

# SISTEMAS DISTRIBUÍDOS

## Capítulo 3

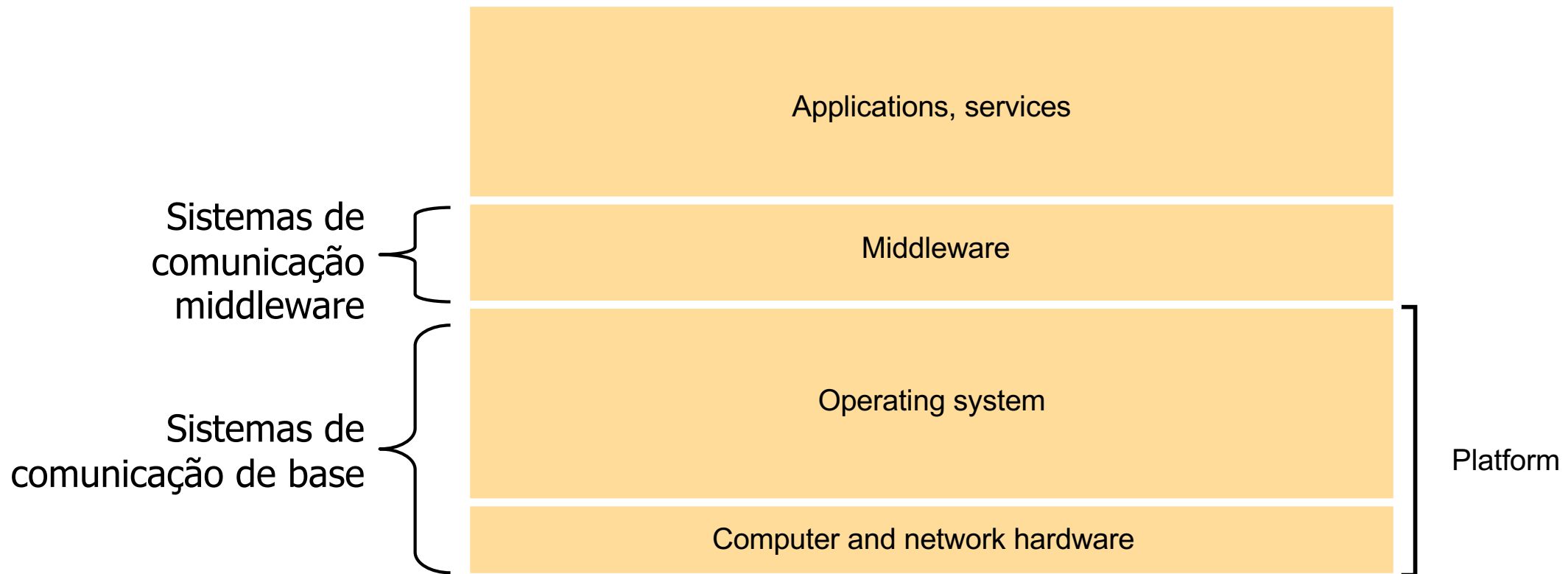
### Comunicação direta

# NOTA PRÉVIA

A estrutura da apresentação é semelhante e utiliza algumas das figuras do livro de base do curso

G. Coulouris, J. Dollimore and T. Kindberg,  
Distributed Systems - Concepts and Design,  
Addison-Wesley, 4th Edition, 2005

# COMUNICAÇÃO NUM SISTEMA DISTRIBUÍDOS



# SISTEMAS DE COMUNICAÇÃO DE BASE

Os sistemas de operação podem suportar a comunicação de dados entre os diferentes computadores envolvidos num sistema distribuído.

Protocolos mais populares:

- TPC/IP
- HTTP

# TCP/IP: UDP

Comunicação por mensagens.

Mensagens podem-se perder, duplicar e chegar fora de ordem.

```
DatagramSocket socket = new DatagramSocket( 9000 ) ;  
  
byte[] buffer = new byte[1500] ;  
DatagramPacket packet = new DatagramPacket( buffer, buffer.length ) ;  
socket.receive( packet ) ;
```

```
byte[] msg = ...  
DatagramSocket socket = new DatagramSocket() ;  
  
DatagramPacket packet = new DatagramPacket( msg, msg.length ) ;  
packet.setAddress( InetAddress.getByName( "servername" ) ) ;  
packet.setPort( 9000 ) ;  
socket.send( packet ) ;
```

# TCP/IP: IP MULTICAST

Comunicação por mensagens com múltiplos recetores.

Cliente envia mensagem para endereço do grupo. Qualquer processo se pode juntar ao grupo para receber mensagens.

Mensagens podem-se perder, duplicar e chegar fora de ordem.

```
MulticastSocket socket = new MulticastSocket( 9000 ) ;  
socket.joinGroup( InetAddress.getByName( "225.10.10.10" ));  
  
byte[] buffer = new byte[1500] ;  
DatagramPacket packet = new DatagramPacket( buffer, buffer.length ) ;  
socket.receive( packet ) ;
```

```
byte[] msg = ...  
MulticastSocket socket = new MulticastSocket() ;  
  
DatagramPacket packet = new DatagramPacket( msg, msg.length) ;  
packet.setAddress( InetAddress.getByName( "225.10.10.10" ) ) ;  
packet.setPort( 9000 ) ;  
socket.send( packet ) ;
```

# TCP/IP: TCP

Dados transmitidos como fluxo contínuo.

Dados chegam de forma fiável a menos que o stream seja quebrado.

```
ServerSocket ss = new ServerSocket( 9000) ;  
while( true ) {  
    Socket cs = ss.accept();  
    ....  
}
```

```
byte[] msg = ...  
Socket cs = new Socket( "servername", 9000) ;  
OutputStream os = cs.getOutputStream() ;  
InputStream is = cs.getInputStream() ;  
  
os.write( msg)  
int b = is.read();
```

# HTTP

Comunicação pedido/resposta sobre TCP, invocando URL.

Dados chegam de forma fiável a menos que o stream seja quebrado.

```
URLConnection con = (URLConnection)
    new URL("http://asc.di.fct.unl.pt/sd/xpto").openConnection();
con.setRequestMethod("POST");
con.setDoOutput(true);
con.setDoInput(true);
OutputStream os = con.getOutputStream();
....
os.flush();
InputStream is = con.getInputStream();
....
```



# HTTP ASSÍNCRONO

Comunicação pedido/resposta, com resposta a ser recebida de forma assíncrona.

- Qual o interesse?

Solução adotada nos browser: JavaScript nativo ou bibliotecas JavaScript (e.g. JQuery)

```
[javascript]
var url = ...
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        result = xmlhttp.response;
        // process result
    }
};
xmlhttp.open("GET", url, true);
xmlhttp.responseType = "json";
xmlhttp.send();
```

## readyState

- 0: request not initialized
- 1: server connection established
- 2: request received
- 3: processing request
- 4: request finished and response is ready

# WEB SOCKETS

Comunicação full-duplex sobre TCP entre clientes e servidores Web.

- Permite notificações dos servidores, streaming.

Suporte generalizado nos browsers.

```
[javascript]
var ws = new WebSocket("ws://asc.di.fct.unl.pt/websocket");

ws.onopen = function() {
    ws.send("Connecting...");
};
ws.onmessage = function (evt) {
    var received_msg = evt.data;
    ...
};
ws.onclose = function() {
    ...
};
```

# HTTP/3 E QUIC

A combinação HTTP/TLS/TCP tem alguns problemas:

- Criação de conexões lenta – handshake TCP + TLS;
- TCP *slow start*;
- Caso haja um erro na propagação dum pacote, esse erro tem impacto na conexão – se uma conexão estiver a ser usada para propagar vários streams (como e.g. quando se vão buscar múltiplas imagens duma página web), todos são afetados.

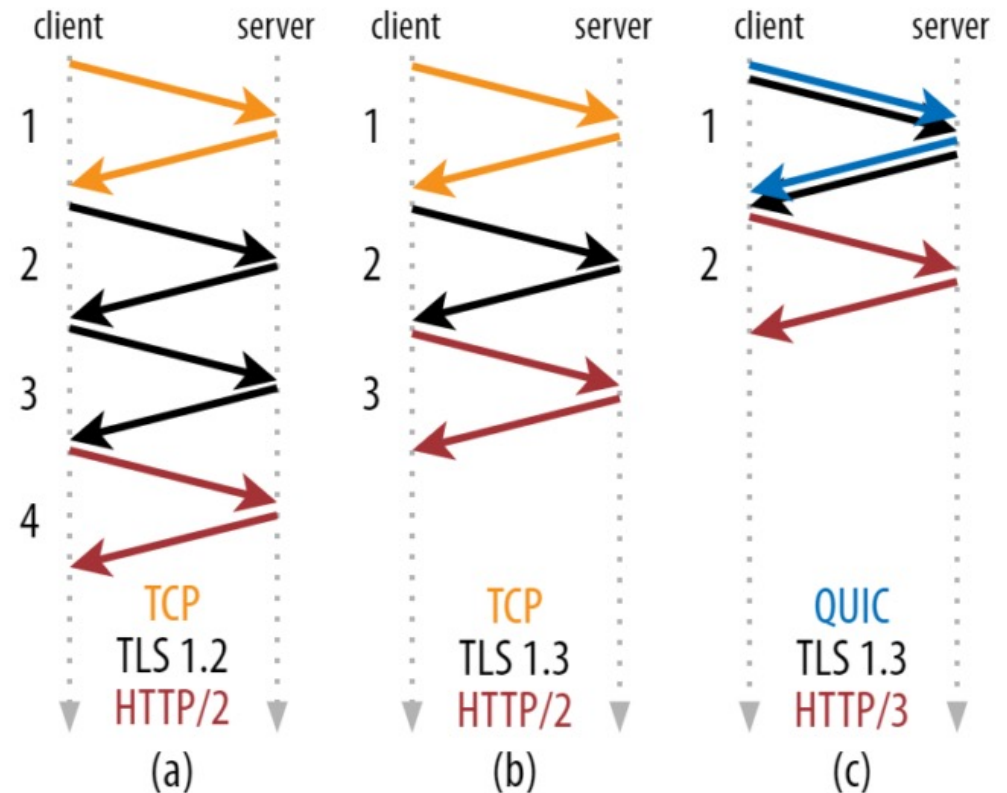


Imagem de: <https://blog.apnic.net/2023/09/25/why-http-3-is-eating-the-world/>

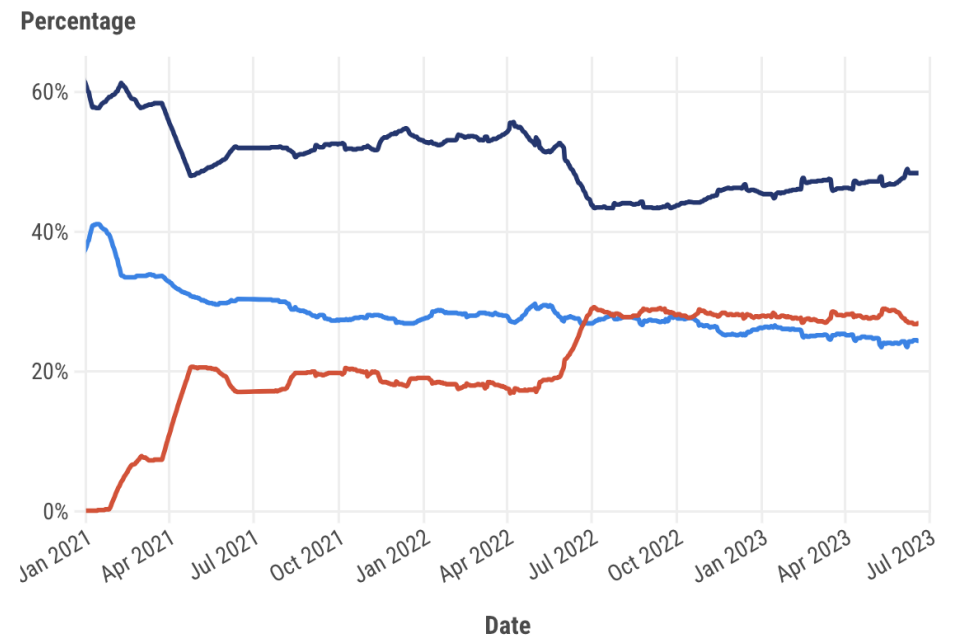
# QUIC

QUIC é um novo protocolo de transporte, contruído usando UDP, com suporte para múltiplos fluxos dentro da mesma conexão; integração com TLS; rápido início de conexão.

## H/3 Adoption Grows Rapidly

HTTP Versions In Use 2021 - 2023

HTTP Version ■ v1.1 ■ v2.0 ■ v3.0



Datasource:  
Mozilla



Imagem de: <https://blog.apnic.net/2023/09/25/why-http-3-is-eating-the-world/>

# HTTP/3

HTTP/3 é a nova versão do HTTP, que integra diretamente com o QUIC.

Suportado pelos browsers mais usados.

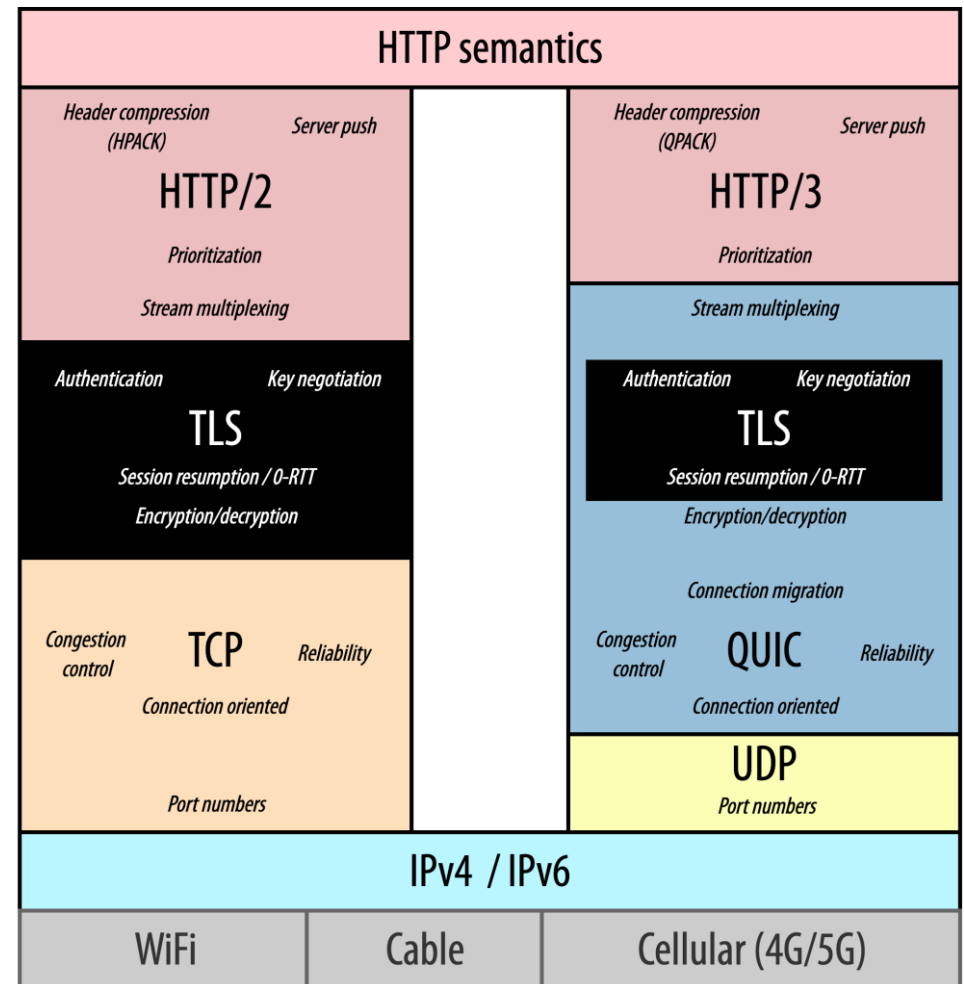


Imagem de: <https://blog.apnic.net/2023/09/25/why-http-3-is-eating-the-world/>

# COMUNICAÇÃO NO NÍVEL MIDDLEWARE

Implementa sistema de comunicação recorrendo às primitivas de comunicação base

Fornece propriedades adicionais, atrasando a entrega das mensagens

- **Definição: Entrega** de uma mensagem num sistema de comunicação representa a ação do sistema disponibilizar a mensagem para ser lida pelas aplicações
- Atrasar a entrega de uma mensagem pode, por exemplo, permitir que a **ordem de entrega** das mensagens seja diferente da **ordem de chegada**.

# FACETAS DA COMUNICAÇÃO

## Forma da interação

- Streams
- Mensagens
  - Ordenação das mensagens

## Número de destinatários

- Ponto-a-ponto
- Multi-ponto (estudado mais tarde)
- Um-de-muitos (anycast)

## Direção de interação

- Uni-direcional
- Bi-direcional

## Tipo de sincronização

- Comunicação síncrona
- Comunicação assíncrona

## Persistência

- Comunicação persistente
- Comunicação volátil

## Fiabilidade (modelo de falhas)

# FORMA DE INTERAÇÃO: STREAMS

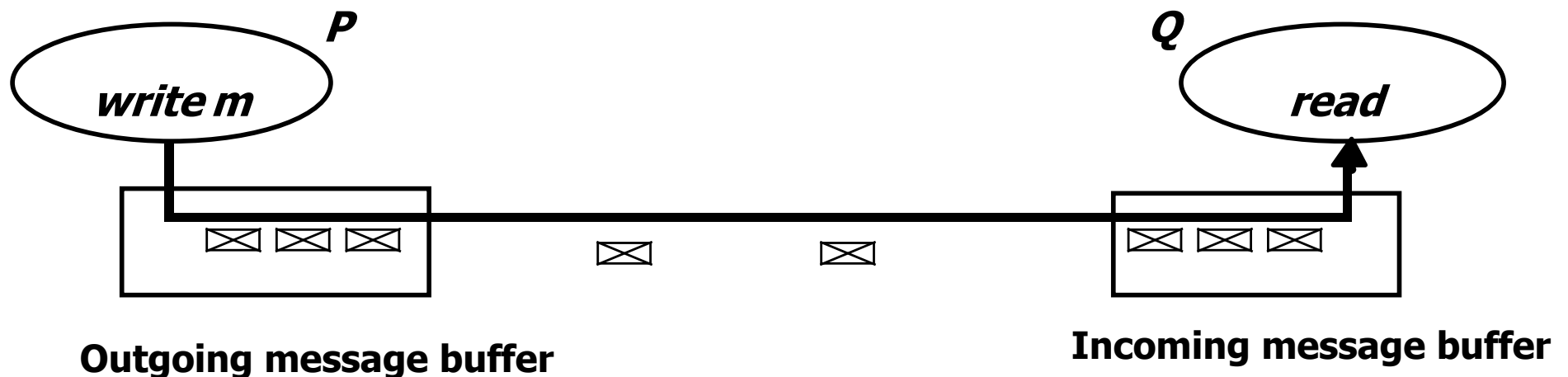
Emissor e recetor estabelecem um fluxo contínuo de dados

Ordem dos dados enviados é mantida;

Fronteira das escritas dos dados não é preservada.

*Exemplos de situações em que é apropriado?*

NOTA: poderíamos implementar sobre UDP... QUIC faz isso.





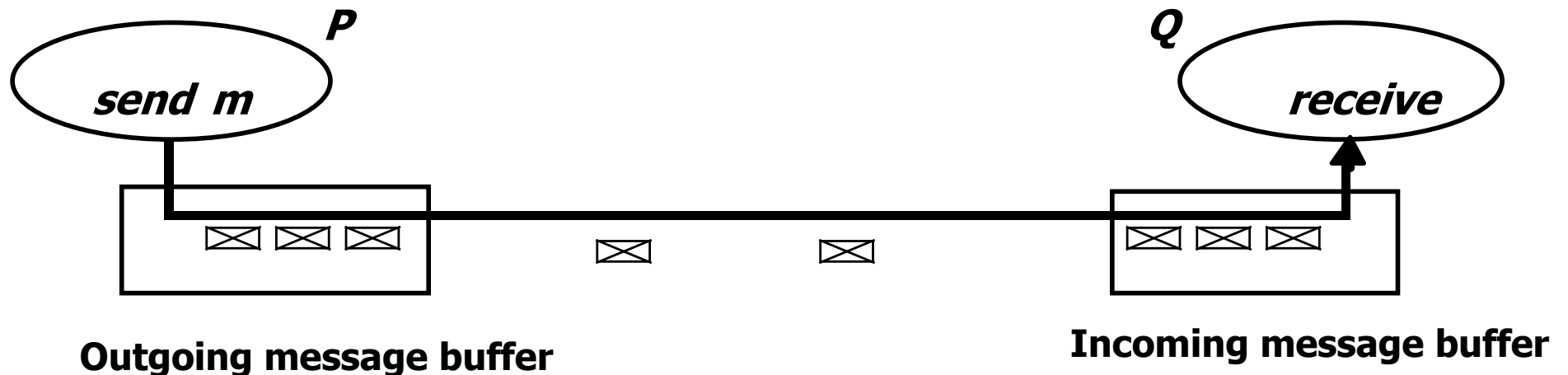
# FORMA DE INTERAÇÃO: MENSAGENS

Emissor e recetor comunicam trocando mensagens

Cada mensagem tem um limite (e dimensão) bem-definida.

Exemplos de situações em que é apropriado?

Como implementar sobre TCP?



# FORMA DE INTERAÇÃO: MENSAGENS

Emissor e recetor comunicam trocando mensagens

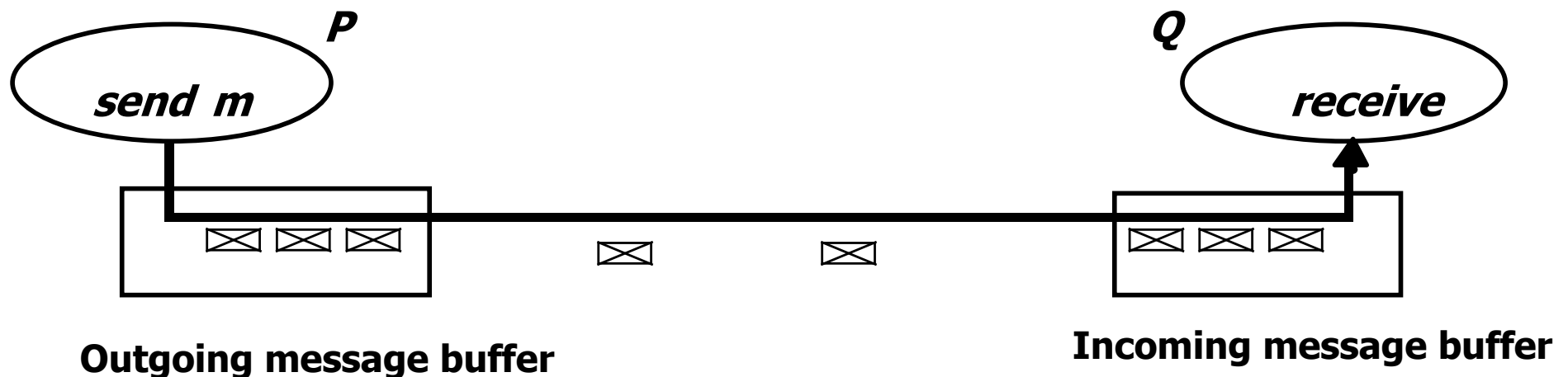
Cada mensagem tem um limite (e dimensão) bem-definida.

Exemplos de situações em que é apropriado?

Como implementar sobre TCP?

<dimensão, dados> ou <dados,delimitador> ou ...

vantagens? desvantagens de cada opção?



# FORMA DE INTERAÇÃO: MENSAGENS

Emissor e recetor comunicam trocando mensagens

Cada mensagem tem um limite (e dimensão) bem-definida.

Exemplos de situações em que é apropriado?

Como implementar sobre TCP?

<dimensão, dados> ou <dados,delimitador> ou ...

No primeiro caso é necessário conhecer a priori a dimensão da mensagem. Para mensagens grandes, geradas dinamicamente poderá obrigar a ter a mensagem em memória antes de a poder enviar. Para mensagens de pequena dimensão é uma boa solução.

No segundo caso, é preciso garantir que o delimitador não ocorre dentro da mensagem, sob pena de o emissor e o receptores ficarem dessincronizados.

Uma terceira abordagem, adequada para mensagens de grande dimensão e geradas dinamicamente, pode-se partir a mensagem numa sequência de blocos de dimensão fixa (pequena), mantendo apenas o último bloco em memória.

# FORMA DE INTERAÇÃO: ORDENAÇÃO DAS MENSAGENS

## Sem garantias de ordem

- Sistema não garante que as mensagens são entregues pela ordem que foram enviadas

## Entrega pela mesma ordem da emissão – FIFO (first in first out)

- Sistema garante que as mensagens dum emissor são entregues pela mesma ordem que foram enviadas. Como implementar em TCP/UDP?
- Haverá outras garantias de ordem?

# NÚMERO DE DESTINATÁRIOS

## Comunicação ponto-a-ponto

- Comunicação entre um emissor e um recetor

## Comunicação multi-ponto

- Comunicação entre um emissor e um conjunto de recetores
- **Broadcast**: envio de 1 emissor para todos os recetores
- **Multicast**: envio de 1 emissor para todos os recetores de um grupo
- **Anycast**: envio de 1 emissor para um recetor de um grupo

# DIRECÇÃO DE INTERACÇÃO

## Comunicação uni-direccional:

- Comunicação apenas num sentido: emissor->recetor

## Comunicação bi-direccional:

- Comunicação nos dois sentidos

# SINCRONIZAÇÃO

## Comunicação assíncrona:

- o emissor só fica bloqueado até o seu pedido de envio ser tomado em consideração
- o recetor fica bloqueado até ser possível receber dados
  - Em geral, o sistema de comunicação do receptor armazena (algumas) mensagens caso não exista nenhum recetor bloqueado no momento da sua recepção. Assim, funciona como um *buffer* entre o emissor e o recetor
  - É possível variante em que o recetor não fica bloqueado e devolve erro ou a receção é efectuada em background

## Comunicação síncrona:

- o emissor fica bloqueado até:
  - o recetor “receber” os dados – comunicação síncrona unidireccional
  - receber a resposta do receptor – comunicação pedido / resposta ou cliente / servidor
- o receptor fica bloqueado até ser possível consumir dados

# PERSISTÊNCIA

**Comunicação volátil:** mensagens apenas são encaminhadas se o recetor existir e estiver a executar, caso contrário são destruídas.

- Exemplo: ???

**Comunicação persistente:** mensagens são guardadas pelo sistema de comunicação até serem consumidas pelos destinatários, que podem não estar a executar. Mensagens são guardadas num receptáculo independente do recetor – mailbox, canal, porta persistente, etc.

- Exemplo: ???



# FIABILIDADE

**Comunicação fiável:** o sistema garante a entrega das mensagens em caso de falha temporária. Como implementar?

**Comunicação não-fiável:** em caso de falha, as mensagens podem-se perder

# PARA SABER MAIS

George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair,

Distributed Systems – Concepts and Design,  
Addison-Wesley, 5th Edition, 2011

- Capítulo 4.1-4.3.