Networking

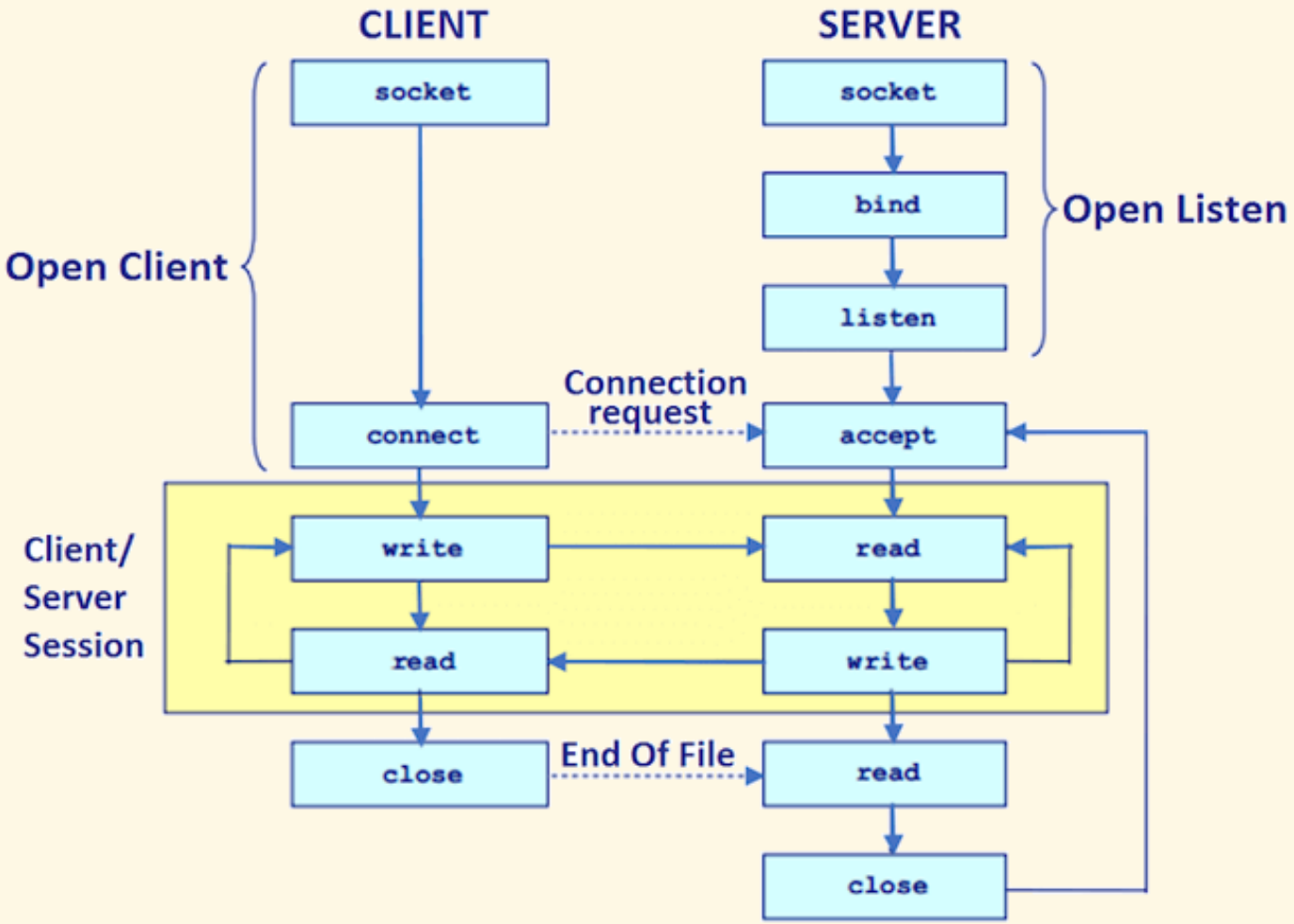
- Connection-Oriented Socket
- Connection-less Socket

Introduction

- Networking is about connecting two or more devices together so that we can share resources
- TCP/IP basics
  - TCP (Transmission Control Protocol): a reliable connection-oriented communication protocol between the sender and receiver
  - UDP (User Datagram Protocol): a connection-less protocol service allowing packet of data to be transferred along two or more nodes
  - IP address, MAC address, protocol, port, socket,…
- `java.net` package has 2 parts:
  - Low-level API: deals with the abstractions of addresses, sockets, network interfaces
  - High-level API: deals with the abstraction of URIs, URLs, and connections

Socket Programming

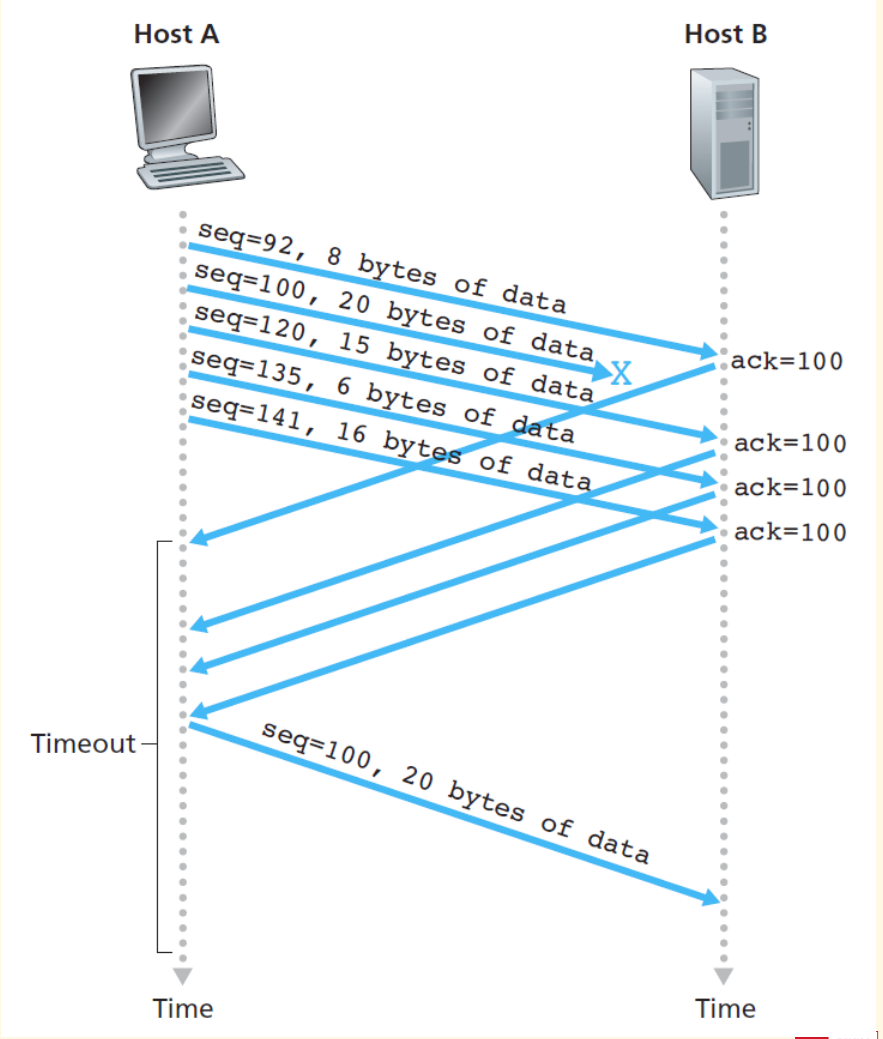
- Socket programming is used for communication between the applications
  - Can be connection-oriented or connection-less
- The most common model is client/server
  - Server waits for connection on a specific port
  - Clients connect to server knowing its address and port



Connection-Oriented Socket

TCP

- A reliable protocol with mechanisms to:
  - Ensure that data is not damaged, lost, duplicated, out of order
  - Data is fully transmitted
  - Recover errors if necessary



Simple Client Connecting to a HTTP Server

```
try (Socket s = new Socket("server-address", 80);
     PrintStream out = new PrintStream(s.getOutputStream());
     DataInputStream in = new DataInputStream(s.getInputStream())
) {
     out.println("GET / HTTP/1.0");
     out.println("");
     out.flush();

     byte[] buffer = new byte[512];
     int n;
     do {
          n = in.read(buffer);
          System.out.print(new String(
               Arrays.copyOfRange(buffer, 0, n)));
     } while (n == buffer.length);
}
```

- Full example: `HttpClientExample.java`

Server

```
try (
    ServerSocket ss = new ServerSocket(3000);
    Socket s = ss.accept();
    PrintWriter out = new PrintWriter(s.getOutputStream());
    BufferedReader in = new BufferedReader(
        new InputStreamReader(s.getInputStream()))
) {
    String pingMessage = in.readLine();
    assert(pingMessage.equals("ping"));

    out.println("pong");
    out.flush();
}
```

- Full example: `EchoServer.java`, `EchoClient.java`

Supporting Multiple Clients

- The `EchoServer` serves only one client then terminates
- To support multiple clients, put the connection handling in a loop (full example in `EchoServerSupportingMultipleClients.java`):

```
while (true) {
    try (
        Socket s = ss.accept();
        PrintWriter out = new PrintWriter(s.getOutputStream());
        BufferedReader in = new BufferedReader(
            new InputStreamReader(s.getInputStream()))
    ) {
        // ...
    }
}
```

- The connections are accepted and processed sequentially:
  - Subsequence incoming connections are queued if another is being processed
  - Only acceptable if the processing is very simple and fast

Multithreaded Server

- In general cases, the server should create a new thread to process each client request, so that other incoming connections can be accepted ASAP
- Example (see `CalculationServer.java`):
  - ```
    while (true) {
        Socket s = ss.accept();
        new Thread(() -> {
            try (s;
                PrintWriter out = new PrintWriter(s.getOutputStream());
                Scanner in = new Scanner(s.getInputStream())
            ) {
                // ...
            }
        }).start();
    }
    ```
    ```

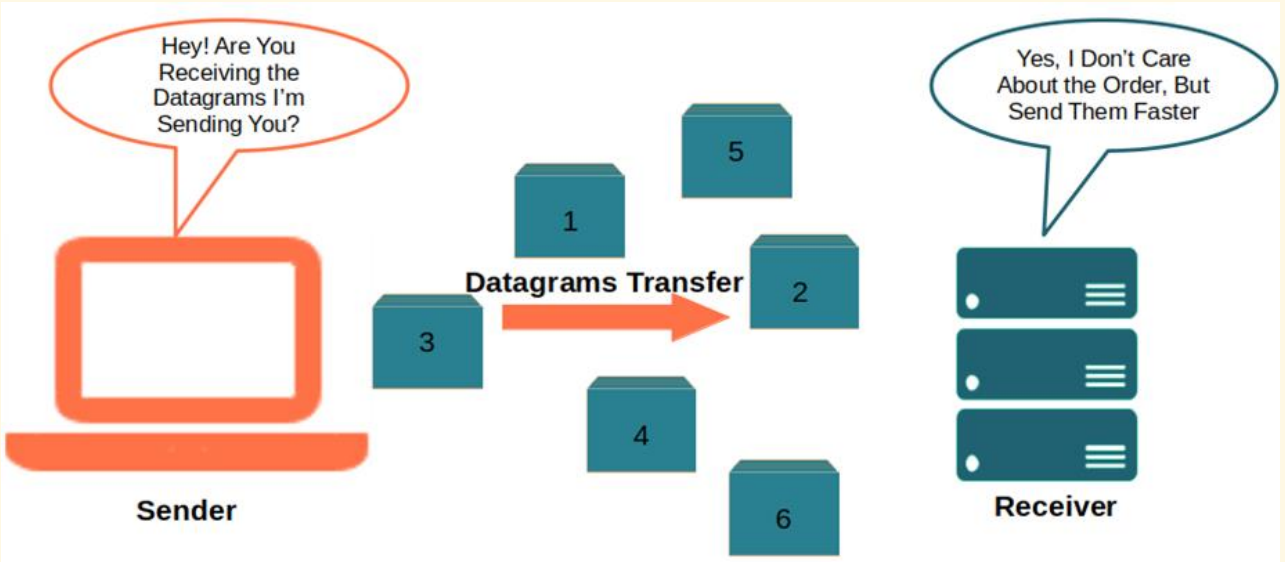
Exercise

- Use the thread pool model to reimplement the calculation server
  - Reuse threads
  - Avoid too many open threads

Connection-less Socket

Introduction

- Some applications that you write to communicate over the network will not require the reliable, point-to-point channel provided by TCP
  - UDP allows applications to send packets of data, called *datagrams*
  - A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, order correctness and content integrity are not guaranteed
  - But it's light and fast
- Useful scenarios:
  - VoIP, live video streaming
  - Multiplayer online games
  - IoT sensors data collection



UDP Client Example

- An IoT sensor node:

```
try (DatagramSocket ds = new DatagramSocket()) {
  for (;;) {
    double value = Math.random() * 20 + 100.;
    String data = value + "@" +
      new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());

    InetAddress ip = InetAddress.getByName("localhost");
    DatagramPacket dp = new DatagramPacket(
      data.getBytes(), data.length(), ip, 3000);
    ds.send(dp);

    System.out.println("Message sent: " + data);

    Thread.sleep(500);
  }
}
```

UDP Server Example

- An IoT sink node:

```
try (DatagramSocket ds = new DatagramSocket(3000)) {
  System.out.println("Sink is ready...");

  for (;;) {
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, buf.length);
    ds.receive(dp);

    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
  }
}
```

Exercises

1. Write a client/server program to transfer a file
2. Write a client/server program allowing to chat from 2 machines