# АРХИТЕКТУРЫ ПО ПРОМЕЖУТОЧНОГО СЛОЯ

# АРХИТЕКТУРЫ ПО ПРОМЕЖУТОЧНОГО СЛОЯ ОСНОВАННЫЕ НА РАСПРЕДЕЛЕННЫХ ОБЪЕКТАХ И КОМПОНЕНТАХ

# ПОНЯТИЕ РАСПРЕДЕЛЕННОГО ОБЪЕКТА

- Ключевой характеристикой распределенных объектов является то, что они позволяют принять **объектно-ориентированную модель программирования** для разработки распределенных систем и, таким образом, скрыть основную сложность программирования распределенных приложений.
- В этом подходе **сущности** РС представлены **объектами**.
- К объектам можно получить доступ через **ссылки на объекты**.
- Объект предоставляет интерфейс, который определяет типы, которые можно использовать для объявления типа переменных или параметры и возвращаемые значения методов если его класс содержит код, который реализует методы этого интерфейса.
- Объекты общаются в основном с помощью метода **вызова удаленного**, но также, возможно, с помощью альтернативная парадигма коммуникации (например, распределенные события). Это относительно простой подход имеет **ряд важных преимуществ**, в том числе следующие:

  • ***Инкапсуляция***, присущая объектно-ориентированным решениям, хорошо подходит для программирования распределенных приложений.

  • Связанное свойство ***абстракции данных*** обеспечивает четкое разделение между спецификация объекта и его реализация, позволяющая программистам работать исключительно с точки зрения ***интерфейсов*** и не беспокоиться о деталях реализации например, используемый язык программирования и операционная система.

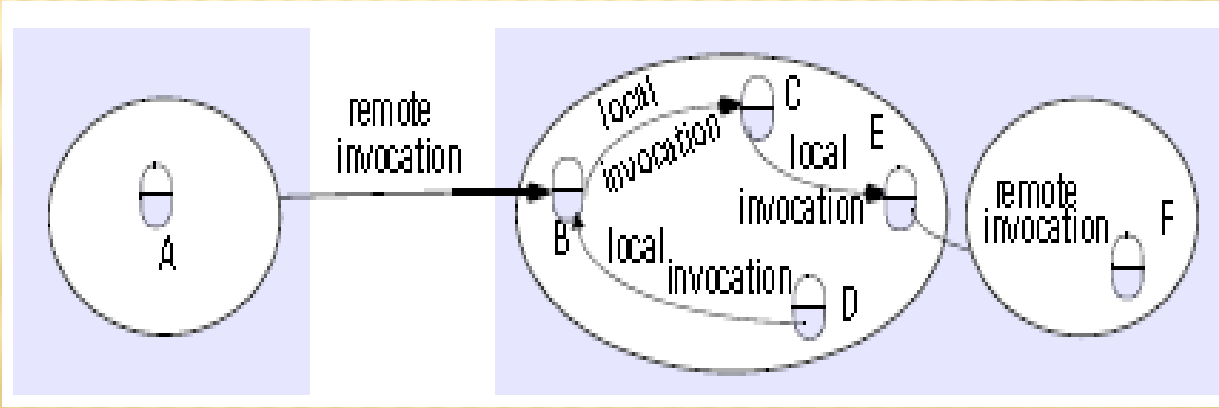
  • Этот подход также позволяет создавать более динамичные и ***расширяемые*** решения для например, позволяя вводить новые объекты или заменять один объект с другим (совместимым) объектом.
- Имеется ряд промежуточных реализаций: Java RMI и CORBA.

# РАСПРЕДЕЛЕННЫЕ ОБЪЕКТЫ

| Объекты | Распределенные объекты | Описание распределенного объекта |
|---|---|---|
| Ссылки на объекты | Ссылки на удаленные объекты | Глобально уникальные ссылки для распределенный объект; может быть передано как параметр. |
| Интерфейсы | Удаленные интерфейсы | Предоставляет абстрактную спецификацию методов, которые могут быть вызваны для удаленного объекта; указывается с использованием языка определения интерфейса (IDL). |
| Действия | Распределенные действия | Инициируются вызовом метода, потенциально может привести к вызову цепи; удаленные вызовы используют RMI в качестве метода коммуникаций. |
| Исключения | Распределенные исключения | Дополнительные исключения, следующие из распределенного характера системы, включая потерю сообщения или отказ процесса. |
| Сборка мусора | Распределенная сборка мусора | Расширенная схема, гарантирующая, что объект будет продолжать существовать, если хотя бы одна ссылка на объект или удаленный объект ссылка существует для этого объекта, в противном случае его следует удалить. Требуется распределенный алгоритм сбора мусора. |

# Ссылка на объект и его интерфейс

- Следующие две фундаментальные концепции лежат в основе распределенная объектная модель:
  - **Ссылки на удаленные объекты:** другие объекты могут вызывать методы удаленного объекта. если у них есть доступ к ссылке на его удаленный объект.
  - **Удаленные интерфейсы**: каждый удаленный объект имеет удаленный интерфейс, который указывает, какой из его методов можно вызывать удаленно. Например, объекты B и F на рисунке 5.12 должен иметь удаленные интерфейсы.



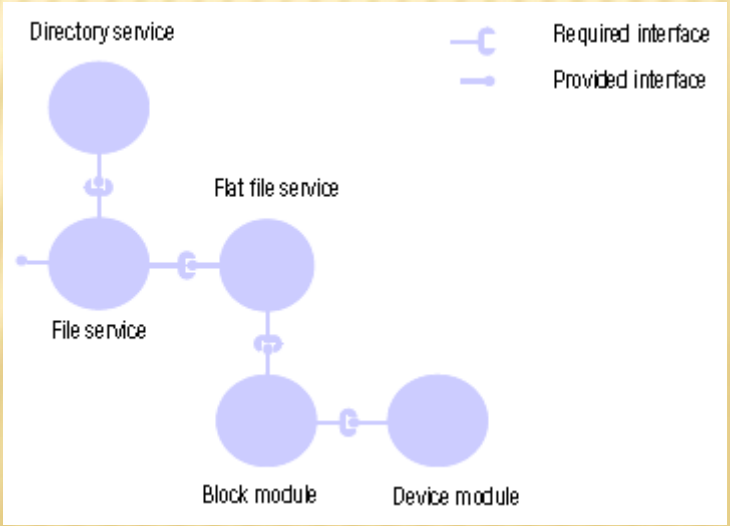

Вызов удаленных и локальных методов

# ОГРАНИЧЕНИЯ РАСПРЕДЕЛЕННЫХ ОБЪЕКТОВ

- **Неявные зависимости**: объектные интерфейсы не описывают, как реализация объекта зависит от объекта, что затрудняет разработку объектно-ориентированных систем (особенно для сторонних разработчиков) и впоследствии управление ими.
- **Сложность программирования**: программирование промежуточного ПО для распределенных объектов приводит к необходимо овладеть многими низкоуровневыми деталями, связанными с реализациями промежуточного программного обеспечения.
- **Отсутствие разделения проблем распространения**: разработчики приложений обязаны рассмотреть детали таких проблем, как безопасность, обработка сбоев и параллелизм, которые во многом похожи от одного приложения к другому.
- **Нет поддержки для развертывания**: объектно-ориентированное промежуточное ПО практически не поддерживает для развертывания (потенциально сложных) конфигураций объектов.

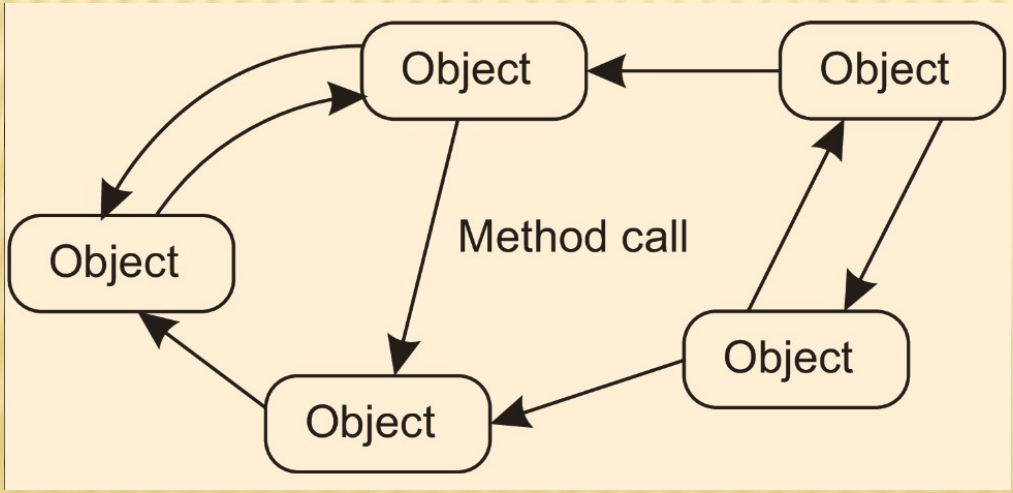
# ПОНЯТИЕ РАСПРЕДЕЛЕННОГО КОМПОНЕНТА

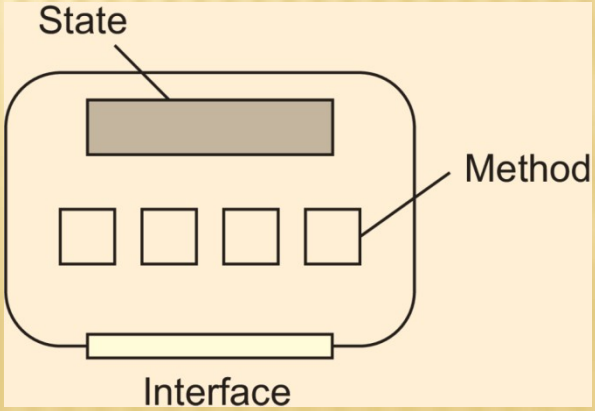
- Компонентные решения являются естественной эволюцией объектно-ориентированных решений. В качестве сущности используется понятие распределенного компонента:
- Распределенный компонент это сущность для которой определены интерфейсы и ее состояние.
- Для компонента могут быть определены следующие интерфейсы:
  - набор предоставленных интерфейсов, то есть интерфейсов, которые компонент предлагает как сервисы для других компонентов;
  - набор необходимых интерфейсов - то есть зависимости, которые этот компонент имеет в терминах других компонентов, которые должны присутствовать и подключаться к этому компоненту для его правильной работы.



# АРХИТЕКТУРА ОСНОВАННАЯ НА ОБЪЕКТАХ (OBJECT-BASED ARCHITECTURE)

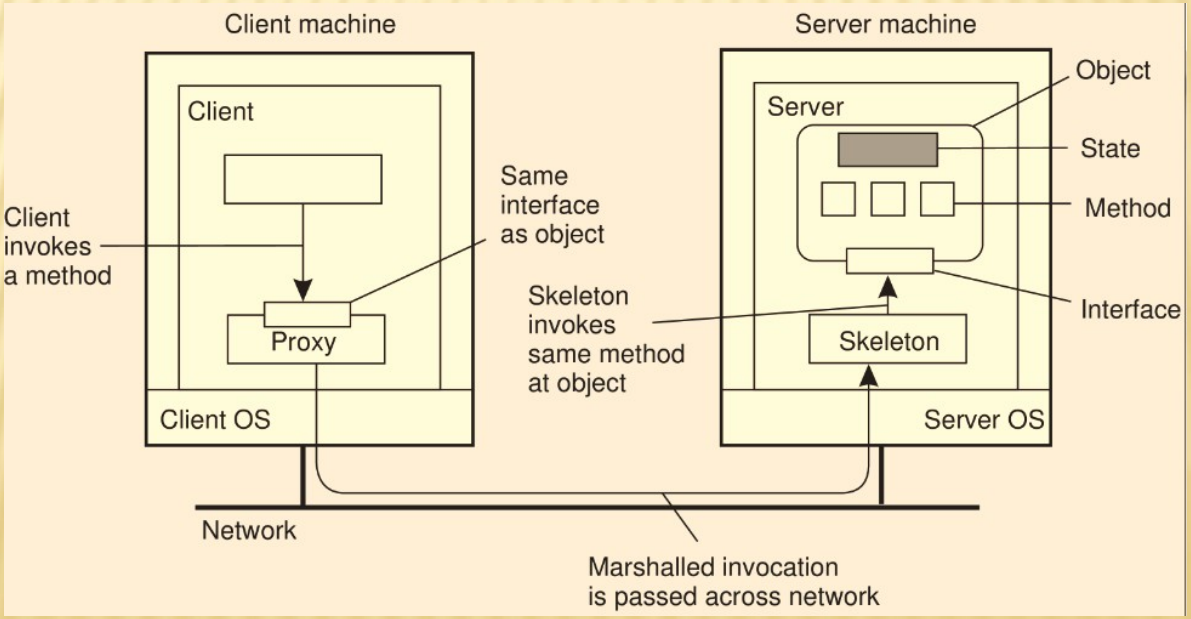
- В этой архитектуре понятие **объект** однозначно соответствует понятию **компонент**.
- Объекты взаимодействуют между собой с помощью механизма **RPC.**
- Эта архитектура предоставляет естественный способ инкапсуляции данных (**состояние объекта**) и операций которые могут выполняться над этими данными (**методы объекта**) объектом.
- **Интерфейс** предоставляемый объектом скрывает детали реализации объекта, что делает объект полностью независимым от его окружения.
- Разделение интерфейсов и реализации объектом этого интерфейса позволяет разместить интерфейс объекта на одной машине, а сам объект на другой машине. Такой объект называется **распределенным**.



# КОММУНИКАЦИИ МЕЖДУ РАСПРЕДЕЛЕННЫМИ ОБЪЕКТАМИ

- Когда клиент связывается с распределенным объектом, реализация интерфейса этого объекта, называемая **proxy**, загружается в адресное пространство клиента.
- Посредник (proxy) это аналог заглушки (stub) клиента в RPC. Он выполняет маршалинг вызова метода в сообщение и извлечение результата из ответного сообщения получаемого от сервера.
- Реальный объект располагается на удаленной машине, где имеется такой же интерфейс как и на клиентской машине. Входящий вызов метода принимает серверная заглушка (stub, также часто называемая skeleton), которая извлекает вызов из сообщения и передает его интерфейсу объекта сервера. Заглушка сервера размещает ответ в сообщение и отправляет его proxy клиента.



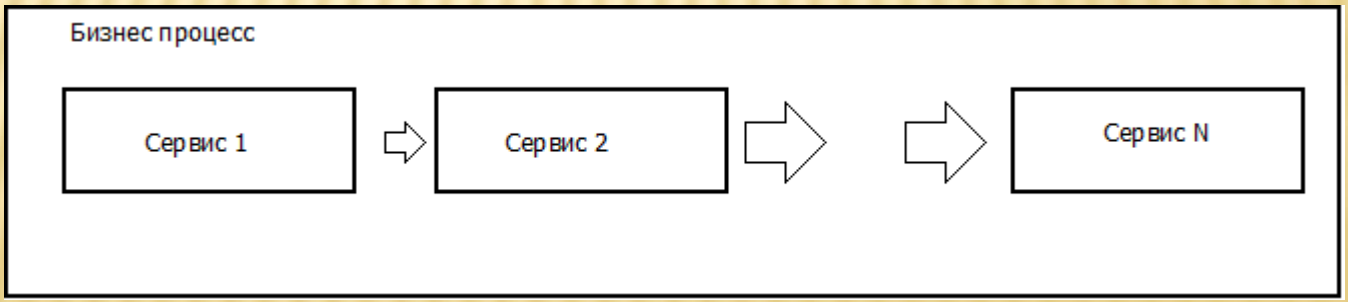
# АРХИТЕКТУРА ОСНОВАННАЯ НА СЕРВИСАХ

# СЕРВИС

- Архитектура распределенных объектов позволяет на ее основе сформировать **сервис** как независимую программную единицу.
- Сервис реализуется как **самодостаточная сущность** созданная на основе распределенного объекта.
- Понятие сервиса как независимой самостоятельной единицы РИС, позволяет определить сервис-ориентированную архитектуру.

# ОСНОВНЫЕ ПОНЯТИЯ СЕРВИС ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ (1)

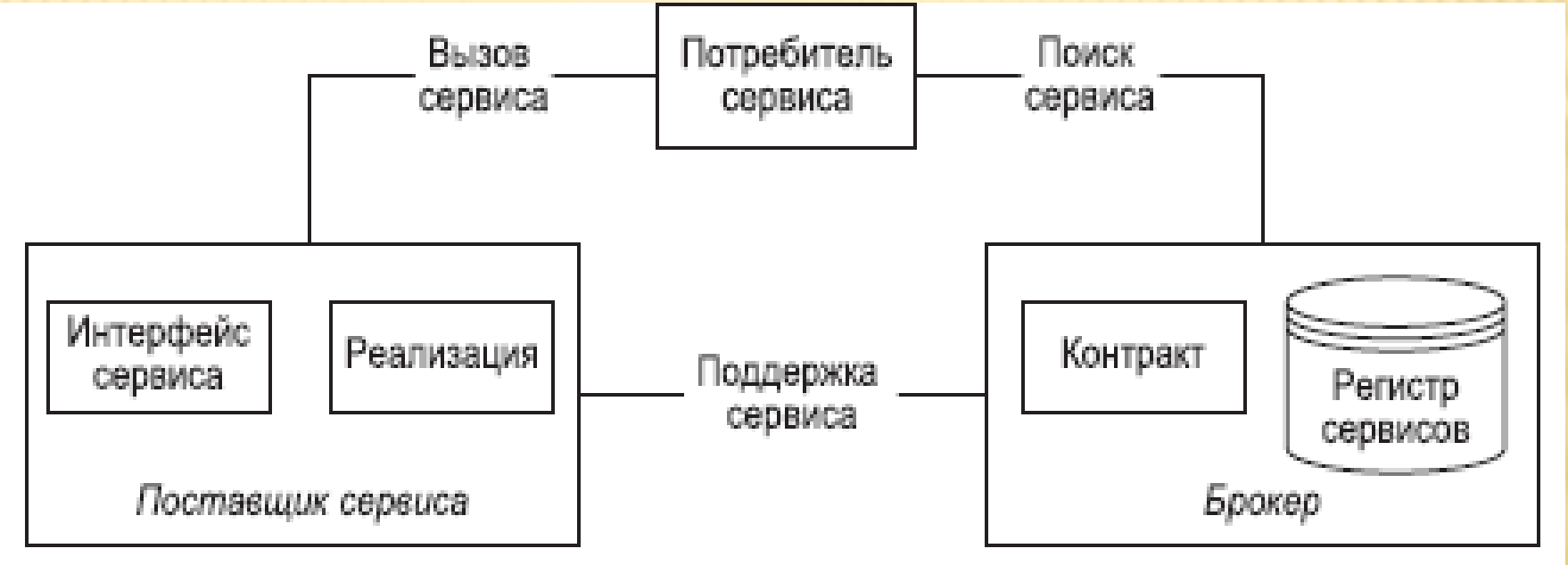
- **СОА –** это способ описания требований и методология предоставления сервисов, независимых от программных платформ и языков разработки, для использования при создании распределенных приложений.
- **Сервис –** это повторно-исполняемая задача в рамках бизнес процесса.
- **Процесс** (бизнес задача) – это композиция составленная из отдельных сервисов.
- Процесс определяет логику взаимодействия сервисов, независимо от их реализации.



# ОСНОВНЫЕ ПОНЯТИЯ СОА (2)

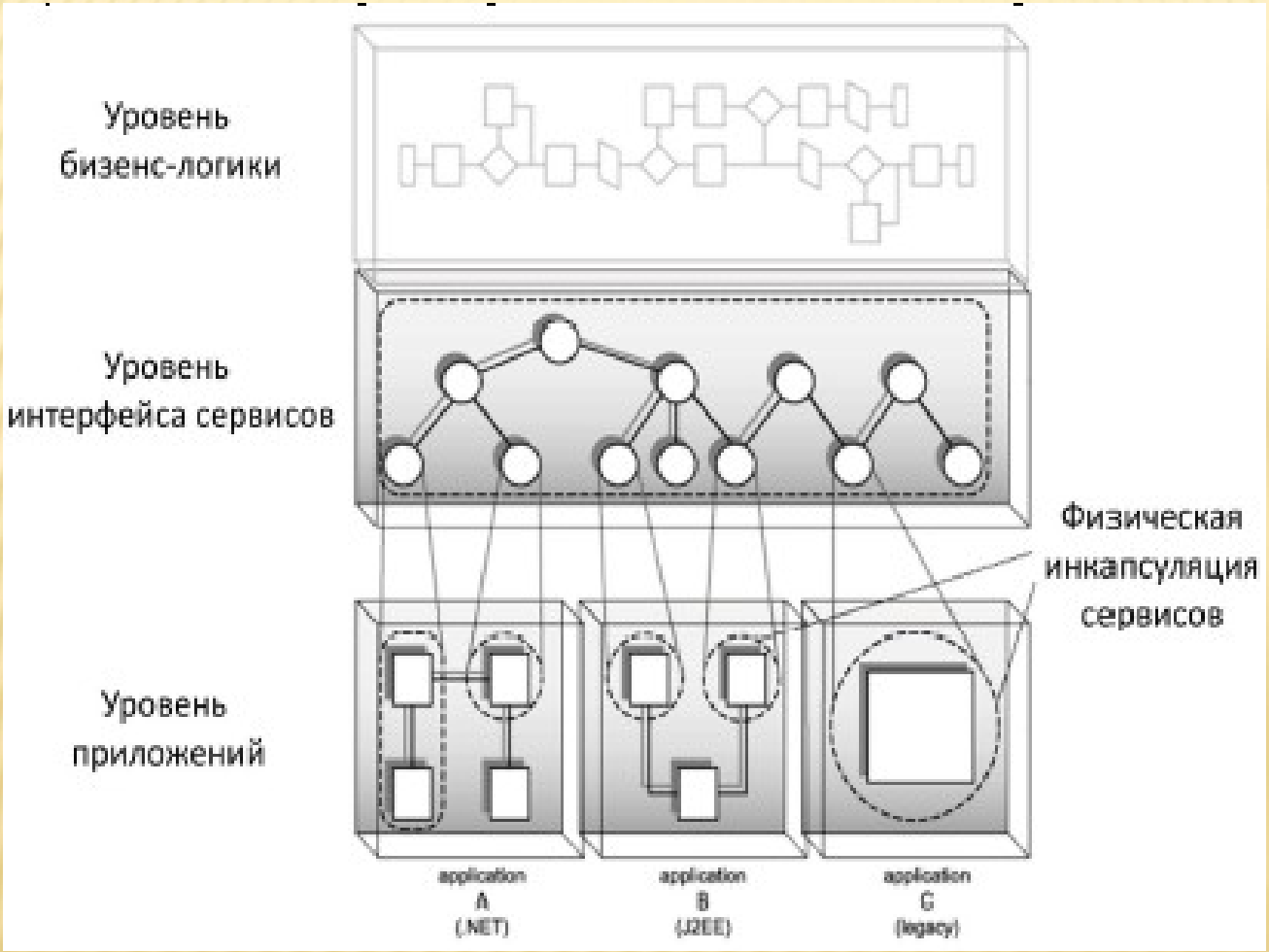
- SOA описывается как  совокупность  сервисов, реализуемых в виде компонентных объектов, систематизированных на основе  обмена сообщениями (message-passing taxonomy).
- Общим примером сообщений, которыми обмениваются сервисы является XML сообщение.
- Для каждого сервиса принято определять:
  - поставщика сервиса (компонент сервис),
  - потребителя сервиса (компонент клиент);
  - брокера - компонент, обеспечивающий взаимодействие поставщика и потребителя. Брокер выполняет функции ПО промежуточного слоя, обеспечивающего взаимодействие поставщика и потребителя.

# ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПОСТАВЩИКОМ И ПОТРЕБИТЕЛЕМ СЕРВИСА




- Клиент обращается к поставщику посылая ему сообщение стандартного формата, содержащее метеданные на основе которых поставщик и будет действовать.
- Поставщик  на основе полученных в сообщении метаданных формирует ответ.
- Ответ принятый от поставщика клиент может использовать по  своему усмотрению.

# АРХИТЕКТУРА ИС ПРЕДПРИЯТИЯ НА ОСНОВЕ SOA




- Уровень бизнес-логики предприятия
- Уровень логики приложения
- Уровень приложения

# ПРИНЦИП СЛАБОЙ СВЯЗИ

- В определение SOA входит понятие слабой связи сервисов. Этот термин подразумевает, что взаимодействующие программные компоненты имеет минимальное знание друг о друге:
  - они находят информацию, которая им нужна для взаимодействия непосредственно перед взаимодействием.

# ДОСТОИНСТВА СЛАБОЙ СВЯЗИ

- ***Гибкость:*** *сервис может быть расположен на любом сервере, а при необходимости* - перемещен. Пока ссылка на этот сервис есть в службе регистрации, предполагаемые клиенты будут в состоянии найти и использовать его.

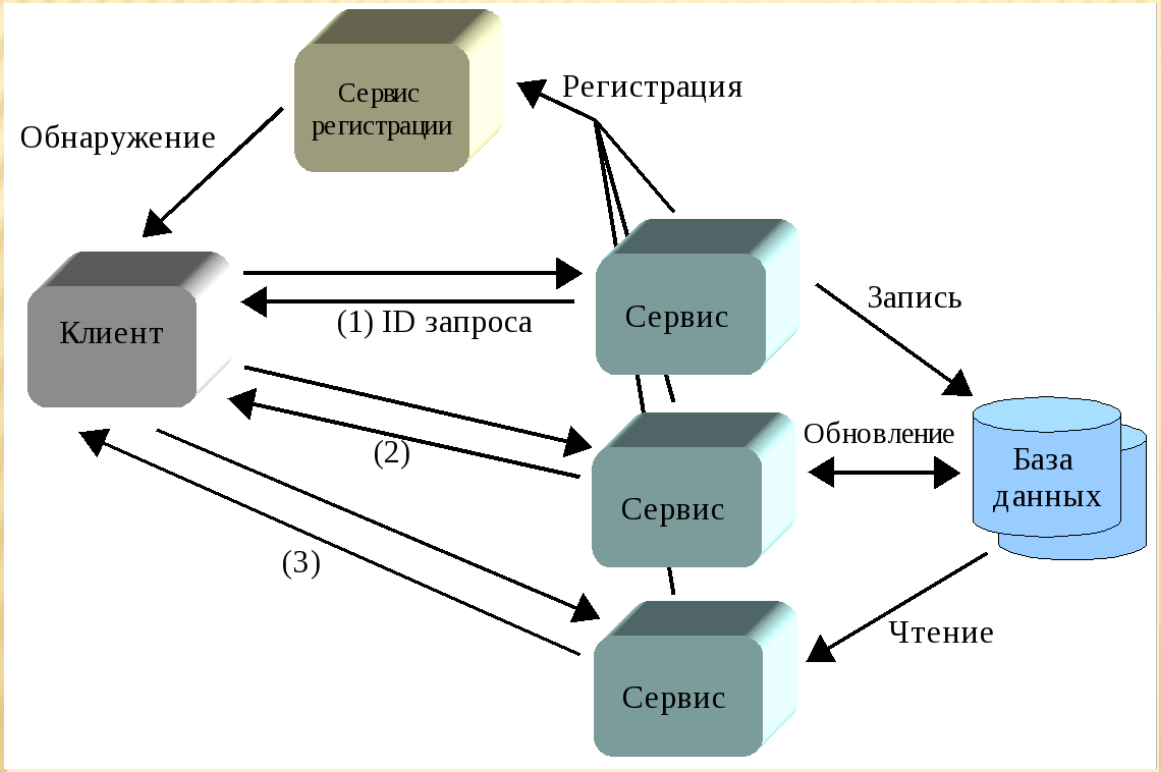
- ***Масшабируемость:*** *функциональные возможности сервиса могут быть расширены* или сужены, поскольку при этом описание сервиса динамически меняется и, соответственно, изменяются и запросы.

- ***Возможность модификации реализации:*** *при условии, что оригинальные* интерфейсы сохраняются, реализация сервиса может быть обновлена без сбоев в обслуживании клиентов.

- ***Отказоустойчивость:*** *Если возникают проблемы в работе сервера, программного* компонента или сегмента сети, или сервис становится недоступным по любой другой причине, клиенты могут сделать запрос к службе регистрации для обнаружения другого сервиса, который предоставляет требуемые услуги.

# СОСТОЯНИЕ СЕРВИСА

- в теории слабосвязанных сервисов важными являются понятия:
  - ≪состояние сервиса≫;
  - ≪сервис без состояний≫.




Для сложных запросов, которые требуют нескольких шагов, сервис сохраняет в своей локальной памяти некоторую информацию ("состояние") о первом шаге, предполагая использовать ее, когда клиент входит с ним в контакт на следующем шаге.

В этом случае, говорят, что сервис обладает состоянием (stateful service), и клиент должен возвратиться к тому же самому сервису при следующем шаге.

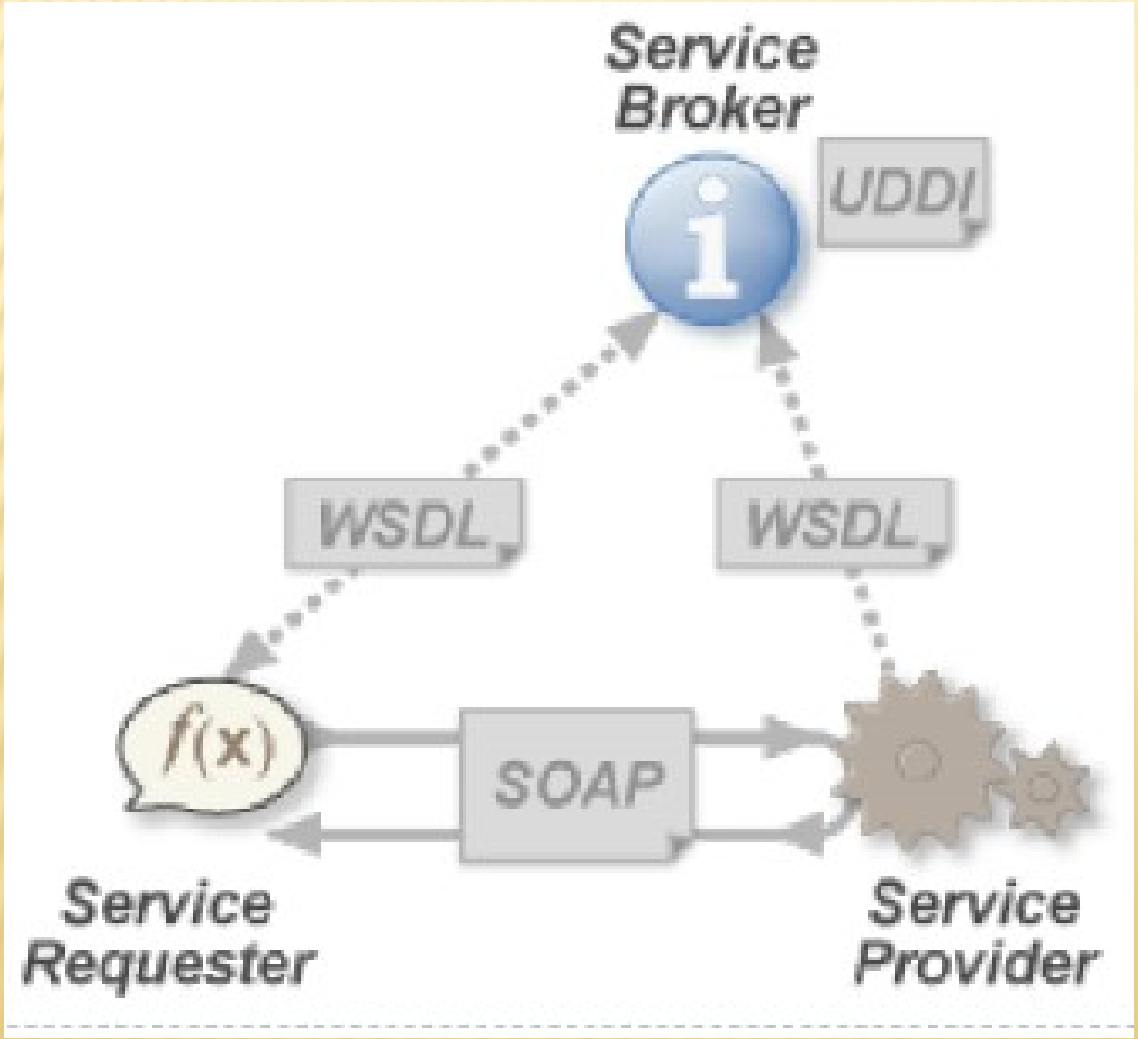
# ВЕБ-СЕРВИСЫ

- Веб-сервисы (или веб-службы) – это распределенные программные компоненты, идентифицируемые своим сетевым адресом, интерфейс которых описан на специальном ≪диалекте≫ языка XML (eXtensible Markup Language), а именно WSDL (Web Service Description Language).
- Другие программные системы могут взаимодействовать с веб-сервисами согласно этому описанию посредством сообщений, основанных на другом ≪диалекте≫ XML - SOAP, и передаваемых с помощью интернет-протоколов

# ВЕБ-СЕРВИСЫ И SOA

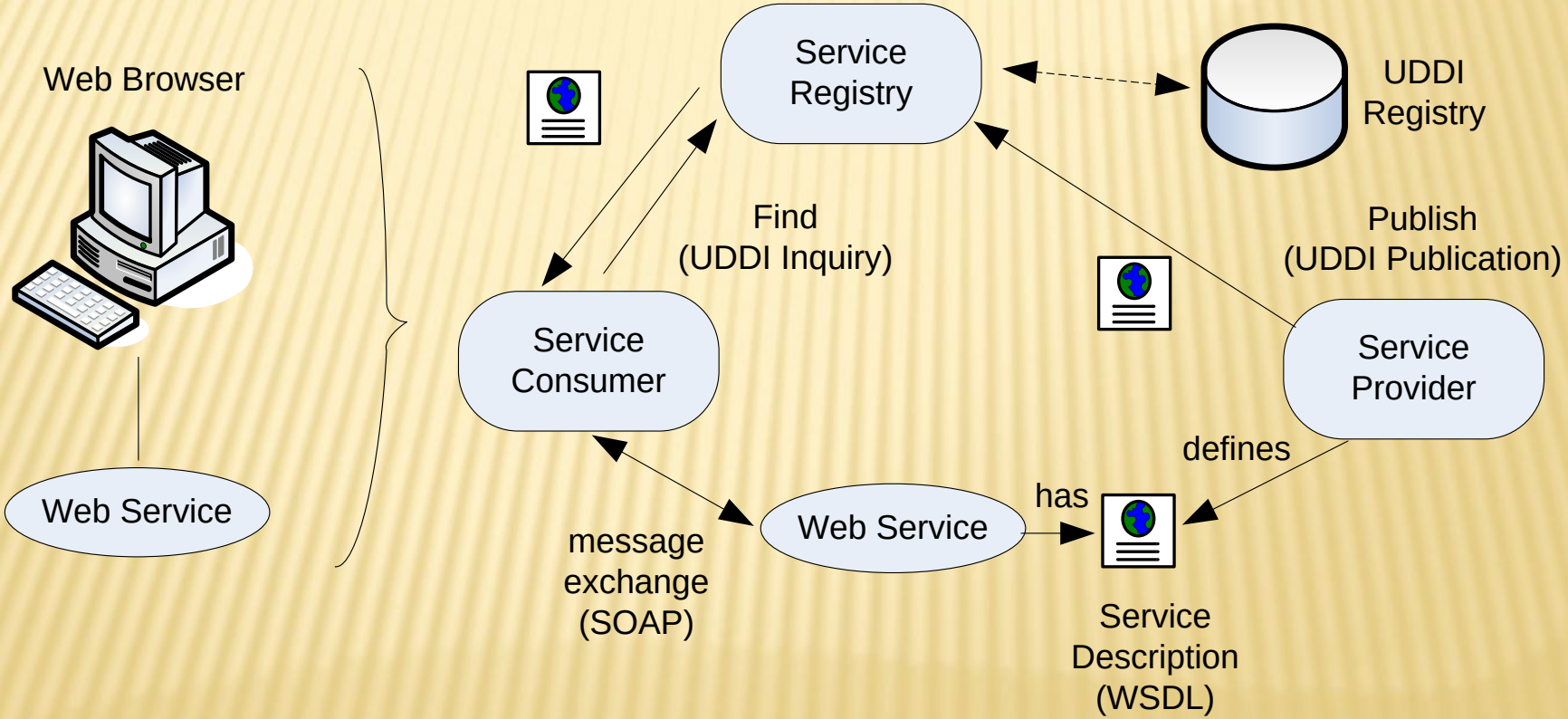
- SOA и веб-сервисы являются ≪ортогональными≫ понятиями:
  - сервисная ориентация – это архитектурный стиль построения системы,
  - веб-сервис – технология реализации сервиса.
- Они, конечно, могут использоваться совместно – как это часто и случается, но они взаимно независимы.
- Веб-сервисы хорошо подходят в качестве строительных блоков SOA-среды

# ВЕБ СЕРВИС




- **Веб-служба**, *веб-сервис* (англ. *web service*) — идентифицируемая программная система со стандартизированными интерфейсами.

# МОДЕЛЬ РАБОТЫ WEB-СЕРВИСА



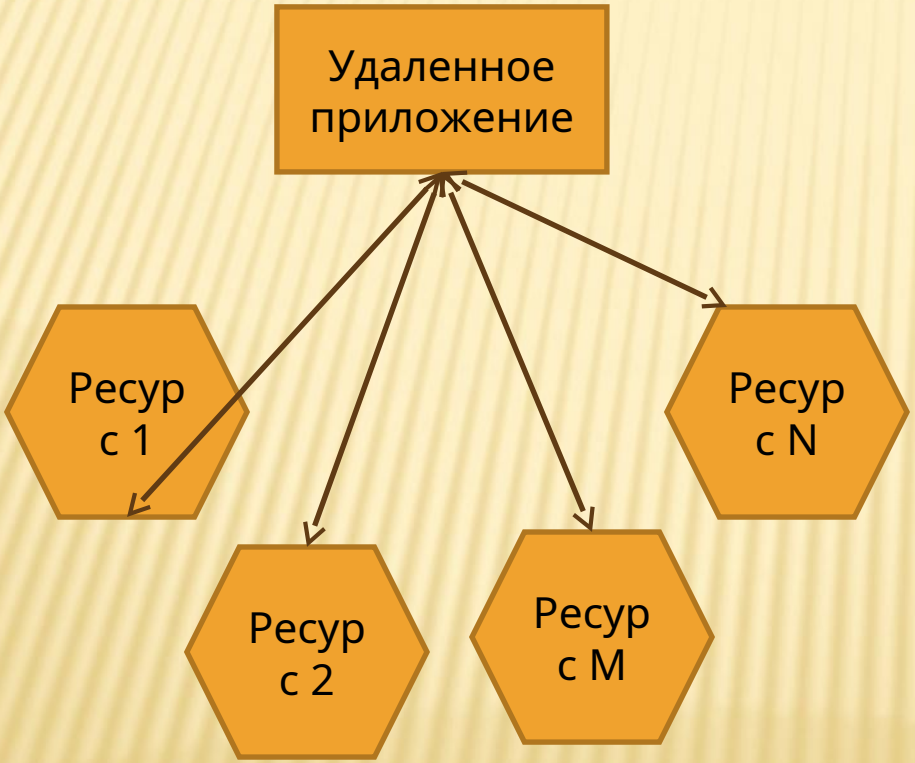
# СТЕК ПРОТОКОЛОВ WEB-СЕРВИСА

# АРХИТЕКТУРА ОСНОВАННАЯ НА РЕСУРСАХ

# АРХИТЕКТУРА ОСНОВАННАЯ НА РЕСУРСАХ

- Рост числа сервисов доступных через Web и создание распределенных систем на основе композиций сервисов привели к необходимости переосмысления архитектуры РИС построенных на базе Web.
- Одной из проблем построения РИС на основанных на Web сервисах явилась высокая сложность обеспечения связи между большим числом различных компонент.
- В качестве альтернативы было предложено рассматривать РИС как коллекцию ресурсов каждый из которых индивидуально управляется своим компонентом (сервисом).

# РЕСУРСО-ЦЕНТРИЧЕСКАЯ АРХИТЕКТУРА




- Ресурсы могут добавляться либо удаляться с помощь приложения (удаленного).
- Такой подход привел формированию понятия ресурсо-центрической архитектуры РИС
- Ресурсный подход получил широкое распространение в WEB в виде Web-сервисов RESTful (Representational State Transfer)

# WEB-СЕРВИСЫ RESTFUL

- Технология REST не привлекла большого внимания в 2000 году, когда Рой Филдинг (Roy Fielding) впервые представил ее в Калифорнийском университете в Ирвайне в своей диссертации "Архитектурные стили и дизайн сетевых архитектур программного обеспечения”
- В 2004 году W3C выпустил определение ещё одного стандарта под названием RESTful.
- За последние несколько лет REST стала преобладающей моделью проектирования Web-сервисов. Фактически REST оказала настолько большое влияние на Web, что практически вытеснила дизайн интерфейса, основанный на SOAP и WSDL, из-за значительного более простого стиля проектирования.
- В последние годы этот стандарт стал довольно популярным. На данный момент он используется многими известными сайтами по всему миру, всеми публичными облачными системами.

# ПРИНЦИПЫ АРХИТЕКТУРЫ RESTFUL

- Архитектура REST (Representational State Transfer) основана на четырех принципах [Fielding, 2000]:
  1. Использование **единой схемы именования**. Идентификация ресурса выполняется посредством URI, который предоставляет глобальное адресное пространство для поиска ресурсов и сервисов.
  2. **Унифицированный интерфейс** – GET извлекает текущее состояние ресурса в некотором представлении. POST передает новое состояние ресурса. Поддерживается всего 4 операции:
     - PUT – создать новый ресурс;
     - GET – получить состояние ресурса в некоторое представление;
     - DELETE – Удалить ресурс;
     - POST – Модифицировать ресурс с помощью изменения его состояния.
  3. **Информативные сообщения** – являются самодостаточными. Ресурсы отделены от их представления таким образом, что их содержимое может быть доступно в различных форматах (например, HTML, XML, текст, RDF, JPEG).
  4. **Отсутствие сессии**. После формирования ответа сервер забывает о клиенте. Взаимодействия через гиперссылки – Для обмена существуют различные технологии (например, переименование URI, cookies и скрытые поля формы). Состояние может быть вставлено в ответное сообщение, чтобы указать допустимое будущее состояние взаимодействия.

# Явное использование HTTP-методов

- Этот основной принцип проектирования REST устанавливает однозначное соответствие между операциями create, read, update и delete (CRUD) и HTTP-методами.
- Согласно этому соответствию:
  - Для создания ресурса на сервере используется POST.
  - Для извлечения ресурса используется GET.
  - Для изменения состояния ресурса или его обновления используется PUT.
  - Для удаления ресурса используется DELETE.

# Примеры правильного и неправильного использования команд HTTP

- Использование GET не по назначению, например добавление данных в базу:
  - GET /adduser?name=Robert HTTP/1.1

- Проблема здесь в основном семантическая. Web-серверы предназначены для ответов на HTTP-запросы GET путем извлечения ресурсов согласно URI запроса (или критерию запроса) и возврата их или их представления в ответе, а не для добавления записи в базу данных.

- Использование GET по назначению, например добавление данных в базу:
- Простым способом решения этой общей проблемы является помещение имен и значений параметров URI запроса в XML-теги. Эти теги (XML-представление создаваемого объекта) можно отправить в теле HTTP-запроса POST, URI которого является родителем объекта:

Занесение:

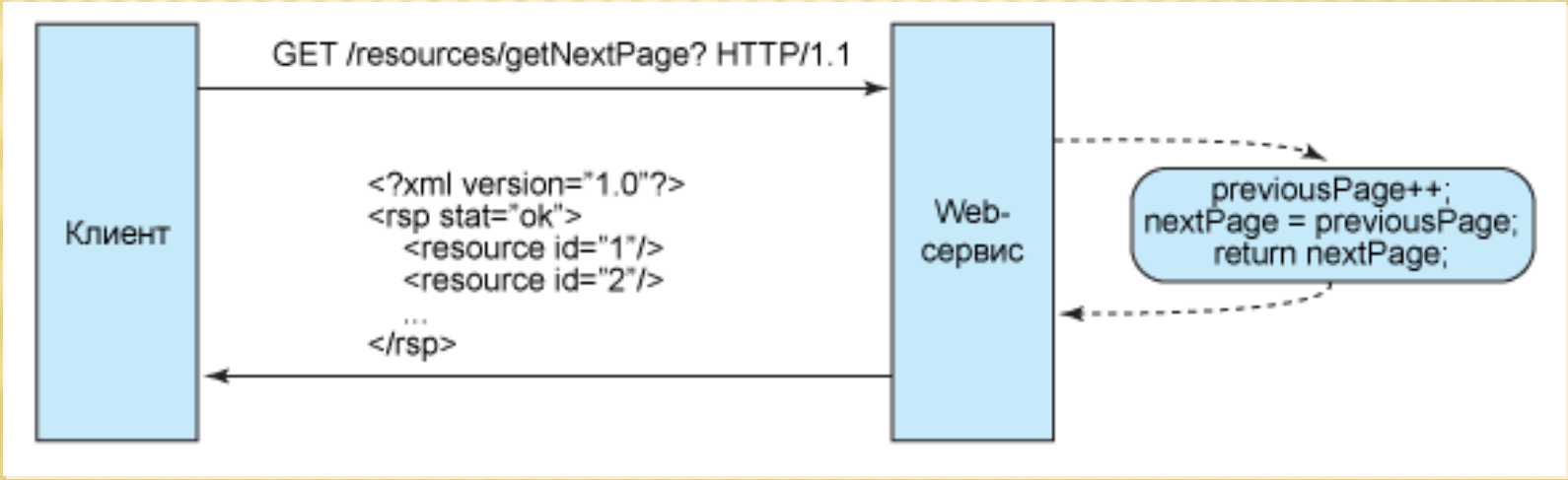
```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

Извлечение:

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

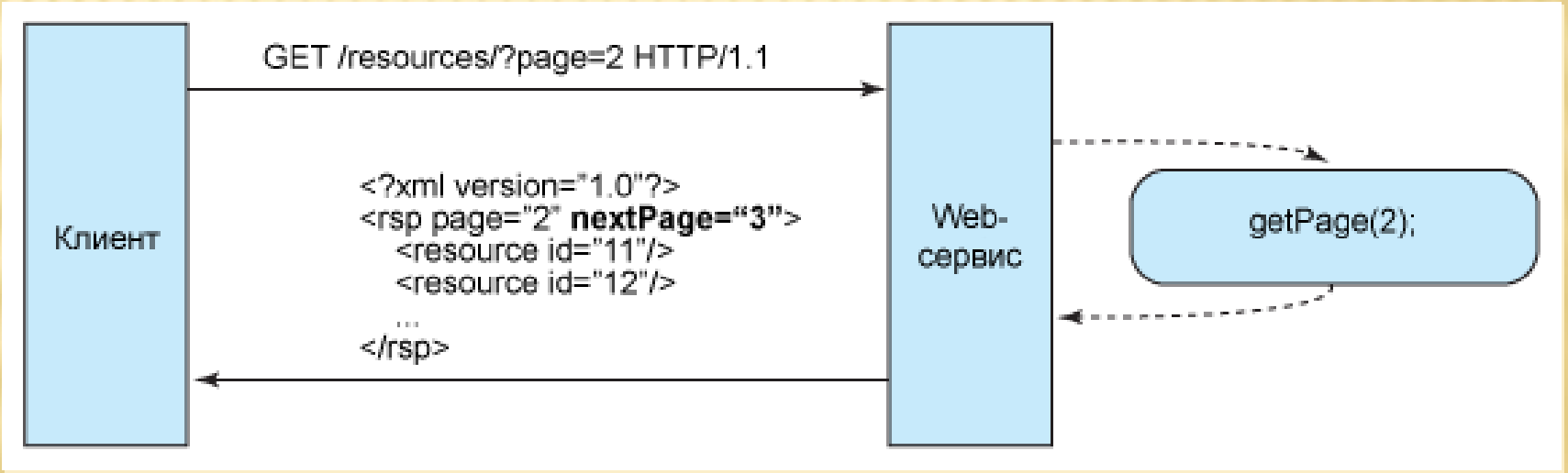
# WEB СЕРВИС С СОХРАНЕНИЕМ СОСТОЯНИЯ

- На рисунке показан сохраняющий состояние сервис, в котором приложение может запросить следующую страницу в многостраничном наборе результатов, полагая, что сервер хранит последовательность переходов приложения по этому набору результатов.
- В этой модели с сохранением состояния сервис наращивает и сохраняет переменную previousPage



# WEB-СЕРВИС БЕЗ СОХРАНЕНИЯ СОСТОЯНИЯ

- В Web-сервисе RESTful сервер отвечает за генерирование ответов и за предоставление интерфейса, позволяющего клиентскому приложению самому хранить свое состояние. Например, при запросе многостраничного набора результатов клиентское приложение должно включать в запрос номер конкретной страницы, а не просто запрашивать *следующую* страницу (см. рисунок 2).



# Структура URI аналогичная структуре каталогов.

- Структура URI должна быть простой, предсказуемой и понятной.
- Один из способов достичь такого уровня удобства использования – построение URI по аналогии со структурой каталогов. Такого рода URI являются иерархическими, исходящим из одного корневого пути, ветвления которого отображают основные функции сервиса.
- Например, в сервисе обсуждений различных тем:
  - http://www.myservice.org/discussion/topics/{topic}
  - http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}
- Дополнительные рекомендации, при обдумывании структуры URI для Web-сервисов RESTful:
- Скрывайте расширения файлов серверных сценариев (.jsp, .php, .asp), чтобы можно было выполнить портирование приложений на другую технологию без изменения URI.
- Используйте только строчные буквы.
- Заменяйте пробелы дефисами или знаками подчеркивания (чем-то одним).
- Старайтесь максимально избегать использования строк запросов.
- Вместо использования кода 404 Not Found для URI, указывающих неполный путь, всегда предоставляйте в качестве ответа ресурс или страницу по умолчанию.

# ПЕРЕДАЧА XML, JSON ИЛИ ОБОИХ

- Это ограничение, тесно связанный с дизайном Web-сервисов RESTful, относится к формату данных, которыми обмениваются приложение и сервис при работе в режиме запрос/ответ или в теле HTTP-запроса. Здесь особенно важны простота, читабельность и связанность.

- **Общеупотребительные MIME-типы, используемые RESTful-сервисами**

| MIME-тип | Тип содержимого |
|---|---|
| JSON | application/json |
| XML | application/xml |
| XHTML | application/xhtml+xml |

# СРАВНЕНИЕ REST И SOAP/XML WEB-СЕРВИСОВ

1.RESTful архитектура стала популярна благодаря своей простоте.

2.SOAP поддерживает 16 операций, RESTful всего 4.

3.В случае RESTful приложение должно предоставить все параметры запроса с помощью одной операции, а в случае SOAP число передаваемых параметров за одну операцию ограничено, поэтому для передачи всех параметров требуется выполнение нескольких операций.

Например, для создания bucket в AWS S3 (SOAP/XML) надо выполнить:

```
import  bucket
bucket.create (“mybucket”)
```

В случае RESTful:

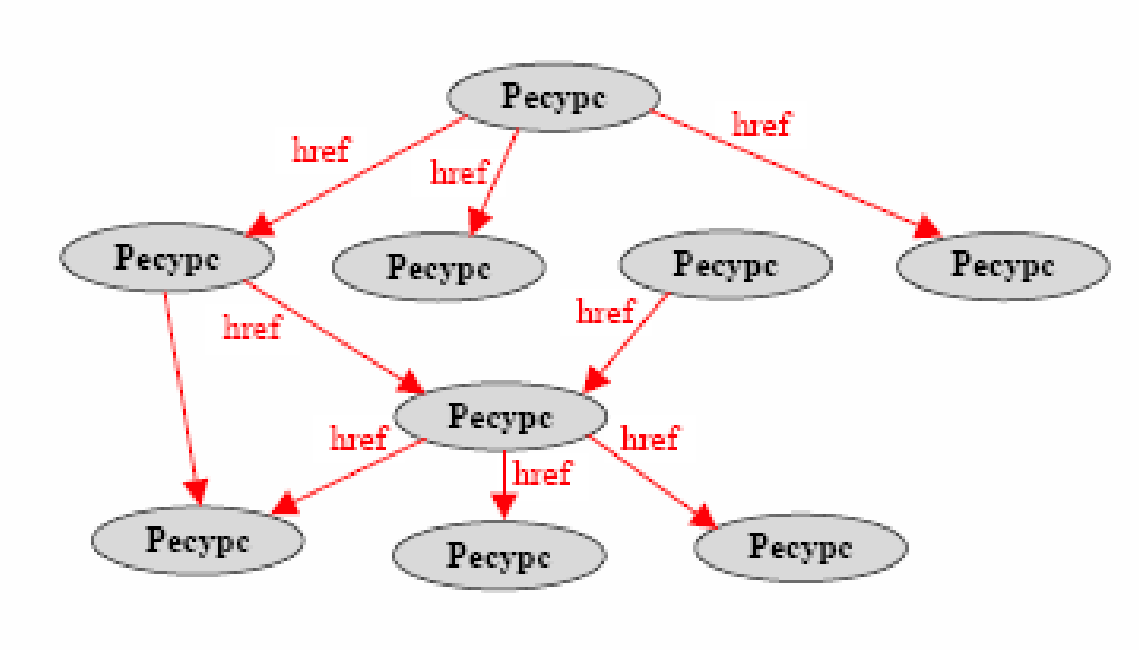
```
PUT “http://mybucket.s3.amazonsws.com/”
```

- Вызов SOAP может порождать синтаксические ошибки, обнаруживаемые во время компиляции, в то время как в случае REST проверка вызова откладывается на время исполнения

# ПОНЯТИЕ СЕМАНТИЧЕСКОГО WEB

# СУЩЕСТВУЮЩИЙ WEB

- Множество HTML-документов(Web-страниц), распределенных в Сети и связанных гипертекстовыми ссылками



# СЕМАНТИЧЕСКИЙ WEB

- Это расширение существующего Web, когда публикуется не только информация в понятном для человека представлении, но и в виде понятном программам, которые способны ее обрабатывать на семантическом уровне.

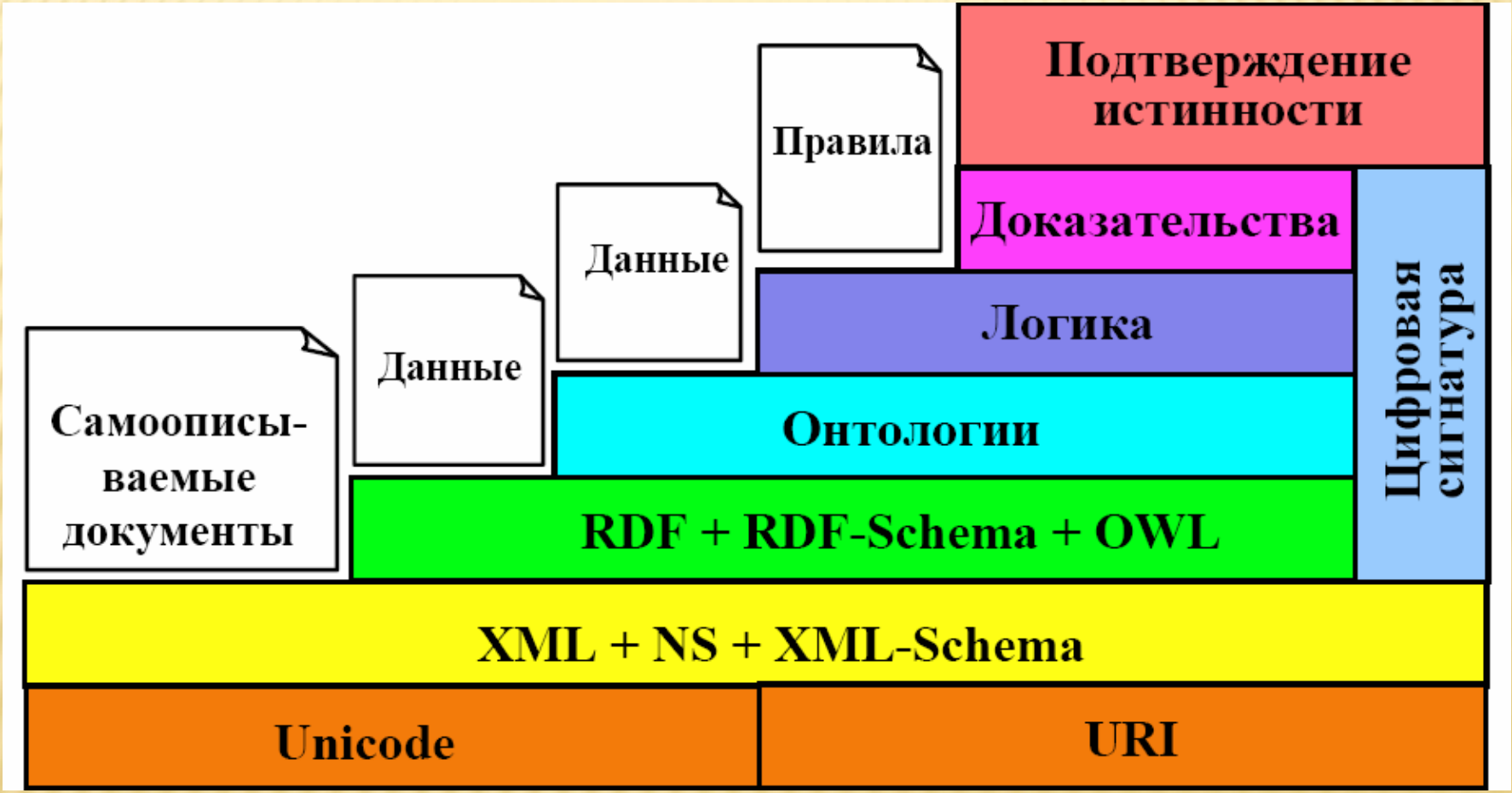
# ТРИ ПРЕДПОЛОЖЕНИЯ О РАСПРЕДЕЛЕННОЙ СЕТИ ДАННЫХ

- ***AAA – Anyone can say Anything about Any Topic :*** (Кто угодно может сказать что угодно о чем угодно)
- ***NNA – Non unique Naming Assumption:***

  (Одинаковые объекты могут иметь различные имена)
- ***OWA – Open World Assumption***

  (Предположение об открытости мира)

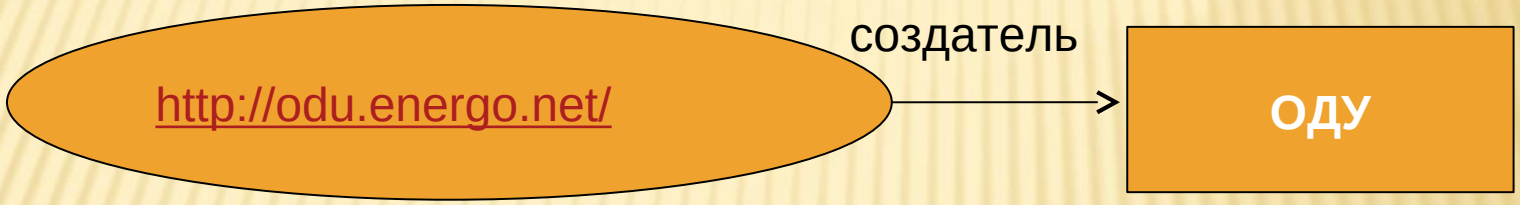
# МНОГОУРОВНЕВОЕ ПРЕДСТАВЛЕНИЕ СЕМАНТИЧЕСКОГО WEB ПО T.BERNERS-LEE



# RDF

- RDF(Resource Description Framework) -язык для описания ресурсов способом, “понятным” компьютеру на семантическом уровне.
  - RDF -официальная рекомендация консорциума W3C с 10.02.2004(http://www.w3.org/RDF)

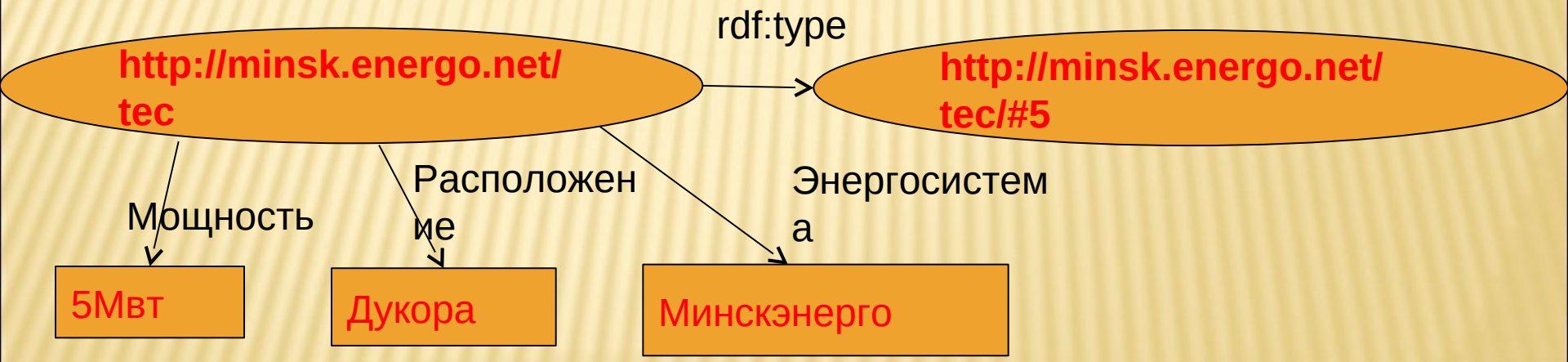
# RDF: ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ



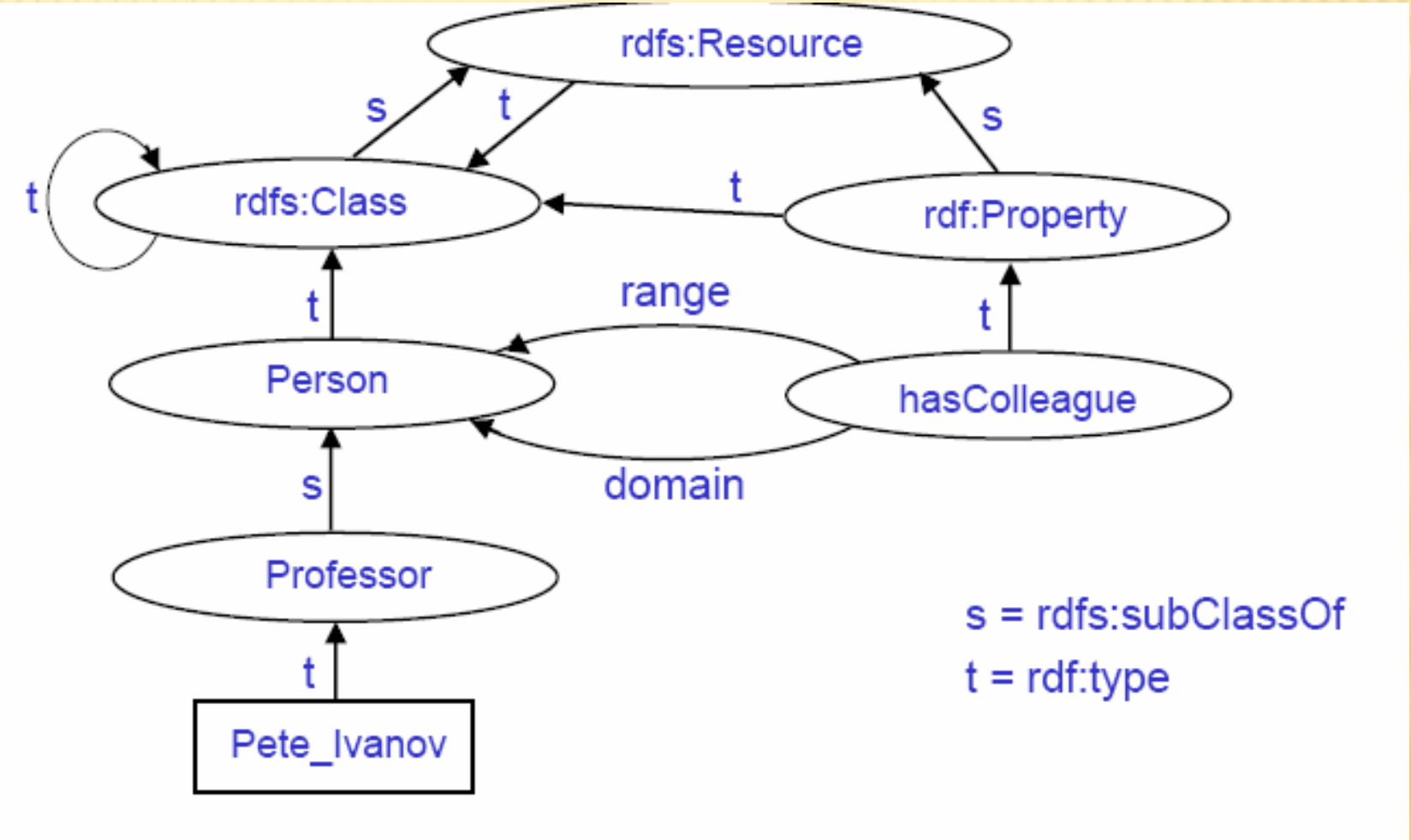

- Свойства также являются ресурсами и имеют URI;
- Имена свойств берутся из пространств имен, которые указываются в виде префикса.

# RDF- МОДЕЛЬ: ПРИМЕР

- Множество утверждений:
  - ТЭЦ-5 тепло-электростанция
  - Установленная мощность  5 Мвт
  - Находиться в п.Дукора Минской области
  - Входит в энергосистему «Минскэнерго»



# ПРИМЕР RDFS


rdfs:Resource
s
t
s
t
rdfs:Class
t
rdf:Property
t
range
t
Person
hasColleague
s
domain
Professor
t
Pete_Ivanov
s = rdfs:subClassOf
t = rdf:type

# НЕДОСТАТКИ RDFS

- Отсутствуют ограничения на области определений терминов и области значений их свойств
- Отсутствуют ограничения существования/кардинальности
- Отсутствуют способы задания отношений:
  - транзитивности;
  - симметричности;
  - инверсности.
- Трудно обеспечить поддержку расуждений

# Язык онтологий для семантического Web

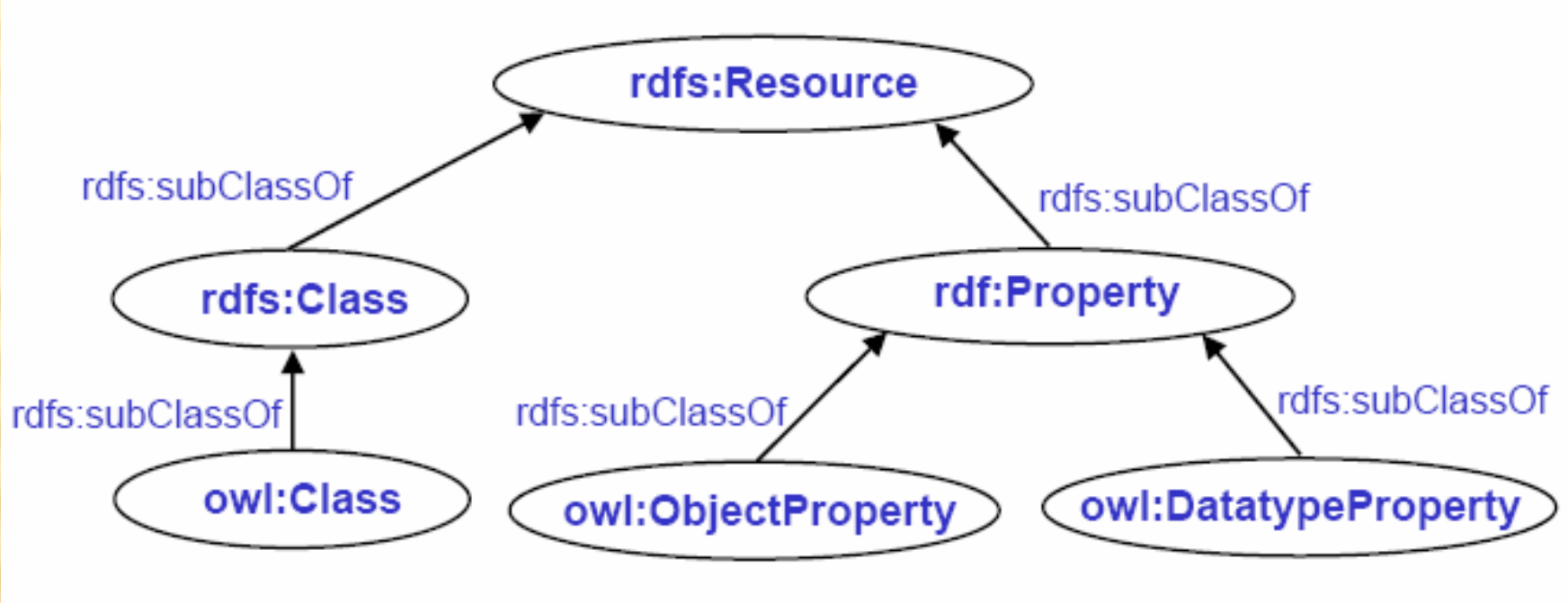
- Язык OWL (Web Ontology Language)
- Три уровня OWL
- –OW L full-объединение OWL-синтаксиса и RDF
- –OWL DL-ограничен фрагментом логики первого порядка (¼DAML+OIL)
- –OWL Lite-“простое для реализации” подмножество OWL DL

# МОДЕЛЬ OWL

OWL имеет объектно- ориентированную модель, основанную на понятиях:

- Объекты/Экземпляры/Индивидуумы:
  - **элементы области**
  - **в логике первого порядка -эквивалентны константам**
- Типы/Классы/Понятия
  - **множества объектов, имеющих общие характеристики**
  - **в логике первого порядка –эквивалентны унарным предикатам**
- Отношения/Свойства/Роли
  - **множества пар (троек) объектов**
  - **в логике первого порядка –эквивалентны бинарным предикатам**

# ОТНОШЕНИЯ МЕЖДУ КЛАССАМИ OWL И RDFS



# МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

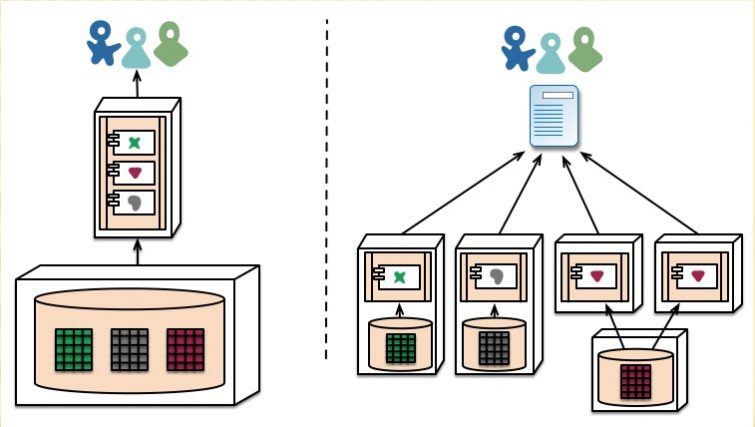
# ПОНЯТИЕ МИКРОСЕРВИСОВ

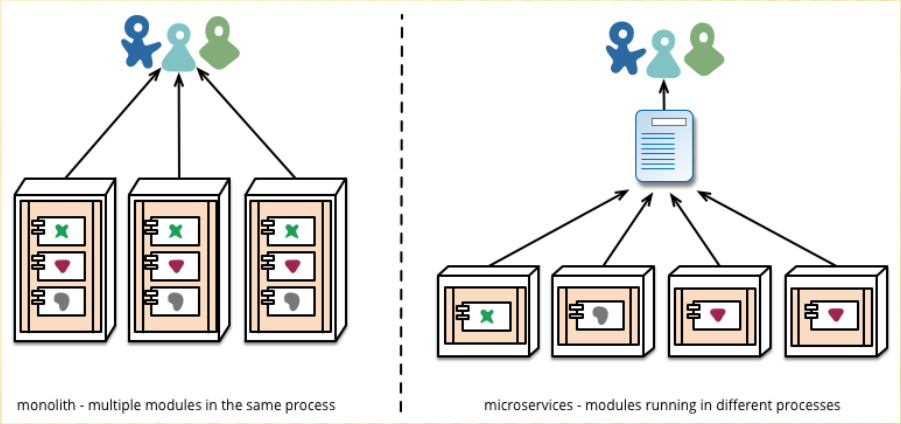
- Микросервисы противопоставляются традиционной монолитной архитектуре. Монолит означает, что компоненты продукта взаимосвязаны и взаимозависимы. Если перестает работать один - все остальные тоже «отваливаются».

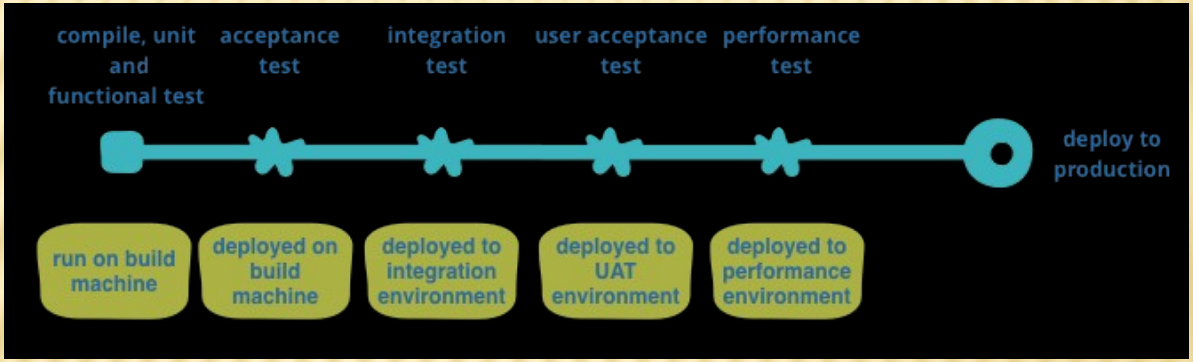
- Предшественником микросервисной архитектуры является сервис-ориентированная архитектура (SOA), которая также разделяет бизнес-логику на компоненты. По сути, микросервисная архитектура - частный случай SOA с набором более строгих правил.
- У микросервисов есть особые свойства, они же преимущества:

- **Гетерогенность**: возможность построить систему с помощью разных языков программирования и технологий;
- **Децентрализованное управление данными**: каждый микросервис содержит свой набор данных, доступный другим микросервисам только через соответствующий интерфейс;
- **Независимость инфраструктуры**: каждый микросервис - независимая единица, поэтому вносить изменения и разворачивать его можно независимо от других;
- **Масштабируемость**: чтобы увеличить производительность системы, нужно расширить только те сервисы, которые в этом нуждаются.

# МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

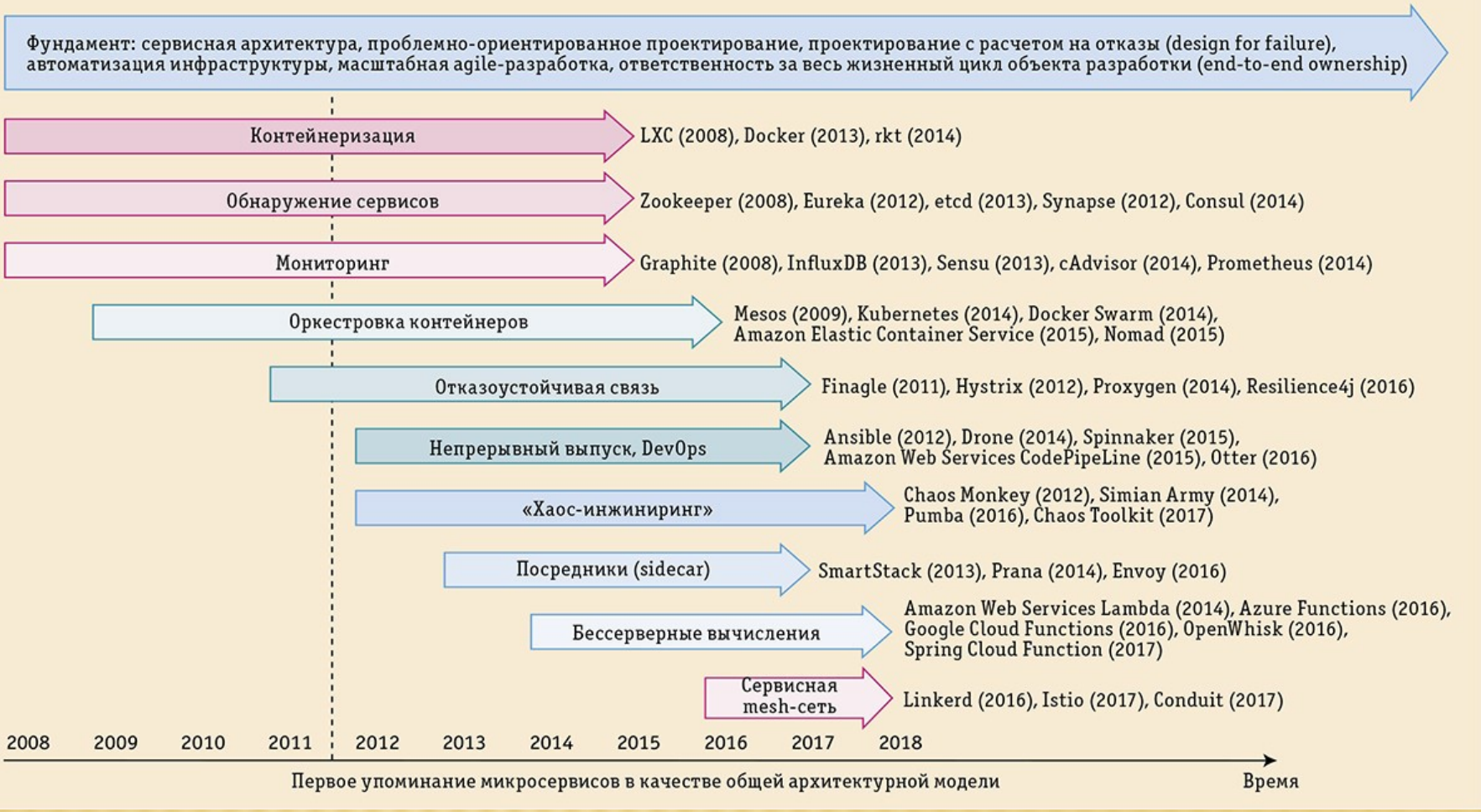




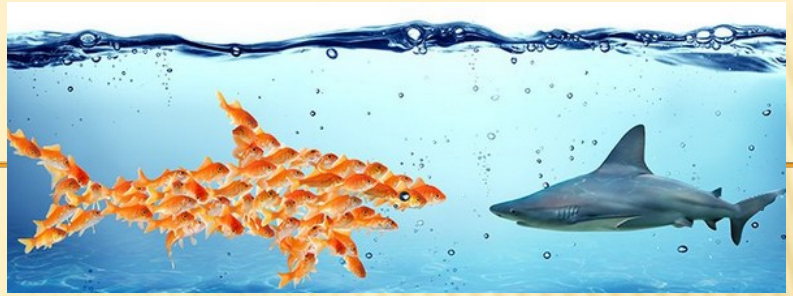




- Множество языков, множество возможностей
- Децентрализованное управление данными
- Использование открытых стандартов типа HTTP
- Автоматизация инфраструктуры
- Проектирование под отказ (с учетом отказов отдельных микросервисов
- Синхронные вызовы считаются опасными
- Каждый сервис работает в отдельном процессе

# ХРОНОЛОГИЯ РАЗВИТИЯ ТЕХНОЛОГИЙ МИКРОСЕРВИСОВ



# МИКРОСЕРВИСЫ ПРОТИВ МОНОЛИТА



- Сегодня скорости разработки сильно возросли. Если в 2005 году можно было разрабатывать продукт несколько лет, сейчас базовую версию нужно выпустить за пару месяцев.

В таких условиях микросервисная архитектура выигрывает у монолита:

- Проще изменить один из микросервисов и сразу внедрить его, чем изменять весь монолит и перезапускать инфраструктуру целиком;
- Новые разработчики легче включаются в работу - для этого им не нужно изучать систему целиком, можно работать только над своей частью;
- Микросервисы не зависят от какой-либо платформы, поэтому внедрять новые технологии проще, чем в монолит.

## Недостатки:

Микросервисы накладывают ограничения на разработку и поддержку продукта:

- Сложность начальной разработки и создания инфраструктуры. Распределенные системы сложнее разрабатывать, т.к. нужно предусмотреть независимость одного микросервиса от сбоя в другом компоненте;
- Нужно правильно выбрать протоколы общения между компонентами, чтобы взаимодействие было максимально эффективно;
- Для распределенной системы сложно поддерживать строгую согласованность:
  - общие части системы нужно либо складывать в общую библиотеку, но тогда при изменении этой библиотеки нужно будет перезапускать и все зависимые микросервисы,
  - либо хранить общий код в каждом из микросервисов, что сложнее поддерживать;

# СПАСИБО ЗА ВНИМАНИЕ !