

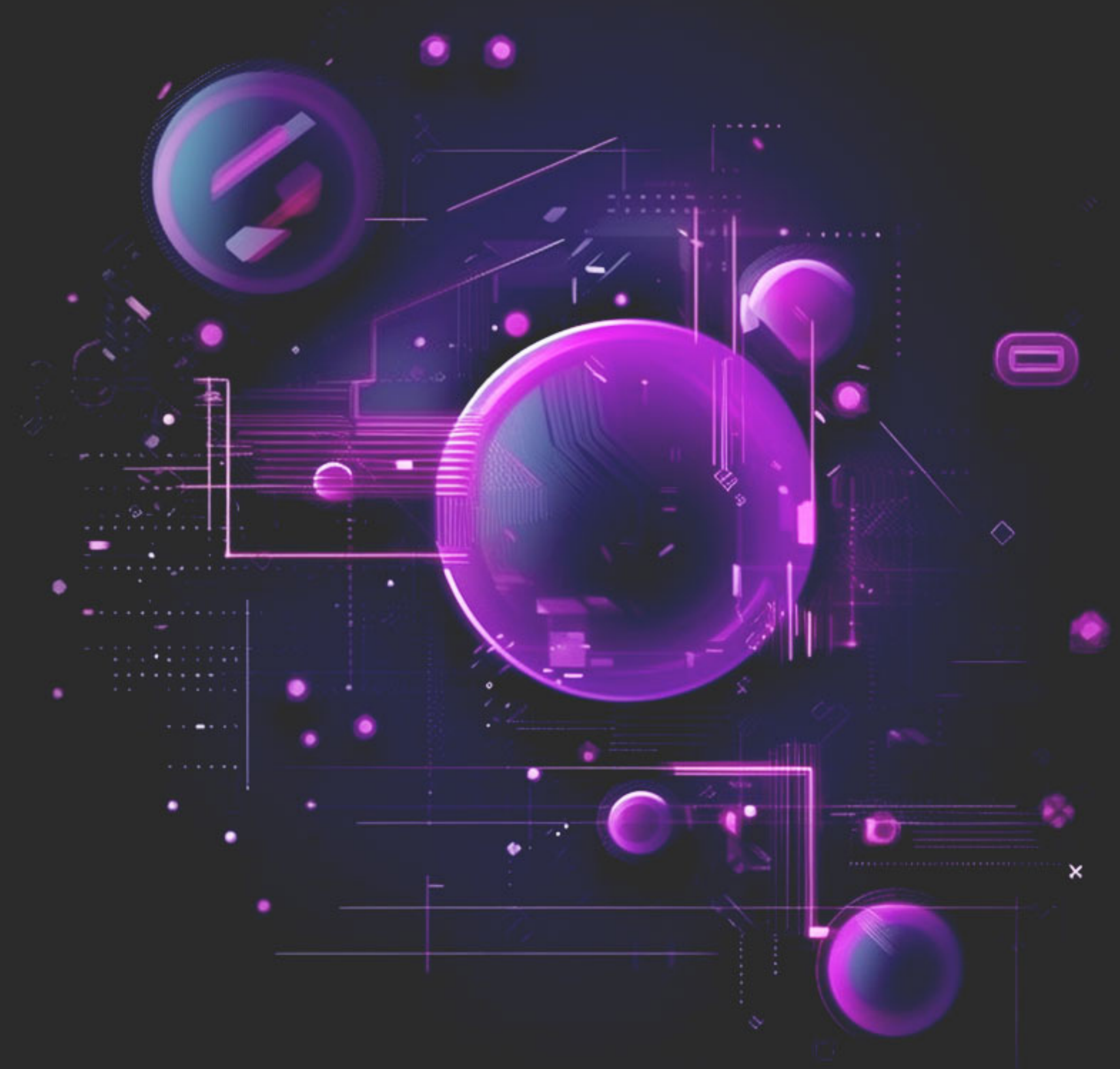


Аналитика многомодульных проектов



О чем доклад?

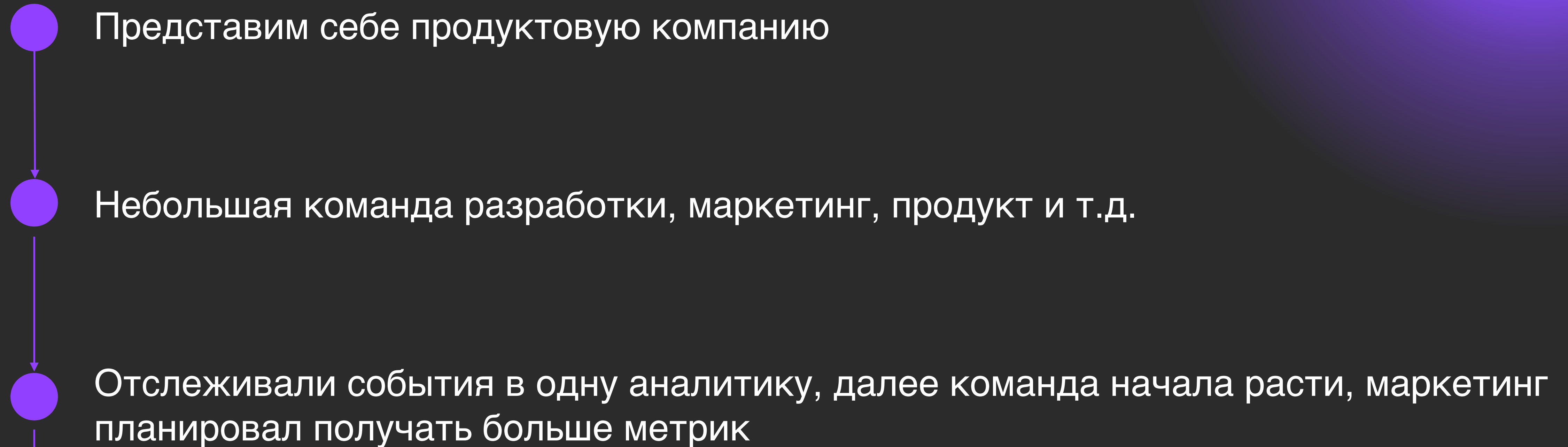
1. История. Причины. Боли.
2. Аналитика в монолит модуле или в каждой фиче?
3. Отправка событий на свой backend
4. Как можно это все завернуть в общую абстракцию



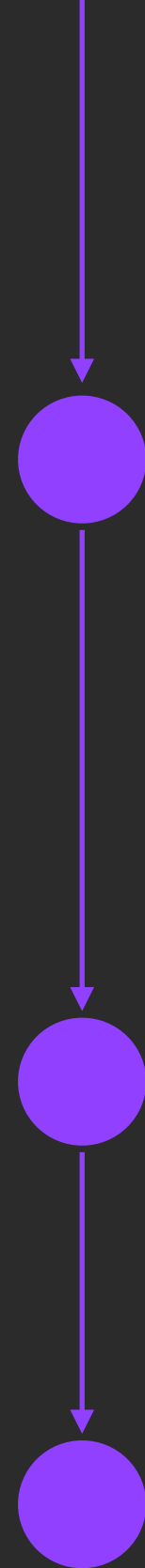
Типичные боли с аналитикой в большом проекте



Типичные боли с аналитикой в большом проекте



Типичные боли с аналитикой в большом проекте

- 
- Разработчики с каждой фичой добавляли все больше и больше разных SDK аналитик, трекеров. Завели свой бекенд для аналитики. Техдолг немного рос, фич становилось больше, команды тоже
 - В итоге через проекте оказалось кучу SDKs аналитик, трекеров, особо между собой не связанных, но примерно имеющих одну и ту же задачу
 - Добавились еще другие проектные команды, которые столкнулись с той же проблемой

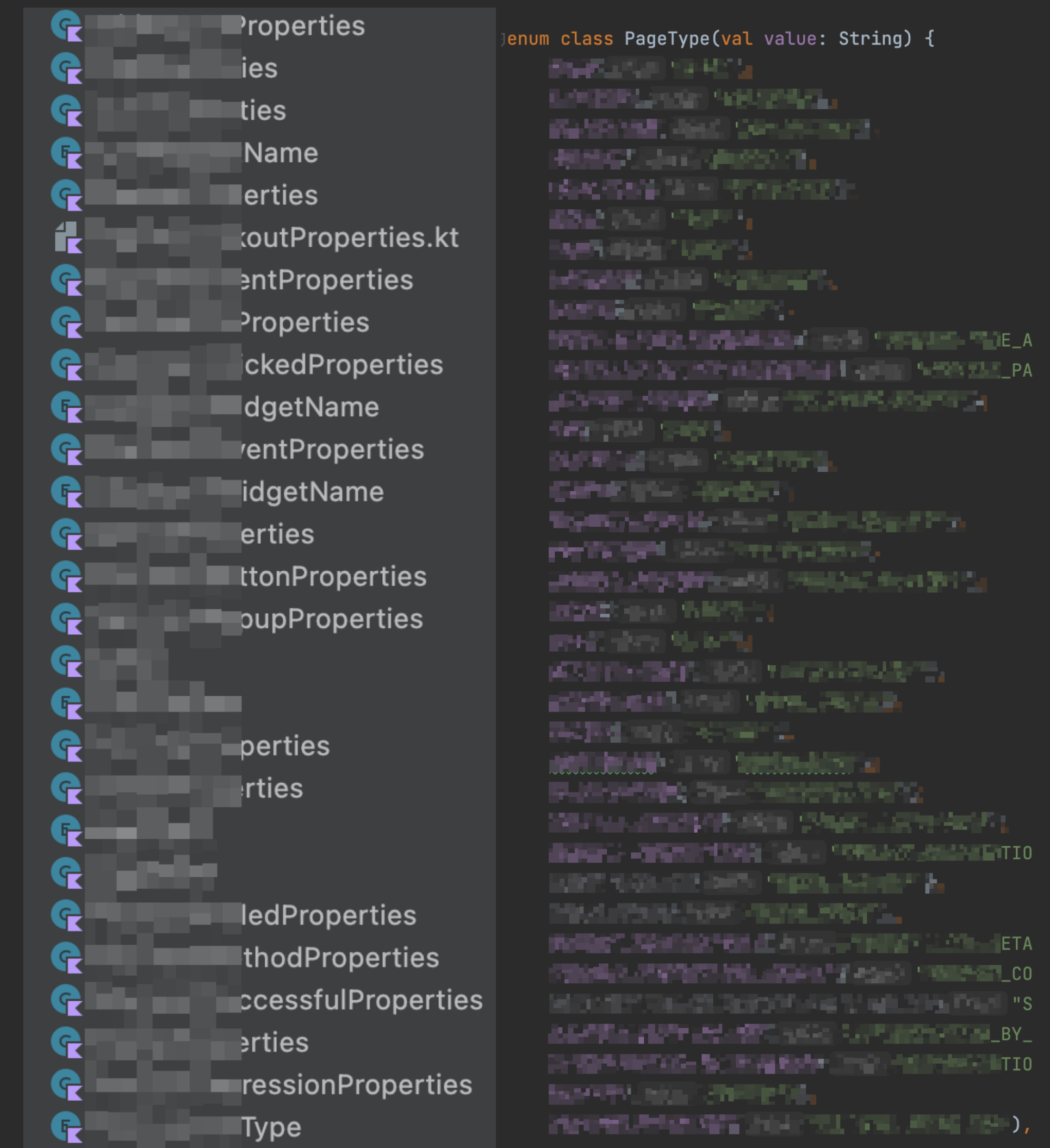
Типичные боли с аналитикой в большом проекте

- Аналитика лежит в модуле **app** (главный)
- В отдельном “монолитном” модуле **ака-analytics** куда попадают все события

Типичные боли с аналитикой в большом проекте

Как это примерно выглядит в кодовой базе:

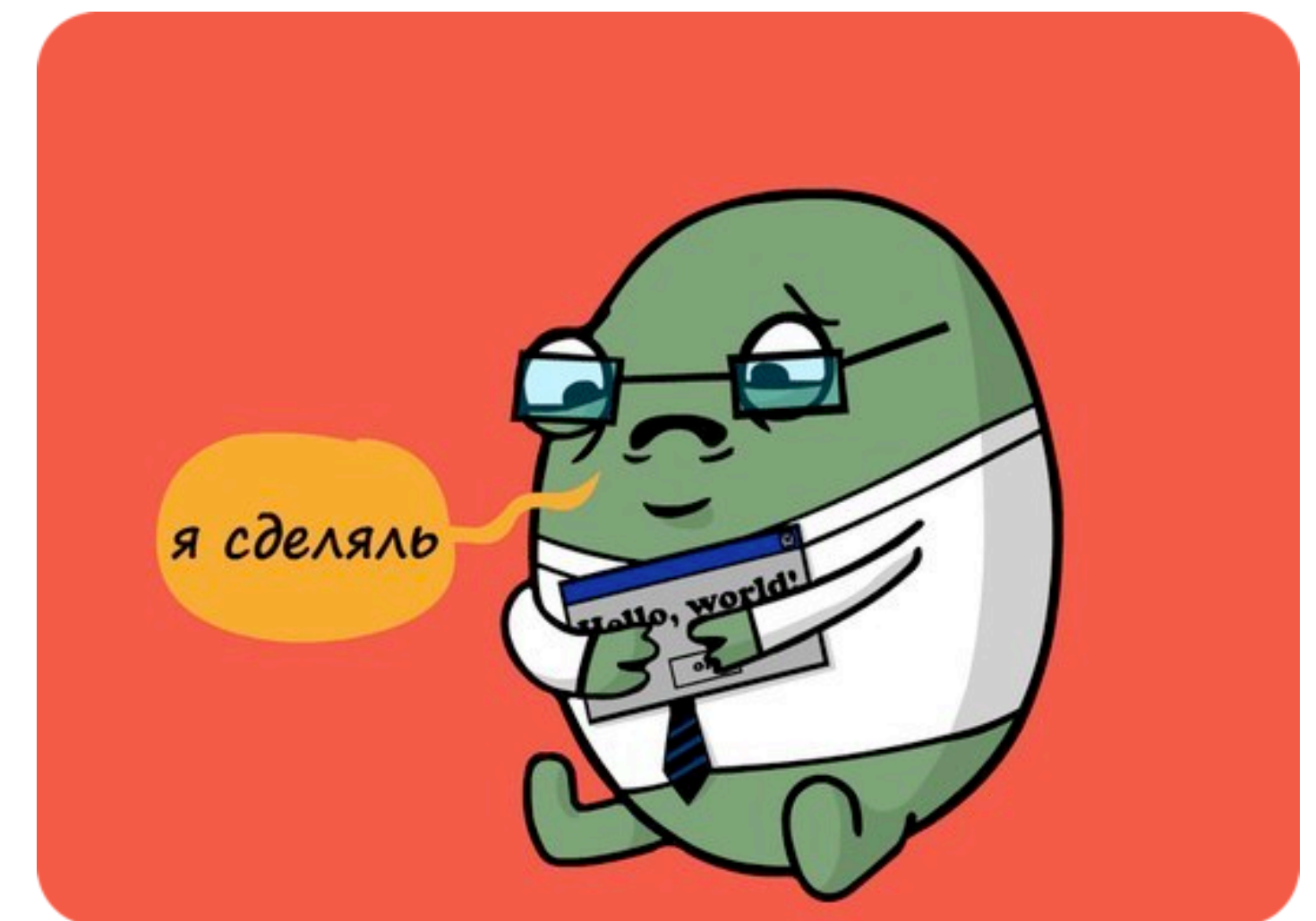
- Весь код в монолите, модуле-аналитики
- Папка: data с N+ классами константы, мапперы (даже если побиты на фичи)



Типичные боли с аналитикой в большом проекте

Плюсы

- легко завести
- можно не думать что к какой фиче, экрану относится
- создается папка раскаге и готово
- тестами покрываем там же (если сможем)



Типичные боли с аналитикой в большом проекте

Минусы

- тяжело поддерживать (разобраться что к какой фиче относится, даже если бить по папкам)
- модуль растёт, файлов больше => стухает
- замедляет скорость сборки => каждое изменение пересобирает почти все модули
- появляется соблазн положить куда-то не туда (“теория разбитых окон”)

Типичные боли с аналитикой в большом проекте

Минусы

- при небольшом изменении приходится прогонять все тесты локально в этом
- постоянные конфликты
- сторонние SDK зачастую синглтоны используются напрямую, дольше покрывать тестами, повторяющаяся работа в разных командах
- мапперы, утилы, хелперы лежат где-то рядом

Типичные боли с аналитикой в большом проекте

Куда подойдет?

В небольшие проекты, MVP, маленькая команда, несколько экранов

Где будет больно?

Многомодульный проект, команда > 5 , много фич, много разных аналитик, метрик, эксперименты

Что с этим можно сделать?

- Организация кода
- Свой сервер для обработки аналитики
- Абстракция для работы с множеством аналитик
- Kotlin DSL билдеры для аналитик
- Качество и тестирование



Организация кода



Организация кода

Плюсы

- Просто (требует минимум дисциплины, максимум CI проверки)
- Аналитика лежит в той фиче где она используется
- Можно локально создавать классы для аналитики и мапперы с ограниченным скоупом

Организация кода

Плюсы

- Проще поддерживать
- Почти не возникают конфликтов
- При необходимости можно скрыть реализацию стороннего SDK в конкретном модуле

Организация кода

Минусы

- Сложнее первого варианта
- Приходится думать, нужна дисциплина
- Чуть больше бойлерплейта

Организация кода

Минусы

- Для каждого экрана отдельный файл аналитики
- Некоторые названия событий могут дублироваться
- Стороннее SDK в случае переиспользования в нескольких модулях придется выносить

Организация кода. По экрану.

```
class MainPageAnalytics(  
    private val analytics: Analytics,  
) {  
  
    fun bannerClicked(bannerId: Long, categoryId: Long?, bannerUrl: String?) {  
        sendEvent {  
            name(EventType.BannerClicked.value)  
            bannerProperties(bannerId, categoryId, bannerUrl)  
        }  
    }  
  
    fun promoImageItemShown(promoImageId: Int, position: Int) {  
        sendViewEvent {  
            name(EventImpression.PromoImageImpression.value)  
            property("page_type", PageType.Main.value)  
            property("promo_image_id", promoImageId)  
            property("position", position)  
        }  
    }  
}
```

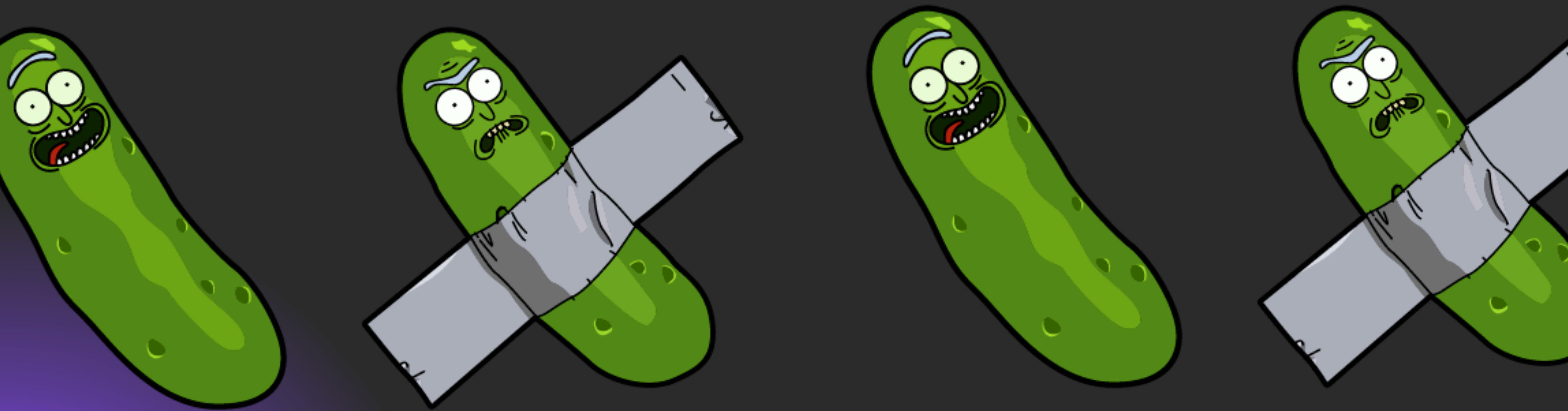
Организация кода. По экрану. MainPageAnalytics

```
private enum class EventType(val value: String) {  
    BannerClicked("BANNER_CLICKED"),  
    PromoImageClicked("PROMO_IMAGE_CLICKED"),  
}  
  
private enum class EventImpression(val value: String) {  
    BannerImpression("BANNER_IMPRESSION"),  
    PromoImageImpression("PROMO_IMAGE_IMPRESSION"),  
}  
  
private enum class PageType(val value: String) {  
    Main("MAIN"),  
}  
  
private enum class WidgetName(val value: String) {  
    PopularCategory("POPULAR_CATEGORY")  
}  
}
```

Организация кода. По юзер-стори.

```
class AddToCart(private val analytics: Analytics) {  
    ...  
    fun click(cart: Cart) {  
        analytics.send(cart.toAnalytics())  
    }  
    ...  
}  
  
class Payment(private val analytics: Analytics) {  
    ...  
    fun sendStartPayment(payment: Payment) {  
        analytics.send(PaymentMapper(payment))  
    }  
    ...  
}
```

Отправка событий на свой сервер



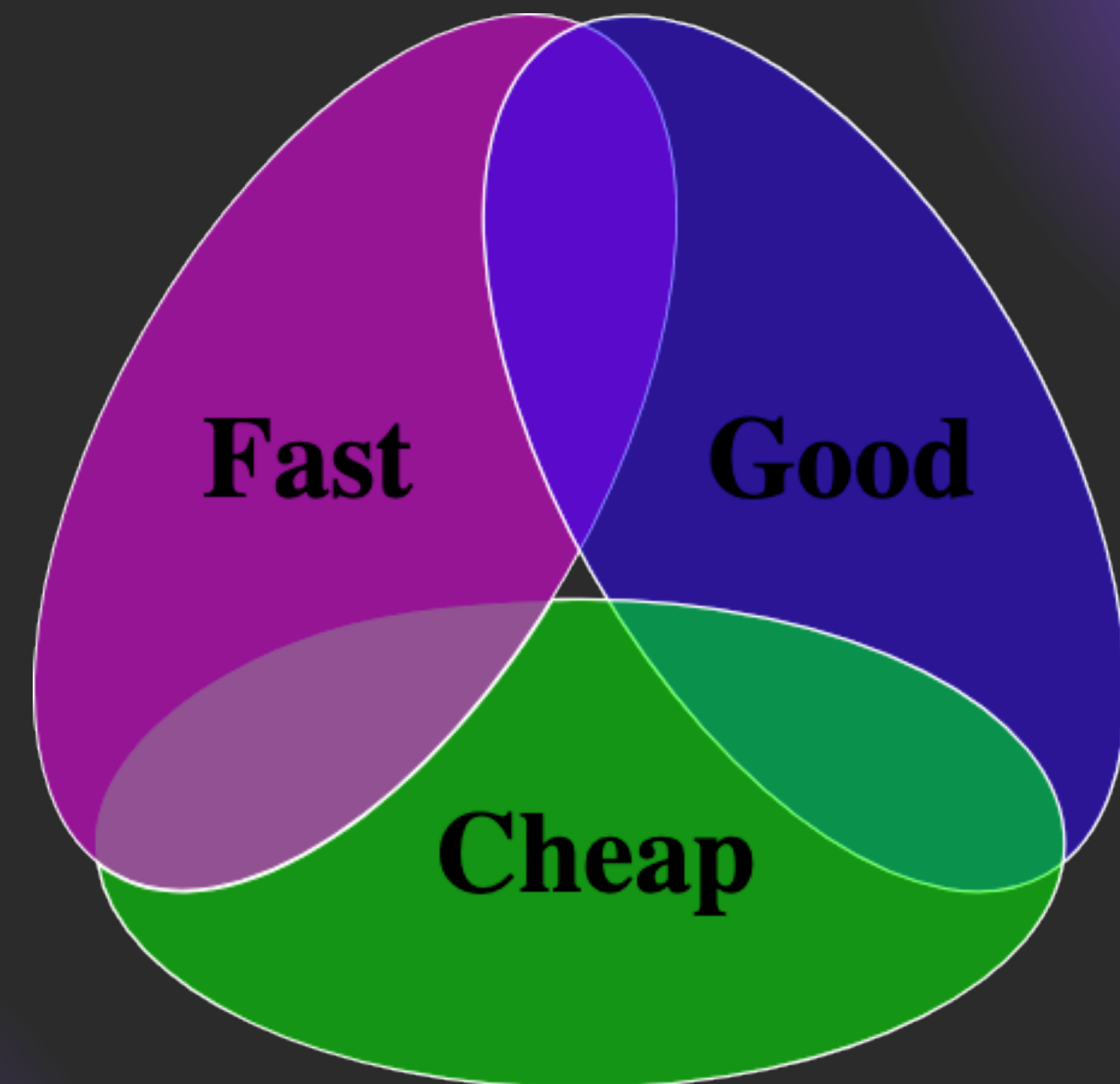
Отправка событий на свой сервер

- ! Нужны ресурсы аналитики и бекенда для создания и поддержки
- ! Не решает все проблемы, часть, иногда приходится все равно добавлять события от сторонних SDK к себе на клиент
- ! Не всегда есть ресурсы или смысл делать сразу на бекенде интеграцию с сторонними SDK

Отправка событий на свой сервер. Варианты.

В зависимости от ресурсов разработки на клиентах рассмотрим варианты:

1. Быстро и легко
2. Быстро и чуть сложнее
3. Медленно и сложнее

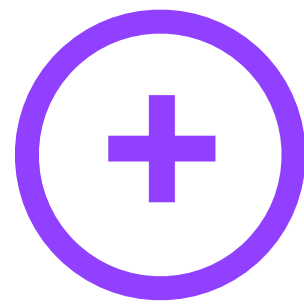


Отправка событий на свой сервер. Быстро и легко.

- Один общий модуль аналитики
- Все события/классы в одном месте
- User related data: installId, app_version, firebase_id
- Device related data: id, os, tokens
- Храним user related data в памяти/sharedPrefs
- Делаем interactor отправки в аналитику в рантайме

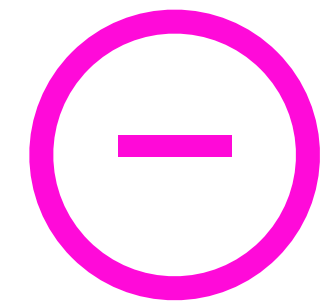
Отправка событий на свой сервер. Быстро и легко.

Плюсы



- Храним user related data в памяти/sharedPrefs
- Делаем один usecase (под капотом отправляет запрос в аналитику)
- Делаем interactor отправки в аналитику в рантайме

Минусы



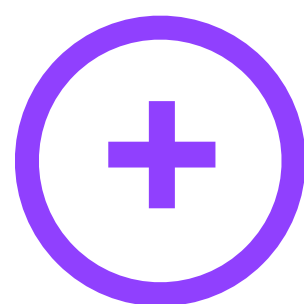
- События могут не дойти: ошибки, не хранятся на диске
- Неконсистентность данных при логине/разлогине/смена токенов
- Обработать ошибки каждый раз на каждом экране
- Все события и классы в одном месте

Отправка событий на свой сервер. Быстро и сложнее.

- Один общий модуль аналитики
- Все ивенты/классы в одном месте
- Делаем один общий интерфейс для отправки событий
- UserProperties записываются как есть
- Делаем interactor сохранения событий в базу данных
- Храним user related data в базе данных
- По очереди отправляем события из база данных на сервер

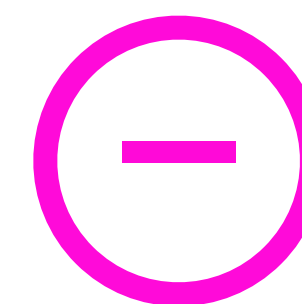
Отправка событий на свой сервер. Быстро и сложнее.

Плюсы



- Данные сохраняются на диске
- Храним user related data в памяти/
sharedPrefs
- Можно использовать worker/сервис/
поток

Минусы



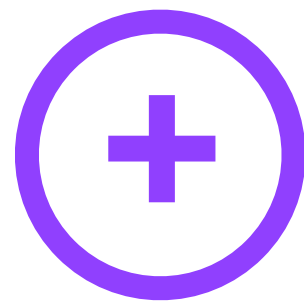
- Обработка ошибок, backoff, retry-и
- Все события и классы в одном месте
- Неконсистентность из-за
возможности смены user-related
данных

Отправка событий на свой сервер. Медленнее и сложнее.

- Отделяем user-related сущности в properties в БД
- Батчим на основе хешей по properties
- Для андроида добавляем - WorkManager
- Чанкуем данные для отправки по хешам properties
- Фича-флагами управляем размером чанка и периодичность

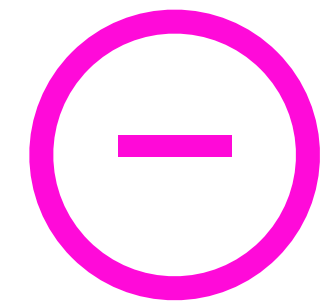
Отправка событий на свой сервер. Медленнее и сложнее.

Плюсы



- Консистентность
- Храним все в базе данных
- Можно конфигурировать размеры чанков и частоту отправки
- Каждое событие уникальное

Минусы



- Нужно больше времени на добавление каких-то пропертей
- Миграции базы данных

Отправка событий на свой сервер.

- Вспомним начало проблем: иногда приходится все равно добавлять события от сторонних SDK к себе на клиент :(
- Вынесли максимально на свой бекенд, но проблема для клиентов частично остается
- Хотелось бы иметь какой-нибудь общий интерфейс для работы с аналитиками
- В будущем нет гарантии, что контракт бекенда не изменится для клиентов

Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Пример.

```
AnalyticsSdk.getInstance().send {  
    newAnalytics {  
        event {  
            type("mobius")  
            parameter("year", "2023")  
        }  
        action("speech")  
        build() // force to check  
    }  
    firebase {  
        name("mobius")  
        property("year", "2023")  
    }  
    somethingElse {  
        customObject("mobius", Data2023)  
    }  
}
```

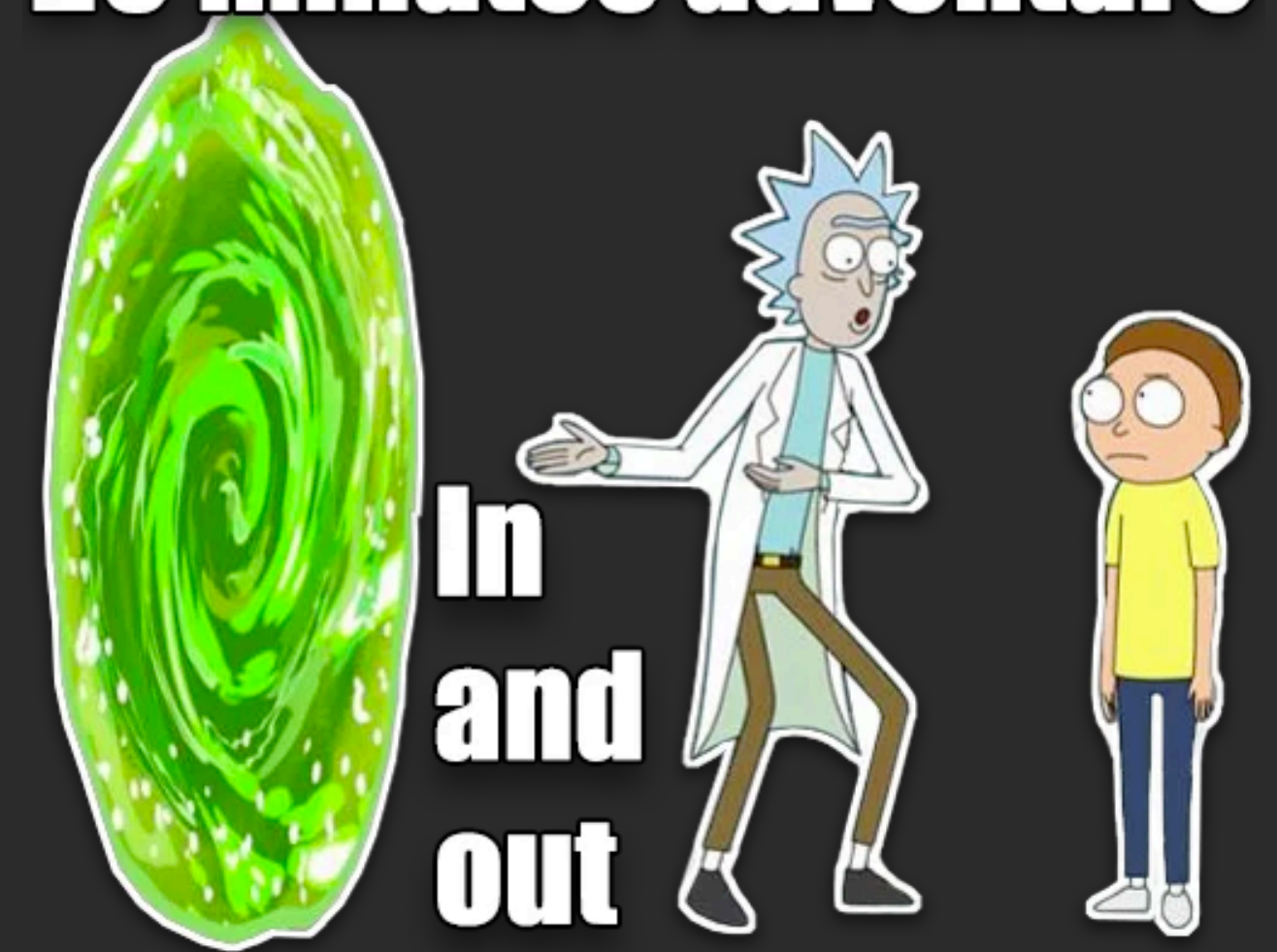
Абстракция для работы с N аналитиками. Common.

- один интерфейс в фича-модулях (ложится на многомодульность)
- подключать новые реализации аналитик отдельно (firebase, backend, etc...)
- конфигурировать события перед отправкой
- отправлять события в разные аналитики

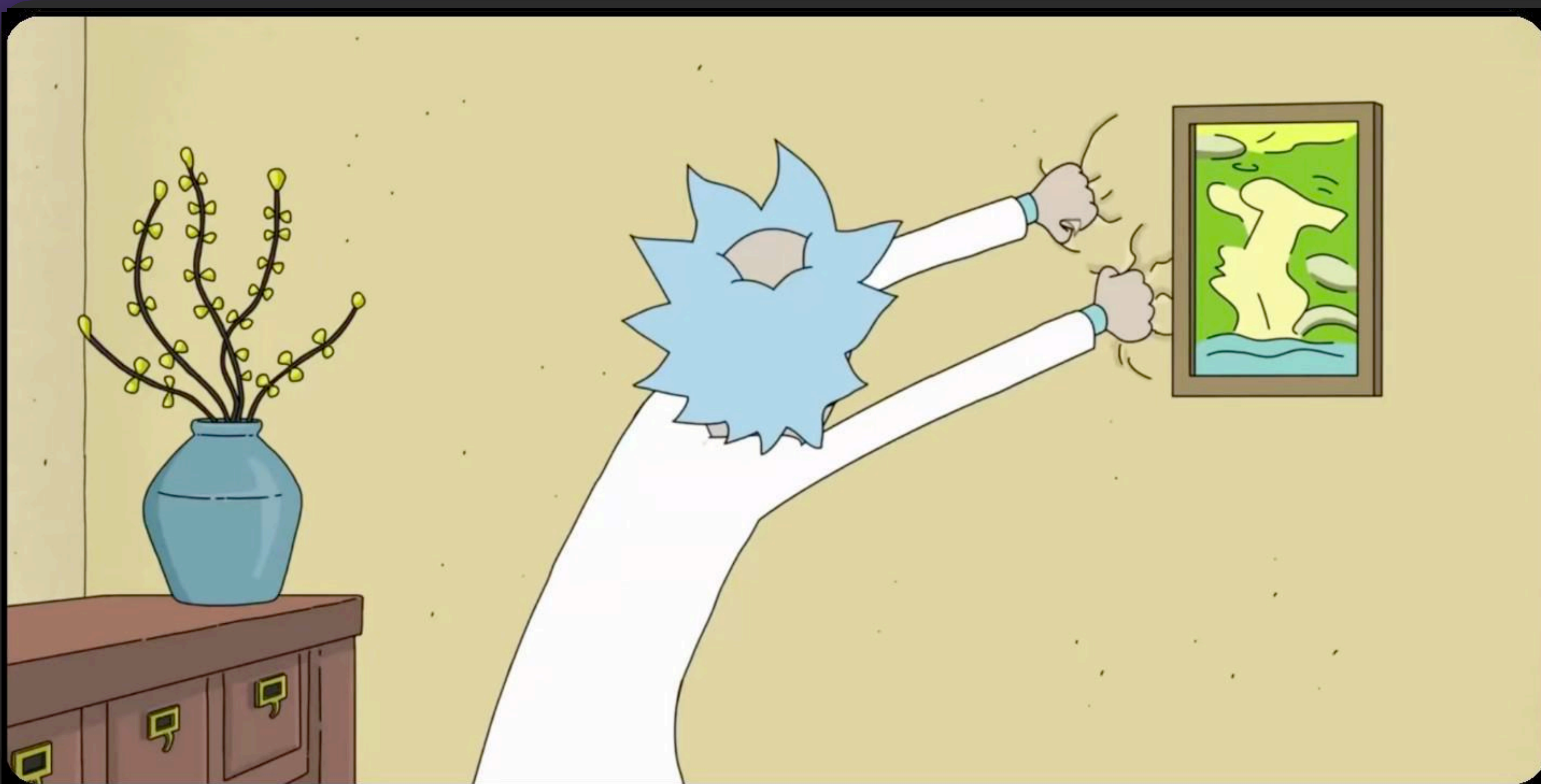
Абстракция для работы с N аналитиками. Common.

- покрывать тестами и тестировать сами события и данные
- нет андроид зависимостей в common модуле
- билдеры Kotlin DSL
- отсутствие миграций в БД, возможность расширения properties клиентами
- минимум библиотечных зависимостей

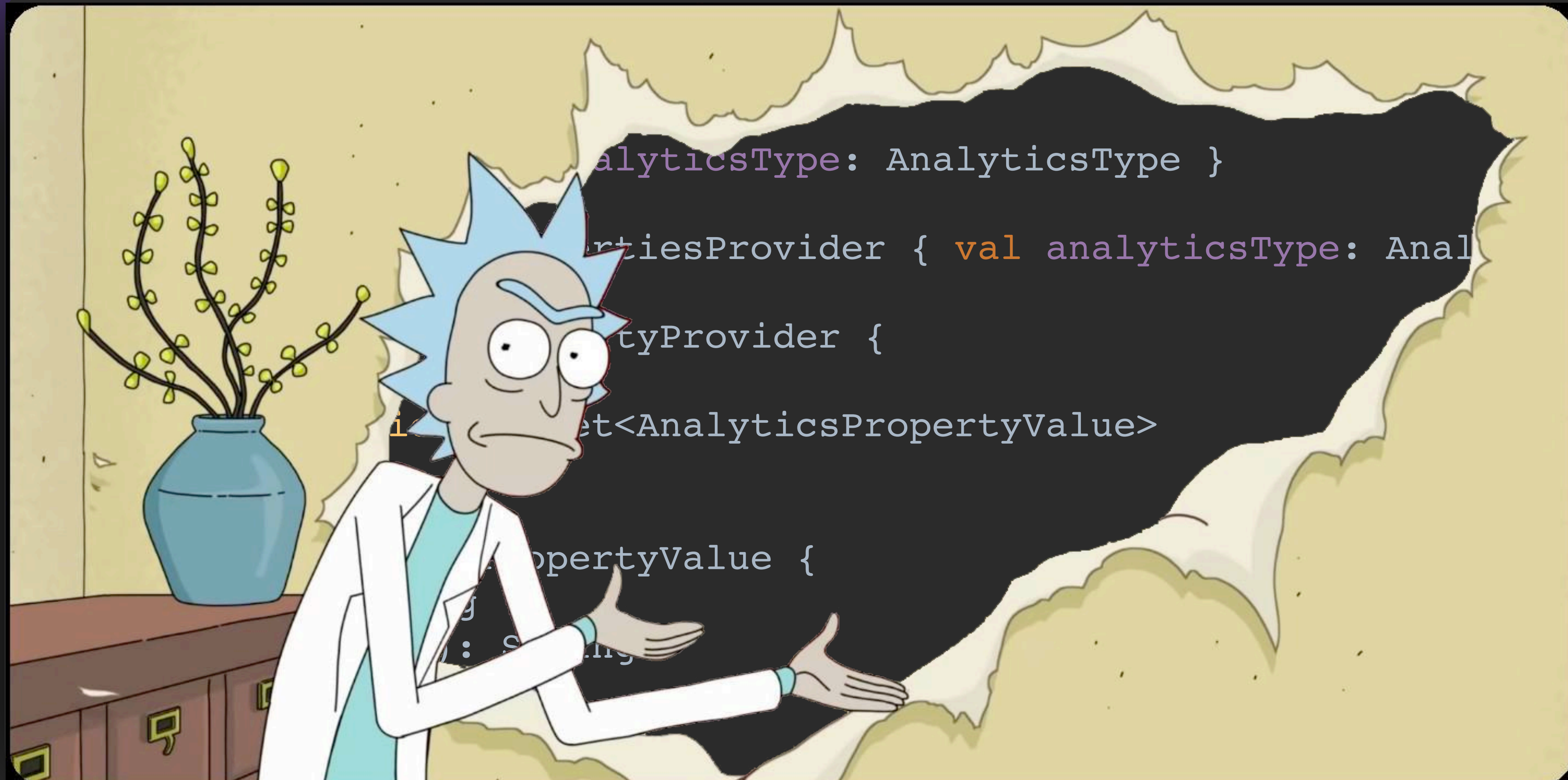
20 minutes adventure



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
const val Firebase: AnalyticsType = "Firebase"

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

class FirebaseEvent(val name: String, val action: String): Event

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

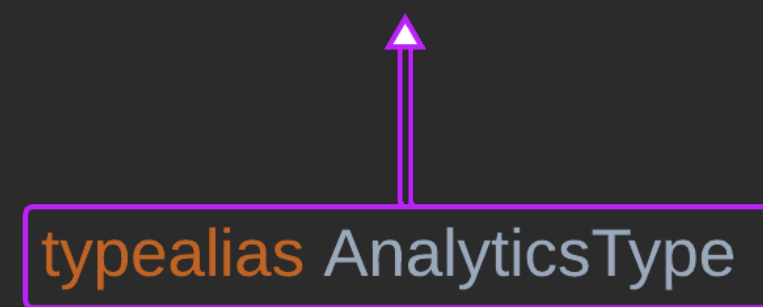
interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

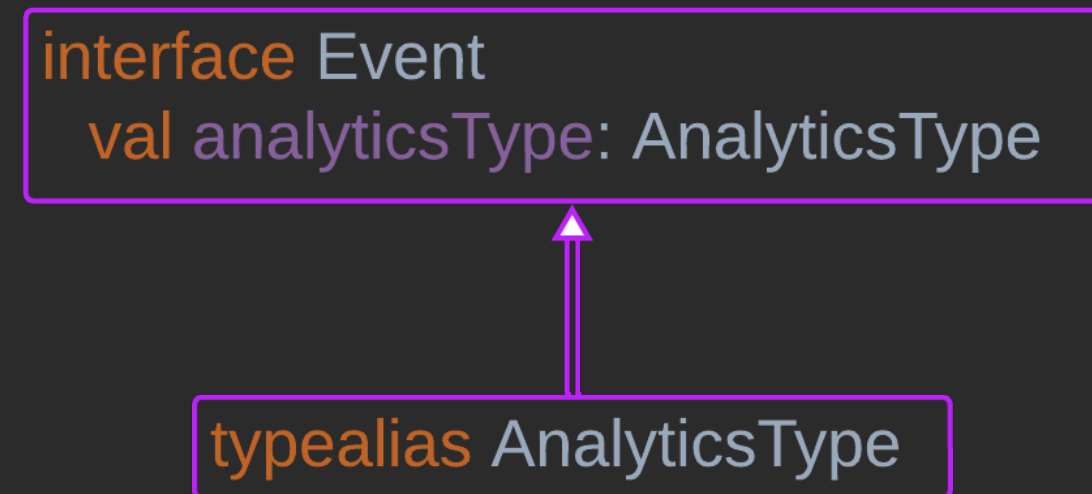
Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType
```

Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

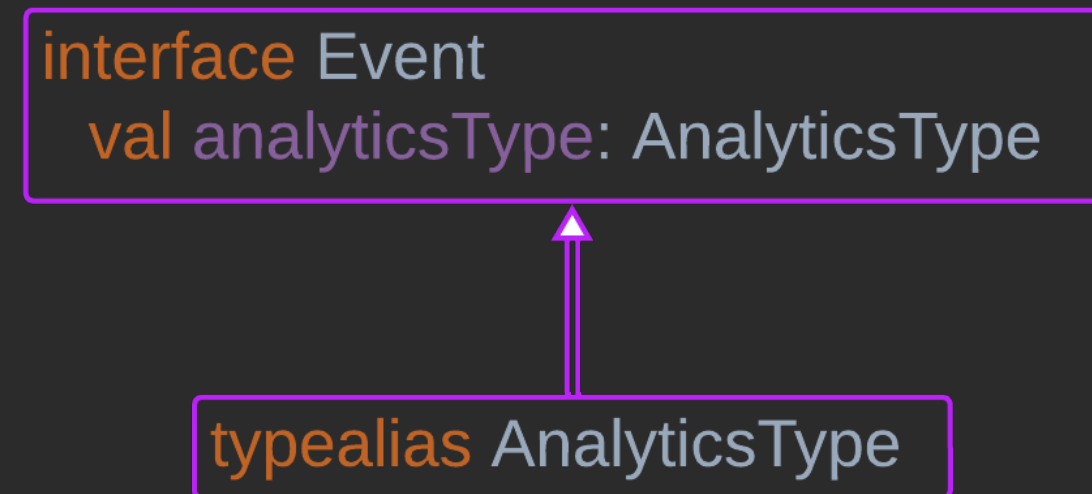
interface Event { val analyticsType: AnalyticsType }

class FirebasePropertiesProvider(
    val device: AnalyticsPropertyProvider,
) : AnalyticsPropertyProvider(Firebase)

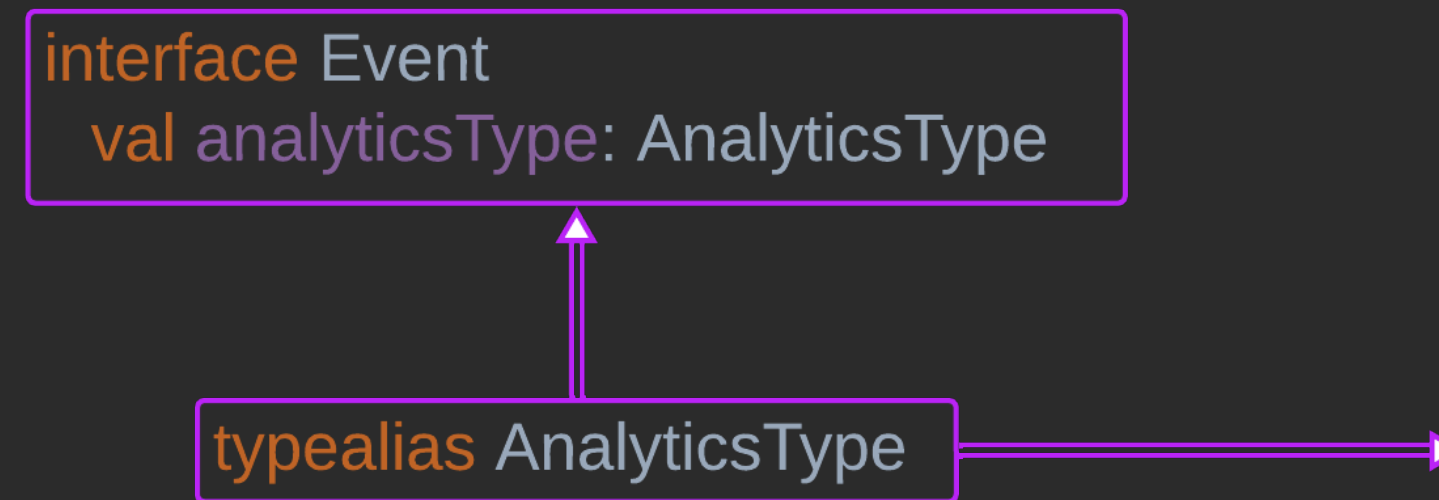
interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

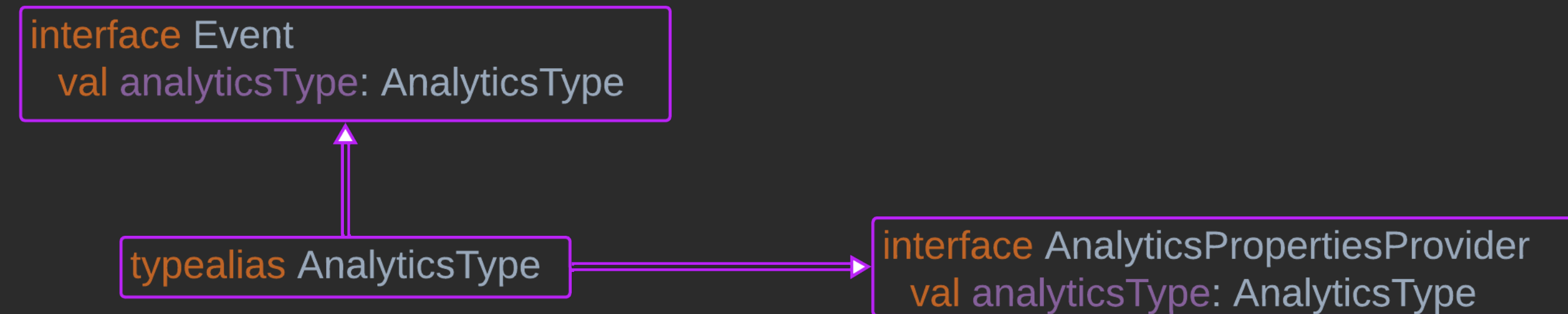
Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

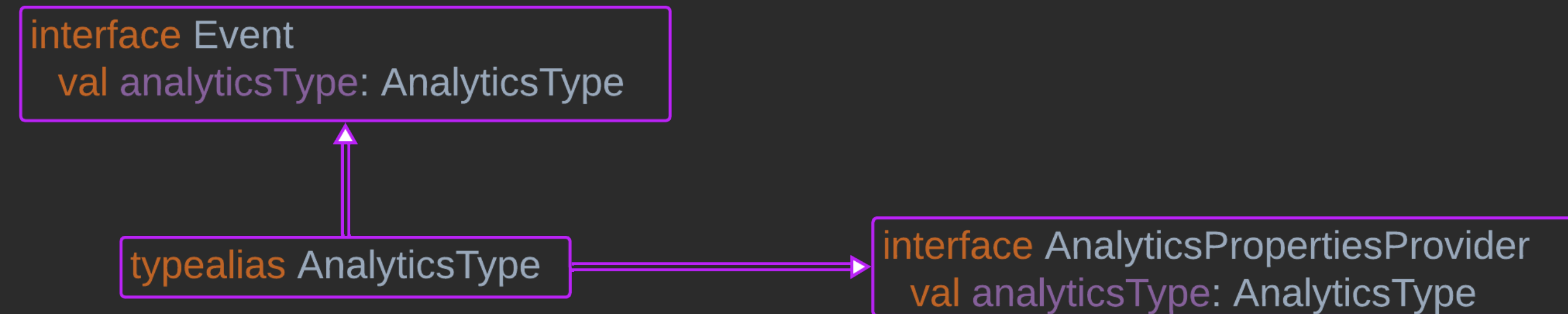
interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

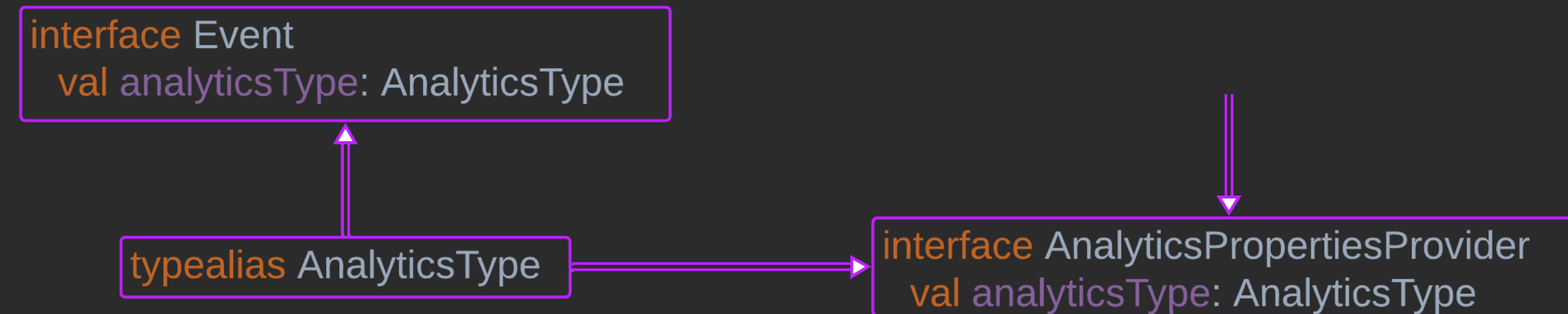
class DeviceAnalyticsPropertyProvider(
    private val values: Set<AnalyticsPropertyValue>,
) : AnalyticsPropertiesProvider {
    override val key: String = "device_properties"
    override fun properties(): Set<AnalyticsPropertyValue> = values
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

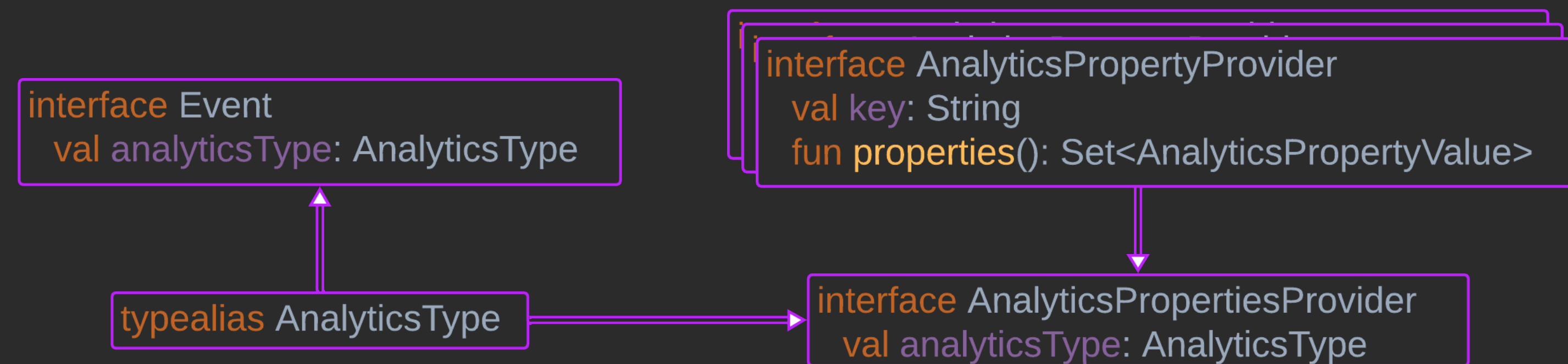
Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

interface AnalyticsPropertyValue {
    val key: String
    fun getValue(): String?
}
```

Абстракция для работы с N аналитиками. Common.

```
typealias AnalyticsType = String

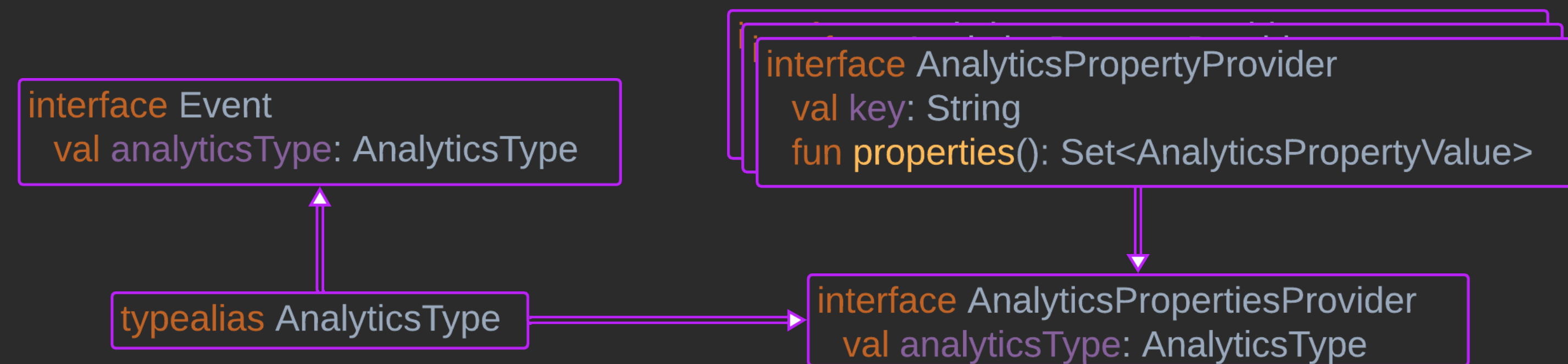
interface Event { val analyticsType: AnalyticsType }

interface AnalyticsPropertiesProvider { val analyticsType: AnalyticsType }

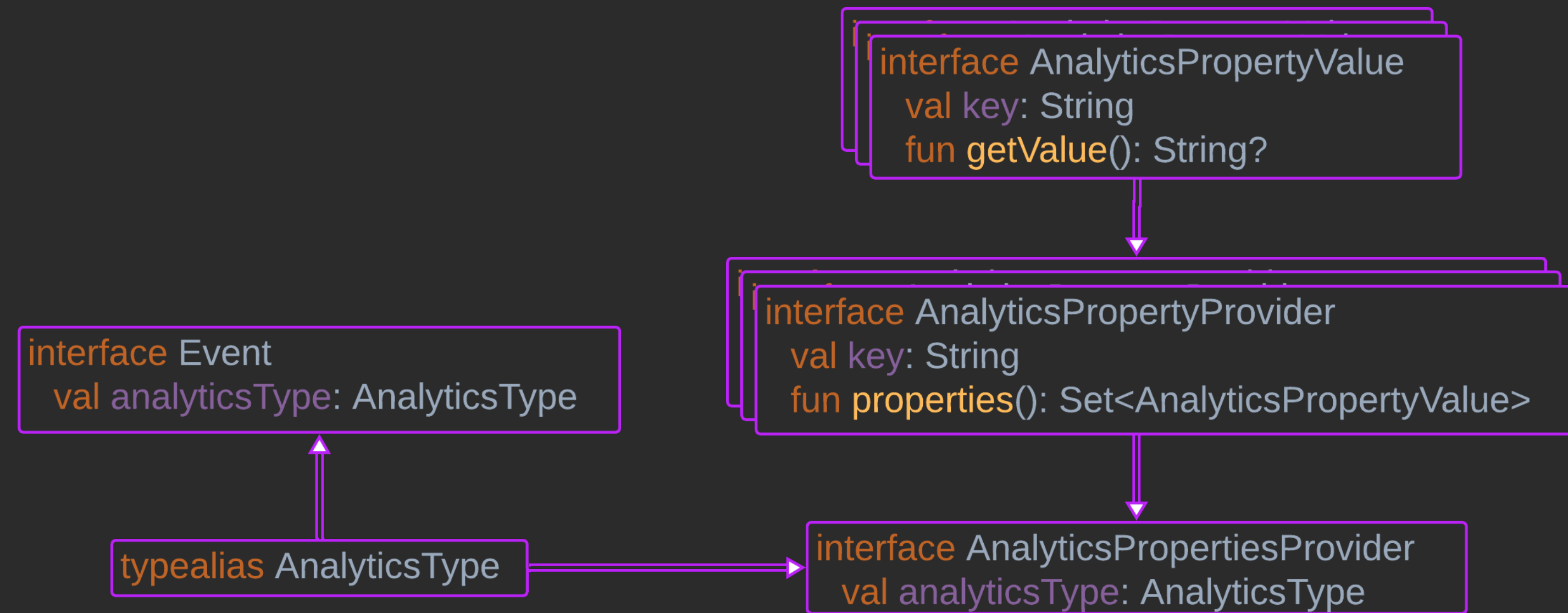
interface AnalyticsPropertyProvider {
    val key: String
    fun properties(): Set<AnalyticsPropertyValue>
}

class AccountIdProperty : AnalyticsPropertyValue {
    override val key: String = "account_id"
    override fun getValue(): String = "mobius2023"
}
```

Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

class Analytics(
    val type: AnalyticsType,
    val propertyProvider: AnalyticsPropertiesProvider,
    val eventSender: AnalyticsEventSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>)
}

@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

class Analytics(
    val type: AnalyticsType,
    val propertyProvider: AnalyticsPropertiesProvider,
    val eventSender: AnalyticsEventSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>())
}

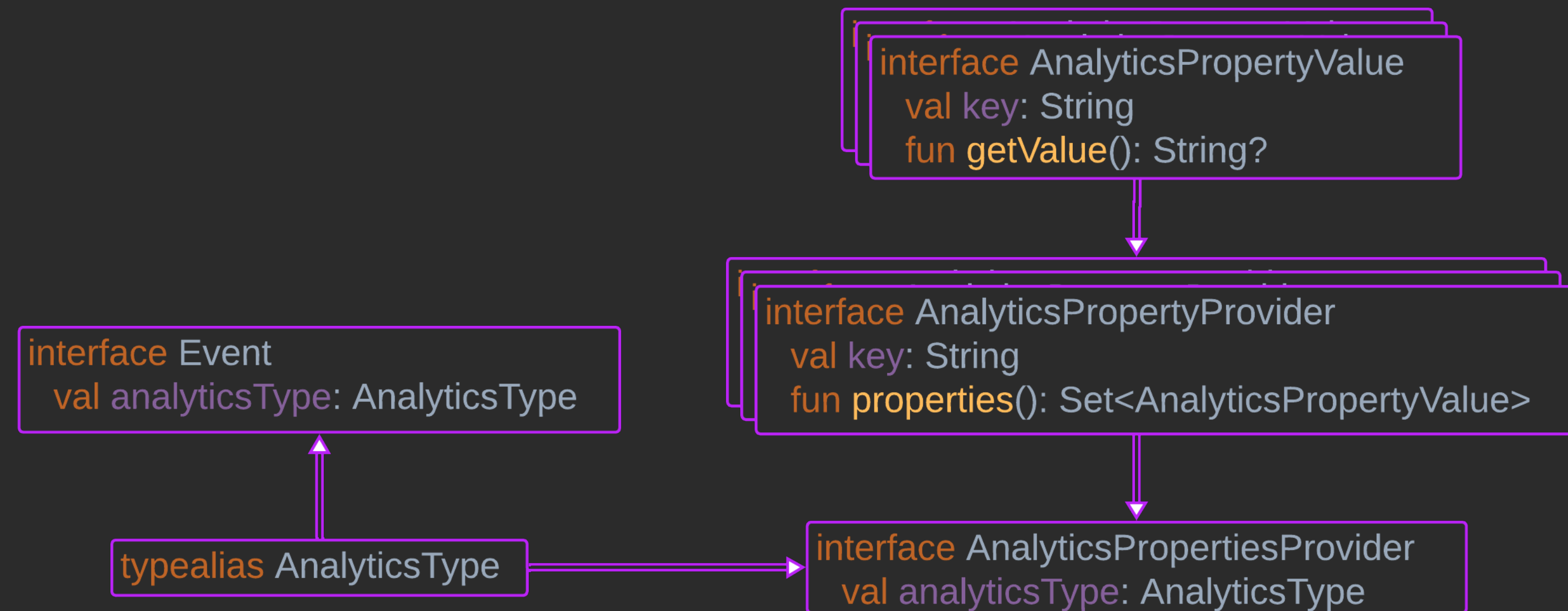
@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

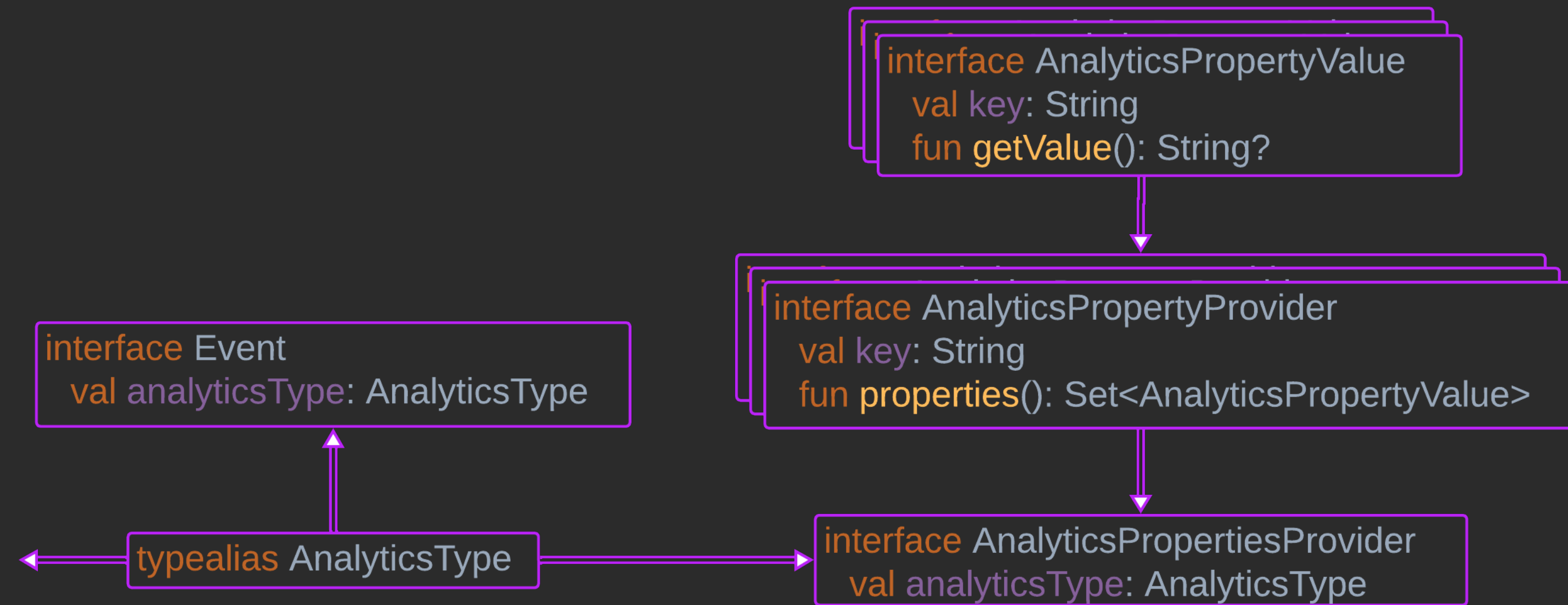
Абстракция для работы с N аналитиками. Common.

```
class FirebaseAnalyticsEventSender(  
    private val firebase: FirebaseSDK,  
    private val firebasePropertiesProvider: FirebasePropertiesProvider,  
) : AnalyticsEventSender {  
    override val type: AnalyticsType = Firebase  
  
    override suspend fun send(event: FirebaseEvent) {  
        firebase.send(event.name, event.action)  
    }  
}
```

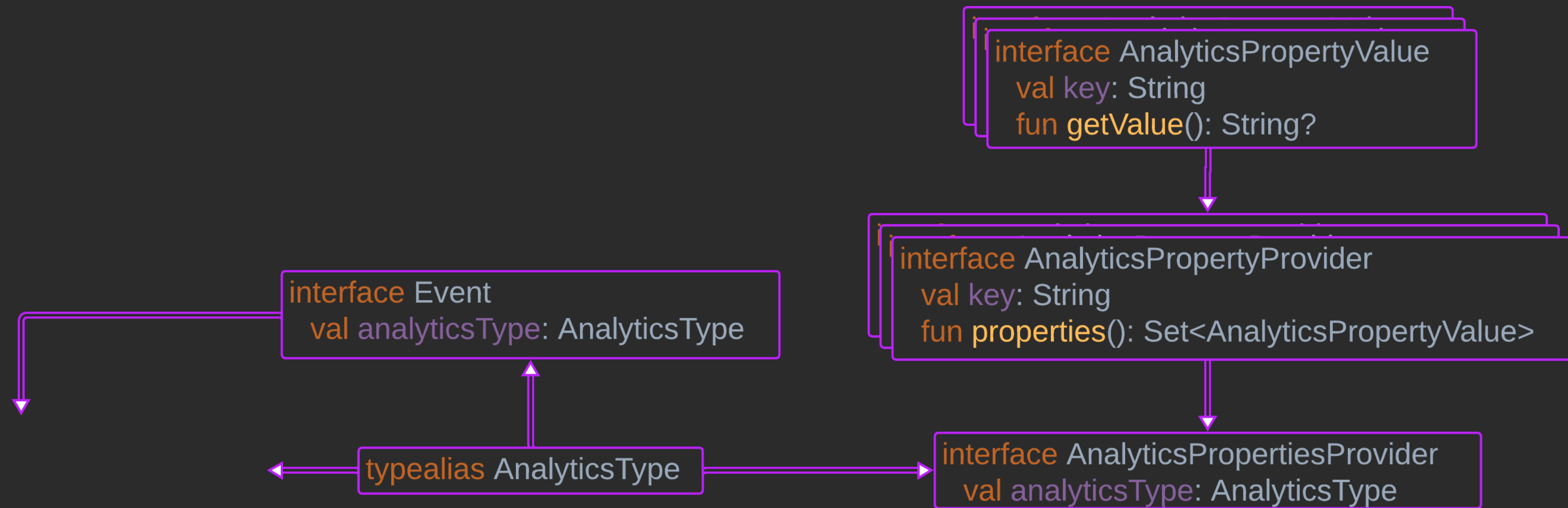
Абстракция для работы с N аналитиками. Common.



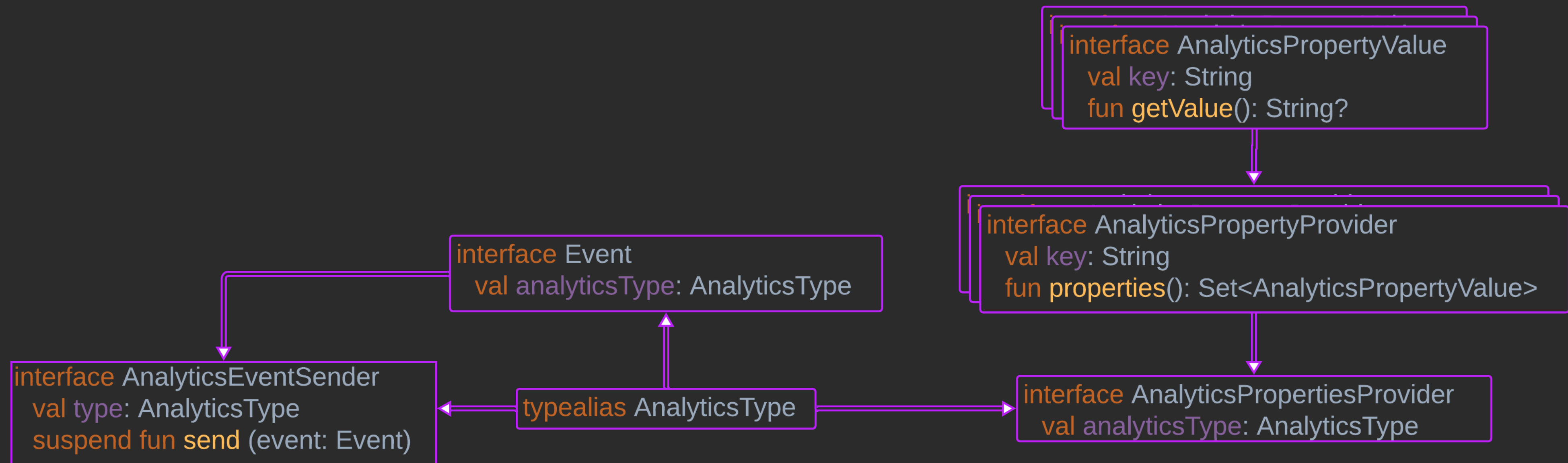
Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

class Analytics(
    val type: AnalyticsType,
    val propertyProvider: AnalyticsPropertiesProvider,
    val eventSender: AnalyticsEventSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>)
}

@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

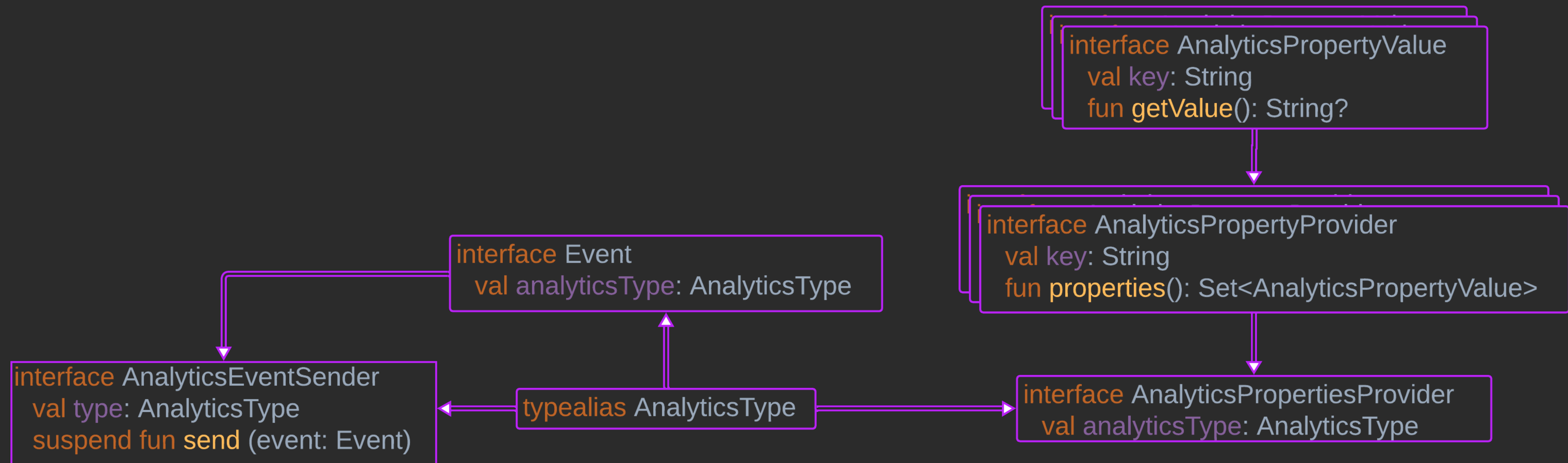
class FirebaseAnalytics(
    val type: AnalyticsType = Firebase,
    val propertyProvider: FirebasePropertiesProvider,
    val eventSender: FirebaseAnalyticsSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>)
}

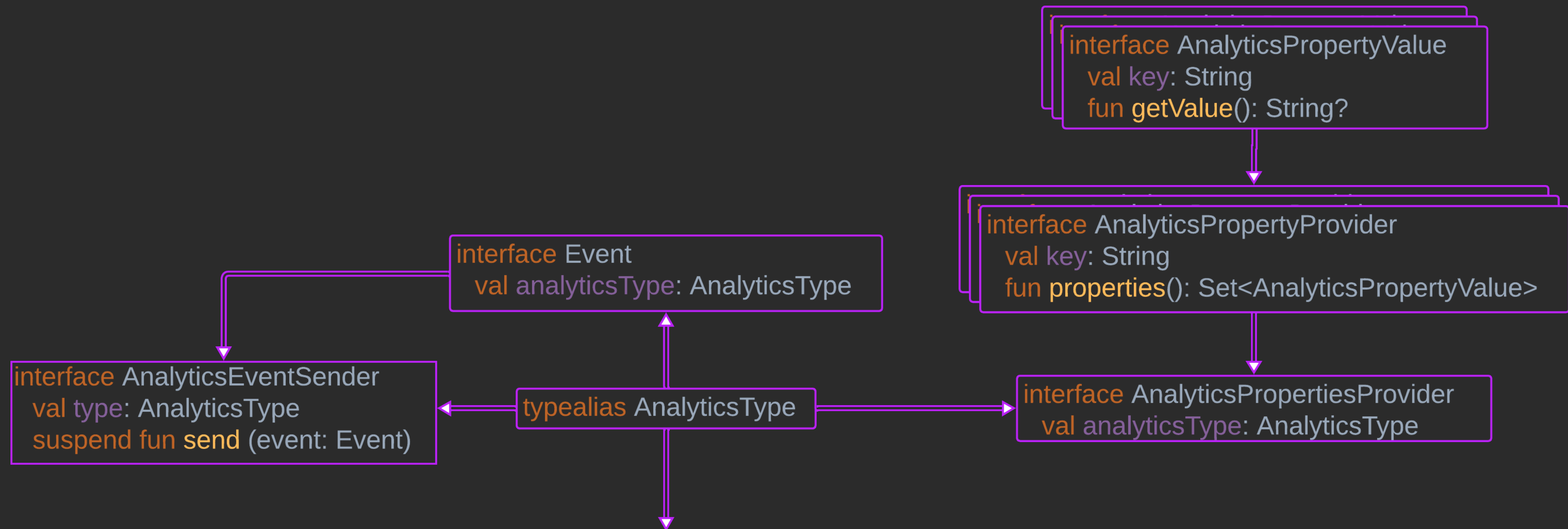
@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

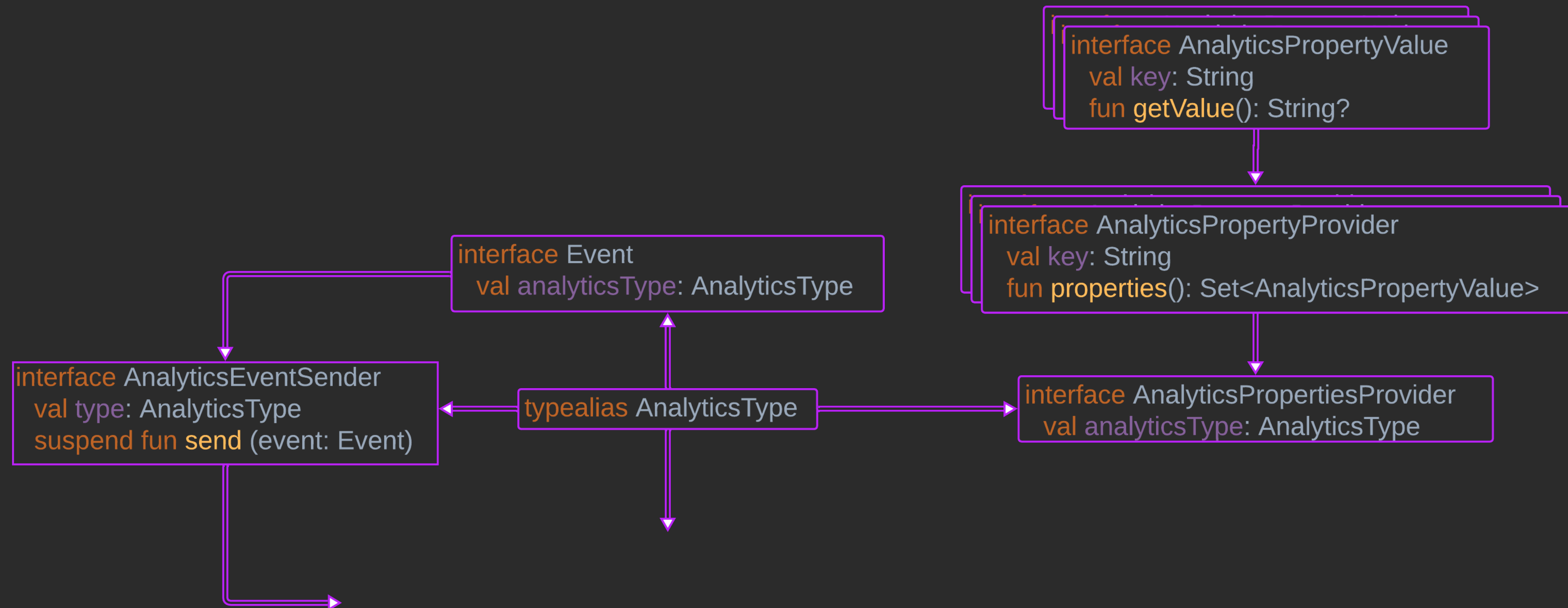
Абстракция для работы с N аналитиками. Common.



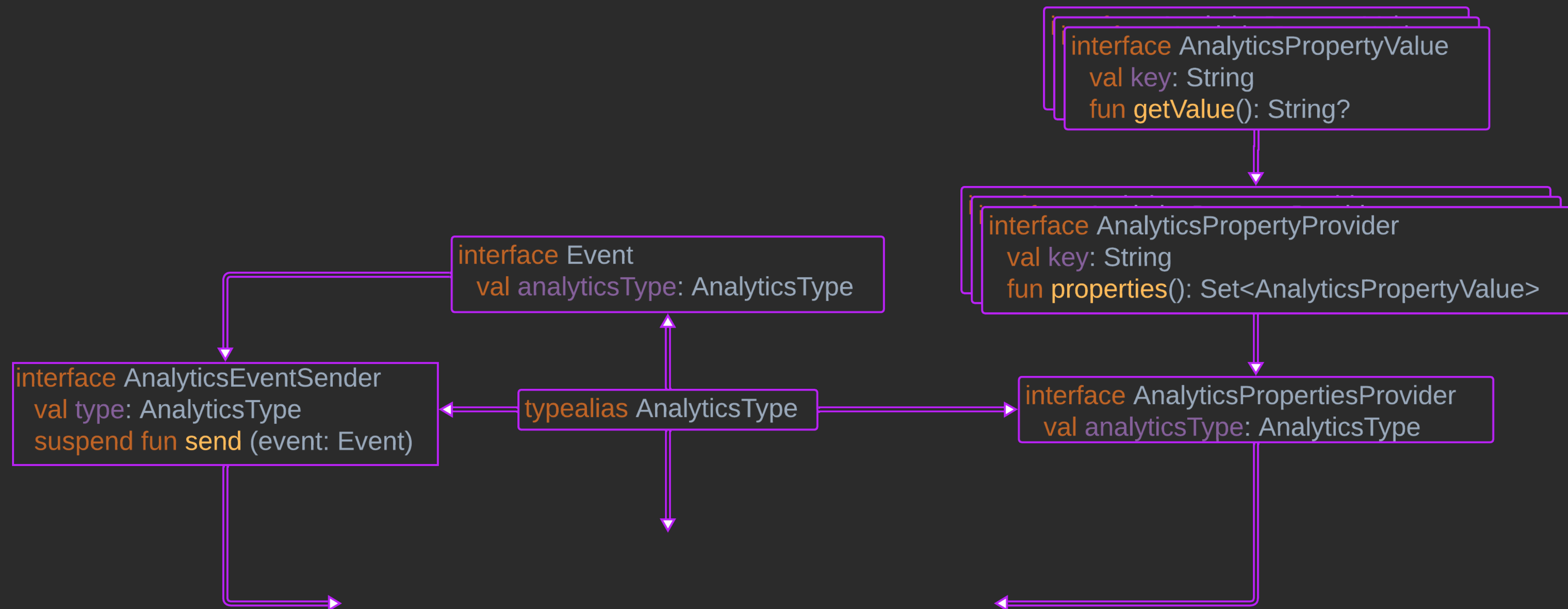
Абстракция для работы с N аналитиками. Common.



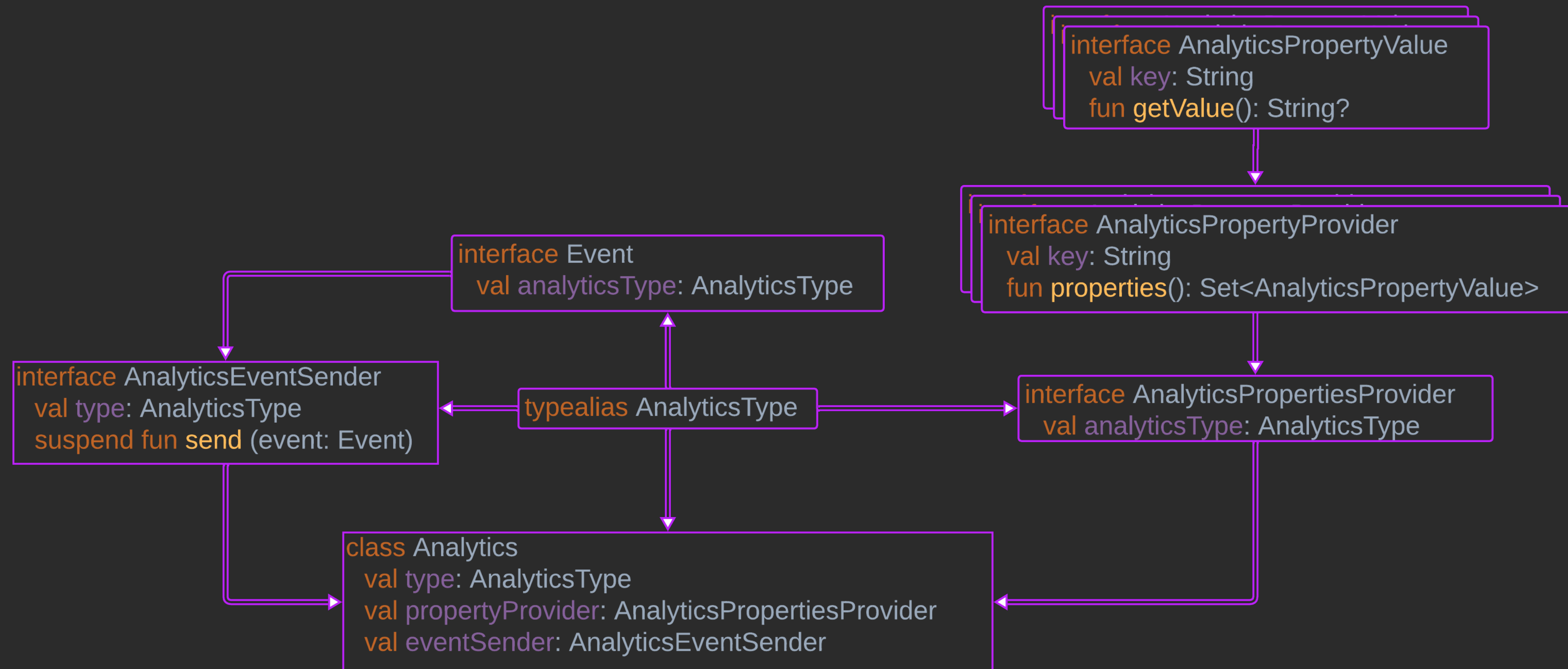
Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

class Analytics(
    val type: AnalyticsType,
    val propertyProvider: AnalyticsPropertiesProvider,
    val eventSender: AnalyticsEventSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>)
}

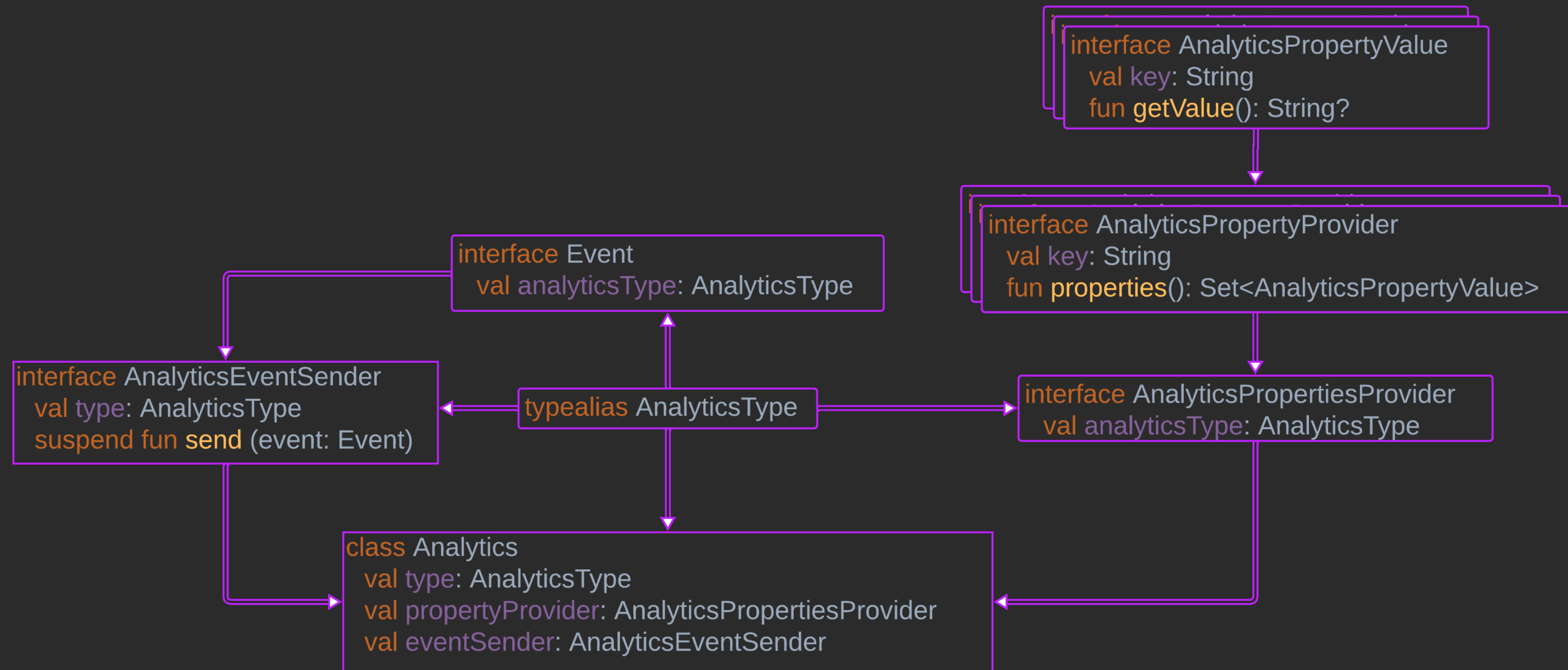
@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

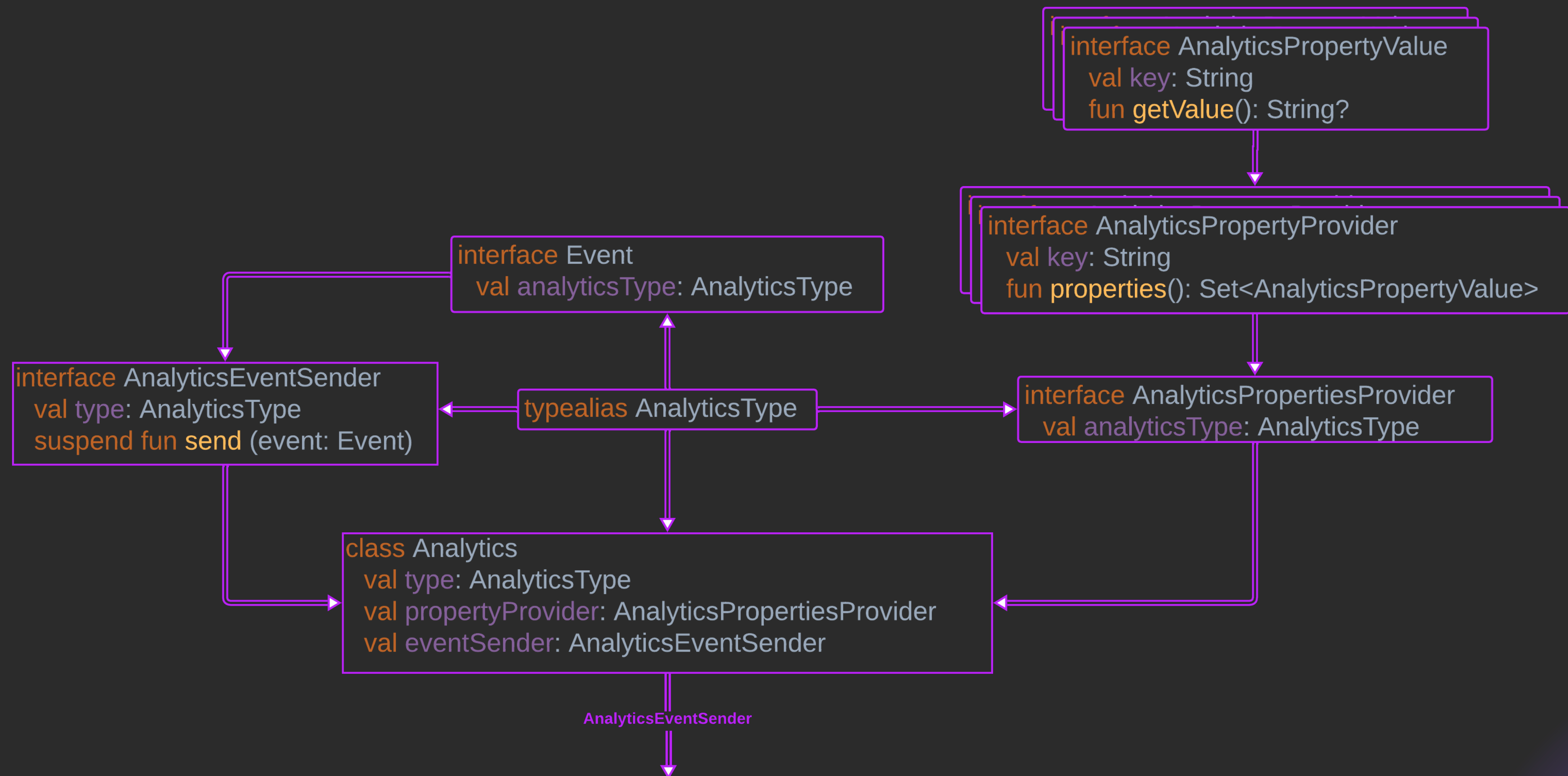
Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {  
    val type: AnalyticsType  
    suspend fun send(event: Event)  
}  
  
class Analytics(  
    val type: AnalyticsType,  
    val propertyProvider: AnalyticsPropertiesProvider,  
    val eventSender: AnalyticsEventSender,  
)  
  
internal class EventSenderImpl(  
    dispatcher: CoroutineDispatcher = Dispatchers.IO,  
    private val eventSenders: List<AnalyticsEventSender>,  
) : EventSender { ... }  
  
@AnalyticsBuilderMarker  
abstract class AnalyticsBuilder  
  
@DslMarker  
annotation class AnalyticsBuilderMarker
```

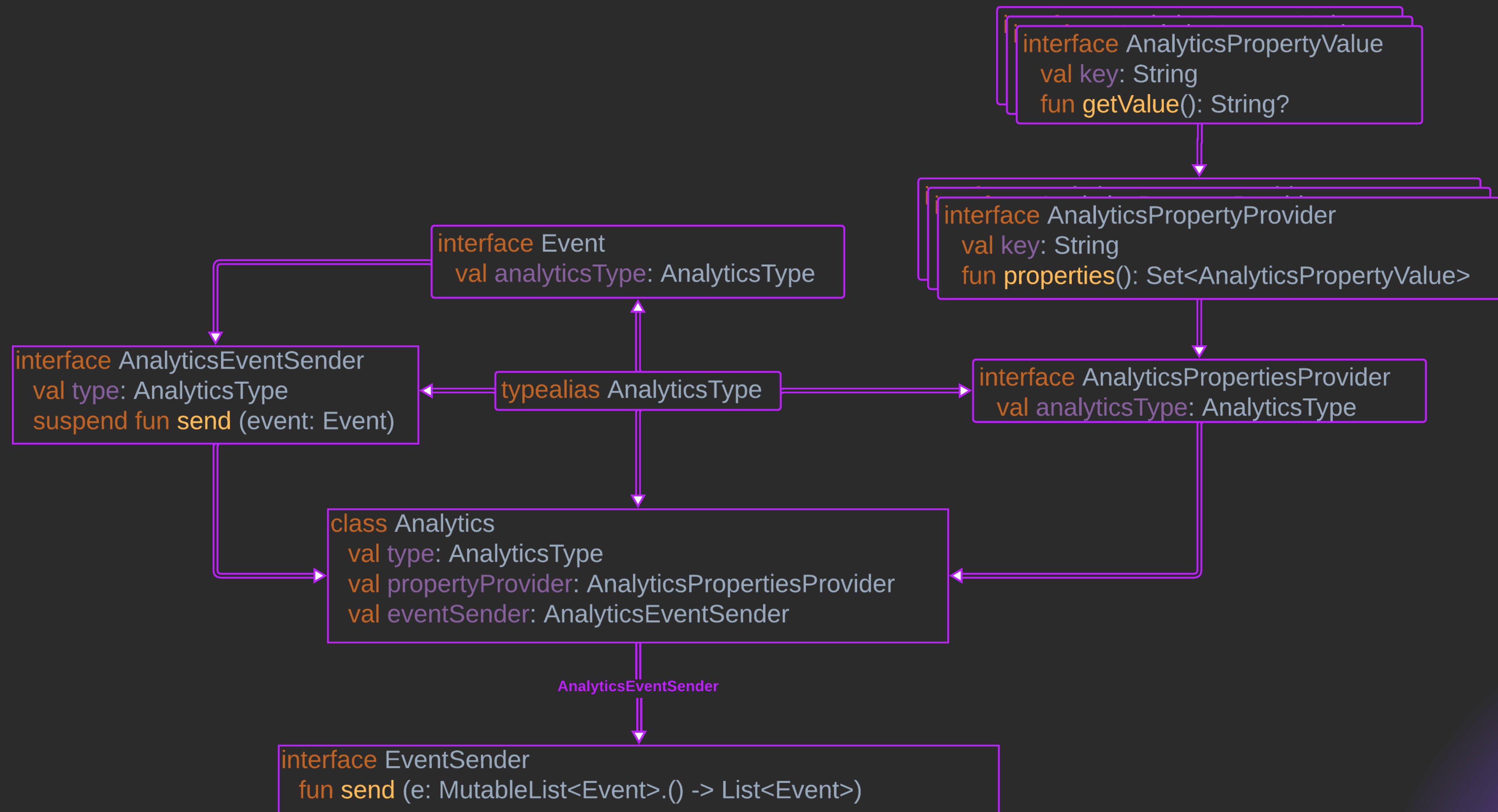
Абстракция для работы с N аналитиками. Common.



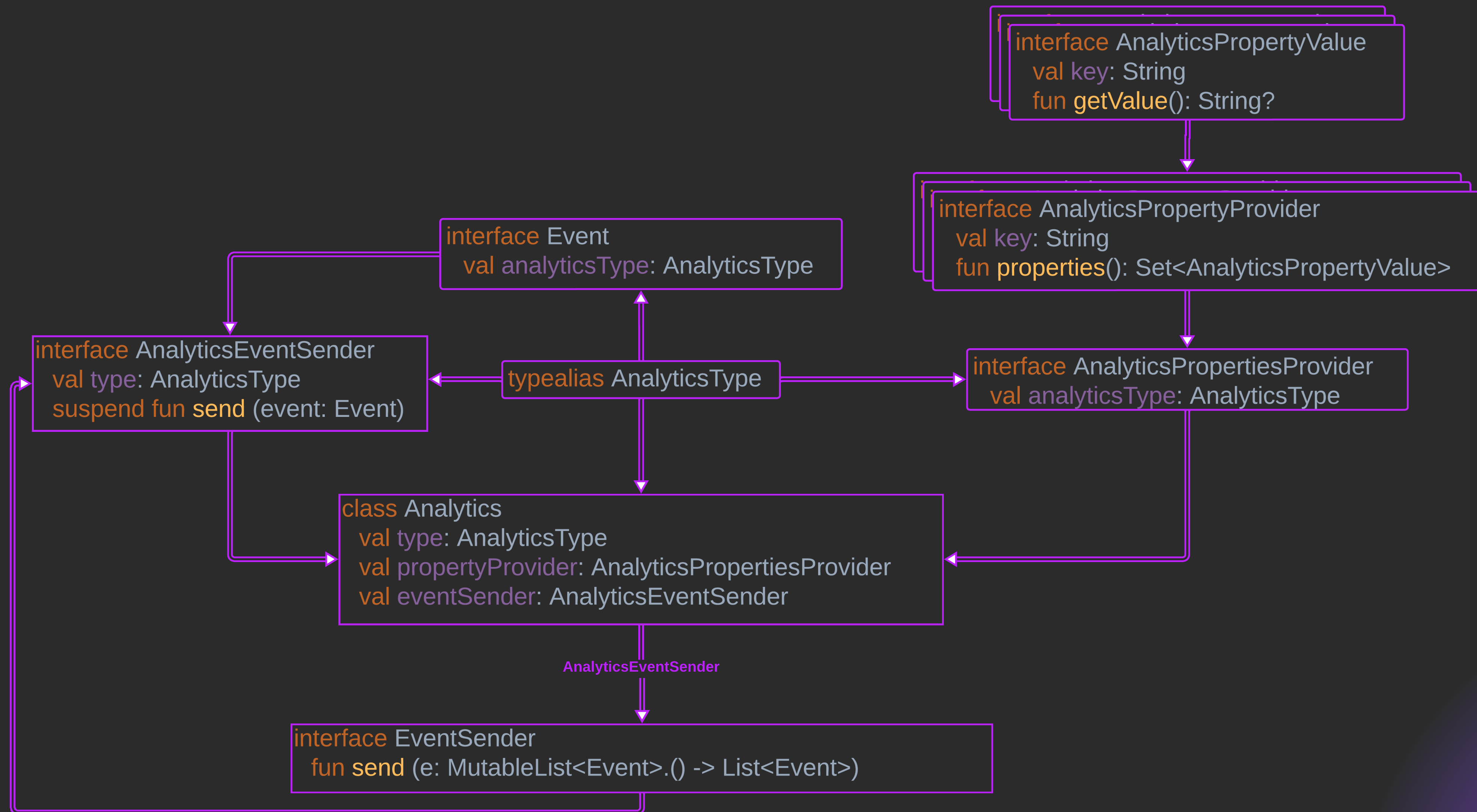
Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.



Абстракция для работы с N аналитиками. Common.

```
interface AnalyticsEventSender {
    val type: AnalyticsType
    suspend fun send(event: Event)
}

class Analytics(
    val type: AnalyticsType,
    val propertyProvider: AnalyticsPropertiesProvider,
    val eventSender: AnalyticsEventSender,
)

interface EventSender {
    fun send(addEvents: MutableList<Event>.<() -> List<Event>)
}

@AnalyticsBuilderMarker
abstract class AnalyticsBuilder

@DslMarker
annotation class AnalyticsBuilderMarker
```

Абстракция для работы с N аналитиками. Пример.



Абстракция для работы с N аналитиками. Пример.

```
class FirebaseAnalyticsBuilder : AnalyticsBuilder() {  
    private var name: String? = null  
    private var action: String? = null  
  
    fun name(name: String?): FirebaseAnalyticsBuilder {  
        this.name = name  
        return this  
    }  
  
    fun action(action: String?): FirebaseAnalyticsBuilder {  
        this.action = action  
        return this  
    }  
  
    fun build(): FirebaseEvent {  
        check(isValid())  
        return FirebaseEvent(name, action)  
    }  
}
```



Абстракция для работы с N аналитиками. Пример.

```
class AccountIdProperty : UserAnalyticsProperties {  
    // do your custom logic here  
    override val key: String = "account_id"  
  
    override fun getValue(): String = "mobius2023"  
}
```

Абстракция для работы с N аналитиками. Пример.

```
val firebase: Analytics = Firebase.create(  
    context,  
    FirebasePropertiesProvider(  
        DeviceAnalyticsPropertyProvider(accountIdProperty),  
    )  
)  
  
AnalyticsSdk.initialize(  
    listOf(  
        ...  
        firebase,  
        ...  
    )  
)
```

Абстракция для работы с N аналитиками. Пример.

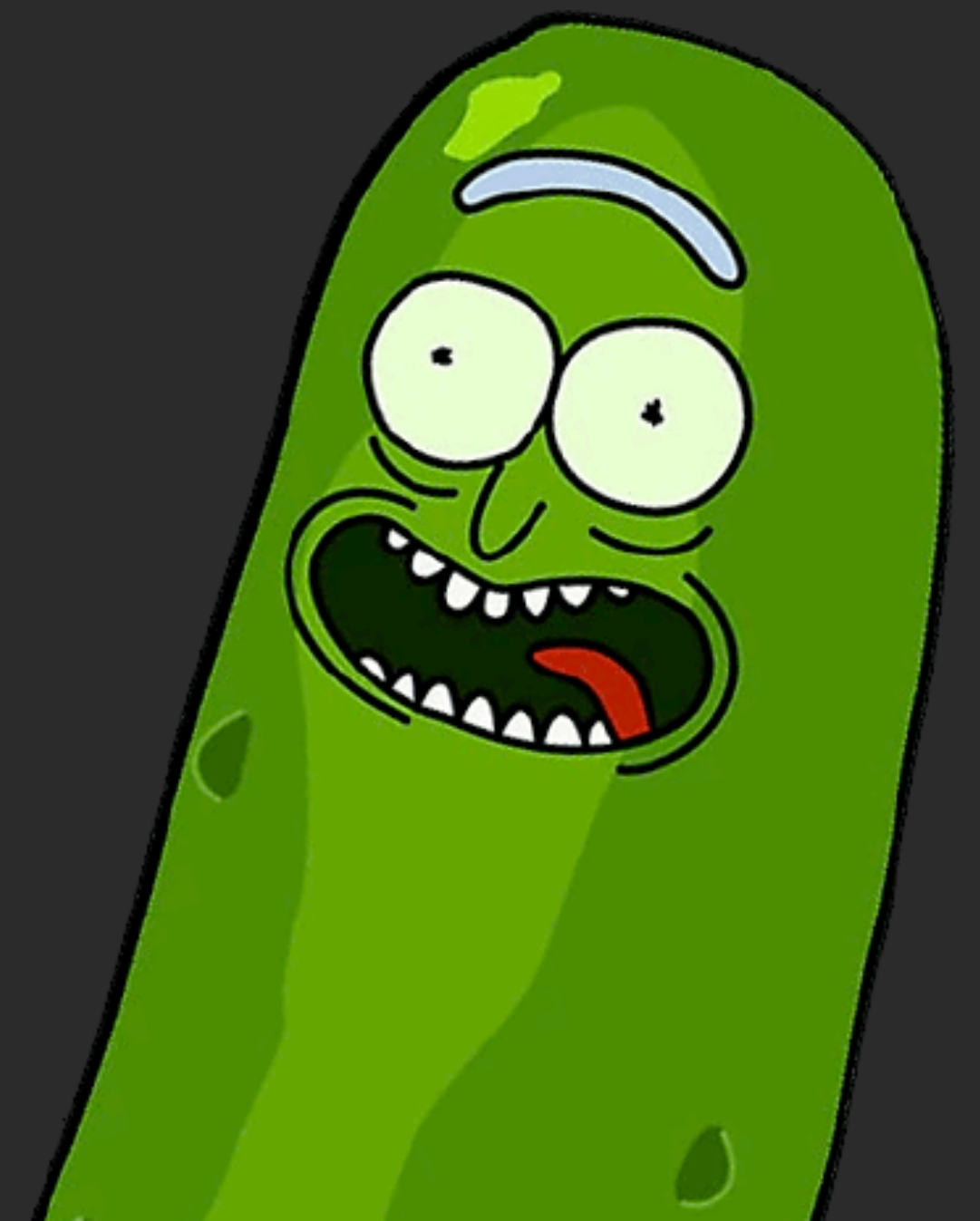
```
fun MutableList<Event>.firebase(  
    build: FirebaseAnalyticsBuilder.() -> FirebaseEvent  
) : List<Event> = also { add(FirebaseAnalyticsBuilder().build()) }
```

Абстракция для работы с N аналитиками. Пример.

```
AnalyticsSdk.getInstance().send {  
    firebase {  
        name("mobius2023")  
        action("speech")  
        build()  
    }  
}
```

```
clickStream { this: ClickStreamBuilder  
    event { this: EventPropertiesBuilder  
        event { this: EventPropertiesBuilder  
    }  
}  
}  
build() ^clickStream  
}
```

Абстракция для работы с N аналитиками. Тесты.



Абстракция для работы с N аналитиками. Тесты.

- тесты на сам факт вызова методов аналитки во вью-модели
- тесты значений которые уходят в аналитику mockk, slot, capture

Абстракция для работы с N аналитиками. Тесты.

```
@Test
fun `Check on promo category clicked`() {
    // given
    val viewModel = createViewModel()
    val position = 2
    val categoryId = 1

    // when
    viewModel.onPromoImageClicked(categoryId, position)

    // then
    verify(atLeast = 1) {
        mainPageAnalytics.promoImageClicked(categoryId, position + 1)
    }
}
```

Абстракция для работы с N аналитиками. Тесты.

```
@Test
fun promoItemImageShown() {
    // given
    val promoImageId = 1
    val position = 2

    // when
    mainPageAnalytics.promoItemImageShown(promoImageId, position)

    // then
    analytics.verifyFirebaseEvent(
        "PROMO_IMAGE",
        "promo_image_id" to promoImageId,
        "position" to position,
        "page_type" to "MAIN",
    )
}
```

Абстракция для работы с N аналитиками. Тесты.

```
fun EventSender.verifyFirebaseEvent(  
    expectedName: String,  
    expectedAction: String? = null,  
    additionalChecks: (FirebaseEvent.() -> Unit)? = null,  
) {  
    val eventSlot = slot<FirebaseEvent>()  
  
    verify {  
        send(capture(eventSlot))  
    }  
  
    verify(  
        eventSlot,  
        expectedName,  
        expectedAction,  
        additionalChecks  
    )  
}
```

Абстракция для работы с N аналитиками. Тесты.

```
fun EventSender.verifyFirebaseEvent(  
    expectedName: String,  
    expectedAction: String? = null,  
    additionalChecks: (FirebaseEvent.() -> Unit)? = null,  
) {  
    val eventSlot = slot<FirebaseEvent>()  
  
    verify {  
        send(capture(eventSlot))  
    }  
  
    verify(  
        eventSlot,  
        expectedName,  
        expectedAction,  
        additionalChecks  
    )  
}
```

Абстракция для работы с N аналитиками. Тесты.

```
fun EventSender.verifyFirebaseEvent(  
    expectedName: String,  
    expectedAction: String? = null,  
    additionalChecks: (FirebaseEvent.() -> Unit)? = null,  
) {  
    val eventSlot = slot<FirebaseEvent>()  
  
    verify {  
        send(capture(eventSlot))  
    }  
  
    verify(  
        eventSlot,  
        expectedName,  
        expectedAction,  
        additionalChecks  
    )  
}
```

Абстракция для работы с N аналитиками. Common.

- один интерфейс в фича-модулях (ложится на многомодульность)
- подключать новые реализации аналитик отдельно (firebase, backend, etc...)
- конфигурировать события перед отправкой
- отправлять события в разные аналитики

Абстракция для работы с N аналитиками. Common.

- покрывать тестами и тестировать сами события и данные
- нет андройд зависимостей в common модуле
- билдеры Kotlin DSL
- отсутствие миграций в БД, возможность расширения properties клиентами
- минимум библиотечных зависимостей

Выводы и результаты

- Вынесли максимум на бекенд, но оставили возможность подключать модули с другими SDK при необходимости
- Все взаимодействие для клиентов происходит через один интерфейс
- Упростили работу фича-команд, других отделов



Выводы и результаты

- События не теряются
- Консистентность
- Проще поддерживать и конфигурировать



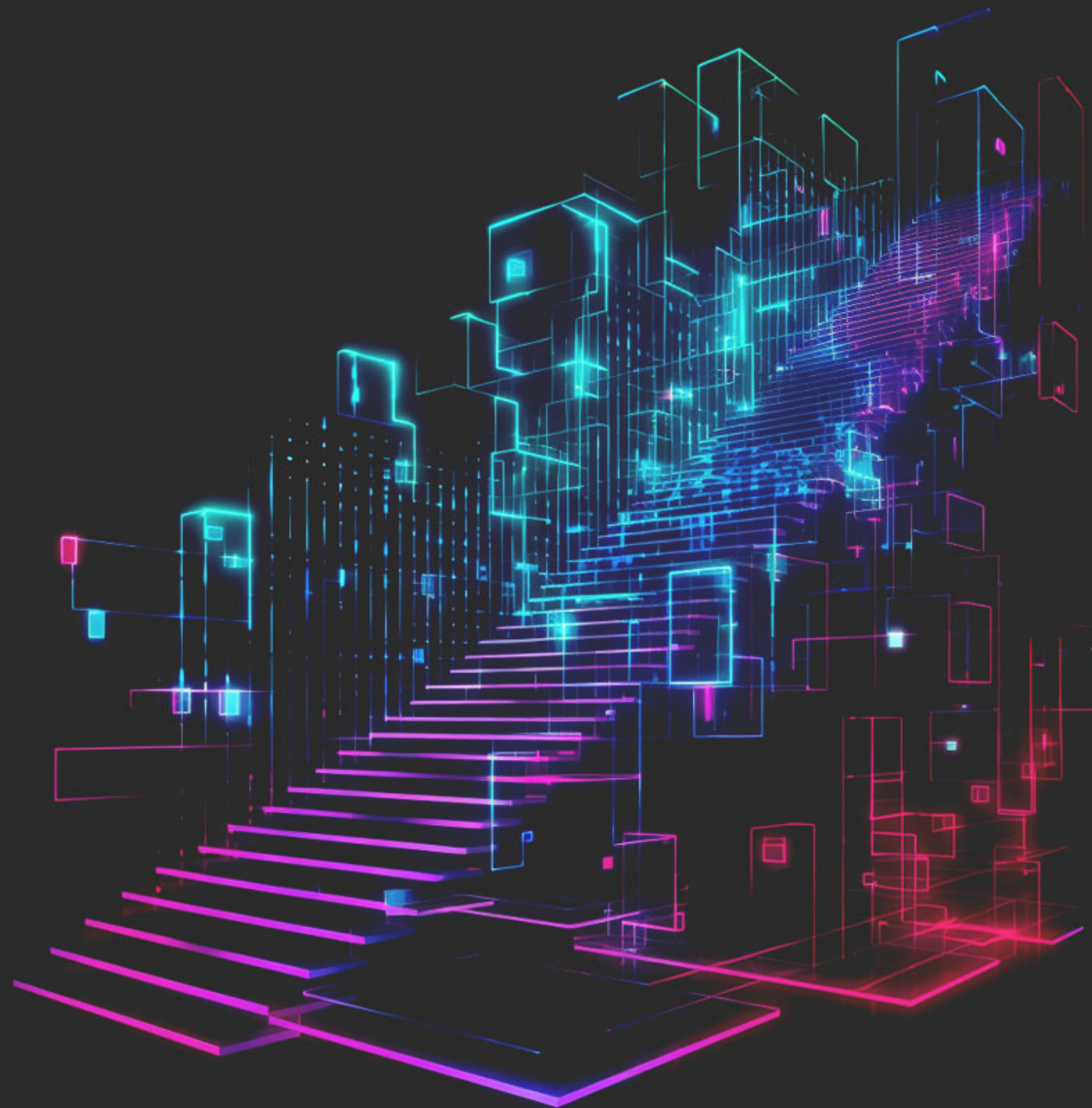
Выводы и результаты

- Увеличили покрытие кода, код стал чище, конфликтов меньше и т.д.
- Аналитика ломается реже, если не сломана изначально :)
- События не тухнут и не теряются в коде



Планы на будущее

- Добавить дебаг меню что бы смотреть на события без прокси
- Так же есть другие клиенты: iOS / Web / Flutter -> KMM?



Риски

- KMM > WorkManager, Room
- Как легко поддерживать? (~2 дня на добавление нового модуля)
- Отправка на наш сервер может сломаться из-за ошибок в базе данных

