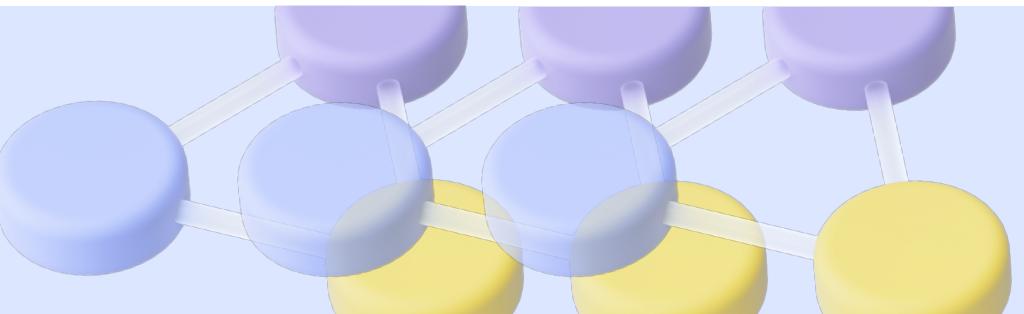




# Как я писал клиента для YDB на Rust, сравнение с Go.

Тимофей Кулин,  
YDB  
старший разработчик

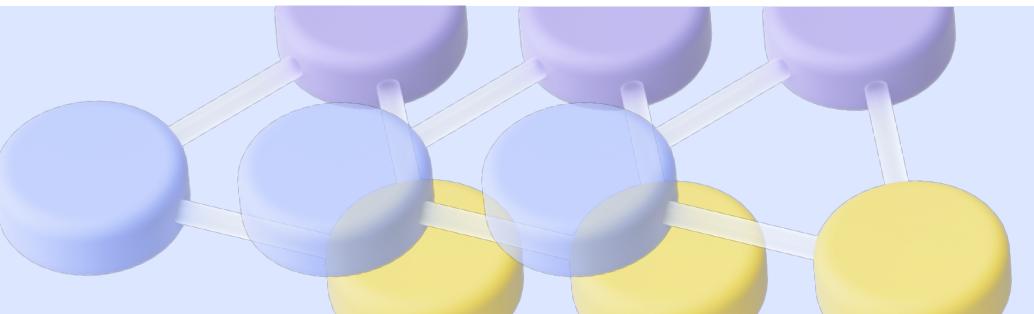
# YDB — Open-Source Distributed SQL Database



# YDB – Open-Source Distributed SQL Database

## Распределённая

- Запускается в нескольких зонах доступности (AZ)
- Переживает пропажу одной зоны доступности + одной стойки без вмешательства человека



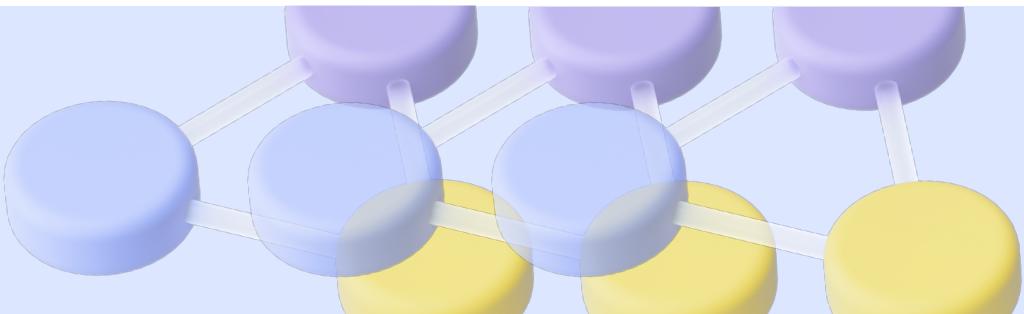
# YDB – Open-Source Distributed SQL Database

## Распределённая

- Запускается в нескольких зонах доступности (AZ)
- Переживает пропажу одной зоны доступности + одной стойки без вмешательства человека

## Для критичных задач

- Работа 24x7
- Обновления без простоев
- Строгая консистентность данных



# YDB – Open-Source Distributed SQL Database

## Распределённая

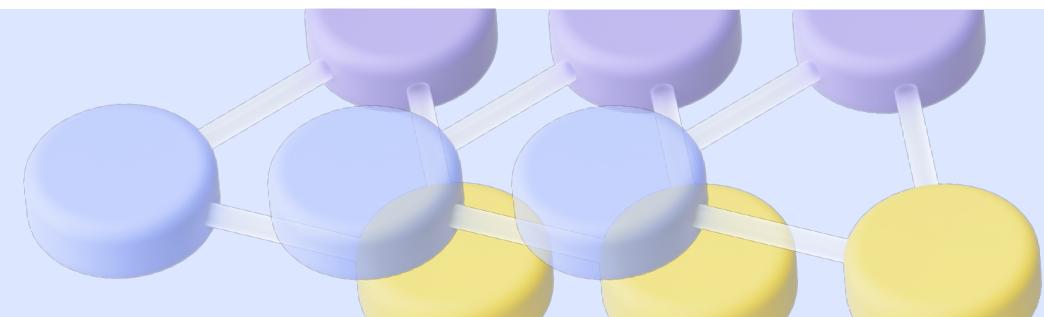
- Запускается в нескольких зонах доступности (AZ)
- Переживает пропажу одной зоны доступности + одной стойки без вмешательства человека

## Для критических задач

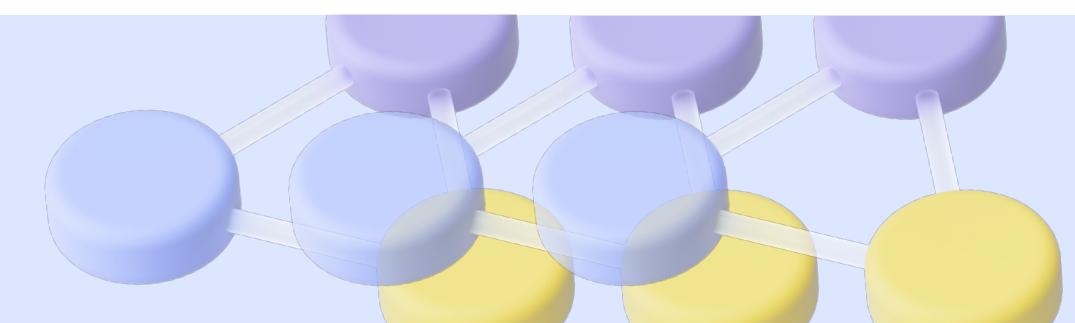
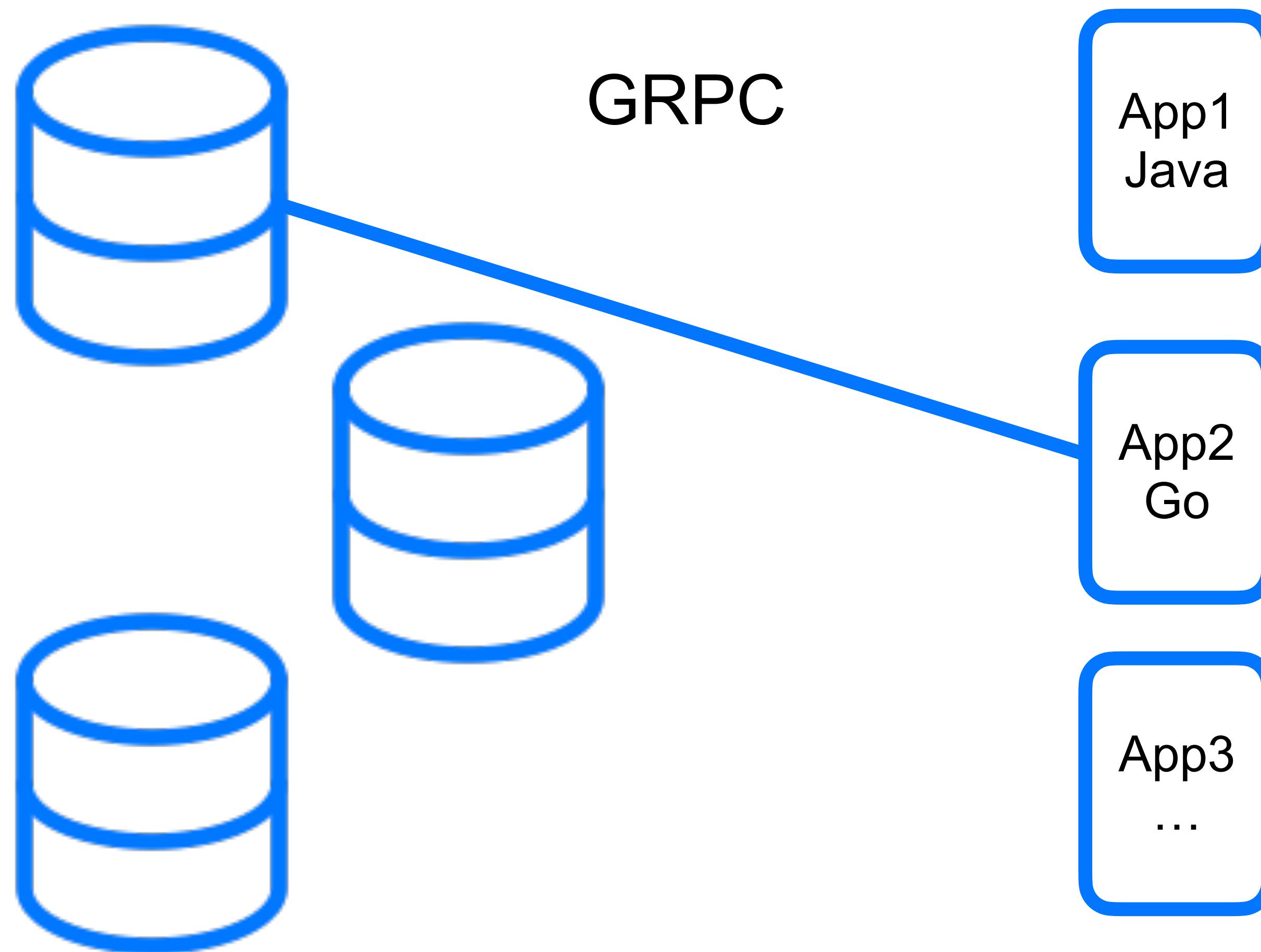
- Работа 24x7
- Обновления без простоев
- Строгая консистентность данных

## Платформа

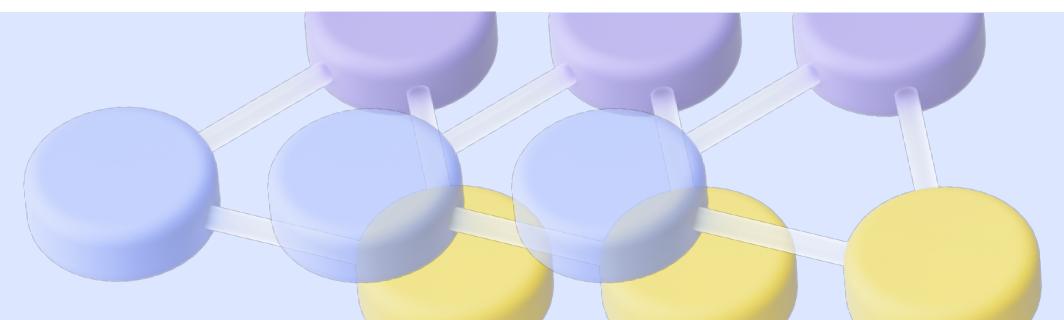
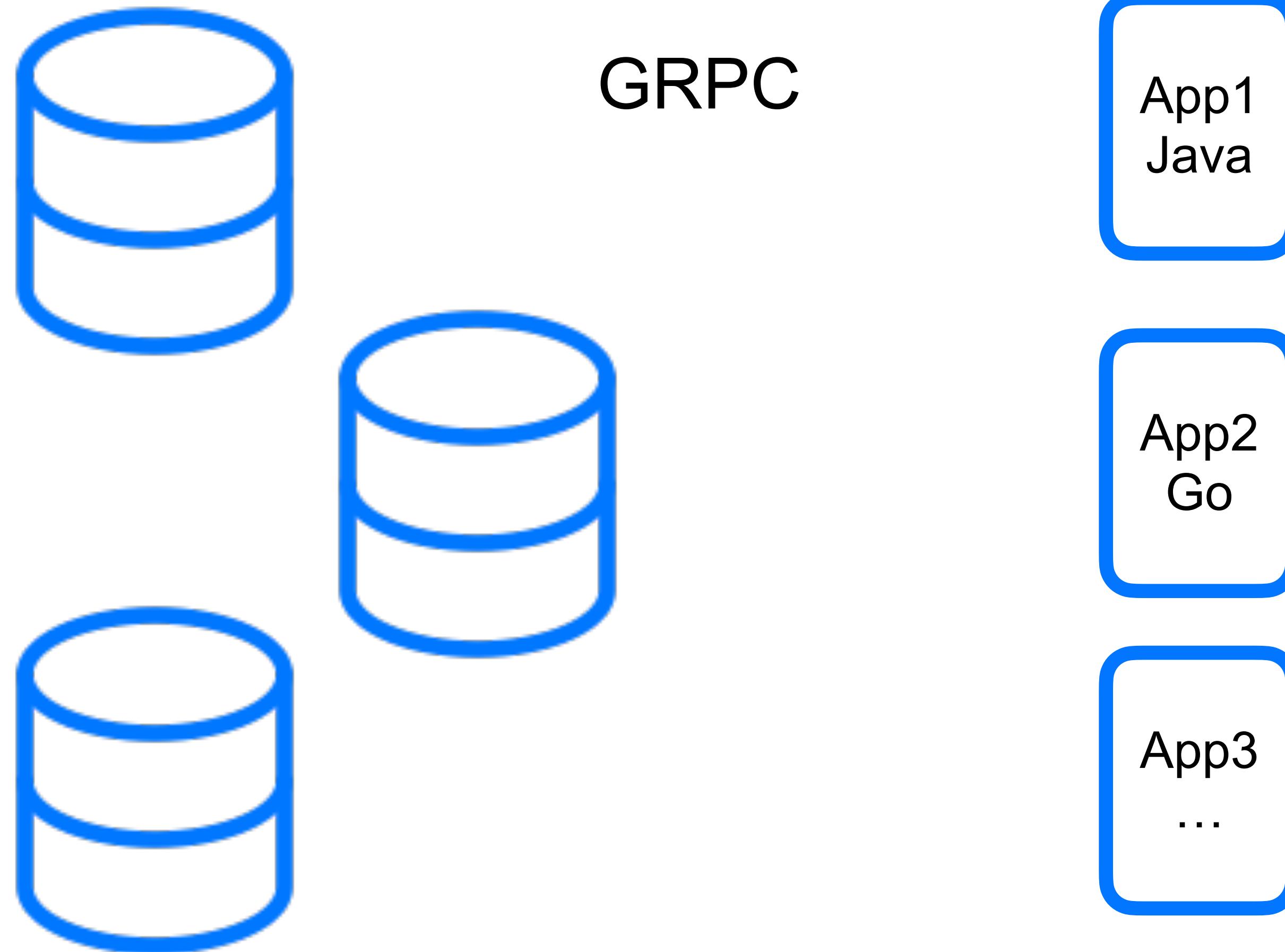
- Таблицы, топики, rate-limiter, ...



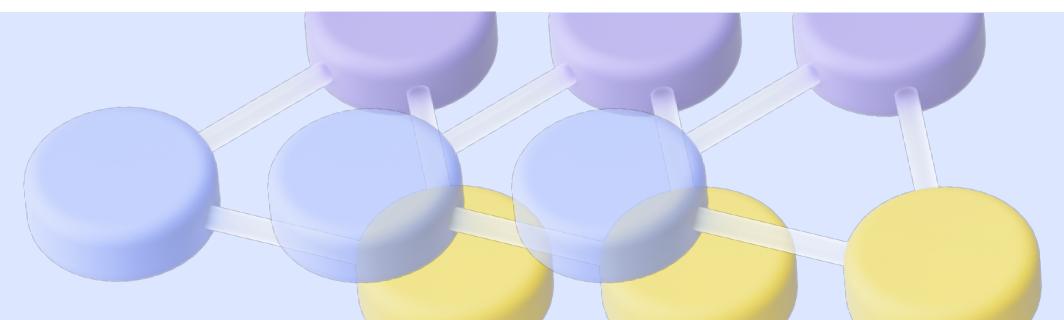
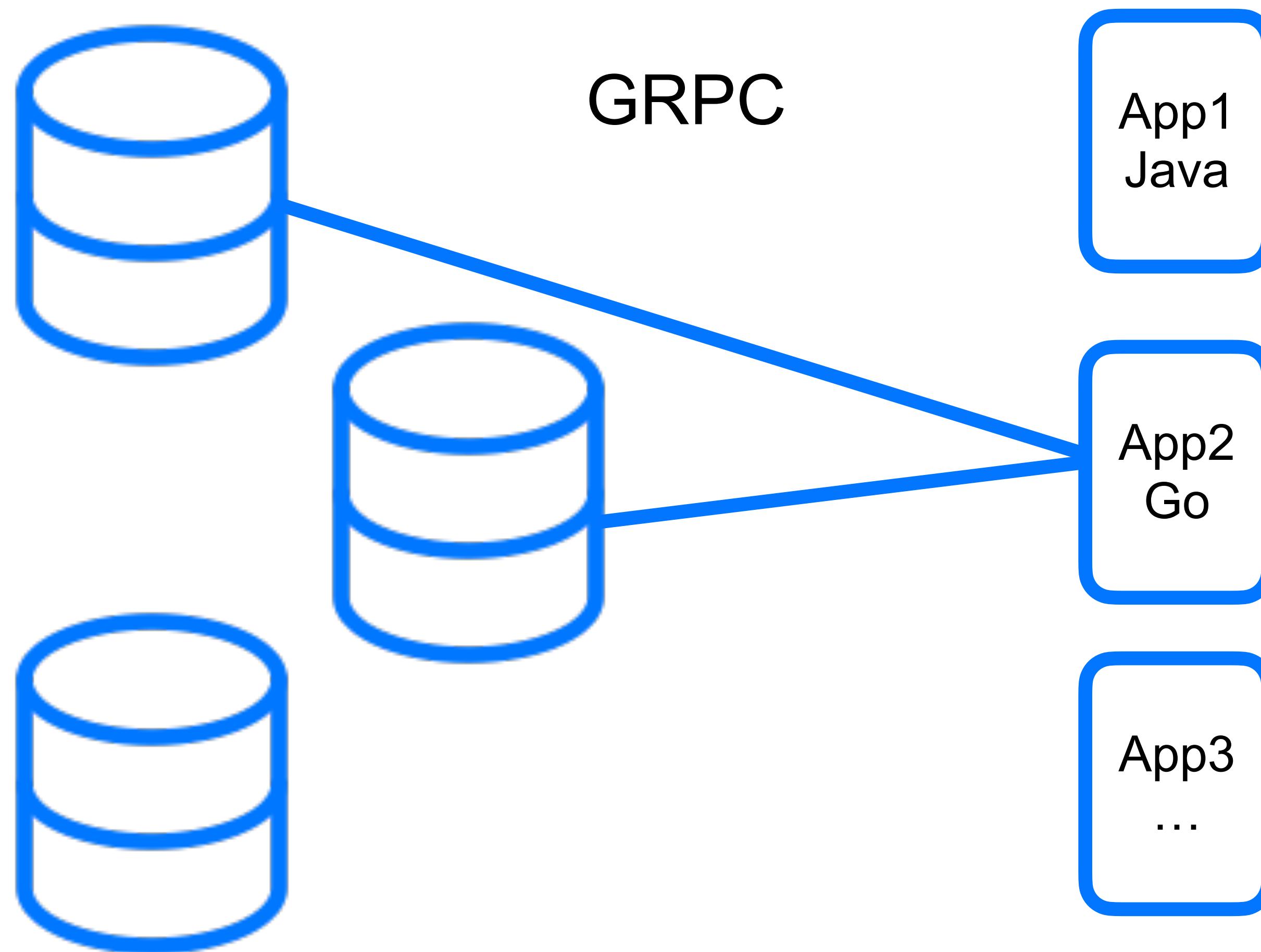
# YDB — распределённая, с клиентской балансировкой



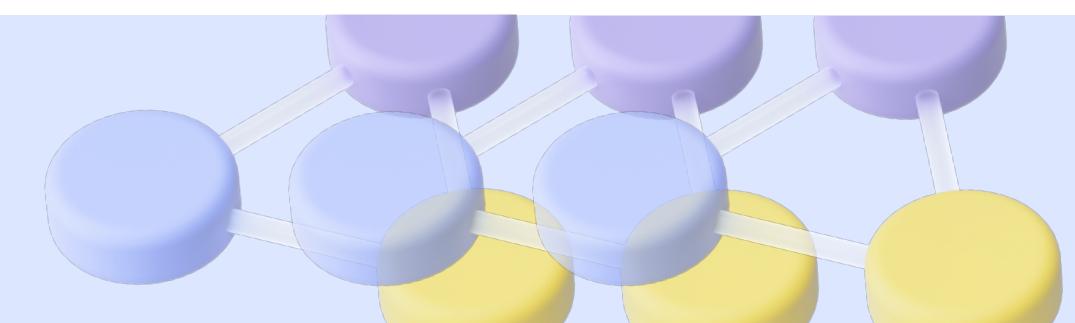
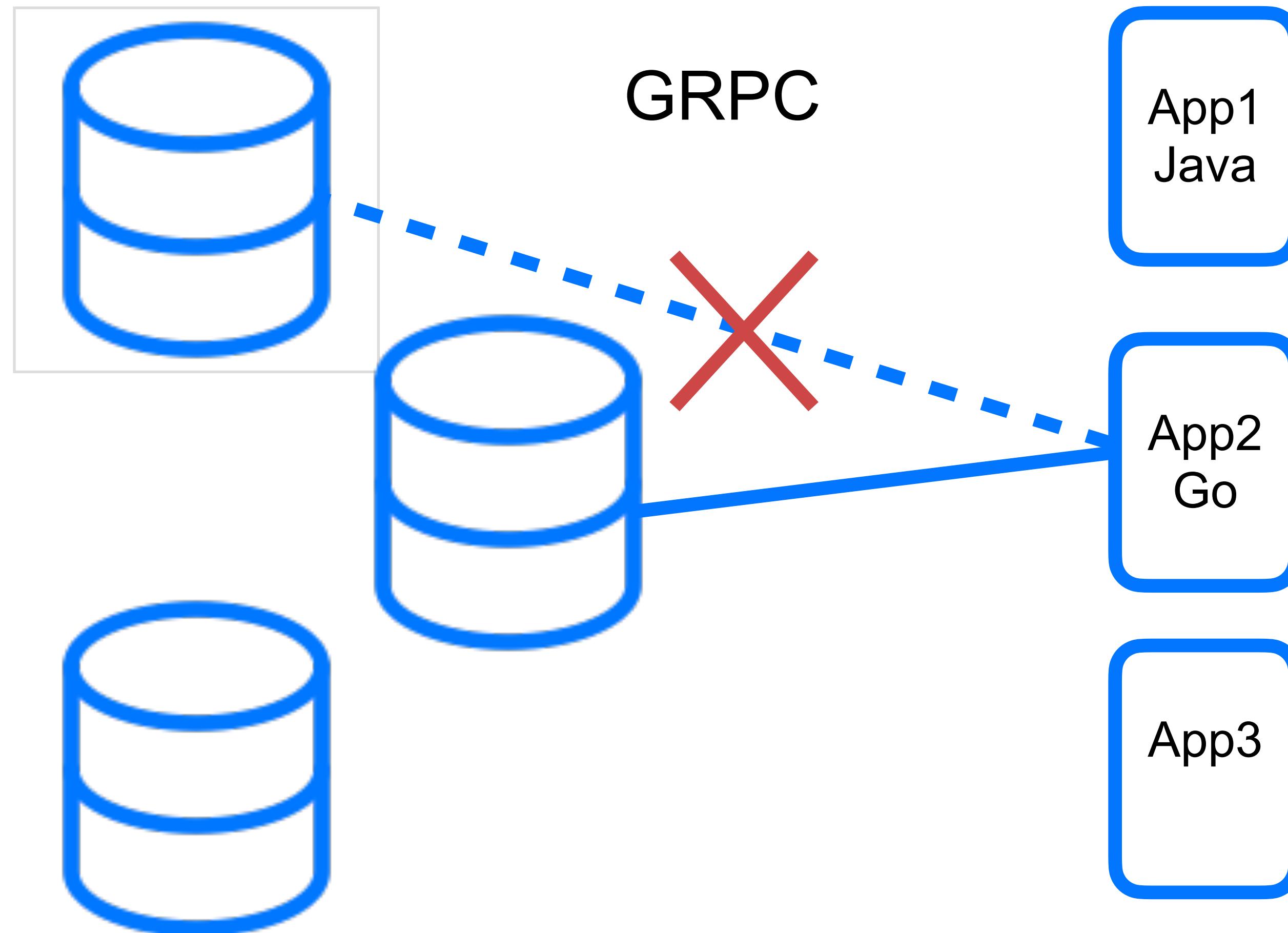
# YDB — распределённая, с клиентской балансировкой



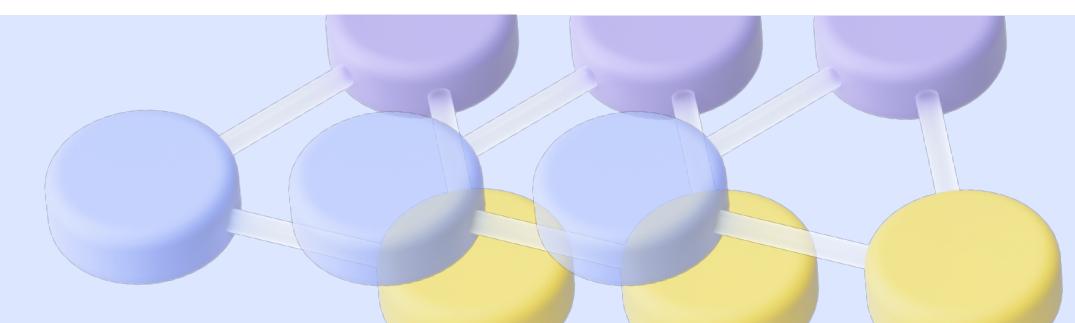
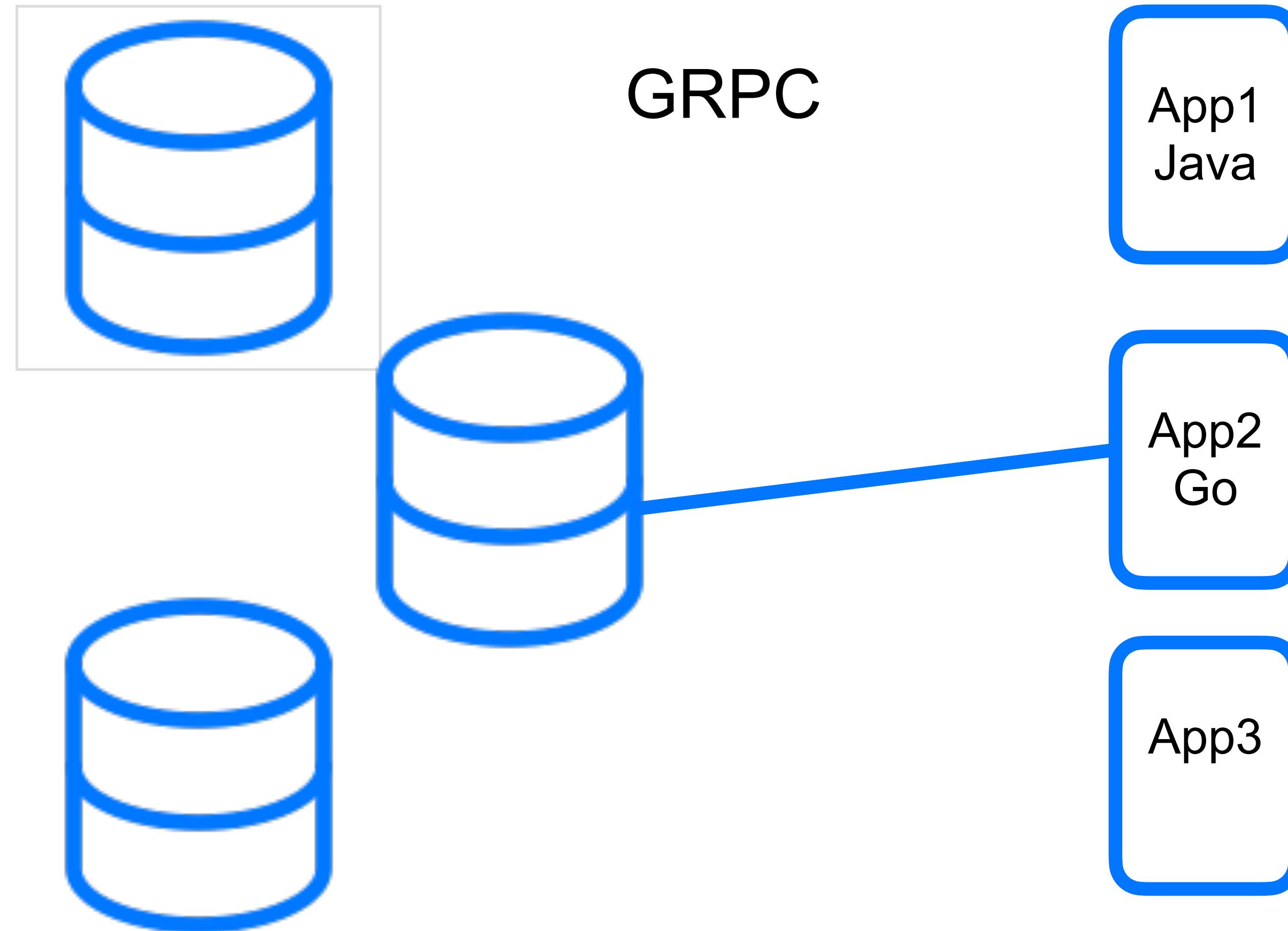
# YDB — распределённая, с клиентской балансировкой



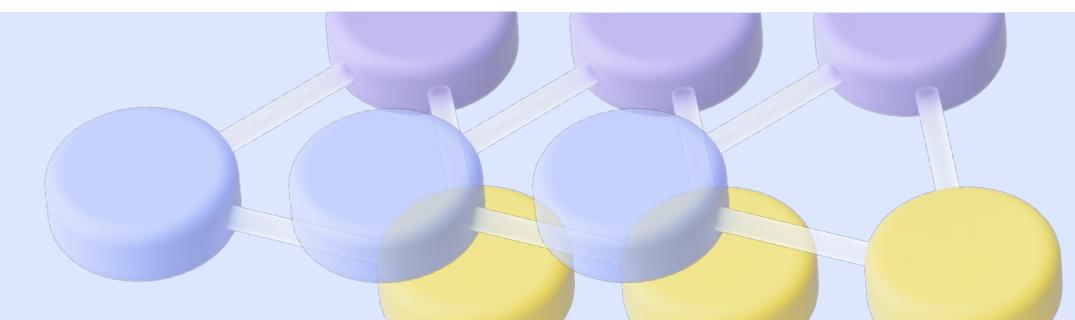
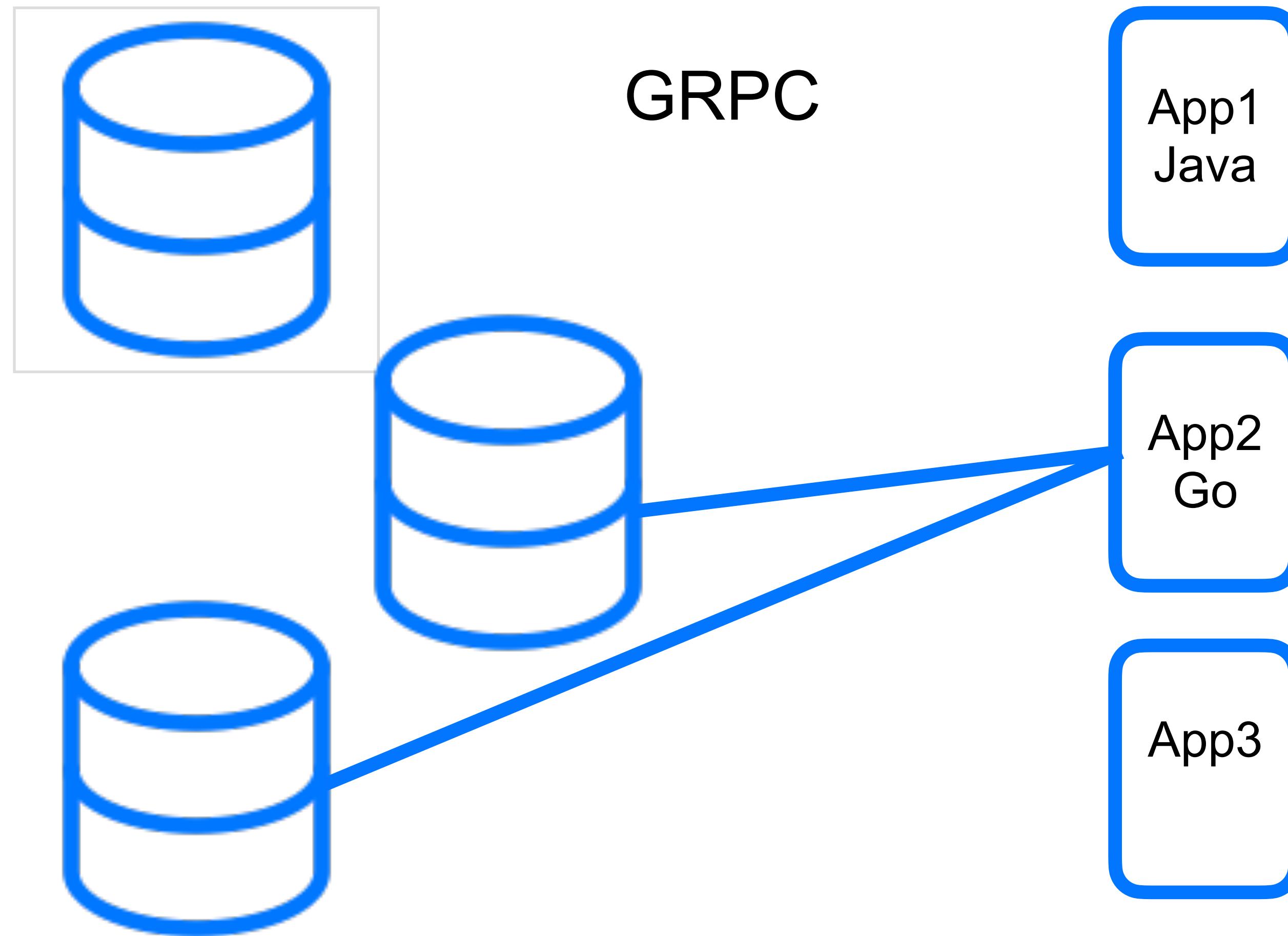
# YDB — распределённая, с клиентской балансировкой



# YDB — распределённая, с клиентской балансировкой



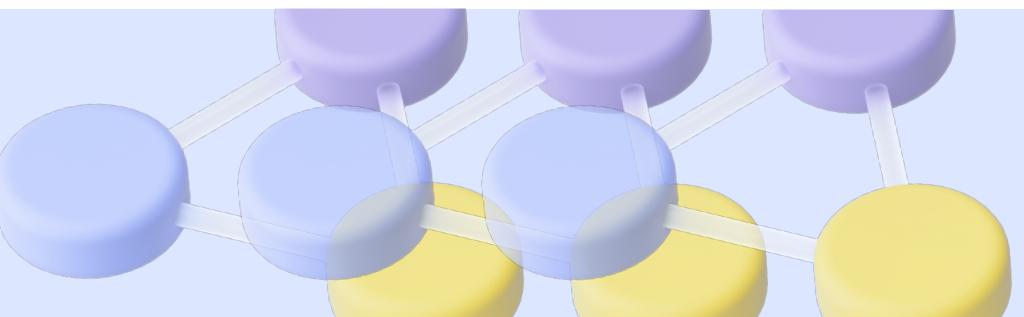
# YDB — распределённая, с клиентской балансировкой



# GRPC/protobuf, генерирование кода

## GO

- protoc ...
- И можно пользоваться



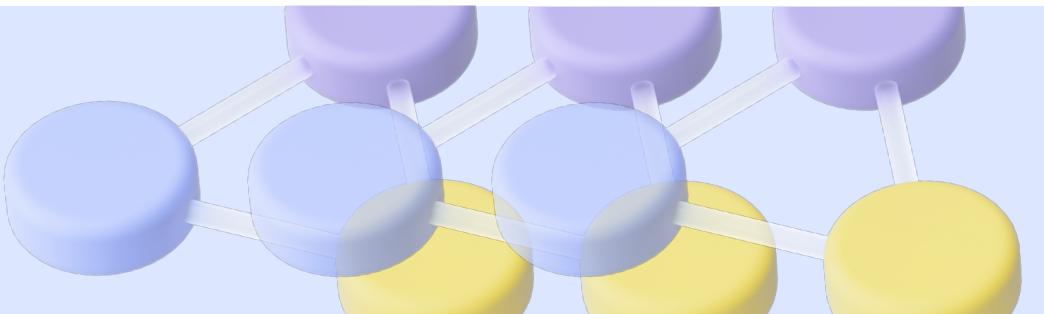
# GRPC/protobuf, генерирование кода

GO

- protoc ...
- И можно пользоваться

Rust

- **tonic\_build**



# GRPC/protobuf, генерирование кода

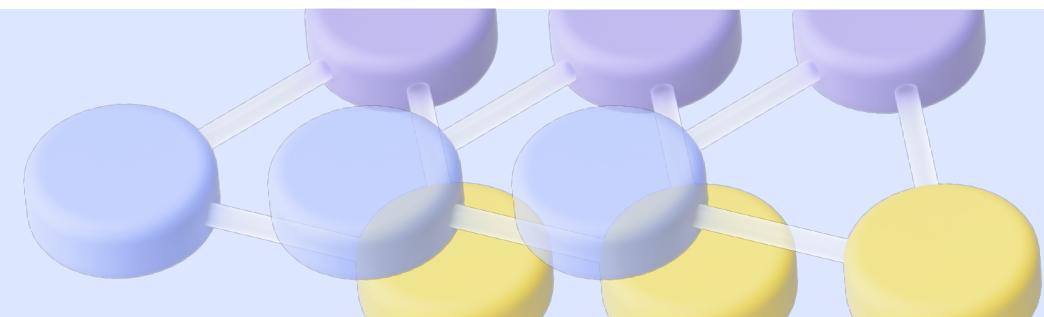
## GO

- protoc ...
- И можно пользоваться

## Rust

- tonic\_build
- **Составить правильный mod.rs**

```
pub mod ydb {  
    include!("ydb.rs");  
    pub mod cms {  
        include!("ydb.cms.rs");  
        pub mod v1 {  
            include!("ydb.cms.v1.rs");  
        }  
    }  
    ...  
}
```



# GRPC/protobuf, генерирование кода

## GO

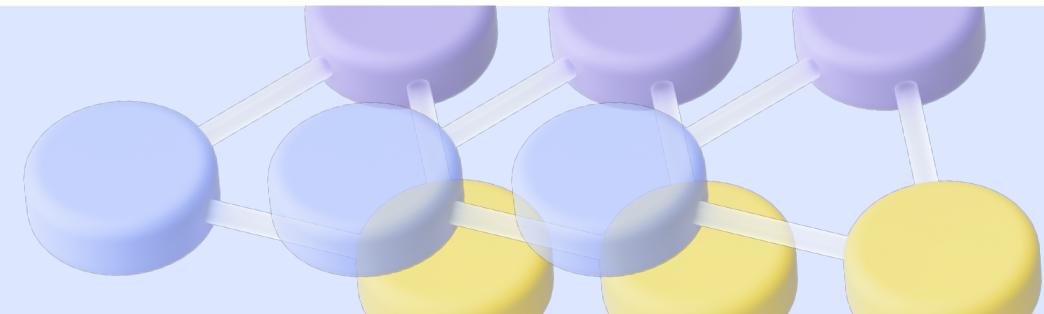
- protoc ...
- и можно пользоваться

## Rust

- tonic\_build
- Составить правильный mod.rs

```
pub mod ydb {  
    include!( "ydb.rs" );  
    pub mod cms {  
        include!( "ydb.cms.rs" );  
        pub mod v1 {  
            include!( "ydb.cms.v1.rs" );  
        }  
    }  
...  
}
```

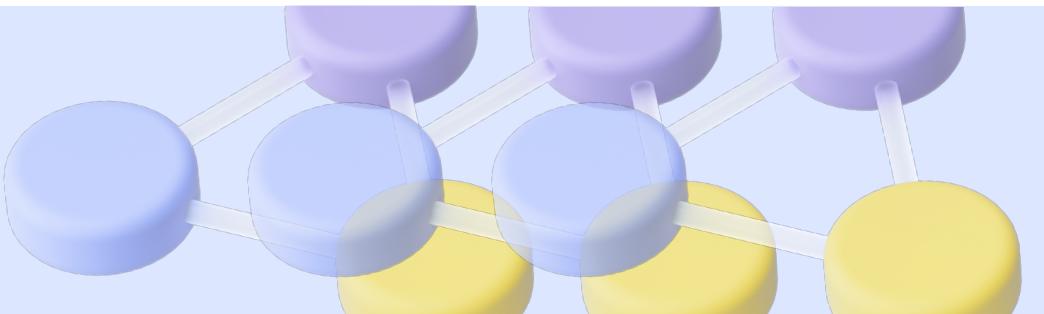
- **Сейчас уже лучше** (prost/tonic научились генерировать файл)



# Protobuf-типы json и другие представления

GO

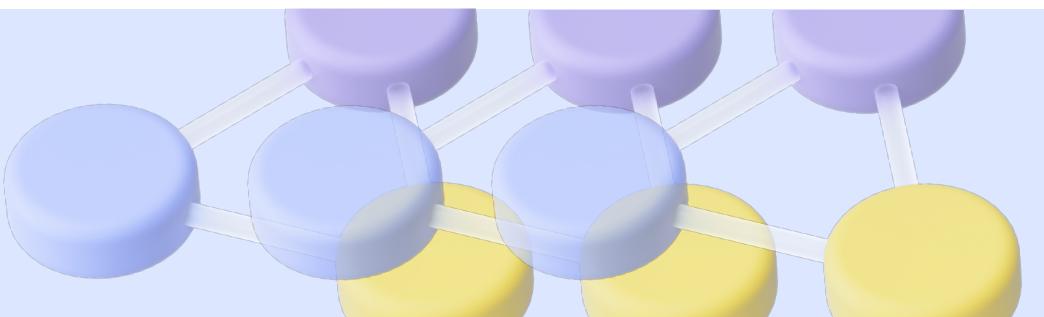
- 



# Protobuf-типы json и другие представления

GO

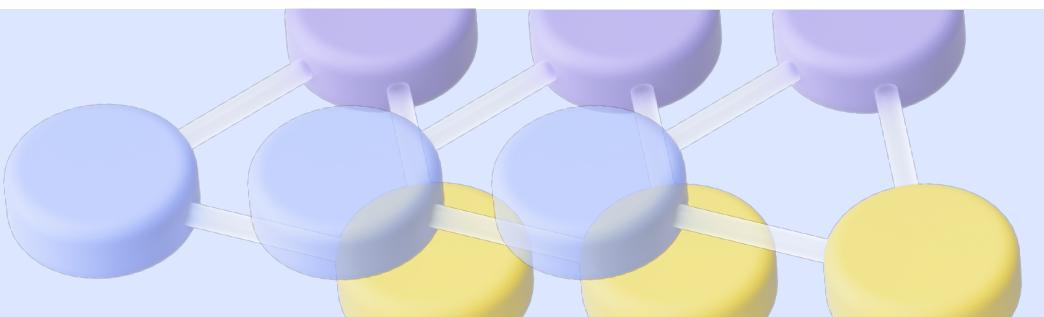
- Можно пользоваться сразу
- 



# Protobuf-типы json и другие представления

## GO

- Можно пользоваться сразу
- В большинство форматов возможна сериализация через `reflection` - не требуется поддержка в пакете со сгенерированным кодом



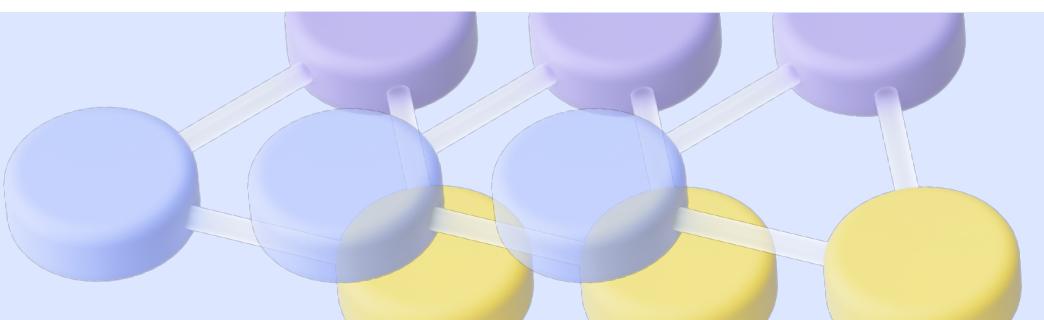
# Protobuf-типы json и другие представления

## GO

- Можно пользоваться сразу
- В большинство форматов возможна сериализация через reflection - не требуется поддержка в пакете со сгенерированным кодом

## Rust

- **Reflection отсутствует, добавлять поддержку нужно на этапе компиляции**



# Protobuf-типы json и другие представления

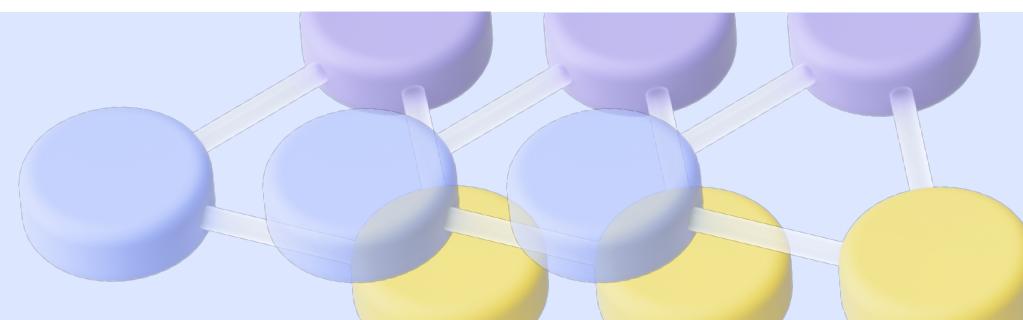
## GO

- Можно пользоваться сразу
- В большинство форматов возможна сериализация через reflection - не требуется поддержка в пакете со сгенерированным кодом

## Rust

- Reflection отсутствует, добавлять поддержку нужно на этапе компиляции
- **Поддержка должна быть добавлена в крейте со сгенерированными протобафами и зависит от поставщика крейта.**

```
.type_attribute(  
    ".Ydb",  
    "#[derive(serde::Serialize, serde::Deserialize)]"  
)  
.type_attribute(  
    "google.protobuf.Timestamp",  
    "#[derive(serde::Serialize,  
    serde::Deserialize)]",  
)  
...
```

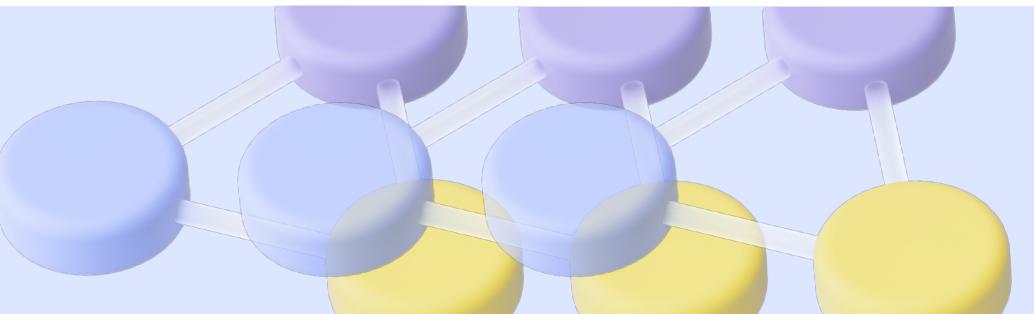


# GRPC, обёртки

## GO

- Генерированный код работает с grpc через интерфейсы.  
Можно заменять реализации без изменения системы типов.

```
func New(cc grpc.ClientConnInterface, config
config.Config) *Client {
    return &Client{
        config: config,
        service:
Ydb_Scheme_V1.NewSchemeServiceClient(cc),
    }
}
type ClientConnInterface interface {
    Invoke(...) error
    NewStream(...) (ClientStream, error)
}
```



# GRPC, обёртки

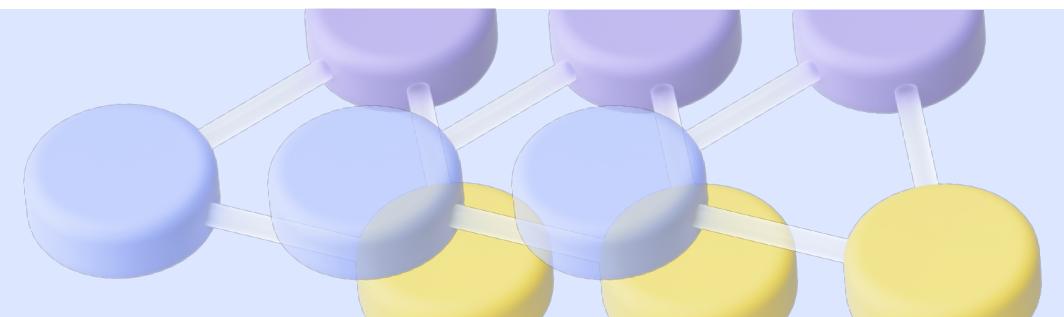
## GO

- Генерированный код работает с grpc через интерфейсы.  
Можно заменять реализации без изменения системы типов.

```
func New(cc grpc.ClientConnInterface, config
config.Config) *Client {
    return &Client{
        config: config,
        service:
Ydb_Scheme_V1.NewSchemeServiceClient(cc),
    }
}
type ClientConnInterface interface {
    Invoke(...) error
    NewStream(...) (ClientStream, error)
}
```

## Rust

- Интерцепторы добавляются только к сервисам**



# GRPC, обёртки

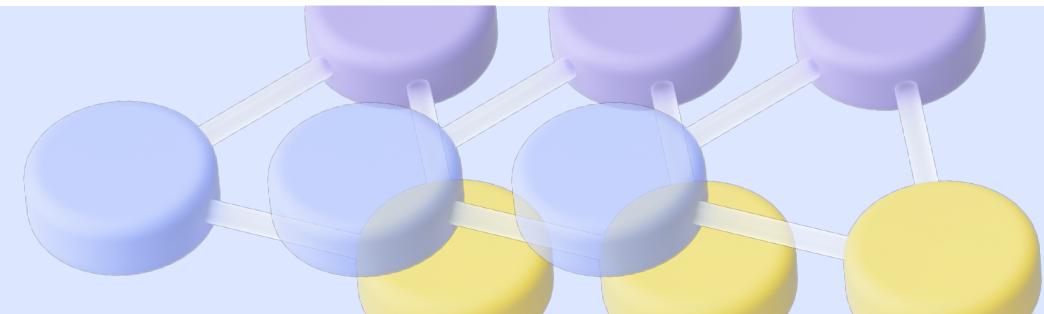
## GO

- Генерированный код работает с grpc через интерфейсы. Можно заменять реализации без изменения системы типов.

```
func New(cc grpc.ClientConnInterface, config
config.Config) *Client {
    return &Client{
        config: config,
        service:
Ydb_Scheme_V1.NewSchemeServiceClient(cc),
    }
}
```

## Rust

- Интерцепторы добавляются только к сервисам
- При добавлении интерцептора - тип меняется**



# GRPC, обёртки

## GO

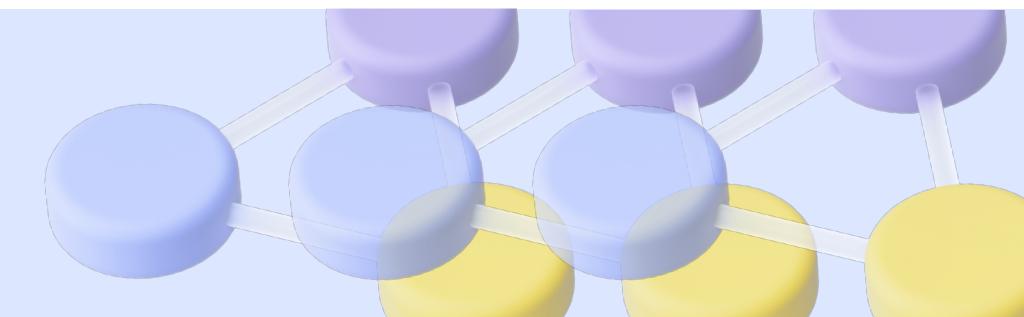
- Генерированный код работает с grpc через интерфейсы.  
Можно заменять реализации без изменения системы  
типов.

```
func New(cc grpc.ClientConnInterface, config
config.Config) *Client {
    return &Client{
        config: config,
        service:
Ydb_Scheme_V1.NewSchemeServiceClient(cc),
    }
}
```

## Rust

- Интерцепторы добавляются только к сервисам
- При добавлении интерцептора - тип меняется
- **Генерированный код - generic, зависящий от других generic-ов**

```
impl<T> SchemeServiceClient<T>
where
    T:
tonic::client::GrpcService<tonic::body::BoxBody>,
    T::Error: Into<StdError>,
    T::ResponseBody: Body<Data = Bytes> + Send +
'static,
    <T::ResponseBody as Body>::Error:
Into<StdError> + Send,
```



# GRPC, своя поддержка интерцепторов

```
pub(crate) struct InterceptedChannel {
    inner: Channel,
    interceptor: MultiInterceptor,
}

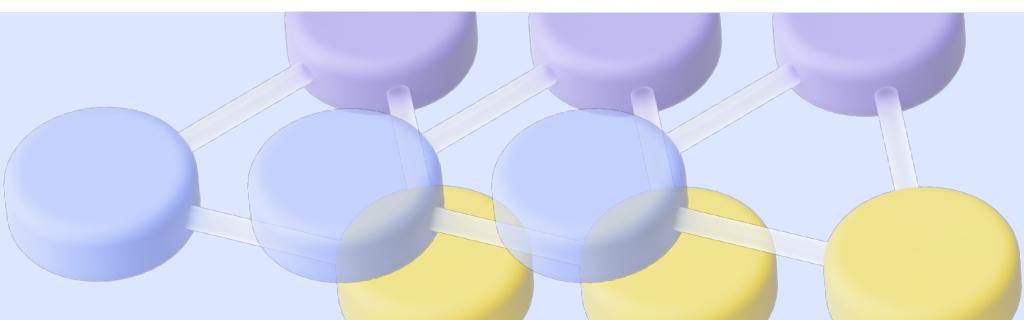
impl tower::Service<InterceptorRequest> for InterceptedChannel {
    type Response = ChannelResponse;
    type Error = InterceptorError;
    type Future = ChannelFuture;
    ...
}

pub(crate) type ChannelResponse = <Channel as tower::Service<InterceptorRequest>>::Response;

pub(crate) enum ChannelFuture {
    Error(Option<InterceptorError>),
    Future(ChannelFutureState),
}

pub(crate) struct ChannelFutureState {
    channel_future: <Channel as tower::Service<InterceptorRequest>>::Future,
    interceptor: MultiInterceptor,
    metadata: RequestMetadata,
}

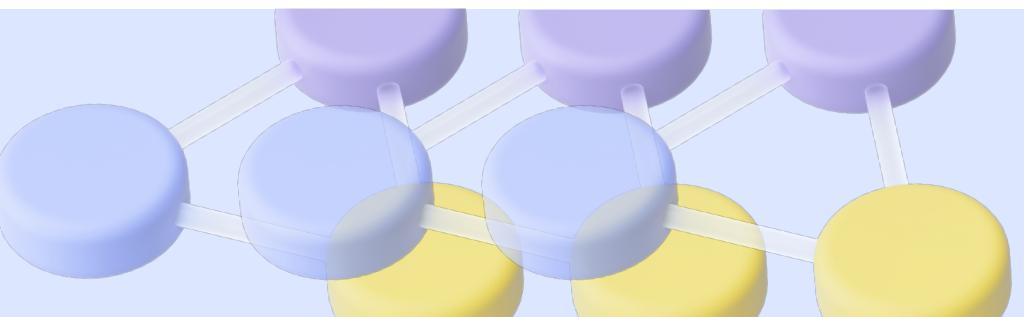
impl Future for ChannelFuture {
    type Output = std::result::Result<ChannelResponse, InterceptorError>;
    ...
}
...
```



# Управление таймаутами

GO

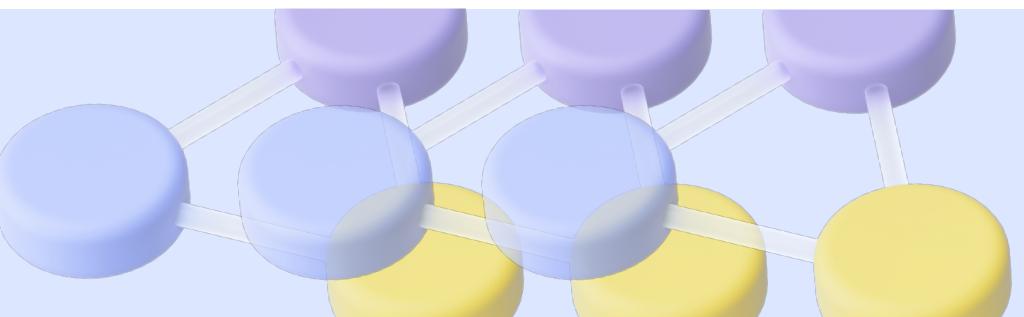
- 



# Управление таймаутами

GO

- Используются контексты из стандартной библиотеки



# Управление таймаутами

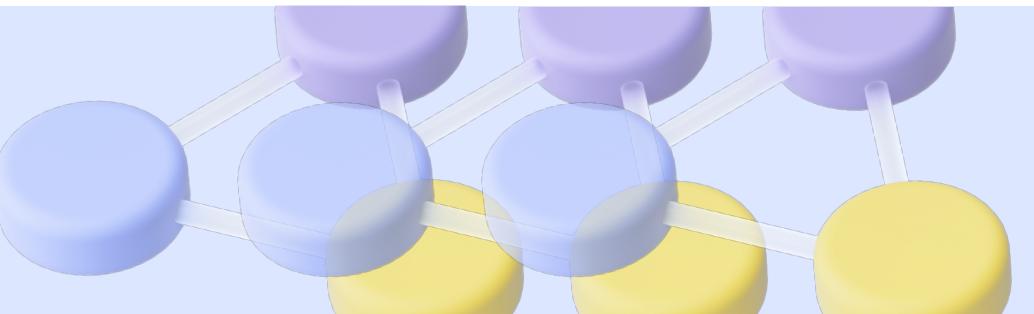
## GO

- Используются контексты из стандартной библиотеки
- **Выполнение кода может отменяться не только по таймауту - контексты можно отменять в любое время.**

```
ctxTimeout, cancel := context.WithTimeout(  
    ctx, time.Second)
```

```
res, err := tx.Execute(  
    ctxTimeout, `SELECT 1`,  
    nil)
```

```
if err != nil {  
    return err  
}  
cancel()
```



# Управление таймаутами

## GO

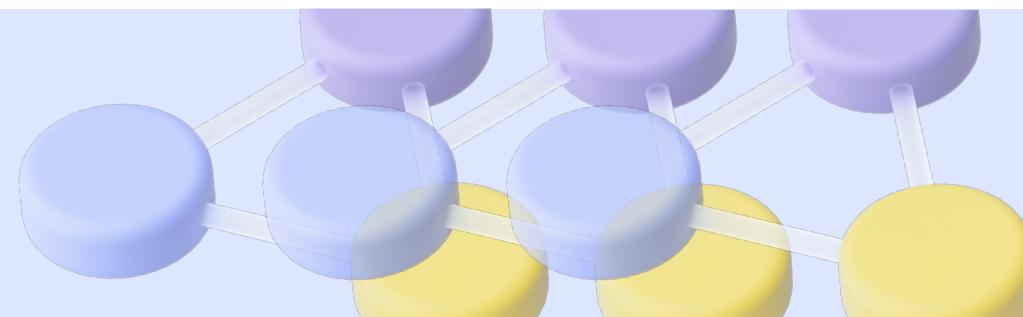
- Используются контексты из стандартной библиотеки
- Выполнение кода может отменяться не только по таймауту - контексты можно отменять в любое время.

```
ctxTimeout, cancel := context.WithTimeout(  
    ctx, time.Second)  
  
res, err := tx.Execute(  
    ctxTimeout, `SELECT 1`,  
    nil)  
  
if err != nil {  
    return err  
}  
cancel()
```

## Rust

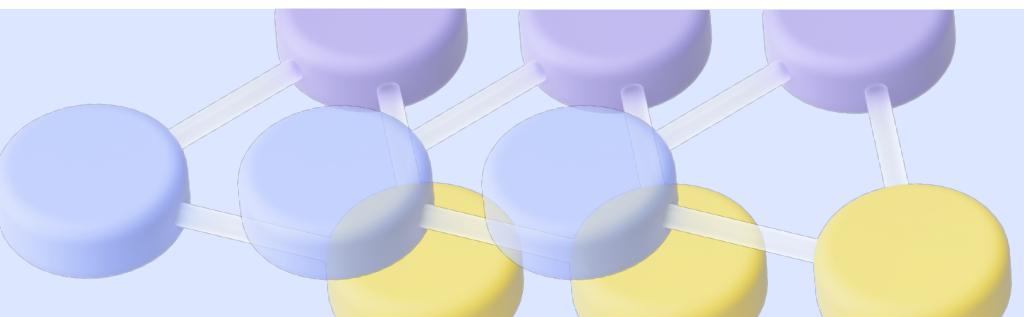
- **Функция `timeout` из `tokio`**

```
let res =  
    tokio::time::timeout(  
        Duration::from_secs(1),  
        tx.query(Query::new("SELECT 1"))  
    ).await??;
```



# Передача параметров запроса

GO

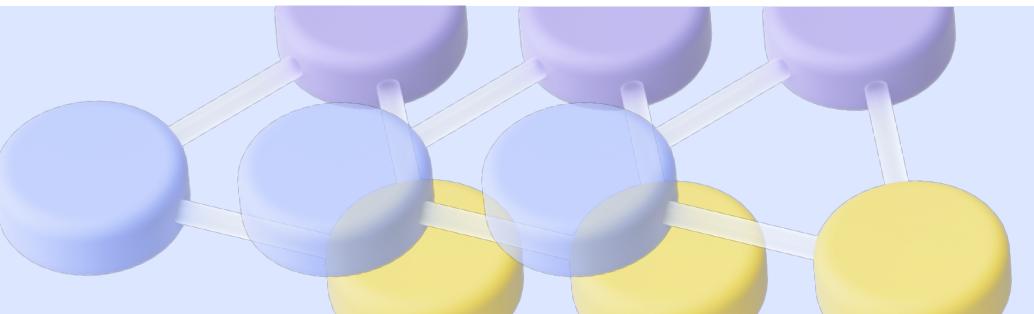


# Передача параметров запроса

## GO

- Со стороны пользователя

```
_ , err := tx.Execute(ctx, `  
DECLARE $val AS Int32;  
SELECT $val AS result`,  
  
table.NewQueryParameters(table.ValueParam("$val",  
types.Int32Value(1))),  
)  
if err != nil {  
    return err  
}
```



# Передача параметров запроса

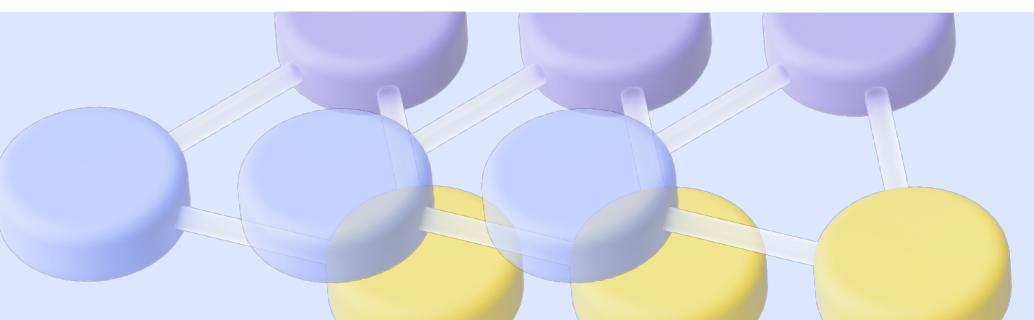
## GO

- Со стороны пользователя

```
_ , err := tx.Execute(ctx, `  
DECLARE $val AS Int32;  
SELECT $val AS result`,  
    table.NewQueryParameters(table.ValueParam("$val",  
types.Int32Value(1))),  
)  
if err != nil {  
    return err  
}
```

- Внутри

```
types/func Int32Value(v int32) Value { return value.Int32Value(v) }  
value/func Int32Value(v int32) int32Value {  
    return int32Value(v)  
}  
type int32Value int32
```



# Передача параметров запроса

## GO

- Со стороны пользователя

```
_ , err := tx.Execute(ctx, `  
DECLARE $val AS Int32;  
SELECT $val AS result`,  
    table.NewQueryParameters(table.ValueParam("$val",  
types.Int32Value(1))),  
)  
if err != nil {  
    return err  
}
```

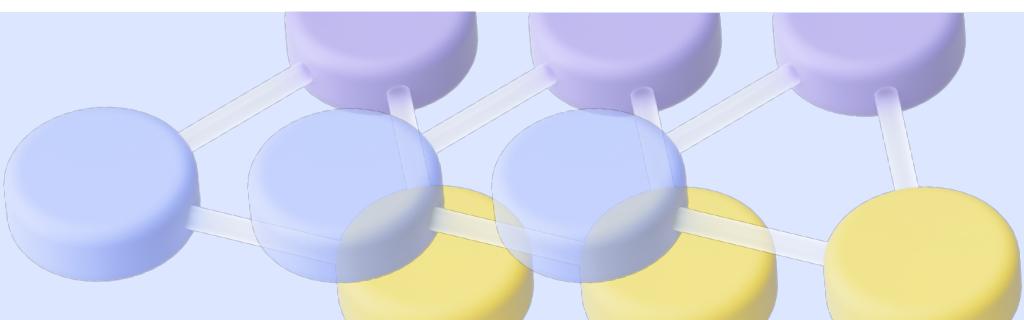
- Внутри

```
types/func Int32Value(v int32) Value { return value.Int32Value(v) }  
value/func Int32Value(v int32) int32Value {  
    return int32Value(v)  
}  
type int32Value int32
```

## Rust

- Со стороны пользователя

```
tx.query(Query::new(  
    "DECLARE $val AS Int32;  
    SELECT $val AS result  
").with_params(ydb_params!({  
    "$val" => 1,  
}));
```



# Передача параметров запроса

## GO

- Со стороны пользователя

```
_ , err := tx.Execute(ctx, `  
DECLARE $val AS Int32;  
SELECT $val AS result`,  
    table.NewQueryParameters(table.ValueParam("$val",  
types.Int32Value(1))),  
)  
if err != nil {  
    return err  
}
```

- Внутри

```
types/func Int32Value(v int32) Value { return value.Int32Value(v) }  
value/func Int32Value(v int32) int32Value {  
    return int32Value(v)  
}  
type int32Value int32
```

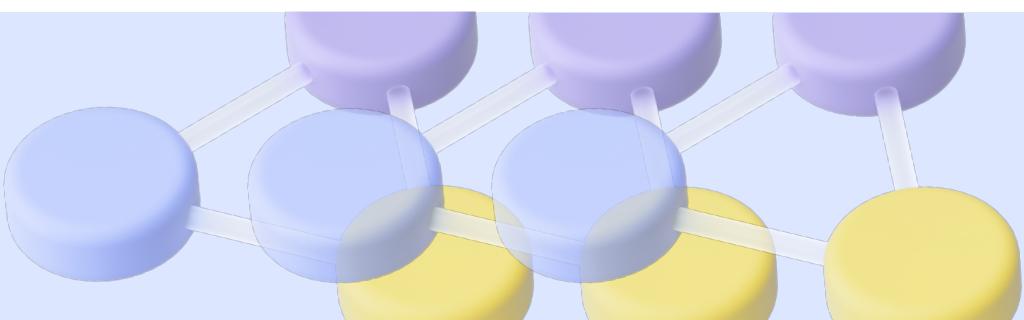
## Rust

- Со стороны пользователя

```
tx.query(Query::new(`  
DECLARE $val AS Int32;  
SELECT $val AS result`  
`).with_params(ydb_params!  
    "$val" => 1,  
)));
```

- Внутри

```
macro_rules! ydb_params {  
    (  
        $($name:expr => $val:expr ),+ $(,)?  
    ) => {  
        std::collections::HashMap::from_iter([  
            $($name.into(), $val.into()),+  
        ])  
    };  
}
```

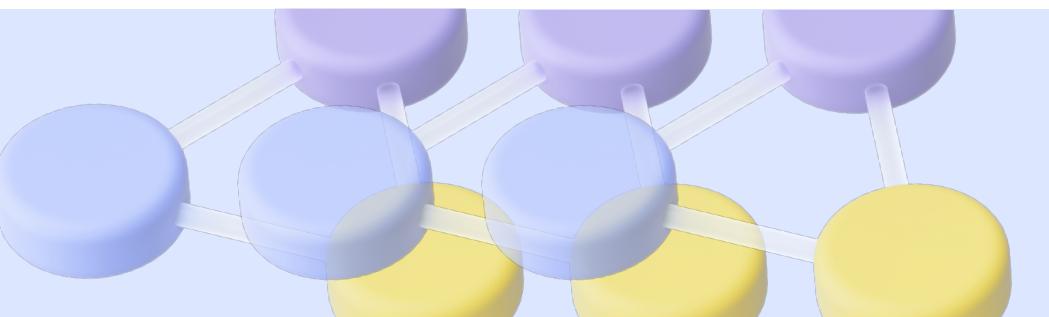


# Чтение результатов в переменные

GO

- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}
```



# Чтение результатов в переменные

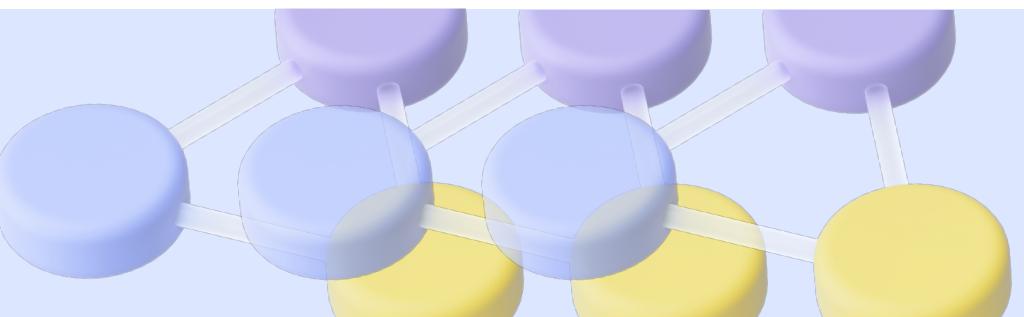
## GO

- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}
```

- Внутри

```
func (s *scanner) scanOptional(v interface{},
defaultValueForOptional bool) {
...
    switch v := v.(type) {
    case **int8:
        if s.isNull() {
            *v = nil
        } else {
            src := s.int8()
            *v = &src
        }
    ...
}
```



# Чтение результатов в переменные

## GO

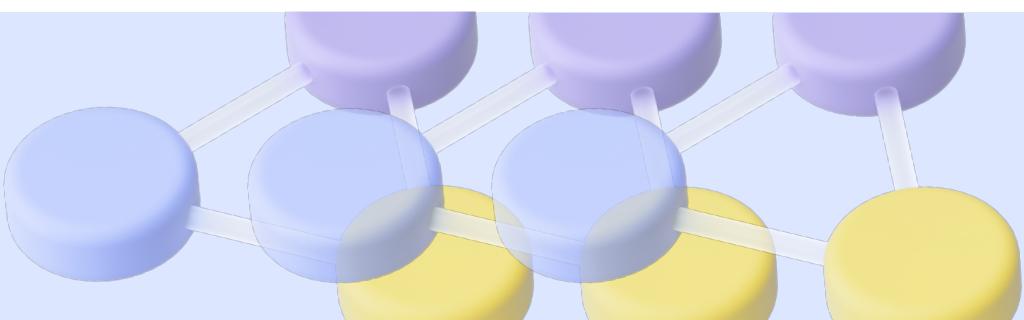
- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}
• Внутри
func (s *scanner) scanOptional(v interface{}, defaultValueForOptional bool) {
...
    switch v := v.(type) {
    case **bool:
        if s.isNull() {
            *v = nil
        } else {
            src := s.bool()
            *v = &src
        }
    case **int8:
        if s.isNull() {
            *v = nil
        } else {
            src := s.int8()
            *v = &src
        }
    }
```

## Rust

- Со стороны пользователя

```
let a: Option<i32> =
row.remove_field(0)?.try_into()?;
let b: Option<String> =
row.remove_field(1)?.try_into?();
```



# Чтение результатов в переменные

## GO

- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}

• Внутри
func (s *scanner) scanOptional(v interface{}, defaultValueForOptional bool) {
...
    switch v := v.(type) {
    case **bool:
        if s.isNull() {
            *v = nil
        } else {
            src := s.bool()
            *v = &src
        }
    case **int8:
        if s.isNull() {
            *v = nil
        } else {
            src := s.int8()
            *v = &src
        }
    }
```

## Rust

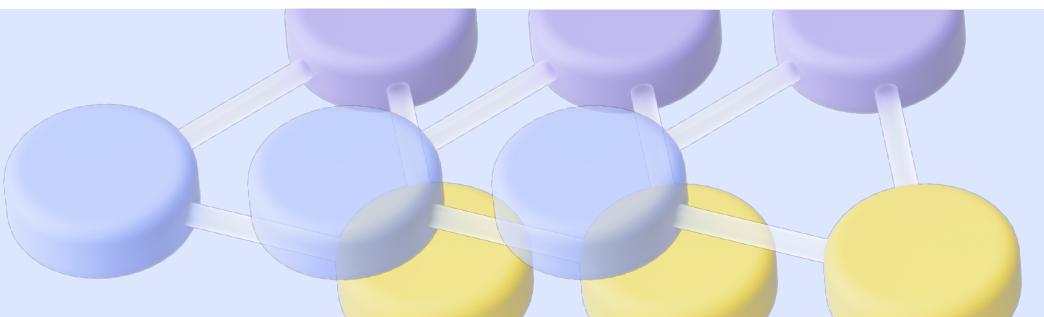
- Со стороны пользователя

```
let a: Option<i32> =
row.remove_field(0)?.try_into()?;
let b: Option<String> =
row.remove_field(1)?.try_into?();
```

- Внутри

```
macro_rules! simple_convert {...};

simple_convert!(i8, Value::Int8);
simple_convert!(i16, Value::Int16, Value::Int8,
Value::UInt8);
simple_convert!(u16, Value::UInt16, Value::UInt8);
```



# Чтение результатов в переменные

## GO

- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}

• Внутри
func (s *scanner) scanOptional(v interface{}, defaultValueForOptional bool) {
...
    switch v := v.(type) {
    case **bool:
        if s.isNull() {
            *v = nil
        } else {
            src := s.bool()
            *v = &src
        }
    case **int8:
        if s.isNull() {
            *v = nil
        } else {
            src := s.int8()
            *v = &src
        }
    }
}
```

**2 варианта  
на 14 строках**

## Rust

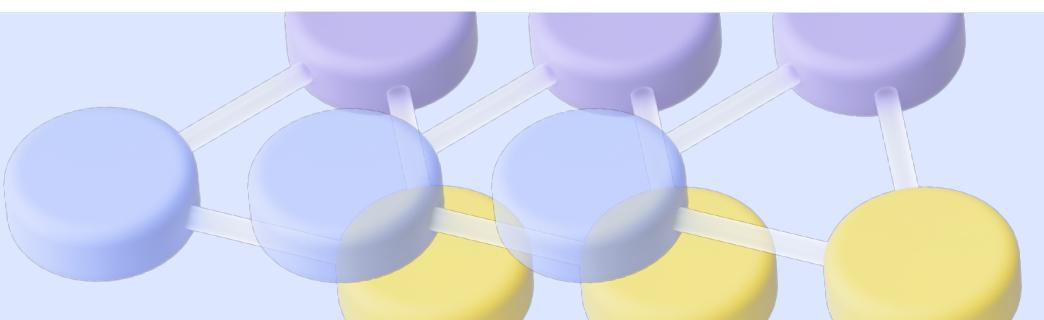
- Со стороны пользователя

```
let a: Option<i32> =
row.remove_field(0)?.try_into()?;
let b: Option<String> =
row.remove_field(1)?.try_into?();
```

- Внутри

```
macro_rules! simple_convert {...};

simple_convert!(i8, Value::Int8);
simple_convert!(i16, Value::Int16, Value::Int8,
Value::UInt8);
simple_convert!(u16, Value::UInt16, Value::UInt8);
```



# Чтение результатов в переменные

## GO

- Со стороны пользователя

```
var a *int32
var b *string
err := res.Scan(&a, &b)
if err != nil {
    return err
}

func (s *scanner) scanOptional(v interface{}, defaultValueForOptional bool) {
...
    switch v := v.(type) {
    case **bool:
        if s.isNull() {
            *v = nil
        } else {
            src := s.bool()
            *v = &src
        }
    case **int8:
        if s.isNull() {
            *v = nil
        } else {
            src := s.int8()
            *v = &src
        }
    }
}
```

2 варианта  
на 14 строках

## Rust

- Со стороны пользователя

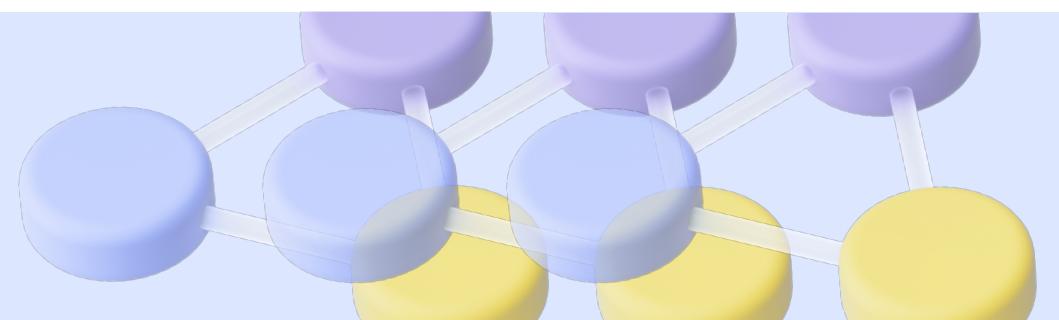
```
let a: Option<i32> =
row.remove_field(0)?.try_into()?;
let b: Option<String> =
row.remove_field(1)?.try_into?();
```

- Внутри

```
macro_rules! simple_convert {...};

simple_convert!(i8, Value::Int8);
simple_convert!(i16, Value::Int16, Value::Int8,
Value::Uint8);
simple_convert!(u16, Value::Uint16, Value::Uint8);
```

6 вариантов  
на 3 строках



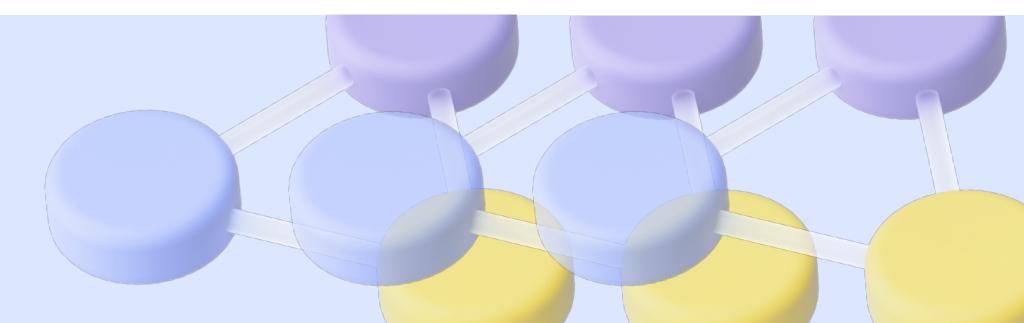
# Конвертация Value, макрос simple\_convert в Rust

```
($native_type:ty, $ydb_value_kind_first:path $($, $ydb_value_kind:path)* $($, )?) => {
    impl From<$native_type> for Value {
        fn from(value: $native_type) -> Self {
            $ydb_value_kind_first(value)
        }
    }

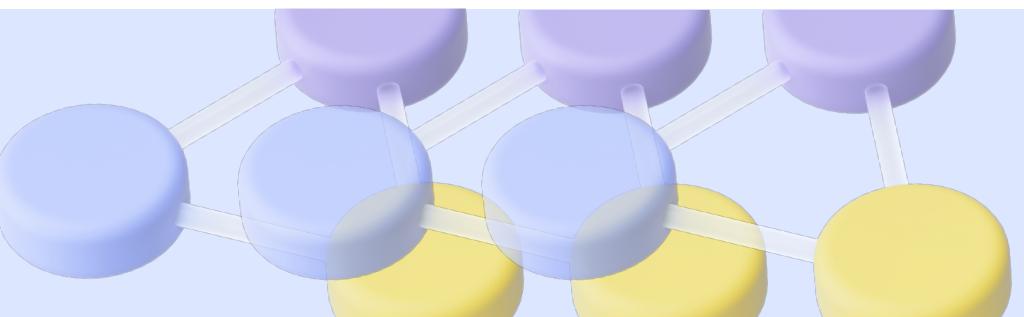
    impl TryFrom<Value> for $native_type {
        type Error = YdbError;

        fn try_from(value: Value) -> Result<Self, Self::Error> {
            match value {
                $ydb_value_kind_first(val) => Ok(val.into()),
                $($ydb_value_kind(val)) => Ok(val.into()), // ← для перечисленных типов
                value => Err(YdbError::Convert(format!(
                    "failed to convert from {} to {}",
                    value.kind_static(),
                    type_name::<Self>(),
                )));
            }
        }
    }
}

...
}
```



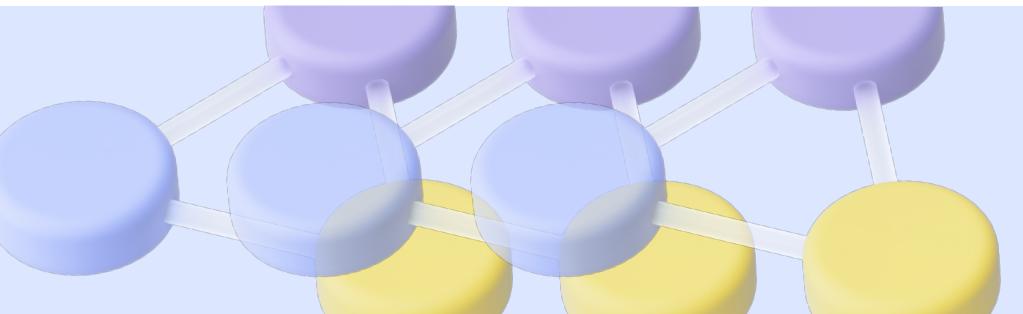
# Возврат сессий в пул



# Возврат сессий в пул

GO

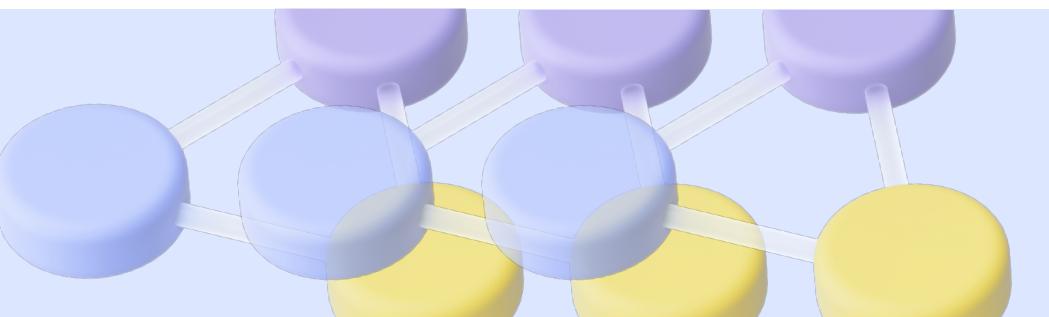
- Нет деструкторов, финализаторы хрупкие и с задержками.



# Возврат сессий в пул

## GO

- Нет деструкторов, финализаторы хрупкие и с задержками.
- **Более-менее надёжный способ - только callback для работы с сессией**



# Возврат сессий в пул

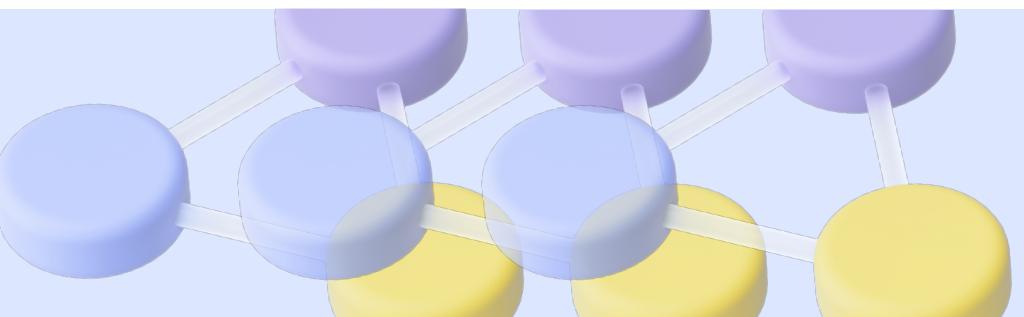
## GO

- Нет деструкторов, финализаторы хрупкие и с задержками.
- Более-менее надёжный способ - только callback для работы с сессией

```
s, err = p.Get(ctx)
if err != nil {
    return xerrors.WithStackTrace(err)
}

defer func() {
    _ = p.Put(ctx, s)
}()

err = op(ctx, s)
if err != nil {
    s.checkError(err)
    return xerrors.WithStackTrace(err)
}
```



# Возврат сессий в пул

## GO

- Нет деструкторов, финализаторы хрупкие и с задержками.
- Более-менее надёжный способ - только callback для работы с сессией

```
s, err = p.Get(ctx)
if err != nil {
    return xerrors.WithStackTrace(err)
}

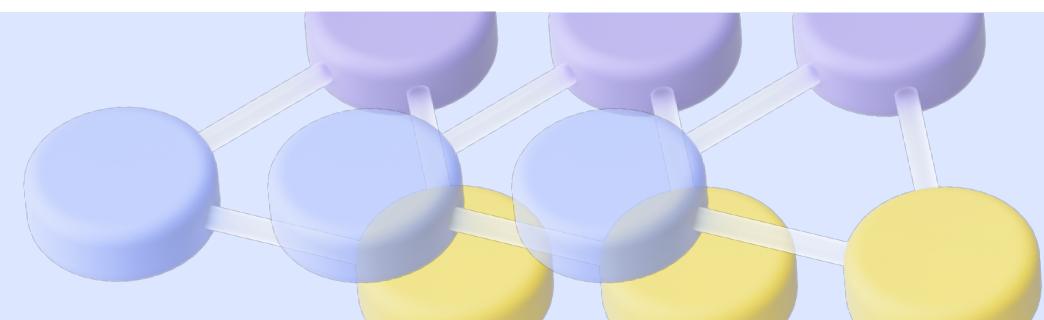
defer func() {
    _ = p.Put(ctx, s)
}()

err = op(ctx, s)
if err != nil {
    s.checkError(err)
    return xerrors.WithStackTrace(err)
}
```

## Rust

- **Есть Drop-trait, но только синхронный**

```
impl Drop for Session {
    fn drop(&mut self) {
        trace!("drop session: {}", &self.id);
        while let Some(on_drop) =
            self.on_drop_callbacks.pop() {
            on_drop(self)
        }
    }
}
```

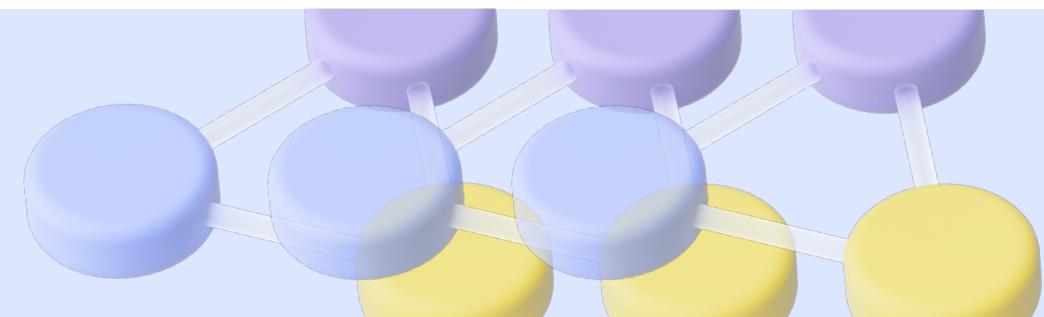


# Открыты для участия

Feature	C++	Python	Go	Java	NodeJS	C#	Rust
Поддержка SSL/TLS (системные сертификаты)	+	+	+	+	+	+	+
Поддержка SSL/TLS (кастомные сертификаты)	+	+	+	+	+	-	-
Возможность настроить/включить GRPC KeepAlive (фоновое поддержание живости соединения)	+	+	+	?	-	-	-
Регулярный прогон тестов SLO на последней версии	+	+/-	+	+	+/-	-	-



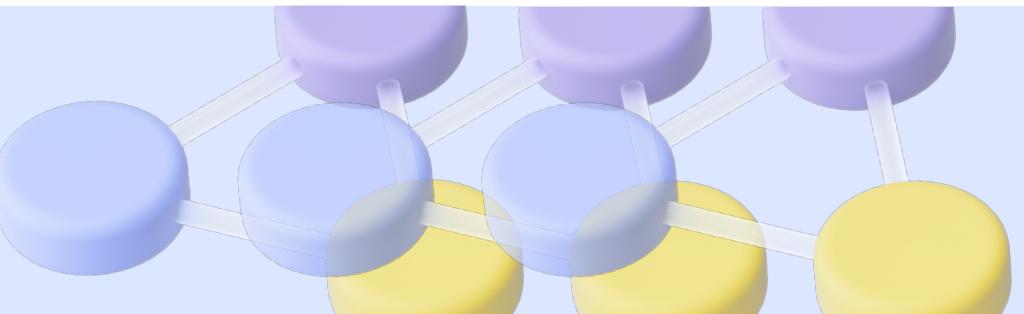
<https://ydb.tech/ru/docs/reference/ydb-sdk/feature-parity>



# Наблюдения

GO

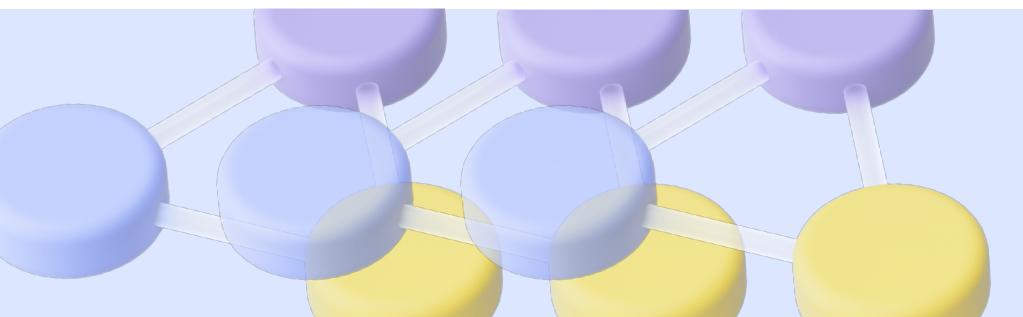
- Типизация доставляет меньше проблем



# Наблюдения

## GO

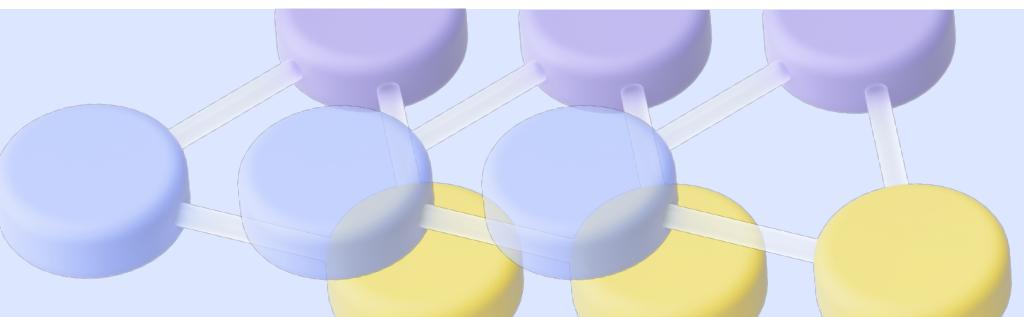
- Типизация доставляет меньше проблем
- **Reflection позволяет работать с неподконтрольными мне типами**



# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами



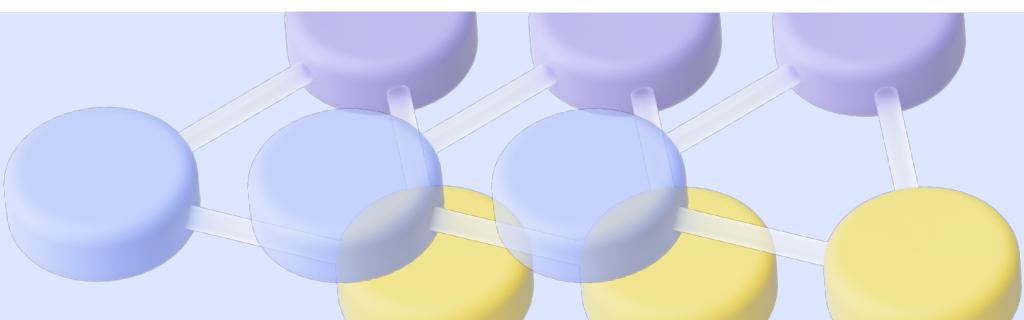
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

## Rust

- **Система типов позволяет делать их удобные преобразования**



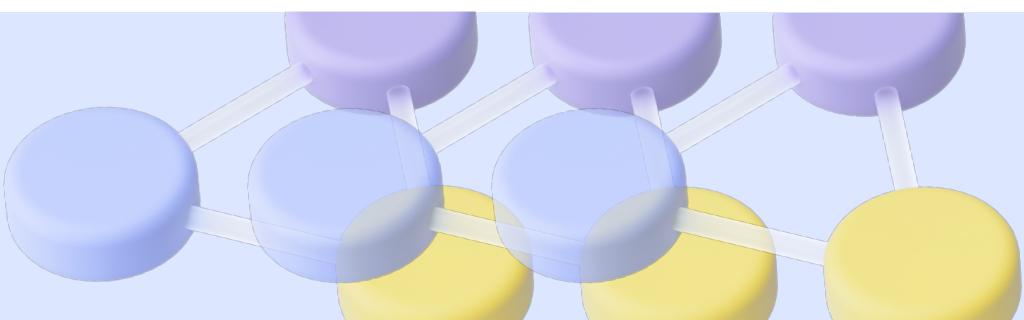
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

## Rust

- Система типов позволяет делать их удобные преобразования
- **По умолчанию принята статическая типизация и generic-и**



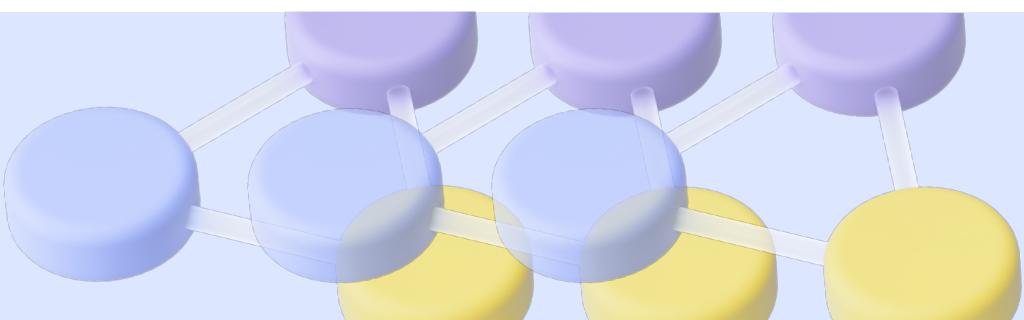
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

## Rust

- Система типов позволяет делать их удобные преобразования
- По умолчанию принята статическая типизация и generic-и
- **Многослойные Generic-и бывает сложно читать**



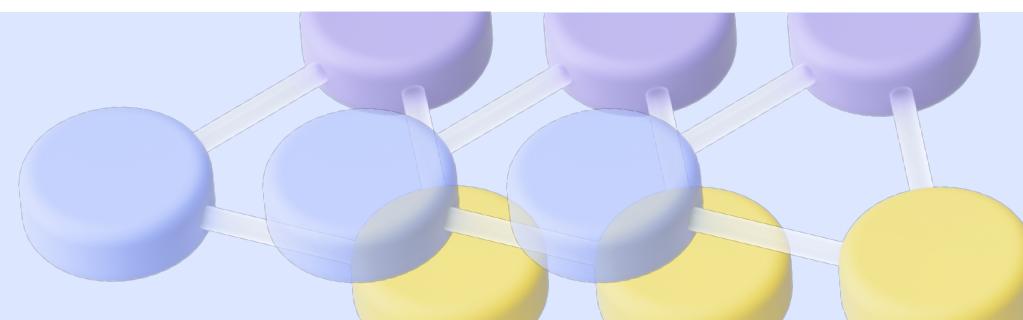
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

## Rust

- Система типов позволяет делать их удобные преобразования
- По умолчанию принята статическая типизация и generic-и
- Многослойные Generic-и бывает сложно читать
- **Макросы позволяют сократить лапшу и сделать удобные helper'ы**



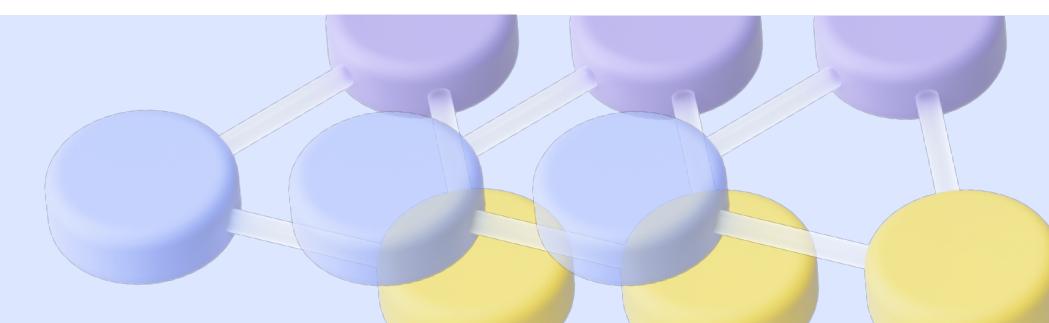
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

## Rust

- Система типов позволяет делать их удобные преобразования
- По умолчанию принята статическая типизация и generic-и
- Многослойные Generic-и бывает сложно читать
- Макросы позволяют сократить лапшу и сделать удобные helper'ы
- **async/await пролезают везде и писать асинхронный код неудобно**



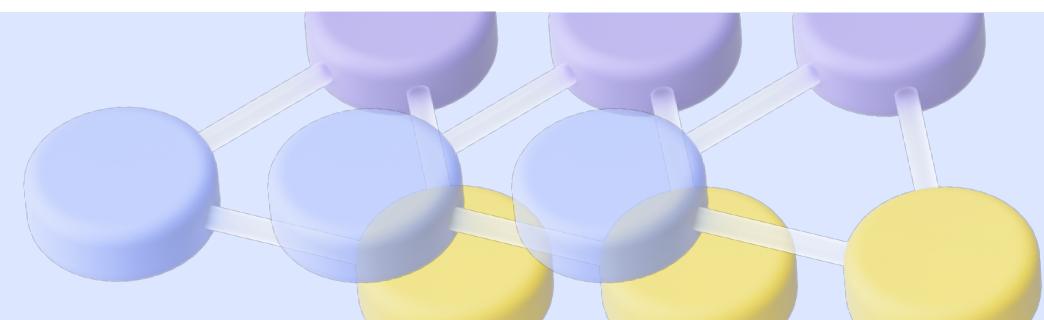
# Наблюдения

## GO

- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

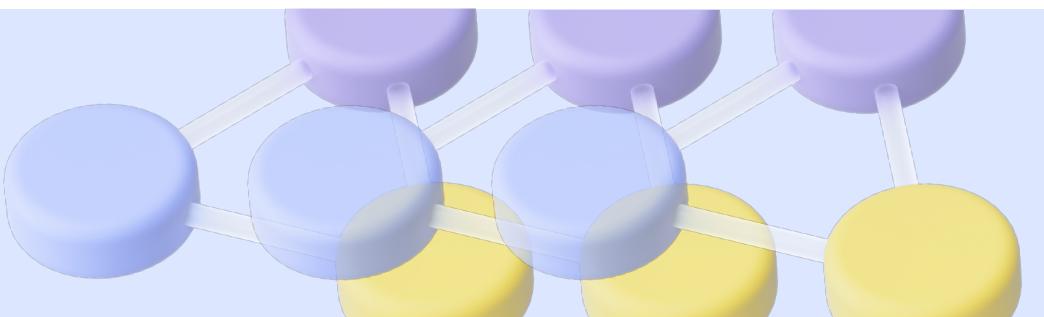
## Rust

- Система типов позволяет делать их удобные преобразования
- По умолчанию принята статическая типизация и generic-и
- Многослойные Generic-и бывает сложно читать
- Макросы позволяют сократить лапшу и сделать удобные helper'ы
- async/await пролезают везде и писать асинхронный код неудобно
- **Нестабильные версии зависимостей (бывают заметные изменения, требующие правки на своей стороне).**



# Выводы

- Не получится просто взять и переписать библиотеку 1 в 1 - слишком разные подходы, потребуется переосмысление
- В Rust заметно больше времени уходит на начальное написание кода
- Rust позволяет сделать интерфейс библиотеки более удобным для пользователя



# Буду рад продолжить общение



Тимофей Кулин  
Yandex, YDB, старший разработчик  
[rekby@yandex-team.ru](mailto:rekby@yandex-team.ru)  
<https://t.me/bobsmit>

[https://t.me/ydb\\_ru](https://t.me/ydb_ru)

<https://github.com/ydb-platform/ydb-rs-sdk>

<https://github.com/ydb-platform/ydb-go-sdk>

<https://github.com/rekby/rustcon-2022>

