Skillbox

# Распределенная архитектура

## Space-Based Architecture Style

**Андрей Гордиенков**

Solution Architect
ABAX

Skillbox

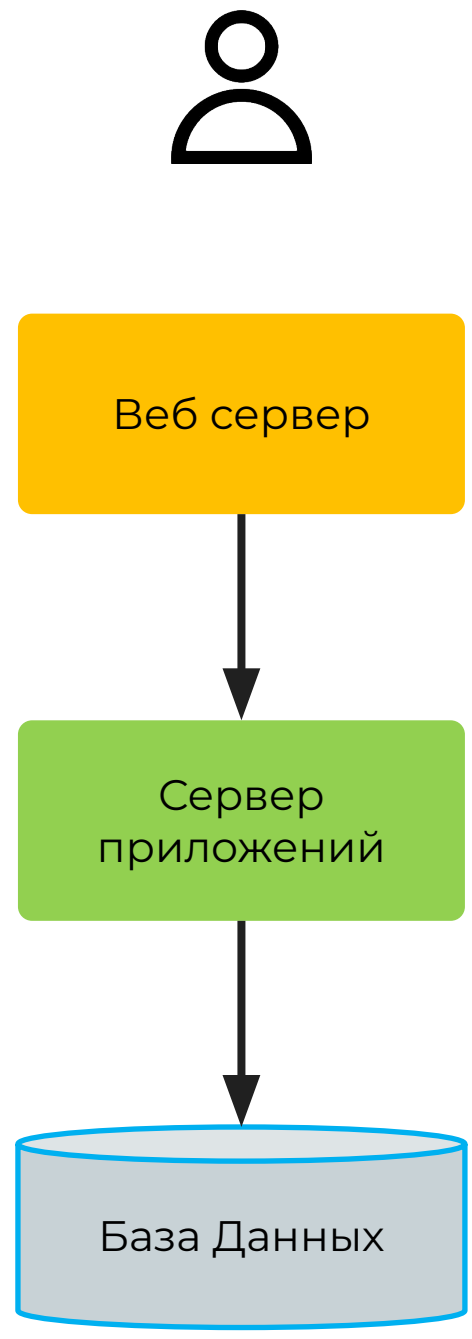
# В прошлом видео

- Рассмотрели архитектурный стиль Service-Oriented, применимость, основные сложности.

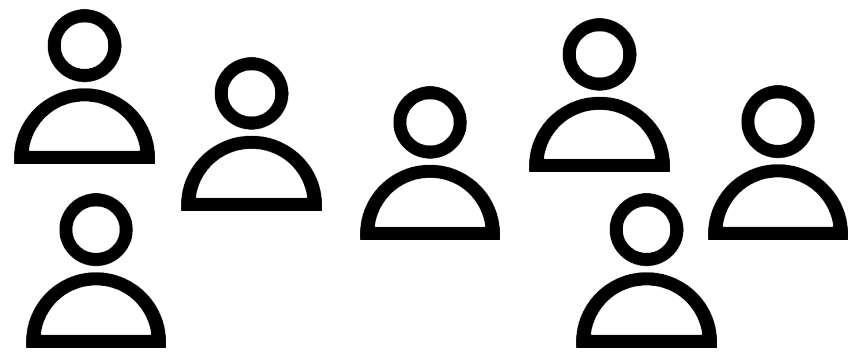
# В этом видео

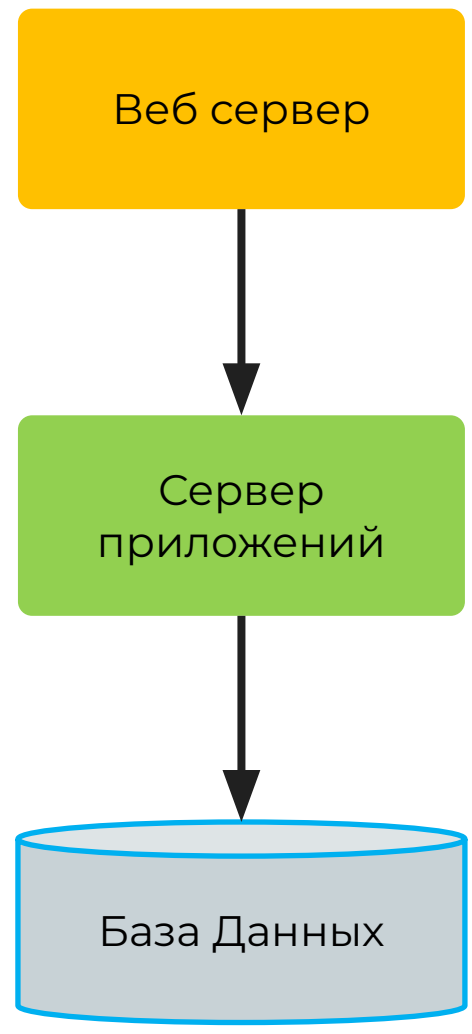
- Что такое Spaced-Based Architecture
- Понятия и принципы
- Отличительные черты

# Постановка проблемы

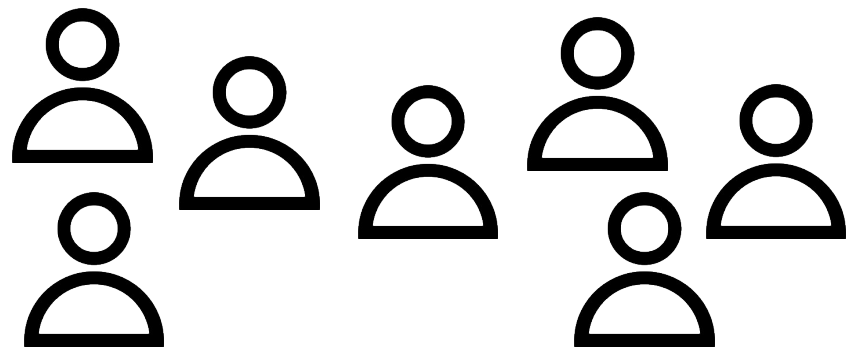


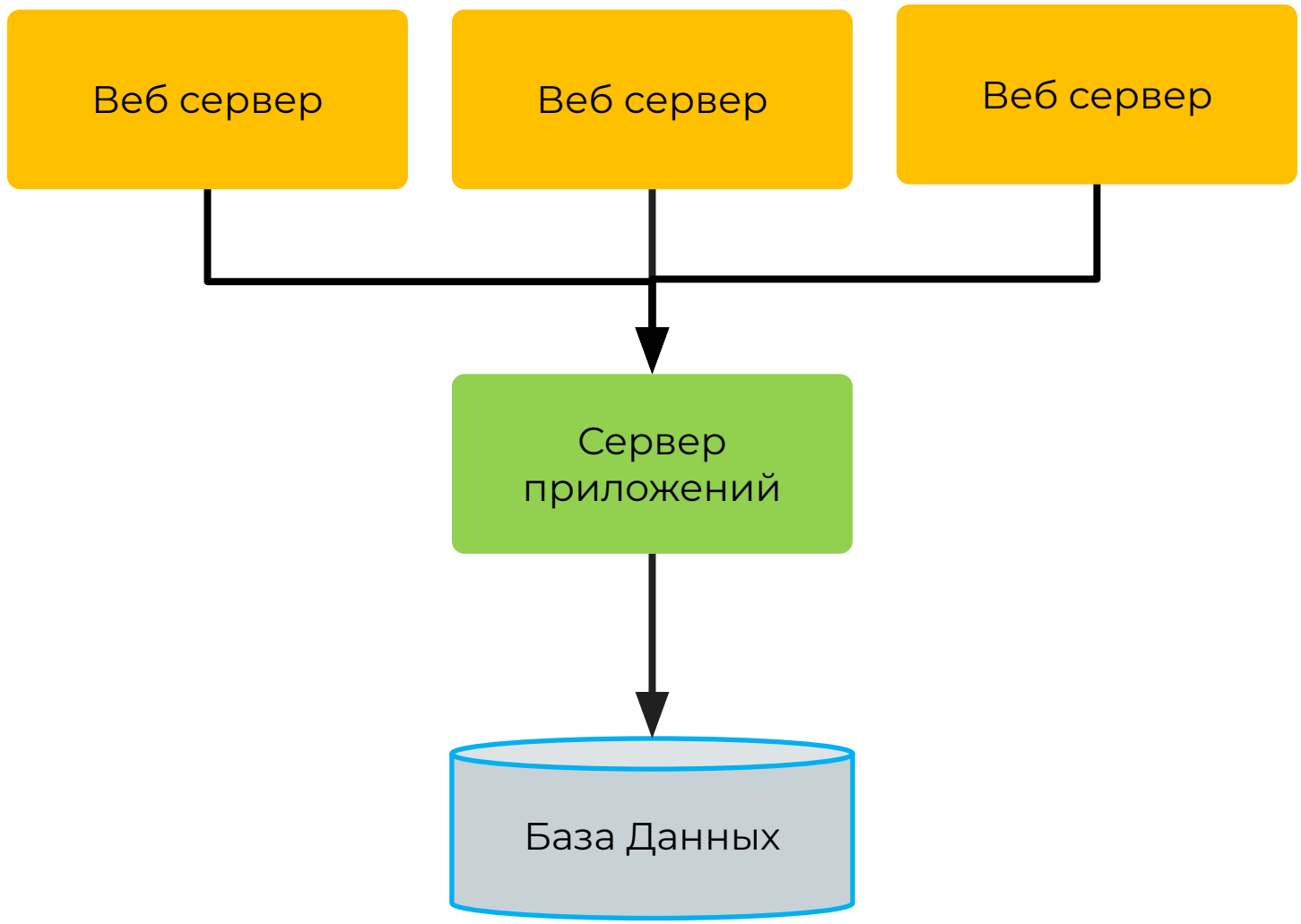
# Постановка проблемы



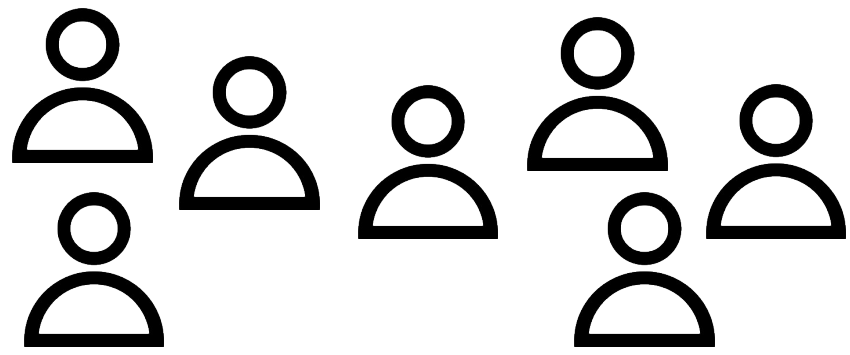


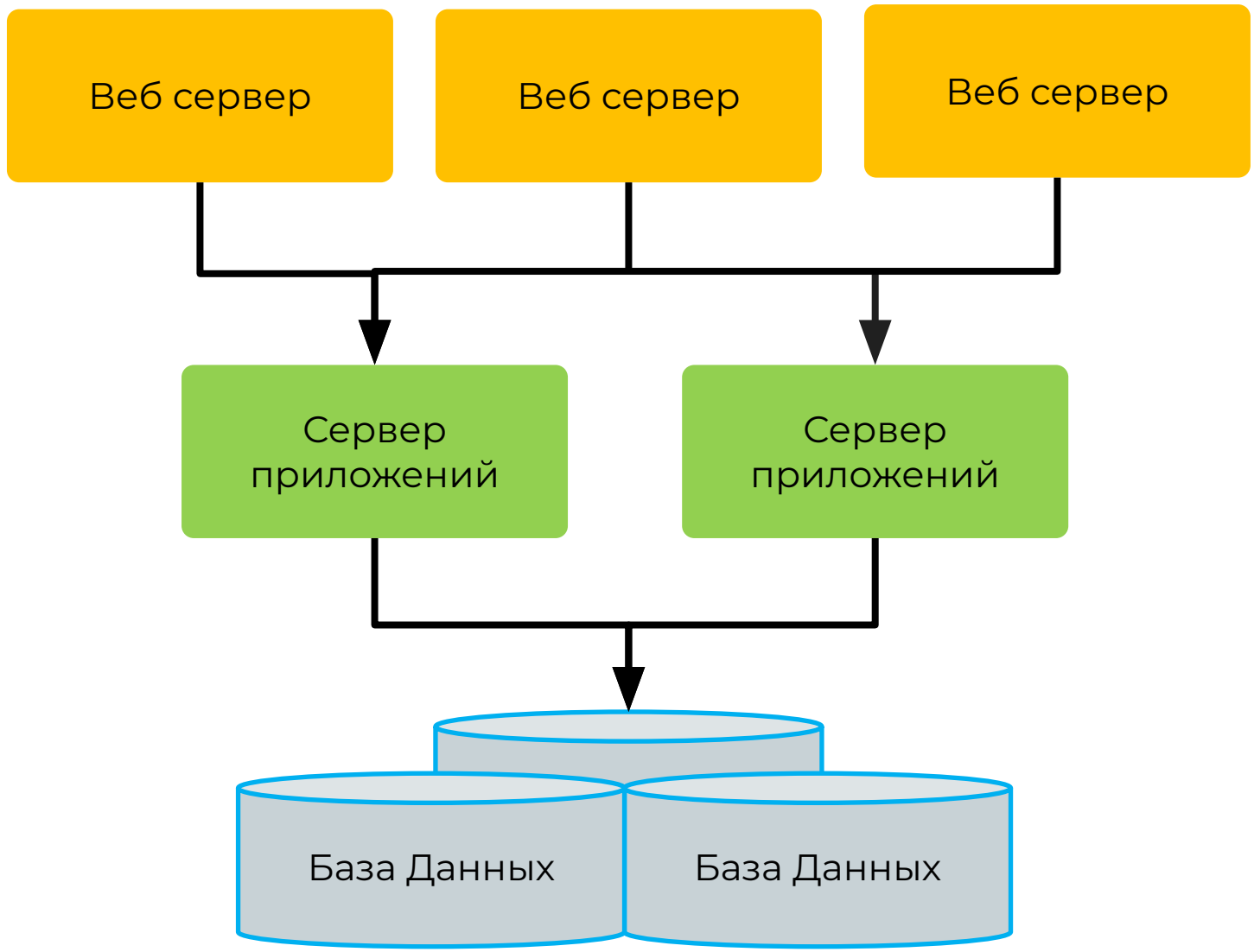
# Постановка проблемы



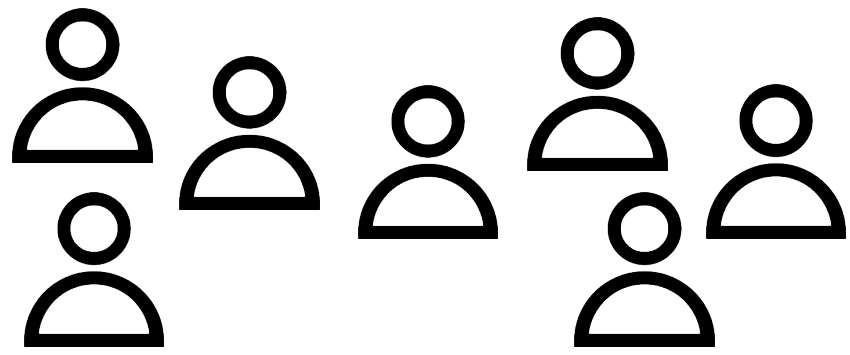


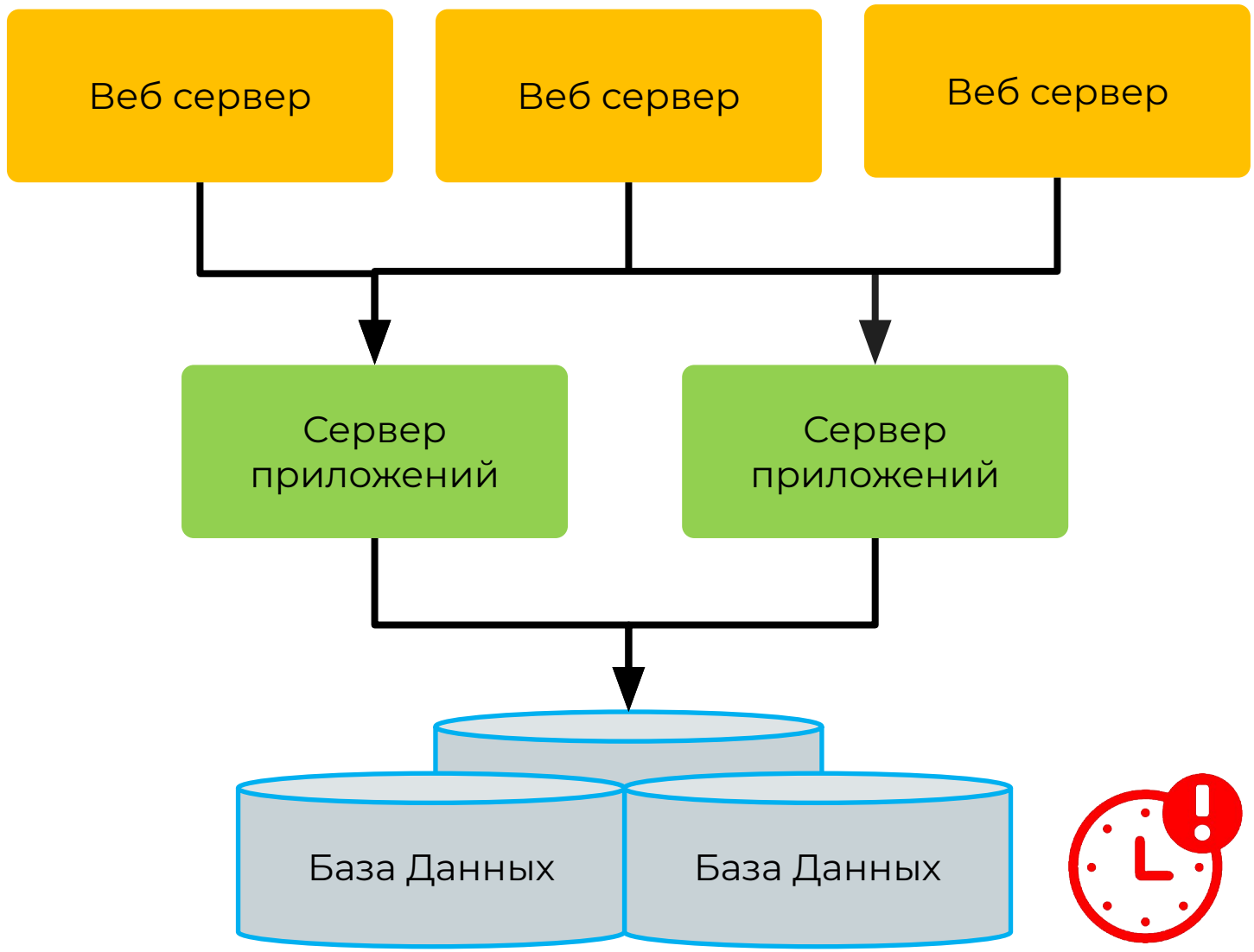
# Постановка проблемы





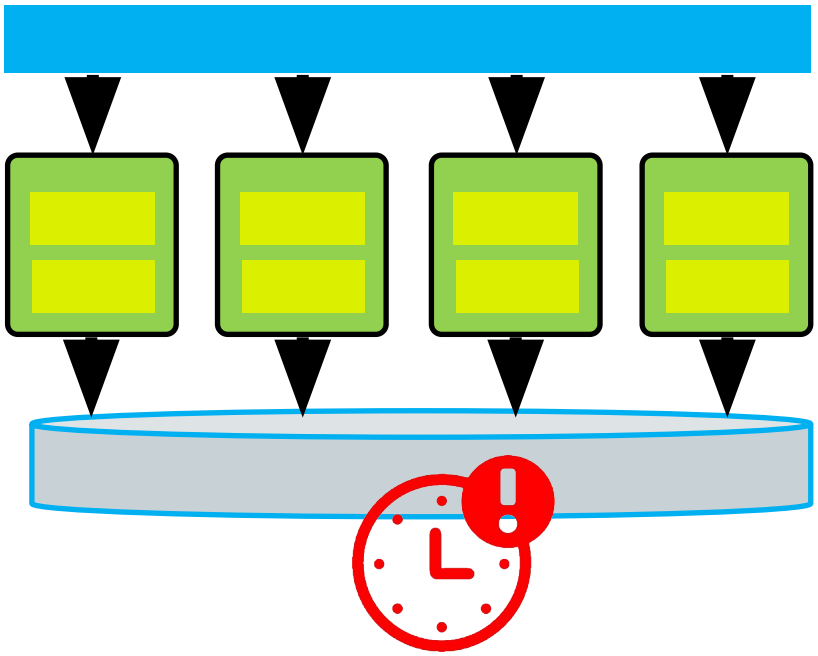
# Постановка проблемы



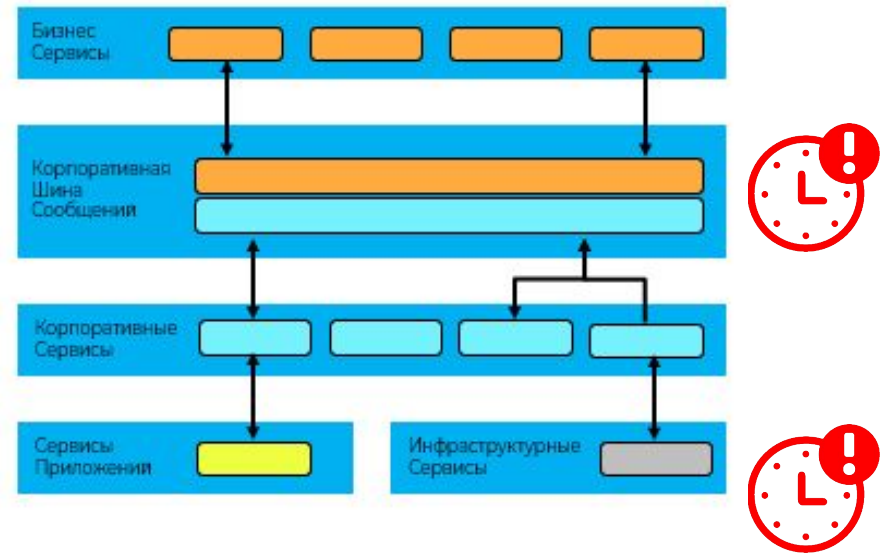


# Сложности сохраняются

Service-Based Architecture



Service-Oriented Architecture



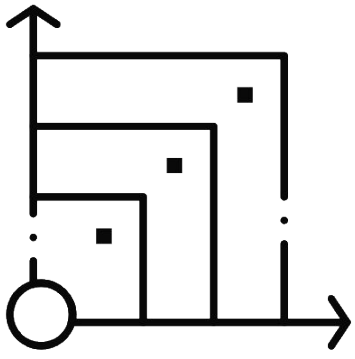

Дорогая синхронизация данных

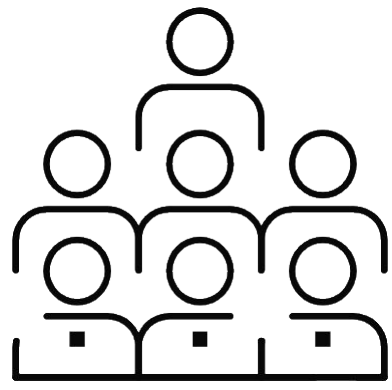
# Решаемые проблемы

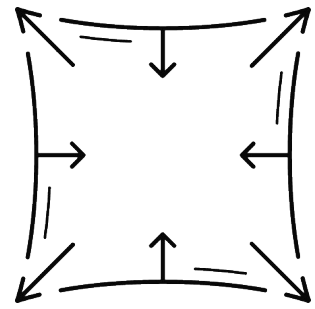
- Экстремальная и переменная масштабируемость любых объемов

- Высокая, непредсказуемая, одновременная пользовательская нагрузка

- Эластичность



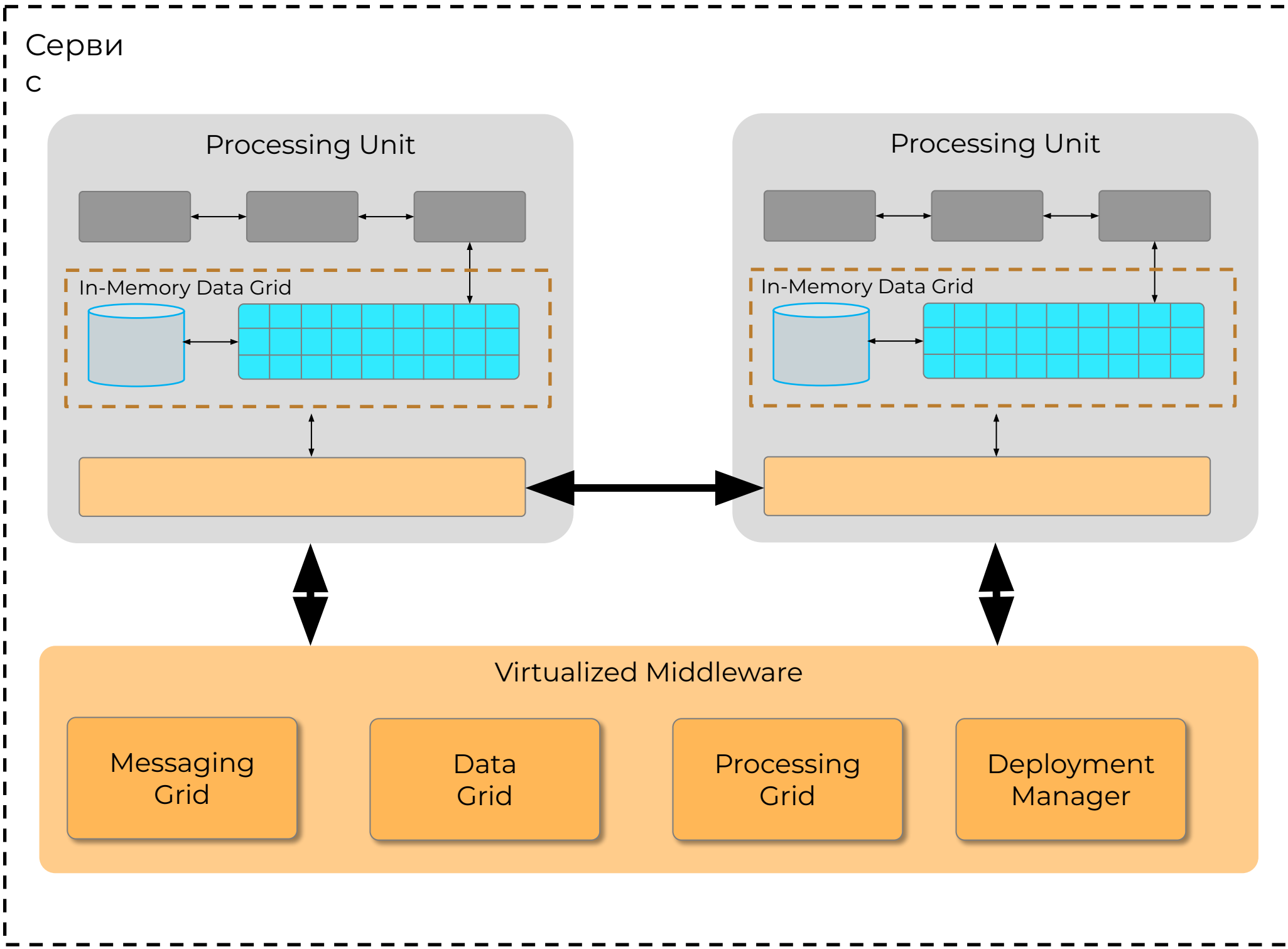




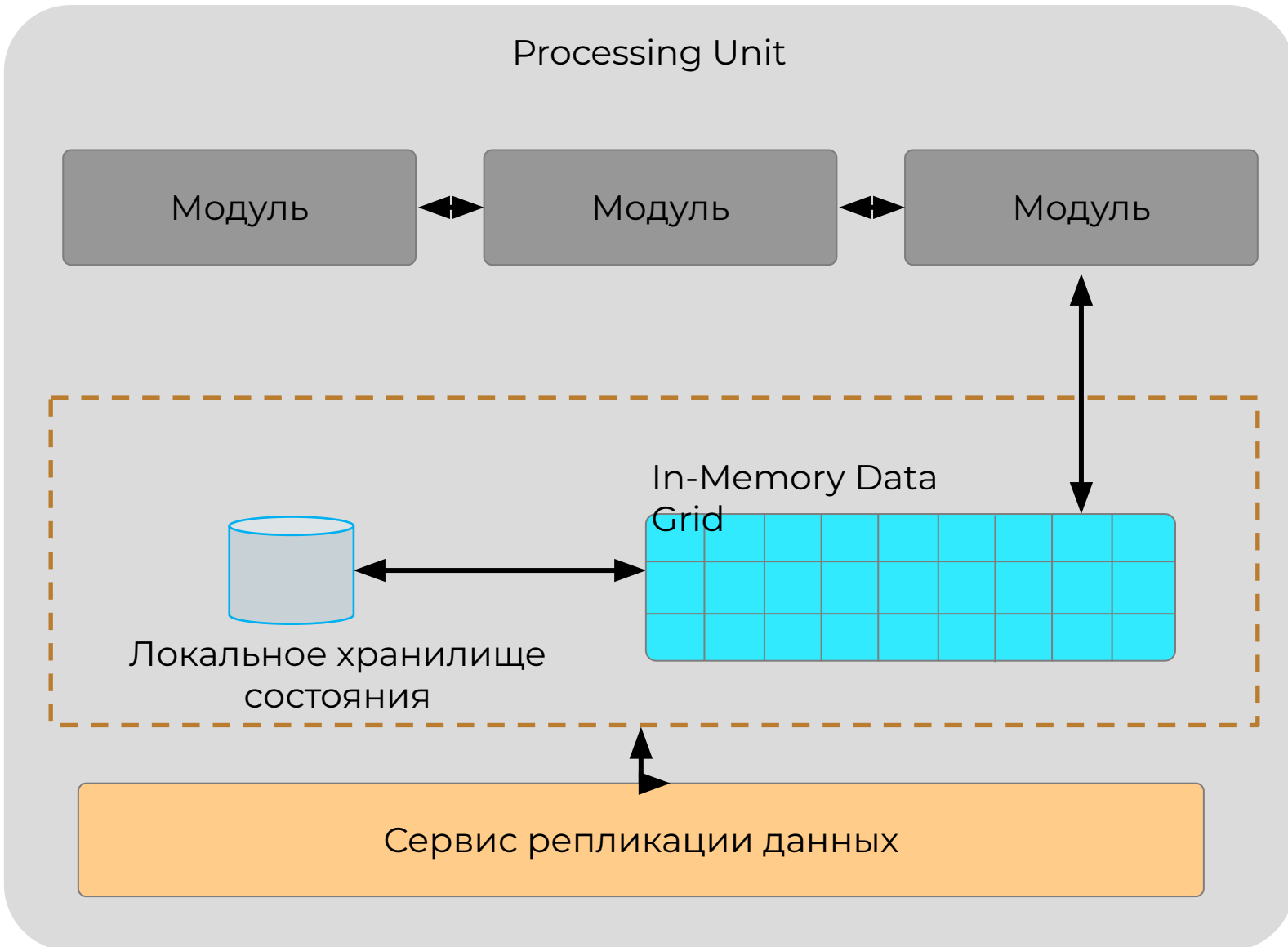

Как сделать монолит правильно
Микросервисы

Skillbox

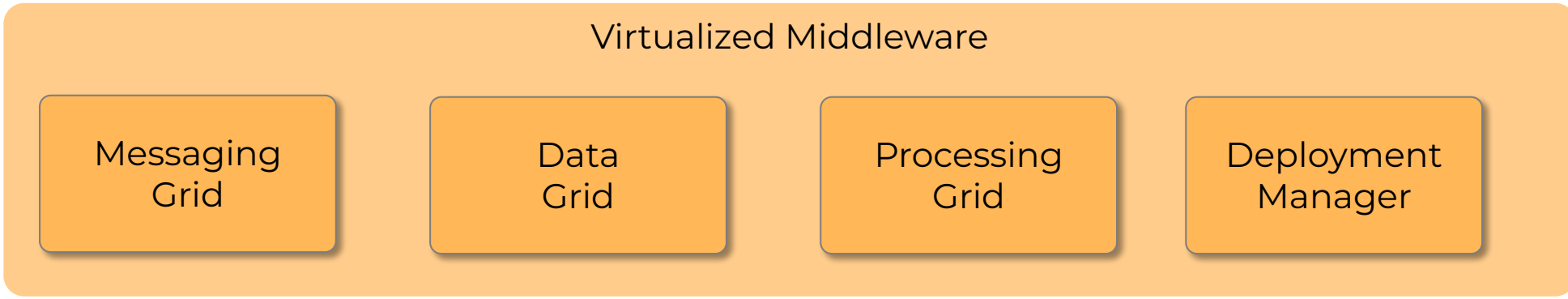
# Топология



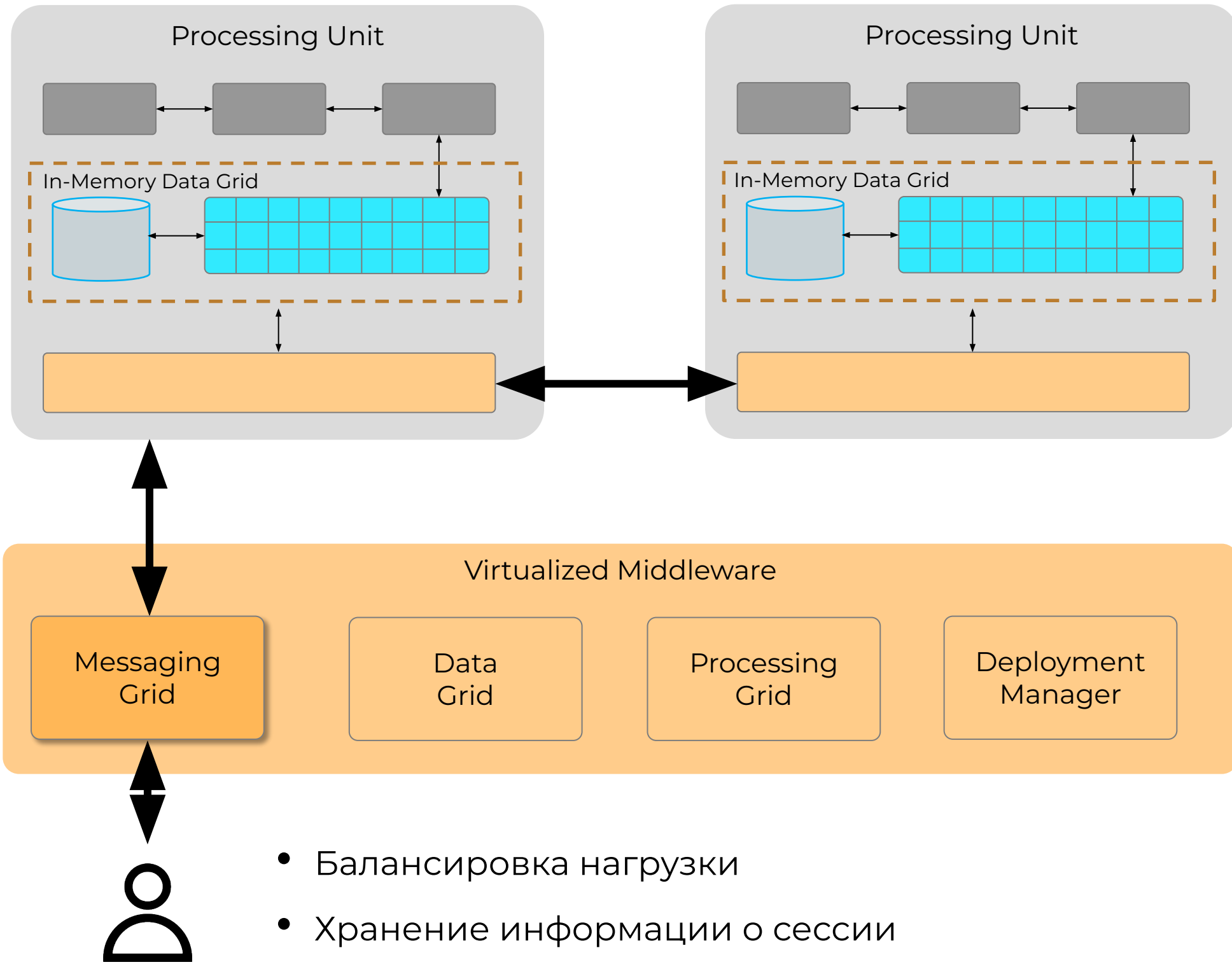
# Processing Unit



# Обслуживающий слой

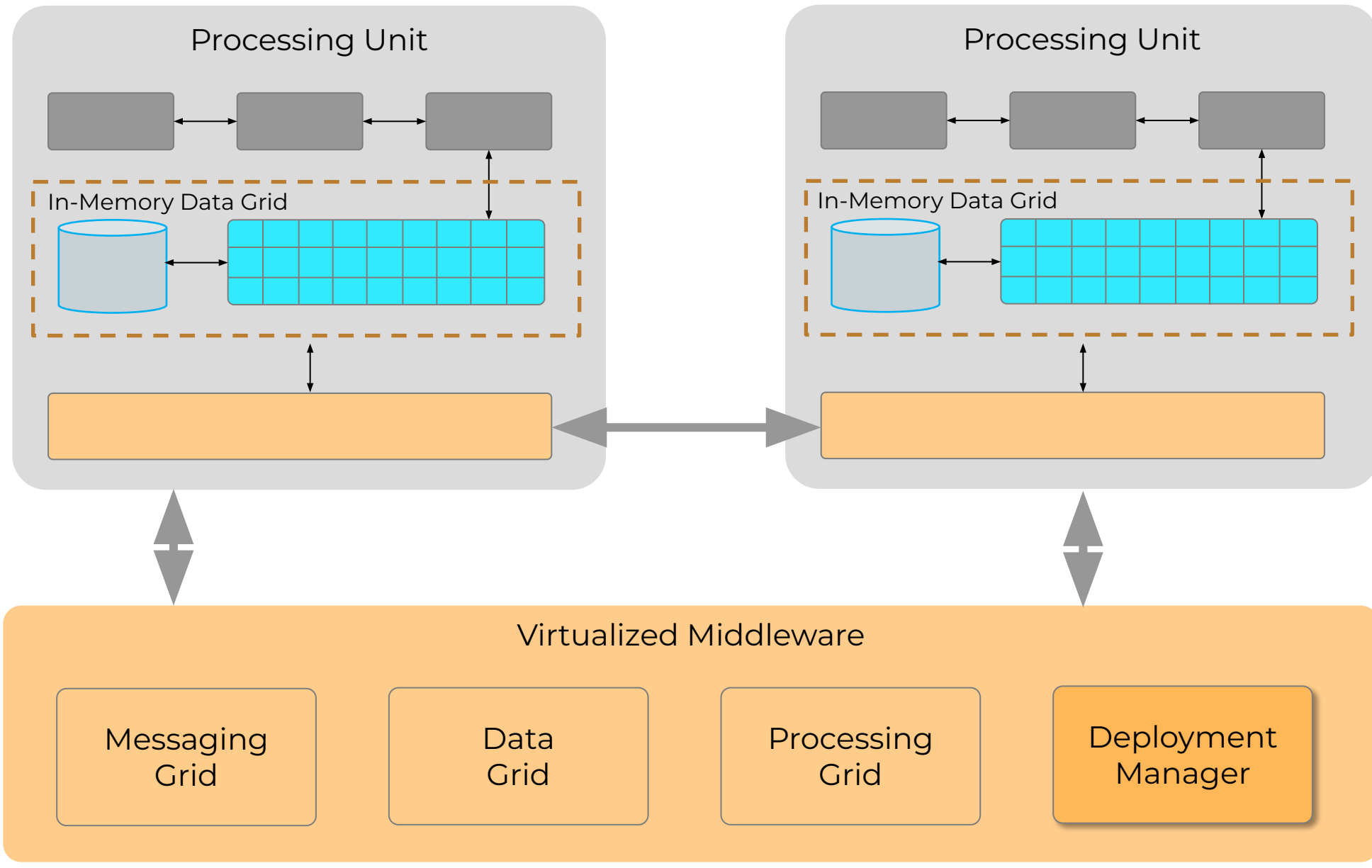


# Messaging Grid



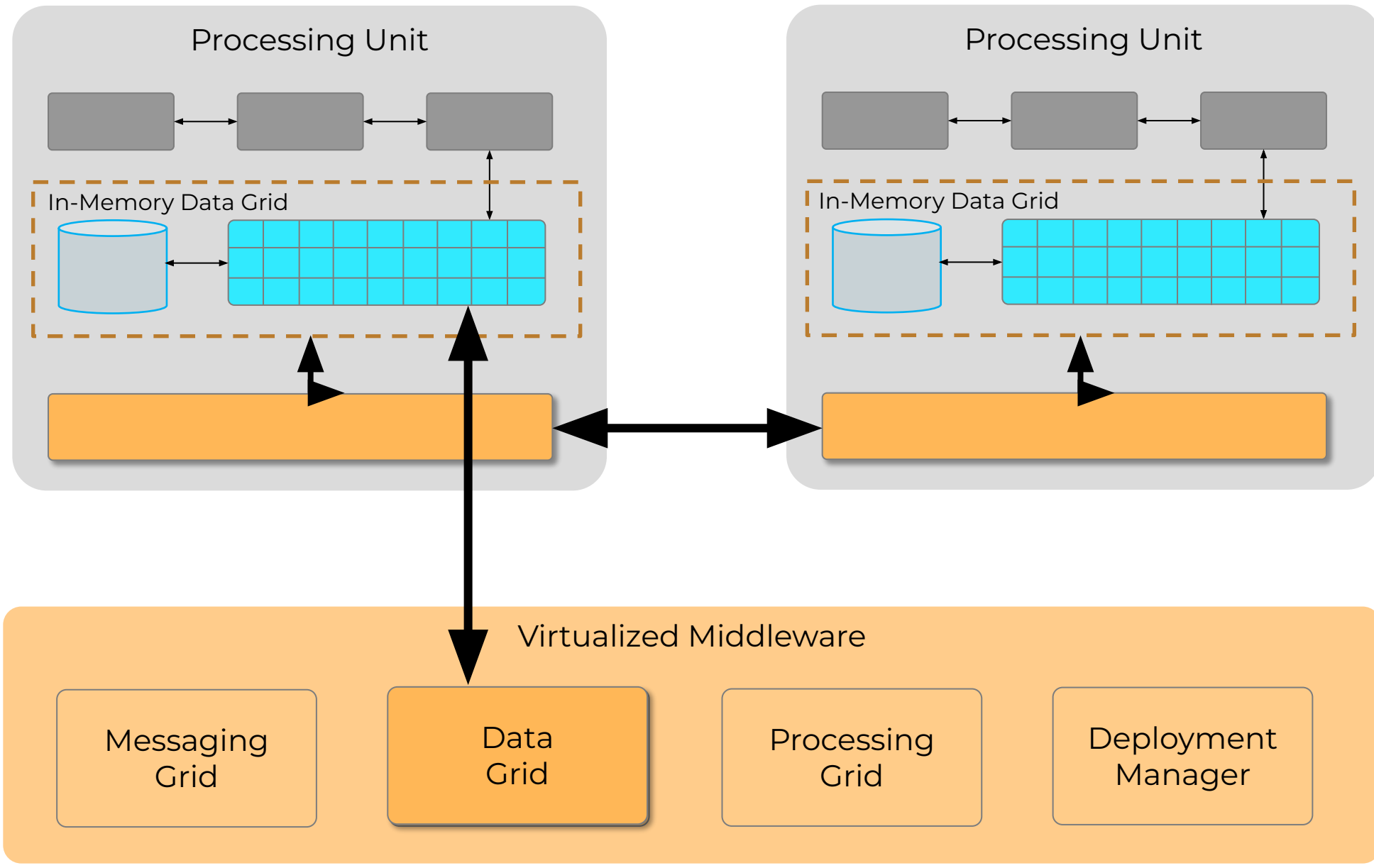

- Балансировка нагрузки
- Хранение информации о сессии

# Deployment Manager



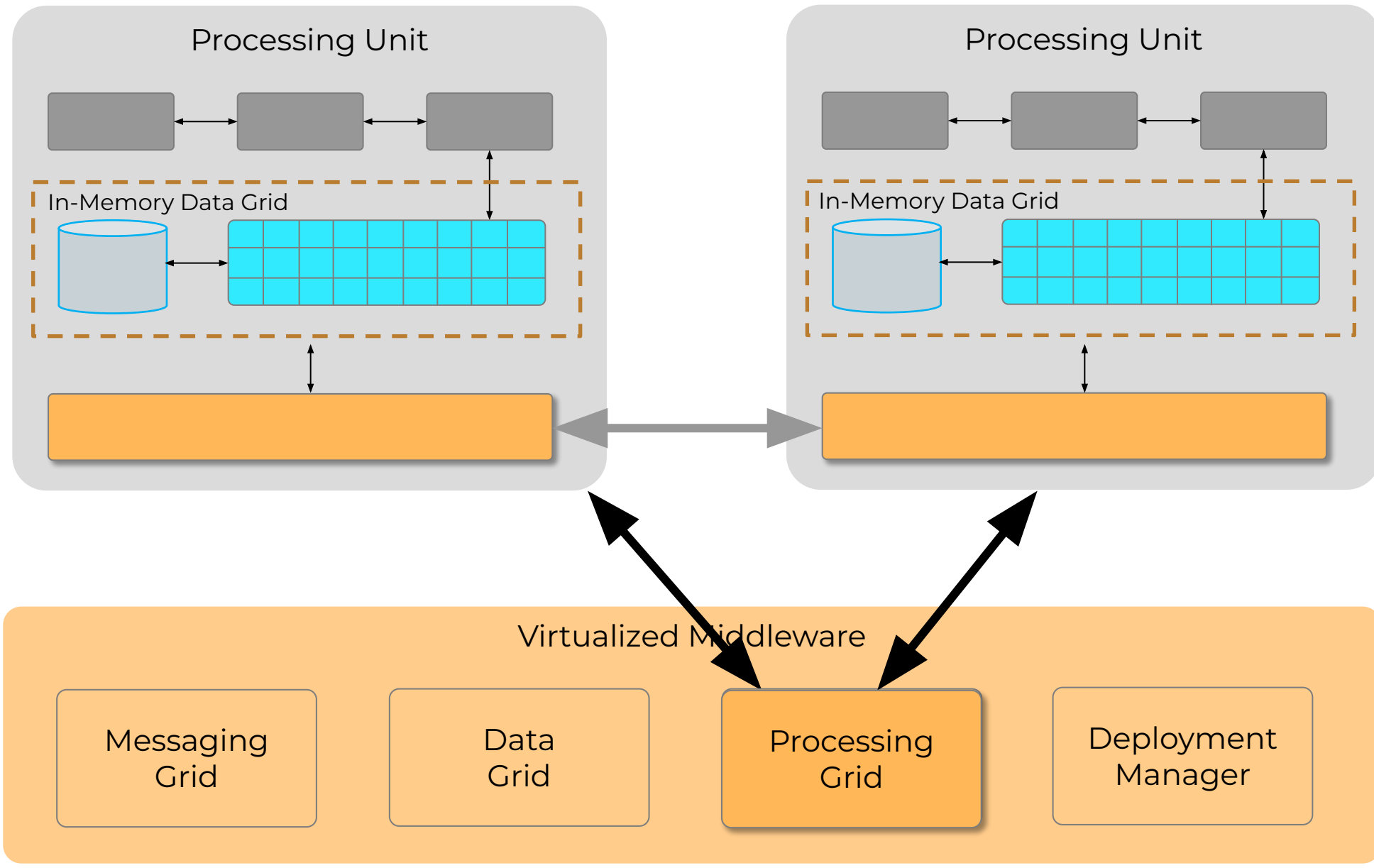

- Управление количеством юнитов в зависимости от нагрузки

# Data Grid



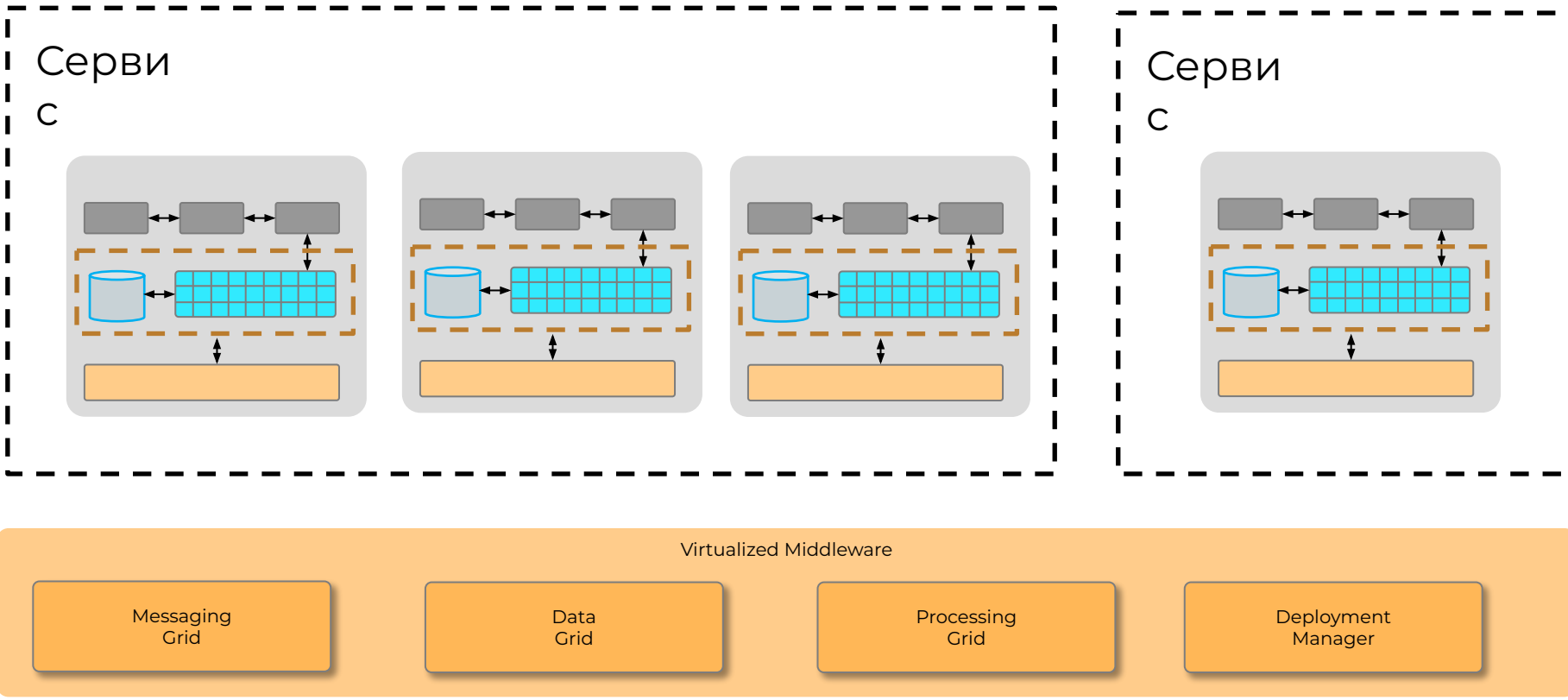

- Быстрая репликация данных

# Processing Grid

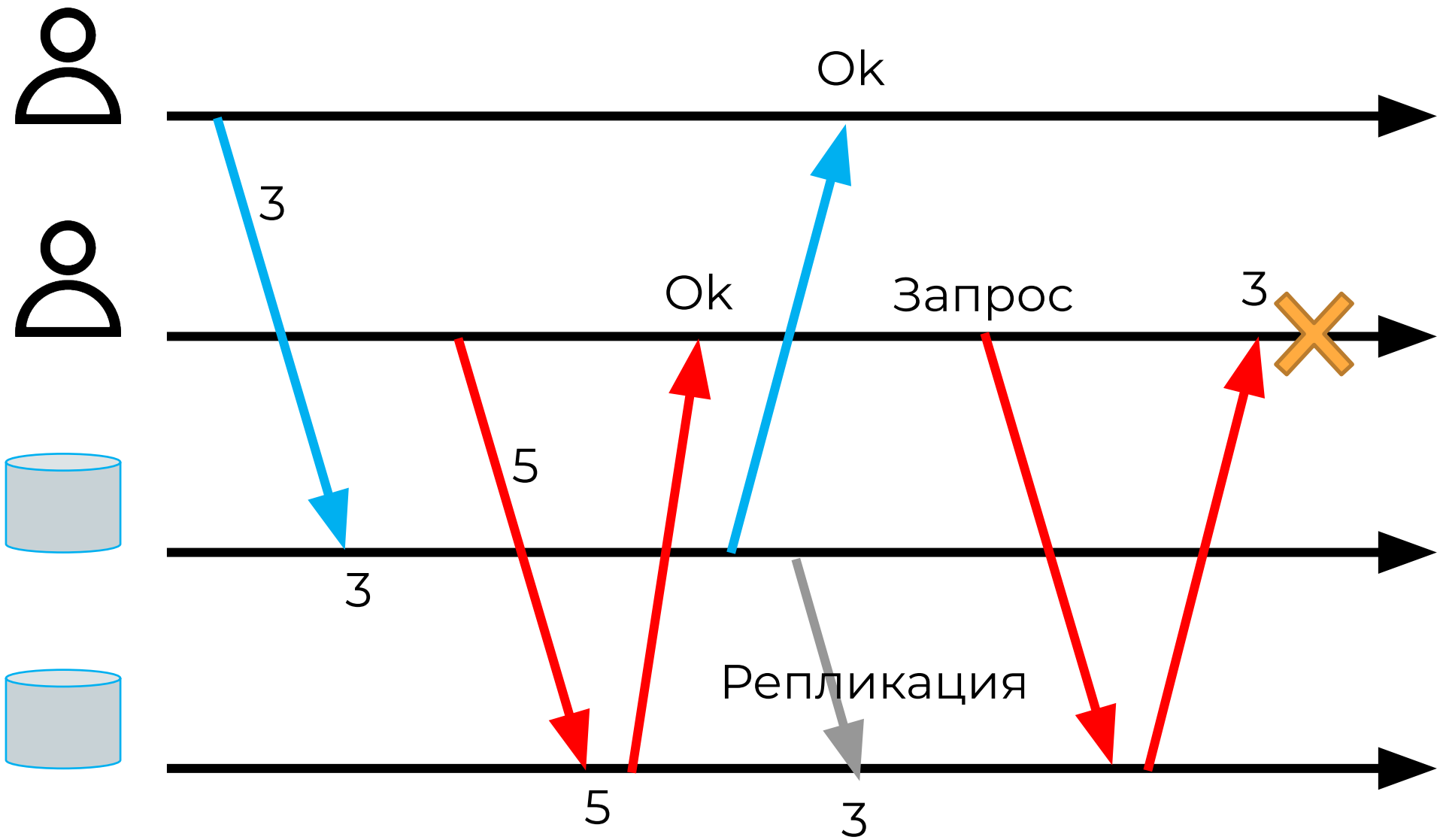



- Медиатор запросов, если необходимо

# Process Unit и сервисы



# Коллизии репликации



# Реплицированный кэш

Компенсационные меры разрешения конфликтных ситуаций, используемые в некоторых реализациях IMDG (In Memory Data Grid):

- обогащение данных доп. информацией:
  1. временной меткой (timestamp);
  2. вектором версий (version vector);

- хранение данных в виде цепочки событий (создания и обновления).

# Реплицированный кэш

$$\text{частота коллизий} \approx IN * \frac{UR^2}{CS} * RL$$

где:

- *IN* (instance number) – количество экземпляров сервиса, использующих одноименный cache;

- *UR* (update rate) – частота обновления данных (т.е. показатель того, насколько часто меняются данные);

- *CS* (cache size) – размер cache, т.е. строк (или <key, value> пар) данных, присутствующих в локальном экземпляре cache;

- *RL* (replication latency) – задержка репликации.

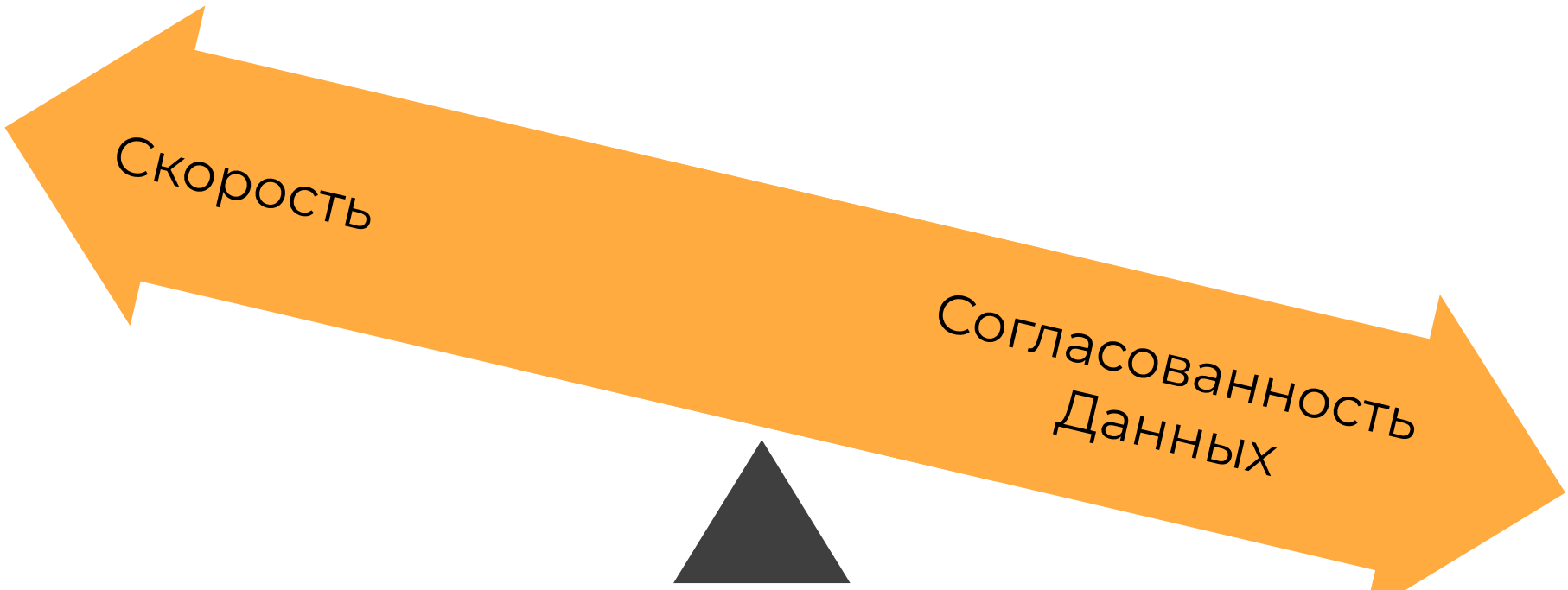
# Пример расчета

| | |
|---|---|
| Скорость обновлений | 20 в секунду |
| Количество юнитов | 5 |
| Размер кэша | 50 000 строк |
| Задержка репликации | 100 мс |

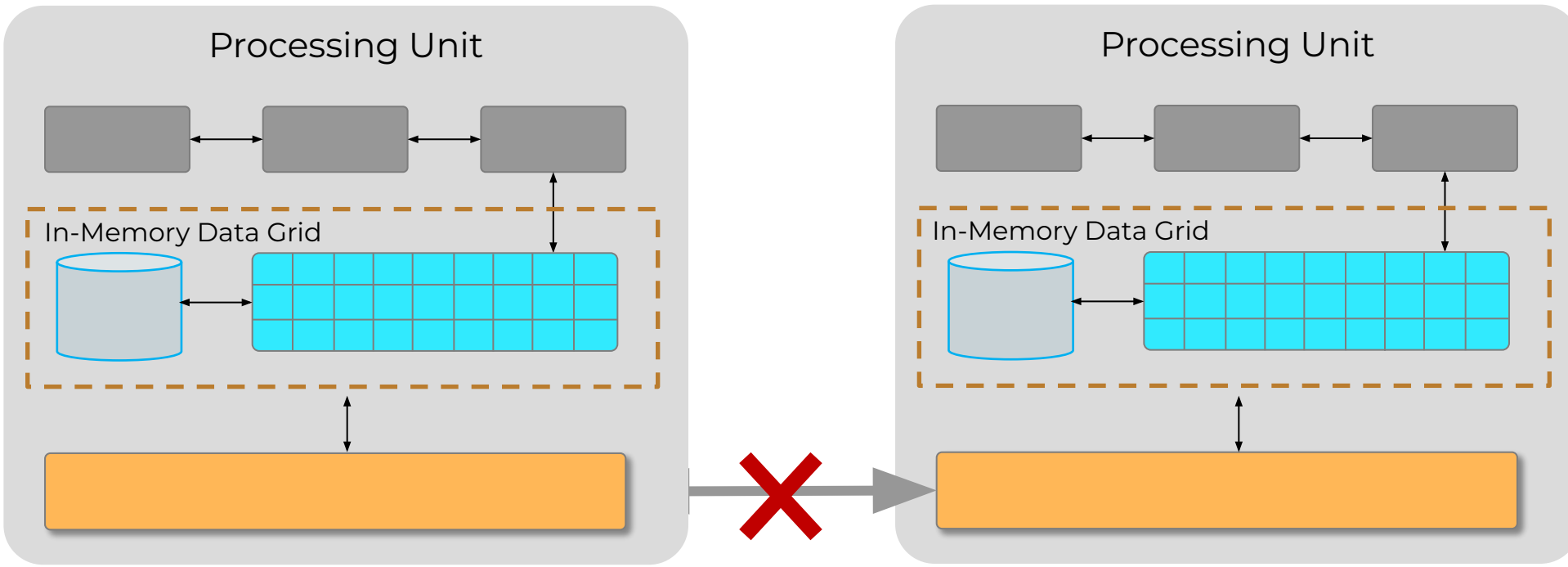
| | |
|---|---|
| Обновления | 72 000 в час |
| Частота коллизий | 14,4 в час |
| Процент | 0,02% |

Skillbox

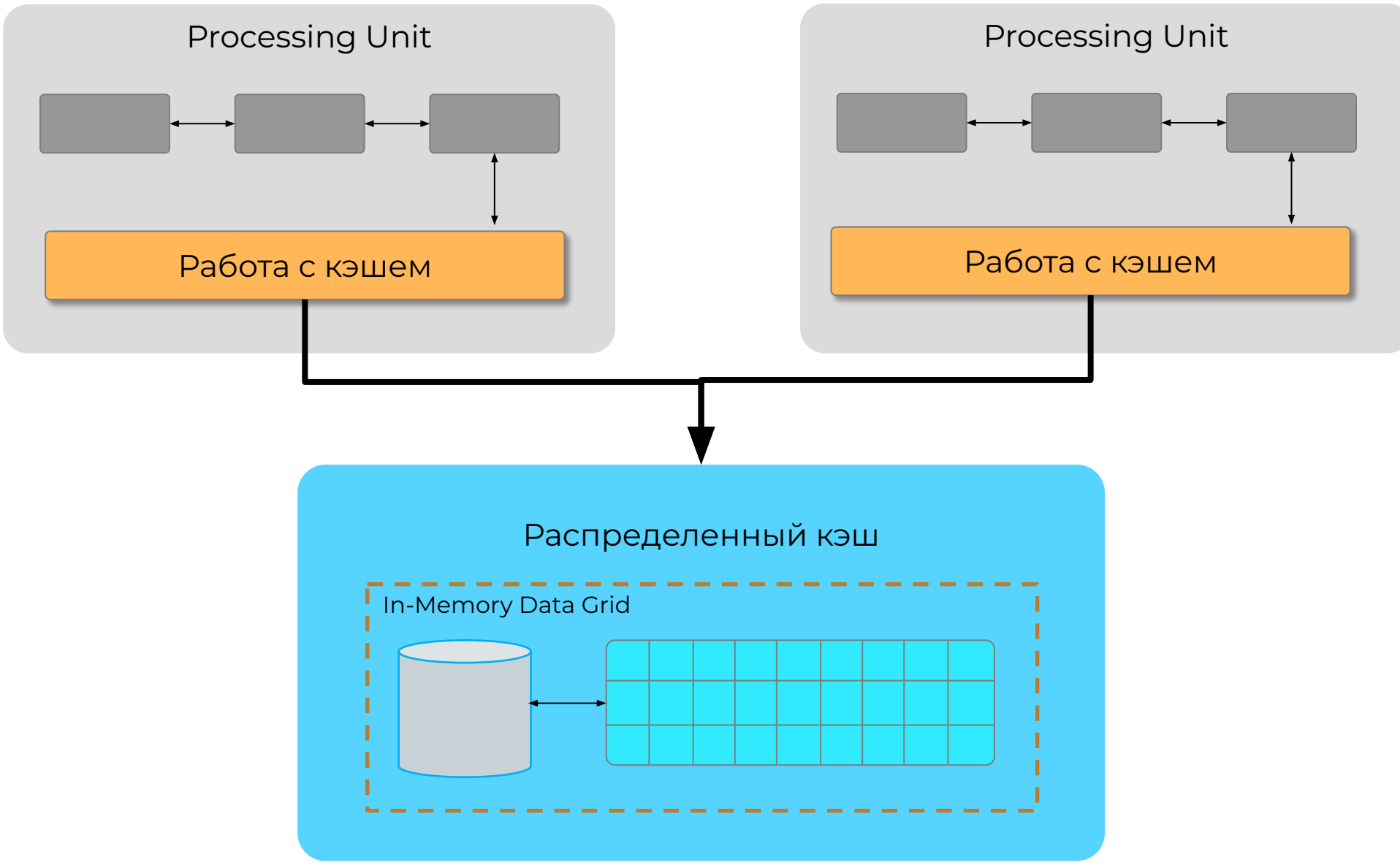
# Реплицированный кэш



# Распределенный кэш



# Распределенный кэш



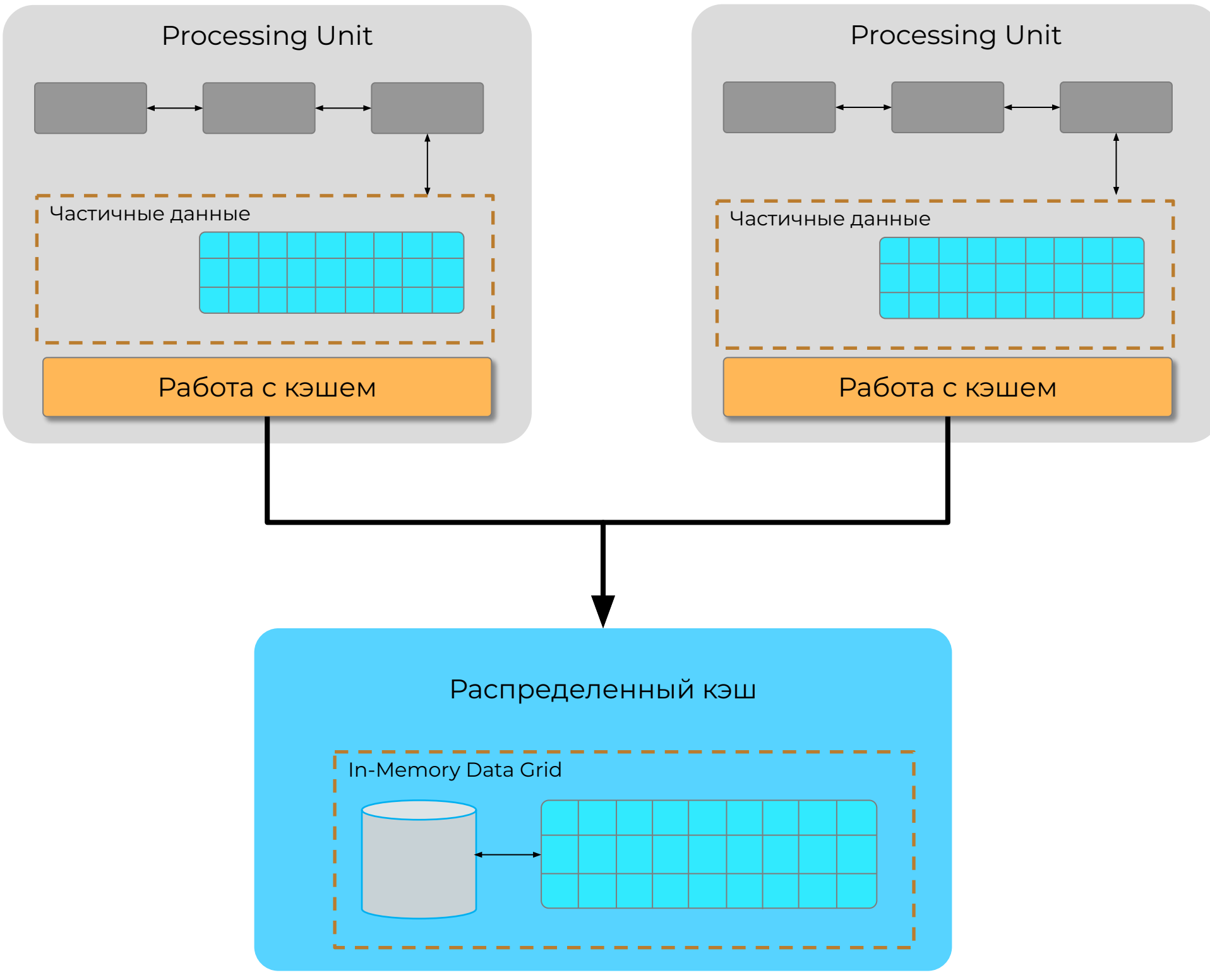
Skillbox

# Сравнение подходов

| Критерий | Replicated Cache | Distributed Cache |
|---|---|---|
| **Оптимизация** | Производительность | Консистентность |
| **Размер cache** | Относительно небольшой (в идеале < 100 МБ) | Любой |
| **Тип данных** | Относительно статичные данные | Часто изменяемые данные |
| **Частота обновлений данных** | Низкая частота (относительно редко) | Высокая частота (частые обновления) |
| **Отказоустойчивость** | Высокая | Низкая |

Skillbox

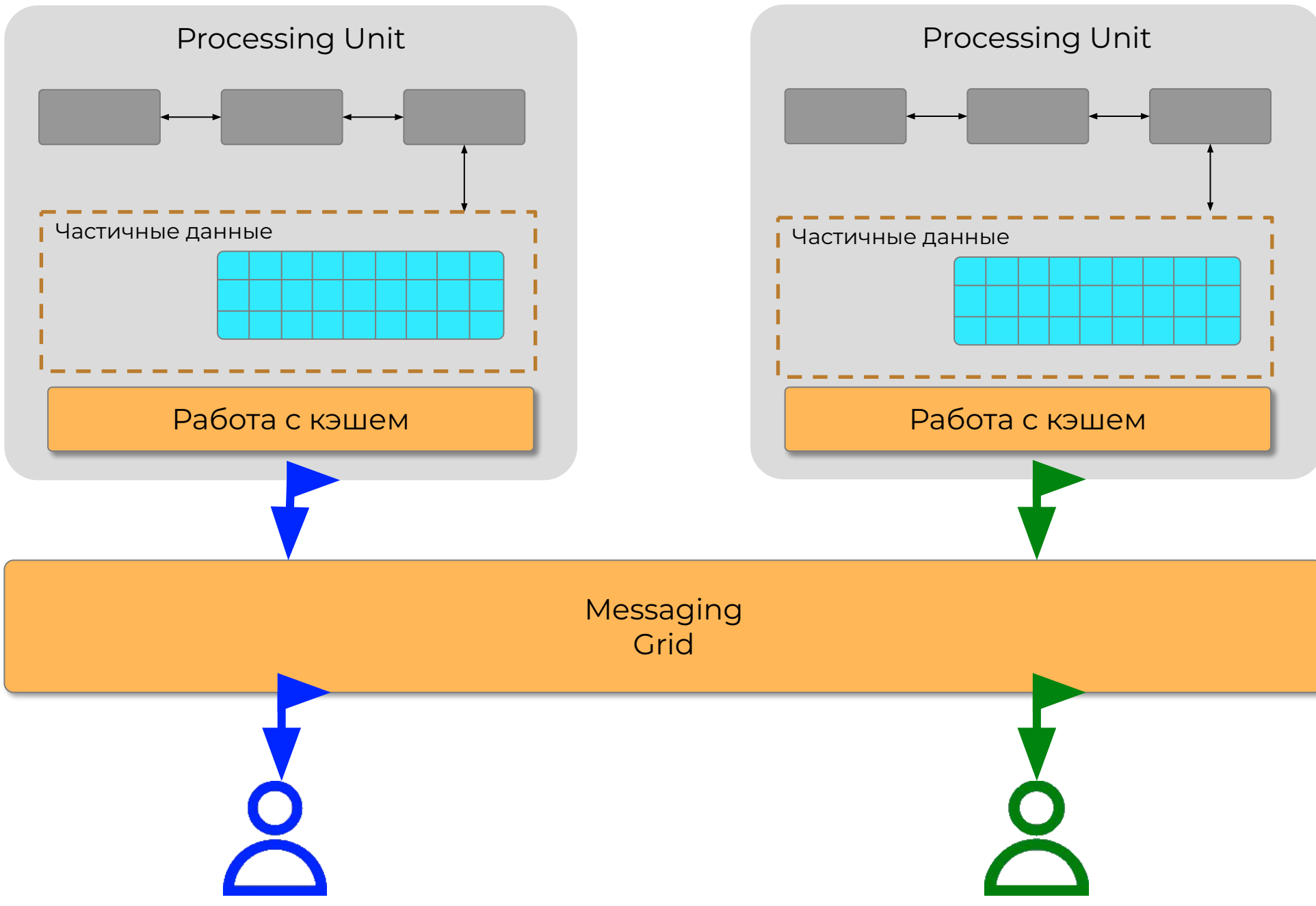
# Near Cache



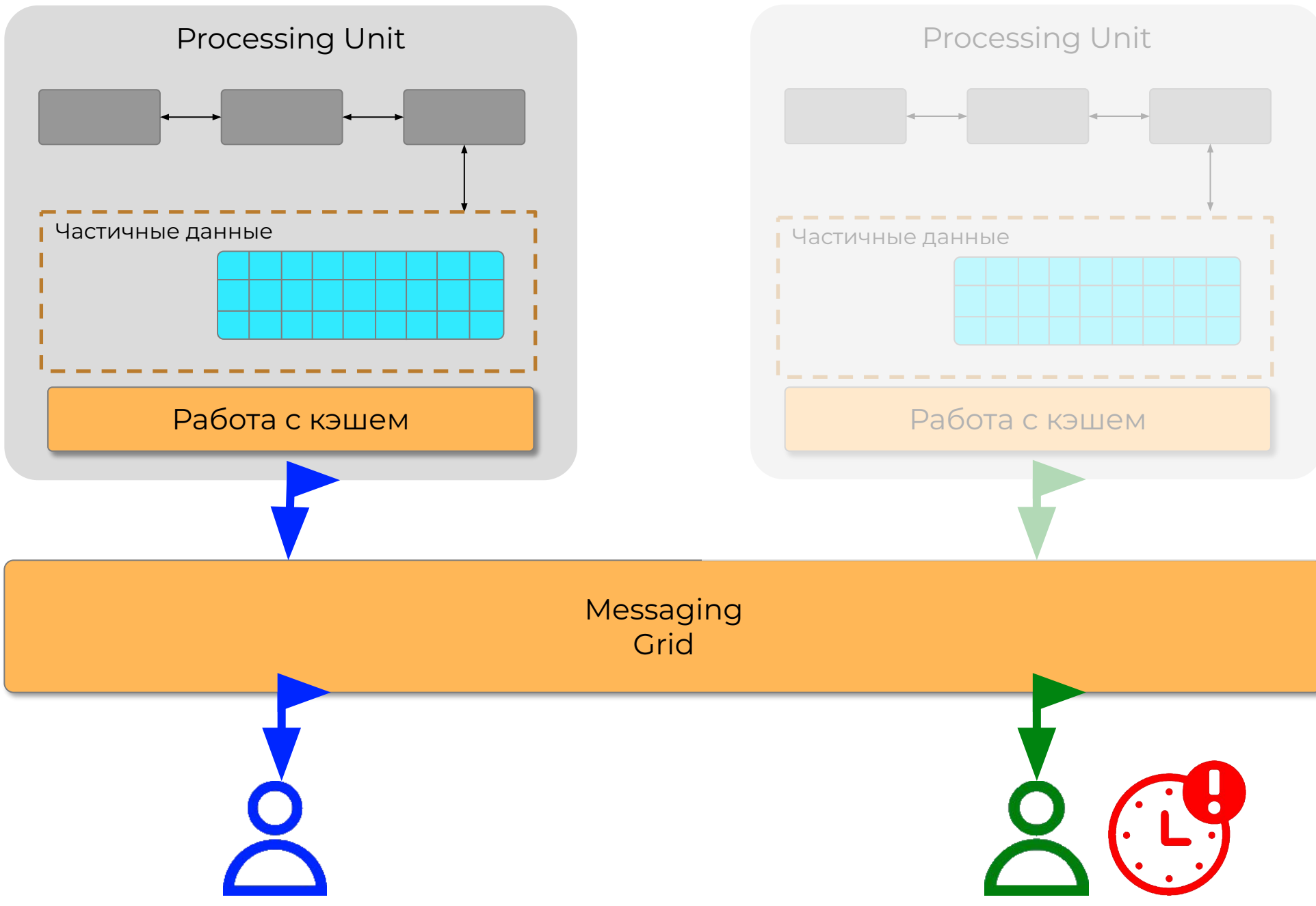
# Стратегии управления

- MRU - most recently used
  последний использованный
- MFU - most frequently used
  часто используемый
- RR - Random replacement
  произвольное удаление

# Near Cache: проблемы



# Near Cache: проблемы



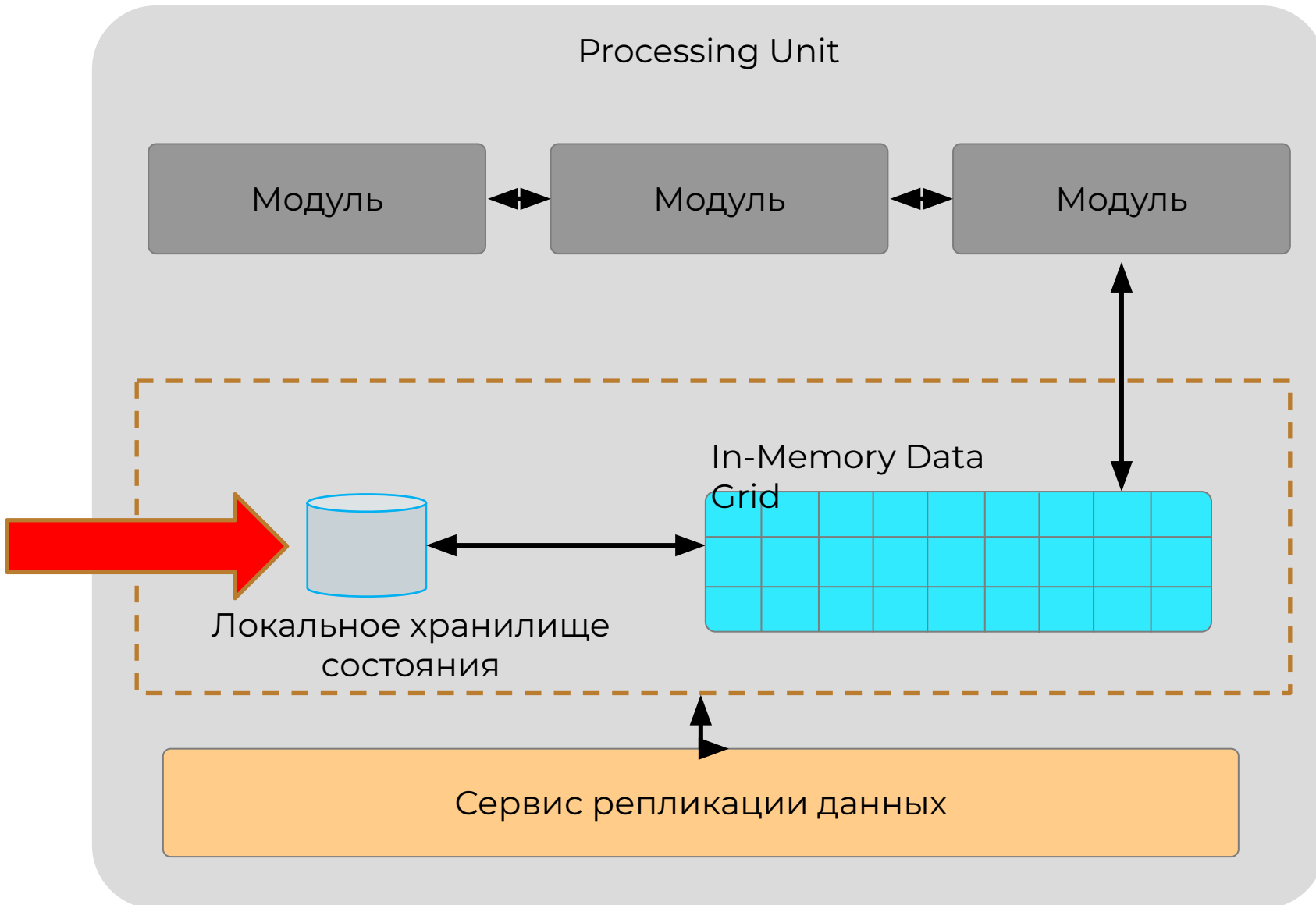
Skillbox

# Работа архитектора заключается в обнаружении и разрешении компромиссов

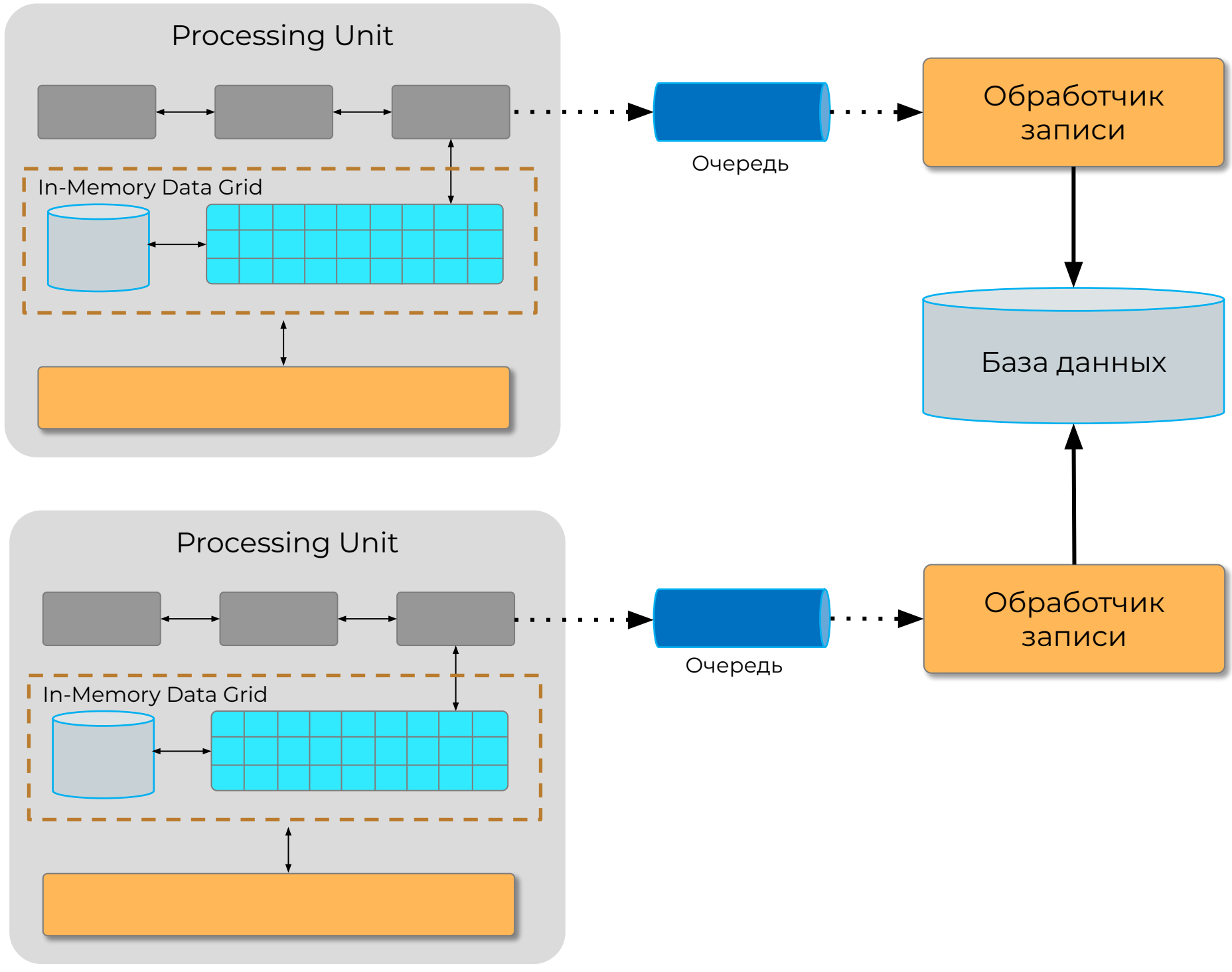
Skillbox

# Как устроен слой данных?

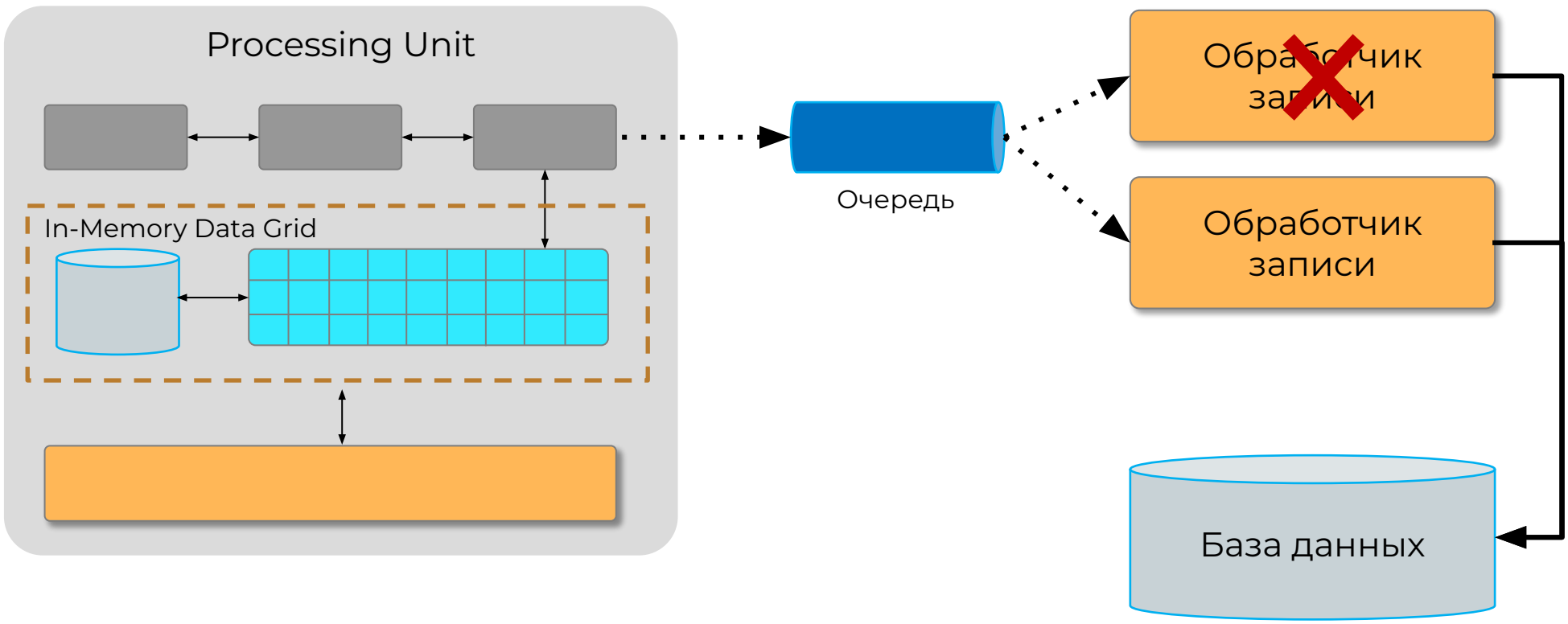
# Коробочное решение



# Запись данных



# Запись данных



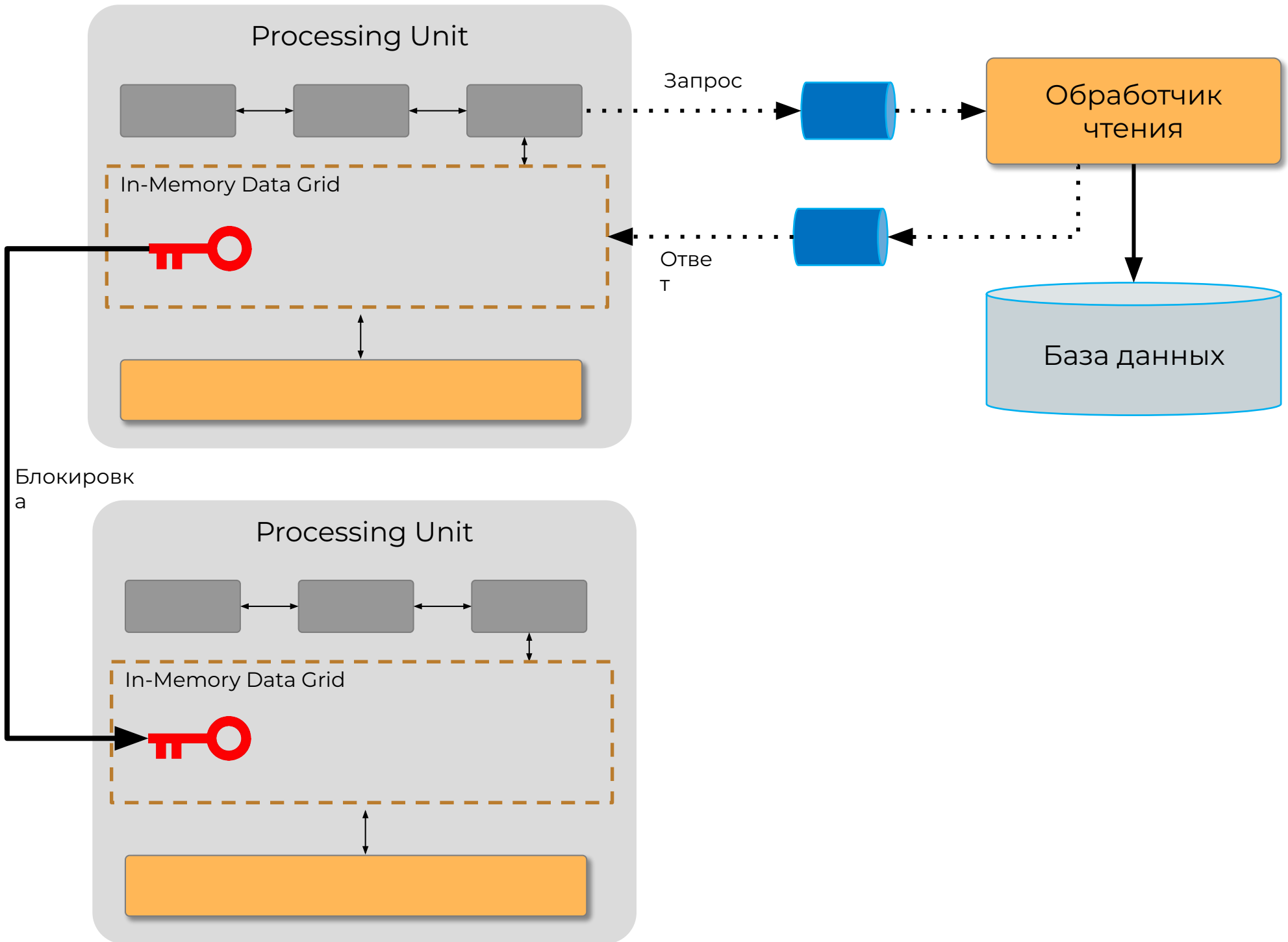

Не стоит создавать дополнительные экземпляры обработчиков записи. Лучше переработать гранулярность сообщений.

# Чтение данных

Происходит только когда:

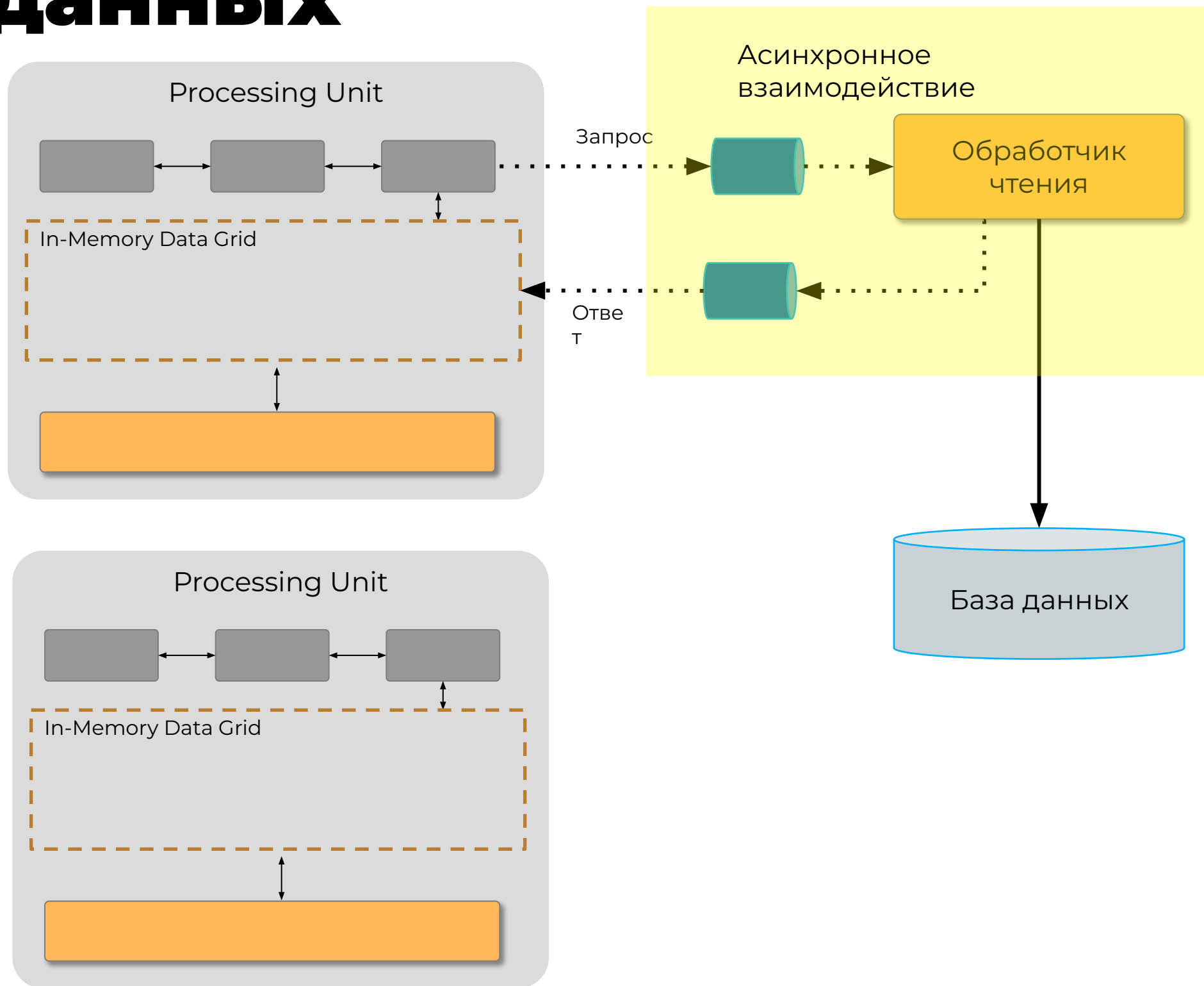
- отказал сервис
- обновляется сервиса
- получение архивных данных не представленных в реплицированном кэше

# Чтение данных



Skillbox

# Обновление структуры данных



# Выводы

**Плюсы:**

- Высокая автоматическая степень масштабируемости
- Высокий уровень отказоустойчивости

**Минусы:**

- Низкая согласованность данных
- Высокая стоимость работ по обеспечению интеграции
- Стоимость специализированного ПО
- Дополнительные требования к грануляции данных

# Самое важное

Использовать только когда нужна скорость обработки данных и быстрый ответ на рост нагрузки, а все остальные возможности исчерпаны

# Что дальше?

- Архитектурный стиль Event-Driven
- Почему CQRS и Event Sourcing не обязательны для его реализации