Вложенные сущности по требованию в API на Spring Data JPA

Мирсаитов Григорий

НЛМК-Информационные технологии

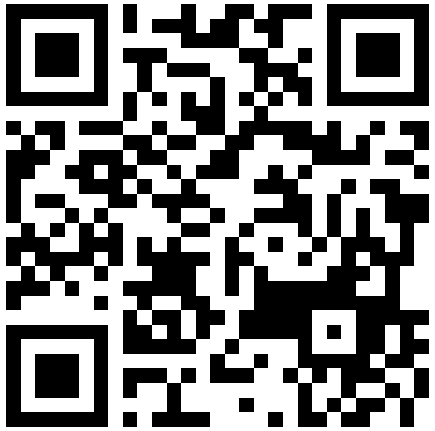
› О себе

Опыт разработки с 2014 года. Писал на ABAP, JavaScript, C#

В Java мире с 2019 года

TeamLead Java в НЛМК-Информационные технологии

Люблю алгоритмы и вопросы производительности



https://habr.com/ru/users/gligor/

Профиль на Хабр

https://www.linkedin.com/in/volan/

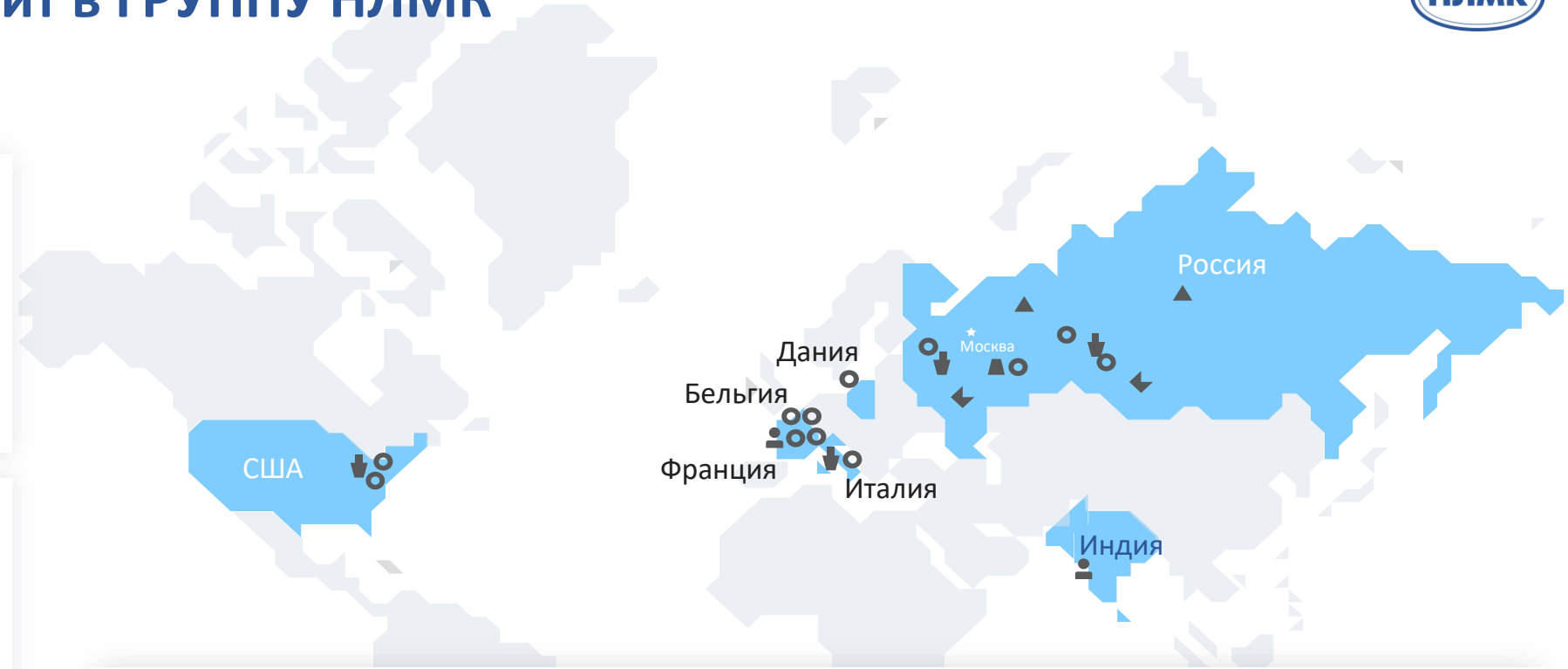
Профиль на linkedin

НЛМК-ИТ – входит в ГРУППУ НЛМК

20
предприятий
в 7 странах мира

ТОП-25
среди производителей стали в мире

50,8 тыс.
сотрудников



Сталь Группы НЛМК используется в знаковых проектах мирового уровня


Большой адронный коллайдер


Ветро-электростанции


Электро-автомобили


Конструкции зданий и сооружений

НЛМК-ИТ: цифровая металлургия

IT для управления производством, финансовых, логистических, кадровых и других систем по промышленным стандартам

>300
объем портфеля ИТ проектов/продуктов

>500
информационных систем

>1 000
сотрудников ИТ-функции

11
Гильдий по IT-компетенциям

Современный стек, собственная цифровая платформа и разработка как по продуктовой, так и по проектной и гибридной методологиям

ПРОФИЛЬНЫЕ СПЕЦИАЛИСТЫ

для всех этапов создания ИТ-продуктов:

- Бизнес- и системные аналитики
- Многоуровневая архитектура
- Разработчики
- Data Science
- QA-инженеры
- DevOps/SRE

› План

НЛМК


Постановка проблемы


GraphQL


Spring Data REST


Общий подход


Графы в Spring Data JPA и их ограничения


https://github.com/Cosium/spring-data-jpa-entity-graph

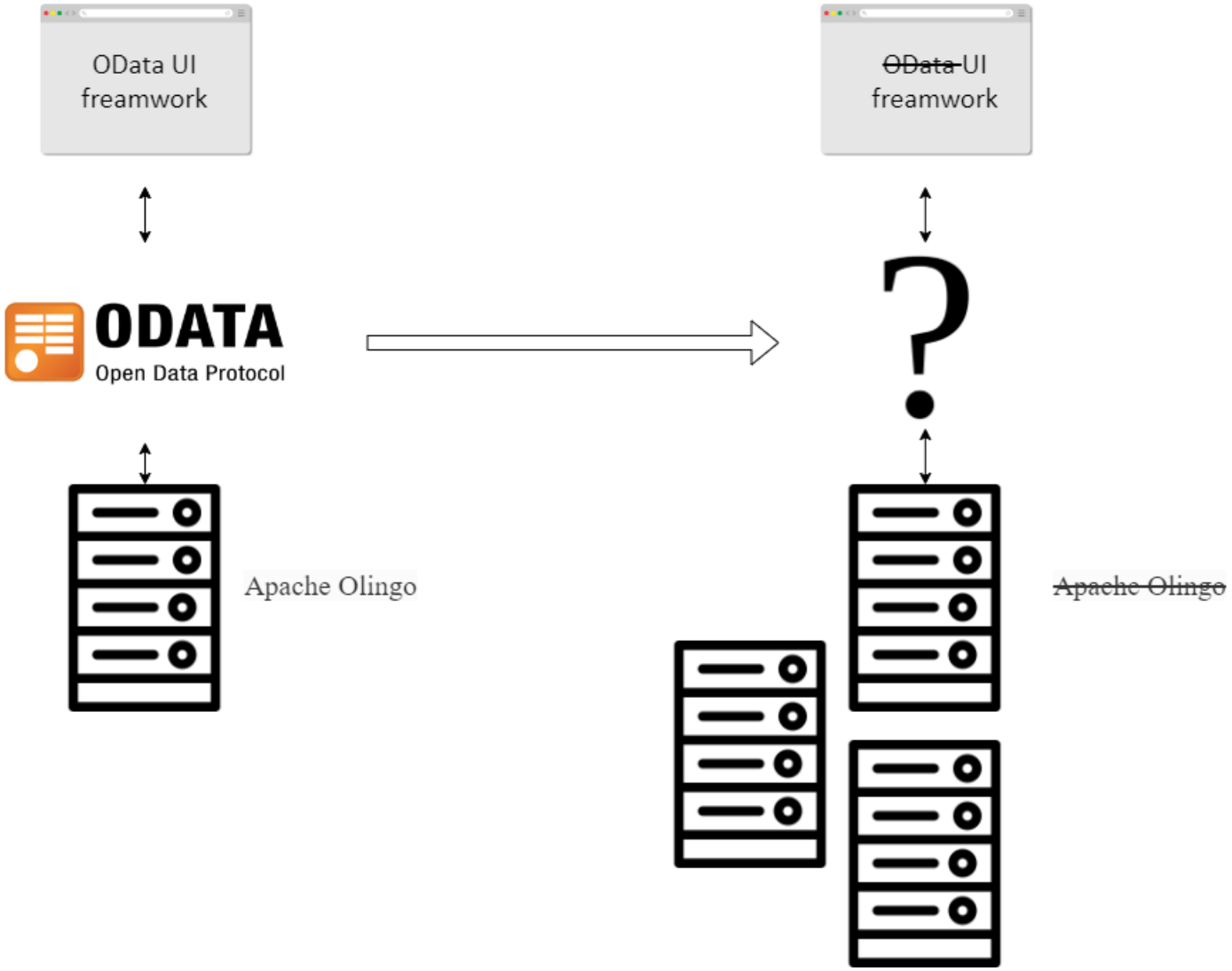
Описание промежуточного слоя конвертации


Соединение все в единую цепочку от запроса до выдачи данных

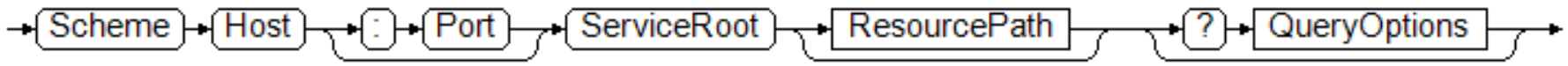

Проблемы


Плюсы и минусы решения

› Миграция системы



› OData синтаксис



https://services.odata.org/OData/OData.svc

service root URL

https://services.odata.org/OData/OData.svc/Category(1)/Products?$top=2&$orderby-name

service root URL | resource path | query options

НЛМК

https://services.odata.org/OData/OData.svc/Categories?$format=json


```
{
        "odata.metadata":"https://services.odata.org/OData/OData.svc/$metadata#Categories",
        "value":[{
                "ID":0,
                "Name":"Food"
        }, {
                "ID":1,
                "Name":"Beverages“
        }, {
                "ID":2,
                "Name":"Electronics“
        }]
}
```


Open Data Protocol

› OData $expand

НЛМК

https://services.odata.org/OData/OData.svc/Categories(0)?$expand=Products&$format=json


```
{
        "odata.metadata":"https://services.odata.org/OData/OData.svc/$metadata#Categories/@Element",
        "Products":[{
                "ID":0,
                "Name":"Bread",
                "Description":"Whole grain bread",
                "ReleaseDate":"1992-01-01T00:00:00",
                "DiscontinuedDate":null,
                "Rating":4,
                "Price":2.5
        }, ...],
        "ID":0,
        "Name":"Food"
}
```


Open Data Protocol

› Может GraphQL?

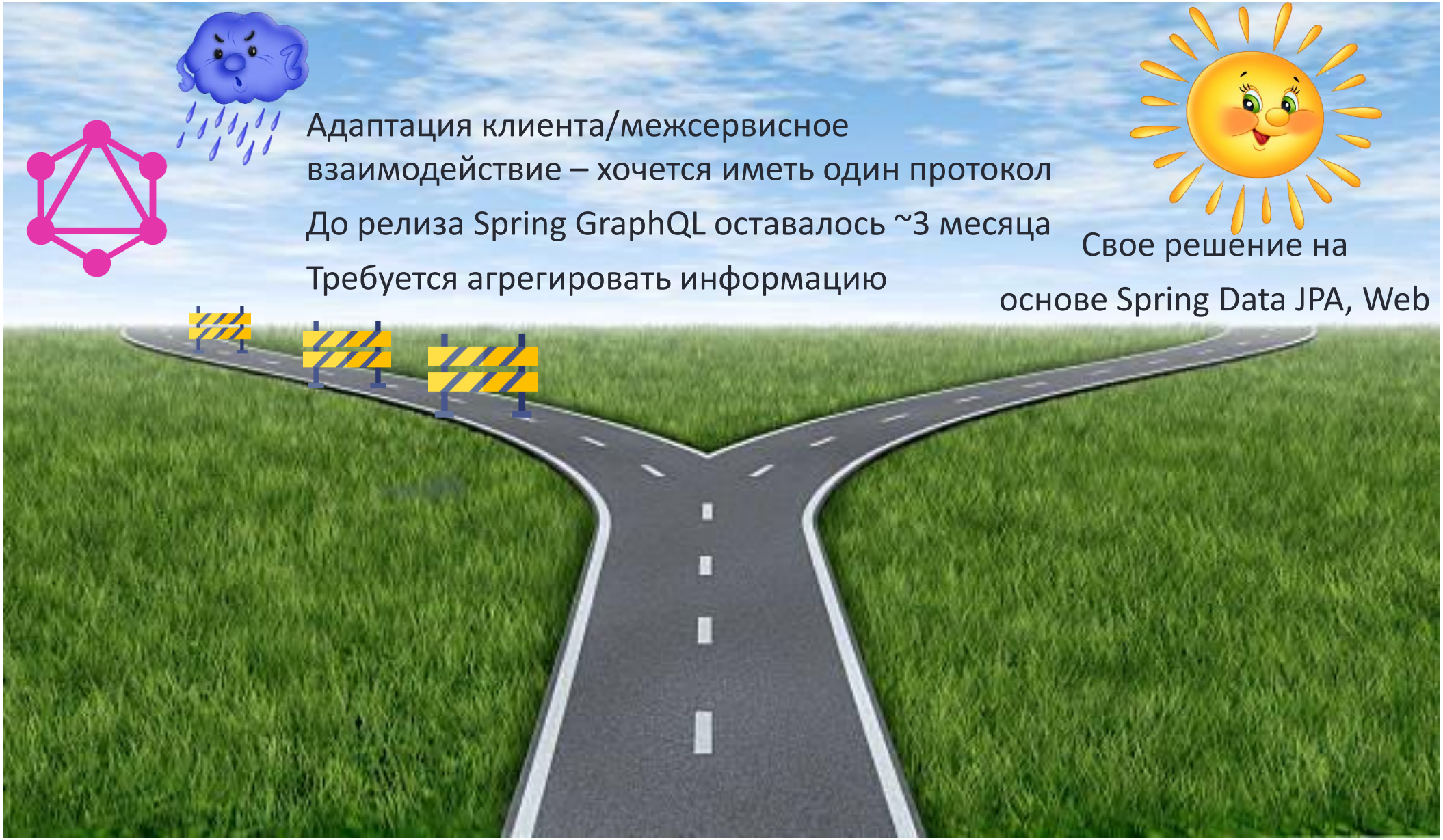
Сервис https://countries.trevorblades.com

Варианты запроса:

- Get запрос, параметр **query={ countries(filter: {code: {eq: "AO"}}) { code name }}**
- Post запрос, тело запроса **query { countries(filter: {code: {eq: "AO"}}) { code name }}**

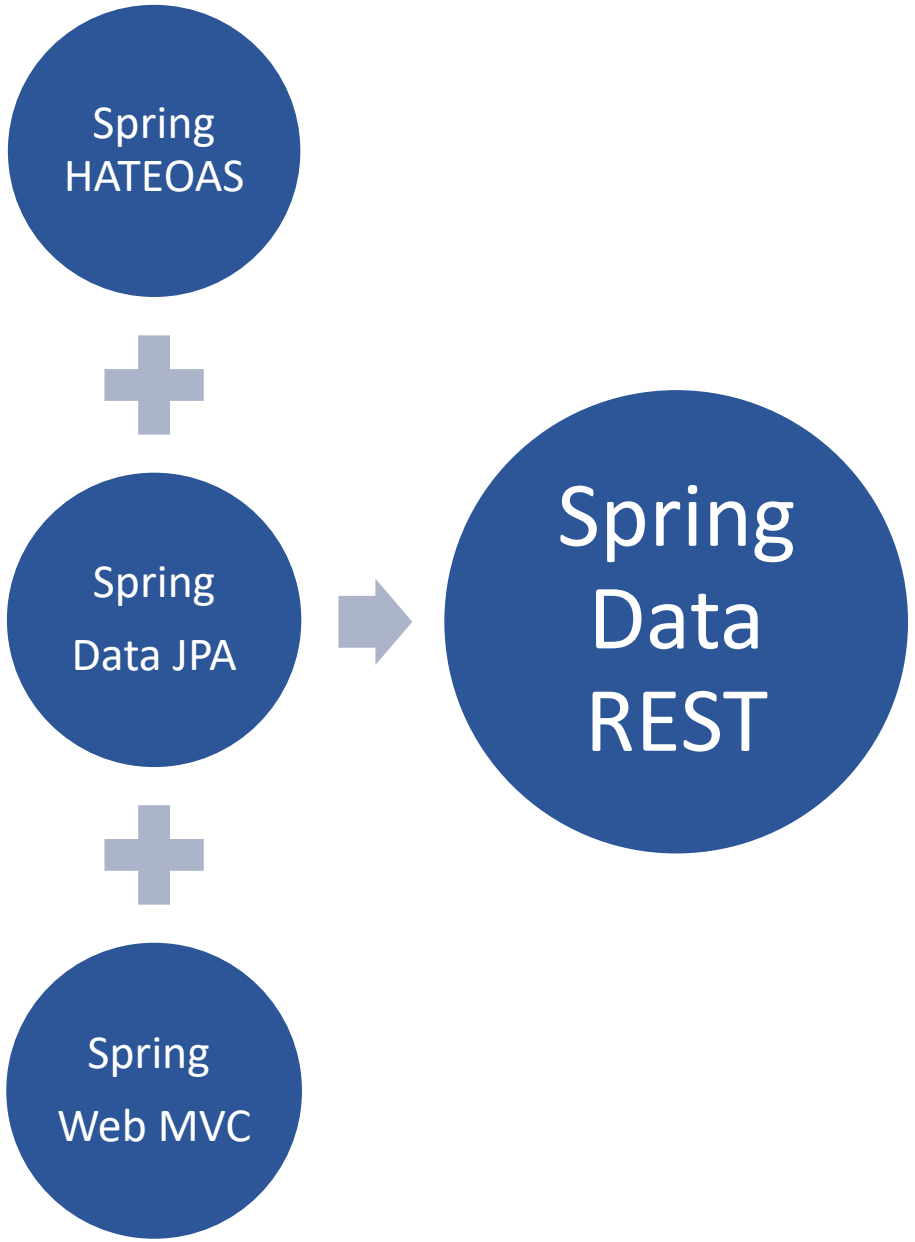

```
{
  "data": {
    "countries": [
      {
        "code": "AO",
        "name": "Angola"
      }
    ]
  }
}
```

› Мы на распутье



› А что со Spring?

НЛМК



НЛМК

```java
@Entity
public class Person {

    @Id
    @GeneratedValue
    private Long id;

    private String firstName, lastName;

    @OneToOne
    private Address address;

    ...

}
```

```java
public interface PersonRepository
        extends CrudRepository<Person, Long> {}

public interface AddressRepository
        extends CrudRepository<Address, Long> {}
```

```
{
        "firstName":"Frodo",
        "lastName":"Baggins",
        "_links":{
                "self":{
                        "href":http://localhost:8080/persons/1
                },
                "address":{
                        "href":http://localhost:8080/persons/1/address
                }
        }
}
```

НЛМК

Убираем репозиторий у Address


```
{
        "firstName":"Frodo",
        "lastName":"Baggins",
        "address":{
                "street":"Bag End",
                "state":"The Shire",
                "country":"Middle Earth“
        },
        "_links":{
                "self":{
                        "href":http://localhost:8080/persons/1
                }
        }
}
```

› Spring Data REST - проекции

А если не хотим включать address?

```
@Projection(name = "noAddresses", types = { Person.class })
interface NoAddresses {

    String getFirstName();

    String getLastName();
}
```

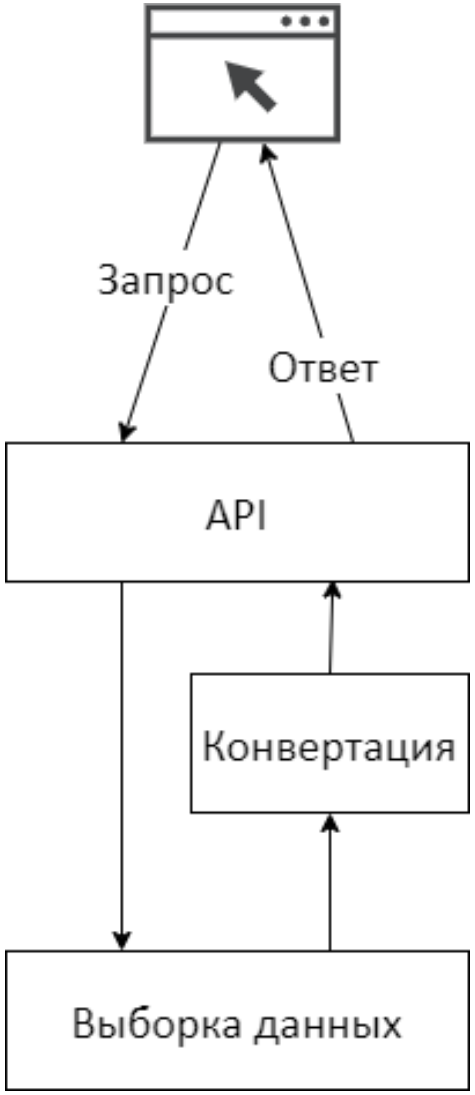
localhost:8080/persons/1?projection=noAddresses

› Spring Data REST

+ Быстрое разворачивание API

- Проекции можно искать, но надо знать названия

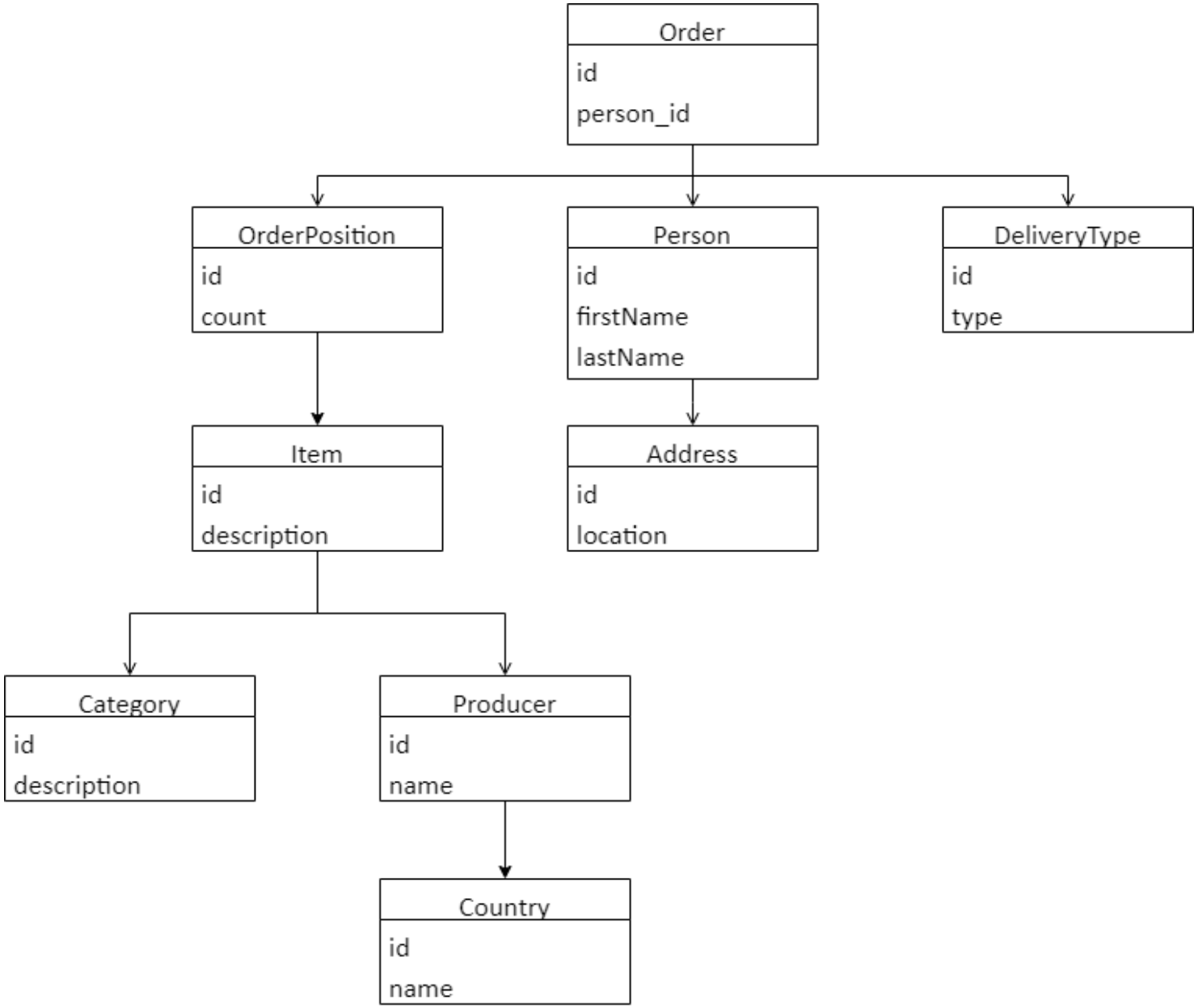
› Можно ли решить задачу с использованием Spring Data JPA?



НЛМК

https://github.com/Volan/expandconverter

› Очертим пример



НЛМК

```
@Entity
@NamedEntityGraph(name = "Person.address",
        attributeNodes = @NamedAttributeNode("address")
)
public class Person {

    @Id
    @GeneratedValue
    private Long id;

    private String firstName, lastName;

    @OneToOne
    private Address address;

}
```

```
public interface PersonRepository extends CrudRepository<Person,
Long> {

    @EntityGraph(value = "Person.address")
    Person findByFirstName(String firstName);

    @EntityGraph(attributePaths = {"address"})
    Person findByLastName(String lastName);

}
```

› Spring Data JPA пока статично

НЛМК

› JSR 338: Java Persistence API, Version 2.1

НЛМК

https://edelivery.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf

Entity Graphs:

- 3.7 Entity Graphs – интерфейсы, принципы работы
- 10.3 EntityGraph Annotations – аннотации

Статика

- NamedEntityGraph
- NamedAttributeNode
- NamedSubgraph

Динамика

- EntityGraph
- AttributeNode
- Subgraph

› https://github.com/Cosium/spring-data-jpa-entity-graph

НЛМК

Making JPA great again

› Совместимость со Spring Data JPA

НЛМК

https://github.com/Cosium/spring-data-jpa-entity-graph?tab=readme-ov-file#compatibility-matrix

Spring Data JPA EntityGraph совместима с Spring Data JPA начиная с версии 1.10.x

Помним про миграцию с javax на jakarta в Spring 6.

› Добавляем динамизма

НЛМК

```
<dependency>
   <groupId>com.cosium.spring.data</groupId>
   <artifactId>spring-data-jpa-entity-graph</artifactId>
   <version>${spring-data-jpa-entity-graph.version}</version>
</dependency>
```

› Добавляем динамизма

НЛМК

```
<dependency>
   <groupId>com.cosium.spring.data</groupId>
   <artifactId>spring-data-jpa-entity-graph</artifactId>
   <version>${spring-data-jpa-entity-graph.version}</version>
</dependency>
```

```
@EnableJpaRepositories(repositoryFactoryBeanClass = EntityGraphJpaRepositoryFactoryBean.class)
```

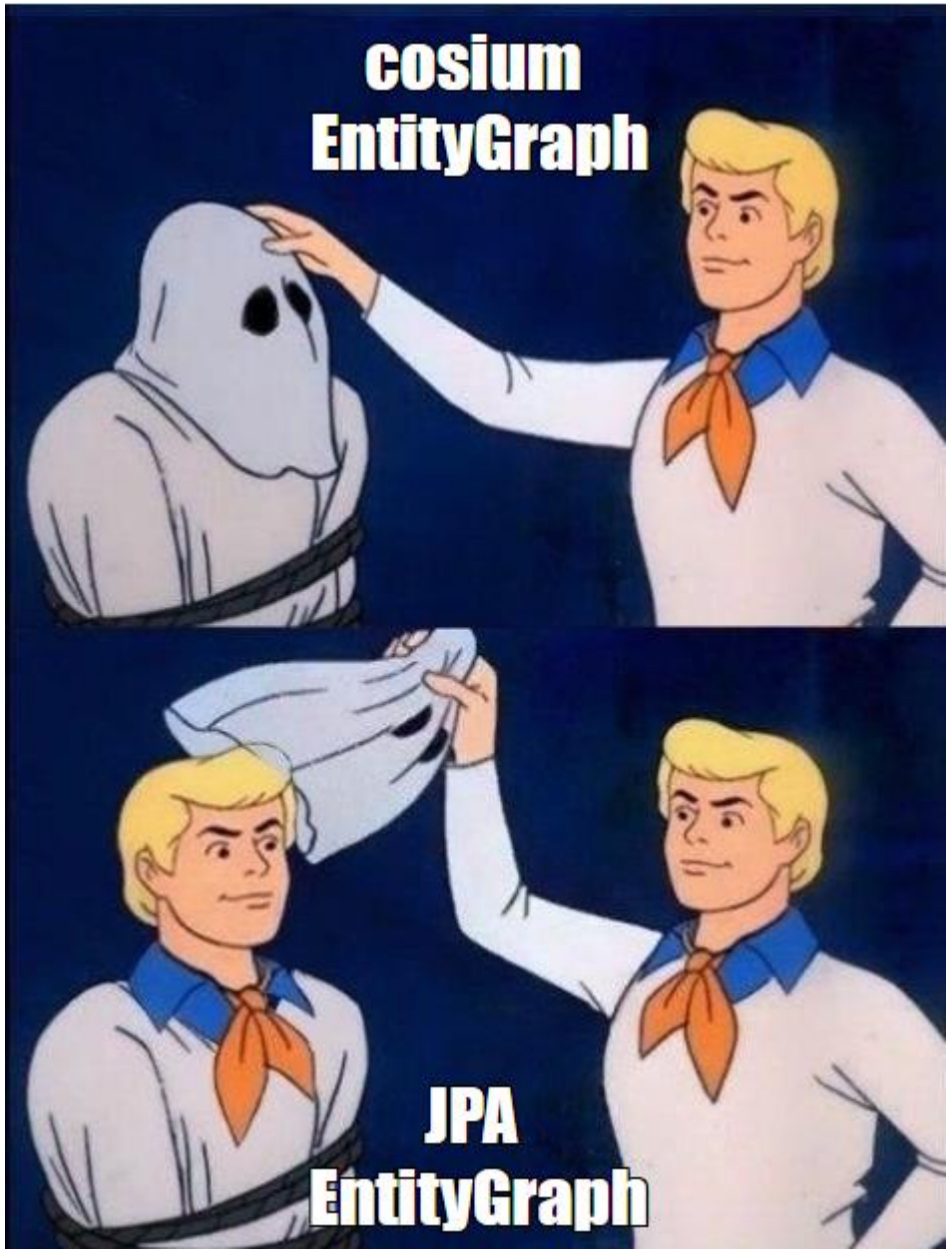
› Добавляем динамизма

НЛМК

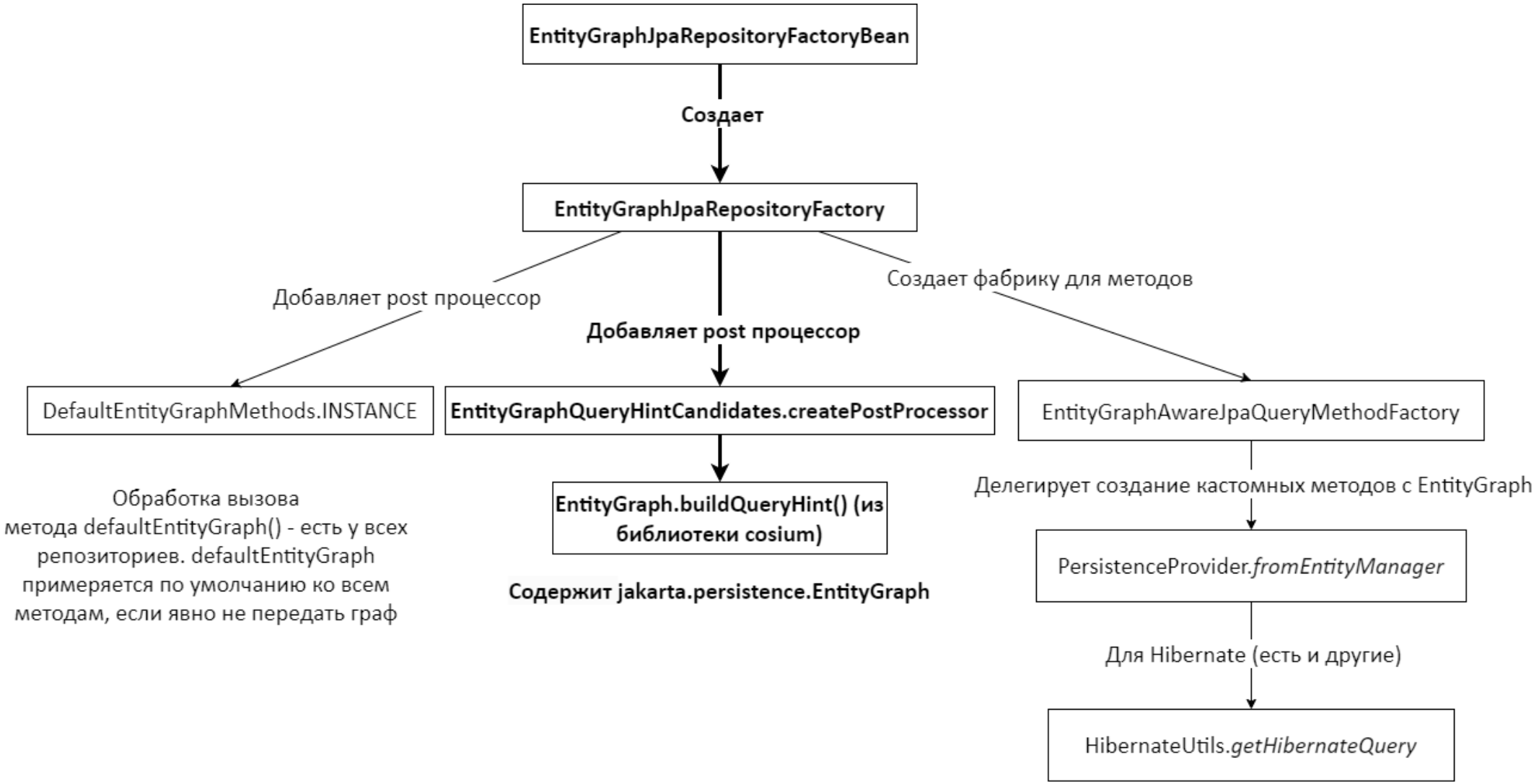
```
<dependency>
   <groupId>com.cosium.spring.data</groupId>
   <artifactId>spring-data-jpa-entity-graph</artifactId>
   <version>${spring-data-jpa-entity-graph.version}</version>
</dependency>
```

```
@EnableJpaRepositories(repositoryFactoryBeanClass = EntityGraphJpaRepositoryFactoryBean.class)
```

```
@Repository
public interface OrderRepository extends EntityGraphPagingAndSortingRepository<Order, Long> {
}
```


cosium
EntityGraph
JPA
EntityGraph

› А что внутри?


EntityGraphJpaRepositoryFactoryBean
Создает
EntityGraphJpaRepositoryFactory
Добавляет post процессор
Добавляет post процессор
Создает фабрику для методов
DefaultEntityGraphMethods.INSTANCE
EntityGraphQueryHintCandidates.createPostProcessor
EntityGraphAwareJpaQueryMethodFactory
Обработка вызова метода defaultEntityGraph() - есть у всех репозиториев. defaultEntityGraph примеряется по умолчанию ко всем методам, если явно не передать граф
EntityGraph.buildQueryHint() (из библиотеки cosium)
Содержит jakarta.persistence.EntityGraph
Делегирует создание кастомных методов с EntityGraph
PersistenceProvider.fromEntityManager
Для Hibernate (есть и другие)
HibernateUtils.getHibernateQuery

› Проверяем

НЛМК

```
@Entity
@Getter
@Table(name = "orders")
@NoArgsConstructor
public class Order {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Long id;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "person_id", referencedColumnName = "id")
    private Person person;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "order")
    private List<OrderPosition> positions;

}
```

› Проверяем

НЛМК

Page<Order> users = **orderRepository.findAll(Pageable.ofSize(10))**;

Hibernate: select o1_0.id,o1_0.person_id from orders o1_0 offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0

› Проверяем

НЛМК

```
Page<Order> users = orderRepository.findAll(Pageable.ofSize(10));

Hibernate: select o1_0.id,o1_0.person_id from orders o1_0 offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0

Page<Order> users = orderRepository.findAll(Pageable.ofSize(10),
                                            DynamicEntityGraph.loading(List.of("person")));

Hibernate: select o1_0.id,p1_0.id,p1_0.address_id,p1_0.first_name,p1_0.last_name from orders
o1_0 left join persons p1_0 on p1_0.id=o1_0.person_id offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0
```

› Проверяем

НЛМК

```
Page<Order> users = orderRepository.findAll(Pageable.ofSize(10));
```

```
Hibernate: select o1_0.id,o1_0.person_id from orders o1_0 offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0
```

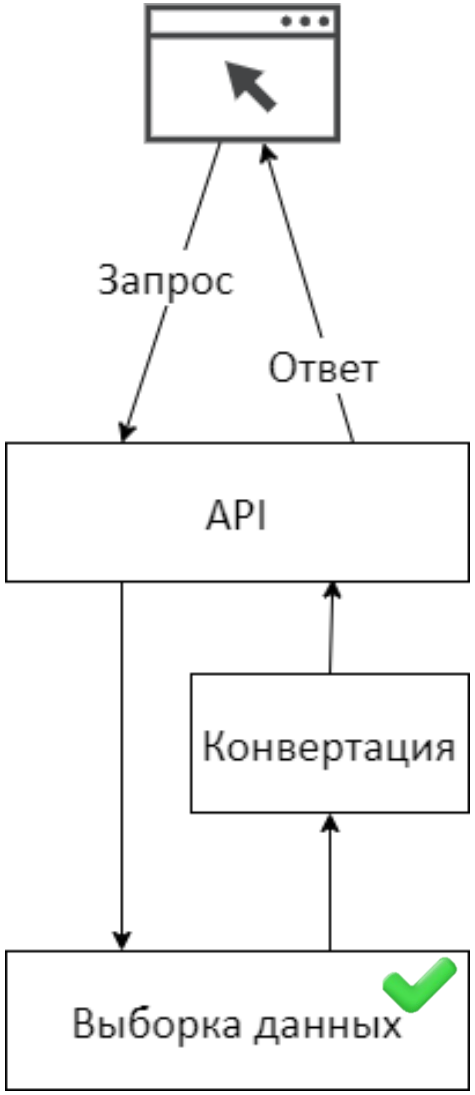
```
Page<Order> users = orderRepository.findAll(Pageable.ofSize(10),
                    DynamicEntityGraph.loading(List.of("person")));
```

```
Hibernate: select o1_0.id,p1_0.id,p1_0.address_id,p1_0.first_name,p1_0.last_name from orders
o1_0 left join persons p1_0 on p1_0.id=o1_0.person_id offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0
```

```
Page<Order> users = orderRepository.findAll(Pageable.ofSize(10),
                    DynamicEntityGraph.loading(List.of("person.address")));
```

```
Hibernate: select o1_0.id,p1_0.id,a1_0.id,a1_0.location,p1_0.first_name,p1_0.last_name from
orders o1_0 left join persons p1_0 on p1_0.id=o1_0.person_id left join addresses a1_0 on
a1_0.id=p1_0.address_id offset ? rows fetch first ? rows only
Hibernate: select count(o1_0.id) from orders o1_0
```

› На этапе выборки данных требования раскрытия данных учитываем



› Что должен делать конвертер?

НЛМК

```
public class Order {
    private Long id;
    private Person person;
    private List<OrderPosition> positions;
}
```

простые поля перекладываем 1 к 1

по запросу

по запросу

```
public class OrderDto {
    private Long id;
    private PersonDto person;
    private List<OrderPositionDto> positions;
}
```

› Стандартный Interface Converter<S,T>

НЛМК

```
@FunctionalInterface
public interface Converter<S, T> {

    @Nullable
    T convert(S source); //Не знает о том, что мы хотим раскрыть

    …

}
```

› Создаем свой интерфейс

НЛМК

```java
public interface ExpandConverter<S, T> {

    String SEPARATOR = ".";

    /**
     * Метод для перекладывания полей сущности S в сущность T
     * @param source исходная сущность
     * @param expandNames имена для раскрытия через SEPARATOR
     * @return целевая сущность
     */
    T convert(S source, Set<String> expandNames);

}
```

Первая имплементация - наивная

Дано – **expand=person.address,positions.order**

Как понять какие сущности открыть на текущем уровне и передать дальше?

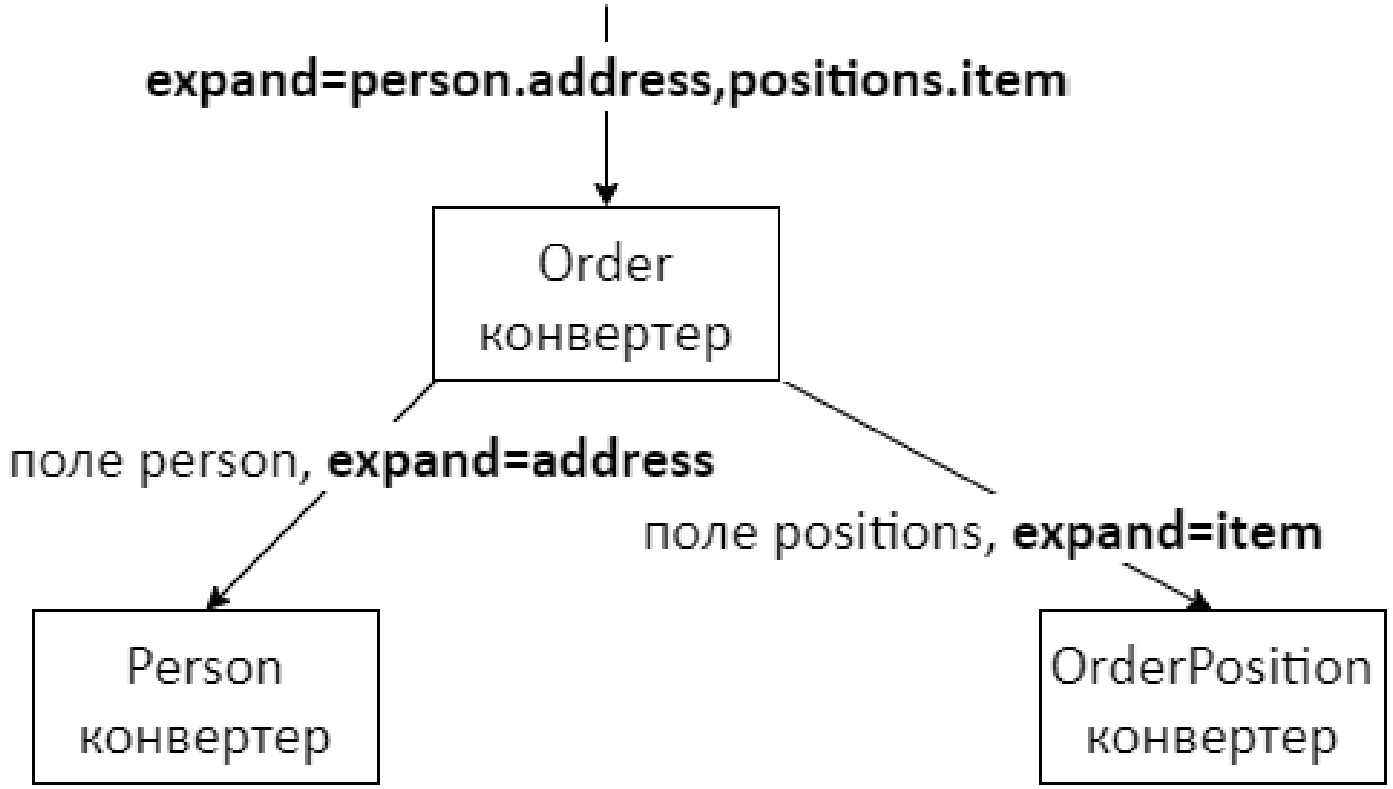
Отрезаем первый элемент до “.” и все что слева это ключ, справа – это путь в рамках этой сущности.

Пример

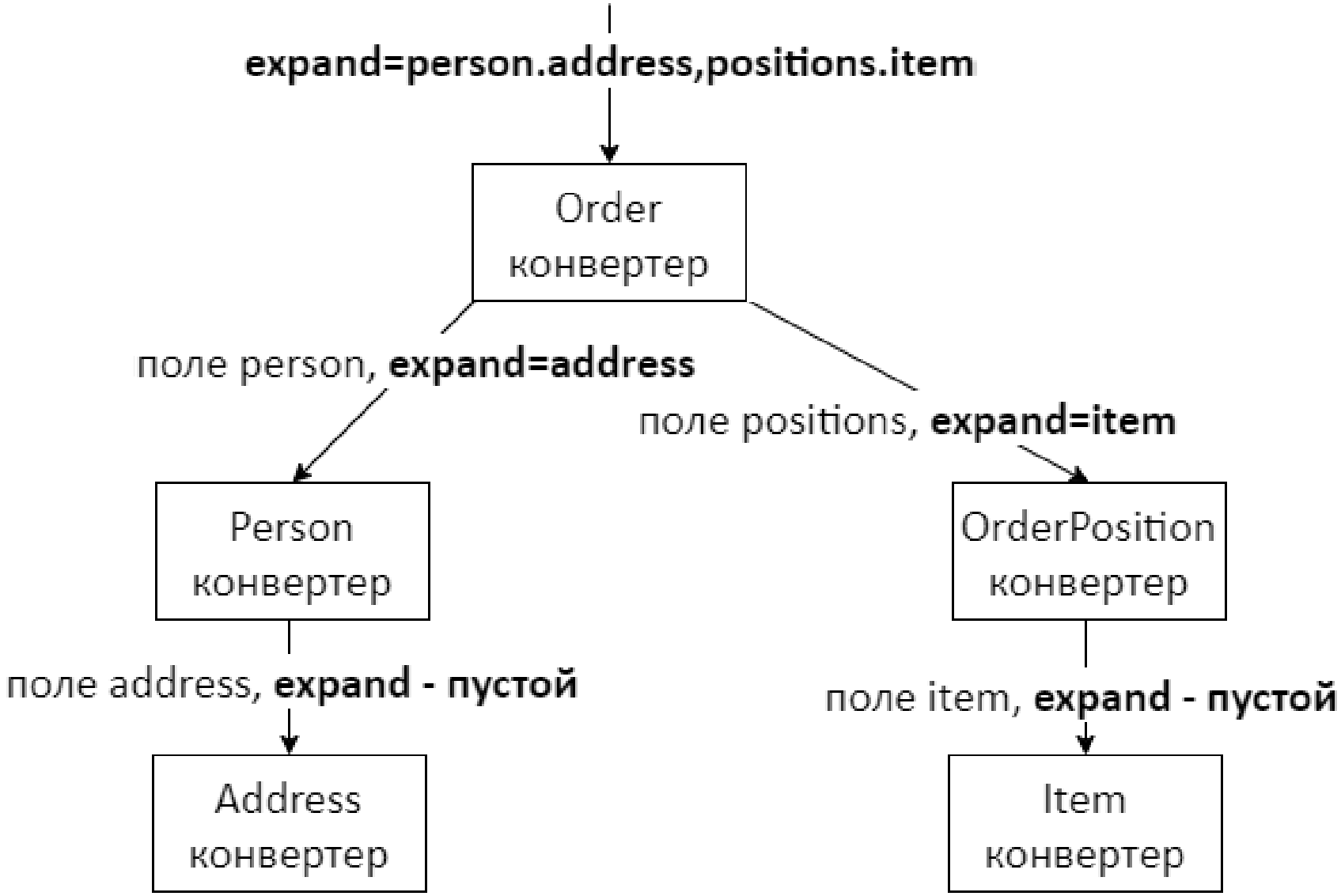
Поле	expand
positions	order
person	address

› Схема конвертации

› Схема конвертации



› Схема конвертации



НЛМК

```
// формируем карту для полей
Map<String, Set<String>> nestedMap = new HashMap<>();
for (String expand : expandNames) {
    String[] keyAndSubPath = getFirstPathAndSubPath(expand);
    String value = keyAndSubPath.length > 1 ? keyAndSubPath[1] : "";
    nestedMap.compute(keyAndSubPath[0], (key, oldValues) -> {
        if (Objects.isNull(oldValues)) {
            Set<String> values = new HashSet<>();
            values.add(value);
            return values;
        } else {
            oldValues.add(value);
            return oldValues;
        }
    });
}
```

› Реализация – наивная

НЛМК

```
public class OrderConverter extends AbstractExpandConverter<Order, OrderDto> {
    private final ExpandConverter<Person, PersonDto> personDtoExpandConverter;
    private final ExpandConverter<OrderPosition, OrderPositionDto> orderPositionDtoExpandConverter;
    private final ExpandConverter<DeliveryType, DeliveryTypeDto> deliveryTypeDtoExpandConverter;









}
```

› Реализация – наивная

НЛМК

```
public class OrderConverter extends AbstractExpandConverter<Order, OrderDto> {
    private final ExpandConverter<Person, PersonDto> personDtoExpandConverter;
    private final ExpandConverter<OrderPosition, OrderPositionDto> orderPositionDtoExpandConverter;
    private final ExpandConverter<DeliveryType, DeliveryTypeDto> deliveryTypeDtoExpandConverter;

    @PostConstruct
    public void init() {
        Map<String, ExpandConverter> expandConverterMap = new HashMap<>();
        expandConverterMap.put("person", personDtoExpandConverter);
        expandConverterMap.put("positions", orderPositionDtoExpandConverter);
        expandConverterMap.put("deliveryType", deliveryTypeDtoExpandConverter);
        super.setExpandConverterMap(expandConverterMap);
    }



}
```

› Реализация – наивная

НЛМК

```
public class OrderConverter extends AbstractExpandConverter<Order, OrderDto> {
    private final ExpandConverter<Person, PersonDto> personDtoExpandConverter;
    private final ExpandConverter<OrderPosition, OrderPositionDto> orderPositionDtoExpandConverter;
    private final ExpandConverter<DeliveryType, DeliveryTypeDto> deliveryTypeDtoExpandConverter;

    @PostConstruct
    public void init() {
        Map<String, ExpandConverter> expandConverterMap = new HashMap<>();
        expandConverterMap.put("person", personDtoExpandConverter);
        expandConverterMap.put("positions", orderPositionDtoExpandConverter);
        expandConverterMap.put("deliveryType", deliveryTypeDtoExpandConverter);
        super.setExpandConverterMap(expandConverterMap);
    }

    protected OrderDto convert(Order source) {
        OrderDto orderDto = new OrderDto();
        orderDto.setId(source.getId());
        return orderDto;
    }
}
```

› Можно проверять пути!

НЛМК

```
/**
 * Проверить, что запрашиваемые поля для раскрытия есть в сущности. Проверяется карта переданная при создании
 * @param expandNames имена для раскрытия через SEPARATOR
 * @exception ExpandParameterException
 */
void checkExpandNames(Set<String> expandNames);
```

› Пример

В конвертере заказов

```
@PostConstruct
public void init() {
    Map<String, ExpandConverter> expandConverterMap = new HashMap<>();
    expandConverterMap.put("person", personDtoExpandConverter);
    expandConverterMap.put("positions", orderPositionDtoExpandConverter);
    expandConverterMap.put("deliveryType", deliveryTypeDtoExpandConverter);
    super.setExpandConverterMap(expandConverterMap);
}
```

person.item ✔

person.items ✘

В конвертере позиций

```
@PostConstruct
public void init() {
    Map<String, ExpandConverter> expandConverterMap = new HashMap<>();
    expandConverterMap.put("item", itemDtoExpandConverter);
    super.setExpandConverterMap(expandConverterMap);
}
```

НЛМК

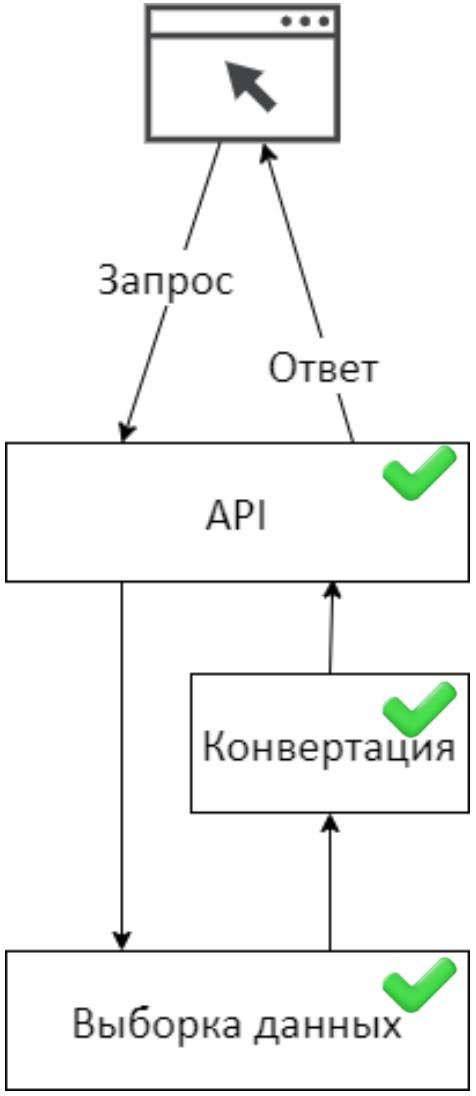
```
public interface OrderController {

    @RequestMapping("api/v1/orders")
    Page<OrderDto> getOrders(@PageableDefault Pageable pageable,
                             @RequestParam(value = "expand", required = false) Set<String> expands);

}
```

› Реализуем сервис

НЛМК

```
@Service
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {

    private final OrderRepository orderRepository;

    private final ExpandConverter<Order, OrderDto> orderDtoExpandConverter;

    …

    @Override
    public Page<OrderDto> getOrdersV1(Pageable pageable, Set<String> expands) {
        Page<Order> users = (Objects.isNull(expands) || expands.isEmpty()  ?
                orderRepository.findAll(pageable) :
                orderRepository.findAll(pageable, DynamicEntityGraph.loading(List.copyOf(expands))));
        return users.map(user -> orderDtoExpandConverter.convert(user, expands));
    }

}
```

› Весь путь учитывает expand параметр



› Проверяем

/api/v1/orders?size=10


```
{
   "content": [
      {
         "id": 1,
         "person": null,
         "positions": null
      }, …
   ],…
}
```

› Проверяем

/api/v1/orders?size=10&**expand=person**

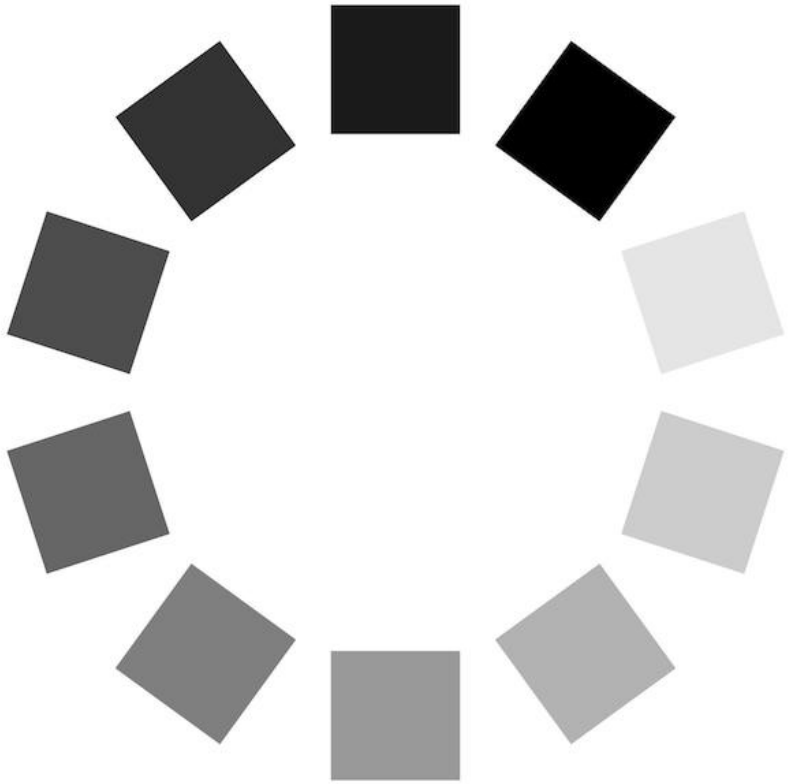

```
{
   "content": [
      {
         "id": 1,
         "person": {
            "id": 39,
            "firstName": "person39",
            "lastName": "person39",
            "address": null
         },
         "positions": null
      }, ...
   ],...
}
```

› Проверяем

/api/v1/orders?size=10&**expand=person.address**

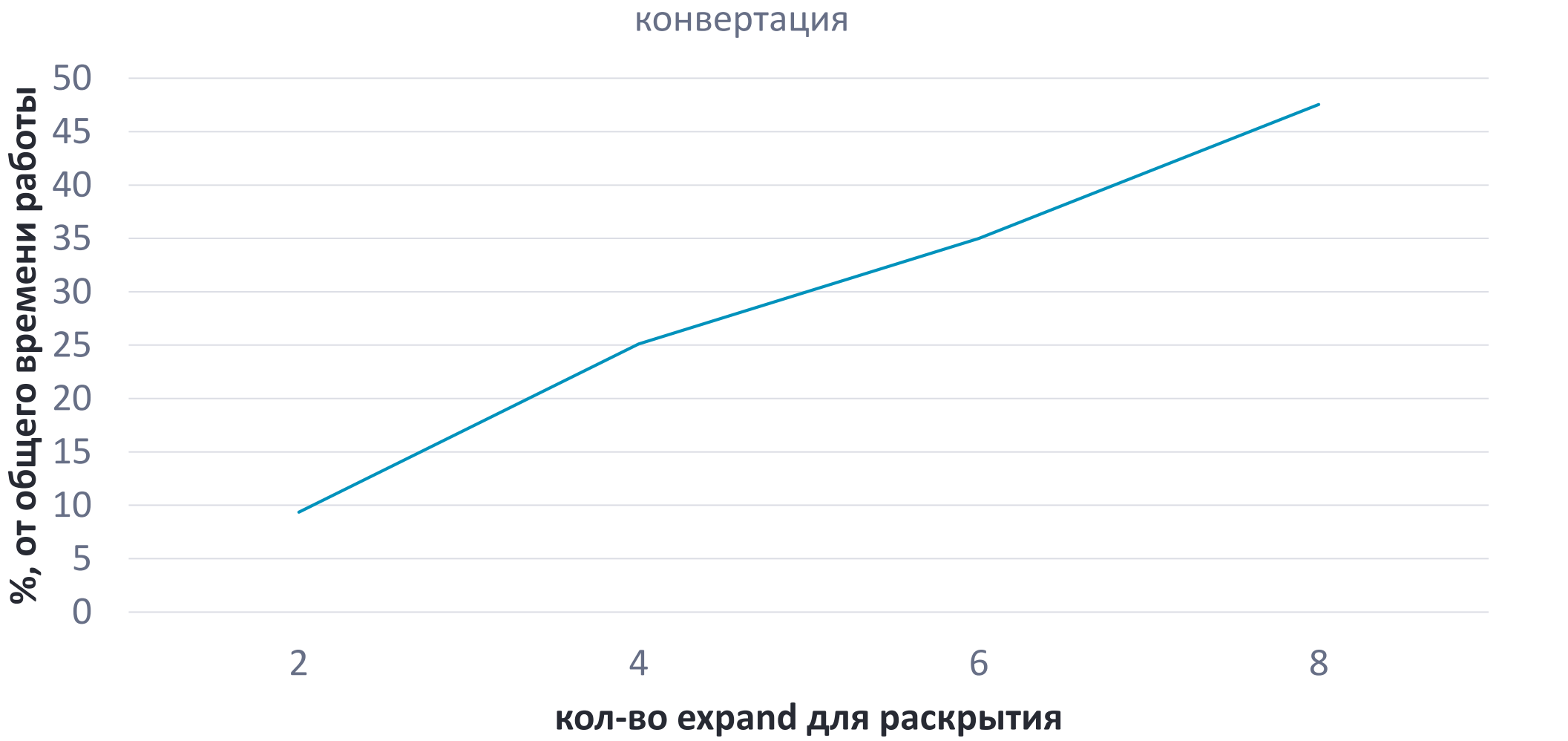

```
{
   "content": [
      {
         "id": 1,
         "person": {
            "id": 39,
            "firstName": "person39",
            "lastName": "person39",
            "address": {
               "id": 39,
               "location": "address39"
            }
         },
         "positions": null
      }, , ...
   ],...
}
```

› Запускаем на больших объемах



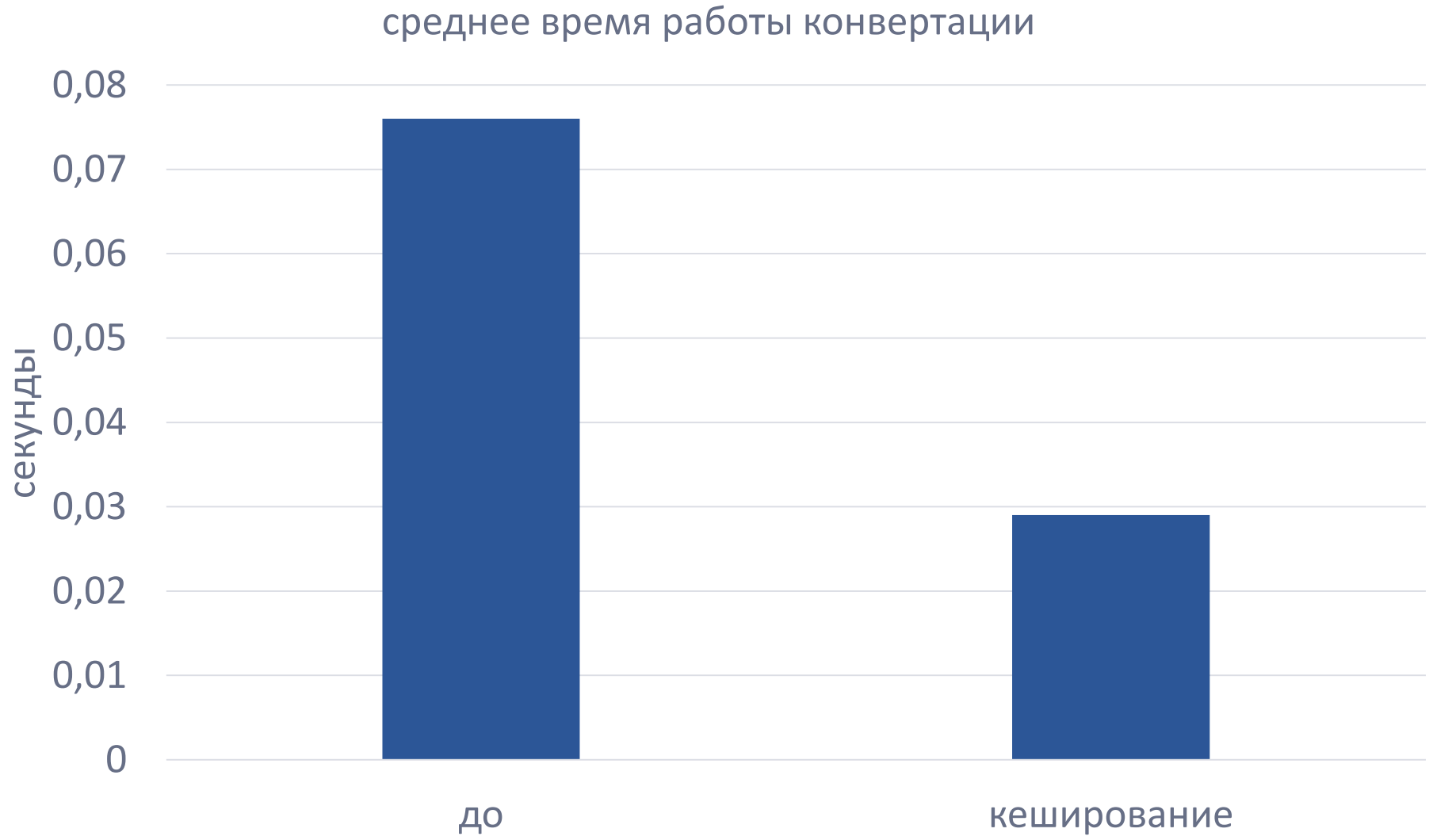
› А давайте замерим

› Замерим время работы методов конвертации



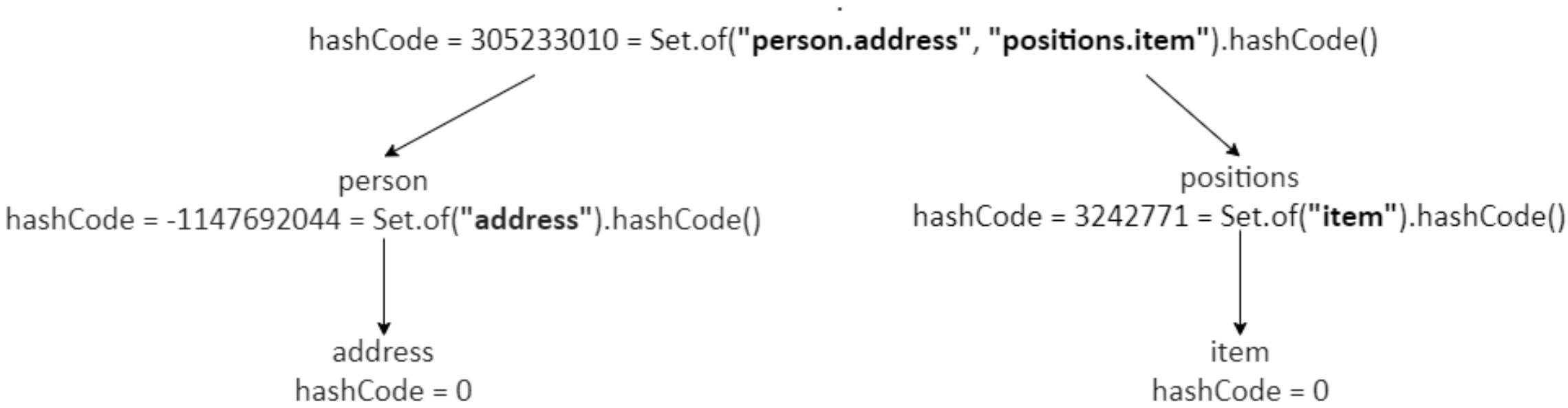
› Что с рефлексией?

Кешируем get/set методы на уровне конвертора. Пример на 2000 заказах



› А со строками как? Создаем структуру Path (дерево пути)

expand=**person.address,positions.item**



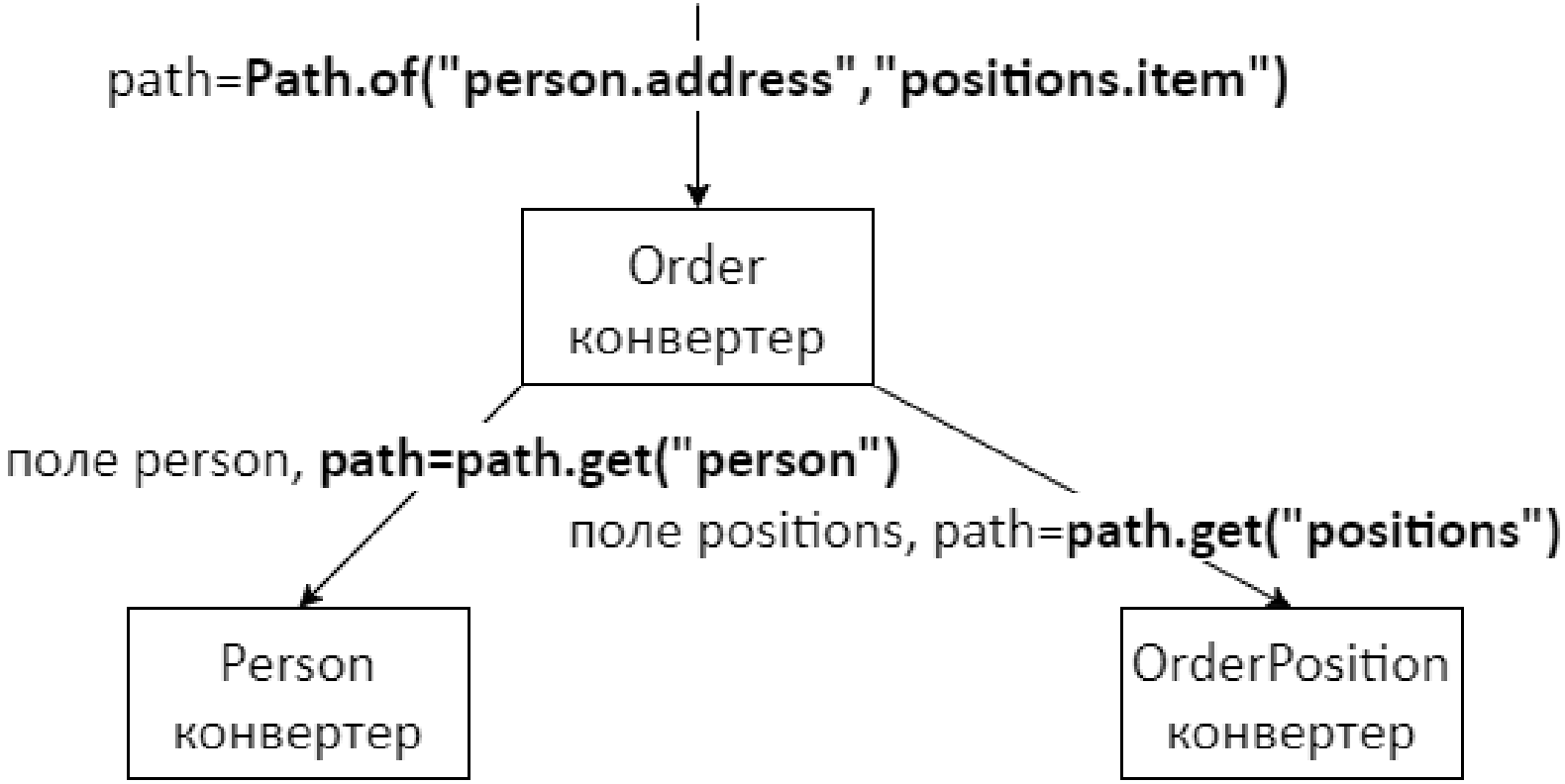
› Добавляем новый метод в интерфейс

НЛМК

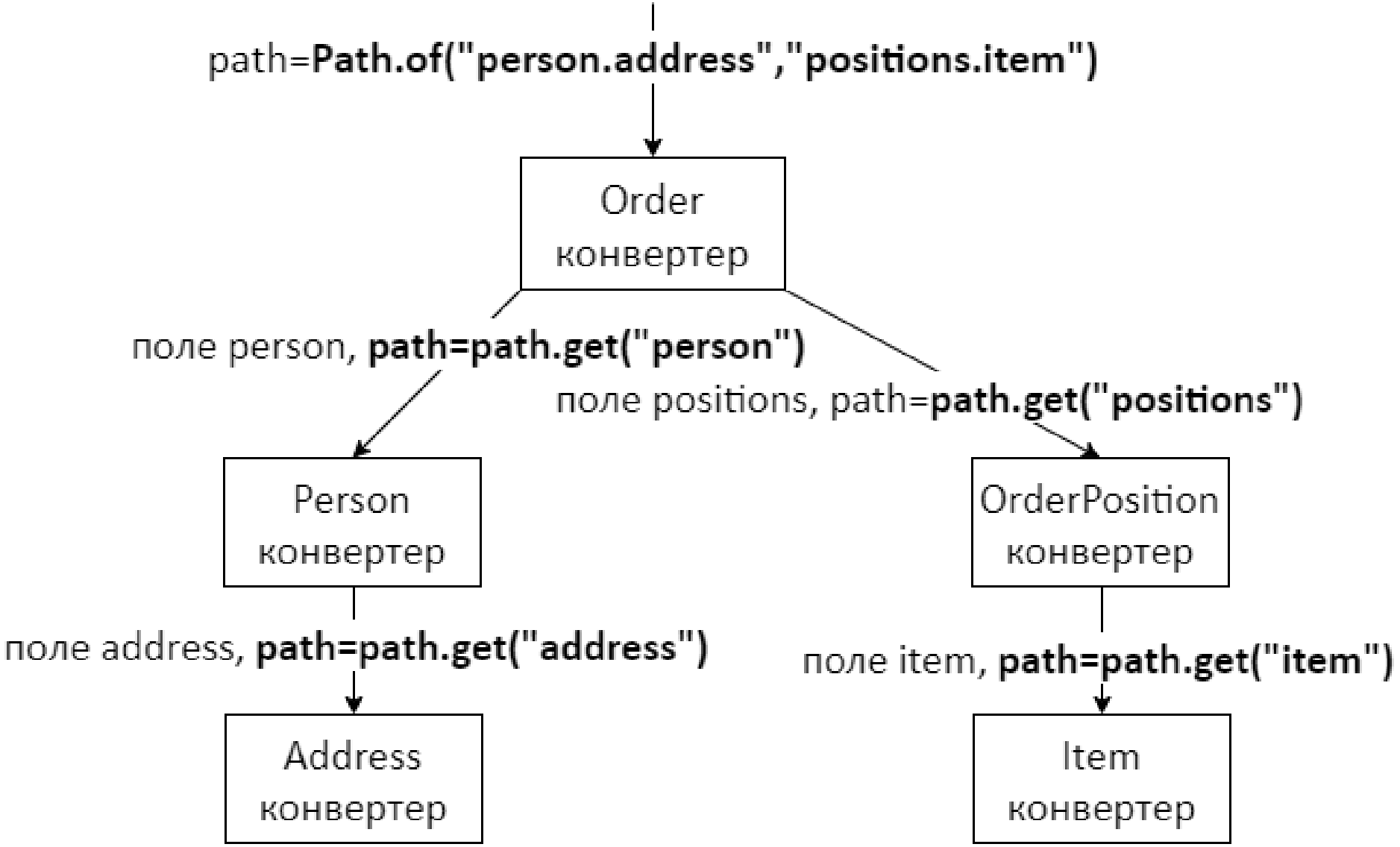
```
public interface ExpandConverter<S, T> {

    …

    /**
     * Метод для перекладывания полей сущности S в сущность T
     * @param source исходная сущность
     * @param path пути для раскрытия
     * @return целевая сущность
     */
    T convert(S source, Path path);

    …

}
```

› Новая схема конвертации

› Новая схема конвертации



› Новая схема конвертации



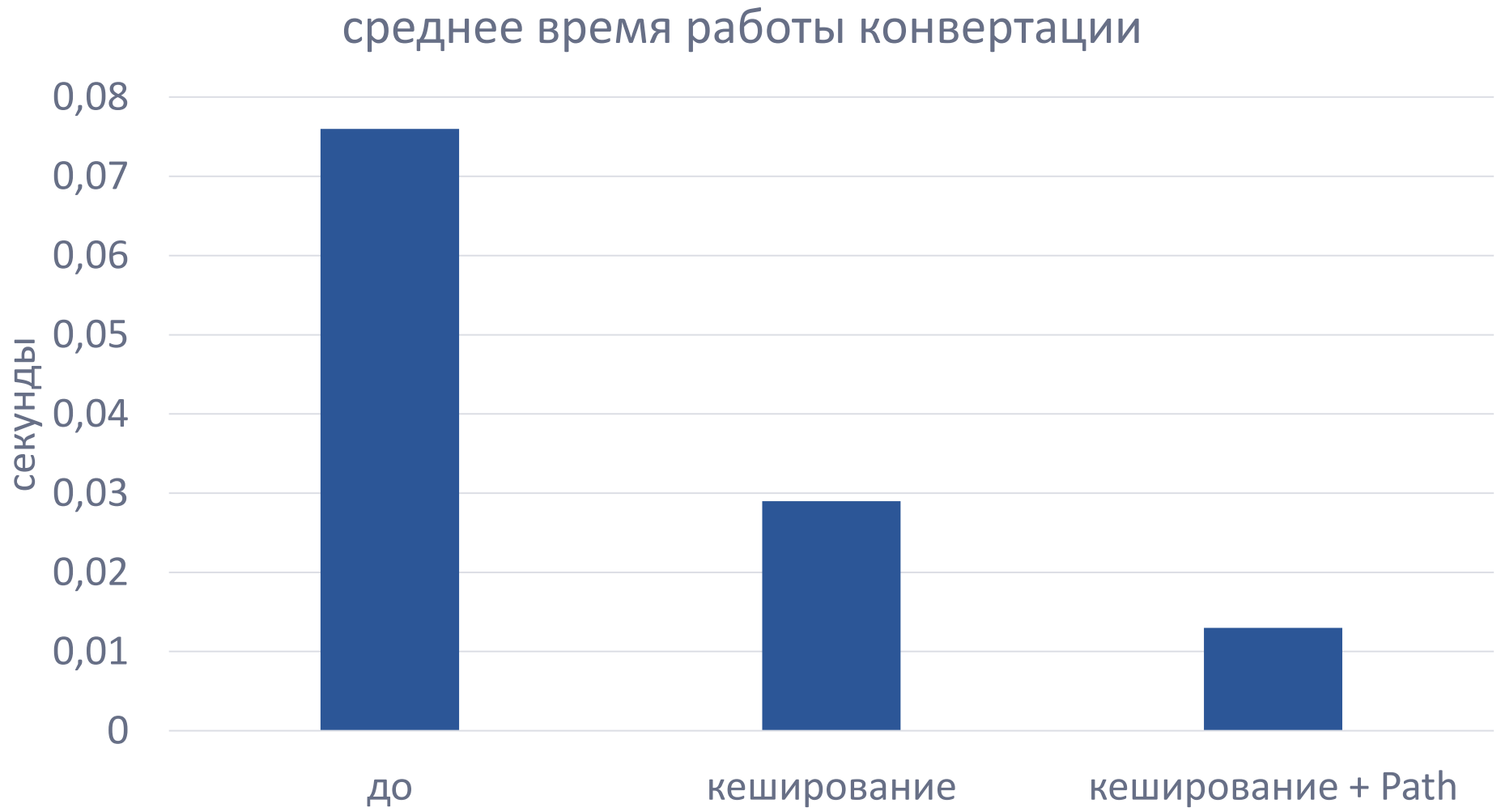
› Меняем логику работы в сервисе

НЛМК

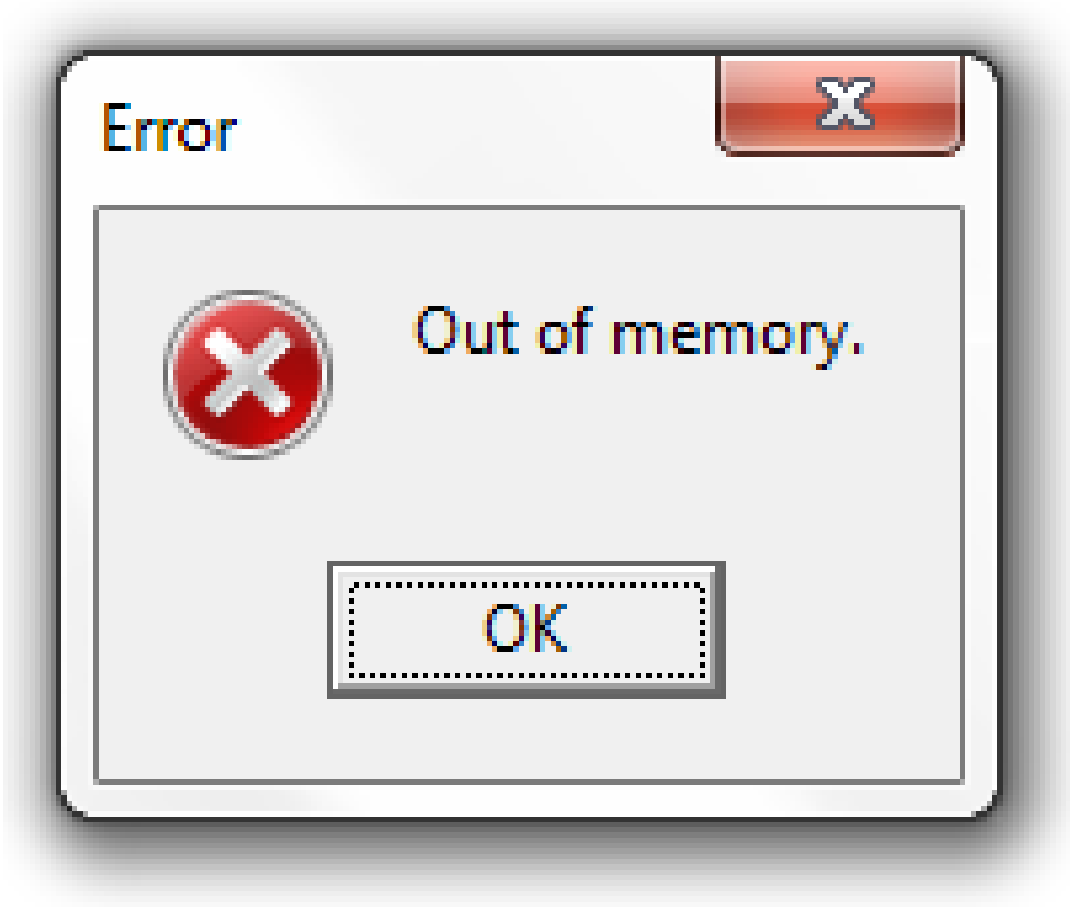
```
@Override
public Page<OrderDto> getOrdersV3(Pageable pageable, Set<String> expands) {
   Page<Order> orders = (Objects.isNull(expands) || expands.isEmpty()  ?
         orderRepository.findAll(pageable) :
         orderRepository.findAll(pageable, DynamicEntityGraph.loading(List.copyOf(expands))));
   return convert(orders, orderConverterBest, expands);
}


public static <S,T> Page<T> convert(Page<S> source, AbstractExpandConverter<S,T> converter,
Set<String> expandNames) {
   if (Objects.isNull(source)) {
      return null;
   }
   Path path = converter.createPath(expandNames, ImmutablePath::new);
   return source.map(s -> converter.convert(s, path));
}
```

› Сравниваем конвертацию

2000 заказов, expand=person.address,positions.item.category,positions.item.producer.country,deliveryType



› Запускаем снова



› Что на уровне выборки?

НЛМК

/api/v2/orders?size=2000&expand=person.address,positions.item.category,deliveryType

```
select
o1_0.id,dt1_0.id,dt1_0.type,p1_0.id,a1_0.id,a1_0.location,p1_0.first_name,p1_0.last_name,p2_0.order
_id,p2_0.id,p2_0.count,i1_0.id,c1_0.id,c1_0.description,i1_0.description from orders o1_0 left join
delivery_types dt1_0 on dt1_0.id=o1_0.delivery_type_id left join persons p1_0 on
p1_0.id=o1_0.person_id left join addresses a1_0 on a1_0.id=p1_0.address_id left join order_positions
p2_0 on o1_0.id=p2_0.order_id left join items i1_0 on i1_0.id=p2_0.item_id left join categories c1_0 on
c1_0.id=i1_0.category_id
```

id bigint	location character varying (100)	first_name character varying (100)	last_name character varying (100)	order_id bigint	id bigint	count bigint	id bigint	id bigint	description character varying (100)
39	address39	person39	person39	1	1	33	15	1	category1
39	address39	person39	person39	1	2	30	100	4	category4
39	address39	person39	person39	1	3	34	58	9	category9
39	address39	person39	person39	1	4	69	62	2	category2
39	address39	person39	person39	1	5	70	12	7	category7
88	address88	person88	person88	2	6	84	98	7	category7

› Какие проблемы?

	id bigint	location character varying (100)	first_name character varying (100)	last_name character varying (100)	order_id bigint	id bigint	count bigint	id bigint	id bigint	description character varying (100)
1	39	address39	person39	person39	1	1	33	15	1	category1
2	39	address39	person39	person39	1	2	30	100	4	category4
3	39	address39	person39	person39	1	3	34	58	9	category9
4	39	address39	person39	person39	1	4	69	62	2	category2
5	39	address39	person39	person39	1	5	70	12	7	category7
6	88	address88	person88	person88	2	6	84	98	7	category7

Данные не меняются, а множатся при выборке из-за позиций.

В общем случае количество строк будет – A*B*C и т.д., где A,B,C – количество сущностей по разворачиваемому полю

› Как решаем? Вариант 1

```
// Запрашиваем данные по заказу
Set<String> orderExpands = Set.of("person.address", "deliveryType");
Page<Order> orders = orderRepository.findAll(pageable,
        DynamicEntityGraph.loading(List.copyOf(orderExpands)));
Page<OrderDto> orderDtos = convert(orders, orderConverterBest, orderExpands);
```

› Как решаем? Вариант 1

```
// Запрашиваем данные по заказу
Set<String> orderExpands = Set.of("person.address", "deliveryType");
Page<Order> orders = orderRepository.findAll(pageable,
        DynamicEntityGraph.loading(List.copyOf(orderExpands)));
Page<OrderDto> orderDtos = convert(orders, orderConverterBest, orderExpands);

// Запрашиваем данные по позиции
Set<Long> orderIds = orderDtos.stream().map(OrderDto::getId).collect(Collectors.toSet());
Set<String> orderPositionExpands = Set.of("item.category");
List<OrderPosition> orderPositions = orderPositionRepository.findByOrderIdIn(orderIds,
        DynamicEntityGraph.loading(List.copyOf(orderPositionExpands)));
Map<Long, List<OrderPositionDto>> orderPositionDtos = convert(orderPositions,
        orderPositionConverterBest, orderPositionExpands).stream()
        .collect(groupingBy(position -> position.getOrderId(), mapping(Function.identity(), toList())));
```

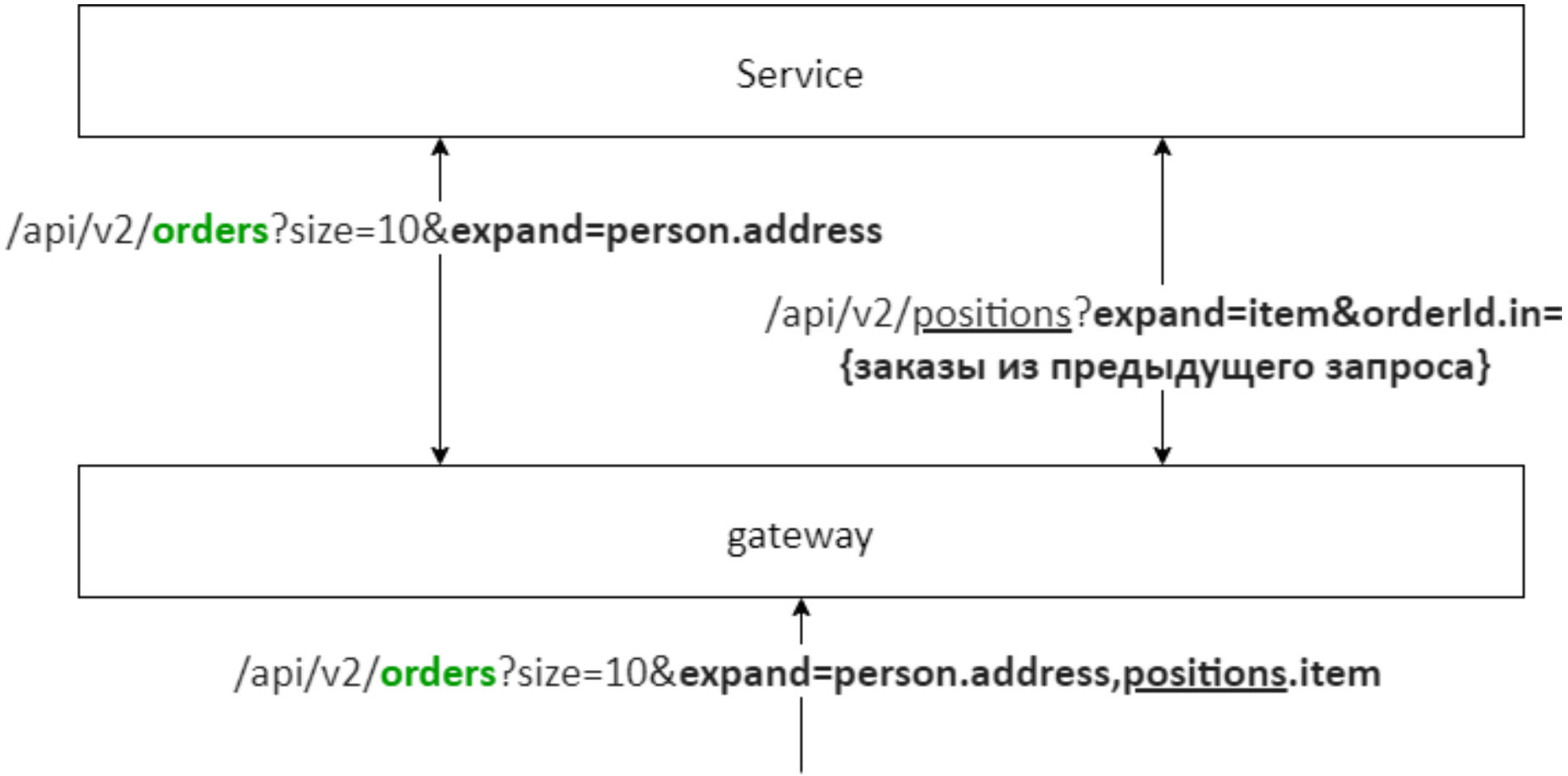
› Как решаем? Вариант 1

НЛМК

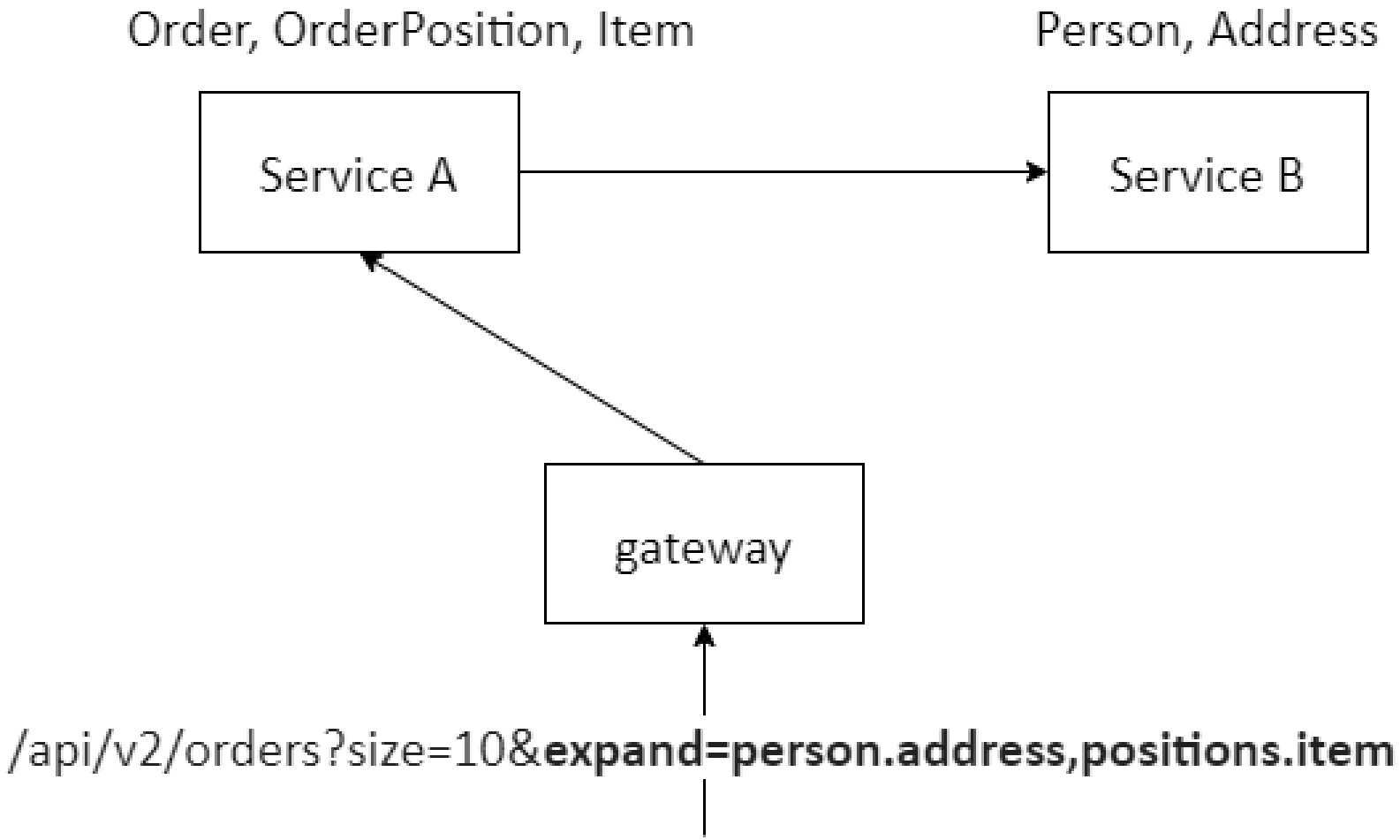
```
// Запрашиваем данные по заказу
Set<String> orderExpands = Set.of("person.address", "deliveryType");
Page<Order> orders = orderRepository.findAll(pageable,
        DynamicEntityGraph.loading(List.copyOf(orderExpands)));
Page<OrderDto> orderDtos = convert(orders, orderConverterBest, orderExpands);

// Запрашиваем данные по позиции
Set<Long> orderIds = orderDtos.stream().map(OrderDto::getId).collect(Collectors.toSet());
Set<String> orderPositionExpands = Set.of("item.category");
List<OrderPosition> orderPositions = orderPositionRepository.findByOrderIdIn(orderIds,
        DynamicEntityGraph.loading(List.copyOf(orderPositionExpands)));
Map<Long, List<OrderPositionDto>> orderPositionDtos = convert(orderPositions,
        orderPositionConverterBest, orderPositionExpands).stream()
        .collect(groupingBy(position -> position.getOrderId(), mapping(Function.identity(), toList())));

// Объединяем
orderDtos.map(orderDto -> {
   orderDto.setPositions(orderPositionDtos.get(orderDto.getId()));
   return orderDto;
});
```


потребние памяти во время работы
60
50
40
30
20
10
0
Мбайт
без сплита
со сплитом

› Ещё варианты?



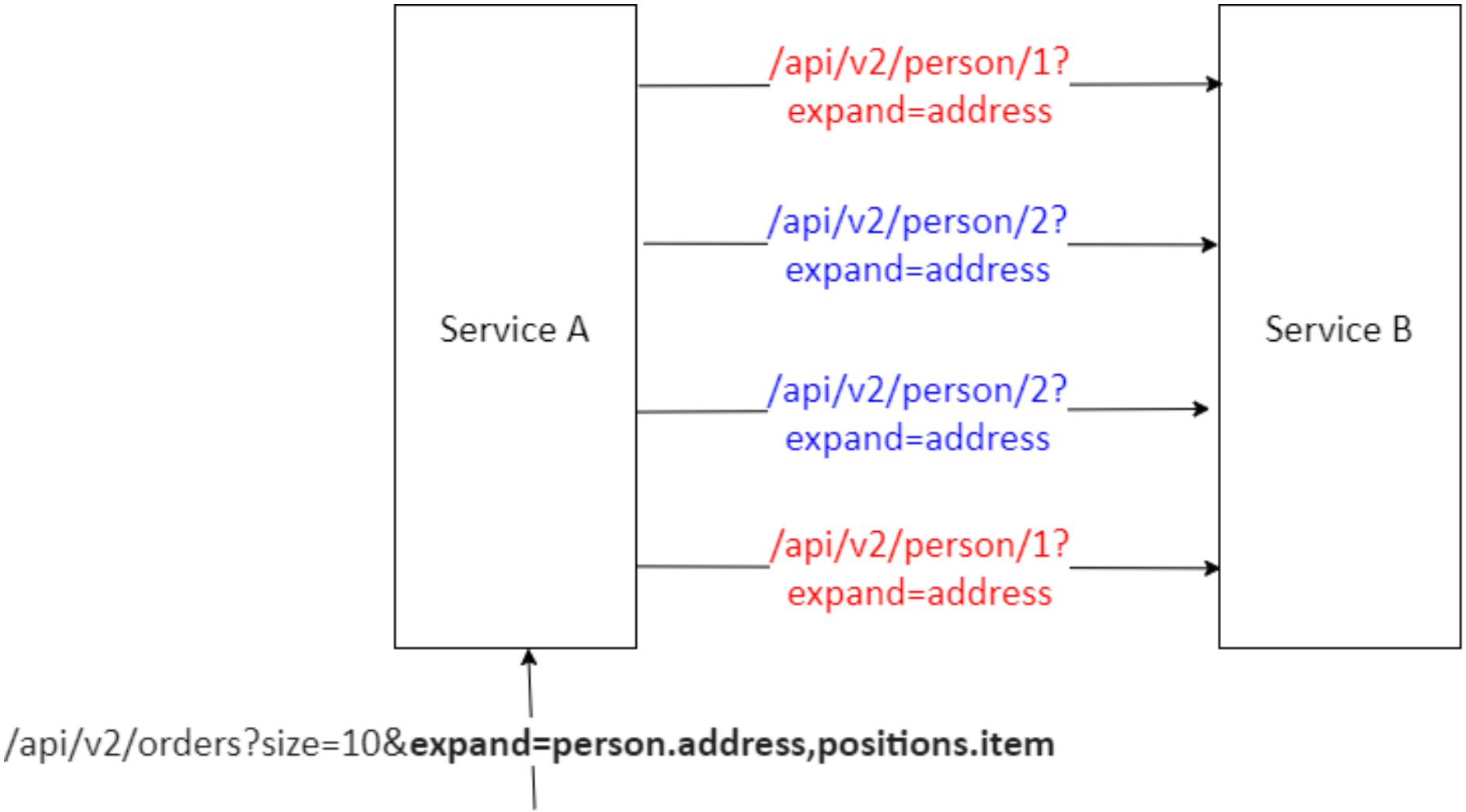
› Новый вызов – разбиваем сервис на 2



› Конвертер person в сервисе A

```
@Component
@RequiredArgsConstructor
public class PersonDtoConverter extends AbstractExpandConverter<String, PersonDto> {
    ...
    /*
     * делаем в методе вызов во внешний сервис
    */
    @Override
    public PersonDto convertSource(String personId, Path path) {
        return personService.getPerson(personId, path.getPaths(false));
    }
    ...
}
```

› Дубляжи в рамках одного запроса от A к B

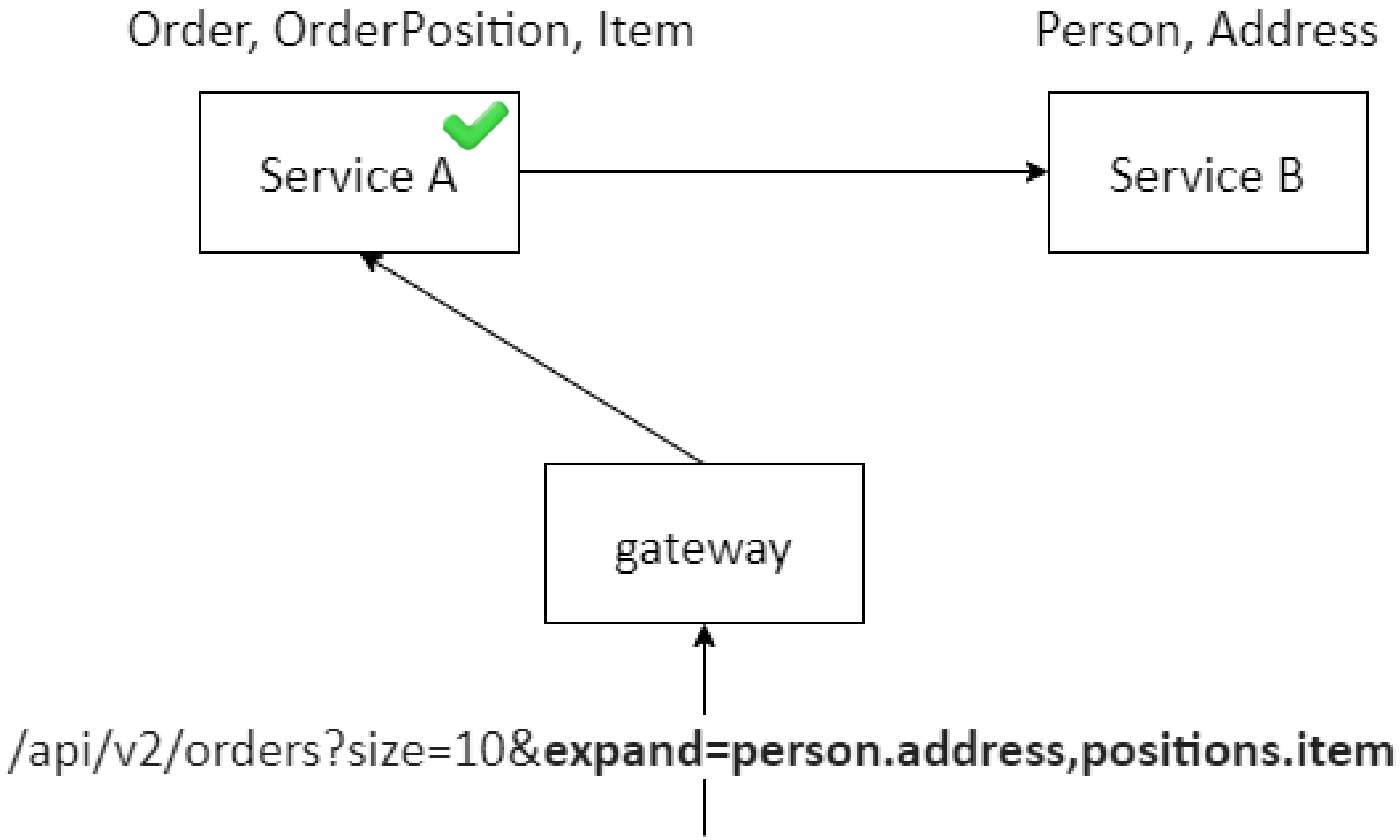


› Что делать? Кешировать!

НЛМК

```
@Component
@RequiredArgsConstructor
public class PersonDtoConverter extends AbstractExpandConverter<String, PersonDto> {
    ...
    /*
     * делаем в методе вызов во внешний сервис
    */
    @Override
    @Cacheable(value = PERSON_CACHE, key = "#personId + '-' + #path.pathsHashCode")
    public Person Dto convertSource(String personId, Path path) {
        return personService.getPerson(personId, path.getPaths(false));
    }
    ...
}
```

› А еще быстрые?

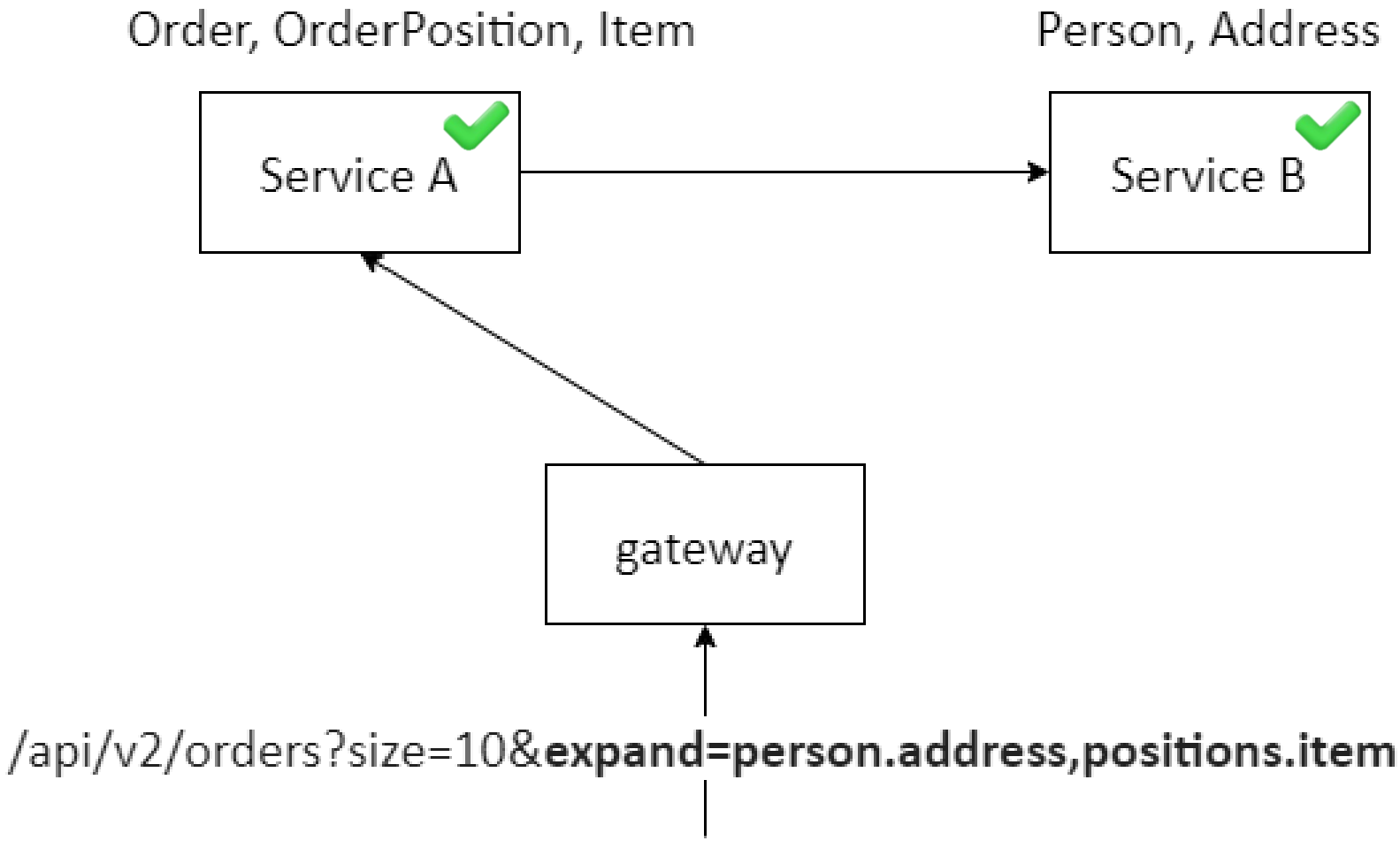


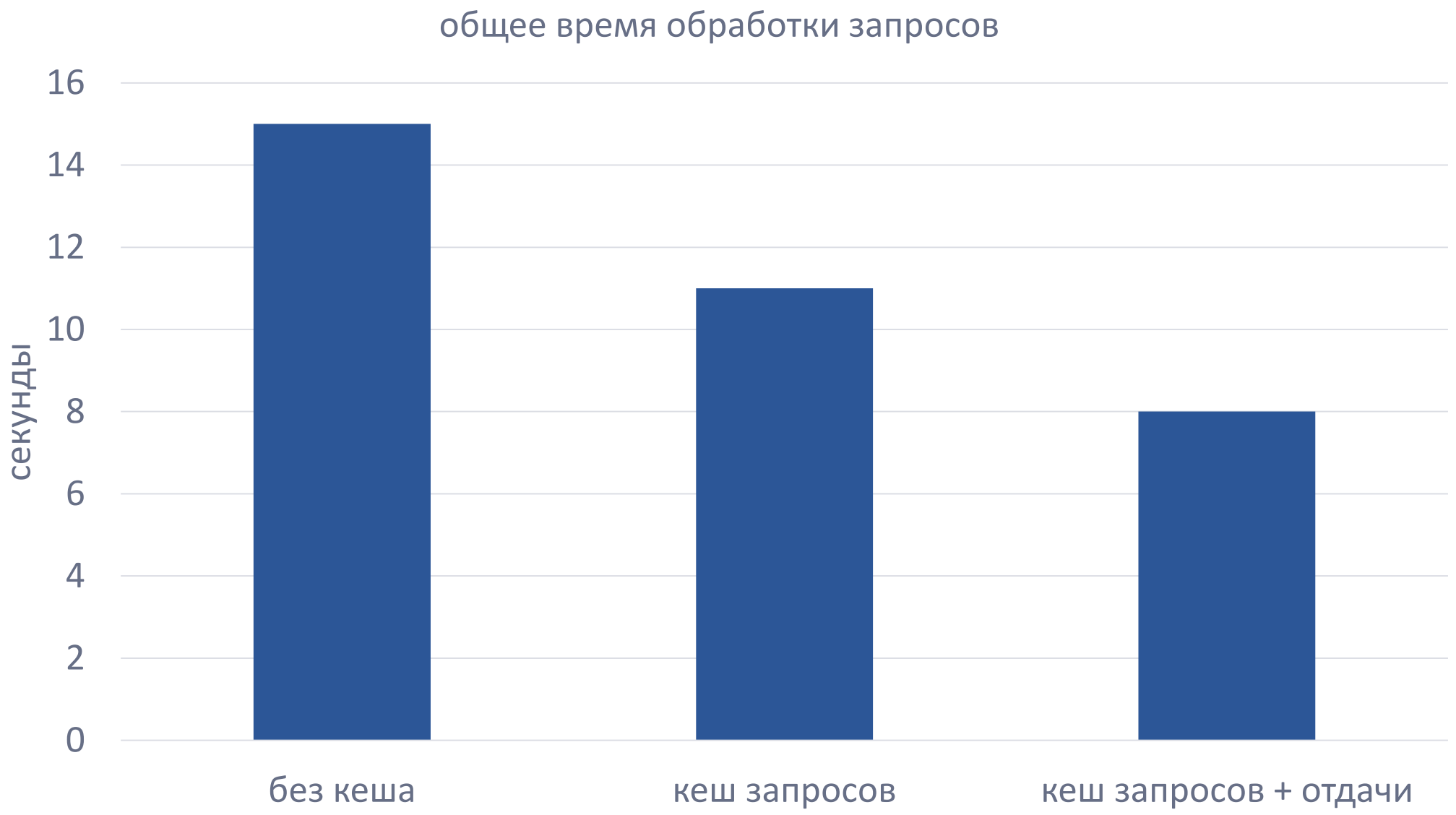
› Можно! Кешируем со стороны отдающего сервиса

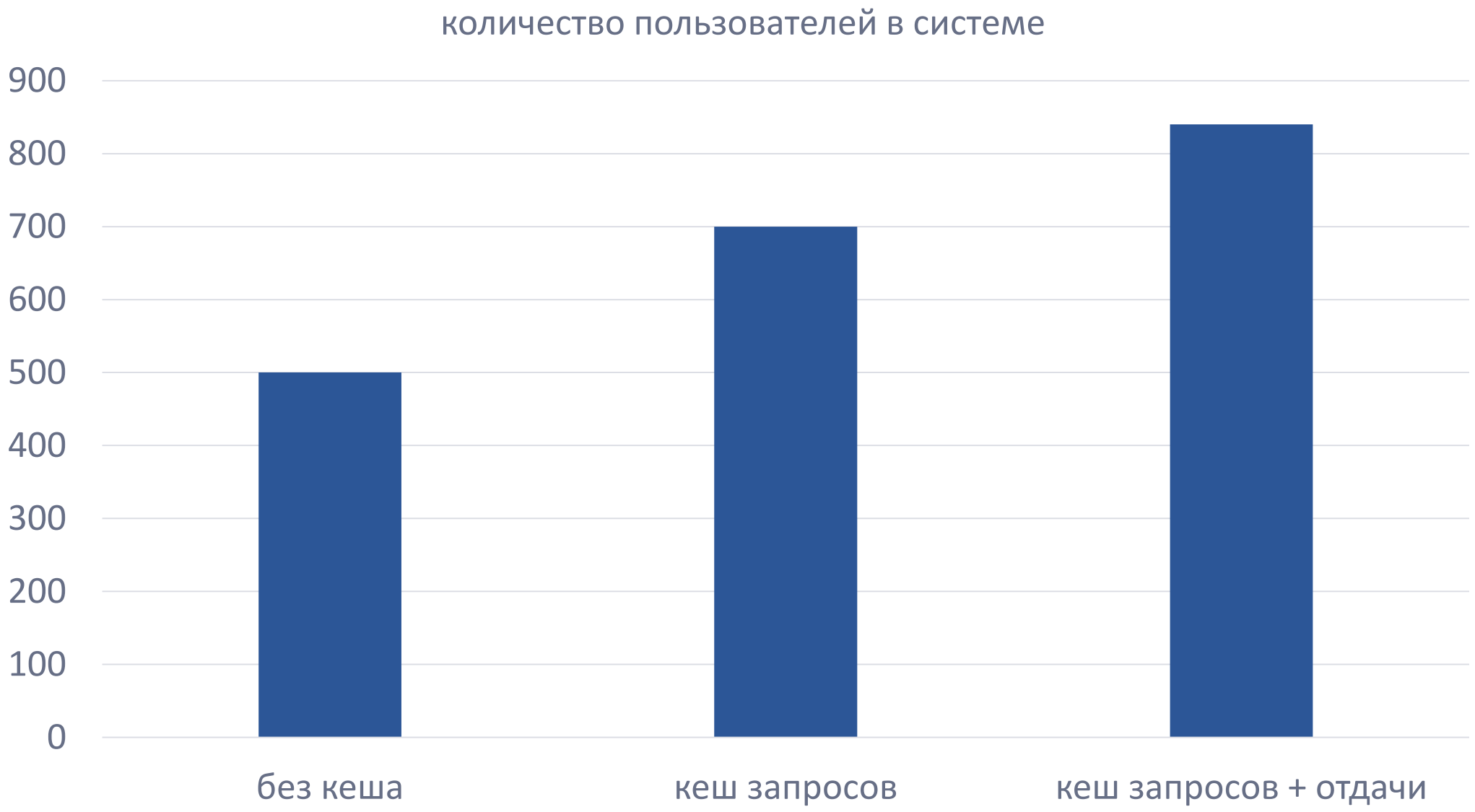
НЛМК

```
@Override
@Cacheable(value = PERSON_CACHE, key = "#personId + '-' + #path.pathsHashCode")
public PersonDto getPerson(String personId, Set<String> expands) {
   Person person = (Objects.isNull(expands) || expands.isEmpty()  ? personRepository.findById(personId) :
         personRepository.findById(personId, DynamicEntityGraph.loading(List.copyOf(expands))))
         .orElseThrow(() -> new EntityNotFoundException("getUser", notFoundUserMessage(personId)));
   return personConverter.convert(person, expands);
}
```

› Закешировали




общее время обработки запросов
16
14
12
10
8
6
4
2
0
секунды
без кеша
кеш запросов
кеш запросов + отдачи


количество пользователей в системе
900
800
700
600
500
400
300
200
100
0
без кеша
кеш запросов
кеш запросов + отдачи

expand=address, cache key=39--1147692044

```
"person": {
        "id": 39,
        "firstName":"person39",
        "lastName": "person39",
        "address": {
                "id": 39,
                "location":"location39"
        }
}
```

```
"person": {
        "id": 39,
        "firstName":"person39",
        "lastName": "person39",
        "address": {
                "id": 39,
                "location":"test"
        }
}
```

НЛМК

Time to live (TTL)

Инвалидация кеша по связям

НЛМК

Spring Boot 3.2.3

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-graphql</artifactId>
</dependency>
```

```
@GraphQlRepository
public interface AddressRepository extends CrudRepository<Address, Long>,
      QuerydslPredicateExecutor<Address> {
}
```

GraphQL endpoint HTTP POST /graphql

› Потестируем

Запрос **{ orders { id } }**

Hibernate: select o1_0.id,o1_0.delivery_type_id,o1_0.person_id from orders o1_0

Запрос **{ orders { id } }**
Hibernate: select o1_0.id,o1_0.delivery_type_id,o1_0.person_id from orders o1_0

Запрос **{ orders { id person { id firstName lastName } } }**
Hibernate: select
o1_0.id,o1_0.delivery_type_id,p1_0.id,p1_0.address_id,p1_0.first_name,p1_0.last_name from
orders o1_0 **left join** persons p1_0 on p1_0.id=o1_0.person_id

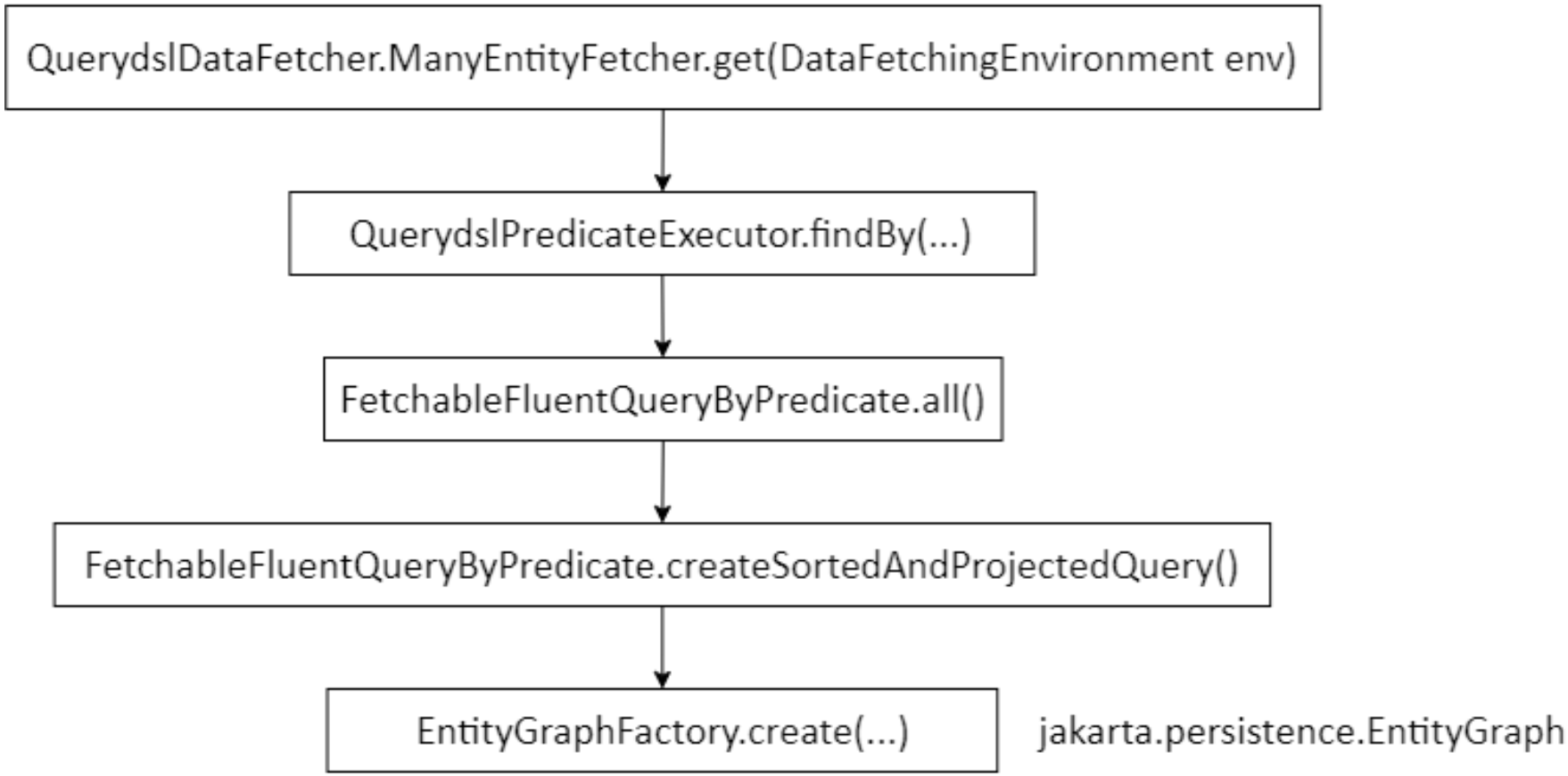
Запрос **{ orders { id } }**
Hibernate: select o1_0.id,o1_0.delivery_type_id,o1_0.person_id from orders o1_0

Запрос **{ orders { id person { id firstName lastName } } }**
Hibernate: select
o1_0.id,o1_0.delivery_type_id,p1_0.id,p1_0.address_id,p1_0.first_name,p1_0.last_name from
orders o1_0 **left join** persons p1_0 on p1_0.id=o1_0.person_id

Запрос **{ orders { id person { id firstName lastName address { id location } } } }**
Hibernate: select
o1_0.id,o1_0.delivery_type_id,p1_0.id,a1_0.id,a1_0.location,p1_0.first_name,p1_0.last_name
from orders o1_0 **left join** persons p1_0 on p1_0.id=o1_0.person_id **left join** addresses a1_0 on
a1_0.id=p1_0.address_id

› А давайте посмотрим что внутри?

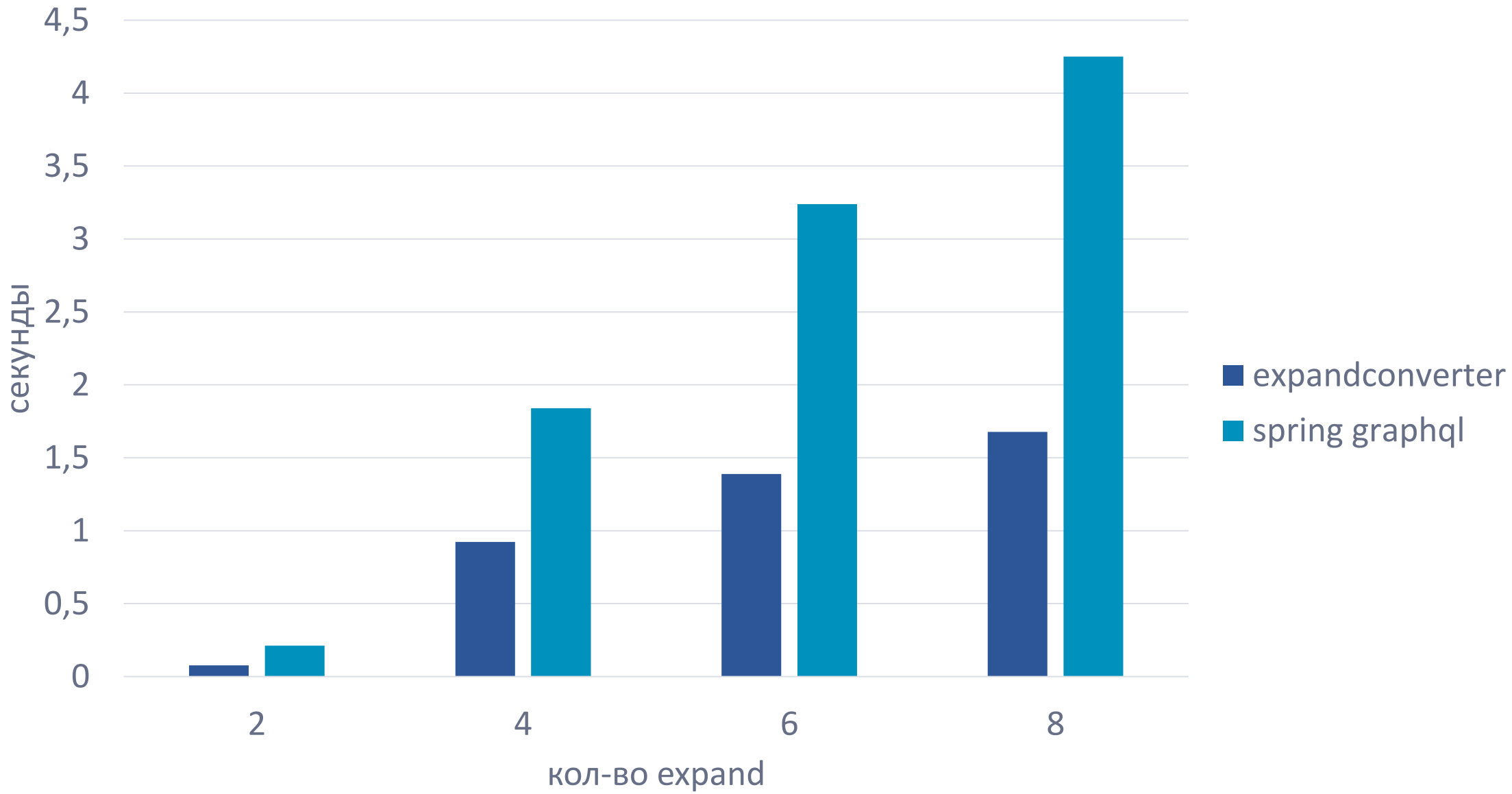
НЛМК


QuerydslDataFetcher.ManyEntityFetcher.get(DataFetchingEnvironment env)
QuerydslPredicateExecutor.findBy(...)
FetchableFluentQueryByPredicate.all()
FetchableFluentQueryByPredicate.createSortedAndProjectedQuery()
EntityGraphFactory.create(...)
jakarta.persistence.EntityGraph

› А давайте сравним!

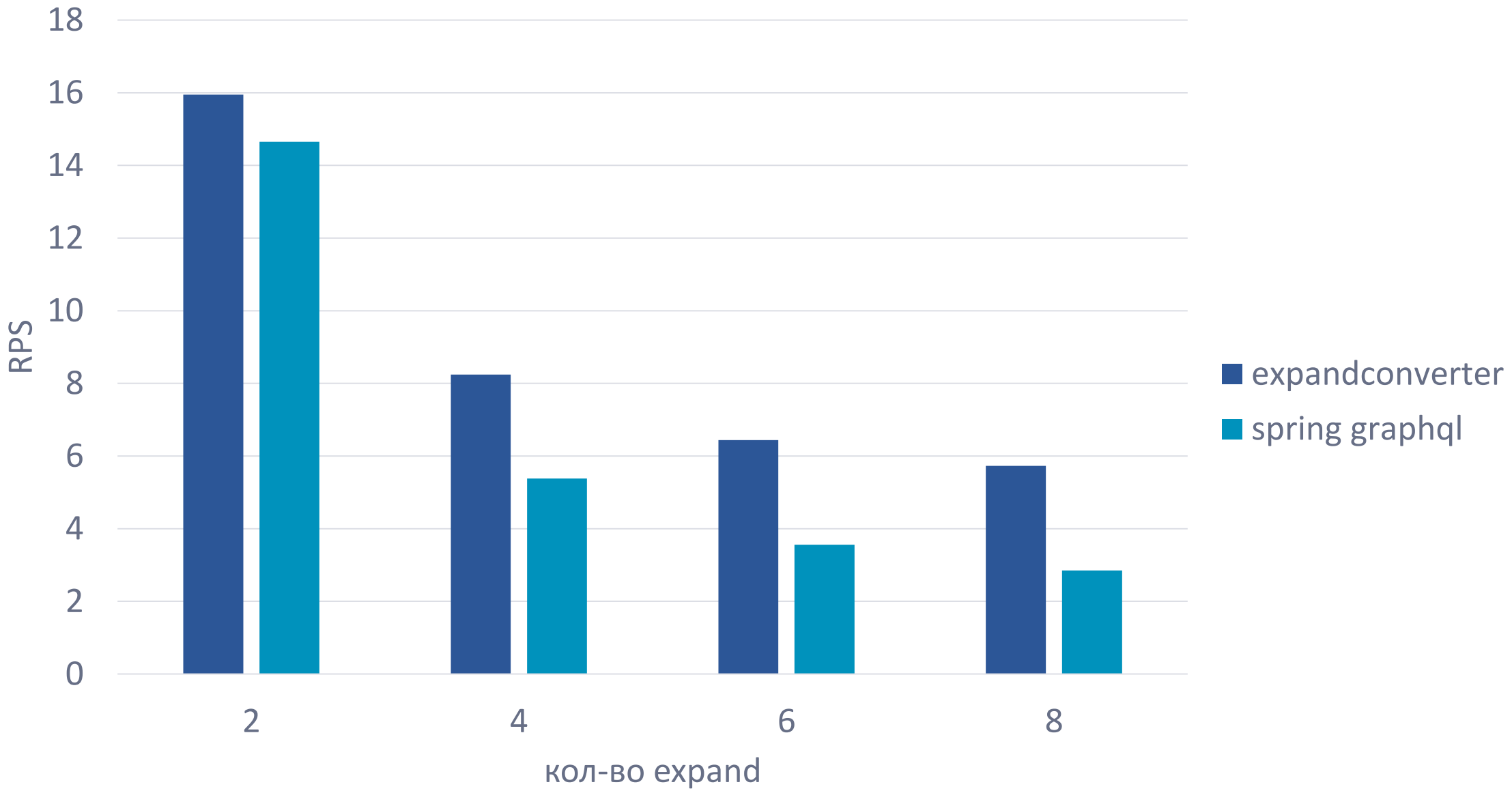
› Результаты

20 пользователей, 5 минут запуск



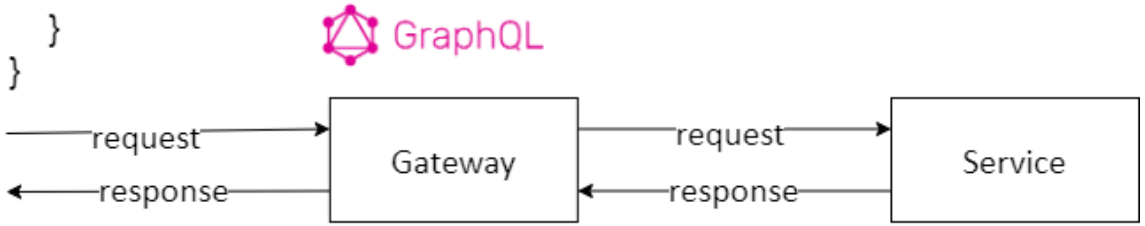
› Результаты

20 пользователей, 5 минут запуск



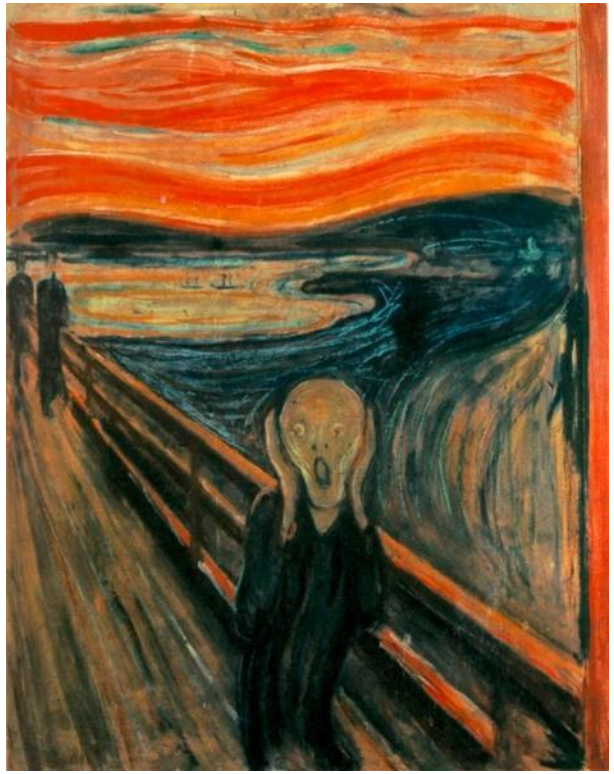
› Плохой пример

```
query {
   persons {
      id
   }
}
```




```
{
   "data": {
      "persons": [{
         "id": 39
      },...]
   }
}
```

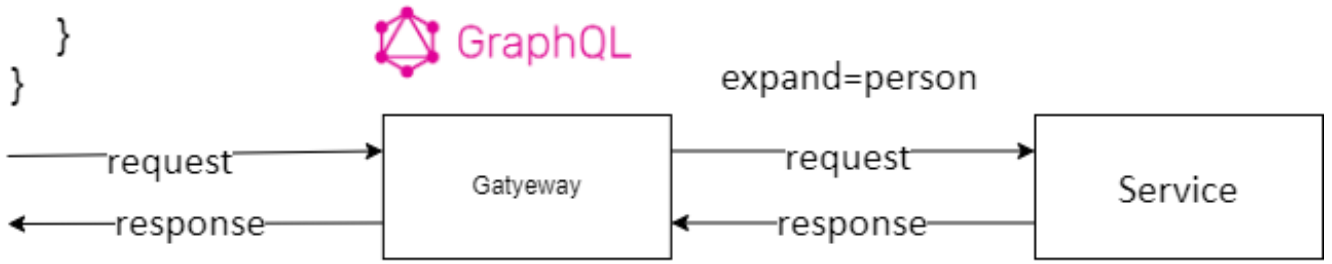
```
{
   "content": [{
      "id": 1,
      "person": {
         "id": 39,
         "firstName":"person39",
         "lastName": "person39",
         "address": {
            "id": 39,
            "location": "address39"
         }
      },
      "deliveryType": {
         "id": 1,
         "type":"type0"
      },
      ...
   }, ...],
   ...
}
```



› Ребята, давайте жить дружно

› Совмещаем

```
query {
   persons {
      id
   }
}
```




```
{
   "data": {
      "persons": [{
         "id": 39
      },...]
   }
}
```

```
{
   "content": [{
      "id": 1,
      "person": {
         "id": 39,
         "firstName":"person39",
         "lastName": "person39",
         "address": null
      },
      "deliveryType": null,
      ...
   }, ...],
   ...
}
```

› Минусы

1. Пока работаем с сущностями 1 к 1 в API из базы
2. Нет возможности запрашивать по полям
3. Много дубляжа в строках параметров – order.positions, order.deliveryType
4. Запросы в базу обрабатывает ORM

› Плюсы

НЛМК

1. Разделения на слои – запрос данных, конвертация, API. Каждый слой можно настраивать независимо друг от друга
2. Один протокол на все – фронт, межсервисное взаимодействие
3. Хорошо подходит для операций чтения со множеством вложенных сущностей
4. Все что предоставляется Spring Data JPA доступно – пагинация, Criteria API и т. д.
5. Не далеко ушли от Spring