Распределенные вычислительные системы

Организация связи между компонентами

Вопросы

- Как расшифровывается и в чем состоит принцип архитектуры NUMA?
- Каким образом использование конвейерных вычислений ускоряет решение научных задач?
- Назовите основные отличительные принципы облачных вычислений.
- Какой компонент РВС не является ресурсом?
- Назовите основные виды прозрачности РВС.
- Назовите 3 признака классификации РВС.

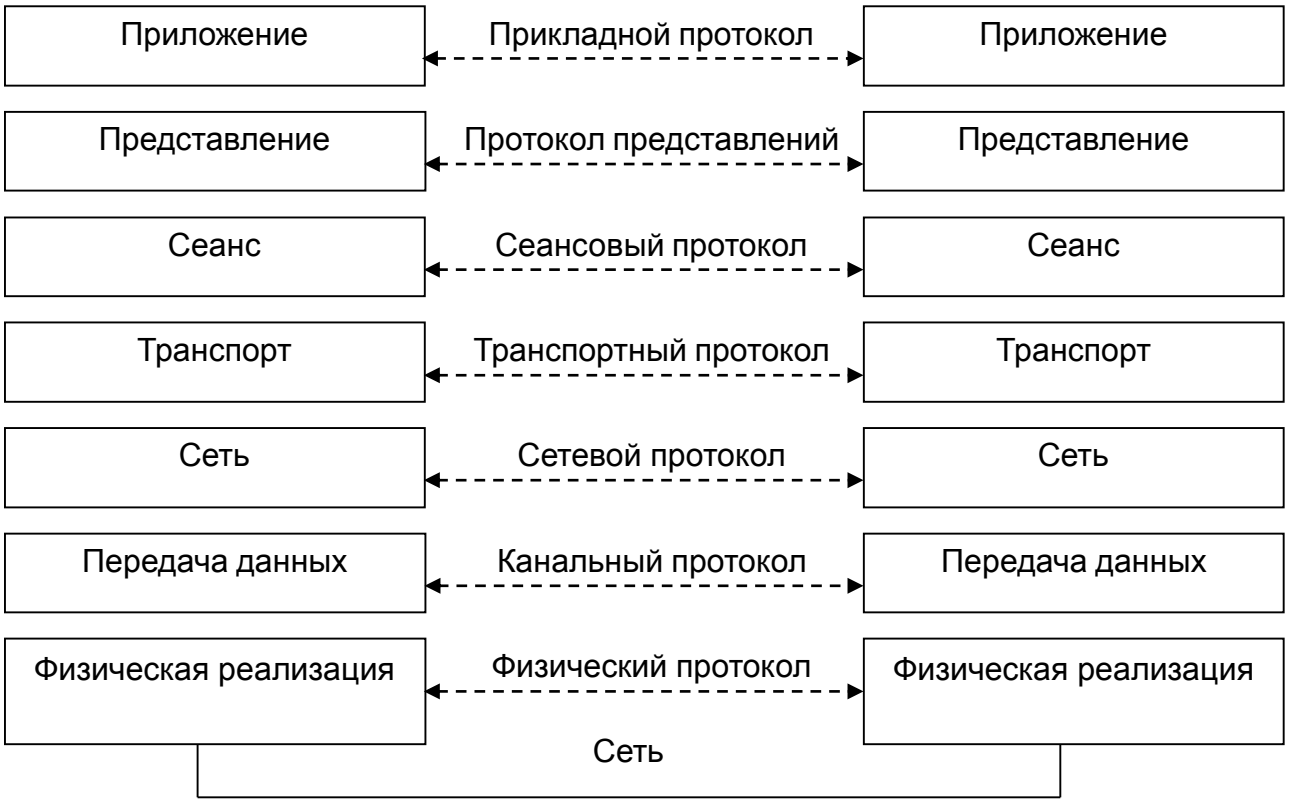
Роль связи в РВС

Протокол

Взаимодействие базируется на протоколах.

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

Стек протоколов OSI



FTP, SMTP, HTTP

XML, JPEG, ASCII, сжатие

PPTP, SMPP

TCP, UDP

IP, IPsec

IEEE 802.11, PPP, PPPoE

Bluetooth, DSL, IRDA

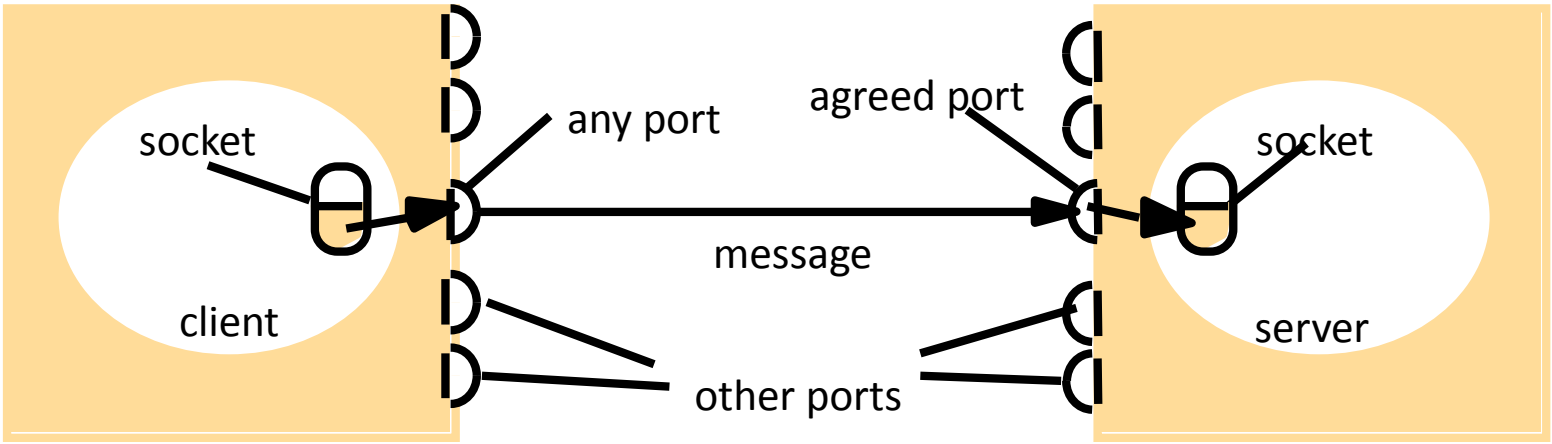
Организация обмена сообщениями

- **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

- Т.е. используется непосредственно транспортный уровень в виде Middleware.



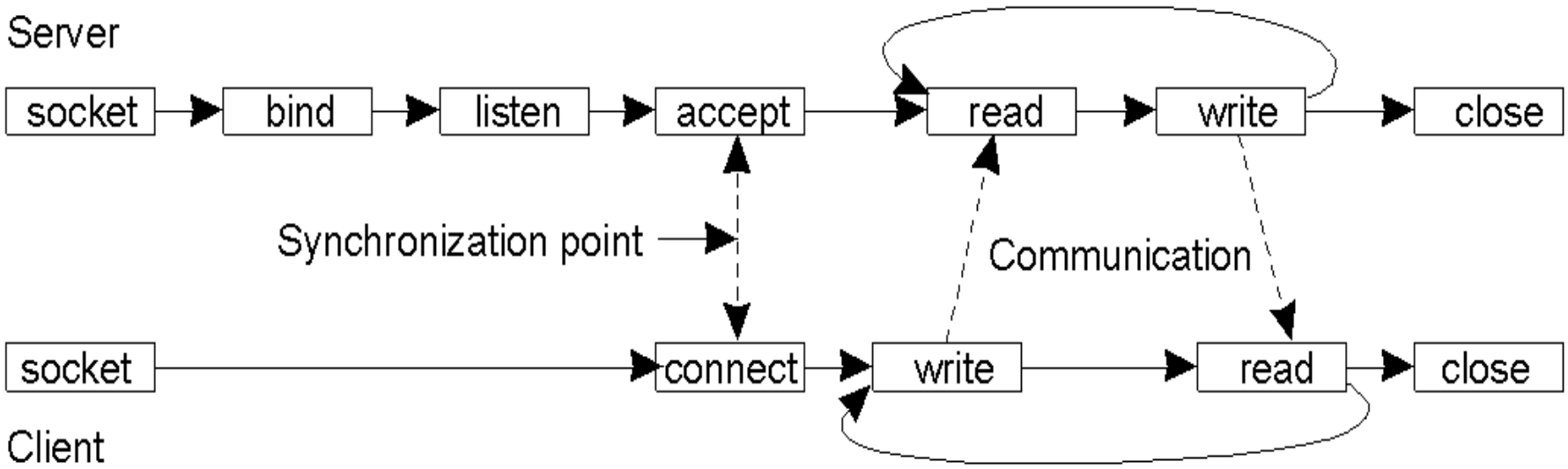
- **Сокет –** абстрактный объект, представляющий конечную точку соединения, обеспечивающий прием и передачу сообщений внешнему (локальному или удаленному) процессу.
- **Сокет TCP/IP –** комбинация IP-адреса и номера порта, например 10.10.10.10:80.
- Интерфейс сокетов впервые появился в BSD Unix.

Berkeley Sockets API (1)

Socket primitives for TCP/IP.

Primitive	**Meaning**
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Berkeley Sockets (2)



Пример реализации сокета

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

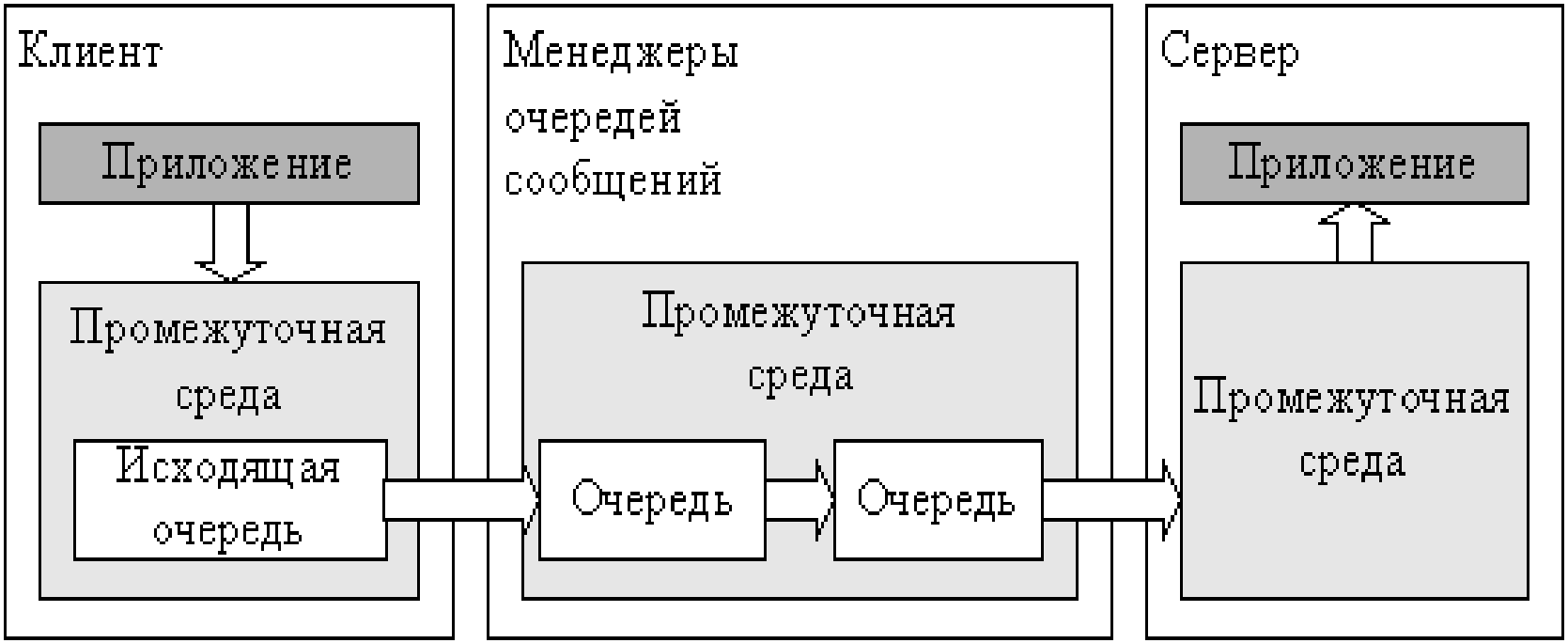
```
Byte[] bytes=new Byte[256];

String data = “text”;

bytes =
  System.Text.Encoding.UTF.Ge
  tBytes(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.UTF
  8.GetString(bytes,0, i);
```

Взаимодействие посредством менеджера сообщений

Использование менеджера сообщений



Использование менеджера сообщений

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

Достоинства

- **Слабосвязанность** программных компонентов обеспечивается посредством единого интерфейса данных
- **Избыточность данных:** парадигма “put-get-delete” позволяет избежать потери данных, даже если они не были обработаны после их получения
- **Масштабируемость и эластичность:** считывать и обрабатывать заявки из очереди могут несколько независимых процессов одновременно. При этом, если один из таких процессов падает, любой другой может взять на себя задачу обработки сообщений
- **Очередность**: сообщения попадают в очередь одно за другим, и, соответственно, обрабатываются в порядке поступления.
- **Буферизация:** если время на обработку сообщения больше, чем время поступления новых сообщений, очередь буферизует новые сообщения и они не теряются.
- **Асинхронность взаимодействия:** время работы клиента и сервера не зависят друг от друга. Клиент может отправить сообщение и продолжить свою работу не дожидаясь ответа.

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.

Службы очередей сообщений

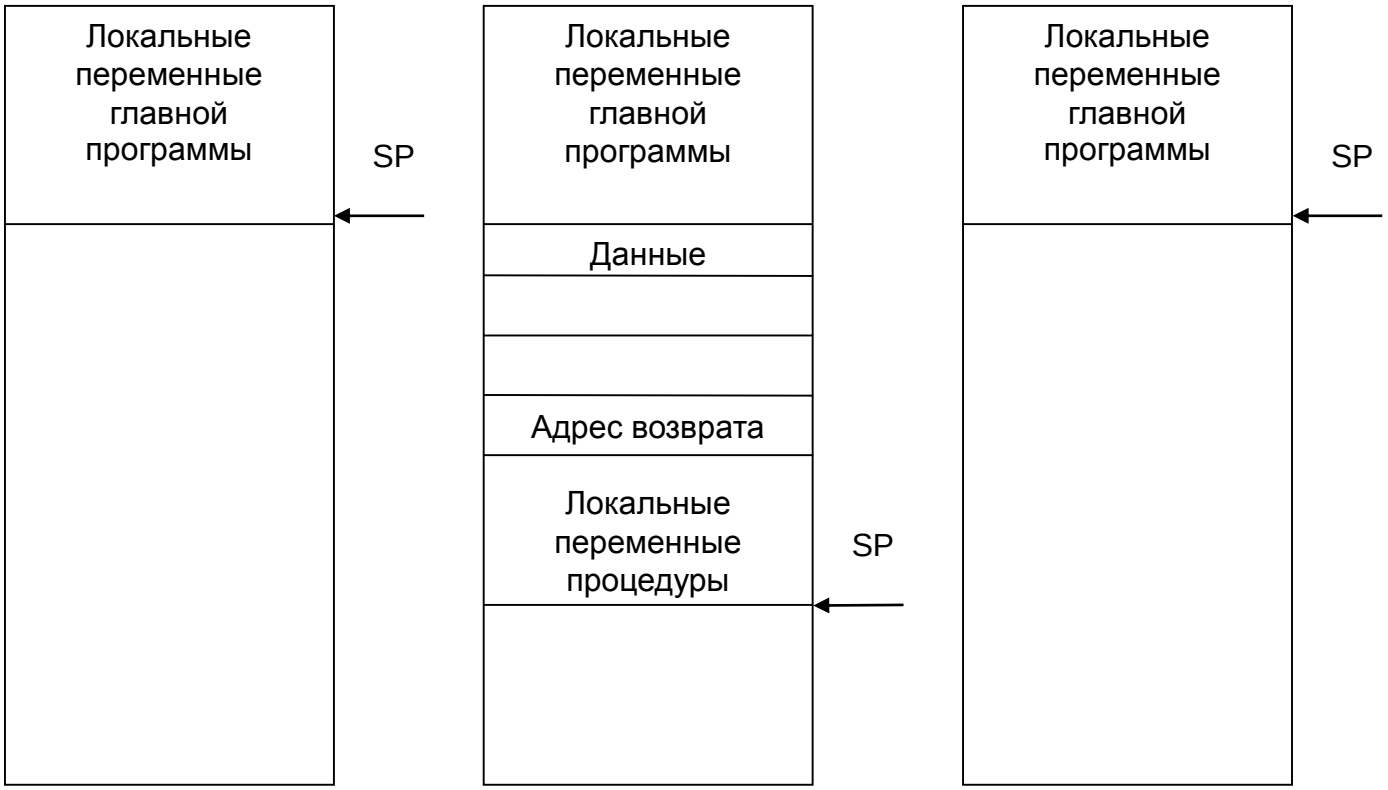
- Службы очередей сообщений (Message Queue Services, MQS) используются в разработке начиная с 1980-х
- IBM WebSphere MQ (~8 000 $ на 100 процессоров).
- Microsoft Message Queuing (MSMQ)
- RabbitMQ (http://www.rabbitmq.com/)
  - Robust messaging for applications
  - Easy to use
  - Runs on all major operating systems
  - Supports a huge number of developer platforms
  - Open source and commercially supported

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



До выполнения вызова

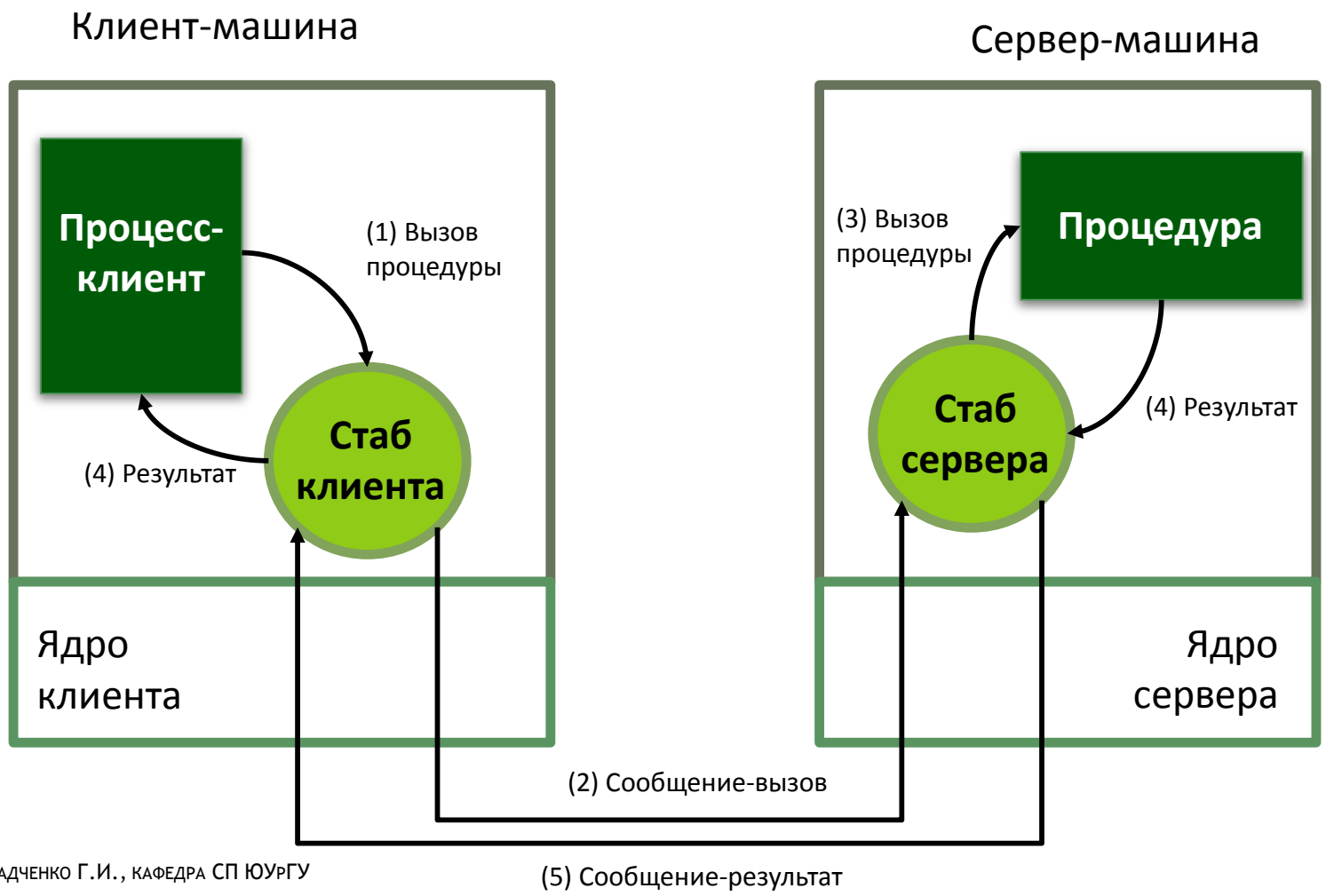
Во время выполнения

После возврата

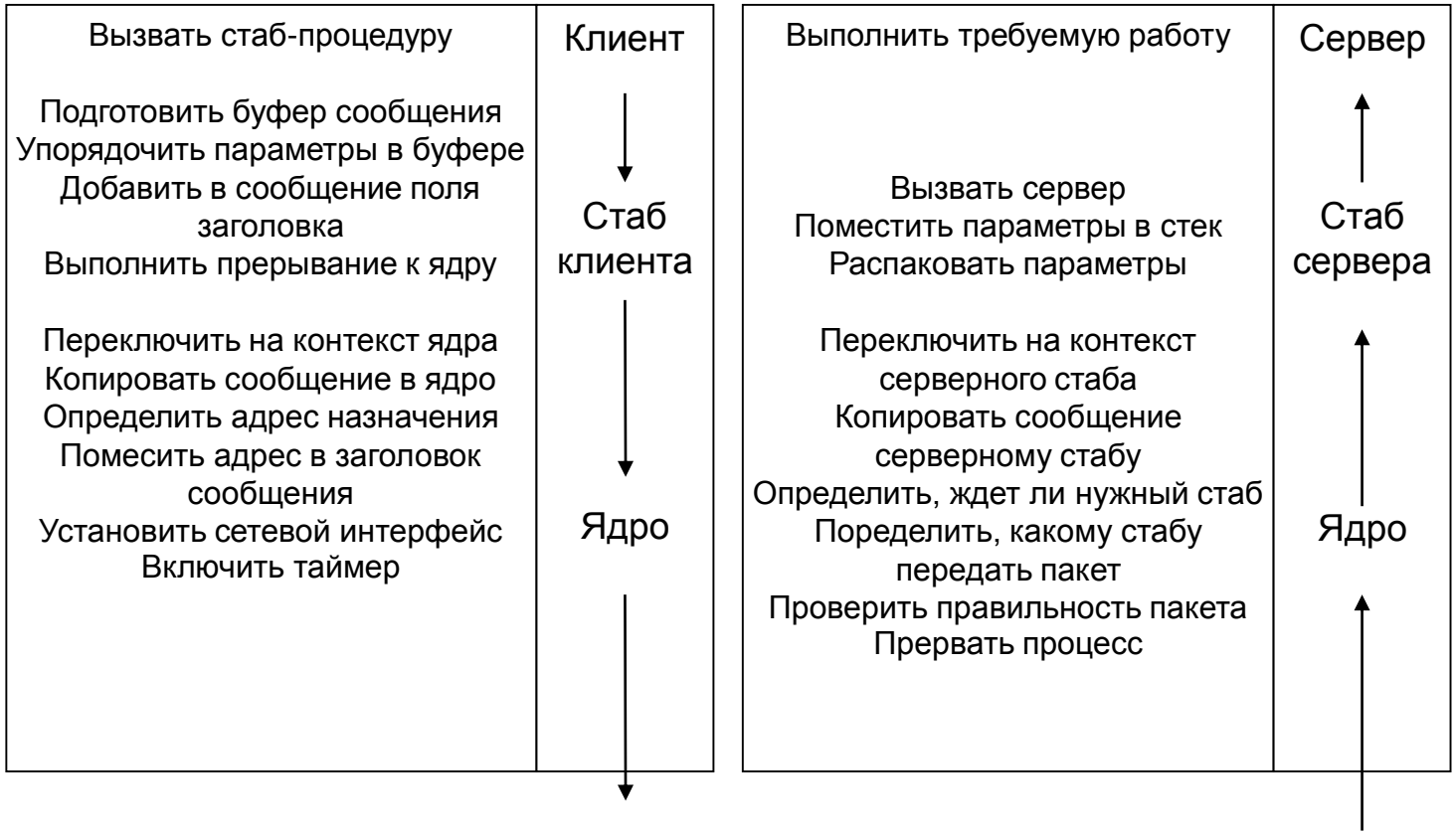
Реализация RPC

- 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



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

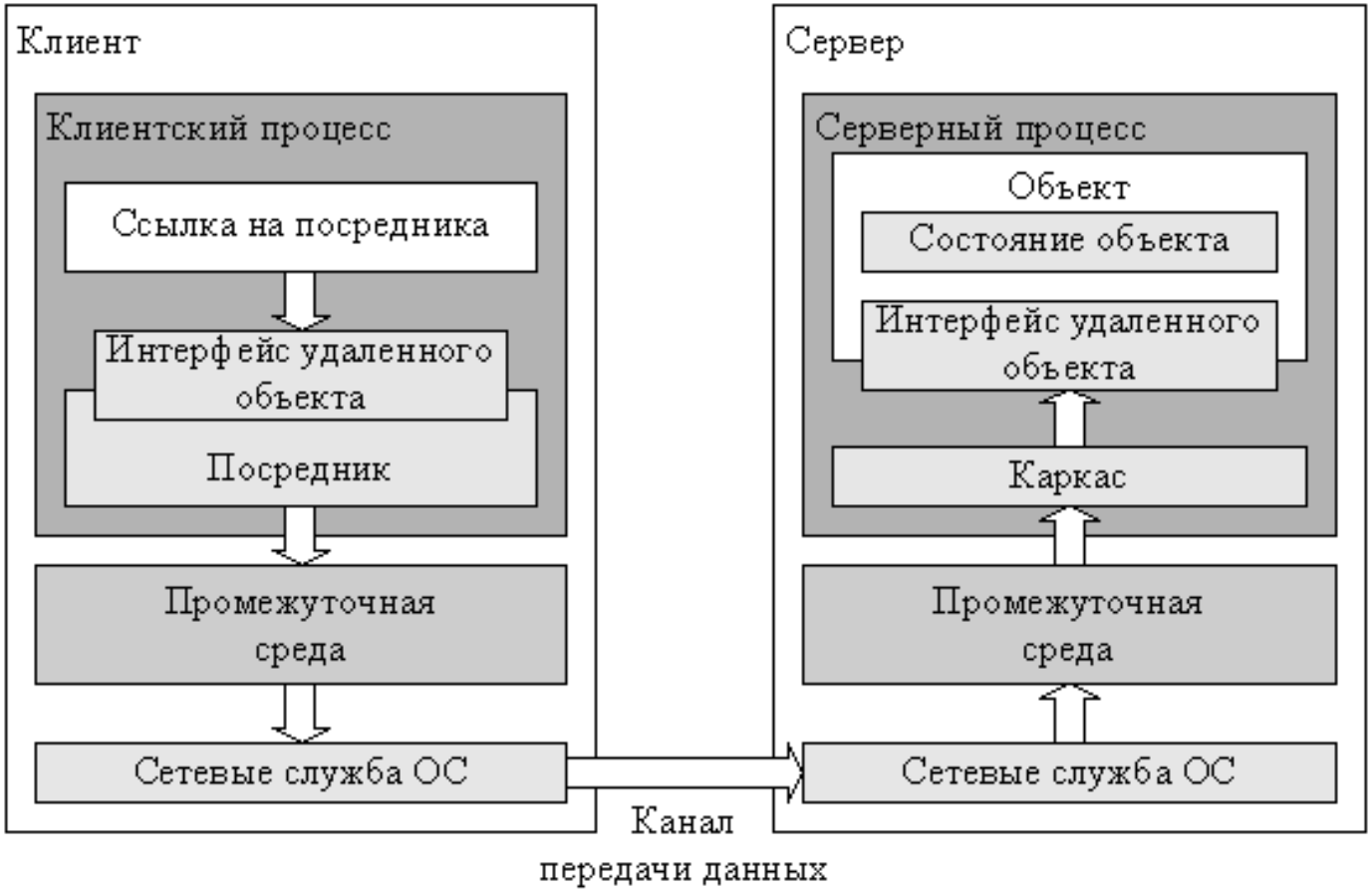
Удаленный объект

- 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

- 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

Использование удаленного объекта



Итого

- **Протокол –** это набор правил и соглашений, описывающий процедуру взаимодействия между компонентами системы.
- Существуют варианты прямой передачи сообщений в PBC и использования менеджеров сообщений.
- Технология RPC используется для вызова функций или процедур в другом адресном пространстве
- Технология RMI – развитие RPC, обеспечивает прозрачный доступ к методам удаленных объектов