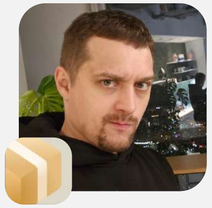
Доставка



Павел Сухов
Ведущий разработчик

Полезные трюки C++

Примерный план

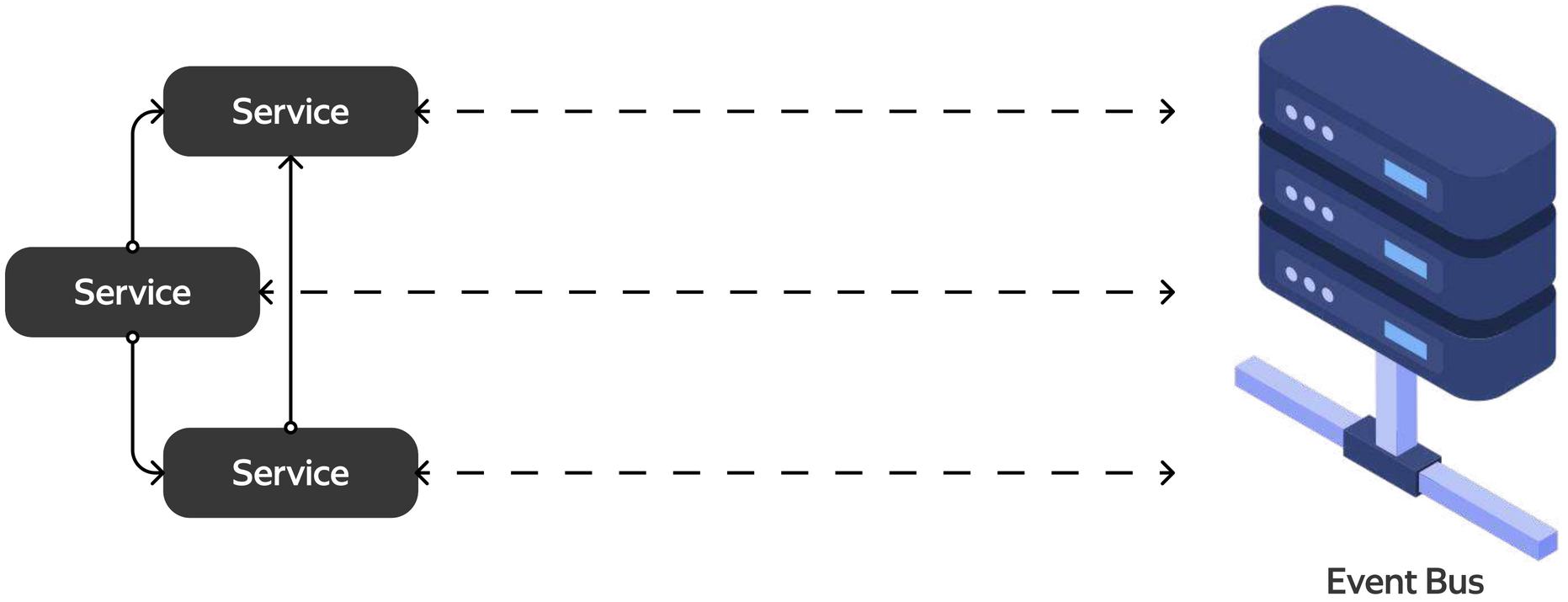
Часть 01

Инфраструктура 

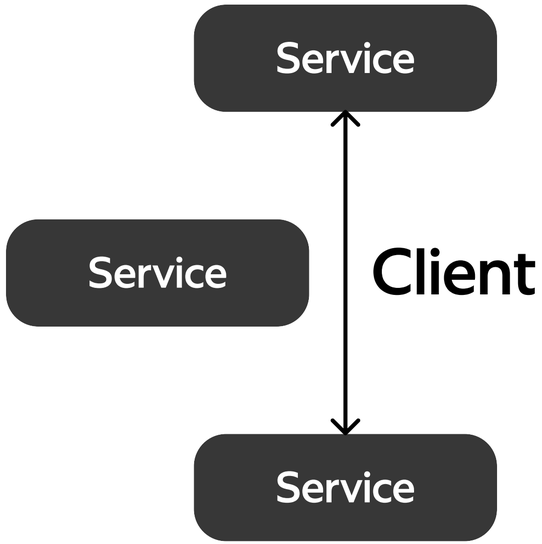
 Доставки

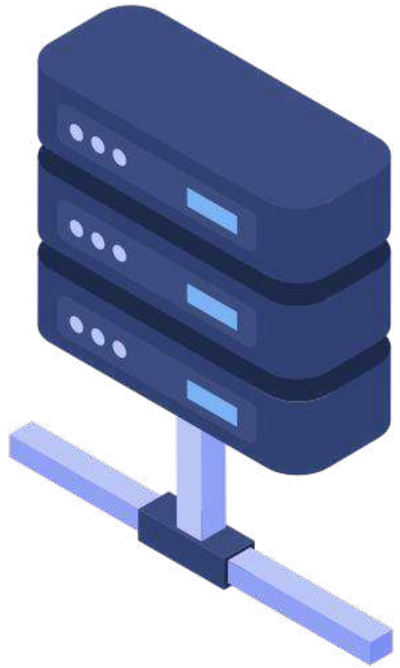
Микросервисная инфраструктура

Микросервисная инфраструктура



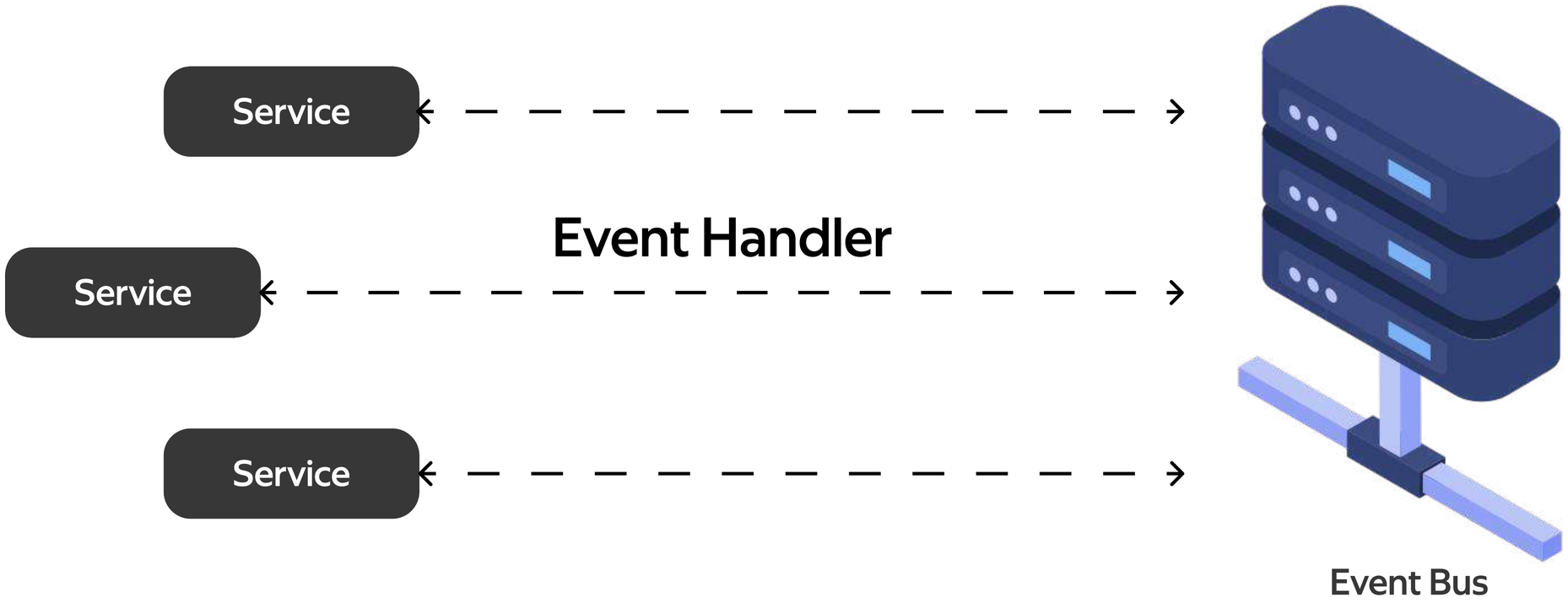
Микросервисная инфраструктура



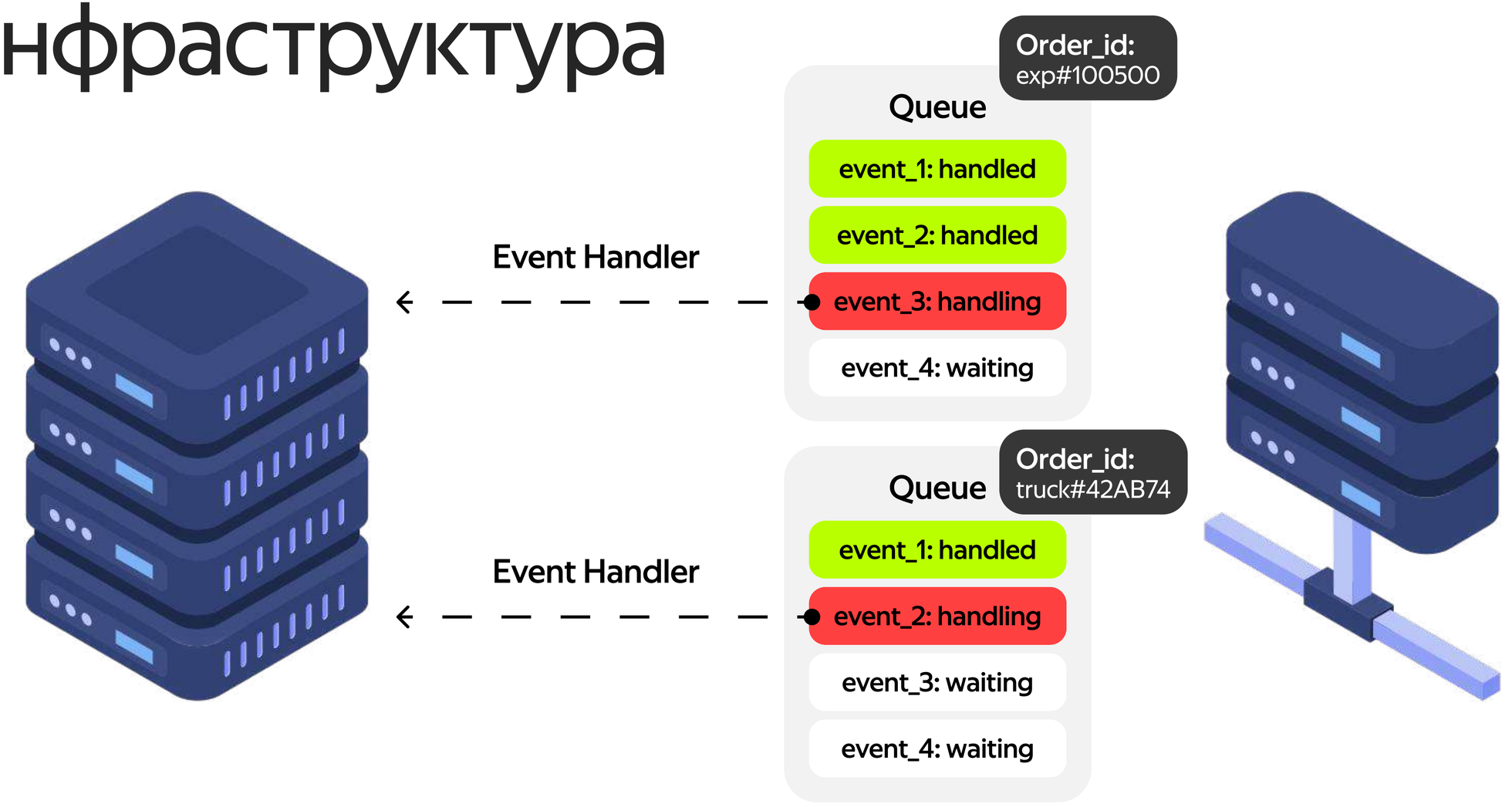


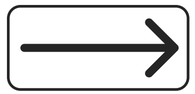
Event Bus

Микросервисная инфраструктура



Микросервисная инфраструктура





Кодогенерация OpenAPI

Кодогенерация OpenAPI

```
OrderCreatedEvent:
    type: object
    required:
    - event_kind
    - order_id
    properties:
    event_kind:
        type: string
        enum: [OrderCreatedEvent]
    order_id:
        type: string
    initial_price:
        type: integer
        format: int64
    contract_id:
        type: string
```

Кодогенерация OpenAPI

```yaml
OrderCreatedEvent:
    type: object
    required:
    - event_kind
    - order_id
    properties:
    event_kind:
        type: string
        enum: [OrderCreatedEvent]
    order_id:
        type: string
    initial_price:
        type: integer
        format: int64
    contract_id:
        type: string
```

```cpp
enum class OrderCreatedEventKind { OrderCreatedEvent };

struct OrderCreatedEvent
{
    OrderCreatedEventKind event_kind =
        OrderCreatedEventKind::OrderCreatedEvent;

    std::string order_id;


    std::optional<int> initial_price;
    std::optional<std::string> contract_id;
};
```

Кодогенерация OpenAPI

```yaml
OrderCreatedEvent:
    type: object
    required:
    - event_kind
    - order_id
    properties:
    event_kind:
        type: string
        enum: [OrderCreatedEvent]
    order_id:
        type: string
    initial_price:
        type: integer
        format: int64
    contract_id:
        type: string
```

```cpp
enum class OrderCreatedEventKind { OrderCreatedEvent };

struct OrderCreatedEvent
{
    OrderCreatedEventKind event_kind =
        OrderCreatedEventKind::OrderCreatedEvent;

    std::string order_id;


    std::optional<int> initial_price;
    std::optional<std::string> contract_id;
};

std::string ToString(OrderCreatedEventKind kind);
template < >
OrderCreatedEvent FromJson<OrderCreatedEvent>(const json& input);
template < >
json ToJson(const OrderCreatedEventKind& input);
```

Кодогенерация OpenAPI

```yaml
paths:
   /events/cargo-finance/v1/process:
       post:
           description: >
               Обработка эвента
           parameters:
             - $ref: '#/components/parameters/OrderId'
           requestBody:
               required: true
               content:
                   application/json:
                       schema:
                           $ref: '#/components/schemas/Request'
```

API Definition

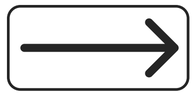
Кодогенерация OpenAPI

```yaml
paths:
   /events/cargo-finance/v1/process:
       post:
           description: >
               Обработка эвента
           parameters:
             - $ref: '#/components/parameters/OrderId'
           requestBody:
               required: true
               content:
                   application/json:
                       schema:
                           $ref: '#/components/schemas/Request'
```

API Definition

```yaml
libraries:
 - yandex-userver-core
 # ...
clients:
 - billing-orders
 - billing-replication
 - cargo-claims
 - cargo-orders
 - cargo-corp
 - cargo-pricing
 # ...
```

Service.yaml



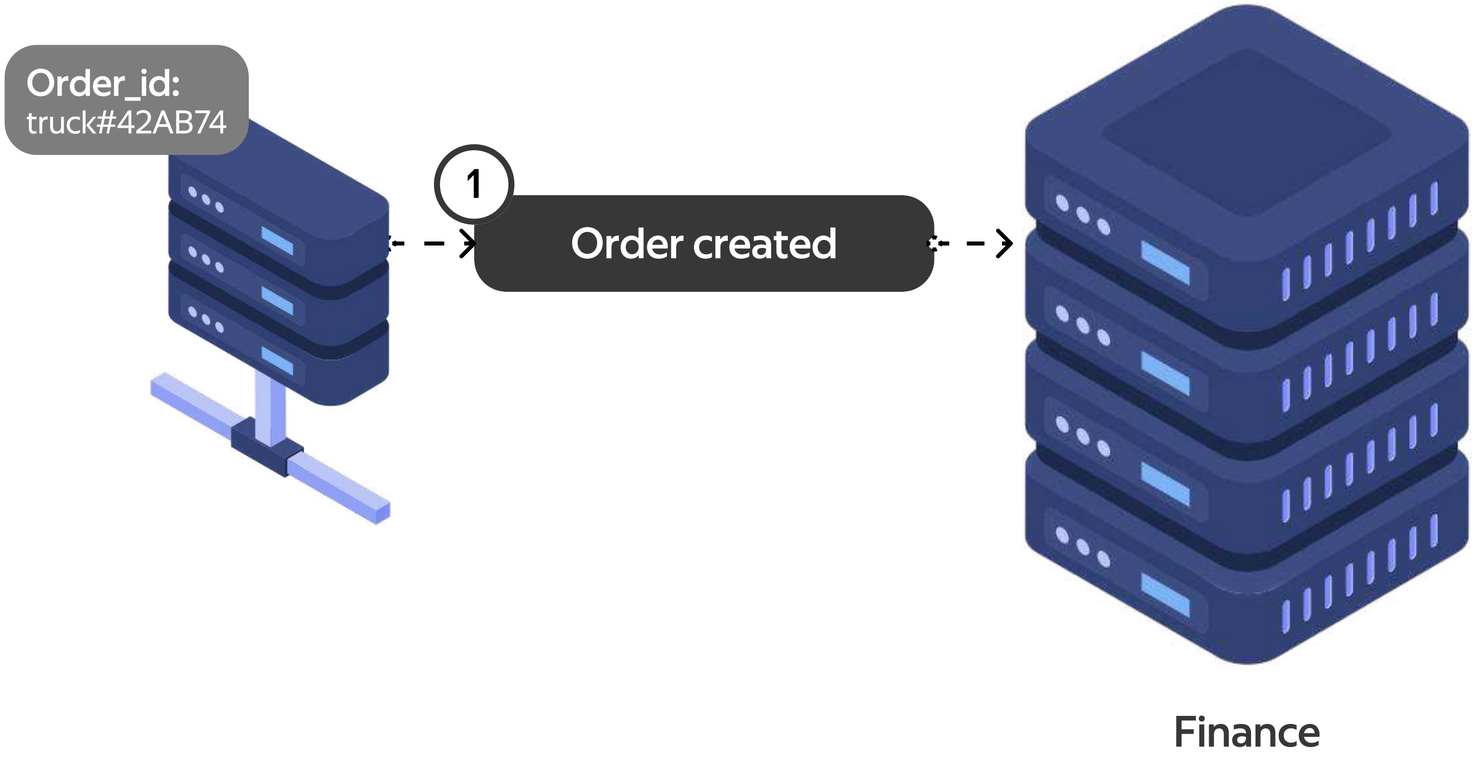
Сервис финансов

Сервис финансов

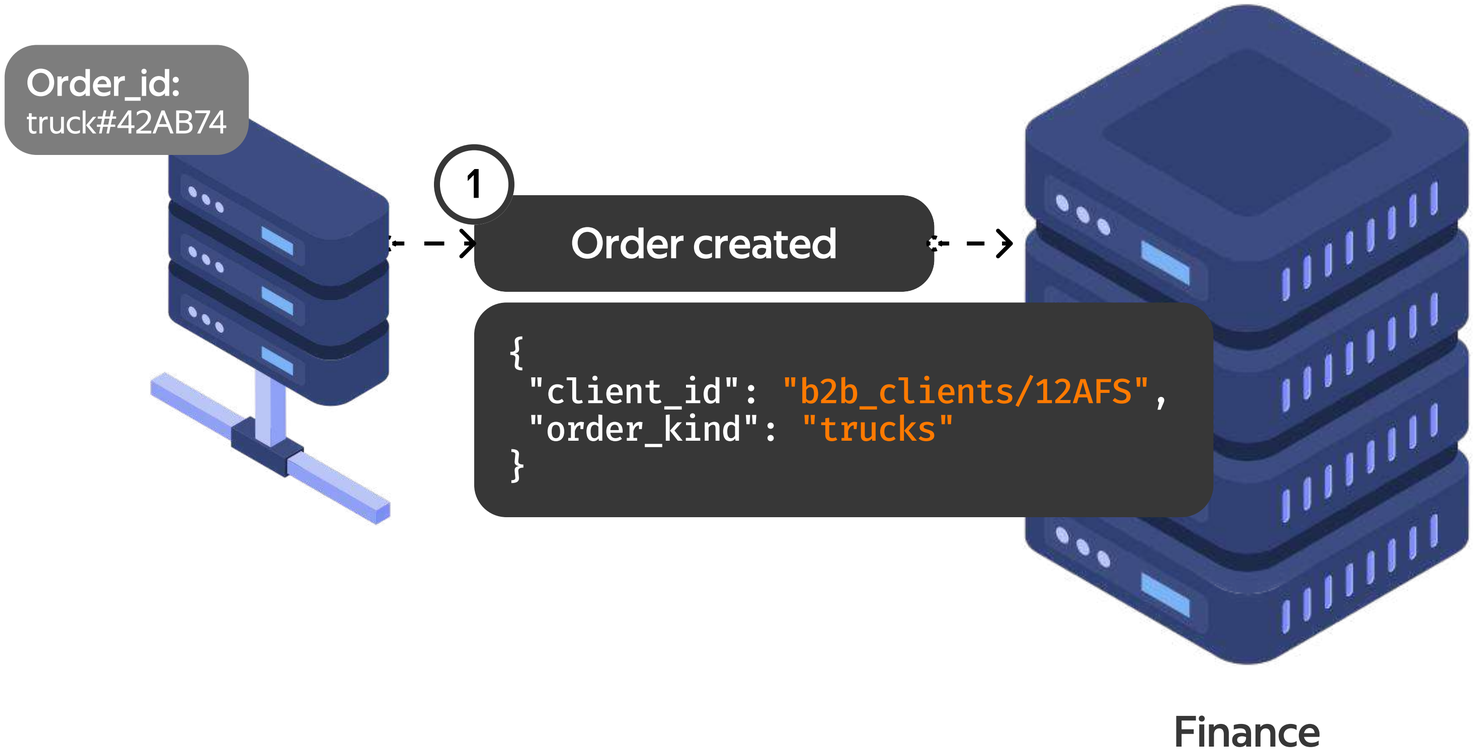
Сервис финансов обеспечивает
финансовое сопровождение цикла заказа

Finance

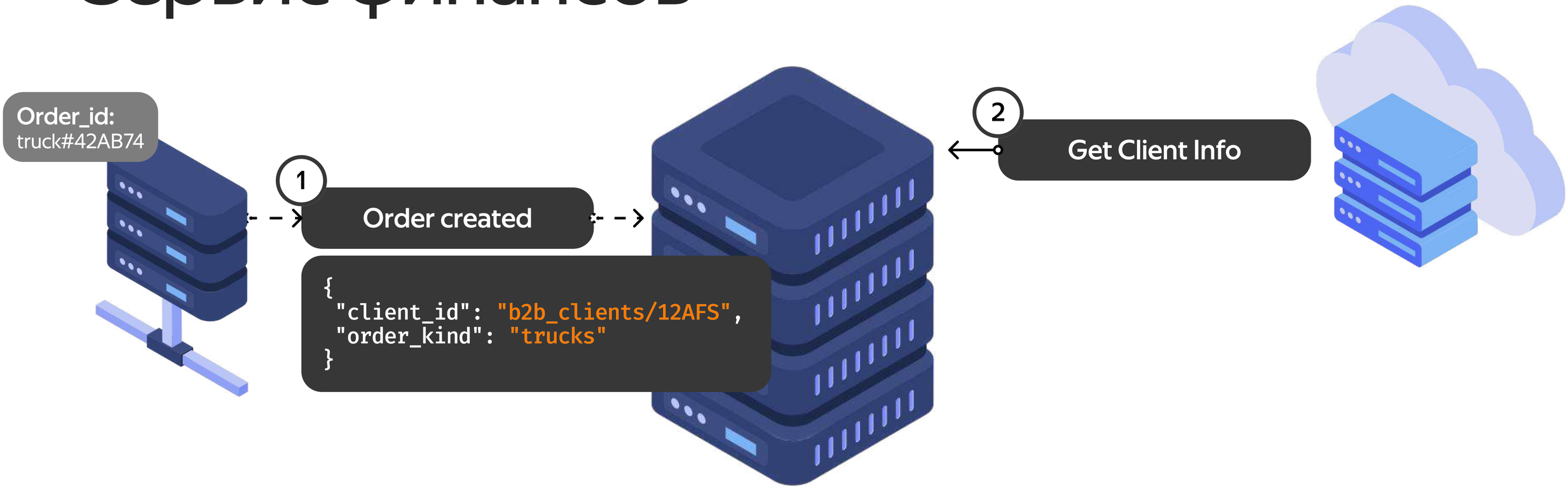
Сервис финансов



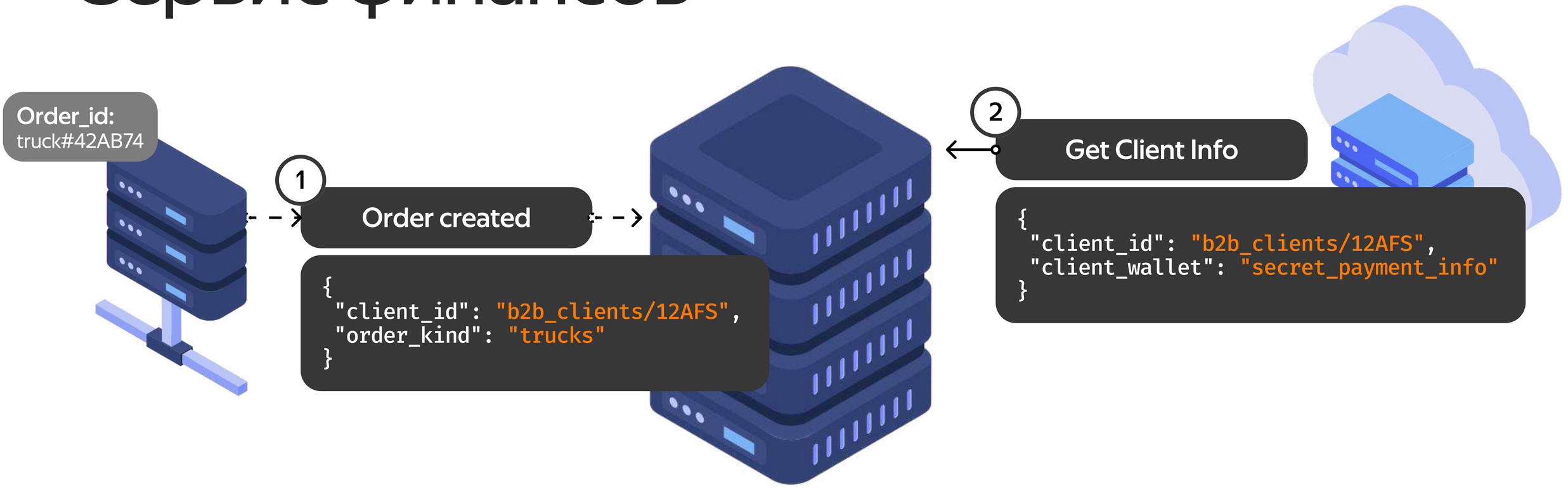
Сервис финансов



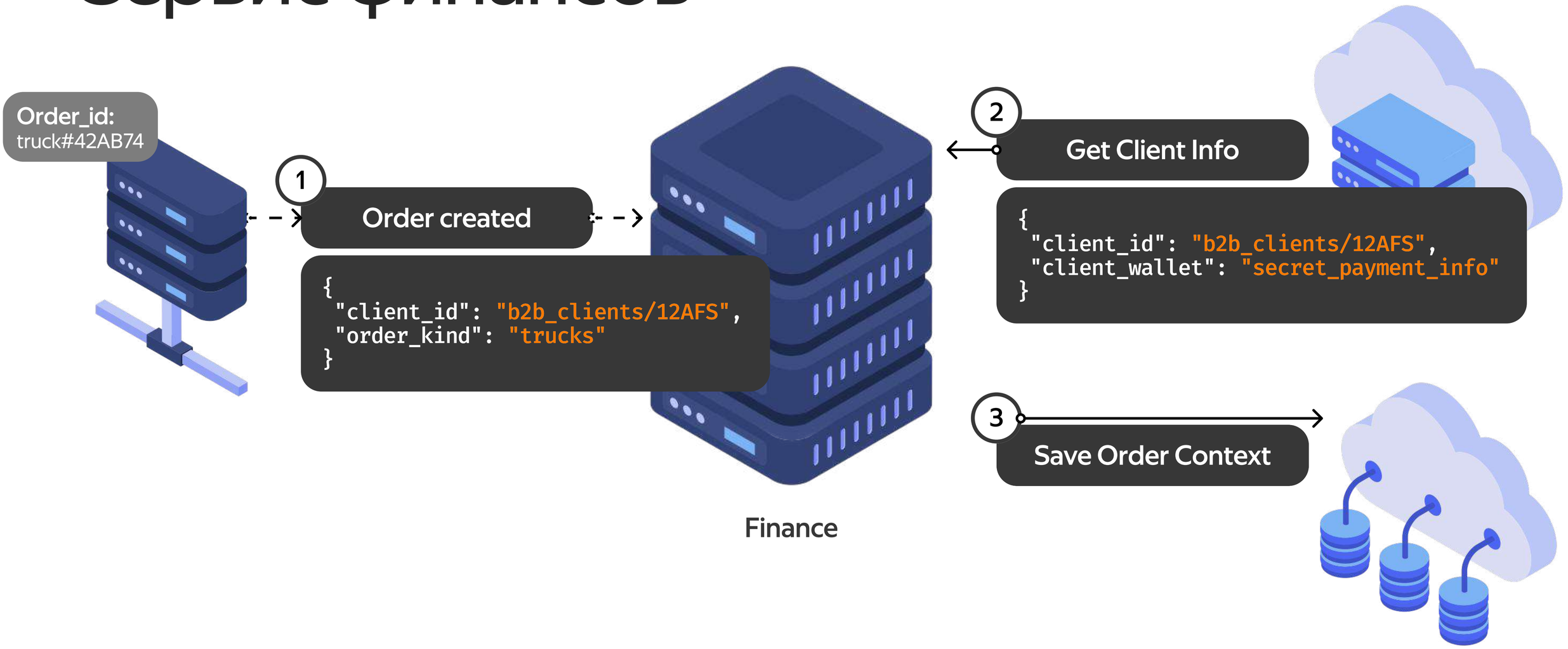
Сервис финансов



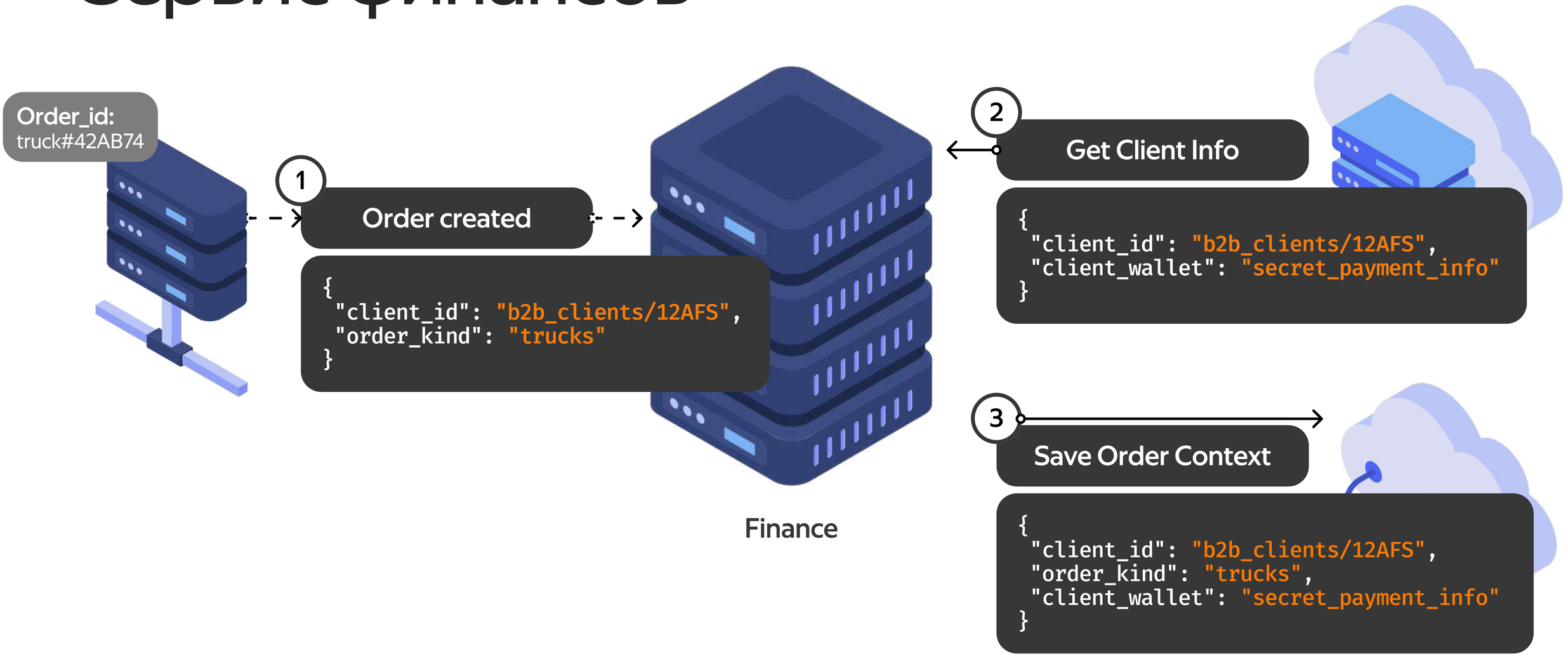
Сервис финансов



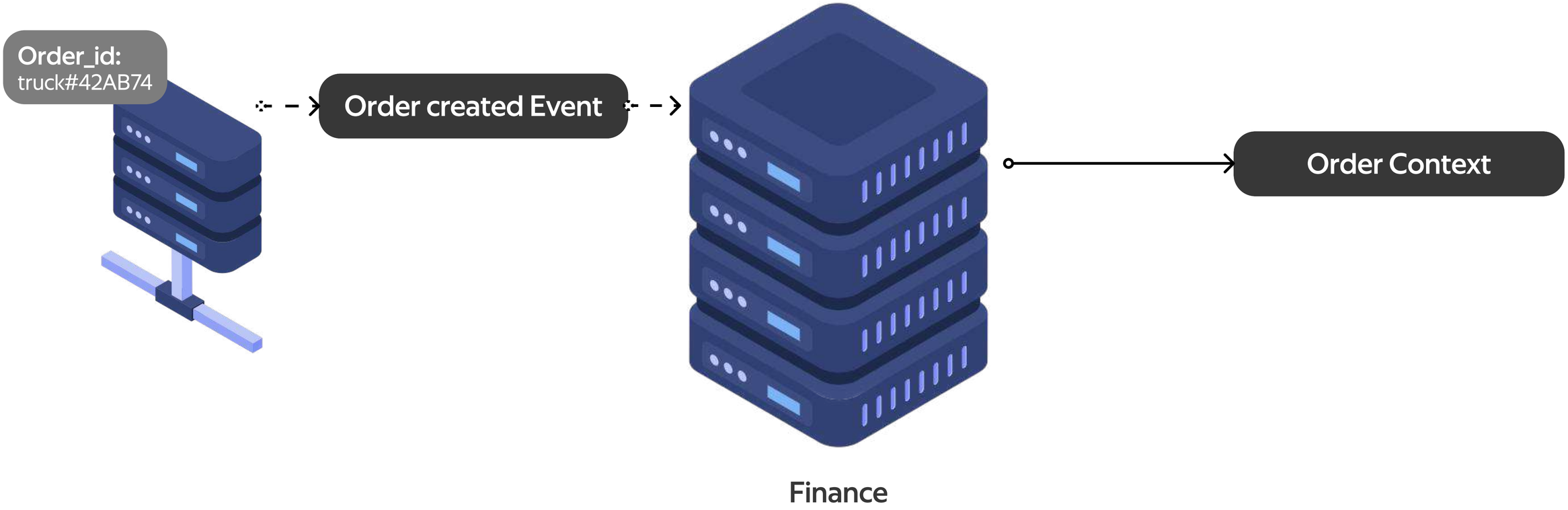
Сервис финансов



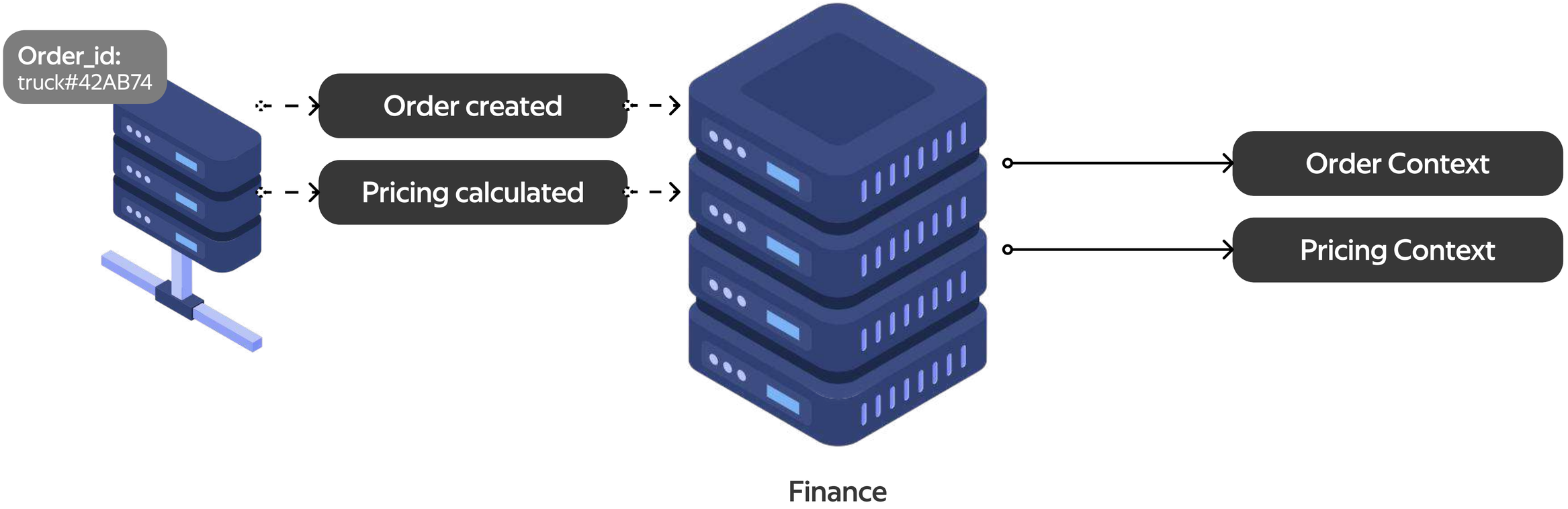
Сервис финансов



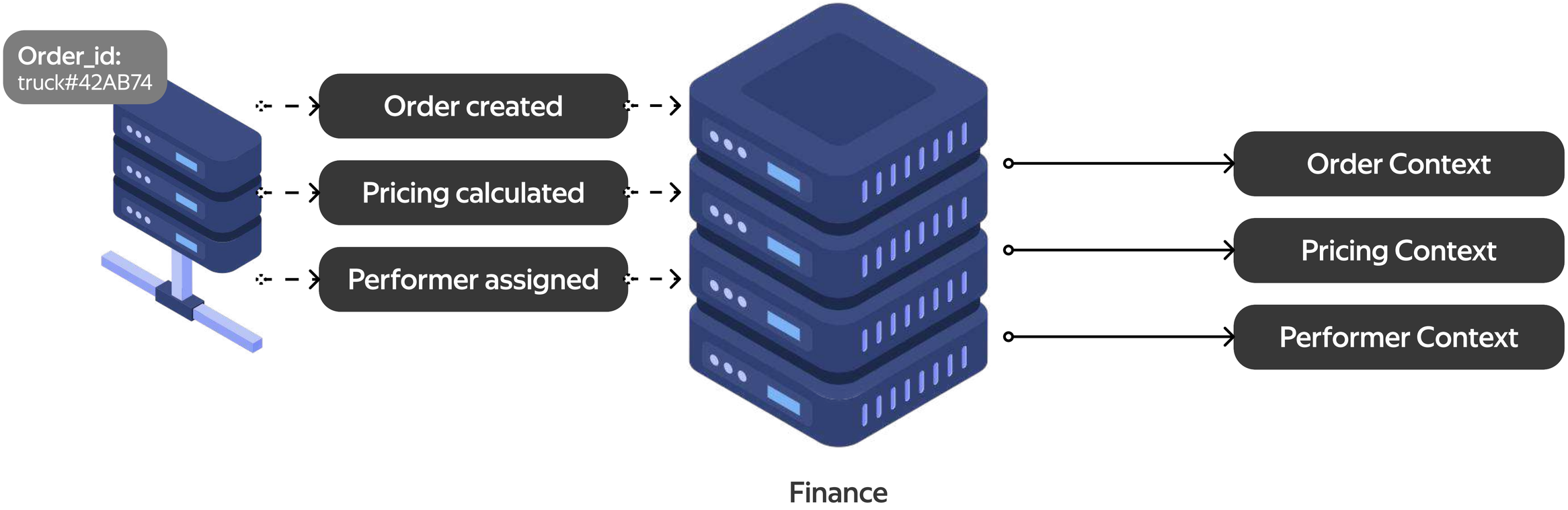
Сервис финансов



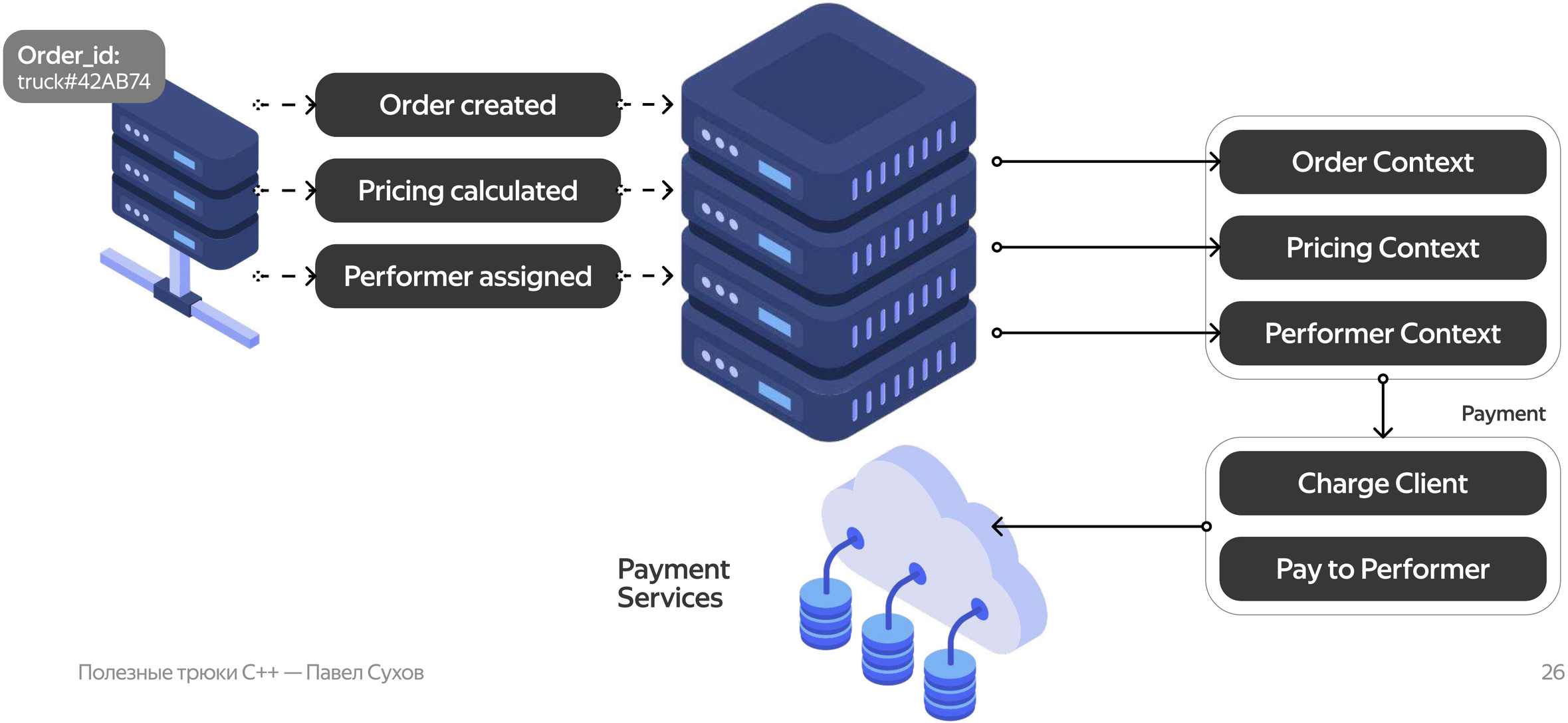
Сервис финансов



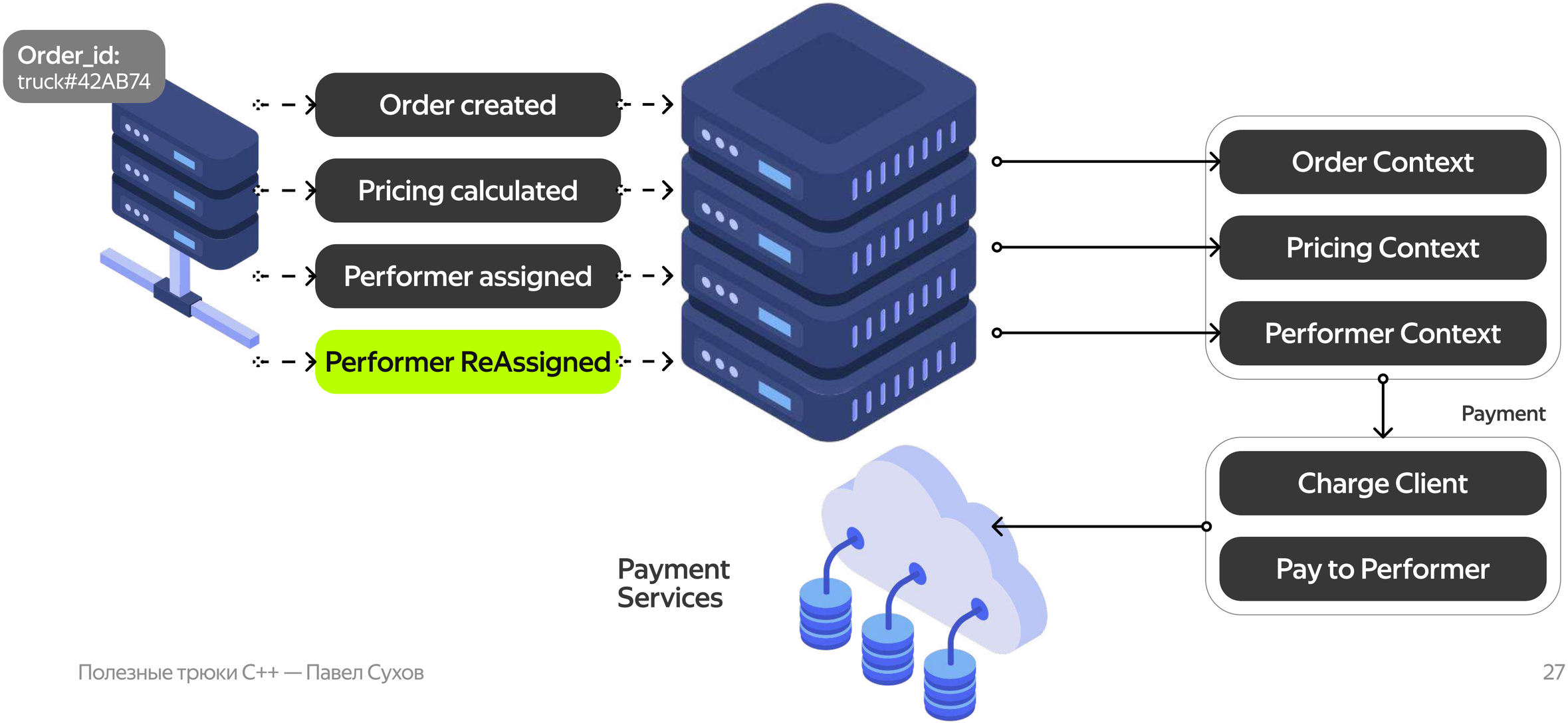
Сервис финансов



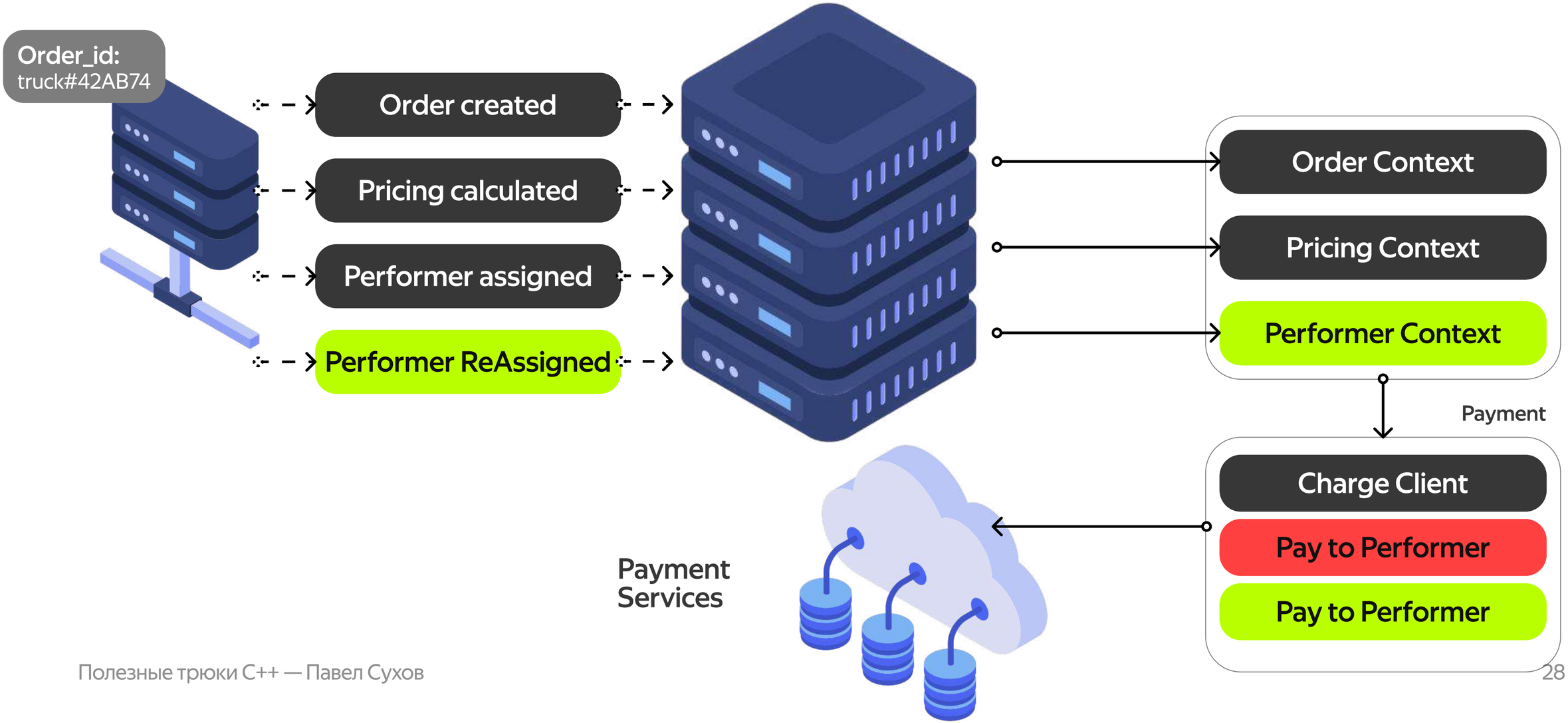
Сервис финансов



Сервис финансов

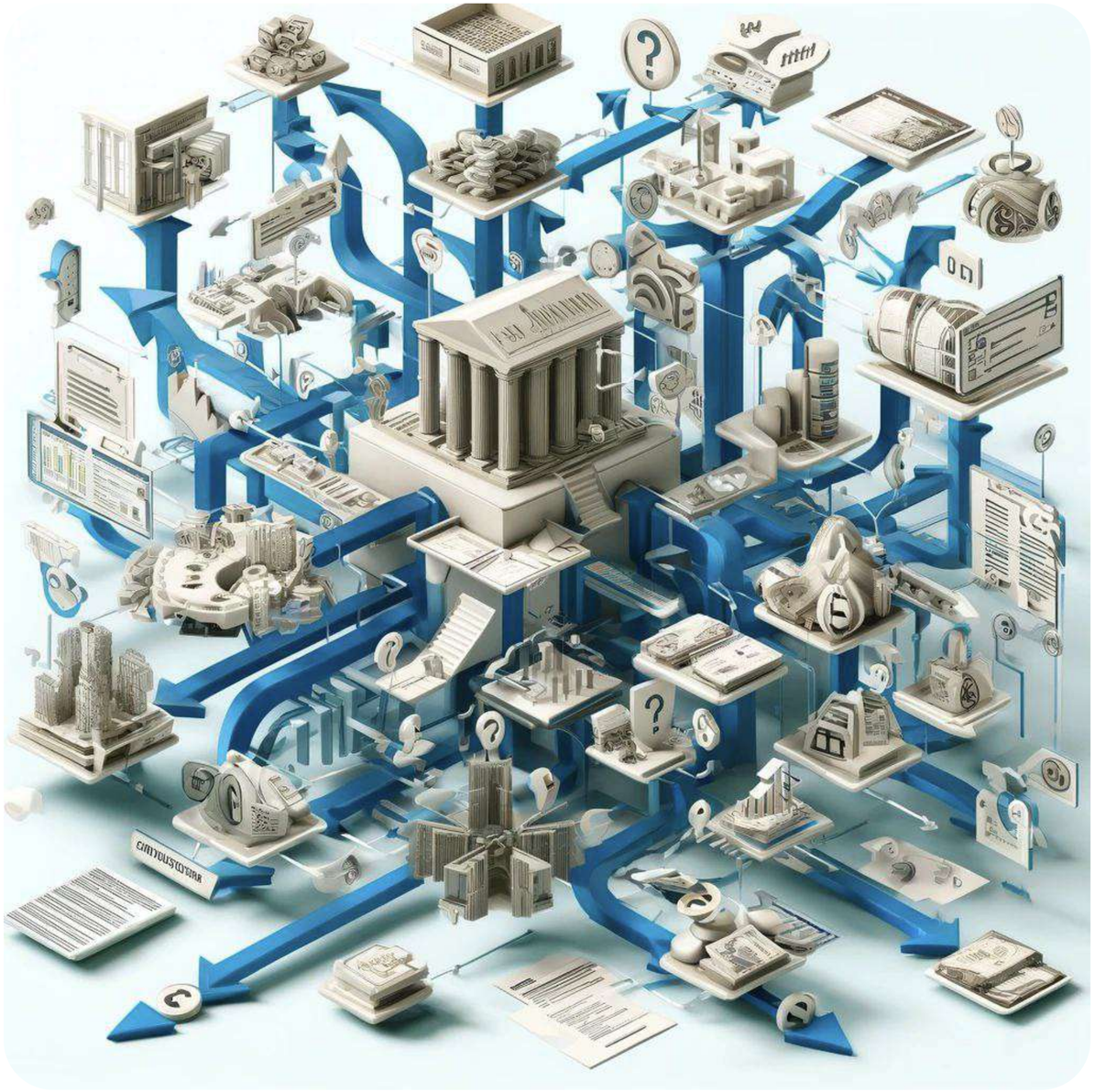


Сервис финансов



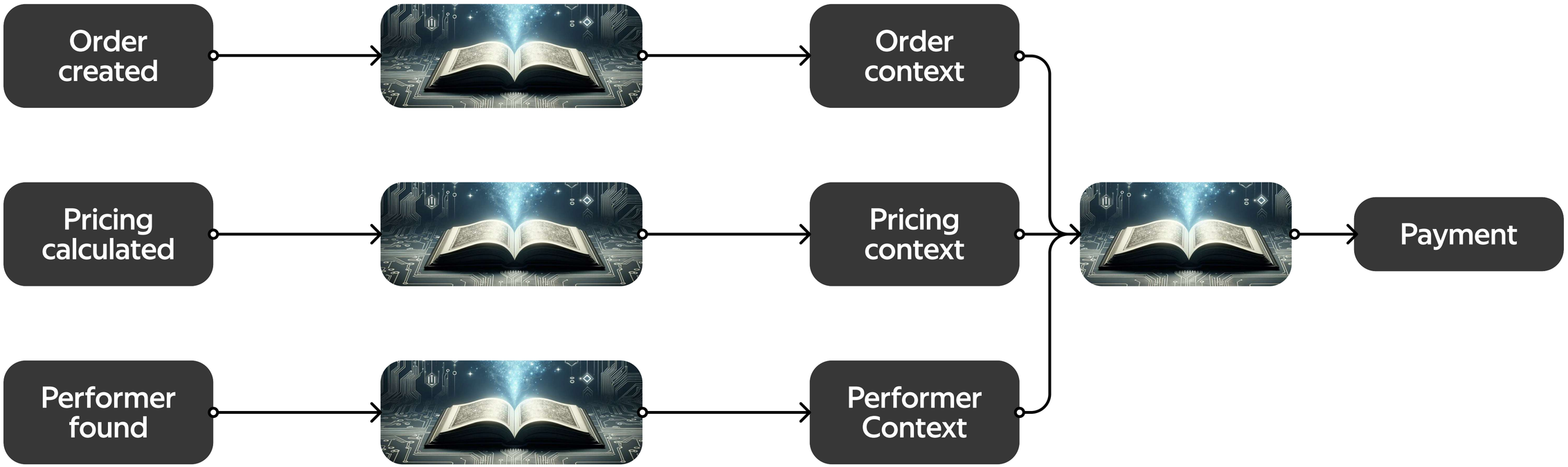
Это в теории...
А на практике?

Практика

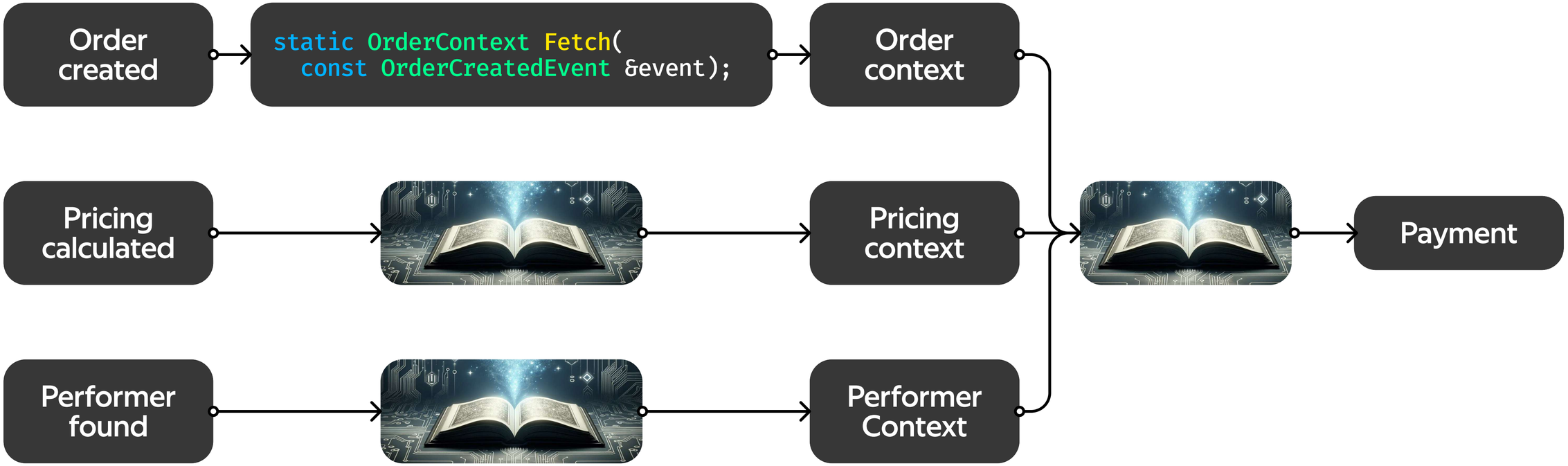


Интерфейс в коде

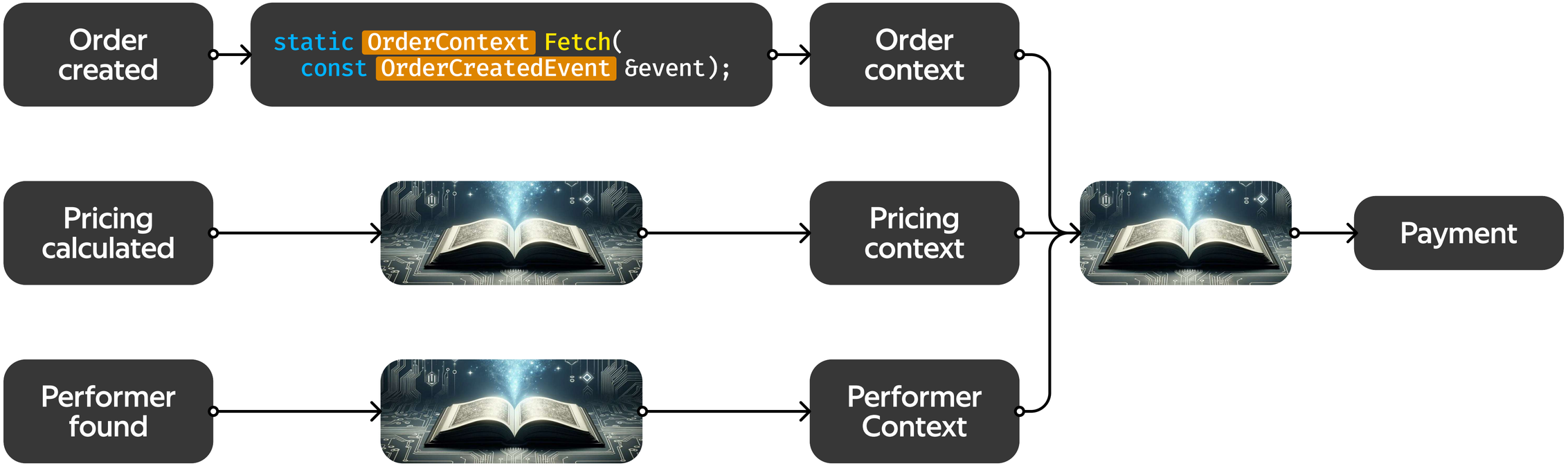
Интерфейс в коде



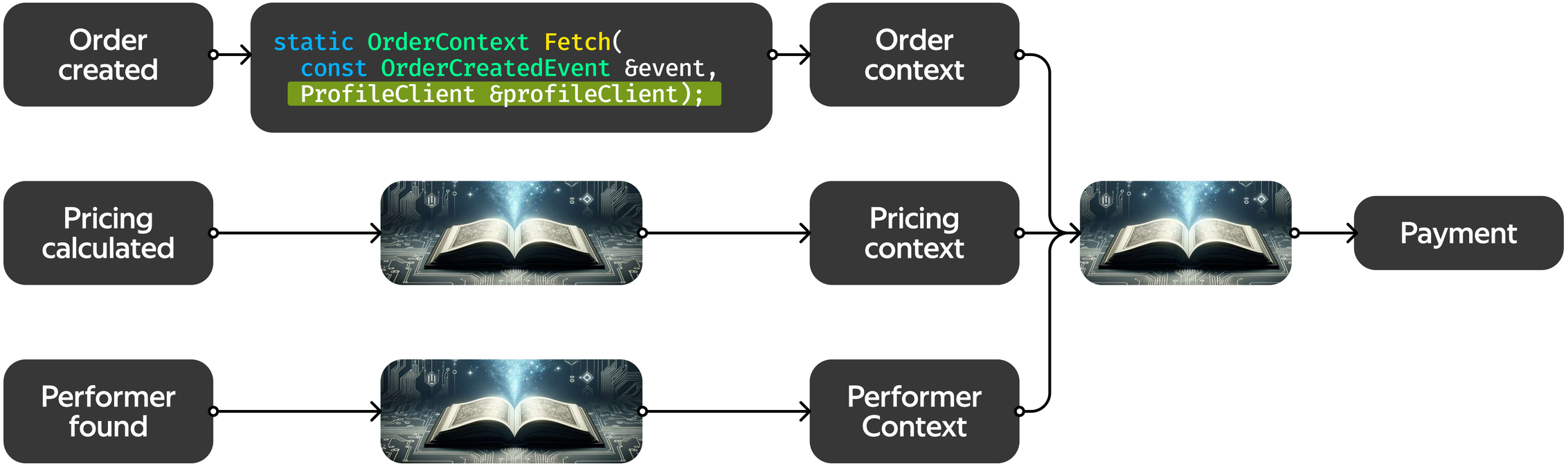
Интерфейс в коде



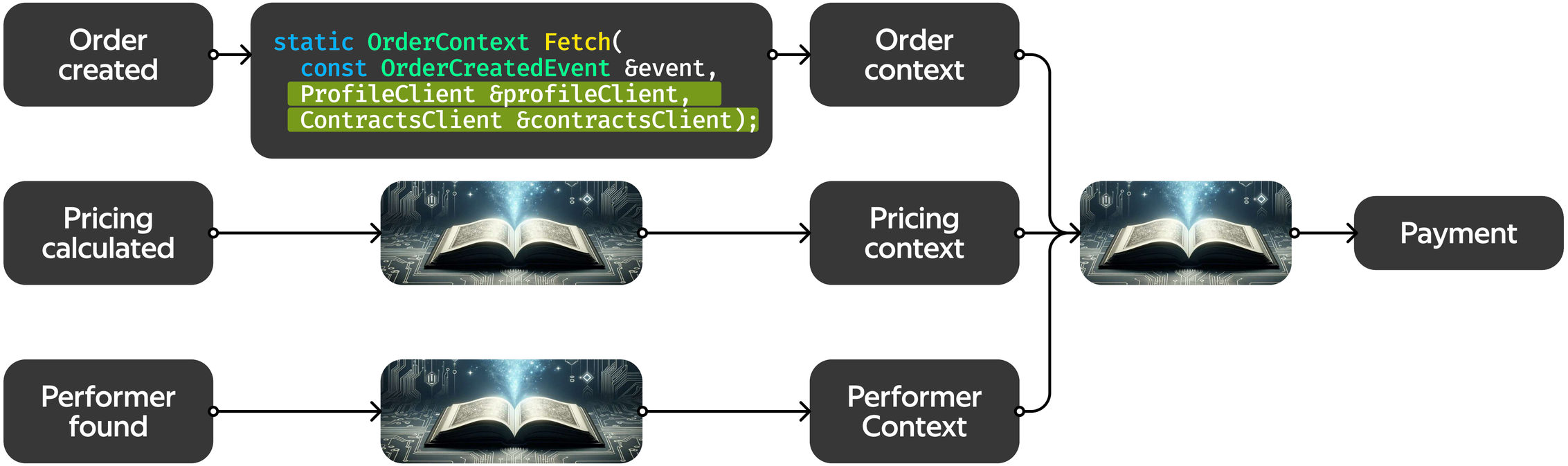
Интерфейс в коде



Интерфейс в коде



Интерфейс в коде



Интерфейс в коде

Order created

```
static OrderContext Fetch(
  const OrderCreatedEvent &event,
  ProfileClient &profileClient,
  ContractsClient &contractsClient);
```

Order context

Pricing calculated

```
static PricingContext Fetch(
  const PriceCalculatedEvent &event,
  PricingClient &pricingClient);
```

Pricing context

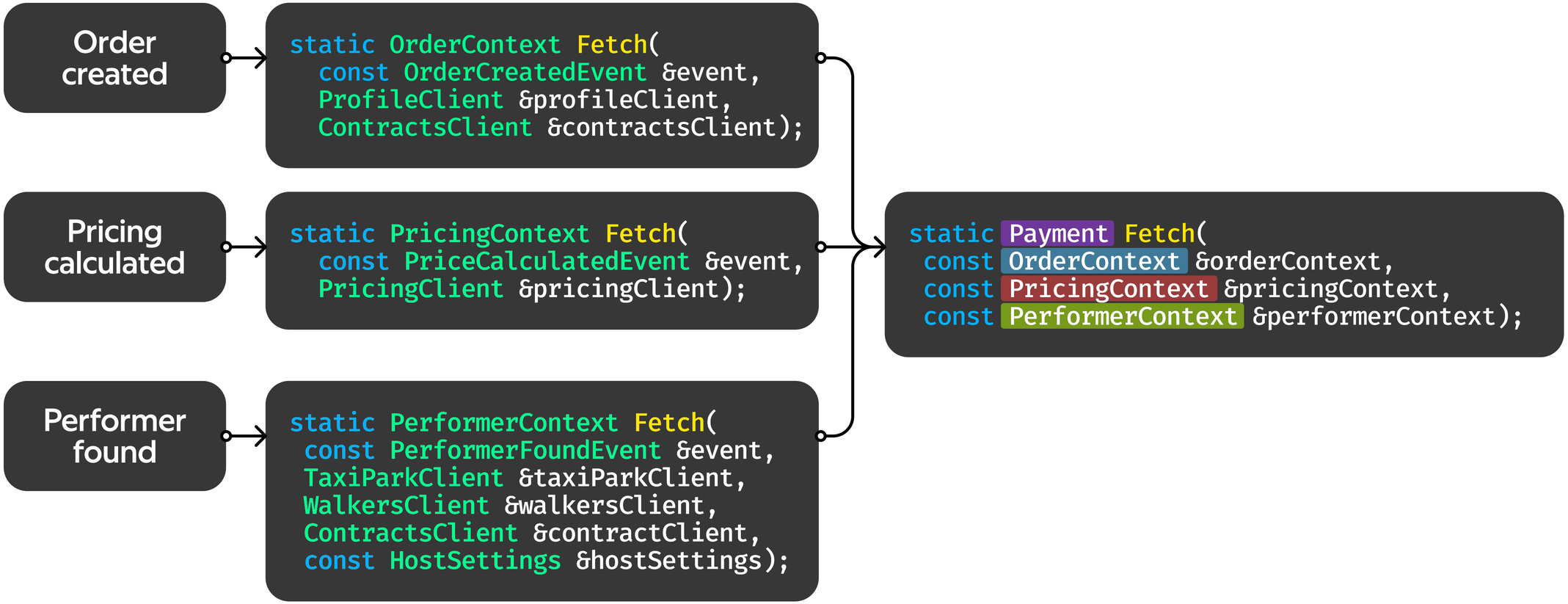
Performer found

```
static PerformerContext Fetch(
 const PerformerFoundEvent &event,
 TaxiParkClient &taxiParkClient,
 WalkersClient &walkersClient,
 ContractsClient &contractClient,
 const HostSettings &hostSettings);
```

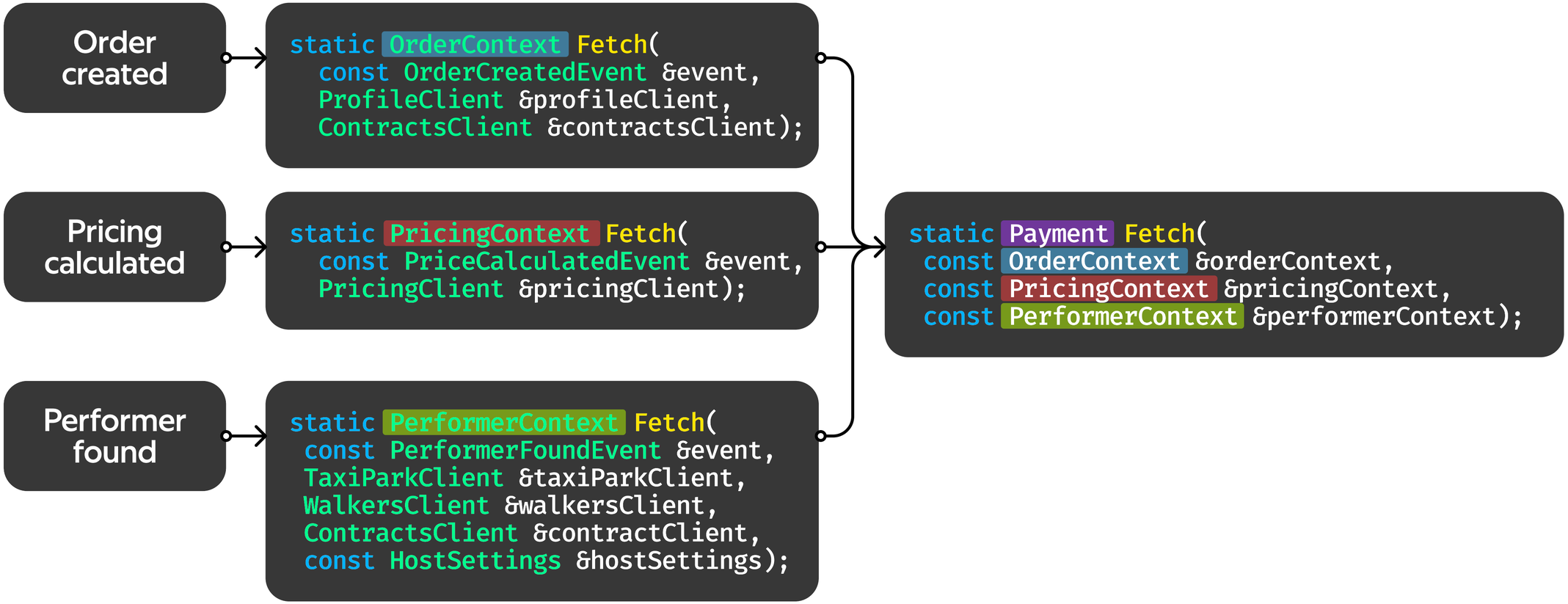
Performer Context

Payment

Интерфейс в коде



Интерфейс в коде



Интерфейс в коде

```
struct OrderCreatedEventFetcher
{
 static OrderContext Fetch(
     const OrderCreatedEvent &event,
     ProfileClient &profileClient,
     ContractsClient &contractsClient);
};

struct PriceCalculatedEventFetcher
{
 static PricingContext Fetch(
     const PriceCalculatedEvent &event,
     PricingClient &pricingClient);
};

struct PerformerFoundEventFetcher
{
 static PerformerContext Fetch(
     const PerformerFoundEvent &event,
     TaxiParkClient &taxiParkClient,
     WalkersClient &walkersClient,
     ContractsClient &contractClient,
     const HostSettings &hostSettings);
};
```

```
struct BuildPaymentFetcher
{
 static Payment Fetch(
     const OrderContext &orderContext,
     const PricingContext &pricingContext,
     const PerformerContext &performerContext);
};
```

Интерфейс в коде

```
struct OrderCreatedEventFetcher
{
 static OrderContext Fetch(
     const OrderCreatedEvent &event,
     ProfileClient &profileClient,
     ContractsClient &contractsClient);
};

struct PriceCalculatedEventFetcher
{
 static PricingContext Fetch(
     const PriceCalculatedEvent &event,
     PricingClient &pricingClient);
};

struct PerformerFoundEventFetcher
{
 static PerformerContext Fetch(
     const PerformerFoundEvent &event,
     TaxiParkClient &taxiParkClient,
     WalkersClient &walkersClient,
     ContractsClient &contractClient,
     const HostSettings &hostSettings);
};
```

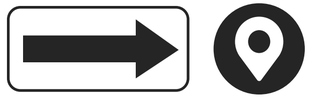
```
struct BuildPaymentFetcher
{
 static Payment Fetch(
     const OrderContext &orderContext,
     const PricingContext &pricingContext,
     const PerformerContext &performerContext);
};




using Pipeline = gate::Pipeline<
   fetchers::OrderCreatedEventFetcher,
   fetchers::PriceCalculatedEventFetcher,
   fetchers::PerformerFoundEventFetcher,
   fetchers::BuildPaymentFetcher>;
```

Интерфейс в коде

```
struct OrderCreatedEventFetcher
{
 static OrderCon
     const Orde
     ProfileCli
     ContractsC
};

struct PriceCal
{
 static Pricing
     const Pric
     PricingCli
};

struct Performe
{
 static PerformerContext Fetch(
     const PerformerFoundEvent &event,
     TaxiParkClient &taxiParkClient,
     WalkersClient &walkersClient,
     ContractsClient &contractClient,
     const HostSettings &hostSettings);
};
```

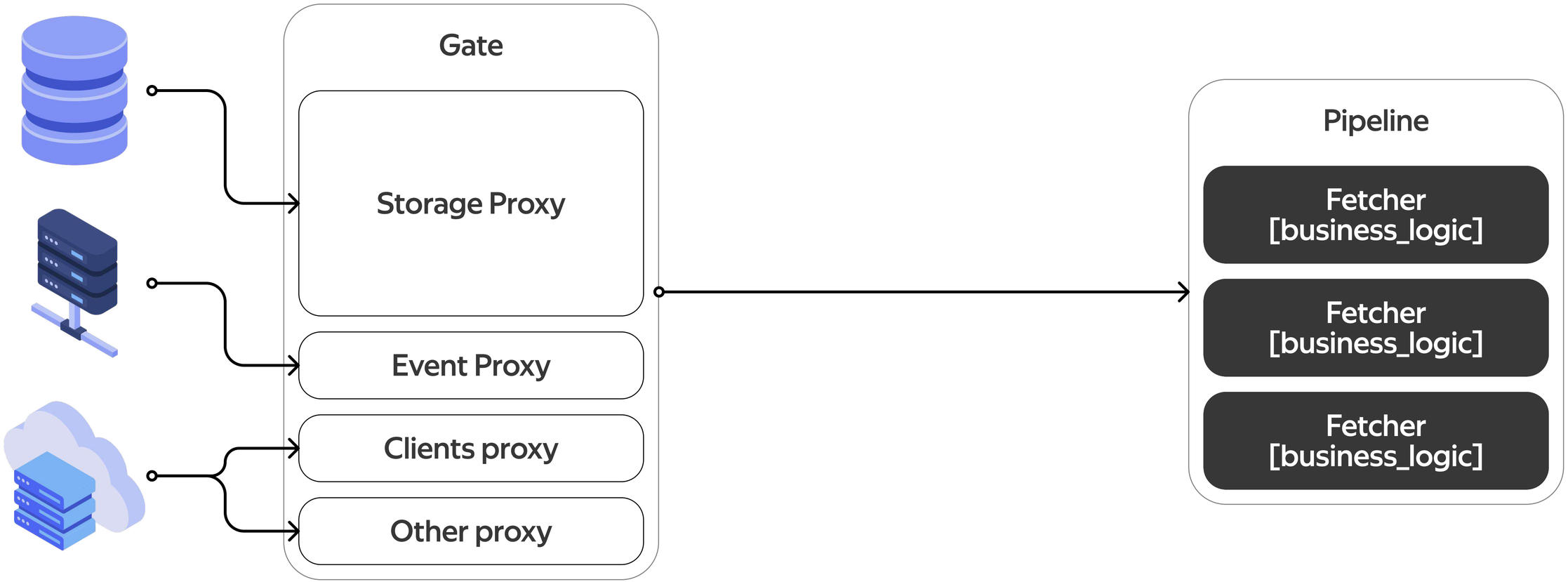
```
struct BuildPaymentFetcher
{
                                    xt,
                                    ontext);
fetchers::PriceCalculatedEventFetcher,
fetchers::PerformerFoundEventFetcher,
fetchers::BuildPaymentFetcher>;
```

Максимальная изоляция **бизнес-логики** от **инфраструктуры**

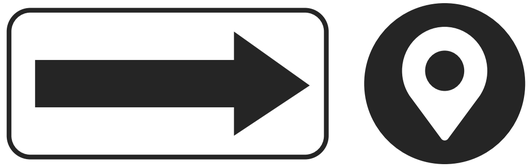


Общая схема решения

Общая схема решения



Часть 02

Storage 
Proxy 

Немного кода

```cpp
using id_index_t = std::size_t;
namespace details
{
  struct type_index final
  {
    [[nodiscard]] static id_index_t next() noexcept
    {
      static std::atomic<id_index_t> value{};
      return value++;
    }
  };
} // details
```

Последовательность неотрицательных чисел

```cpp
using id_index_t = std::size_t;
namespace details
{
  struct type_index final
  {
    [[nodiscard]] static id_index_t next() noexcept
    {
      static std::atomic<id_index_t> value{};
      return value++;
    }
  };
} // details
```

```cpp
EXPECT_EQ(0, gate::details::type_index::next());
EXPECT_EQ(1, gate::details::type_index::next());
EXPECT_EQ(2, gate::details::type_index::next());
EXPECT_EQ(3, gate::details::type_index::next());
```

Немного кода

```cpp
template <typename Type>
struct type_index final
{
  [[nodiscard]] static id_index_t value() noexcept
  {
    static const id_index_t value = details::type_index::next();
    return value;
  }
};
```

Индекс типа

```
template <typename Type>
struct type_index final
{
  [[nodiscard]] static id_index_t value() noexcept
  {
    static const id_index_t value = details::type_index::next();
    return value;
  }
};
```

```
struct T { };
class B : T { int i; };
class C { std::string s; };

EXPECT_EQ(0, gate::type_index<T>::value());
EXPECT_EQ(1, gate::type_index<B>::value());
EXPECT_EQ(0, gate::type_index<T>::value());
EXPECT_EQ(2, gate::type_index<std::string>::value());
EXPECT_EQ(3, gate::type_index<int>::value());

{ // verify
  EXPECT_EQ(0, gate::type_index<T>::value());
  EXPECT_EQ(1, gate::type_index<B>::value());
  EXPECT_EQ(2, gate::type_index<std::string>::value());
  EXPECT_EQ(3, gate::type_index<int>::value());
}
```

Устойчивость к алиасам

```
EXPECT_EQ(1, gate::type_index<B>());
EXPECT_EQ(4, gate::type_index<C>());

using CAlias = C;
typedef B BTypedef;

EXPECT_EQ(4, gate::type_index<CAlias>());
EXPECT_EQ(1, gate::type_index<BTypedef>());
```

Немного кода

```cpp
template <typename Type>
struct type_index final
{
  [[nodiscard]] static id_index_t value() noexcept
  {
    static const id_index_t value = details::type_index::next();
    return value;
  }
};
```

Перезагрузка оператора

```
template <typename Type>
struct type_index final
{
  [[nodiscard]] static id_index_t value() noexcept
  {
    static const id_index_t value = details::type_index::next();
    return value;
  }

  [[nodiscard]] constexpr operator id_index_t() const noexcept
  {
    return value();
  }
};
```

Перезагрузка оператора

```cpp
template <typename Type>
struct type_index final
{
  [[nodiscard]] static id_index_t value() noexcept
  {
    static const id_index_t value = details::type_index::next();
    return value;
  }

  [[nodiscard]] constexpr operator id_index_t() const noexcept
  {
    return value();
  }
};
```

```cpp
EXPECT_EQ(0, gate::type_index<T>());
EXPECT_EQ(1, gate::type_index<B>());
EXPECT_EQ(0, gate::type_index<T>());
EXPECT_EQ(2, gate::type_index<std::string>());
EXPECT_EQ(3, gate::type_index<int>());
```

Универсальный registry

Универсальный registry

```
registry.emplace(B{});
registry.emplace(A{.member = 1, .another_member = 2});
// ...

if (registry.all_of<A, B, C, D>())
{
    const auto &[a, b] = registry.last<A, B>();
    // ...
    // Do something
    // ...
    for (const auto &c : registry.all<C>())
    {
        // ...
        // Do something
        // ...
    }
    const auto &d = registry.last<D>();
    // ...
    // Do something
    // ...
}
```

Универсальный registry

Универсальный registry

```
using pool_type = std::unordered_map<id_index_t, std::any>;

class registry
{




private:
  pool_type pool_;
};
```

Универсальный registry

```
using pool_type = std::unordered_map<id_index_t, std::any>;

class registry
{
  template <typename T>
  [[nodiscard]] auto &storage(id_index_t id = type_index<T>())
  {

  }

private:
  pool_type pool_;
};
```

Универсальный registry

```
using pool_type = std::unordered_map<id_index_t, std::any>;

class registry
{
  template <typename T>
  [[nodiscard]] auto &storage(id_index_t id = type_index<T>())
  {
    using storage_t = std::vector<T>;
    auto& st = pool_[id];

    if (!st.has_value())
    {
      st.emplace<storage_t>();
    }

    return std::any_cast<storage_t &>(st);
  }

private:
  pool_type pool_;
};
```

Универсальный registry

```cpp
using pool_type = std::unordered_map<id_index_t, std::any>;

class registry
{
  template <typename T>
  [[nodiscard]] auto &storage(id_index_t id = type_index<T>())
  {
    using storage_t = std::vector<T>;
    auto& st = pool_[id]; // any for type T

    if (!st.has_value())
    {
      st.emplace<storage_t>();
    }

    return std::any_cast<storage_t &>(st);
  }

private:
  pool_type pool_;
};
```

Универсальный registry

```cpp
using pool_type = std::unordered_map<id_index_t, std::any>;

class registry
{
  template <typename T>
  [[nodiscard]] auto &storage(id_index_t id = type_index<T>())
  {
    using storage_t = std::vector<T>;
    auto& st = pool_[id]; // any for type T

    if (!st.has_value())
    {
      st.emplace<storage_t>();
    }

    return std::any_cast<storage_t &>(st); // std::vector<T>&
  }

private:
  pool_type pool_;
};
```

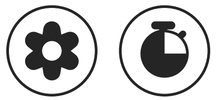
Запись в registry

```
class registry
{
 // ...
 public:
  template<typename T, typename... Args>
  auto& emplace(Args&&... args) {
    return storage<T>().emplace_back(T{std::forward<Args>(args)...});
  }

  template <typename T>
  auto &emplace(T &&arg)
  {
    return storage<T>().emplace_back(std::forward<T>(arg));
  }
  // ...
};
```

Запись в registry

```
template<typename T, typename... Args>
auto& emplace(Args&&... args) {
  return storage<T>().emplace_back(T{std::forward<Args>(args)...});
}

template <typename T>
auto &emplace(T &&arg)
{
  return storage<T>().emplace_back(std::forward<T>(arg));
}
```

```
struct T { int ta = 12; int tb = 13; int tc = 14; };
class B : T { int a = 0; int b = 0; };
class C { int a = 0; };

gate::registry registry;

registry.emplace(B{});
registry.emplace(T{.ta = 1, .tb = 2, .tc = 3});
registry.emplace<T>(3, 4, 5);
```



Чтение из registry

Чтение из registry

```
class registry
{
  // ...

  template <typename T>
  [[nodiscard]] const auto *storage(id_index_t id = type_index<T>()) const
  {
    using storage_t = std::vector<T>;
    if (const auto it = pool_.find(id); it != pool_.cend())
    {
      auto& store_ref = std::any_cast<const storage_t &>(it->second);
      return &store_ref;
    }
    return static_cast<const storage_t *>(nullptr);
  }


  // ...
};
```

Чтение из registry

```
class registry
{
  // ...

  template <typename T>
  [[nodiscard]] const auto *storage(id_index_t id = type_index<T>()) const
  {
    using storage_t = std::vector<T>;
    if (const auto it = pool_.find(id); it != pool_.cend())
    {
      auto& store_ref = std::any_cast<const storage_t &>(it->second);
      return &store_ref;
    }
    return static_cast<const storage_t *>(nullptr);
  }


  // ...
};
```

Чтение из registry

```
class registry
{
 // ...
 template <typename T>
 [[nodiscard]] bool has() const
 {
   auto *cpool = storage<std::decay_t<T>>();
   return cpool && !cpool->empty();
 }

 // ...
};
```

Чтение из registry

```
class registry
{
 // ...
 template <typename T>
 [[nodiscard]] bool has() const
 {
   auto *cpool = storage<std::decay_t<T>>();
   return cpool && !cpool->empty();
 }
public:
 template <typename... T>
 [[nodiscard]] bool all_of() const
 {
   return (has<T>() && ...);
 }
 template <typename... T>
 [[nodiscard]] bool any_of() const
 {
   return (has<T>() || ...);
 }
 // ...
};
```

Чтение из registry

```cpp
class registry
{
 // ...
 template <typename T>
 [[nodiscard]] bool has() const
 {
   auto *cpool = storage<std::decay_t<T>>();
   return cpool && !cpool->empty();
 }
public:
 template <typename... T>
 [[nodiscard]] bool all_of() const
 {
   return (has<T>() && ...);
 }
 template <typename... T>
 [[nodiscard]] bool any_of() const
 {
   return (has<T>() || ...);
 }
 // ...
};
```

```cpp
gate::registry registry;

registry.emplace(B{});
registry.emplace(T{.ta = 1, .tb = 2, .tc = 3});
registry.emplace<T>(3, 4, 5);

EXPECT_TRUE(registry.all_of<T>());
EXPECT_TRUE(registry.any_of<B>());
EXPECT_FALSE(registry.any_of<C>());

EXPECT_TRUE((registry.any_of<C, B>()));
EXPECT_TRUE((registry.any_of<CAlias, BTypedef>()));
EXPECT_FALSE((registry.all_of<C, B>()));
EXPECT_FALSE((registry.all_of<C, B>()));

EXPECT_TRUE((registry.any_of<C, B, T>()));
EXPECT_FALSE((registry.all_of<C, B, T>()));

EXPECT_TRUE((registry.any_of<CAlias, BTypedef, T>()));
EXPECT_FALSE((registry.all_of<CAlias, BTypedef, T>()));

EXPECT_TRUE((registry.all_of<>()));
EXPECT_FALSE((registry.any_of<>()));
```

Небольшое отступление

Небольшое отступление

```cpp
template <typename T, typename... Args>
requires std::same_as<T, std::decay_t<T>>
auto &emplace(Args &&...args)
{
  return storage<T>().emplace_back(T{std::forward<Args>(args)...});
}

template <typename T>
requires std::same_as<T, std::decay_t<T>>
auto &emplace(T &&arg)
{
  return storage<T>().emplace_back(std::forward<T>(arg));
}

template <typename... T>
requires (std::same_as<T, std::decay_t<T>> && ...)
[[nodiscard]] bool all_of() const { return (has<T>() && ...); }

template <typename... T>
requires (std::same_as<T, std::decay_t<T>> && ...)
[[nodiscard]] bool any_of() const { return (has<T>() || ...); }
```

Небольшое отступление

```
template<typename T>
concept DecayType = std::same_as<T, std::decay_t<T>>;
```

Чтение из registry

```
class registry
{
  // ...
  template <typename T>
  [[nodiscard]] const auto &all() const
  {
    return *storage<T>();
  }
  // ...
};
```

Чтение из registry

```
class registry
{
  // ...
  template <typename T>
  [[nodiscard]] const auto &all() const
  {
    return *storage<T>();
  }
  // ...
};
```

```
gate::registry registry;
registry.emplace<T>(1, 2, 3);
registry.emplace<T>(3, 4, 5);

const auto &all_ts = registry.all<T>();
ASSERT_THAT(
    all_ts,
    testing::ElementsAre(T{1, 2, 3}, T{3, 4, 5}));
```

Чтение из registry

```
class registry
{
  // ...
  template <typename... T>
  [[nodiscard]] decltype(auto) last() const
  {
    if constexpr (sizeof...(T) == 1u)
    {
      return (storage<T>()->back(), ...);
    }
    else
    {
      return std::forward_as_tuple(last<T>()...);
    }
  }
  // ...
};
```

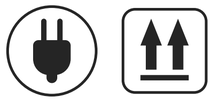
```
gate::registry registry;
registry.emplace(B{{12, 13, 14}, 0, 0});
registry.emplace<T>(1, 2, 3);
registry.emplace<T>(3, 4, 5);


const auto &lt = registry.last<T>();
EXPECT_EQ(lt, (T{3, 4, 5}));

const auto &lb = registry.last<B>();
EXPECT_EQ(lb, (B{{12, 13, 14}, 0, 0}));



const auto &[bref, tref] = registry.last<B, T>();
EXPECT_EQ(tref, (T{3, 4, 5}));
EXPECT_EQ(bref, (B{{12, 13, 14}, 0, 0}));
```

Задача со звездочкой

Задача со звездочкой

```
template <typename... T>
[[nodiscard]] decltype(auto) last() const
{
  if constexpr (sizeof...(T) == 1u)
  {
    return (storage<T>()->back(), ...);
  }
  else
  {
    return std::forward_as_tuple(last<T>()...);
  }
}
```

Почему `decltype(auto)`
а не `auto` ?



Registry

Registry

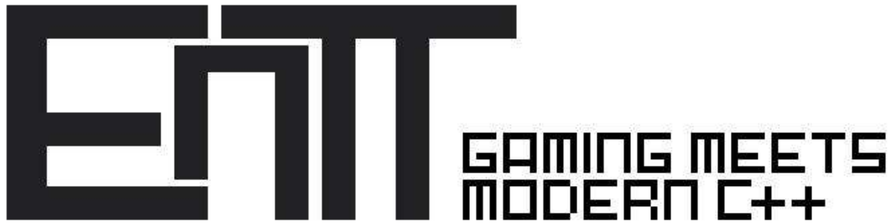
```
if (registry.all_of<A, B, C, D>())
{
    const auto &[a, b] = registry.last<A, B>();
    // ...
    // Do something
    // ...
    for (const auto &c : registry.all<C>())
    {
        // ...
        // Do something
        // ...
    }
    const auto &d = registry.last<D>();
    // ...
    // Do something
    // ...
}
```

Bool Registry

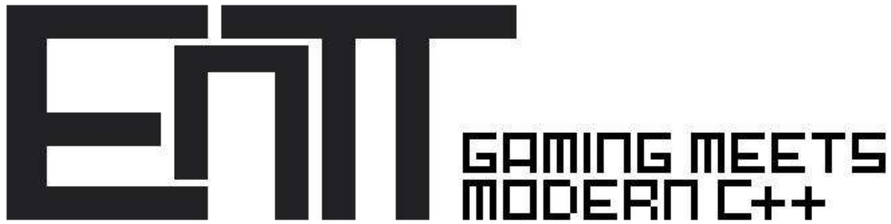
```
gate::BoolRegistry registry;

registry.set<T1>();
registry.set<T2>();

EXPECT_TRUE((registry.test<T1>()));
EXPECT_TRUE((registry.test<T2>()));
EXPECT_FALSE((registry.test<T3>()));

registry.reset<T2>();

EXPECT_TRUE((registry.test<T1>()));
EXPECT_FALSE((registry.test<T2>()));
EXPECT_FALSE((registry.test<T3>()));
```

Пустой слайд
(спасибо, кэп!)

Библиотека ENTT

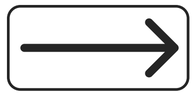


Библиотека ENTT



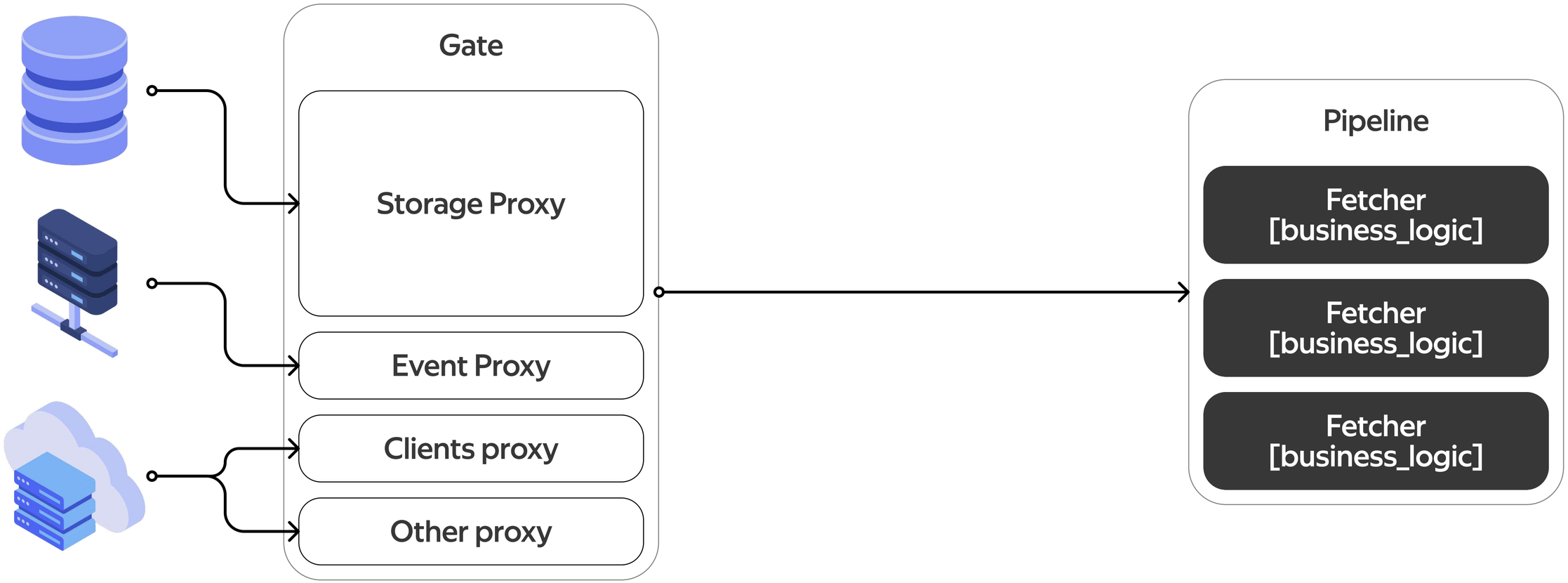
Реализация паттерна ECS
(Entity Component System)

C++ 17

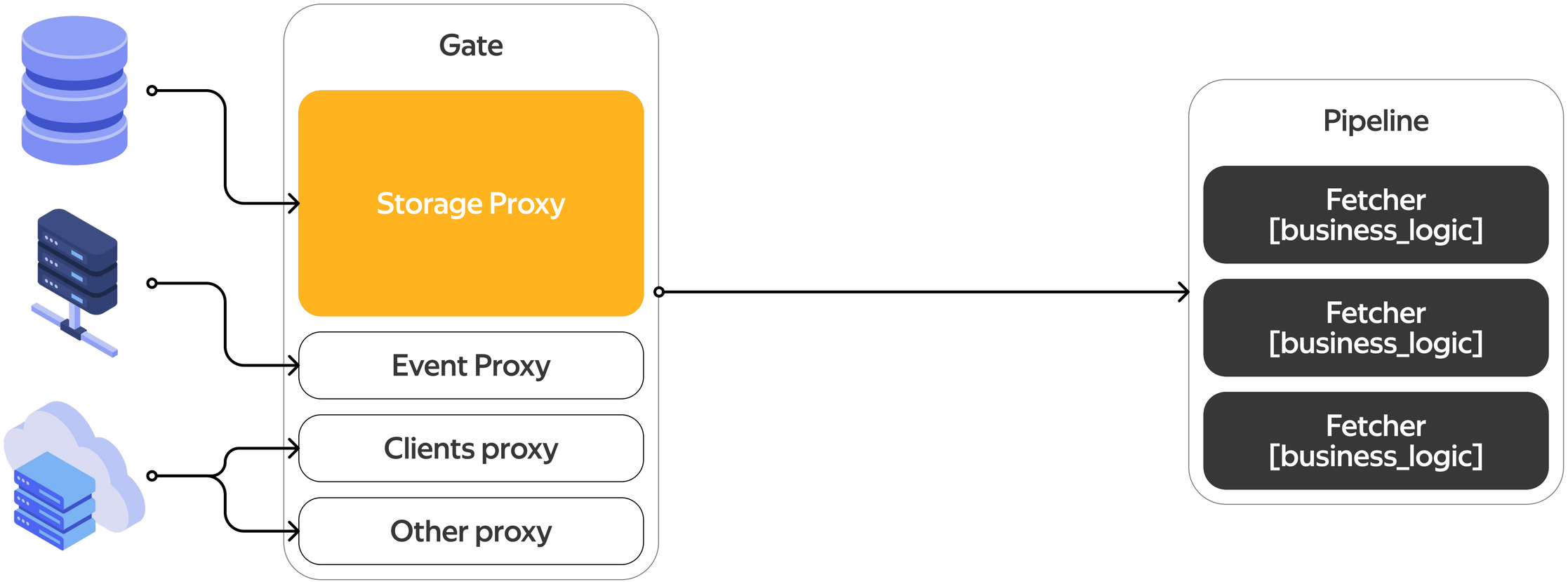


Общая схема решения

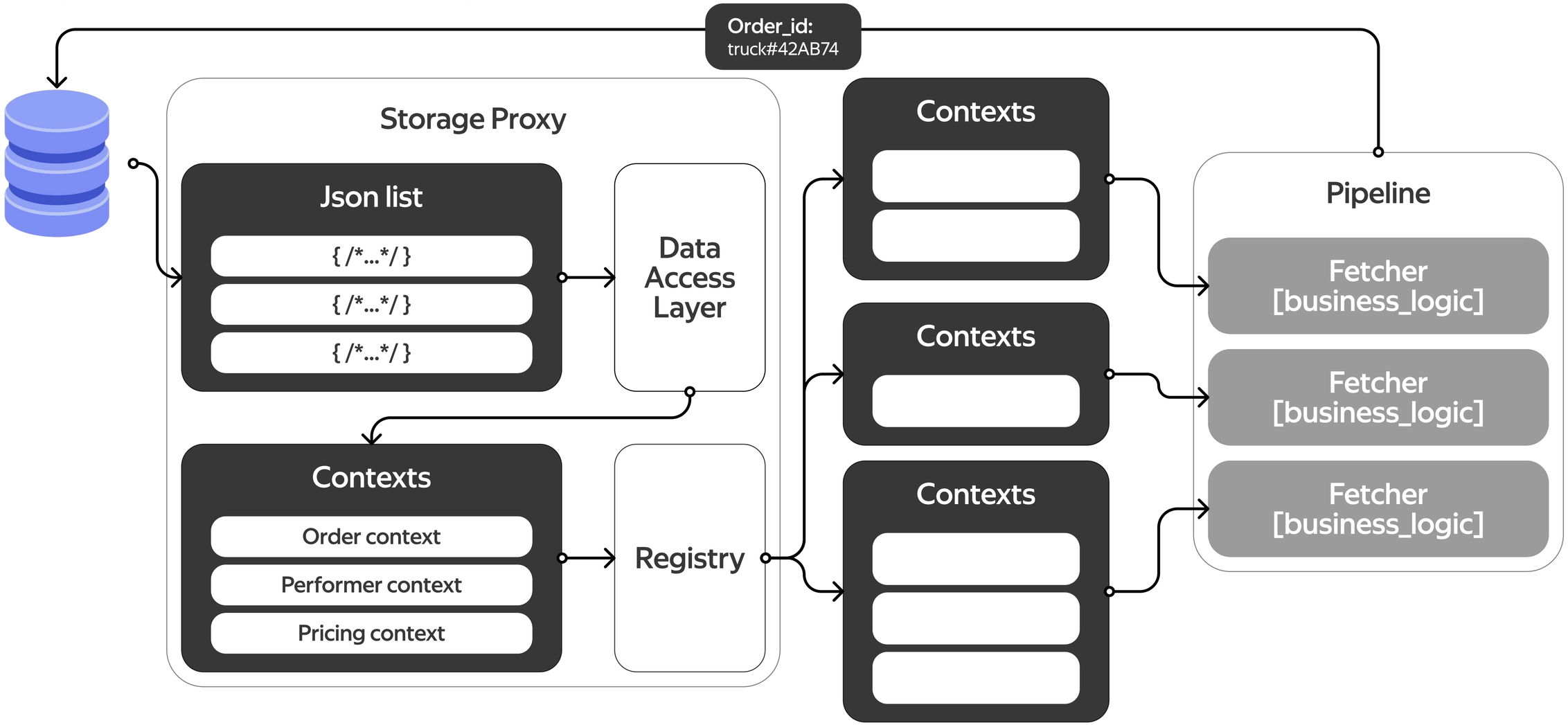
Общая схема решения



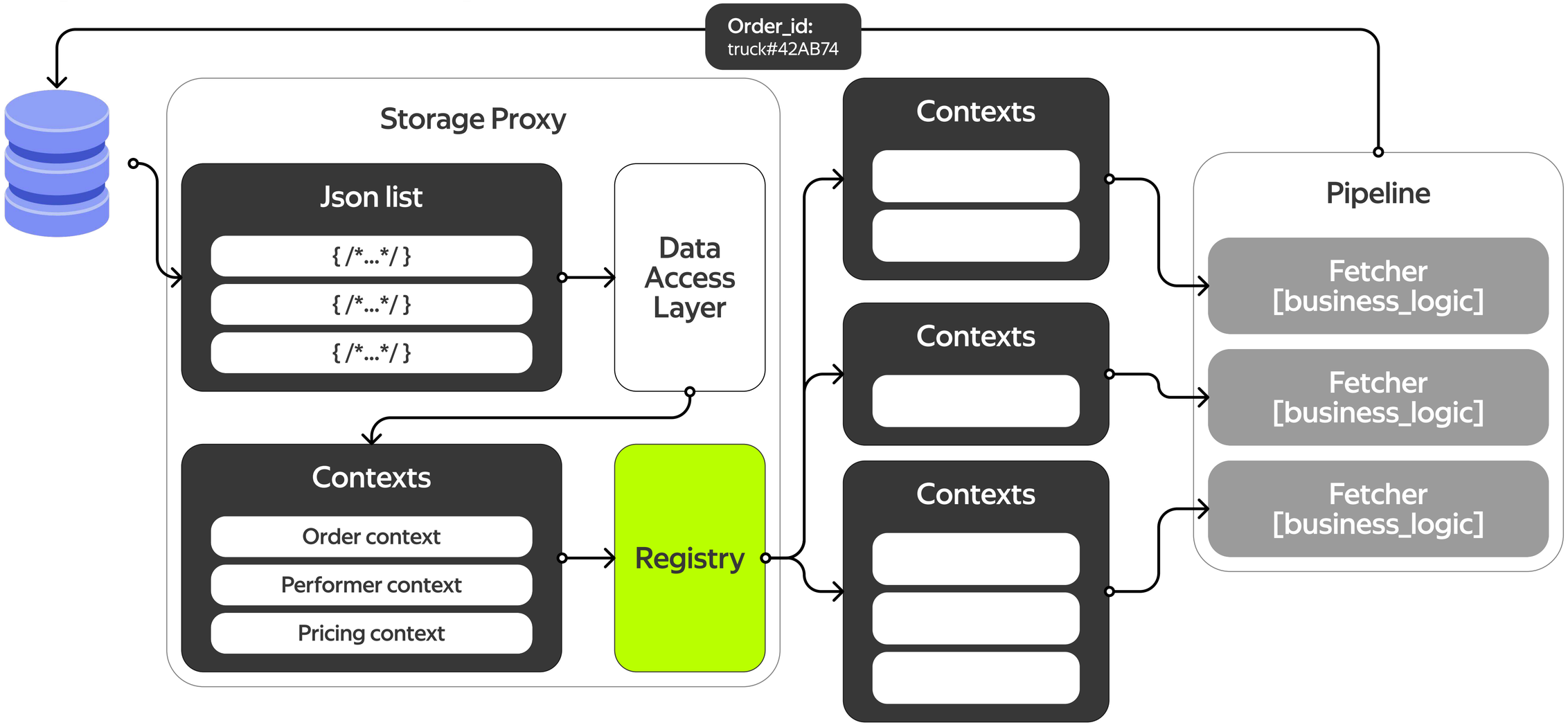
Общая схема решения



Прокси до хранилища

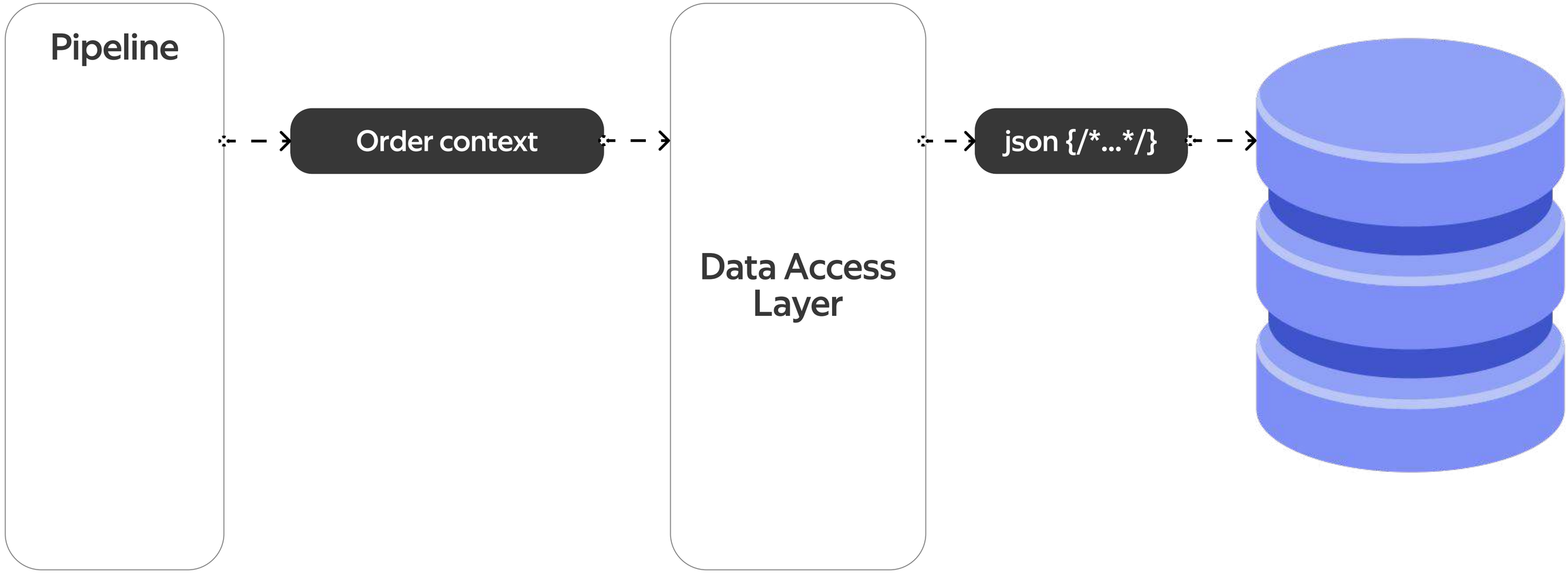


Прокси до хранилища

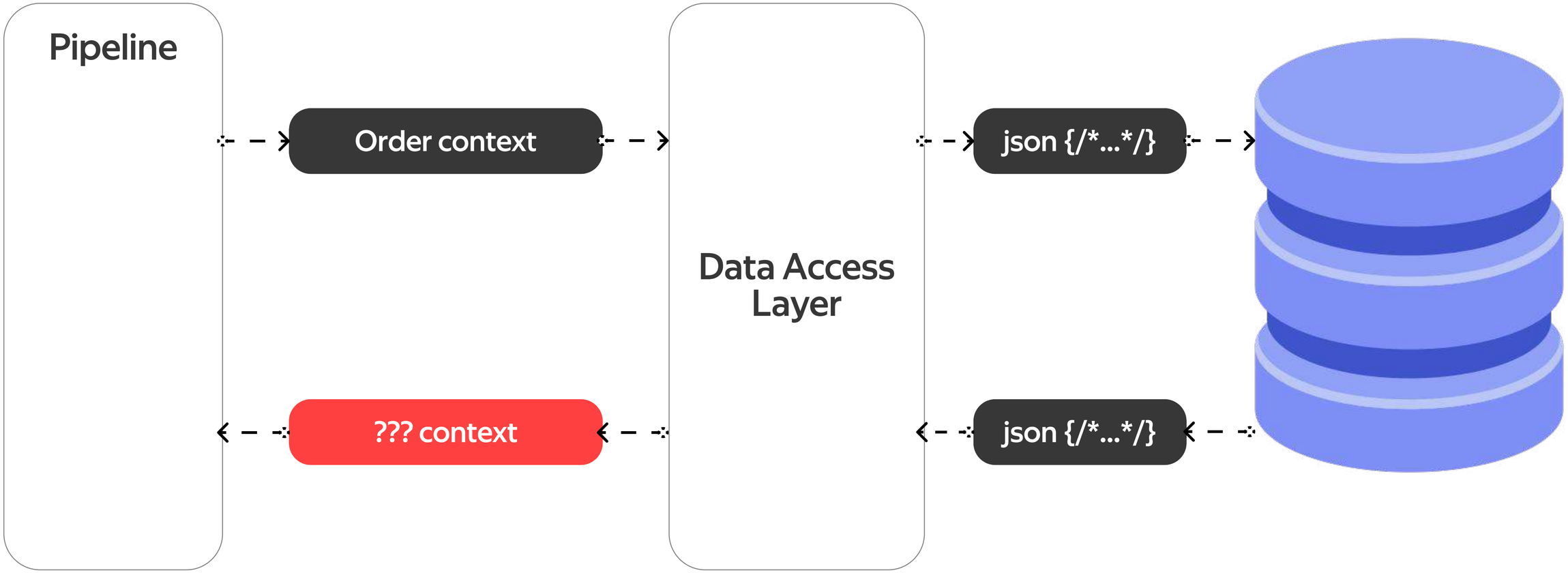


Data Access Layer (DAL)

Data Access Layer (DAL)



Data Access Layer (DAL)



Контекст

Контекст

```
OrderContext:
      type: object
      required:
      - context_kind
      - order_id
      - client_id
      properties:
      context_kind:
          type: string
          enum: [OrderContext]
      order_id:
          type: string
      client_id:
          type: string
      client_wallet:
          type: string
```

Контекст

```
OrderContext:
      type: object
      required:
      - context_kind
      - order_id
      - client_id
      properties:
        context_kind:
            type: string
            enum: [OrderContext]
        order_id:
            type: string
        client_id:
            type: string
        client_wallet:
            type: string
```

Контекст

```yaml
OrderContext:
    type: object
    required:
    - context_kind
    - order_id
    - client_id
    properties:
      context_kind:
        type: string
        enum: [OrderContext]
      order_id:
        type: string
      client_id:
        type: string
      client_wallet:
        type: string
```

```cpp
enum class OrderContextKind
{
  OrderContext
};

struct OrderContext
{
  OrderContextKind context_kind =
      OrderContextKind::OrderContext;

  std::string order_id;
  std::string client_id;
  std::string client_wallet;
};

std::string ToString(OrderContextKind kind);

template <>
OrderContext FromJson<OrderContext>(const json &input);
template <>
json ToJson(const OrderContext &input);
```

Парсер

Парсер

```
template <typename C>
static void Parse(const json &value, gate::registry &registry)
{
  const auto type_context = ToString(C{}.context_kind);
  const auto value_context = value["context_kind"];
  if (type_context == value_context)
  {
    registry.emplace(contexts::FromJson<C>(value));
  }
}
```

Парсер

```
template <typename C>
static bool Parse(const json &value, gate::registry &registry)
{
  const auto type_context = ToString(C{}.context_kind);
  const auto value_context = value["context_kind"];
  if (type_context == value_context)
  {
    registry.emplace(contexts::FromJson<C>(value));
    return true;
  }
  return false;
}

template <typename... T>
static void ParseValue(const json &value, gate::registry &registry)
{
  (Parse<T>(value, registry) || ...);
}
```

Парсер

```
static void FillRegistry(const std::vector<json> &values, gate::registry &registry)
{
  for (const auto &value : values)
  {
    ParseValue(value, registry);
  }
}
```

Парсер

```
{
  using TheParser =
        gate::Parser<PricingContext, OrderContext, PerformerContext>;

  gate::registry registry;

  TheParser::FillRegistry({pricing1, order, pricing2}, registry);

  EXPECT_TRUE((registry.all_of<PricingContext, OrderContext>()));
  EXPECT_FALSE((registry.all_of<PerformerContext>()));

  EXPECT_EQ("cargo_express", registry.last<PricingContext>().tariff);
}
```

Выявление типов

Выявление типов

```
struct PriceCalculatedEventFetcher
{
 static PricingContext Fetch(
     const PriceCalculatedEvent &event,
     PricingClient &pricingClient);
};

using Pipeline = gate::Pipeline<
   fetchers::OrderCreatedEventFetcher,
   fetchers::PriceCalculatedEventFetcher,
   fetchers::PerformerFoundEventFetcher,
   fetchers::BuildPaymentFetcher>;
```

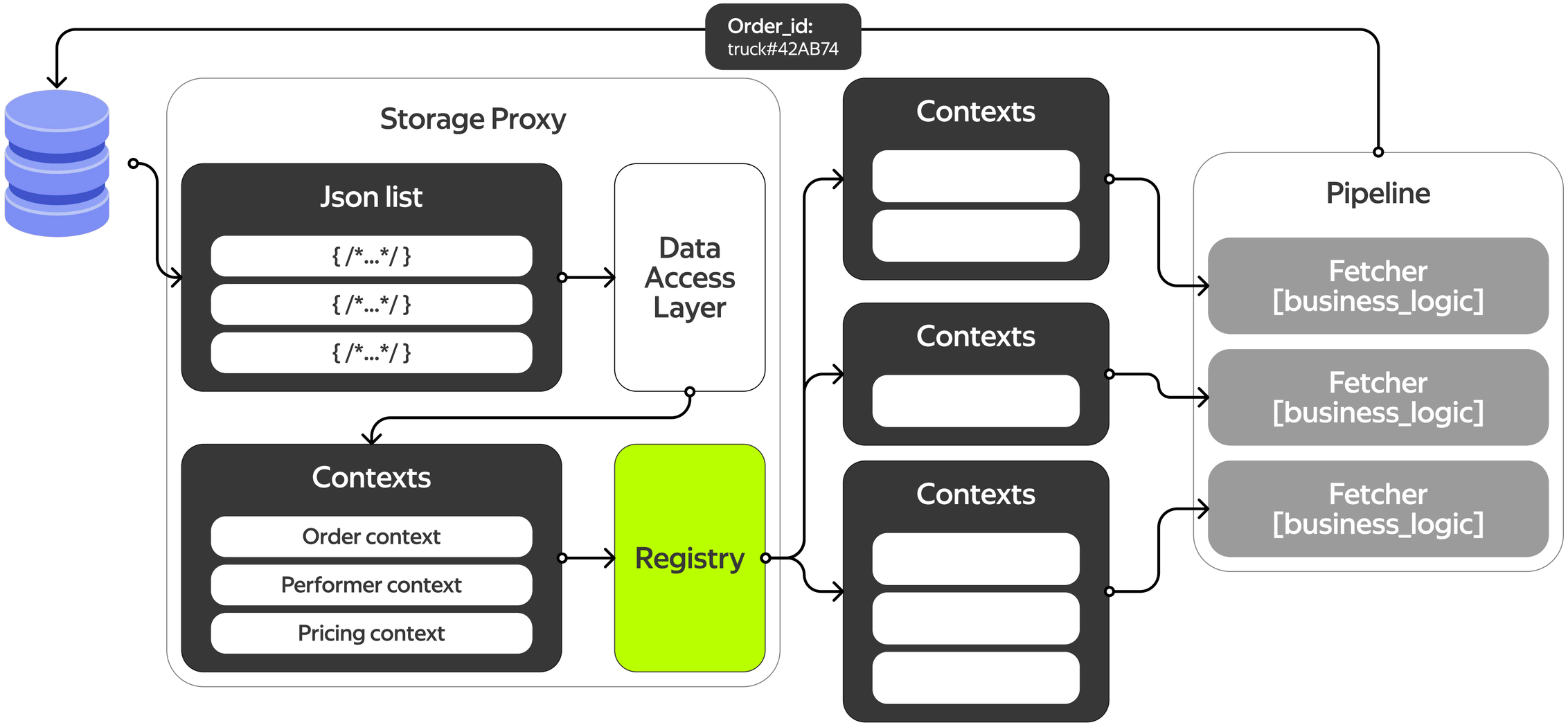
Выявление типов

```
template <typename F>
struct TypeHelper;

template <typename R, typename... Args>
struct TypeHelper<R(Args...)>
{
  using ReturnType = R;
  void ForArgs() {
    ForArgsImpl<Args...>();
  }
  void ForAllTypes() {
    ForArgsImpl<R, Args...>();
  }
};
```

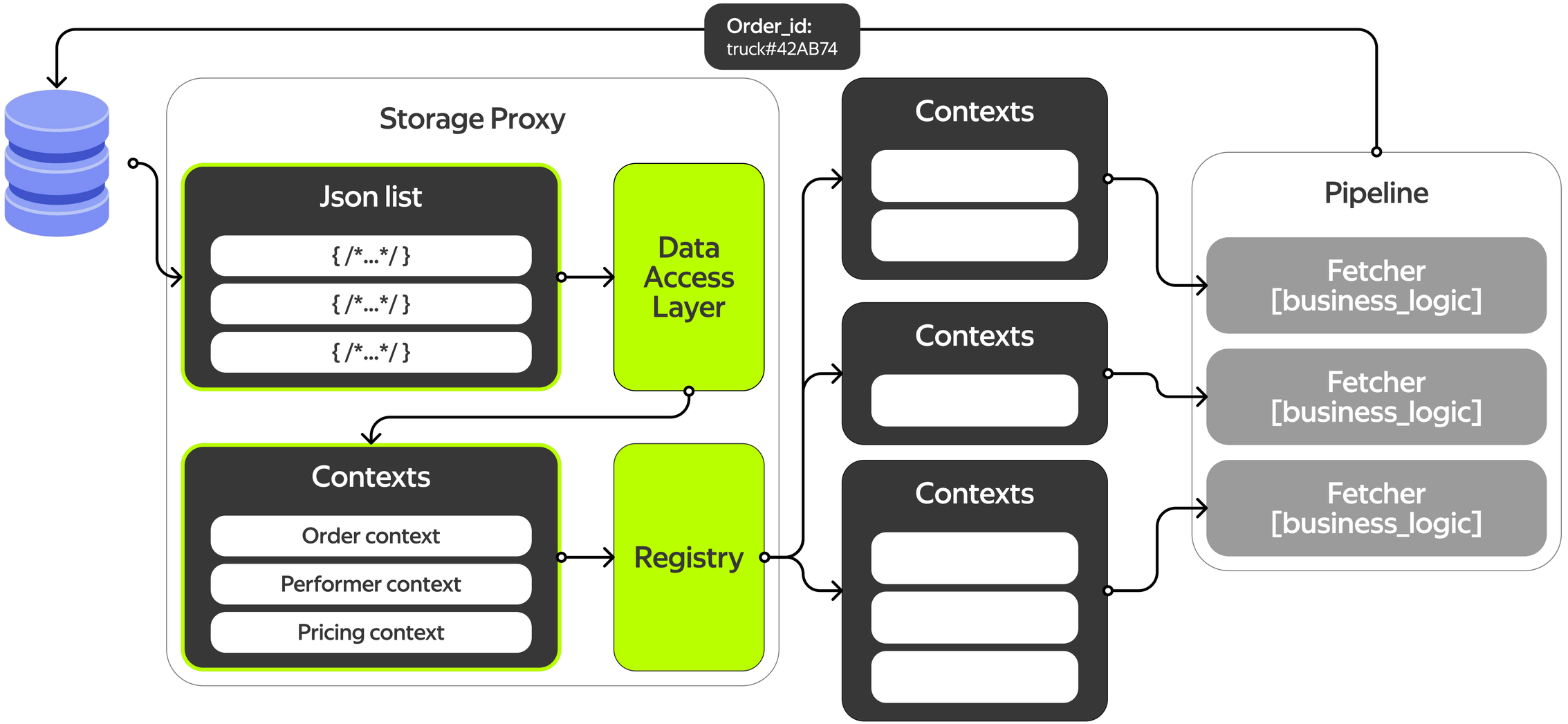
Выявление типов

```
template <typename F>
struct TypeHelper;

template <typename R, typename... Args>
struct TypeHelper<R(Args...)>
{
  // ...
};

template <typename T>
using FetcherHelper = TypeHelper<decltype(T::Fetch)>;
```

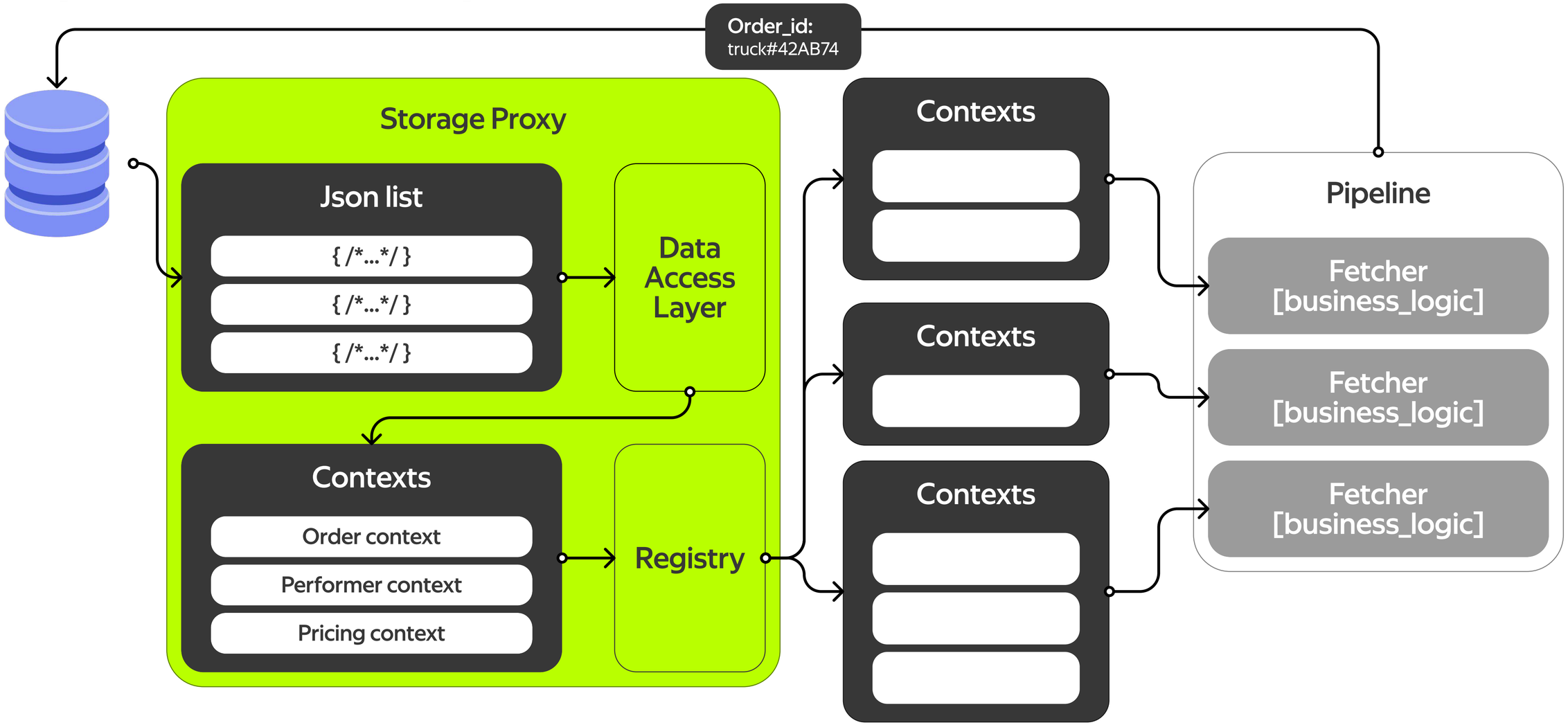
Прокси до хранилища



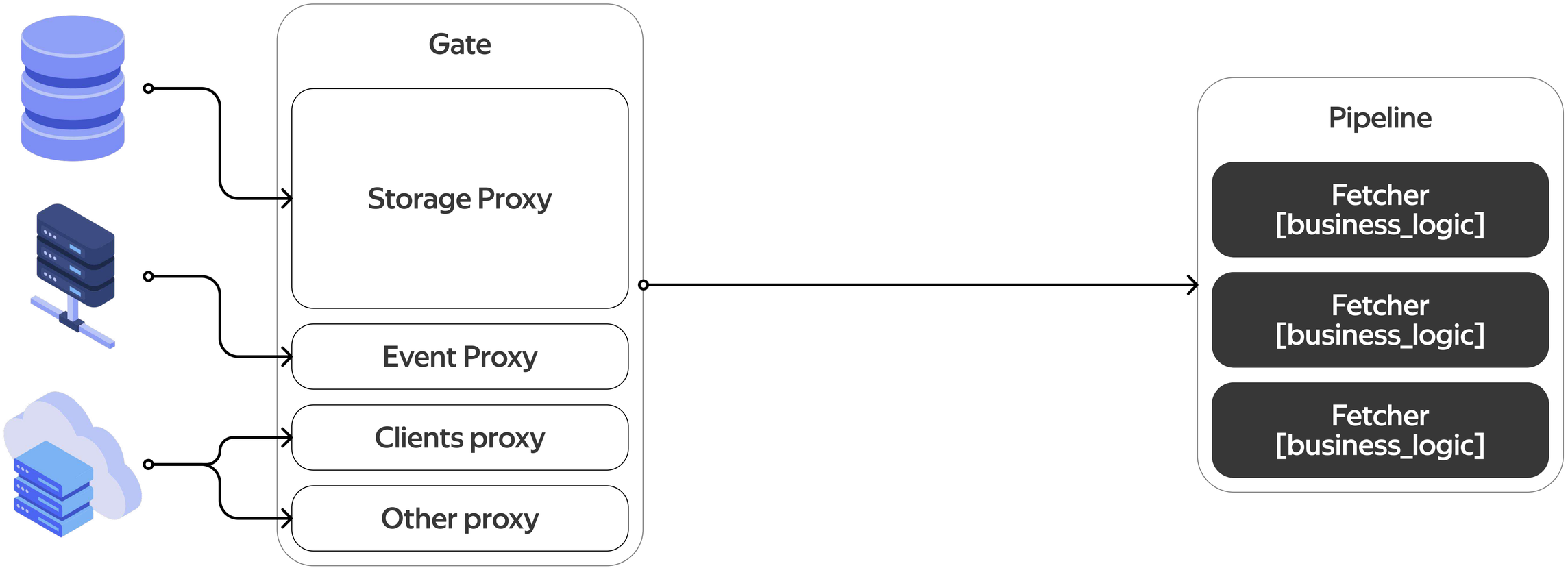
Прокси до хранилища



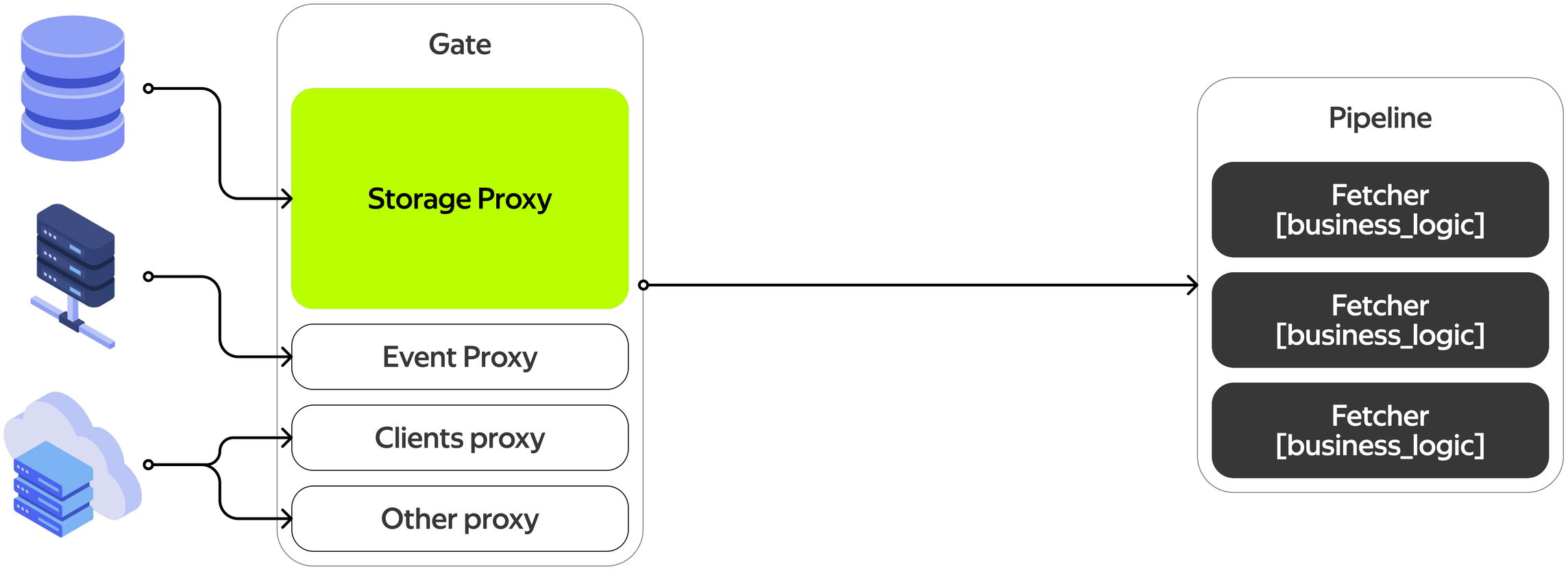
Прокси до хранилища



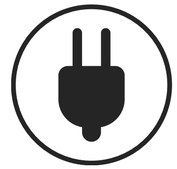
Общая схема решения



Общая схема решения



Часть 03

Вспомогательные   трюки

Вспомогательная фича

```
namespace my_namespace
{
  struct MyStruct
  {
  };
}

int main()
{
  std::cout << typeid(my_namespace::MyStruct).name() << std::endl;
  std::cout << typeid(std::string).name() << std::endl;
  std::cout << typeid(int).name() << std::endl;
}
```

Вспомогательная фича

```
namespace my_namespace
{
  struct MyStruct
  {
  };
}

int main()
{
  std::cout << typeid(my_namespace::MyStruct).name() << std::endl;
  std::cout << typeid(std::string).name() << std::endl;
  std::cout << typeid(int).name() << std::endl;
}


/*
N12my_namespace8MyStructE
NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
i
*/
```

Вспомогательная фича

```
 template <typename T>
 constexpr std::string_view wrapped_type_name()
 {
#ifdef __cpp_lib_source_location
   return std::source_location::current().function_name();
#else
   return __PRETTY_FUNCTION__;
#endif
 }
```

Вспомогательная фича

```
 template <typename T>
 constexpr std::string_view wrapped_type_name()
 {
#ifdef __cpp_lib_source_location
   return std::source_location::current().function_name();
#else
   return __PRETTY_FUNCTION__;
#endif
 }

using MyStruct = my_namespace::MyStruct;

int main()
{
 std::cout << gate::wrapped_type_name<MyStruct>() << std::endl;
}
/*
constexpr std::string_view gate::wrapped_type_name()
   [with T = my_namespace::MyStruct; std::string_view =
     std::basic_string_view<char>]
*/
```

Вспомогательная фича

```
 template <typename T>
 constexpr std::string_view wrapped_type_name()
 {
#ifdef __cpp_lib_source_location
   return std::source_location::current().function_name();
#else
   return __PRETTY_FUNCTION__;
#endif
 }

using MyStruct = my_namespace::MyStruct;

int main()
{
 std::cout << gate::wrapped_type_name<MyStruct>() << std::endl;
}
/*
constexpr std::string_view gate::wrapped_type_name()
   [with T = my_namespace::MyStruct; std::string_view =
     std::basic_string_view<char>]
*/
```

Вспомогательная фича

```
/*
constexpr std::string_view gate::wrapped_type_name()
   [with T = void; std::string_view =
     std::basic_string_view<char>]
*/

 constexpr std::size_t wrapped_type_name_prefix_length()
 {
   return wrapped_type_name<void>().find("void");
 }
 constexpr std::size_t wrapped_type_name_suffix_length()
 {
   return wrapped_type_name<void>().length() -
          wrapped_type_name_prefix_length() -
          std::string_view("void").length();
 }
```

Вспомогательная фича

```
template <typename T>
constexpr std::string_view type_name()
{
  constexpr auto wrapped_name = detail::wrapped_type_name<T>();
  constexpr auto prefix_length = detail::wrapped_type_name_prefix_length();
  constexpr auto suffix_length = detail::wrapped_type_name_suffix_length();
  constexpr auto type_name_length =
      wrapped_name.length() - prefix_length - suffix_length;

  return wrapped_name.substr(prefix_length, type_name_length);
}
```

Вспомогательная фича

```
std::cout << typeid(my_namespace::MyStruct).name() << std::endl;
std::cout << typeid(std::string).name() << std::endl;
std::cout << typeid(int).name() << std::endl;

/*
N12my_namespace8MyStructE
NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
i
*/
```

```
std::cout << gate::type_name<my_namespace::MyStruct>() << std::endl;
std::cout << gate::type_name<MyStruct>() << std::endl;
std::cout << gate::type_name<std::string>() << std::endl;
std::cout << gate::type_name<int>() << std::endl;

/*
my_namespace::MyStruct
my_namespace::MyStruct
std::__cxx11::basic_string<char>
int
*/
```

Самая крутая и полезная фича

Концепты

```
template <typename T>
concept Optional = std::same_as<T, std::optional<typename T::value_type>>;
```

Концепты

```
template <typename T>
concept Optional = std::same_as<T, std::optional<typename T::value_type>>;


template <Optional T>
void f(const T &t)
{
 if (t.has_value())
 {
   std::cout << "optional: " << t.value() << std::endl;
   return;
 }
 std::cout << "empty: " << gate::type_name<T>() << std::endl;
}

template <typename T>
void f(const T &t)
{
 std::cout << "value: " << t << std::endl;
}
```

Концепты

```cpp
template <typename T>
concept Optional = std::same_as<T, std::optional<typename T::value_type>>;


template <Optional T>
void f(const T &t)
{
 if (t.has_value())
 {
   std::cout << "optional: " << t.value() << std::endl;
   return;
 }
 std::cout << "empty: " << gate::type_name<T>() << std::endl;
}

template <typename T>
void f(const T &t)
{
 std::cout << "value: " << t << std::endl;
}
```

Концепты

```
template <Optional T>
void f(const T &t)
{
 if (t.has_value())
 {
   std::cout << "optional: "
  << t.value() << std::endl;
   return;
 }
 std::cout << "empty: "
           << gate::type_name<T>() << std::endl;
}


template <typename T>
void f(const T &t)
{
 std::cout << "value: " << t << std::endl;
}
```

```
{
 std::optional<std::string> s;

 f(s);
 f("Just do nothing");
 s = "Just do it";
 f(s);
 f(std::make_optional(123));
 f(321);
}


/*
empty: std::optional<std::__cxx11::basic_string<char> >
value: Just do nothing
optional: Just do it
optional: 123
value: 321
*/
```

Концепты

```
template <typename T>
concept Optional = std::same_as<T, std::optional<typename T::value_type>>;

template <typename T>
concept Context = requires(T t) {
  {
    ToString(t.context_kind)
  } → std::convertible_to<std::string>;
};

template <typename T>
concept Event = requires(T t) {
  {
    ToString(t.event_kind)
  } → std::convertible_to<std::string>;
};
```

Концепты

```
template <Context C>
static bool Parse(const json &value, gate::registry &registry)
{
  // ... Parse Context
}

template <typename C>
static bool Parse(const json &, gate::registry &)
{
  // ... Do nothing
}
```

Вызов функции

Вызов функции

```
struct BuildPaymentFetcher
{
  static Payment Fetch(
      const OrderContext &orderContext,
      const PricingContext &pricingContext,
      const PerformerContext &performerContext);
};
```

Вызов функции

```
struct BuildPaymentFetcher
{
  static Payment Fetch(
      const OrderContext &orderContext,
      const PricingContext &pricingContext,
      const PerformerContext &performerContext);
};


auto res = BuildPaymentFetcher::Fetch(registry.last<Args>()... );
```

Вызов функции

```
namespace details
{
  template <typename Fetcher, typename Func>
  struct TriggerHelper;

  template <typename Fetcher, typename R, typename... Args>
  struct TriggerHelper<Fetcher, R(Args...)>
  {
    static bool CanTrigger(const gate::registry &registry)
    {
      return registry.all_of<Args...>();
    }

}

template <typename F>
using FetcherHelper = details::TriggerHelper<F, decltype(F::Fetch)>;
```

Вызов функции

```
namespace details
{
  template <typename Fetcher, typename Func>
  struct TriggerHelper;

  template <typename Fetcher, typename R, typename ... Args>
  struct TriggerHelper<Fetcher, R(Args ... )>
  {
    static bool CanTrigger(const gate::registry &registry)
    {
      return registry.all_of<Args ... >();
    }

}

template <typename F>
using FetcherHelper = details::TriggerHelper<F, decltype(F::Fetch)>;
```

Вызов функции

```
namespace details
{
  template <typename Fetcher, typename Func>
  struct TriggerHelper;

  template <typename Fetcher, typename R, typename... Args>
  struct TriggerHelper<Fetcher, R(Args...)>
  {
    static bool CanTrigger(const gate::registry &registry)
    {
      return registry.all_of<Args...>();
    }
    static R Trigger(const gate::registry &registry)
    {
      auto res = Fetcher::Fetch(registry.last<Args>()...);
      return res;
    }
  };
}
template <typename F>
using FetcherHelper = details::TriggerHelper<F, decltype(F::Fetch)>;
```

Вызов функции

```cpp
namespace details
{
  template <typename Fetcher, typename Func>
  struct TriggerHelper;

  template <typename Fetcher, typename R, typename... Args>
  struct TriggerHelper<Fetcher, R(Args...)>
  {
    static bool CanTrigger(const gate::registry &registry)
    {
      return registry.all_of<Args...>();
    }
    static R Trigger(const gate::registry &registry)
    {
      auto res = Fetcher::Fetch(registry.last<Args>()...);
      return res;
    }
  };
}
template <typename F>
using FetcherHelper = details::TriggerHelper<F, decltype(F::Fetch)>;
```

Вызов функции

```
gate::registry registry;

registry.emplace(contexts::PricingContext{});
registry.emplace(contexts::OrderContext{});

EXPECT_TRUE(gate::FetcherHelper<TwoContextFetcher>::CanTrigger(registry));
EXPECT_FALSE(gate::FetcherHelper<AllContextsFetcher>::CanTrigger(registry));

gate::FetcherHelper<TwoContextFetcher>::Trigger(registry);
```

Вызов функции

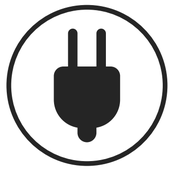
```
struct BuildPaymentFetcher
{
  static Payment Fetch(
      const OrderContext &orderContext,
      const PricingContext &pricingContext,
      const PerformerContext &performerContext);
};
```

Вызов функции

```
struct BuildPaymentFetcher
{
  static Payment Fetch(
      const OrderContext &orderContext,
      const PricingContext &pricingContext,
      const PerformerContext &performerContext);
};
```

```
struct PerformerFoundEventFetcher
{
  static PerformerContext Fetch(
      const PerformerFoundEvent &event,
      TaxiParkClient &taxiParkClient,
      WalkersClient &walkersClient,
      ContractsClient &contractClient,
      const HostSettings &hostSettings);
};
```

Часть 04

Шлюз 

Концепты

```
template <typename T>
concept Optional = std::same_as<T, std::optional<typename T::value_type>>;

template <typename T>
concept Context = requires(T t) {
  {
    ToString(t.context_kind)
  } → std::convertible_to<std::string>;
};

template <typename T>
concept Event = requires(T t) {
  {
    ToString(t.event_kind)
  } → std::convertible_to<std::string>;
};
```

Шлюз

```
class Gate
{
public:


private:
 clients::Dependencies deps_;
 gate::registry &registry_;
 nlohmann::json event_;
};
```

Шлюз

```
class Gate
{
public:
 template <Event T>
 [[nodiscard]] bool has()
 {
   return event_["event_kind"] == events::ToString(T{}.event_kind);
 }

private:
 clients::Dependencies deps_;
 gate::registry &registry_;
 nlohmann::json event_;
};
```

Шлюз

```
class Gate
{
public:
 template <Event T>
 [[nodiscard]] bool has()
 {
   return event_["event_kind"] == events::ToString(T{}.event_kind);
 }

private:
 clients::Dependencies deps_;
 gate::registry &registry_;
 nlohmann::json event_;
};
```

Шлюз

```
class Gate
{
public:
 template <Event T>
 [[nodiscard]] bool has()
 {
   return event_["event_kind"] == events::ToString(T{}.event_kind);
 }

 template <ContextArgument T>
 [[nodiscard]] bool has()
 {
   return registry_.all_of<std::decay_t<T>>();
 }

private:
 clients::Dependencies deps_;
 gate::registry &registry_;
 nlohmann::json event_;
};
```

Шлюз

```cpp
template <Event T>
[[nodiscard]] bool has()
{
  return event_["event_kind"] == events::ToString(T{}.event_kind);
}

template <ContextArgument T>
[[nodiscard]] bool has()
{
  return registry_.all_of<std::decay_t<T>>();
}

template <clients::DependencyType T>
[[nodiscard]] bool has()
{
  return true;
}

template <Optional T>
[[nodiscard]] bool has()
{
  return true;
}
```

Шлюз

```
template <typename Fetcher, typename R, typename... Args>
struct TriggerHelper<Fetcher, R(Args...)>
{
 static bool CanTrigger(const gate::registry &registry)
 {
   return registry.all_of<Args...>();
 }

 // ...

};
```

Шлюз

```
template <typename Fetcher, typename R, typename... Args>
struct TriggerHelper<Fetcher, R(Args...)>
{
 static bool CanTrigger(const gate::registry &registry)
 static bool CanTrigger(auto &gate)

 {
   return registry.all_of<Args...>();
   return (gate.template has<Args>() && ...);
 }

 // ...

};
```

Шлюз

```
template <typename Fetcher, typename R, typename... Args>
struct TriggerHelper<Fetcher, R(Args...)>
{
 static bool CanTrigger(auto &gate)
 {
   return (gate.template has<Args>() && ...);
 }

};
```

```
 EXPECT_FALSE(gate::FetcherHelper<TestEverythingetcher>::CanTrigger(g));
 registry.emplace(contexts::PerformerContext{});
 EXPECT_TRUE(gate::FetcherHelper<TestEverythingetcher>::CanTrigger(g));
```

Шлюз

```
class Gate
{
public:

  template <Event T>
  [[nodiscard]] auto get()
  {
    return events::FromJson<std::decay_t<T>>(event_);
  }

  template <ContextArgument T>
  [[nodiscard]] T get()
  {
    return registry_.last<std::decay_t<T>>();
  }

};
```

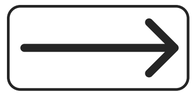
Шлюз

```
template <ContextArgument T>
[[nodiscard]] T get()
{
  return registry_.last<std::decay_t<T>>();
}

template <clients::DependencyType T>
[[nodiscard]] auto& get()
{
  return deps_.Get<T>();
}

template <Optional T>
[[nodiscard]] T get()
{
  if (has<typename T::value_type>())
  {
    return T(get<typename T::value_type>());
  }
  return T{};
}
```

Шлюз

```
struct TestEverythingetcher
{
  static contexts::OrderContext Fetch(
      const events::OrderCreatedEvent &event,
      std::optional<contexts::OrderContext> orderContext,
      const contexts::PerformerContext &performerContext,
      clients::PricingClient &client)
  {
    return contexts::OrderContext{.order_id = "From test"};
  }
};
```

```
auto res = gate::FetcherHelper<TestEverythingetcher>::Trigger(g);

EXPECT_EQ("From test", res.order_id);
```



Расширения шлюза

Расширения шлюза

```
template <typename T>
concept ContextHistory = requires {
  typename std::decay_t<T>::value_type;
  typename std::decay_t<T>::allocator_type;
  requires Context<typename std::decay_t<T>::value_type>;
  requires std::same_as<
    T,
    const std::vector<typename std::decay_t<T>::value_type,
                      typename std::decay_t<T>::allocator_type>&>;
};
```

Расширения шлюза

```cpp
template <typename T>
concept ContextHistory = requires {
  typename std::decay_t<T>::value_type;
  typename std::decay_t<T>::allocator_type;
  requires Context<typename std::decay_t<T>::value_type>;
  requires std::same_as<
    T,
    const std::vector<typename std::decay_t<T>::value_type,
                      typename std::decay_t<T>::allocator_type>&>;
};
```

```cpp
  template <ContextHistory T>
  [[nodiscard]] bool has()
  {
    return has<typename std::decay_t<T>::value_type>();
  }

  template <ContextHistory T>
  [[nodiscard]] const auto& get()
  {
    return registry_.all<typename std::decay_t<T>::value_type>();
  }
```

Расширения шлюза

```
struct TestWithHistory
{
  static contexts::OrderContext Fetch(
      const events::OrderCreatedEvent &event,
      const std::vector<contexts::PerformerContext> &performerContext)
  {
    auto val = "History size "s + std::to_string(performerContext.size());
    return contexts::OrderContext{.order_id = val};
  }
};
```

```
EXPECT_TRUE(gate::FetcherHelper<TestWithHistory>::CanTrigger(g));
auto hres = gate::FetcherHelper<TestWithHistory>::Trigger(g);
EXPECT_EQ("History size 2", hres.order_id);
```

Расширения шлюза: тулкиты

Расширения шлюза: тулкиты

```
struct PerformerFoundEventFetcher
{
 static PerformerContext Fetch(
      const PerformerFoundEvent &event,
      PricingClient &taxiParkClient,
      TaxiParkClient &taxiParkClient,
      WalkersClient &walkersClient,
      ContractsClient &contractClient,
      const HostSettings &hostSettings);
};
```

Расширения шлюза: тулкиты

```
struct TestToolkit
{
  static TestToolkit MakeToolkit(
      clients::TaxiParkClient &taxiParkClient,
      clients::WalkersClient &walkersClient,
      clients::ContractsClient &contractClient);

  std::string GetWallet(std::string performer_id);
  std::string GetContract(std::string performer_id,
                          std::string legal_scheme);

};
```

Расширения шлюза: тулкиты

```
struct TestToolkit
{
  static TestToolkit MakeToolkit(
      clients::TaxiParkClient &taxiParkClient,
      clients::WalkersClient &walkersClient,
      clients::ContractsClient &contractClient);

  std::string GetWallet(std::string performer_id);
  std::string GetContract(std::string performer_id,
                          std::string legal_scheme);

};
```

Расширения шлюза

```
template <typename T>
concept Toolkit =
    std::same_as<T, typename TypeHelper<decltype(T::MakeToolkit)>::ReturnType>;


  template <Toolkit T>
  [[nodiscard]] bool has()
  {
    return ToolkitHelper<T>::CanTrigger(*this);
  }

  template <Toolkit T>
  [[nodiscard]] auto get()
  {
    return ToolkitHelper<T>::Trigger(*this);
  }
```

Расширения шлюза

```cpp
template <typename T>
concept Toolkit =
    std::same_as<T, typename TypeHelper<decltype(T::MakeToolkit)>::ReturnType>;


  template <Toolkit T>
  [[nodiscard]] bool has()
  {
    return ToolkitHelper<T>::CanTrigger(*this);
  }

  template <Toolkit T>
  [[nodiscard]] auto get()
  {
    return ToolkitHelper<T>::Trigger(*this);
  }
```

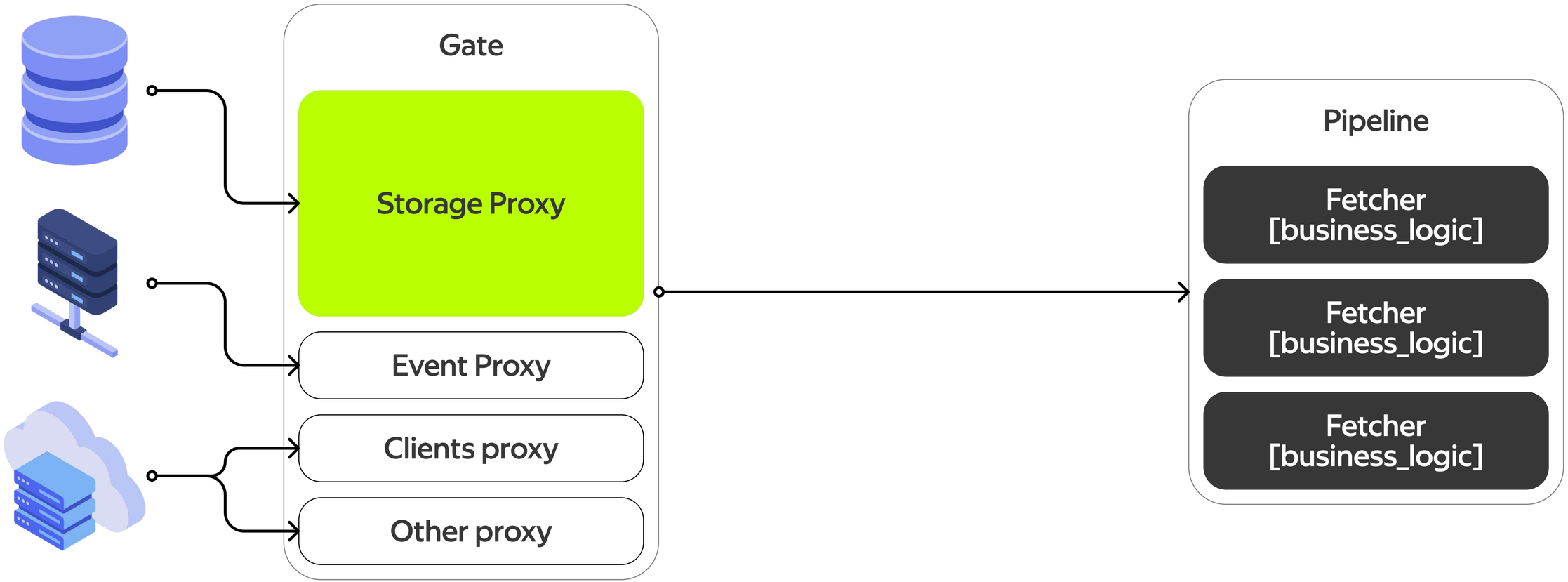
Расширения шлюза

```
struct PerformerFoundEventFetcher
{
 static PerformerContext Fetch(
      const PerformerFoundEvent &event,
      PricingClient &taxiParkClient,
      TaxiParkClient &taxiParkClient,
      WalkersClient &walkersClient,
      ContractsClient &contractClient,
      PerformerToolkit performerToolkit,
      const HostSettings &hostSettings);
};
```

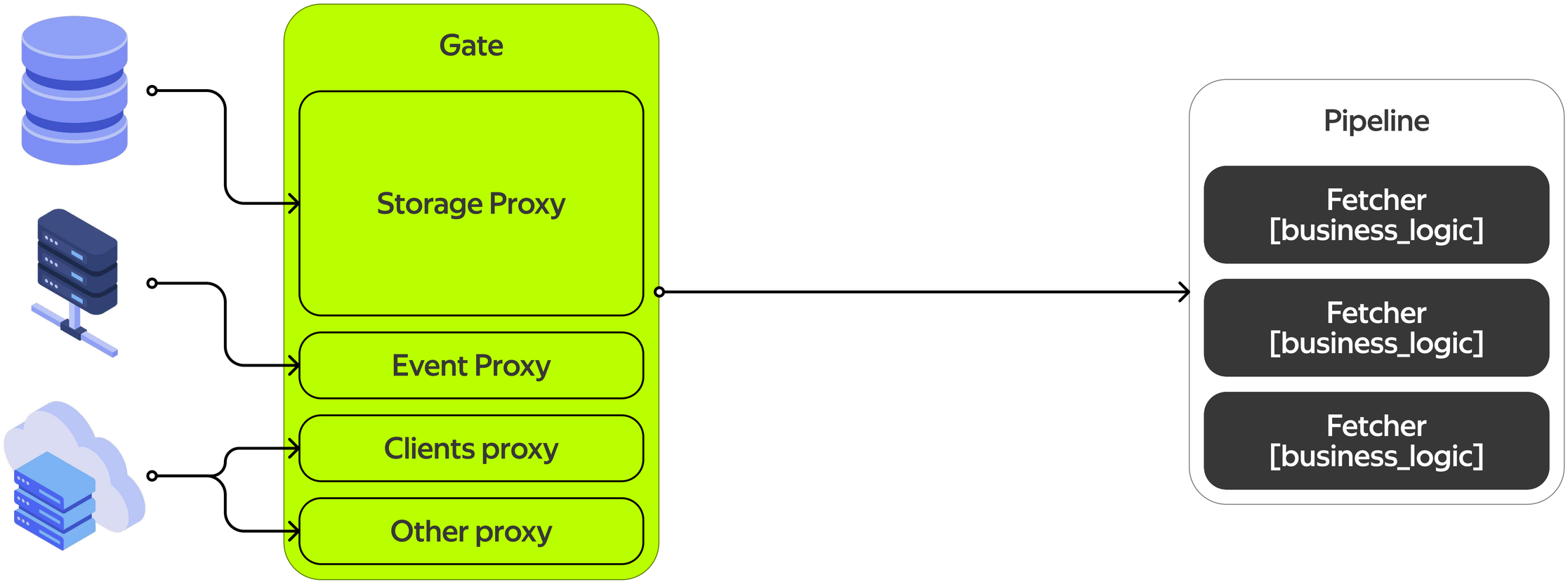
Расширения шлюза

```
struct TestToolkit
{
  static TestToolkit MakeToolkit(
      clients::TaxiParkClient &taxiParkClient,
      clients::WalkersClient &walkersClient,
      clients::ContractsClient &contractClient);
      // ...
};

struct TestWithToolkit
{
   static contexts::OrderContext Fetch(
       const events::OrderCreatedEvent &event,
       TestToolkit toolkit);
 };

EXPECT_TRUE(gate::FetcherHelper<TestWithToolkit>::CanTrigger(g));
auto tres = gate::FetcherHelper<TestWithToolkit>::Trigger(g);
EXPECT_EQ("aa", tres.order_id);
```

Общая схема решения



Общая схема решения



Часть 05

Пайплайн 

Пайплайн

```
using Pipeline = gate::Pipeline<
    fetchers::OrderCreatedEventFetcher,
    fetchers::PriceCalculatedEventFetcher,
    fetchers::PerformerFoundEventFetcher,
    fetchers::BuildPaymentFetcher>;
```

Пайплайн

```
template <typename... Fetchers>
struct Pipeline
{


};
```

Пайплайн

```
template <typename ... Fetchers>
struct Pipeline
{

 template <typename Fetcher>
 ProcessOne(gate &gate)
 {
   if (FetcherHelper<Fetcher>::CanTrigger(gate))
   {
     FetcherHelper<Fetcher>::Trigger(gate);
   }
 }

};
```

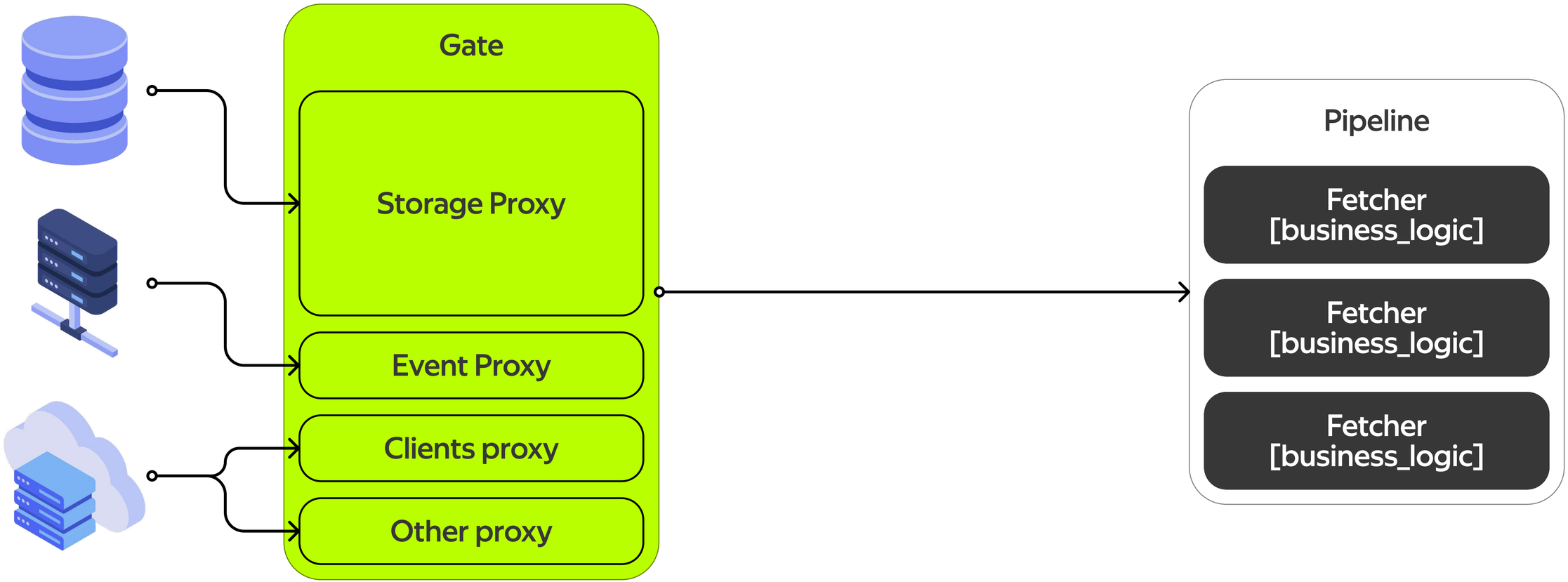
Пайплайн

```
template <typename... Fetchers>
struct Pipeline
{

 template <typename Fetcher>
 ProcessOne(gate &gate)
 {
   if (FetcherHelper<Fetcher>::CanTrigger(gate))
   {
     FetcherHelper<Fetcher>::Trigger(gate);
   }
 }


 void Process(gate &gate)
 {
   (ProcessOne<Fetchers>(), ... );
 }
};
```

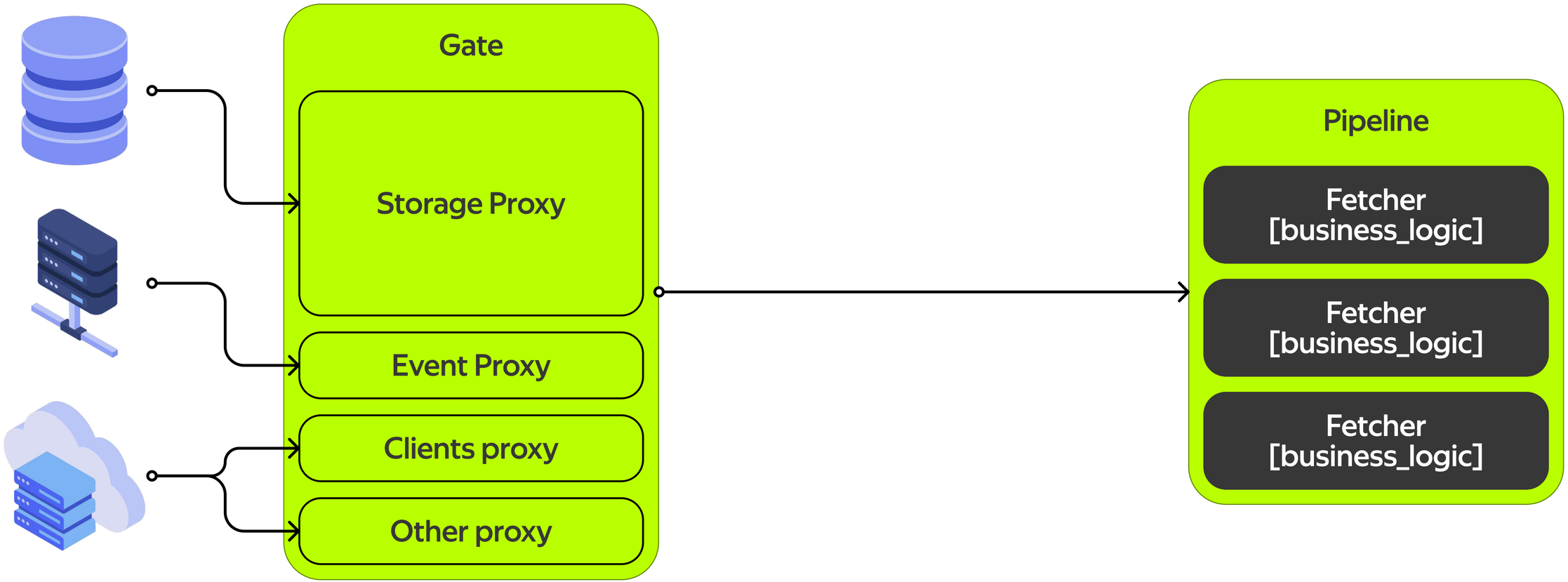
Пайплайн

```
template <typename... Fetchers>
struct Pipeline
{
  template <typename Fetcher>
  ProcessOne(gate &gate)
  {
    if (FetcherHelper<Fetcher>::CanTrigger(gate))
    {
      LOG() << "Starting " << type_name<Fetcher>();
      try
      {
        FetcherHelper<Fetcher>::Trigger(gate);
        LOG() << "Finished " << type_name<Fetcher>();
      }
      catch (const std::exception &e)
      {
        LOG() << "Fetcher " << type_name<Fetcher>()
              << " failed: " << e.what();
      }
    }
  }
};
```

Общая схема решения



Общая схема решения



Интерфейс в коде

```
struct OrderCreatedEventFetcher
{
 static OrderContext Fetch(
     const OrderCreatedEvent &event,
     ProfileClient &profileClient,
     ContractsClient &contractsClient);
};

struct PriceCalculatedEventFetcher
{
 static PricingContext Fetch(
     const PriceCalculatedEvent &event,
     PricingClient &pricingClient);
};

struct PerformerFoundEventFetcher
{
 static PerformerContext Fetch(
     const PerformerFoundEvent &event,
     TaxiParkClient &taxiParkClient,
     WalkersClient &walkersClient,
     ContractsClient &contractClient,
     const HostSettings &hostSettings);
};
```

```
struct BuildPaymentFetcher
{
 static Payment Fetch(
     const OrderContext &orderContext,
     const PricingContext &pricingContext,
     const PerformerContext &performerContext);
};




using Pipeline = gate::Pipeline<
   fetchers::OrderCreatedEventFetcher,
   fetchers::PriceCalculatedEventFetcher,
   fetchers::PerformerFoundEventFetcher,
   fetchers::BuildPaymentFetcher>;
```

А еще

А еще

Статические проверки

```
/home/pasukhov/arcadia/taxi/uservices/services/cargo-
finance/src/alternative_flow/claims/fetch_status_update.hpp:16:13: error: constraints not
satisfied for class template 'maybe' [with T = handlers::ClaimsStatusUpdatedContext]
      gate::maybe<handlers::ClaimsStatusUpdatedContext> last_status);
            ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/pasukhov/arcadia/taxi/uservices/services/cargo-finance/src/gate/utils.hpp:132:11:
note: because 'handlers::ClaimsStatusUpdatedContext' does not satisfy 'ContextDecayType'
template <ContextDecayType T>
          ^
/home/pasukhov/arcadia/taxi/uservices/services/cargo-finance/src/gate/utils.hpp:88:28:
note: because 'handlers::ClaimsStatusUpdatedContext' does not satisfy 'ContextType'
concept ContextDecayType = ContextType<T> && std::same_as<T, std::decay_t<T>>;
                           ^
/home/pasukhov/arcadia/taxi/uservices/services/cargo-finance/src/gate/utils.hpp:83:44:
note: because 'ToString(context.context_kind)' would be invalid: no member named
'context_kind' in 'handlers::ClaimsStatusUpdatedContext'
                        ToString(context.context_kind)
                                         ^

                                         ^
2 errors generated.
```

А еще

Статические проверки

```
      fetcher::Pipeline<alternative_flow_trucks::FetchInitTime,
                ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/pasukhov/arcadia/taxi/uservices/services/cargo-finance/src/gate/fetcher.hpp:185:10: note: because
'CorrectPipeline<alternative_flow::trucks::FetchInitTime, alternative_flow::trucks::SaveStartTime,
alternative_flow::trucks::FetchObligationSetup, alternative_flow::trucks::FetchInitiateTransfer,
alternative_flow::trucks::FetchSendShippingAct, alternative_flow::trucks::ValidatePayments,
alternative_flow::trucks::FetchActualContract, alternative_flow::trucks::FetchCarrierContext,
alternative_flow::trucks::FetchShipperContext, alternative_flow::trucks::BuildSum2Pay>' evaluated to
false
requires CorrectPipeline<Fetchers...>
         ^
/home/pasukhov/arcadia/taxi/uservices/services/cargo-finance/src/gate/fetcher_concepts.hpp:193:5: note:
because 'ValidPipelineOrder<alternative_flow::trucks::FetchInitTime,
alternative_flow::trucks::SaveStartTime, alternative_flow::trucks::FetchObligationSetup,
alternative_flow::trucks::FetchInitiateTransfer, alternative_flow::trucks::FetchSendShippingAct,
alternative_flow::trucks::ValidatePayments, alternative_flow::trucks::FetchActualContract,
alternative_flow::trucks::FetchCarrierContext, alternative_flow::trucks::FetchShipperContext,
alternative_flow::trucks::BuildSum2Pay>' evaluated to false
    ValidPipelineOrder<Fetchers...>;
    ^
```

А еще

Запуск только нужных обработчиков

```cpp
template <typename Fetcher>
static void ExecuteOne(auto& gate) {
  if (!FetcherHelper<Fetcher>::CanTrigger(gate)) {
    return;
  }

  if (!FetcherHelper<Fetcher>::ShouldTrigger(gate)) {
    return;
  }

  if (ProcessingDisabled<Fetcher>(gate)) {
    return;
  }

  try {
    auto res = FetcherHelper<Fetcher>::Fetch(gate);
    gate.emplace(std::move(res));
  } catch (const std::exception& e) {
    throw;
  }
}
```

А еще

Дополнительные настройки вызова

```
struct AlwaysRebuildFetcher {
 static constexpr bool kAlwaysRebuild = true;

 static TestContextCounter Fetch(const Maybe<TestContextCounter> last);
};
```

А еще

Слияние нескольких шлюзов

```
template <template <typename> typename... Ts>
struct FusionGate : public Ts<FusionGate<Ts...>>... {

 using Ts<FusionGate>::has...;
 using Ts<FusionGate>::get...;

};
```

И многое другое

Место для послесловия

Доставка

@cadovvl

Спасибо