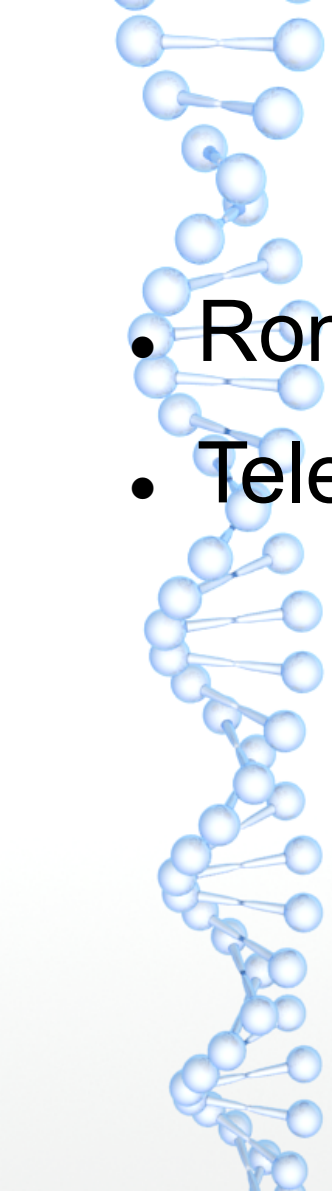
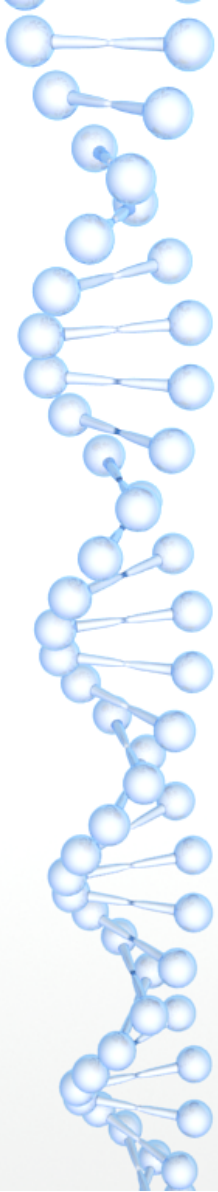


По пути из Kafka в NiFi

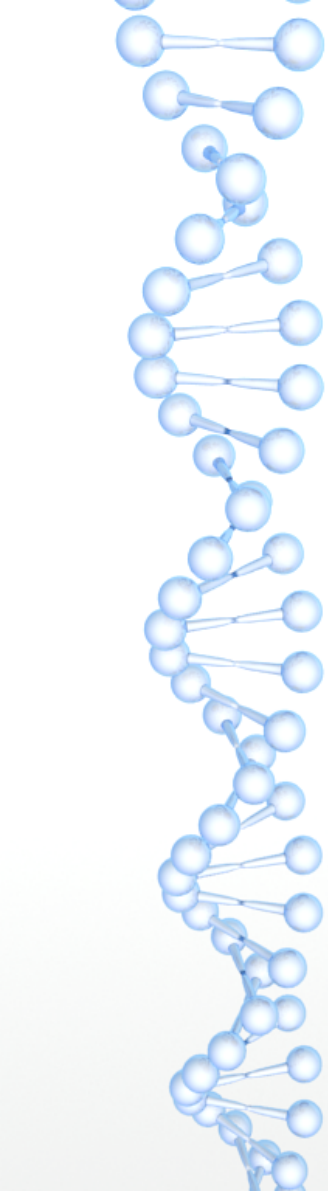
- Roman Korobeinikov, VirtualHealth.com
- Telegram: @roman korobeinikov



Helios от VirtualHealth – это care management platform.

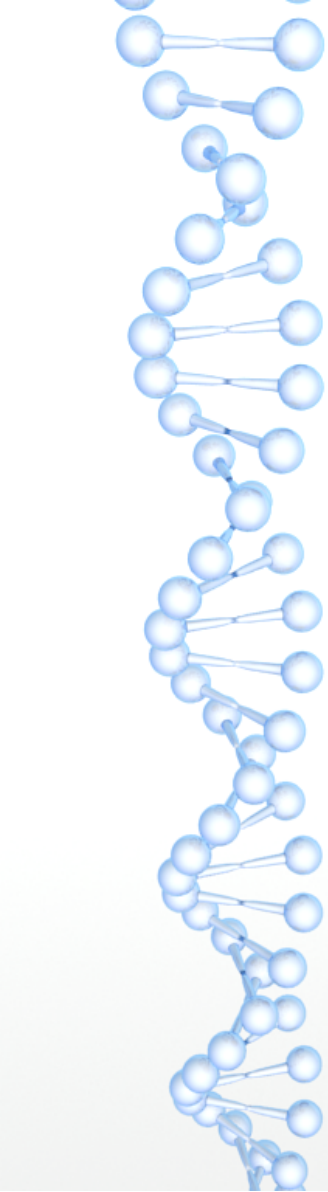
По-русски: система для управления процессом оказания медицинской помощи с точки зрения страховой компании.

Основные пользователи — медицинские учреждения и страховые компании.



Зачем **Helios** пользовательские данные и ETL для них?

Большинство существующих медицинских систем узко-специализированны, после реформы 2011 года (Obamacare) появилась необходимость унификации этих данных, VirtualHealth решил эту задачу. Клиенты пользуются и своими системами, и нашей.

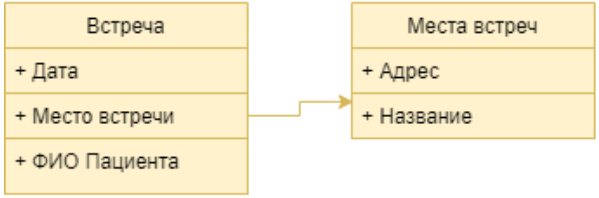


Что представляют собой интеграции

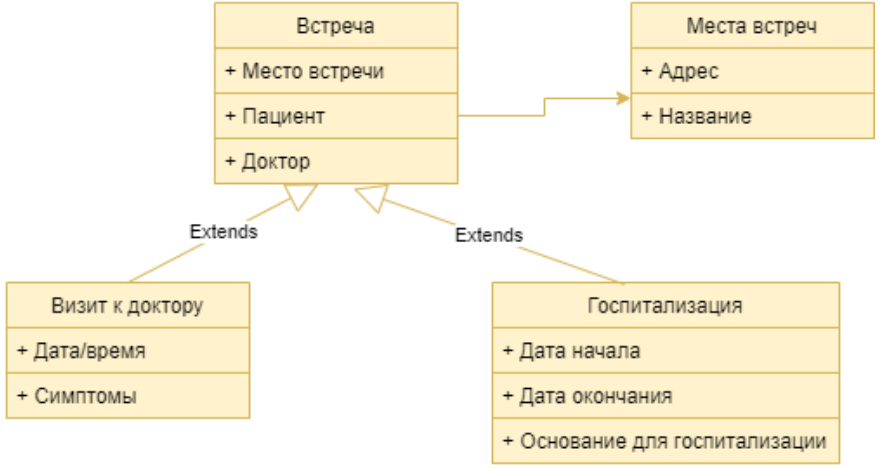
Каждая интеграция — передача данных из **одной** предметно-ориентированной системы в **другую**, при этом обе системы моделируют одну и ту же предметную область (национальное здравоохранение).

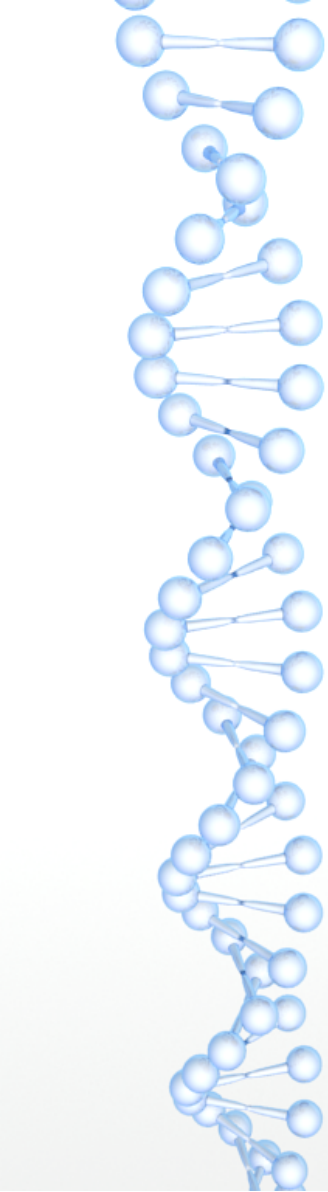
Разные модели одной предметной области

СИСТЕМА №1



СИСТЕМА №2

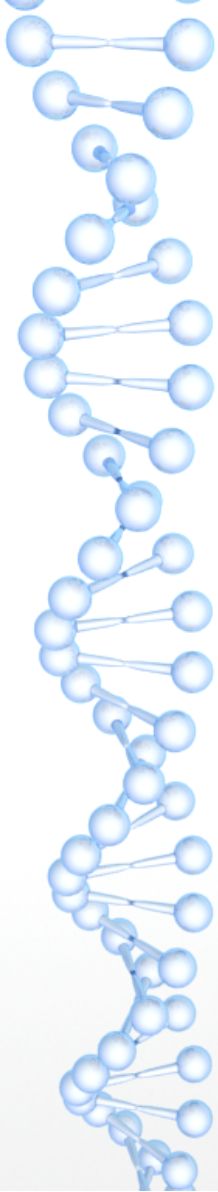




Зоопарк интеграций

- Проект развивался как стартап, архитектура - LAMP-стек. Интегрировались как можно быстрей, у каждого клиента были свои пожелания по интеграции, поэтому **каждая клиентская интеграция** — много кастомного PHP-кода.

Единичная интеграция



Extract

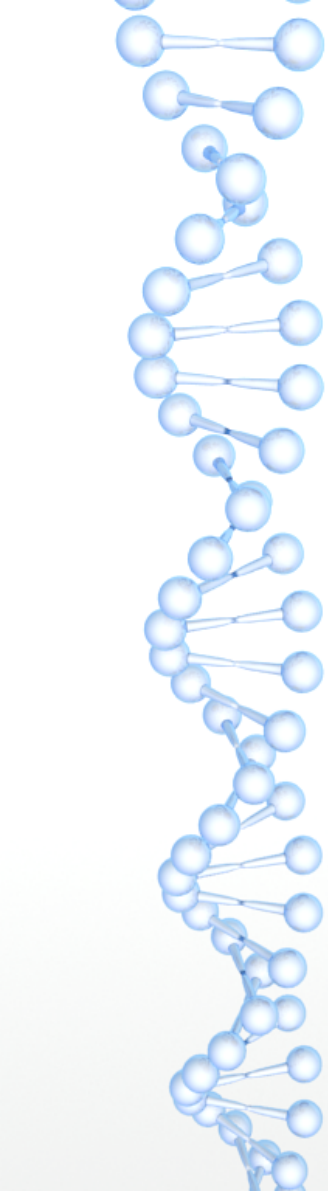
- авторизация клиента
- чтение порции данных
- регистрация клиентского запроса

Transform

- трансформация данных из модели клиента в модель VirtualHealth
- валидация внешних данных относительно внутренних данных

Load

- загрузка данных в БД системы

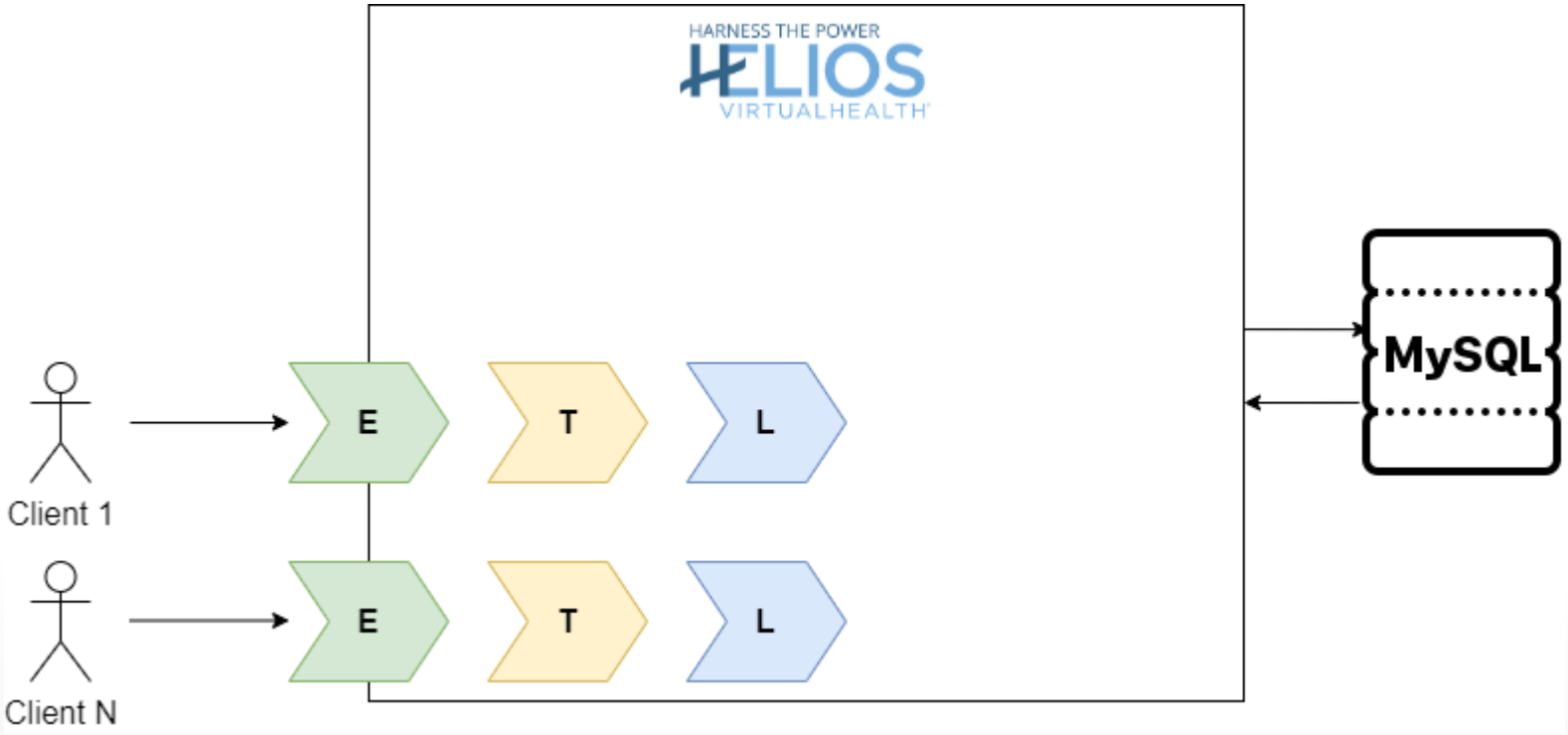


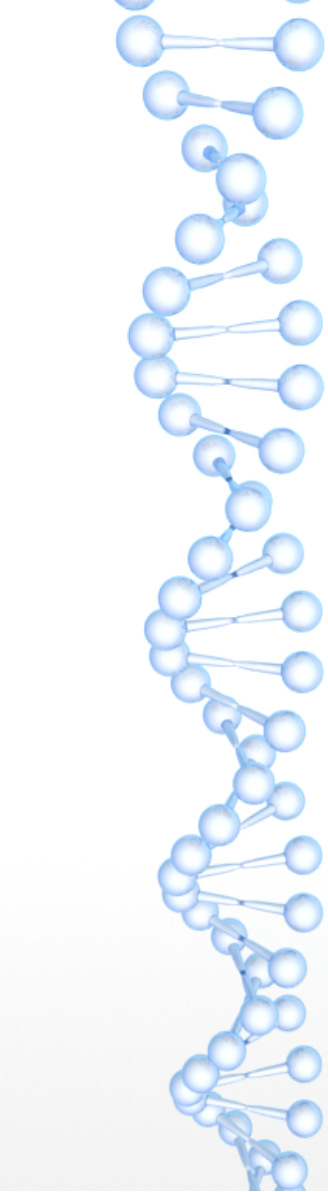
Для интеграции главное — строго однократная обработка

- Главное требование к процедуре обработки интеграционных данных — **соблюдение** семантики «exactly once».

- Обработка в режиме реального времени необязательна.

Интеграции как часть системы

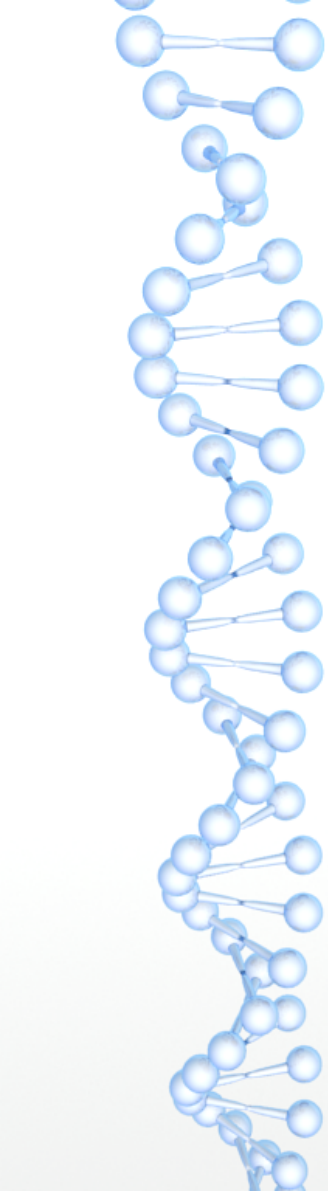




Переходим на Apache NiFi

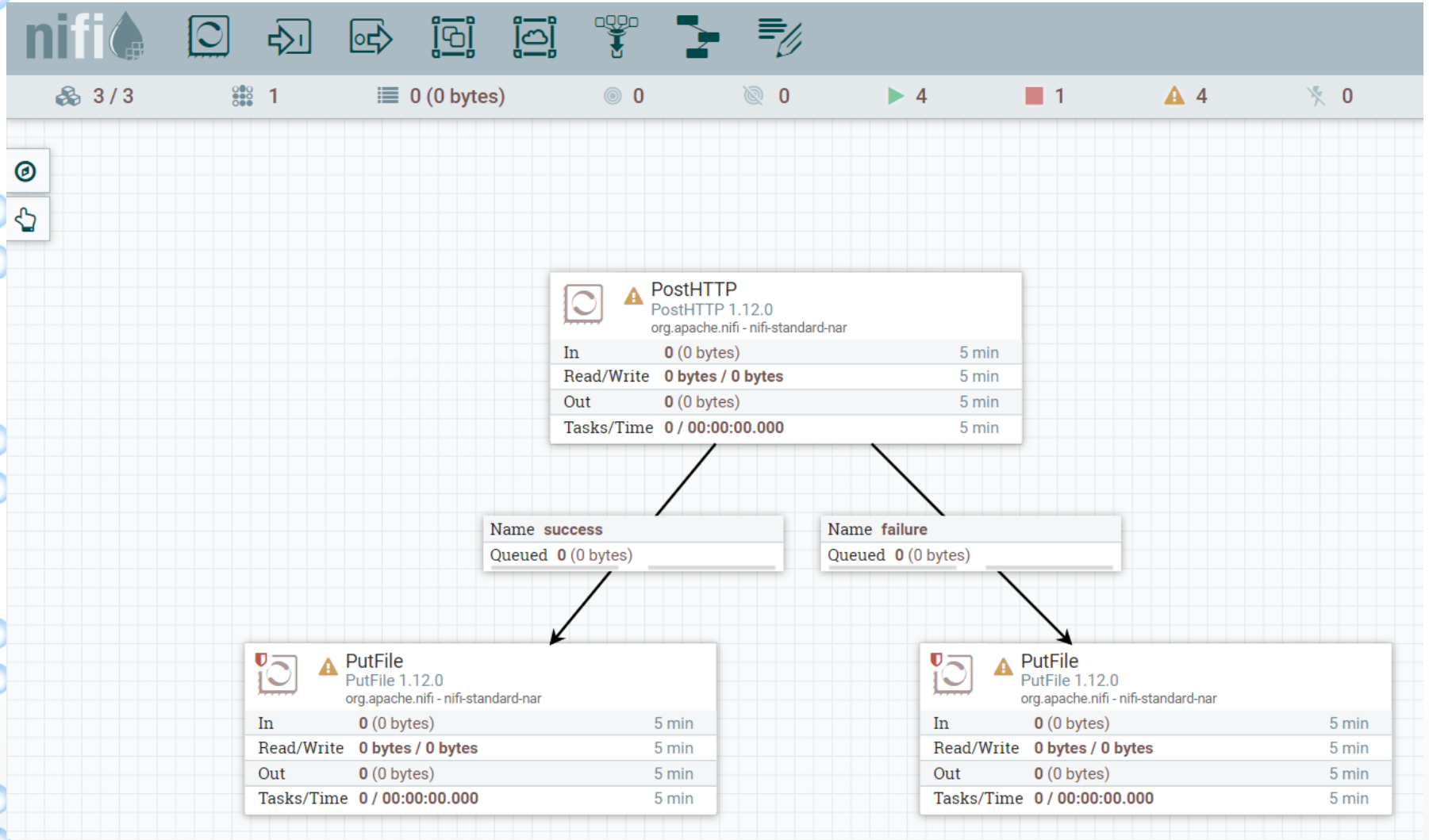
- С ростом числа клиентов стали видны **инфраструктурные задачи**, общие для всех интеграций. Apache NiFi удобным образом решает вопрос их создания и поддержки.

- С ростом числа клиентов **монолиту** становится всё хуже.

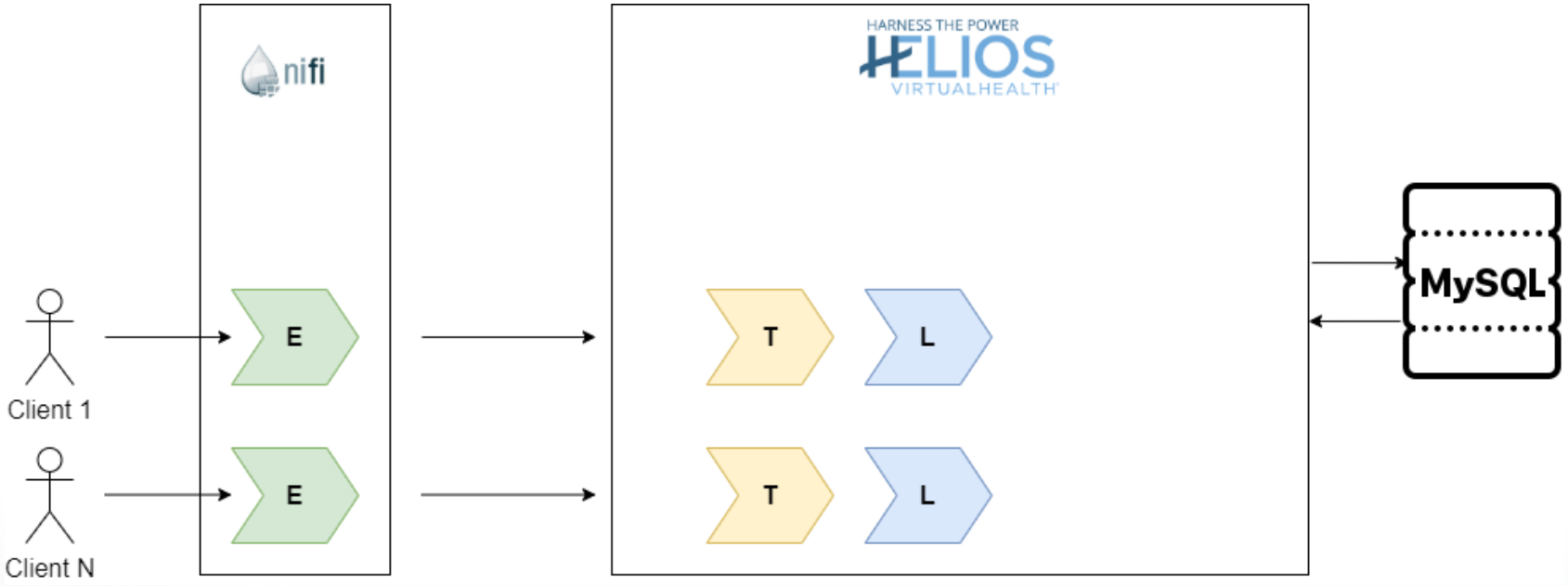


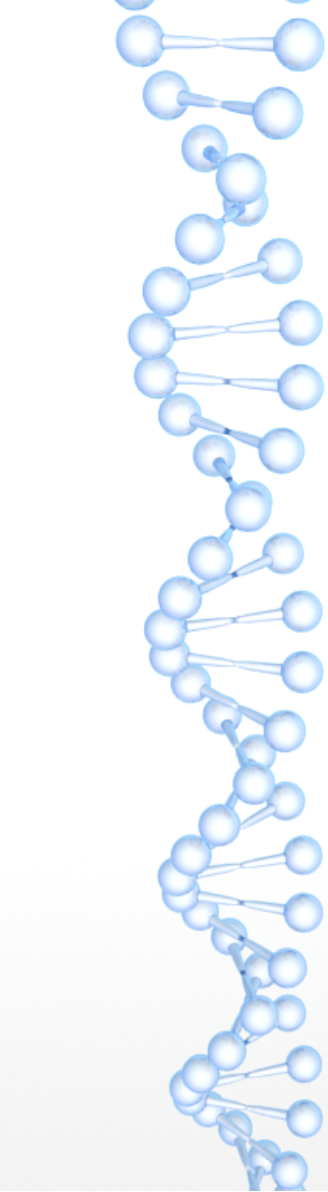
Apache NiFi

- платформа обработки событий, позволяющая управлять потоками данных из разнообразных источников в режиме реального времени с использованием графического интерфейса
- Основные элементы - **Flow Files** (потоки файлов), **Flow File Processors** (обработчики данных), **Connections** (передача данных от процессора к процессору)


nifi
3 / 3
1
0 (0 bytes)
0
0
4
1
4
0
PostHTTP
PostHTTP 1.12.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Name failure
Queued 0 (0 bytes)
PutFile
PutFile 1.12.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
PutFile
PutFile 1.12.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Переносим Extraction в NiFi



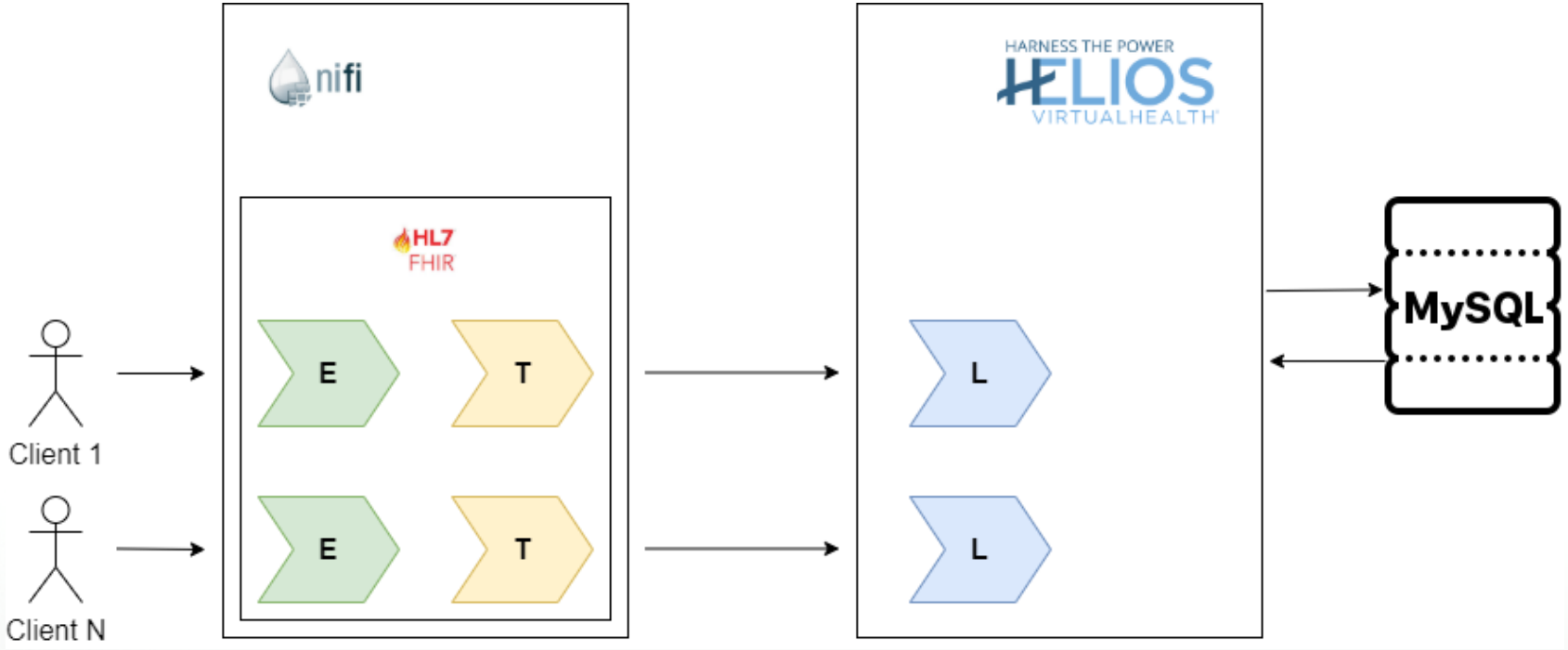


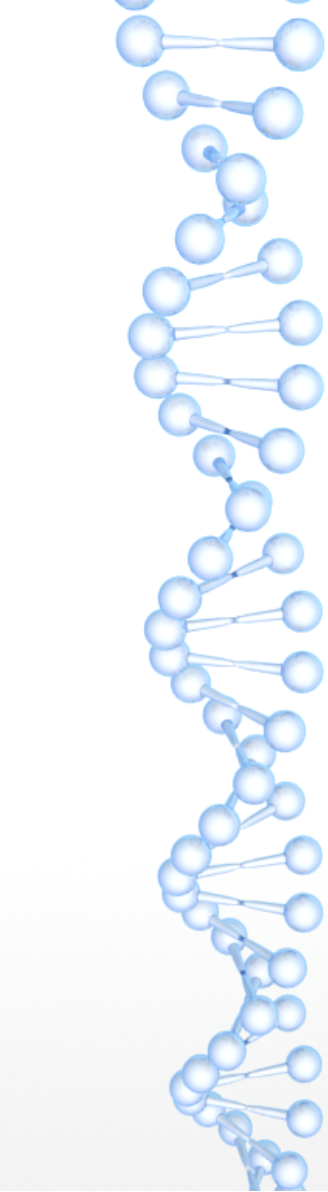
От множественных доменных моделей к HL7

- *HL7 — «всеобъемлющая» модель данных в области здравоохранения.*
- *HL7 FHIR — стандарт обмена медицинской информацией.*

- VirtualHealth **отказывается** от кастомных клиентских интеграций в пользу единой для всех интеграции на базе HL7 FHIR.

Переносим Transform в NiFi

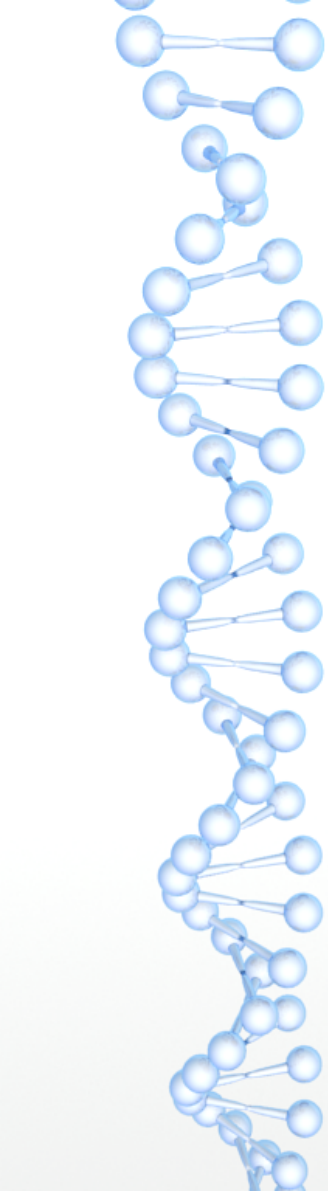




HL7 FHIR — оставляем только REST-обмен

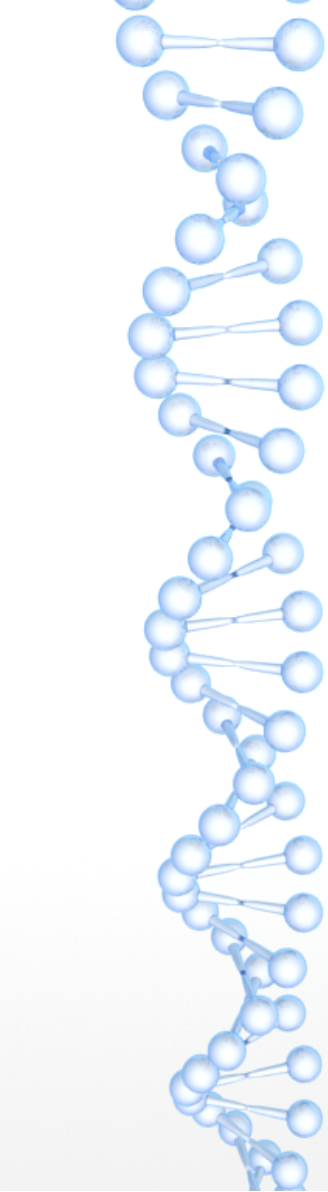
- Отказываемся от кастом-интеграций, в т.ч. от тех, которые используют persistent-источники данных (SFTP, AWS S3, …), загрузки по расписанию и оставляем только REST.

- Встают вопросы **обеспечения пропускной способности** и защиты от потери входящих данных **в процессе обработки**.



Добавляем очередь заданий

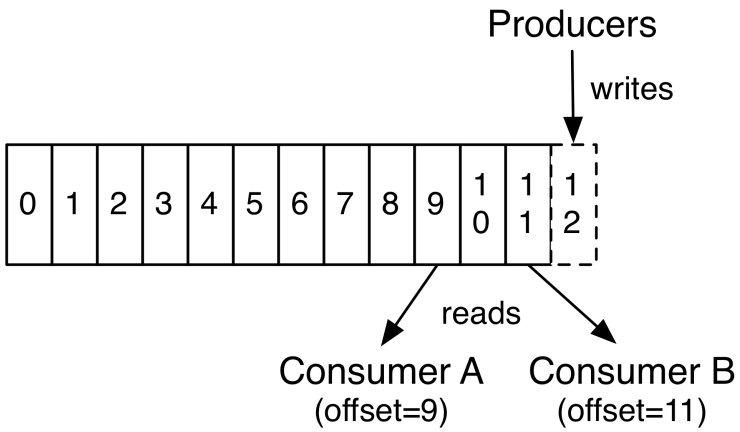
- Входящие интеграционные данные помещаем в персистентную очередь и обрабатываем их по мере возможности.
- В качестве очереди выбрали Apache Kafka. Плюсы: надежность, ~~производительность~~, *поддержка со стороны AWS*.
- Отвергнутые альтернативы:
- RDBMS - «овер-инжениринг».
- Файловые хранилища - «андер-инжениринг».
- Очереди сообщений (RabbitMQ, Apache Pulsar,



Kafka — платформа для создания и обработки потоков информации.

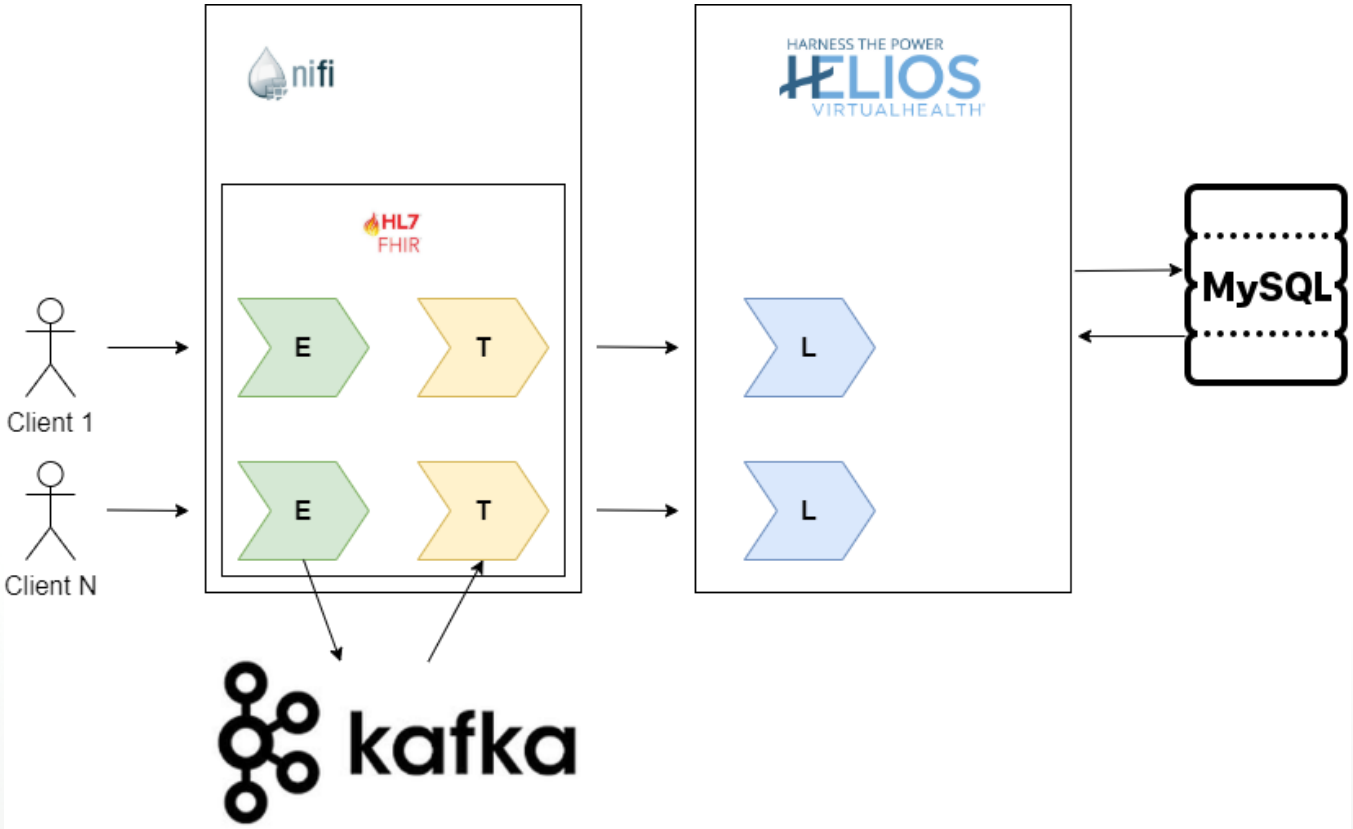
- **Журнал событий (event log)**, в который публикуются сообщения. Разделен на топики.
- **Продюсеры (producers, publishers)**, публикуют сообщения в журнал событий.
- **Консюмеры (consumers)**, потребляют сообщения из журнала событий.

Адресация сообщений Kafka

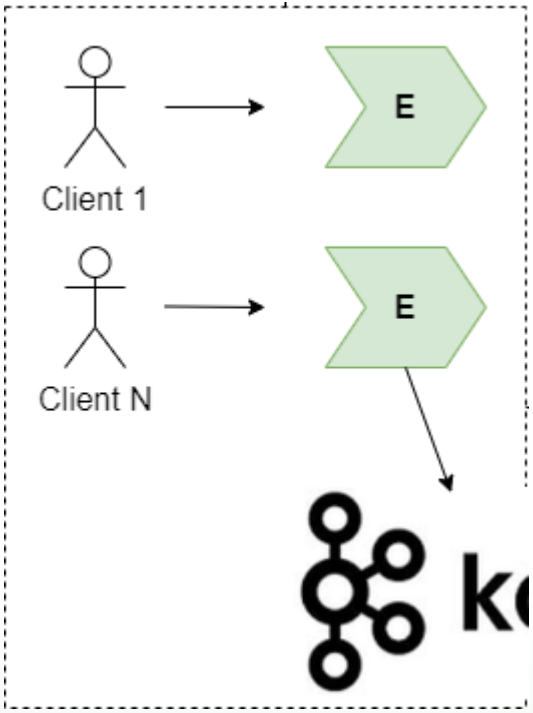


- Любое сообщение однозначно адресуется набором {Topic; Partition; Offset}
- Отслеживать офсет (указатель) чтения может как Consumer, так и

Kafka — **журнал** заданий.

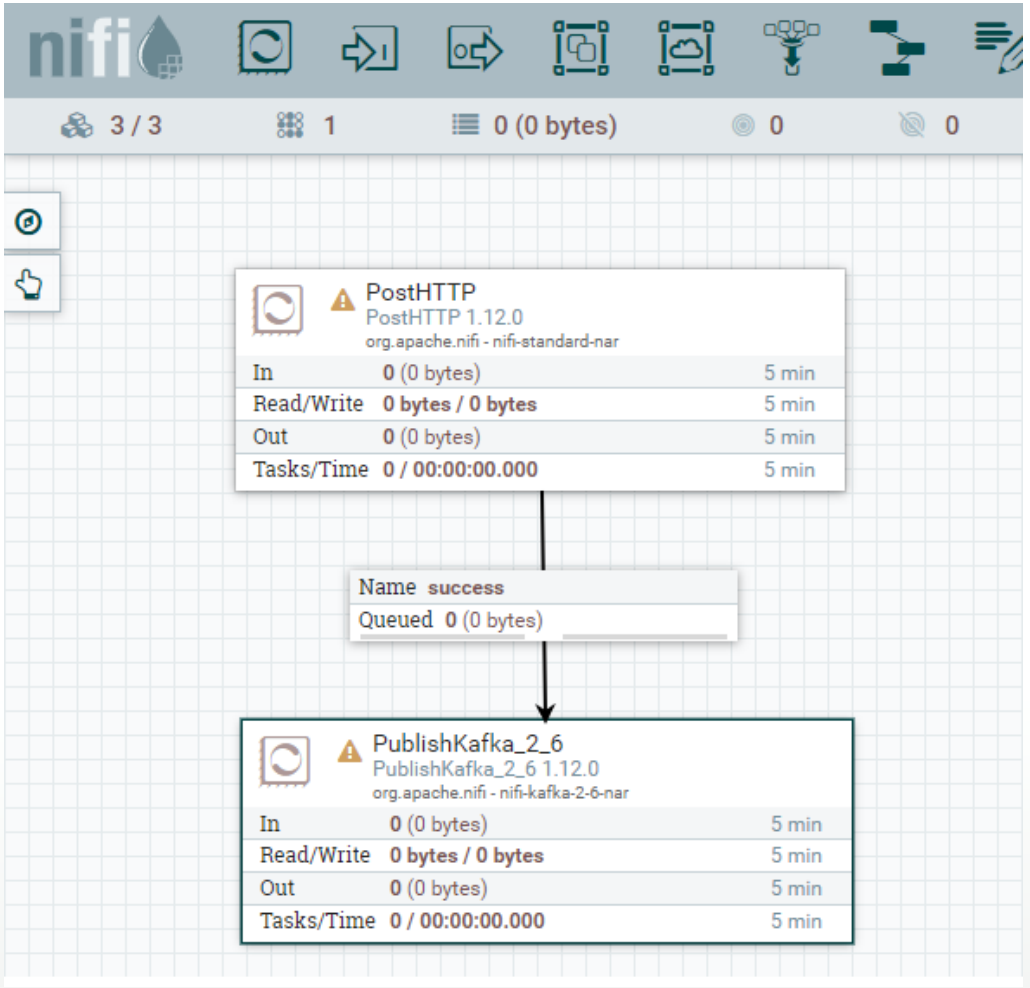


Exactly once при получении клиентских данных

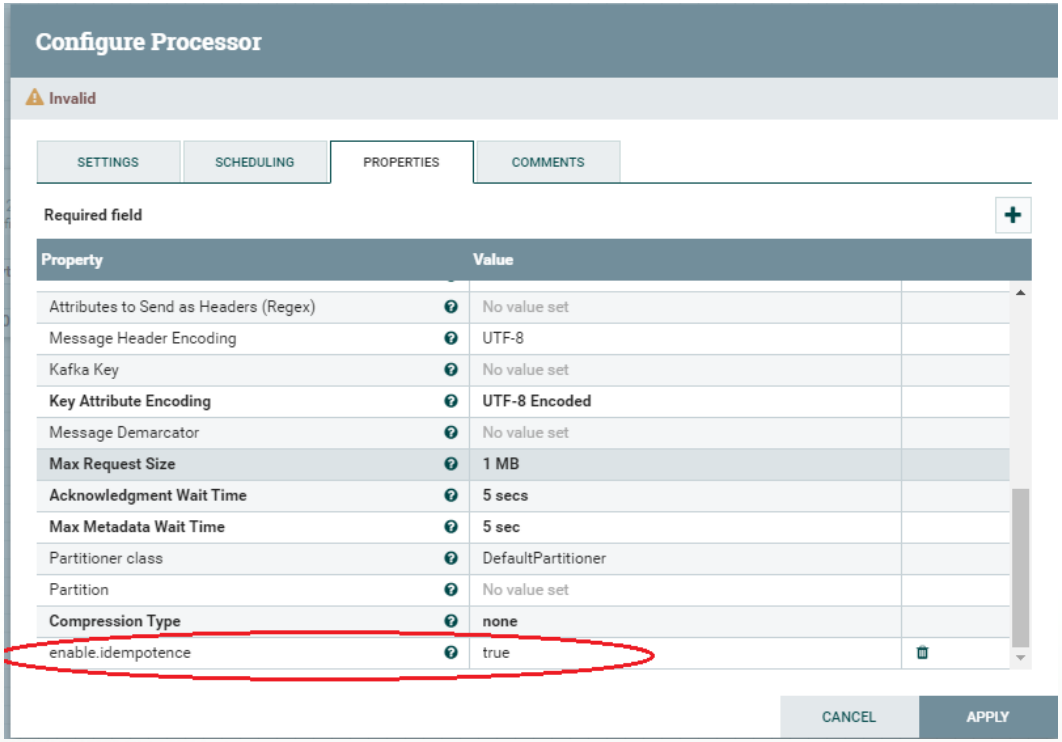


- Отдельный NiFi data-flow
- Устанавливаем enable.idempotence=true для Kafka-продюсера
- Отдаем ответ клиенту только после успешной записи в Kafka

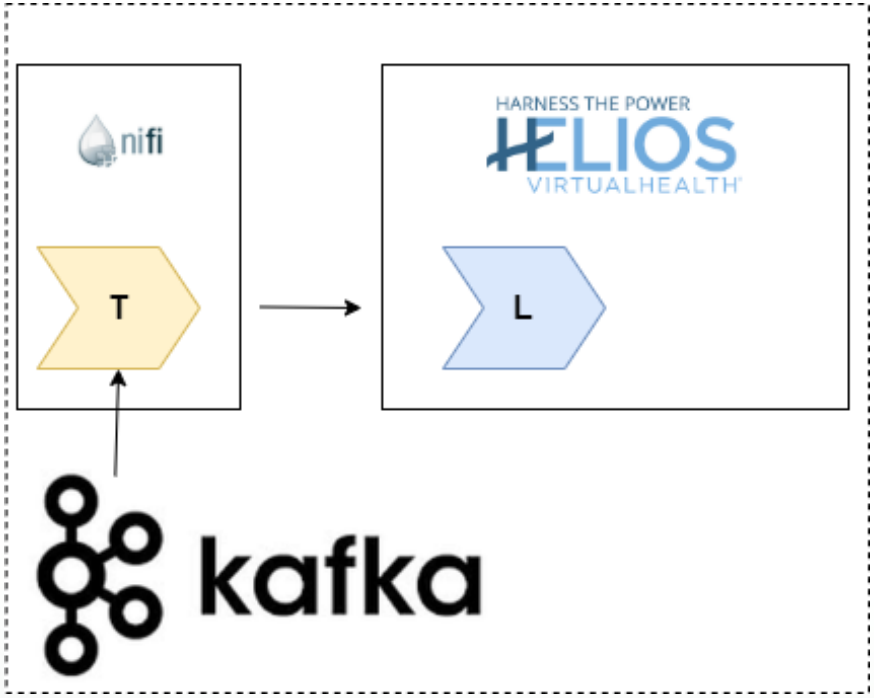
Ingress data-flow



KafkaProducer: enable.idempotence = true



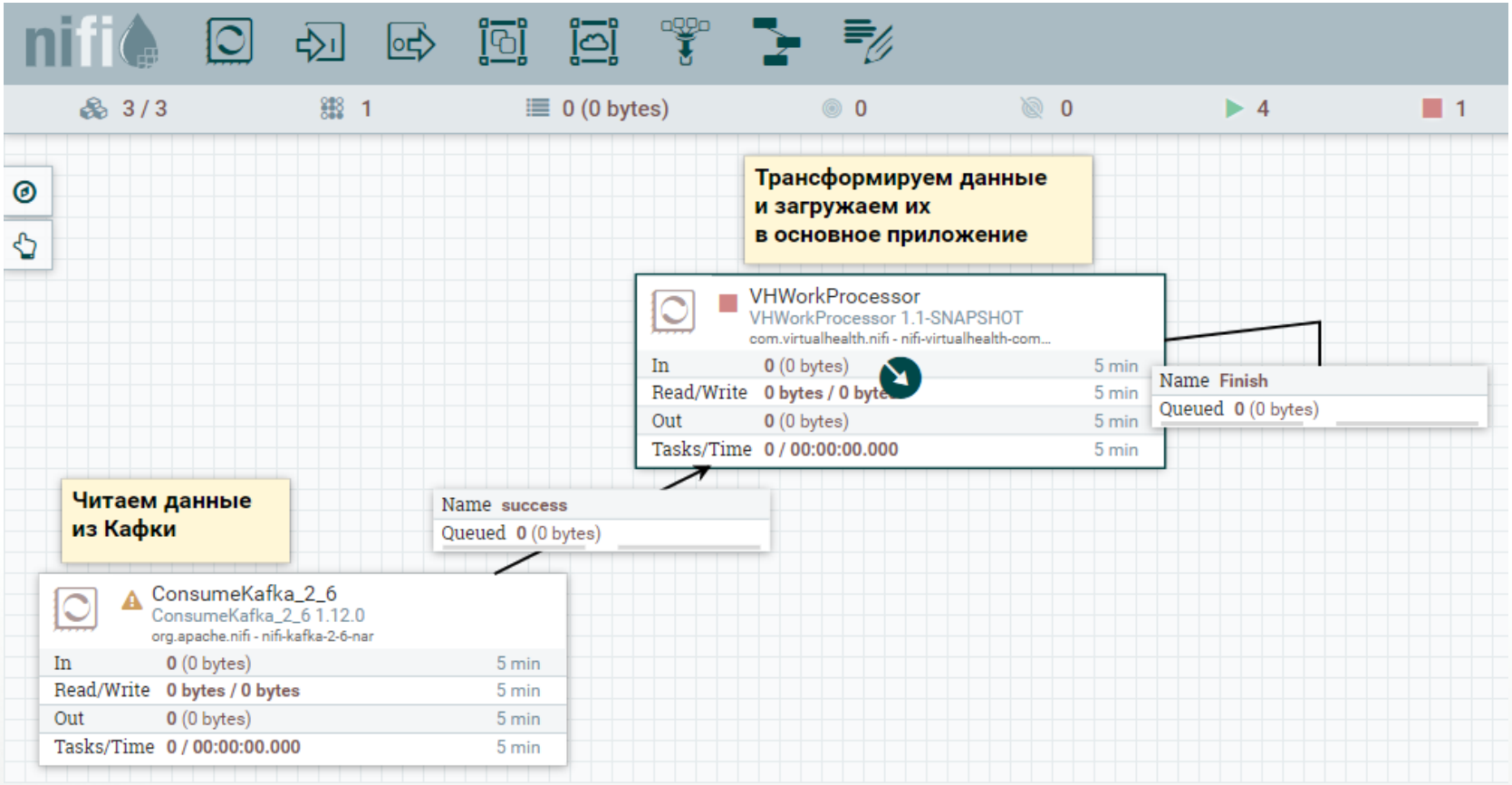
Обработка данных из очереди

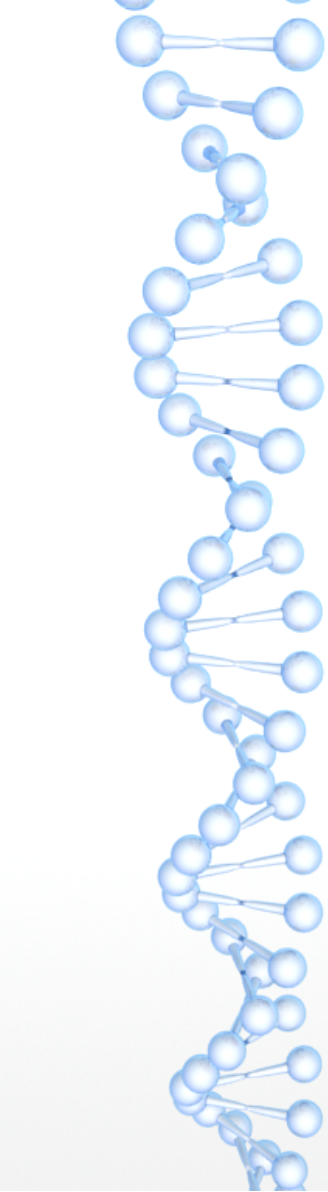


- Прочитать из Кафки
- Трансформировать
- Загрузить в основное приложение

Обрабатываем задачи из очереди

- Простой data-flow с дефолтным Kafka consumer.

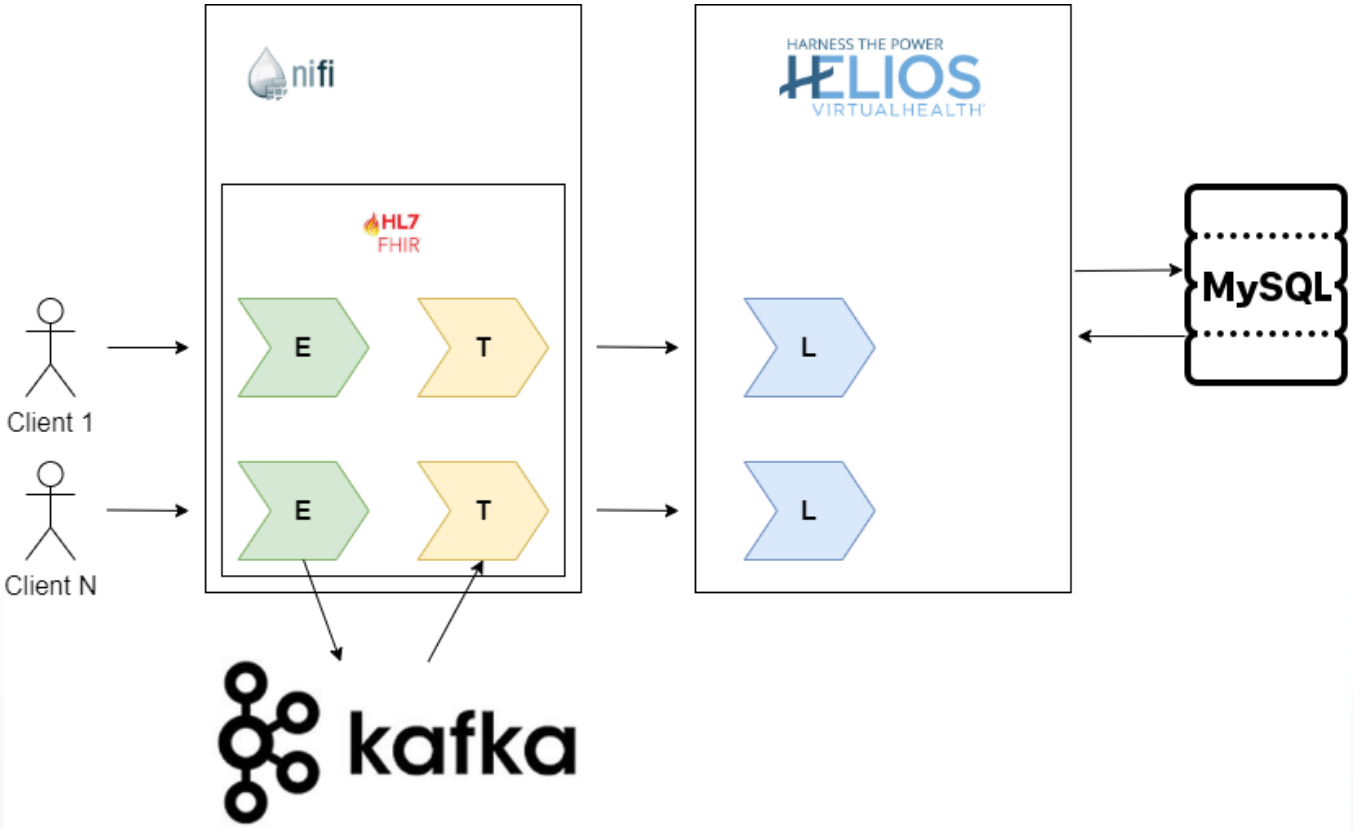


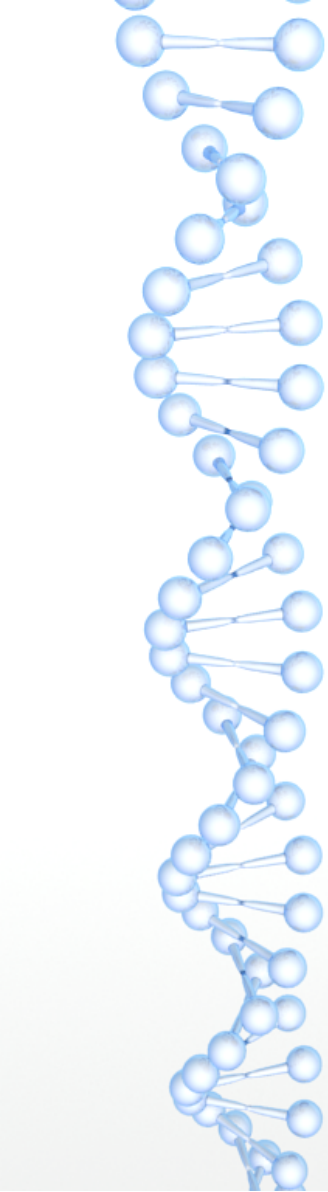


Data-flow с дефолтным Kafka consumer — плохой, негодный.

- Дефолтный NiFi-процессор ConsumeKafka_x_y - читает данные из топика, передает следующему обработчику внутри пайплайна и коммитит прочитанные сообщения.
- Нам это не подходит

Картина целиком

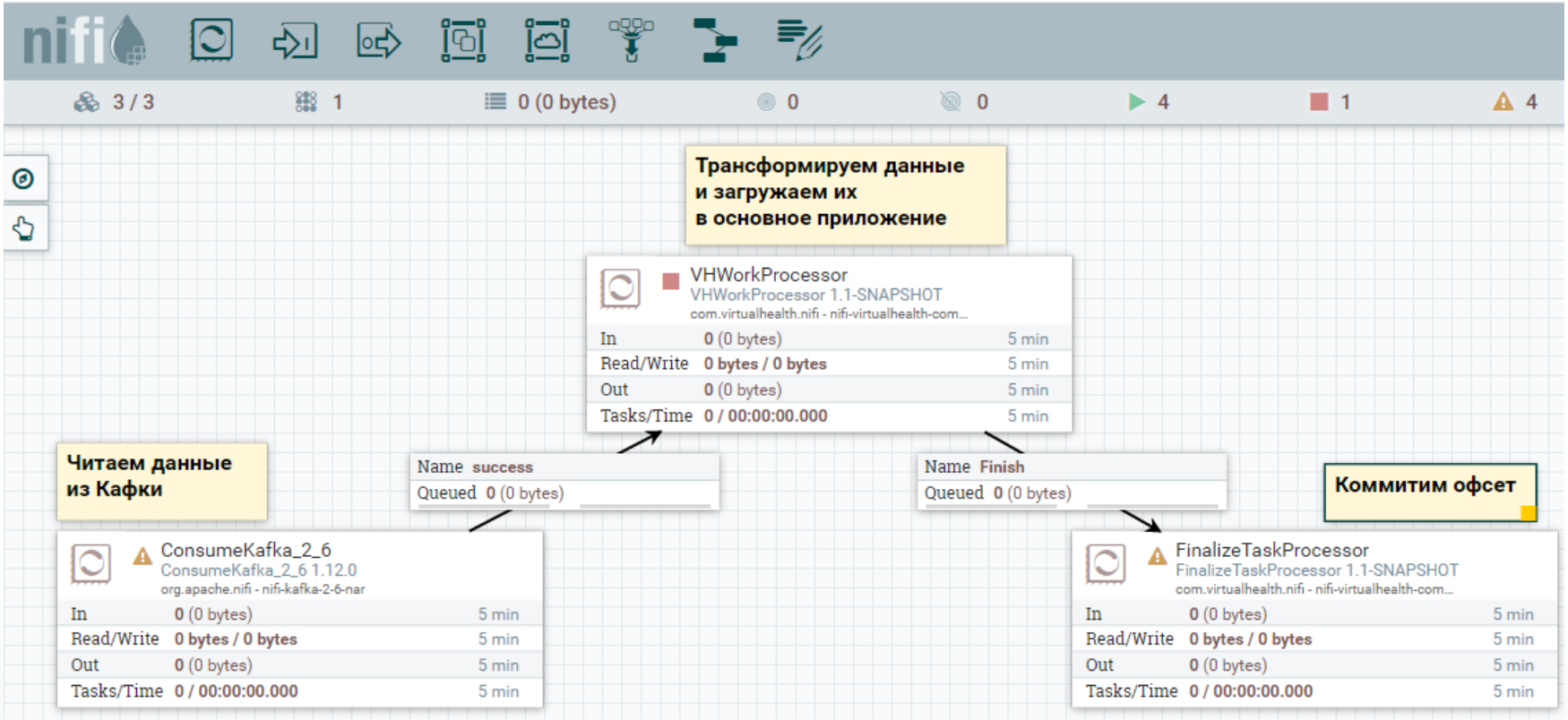




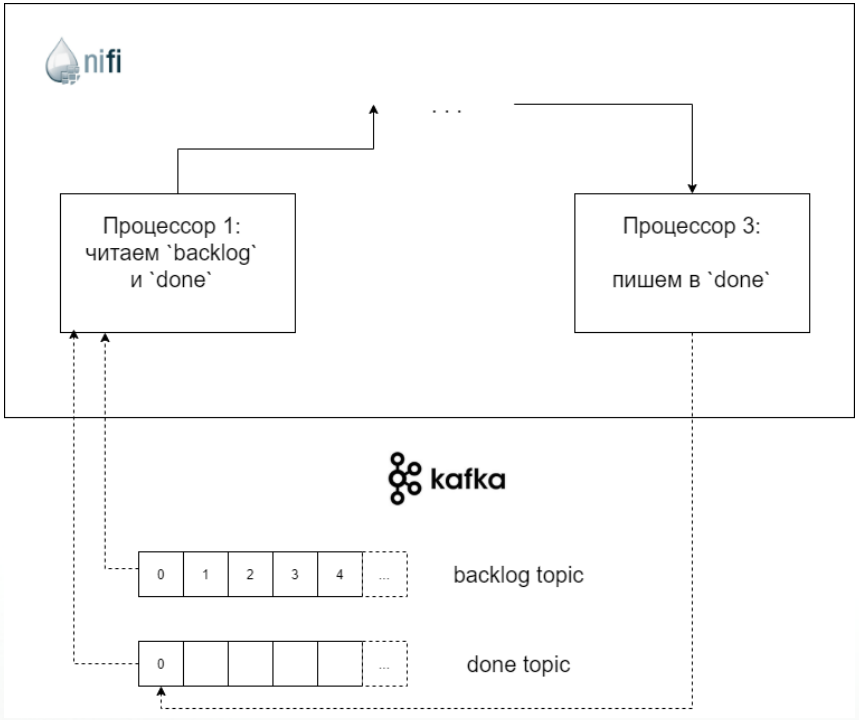
Боль реального использования

- Боль 1 — узел NiFi выходит из строя (OOM из-за некорректно реализованного процессора)
- Боль 2 — задачи, зависшие во внешней системе (PHP-монолите)
- Боль 3 — отсутствие back-pressure на уровне логики NiFi-процессора

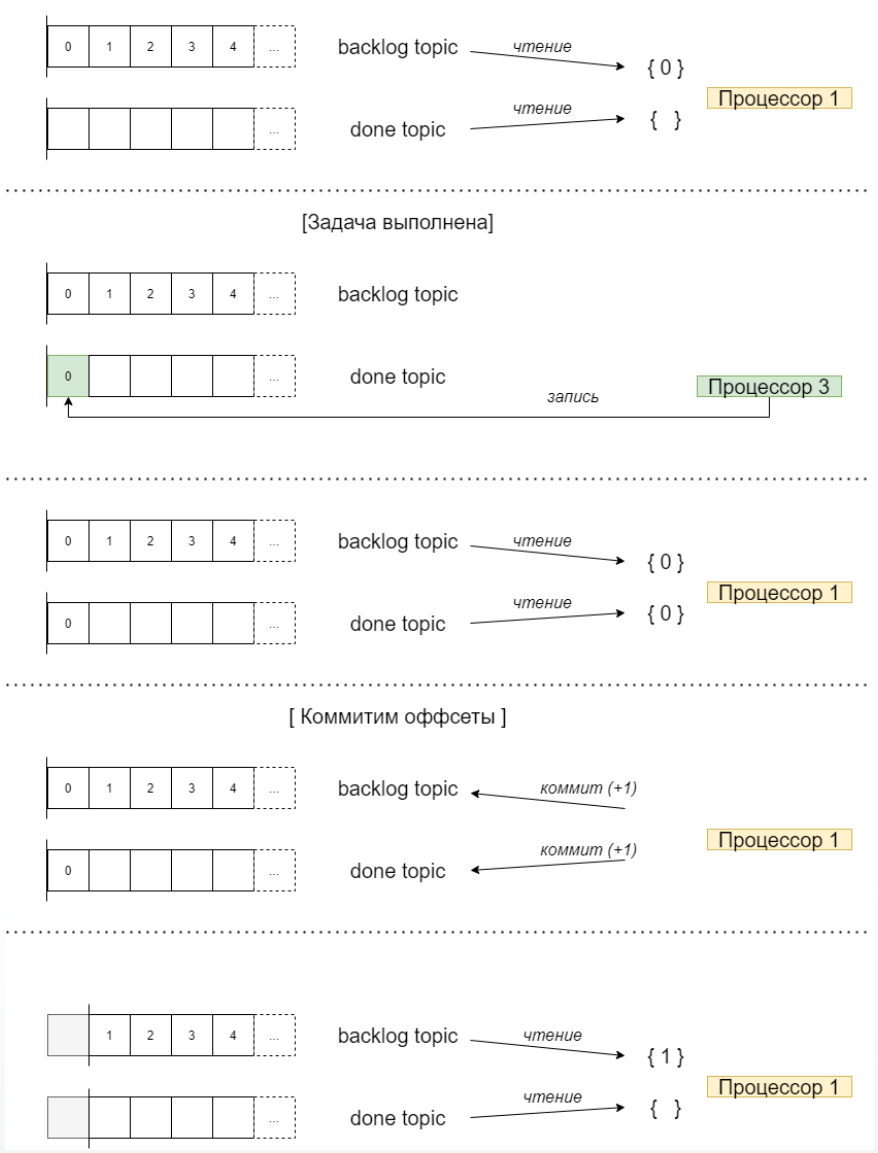
Коммитим оффсет после обработки

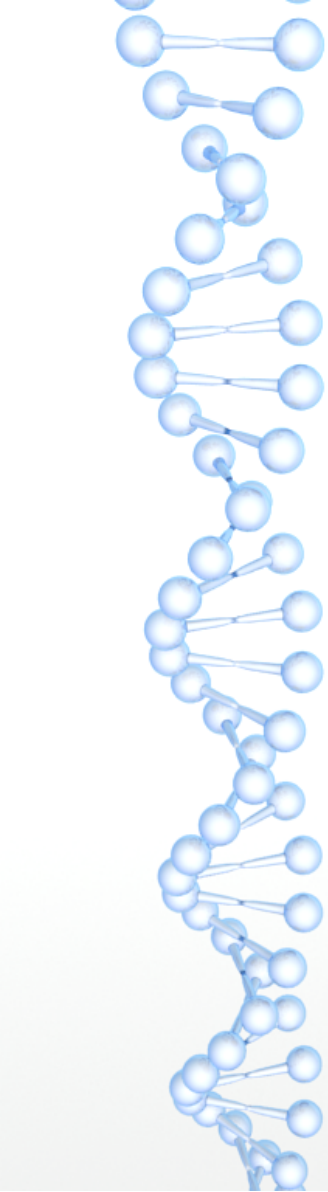


Очередь выполненных задач `done`



- **Цикл работы `Процессора 1`:**
- Читает из `backlog` и из `done` по 1 сообщению
- Коммитит их оффсеты, после того, как они появились в `done`


0 1 2 3 4 ...
backlog topic
чтение
{ 0 }
done topic
чтение
{ }
Процессор 1
[Задача выполнена]
backlog topic
done topic
запись
Процессор 3
[Коммитим оффсеты]
коммит (+1)
коммит (+1)
{ 1 }

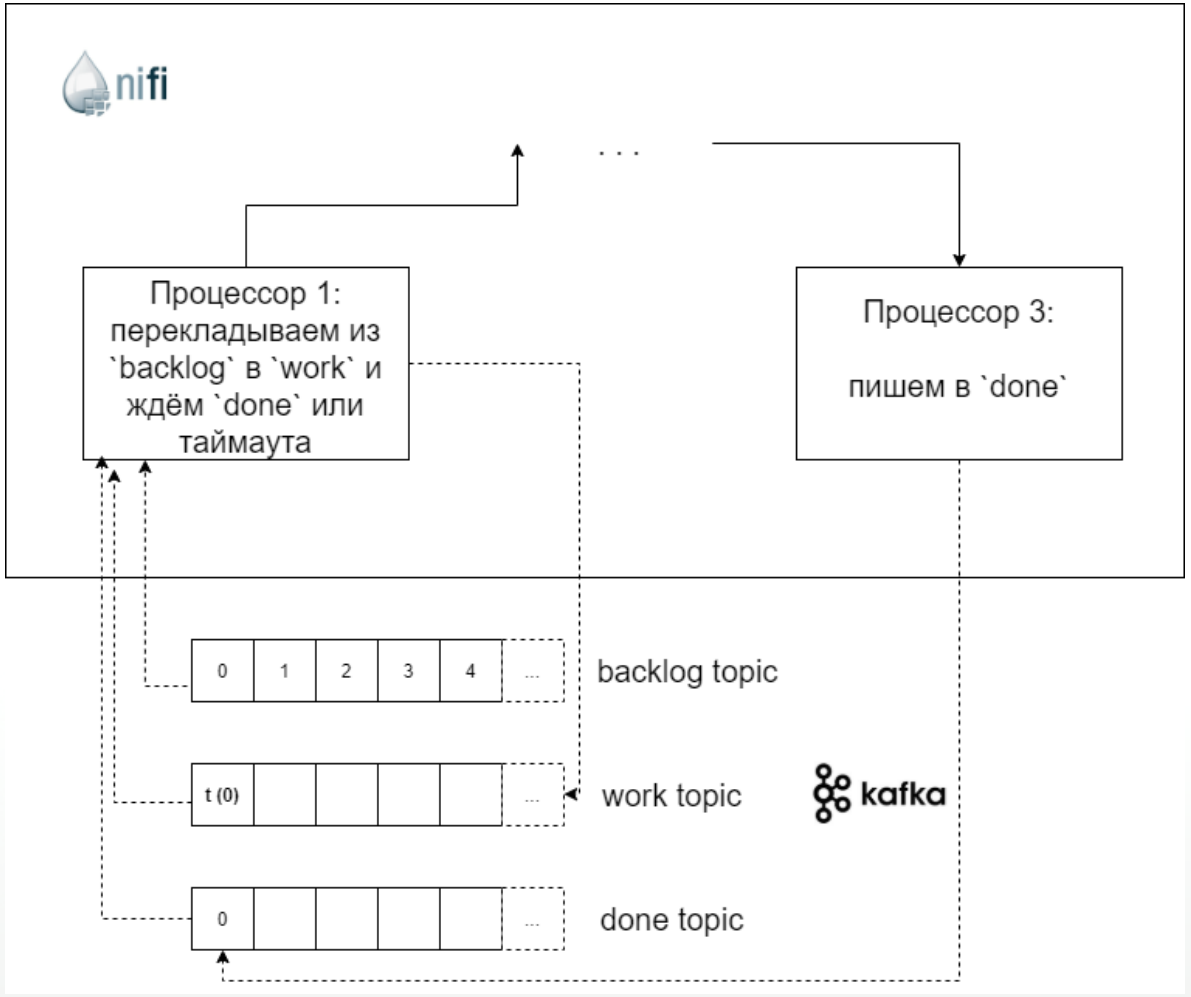


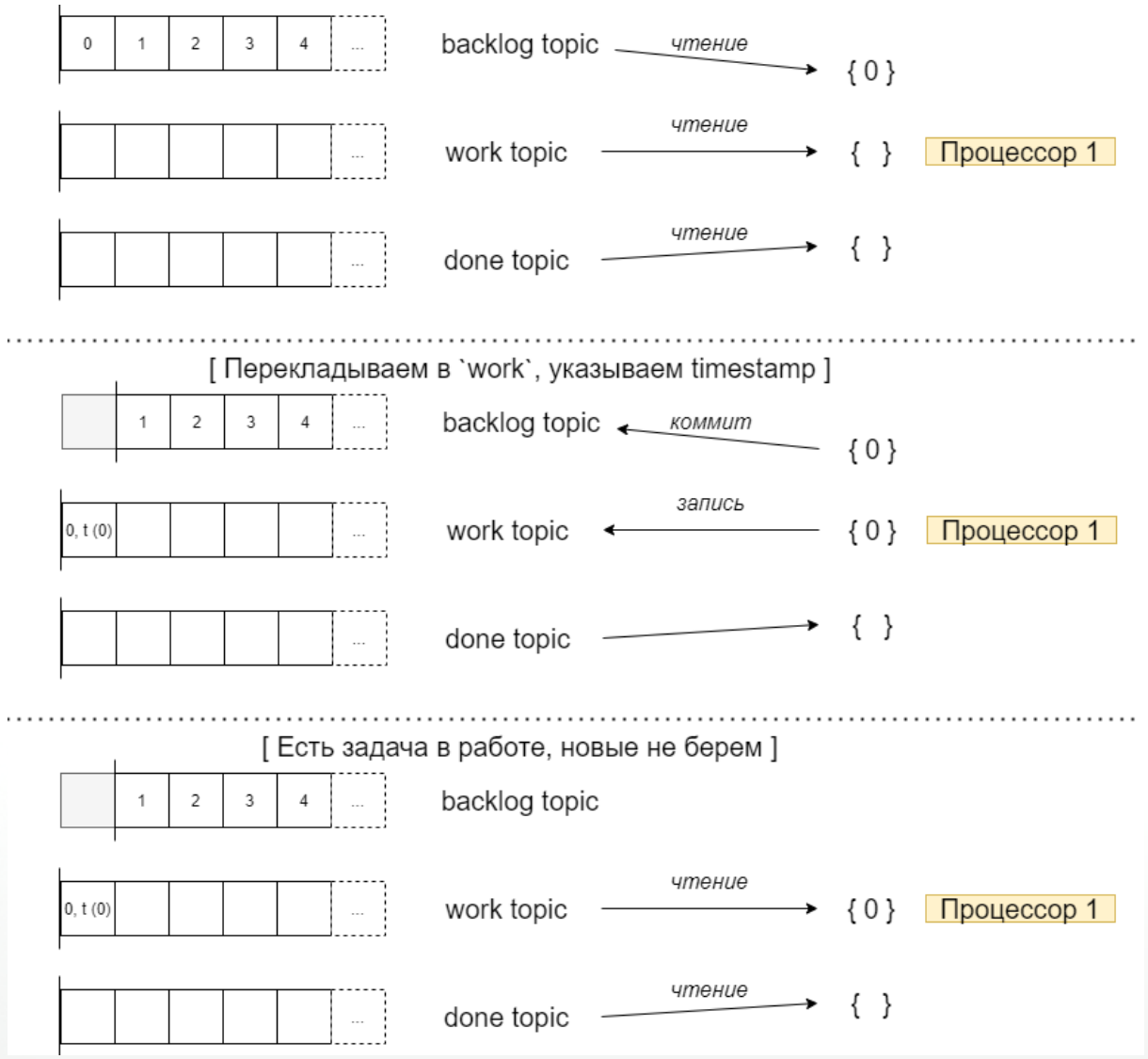
Решаем проблему зависших задач

- Ведем журнал задач, взятых в работу
- Храним метку времени начала обработки

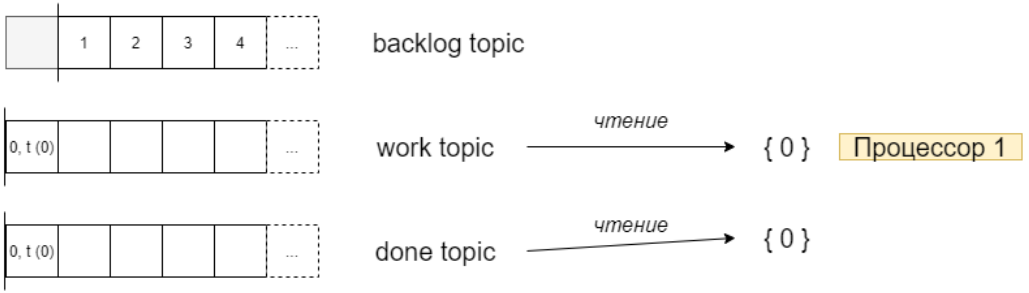
- * За идемпотентность обработки отвечает основная система (PHP-монолит)

`work` - журнал текущих задач

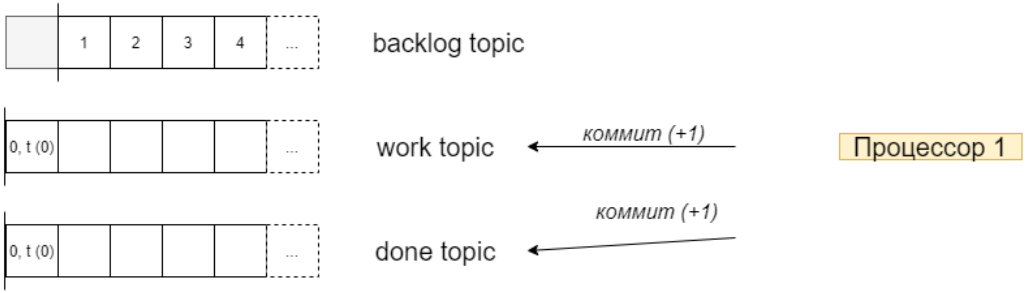



0 1 2 3 4 ...
backlog topic
чтение
{ 0 }
work topic
чтение
{ }
Процессор 1
done topic
чтение
{ }
[Перекладываем в `work`, указываем timestamp]
1 2 3 4 ...
backlog topic
коммит
{ 0 }
0, t (0)
work topic
запись
{ 0 }
Процессор 1
done topic
{ }
[Есть задача в работе, новые не берем]
1 2 3 4 ...
backlog topic
0, t (0)
work topic
чтение
{ 0 }
Процессор 1
done topic
чтение
{ }

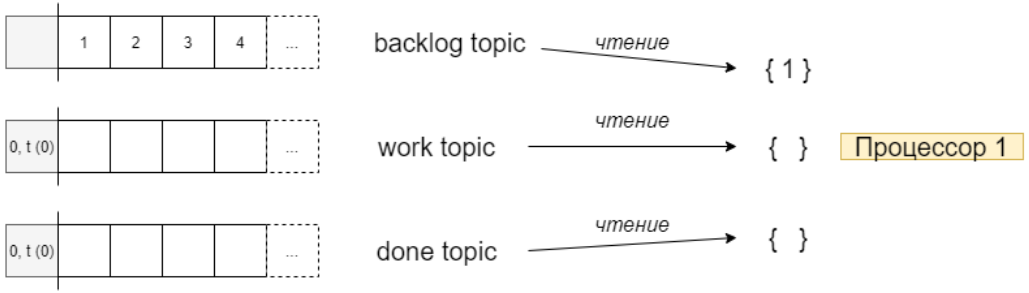
[Сценарий 1, задача выполнена]



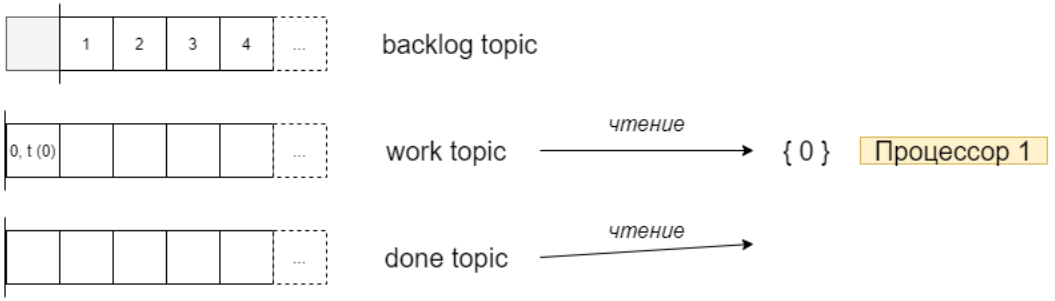
[Коммитим оффсеты]



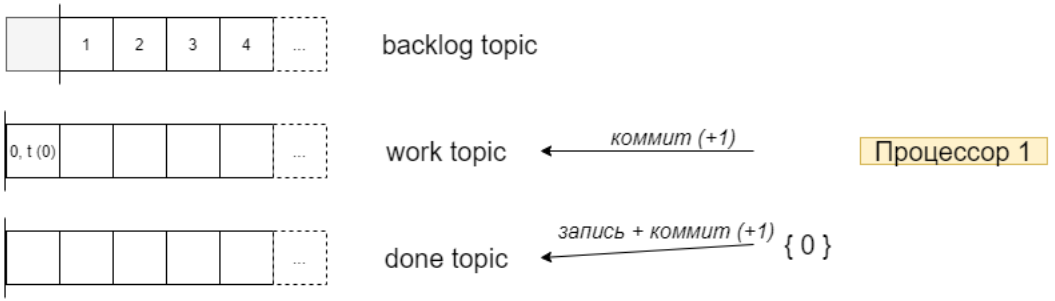
[Нет задач в работе, берем новую из `backlog`]



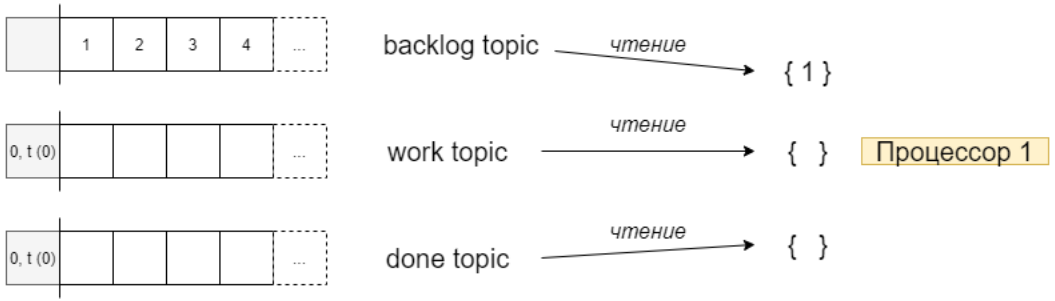
[Сценарий 2, задача зависла]

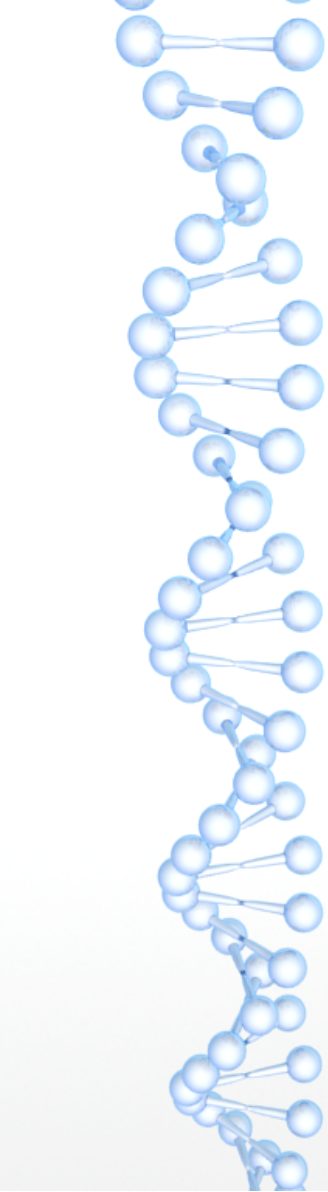


[Помещаем в `done` принудительно, коммитим оффсеты]



[Нет задач в работе, берем новую из `backlog`]

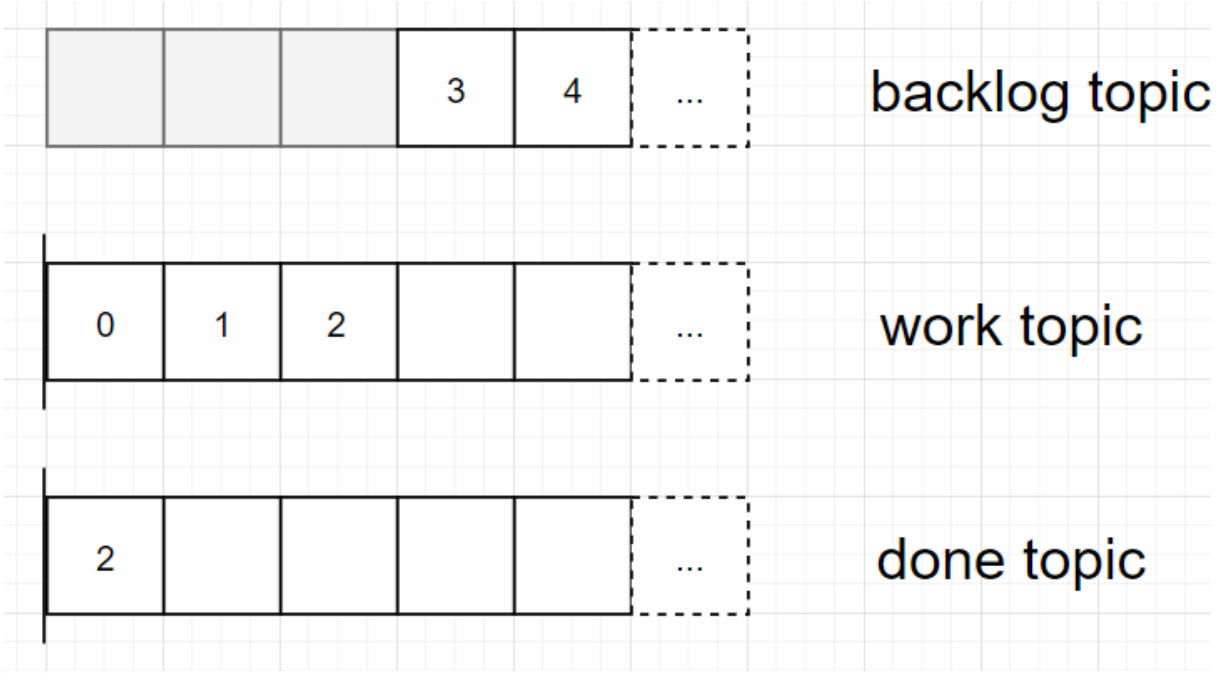




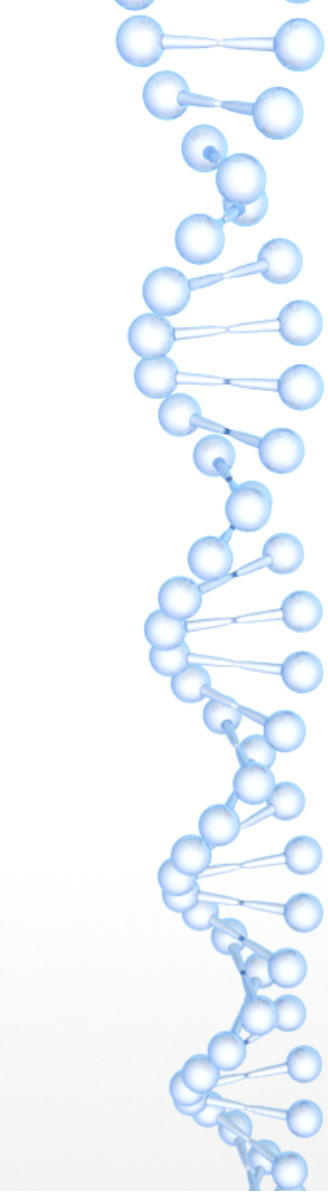
Многопоточная обработка

- Несколько задач выполняются параллельно
- Обработка разных задач занимает разное время
- Задачи завершаются не в том порядке, в котором взяты в работу

Старый алгоритм не работает

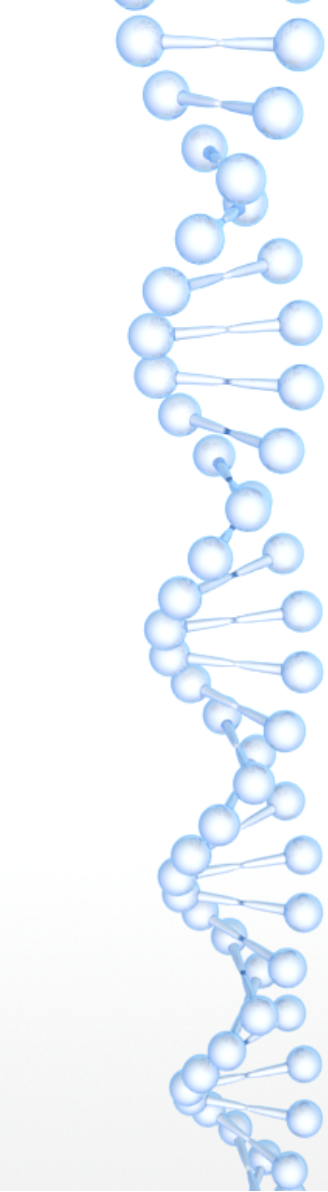


1. Задача 2 выполнена, но «забывать» её нельзя!
2. Освободился один поток обработки, надо взять ещё одну задачу.



Требования к многопоточной обработке

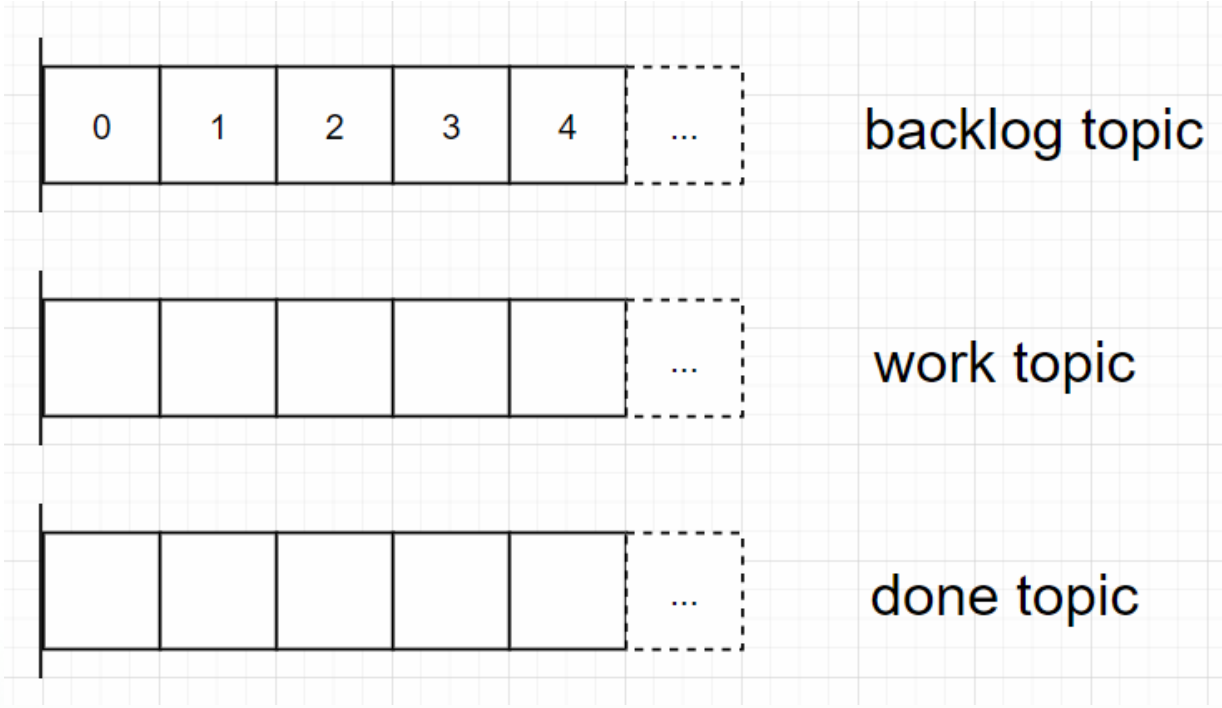
- Необходимо брать новые задачи, когда есть свободные потоки
- Пропускная способность обработки ограничена и заранее известна (*maxThreadsAmount*)
- Задачи в `work` необходимо коммитить по порядку
- Задачи в `done` появляются в произвольном порядке



Обобщаем алгоритм

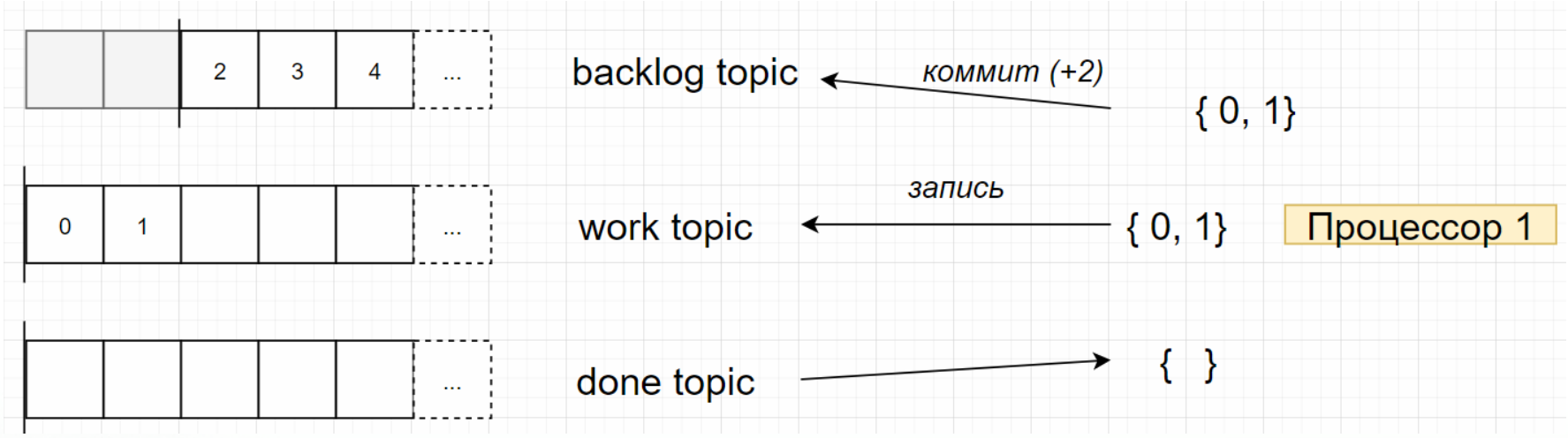
- Читаем `work` и `done`, затем:
- 1. Обходим `work` в поисках кандидатов на «забывание»
- 2. Обходим `done` в поисках кандидатов на «забывание»
- 3. Проверяем, появились ли свободные пототки, чтобы взять в работу новые задачи

Трассируем обработку в 2 потока

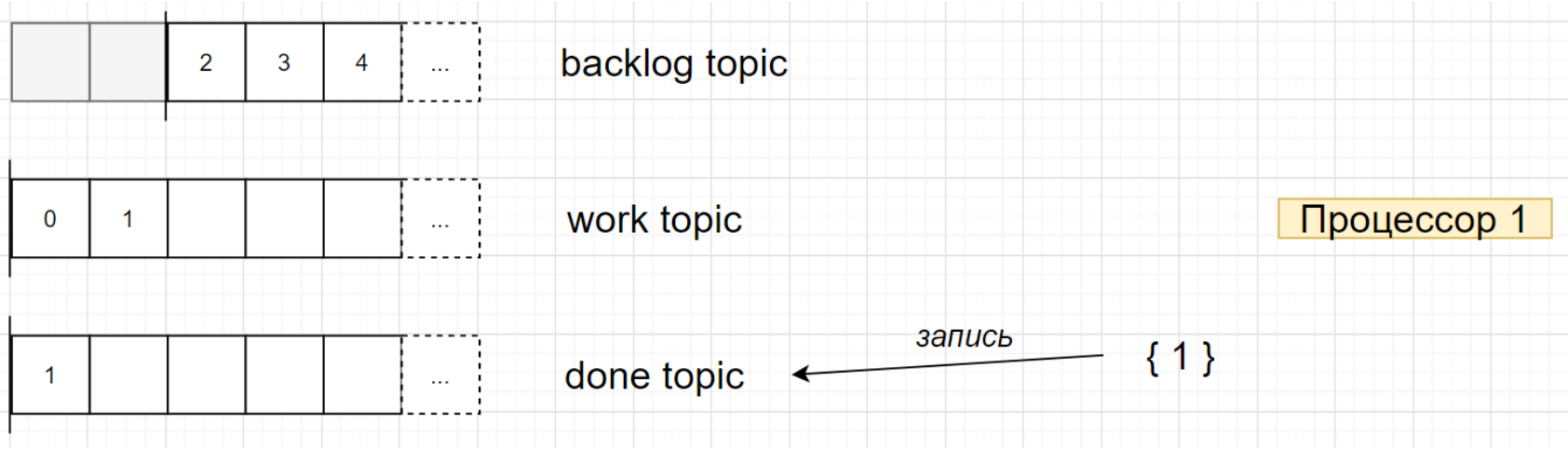


Согласно алгоритма — коммитить нечего, но можем взять в работу 2 задачи

Взяли в работу 2 задачи

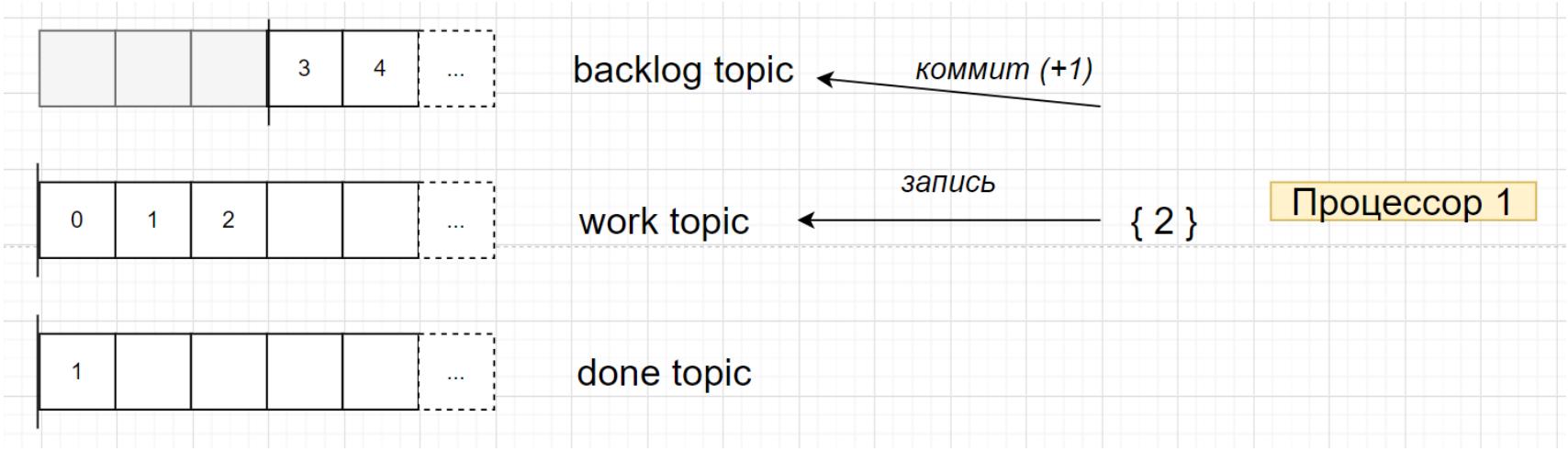


Выполнена задача 1

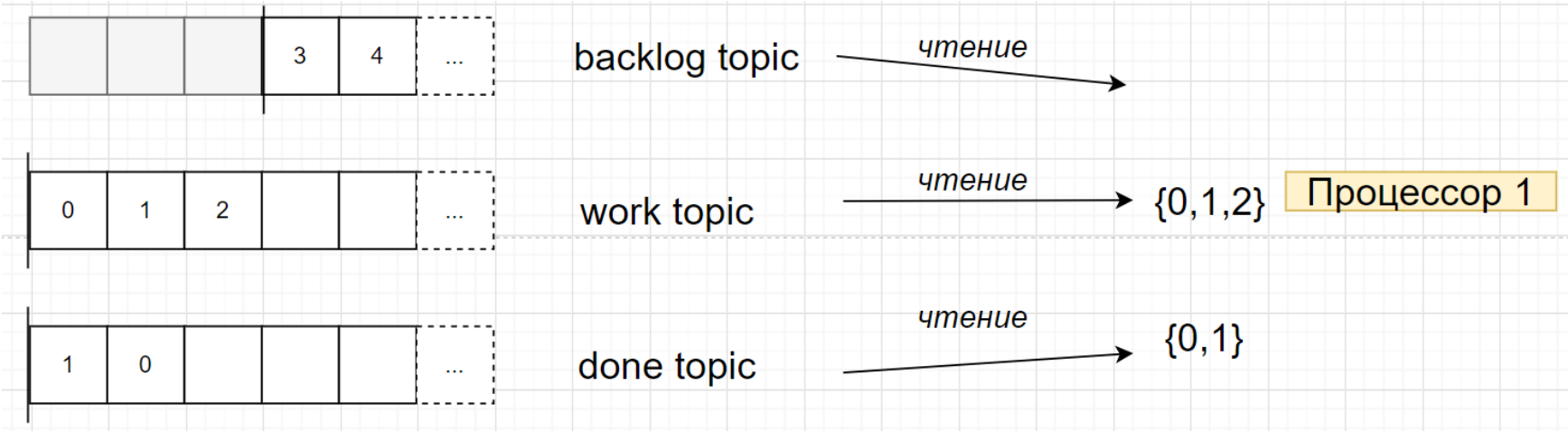


Офсеты не коммитим, но можем взять в работу ещё 1 задачу

Берем в работу задачу 2

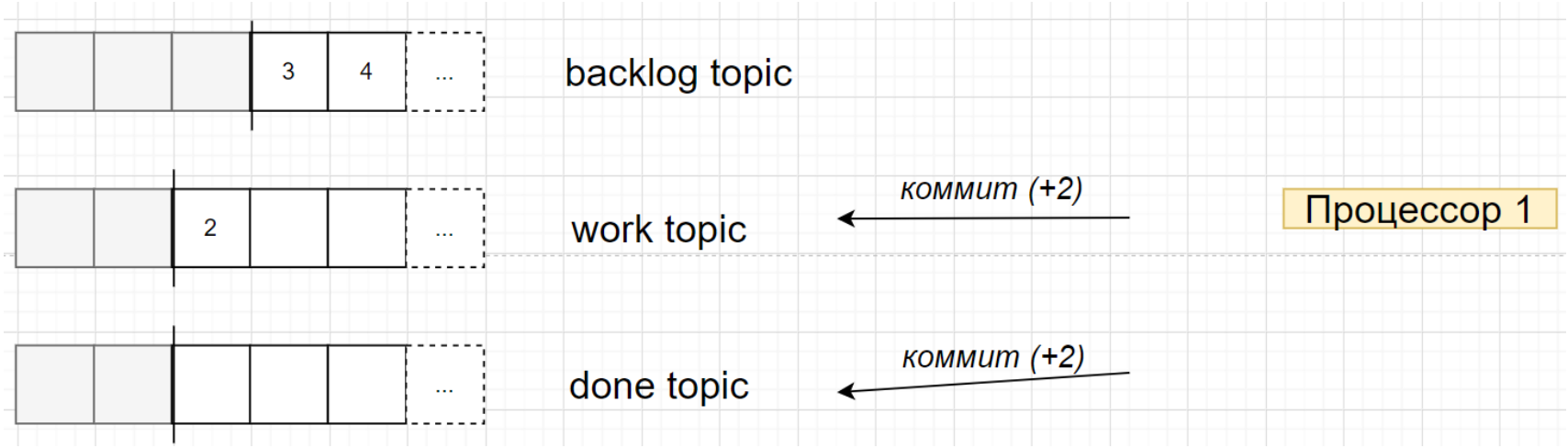


Выполнена задача 0

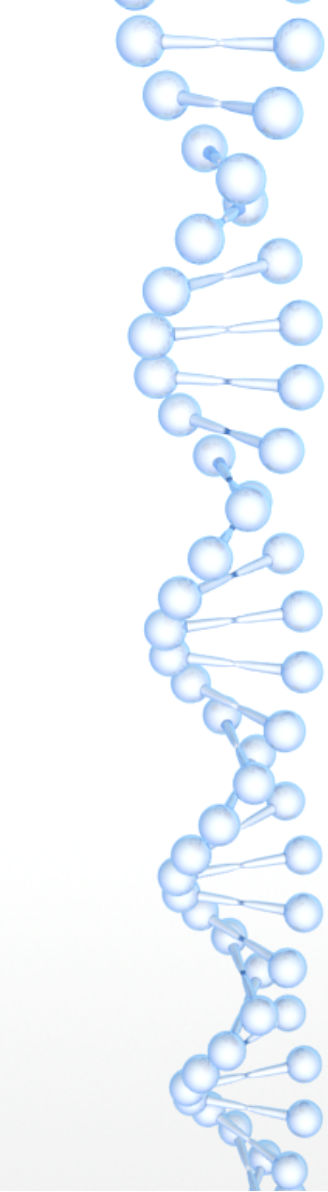


Согласно алгоритма можем сдвинуть указатели для задач 0 и 1 в `work` и `done`.

Задачи 0 и 1 полностью обработаны.



Рабочие очереди освобождены от выполненных задач.
Продолжаем работу аналогично (пока есть задачи в `backlog`).



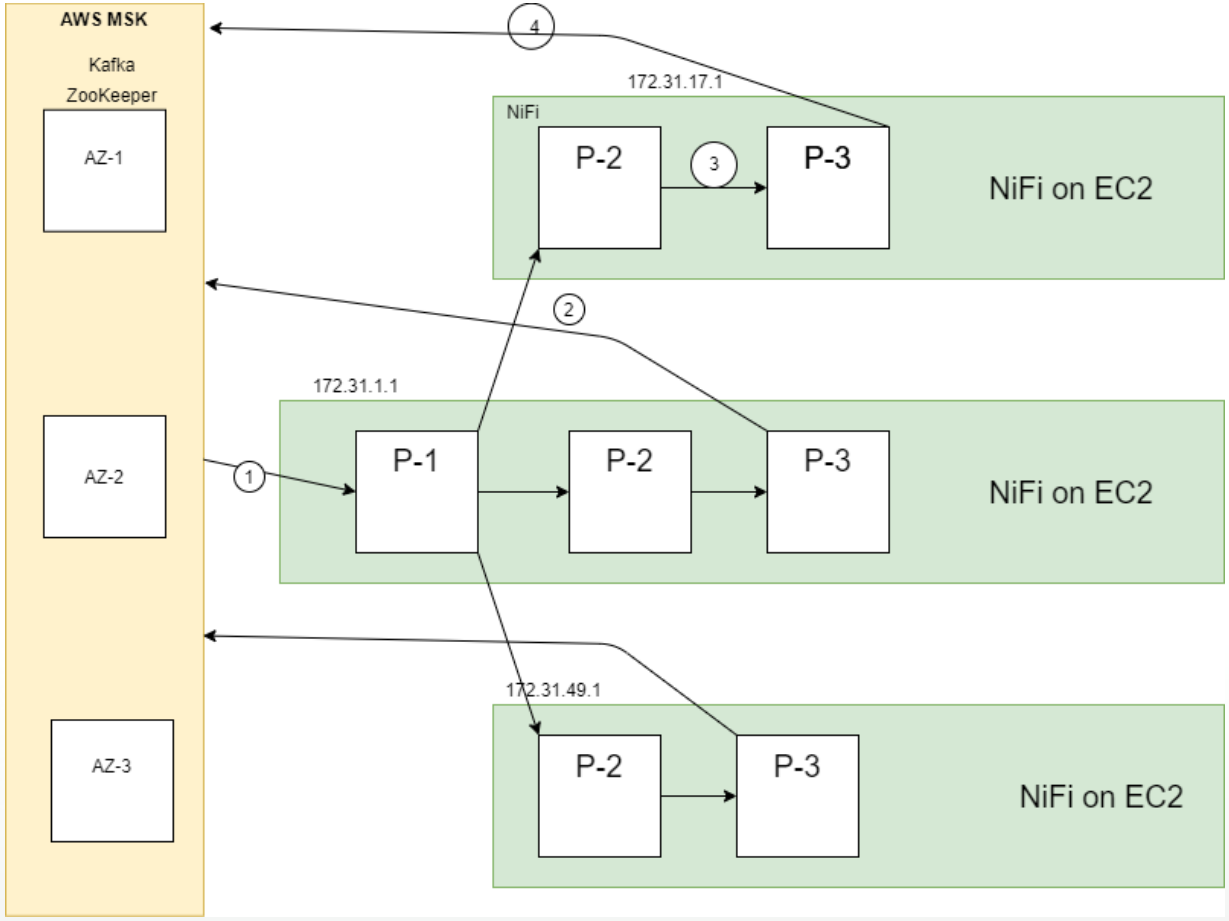
Алгоритм для N потоков

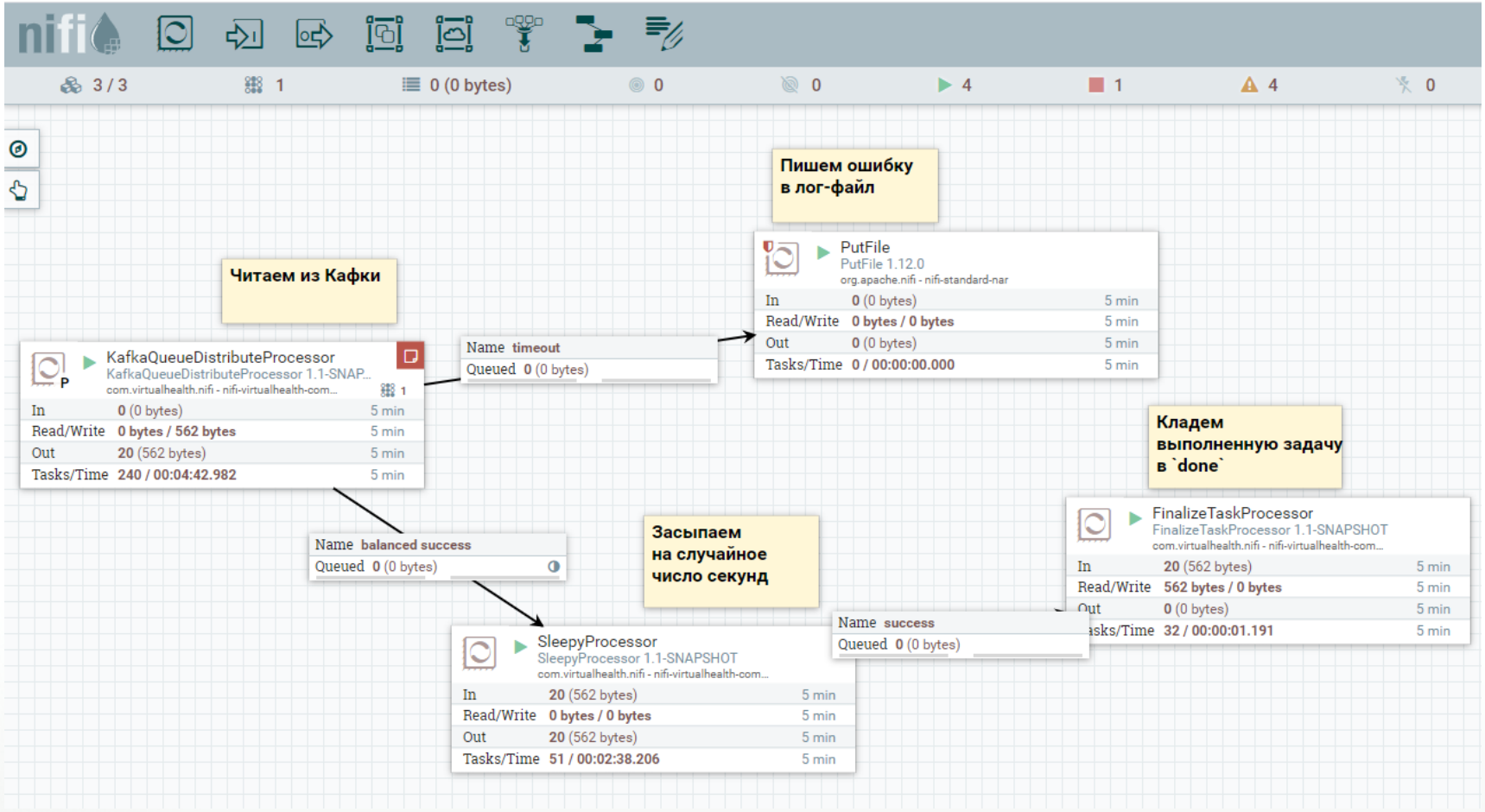
- Полностью читаем топики `work`, `done`
- 1. Проходим всё содержимое `work` по порядку;

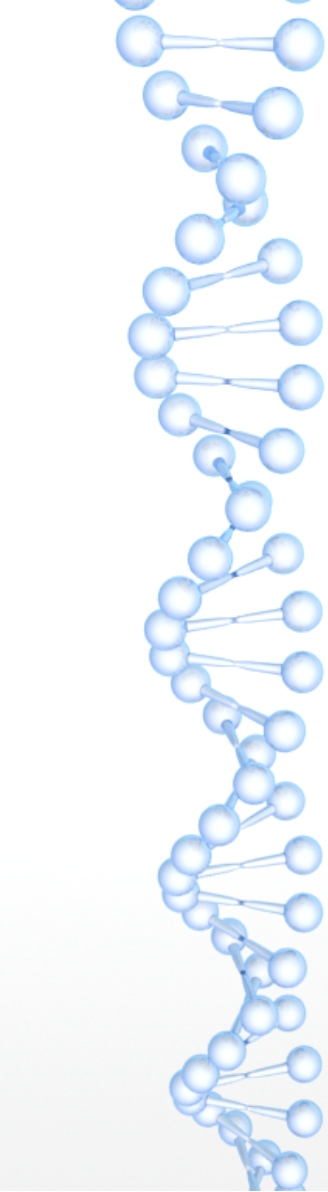
пока текущая задача с оффсетом i является выполненной (найдена в `done`), то *{ work.commit(+1); maxOffsetDone = i; }*

- 2. Проходим всё содержимое `done` по порядку;
- пока текущая задача имеет оффсет < maxOffsetDone,
- выполняем *done.commit(+1)*
- 3. Берем из `backlog` новые задачи в количестве **maxAvailableThreads - | work \ done |**

Дальше демо.


AWS MSK
Kafka
ZooKeeper
AZ-1
AZ-2
AZ-3
4
172.31.17.1
NiFi
P-2
3
P-3
NiFi on EC2
2
172.31.1.1
1
P-1
P-2
P-3
NiFi on EC2
172.31.49.1
P-2
P-3
NiFi on EC2


3 / 3
1
0 (0 bytes)
0
0
4
1
4
0
Пишем ошибку в лог-файл
Читаем из Кафки
PutFile
PutFile 1.12.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
KafkaQueueDistributeProcessor
KafkaQueueDistributeProcessor 1.1-SNAP...
com.virtualhealth.nifi - nifi-virtualhealth-com...
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 562 bytes 5 min
Out 20 (562 bytes) 5 min
Tasks/Time 240 / 00:04:42.982 5 min
Name timeout
Queued 0 (0 bytes)
Кладем выполненную задачу в `done`
FinalizeTaskProcessor
FinalizeTaskProcessor 1.1-SNAPSHOT
com.virtualhealth.nifi - nifi-virtualhealth-com...
In 20 (562 bytes) 5 min
Read/Write 562 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 32 / 00:00:01.191 5 min
Name balanced success
Queued 0 (0 bytes)
Засыпаем на случайное число секунд
SleepyProcessor
SleepyProcessor 1.1-SNAPSHOT
com.virtualhealth.nifi - nifi-virtualhealth-com...
In 20 (562 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 20 (562 bytes) 5 min
Tasks/Time 51 / 00:02:38.206 5 min
Name success
Queued 0 (0 bytes)



Ещё дальше?

- Масштабирование: работа с *partitions*
- Масштабирование: уход от *primary node only*
- ...