SEG 2105 - LECTURE 03

Basing Software Development on Reusable Technology

BUILDING ON THE EXPERIENCE OF OTHERS

SOFTWARE ENGINEERS SHOULD AVOID RE-DEVELOPING SOFTWARE ALREADY DEVELOPED

Types of reuse

EXPERTISE	LIBRARIES	FRAMEWORKS
STANDARD DESIGN	COMMANDS	APPLICATIONS

FRAMEWORKS: REUSABLE SUBSYSTEMS

Web Frameworks...
they do different things, but related

Authentication Frameworks...
Applications are different, but all need
some way to authorize / authenticate its users

Frameworks to promote reuse

INCOMPLETE	HOOKS	SERVICES
SLOTS	EXTENSION POINTS	APIS

Convention over Configuration

One of the early productivity mottos of Rails went: “You’re not a beautiful and unique snowflake”. It postulated that by giving up vain individuality, you can leapfrog the toils of mundane decisions, and make faster progress in areas that really matter.

Who cares what format your database primary keys are described by? Does it really matter whether it's “id”, “postId”, “posts_id”, or “pid”? Is this a decision that’s worthy of recurrent deliberation? No.

Part of the Rails’ mission is to swing its machete at the thick, and ever growing, jungle of recurring decisions that face developers creating information systems for the web. There are thousands of such decisions that just need to be made once, and if someone else can do it for you, all the better.

https://rubyonrails.org/doctrine/#convention-over-configuration

```
CREATE TABLE employees (
    id int(11) NOT NULL auto_increment,
    email varchar(255),
    name varchar(255),
    first_name varchar(255),
    last_name varchar(255),
    UNIQUE KEY email,
    PRIMARY KEY (id)
);
```

```sql
CREATE TABLE employees (
   id int(11) NOT NULL auto_increment,
   email varchar(255),
   name varchar(255),
   first_name varchar(255),
   last_name varchar(255),
   UNIQUE KEY email,
   PRIMARY KEY (id)
);
```

CONFIGURATION

```java
@Entity
@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL)
@Table(name = "employees", uniqueConstraints = {
        @UniqueConstraint(columnNames = "id"),
        @UniqueConstraint(columnNames = "email") })
public class EmployeeEntity implements Serializable {

    private static final long serialVersionUID = -1798070786993154676L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", unique = true, nullable = false)
    private Integer employeeId;

    @Column(name = "email", unique = true, nullable = false, length = 255)
    private String email;

    @Column(name = "first_name", unique = false, nullable = false, length = 255)
    private String firstName;

    @Column(name = "last_name", unique = false, nullable = false, length = 255)
    private String lastName;

    // Accessors and mutators for all four fields
}
```

```
CREATE TABLE employees (
   id int(11) NOT NULL auto_increment,
   email varchar(255),
   name varchar(255),
   first_name varchar(255),
   last_name varchar(255),
   UNIQUE KEY email,
   PRIMARY KEY (id)
);
```

```
class Employee < ApplicationRecord
  validates :email, uniqueness: true
end
```

CONVENTION

Object-oriented frameworks

A framework is composed of a library of classes.

The **API** is defined by the set of **all public** methods of these classes.

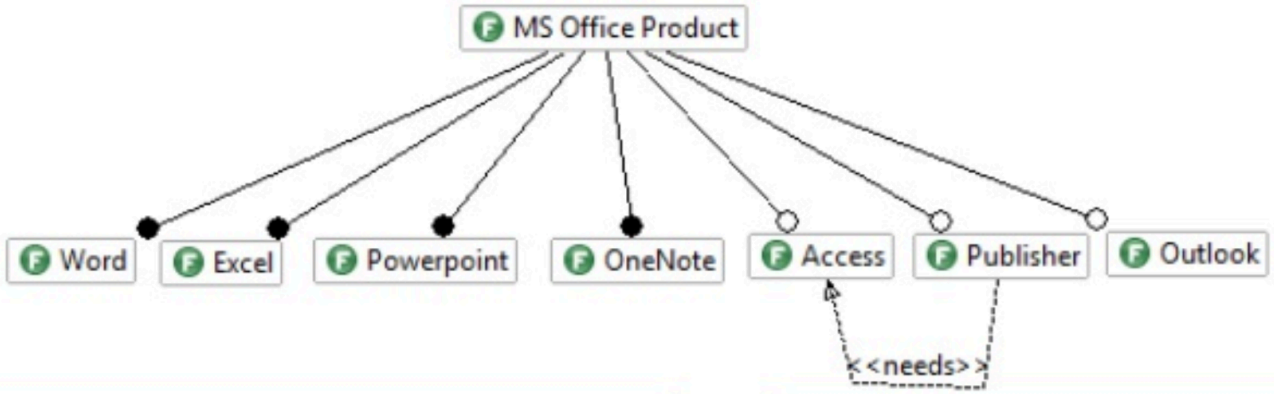
Some of the classes will normally be **abstract** and there are often many **interfaces**

PRODUCT LINES

MS Office

	Word	Excel	PowerPoint	OneNote	Outlook	Publisher	Access
Office Professional 2010	●	●	●	●	●	●	●
Office Home and Business 2010	●	●	●	●	●		
Office Home and Student 2010	●	●	●	●			

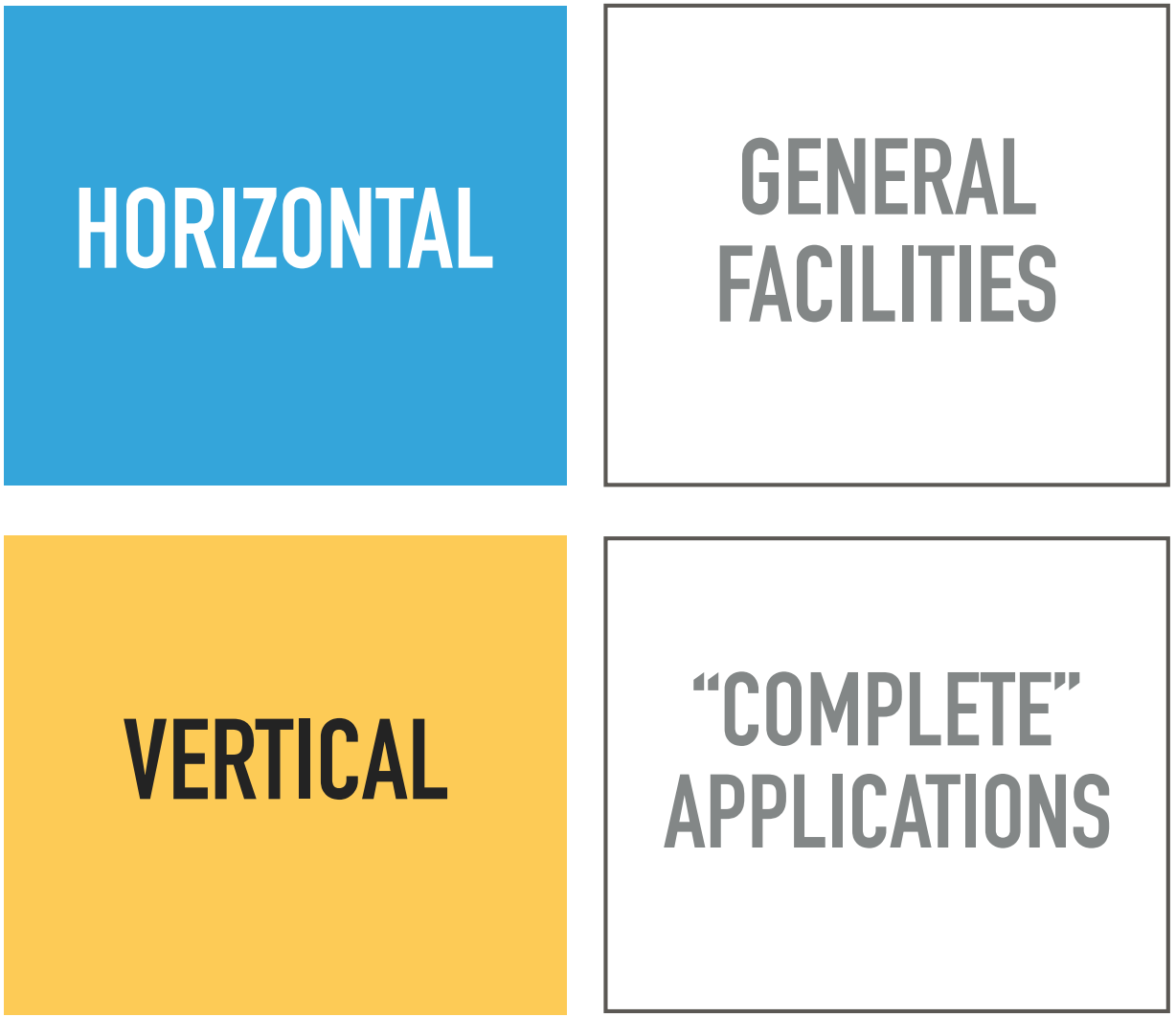
https://www.slideshare.net/jaime.chavarriaga/supporting-architectural-variabiality-in-software-product-lines

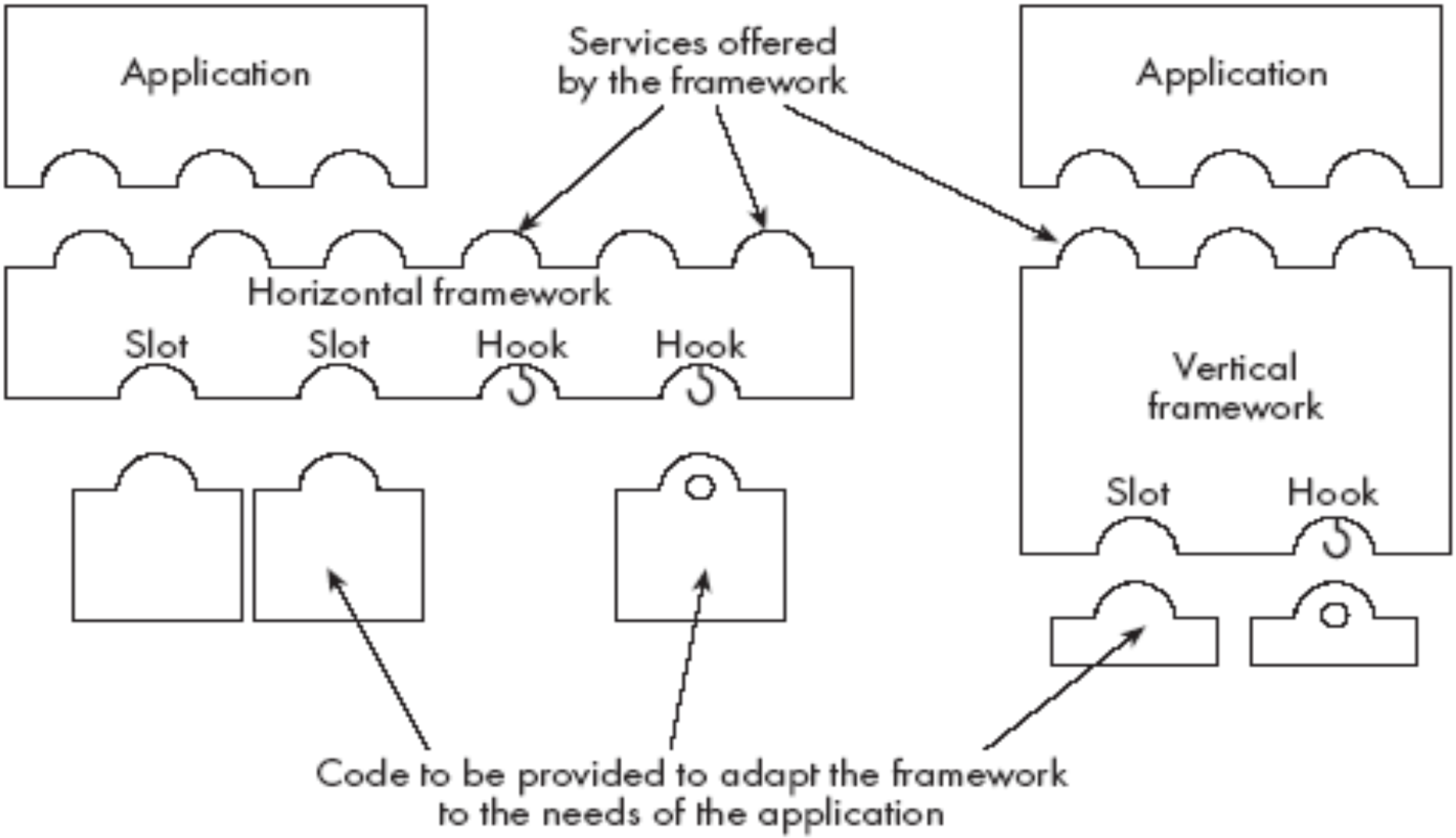


- C_1 = { Word, Excel, Powerpoint, OneNote }
- C_2 = { Word, Excel, Powerpoint, OneNote, Access}
- C_3 = { Word, Excel, Powerpoint, OneNote, Access, Publisher}
- C_4 = { Word, Excel, Powerpoint, OneNote, Outlook}
- C_5 = { Word, Excel, Powerpoint, OneNote, Access, Outlook}
- C_6 = { Word, Excel, Powerpoint, OneNote, Access, Publisher, Outlook}

Commonalities **Variabilities**

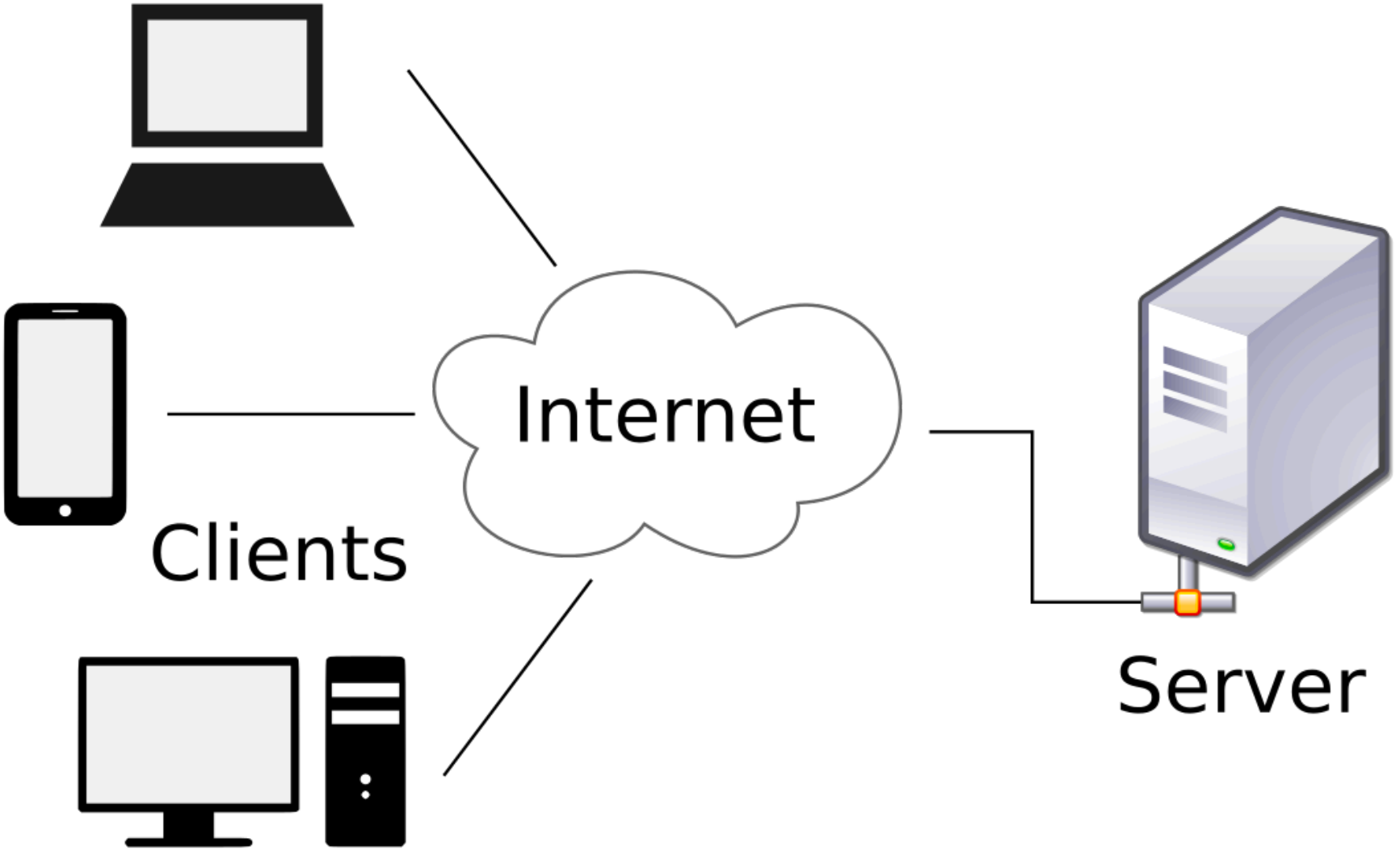
Types of frameworks



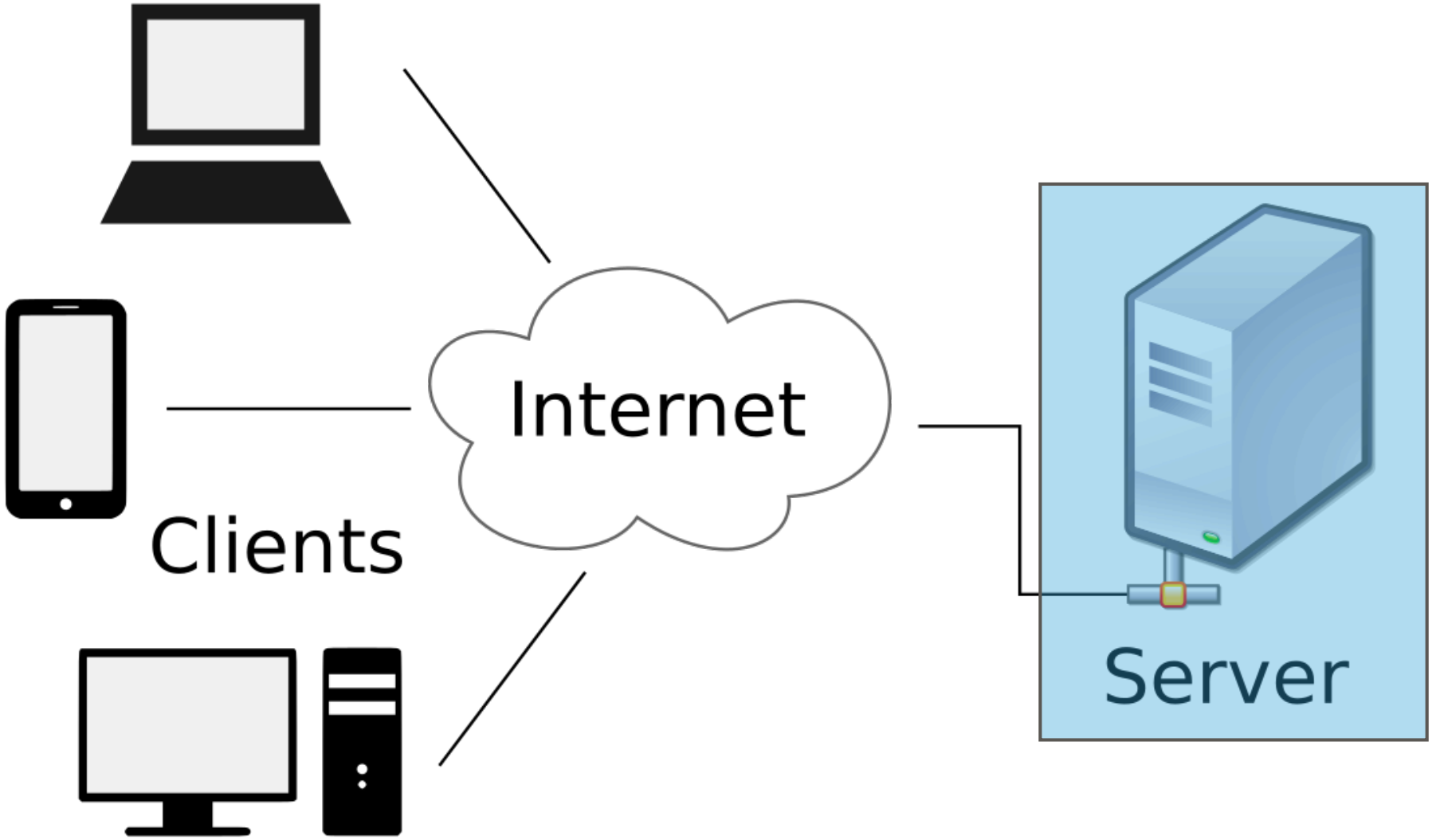

Application
Services offered
by the framework
Application
Horizontal framework
Slot
Slot
Hook
Hook
Vertical
framework
Slot
Hook
Code to be provided to adapt the framework
to the needs of the application

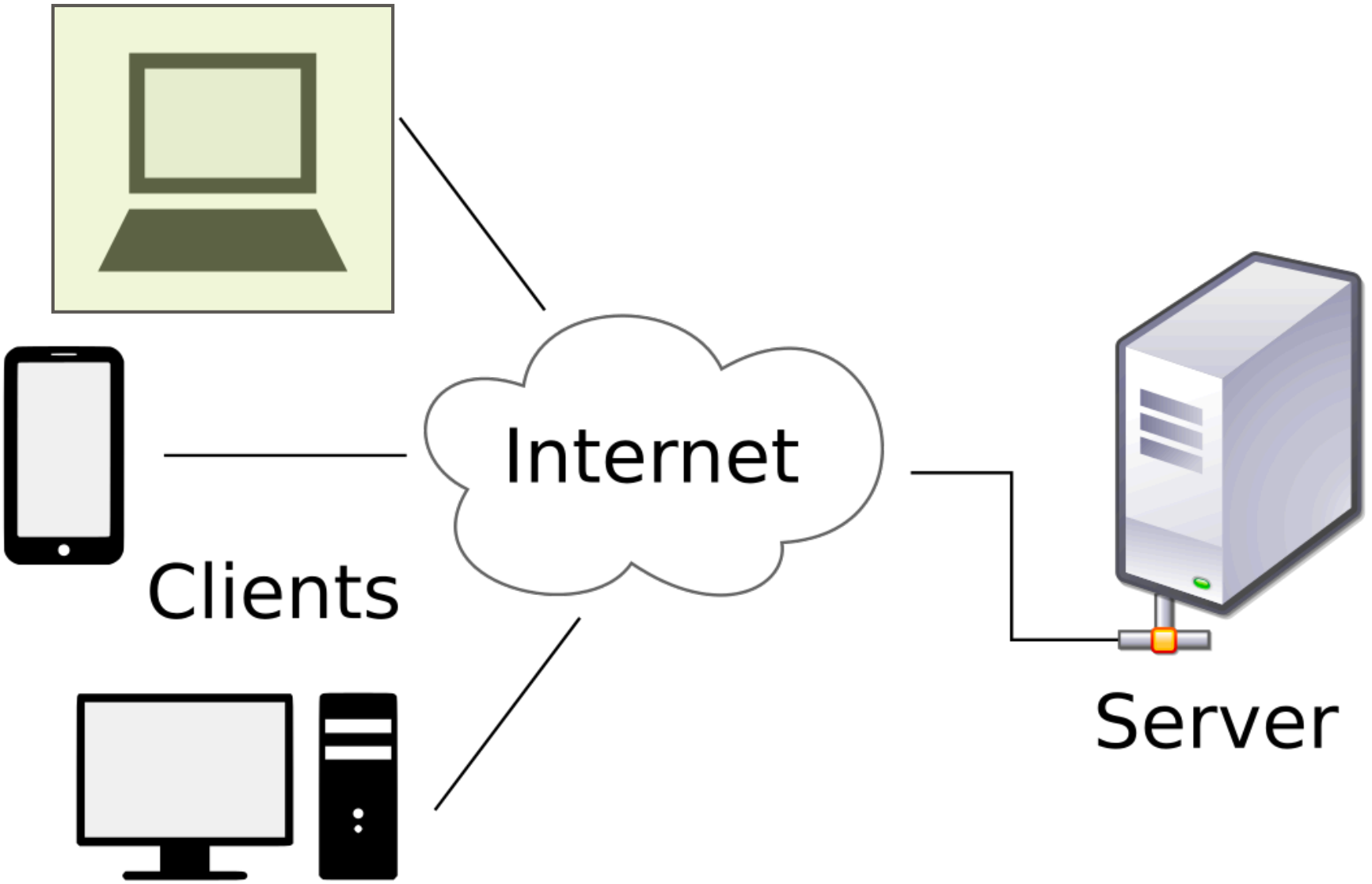
THE CLIENT-SERVER

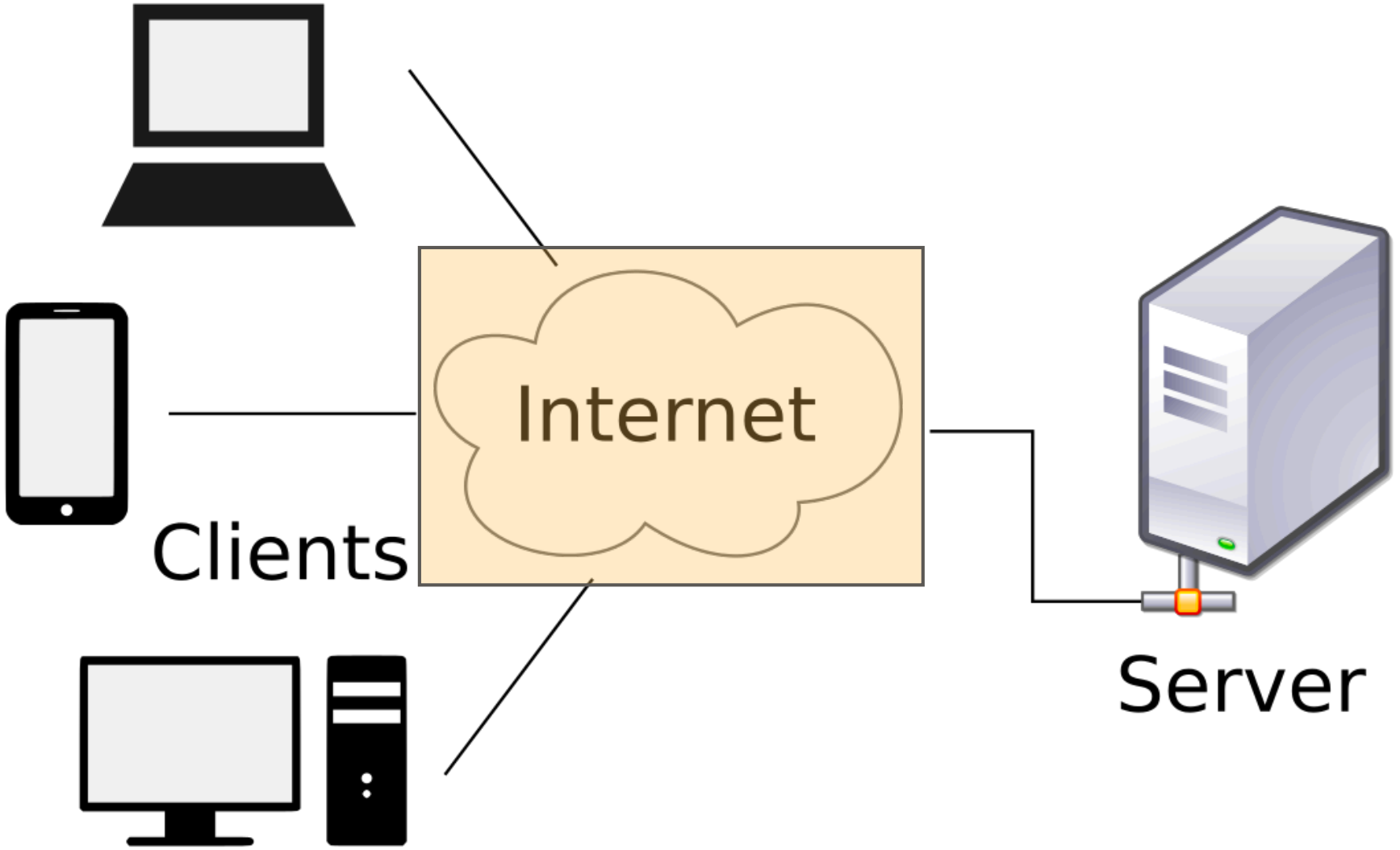
ARCHITECTURE



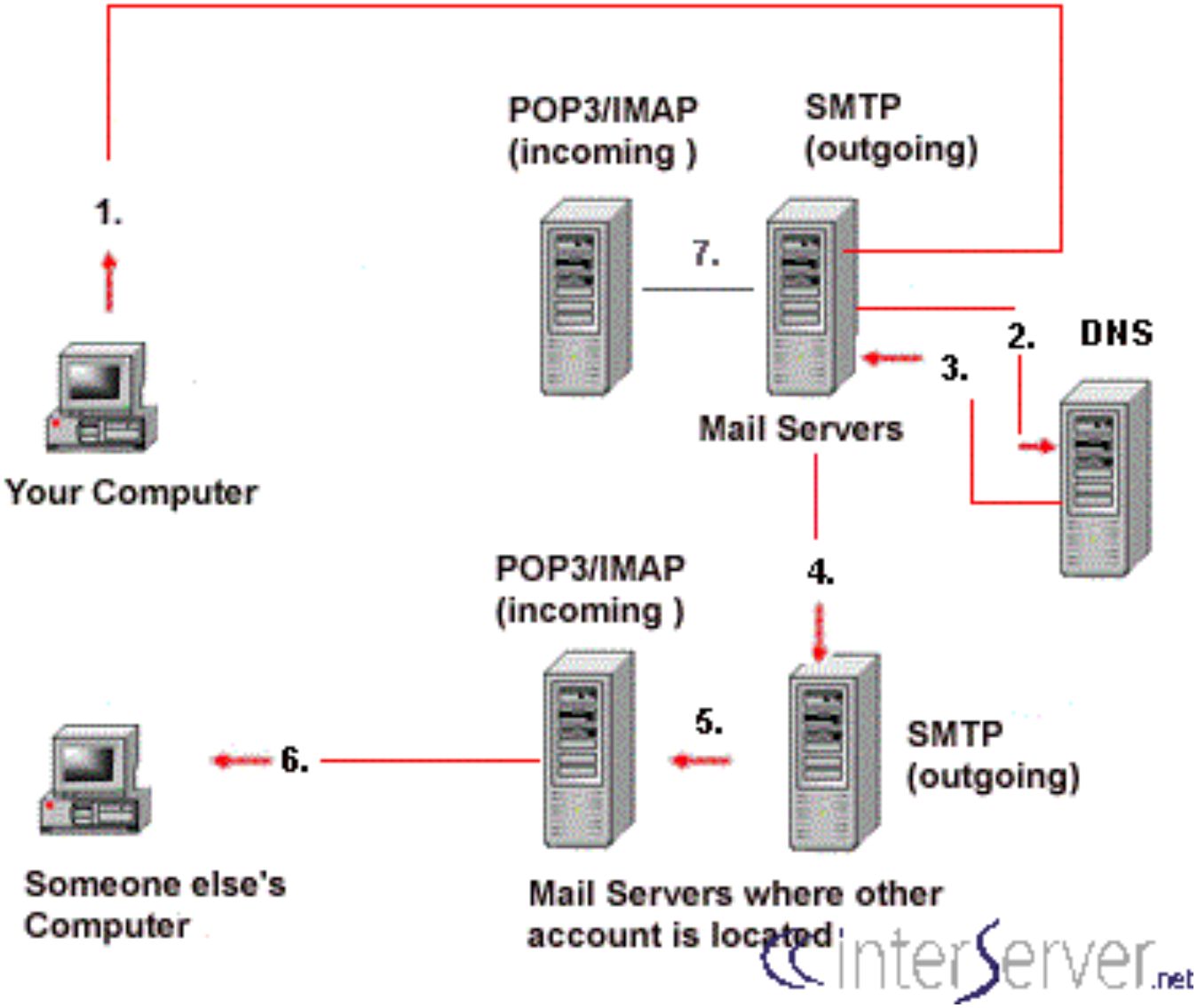
https://en.wikipedia.org/wiki/Client%E2%80%93server_model


Internet
Clients
Server


Internet
Clients
Server

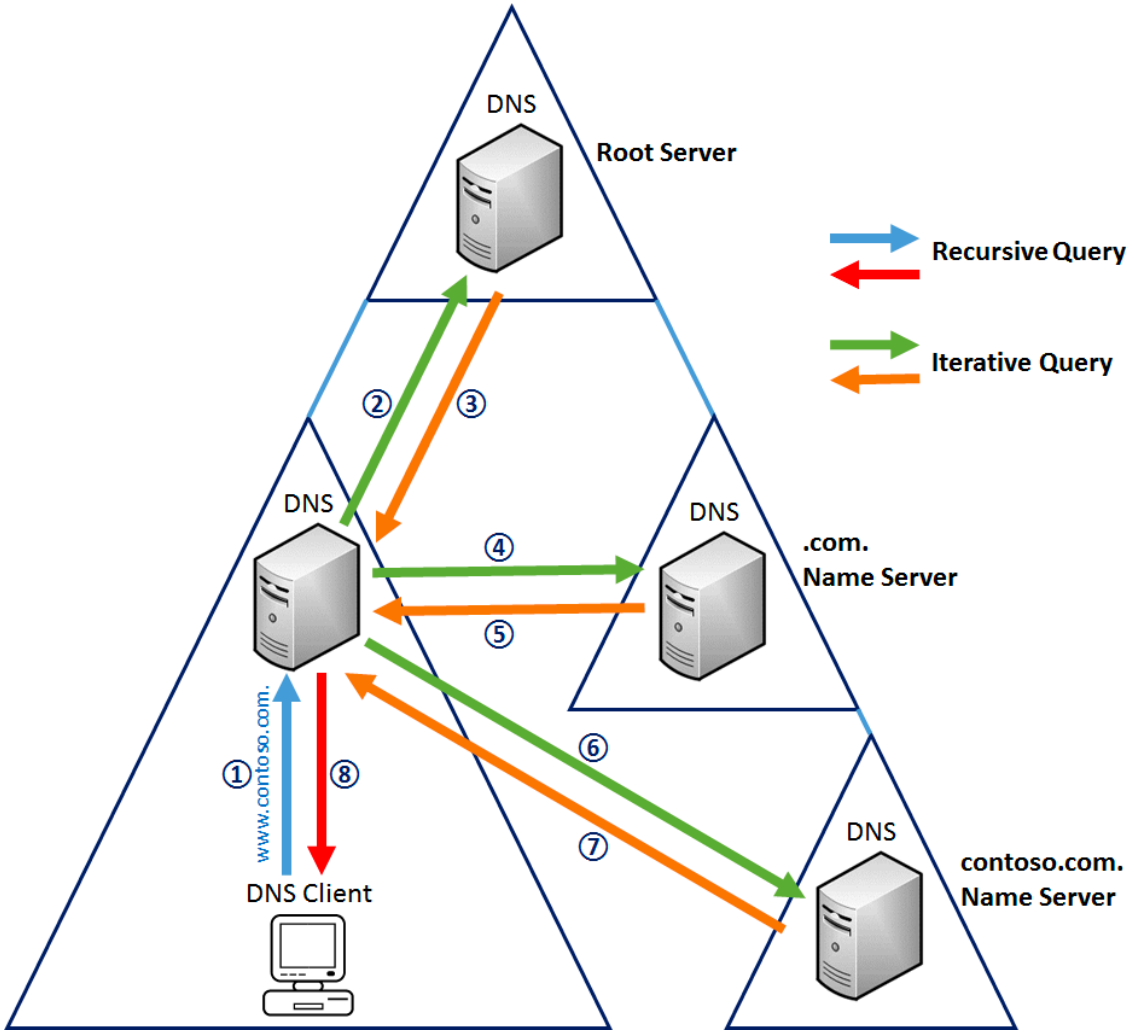

Internet
Clients
Server

EMAIL



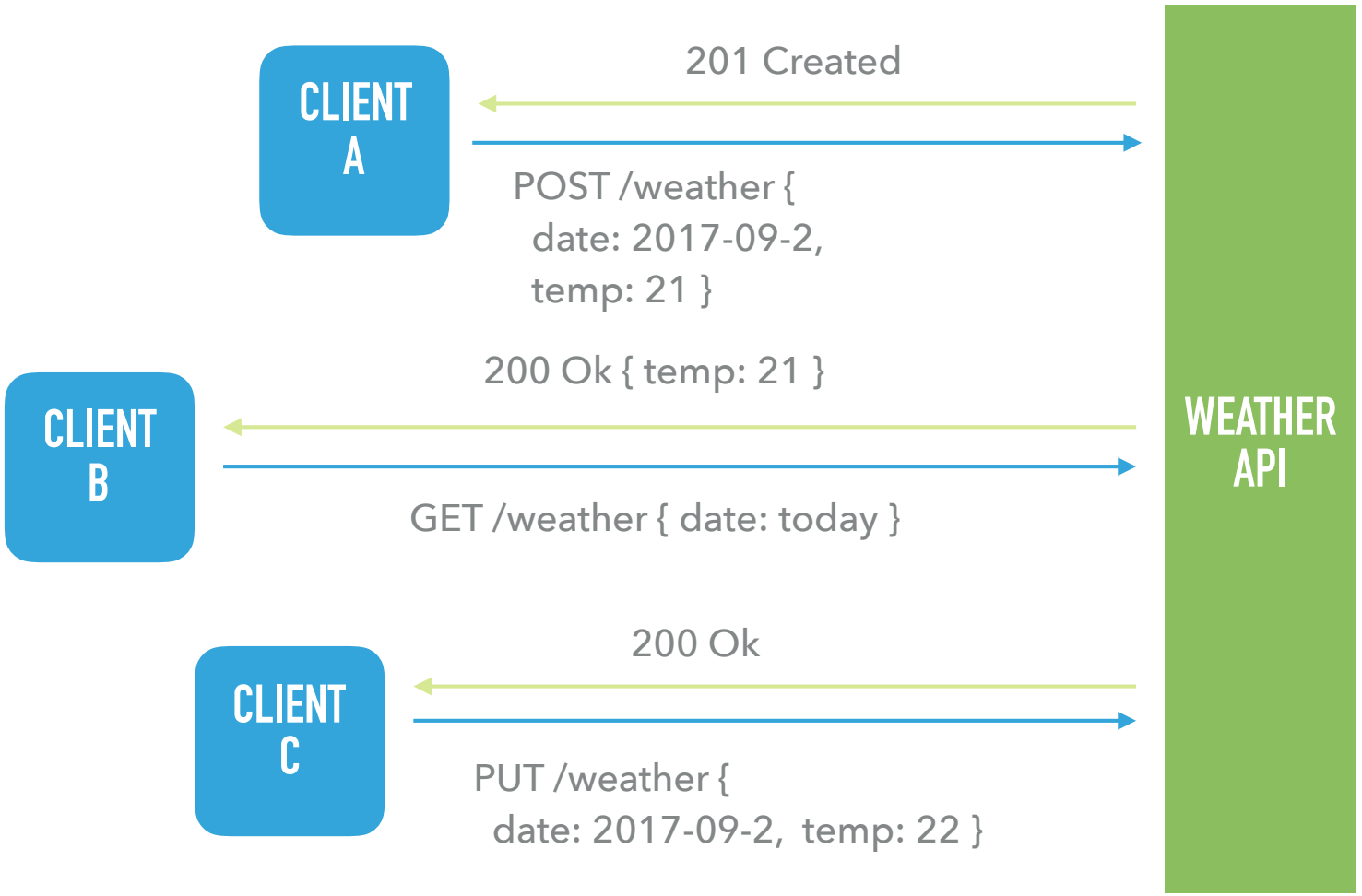
https://www.interserver.net/tips/kb/exactly-emails-works-steps-explanation/

DOMAIN NAME SYSTEM (DNS)



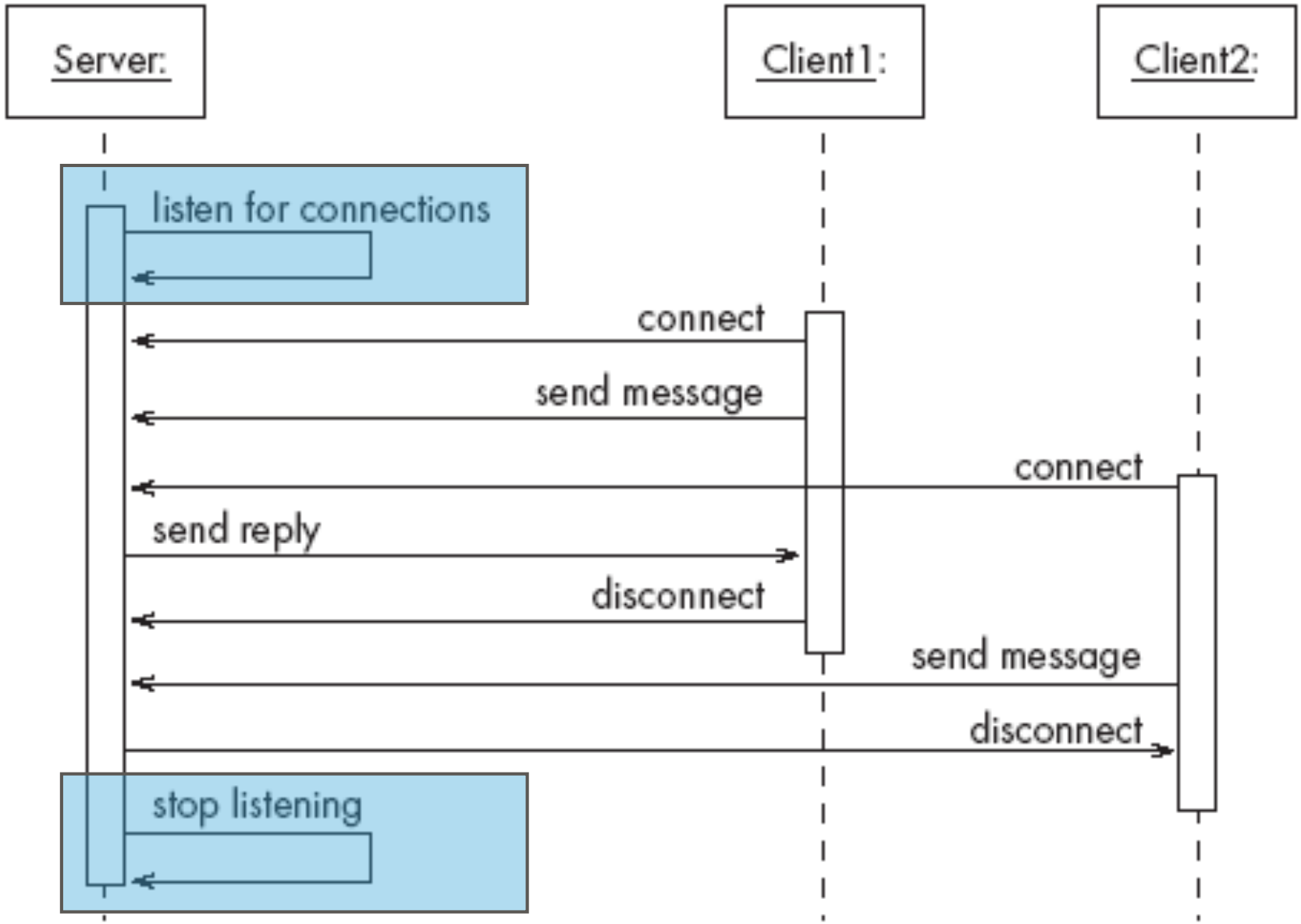
https://gitlearning.wordpress.com/2015/06/23/dns-server/

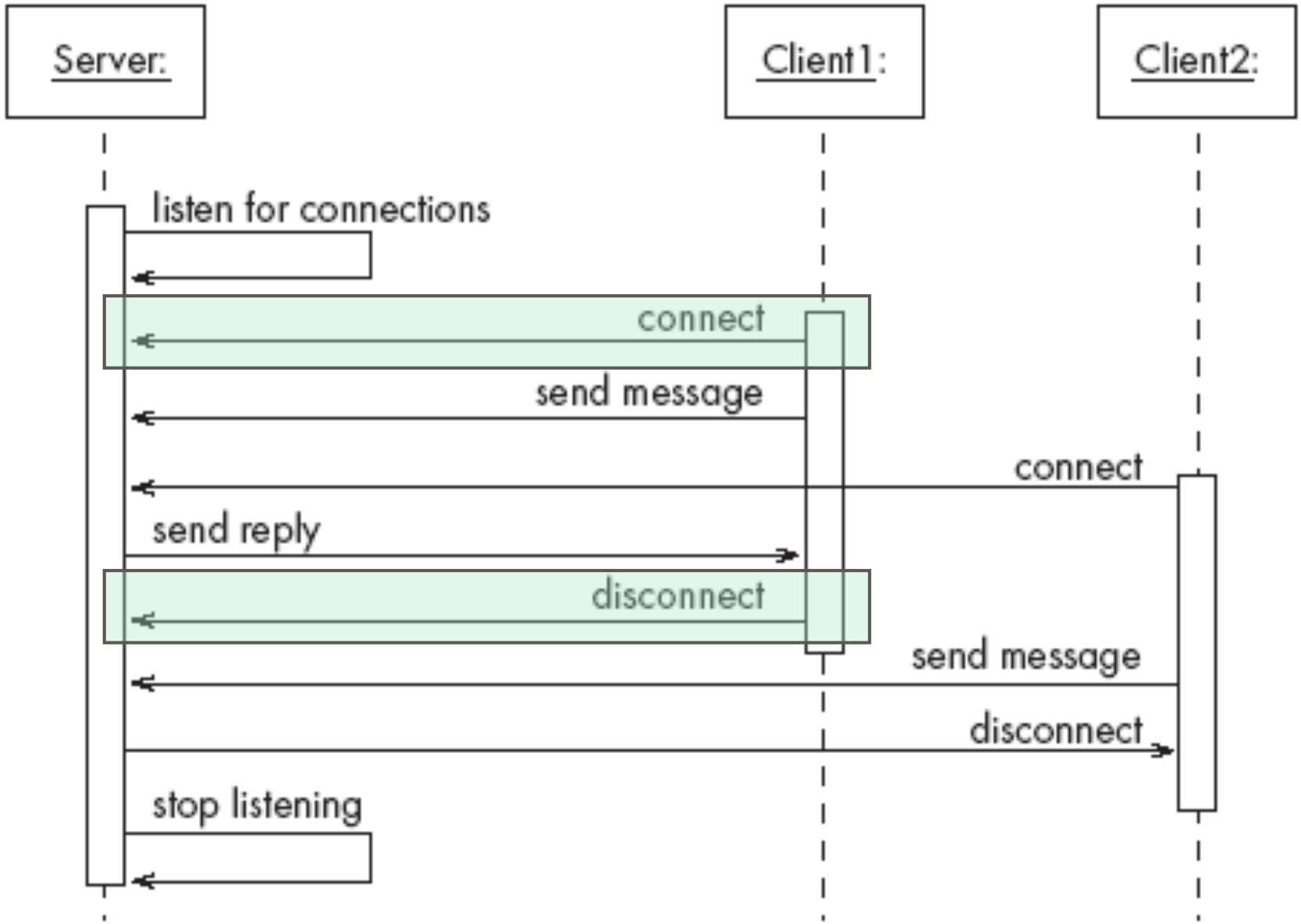
REST (AND GRAPHQL) APIS ARE CLIENT SERVER

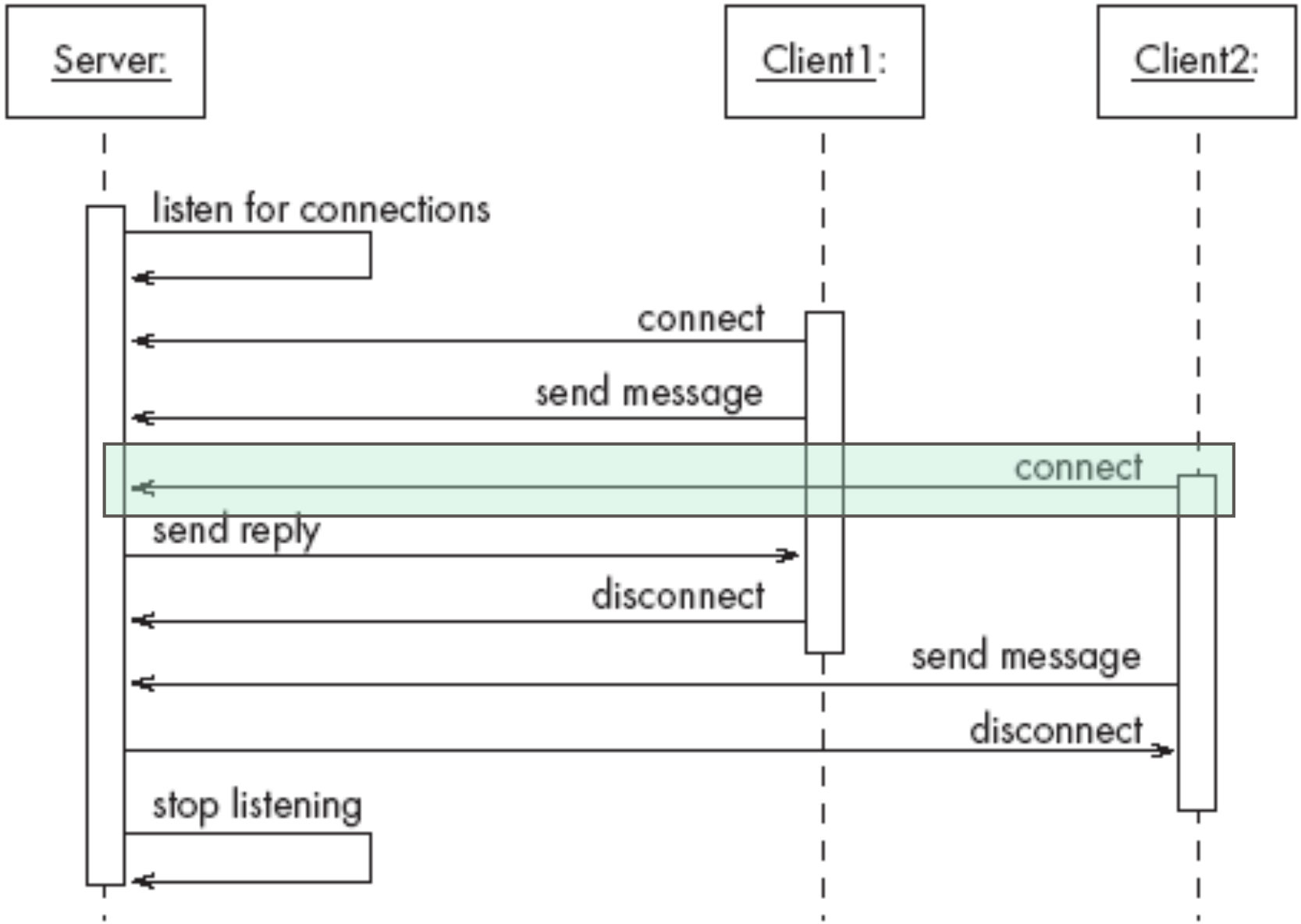


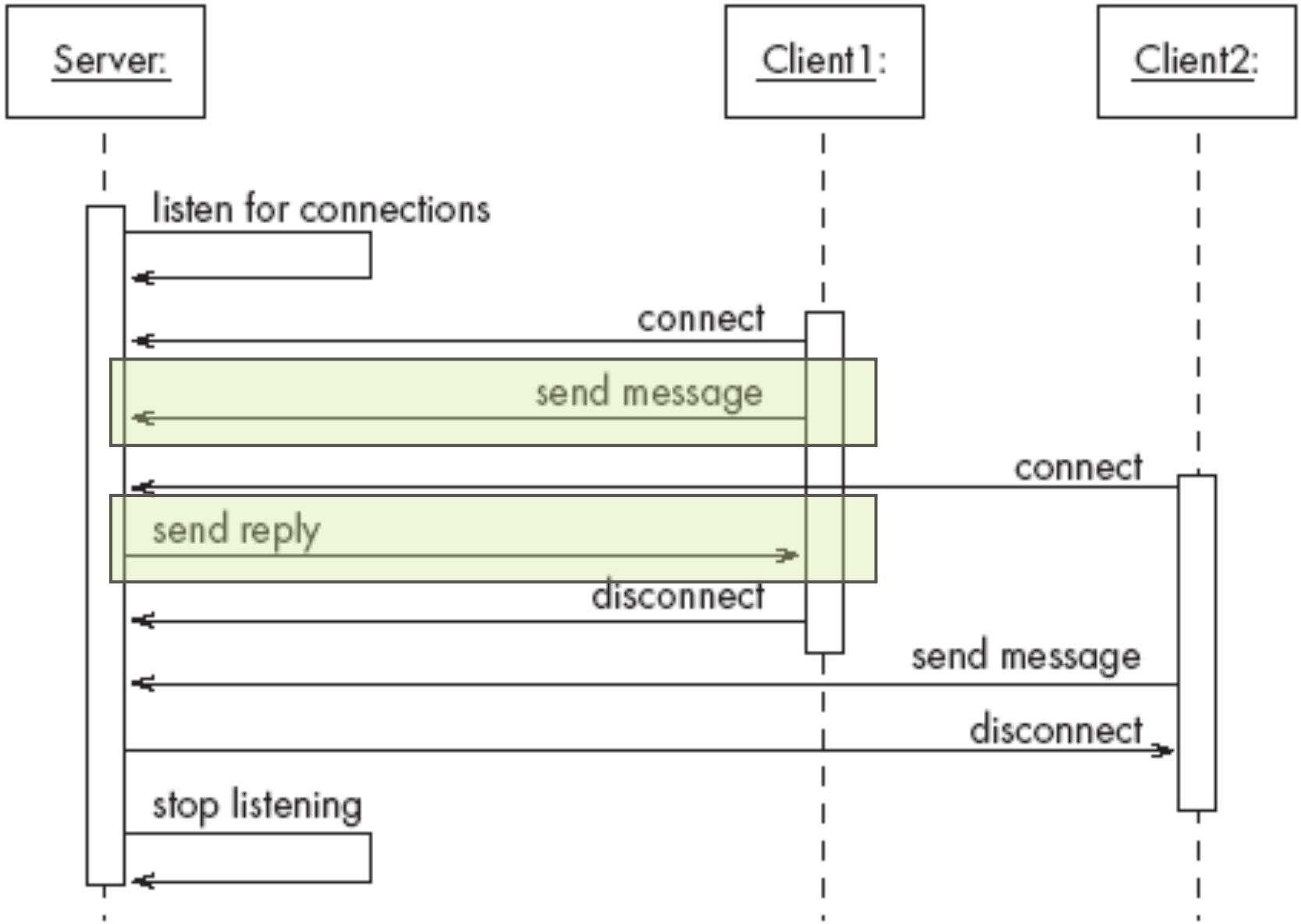
Examples of client-server systems

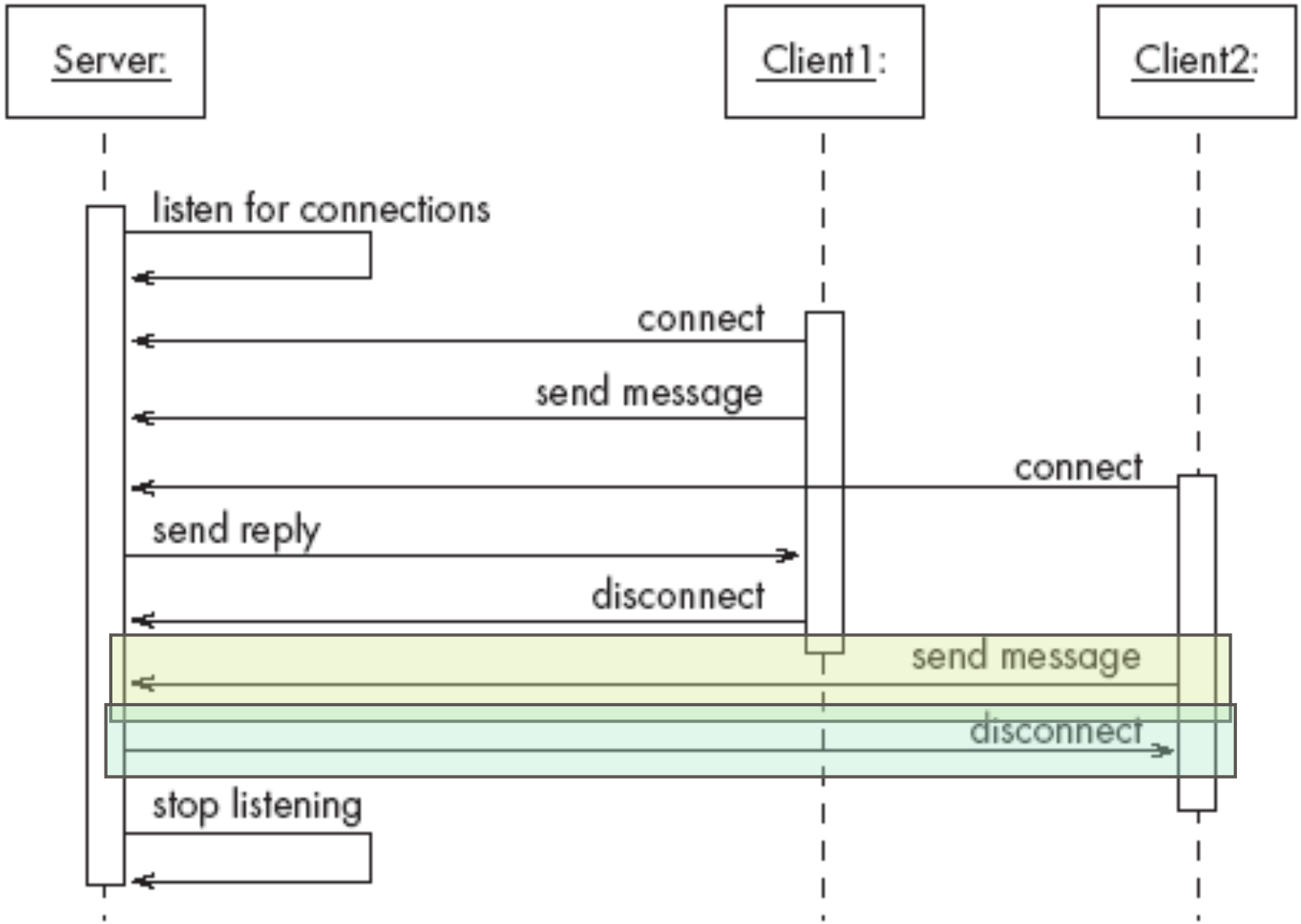
- The World Wide Web
- Network File System
- Transaction Processing System
- Remote Display System
- Communication System
- Database System
- Email
- DNS


Server:
Client1:
Client2:
listen for connections
connect
send message
connect
send reply
disconnect
send message
disconnect
stop listening

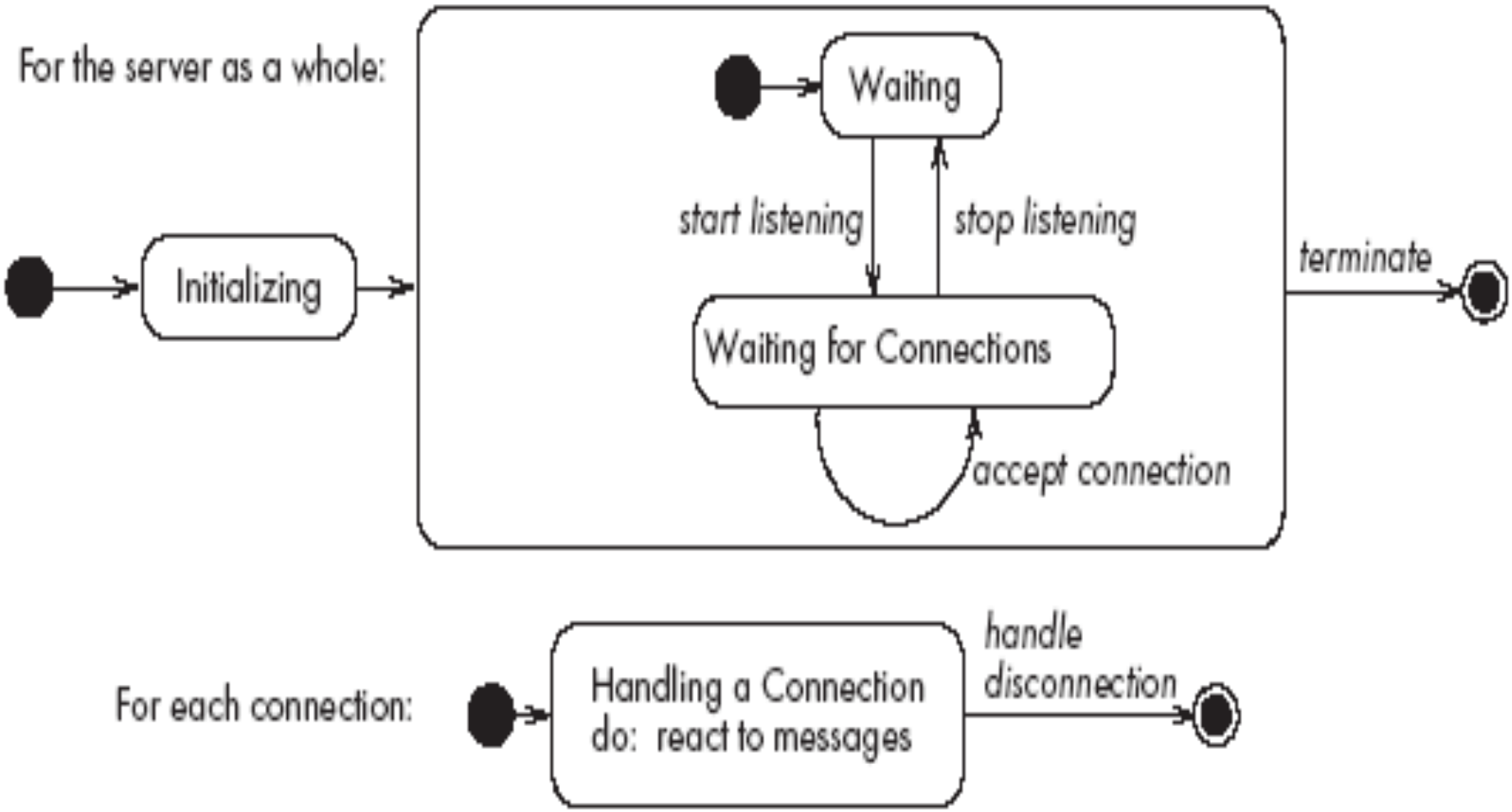

Server:
Client1:
Client2:
listen for connections
connect
send message
connect
send reply
disconnect
send message
disconnect
stop listening

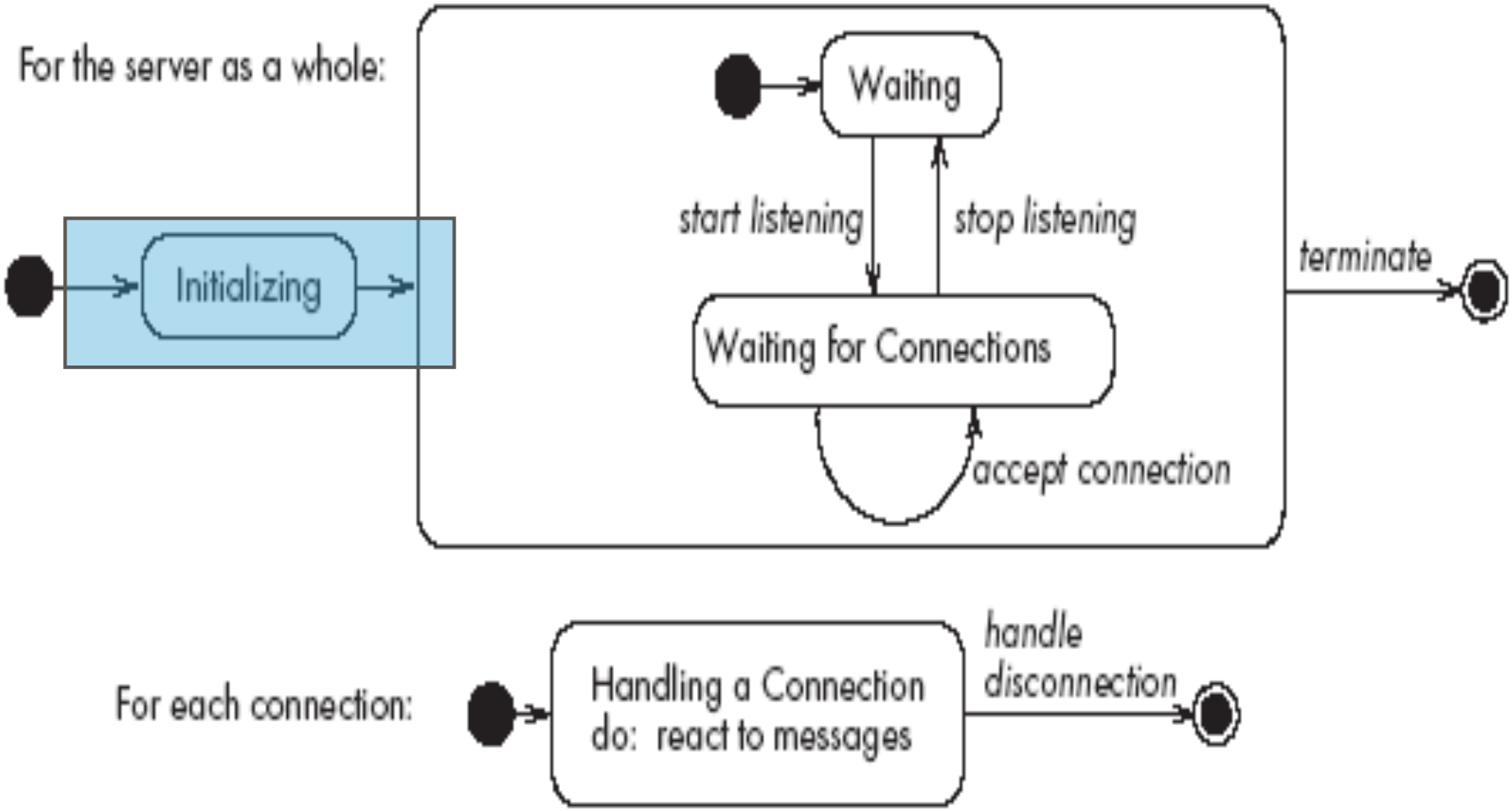

Server:
Client1:
Client2:
listen for connections
connect
send message
connect
send reply
disconnect
send message
disconnect
stop listening

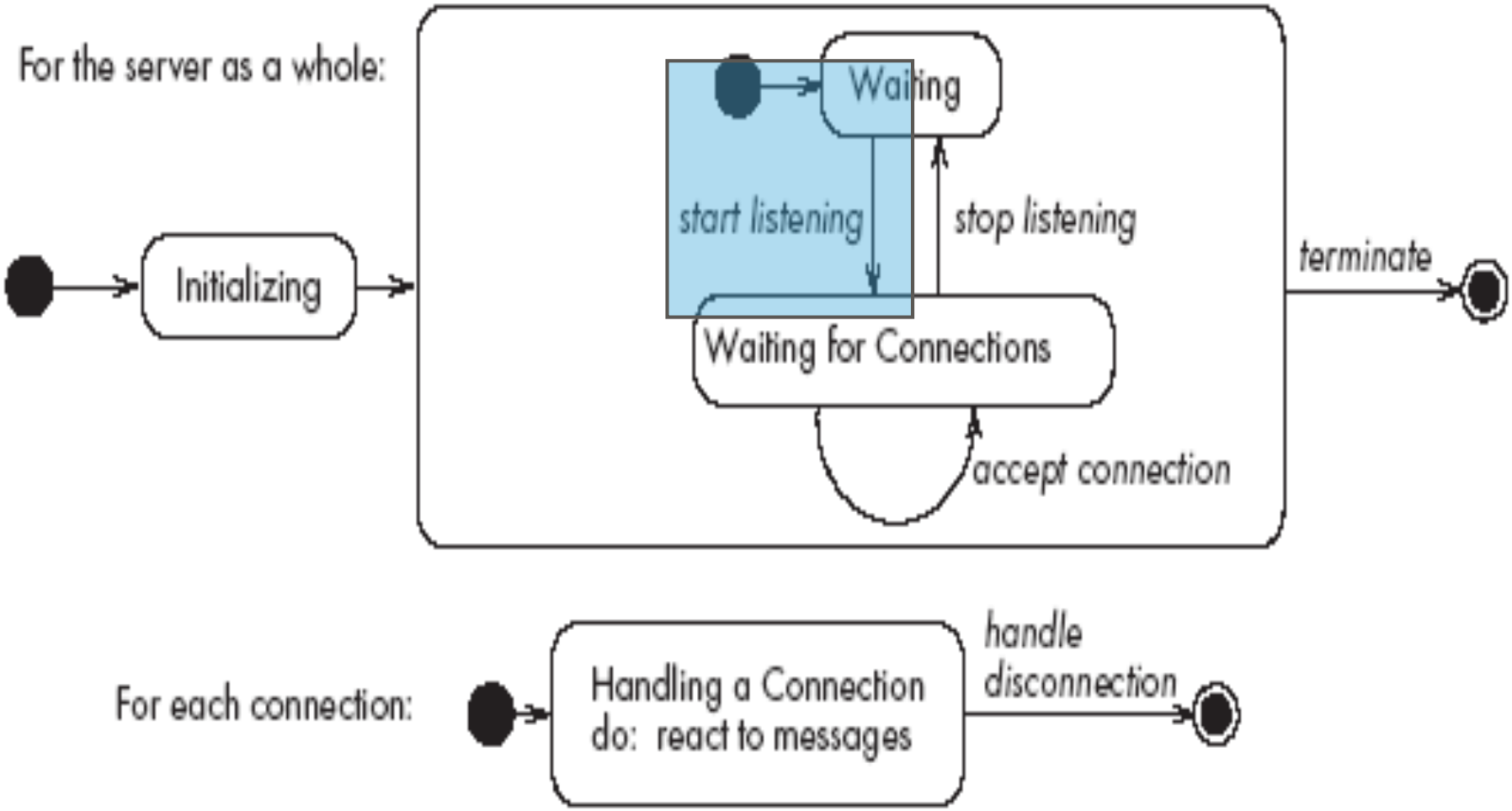

Server:
Client1:
Client2:
listen for connections
connect
send message
connect
send reply
disconnect
send message
disconnect
stop listening

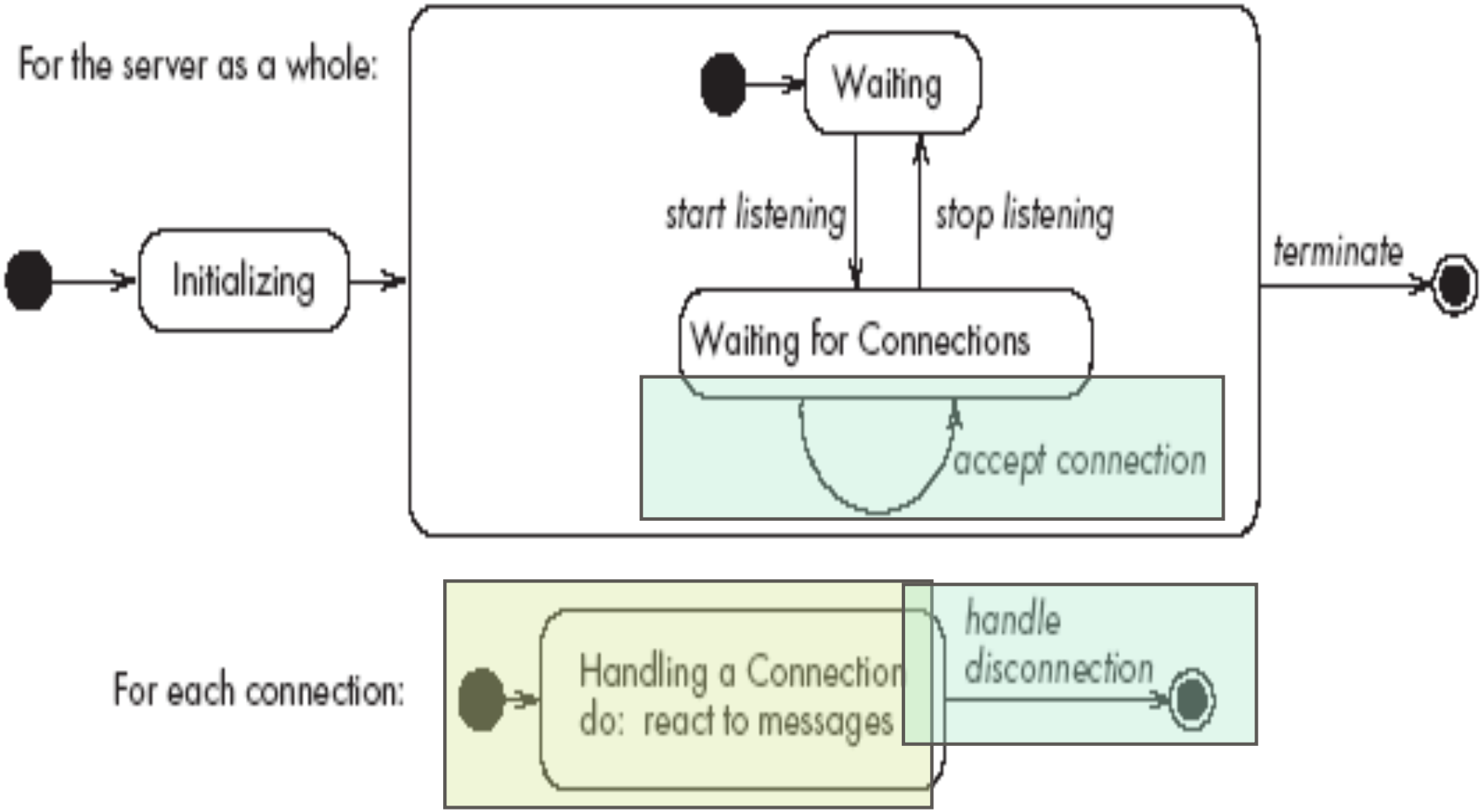

Server:
Client1:
Client2:
listen for connections
connect
send message
connect
send reply
disconnect
send message
disconnect
stop listening

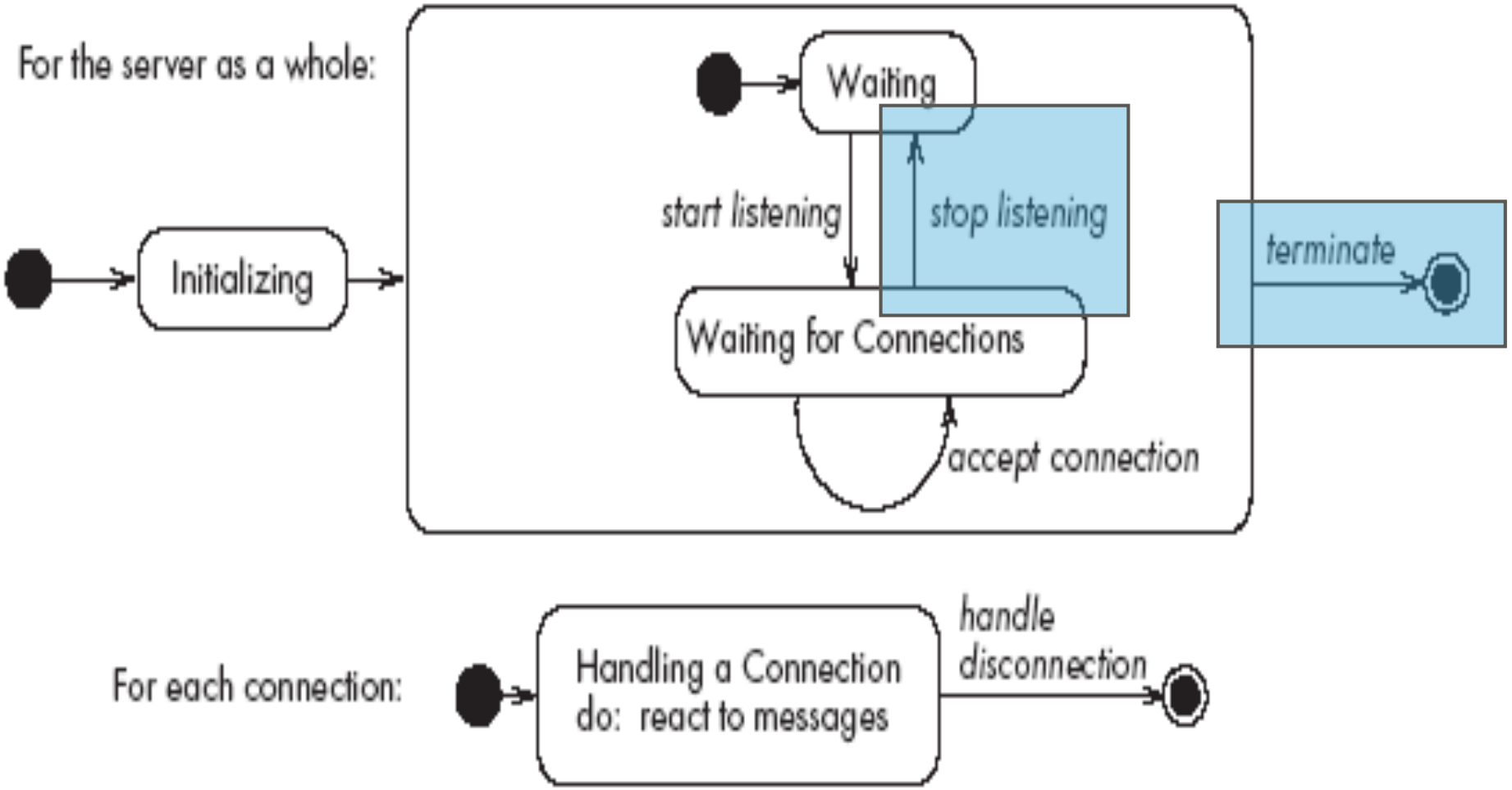
ACTIVITIES OF A SERVER



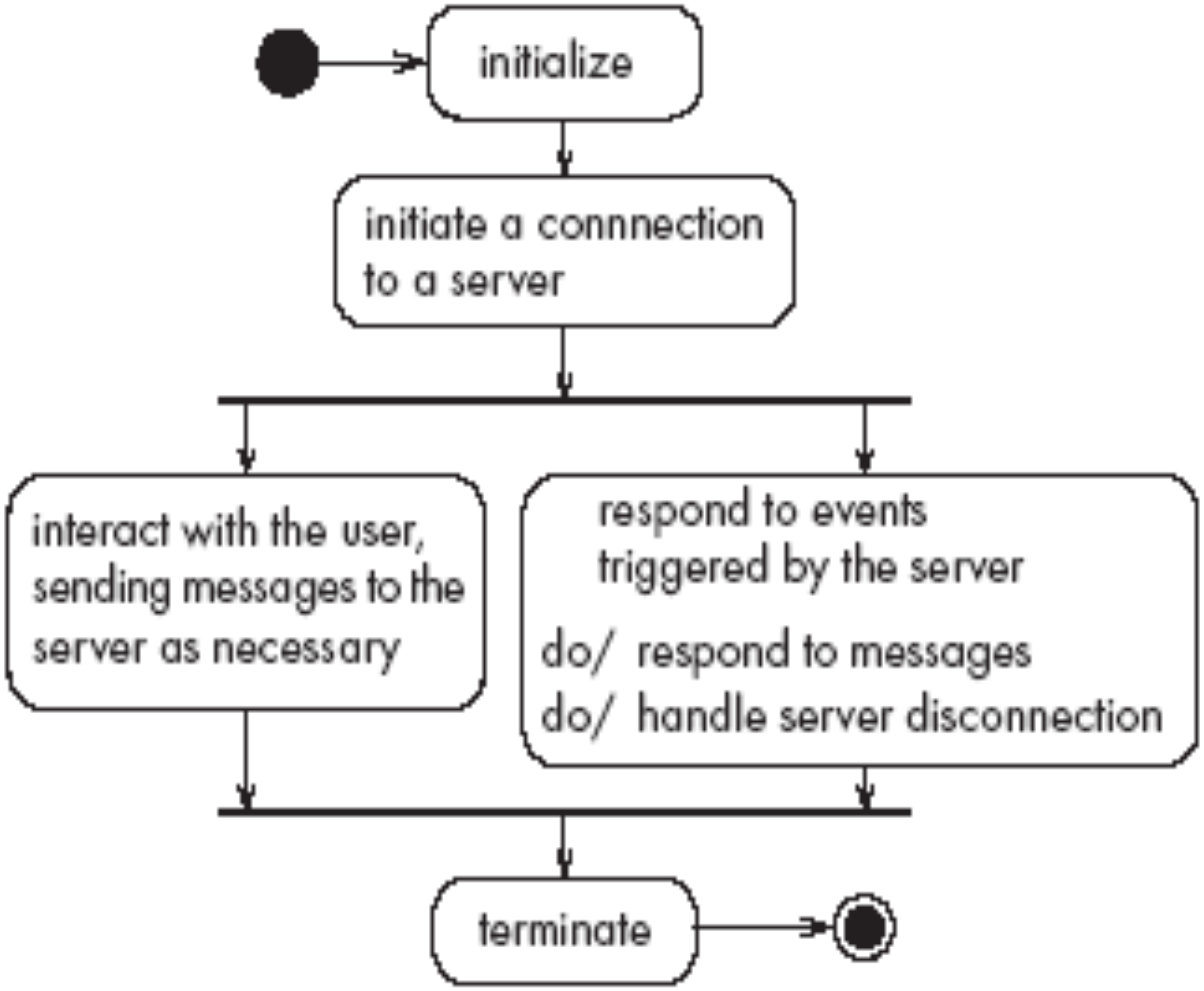

For the server as a whole:
Waiting
start listening
stop listening
terminate
Initializing
Waiting for Connections
accept connection
For each connection:
Handling a Connection
do: react to messages
handle
disconnection

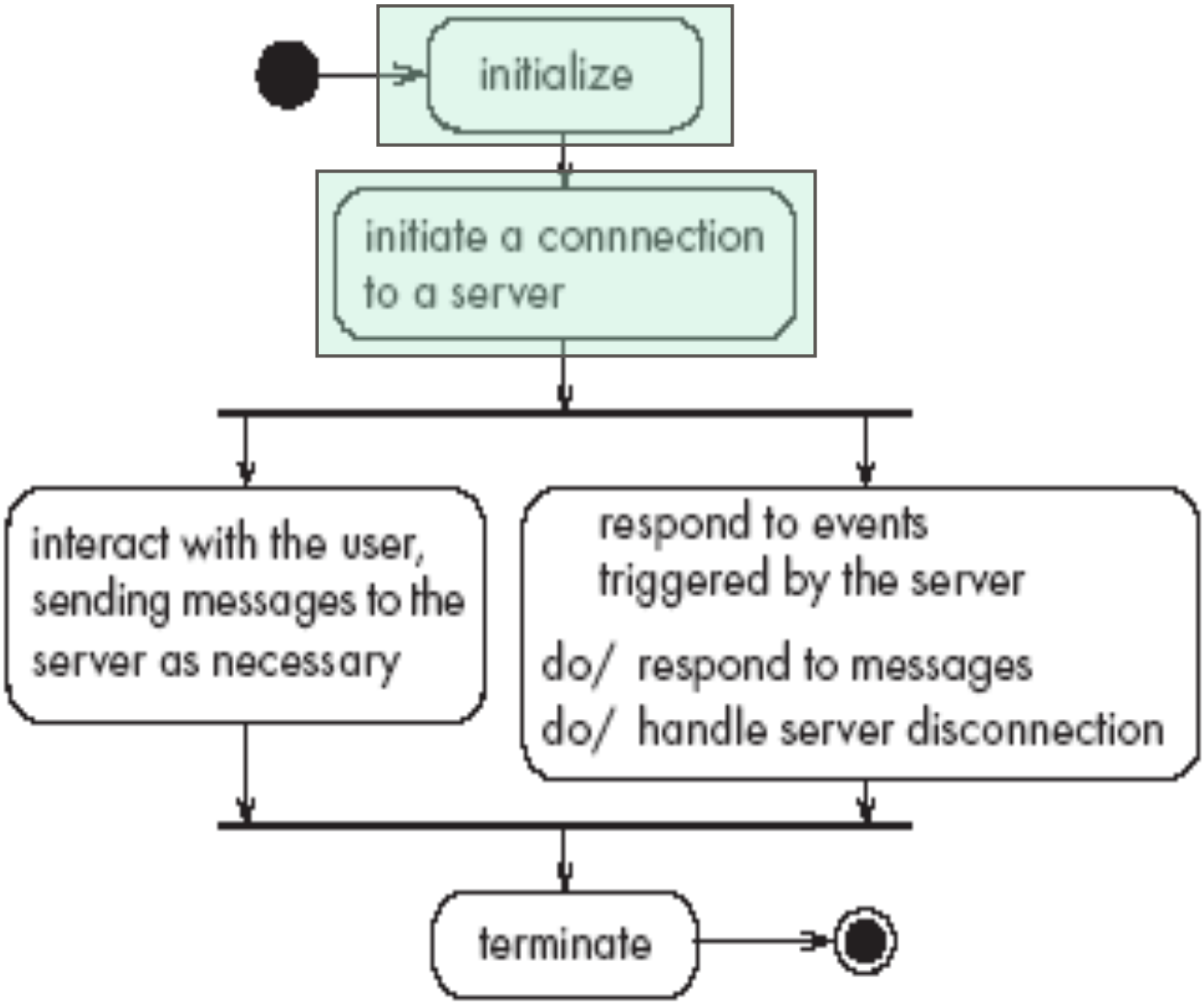

For the server as a whole:
Initializing
Waiting
start listening
stop listening
Waiting for Connections
accept connection
terminate
For each connection:
Handling a Connection
do: react to messages
handle
disconnection

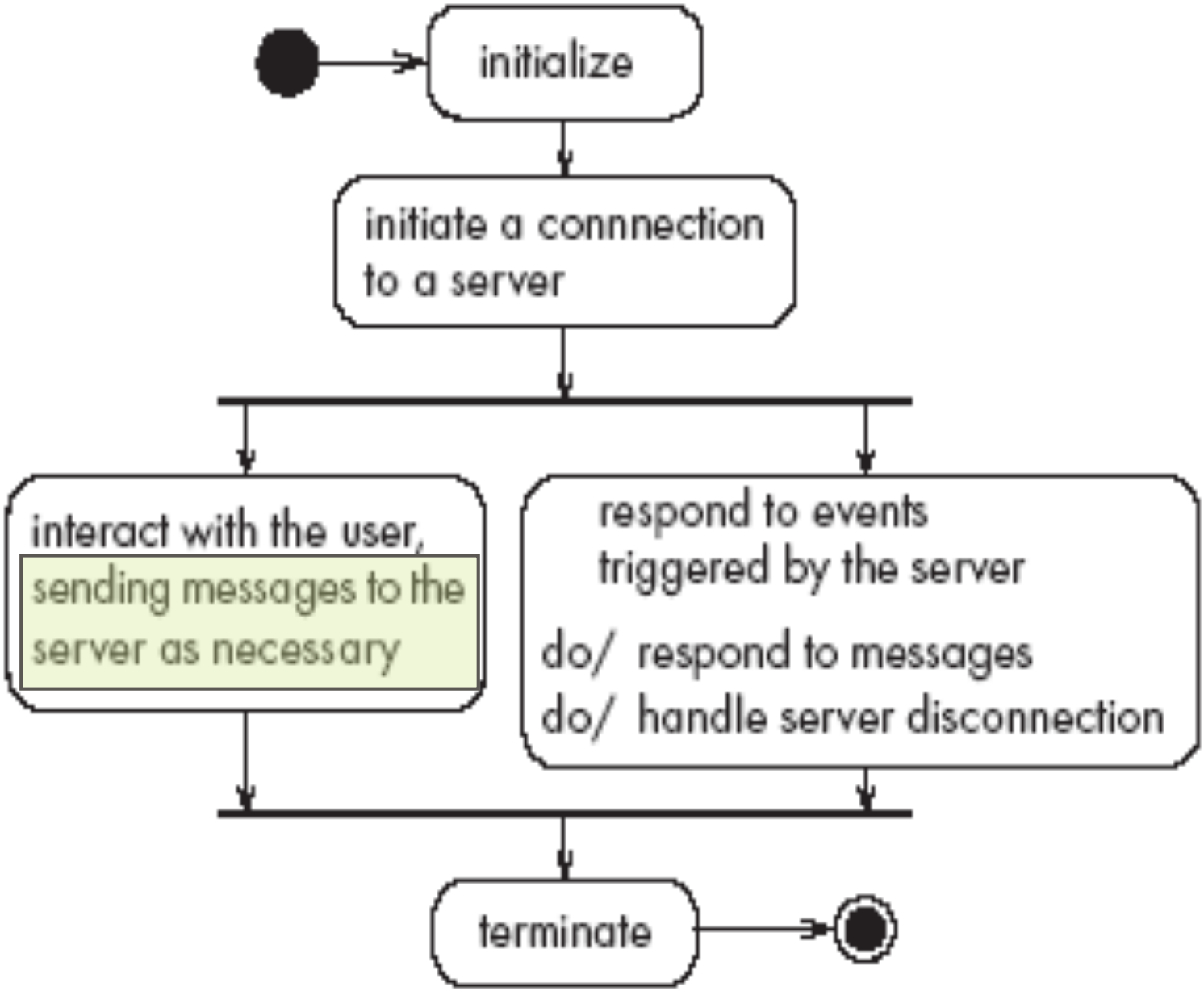

For the server as a whole:
Waiting
start listening
stop listening
terminate
Initializing
Waiting for Connections
accept connection
For each connection:
handle
disconnection
Handling a Connection
do: react to messages

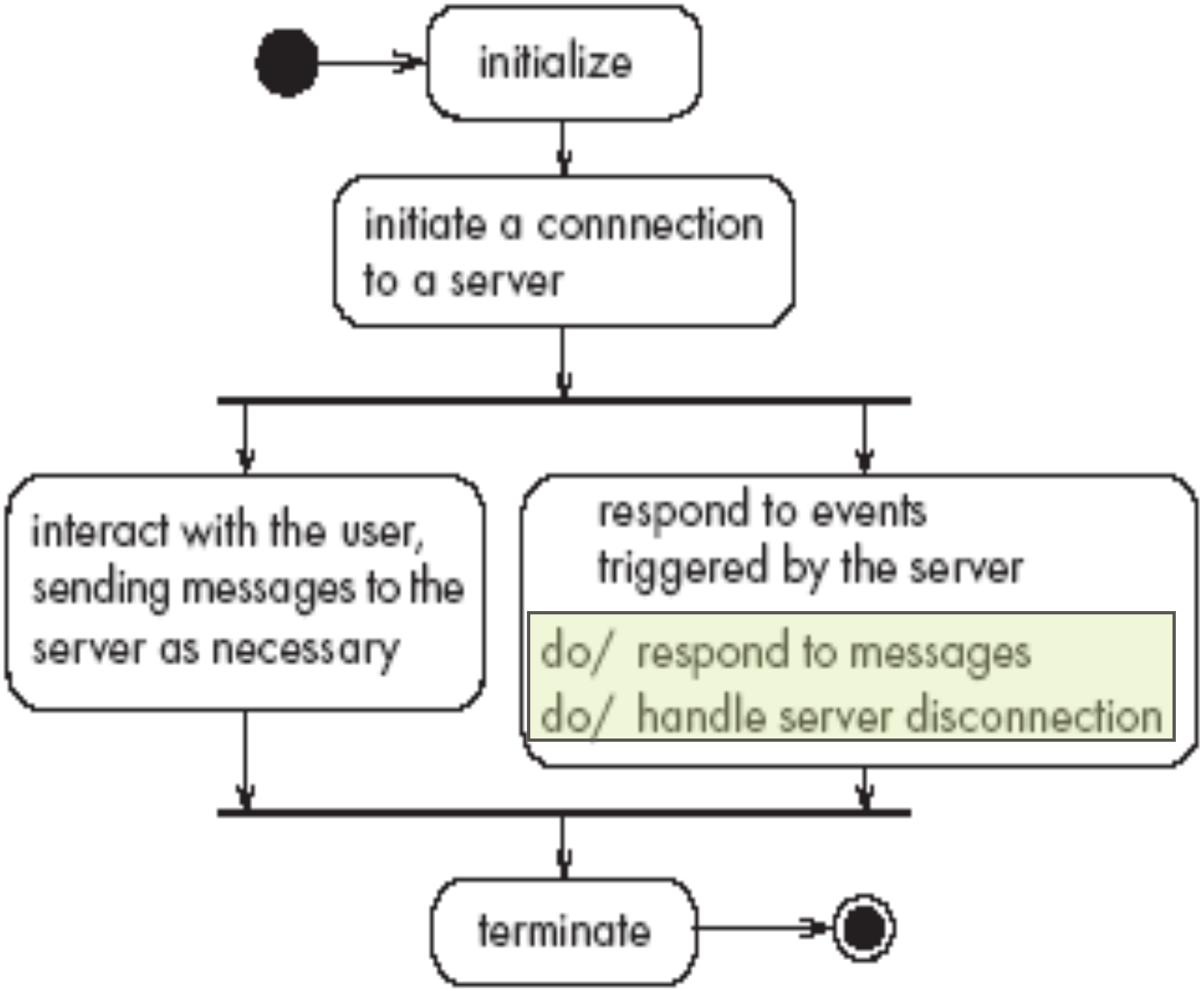

For the server as a whole:
Waiting
start listening
stop listening
terminate
Initializing
Waiting for Connections
accept connection
For each connection:
Handling a Connection
do: react to messages
handle
disconnection

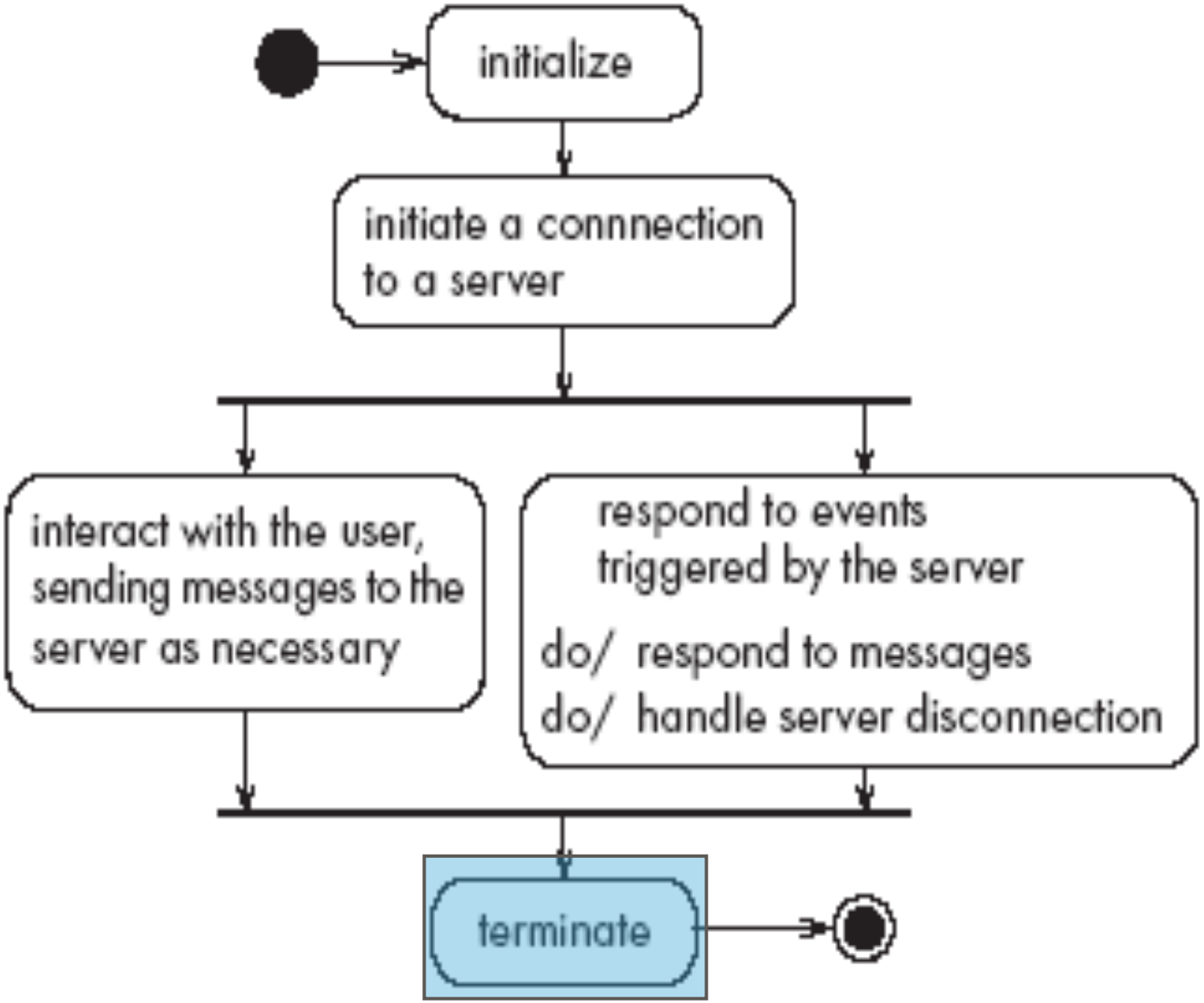
ACTIVITIES OF A CLIENT



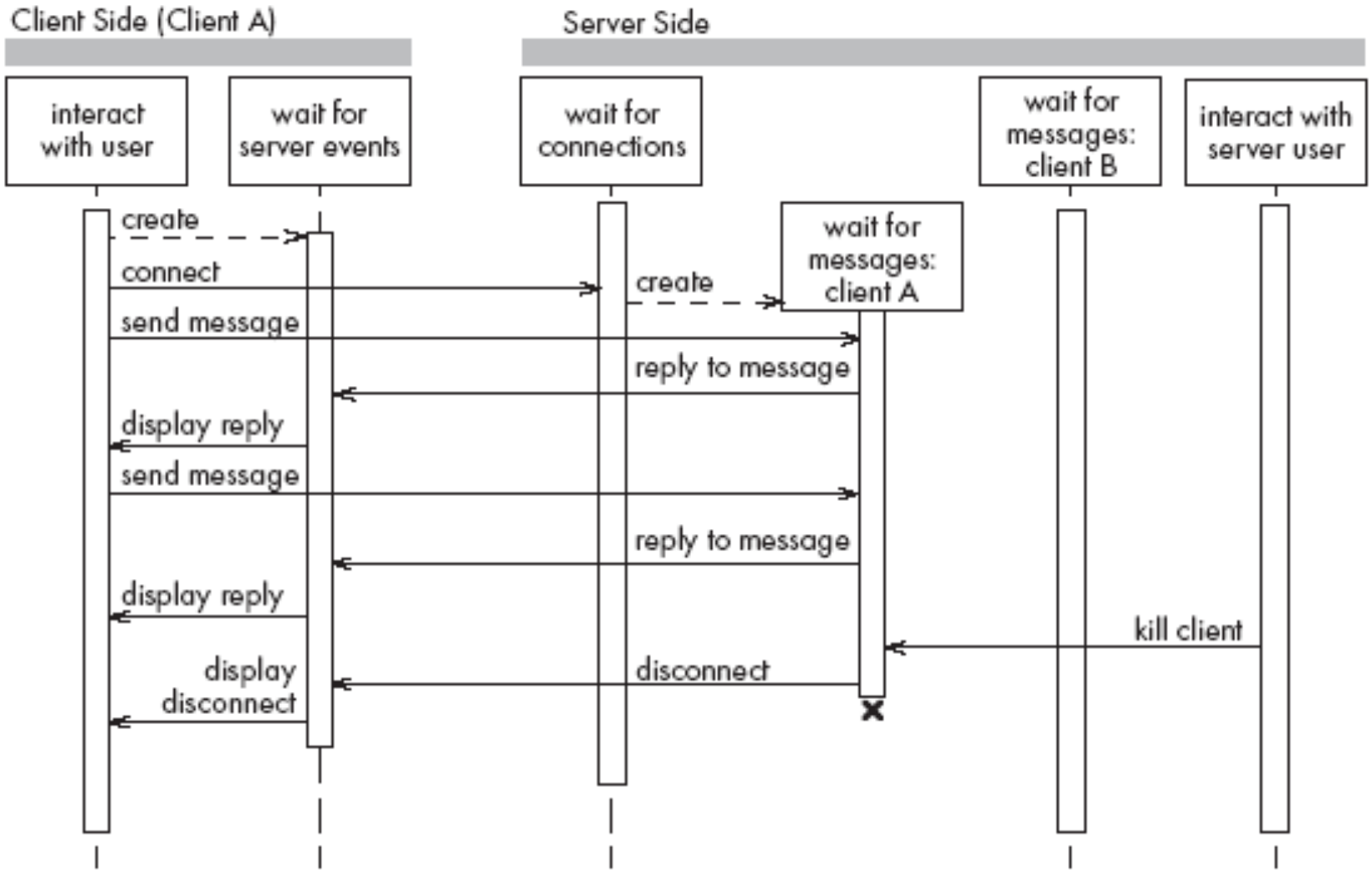

initialize
initiate a connnection
to a server
interact with the user,
sending messages to the
server as necessary
respond to events
triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate

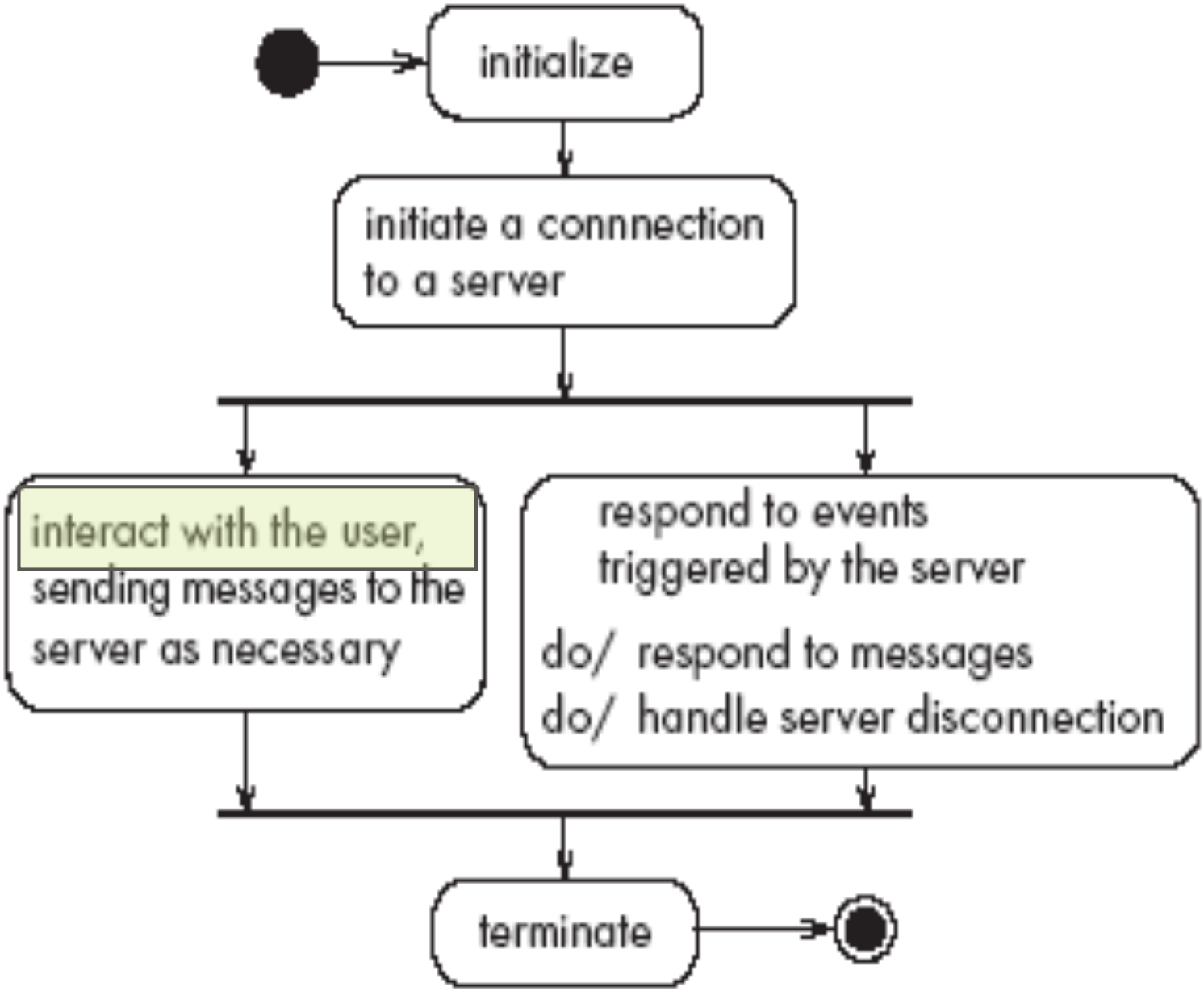

initialize
initiate a connnection
to a server
interact with the user,
sending messages to the
server as necessary
respond to events
triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate


initialize
initiate a connnection to a server
interact with the user, sending messages to the server as necessary
respond to events triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate

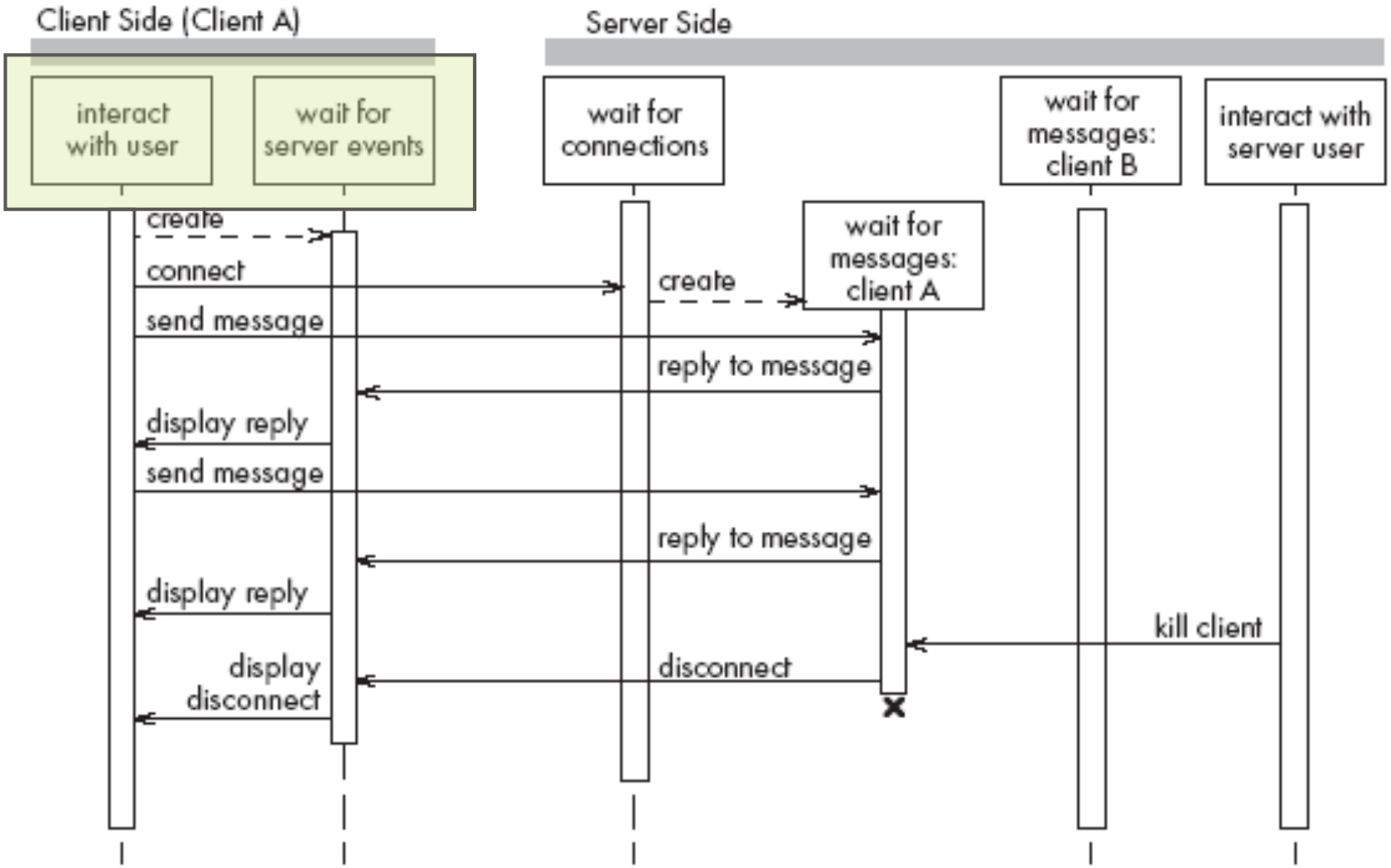

initialize
initiate a connnection to a server
interact with the user, sending messages to the server as necessary
respond to events triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate

THREADS IN A CLIENT–SERVER SYSTEM

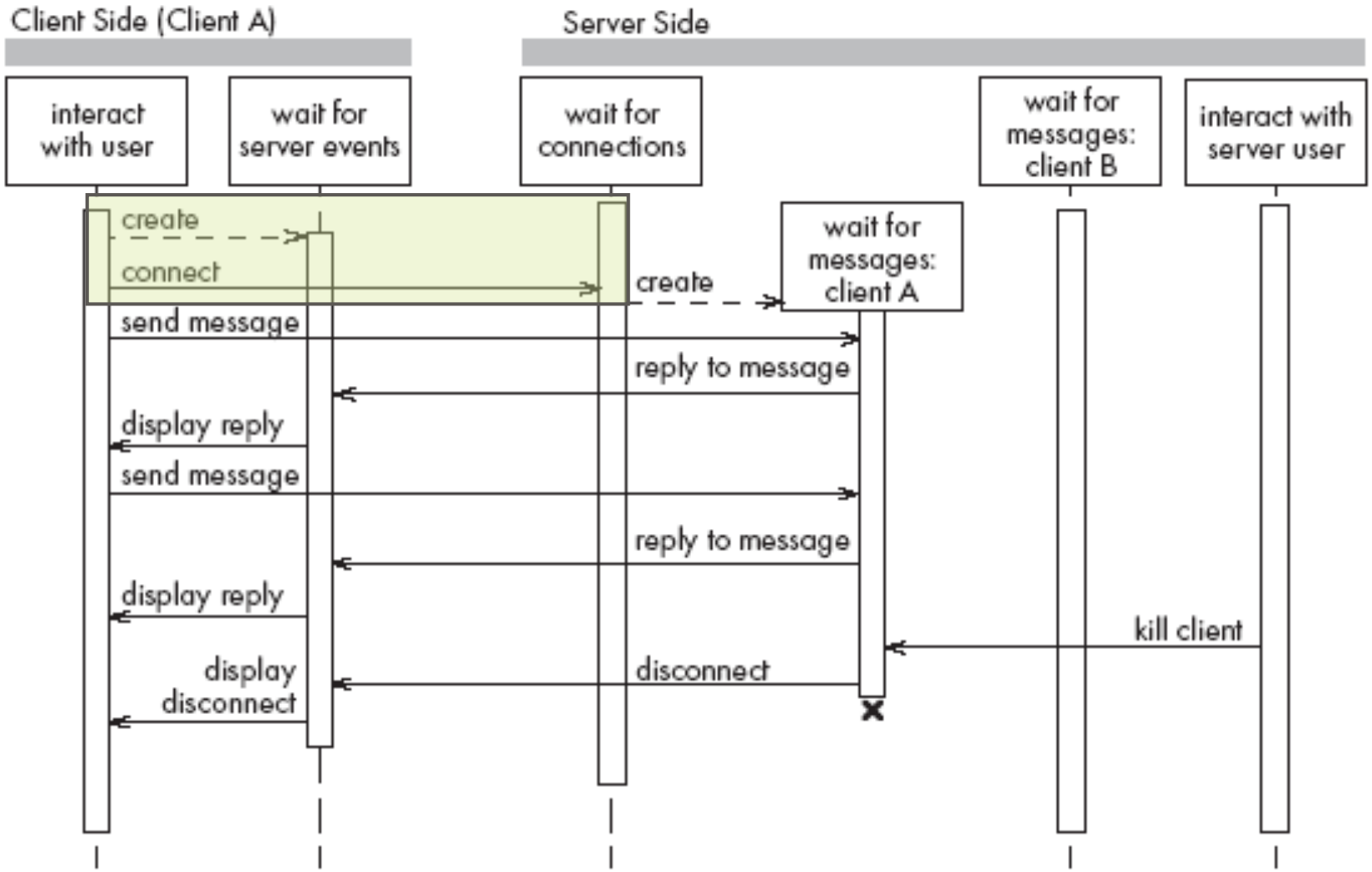



initialize
initiate a connnection to a server
interact with the user, sending messages to the server as necessary
respond to events triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate

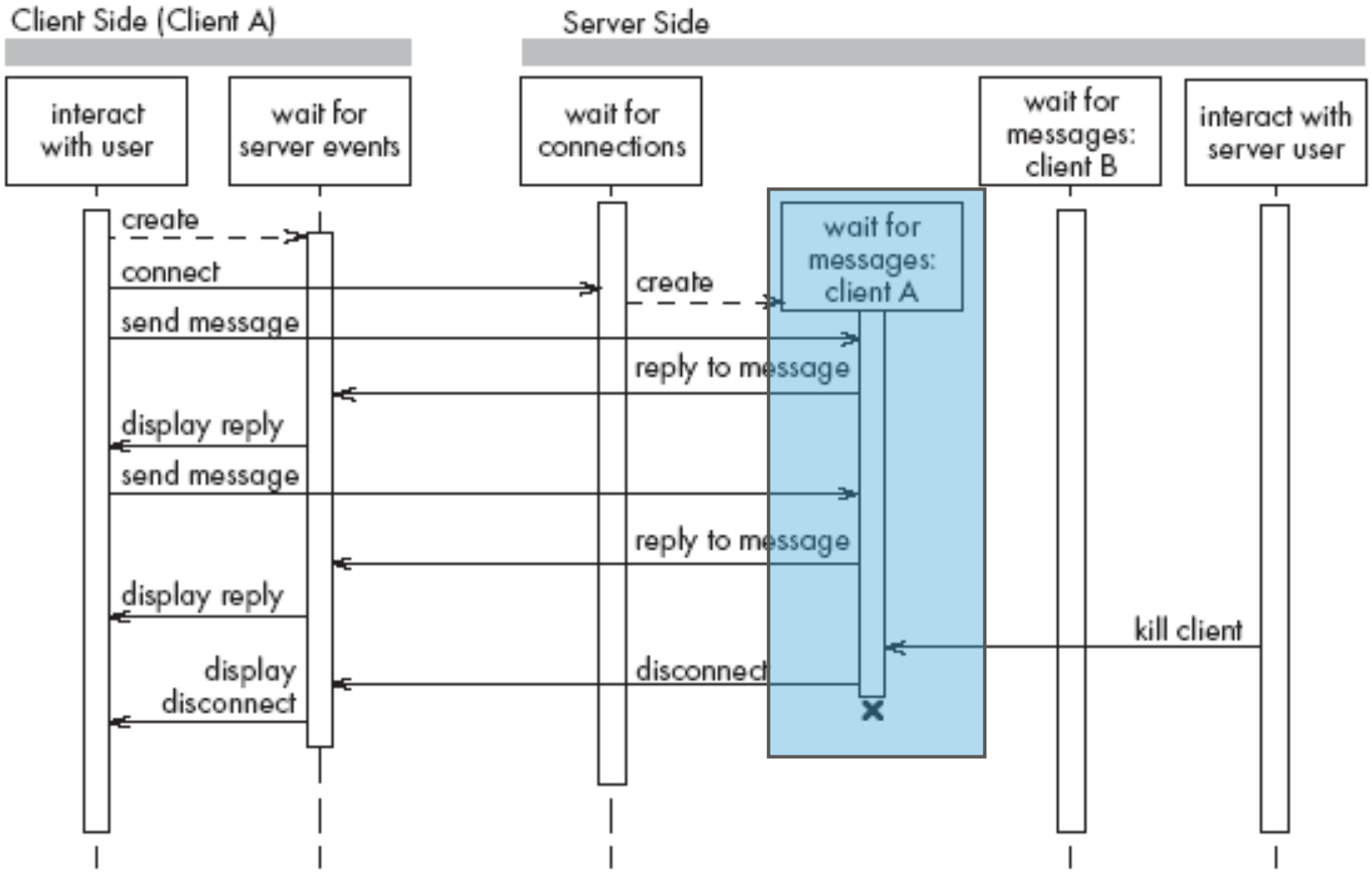
THREADS IN A CLIENT–SERVER SYSTEM



THREADS IN A CLIENT-SERVER SYSTEM



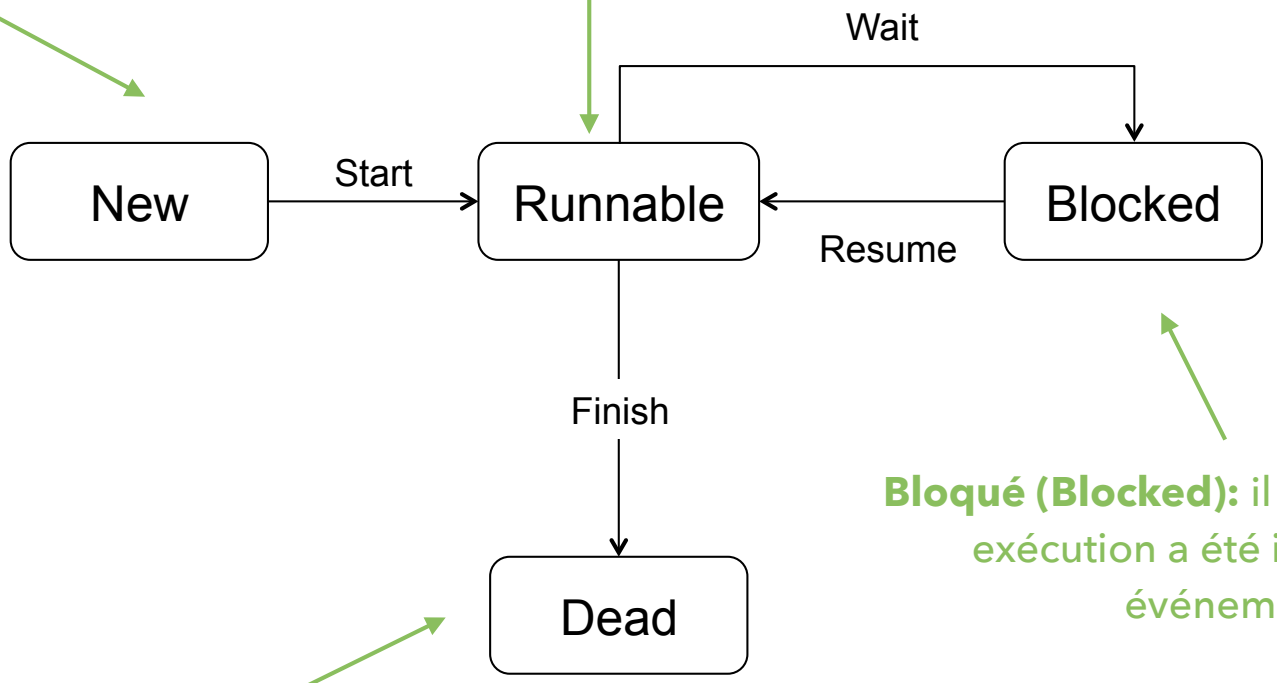
THREADS IN A CLIENT-SERVER SYSTEM



Thread States

Exécutable ou prêt (Runnable or Ready): il exécute couramment ou il est prêt à exécuter

Nouveau (New): il a été créé, mais n'a pas encore commencer son exécution

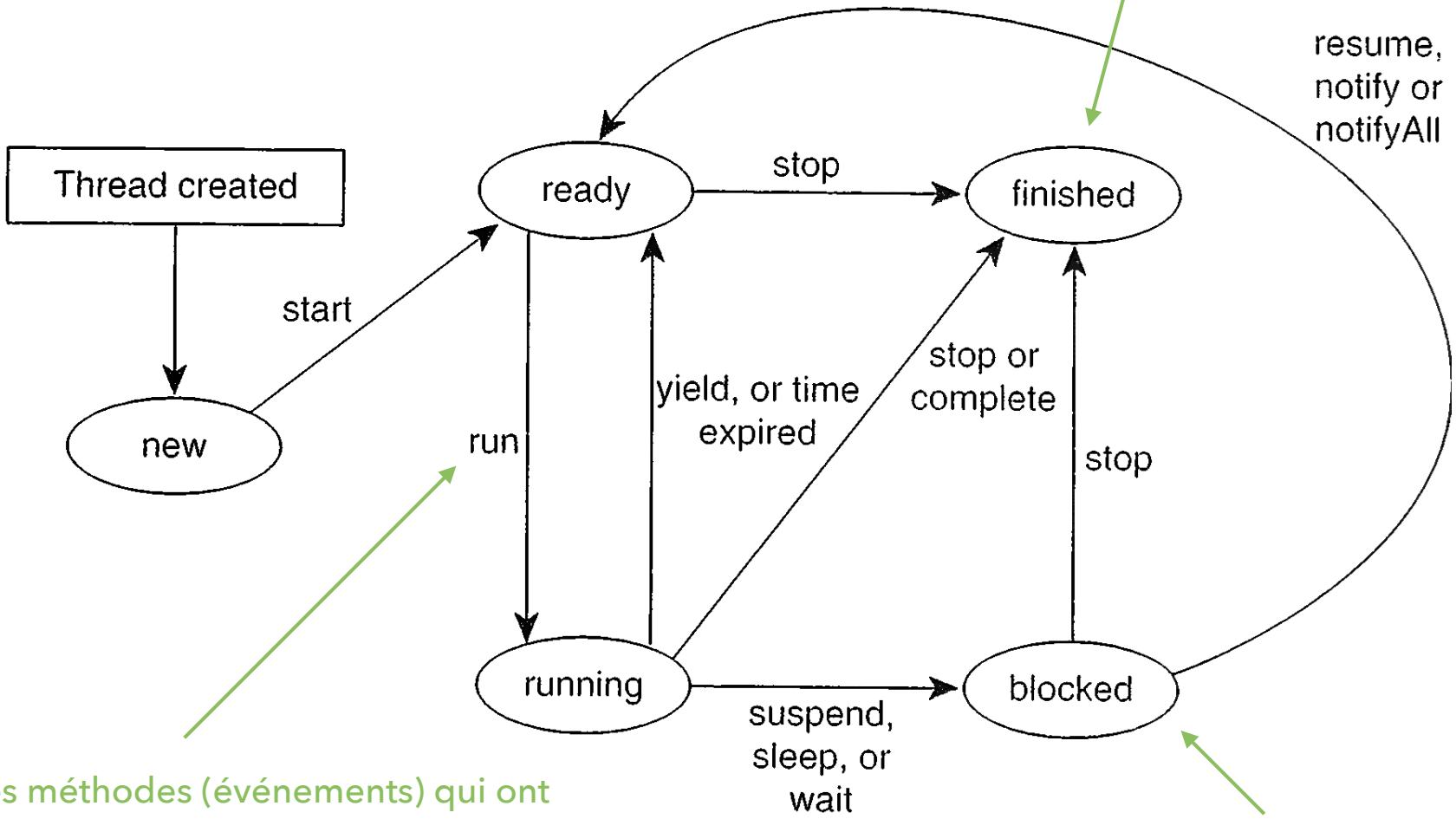


Bloqué (Blocked): il exécutait, mais son exécution a été interrompue par un événement parmi plusieurs

Mort (Dead): il n'est plus actif dans aucun sens

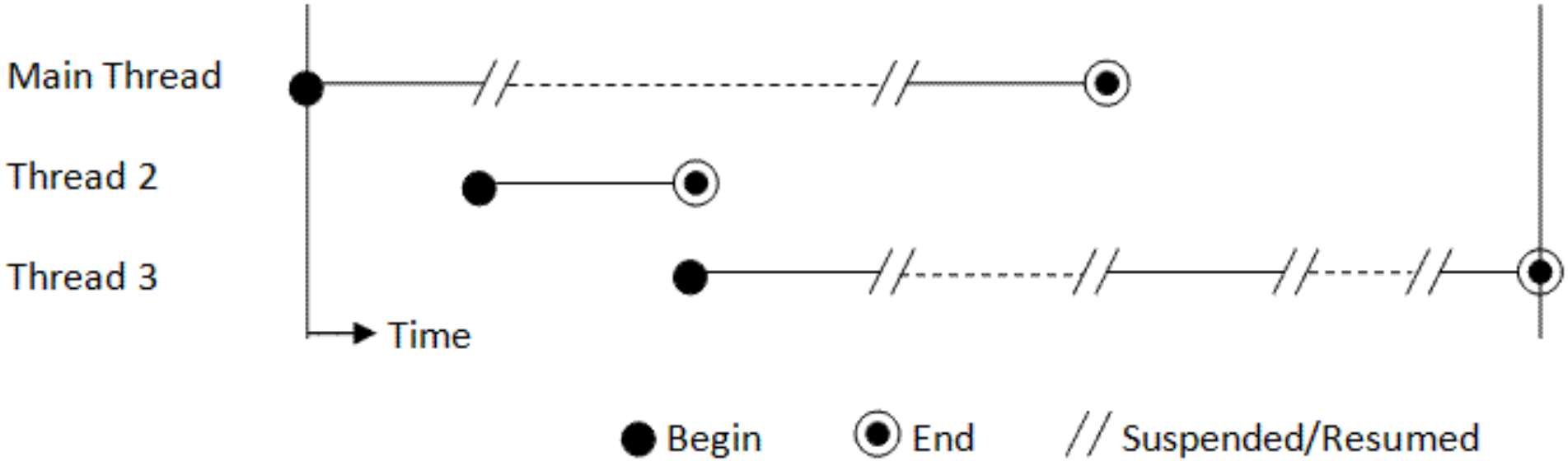
ÉTATS DE TÂCHES EN JAVA

Moins morbide que « dead »



Les méthodes (événements) qui ont déclenché les changements d'états.

« Blocked » est séparé de « ready » (les deux attendent d'être exécutés)


Main Thread
Thread 2
Thread 3
Time
Begin
End
// Suspended/Resumed

```
public class MyThread extends Thread {
    public MyThread() {
        super("MyThread");
    }
    public void run() {
        //Code
    }
}
```

« extend » le classe de base Thread

Votre code concurrent

Commence le táche concurrent

```
MyThread thread = new MyThread();
thread.start()
```

```
public class MyRunnable implements Runnable {
    public void run() {
        //Code
    }
}
```

« implement » l' interface Runnable

Votre code concurrent

Commence le táche concurrent

```
Thread thread = new Thread(new MyRunnable());
thread.start()
```

Méthode préférables, pourquoi?

```
public class Helloworld {

    public static void main(String[] args) throws InterruptedException {
       System.out.println("Before");
       Thread.sleep(2000);
       System.out.println("After");
    }
}
```

2 seconds

Method statique, alors c'est le täche d'execution qui dors.

L'heure n'est pas exact

On garde les moniteurs et verrous.

On peut l'interrompre avec un InterruptedException

```
public class ThreadJoinExample {

    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable(), "t1");
        Thread t2 = new Thread(new MyRunnable(), "t2");
        Thread t3 = new Thread(new MyRunnable(), "t3");

        t1.start();

        //start second thread after waiting for 2 seconds or if it's dead
        try {
            t1.join(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        t2.start();

        //start third thread only when first thread is dead
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        t3.start();

        //let all threads finish execution before finishing main thread
        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        System.out.println("All threads are dead, exiting main thread");
    }

}
```

« join » veut dire attend pour moi SVP

Les tâches

```
class MyRunnable implements Runnable{

    @Override
    public void run() {
        System.out.println("Thread started::: »
          +Thread.currentThread().getName());
        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Thread ended::: »
          +Thread.currentThread().getName());
    }

}
```

Thread started:::t1

Thread started:::t2

Thread ended:::t1

Thread started:::t3

Thread ended:::t2

Thread ended:::t3

All threads are dead, exiting main thread

```
public class Main {

  public static void main(String[] args) {
    Thread t1 = new Thread(new LongCounter(), "t1");
    Thread t2 = new Thread(new LongCounter(), "t2");
    Thread t3 = new Thread(new LongCounter(), "t3");

    System.out.println("Start counting...");

    t1.start();
    t2.start();
    t3.start();

    try {
      t1.join();
      t2.join();
      t3.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Done counting.");
  }

}
```

Fait plusieurs choses au même temps

Mais attendez que tout soit fini

```
class LongCounter implements Runnable{
    @Override
    public void run() {
        String myName = Thread.currentThread().getName();
        try {
            long countTime = (long)(Math.random() * 1000);
            Thread.sleep(countTime);
            System.out.println(myName + " : " + countTime);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

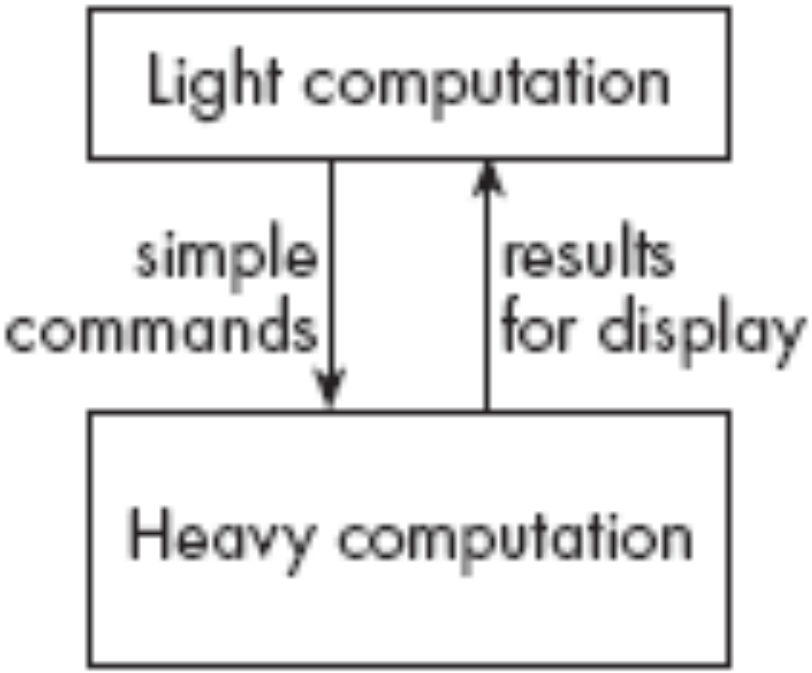
Fait un long calcul

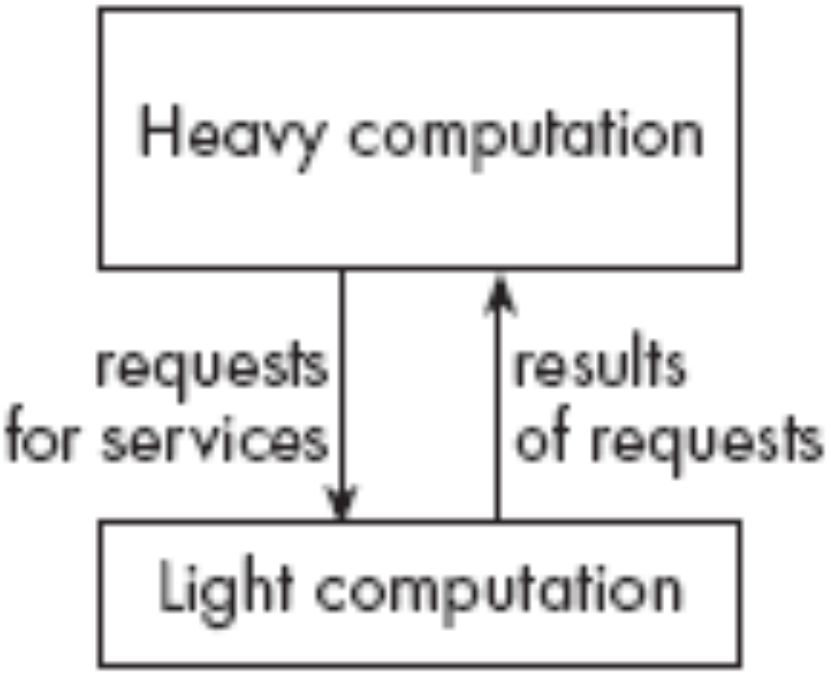
Stockez la réponse quelque part

```
$ javac Main.java ; java Main
Start counting...
t1 : 245
t2 : 469
t3 : 705
Done counting.
```

```
$ javac Main.java ; java Main
Start counting...
t3 : 144
t2 : 284
t1 : 312
Done counting.
```

THIN- VERSUS FAT-CLIENT SYSTEMS





Tasks to develop client/server application

WORK TO BE DONE

SET OF MESSAGES (LANGUAGE)

WORK TO BE DISTRIBUTED

COMMUNICATION

Advantages of client-server systems

DISTRIBUTION

SIMPLER

CONCURRENCY

DISTANCE

DATA STORAGE OPTIONS

MULTI-TENANT

Technology Needed to Build Client-Server Systems

INTERNET PROTOCOL (IP)	TRANSMISSION CONTROL PROTOCOL (TCP)	IP / HOSTNAME
MESSAGE ROUTING AND PACKETS	CONNECTIONS AND DELIVERY ASSURANCE	ADDRESS AND MAILBOX

ESTABLISHING A CONNECTION IN JAVA

Package java.net

Provides the classes for implementing networking applications.

See: Description

Interface Summary

Interface	Description
ContentHandlerFactory	This interface defines a factory for content handlers.
CookiePolicy	CookiePolicy implementations decide which cookies should be acce
CookieStore	A CookieStore object represents a storage for cookie.
DatagramSocketImplFactory	This interface defines a factory for datagram socket implementatior
FileNameMap	A simple interface which provides a mechanism to map between a f
ProtocolFamily	Represents a family of communication protocols.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOption<T>	A socket option associated with a socket.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

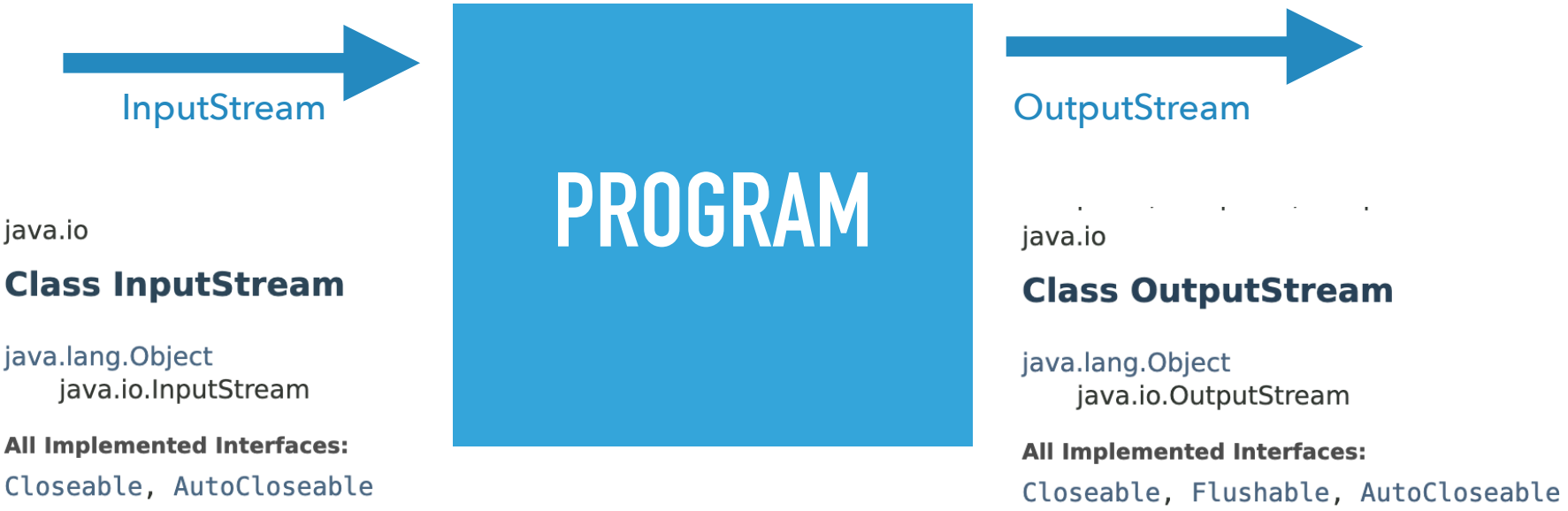
Before a connection can be established,
the server must start listening to one of the ports:

```
ServerSocket serverSocket = new ServerSocket(port);
Socket clientSocket = serverSocket.accept();
```

```
Socket clientSocket = new Socket(host, port);
```

For a client to connect to a server

EXCHANGING INFORMATION IN JAVA




```
input = clientSocket.getInputStream();

output = clientSocket.getOutputStream();
```

SENDING AND RECEIVING

MESSAGES

Raw Bytes (without any filters)

```
output.write(msg);
msg = input.read();
```

DataInputStream / DataOutputStream filters

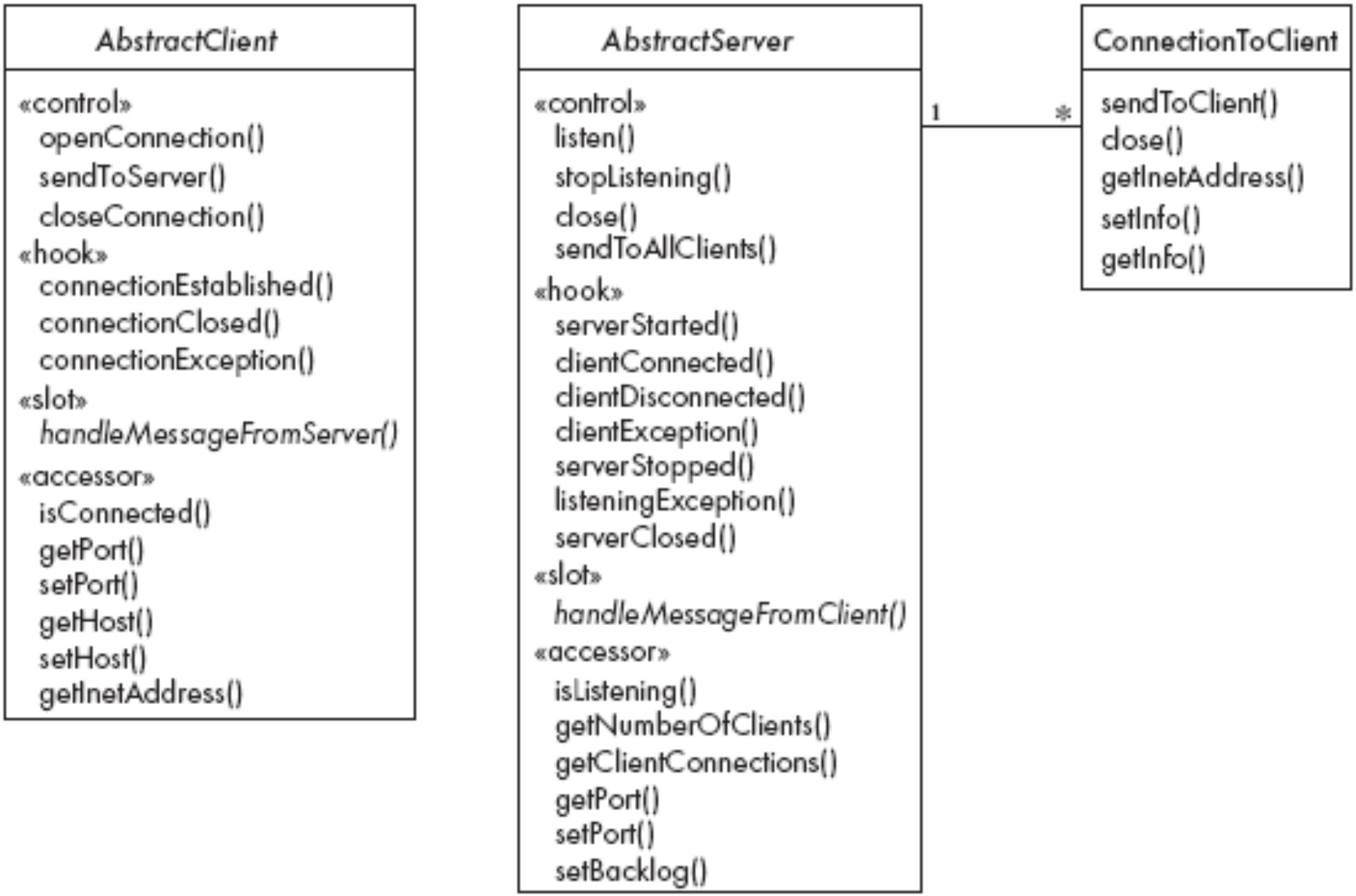
```
output.writeDouble(msg);
msg = input.readDouble();
```

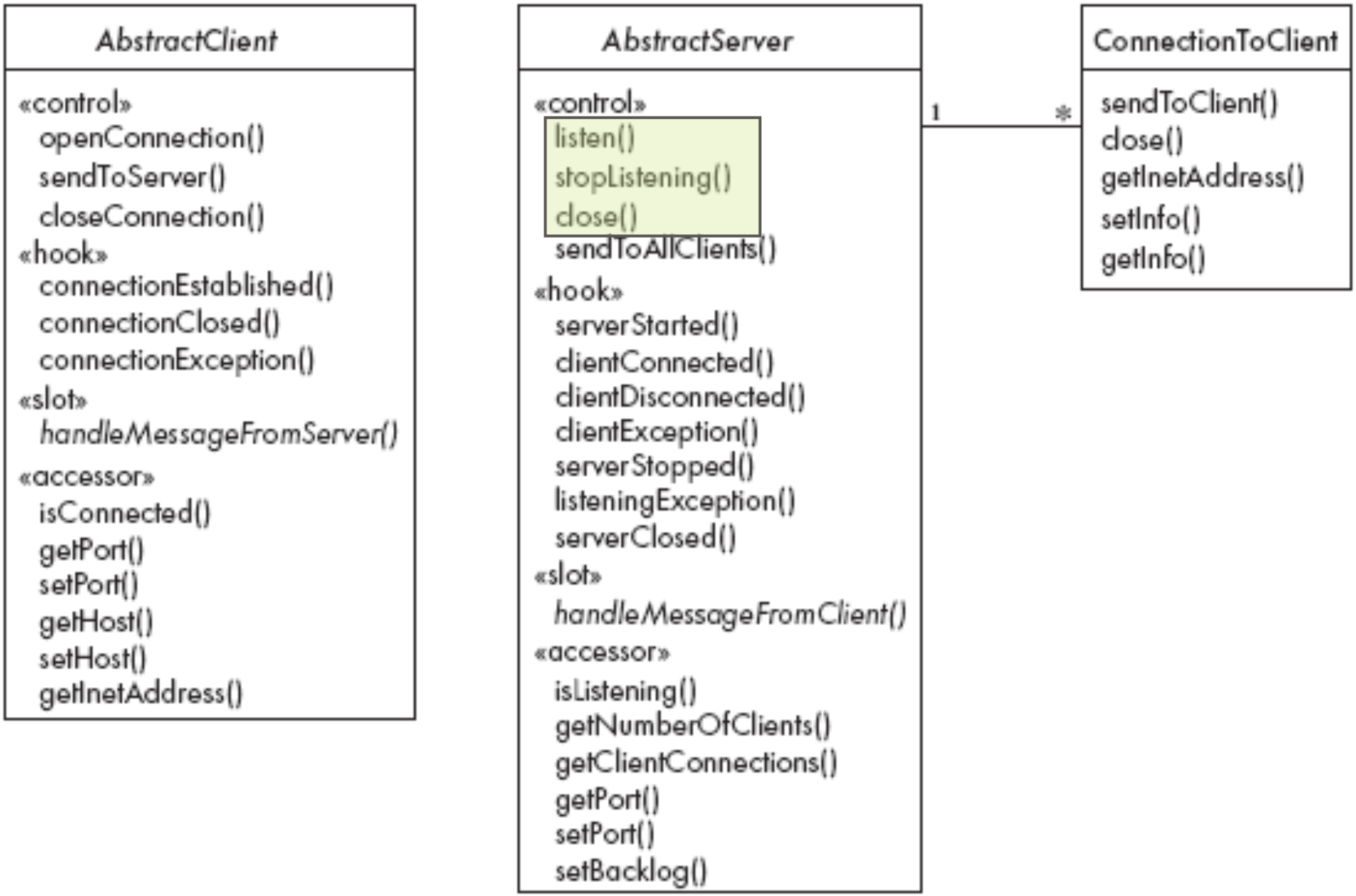
ObjectInputStream / ObjectOutputStream filters

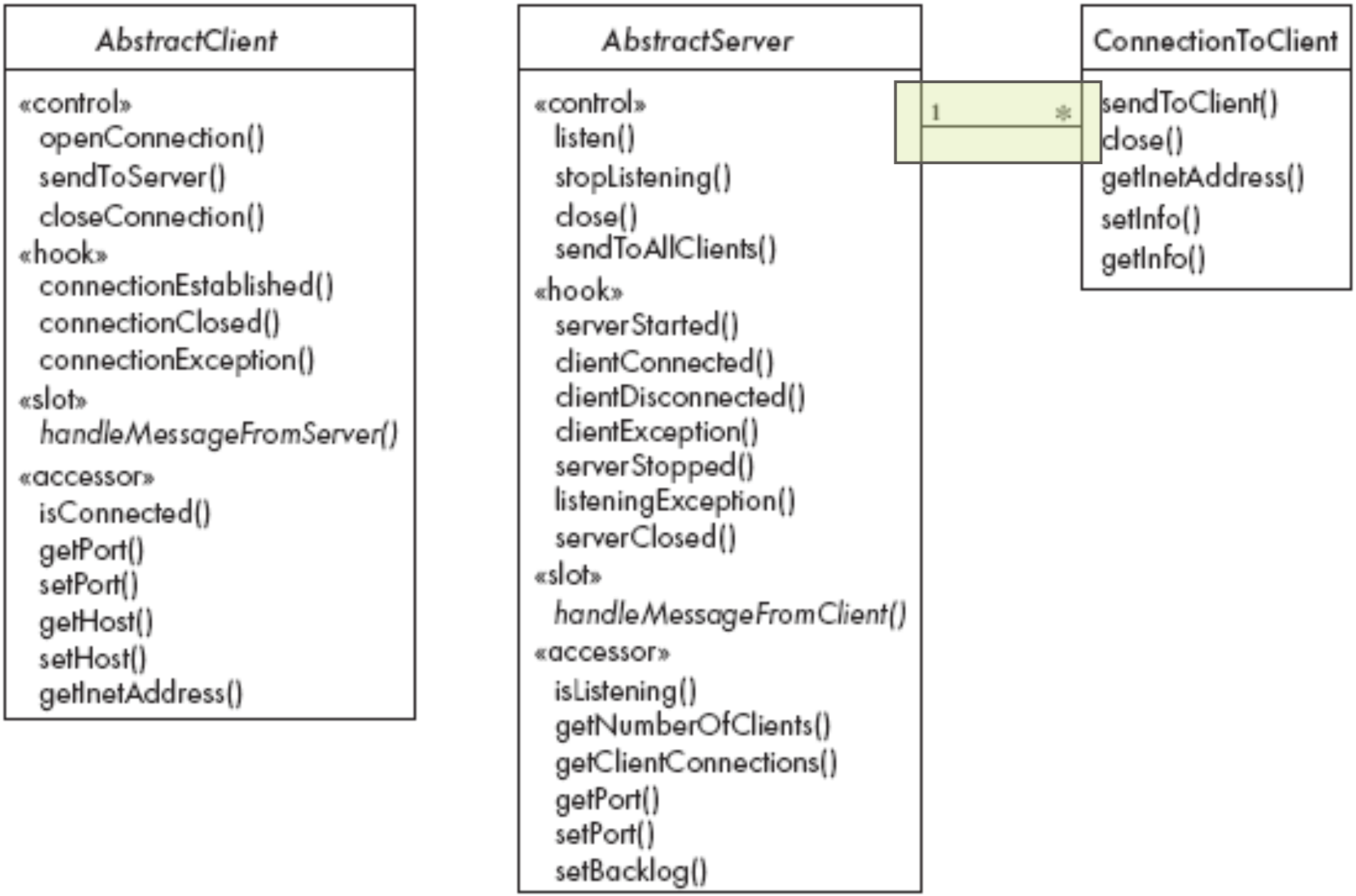
```
output.writeObject(msg);
msg = input.readObject();
```

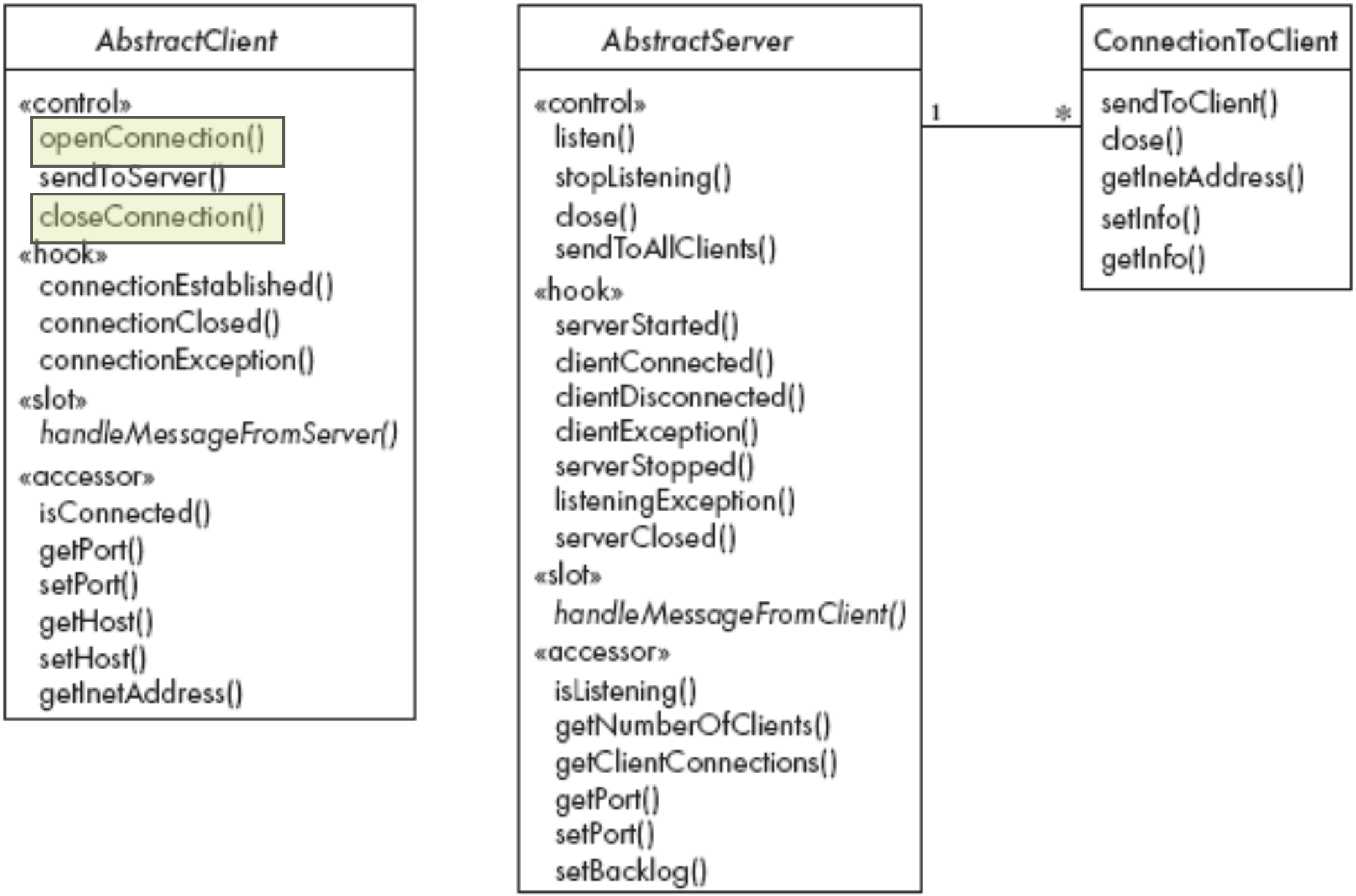
OBJECT CLIENT-SERVER FRAMEWORK

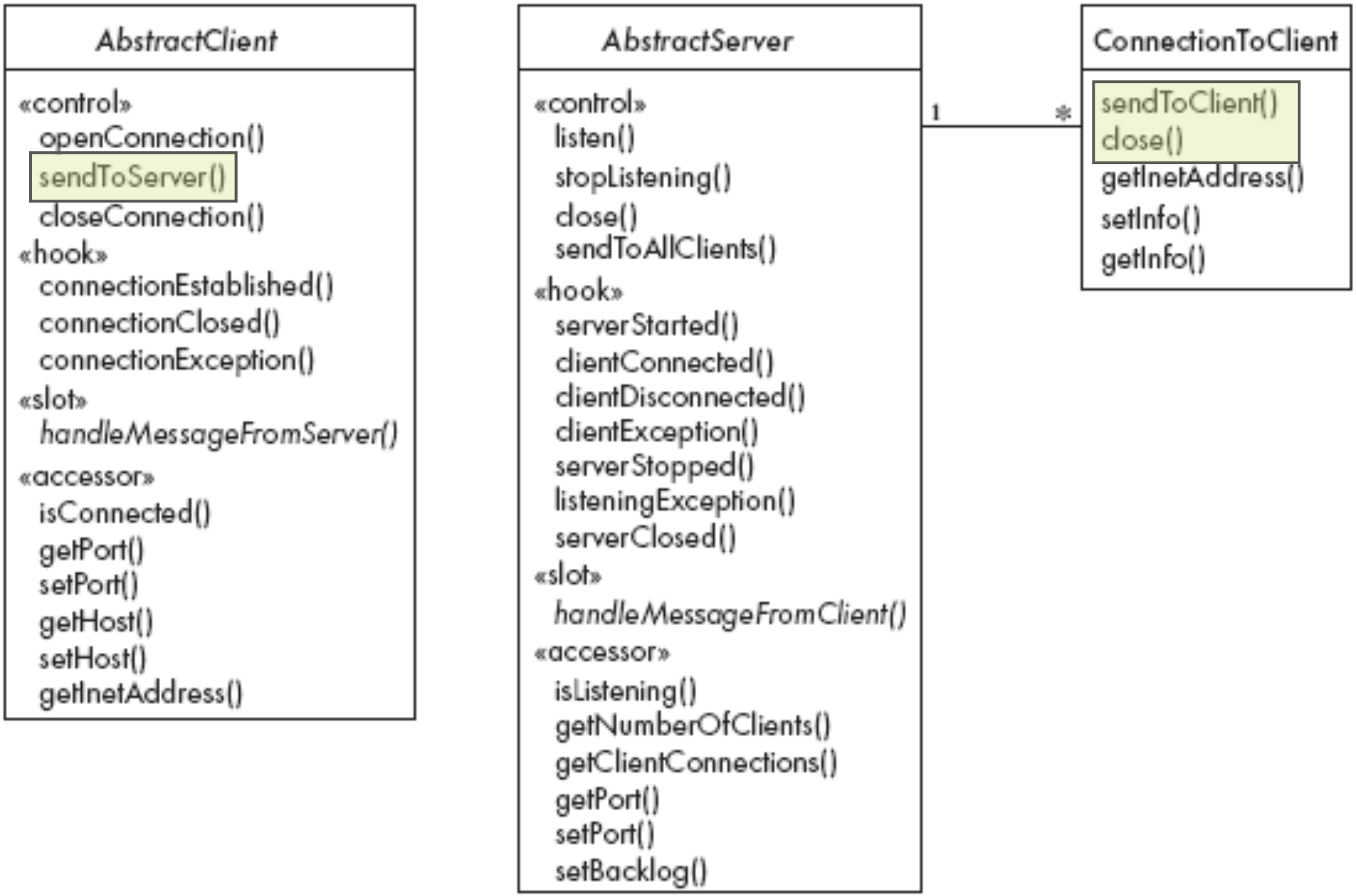
OCSF


AbstractClient
«control»
openConnection()
sendToServer()
closeConnection()
«hook»
connectionEstablished()
connectionClosed()
connectionException()
«slot»
handleMessageFromServer()
«accessor»
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
AbstractServer
«control»
listen()
stopListening()
close()
sendToAllClients()
«hook»
serverStarted()
clientConnected()
clientDisconnected()
clientException()
serverStopped()
listeningException()
serverClosed()
«slot»
handleMessageFromClient()
«accessor»
isListening()
getNumberOfClients()
getClientConnections()
getPort()
setPort()
setBacklog()
1
*
ConnectionToClient
sendToClient()
close()
getInetAddress()
setInfo()
getInfo()


AbstractClient
«control»
openConnection()
sendToServer()
closeConnection()
«hook»
connectionEstablished()
connectionClosed()
connectionException()
«slot»
handleMessageFromServer()
«accessor»
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
AbstractServer
«control»
listen()
stopListening()
close()
sendToAllClients()
«hook»
serverStarted()
clientConnected()
clientDisconnected()
clientException()
serverStopped()
listeningException()
serverClosed()
«slot»
handleMessageFromClient()
«accessor»
isListening()
getNumberOfClients()
getClientConnections()
getPort()
setPort()
setBacklog()
1
*
ConnectionToClient
sendToClient()
close()
getInetAddress()
setInfo()
getInfo()


AbstractClient
«control»
openConnection()
sendToServer()
closeConnection()
«hook»
connectionEstablished()
connectionClosed()
connectionException()
«slot»
handleMessageFromServer()
«accessor»
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
AbstractServer
«control»
listen()
stopListening()
close()
sendToAllClients()
«hook»
serverStarted()
clientConnected()
clientDisconnected()
clientException()
serverStopped()
listeningException()
serverClosed()
«slot»
handleMessageFromClient()
«accessor»
isListening()
getNumberOfClients()
getClientConnections()
getPort()
setPort()
setBacklog()
1
*
ConnectionToClient
sendToClient()
close()
getInetAddress()
setInfo()
getInfo()


AbstractClient
«control»
openConnection()
sendToServer()
closeConnection()
«hook»
connectionEstablished()
connectionClosed()
connectionException()
«slot»
handleMessageFromServer()
«accessor»
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
AbstractServer
«control»
listen()
stopListening()
close()
sendToAllClients()
«hook»
serverStarted()
clientConnected()
clientDisconnected()
clientException()
serverStopped()
listeningException()
serverClosed()
«slot»
handleMessageFromClient()
«accessor»
isListening()
getNumberOfClients()
getClientConnections()
getPort()
setPort()
setBacklog()
1
*
ConnectionToClient
sendToClient()
close()
getInetAddress()
setInfo()
getInfo()


AbstractClient
«control»
openConnection()
sendToServer()
closeConnection()
«hook»
connectionEstablished()
connectionClosed()
connectionException()
«slot»
handleMessageFromServer()
«accessor»
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
AbstractServer
«control»
listen()
stopListening()
close()
sendToAllClients()
«hook»
serverStarted()
clientConnected()
clientDisconnected()
clientException()
serverStopped()
listeningException()
serverClosed()
«slot»
handleMessageFromClient()
«accessor»
isListening()
getNumberOfClients()
getClientConnections()
getPort()
setPort()
setBacklog()
1
*
ConnectionToClient
sendToClient()
close()
getInetAddress()
setInfo()
getInfo()

Extending OCSF

SUBCLASS

CALL PUBLIC METHOD

OVERRIDE SLOTS AND HOOKS

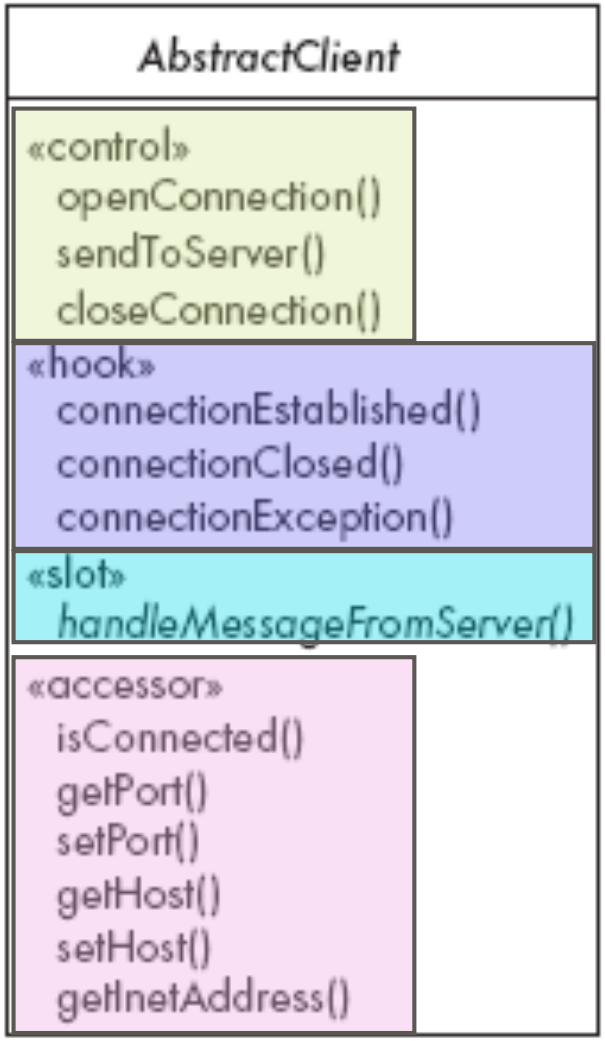
DON'T EDIT THOSE CLASSES

PROVIDED BY FRAMEWORK

EXPLICITLY DESIGNED TO BE OVERWRITTEN

THE CLIENT SIDE

- Must be subclassed
- Must implement **handleMessageFromServer**
- Could overwrite hooks
- Implements the **Runnable** interface (**run** method to loop forever*)



USING ABSTRACTCLIENT

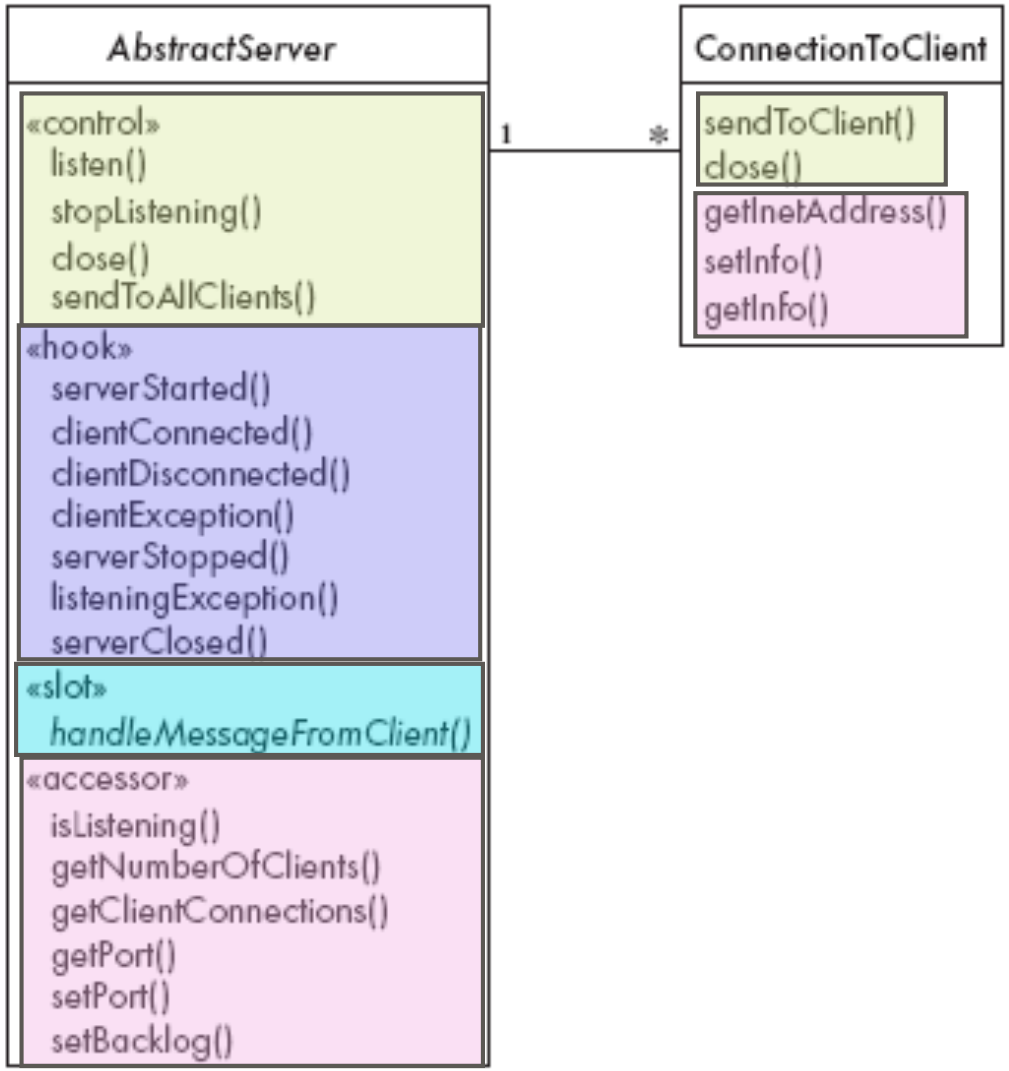
- Create a subclass of **AbstractClient**
- Implement **handleMessageFromServer** slot method
- Write code that:
  - Instantiate subclass
  - Calls **openConnection**
  - Sends messages using the **sendToServer**
- Implement callbacks
  - **connectionClosed**
  - **connectionException**

Shhhh, internals of Abstract Client

- A **Socket** which keeps all the information about the **connection to the server**
- Two streams, an **ObjectOutputStream** and an **ObjectInputStream**
- A **Thread** that runs using AbstractClient's **run** method
- Two variables storing the **host** and **port** of the server

THE SERVER SIDE

- **AbstractServer** to listen to new connections
  - Must implement **handleMessageFromClient**
  - Could overwrite hooks
- **ConnectionToClient** threads that handle connections to clients



USING ABSTRACTSERVER AND CONNECTIONTOCLIENT

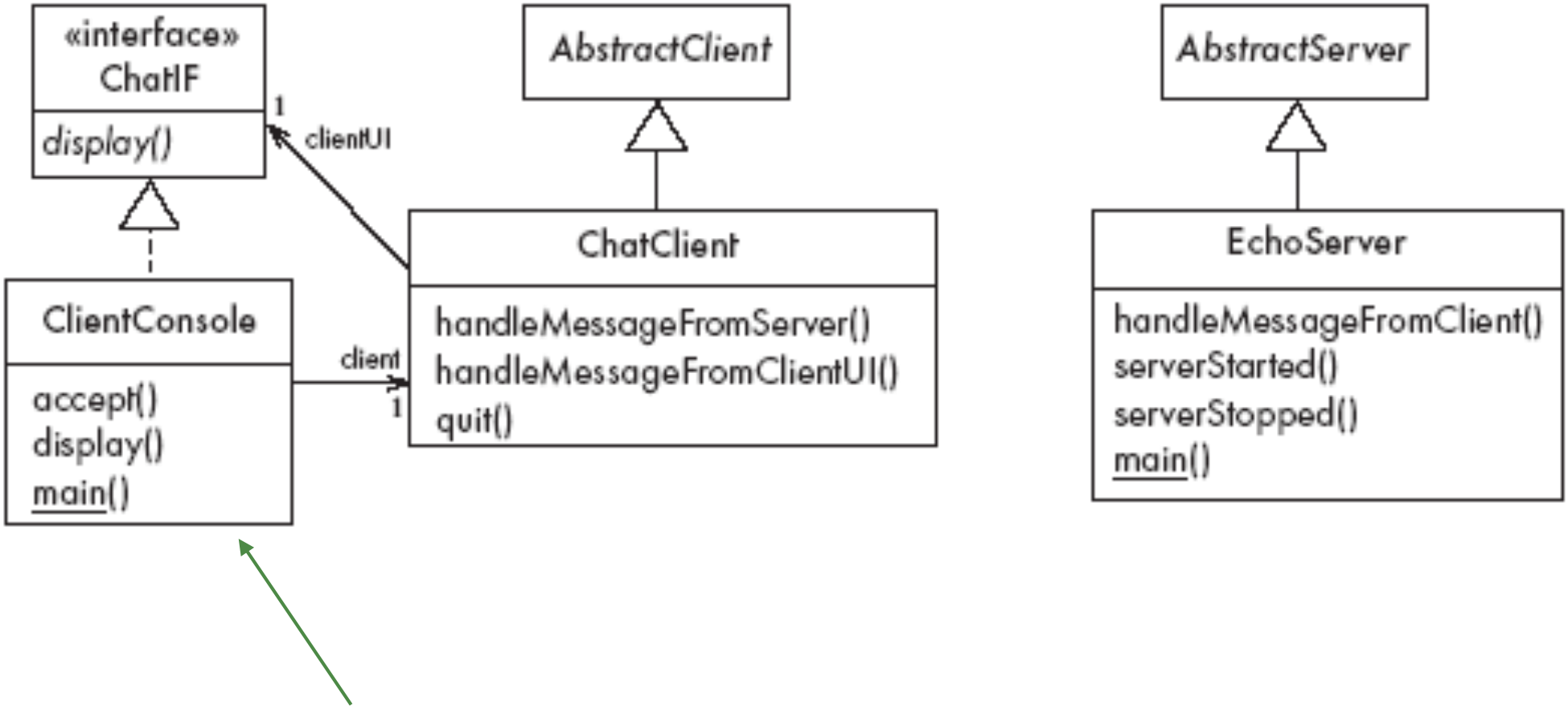
- Create a subclass of **AbstractServer**
- Implement the slot method **handleMessageFromClient**
- Write code that:
  - **Creates an instance** of the subclass of AbstractServer
  - Calls the **listen** method
  - Sends messages to clients, using:
    - the **getClientConnections** and **sendToClient** service methods
    - or **sendToAllClients**

Shhhh, Internals of AbstractServer and ConnectionToClient

- The **setInfo** and **getInfo** methods make use of a Java class called **HashMap**
- Many methods in the server side are **synchronized**
- The **collection** of instances of **ConnectionToClient** is stored using a special class called **ThreadGroup**
- The server must **pause** from listening every 500ms to see if the **stopListening** method has been called
  - if not, then it resumes listening immediately

AN INSTANT MESSAGING APPLICATION:

SIMPLECHAT


«interface»
ChatIF
display()
1
clientUI
AbstractClient
AbstractServer
ChatClient
handleMessageFromServer()
handleMessageFromClientUI()
quit()
EchoServer
handleMessageFromClient()
serverStarted()
serverStopped()
main()
ClientConsole
accept()
display()
main()
client
1
Can eventually be replaced by ClientGUI

ECHOSERVER (SUBCLASS OF ABSTRACTSERVER)

- Starts a new instance and listens for client connections
- Three callbacks just print out the message to the server

```
public void handleMessageFromClient
(Object msg, ConnectionToClient client) {

   System.out.println(
     "Message received: "
     + msg + " from " + client);

    this.sendToAllClients(msg);
}
```

CHATCLIENT (SUBCLASS OF ABSTRACTCLIENT)

- When the client program starts, it creates instances of two classes:
  - ChatClient (here)
  - ClientConsole (next)

```
public void handleMessageFromServer
   (Object msg) {
   clientUI.display(msg.toString());
}
```

CLIENTCONSOLE

- User interface class that implements **ChatIF**
- Display outputs to the console
- Accepts user input by calling accept in its run method
- Sends all user by calling its handleMessageFromClientUI

```
public void handleMessageFromClientUI
  (String message) {
  try {
    sendToServer(message);
  }
  catch(IOException e) {
    clientUI.display (
       "Could not send message. " +
       "Terminating client.");
    quit();
  }
}
```

RISKS WHEN REUSING

TECHNOLOGY

POOR QUALITY REUSABLE COMPONENTS

- Ensure that the developers of the reusable technology:
  - follow good software engineering practices
  - are willing to provide active support

Compatibility not maintained

- Avoid obscure features
- Only re-use technology that others are also re-using

RISKS WHEN DEVELOPING

REUSABLE TECHNOLOGY

Investment uncertainty

- Plan the development of the reusable technology, just as if it was a product for a client

The 'not invented here syndrome'

- Build confidence in the reusable technology by:
  - Guaranteeing support
  - Ensuring it is of high quality
  - Responding to the needs of its users

Competition

- The reusable technology must be as useful and as high quality as possible

Divergence

- (tendency of various groups to change technology in different ways)
- Design it to be general enough, test it and review it in advance

RISKS WHEN ADOPTING A

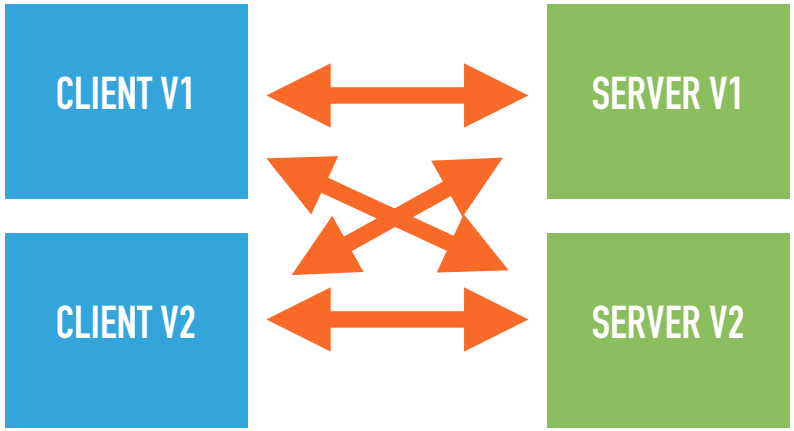
CLIENT-SERVER APPROACH

Security

- Security is a big problem with no perfect solutions: consider the use of encryption, firewalls, ...

NEED FOR ADAPTIVE MAINTENANCE

- Ensure that all software is
  - forward proof and
  - backward compatible
- with other versions of clients and servers



COMMUNICATIONS

PROTOCOLS

protocol noun

pro·to·col | \ ˈprō-tə-ˌkȯl, -ˌkōl, -ˌkäl, -kəl\

Definition of *protocol*

1 : an original draft, minute, or record of a document or transaction

2 **a** : a preliminary memorandum often formulated and signed by diplomatic negotiators as a basis for a final convention or treaty

b : the records or minutes of a diplomatic conference or congress that show officially the agreements arrived at by the negotiators

3 **a** : a code prescribing strict adherence to correct etiquette and precedence (as in diplomatic exchange and in the military services)

// a breach of *protocol*

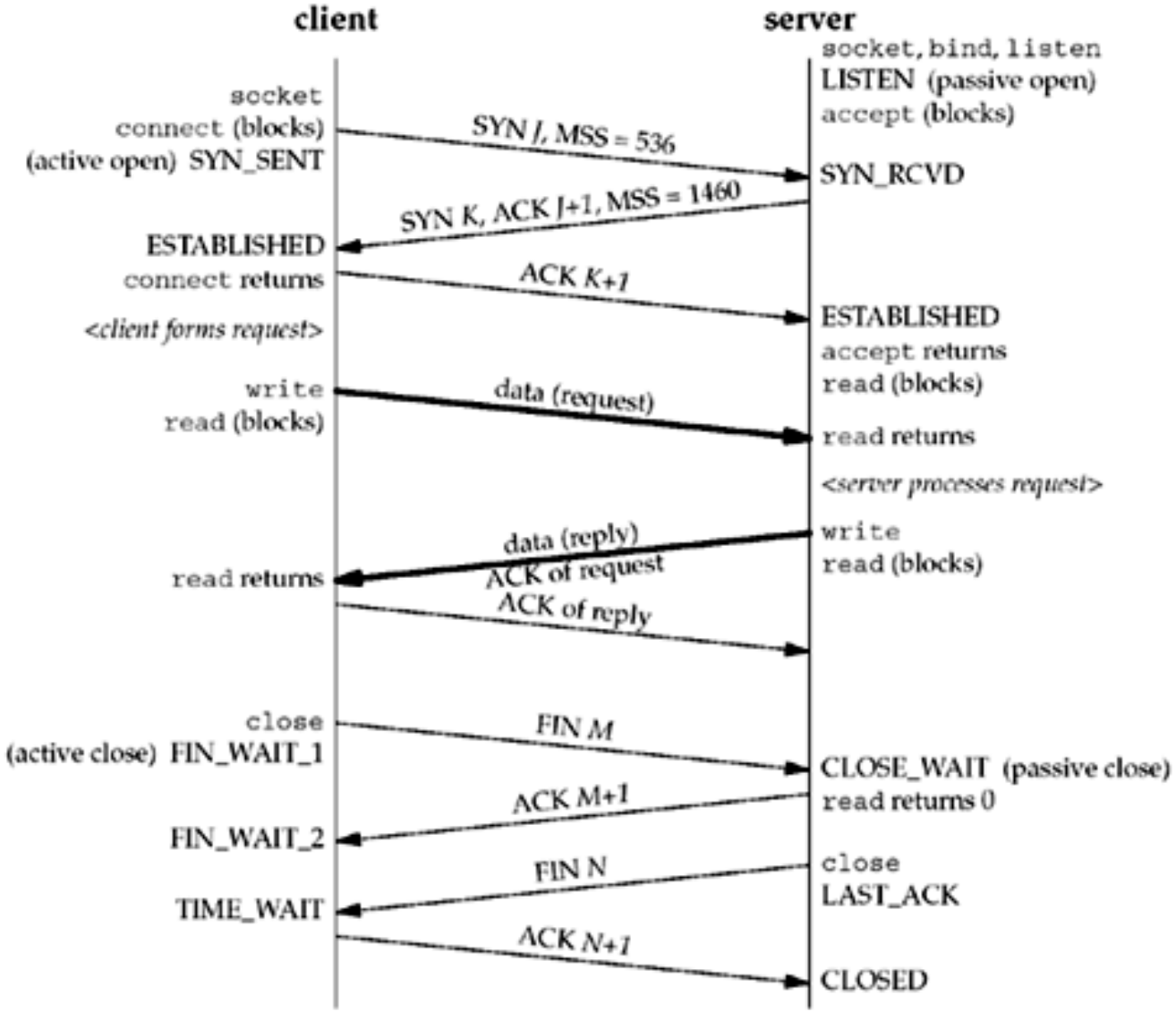
b : a set of conventions governing the treatment and especially the formatting of data in an electronic communications system

// network *protocols*

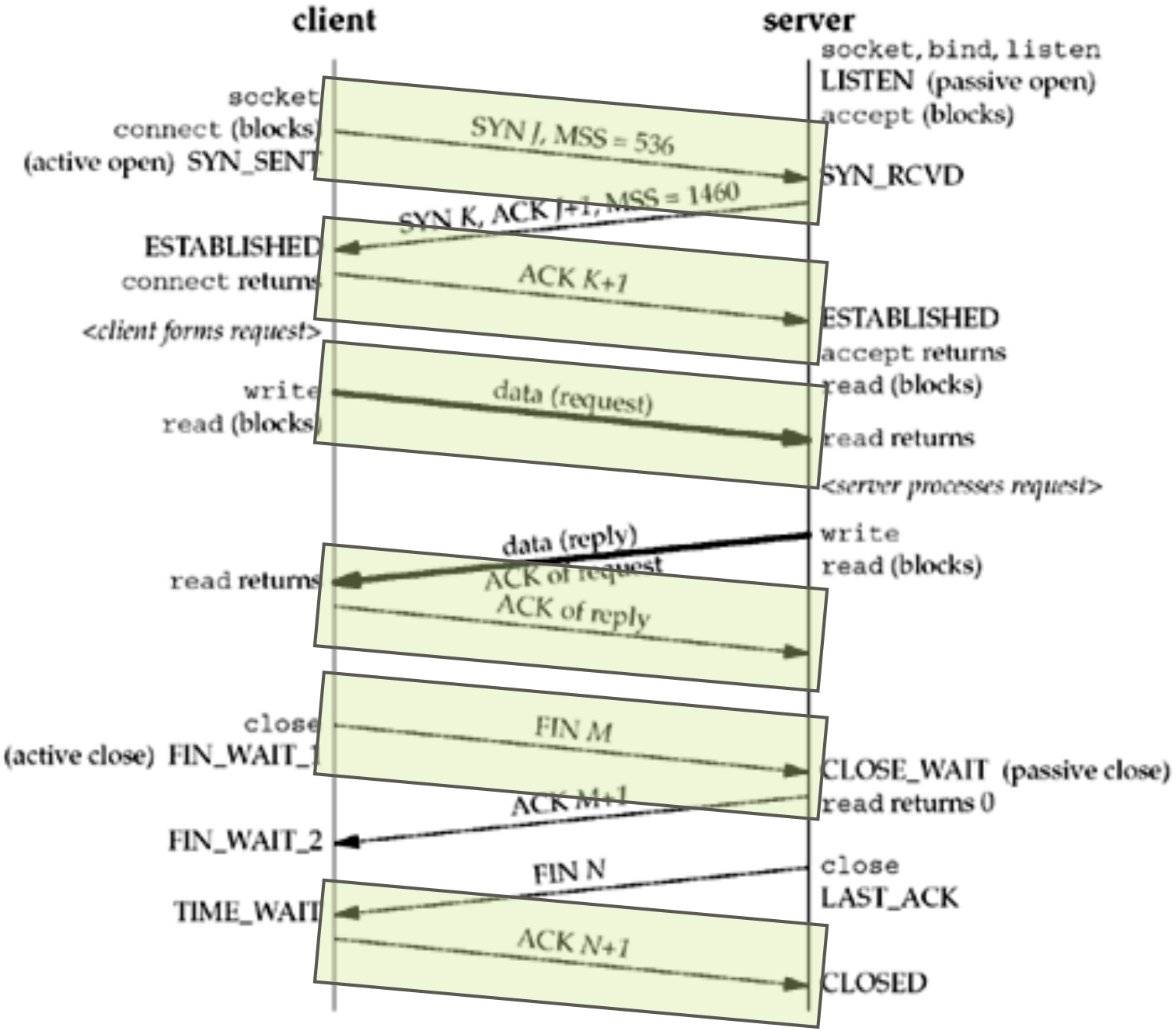
c : CONVENTION sense 3a,b

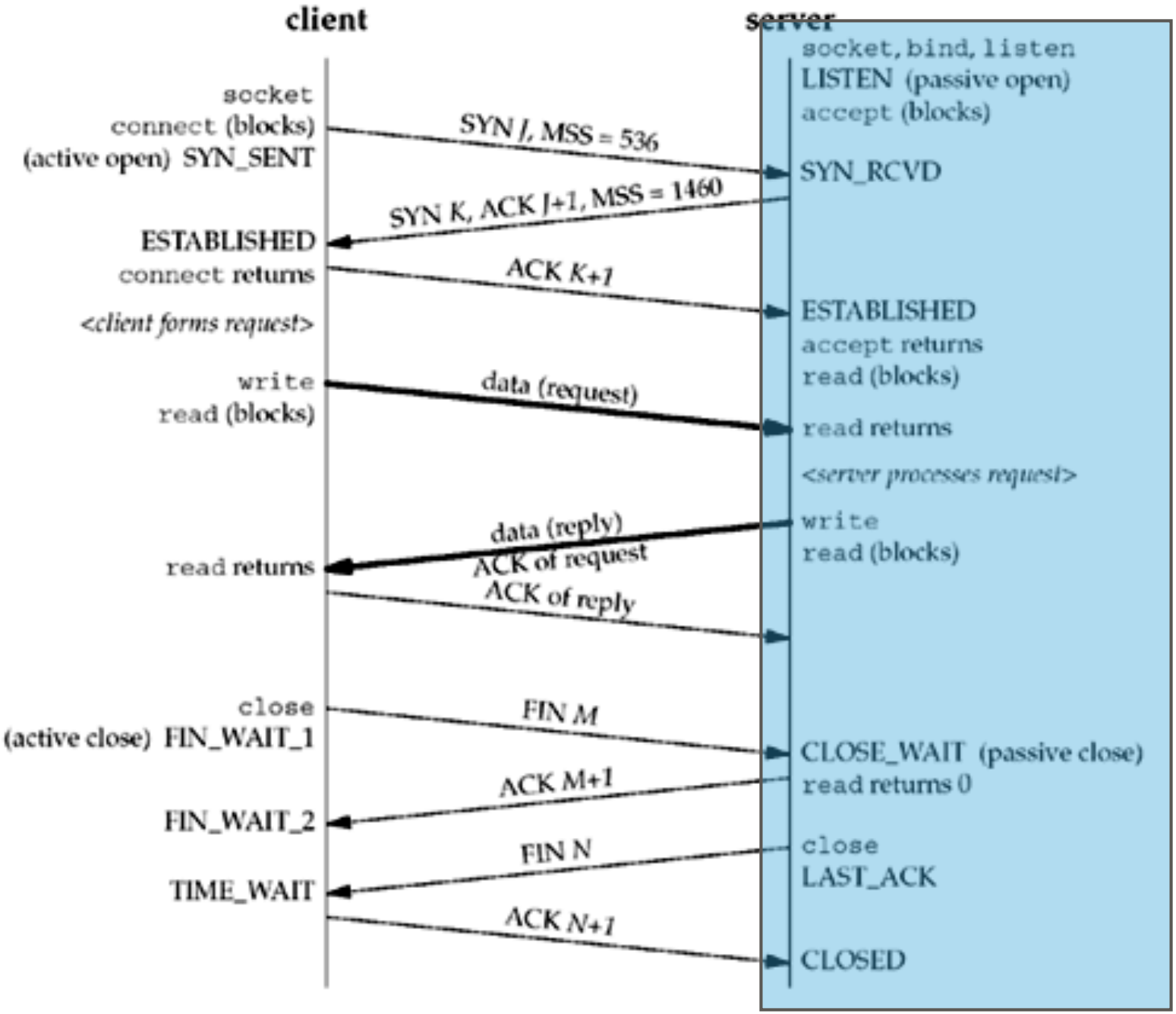
4 : a detailed plan of a scientific or medical experiment, treatment, or procedure

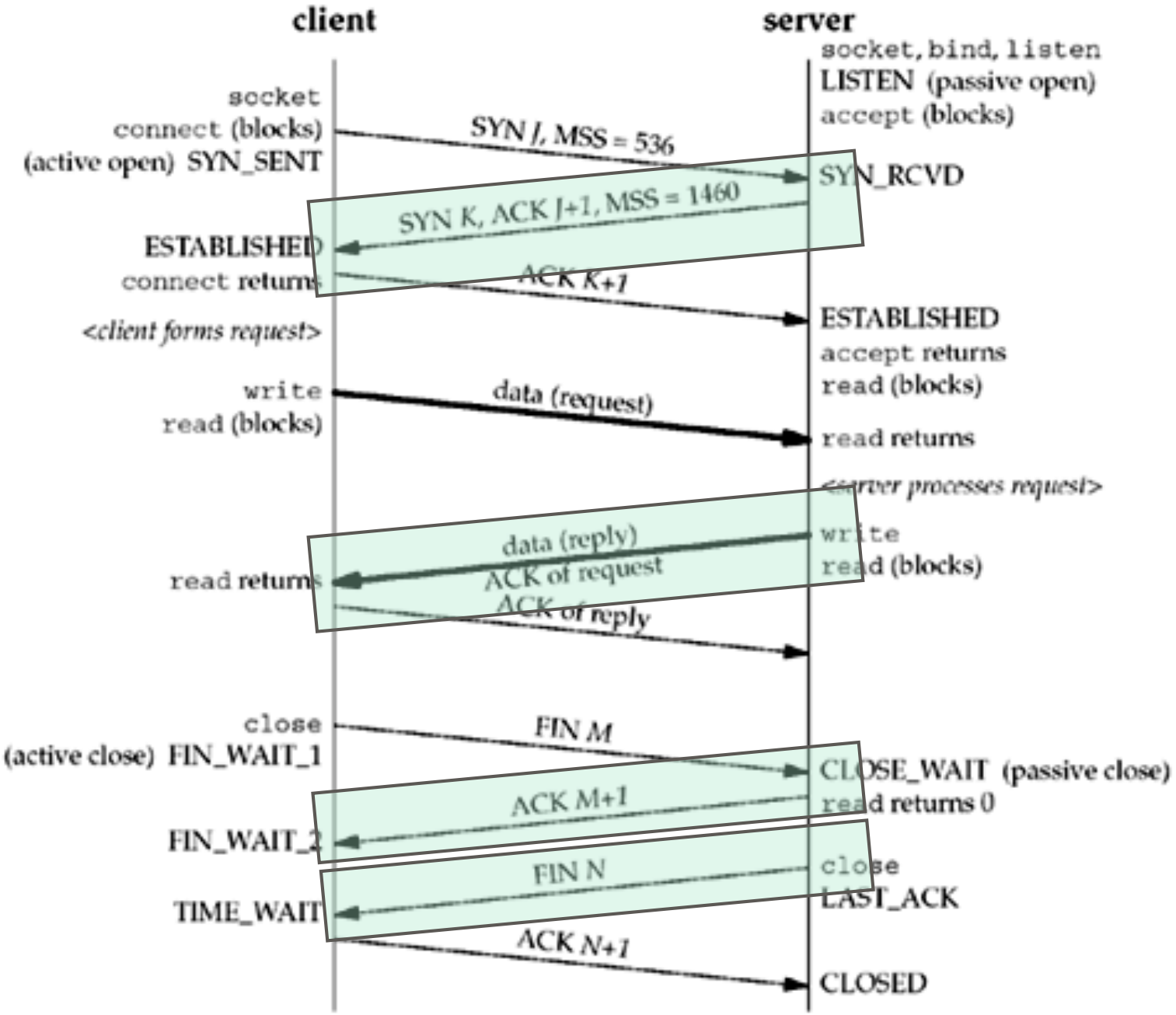
https://www.merriam-webster.com/dictionary/protocol

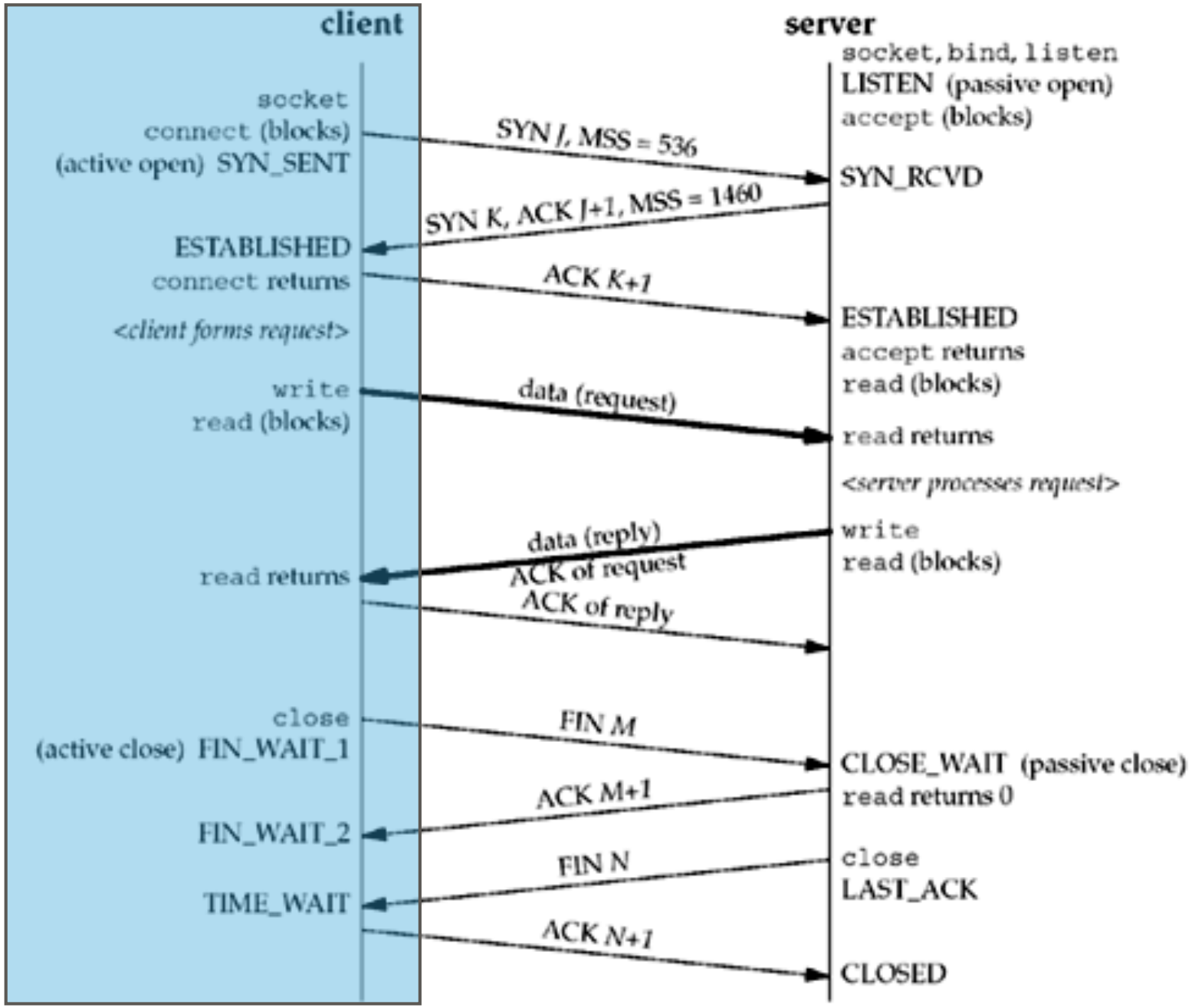


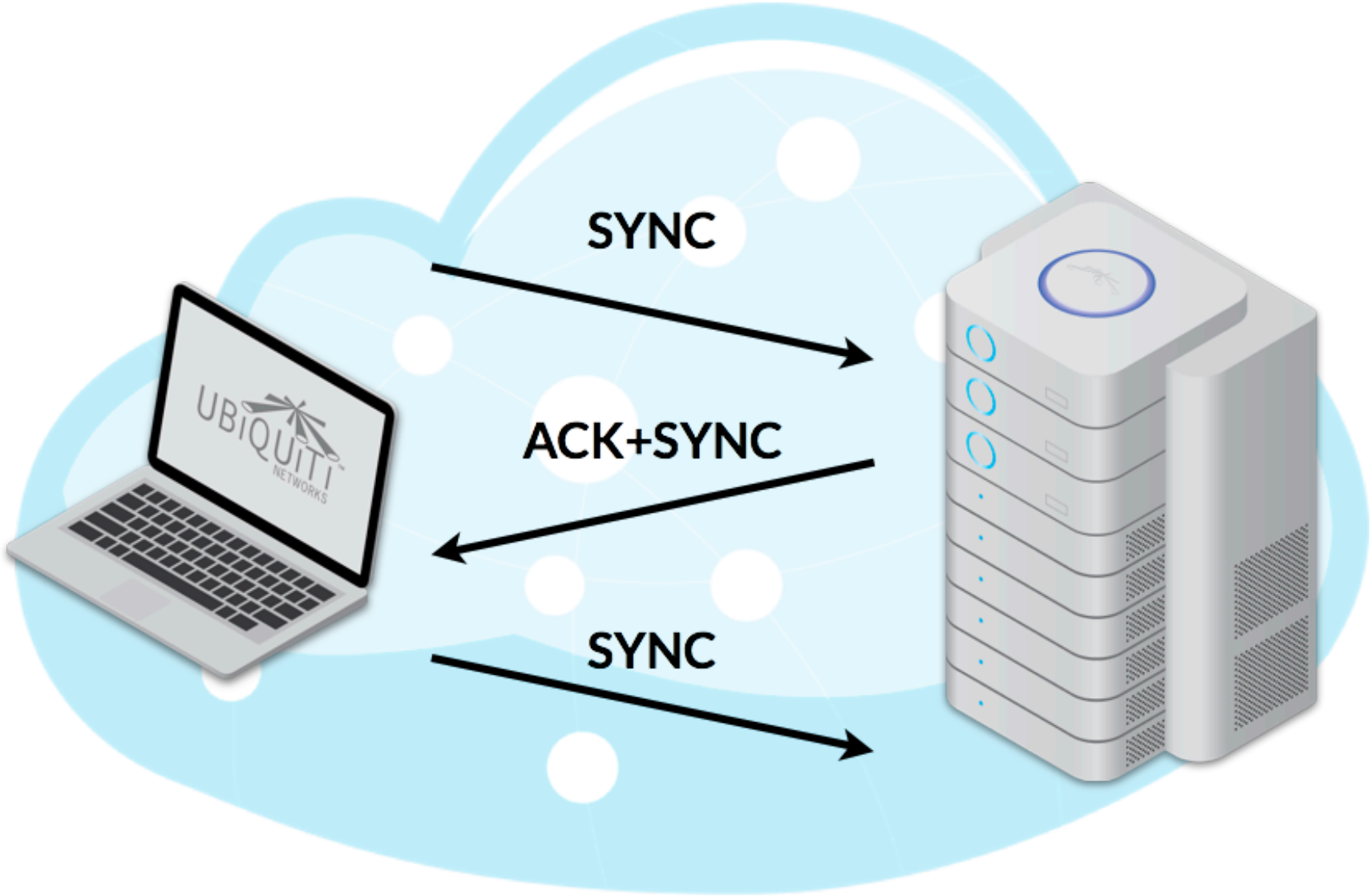
https://notes.shichao.io/unp/ch5/#tcp-echo-server-str_echo-function


client
server
socket, bind, listen
LISTEN (passive open)
accept (blocks)
socket
connect (blocks)
(active open) SYN_SENT
SYN J, MSS = 536
SYN_RCVD
SYN K, ACK J+1, MSS = 1460
ESTABLISHED
connect returns
ACK K+1
ESTABLISHED
<client forms request>
accept returns
read (blocks)
write
read (blocks)
data (request)
read returns
<server processes request>
write
read (blocks)
data (reply)
read returns
ACK of request
ACK of reply
close
(active close) FIN_WAIT_1
FIN M
CLOSE_WAIT (passive close)
read returns 0
ACK M+1
FIN_WAIT_2
close
LAST_ACK
FIN N
TIME_WAIT
ACK N+1
CLOSED

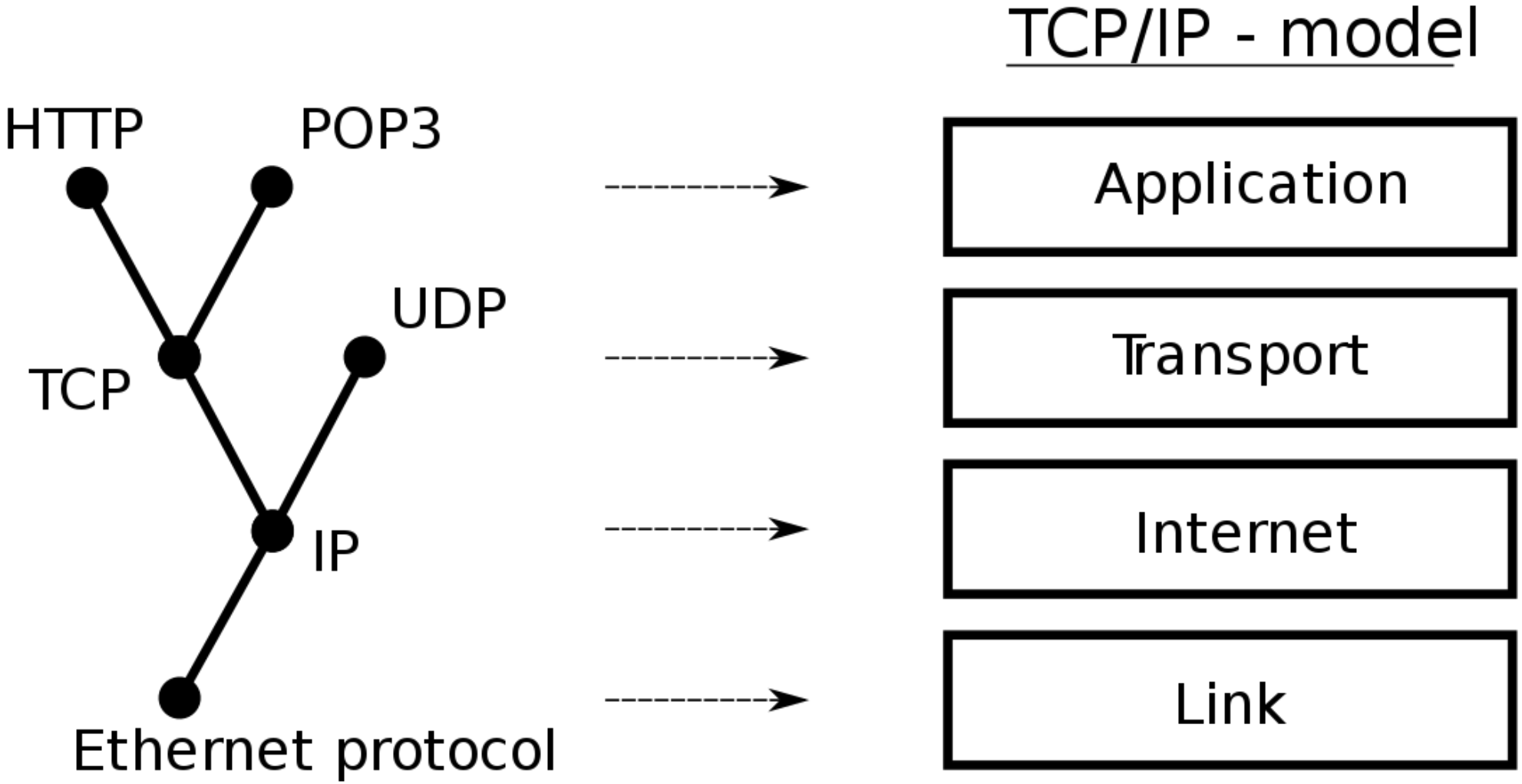

client
server
socket, bind, listen
LISTEN (passive open)
accept (blocks)
socket
connect (blocks)
(active open) SYN_SENT
SYN J, MSS = 536
SYN_RCVD
SYN K, ACK J+1, MSS = 1460
ESTABLISHED
connect returns
ACK K+1
<client forms request>
ESTABLISHED
accept returns
write
read (blocks)
data (request)
read returns
<server processes request>
write
read (blocks)
data (reply)
ACK of request
read returns
ACK of reply
close
(active close) FIN_WAIT_1
FIN M
CLOSE_WAIT (passive close)
read returns 0
ACK M+1
FIN_WAIT_2
close
LAST_ACK
FIN N
TIME_WAIT
ACK N+1
CLOSED


client
server
socket, bind, listen
LISTEN (passive open)
accept (blocks)
socket
connect (blocks)
(active open) SYN_SENT
SYN J, MSS = 536
SYN_RCVD
SYN K, ACK J+1, MSS = 1460
ESTABLISHED
connect returns
ACK K+1
ESTABLISHED
<client forms request>
accept returns
read (blocks)
write
read (blocks)
data (request)
read returns
<server processes request>
write
read (blocks)
data (reply)
ACK of request
read returns
ACK of reply
close
(active close) FIN_WAIT_1
FIN M
CLOSE_WAIT (passive close)
ACK M+1
read returns 0
FIN_WAIT_2
FIN N
close
TIME_WAIT
LAST_ACK
ACK N+1
CLOSED


client
server
socket, bind, listen
LISTEN (passive open)
accept (blocks)
socket
connect (blocks)
(active open) SYN_SENT
SYN J, MSS = 536
SYN_RCVD
SYN K, ACK J+1, MSS = 1460
ESTABLISHED
connect returns
ACK K+1
<client forms request>
ESTABLISHED
accept returns
write
read (blocks)
data (request)
read returns
<server processes request>
write
read (blocks)
data (reply)
ACK of request
read returns
ACK of reply
close
(active close) FIN_WAIT_1
FIN M
CLOSE_WAIT (passive close)
ACK M+1
read returns 0
FIN_WAIT_2
FIN N
close
LAST_ACK
TIME_WAIT
ACK N+1
CLOSED



http://titantesting.com/kb/249

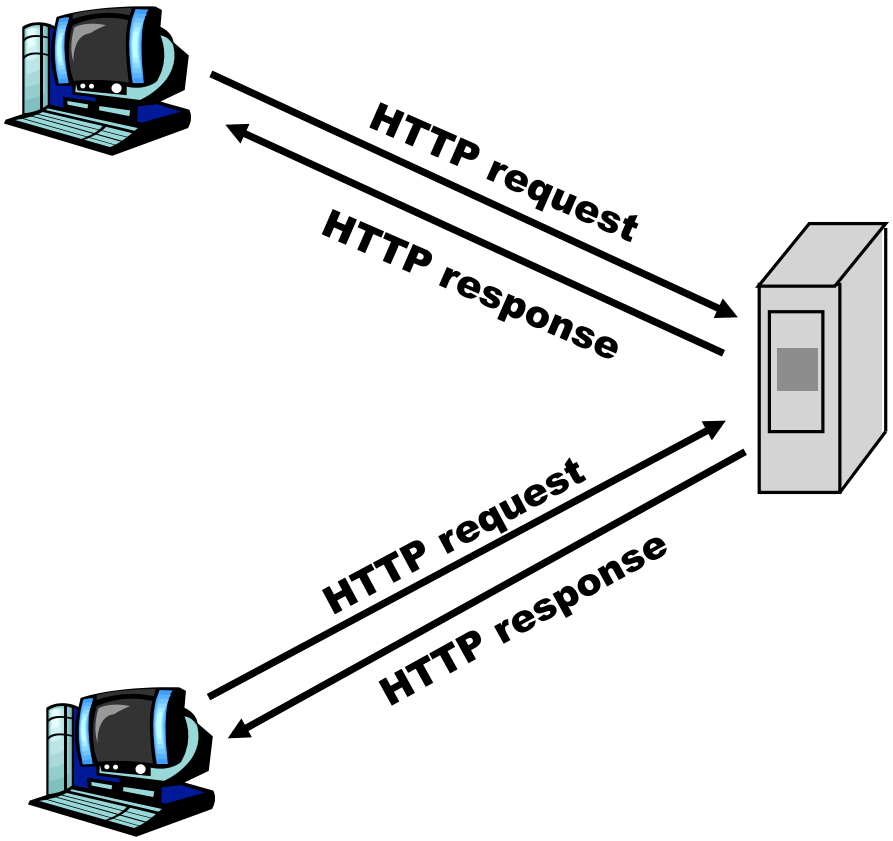


https://en.wikipedia.org/wiki/Communication_protocol

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. ... Hello, would you like to hear a TCP joke?"

MANY PROTOCOLS FOR CLIENT/SERVER COMMUNICATIONS

- HTTP
  - HTTP 1.0: RFC 1945
  - HTTP 1.1: RFC 2068
  - HTTP/2: RFC 7540 (published Mai 2015)



REFERENCES

- https://en.wikipedia.org/wiki/Communication_protocol