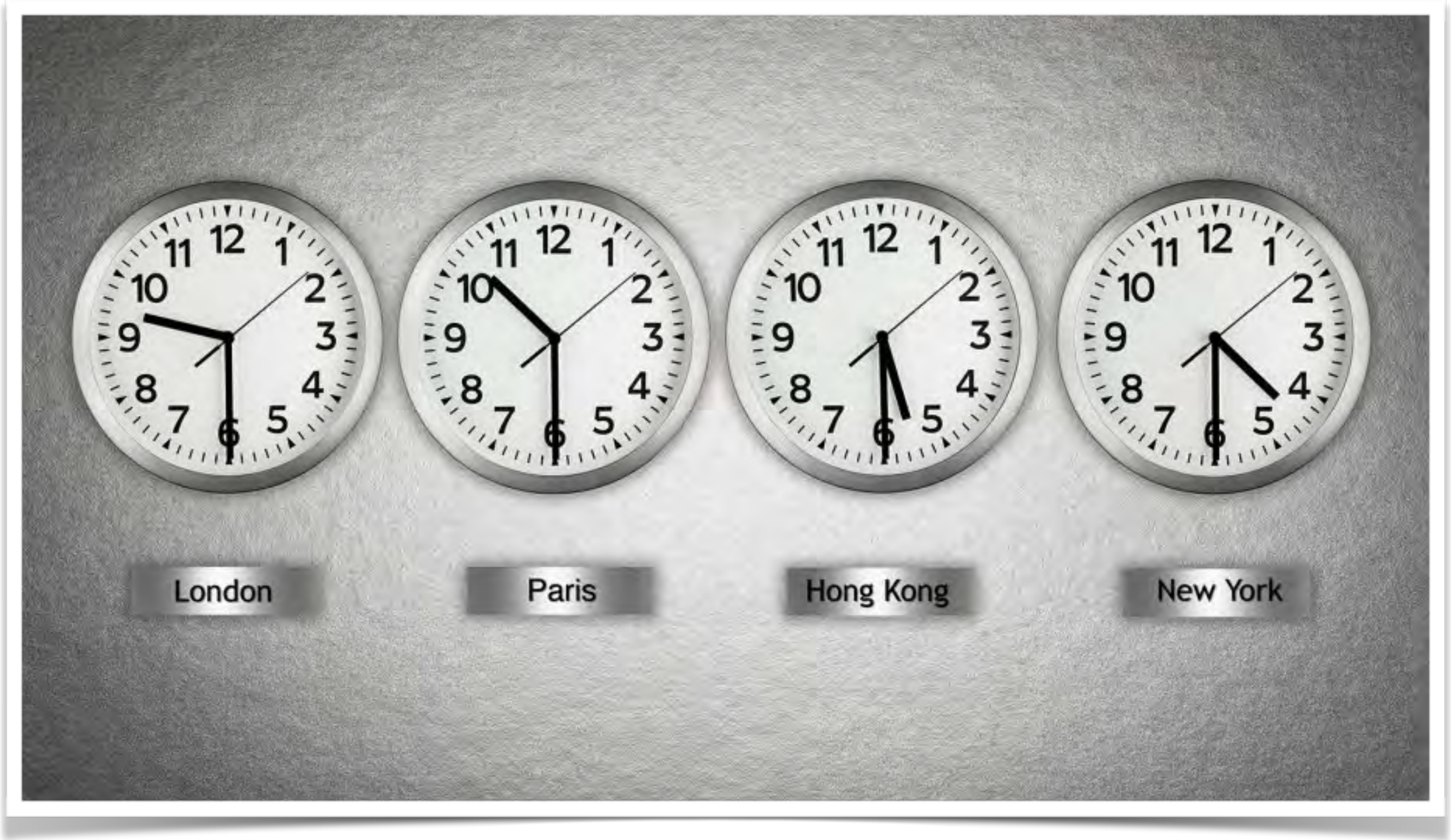
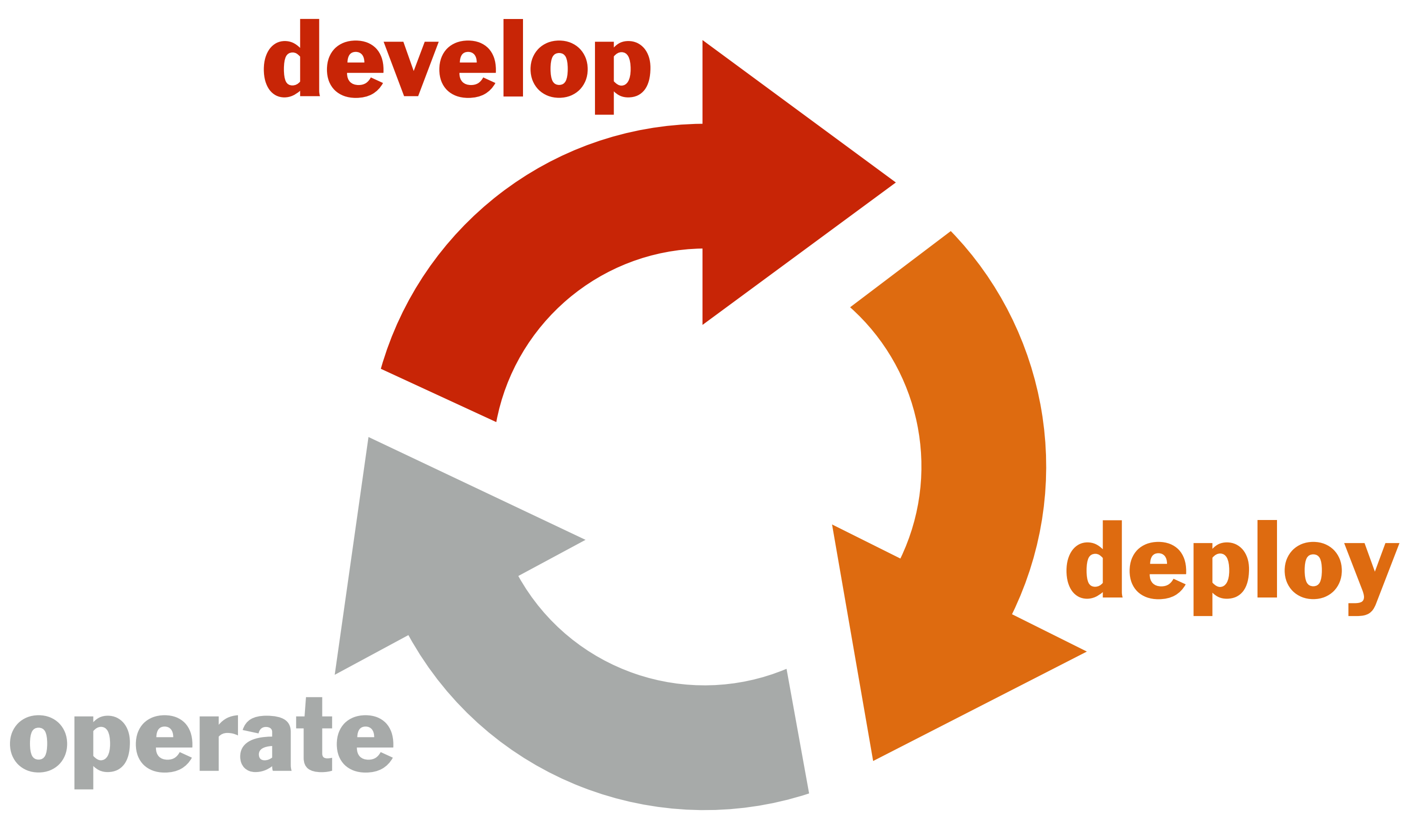
asynchronous



interactions

learning objectives



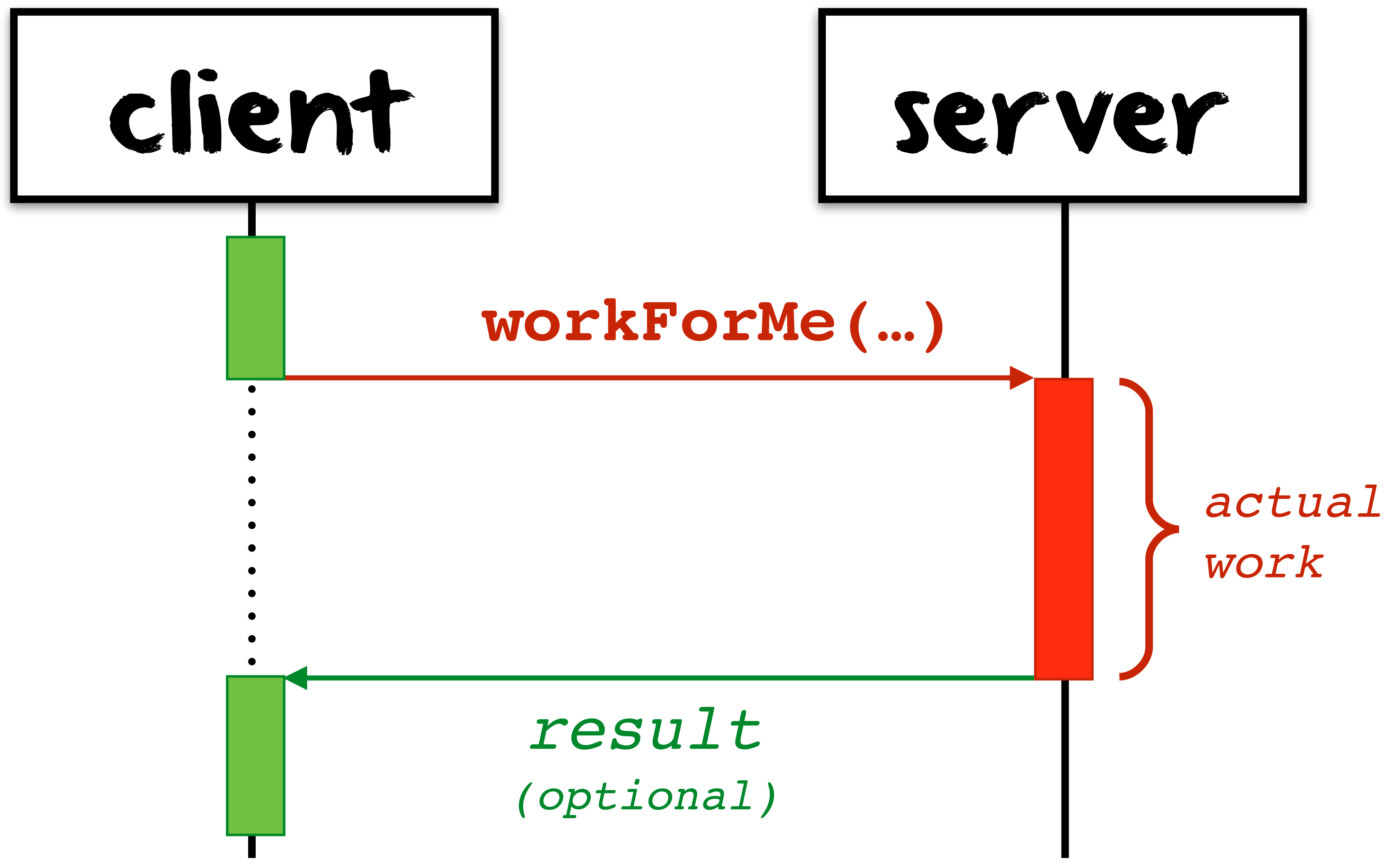
- learn what asynchronous interactions are
- learn about asynchronous methods in java
- learn about tcp/udp sockets and web sockets
- learn about message-oriented middleware and jms*

*java messaging service

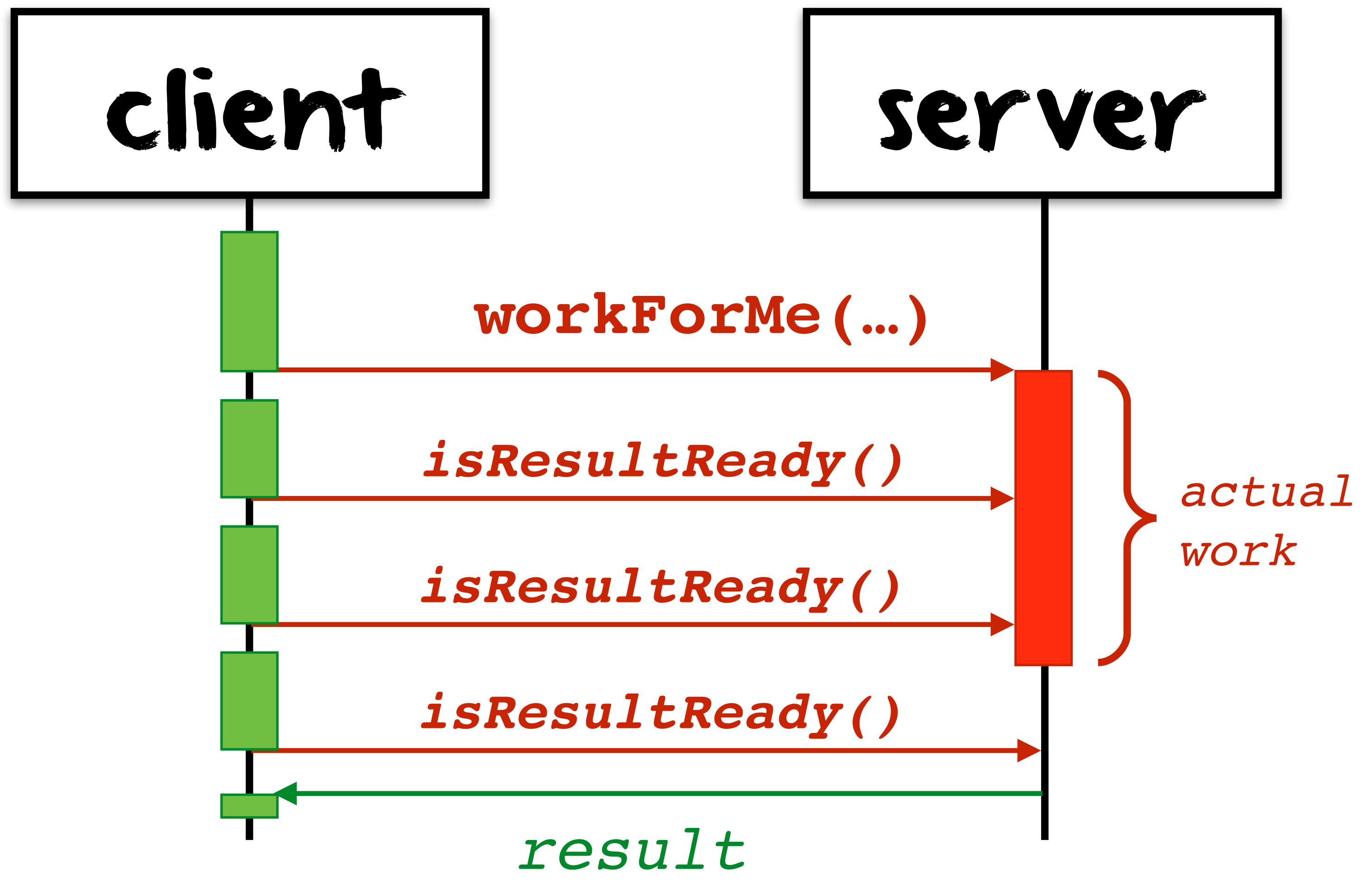
what is an asynchronous interaction?

no blocking of the client until the server is done

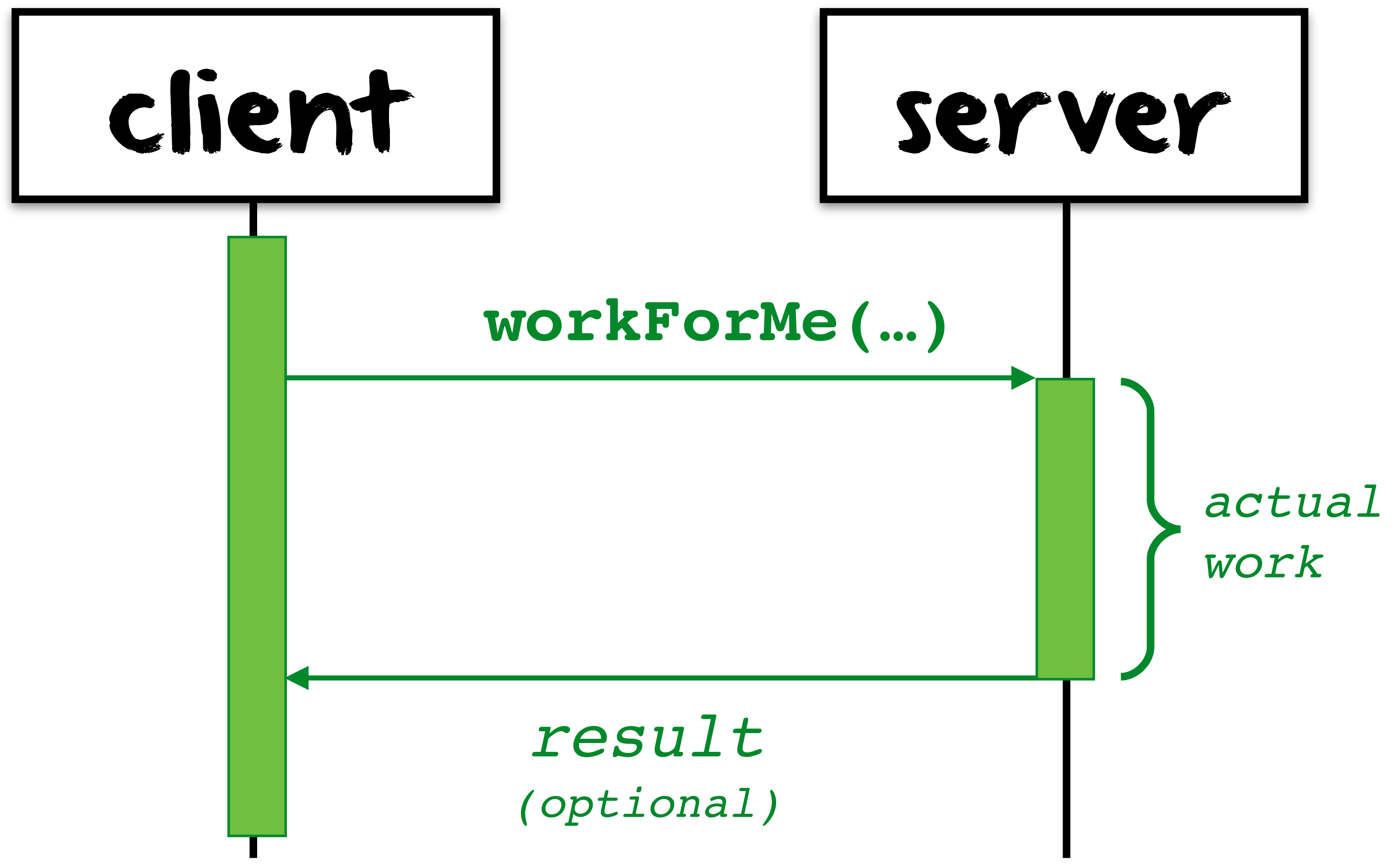
no polling by the client when a result is expected from the server



synchronous



polling



asynchronous

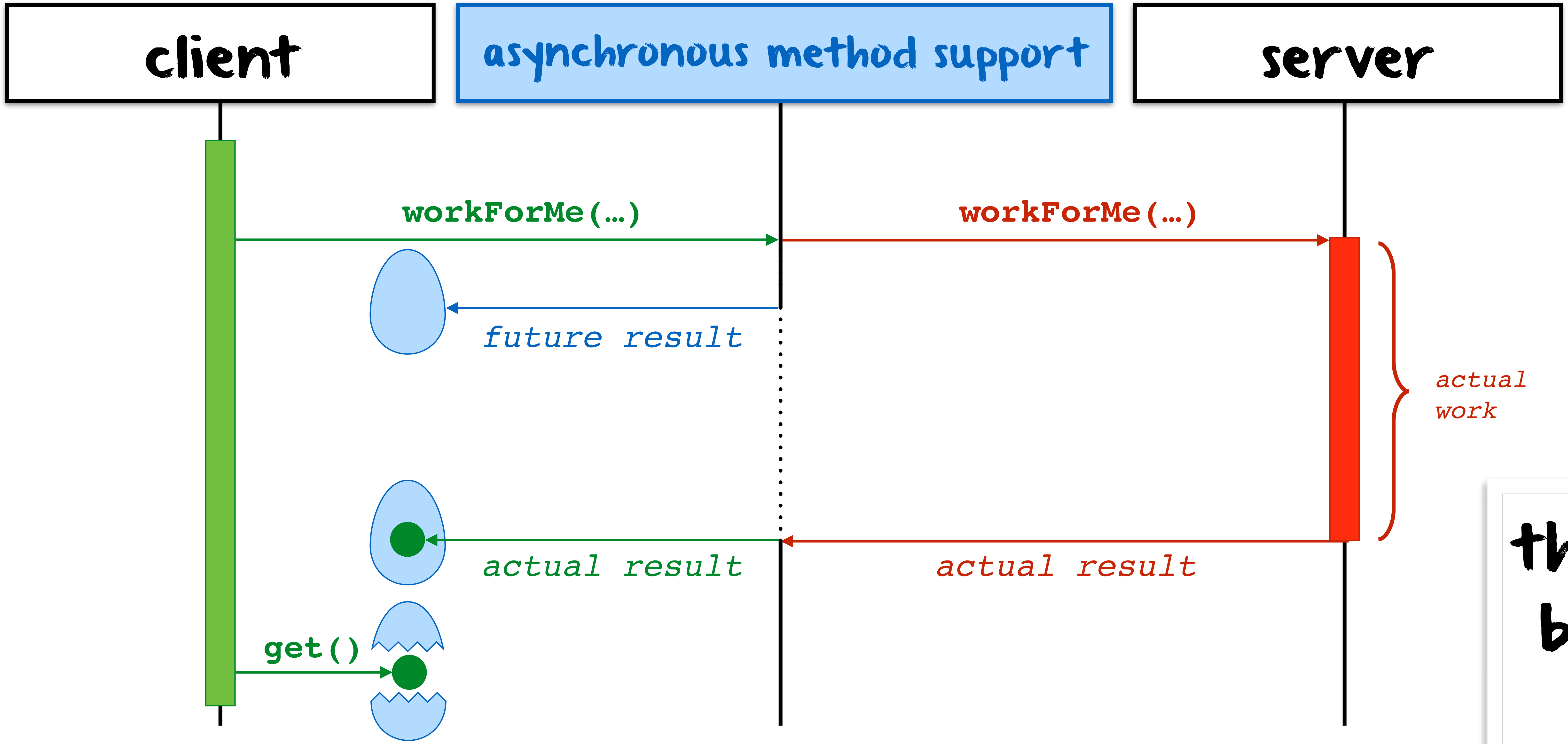
asynchronous interactions allow to achieve time decoupling

asynchronous methods

they rely on the notion of future objects

these objects are also called promises

a session bean can implement asynchronous methods



the container returns the control to the client before the method is actually invoked in background

the client can try to get but might be blocked if it is not ready yet

asynchronous methods

server side

```
@Remote
public interface PortfolioRemote {
 public Future<Double> computeValue();
}
```

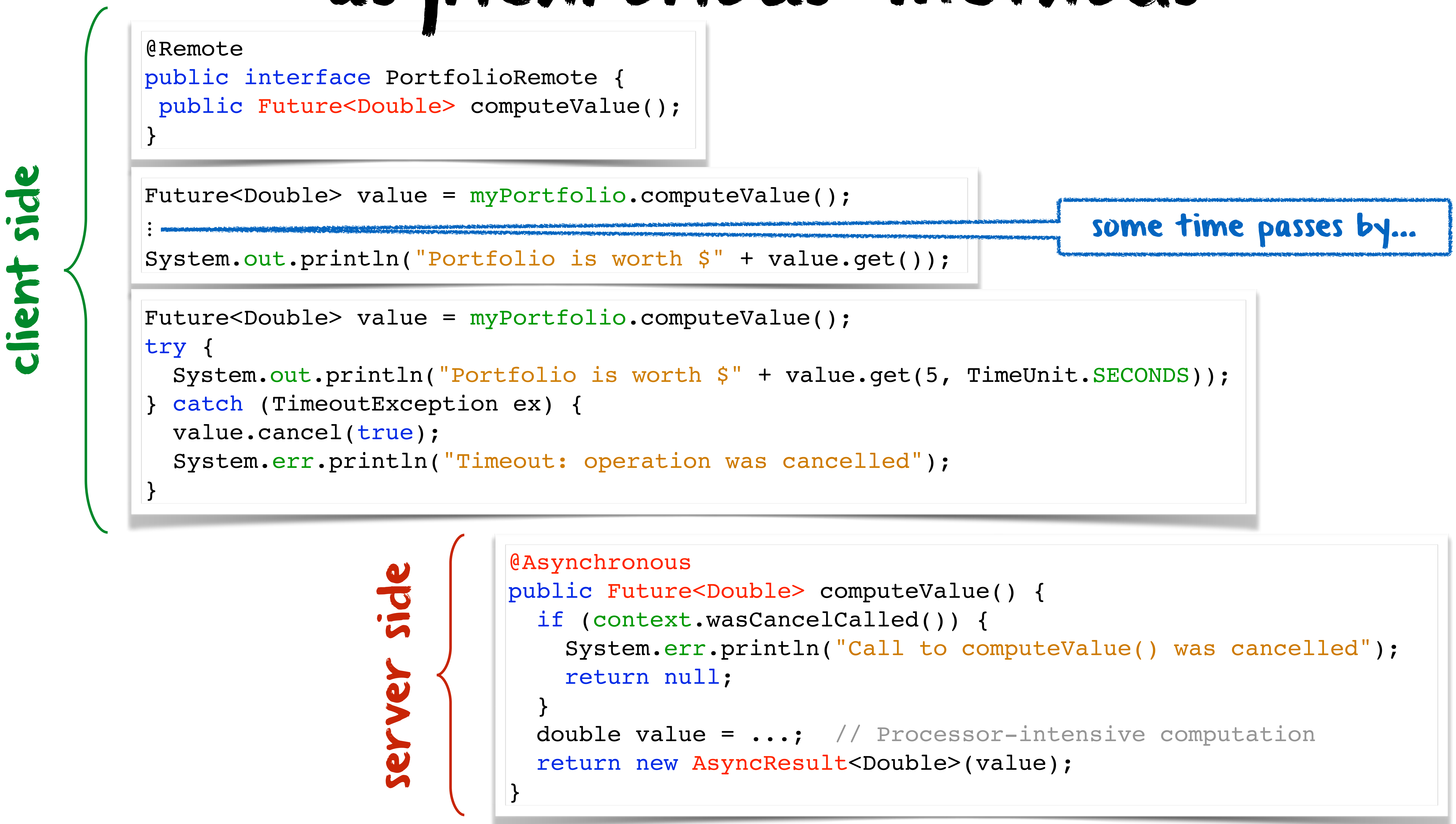
an asynchronous method must return **void** or a **Future<V>** object

```
@Stateful
public class Portfolio implements PortfolioRemote {
  @Resource
  SessionContext context;
  :
  @Asynchronous
  public Future<Double> computeValue() {
    double value = ...;  // Processor-intensive computation
    return new AsyncResult<Double>(value);
  }
}
```

if it returns **void**, it cannot declare exceptions

the client can use the **Future<V>** object to retrieve the actual result *or to cancel the invocation*

asynchronous methods



client side

```
@Remote
public interface PortfolioRemote {
 public Future<Double> computeValue();
}
```

```
Future<Double> value = myPortfolio.computeValue();
:
System.out.println("Portfolio is worth $" + value.get());
```

some time passes by...

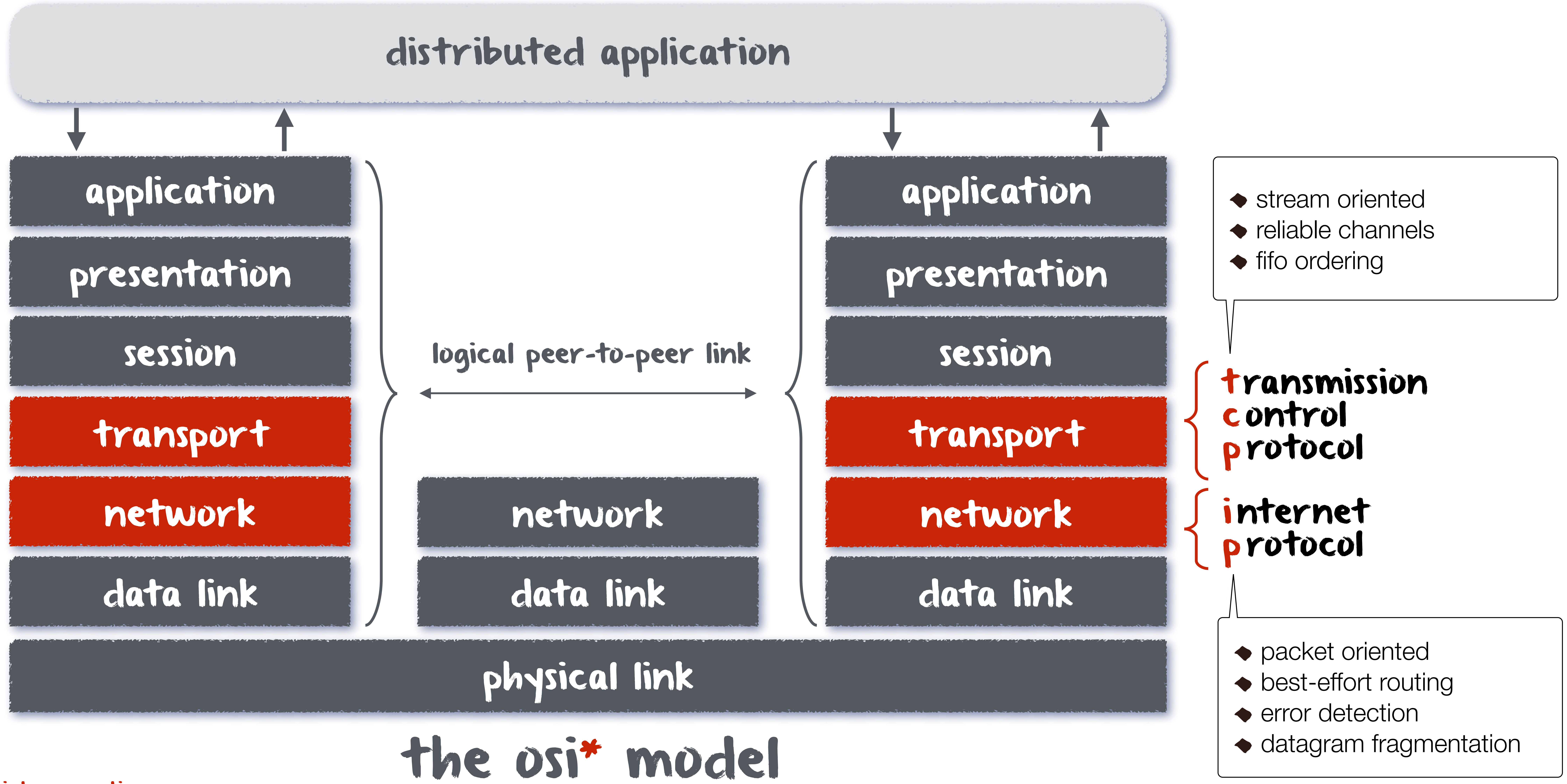
```
Future<Double> value = myPortfolio.computeValue();
try {
  System.out.println("Portfolio is worth $" + value.get(5, TimeUnit.SECONDS));
} catch (TimeoutException ex) {
  value.cancel(true);
  System.err.println("Timeout: operation was cancelled");
}
```

server side

```
@Asynchronous
public Future<Double> computeValue() {
  if (context.wasCancelCalled()) {
    System.err.println("Call to computeValue() was cancelled");
    return null;
  }
  double value = ...;  // Processor-intensive computation
  return new AsyncResult<Double>(value);
}
```

asynchronous messaging

using sockets



- stream oriented
- reliable channels
- fifo ordering

transmission **c**ontrol **p**rotocol

internet **p**rotocol

- packet oriented
- best-effort routing
- error detection
- datagram fragmentation

*open systems interconnection

asynchronous messaging

using sockets

internet protocol

an **ip address** is used by the ip protocol to address computers and routers

an ip **v4** address consists of 32-bits (4 bytes) and is often written in **dotted decimal format**, e.g., 130.223.171.8

Class	First byte	Networks			Hosts		Address format	
A	1→126	$2^7 - 2$	=	126	$2^{24} - 2 =$	16’777’214	net id	host id
B	128→191	2^{14}	=	16’384	$2^{16} - 2 =$	65’534	net id	host id
C	192→223	2^{21}	=	2’097’152	$2^8 - 2 =$	254	net id	host id
D	224→239						multicast	
E	240→247						reserved	

asynchronous messaging

using sockets

internet protocol

ip v4 address

Class	Format
A	0NNNNNNN.HHHHHHHH.HHHHHHHH.HHHHHHHH
B	10NNNNNN.NNNNNNNN.HHHHHHHH.HHHHHHHH
C	110NNNNN.NNNNNNNN.NNNNNNNN.HHHHHHHH
D	1110MMMM.MMMMMMMM.MMMMMMMM.MMMMMMMM
E	1111RRRR.RRRRRRRR.RRRRRRRR.RRRRRRRR

N network ID bits

H host ID bits

M multicast address bit

R reserved bits

asynchronous messaging

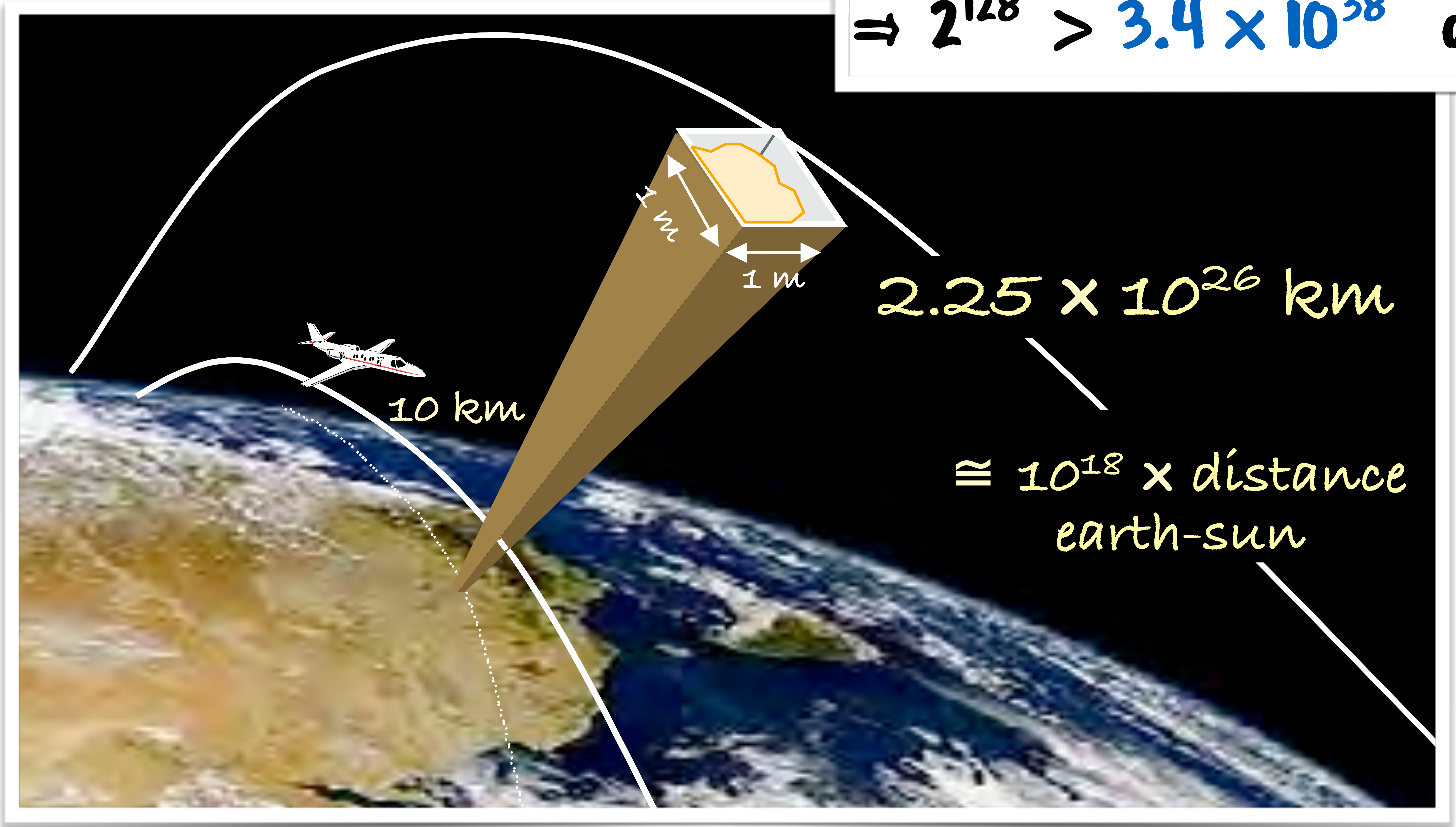
using sockets

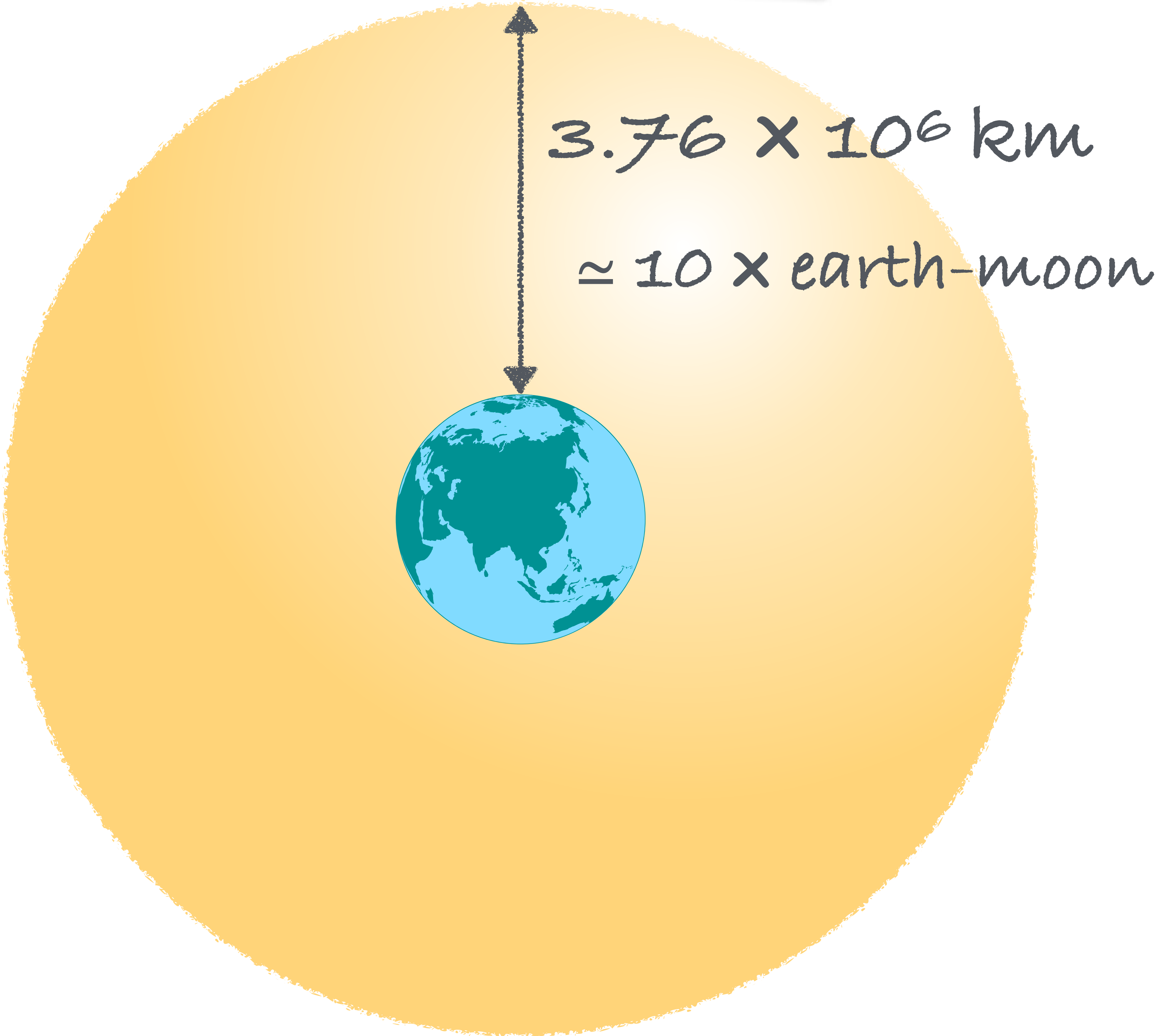
internet protocol

ip v6 address

addresses encoded on 128 bits

⇒ $2^{128} > 3.4 \times 10^{38}$ available addresses





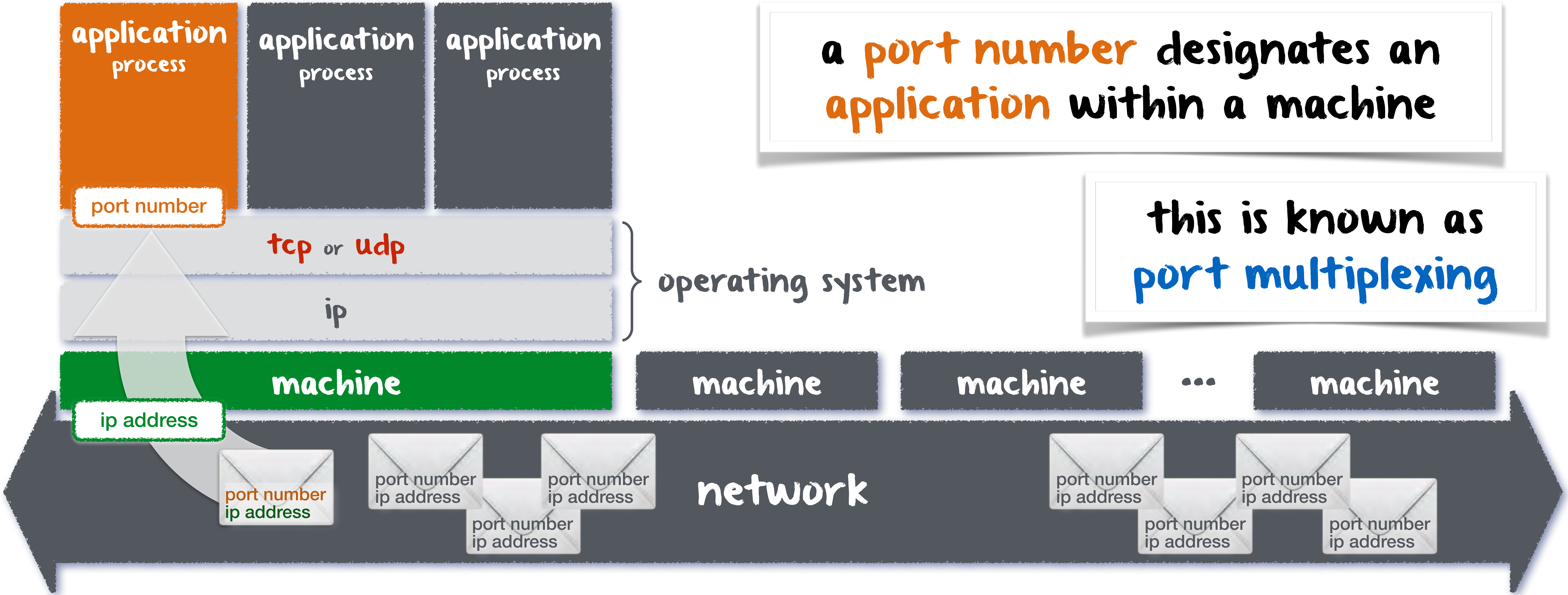
asynchronous messaging
using sockets

addressing applications

an ip address designates a machine

a port number designates an application within a machine

this is known as port multiplexing



asynchronous messaging

using sockets

sockets are programming abstractions representing bidirectional communication channels between processes

there exists two types of sockets, namely tcp sockets and upd sockets

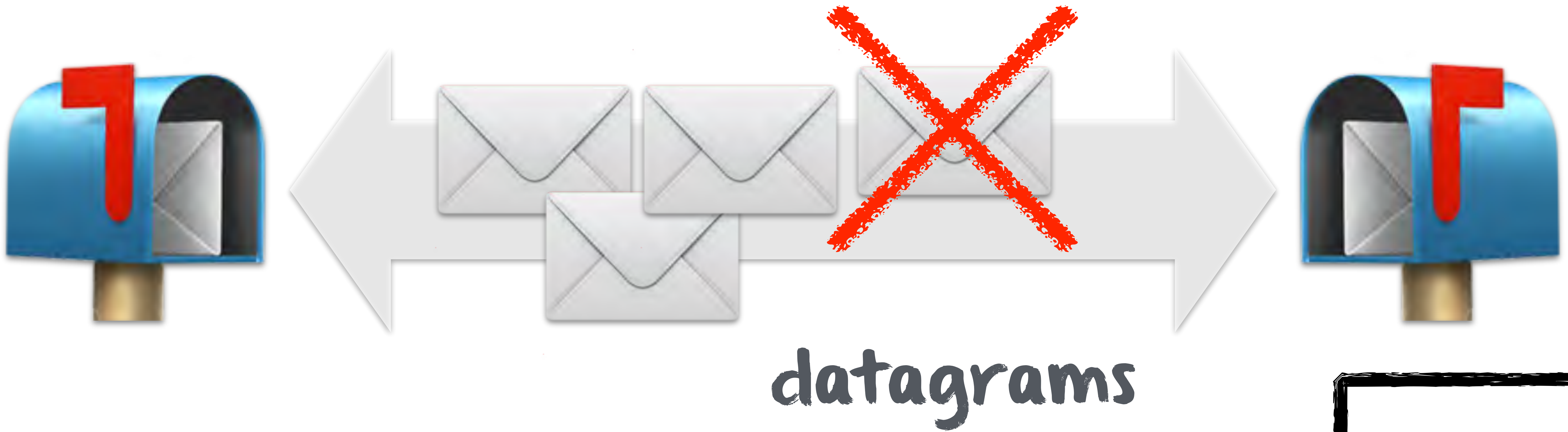
in java, sockets are instances of various classes found in the `java.net` package

asynchronous messaging

using sockets

transmission
control
protocol

...bla
bla
bla
bla
bla
bla...
stream



user
datagram
protocol

tcp and udp exhibit dual features

	connection oriented	reliable channels	fifo ordering	message oriented
TCP	YES	YES	YES	NO
UDP	NO	NO	NO	YES

asynchronous messaging
using sockets

tcp sockets

since tcp is connection-oriented, we have two classes for tcp sockets in java

client

```
public class Socket {
    ⋮
    public Socket(String host, int port) {…}
    public OutputStream getOutputStream() {…}
    public InputStream getInputStream() {{…}
    public void close() {…}
    ⋮
}
```

server

```
public class ServerSocket {
    ⋮
    public ServerSocket(int port) {…}
    public Socket accept() {…}
    ⋮
}
```

this captures the asymmetry when establishing a communication channel

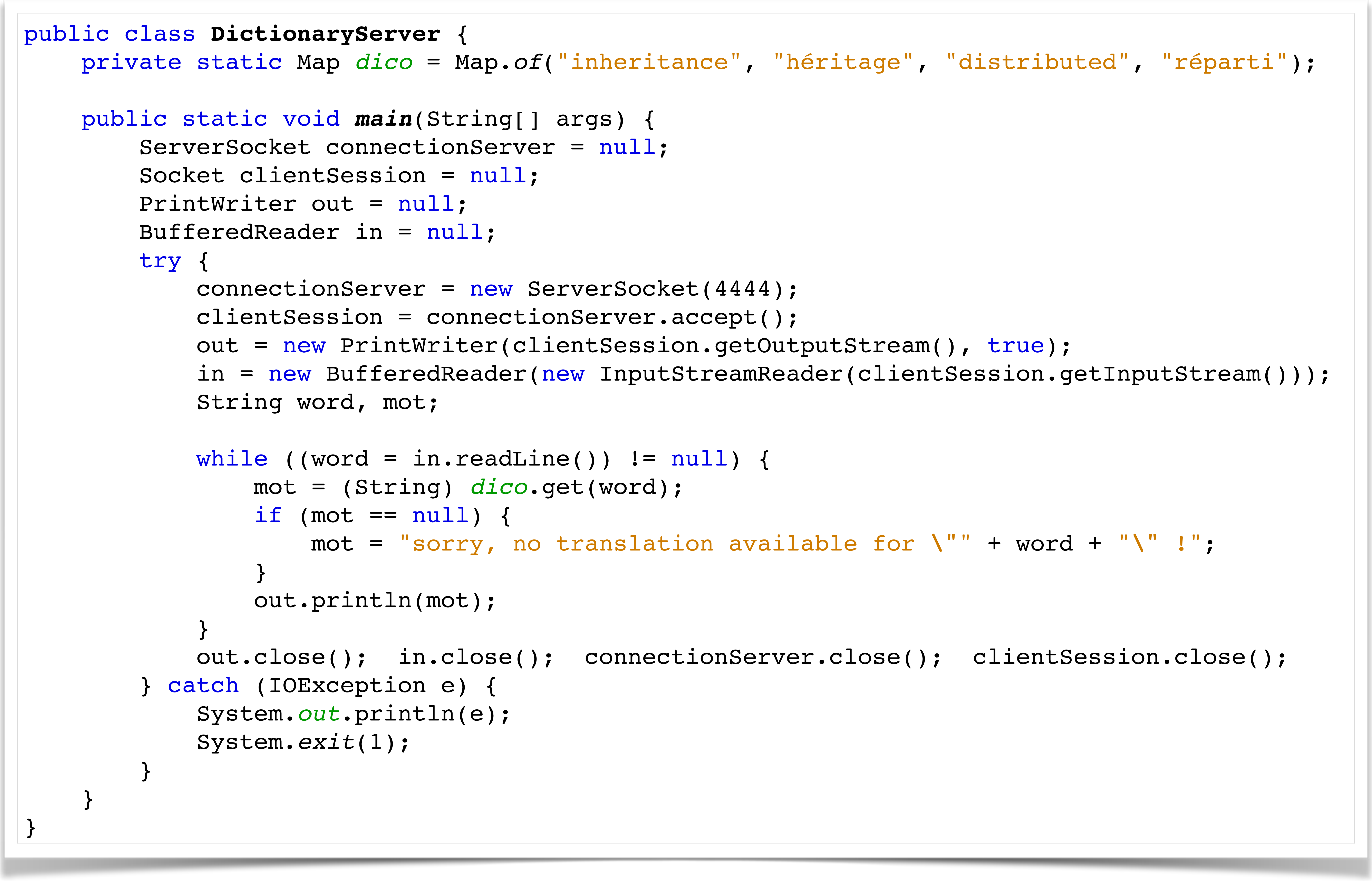
asynchronous messaging

using sockets

tcp

sockets

server



```
public class DictionaryServer {
    private static Map dico = Map.of("inheritance", "héritage", "distributed", "réparti");

    public static void main(String[] args) {
        ServerSocket connectionServer = null;
        Socket clientSession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            connectionServer = new ServerSocket(4444);
            clientSession = connectionServer.accept();
            out = new PrintWriter(clientSession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
            String word, mot;

            while ((word = in.readLine()) != null) {
                mot = (String) dico.get(word);
                if (mot == null) {
                    mot = "sorry, no translation available for \"" + word + "\" !";
                }
                out.println(mot);
            }
            out.close();  in.close();  connectionServer.close();  clientSession.close();
        } catch (IOException e) {
            System.out.println(e);
            System.exit(1);
        }
    }
}
```

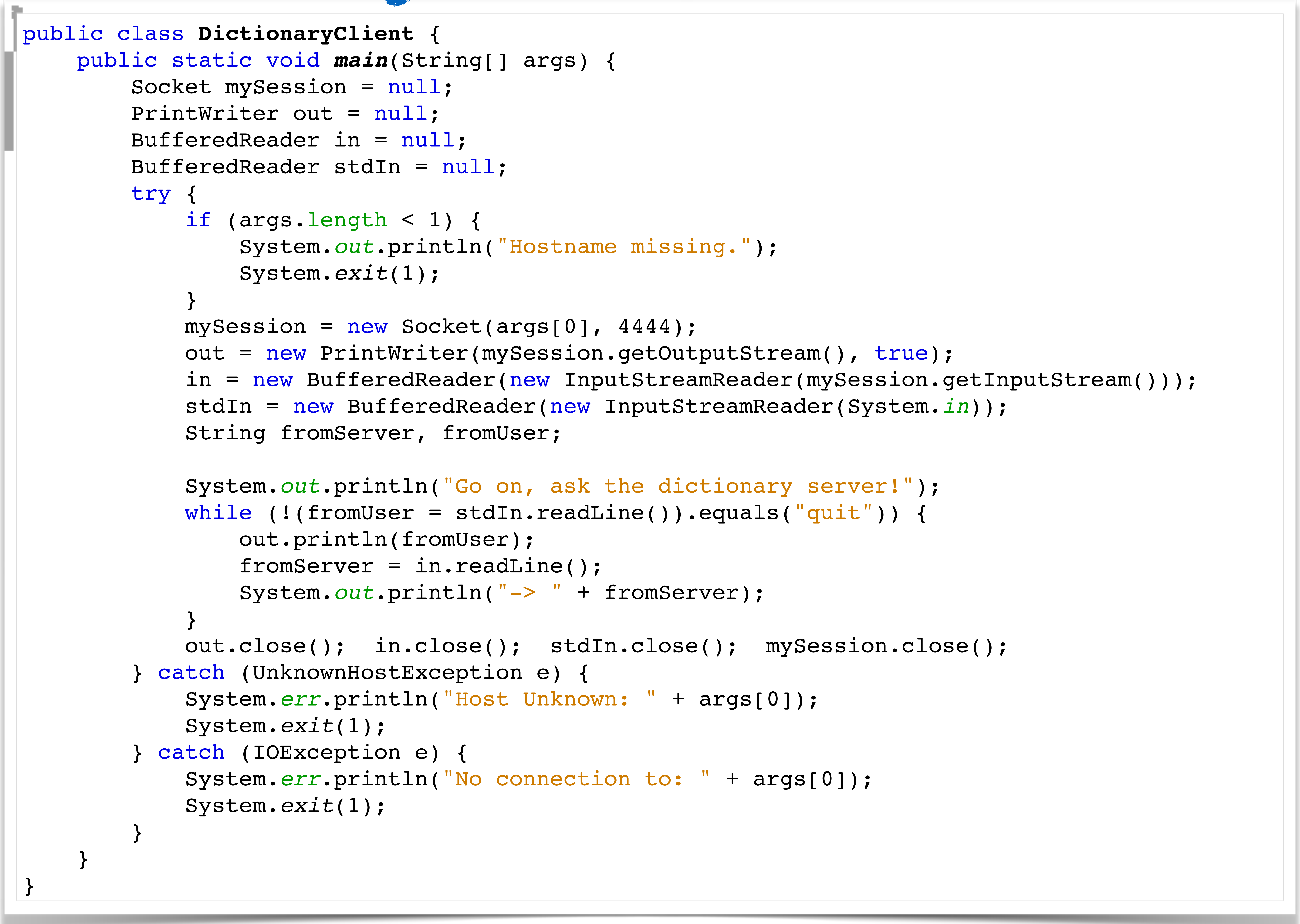
asynchronous messaging

using sockets

tcp

sockets

client



```
public class DictionaryClient {
    public static void main(String[] args) {
        Socket mySession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader stdIn = null;
        try {
            if (args.length < 1) {
                System.out.println("Hostname missing.");
                System.exit(1);
            }
            mySession = new Socket(args[0], 4444);
            out = new PrintWriter(mySession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(mySession.getInputStream()));
            stdIn = new BufferedReader(new InputStreamReader(System.in));
            String fromServer, fromUser;

            System.out.println("Go on, ask the dictionary server!");
            while (!(fromUser = stdIn.readLine()).equals("quit")) {
                out.println(fromUser);
                fromServer = in.readLine();
                System.out.println("-> " + fromServer);
            }
            out.close();  in.close();  stdIn.close();  mySession.close();
        } catch (UnknownHostException e) {
            System.err.println("Host Unknown: " + args[0]);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("No connection to: " + args[0]);
            System.exit(1);
        }
    }
}
```

asynchronous messaging

using sockets

the concept of streams

(unix and java)

streams offer a unified programming abstraction for reading and writing data

streams can encapsulate various types of data sources, e.g., files, byte arrays in memory, sockets, etc.

streams can encapsulate other streams to stack up processing of the data

in java, streams are instances of various classes found in the `java.io` package

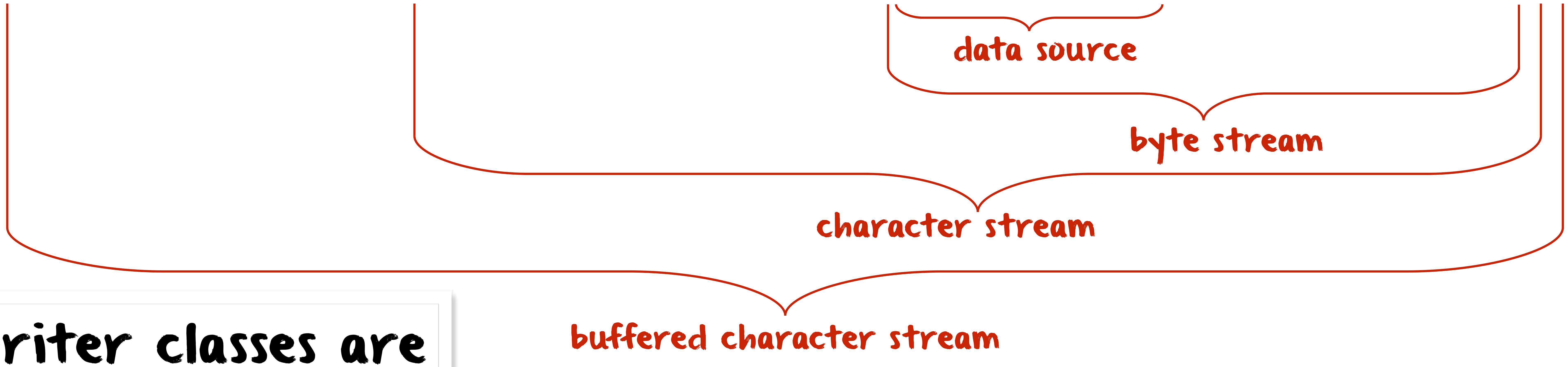
asynchronous messaging

using sockets

the concept of streams

(unix and java)

```
Socket clientSession= connectionServer.accept();
BufferedReader in= new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
```



data source

byte stream

character stream

buffered character stream

printer and writer classes are special streams manipulating only characters

standard operating systems-level input and output streams are also accessed via java streams (System.in & System.out)

asynchronous messaging

using sockets

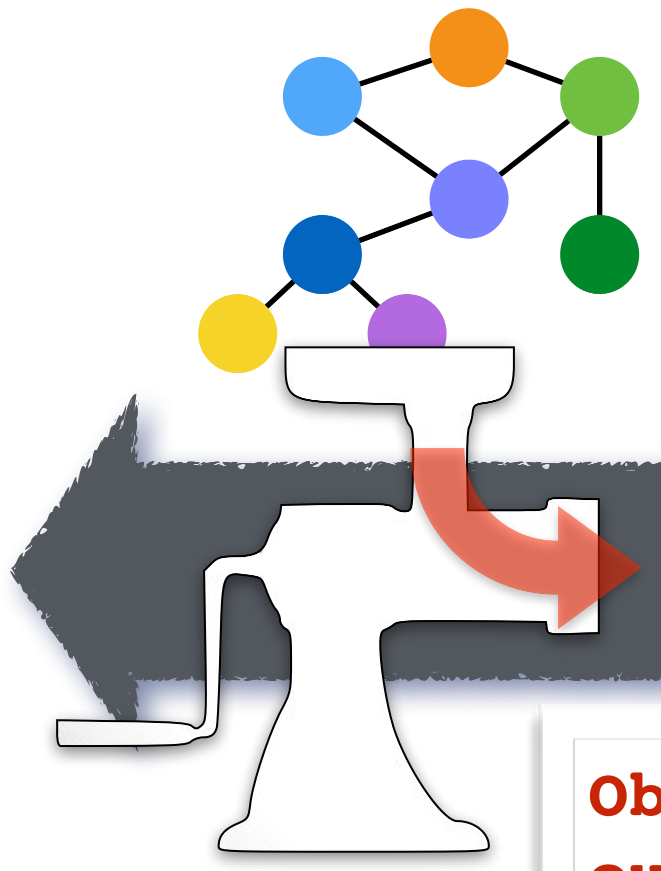
the concept of object streams

fact — the network knows nothing about objects, only bytes

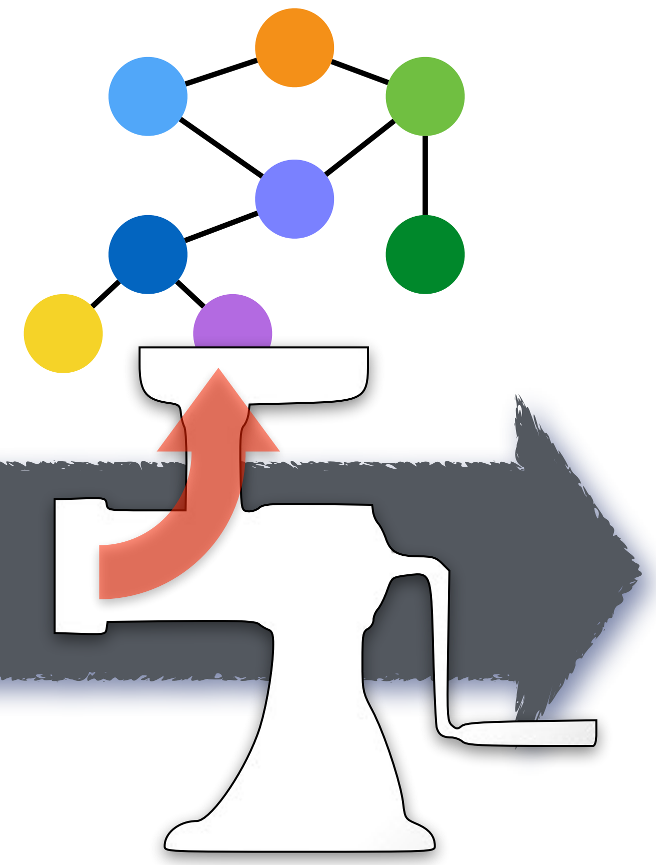
problem — so how can we send an object graph across the network?

solution — any java object can be encoded into a stream of bytes and recreated from it by implementing the `java.io.Serializable` interface

this process is known as object serialization



1011011101011001010101010110010111001001101



sender

receiver

```
ObjectOutputStream out = new ObjectOutputStream(sessionWithServer.getOutputStream());
out.writeObject(senderCollection);
```

```
ObjectInputStream in = new ObjectInputStream(sessionWithClient.getInputStream());
Collection receiverCollection = (Collection) in.readObject();
```

asynchronous messaging

using sockets

udp sockets

since udp is connectionless, we have only one class for udp sockets in java

```
public class DatagramSocket {
    ⋮
    public DatagramSocket() {…}
    public DatagramSocket(int port) {…}
    public void send(DatagramPacket packet) {…}
    public void receive(DatagramPacket packet) {…}
    public void close() {…}
    ⋮
}
```

```
public class DatagramPacket {
    ⋮
    public DatagramPacket(…) {…}
    public byte[] getData() {…}
    public InetAddress getAddress() {…}
    ⋮
}
```

the DatagramPacket class is key when working with udp sockets

it captures the message-oriented nature of udp sockets

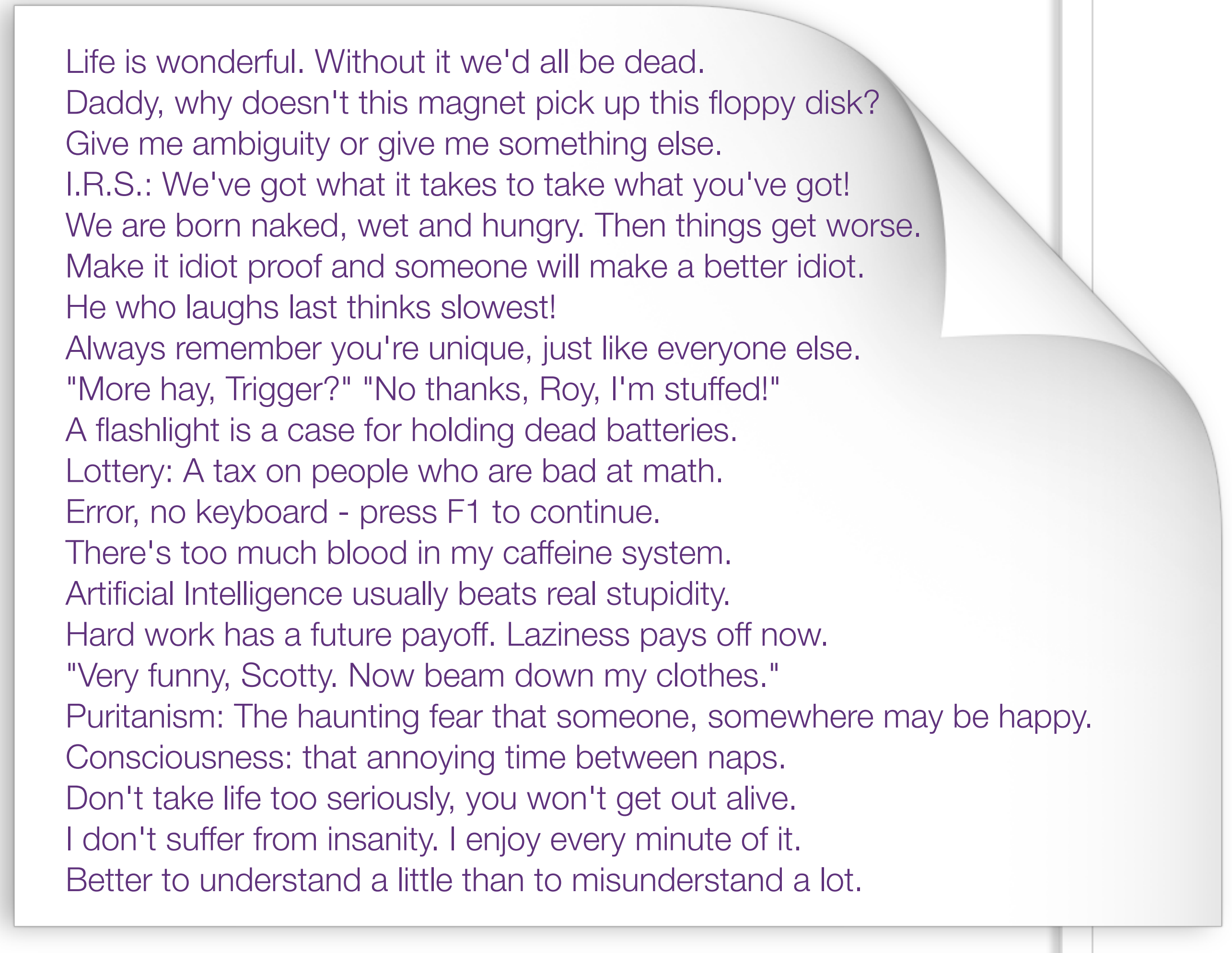
asynchronous messaging

using sockets

udp sockets

```
public class QuoteServer {
   public static void main(String[] args) throws Exception {
        DatagramSocket socket = null;
        BufferedReader in = null;
        socket = new DatagramSocket(4445);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;

        while (moreQuotes) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else { buf = quote.getBytes(); }
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
        }
        socket.close();
    }
}
```

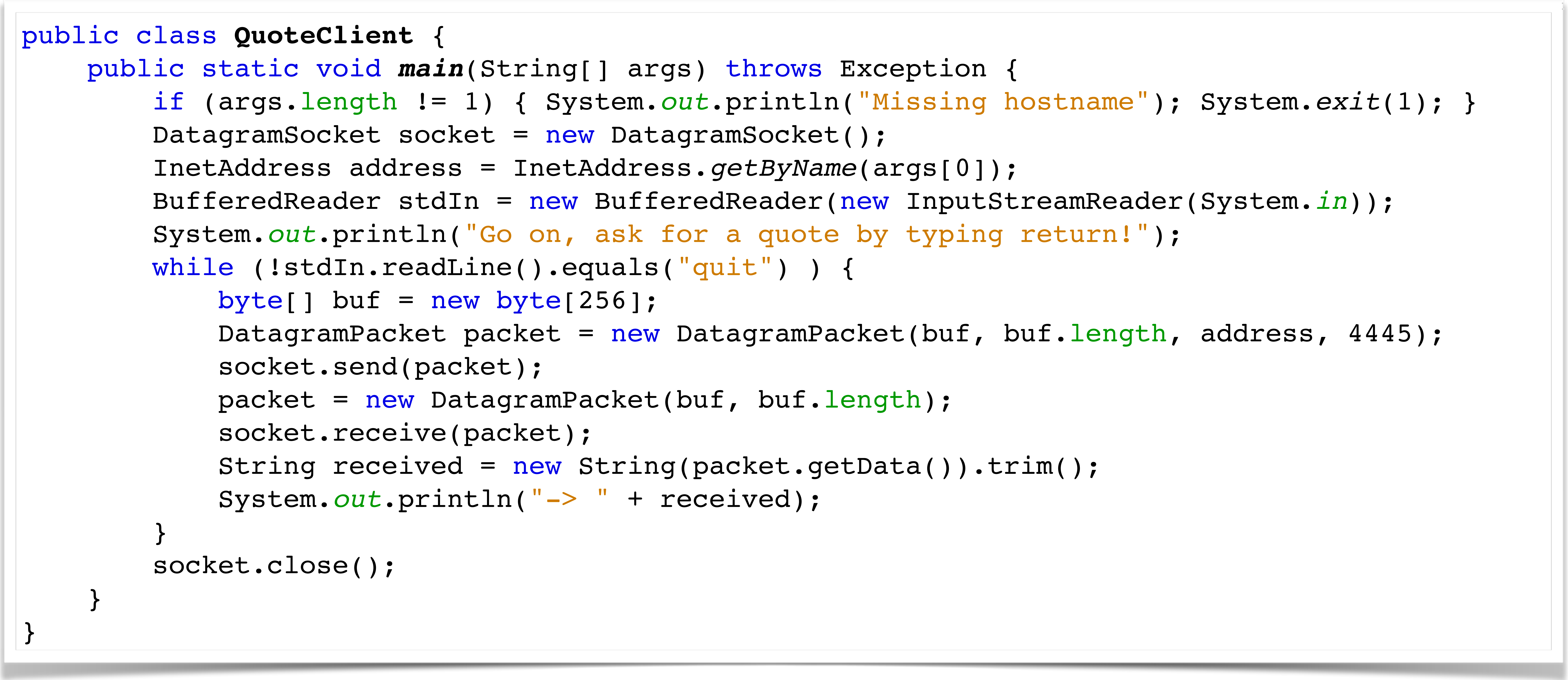


```
Life is wonderful. Without it we'd all be dead.
Daddy, why doesn't this magnet pick up this floppy disk?
Give me ambiguity or give me something else.
I.R.S.: We've got what it takes to take what you've got!
We are born naked, wet and hungry. Then things get worse.
Make it idiot proof and someone will make a better idiot.
He who laughs last thinks slowest!
Always remember you're unique, just like everyone else.
"More hay, Trigger?" "No thanks, Roy, I'm stuffed!"
A flashlight is a case for holding dead batteries.
Lottery: A tax on people who are bad at math.
Error, no keyboard - press F1 to continue.
There's too much blood in my caffeine system.
Artificial Intelligence usually beats real stupidity.
Hard work has a future payoff. Laziness pays off now.
"Very funny, Scotty. Now beam down my clothes."
Puritanism: The haunting fear that someone, somewhere may be happy.
Consciousness: that annoying time between naps.
Don't take life too seriously, you won't get out alive.
I don't suffer from insanity. I enjoy every minute of it.
Better to understand a little than to misunderstand a lot.
```

asynchronous messaging

using sockets

udp sockets



```
public class QuoteClient {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) { System.out.println("Missing hostname"); System.exit(1); }
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName(args[0]);
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Go on, ask for a quote by typing return!");
        while (!stdIn.readLine().equals("quit") ) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
            socket.send(packet);
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData()).trim();
            System.out.println("-> " + received);
        }
        socket.close();
    }
}
```

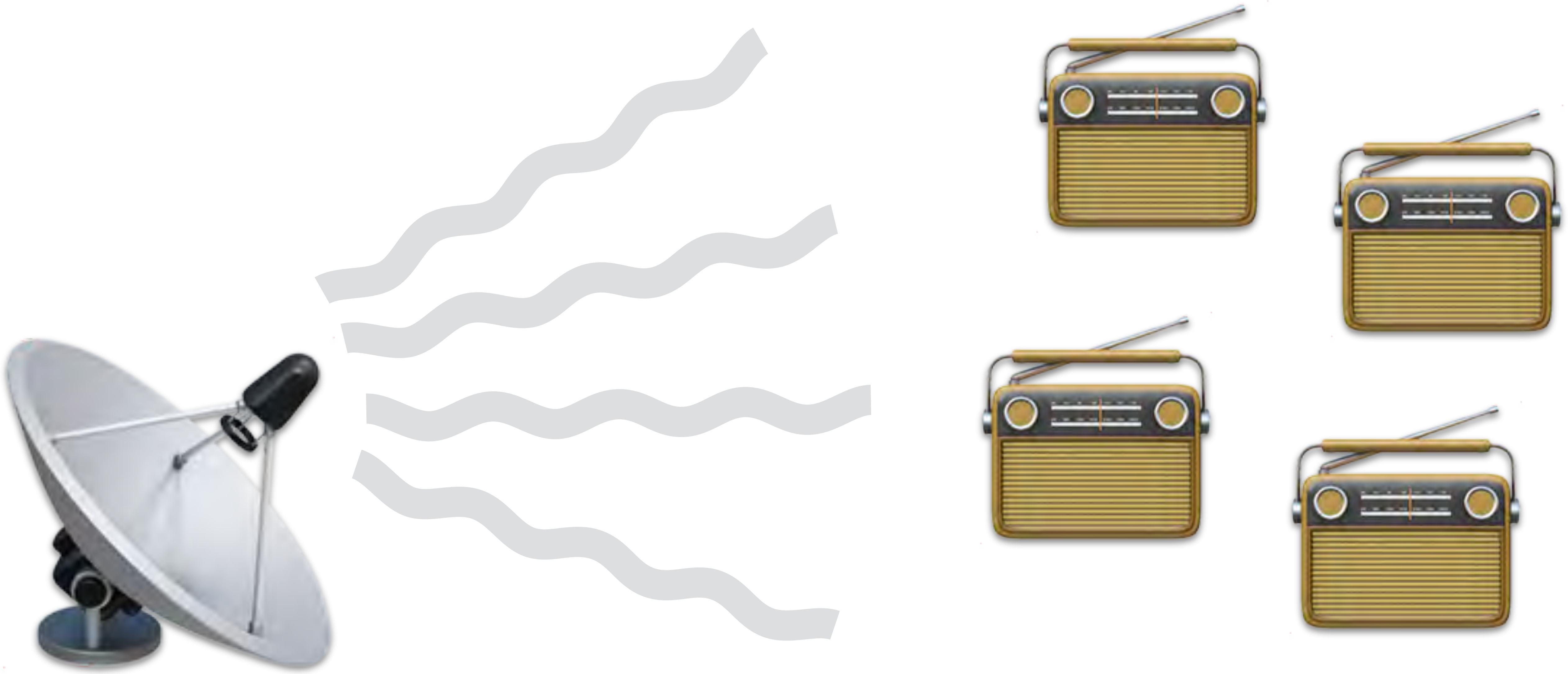
asynchronous messaging

using sockets

user
datagram
protocol

one-to-one communication

one-to-many communication



udp multicast

a multicast address lies in range [224.0.0.0 , 239.255.255.255] and defines a multicast group

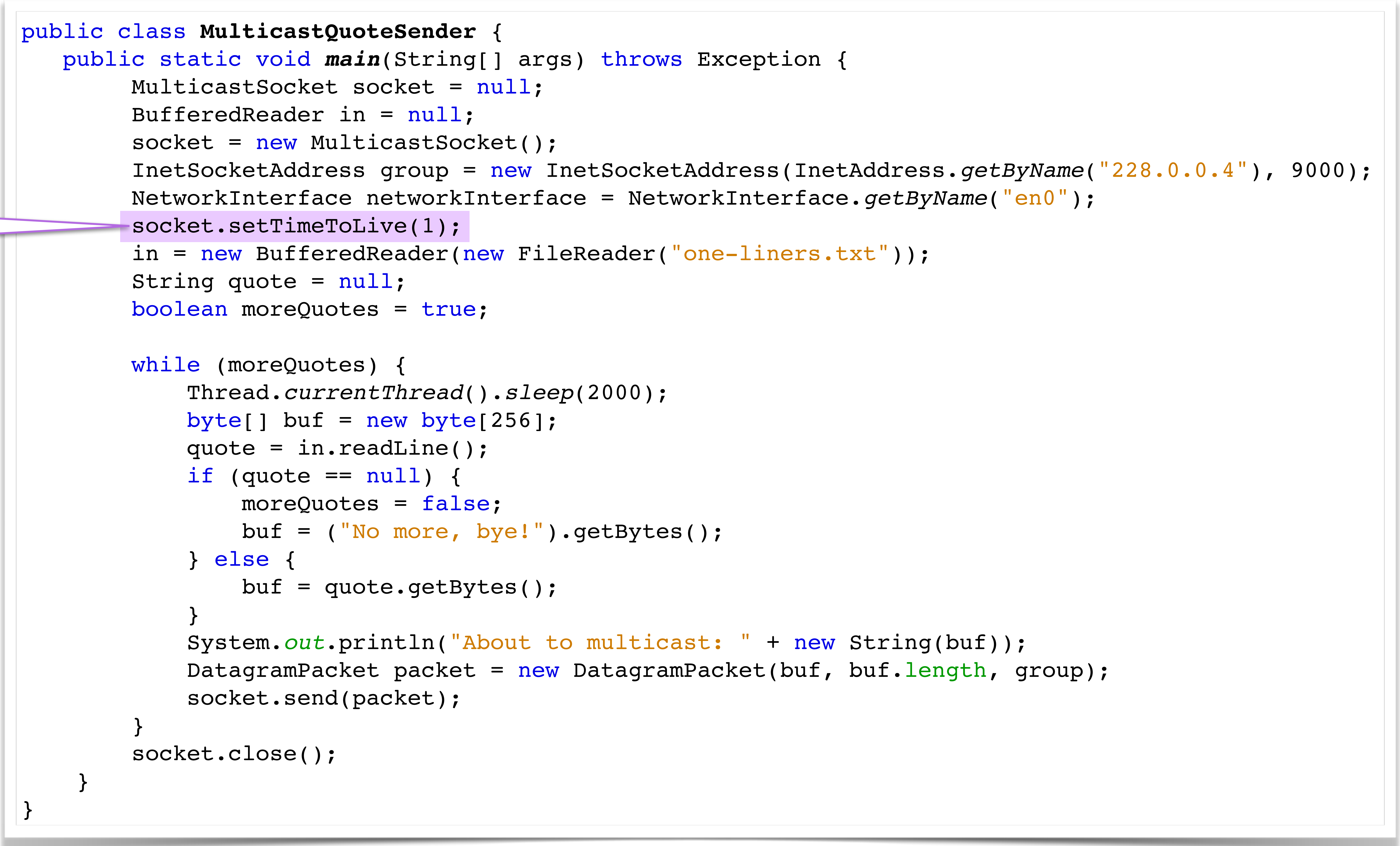
in java, udp multicast is accessed via `MulticastSocket`, a subclass of `DatagramSocket`

asynchronous messaging

using sockets

udp multicast

maximum number of routers a multicast packet can go through before being deleted



```java
public class MulticastQuoteSender {
   public static void main(String[] args) throws Exception {
        MulticastSocket socket = null;
        BufferedReader in = null;
        socket = new MulticastSocket();
        InetSocketAddress group = new InetSocketAddress(InetAddress.getByName("228.0.0.4"), 9000);
        NetworkInterface networkInterface = NetworkInterface.getByName("en0");
        socket.setTimeToLive(1);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;

        while (moreQuotes) {
            Thread.currentThread().sleep(2000);
            byte[] buf = new byte[256];
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else {
                buf = quote.getBytes();
            }
            System.out.println("About to multicast: " + new String(buf));
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group);
            socket.send(packet);
        }
        socket.close();
    }
}
```

asynchronous messaging

using sockets

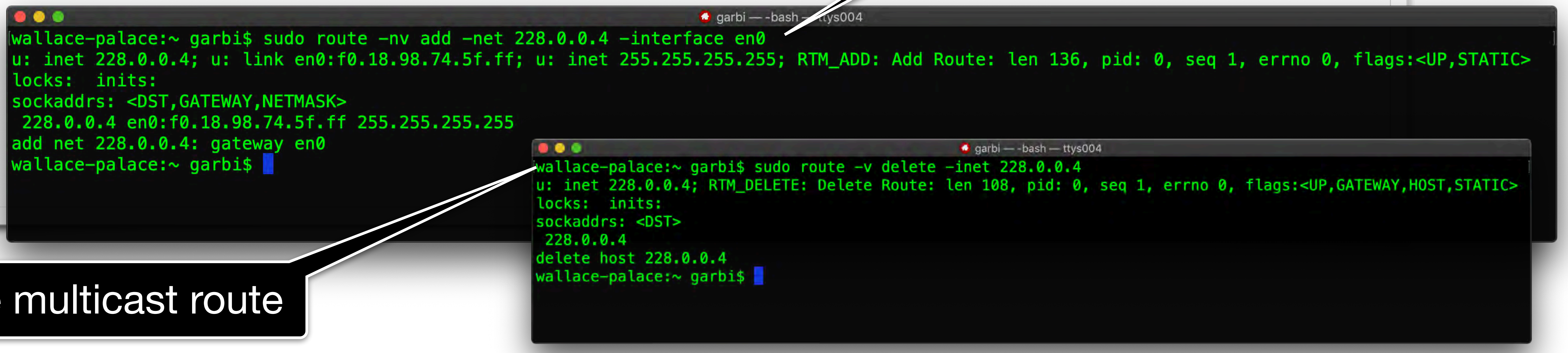
udp multicast

```java
public class MulticastQuoteReceiver {

    public static void main(String[] args) throws Exception {
        try (MulticastSocket socket = new MulticastSocket(9000)) {
            InetSocketAddress group = new InetSocketAddress(InetAddress.getByName("228.0.0.4"), 9000);
            NetworkInterface netInterface = NetworkInterface.getByName("en0");
            socket.joinGroup(group, netInterface);
            while (true) {
                byte[] buf = new byte[256];
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                System.out.print("Waiting for the next quote: ");
                socket.receive(packet);
                String received = new String(packet.getData());
                System.out.println(received.trim());
                if (received.contains("bye")) {
                    break;
                }
            }
            socket.leaveGroup(group, netInterface);
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

tuning in

tuning out



creating a multicast route

```
wallace-palace:~ garbi$ sudo route -nv add -net 228.0.0.4 -interface en0
u: inet 228.0.0.4; u: link en0:f0.18.98.74.5f.ff; u: inet 255.255.255.255; RTM_ADD: Add Route: len 136, pid: 0, seq 1, errno 0, flags:<UP,STATIC>
locks:  inits:
sockaddrs: <DST,GATEWAY,NETMASK>
 228.0.0.4 en0:f0.18.98.74.5f.ff 255.255.255.255
add net 228.0.0.4: gateway en0
wallace-palace:~ garbi$
```

deleting the multicast route

```
wallace-palace:~ garbi$ sudo route -v delete -inet 228.0.0.4
u: inet 228.0.0.4; RTM_DELETE: Delete Route: len 108, pid: 0, seq 1, errno 0, flags:<UP,GATEWAY,HOST,STATIC>
locks:  inits:
sockaddrs: <DST>
 228.0.0.4
delete host 228.0.0.4
wallace-palace:~ garbi$
```