Aula 10

Internet e Programação com Sockets

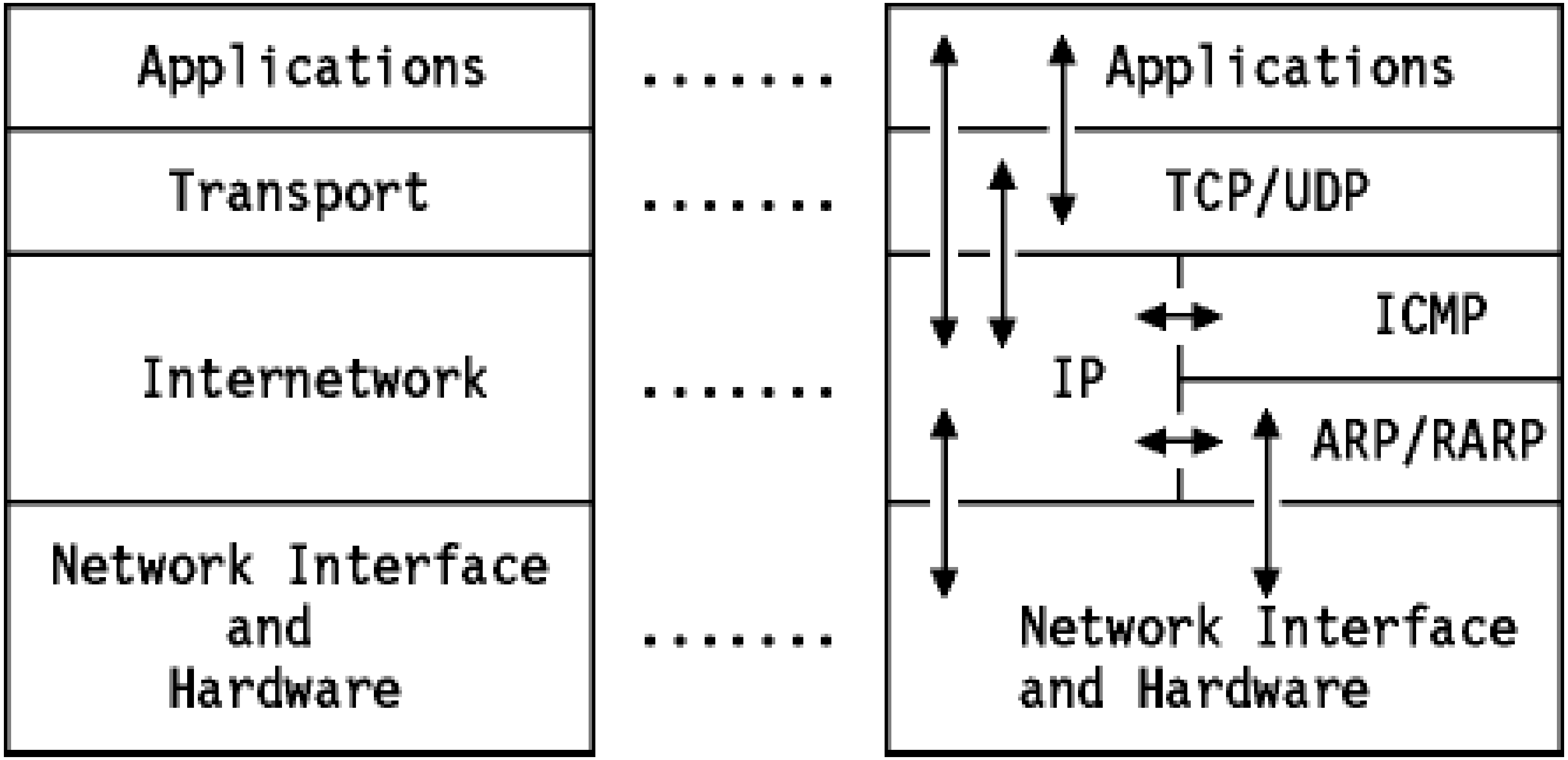
Objetivos

- Internet e seus protocolos.
- Sockets com Python

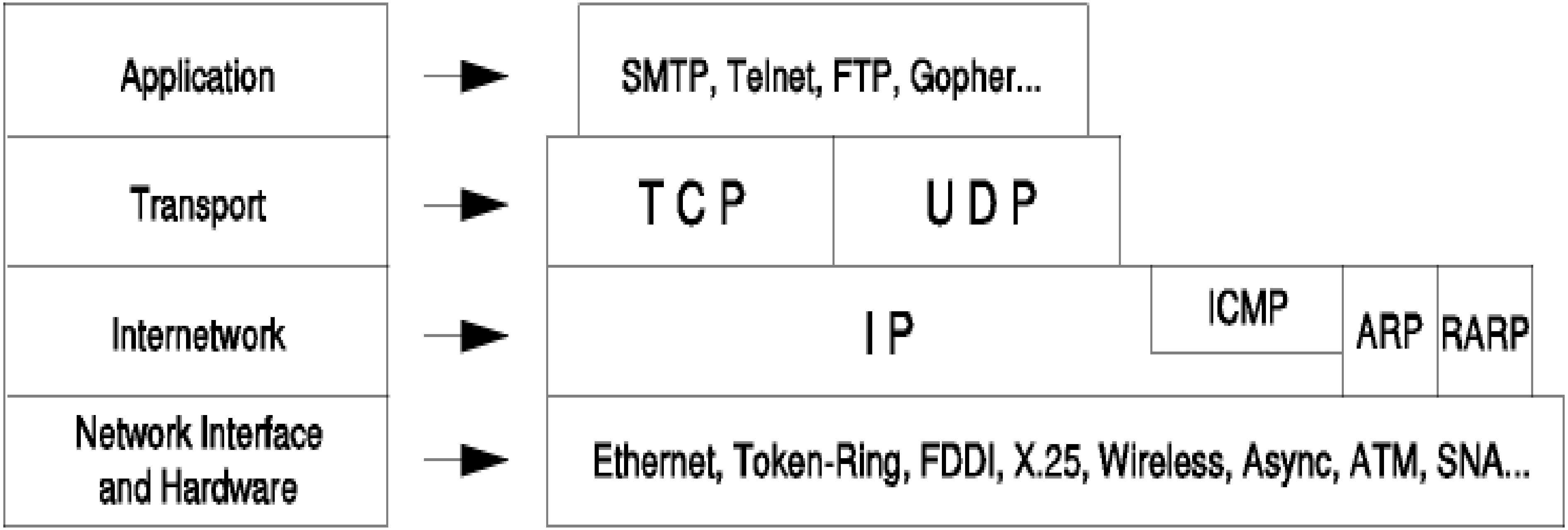
Internet

- Criada a partir da rede ARPANET do DoD.
- A Arpanet tinha como propósito a interligação de centros de pesquisas financiados pelo DARPA e a criação de uma rede robusta através da tecnologia de redes comutadas por pacotes.
- A ARPANET a partir das universidades se ramificou pelo mundo originando a Internet.

Arquitetura



Familia de protocolos do TCP/IP

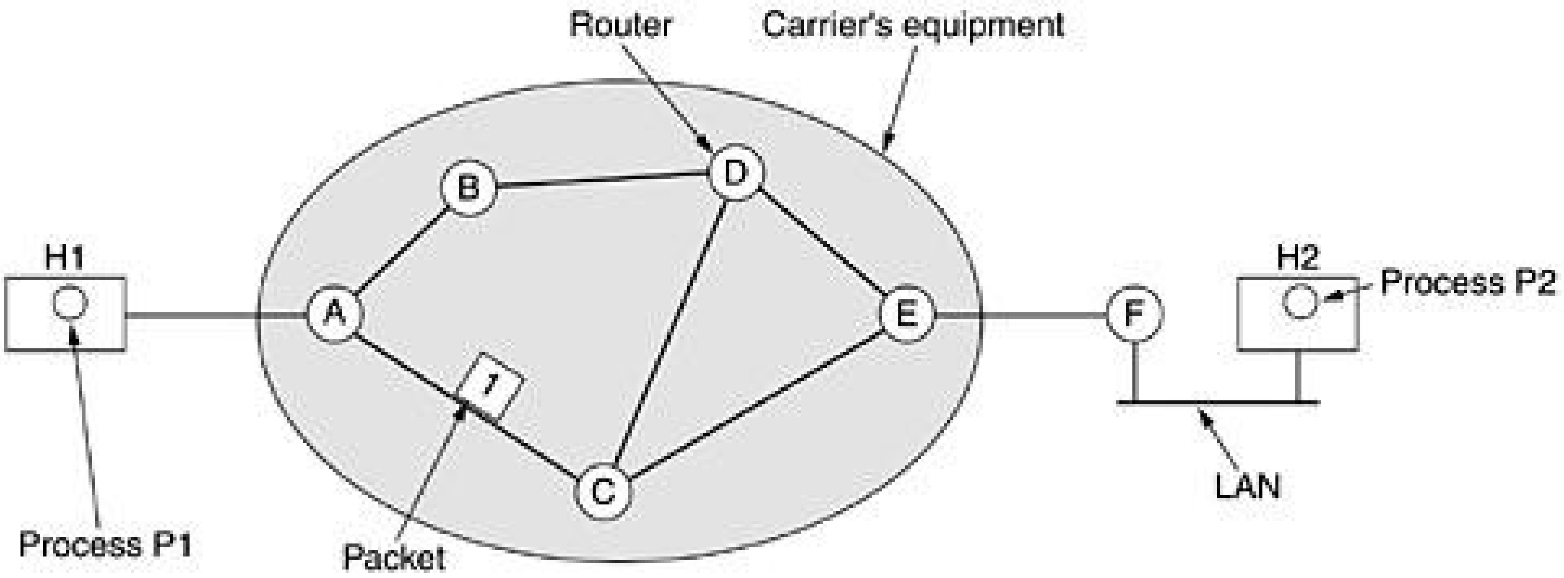


Application → SMTP, Telnet, FTP, Gopher...

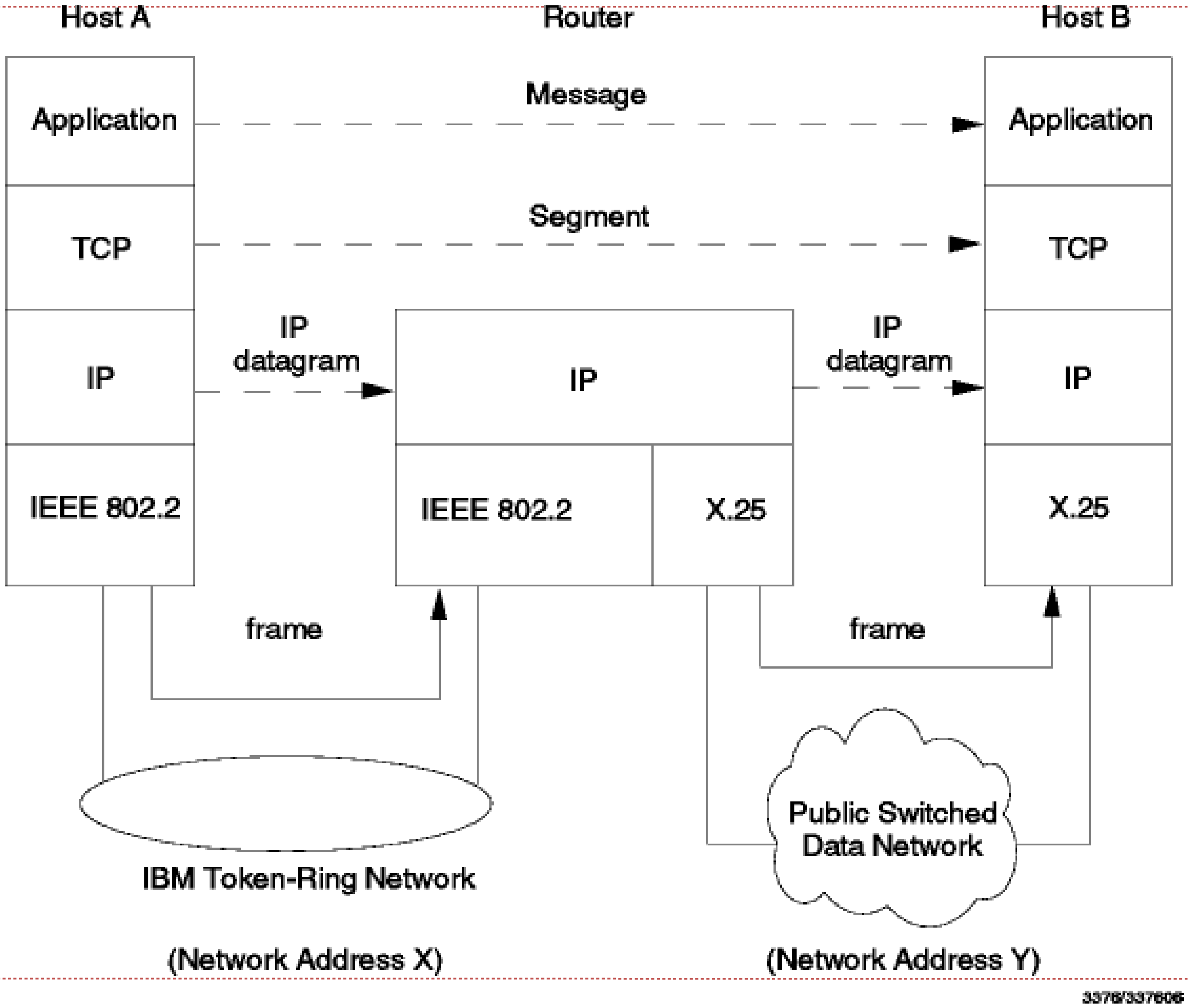
Transport → TCP | UDP

Internetwork → IP | ICMP | ARP | RARP

Network Interface and Hardware → Ethernet, Token-Ring, FDDI, X.25, Wireless, Async, ATM, SNA...

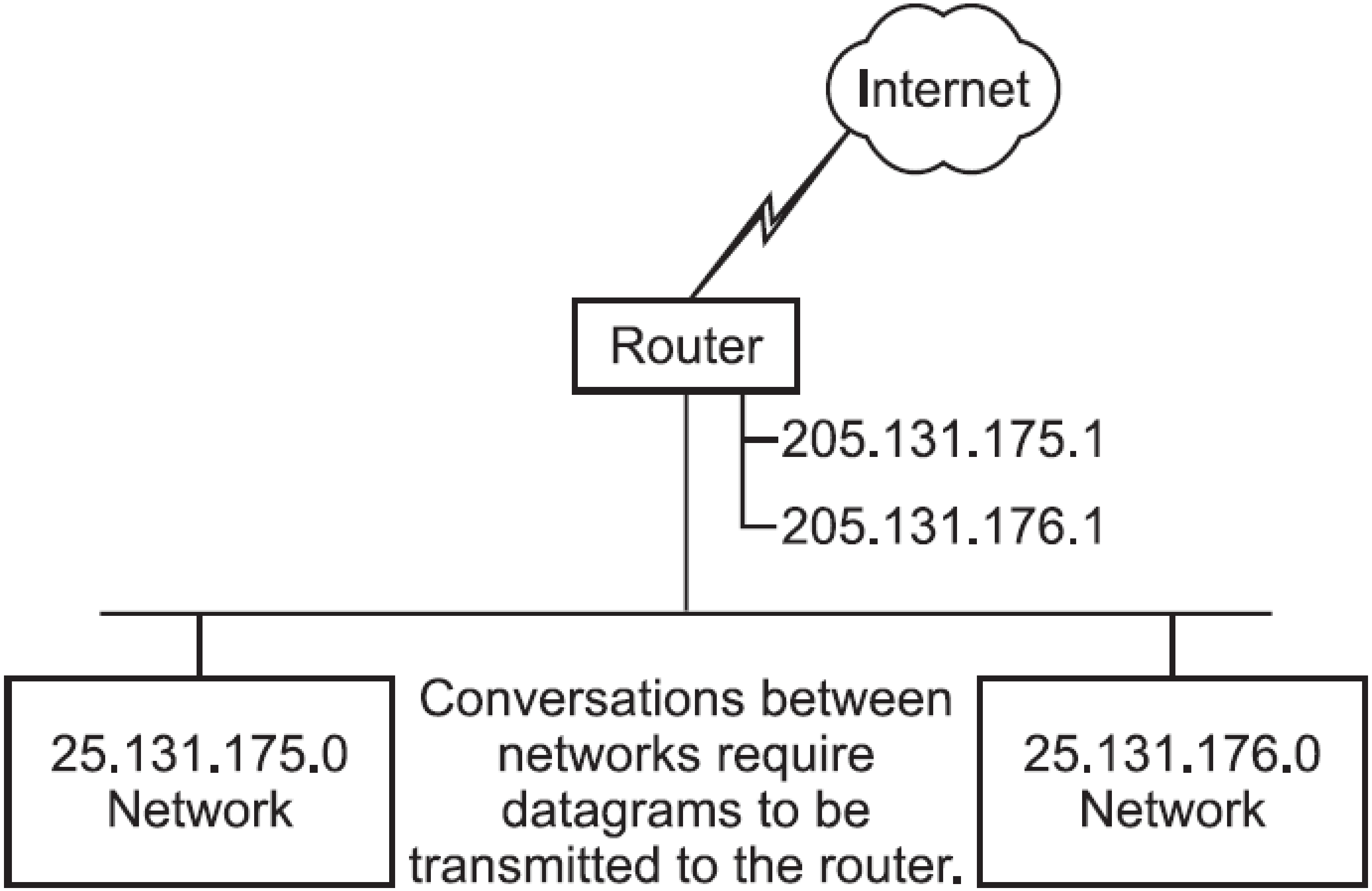

Router
Carrier's equipment
B
D
H1
A
E
F
H2
Process P2
1
C
LAN
Process P1
Packet

Roteamento IP



Datagrama IP

- Datagrama é um pacote de dados com endereço de origem e destino.
- Roteadores analisam o endereço de destino e fazem uma cópia do pacote na rede que possui o melhor caminho para o destino.
- Os pacotes não possuem autenticação.
- Os endereços podem ser falsificados.



Internet

Router

205.131.175.1

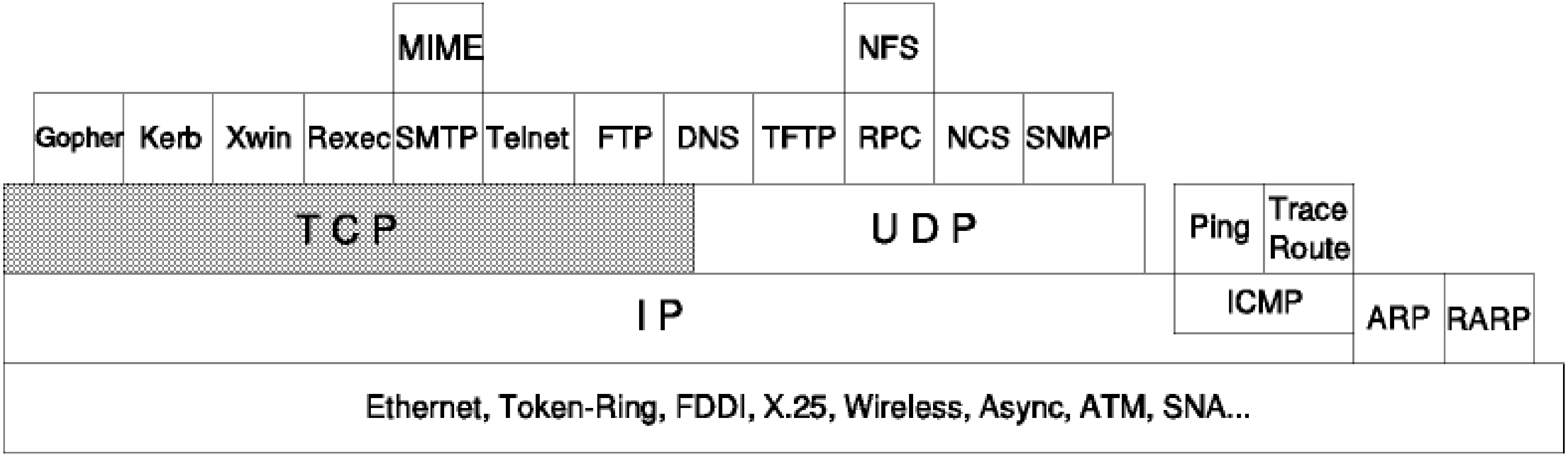
205.131.176.1

25.131.175.0
Network

Conversations between networks require datagrams to be transmitted to the router.

25.131.176.0
Network

TCP



MIME NFS

Gopher Kerb Xwin Rexec SMTP Telnet FTP DNS TFTP RPC NCS SNMP

T C P

U D P

Ping

Trace Route

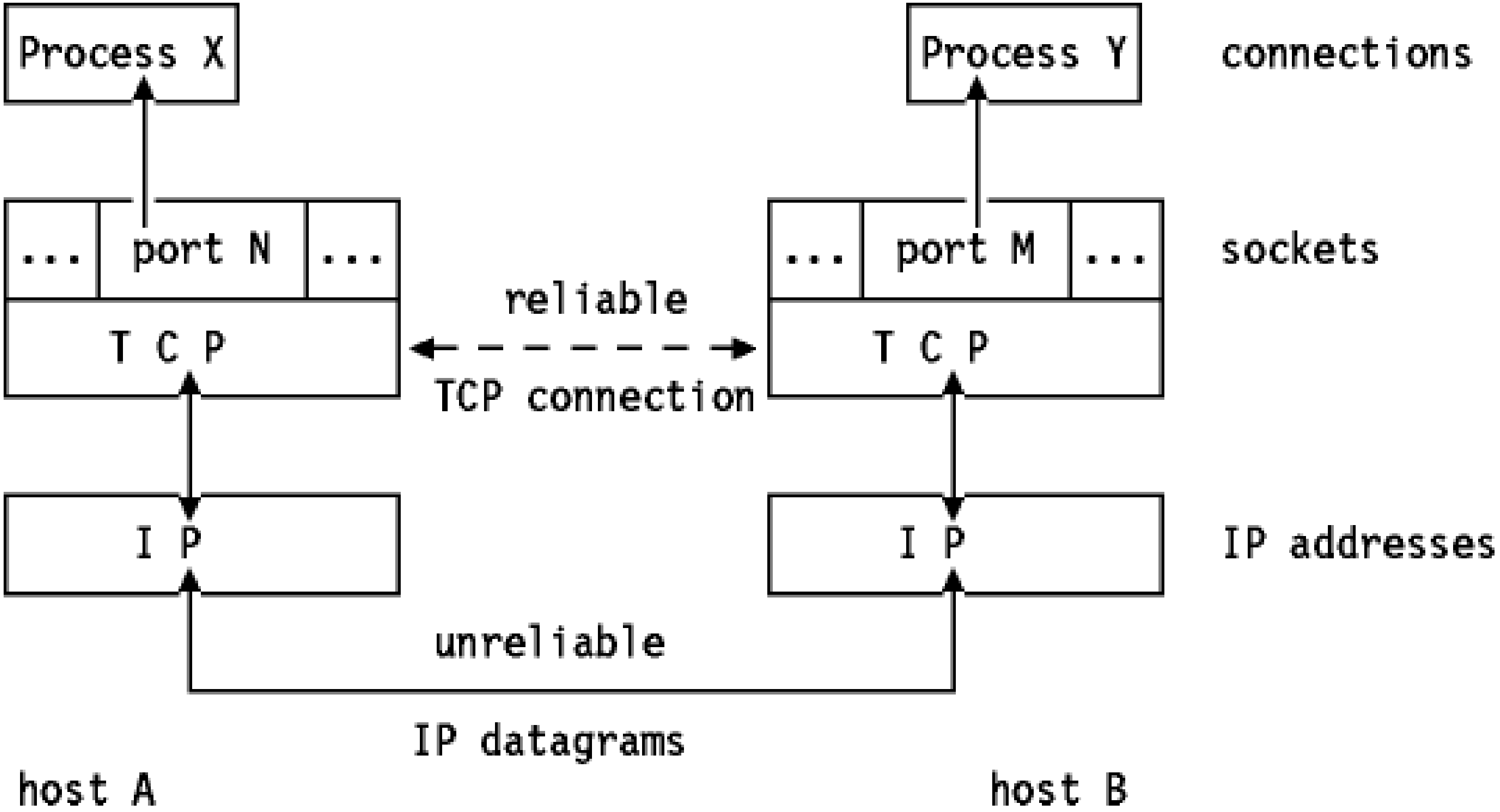
I P

ICMP

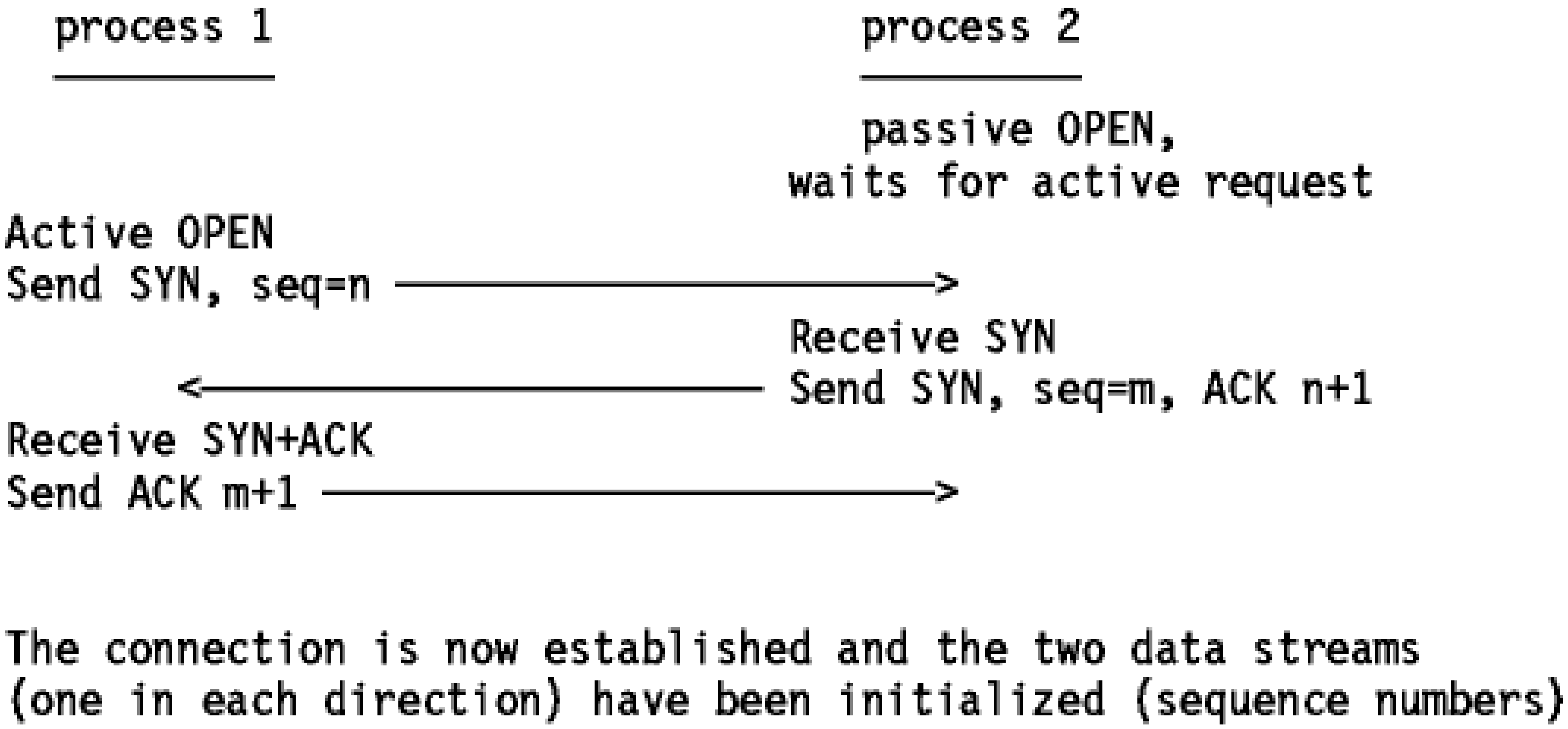
ARP RARP

Ethernet, Token-Ring, FDDI, X.25, Wireless, Async, ATM, SNA...

Conexão TCP


Process X
Process Y
connections
...
port N
...
...
port M
...
sockets
reliable
T C P
T C P
TCP connection
I P
I P
IP addresses
unreliable
IP datagrams
host A
host B

Estabelecimento de conexão

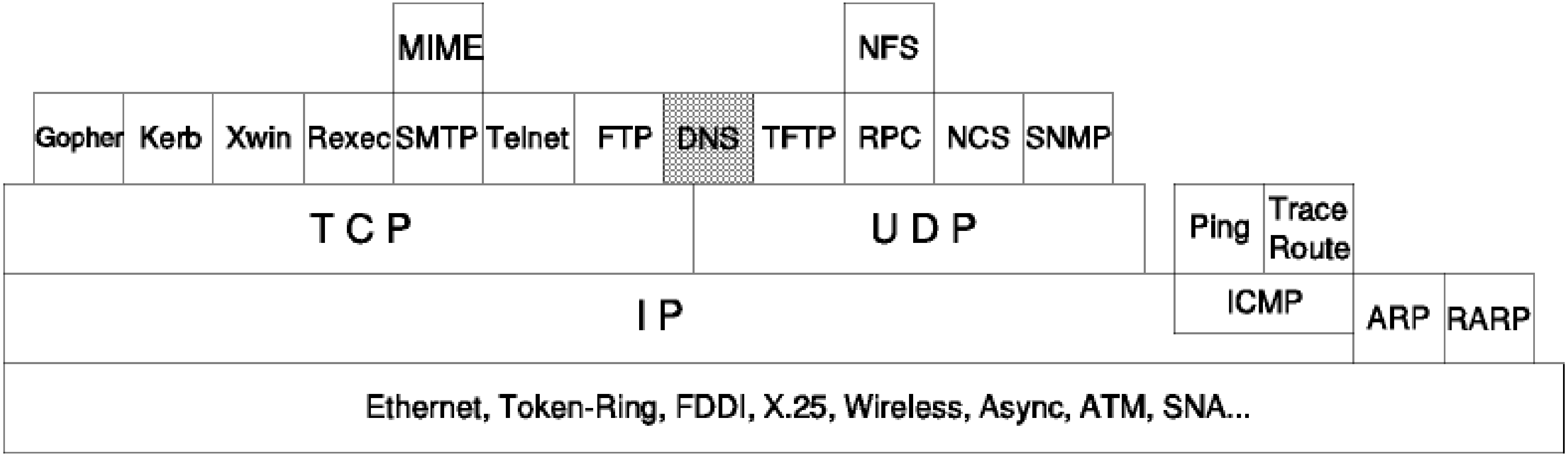


```
   process 1                           process 2
   _________                           _________

                                       passive OPEN,
                                    waits for active request
Active OPEN
Send SYN, seq=n  ----------------------->
                                    Receive SYN
      <-------------------------------- Send SYN, seq=m, ACK n+1
Receive SYN+ACK
Send ACK m+1  -------------------------->
```

The connection is now established and the two data streams
(one in each direction) have been initialized (sequence numbers)

DNS



MIME NFS

Gopher Kerb Xwin Rexec SMTP Telnet FTP DNS TFTP RPC NCS SNMP

T C P

U D P

Ping

Trace Route

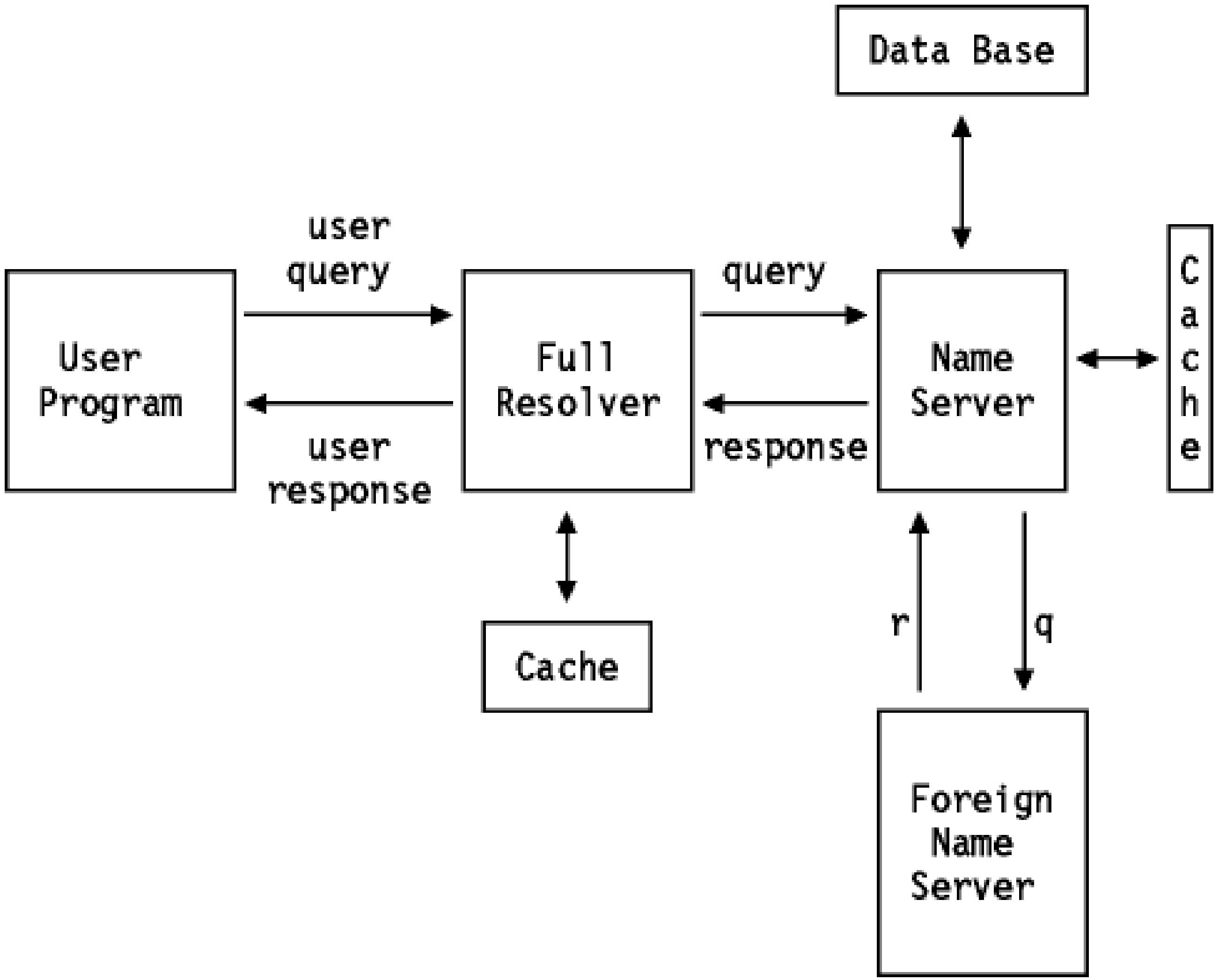
I P

ICMP

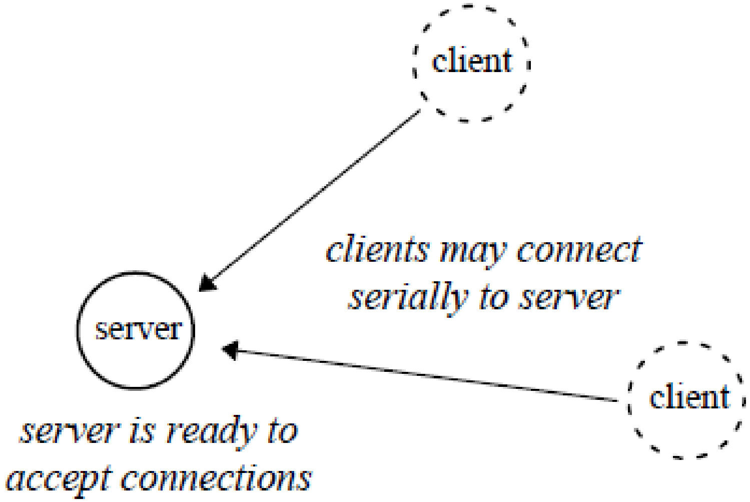
ARP RARP

Ethernet, Token-Ring, FDDI, X.25, Wireless, Async, ATM, SNA...

Processo de resolução de nomes

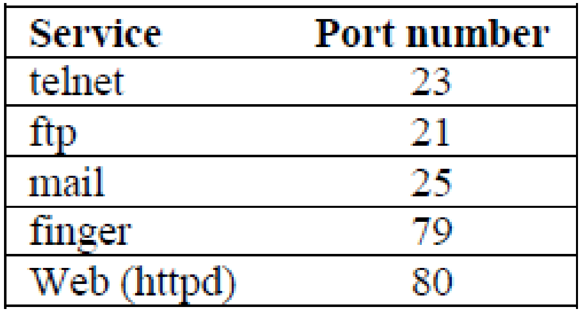


Clientes e Servidores



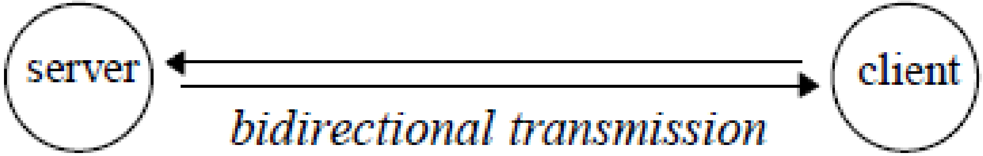
clients may connect serially to server

server is ready to accept connections



Service	Port number
telnet	23
ftp	21
mail	25
finger	79
Web (httpd)	80

Conexão socket em Python



```
s = socket.socket (socket_family, socket_type, protocol = 0)
```

socket_family – AF_UNIX or AF_INET
socket_type – SOCK_STREAM or SOCK_DGRAM.
protocol – This is usually left out, defaulting to 0.

Métodos para servidor

S.No.	Method & Description
1	**s.bind()** This method binds address (hostname, port number pair) to socket.
2	**s.listen()** This method sets up and start TCP listener.
3	**s.accept()** This passively accept TCP client connection, waiting until connection arrives (blocking).

Método para cliente

S.No.	Method & Description
1	**s.connect()** This method actively initiates TCP server connection.

Métodos gerais

S.No.	Method & Description
1	**s.recv()** This method receives TCP message
2	**s.send()** This method transmits TCP message
3	**s.recvfrom()** This method receives UDP message
4	**s.sendto()** This method transmits UDP message
5	**s.close()** This method closes socket

Obter endereco de host

```
import socket
def print_machine_info():
    host_name = socket.gethostname()
    ip_address = 
socket.gethostbyname(host_name)
    print( "Host name: %s" % host_name)
    print ("IP address: %s" % ip_address)
if __name__ == '__main__':
    print_machine_info()
```

Obter endereço de host remoto

```
import socket
def get_remote_machine_info():
    remote_host = 'www.ita.br'
    print ("IP address: %s"
%socket.gethostbyname(remote_host))
if __name__ == '__main__':
    get_remote_machine_info()
```

Um servidor simples

```
#!/usr/bin/env python
from socket import *
from time import ctime
HOST = ''
PORT = 21567
BUFSIZ = 1024

ADDR = (HOST, PORT)
tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock.bind(ADDR)
tcpSerSock.listen(5)

while True:
    print('waiting for connection...')
    tcpCliSock, addr = tcpSerSock.accept()
    print('...connected from:', addr)
    while True:
        data = tcpCliSock.recv(BUFSIZ)
        if not data:
            break
        strdata=data.decode('utf-8')
        print(strdata)
        tcpCliSock.send((ctime()+' '+strdata).encode('utf-8'))
    tcpCliSock.close()
tcpSerSock.close()
```

Um cliente simples

```
#!/usr/bin/env python

from socket import *

HOST = '127.0.0.1' # or 'localhost'
PORT = 21567
BUFSIZ = 1024

tcpCliSock = socket(AF_INET, SOCK_STREAM)
ADDR = (HOST, PORT)
tcpCliSock.connect(ADDR)
host=tcpCliSock.getsockname() #  print client host name
print(host)

while True:
    data = (input('> ')).encode('utf-8')
    if not data:
        break
    tcpCliSock.send(data)
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print(data.decode('utf-8'))
tcpCliSock.close()
```

Um servidor multithread

```
import socket
import threading
from time import ctime

class ThreadedServer(object):
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
        self.sock.bind((self.host, self.port))

    def listen(self):
        self.sock.listen(5)
        while True:
            client, address = self.sock.accept()
            client.settimeout(60)
            threading.Thread(target = self.listenToClient,args =
(client,address)).start()
```

```
  def listenToClient(self, client, address):
        size = 1024
        while True:
            try:
                data = client.recv(size)
                if data:
                    # Set the response to echo back the recieved data
                    strdata=data.decode('utf-8')
                    print(strdata)
                    client.send((ctime()+' '+strdata).encode('utf-8'))
                else:
                    raise error('Client disconnected')
            except:
                client.close()
                print("Exception")
                return False
if __name__ == "__main__":
    while True:
        port_num = input("Port? ")
        try:
            port_num = int(port_num)
            break
        except ValueError:
            pass
```

```
    ThreadedServer('',port_num).listen()
```

Exercicio

- Desenvolver um servidor de chat para multiplos usuários.