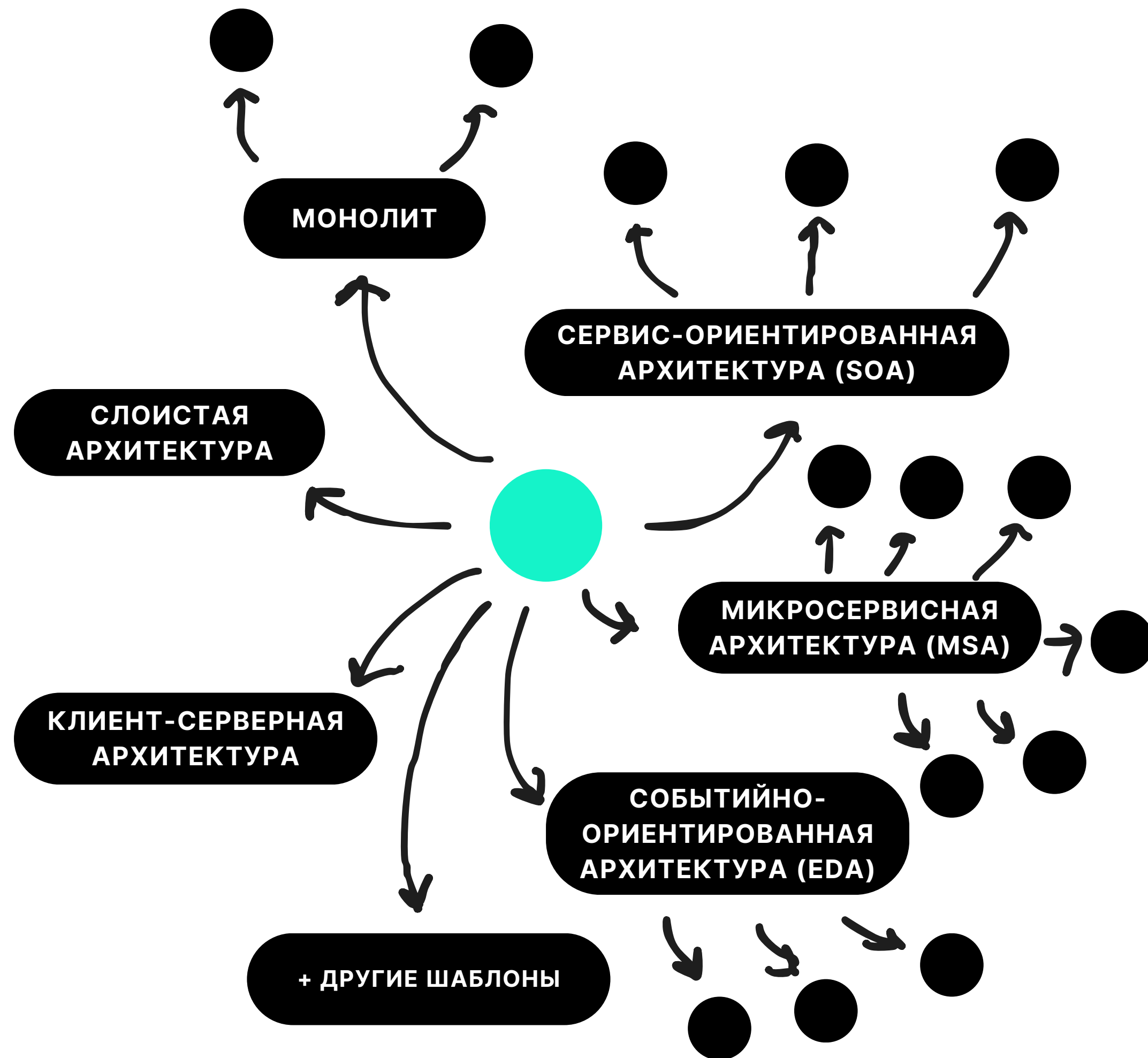


7 ОСНОВНЫХ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ АРХИТЕКТУРЫ

МИНИ-КНИГА ДЛЯ СИСТЕМНЫХ АНАЛИТИКОВ

ЕКАТЕРИНА АНАНЬЕВА
GETANALYST.RU



Системные аналитики описывают внутреннюю логику работы приложений: связи между данными на пользовательских экранах, в базе данных и API, интеграции с другими системами, алгоритмы обработки данных и другие технические детали.

В простых проектах архитектура может не вызывать особых вопросов, так как обычно используется один из вариантов монолитной архитектуры. А вот в сложных продуктовых компаниях, таких как банки, маркетплейсы и страховые компании, базовых знаний недостаточно. Здесь чаще встречается сложная сервисная или микросервисная архитектура.

Для таких проектов аналитикам важно понимать архитектуру систем, чтобы грамотно подходить к проектированию новых функций и обеспечивать их корректную интеграцию в существующую "экосистему" проекта.

В этой книге вы найдете **7 основных шаблонов проектирования архитектуры, которые важно знать и понимать СА.**

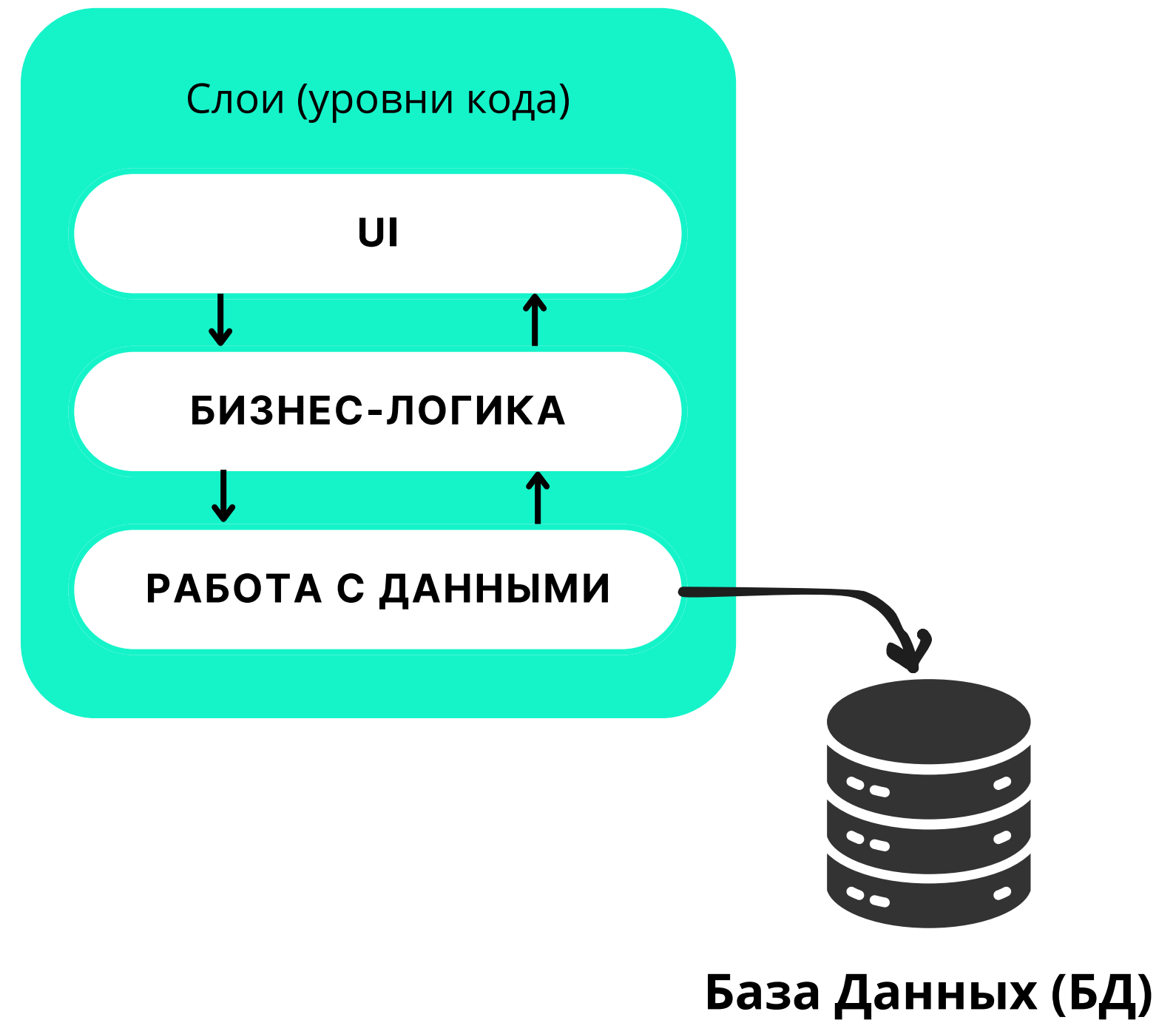
1. МОНОЛИТ

Monolithic architecture

Подход к построению программного обеспечения, при котором все функции и компоненты системы объединены в один единый блок.

В такой архитектуре весь **код приложения работает как одно целое**, и все части программы тесно связаны между собой, что упрощает разработку и развертывание на начальных этапах, но затрудняет масштабирование и внесение изменений в будущем.

Монолитное приложение

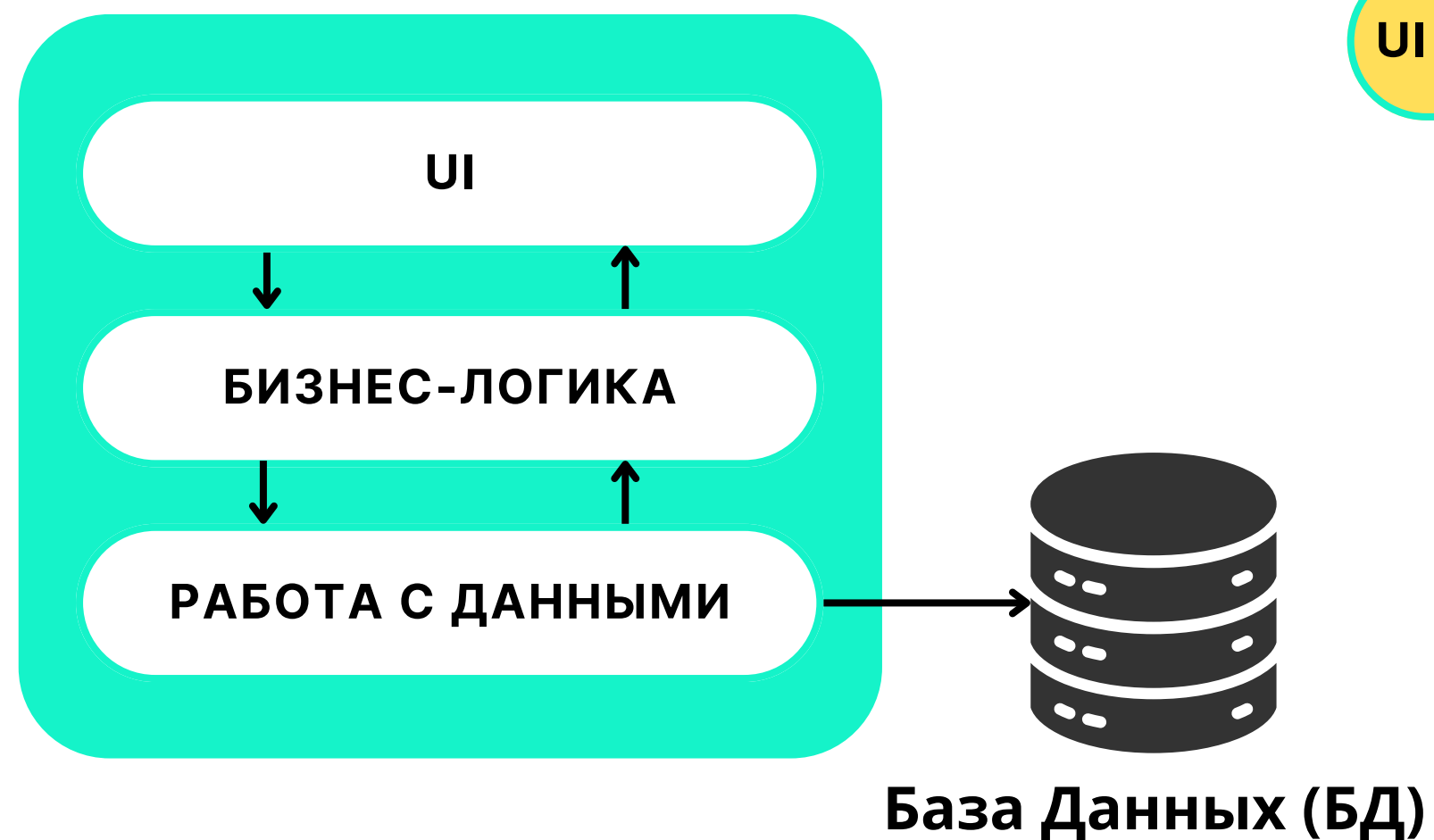


1. МОНОЛИТ

Monolithic architecture

Классический монолит

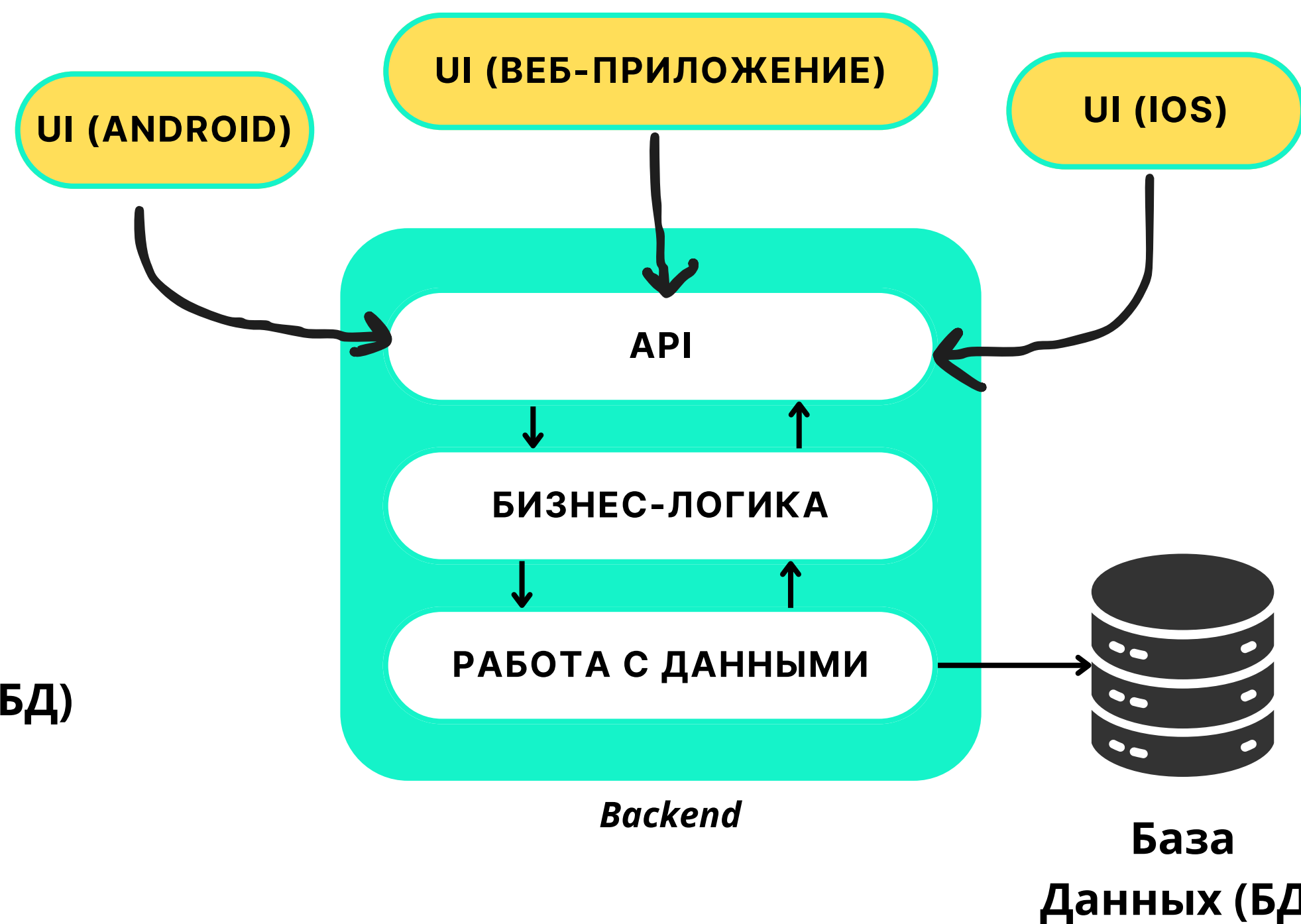
Код UI (веб-приложения) и сервера
в одном месте



Современный монолит

Код UI и сервера разделен, есть API (например, REST API)

Желтые компоненты - Frontend-ы



1. МОНОЛИТ

Monolithic architecture

Что важно знать системному аналитику

- Работа системного аналитика на проектах с классическим монолитом сводится к минимуму:
 - собрать требования,
 - описать экран и сделать его самому или с дизайнером,
 - описать поведение экрана и передать разработчикам, чтобы далее во всех “внутренностях” разбирались они.
- В проектах с современной монолитной архитектурой от аналитика будут требовать знание и понимание REST API, так как его чаще всего выбирают для реализации. Аналитик будет:
 - описывать вызовы API с UI при постановке задач, а не просто поведение экрана,
 - делить задачи на Backend (сервер и БД) и Frontend (UI - работа экранов).
- Многие классические монолитные проекты простые и для них достаточно нанять Junior CA, которому не обязательно знать БД и API.

2. СЛОИСТАЯ АРХИТЕКТУРА

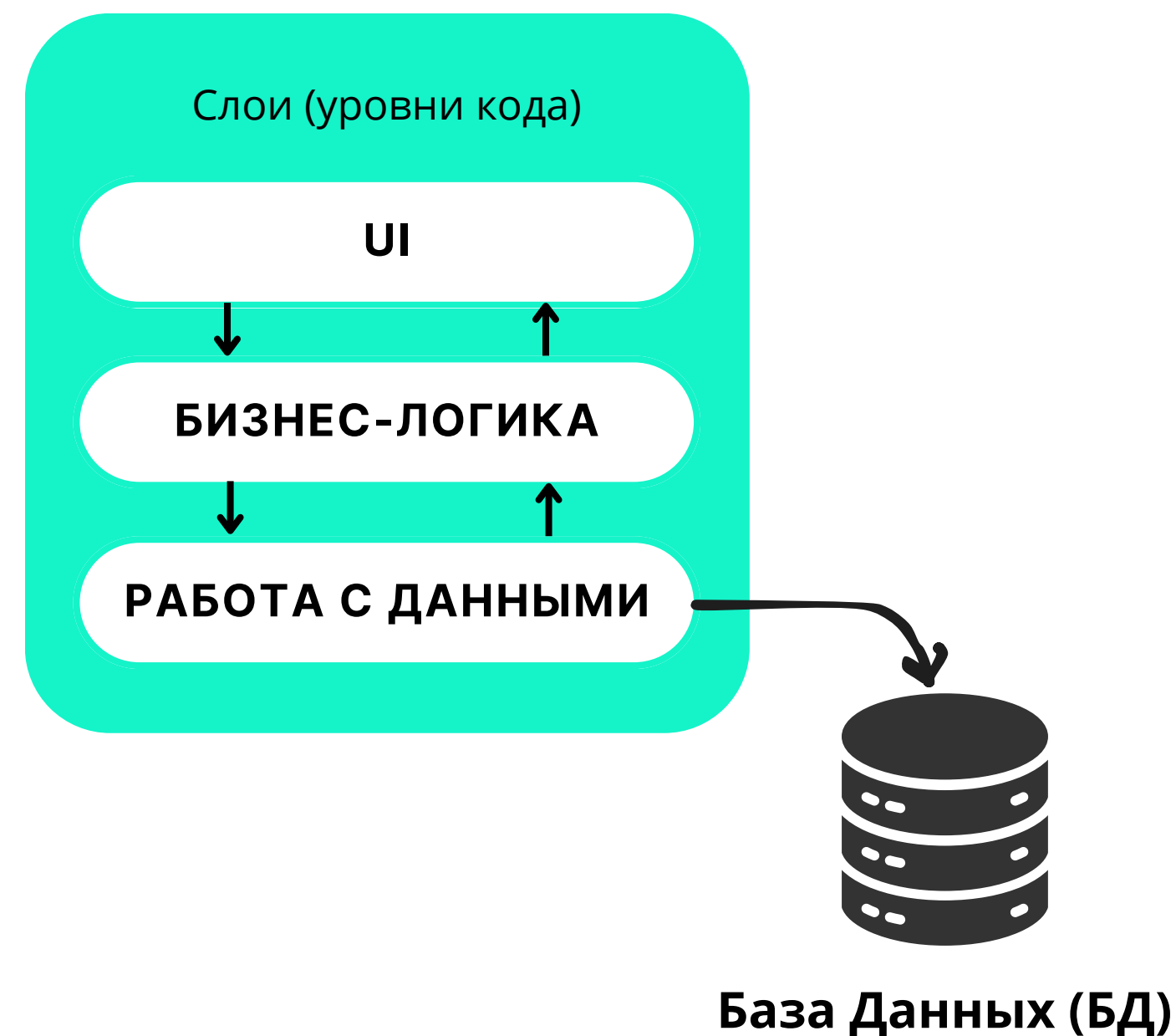
Layered Architecture Pattern

Архитектурный подход, в котором приложение разделено на уровни (слои), каждый из которых выполняет определённую функцию и отвечает за свой набор задач.

Например, обычно выделяют слой представления (интерфейс пользователя), бизнес-логику (обработка данных), слой доступа к данным и слой хранения данных (база данных).

Каждый слой взаимодействует только с соседними слоями, что упрощает тестирование, поддержку и замену отдельных частей системы без изменения всей структуры приложения.

Монолитное приложение



2. СЛОИСТАЯ АРХИТЕКТУРА

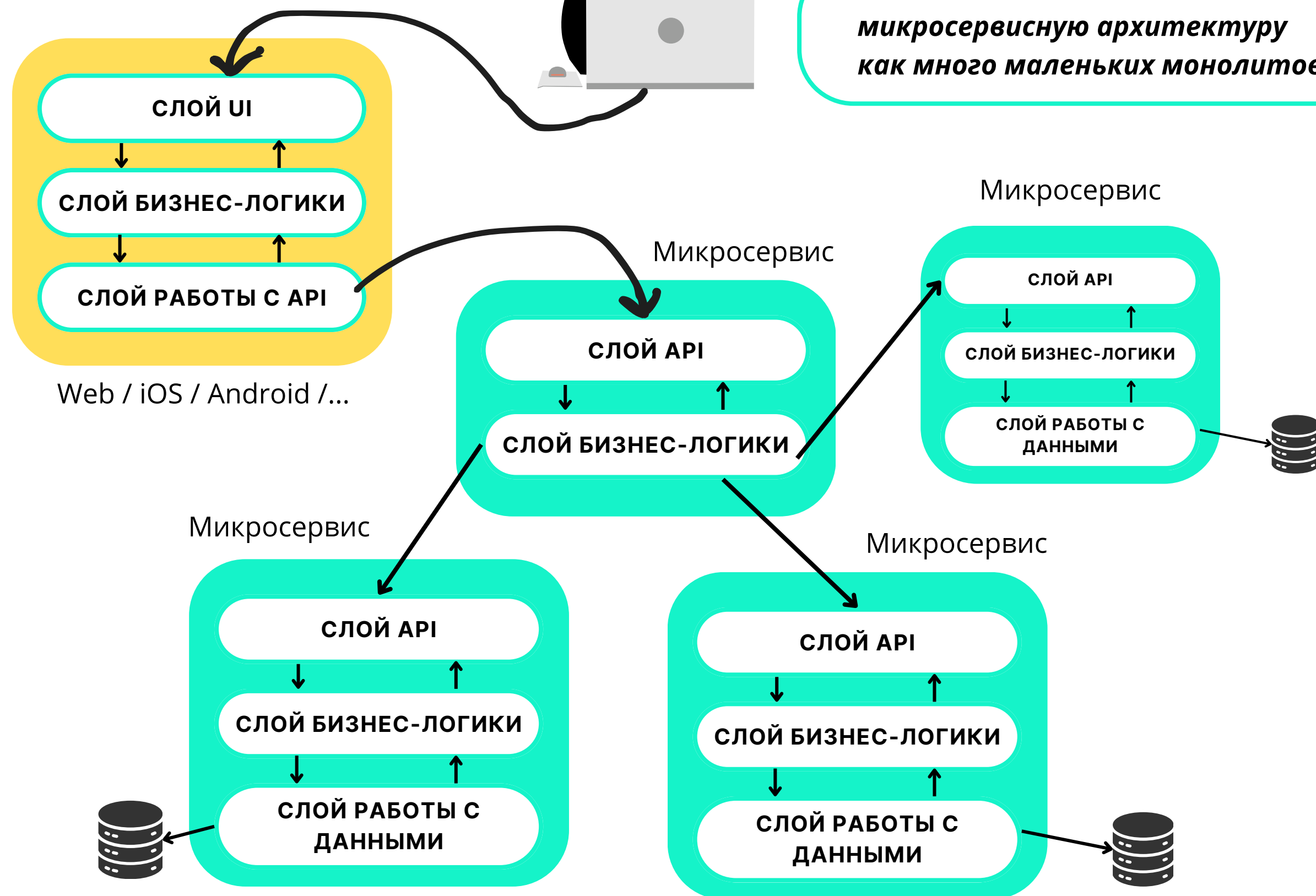
Layered Architecture Pattern

Слоистая архитектура часто используется как часть монолитной архитектуры, особенно в традиционных приложениях, где все слои собраны в один монолитный блок, работающий как единое целое.

Однако слоистая архитектура может применяться и в других архитектурных подходах.



P.S. Пока понимайте микросервисную архитектуру как много маленьких монолитов



2. СЛОИСТАЯ АРХИТЕКТУРА

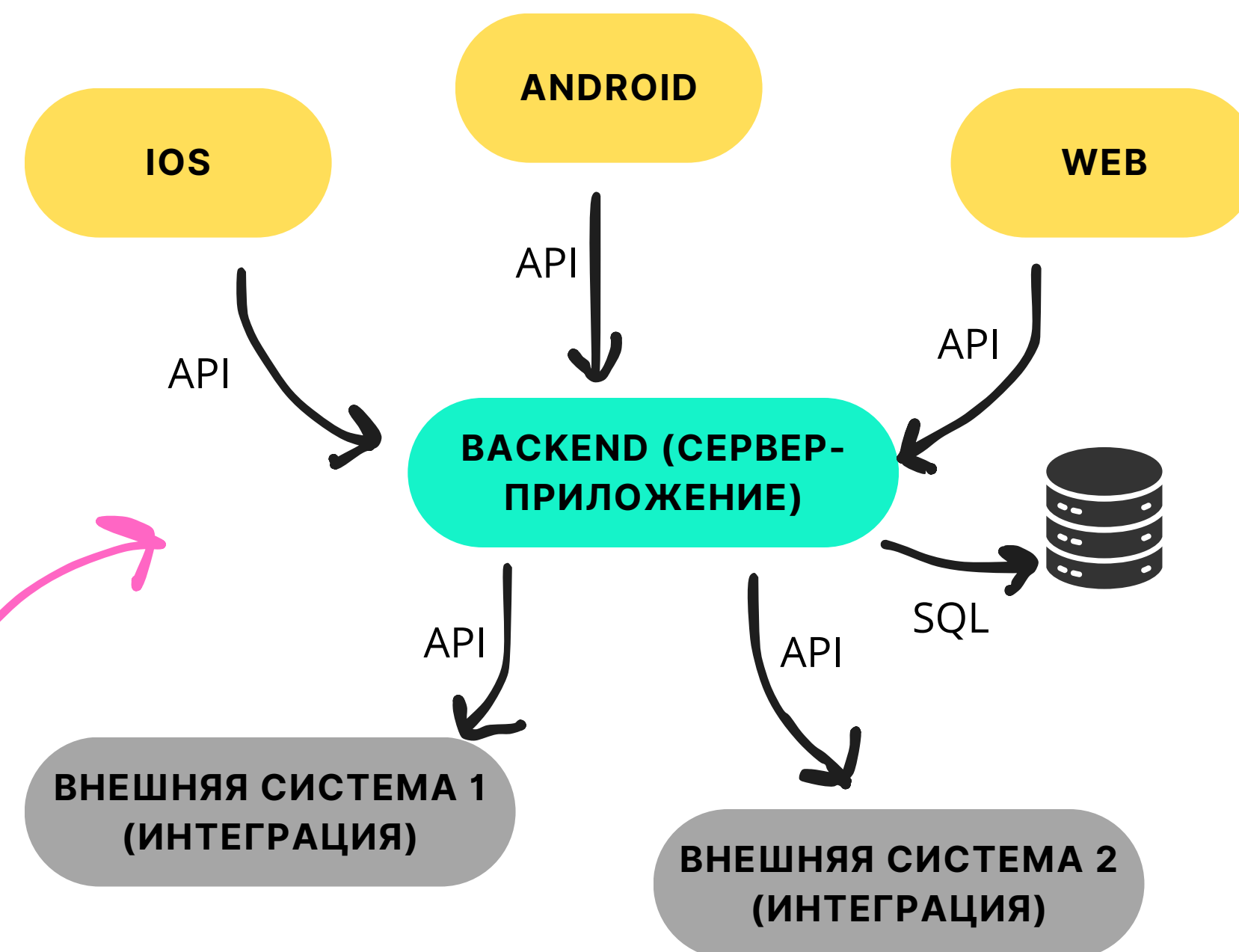
Layered Architecture Pattern

Что важно знать системному аналитику

Слоистая архитектура больше важна для разработчиков, с точки зрения организации программного кода.

На общей схеме архитектуры IT-продукта слои кода обычно отражают на более детализированных уровнях (*пример: C4 / Component*), но не на общем.

Системному аналитику который развивается в Solutions Architect полезно понимать этот шаблон, но фактическую его реализацию можно проверить только через код.



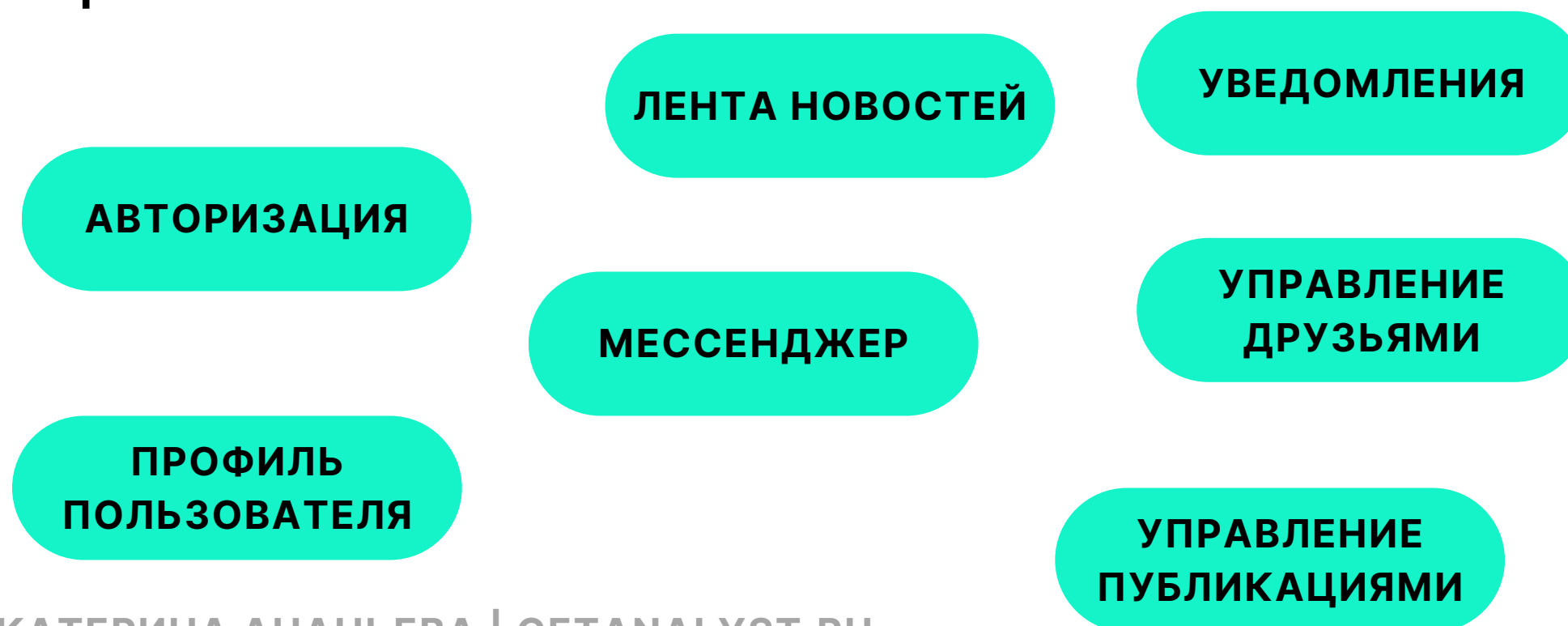
3. МОДУЛЬНАЯ АРХИТЕКТУРА

Modular Architecture

Это подход к построению программного обеспечения, при котором система делится на независимые модули, каждый из которых отвечает за определённую функцию или часть функциональности.

Модули можно выделять как в сервер-приложениях Backend, так и во Frontend-приложениях с UI.

Пример (сначала подумайте про контекст мобильного приложения, а потом про контекст сервера):
Социальная сеть



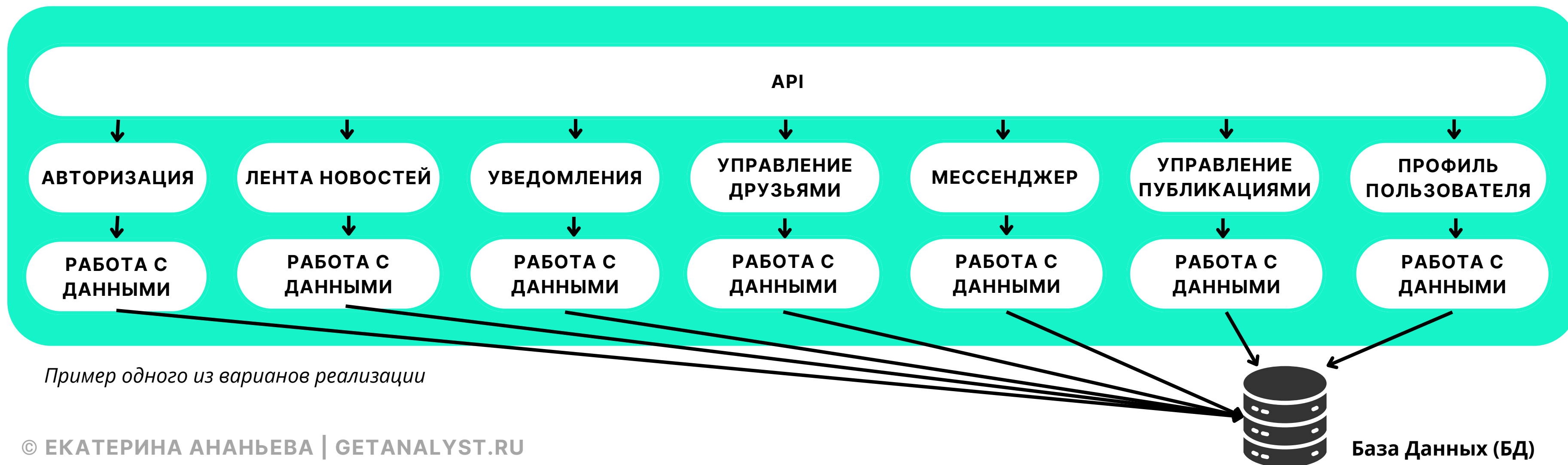
Каждый модуль решает свои задачи, и изменения в одном модуле (например, добавление новых функций в модуль сообщений) не требуют изменений в других модулях.

3. МОДУЛЬНАЯ АРХИТЕКТУРА

Modular Architecture

Модульная архитектура может быть реализована совместно с другими архитектурными шаблонами.

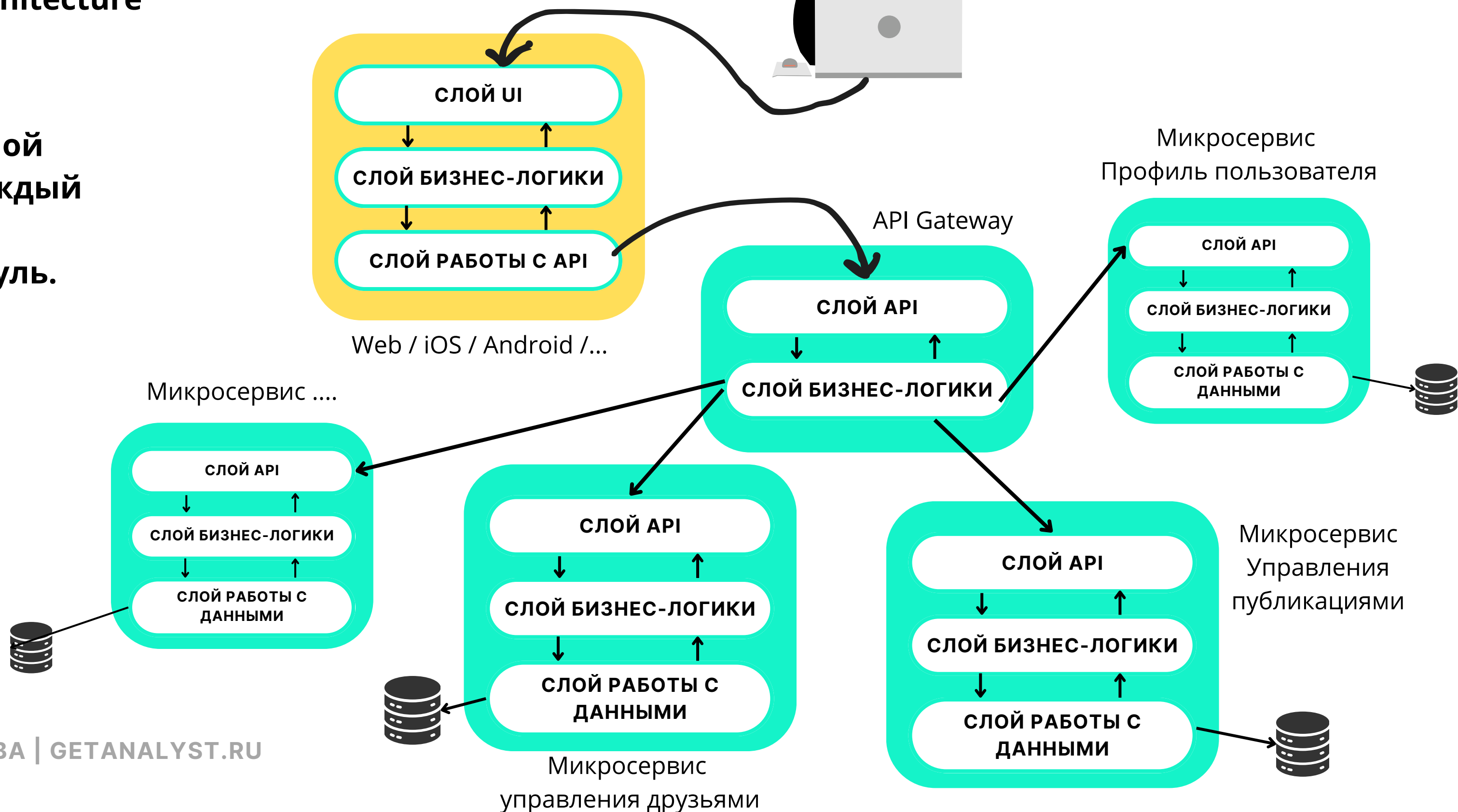
В монолите каждый модуль - это модуль монолита. Монолит, разделенный на модули, имеет шаблон архитектуры именуемый **“Модульный монолит”**.



3. МОДУЛЬНАЯ АРХИТЕКТУРА

Modular Architecture

В микросервисной архитектуре каждый микросервис - отдельный модуль.



3. МОДУЛЬНАЯ АРХИТЕКТУРА

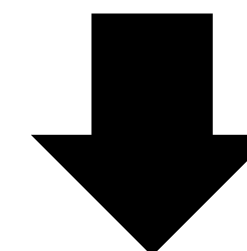
Modular Architecture

Что важно знать системному аналитику

Системного аналитика могут привлекать к процессу декомпозиции системы на модули.

Обычно необходимость в этом возникает, когда проект разрастается и настает время деления монолита на микросервисы.

Системные аналитики играют здесь ключевую роль, так как именно они лучше всех разбираются в предметной области, контекстах системы и знают все её данные и функциональность. За счет этого они помогают не упускать детали, и логически правильно определить все нужные модули и взаимодействие между ними.



МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

МИКРОСЕРВИС

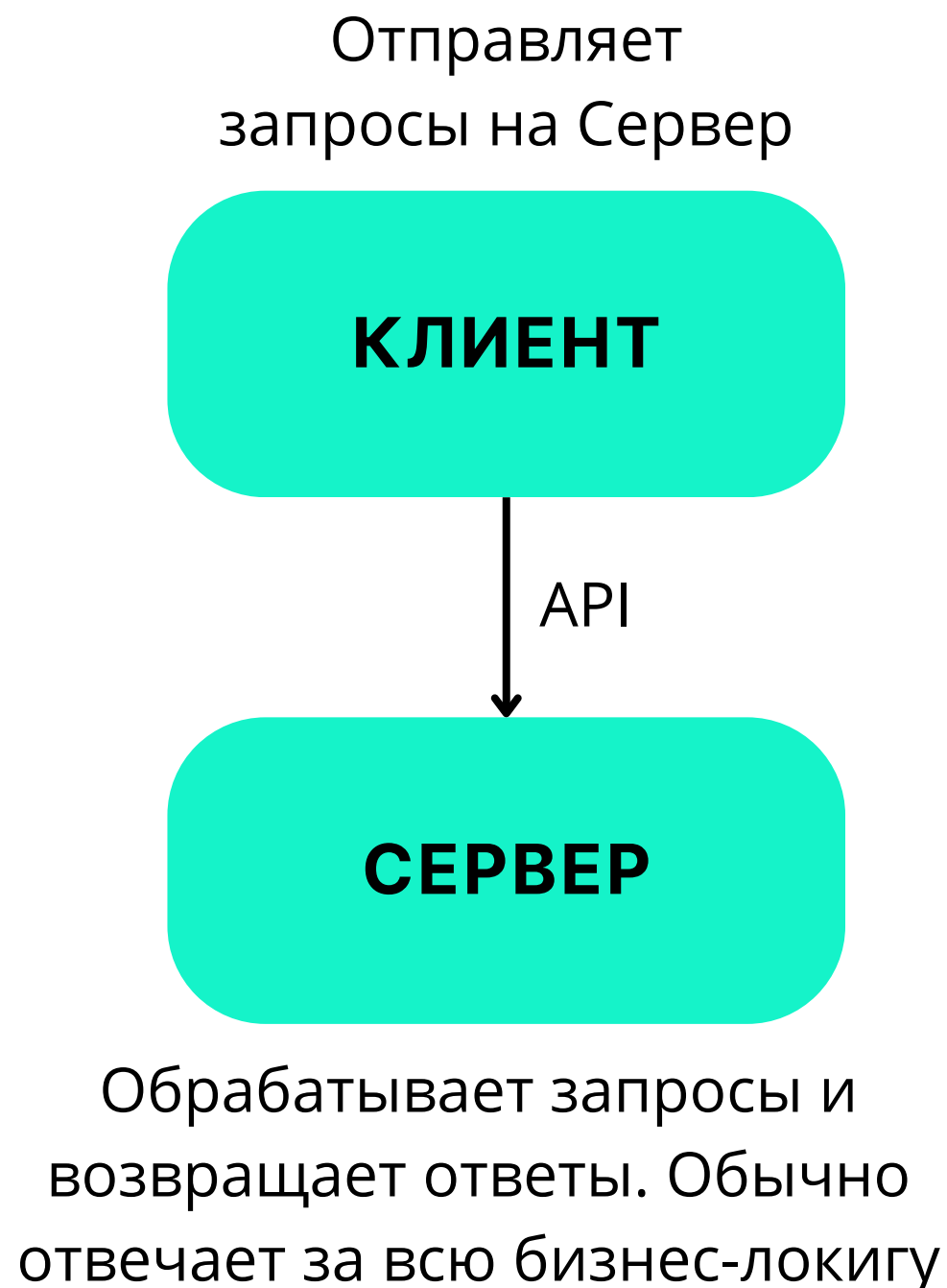
МИКРОСЕРВИС

4. КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

Это способ организации программного обеспечения, при котором система делится на две основные части: клиент и сервер.

- **Клиент** — это приложение, система или устройство, которое запрашивает данные или услуги у сервера.
- **Сервер** — обрабатывает эти запросы и отправляет ответ клиенту.

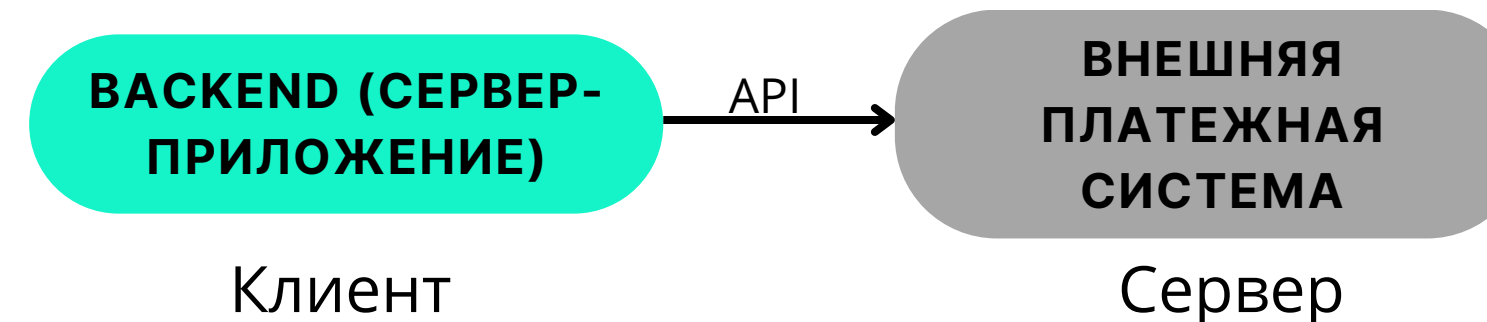
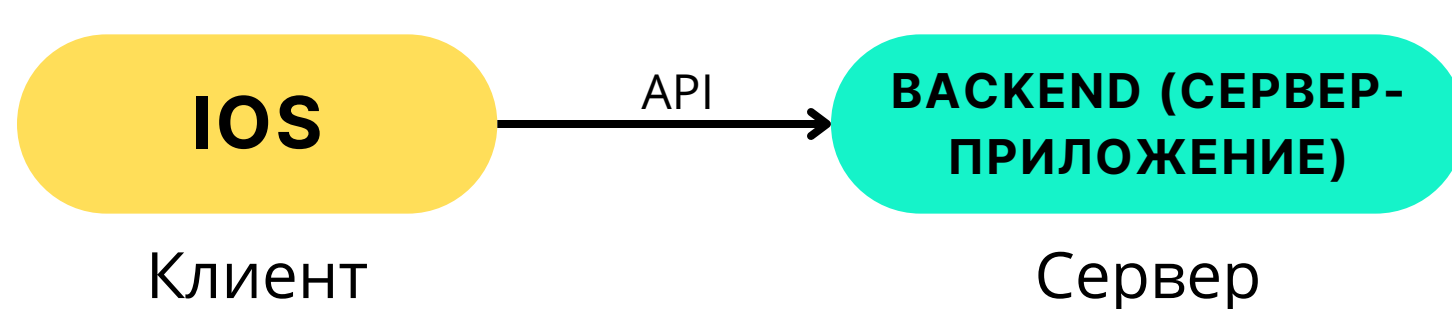
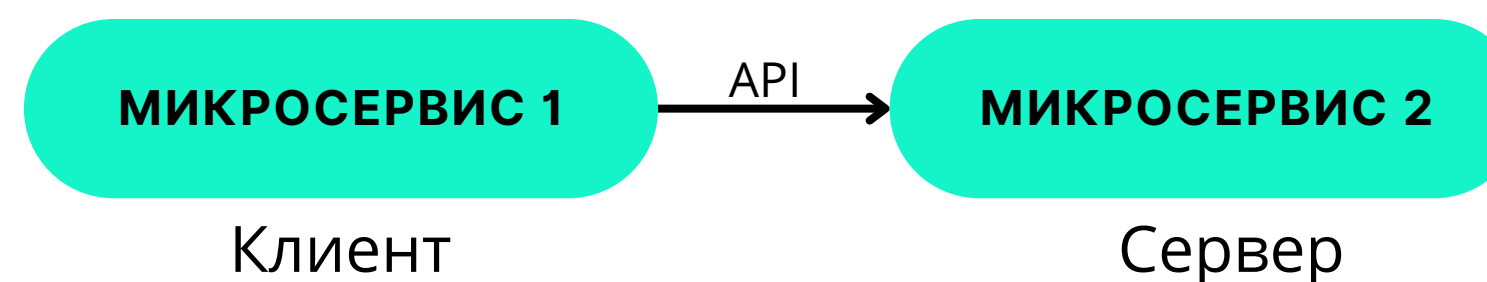
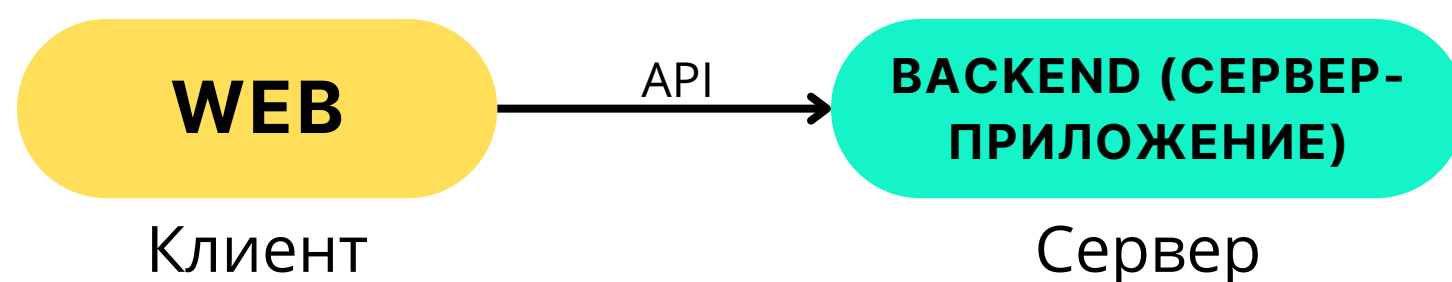
Этот подход позволяет разделить функции между устройствами, оптимизируя обработку данных и улучшая взаимодействие.



4. КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

Важно понимать, что в клиент-серверном взаимодействии клиентом и сервером могут выступать абсолютно разные компоненты системы, а также внешние системы.

Все системы, которые не являются классическими монолитами или имеют интеграции с внешними системами используют клиент-серверный подход для взаимодействия.



4. КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

Что важно знать системному аналитику

Системному аналитику важно понимание клиент-серверного подхода в архитектуре, так как он наиболее популярен в современной разработке.

Следующие принципы важны для постановки задач в такой архитектуре:

- “Тонкий клиент” - клиент делает минимальные проверки вводимых пользователем данных и реализует минимум алгоритмов. Вся сложная логика и обработка данных отправляется на сервер.
- При описании взаимодействия важно делить задачи и не добавлять в постановку задачи на клиента логику сервера. Для клиента процессы, реализуемые на сервере, не должны быть известны.

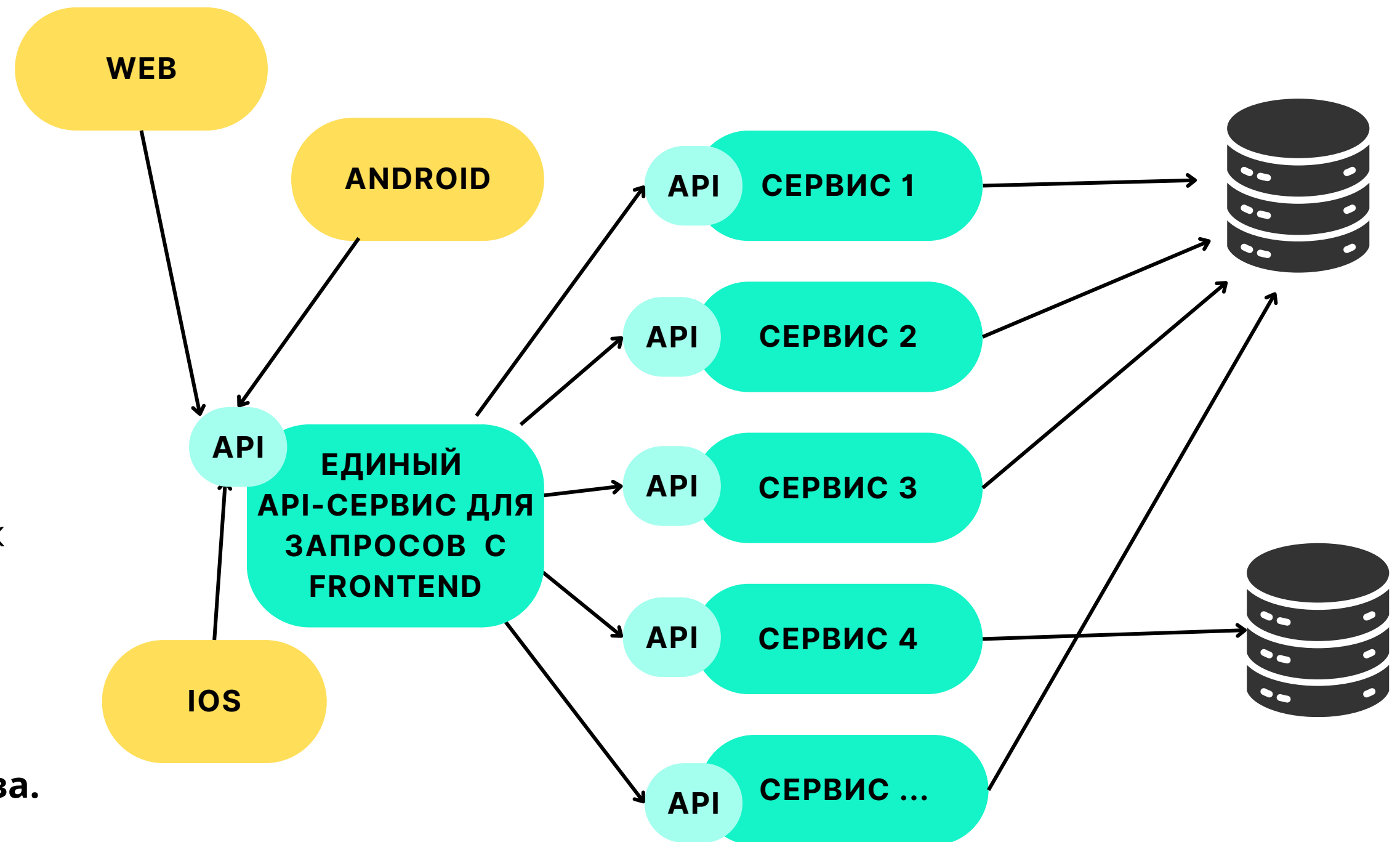
5. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (SOA)

Service Oriented Architecture

Подход к созданию программных систем, в котором функциональность приложения разделена на отдельные сервисы. Каждый сервис выполняет конкретную задачу и взаимодействует с другими сервисами по сети, например, через API.

Сервисная архитектура помогает создавать гибкие и масштабируемые приложения, так как **каждый сервис можно разрабатывать, развертывать и обновлять независимо от других.**

Каждый сервис = Отдельная кодовая база. Сервисы могут иметь общие БД.



Пример одного из вариантов реализации

5. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (SOA)

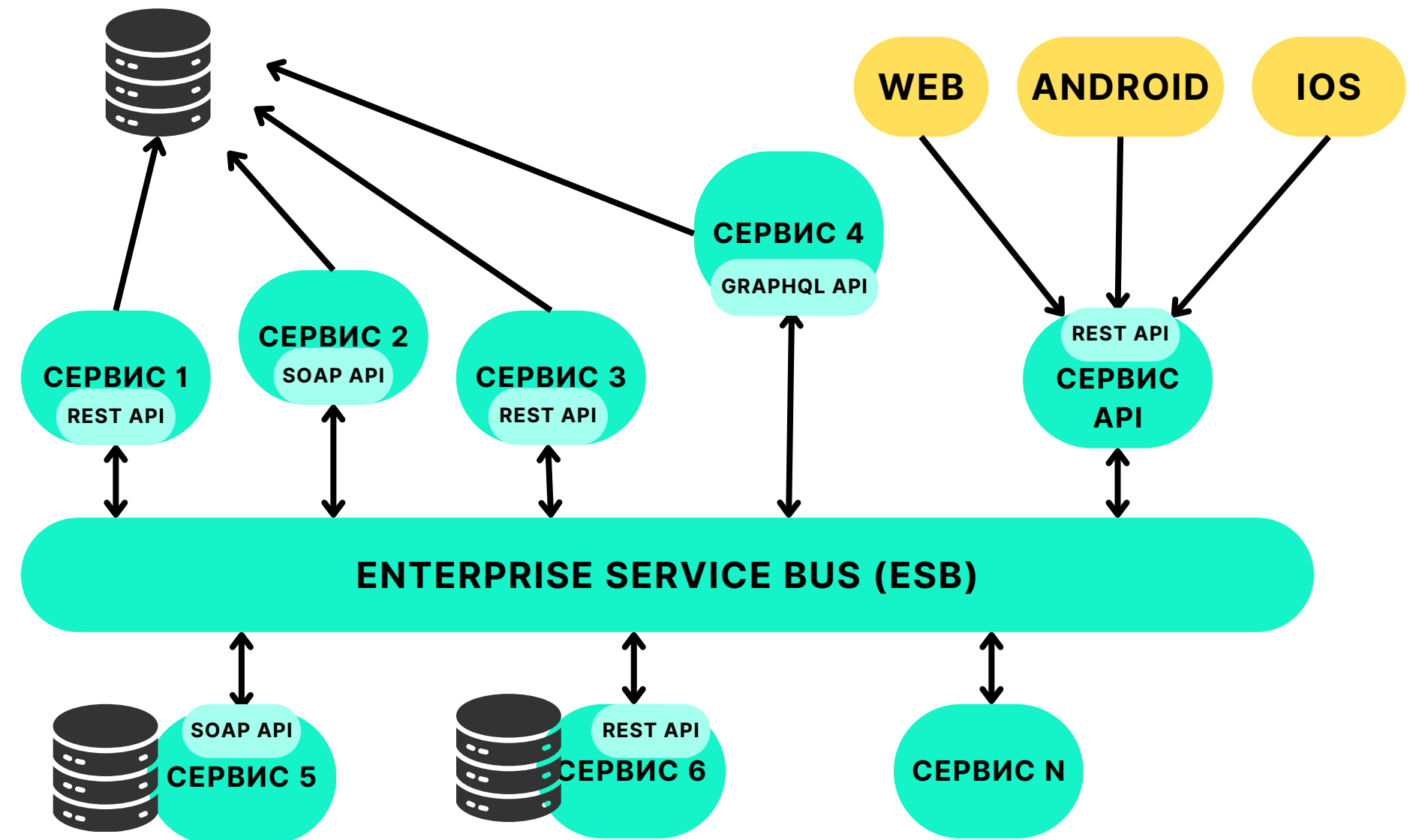
Service Oriented Architecture

В такой архитектуре может присутствовать **шина корпоративных данных (Enterprise Service Bus, ESB)** — это архитектурный компонент, который используется для обмена данными между различными сервисами и приложениями в рамках сложной системы.

Шина данных действует как центральный канал, через который проходят все сообщения и запросы между сервисами, обеспечивая единый способ коммуникации и интеграции.

Основная задача шины данных — упростить интеграцию в системах, где множество сервисов должны обмениваться информацией. Вместо того чтобы каждый сервис напрямую взаимодействовал со всеми другими, они общаются через шину. Это снижает сложность и количество соединений, повышает гибкость системы и упрощает добавление новых сервисов.

Пример одного из вариантов реализации



5. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (SOA)

Service Oriented Architecture

Что важно знать системному аналитику

Понимание принципов организации SOA систем позволит системному аналитику правильно и четко описывать задачи на разработчиков серверной части приложений (Backend), где важно:

- детально расписать внутреннее межсервисное взаимодействие и преобразование форматов данных,
- правильно определять в какой сервис относить новую функциональность,
- в какой БД нужно сделать доработки, чтобы хранить новые данные, которых ранее не было,
- понимать, какие подсистемы будут недоступны в процессе реализа, и как это повлияет на пользователей.

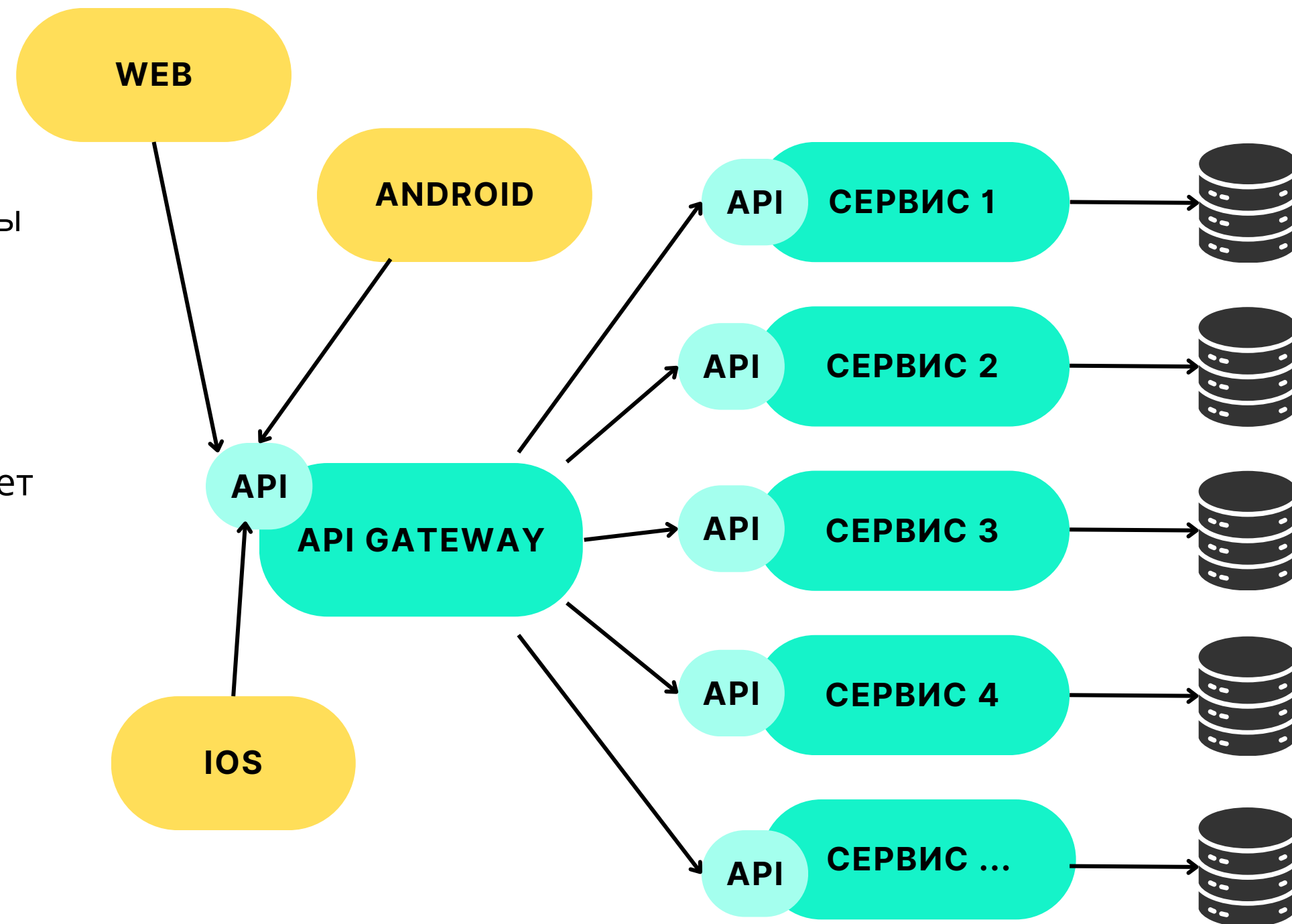
6. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА (MSA)

Microservices Architecture

Подход к созданию программного обеспечения, при котором приложение делится на небольшие независимые сервисы (микросервисы).

Каждый микросервис отвечает за выполнение конкретной функции, работает автономно, имеет свою БД и взаимодействует с другими микросервисами через четко определенные интерфейсы (обычно через API).

Этот подход позволяет легко обновлять, масштабировать и развивать отдельные части приложения без влияния на всю систему.



Пример одного из вариантов реализации

6. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА (MSA)

Microservices Architecture

Главное отличие от сервисной архитектуры - у каждого микросервиса своя БД, что при обновлении одной БД делает недоступным только один сервис, а не сразу несколько.

Дополнительные шаблоны, которые применяются вместе с MSA:

- Хореография
- Оркестрация
- API Gateway
- Душитель
- БД на сервис
- Saga
- Event-Driven Architecture (Событийно-ориентированная архитектура)
- и другие.

6. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА (MSA)

Microservices Architecture

Главное отличие от сервисной архитектуры - у каждого микросервиса своя БД, что при обновлении одной БД делает недоступным только один сервис, а не сразу несколько.

Дополнительные шаблоны, которые применяются вместе с MSA:

- **Хореография**
- **Оркестрация**
- **API Gateway**
- **Event-Driven Architecture (Событийно-ориентированная архитектура)**
- Душител
- и другие.

Выделенные шаблоны основные, которые важно и нужно знать при работе с MSA или при переезде на неё с монолита.

6. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА (MSA)

Microservices Architecture

Что важно знать системному аналитику

Понимание принципов организации MSA систем позволит системному аналитику правильно и четко описывать задачи на разработчиков серверной части приложений (Backend).

Всё, что справедливо для знания SOA, также справедливо и для MSA.

Сегодня всё больше проектов переходят от монолитной к микросервисной архитектуре. **При найме на проекты с MSA обычно берут только опытных Middle+ и Senior системных аналитиков,** которые уже знакомы с принципами проектирования и работы с микросервисной архитектурой.

7. СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (EDA)

Event-Driven Architecture

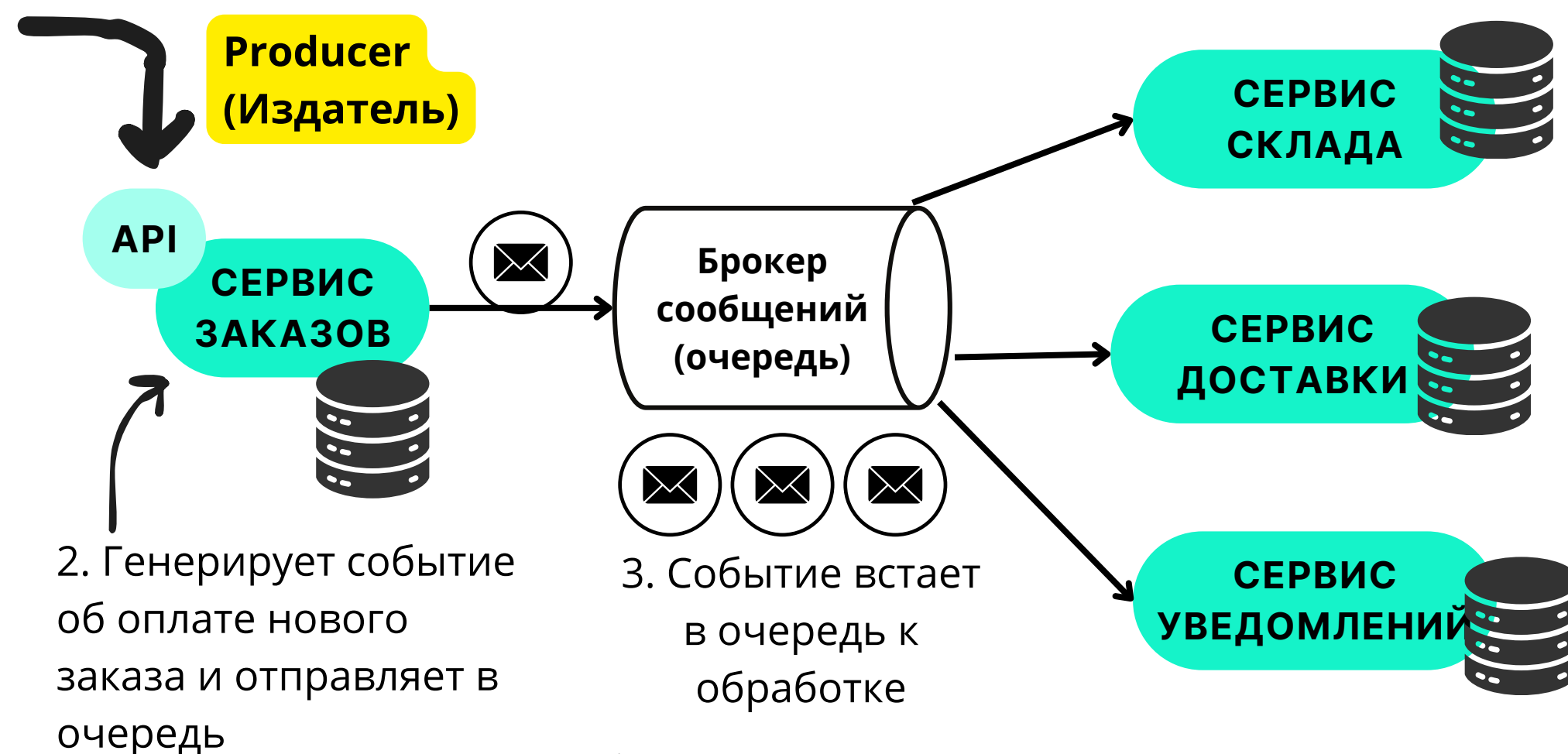
Подход, при котором взаимодействие между компонентами системы происходит через события.

Событие — это сообщение, которое сообщает, что произошло какое-то изменение или действие, например, "заказ создан" или "платёж завершён". Компоненты системы "слушают" события и реагируют на них, выполняя определённые действия.

В EDA система работает асинхронно, то есть события обрабатываются по мере их поступления, что повышает гибкость и масштабируемость системы.

Для реализации EDA обычно используют брокеры сообщений, такие как RabbitMQ или Kafka.

1. Обработка API запроса на создание заказа



7. СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (EDA)



Event-Driven Architecture

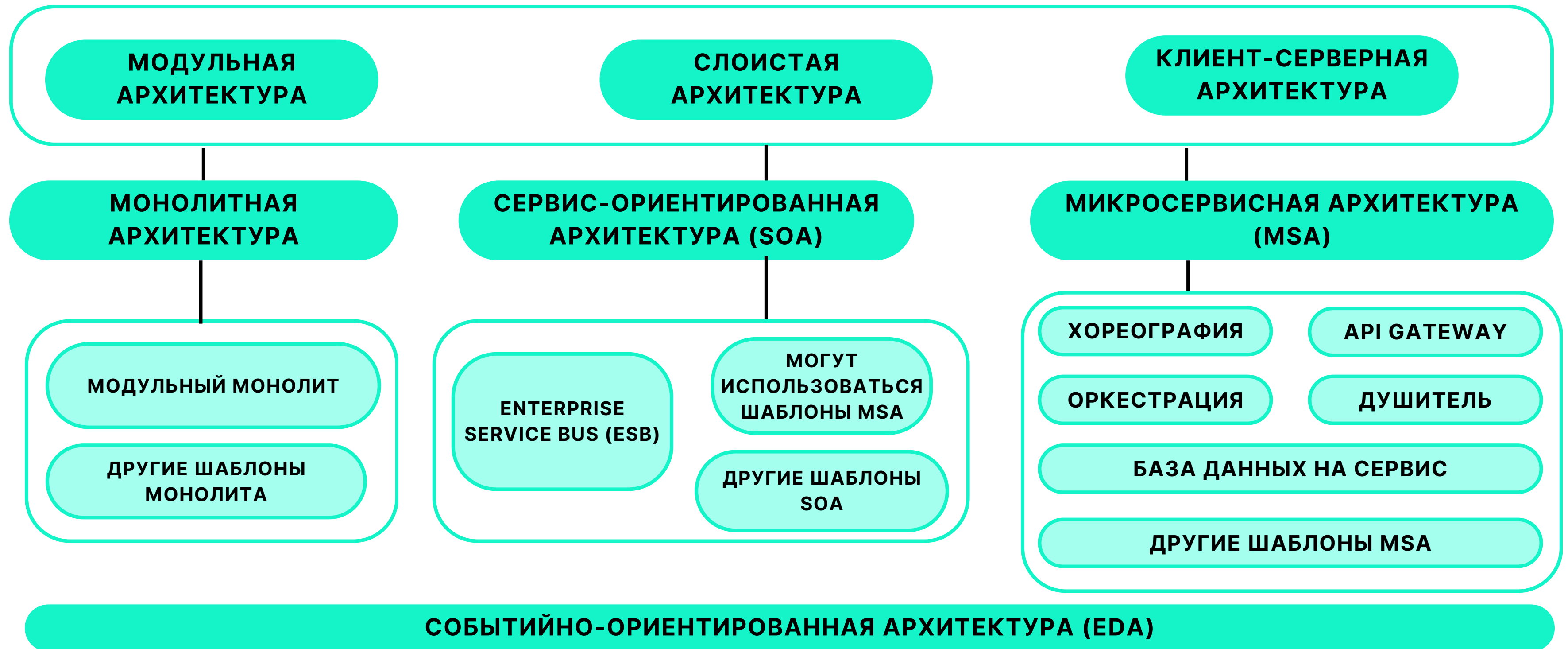
Что важно знать системному аналитику

Понимание принципов организации событийно-ориентированной архитектуры (EDA) позволяет системному аналитику работать с системами, построенными на асинхронном взаимодействии, когда события, происходящие в системе, могут обрабатываться в фоне и не блокировать основные сценарии работы системы.

Аналитик должен уметь:

- Чётко описывать события, которые генерируются системой, а также сервисы-источники событий. Это включает детали о триггерах событий (что и когда вызывает событие) и структуре данных каждого события.
- Детально проработать последовательность событий, их подписчиков и обработчиков, а также предусмотреть возможные точки сбоя.
- Учитывать, где будут храниться данные, связанные с событиями. Некоторые события требуют долговременного хранения, а другие могут быть временными и обрабатываться в реальном времени.
- Поскольку EDA-система может обрабатывать события асинхронно, важно учитывать возможные задержки и сбои в обработке событий. Аналитик должен понимать, какие сервисы могут быть недоступны в процессе реализации, и как это отразится на пользовательском опыте, а также предусмотреть механизмы повторной обработки событий и планировать, как будут информироваться пользователи в случае задержек.


КАРТА ПО ОСНОВНЫМ ШАБЛОНАМ АРХИТЕКТУРЫ



АРХИТЕКТУРА ДЛЯ АНАЛИТИКОВ

ПОЛЕЗНЫЕ ССЫЛКИ

Архитектура систем для аналитиков: ТОП-5 нотаций моделирования архитектуры

<https://getanalyst.ru/database/system-analysts-architect-notations> 

Нотация моделирования архитектуры C4 — примеры диаграмм и инструменты

<https://habr.com/ru/articles/778726>

Книга “5 сервисов и микросервисов, которые есть почти в каждом проекте”

<https://t.me/getanalysts/1849>

Разбор проекта GetTickets: распил монолита на микросервисы

<https://t.me/getanalysts/1440>

Практическая программа “Архитектура систем” для системных аналитиков

<https://getanalyst.ru/education/architecture>

