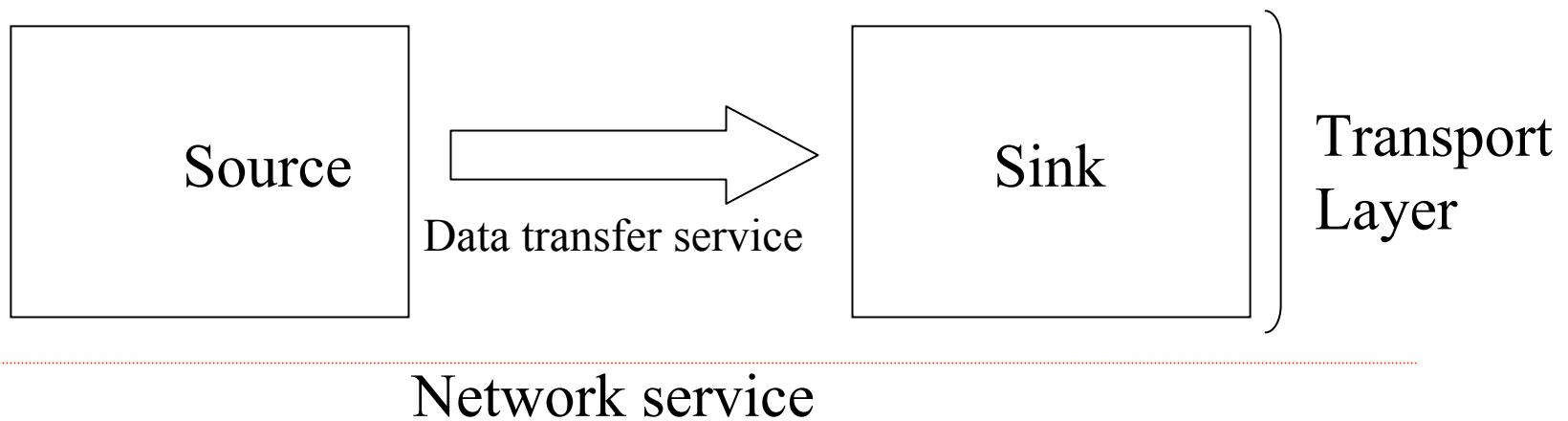


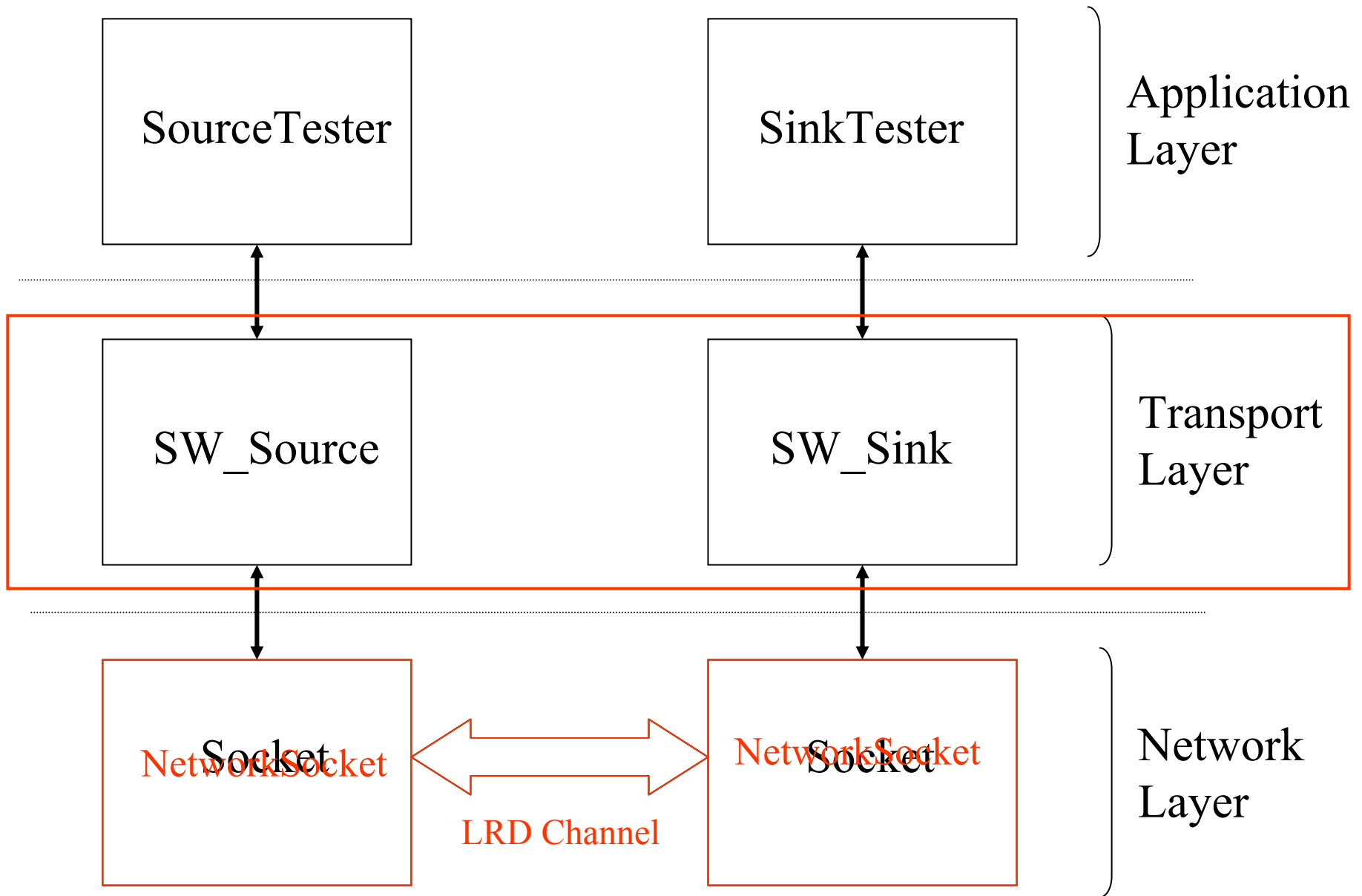
Homework Assignment 5:

A Sliding Window Data Transfer Protocol

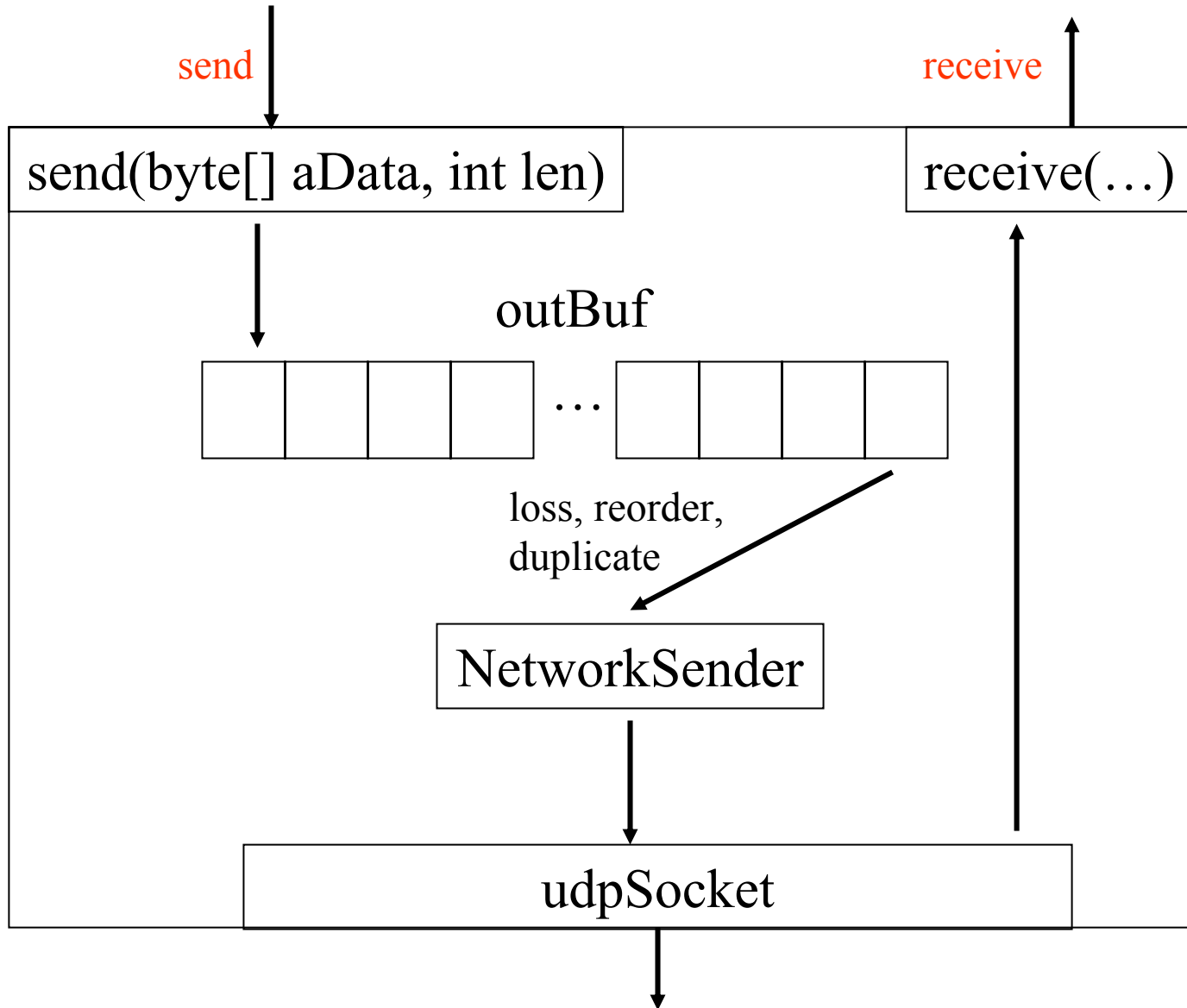


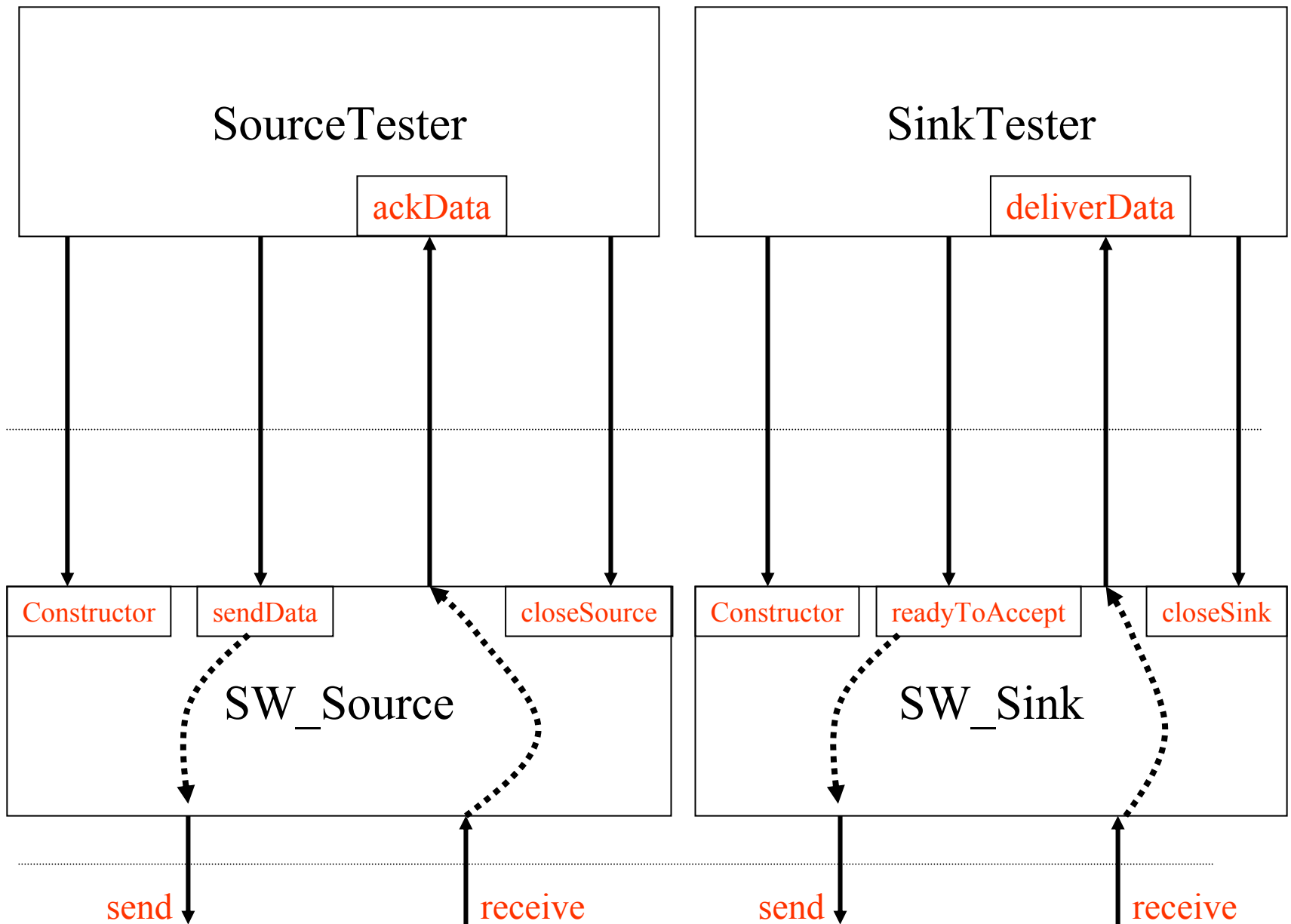
Reliable data transfer means:

- Data is delivered in the same order without loss or duplication.
- Data sent is eventually delivered.
- Provided that:
 - Source and sink are always connected and correctly initialized.



NetworkSocket



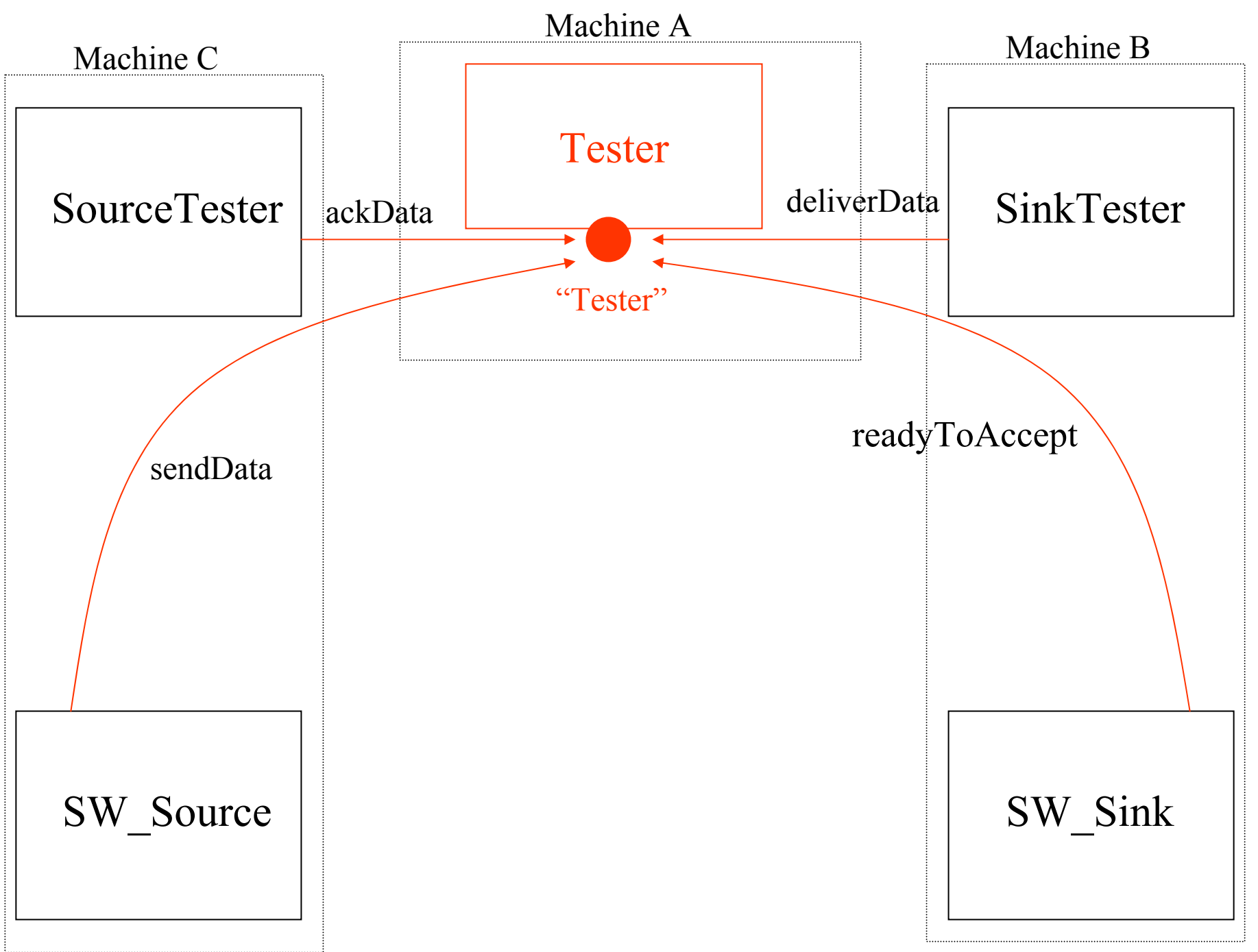


Blocking vs. nonblocking

- Blocking TCP/IP
 - Method `sendData(byte data[])` blocks till all bytes of data are sent and acked.
 - User cannot call another `sendData` unless the current one is executed.
- Nonblocking TCP/IP
 - Method `sendData(byte data[])` adds data to the buffer, and returns to the caller.
 - Data is sent later.

Testing and Debugging

- Program involves multiple computers.
- Ability to discover errors becomes harder.
- A research method to test distributed systems
 - Specify interactions between different entities of a distributed system in a single `service` file.
 - Specifications of interactions describe the relations between different interactions
 - Use Remote Method Invocation (RMI) to implement this test method
 - *Lock before issuing an RMI call!*



```
class SourceTester {
    TesterInter tester; // lookup(...)

    .....

    public void sendData(byte[] data){
        try {
            tester.lock();
            tester.sendData(data);
            tester.printLog(Thread.currentThread().getName() +
                ":SW_Source:acceptData"+"");
            checkAssertions(true);

            // method body
            // .....
            tester.unlock();
        } catch (RemoteException re) {
            re.printStackTrace();
        }
    }
}
```

Tester Execution

- Change “cicada.cs.yale.edu” to an appropriate machine, e.g., “csy02.cs.wmich.edu”.
 - Execute Tester on machine A.
 - Execute Sink on machine B.
 - Execute Source on machine C.
-
- Machines A, B and C may be the same machine during development.

Assertions

- Assertion definition.
- Have method `checkAssertion` at the start of every method.

```
• class DT ...{
• ...
• public void sendData(...) throws RemoteException {
•     synchronized (tester.lock){
•         ...
•         checkAssertions(true);
•         ...
•     }
• }
• ...
• public void deliverData(...) throws RemoteException {
•     synchronized(tester.lock){
•         ...
•         checkAssertions(true);
•         ...
•     } }
• ...
• void checkAssertions(boolean debugInfo){
•     try{
•         tester.printLog(":");
•         if ( ( srcBufUsed >= 0 ) && ( srcBufUsed <= srcBufSize ) )
•             tester.printLog(":bufCondition(true)");
•         else
•             tester.printLog(":bufCondition(false)");
•         tester.printlnLog("");
•     } catch (RemoteException re) {}
• } }
• } //end class DT
```

Synchronization

- A simple solution to obtain a mutual-exclusive lock for a code block
 - Define a *static Object*
 - Use *synchronized* statement
- For example:

```
static Object lock = new Object();  
... ..  
Synchronized (lock) {  
    // do things exclusively  
    .....  
}
```

Common Synchronization Errors

- lock() not followed by unlock().

```
void xyz () {  
    .....  
    lock ();  
    if (condition)  
        break;  
    .....  
    unlock ();  
}
```

Fix:

```
void xyz () {  
    .....  
    lock ();  
    if (condition) {  
        unlock ();  
        break;  
    }  
    .....  
    unlock ();  
}
```

Common Synchronization Errors

- lock() not followed by unlock().

```
void xyz () {  
    .....  
    lock ();  
    while (condition)  
        wait ();  
    .....  
    unlock ();  
}
```

Fix:

```
void xyz () {  
    .....  
    lock ();  
    while (condition) {  
        unlock ();  
        wait ();  
        lock ();  
    }  
    .....  
    unlock ();  
}
```

Common Synchronization Errors

- `lock()` not followed by `unlock()`.

```
void abc() {  
    .....  
    lock();  
    xyz();  
    .....  
    unlock();  
}  
  
void xyz() {  
    .....  
    lock();  
    .....  
    unlock();  
}
```

Fix:

```
void xyz() {  
    .....  
    //lock();  
    .....  
    //unlock();  
}
```

How to start?

1. Implement "stop and wait" protocol.
2. Increase the send window.
3. Increase LRD probabilities.
4. Change the sending window from fixed window to adaptable window.
5. Make random scenarios.

Design Specifications

- Demonstrate that you have thought about the problem and your solution is plausible
 - Use diagrams and/or pseudo-code
 - Concise and simple
- Demonstrate how you will test your code
 - Check for edge/boundary cases
 - Accurately determine correctness of your code
- *Due date: March 17, 2005*