SOA
Service Oriented Architecture

В корпоративных приложениях мы застряли в архитектуре клиент-сервер. Многие просто до конца не понимают, что такое SOA, где использовать, и какие мы получаем преимущества.

SOA — мечта индустрии программирования о замене «кустарного» кодирования программ «от и до» на «промышленную» сборку приложений из «стандартных комплектующих», как в автомобильной, или других традиционных отраслях промышленности.

Сервисы в архитектурах информационных систем (ИС) организаций существенно меняют их бизнес-процессы в лучшую сторону.

Мы рассмотрим технологии разработки и использования сервисов **ASMX**, **gRPC**, **WCF** в ИС.

Цели SOA:

Для крупных Информационных Систем (ИС), уровня предприятия, и выше:

- сокращение процесса разработки,
- расширение повторного использования кода,
- независимость от используемых платформ, инструментов, языков разработки,
- повышение масштабируемости создаваемых систем,

SOA и информационные системы (ИС) компаний

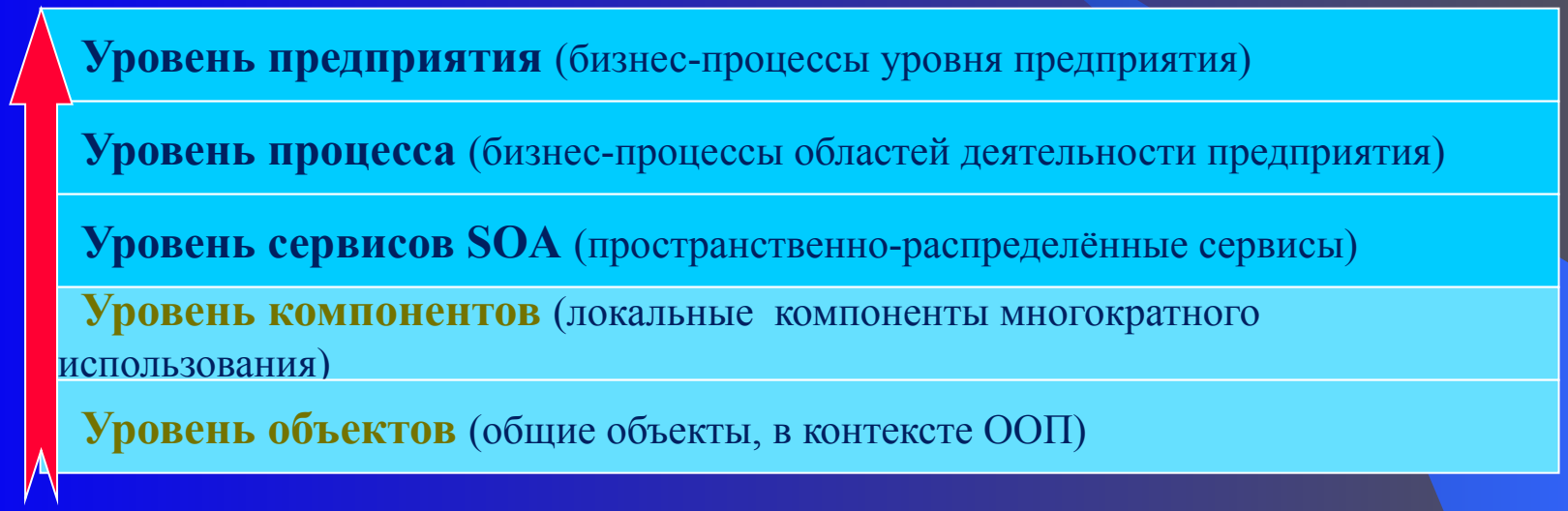
Теперь компании перестают быть зависимыми от поставщиков ПО в виде готовых ИС. Логика каждой ИС теперь строиться внутри компании в виде сборки (вызова) нужных служб. Бывшие поставщики ИС теперь ориентируются на предоставление сервисов.

Внедряя SOA компании создают у себя *виртуальные, распределённые ИС* .

Внимание бизнеса переключается с названия и поставщиков ИС на качество и доступность нового сервиса. Меняются правила работы с поставщиками систем, меняется даже организационная структура компаний.

Уровни абстракции SOA

С точки зрения SOA декомпозиция корпоративной ИС может осуществляться на нескольких уровнях абстракции



Жизненный цикл корпоративной системы «распадается» на жизненные циклы составляющих ее компонентов. Декомпозиция позволяет не только оперативно реагировать на реструктуризацию бизнес-процессов, но и делает процесс развития ИС более предсказуемым и устойчивым.

Разработка и внедрение SOA

На разработчиков служб ложится высокая ответственность , т .к. их работа «встраивается» и существенно влияет на ИС многих компаний.

Разработка или покупка готовых ИС (без SOA) – это мина замедленного действия. Когда-то любая ИС устаревает, с её смертью у вас ничего не остается – ни бизнес-процессов, ни сервисов, ни модели данных. Время потраченное на проектирование бизнес- и технологических процессов, уникальные идеи и находки окажутся бесполезными.

При разработке новых ИС необходимо выделять сервисы, которые уже существуют или необходимы и будут многократно востребованы. Создание повторно используемого компонента примерно в 2.5 раз дороже простого, т. е. сервис целесообразно создавать при условии дальнейшего использования минимум 3 раза.

После перехода на SOA с каждым годом возрастает повторное использование сервисов – 10% в первый год, 20% во второй, 30% в третий, что напрямую сказывается на сокращении затрат.

Доступ к данным в SOA

Существуют два способа доступа к данным в распределённых ИС:

1. **REST** (Representational State Transfer – передача состояния представления) это клиент-серверный доступ к *данным* по HTTP (REST не протокол, а архитектурный стиль в HTTP):
   - Вся логика крутится вокруг ресурсов (*данных*), а не операций с ними;
   - Каждый блок некоторой информации является ресурсом с уникальным URI;
   - Все запросы реализуются по HTTP, методами Post, Get, Put и Delete, а данные могут передаваться в любом формате (не только XML);
   - Операции клиента с сервером реализуются без сессий.

2. **RPC** (Remote Procedure Call – удалённый вызов процедур) это сервисы (процедуры), которые предоставляют доступ к удалённым *операциям* (методам обработки данных) по особым протоколам:
   - SOAP, JSON (JavaScript Object Notation), .NET Remoting, Java RMI…
   - Двоичный Protobuf в gRPC.

REST или RPC для Web-сервисов?

Web-сервисы SOAP (XML) – это целое семейство дополнительных протоколов и стандартов. Сервисы хорошо справляются с ошибками, сложными транзакциями с сохранением состояния (цепочка операций как одна транзакция, например, в случае банковских переводов), безопасностью, надёжностью связи… – целесообразно использовать для разработки сложных коллективных проектов ИС.

Технологии REST и RPC – не конкуренты. Они представляют разные весовые категории. REST-доступ с JSON или gRPC лучше подходят для работы с микросервисами (подвид SOA, небольшие сервисы не предназначенные для сторонних пользователей) и в мобильных приложениях. Корпоративные, интернет пользователи наоборот чаще выбирают сервисы RPC SOAP.

Для любителей REST предлагаются т.н. RESTful API – это API для Web-сервисов, удовлетворяющий принципам REST. В Visual Studio он называется Web API.

Сервисы: ASMX, gRPC, WCF

Технология ASMX уже давно не развивается, но продолжает широко использоваться. Google «видит» обсуждение сервисов:

212 000 ASMX, 23 000 WCF (2020)

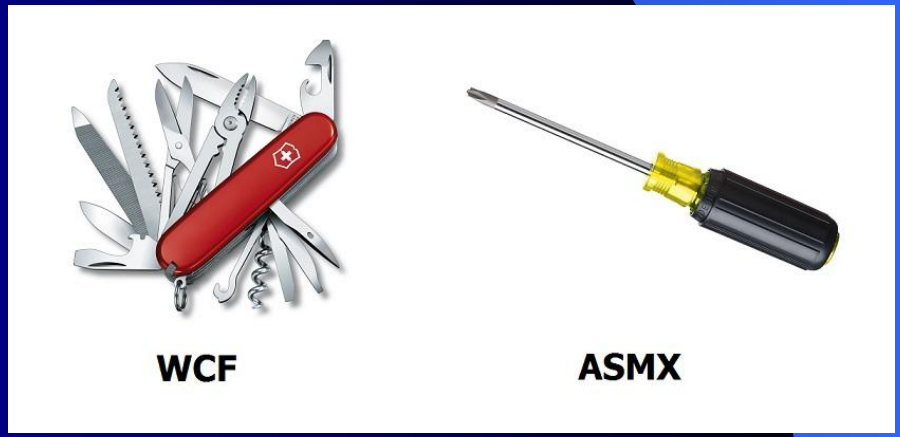
1 830 000 ASMX, 18 100 000 WCF, 3 660 000 gRPC (2021)

Плюсы ASMX:

- Лёгкость в изучении и разработке,
- Лёгкость конфигурирования.

Плюсы WCF:

- Актуальная технология,
- Разнообразные возможности транспорта данных и хостинга сервиса,
- Реализации множества стандартов безопасности, надёжности, сбоев, транзакционности...



WCF и .Net Core

WCF стала большой, громоздкой и только Windows средой, чтобы легко перенести в кроссплатформенную .Net Core . Кроме того, Microsoft выбрали модный сегодня RESTful API и в .Net Core пока отказались от SOAP.

Разработчиками .Net Foundation был принят порт (Core WCF) для .Net Core, что даёт сообществу WCF повод для оптимизма. Предстоит пройти долгий путь, прежде чем они смогут использовать все хорошее, что есть в экосистеме .Net Core.

Как «новая урезанная версия WCF» для .Net Core сегодня развивается **gRPC** — это фреймворк для скоростных микросервисов от Google. Основан на транспортном протоколе HTTP/2 (только HTTP) и протоколе Protobuf — контракт обмена данными (бинарная Google альтернатива JSON или SOAP XML), поддерживает более 10 языков для Windows, Linux, Mac.
… однако, реализация gRPC API намного медленнее, чем реализация популярного REST API (отсутствие встроенной поддержки gRPC в сторонних инструментах).

gRPC

Language	Platform	Compilers / SDK
C/C++	Linux, Mac	GCC 4.9+, Clang 3.4+
C/C++	Windows 7+	Visual Studio 2015+
C#	Linux, Mac	.NET Core, Mono 4+
C#	Windows 7+	.NET Core, NET 4.5+
Dart	Windows, Linux, Mac	Dart 2.2+
Go	Windows, Linux, Mac	Go 1.13+
Java	Windows, Linux, Mac	JDK 8 recommended (Jelly Bean+ for Android)
Kotlin/JVM	Windows, Linux, Mac	Kotlin 1.3+
Node.js	Windows, Linux, Mac	Node v4+
Objective-C	Mac OS X 10.11+, iOS 7.0+	Xcode 7.2+
PHP (beta)	Linux, Mac	PHP 5.5+, PHP 7.0+
Python	Windows, Linux, Mac	Python 2.7, Python 3.4+
Ruby	Windows, Linux, Mac	Ruby 2.3+

Protobuf и HTTP 2

Формат обмена сообщениями Protobuf:

- Независимость от платформы и языка, например как JSON;
- Сериализует и десериализует структурированные данные для передачи в двоичном формате по HTTP;
- Поскольку он является сильно сжатым форматом, он не обеспечивает удобочитаемости JSON, XML;
- Передача данных происходит быстрее (в 7-10 раз), потому что Protobuf уменьшает размер сообщений.

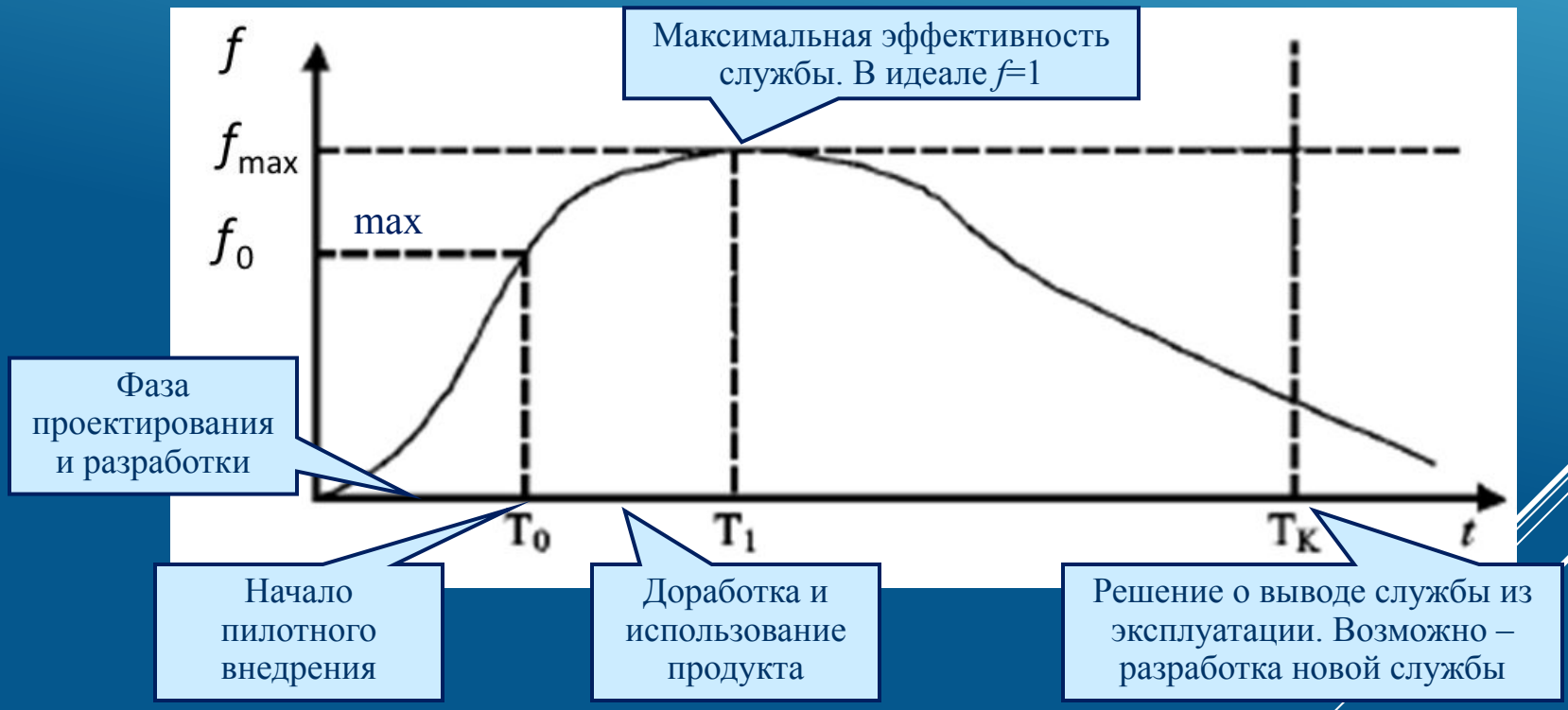
Передача и приём данных осуществляется через «заглушки» (точки доступа) сервиса и клиента с одинаковыми протоколами Protobuf.

gRPC использует протокол HTTP 2:

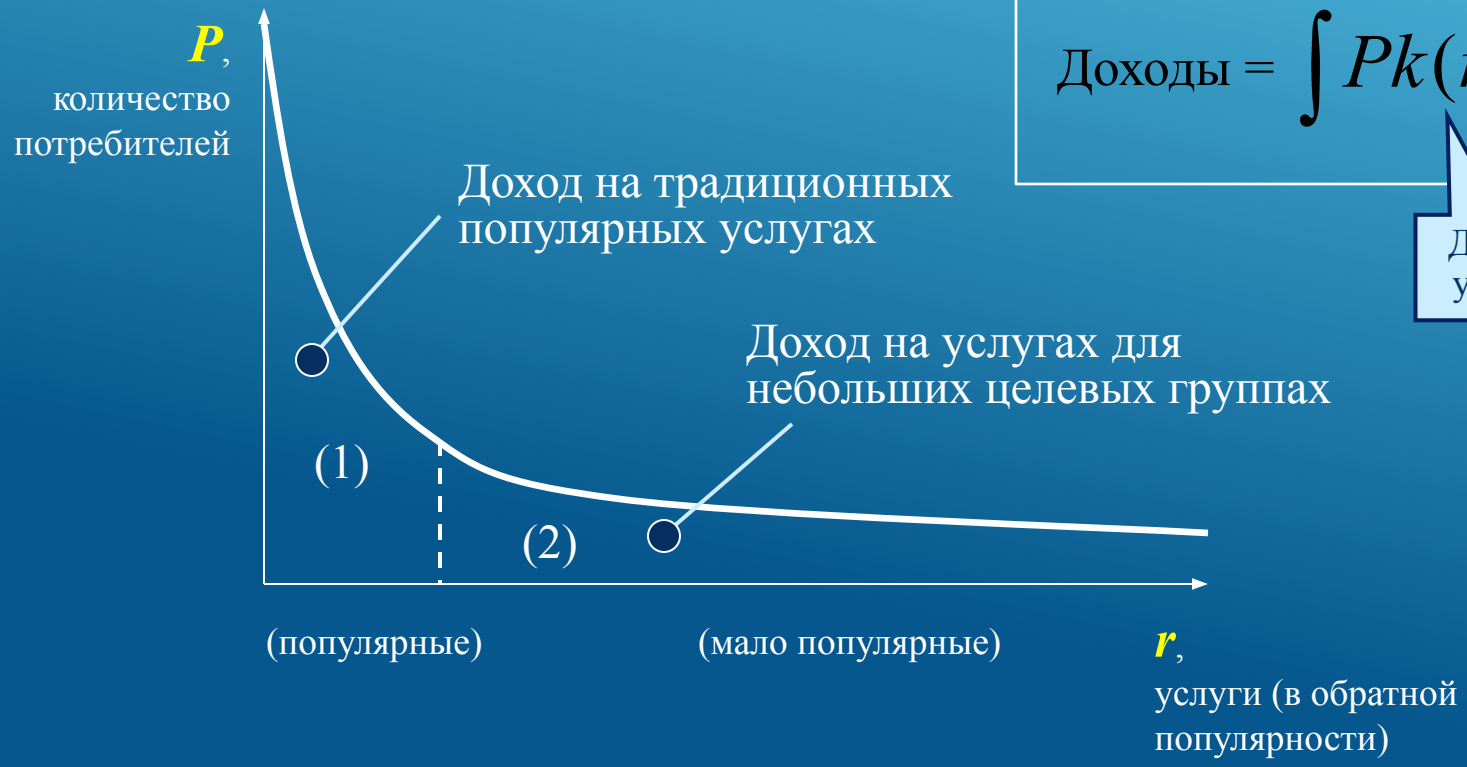
- Обслуживание множества запросов и ответов одновременно;
- Поддержка двунаправленной связи (долговременные соединения, или потоковая связь) между клиентом и сервисом. Клиент и сервис передают данные друг другу без особого порядка. Клиент инициирует такой вид двунаправленной потоковой передачи. Он же завершает соединение.

ОЦЕНКА ЭФФЕКТИВНОСТИ СЛУЖБЫ

Рассмотрим график $f(F,S)$ функции оценивающая соотношение между имеющейся функциональностью службы $F(t)$ и степенью удовлетворения бизнес-требованиям $S(t)$.



ОЦЕНКА ДОХОДОВ ОТ ВНЕДРЕНИЯ.
ГРАФИК «ДЛИННОГО ХВОСТА»



$$\text{Доходы} = \int Pk(r)dr$$

Доход от услуги r

Доходы (2) соизмеримы с популярными (1). Разрабатывать ИС в области (2) невыгодно, а модернизировать службы – быстро и дёшево, их можно предложить «глобально широкой» аудитории, т. е. – выгодно!

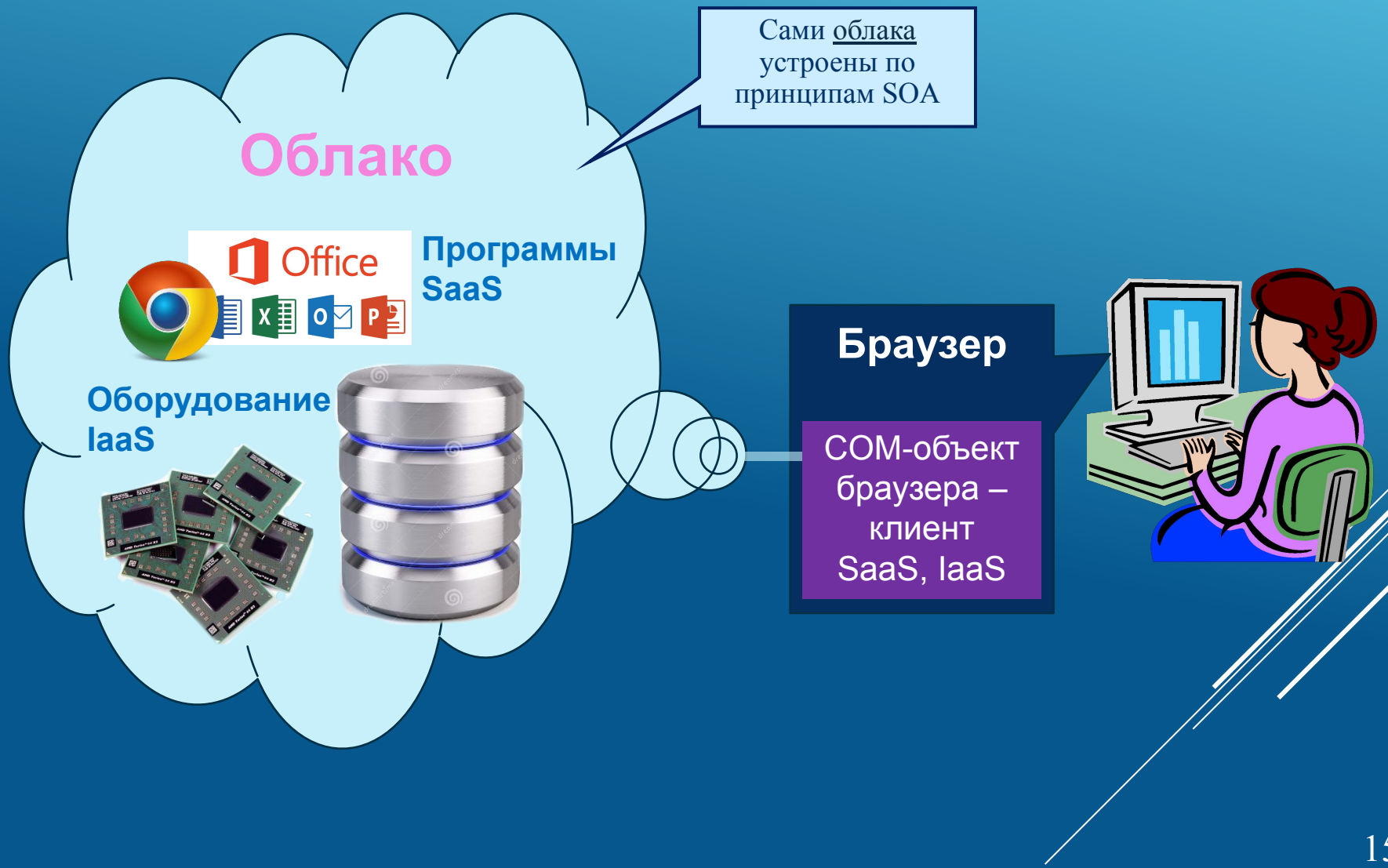
БИЗНЕС-МОДЕЛИ В SOA

Сегодня выделяют (National Institute of Standards and Technology)

Три основные бизнес-модели для разработчиков сервисов:

1. **Software as a service (SaaS)** – приложения, которые поставляется конечному пользователю как службы через Internet,
2. **Platform as a service (PaaS)** – платформа разработки и развертывания приложений поставляется в виде службы для разработчиков, позволяющей быстро создавать и развертывать приложения SaaS,
3. **Infrastructure as a service (IaaS)** – оборудование, такое как вычислительные серверы, системы хранения и сетевые элементы, предоставляются в виде служб.

ПРИМЕР СХЕМЫ БИЗНЕС-МОДЕЛИ SOA



ЕДИНОЕ ПРОГРАММНОЕ ЯДРО В ОБЛАКЕ ДЛЯ SaaS

Ключевым фактором, объясняющим экономическую целесообразность SaaS, является «эффект масштаба» — провайдер SaaS обслуживает единое программное ядро, которым пользуются все клиенты, и потому тратит меньшее количество ресурсов в сравнении с управлением отдельными копиями программного обеспечения для каждого заказчика.

Кроме того, использование единого программного ядра позволяет планировать вычислительные мощности и уменьшает пиковые нагрузки для отдельных заказчиков. Все это позволяет SaaS-провайдерам существенно снизить стоимость эксплуатации ПО. В результате стоимость услуг для конечного пользователя такого ПО становится ниже издержек, возникающих при использовании классической модели лицензирования.

SaaS-провайдер способен предложить уровень обслуживания и поддержки ПО в работоспособном состоянии, недоступный для внутренних IT-отделов компаний.

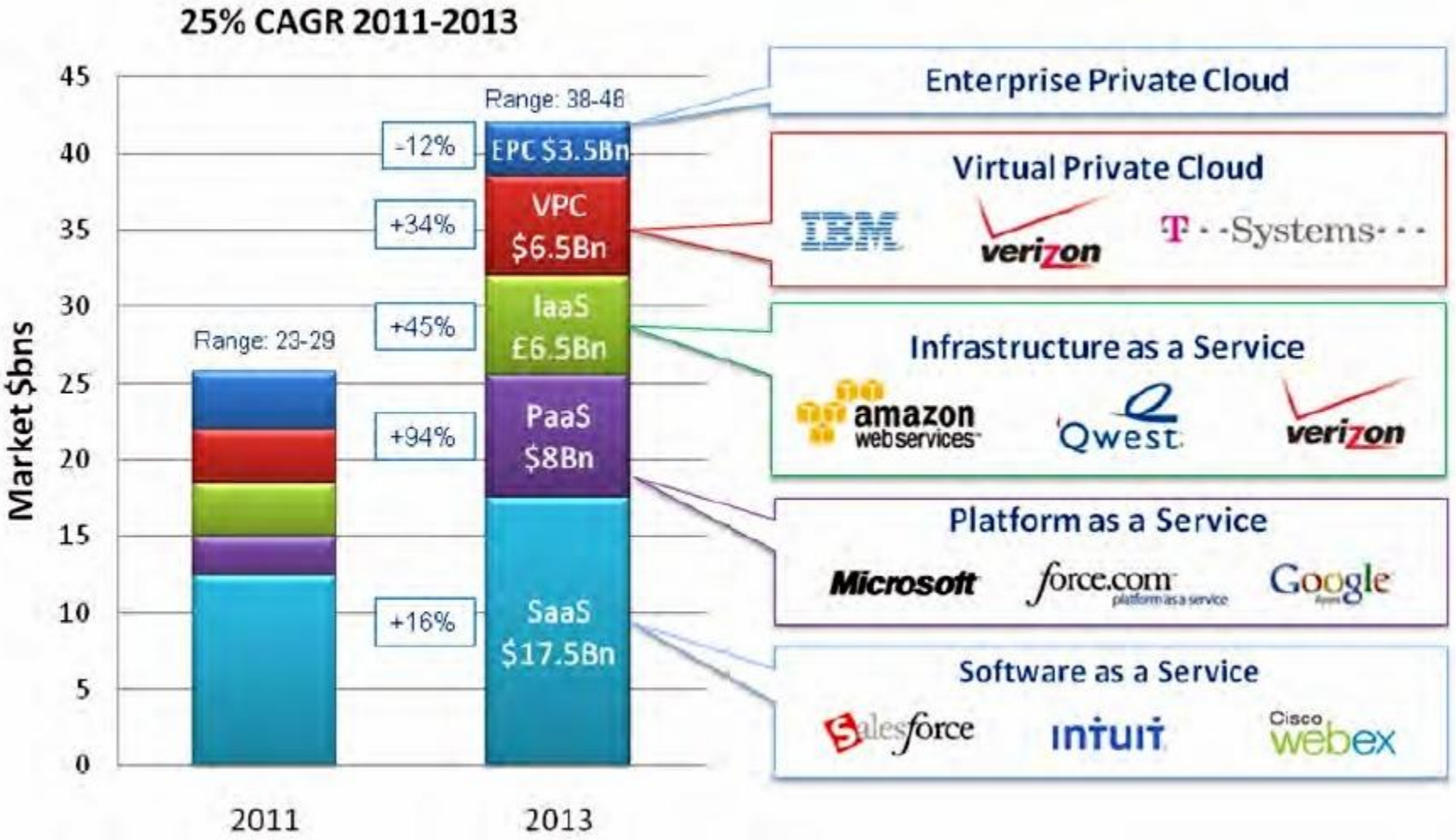
ПОЛОЖИТЕЛЬНЫЕ ФАКТОРЫ SaaS ДЛЯ ПОЛЬЗОВАТЕЛЕЙ

- Не нужна установка ПО на рабочие места пользователей — доступ к ПО осуществляется через обычный Web-браузер;
- Радикальное сокращение затрат на развёртывание системы в организации. Это расходы на аренду помещения, организацию дата-центра, оплату труда сотрудников и т. д.;
- Сокращение затрат на техническую поддержку и обновление развернутых систем (вплоть до их полного отсутствия);
- Скорость внедрения, обусловленная отсутствием затрат времени на развертывание системы;
- Понятный интерфейс — большинство сотрудников уже привыкли к использованию веб-сервисов;
- Ясность и предсказуемость платежей, защита инвестиций;
- Мультиплатформенность — пользователь не зависит от программно-аппаратной платформы, выбранной разработчиком;
- Возможность получить более высокий уровень обслуживания ПО.

ПОЛОЖИТЕЛЬНЫЕ ФАКТОРЫ SaaS ДЛЯ РАЗРАБОТЧИКОВ

- Рост популярности веб-сервисов для конечных пользователей;
- Быстрые процессы внедрения и сравнительно низкие затраты ресурсов на обслуживание конкретного клиента;
- Лёгкое проникновение на глобальные рынки (в конце «хвоста»);
- Отсутствие проблем с нелицензионным распространением ПО;
- В отличие от классической модели, SaaS-клиент привязывается к разработчику — он не может отказаться от услуг разработчика и продолжать использовать систему. Таким образом, обеспечивается защита инвестиций разработчика в процесс продаж;
- В долгосрочном периоде доходы от SaaS могут превысить доходы от продаж лицензий и оказания технической поддержки;
- Разработчик выбирает рабочую программно-аппаратную платформу из соображений её технико-экономической эффективности а не из соображений её распространенности у возможных пользователей ПО.

Cloud services: market forecast and current players



Source: Bain Analysis, Forrester, IDC, Gartner, William Blair & Co.

ДОЛЯ SaaS В ОБОРОТЕ ПО