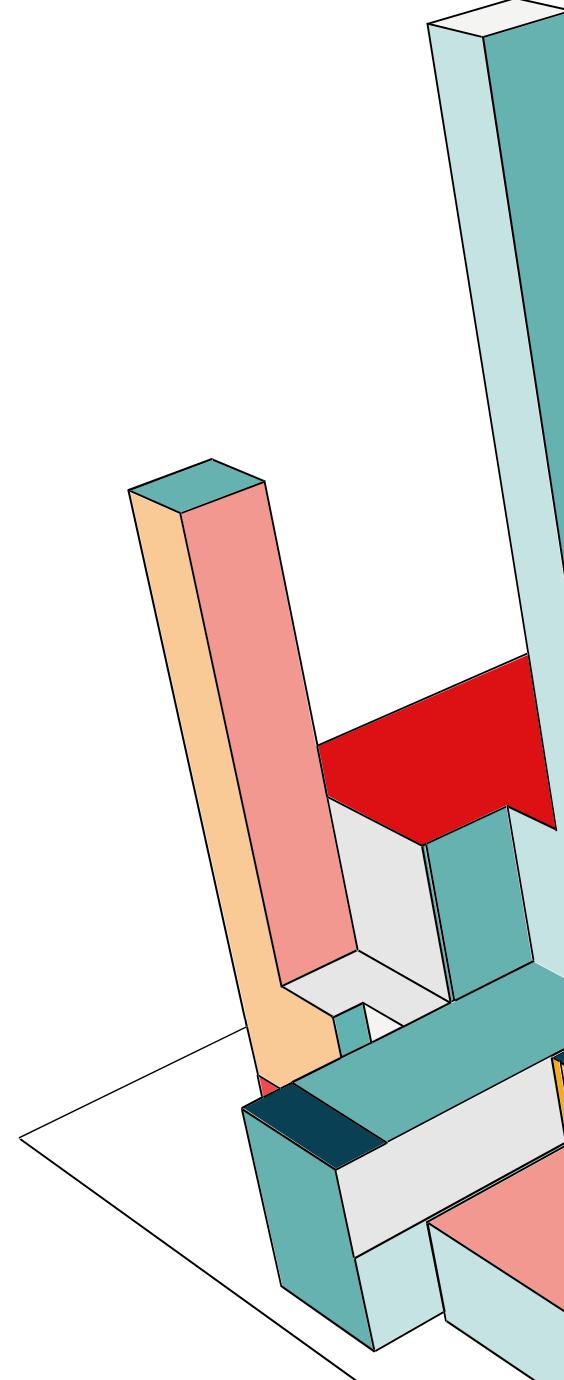
The background features a collection of colorful, 3D-style geometric shapes in shades of red, orange, yellow, teal, and light blue, arranged in a scattered, overlapping manner.

КОНСТРУИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПЛАН ЛЕКЦИИ № 11

Асинхронное межсервисное взаимодействие

1. Архитектурные паттерны межсервисного взаимодействия
2. Message Bus
3. Как правильно выбирать очередь



ТИПЫ МИКРОСЕРВИСНЫХ АРХИТЕКТУР

- Распределенный монолит
- Слоёные микросервисы
- Domain-based микросервисы
- Stateless vs Statefull микросервисы



РАСПРЕДЕЛЁННЫЙ МОНОЛИТ

Распределённый монолит — архитектурный паттерн, в котором **компоненты системы**, хотя и разёрнуты в виде отдельных сервисов, фактически зависят друг от друга настолько, что **не могут развиваться и масштабироваться независимо**

Некоторые признаки распределённого монолита:

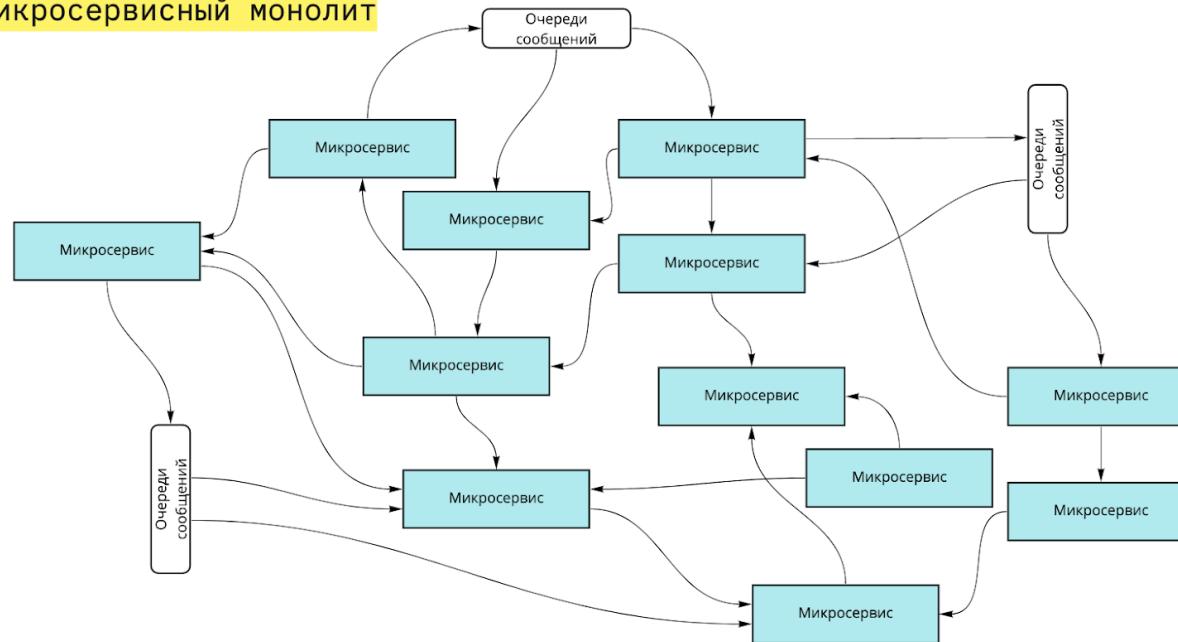
1. Высокая связанность между сервисами.
2. Глобальные транзакции
3. Отсутствие автономности сервисов.
4. Общие схемы данных.



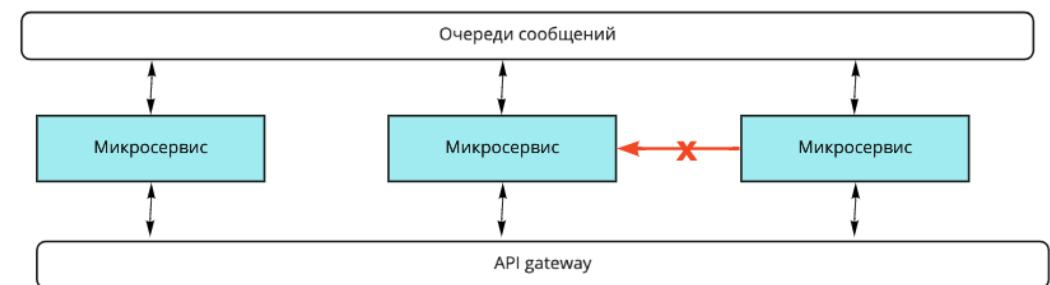
Монолитное приложение



Микросервисный монолит



Микросервисы



«СЛОЁНЫЕ» МИКРОСЕРВИСЫ

Микросервисы организуются в домены и слои, аналогично принципам слоистых архитектур вроде Clean или Onion Architecture.

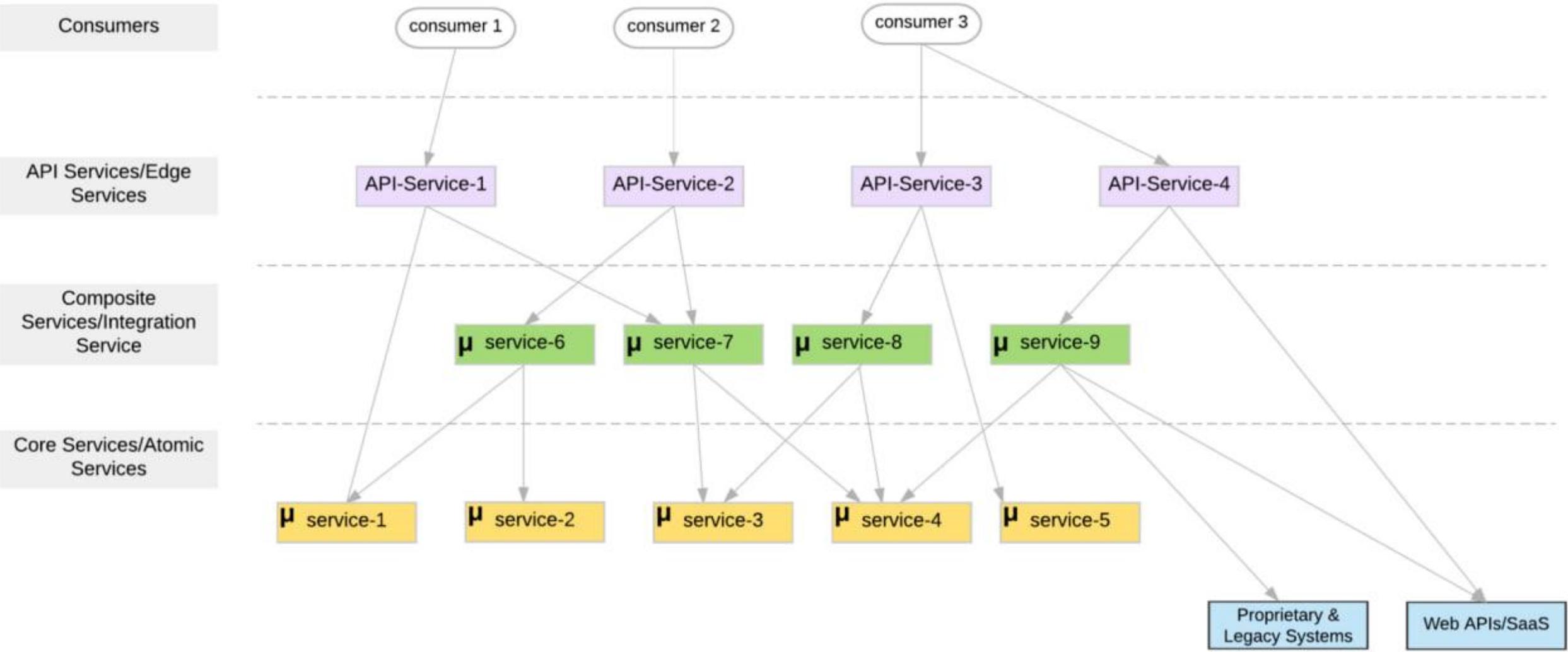
Организует код в виде **строго разделенных и изолированных друг от друга модулей (слоев)**, которые остаются внутри единого приложения:

- Слой domain API
- Слой business API
- Слой backend for fronted
- Слой клиентского web-app

Uber классифицировал 2200 сервисов в 70 доменов и сократил onboarding на 25-50%.

Подход эволюционировал из монолита (2012) для масштаба на тысячи инженеров





FEATURE-BASED МИКРОСЕРВИСЫ

Подход к созданию приложений, при котором **система разделена на независимые микросервисы, каждый из которых отвечает за конкретную функцию.**

Каждый микросервис работает автономно, в отдельном процессе. Это значит, что сбой в одном сервисе не влияет на работу всей системы и снижает риск глобального отказа.

Примеры:

- микросервис для аутентификации и авторизации пользователей;
- микросервис для обработки платежей;
- микросервис для управления содержимым сайта.



DOMAIN-BASED МИКРОСЕРВИСЫ

Архитектурный подход, в котором границы сервисов определяются бизнес-доменами и их задачами, а не техническими слоями или командами разработки.

- **Масштабируемость по бизнес-функциям**
Можно независимо развивать, масштабировать и обновлять отдельные части системы.
- **Прокачка доменной экспертизы**
Команды сосредоточены на конкретном бизнес-контексте, глубже понимают задачи и процессы.
- **Гибкость и устойчивость**
Ошибки и изменения в одном домене не влияют напрямую на другие.

Фокус на DDD

Проведение границ сервисов основано на понятиях Domain-Driven Design:

Bounded Context — основная единица декомпозиции

Ubiquitous Language — единое словарь внутри домена



STATELESS VS STATEFULL МИКРОСЕРВИСЫ

Stateless микросервисы не сохраняют состояние между запросами: каждый запрос содержит всю необходимую информацию и обрабатывается независимо. Это обеспечивает простое горизонтальное масштабирование, так как любой экземпляр сервиса может обработать любой запрос без синхронизации состояния.

Stateful микросервисы хранят состояние (например, сессии пользователей или данные корзины) на сервере между запросами. Это упрощает логику для персонализированных взаимодействий, но усложняет масштабирование из-за необходимости синхронизации состояния между экземплярами.

Аспект	Stateless	Stateful
Масштабируемость	Простая, любой инстанс справится	Сложная, нужна синхронизация
Отказоустойчивость	Высокая, сбой не влияет на другие	Низкая, риск потери состояния
Сложность разработки	Низкая	Высокая из-за управления состоянием



ПОДХОДЫ К ВЗАИМОДЕЙСТВИЮ МИКРОСЕРВИСОВ

Обмен файлами

- Такой подход применяется для передачи больших данных или когда требуется временное хранение.

Общая база данных

- Использование одной общей базы данных для нескольких микросервисов считается антипаттерном, каждый микросервис должен владеть своей отдельной базой

Синхронные вызовы

- При синхронных вызовах микросервис отправляет запрос другому сервису и ждет ответа до продолжения работы. REST API или gRPC.

Очередь сообщений

- Асинхронный подход с использованием **очередей сообщений** позволяет микросервисам обмениваться данными и событиями без ожидания ответа. Очереди служат буфером для хранения сообщений. В очередях реализуются методы pull и push, часто применяется модель издаватель-подписчик (Pub/Sub).

Общая база данных нежелательна, синхронные вызовы применимы для простых и быстрых операций, очереди сообщений — для асинхронного, устойчивого взаимодействия, а обмен файлами — для передачи больших или даже неструктурированных данных во внешних интеграциях.



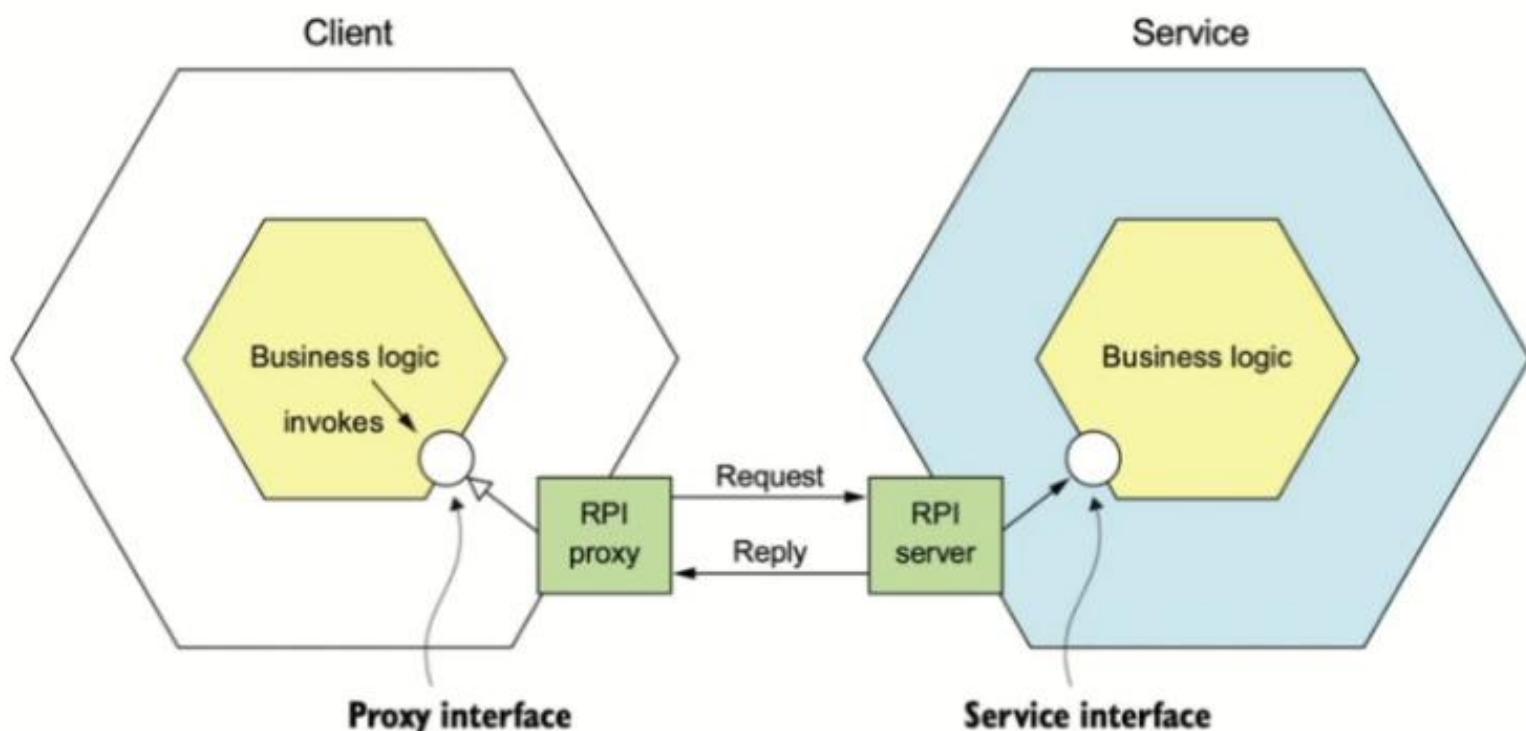
Request-reply протоколы взаимодействия

Request-reply – протоколы, в которых клиент посылает запрос, сервис обрабатывает и отправляет ответ

Примеры:

HTTP: REST, SOAP, graphQL

gRPC

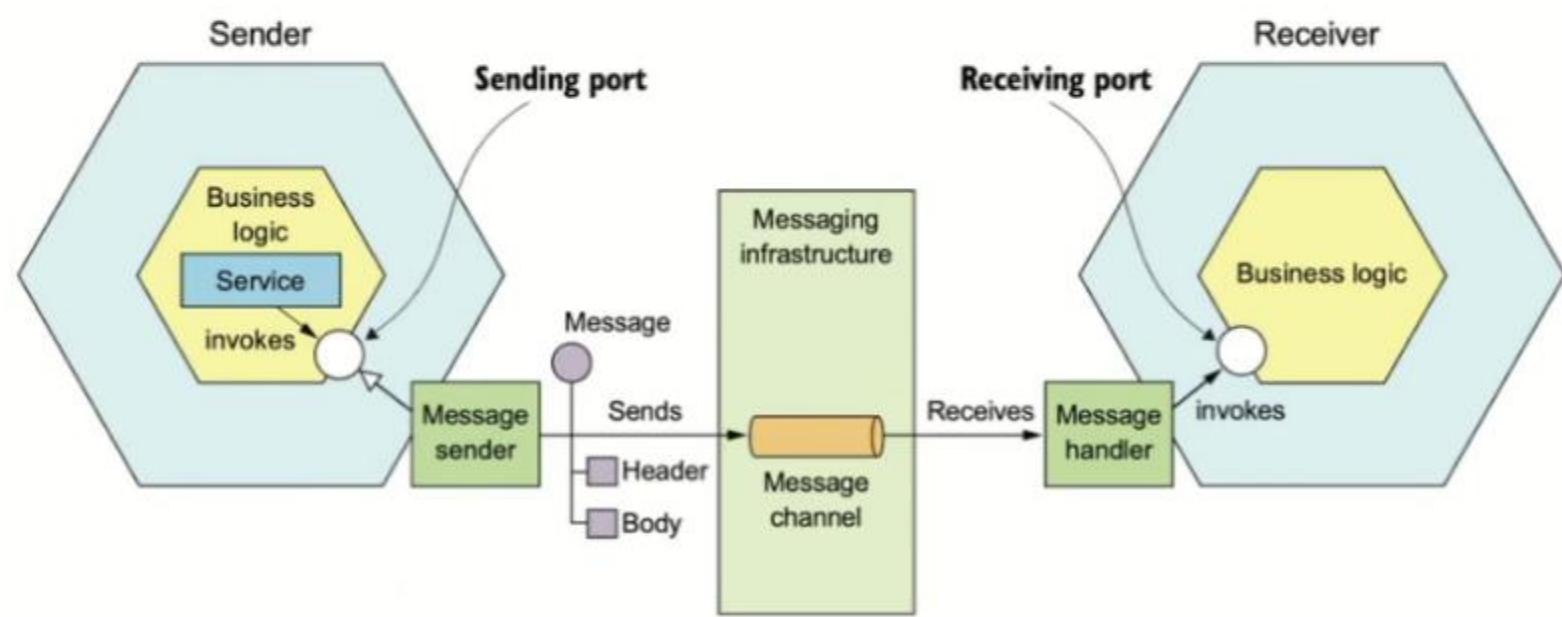


Message-based протоколы взаимодействия

Message-based - протоколы, в которых клиент отправляет сообщение через брокер сообщений, а сервис сообщение читает из брокера сообщений

Примеры:

Amqp, Kafka, Mqtt, ZeroMQ и т.д.



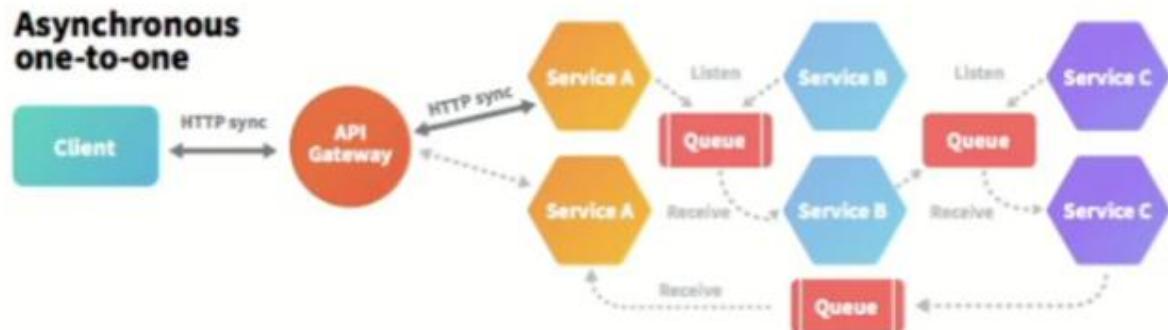
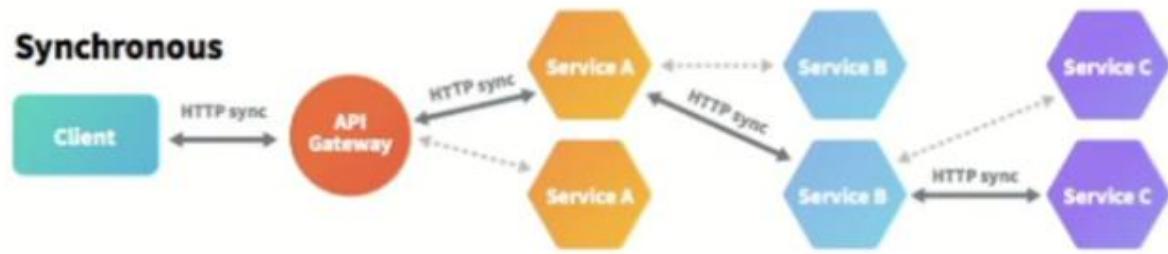
Request-response vs message based

Request-response:

- Проще дебаг
- Latency ниже

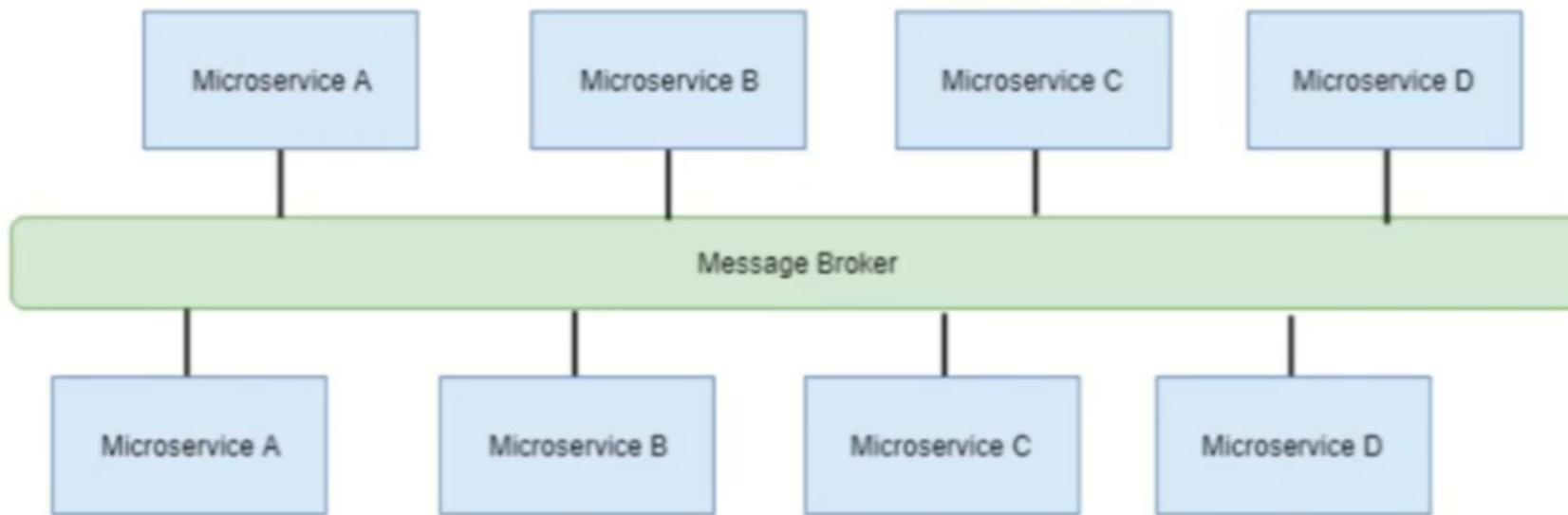
Message based

- Дебаг сложнее
- Latency за счет посредников выше
- Разделение сервисов
- Роутинг
- Массовая отправка (kafka)



Message Bus

А почему бы не организовать асинхронное взаимодействие всех сервисов через очередь?



Плюсы Message Bus

Особенности такого решения:

- Service discovery - сервисам не нужно знать где и кто находится, достаточно просто бросить сообщение в очередь
- Выше доступность – если сервис упал, то он просто не берет запросы из очереди
- Ниже вероятность каскадных падений
- Горизонтальное масштабирование до некоторых пределов
- Увеличивает эффективное использование ресурсов сервисов, написанных на языках не поддерживающих асинхронное программи



Минусы Message Bus

Есть и недостатки:

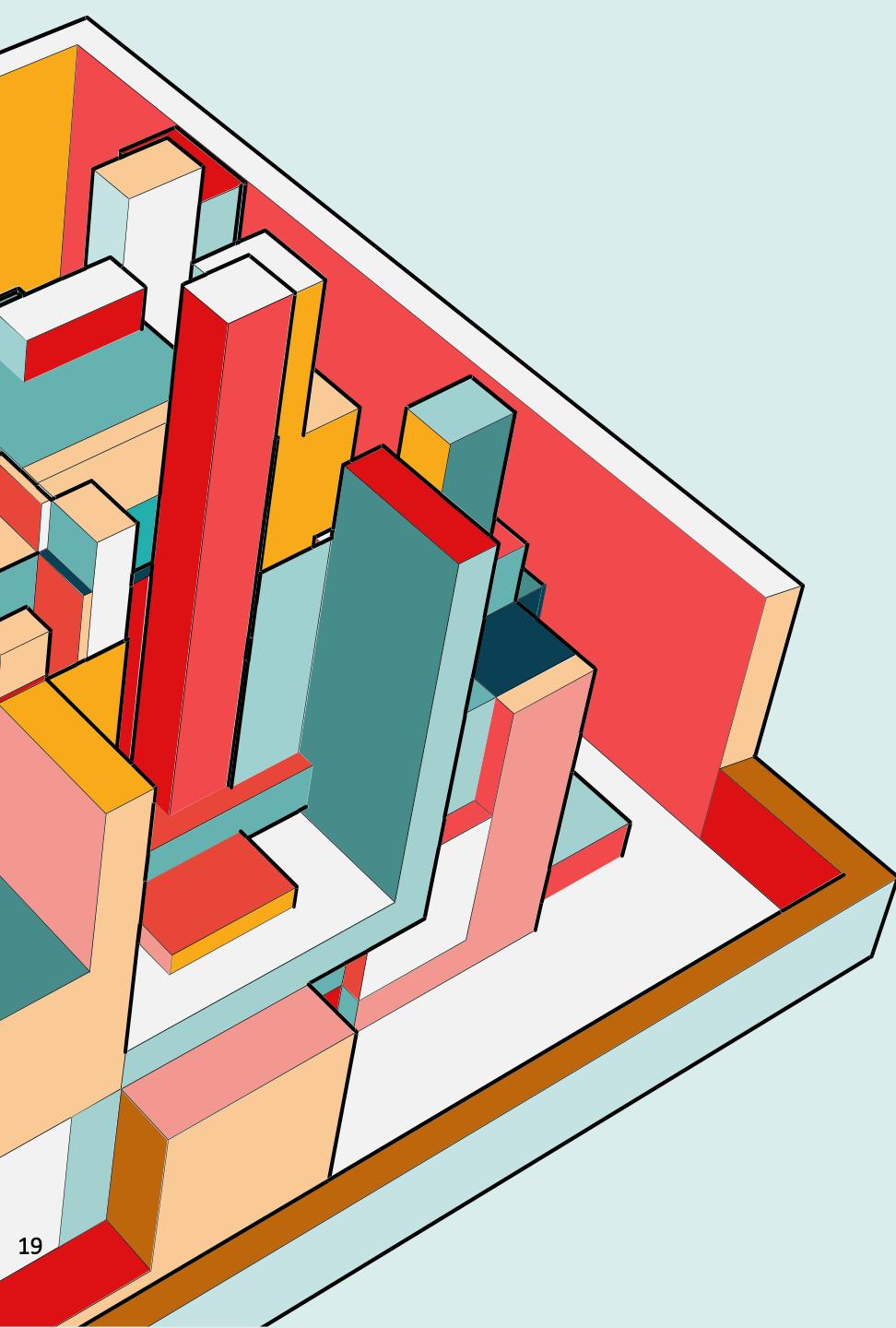
- latency увеличивается
- единая точка отказа
- отладка и обнаружение проблем становится значительно сложнее



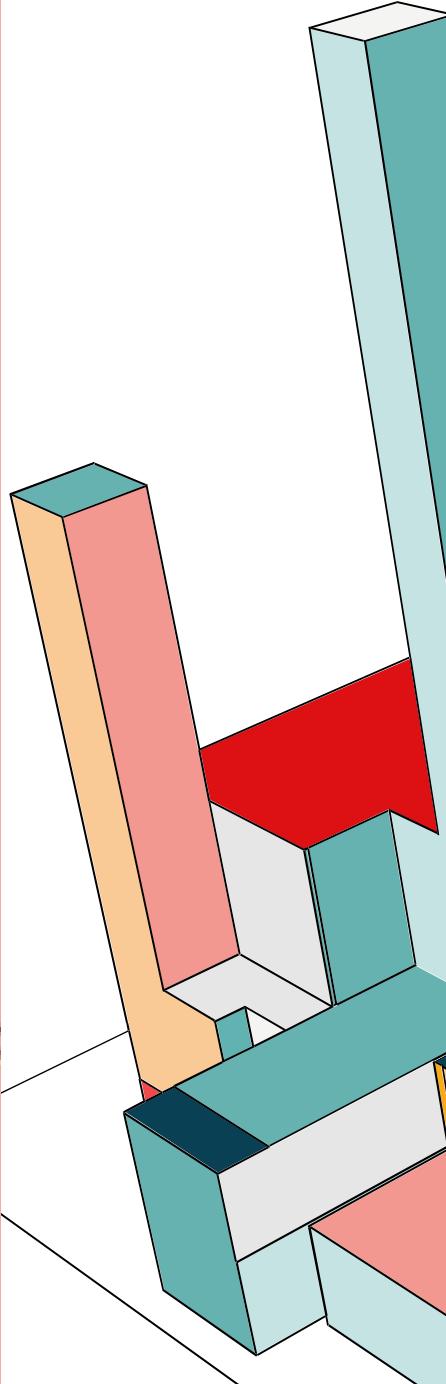
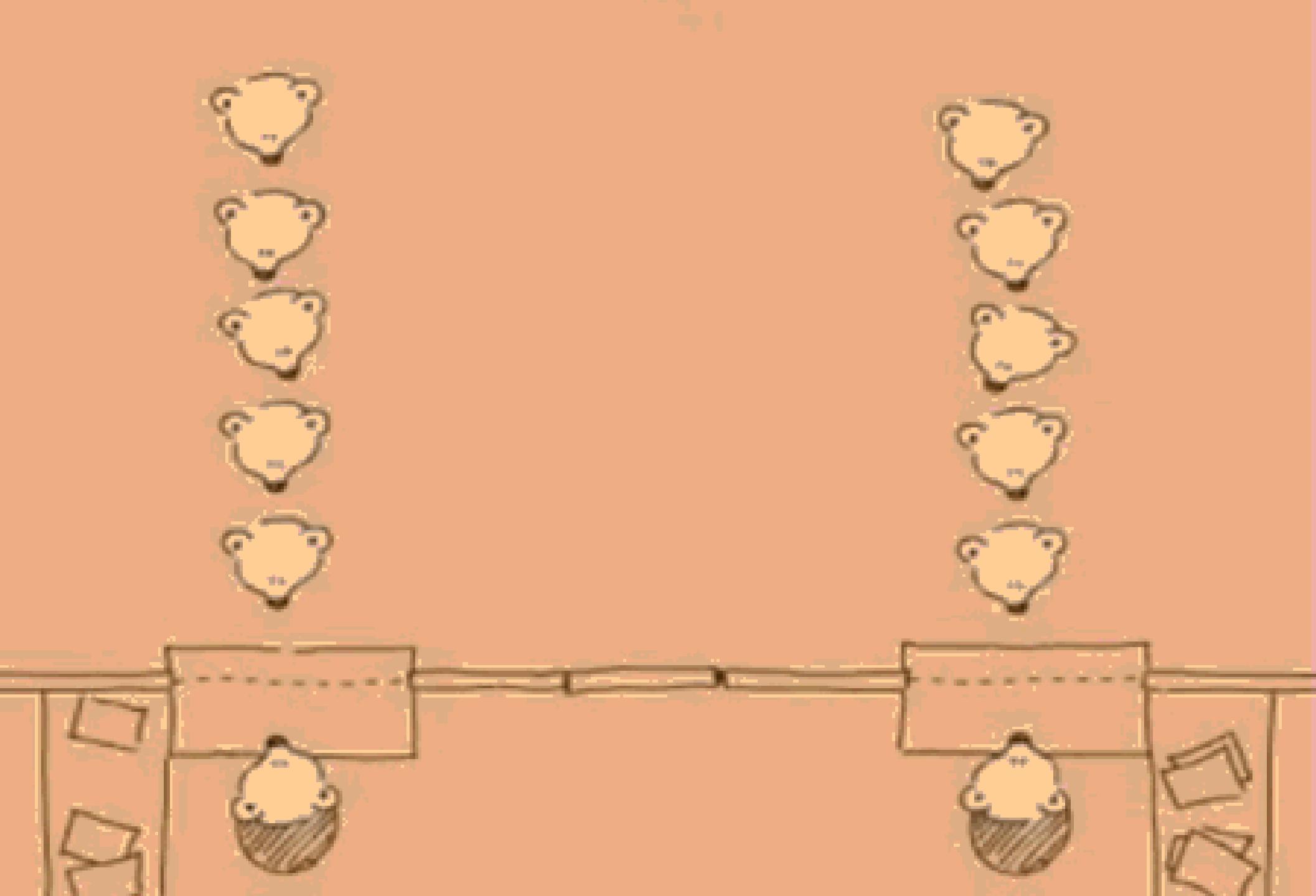
Особенности Message Bus

- В микросервисной архитектуре довольно большая часть взаимодействий все-равно будет синхронной – например, когда нам нужно получить какие-то данные от какого-то сервиса.
- Для синхронного взаимодействия крайне неудобно использование асинхронного протокола (пробрасывание correlation-id и т.д и т.п)
- Оркестратора(типа Kubernetes-а) закрывает потребность в service discovery, горизонтальном масштабировании, доступности: retry-ями и circuit-breaker-ами
- Поэтому в чистом виде message bus в современном мире встречается редко.





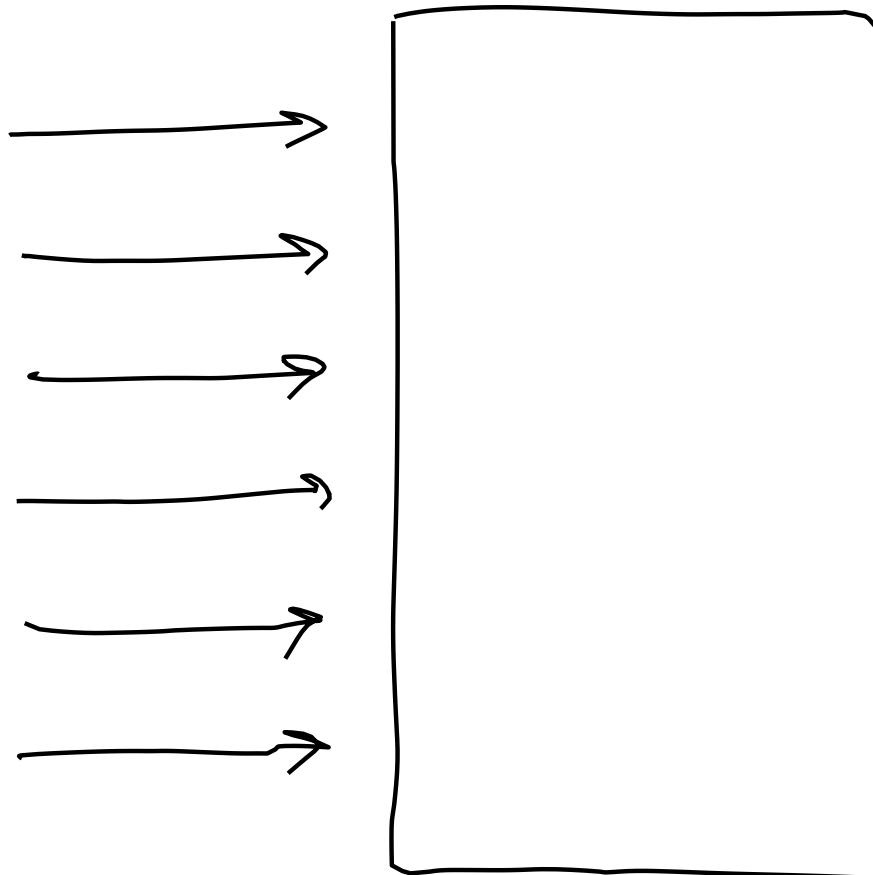
КАК ПРАВИЛЬНО ВЫБИРАТЬ ОЧЕРЕДЬ



ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач

РАСПРЕДЕЛЕНИЕ ЗАДАЧ

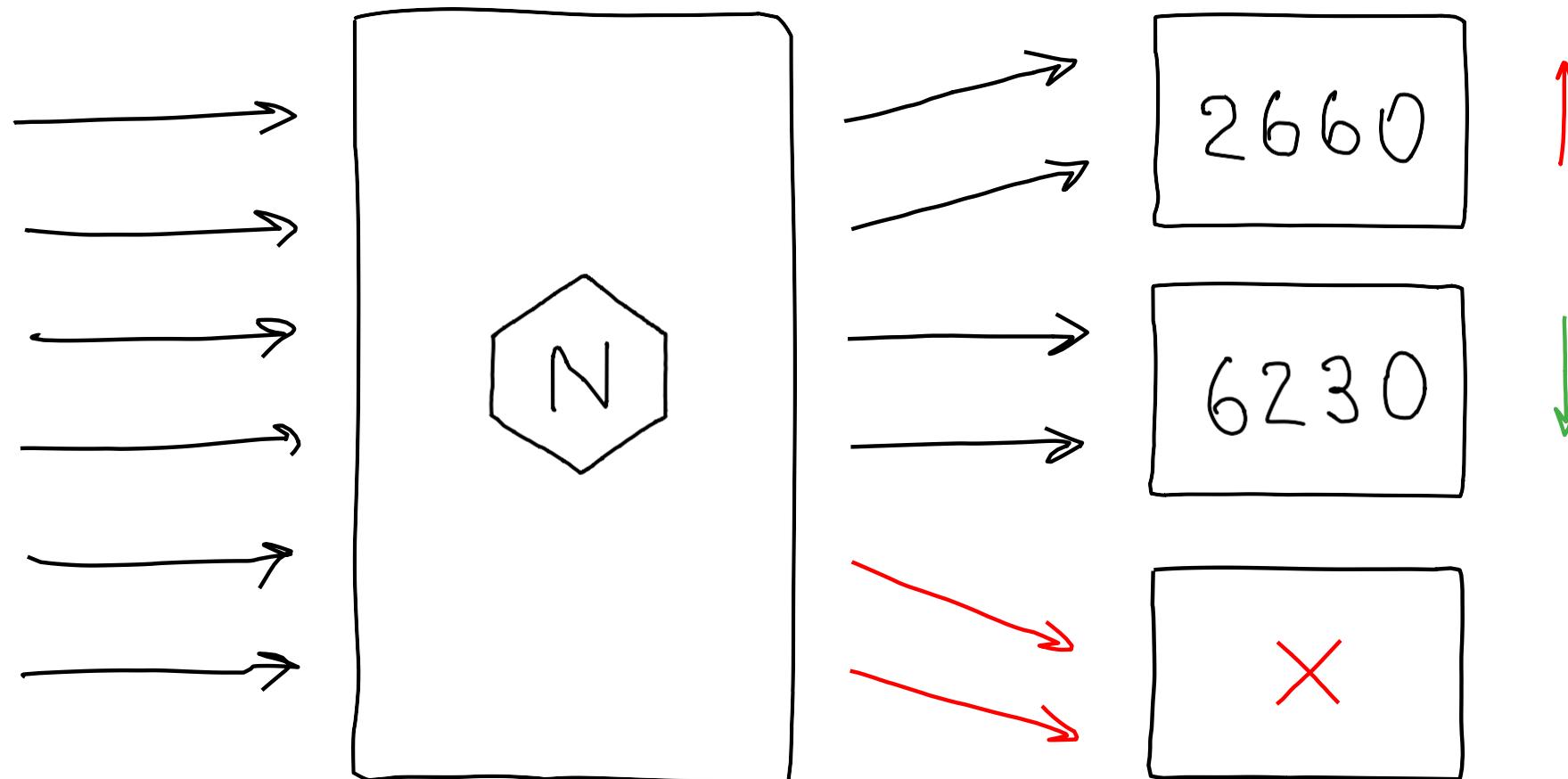


2660

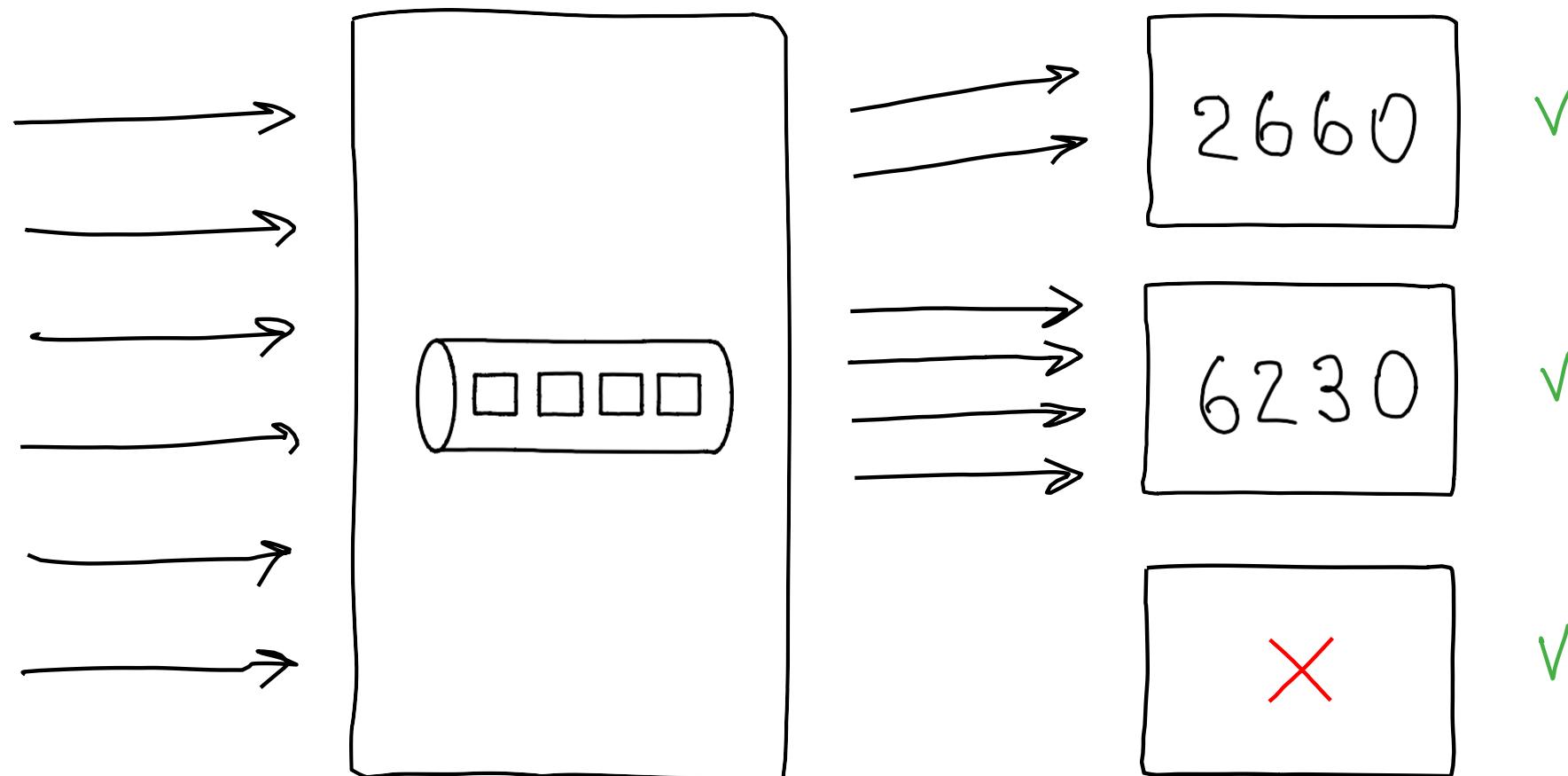
6230

✗

РАСПРЕДЕЛЕНИЕ ЗАДАЧ



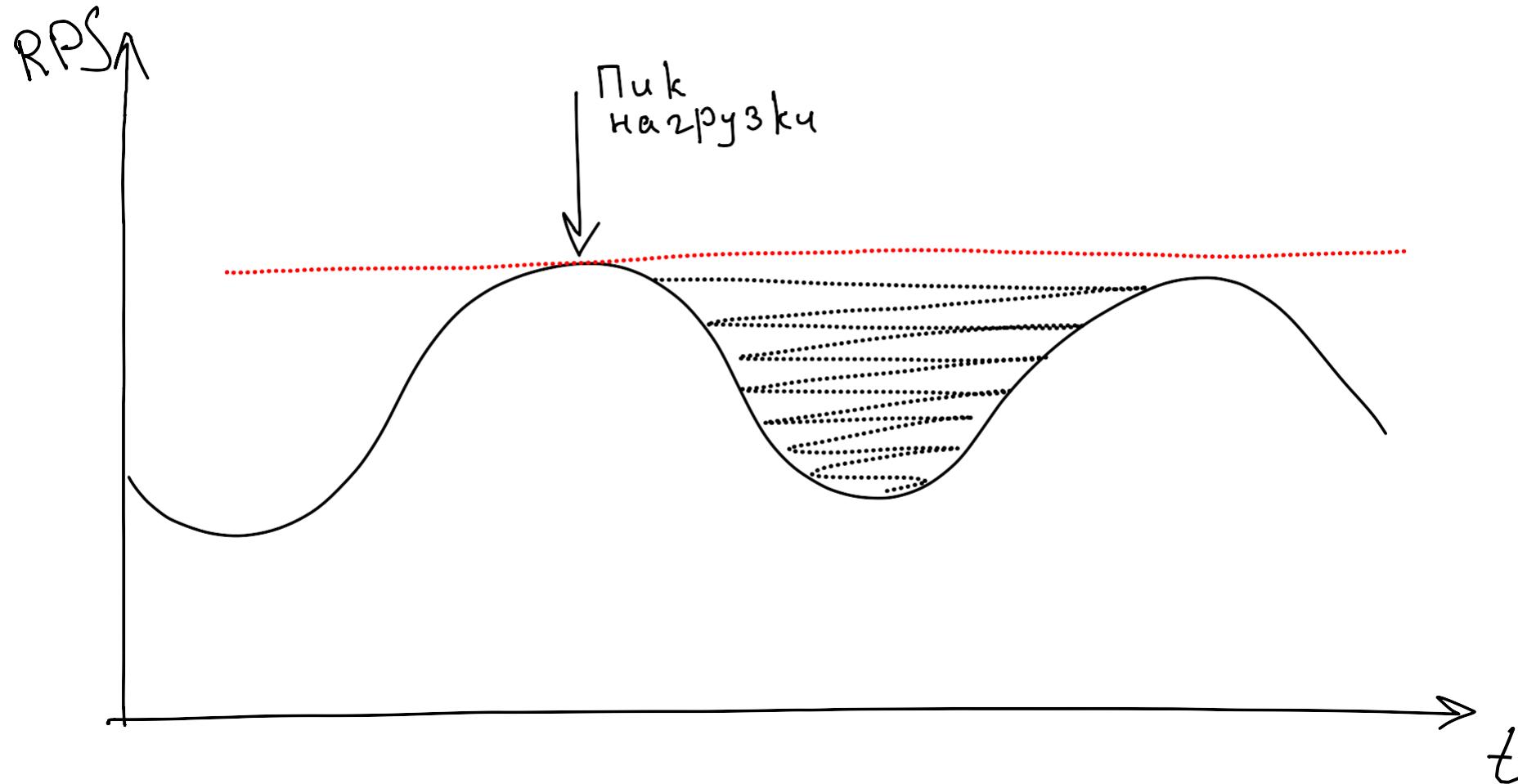
РАСПРЕДЕЛЕНИЕ ЗАДАЧ



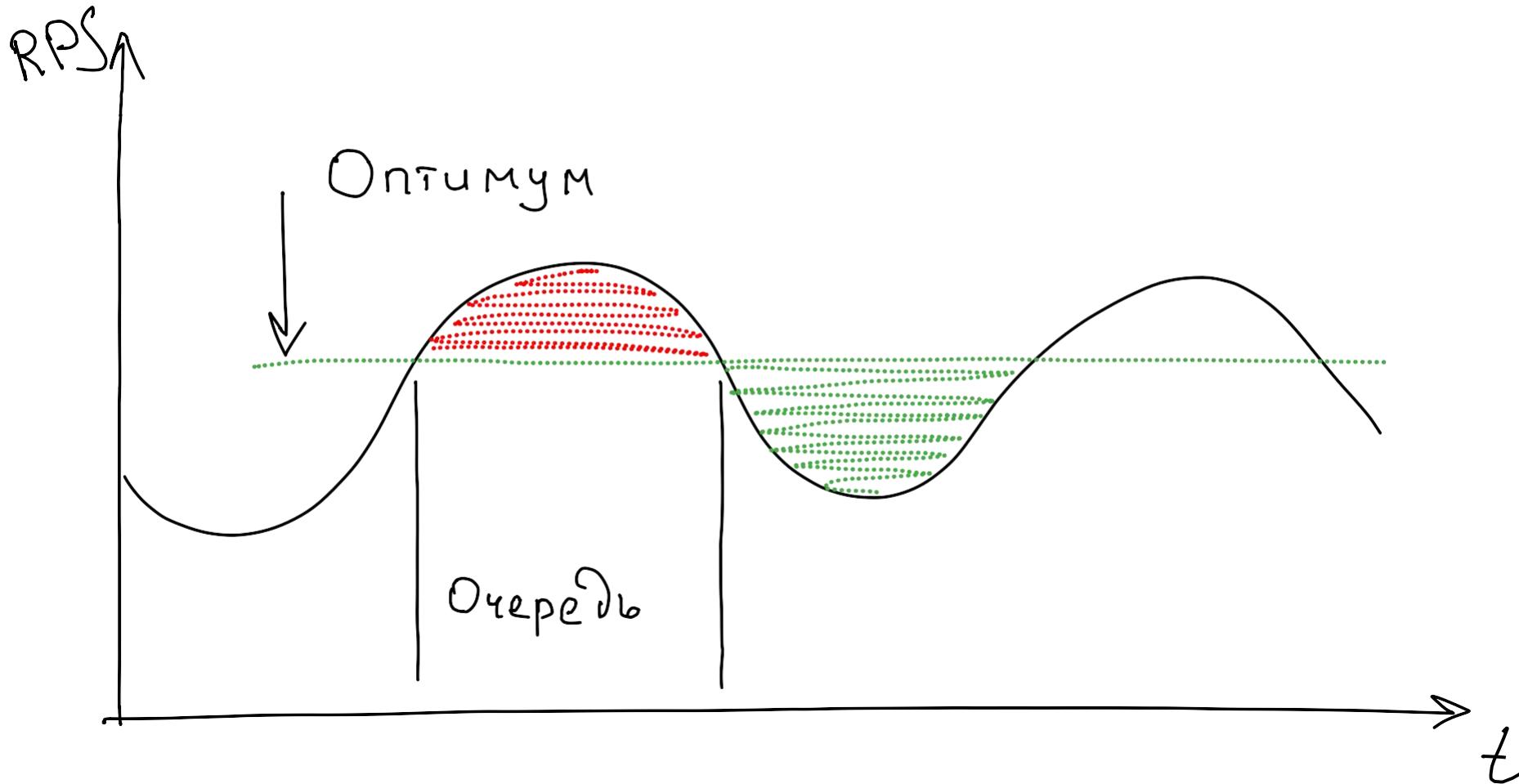
ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения

ПЛАНИРОВАНИЕ ИСПОЛНЕНИЯ



ПЛАНИРОВАНИЕ ИСПОЛНЕНИЯ



ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов

ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов
- Репликация сообщений

ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов
- Репликация сообщений
- Отказоустойчивость, надёжность, гарантия доставки

ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов
- Репликация сообщений
- Отказоустойчивость, надёжность, гарантия доставки
- Коммуникация микросервисов

ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов
- Репликация сообщений
- Отказоустойчивость, надёжность, гарантия доставки
- Коммуникация микросервисов
- Событийная архитектура (Event Sourcing)

ЗАЧЕМ НУЖНЫ ОЧЕРЕДИ?

- Распределение задач
- Планирование исполнения
- Честность выделения ресурсов
- Репликация сообщений
- Отказоустойчивость, надёжность, гарантия доставки
- Коммуникация микросервисов
- Событийная архитектура (Event Sourcing)
- Потоковая архитектура (Streaming)

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»

IRQ

NCQ

Hardware Buffers

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
 - epoll / kqueue
 - networking
 - signal handling

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
- Приложения

Cross thread
IPC

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
- Приложения
- Сетевые взаимодействия

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
- Приложения
- Сетевые взаимодействия
- Распределённые системы

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

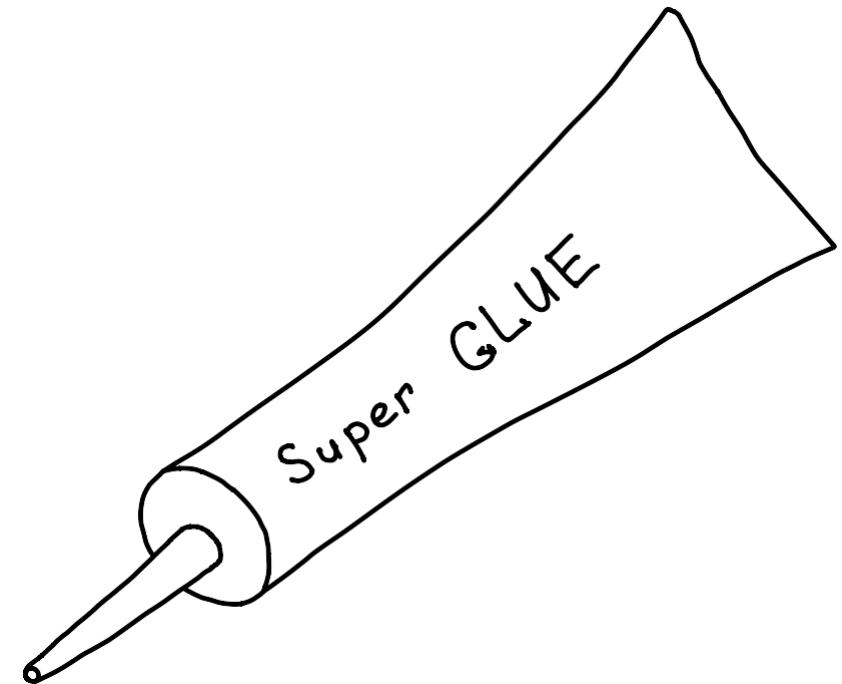
- «Железо»
- Ядро операционной системы
- Приложения
- Сетевые взаимодействия
- Распределённые системы
- Стык разных бизнесов

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
- Приложения
- Сетевые взаимодействия
- Распределённые системы
- Стык разных бизнесов
- Фактически — везде

ГДЕ ПРИМЕНЯЮТСЯ ОЧЕРЕДИ?

- «Железо»
- Ядро операционной системы
- Приложения
- Сетевые взаимодействия
- Распределённые системы
- Стык разных бизнесов
- Фактически — везде

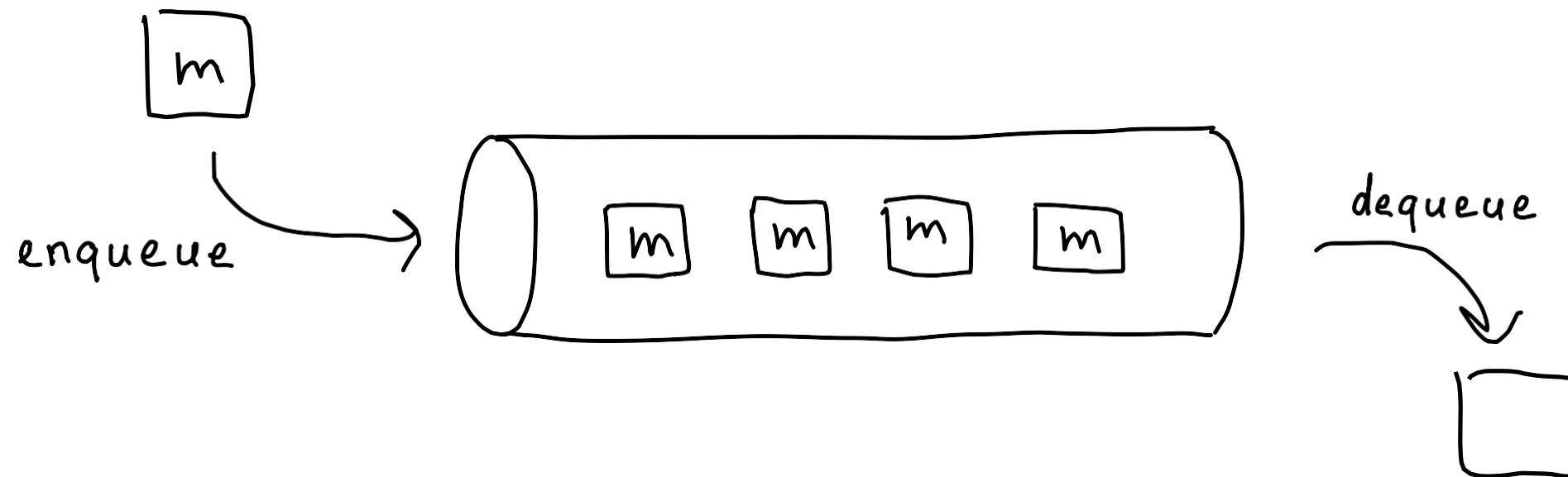


Очереди — это клей

ЧТО ТАКОЕ ОЧЕРЕДЬ?

- Средство коммуникации при помощи сообщений

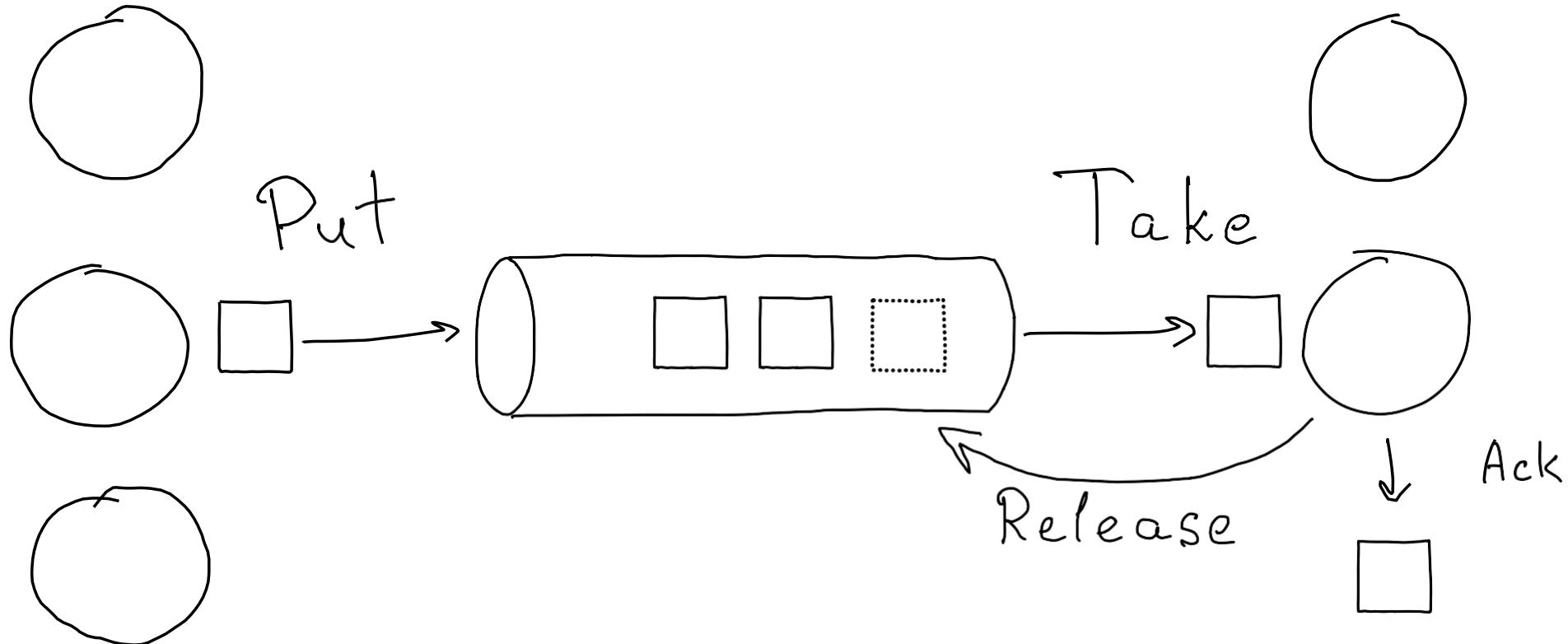
ЧТО ТАКОЕ ОЧЕРЕДЬ?



ЧТО ТАКОЕ ОЧЕРЕДЬ?

- Средство коммуникации при помощи сообщений
- Подход Put/Take: $1 \rightarrow 1$

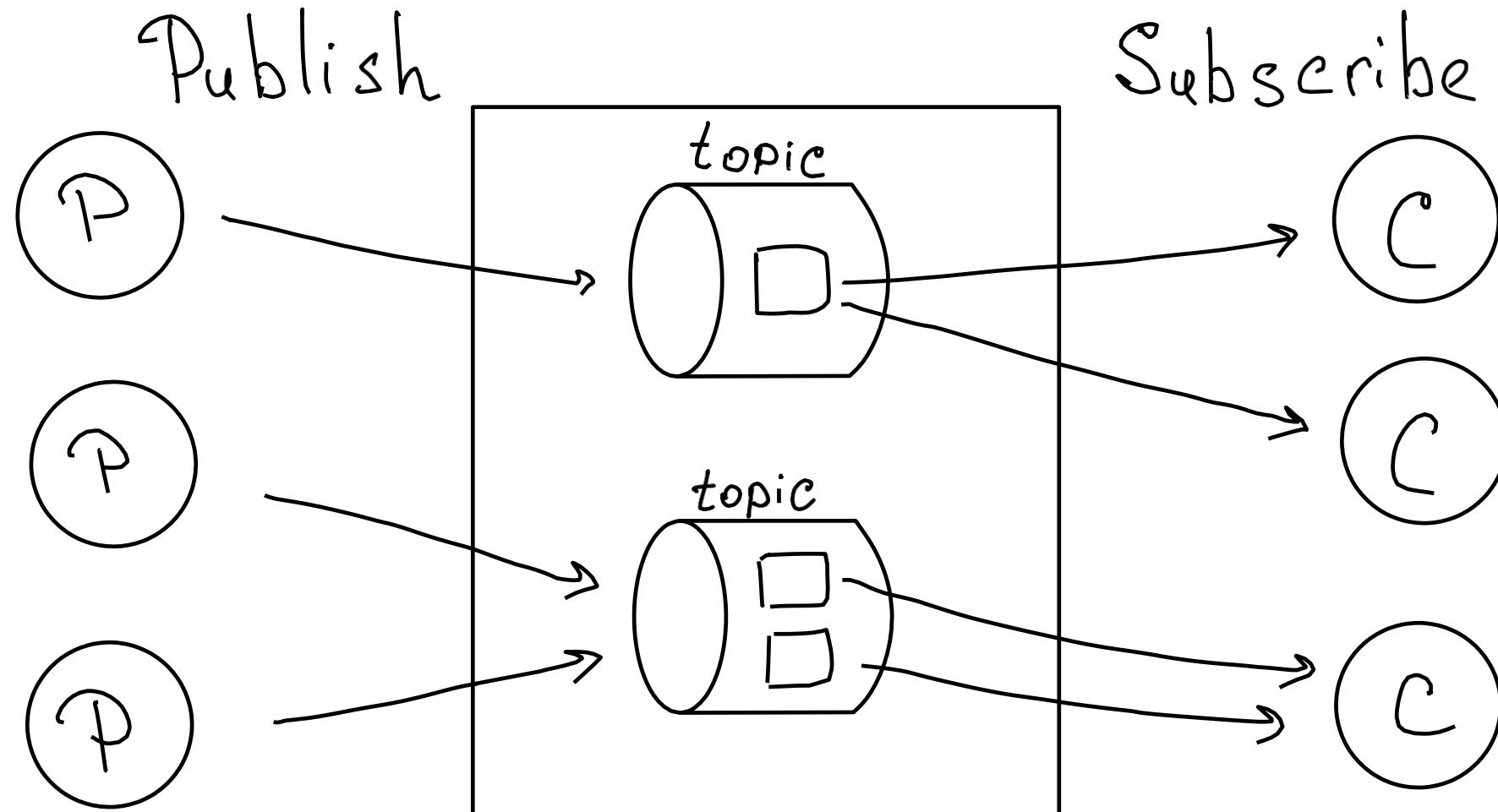
ПОДХОД PUT/TAKE



ЧТО ТАКОЕ ОЧЕРЕДЬ?

- Средство коммуникации при помощи сообщений
- Подход Put/Take: $1 \rightarrow 1$
- Подход Publish/Subscribe: $1 \rightarrow *$

ПОДХОД PUB/SUB



ЧТО ТАКОЕ ОЧЕРЕДЬ?

- Средство коммуникации при помощи сообщений
- Подход Put/Take: $1 \rightarrow 1$
- Подход Publish/Subscribe: $1 \rightarrow *$
- Подход Request/Response: $1 \leftrightharpoons 1$

ЧТО ТАКОЕ ОЧЕРЕДЬ?

- Средство коммуникации при помощи сообщений
- Подход Put/Take: $1 \rightarrow 1$
- Подход Publish/Subscribe: $1 \rightarrow *$
- Подход Request/Response: $1 \leftrightharpoons 1$
- Протоколы: AMQP, MQTT, STOMP, NATS, ZeroMQ, ...

КАКИЕ ЕСТЬ ВАРИАНТЫ?

- Облачные решения
 - Amazon SQS
 - Mail.ru Cloud Queues
 - Yandex Message Queue
 - CloudAMQP
 - ...

КАКИЕ ЕСТЬ ВАРИАНТЫ?

- Облачные решения
 - Amazon **SQS**, Mail.ru Cloud Queues, Yandex MQ, CloudAMQP, ...
- Специализированные брокеры
 - **RabbitMQ**
 - **Apache Kafka**
 - **ActiveMQ**
 - Tarantool Queue
 - **NATS**
 - **NSQ**
 - ...

КАКИЕ ЕСТЬ ВАРИАНТЫ?

- Облачные решения
 - Amazon **SQS**, Mail.ru Cloud Queues, Yandex MQ, CloudAMQP, ...
- Специализированные брокеры
 - **RabbitMQ**, Apache **Kafka**, ActiveMQ, Tarantool Queue, **NATS**, NSQ, Beanstalkd, ...
- Реализация очереди с помощью СУБД
 - PgQueue
 - Tarantool
 - Redis
 - ...

КАКИЕ ЕСТЬ ВАРИАНТЫ?

- Облачные решения
 - Amazon **SQS**, Mail.ru Cloud Queues, Yandex MQ, CloudAMQP, ...
- Специализированные брокеры
 - **RabbitMQ**, Apache **Kafka**, ActiveMQ, Tarantool Queue, **NATS**, NSQ, Beanstalkd, ...
- Реализация очереди с помощью СУБД
 - PgQueue, Tarantool, Redis, ...
- «Сокеты на стероидах»
 - NATS, ZeroMQ

ОСНОВНЫЕ КАНДИДАТЫ

- Apache Kafka
 - Распределённый лог сообщений для стриминга

ОСНОВНЫЕ КАНДИДАТЫ

- Apache Kafka
 - Распределённый лог сообщений для стриминга
- RabbitMQ
 - Традиционный брокер с протоколом AMQP

ОСНОВНЫЕ КАНДИДАТЫ

- Apache Kafka
 - Распределённый лог сообщений для стриминга
- RabbitMQ
 - Традиционный брокер с протоколом AMQP
- Managed Cloud Queue (SQS/MQ/...)
 - Максимальное удобство в облаках

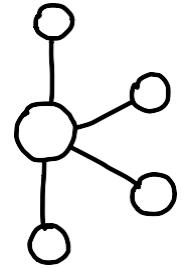
ОСНОВНЫЕ КАНДИДАТЫ

- Apache Kafka
 - Распределённый лог сообщений для стриминга
- RabbitMQ
 - Традиционный брокер с протоколом AMQP
- Managed Cloud Queue (SQS/MQ/...)
 - Максимальное удобство в облаках
- NATS
 - Связующее звено для микросервисов

ОСНОВНЫЕ КАНДИДАТЫ

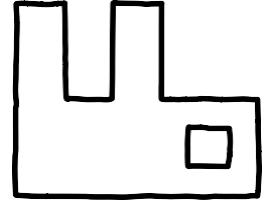
- Apache Kafka
 - Распределённый лог сообщений для стриминга
- RabbitMQ
 - Традиционный брокер с протоколом AMQP
- Managed Cloud Queue (SQS/MQ/...)
 - Максимальное удобство в облаках
- NATS
 - Связующее звено для микросервисов
- Tarantool
 - Платформа для произвольных очередей

APACHE KAFKA



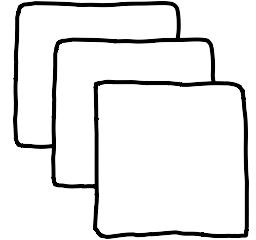
- Реплицированный шардированный лог сообщений
- Строгая упорядоченность (FIFO)
- Ограничена по количеству потребителей
- Повторное проигрывание последовательности
- Интеграция с экосистемой Apache
- Основные сценарии использования
 - Анализ данных. Логи, метрики, аудит
 - Производительный процессинг потоковых данных
 - Репликация данных

RABBITMQ



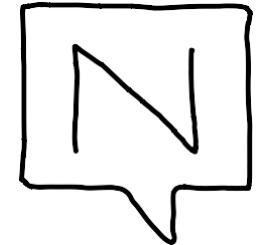
- Протоколы: AMQP, MQTT, STOMP
- Приоритеты, отложенные и фоновые задачи
- Нет ограничений на количество потребителей
- Хранение: память, диск, репликация, кворум
- Простой в освоении. Сложный в отказоустойчивости
- Основные сценарии использования
 - Традиционный pub/sub брокер
 - Слой соединения микросервисов. Шина сообщений

MANAGED CLOUD QUEUE



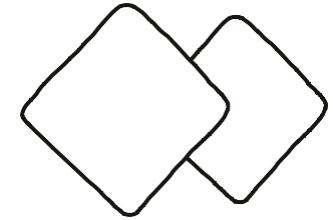
- Надёжная, автоматически масштабируемая очередь
- Протокол без состояния: простая коммуникация
- Стандартизированное API
- Минимум затрат при низком потреблении
- Основные сценарии использования
 - Коммуникация между сервисами в облаке
 - Связующее звено для S3 и Lambda

NATS MESSAGING



- Быстрый неперсистентный обмен сообщениями
- Высокая производительность и масштабируемость
- Любые сценарии: pub/sub, put/take, req/res
- При использовании JetStream
 - Потоковая обработка
 - Надёжное хранение, RAFT cluster
- Основные сценарии использования
 - Инструмент для общения в распределённых системах

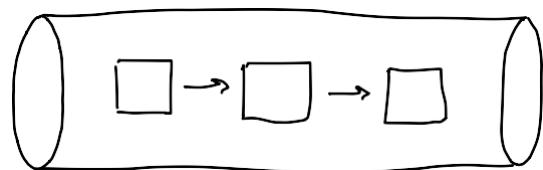
TARANTOOL



- Готовый брокер с репликацией (Tarantool Queue)
- Интеграция со стриминговыми очередями
- Модули для построения собственных очередей
- Любая произвольная логика
- Транзакционность в рамках одного брокера
- Основные сценарии использования
 - Производительный брокер для традиционных сценариев
 - Построение сложных очередей с собственной логикой

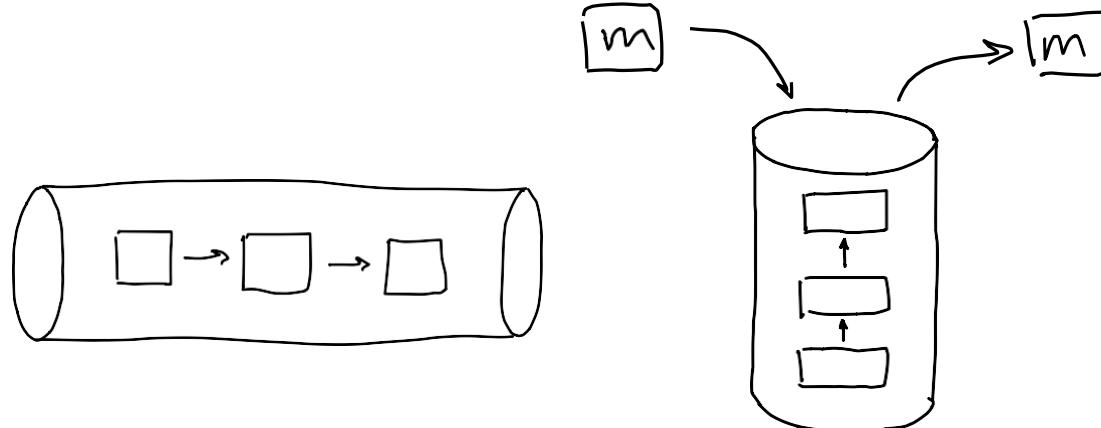
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS



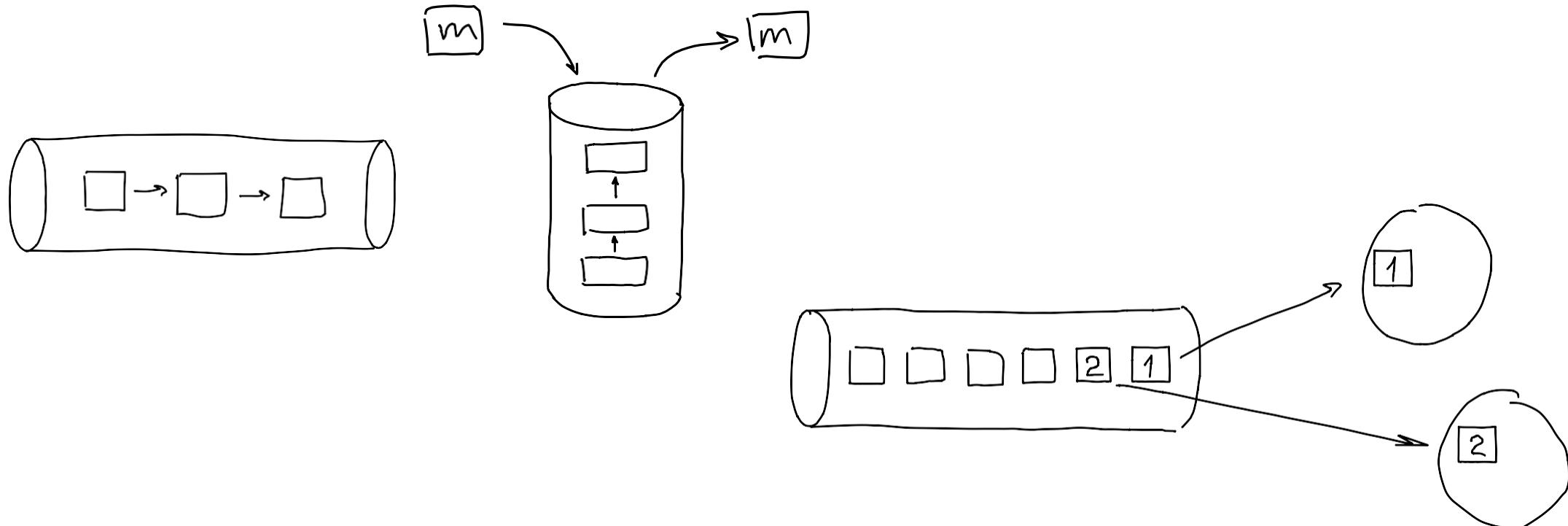
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS



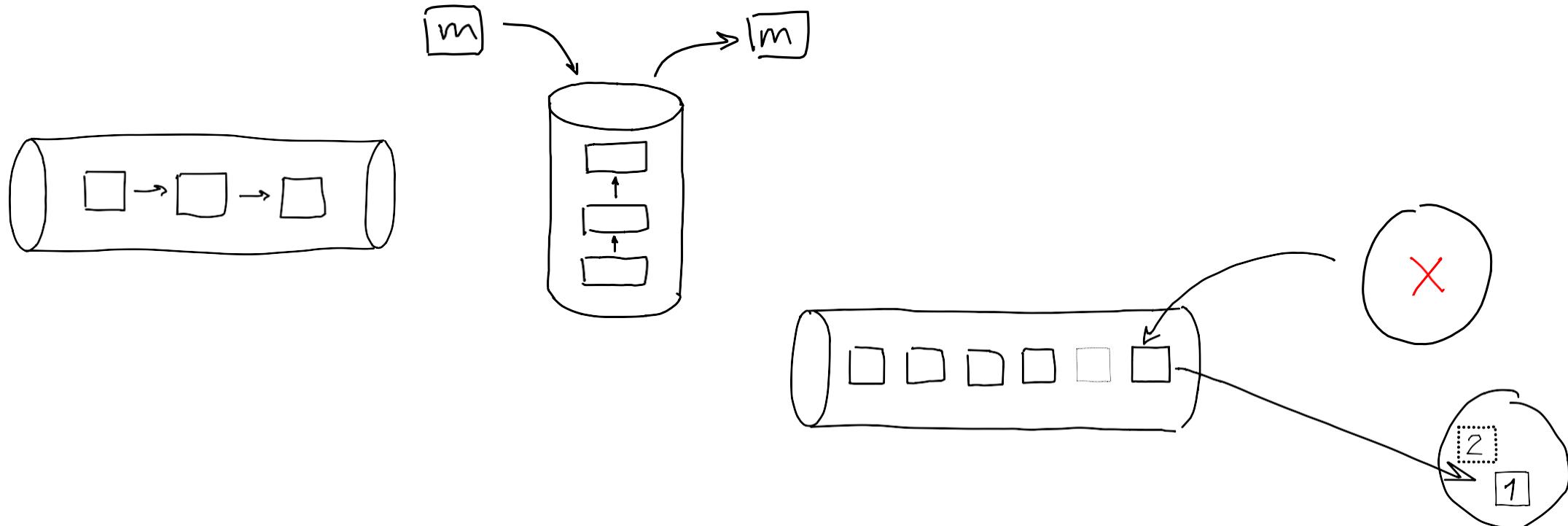
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS



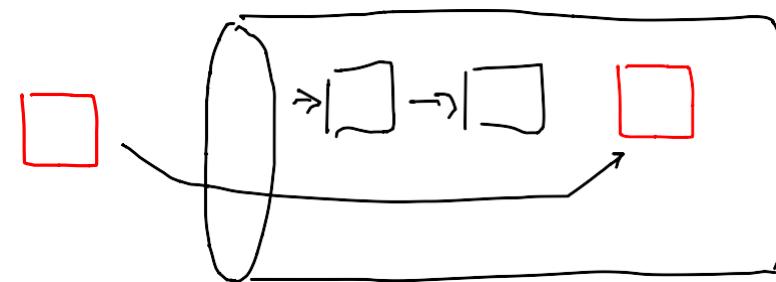
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS



ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений



ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений
- Организация подочередей

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений
- Организация подочередей
- Повтор, отложенные задачи, повтор с задержкой

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений
- Организация подочередей
- Повтор, отложенные задачи, повтор с задержкой
- Dead letter queue (и упорядочивание)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений
- Организация подочередей
- Повтор, отложенные задачи, повтор с задержкой
- Dead letter queue (и упорядочивание)
- Созависимые задачи

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- FIFO, LIFO, Best Effort, QoS
- Приоритизация сообщений
- Организация подочередей
- Повтор, отложенные задачи, повтор с задержкой
- Dead letter queue (и упорядочивание)
- Созависимые задачи
- TTL, TTR, Putback

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- Приоритизация и голодание (*Starvation*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- Приоритизация и голодание (*Starvation*)
- Пропускная способность (*Throughput*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- Приоритизация и голодание (*Starvation*)
- Пропускная способность (*Throughput*)
- Производительность (*Performance*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- Приоритизация и голодание (*Starvation*)
- Пропускная способность (*Throughput*)
- Производительность (*Performance*)
- Масштабируемость (*Scalability*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

- Приоритизация и голодание (*Starvation*)
- Пропускная способность (*Throughput*)
- Производительность (*Performance*)
- Масштабируемость (*Scalability*)
- Ограниченност (*Capacity*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: АЛГОРИТМЫ

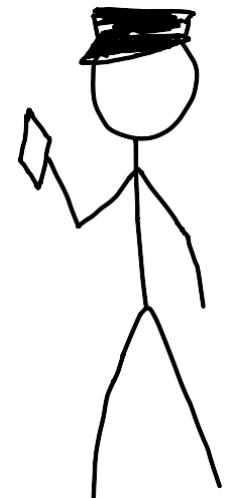
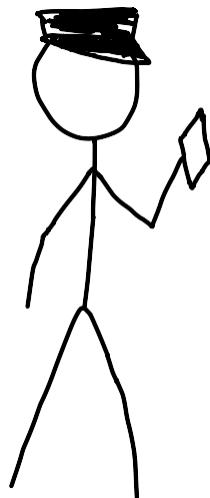
- Приоритизация и голодание (*Starvation*)
- Пропускная способность (*Throughput*)
- Производительность (*Performance*)
- Масштабируемость (*Scalability*)
- Ограниченност (*Capacity*)
- Сохранность сообщений (*Durability*)*

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: СЕТЬ

- Undefined behavior

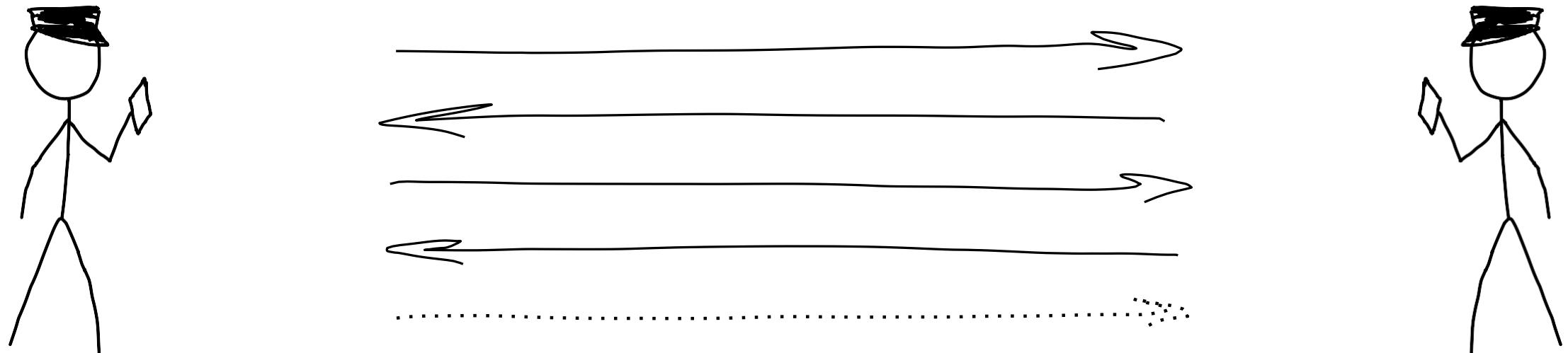
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: СЕТЬ

- Проблема двух генералов



ПРОБЛЕМЫ ОЧЕРЕДЕЙ: СЕТЬ

- Проблема двух генералов

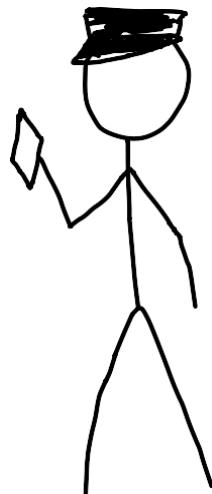
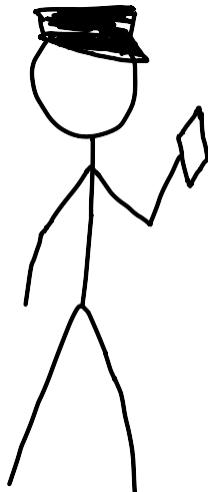


ПРОБЛЕМЫ ОЧЕРЕДЕЙ: СЕТЬ

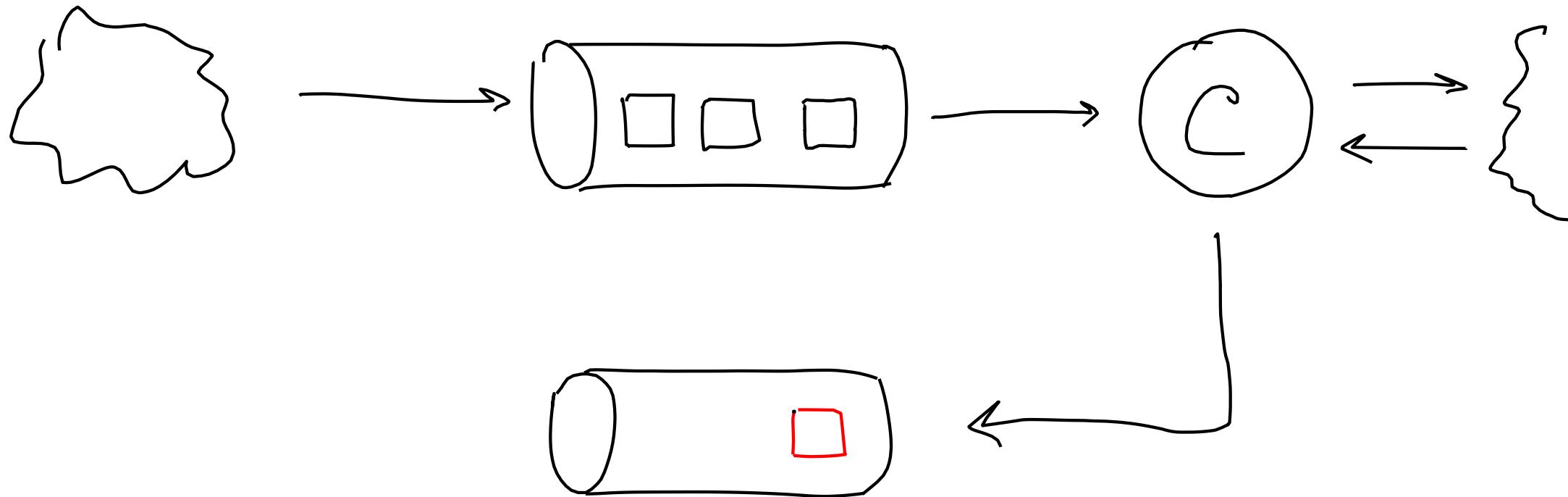
- Проблема двух генералов

*Есть две сложные проблемы
в распределённых системах:*

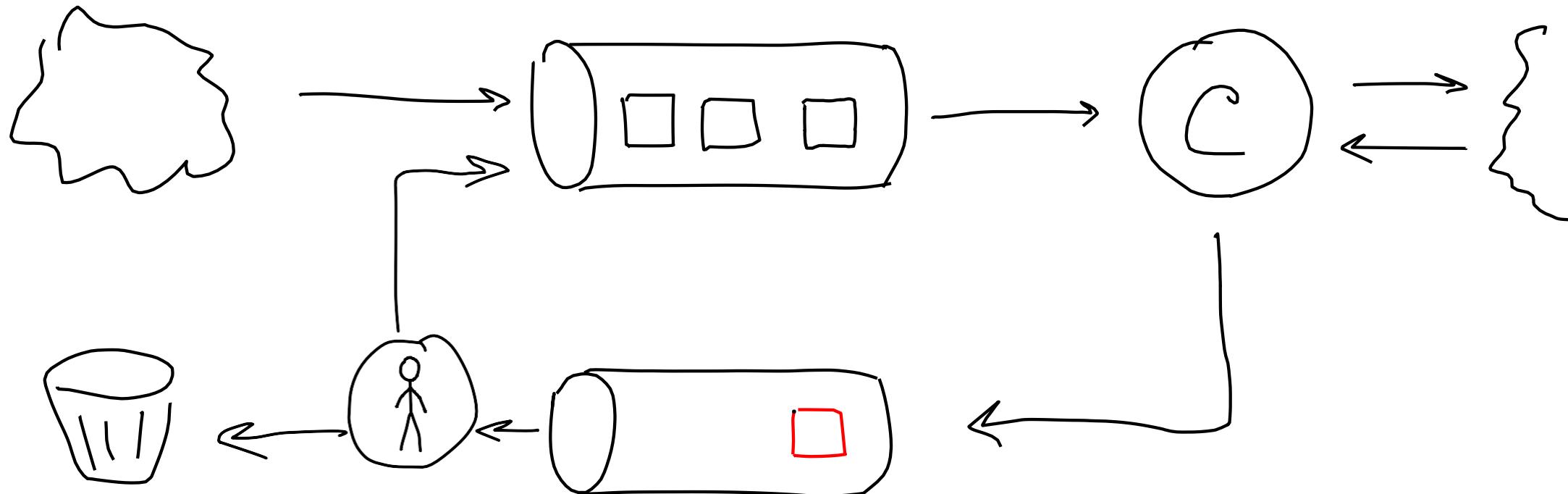
2. Доставка строго один раз
1. Строгий порядок сообщений
2. Доставка строго один раз



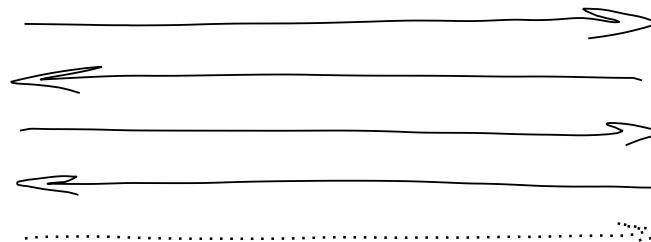
EXACTLY ONCE



EXACTLY ONCE



«РЕШЕНИЕ» ПРОБЛЕМЫ «EXACTLY ONCE»



ПРОБЛЕМЫ ОЧЕРЕДЕЙ: СЕТЬ И ДИСК

- Пропускная способность (*Throughput*)
- Задержка в обработке (*Latency*)

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: ОТКАЗЫ

- Оборудование
 - Диск
 - Хост
 - Дата-центр

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: ОТКАЗЫ

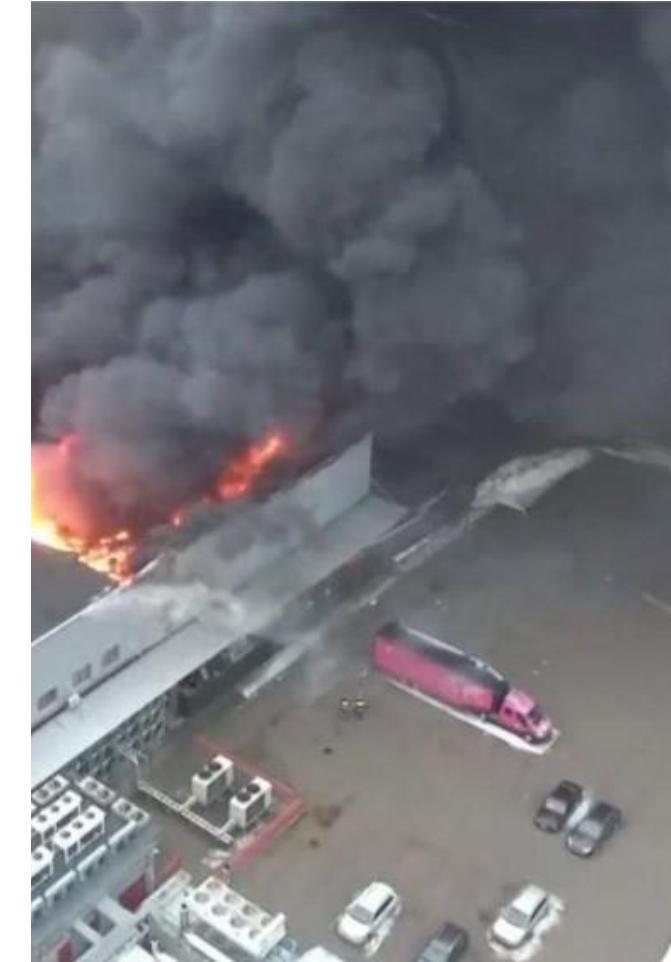
- Оборудование
 - Диск
 - Хост
 - Дата-центр
- Временный отказ
 - Питание
 - Сеть
 - Split brain

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: ОТКАЗЫ

- Оборудование
 - Диск
 - Хост
 - Дата-центр
- Временный отказ
 - Питание
 - Сеть
 - Split brain
- Отказ навсегда
 - Физическое уничтожение

ПРОБЛЕМЫ ОЧЕРЕДЕЙ: ОТКАЗЫ

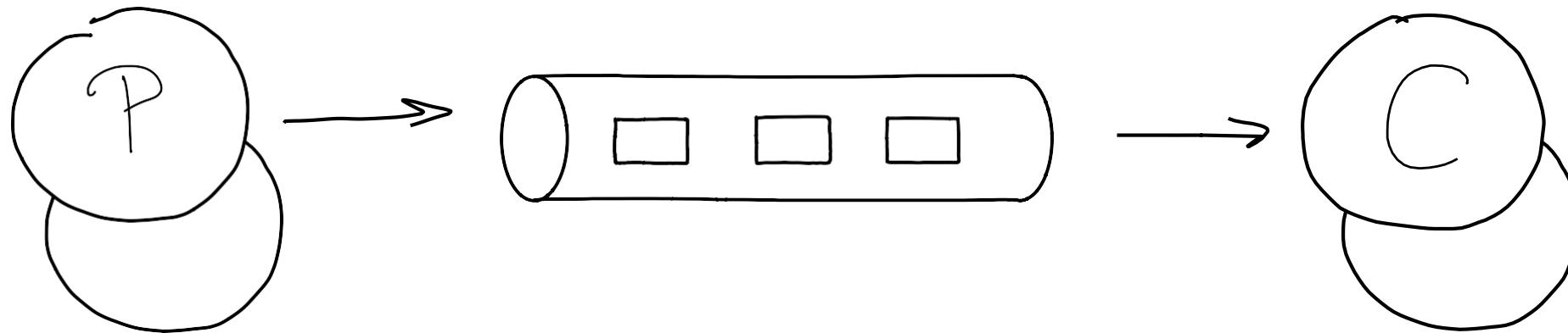
- Оборудование
 - Диск
 - Хост
 - Дата-центр
- Временный отказ
 - Питание
 - Сеть
 - Split brain
- Отказ навсегда
 - Физическое уничтожение



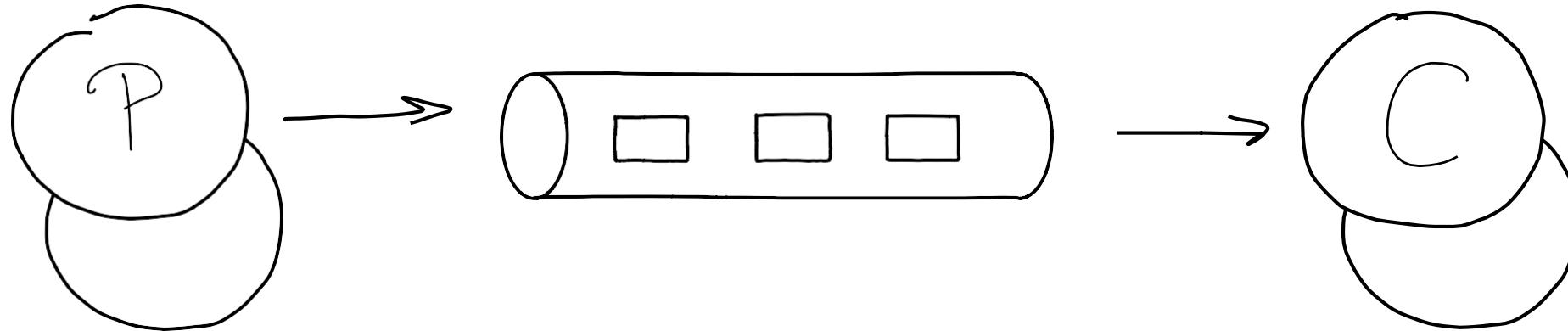
ПРОБЛЕМЫ ОЧЕРЕДЕЙ: ОТКАЗЫ

- Доступность (*Availability*)
 - Возможность сохранить сообщение
- Надёжность (*Durability*)
 - Гарантия сохранности и доставки сообщения

ТОПОЛОГИИ ОЧЕРЕДЕЙ: SINGLE INSTANCE

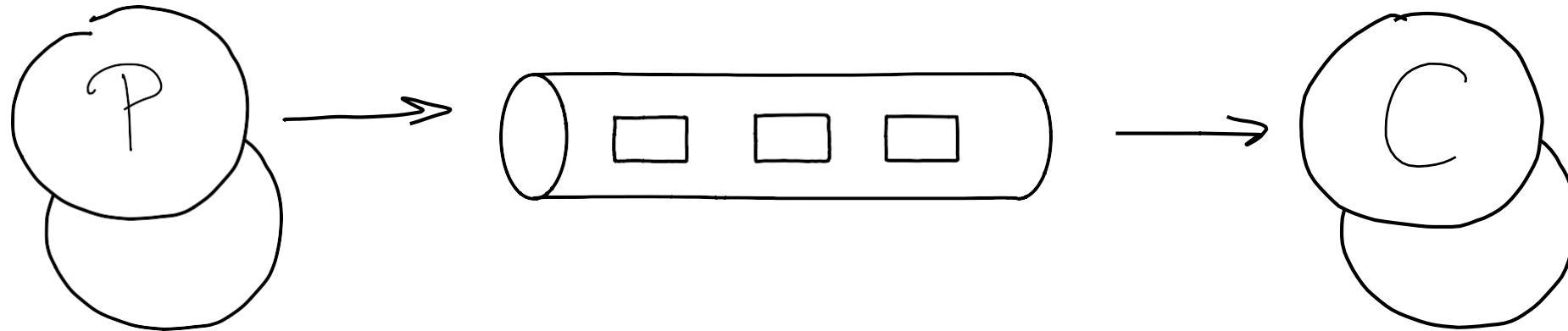


ТОПОЛОГИИ ОЧЕРЕДЕЙ: SINGLE INSTANCE



- Масштабируемость:
нет Гарантии: $X \leq 1, X \geq 1$

ТОПОЛОГИИ ОЧЕРЕДЕЙ: SINGLE INSTANCE



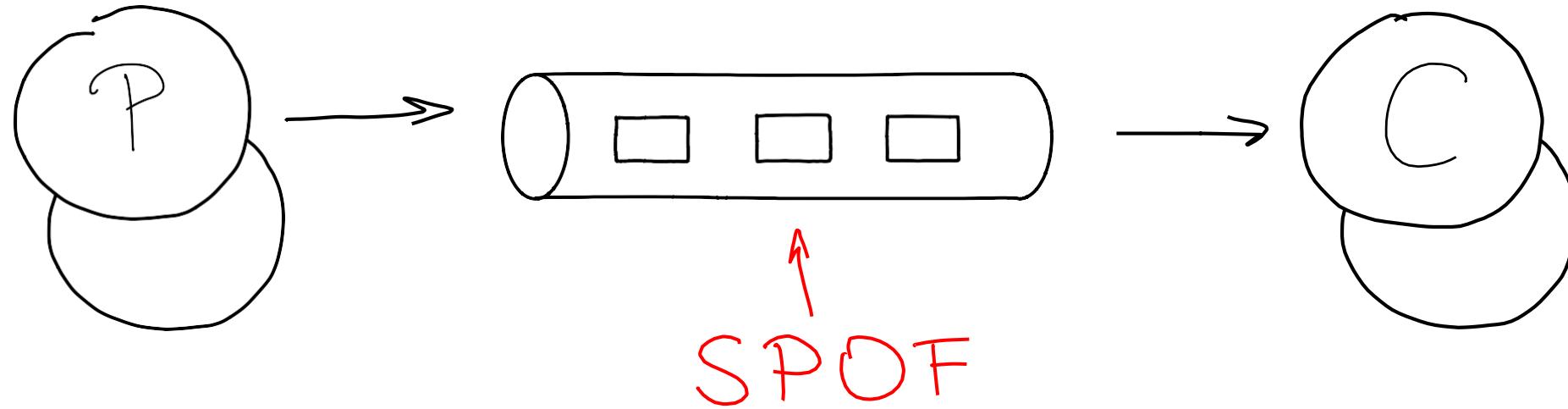
Масштабируемость: **нет**

Гарантии: **$X \leq 1$, $X \geq 1$**

Доступность: **низкая**

Надёжность: **низкая**

ТОПОЛОГИИ ОЧЕРЕДЕЙ: SINGLE INSTANCE



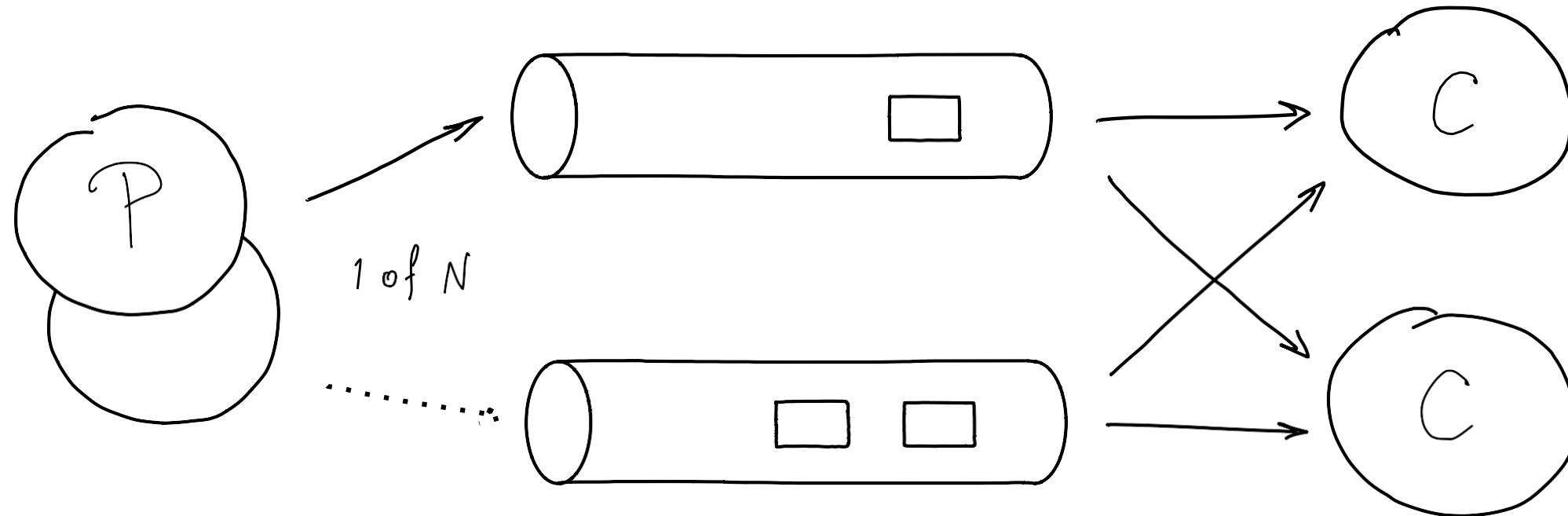
Масштабируемость: **нет**

Гарантии: **$X \leq 1$, $X \geq 1$**

Доступность: **низкая**

Надёжность: **низкая**

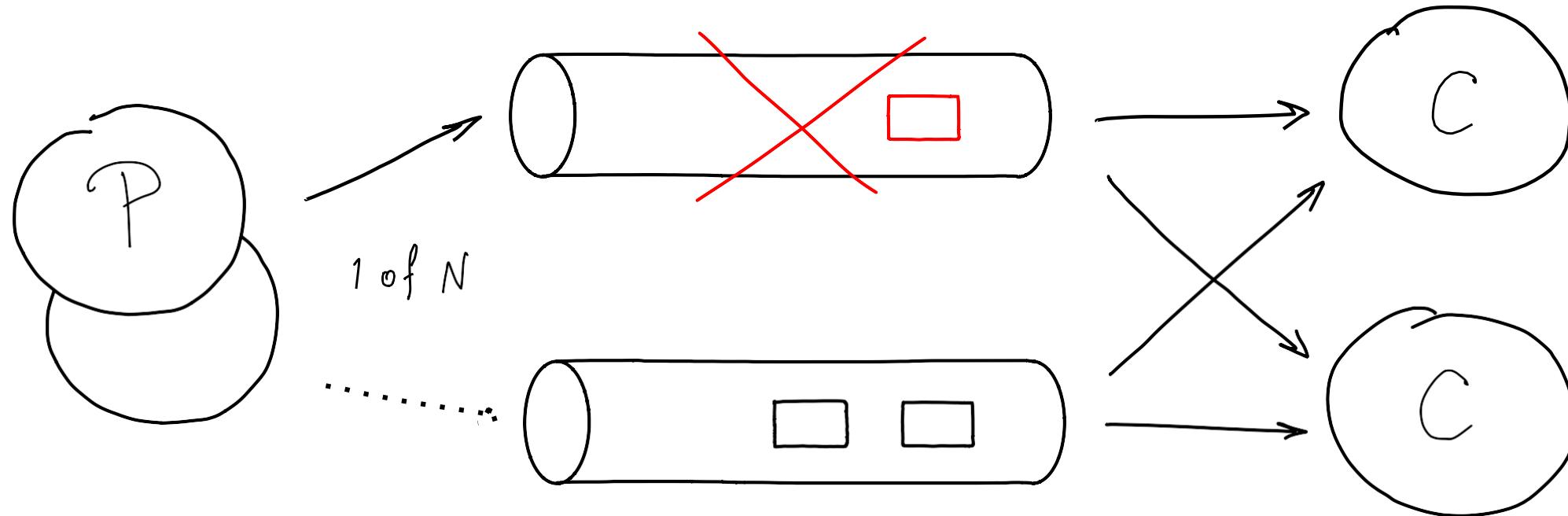
ТОПОЛОГИИ ОЧЕРЕДЕЙ: MULTI-INSTANCE



Масштабируемость: да

Гарантии: $X \leq 1$, $X \geq 1$

НЕСКОЛЬКО ОЧЕРЕДЕЙ, КЛАДЁМ В 1



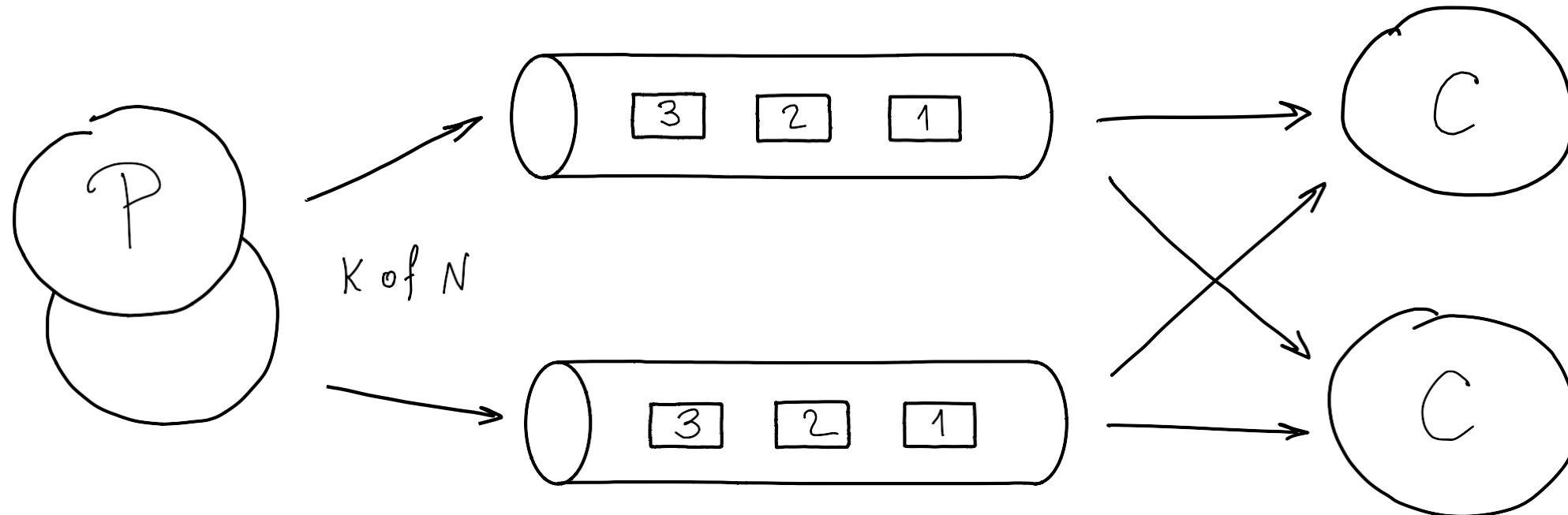
Масштабируемость: **да**

Гарантии: **$X \leq 1$, $X \geq 1$**

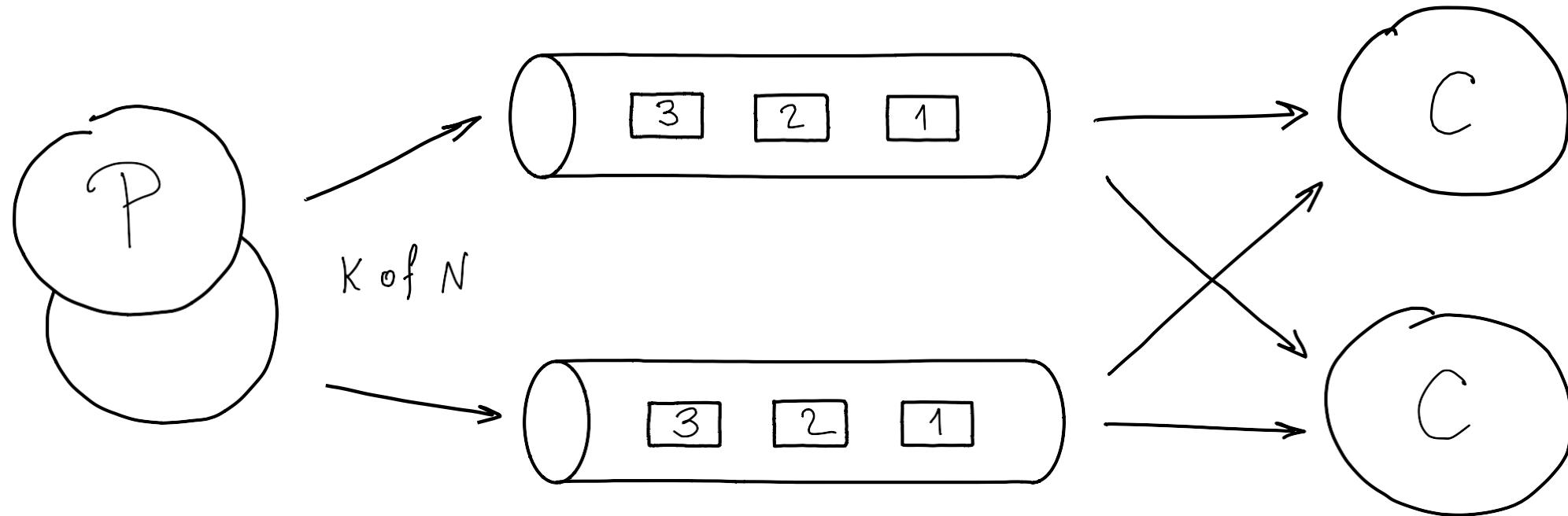
Доступность: **высокая**

Надёжность: **средняя**

НЕСКОЛЬКО ОЧЕРЕДЕЙ, КЛАДЁМ В К ИЗ N



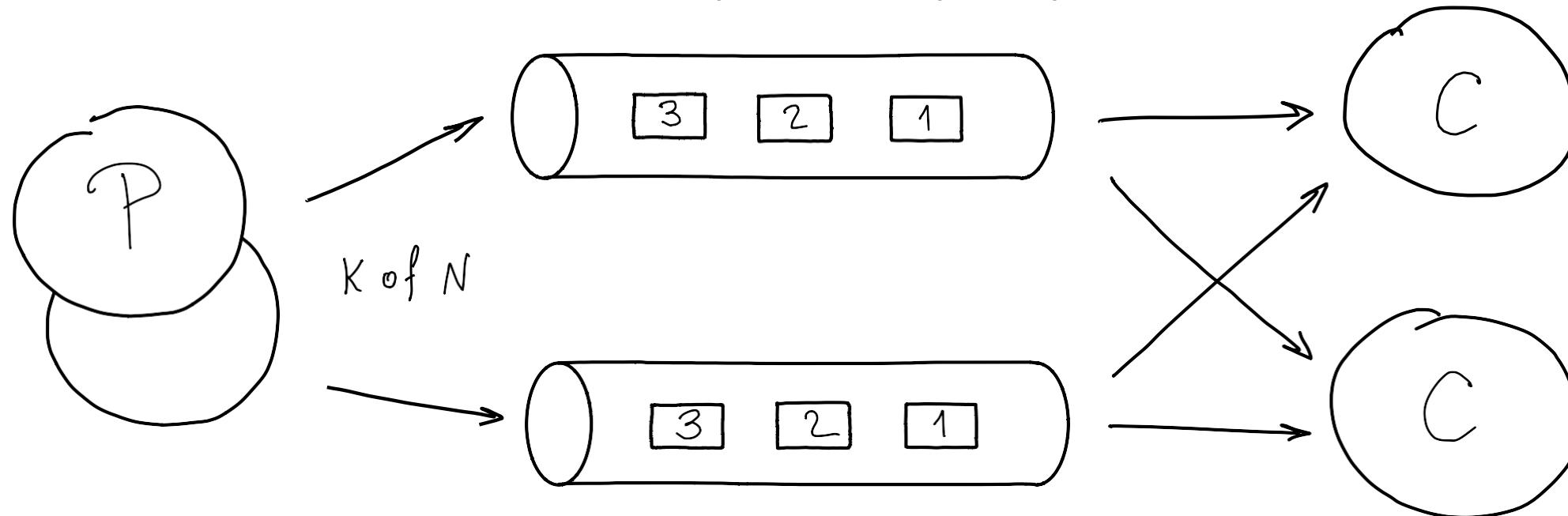
НЕСКОЛЬКО ОЧЕРЕДЕЙ, КЛАДЁМ В К ИЗ N



Доступность: **высокая**
Надёжность: **высокая**

НЕСКОЛЬКО ОЧЕРЕДЕЙ, КЛАДЁМ В К ИЗ N

Подход: «многократное дублирование»



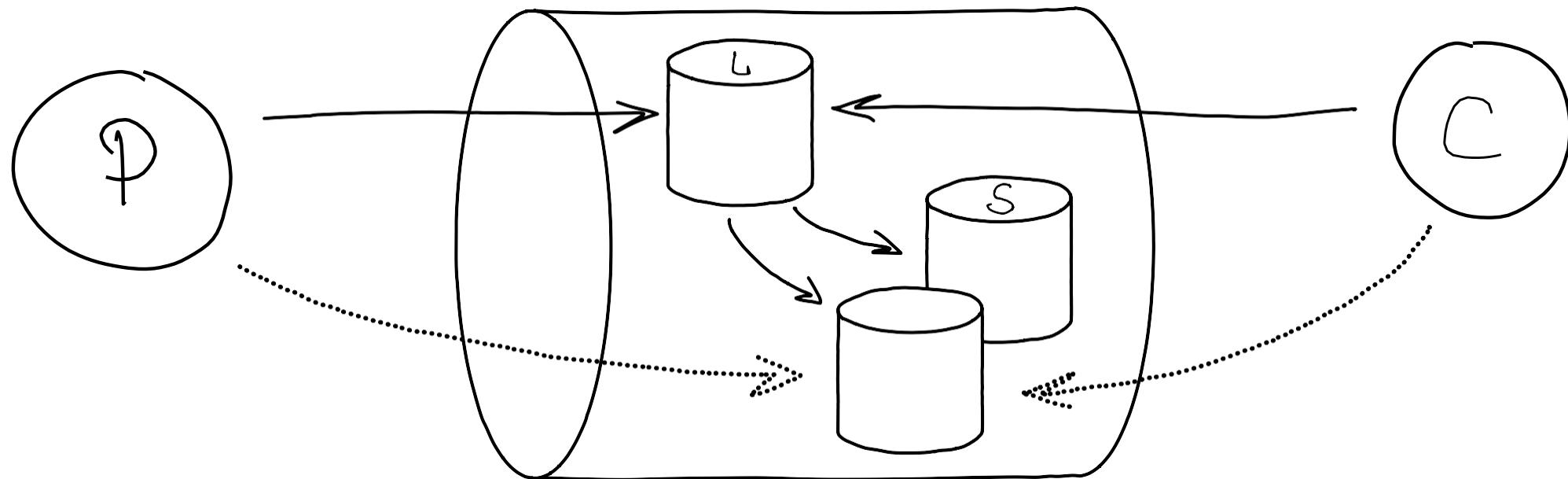
Масштабируемость: **да**

Гарантии: **$X \geq K$**

Доступность: **высокая**

Надёжность: **высокая**

РЕПЛИКАЦИЯ



Коммуникация
с лидером

Реплики
в ожидании

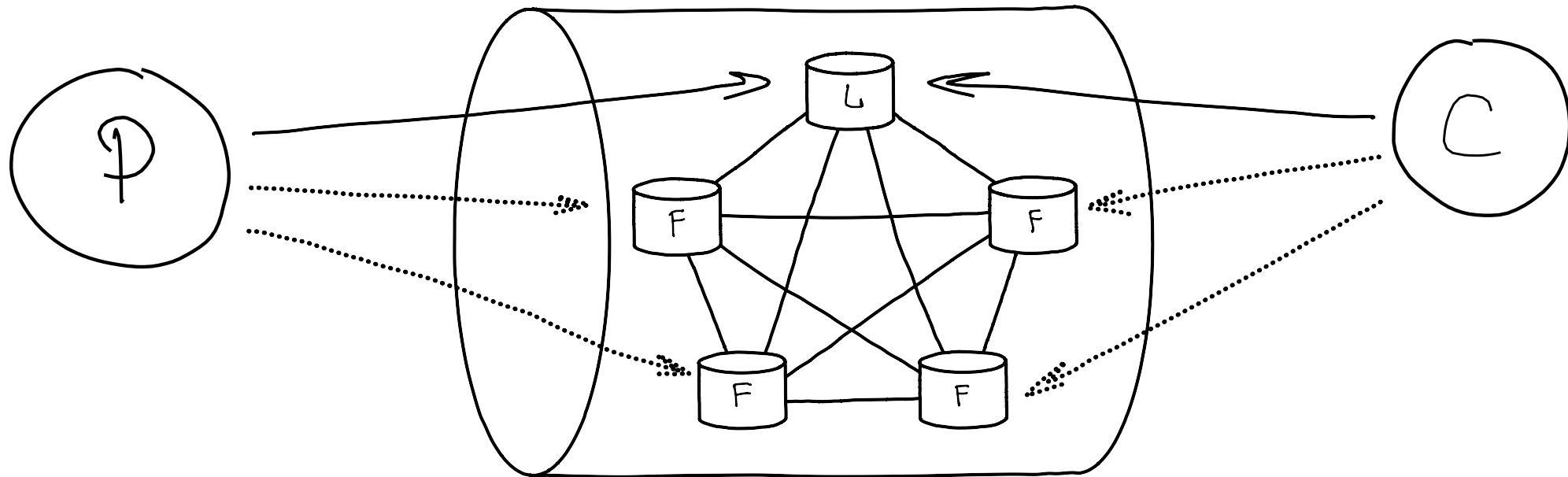
РЕПЛИЦИРОВАННЫЕ ОЧЕРЕДИ, 1 из N



Масштабируемость: **да**
Гарантии: **$X \approx 1$ ($X \geq 1$)**

Доступность: **высокая**
Надёжность: **высокая**

ПОДХОД БАЗ ДАННЫХ: КВОРУМ

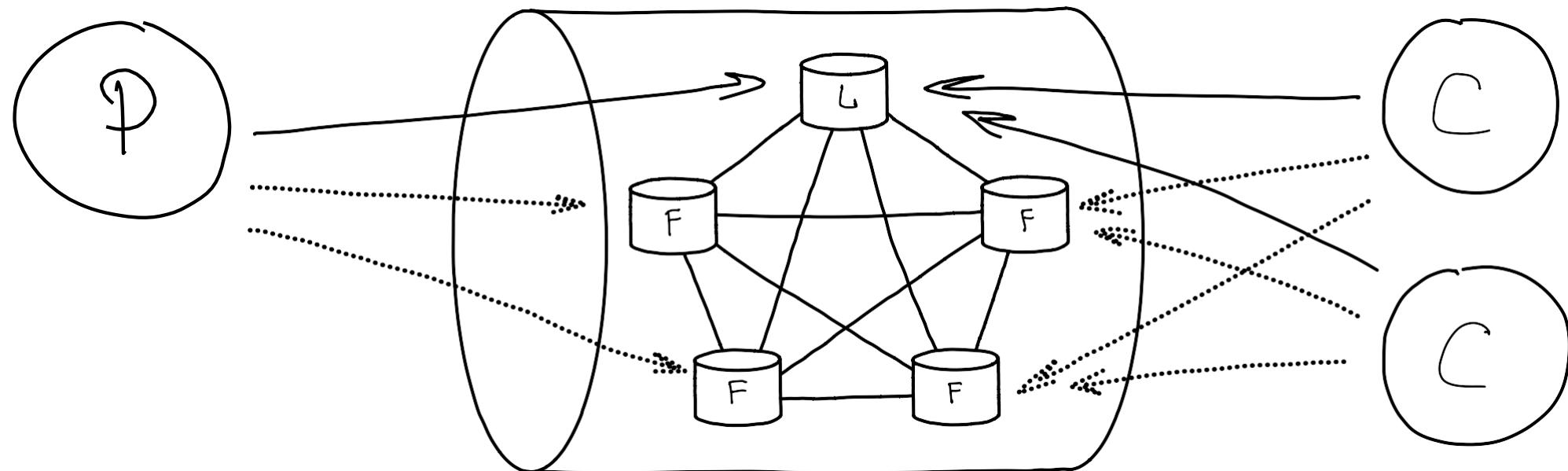


Кворумная запись защищает от потерь данных

Кворум гарантирует консистентность $X \rightarrow 1$ ($X \geq 1$)

Надёжно, но медленно

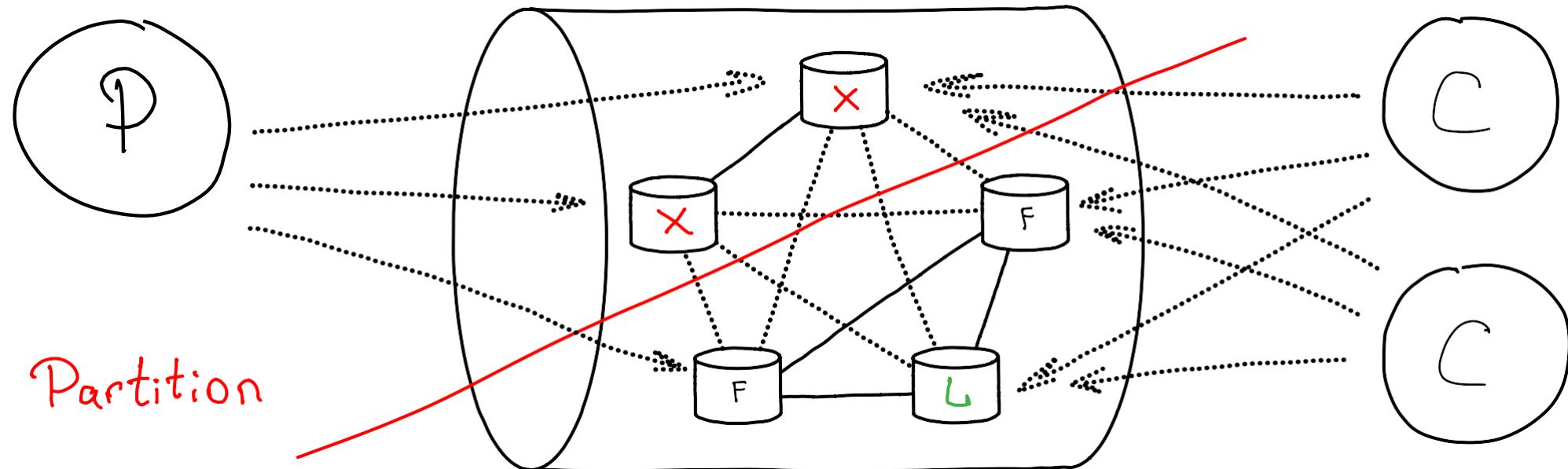
КВОРУМНЫЙ КЛАСТЕР ОЧЕРЕДИ



Гарантии: $X \approx 1$ ($X \geq 1$)

Надёжность: высокая

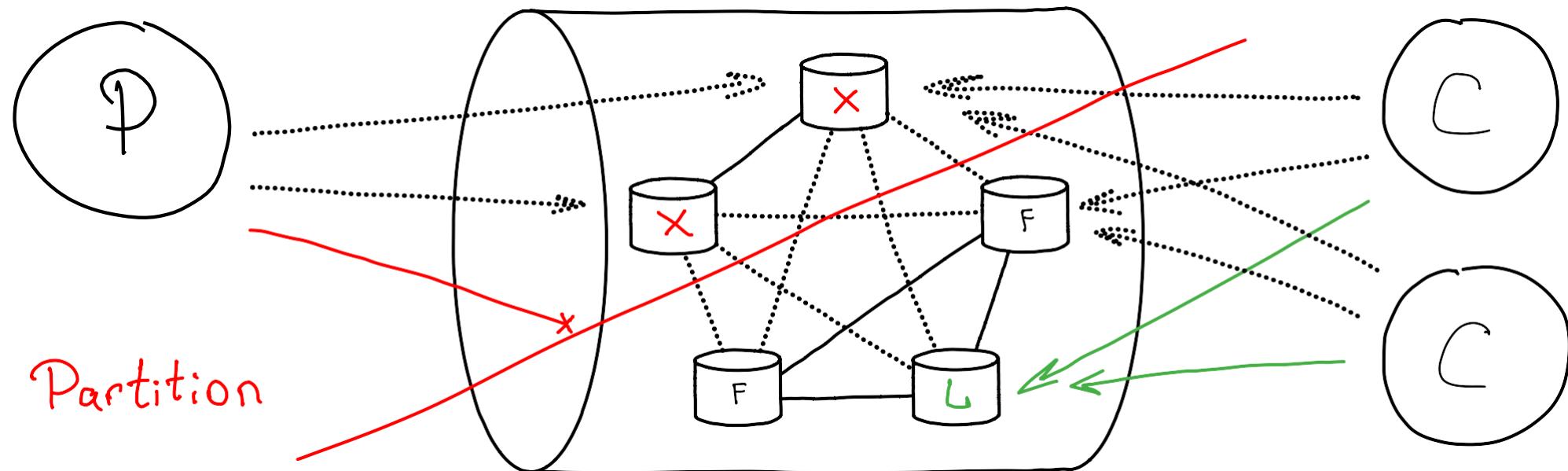
КВОРУМНЫЙ КЛАСТЕР ОЧЕРЕДИ



Гарантии: $X \approx 1$ ($X \geq 1$)

Доступность: ограничена
Надёжность: высокая

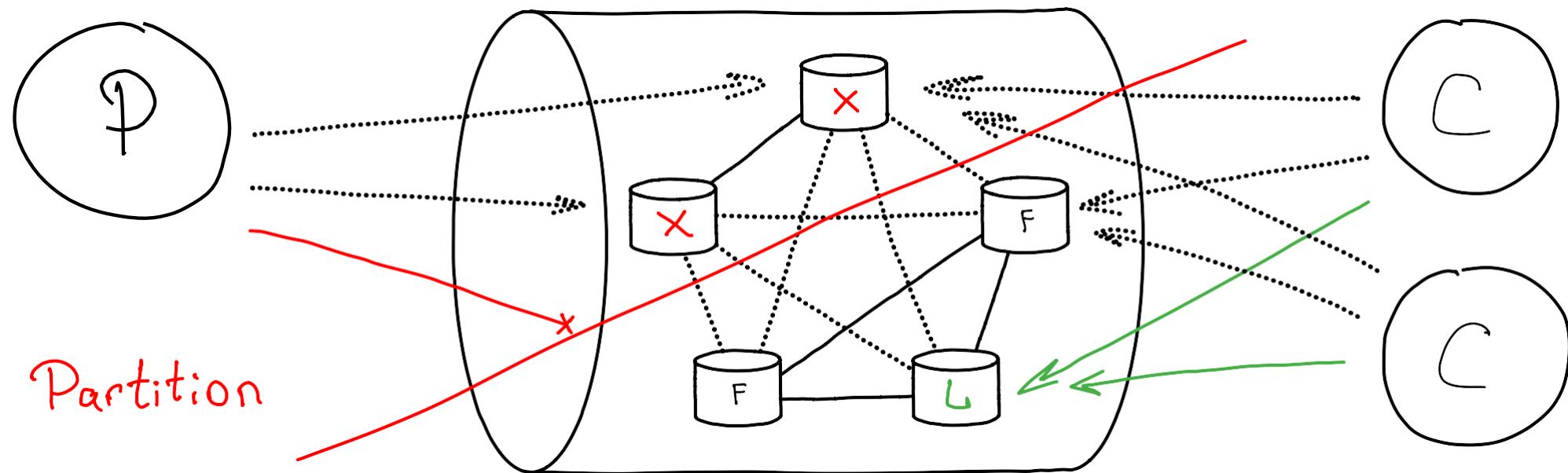
КВОРУМНЫЙ КЛАСТЕР ОЧЕРЕДИ



Гарантии: $X \approx 1$ ($X \geq 1$)

Доступность: ограничена
Надёжность: высокая

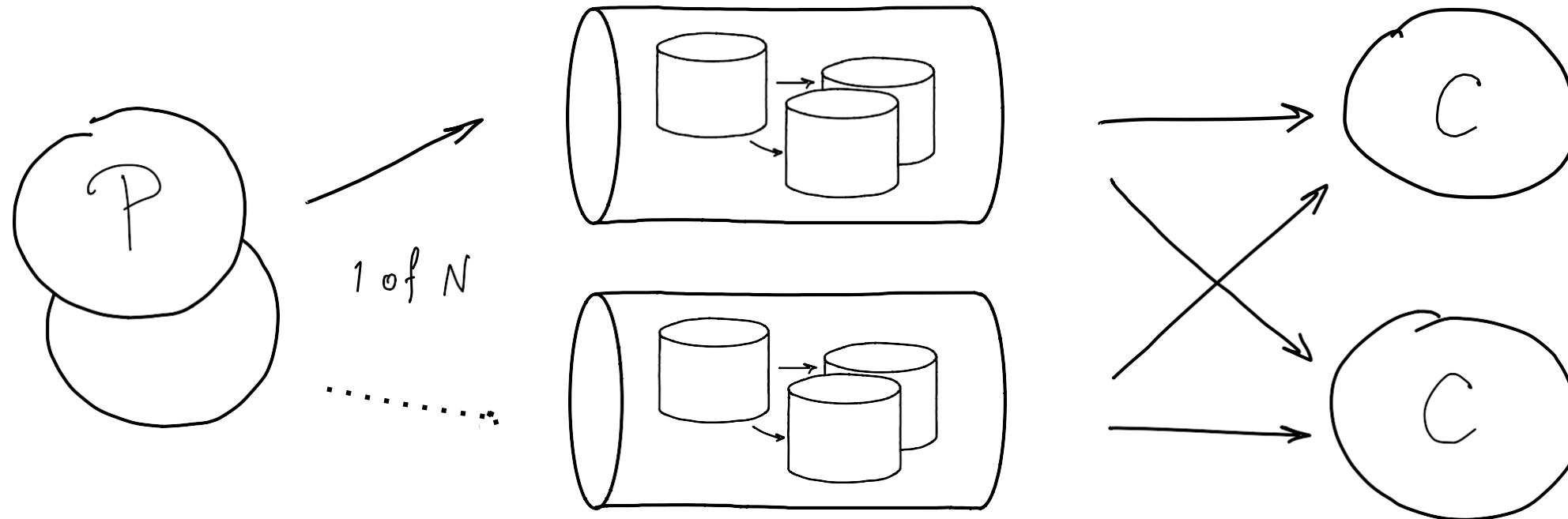
КВОРУМНЫЙ КЛАСТЕР ОЧЕРЕДИ



Масштабируемость: **нет**
Гарантии: $X \approx 1$ ($X \geq 1$)

Доступность: **ограничена**
Надёжность: **высокая**

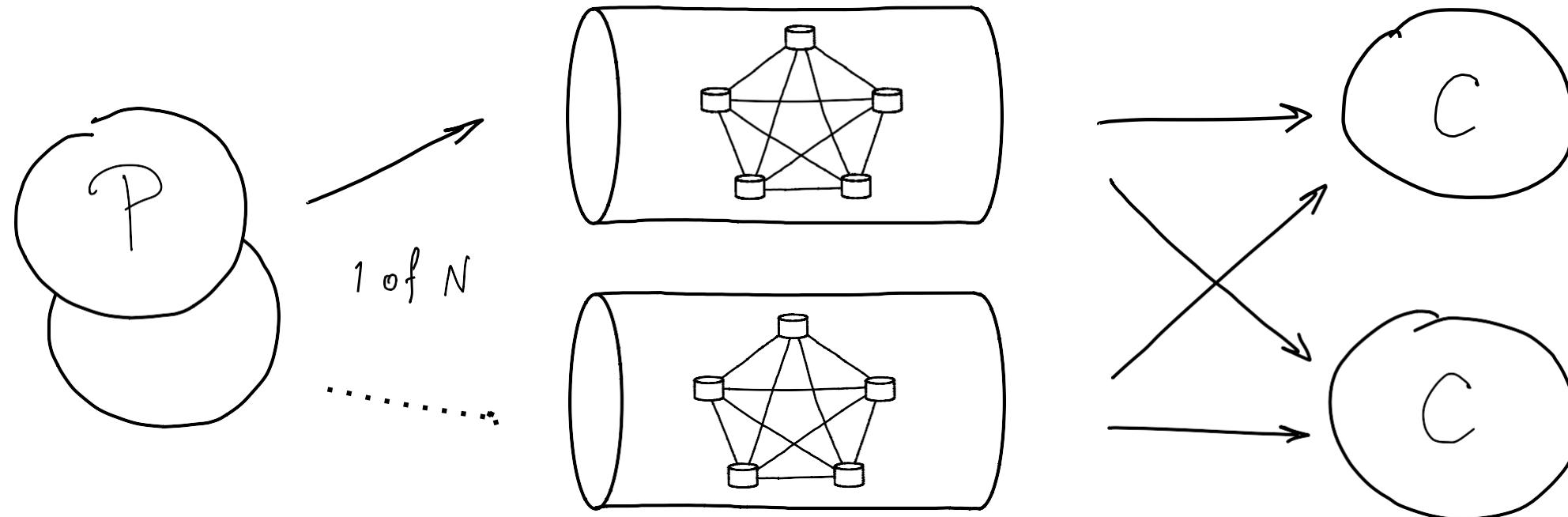
РЕПЛИЦИРОВАННЫЕ ОЧЕРЕДИ, 1 ИЗ N



Масштабируемость: **да**
Гарантии: **$X \approx 1$ ($X \geq 1$)**

Доступность: **высокая**
Надёжность: **высокая**

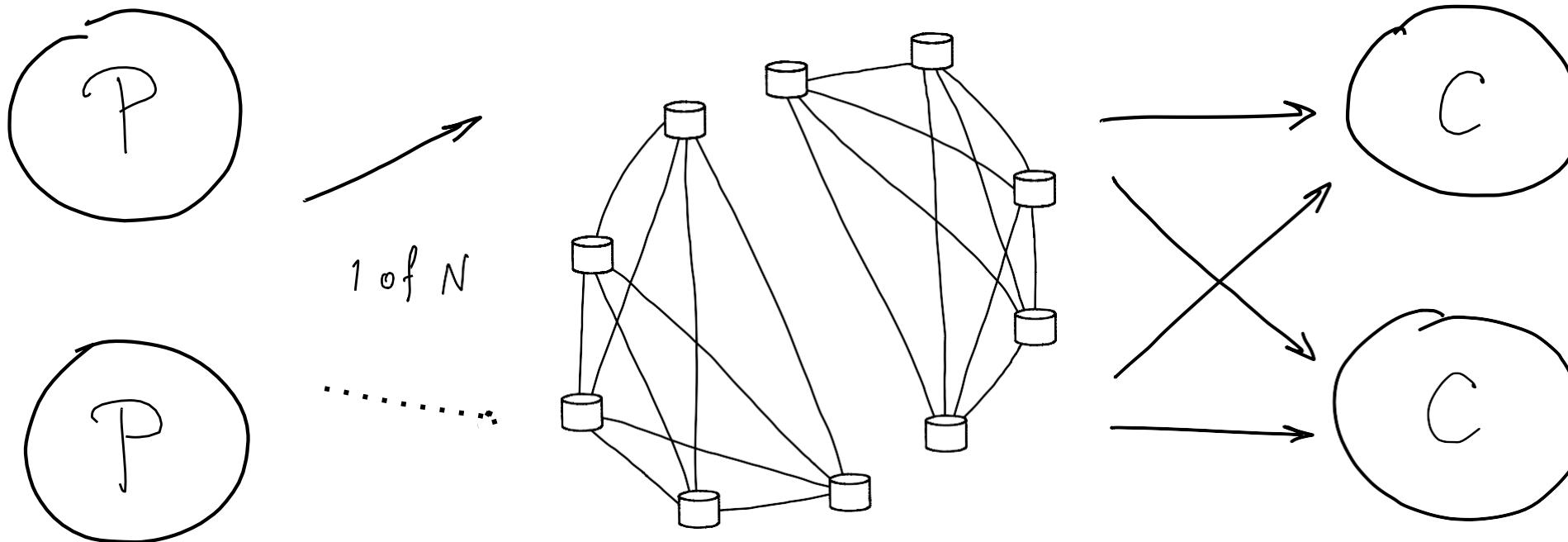
КВОРУМНЫЕ ОЧЕРЕДИ, 1 ИЗ N



Масштабируемость: **да**
Гарантии: **$X \approx 1$ ($X \geq 1$)**

Доступность: **высокая**
Надёжность: **высокая**

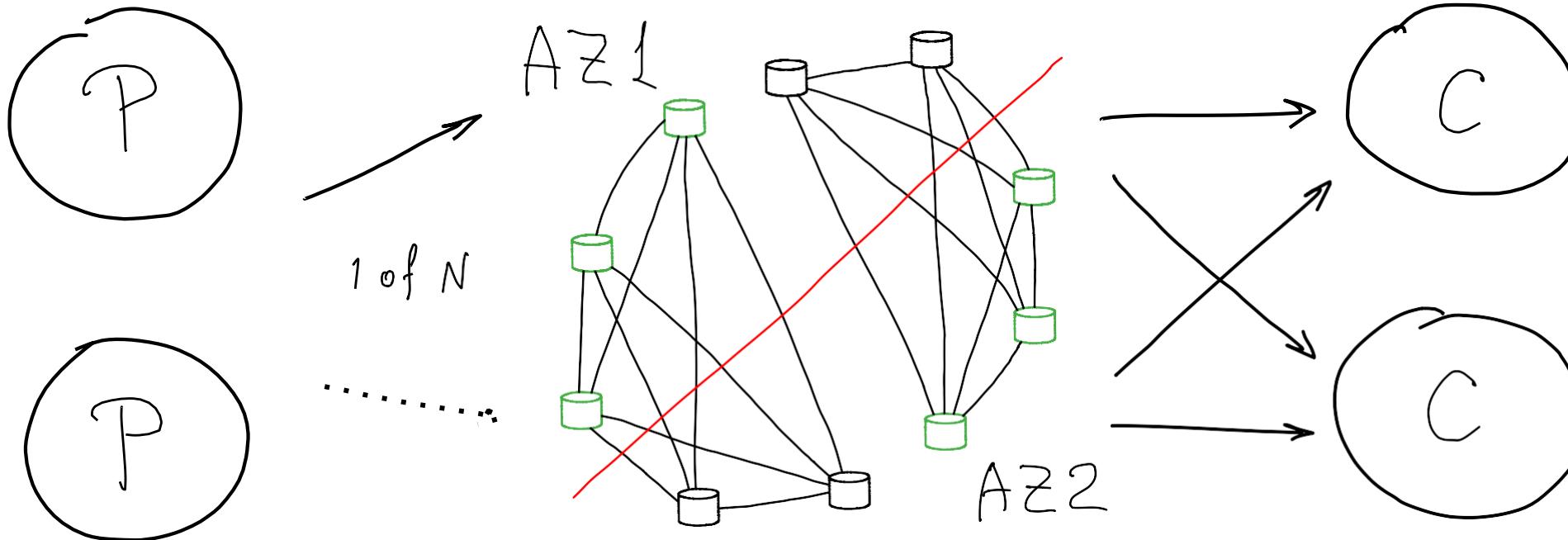
КВОРУМНЫЕ ОЧЕРЕДИ, 1 ИЗ N



Масштабируемость: **да**
Гарантии: **$X \approx 1$ ($X \geq 1$)**

Доступность: **высокая**
Надёжность: **высокая**

КВОРУМНЫЕ ОЧЕРЕДИ, 1 ИЗ N



Масштабируемость: **да**
Гарантии: **$X \approx 1$ ($X \geq 1$)**

Доступность: **высокая**
Надёжность: **высокая**

ПРОТОКОЛЫ ОЧЕРЕДЕЙ И ОГРАНИЧЕНИЯ

**Состояние задачи
в соединении**

Без состояния (HTTP/REST/SQS)

ПРОТОКОЛЫ ОЧЕРЕДЕЙ И ОГРАНИЧЕНИЯ

Состояние задачи в соединении

- Низкая задержка
- Мгновенный возврат
- Сложно масштабировать
- Жизненный цикл

Без состояния (HTTP/REST/SQS)

ПРОТОКОЛЫ ОЧЕРЕДЕЙ И ОГРАНИЧЕНИЯ

Состояние задачи в соединении

- Низкая задержка
- Мгновенный возврат
- Сложно масштабировать
- Жизненный цикл

Без состояния (HTTP/REST/SQS)

- Масштабирование
- HTTP-балансировка
- Нужен автозврат

МОНИТОРИНГ И ЭКСПЛУАТАЦИЯ

- Размеры очереди
 - Очередь всегда ограничена

МОНИТОРИНГ И ЭКСПЛУАТАЦИЯ

- Размеры очереди
 - Очередь всегда ограничена
- Время
 - Полная обработка сообщения (QoS)
 - Время исполнения

МОНИТОРИНГ И ЭКСПЛУАТАЦИЯ

- Размеры очереди
 - Очередь всегда ограничена
- Время
 - Полная обработка сообщения (QoS)
 - Время исполнения
- Количество повторов и потерь/отказов

МОНИТОРИНГ И ЭКСПЛУАТАЦИЯ

- Размеры очереди
 - Очередь всегда ограничена
- Время
 - Полная обработка сообщения (QoS)
 - Время исполнения
- Количество повторов и потерь/отказов
- Поток сообщений

МОНИТОРИНГ И ЭКСПЛУАТАЦИЯ

- Размеры очереди
 - Очередь всегда ограничена
- Время
 - Полная обработка сообщения (QoS)
 - Время исполнения
- Количество повторов и потерь/отказов
- Поток сообщений
- **Логируйте сообщения!**

ЭКСПЛУАТАЦИЯ: ПЛАНИРУЙТЕ ОТКАЗ

- Настраивайте политики отказа
 - Перестаньте принимать новое
 - Уничтожьте старое
 - «Спасайте» выживших

ЭКСПЛУАТАЦИЯ: ПЛАНИРУЙТЕ ОТКАЗ

- Настраивайте политики отказа
 - Перестаньте принимать новое
 - Уничтожьте старое
 - «Спасайте» выживших
- Запланируйте падение
 - Для того, чтобы подняться

ЭКСПЛУАТАЦИЯ: ПЛАНИРУЙТЕ ОТКАЗ

- Настраивайте политики отказа
 - Перестаньте принимать новое
 - Уничтожьте старое
 - «Спасайте» выживших
- Запланируйте падение
 - Для того, чтобы подняться

Не тот велик, кто
никогда не падал,
а тот велик — кто падал и
вставал!



ЧТО ЖЕ ВЗЯТЬ?

- Тolerантность сервиса к потерям
- Организация передачи сообщений
- Высокая пропускная способность, масштабируемость

ЧТО ЖЕ ВЗЯТЬ?

- Тolerантность сервиса к потерям
 - Организация передачи сообщений
 - Высокая пропускная способность, масштабируемость
-
- **NATS**
 - NSQ
 - ZeroMQ

ЧТО ЖЕ ВЗЯТЬ?

- Быстро попробовать
- Соединить микросервисы или k8s
- Сервис работает в облаке

ЧТО ЖЕ ВЗЯТЬ?

- Быстро попробовать
- Соединить микросервисы или k8s
- Сервис работает в облаке
- **SQS** (Simple Queue Service)
 - Amazon, Mail.ru Cloud, Yandex, ...
- CloudAMQP
- Простые брокеры: RabbitMQ, NATS
(следите за надёжностью)

ЧТО ЖЕ ВЗЯТЬ?

- Организовать стриминговую архитектуру
- Нужна высокая сохранность
- Требуется строгий FIFO

ЧТО ЖЕ ВЗЯТЬ?

- Организовать стриминговую архитектуру
 - Нужна высокая сохранность
 - Требуется строгий FIFO
-
- Apache **Kafka**
 - NATS JetStream
 - Tarantool Enterprise

ЧТО ЖЕ ВЗЯТЬ?

- Сложные сценарии очередей
- Отложенные задачи, перепостановка
- Произвольные топологии, собственные алгоритмы
- Зависимые сценарии

ЧТО ЖЕ ВЗЯТЬ?

- Сложные сценарии очередей
 - Отложенные задачи, перепостановка
 - Произвольные топологии, собственные алгоритмы
 - Зависимые сценарии
-
- **RabbitMQ**
 - Tarantool Queue / Tarantool

СПАСИБО!

Виденин Сергей

@videninserg

