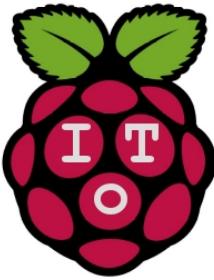


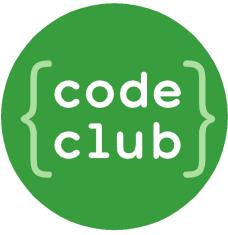
Internet of Things



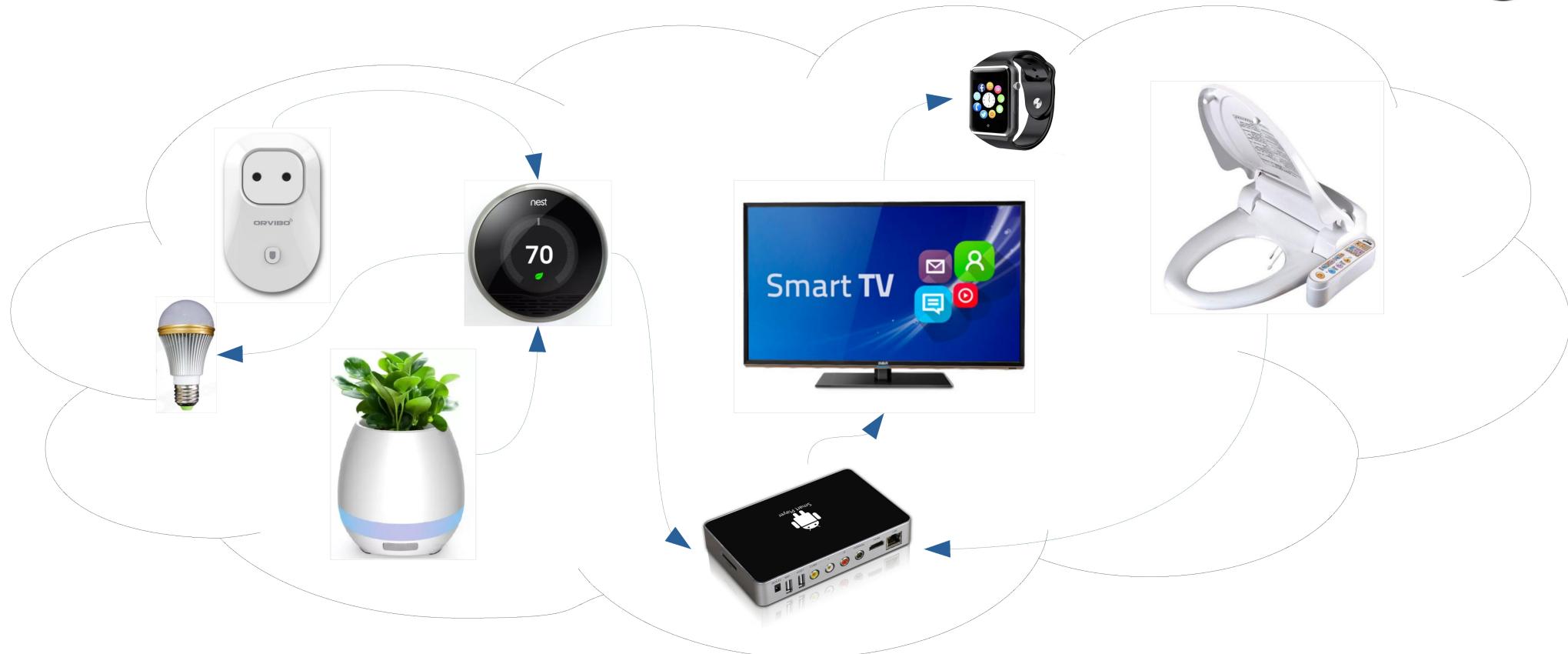
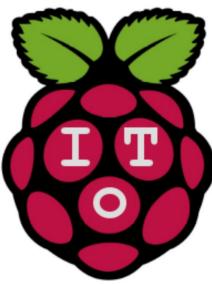
Сетевое взаимодействие
«умных вещей»

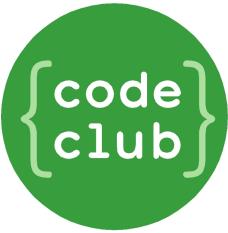
Шадринск
2018-2019

M. V. Шохирев

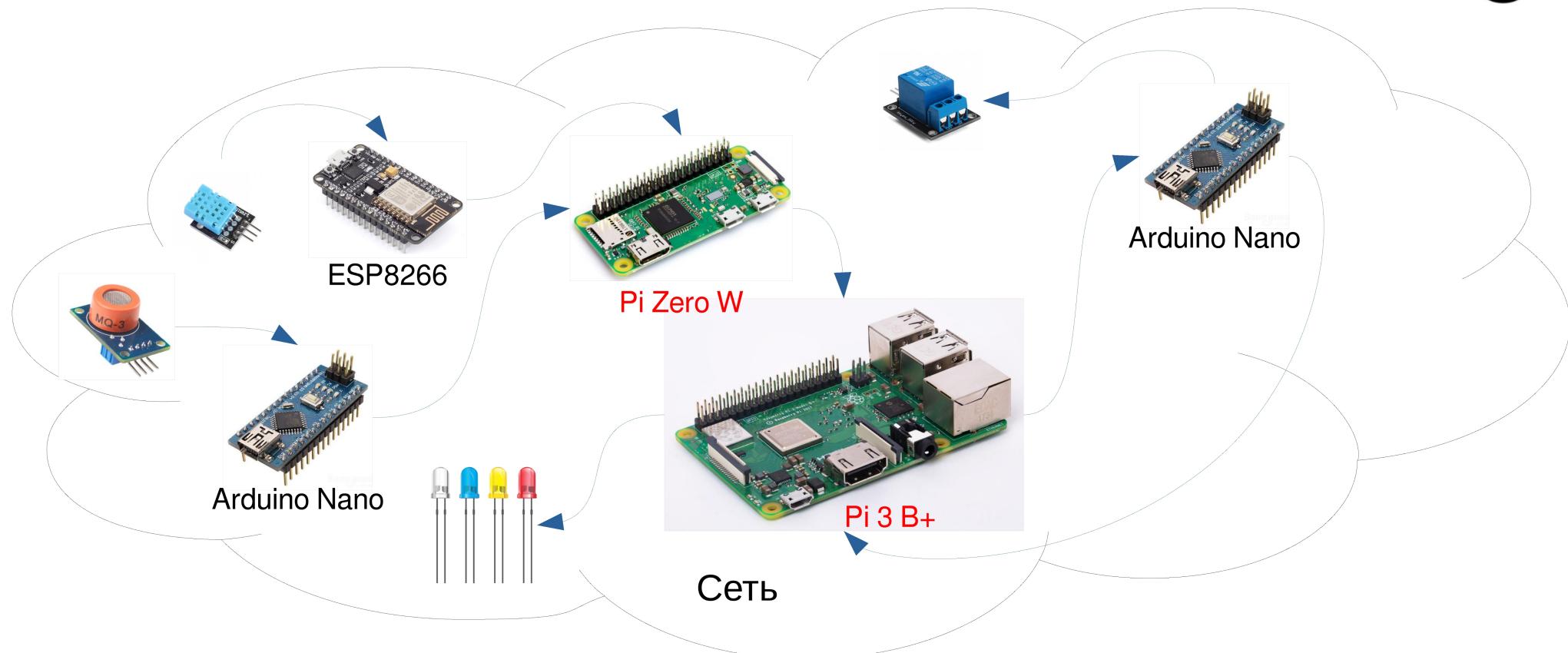
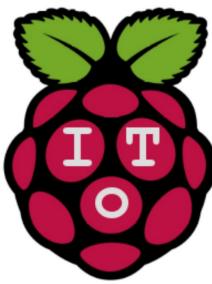


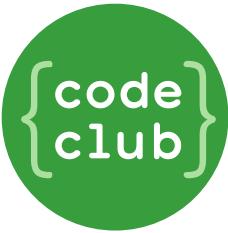
IoT = сеть «умных вещей»



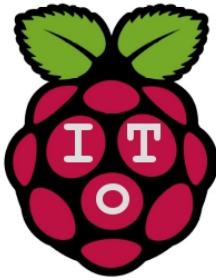


ИОТ = сеть контроллеров!

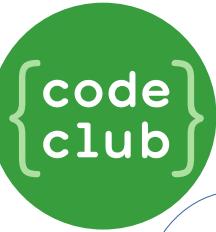




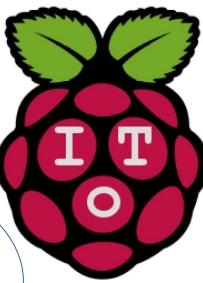
Технологии IoT



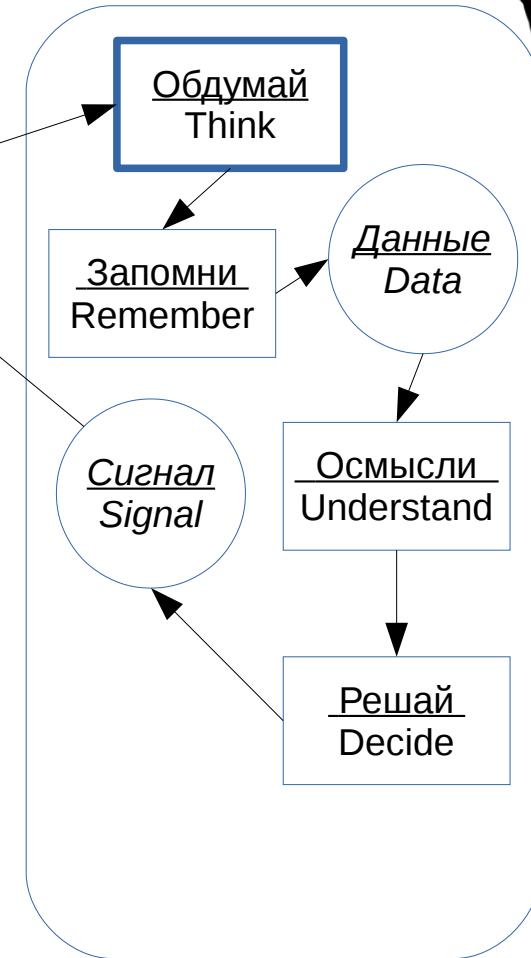
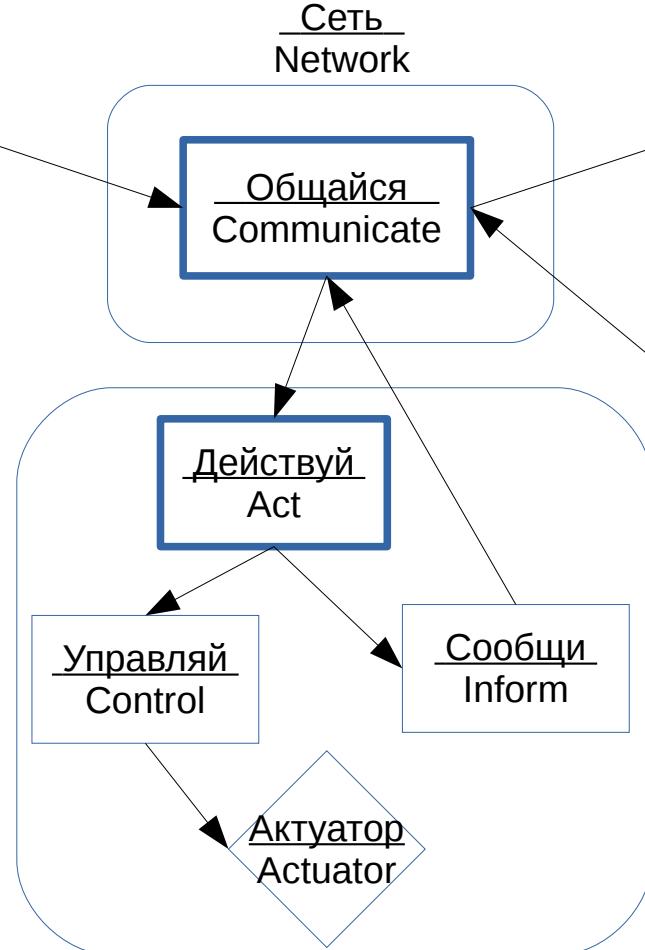
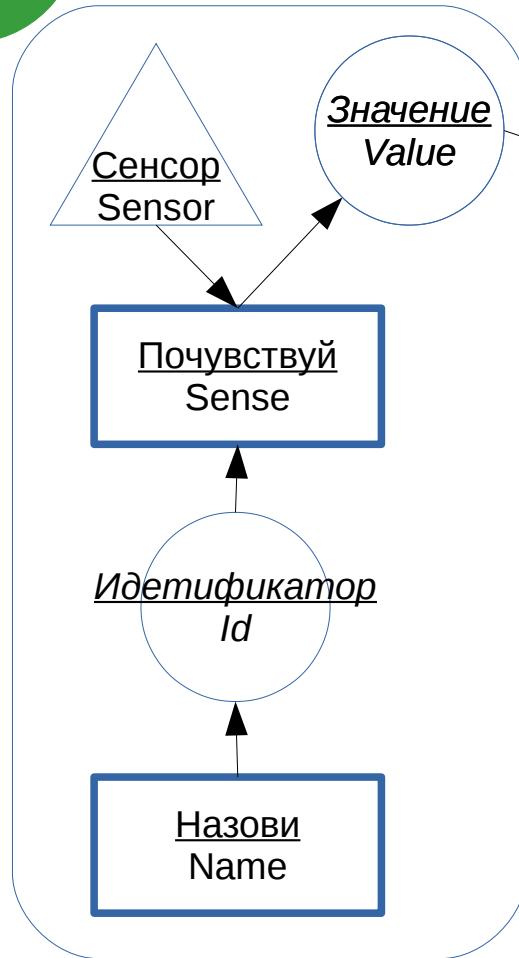
- Идентификация (Identification)
- Сбор данных с датчиков (Sensing)
- Связь по сети (Communication) = *Networking ~ сетевое взаимодействие*
- Хранение данных (Data Storing)
- Обработка данных (Data Processing)
- Выполнение действий (Acting)



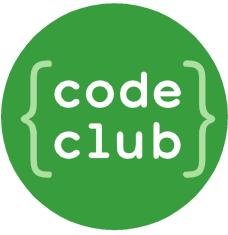
Взаимодействие в IoT



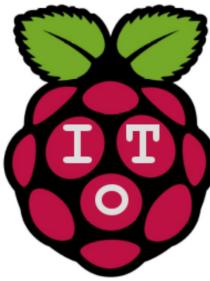
Клиент
Client



Сервер
Server



Распределённая система



Узел сети
Node

Чтение
показаний
датчиков

Чтение
показаний
датчиков

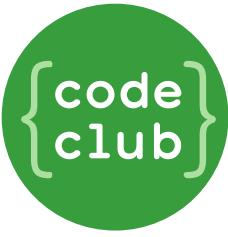
Сбор
данных

Хранение
данных

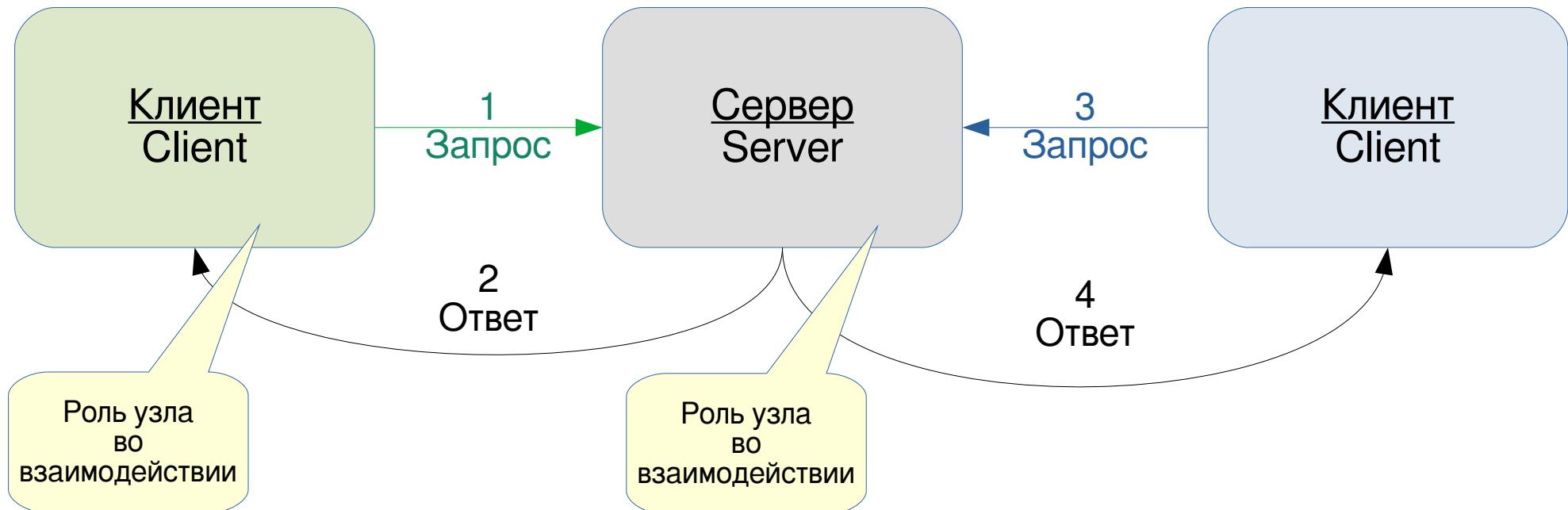
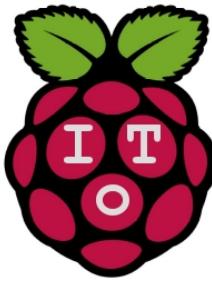
Анализ
данных
Принятие
решений
Отправка
команд

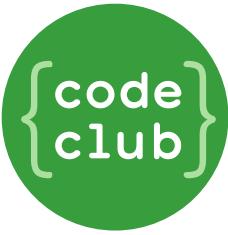
Выполнение
команд
Обратная
связь о
выполнении

Сеть

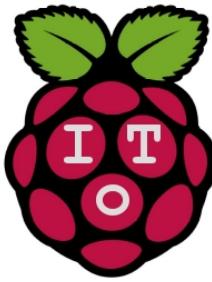


Архитектура клиент–сервер



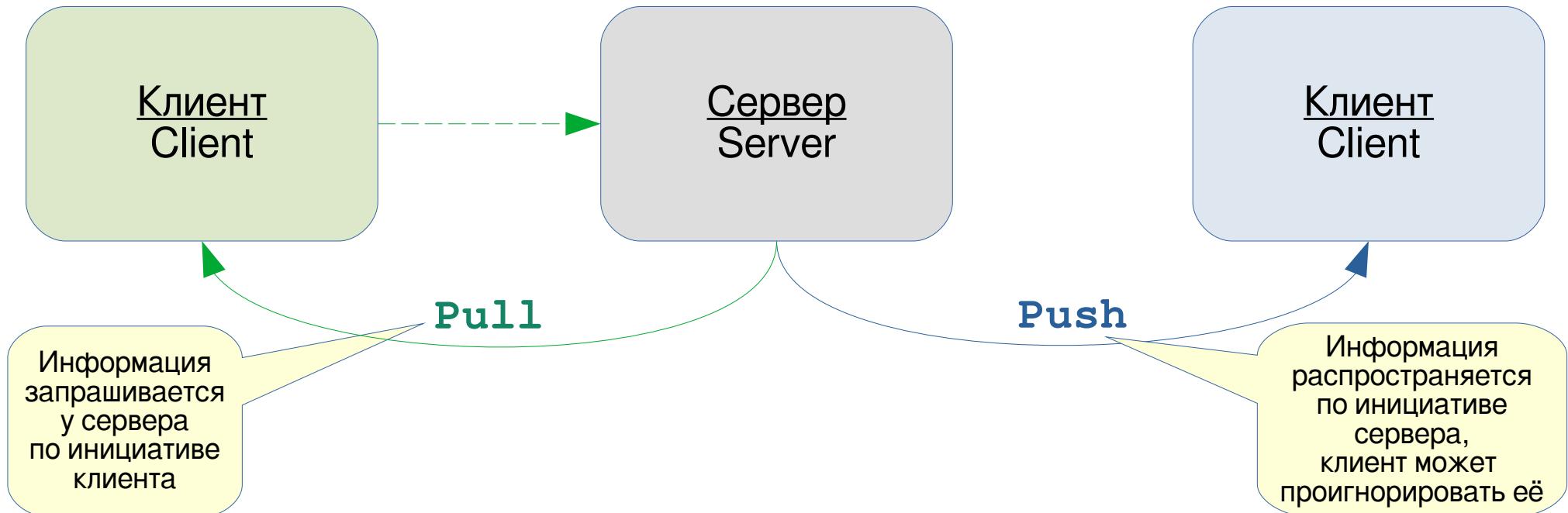


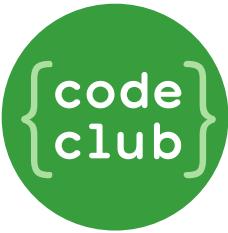
Модели взаимодействия



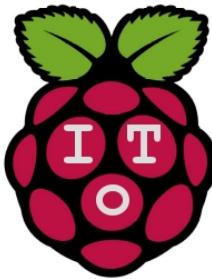
«Запрос — Ответ»

«Вещание» / «Подписка»



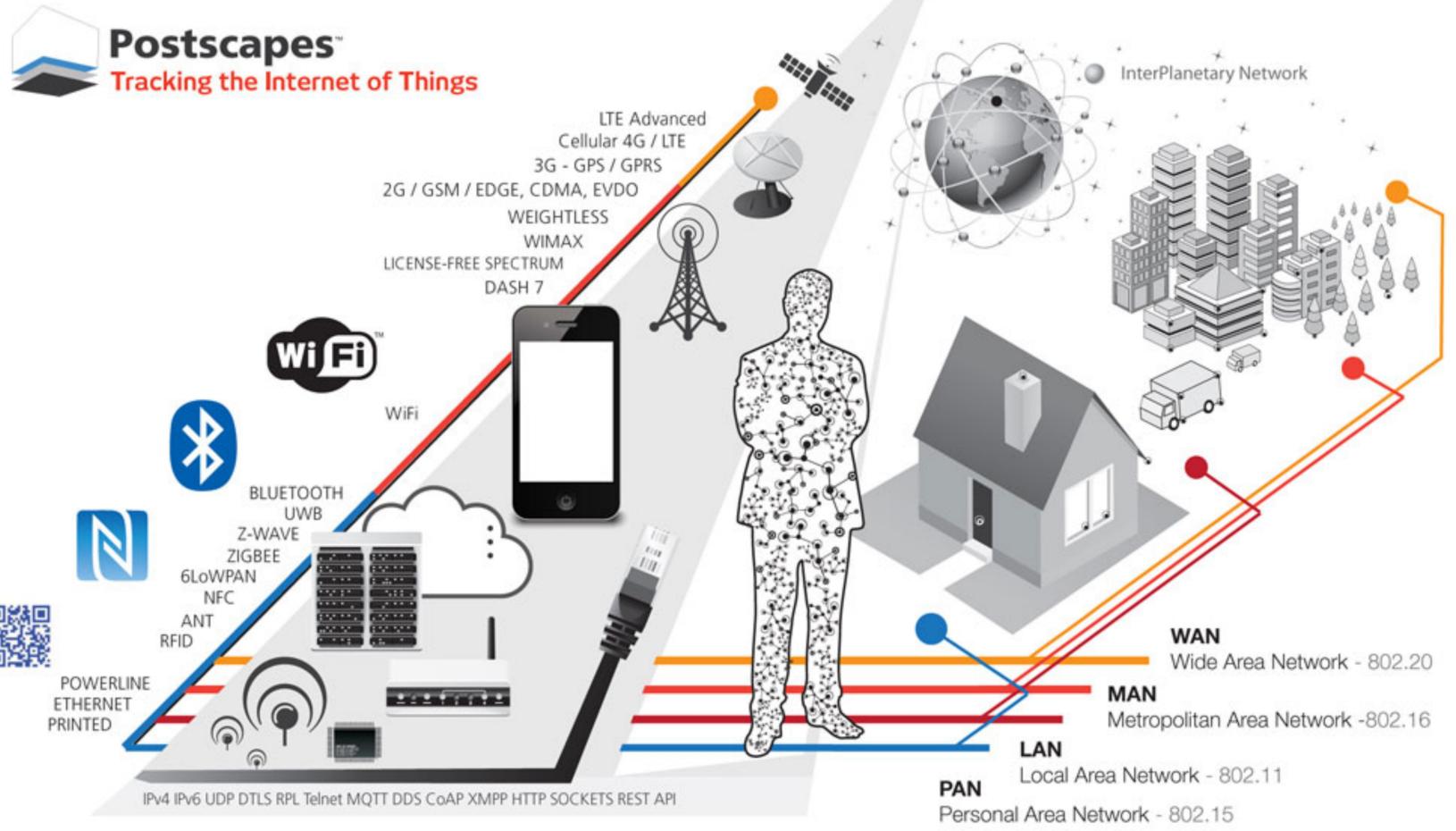
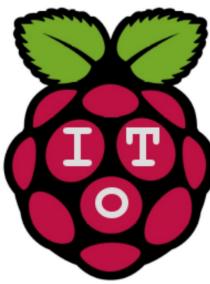


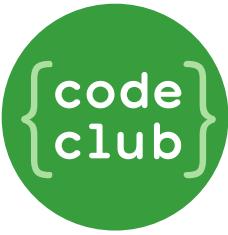
Сети разного охвата



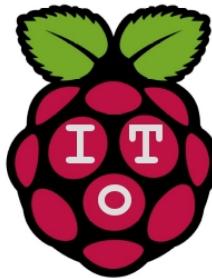
- Interplanetary Internet (**DTN**) ~ межпланетная сеть
- Internet ~ всепланетная сеть
- **WAN** (Wide Area Network) ~ межрегиональная сеть
- **MAN** (Metropolitan Area Network) ~ городская сеть
- **CAN** (Campus Area Network) ~ университетская сеть
- **LAN** (Local Area Network) ~ локальная вычислительная сеть
- **HAN** (Home Area Network) ~ домашняя сеть
- **NAN** (Near-me Area Network) ~ близлежащая сеть
- **CAN** (Car / Electronics Area Network) ~ автомобильная сеть
- **PAN** (Personal Area Network) ~ персональная сеть
- **BAN** (Body Area Network) ~ нательная сеть
- **NFC** (Near Field Communication) ~ связь ближнего действия
- NanoNetwork (IEEE P1906.1) ~ связь между нано-роботами

Масштабы сетей

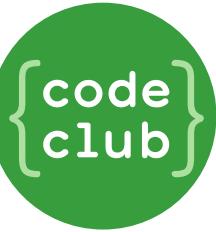




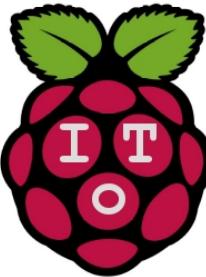
Транспортные протоколы



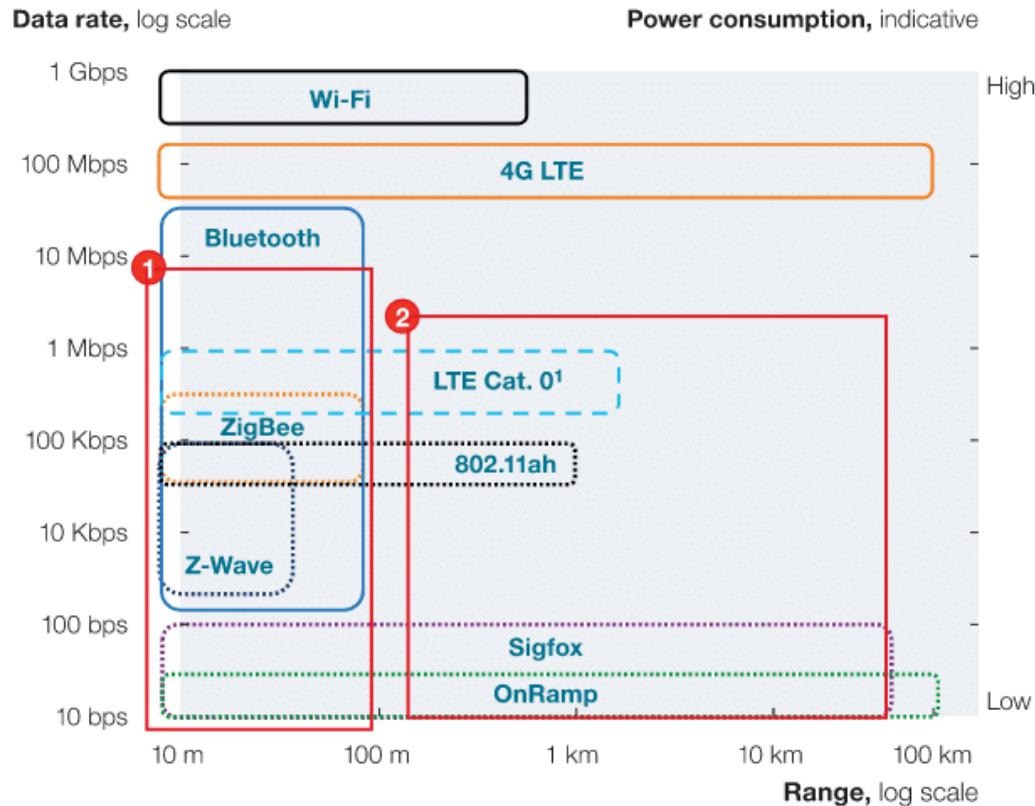
- NFC — бесконтактная связь ближнего действия: < 20 см
- RFID — радиочастотная идентификация: от 20 см до 300 м
- IrDA (Infra-Red Data Association) — настольная связь: 5-50 см
- **UWB (Ultra-Wide Band)** — связь с устройствами: до 3 м
- Bluetooth / BLE, 6LoWPAN — в помещениях (WPAN): до 10 м
- EnOcean — в помещениях: до 30 м, на открытом месте до 300 м
- ZigBee, Z-Wave — беспроводные сенсорные сети (WHAN): до 100 м
- **Wi-Fi** — в здании (WLAN): десятки метров
- Сотовая связь (GPRS/2G/3G/4G) — в населённом пункте (WMAN): км
- WiMax — в нескольких населённых пунктах: десятки км
- **LoRaWAN — энергоэффективная связь дальнего действия (LPWAN)**
- Связь через ЛЭП (PLC = Power Line Communication)
- Спутниковая связь — между странами: тысячи км



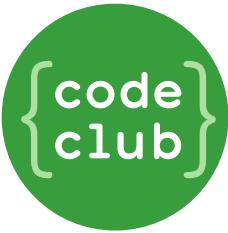
Скорость и охват



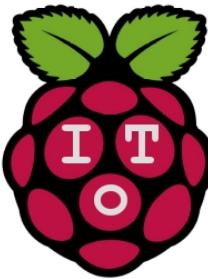
— Widely adopted New standard - - - Established, adoption ongoing



- ①
 - Несколько конкурирующих стандартов на низкоскоростные сети небольшого охвата .
 - Нет совместимости между протоколами.
- ②
 - Широкое поле для стандартизации для низкоскоростных сетей большого охвата с низким энергопотреблением.

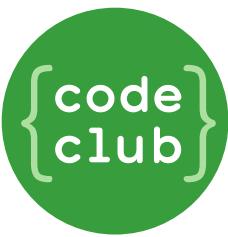


Протоколы обмена

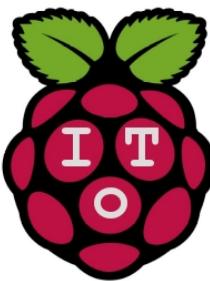


В проектах IoT применяются разные протоколы обмена данными:

- Специализированные:
 - **MQTT** (*Message Queue Telemetry Transport*)
 - **AMQP** (*Advanced Message Queuing Protocol*)
 - **CoAP** (*Constrained Application Protocol*)
 - **STOMP** (*Simple / Streaming Text Oriented Message Protocol*)
 - **DDS** (*Data Distribution Service*)
 - **XMPP-IoT** (*Jabber XMPP for IoT*)
 - **LWM2M** (*Lightweight M2M*)
- Универсальные:
 - **RESTful HTTP**
 - **SOAP** (*Simple Object Access Protocol*)
 - **Matrix**
 - **WMI / WBEM**
 - ...

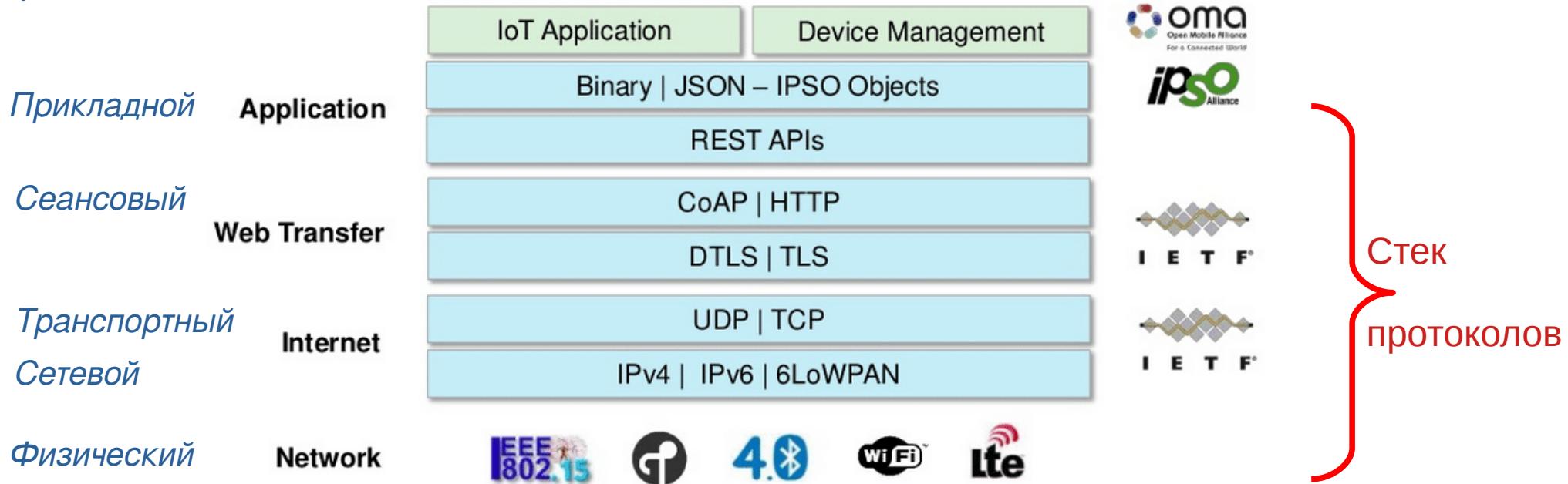


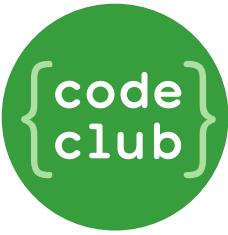
Стек протоколов



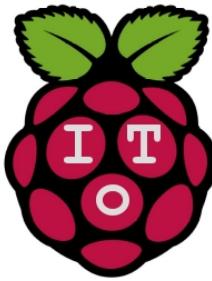
Что значит **И** в **IoT**:

Уровни сетевой модели OSI





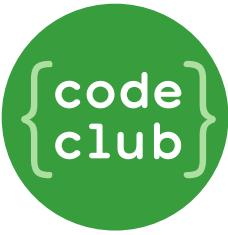
Особенности сетей ИoT



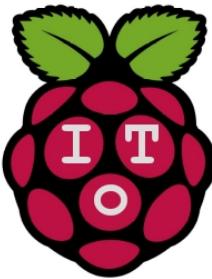
Сложности построения сетей IoT:

- использование различных программно-аппаратных платформ
- применение разных сетевых технологий
- фрагментарность сети (удалённость отдельных сегментов)
- нестабильность связи
- возможная временная недоступность узлов сети
- необходимость интеграции решений от разных производителей

Требуется повышенная надёжность работы, исправление ошибок, дублирование каналов передачи данных, резервирование хостов, самовосстановляемость, защита от злоумышленников.



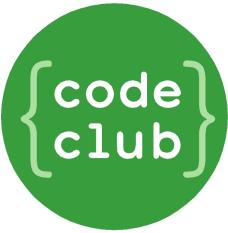
REST



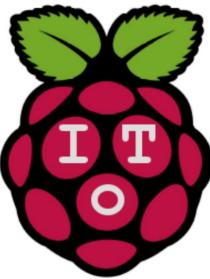
Распределённые системы эффективно работают на основе архитектурного стиля **REST** (Representational State Transfer), который основан на 3-х общеизвестных стандартах (RFC), на которых основана Всемирная Паутина (WWW):

- I. Протокол **HTTP** — для доступа к данным (ресурсам).
- II. Идентификаторы **URI** — для именования ресурсов.
- III. Форматы **MIME** — для представления данных разных типов.

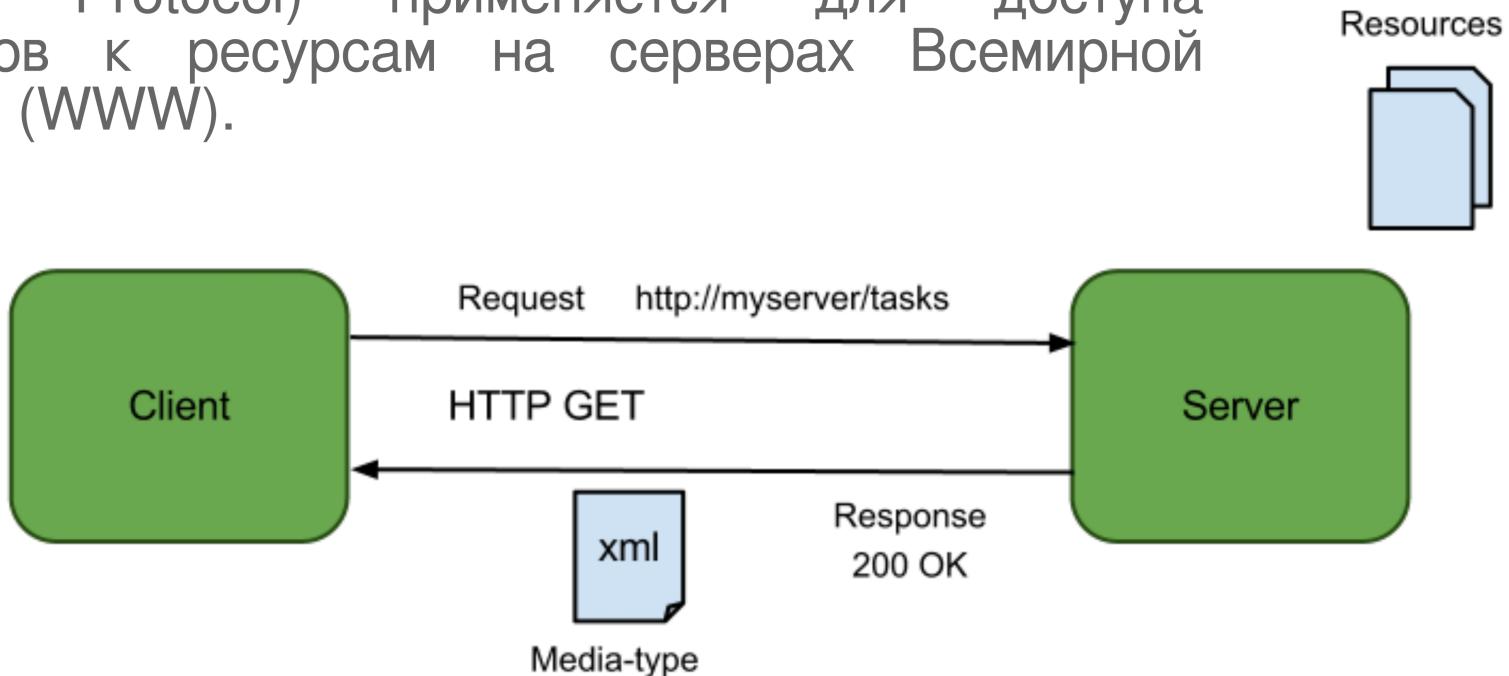
REST используется как программный интерфейс (API) для доступа с клиента к ресурсам (данным, расположенным на сервере) и их изменения (если это позволено).

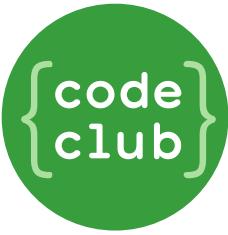


HTTP

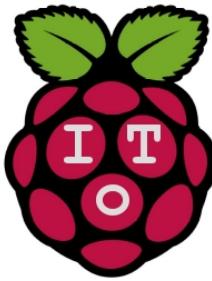


I. Протокол передачи гипертекста **HTTP** (HyperText Transfer Protocol) применяется для доступа браузеров к ресурсам на серверах Всемирной Паутины (WWW).





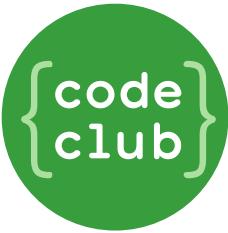
Изменение данных



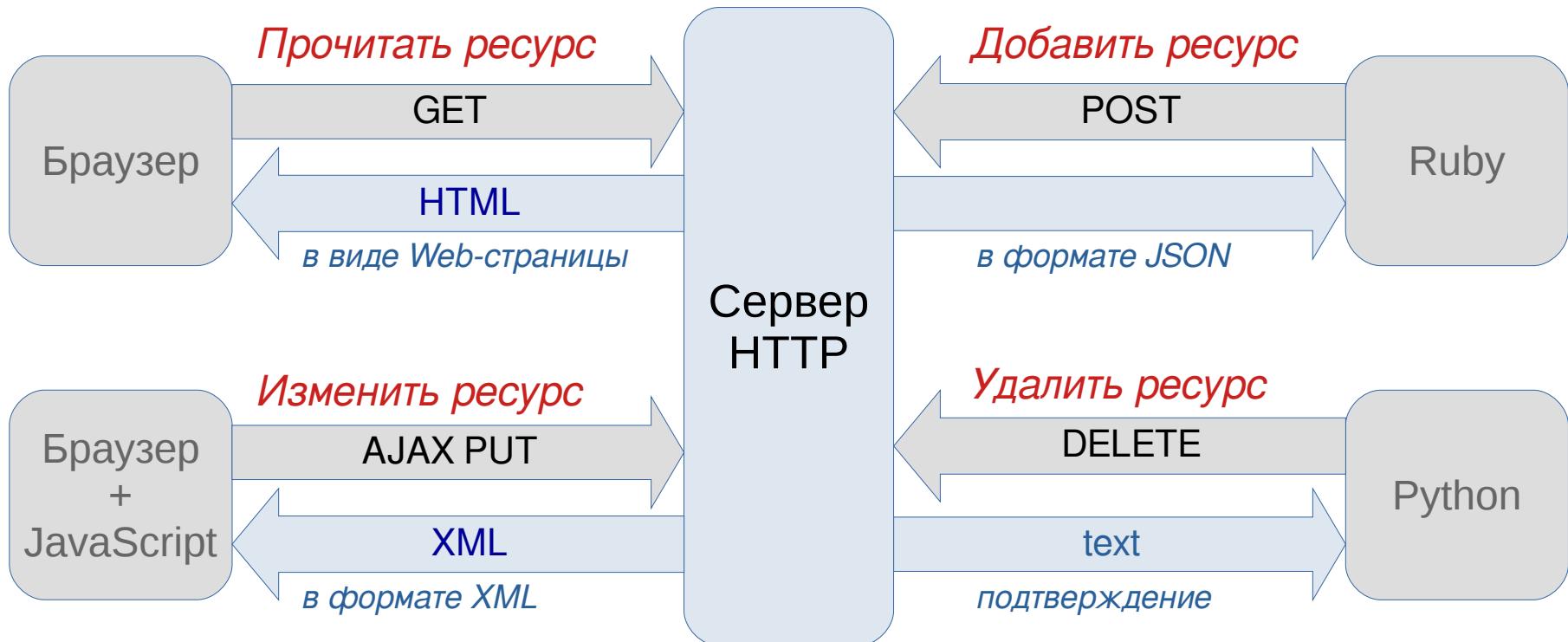
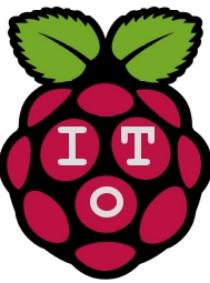
В протоколе **HTTP** предусмотрен стандартный набор операций (POST, GET, PUT, DELETE) для манипулирования данными: добавить, прочитать, изменить, удалить (Create, Read, Update, Delete = CRUD).

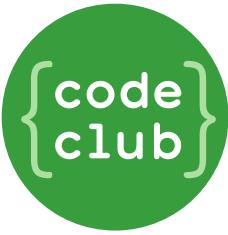
GET /resource/collection	запросить коллекцию ресурсов
POST /resource/collection	добавить единичный ресурс в коллекцию
GET /resource/collection/ <i>id</i>	запросить единичный ресурс
PUT /resource/collection/ <i>id</i>	изменить ресурс (или коллекцию)
DELETE /resource/collection/ <i>id</i>	удалить ресурс (или коллекцию)

Операции протокола **HTTP** соответствуют основным командам **SQL** для изменения информации в базах данных: INSERT, SELECT, UPDATE, DELETE).

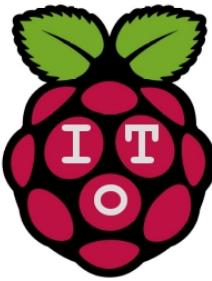


REST





URL



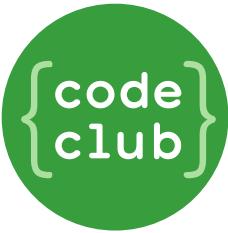
II. Универсальный указатель ресурса **URL** (Uniform Resource Locator) применяется для именования ресурсов Всемирной Паутины (WWW). Он применяется в **REST** для идентификации ресурсов, которые подразделяются на 2 вида:

- единичные ресурсы (объекты) и
- коллекции (списки, наборы, массивы) ресурсов.

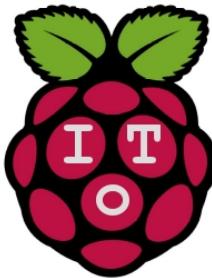
Но доступ к любому из видов производится однотипно:

- server.net/resource/collection — коллекция;
- server.net/resource/collection/filter — часть коллекции;
- server.net/resource/collection/id — ресурс;
- application.info/class/ — список объектов класса;
- application.info/class/id — объект из списка;
- application.info/class/subclass — объекты подкласса;
- localhost/path/to/dir/ — список файлов в каталоге;
- localhost/path/to/dir/file.txt — файл;

URL — один из видов уникального идентификатора ресурса **URI** (Uniform Resource Identifier).

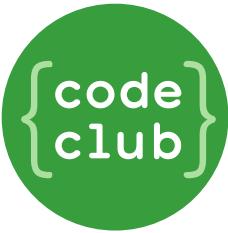


MIME: типы данных

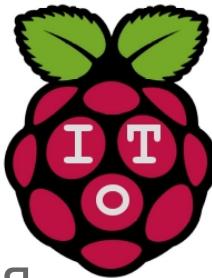


III. Типы данных **MIME** (Multipurpose Internet Mail Extensions) — стандартизованные представления данных (media types), в т. ч. мультимедийных, были разработаны ещё для вложений электронной почты и дополнены новыми типами после появления WWW:

- text/csv - Comma-Separated Values, CSV (RFC 4180);
- text/html - HyperText Markup Language, HTML (RFC 2854);
- text/plain - текстовые данные (RFC 2046, 3676);
- application/json - JavaScript Object Notation, JSON (RFC 4627);
- application/xml - eXtensible Markup Language, XML;
- application/octet-stream - двоичный файл (RFC 2046);
- image/png - Portable Network Graphics, PNG (RFC 2083);
- audio/mpeg - MPEG-аудио, включая MP3 (RFC 3003);
- video/mpeg - MPEG-1 (RFC 2045, 2046);



Форматы данных



При передаче данных по сети используются разные представления данных:

- Текст:
 - JSON, XML, Key=Value, CSV, YAML, ...
- Двоичные данные.

JSON

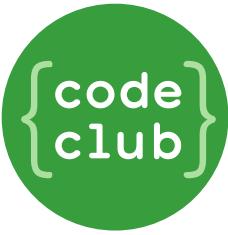
```
{"System": {  
    "Name": "ESP_Easy",  
    "Unit": 4},  
  "Sensors":  
  [{"TaskName": "T_P_H",  
   "Temperature": 26.21,  
   "Humidity": 19.48,  
   "Pressure": 742.01},  
   {"TaskName": "Screen"},  
  ]}
```

XML

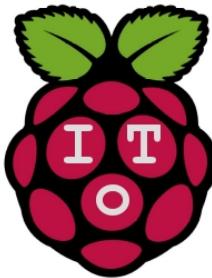
```
<Data>  
  <System Name="ESP_Easy" Unit=4 />  
  <Sensors>  
    <Task Name="T_P_H">  
      <Temperature>26.21</Temperature>  
      <Humidity>19.48</Humidity>  
      <Pressure>742.01</Pressure>  
    </Task>  
    <Task Name="Screen" />  
  </Sensors>  
</Data>
```

Key=Value

```
Name="ESP_Easy"  
Unit=4  
Sensor="T_P_H"  
Temperature=26.21  
Humidity=19.48  
Pressure=742.01
```



MIME : представления



1. Клиент может запросить у HTTP-сервера требуемое представление ресурса (media type) одним из способов:

- как часть URI (выглядит, как суффикс файла):

resource.HTML

resource.XML

resource.JSON

resource.PDF

- в заголовке Accept запроса HTTP:

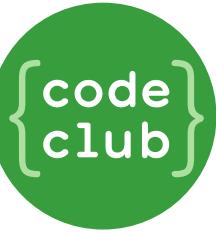
Accept: text/html

Accept: application/xml

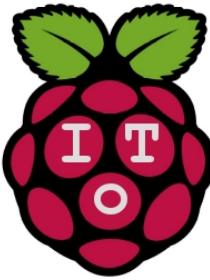
Accept: application/json

Accept: application/pdf

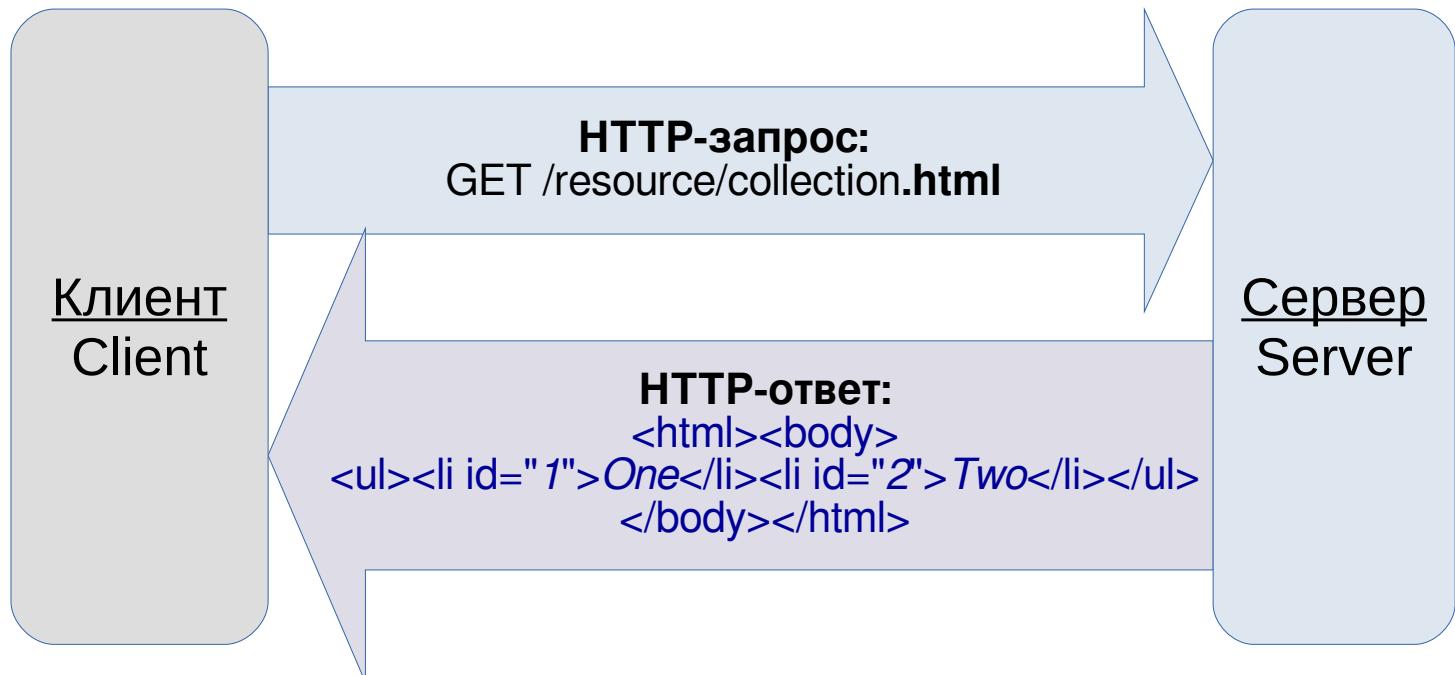
2. Сервер сообщает о представлении передаваемого ресурса в заголовке Content-Type ответа HTTP.

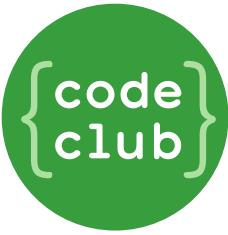


REST: доступ к данным

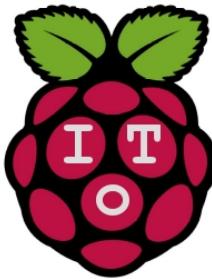


- Ресурс уникально **идентифицируется** по URI.
- **REST-запрос:** глагол HTTP (GET / PUT / POST / DELETE) для выполнения действия.
- **Данные** для обновления: в параметрах запроса.
- Желаемое **представление:** в параметрах запроса.
- **Ресурс** в нужном представлении — в HTTP-ответе.



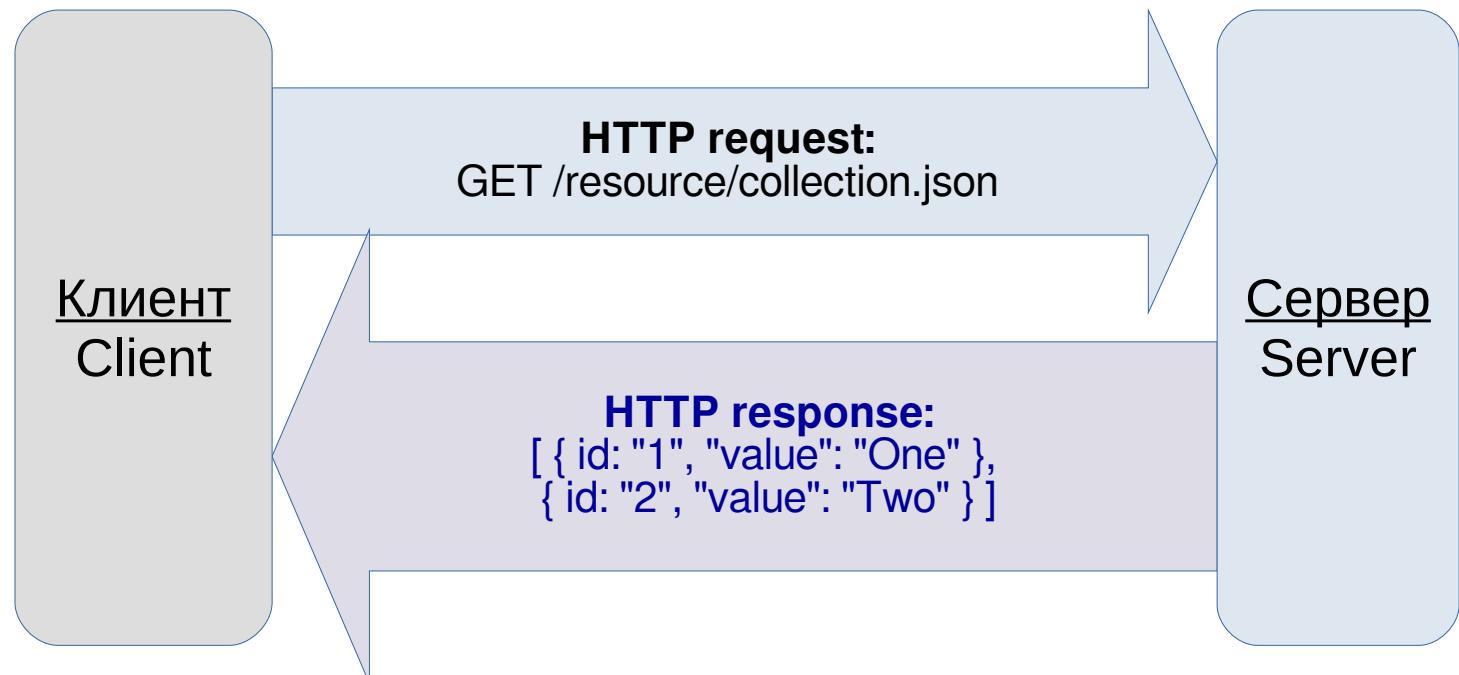


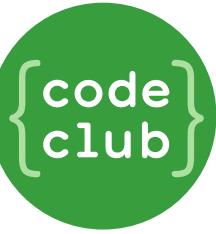
REST: представление данных



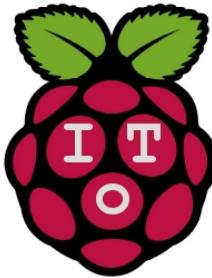
Клиент может
запросить тот же самый
ресурс в другом
представлении (JSON).

Сервер может
преобразовать ресурс
из неизвестной для
клиента формы
хранения (реляционная
БД) в запрошенное
представление.





REST: примеры запросов



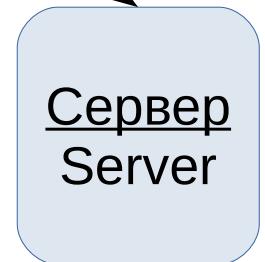
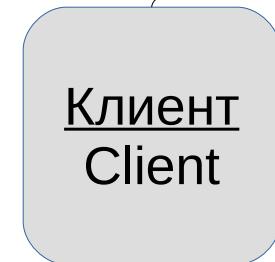
GET /meteostation/values/json



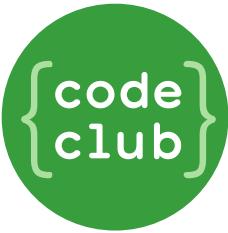
[{ "sensor": "Температура",
"value": 25.5, "unit": "Celsius" },
 { "sensor": "Влажность",
"value": 48.0, "unit": "procent" },
 { "sensor": "Давление",
"value": 749.3, "unit": "mm Hg" }]



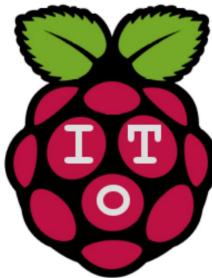
GET /meteostation/values/t.json



{"sensor": "Температура", "value": 25.5, "type": "Celsius"}



Ruby: клиент HTTP



Класс `URI` применяется для манипуляциями с адресом, а класс `Net::HTTP` — для создания запросов, отправки их на сервер и получения ответов.

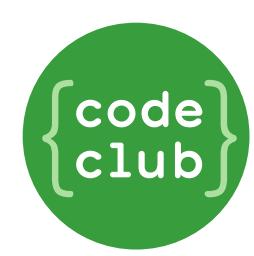
```
require "net/https"
require "json"

host = "192.168.43.141"                                # имя или адрес узла

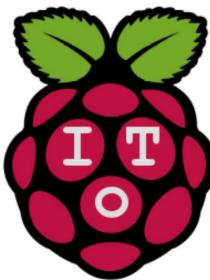
uri = URI.parse("http://#{host}/json")      # URL ресурса
http = Net::HTTP.new(uri.host, uri.port) # новый клиент

get_request = Net::HTTP::Get.new(uri.request_uri) # запрос по GET
response = http.request(get_request)           # получить ответ на запрос
text = response.body                            # извлечь тело ответа

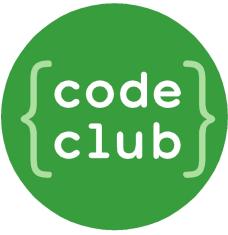
hash1 = JSON.parse(text)                         # JSON → hash
p hash1["Sensors"][0]["Pressure"]               # значение по ключу
```



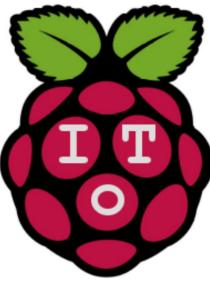
Ruby : сервер HTTP



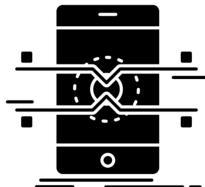
С помощью класса **TCPServer** создаётся объект «сервер протокола TCP», который слушает заданный порт, принимает запросы, отправляет ответ.



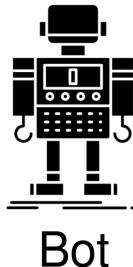
[matrix]



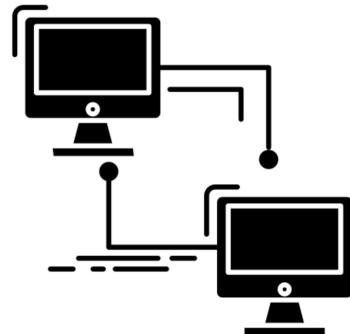
Применение протокола Matrix позволяет создавать децентрализованные системы IoT с защищённой передачей данных по сети. Matrix хорошо подходит для объединения разного программного обеспечения за счёт создания межсетевых мостов, а также публиковать или потреблять данные, работая напрямую с устройствами сбора данных. Программный интерфейс (API) протокола Matrix основан на REST и JSON.



Client



Bot

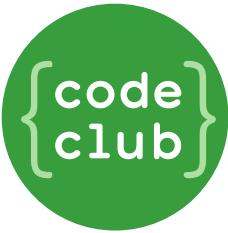


Bridge

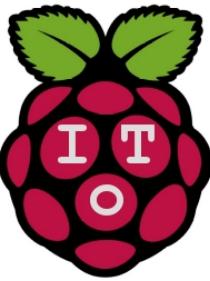


SDK

C, C++, C#,
Objective-C,
Java, Kotlin,
JavaScript,
Ruby, Perl,
Go, Lua,
Rust, Lisp,
Swift, Elixir,
Python,
PHP,
shell/bash



Что?



По-вашему, что важнее в IoT:

1. Идентификация
2. Сбор данных с датчиков
3. Связь по сети
4. Хранение данных
5. Обработка данных
6. Выполнение действий