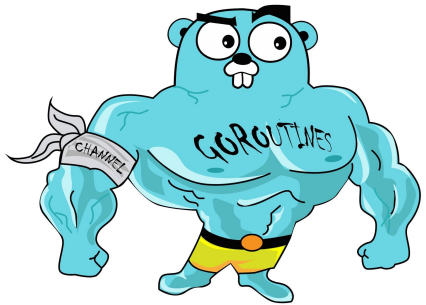
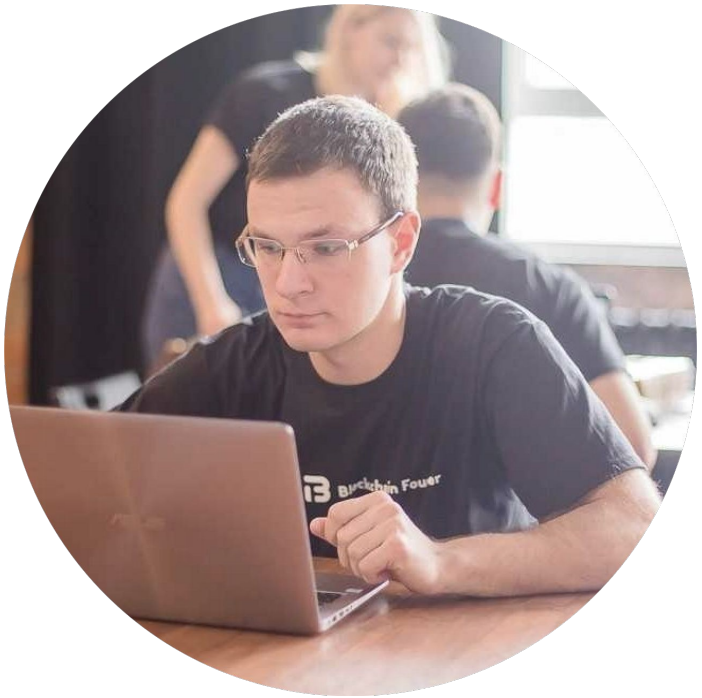
Чем отличается Saga от Workflow



Обо мне



- 🚀 Старший разработчик;
- 💡 Опыт backend более 5 лет;
- 👨‍💼 Провожу собеседования в секции go/system design;
- 📣 Спикер на CodeFest13;
- 👨‍🎓 Окончил МГТУ им Н.Э. Баумана.

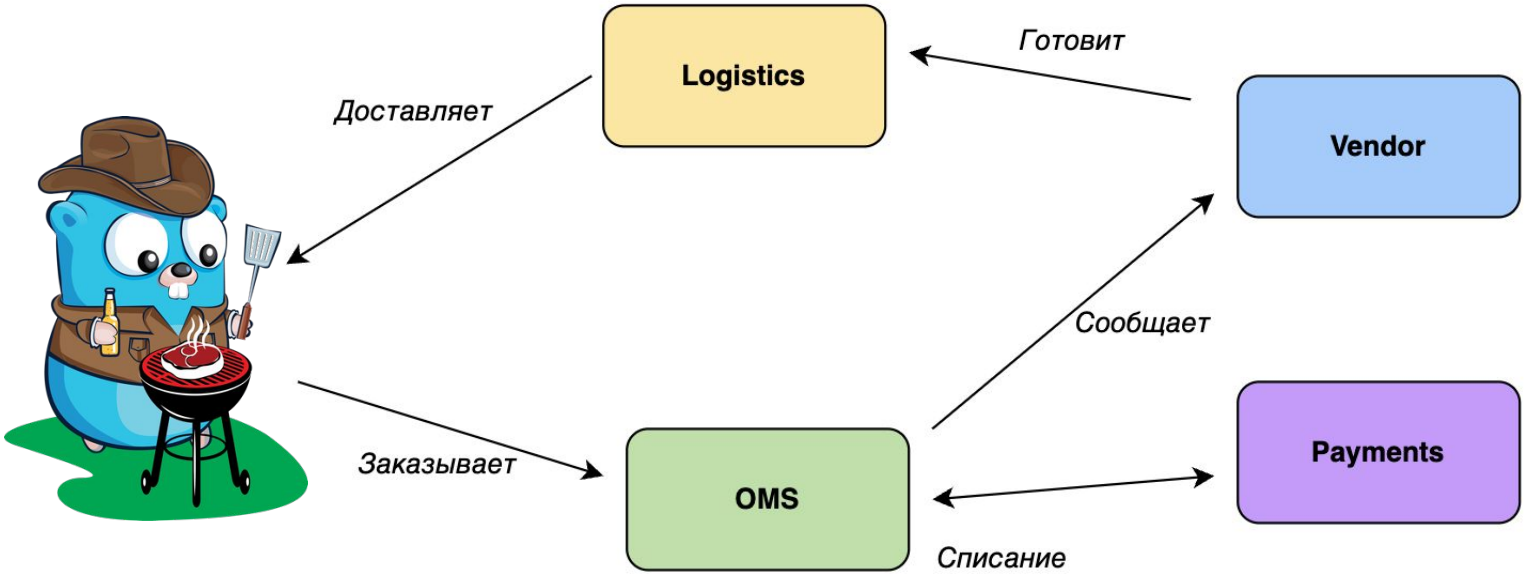
https://t.me/backend_architecture

План

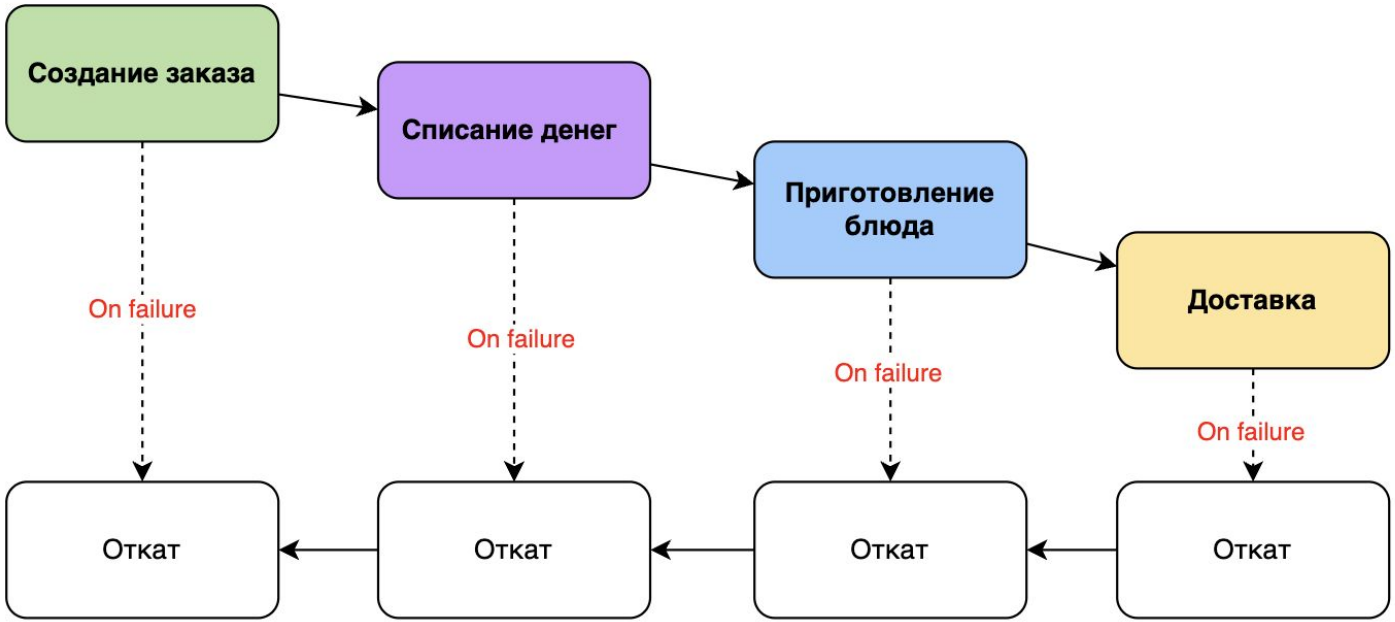
- Введение
- Проблематика
- Типовые решения
  - 2PC
  - Saga - оркестрация
  - Saga - хореография
  - Workflow
- Отличие saga от workflow
- Готовые решения для реализации

Введение

Мы строим Foodtech-сервис, основной сценарий это заказ еды пользователем.

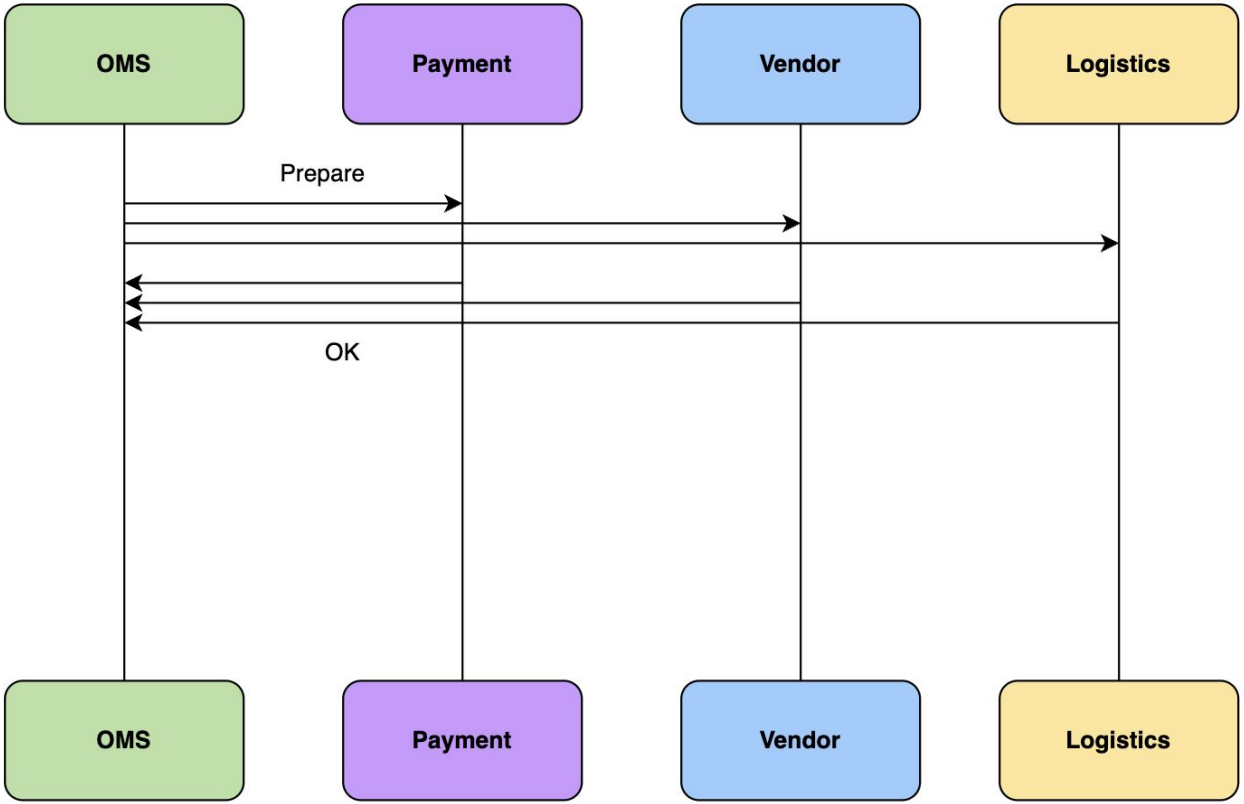


Проблематика

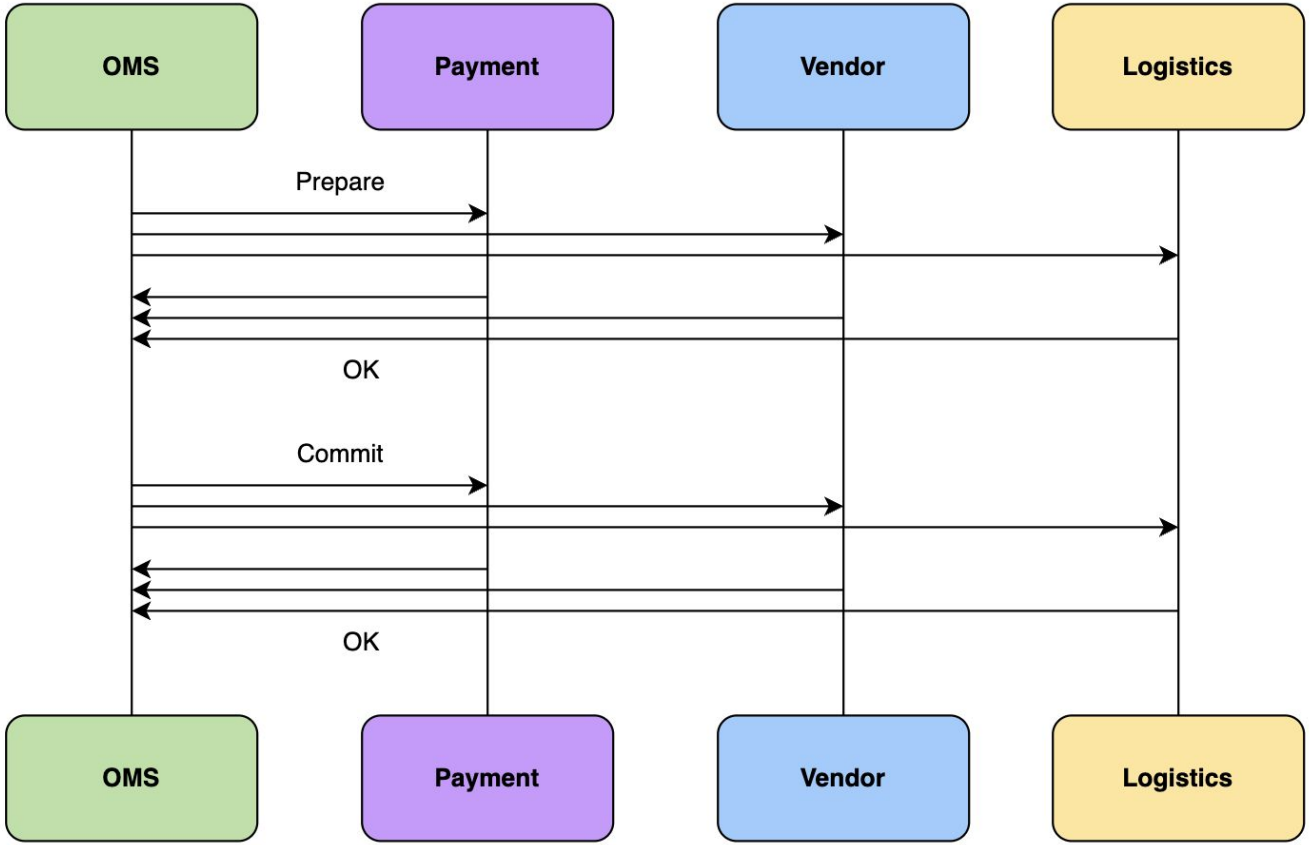


2PC

2PC


OMS
Payment
Vendor
Logistics
Prepare
OK
OMS
Payment
Vendor
Logistics

2PC


OMS
Payment
Vendor
Logistics
Prepare
OK
Commit
OK
OMS
Payment
Vendor
Logistics

2PC

❌ Нам этот вариант не подходит, так как не нужно сразу фиксировать какое-то состояние в нескольких БД

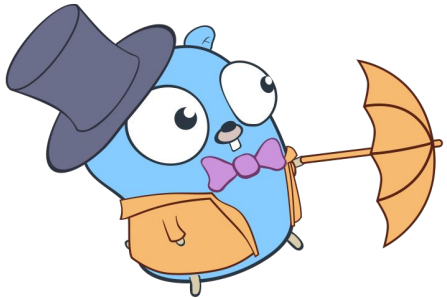
🤓 К примеру, мы хотим отправлять заказ в ресторан, только после списания средств с карты.

Saga

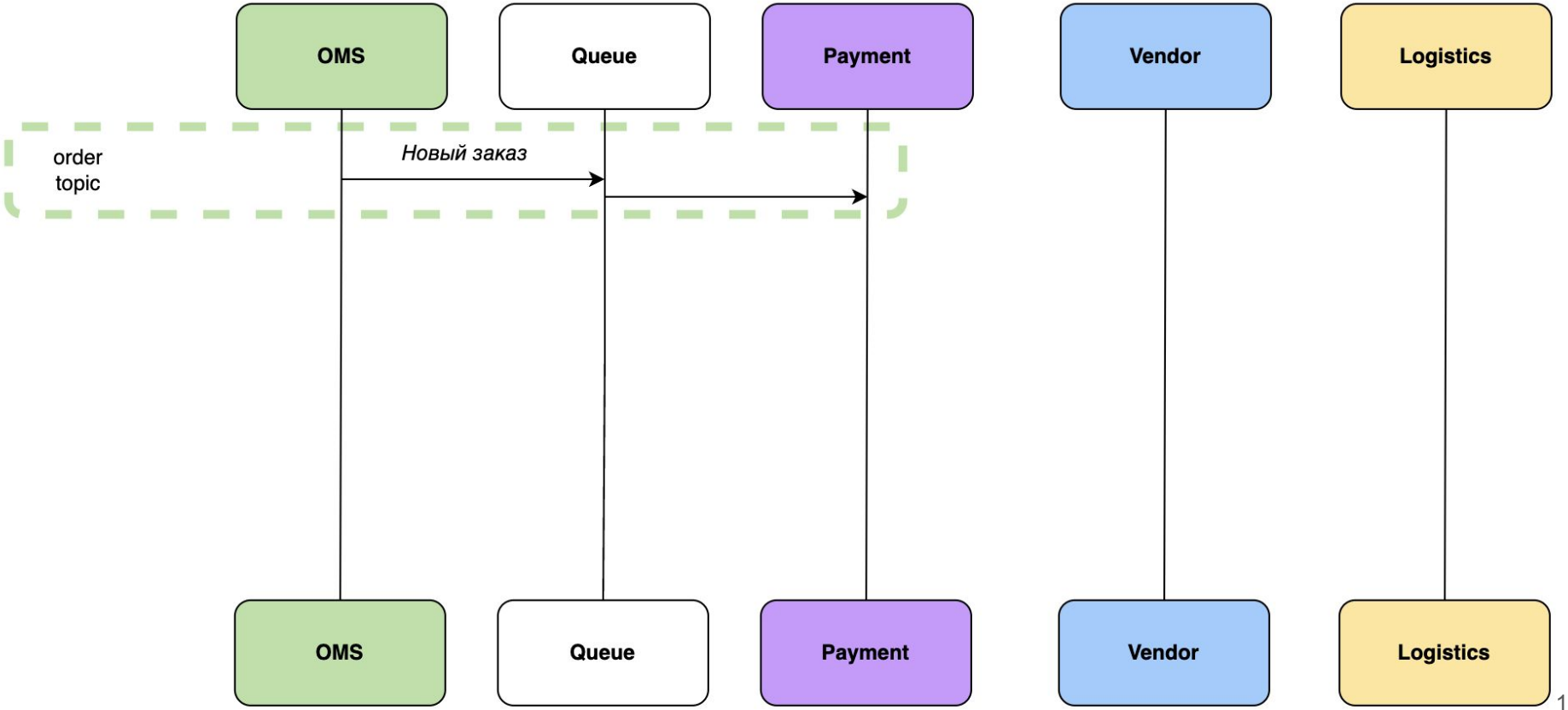
Хореография

Saga - хореография

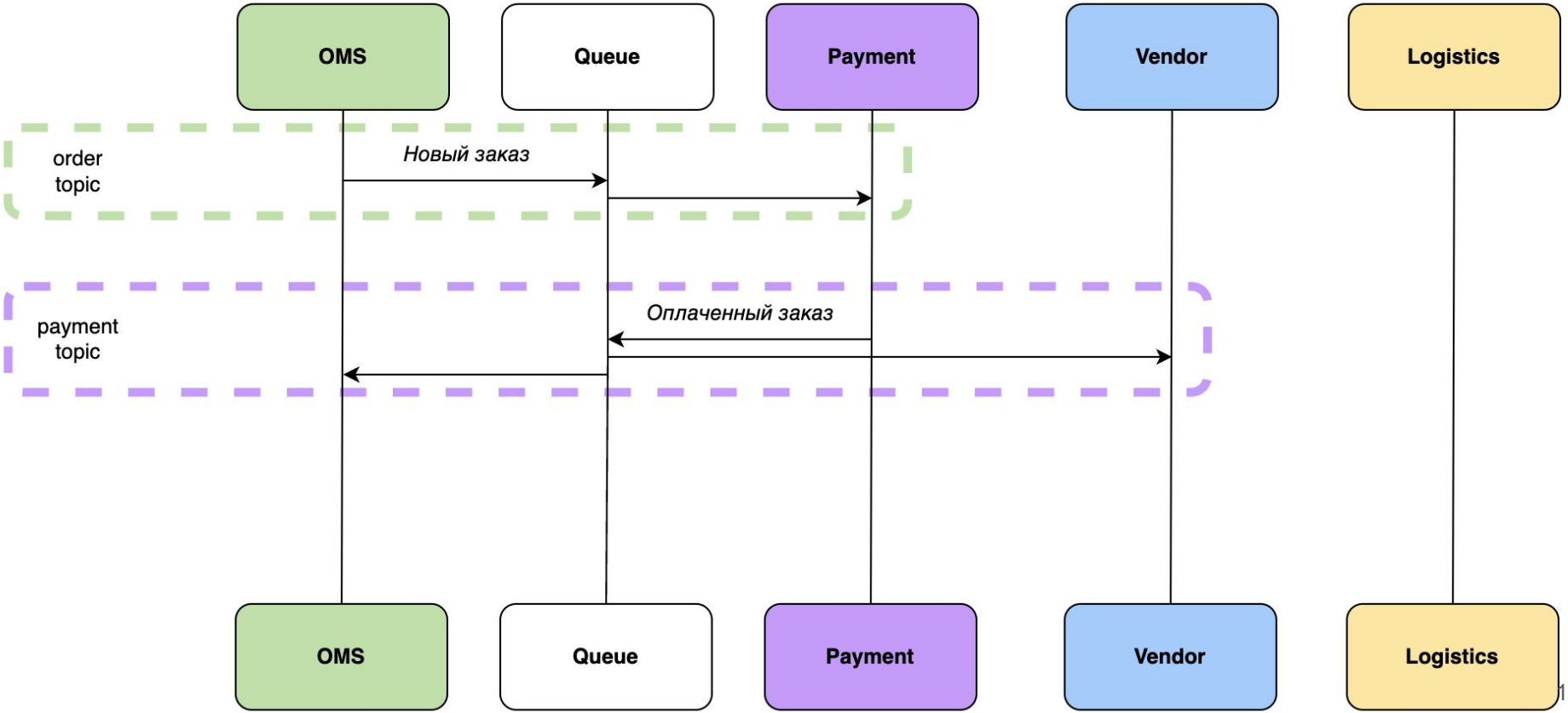
📌 каждая транзакция публикует события, которые запускают транзакции в других сервисах



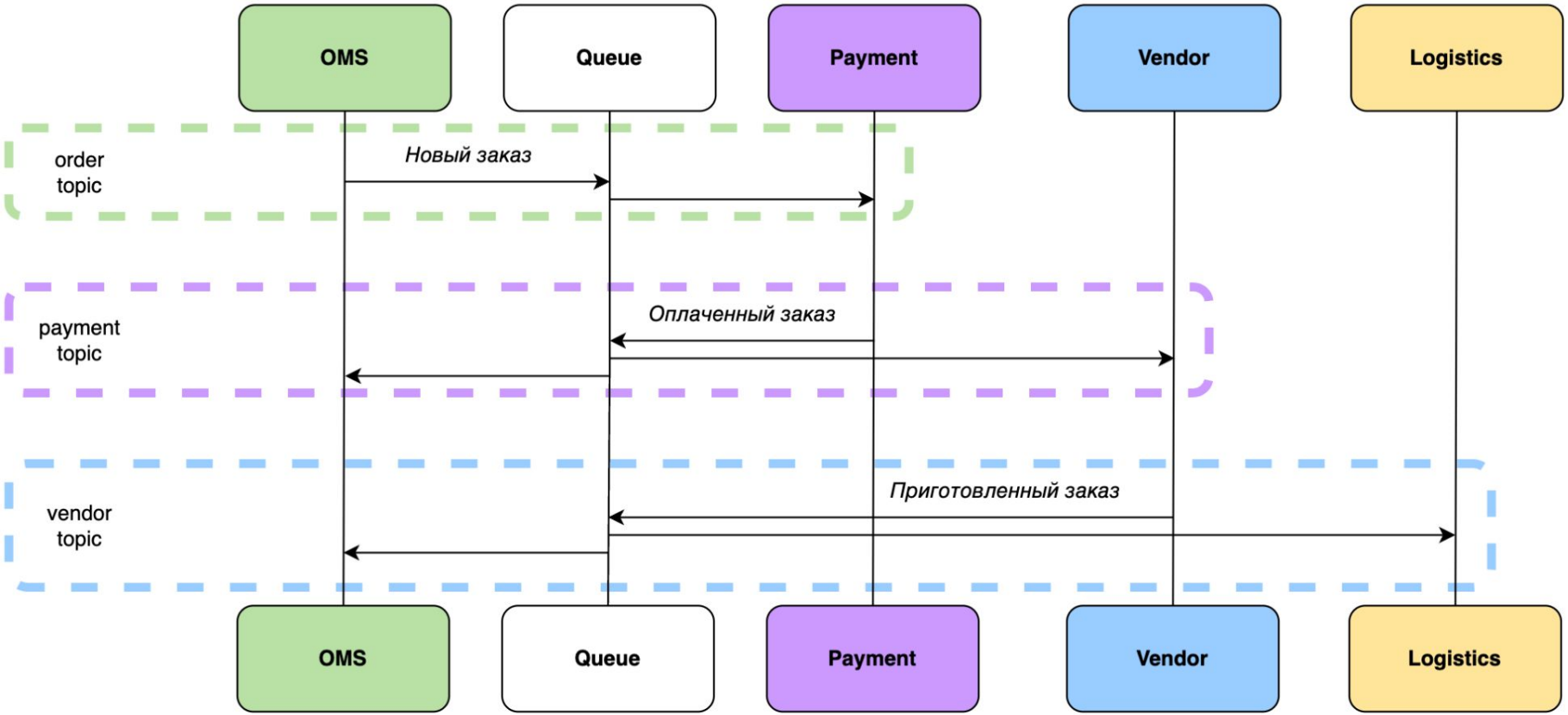
Saga - хореография



Saga - хореография



Saga - хореография



Saga - хореография

❌ Вариант хороший, но при увеличении количества участников транзакции взаимосвязей между компонентами => компоненты умные, содержат информацию о том, кто за кем следует.

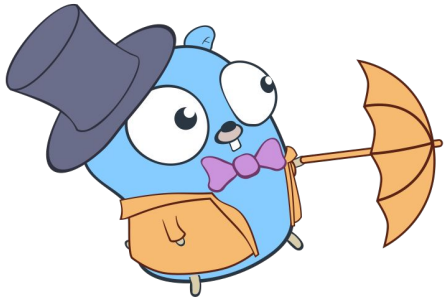
🤓 К примеру, мы хотим усложнить процесс, добавить еще шагов.

Saga

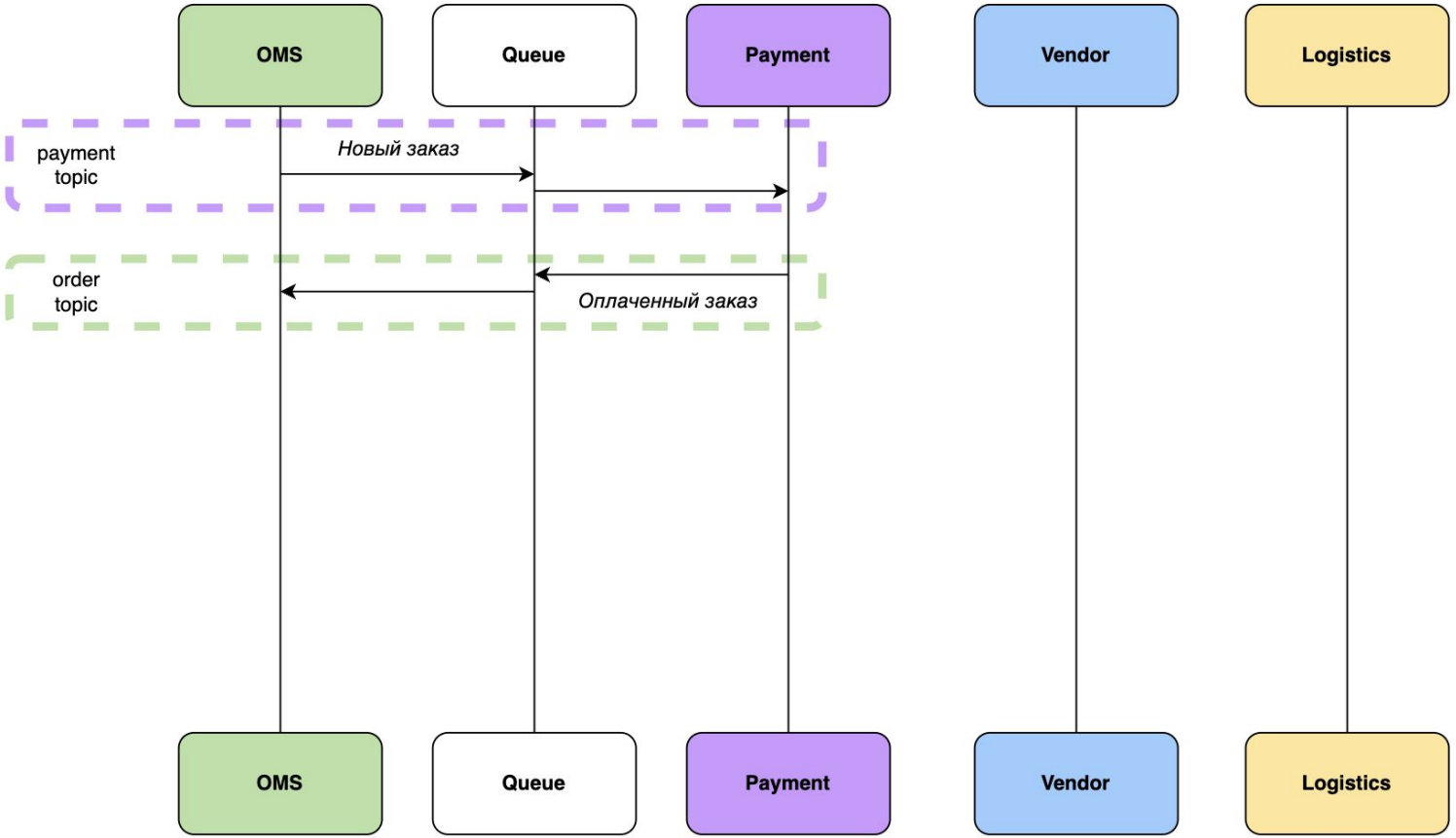
Оркестрация

Saga - оркестрация

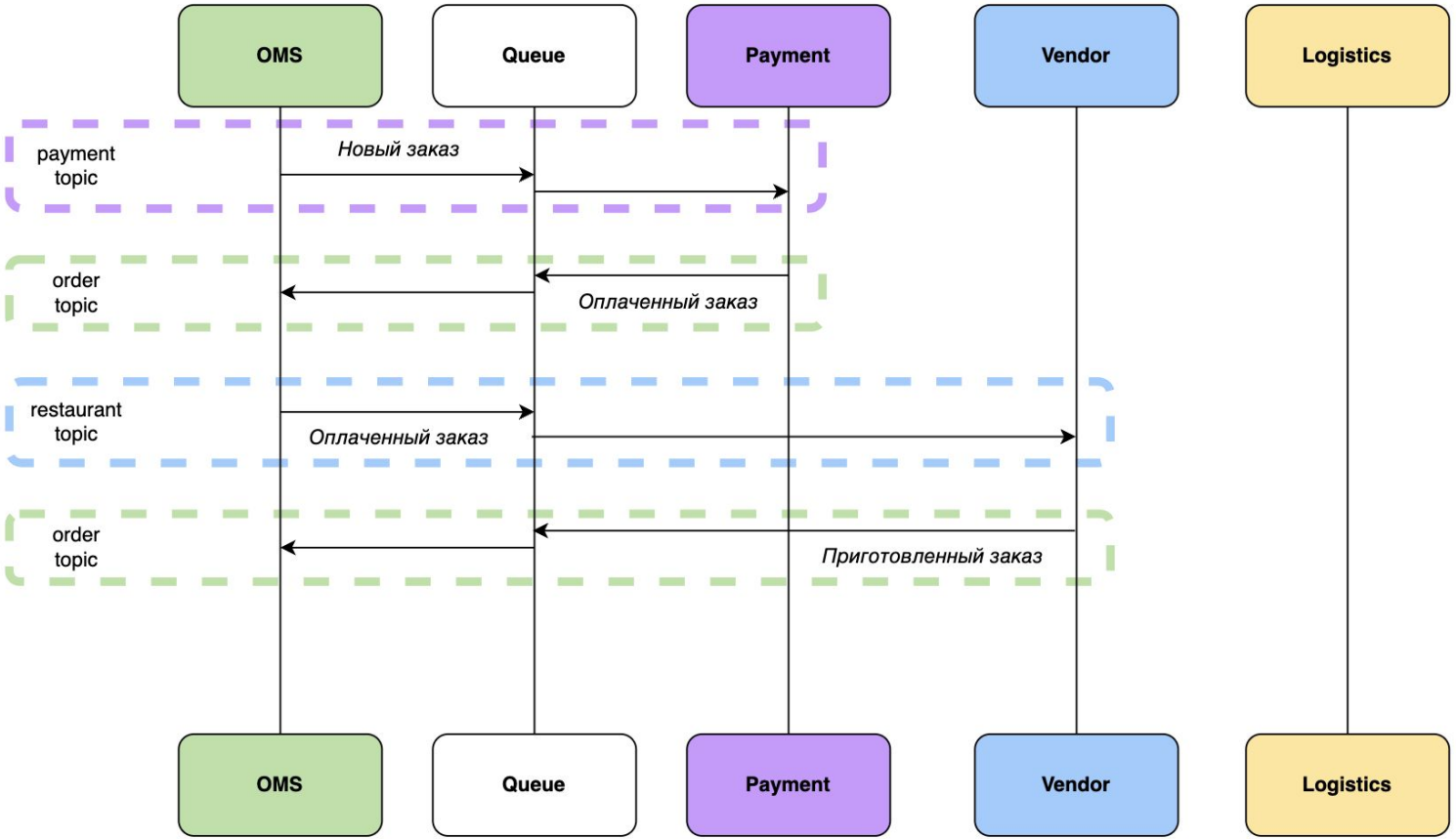
📌 оркестратор говорит участникам, какие транзакции должны быть запущены



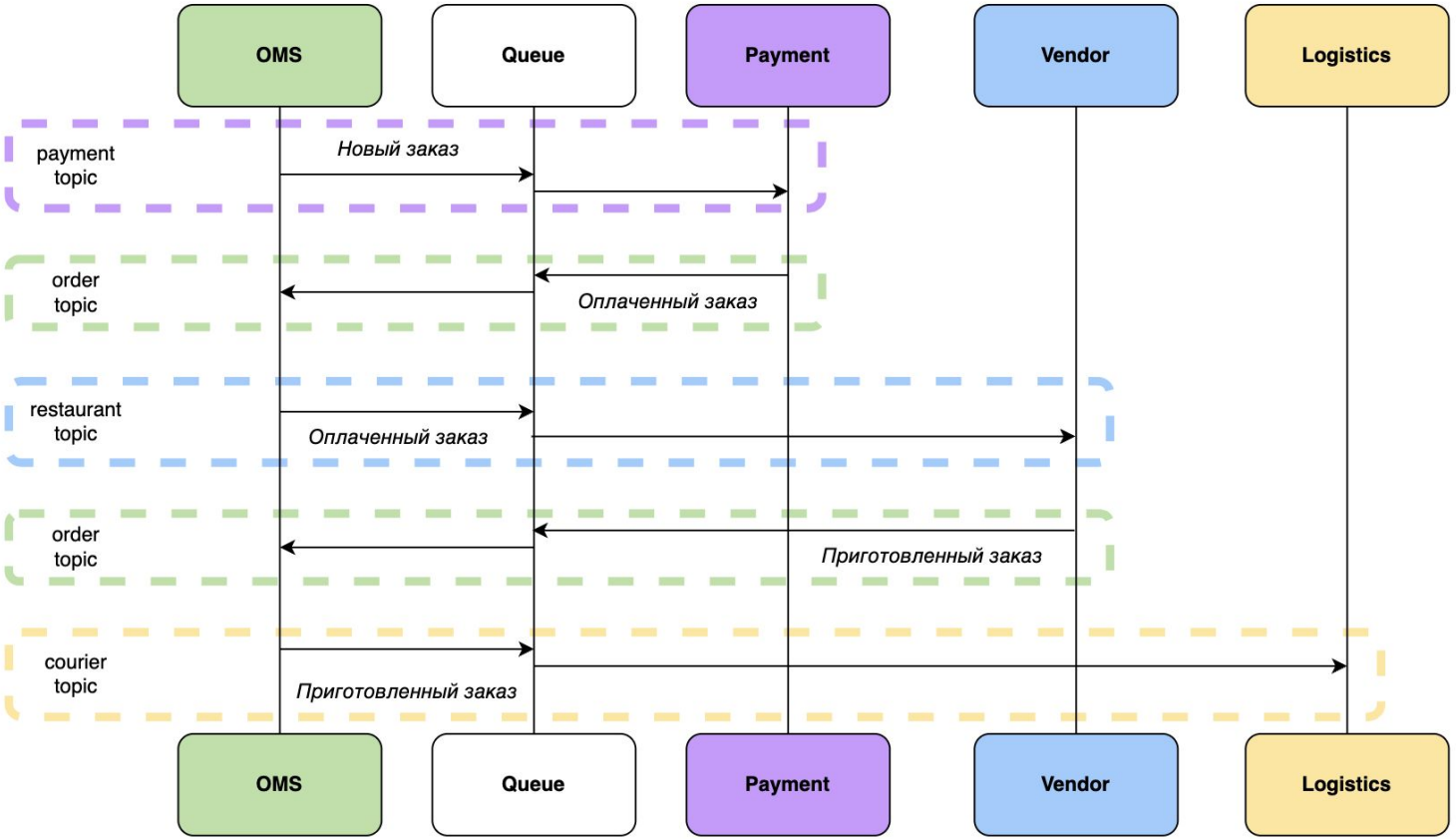
Saga - оркестрация



Saga - оркестрация



Saga - оркестрация



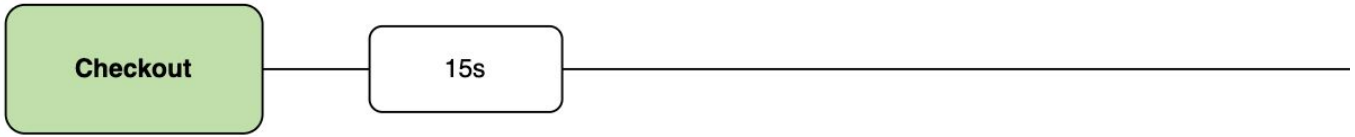
Saga - оркестрация

❌ Движением заказа между участниками транзакции управляет оркестратор - OMS

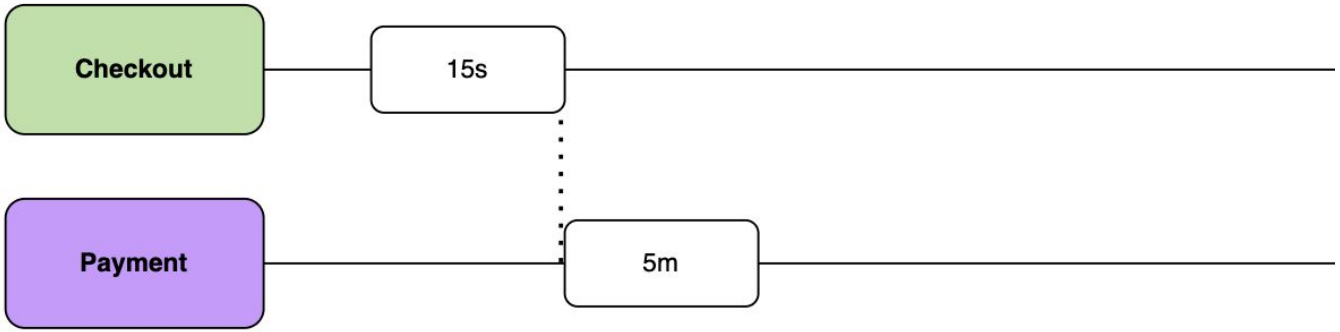
🤓 Появляется единая точка отказа

Workflow

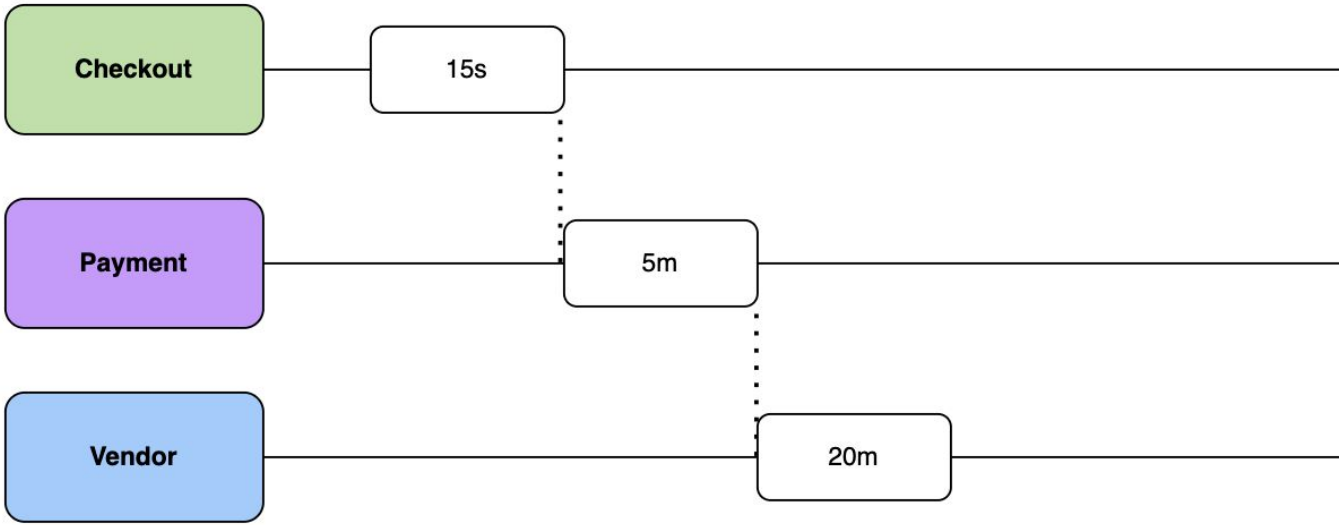
Workflow



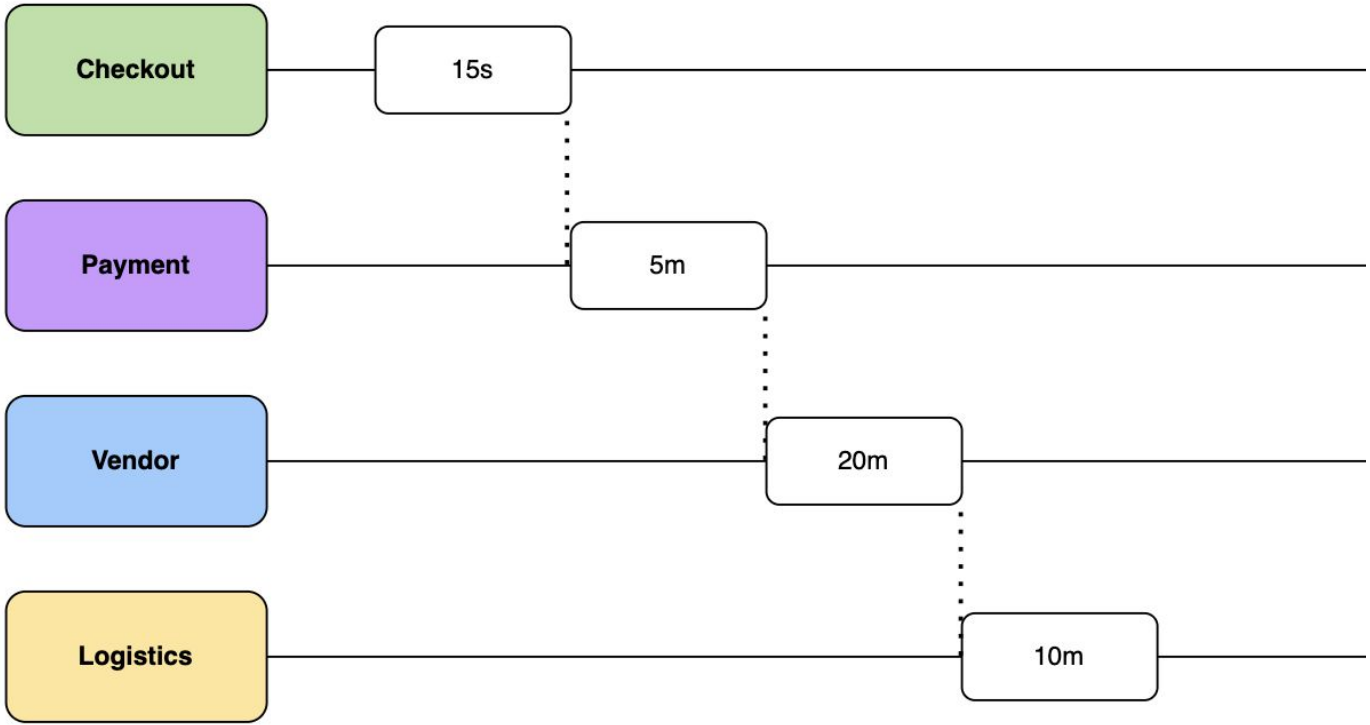
Workflow



Workflow



Workflow



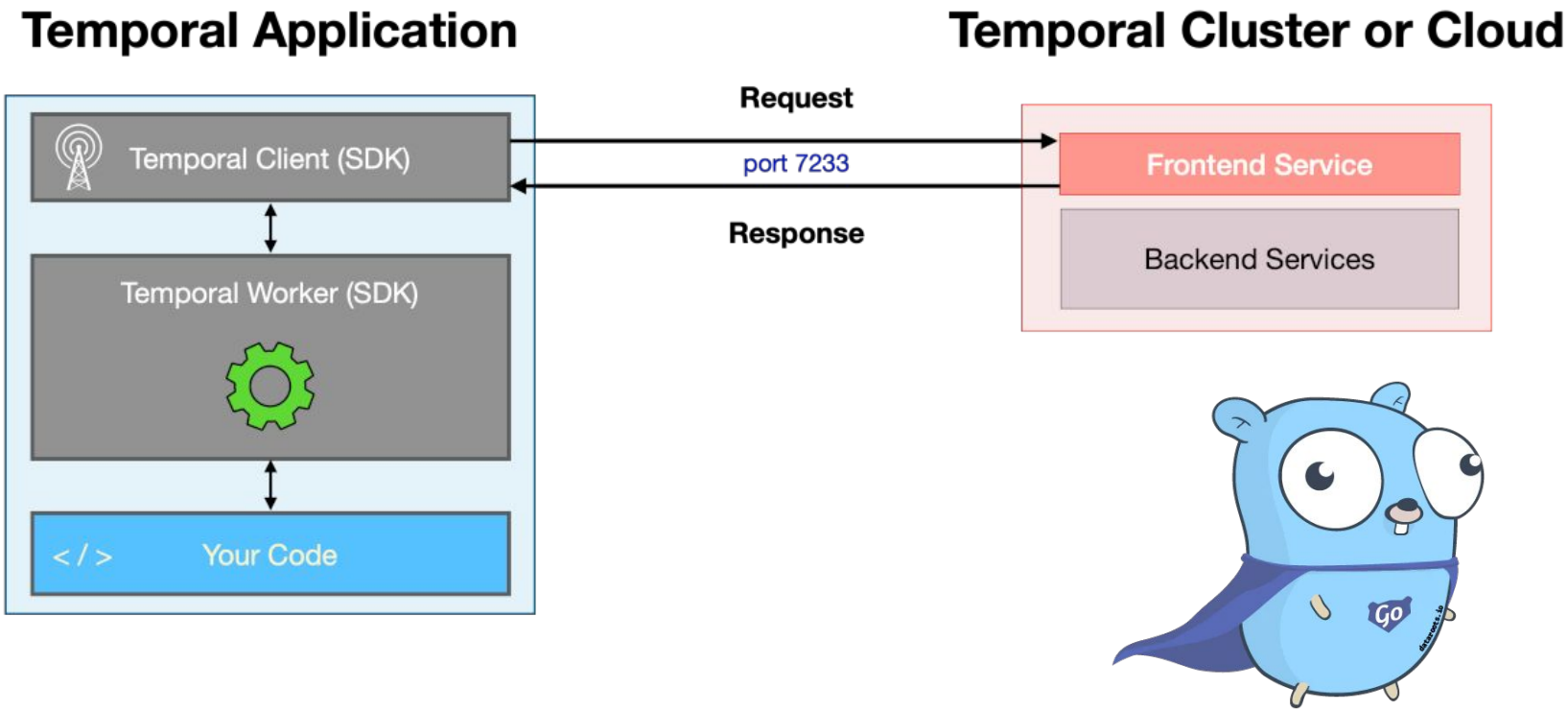
Отличие saga от workflow

Saga - частный случай workflow. На практике мы хотим отказоустойчивый оркестратор, который поддерживает политики таймеров, ретраев, взаимодействий между дочерними workflow.

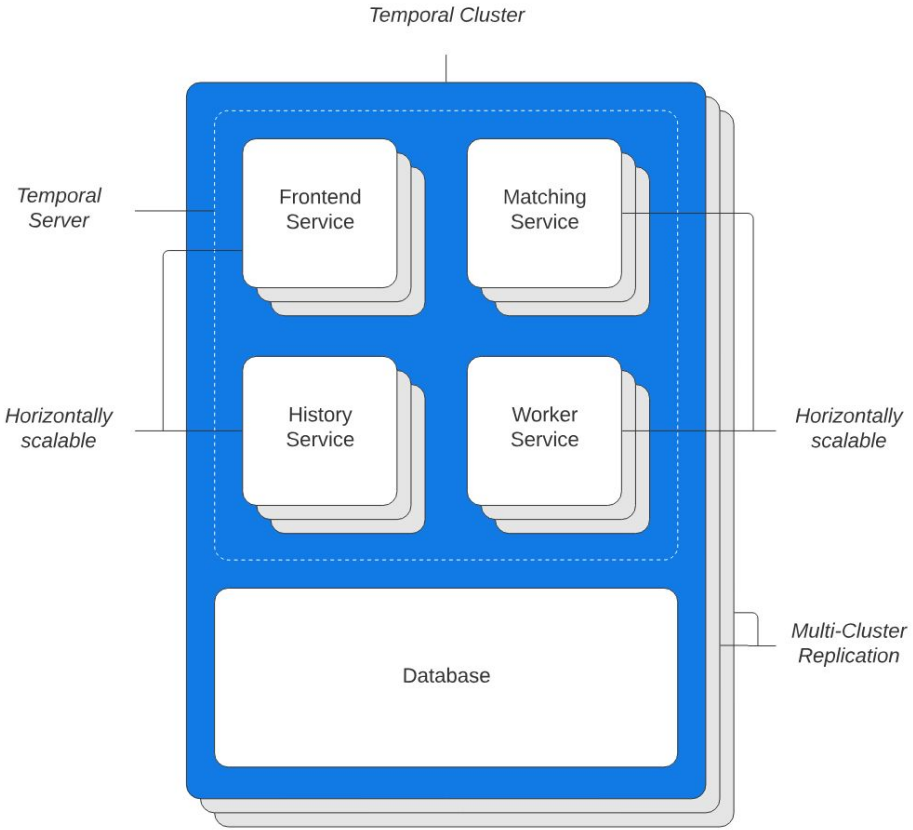
Практика

Temporal

Temporal


Temporal Application
Temporal Cluster or Cloud
Request
port 7233
Response
Temporal Client (SDK)
Temporal Worker (SDK)
< / >
Your Code
Frontend Service
Backend Services
Go

Temporal


Temporal Cluster
Temporal Server
Frontend Service
Matching Service
History Service
Worker Service
Horizontally scalable
Horizontally scalable
Database
Multi-Cluster Replication

Payments

```go
func CreatePayment(ctx context.Context, order *models.Order) (*uc.RegisterPaymentDTO, error) {
    logger := activity.GetLogger(ctx)

    registeredPayment, err := RegisterPayment(ctx, order)
    if err != nil {
        logger.Error( msg: "Failed create payment",  keyvals...: "Error", err)
        return registeredPayment, errors.WithStack(err)
    }

    return registeredPayment, nil
}
```

🤓 Идемпотентность

Activity - Options

```
11  var CreatePaymentOptions = workflow.ActivityOptions{
12      StartToCloseTimeout: time.Second * 20,
13      RetryPolicy: &temporal.RetryPolicy{
14          MaximumAttempts: 5,
15      },
16  }

18  func WithCreatePaymentOptions(ctx workflow.Context) workflow.Context {
19      return workflow.WithActivityOptions(ctx, CreatePaymentOptions)
20  }
```

Activity

```
30  func CreatePaymentExecute(ctx workflow.Context, order *models.Order) (*uc.RegisterPaymentDTO, error) {
31      var registeredPayment uc.RegisterPaymentDTO
32
33      if err := workflow.ExecuteActivity(
34          WithCreatePaymentOptions(ctx),
35          CreatePayment,
36          order,
37      ).Get(ctx, &registeredPayment); err != nil {
38          return nil, errors.WithStack(err)
39      }
40
41      return &registeredPayment, nil
42  }
```

Workflow

```
27  func Checkout(ctx workflow.Context, checkout *models.OrderCheckout) (*models.Order, error) {
28      var (
29          order             models.Order
30          registeredPayment *uc.RegisterPaymentDTO
31          err               error
32      )
33
34      logger := workflow.GetLogger(ctx)
35
36      if registeredPayment, err = activities.CreatePaymentExecute(ctx, &order); err != nil {
37          order.Status = models.OrderStatusCancel
38          logger.Error( msg: "Payment creation failed",  keyvals...: "Error", err)
39          return nil, ErrorPayment
40      }
41
42      // Заполняем информацию о платеже
43      order.Payment.ID = registeredPayment.PaymentID
44
45      // ....
46
47      return &order, nil
48  }
```

Workflow - Options

```
17  func CheckoutProcess(ctx context.Context, id string, dto *models.OrderCheckout) (*models.Order, error) {
18      options := client.StartWorkflowOptions{
19          ID:                  id,
20          TaskQueue:           checkout.Queue,
21          WorkflowRunTimeout: time.Minute * 10,
22      }
23      w, err := tcl.ExecuteWorkflow(ctx, options, checkout.Checkout, dto)
24      if err != nil {
25          return nil, err
26      }
27      var order models.Order
28      if err = w.Get(ctx, &order); err != nil {
29          return nil, err
30      }
31
32      return &order, nil
33  }
```

Quick Start

```
176  func ConfigureCheckout() error {  1 usage
177      w := temporal.NewWorker(
178          worker.New(client, checkout.Queue, worker.Options{
179              MaxConcurrentActivityTaskPollers: 4,
180          }),
181      )
182
183      w.RegisterWorkflow(checkout.Checkout)
184      w.RegisterActivity(activities.CreatePayment)
185      if err := w.Start(); err != nil {
186          return err
187      }
188
189      return nil
190  }
```

Saga

Temporal - Saga

```
32      ctx = workflow.WithActivityOptions(ctx, options)
33
34      err = workflow.ExecuteActivity(ctx, Payment, transaction).Get(ctx, valuePtr: nil)
35      if err != nil {
36          return err
37      }
38
39      defer func() {
40          if err != nil {
41              errCompensation := workflow.ExecuteActivity(ctx, PaymentCompensation, transaction).
42                  Get(ctx, valuePtr: nil)
43              err = multierr.Append(err, errCompensation)
44          }
45      }()
```

Workflow

Temporal - Workflow

```
265  func startVendorOrderWorkflow(ctx workflow.Context, order *models.Order) error {  1 usage
266      options := workflow.ChildWorkflowOptions{
267          WorkflowID:        vendororder.WorkflowID(order.ID),
268          TaskQueue:         vendororder.Queue,
269          ParentClosePolicy: enums.PARENT_CLOSE_POLICY_REQUEST_CANCEL,
270      }
271      newCtx := workflow.WithChildOptions(ctx, options)
272      childWorkflowFuture := workflow.ExecuteChildWorkflow(
273          newCtx, vendororder.VendorOrder, vendororderadapter.TransformVendorOrder(order),
274      )
275
276      var childWE workflow.Execution // nolint
277      err := childWorkflowFuture.GetChildWorkflowExecution().Get(newCtx, &childWE)
278
279      return err
280  }
```

Заключение

Мышление с помощью workflow позволяет смотреть на проект на уровень выше, чем в случае с saga. Уровень абстракции выше => можем покрыть больше кейсов. Для оркестрации заказов OMS, мы используем Temporal.