Проектирование ПО

Лекция 13: Проектирование распределённых приложений. Часть 1

Тимофей Брыксин
timofey.bryksin@gmail.com

Распределённые системы

- компоненты находятся в компьютерной сети
- взаимодействуют через обмен сообщениями
- основное назначение -- работа с общими ресурсами
- особенности
  - параллельная работа
  - независимые отказы
  - отсутствие единого времени

Fallacies of Distributed Computing

- сеть надёжна
- задержка (latency) равна нулю
- пропускная способность бесконечна
- сеть безопасна
- топология сети неизменна
- администрирование сети централизовано
- передача данных “бесплатна”
- сеть однородна

Архитектура распределённых систем

- Какие сущности взаимодействуют между собой в распределённой системе?
- Как они взаимодействуют?
- Какие (возможно изменяющиеся) роли и ответственности имеют эти сущности в рамках всей архитектуры?
- Как они размещаются на физическую инфраструктуру?

Виды сущностей

- узлы-процессы-потоки
- объекты
- компоненты
- веб-сервисы

Виды взаимодействия

- межпроцессное взаимодействие
- удалённые вызовы
  - протоколы вида “запрос-ответ”
  - удалённые вызовы процедур (remote procedure calls, RPC)
  - Удалённые вызовы методов (remote method invocation, RMI)
- неявное взаимодействие
  - групповое взаимодействие
  - модель “издатель-подписчик”
  - очереди сообщений
  - распределённая общая память

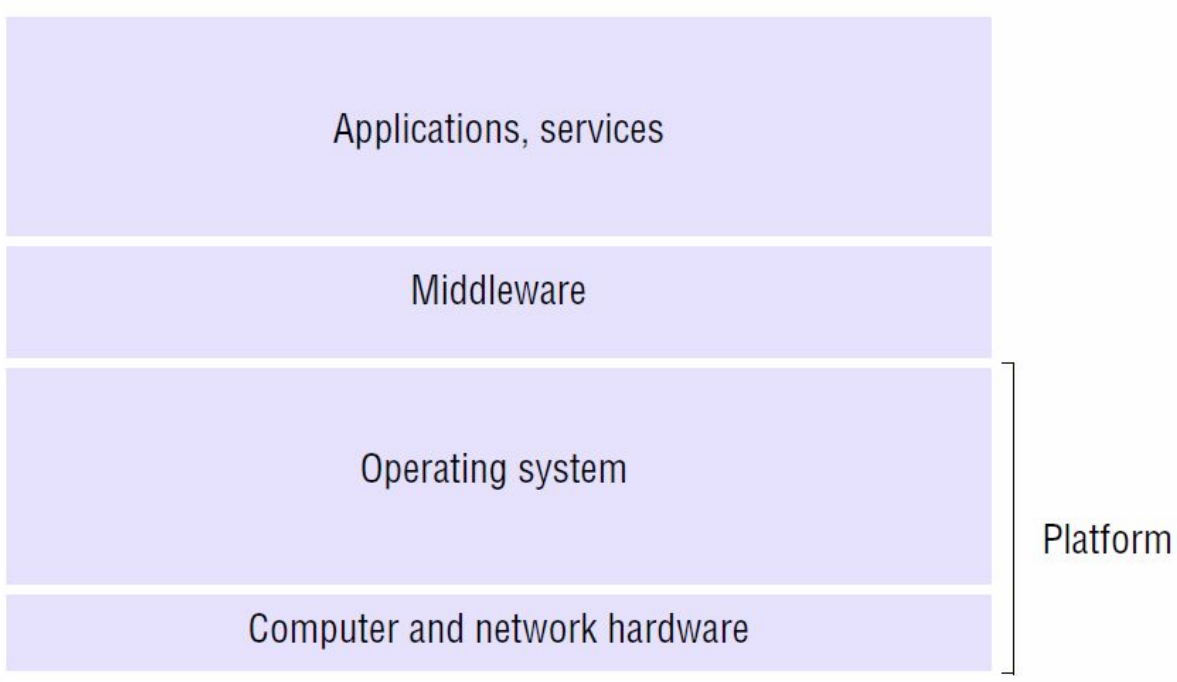
Роли и обязанности

- клиент-сервер
- peer-to-peer

Варианты размещения

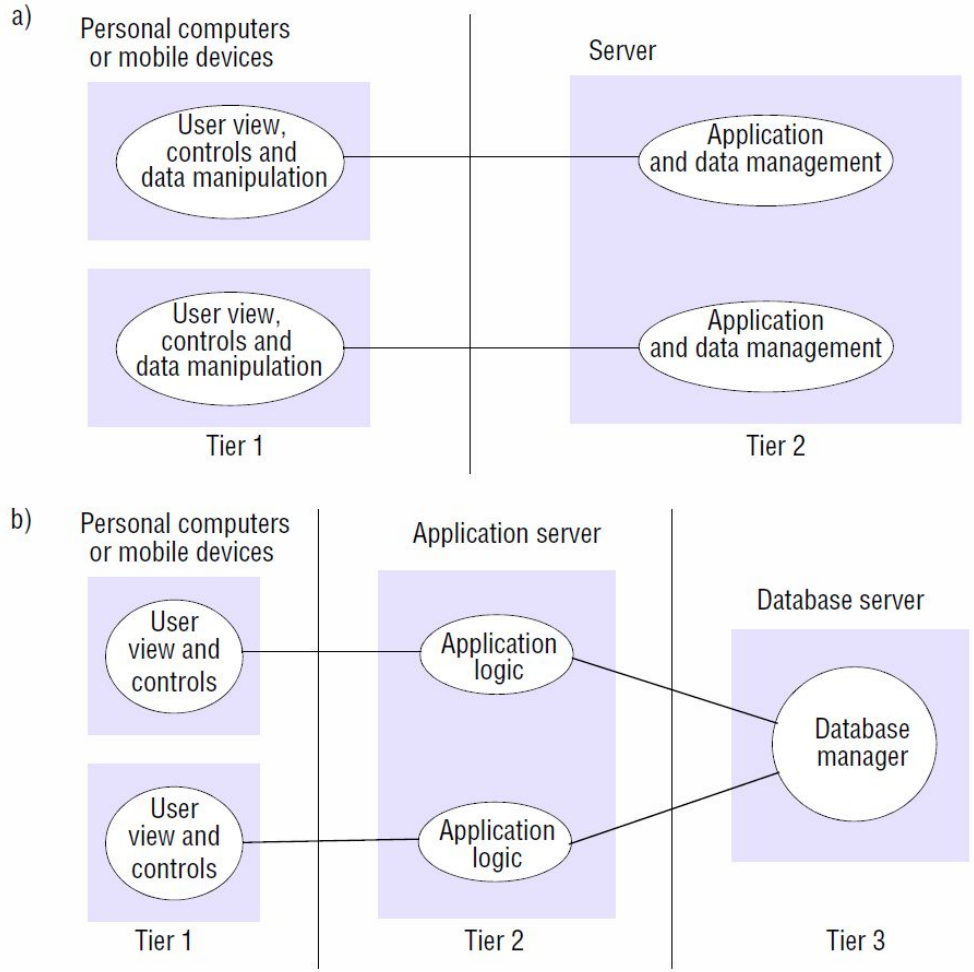
- разбиение сервисов по нескольким серверам
- кэширование
- мобильный код
- мобильный агент

Layered architecture

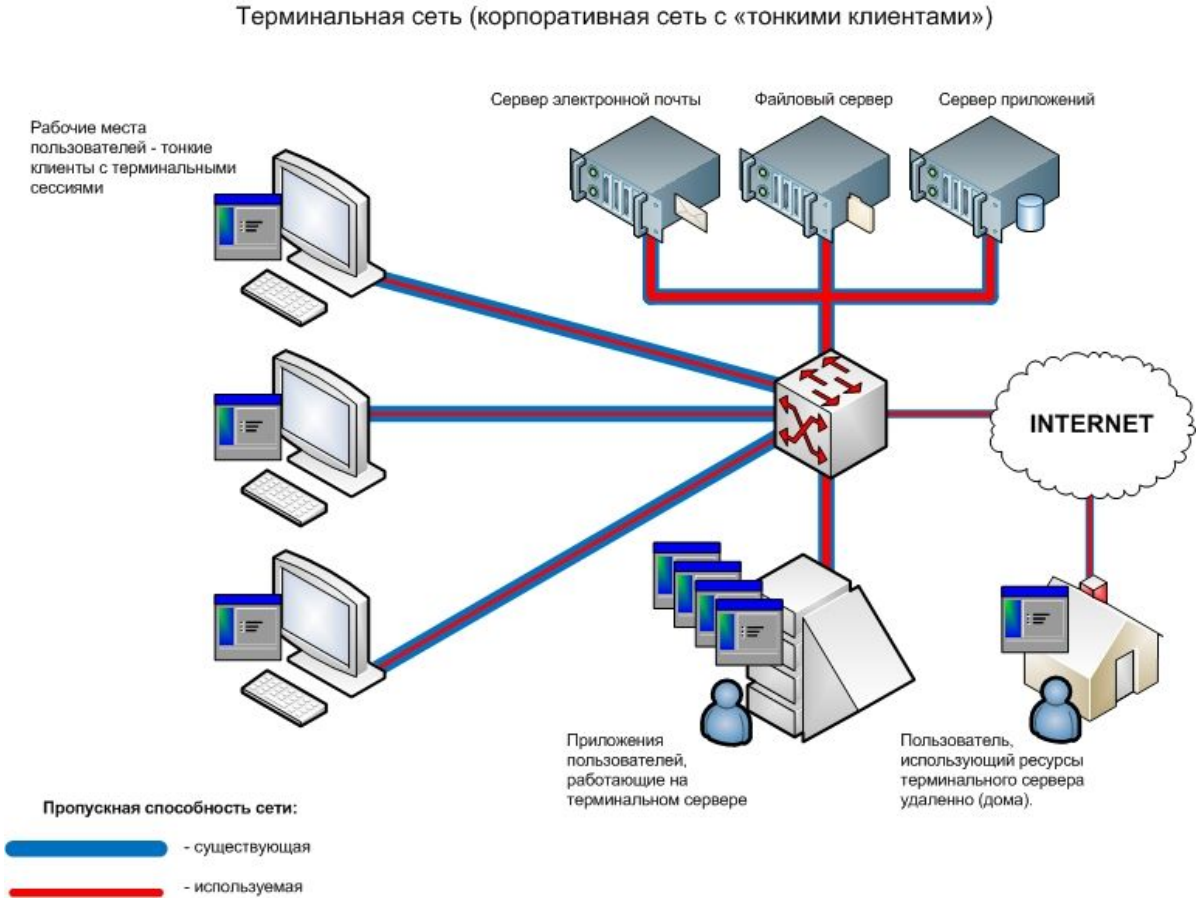


Tiered architecture

- клиент-сервер
- трехзвенная архитектура
- N-звенная архитектура



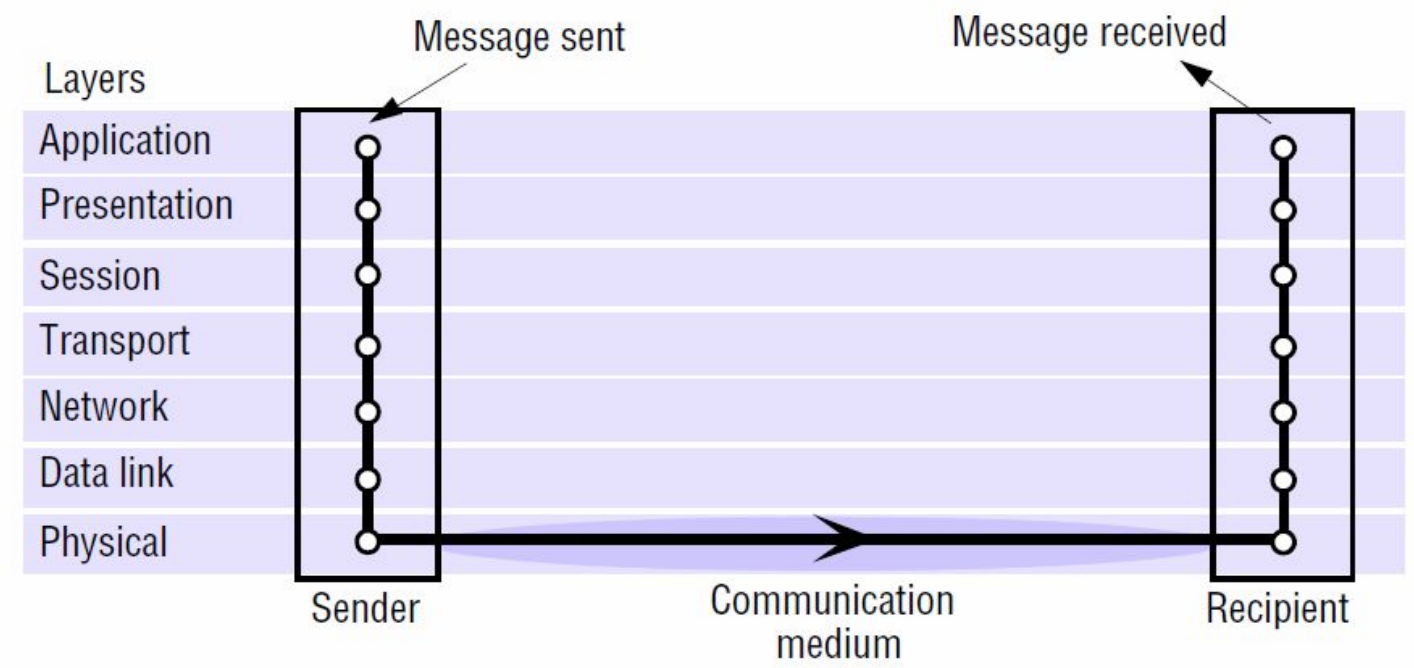
Тонкий клиент



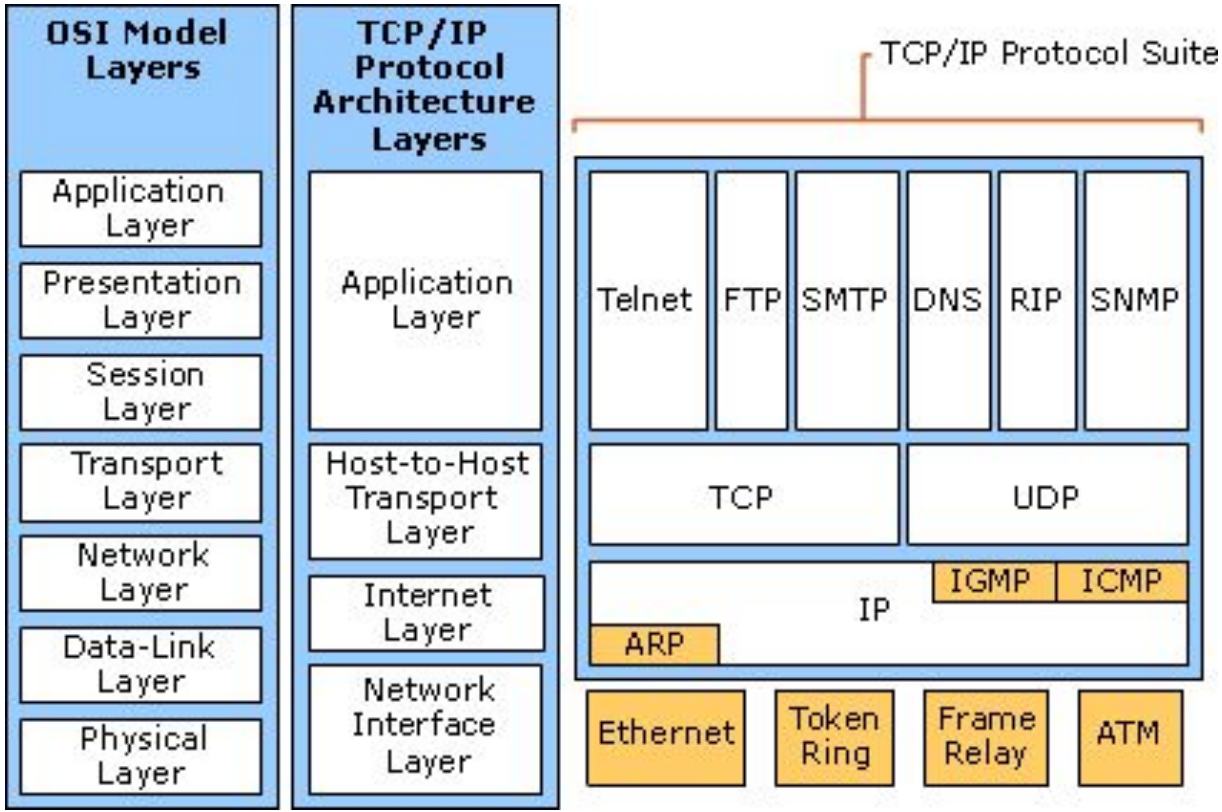
Межпроцессное сетевое взаимодействие

- пакетная передача данных
  - асинхронное взаимодействие
  - полезная информация + заголовок
  - единый канал связи
- протоколы
  - последовательность сообщений
  - формат сообщений

Модель OSI

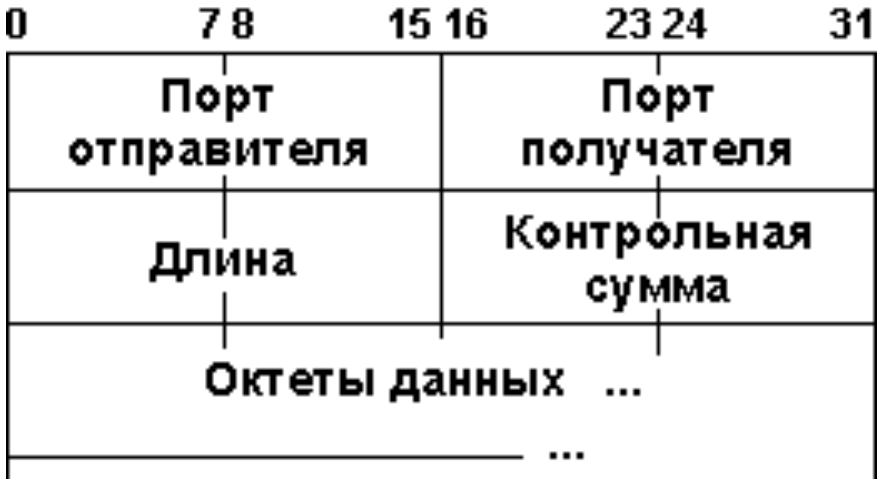

Message sent
Message received
Layers
Application
Presentation
Session
Transport
Network
Data link
Physical
Sender
Communication medium
Recipient

Стек протоколов TCP/IP



Протокол UDP

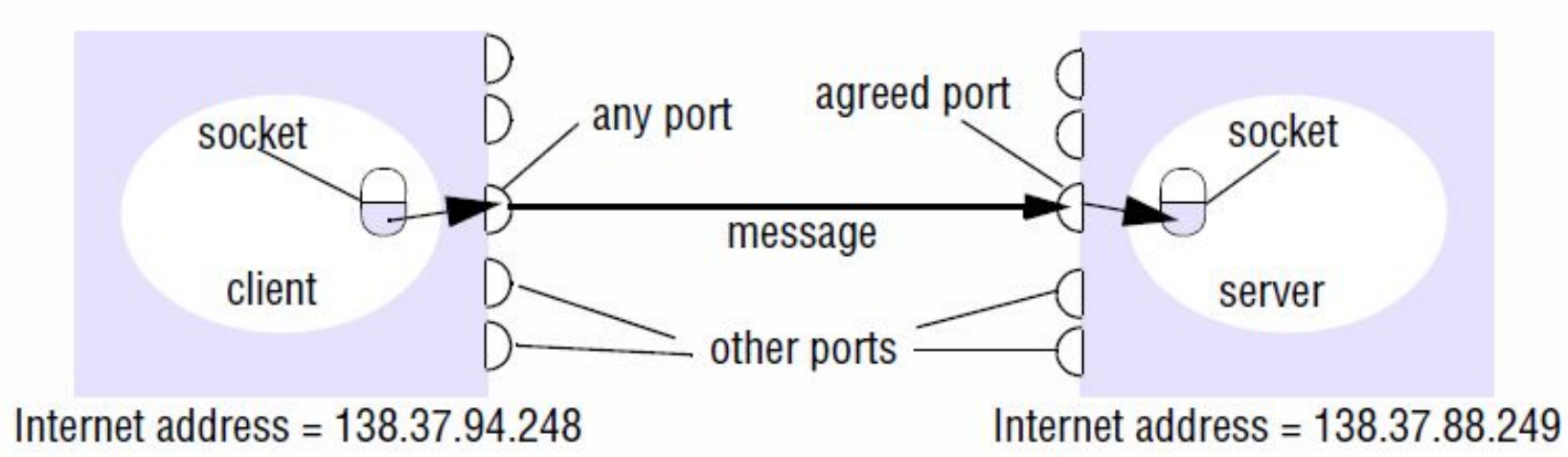
- легковесный
- быстрый
- ненадёжный



Протокол TCP

- абстракция соединения
- абстракция потока
- гарантия корректности
- упорядочение пакетов
- управление потоком данных
- повторная передача данных
- буферизация

Абстракция сокета

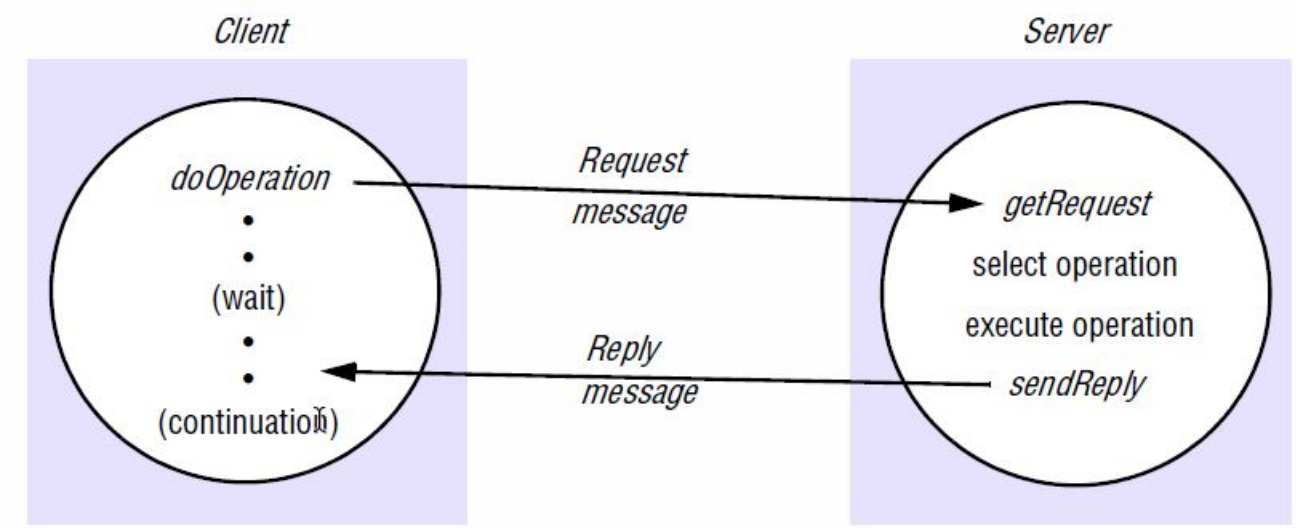


Маршалинг

- преобразование структур данных в массив байтов
- разные форматы данных
  - числа с плавающей запятой
  - little-endian, big-endian
  - кодировки
- данные при передаче не меняются
- симметричная процедура десериализации

Протоколы “запрос-ответ”

- запрос, действие, ответ
- преимущественно синхронные вызовы



“Запрос-ответ” поверх UDP

- + уведомления не нужны
- + установление соединения -- в два раза больше сообщений
- + управление потоком не имеет смысла
- - потери пакетов
  - таймаут + повторный запрос на уровне бизнес-логики
  - защита от повторного выполнения операции (хранение “истории”)
  - новый запрос как подтверждение получения прошлого
- - неопределённый порядок пакетов

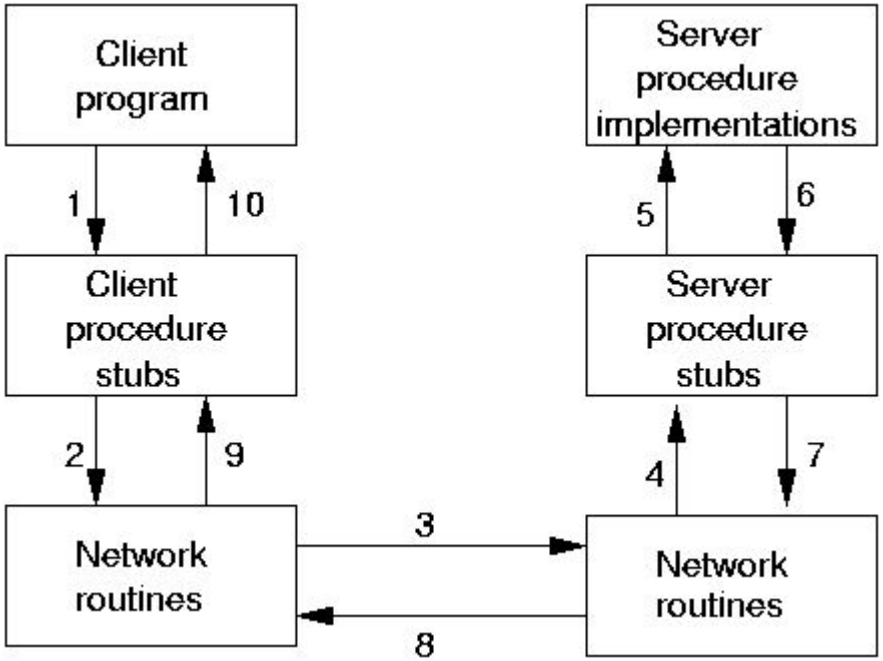
“Запрос-ответ” поверх TCP

- \+ использование потоков вместо набора пакетов
  - удобная отправка больших объёмов данных
  - один поток на всё взаимодействие
- \+ интеграция с потоками ОО-языков
- \+ надёжность доставки
  - отсутствие необходимости проверок на уровне бизнес-логики
  - уведомления в пакетах с ответом
  - упрощение реализации
- \- тяжеловесность коммуникации

HTTP

- пример протокола “запрос-ответ”
- реализован поверх TCP
- соединение на всё время взаимодействия
- маршалинг данных в ASCII
  - MIME
- HTTP 2.0
  - бинарный протокол
  - обязательное шифрование
  - мультиплексирование запросов в одном TCP соединении
  - “предсказывающая посылка данных”

Удалённые вызовы процедур (RPC)



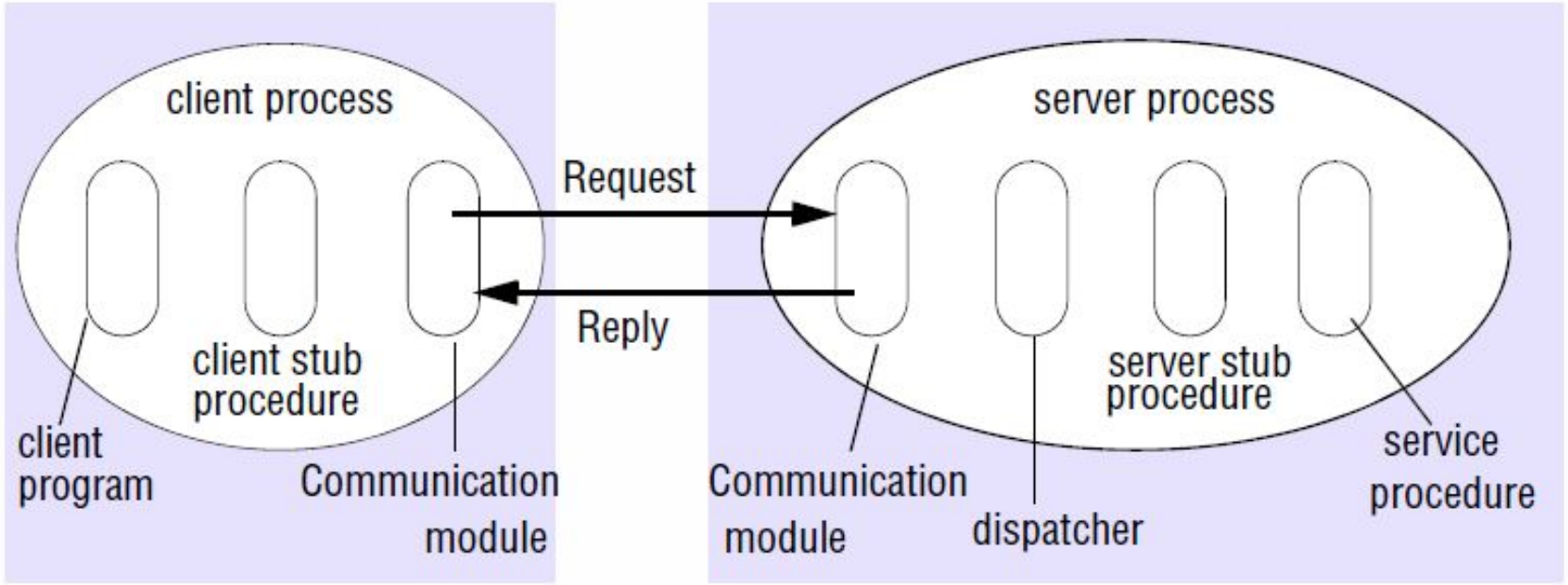
Семантика RPC вызовов

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Прозрачность RPC вызовов

- изначальная цель -- максимальная похожесть на обычные вызовы
  - location and access transparency
- удалённые вызовы более уязвимы отказам
  - нужно понимать разницу между отказом сети и отказом сервиса
  - клиенты должны знать о задержках при передаче данных
    - возможность прервать вызов
- явная маркировка удалённых вызовов?
  - прозрачность синтаксиса
  - явное отличие в интерфейсах
    - указание семантики вызова

Структура RPC middleware

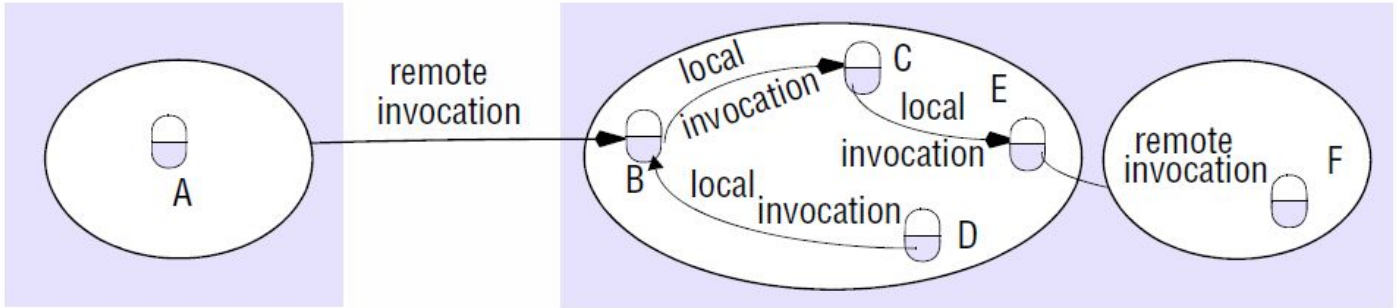


Удалённые вызовы методов (RMI)

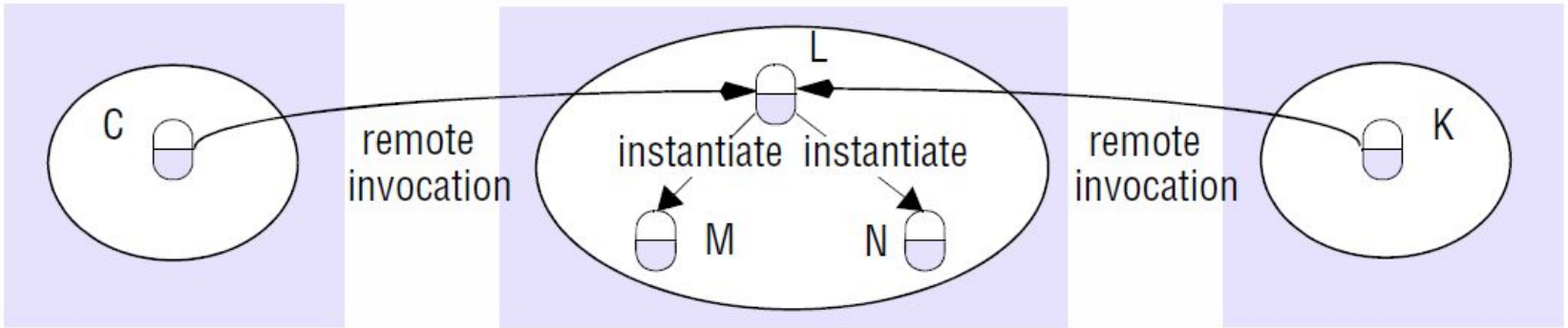
- продолжение идей RPC
  - программирование через интерфейсы
  - работа поверх протоколов “запрос-ответ”
  - at-least-once или at-most-once семантика вызовов
  - прозрачность синтаксиса вызовов
- особенности ОО-программ
  - наследование, полиморфизм
  - передача параметров по ссылкам
  - исключения
  - распределённая сборка мусора

Локальные и удалённые вызовы

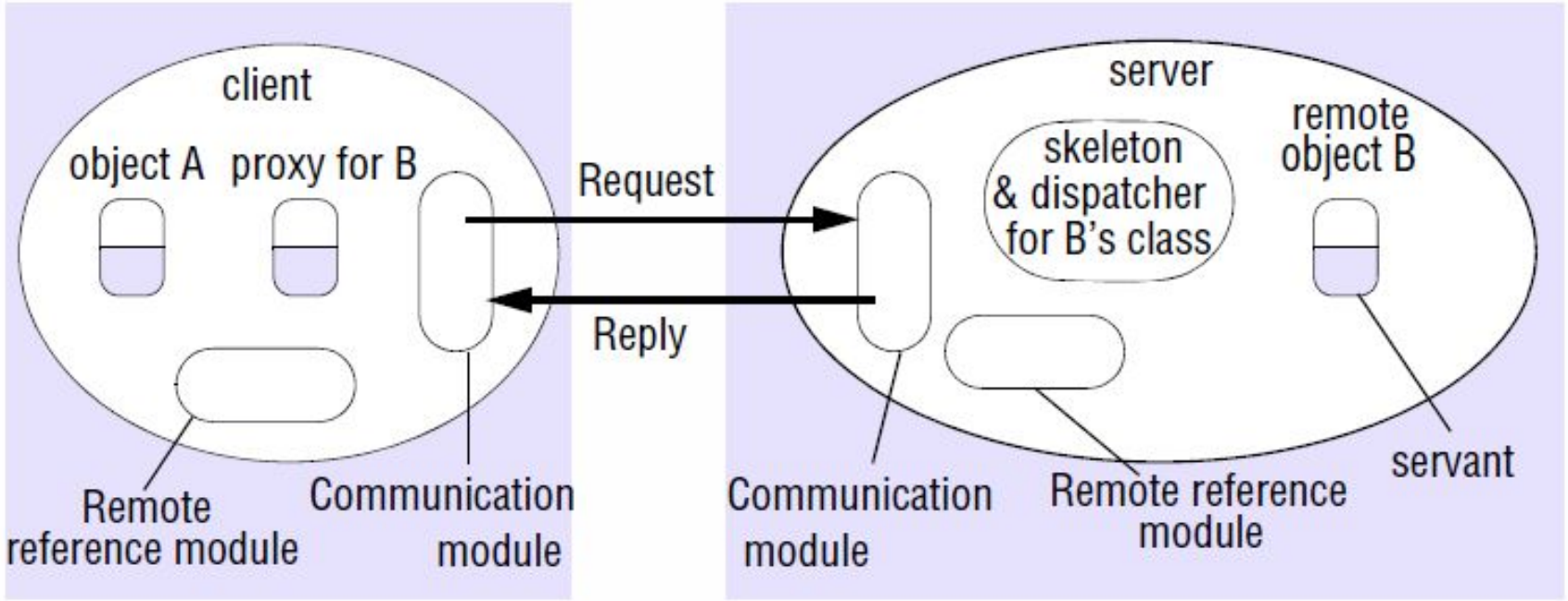
- локальные и удалённые объекты
- интерфейсы удалённых объектов
- ссылки на удалённые объекты
  - как параметры или результаты удалённых вызовов



Создание удалённых объектов



Структура RMI middleware



Web-сервисы

- перенос специализации клиент-сервера в web
- сложные приложения как интеграция веб-сервисов
- HTTP-запрос для выполнения команды
  - асинхронное взаимодействие
  - ответ-запрос
  - событийные схемы
- XML как основной формат сообщений
  - SOAP/WSDL/UDDI
  - XML-RPC
  - REST

SOAP-ориентированные сервисы

- Simple Object Access Protocol
- Web Services Description Language
- Universal Discovery, Description and Integration

SOAP-сообщение

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
            <n:priority>1</n:priority>
            <n:expires>2001-06-22T14:00:00-05:00</n:expires>
        </n:alertcontrol>
    </env:Header>
    <env:Body>
        <m:alert xmlns:m="http://example.org/alert">
            <m:msg>Get up at 6:30 AM</m:msg>
        </m:alert>
    </env:Body>
</env:Envelope>
```

Достоинства SOAP-based сервисов

- автоматический режим описания сервисов
- автоматическая поддержка описаний SOAP-клиентом
- автоматическая валидация сообщений
  - валидность xml
  - проверка по схеме
  - проверка SOAP-сервером
- работа через HTTP
  - хоть через обычный GET

Недостатки SOAP-based сервисов

- ОГРОМНЫЙ размер сообщений
- сложность описаний на клиенте и сервере
- один запрос-один ответ
  - поддержка транзакций на уровне бизнес-логики
- сложности миграции при изменении описания