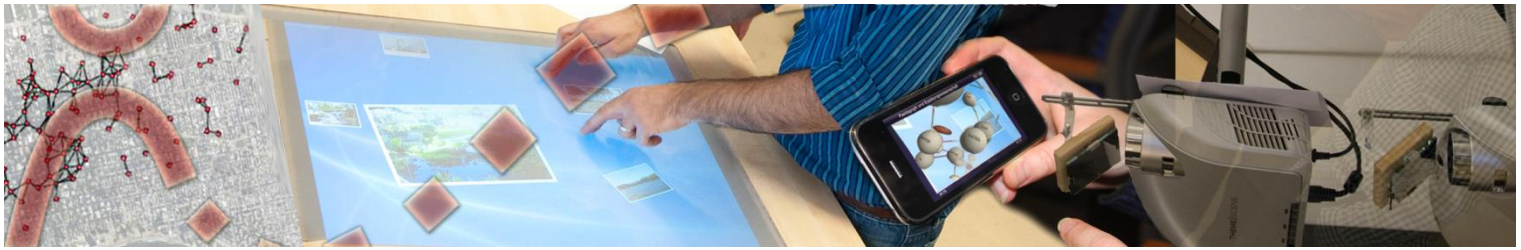
Telecooperation Lab
Prof. Dr. Max Mühlhäuser

Telekooperation 1: Exercise WS15/16

Michael Stein, MSc.
michael.stein@tk.informatik.tu-darmstadt.de
Jens Heuschkel, MSc.
jens.heuschkel@tk.informatik.tu-darmstadt.de



TK1 – EXERCISE

- Solution 3rd Exercise
- 4th Exercise

Task 1.1: Threading (2 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

Explain in a few sentences (in your own words) the introduced concepts of a Monitor.

Task 1.1: Threading (2 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Monitor** (here: Tanenbaum; terms in literature not 100% consistent)
  - Programming-language construct
  - A monitor is a „module“ containing variables and procedures
  - Data encapsulation, because variables can only be accessed via procedures
  - If process A executes a procedure (enters the monitor), then a process B trying to execute a procedure of the same monitor, will be blocked until A exits.
  - Every Java object has a built-in monitor lock: use **synchronized**, e.g.:

```
public class CountingIntMonitor {
    private int value;
    public synchronized int value()  { return value; }
    public synchronized void increment()
                                { value = value + 1; }
}
```

- **synchronized** has two effects:
  - Concurrent invocations **cannot interleave**
  - Establishes a **happened-before relationship** with any subsequent call

Task 1.2: Transparency in Java RMI (4 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

- What *transparency* features are present in Java RMI?

Task 1.2: Transparency in Java RMI (4 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

Transparency: abstracts from a specific aspect

- Access Transparency: local & remote resources accessed using identical op's
- Location T.: resources accessed w/o knowledge of their physical/network location
- Concurrency T.: several processes operate concurrently using shared resources without interference between them
- Replication T.: multiple instances of resources used (→ reliability, performance) w/o knowledge of replicas by users & programmers
- Failure T.: concealment of faults, allowing users & programs to complete tasks despite HW/SW failure
- Mobility T.: movement of resources / clients w/o affecting users & programs
- Performance T.:
  - local/remote op (exec, data access) don't differ by orders of magnitude (most persistent problem!)
  - allows system to reconfigure to improve performance as loads vary
- Scaling T.: allows system & applications to expand in scale w/o change to sys. structure or application algo's

Task 1.2: Transparency in Java RMI (4 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Access Transparency: local & remote resources accessed using identical op's
  Java RMI: YES (see right)

```
synchronized (clients) {
    for (IWhiteboardClient client : clients.values()) {
        if (client != null) {
            client.receiveLine(start, end, color);
        }
    }
}
```

- Location T.: resources accessed w/o knowledge of their physical/network location
  Java RMI: Yes (somewhat). Actual location irrelevant when making the call /
  No. Address needed for lookup

```
client = new WhiteboardClientImpl();
Naming.rebind("rmi://localhost/" + clientName, client);
```

- Concurrency T.: several processes operate concurrently using shared resources without interference between them
  Java RMI: Yes (somewhat). Server is not synchronized by itself. However, this can be easily done using the "synchronized" keyword.

- Replication T.: multiple instances of resources used (→ reliability, performance) w/o knowledge of replicas by users & programmers
  Java RMI: NO. Java RMI will not make any automatic copies.

Task 1.2: Transparency in Java RMI (4 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Failure T.: concealment of faults, allowing users & programs to complete tasks despite HW/SW failure
  Java RMI: NO. No failure tolerance for hardware failures. TCP-like guarantees, no retry-mechanism.

- Mobility T.: movement of resources / clients w/o affecting users & programs
  Java RMI: NO. The resulting address changes and connection losses are not handled automatically by Java RMI.

- Performance T.: local/remote op (exec, data access) don't differ by orders of magnitude (most persistent problem!)
  Java RMI: NO. Because of communication delay the execution time of a remote call is usually significantly slower. Furthermore, there is an overhead to establish the connection.

- Scaling T.: allows system & applications to expand in scale w/o change to sys. structure or application algo's.
  Java RMI: YES, client and server are decoupled by the RMI registry so that the servers can scale without the clients noticing.

Task 1.3: RMI - single-threaded vs. multi-threaded (3 P.)

TECHNISCHE UNIVERSITÄT DARMSTADT

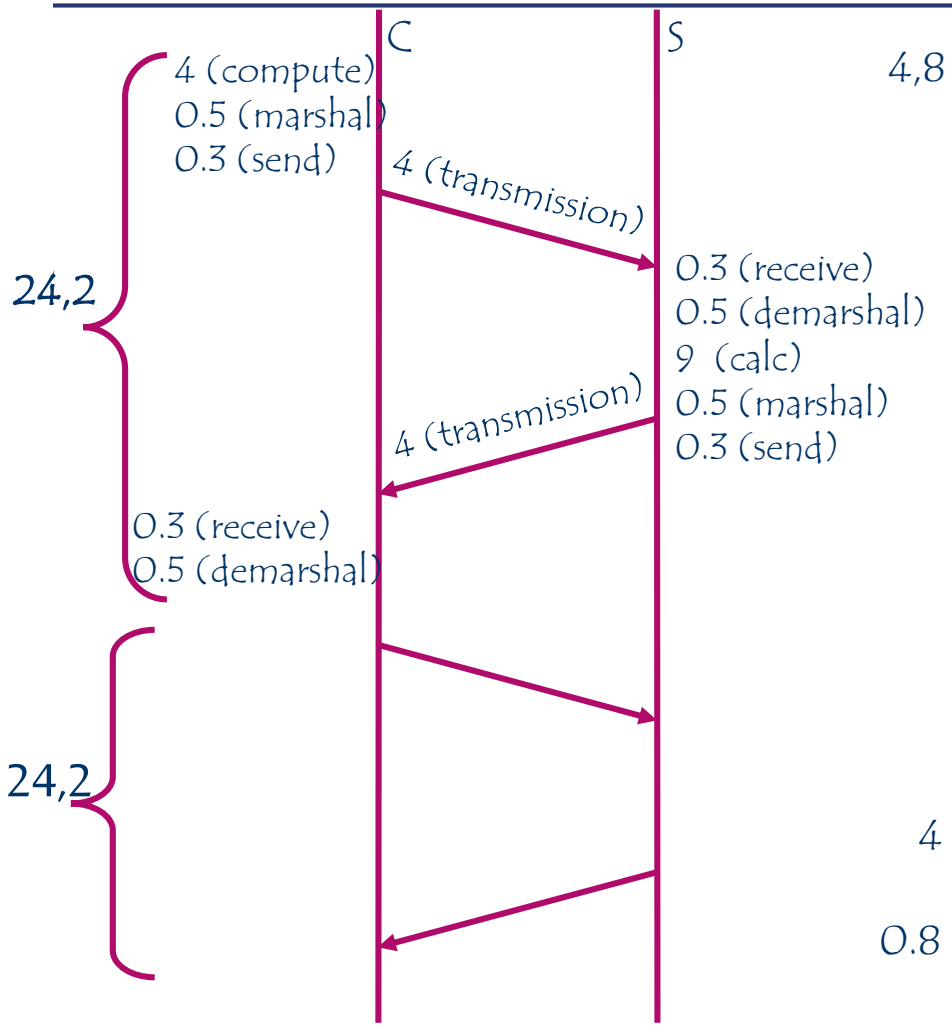
A client executes RMI on a server. The client requires 4 ms to compute the arguments for each request, and the server requires 9 ms to process each request. The process time of the local operating system of each send or receive operation is 0.3 ms and the network time to transfer the request or response is 4 ms. The Marshalling and Demarshalling takes 1 ms in total per message.

Estimate the time, which the client requires to generate two requests and obtain a refund, if

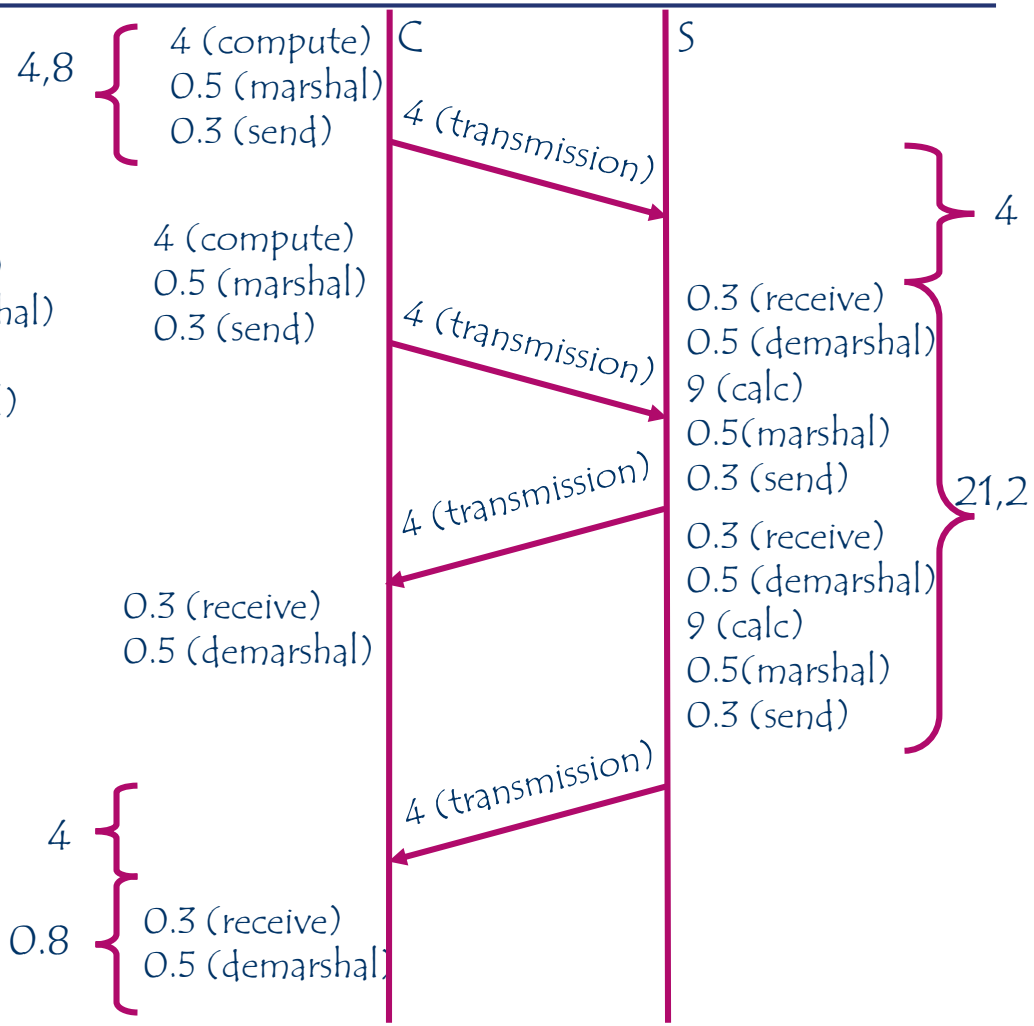
1. it is single-threaded

2. it has two threads, which can generate concurrent requests on a single processor. The server, which has also one processor, processes the requests in order of the received message.

Task 1.3: RMI - single-threaded vs. multi-threaded (3 P.)



Time for two calls ≈ 48,4ms



Time for two calls ≈ 4,8 + 4 + 21,2 + 4 + 0,8 = 34,8ms