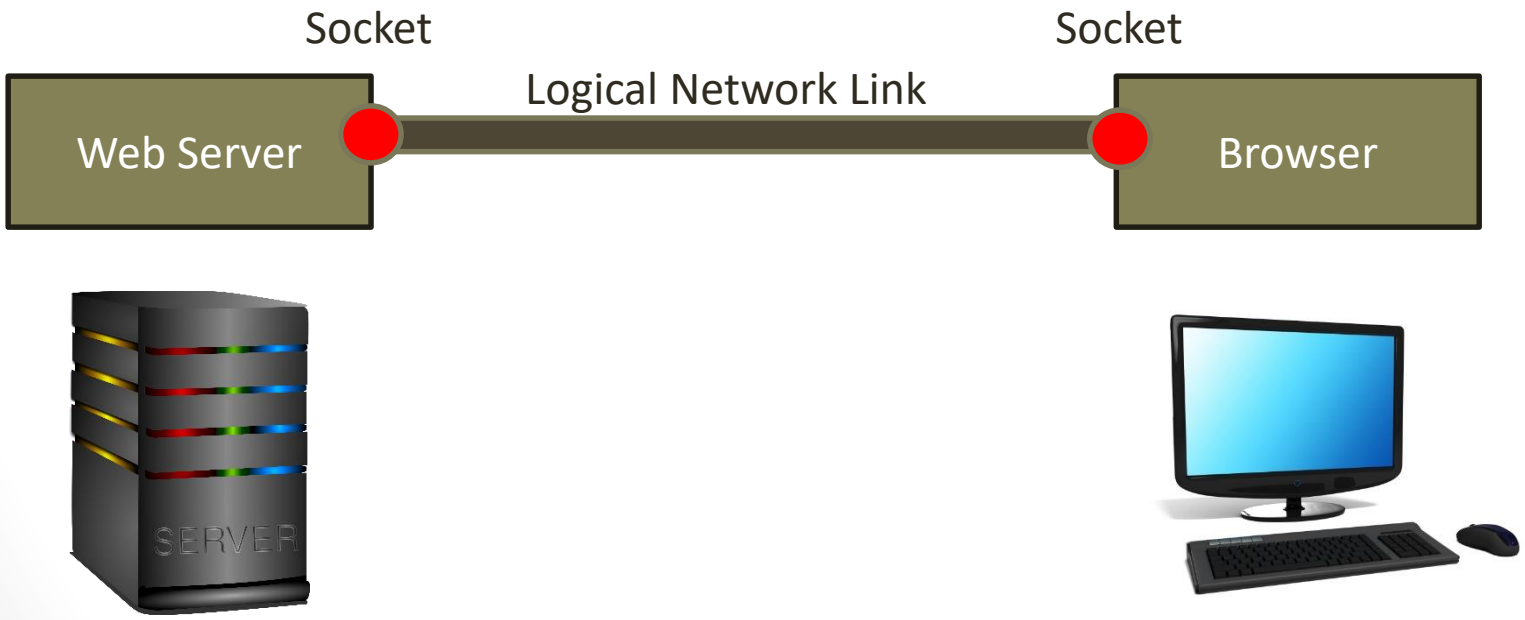
Socket programming with Java

Περιεχόμενα

- Internet Sockets:
  - Stream Sockets
  - Datagram Sockets
- Internet Sockets σε Java
- Threads

Τι είναι τα Internet sockets?

- Τελικό σημείο (end-point) μίας bi-directional ροής επικοινωνίας σε ένα IP-based δίκτυο υπολογιστών, e.g. Internet.



Sockets

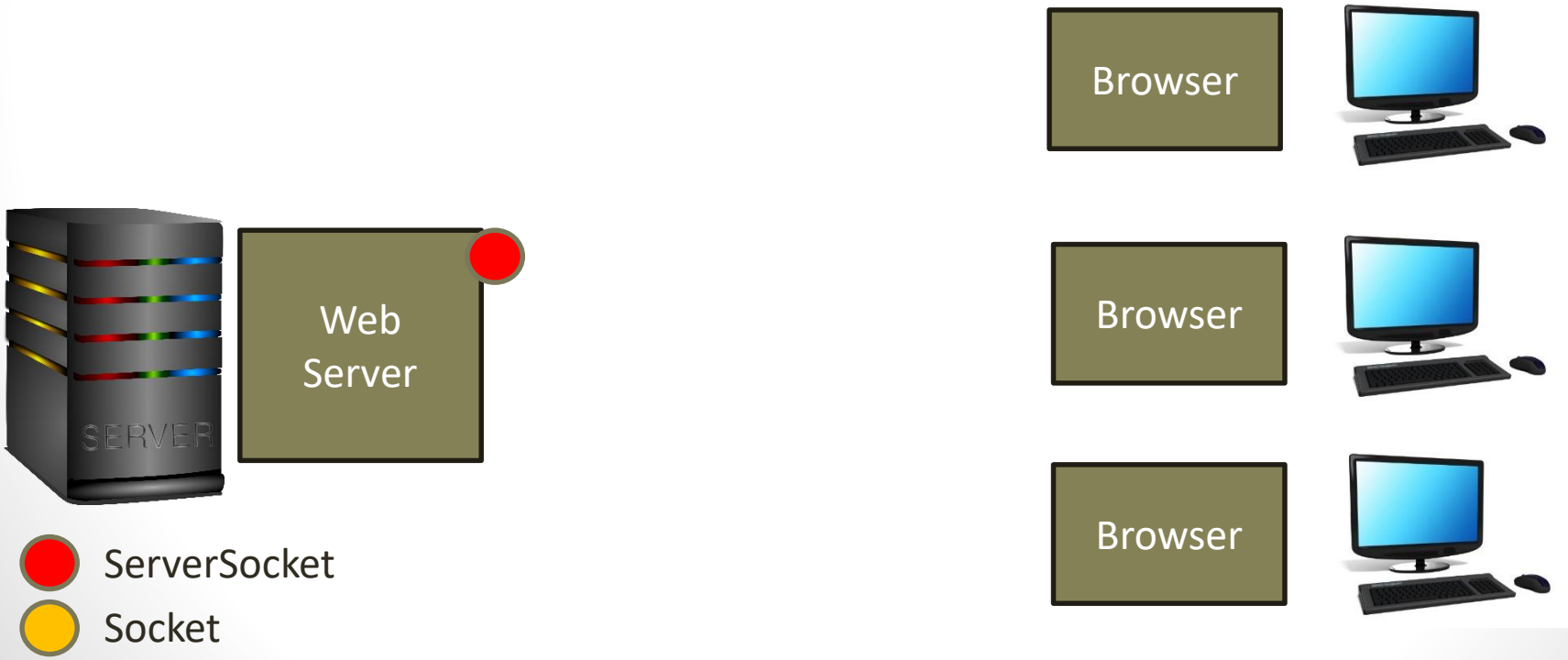
- Socket number
  - *Μοναδικός* ακέραιος αριθμός που χαρακτηρίζει ένα συγκεκριμένο socket σε ένα λειτουργικό σύστημα
- Socket address
  - Συνδυασμός της IP διεύθυνσης ενός Η/Υ και μίας πόρτας που αντιστοιχεί σε μία συγκεκριμένη εφαρμογή
    - Local IP address and port: 192.168.1.2:8000

Είδη Internet sockets

- Stream sockets
  - χρησιμοποιούν το πρωτόκολλο TCP:
    - εγγυημένη παράδοση μηνυμάτων και στη σωστή σειρά
    - overhead λόγω εγκαθίδρυσης σύνδεσης, μηνυμάτων επιβεβαίωσης κλπ.
- Datagram sockets
  - χρησιμοποιούν το πρωτόκολλο UDP:
    - δεν παρέχουν καμία εγγύηση
- Raw sockets
  - Παρακάμπτουν το επίπεδο μεταφοράς
  - Χρησιμοποιούνται κυρίως για πρωτόκολλα δρομολόγησης, π.χ. OSPF

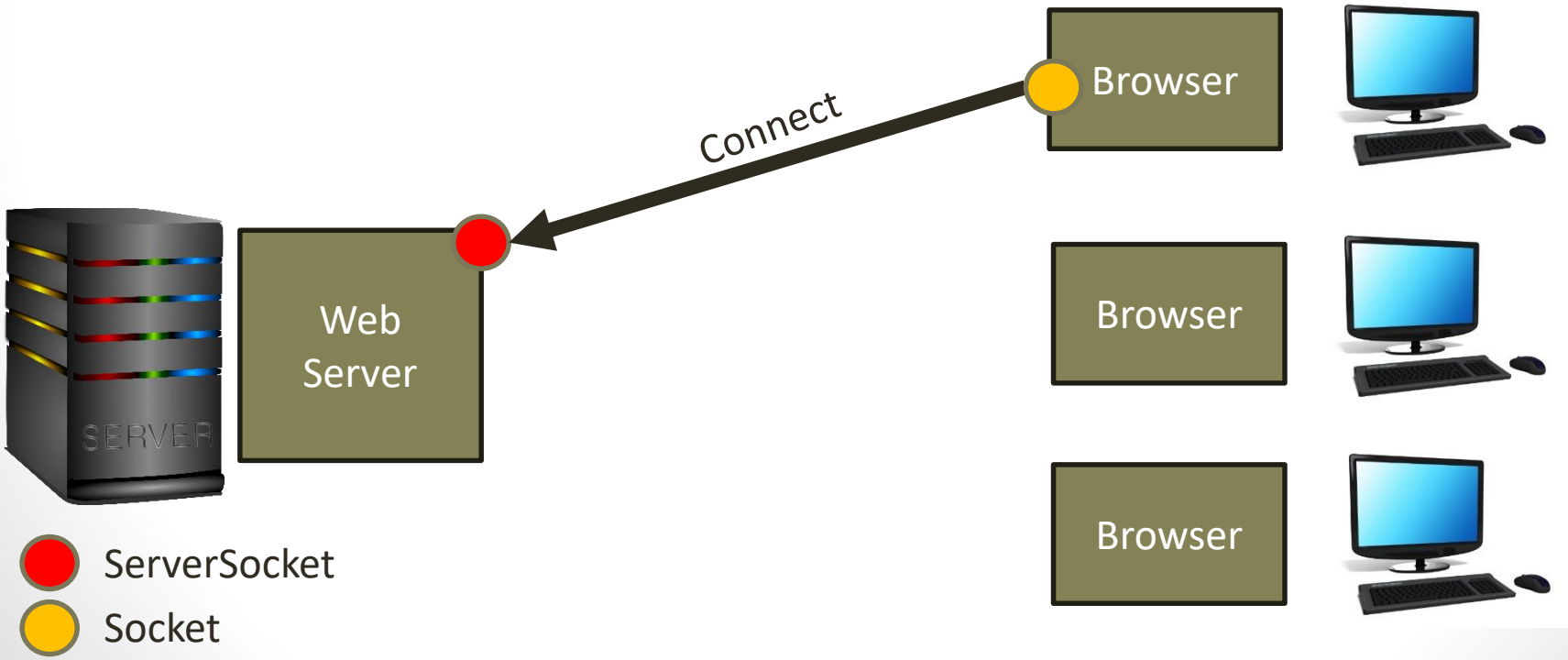
Stream sockets

- Ο TCP Server φτιάχνει ένα socket ειδικού τύπου (ServerSocket) και το βάζει να ακούει σε μία πόρτα (πχ 80) στην οποία περιμένει συνδέσεις από τους clients.



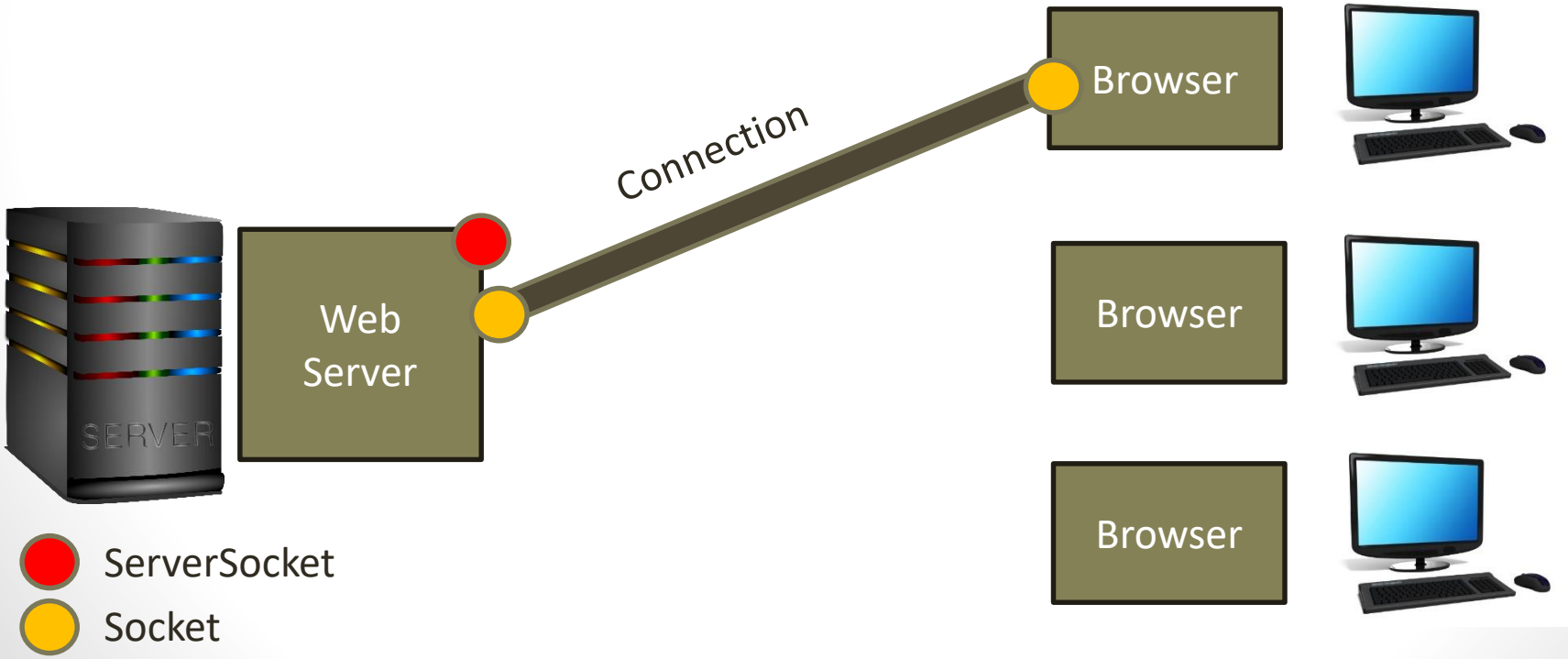
Stream sockets

- Ένας client φτιάχνει ένα socket με τυχαία τοπική θύρα και συνδέεται στον Server χρησιμοποιώντας την διεύθυνση ip και την πόρτα του Server.



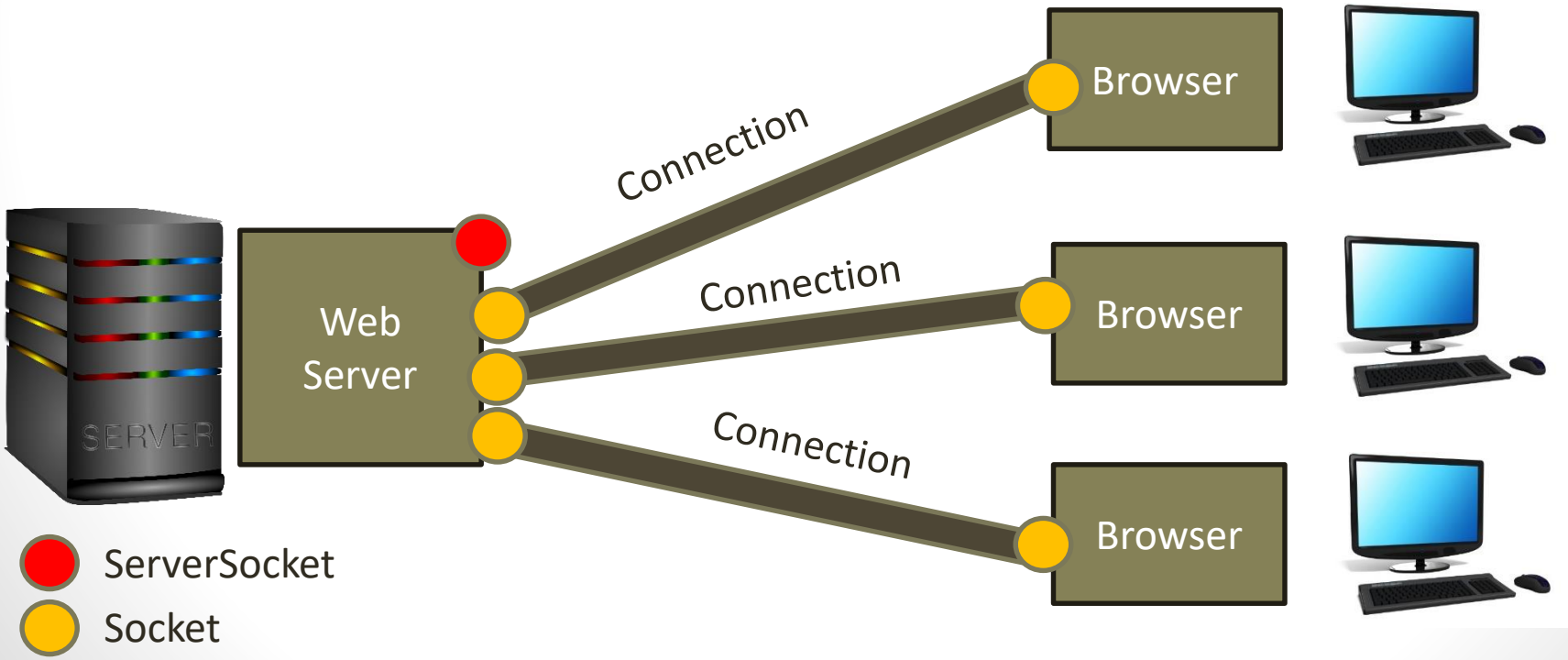
Stream sockets

- Ο TCP Server αποδέχεται την αίτηση σύνδεσης και δημιουργεί ένα νέο socket με το οποίο θα επικοινωνεί με τον συγκεκριμένο client



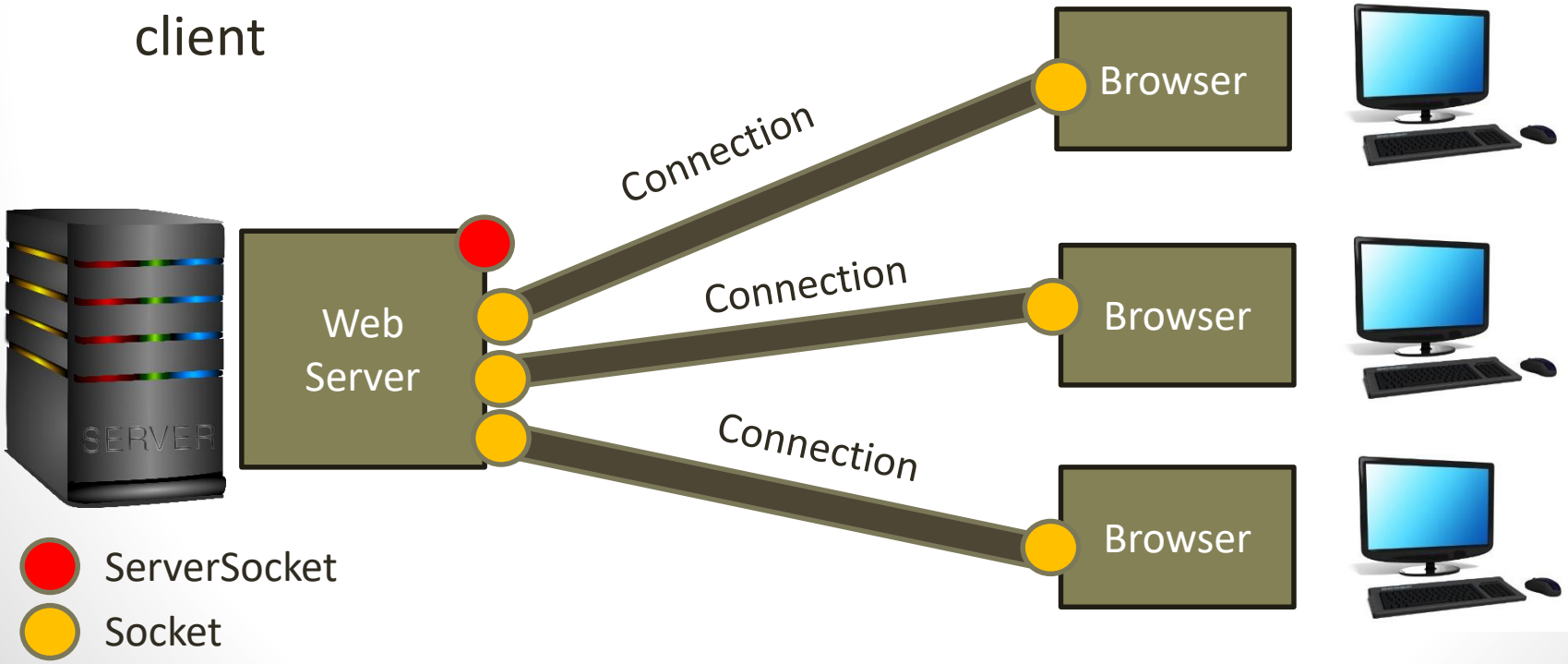
Stream sockets

- Ο TCP Server συνεχίζει να ακούει στο ServerSocket και δημιουργεί ένα νέο socket για κάθε νέο client.



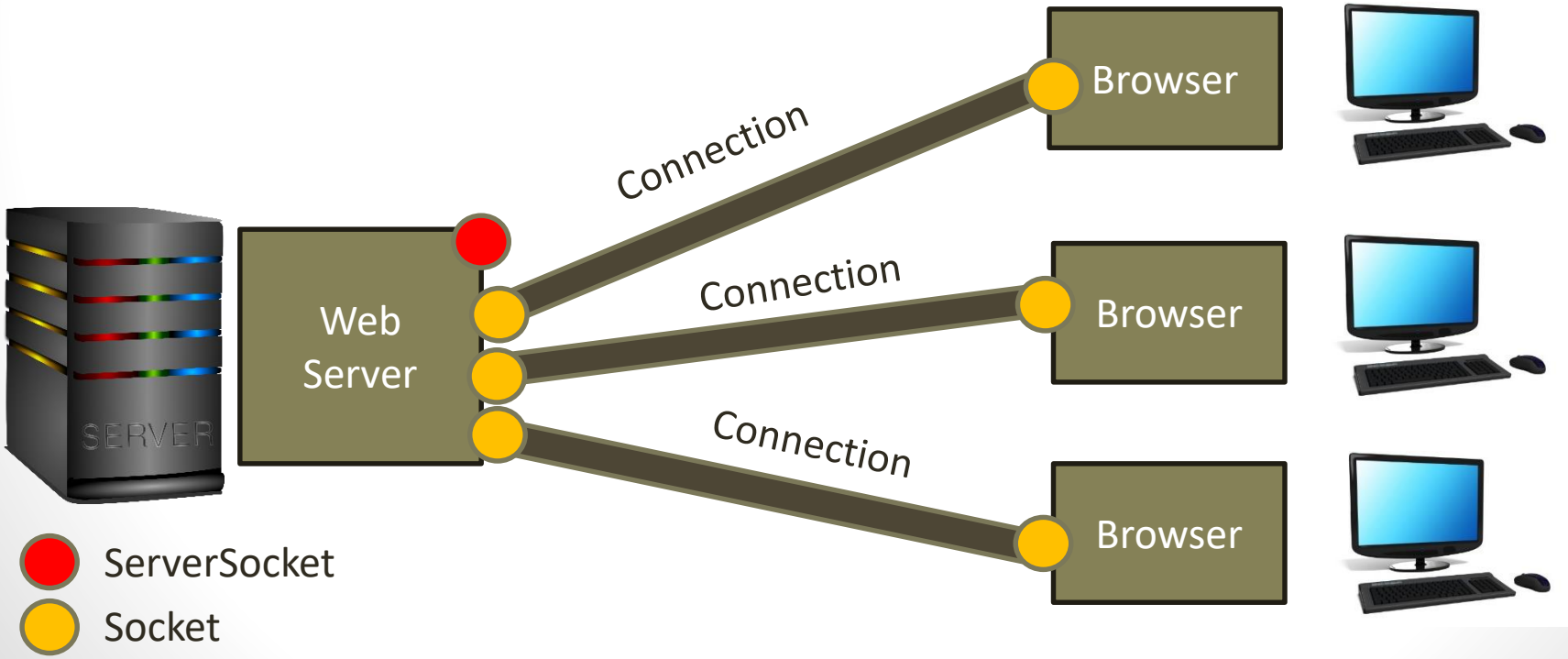
Stream sockets

- Τα sockets του WebServer έχουν την ίδια socket address (ip / port) καθώς δημιουργήθηκαν από το ίδιο ServerSocket.
- Για να χαρακτηρίσουμε μοναδικά μια σύνδεση χρειαζόμαστε και το socket address του αντίστοιχου client



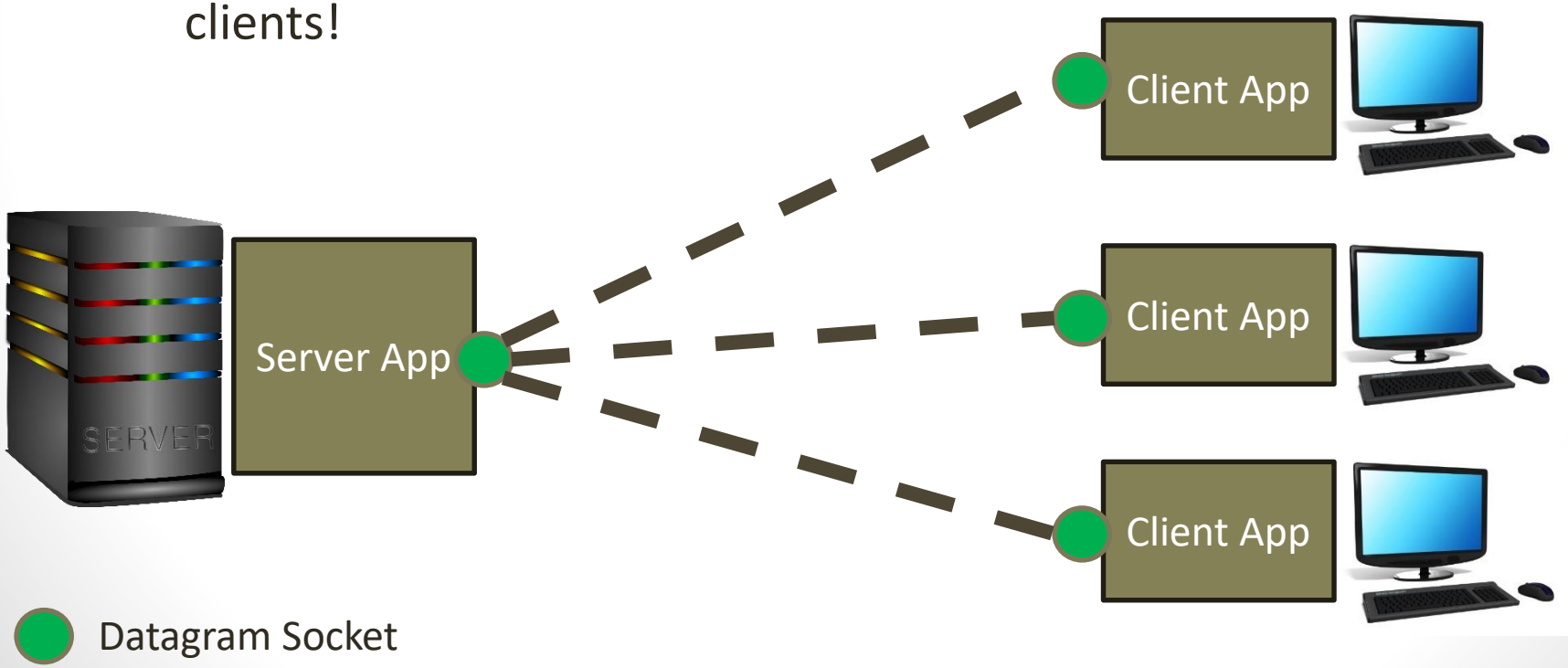
Stream sockets

- Κάθε socket χρησιμοποιείται από τον κάτοχο του για να επικοινωνήσει με το άλλο άκρο.



Datagram sockets

- UDP server εξυπηρετεί όλα τα εισερχόμενα πακέτα δεδομένων από όλους τους clients ακολουθιακά μέσω του ίδιου socket
  - Δεν δημιουργούνται νέα sockets για την εξυπηρέτηση των clients!



Socket Programming σε JAVA

- Stream Sockets
  - java.net.ServerSocket
  - java.net.Socket
- Datagram Sockets
  - java.net.DatagramSocket
  - java.net.DatagramPacket

STREAM SOCKETS ΣΕ JAVA (TCP)

Κλάση java.net.ServerSocket

- public ServerSocket(int server_port)
  - throws IOException
- public ServerSocket(int server_port, int queue_length)
  - throws IOException
- Όταν κατασκευάζουμε ένα αντικείμενο ServerSocket ουσιαστικά φτιάχνουμε ένα Server Socket που ακούει στην πόρτα *server_port*.
- Η παράμετρος *queue_length* δηλώνει τον μέγιστο αριθμό clients που μπορούν να έχουν κάνει αίτηση σύνδεσης αλλά δεν έχουν γίνει ακόμα accepted.

Κλάση java.net.ServerSocket

- public Socket accept():
  - Η μέθοδος αυτή μπλοκάρει την εκτέλεση του προγράμματος μέχρις ότου κάποιος client να συνδεθεί στο Server Socket. Όταν συμβεί αυτό επιστρέφει ένα αντικείμενο java.net.Socket μέσω του οποίου επικοινωνεί ο Server με τον client που μόλις συνδέθηκε.

Κλάση java.net.Socket

- Η κλάση java.net.Socket επιτρέπει την δημιουργία αντικειμένων που εκτελούν και τις τέσσερις λειτουργίες των sockets.
  - Σύνδεση με απομακρυσμένα μηχανήματα
  - Αποστολή δεδομένων
  - Λήψη δεδομένων
  - Κλείσιμο σύνδεσης

Δημιουργία ενός Socket

- Η κατασκευή ενός Socket από τον Server γίνεται μέσω της μεθόδου accept() του ServerSocket.

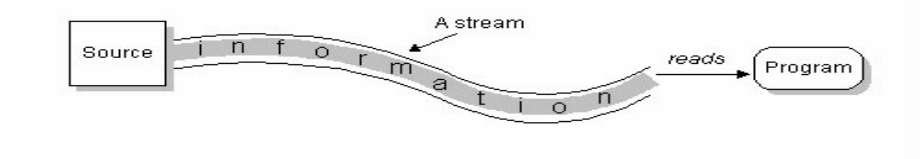
- Οι Clients δημιουργούν Sockets μέσω των κατασκευαστών προκειμένου να συνδεθούν στον server.

Δημιουργία ενός Socket

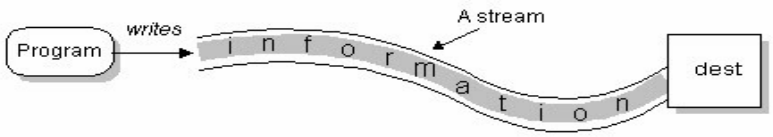
- public Socket(String host, int server_port)
- public Socket(InetAddress host, int server_port)
- Throws:
  - UnknownHostException
  - IOException
- Ο *host* μπορεί να προσδιορίζεται είτε από μια συμβολοσειρά (π.χ. “aueb.gr") είτε ως ένα αντικείμενο InetAddress.
- Η θύρα πρέπει να είναι ένας ακέραιος μεταξύ 1 και 65535.
  - Socket socket = new Socket (“www.cs.aueb.gr”, 80)

Αποστολή και Λήψη δεδομένων

- Πώς επικοινωνούν οι απομακρυσμένες διεργασίες?
  - Μέσω streams
- **InputStream inStr = sock.getInputStream();**
  - Για να διαβάζουμε ό,τι στέλνεται από την άλλη πλευρά της σύνδεσης



- **OutputStream outStr = sock.getOutputStream();**
  - Για να στείλουμε στην άλλη πλευρά της σύνδεσης



Echo – Server

```
ServerSocket server_socket = new ServerSocket(500);
while (true) {
    // Wait for connection
    Socket connection = server_socket.accept();

    // get Input and Output streams
    ObjectOutputStream out = new
        ObjectOutputStream(connection.getOutputStream());
    ObjectInputStream in = new
        ObjectInputStream(connection.getInputStream());

    //get message and send it back
    String message = (String) in.readObject();
    out.writeObject(message);
    out.flush();
    connection.close();
}
```

Echo – Client

```
//Connect to server
Socket connection = new Socket("192.168.1.5", 500);

// get Input and Output streams
ObjectOutputStream out = new
        ObjectOutputStream(connection.getOutputStream());
ObjectInputStream in = new
        ObjectInputStream(connection.getInputStream());

//send message get it back and print it
out.writeObject("hello TCP");
out.flush();
String message = (String) in.readObject();
System.out.println(message);
connection.close();
```

Interface Serializable

- Όταν μια κλάση κάνει implement το interface Serializable δηλώνει ότι μπορεί να σειριακοποιηθεί.
- Με αυτόν τον τρόπο μπορούμε να περάσουμε αντικείμενα αυτής της κλάσης ως όρισμα στις μεθόδους writeObject() και readObject() των ObjectOutputStream και ObjectInputStream αντίστοιχα.
- Όταν φτιάχνουμε πολύπλοκα πρωτόκολλα εφαρμογής συνήθως βολεύει να έχουμε μια custom κλάση Message η οποία συμπεριλαμβάνει όλων των ειδών τα πεδία που μπορεί να θέλουμε να στείλουμε μαζί με ένα πεδίο int message_type που δηλώνει τον τύπο του μηνύματος.
- Έτσι μπορούμε στη μεριά του παραλήπτη να εκτελούμε ενέργειες βάση του message_type.

DATAGRAM SOCKETS ΣΕ JAVA (UDP)

Datagram sockets σε JAVA

- Η Java υποστηρίζει datagram sockets μέσω των κλάσεων:
  - java.net.DatagramSocket
  - java.net.DatagramPacket
- Ένα datagram socket χρησιμοποιείται για την αποστολή και λήψη πακέτων (αντικείμενα DatagramPacket).

java.net.DatagramSocket

- Δεν υπάρχει διάκριση ανάμεσα σε ένα UDP socket και UDP server socket.
- Σε αντίθεση με τα TCP sockets, ένα DatagramSocket μπορεί να στείλει σε πολλές και διαφορετικές διευθύνσεις.
- Η διεύθυνση προορισμού των δεδομένων αποθηκεύεται στο πακέτο και όχι στο socket.

java.net.DatagramSocket

- public DatagramSocket()
- public DatagramSocket(int port)
- Όταν κατασκευάζουμε ένα αντικείμενο DatagramSocket ουσιαστικά φτιάχνουμε ένα Datagram Socket που ακούει στην πόρτα *port*.
- Σε περίπτωση που δεν δώσουμε το όρισμα *port* η πόρτα επιλέγεται τυχαία.

java.net.DatagramPacket

- public DatagramPacket(byte[] buf, int length)
- public DatagramSocket(byte[] buf, int length,
  InnetAddress address, int port)
- Αντικείμενα της κλάσης DatagramPacket αντιπροσωπεύουν UDP πακέτα που στέλνονται / λαμβάνονται μέσω ενός DatagramSocket

Αποστολή Datagram Packet

- Για να στείλουμε δεδομένα:
  - Μετατρέπουμε τα δεδομένα σε πίνακα byte.
  - Περνάμε στον κατασκευαστή DatagramPacket():
    - τον πίνακα byte
    - το μήκος των δεδομένων του πίνακα (τις περισσότερες φορές αυτό θα είναι το μήκος του πίνακα)
    - InetAddress
    - port
  - Τέλος στέλνουμε το πακέτο με τη μέθοδο send() ενός DatagramSocket αντικειμένου.

Αποστολή Datagram Packet

```
InetAddress host = InetAddess.getByName(“aueb.gr");
int port = 500;
String s = "My first UDP Packet";
byte[] b = s.getBytes();
DatagramPacket dp = new
        DatagramPacket(b, b.length, host, port);
DatagramSocket socket = new DatagramSocket();
socket.send(dp);
```

Παραλαβή Datagram Packet

- Για να λάβουμε δεδομένα
  - Δημιουργούμε έναν κενό πίνακα byte
  - Περνάμε αυτόν τον πίνακα byte και το μήκος του στον κατασκευαστή DatagramPacket()
  - Γεμίζουμε το DatagramPacket περνώντας το ως όρισμα στη μέθοδο receive() ενός DatagramSocket αντικειμένου.
  - Τέλος λαμβάνουμε τα δεδομένα και το μήκος τους με τις μεθόδους getData() και getLength() του DatagramPacket.

Παραλαβή Datagram Packet

```
byte[] rec_data = new byte[512];
DatagramPacket rec_packet = new
        DatagramPacket(rec_data, rec_data .length);

DatagramSocket socket = new DatagramSocket(500);
socket.receive(rec_packet);

// rec_packet.getData();
// rec_packet.getLength();
// rec_packet.getAddress();
// rec_packet.getPort();
```

Echo – Server

```
//create a DatagramSocket and bind it to port 500
DatagramSocket socket = new DatagramSocket(500);
while (true) {
    //receive data
    byte[] rec_data = new byte[512];
    DatagramPacket rec_packet = new DatagramPacket(rec_data , rec_data .length)
    socket.receive(rec_packet);

    //send them back
    InetAddress address = rec_packet.getAddress();
    int port = rec_packet.getPort();
    DatagramPacket packet = new DatagramPacket(rec_packet.getData(),
                                        rec_packet.getLength(), address, port);
    socket.send(packet)
}
```

Echo – Client

```
//create a DatagramSocket and bind it to a random port
DatagramSocket socket = new DatagramSocket();
byte[] data =  “Hello World”;
InetAddress address = InetAddress.getByName(“192.168.1.5”);
DatagramPacket packet = new DatagramPacket(data, data.length, address, 500);
socket.send(packet);

byte[] rec_data = new byte[512];
DatagramPacket rec_packet = new DatagramPacket(rec_data, rec_data .length);
socket.receive(rec_packet);

System.out.println(new String(rec_packet.getData(), 0, rec_packet.getLength()));
socket.close();
```

JAVA THREADS

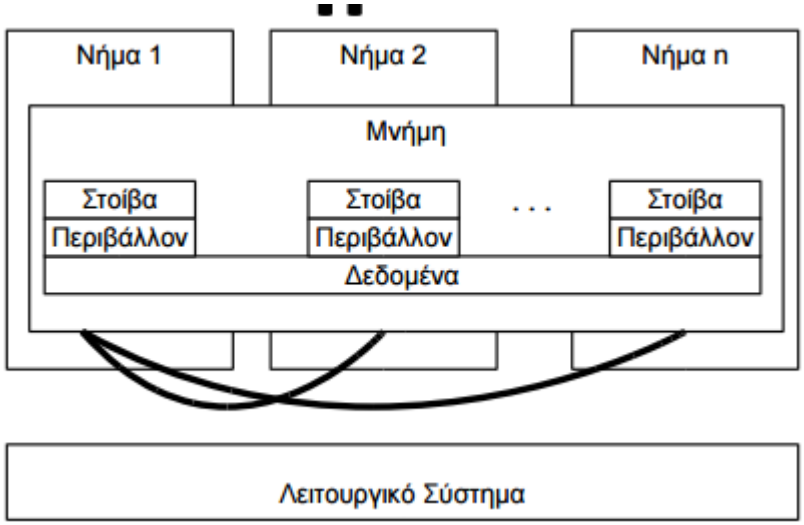
File – Server

```
ServerSocket server_socket = new ServerSocket(500);
while (true) {
    // Wait for connection
    Socket connection = server_socket .accept();
    //**SEND BIG FILE TO CLIENT**
    connection.close();
}
```

- Το πρόβλημα με τον προηγούμενο κώδικα είναι ότι ο server εξυπηρετεί τους clients ακολουθιακά. Αυτό έχει ως αποτέλεσμα κάποιοι clients να περιμένουν πολύ.
- Λυση: threads

Java Threads

- Νήμα: μια ροή ελέγχου σε μια διεργασία
  - Επιτρέπονται πολλά νήματα ανά διεργασία
  - Όλα τα νήματα μοιράζονται την ίδια μνήμη
  - Χωριστή στοίβα, καταχωρητές, μετρητής
  - Ψευδοταυτόχρονη / παράλληλη εκτέλεση νημάτων
  - Three thread states: running, ready, blocked



Γιατί χρειαζόμαστε threads?

- Παράλληλη επεξεργασία
  - πχ. web server:
  1. Threads για παράλληλη αποστολή αρχείων σε κάθε client
  2. Threads για παράλληλη εξυπηρέτηση πολλών clients
- Γρήγορη επικοινωνία μέσω shared memory
- Χρήση όλων των διαθέσιμων επεξεργαστών
- Επίδοση – λιγότερος χρόνος για create()/destroy() ενός thread σε σχέση με process

Πως κατασκευάζουμε threads?

- Δύο τρόποι:
  - Επέκταση της κλάση Thread
  - Υλοποιούμε τη διεπαφή Runnable και περνάμε το αντικείμενο μέσω του κατασκευαστή Thread.
- Σε κάθε περίπτωση πρέπει να υλοποιήσουμε τη μέθοδο **run()** η οποία θα περιέχει τον κώδικα που θα εκτελέσει το thread.
- Αφού κατασκευάσουμε ένα thread το εκτελούμε με την μέθοδο **start()**.

Παραδείγμα Thread

```
public class MyThread extends Thread {
        String input;

        public MyThread(String input) { this.input = input;}

        public void run() {
                for (int i = 0; i < 10; i++) {
                System.out.println(i + ": " + input);
                try {
                        sleep((int) (Math.random() * 500));
                } catch (InterruptedException e) {}
        }
}
```

Στην Main:

```
(new MyThread("Computer")).start();
(new MyThread("Science ")).start();
```

Output

0: Computer

0: Science

1: Science

1: Computer

2: Computer

2: Science

3: Computer

4: Computer

3: Science

4: Science

File – Server Revisited

```
ServerSocket server_socket = new ServerSocket(4321);
while (true) {
    // Wait for connection
    Socket connection = server_socket .accept();
    Thread t  = new ServeClientThread(connection);
    t.start();
}
```

- Τώρα κάθε client εξυπηρετείται παράλληλα από διαφορετικό thread.

Συγχρονισμός

- Πολλές φορές θέλουμε να απαγορεύουμε σε δύο ή περισσότερα threads να προσπελάζουν το ίδιο αντικείμενο ταυτόχρονα γιατί μπορεί να οδηγήσει σε απρόσμενα αποτελέσματα.

Παράδειγμα

```
Class Counter{
        private int c = 0
        public void increment(){ c++; }
        public void decrement(){ c--; }
}
```

- Το c++ εκτελείται ως εξής :
  - Διάβασε την τιμή του c
  - Αύξησε την τιμή αυτή κατά 1
  - Αποθήκευσε το αποτέλεσμα πίσω στο c
- Αντίστοιχα και για το c--

Παράδειγμα

- Counter counter = new Counter();
- Thread A εκτελεί counter.increment();
- Thread B εκτελεί counter.decrement();

- Σενάριο:
  1. Thread A: Διάβασε την τιμή του c -> 0
  2. Thread B: Διάβασε την τιμή του c -> 0
  3. Thread A: Αύξησε την τιμή αυτή κατά 1 -> 1
  4. Thread B: Μείωσε την τιμή αυτή κατά 1 -> -1
  5. Thread A: Αποθήκευσε το αποτέλεσμα πίσω στο c -> 1
  6. Thread B: Αποθήκευσε το αποτέλεσμα πίσω στο c -> -1

- Η τελική τιμή θα είναι -1 ενώ εμείς θα θέλαμε να είναι 0.

Keyword synchronized

- Δε θέλουμε δυο ή περισσότερα νήματα να ανανεώνουν μια κοινή δομή δεδομένων ή μεταβλητή ή αντικείμενο κλπ.
- Οι synchronized μέθοδοι ενός αντικειμένου μπορούν να κληθούν ταυτόχρονα από πολλά threads άλλα εκτελούνται με τη σειρά, όχι ταυτόχρονα.

```
public synchronized void  increment(){ c++; }
Public synchronized void decrement(){ c--; }
```