DOTNEXT

Модульная

# ~~Гибридная~~ архитектура

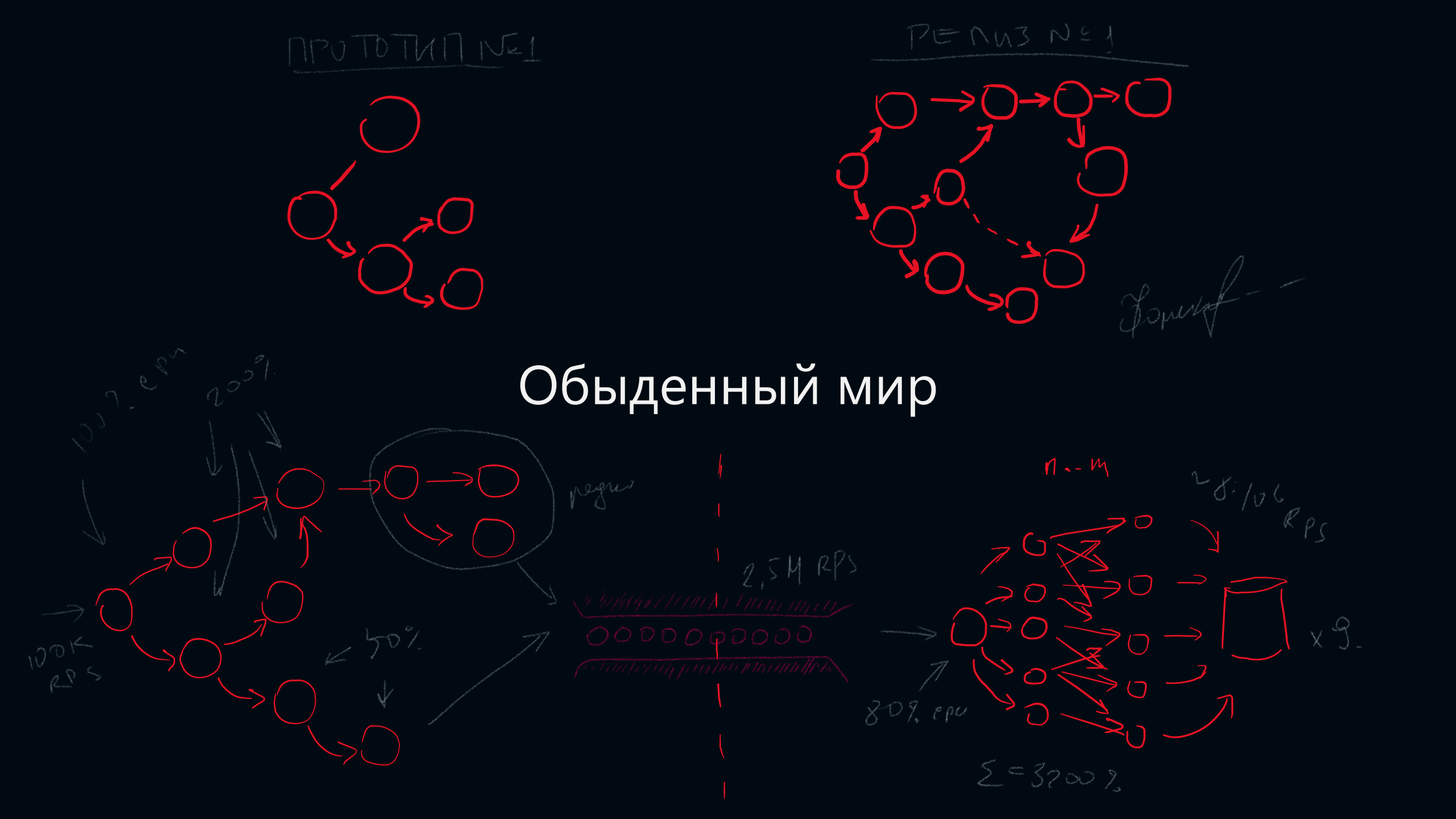
Слияние микросервисов в монолит по необходимости

Сидристый Станислав

DOTNEXT

# Станислав Сидристый

- Системный архитектор в Центре Речевых Технологий
- C#, C/C++, C++/CLI когда необходимо
- Книга: https://github.com/sidristij/dotnetbook
- telegram: @sidristij
- sunex.development@gmail.com




# Обыденный мир

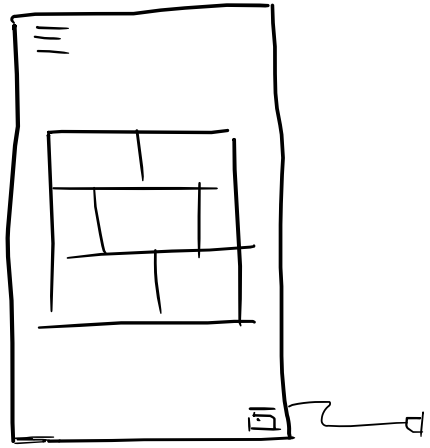
# Что дано

- Система делается под понятное будущее:
  - высокая нагрузка, вызванная:
    - как внутренними (переработка данных, составление отчётов, статистик и проч.)
    - так и внешними причинами (затягивание задач через интеграции, работа пользователей)
  - динамическая нагрузка на разные участки системы:
    - днём – одни, ночью – другие
    - было бы хорошо динамически сужать/расширять кол-во сервисов
  - сейчас нагрузки нет => нужная изначально простая конфигурация
- Наоборот: сейчас огромная система, но есть клиенты, которые хотят *по-простому*
- Проект на старте и потому нет ясности, куда идти

Что делать?

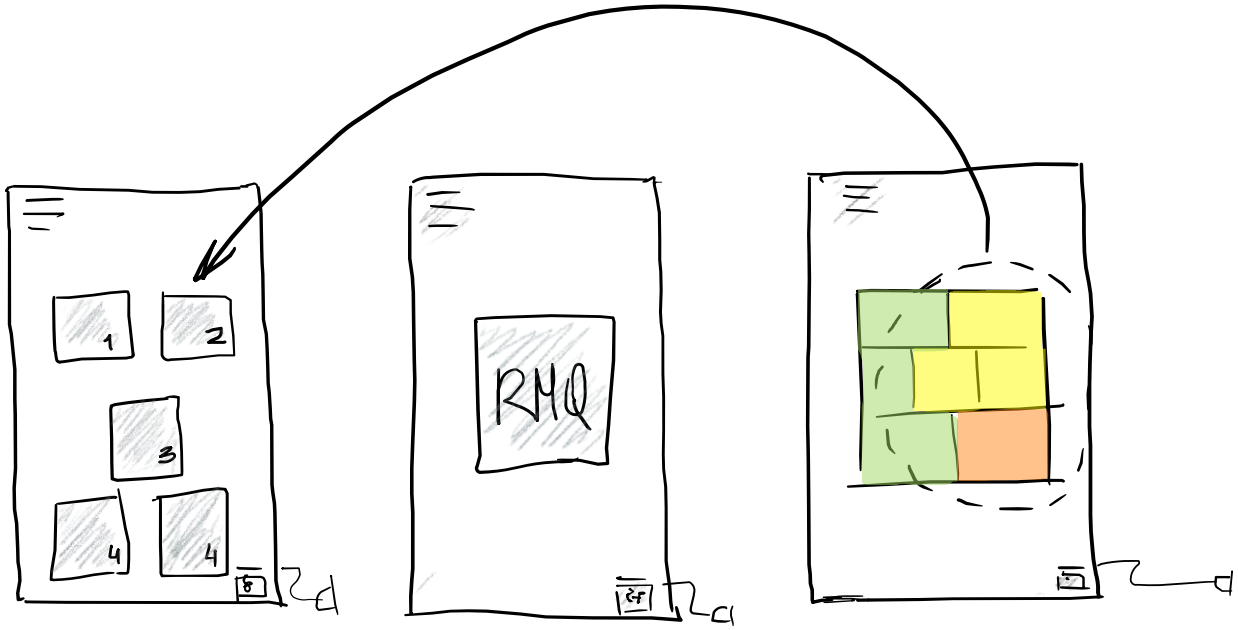
# МОНОЛИТ

- Растёт нагрузка + возникают службы, фоновые задачи с неравномерной нагрузкой и собственным состоянием
- Проходит год-два, хочется разделить на сервисы, оправить к k8s



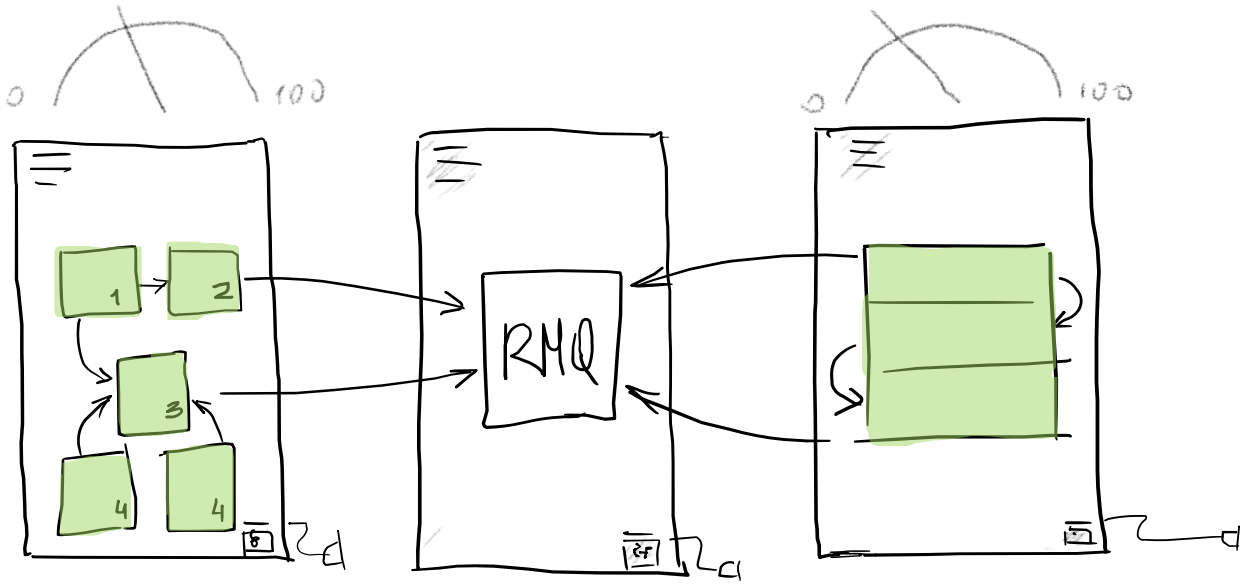
# МОНОЛИТ

- Растёт нагрузка + возникают службы, фоновые задачи с неравномерной нагрузкой и собственным состоянием
- Проходит год-два, хочется разделить на сервисы, оправить к k8s



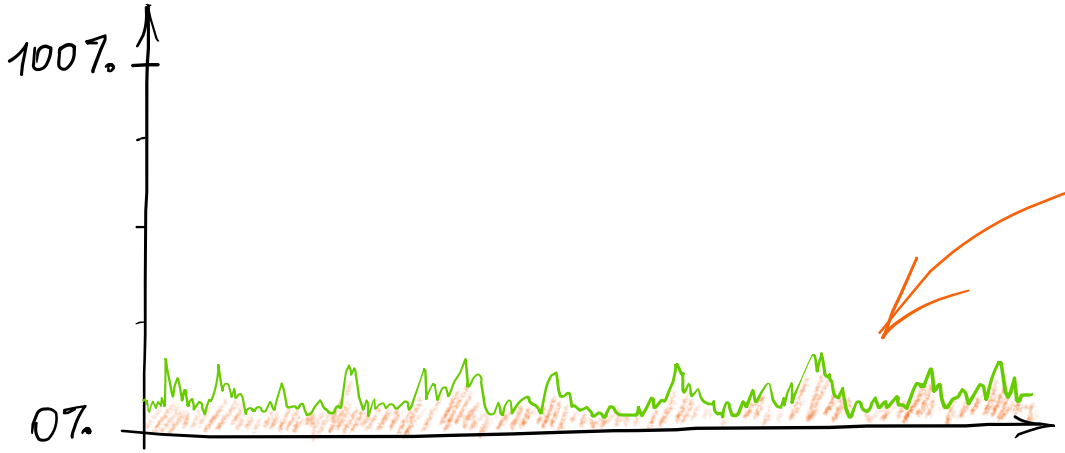
# МОНОЛИТ

- Растёт нагрузка + возникают службы, фоновые задачи с неравномерной нагрузкой и собственным состоянием
- Проходит год-два, хочется разделить на сервисы, оправить к k8s

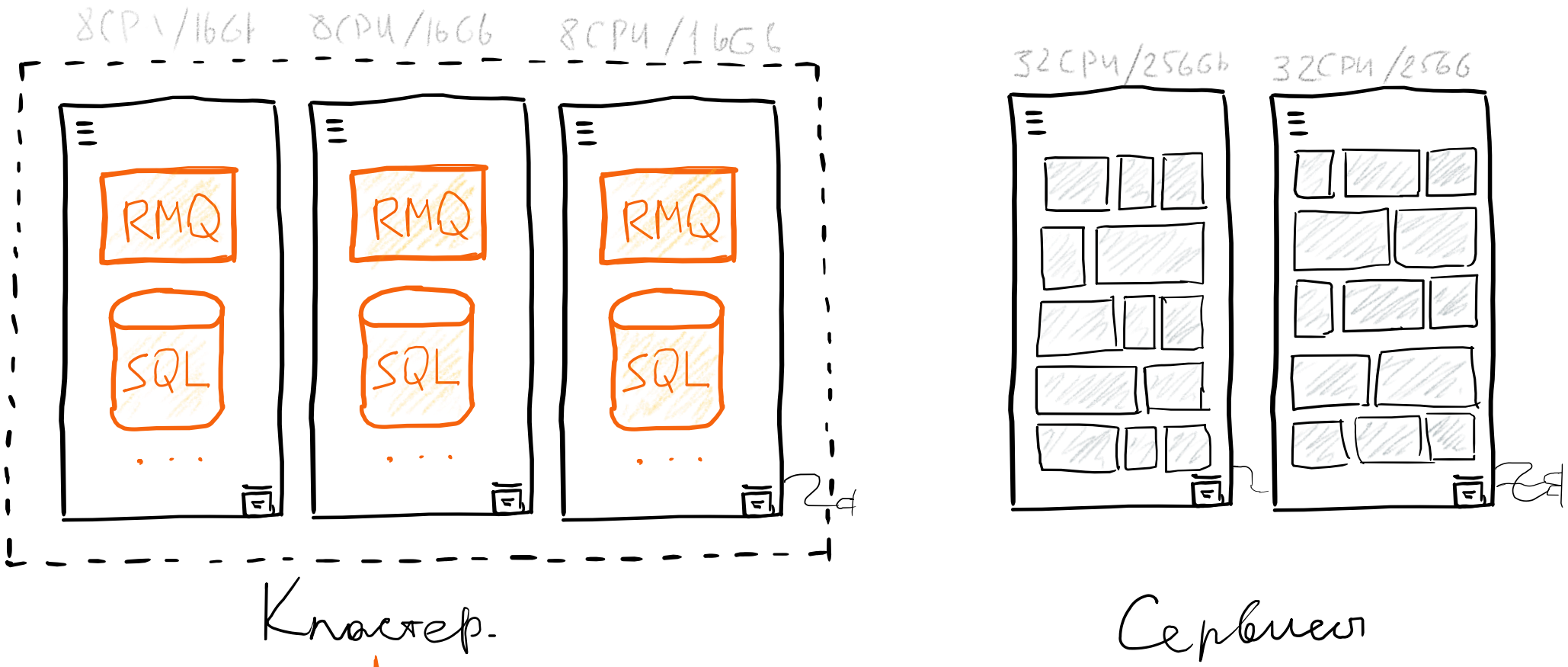


# микросервисная

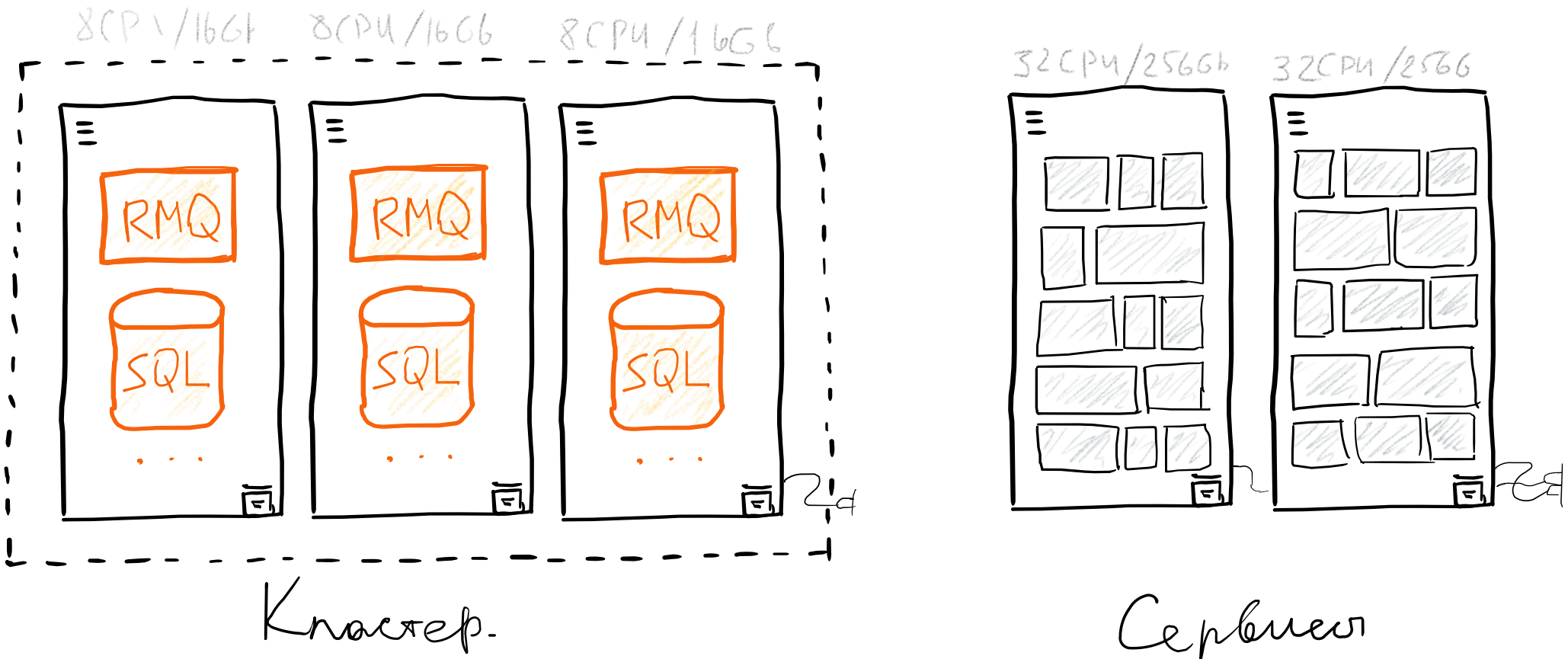
- Ожидание большой неравномерной нагрузки *в будущем*
- Затраты на деплой в k8s, поддержка кластеров k8s, redis, rmq, mongodb/postgres/…
- Все сервисы – stateless, танцы со statefull
- CI/CD
- однако…




на каждом из N серверов !

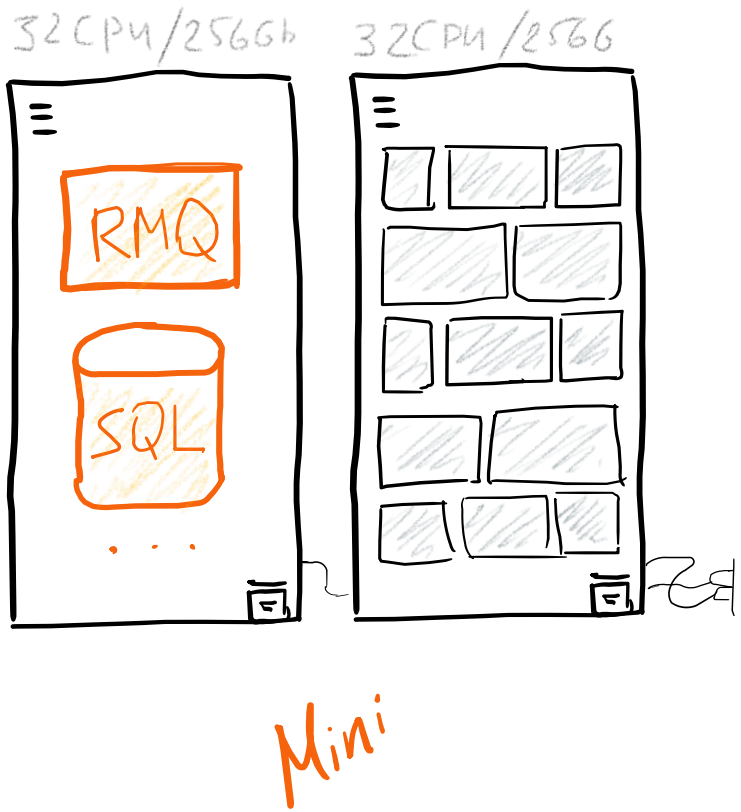



Как итог инфра на 3 для кластеризации и сервисы – каждый в 1 экземпляре




Как итог инфра на 3 для кластеризации и сервисы – каждый в 1 экземпляре

# микросервисная




Либо вообще микро-вариант для пробы, базовой нагрузки, без гарантий

# микросервисная

- однако, сохраняется возможность правильного деплоя на N серверов с k8s
- и возможностью горизонтального масштабирования

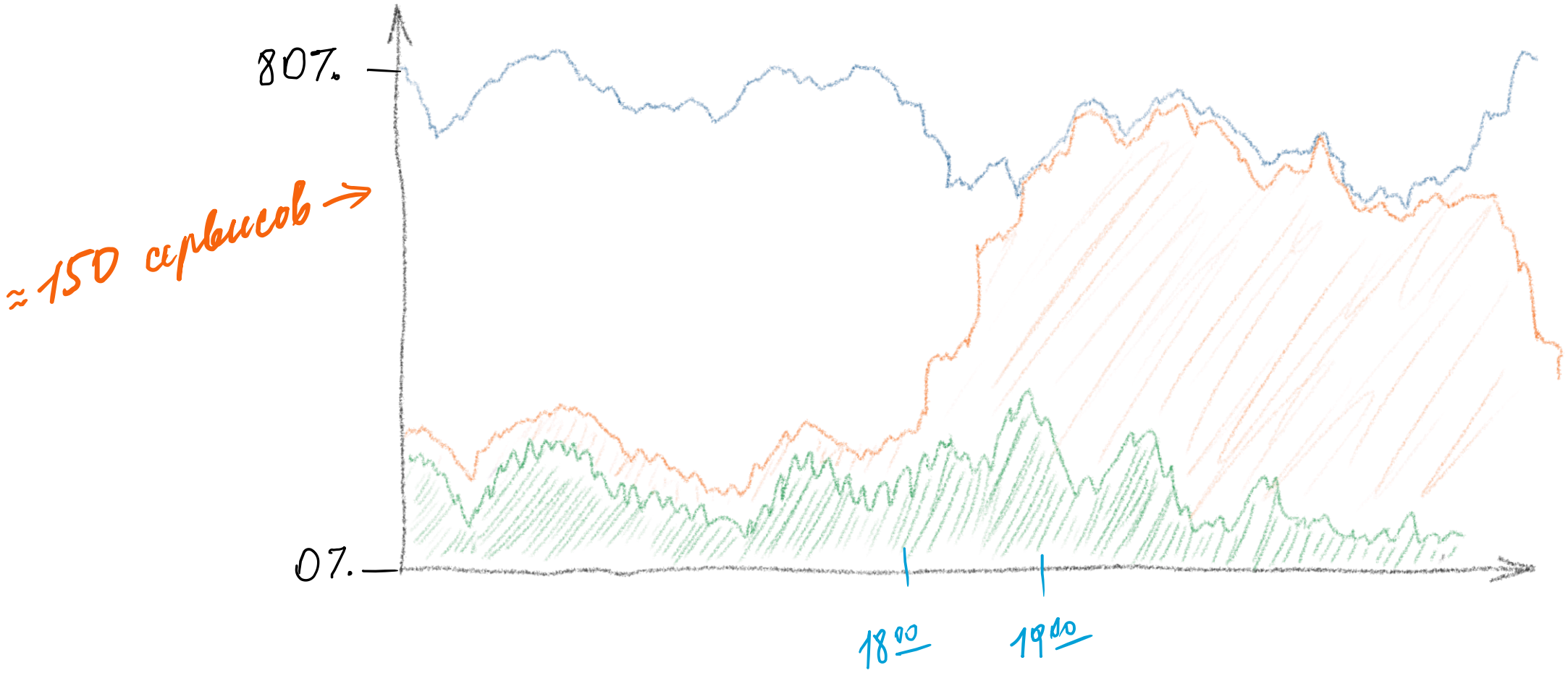
НО!

Чтобы не столкнуться с неработающим комплексом в будущем:

- важно иметь деплой на много серверов в тестовой зоне
- иметь такой же деплой для разработки

# Проблема общих ресурсов

# Много сервисов, мало серверов




80% на 0-1% CPU, часть активна днём, часть - ночью

# Много сервисов, мало серверов

Ограничения:

- в docker: по ядрам и по памяти (1-2 ядра, от 512 Мб RAM)
- помним про Workstation/Server GC
- и про ThreadPool

Однако:

- при 150 сервисах на 32 ядра у каждого quota = 1 CPU
- т.е. при неудачном стечении обстоятельств это потребность выше 32 ядер

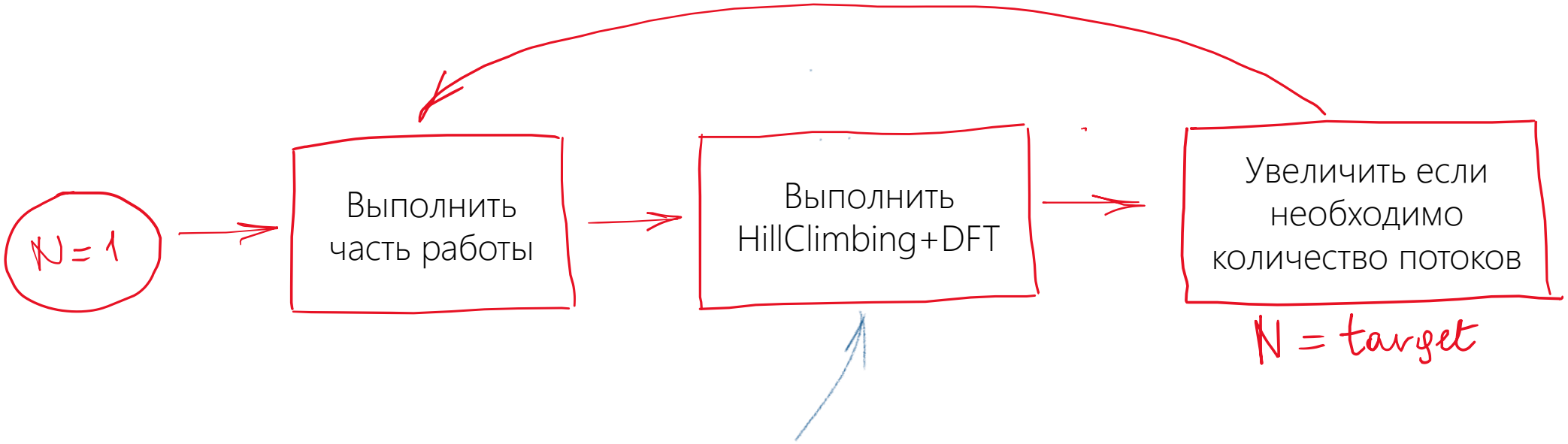
# Много сервисов, мало серверов

К чему ведёт:

- растёт очередь задач в ThreadPool
- увеличивается его уровень параллелизма
- падает общая производительность
- растёт очередь задач в ThreadPool
- увеличивается его уровень параллелизма
- падает общая производительность
- растёт очередь задач в ThreadPool
- увеличивается его уровень параллелизма
- падает общая производительность
- ...

# Много сервисов, мало серверов

Замедление ThreadPool




т.к. при высокой нагрузке скорость упадёт, кол-во потоков вырастет

Без выставления максимума, процесс роста и деления общих ядер убьёт систему

# Много сервисов, мало серверов

Как достичь безопасности?

- Сумма всех требований к количеству ядер CPU не должна быть сильно больше их реального количества;
- Иначе существует риск внезапной смерти всей системы.

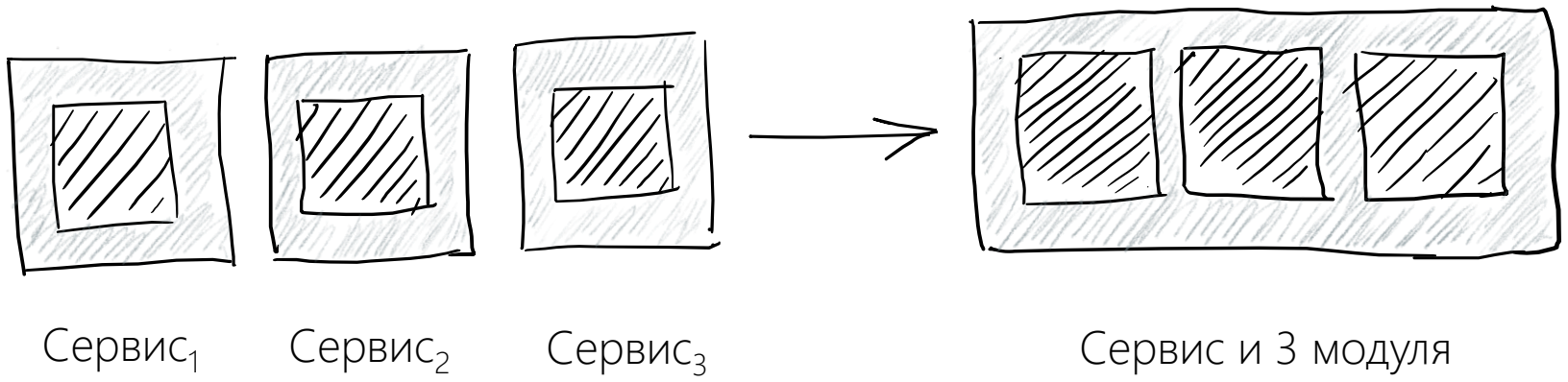
Выход?

- объединить сервисы

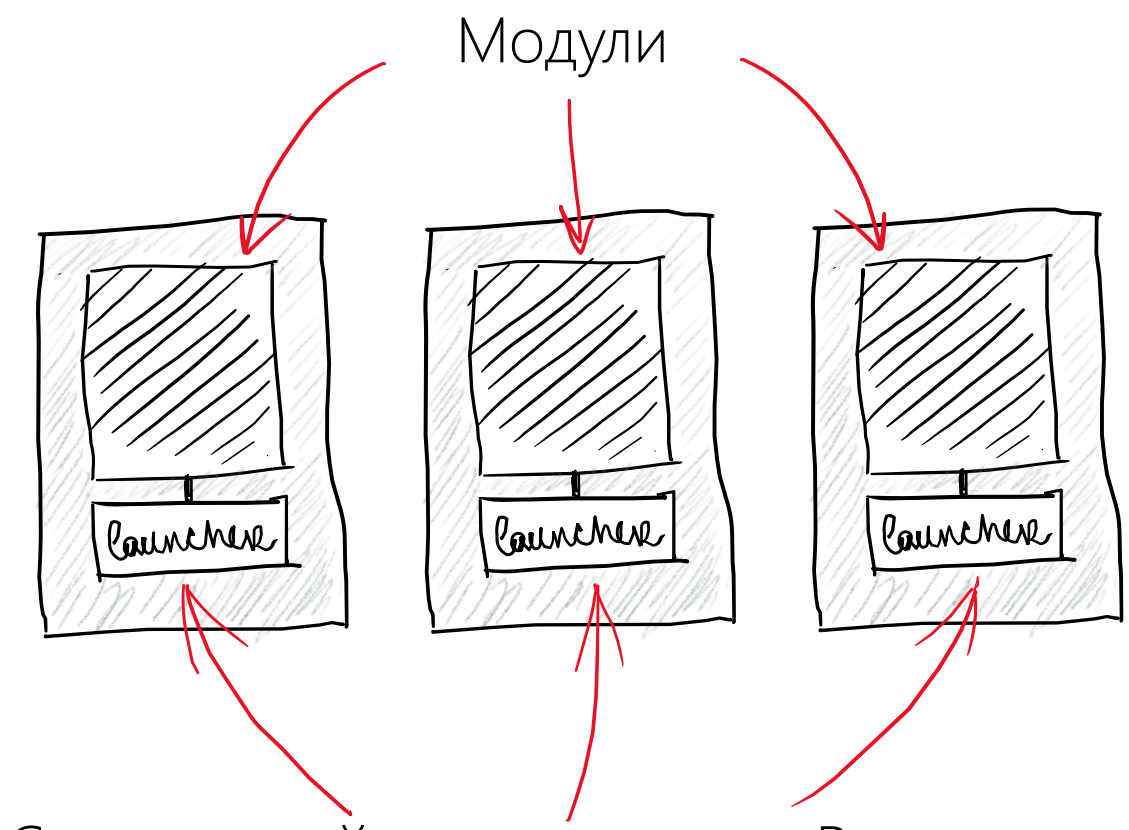
# Модульность

# Модульность

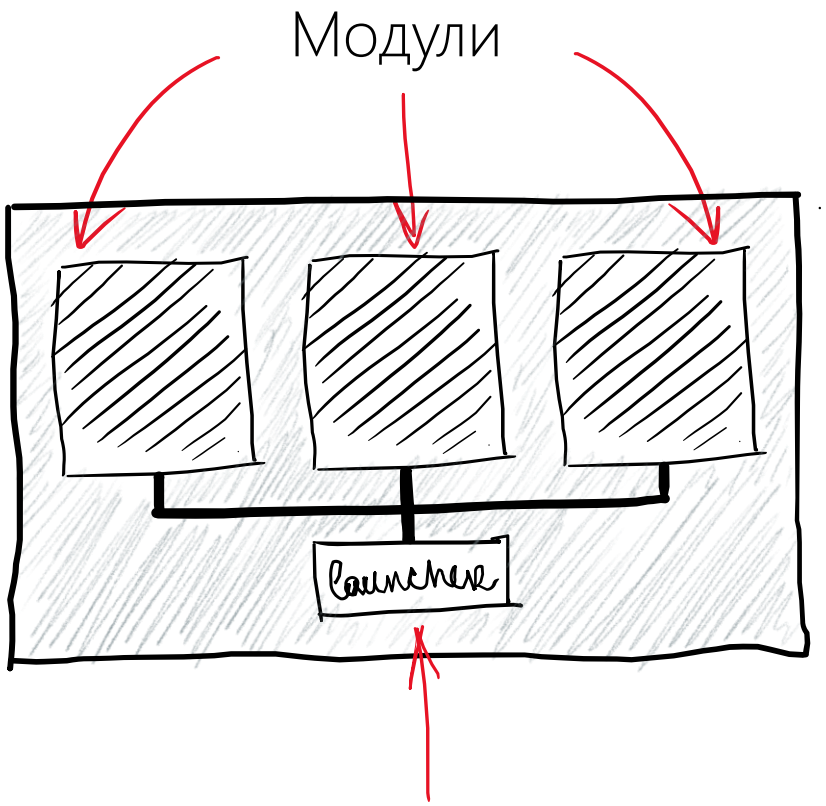
- Пусть весь код сервиса – это некий модуль
- А сам сервис – его оболочка
- Тогда возникает *осторожное* желание в рамках одного сервиса иметь несколько модулей



# Модульность



# Модульность




Стандартный запуск модуля, Program.cs

# Модульность

Что это даст?

- ThreadPools из нескольких станут одним => меньше вероятность разрастания
- Память нескольких станет единой => меньше SOH/LOH/POH куч выделено
- Объединённые сервисы в целом станут меньше потреблять

# Какими будут результаты

# Результаты

При простое

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис 1 | 281 | 0-1 |
| Сервис 2 | 290 | 0-1 |
| Сервис 3 | 269 | 0-1 |
| Сервис 4 | 296 | 0-2 |
| Сервис 5 | 272 | 0-1 |
| Сервис 6 | 259 | 0-1 |
| Сервис 7 | 100 | 0-1 |
| Сервис 8 | 281 | 4-8 |
| Сервис 9 | 271 | 0-1 |
| Сервис 10 | 270 | 0-1 |
| Сервис 11 | 266 | 0-1 |
| СУММА : | 2855 | 4-19 |

# Результаты

При простое

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис 1 | 281 | 0-1 |
| Сервис 2 | 290 | 0-1 |
| Сервис 3 | 269 | 0-1 |
| Сервис 4 | 296 | 0-2 |
| Сервис 5 | 272 | 0-1 |
| Сервис 6 | 259 | 0-1 |
| Сервис 7 | 100 | 0-1 |
| Сервис 8 | 281 | 4-8 |
| Сервис 9 | 271 | 0-1 |
| Сервис 10 | 270 | 0-1 |
| Сервис 11 | 266 | 0-1 |
| СУММА : | 2855 | 4-19 |

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис | 410 | 3-10 |
| | x6.9 меньше | X2 меньше |

# Результаты

Под нагрузкой

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис 1 | 345 | 10 |
| Сервис 2 | 330 | 20 |
| Сервис 3 | 315 | 1 |
| Сервис 4 | 308 | 1 |
| Сервис 5 | 375 | 107 |
| Сервис 6 | 311 | 3 |
| Сервис 7 | 307 | 10 |
| Сервис 8 | 110 | 14 |
| Сервис 9 | 375 | 11 |
| Сервис 10 | 445 | 90 |
| Сервис 11 | 430 | 80 |
| СУММА : | 3651 | 347 |

# Результаты

Под нагрузкой

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис 1 | 345 | 10 |
| Сервис 2 | 330 | 20 |
| Сервис 3 | 315 | 1 |
| Сервис 4 | 308 | 1 |
| Сервис 5 | 375 | 107 |
| Сервис 6 | 311 | 3 |
| Сервис 7 | 307 | 10 |
| Сервис 8 | 110 | 14 |
| Сервис 9 | 375 | 11 |
| Сервис 10 | 445 | 90 |
| Сервис 11 | 430 | 80 |
| СУММА : | 3651 | 347 |

| Сервис | Память, MB | CPU usage |
|---|---|---|
| Сервис | 640 | 170 |
| | x5.5 меньше | x4 меньше |

# решение

# Решение разбиения на модули

- Стартом asp.net заведует startup проект
- Если структура та же, но не startup, стартовать не будет
- Потому создаём (генерируем) новый startup проект, умеющий запускать все зависимые

# Код

```
/// </summary>
public void Enqueue(ExecutionUnitAsync<TPoolParameter> unit, object outer = default, bool preferLocal = true)
{
    PoolWork poolWork = default;
    poolWork.Init(unit, outer);
    _defaultQueue.Enqueue(poolWork, preferLocal);
}

/// <summary>
/// Initialize with WaitHandle and non-parametrized continuation
/// </summary>
public void RegisterWaitForSingleObject(WaitHandle handle, ExecutionUnit unit, object outer = default, TimeSpan timeout = def
{
    var wfsoState = new WaitForSingleObjectState
    {
        Delegate = unit,
        Timeout = timeout,
        Handle = handle,
        InternalState = outer
    };

    void ExecutionUnitCallback(object args)
    {
        var workload = (WaitForSingleObjectState)args;
        workload.Handle.WaitOne(workload.Timeout);
        workload.Delegate(workload.InternalState);
    }
```

# локальные вызовы

# Локальные вызовы

Если модули находятся в одном сервисе, но при этом вызовы – через HTTP или MQ

- Накладные расходы на сеть
- Накладные расходы на CPU
- Накладные расходы на память

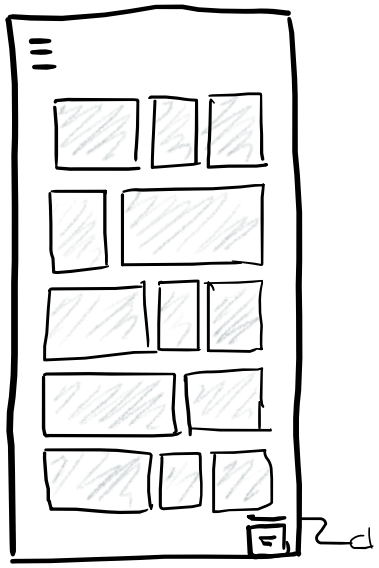
# Локальные вызовы

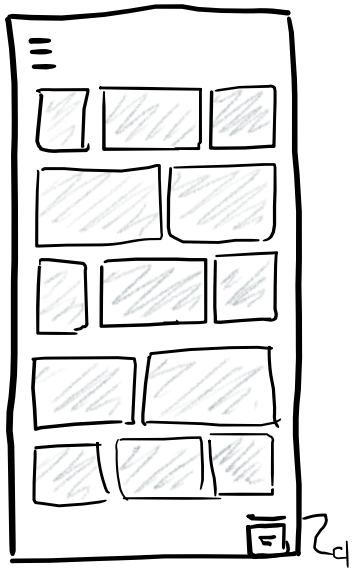
Если модули находятся в одном сервисе, но при этом вызовы – через HTTP или MQ

- Сильно медленнее:
  - Сериализовать
  - Сформировать HTTP запрос в буфере
  - Отправить
    - TCP/IP
  - Получить пакет
  - Разобрать HTTP ответ
  - Десериализуем
  - Опа!
    - TCP/IP
  - Получить пакет
  - Разобрать HTTP запрос
  - Десериализуем
  - Делаем что надо
  - Сериализовать ответ
  - Сформировать HTTP ответ в буфере
  - Отправить
    - TCP/IP

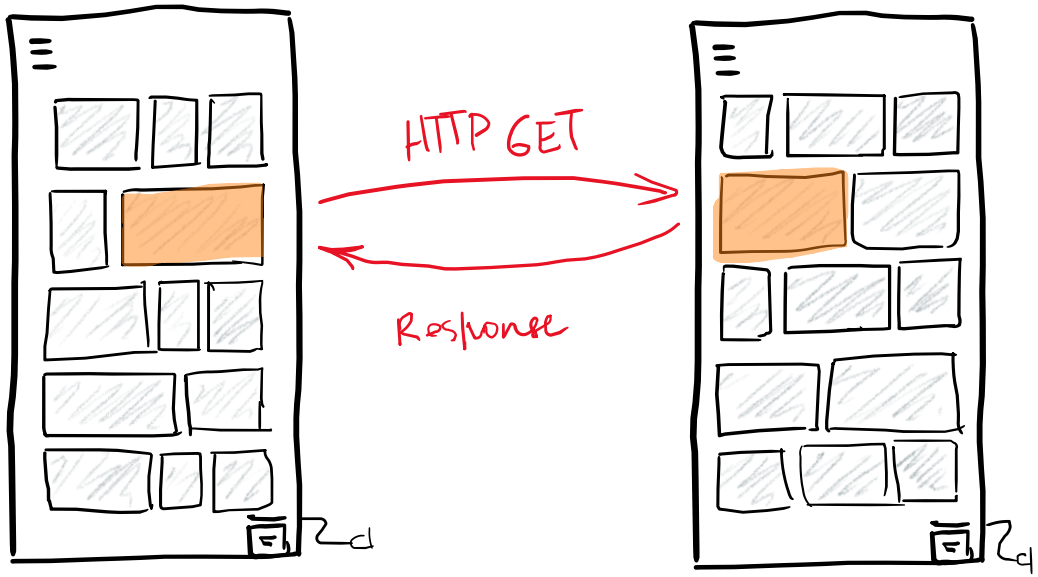
# Доводим до финального решения

# Доводим до финального решения

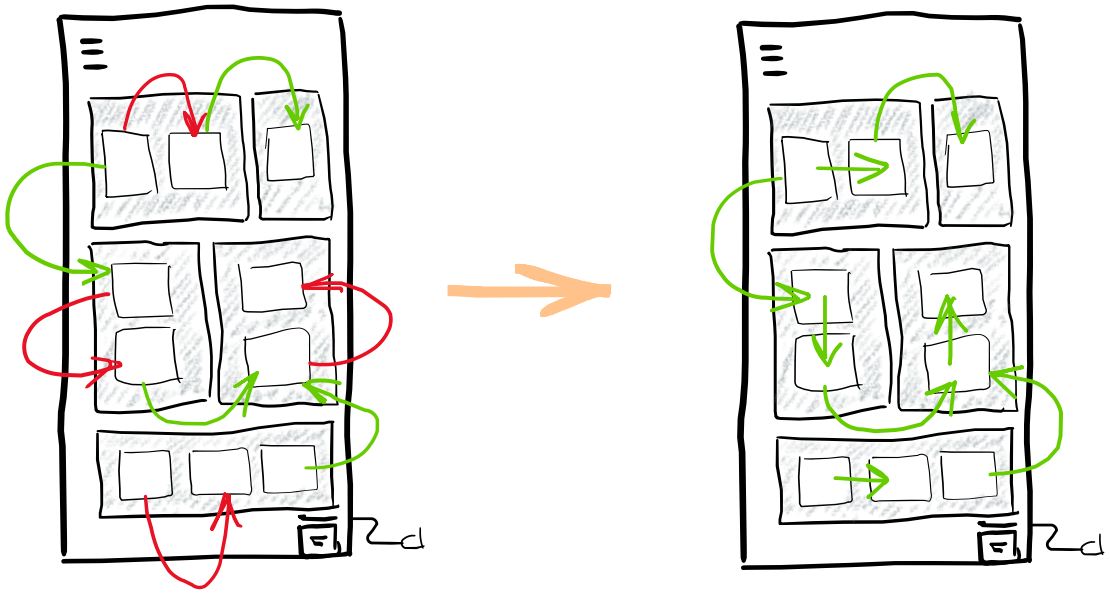




# Доводим до финального решения



# Доводим до финального решения



HTTP/RMQ вызов соседнего модуля сервиса

обычный вызов метода

# развязка

# Локальные вызовы – локально

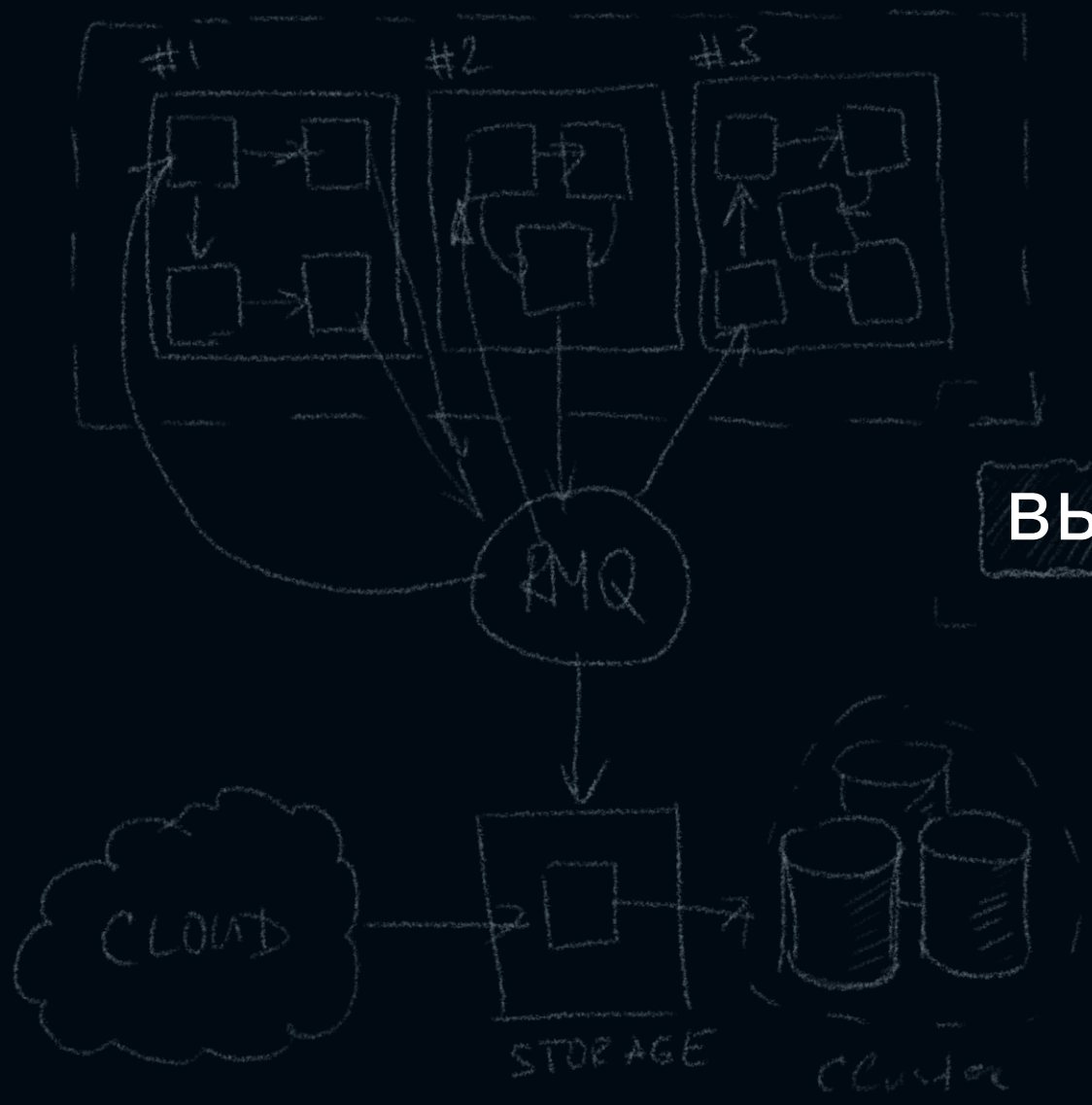
HTTP:

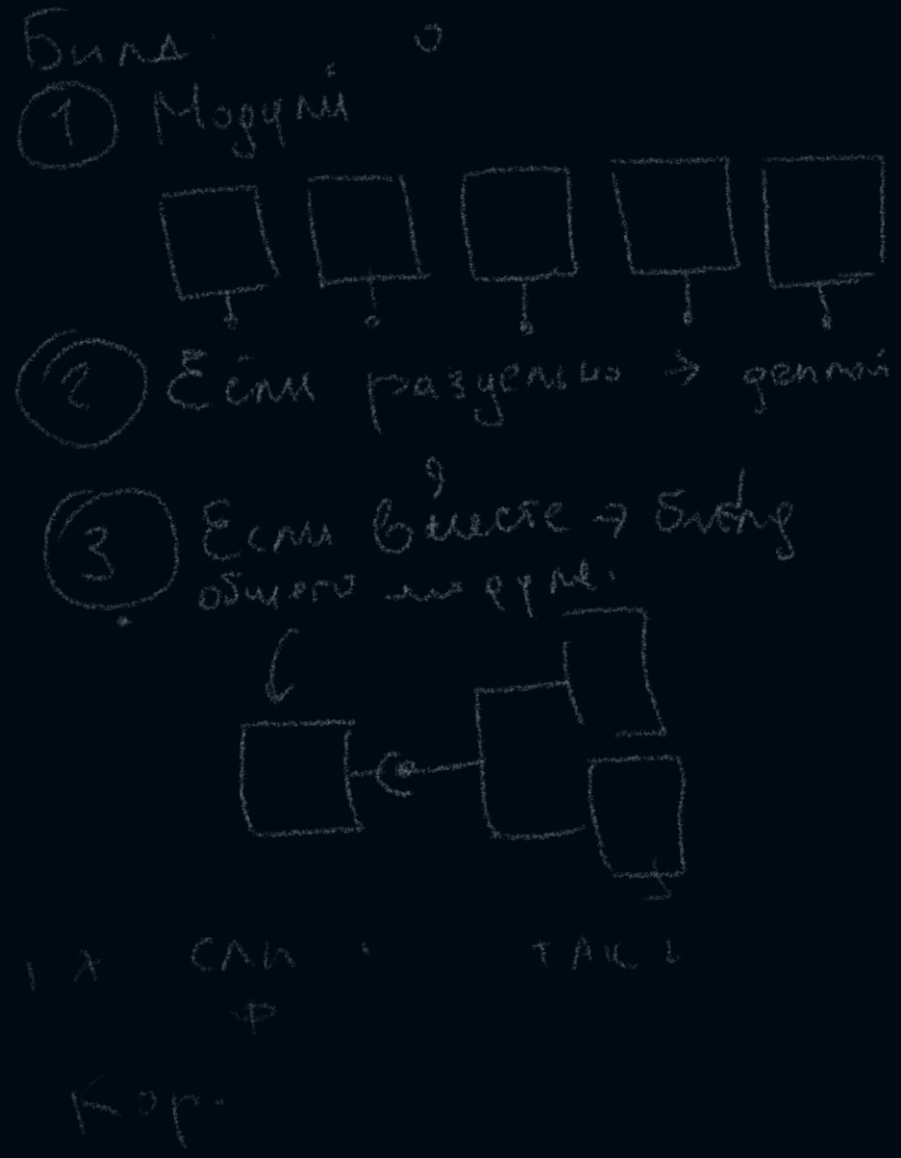
- Делаем интерфейс ISomeLibraryClient
- Для HTTP сервисов реализуем SomeLibraryRemoteClient
- Для локальных вызовов реализуем SomeLibraryLocalClient
- Фабрика для предоставления LocalClient если сам сервис находится в том же сервисе и RemoteClient если нет

RMQ?

- Разработать оболочку над клиентом
- См. HTTP

# ВЫВОДЫ


#1
#2
#3
RMQ
CLOUD
STORAGE


Модуль
Если разумно → делай
Если вместе → build
общего

# Выводы

- Если замечать проблемы, придёт понимание, как их решать
- Можно менять стандартные, привычные механизмы
- Решение может получиться лучше чем у профи в .NET Team
- Если не попробуешь – никогда не узнаешь.

# QA