Распределенные вычислительные системы

Организация связи между компонентами

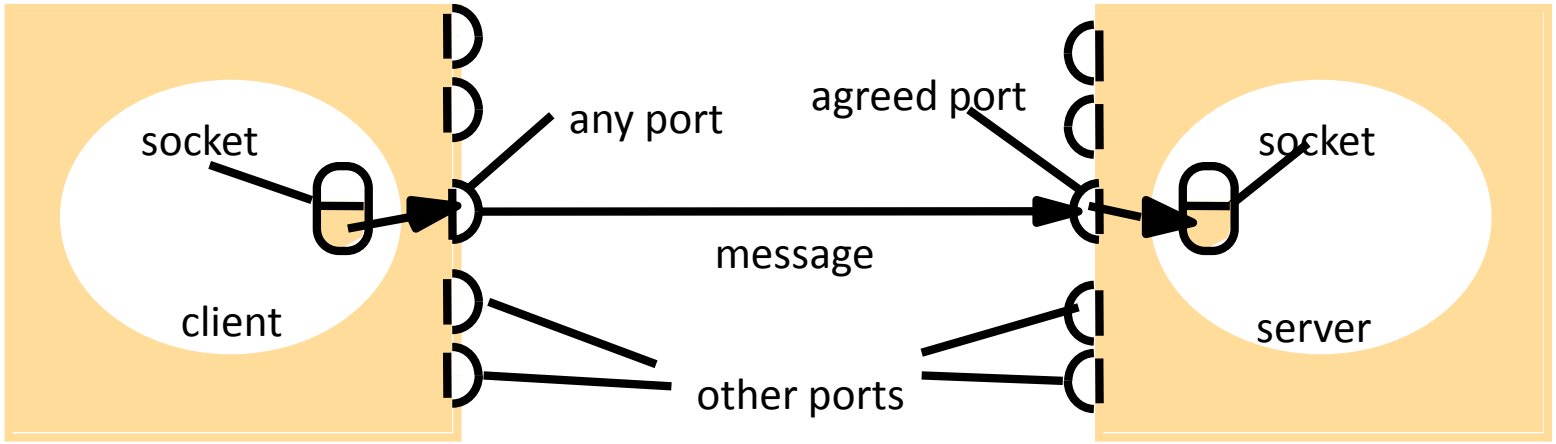
Организация обмена сообщениями

- **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.



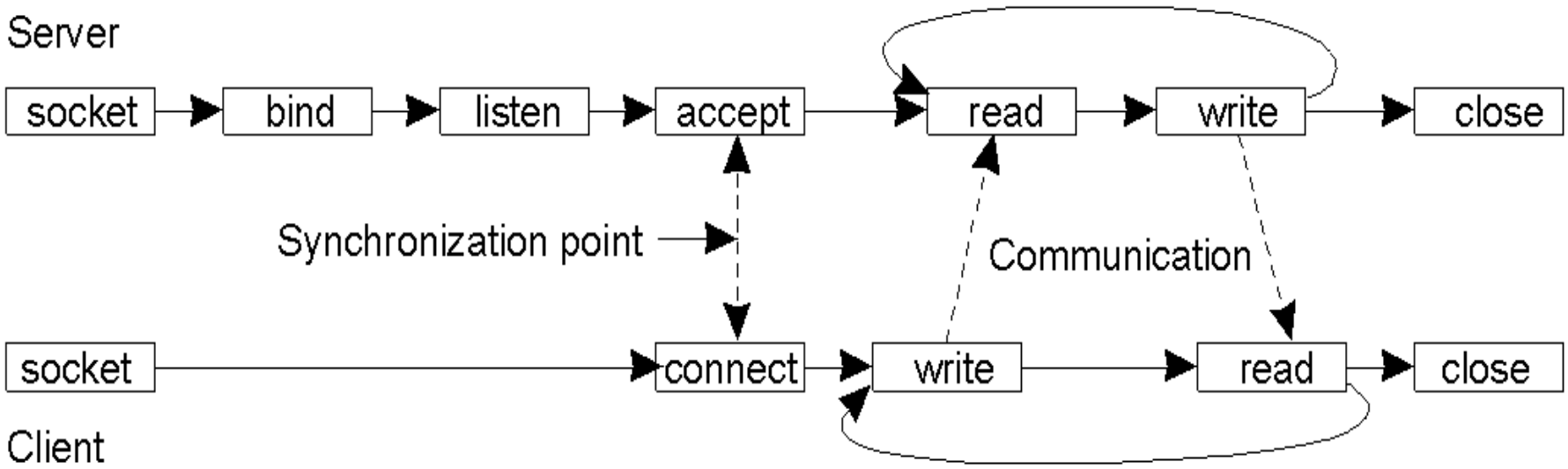
- **Сокет –** абстрактный объект, представляющий конечную точку соединения, обеспечивающий прием и передачу сообщений внешнему (локальному или удаленному) процессу.
- **Сокет TCP/IP –** комбинация IP-адреса и номера порта, например 10.10.10.10:80.
- Интерфейс сокетов впервые появился в BSD Unix.

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);
```

RPC-remote procedure call
RMI-remote method invocation

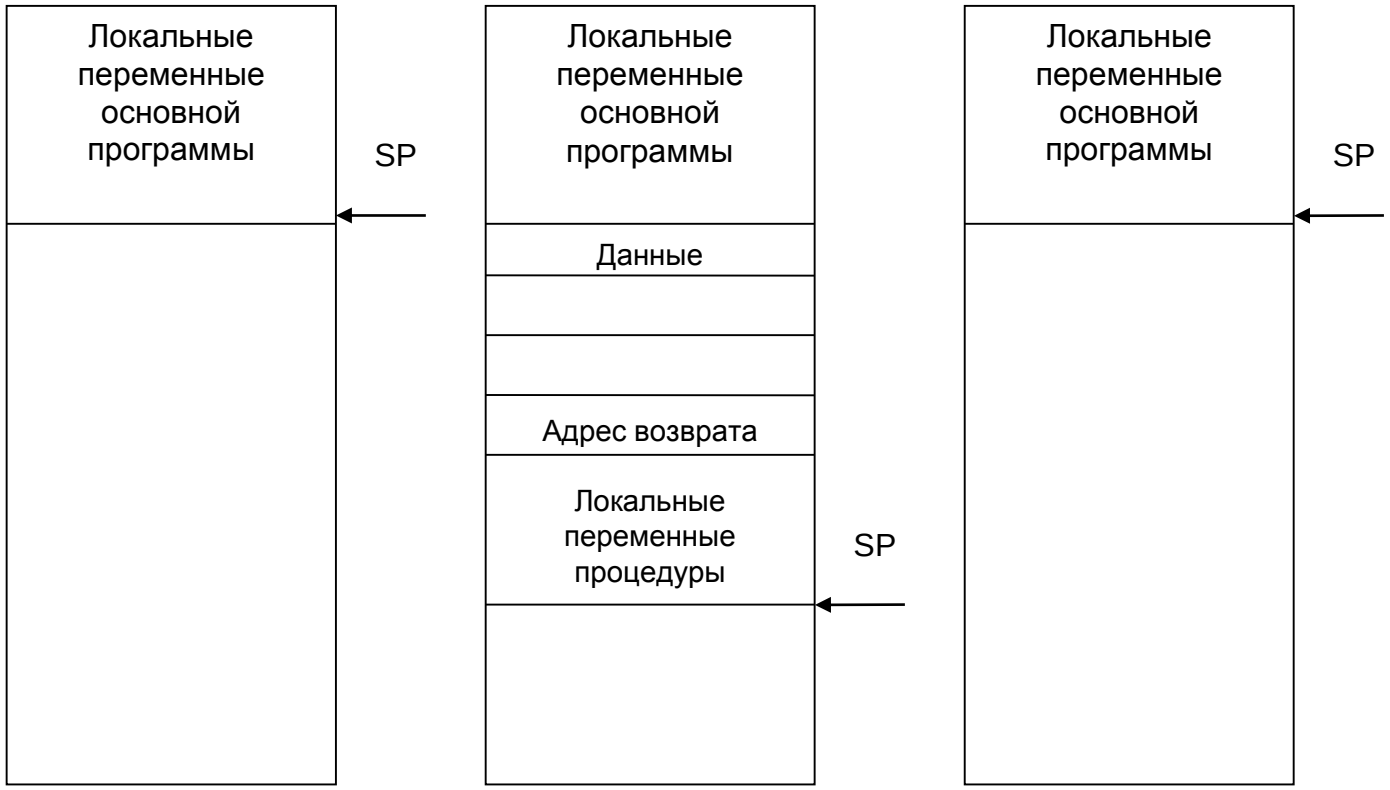
Технологии удаленного вызова

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

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

Вызов локальной процедуры



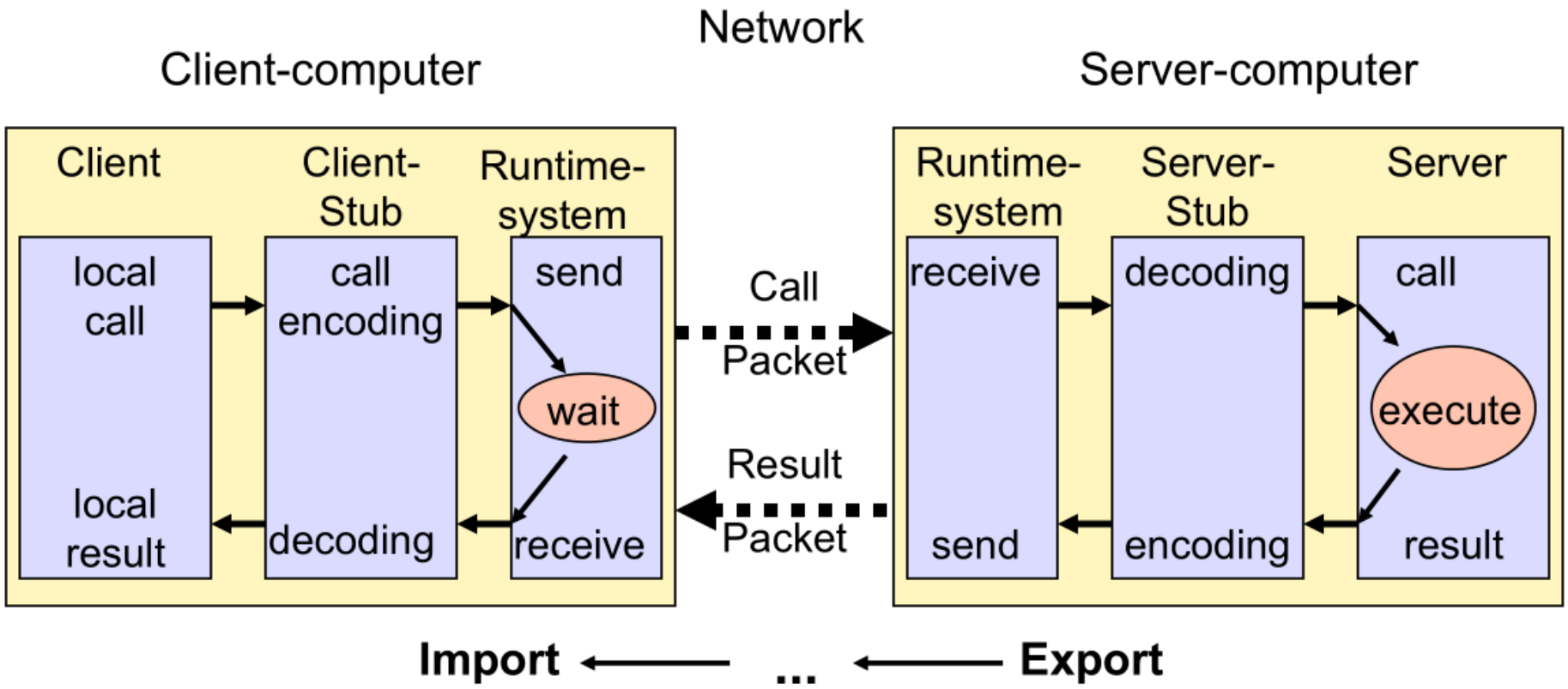
До выполнения вызова

Во время выполнения вызова

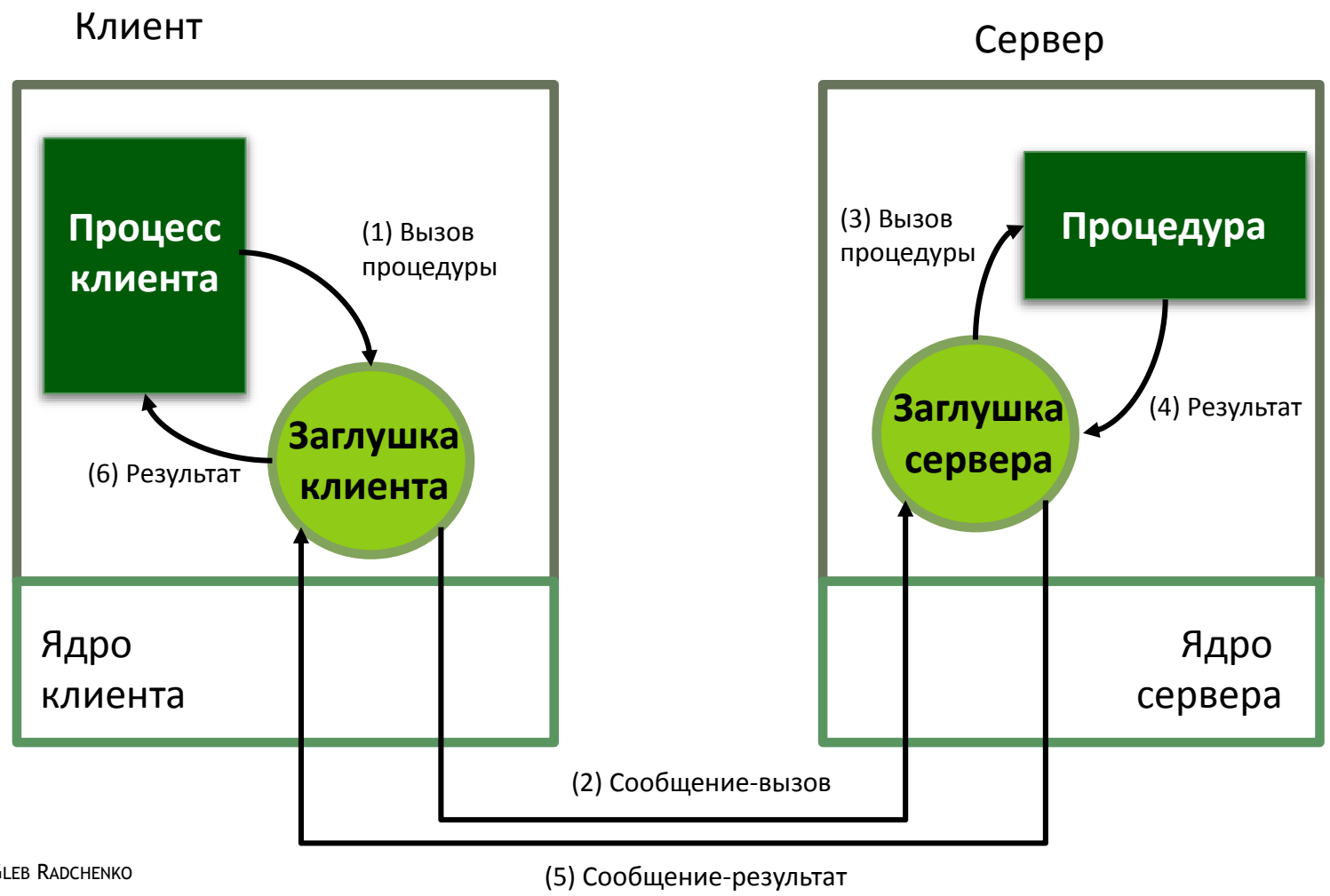
Возвращение

RPC

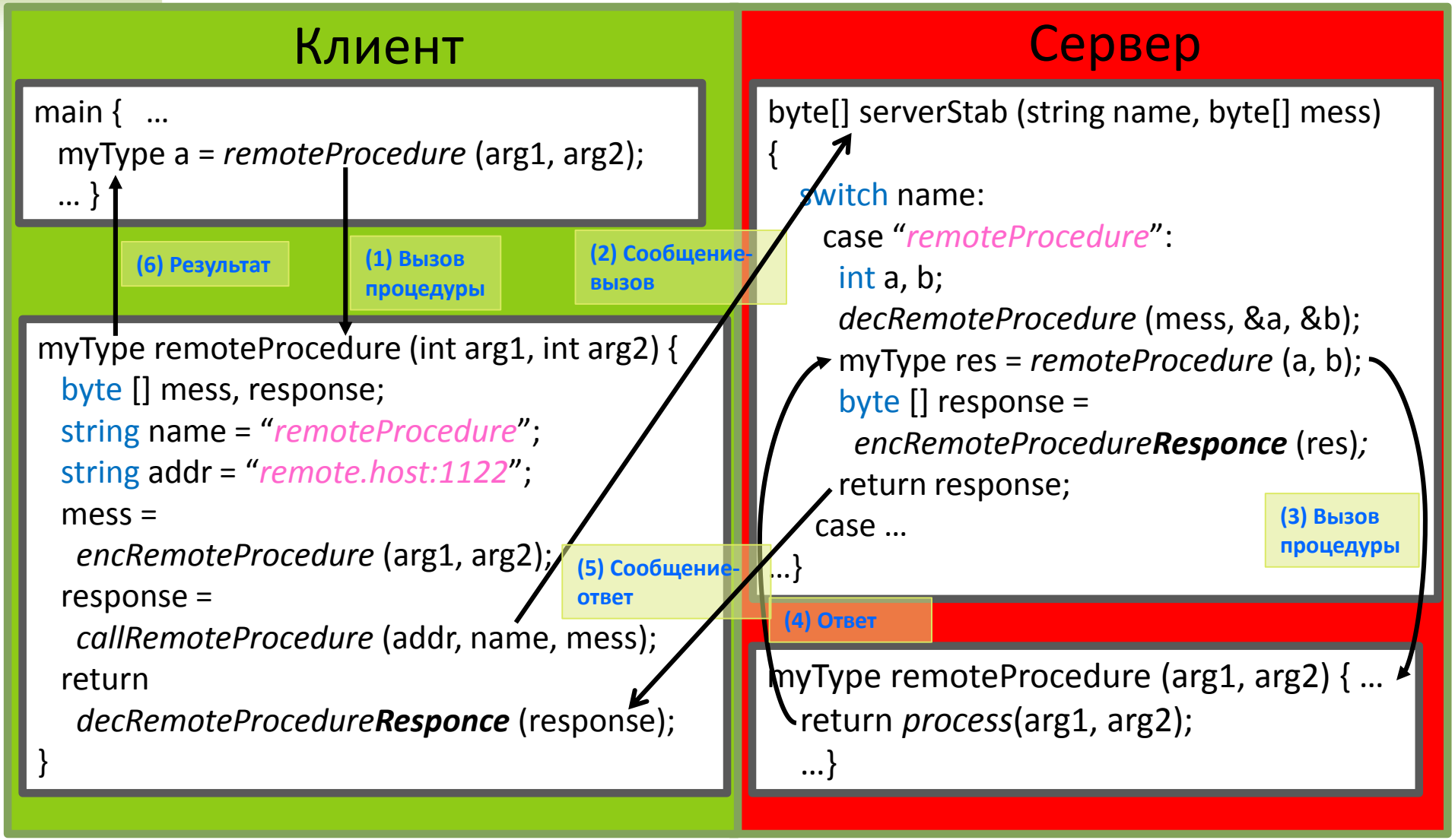
- Разработчики знакомы с концепцией вызова методов
- Хорошо спроектированные процедуры могут выполняться изолированно
- Нет причин, почему бы не выполнять процедуры на удаленной машине



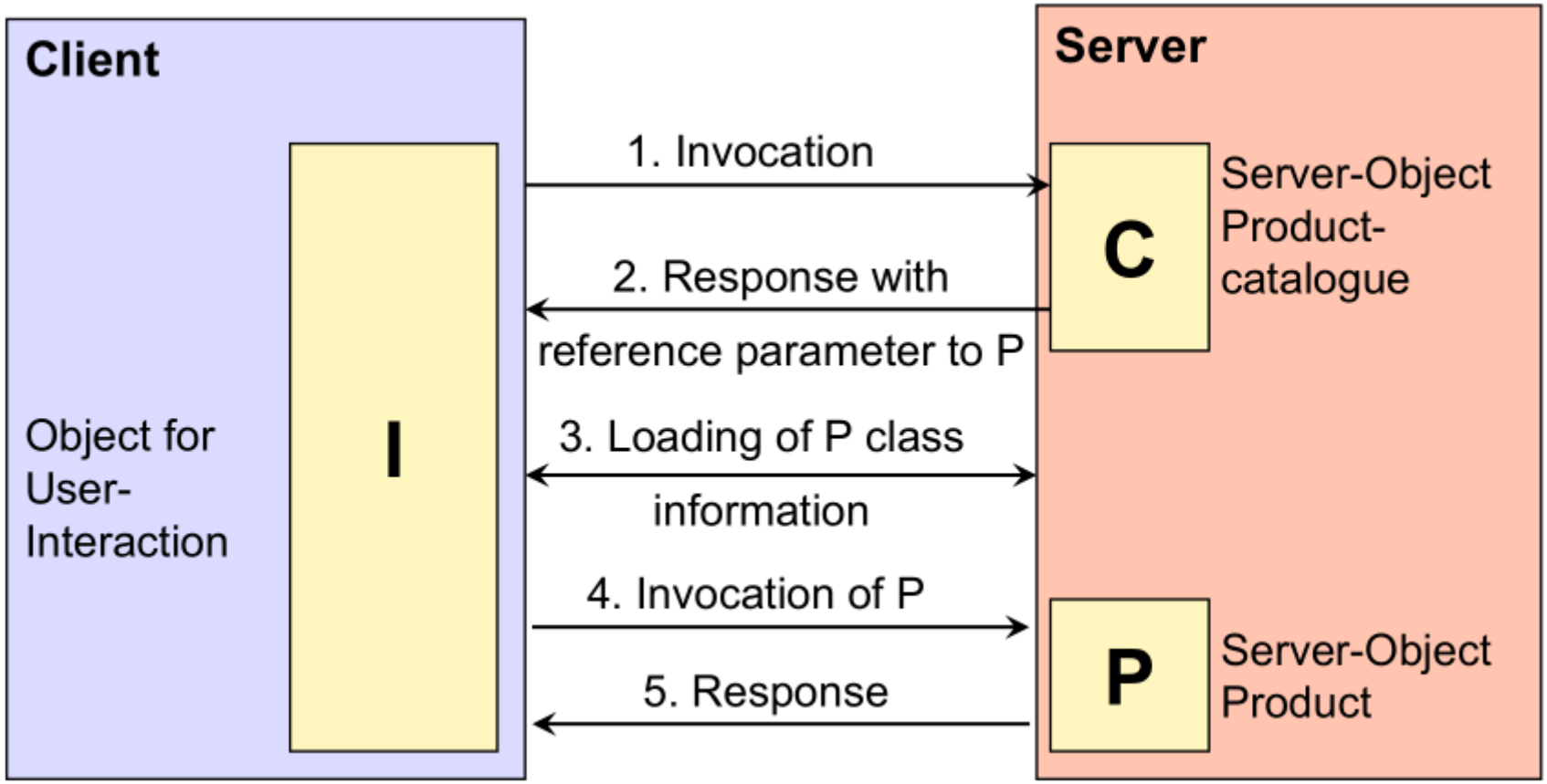
PRC



RPC псевдо-код



Remote Method Invocation RMI

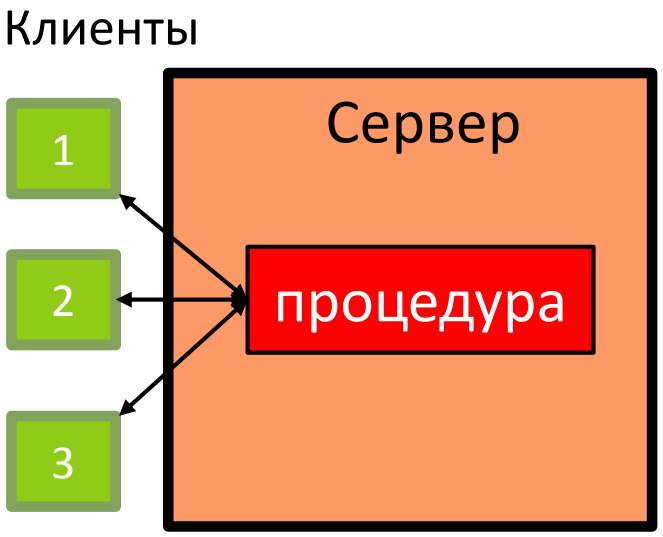


Удаленный объект

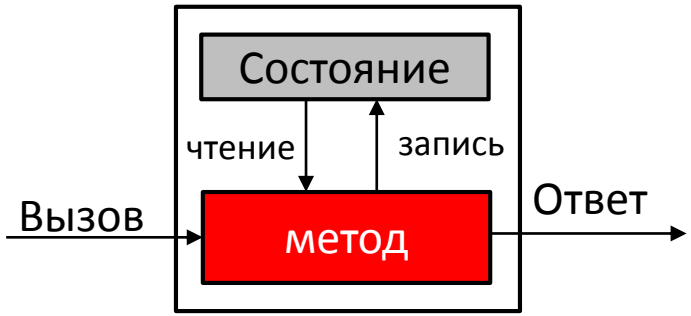
- Удаленный объект – это коллекция данных, определяющих его **состояние.** Это состояние может быть изменено посредством вызова его методов.
- Методы и поля объекта, которые могут быть использованы посредством удаленных вызовов, доступны через **внешний интерфейс** объекта.

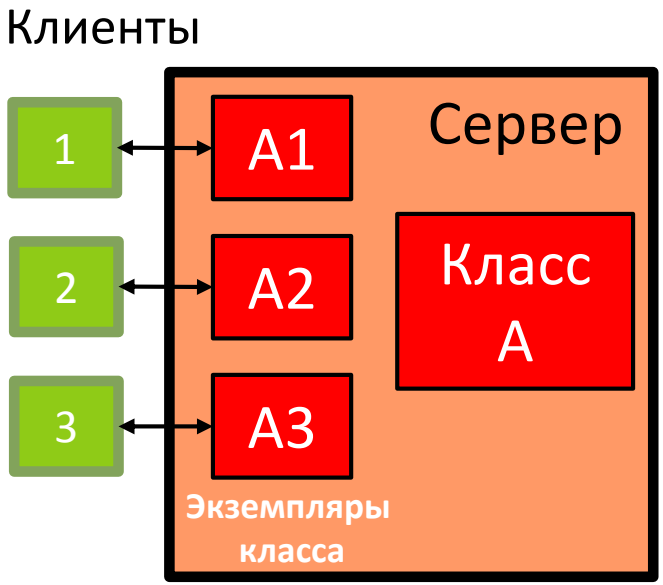
RPC VS RMI

Удаленная процедура



Удаленный объект

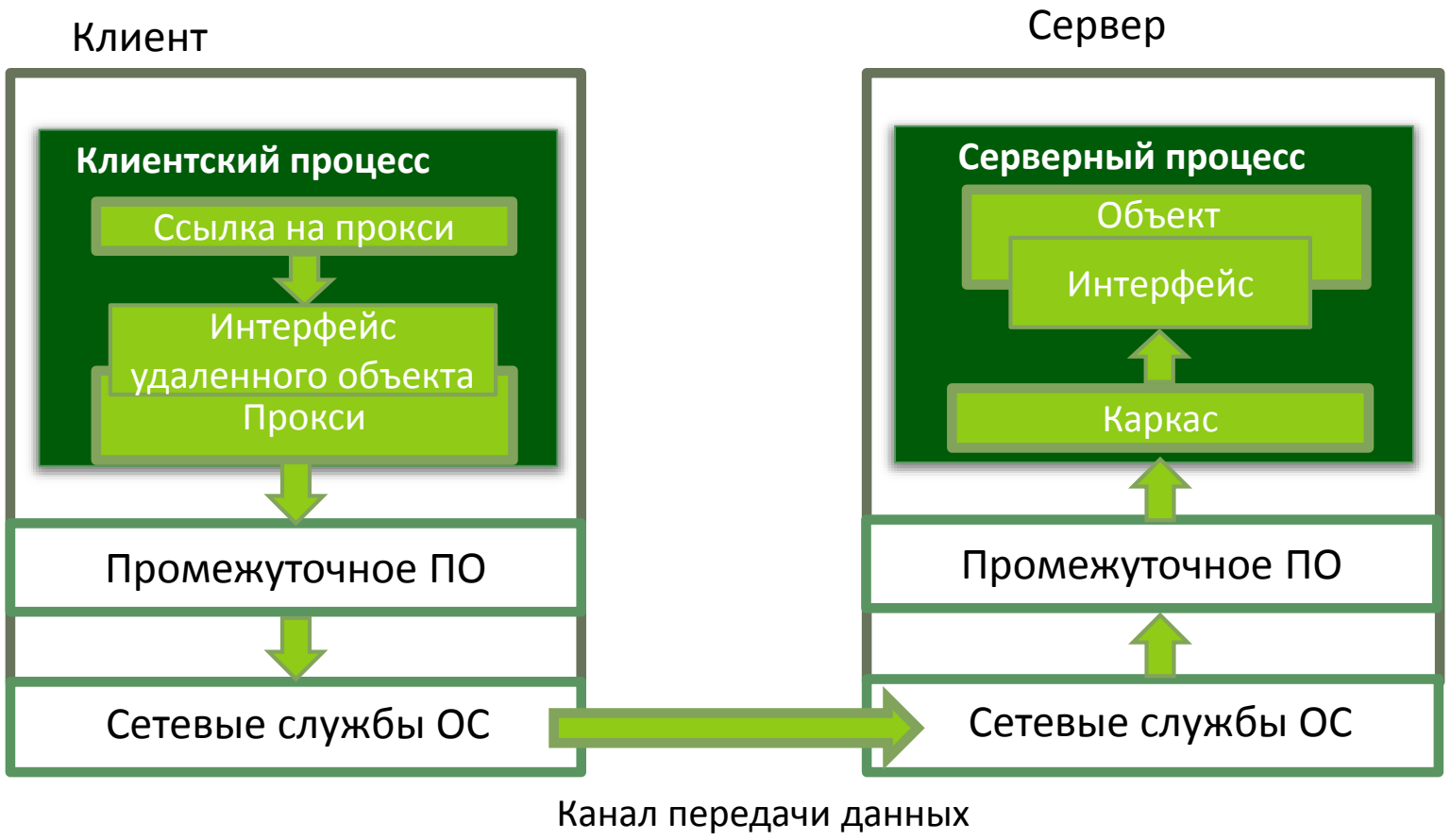




Proxy (заместитель) и Sceleton (каркас)

- Клиентская заглушка, что вызывает удаленный объект, называется прокси (или заместитель).
- **Прокси** реализует тот же интерфейс, что и удаленный объект.
- Серверная заглушка называется каркас (Sceleton в Java RMI)
- Остов связывается с конкретным экземпляром удаленного объекта и вызывает метод с требуемыми параметрами.

Использование удаленного объекта



Remote Method Invocation RMI

Интерфейс:

```
public interface ProductCatalogue extends java.rmi.Remote
{
   ProductDescription[] searchProduct(String productType) throws java.rmi.RemoteException;
   Product provideProduct(ProductDescription d) throws java.rmi.RemoteException;
   int deleteProduct(ProductDescription d) throws java.rmi.RemoteException;
   int updateProduct(Product p) throws java.rmi.RemoteException;
   ...
}
```

Сервер – реализация интерфейса:

```
public class ProductCatalogueImpl extends java.rmi.server.UnicastRemoteObject
implements ProductCatalogue
{
   public ProductCatalogueImpl() throws java.rmi.RemoteException
   {   super();   }

   public ProductDescription[] searchProduct(String productType)
   throws java.rmi.RemoteException
   {
      ProductDescription[] desc = ProductCatalogue.getDescriptionByType(productType);
      return desc;
   }
    ...
}
```

Remote Method Invocation RMI

Реализация сервера:

```
public class ProductCatalogueServer {
   public ProductCatalogueServer() {
      try {
         ProductCatalogue c = new ProductCatalogueImpl();
         Naming.rebind("rmi://localhost:1099/ProductCatalogueService", c);
      }
      catch (Exception e) {…}
   }
   public static void main(String args[]) {
      new ProductCatalogueServer();
   }
}
```

Реализация клиента:

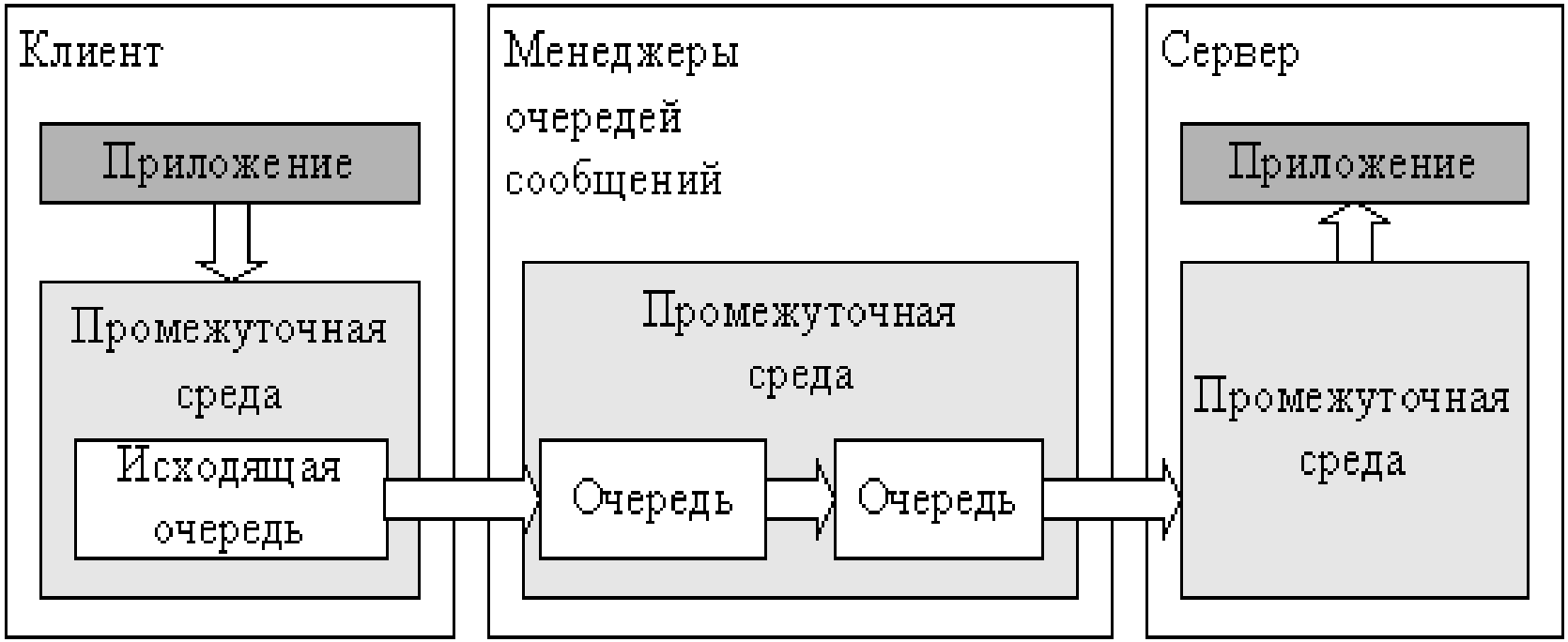
```
public class ProductCatalogueClient {
   public static void main(String[] args)
   {
      try {
         ProductCatalogue c= (ProductCatalogue)Naming.lookup(
            "rmi://hostname/ProductCatalogueService");
         System.out.println( c.searchProduct("book");
      }
catch (Exception e) {…}
   }
}
```

Взаимодействие посредством менеджера сообщений

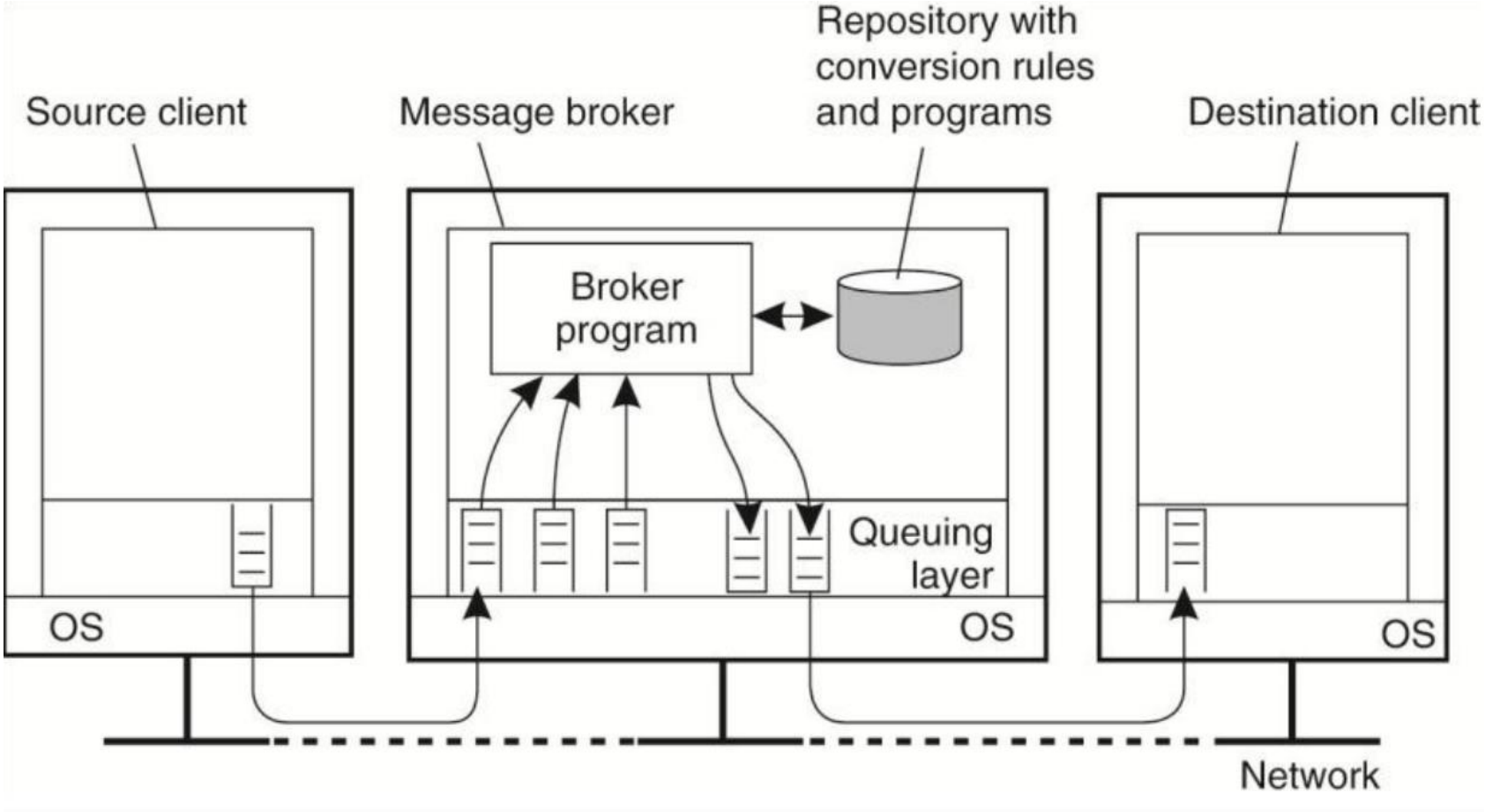
Менеджеры (очереди) сообщений

- *Очередь сообщений* – это ПО промежуточного слой, обеспечивающие управление взаимодействием между элементами РВС, посредством сбора, хранения и маршрутизации сообщений между процессами.
- Очередь сообщений позволяет работающим в различное время приложениям взаимодействовать в гетерогенных сетях и системах, которые могут временно отключаться от сети.
- Приложения отправляют, получают и считывают (то есть читают без удаления) сообщения из очередей.

Использование менеджера сообщений

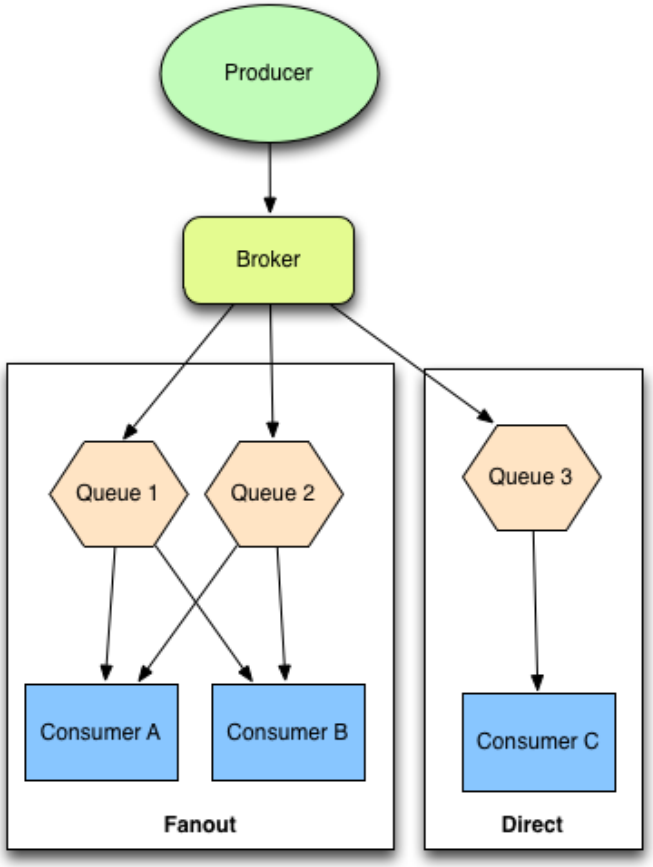


Использование менеджера сообщений



Менеджеры (очереди) сообщений

- Очередь сообщения предоставляет интерфейс для *поставщика (producer)* и *потребителя (consumer)* сообщений.
- Менеджер сообщений может обеспечить распределение сообщений в различные очереди, обеспечивая распределение нагрузки между задачами и возможность горизонтального масштабирования.



Использование менеджера сообщений

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.
- **Высокая прозрачность:** вся коммуникация проходит через менеджер сообщений. Таким образом, можно в любой момент времени оценить, какие сообщения, от кого и кому поступали.

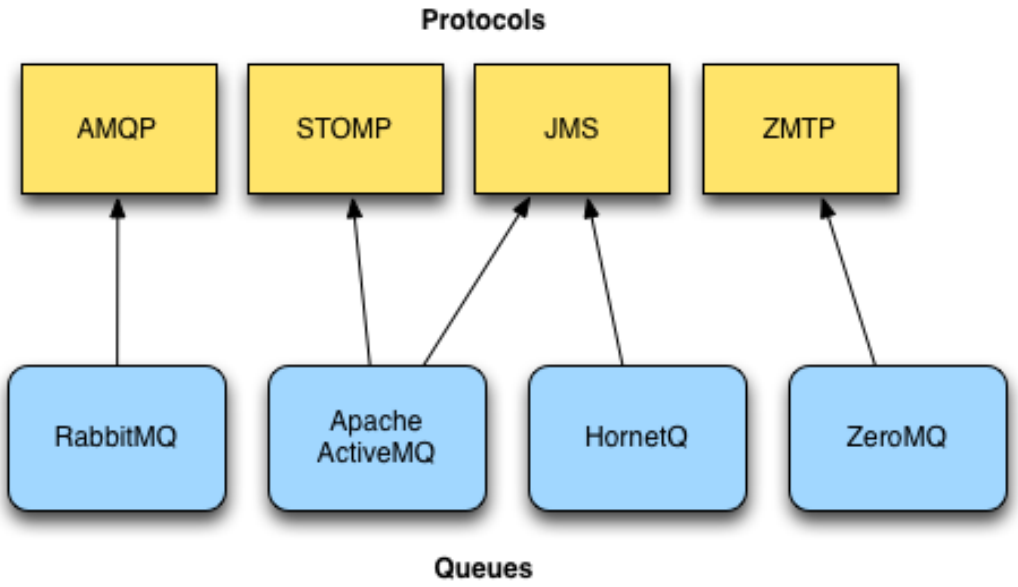
Использование менеджера сообщений

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).
- JMS – Java Message Service
- Microsoft Message Queuing (MSMQ - .NET)



RabbitMQ vs ActiveMQ

- RabbitMQ (http://www.rabbitmq.com/)
  - Создан на основе Erlang
  - Работает на всех основных операционных системах
  - Поддерживает огромное количество платформ для разработчиков (чаще всего – Python, PHP, Ruby)
  - Конфигурация – на основе Erlang
- Apache ActiveMQ (http://activemq.apache.org/)
  - Построен на основе Java Message Service
  - Чаще всего используется совместно с Java-стеком (Java, Scala, Clojure и др).
  - Также поддерживает STOMP (Ruby, PHP, Python).
  - Конфигурация – на основе XML

Пример использования RabbitMQ

- RabbitMQ as a Service: http://www.cloudamqp.com/
- Пример приложения: https://github.com/cloudamqp/java-amqp-example
- Запускаем Sender (OneOffProcess.java), потом Receiver (WorkerProcess.java)

Sender:

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
String message = "Hello CloudAMQP!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

Receiver:

```
while (true) {
      QueueingConsumer.Delivery delivery = consumer.nextDelivery();
      String message = new String(delivery.getBody());
      System.out.println(" [x] Received '" + message + "'");
    }
```

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