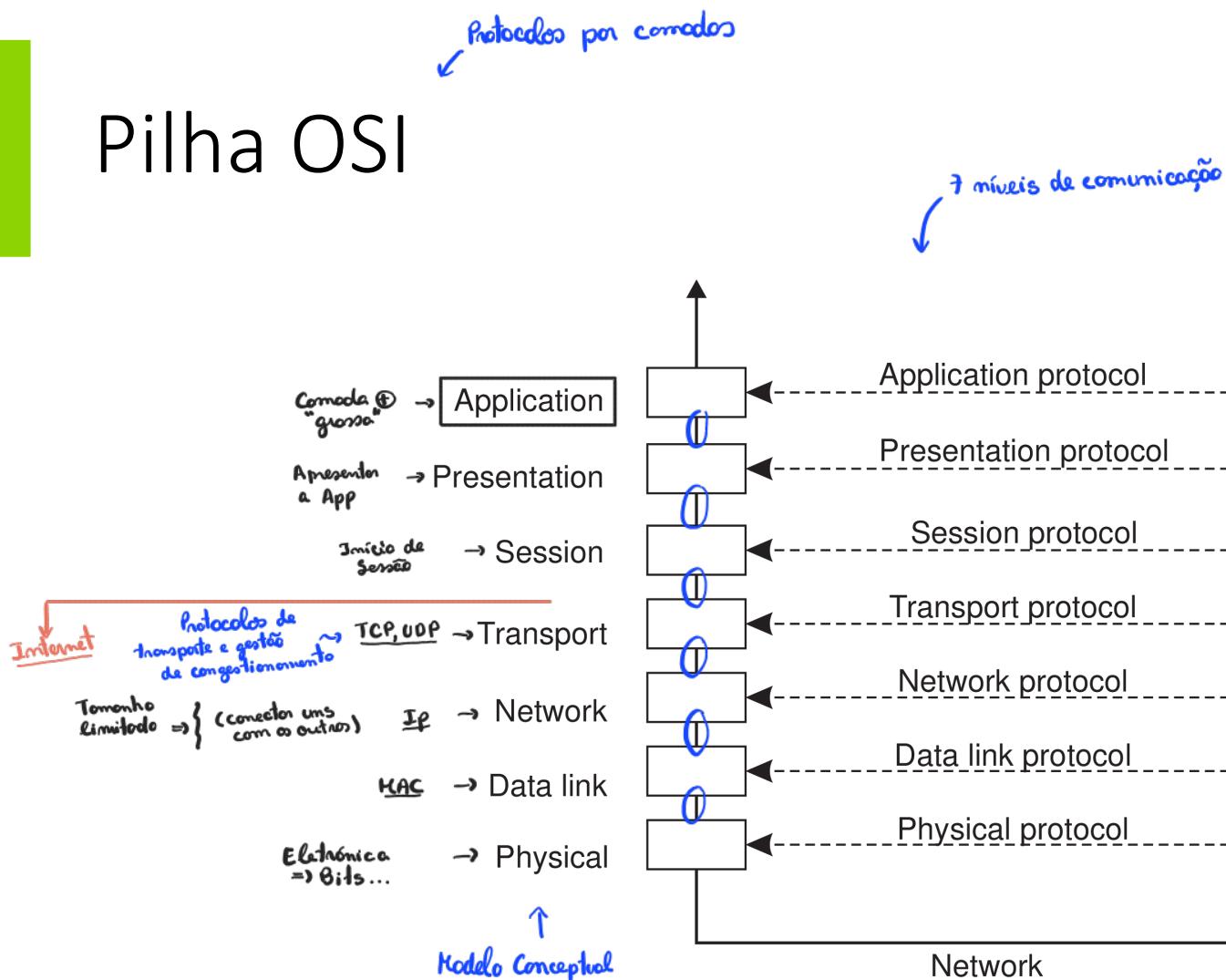


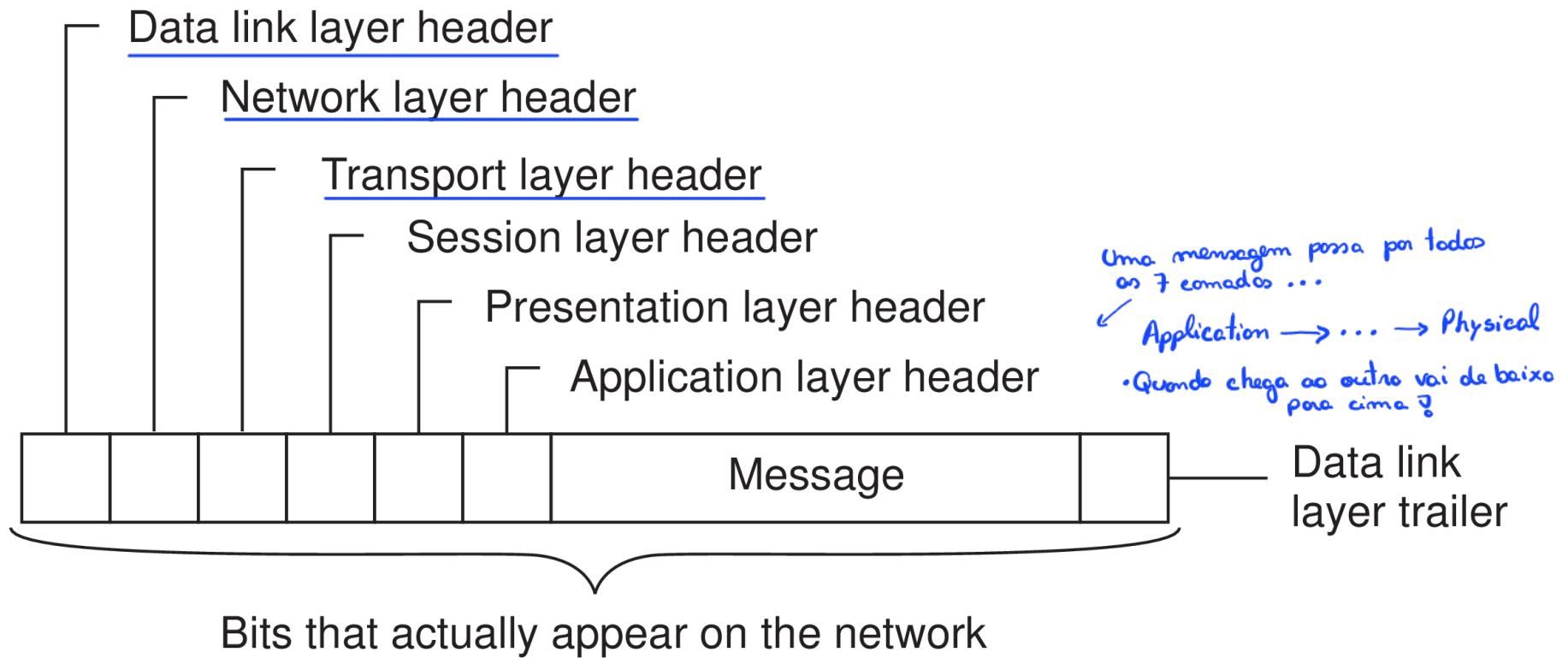
Comunicações



Pilha OSI



Pacote de dados (*Simplificado*)



Camadas de baixo nível

- Camada Física:
 - codificação de bits
- Camada de Ligação:
 - codificação de uma serie de bits num frame com controlo de erros e fluxo
→ Controlamos o fluxo!
- Camada de Rede:
 - como pacotes são roteados numa rede de computadores

Camada de Transporte

- Camada de suporte à comunicação da maioria dos sistemas distribuídos
 - Protocolos principais:
 - **TCP**: orientado à ligação, fiável, orientado aos fluxos de informação (*stream*)
 - **UDP**: comunicação não fiável (best-effort)
 - SCTP: sólido...
- O que nós normalmente configuramos
- Pega na mensagem e parte em pedaços numerados, e envia!
- ↑
Netflix é em stream...
- ④ Rápido (não tão fiável)
④ Controlo de fluxo (ele não para para retransmissão)
- e.g.: Jogos
Chamadas de Voz
Streaming

Camada Middleware

Todos estes
middlewares acabam por
substituir os camadas de
Session & Presentation

Exemplos:

- DNS (Application Layer)
- Authentication
- RPC → Remote Procedure Call

↳ Fornece a facilidade de executar
um processo num computador
remoto

→ Acesso transparente ...

- Middleware permite partilha de serviços e protocolos comuns entre diversas aplicações

→ Mecanismos funcionalidades partilhados

- Conjunto rico de protocolos de comunicação
- (Un)marshaling dos dados, necessário à integração de sistemas
- Protocolos de endereçamento, para partilha fácil de recursos
- Protocolos de cifragem, para comunicações seguras
- Mecanismos potenciadores de escalabilidade tais como replicação e caching

??
Muito Importante

→ marshaling: (usar JSON)

• converter as estruturas
de dados em bytes

→ unmarshaling:

• converter de bytes
para estruturas

{ Transporte

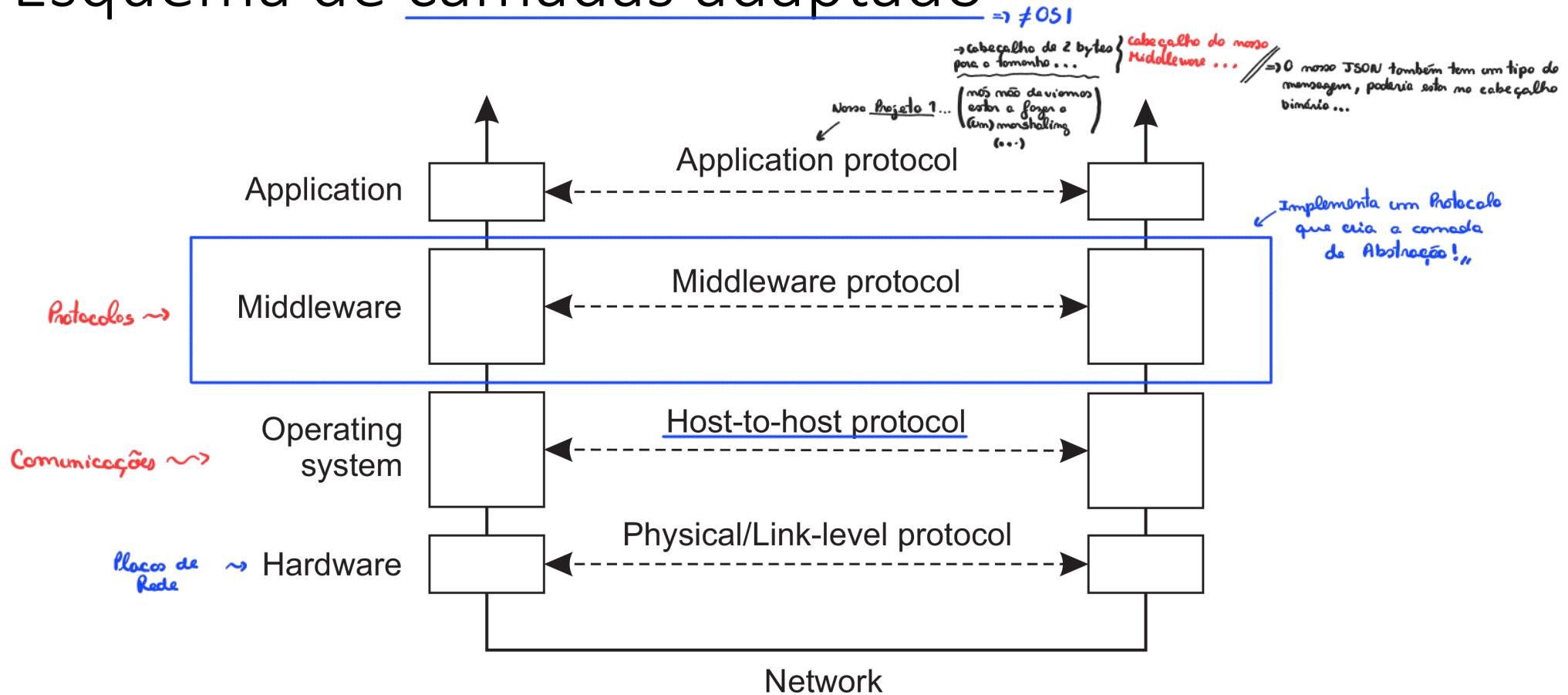
Recebo e espero que
ponha computar ...
↓ (Comunicação Assíncrona)

→ Middleware faz este serviço

⇒ fiável, mesmo quando à perda de
informação ...

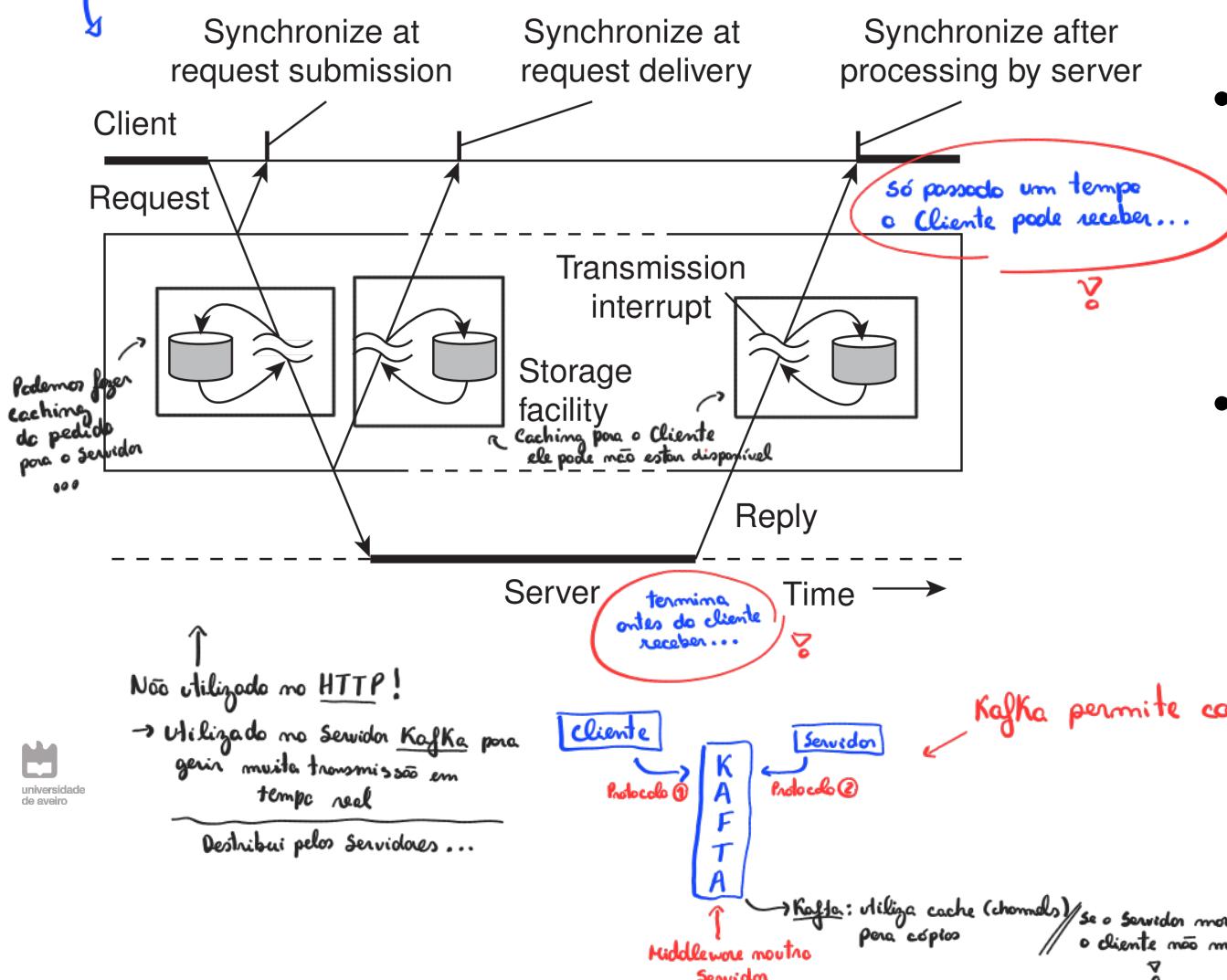
- Comunicação
- Conversão
- Segurança
- (...)

Esquema de camadas adaptado



Tipos de comunicação

Persistente!



- Transiente vs Persistente

- Servidor descarta mensagens quando estas não podem ser entregues (apenas guarda durante a comunicação) Transient
- Servidor guarda mensagens até poderem ser entregues (e.g., email) Persistent

- Assíncrono vs Síncrono

- (Kafka)
 - ele envia e continua a executar
 - ele executa e fica à espera...
 - Mensagem é guardada temporariamente pelo Middleware!
- À espera:
 - middleware avisa que a sessão terminou
 - só a mensagem chegará
 - só receber resposta...

→ Uma óbvia combinação é Síncrono + Persistente

Kafka: utiliza cache (chomps) para cópias

se os servidores morrem o cliente não morre

Cliente/Servidor

- Modelo Cliente/Servidor é baseado no modelo de comunicações transiente síncrono.
 - Cliente e Servidor têm que estar activos durante o tempo de comunicação
 - Cliente faz pedido e **bloqueia até receber resposta**
 - Servidor essencialmente espera por pedidos, e processa os mesmos subsequentemente.
- Problemas:
 - Cliente não pode fazer nada enquanto espera *→ e-mail...*
 - Falhas têm que ser processadas imediatamente (cliente em espera)
 - Modelo pode não se adaptar ao problema (ex. e-mail)
Queremos que seja Persistente!

Mensagens

- Middleware orientado a mensagens *send() não bloqueia...*
- Comunicação **Persistente e Assíncrona.** *?? Troy Transparência !*
- Processos trocam mensagens, que são colocadas em filas de espera
- Emissor não precisa de esperar por uma resposta imediata
- Tolerância a falhas disponibilizada pelo Middleware. *==*

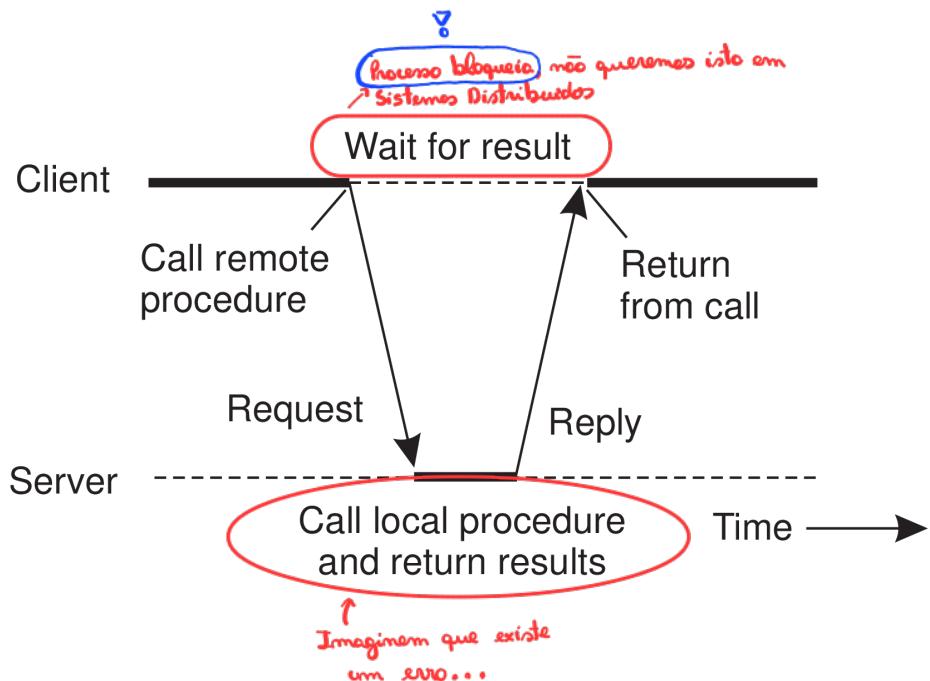
O standard:

- gRPC \Rightarrow
 - Lida com a diferença dos formatos
 - Fazem a conversão todos dos tipos
 - Mas, são comunicações Síncronas

Maior parte da indústria utiliza!

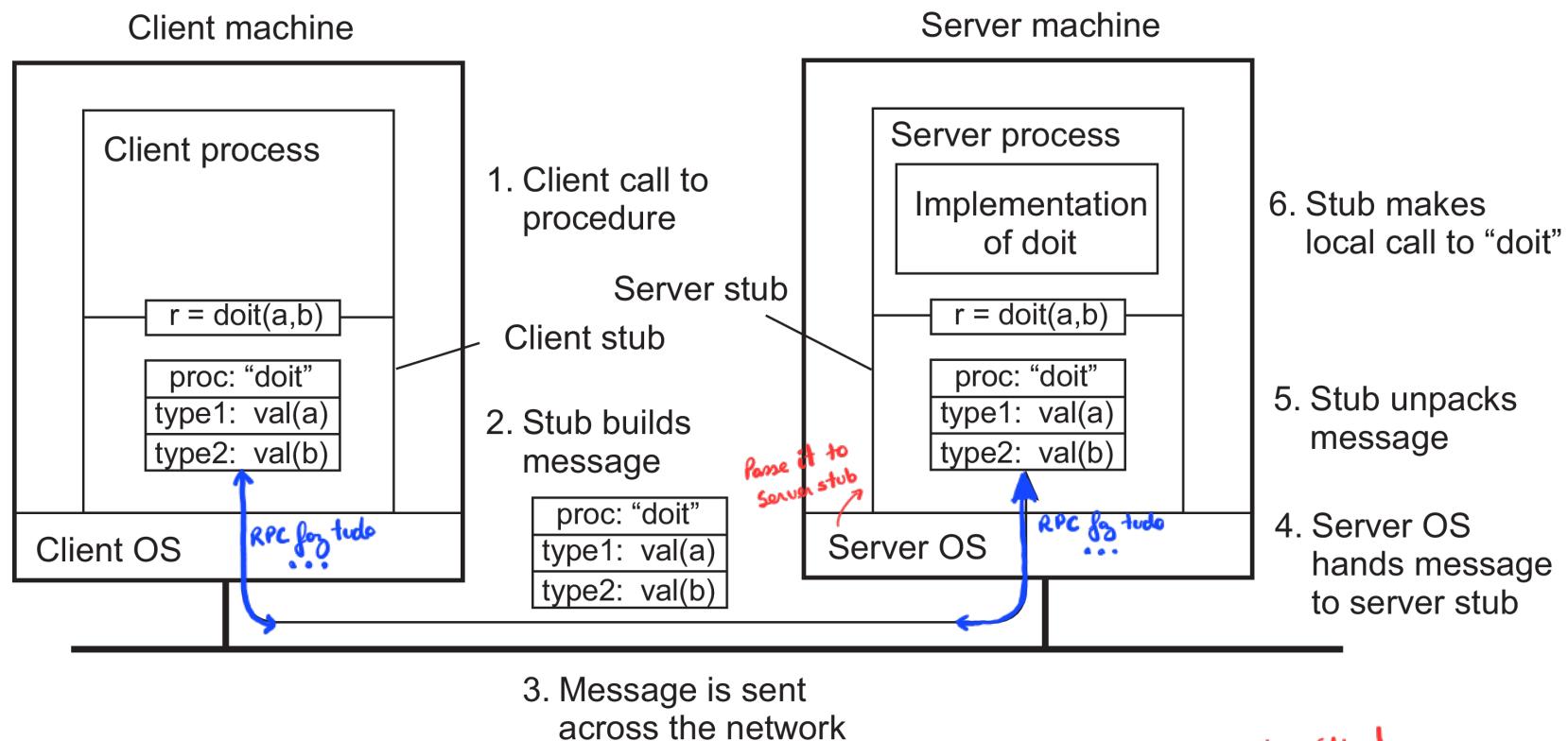
- Não precisamos de saber falar moda...

Remote Procedure Call (RPC)



- Programadores estão familiarizados com modelo procedural
- Procedimentos quando bem construídos operam de forma isolada
- Não existe nenhuma razão para não executar procedimentos noutra máquina.

Operação de um RPC



• Como os bytes vão ser interpretados pelo servidor!

RPC: passagem de parâmetros

- Cliente e Servidor podem ter representações diferentes dos dados (*little endian vs big endian*)
- Empacotamento de um parâmetro significa transformar um valor numa sequência de bytes
- Cliente e Servidor têm que concordar com a mesma codificação
- Como são representados os valores básicos (*int, float, char*) ?
- Como são representados dados complexos (*arrays, objects*) ?

RPC suporta isto!
• Poucos utilizões pois é preciso programar de forma assíncrona ...

Programação assíncrona:

loop

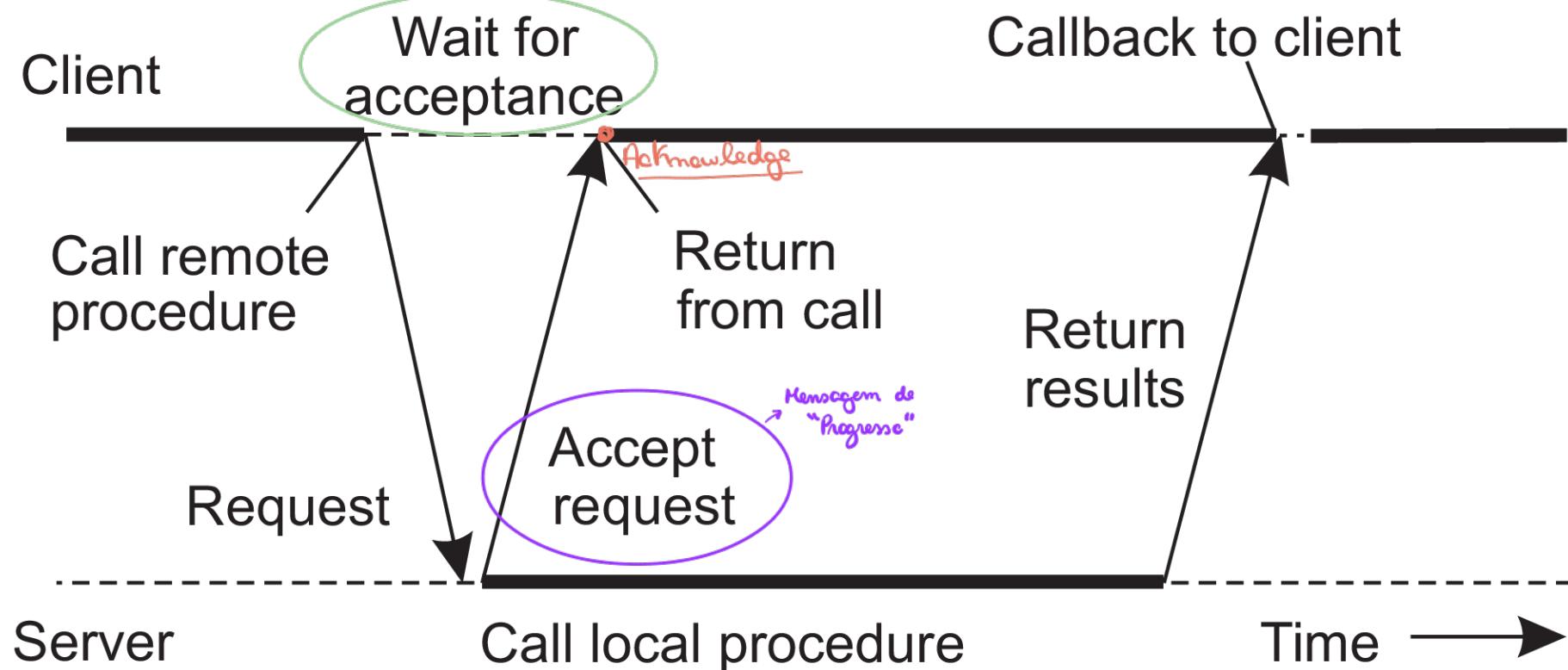


• Onde só queremos executar uma função
• Vai para a fila
• Vai para a fila

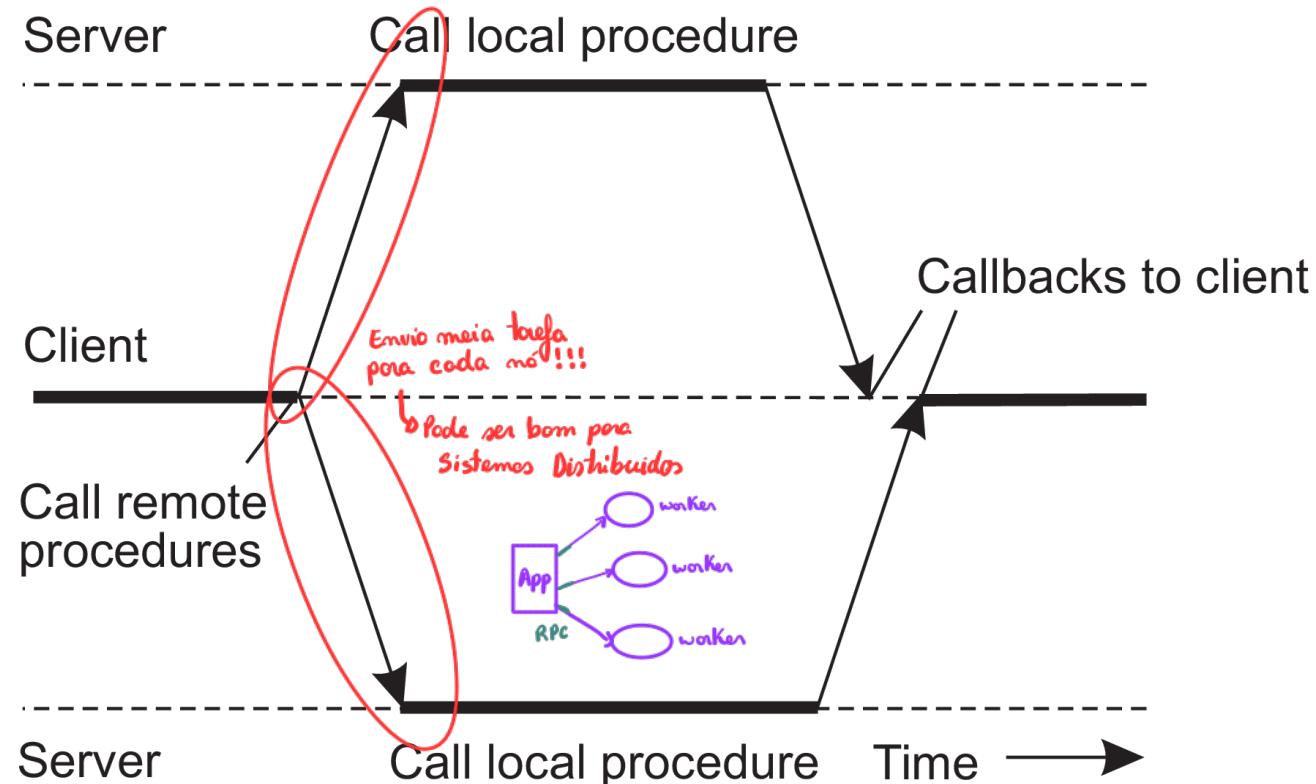
• Assim, não fico bloqueado ...

RPC's assíncronos !

Muito menor que o do RPC...



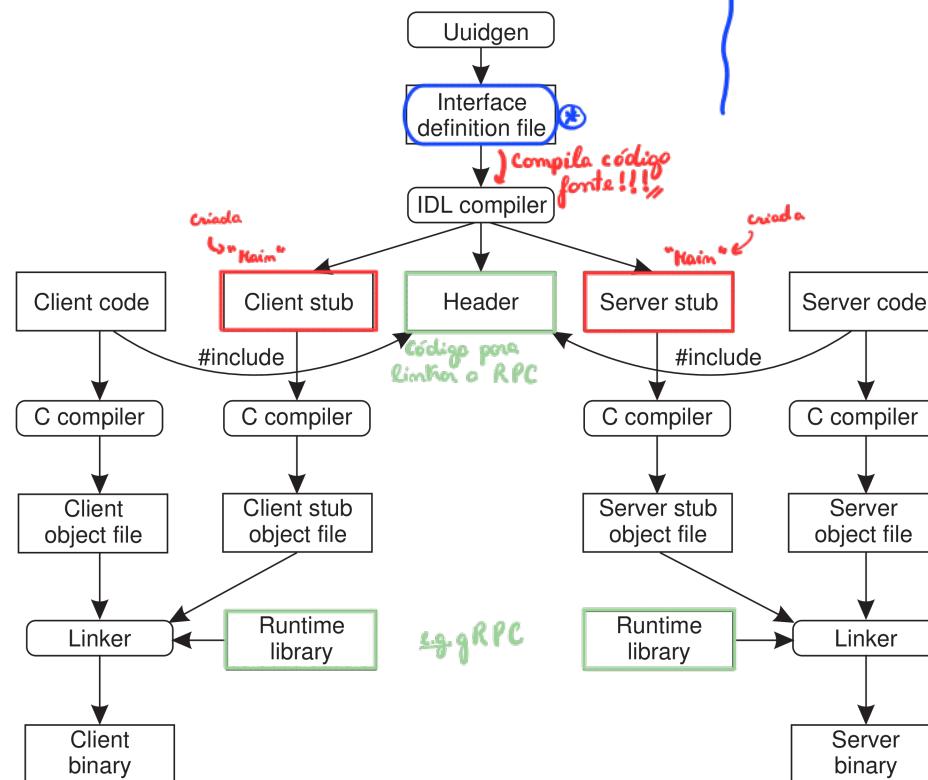
Múltiplas chamadas RPC



RPC na prática (em Linguagem C)

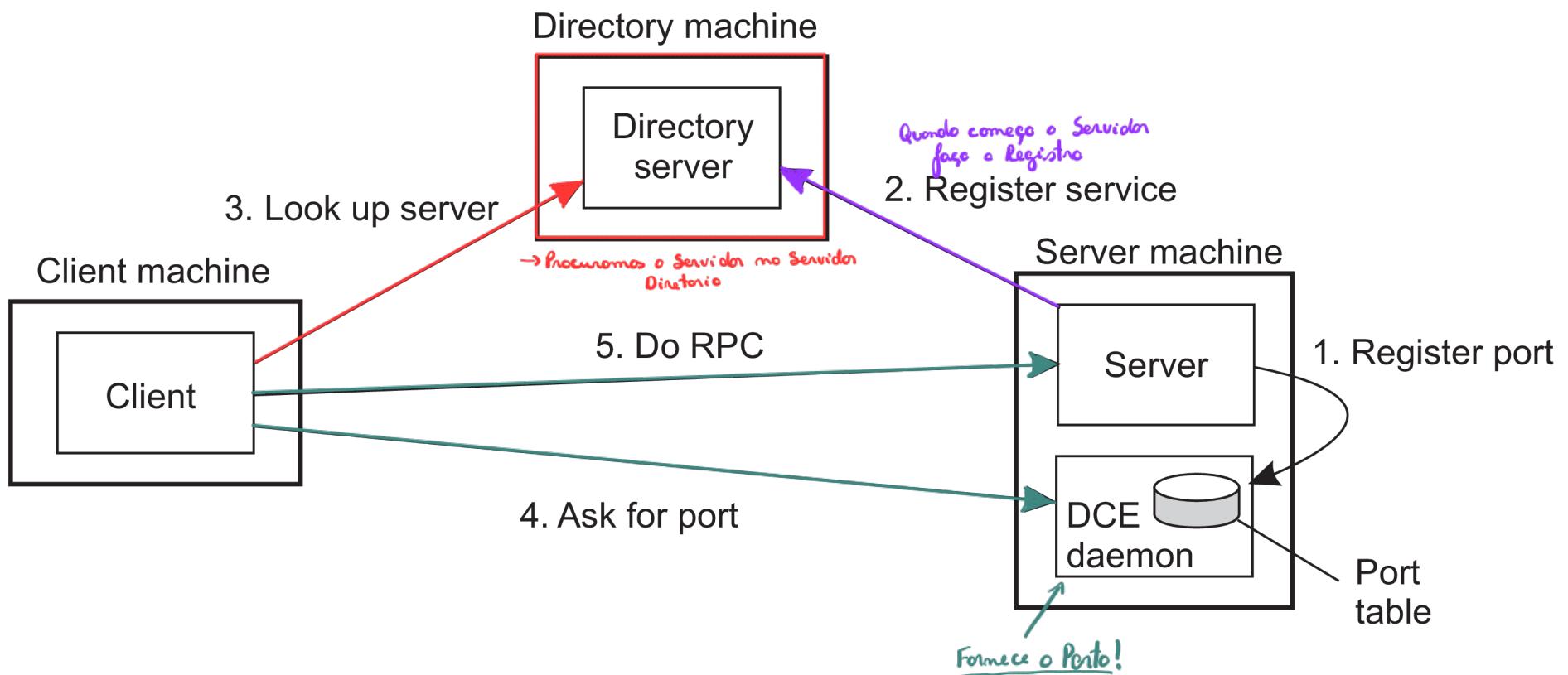
→ A mais complicada !!!

→ termos de compilador! } }=> Gerar identificadores únicos para cada função
ficheiro de interface @, usado pelo compilador de RPC!

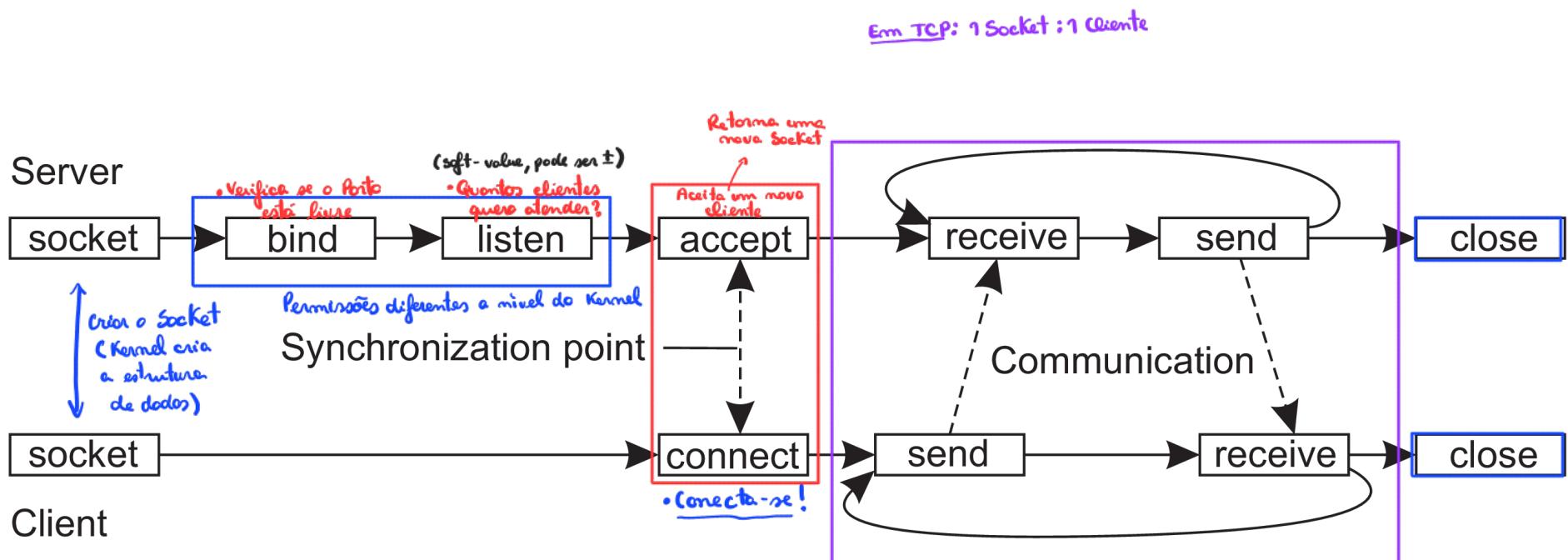


Ligaçāo Cliente-Servidor (DCE)

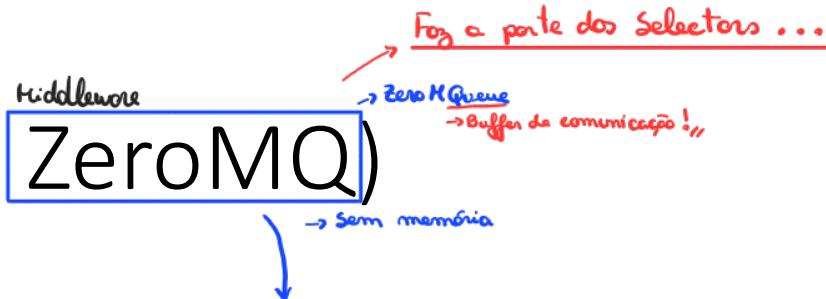
→ Como é que o Cliente RPC encontra o Servidor? Queremos uma forma automática...



Sockets TCP



Sockets (com ZeroMQ)



- Camada de abstração de alto nível para sockets
- Criação de pares P, Q (um para enviar outro para receber)
 → Envio, Receber
- Toda comunicação é assíncrona.

- Padrões:
 - Request-reply
 - Publish-subscribe
 - Pipeline
 - Vários pedidos
 - Vários respostas

→ Faz subscribe e vai receber quando alguém fizer Publish desse conteúdo ...

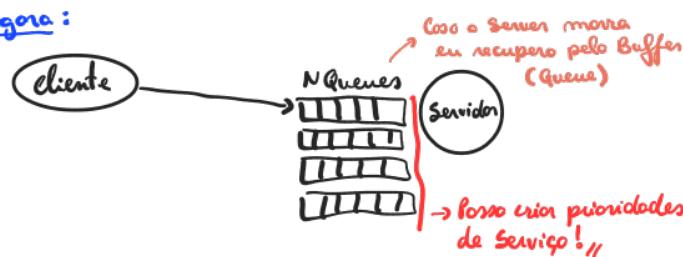
Comunicação Persistente

↓
Tipicamente usado quando
os mensagens podem
demorar tempo...

Queues (Filas)

Ate agora: Se o Servidor morrer os Clientes todos morrem !

Agora:

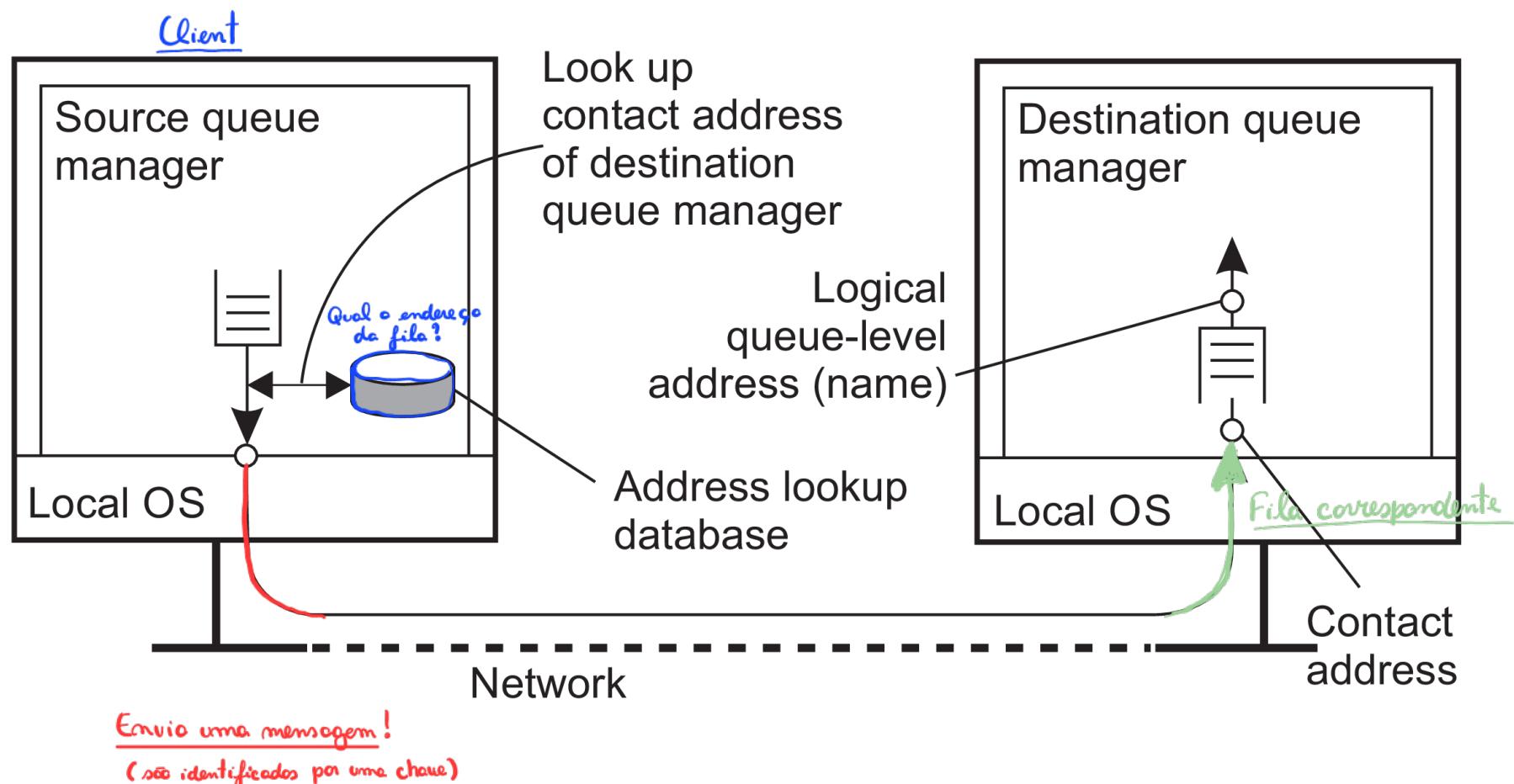


→ Cada aplicação tem a
seu próprio fila ...

- Comunicação assíncrona persistente através de um middleware de filas (queues). As filas são buffers nos servidores de comunicação
- Operações:
 - Put – adiciona uma mensagem a uma fila específica
 - Get – bloqueia até que haja uma mensagem na fila, remove a primeira mensagem
 - Poll – verifica a existência de mensagens numa dada fila, remove a primeira mensagem. Nunca bloqueia
 - Notify – instala um *handler* que será chamado quando uma mensagem for colocada numa determinada fila.

Modelo Geral

- Gerir esses filos como um administrador pode ser complicado ...



Message Broker

→ negociador

Muito utilizados:

- ActiveMQ (Mais antigo...)
- Kafka (Open Source)
- rabbitmq (funciona em computadores pequenos)

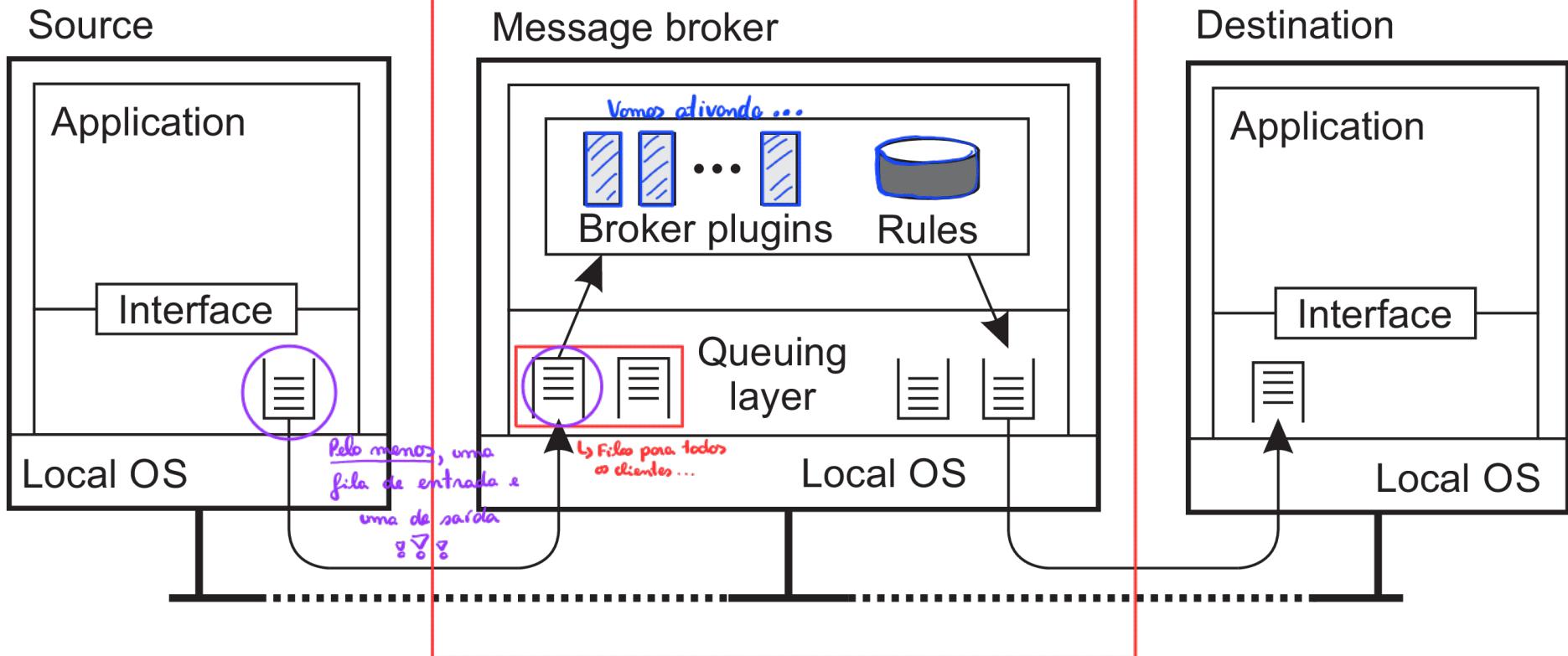
Evita termos $O(N^2)$
wrappers!

⇒ Servidor que gera o Servidor e o Cliente
↳ Único ponto de falha
"Eficientes" / "Estáveis"

- Num sistema de message queueing (MQ) é assumido um protocolo de mensagens comum.
- Funções do Broker:
 - Transformar mensagens no formato correcto
 - e.g.: JSON
 - AMQP → Muito utilizado!
 - Application Gateway
 - Ponte de entrada exposta na fila do Message Broker
 - Funcionalidades de roteamento (ex. publish-subscribe)
 - uma mensagem para vários filhos (e.g. efeitos de log)
 - Ponte da Segurança (ele é um nó central)
 - cifra
 - Autenticação
 - ...

Arquitectura de um Message Broker

Tudo ligado Aqui!



→ Poderia não ter queues...
(poderia ser só um formador)
de mensagens

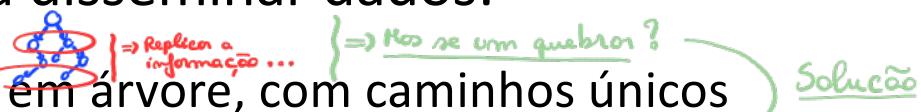
Multicast em redes overlay não é ótimo!

Application-level Multicasting

→ Tenho uma lista de quem está interessado! //

- Organizar nós de um sistema distribuído numa rede *overlay* e usar essa rede para disseminar dados:

- Tipicamente em árvore, com caminhos únicos
 - Cada caminho entre os nós, só chega uma vez! // (árvore)
- Alternativamente em mesh, requere algum tipo de routing



Solução

Um "grafo", chego
ao mesmo nó com
vários caminhos

⊕ Fidável

A nível da Aplicação ☺☺☺
→ Mas cuidado com a redundância das mensagens recebidas



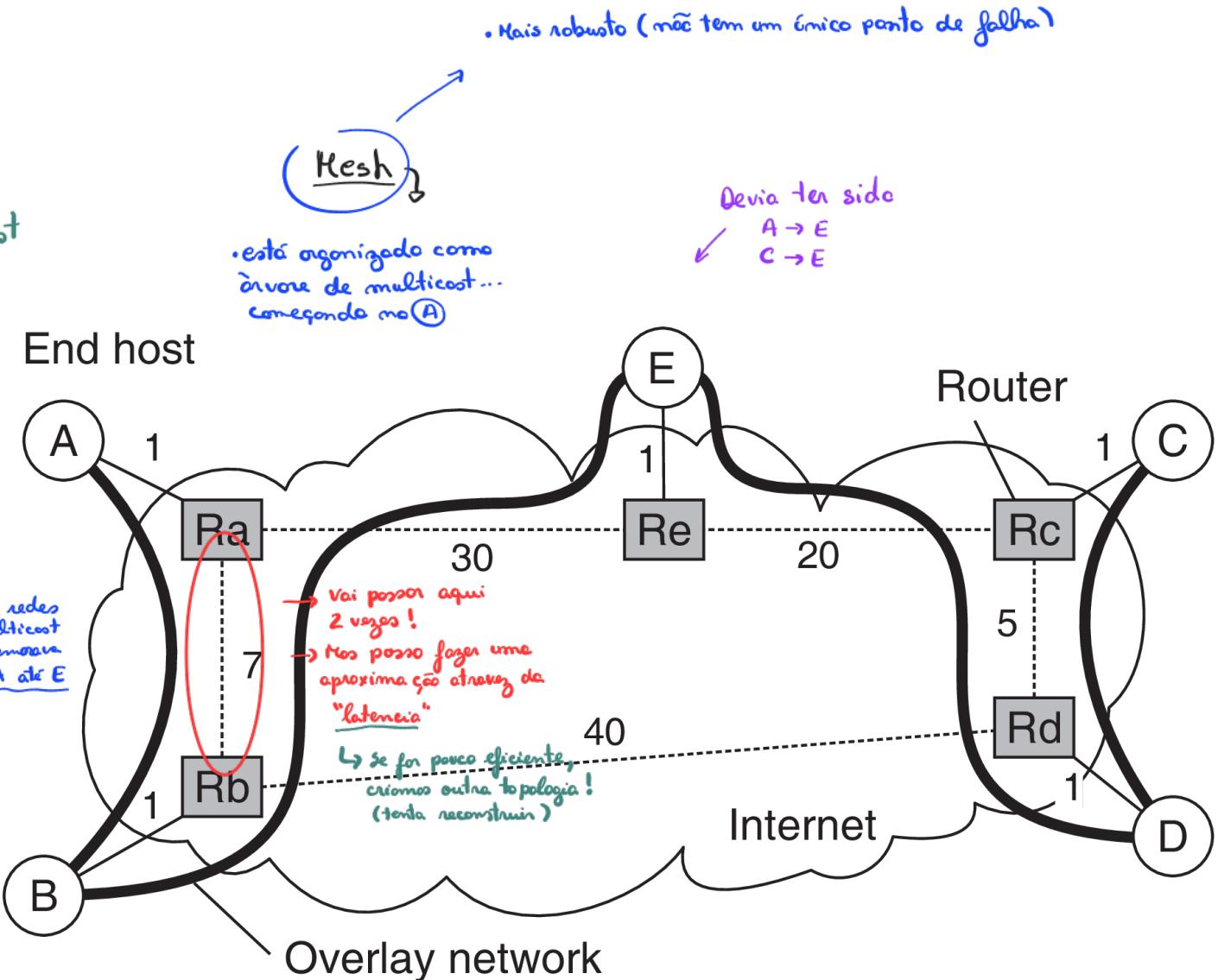
Application Level Multicast

ALM: custos

Link stress: quantas vezes a mesma mensagem passa no mesmo link de rede?

Stretch: rácio entre o atraso entre o caminho ALM e o caminho de rede.

three cost: (...)



Mas pode haver loops:

→ TTL : Time To Live
 ① Reenvio só para quem está mais perto

Normalmente constrói uma rede overlay por grupo multicast
 ...
 (senão broadcast era igual a multicast)

Ignorar duplicados!

Flooding
 "envia para todos"
 → Na internet eu não consigo enviar para toda a gente!
 → Preciso de comandos de Replicação

Um nó envia uma mensagem para cada um dos seus vizinhos. Cada vizinho re-envia a mensagem para os seus vizinhos à excepção de P, e só se não tinha recebido a mensagem anteriormente.

Outros algoritmos:

- Epidimológicos ①
- Anti-entropia {→ Entropia da física}
- Rumores {→ com random generators
 → Bastante utilizado}

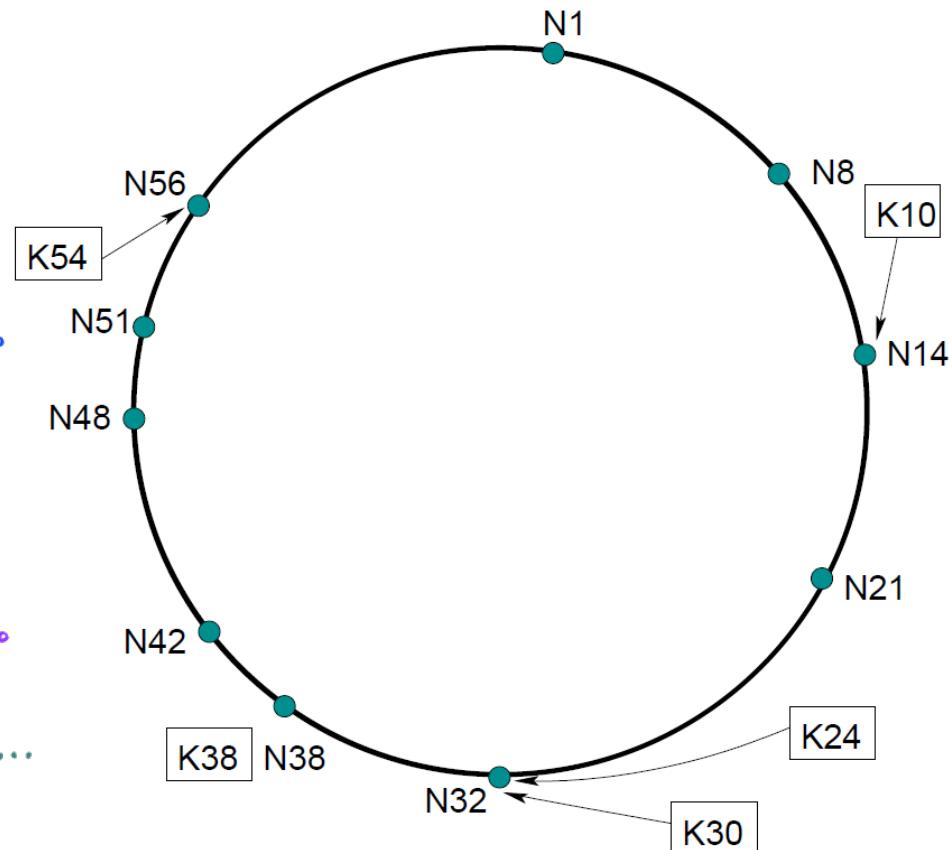
Number of nodes	$p_{edge} = 0.6$	$p_{edge} = 0.4$	$p_{edge} = 0.2$
100	~60	~30	~10
500	~300	~150	~50
1000	~600	~300	~100

Chord

- Originalmente proposto por Ion Stoica et al:
 - <https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>
Peer-to-Peer
- É um protocolo de pesquisa P2P escalável para aplicações da Internet
- É também um algoritmo para hash tables distribuídas (DHT) **Distributed Hash Table**
 -  Slots distribuídos por múltiplos computadores e.g.: Bit-Torrent
- Disponibiliza um meio determinístico de localizar recursos, serviços e pares.
- Recursos são expressos como pares Key , Value
- Balançamento de carga através de hashing consistente
 - routing tables pequenas: $\log n$ \Rightarrow Quem existe no node? (não todos)
 - routing delay baixo: $\log n$ hops \Rightarrow Para cada tudo faço saltinhos...
- protocolo para gestão de entradas/saídas rápido (polylog time)

Hash Tables: Procura $\approx O(1)$

P2P estruturado



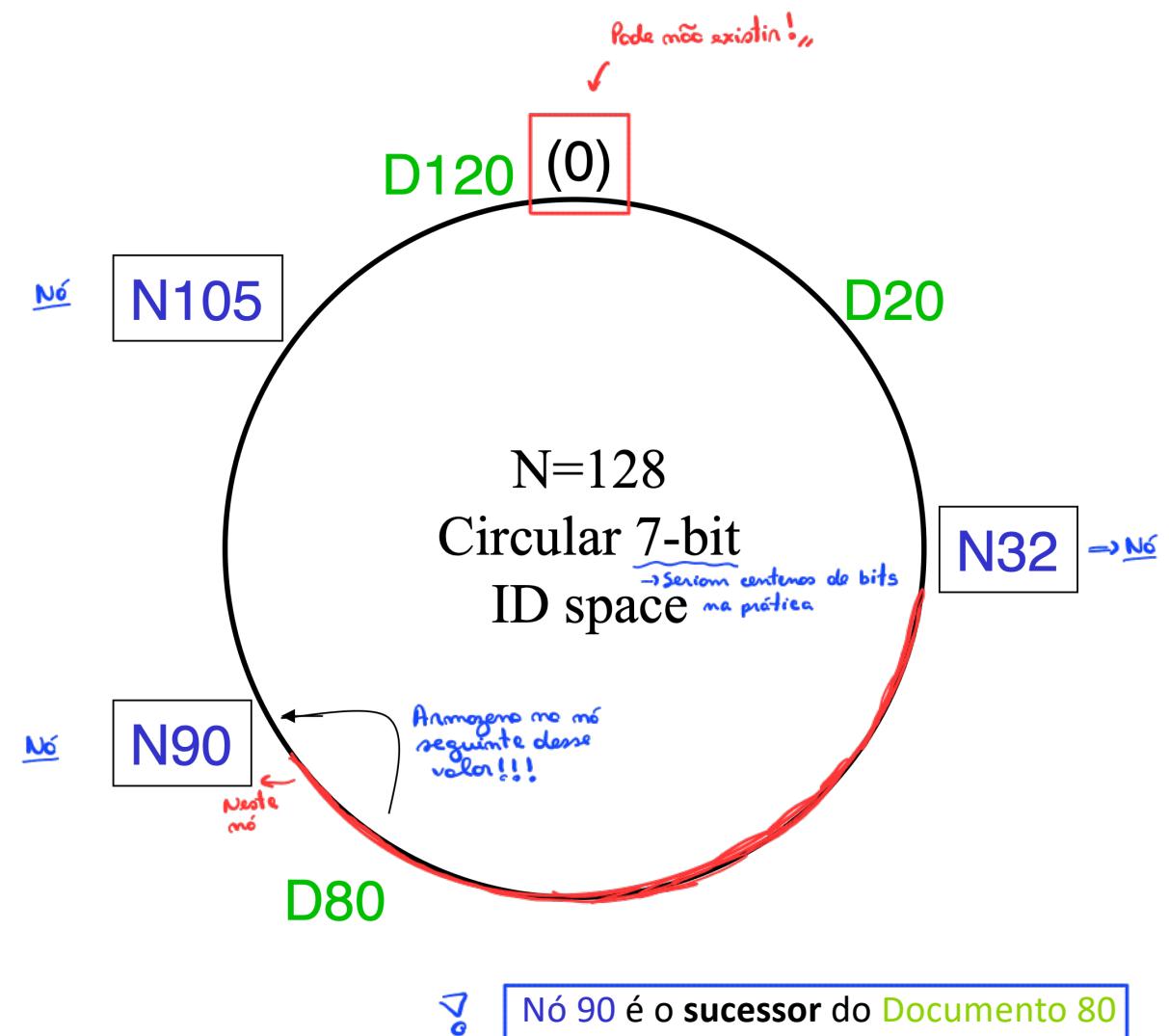
Consistent Hashing

Atribuição de identificadores de nós e objectos apartir de uma chave de **m-bit**

Ordena os nós em volta de um anél de identificadores recorrendo às chaves para ordenar ($0 \rightarrow 2^m - 1$). Ao anél damos o nome de **Chord Ring**.

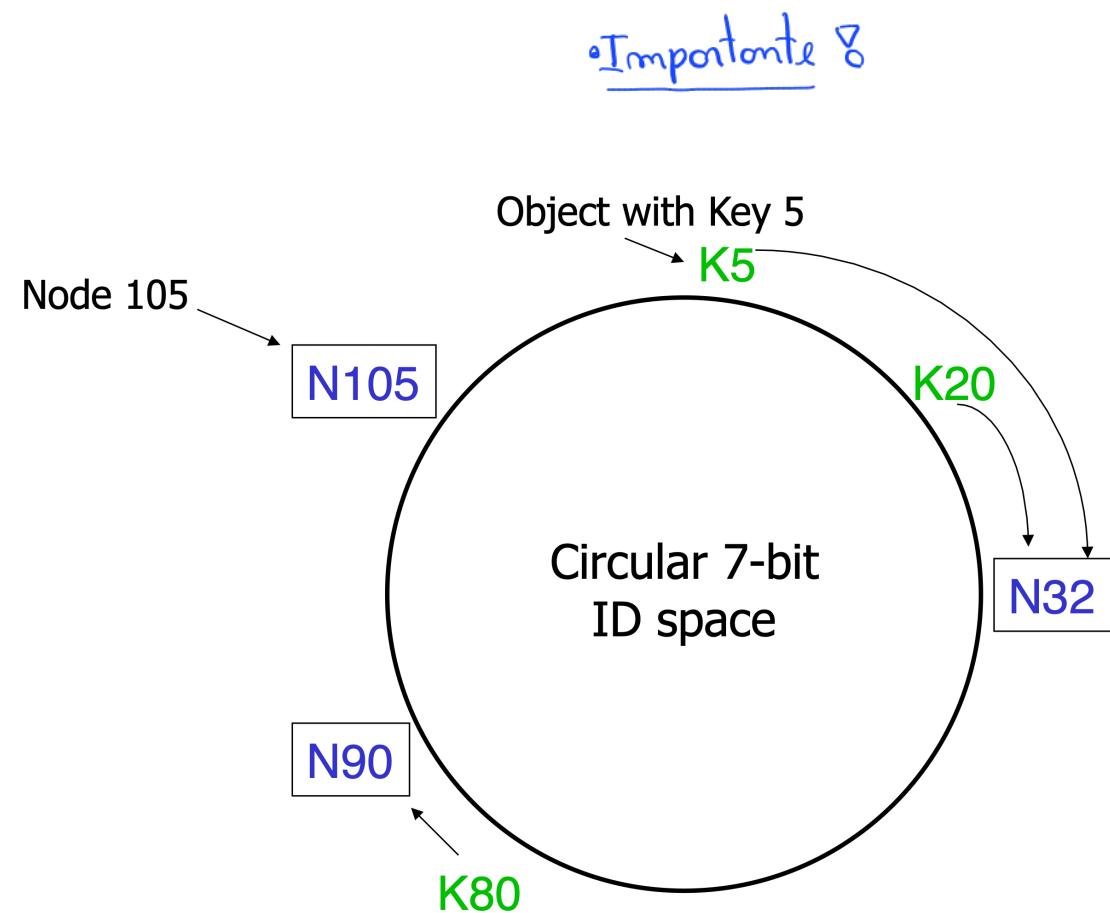
Objecto com a chave k é atribuido ao **primeiro nó** cuja chave seja $\geq k$ (chamamos de **nó successor** da chave k)

→ Na teoria teria 120 nós!
• Solução: Hash table distribuída que vai guardar no nó imediatamente a seguir ...



Consistent hashing (2)

Um objecto com a chave **k** é armazenado no seu **sucessor** (nó com chave $\geq k$)



The log N Fingers

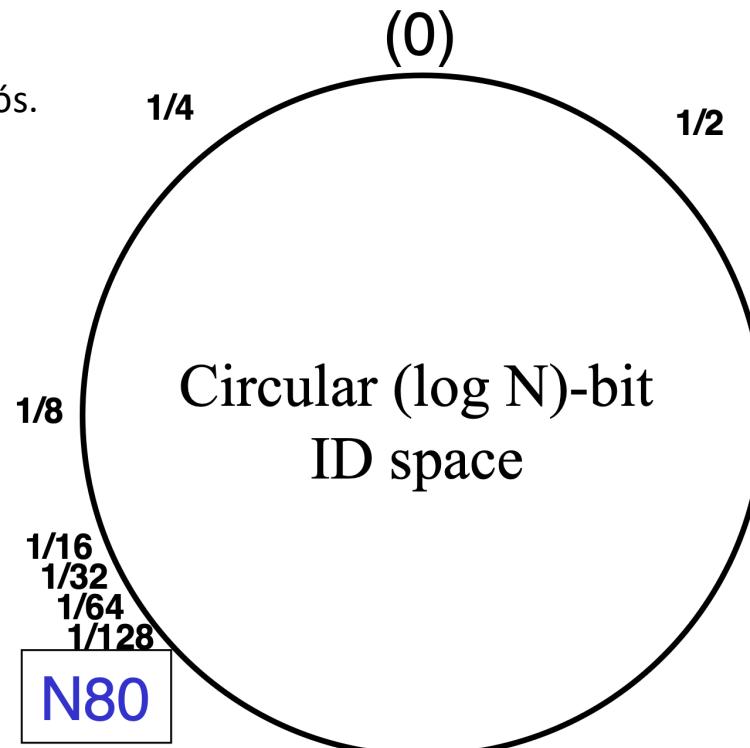
• saltos entre os nós tem de ser $\log n$!

Cada nó conhece apenas **log N** outros nós.

Pesquisa binária
de saltos

i	$n+2^i$
0	81
1	82
2	84
3	88
4	96
5	112
6	144 → 16

Finger Table ...

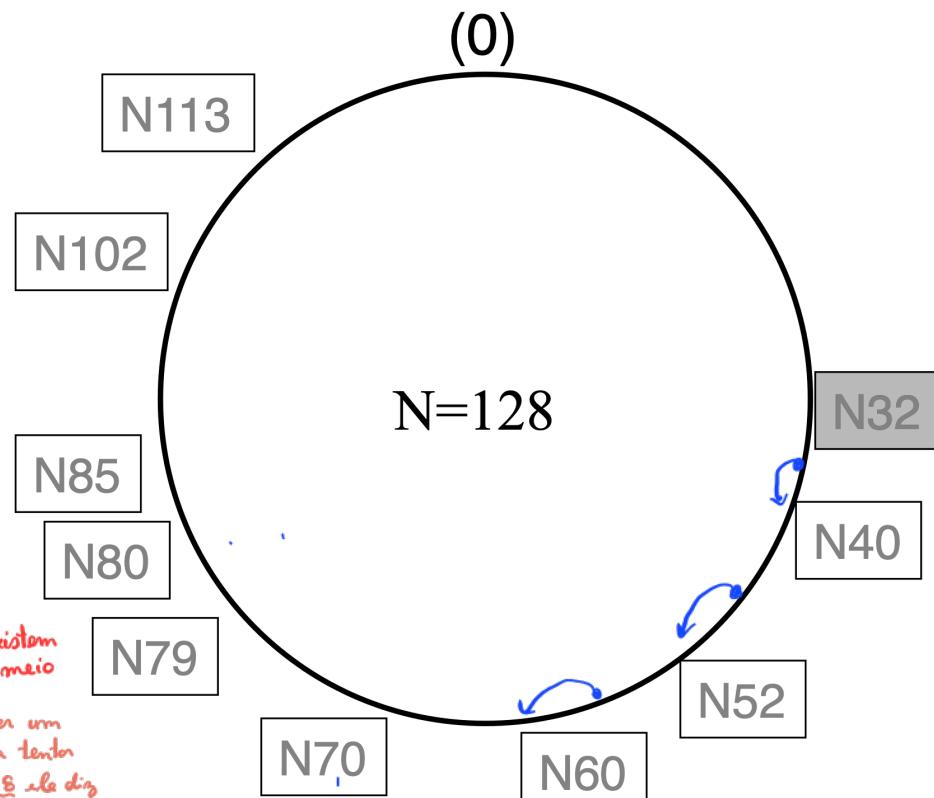


Chord Finger Table

Entrada i de um nó: **primeiro** nó $\geq n + 2^{i-1}$

i	key	node
0	$33 \rightarrow 33$	N40
1	$34 \rightarrow 35$	N40
2	$36 \rightarrow 39$	N40
3	$40 \rightarrow 47$	N40
4	$48 \rightarrow 63$	N52
5	$64 \rightarrow 95$	N70
6	$96 \rightarrow 31$	N102

Ele tem de
 saber que não existem
 nós aqui no meio
 \Rightarrow Nós vamos fazer um
 pedido "fake" para tentar
 meter um valor 48 ele diz
 que fica no 52
 \Rightarrow O github faz um ficheiro log ...



Lookup

- Node 32
 - $\text{lookup}(82): 32 \rightarrow 70 \rightarrow 80 \rightarrow 85$

↗ *guindado no nó 70*
 ↗ *Acabo por encontrar...*

key	node
$33 \rightarrow 33$	N40
$34 \rightarrow 35$	N40
$36 \rightarrow 39$	N40
$40 \rightarrow 47$	N40
$48 \rightarrow 63$	N52
$64 \rightarrow 95$	N70
$96 \rightarrow 31$	N102

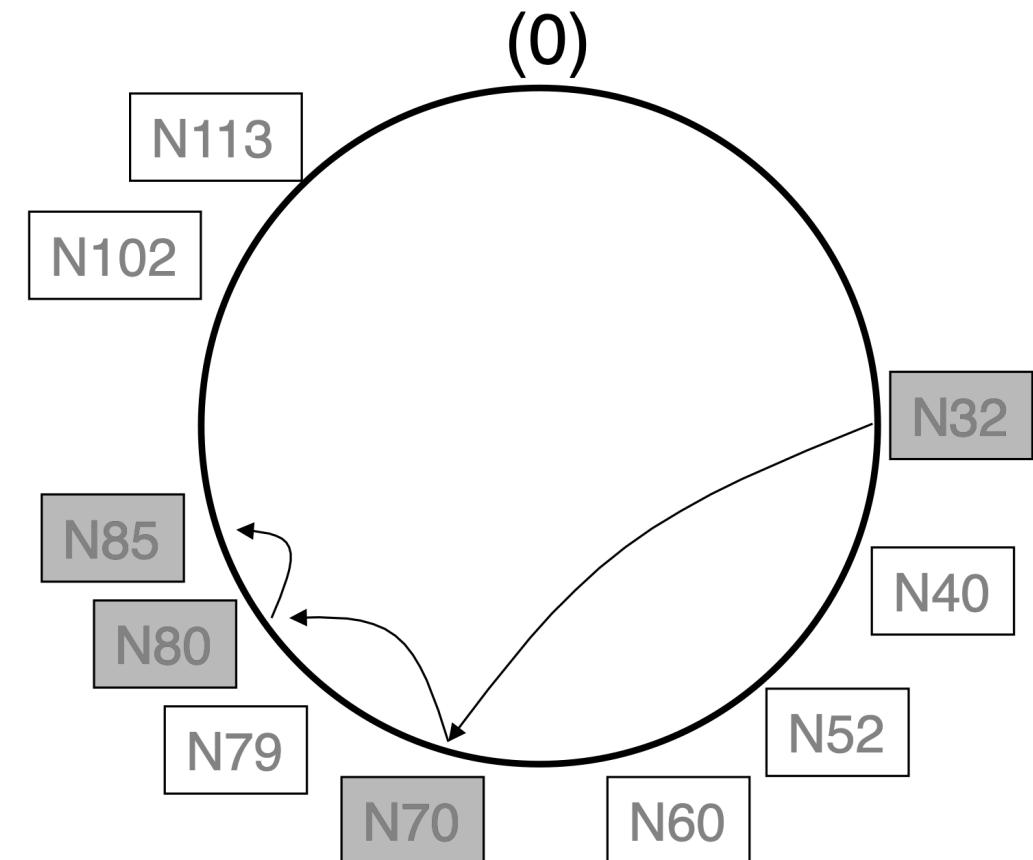
N32 Finger Table

key	node
$71 \rightarrow 71$	N79
$72 \rightarrow 73$	N79
$74 \rightarrow 77$	N79
$78 \rightarrow 85$	N80
$86 \rightarrow 101$	N102
$102 \rightarrow 5$	N102
$6 \rightarrow 69$	N32

N70 Finger Table

key	node
$81 \rightarrow 81$	N85
$82 \rightarrow 83$	N85
$84 \rightarrow 87$	N85
$88 \rightarrow 95$	N102
$96 \rightarrow 111$	N102
$112 \rightarrow 15$	N113
$16 \rightarrow 79$	N32

N80 Finger Table



✓ Por e tirar mós!
 → Distribuir a info. com este mó!»

Join

1. Iniciar o predecessor e finger tables do novo nó.
 Conhecimento do predecessor é útil na estabilização
2. Actualizar o predecessor e finger tables dos nós existentes. → contacta os finger tables
 Notificar os nós que devem incluir o novo nó nas suas tabelas.
3. Transferir objectos que o necessitem para o novo nó.

$O(\log b \log n)$

\Rightarrow e.g.: Nas testes do prof. demoraram bastante tempo, têm de estabilizar...

