управление данными в микросервисах на C#

реактивное межсервисное взаимодействие

проблемы явного межсервисного взаимодействия

явное межсервисное взаимодействие

проблемы

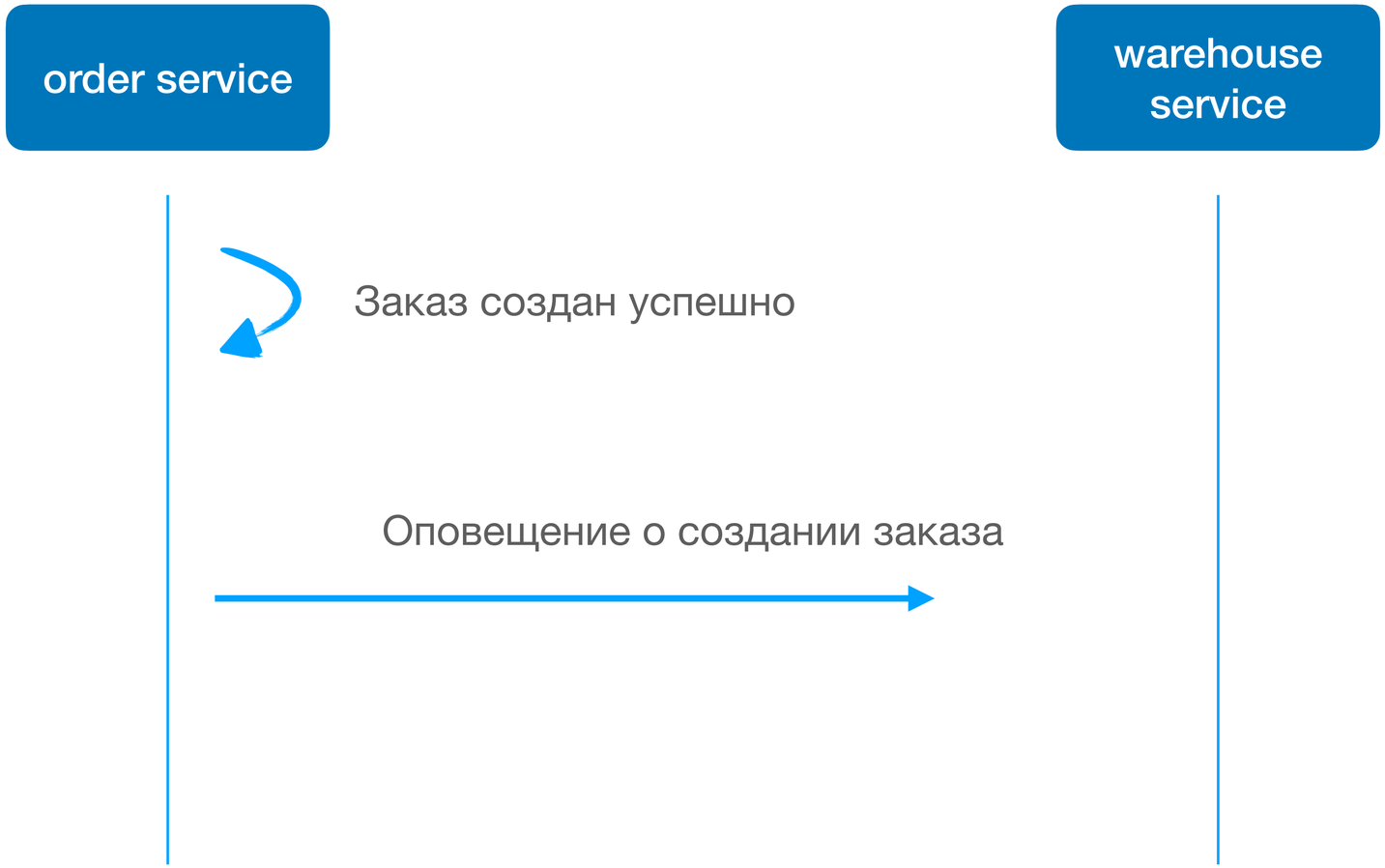
- сильная связанность модулей системы
- необходимость обработки ошибок на стороне сервиса-отправителя
- обработка transient ошибок на стороне сервиса-отправителя
- концентрация ответственности за интеграции

явное межсервисное взаимодействие

проблемы

явное межсервисное взаимодействие

проблемы

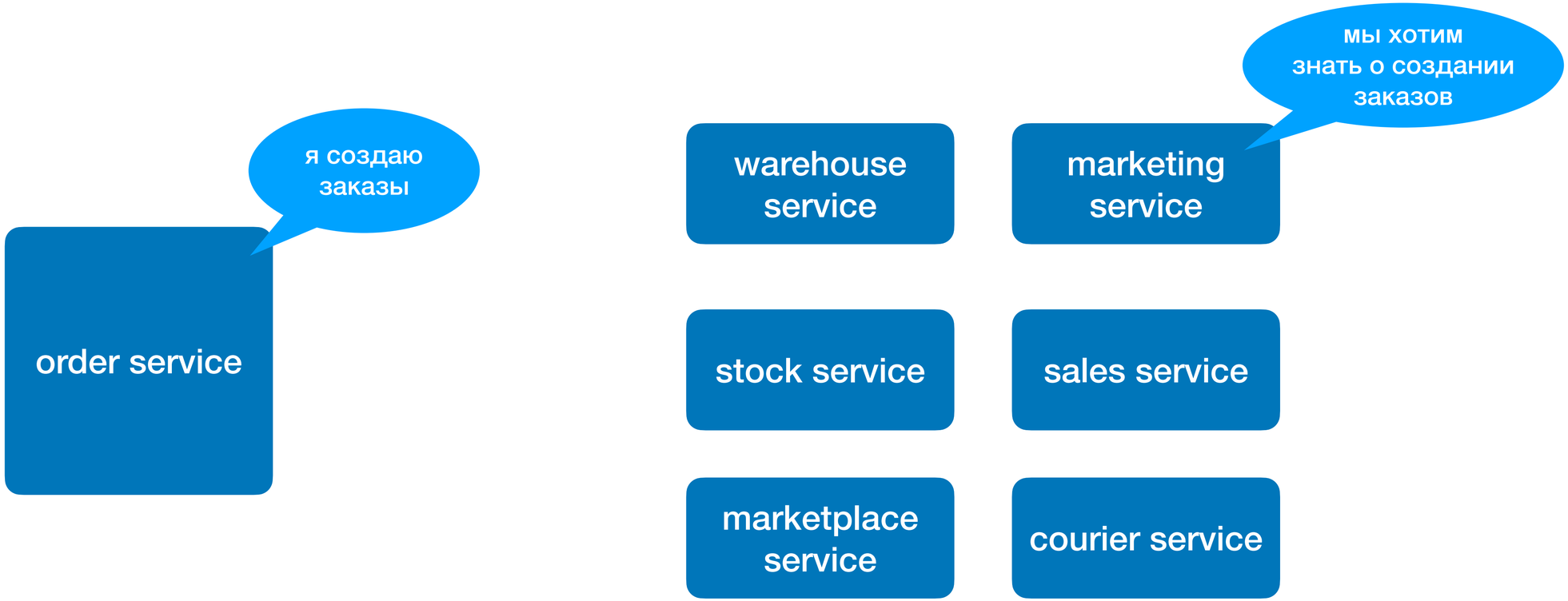


явное межсервисное взаимодействие

проблемы

явное межсервисное взаимодействие

проблемы



концепция реактивного межсервисного взаимодействия

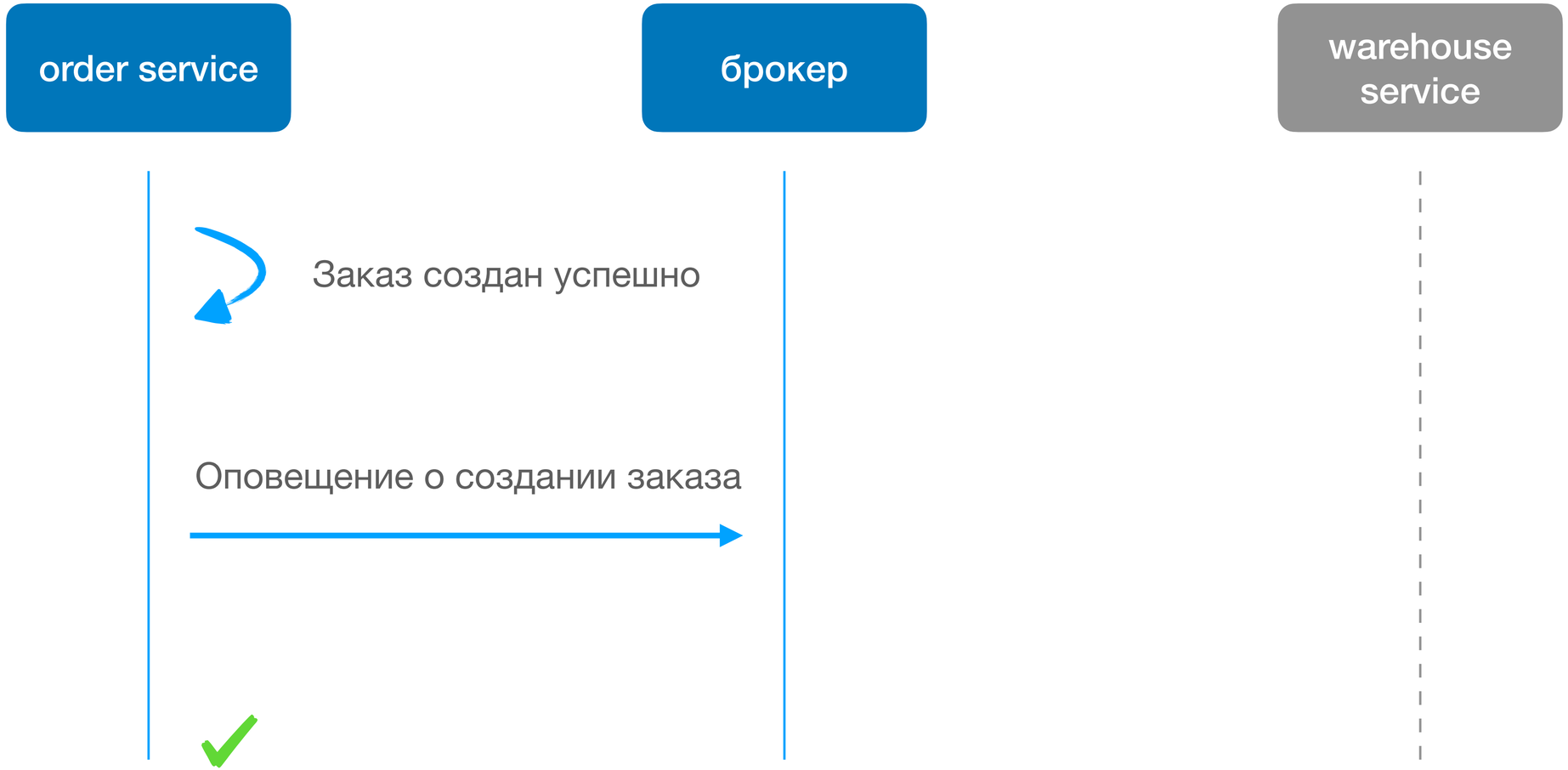
реактивное межсервисное взаимодействие

концепция

- отказ от явного межсервисного взаимодействия
- использование посредника (брокера) для отправки сообщений
- сервисы-отправители не завязаны на контракты сервисов-получателей
- сервисы-отправители не завязаны на ошибки сервисов-получателей
- ответственность за реализацию новых интеграций лежит на сервисах-получателях

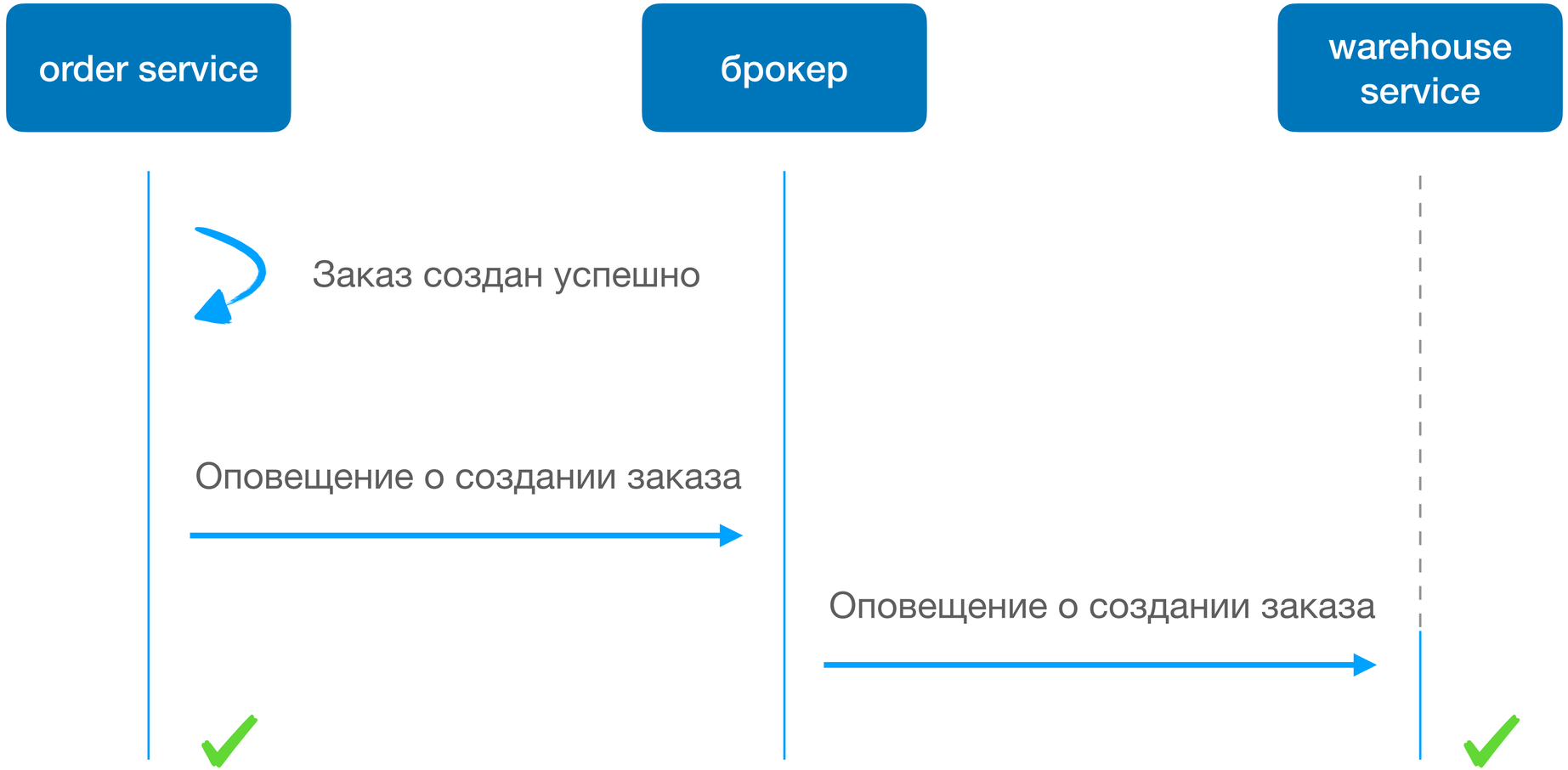
реактивное межсервисное взаимодействие

концепция



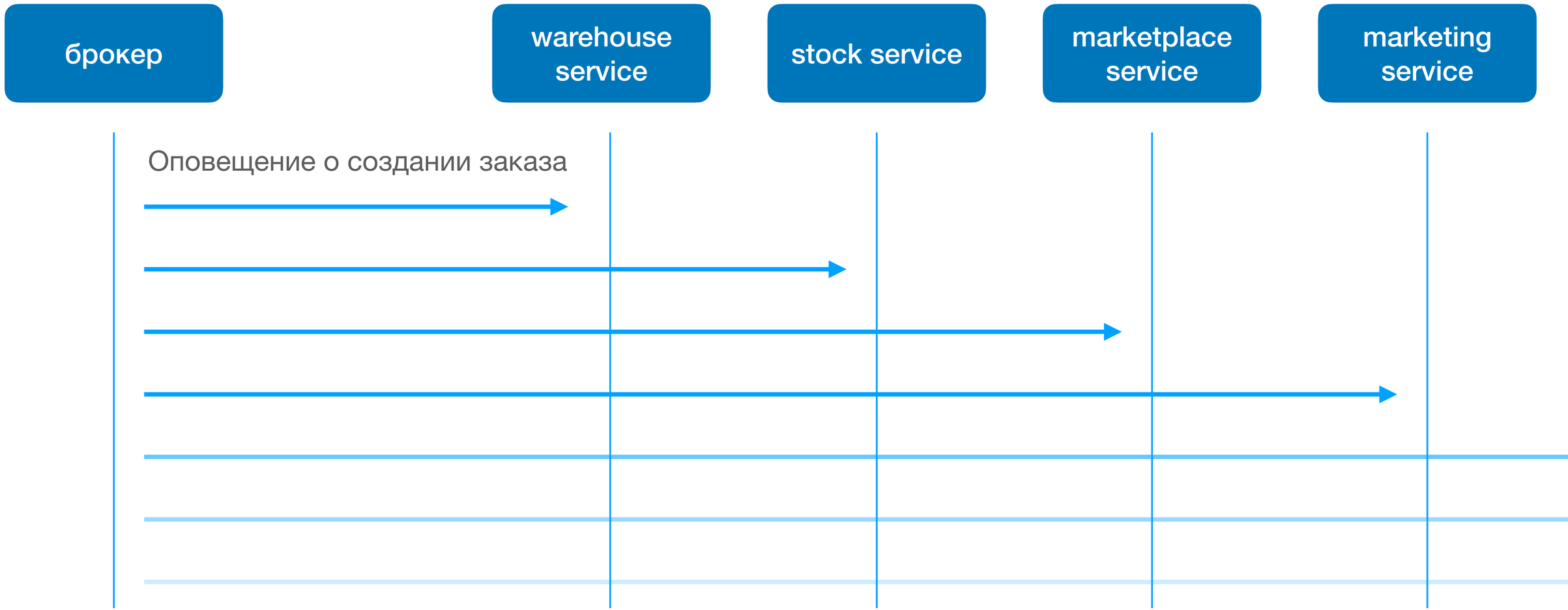
реактивное межсервисное взаимодействие

концепция



реактивное межсервисное взаимодействие

концепция



модели
работы с брокерами

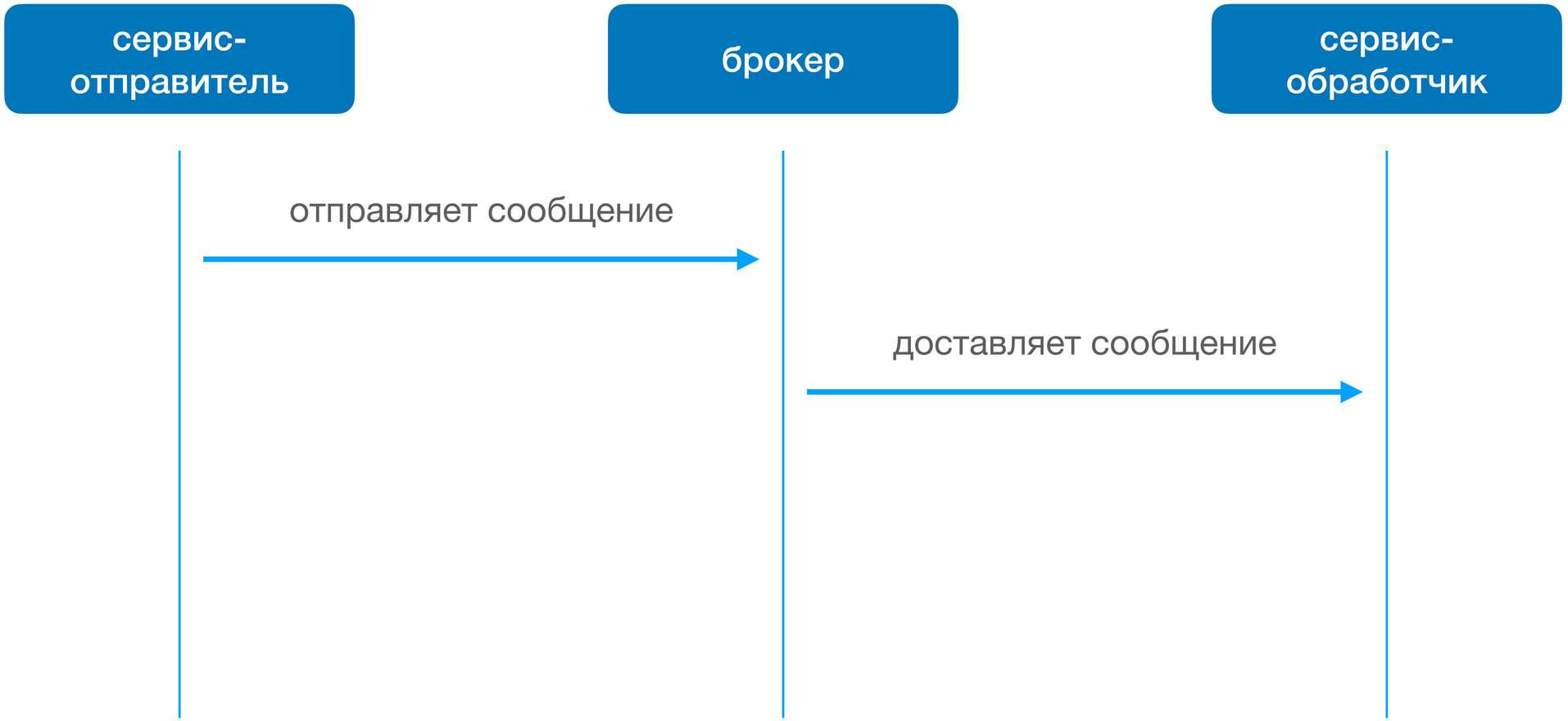
модели работы с брокерами

push модель

- ответственность за маршрутизацию и доставку сообщений лежит на брокере
- более сложная реализация брокера, потенциально более низкая производительность
- позволяет инициировать работу не со стороны обработчика сообщений (удобно с облачными функциями)

модели работы с брокерами

push модель



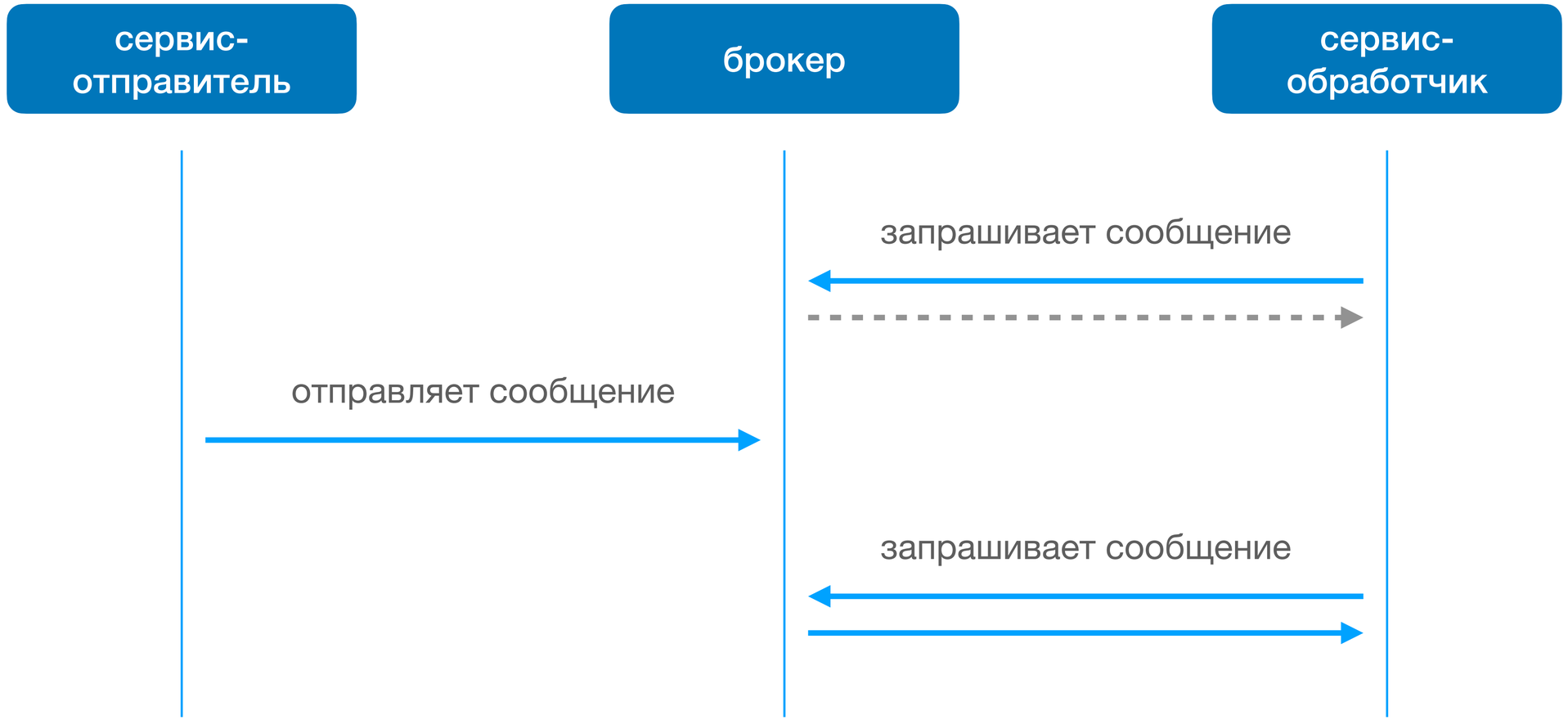
модели работы с брокерами

pull модель

- брокер не несёт ответственность за доставку сообщений
- сервисы-обработчики сами обращаются к брокеру для получения сообщений
- более простая реализация брокера, потенциально большая производительность

модели работы с брокерами

pull модель



стратегии реактивного взаимодействия

стратегии реактивного взаимодействия

event publisher

- подразумевает одностороннее взаимодействие между сервисами
- сервис-издатель отправляет сообщения в брокер
- сервис-подписчик получает и обрабатывает сообщения из брокера

- более низкая связанность операций
- абстрагируемся от устройства обеих сторон

стратегии реактивного взаимодействия

input/output channel

Интеграция через явное взаимодействие может быть не применима если:

- операция выполняется крайне долго
- операция требует подтверждения/каких-либо действий от пользователей стороннего сервиса
- сервис имеет крайне большое количество обращений для вызова этой операции

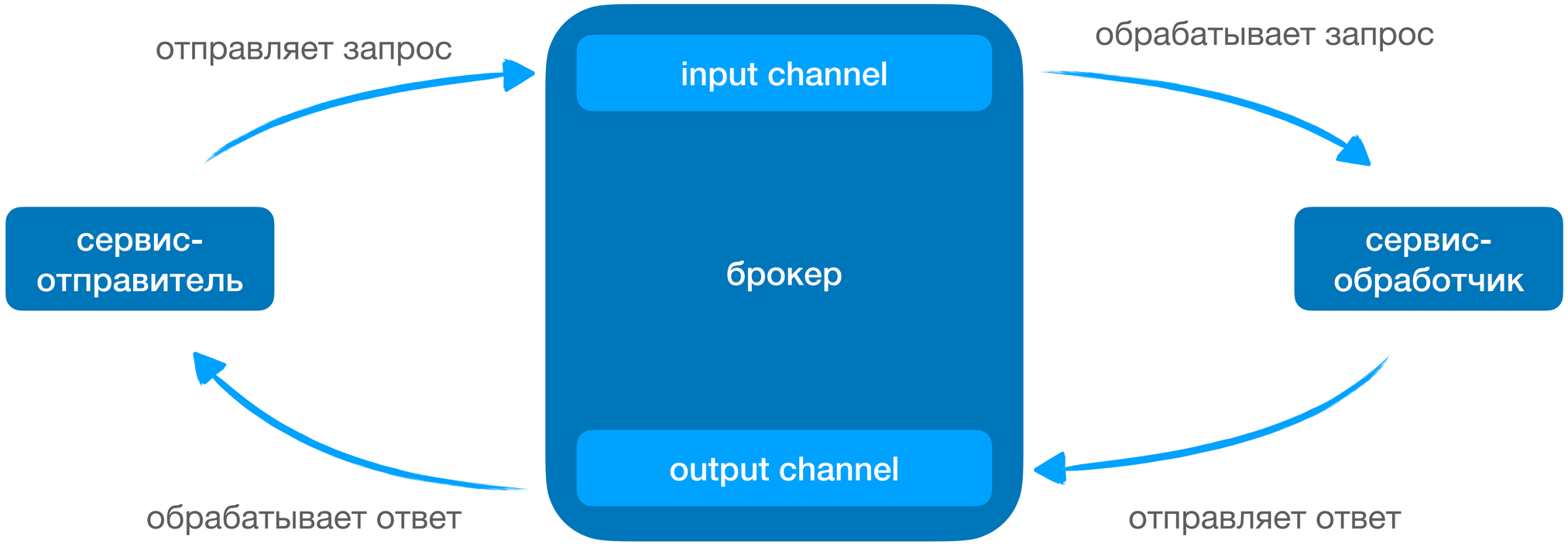
стратегии реактивного взаимодействия

input/output channel

- используем брокер в качестве входного и выходного канала какой-либо операции
- есть один поток сообщений-запросов
- есть один поток сообщений-результатов
- сервис обрабатывает запросы по мере своих сил
- сервис-обработчик абстрагирован от клиентов куда нужно доставить результат

стратегии реактивного взаимодействия

input/output channel



input/output channel

correlation id

- какой-либо идентификатор операции
- пишется сервисом-обработчиком в результат
- используется сервисом-отправителем для сопоставления результата

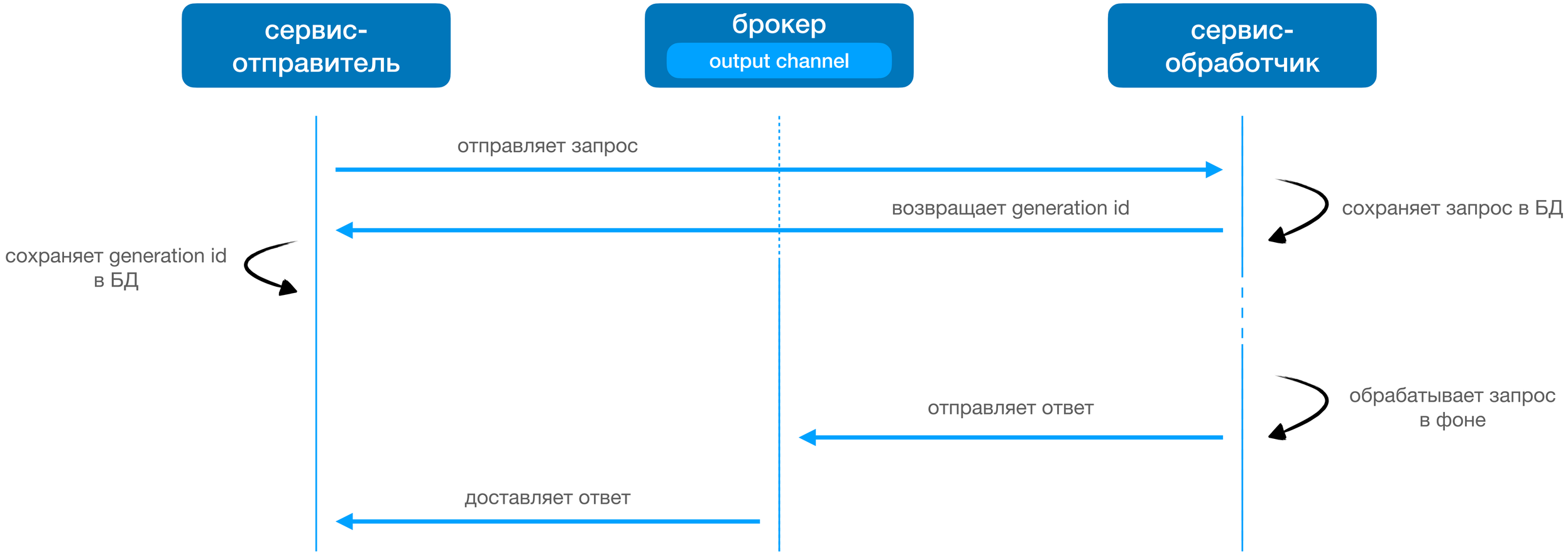
correlation id

generation id

- идентификатор, создаваемый сервисом-обработчиком
- возможно когда сервис инициирует операцию явно, но обрабатывает её уже в фоне

correlation id

generation id



correlation id

idempotence key

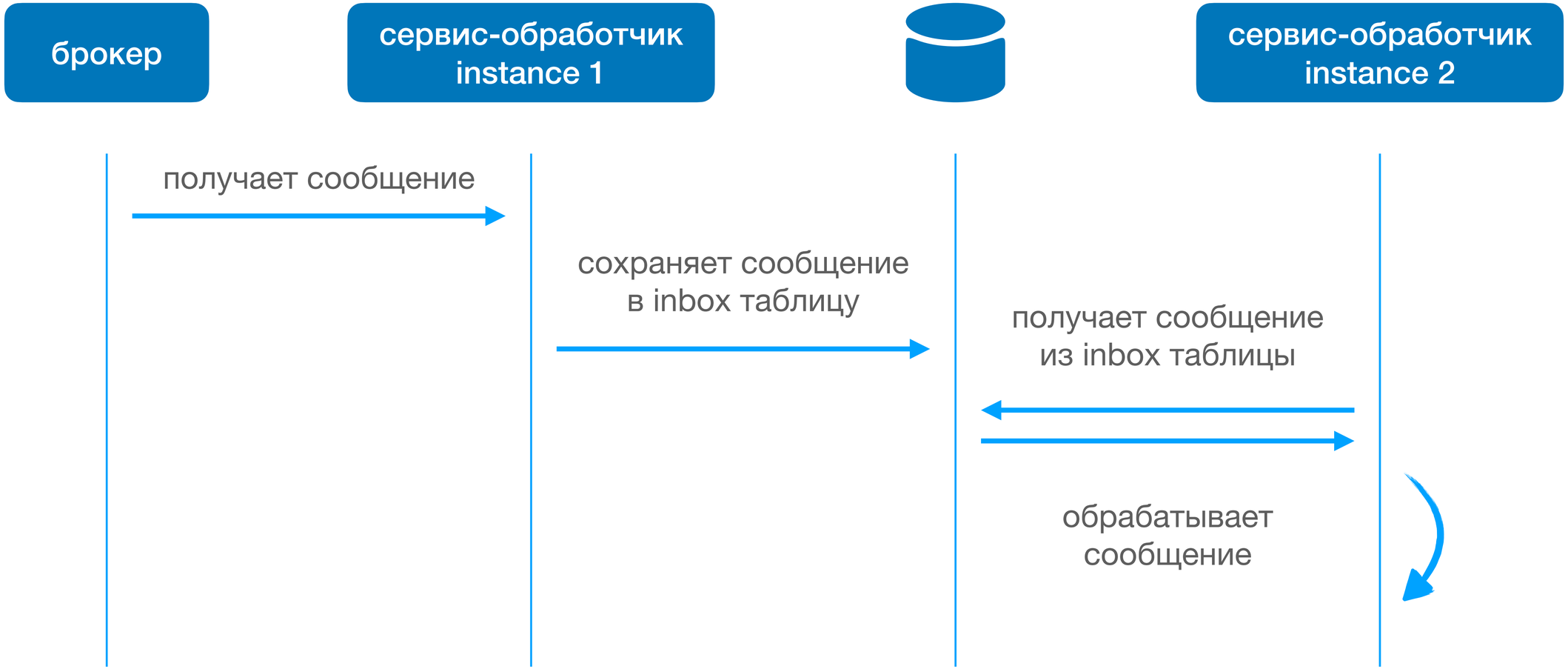
- идентификатор, передаваемый в запрос сервисом-отправителем
- может быть использован для дедубликации запросов
- используется и при явном взаимодействии

паттерны
inbox и outbox

inbox

- при получении сообщений – складываем их в базу данных
- сообщения из базы данных обрабатываем в фоновом режиме
- позволяет предотвратить сбор лага при рассинхроне в различных потоках сообщений

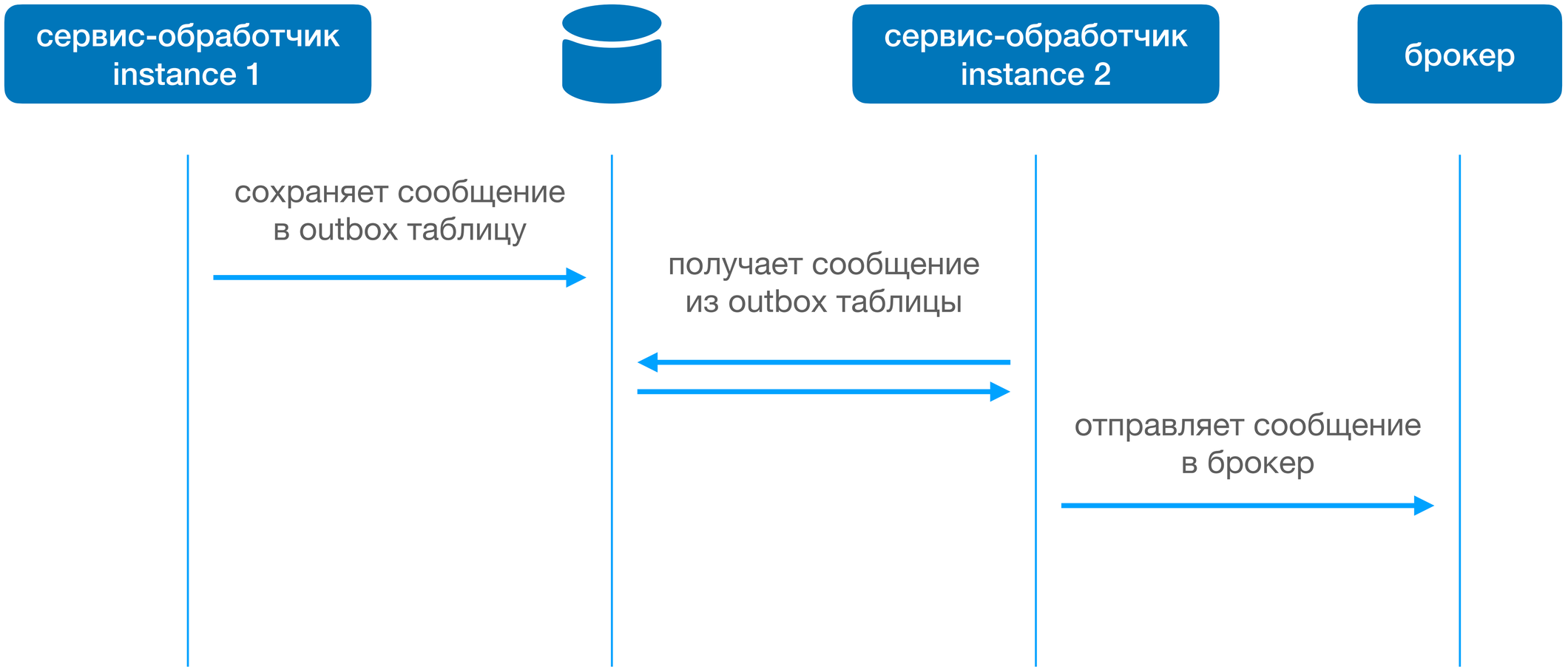
inbox


брокер
сервис-обработчик instance 1
сервис-обработчик instance 2
получает сообщение
сохраняет сообщение в inbox таблицу
получает сообщение из inbox таблицы
обрабатывает сообщение

outbox

- при необходимости отправки сообщений – складываем их в базу данных
- в фоне разбираем outbox таблицу и отправляем сообщения брокеру
- позволяет повысить отказоустойчивость с случаях отказа брокера
- позволяет использовать транзакции для отправки сообщений в брокер

outbox


сервис-обработчик
instance 1
сервис-обработчик
instance 2
брокер
сохраняет сообщение
в outbox таблицу
получает сообщение
из outbox таблицы
отправляет сообщение
в брокер

event driven architecture

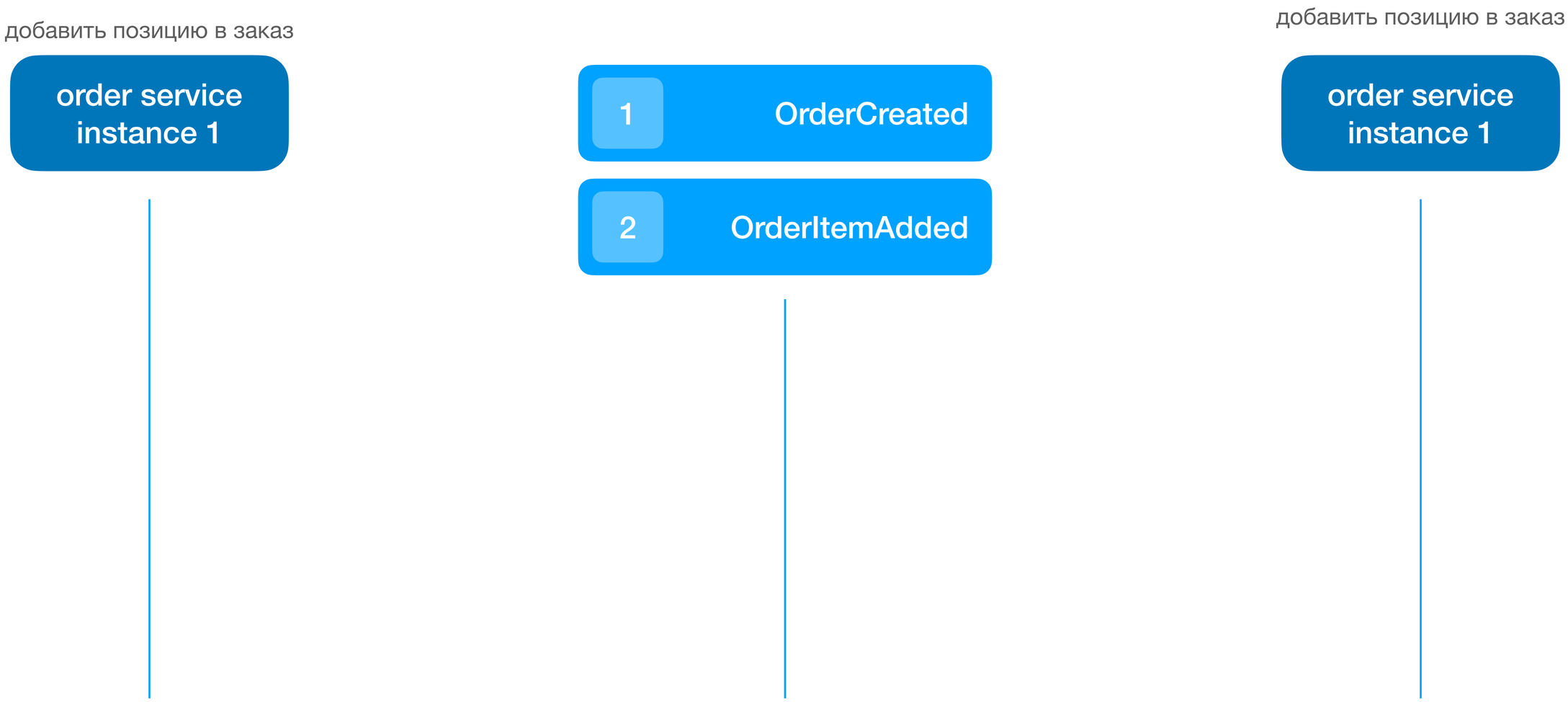
event driven architecture

event sourcing

- данные сущностей хранятся в виде последовательности событий
- на основе этих событий в логике приложения собираются “слепки” сущностей
- позволяет реализовывать атомарность при конкурентных изменениях за счёт оптимистичных блокировок

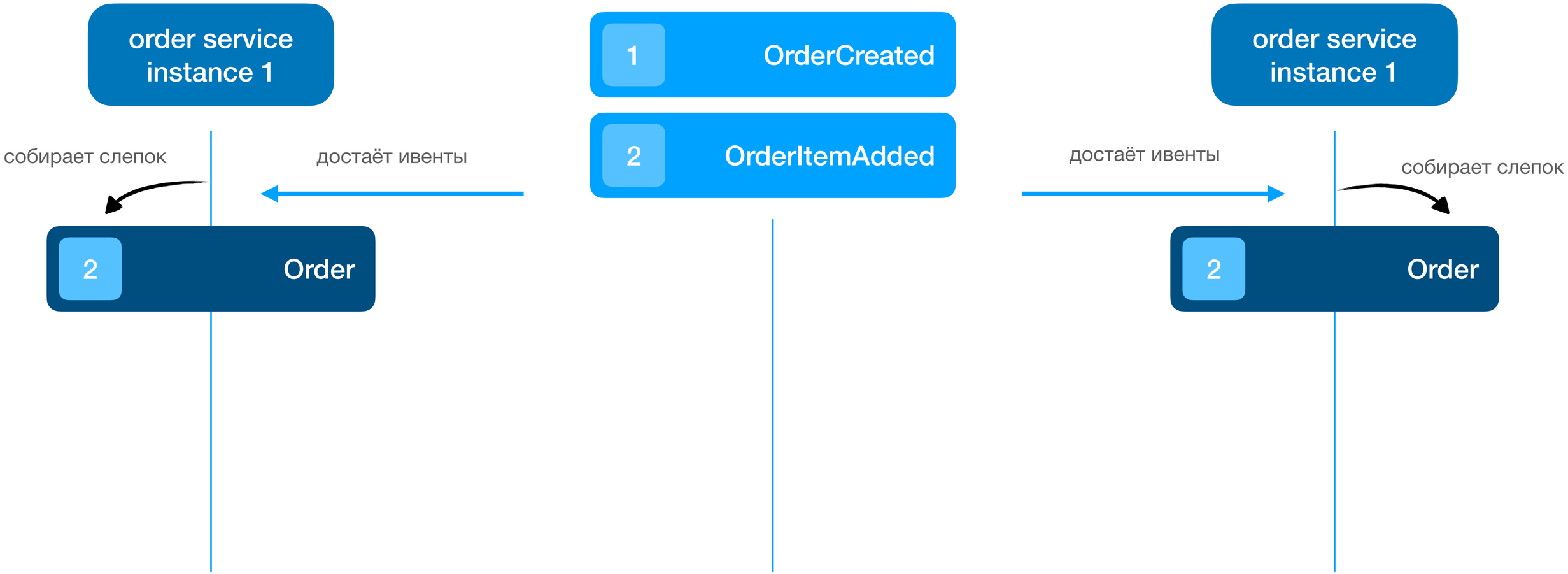
event driven architecture

event sourcing



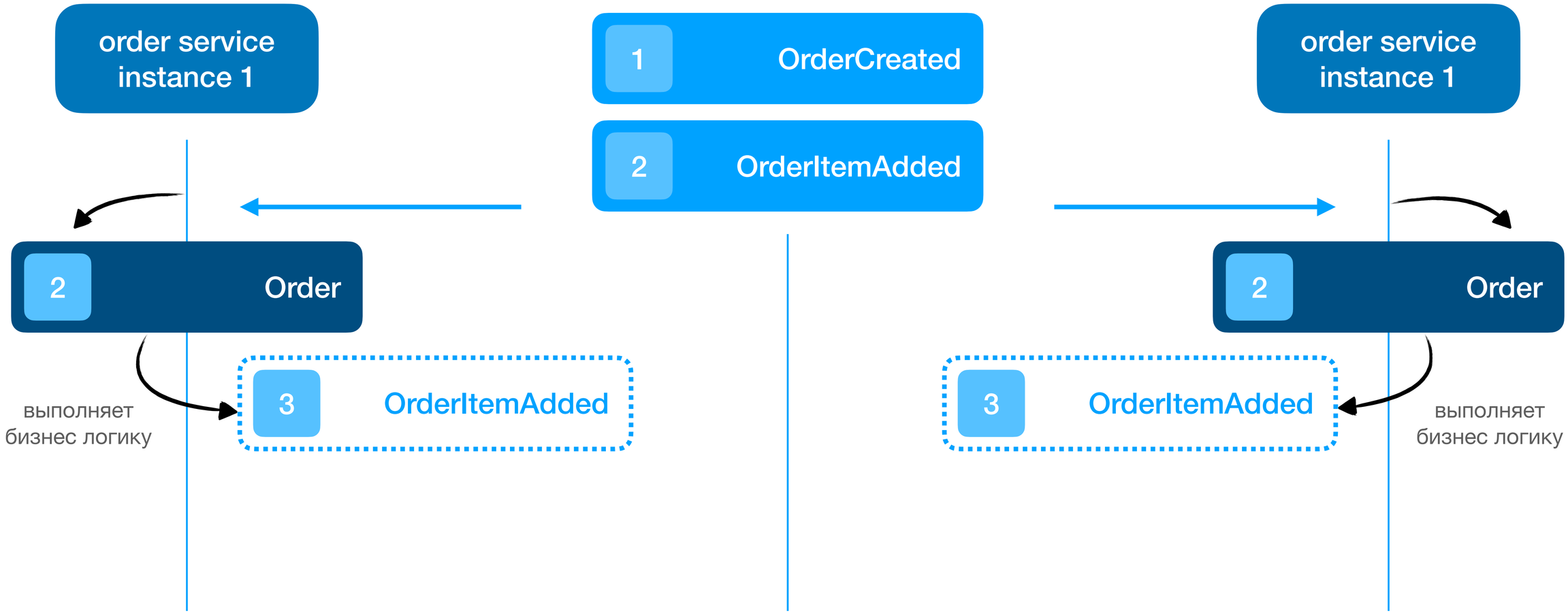
event driven architecture

event sourcing



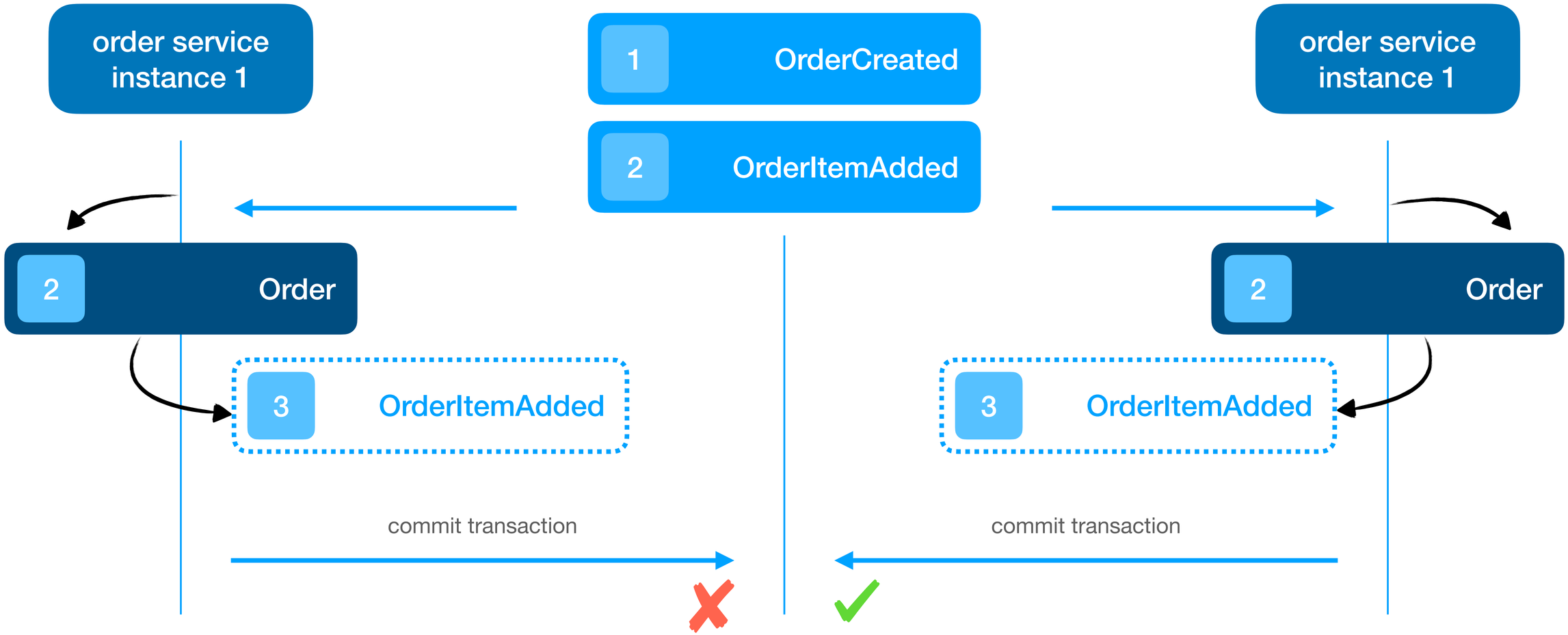
event driven architecture

event sourcing



event driven architecture

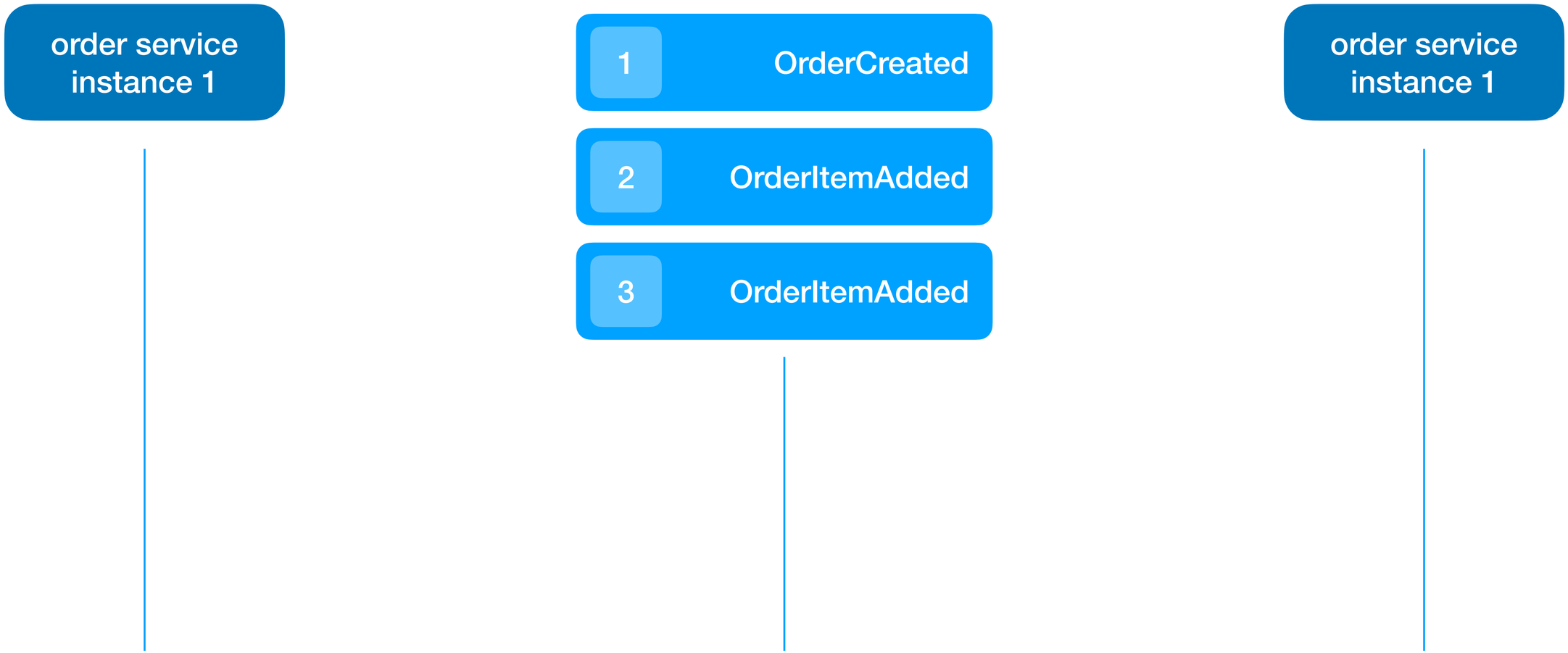
event sourcing



event driven architecture

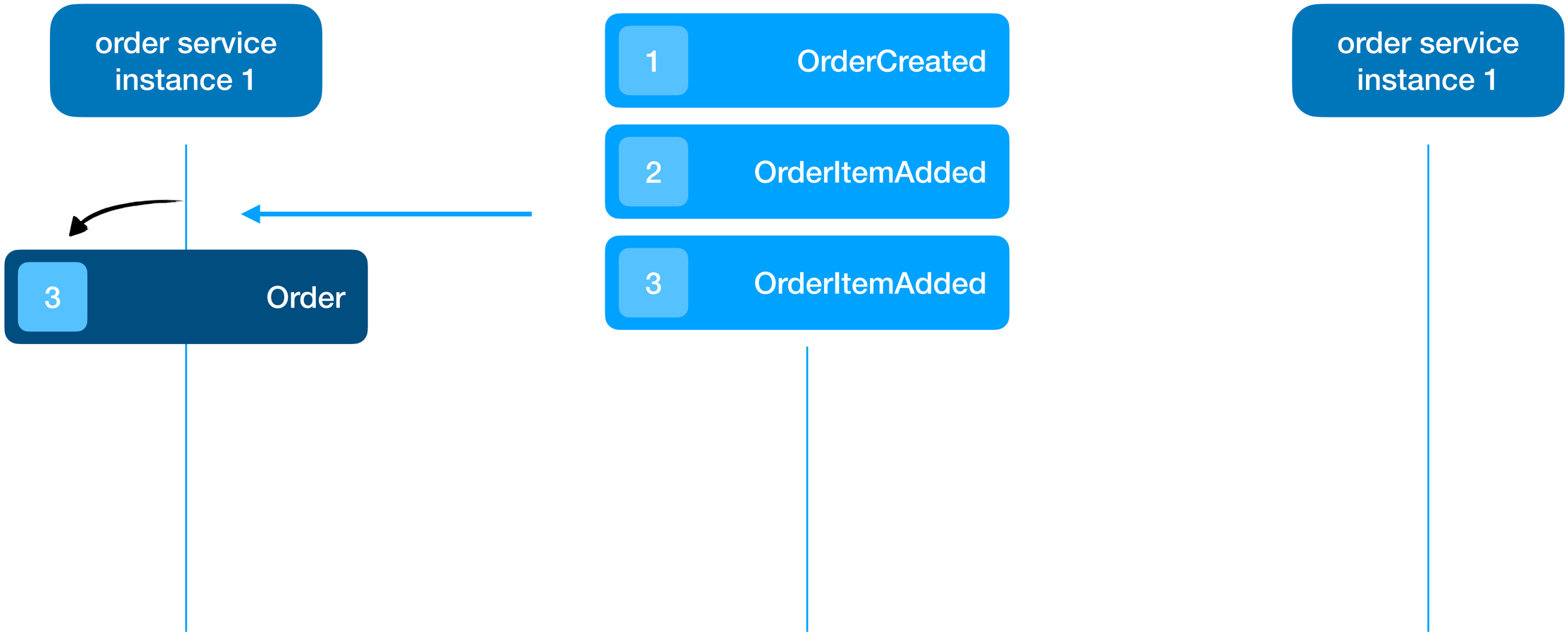
event sourcing

повторяет попытку
выполнения операции



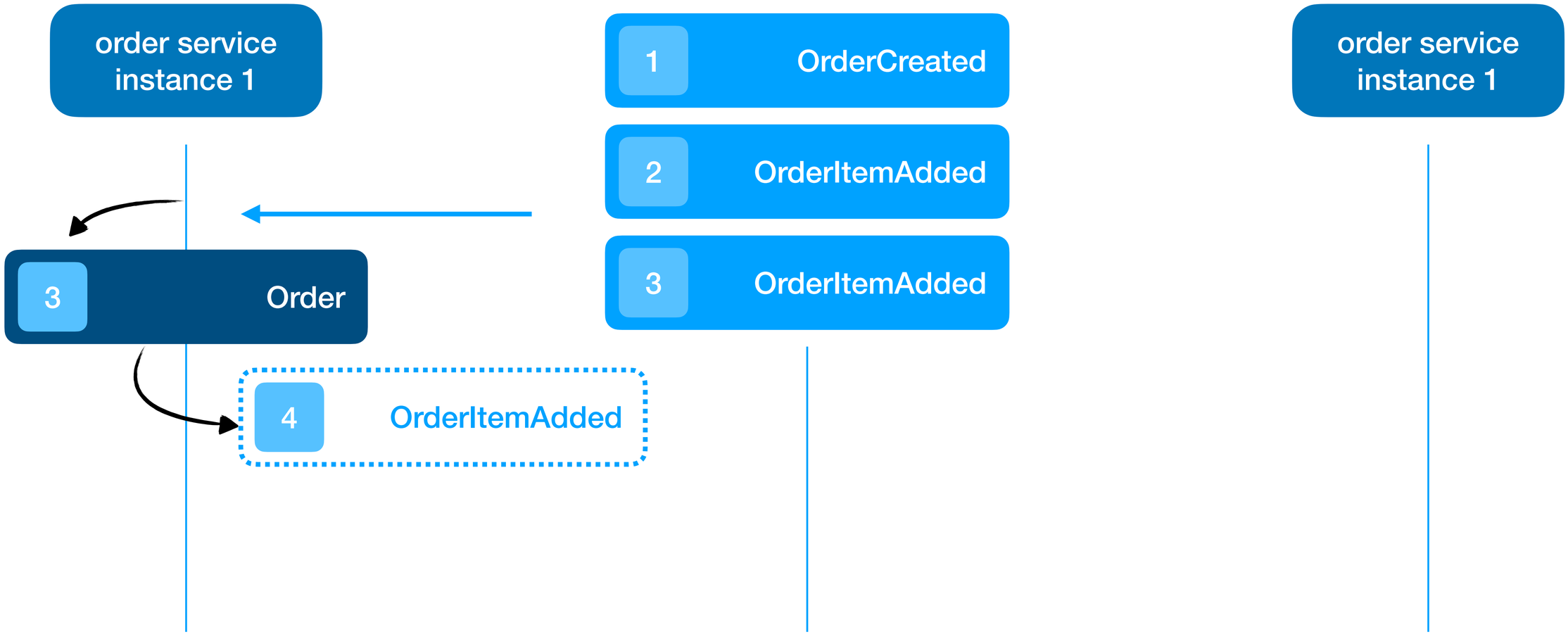
event driven architecture

event sourcing



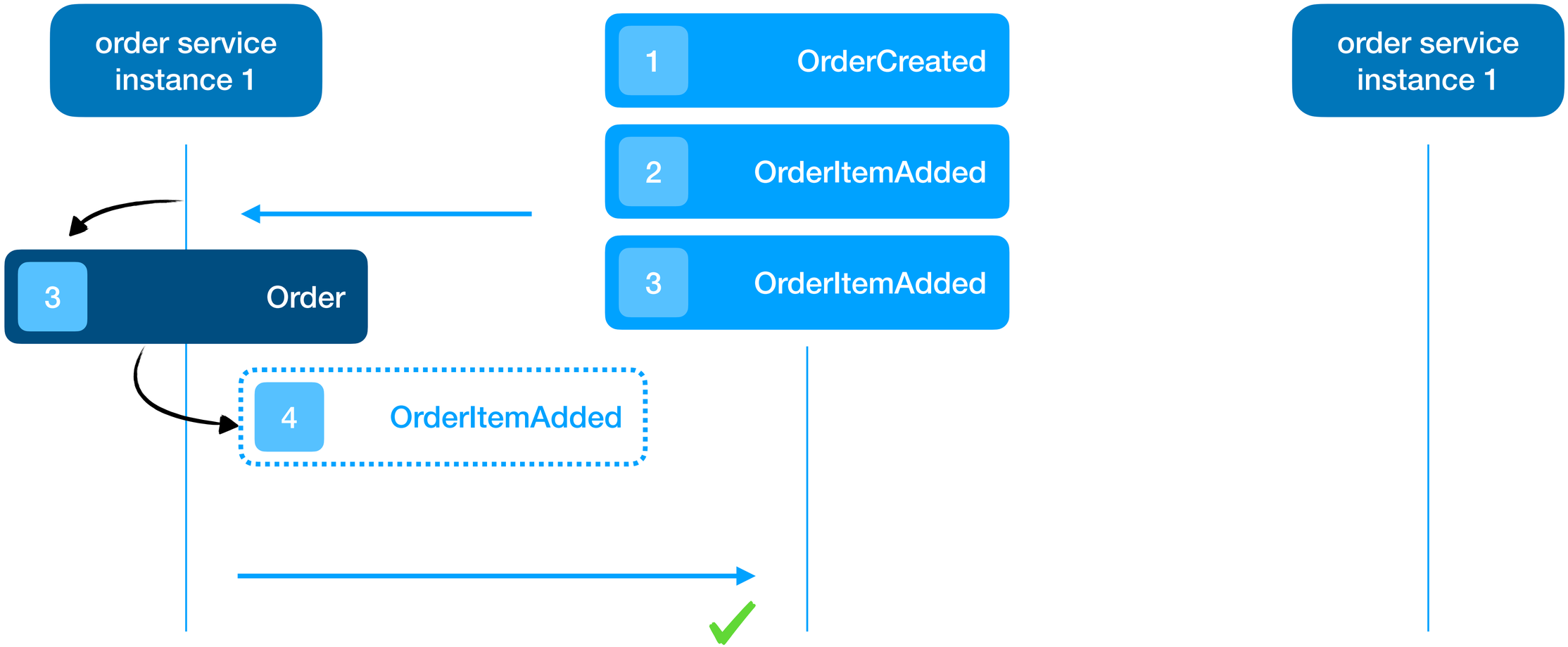
event driven architecture

event sourcing



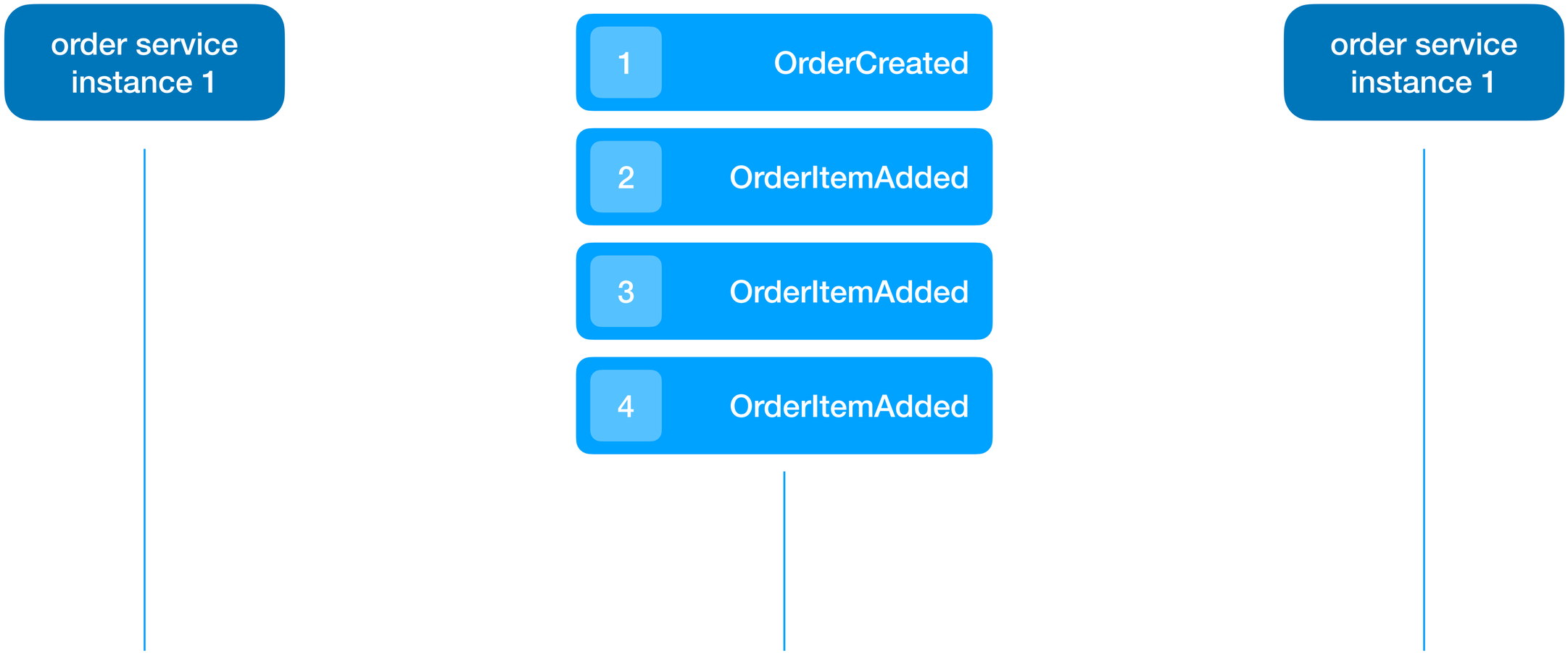
event driven architecture

event sourcing



event driven architecture

event sourcing



event driven architecture

read model

- чем больше ивентов по сущности – тем дороже формировать её слепок
- read model – отдельная модель данных, где хранится слепок сущности
- позволяет успростить операции получения данных за счёт денормализации
- позволяет упростить добавления атрибутов слепка за счёт возможности пересчёта read модели