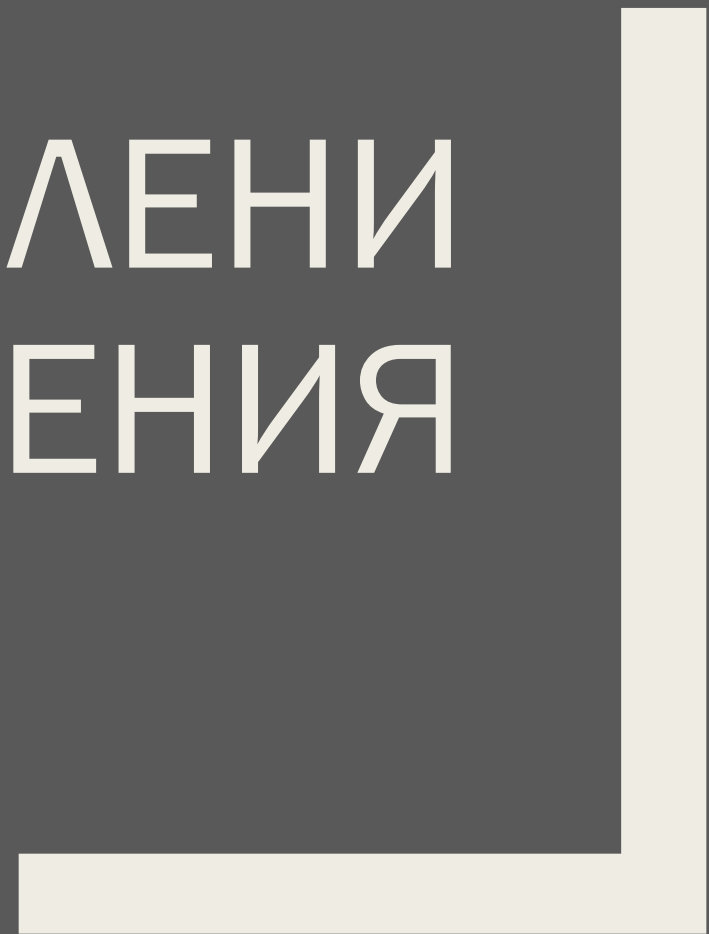


РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ



История на приложенията

- Първо поколение - Централизираните изчисления (Centralized computing).
- Второ поколение - Разпределени приложения (Distributed applications).

Централизираните изчисления (Centralized computing).

- Централизираните изчисления обединяват и контролират всеки аспект от приложението включително бизнес процесите, дата мениджмънта както и графичната визуализация.

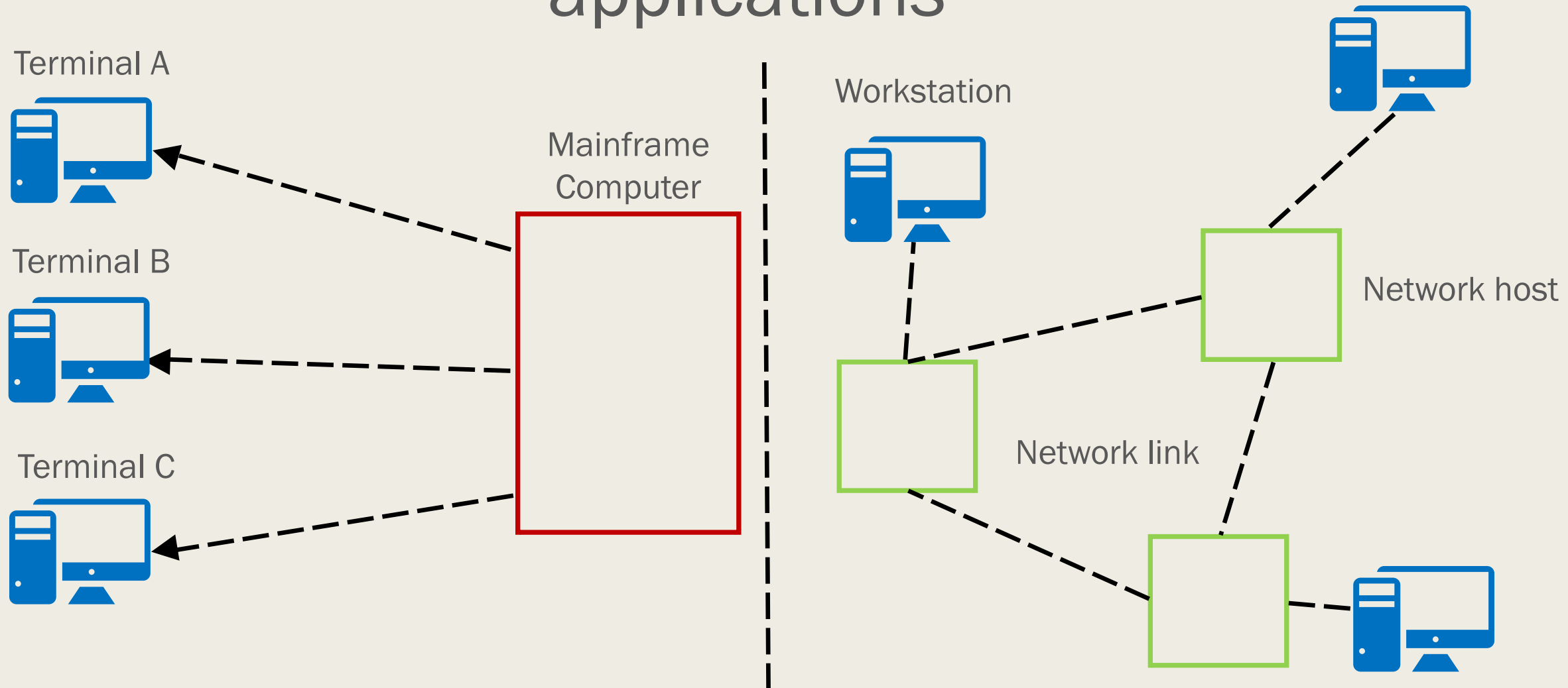
Проблеми на Централизираните изчисления

- Цялата тежест на процеса, включително достъпа до данните, бизнес логиката и презентационната логика са представени в едно приложение на една физическа машина
- Големите и комплексни приложения са трудни за поддръжка и развитие
- Тяхната откъсната и комплексна логика прави интеграцията им с други приложения и платформи изключително трудна

Разпределени приложения

- Разпределено приложение е приложение физически разделено на отделни компоненти, обединението, на които описва пълната функционалност на приложението.

Centralized computing vs Distributed applications



Разпределено програмиране

- Разпределеното програмиране се характеризира от няколко физически компонента работещи в синхрон като една обединена система.

Физически компоненти?

- Могат да бъдат както ПРОЦЕСОРНИ ЯДРА така и КОМПЮТРИ НАМИРАЩИ СЕ В ЕДНА МРЕЖА.

Основата на разпределеното програмиране е оптимизацията.

- Ако даден компютър може да приключи определена задача за 5 секунди, то 5 такива машини трябва да приключат същата задача за 1 секунда.

Проблема се състои в това да се съчетаят отделните модули и системи да работят паралелно.

Пример



Основни принципи на разпределените приложения

- В книгата си „Distributed .NET Programing in C#“ Том Барнаби описва пет основни принципа на разпределените приложения.

5-те принципа на разпределените приложения

Distribute Sparingly

Localize Related Concerns

Use Chunky
Instead of Chatty
Interfaces

Prefer Stateless Over
Stateful Objects

Program to an
Interface, Not an
Implementation

Умерено разпределение (Distribute Sparingly)

- Този принцип се основава на базови факти свързани с изчислителните машини.
Извикването на метод от инстанциран обект в различен процес е сто пъти по – бавно от извикването на метод от самият процес.
Преместването на обект от една машина на друга в една и съща мрежа и извикването на метод от него може да забави процеса на изпълнение до 10 пъти.

Кога трябва да разпределяме?

Отговор

- Само когато се налага.
- Най – често започваме да разделяме една система от базата от данни. Доста често базата от данни се намира на отделна машина, наречена сървър, с други думи е разпределена релативно в отделен слой.

Причини

- Базата от данни е сложен, скъп и обикновено доста тежък софтуер, нуждаещ се от силен хардуер.
- Базата от данни може да съдържа релационни данни и да е споделена между много приложения. Това е възможно само ако всички приложенията имат достъп до базата данни.
- Базите от данни са създадени да работят като отдалечени физически слоеве.

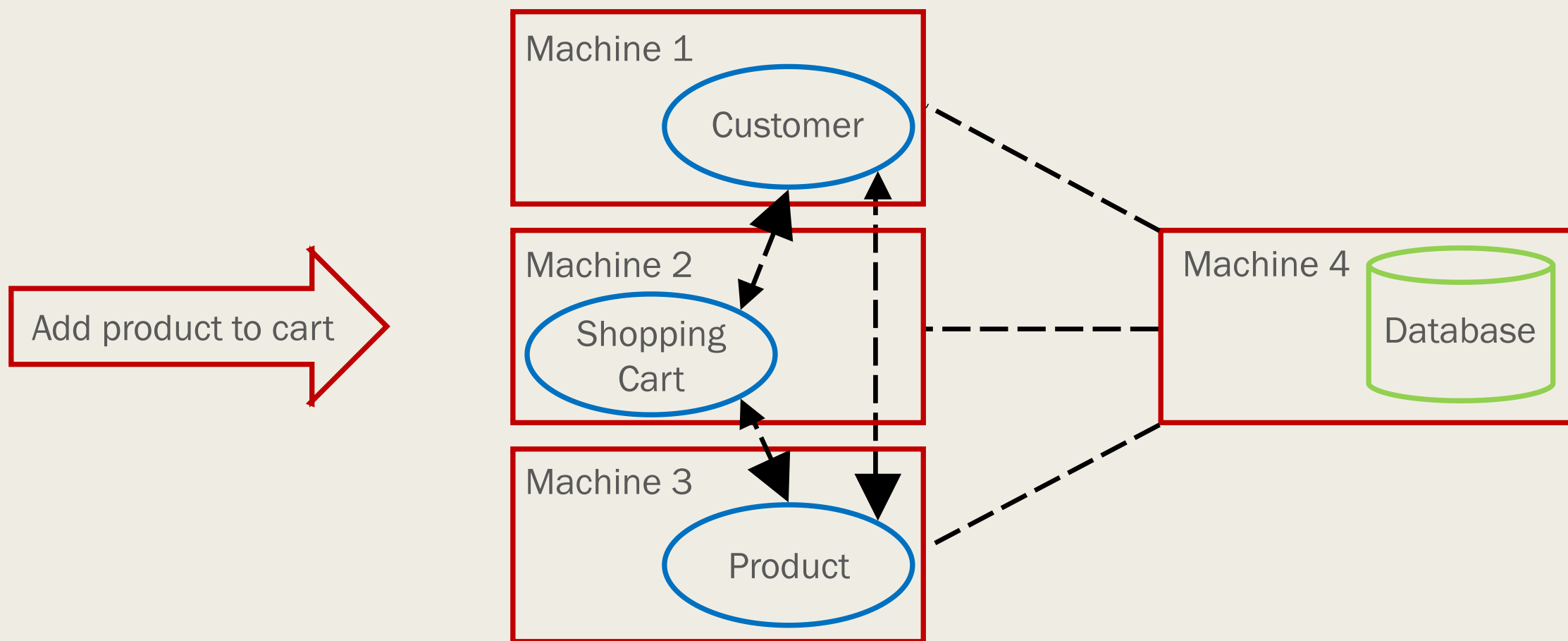
Пример

- Ако разгледаме като пример презентационната логика, тук нещата изглеждат малко по - сложни. Най – важното нещо е да знаем пълната спецификация на приложението, ако на потребителите им се налага да достъпват отдалечен сървър или терминал(като на пример АТМ на някоя банка) може би е добра идея част от логиката да бъде изнесена там.

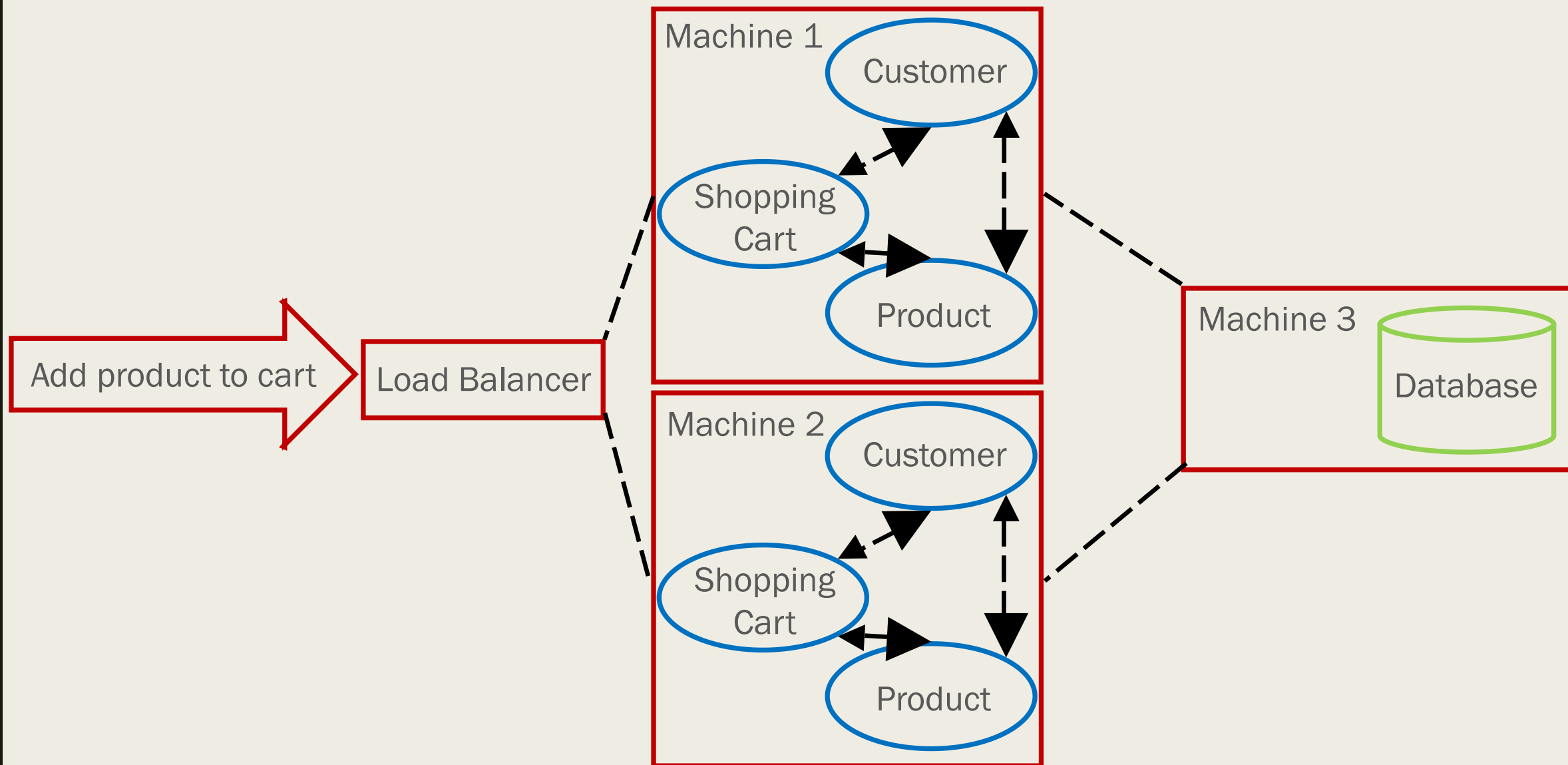
Локализиране на свързващите отношения(Localize Related Concerns)

- Ако бизнес логиката на едно приложение бъде разделена на отделни машини, то много внимателно трябва да се подсигури, комуникацията между най – често общуващите компоненти. С други думи трябва да се локализират свързаните модули.

Паралелно изпълнение на отделни процеси от бизнес логика



Разпределение на натоварването



Малък интерфейс вместо комуникативен(Use Chunky Instead of Chatty Interfaces)

- В основата на този принцип е заложена идеята функционалността да е максимално близо до потребителя, който ще я използва.

Пример

- Комуникация между два сървъра намиращи се на големи разстояния един от друг.
- На сървър 1 се намира имплементация на обекта Customer, а сървър 2 се опитва да ги достъпи.


```
class Customer
{
    public string FirstName() { get; set; }
    public string LastName() { get; set; }
    public string Email() { get; set; }
    // etc. for Street, State, City, Zip, Phone ...

    public void Create();
    public void Save();
}
```

Chatty interface.

```
class Customer
{
    public void Create(string FirstName, string LastName, string
Email, /* etc for Street, State, City, Zip, Phone ... */);

    public void Save(string FirstName, string LastName, string
Email, /* etc for Street, State, City, Zip, Phone ... */);
}
```

Chunky interfaces

```
[Serializable]
class CustomerData
{
    public string FirstName() { get; set; }
    public string LastName() { get; set; }
    public string Email() { get; set; }
    // etc for Street, State, City, Zip, Phone ...
}
class Customer
{
    public void Create(CustomerData data);
    public void Save(CustomerData data);
}
```

Оптимизация на Chunky interfaces

Prefer Stateless Over Stateful Objects

- Препоръчителен и не задължителен.
„Stateless“ обектите са за предпочитане пред „Stateful“ обектите.

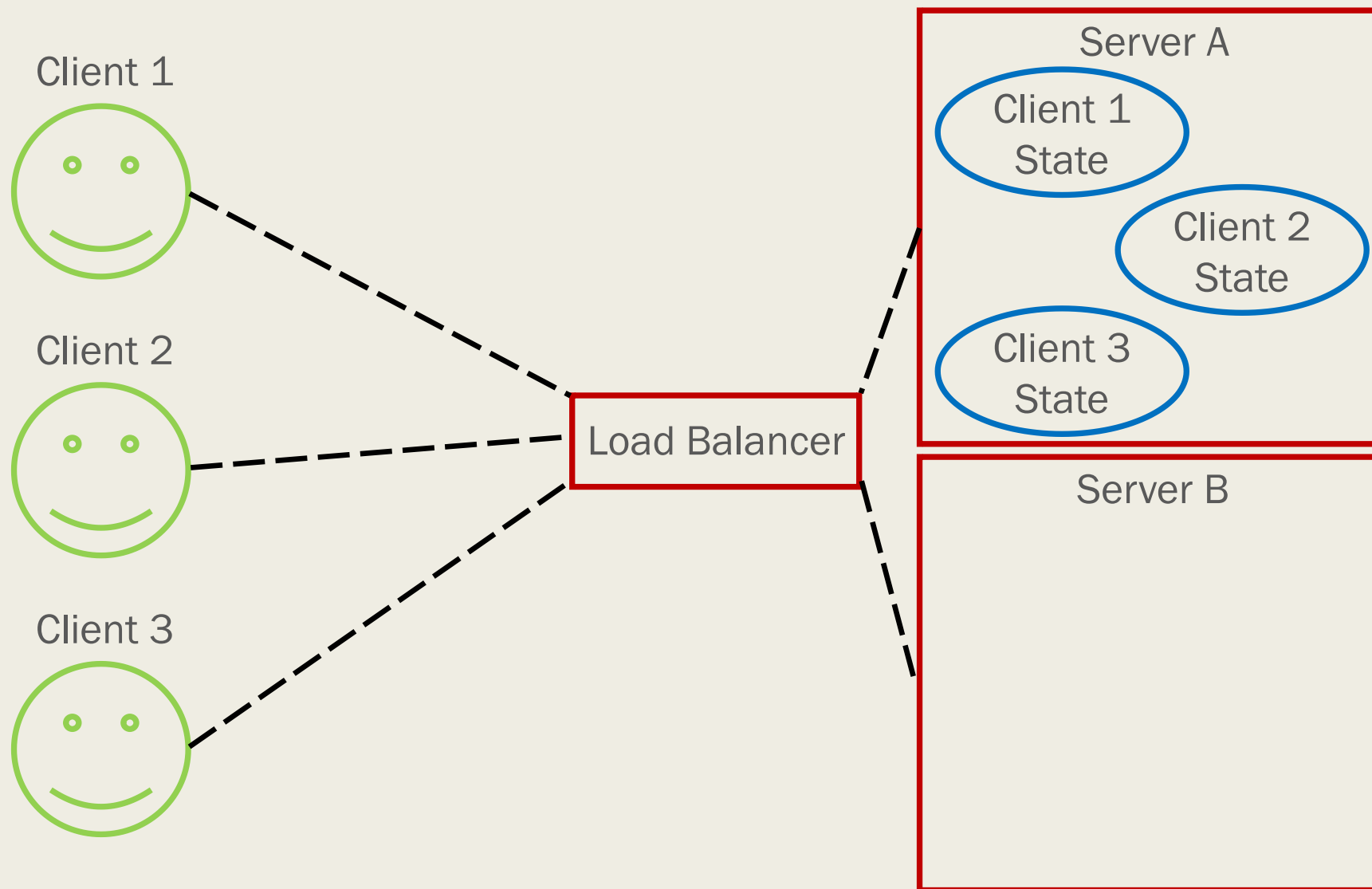
Какво е “Stateless”?

- Това са обекти, които могат безопасно да бъдат създавани и унищожавани между различни методи и процедури. Не е задължително даден метод да унищожи инстанцията на обект след като бъде изпълнен за да бъде „stateless“. Но ако това се случи, премахването на обекта не трябва да окаже промяна в поведението на приложението.

Проблеми на “Stateful” обектите

- Тези обекти съществуват на сървъра за удължен период от време. В този период от време може да се буферира голямо количество системни ресурси. Биха могли дори да предотвратят заделянето на ресурси за други обекти.
- Другата негативна черта е намаляването на ефективността на duplicating и load balancing приложенията разделени между много сървъри

Схема за идентификация на



Program to an Interface, Not an Implementation

- В основата на разпределеното програмиране стои интерфейс базираното програмиране. Проблема, който се решава с този подход е повече свързан с по - лесното прехвърляне на проект върху отдалечен сървър, а не с неговата бързина или разширяемост.

ВЪПРОСИ ?





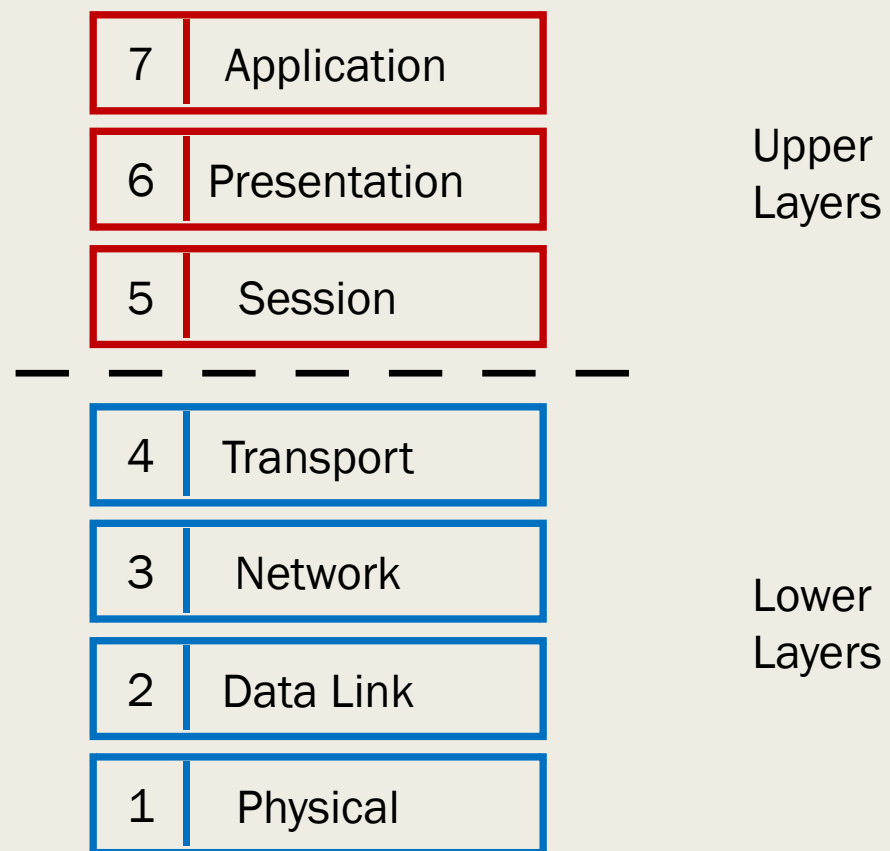
РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

МРЕЖАТА



OSI model



Приложният слой

- Приложният слой е най-горният слой от OSI модела, който се отнася за приложения (програми) като Интернет браузъри, мениджъри за отдалечено управление, клиенти за обмен на съобщения.

Представителен слой

- Представителният слой се грижи за представяне на данните във вид, разбираем за получателя, като осигурява общия им формат за различни платформи.

Сесиен слой

- Сесиен слой управлява създаването (и съответно прекъсването) на сесиите (диалога) между представителните слоеве на две (или повече) системи.

Транспортен слой

- Транспортен слой осигурява комуникация от край до край (end-to-end) между процеси, изпълнявани на различни сървъри. TCP, UDP

Мрежов слой

- Мрежов слой има за основна цел да задава логически адреси на източника и местоназначението, както и да определя най-добрия път за маршрутизиране на данните.

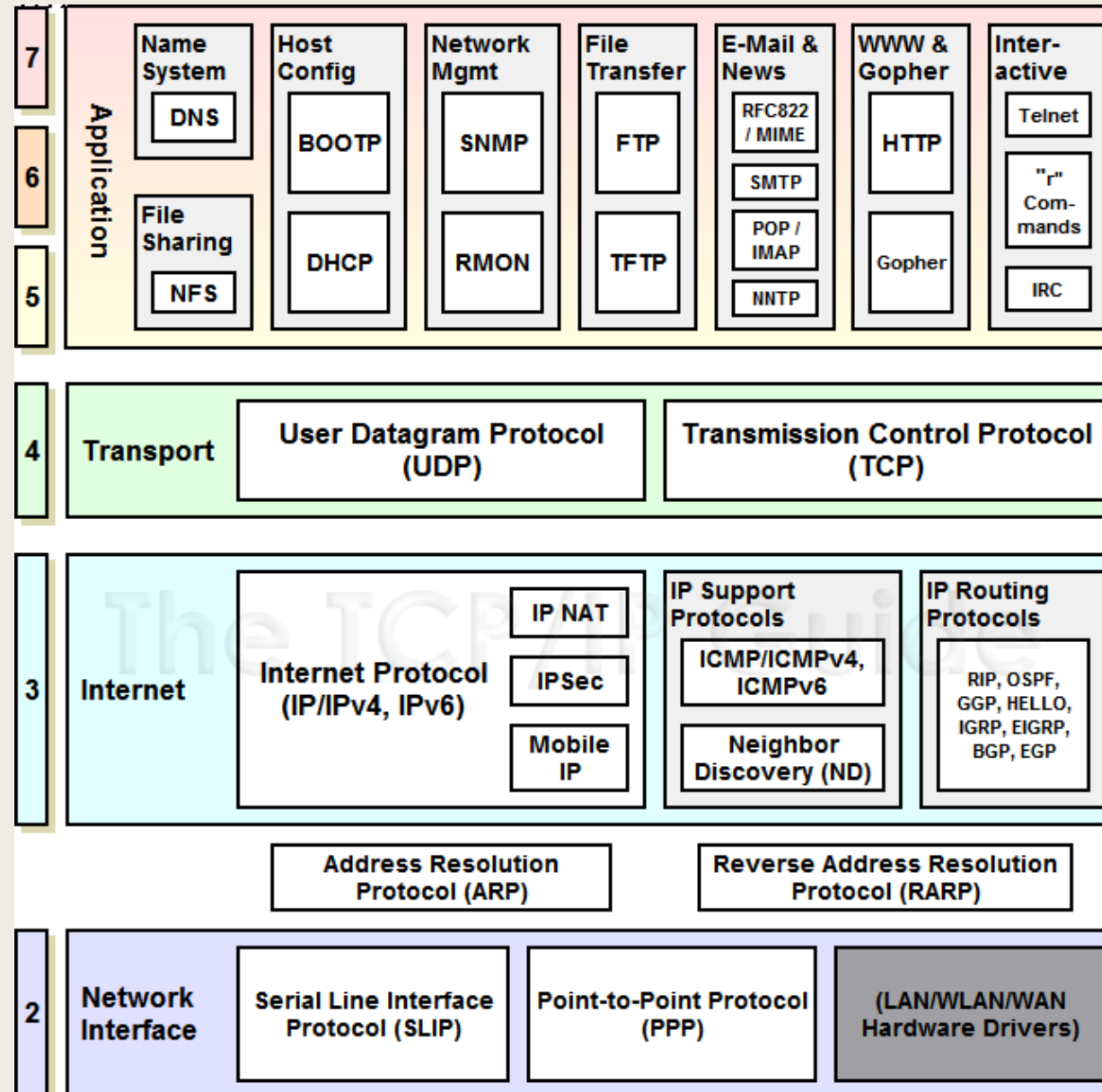
Канален слой

- Канален слой има за цел да предава и да приема кадри, а също така отговаря за тяхното физическо адресиране. Каналният слой се разделя на два подслоя, LLC и MAC, като първият добавя още контролна информация, служеща за правилното транспортиране на данните, а вторият осигурява достъп до преносната среда (медията).

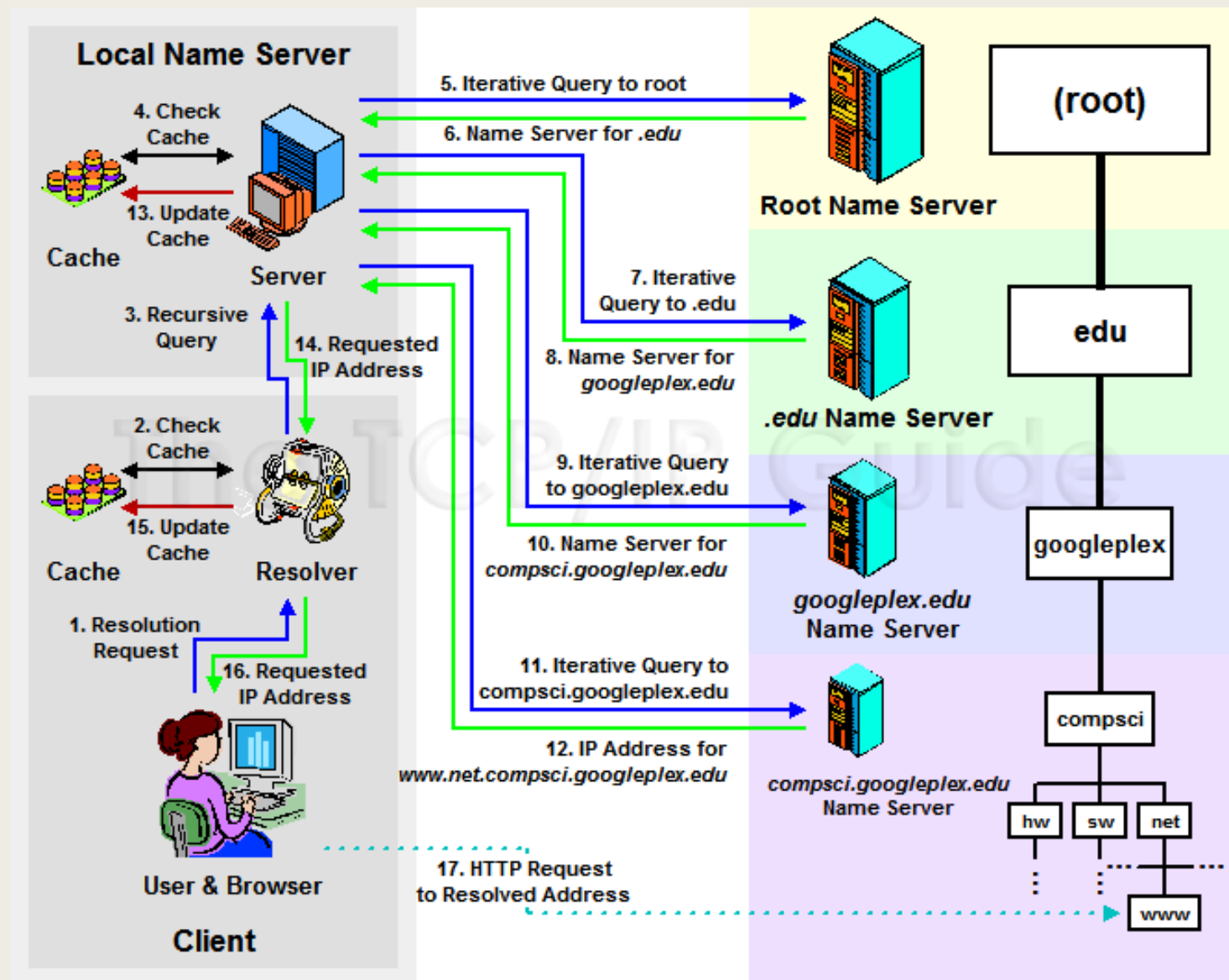
Физически слой

- Физически слой е най-долният слой от модела и работи само с единици и нули (битове), изграждащи кадъра.

TCP/IP / UDP протоколи в OSI модела

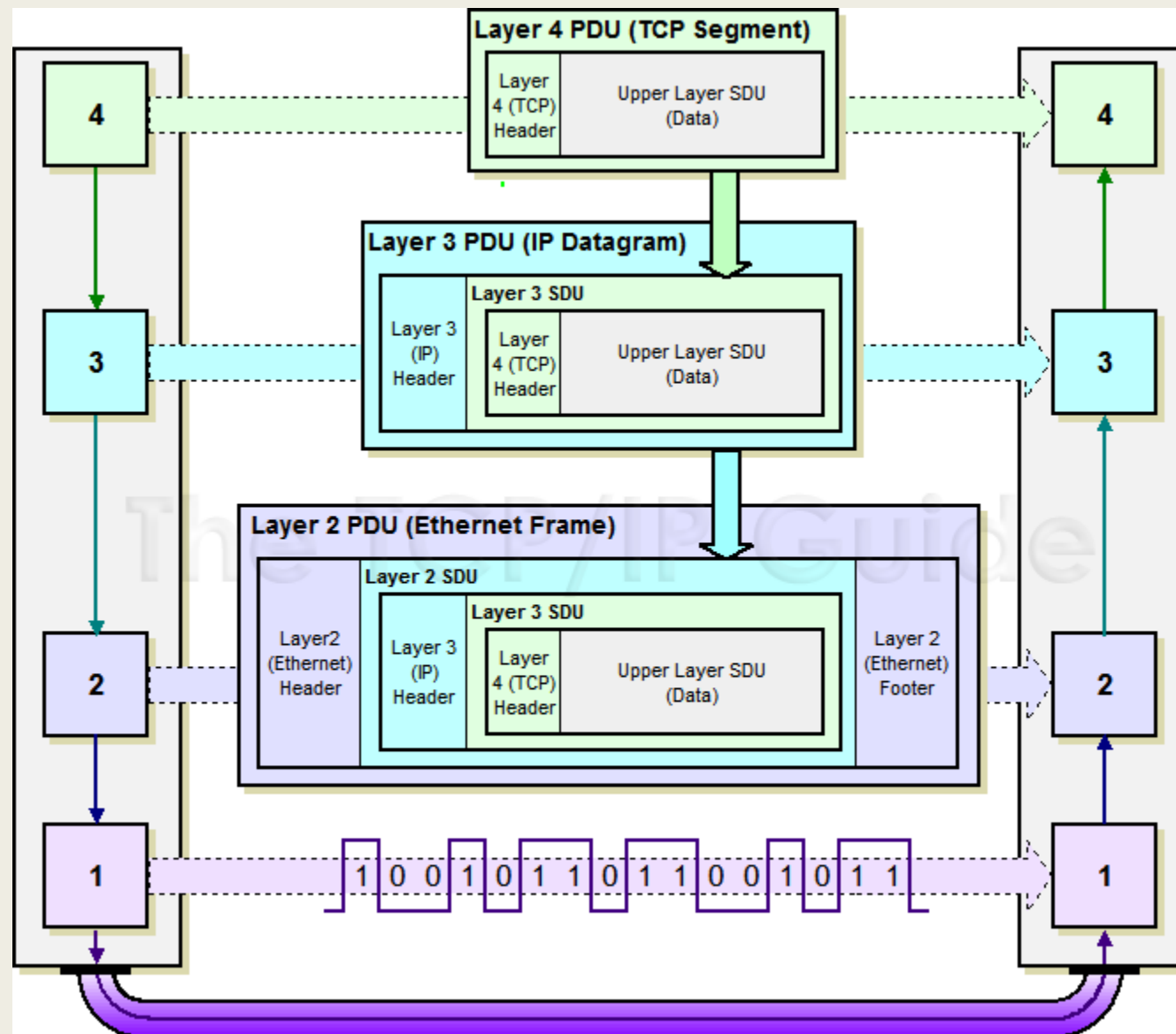


DNS и процеса на запитване за адрес



HTTP / HTTPS

- Протокол за пренос на хипертекст (на английски: Hypertext Transfer Protocol, HTTP) е мрежов протокол, от приложния слой на OSI модела, за пренос на информация в компютърни мрежи.
- Стандартизирана комуникация (<https://datatracker.ietf.org/wg/httpbis/charter>)



HTTP / HTTPS

HTTP

- Слуша на порт 80
- Некриптирани данни

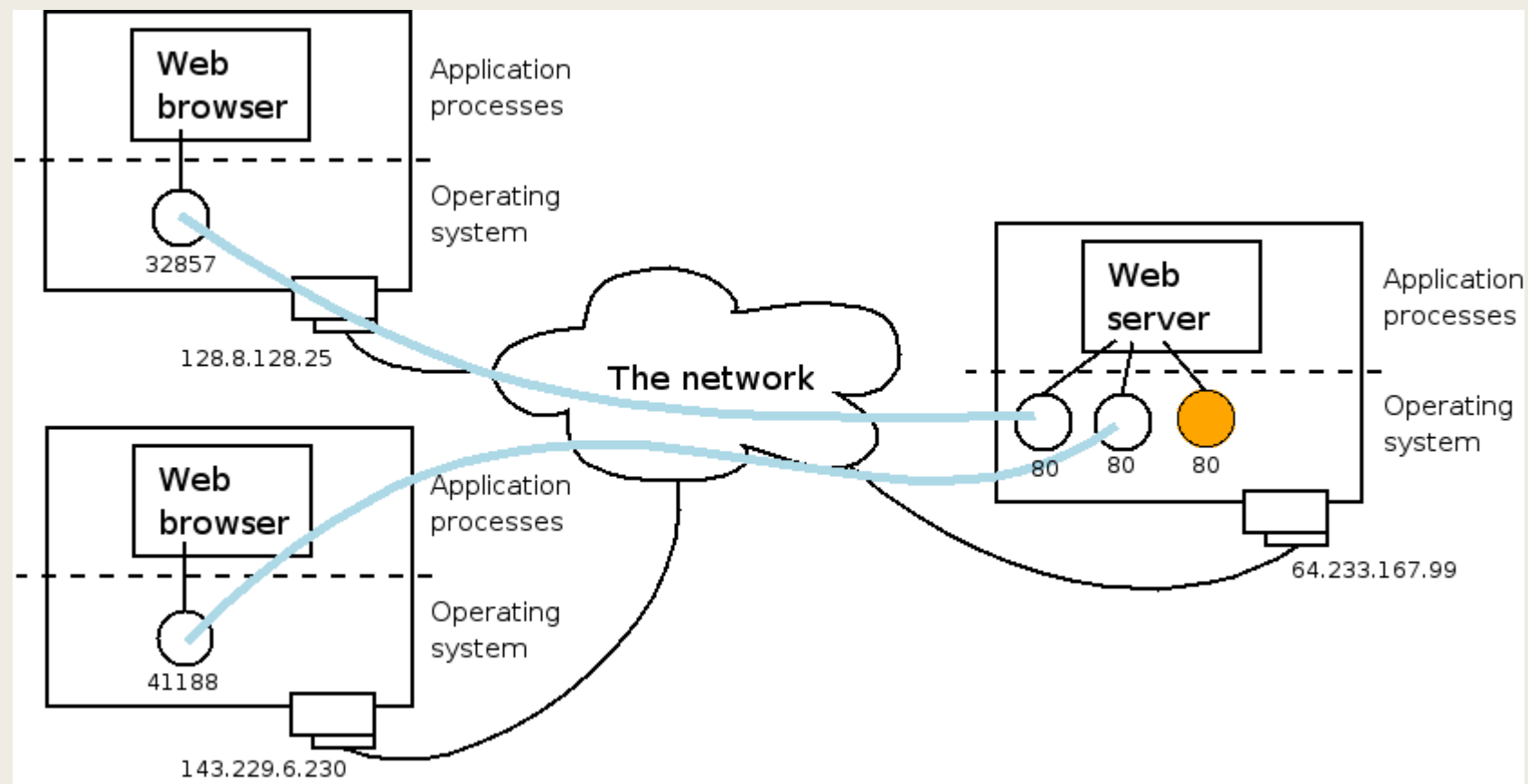
HTTPS

- Слуша на порт 443
- Ползва TLS, SSL или друг протокол за криптиране

Сокети (sockets)

- Софтуерна абстракция за представяне на двата края (терминала) за връзка между машините
- Позволява няколко приложения на една машина да споделят един и същи IP адрес
- Listening (слушащ) сокет – двойката [Destination IP, Destination Порт], представляваща отворен край (терминал) на връзка, към който клиенти могат да се свържат.

Сокети (sockets)



HTTP Message

Request Message

- Method
- Uri
- Version
- Headers
- (Message Body)

Response Message

- Version
- Status Code
- Reason Phrase
- Headers
- (Message Body)

HTTP Headers

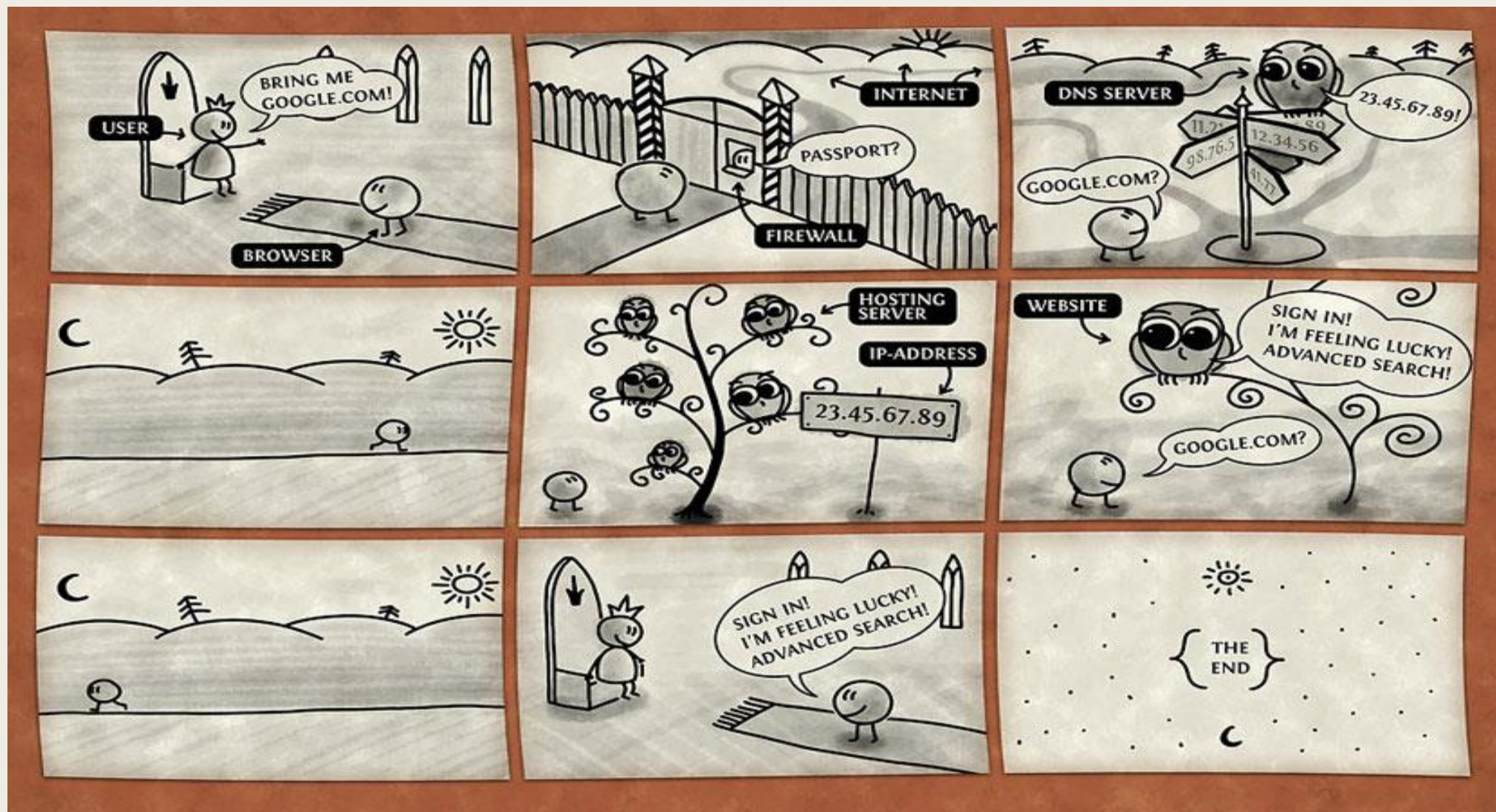
Request Fields

- Host
- User-Agent
- Cookie
- ...

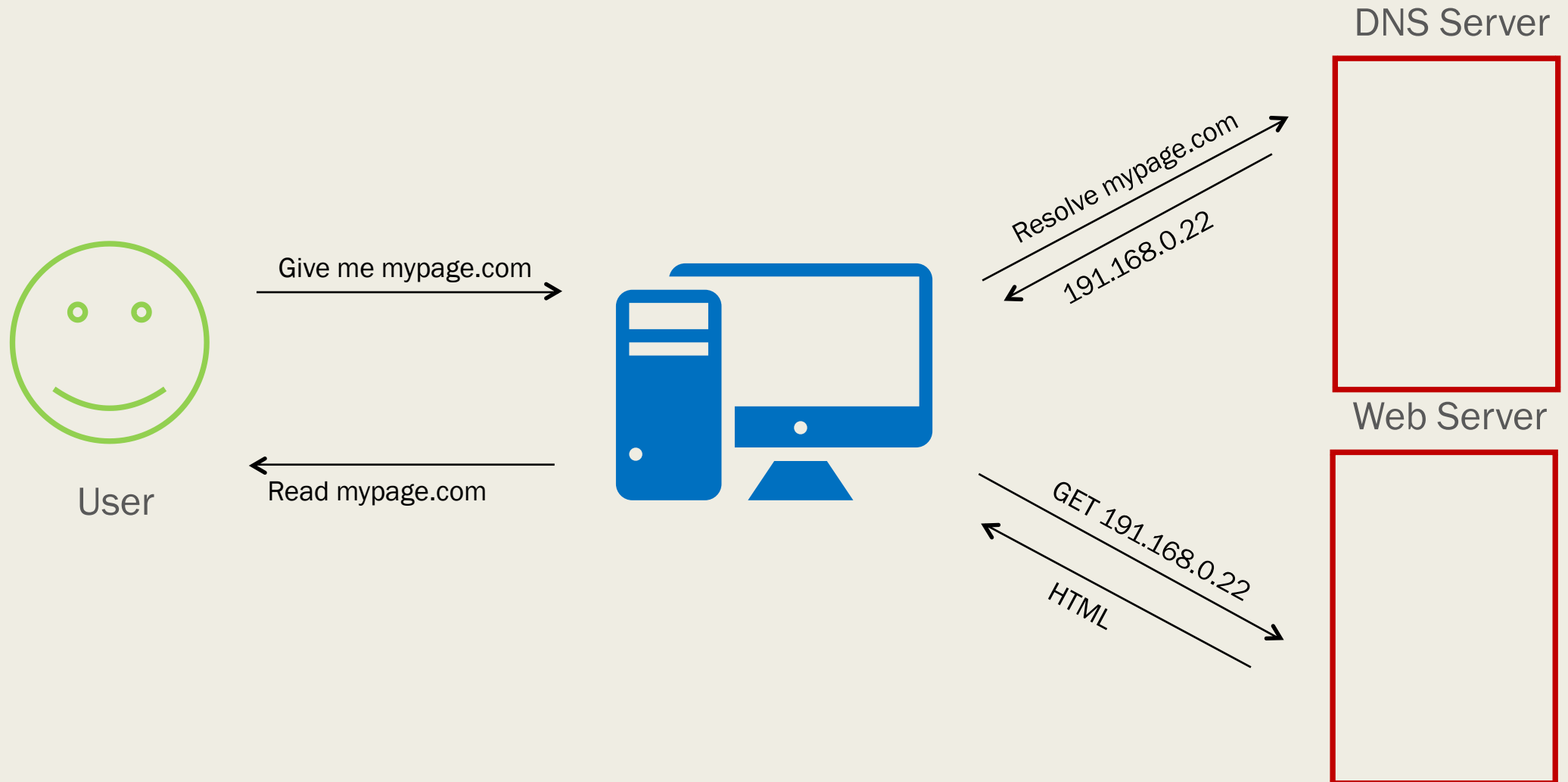
Response Fields

- Date
- Content-Type
- Content-Length
- Last-Modified
- Expires
- ...

Примерна операция – илюстрирана



Примерна операция – илюстрирана



Популярни уеб сървъри

Уеб Сървъри:

- Apache
- Microsoft IIS
(Internet Information Services)
- Nginx
- Google GWS
(използван вътрешно от Google)

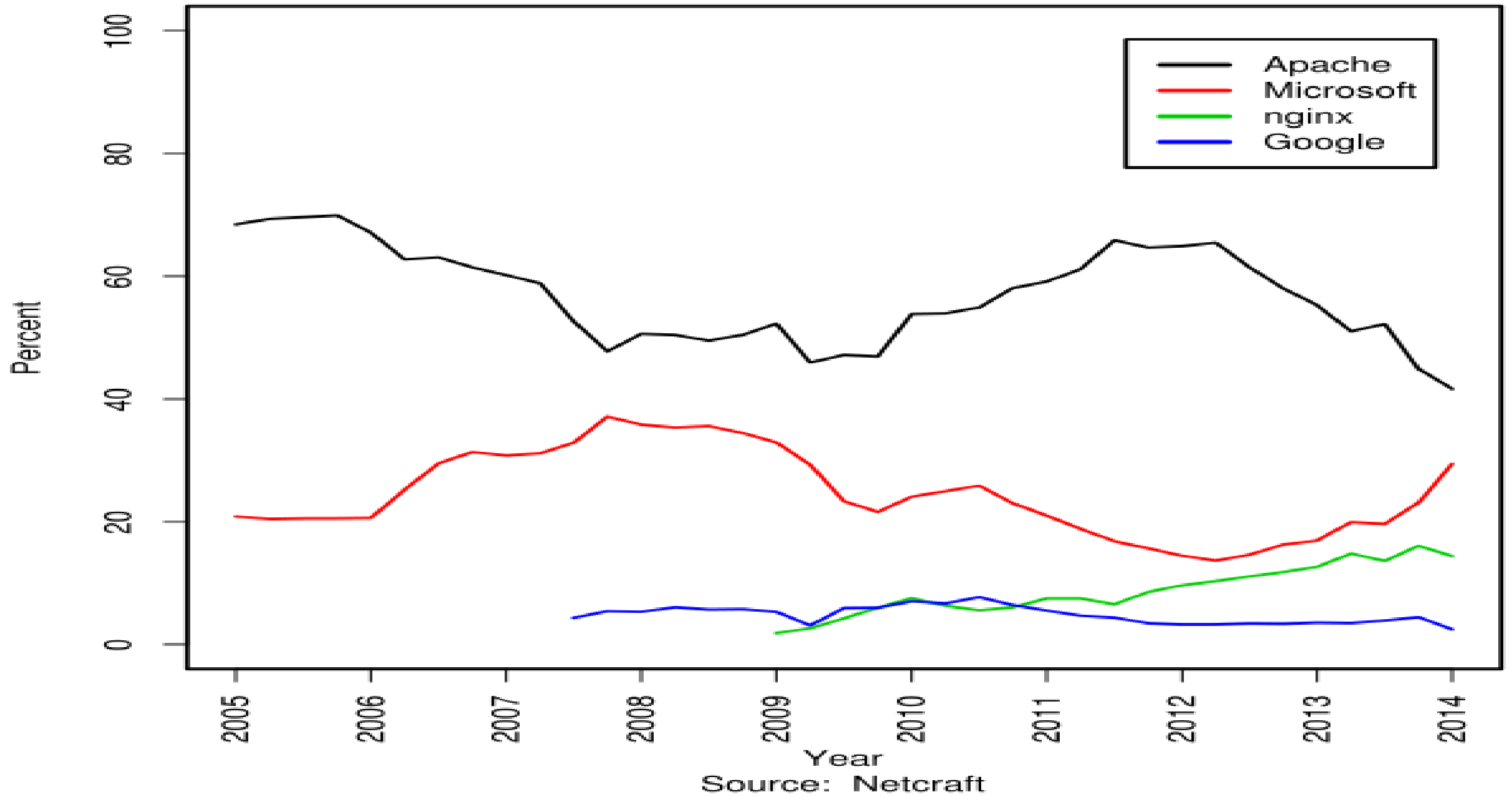
Специализирани сървъри:

- Apache Tomcat (Java)
- Jetty (Java)
- Node.js (JavaScript)

Само-хостващи се услуги

- ASP.NET Web API Self Host
- GO – вграден Self Host
- Java 6 Web API Self Host

Usage share of web servers



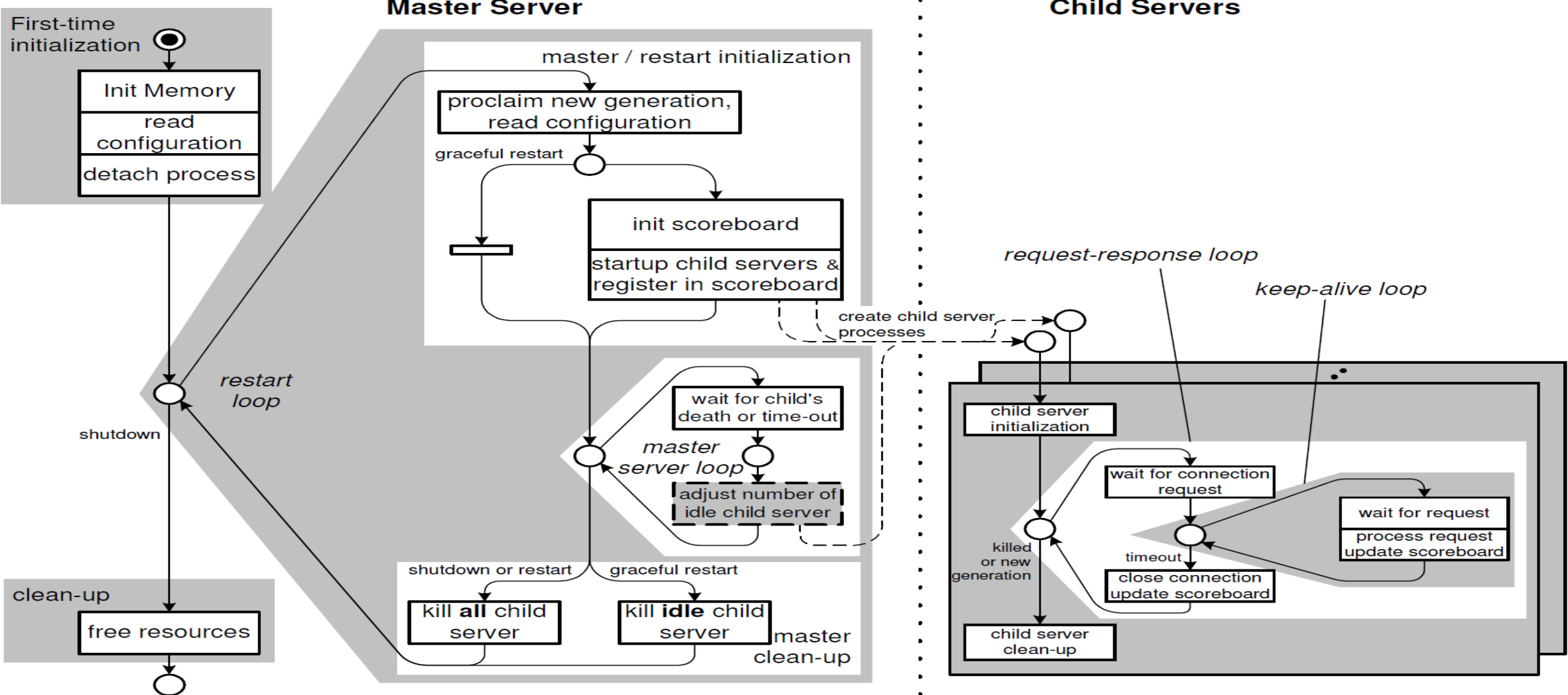
Server	CGI	FCGI	SCGI	WSGI	Java Servlets	SSI	ISAPI	SSJS	Administration console
Apache HTTP Server	Yes	Yes	Yes	Yes	No	Yes	Yes	Unknown	Yes
Apache Tomcat	Yes	No	Unknown	No	Yes	Yes	No	Unknown	Yes
Internet Information Services	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Jetty	Yes	Unknown	Unknown	No	Yes	Unknown	Unknown	Yes	Unknown
lighttpd	Yes	Yes	Yes	No	No	Yes	No	Unknown	No
nginx	No	Yes	Yes	Yes	No	Yes	No	Unknown	Yes

Basic access authentication

Digest access authentication

SSL/TSL криптиране през HTTPS

Виртуални хостове (virtual hosts)



Обща схема на работа на Apache 2

ВЪПРОСИ ?





РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

SOAP WEB SERVICES



Web application vs Web services

Web application

- Уеб приложенията са създадени за да бъдат използвани от крайните потребителите чрез уеб клиентски софтуер.

Примери за web applications:

- Facebook.com

- Twitter.com

- Google.com

- Github.com

и много други

Web services

- Уеб услугите са създадени за да бъдат използвани от други софтуерни приложения.

Web service: W3C definition

- “A software system designed to support interoperable machine-to-machine interaction over a internet...”

Защо са ни необходими Web services?

- За да се свързваме към вече съществуващ софтуер
- За да преизползваме вече съществуващата функционалност

Свързаност

- Свързване със съществуващ софтуер
 - *Уеб услугите помагат за справянето с проблеми с оперативната свързаност*
 - Предоставяйки на различни приложения възможност да обменят информация
 - *Уеб услугите предоставят възможност за свързването на две приложения написани на различни езици или работещи на различни платформи*

Преизползване

- Преизползваме вече съществуващата функционалност
 - *Идеята е всеки потребител да има достъп до вече съществуващата функционалност, както и да може да я преизползва.*

Формат на съобщенията

- JSON
- XML

JSON

- JSON (JavaScript Object Notation) е опростен формат за обмяна на данни, удобен за работа както за хората, така и за компютрите. Той е базиран на едно подмножество на езика за програмиране JavaScript, Standard ECMA-262 3rd Edition - от декември 1999.

JSON реализация

- Колекция от двойки име/стойност. В различните езици, това се реализира като обект, запис, структура, речник, хеш таблица, именован списък, или асоциативен масив.
- Подреден списък от стойности. В повечето езици, това се реализира чрез масив, вектор, списък или последователност.

JSON

```
1 {  
2   "name": "Christopher Co",  
3   "age": 29,  
4   "level": 7,  
5   "gender": "M",  
6   "status": "good"  
7 }
```

XML

- eXtensible Markup Language – разширяем маркиращ език (метаезик), дефиниращ правила за създаване на специализирани маркиращи езици. Сам по себе си той е безполезен, защото указва само как да бъде структуриран един документ.

```
<person>  
  <name>Иван Димитров Георгиев</name>  
  <country>България</country>  
  <language>български</language>  
  <language>руски</language>  
</person>
```

XML документ

- Уеб услугите нямат потребителски интерфейс
- Не е възможен достъп без специален софтуер
 - *Този софтуер трябва да позволява изграждането на XML (или JSON) заявка посредством собствен интерфейс*
 - *Най – често потребителския интерфейс е текстов редактор*

Web services

■ SOAP



■ RESTful



Какво представлява SOAP-based services?

- SOAP е платформа обединяваща технологиите XML и HTTP
 - *HTTP*
 - най – използвания протокол
 - *XML*
 - Предоставя език, който може да бъде използван от много и различни платформи и езици за програмиране
 - Предоставя сложни съобщения и функционалност

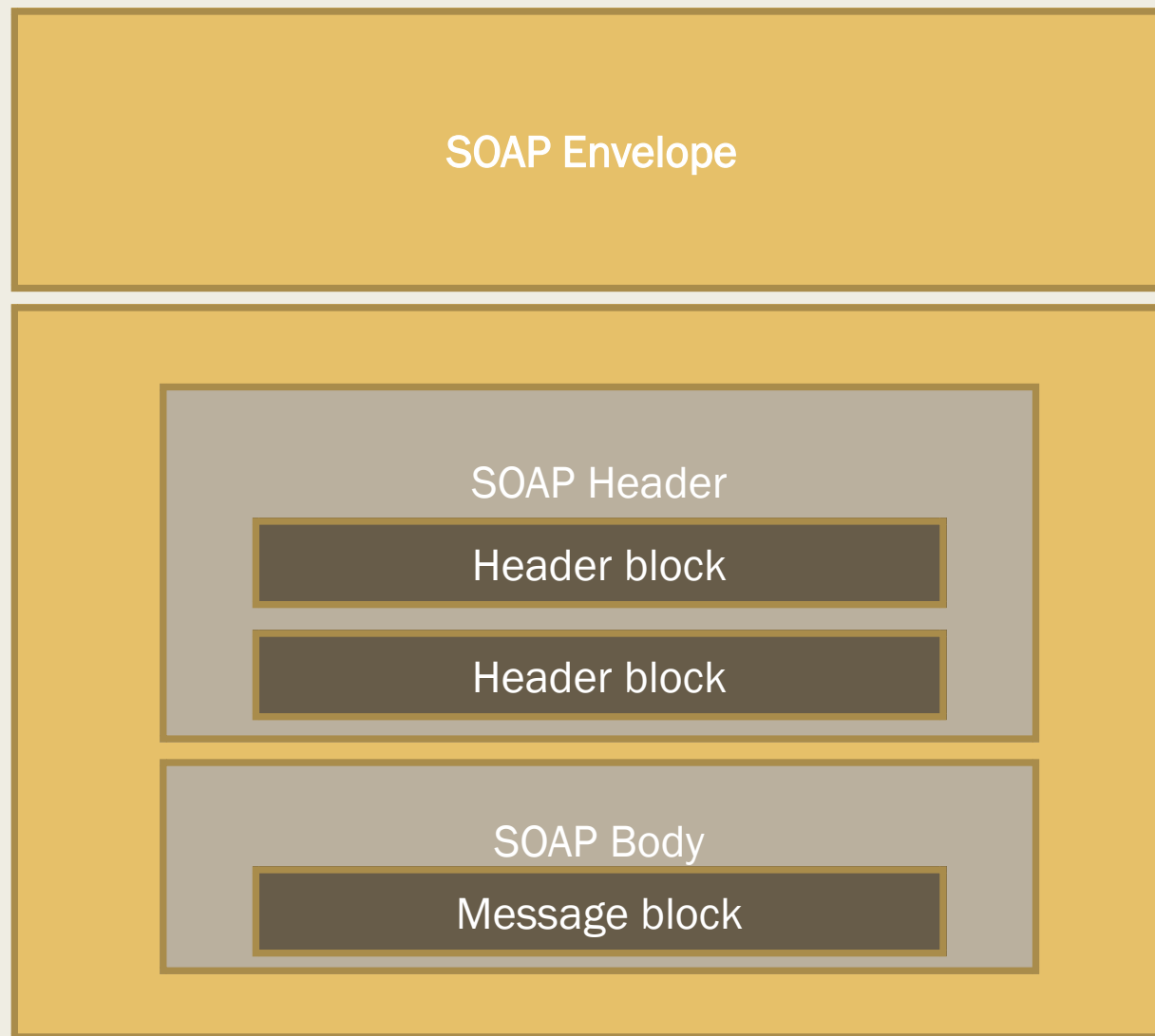
Елементи – SOAP базирани услуги

- SOAP
 - *Simple Object Access Protocol*
- WSDL
 - *Web Service Description Language*
- UDDI
 - *Universal Description, Discovery, and Integration*

SOAP е:

- Протокол за общуване
- Услуга за комуникация между приложения
- Формат на изпращане на съобщения
- Създаден за комуникация през интернет
- Платформено независим
- Езиково независим
- Базиран на XML
- Опростен и разширяем
- Позволява заобикалянето на firewall
- Разработван като W3C стандарт

SOAP структура



SOAP съобщение



WSDL е:

- WSDL е XML базиран език
 - *Услуга за описание на уеб услуги и достъпът до тях*
- Стандарт за Web Services Descriptions Language
- Описан с XML
- WSDL представлява XML документи
- Използва се за описание на уеб услуги
- Използва се и за описание на локални уеб услуги
- Все още не е W3C стандарт

WSDL Структура

- Definition
- TargetNamespace
- DataTypes
- Messages
- Porttype
- Bindings
- Service

WSDL документ

```
<!-- WSDL definition structure -->
<definitions
    name="Guru99Service"
    targetNamespace=http://example.org/math/
    xmlns=http://schemas.xmlsoap.org/wsdl/>
    <!-- abstract definitions -->
    <types> ...
        <message> ...
        <portType> ...

    <!-- concrete definitions -->
    <binding> ...
    <service> ...
</definition>
```

WSDL Елементи

■ <type>

- Описани са всички комплексни типове, които ще бъдат използвани при размяната на съобщения между сървъра и клиента.

string, integer

■ <messages>

- Дефинира се съобщението

■ <portType>

- Използва се за да капсулира всяко входящо и изходящо съобщение в логическа операция. Име на операцията

"GetEmployee"

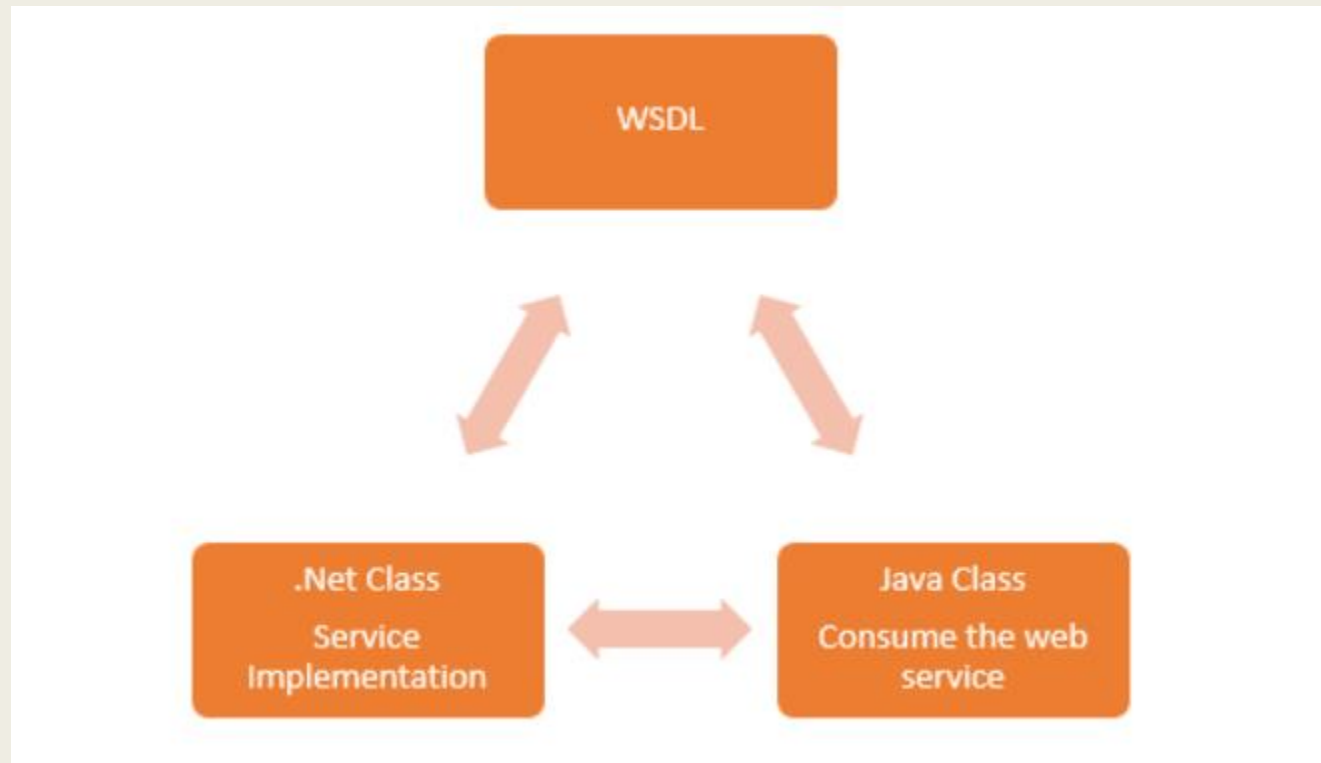
■ <binding>

- Използва се за да свърже конкретна операция към конкретен порт

■ <service>

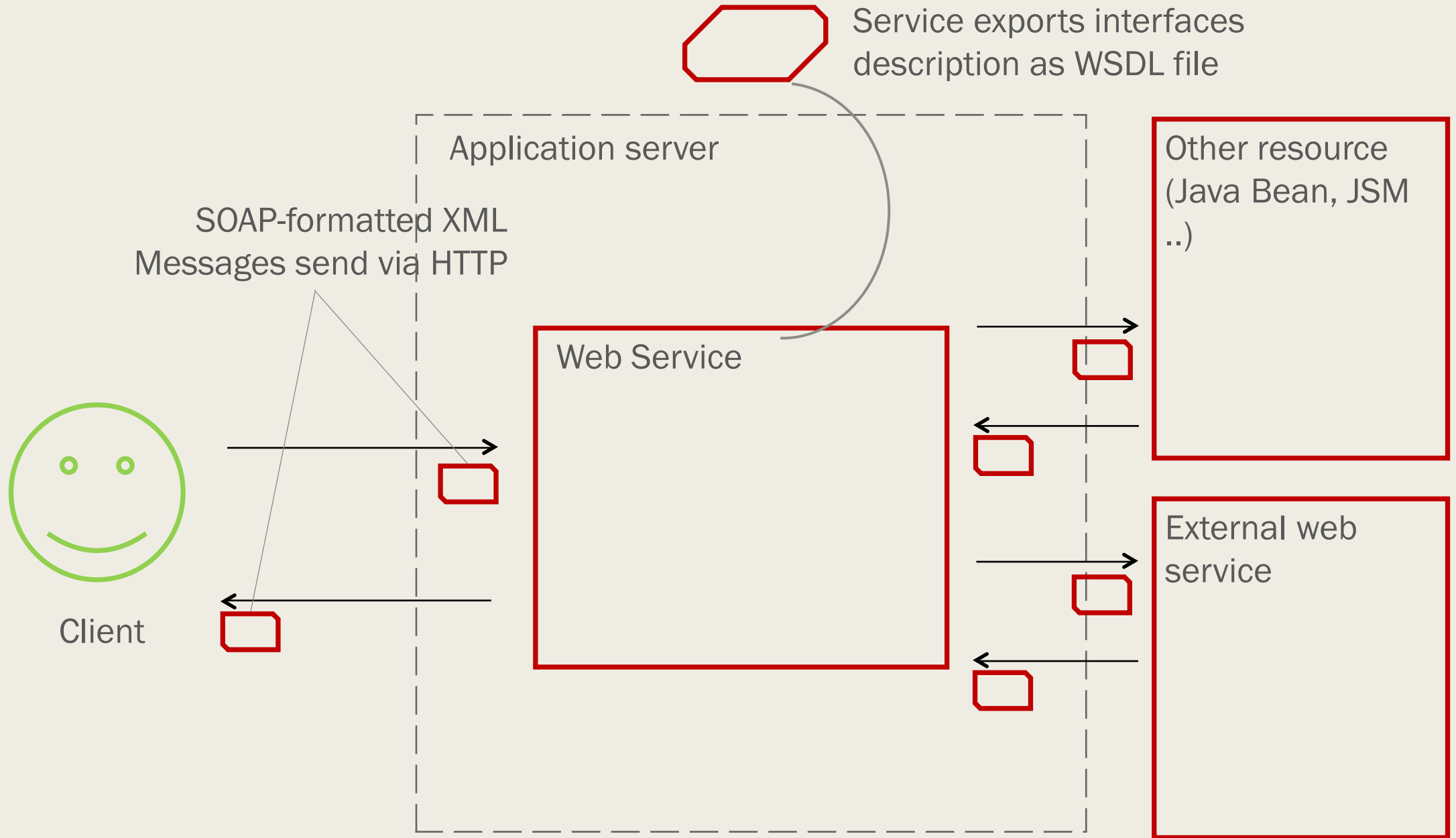
- Записва се името на конкретната услуга
http://localhost/Guru/Tutorial.asmx

Благодарение на WSDL може да
имаме следната комуникация



UDDI e:

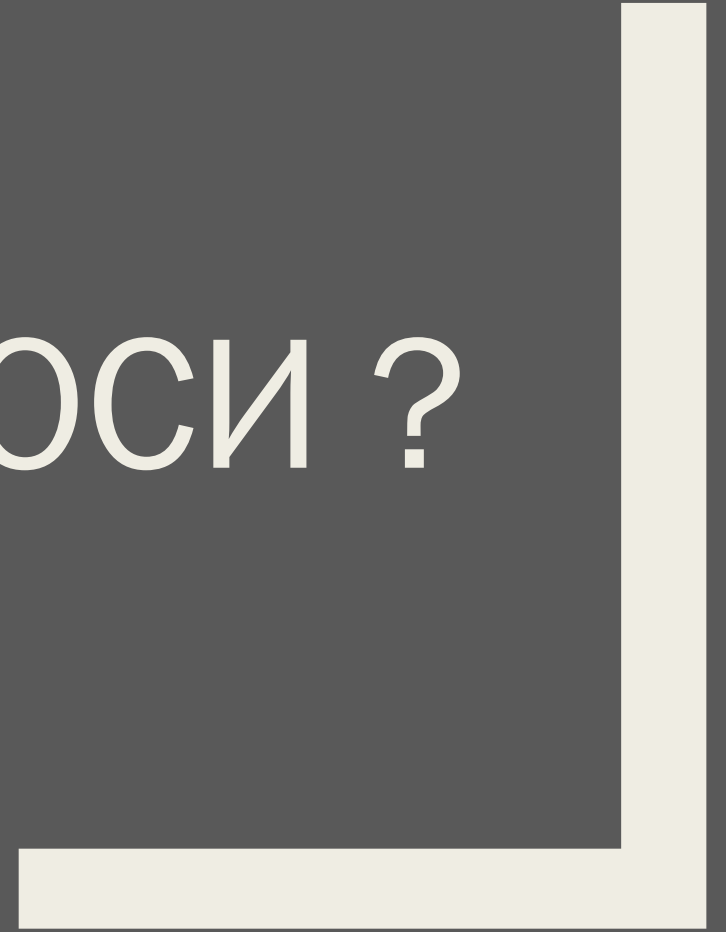
- Спецификация за разпределено регистриране на уеб услуги
- Независим от вида на платформата за разработка
- Рамка с отворен код
- Може да общува посредством SOAP, CORBA и JAVA RMI Protocol
- Използва WSDL за да опише интерфейс за уеб услуги



SOAP сигурност

- SOAP предлага два начина за осигуряване на сигурна връзка:
 - Чрез специфичен елемент наречен *UsernameToken* (изпращат се потребителско име и парола)
 - Другия начин е *Binary Token* чрез *BinarySecurityToken* (като се използват X.509 сертификати)

ВЪПРОСИ ?



РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

RESTFUL WEB SERVICES



Какво е REST?

- Representational State Transfer (REST) е архитектурен модел за изграждане на разпределени системи. Уеб е пример за такава система.

REST технологии

- REST не е обвързан с конкретно множество от технологии. Но най – често използваните технологии са:
 - *URI*
 - *HTTP* глаголите
 - *XML* и *JSON*

Ограниченията на архитектурния стил REST засягат следните архитектурни свойства

- Производителност при взаимодействие на компоненти; (ефективност на мрежата)
- Мащабируемост, позволяваща поддръжката на голям брой компоненти и взаимодействия между компонентите;
- Простота на единия интерфейс;
- Изменяемост на компонентите, за да отговори на променящите се нужди (дори при работещо приложение);

- Видимост на комуникацията между компонентите от услуги агенти;
- Преносимост на компонентите;
- Надеждност при възникването на проблеми на системно ниво в компоненти, конектори или данните.

REST ограничения

- Клиент-сървър архитектура (Client-server architecture)
 - *Принципът, който стои зад ограниченията клиент-сървър, е разделянето на проблемите. Разделянето на проблемите на потребителския интерфейс от проблемите за съхранение на данни подобрява преносимостта на потребителските интерфейси в множество платформи.*

■ Statelessness

- *Комуникацията клиент-сървър е ограничена от това, че клиентският контекст не се съхранява на сървъра между заявките. Всяка заявка от всеки клиент съдържа цялата информация, необходима за обслужване на заявката, и състоянието на сесията се съхранява в клиента. Състоянието на сесията може да бъде прехвърлено от сървъра към друга услуга, чрез база данни, за да поддържа устойчиво състояние за период и да позволява верификация.*

■ Cacheability

- *Както в World Wide Web, клиентите и посредниците могат да кешират отговорите. Добре управляваният кеш елиминира допълнителните заявки между клиент-сървър и подобрява, производителността и мащабируемостта.*

■ Layered system

- Клиентът не може да каже дали е свързан директно към крайния сървър или към посредник. Ако proxy (reverse proxy), или балансър (load balancer) е поставен между клиента и сървъра, това няма да повлияе на комуникацията и няма да е необходимо да актуализираме клиентския или сървърния код. Посредническите сървъри могат да подобрят мащабируемостта на системата, като позволяват балансиране на натоварването и предоставяне на споделен кеш.

■ Code on demand - опционалност

- Сървърите могат временно да разширят или персонализират функционалността на клиента, като прехвърлят изпълним код: например компилирани компоненти като Java applet или клиентски скриптове като JavaScript.

■ Uniform interface

- *Това е основното ограничение за това как трябва да работят REST уеб услугите. В основата е разделянето на архитектурата от компонентите и тяхното независимо развиване.*

Идентификация на ресурса в заявките; Манипулиране на ресурси чрез репрезентации; Самоописателни съобщения.

Какво представлява RESTful web service?

- Всяка веб услуга, която отговаря на REST ограниченията може да бъде наричана RESTful web service (веб услуга).

Ако някое ограничение не е изпълнено системата не може да се нарече RESTful.

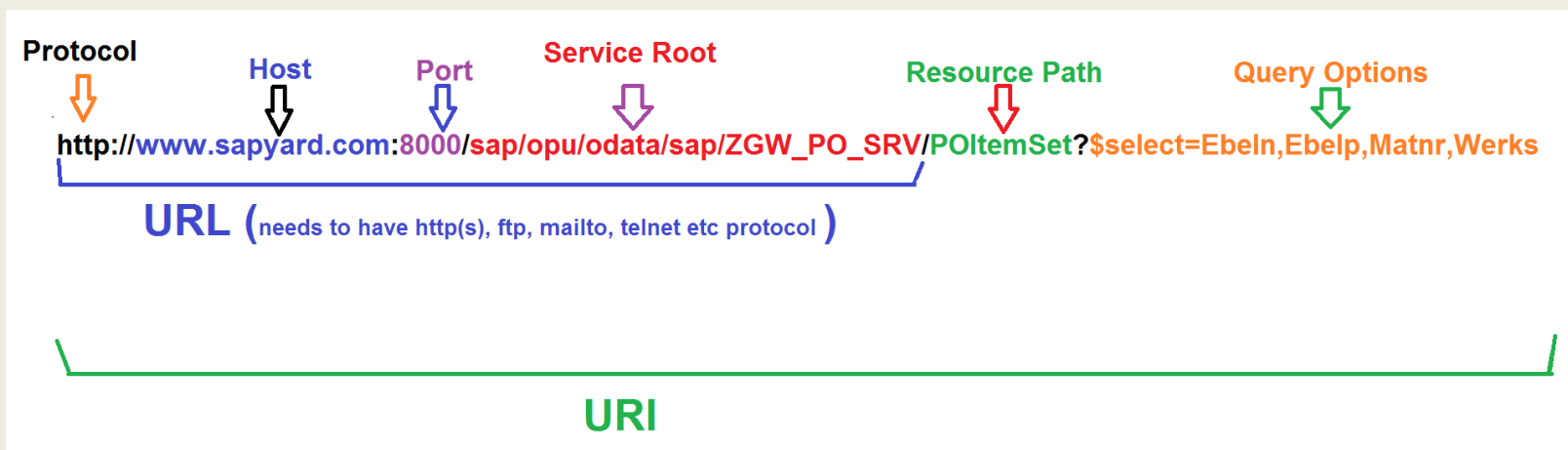
Основни елементи на RESTful имплементацията

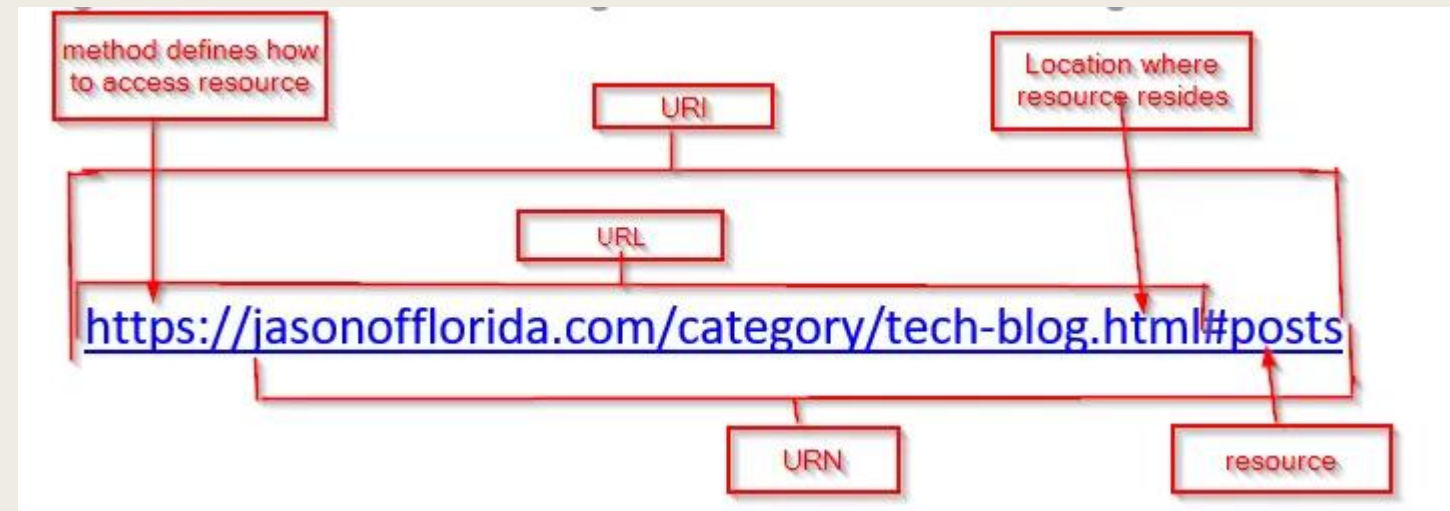
- Ресурси (Resources)
- Глаголи на заявки (Request verbs)
- Допълнителни данни на заявката (Request Headers)
- Тяло на заявката (Request Body)
- Тяло на отговора (Response Body)
- Статус на отговора (Response Status codes)

Преди да поговорим за ресурси
трябва да изясним какво са URI, URL
и URN

Какво е URI и URL

- URI е Uniform Resource Identifier е низ от знаци, който недвусмислено идентифицира определен ресурс. Най-често срещаната форма на URI е Uniform Resource Locator (URL), често наричан неофициално уеб адрес.





RESTful URLs или Clean URLs

- RESTful URLs или Clean URLs представляват едно и също нещо. Идеята е да се подобри преизползваемостта и достъпа до услугите и уеб сайтовете от неексперти. Това се постига чрез преработката на querystring.

Примери за Clean URLs

Uncleaned URL	Clean URL
http://example.com/index.php?page=name	http://example.com/name
http://example.com/about.html	http://example.com/about
http://example.com/index.php?page=consulting/marketing	http://example.com/consulting/marketing
http://example.com/products?category=12&pid=25	http://example.com/products/12/25
http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss	http://example.com/news.rss
http://example.com/services/index.jsp?category=legal&id=patents	http://example.com/services/legal/patents
http://example.com/kb/index.php?cat=8&id=41	http://example.com/kb/8/41
http://example.com/index.php?mod=profiles&id=193	http://example.com/profiles/193
http://en.wikipedia.org/w/index.php?title=Clean_URL	http://en.wikipedia.org/wiki/Clean_URL

Ресурси (Resources)

- Ресурсите са дефинирани от URI
 - *Ресурсите не могат да бъдат достъпни или манипулирани директно*
 - *RESTful работи с ресурсни репрезентации*

Нека предположим, че уеб приложение на сървър има записи на няколко служители. Да приемем, че URL адресът на уеб приложението е `http://demo.com`. Сега, за да получите достъп до ресурс за запис на служители чрез REST, човек може да издаде командата `http://demo.com/employee/1` - Тази команда казва на уеб сървъра да предостави подробности за служителя, чийто номер на служителя е 1

Какво наричаме съществителни в RESTful

- Съществителните са имената на ресурсите
 - При повечето дизайни, тези имена са URI-те
 - URI дизайна е много важна част от REST-базираният системен дизайн
- Всичко значимо би трябвало да е именувано
 - Поддържайки добре създадени имена (RESTful URLs)

Глаголи на заявки (Request verbs)

- Операциите, които могат да бъдат извършвани върху ресурси
- Основната идея на REST е да използва само универсални глаголи
 - *Универсалните глаголи могат да бъдат приложени върху всички съществителни*

- За повечето приложения, основните глаголи на HTTP са достатъчни
 - *GET*: Връща репрезентация на ресурс (трябва да няма странични ефекти)
 - *PUT/PATCH*: Прехвърля репрезентация от конкретен ресурс (презаписва вече съществуваща такава)
 - *POST*: Добавя репрезентация към конкретен ресурс
 - *DELETE*: Премахва репрезентация
- Глаголите съответстват на най-популярните операции
 - *CRUD*: *Create, Read, Update, Delete* (Създаване, Четене, Промяна, Изтриване)

Допълнителни данни на заявката (Request Headers)

- Това са допълнителни инструкции, изпращани със заявката. Те могат да определят типа на необходимия отговор или подробностите за верификациите.

Тяло на заявката (Request Body)

- Представява информацията изпращана със заявка. Най – често се използва в комбинация с глаголите POST, PUT и PATCH.

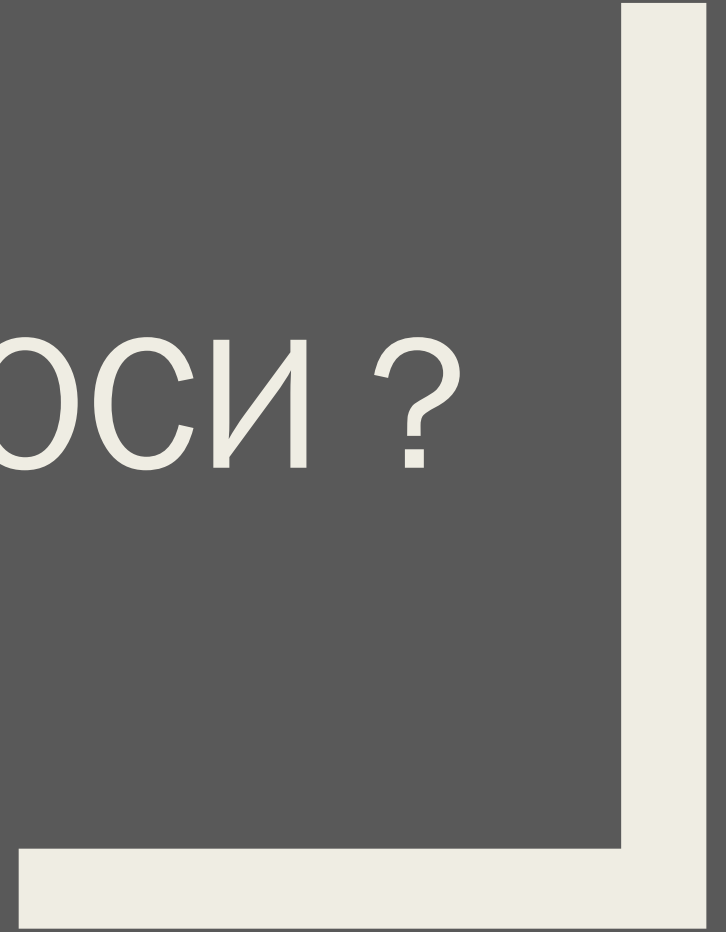
Тяло на отговора (Response Body)

- Представява информацията получавана при отговор на заявката. Тя може да бъде предоставена в различни формати: обикновен текст, XML, HTML и JSON.

Заклучение

- SOAP представлява Simple Object Access Protocol където REST представлява Representational State Transfer.
- SOAP е протокол, докато REST е архитектурен модел.
- SOAP използва сервизни интерфейси, за да изложи функционалността си на клиентските приложения, докато REST използва локални услуги за достъп до компонентите на хардуерното устройство.
- SOAP се нуждае от голям bandwidth за използването му, докато REST не се нуждае от голям bandwidth.
- SOAP работи само с XML формати, докато REST работи с обикновен текст, XML, HTML и JSON.
- SOAP не може да използва REST, докато REST може да използва SOAP.

ВЪПРОСИ ?

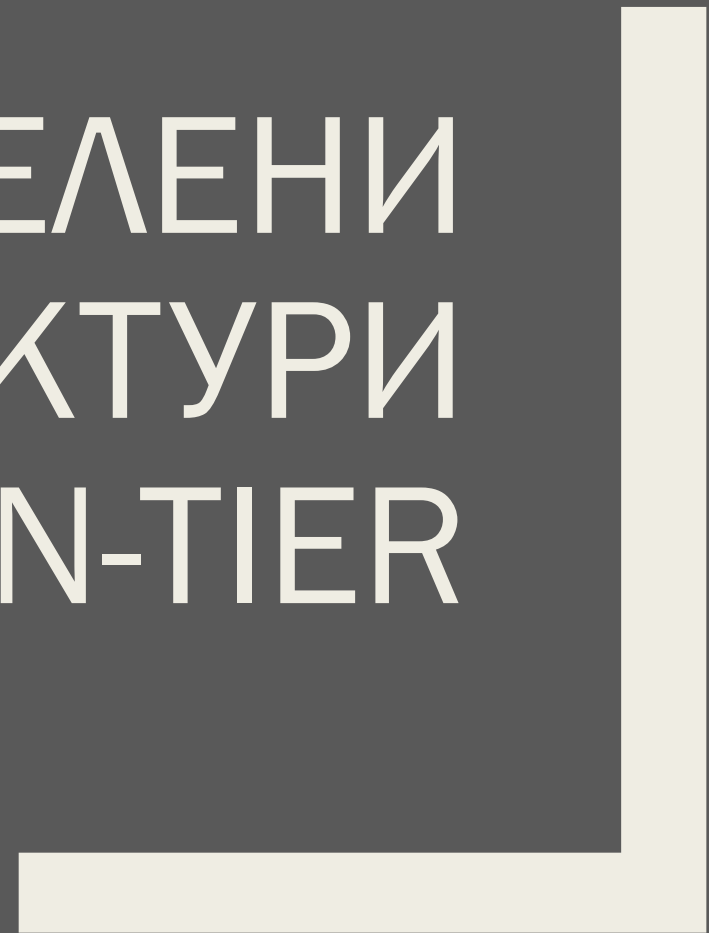




РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

РАЗПРЕДЕЛЕНИ АРХИТЕКТУРИ CLIENT-SERVER И N-TIER



Софтуерна архитектура

- Софтуерната архитектура е съвкупност от важни решения за организацията на програмните системи.

История

- Софтуерна архитектура като концепция има своите корени в изследванията на Едсгер Дейкстра през 1968 г. и Дейвид Парнас в началото на 1970.
- Добива широка популярност като термин в началото на 1990 години.

Архитектура

- Софтуерните архитектури са необходими за аргументиране на софтуерните системи. Всяка архитектура се съставява от софтуерни елементи, връзките между тях и свойствата на двете (елементи и връзки).

Основни твърдения:

- Софтуерните архитектури са важни за успешното разработване на софтуерни системи.
- Софтуерните системи се изграждат за да удовлетворят бизнес целите на организациите.
- Архитектурата е мост между (често абстрактни) бизнес целите и крайната система.
- Софтуерните архитектури могат да бъдат проектирани, анализирани, документирани и реализирани с определени технологии, така че да бъдат постигнати целите на бизнеса.

Архитектурни дейности

■ Архитектурен анализ

- *Включва процес на разбиране на средата, в която ще работи планираната система или системи, както и определяне на бизнес изискванията.*

■ Архитектурен синтез

- *Синтез или дизайн е процеса на създаване на архитектура.*

■ Оценяване на архитектурата

- *Процес на определяне на това колко добре настоящия дизайн или част от него отговаря на изискванията, получени по време на анализа.*

■ Еволюция на архитектурата

- *Процес на поддържане и адаптиране на съществуваща софтуерна архитектура. Целта е тя да продължи да отговаря на изискванията и промените в заобикалящата и среда.*

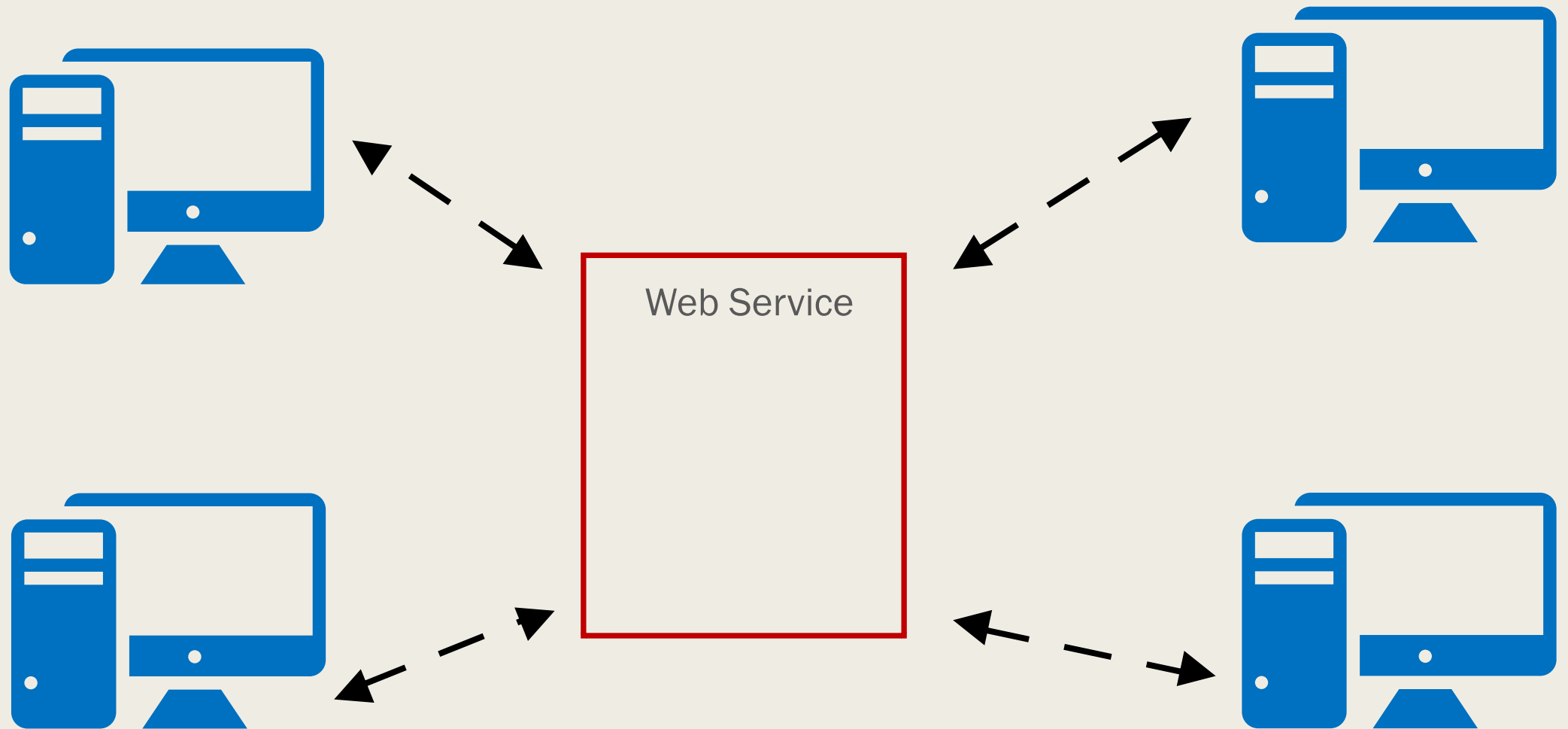
Архитектури, които ще разгледаме

- Client-Server
- N-tiers (3-tiers)
- SOA
- Microservices

Client-Server

- Моделът клиент-сървър е разпределена архитектура на приложения, която разделя задачите (натоварването) между доставчиците на ресурс (услуга), наречени сървъри и инициатори, наречени клиенти.

Какво е Client-Server?



Какво е клиент-сървър приложение?

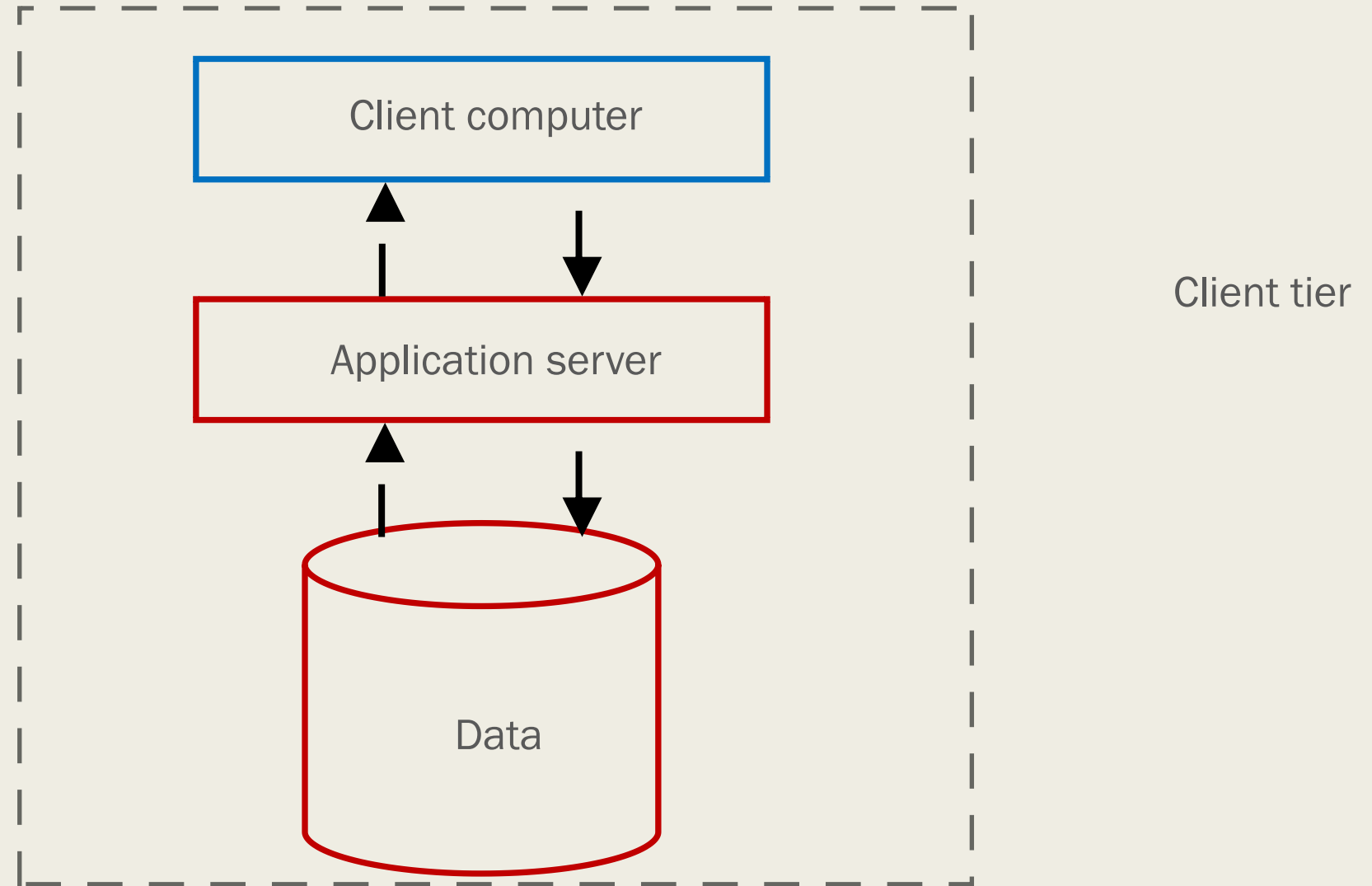
- Посредством клиента, потребителят взаимодейства с приложението
- Един сървър може да обслужва много клиенти, и да предоставя функционалности, които не могат да се реализират при клиента в изолация
- Когато клиента има нужда от информация или да предаде на сървъра информация в следствие интеракцията на потребителя с приложението, клиента може да изпрати съобщение към сървъра
- В отговор на това съобщение, сървъра изпълнява необходимите действия и може да върне съобщение (наречено отговор)

- При някои клиент-сървър приложения сървъра може да изпраща съобщения към клиента, които да не са задължително отговор на съобщения изпратени от клиента
- Често клиента и сървъра комуникират през компютърната мрежа (локална мрежа или интернет) и се намират на различен хардуер

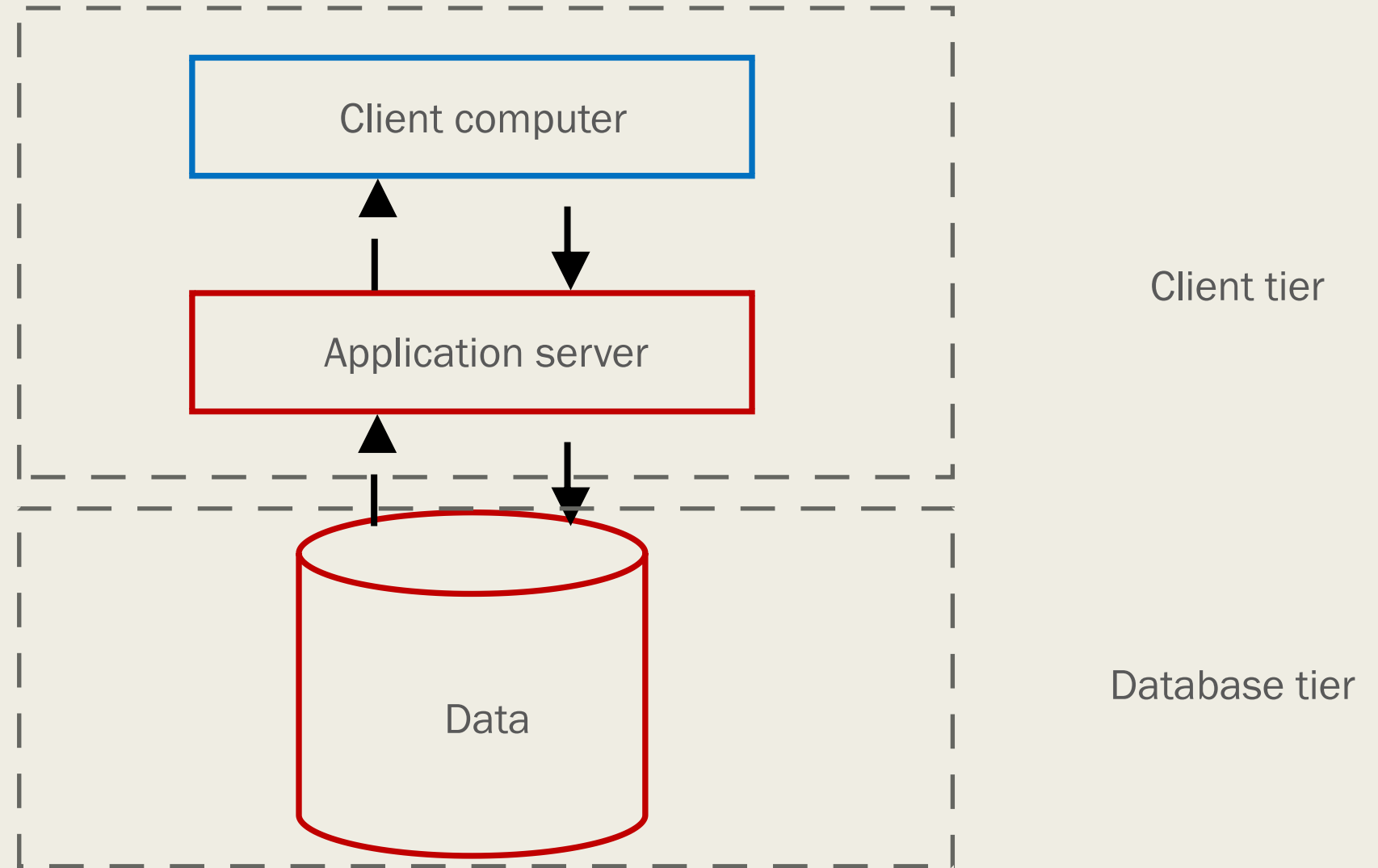
N-tier(3-tier)

- Многослойната архитектура е вид клиент – сървър архитектура, при която бизнес логиката е разделена на отделни компоненти. Най – често презентационната, бизнес логиката и логиката за управление на даните са отделени физически.

1-tier architecture



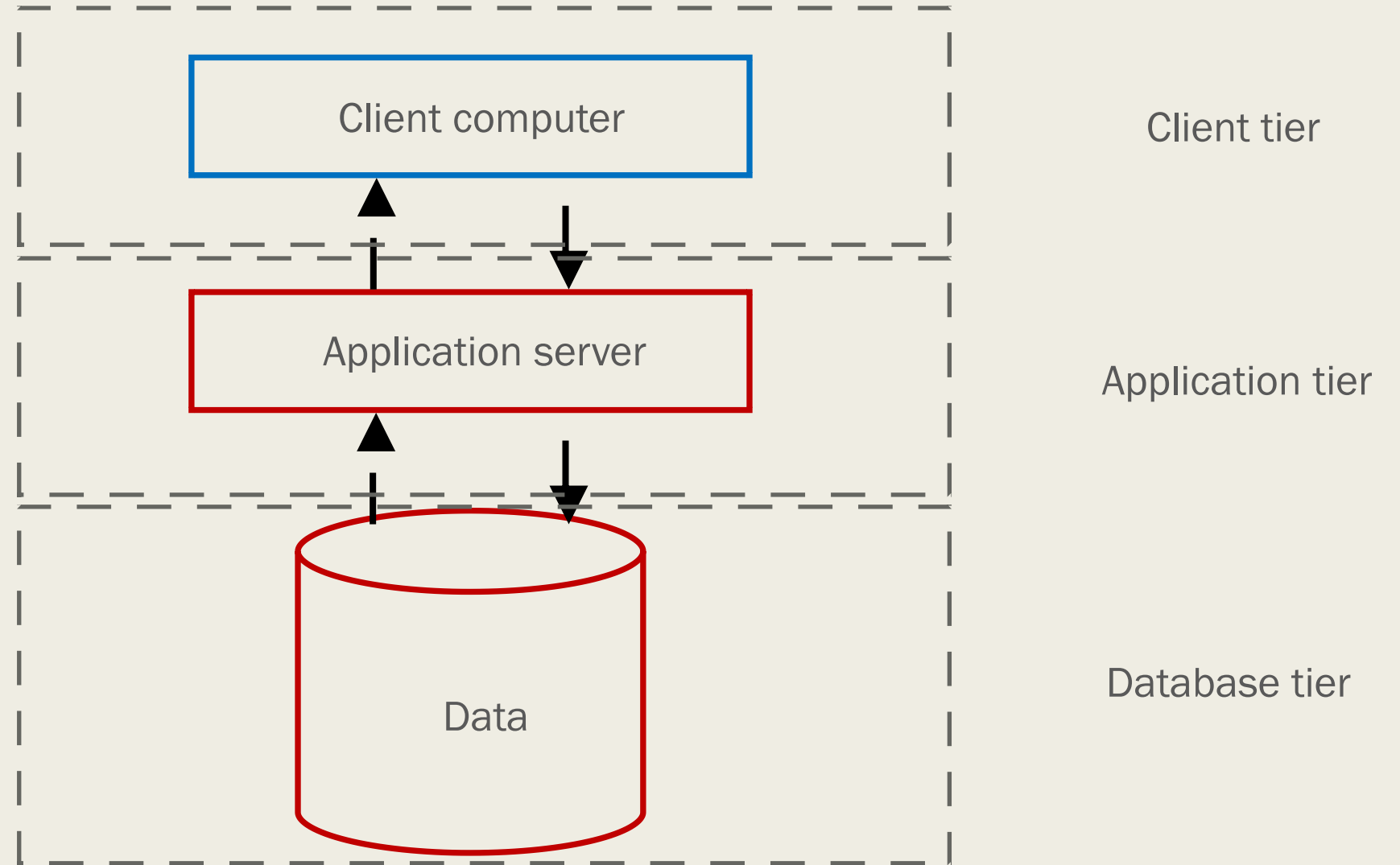
2-tier architecture



Трислойна архитектура

- Презентационен слой
 - Презентационния слой е слоя, до който потребителят има директен достъп
- Слой за бизнес логика (БЛ)
 - БЛ слоя контролира функционалността на приложението (бизнес логиката) и е съсредоточен в обработката на данните.
- Слой за данните
 - Този слой се състои от сървър база данни (СУБД). Служи за управление на данните.

3-tier architecture



Типове N-tier

- Closed layer architecture

- *В затворената архитектура, всеки слой може да се свърже само с най – близкия до него.*

- Open layer architecture

- *В отворената архитектура, всеки слой може да се свърже с всеки слой намиращ се под него.*

Кога да използваме N-Tier?

- При опростени уеб приложения
- Миграция на локални приложения към облачна среда (AWS, Azure, Google cloud)
- При обединена разработка на локално и облачно базирано приложения

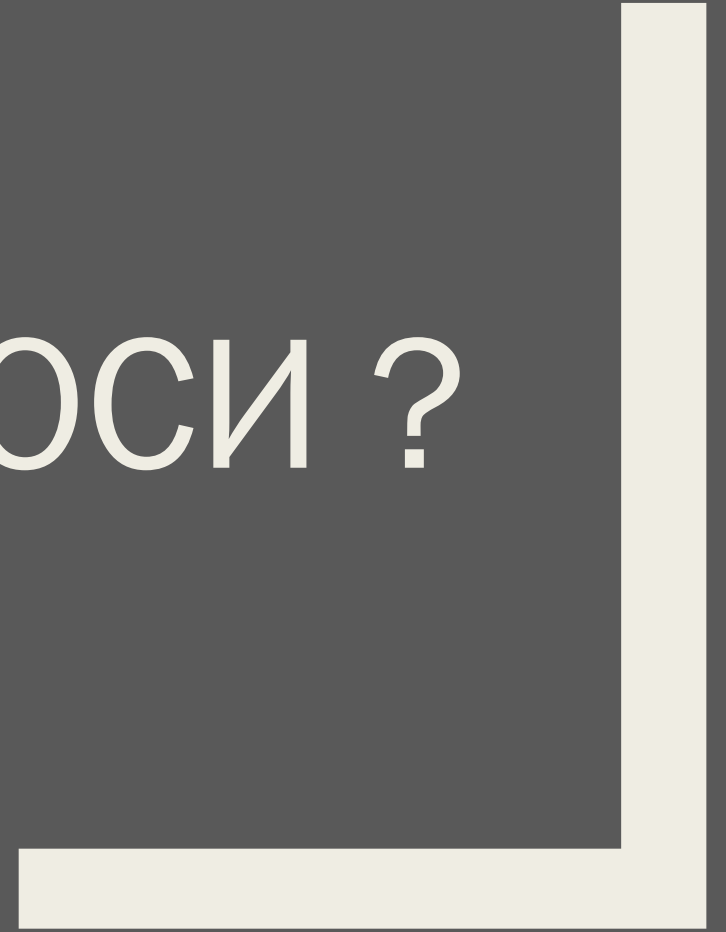
Предимства

- Преносимост на едно приложение между облачна и локална платформа, или различни облачни платформи
- По – малко време за обучение на голяма част от разработчиците
- Естествена еволюция от традиционния модел на разработка
- Отворен към комбинация на различни среди на разработка (Windows, Linux, Unix ...)

DEMO N-TIER

https://github.com/pkyurkchiev/n-layer_skeleton_net-core

ВЪПРОСИ ?

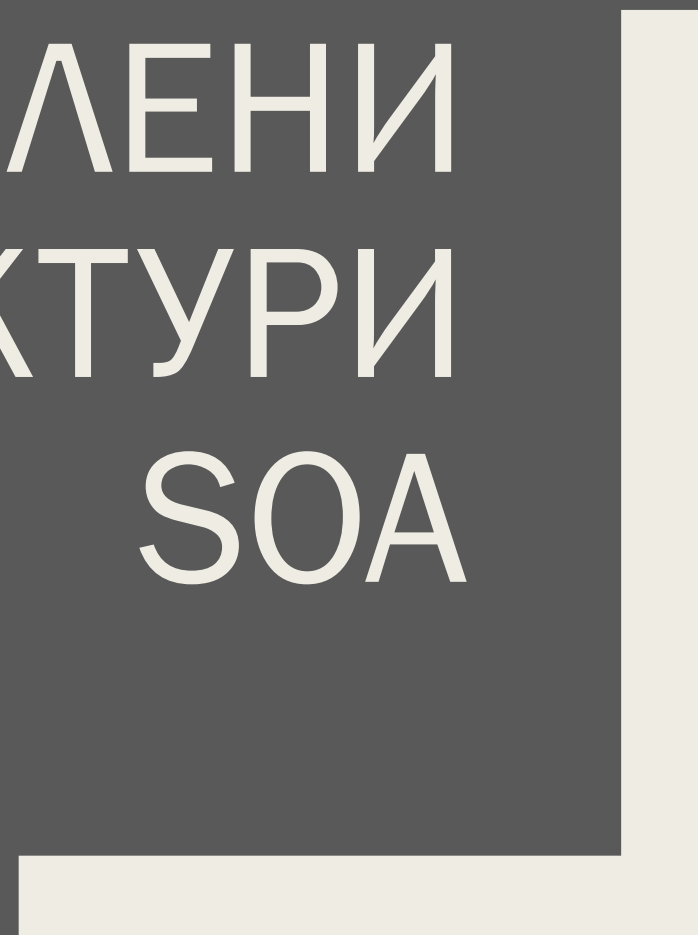




РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

РАЗПРЕДЕЛЕНИ АРХИТЕКТУРИ SOA

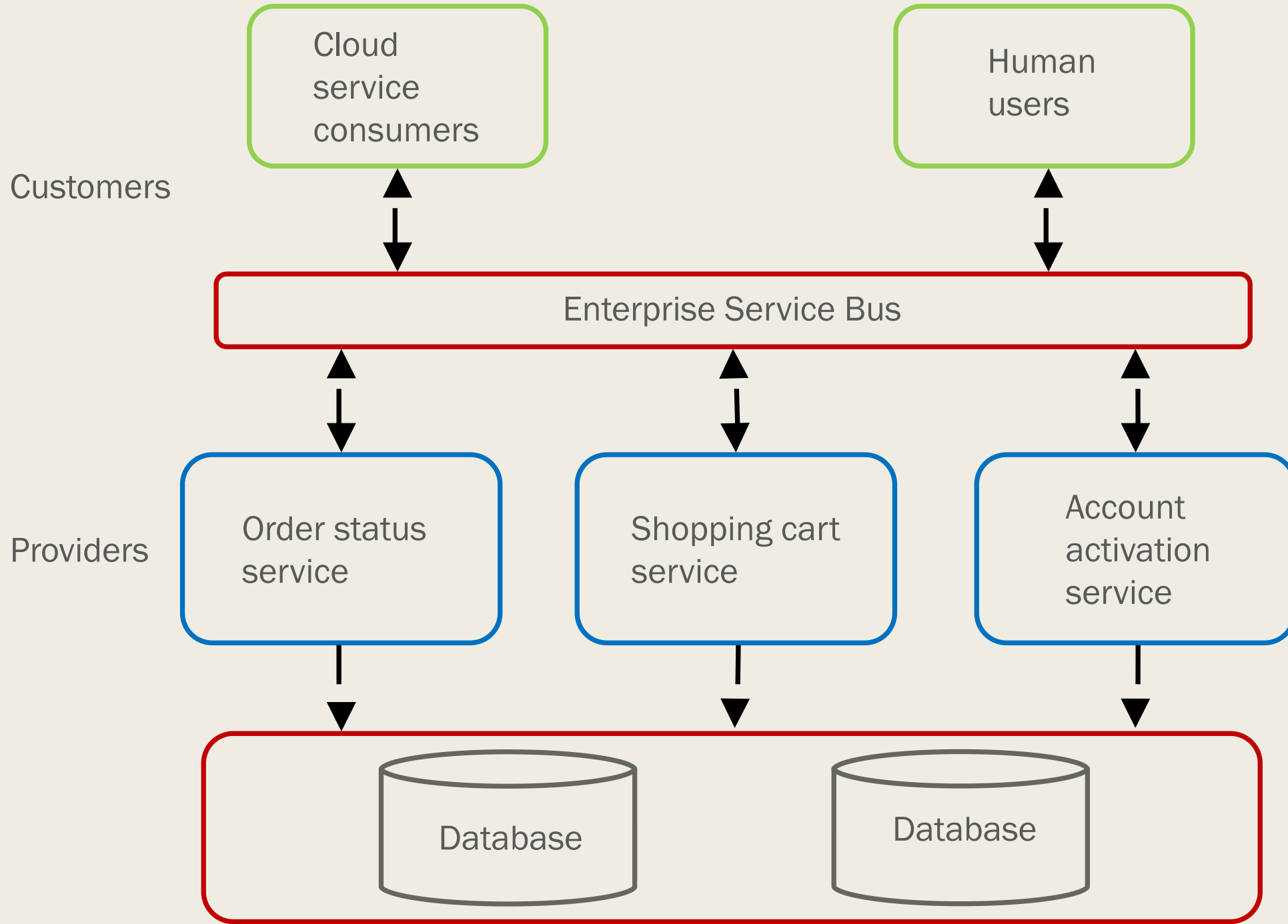


Service-oriented architecture (SOA)

- SOA е стил на софтуерен дизайн, при който услугите се предоставят посредством софтуерни компоненти, чрез комуникационен протокол помежду им през интернет мрежата.

SOA основни компоненти

- Service provider
 - Той създава услугата и предоставя информацията за нея към съответния регистър (Предоставя достъп до нея).
- Service broker, service registry or service repository
 - Основната му функция е да предостави информацията за веб услугата на всеки потенциален клиент.
- Service requester/consumer
 - Това са всички клиенти, които достъпват и използват информацията предоставена от услугите.



SOA и Web 2.0

Web 2.0

- User generated content
- Services Model
 - *Сървис ориентира архитектура*
- Поява на RSS, Web API ...

SOA услуги

- Услуги са градивния блок на SOA архитектурата. Те могат да бъдат изградени от други услуги. Представяват черна кутия за потребителя(той не знае как те работят). Услуги представят бизнес логиката на приложението.

Характеристики на услугите

- Автономни – независими са от останалата част на системата
- Stateless – не знаят за предишни изпратени от вас заявки (не пазят състоянието на заявката)
- Приемат заявка и връщат отговор
- Добре описани са
- Стъпват на стандартизирани интерфейси

Характеристики на услугите

- Комуникацията се осъществява през стандартни протоколи:
 - *HTTP, FTP, SMTP, RPC, MSMQ....*
 - *JSON, XML, SOAP, RSS, WS-....*
- Платформено независими са

Свойства на услугите

- Услугата представлява бизнес активност със специален изход(резултат).
- Услугата е самостоятелна, самосъдържаща се.
- Услугата е черна кутия за ползвателите.
- Услугата може да се състои от други скрити отдолу услуги.

Принципи на SOA – препокриват тези на услугите

- Стандартизирани условия на услугите
 - *Услугите се придържат към стандартни комуникационни условия, като се дефинират от един или повече документи, описващи услугите за дадения набор от уеб услуги.*
- Автономност на връзката между услугите (част от loose coupling)
 - *Връзката между услугите е минимализирана до ниво, в което те знаят само за тяхното съществуване.*
- Прозрачност на местоположението на услугите (част от loose coupling)
 - *Услугите могат да бъдат извикани от всякъде в мрежата, без значение къде се намират.*

■ Дълголетие на услугата

- *Услугите могат да се проектират така, че да имат дълъг живот. Където е възможно услугите трябва да избягват налаганите от ползвателите форми на промяна на бизнес логиката им. В сила трябва да е следното правило "ако извикаш една услуга днес, трябва да можеш да извикаш същата услуга и утре".*

■ Абстрактни услуги

- *Услугите трябва да работят като черна кутия, по този начин тяхната вътрешна логика е скрита от ползвателите.*

■ Автономни услуги

- *Услугите са независими и контролират функционалностите, които се скриват, от страна на Design-time(времето за разработка) и run-time(времето за действие).*

■ Услуги без състояние

- *Услугите нямат състояние, те или връщат положителен резултат, или връщат изключение(exception). Едно извикване на услугата не трябва да влияе на друго такова.*

■ Детайлност на услугата

- *Принцип целящ да подsigури адекватния размер и обхват на услугата. Функционалностите предоставени на ползвателя от услугата трябва да бъдат уместни.*

■ Нормализиране на услуга

- *Услугите се декомпонизират и консолидират(нормализират), за да се намали излишното. За някои услуги това не може да се направи. Това са случаите, в които бързодействието, лесния достъп и агрегацията са нужни.*

■ Композитност на услуги

- *Услугите могат да се използват за да се композират други услуги.*

■ Откриване на услуги

- *Услугите имат специални комуникационни метаданни, чрез които услугите лесно могат да се откриват и достъпват.*

■ Преизползване на услуги

- *Логиката е разделена на малки парчета, което позволява създаването на отделни малки услуги и тяхното лесно преизползване.*

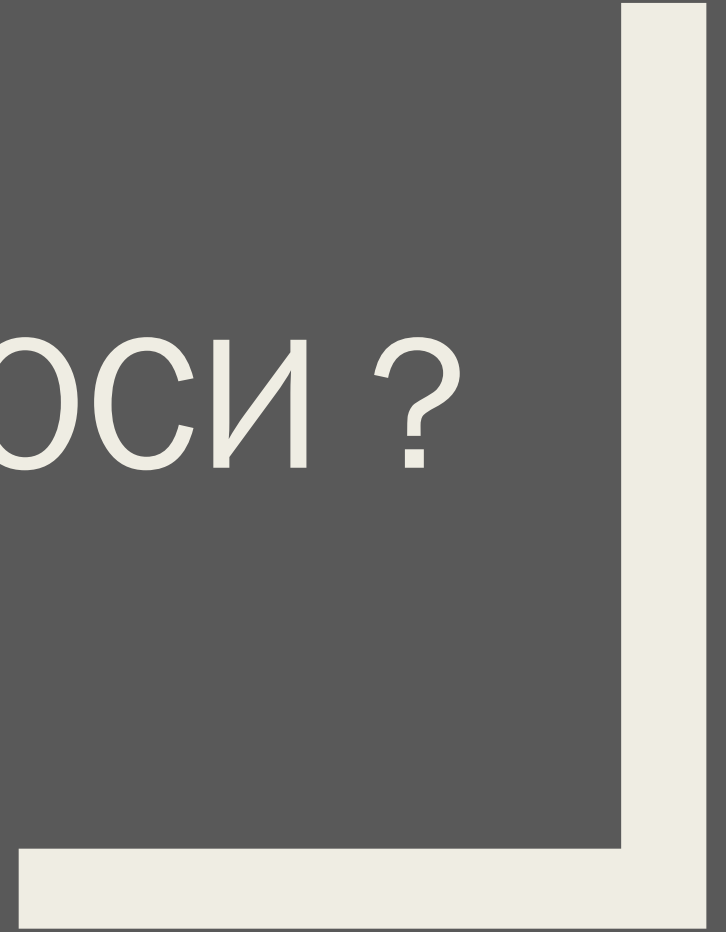
■ Енкапсулация на услуги

- *Много услуги, които в началото не са планирани като SOA, могат да се енкапсулират(обвият) и да станат част от SOA архитектура.*

Подходи на имплементация и технологии

- Web services based on WSDL and SOAP
- Messaging, e.g., with ActiveMQ, JMS, RabbitMQ
- RESTful HTTP, with Representational state transfer (REST) constituting its own constraints-based architectural style
- OPC-UA
- WCF (Microsoft's implementation of Web services, forming a part of WCF)
- Apache Thrift
- SORCER

ВЪПРОСИ ?

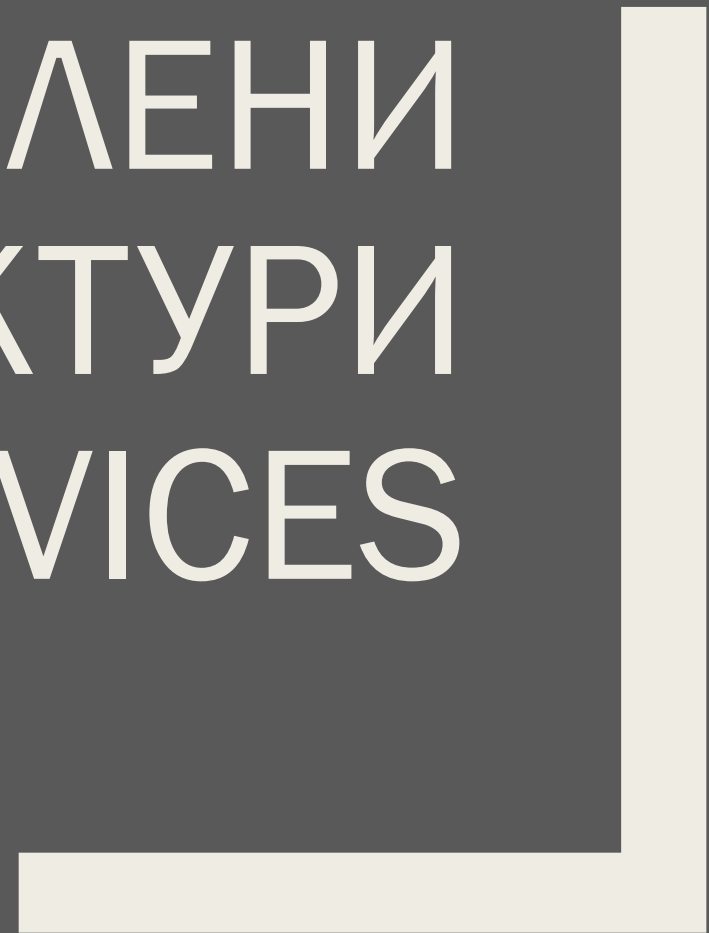




РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

РАЗПРЕДЕЛЕНИ АРХИТЕКТУРИ MICROSERVICES



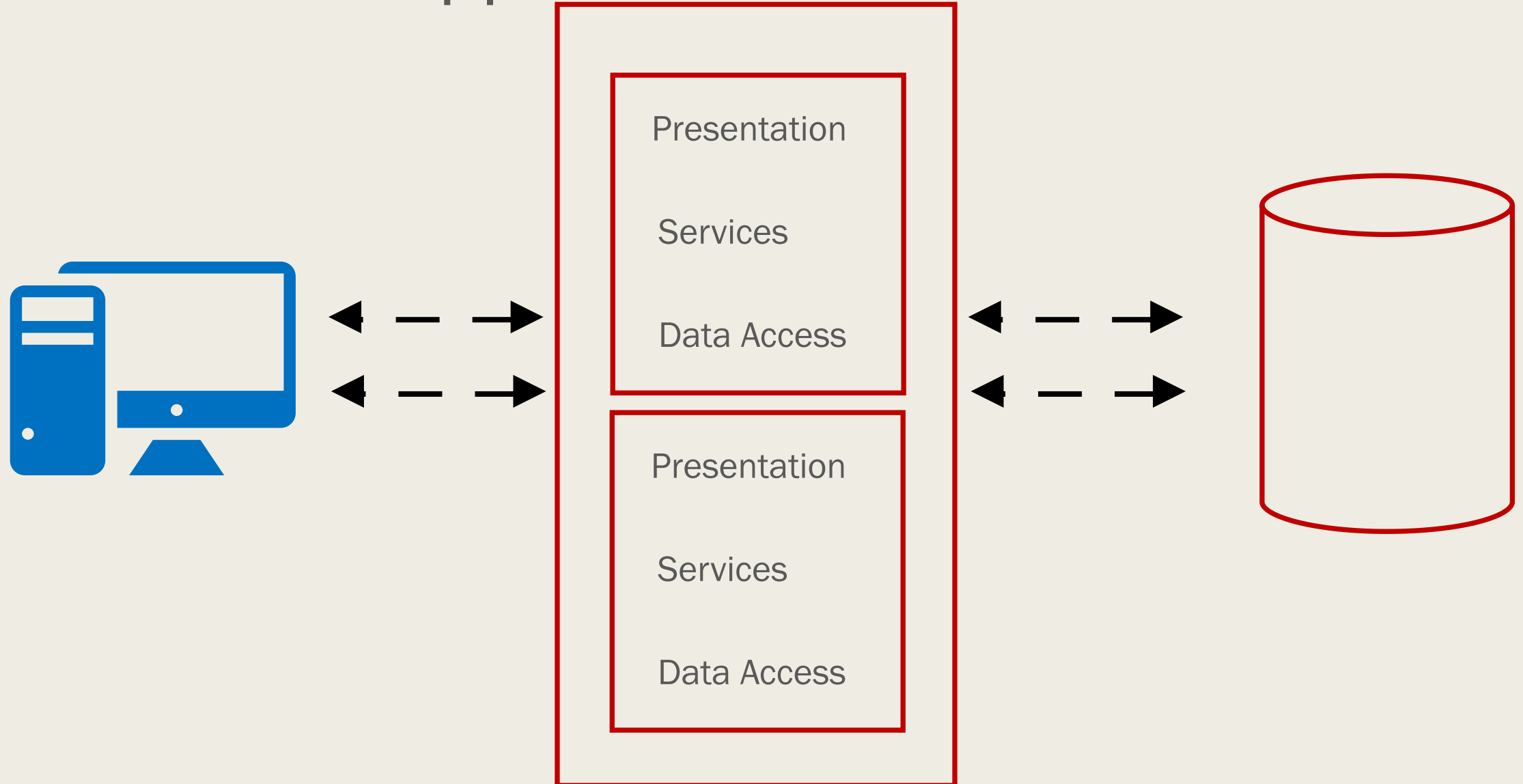
Традиционна Monolithic architecture

- Монолитният софтуер е проектиран да бъде самостоятелен; компонентите на програмата са взаимосвързани и взаимозависими, а не слабо свързани, както е случаят с модулните софтуерни програми. В строго свързана архитектура, всеки компонент и свързаните с него компоненти трябва да присъстват, за да може кодът да бъде изпълнен или компилиран.

Нужна ли е промяна?

- Монолитните приложения могат да се превърнат в "Мега приложение". Ситуация, в която никои разработчик не познава пълната функционалност на приложението
- Ограничена преизползваемост
- Разширението на монолитно приложение е голямо предизвикателство
- По дефиниция монолитните приложения са разработвани само от точно дефиниран технологичен стек. Това от своя страна може силно да лимитира разработката

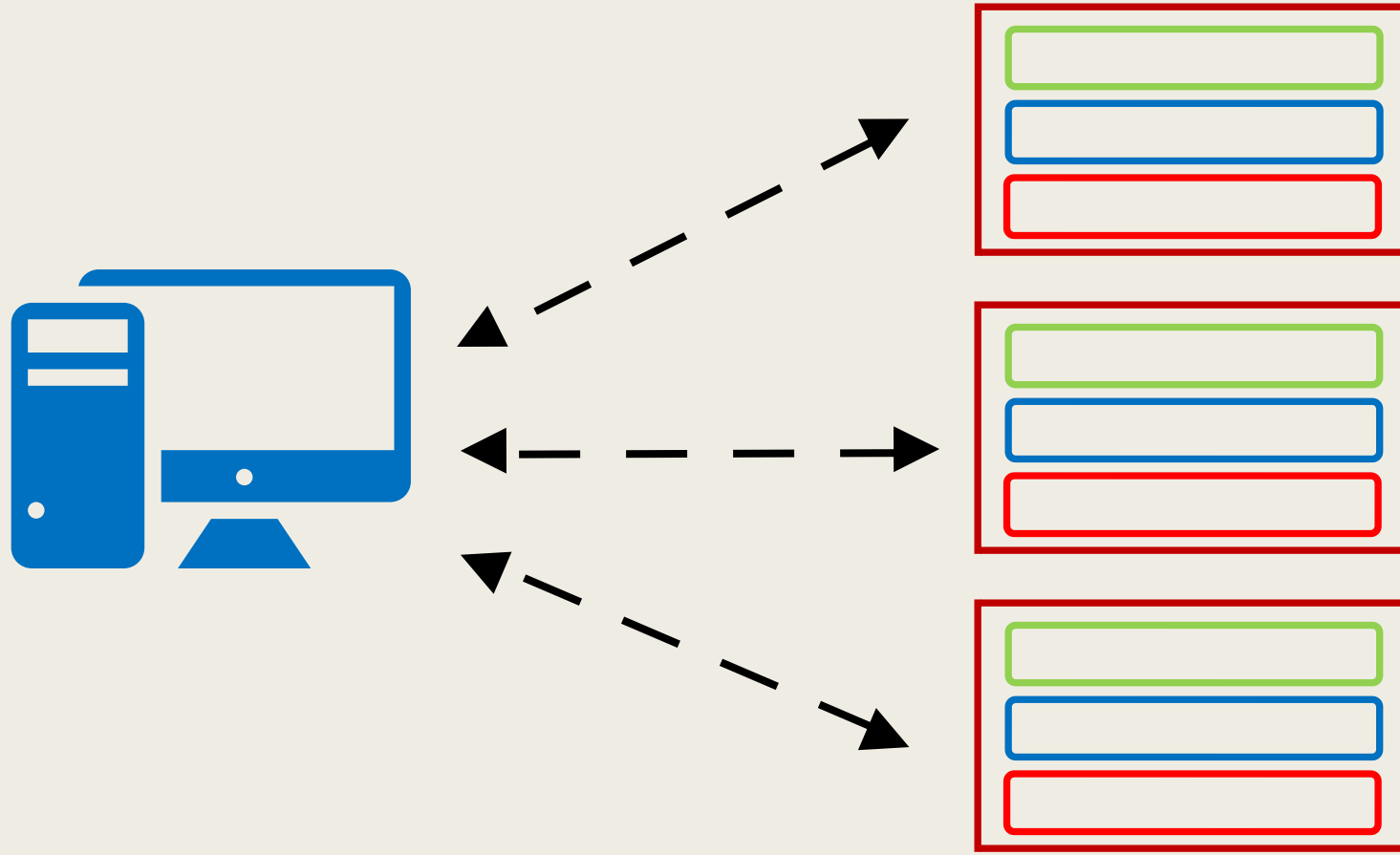
Monolithic application



Microservices

- Microservices architecture е архитектурен стил, който структурира приложение като съвкупност от свободно свързани услуги, които заедно предоставят бизнес логиката на системата.

Microservices application

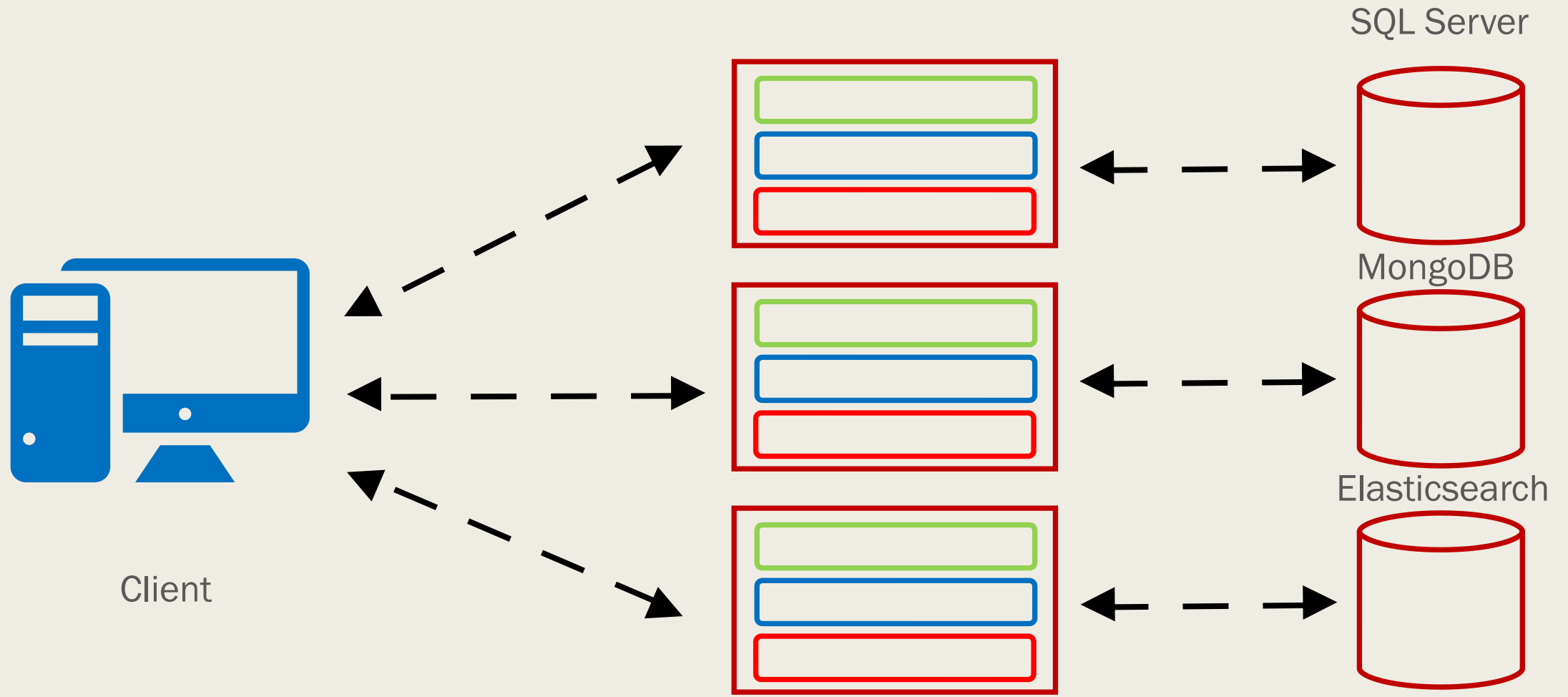


Психология на microservices

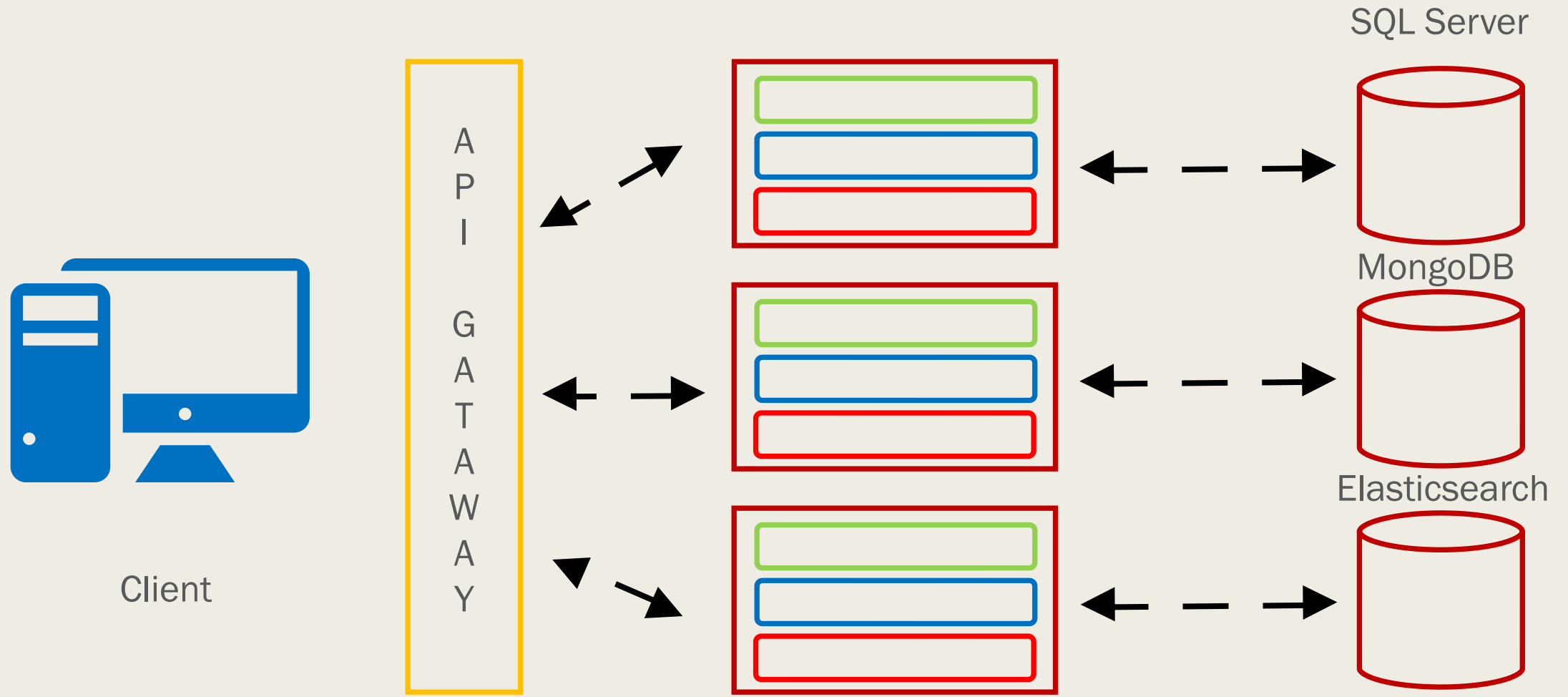
- Услугите трябва да са малки, добре структурирани за да могат да изпълняват само една функция
- Архитектурата трябва да обхваща автоматизираното тестване и внедряване
- Всяка услуга е еластична, композиционна, минимална и пълна.

Direct client communication vs API Gateway

Microservices implementation 1



Microservices implementation 2

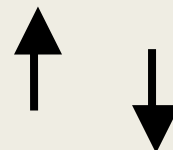


Synchronous vs Asynchronous Microservices communication

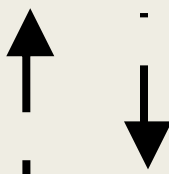
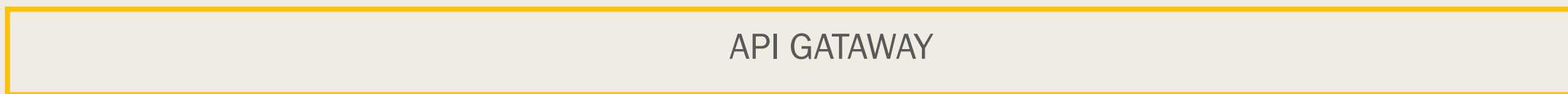


Синхронна комуникация
е възможна но трябва да
се внимава

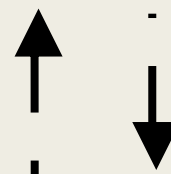
Http Sync



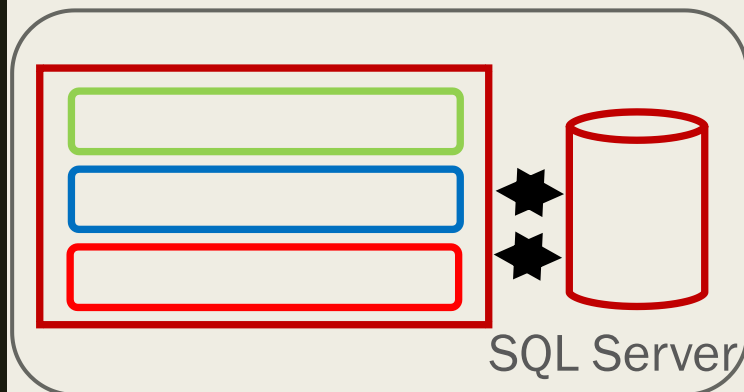
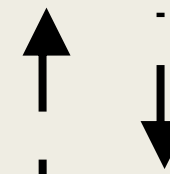
API GATAWAY



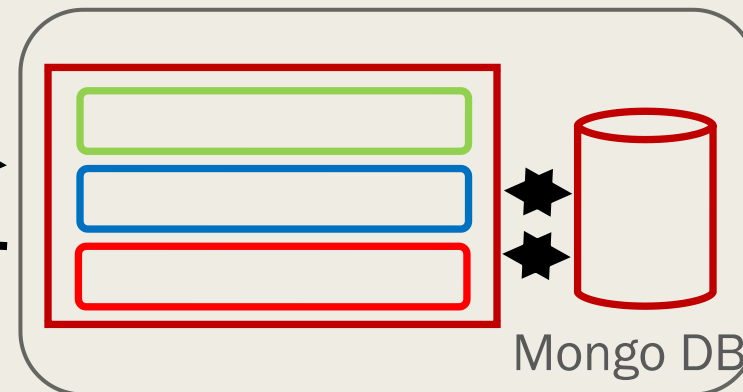
Http Sync



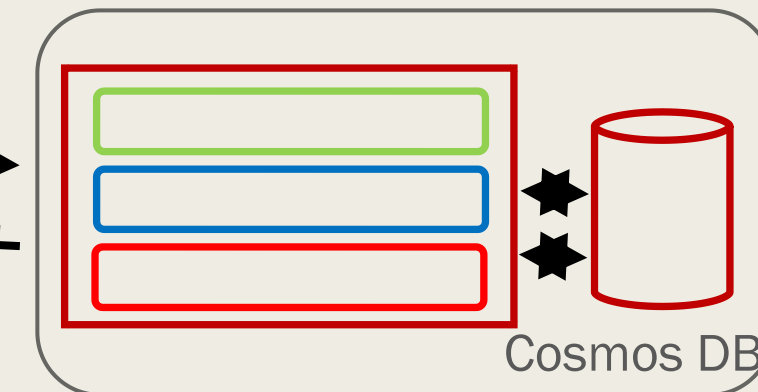
Http Sync



SQL Server



Mongo DB



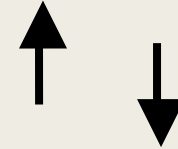
Cosmos DB

Synchronous
communication

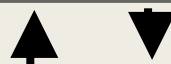
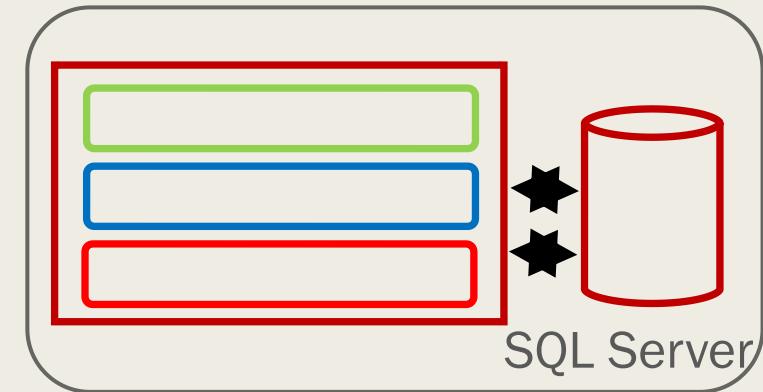
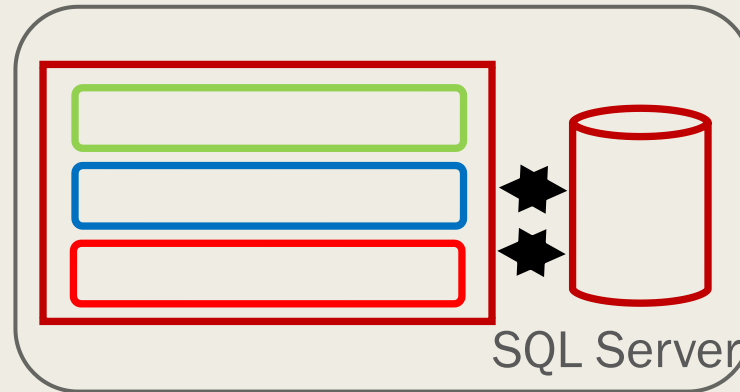
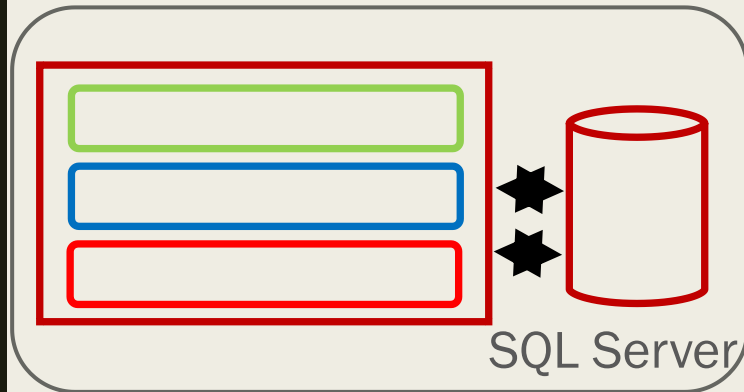
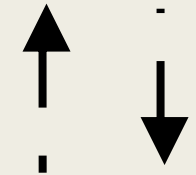
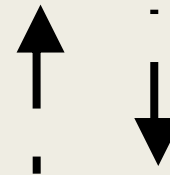
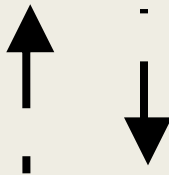


Асинхронната
комуникация използва
асинхронни съобщения

Http Sync



API GATAWAY



Event Bus – Publish/Subscriber channel

Основни предимства

- Възможност за разширение – както хоризонтално така и вертикално
- Модулна структура
- Осигуря процес на непрекъснато обновяване.
 - *DevOps(CI/CD)*

Недостатъци

- Тестването и разпространението е по – трудно
- Асинхронната вътрешната комуникация между отделните услуги е значително по – тежка, от комуникацията между услуги изградени на основата на monolithic architecture
- Преместването на отговорностите(логика) между услуги е по-трудно
 - *Това може да включва комуникация между различни екипи, пренаписване на функционалност на друг език или поставянето на логиката в друга инфраструктура*

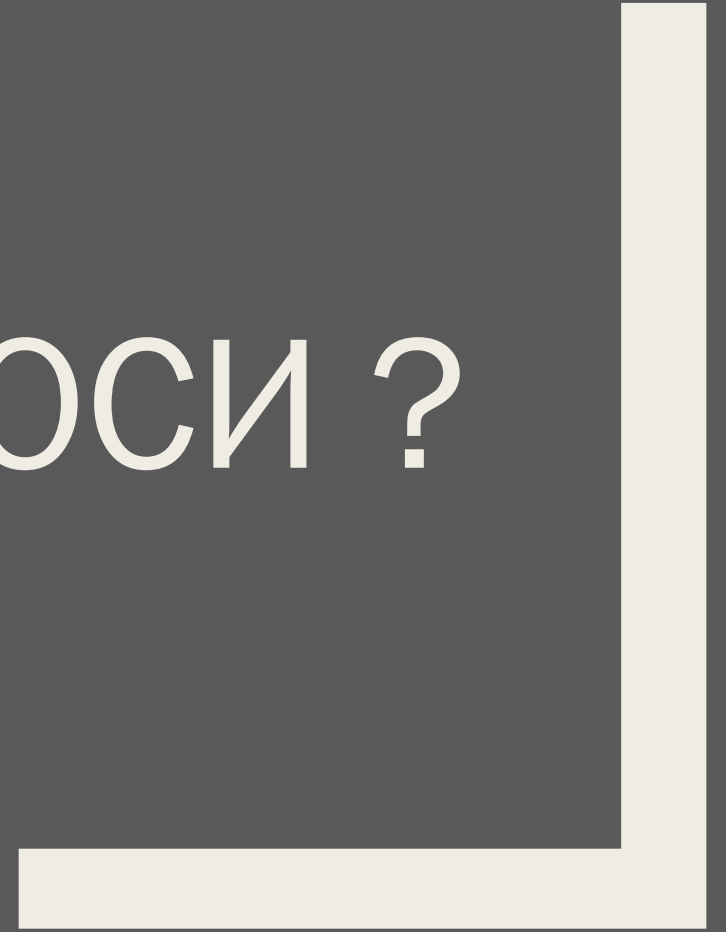
Недостатъци

- Разглеждането размера на услугите като основен структуриращ механизъм може да доведе до твърде много услуги, докато алтернативата на вътрешната модулация може да доведе до по - опростен дизайн

DEMO MICROSERVICES

https://github.com/pkyurkchiev/microservices_skeleton_net-core

ВЪПРОСИ ?



РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

GRPC



Какво е Remote procedure call (RPC)?

- RPC е модел за мрежово програмиране или техника за комуникация между процеси, използвана за комуникация от точка до точка между софтуерните приложения.
- RPC е протокол, който една програма може да използва, за да поиска услуга от друга програма, намираща се на отдалечен компютър, без да е необходимо да разбира подробностите на свързващата ги мрежата.

Модел на изпълнение

- RPC използва client-server модела.
- Начинът, по който работи RPC е: подател или клиент създава заявка под формата на процедура, функция или повикване към отдалечен сървър, сървъра приема заявката и започва обработката и. Когато отдалеченият сървър обработи заявката изпраща отговор към клиента и той продължава своя процес на работа.

- Използването на „lightweight processes“ или „threads“, които споделят едно и също адресно пространство, позволява няколко RPC заявка да се изпълняват едновременно.

Какво е gRPC?

- Представява рамка за разработка на RPC системи. Рамката първоначално е изработена от google за техни вътрешни нужди. Сега проекта е с отводен код.

ОСНОВНИ КОМПОНЕНТИ

- HTTP/2
- Protobuf serialization (proto3)
- Interface Definition Language (IDL)

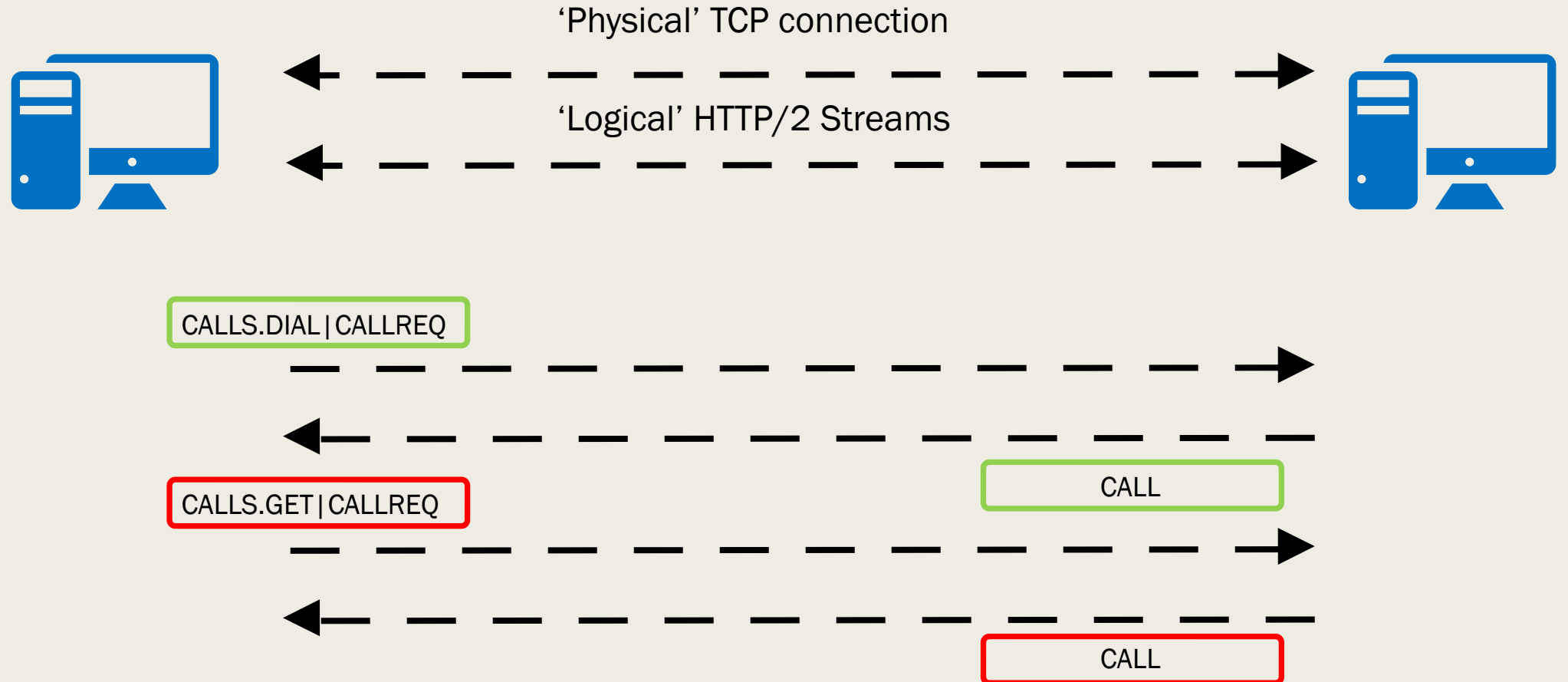
Начини на комуникация

- Unary RPC
- Server streaming RPC
- Client streaming RPC
- Bidirectional streaming RPC
 - Deadlines/Timeouts

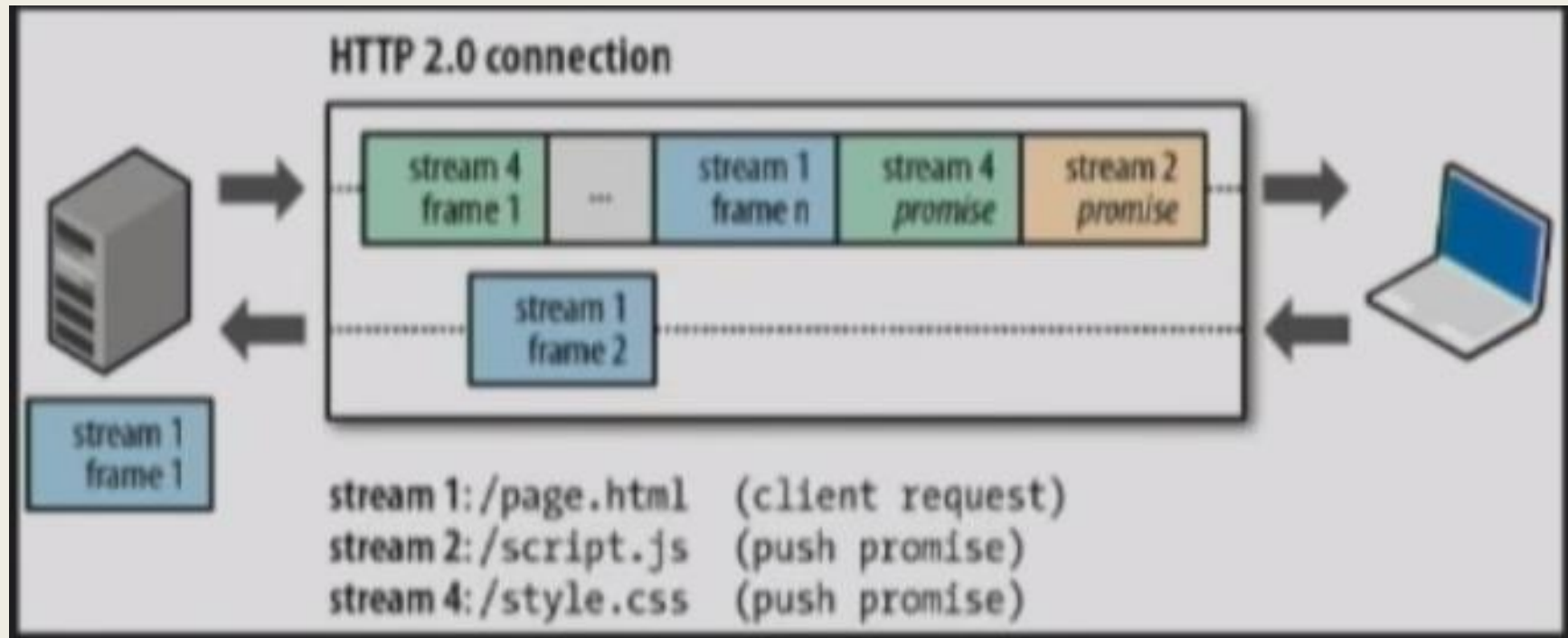
Легенда

- **Unary** - клиента изпраща едно запитване до сървъра и получава един отговор
- **Stream** – клиента изпраща запитване до сървъра и получава стрeam за четене (последователност от съобщения)

GRPC: on the wire 1



GRPC: on the wire 2



Верификация

- gRPC поддържа TLS и токън базирана верификация.
- TLS е криптографски протокол за предаване на информация по компютърната мрежата.

Имплементация

- 3 високо производителни събитийно базирани имплементации
 - C
 - Ruby, Python, node.js, PHP, C#, Objective-C, C++ всички са “C” базирани
 - *Java*
 - Netty + boringSSL via JNI
 - Go
 - Чиста имплементация на Go използваща stdlib crypto/tls пакет

RESTful vs gRPC

- Ресурс базиран
 - IDL опционален
 - Синхронен по подразбиране
 - Unary
 - Перфектен за serverless системи
- API базиран
 - IDL насочен
 - Асинхронен по природа
 - Streaming или Unary
 - Най – важна е производителността

RESTful vs gRPC

- Използвай: когато трябва бързо да се сглоби нещо или ако трябва да е лесно разбираемо от крайния потребител.
- Не използвай: когато се изисква някакъв вид проверка на типа данни или трябва да се намали размера на предавана информация.
- Използвай: когато има комуникация между `microservices` намиращи се в един клъстер или се изисква бързодействие.
- Не използвай: когато трябва да се предава информация между браузър и `back-end` услуги.

DEMO GRPC

examples/GrpcGreeter

ВЪПРОСИ ?





РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

CLOUD COMPUTING



Облак (Cloud)

- Изчисления в облака или така наречените Cloud computing, представлява предоставянето на онлайн услуги, а не на конкретни продукти на потребителите. Това е термин произтичащ от областта на информационните технологии, който означава, че се предлагат споделени ресурси, софтуер и информация по глобалната мрежата.

Началото

- Терминът „cloud computing“ е въведен от Ерик Шмидт от Google през 9 август 2006 г. на конференция, посветена на машините за търсене, на която обяснява един възможен подход към софтуера като услуга (SaaS).

Имаме идея. Как да я реализираме?

- Управление на средата
 - Мрежа
 - Проблеми с дисковото пространство
 - Дисково пространство
 - Памет
 - Ъпдайти
 - Пачове
 - Routers
 - Load balancing
- и много други

Основни характеристики на облака

- Самообслужване при наличие на потребност (on-demand self-service);
- Повсеместен мрежов достъп: достъпност на услугите от всяка точка на света и през всички възможни стандартни устройства, осигуряващи достъп до интернет;
- Ресурсите за обработка и съхранение на данните на всички потребители са балансирано разпределени в рамките на една обща инфраструктура, като за отделните потребители не се заделят точно определени ресурси и мощности;

- Рязко променлива еластичност на търсенето: потребителите могат произволно да увеличават или намалят капацитета на търсеното обслужване;
- Варираща според потреблението цена (pay-per-use): заплащането за обслужване се определя от потреблението на база използваните изчислителни мощности, широколентов достъп и/или компютърна памет.

Основни характеристики на облака

- Много наематели/Multi-tenancy
 - *Едно приложение поддържа множество “класове” от потребители*
 - *Клас потребител има своя база данни, права на достъп*
 - *Различни класове потребители се управляват и дистанцират от приложението*
- Measured service (измерима услуга)
 - *Всеки ресурс се използва с измерима единица*
 - *Контрол на процента заетост*
 - *Оптимизация на заетостта на ресурсите*
 - *Следена и управление на заетостта на ресурсите*

Възможности за доставка на услуги

- Софтуер като услуга (SaaS)
- Инфраструктура като услуга (IaaS)
- Платформа като услуга (PaaS)
- Бизнес процес като услуга (BPaaS)

Софтуер като услуга (SaaS)

- Начин за предоставяне на софтуерни приложения през Интернет. Нарича се още веб-базиран софтуер, софтуер при поискване или хостнат софтуер. Клиентите получават достъп от множество клиентски устройства до различни приложения без да е необходимо да закупуват скъпи софтуерни или хардуерни решения, като същевременно елиминират необходимостта от комплексни управленски процеси.

Инфраструктура като услуга (IaaS)

- Форма на облачни услуги, при която потребителите използват необходимите им компютърни ресурси за поддръжка на оперативните дейности – като дискови масиви, сървъри, мрежи и др. Както и при всяка друга форма на облачна услуга потребителите не трябва да притежават оборудването и активите, а заплащат за използването на услуга, които им се предоставя върху споделена инфраструктура.

Платформа като услуга (PaaS)

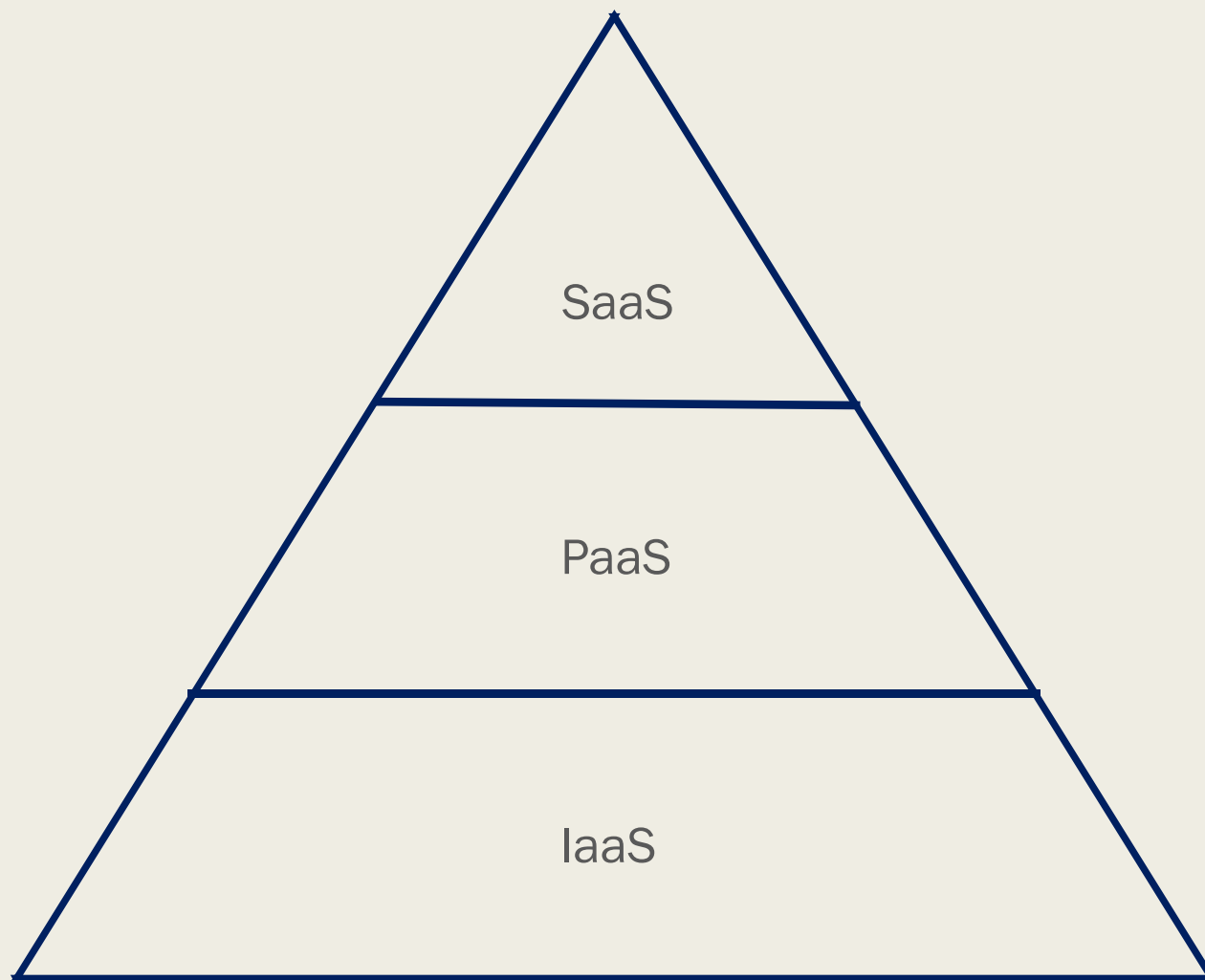
- Осигурява цялата необходима инфраструктура за разработка и стартиране на приложения в Интернет. С други думи, това е начин да се наеме хардуер, операционни системи, дисково пространство и мрежов капацитет през Интернет. По този начин ИТ отделите могат да се съсредоточат върху иновациите, вместо върху създаването и поддръжката на сложна инфраструктура.

Модели на доставка и представители

Google Docs
Freshbooks
Gmail
Salesforce
Office 365

Force.com
APP Engine
Azure, AWS

Rackspace.com
Go Grid
AWS, Azure



End Customers

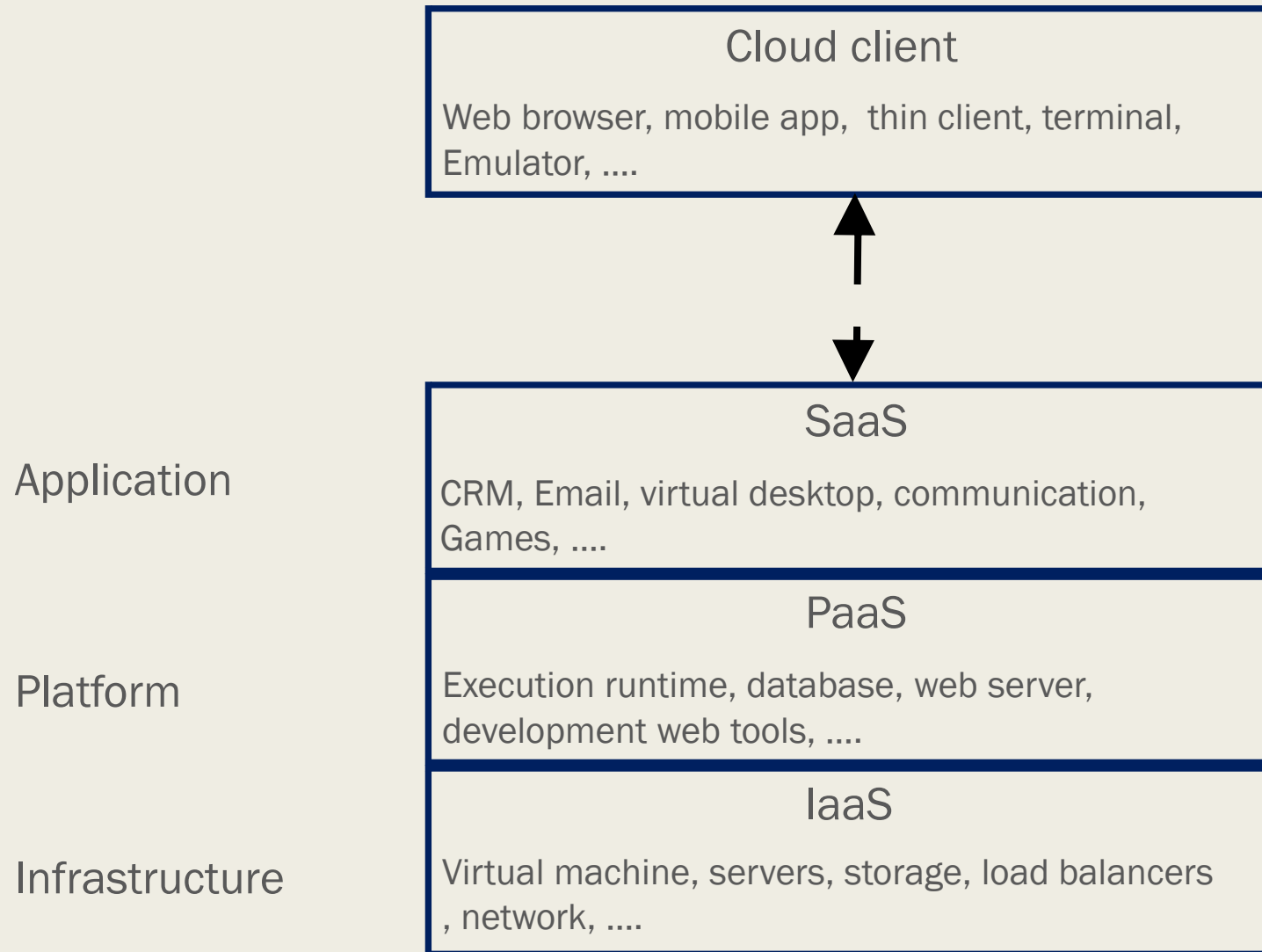
Developers

Sys admins

Бизнес процес като услуга (BPaaS)

- На практика е всеки хоризонтален или вертикален бизнес процес, доставян чрез облачни услуги. Обикновено е автоматизиран, което позволява на компаниите да ползват резултатите без на практика да притежават активите, които ги осигуряват. Решенията се предоставят чрез уеб-базирано интегриране на услуги (Web-centric Service Integration) върху споделена инфраструктура, която може да се ползва от множество клиенти.

Представяне по слоеве

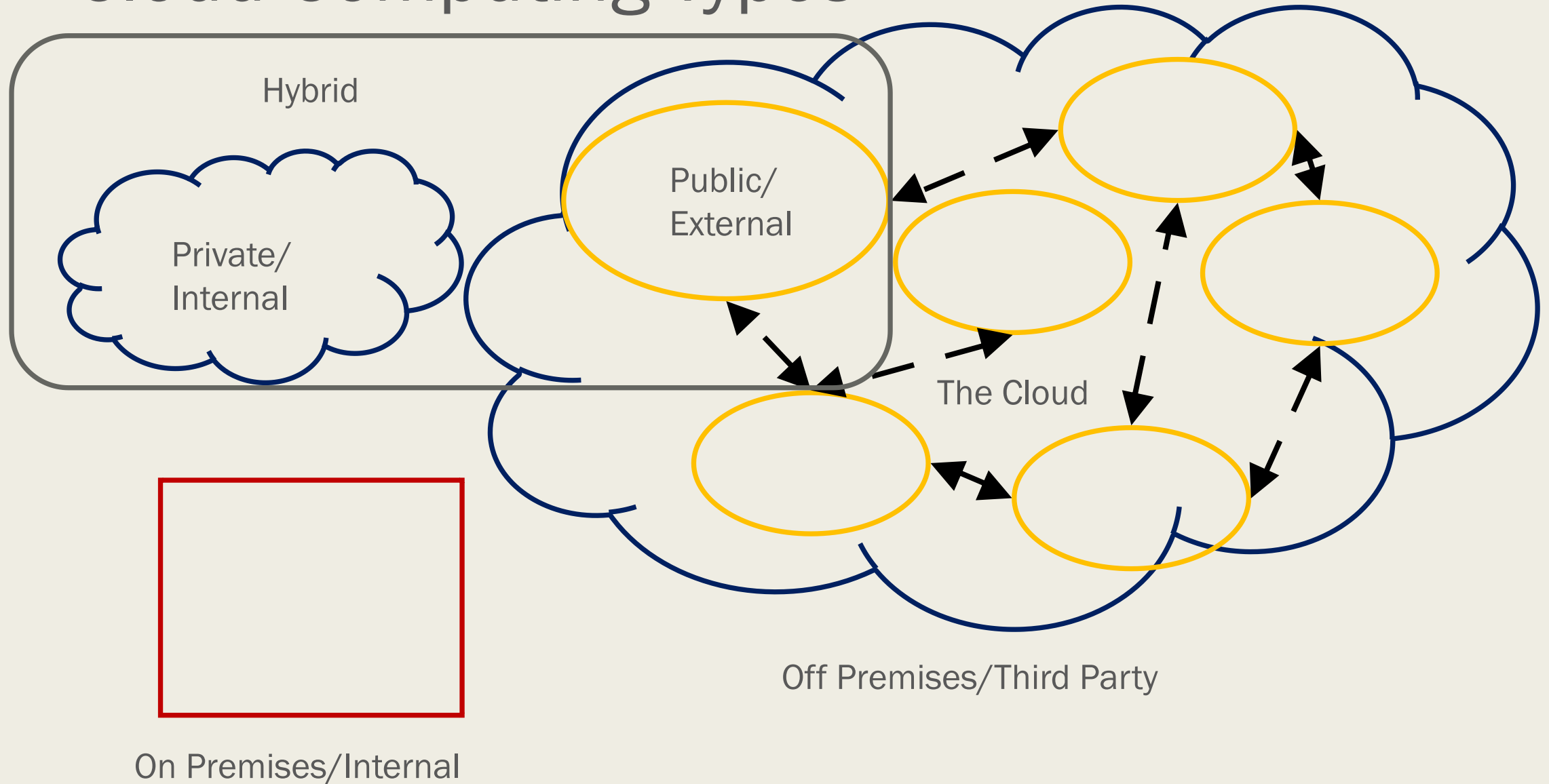


Класифицирани по техния характер

- Частен облак: инфраструктурата на облака се притежава или наема от една организация и се използва само и единствено от нея.
- Общностен облак: инфраструктурата на облака се споделя от няколко организации и служи за поддържането на специфична общност от потребители, които споделят обща мисия, обща политика, общи изисквания към информационната сигурност и др.
- Публичен облак: инфраструктурата на облака се притежава от една организация, която продава „облачни“ услуги на широката аудитория.

- Хибриден облак: инфраструктурата на облака е съчетание на два или повече облака (частен, общностен, публичен), които остават разграничени, въпреки че са свързани посредством стандартизирана или собственическа технология.

Cloud Computing Types



Недостатъци на облачните услуги

- Наличност на обслужването
- Сигурност и неприкосновеност на личните данни
- Поддръжка
- Оперативна съвместимост
- Съгласуваност

Облачни услуги

- Amazon Web Services
- Microsoft Azure
- Alibaba Cloud
- Google Cloud

Microsoft Azure

- Azure е облачна услуга разработена от Microsoft за създаване, предоставяне и управление на приложения и услуги в интернет посредством Microsoft-managed data centers. Azure предоставя софтуер като услуга, платформа като услуга и инфраструктура като услуга, както и поддръжка на много езици за програмиране, инструменти и рамки за разработка.

История

- За първи път през Октомври 2008 година идеята за Azure е представена а през Февруари 2010 е пусната за първи път под името Windows Azure. През 2014 година услугата е преименувана на Microsoft Azure.

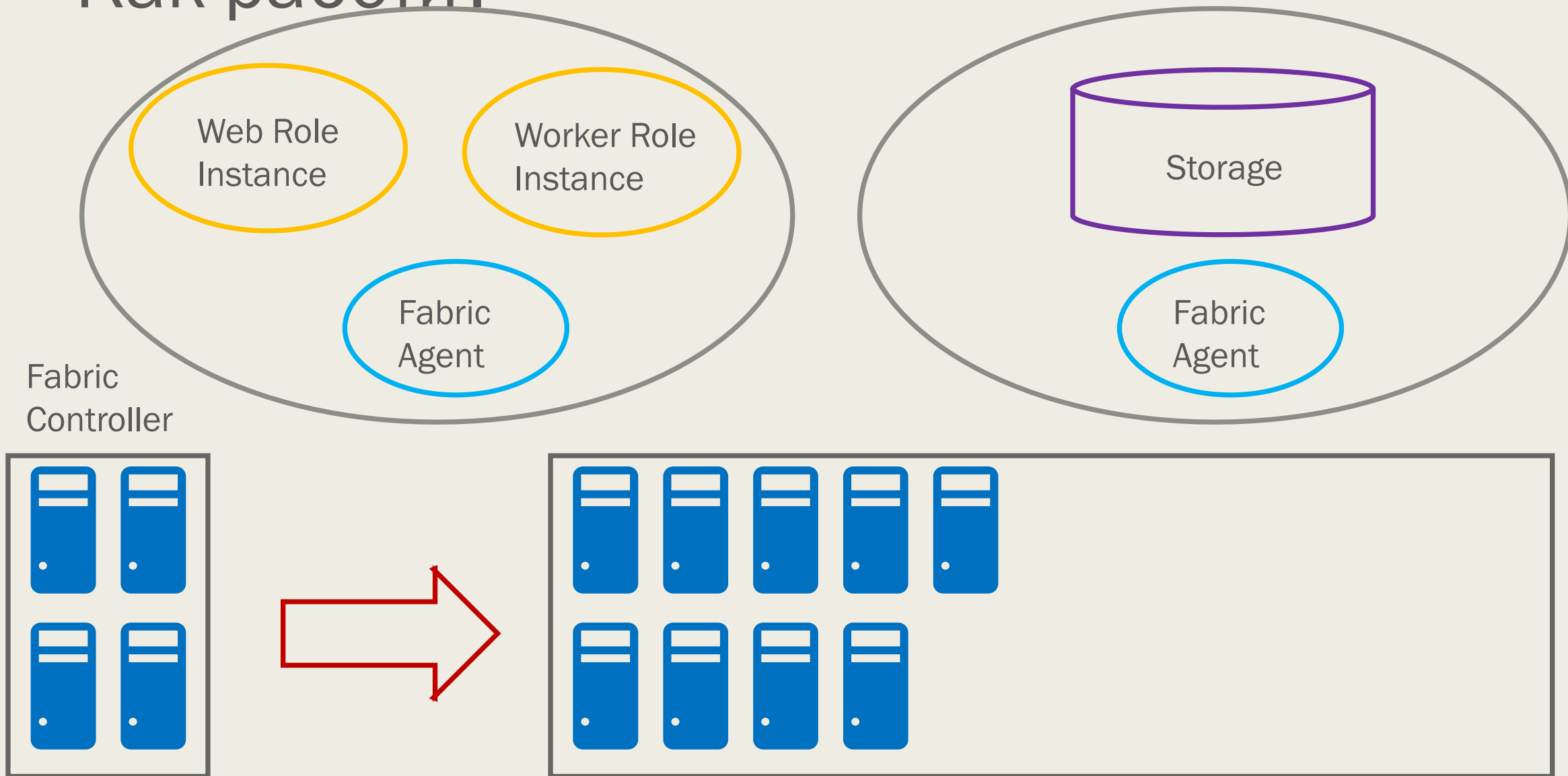
Описание

- Microsoft Azure предоставя над 600 отделни услуги като най – известни сред тях са: Virtual machines, Websites, WebJobs, Storage Services, Table Service, Blob Service, Queue Service, Azure Search, CosmoseDB, Redis Cache, Microsoft Azure Machine Learning (Azure ML). Azure е достъпен в повече от 34 региона в света.
- Microsoft Azure използва специализирана операционна система, работеща като “fabric layer”, който представлява клъстер от машина подържан от Microsoft`s data centers. Microsoft Azure бива описван като “Cloud layer” поставен върху n на брой Windows Server системи, използващи Windows Server 2008 със обработена версия на Hyper-V.

Как Microsoft решават нашите проблеми?

- Съвкупност от виртуални машини
- Управлявани от fabric controller

Как работи?



Web Role vs Worker Role

Web and Worker Role

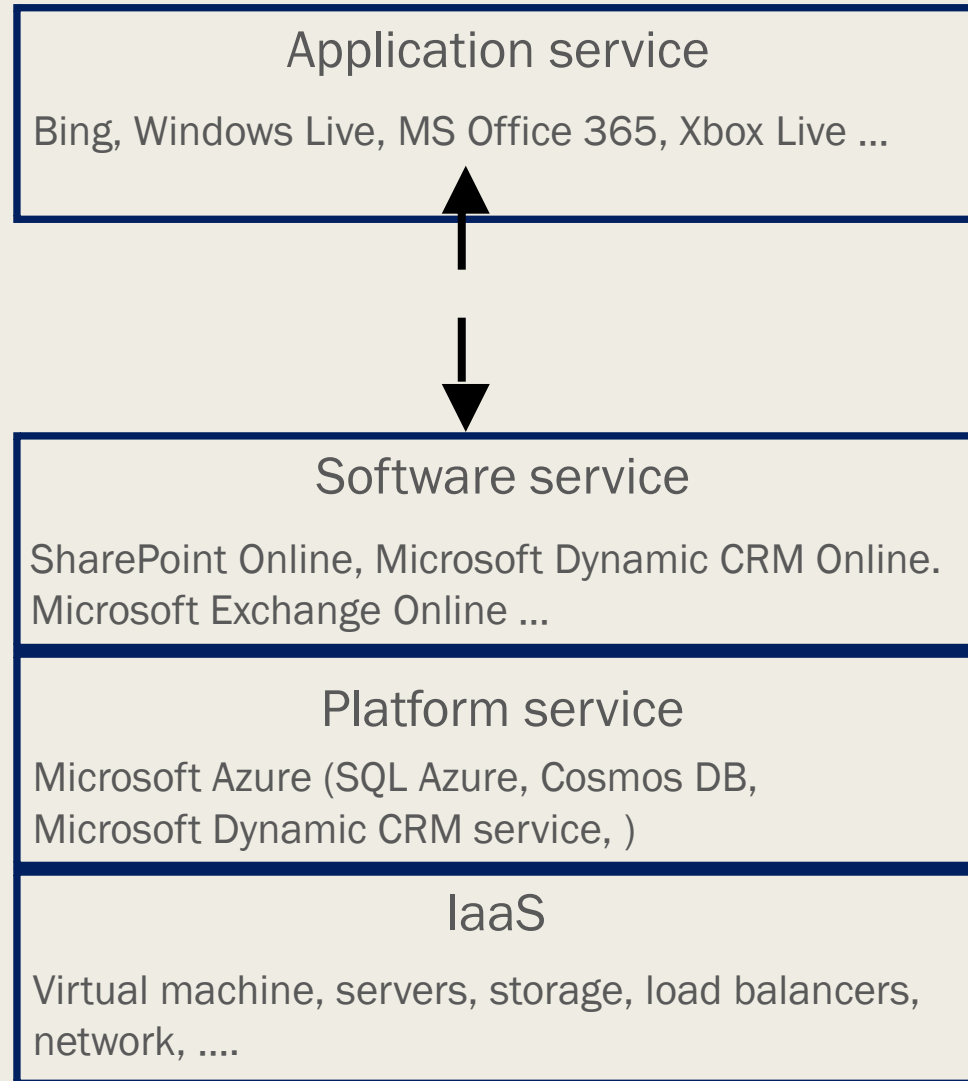
- Web Role

- *Автоматично хоства приложението ви в IIS...*

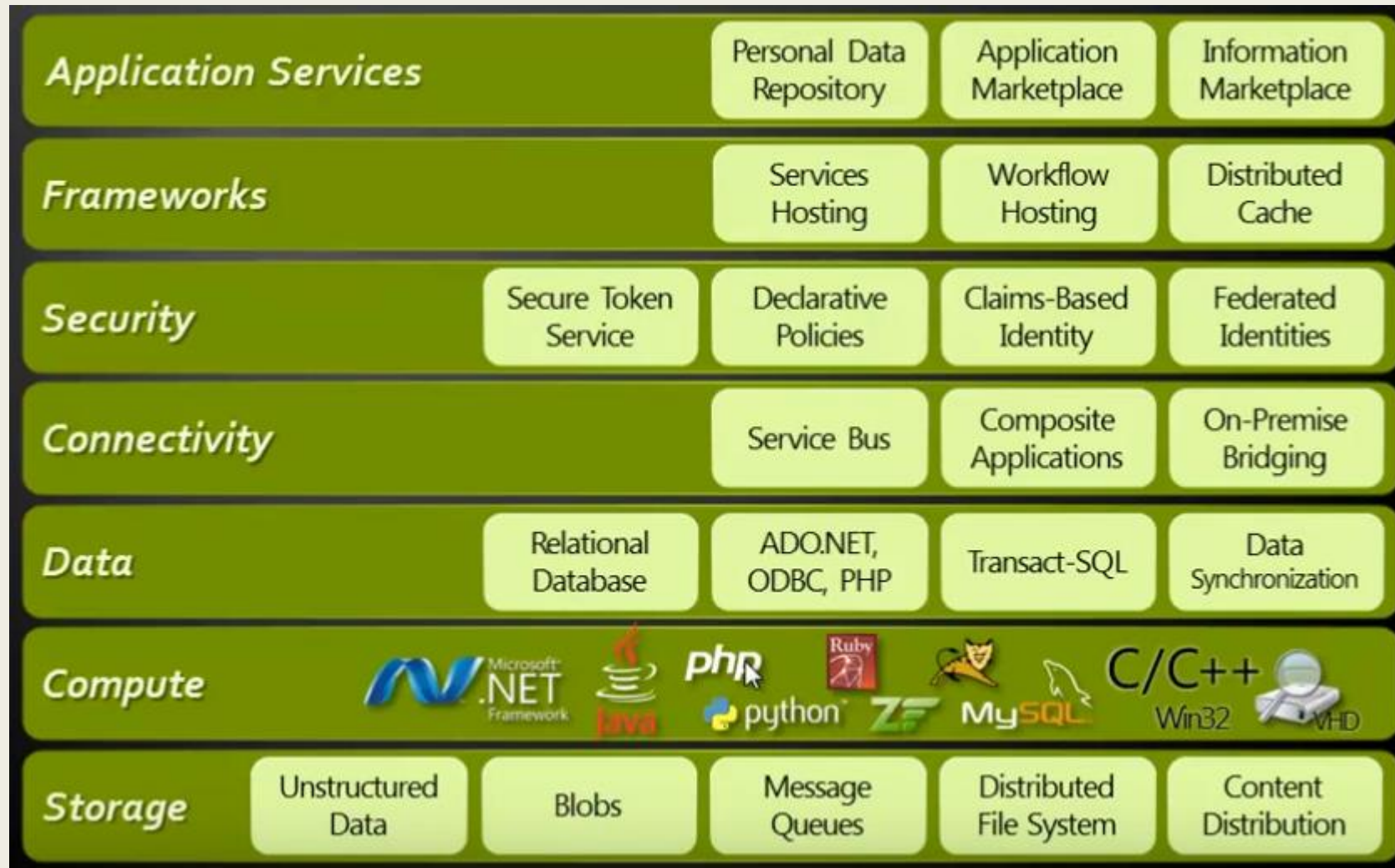
- Worker Role

- *Не използва IIS, а стартира приложението като standalone*

Категории услуги в Microsoft Cloud



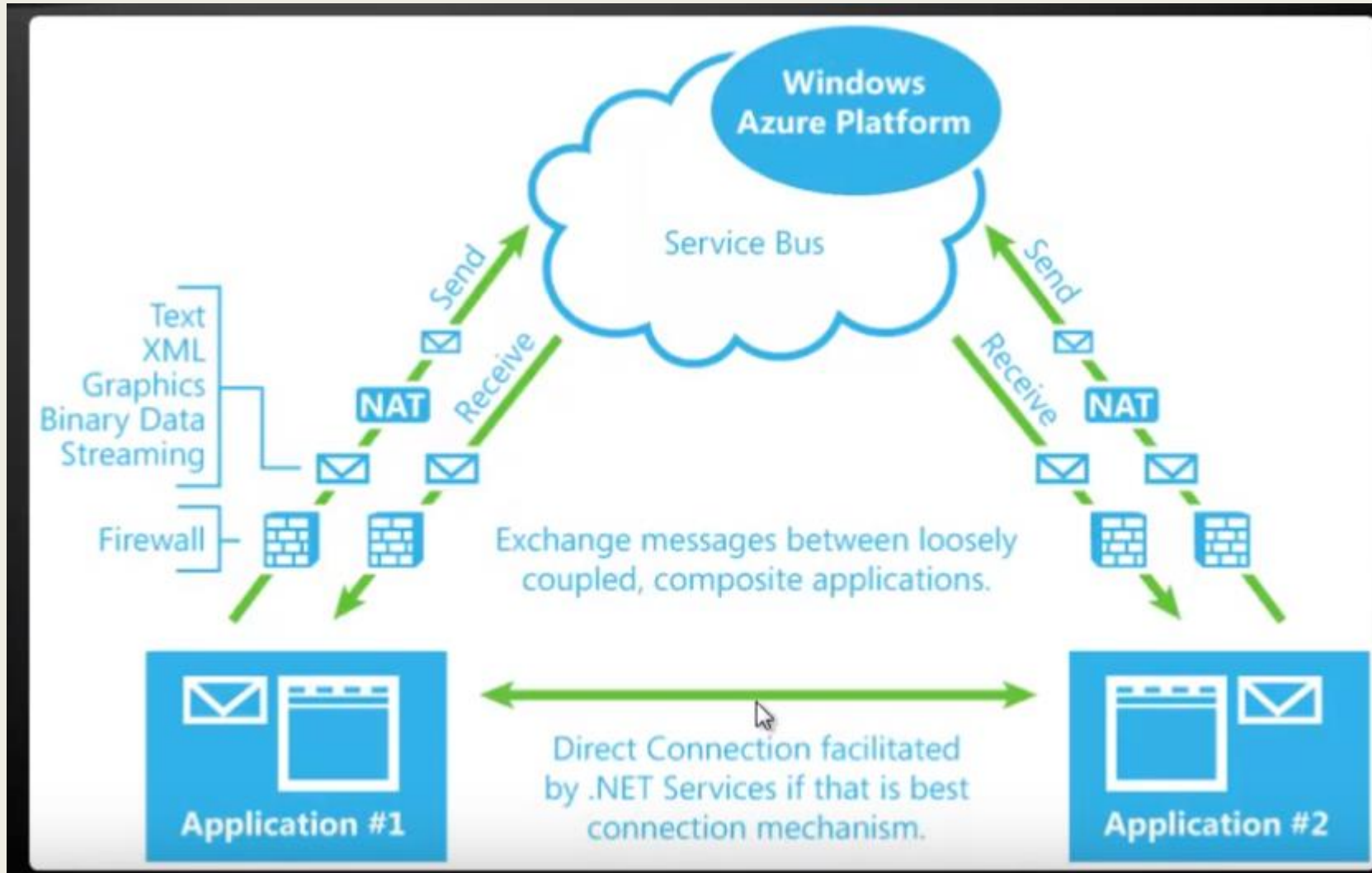
Microsoft Azure Platform



Microsoft Azure Platform



Azure Service Bus



Amazon Web Services (AWS)

- AWS е колекция от отдалечени компютърни услуги (remote computing services), наричани още уеб услуги (web services), които съставят една cloud computing платформа, предлагана от Amazon.com.

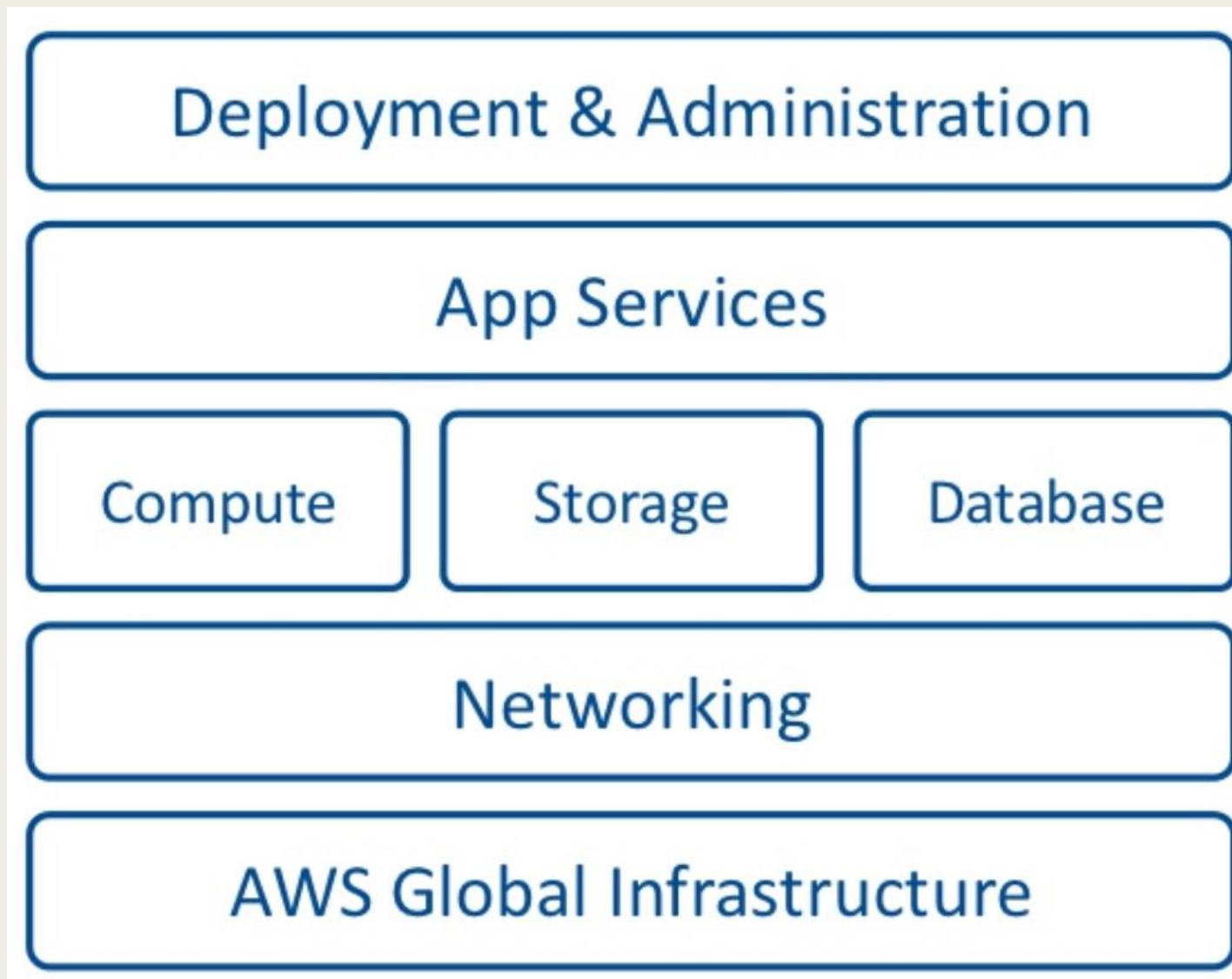
История

- За първи път услугата е представена през 2006 година. Голяма част от набора с услуги е достъпен директно от клиентите. Amazon Web Services предоставя достъп посредством HTTP, използвайки REST архитектурата и SOAP протокол. През юни 2007 година, услугата е използва от повече от 180 000 програмисти.

Описание

- Основните услуги са Amazon EC2 и Amazon S3. Тези продукти се предлагат на пазара като услуга за предоставяне на изчислителни мощности по-бързо и по - евтино отколкото ако клиентската фирма трябва сама да създаде физическа сървърна група. AWS се намира в 11 географски региона. Всеки регион има множество достъпни зони, които са обособени центрове за данни, предоставящи AWS услуги. Достъпните зони са изолирани една от друга, за да се предотврати прекъсване на тяхното разпространение между зоните.

Структура



Global Infrastructure

■ Региони

Независима колекция от AWS ресурси

■ Зони

Физически разделени с типичен столичен район



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Networking

- Директна връзка
- VPN Connection
- Virtual Private cloud
- Route 53

Широко достъпна и
разширяема домейн
система



Deployment & Administration

App Services

Compute

Storage

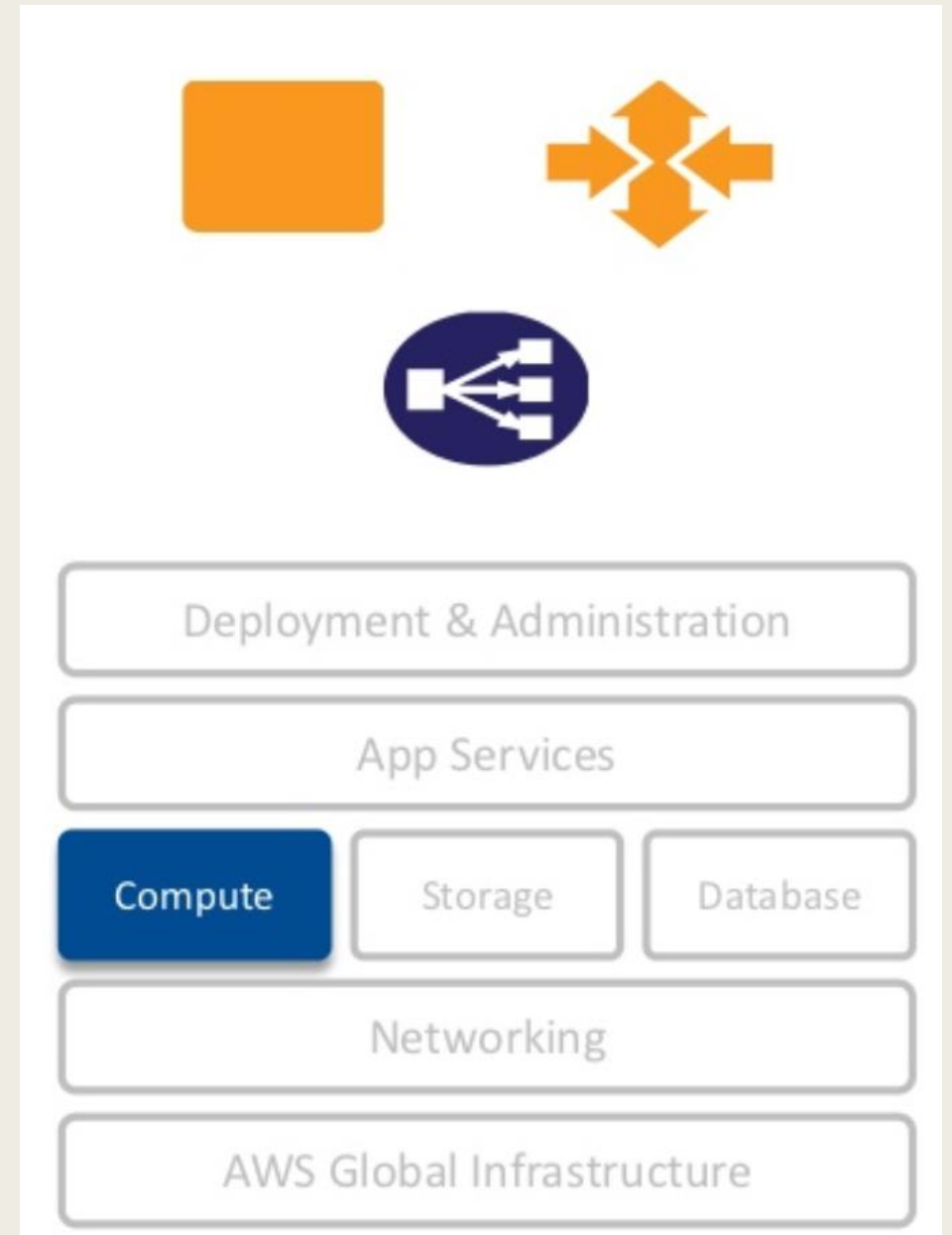
Database

Networking

AWS Global Infrastructure

Compute

- Elastic Compute Cloud (EC2) – CPU, RAM, storage ...
- Автоматично расширение
- Elastic Load Balancing



Storage

■ S3

Място за съхранение, до 5TB големина на обекти

■ Elastic Block Store

Високо скоростен блок за съхранение от 1GB до 1TB



Deployment & Administration

App Services

Compute

Storage

Database

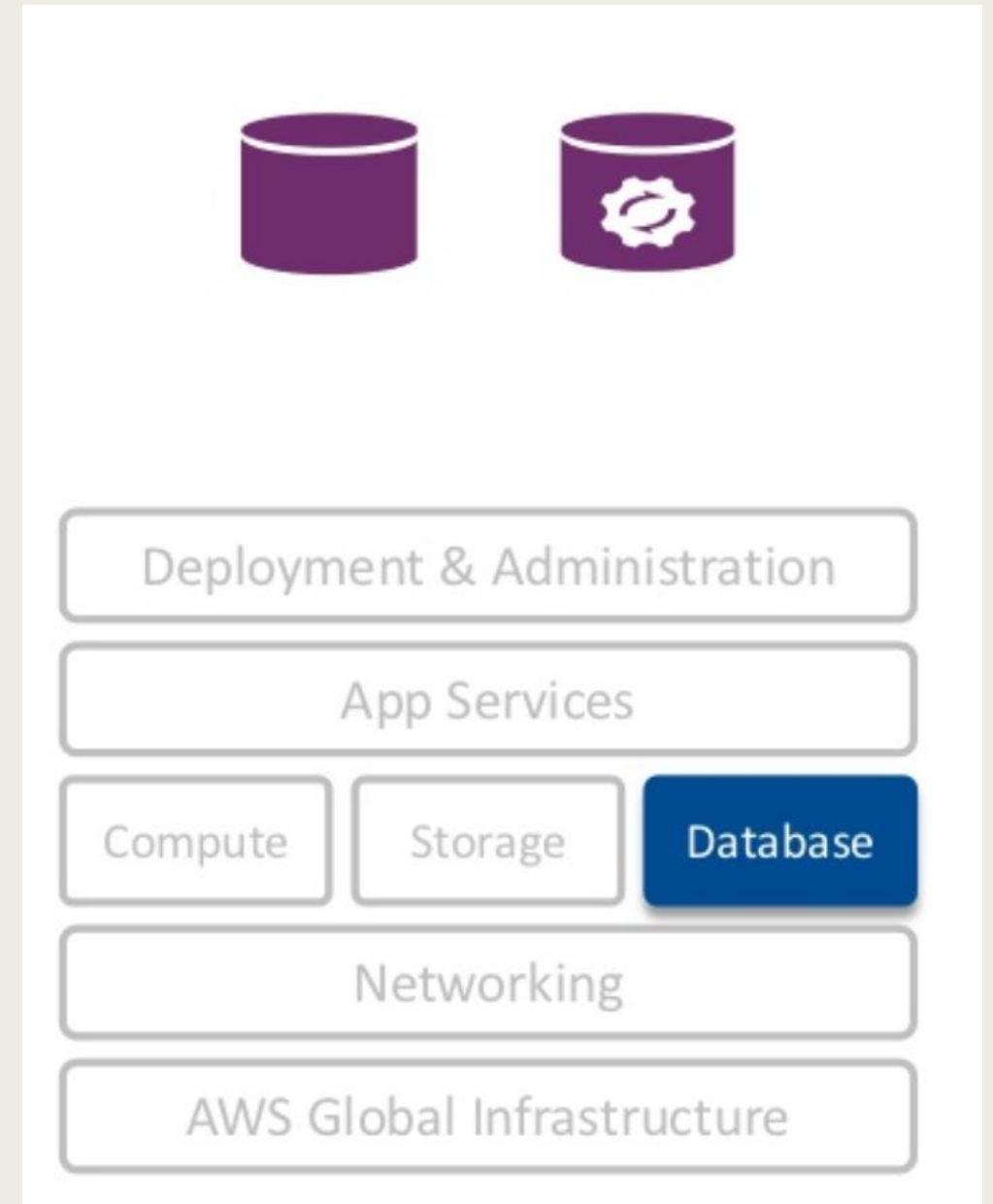
Networking

AWS Global Infrastructure

Database

- Включва различни версии на релационни бази данни
- DynamoDB

Разработка на Amazon от тип NoSQL



Видове бази данни

Self-Managed



Database Server on Amazon EC2

Your choice of database running on Amazon EC2

Bring Your Own License (BYOL)

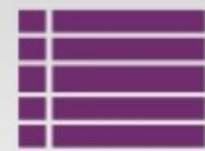
Managed Databases



Amazon Relational Database Service (RDS)

Oracle or MySQL offered as a service

Flexible Licensing: BYOL or License Included



Amazon SimpleDB NoSQL Database

Non-relational model; indices and queries

Zero admin overhead

Application services

- Amazon SQS

Опашка за събития

- Simple Workflow

Координатор на процеси

- Amazon SES

Емайл услуга

- Cloud Front

Улеснява разпределението на информация до потребителя



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Deployment and Administration

- Среди за разработка
C#, Ruby, Java ...
 - Система за идентификация
 - Elastic Beanstalk
- One-click deployment



Deployment & Administration

App Services

Compute

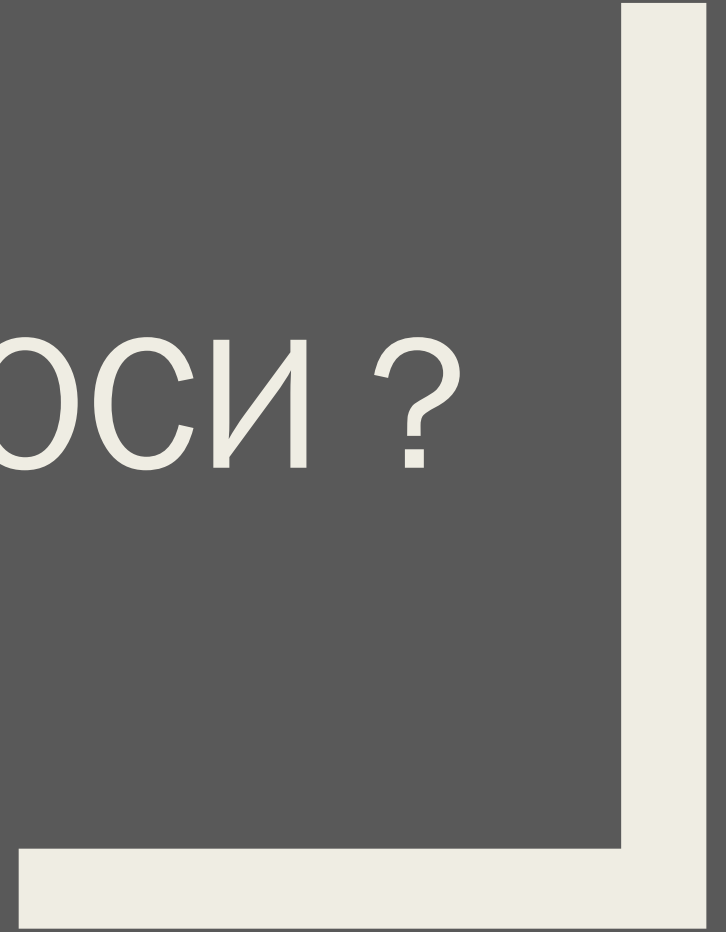
Storage

Database

Networking

AWS Global Infrastructure

ВЪПРОСИ ?





РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

CLOUD DATABASES



Какво са облачно базираните бази данни?

- Cloud database е база данни работеща на облачно базирана платформа, достъпа до нея се осъществява чрез уеб услуги. База данни като услуга се грижи за мащабируемостта и достъпността до данните.

Типове на база модел на работа

- SQL база данни
- NoSQL база данни

Типове на база модел на предоставяне

- Вирутални машини
- База данни като услуга(Database-as-a-service (DBaaS))

Виртуална машина

- Облачните платформи позволяват на потребителите да купуват копия на виртуални машини за ограничен период от време, на които могат да се стартират база данни. Потребителите могат да избират между добавянето на собствени копия или използването на вече готови копия на системи с оптимизирани версии на база данни.

Примери

- SQL Server
- Oracle SQL
- MySQL
- MongoDB
- Cassandra database

База данни като услуга(Database-as-a-service (DBaaS))

- С база данни като услуга собствениците на приложения не са длъжни сами да инсталират и поддържат базата данни. Вместо това, доставчикът на услуги за база данни поема отговорността за инсталирането и поддръжката на базата данни, а собствениците на приложения се таксуват в зависимост от използването на услугата.

Примери

- Amazon Relational Database Service
- Clustrix Database as a Service
- EnterpriseDB Postgres Plus Cloud Database
- Microsoft Azure SQL Database
- **Amazon DynamoDB**
- Amazon SimpleDB
- Azure Cosmos DB
- Cloudbant Data Layer
- **Firebase database**

Amazon DynamoDB

- Amazon DynamoDB е напълно управляема NoSQL база данни услуга, която се предлага от Amazon.com като част от Amazon Web Service portfolio;
- DynamoDB е NoSQL документно – ориентирана база данни от тип key – value;
- За пръв път Amazon DynamoDB е представена през 2012 година от Amazon CTO Werner Vogels;
- DynamoDB използва асинхронни репликации между голям брой дата центрове за висока издръжливост и достъпност;

Производительность и параметры

- Requests and throttling
- Errors: ConditionalCheckFailedRequests, UserErrors, SystemErrors
- Metrics related to Global Secondary Index creation

Езици и рамки за разработка подържани от DynamoDB

- Java, Node.js, Go, C# .NET, Perl, PHP, Python, Ruby, Haskell and Erlang.

```
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.DocumentModel;  
using Amazon.Runtime;  
  
private static AmazonDynamoDBClient client = new  
AmazonDynamoDBClient();  
private static string tableName = "ProductCatalog";
```

Връзка с AWS SDK и създаване на обект DB

Създаване на таблица - Create

```
var request = new CreateTableRequest
{
    AttributeDefinitions = new List<AttributeDefinition>()
    {
        new AttributeDefinition
        {
            AttributeName = "Id",
            AttributeType = "N"
        },
        new AttributeDefinition
        {
            AttributeName = "ReplyDateTime",
            AttributeType = "N"
        }
    },
};
```

Деклариране на таблица

```
KeySchema = new List<KeySchemaElement>
{
    new KeySchemaElement
    {
        AttributeName = "Id",
        KeyType = "HASH" //Partition key
    },
    new KeySchemaElement
    {
        AttributeName = "ReplyDateTime",
        KeyType = "RANGE" //Sort key
    }
},
```

Деклариране на таблица

```
ProvisionedThroughput = new ProvisionedThroughput
{
    ReadCapacityUnits = 5,
    WriteCapacityUnits = 6
},
TableName = "ProductCatalog"
};

var response = client.CreateTable(request);
```

Създаване на таблица

Създаване на запис - Put

```
Table productCatalog = Table.LoadTable(client, tableName);
private static int sampleBookId = 555;
var book = new Document();
    book["Id"] = sampleBookId;
    book["Title"] = "Book " + sampleBookId;
    book["Price"] = 19.99;
    book["ISBN"] = "111-1111111111";
    book["Authors"] = new List<string>
        { "Author 1", "Author 2", "Author 3" };
    book["PageCount"] = 500;
    book["Dimensions"] = "8.5x11x.5";
    book["InPublication"] = new DynamoDBBool(true);
    book["InStock"] = new DynamoDBBool(false);
    book["QuantityOnHand"] = 0;
productCatalog.PutItem(book);
```

Създаване на запис

Извличане на запис - Get

```
Table productCatalog = Table.LoadTable(client, tableName);
GetItemOperationConfig config = new GetItemOperationConfig
{
    AttributesToGet = new List<string>
        { "Id", "ISBN", "Title", "Authors", "Price" },
    ConsistentRead = true
};
Document document = productCatalog.GetItem(sampleBookId,
config);
Console.WriteLine("RetrieveBook: Printing book retrieved...");
```

Извличане

Промяна на запис - Update

```
Table productCatalog = Table.LoadTable(client, tableName);  
int partitionKey = sampleBookId;  
var book = new Document();  
    book["Id"] = partitionKey;  
    book["Price"] = 29.99;  
  
// For conditional price update, creating a condition  
expression.  
Expression expr = new Expression();  
expr.ExpressionStatement = "Price = :val";  
expr.ExpressionAttributeValues[":val"] = 19.00;
```

Декларация

```
// Optional parameters.  
UpdateItemOperationConfig config = new  
UpdateItemOperationConfig  
{  
    ConditionalExpression = expr,  
    ReturnValues = ReturnValues.AllNewAttributes  
};  
Document updatedBook = productCatalog.UpdateItem(book,  
config);
```

Промяна и извличане

Премахване на запис - Delete


```
Table productCatalog = Table.LoadTable(client, tableName);  
// Optional configuration.  
DeleteItemOperationConfig config = new  
DeleteItemOperationConfig  
{  
    // Return the deleted item.  
    ReturnValues = ReturnValues.AllOldAttributes  
};  
Document document = productCatalog.DeleteItem(sampleBookId,  
config);
```

Изтриване

Търсене – Query and Scan

```
Table table = Table.LoadTable(client, "Reply");
DateTime twoWeeksAgoDate = DateTime.UtcNow -
    TimeSpan.FromDays(15);
QueryOperationConfig config = new QueryOperationConfig()
{
    HashKey = "DynamoDB Thread 2", //Partition key
    AttributesToGet = new List<string>
        { "Subject", "ReplyDateTime", "PostedBy" },
    ConsistentRead = true,
    Filter = new RangeFilter(QueryOperator.GreaterThan,
        twoWeeksAgoDate)
};

Search search = table.Query(config);
```

Извличане на резултати - Query

```
string tableName = "Thread";  
Table ThreadTable = Table.LoadTable(client, tableName);  
  
ScanFilter scanFilter = new ScanFilter();  
scanFilter.AddCondition("ForumId", ScanOperator.Equal, 101);  
scanFilter.AddCondition("Tags", ScanOperator.Contains,  
"sortkey");  
  
Search search = ThreadTable.Scan(scanFilter);
```

Извличане на резултати - Scan

Много моделни бази данни (Multi-model database)

- Много моделните бази данни са бази от данни, които могат да съхраняват, индексират и извличат информация от повече от един модел на данни. До преди няколко години базите от данни можеха да работят само с един модел на данни: relational database, document-oriented database, graph database or triplestore. Бази от данни, които могат да комбинират няколко от тези модели се наричат много моделни.

Много моделни бази данни

- **ArangoDB** – document (JSON), graph, key-value
- **Cosmos DB** – document (JSON), key-value, SQL
- **Couchbase** – document (JSON), key-value, N1QL
- **Oracle Database** – relational, document (JSON and XML), graph triplestore, property graph, key-value, objects
- **Redis** – key-value, document (JSON), property graph, streaming, time-series

Microsoft Azure Cosmos DB

- Базата данни е част от Azure;
- Azure Cosmos DB представлява много моделна база данни;
- Базата данни не разполага с определена схема;
- Azure Cosmos DB разполага с автоматичен механизъм за индексиране на информацията;
- Базата данни представлява backend as a service;

Azure Cosmos DB – обединява следните NoSQL и SQL APIs

- SQL APIs
- Cassandra APIs
- MongoDB APIs
- Gremlin APIs
- Azure Tables Storage APIs
- Etcd APIs

DEMO AZURE COSMOS DB

`examples/AzureCosmosDB.DatabaseManagement`

Google Firebase и Firebase database

- Firebase е мобилна и уеб платформа за разработка на приложения собственост на Google;
- Firebase database е NoSQL база данни от тип key – value;
- Представява база данни като услуга, част от Google cloud;
- Предоставят се два типа бази данни realtime database and backend as a service;

C# sdk

- FireSharp
 - *Install-Package FireSharp*
- FirebaseSharp
 - *Install-Package FirebaseSharp*
- FirebaseDatabase.net
 - *Install-Package FirebaseDatabase.net*

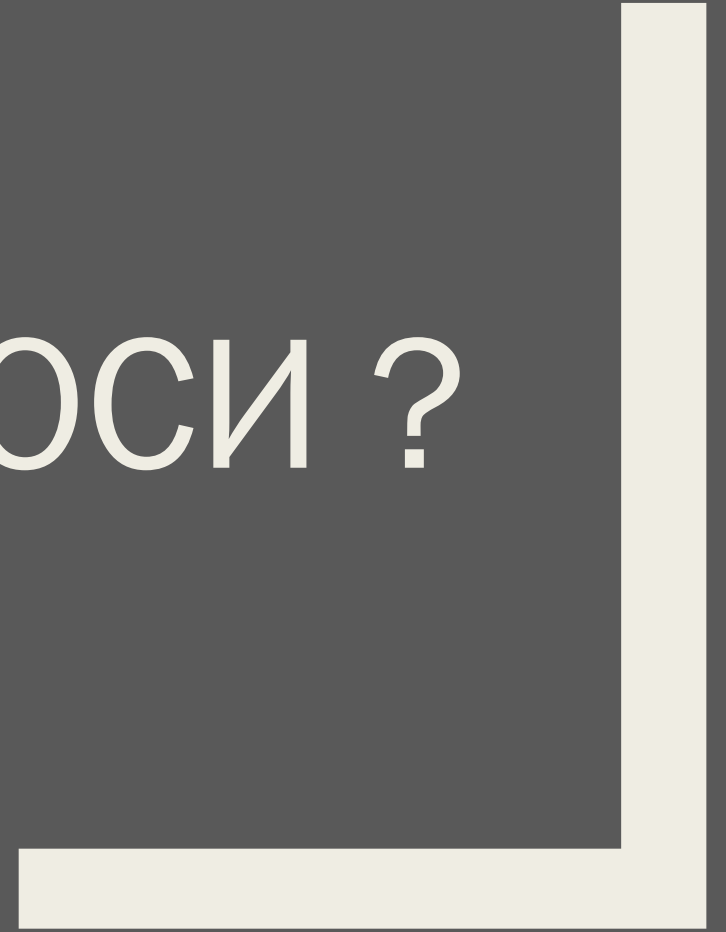
DEMO GOOGLE FIREBASE DATABASE

examples/DinosaurCatalog

DEMO GOOGLE FIREBASE DATABASE

examples/ConsoleChat

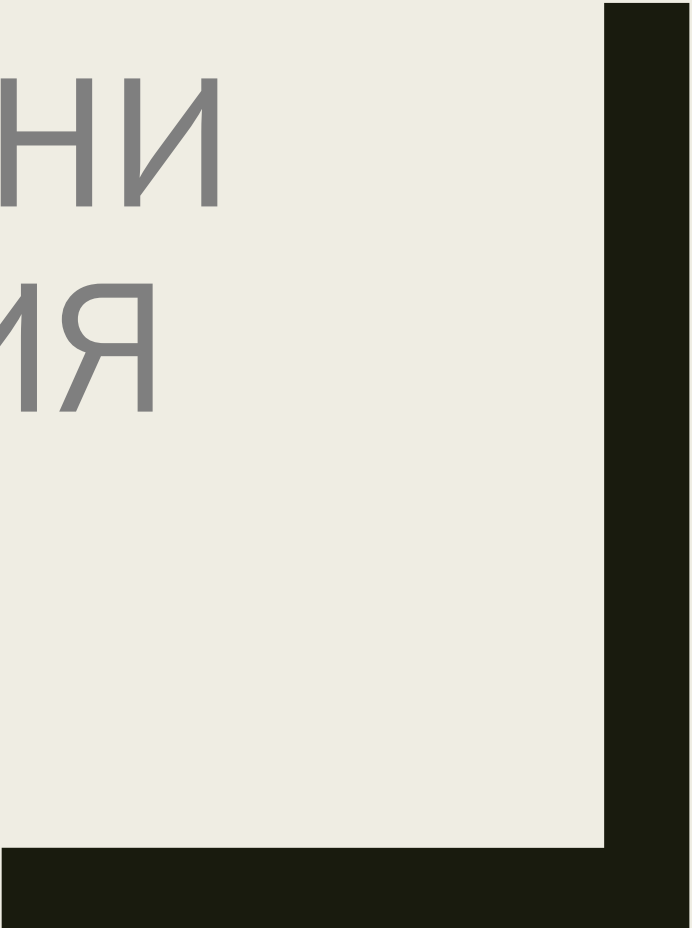
ВЪПРОСИ ?





РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev



АРХИТЕКТУРА „БЕЗ СЪРВЪР“

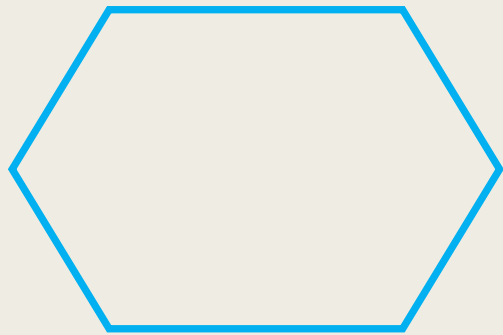
Традиционна архитектура

Традиционна архитектура

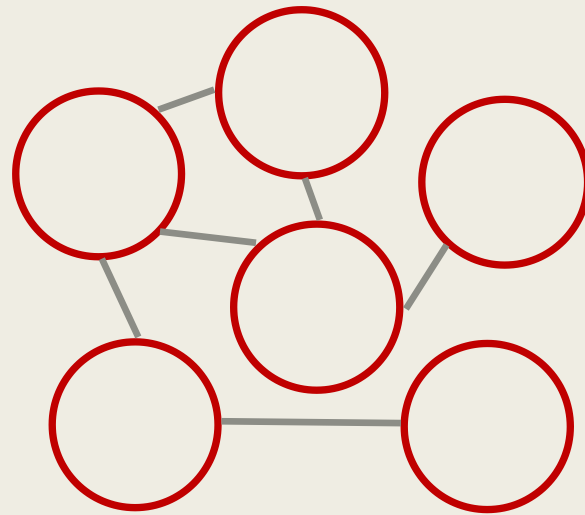
3 tier client-oriented



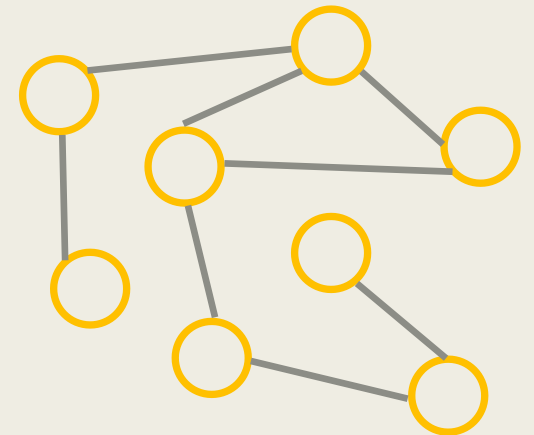
Еволюцията на бизнес логиката



Monolith



Microservices



Functions

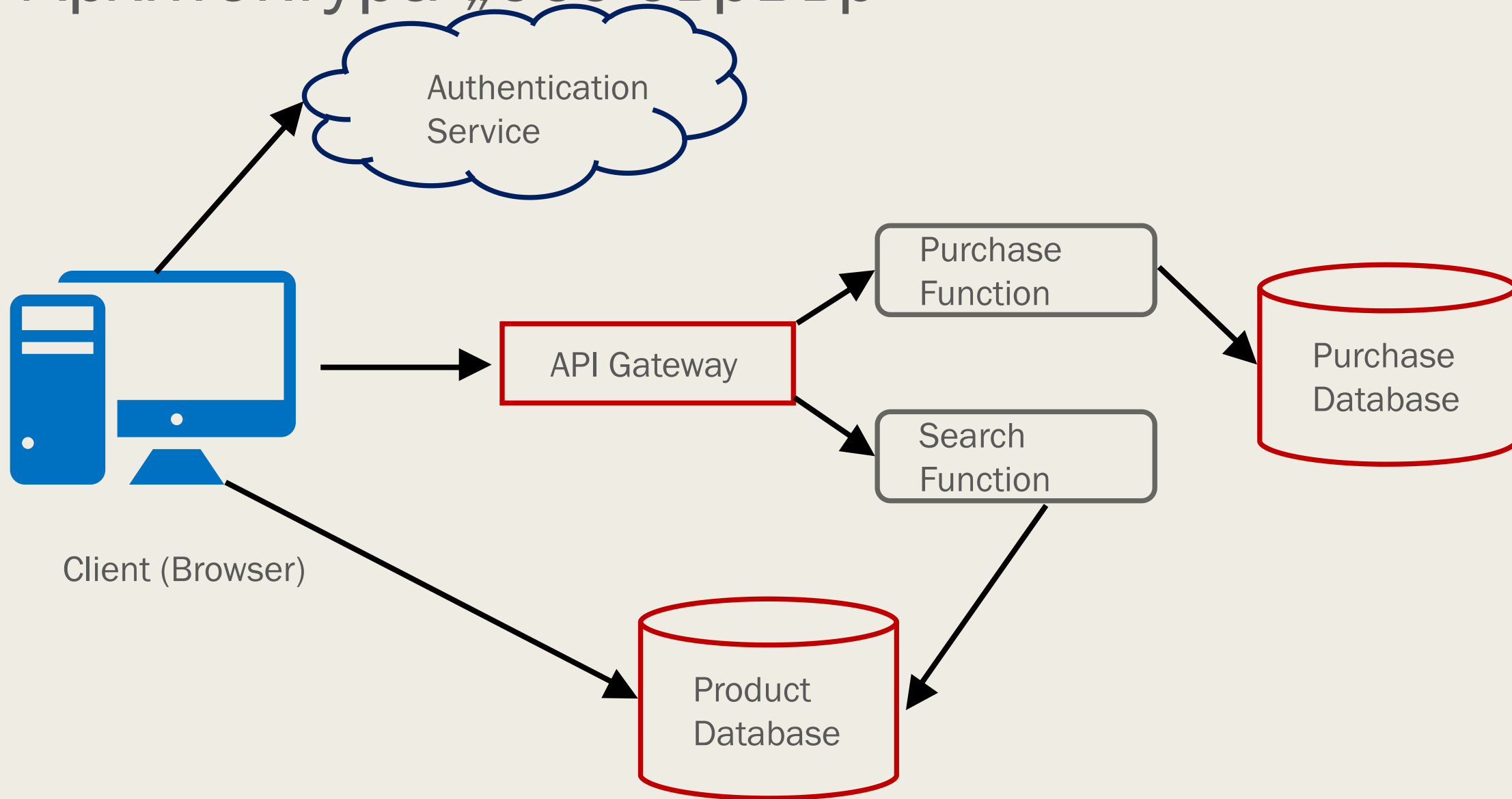
Какво е архитектура „без сървър“?

- „Без сървър“ архитектура наричаме приложенията, които в голямата си част или изцяло зависят от трета страна за обработката и обслужването на техните нужни. Този тип услуги е описван като BaaS ((Mobile) Backend as a Service).



- „Без сървър“ архитектура може да означава и приложения, на които голяма част от логиката е написана от програмисти, но е качена на отдалечени контейнери, които се извикват и изпълняват при определени събития. Тези контейнери са напълно поддържани от трета страна. За тях можем да мислим като за (Functions as a service / FaaS).

Архитектура „без сървър“



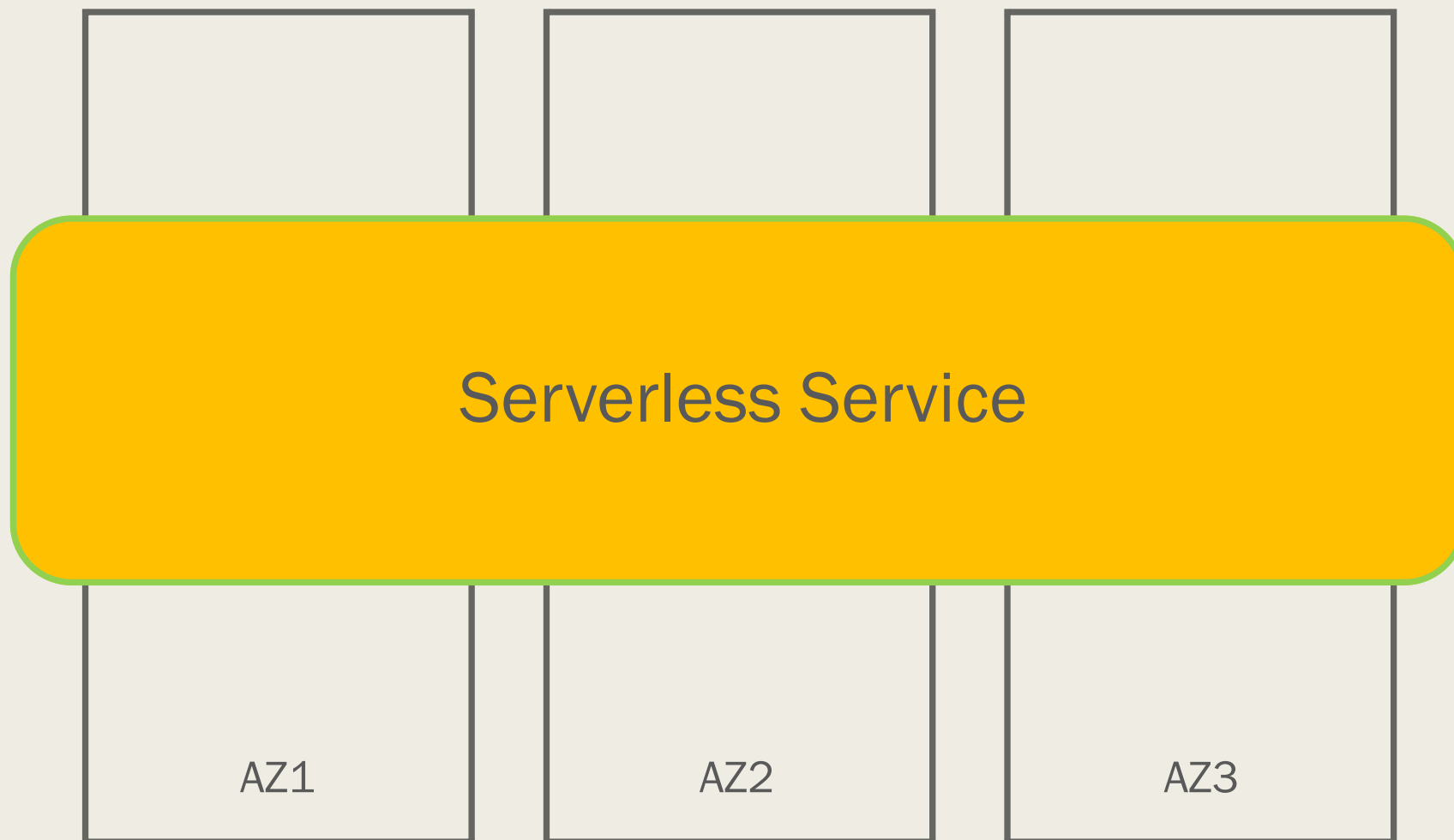
Примери за Functions

- Microsoft Azure
 - *Azure Functions - C#, F#, Node.js, Java, PHP ...*
- Amazon AWS
 - *Lambda Functions – Node.js, Java, C#, Go, Python ..*
- Google Firebase
 - *Firebase Functions – Node.js*

„Без сървър“ означава

- Без нужда от собствен сървър или от управлението му
- Плащаме само това, което използваме
- Разширение само според нашите потребности
- Достъпност и толерантност към средата

Регионална услуга



Добри практики Functions

- Да се минимизира големината на пакетите
- Да се разделя handler от основната логика на функцията
- Да се използват Environment Variables за модифициране поведението
- Да се възползваме от “Max Memory Used” за да определим правилно големината на function
- Да се премахнат големите неизползвани функции

AWS Lambda functions

Анатомия на Lambda Functions

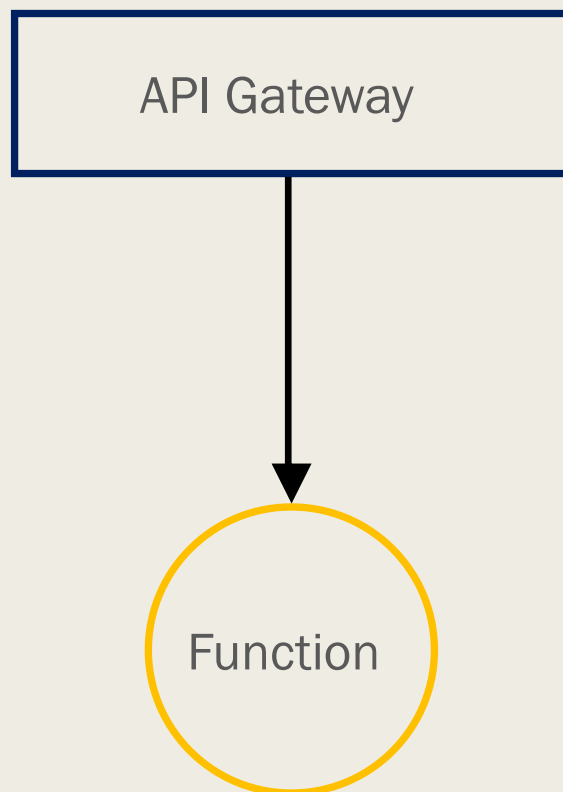
- Handler() function
 - *Функцията да бъде извикана при възникването на някакъв обект*
- Event object
 - *Изпращаната информация по време на извикване на функцията*
- Context object
 - *Достъпен метод отговарящ при изпълнението на функцията*

```
exports.myHandler = function(event, context, callback) {  
    console.log("value1 = " + event.key1);  
    console.log("value2 = " + event.key2);  
    callback(null, "some success message");  
}
```

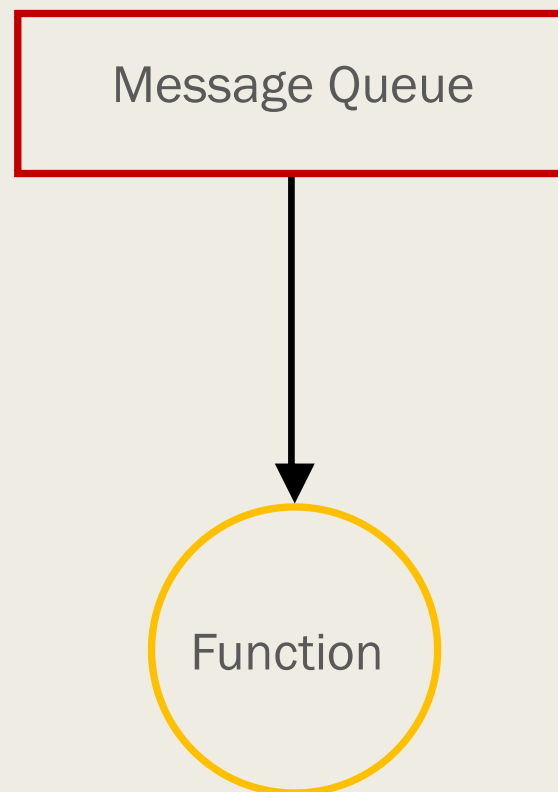
AWS Lambda function

Functions модел на изпълнение

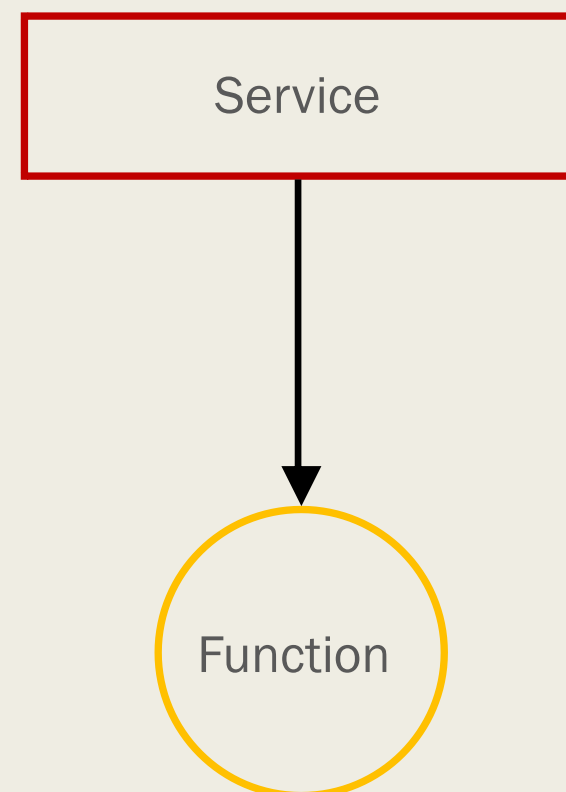
Synchronous (push)



Asynchronous (event)



Stream-based

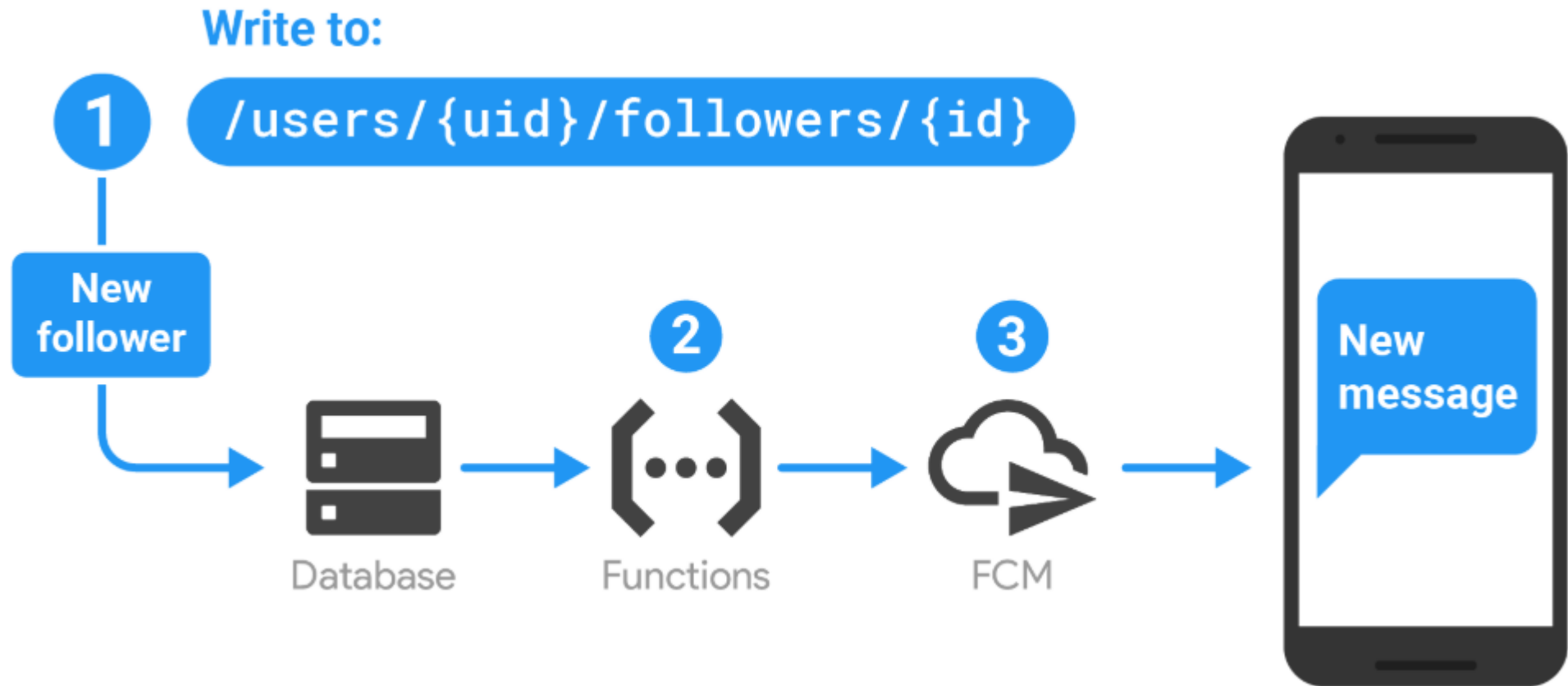


Google Firebase Functions

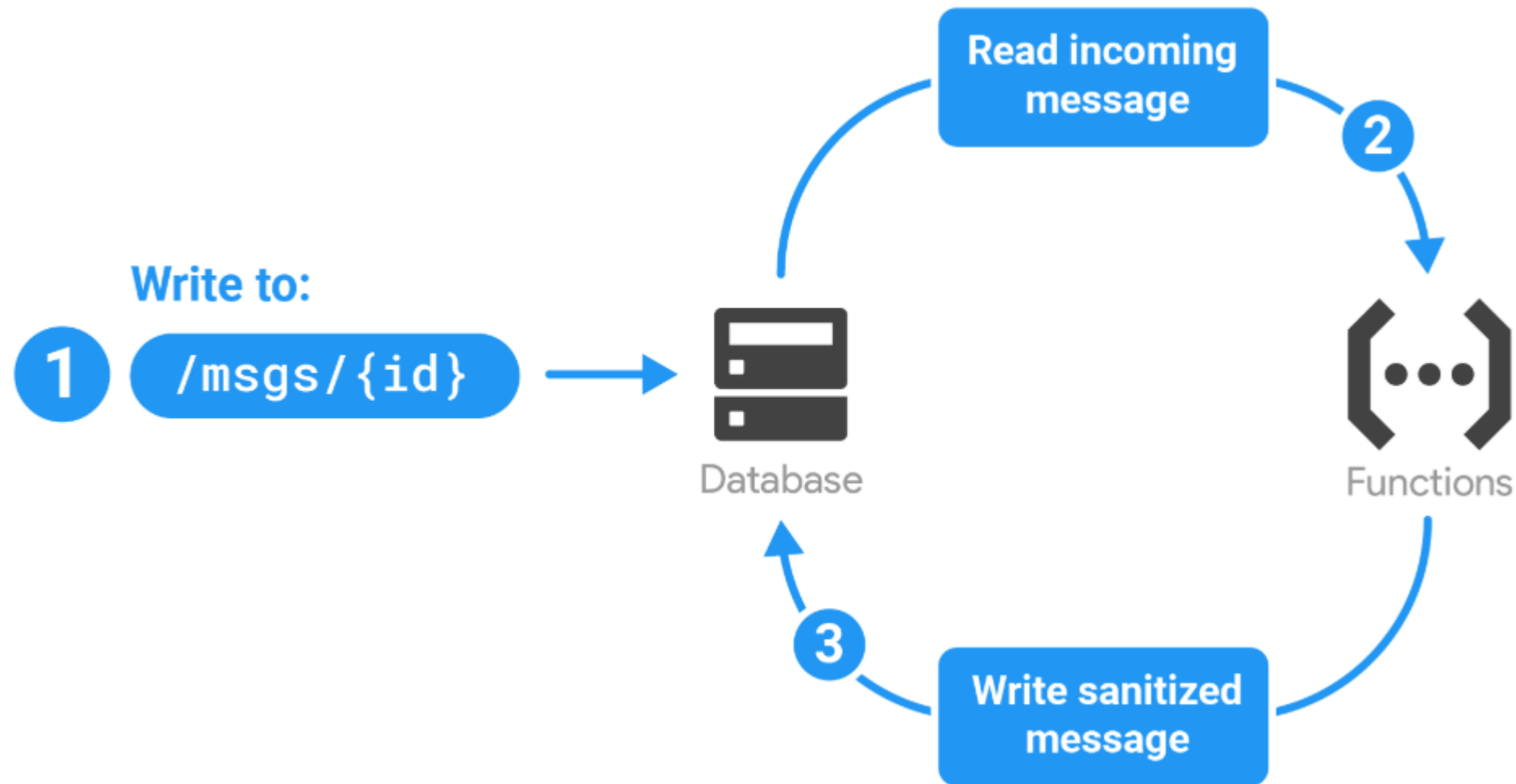
Интеграция на Firebase platform

- Cloud Firestore Triggers
- Realtime Database Triggers
- Remote Config Triggers
- Firebase Authentication Triggers
- Google Analytics for Firebase Triggers
- Crashlytics Triggers
- Cloud Storage Triggers
- Cloud Pub/Sub Triggers
- HTTP Triggers
-

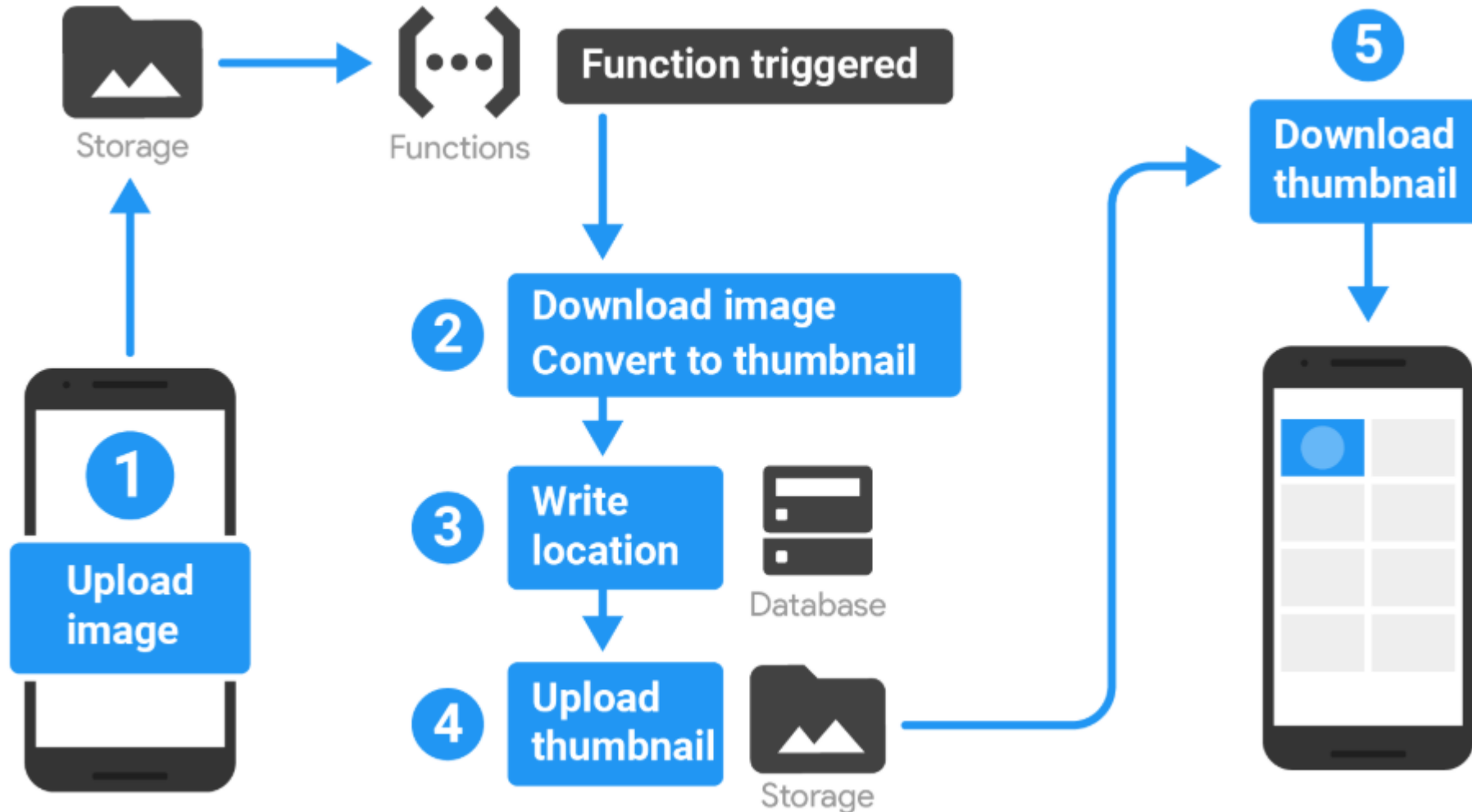
Уведомяване на потребител



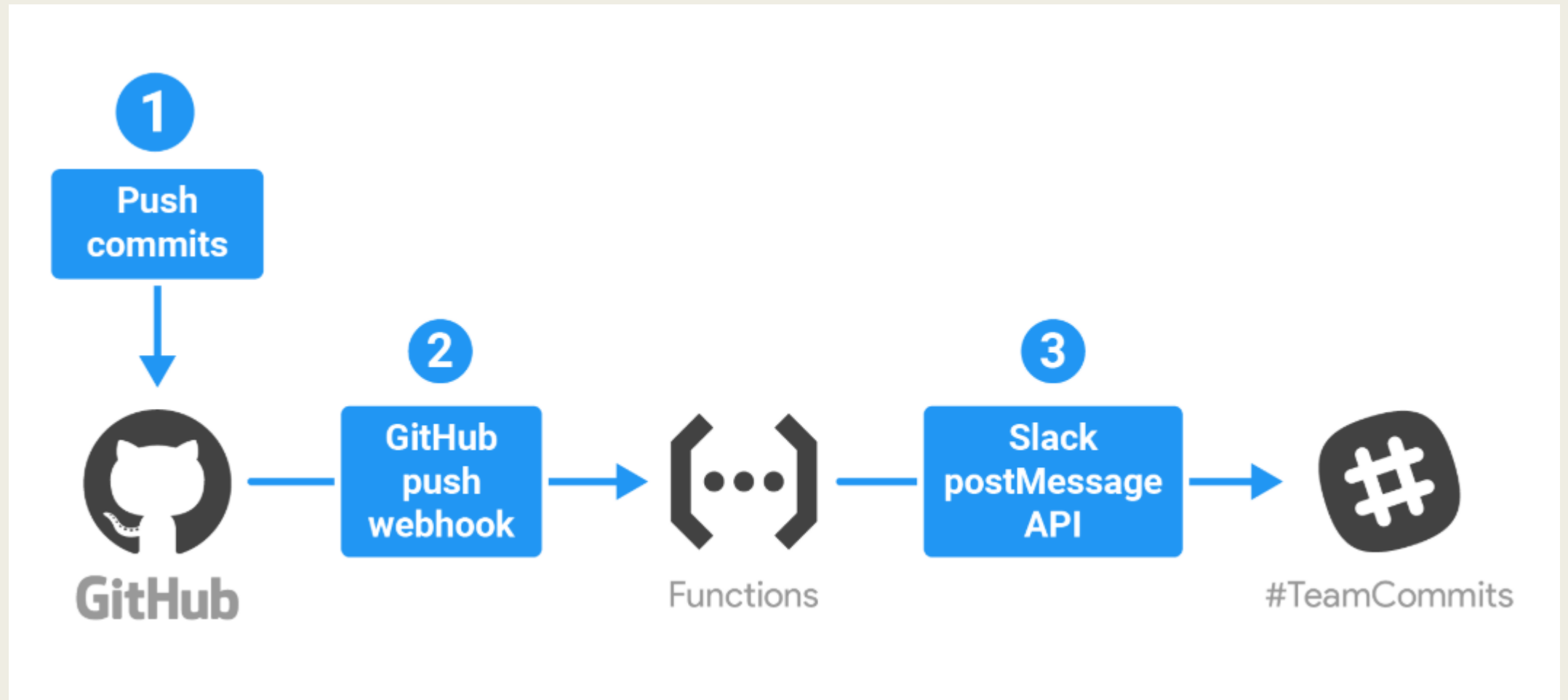
Realtime Database интеграция



Обработка на сложни задачи



Интеграция с трети услуги



DEMO FIREBASE FUNCTION 1

[examples/helloWorld-firebase-function-ts](#)

DEMO FIREBASE FUNCTION 2

[examples/translate-firebase-function-js](#)

Azure Functions

Интеграция на Azure function

Type	1.x	2.x and higher ¹	Trigger	Input	Output
Blob storage	✓	✓	✓	✓	✓
Azure Cosmos DB	✓	✓	✓	✓	✓
Azure SQL (preview)		✓		✓	✓
Dapr ↗ ³		✓	✓	✓	✓
Event Grid	✓	✓	✓		✓
Event Hubs	✓	✓	✓		✓
HTTP & webhooks	✓	✓	✓		✓
IoT Hub	✓	✓	✓		
Kafka ↗ ²		✓	✓		✓
Mobile Apps	✓			✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓	✓		✓
RabbitMQ ²		✓	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
SignalR		✓	✓	✓	✓
Table storage	✓	✓		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓

Azure functions 1 vs 2 vs 3 vs 4

Language	1.x	2.x	3.x	4.x
C#	GA (.NET Framework 4.8)	GA (.NET Core 2.1 ¹)	GA (.NET Core 3.1) GA (.NET 5.0)	GA (.NET 6.0) Preview (.NET Framework 4.8)
JavaScript	GA (Node.js 6)	GA (Node.js 10 & 8)	GA (Node.js 14, 12, & 10)	GA (Node.js 14) GA (Node.js 16)
F#	GA (.NET Framework 4.8)	GA (.NET Core 2.1 ¹)	GA (.NET Core 3.1)	GA (.NET 6.0)
Java	N/A	GA (Java 8)	GA (Java 11 & 8)	GA (Java 11 & 8)
PowerShell	N/A	GA (PowerShell Core 6)	GA (PowerShell 7.0 & Core 6)	GA (PowerShell 7.0) Preview (PowerShell 7.2)
Python	N/A	GA (Python 3.7 & 3.6)	GA (Python 3.9, 3.8, 3.7, & 3.6)	GA (Python 3.9, 3.8, 3.7)
TypeScript ²	N/A	GA	GA	GA

DEMO AZURE FUNCTIONS

examples/ToDoOperations

ВЪПРОСИ ?

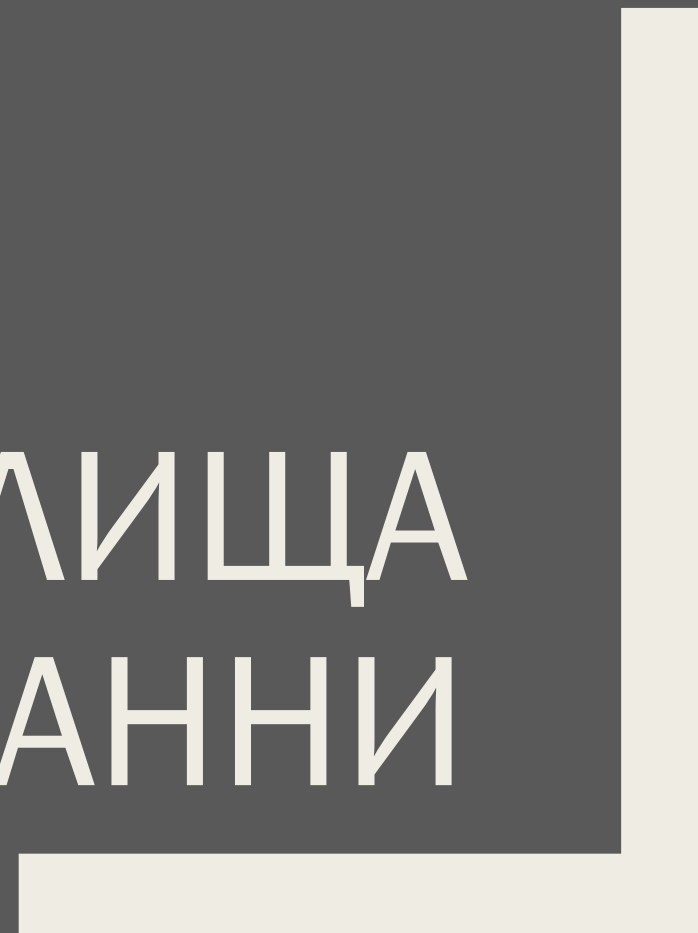




РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

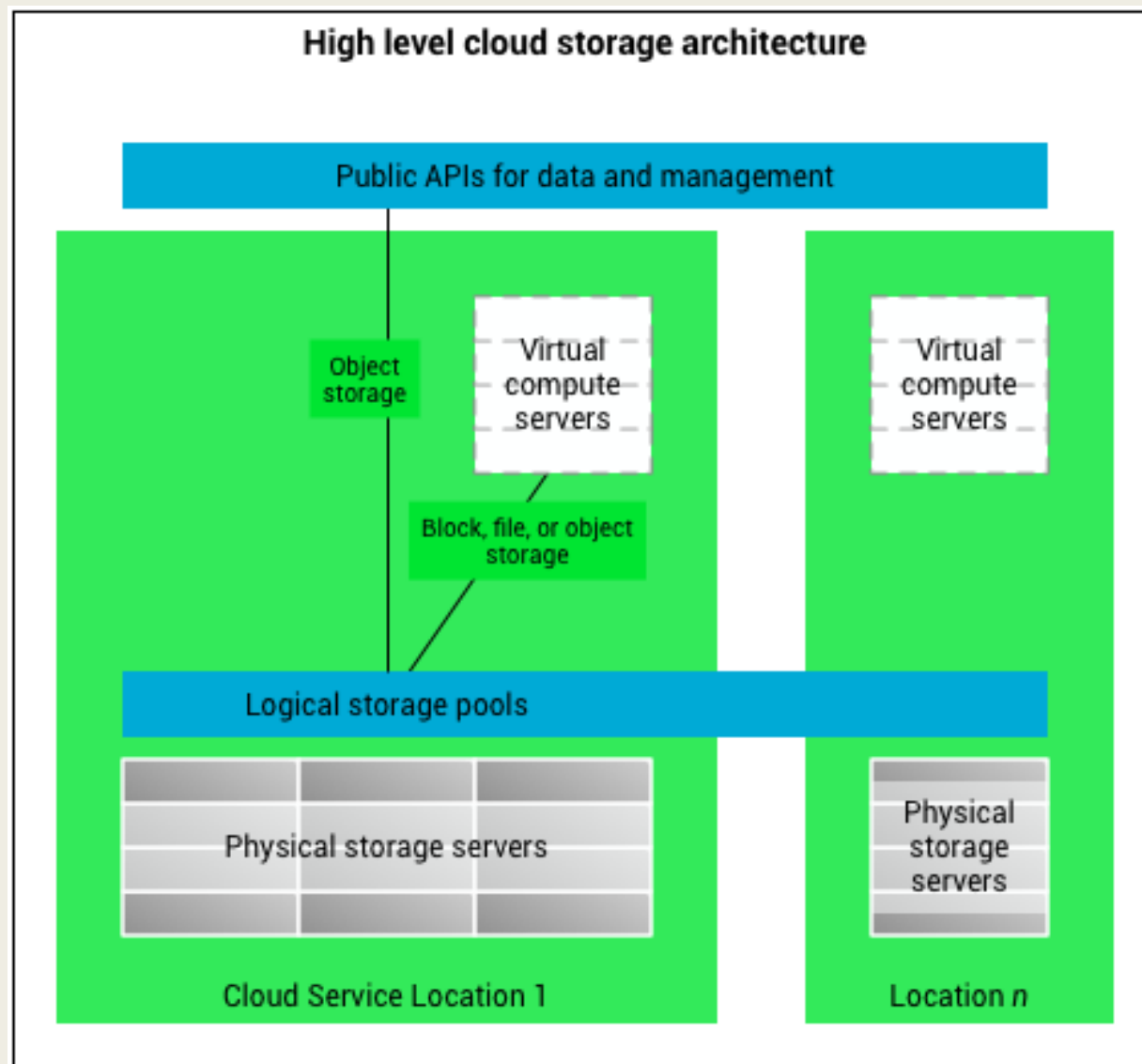
ОБЛАЧНИ ХРАНИЛИЩА ЗА ДАННИ



Облачни хранилища за данни

- Облачното хранилище за данни представлява модел на компютърното хранилище за информация, което съхранява дигитална информация в хранилища намиращи се в един или няколко логически пулове (logical pools).

Архитектура



Представители

- Microsoft Azure Storage
- Google Firebase Storage
- Amazon S3

Firebase storage

DEMO FIREBASE STORAGE

`examples/generateThumbnail-functions-js`

Azure Storage

Azure storage

- Azure Blob Containers
- Azure Queues
- Azure Tables

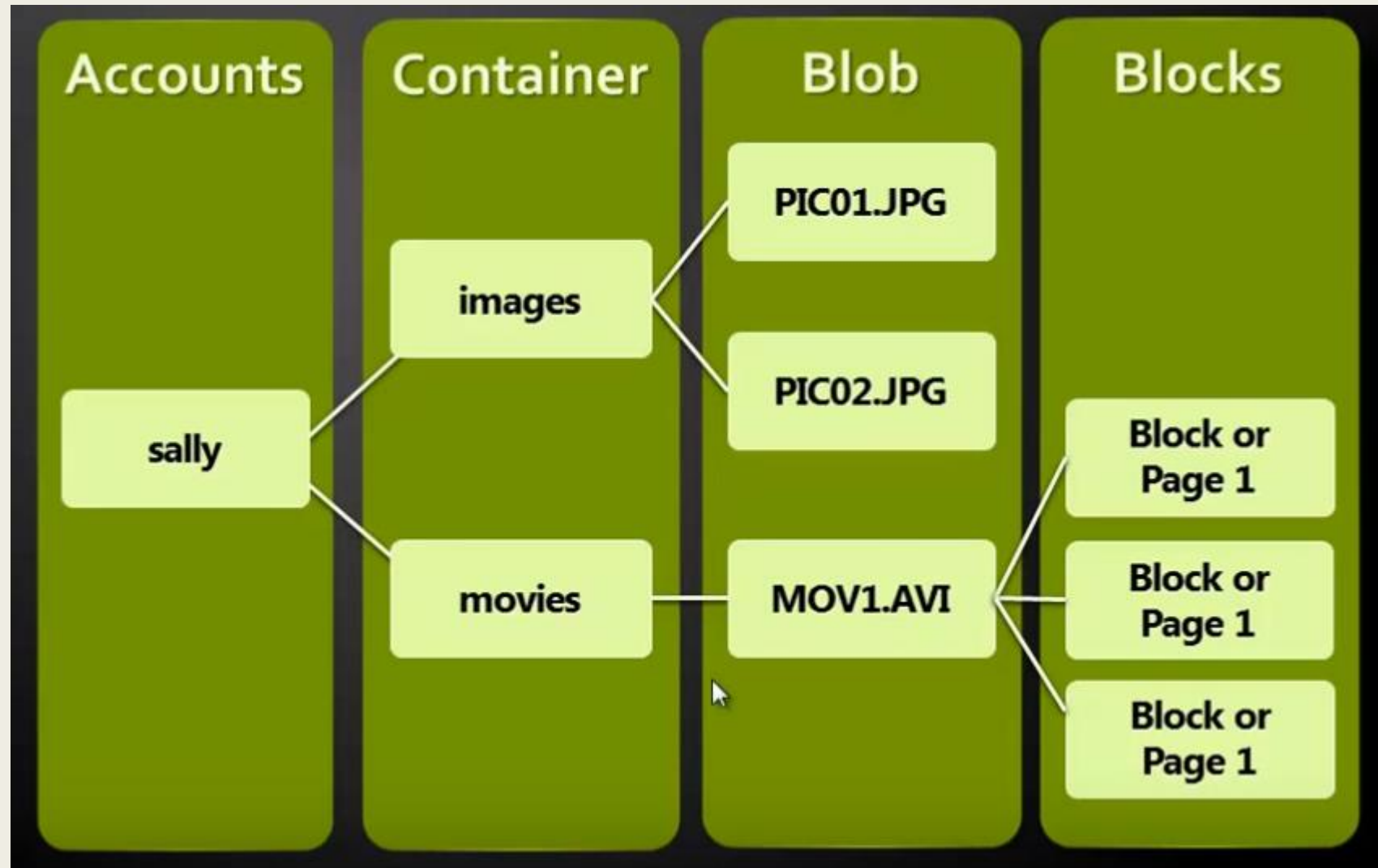
Какво са Azure Blob Storage?

- Azure Blob Storage представлява решение за облачно съхранение разработено от Microsoft. Blob storage е оптимизиран за работа с масивни количества неструктурирана информация.

Приложение на Azure Blob

- Доставка на снимки и документи директно до брауъра;
- Съхранение на файлове за разпределен достъп;
- Стрийминг на видео и аудио;
- Съхранение на log файлове;
- Съхранение на информация за backup;
- Съхранение на информация за анализи.

Blob Storage Concepts



Какво са Azure Queues?

- Azure Queues представлява услуга за съхранение на голямо количество от съобщения, които могат да бъдат достъпни от всяка една точка посредством HTTP и HTTPS. Единично съобщение от опашката може да достигне до 64 KB.

Приложение на Azure Queues

- Създаване на backlog от задачи, които могат да се изпълняват асинхронно;
- Прехвърляне на съобщения между Azure web role и Azure worker role.

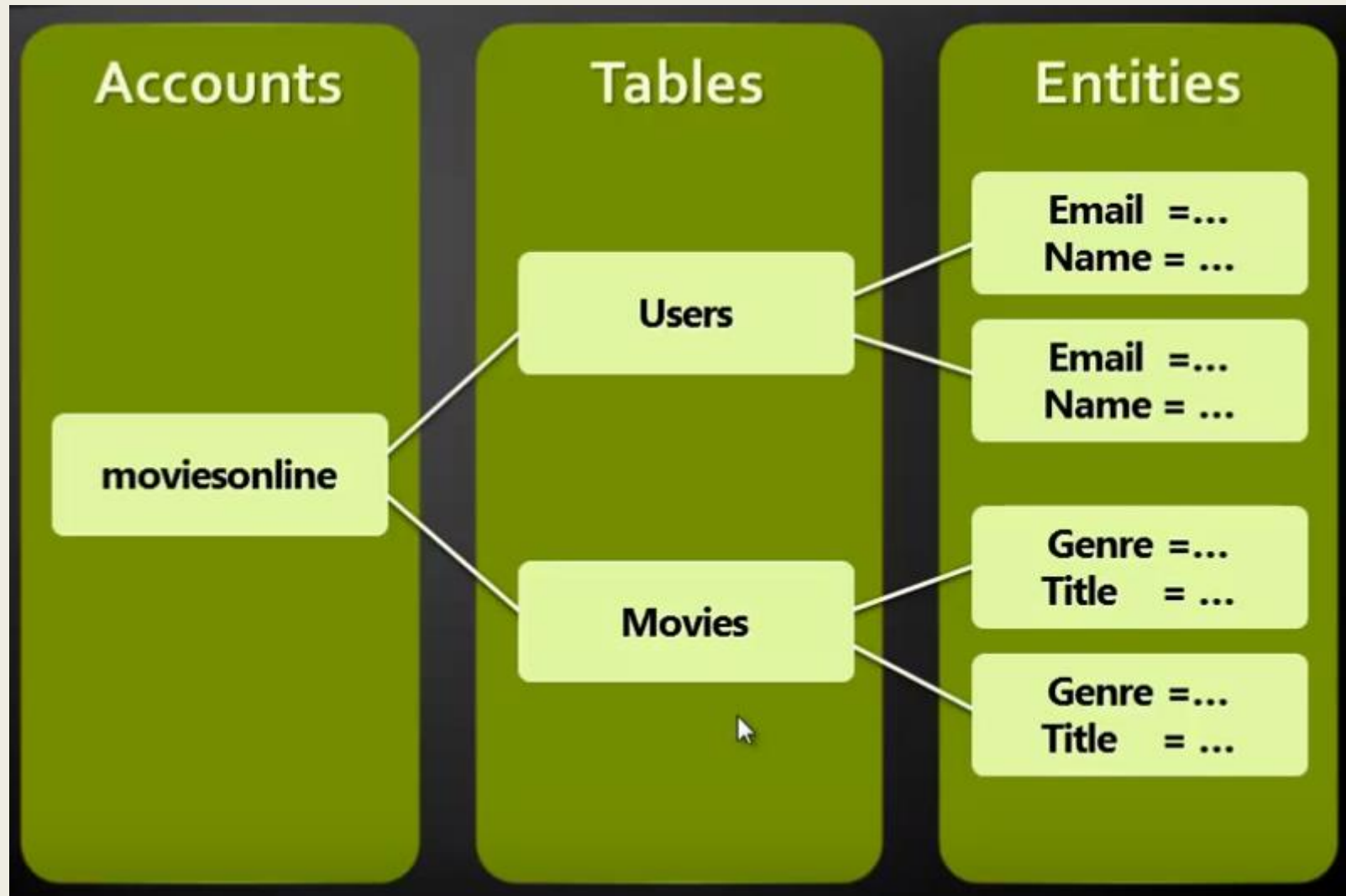
Какво са Azure Tables?

- Azure Tables представлява услуга предоставяща възможност за съхранение на голямо количество структурирана информация. Услугата представлява NoSQL хранилище за информация.

Azure Tables спецификация:

- Служи за съхранение на ТВ с информация;
- Съхранената информация не трябва да има сложна json структура, външни ключове и складирани процедури (stored procedures). Информацията трябва да е представена под формата на денормализирани данни;
- Бързо извличане на данни чрез използването на клъстерни индекси;
- Достъпа до данните се осъществява чрез OData протокол и Linq заявки изградени върху WCF Data Service .NET Libraries.

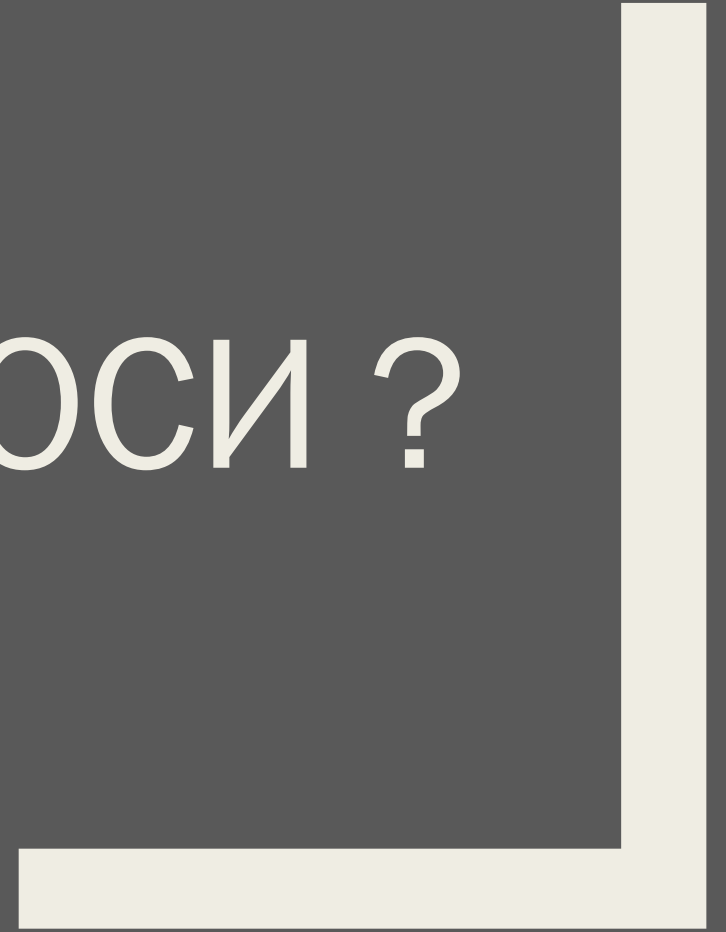
Table Storage Concepts



DEMO AZURE STORAGE

`examples/AzureDataStorage.Management`

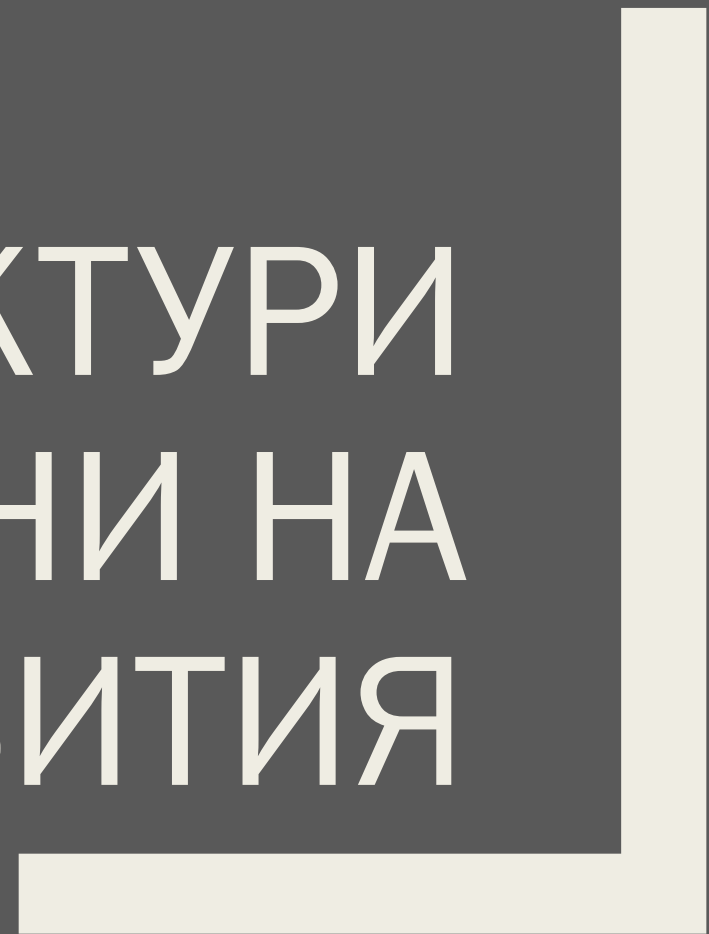
ВЪПРОСИ ?



РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

АРХИТЕКТУРИ БАЗИРАНИ НА СЪБИТИЯ



Какво е event?

- Комплект от информация за определен обект случващ се в определено време. Обикновено капсулира в себе си съобщения.

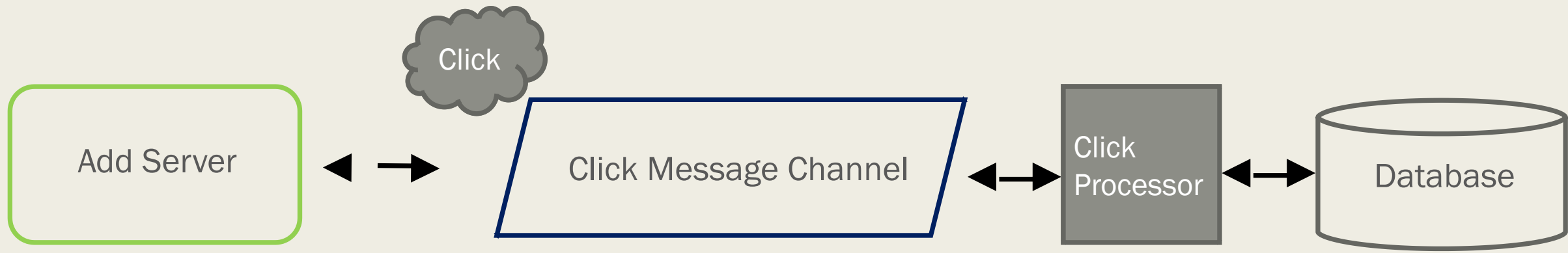
Характеристики на event

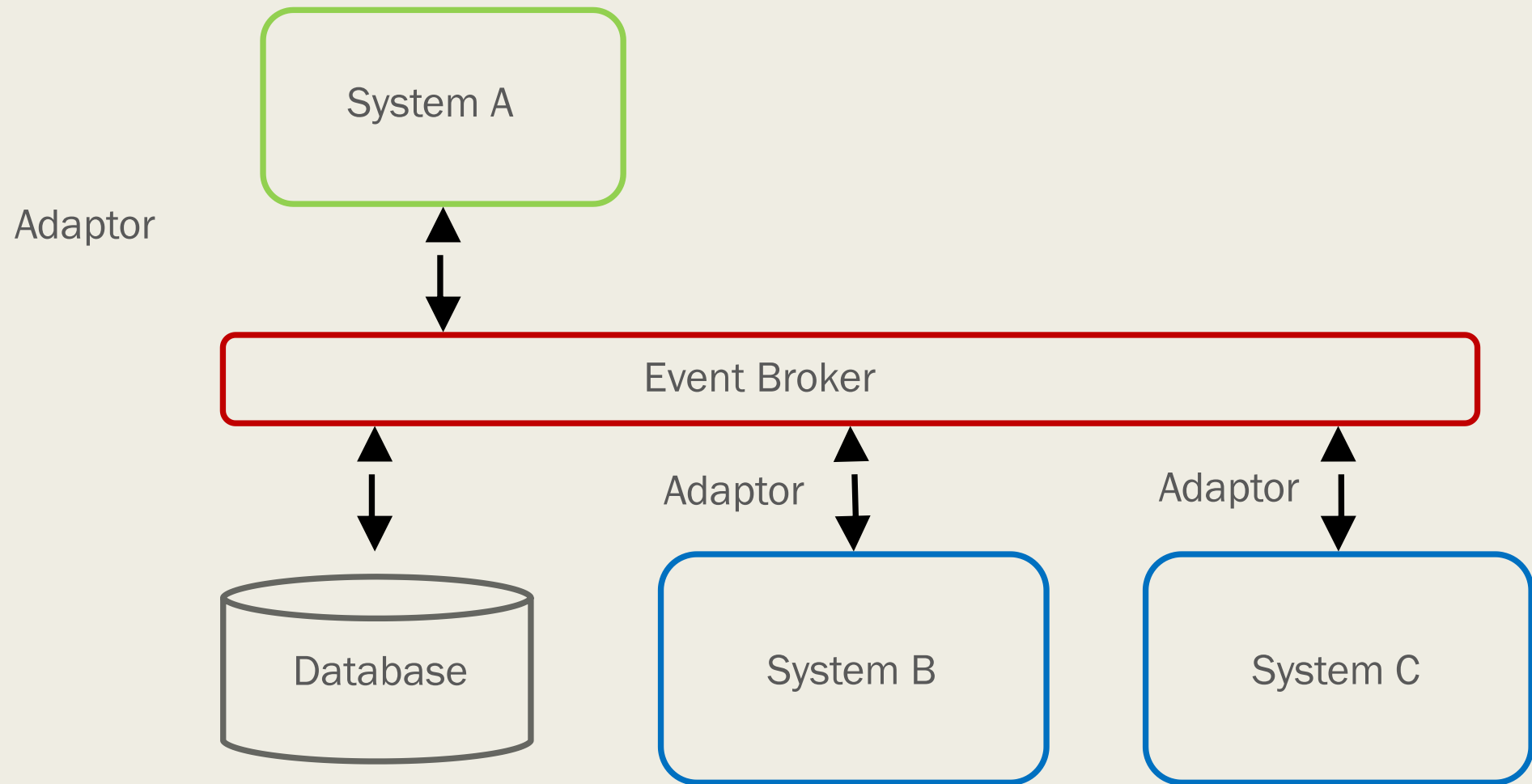
- Представят промяна в статуса
- Самостоятелни
 - *Пълно представяне на конкретно събитие*
 - *Не се отнася към друга структура на данни*
 - *Намалява зависимостите и разхлабените връзки*
- Уникално представен
 - *Активира еднозначно боравене със събитието*
 - *Позволява връзка към свързани събития*
- Време свързани, а не време зависими

- Изградени от съобщения
- Observable
 - *Публикуваните събития могат да бъдат следени от много консуматори*

Event Driven Architecture (EDA)

- Метод за създаване на Ентерпрайз приложения, в които системата се управлява благодарение на потока от събития между отделните компоненти. Тази архитектура се окомплектова добре с асинхронни събития в непредвидими среди. Архитектура за създаване на разпределени приложения.





Publisher-subscriber communication mechanism

- Разпределен модел за събития
- Събитията са свързани с логически модел - „Event Stream“ / “Topic”
- Topic е дървовиден базов модел или пространство, което прави организацията на събития много лесна

Събития Producer / Consumer

Event Producer

- Публикува съобщение, което представлява събитие

Event Consumer

- Свързва се със събитието посредством Topic селектор
- Обработва събитие асинхронно

Примери

- Facebook + Messenger
- Twitter + Twitter application
- AirBNB + AirBNB application

....

С какво EDA е по различна от SOA?

SOA

- Приложенията се създават в дизайн фазата
- Линеино поведение между услугите
- Предсказуемо поведение
- Request / Response

EDA

- Приложенията се създават в run-time
- Асинхронни компоненти
- Реактивно поведение
- Естествено допълва разпределените системи

EDA допълва SOA

Какво е event-driven messaging?

- Event-driven messaging представлява софтуерен шаблон за дизайн, които често се използва в комбинация със SOA. EDM дава възможност на потребителите на услуги, които се интересуват от събития, случващи се в периферията на даден доставчик на услуги, да получават уведомления за тях.

SOA шаблон: Event-driven messaging

■ Проблем

- *Как автоматично можем да известяваме клиента на услугите при възникването на събитие?*

■ Решение

- *Клиента на системата трябва да стане subscriber на услугите. Услугите автоматично известяват техните клиента за възникнали нови събития.*

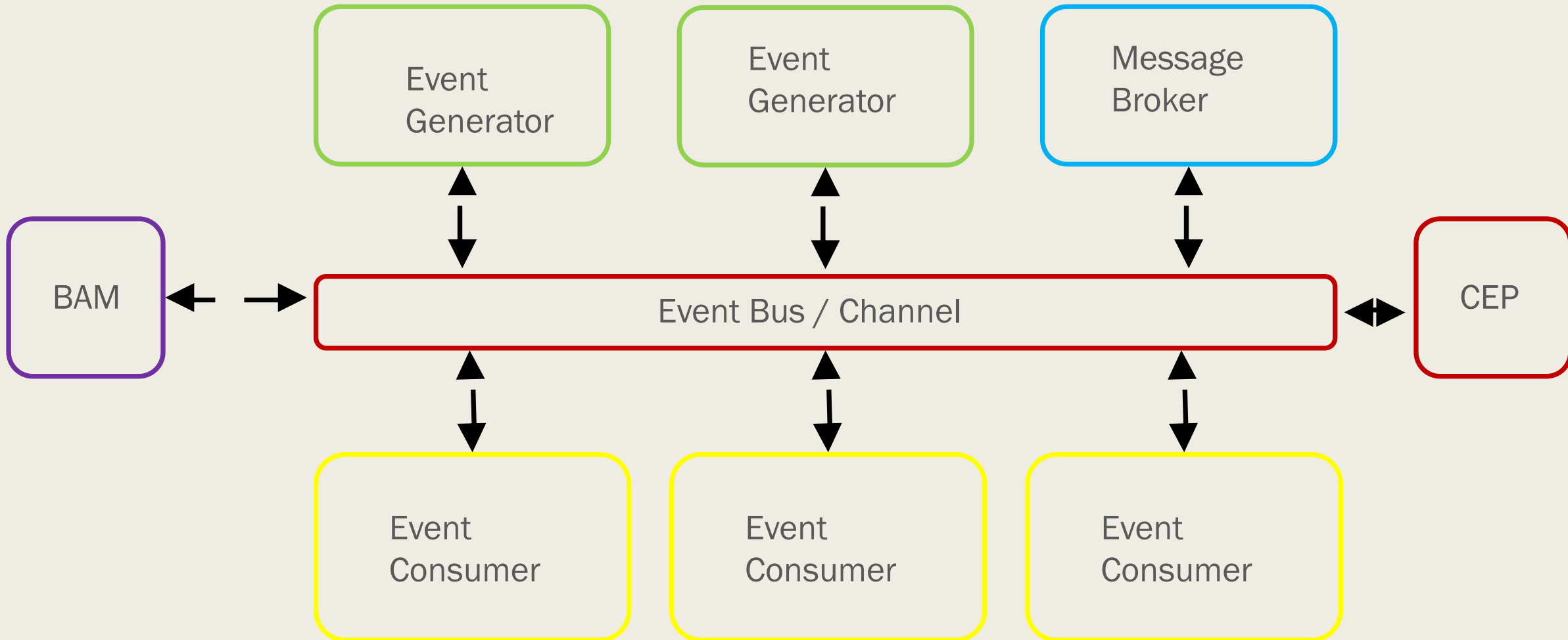
■ Приложение

- *Messaging Framework предлага възможност за разработка на приложения поддържащи publisher / subscriber шаблони.*

Event Transports technologies

- JSM
- WS-Eventing / SOAP
- AMQP
- XMPP
- MQTT

EDA със SOA



Event Bus / Channel

- Доста често Ентерпрайз Service Bus
- Свързва създатели и консуматори посредством шина
- Интегрира несъбитийно базирани системи

Message Broker

- Поддържа различни шаблони за съобщения
- Базиран на опашка publisher / subscribe
- Сигурен доставчик на съобщения / събития

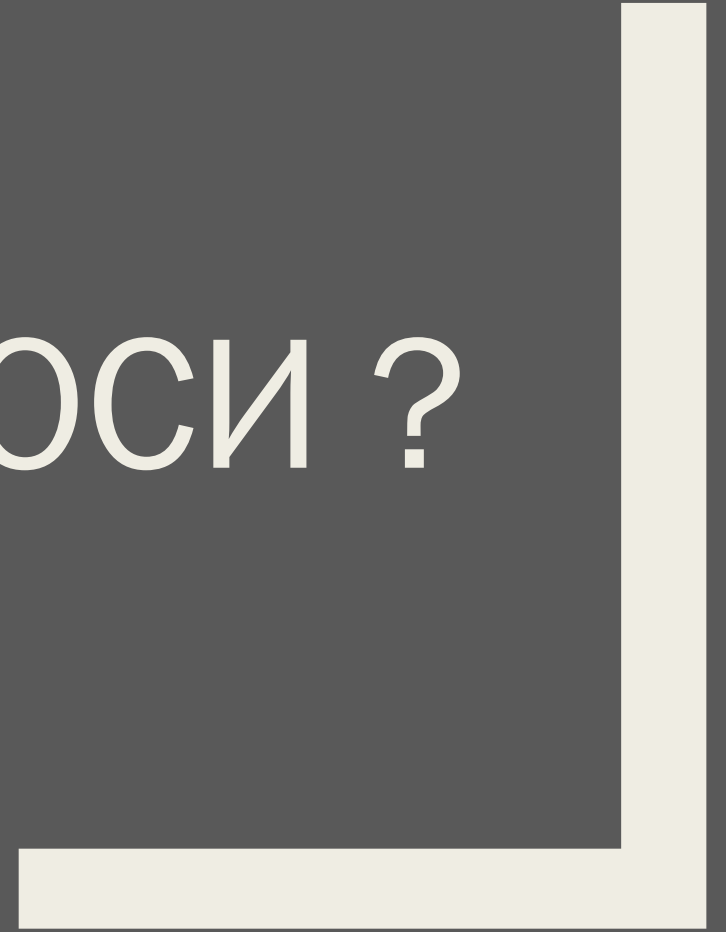
Business Activity Monitoring

- Позволява наблюдаване на целия процес по разпространение на съобщенията
- Идентифициране / събиране / следене на бизнес транзакции

Complex Event Processing

- Позволява четене на повече от един непрекъснат поток от събития

ВЪПРОСИ ?





РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

ИНТЕРНЕТ НА НЕЩАТА(IOT)

История на интернет на нещата

- 1800s – Появяват се първите електрически уреди (телеграфа факс машината, радио ...)
- 1926 – Никола Тесла изобретява безжичната свързаност
- 1989 – Tim Berners-Lee предлага идеята за WWW
- 1990 – Първите свързани устройства са създадени (тостер и вендинг машина)
- MID 1990s – Възход на интернет и на експерименталните неща
- 1999 – Kevin Ashton измисля “IoT” и създава MIT Auto-ID Center

- 2000 – LG анонсира планове за създаване на умен хладилник
- 2005 – ООН за първи път споменава IoT в официален документ
- 2011 – Internet Protocol Version 6 (IPv6) се въвежда
- 2013 – Intel създава „Internet of things solutions group“

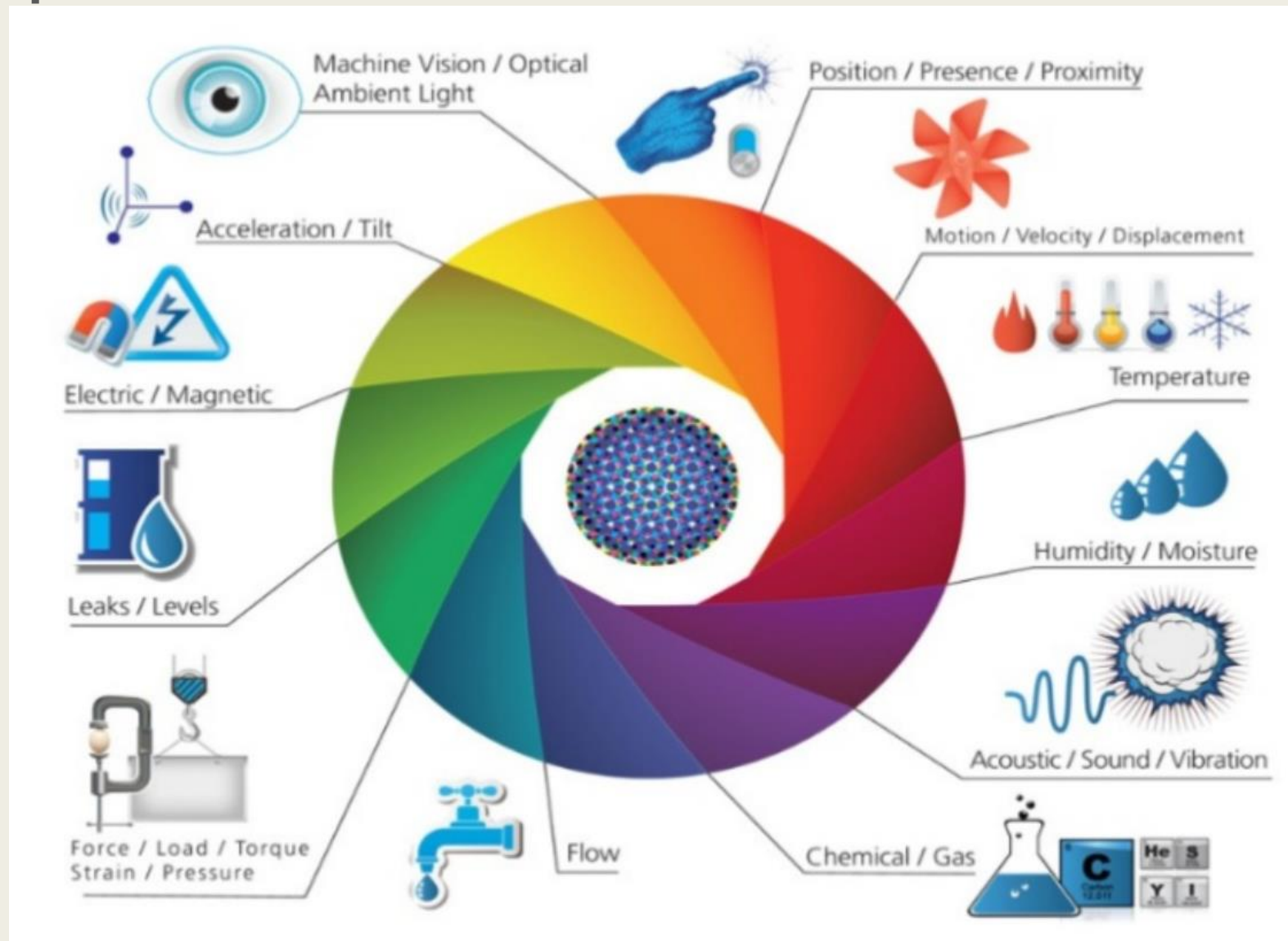
Интернет на нещата (IoT)

- Интернет на нещата представлява интелигентна свързаност между хора и устройства обменящи информация помежду си с цел създаване на стойност.

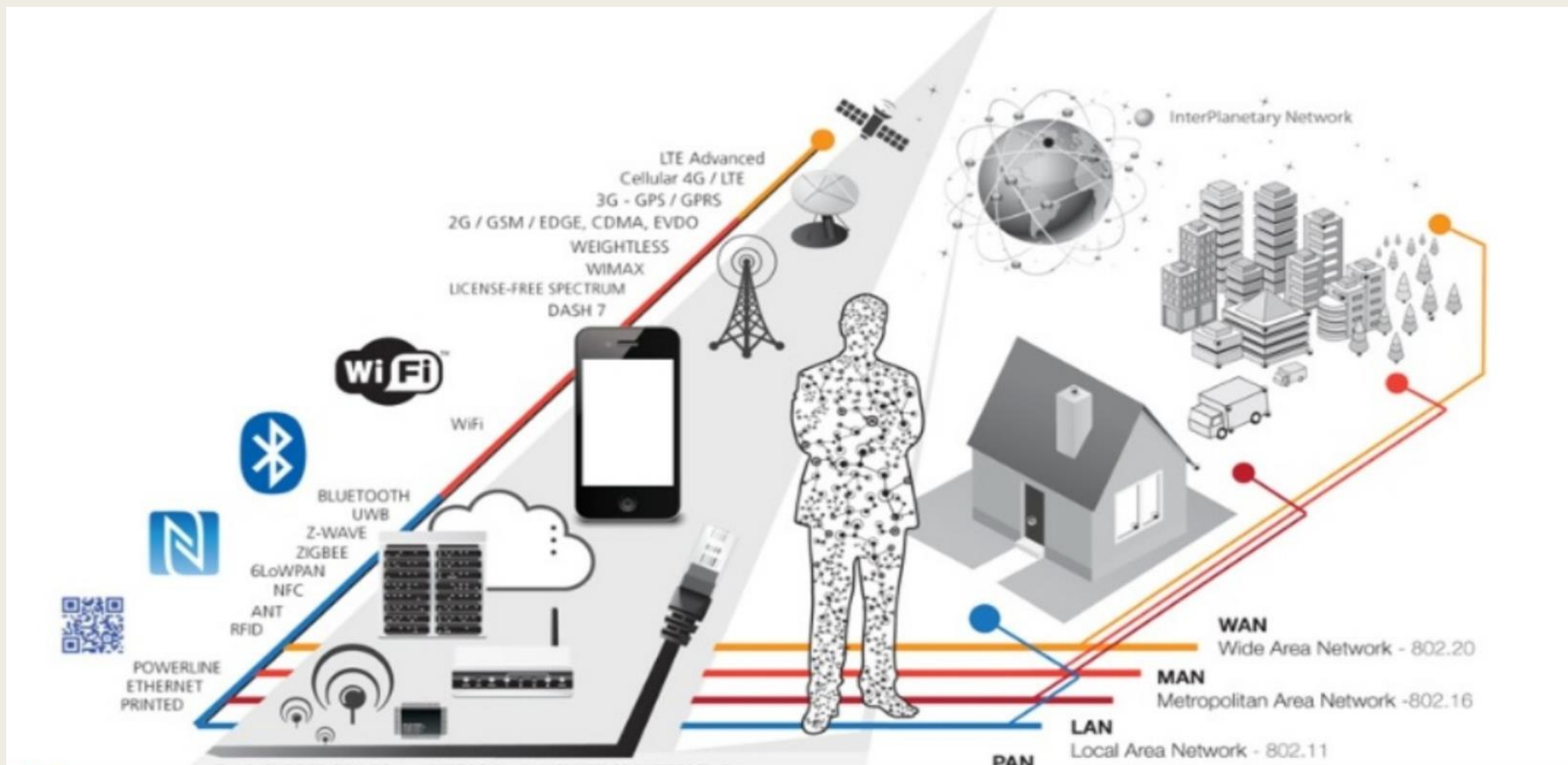
Умните устройства и Интернет на нещата са движени от комбинация на

- Сензори
- Свързаност
- Хора и процеси

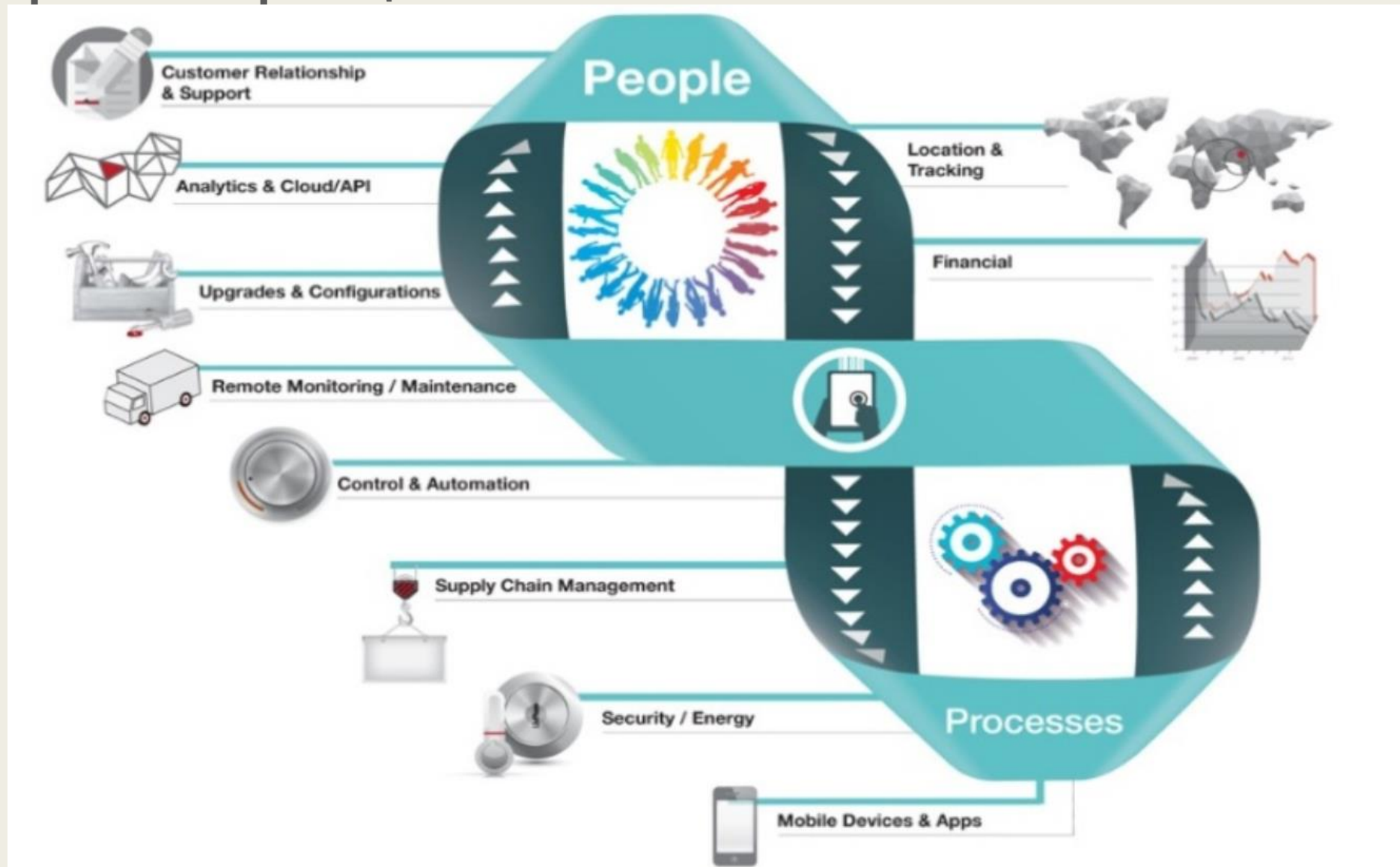
Сензорите могат да бъдат



Свързаност



Хора и процеси



Интернет на нещата (IoT)

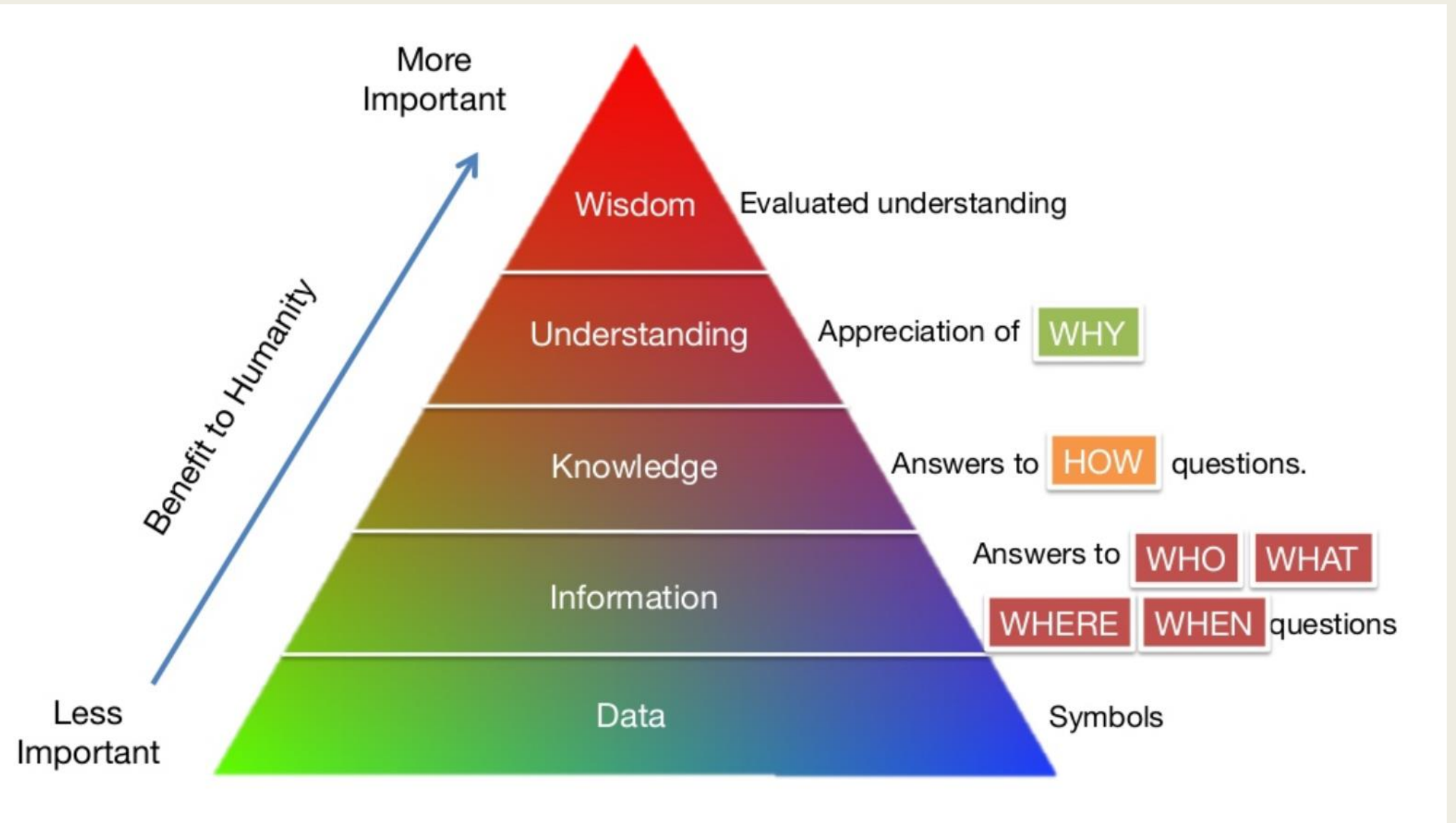
- Различните комбинации между тези обекти създават нови типове умни приложения и услуги. Като:
 - *Сензори за паркиране*
 - *Свързани автомобили*
 - *Умни термометри*
 - *Умни проследяващи устройства*
 - и други.*

Отключване потенциала на IoT

- Подобряване на производителността
- Намаляване на разходите
- Създаване на новаторски услуги
- Нови потоци на доходи

Интернет на всичко (IoE)

- Според проучване на Cisco в следващото десетилетие пазара на IoT ще достигне до 1.9 трилиона долара.



5-те въпроса пред IoT

Как информацията се създава от IoT?



Как хората се вписват?

- IoT прави устройствата по – добри и по ефективни версии на вече съществуващите такива.
- Ако се вгледаме по дълбоко IoT устройствата не са само умни устройства, а устройства и услуги които правят хората по „умни“.

- Хора срещу Компании: Кой е печелившия?
- Как компаниите ще опазят информацията?
- Как IoT създава стойност по веригата за доставка?
 - *Интернет на нещата отваря възможност за нови подобрения и позволява на компаниите да намерят път към захранване на веригата за доставки.*

ВЪПРОСИ ?

