

# РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Организация связи между компонентами

# Роль связи в РВС

Взаимодействие базируется на протоколах.

**Протокол** – это набор правил и соглашений, описывающий процедуру взаимодействия между компонентами системы.

# СТЕК ПРОТОКОЛОВ OSI



# ОРГАНИЗАЦИЯ ОБМЕНА СООБЩЕНИЯМИ

5

## ◎ Прямая передача сообщений

- ◎ возможна только если принимающая сторона готова к приему сообщения в этот момент времени

## ◎ Использование менеджера сообщений

- ◎ компонента высылает сообщение в очередь менеджера, из которой, в дальнейшем, принимающая сторона извлекает полученное сообщение

# ПРЯМАЯ ПЕРЕДАЧА СООБЩЕНИЙ: СОКЕТЫ

# ПРЯМАЯ ПЕРЕДАЧА СООБЩЕНИЙ: СОКЕТЫ

7

- ⊙ Используется на основе интерфейса сокетов
- ⊙ Т.е. используется непосредственно транспортный уровень в виде Middleware.
- ⊙ **Сокет** – абстрактный объект, представляющий конечную точку соединения.
- ⊙ **Сокет TCP/IP** – комбинация IP-адреса и номера порта, например 10.10.10.10:80.
- ⊙ Интерфейс сокетов впервые появился в BSD Unix.

# ПРИМЕР РЕАЛИЗАЦИИ СОКЕТА

8

Язык C# поддерживает два типа сетевых соединений:

- ⊙ серверные, реализуемые с помощью объектов класса TcpListener;
- ⊙ клиентские, реализуемые с помощью объектов класса TcpClient.

# ОБЪЕКТЫ TCPLISTENER И TCPCLIENT

- ◎ Объект класса TcpListener позволяет **только прослушивать** определенный порт компьютера.
- ◎ **Любые процессы передачи** данных через этот сокет **осуществляются с использованием объекта TcpClient.**
- ◎ TcpClient возвращается методом `AcceptTcpClient()` класса TcpListener, что обеспечивает сам процесс прослушивания порта.

# ПРИМЕР СОЗДАНИЯ СЕРВЕРА

```
using System.Net;
using System.Net.Sockets;

Int32 port = 13000;

IPAddress localAddr =
    IPAddress.Parse("127.0.0.1");

TcpListener server = new
    TcpListener(localAddr, port);

server.Start();

//Начинаем прослушивание порта
TcpClient client = server.AcceptTcpClient();
//После подключения создаем поток сообщений
NetworkStream stream = client.GetStream();
```

# ОБМЕН СООБЩЕНИЯМИ

## Запись сообщений

```
Byte[] bytes=new Byte[256];  
String data = "text";  
  
bytes =  
    System.Text.Encoding.UTF8.  
    GetBytes(data);  
  
stream.Write(bytes, 0,  
    bytes.Length);
```

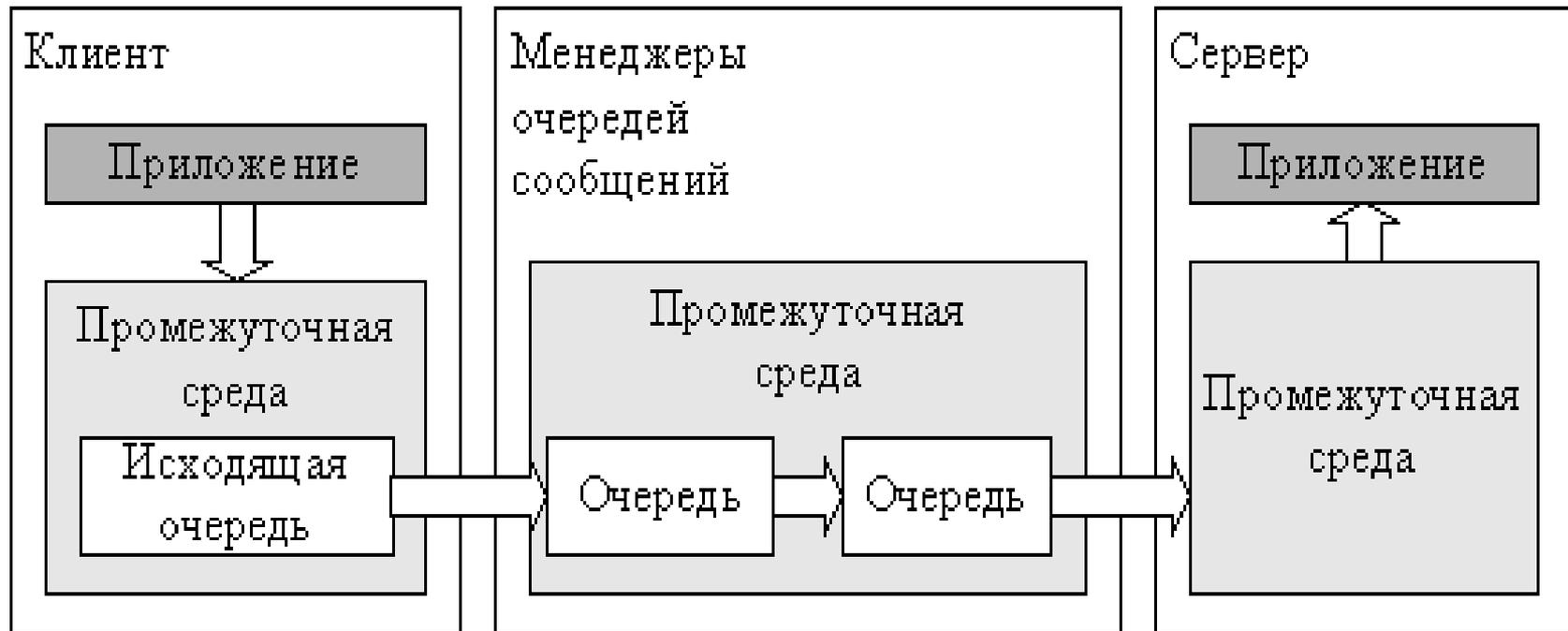
## Чтение сообщений

```
Byte[] bytes=new Byte[256];  
String data = null;  
  
int i = stream.Read(bytes, 0,  
    bytes.Length);  
  
data=System.Text.Encoding.UTF8.  
    GetString(bytes,0, i);
```

# ВЗАИМОДЕЙСТВИЕ ПОСРЕДСТВОМ МЕНЕДЖЕРА СООБЩЕНИЙ

# ИСПОЛЬЗОВАНИЕ МЕНЕДЖЕРА СООБЩЕНИЙ

13



# ИСПОЛЬЗОВАНИЕ МЕНЕДЖЕРА СООБЩЕНИЙ

Использование очередей сообщений ориентировано на **асинхронный** обмен данными.

## Достоинства

- ⊙ время функционирования сервера не связано со временем работы клиентов;
- ⊙ независимость промежуточной среды от средства разработки компонент и используемого языка программирования;
- ⊙ считывать и обрабатывать заявки из очереди могут несколько независимых компонент, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы.

# ИСПОЛЬЗОВАНИЕ МЕНЕДЖЕРА СООБЩЕНИЙ

## Недостатки

- ⊙ необходимость явного использования очередей распределенным приложением;
- ⊙ сложность реализации **синхронного** обмена;
- ⊙ определенные накладные расходы на использование менеджеров очередей;
- ⊙ сложность получения ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

# УДАЛЕННЫЙ ВЫЗОВ ПРОЦЕДУР

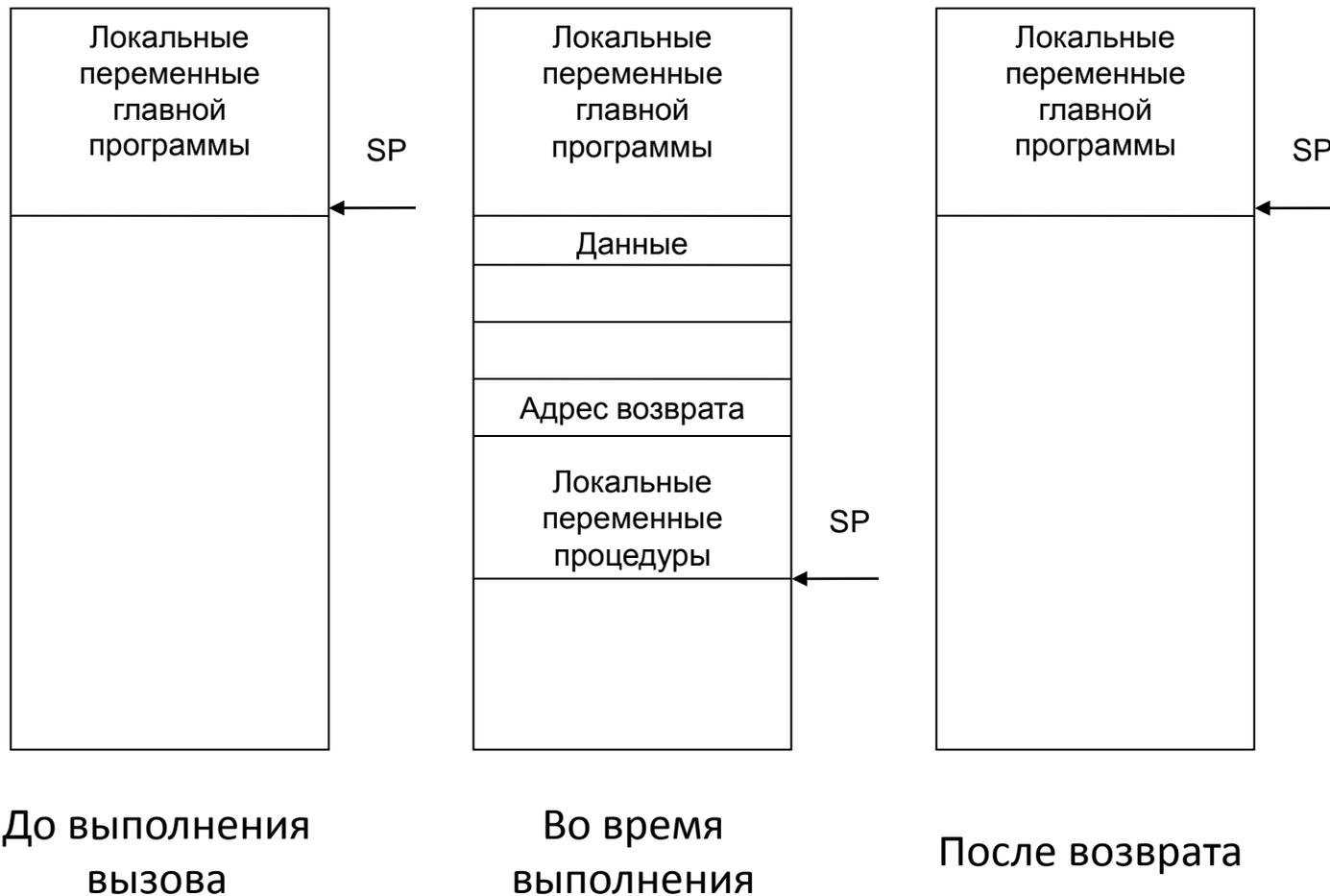
## REMOTE PROCEDURE CALL - RPC

## REMOTE METHOD INVOCATION - RMI

# Технология RPC

- ◎ **Удаленный вызов процедур** (от англ. Remote Procedure Call, RPC) — технология, позволяющая компьютерным программам вызывать функции или процедуры в другом адресном пространстве.

# СОСТОЯНИЕ СТЕКА ПРИ ВЫЗОВЕ ЛОКАЛЬНОЙ ПРОЦЕДУРЫ

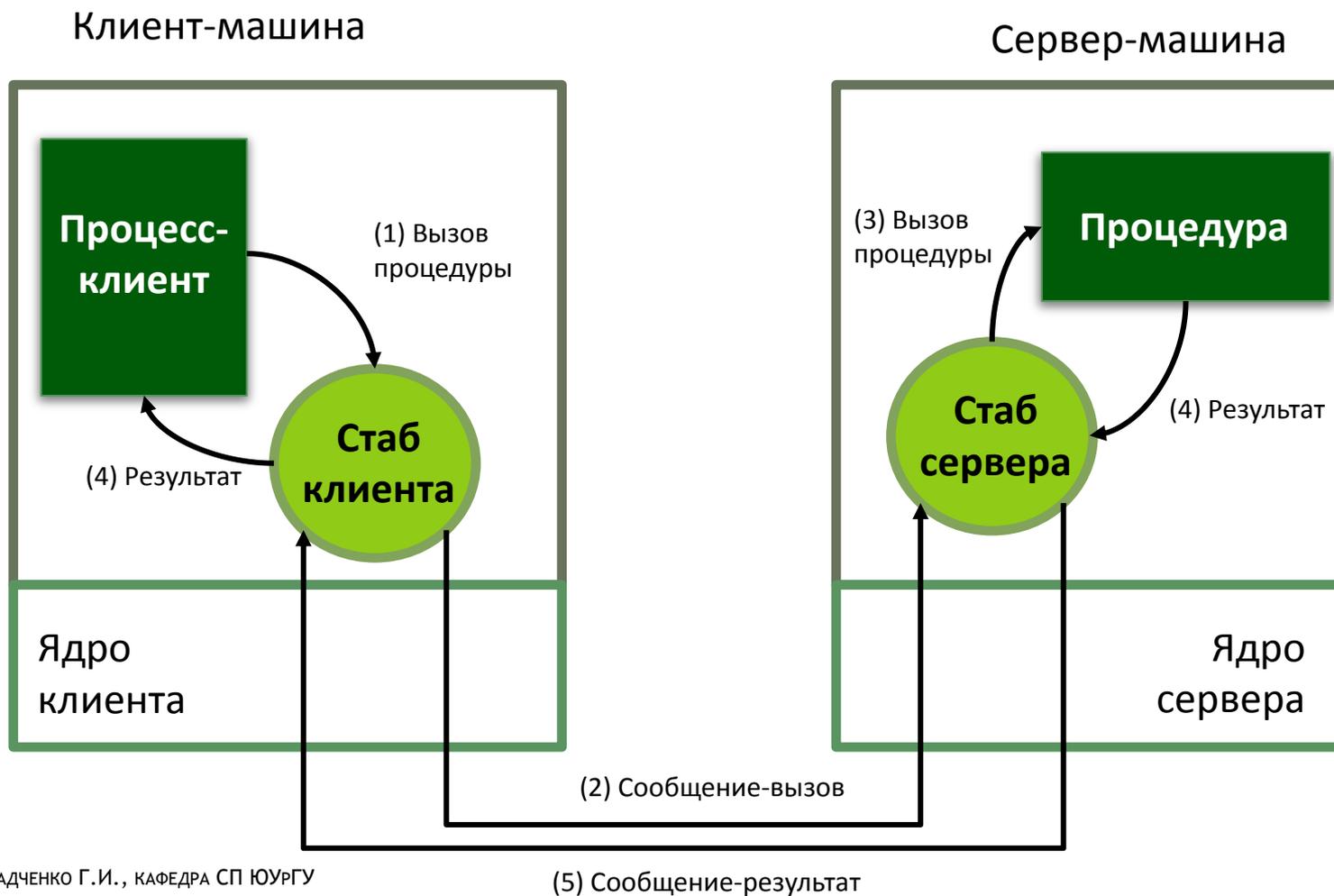


# РЕАЛИЗАЦИЯ RPC

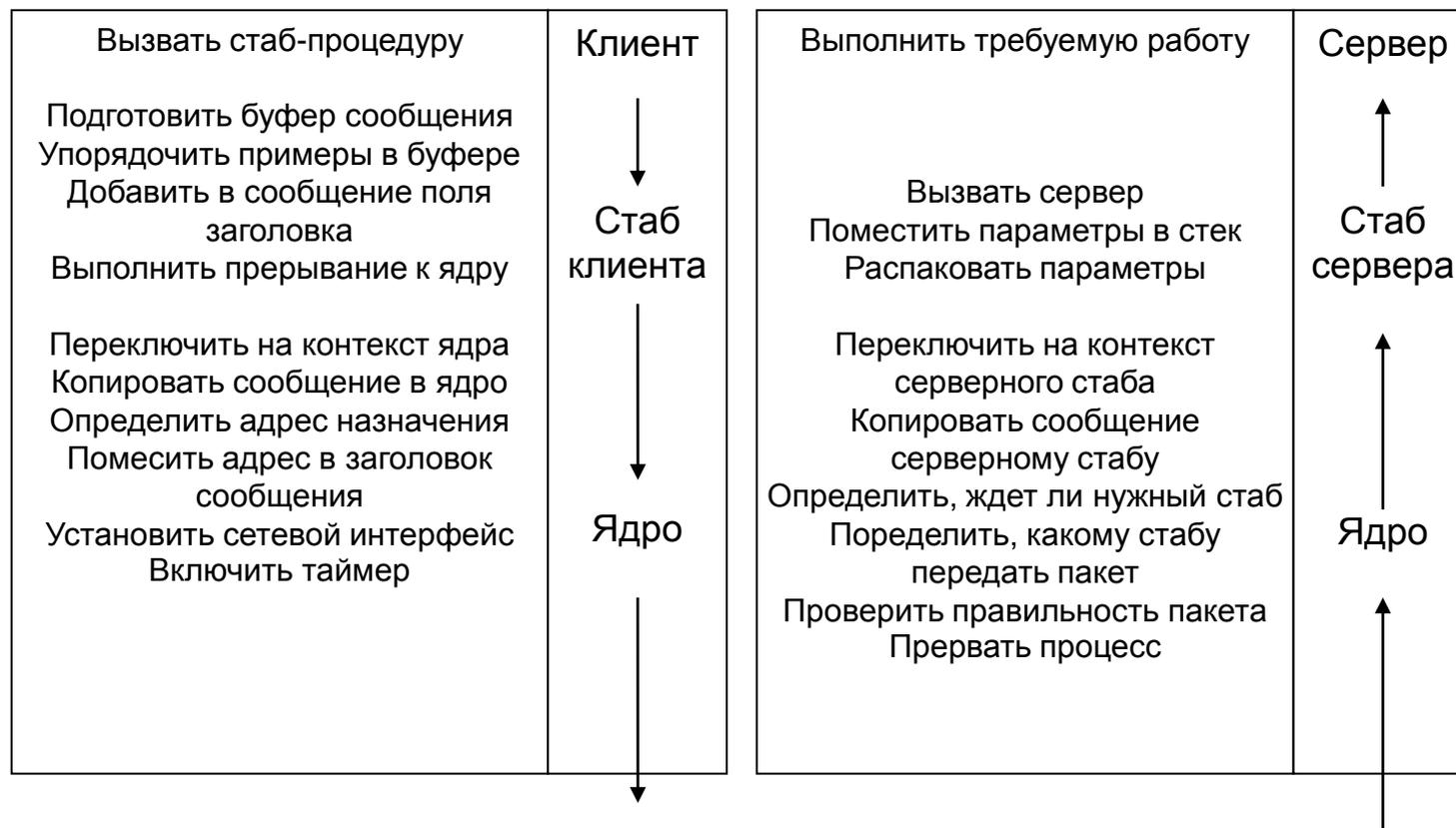
- ◎ Идея: вызов удаленной процедуры «прозрачен» для локального процесса
- ◎ Вместо локальной процедуры помещается «клиентский **стаб**» (**stub** – заглушка).
- ◎ Он вызывается также, как и локальная процедура, но вместо исполнения производит передача сообщения ядру удаленной машины.

# ПОРЯДОК ВЫЗОВА УДАЛЕННОЙ ПРОЦЕДУРЫ

20



# ЭТАПЫ ВЫПОЛНЕНИЯ RPC



# УДАЛЕННЫЙ ВЫЗОВ МЕТОДОВ

С точки зрения ООП была реализована концепция использования удаленных объектов Remote Method Invocation (RMI).

- ◎ **RMI** позволяет обеспечить прозрачный доступ к методам удаленных объектов, обеспечивая
  - ◎ доставку параметров вызываемого метода,
  - ◎ сообщение объекту о необходимости выполнения метода
  - ◎ и передачу возвращаемого значения клиенту обратно

# УДАЛЕННЫЙ ОБЪЕКТ

- ◎ Удаленный объект – это совокупность некоторых данных, определяющих его состояние. Это состояние можно изменить путем вызова некоторых его методов.
- ◎ Методы и поля объекта, которые могут использоваться через удаленные вызовы доступны через **внешний интерфейс** класса объекта.

# ПОСРЕДНИК (PROXY) И КАРКАС

24

- ◎ Клиентская заглушка для вызова удаленного объекта называется **посредником (proxy)**.
- ◎ **Посредник** реализует тот же интерфейс, что и удаленный объект.
- ◎ Заглушка на стороне **сервера** называется **каркасом (skeleton в Java RMI)**
- ◎ **Каркас** связывается с определенным экземпляром удаленного объекта и вызывает необходимый метод с требуемыми параметрами

# МАРШАЛИЗАЦИЯ

- ◎ Для передачи параметров по сети используется **маршализация** (marshalling) - процесс преобразования параметров для передачи их между процессами при удаленном вызове
- ◎ Маршализация
  - ◎ **по ссылке** – экземпляр удаленного объекта находится на сервере и не покидает его, а для доступа используются посредники
  - ◎ **по значению** – удаленный объект сериализуется и его копия передается в другой процес

# ИСПОЛЬЗОВАНИЕ УДАЛЕННОГО ОБЪЕКТА

