Network Programming

Pemrograman sistem dan jaringan

Henry Saptono
2019

Network programming ?

- Pemrograman jaringan (network programming) adalah menulis kode program yang dapat melakukan komunikasi dengan program (proses) lainnya melalui jaringan komputer.
- Jaringan dapat berupa jaringan logis, jaringan lokal komputer (LAN), atau yang secara fisik terhubung ke jaringan eksternal seperti Internet

Network programming

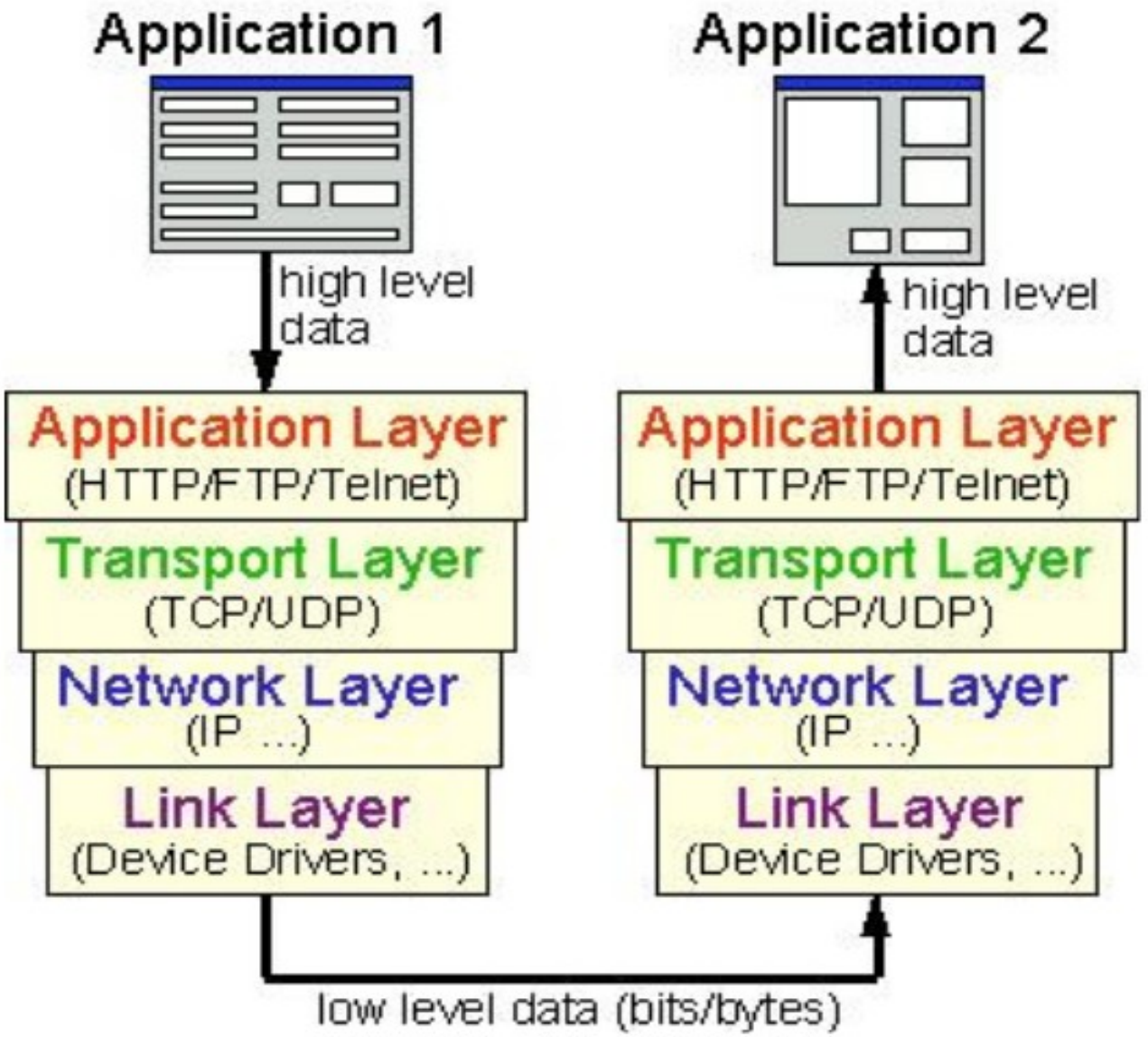
- **Komponen kunci:**
  - **Internet protocols:**
    - IP, TCP, UDP dan lain lain.
  - **Sockets:**
    - API - application programming interface

Internet protocols

- Protokol adalah pola standar pertukaran informasi
- Internet Protocol (IP)
  - IP adalah standar protokol yang secara defacto digunakan untuk komunikasi didalam jaringan komputer (internet)
  - Sederhana dan elegan, antarmuka Unix

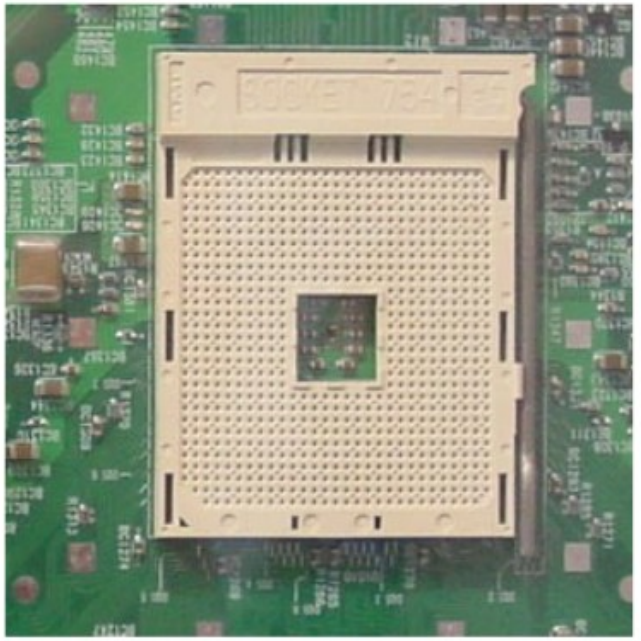
Internet layers

- komunikasi internet dibangun dari beberapa lapisan:



Sockets ?

- Socket (indonesia: soket)
- Berbagai soket ... Ada kesamaan?

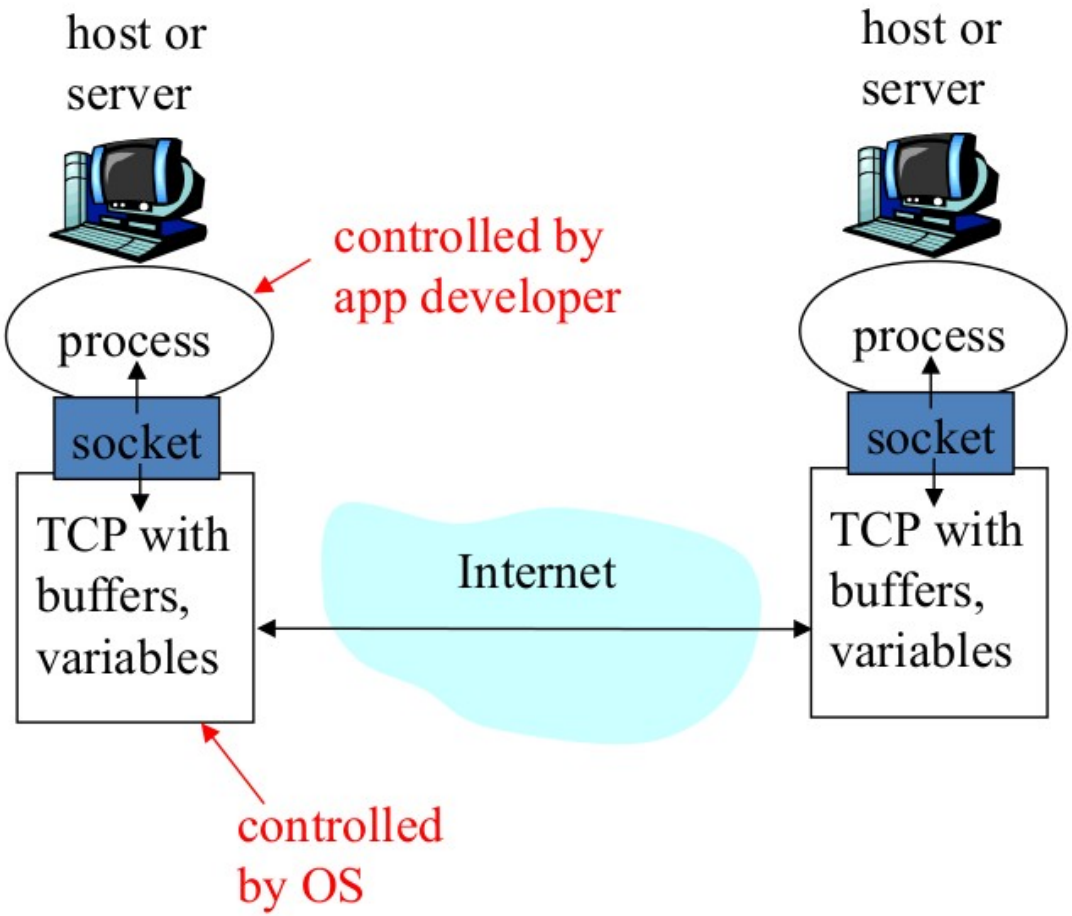


Sockets ?

- Dalam bahasa indonesia soket yang memiliki mana
  - lubang pada suatu bagian alat tempat melekat sesuatu;
  - penyambung pipa yang berulir dari dalamnya
- Dalam istilah dunia komputer / jaringan. Socket adalah titik akhir dari saluran komunikasi dua arah.
- Soket dapat berkomunikasi dalam suatu proses, antara proses pada mesin yang sama, atau antara proses yang tersebar dalam jaringan lokal atau internet.

Sockets ?

- Soket analog dengan pintu:
  - proses pengiriman mendoror pesan keluar
  - proses pengiriman mengasumsikan infrastruktur transportasi di sisi lain pintu yang membawa pesan ke soket pada proses penerima
  - aplikasi yang dibuat, dikendalikan oleh OS
  - koneksi antar soket di set-up/dikelola oleh OS



Socket API

- Socket dan socket API digunakan untuk mengirim pesan melalui jaringan. Mereka menyediakan bentuk komunikasi antar-proses (*Inter Process Communications*).
- *A transport layer service interface*
  - *Introduced in 1981 by BSD 4.1*
  - *Implemented as library and/or system calls*
  - *Low-level networking interface*
  - *Similar interfaces to TCP and UDP*
  - *Also interface to IP (for super-user); “raw sockets”*

Socket API

- Penggunaannya berasal dari ARPANET pada tahun 1971 dan kemudian menjadi API dalam sistem operasi Berkeley Software Distribution (BSD) yang dirilis pada tahun 1983 yang disebut **soket Berkeley atau socket BSD.**
- Ada juga **socket domain Unix**, yang hanya dapat digunakan untuk berkomunikasi antar proses pada host yang sama.

Socket API

- Ketika Internet lepas landas pada **1990-an** dengan **World Wide Web**, begitu pula **pemrograman jaringan**. **Web server** dan **web browser** bukan satu-satunya aplikasi yang memanfaatkan jaringan yang terhubung dan menggunakan soket. Aplikasi klien-server dari semua jenis dan ukuran mulai digunakan secara luas.
- Hari ini, meskipun protokol dasar yang digunakan oleh socket API telah berkembang selama bertahun-tahun, dan adanya hal-hal baru, **API tingkat rendah tetap sama.**

Socket API

- Jenis aplikasi soket yang paling umum adalah aplikasi klien-server, di mana satu sisi bertindak sebagai server dan menunggu koneksi dari klien.

Pemrograman jaringan dengan Python

Python Network programming

- Python menyediakan dua tingkat akses ke layanan jaringan:
  - ***Low-Level Access***
    - Pada *low level access*, Anda dapat mengakses dukungan soket dasar dalam sistem operasi yang mendasarinya, yang memungkinkan Anda untuk mengimplementasikan klien dan server untuk kedua protokol berorientasi koneksi dan tanpa sambungan.
  - ***High-Level Access***
    - Python juga memiliki pustaka yang menyediakan akses tingkat lebih tinggi (*high level access*) ke protokol jaringan tingkat aplikasi tertentu, seperti FTP, HTTP, dan lain lain.

Socket API Overview

- Modul soket Python menyediakan antarmuka ke Berkeley sockets API. Ini adalah modul yang akan kita gunakan dalam pembahasan materi pemrograman jaringan
- Fungsi dan metode utama dari socket API dalam modul ini adalah:

  Socket(), bind(), listen(), accept(), connect(), connect_ex(), send(), recv(), close()

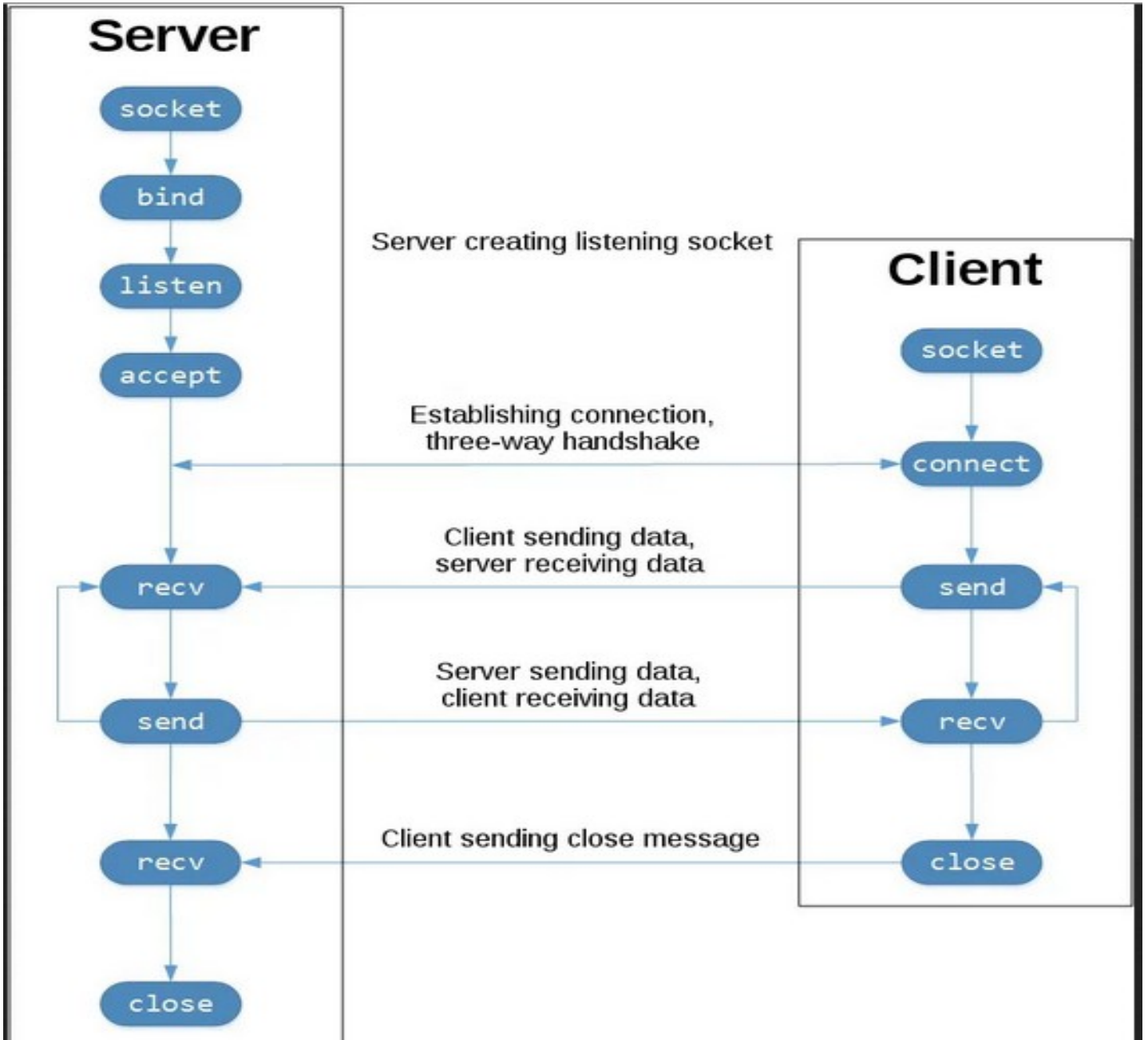
TCP Sockets

- Mengapa Anda harus menggunakan TCP? Protokol Kontrol Transmisi (TCP):
  - Dapat diandalkan: paket yang di-drop dalam jaringan terdeteksi dan dikirim kembali oleh pengirim.
  - Memiliki *in-order data delivery*: data dibaca oleh aplikasi penerima dalam urutan yang ditulis oleh pengirim.
- Sebaliknya, User Datagram Protocol (UDP) Socket yang dibuat dengan tidak dapat diandalkan, dan data yang dibaca oleh penerima dapat tidak sesuai pesanan dari tulisan pengirim.

TCP Sockets

- Mengapa ini penting? Jaringan adalah *best-effort delivery system*. Tidak ada jaminan bahwa data yang dikirim akan mencapai tujuannya atau bahwa program Anda akan menerima apa yang telah dikirimkan kepada program Anda.
- Perangkat jaringan (misalnya, router dan switch), memiliki bandwidth terbatas dan keterbatasan sistem inheren mereka sendiri. Mereka memiliki CPU, memori, bus, dan buffer paket antarmuka. TCP membebaskan program Anda dari keharusan khawatir tentang kehilangan paket, data yang tiba-tiba rusak, dan banyak hal lain yang selalu terjadi ketika program berkomunikasi melalui jaringan.

TCP Socket Flow



TCP Socket Flow

- perhatikan panggilan API yang dibuat server untuk menyiapkan soket "**listening**":
  - socket()
  - bind()
  - listen()
  - Accept()
- Soket “**listening**” Itu mendengarkan koneksi dari klien. Ketika klien terhubung, server memanggil **accept()** untuk menerima, atau menyelesaikan, koneksi.

TCP Socket Flow

- klien memanggil **connect()** untuk membuat koneksi ke server dan memulai jabat tangan tiga arah (*three-way handshake*).
- Langkah jabat tangan penting karena memastikan bahwa setiap sisi koneksi dapat dijangkau dalam jaringan, dengan kata lain bahwa klien dapat mencapai server dan sebaliknya.

TCP Socket Flow

- Di bagian tengah adalah bagian pulang-pergi, di mana data dipertukarkan antara klien dan server menggunakan panggilan untuk mengirim **send()** dan menerima **recv().**
- Di bagian bawah, klien dan server menutupsoket masing-masing. dengan memanggil methode **close()**

Example:
Echo Client and Server

Echo Server

```
import socket
HOST = '127.0.0.1'
address (localhost)
PORT = 65432

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()

while True:
   conn, addr = s.accept()
   print('Connected by', addr)
   data = conn.recv(1024)
   if not data:
                continue
   conn.sendall(data)
```

Echo Client

```
import socket
HOST = '127.0.0.1'
PORT = 65432
S = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.sendall(b'Hello, world')
data = s.recv(1024)

print('Received', repr(data))
```

Menjalankan server dan client

- Untuk menjalankan program echo-server, sbb:

```
$ python echo-server.py
```

- Untuk menjalankan program echo-client, sbb:

```
$ python echo-client.py
```

Melihat Status Soket

- Untuk melihat keadaan soket saat ini di komputer(host) Anda, gunakan perintah **netstat**.
- Inilah output netstat dari linux setelah memulai server:

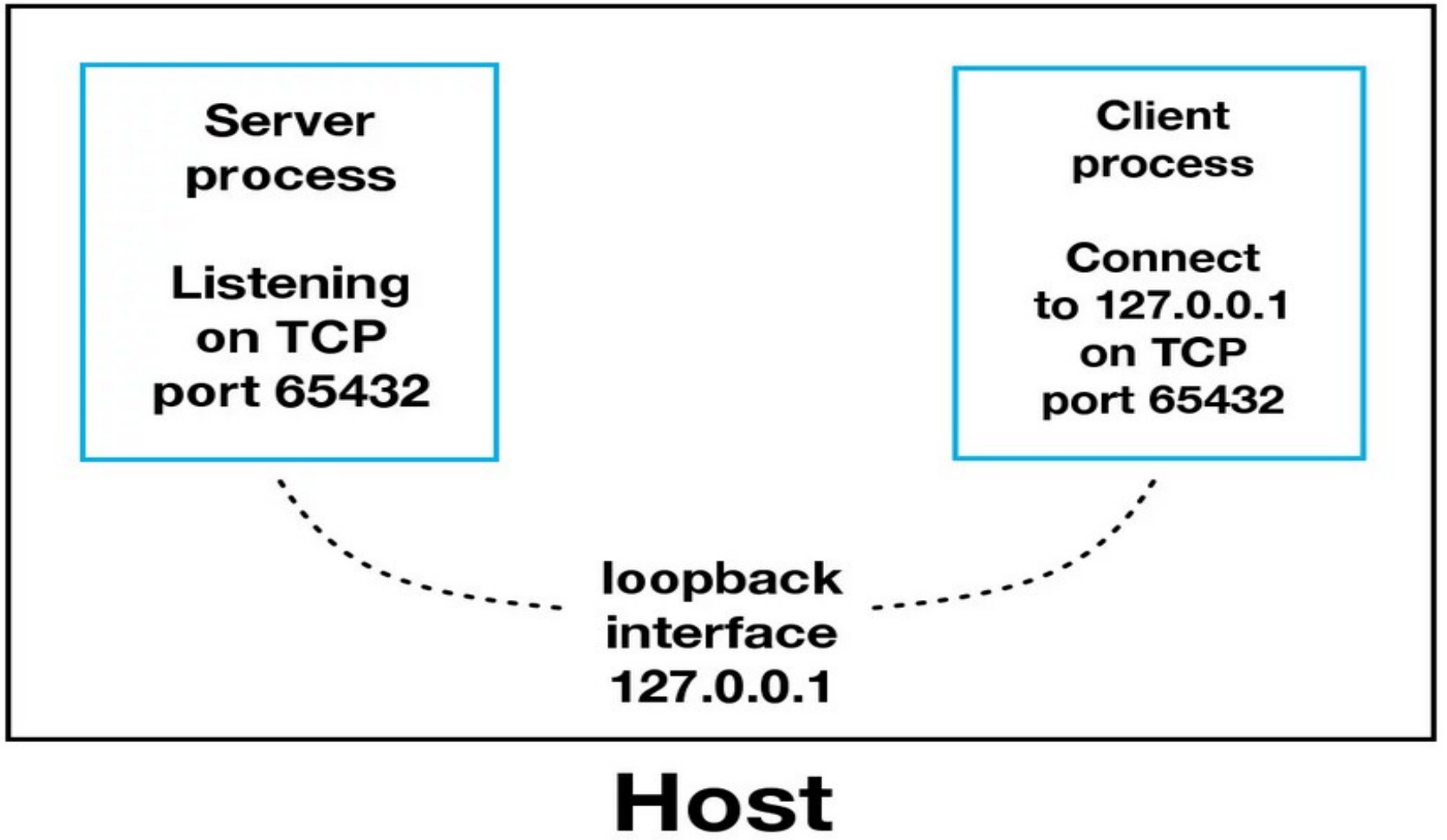
$ netstat -tanp

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	127.0.0.1:65432	0.0.0.0:*	LISTEN	7551/python3

Communication Breakdown

- Mari kita lihat lebih dekat bagaimana klien dan server berkomunikasi satu sama lain:



Communication Breakdown

- Saat menggunakan antarmuka loopback (alamat IPv4 127.0.0.1 atau alamat IPv6 :: 1), data tidak pernah meninggalkan host atau menyentuh jaringan eksternal.
- Dalam diagram di atas, antarmuka loopback terdapat di dalam host. Ini mewakili sifat internal antarmuka loopback dan bahwa koneksi dan data yang transit ke lokal host. Inilah sebabnya mengapa Anda juga akan mendengar antarmuka loopback dan alamat IP 127.0.0.1 atau :: 1 disebut sebagai "localhost."

Communication Breakdown

- Saat Anda menggunakan alamat IP selain dari 127.0.0.1 atau :: 1 di aplikasi Anda, ini mungkin terikat ke antarmuka Ethernet yang terhubung ke jaringan eksternal