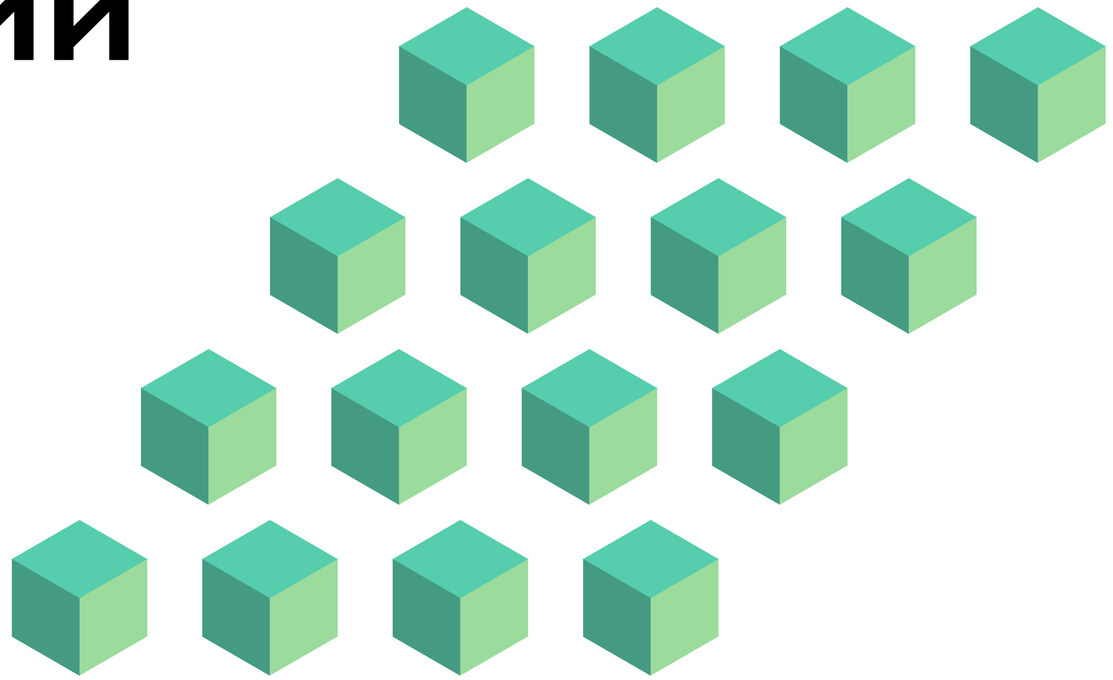
Основные принципы микросервисов и их реализации



Микросервисы

1. Что это такое?
2. Как это оценить?
3. Зачем это нужно?
4. Как это делать?

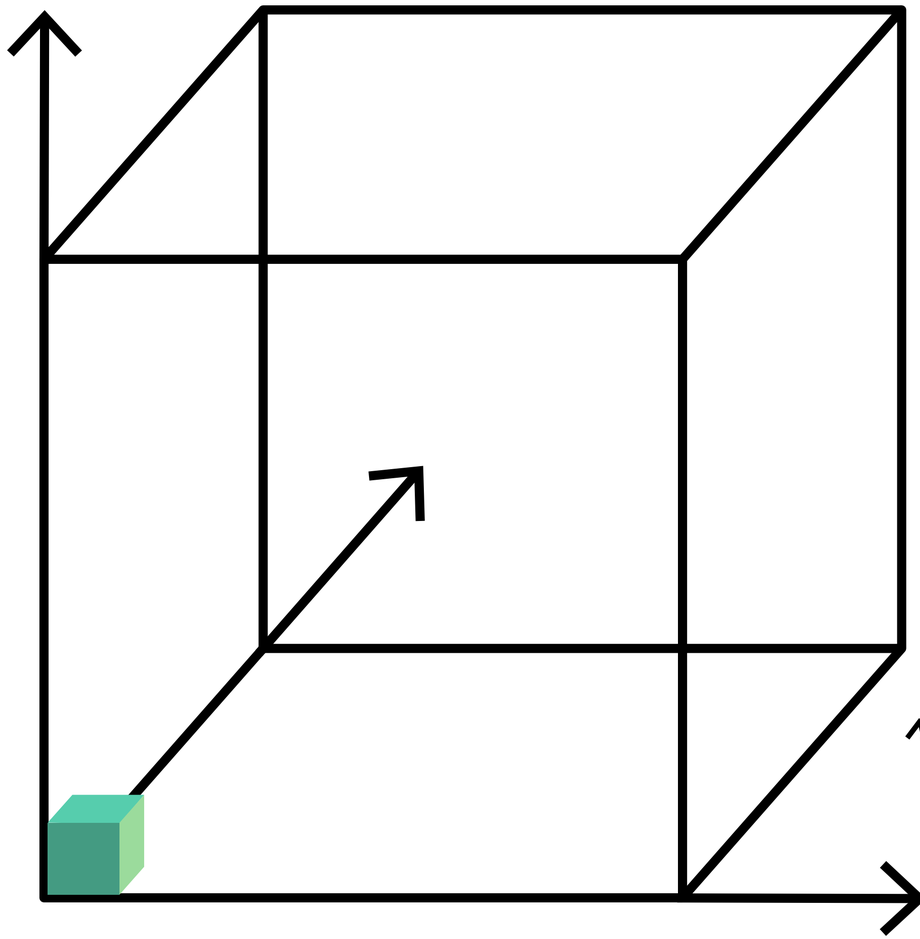
1

Что такое микросервисы?

Определение микросервисов

1. Набор независимых, но связанных между собой сервисов
2. Построены вокруг бизнес-потребностей
3. Развертываются независимо
4. Имеют минимум централизованного управления

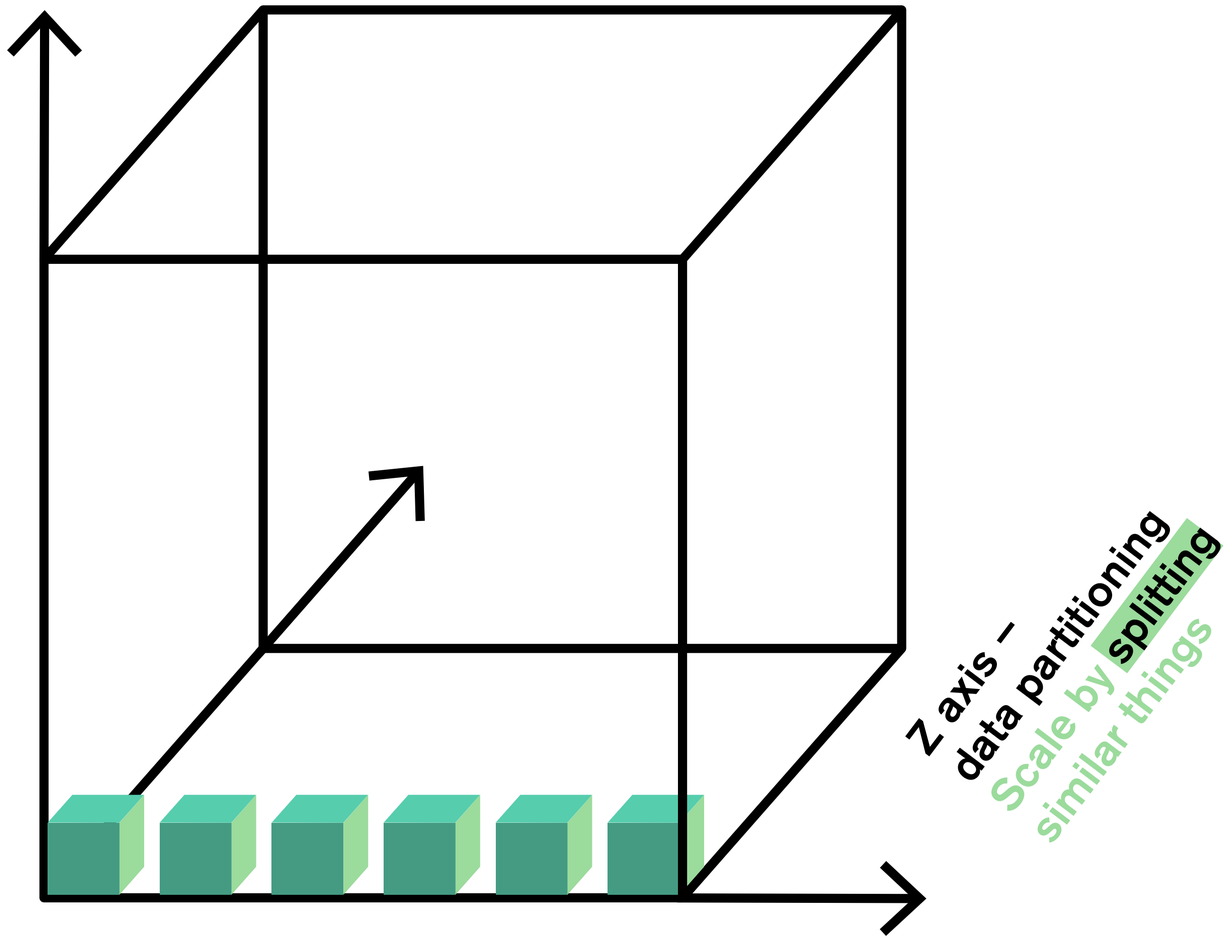
Y axis –
functional
decomposition
Scale by **splitting**
different things



Z axis –
data partitioning
Scale by **splitting**
similar things

X axis – horizontal duplication
Scale by **cloning**

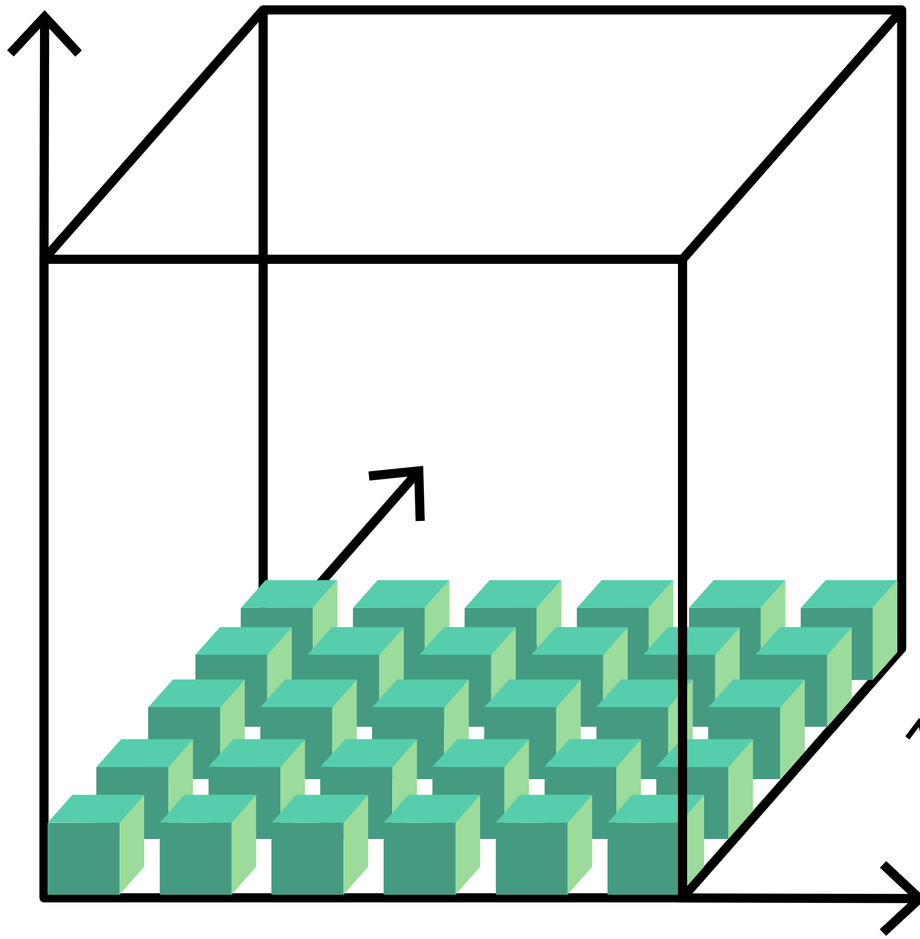
Y axis –
functional
decomposition
Scale by splitting
different things



Z axis –
data partitioning
Scale by splitting
similar things

X axis – horizontal duplication
Scale by cloning

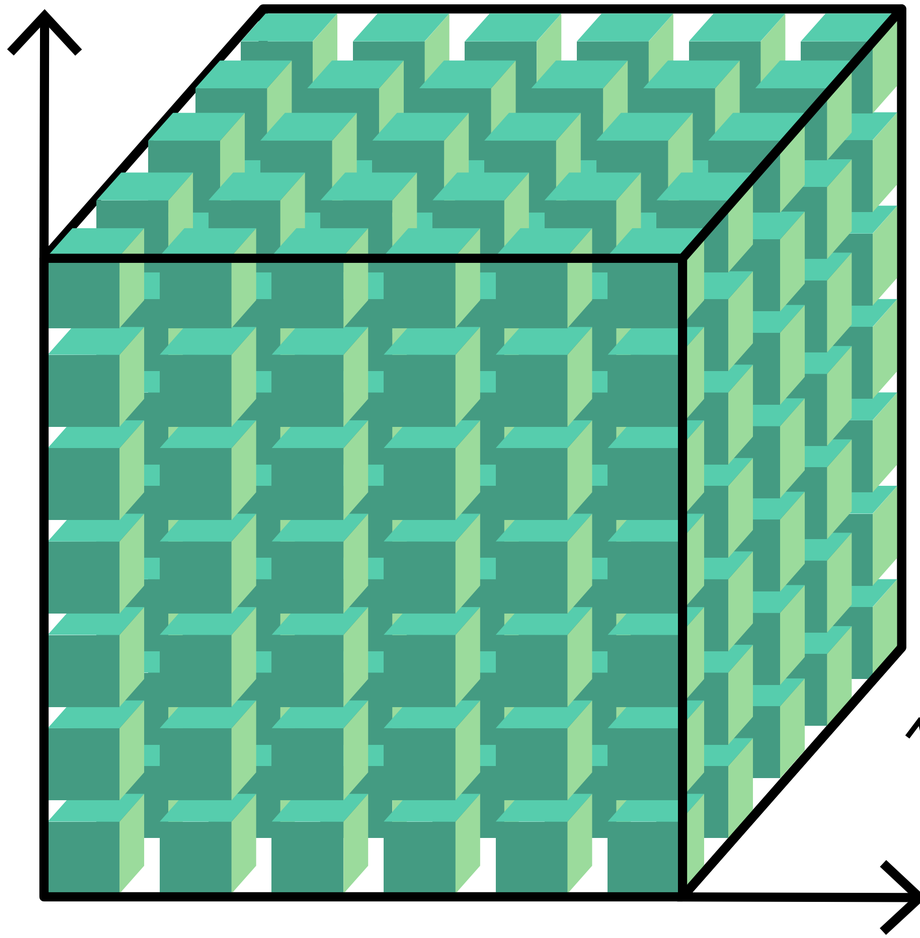
Y axis –
functional
decomposition
Scale by splitting
different things



Z axis –
data partitioning
Scale by splitting
similar things

X axis – horizontal duplication
Scale by cloning

Y axis –
functional
decomposition
Scale by **splitting**
different things



Z axis –
data partitioning
Scale by **splitting**
similar things

X axis – horizontal duplication
Scale by **cloning**

Обещания микросервисов

1. Инкапсуляция бизнес-логики
2. Независимость в выборе технологий
3. Независимость в развертывании отдельного сервиса
4. Независимое внесение изменений
5. Меньшее значение Time To Market
6. Быстрая адаптация к изменениям

Очевидные трудности

1. Распределенная система
2. Несогласованность данных
3. Сложность дизайна целой системы

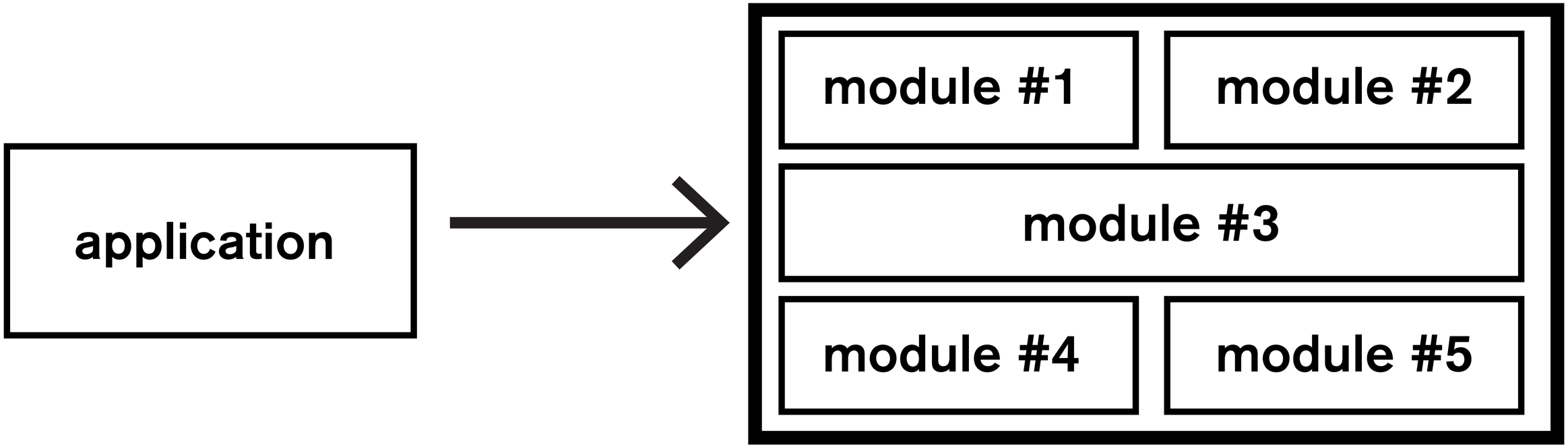
2

Что такое микросервисы?

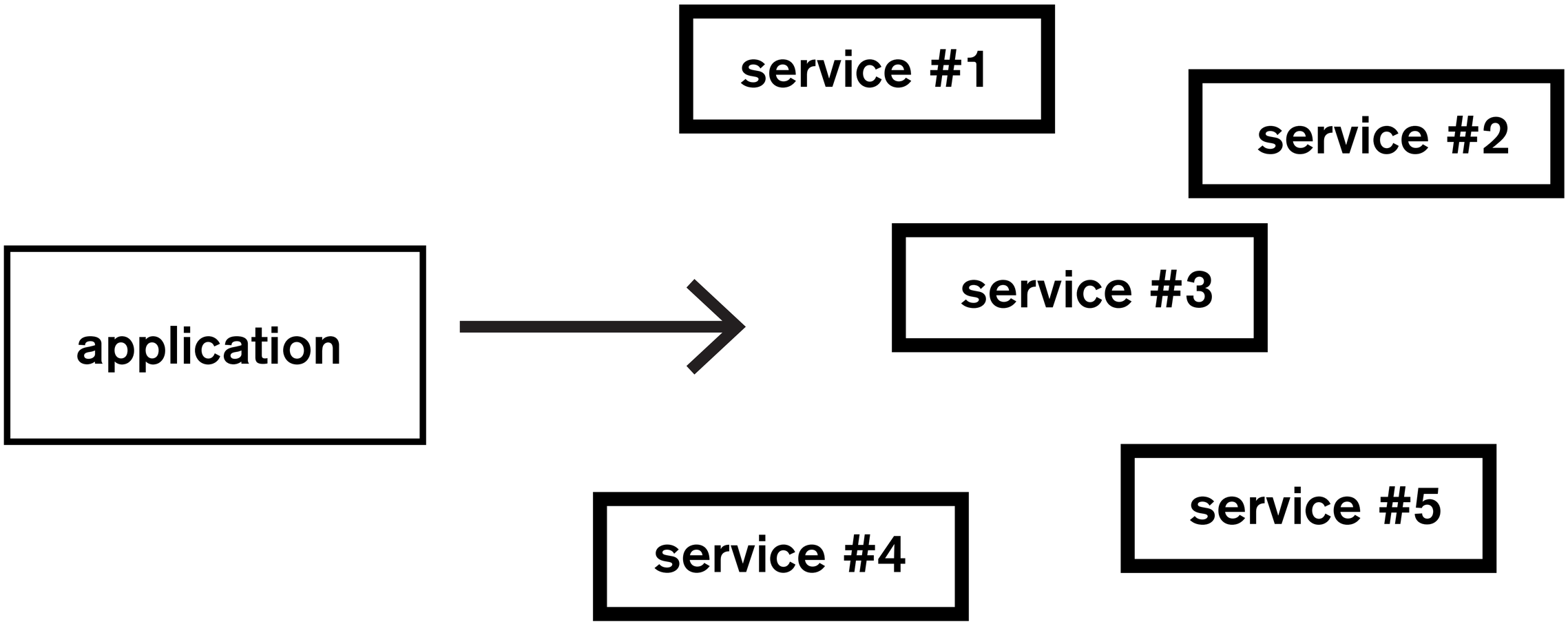
Параметры оценки

1. Гибкость при разработке
2. Простота развертывания
3. Тестируемость
4. Масштабируемость
5. Простота разработки
6. Производительность

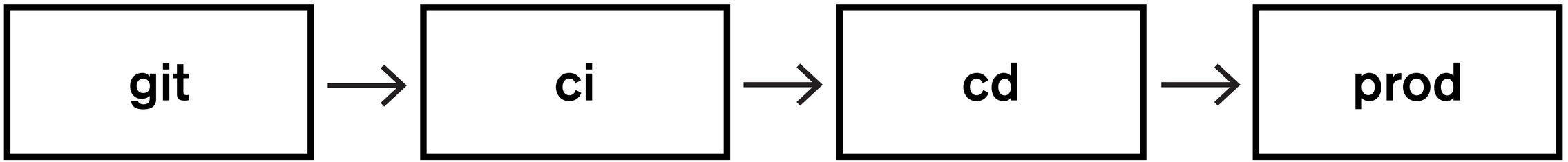
Гибкость при разработке, монолит



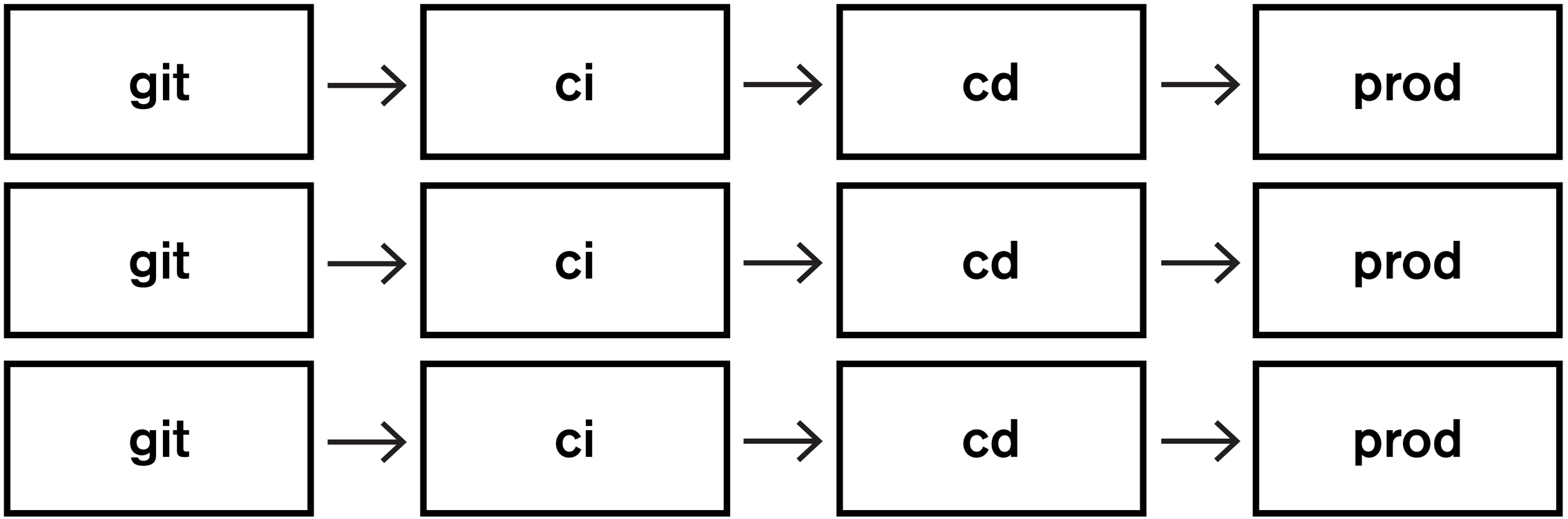
Гибкость при разработке, микросервисы



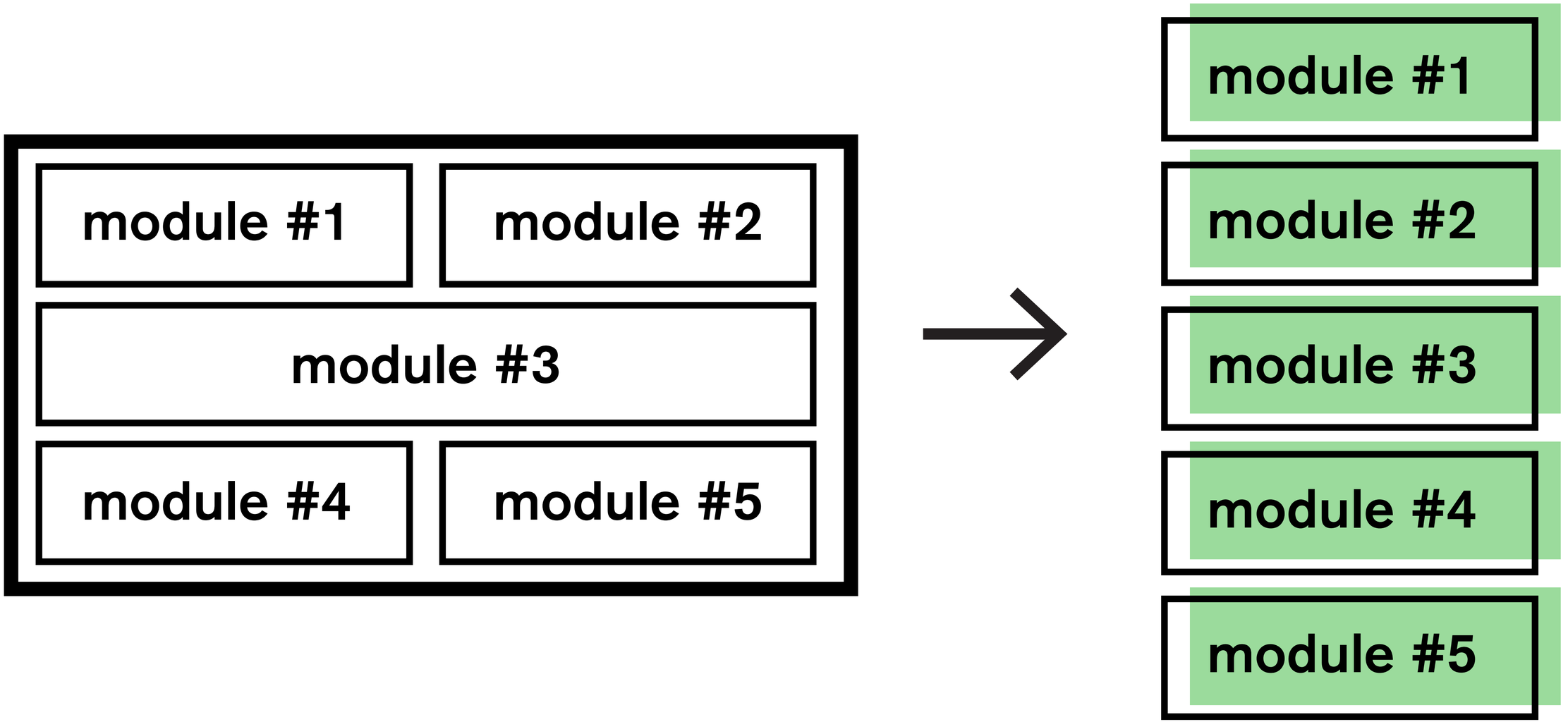
Простота развертывания, монолит



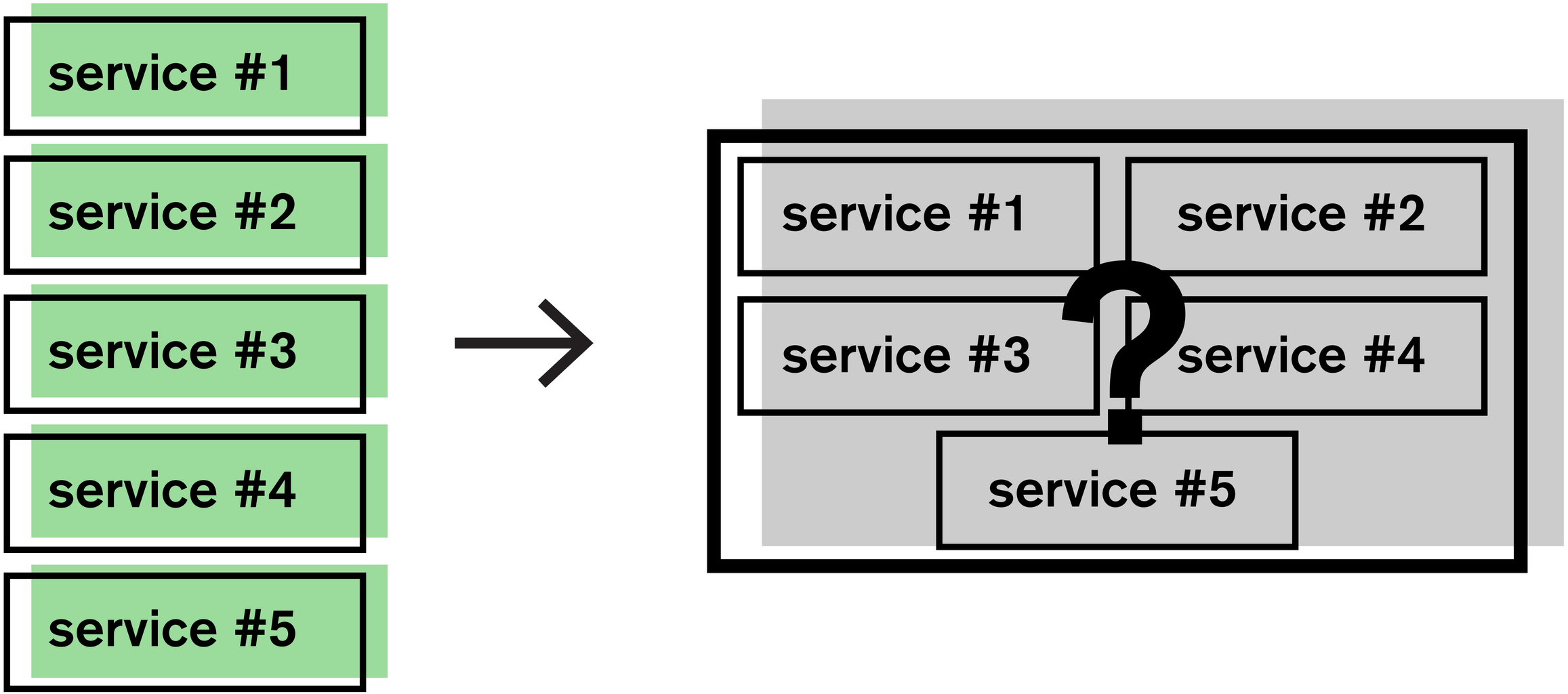
Простота развертывания, микросервисы



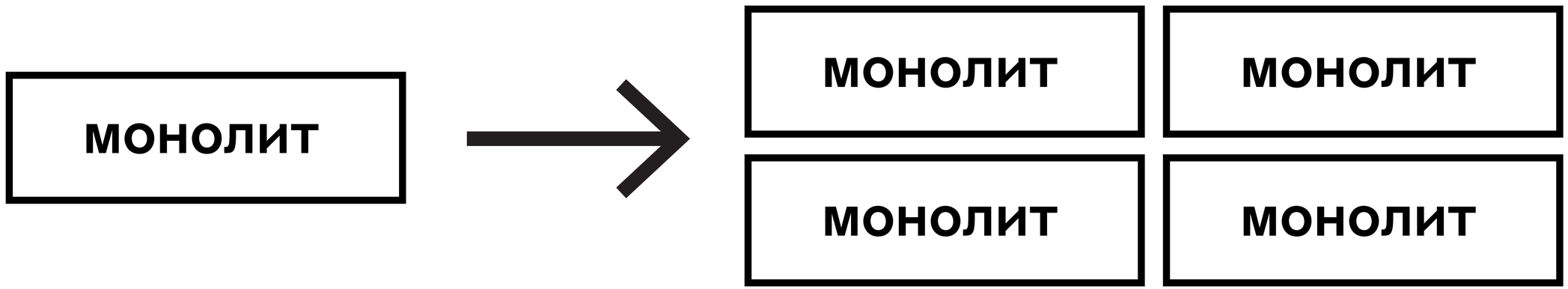
Тестируемость, монолит



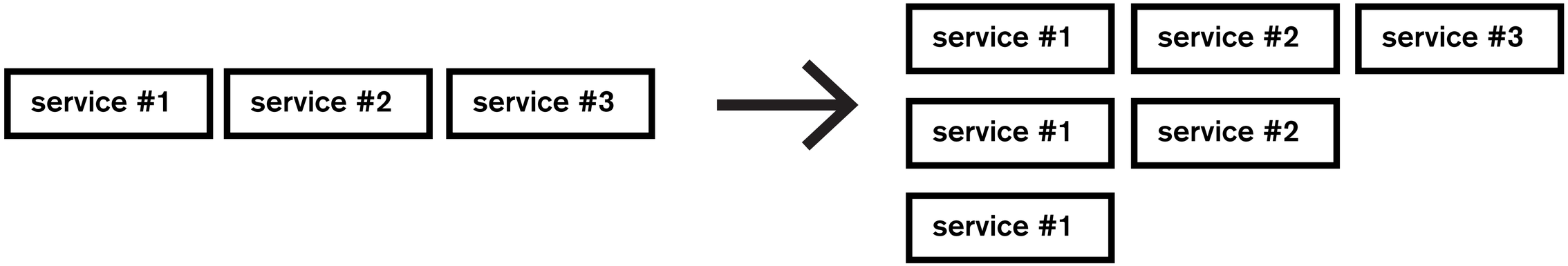
Тестируемость, микросервисы



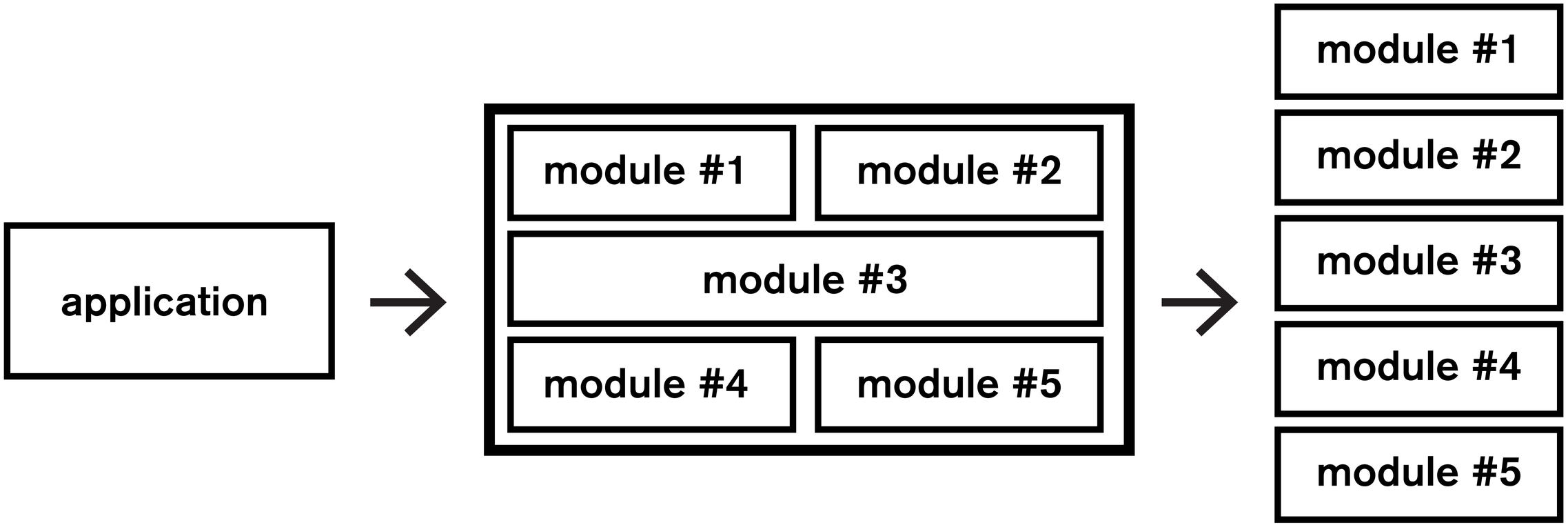
Масштабируемость, монолит



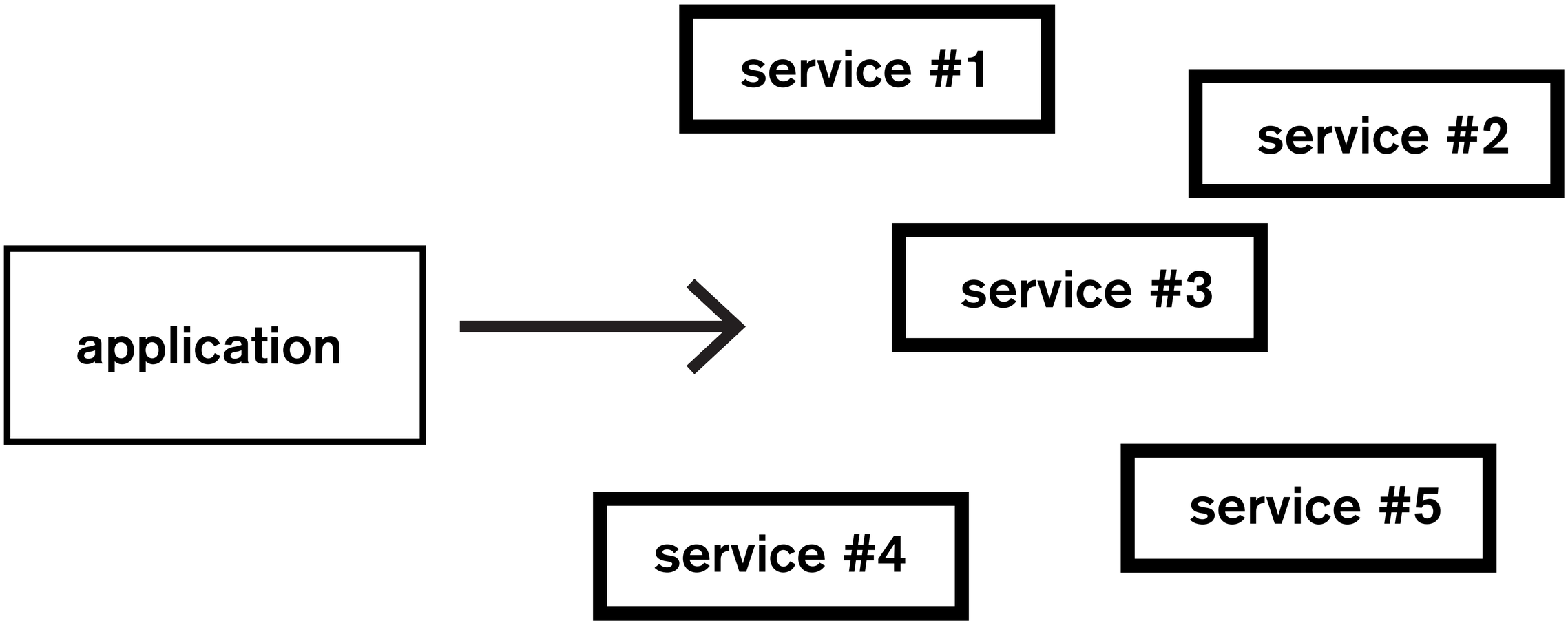
Масштабируемость, микросервисы



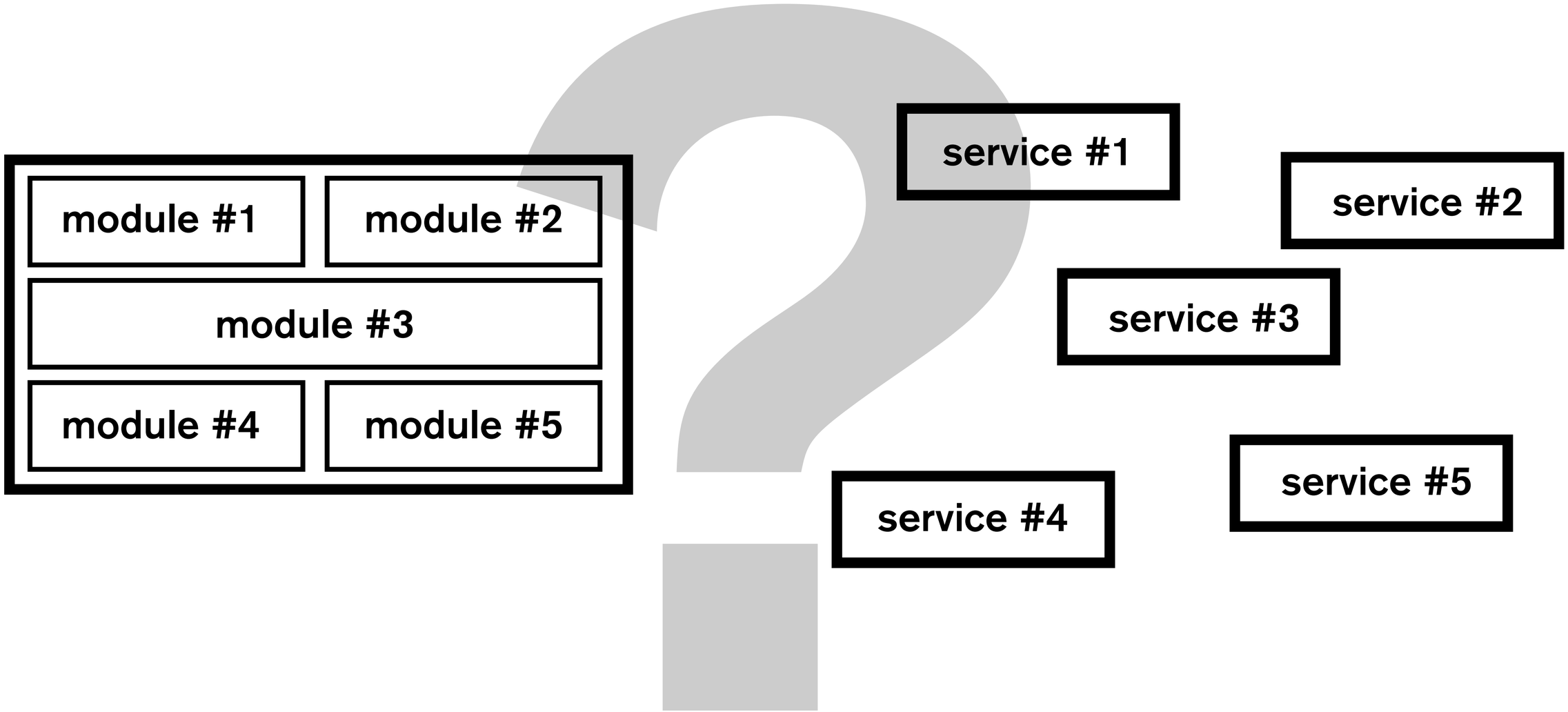
Простота разработки, монолит



Простота разработки, микросервисы

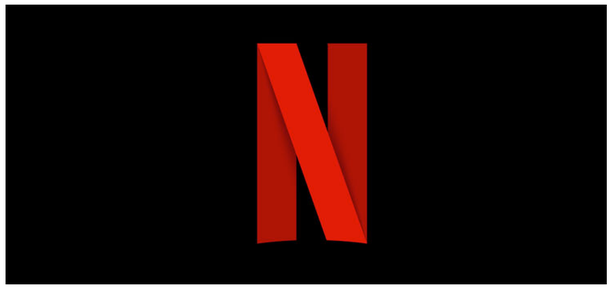


Производительность

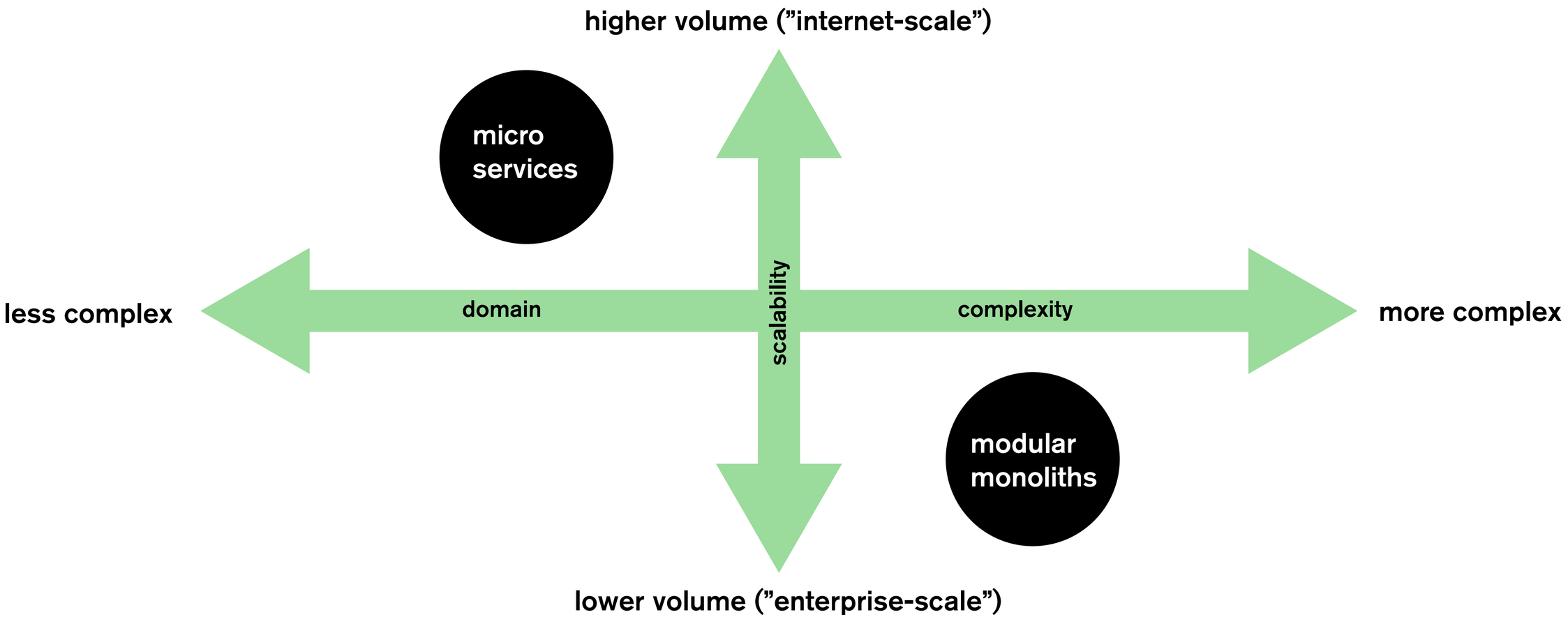



3
Зачем нужны микросервисы?

Кто их использует



Когда их использовать



Когда еще их использовать

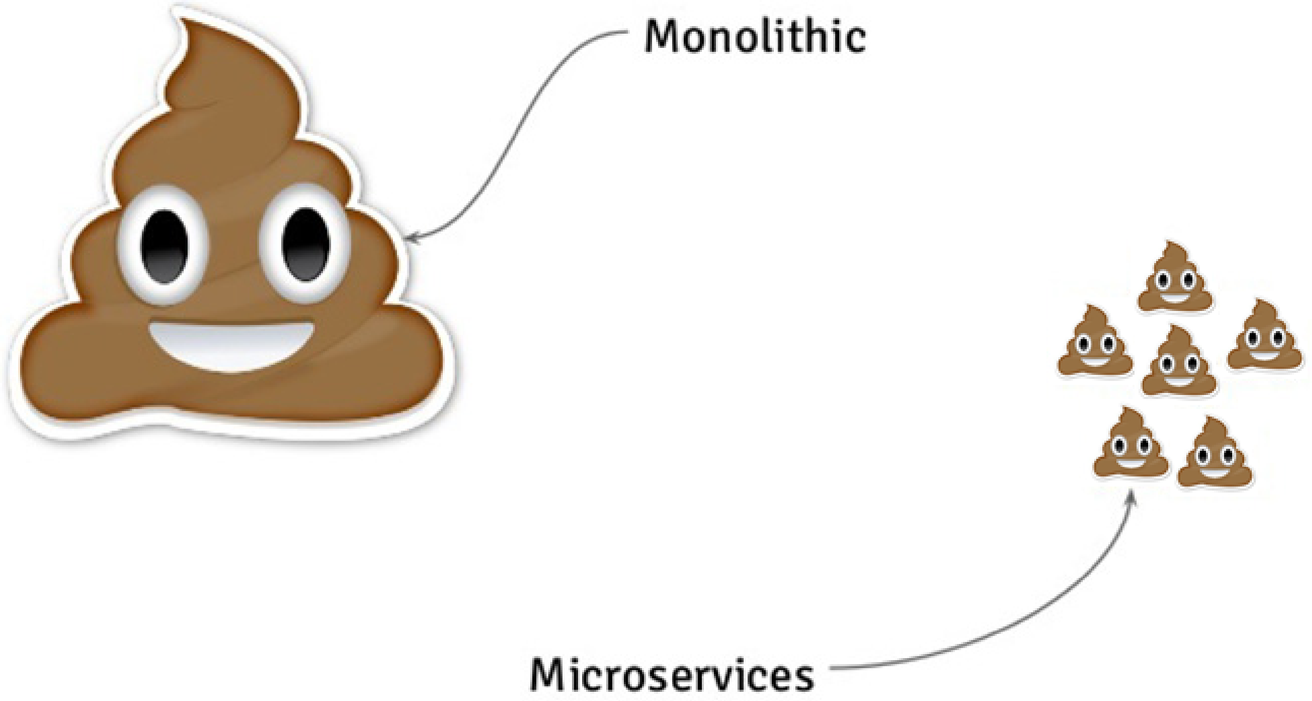
1. Необходимо выпускать новый функционал часто
2. Можно получить большие преимущества при написании разных частей системы на разных языках
3. Есть необходимость использовать разные базы данных для разных задач

Когда лучше не использовать

1. Не знаете зачем они
2. Нет соответствующей инфраструктуры

Álvaro Sánchez
@alvaro_sanchez

Monolithic vs Microservices



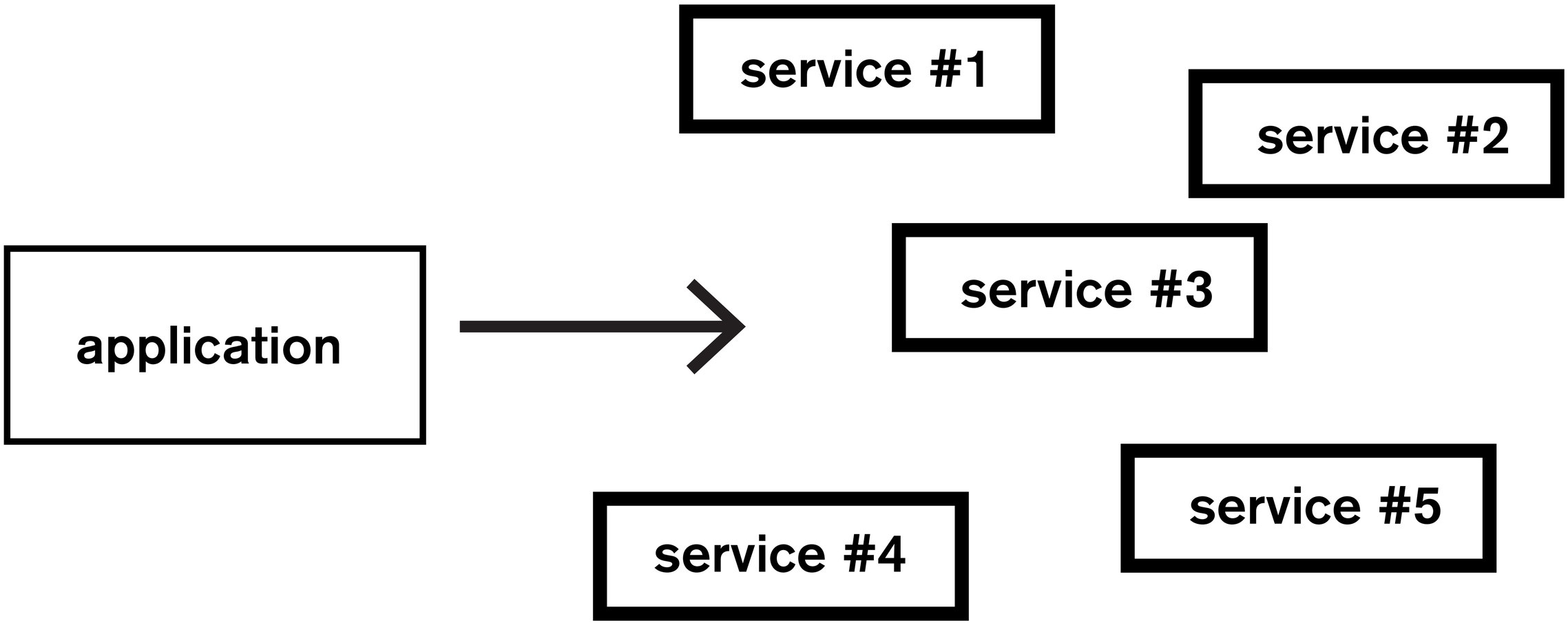
4

Как строить микросервисы?

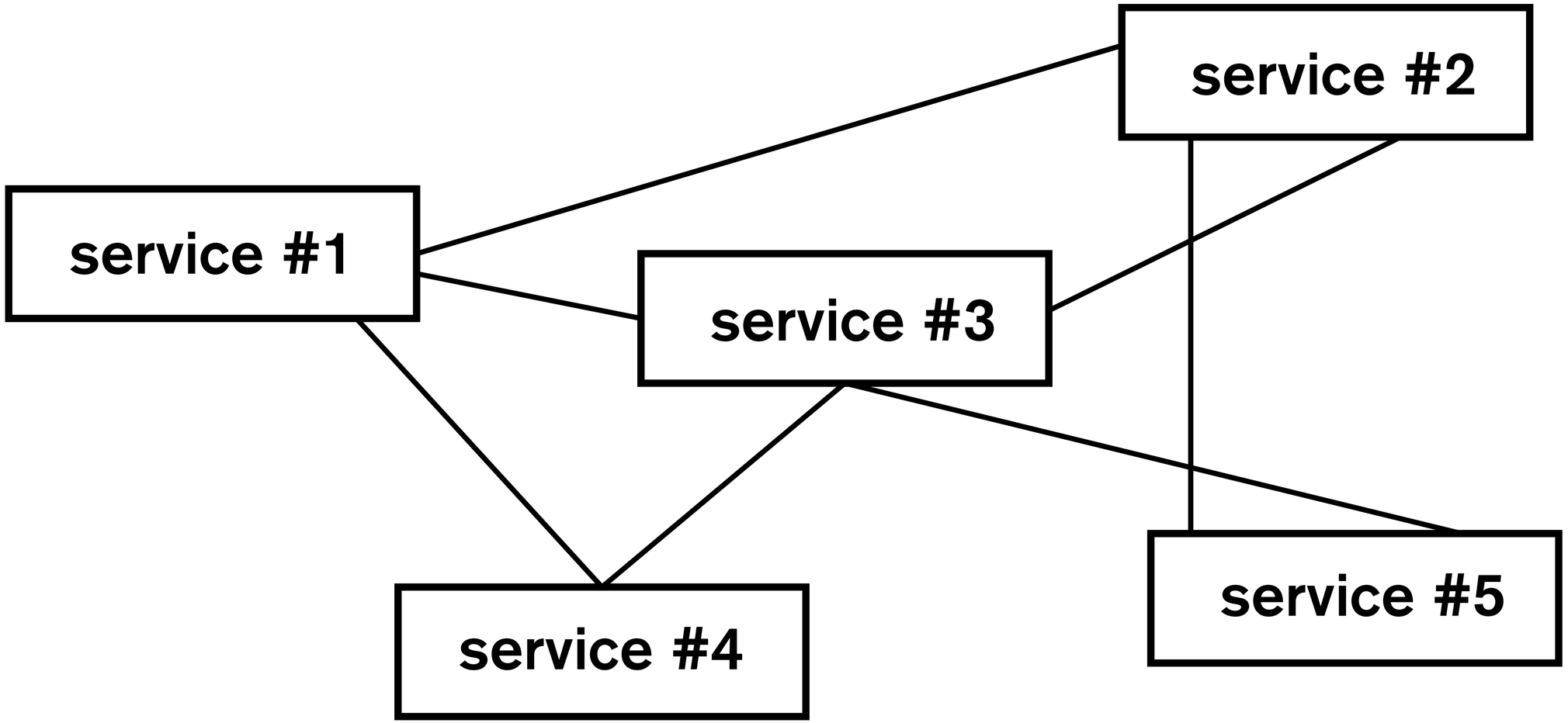
Как строить микросервисы

1. Функциональная декомпозиция
2. Взаимодействие с пользователями системы
3. Взаимодействие между сервисами
4. Обеспечение согласованности данных
5. Service discovery
6. Тестирование
7. Логирование

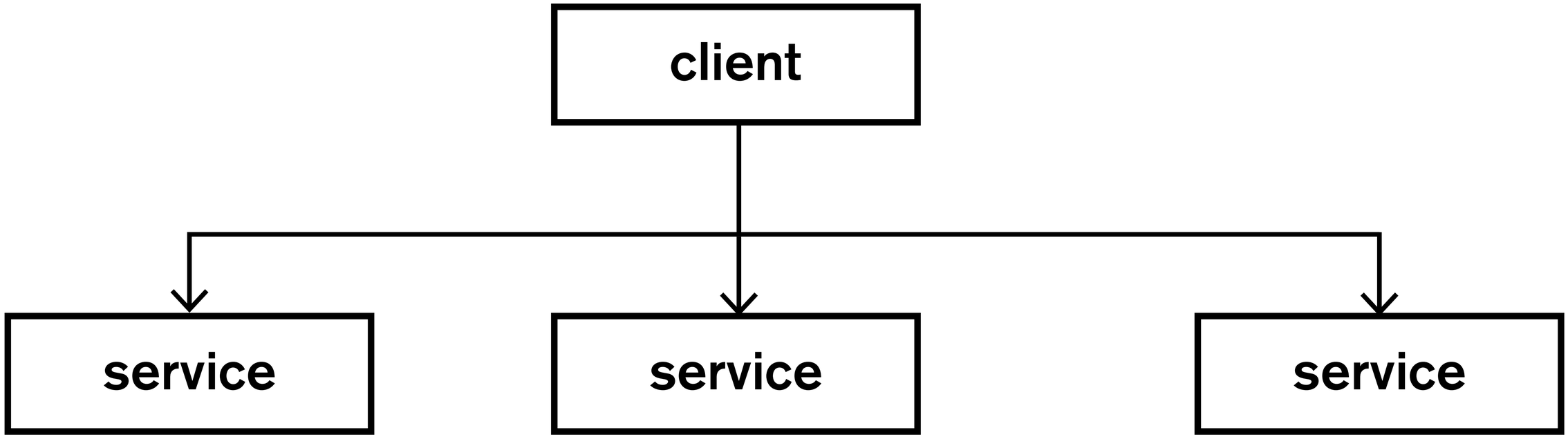
Функциональная декомпозиция



Сильная гранулярность



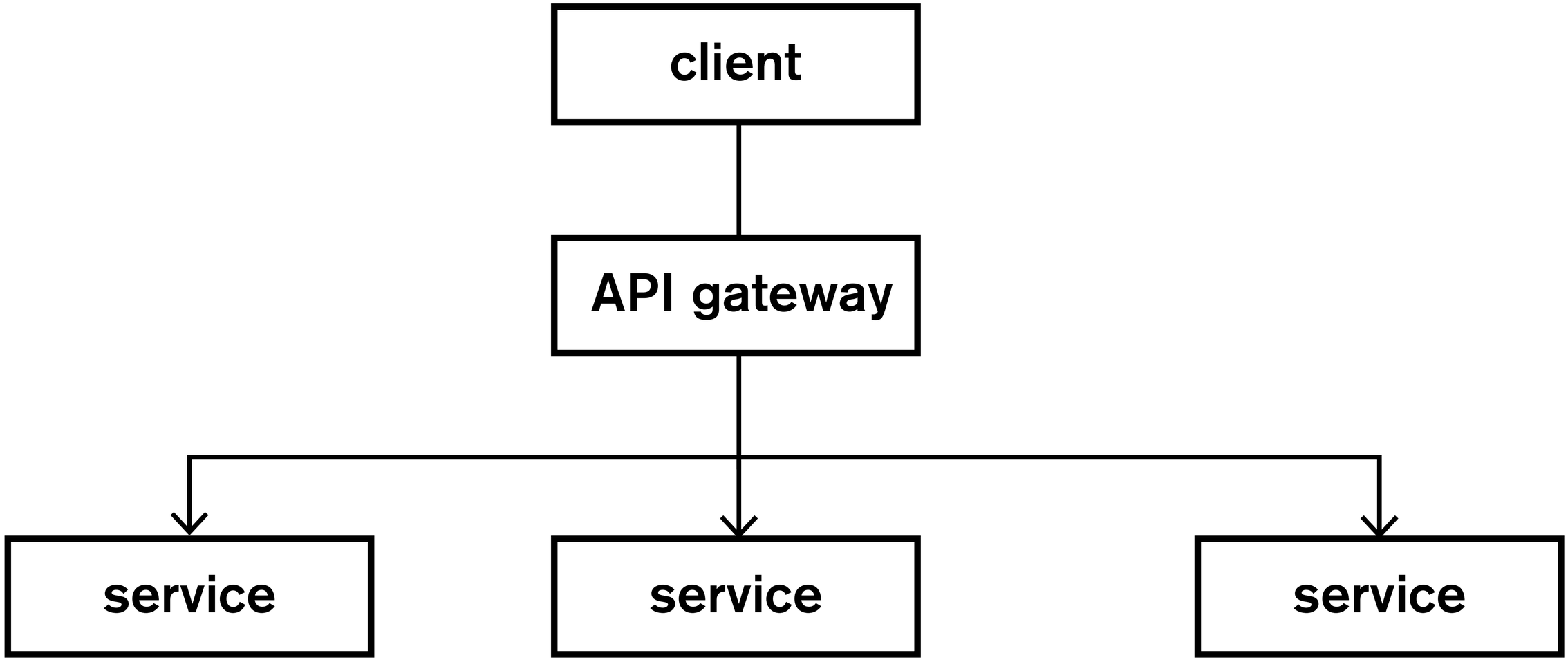
Взаимодействие с пользователями



Минусы прямого взаимодействия

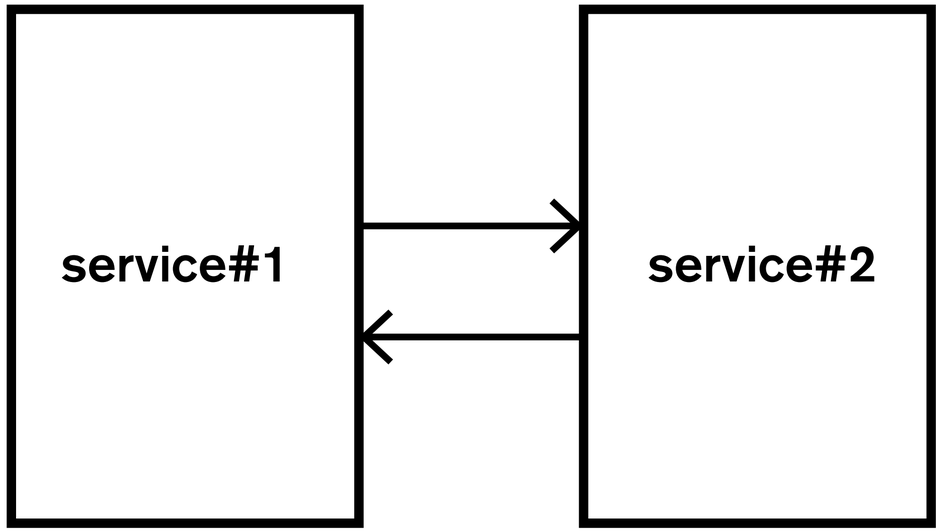
1. Больше сервисов — больше запросов
2. Протоколы могут отличаться от HTTP
3. Затрудняет рефакторинг

API Gateway

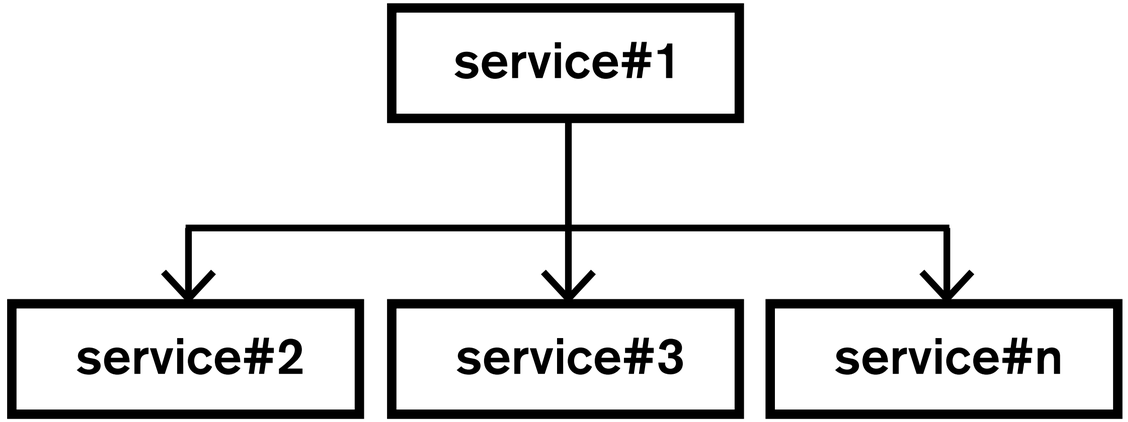


Взаимодействие между сервисами

Синхронные запросы



Асинхронные уведомления



Синхронные запросы

1. HTTP
2. Akka.nete, MS Orleans
3. WCF
4. Protobuf
5. Apache Thrift
6. Linkerd

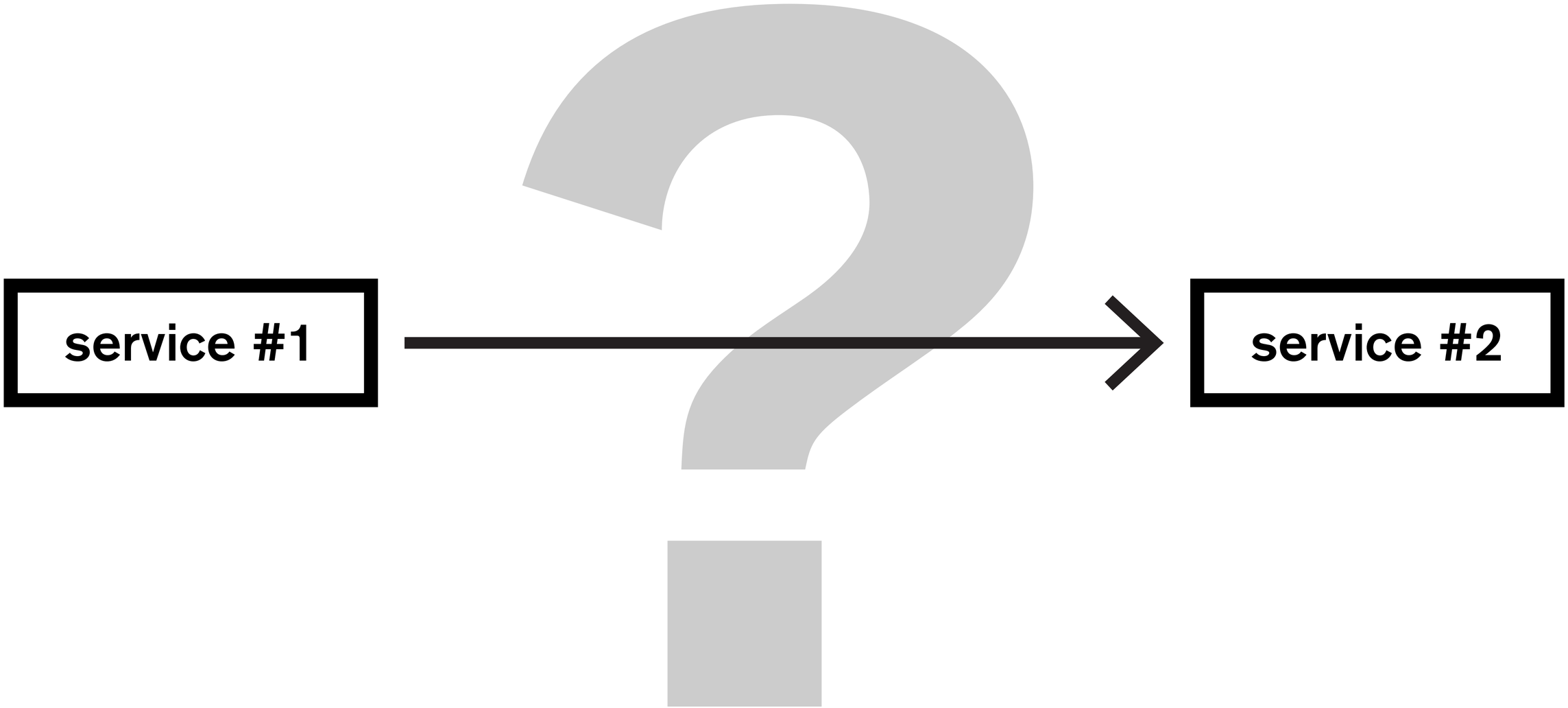
Обеспечение согласованности данных

1. После совершения пользователем операции в одном из сервисов изменились данные и нужно эти изменения отправить в другой сервис.
2. Нам необходимо построить отчет по неким показателям, данные для составления отчета распределены по нескольким микросервисам.

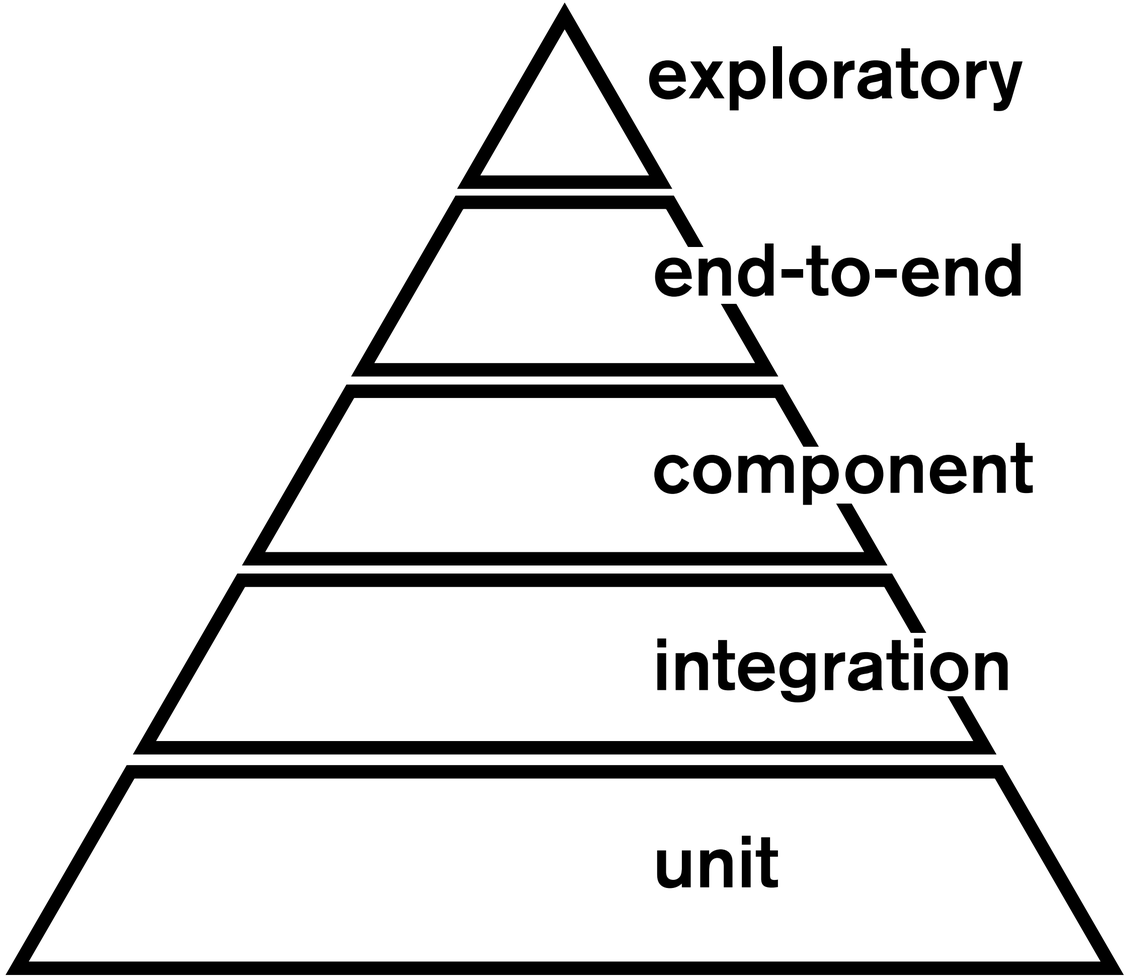
Асинхронные уведомления

1. Apache Kafka
2. RabbitMQ
3. Akka.net, MS Orleans

Service discovery



Тестирование



Логирование

1. Уникальные идентификаторы сервисов
2. Уникальный идентификатор для каждого запроса
3. Использование UTC timestamp
4. Агрегация логов

Распространенные ошибки

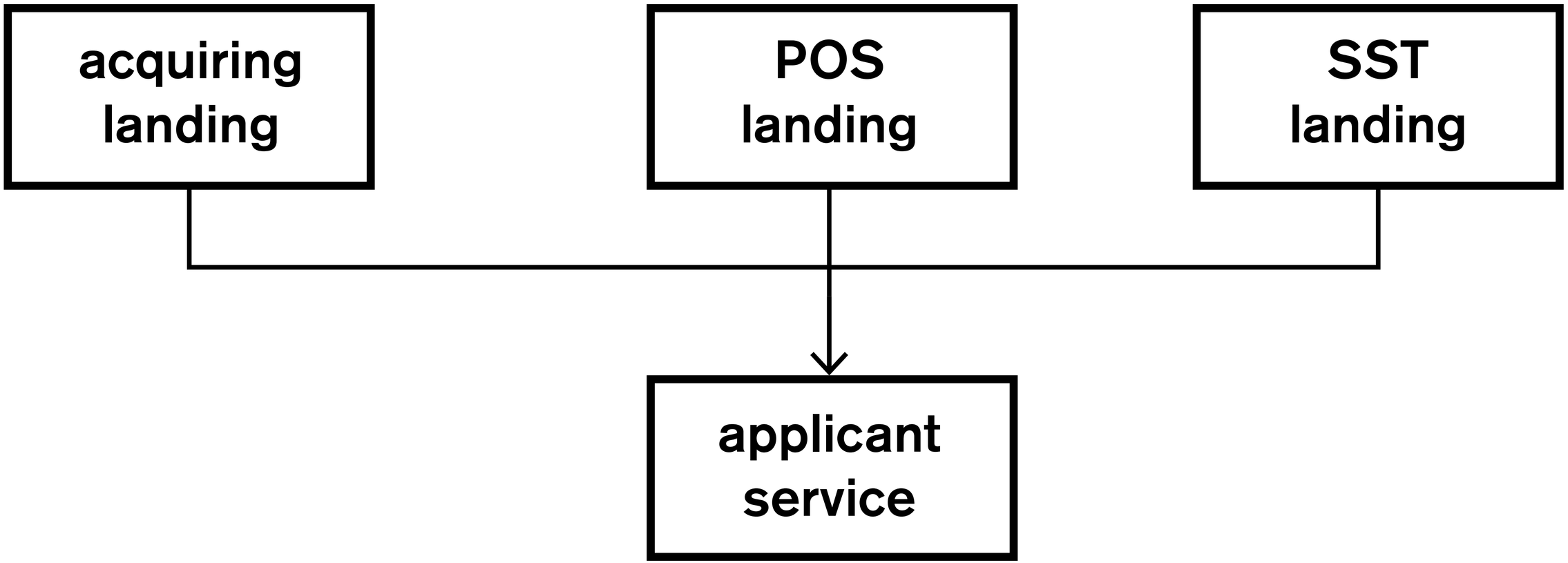
1. Использование таймаутов при запросах
2. Частое переиспользование кода
3. Статичные контракты
4. Недоступность целого из-за отказа части
5. Микролитизация

Пример

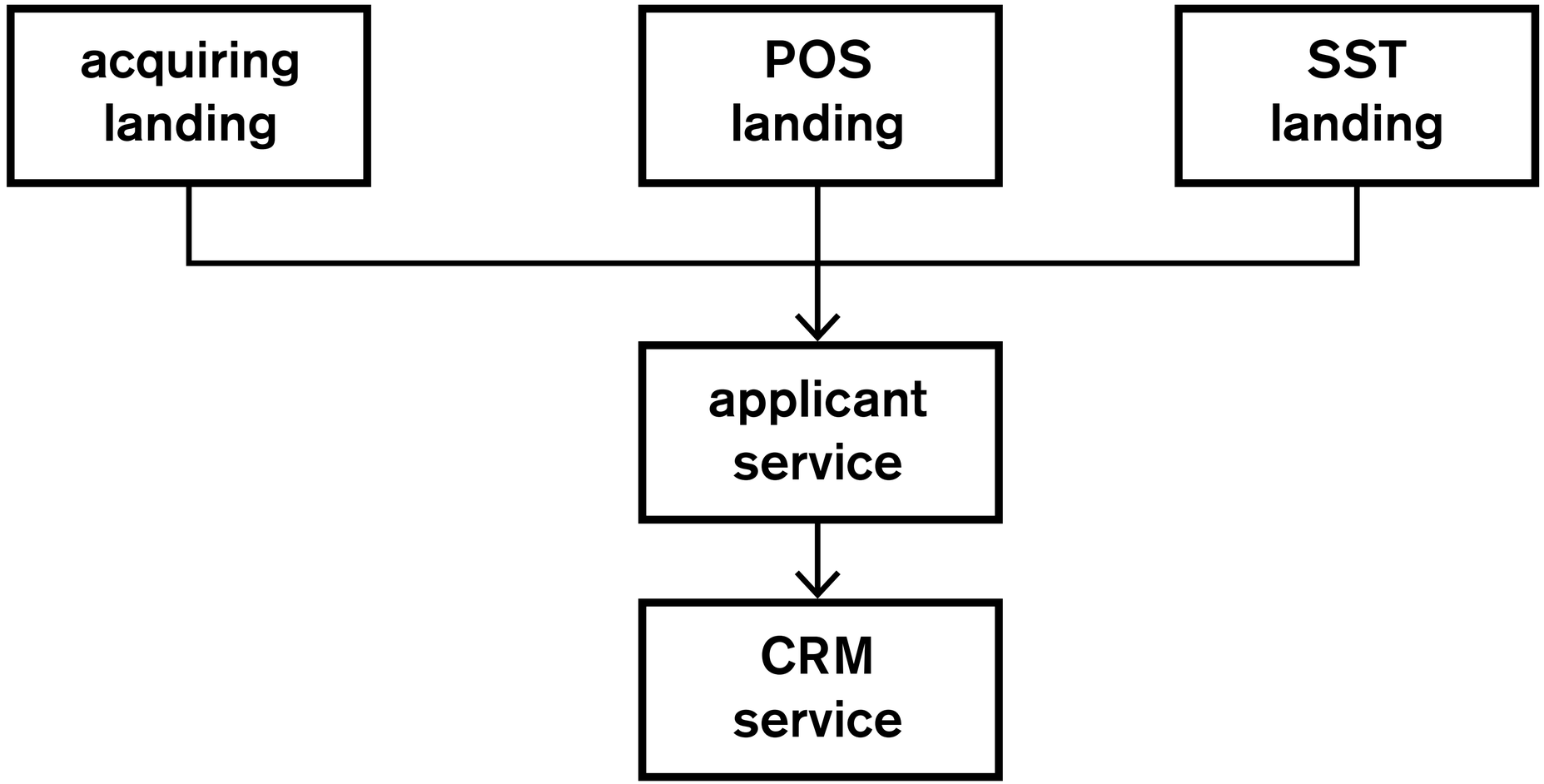
Исходная позиция

интернет-эквайринг	POS в магазинах	терминалы самообслуживания

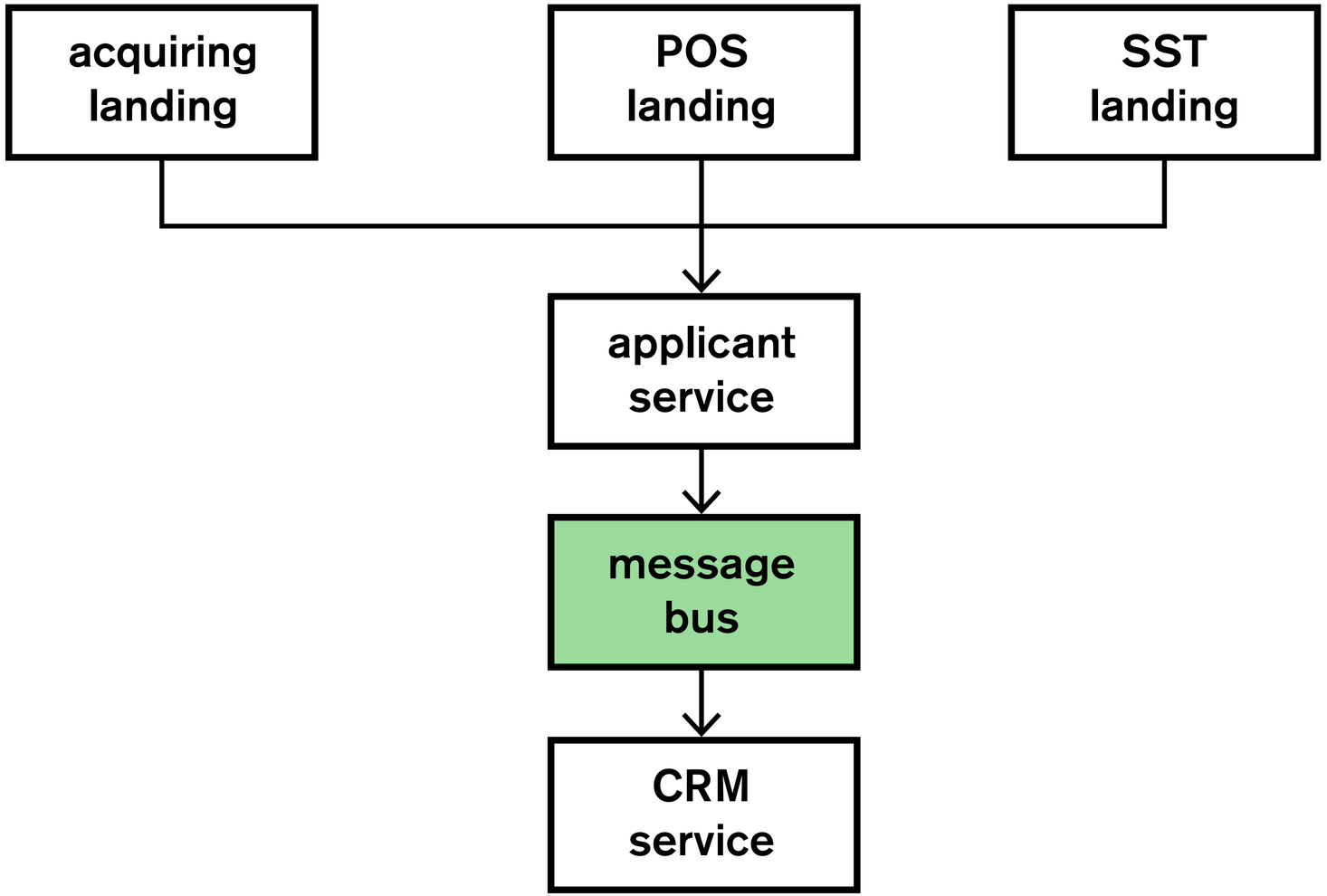
Прием заявок



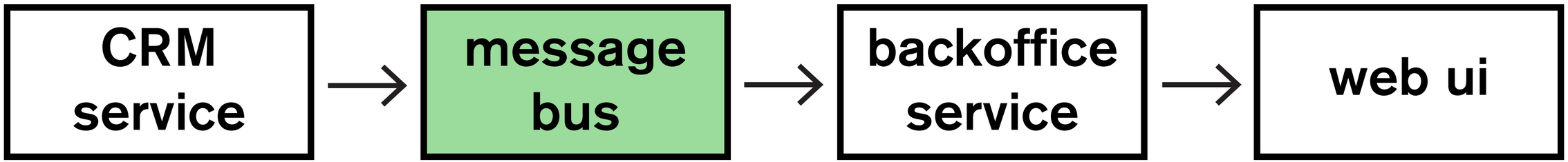
CRM


acquiring landing
POS landing
SST landing
applicant service
CRM service

Интеграция сервисов



Сервис для клиентов



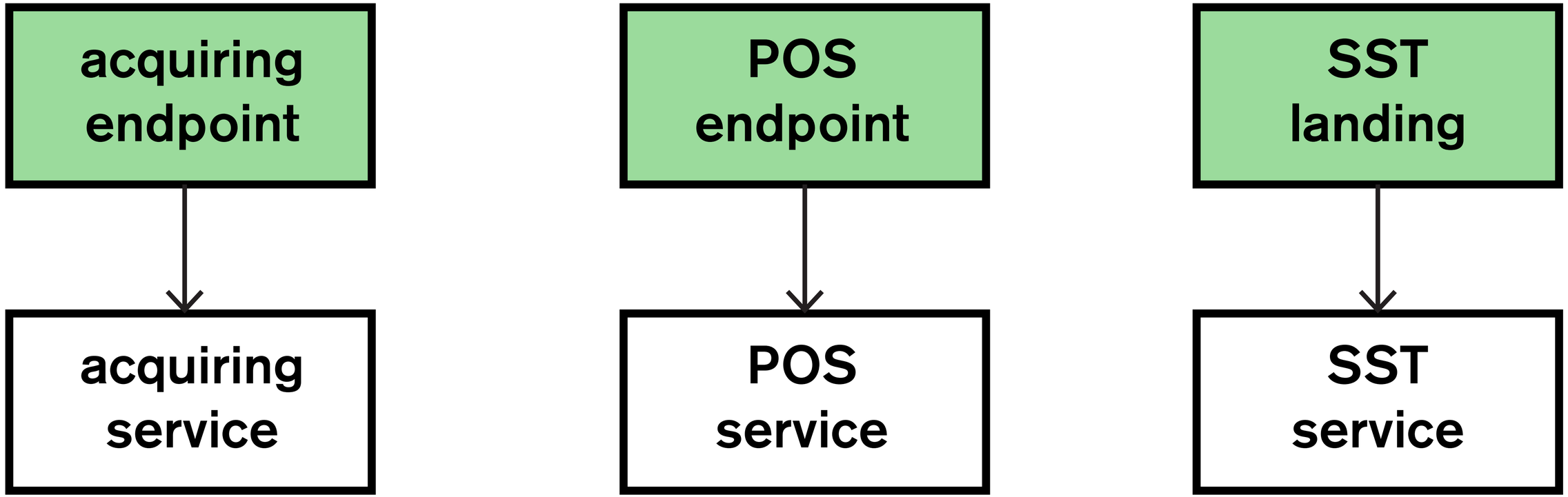
Еще точки входа

acquiring endpoint

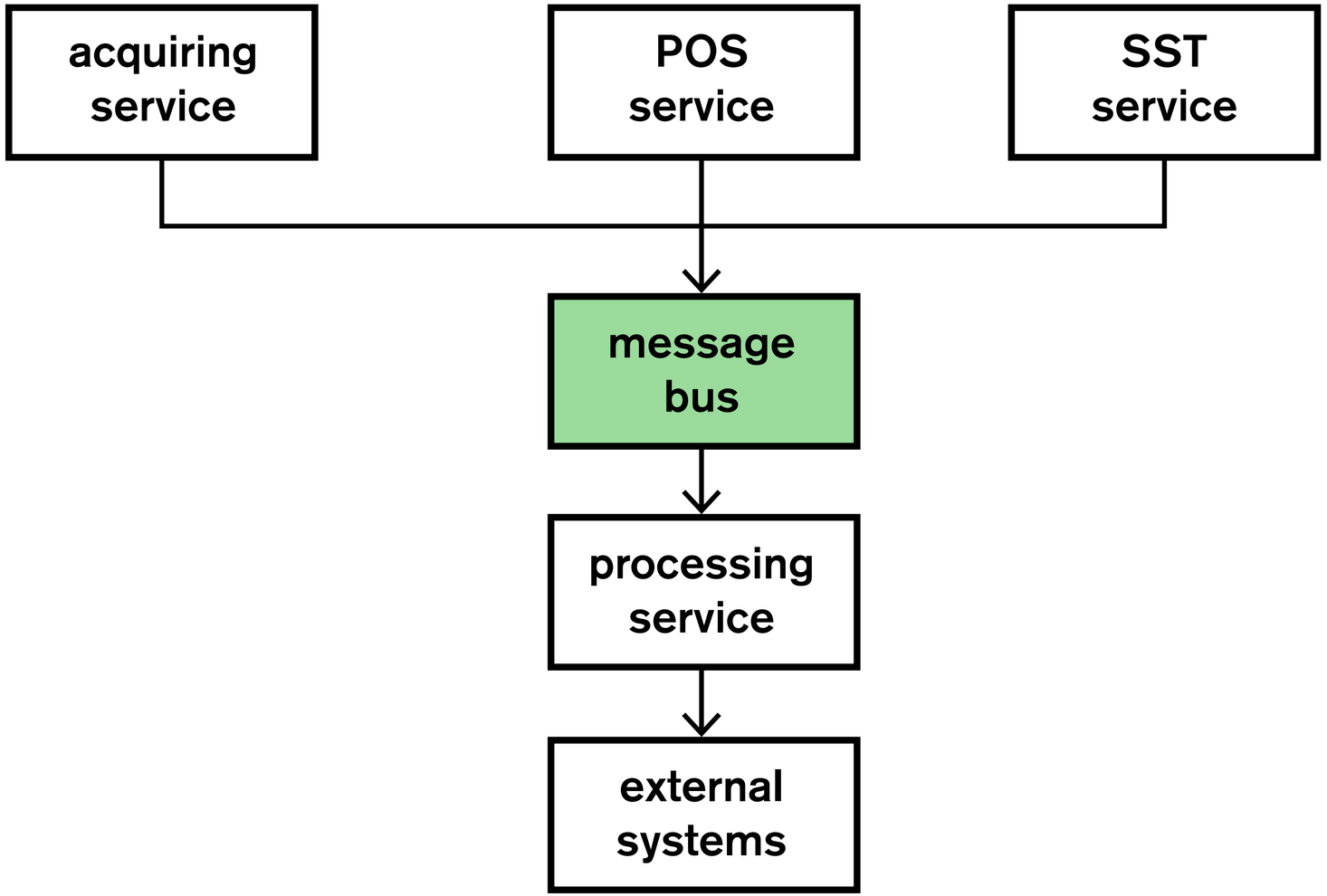
POS endpoint

SST landing

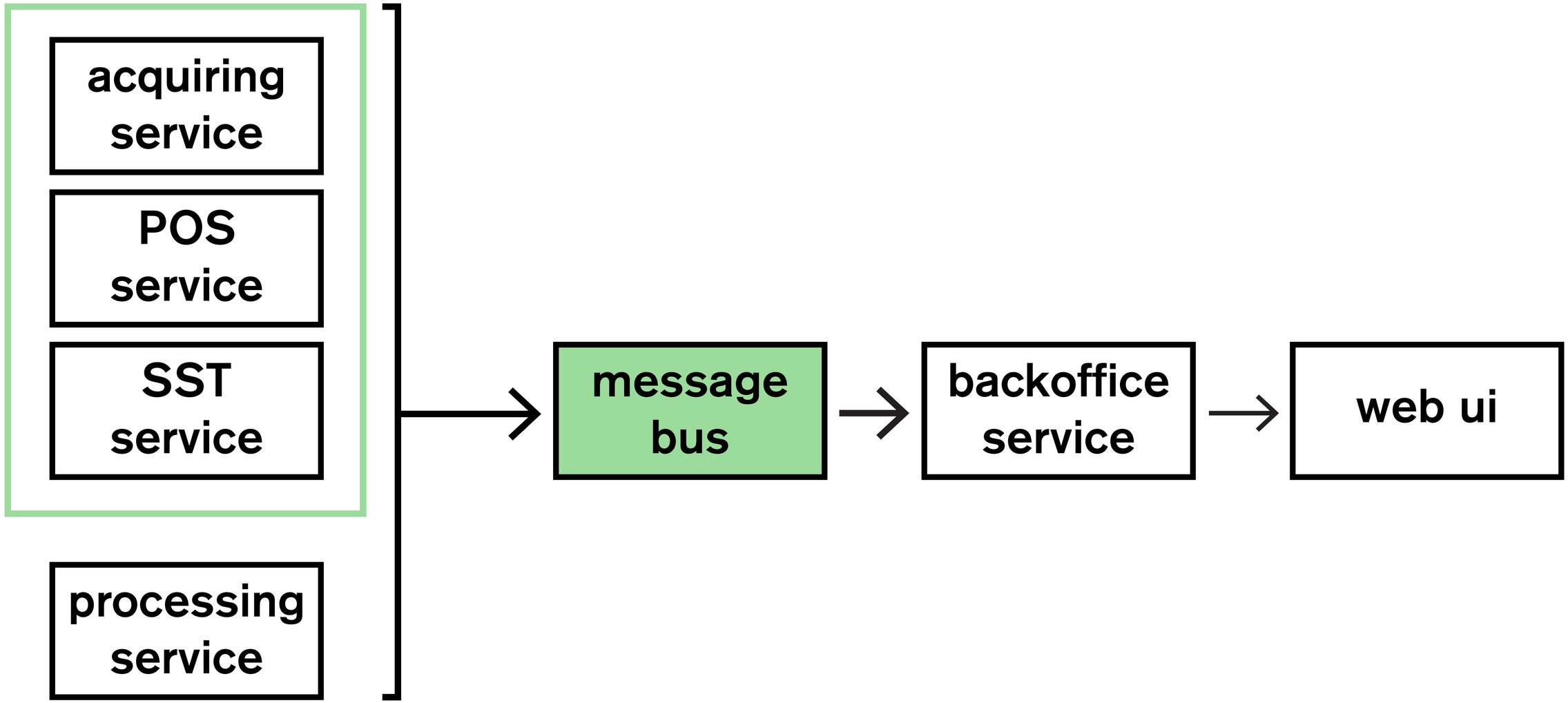
Прием средств и переводов



Связь с процессингом



Связь с клиентом



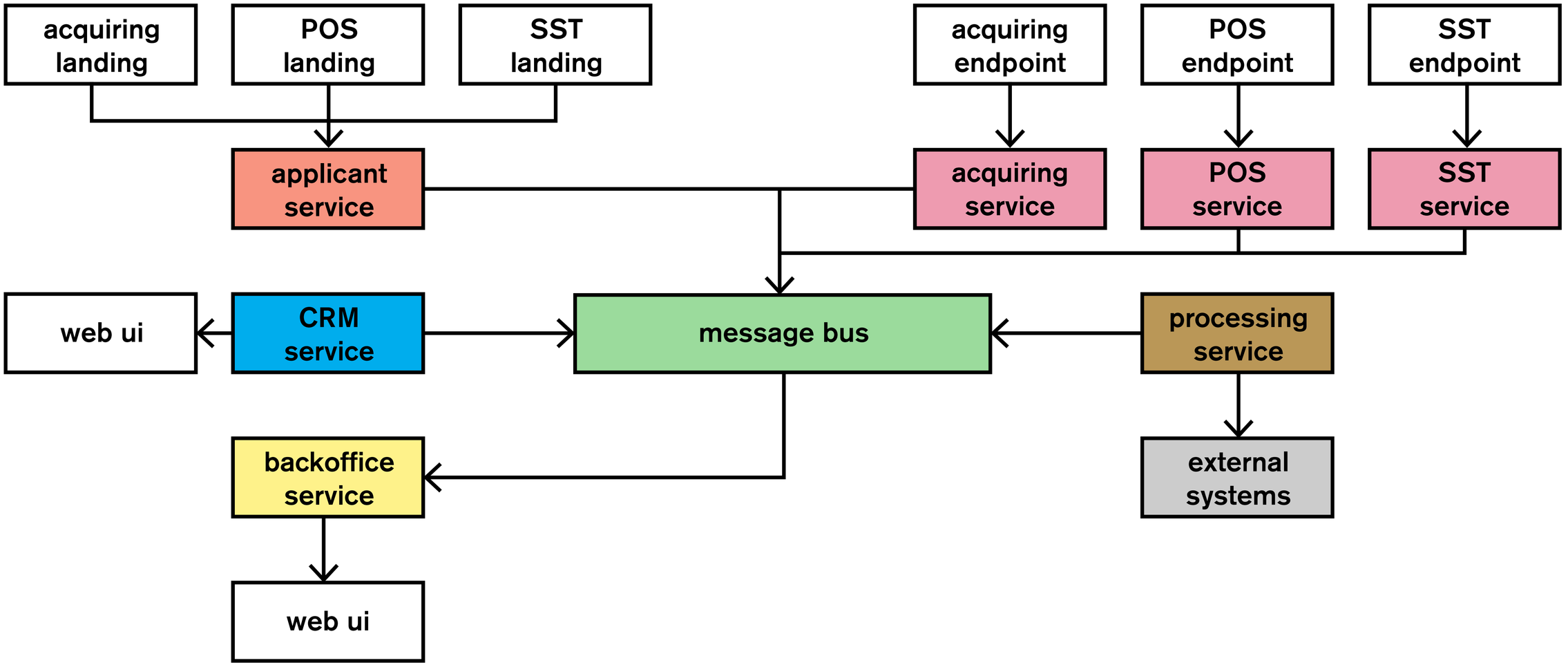
Объединение запросов

Запрос на списание получен.
ID: xxxx-xxxx-xxxx-xxxx

Списание прошло успешно.
ID: xxxx-xxxx-xxxx-xxxx

Средства переведены клиенту.
ID: xxxx-xxxx-xxxx-xxxx

Итоговая архитектура



Что в итоге?

Pros&Cons

+ Распределенная система
+ Гибкая разработка
+ Легко масштабируются
+ Низкий TTM для нового функционала

- Распределенная система
- Сложное проектирование
- Проблемы при тестировании

You shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile

Martin Fowler