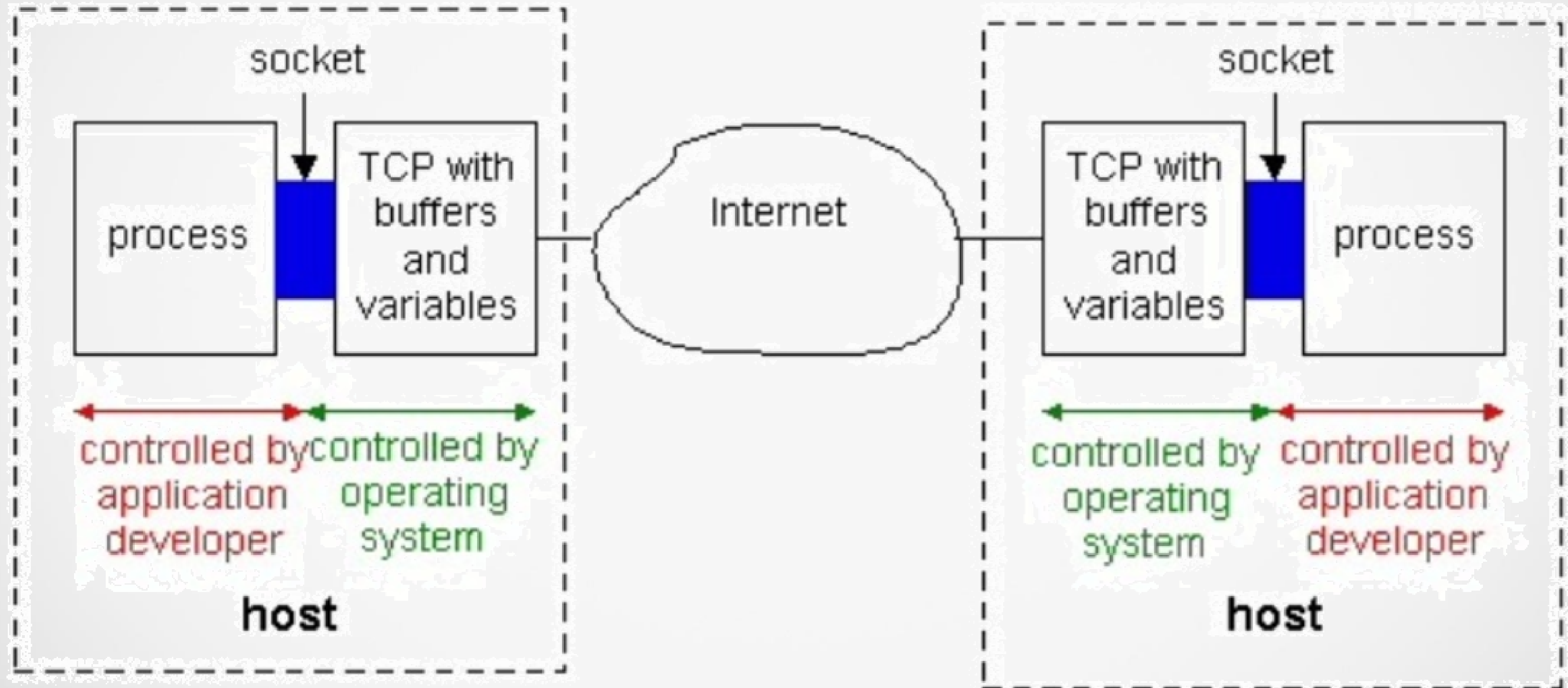
Client-Server – TCP/IP - Kodierung

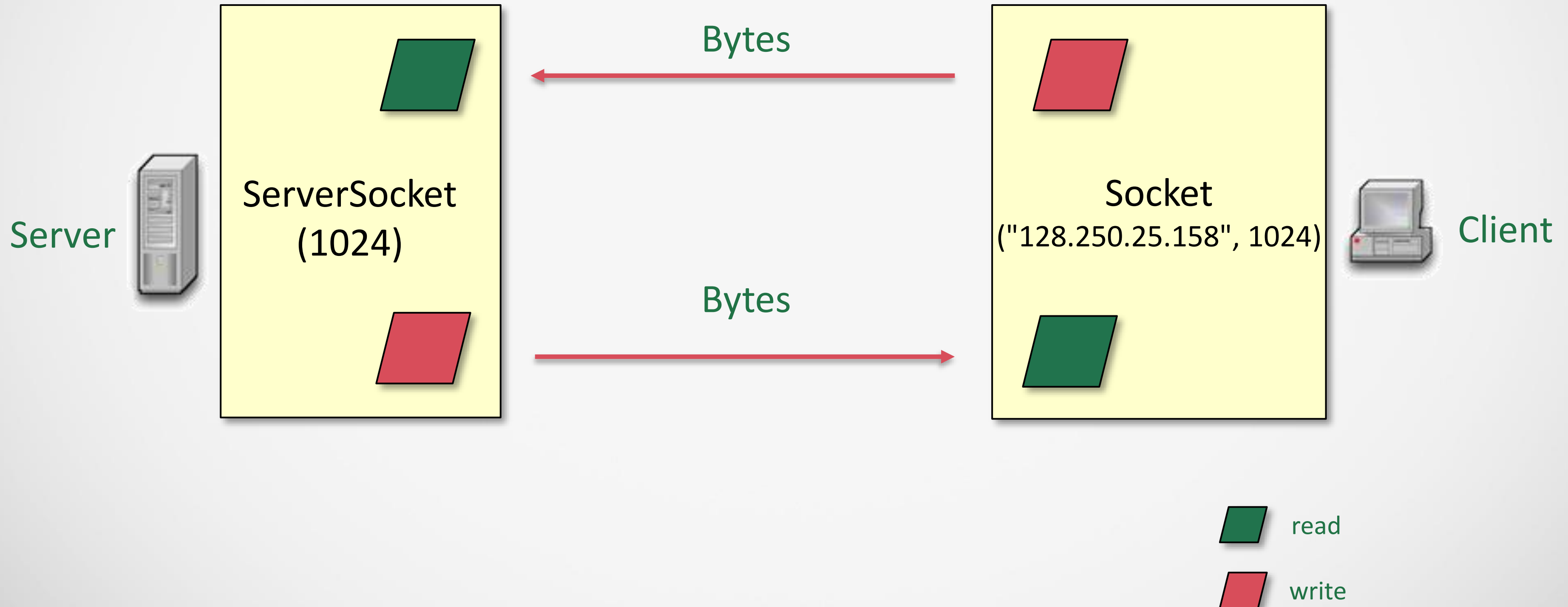
Die Socketklassen

- ✓ Ein Socket (engl. Sockel) ist eine bidirektionale Netzwerk-Kommunikationsschnittstelle, deren Verwaltung das Betriebssystem übernimmt.
- ✓ Die Kombination aus IP-Adresse und einer Portnummer (RFC 793 ,original TCP Spezification)
- ✓ Die Kommunikation findet zwischen einem Server und einem Client über einen definierten Port statt.
- ✓ Java hat vordefinierte Klassen in dem Packet java.net, die die Funktionalität bereitstellen
- ✓ Die beiden Schlüsselklassen für die Erstellung von Server-Client-Programme sind:
  - ✓ ServerSocket
  - ✓ Socket

Prinzip der Socketprogrammierung



Ablauf eines Socket-Programms



Server-Kodierung I

1. `ServerSocket serversocket = new ServerSocket(1024);`

   Öffnet einen ServerSocket um einem bestimmten Port zu nach Anfragen zu lauschen

2. `socket = serversocket.accept();`

   Ist eine Verbindungsanfrage da, wird ein neuer Socket erstellt, durch den der Server die Daten mit dem Client durch Input-und Outputstreams austauscht: accept() gibt den Socket zurück, der die Verbindung zum Server herstellt: Wird erst ausgeführt, wenn ein Client sich an 1024 anmeldet!

3. `fromClient = socket.readLine();`

   Lesen des gesendeten Textes vom Clienten

Server-Kodierung II

4.

```
socket.write(fromClient + "\n");
```

Zurückschreiben des Textes zum Clienten

5.

```
while(!fromClient.equals(".."));
```

Solange der Client nicht „..“ sendet wird auf den nächsten Text gewartet

6.

```
socket.close();
serverSocket.close();
```

Beide Sockets schließen

Client-Kodierung I

1. `socket = new Socket("localhost",1024);`
   - ✓ Erstellt einen neuen Socket
   - ✓ „localhost“ ist der aktuelle PC und kann(muss) durch die IP-Adresse ersetzt werden, wenn die Kommunikation über mehrere PCs geht
2. `if(socket.connect()){`
   - ✓ Überprüfen, ob eine Verbindung zu einem Server auf 1024 zustande gekommen ist!
3. `socket.write(fromClient);`

   Schreiben zum Server
4. `fromServer = socket.readLine();`

   Lesen der gespiegelten Nachricht vom Server

Client-Kodierung II

5.

```
} while(!fromServer.equals(".."));
```

Solange nicht „..“ vom Server gekommen ist, ist der Client bereit für die nächste Eingabe!

6.

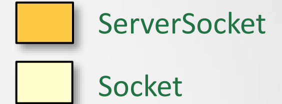
```
socket.close();
```

Schließt den Socket

Arten von Sockets

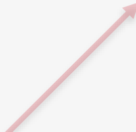
Client

Server



serversocket =
new ServerSocket(1024);

Horcht auf eingehende Verbindungsanfragen von Clients



Fordert die Verbindung an und sorgt für Datenaustausch

socket =
new Socket("localhost",1024);

socket =
serverSocket.accept();

Erstellt die Verbindung und sorgt für Datenaustausch

Einführungsbeispiel Server-Client: Server

```
String fromClient;
ServerSocket serverSocket = new ServerSocket(1024);
Socket socket = serverSocket.accept();
do{
   fromClient = socket.readLine();
   System.out.println(fromClient);
   socket.write(fromClient + "\n");
} while(!fromClient.equals(".."));
socket.close();
serverSocket.close();
```

Erstellt einen ServerSocket gebunden an Port 1024 und wartet in dieser Zeile!

Sobald ein Client sich auf Port 1024 verbindet wird ein Arbeitsocket erzeugt und diese Zeile ausgeführt!

Socket lauscht auf dem Port 1024 auf ankommende Nachrichten (und gibt sie dann aus)

Socket schreibt die gleiche Nachricht dem Clienten zurück

Solange der Client nicht „..“ eingibt wird die Wiederholungsanweisung weiter ausgeführt!

Beide Sockets schließen

->Aufgabe 1

Einführungsbeispiel Server-Client: Client

```
Scanner scanner = new Scanner(System.in);
String fromClient, fromServer;
Socket socket = new Socket("localhost", 1024);
if(socket.connect()){
   do{
      System.out.println(".. = Programmende!");
      fromClient = (scanner.next() + "\n");
      socket.write(fromClient);
      fromServer = socket.readLine();
   } while(!fromServer.equals(".."));
 socket.close();
 }
}
```

ServerSocket an Port 1024;
Senden durch localhost!

Haben wir eine Verbindung?

„Schreiben“ des Textes
auf dem Server

„Lesen“ des Textes
auf dem Server

Solange der Benutzer nicht „..“
sendet

Schließen des Sockets

Abiturklassen Socket und ServerSocket

ServerSocket
- localPort: int - serverSocket: ServerSocket
C ServerSocket(localPort:int) + accept(): Socket + close(): void

Socket
- hostname: String - port: int - socket: Socket - reader: BufferedReader
C Socket(hostname: String, port:int) C Socket(socket: Socket) + close(): void + connect(): boolean + dataAvailable(): int + read(): int + read(b: byte[], len:int): int + readLine(): String + write(b: int): void + write(b: byte[], len:int): void + write(s: String): void

Abiturklassen Socket und ServerSocket - Probleme

```
Socket socket = new Socket ("127.0.0.1", 9999);
```

```
Socket socket = new Socket ("127.0.0.1", 9999);
if (socket.connect()){
   // Connection Action ☺
}
```

Es muss für eine Verbindung connect aufgerufen werden!

Abiturklassen Socket und ServerSocket – Probleme II

try{…………} – catch{………}

Konsequenterweise braucht die Socket-Klasse kein try-catch, die ServerSocket-Klasse aber schon!

Abiturklassen Socket und ServerSocket – Probleme III

```
socket.write(msgFromClient+"\n");
```

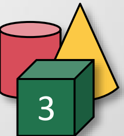
Es gibt keine writeLine analog zu readLine!
Konsequenz: Ohne „\n“ bleibt das Programm beim nächsten „readLine“ „hängen“!

Port Scanner

```
try
  {
   Socket socket = new Socket("localhost",i);
   System.out.print(" Hier laeuft ein Server!");
   socket.close();
  }
  catch(IOException e){
     .........
  }
```

Versuchen zum Server Verbindung herzustellen

Evtl. Fehlermeldung



Ermitteln von IP und Portadresse von Client/Server

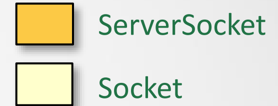
```
System.out.println ("Server kontaktiert: "
            + server.getInetAddress()
            + server.getPort());
```

Parameter des Kommunikationspartners

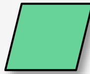
```
System.out.println ("Server kontaktiert: "
            + server.getLocalAddress()
            + server.getLocalPort());
```

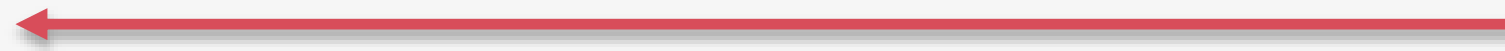
Eigene Parameter

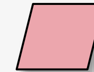
Echo-Server I - Prinzip



Echo-Server II – Senden & Empfangen

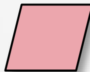


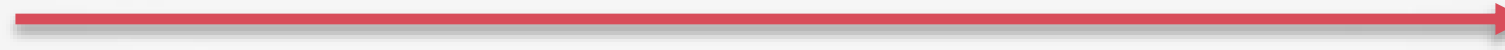


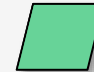


strFromAndToClient = socket.read();

socket.write(strToServer);







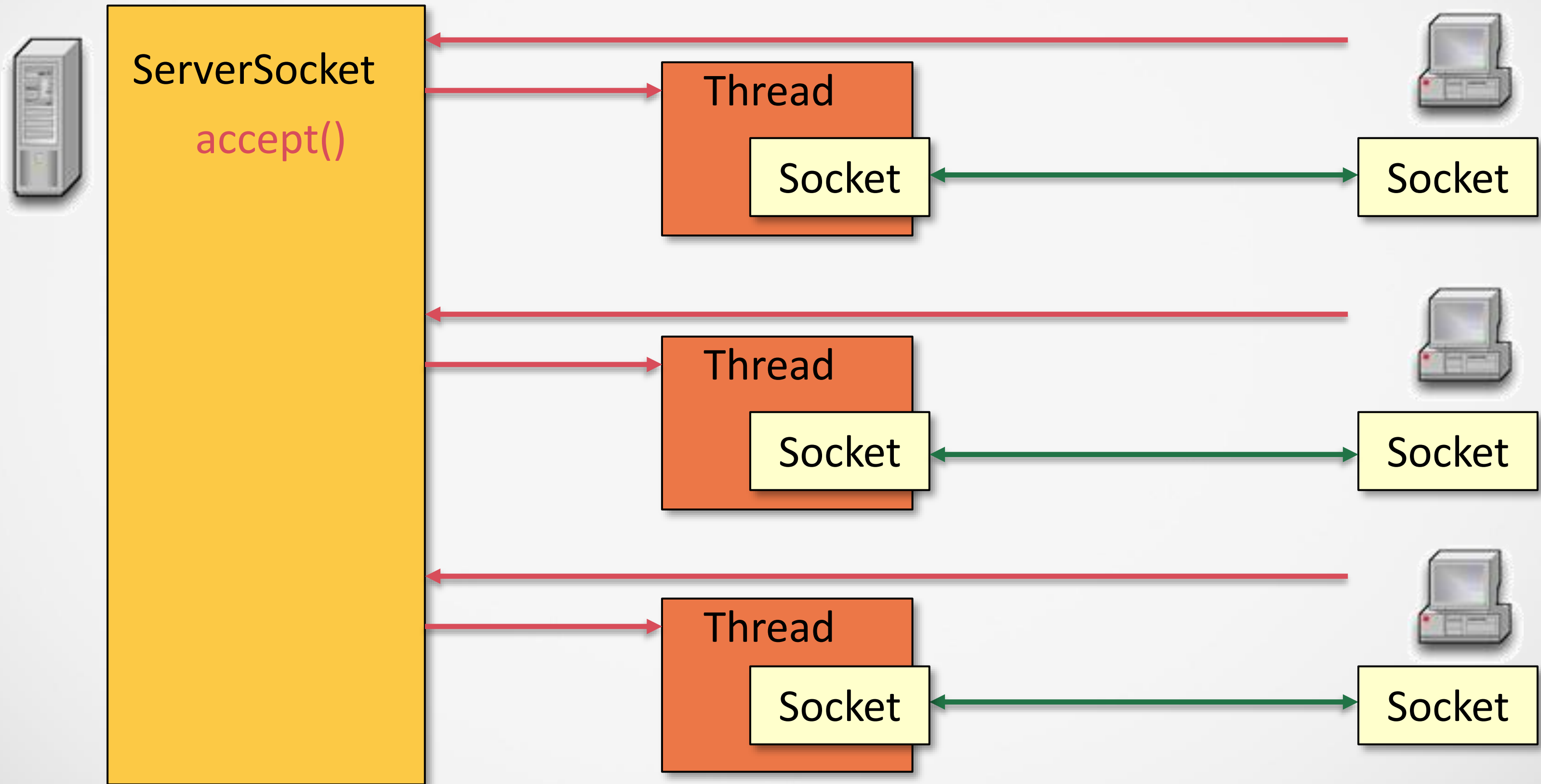
socket. write(strFromAndToClient);

strFromServer = socket.read();

Echo-Server mit Threads I - Motivation

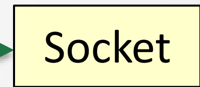
- ✓ Der Echo-Server kann nur mit einem Klienten kommunizieren. Erst wenn dieser sich abgemeldet hat, dann kann der nächste sich anmelden, wenn der Server wieder bei der Methode accept() angelangt ist
- ✓ Lösung: Threads, wo jeweils ein Thread mit einem Clienten kommuniziert
- ✓ Regel: Threads werden eingesetzt, sobald ein Prozess sich an mehreren Stellen in seinem Programmablauf "gleichzeitig" aufhalten soll

Echo-Server mit Threads II - Prinzip

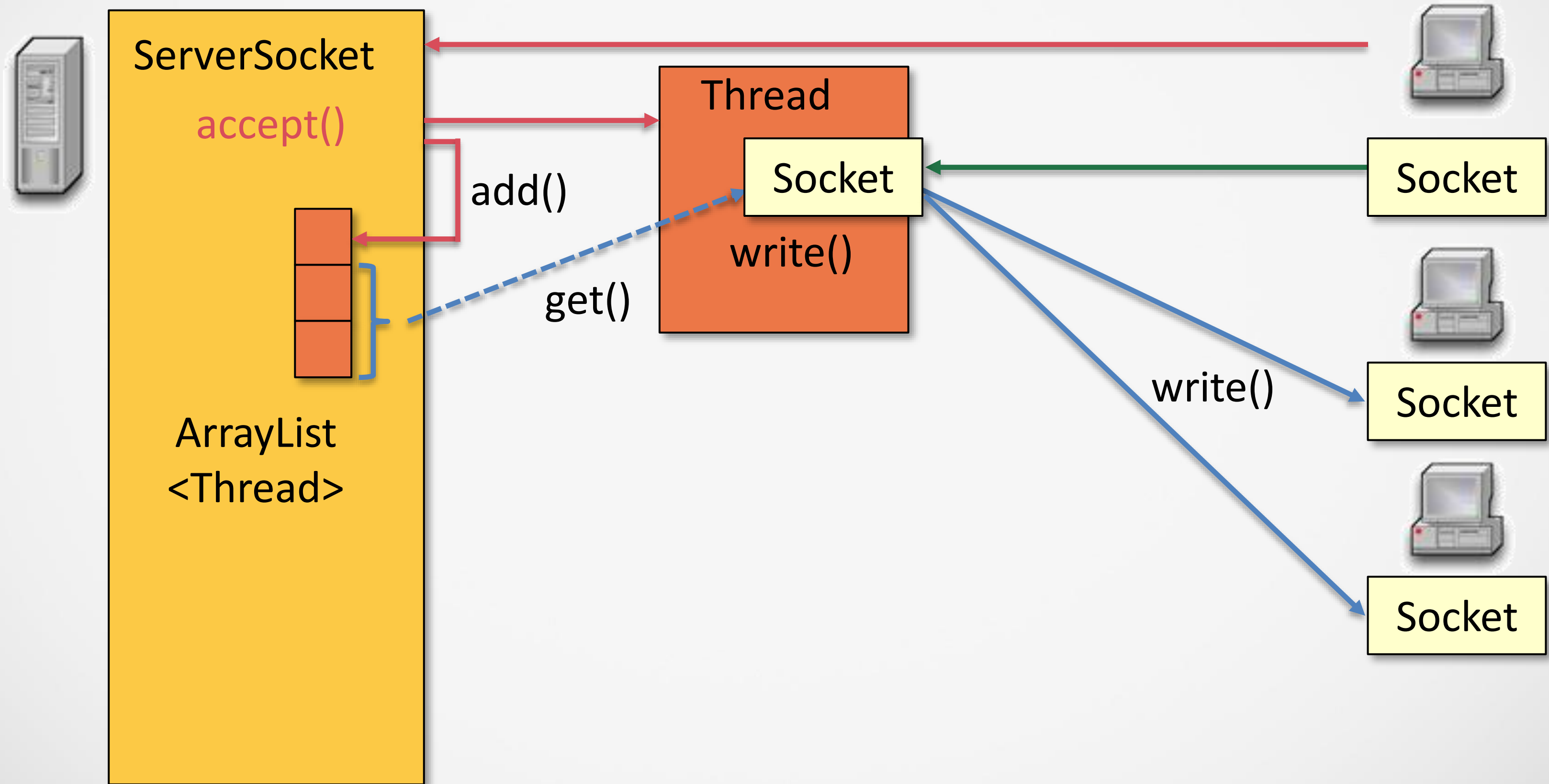


Echo-Server mit Threads III - Kodierung

```
ServerSocket serverSocket = new ServerSocket(...);
while (true) {
  Socket socket = serverSocket.accept();
  new SocketThread(socket).start();
}
```



MultiChat-Server mit Threads – Prinzip **Server**



MultiChat-Server mit Threads – Kodierungs Tricks - Server

```
arrThreads=new ArrayList<ChatServerThread>()
………….
ChatServerThread thdChatServer=new ChatServerThread(socket, arrThreads);
………….
arrThreads.add(thdChatServer);
```

ChatServer

```
public class ChatServerThread extends Thread{
 private ArrayList<ChatServerThread> arrThreads;
 …………………….
 ChatServerThread(Socket socket,ArrayList<ChatServerThread> arrThreads){
  this. arrThreads=arrThreads;
  ……………………….
 }
```

ChatServerThread

Kopie der Adresse

Übergabe der Adresse

MultiChat-Server mit Threads – Prinzip Client