合肥學院
HEFEI UNIVERSITY

Distributed Computing
Lesson 14: Remote Method Invocation

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Overview

- How can we run procedures on a *different* computer?
- What are remote procedure calls and remote method invocation?
- What different technologies exist for that purpose?
- Example implementation in `Java`

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread`

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread` and therefore
  - blocking until the function is finished

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread` and therefore
  - blocking until the function is finished
- What are procedure/function calls good for?

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread` and therefore
  - blocking until the function is finished
- What are procedure/function calls good for?
  - Modular software development [1, 2]

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread` and therefore
  - blocking until the function is finished
- What are procedure/function calls good for?
  - Modular software development [1, 2]
  - Smaller programs

Local Procedure Call

- Procedural programming languages like `Java` and `C` offer us the concept of function calls: e.g., `double compute(double a, double b)`
- Such a local function call (normally) has the following features:
  - executed in the calling `Thread` and therefore
  - blocking until the function is finished
- What are procedure/function calls good for?
  - Modular software development [1, 2]
  - Smaller programs
  - Separation of concerns (SoC) [3, 4]

Remote Procedure Call

- Execute a procedure on another computer... Why?

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer
  - Code of the procedure resides on other computer, is unknown to our computer (we just know procedure's signature), and must be executed there

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer
  - Code of the procedure resides on other computer, is unknown to our computer (we just know procedure's signature), and must be executed there
  - Combine advantages of modular programming with client/server systems, e.g.,

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer
  - Code of the procedure resides on other computer, is unknown to our computer (we just know procedure's signature), and must be executed there
  - Combine advantages of modular programming with client/server systems, e.g.,
  - Perform a location-dependent service

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer
  - Code of the procedure resides on other computer, is unknown to our computer (we just know procedure's signature), and must be executed there
  - Combine advantages of modular programming with client/server systems, e.g.,
  - Perform a location-dependent service
  - Centralized maintenance and updating

Remote Procedure Call

- Execute a procedure on another computer... Why?
  - Implement functionality (e.g., application logic) in a program that can be accessed from another program (e.g., front end) running on a different computer
  - Code of the procedure resides on other computer, is unknown to our computer (we just know procedure's signature), and must be executed there
  - Combine advantages of modular programming with client/server systems, e.g.,
  - Perform a location-dependent service
  - Centralized maintenance and updating
  - Platform independent binding of services
  - ...

Remote Procedure Call

- Execute a procedure on another computer... Why?
- Remote Procedure Call (RPC) [5]

Remote Procedure Call

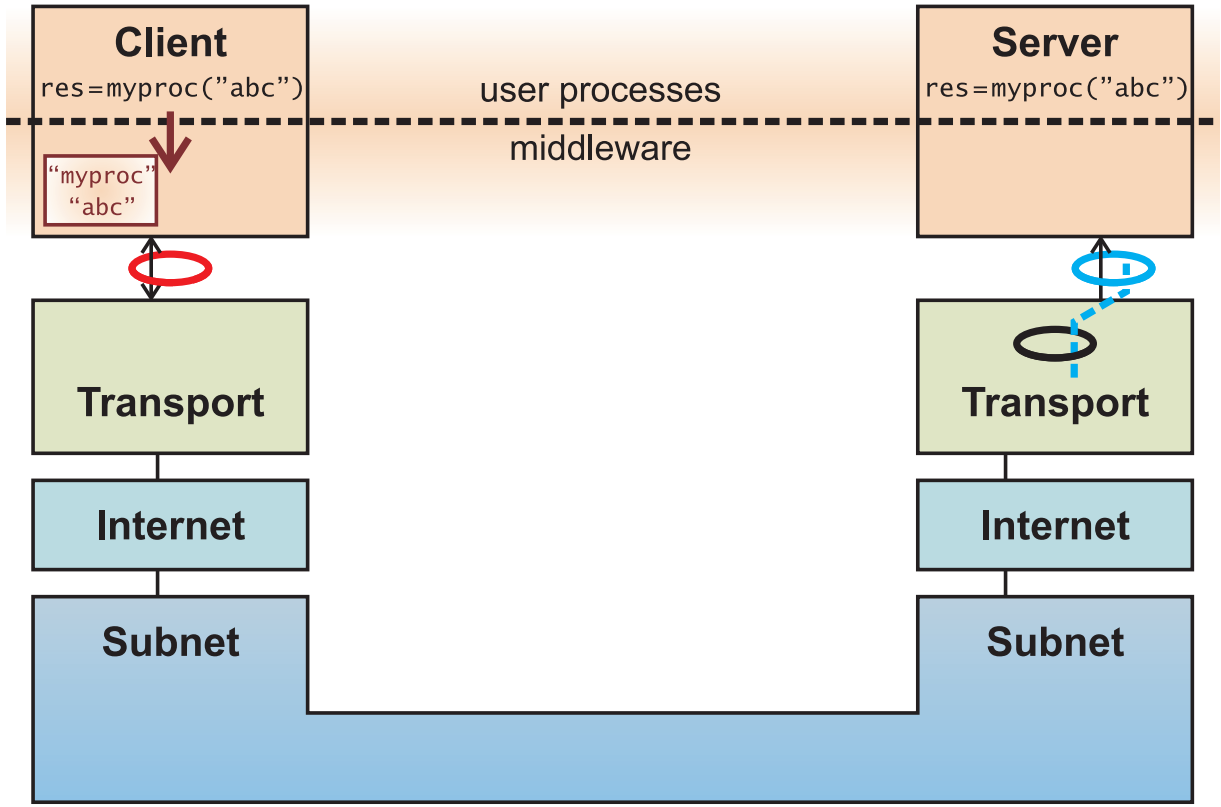
- Execute a procedure on another computer... Why?
- Remote Procedure Call (RPC) [5]:
  - call/execute a function in another process (usually on another computer)

Remote Procedure Call

- Execute a procedure on another computer... Why?
- Remote Procedure Call (RPC) [5]:
  - call/execute a function in another process (usually on another computer)
  - necessary communication offered by software framework

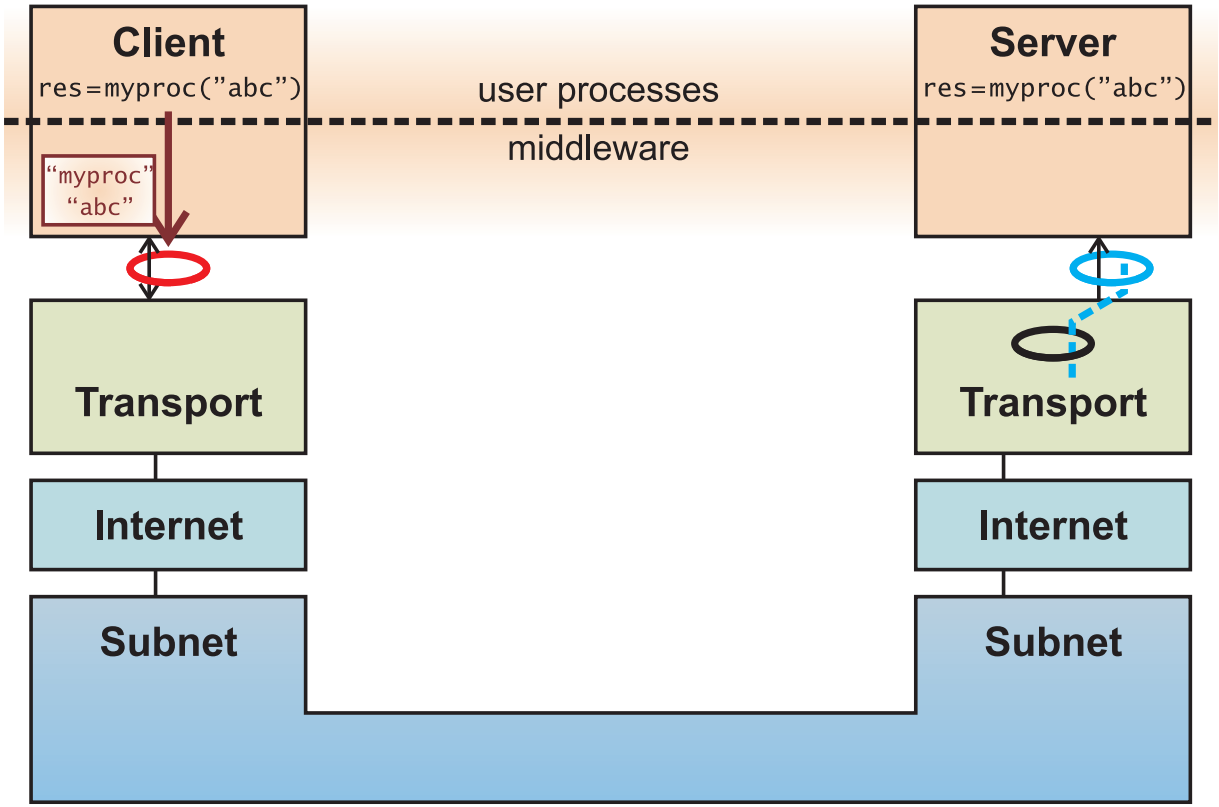
Remote Procedure Call

- Execute a procedure on another computer... Why?
- Remote Procedure Call (RPC) [5]:
  - call/execute a function in another process (usually on another computer)
  - necessary communication offered by software framework
  - from programmer's perspective: looks exactly like local procedure call



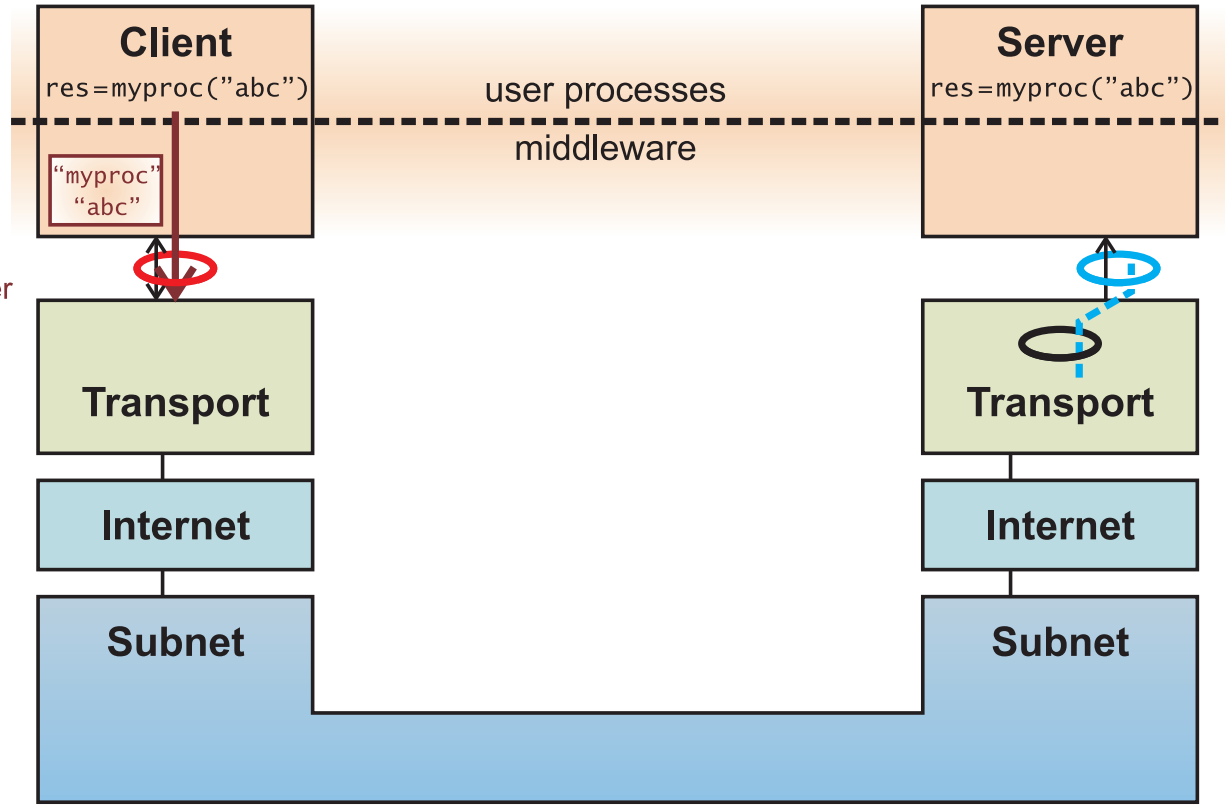
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message



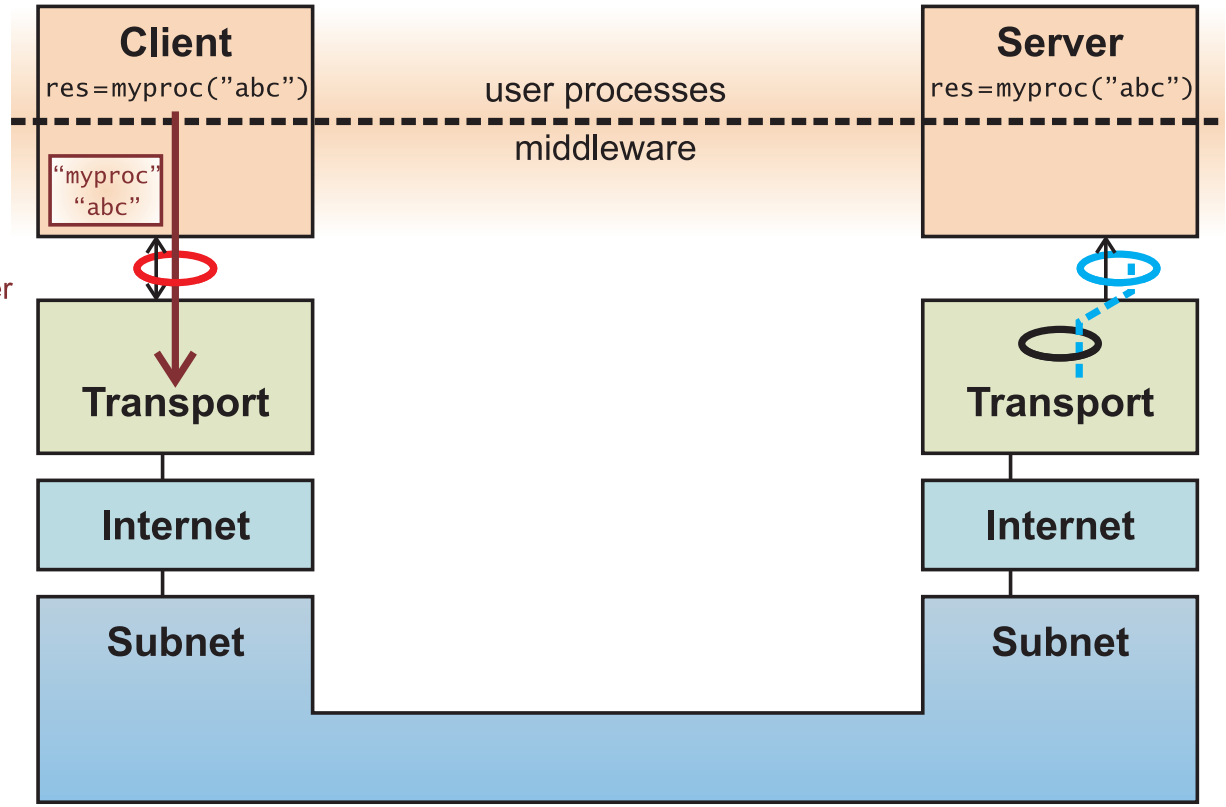
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



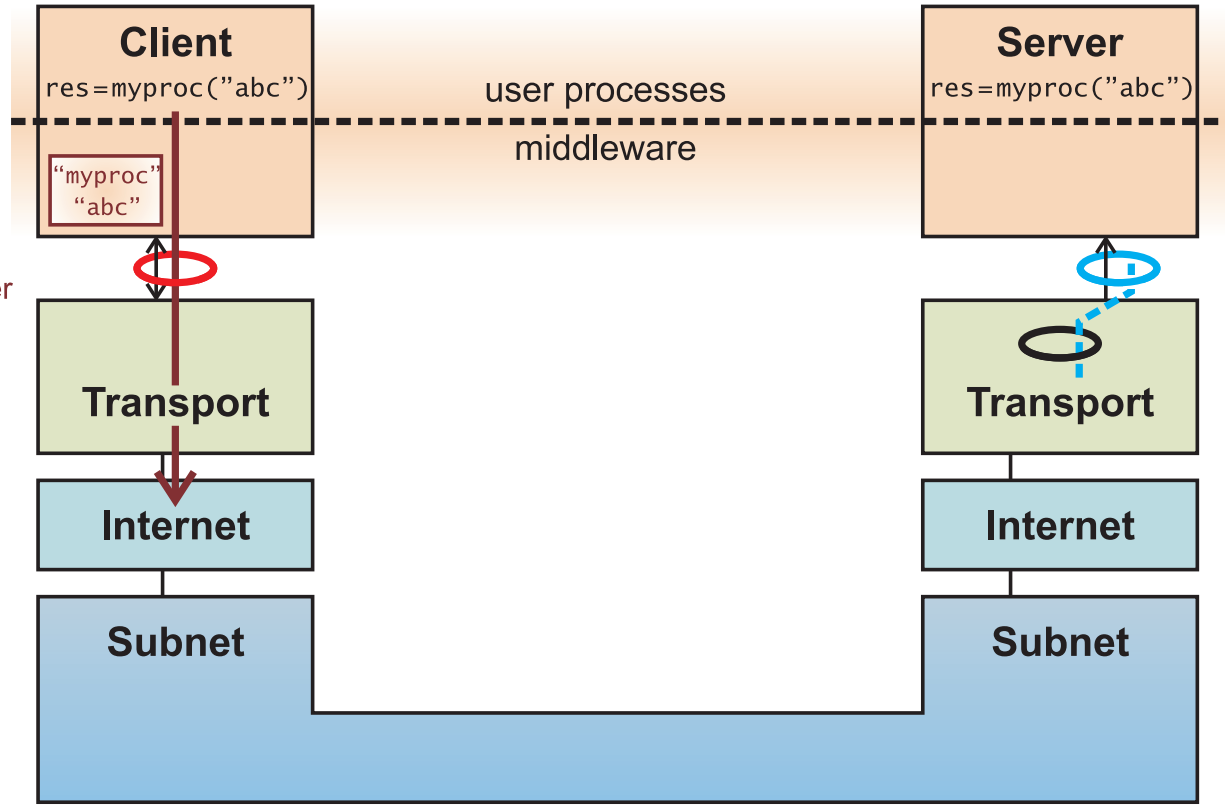
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



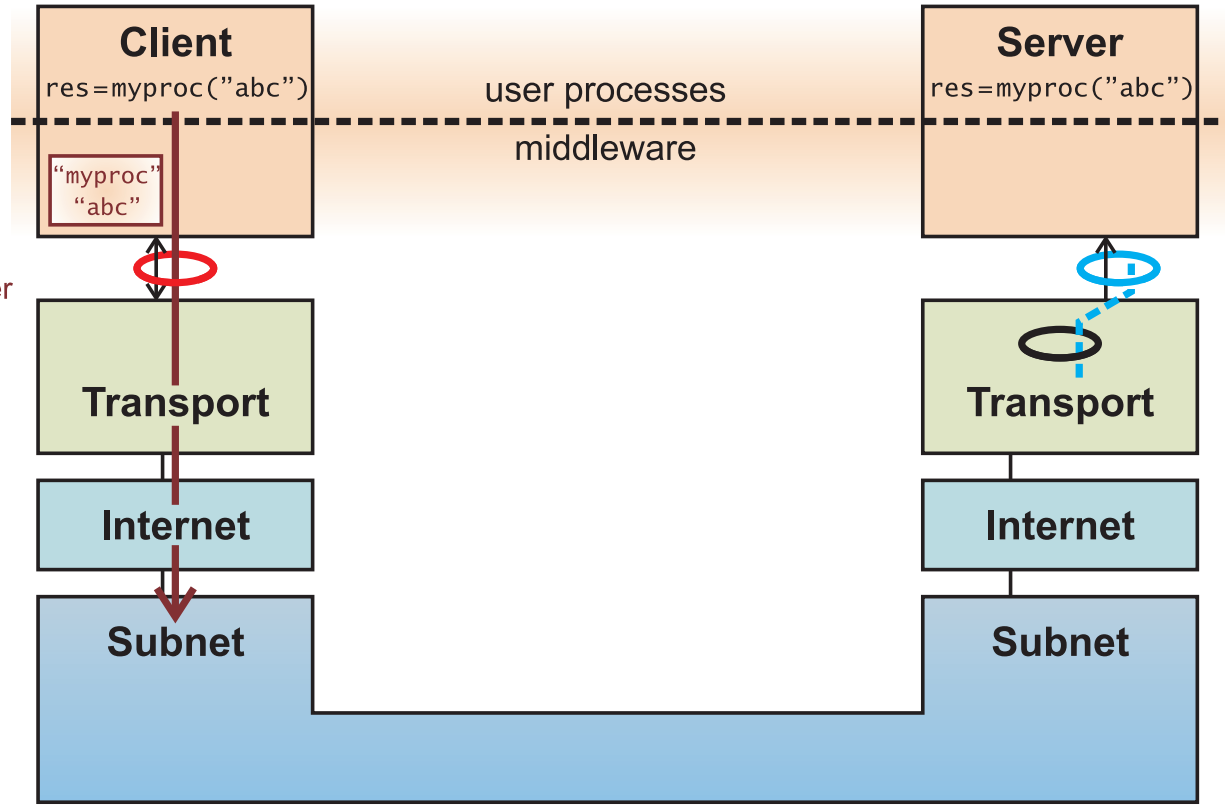
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



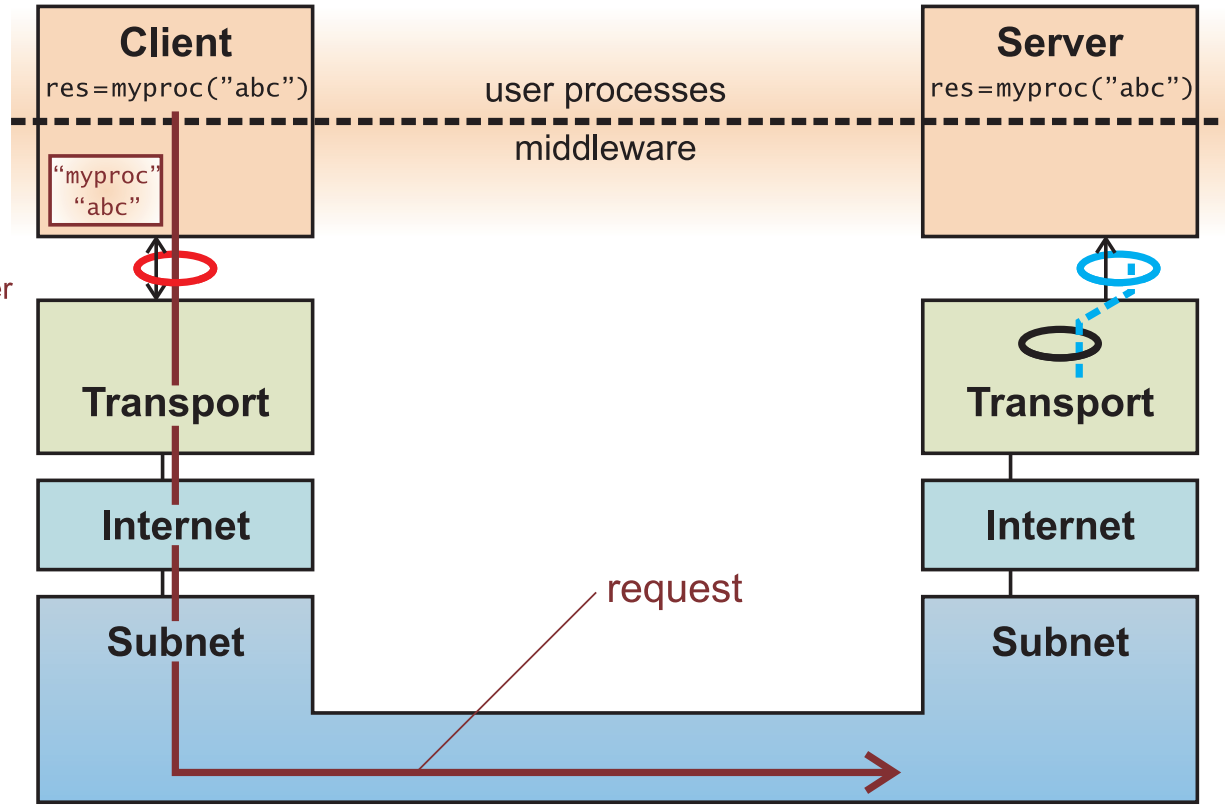
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



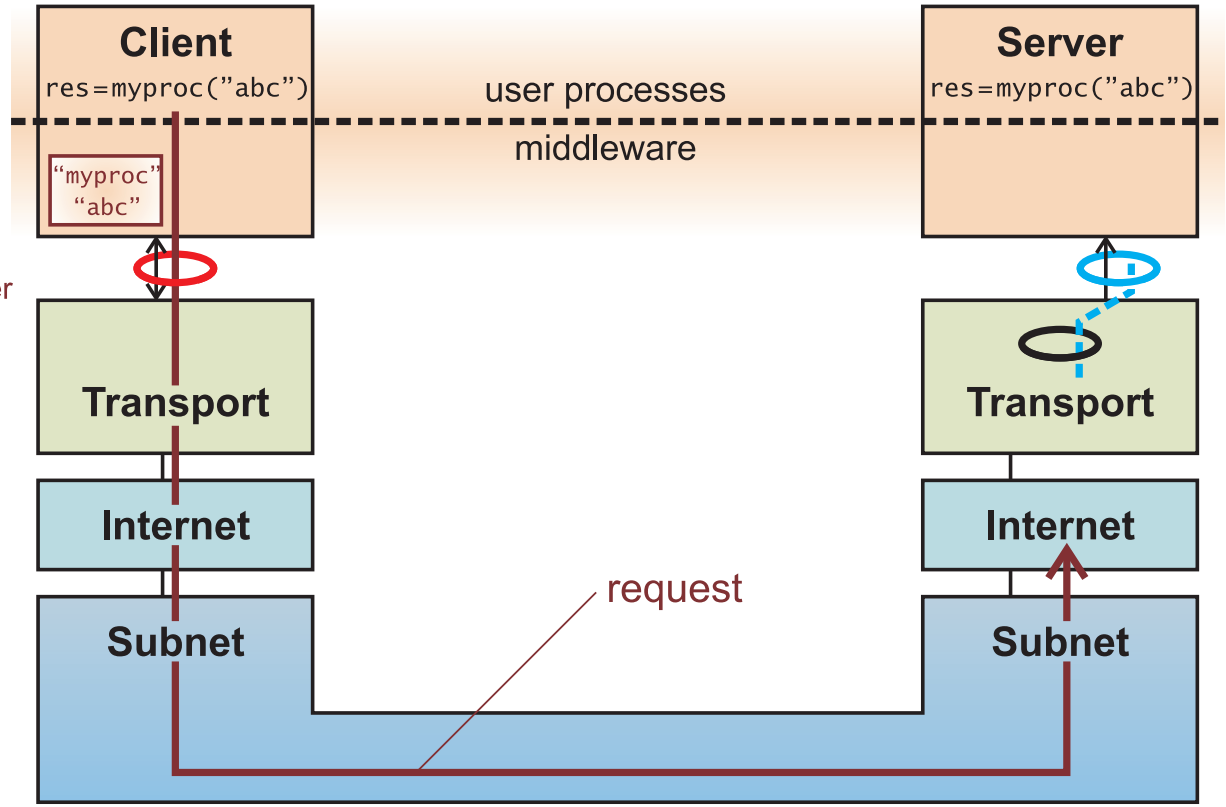
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



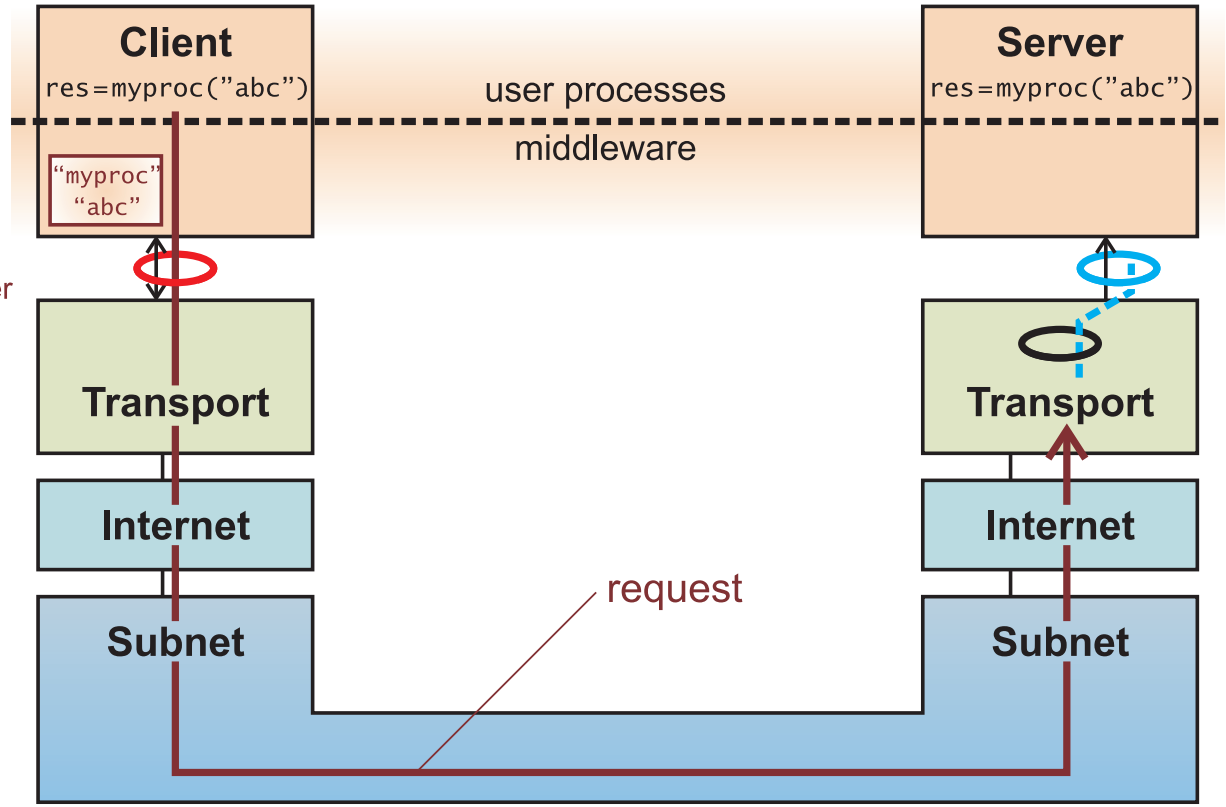
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



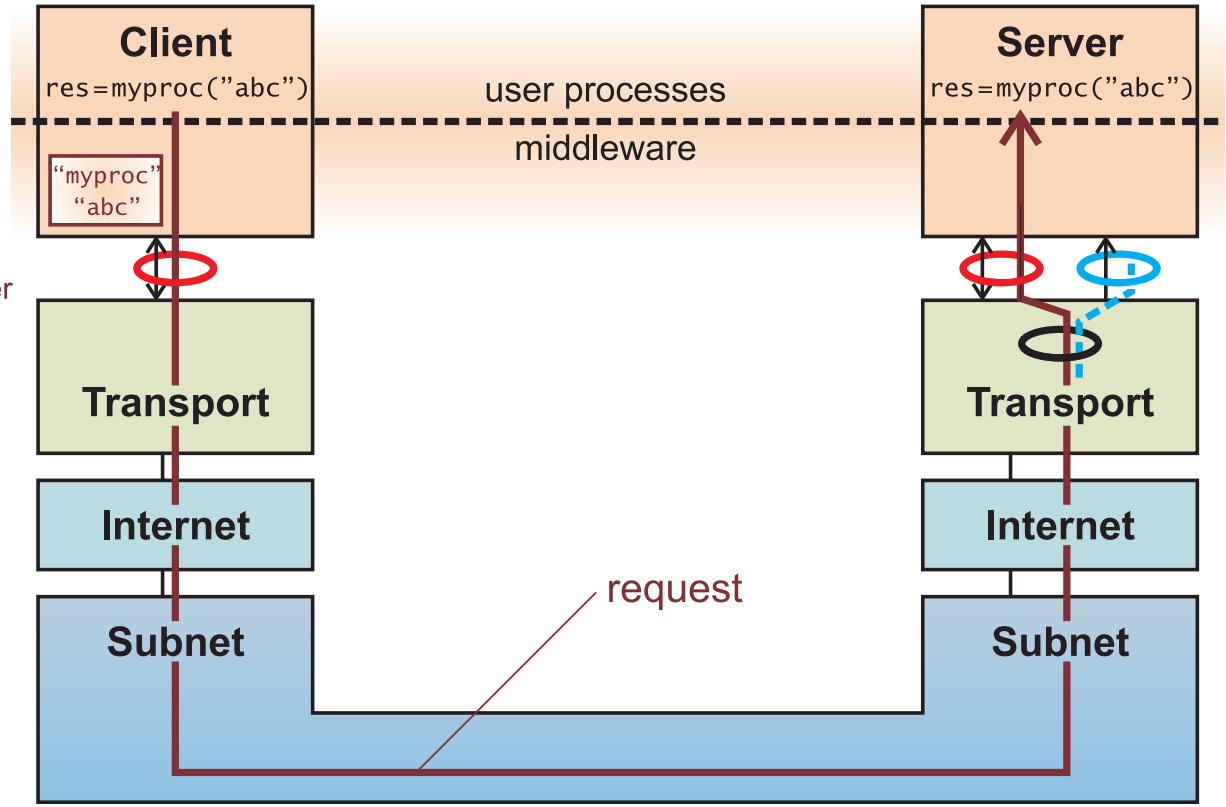
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



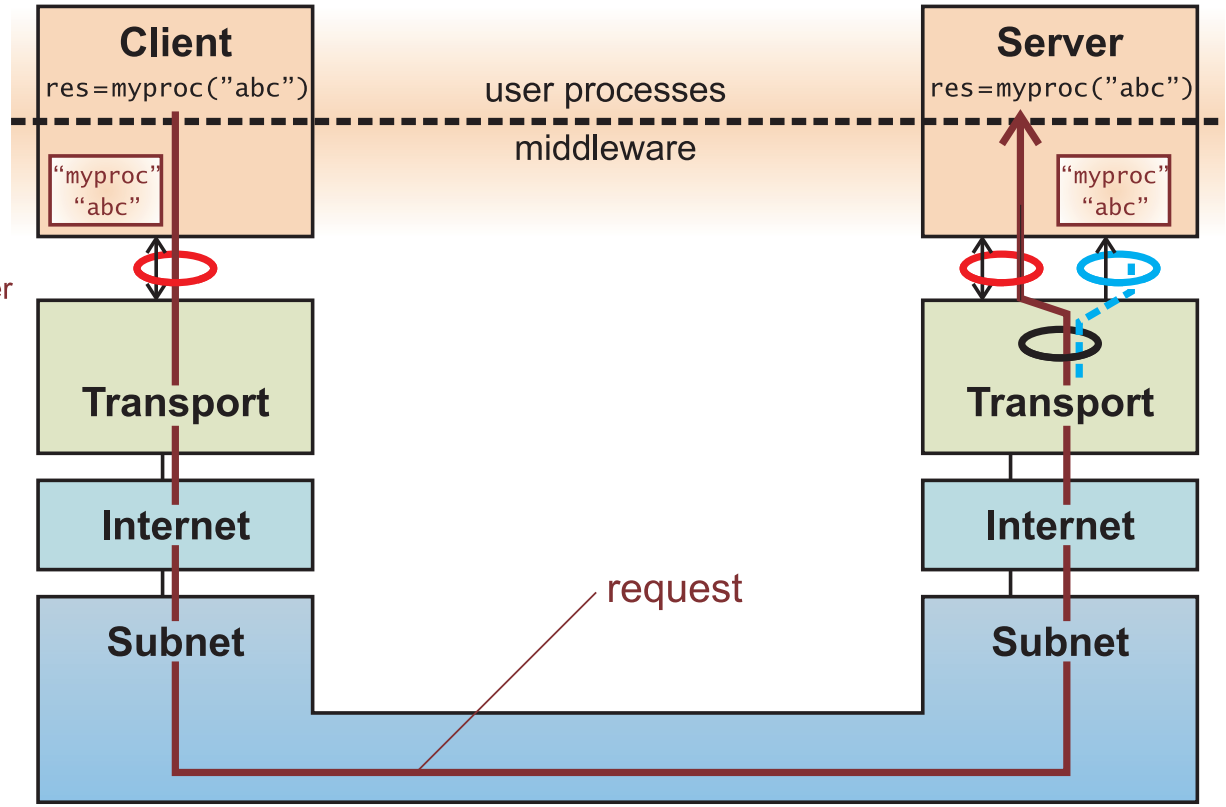
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



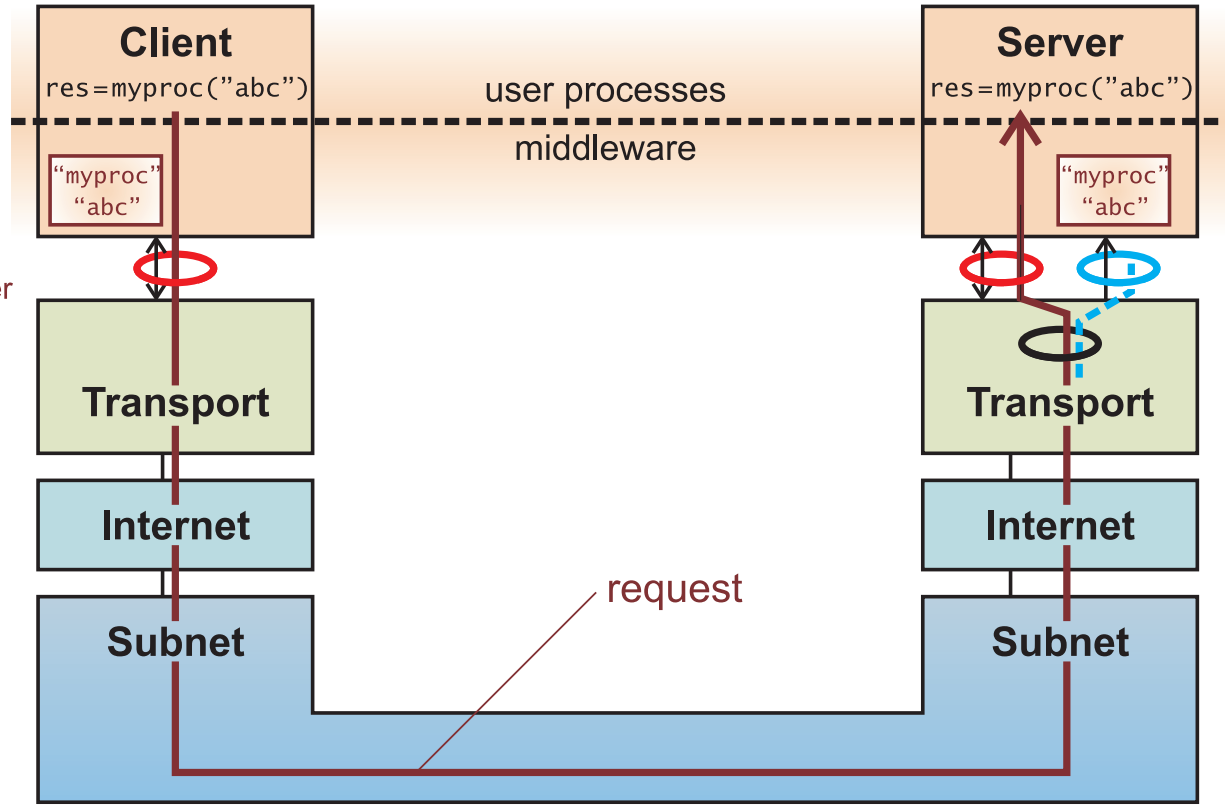
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server

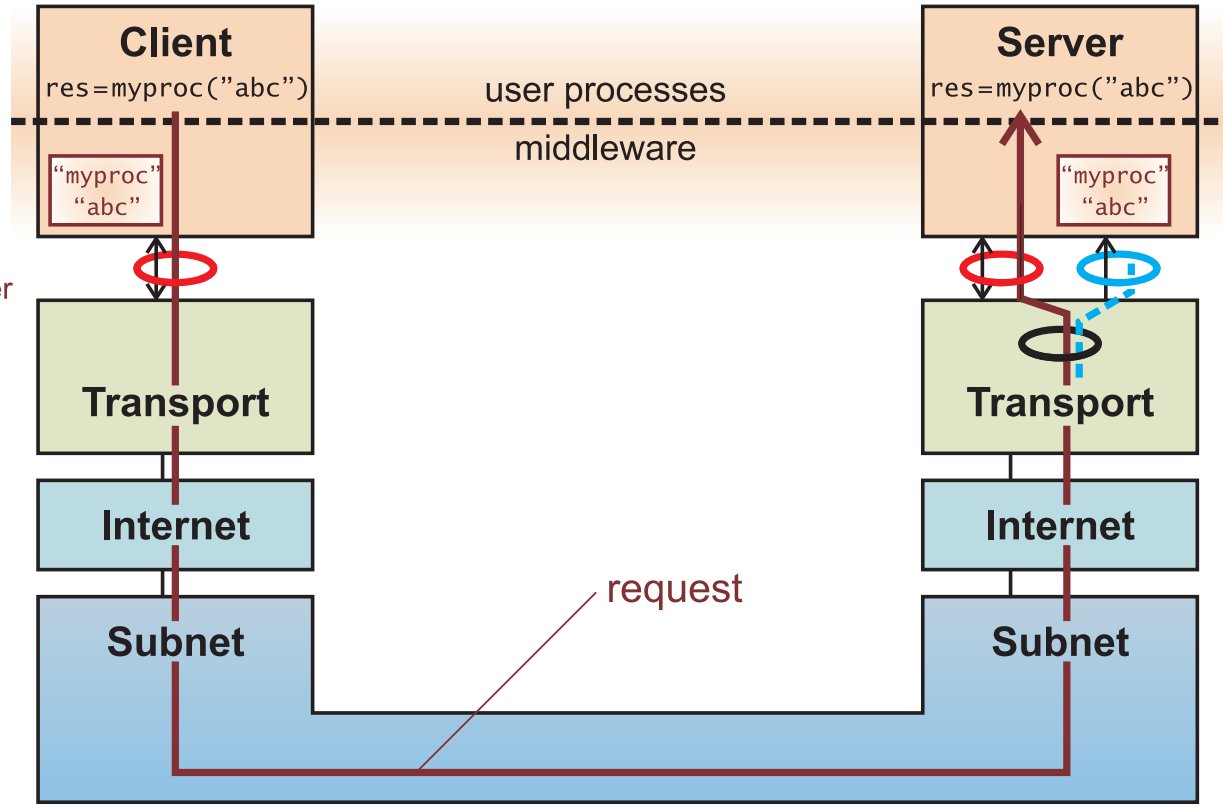


3. map procedure/object id to a real method

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

RPC

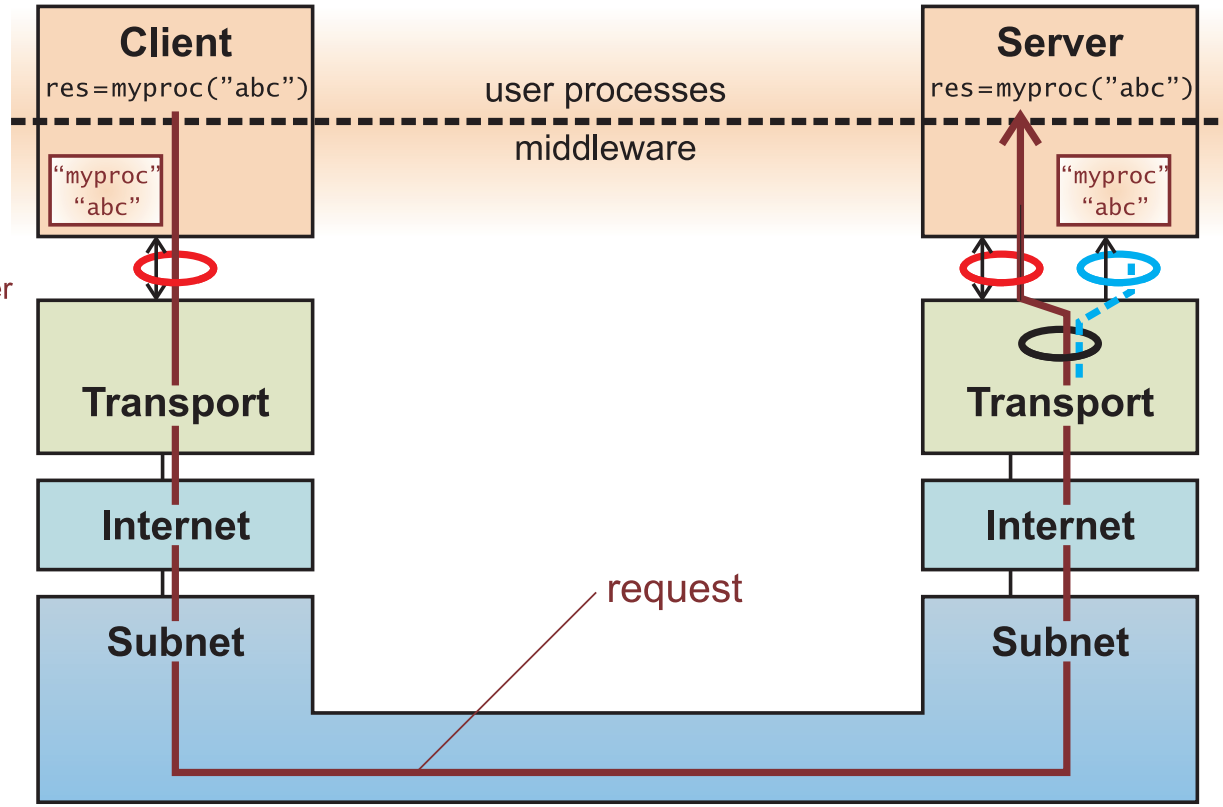
1. packing procedure id, and marshalled parameters into request message
2. send message to server



3. map procedure/object id to a real method
4. unmarshall received parameters

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

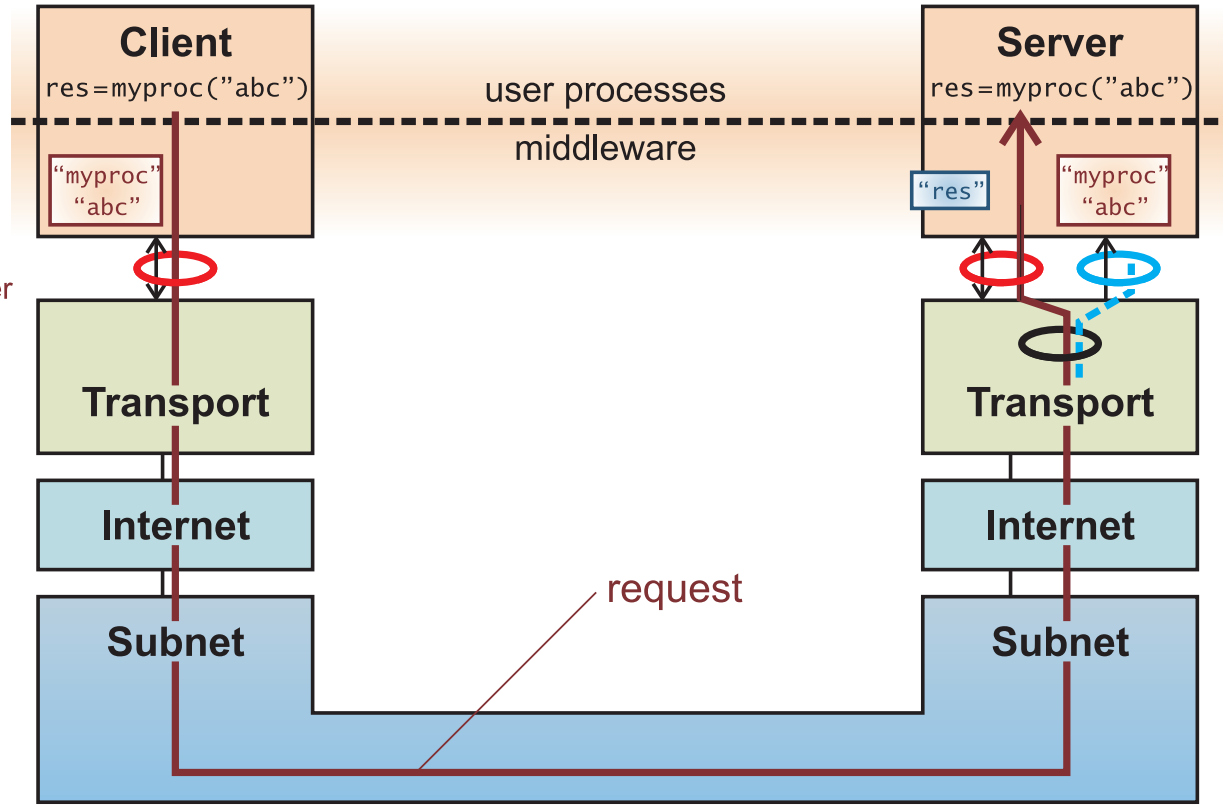
1. packing procedure id, and marshalled parameters into request message
2. send message to server



3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method

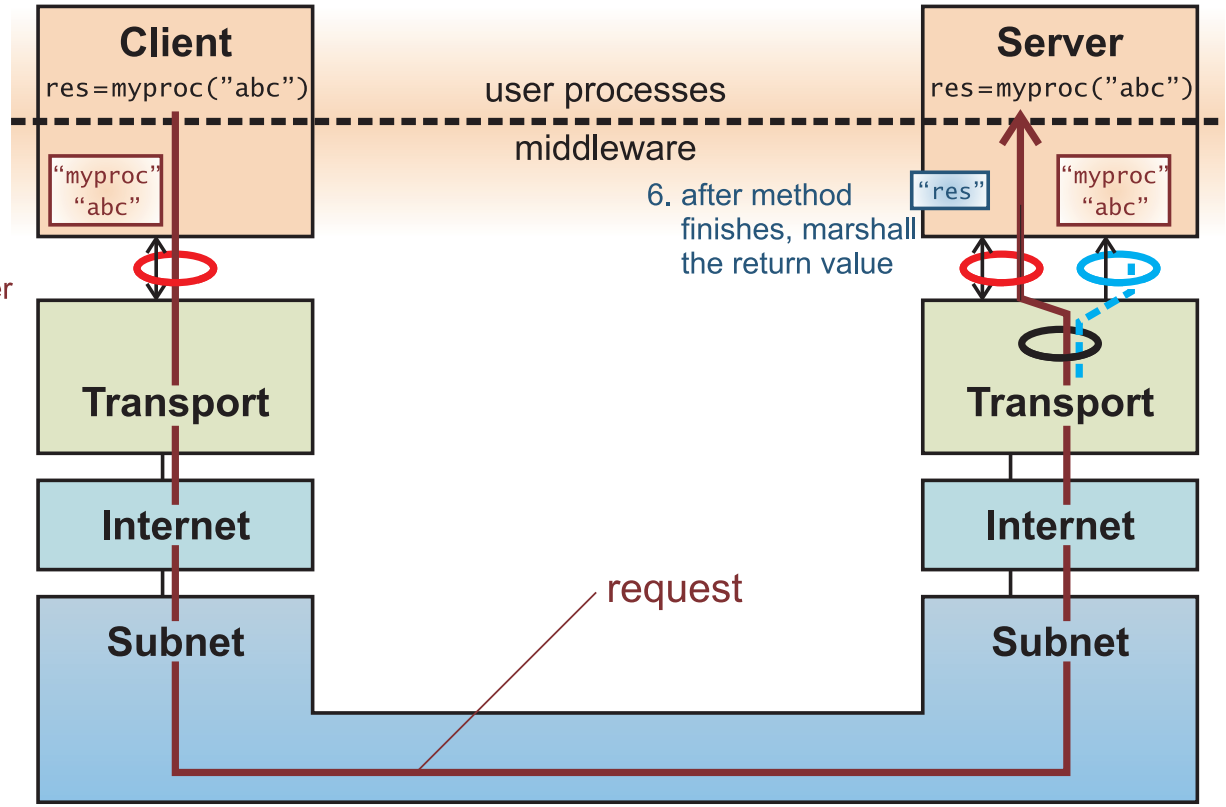
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



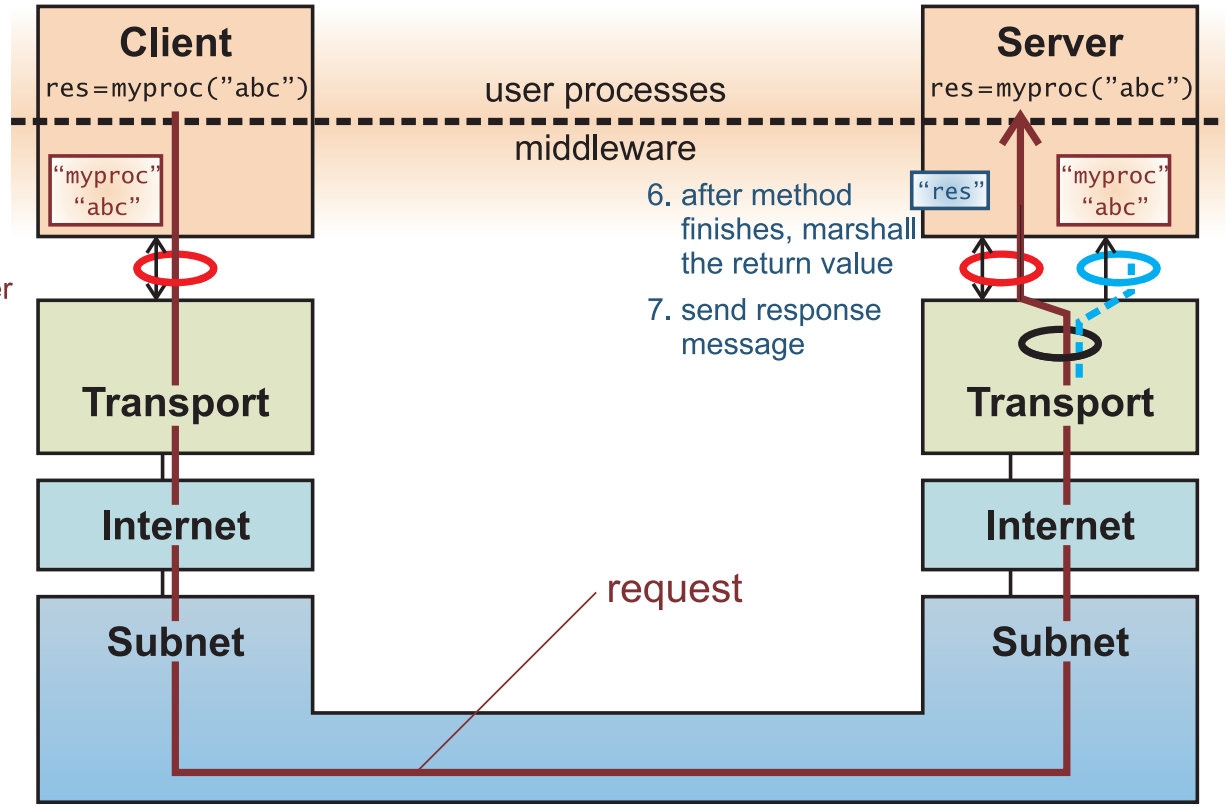
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



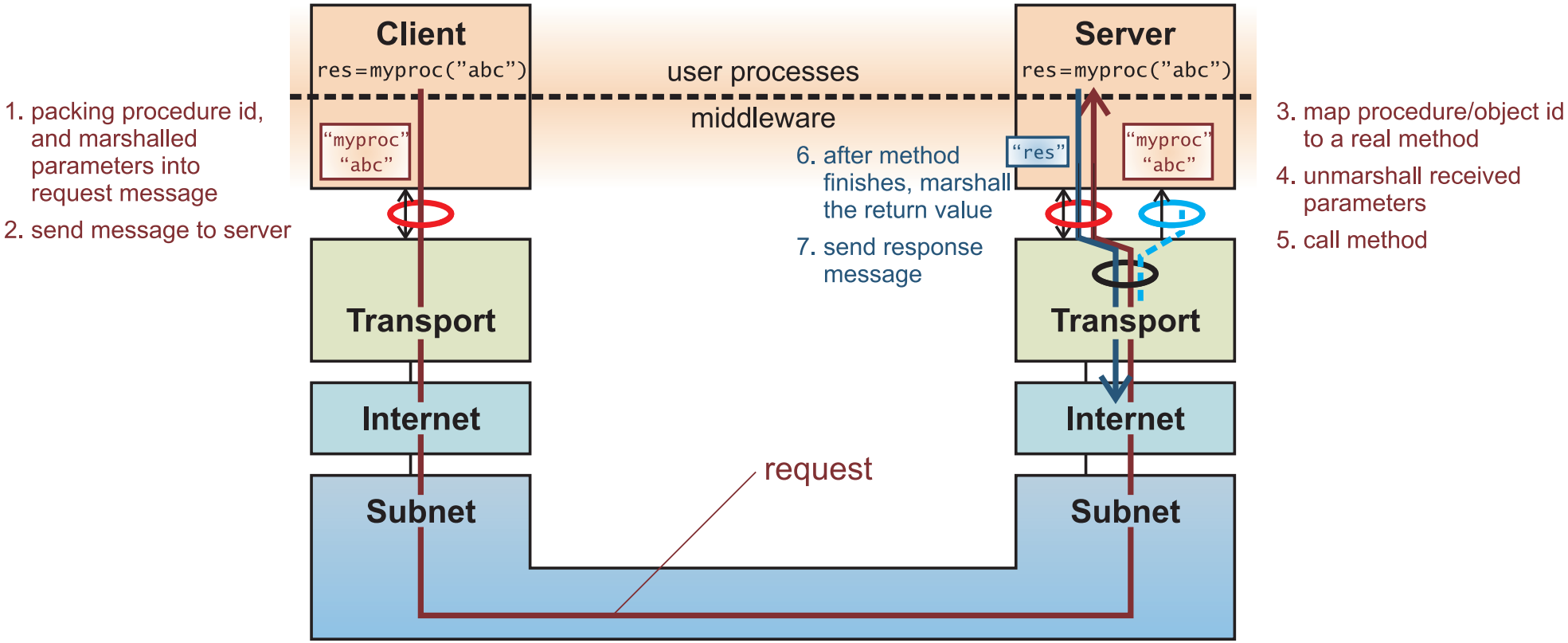
1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

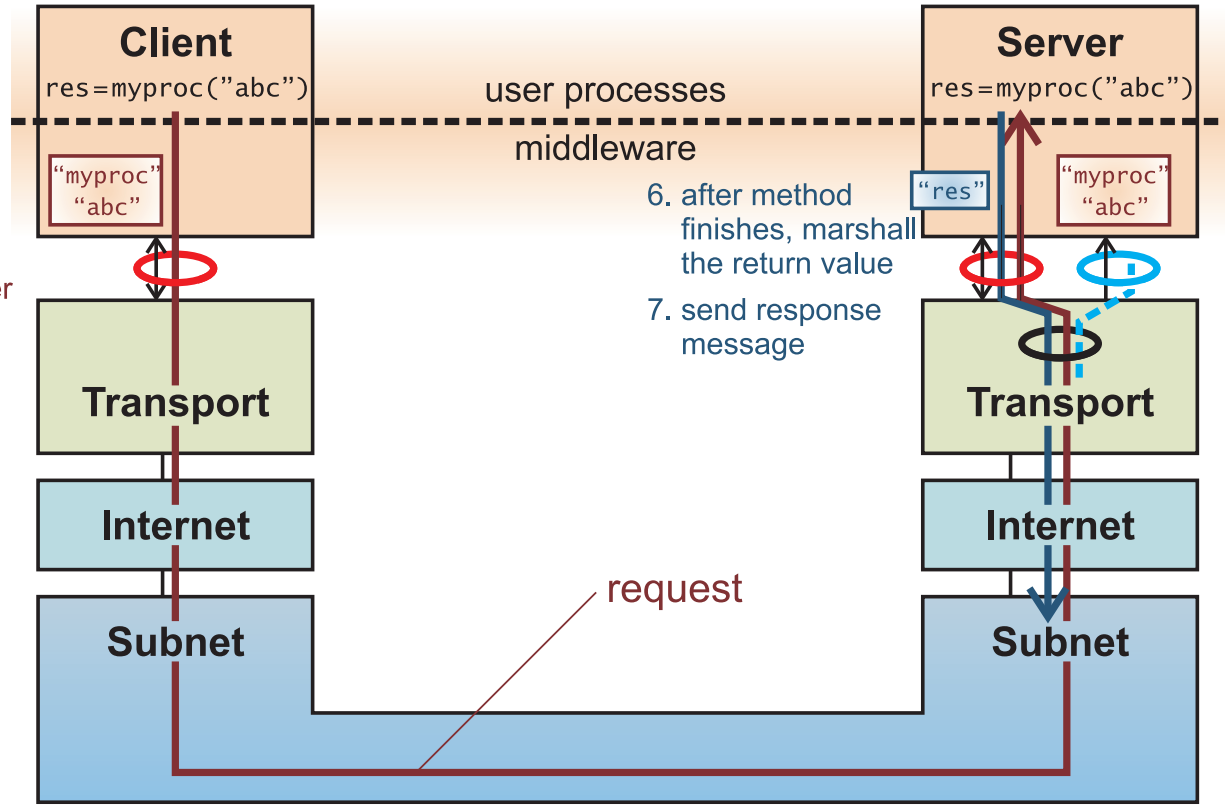


1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

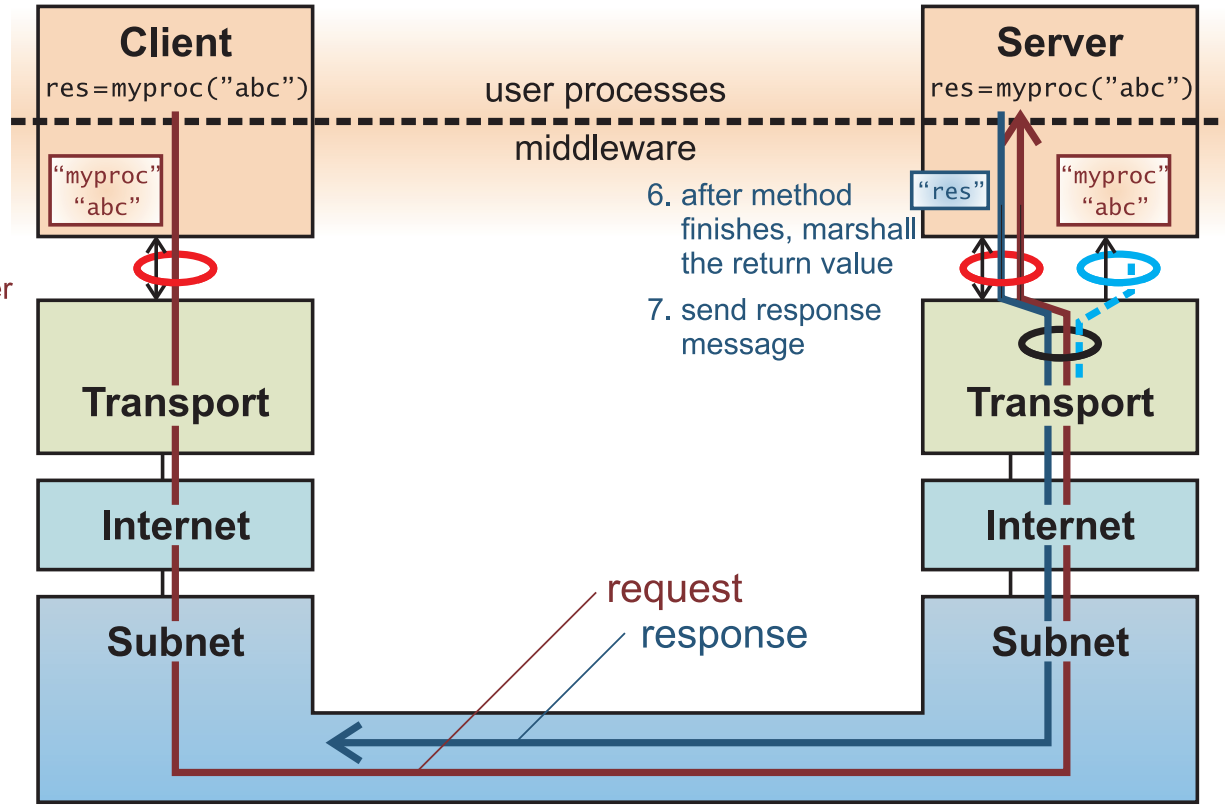


marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



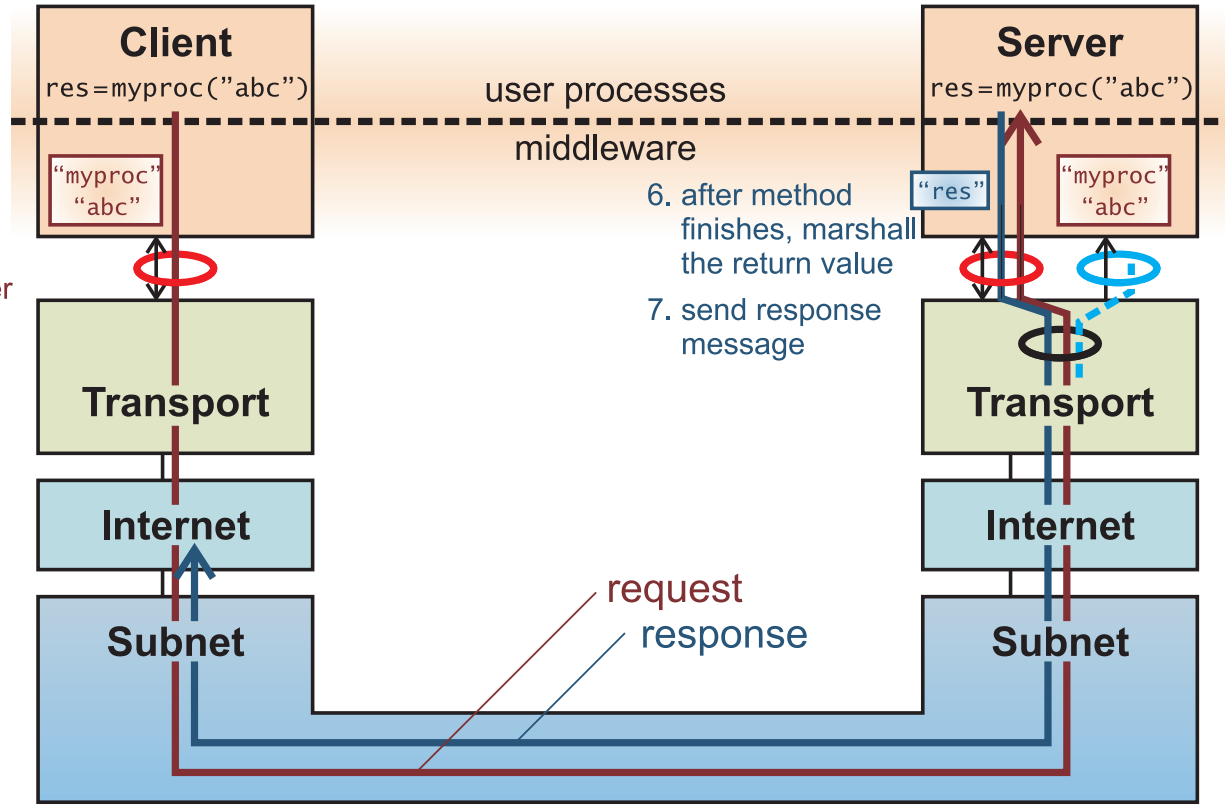
1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



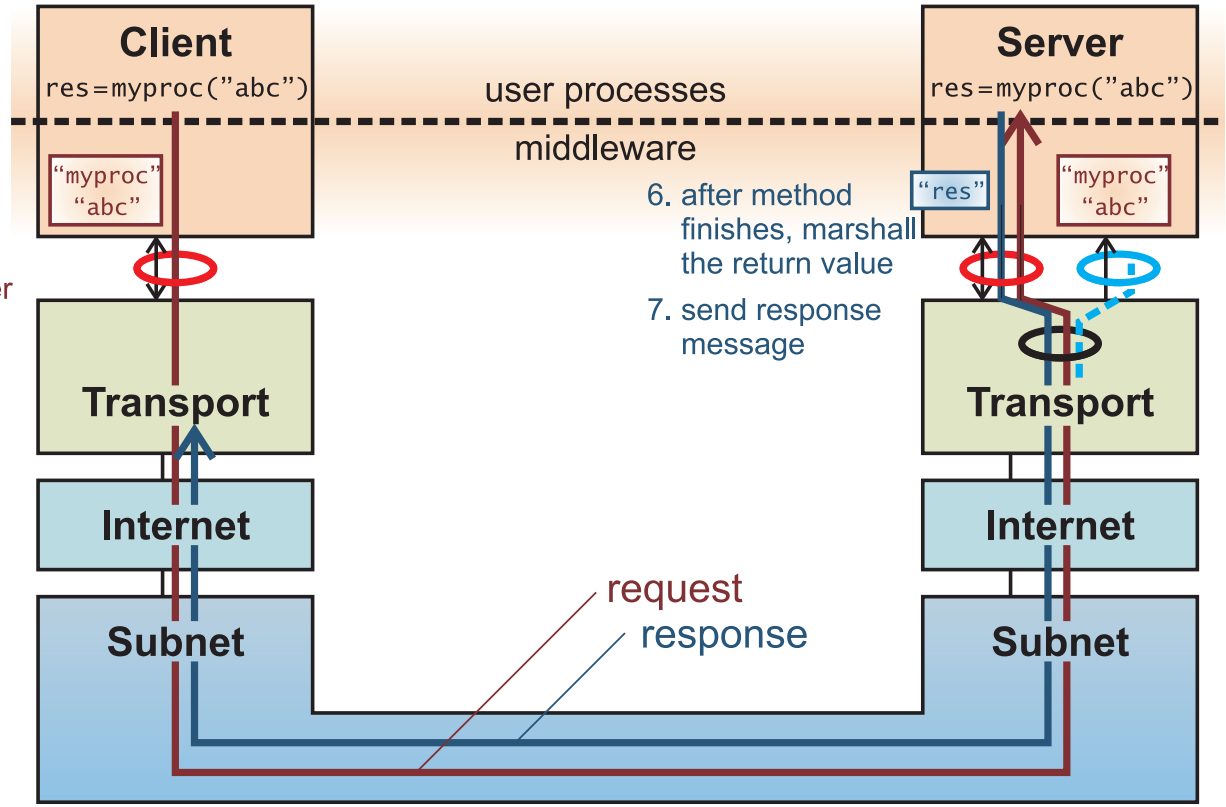
1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



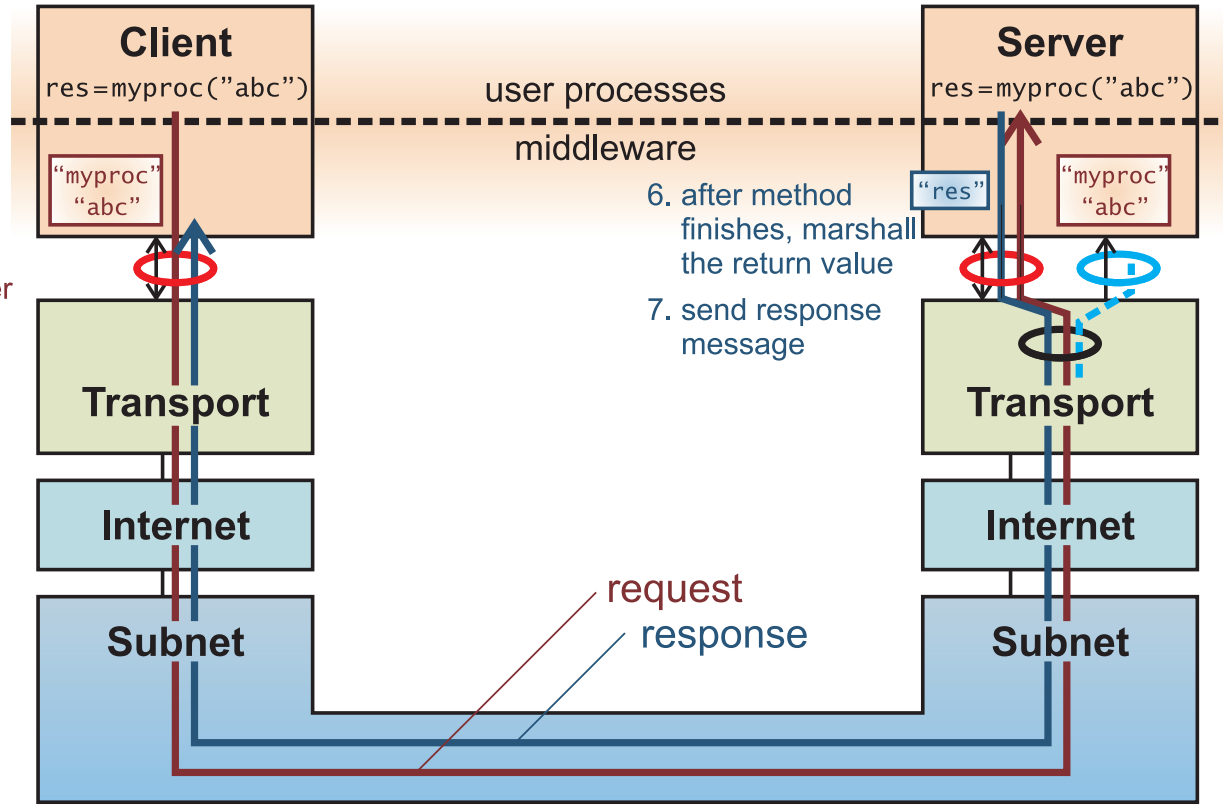
1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



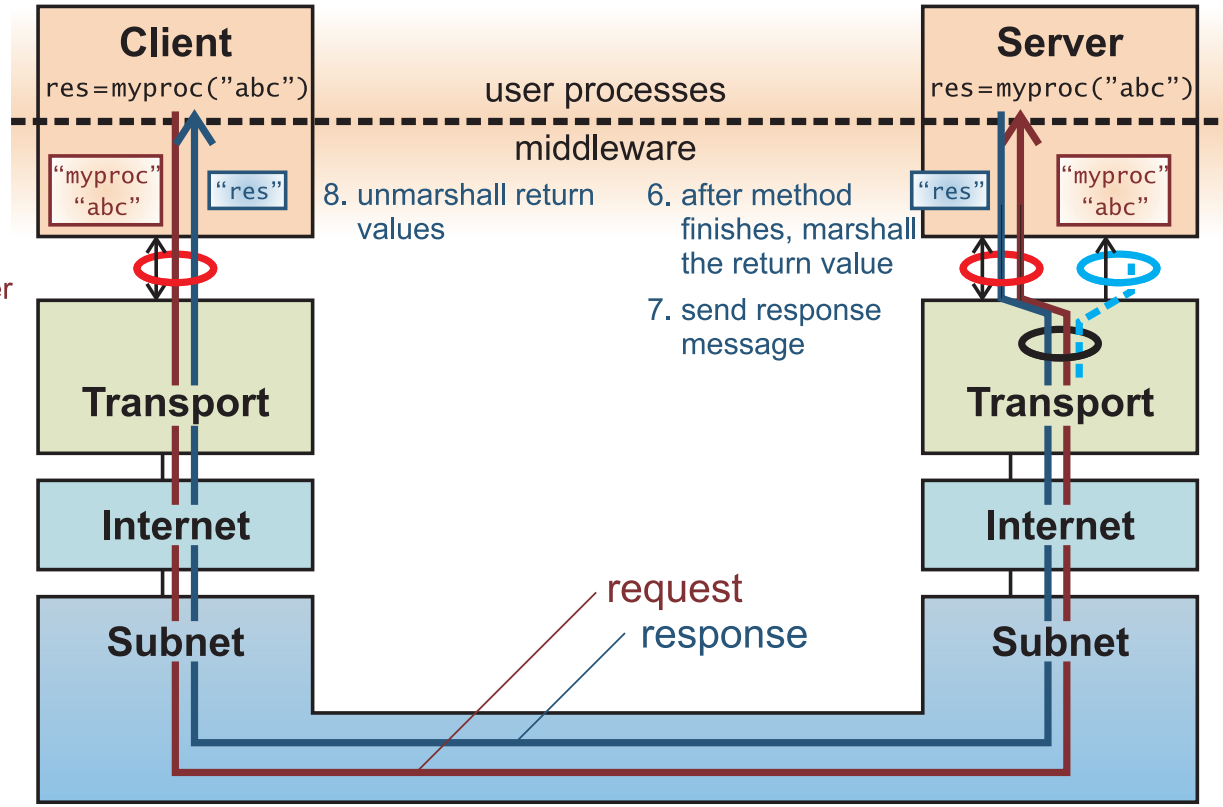
1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message

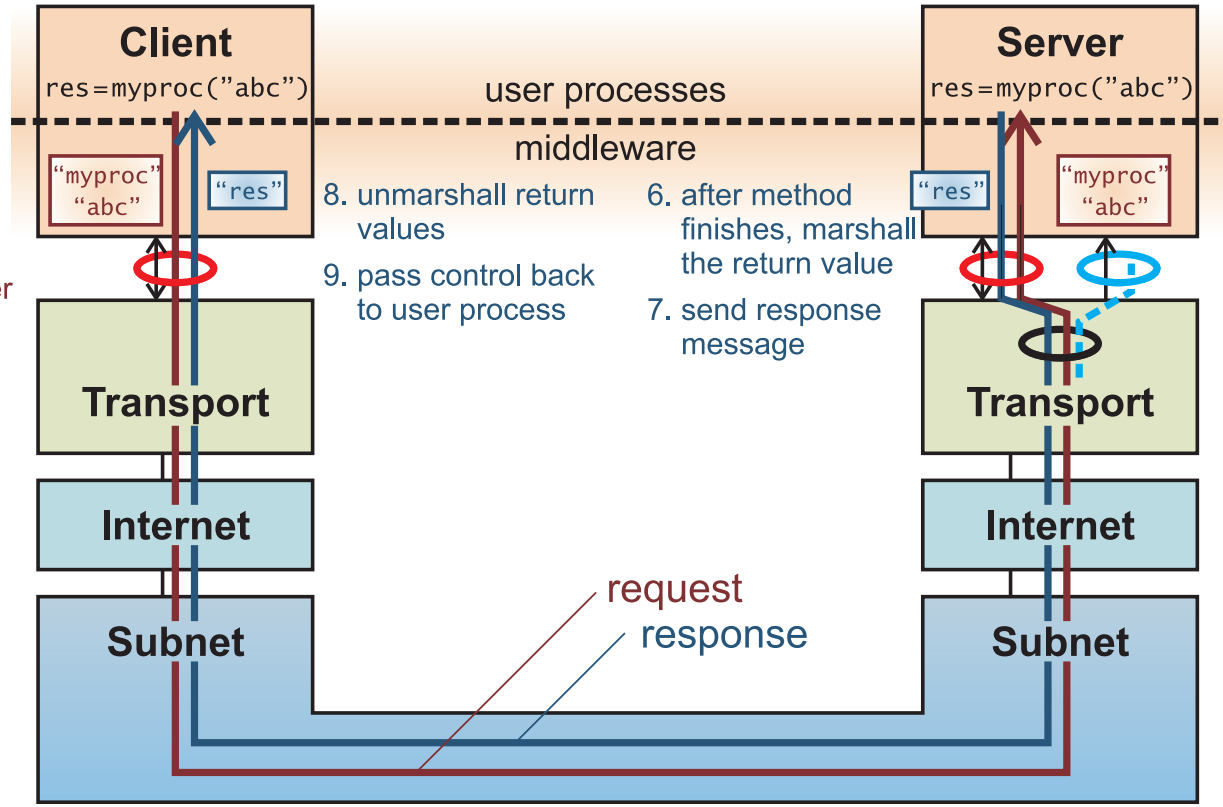
marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation



1. packing procedure id, and marshalled parameters into request message
2. send message to server
3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method
6. after method finishes, marshall the return value
7. send response message
8. unmarshall return values

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

1. packing procedure id, and marshalled parameters into request message
2. send message to server



3. map procedure/object id to a real method
4. unmarshall received parameters
5. call method

marshalling: translate procedure parameters into platform-independent representation
unmarshalling:translate marshalled data structures back to platform-specific representation

Principle

- The basic principle behind RPC are “Function Proxies”

Principle

- The basic principle behind RPC are "Function Proxies"
- Say, we want to perform a remote invocation of a function

```
double calculate(double x, double y)
```

Principle

- The basic principle behind RPC are “Function Proxies”
- Say, we want to perform a remote invocation of a function

```
double calculate(double x, double y)
```

- Goals

Principle

- The basic principle behind RPC are "Function Proxies"
- Say, we want to perform a remote invocation of a function

```
double calculate(double x, double y)
```

- Goals
  - Function which is implemented on another computer is actually executed on that other computer, but triggered by our process

Principle

- The basic principle behind RPC are "Function Proxies"
- Say, we want to perform a remote invocation of a function

```
double calculate(double x, double y)
```

- Goals
  - Function which is implemented on another computer is actually executed on that other computer, but triggered by our process
  - From the programmer's and program's perspective, it looks like it is called in the current process

Remember

- We can use sockets to realize some sort of *remote* procedure call...

Listing: [TCPServerStructuredData.java]: A server perform a specific "method".

```
import java.io.DataInputStream;   import java.io.DataOutputStream;
import java.net.ServerSocket;     import java.net.Socket;

public class TCPServerStructuredData {
  public static final void main(final String[] args) {
    ServerSocket       server;       Socket             client;
    DataOutputStream   dos;          DataInputStream    dis;
    String             s;            long               a, b,r;

    try {
      server = new ServerSocket(9996);//1 + 2)

      for (int j = 5; (--j) >= 0;) { //process only 5 clients, so I can show 5) below
        client = server.accept();    //3)

        dis = new DataInputStream(client.getInputStream()); //4 + 3
        s   = dis.readUTF();       //read an UTF-encoded string: the operation
        r   = a = dis.readLong(); //read a 64 bit long integer
        b   = dis.readLong();      //read another 64 bit long int
        if ("add".equalsIgnoreCase(s)) { r += b; } else { // add
          if ("sub".equalsIgnoreCase(s)) { r -= b;  }     // subtract
        }  //4 + 3)

        System.out.println(s + "(" + a + ",␣" + b + ")␣=␣" + r + "␣to␣" +
            client.getRemoteSocketAddress());

        dos = new DataOutputStream(client.getOutputStream()); //marshall output
        dos.writeLong(r); //write 64bit long integer: 4 + 3)
        dos.close(); // flush and close

        client.close(); //4)
      }
      server.close(); //5)
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Client Part of the Example

Listing: [TCPClientStructuredData.java]: A client asking the server to perform a specific "method".

```
import java.io.DataInputStream;   import java.io.DataOutputStream;
import java.net.InetAddress;      import java.net.Socket;

public class TCPClientStructuredData {

  public static final void main(final String[] args) {
    Socket           client;     InetAddress       ia;
    DataOutputStream dos;        DataInputStream   dis;

    try {
      ia = InetAddress.getByName("localhost");

      client = new Socket(ia, 9996); //1+2)

      dos = new DataOutputStream(client.getOutputStream()); //marshall data
      dos.writeUTF("sub"); //send operation name 3)
      dos.writeLong(9876); //send 64bit long integer
      dos.writeLong(1234); //send another 64bit long integer
      dos.flush();  //flush is important, otherwise stuff may just be buffered!

      dis = new DataInputStream(client.getInputStream()); // unmashall input
      System.out.println("Result: " + dis.readLong());    //3)

      client.close(); //4)
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Basic Idea

- Making a RPC framework by itself is not hard

Basic Idea

- Making a RPC framework by itself is not hard
- Client side

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name
  - sends error message back if none found

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name
  - sends error message back if none found
  - receives the procedure parameters

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name
  - sends error message back if none found
  - receives the procedure parameters
  - checks whether parameter types fits to procedure (send error if it doesn't fit)

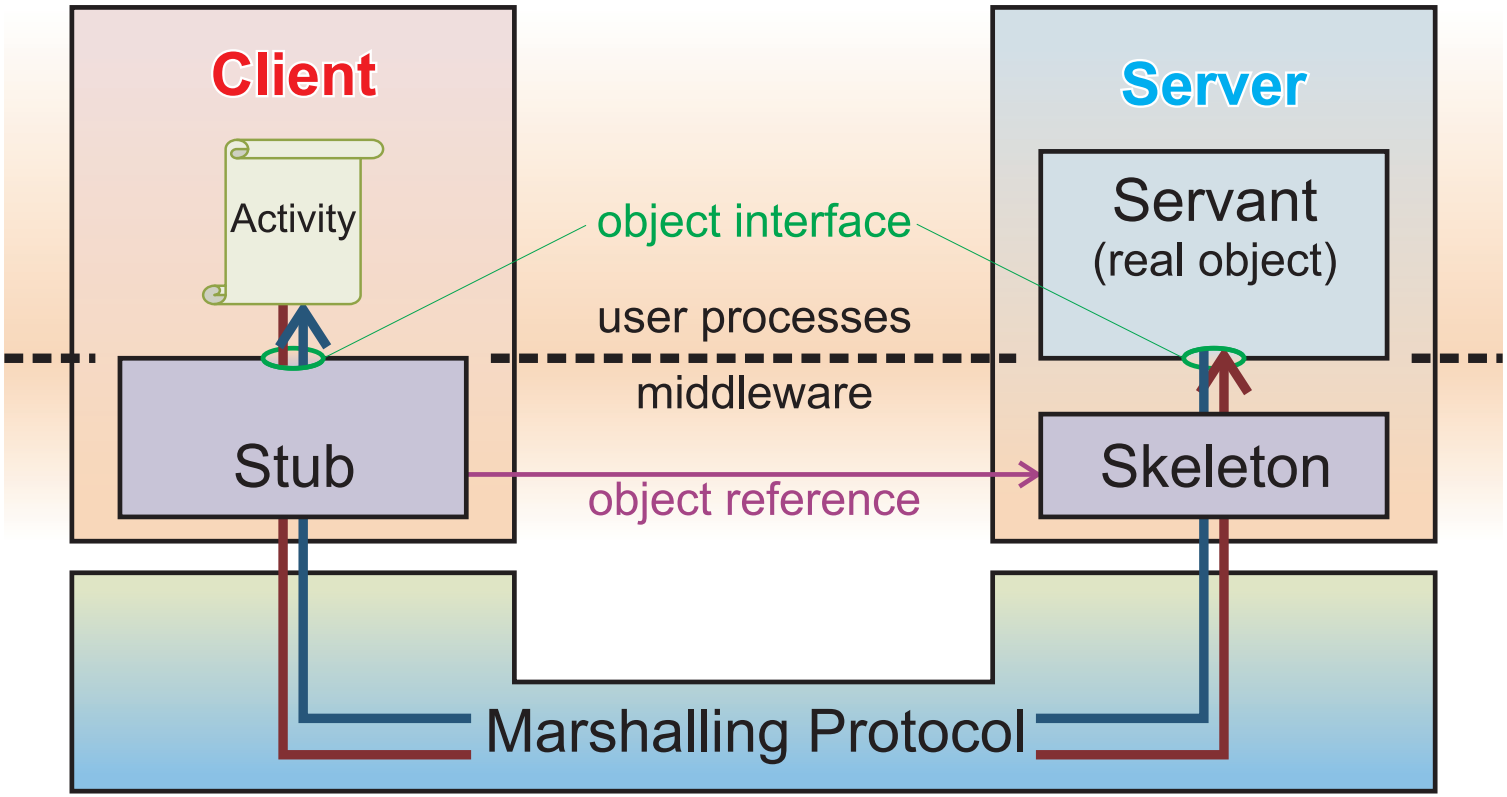
Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name
  - sends error message back if none found
  - receives the procedure parameters
  - checks whether parameter types fits to procedure (send error if it doesn't fit)
  - calls the procedure

Basic Idea

- Making a RPC framework by itself is not hard
- Client side: function stub (proxy) that can be called "normally"
  - send the procedure name
  - send the procedure parameters
  - wait for result...
- Server side: a skeleton function that
  - receives procedure name
  - finds the procedure implementation/pointer to code that fits to the name
  - sends error message back if none found
  - receives the procedure parameters
  - checks whether parameter types fits to procedure (send error if it doesn't fit)
  - calls the procedure
  - sends back the result

- Remote Method Invocation (RMI): RPC for distributed objects
- Stub: implements object interface [6, 7] and marshalls calls
- Skeleton: unmarshalls calls and delegates them to real object



RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
  - application servers: separation of concerns
- Advantages
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
  - application servers: separation of concerns
  - interconnect heterogeneous systems
- Advantages
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency

- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language

- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
  - security and access rights
- Disadvantages

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
  - security and access rights
- Disadvantages
  - security

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
  - security and access rights
- Disadvantages
  - security
  - always much slower than a local invocation

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
  - security and access rights
- Disadvantages
  - security
  - always much slower than a local invocation
  - complex implementation

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
  - versatile: anticipation of change
  - location transparency
  - geographical distribution
  - often independent from programming language
  - allows for replication of objects
  - security and access rights
- Disadvantages
  - security
  - always much slower than a local invocation
  - complex implementation
  - messages may get lost/modified/doubled

RMI: Pros and Cons

- What are the reasons, advantages, and disadvantages of remote method invocation?
- Reasons
- Advantages
- Disadvantages
  - security
  - always much slower than a local invocation
  - complex implementation
  - messages may get lost/modified/doubled
  - distributed garbage collection

Examples

- Common Object Request Broker Architecture (CORBA) [8–13]

Examples

- Common Object Request Broker Architecture (CORBA) [8–13]
- Distributed Component Object Model (DCOM) [14]

Examples

- Common Object Request Broker Architecture (CORBA) [8–13]
- Distributed Component Object Model (DCOM) [14]
- .NET Remoting [15]

Examples

- Common Object Request Broker Architecture (CORBA) [8–13]
- Distributed Component Object Model (DCOM) [14]
- .NET Remoting [15]
- Java Remote Method Invocation (RMI) [13, 16, 17]

Examples

- Common Object Request Broker Architecture (CORBA) [8–13]
- Distributed Component Object Model (DCOM) [14]
- .NET Remoting [15]
- Java Remote Method Invocation (RMI) [13, 16, 17]
- XPCOM (Cross Platform Component Object Model) [18]
- ...

Common Object Request Broker Architecture (CORBA) [9–13]

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations
- standardized by Object Management Group (OMG)

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations
- standardized by Object Management Group (OMG)
- fast, mature technology, language/Os independent, strong data types

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations
- standardized by Object Management Group (OMG)
- fast, mature technology, language/Os independent, strong data types
- very complex, standard very verbose: "design by committee"

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations
- standardized by Object Management Group (OMG)
- fast, mature technology, language/Os independent, strong data types
- very complex, standard very verbose: "design by committee"
- incompatibilities between implementations

RMI: CORBA

Common Object Request Broker Architecture (CORBA) [9–13]

- remote access to objects distributed over the system
- Interface Definition Language (IDL) to specify object interfaces
- mapping of IDL to programming languages: Code Generation
- application interacts with objects through Object Request Broker (ORB)
- naming service: maps names to locations
- standardized by Object Management Group (OMG)
- fast, mature technology, language/Os independent, strong data types
- very complex, standard very verbose: "design by committee"
- incompatibilities between implementations
- firewalls may block protocol

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)
- memory management via reference counting

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)
- memory management via reference counting
- interfaces described via IDL

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)
- memory management via reference counting
- interfaces described via IDL
- fast, mature technology, strong data types

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)
- memory management via reference counting
- interfaces described via IDL
- fast, mature technology, strong data types
- bound to Microsoft, supported mainly by Microsoft programming languages

RMI: DCOM

Distributed Component Object Model (DCOM) [14]

- developed by Microsoft
- object-oriented RPC system based on Distributed Computing Environment (DCE) and Compound Object Model (COM)
- memory management via reference counting
- interfaces described via IDL
- fast, mature technology, strong data types
- bound to Microsoft, supported mainly by Microsoft programming languages
- firewalls

RMI: .NET Remoting

.NET Remoting [15]

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework
- memory management via garbage collection

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework
- memory management via garbage collection
- fully transparent

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework
- memory management via garbage collection
- fully transparent
- can use both, TCP or SOAP[1] via HTTP (so no firewall problems)

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework
- memory management via garbage collection
- fully transparent
- can use both, TCP or SOAP[1] via HTTP (so no firewall problems)
- fast, mature technology, strong data types

[1] see a later lesson...

RMI: .NET Remoting

.NET Remoting [15]

- developed by Microsoft, replaces DCOM
- based on .NET Framework
- memory management via garbage collection
- fully transparent
- can use both, TCP or SOAP[1] via HTTP (so no firewall problems)
- fast, mature technology, strong data types
- bound to Microsoft, supported mainly by Microsoft programming languages

[1] see a later lesson...

Java Remote Method Invocation (RMI) [13, 16, 17]

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry
- names can be looked up and object references are obtained from the registry

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry
- names can be looked up and object references are obtained from the registry
- fast, mature technology, strong data types

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry
- names can be looked up and object references are obtained from the registry
- fast, mature technology, strong data types
- less complicated than .NET Remoting

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry
- names can be looked up and object references are obtained from the registry
- fast, mature technology, strong data types
- less complicated than .NET Remoting
- bound to Java, supported, firewalls

Java Remote Method Invocation (RMI) [13, 16, 17]

- developed by Sun / Java technology
- fully transparent
- objects bound to names in registry
- names can be looked up and object references are obtained from the registry
- fast, mature technology, strong data types
- less complicated than .NET Remoting
- bound to Java, supported, firewalls
- firewalls

Example for Java RMI

- Functionality specified as `interface`

Example for Java RMI

- Functionality specified as `interface`
- Servant object (on server side) implements the functionality

Example for Java RMI

- Functionality specified as `interface`
- Servant object (on server side) implements the functionality
- A name is assigned to the Servant and managed by a registry

Example for Java RMI

- Functionality specified as `interface`
- Servant object (on server side) implements the functionality
- A name is assigned to the Servant and managed by a registry
- Client can request access to the object from the registry

Example for Java RMI

- Functionality specified as `interface`
- Servant object (on server side) implements the functionality
- A name is assigned to the Servant and managed by a registry
- Client can request access to the object from the registry
- Stub on client side is an automatically generated instance of the interface

Example for Java RMI

- Functionality specified as `interface`
- Servant object (on server side) implements the functionality
- A name is assigned to the Servant and managed by a registry
- Client can request access to the object from the registry
- Stub on client side is an automatically generated instance of the interface
- Client can now access the object exactly as if it was a local object

RemotePrintInterface

Listing: [RemotePrintInterface.java]: The interface specifying the functionality.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

// the RemotePrint interface
public interface RemotePrintInterface extends Remote {

  /** A method to be remote-accessible
   * @param what the string to print
   * @throws RemoteException a possible exception */
  public abstract void print(final String what) throws
      RemoteException;

}
```

RemotePrintServer

Listing: [RemotePrintServer.java]: The server implementing the functionality.

```
import java.rmi.RemoteException;          import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;        import java.rmi.server.UnicastRemoteObject;

public class RemotePrintServer extends UnicastRemoteObject implements
    RemotePrintInterface {
  RemotePrintServer() throws RemoteException {
    super();
  }

 // the actual implementation of the method specified by RemotePrintInterface
  @Override
  public void print(final String what) throws RemoteException {
    System.out.println(what);
  }

  public static final void main(final String args[]) {
    Registry registry;

    try {
      // create the (local) object registry
      registry = LocateRegistry.createRegistry(9999);
      // bind the object to the name "server"
      registry.rebind("server", new RemotePrintServer());
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

RemotePrintClient

Listing: [RemotePrintClient.java]: The client remotely using the functionality.

```
import java.rmi.registry.LocateRegistry;      import java.rmi.registry.Registry;

public class RemotePrintClient { // the remote print rmi client
  public static final void main(final String args[]) {
    RemotePrintInterface rmiServer;         Registry registry;

    try {
      // find the (local) object registry
      registry = LocateRegistry.getRegistry(9999);

      // find the server object
      rmiServer = (RemotePrintInterface) (registry.lookup("server"));

      rmiServer.print("Hello World"); //$NON-NLS-1$
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

RemotePrintClientErroneous

Listing: [RemotePrintClientErroneous.java]: The client remotely using the functionality wrongly.

```
import java.rmi.registry.LocateRegistry;      import java.rmi.registry.Registry;

public class RemotePrintClientErroneous { // the erroneous remote print client
  public static final void main(final String args[]) {
    RemotePrintInterface rmiServer;         Registry registry;

    try {
      // find the (local) object registry
      registry = LocateRegistry.getRegistry(9999);

      //! invalid cast to server class !
      rmiServer = ((RemotePrintServer)(registry.lookup("server")));

      rmiServer.print("Hello World"); //$NON-NLS-1$
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

An erroneous client...

- Why?

An erroneous client...

- Why?
- Because the object instance returned by `registry.lookup` is a proxy

An erroneous client...

- Why?
- Because the object instance returned by `registry.lookup` is a proxy
- The real object instance (servant) exists in another JVM / on another computer

An erroneous client...

- Why?
- Because the object instance returned by `registry.lookup` is a proxy
- The real object instance (servant) exists in another JVM / on another computer
- The proxy on the client side is a dynamically created object implementing the `RemotePrintInterface` interface

An erroneous client...

- Why?
- Because the object instance returned by `registry.lookup` is a proxy
- The real object instance (servant) exists in another JVM / on another computer
- The proxy on the client side is a dynamically created object implementing the `RemotePrintInterface` interface
- It has a class different from `RemotePrintServer`

An erroneous client...

- Why?
- Because the object instance returned by `registry.lookup` is a proxy
- The real object instance (servant) exists in another JVM / on another computer
- The proxy on the client side is a dynamically created object implementing the `RemotePrintInterface` interface
- It has a class different from `RemotePrintServer`
- And thus, cannot be cast to `RemotePrintServer`

Summary

- RPC: call a procedure on another computer
- RMI: call a method (i.e., access an object) on a different computer
- Many different technologies
- Java RMI: quite simple to use

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. second edition. ISBN 9780133056990.
2. Peter Naur and Brian Randell, editors. *Software Engineering – Report on a Conference Sponsored by the NATO Science Committee*, Garmisch, Bavaria, Germany, October 7–11, 1968. Brussels, Belgium: NATO, Scientific Affairs Division. URL http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF. Chairman: Professor Dr. Fritz L. Bauer, Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms. Published: January 1969.
3. Edsger Wybe Dijkstra. On the role of scientific thought. Technical report, The Netherlands, Nuenen: Burroughs Corporation, August 30, 1974. URL http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html.
4. Edsger Wybe Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, Texts and Monographs in Computer Science, pages 60–65. Berlin, Germany: Springer-Verlag GmbH, 1982.
5. Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems (ACM TOCS)*, 2(1):39–59, February 1984. doi: 10.1145/2080.357392. URL http://nd.edu/~dthain/courses/cse598z/fall2004/papers/birrell-rpc.pdf.
6. Philip Shaw. *Java Interface Design FAQ*. Norwich, UK: Metacentric Internet Limited, One Percent Better, 1.0 edition, 2010. ISBN 978-1-907863-03-5. URL http://books.google.de/books?id=Lmqh1oNRpb0C.
7. Nell Dale, Daniel Joyce, and Chip Weems. *Object-Oriented Data Structures Using Java*. Sudbury, MA, USA: Jones & Bartlett Publishers, 2011. ISBN 1449613543 and 9781449613549. URL http://books.google.de/books?id=GEJ_Jp6mUpgC.
8. David Chappell. The trouble with corba, May 1998. URL http://www.davidchappell.com/articles/article_Trouble_CORBA.html.
9. Robert Orfali, Dan Harkey, and Daniel J. Edwards. *Client/Server Survival Guide*. Wiley Computer Publishing. New York, NY, USA: John Wiley & Sons Ltd., 1999. ISBN 0471316156 and 9780471316152. URL http://books.google.de/books?id=fBteTDjMwScC.
10. USA: Electronic Data Systems Corporation (EDS) et al. Plano, TX. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 1: CORBA Interfaces*, volume formal/2011-11-01. Needham, MA, USA: Object Management Group (OMG), November 2011. URL http://www.omg.org/spec/CORBA/3.2/Interfaces/PDF/.
11. USA: Electronic Data Systems Corporation (EDS) et al. Plano, TX. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 2: CORBA Interoperability*, volume formal/2011-11-02. Needham, MA, USA: Object Management Group (OMG), November 2011. URL http://www.omg.org/spec/CORBA/3.2/Interoperability/PDF.

Bibliography II

12. USA: Electronic Data Systems Corporation (EDS) et al. Plano, TX. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 3: CORBA Component Model*, volume formal/2011-11-03. Needham, MA, USA: Object Management Group (OMG), November 2011. URL `http://www.omg.org/spec/CORBA/3.2/Components/PDF`.
13. Josef Stepisnik. *Distributed Object-Oriented Architectures: Sockets, Java RMI and CORBA*. Hamburg, Germany: Diplomica Verlag GmbH, 2007. ISBN 3836650339 and 9783836650335. URL `http://books.google.de/books?id=qNGTzYdJt18C`.
14. *[MS-DCOM]: Distributed Component Model (DCOM) Remote Protocol Specification*, volume v20111214. Redmond, WA, USA: Microsoft Corporation, December 14, 2011. URL `http://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-DCOM%5D.pdf`.
15. Dino Esposito. .net remoting – design and develop seamless distributed applications for the common language runtime. *Microsoft Developer Network – MSDN Magazin*, 2(10), October 2002. URL `http://msdn.microsoft.com/en-us/magazine/cc188927.aspx`.
16. Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the java system. In Douglas C. Schmidt and Doug Lea, editors, *Proceedings of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS)*, Toronto, ON, Canada, 1996. URL `http://pdos.csail.mit.edu/6.824/papers/waldo-rmi.pdf`.
17. William Grosso. *Java RMI*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2011. ISBN 1449315356 and 9781449315351. URL `http://books.google.de/books?id=TeK5uL2dWwQC`.
18. Nigel McFarland. *Rapid Application Development With Mozilla*. Bruce Perens' Open Source Series. Upper Saddle River, NJ, USA: Prentice Hall Professional, 2004. ISBN 0131423436 and 9780131423435. URL `http://books.google.de/books?id=gKeRXPkSAVMC`.