Распределенные вычислительные системы

Форматы сообщений, клиент-серверная модель взаимодействия

Форматы сообщений

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 сериализируется и его копия передается в другой процес

Форматы сериализации данных

- Текстовые (платформо-независимые):
  - XML
  - JSON
  - YAML
- Бинарные
  - Protocol Buffers (Google)
  - Byte stream (очень неэффективные):
    - java.io.Serializable interface
    - .NET Serializable attribute
  - MessagePack

XML vs JSON

XML

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101,
кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</
phoneNumber>
    <phoneNumber>916 123-4567</
phoneNumber>
  </phoneNumbers>
</person>
```

363 символа

JSON

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address":  {
      "streetAddress": "Московское ш.,
101, кв.101",
      "city": "Ленинград",
      "postalCode": 101101
   },
   "phoneNumbers":  [
      "812 123-1234",
      "916 123-4567"
   ]
}
```

316 символов

Google Protocol Buffers

- Protocol Buffers — язык описания данных, предложенный Google в 2008 году, как альтернатива XML. Предполагается, что Protocol Buffers проще и легче, чем XML.
- Сначала должна быть описана структура данных, которая затем компилируется в классы, представляющие эти структуры. Вместе с классами идет код их сериализации в компактный формат представления.
- Примечательно, что можно добавлять к уже созданной структуре данных новые поля без потери совместимости с предыдущей версией: при анализе старых записей новые поля просто будут игнорироваться.
- PS: Используется в Diablo 3 ;0)

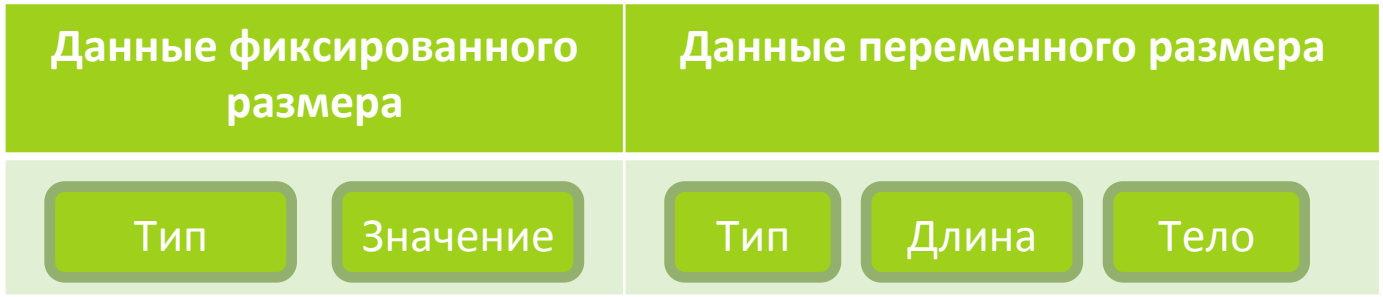
PROTOBUF

```
message Car {
    required string model = 1;

    enum BodyType {
        sedan = 0;
        hatchback = 1;
        SUV = 2;
    }

    required BodyType type = 2 [default = sedan];
    optional string color = 3;
    required int32 year = 4;

    message Owner {
        required string name = 1;
        required string lastName = 2;
        required int64 driverLicense = 3;
    }

    repeated Owner previousOwner = 5;
}
```

.proto - файл

MessagePack

- MessagePack – это бинарный формат предоставления данных, структура которого напоминает JSON (только более быстрый и более компактный).
- Был спроектирован, чтобы обеспечивать прозрачную конвертацию в JSON



JSON vs MessagePack

	JSON		MessagePack	
null	null	4 bytes	c0	1 byte
Integer	10	2 bytes	0a	1 byte
Array	[20]	4 bytes	91 14	2 bytes
String	”30”	4 bytes	a2 ‘3’ ‘0’	3 bytes
Map	{“40”:n	11 bytes	81 a1 ‘	5 bytes

Architecture of MessagePack by Sadayuki Furuhashi http://www.slideshare.net/frsyuki/architecture-of-messagepack

MessagePack

JSON

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address":  {
      "streetAddress": "Московское ш., 101, кв.101",
      "city": "Ленинград",
      "postalCode": 101101
   },
   "phoneNumbers":  [
      "812 123-1234",
      "916 123-4567"
   ]
}
```

338 bytes

MessagePack (hex)

84 a9 66 69 72 73 74 4e 61 6d 65 a8 d0 98 d0
b2 d0 b0 d0 bd a8 6c 61 73 74 4e 61 6d 65 ac
d0 98 d0 b2 d0 b0 d0 bd d0 be d0 b2 a7 61 64
64 72 65 73 73 83 ad 73 74 72 65 65 74 41 64
64 72 65 73 73 da 00 27 d0 9c d0 be d1 81 d0
ba d0 be d0 b2 d1 81 d0 ba d0 be d0 b5 20 d1
88 2e 2c 20 31 30 31 2c 20 d0 ba d0 b2 2e 31
30 31 a4 63 69 74 79 b2 d0 9b d0 b5 d0 bd d0
b8 d0 bd d0 b3 d1 80 d0 b0 d0 b4 aa 70 6f 73
74 61 6c 43 6f 64 65 ce 00 01 8a ed ac 70 68 6f
6e 65 4e 75 6d 62 65 72 73 92 ac 38 31 32 20
31 32 33 2d 31 32 33 34 ac 39 31 36 20 31 32
33 2d 34 35 36 37

187 bytes 55 %

http://msgpack.org/

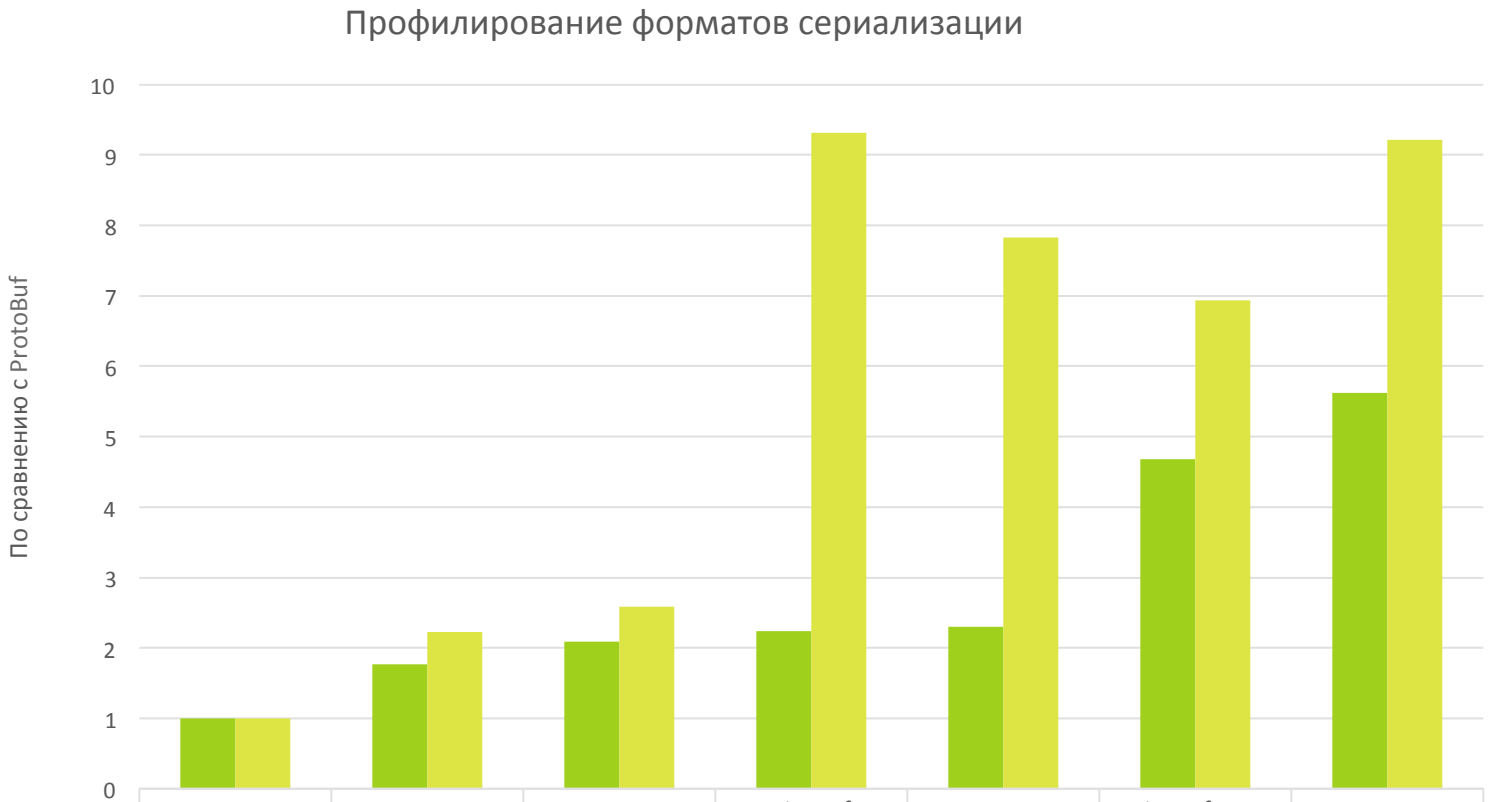
MessagePack

- JSON+ZIP VS MessagePack:
  - На 50% падает производительность при архивировании / разархивировании
  - Невозможно работать с данными в режиме потока (напрямую), необходима обязательная предварительная разархивация

Форматы обмена данными

- JSON
  - человеко-читаемый / редактируемый
  - может быть разобран без предварительного знания схемы
  - отличная поддержка браузеров (JavaScript native class description)
  - компактнее, чем XML
- XML
  - человеко-читаемый / редактируемый
  - может быть разобран без предварительного знания схемы
  - стандарт для многих протоколов (SOAP и т.п.)
  - хорошая инструментальная поддержка (XSD, XSLT, SAX, DOM и т.д.)
  - не компактный, большой объем «лишних» описаний
- Protobuf (Google), MessagePack
  - очень компактный (плотная упаковка данных)
  - очень быстрая обработка
  - не предназначен для чтения/редактирования человеком
- Protobuf (Google):
  - встроенная поддержка версий протокола (при изменении протокола, клиенты могут работать со старой версией, пока не обновятся)
  - без знания схемы очень тяжело разобрать (бинарный формат данных, внутренне не однозначен, требуется схема для разбора)

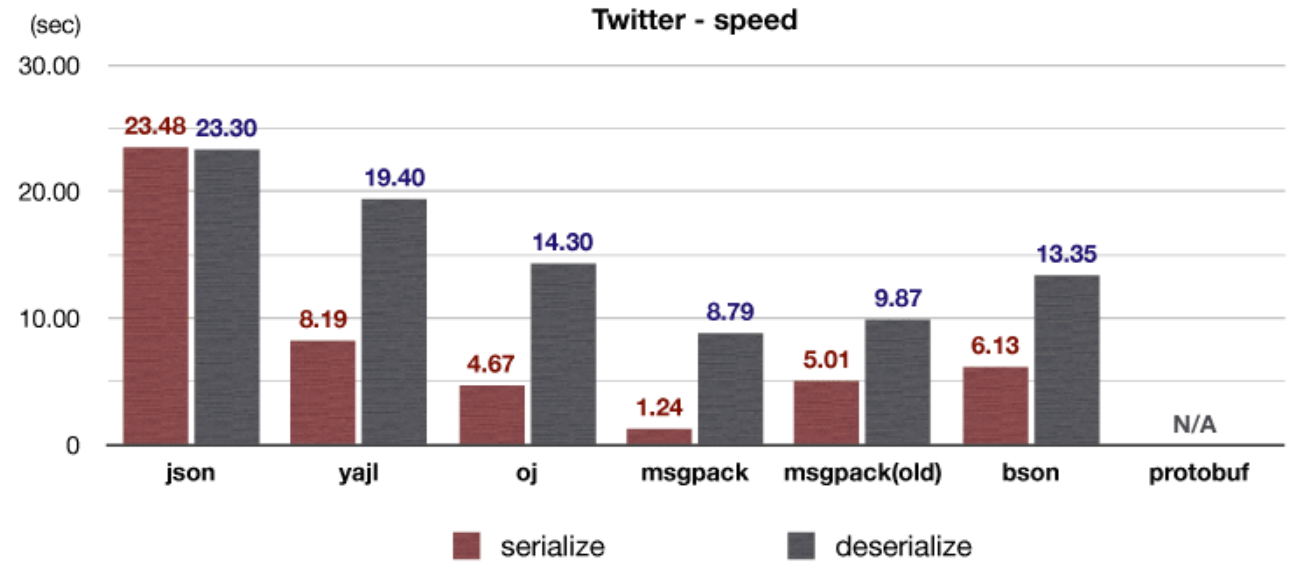
Форматы сериализации

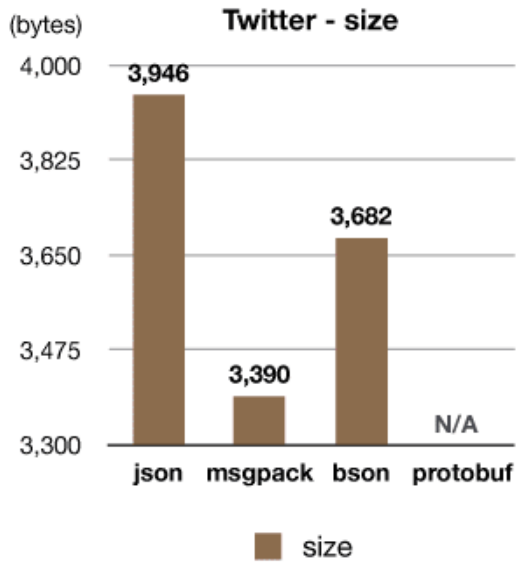


	ProtoBuf.net	ServiceStack TypeSerializer	ServiceStack JsonSerializer	Microsoft JsonDataContract Serializer	NewtonSoft.Json	Microsoft XML DataContractSerializer	Microsoft BinaryFormatter
Больше (раз)	1	1,77	2,09	2,24	2,3	4,68	5,62
Медленнее (раз)	1	2,23	2,58	9,31	7,83	6,93	9,21

Basically protocol buffers (protobuf-net) is around **7x** quicker than the fastest Base class library Serializer in .NET (XML DataContractSerializer). Its also smaller than the competition as it is also **2.2x** smaller than Microsoft's most compact serialization format (JsonDataContractSerializer).

http://www.servicestack.net/benchmarks/NorthwindDatabaseRowsSerialization.100000-times.2010-08-17.html





Adam Leonard. MessagePack for Ruby version 5
https://gist.github.com/adamjleonard/5274733

Когда использовать какие форматы?

- **XML**
  - Если система предоставляет публичный API в виде XML Веб-сервиса (изучим их позже)
  - Работаете с «классической» системой, в которой уже используется XML в качестве стандарта обмена данными
  - Требуются стандартные инструменты верификации и трансформации (например, в HTML) (XSD, XSLT, SAX, DOM и т.д.)
- **JSON**
  - Если система предоставляет публичный API в виде REST-сервиса (изучим их позже)
  - Если клиенты, в большинстве своем, реализуются на JavaScript
  - Более компактный чем XML (в 2-2.5 раза) и самый простой для чтения/редактирования – соответственно, везде, где вы хотели бы использовать XML, подумайте, может быть стоит использовать JSON.
- **MessagePack** – если требуется высокая скорость и компактность, совместимость с JSON, но не требуется редактирование/чтение человеком
- **Protobuf**
  - Если для вас прежде всего важен объем и скорость обработки данных
  - Если важна возможность простого обновления протокола
  - Если реализуется внутренний (не публичный) протокол обмена

Модель взаимодействия клиент-сервер

Проблема: централизация VS децентрализация

- Это одна из старейших проблем в IT
- Пример:
  - *King J.L. Centralized versus decentralized computing: organizational considerations and management options // ACM Computing Surveys. Vol. 15, Issue 4. 1983. P. 319-349.*

До 70-х годов: централизованная модель

- До середины 70-х годов прошлого века доминировала централизованная модель:
  - Высокая стоимость телекоммуникационного оборудования
  - Слабая мощность вычислительных систем

80-е – 90-е: мейнфреймы

- Появление систем разделения времени и удаленных терминалов - предпосылка возникновения клиент-серверной архитектуры.
- Ресурсы мейнфреймов предоставлялись конечным пользователям посредством удаленного соединения.
- Дальнейшее развитие телекоммуникационных систем и появление персональных компьютеров дало толчок развитию клиент-серверной парадигме обработки данных

Клиент-серверная архитектура

- Согласно парадигме клиент-серверной архитектуры:
  - один или несколько клиентов и один или несколько серверов
  - совместно с базовой операционной системой
  - и средой взаимодействия
  - образуют единую систему, обеспечивающую распределенные вычисления, анализ и представление данных

Применение клент-серверной модели

- Процесс разработки распределенных приложений достаточно сложен и одной из наиболее важных задач является решение того, как
  - *функциональность приложения должна быть распределена между клиентской и серверной частью.*

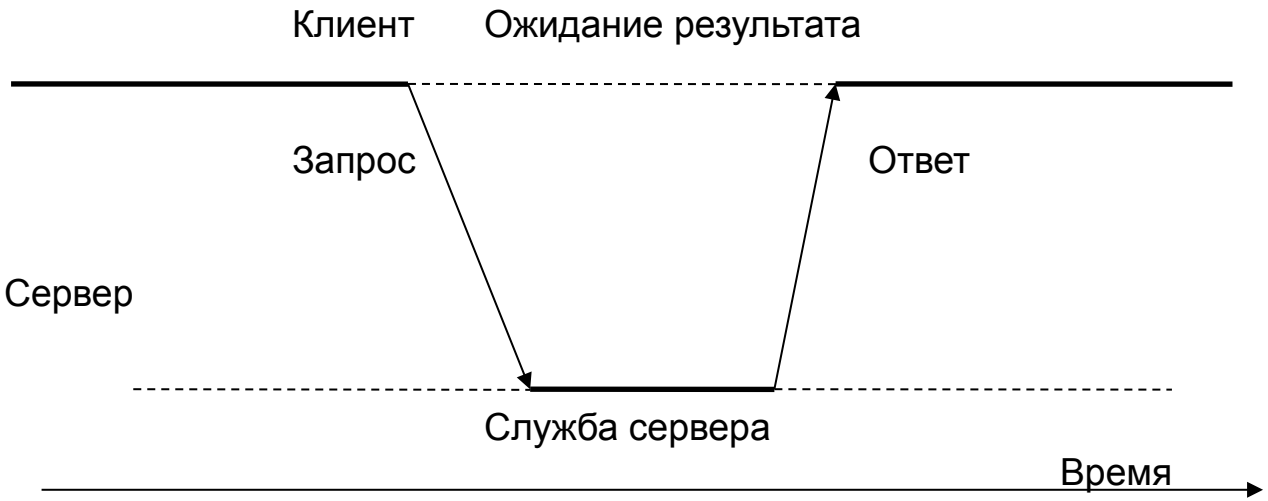
Алгоритм работы клиент-серверной архитектуры

В классическом случае данная схема функционирует следующим образом:

- клиент формирует и посылает запрос на сервер;
- сервер производит необходимые манипуляции с данными, формирует результат и передаёт его клиенту;
- клиент получает результат, отображает его на устройстве вывода и ждет дальнейших действий пользователя.

Цикл повторяется, пока пользователь не закончит работу с сервером.

Обобщение взаимодействия «Клиент-Сервер»



Надежность сети

- Если базовая сеть надежна как локальная сеть, взаимодействие может быть реализовано простым протоколом, без установления соединения (выигрыш эффективности)
- Что, если сообщения пропадают? Если теряется ответ?
  - Отправлять повторно?
  - Надеяться на лучшее?
- А если это операция перевода 10 000$ с одного счета на другой?

Надежные протоколы соединения

- В прикладных протоколах используется TCP/IP:
  - До посылки запроса клиент должен установить соединение
  - Сервер использует то же соединение для посылки ответа

Уровни клиент-серверной архитектуры

Уровни клиент-серверной архитектуры

- Сегодня выделяют 3 основных уровня клиент-серверной архитектуры:
  - Уровень представления (пользовательского интерфейса)
  - Уровень бизнес-логики (обработки)
  - Уровень данных

Уровень представления

- Обычно реализуется на клиентах
- Организует методы взаимодействия с приложением
- Простейший вариант:
  - символьный дисплей (терминал) к мейнфрейму

Уровень бизнес-логики

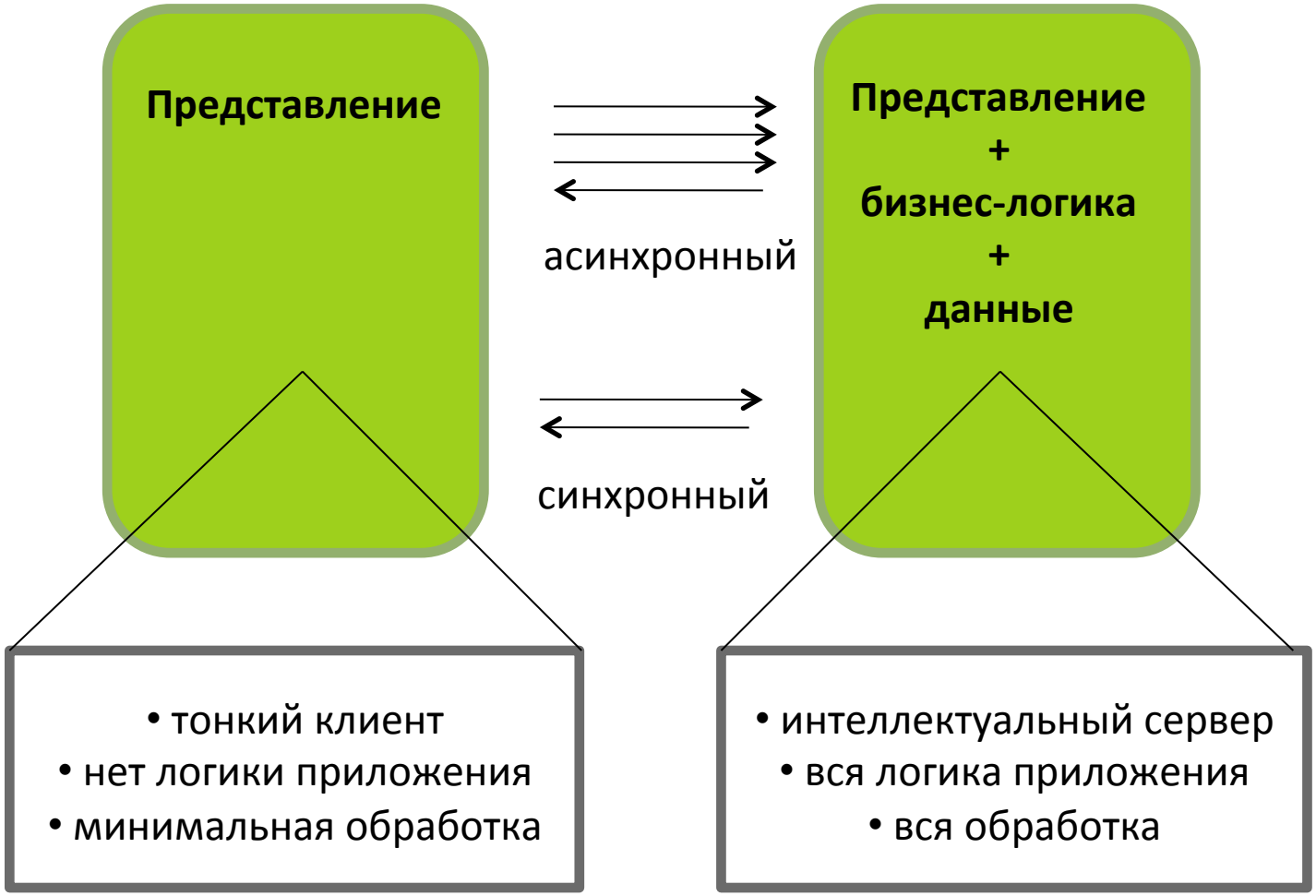
- Бизнес-логика – это совокупность правил, принципов и зависимостей поведения объектов предметной области системы.
- Синоним: логика предметной области (Domain Logic).
- Пример:
  - формула расчета зарплаты + налоги;
  - оценка качества обучения на основе оценок студента;
  - отказ от отеля при отмене рейса авиакомпанией.

Уровень данных

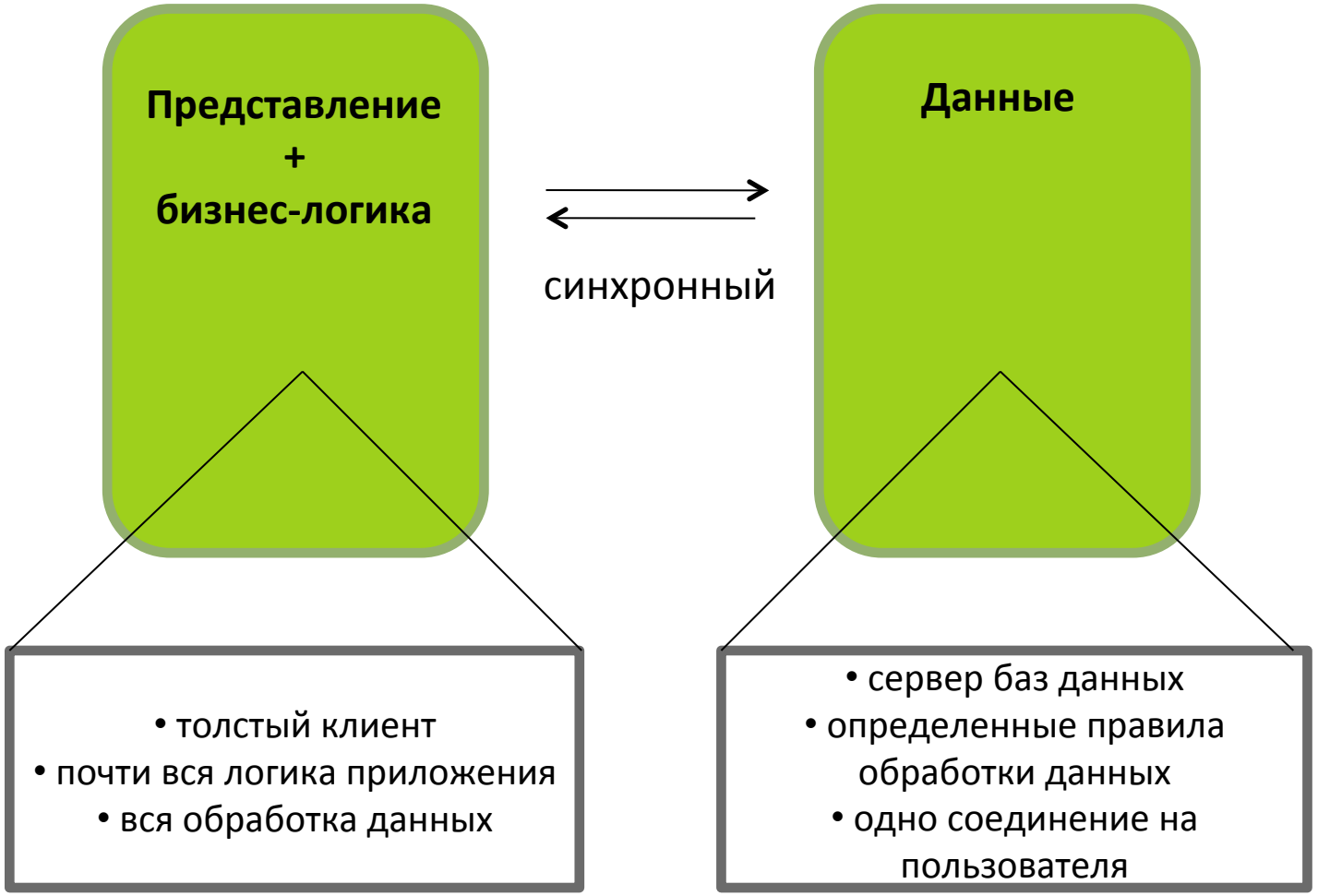
- Программы, которые предоставляют данные обрабатывающим приложениям
- требование СОХРАННОСТИ: когда приложение не работает, данные должны сохранятся в определенном месте;
- требование ЦЕЛОСТНОСТИ: метаданные (описание таблиц, ограничения и т.п.) должны исполнятся и проверяться на этом уровне
- Обычно реализуется реляционной БД

История и типы клиент-серверной архитектуры

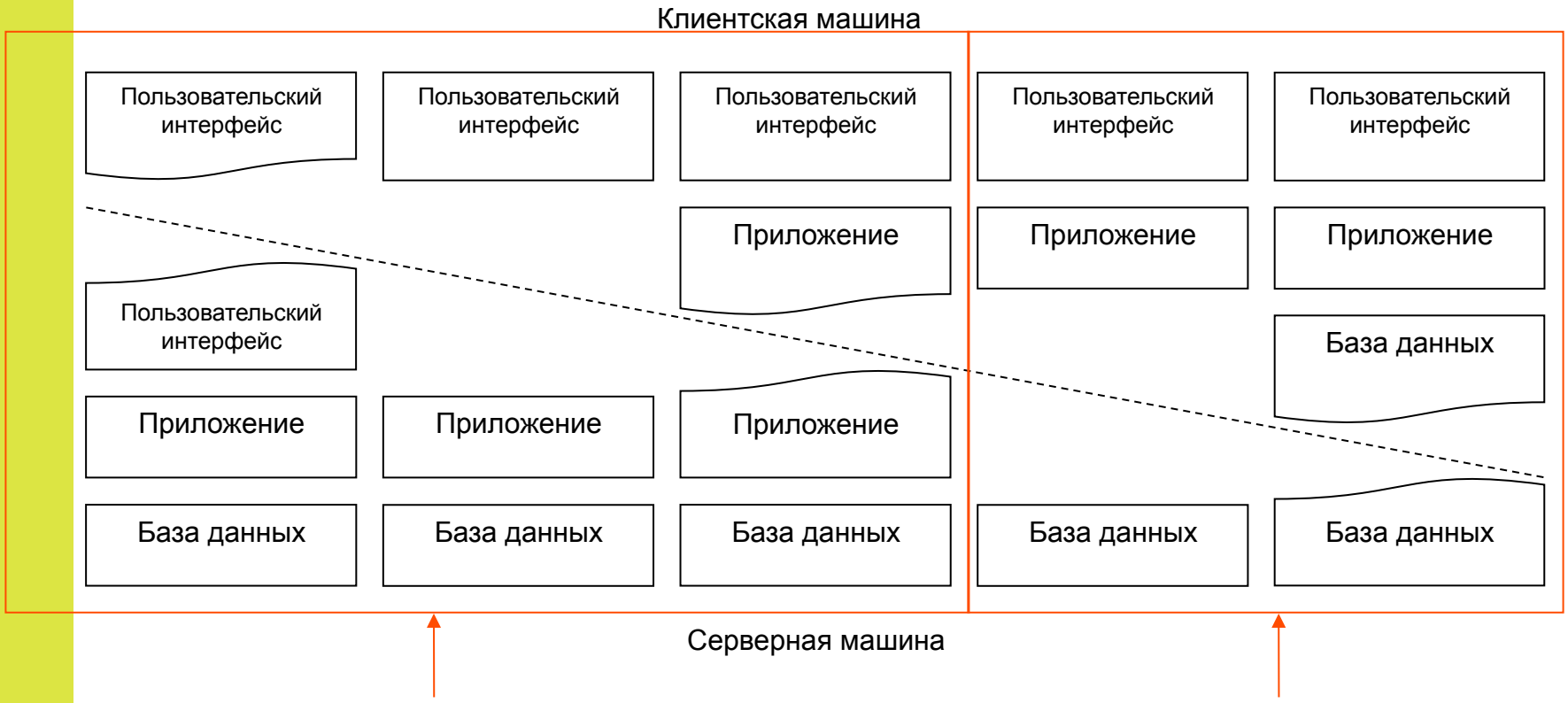
Однозвенная клиент-серверная архиетктура



Двузвенная архитектура



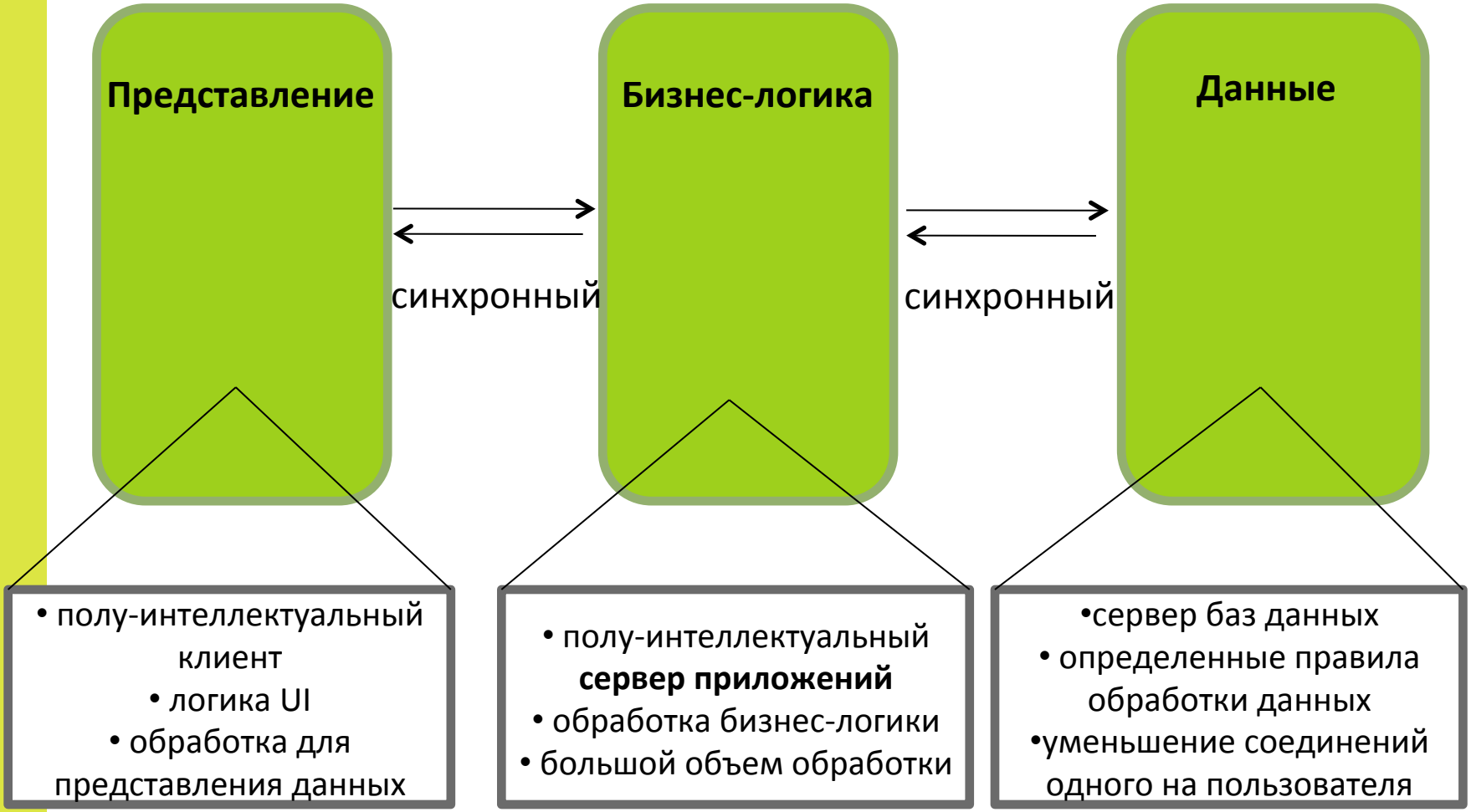
Альтернативные варианты двузвенной архитектуры



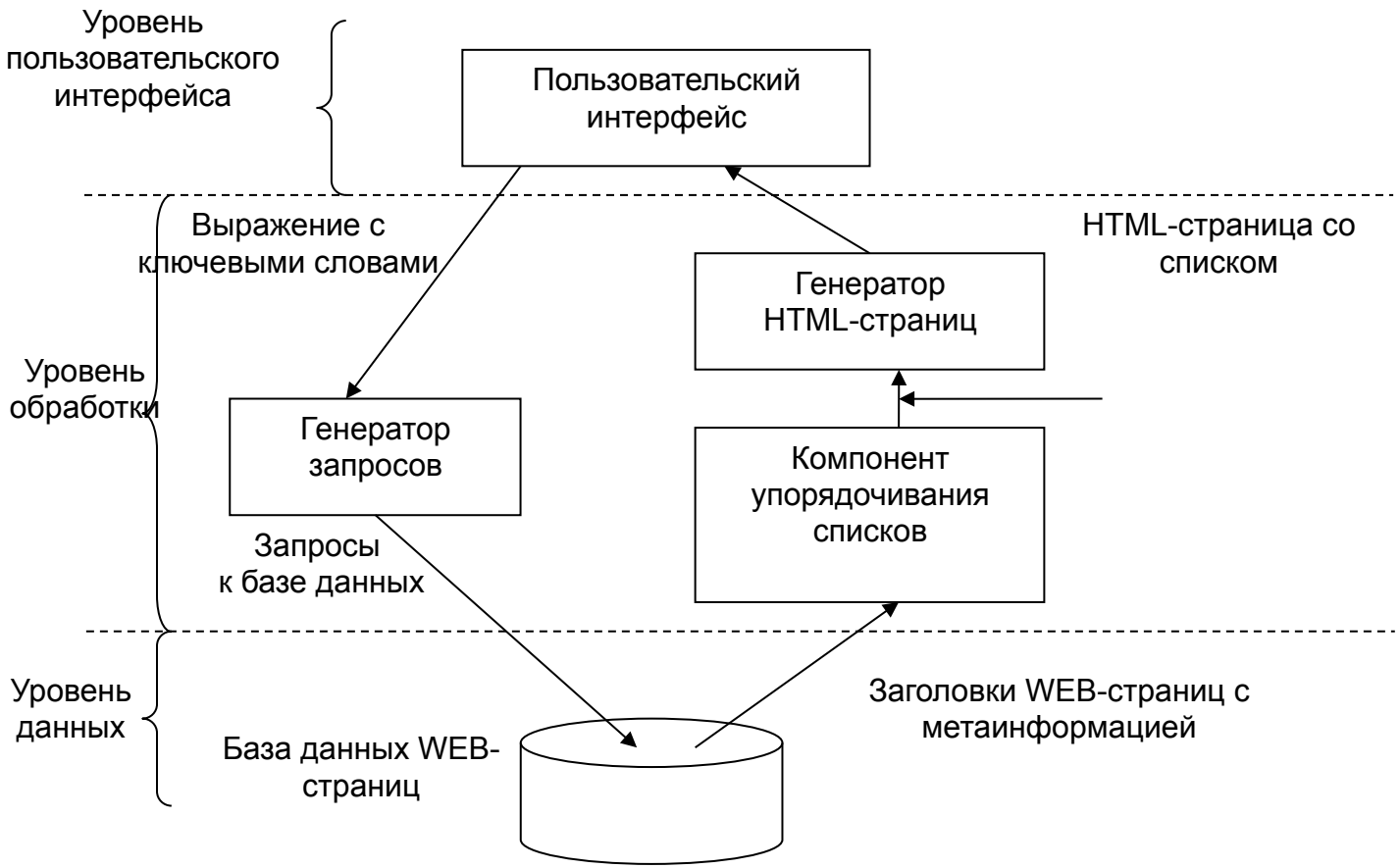
Минусы двузвенной архитектуры

- Чрезвычайные затраты на поддержание рабочих станций, которые должны обрабатывать бизнес-логику
- Чрезвычайная сложность обновления приложения при незначительном изменении бизнес-логики (необходимо переустановить все клиенты)
- Каждая рабочая станция – уникальный набор ПО, который может конфликтовать с клиентом и влиять на его работу

Трехзвенная архитектура



Пример трехзвенной архитектуры – поисковая машина



Современный пример многозвенной архитектуры

- 1. Браузер клиента->
  - 2. Сервер IIS->
    - 3. Исполняющая среда ASP.NET 2.0->
      - 4. Провайдер данных ADO.NET 2.0 ->
        - 5. Сервер MySQL ->
      - 6. Провайдер данных ADO.NET 2.0 ->
    - 7. Исполняющая среда ASP.NET 2.0->
  - 8. Сервер IIS ->
- 9. Браузер клиента