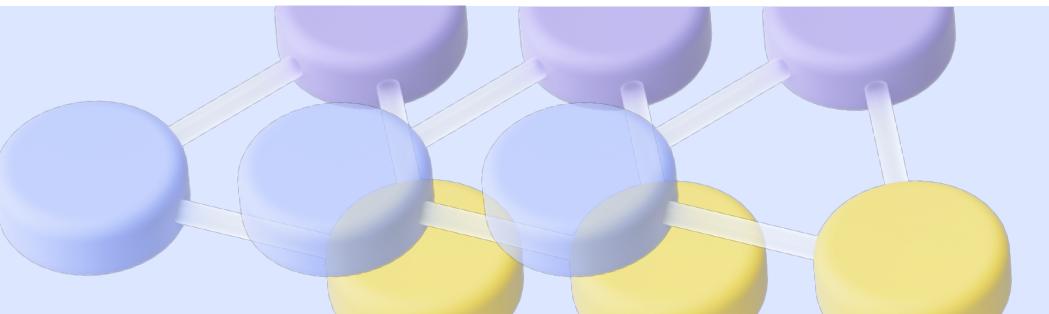




Как я писал клиента для YDB на Rust, сравнение с Go.

Тимофей Кулин,
YDB
старший разработчик

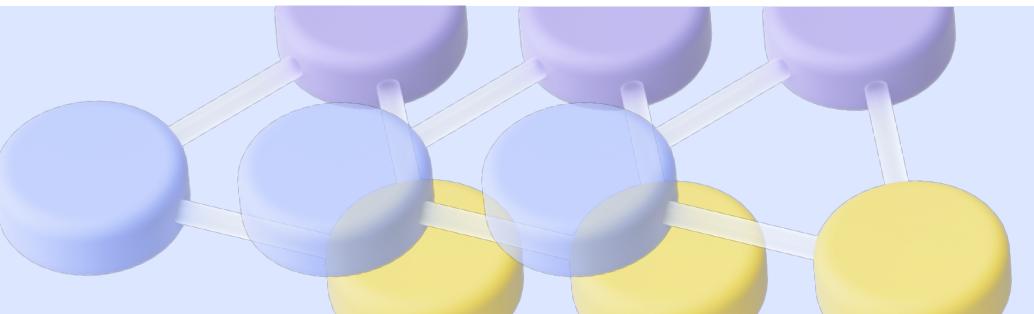
YDB — Open-Source Distributed SQL Database



YDB – Open-Source Distributed SQL Database

Распределённая

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



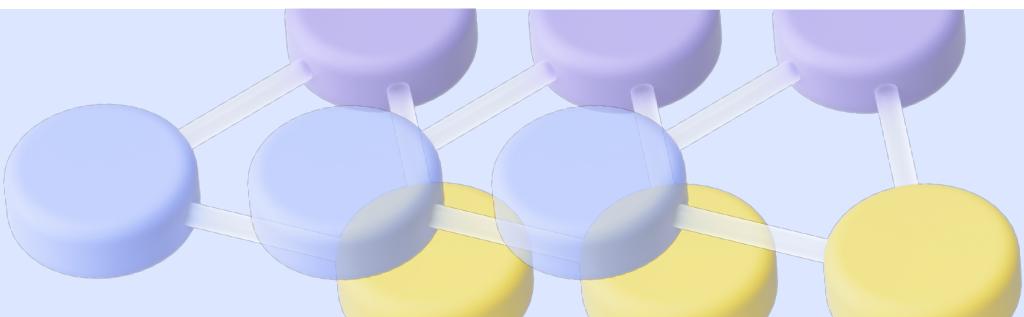
YDB – Open-Source Distributed SQL Database

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

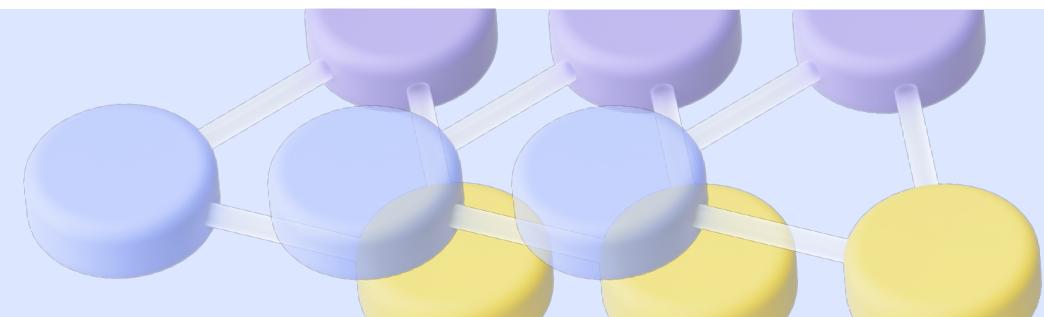
- Запускается в нескольких зонах доступности (AZ)
- Переживает пропажу одной зоны доступности + одной стойки без вмешательства человека

Для критичных задач

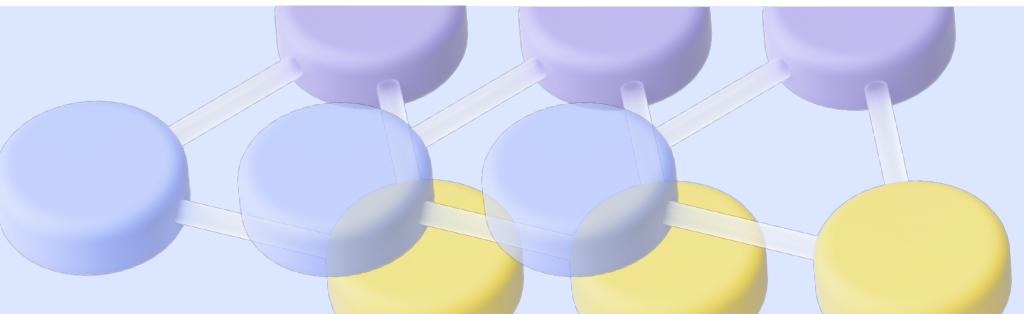
- Работа 24x7
- Обновления без простоев
- Строгая консистентность данных

Платформа

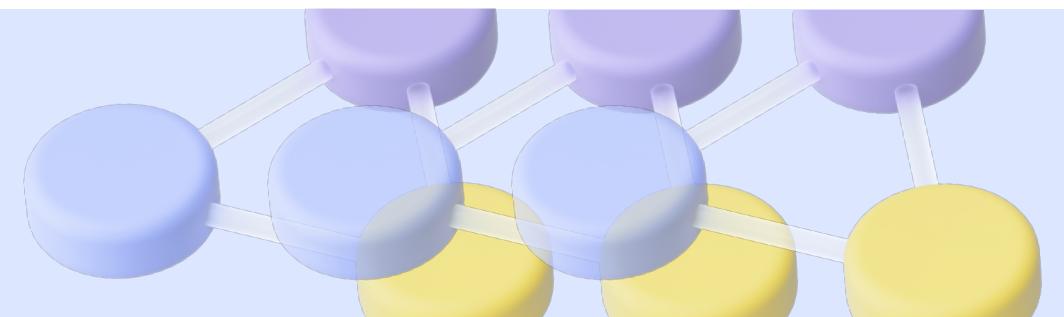
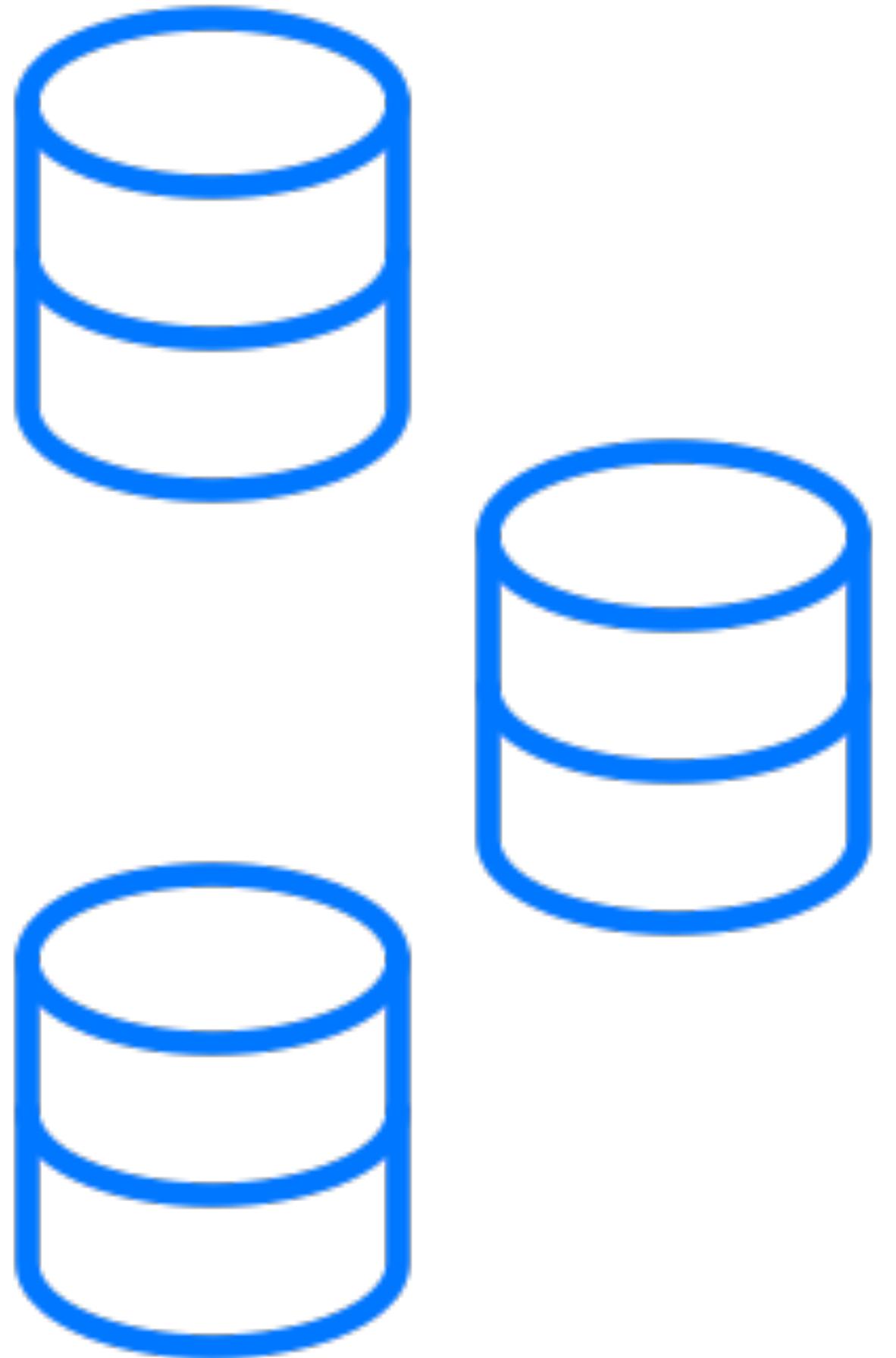
- Таблицы, топики, rate-limiter, ...



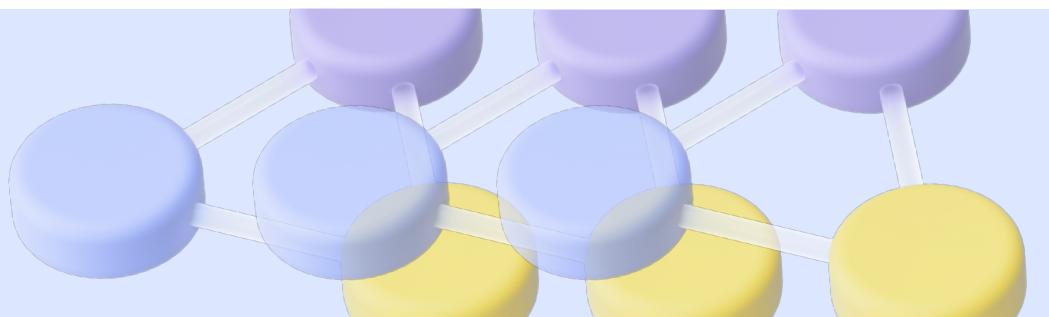
YDB — распределённая БД



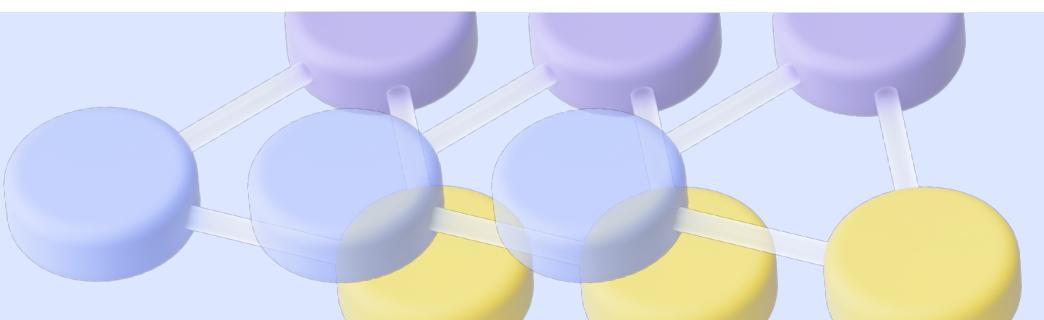
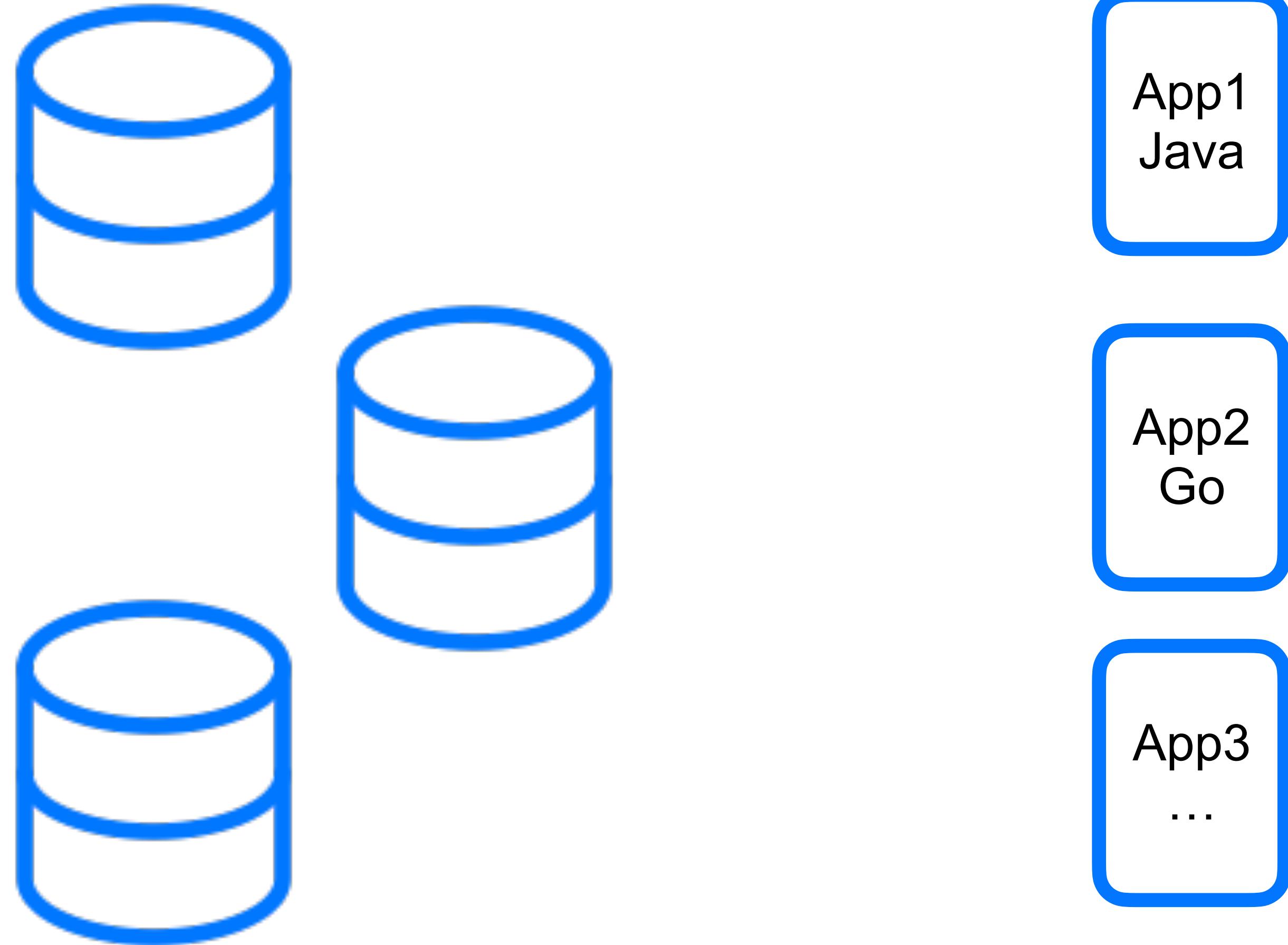
YDB – распределённая Бд



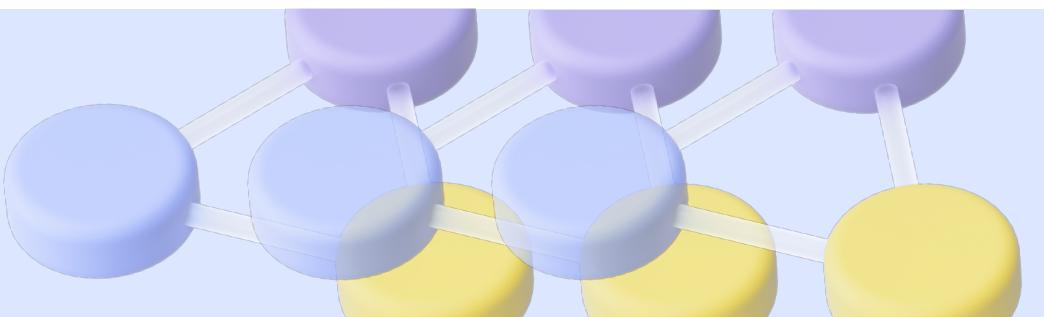
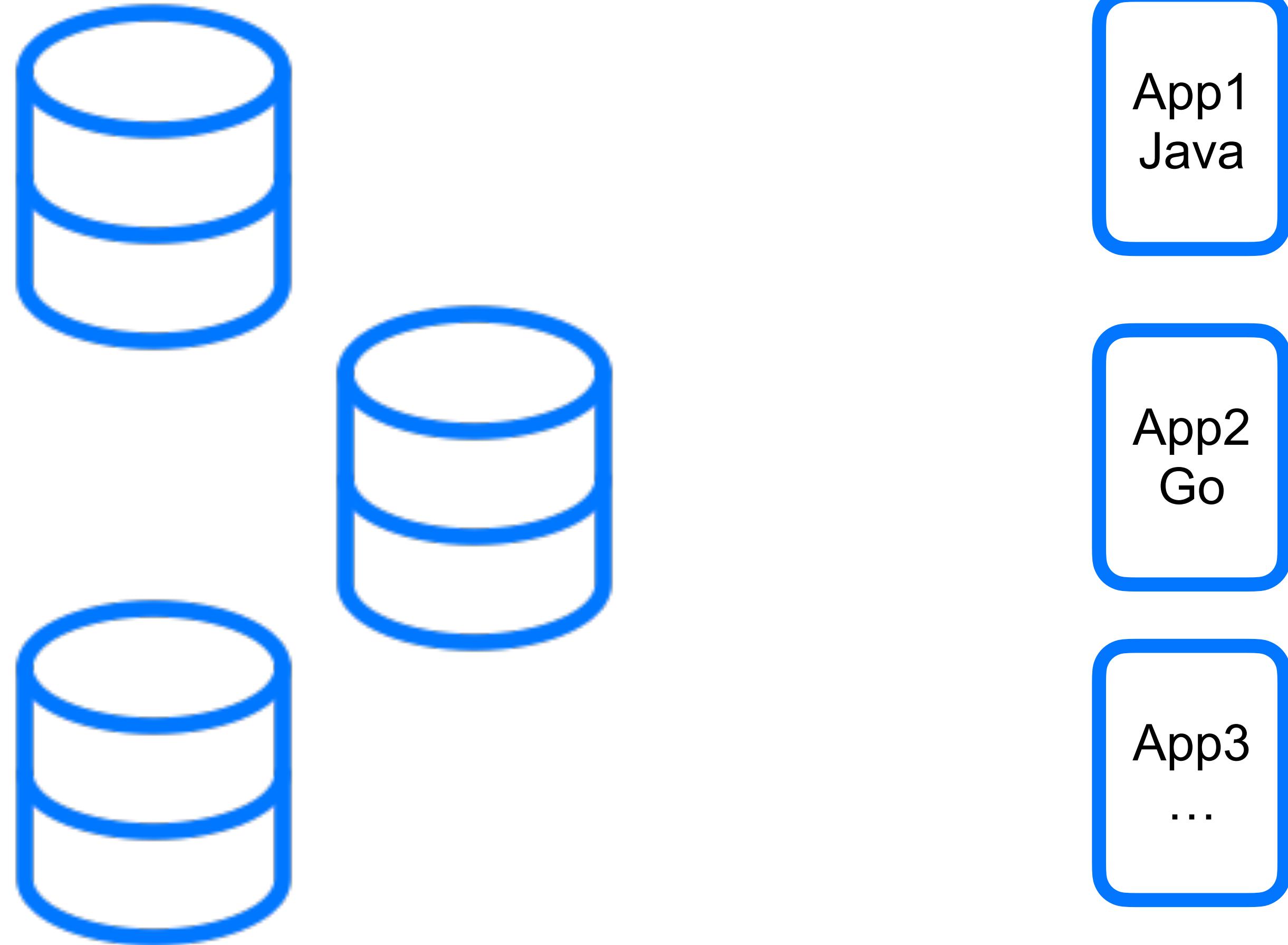
YDB – распределённая Бд



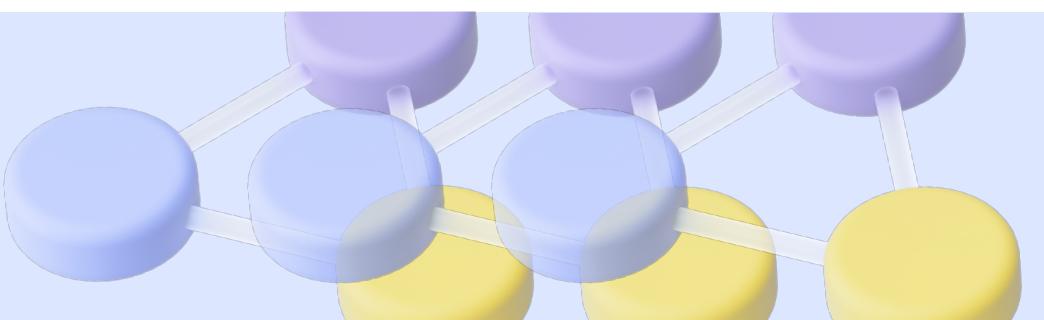
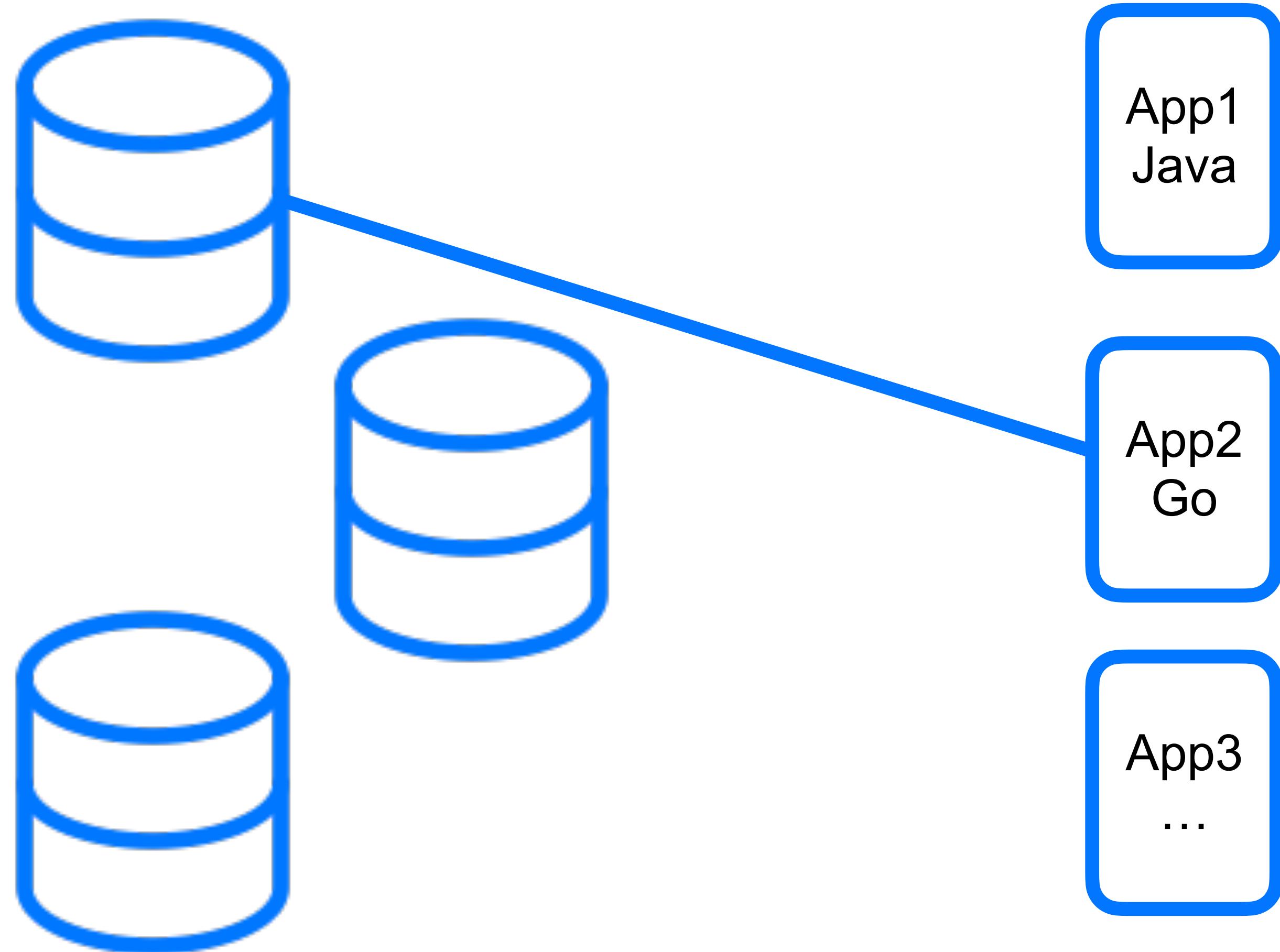
YDB – распределённая БД с клиентской балансировкой



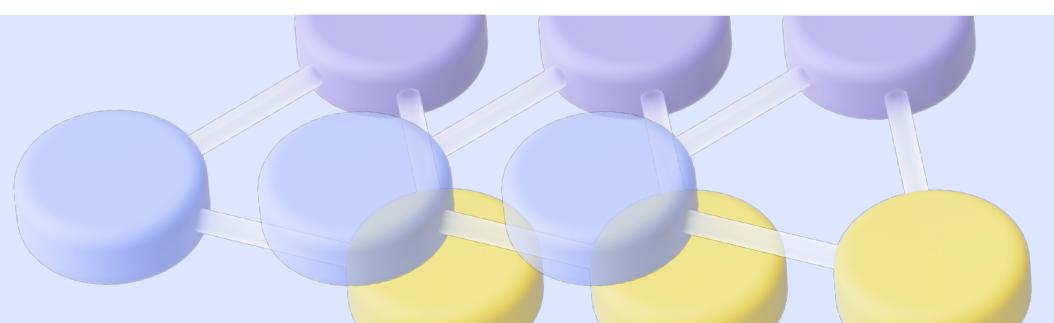
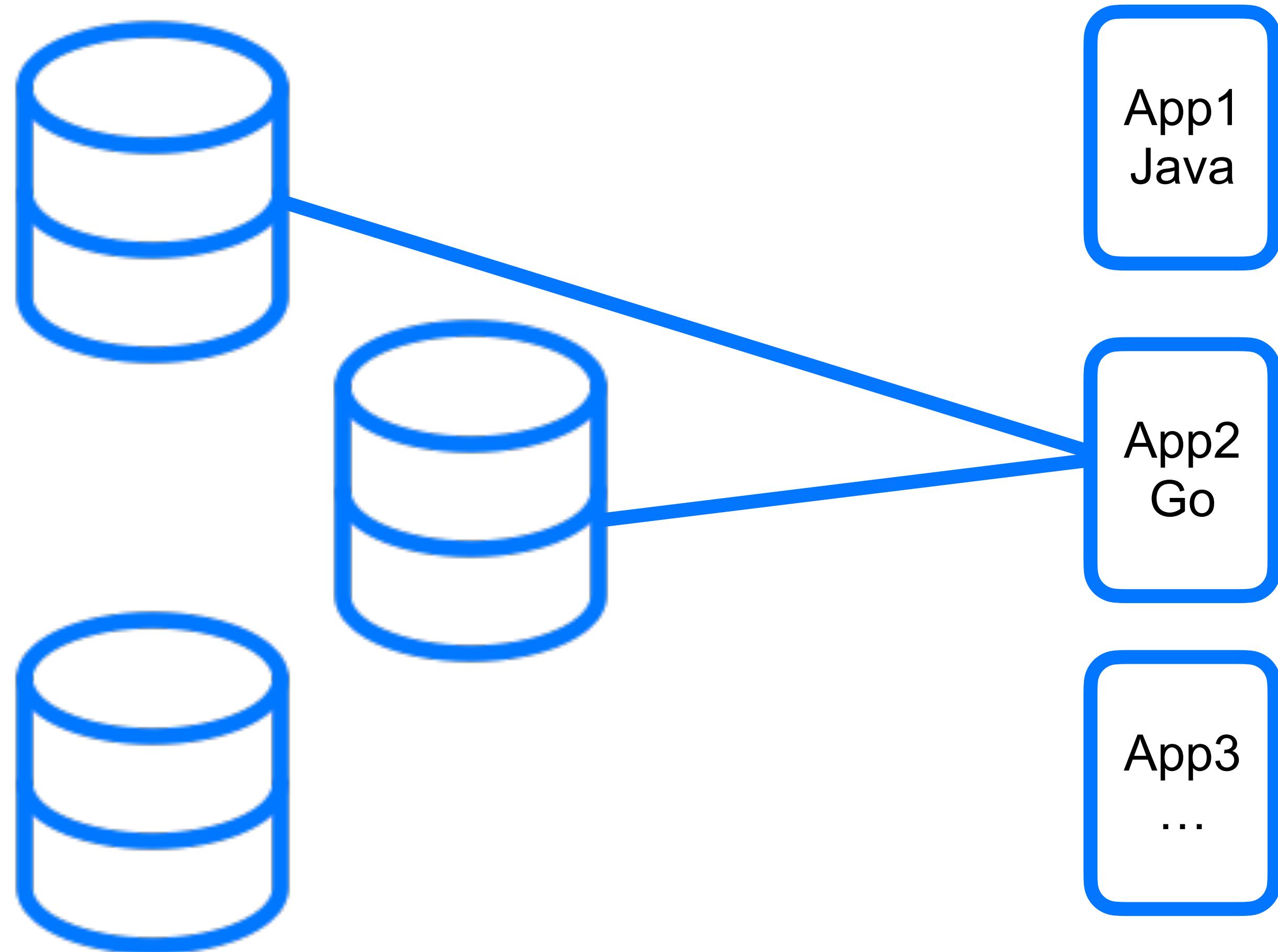
YDB – распределённая БД с клиентской балансировкой



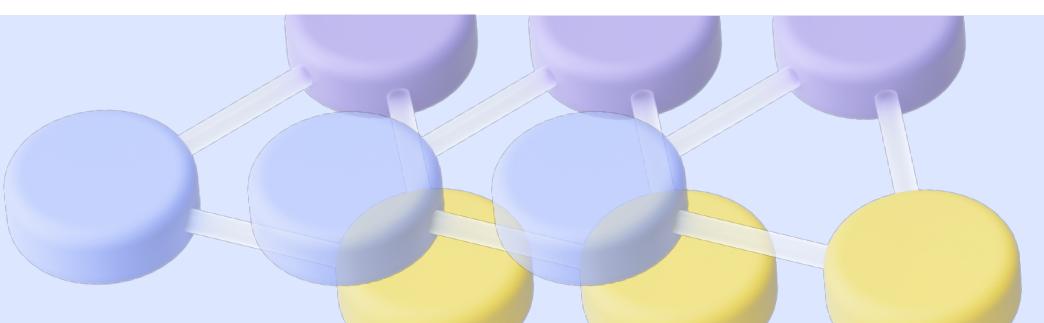
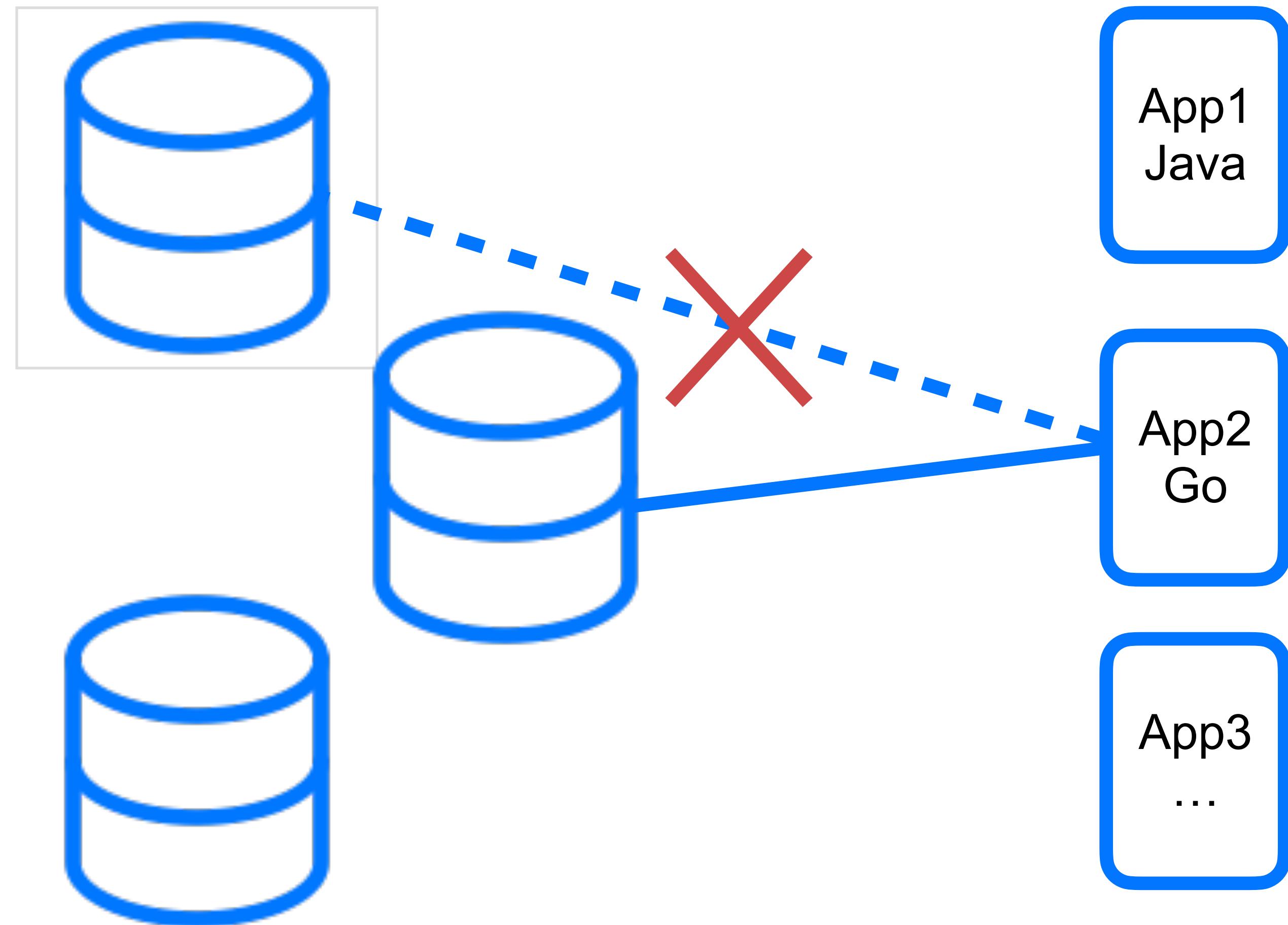
YDB – распределённая БД с клиентской балансировкой



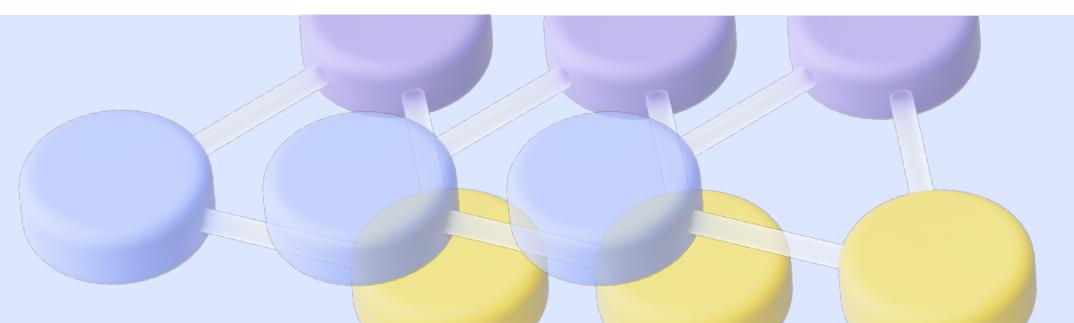
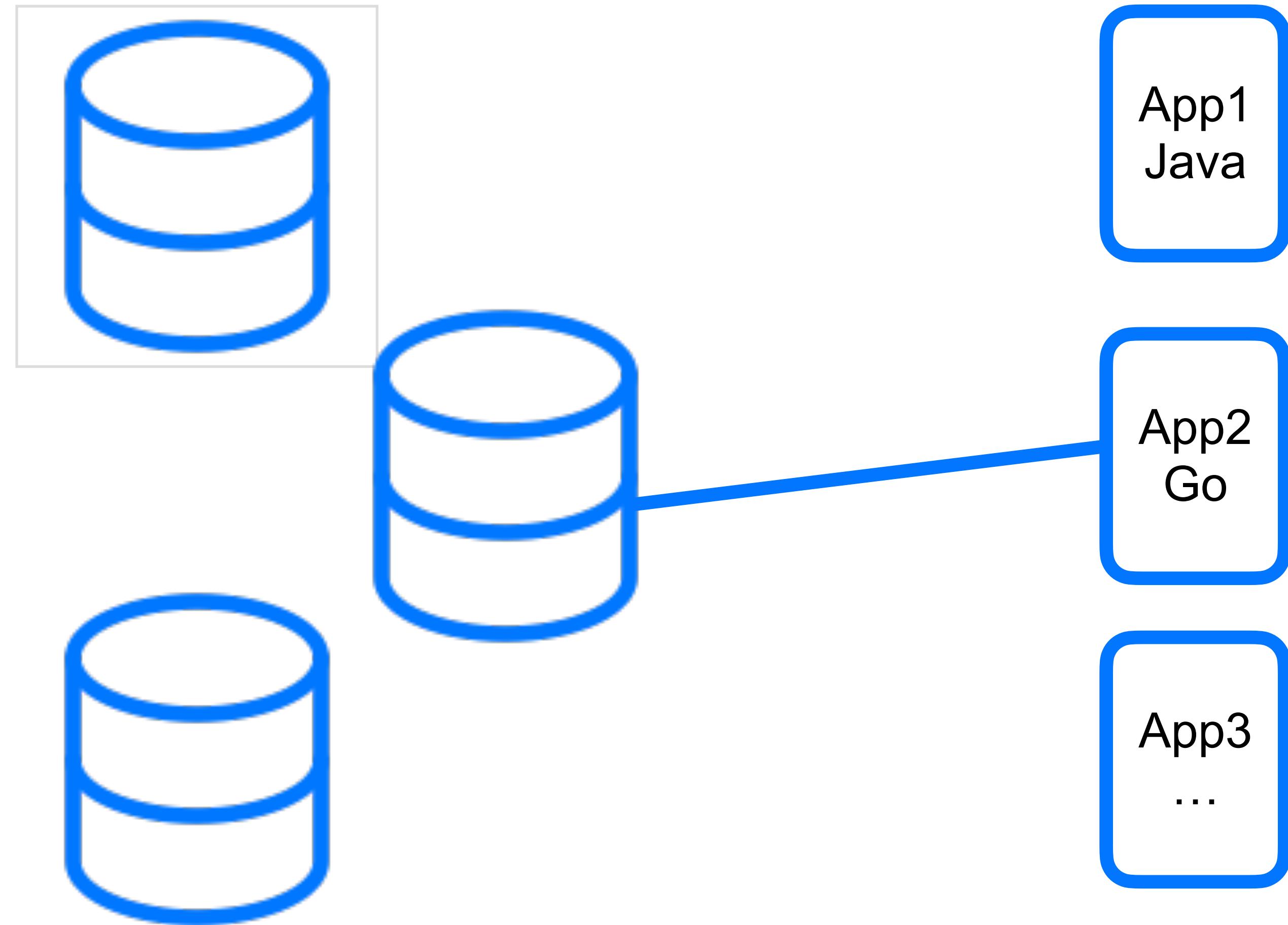
YDB – распределённая БД с клиентской балансировкой



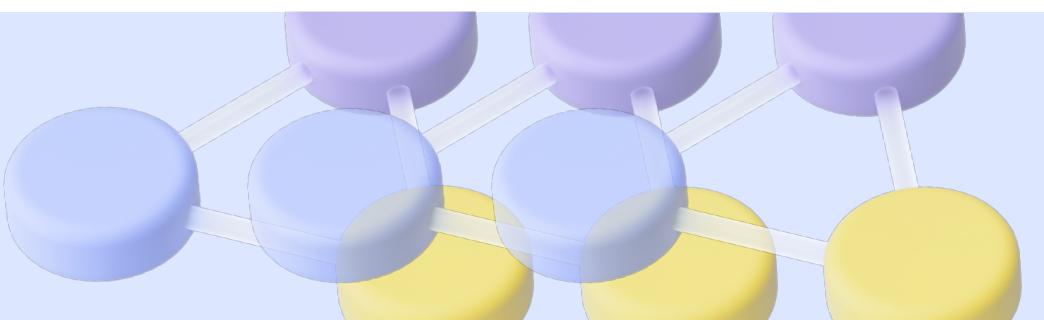
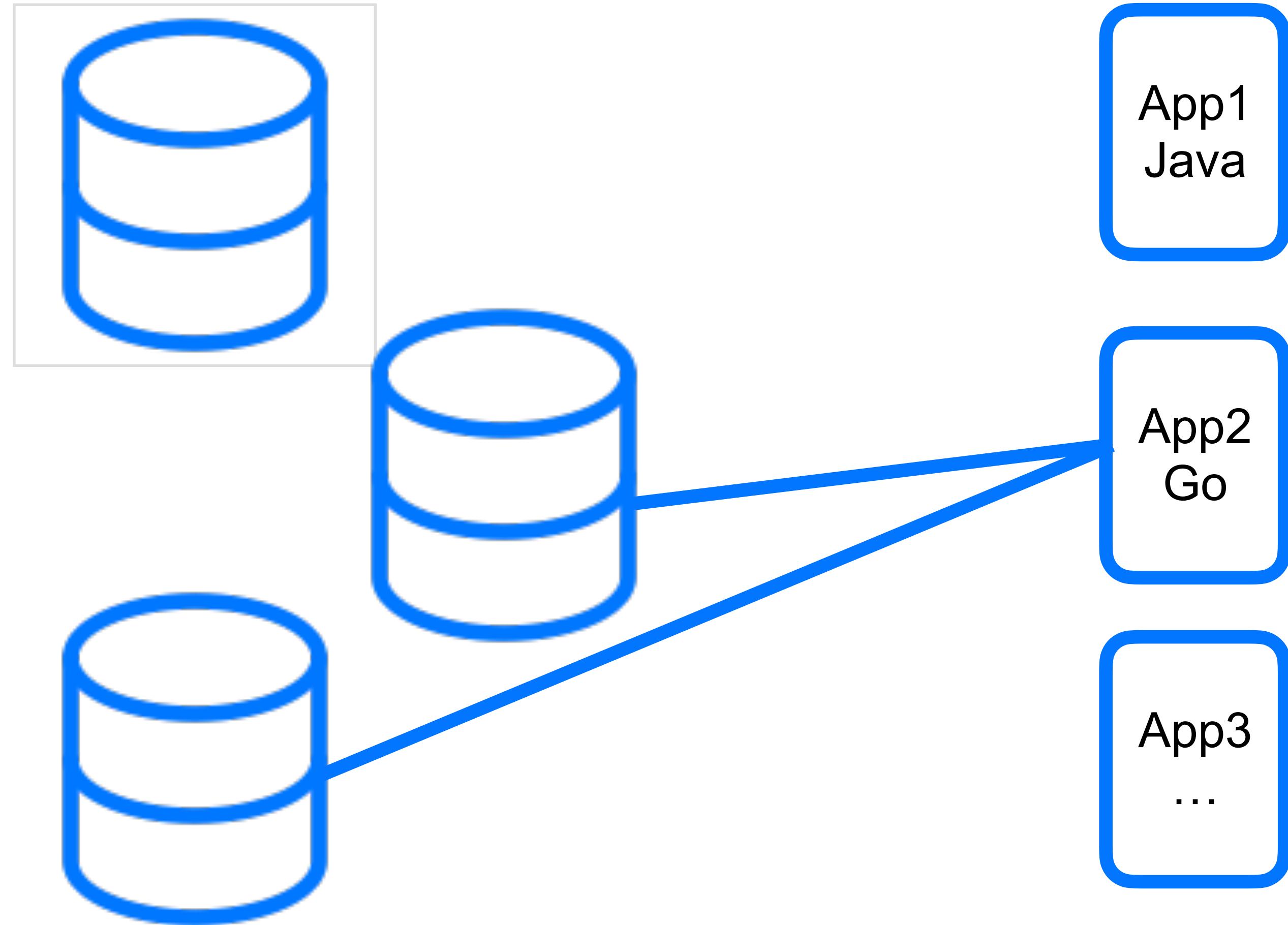
YDB – распределённая БД с клиентской балансировкой



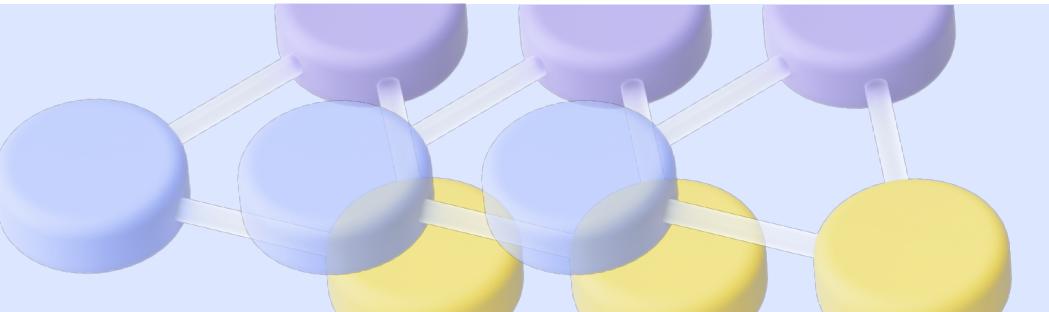
YDB – распределённая БД с клиентской балансировкой



YDB – распределённая БД с клиентской балансировкой



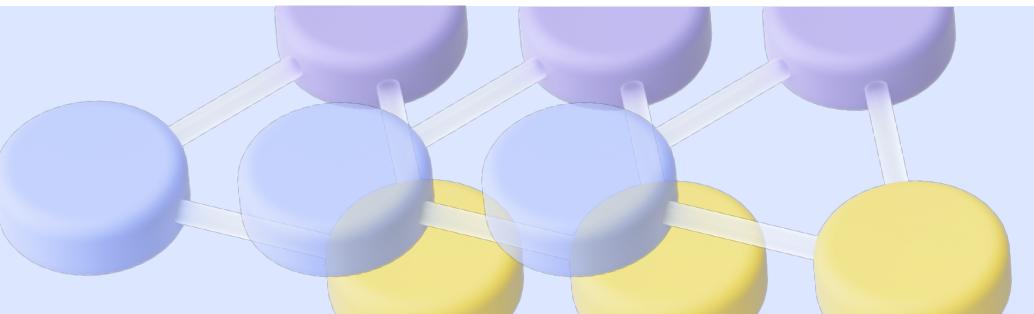
GRPC/protobuf



GRPC/protobuf

Protobuf

```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```



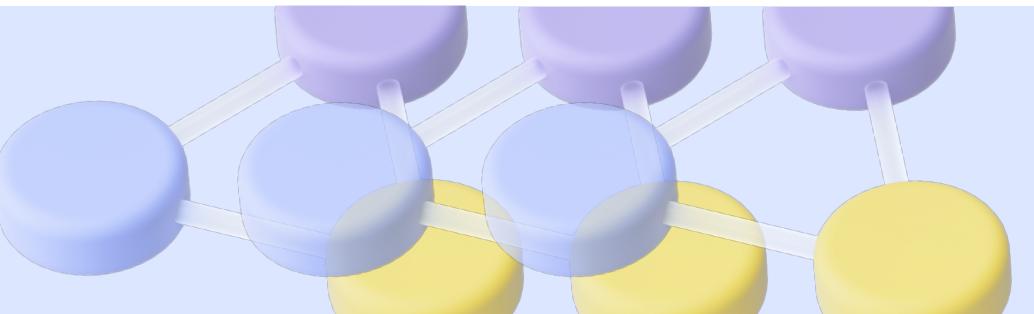
GRPC/protobuf

Protobuf

```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```

Языки

- C++



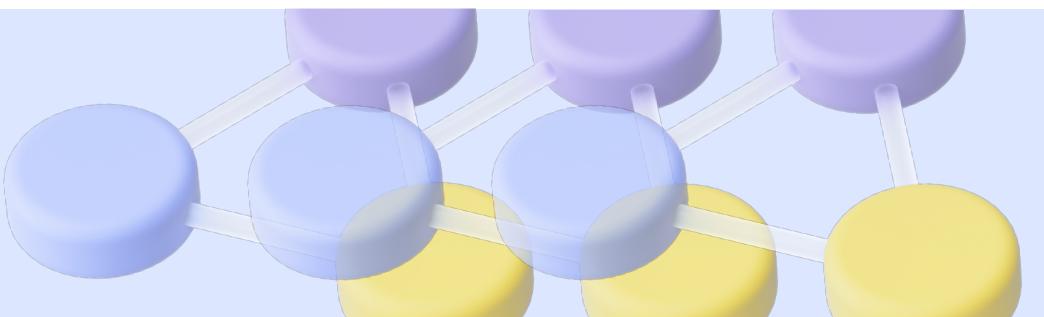
GRPC/protobuf

Protobuf

```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```

Языки

- C++
- Go
-



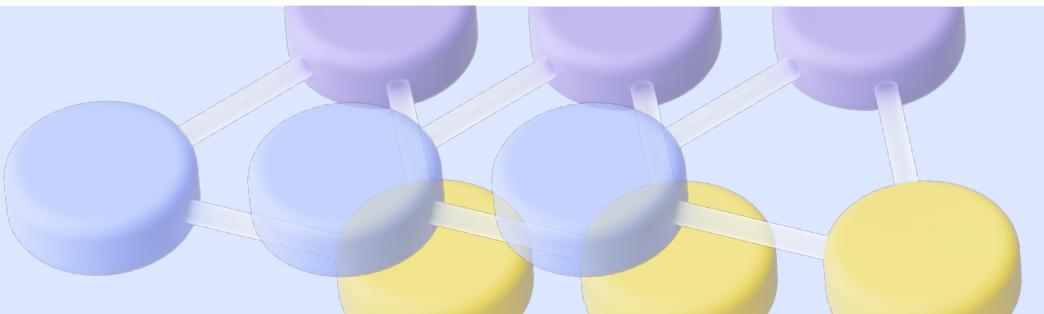
GRPC/protobuf

Protobuf

```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```

Языки

- C++
- Go
- **Java**



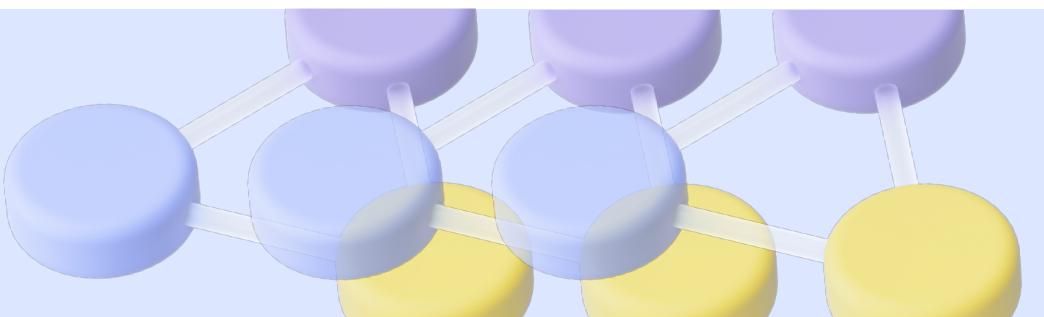
GRPC/protobuf

Protobuf

```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```

Языки

- C++
- Go
- Java
- ...



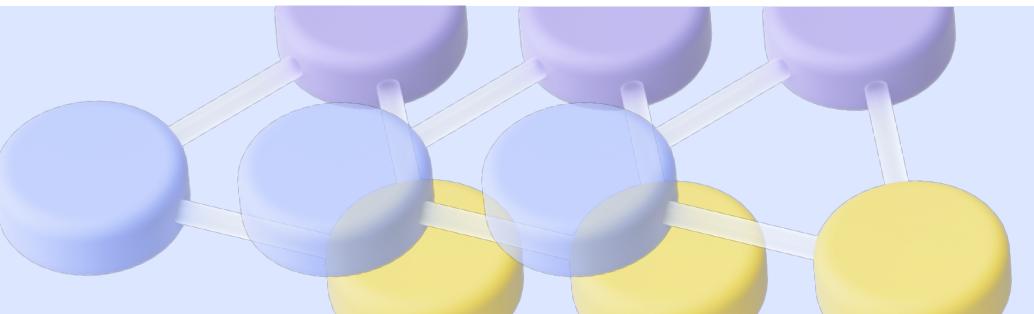
GRPC/protobuf

Protobuf

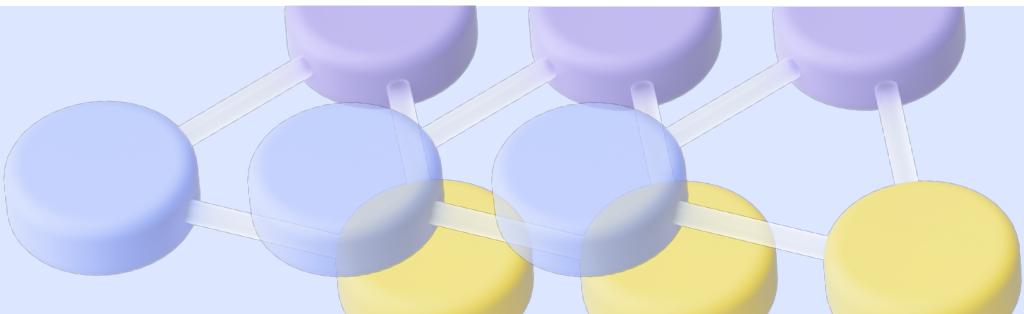
```
// Represents table-like structure with
// ordered set of rows and columns
message ResultSet {
    // Metadata of columns
    repeated Column columns = 1;
    // Rows of table
    repeated Value rows = 2;
    // Flag indicates the result was truncated
    bool truncated = 3;
}
```

Языки

- C++
- Go
- Java
- ... Rust



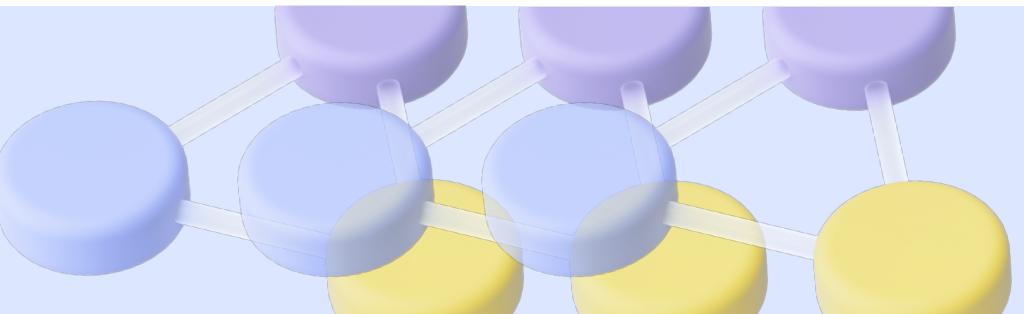
GRPC/protobuf, генерирование кода



GRPC/protobuf, генерирование кода

GO

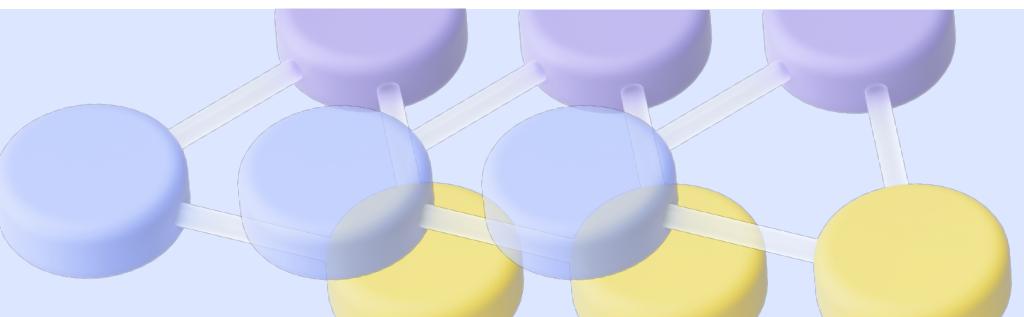
- protoc ...
-



GRPC/protobuf, генерирование кода

GO

- protoc ...
- И МОЖНО ПОЛЬЗОВАТЬСЯ



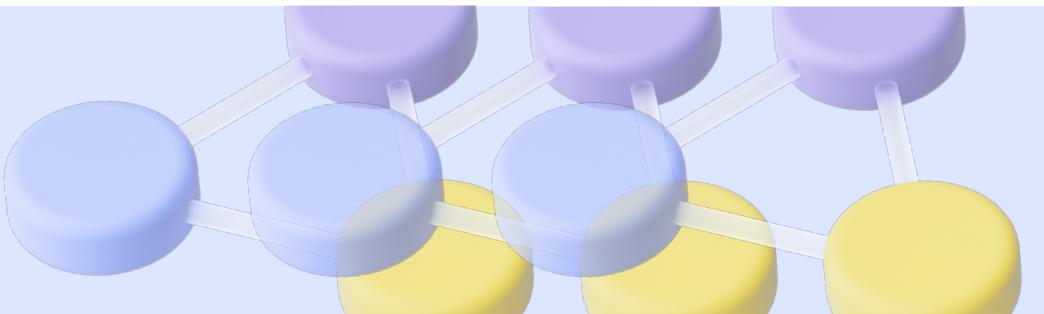
GRPC/protobuf, генерирование кода

GO

- protoc ...
- И МОЖНО ПОЛЬЗОВАТЬСЯ

Rust

-



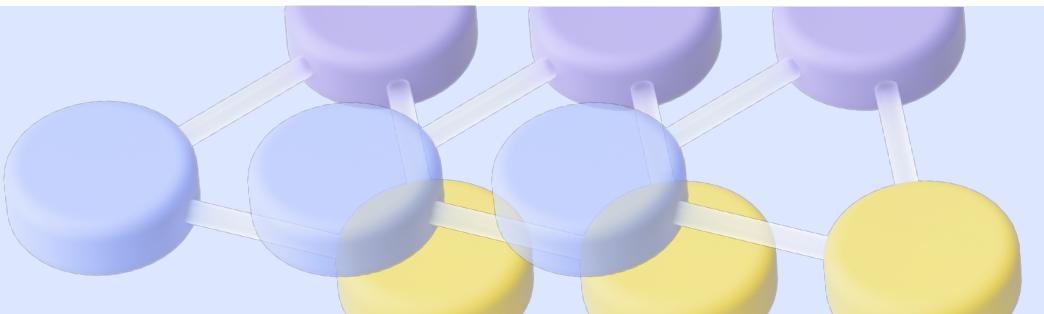
GRPC/protobuf, генерирование кода

GO

- protoc ...
- И можно пользоваться

Rust

- **tonic**



GRPC/protobuf, генерирование кода

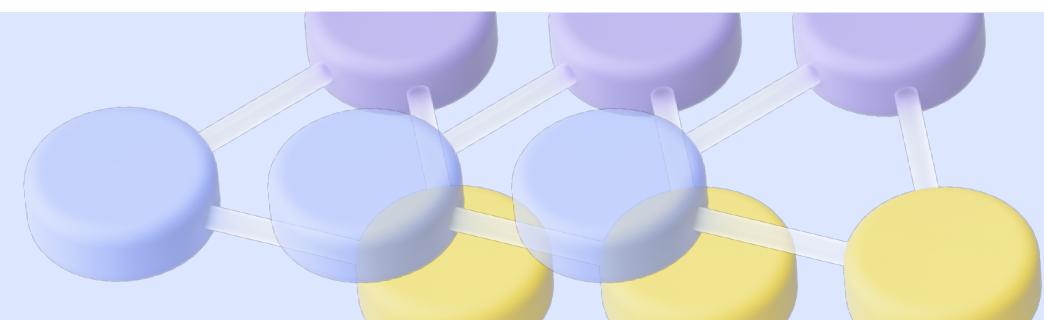
GO

- protoc ...
- И можно пользоваться

Rust

- tonic
- **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

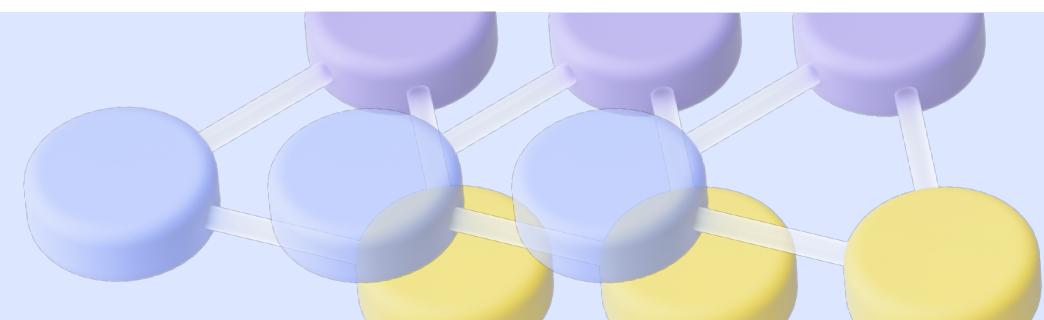
- protoc ...
- и можно пользоваться

Rust

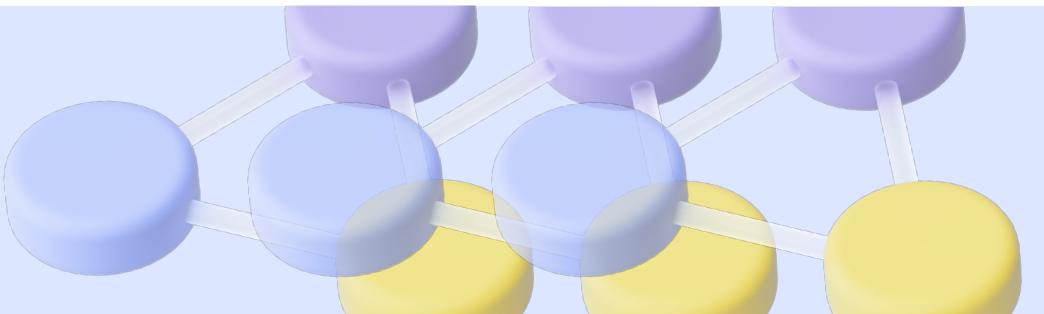
- tonic
- Составить правильный mod.rs

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

- **Сейчас уже лучше**

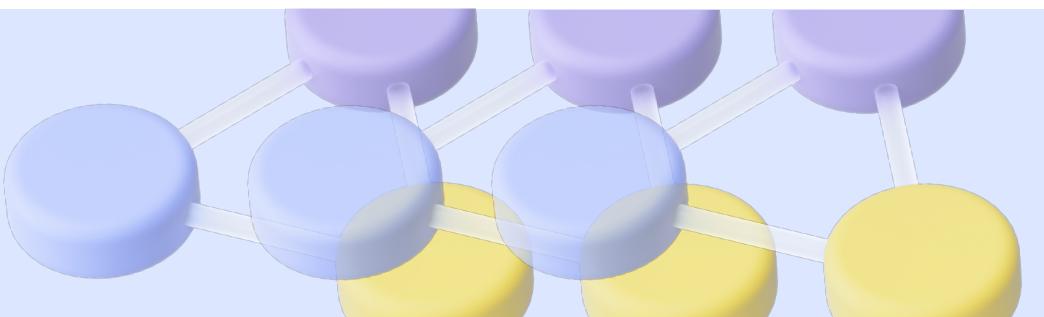


Protobuf-типы json и другие представления



Protobuf-типы json и другие представления

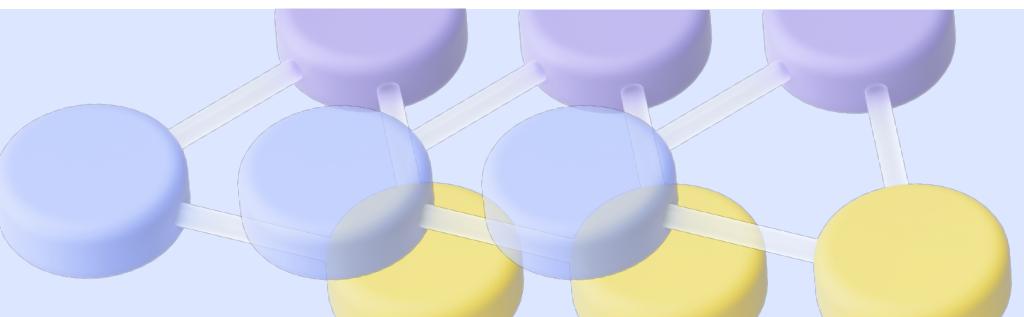
GO



Protobuf-типы json и другие представления

GO

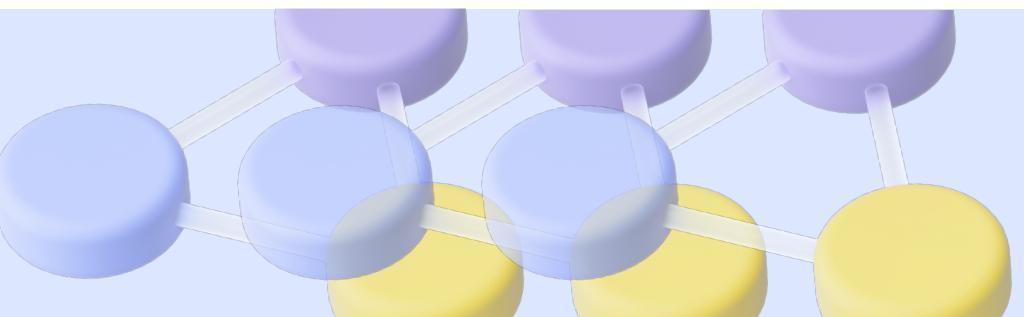
- Сериализация через reflection: не требуется поддержка в пакете со сгенерированным кодом



Protobuf-типы json и другие представления

GO

- Сериализация через reflection: не требуется поддержка в пакете со сгенерированным кодом
- **Можно пользоваться сразу**

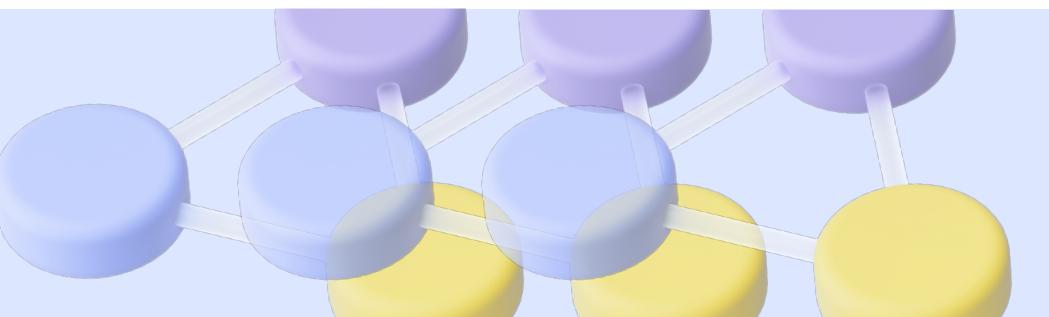


Protobuf-типы json и другие представления

GO

- Сериализация через reflection: не требуется поддержка в пакете со сгенерированным кодом
- Можно пользоваться сразу

Rust



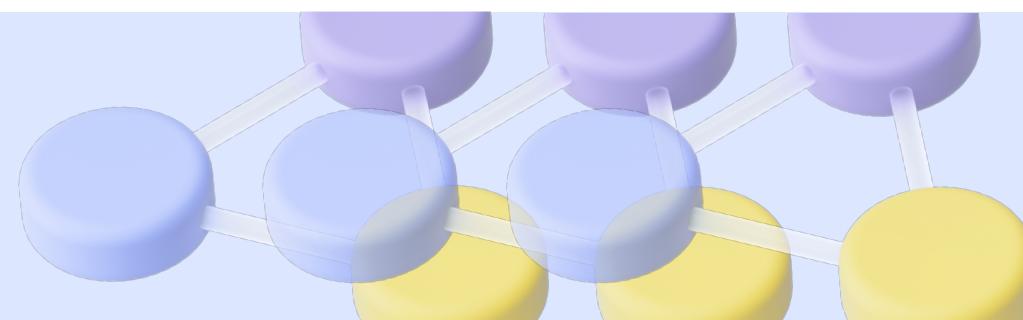
Protobuf-типы json и другие представления

GO

- Сериализация через reflection: не требуется поддержка в пакете со сгенерированным кодом
- Можно пользоваться сразу

Rust

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



Protobuf-типы json и другие представления

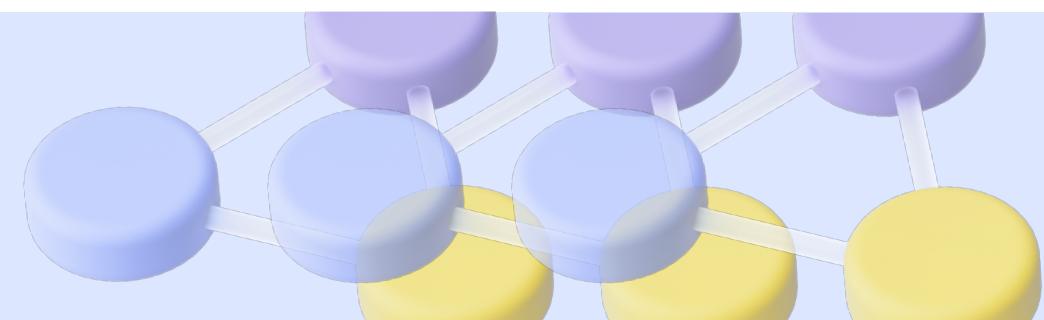
GO

- Сериализация через reflection: не требуется поддержка в пакете со сгенерированным кодом
- Можно пользоваться сразу

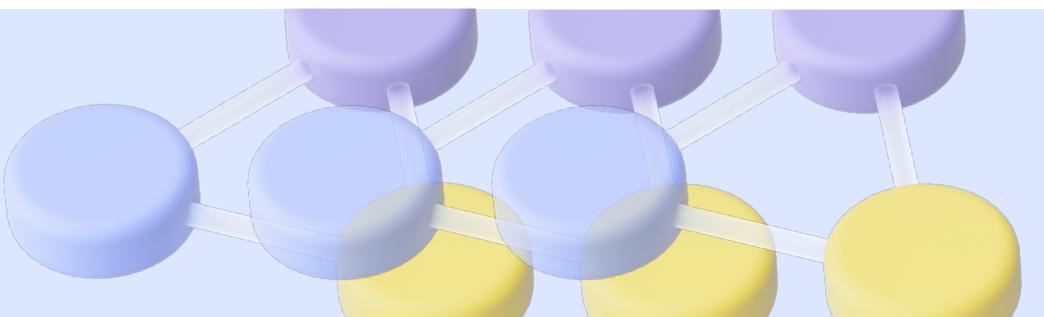
Rust

- 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)]",  
)  
...
```

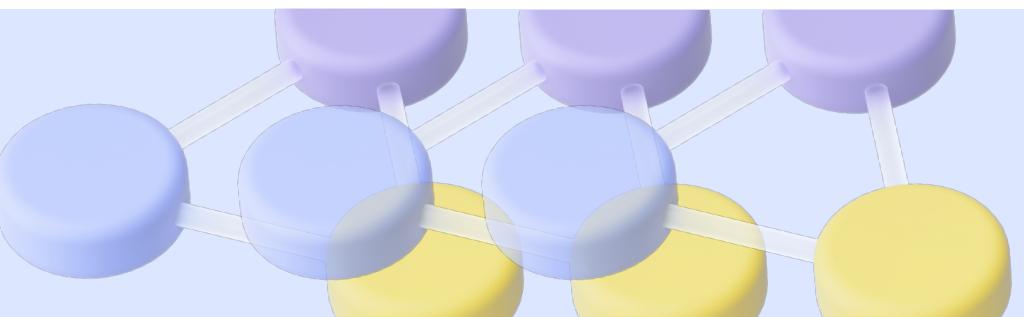


GRPC, обёртки



GRPC, обёртки

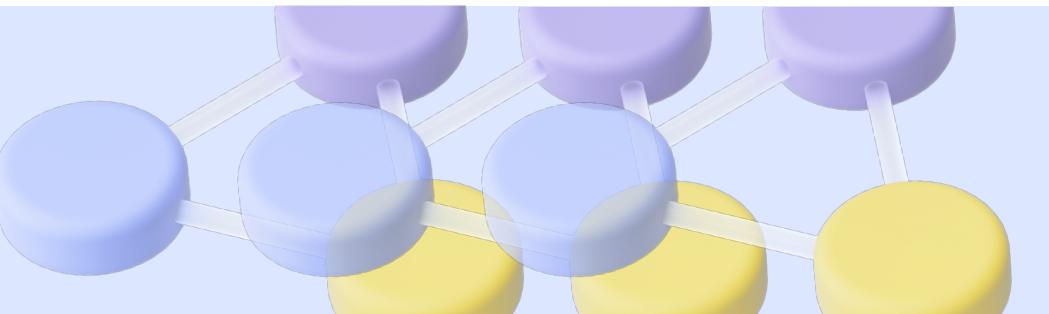
GO



GRPC, обёртки

GO

- Генерированный код работает с grpc через интерфейсы. Можно подменять транспорт без изменения типа клиента.

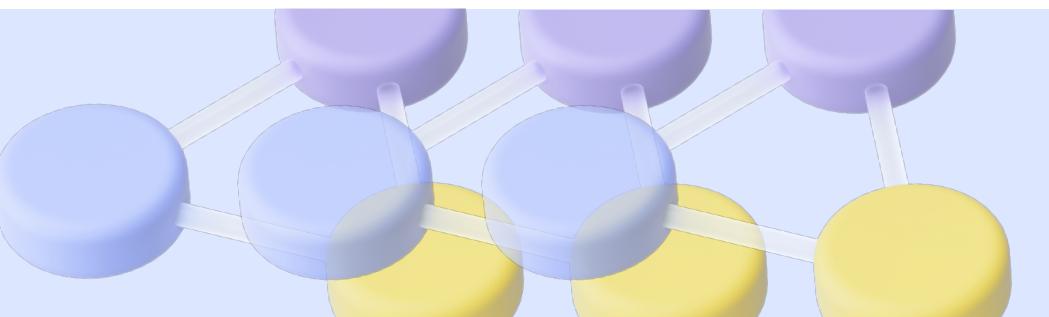


GRPC, обёртки

GO

- Генерированный код работает с grpc через интерфейсы.
Можно подменять транспорт без изменения типа клиента.

```
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

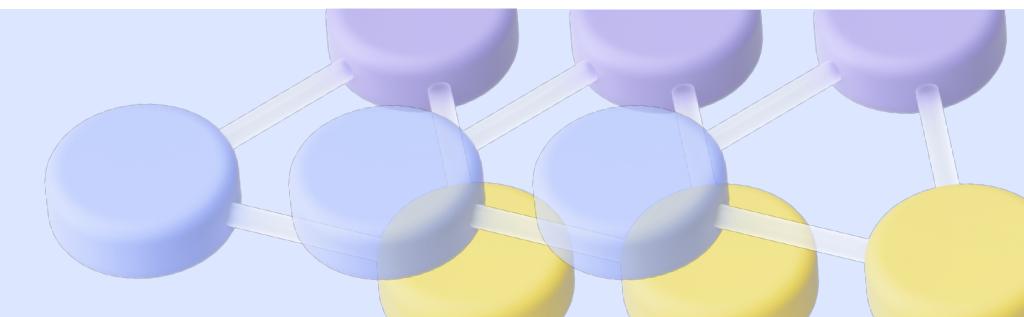
GO

- Генерированный код работает с grpc через интерфейсы.
Можно подменять транспорт без изменения типа клиента.

```
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

- Свои обработчики добавляются к коду клиента**



GRPC, обёртки

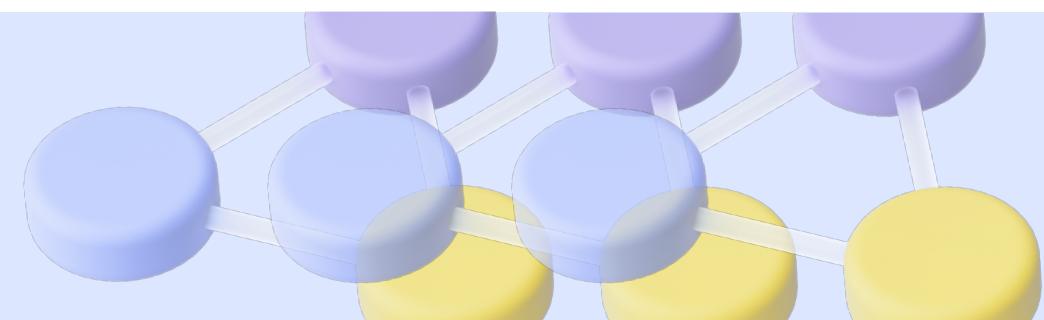
GO

- Генерированный код работает с grpc через интерфейсы.
Можно подменять транспорт без изменения типа клиента.

```
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

- Свои обработчики добавляются к коду клиента
- При добавлении обработчика меняется тип клиента**



GRPC, обёртки

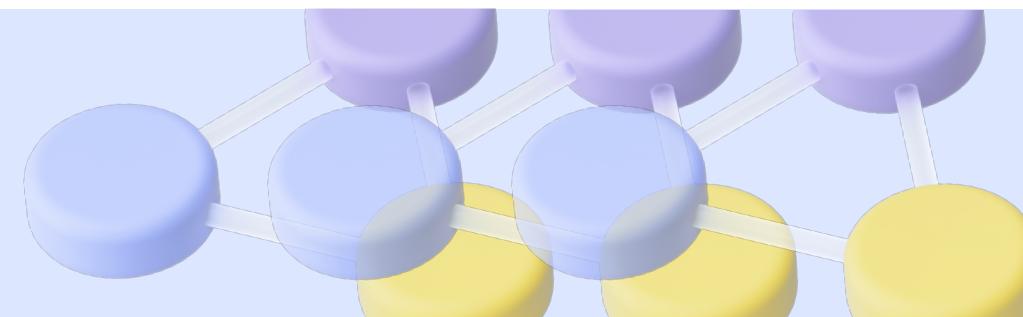
GO

- Генерированный код работает с grpc через интерфейсы.
Можно подменять транспорт без изменения типа клиента.

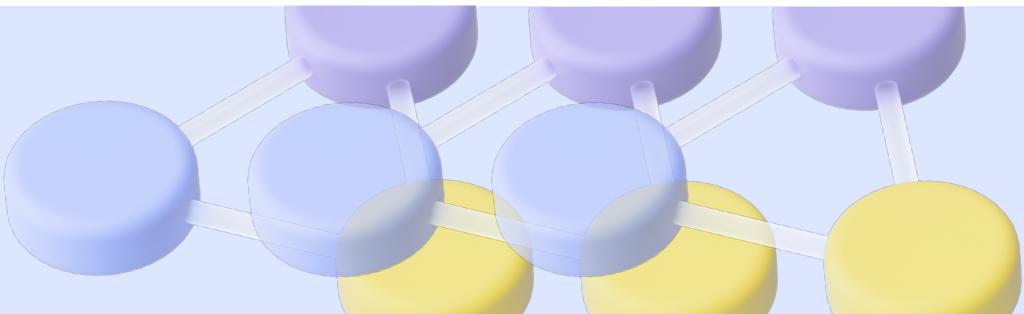
```
func New(cc grpc.ClientConnInterface, config
config.Config) *Client {
    return &Client{
        config: config,
        service:
Ydb_Scheme_V1.NewSchemeServiceClient(cc),
    }
}
type ClientConnInterface interface {
    Invoke(...) error
    NewStream(...) (ClientStream, error)
}
```

Rust

- Свои обработчики добавляются к коду клиента
- При добавлении обработчика меняется тип клиента
- При подмене транспорта тоже меняется тип клиента**

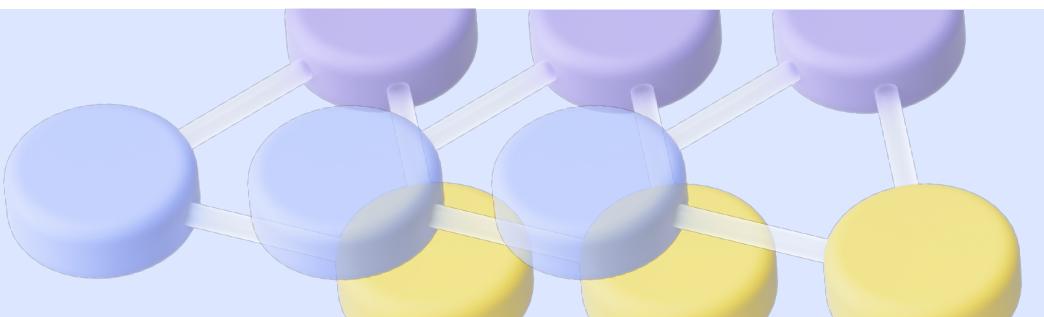


GRPC, своя поддержка обработчиков в транспорте

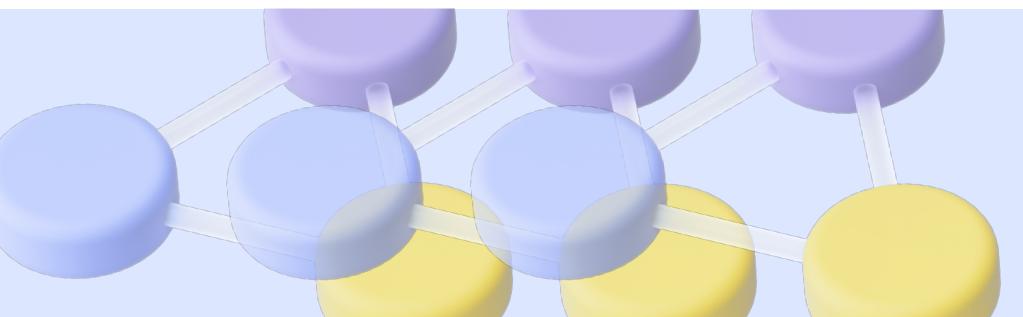


GRPC, своя поддержка обработчиков в транспорте

```
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;  
...  
...
```

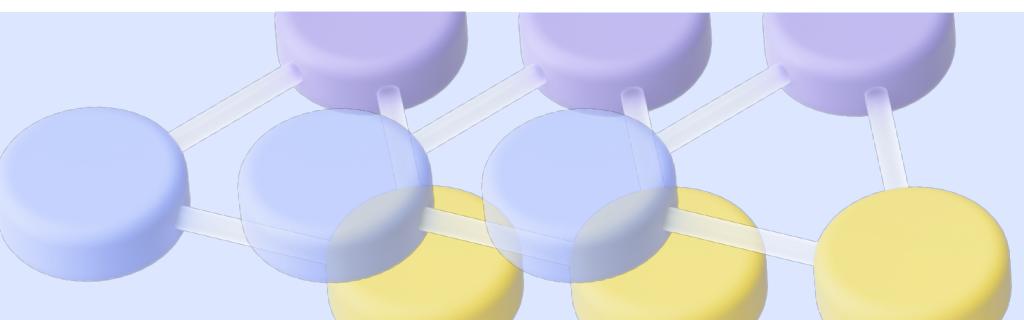


Управление таймаутами



Управление таймаутами

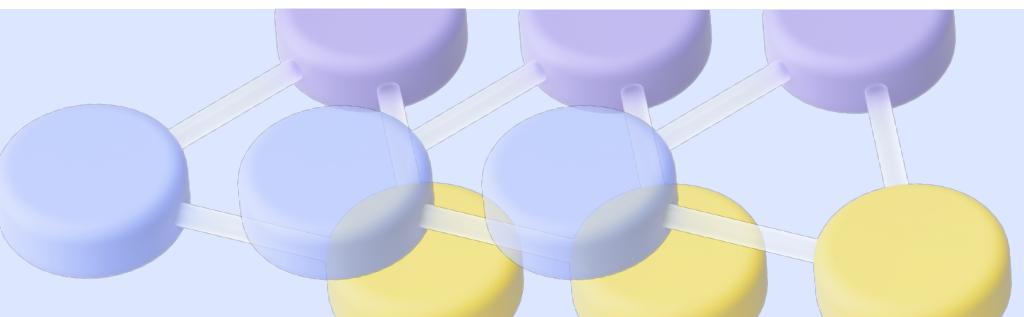
GO



Управление таймаутами

GO

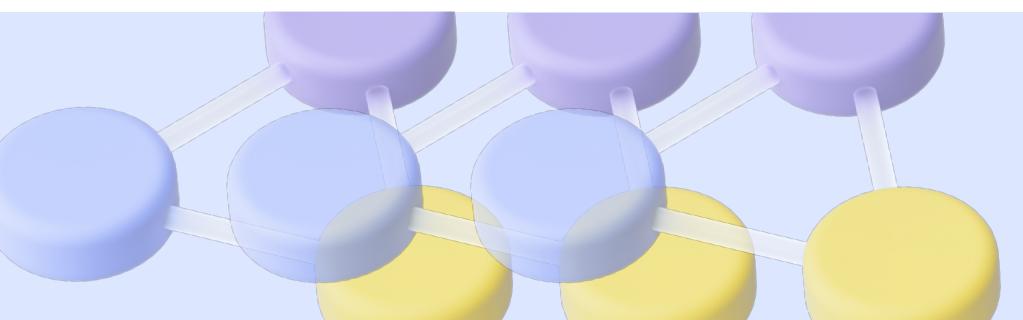
- Используются контексты из стандартной библиотеки



Управление таймаутами

GO

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



Управление таймаутами

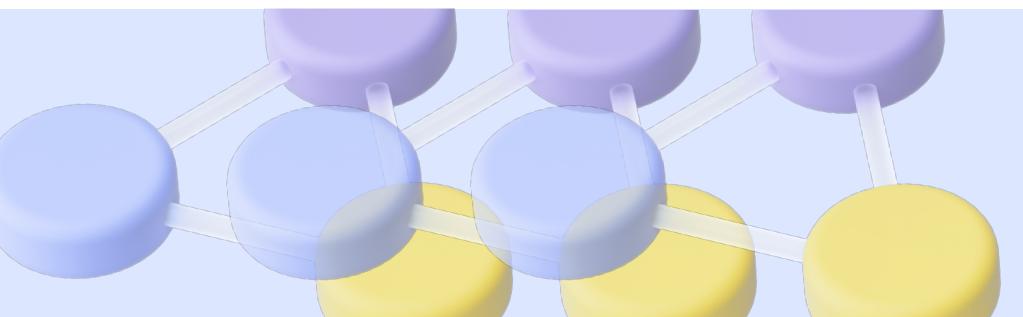
GO

- 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, _ := tx.Execute(  
    ctxTimeout, `SELECT 1`,  
    nil)
```

```
cancel()
```



Управление таймаутами

GO

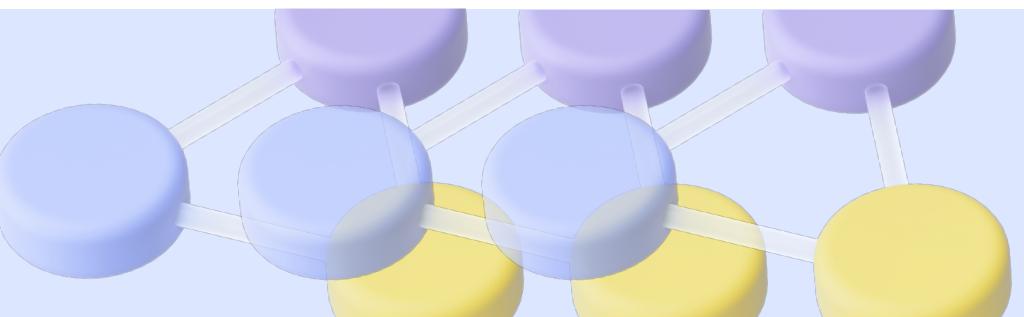
- Используются контексты из стандартной библиотеки
- Выполнение кода может отменяться не только по таймауту - контексты можно отменять в любое время.

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

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

```
cancel()
```

Rust



Управление таймаутами

GO

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

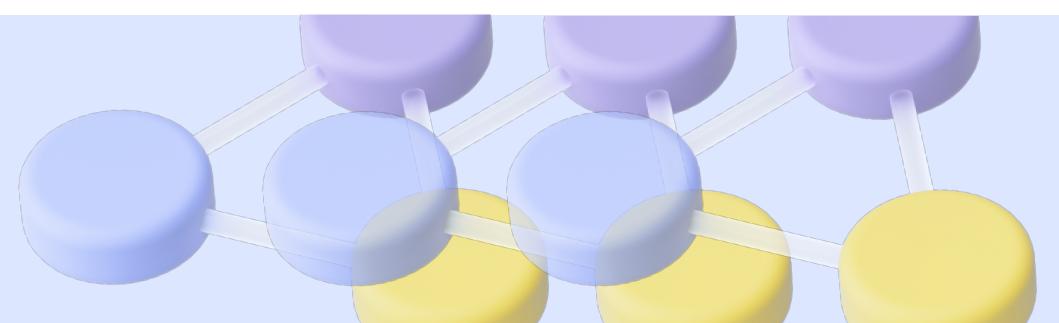
```
ctxTimeout, cancel := context.WithTimeout(  
    ctx, time.Second)
```

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

```
cancel()
```

Rust

- Функция **timeout** из **tokio**



Управление таймаутами

GO

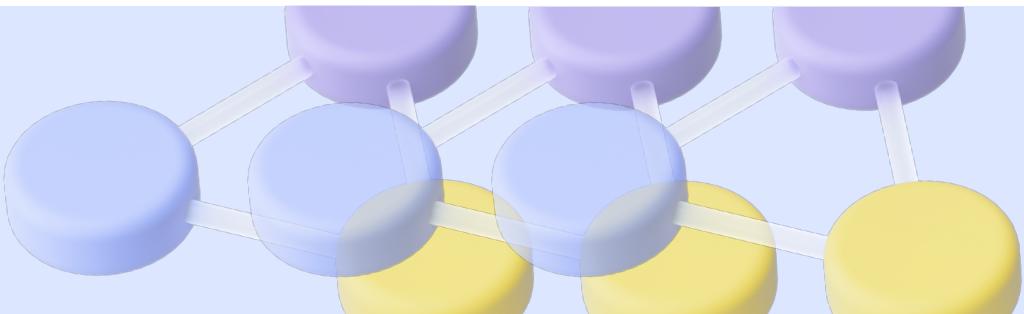
- Используются контексты из стандартной библиотеки
- Выполнение кода может отменяться не только по таймауту - контексты можно отменять в любое время.

```
ctxTimeout, cancel := context.WithTimeout(  
    ctx, time.Second)  
  
res, _ := tx.Execute(  
    ctxTimeout, `SELECT 1`,  
    nil)  
  
cancel()
```

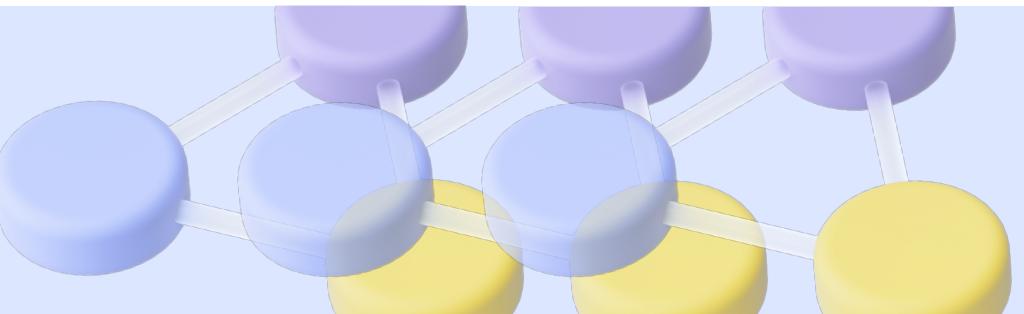
Rust

- Функция **timeout** из **tokio**

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

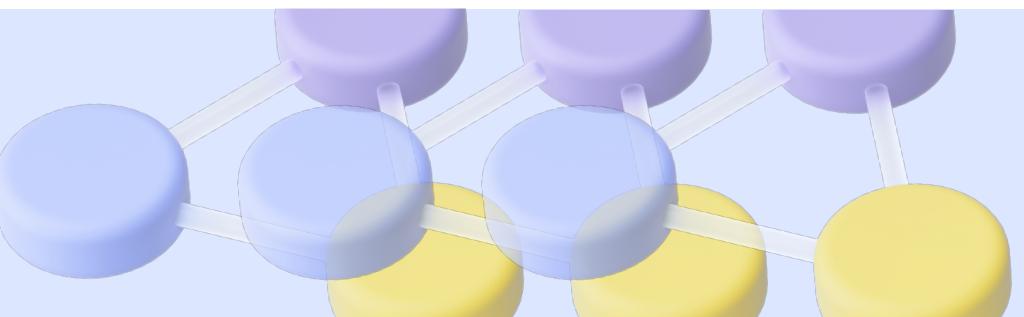


Передача параметров запроса



Передача параметров запроса

GO

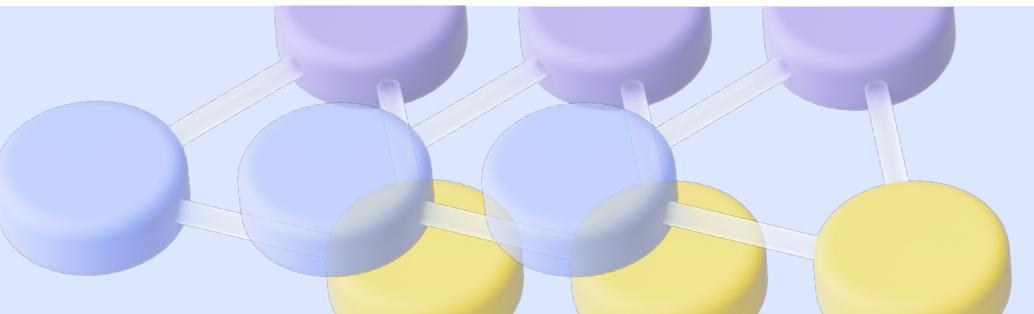


Передача параметров запроса

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

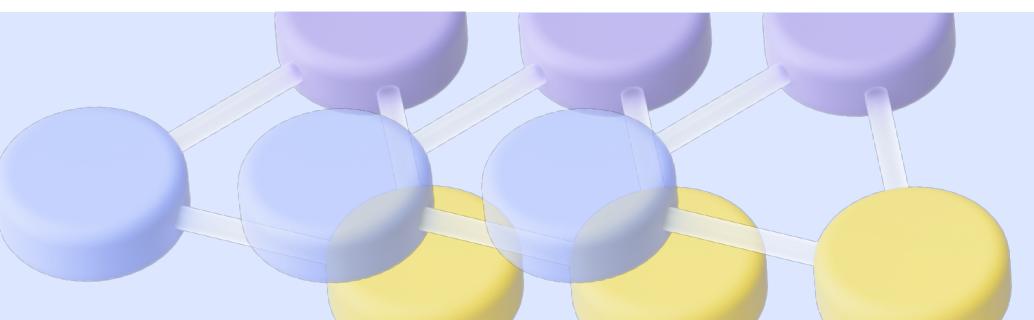
GO

- 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  
}
```

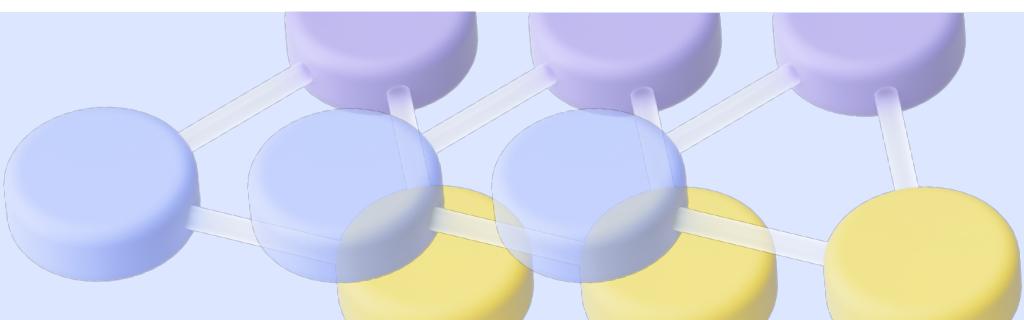
- Внутри

```
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
```

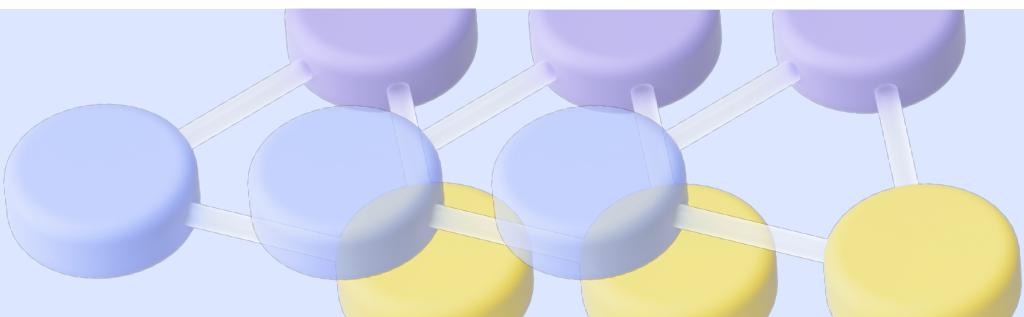
Rust

- Со стороны пользователя

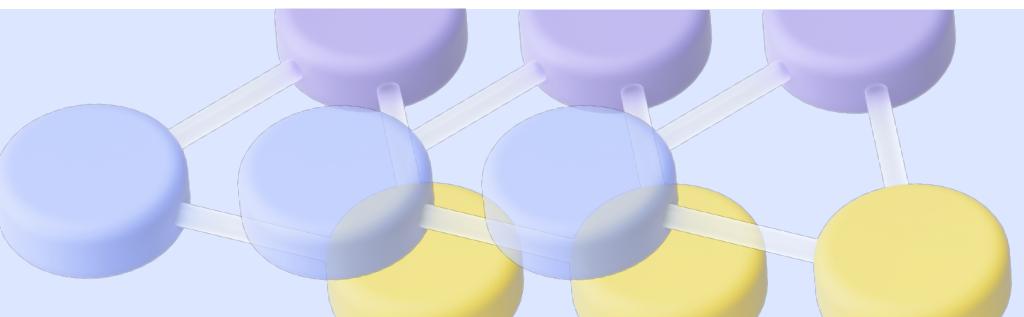
```
tx.query(Query::new(`  
DECLARE $val AS Int32;  
SELECT $val AS result`  
`).with_params(ydb_params!  
    "$val" => 1,  
)));
```

- Внутри

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

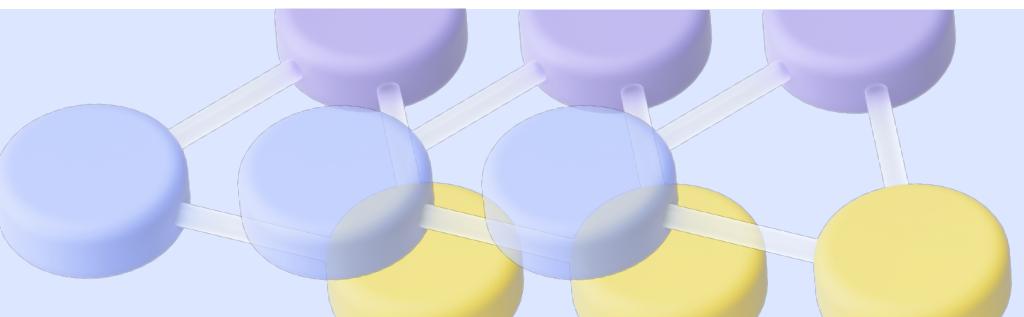


Чтение результатов в переменные



Чтение результатов в переменные

GO

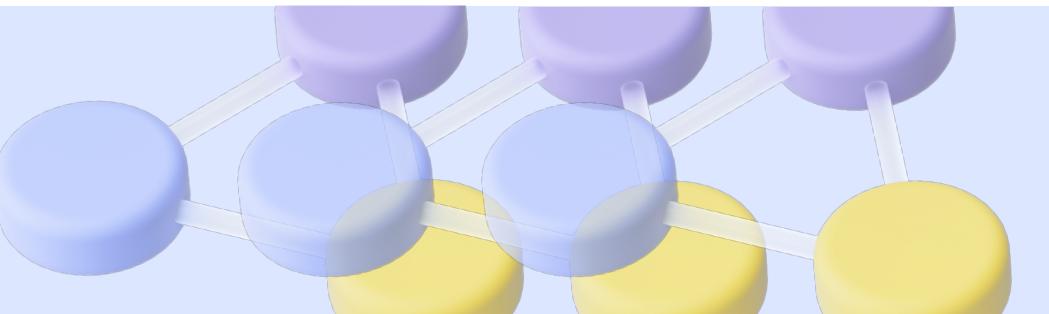


Чтение результатов в переменные

GO

- Со стороны пользователя

```
var a int32  
var b string  
err := res.Scan(&a, &b)
```



Чтение результатов в переменные

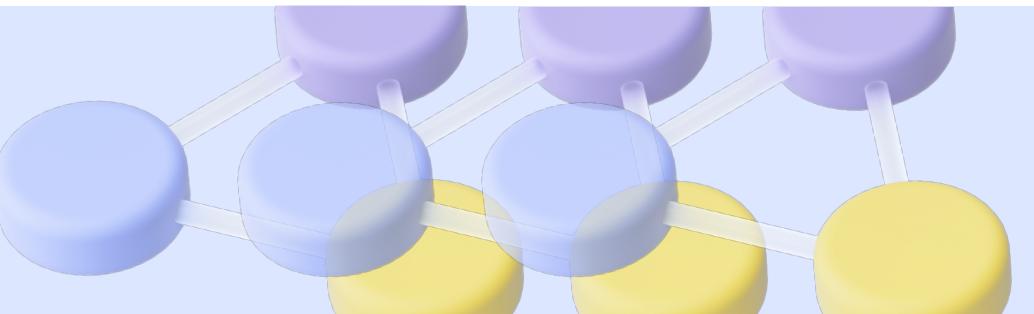
GO

- Со стороны пользователя

```
var a int32
var b string
err := res.Scan(&a, &b)
```

- Внутри

```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```



Чтение результатов в переменные

GO

- Со стороны пользователя

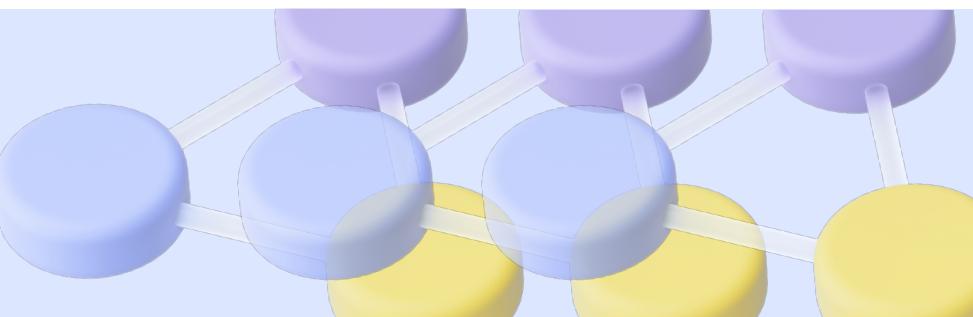
```
var a int32
var b string
err := res.Scan(&a, &b)
```

- Внутри

```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```

Rust

•



Чтение результатов в переменные

GO

- Со стороны пользователя

```
var a int32
var b string
err := res.Scan(&a, &b)
```

- Внутри

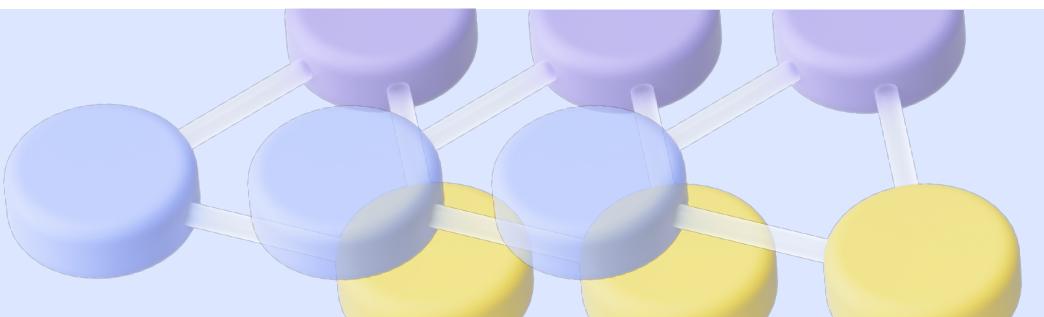
```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```

Rust

- Со стороны пользователя

```
let a: i32 = row.remove_field(0)?.try_into()?;
let b: 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)
```

- Внутри

```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```

Rust

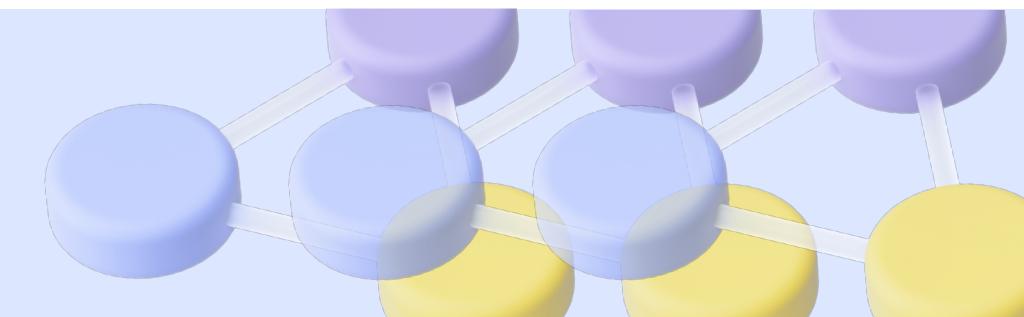
- Со стороны пользователя

```
let a: i32 = row.remove_field(0)?.try_into()?;
let b: 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)
```

- Внутри

```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```

Rust

- Со стороны пользователя

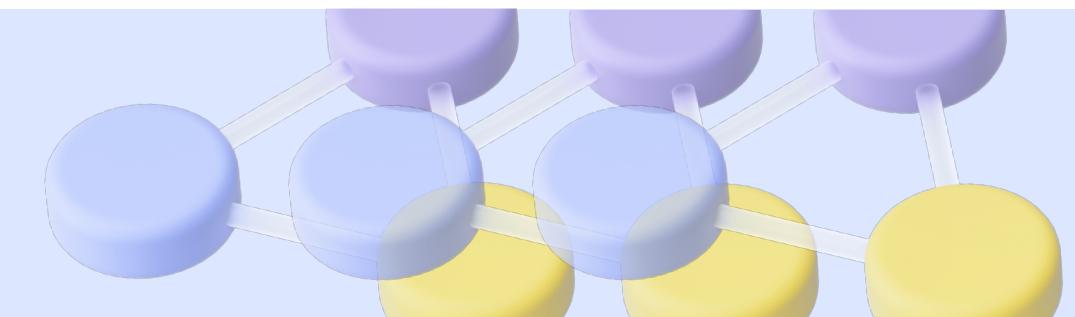
```
let a: i32 = row.remove_field(0)?.try_into()?;
let b: String = row.remove_field(1)?.try_into()?
```

- 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);
```

3 варианта
на 6 строках



Чтение результатов в переменные

GO

- Со стороны пользователя

```
var a int32
var b string
err := res.Scan(&a, &b)
```

- Внутри

```
func (s *scanner) scanRequired(v interface{}) {
switch v := v.(type) {
case *bool:
    *v = s.bool()
case *int8:
    *v = s.int8()
case *int16:
    *v = s.int16()
```

3 варианта
на 6 строках

Rust

- Со стороны пользователя

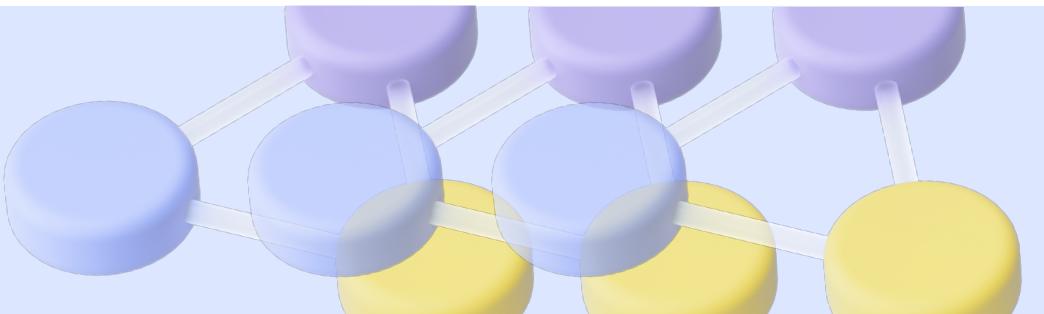
```
let a: i32 = row.remove_field(0)?.try_into()?;
let b: String = row.remove_field(1)?.try_into()?
```

- 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



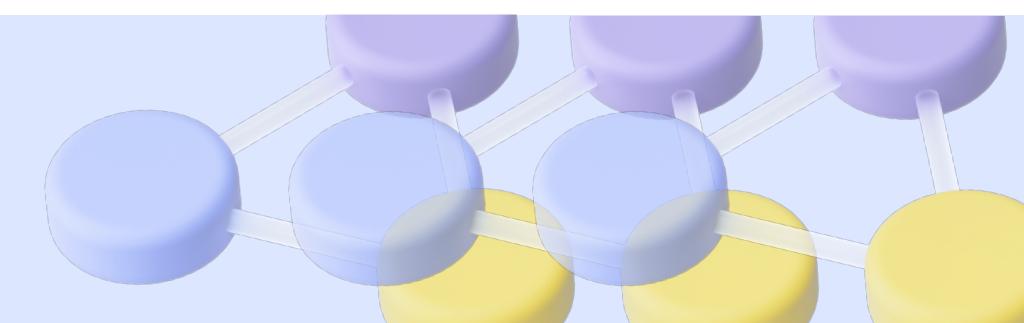
Конвертация Value, макрос simple_convert в Rust

```
($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)
        }
    }

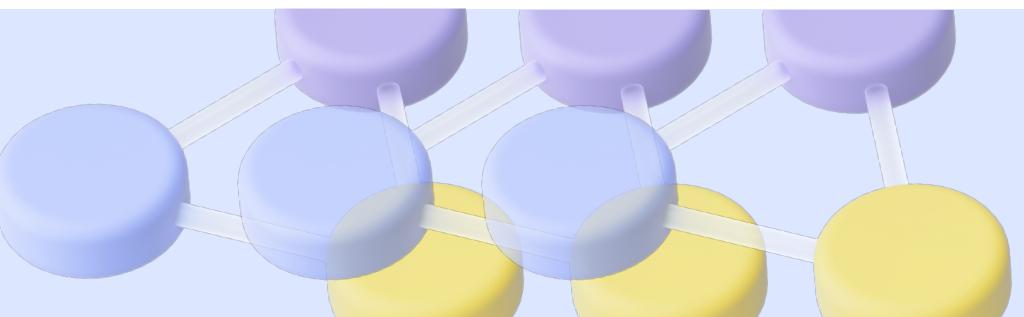
    impl TryFrom<Value> for $native_type {
        type Error = YdbError;

        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>(),
                )));
            }
        }
    }
}

...
}
```



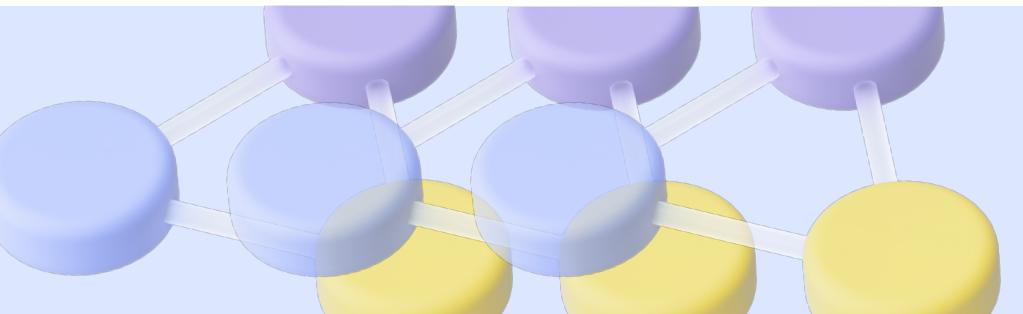
Возврат сессий в пул



Возврат сессий в пул

GO

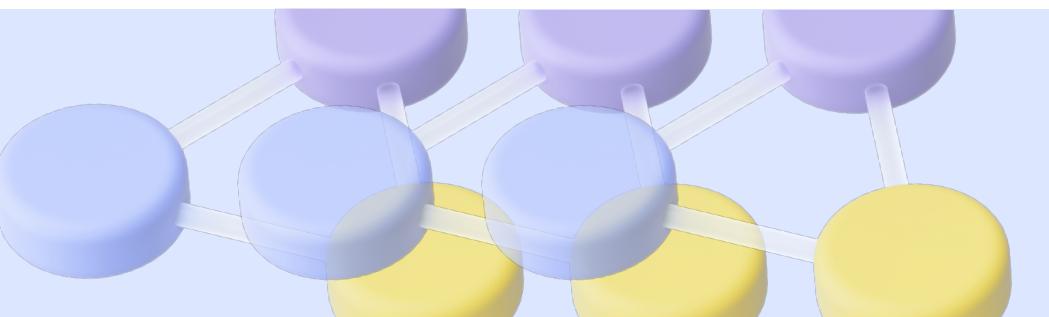
- **Нет деструкторов, финализаторы хрупкие и с задержками.**



Возврат сессий в пул

GO

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



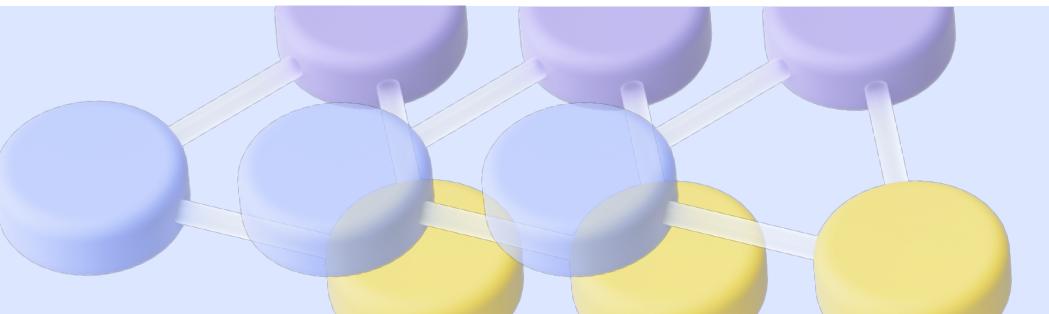
Возврат сессий в пул

GO

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

```
s, _ = p.Get(ctx)  
defer p.Put(ctx, s)
```

```
op(ctx, s) // <- надеяться на лучшее
```



Возврат сессий в пул

GO

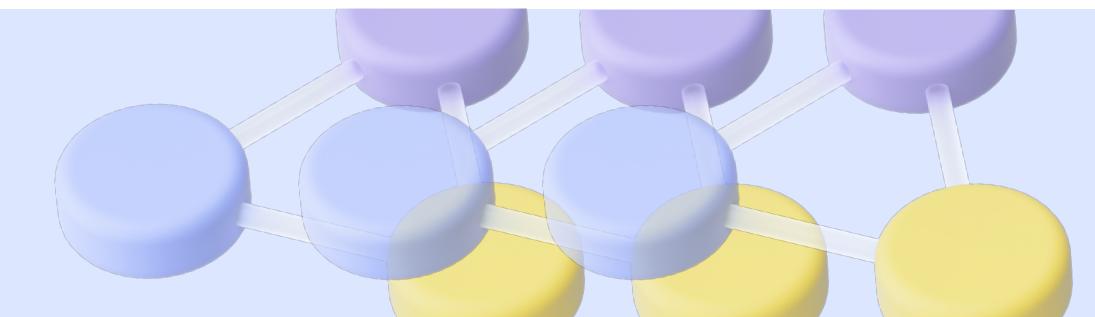
- Нет деструкторов, финализаторы хрупкие и с задержками.
- Более-менее надёжный способ - только callback для работы с сессией

```
s, _ = p.Get(ctx)  
defer p.Put(ctx, s)
```

op(ctx, s) // <- надеяться на лучшее

Rust

- **Есть Drop-trait**



Возврат сессий в пул

GO

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

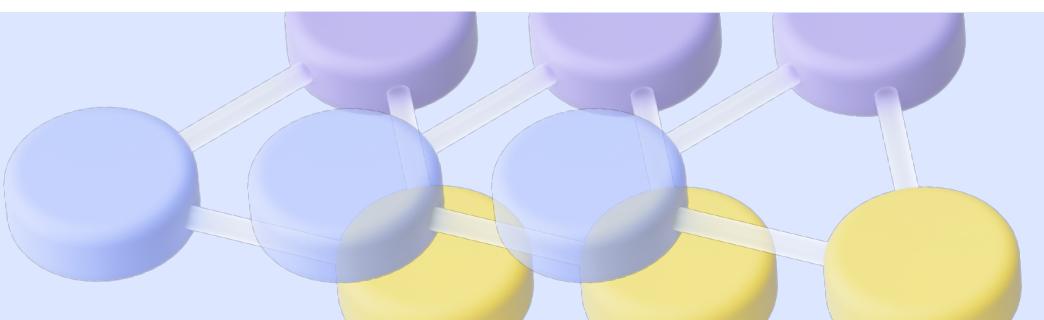
```
s, _ = p.Get(ctx)
defer p.Put(ctx, s)

op(ctx, s) // <- надеяться на лучшее
```

Rust

- **Есть Drop-trait**

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

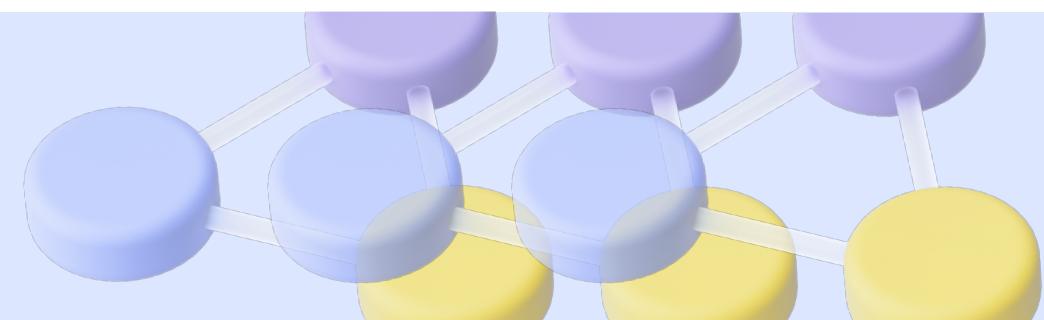


Есть задачи для развития

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	+	+/-	+	+	+/-	-	-



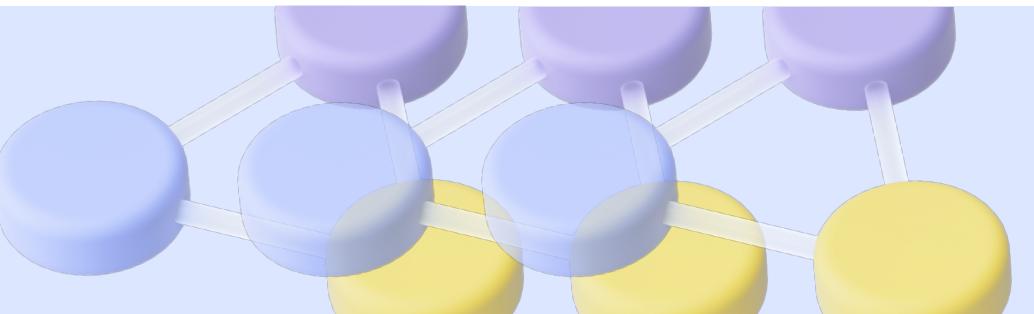
<https://ydb.tech/ru/docs/reference/ydb-sdk/feature-parity>



Наблюдения

GO

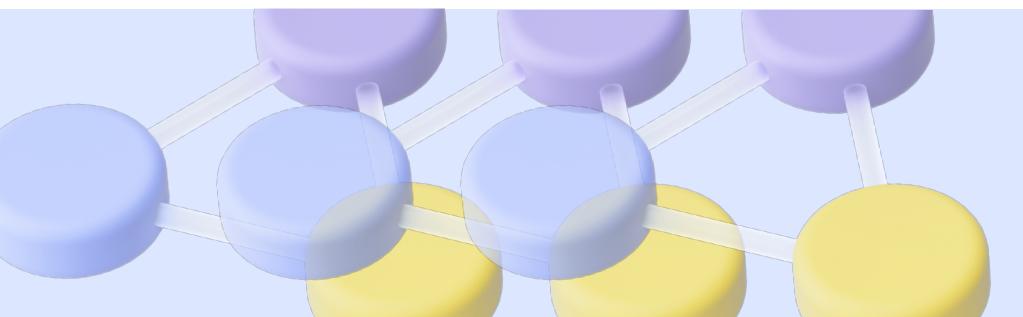
- Типизация доставляет меньше проблем



Наблюдения

GO

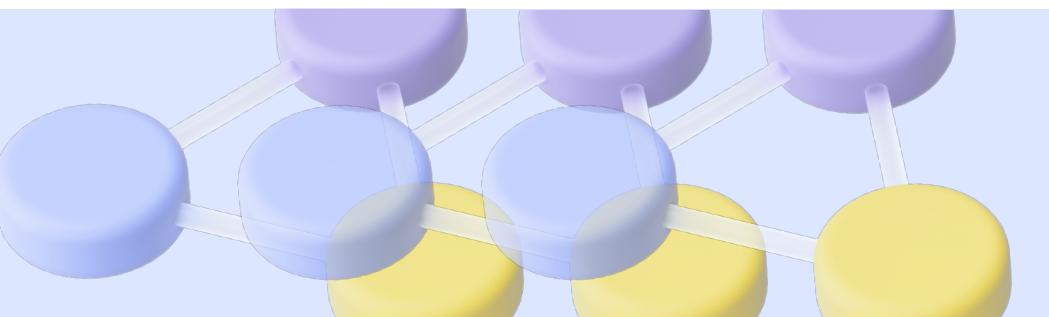
- Типизация доставляет меньше проблем
- **Reflection позволяет работать с неподконтрольными мне типами**



Наблюдения

GO

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



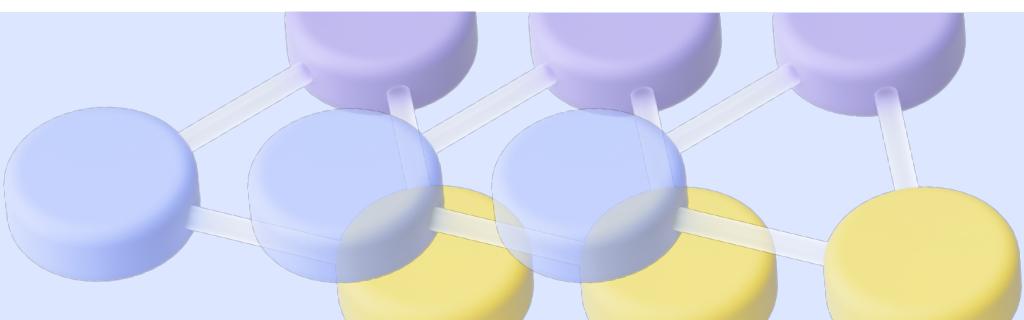
Наблюдения

GO

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

Rust

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



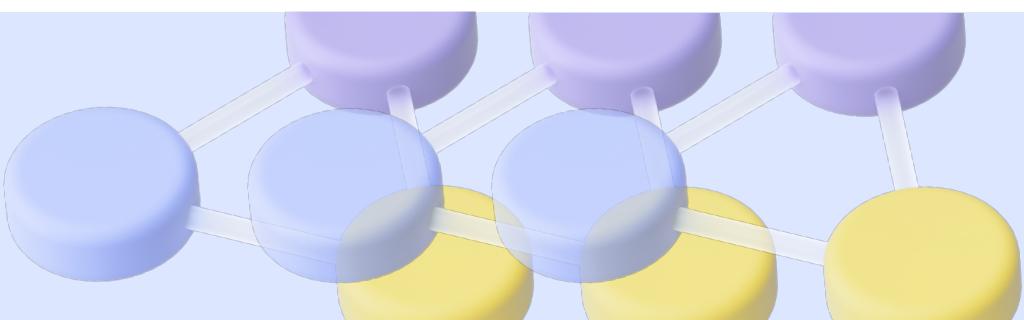
Наблюдения

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**



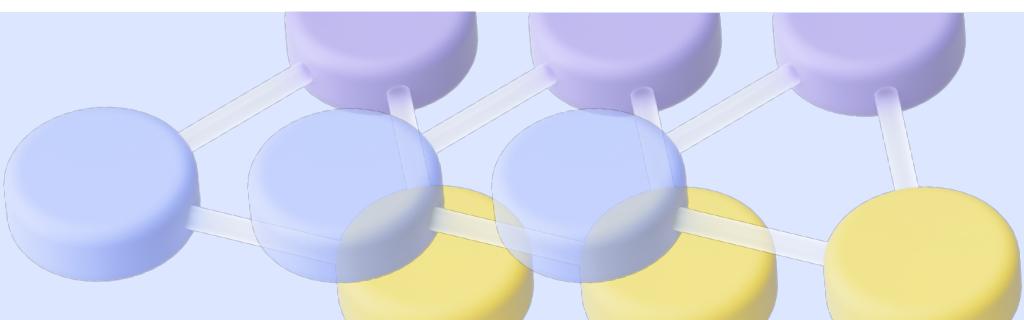
Наблюдения

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**



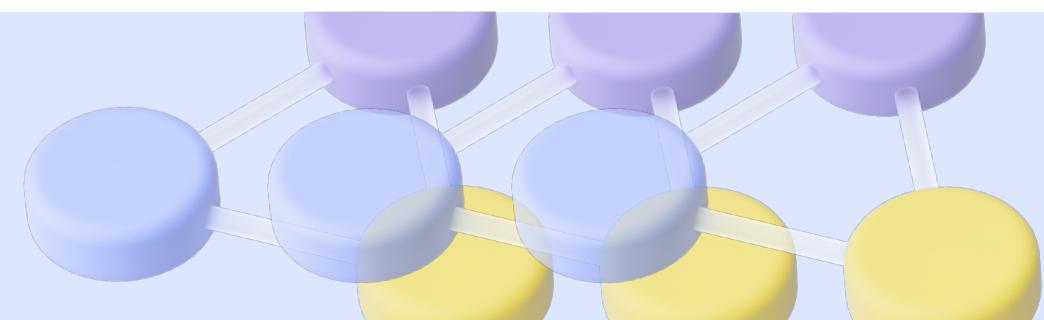
Наблюдения

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**



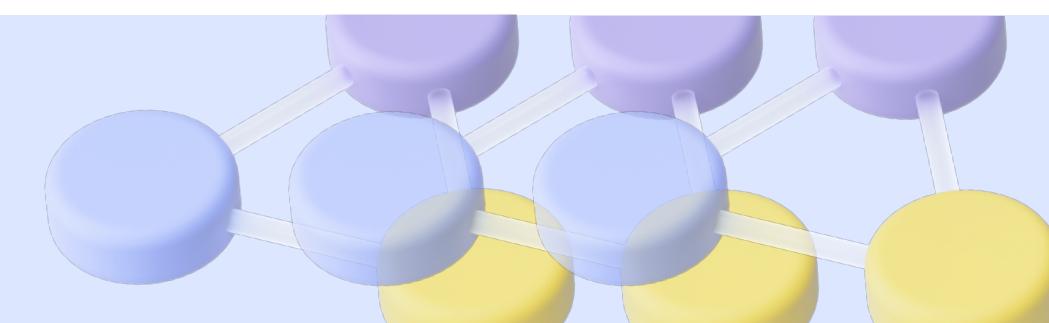
Наблюдения

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 как вирус распространяются на весь код**



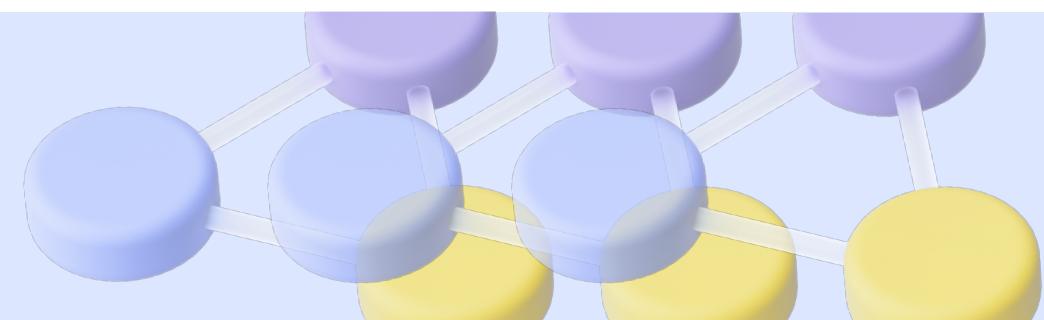
Наблюдения

GO

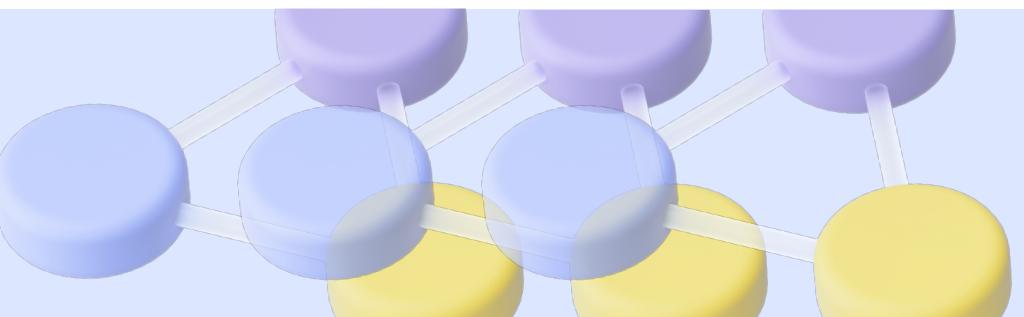
- Типизация доставляет меньше проблем
- Reflection позволяет работать с неподконтрольными мне типами
- Удобное управление таймаутами

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 как вирус распространяются на весь код
- **Нестабильные версии зависимостей (бывают заметные изменения, требующие правки на своей стороне).**

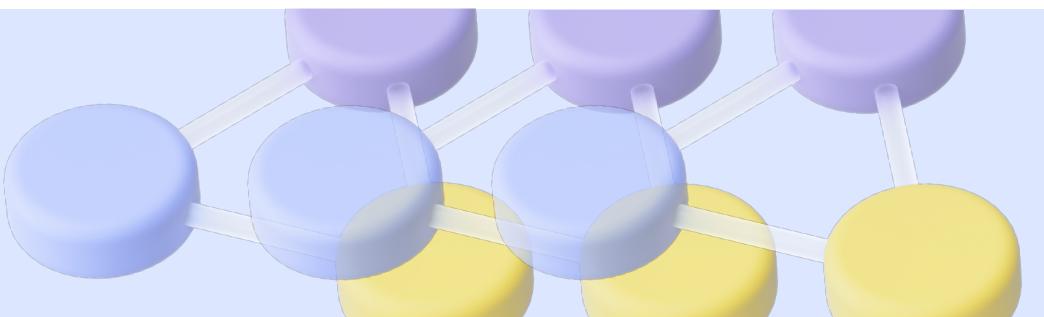


Выводы



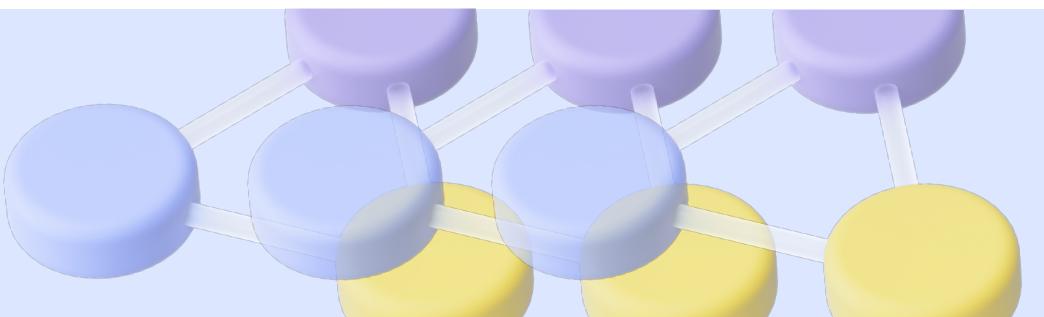
Выводы

- Не получится просто взять и переписать библиотеку 1 в 1 - слишком разные подходы, потребуется переосмысление



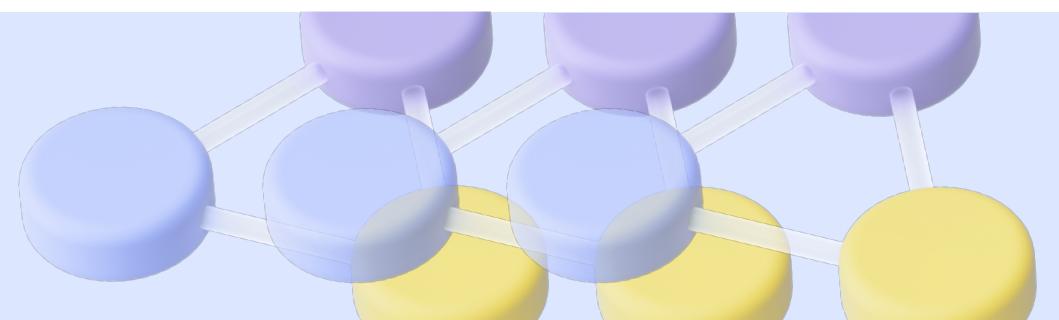
Выводы

- 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**



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
<https://t.me/bobsmit>

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>

