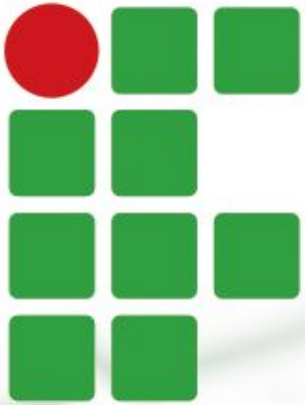


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

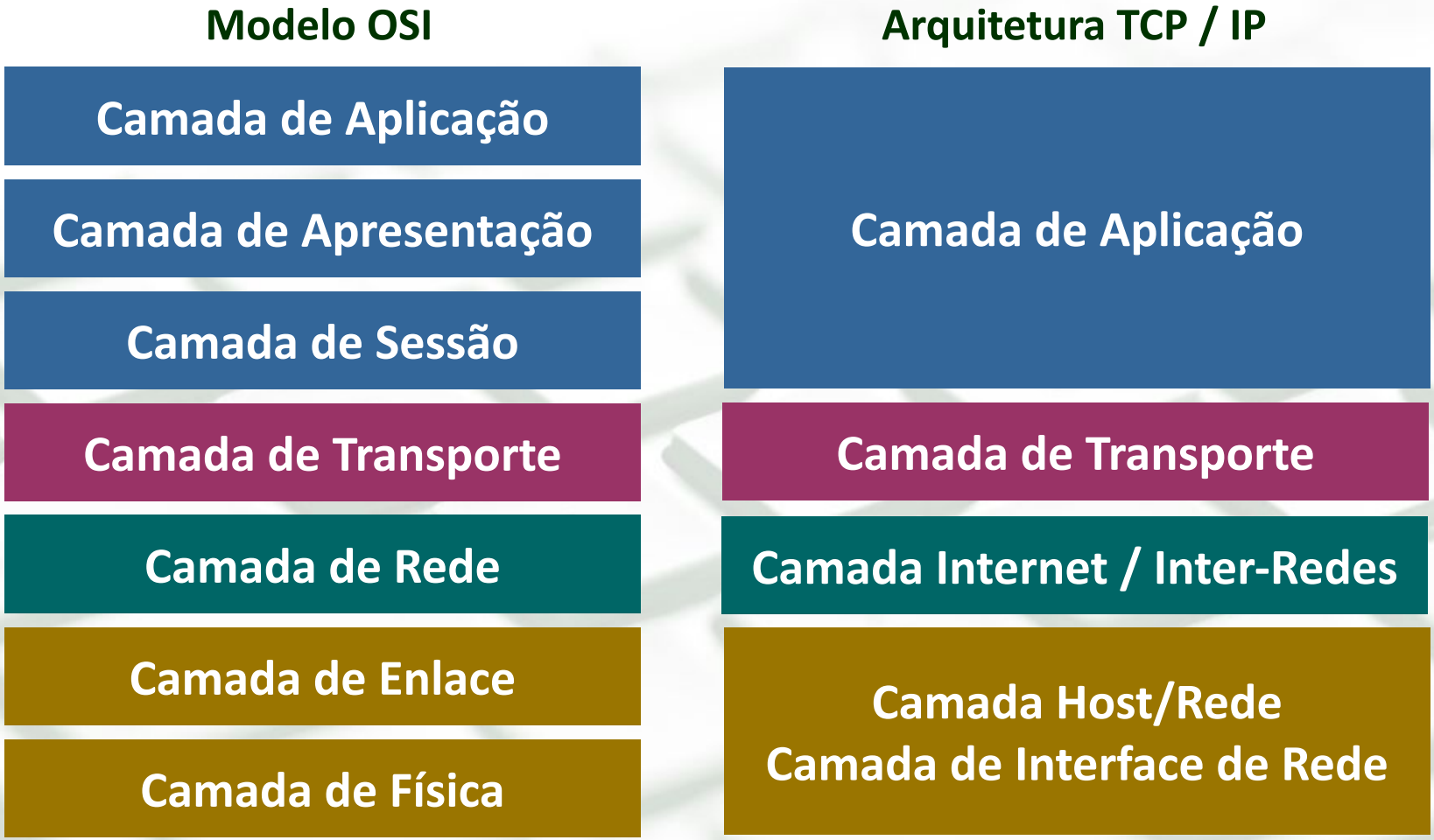
Sistemas Distribuídos

- Sockets -

Prof. MSc. Adriano Antunes Prates

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

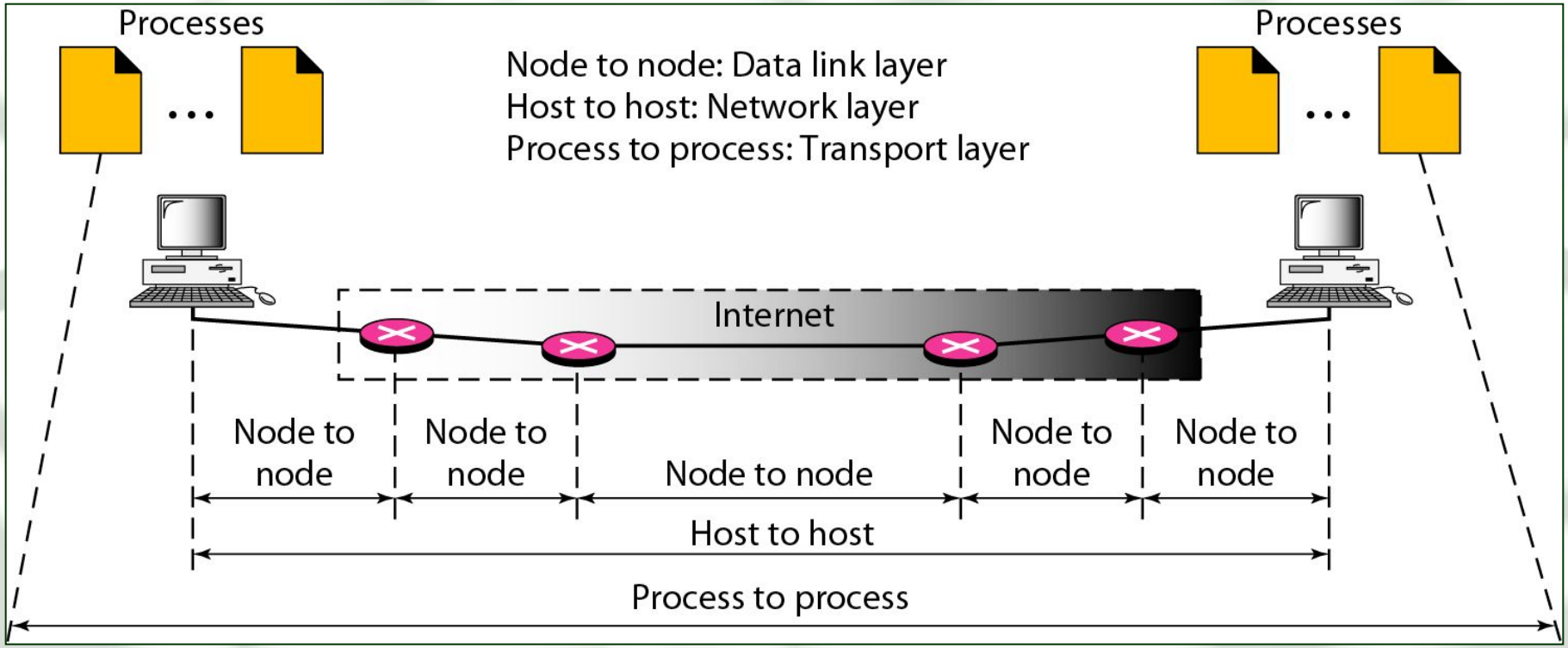
Arquitetura TCP/IP



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP

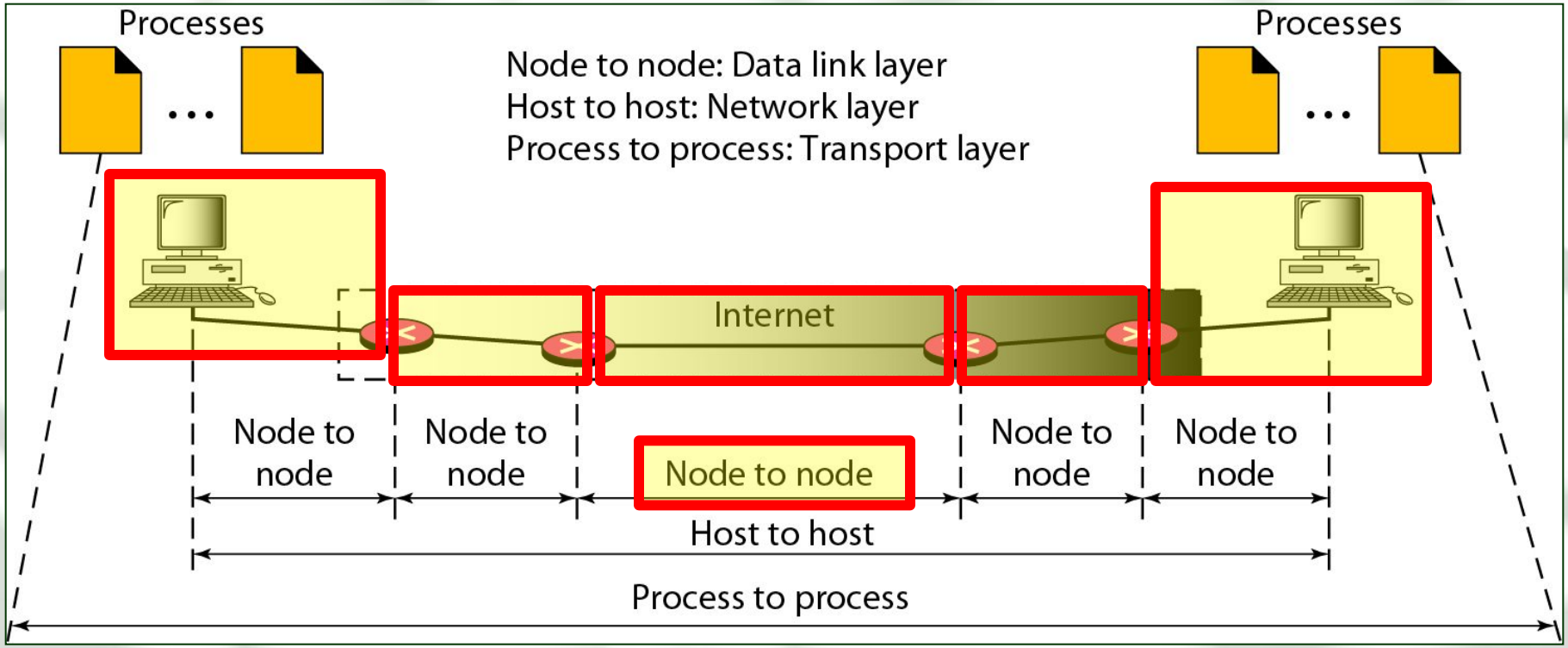
- Modelo de Comunicação em Camadas.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP

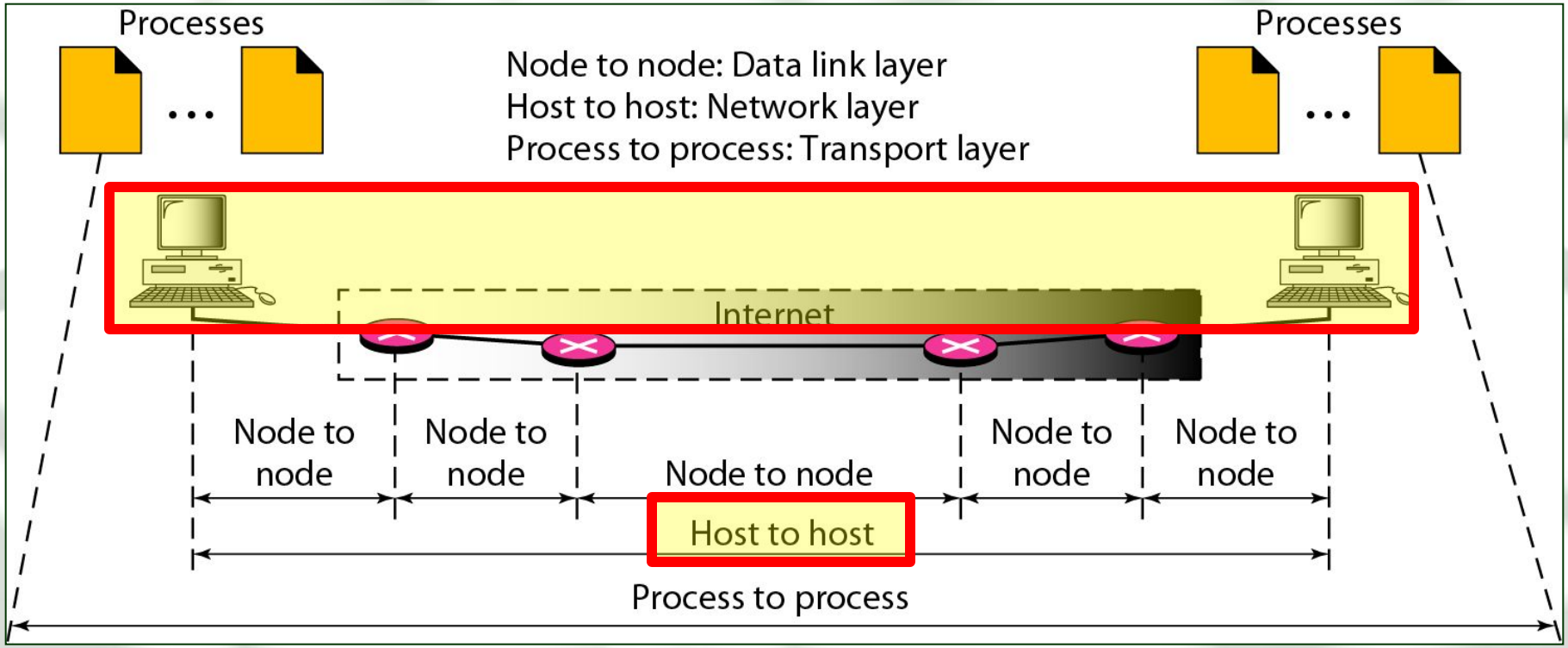
- Comunicação ***node-to-node*** **=> Enlace.**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP

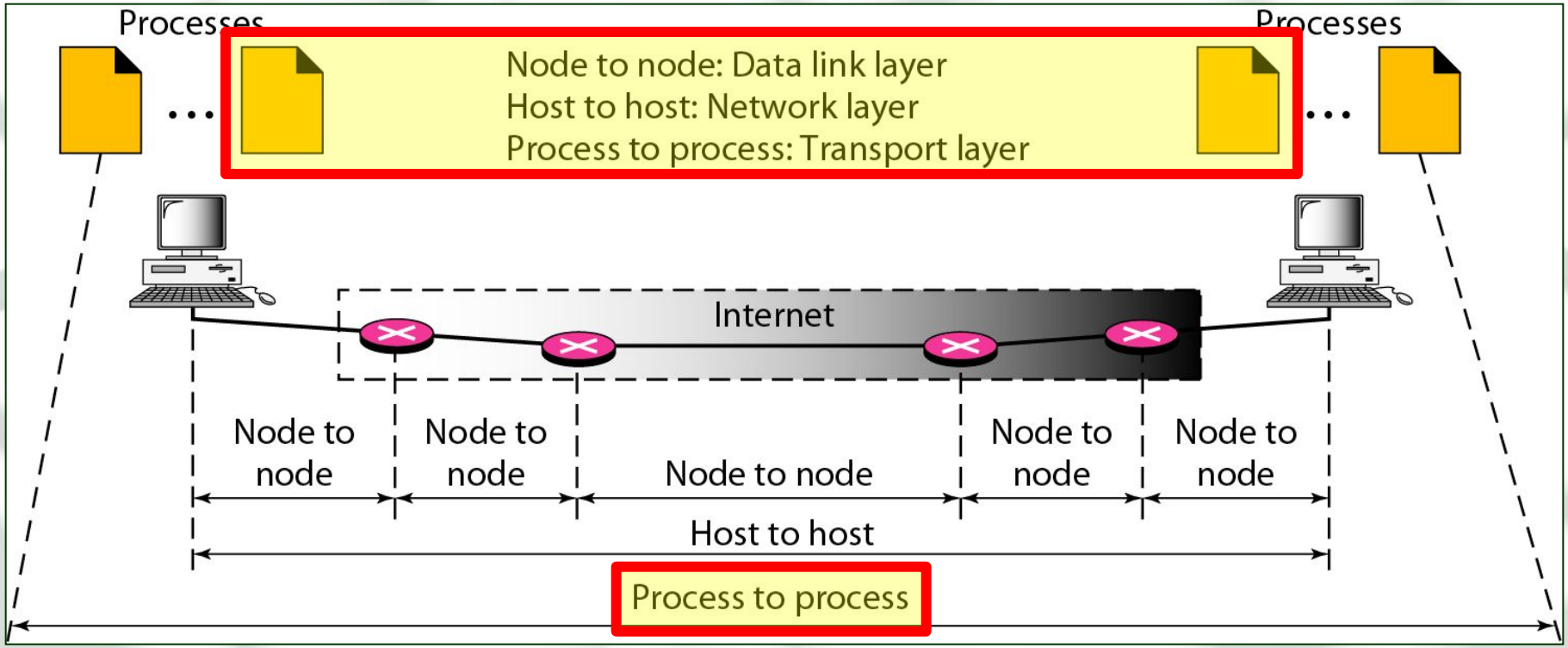
- Comunicação ***host-to-host*** **=> Rede.**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

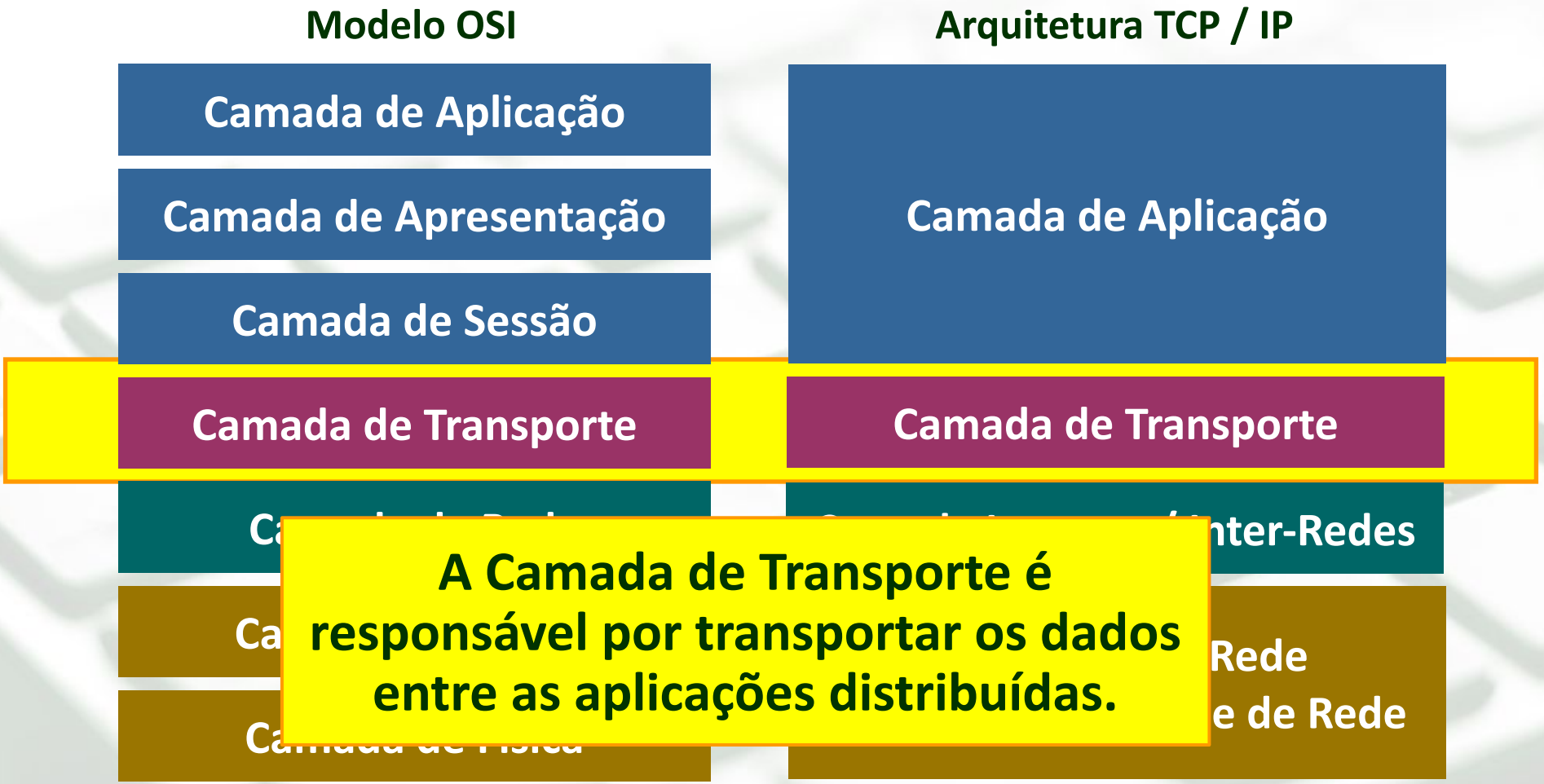
Arquitetura TCP/IP

- Comunicação ***process-to-process*** => **Transporte.**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP

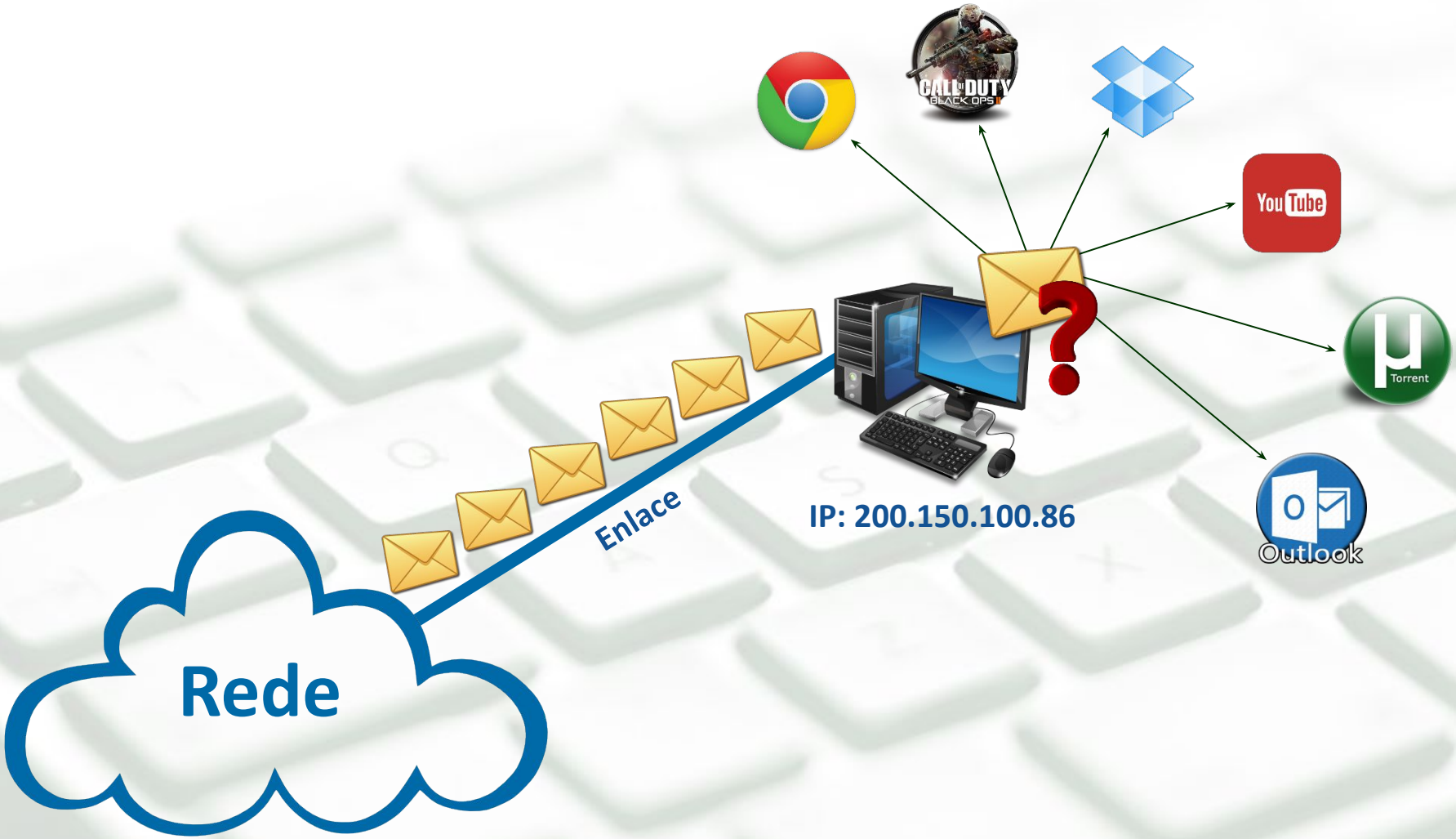


Camada de Transporte

- O sentido real de uma rede de computadores é fazer com que **aplicações (processos) troquem informações entre si** => **IPC (*Inter-Process Communication*).**
- Cada *host* porém, pode estar executando, paralelamente e concorrentemente, inúmeros processos que consomem a mesma rede de comunicação.
- A **Camada de Transporte** é a responsável pelo gerenciamento da comunicação **inter-processos** em execução nos *endpoints*.
- *Vamos entender melhor como isso funciona…*

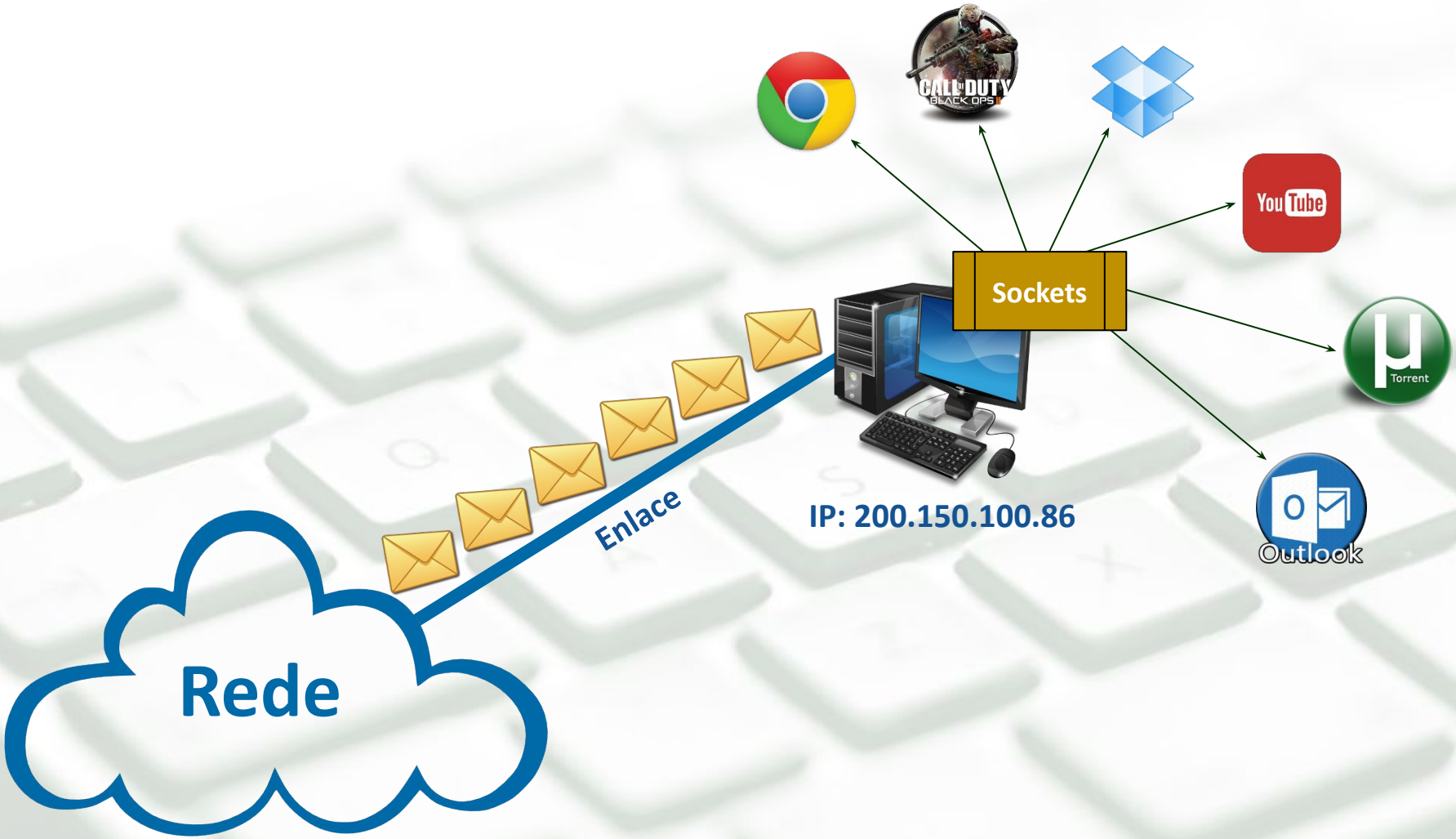
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP



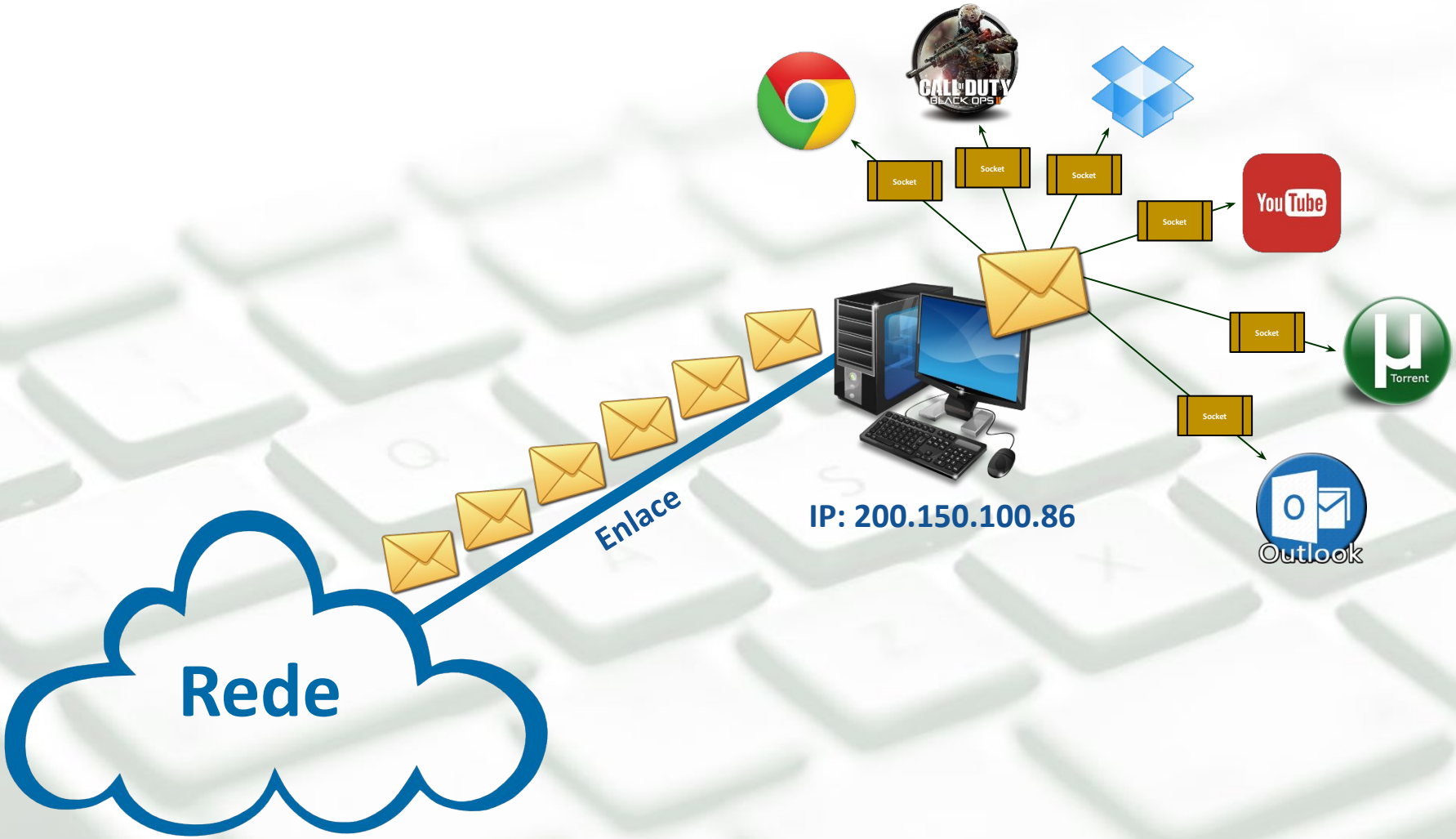
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Arquitetura TCP/IP



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

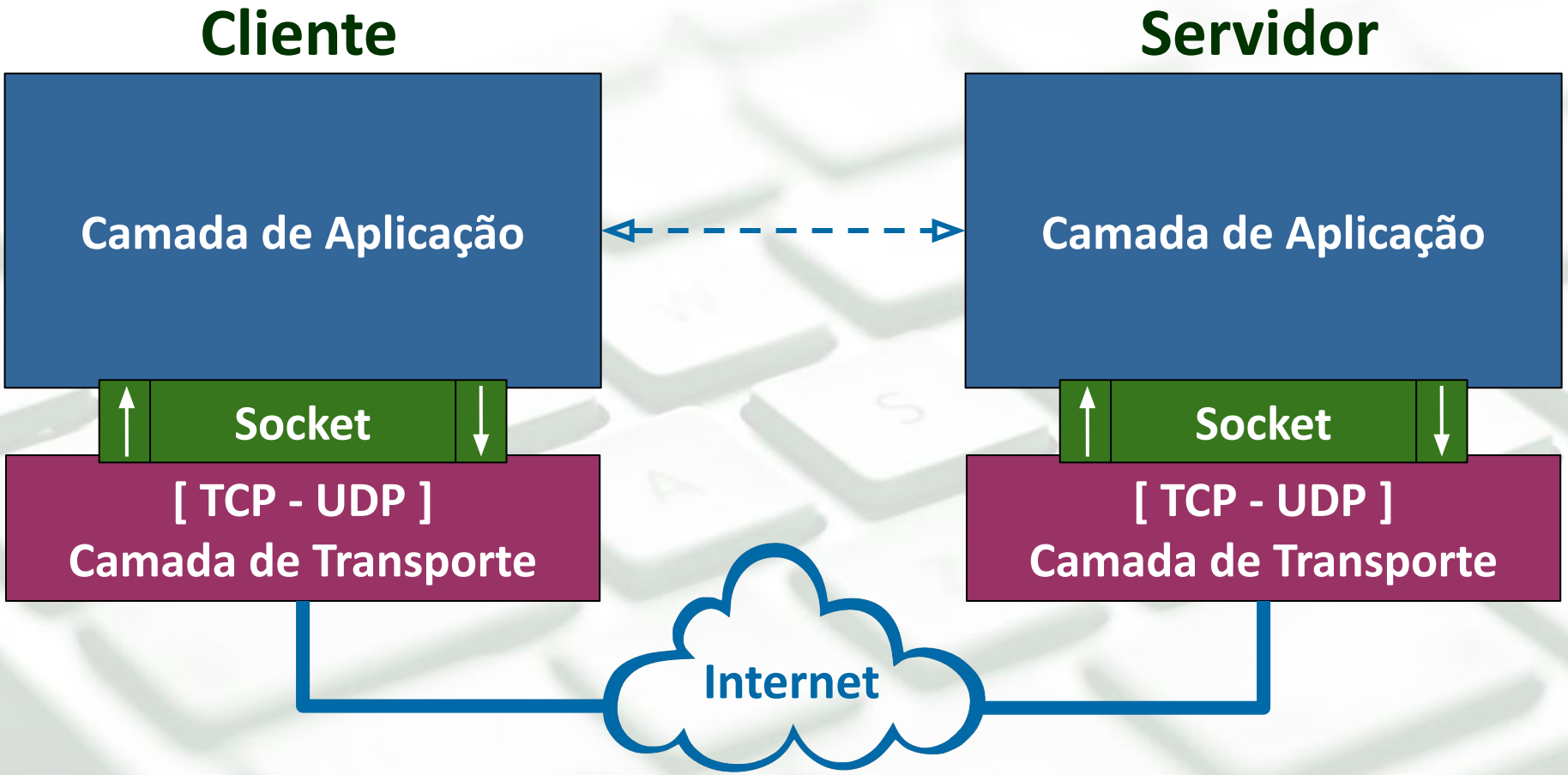
Sockets

- **Socket (soquete)** é uma **API** (*Application Programming Interface*) que abstrai (*simplifica*) o uso da camada de rede para as aplicações (IPC).

- A API ***Socket*** permite que uma aplicação se comunique através da rede **sem que o desenvolvedor se preocupe com os detalhes técnicos de baixo nível da implementação da pilha TCP/IP**.

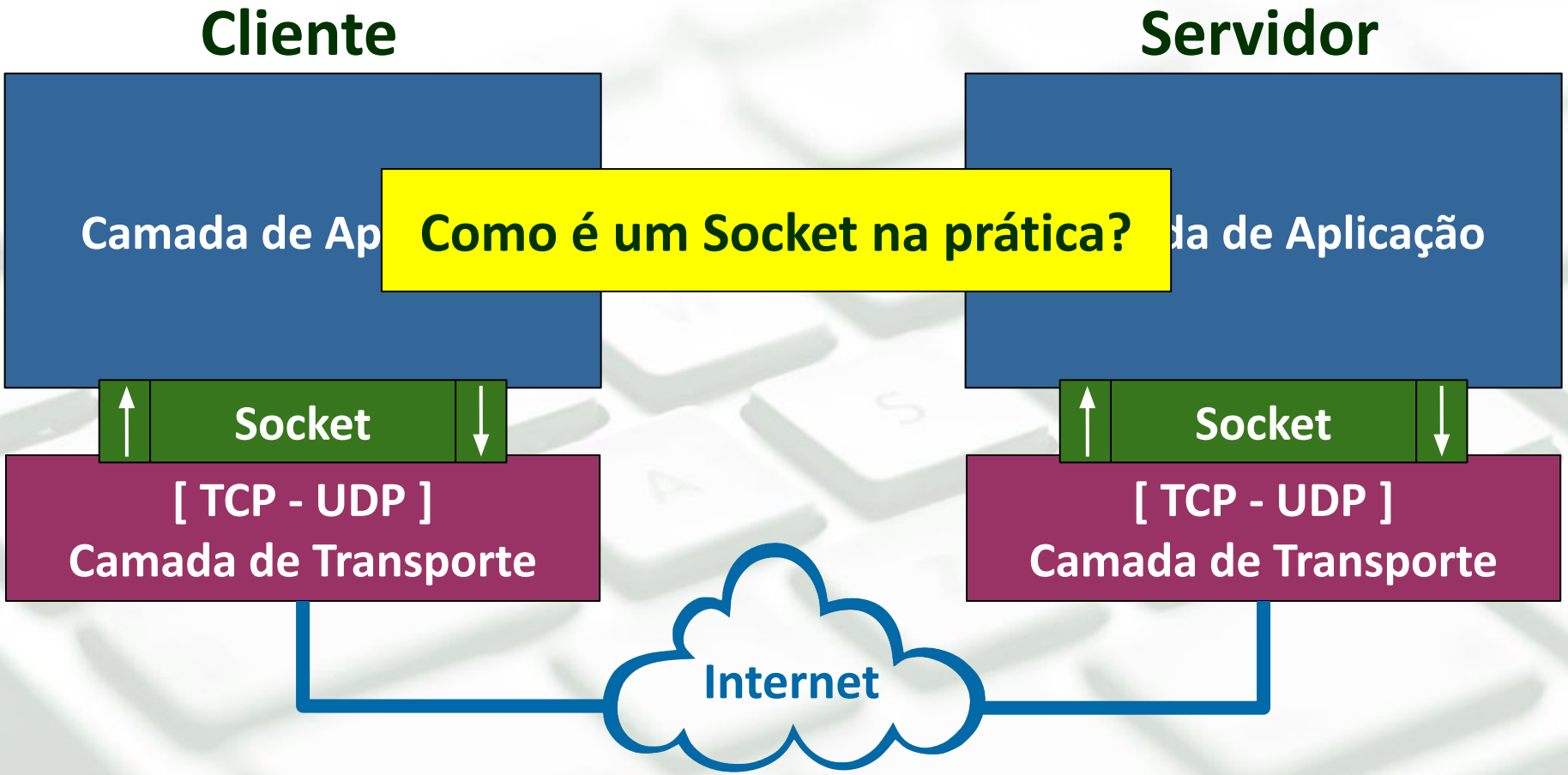
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sockets



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sockets



Sockets

- A representação de um socket se resume à combinação de duas informações:
  - Endereço IP da interface do *endpoint*;
  - Porta de comunicação, TCP ou UDP;

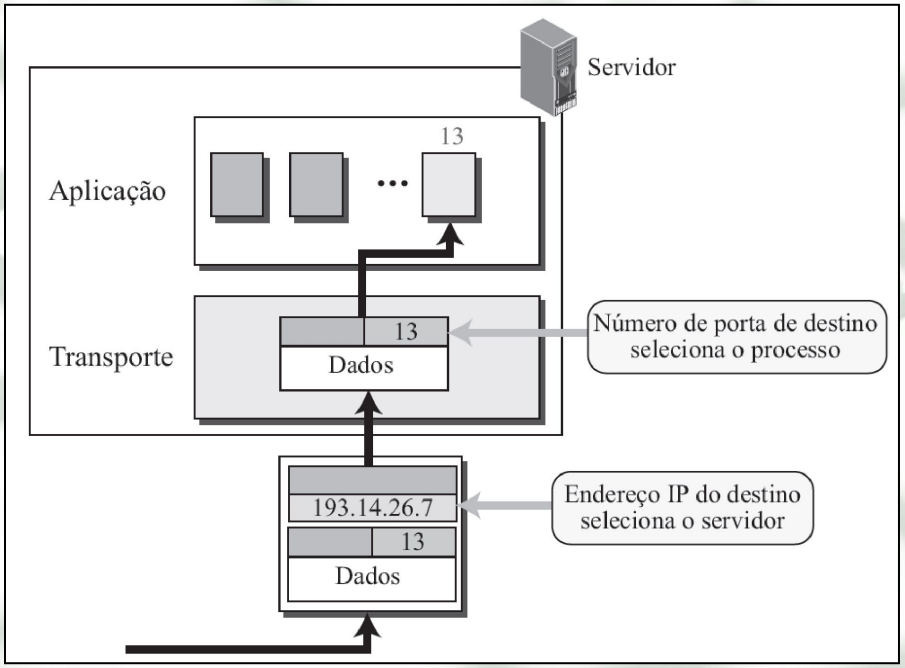
ENDEREÇO_IP : PORTA_DE_COMUNICAÇÃO

Uma porta mapeia um determinado processo que está utilizando a rede de comunicação naquele endpoint.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sockets

- Perceba que para todo nível de comunicação, um esquema de endereçamento é adotado...
- Em nível de Enlace...
  - MAC *Address*
- Em nível de Rede...
  - IP *Address*
- Em nível de **Transporte**...
  - **Port Number.**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sockets

- Retorno do comando: **`netstat -tunp`**

t	**tcp**
u	**udp**
n	**valores numéricos**
p	**processos**
l	**estado listen: servidor**
a	**todas, como cliente e servidor**

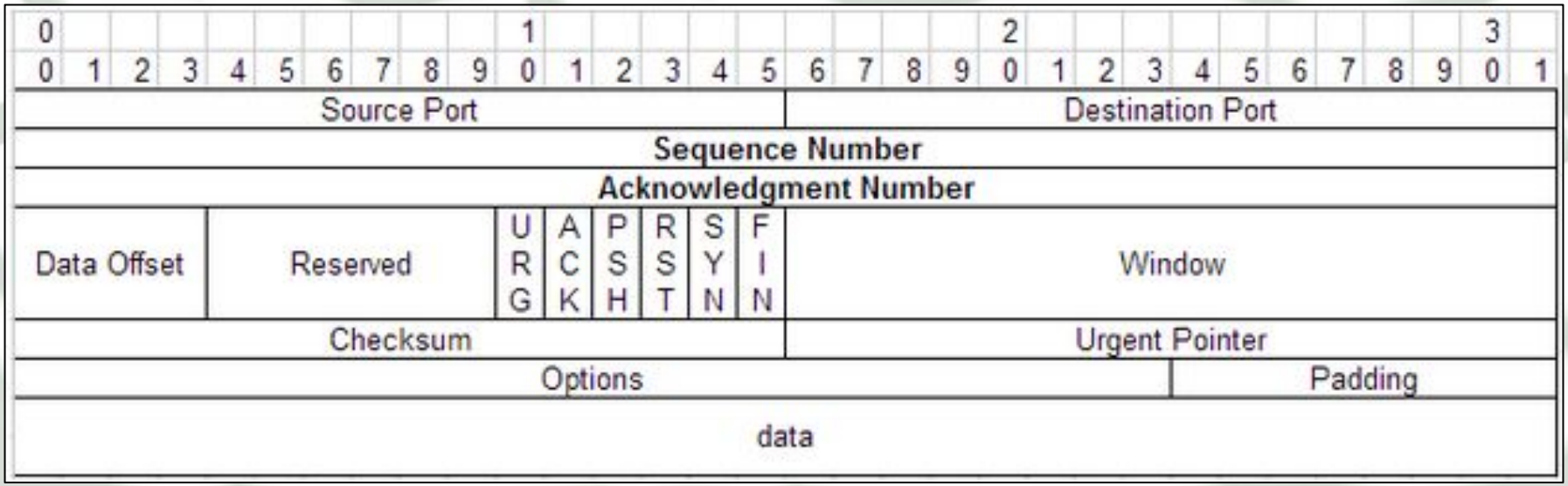
```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
adriano@adriano-All-Series:~$ netstat -tunp
(Nem todos os processos puderam ser identificados, informações sobre processos
 de outrem não serão mostrados, você deve ser root para vê-los todos.)
Conexões Internet Ativas (sem os servidores)
Proto Recv-Q Send-Q Endereço Local           Endereço Remoto         Estado       PID/Program name
tcp        0      0 10.0.0.109:34212        162.125.19.131:443      ESTABELECIDA 4990/dropbox
tcp       32      0 10.0.0.109:53798        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp        0      0 10.0.0.109:44916        157.240.226.17:443      ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:44884        157.240.226.17:443      ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:40936        142.251.128.37:443      ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:46584        31.13.74.52:443         ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:52556        172.67.68.82:443        ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:44840        157.240.226.17:443      ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:37594        172.217.192.188:5228    ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:52206        192.16.58.8:80          ESTABELECIDA 7001/chrome --type=
tcp       32      0 10.0.0.109:53806        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp       32      0 10.0.0.109:53796        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp        0      0 10.0.0.109:40108        31.13.74.18:443         ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:35610        151.101.178.133:443     ESTABELECIDA 7001/chrome --type=
tcp       32      0 10.0.0.109:53800        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp       32      0 10.0.0.109:53804        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp       32      0 10.0.0.109:53802        192.184.81.204:443      ESPERANDO_FECHAR 7001/chrome --type=
tcp        0      0 10.0.0.109:40100        31.13.74.18:443         ESTABELECIDA 7001/chrome --type=
tcp        0      0 10.0.0.109:55378        162.125.19.9:443        ESTABELECIDA 4990/dropbox
udp        0      0 10.0.0.109:47274        142.250.218.14:443      ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:60564        142.251.129.202:443     ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:53029        216.58.222.14:443       ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:33630        142.251.132.227:443     ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:41914        64.233.186.189:443      ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:37960        142.251.129.202:443     ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:46359        142.250.219.3:443       ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:38226        172.217.30.164:443      ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:46614        142.251.129.202:443     ESTABELECIDA 7001/chrome --type=
udp        0      0 10.0.0.109:59018        216.239.32.116:443      ESTABELECIDA 7001/chrome --type=
adriano@adriano-All-Series:~$
adriano@adriano-All-Series:~$
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Portas de Comunicação

- Na arquitetura TCP/IP, uma porta de comunicação é um número inteiro de 16 bits.
  - Valores entre 0 <=> 65.535

Header Protocolo TCP:



Portas de Comunicação

- Na arquitetura TCP/IP, uma porta de comunicação é um número inteiro de 16 bits.
  - Valores entre 0 <=> 65.535

Nome	Faixa	Descrição
Portas Conhecidas	0 – 1023	Atribuídas e controladas pela IANA.
Portas Registradas	1024 – 49151	Registro junto à IANA. Link para visualização
Portas Dinâmicas	49152 – 65535	Portas para uso geral.

Portas de Comunicação

- Na arquitetura TCP/IP
  é um número inteiro
  - Valores entre 0 <=> 65

Nome	Faixa
Portas Conhecidas	0 – 1023
Portas Registradas	1024 – 4
Portas Dinâmicas	49152 –

Algumas Portas Conhecidas e Registradas

20 & 21: FTP
22: Secure Shell (SSH)
23: Telnet
25: SMTP
53: Domain Name System (DNS)
67 & 68: DCHP
80: HTTP Padrão
110: Post Office Protocol (POP3)
143: IMAP
161: SNMP
443: HTTP Secure (HTTPS)
3306: MySQL
5004: Real Time Protocol (RTP)
5938: TeamViewer

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Modelo Cliente x Servidor

Cliente
IP: 200.20.30.85

Internet

WEB Server
IP: 210.120.15.10
Porta Conhecida: 80

O IP e porta (socket) do processo servidor deve ser conhecido.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Modelo Cliente x Servidor

Cliente
IP: 200.20.30.85

Porta Origem = <Porta aleatória (dinâmica)>

Porta Destino = 80 (HTTP)

WEB Server
IP: 210.120.15.10
Porta Conhecida: 80

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Modelo Cliente x Servidor

Socket Cliente

Google Chrome
200.20.30.85:58569

Cliente
IP: 200.20.30.85

Socket Origem = 200.20.30.85:58569

Socket Destino = 210.120.15.10:80

Internet

Socket Servidor

Apache Server
210.120.15.10:80

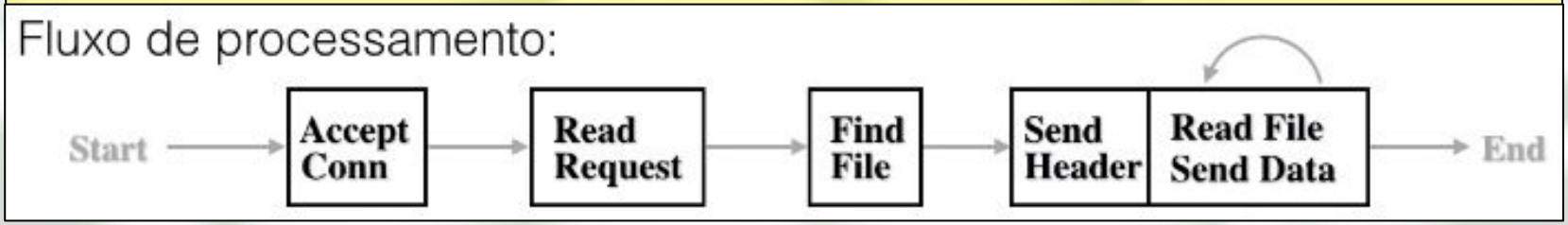
WEB Server
IP: 210.120.15.10
Porta Conhecida: 80

A combinação IP:PORTA identifica de forma exclusiva um processo em execução na Internet.

Implementação do Servidor

- Servidor Mono-Client
  - Socket único para tratar requisições de clientes.
  - Bloqueante e limitante.
  - Um novo cliente só pode ser atendido após o anterior ter finalizado seu atendimento (apenas 1 socket disponível).

Exemplo de Funcionamento de um Servidor WEB

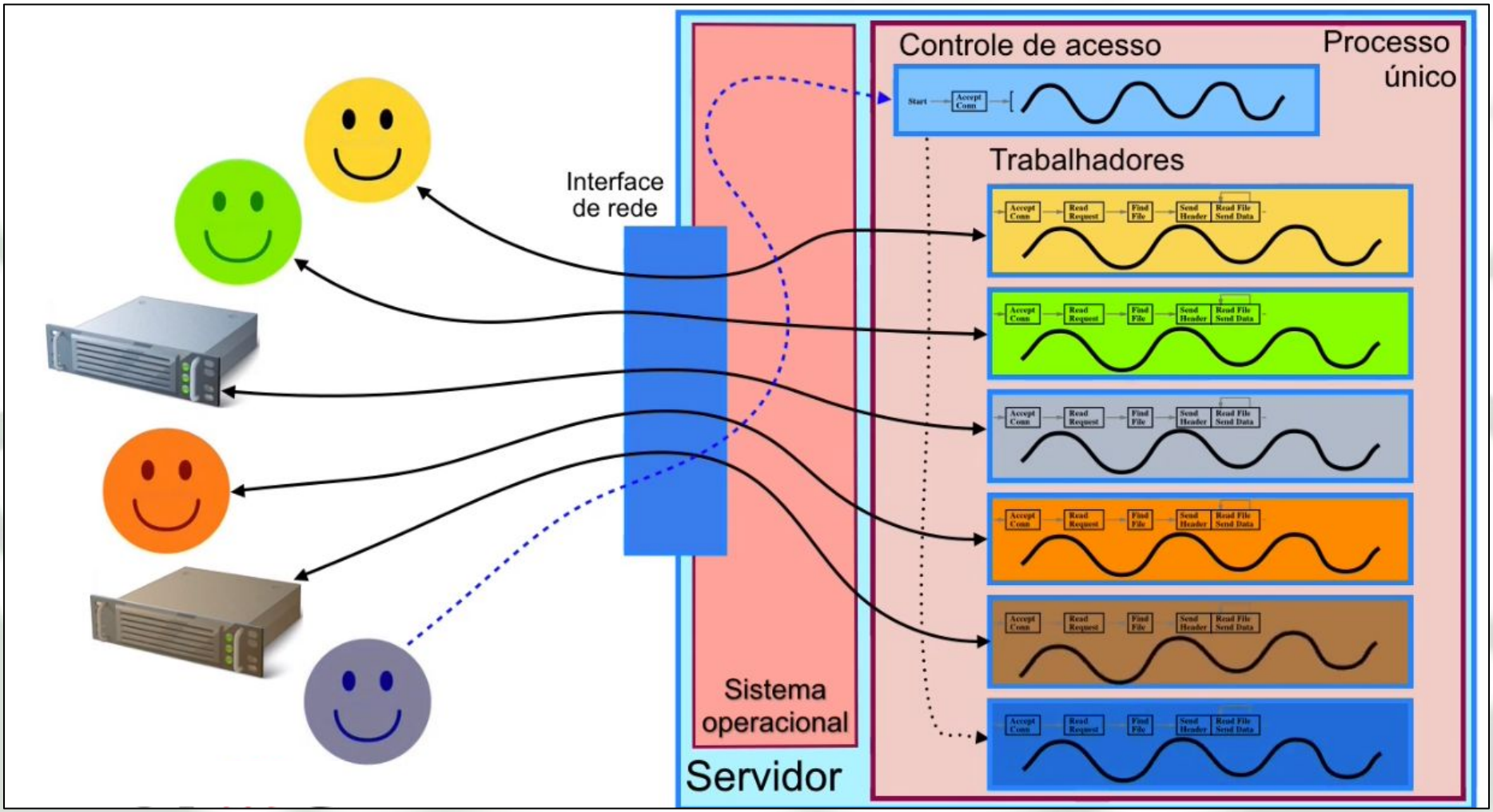


Implementação do Servidor

- Servidor Multi-Client
  - Sockets criados sob-demanda e dedicados para atendimento de cada cliente conectado.
  - A distribuição das tarefas de atendimento (requisições) pode ser baseado em múltiplos processos ou *threads*.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Servidor Multithread



Berkeley Sockets

- Em termos de desenvolvimento, a API Socket criada pela ***Universidade de Berkeley*** (1983) tornou-se padrão para a grande maioria das implementações em diversas linguagens de programação, inclusive Python.
- A API Berkeley consiste em um conjunto definido de funções primitivas para gerenciamento dos *sockets,* que veremos em detalhes a seguir…

Berkeley Sockets

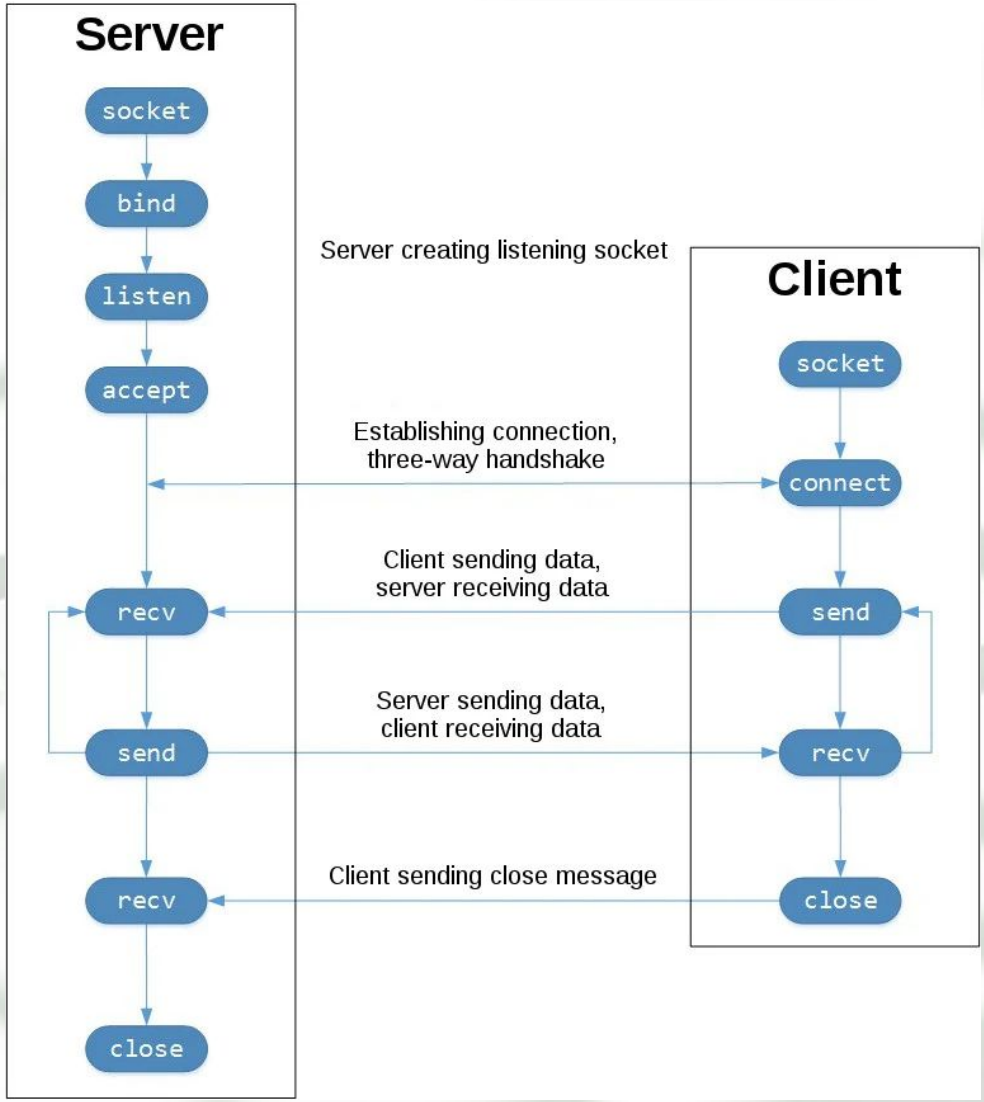
- Principais Primitivas

```
Socket()     # Cria a estrutura de dados para comunicação

#Métodos Server
.bind()      # Associa endereço e porta local à um Socket
.listen()    # Coloca-se à disposição para aceitar conexões
.accept()    # Bloqueia processo até obter conexão

#Métodos Client
.connect()   # Tenta estabelecer conexão remota com Servidor

#Métodos Client-Server
.send()      # Envia dados pela conexão
.recv()      # Recebe dados pela conexão
.close()     # Finaliza conexão
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Fluxo Sockets

- Fluxo de Primitivas
  - Padrão **TCP**

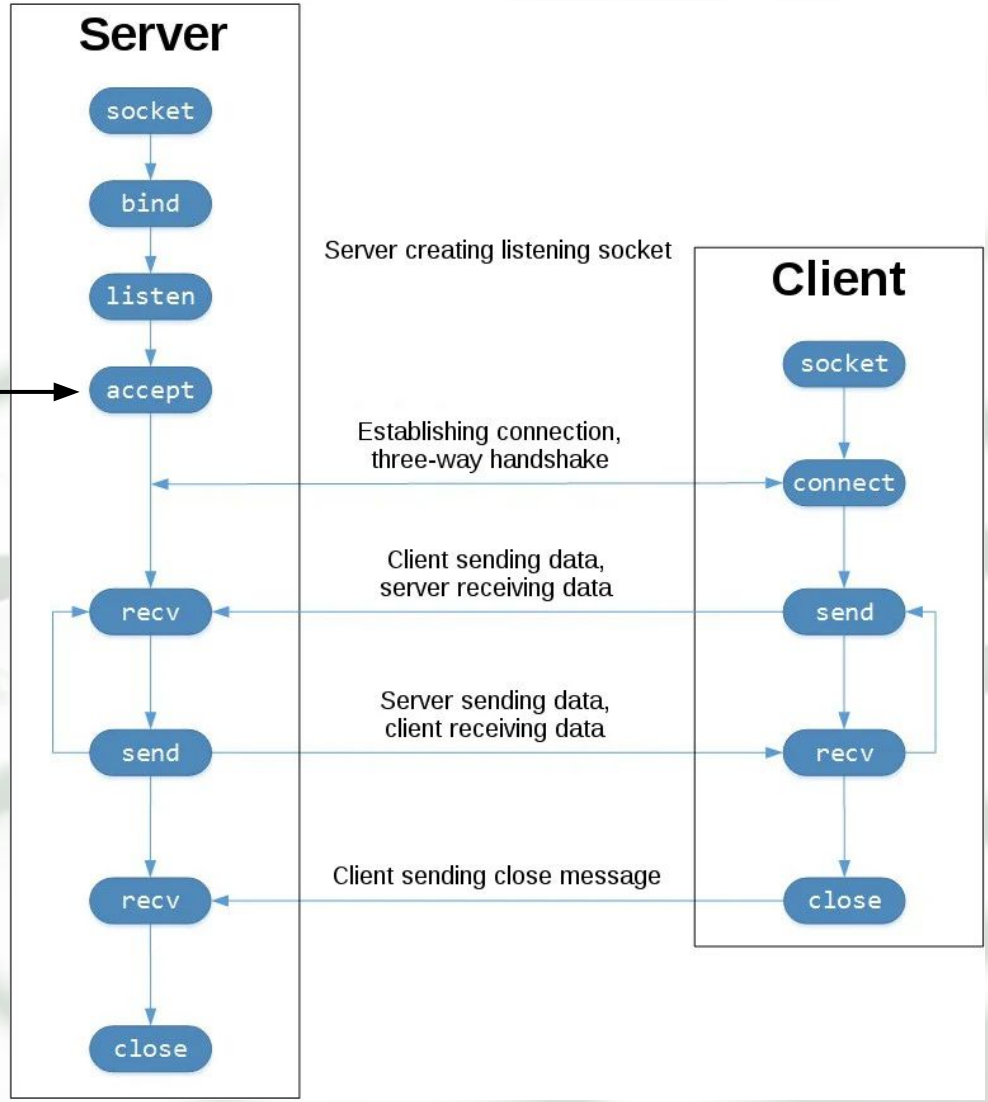


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Fluxo Sockets

- Fluxo de Primitivas
  - Padrão **TCP**

O retorno do método **accept** é um novo objeto **Socket()** contendo as informações do cliente (host, port)



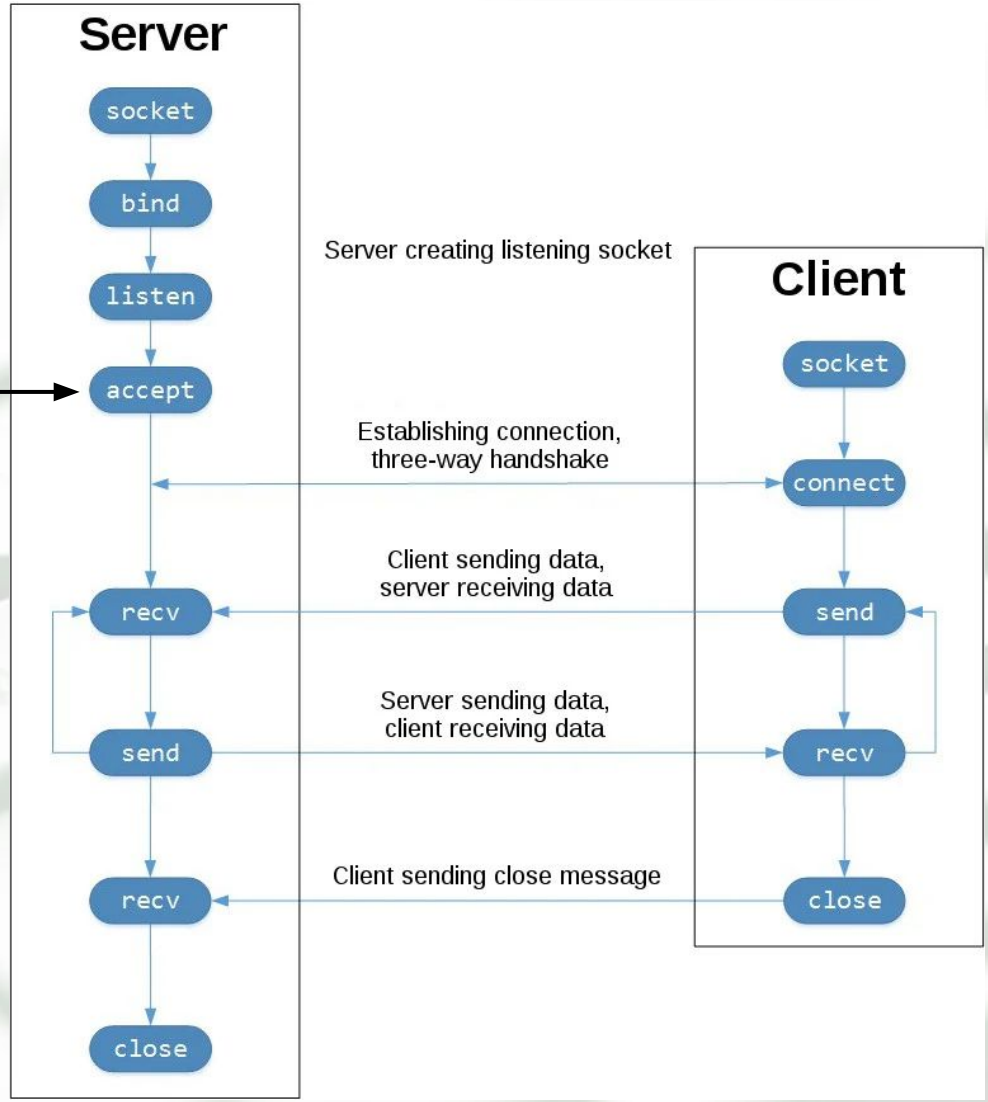
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Fluxo Sockets

- Fluxo de Primitivas
  - Padrão **TCP**

O retorno do método **accept** é um novo objeto **Socket()** contendo as informações do cliente (host, port)

Isso é muito útil pois permite a criação de um *socket* apenas para "aceitar conexões", enquanto outros sockets (oriundos deste) podem tratar cada cliente exclusivamente. **Servidor multi-client!!!**



Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'   # Interface padrão localhost / loopback
PORT = 65432         # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execuç
```

AF_INET == IPv4
AF_INET6 == IPv6
SOCK_STREAM == TCP
SOCK_DGRAM == UDP

```python
while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen() # Aguarda conexão de cliente (bloqueante)
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

conn == Socket para atender cliente
addr == Informações da conexão

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

Mensagem recebida pelo socket criado anteriormente

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close()
```

Dados enviados e recebidos devem ser codificados (serialização)

Servidor Básico Python

```python
import socket

HOST = '127.0.0.1'  # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
  conn, addr = s.accept()
  print(f'Conectado por {addr}')
  data = conn.recv(1024).decode()
  print(f'Recebido: {data}')
  conn.send(str('Hello Client').encode())
  conn.close() # Fechamento do Socket
```

Cliente Básico Python

```python
import socket

# Dados(alvo) para conexão com Servidor
SERVER = ("127.0.0.1",65432)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
  s.connect(SERVER)
  msg = "Hello Server"
  s.send(msg.encode())
  resp = s.recv(1024).decode()
  print(f'Recebido: {resp}')
except Exception as e:
  print(e)
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #07-01

- Lembra-se do **Netcat** (nc)?
  - Utilizado na disciplina de *Administração de Redes*?
- Implemente de uma aplicação similar ao Netcat.
  - *Várias trocas de mensagens numa mesma sessão.*
  - *Comportamento síncrono (pergunta-resposta).*
  - *Mesmo script, tanto para lado cliente quanto lado servidor.*
  - *Qualquer lado usa `'exit'` para terminar a sessão.*

```
# Chamada para Executar Servidor:
python lab_nc.py -lp 9007
```

```
# Chamada para Executar Cliente:
python lab_nc.py 127.0.0.1 9007
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #07-02

- Melhore a implementação da aplicação anterior, permitindo que a troca de mensagens seja **assíncrona.**
  - *Cliente e/ou servidor podem enviar uma ou mais mensagens sem ter que esperar a resposta do outro lado.*
  - *O que acontece quando dois clientes distintos tentam se comunicar com o server simultaneamente?*

```
# Chamada para Executar Servidor:
python lab_nc.py -lp 9007
```

```
# Chamada para Executar Cliente:
python lab_nc.py 127.0.0.1 9007
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #07-03

- No último Lab, adapte a aplicação para ser ***multi-client***.
  - *Ou seja, o servidor deve permitir a comunicação entre N clientes conectados na aplicação (estilo chat em grupo).*
  - *Utilize dicionário e serialização de objetos para poder enviar o **nome** e a **mensagem** de cada usuário do chat.*
  - *Sugestão: Implemente GUI*

```
# Chamada para Executar Servidor:
python lab_nc.py -lp 9007
```

```
# Chamada para Executar Cliente:
python lab_nc.py 127.0.0.1 9007
```

Referências

- VAN STEEN, Maarten; TANENBAUM, Andrew S. Distributed systems. Leiden, The Netherlands: Maarten van Steen, 2017.
- MENDES, Eduardo. Lives de Python. YouTube Channel. https://github.com/dunossauro/live-de-python
- GUEDES, Dorgival. Notas de aula, UFMG. YouTube Channel: https://www.youtube.com/channel/UCJQHsVoqmkygpOXtGfKECFw