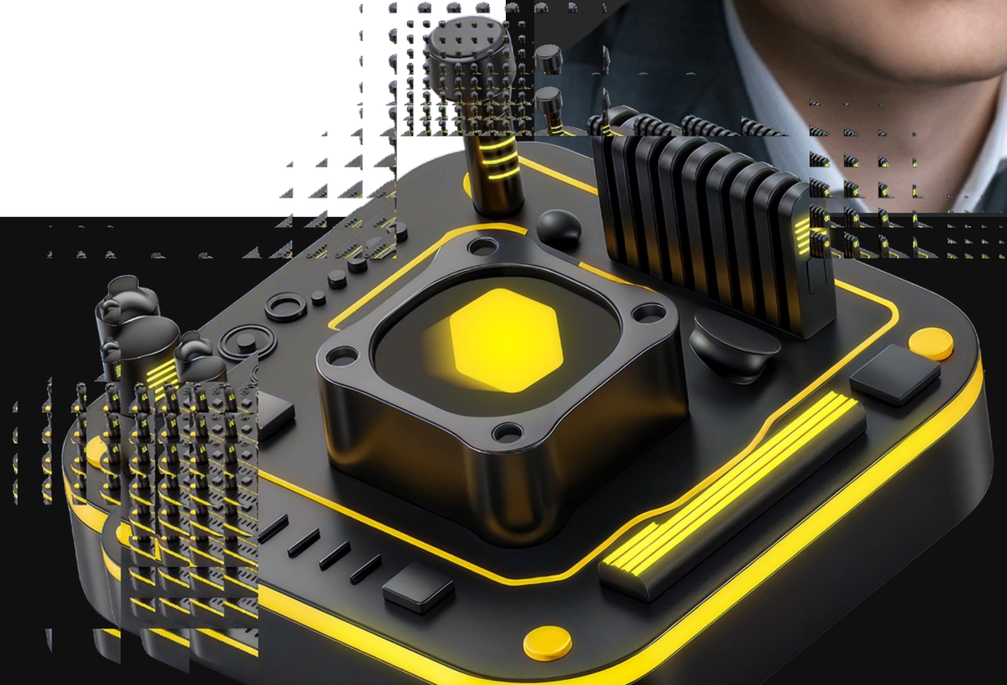
из C++ делаем Go-lang

Полухин Антон
Эксперт разработчик C++



ГОРОДСКИЕ СЕРВИСЫ

ГОРОДСКИЕ СЕРВИСЫ

О ЧЁМ ПОГОВОРИМ

Зачем?!?!?! 01

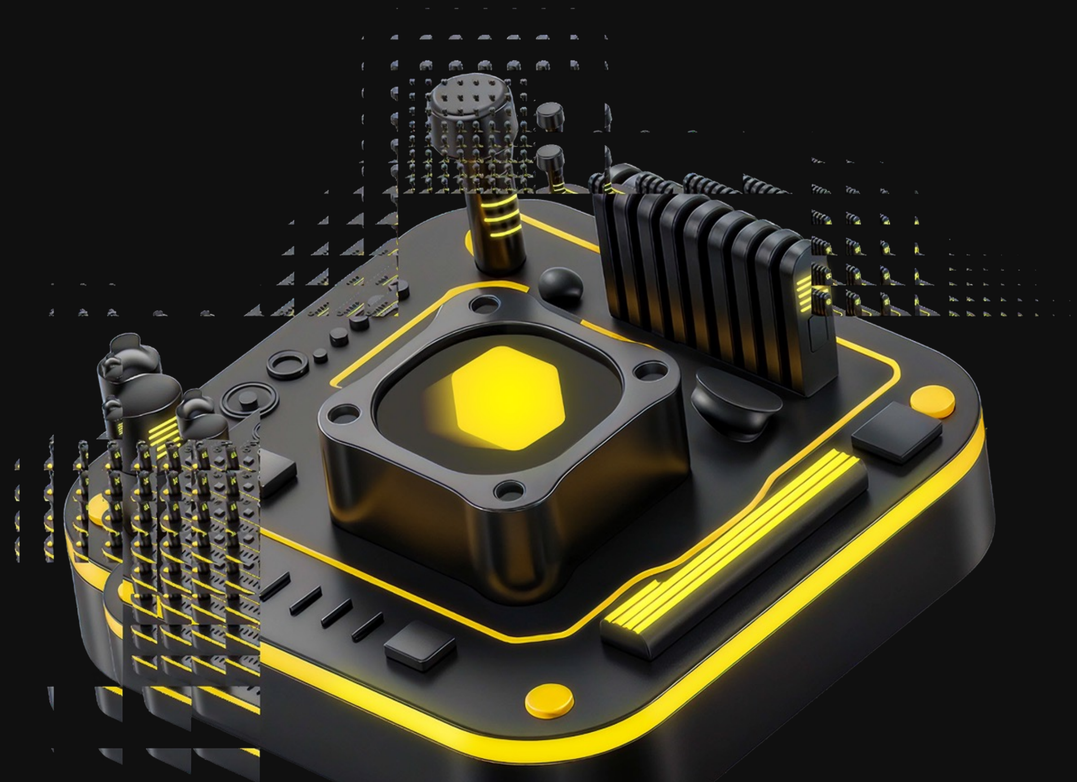
Как? 02

Получился ли Go-lang? 03

Достаточно ли этого для production? 04

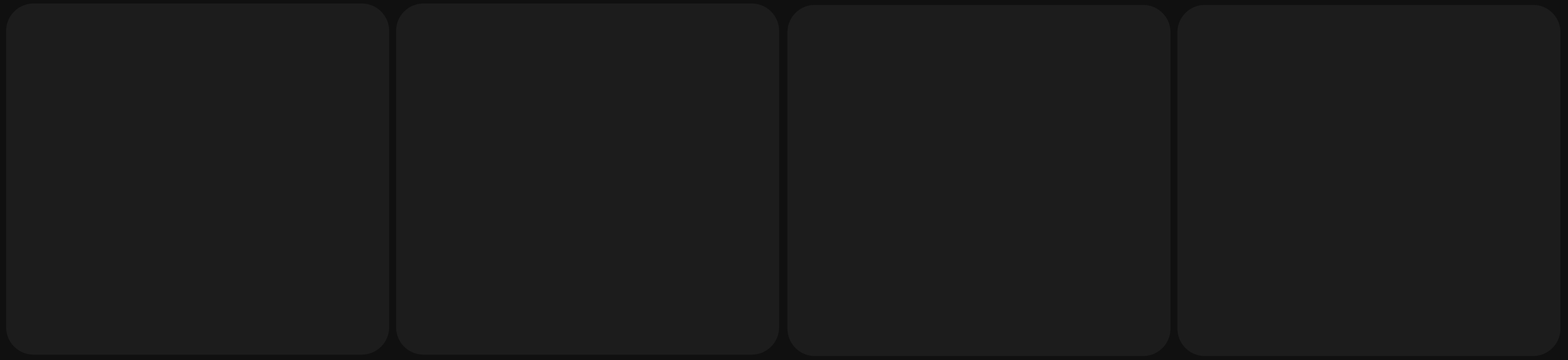
01

зачем!?!?!



зачем?

что было в компании на начало проекта



зачем?

что было в компании на начало проекта

01

Огромная кодовая база на C++

зачем?

что было в компании на начало проекта

01

Огромная кодовая база на C++

02

Большой штат C++ разработчиков

зачем?

что было в компании на начало проекта

01

Огромная кодовая база на C++

02

Большой штат C++ разработчиков

03

Высокая нагрузка на железо, CPU интенсивные задачи

зачем?

что было в компании на начало проекта

01

Огромная кодовая база на C++

02

Большой штат C++ разработчиков

03

Высокая нагрузка на железо, CPU интенсивные задачи

04

Ожидалось большое количество IO-bound задач

зачем?

что было в мире на начало проекта

зачем?

что было в мире на начало проекта

01

Для C++ не было ничего удобного для работы с IO-bound задачами

зачем?

что было в мире на начало проекта

01

Для C++ не было ничего удобного для работы с IO-bound задачами

02

Другие языки влекут за собой увеличение расходов

зачем?

что было в мире на начало проекта

01

Для C++ не было ничего удобного для работы с IO-bound задачами

02

Другие языки влекут за собой увеличение расходов

03

Senior разработчиков в мире мало

зачем?

что было в мире на начало проекта

01

Для C++ не было ничего удобного для работы с IO-bound задачами

02

Другие языки влекут за собой увеличение расходов

03

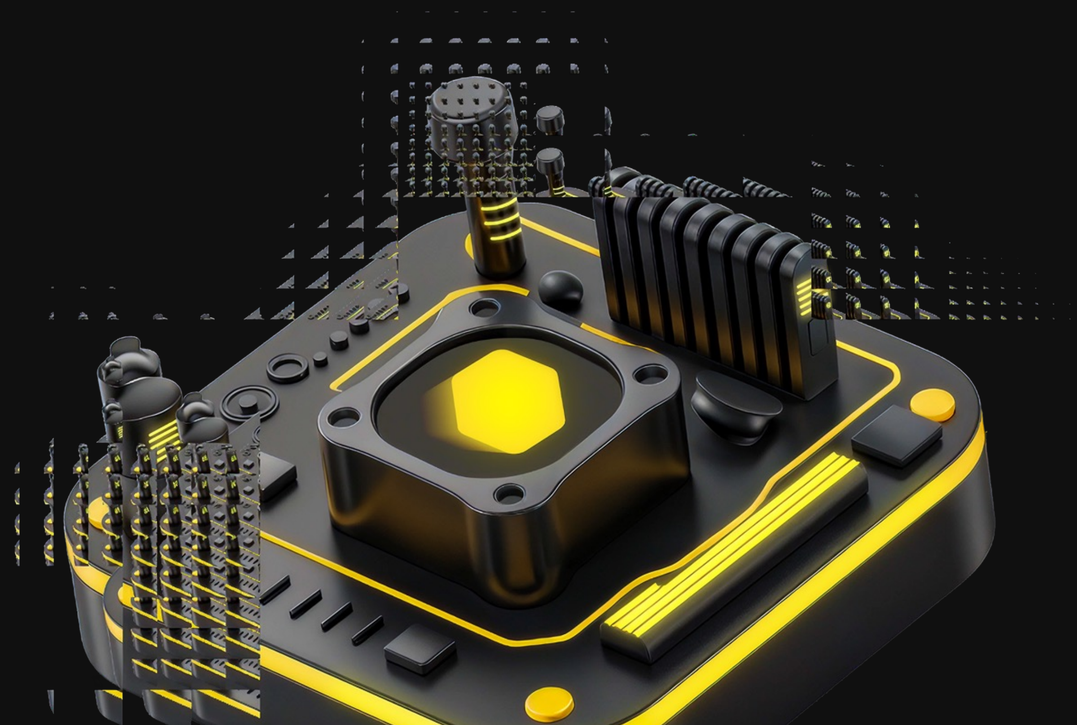
Senior разработчиков в мире мало

04

Другие языки/фреймворки не решают проблемы production

итак

C++, простота, production-oriented

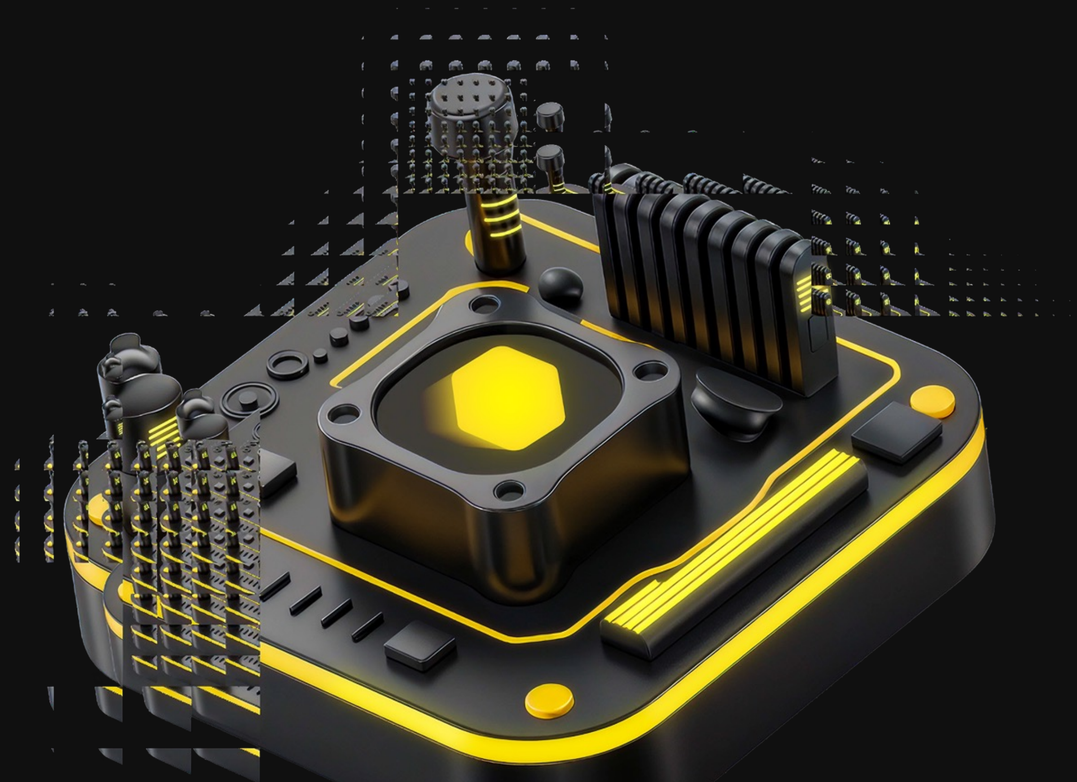


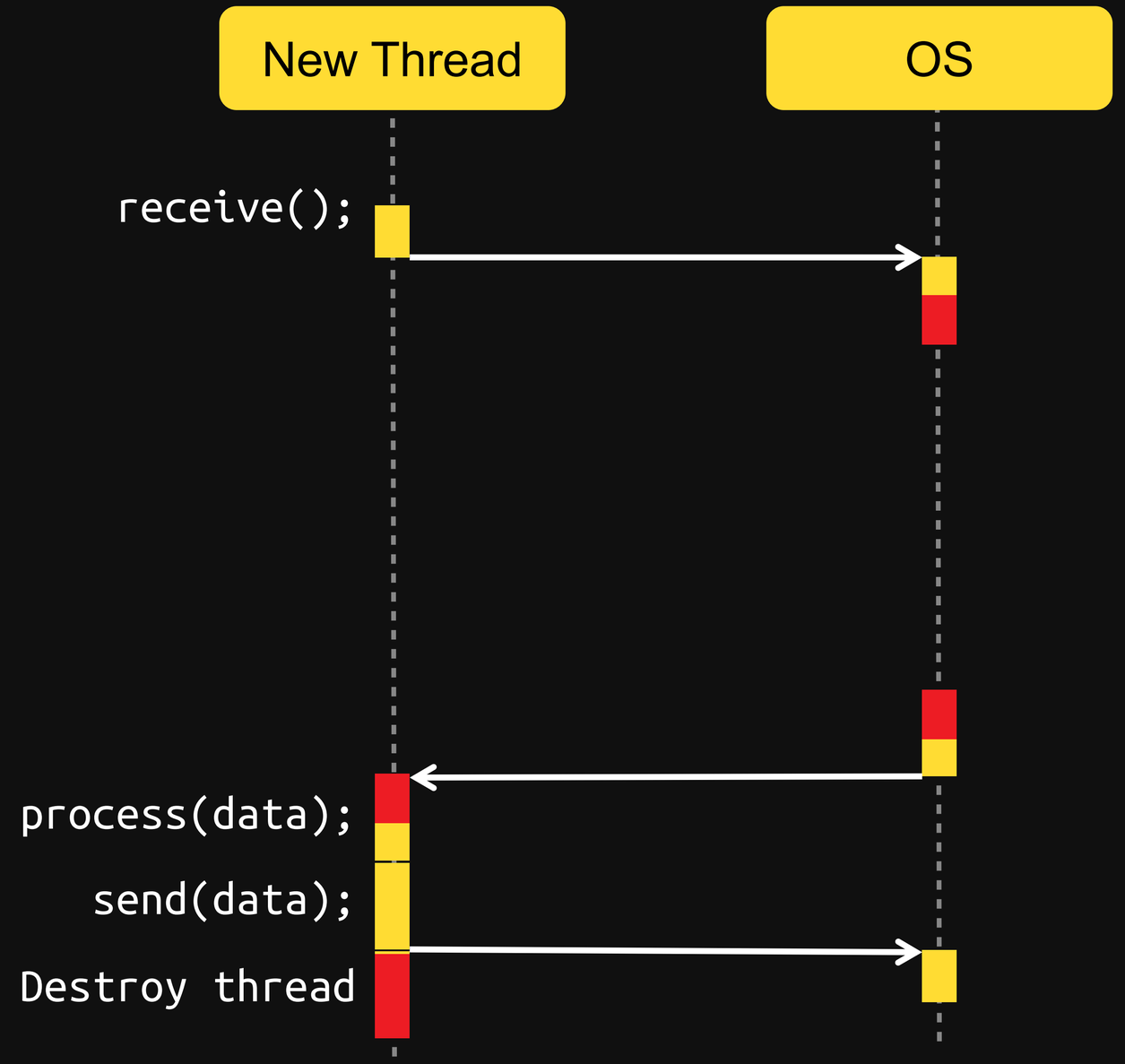

Шедеврум

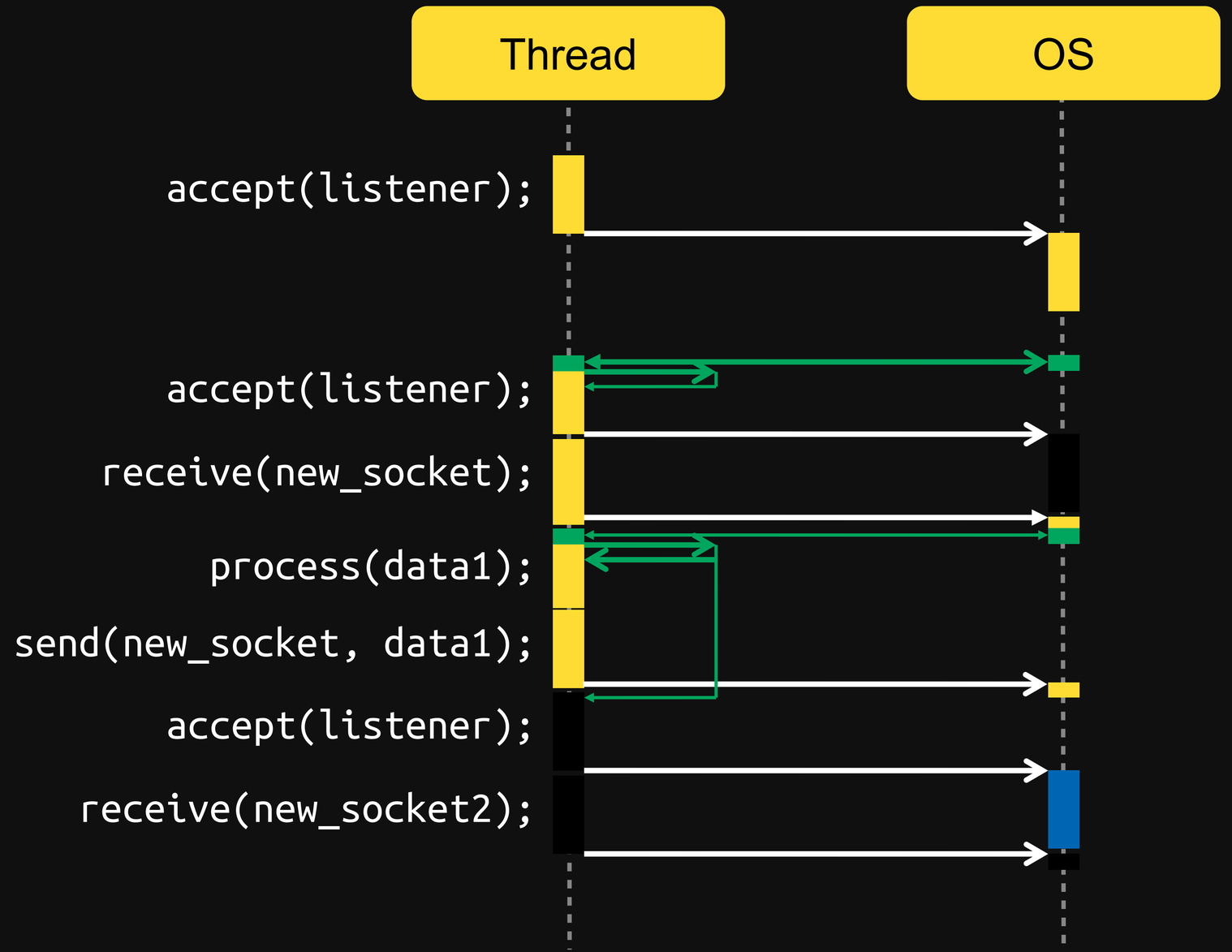

Шедеврум

01

как?




New Thread
OS
receive();
process(data);
send(data);
Destroy thread


Thread
OS
accept(listener);
accept(listener);
receive(new_socket);
process(data1);
send(new_socket, data1);
accept(listener);
receive(new_socket2);

callback hell

```
void View::Handle(Request&& request, const Dependencies& dependencies, Response
response) {
  dependencies.pg->GetCluster(
    [request = std::move(request), response](auto cluster)
  {
    cluster->Begin(storages::postgres::ClusterHostType::kMaster,
      [request = std::move(request), response](auto& trx)
    {
      const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
      psql::Execute(trx, statement, request.id,
        [request = std::move(request), response, trx = std::move(trx)](auto& res)
      {
        auto row = res[0];
        if (!row["ok"].As<bool>()) {
          if (LogDebug()) {
```

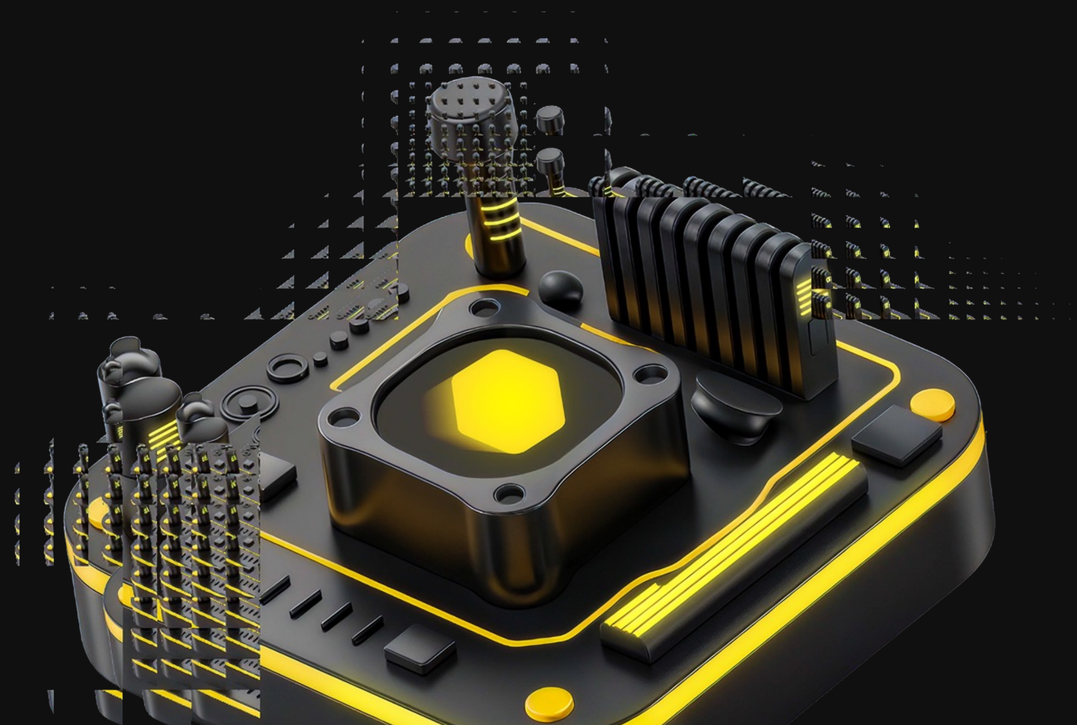
callback hell

```
GetSomeInfoFromDb([id = request.id](auto info) {
                  LOG_DEBUG() << id << " is not OK of " << info;
              });
          }
          *response = Response400{};
        }
        psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar,
          [row = std::move(row), trx = std::move(trx), response]()
        {
          trx.Commit([row = std::move(row), response]() {
            *response = Response200{row["baz"].As<std::string>()};
          });
        });
      });
    });
  });
}
```

callback hell

```
  });
}
```

50-летняя идея



coroutines stackless

```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = co_await dependencies.pg->GetCluster();
 auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = (co_await psql::Execute(trx, statement, request.id))[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << co_await GetSomeInfoFromDb();
   return Response400();
 }

 co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 co_await trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

coroutines stackless

```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = co_await dependencies.pg->GetCluster();
 auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = (co_await psql::Execute(trx, statement, request.id))[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << co_await GetSomeInfoFromDb();
   return Response400();
 }

 co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 co_await trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

coroutines stackless

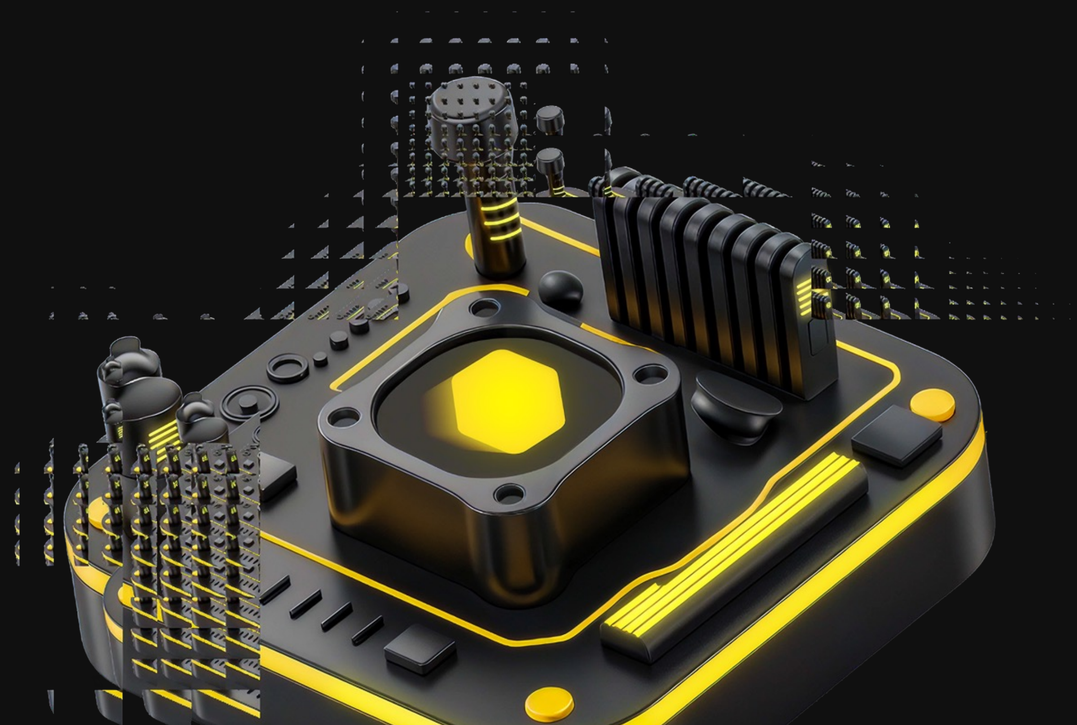
```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = co_await dependencies.pg->GetCluster();
 auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = (co_await psql::Execute(trx, statement, request.id))[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << co_await GetSomeInfoFromDb();
   return Response400();
 }

 co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 co_await trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

50-летняя идея без поддержки компилятора



coroutines stackfull

```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = dependencies.pg->GetCluster();
 auto trx = cluster->Begin(storages::postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = psql::Execute(trx, statement, request.id)[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << GetSomeInfoFromDb();
   return Response400();
 }

 psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

coroutines stackless

```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = co_await dependencies.pg->GetCluster();
 auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = (co_await psql::Execute(trx, statement, request.id))[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << co_await GetSomeInfoFromDb();
   return Response400();
 }

 co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 co_await trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

coroutines stackfull

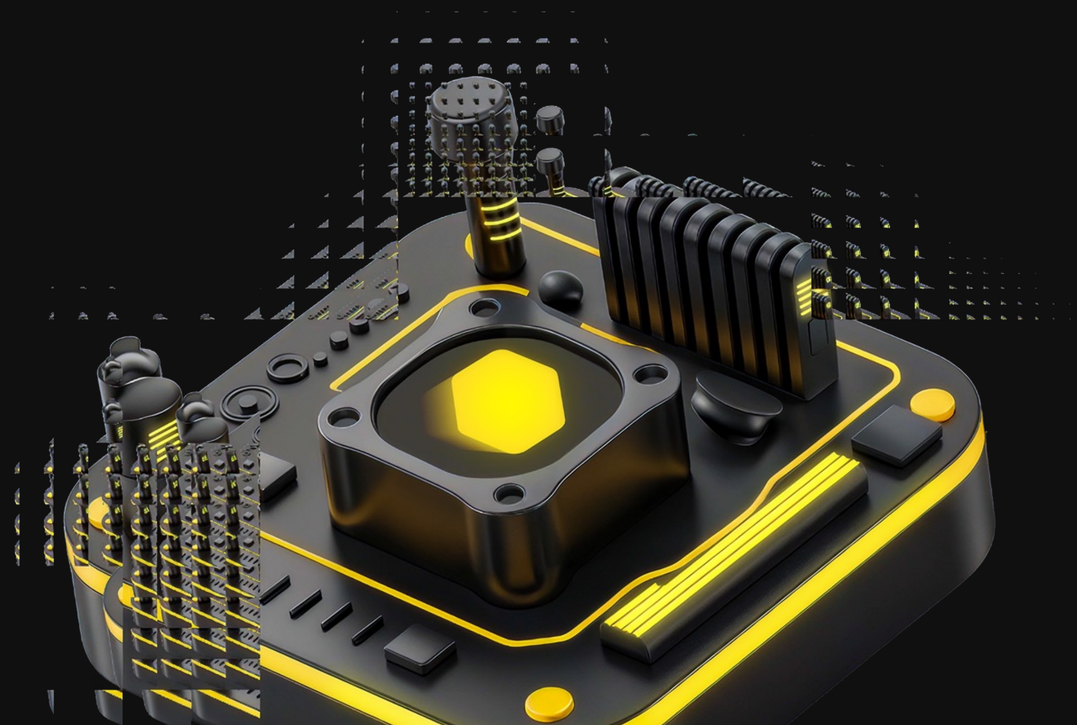
```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
 auto cluster = dependencies.pg->GetCluster();
 auto trx = cluster->Begin(storages::postgres::ClusterHostType::kMaster);

 const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
 auto row = psql::Execute(trx, statement, request.id)[0];
 if (!row["ok"].As<bool>()) {
   LOG_DEBUG() << request.id << " is not OK of " << GetSomeInfoFromDb();
   return Response400();
 }

 psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
 trx.Commit();

 return Response200{row["baz"].As<std::string>()};
}
```

GC?



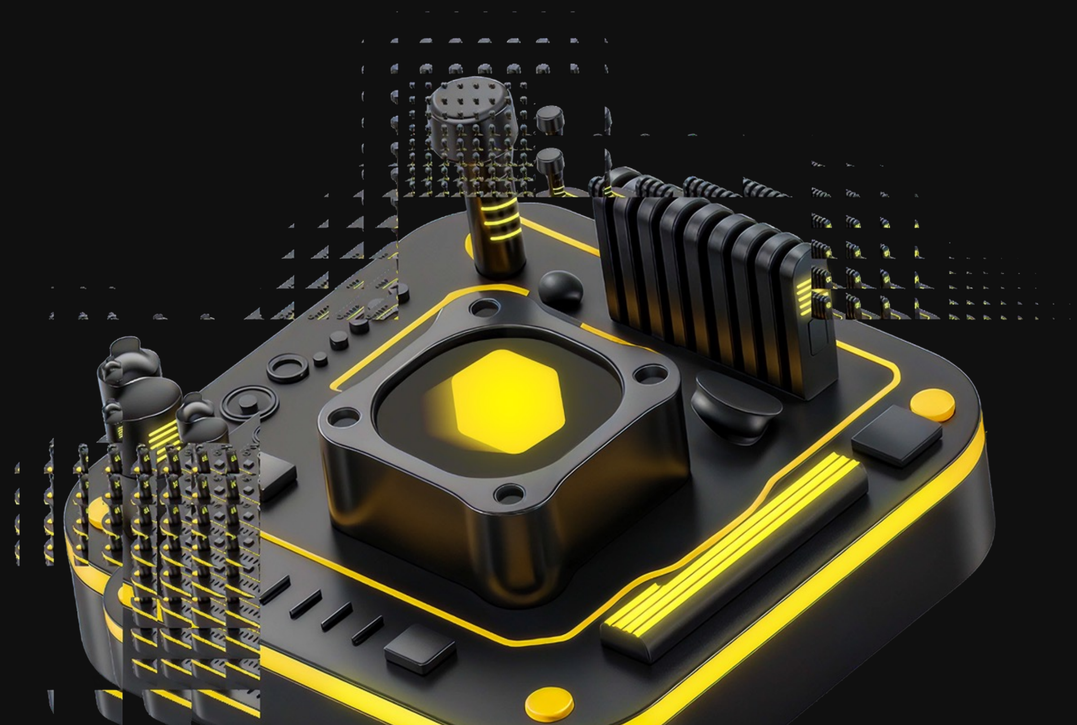
garbage collector

```
}
```

garbage collector

```
userver::rcu::Variable<T>
```

reflection or codegen?



reflection or codegen

```
 userver_add_sql_library(
    ${PROJECT_NAME}_sql
    NAMESPACE real_medium
    OUTPUT_DIR ${CMAKE_CURRENT_BINARY_DIR}
    SQL_FILES src/db/*.sql
)

file(GLOB_RECURSE SCHEMAS ${CMAKE_CURRENT_SOURCE_DIR}/docs/*.yaml)
userver_target_generate_chaotic(${PROJECT_NAME}-chgen
    LAYOUT "/components/schemas/([^/]*)/=real_medium::handlers::{0}"
    GENERATE_SERIALIZERS
    OUTPUT_DIR ${CMAKE_CURRENT_BINARY_DIR}/src
    SCHEMAS ${SCHEMAS}
    RELATIVE_TO ${CMAKE_CURRENT_SOURCE_DIR}
)
```

reflection or codegen

```
 #include <boost/pfr/core.hpp>

struct sample {
    char c;
    float f;
};

sample var{};
boost::pfr::get<1>(var) = 42.01f;
boost::pfr::get<char>(var) = 'A';

std::cout << var.c << var.f; // A 42.01
```

что мы ещё добавили

01 Асинхронное логирование

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

03 Примитивы синхронизаций

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

03 Примитивы синхронизаций

04 Драйвера баз данных

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

03 Примитивы синхронизаций

04 Драйвера баз данных

05 Юнит тесты

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

03 Примитивы синхронизаций

04 Драйвера баз данных

05 Юнит тесты

06 Документация

что мы ещё добавили

01 Асинхронное логирование

02 Метрики

03 Примитивы синхронизаций

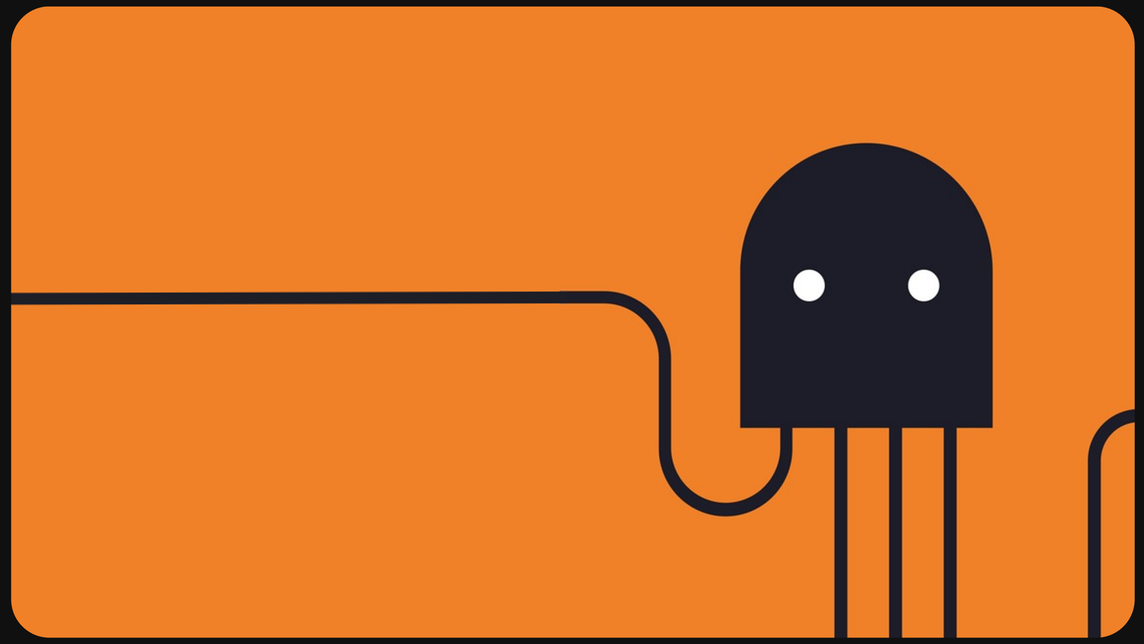
04 Драйвера баз данных

05 Юнит тесты

06 Документация

07 Отладочные скрипты

userver

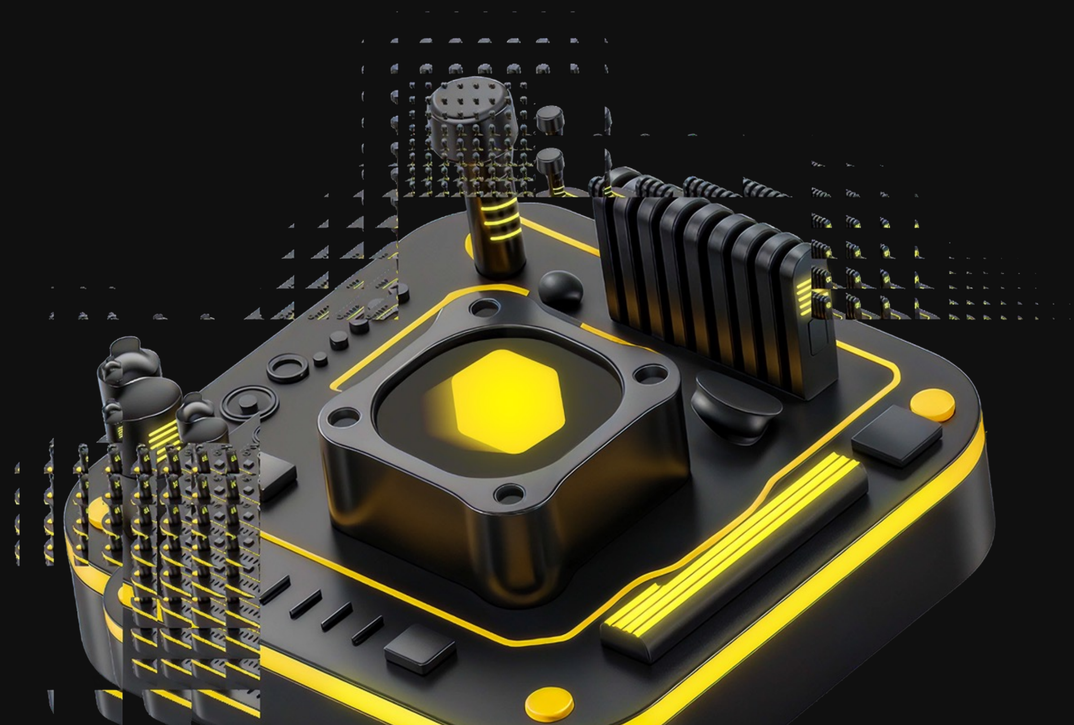


в итоге

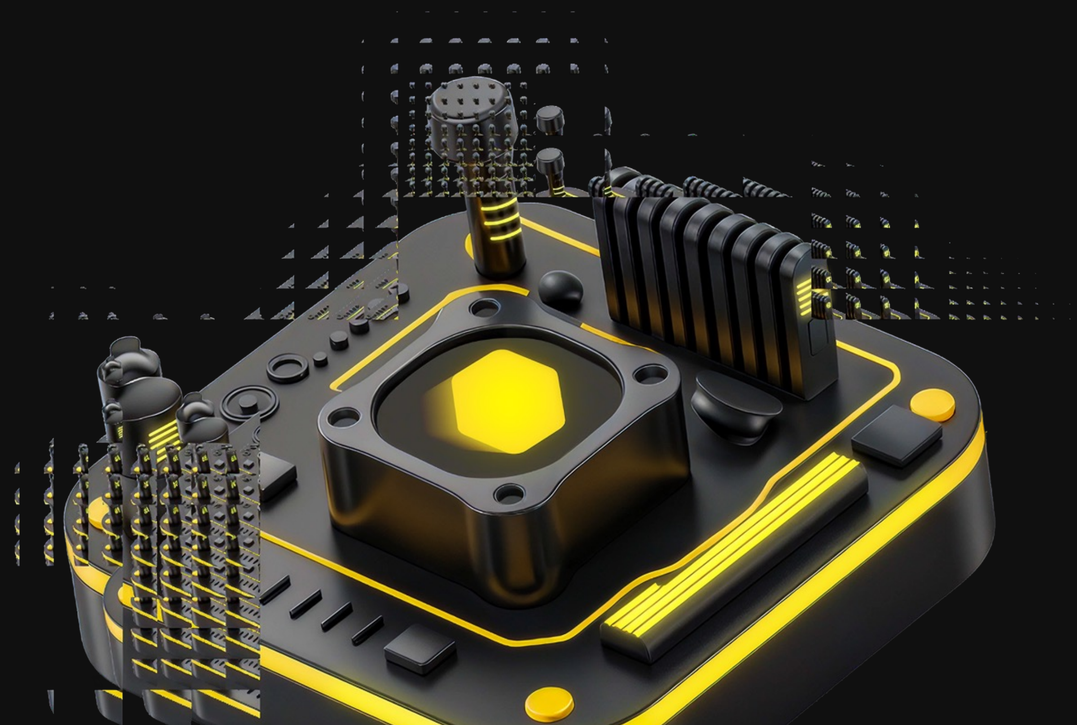
почти Go

притом с генериками и шаблонами

достаточно ли этого для прода?



делаем троллейбус «блестящим»



01 Теневые реплики

что мы ещё добавили

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

06 Консистентность по логам, трейсам и метрикам

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

06 Консистентность по логам, трейсам и метрикам

07 Исключения :)

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

06 Консистентность по логам, трейсам и метрикам

07 Исключения :)

08 Congestion Control

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

06 Консистентность по логам, трейсам и метрикам

07 Исключения :)

08 Congestion Control

09 Оптимизации!

что мы ещё добавили

01 Теневые реплики

02 Динамические конфиги

03 Функциональные тесты

04 Deadline Propagation

05 Таймауты и ретраи

06 Консистентность по логам, трейсам и метрикам

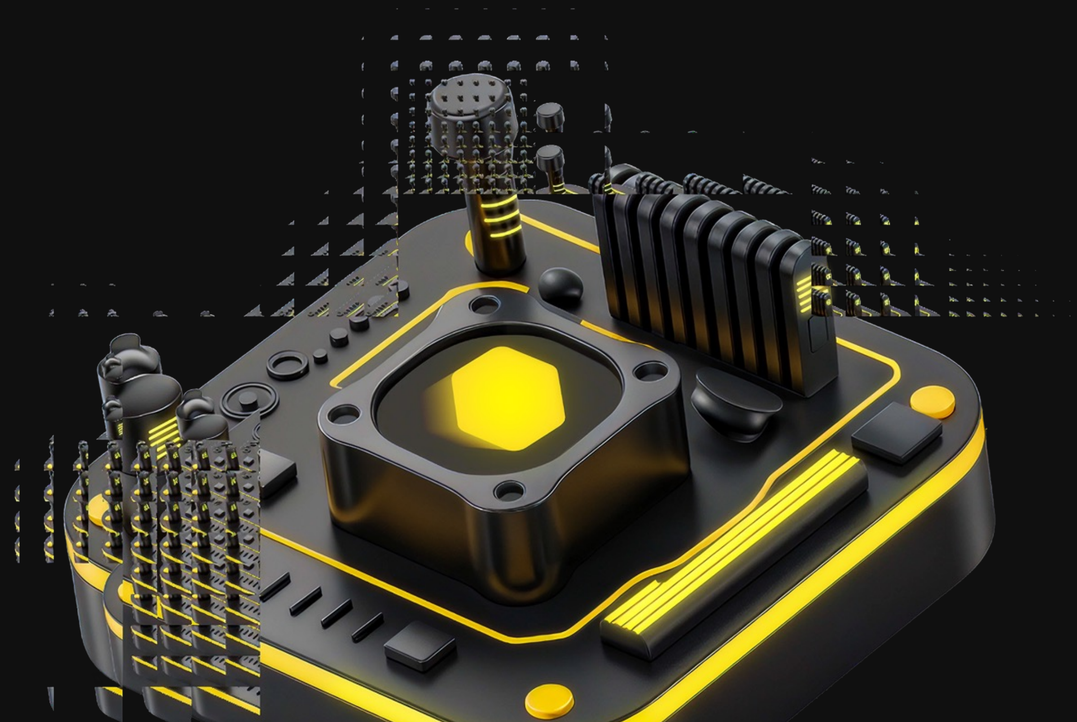
07 Исключения :)

08 Congestion Control

09 Оптимизации!

10 Инструменты профилирования

ВЫВОДЫ



ГОРОДСКИЕ СЕРВИСЫ

Яндекс Techplatform meetup

готов ответить на вопросы

Антон Полухин

Эксперт разработчик C++