РЕАЛИЗАЦИЯ КЭША СООБЩЕНИЙ ВКОНТАКТЕ

СОРОКИН АЛЕКСАНДР, ВКОНТАКТЕ

http://vk.com/alexoro

uas.sorokin@gmail.com

О ЧЕМ ПОЙДЕТ РЕЧЬ

01

Как хранить любые загруженные сообщения не последовательно и максимально долго

02

Как хранить в базе данных такую сложную сущность, как сообщение

03

Как ускорить SQLite-запросы

О ЧЕМ ПОЙДЕТ РЕЧЬ

01

Как хранить любые загруженные сообщения не последовательно и максимально долго

02

Как хранить в базе данных такую сложную сущность, как сообщение

03

Как ускорить SQLite-запросы

ОТКУДА В КЭШЕ ПОЯВЛЯЮТСЯ СООБЩЕНИЯ

- Взаимодействие с UI (открытие экрана, подгрузка)
- События из long-poll соединения (активно во время работы приложения)
- История событий за период отсутствия (загружается во время запуска приложения)

ОТКУДА В КЭШЕ ПОЯВЛЯЮТСЯ СООБЩЕНИЯ

- Взаимодействие с UI (открытие экрана, подгрузка)
- События из long-poll соединения (активно во время работы приложения)
- История событий за период отсутствия (загружается во время запуска приложения)

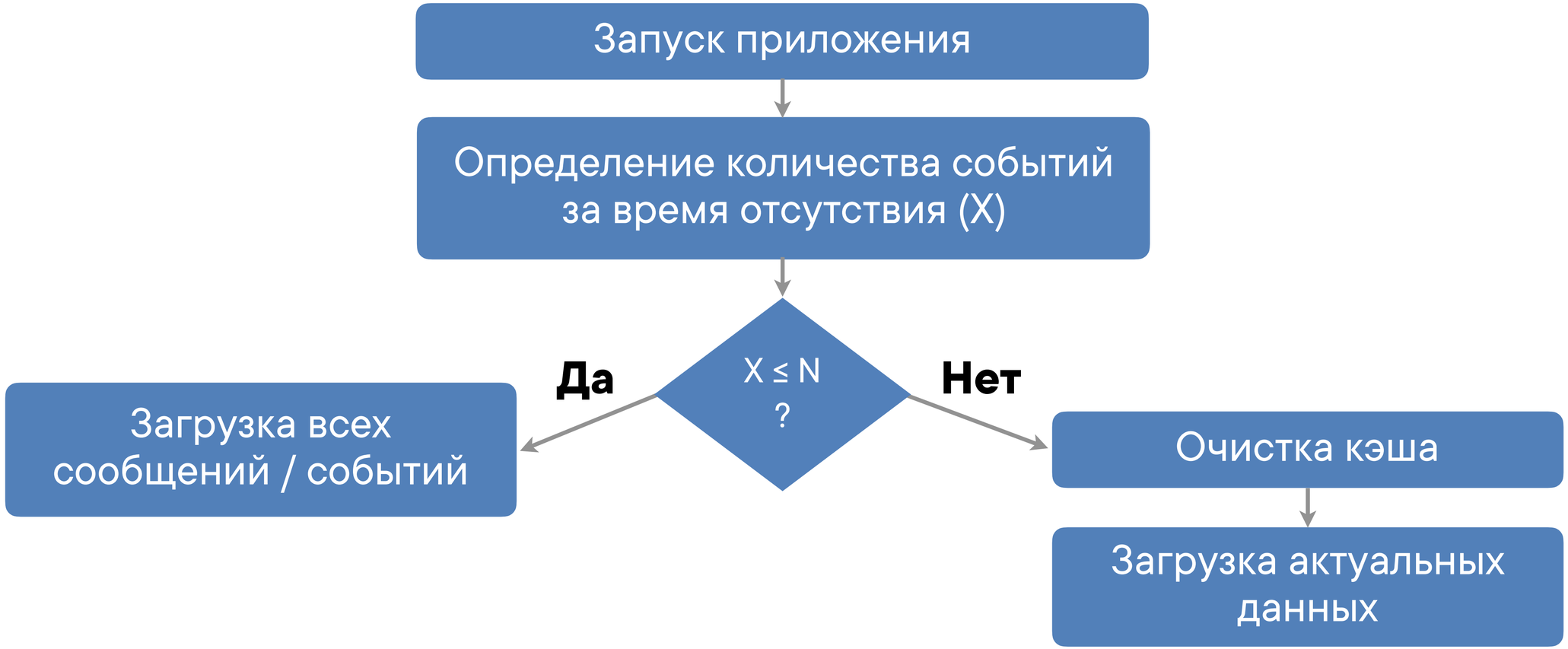
ОТКУДА В КЭШЕ ПОЯВЛЯЮТСЯ СООБЩЕНИЯ

- Взаимодействие с UI (открытие экрана, подгрузка)
- События из long-poll соединения (активно во время работы приложения)
- История событий за период отсутствия (загружается во время запуска приложения)

ОТКУДА В КЭШЕ ПОЯВЛЯЮТСЯ СООБЩЕНИЯ

- Взаимодействие с UI (открытие экрана, подгрузка)
- События из long-poll соединения (активно во время работы приложения)
- История событий за период отсутствия (загружается во время запуска приложения)

АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ДО)



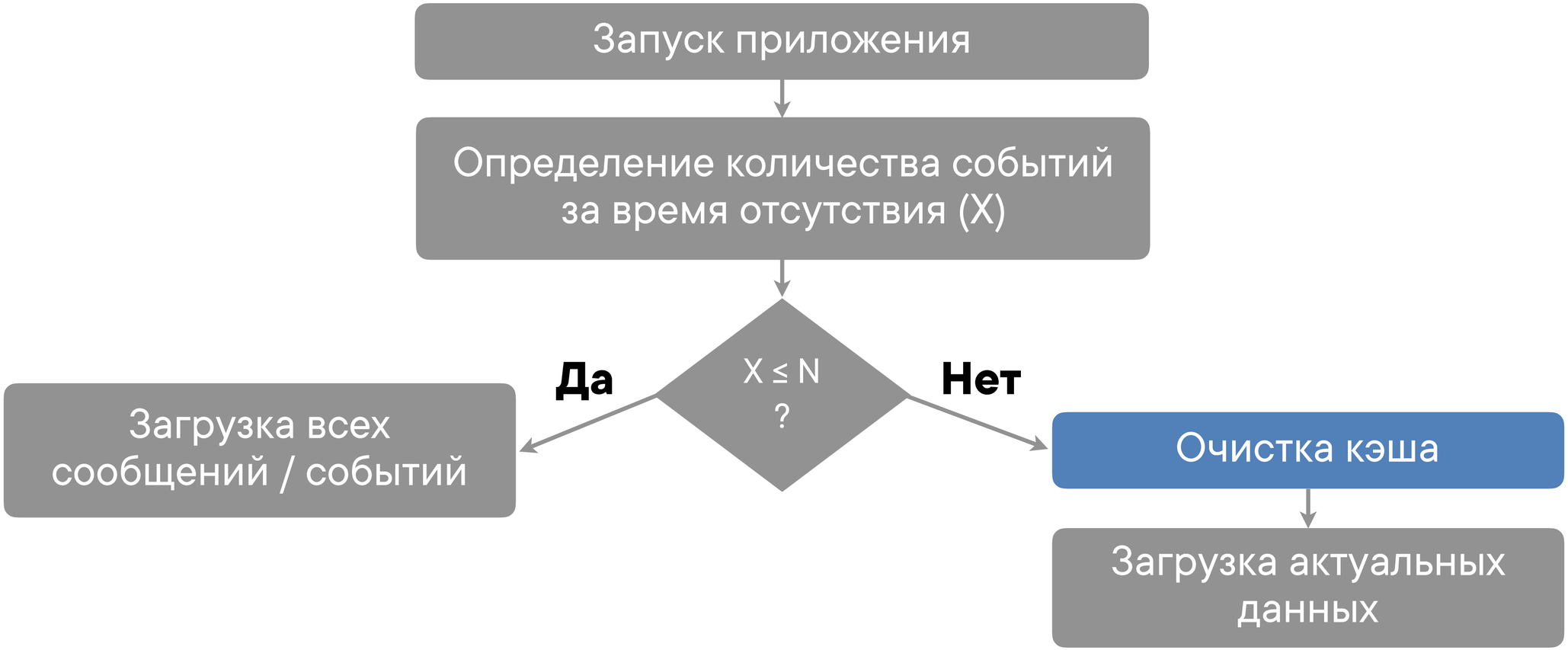
АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ДО)

ПРОБЛЕМА:

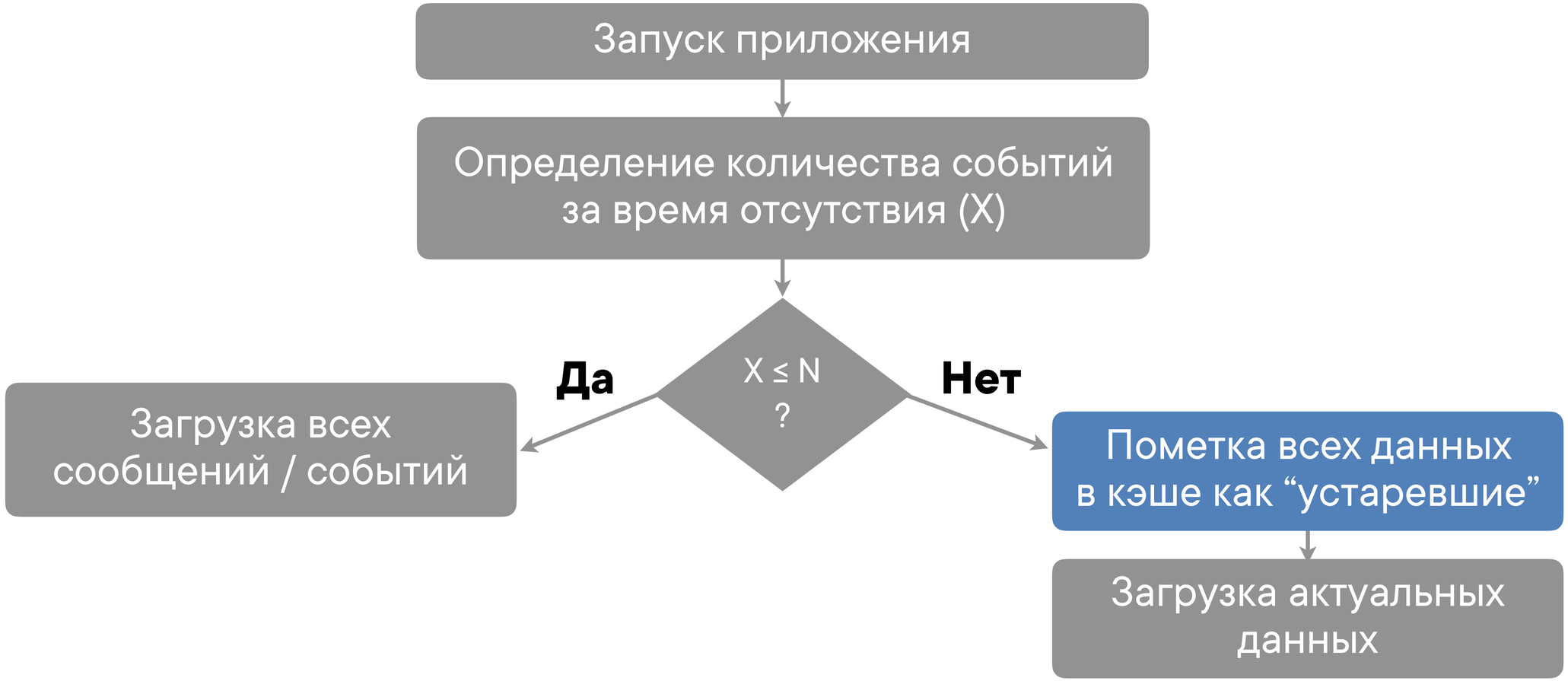
С вероятностью ~99.9% сообщения в кэше
не изменились (редактирование / удаление)

Но мы их все равно удаляем при очистке кэша

АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ДО)



АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ПОСЛЕ)



АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ПОСЛЕ)

ПРЕИМУЩЕСТВА:

Хранение всех сообщений в кэше, которые когда-либо были загружены

Улучшение локального поиска

Возможность сразу показать старые сообщения и обновлять их в фоне

АЛГОРИТМ СИНХРОНИЗАЦИИ ИСТОРИИ СОБЫТИЙ ЗА ПЕРИОД ОТСУТСТВИЯ (ПОСЛЕ)

ПРОБЛЕМЫ:

Так как не происходит загрузка всех событий за период отсутствия, то история в кэше становится “разрывистой”

Непонятно, как хранить разрывистую историю

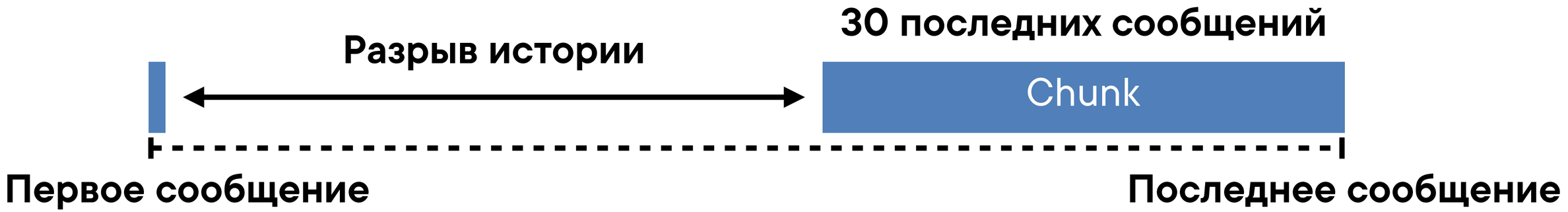
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

Полная история может быть представлена как большой отрезок

Сообщение - это точка на отрезке

Группа сообщений, находящихся рядом, - набор точек или chunk



ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

01

Взаимодействие с UI

открытие экрана, подгрузка

02

События из long-poll соединения

активно во время работы приложения

03

История событий за период отсутствия

загружается во время запуска приложения

ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

01

Взаимодействие с UI

открытие экрана, подгрузка

02

События из long-poll соединения

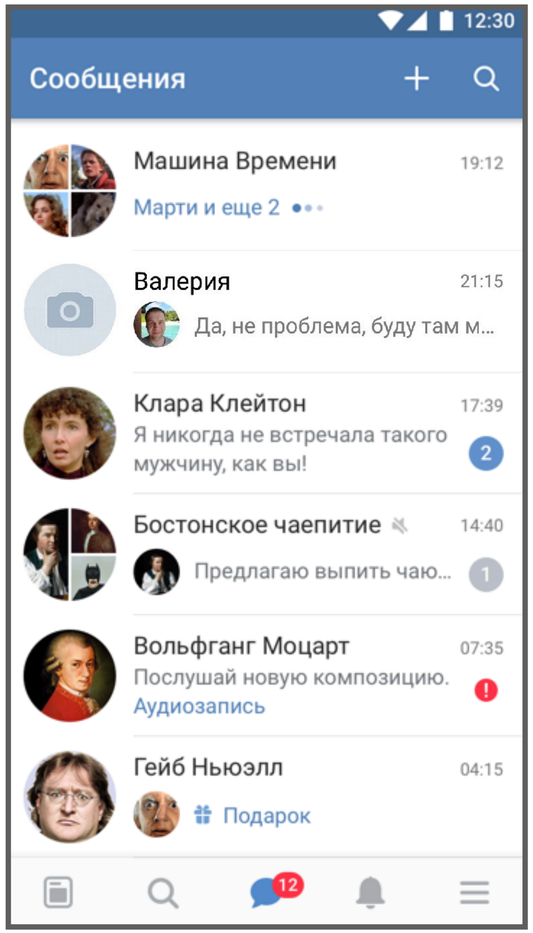
активно во время работы приложения

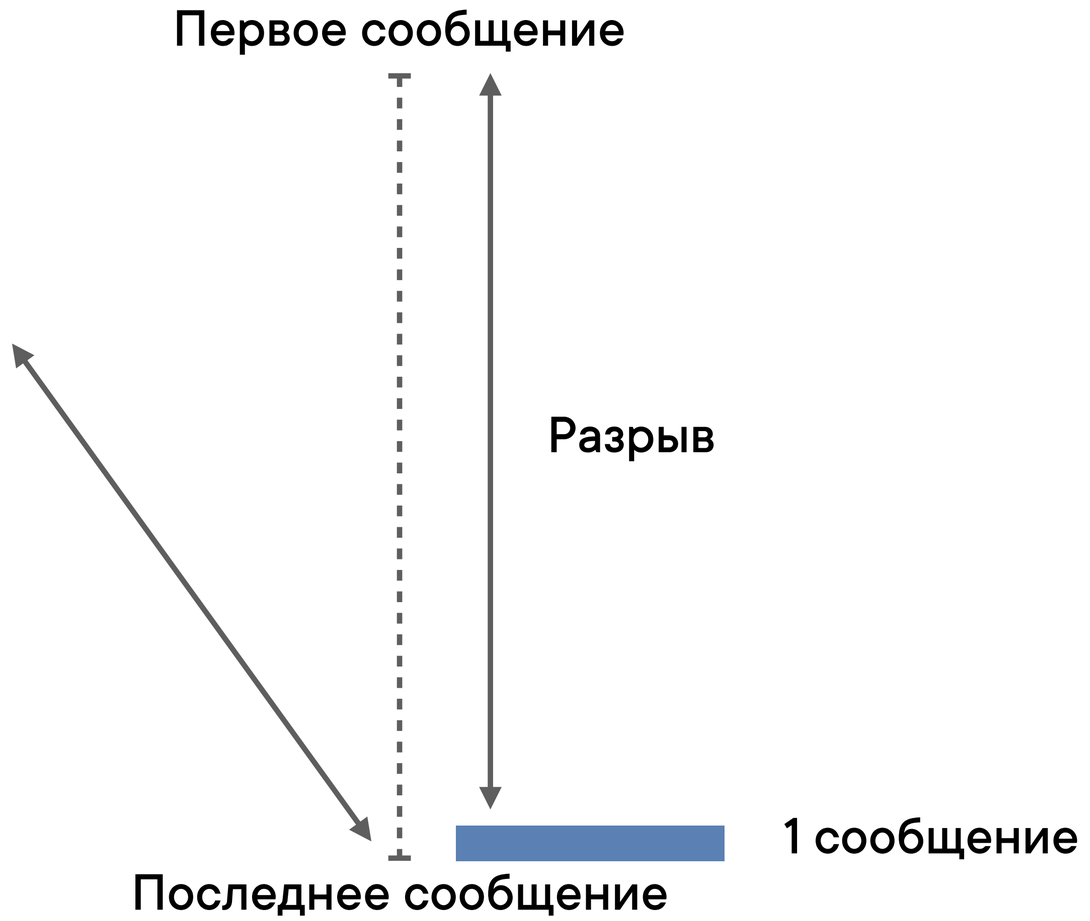
03

История событий за период отсутствия

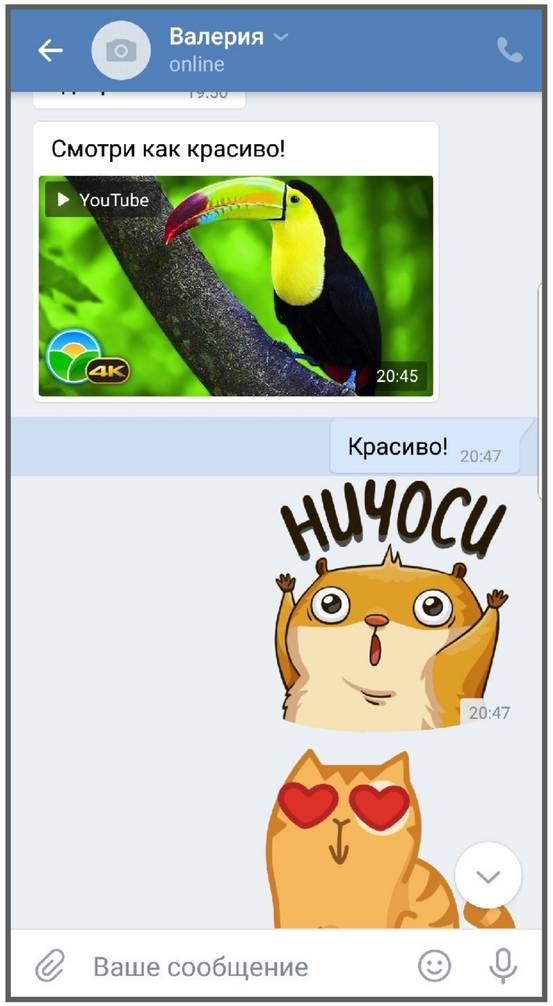
загружается во время запуска приложения

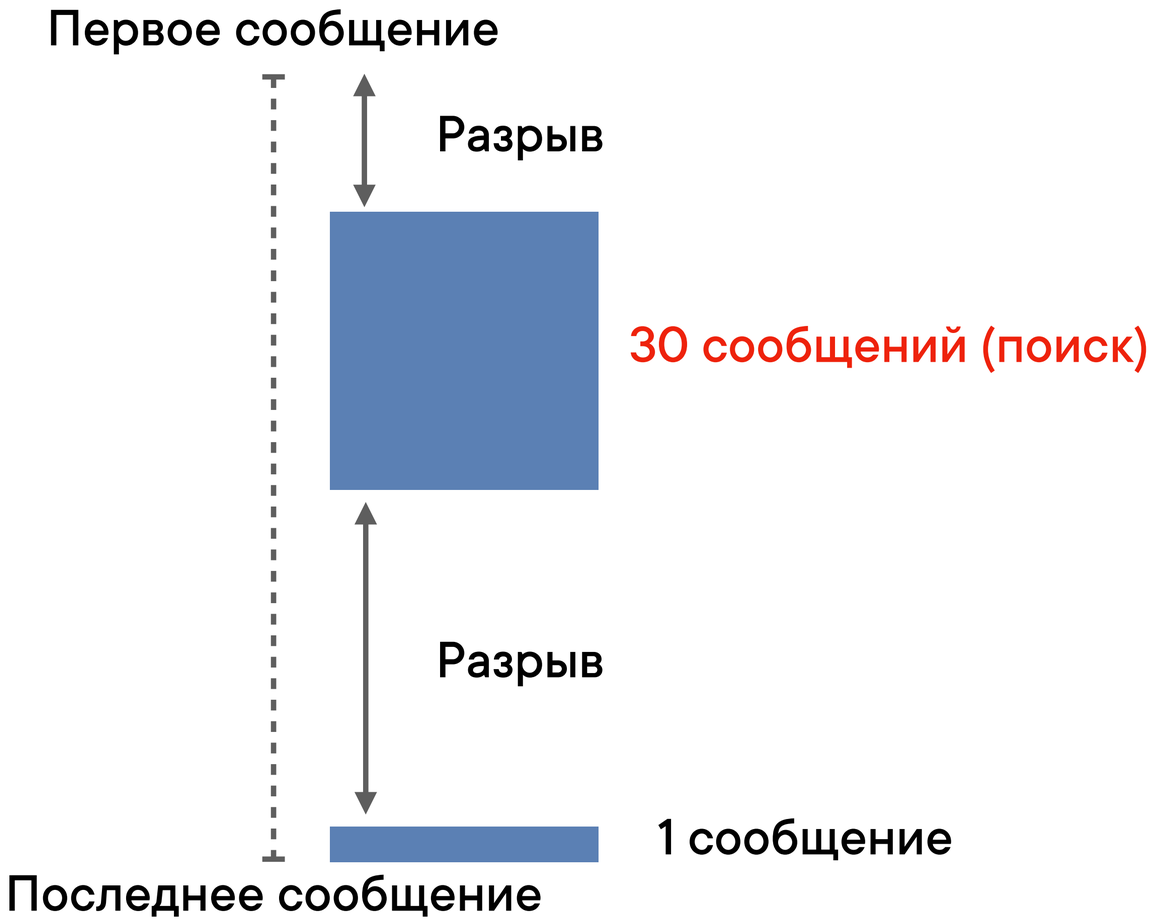
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



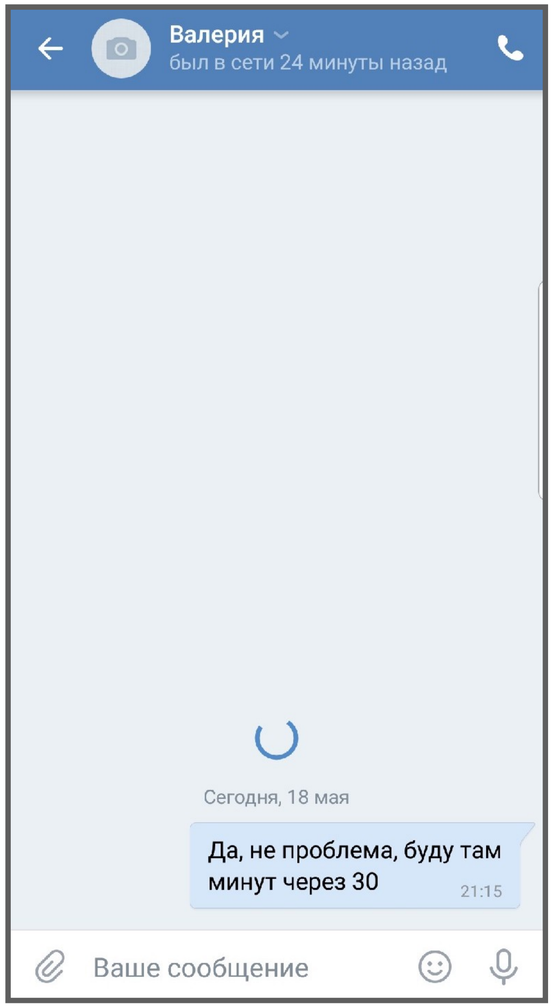


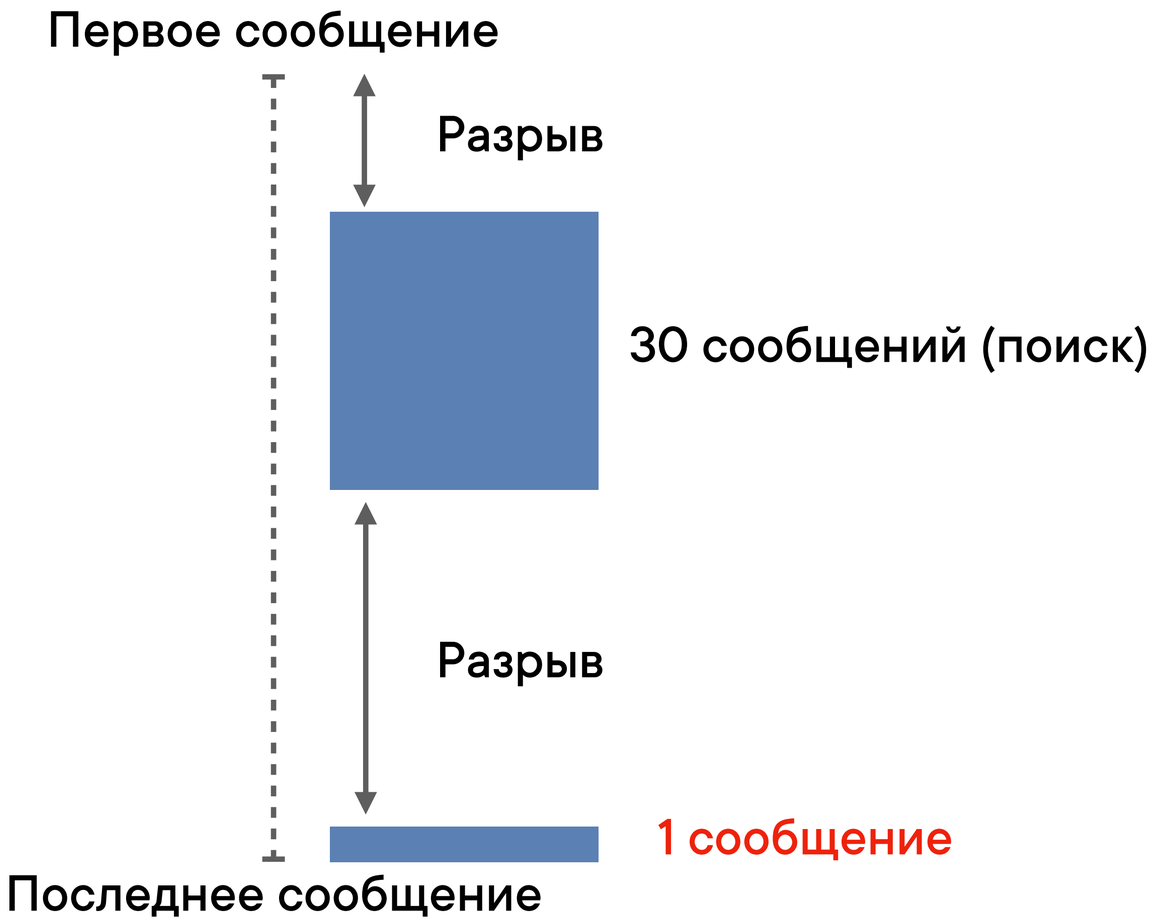
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



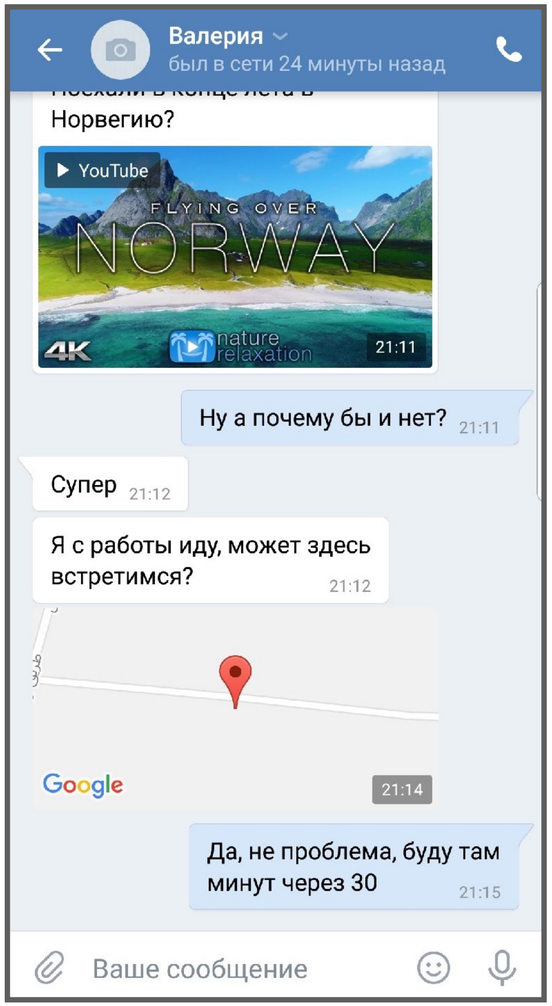


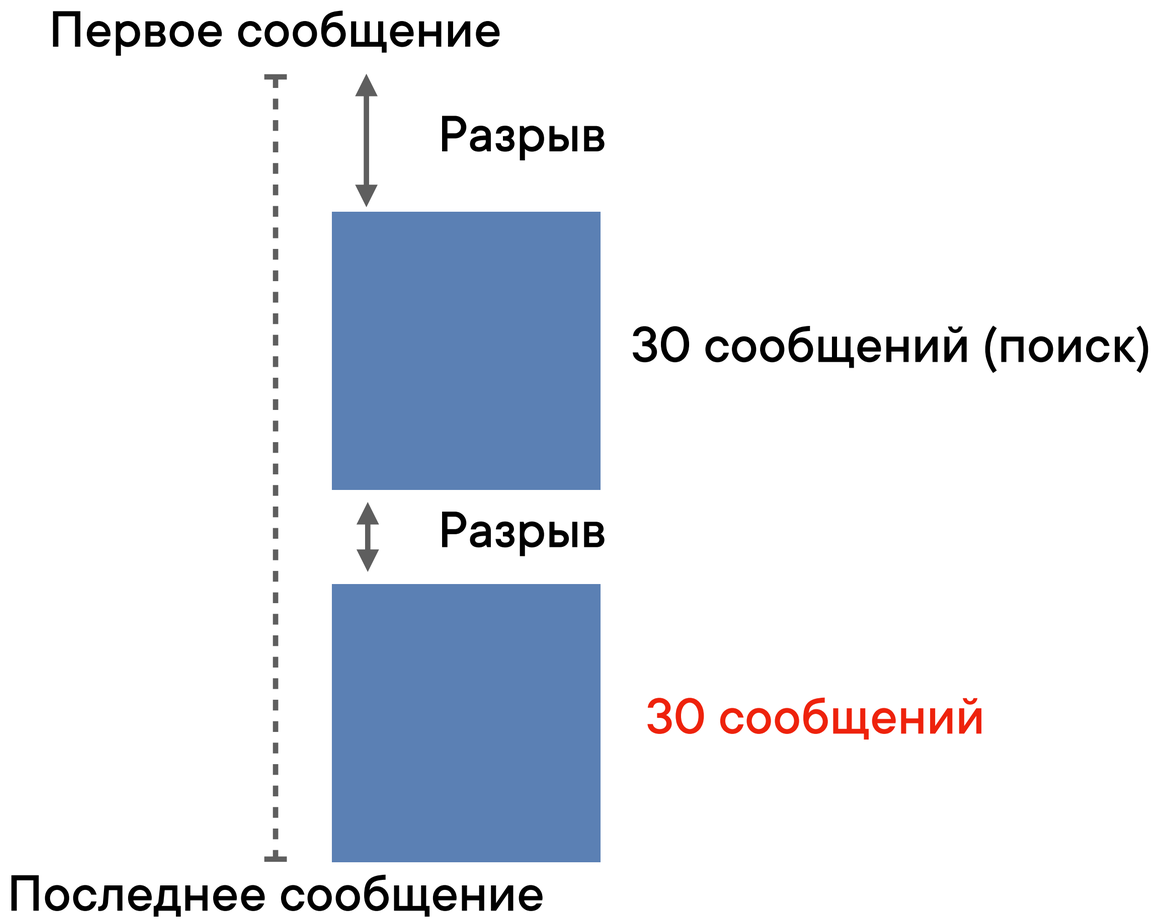
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



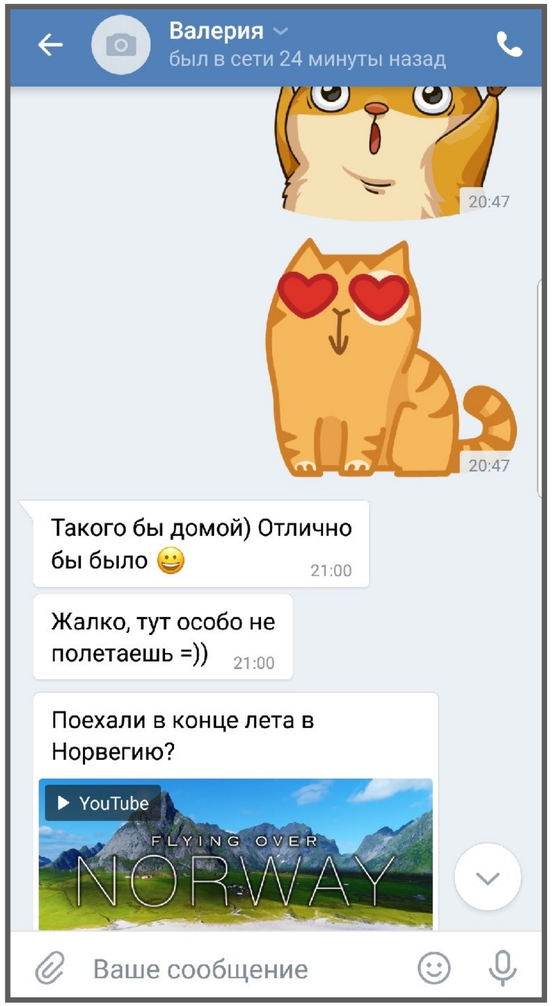


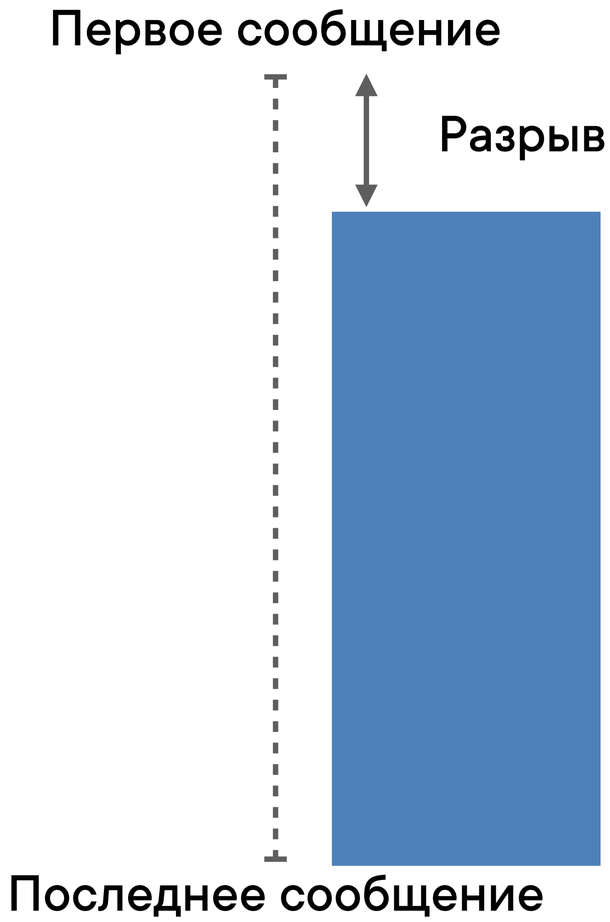
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ





ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ





Сообщения из Поиска + Последние

ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

01

Взаимодействие с UI

открытие экрана, подгрузка

02

События из long-poll соединения

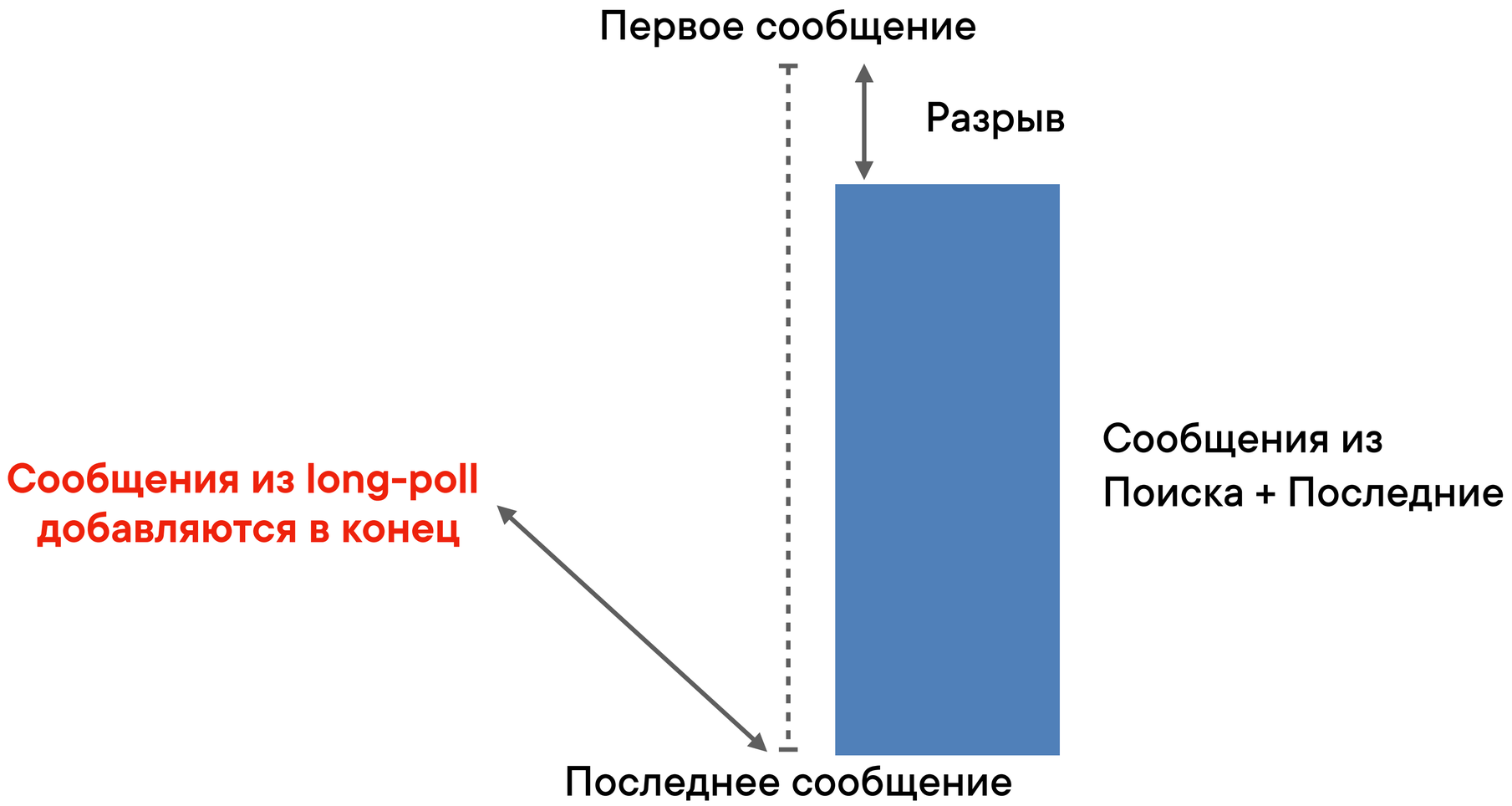
активно во время работы приложения

03

История событий за период отсутствия

загружается во время запуска приложения

ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ

01

Взаимодействие с UI

открытие экрана, подгрузка

02

События из long-poll соединения

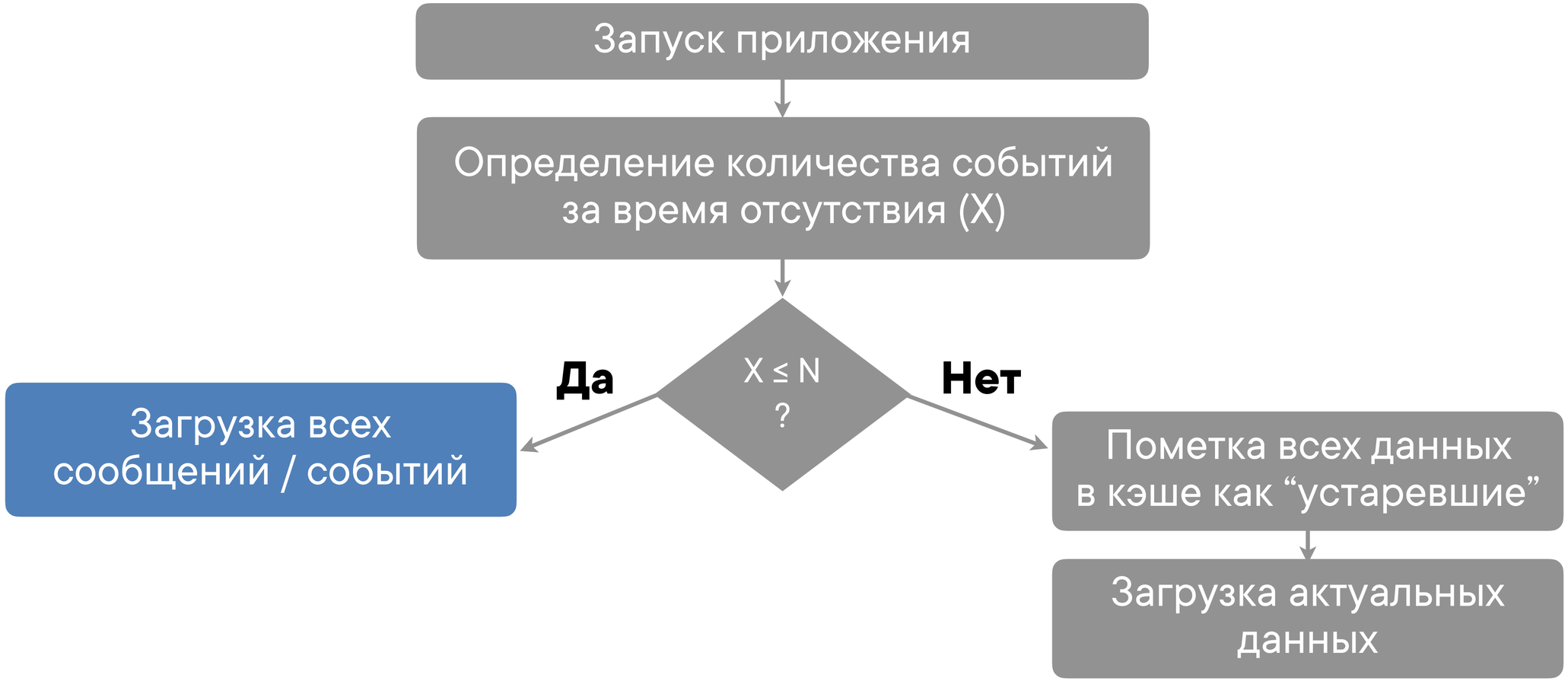
активно во время работы приложения

03

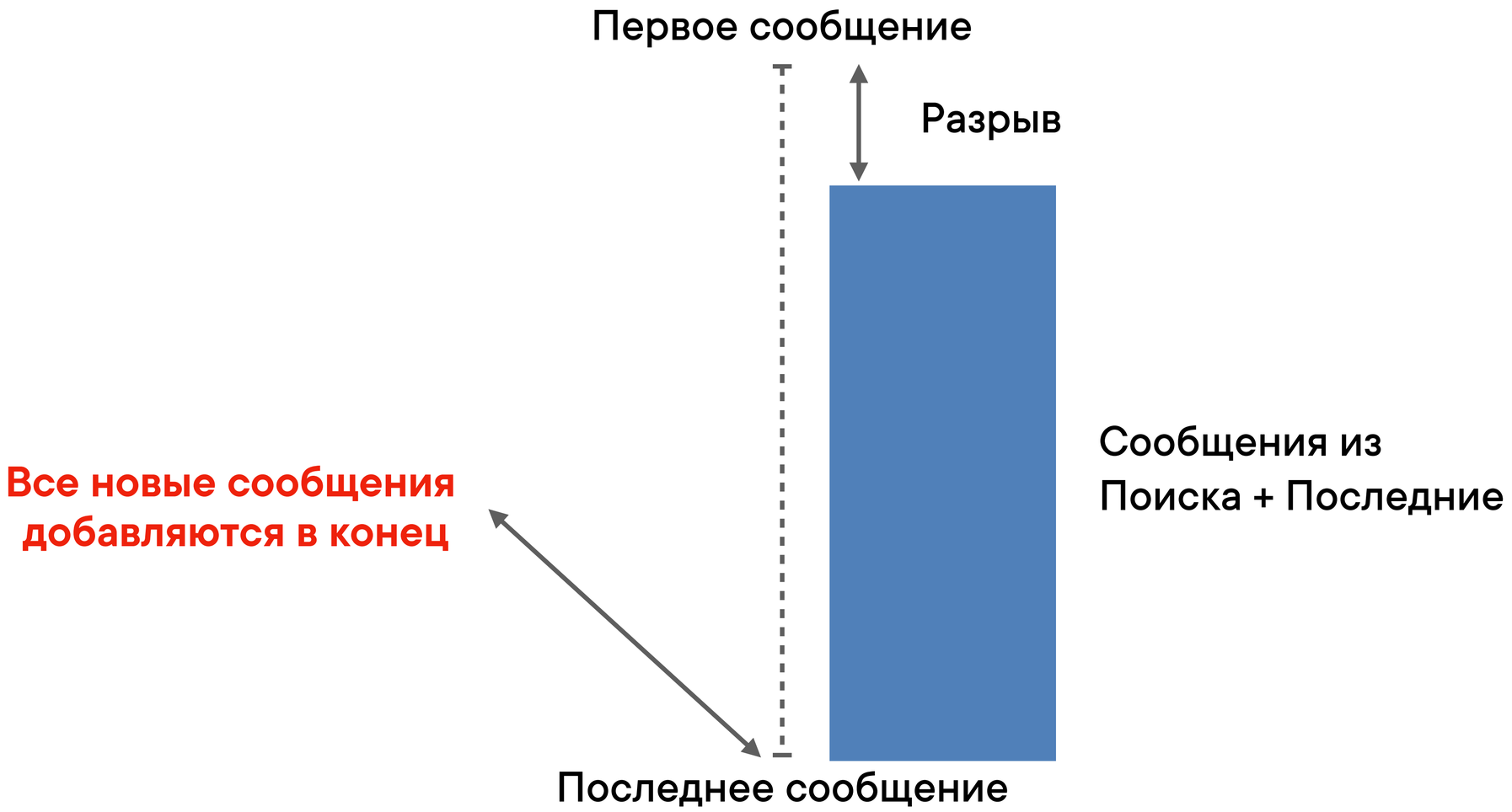
История событий за период отсутствия

загружается во время запуска приложения

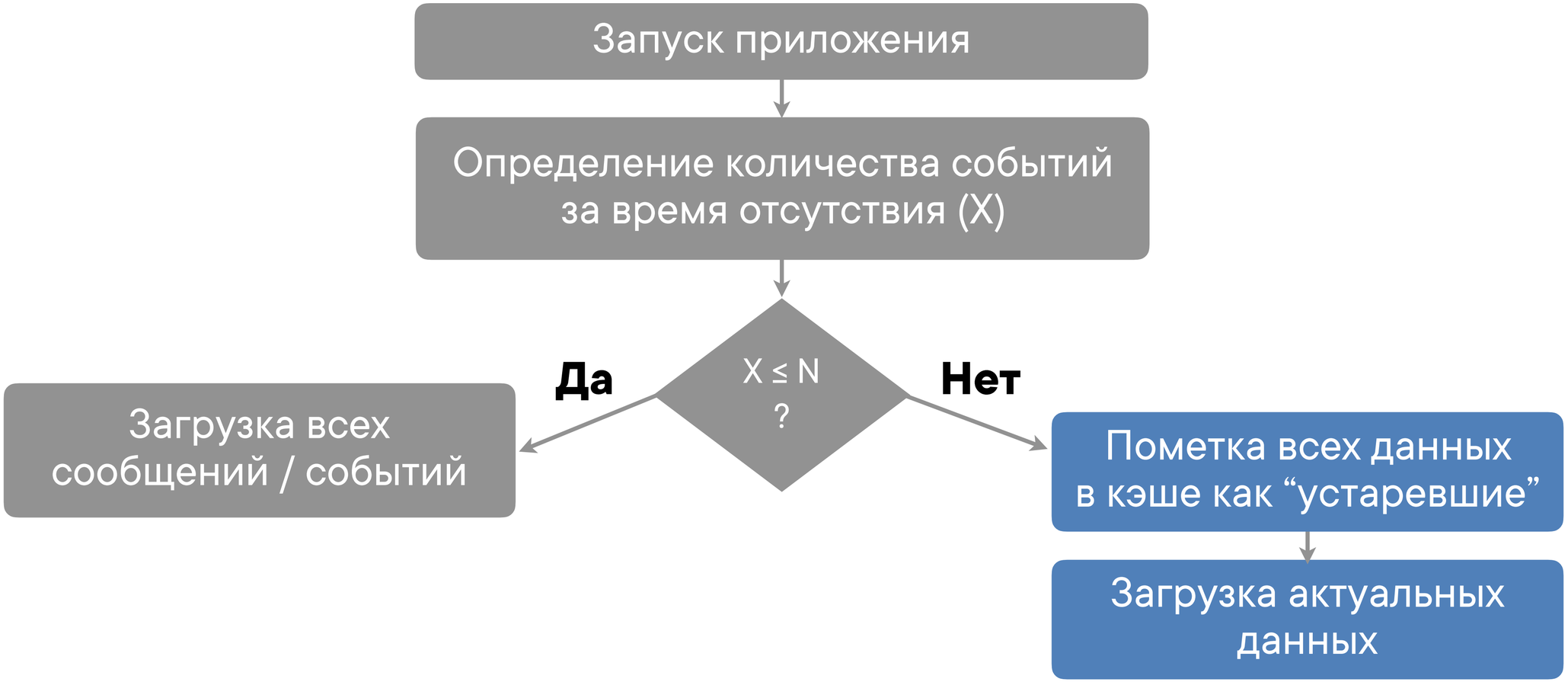
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



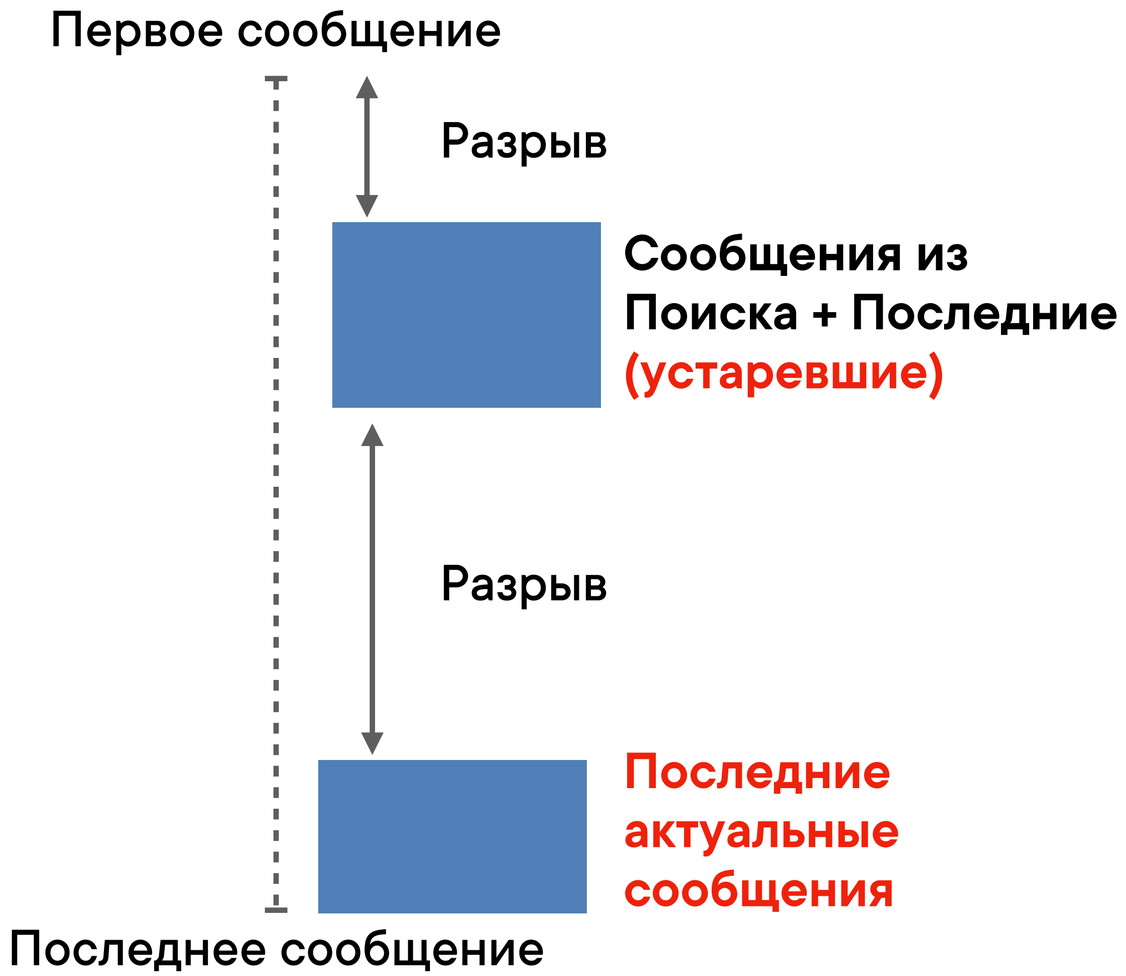
ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



ХРАНЕНИЕ РАЗРЫВА ИСТОРИИ

ХРАНЕНИЕ РАЗРЫВА ИСТОРИИ

КАЖДОЕ СООБЩЕНИЕ СОДЕРЖИТ ПОЛЯ:

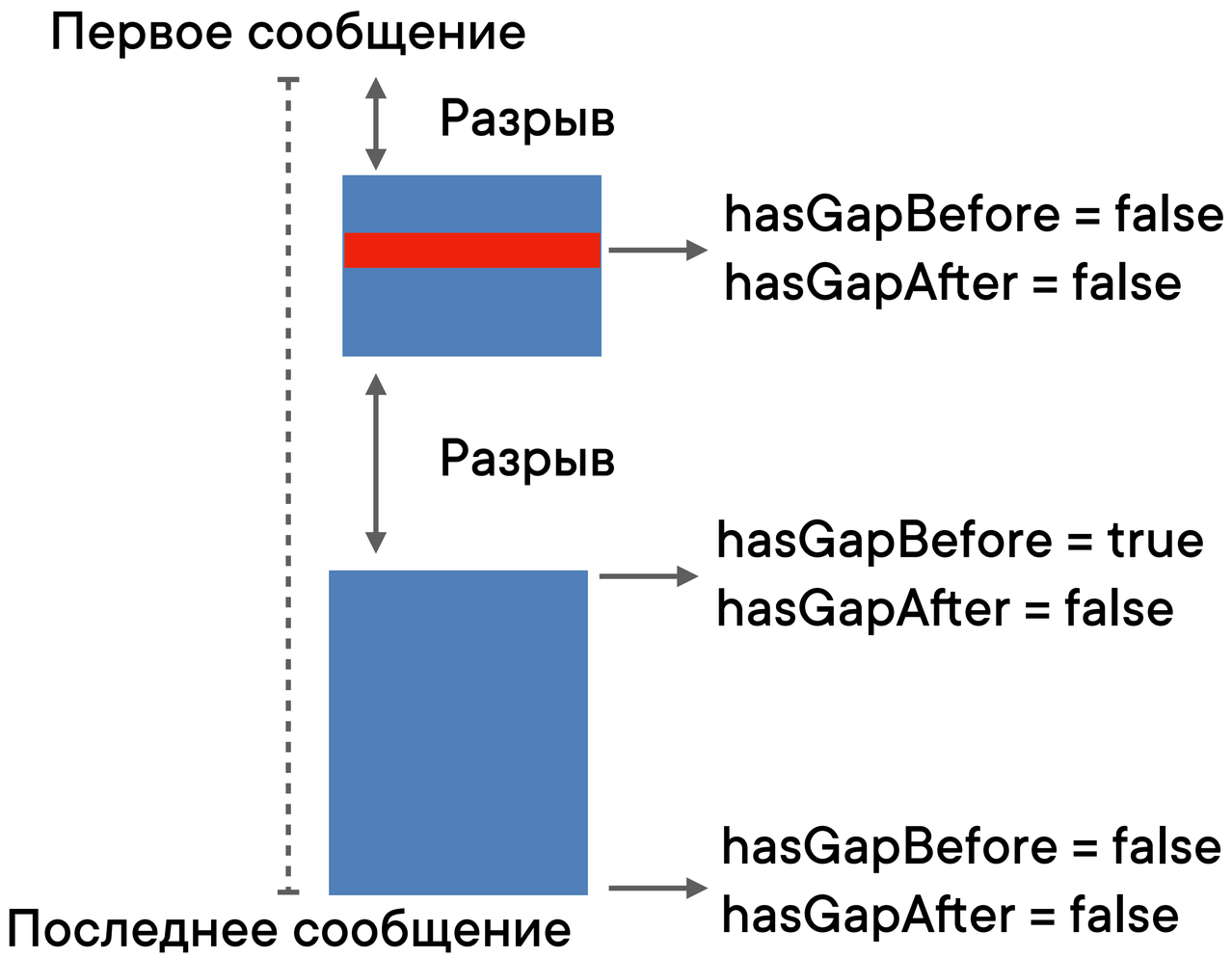
hasGapBefore

есть ли перед сообщением разрыв

hasGapAfter

есть ли после сообщения разрыв

ХРАНЕНИЕ РАЗРЫВА ИСТОРИИ



ХРАНЕНИЕ РАЗРЫВА ИСТОРИИ

- Значения высчитываются при каждой записи сообщений в базу
- Перед отдачей данных на UI отсекаются сообщения, где hasGapBefore/ hasGapAfter = true, чтобы корректно показывать индикатор подгрузки

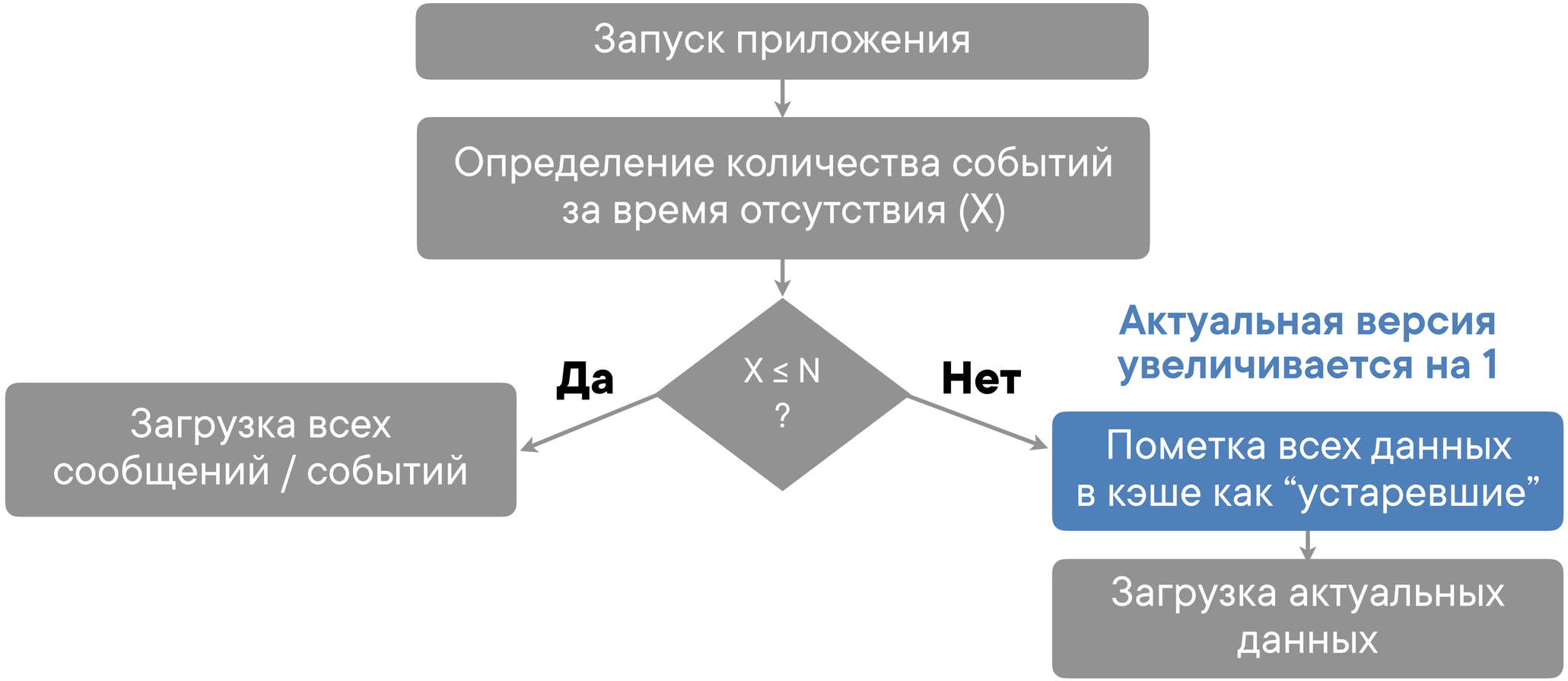
А КАК ПОНЯТЬ, ЧТО СООБЩЕНИЕ “УСТАРЕЛО”?

А ЧТО ЕСЛИ СТАРЫЕ СООБЩЕНИЯ В КЭШЕ ИЗМЕНИЛИСЬ?

- Вводим понятие version: Int
- Каждое сообщение содержит поле version
- Движок сообщений хранит актуальную version

Если версии не совпадают - сообщение устаревшее

ПРИМЕР НАПОЛНЕНИЯ ИСТОРИИ СООБЩЕНИЙ



РЕАЛИЗАЦИЯ ХРАНЕНИЯ ИСТОРИИ СООБЩЕНИЙ. ВЫВОДЫ

РЕАЛИЗАЦИЯ ХРАНЕНИЯ ИСТОРИИ СООБЩЕНИЙ. ВЫВОДЫ

ПРИНЦИП ЗАГРУЗКИ И ХРАНЕНИЯ:

- При старте приложения можно загружать только часть событий/сообщений за период отсутствия.
- История переписки в кэше может содержать разрывы.
- Если неизвестно, что произошло с сообщением, то можно пометить его как “устаревшее”, чтобы позднее обновить.

РЕАЛИЗАЦИЯ ХРАНЕНИЯ ИСТОРИИ СООБЩЕНИЙ. ВЫВОДЫ

ПРЕИМУЩЕСТВА:

Более быстрая синхронизация при старте приложения

Разрывы позволяют хранить историю фрагментами

Кэш становится долгосрочным

О ЧЕМ ПОЙДЕТ РЕЧЬ

01

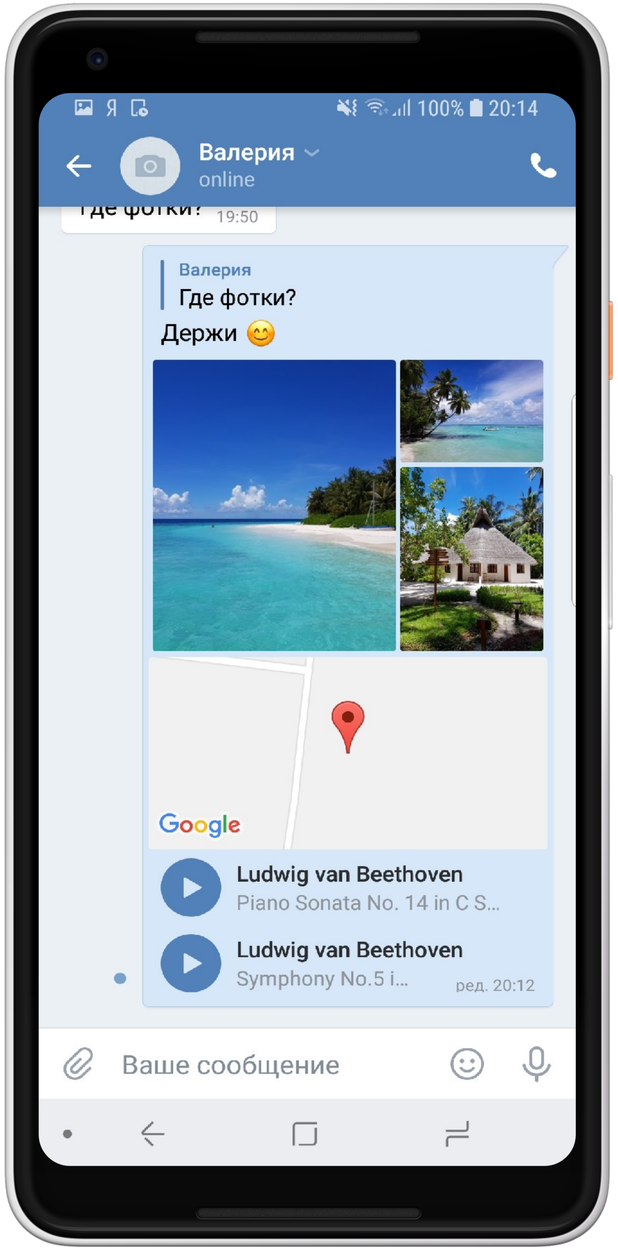
Как хранить любые загруженные сообщения не последовательно и максимально долго

02

Как хранить в базе данных такую сложную сущность, как сообщение

03

Как ускорить SQLite-запросы



СТРУКТУРА СООБЩЕНИЯ

- текст
- список на 10 аттачей
  - количество типов аттачей ~25
  - аттач может содержать сложные структуры (опросы, плейлисты и т.д.)
- ответ на какое-либо сообщение
- дерево пересланных сообщений
  - до 100 пересланных на одном уровне
  - до 40 пересланных в глубину

СПОСОБЫ РЕШЕНИЯ

01

Правильное с точки зрения теории баз данных

02

Рабочее с точки зрения производительности

СПОСОБЫ РЕШЕНИЯ

01

Правильное с точки зрения теории баз данных

02

Рабочее с точки зрения производительности

РЕШЕНИЕ С ТОЧКИ ЗРЕНИЯ ТЕОРИИ БАЗ ДАННЫХ

msg
id
text
time

РЕШЕНИЕ С ТОЧКИ ЗРЕНИЯ ТЕОРИИ БАЗ ДАННЫХ

msg
id
text
time

msg_reply
msg_id
reply_id

reply
id
text
time

msg_fwds
msg_id
fwd_id
left*
right*

fwd
id
text
time

* Алгоритм Nested Set для хранения деревьев в таблице

РЕШЕНИЕ С ТОЧКИ ЗРЕНИЯ ТЕОРИИ БАЗ ДАННЫХ

msg
id
text
time

msg_reply
msg_id
reply_id

reply
id
text
time

msg_attach_photo
msg_id
reply_id
fwd_id
attach_id

attach_photo
id
image
description

msg_fwds
msg_id
fwd_id
left*
right*

fwd
id
text
time

msg_attach_poll
msg_id
reply_id
fwd_id
attach_id

attach_poll
id
question
expire_date

attach_poll_answers
poll_id
answer

Ещё примерно 60 таблиц для аттачей

* Алгоритм Nested Set для хранения деревьев в таблице

А ДАВАЙТЕ ЗАГРУЗИМ ПОСЛЕДНИЕ 50 СООБЩЕНИЙ В ДИАЛОГЕ?

СПОСОБЫ РЕШЕНИЯ

01

Правильное с точки зрения теории баз данных

02

Рабочее с точки зрения производительности

РАБОЧЕЕ РЕШЕНИЕ С ТОЧКИ ЗРЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

messages
id
text
time
reply
fwd
attaches

- Создается утилита для конвертации ответов / пересланных / аттачей в byte[] и обратно (аналогично Parcelable)
- Таблица сообщений содержит nullable blob поля

А КАК ЖЕ МИГРАЦИЯ ДАННЫХ?

МИГРАЦИЯ ДАННЫХ СООБЩЕНИЯ

messages
id
text
time
reply
fwd
attaches
version

msg_attaches
msg_id
attach_id
attach_type

- Находим сообщения, которые необходимо обновить
- Сбрасываем нужные поля в null
- Меняем version на 0
- Сообщение автоматически становится устаревшим и будет позднее обновлено

ХРАНЕНИЕ ДАННЫХ СООБЩЕНИЯ. ВЫВОДЫ

ХРАНЕНИЕ ДАННЫХ СООБЩЕНИЯ. ВЫВОДЫ

- Сложная структура хранится в сериализованном виде в одной SQLite-таблице
- Увеличивается скорость вставки / выборки (за счет меньшего количества запросов)
- Легкая схема поддержки и развития

О ЧЕМ ПОЙДЕТ РЕЧЬ

01

Как хранить любые загруженные сообщения не последовательно и максимально долго

02

Как хранить в базе данных такую сложную сущность, как сообщение

03

Как ускорить SQLite-запросы

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы

ИНДЕКСЫ

КОГДА ИСПОЛЬЗОВАТЬ:

Выборка малого количества значений в объемных таблицах

Индексируемое значение более-менее уникально

ИНДЕКСЫ

messages
rowid
id
text
time

rowid:

- Автоматически создается SQLite
- Auto-increment значение
- Все записи в таблице отсортированы по rowid
- Алгоритмическая сложность поиска по rowid - O(log n)

ИНДЕКСЫ

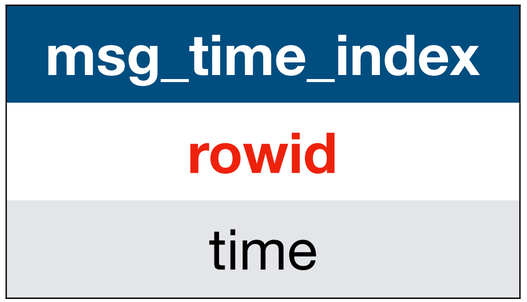
SELECT * FROM messages WHERE time = ?

messages
id
text
time

- Таблица ничего не знает про значения time
- Для поиска нужной записи потребуется полный перебор таблицы
- Алгоритмическая сложность поиска по time - O(n)

ИНДЕКСЫ

CREATE INDEX msg_time_index ON messages(time)



- Создается структура, содержащая в себе time и rowid в таблице messages
- Структура отсортирована по time
- Алгоритмическая сложность поиска O(log n)

ИНДЕКСЫ

SELECT * FROM messages WHERE time = 400

Содержимое индекса

time	rowid
100	1
150	3
250	4
400	2
500	5

O (log n)

Содержимое таблицы messages

rowid	id	text	time
1	12	aaa	100
2	31	ddd	400
3	25	bbb	150
4	30	ccc	250
5	42	eee	500

O (log n)

ИНДЕКСЫ

messages
id
text
time

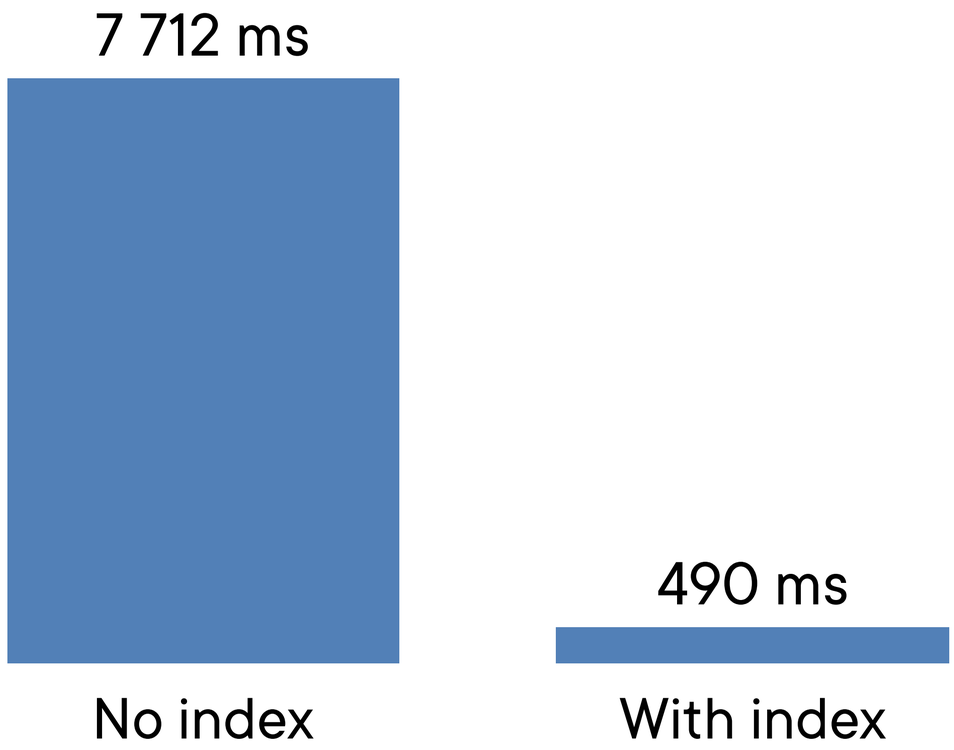
Тест производительности

- Таблица messages на 10.000 записей
- Оценка длительности 5.000 запросов относительно time с индексом и без

https://vk.cc/9p7JFW

ИНДЕКСЫ

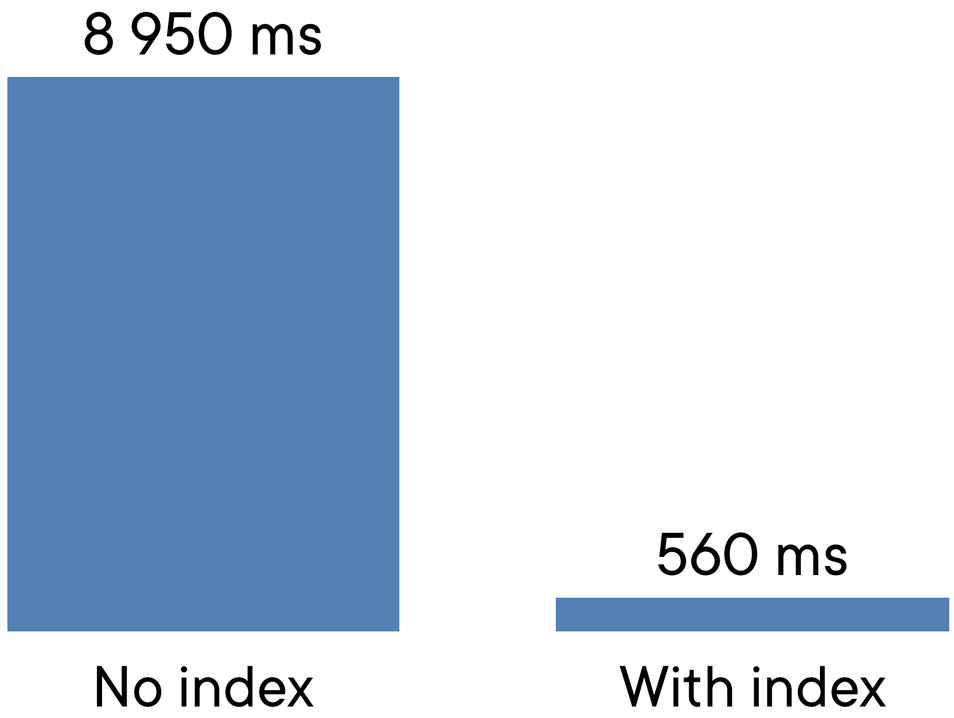
SELECT id FROM messages WHERE time = ?



~16X
быстрее

ИНДЕКСЫ

SELECT id FROM messages WHERE time < ?
ORDER BY time DESC LIMIT 50



~16X
быстрее

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах

SELECT C PLACEHOLDER

```
val sql = “SELECT * FROM messages WHERE time = ?”
val args = arrayOf(“1”)
db.rawQuery(sql, args)
```

SELECT С PLACEHOLDER

КОГДА ИСПОЛЬЗОВАТЬ:

Наличие частых SELECT-запросов

SELECT С PLACEHOLDER

ПРИНЦИП РАБОТЫ:

- Все SQLite-запросы сводятся к SQLiteConnection.java#acquirePreparedStatement(sql)
- Внутри используется LruCache SQL-запросов, где:
  - Ключ - sql-запрос
  - Значение - подготовленный / скомпилированный sql-запрос
- Если используется placeholder, то можно взять ранее скомпилированный запрос

SELECT C PLACEHOLDER

messages
id
text
time

Тест производительности

- Таблица messages на 10.000 записей
- Оценка длительности 5.000 запросов относительно time, когда time передается либо напрямую в sql-запрос, либо через placeholder

https://vk.cc/9p7JFW

SELECT C PLACEHOLDER

Запросы без placeholder

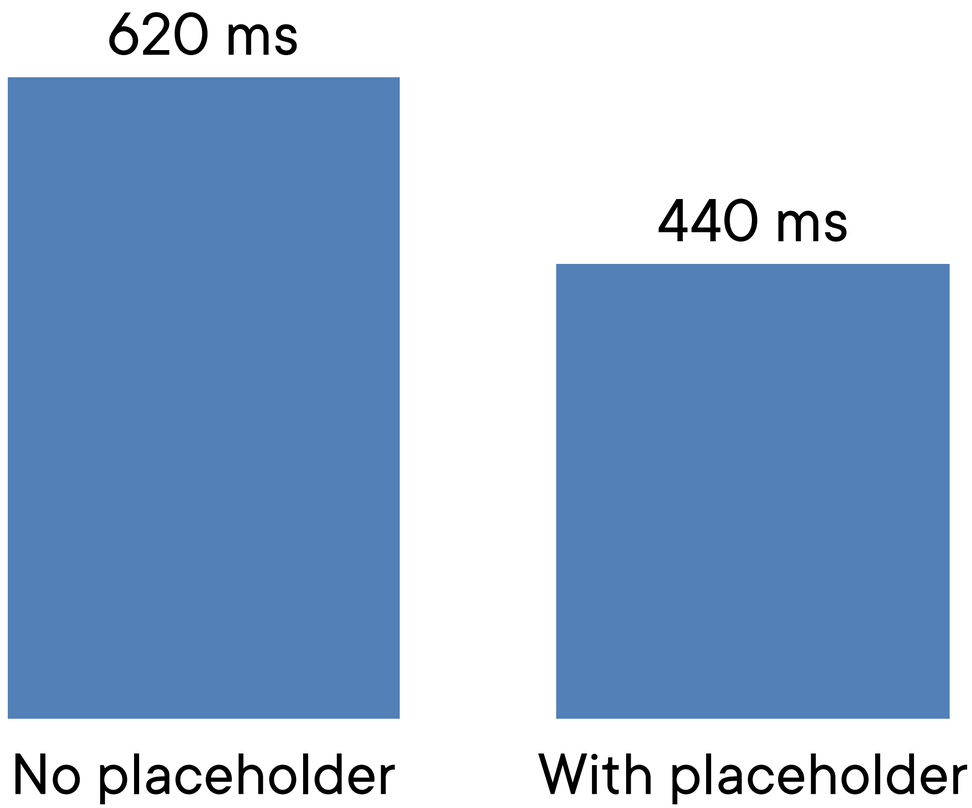
```
val queries = mutableListOf<String>()
val args = mutableListOf<Array<String>?>()
for (i in 0 until 5000) {
    queries.add("SELECT * FROM messages WHERE time = $i")
    args.add(null)
}
for (i in 0 until 5000) {
    db.rawQuery(queries[i], args[i])
}
```

SELECT С PLACEHOLDER

Запросы с placeholder

```
val queries = mutableListOf<String>()
val args = mutableListOf<Array<String>?>()
for (i in 0 until 5000) {
    queries.add("SELECT * FROM messages WHERE time = ?")
    args.add(arrayOf("$i"))
}
for (i in 0 until 5000) {
    db.rawQuery(queries[i], args[i])
}
```

SELECT C PLACEHOLDER



~1,4X
быстрее

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов

НАСТРОЙКА РАЗМЕРА КЭША ДЛЯ СКОМПИЛИРОВАННЫХ SQL-ЗАПРОСОВ

```
class OpenHelper(context: Context, name: String)
    : SQLiteOpenHelper(context, name, null, 1) {

    override fun onConfigure(db: SQLiteDatabase) {
        // Default = 25
        // MAX_SQL_CACHE_SIZE = 100
        db.setMaxSqlCacheSize(SQLiteDatabase.MAX_SQL_CACHE_SIZE)
    }

}
```

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов
- Включить WAL (Write Ahead Log)

WAL

КОГДА ИСПОЛЬЗОВАТЬ:

Многопоточный доступ к БД
Частые чтение/запись

JOURNAL VS WAL

В SQLite есть два режима обеспечения атомарных операций

01

Режим журнала

Journal

02

Режим Write Ahead Log

WAL

JOURNAL

Принцип работы при изменении данных:

Создается отдельный файл - journal → Backup оригинальных данных в journal → Запись новых данных в базу данных → Удаление journal

JOURNAL

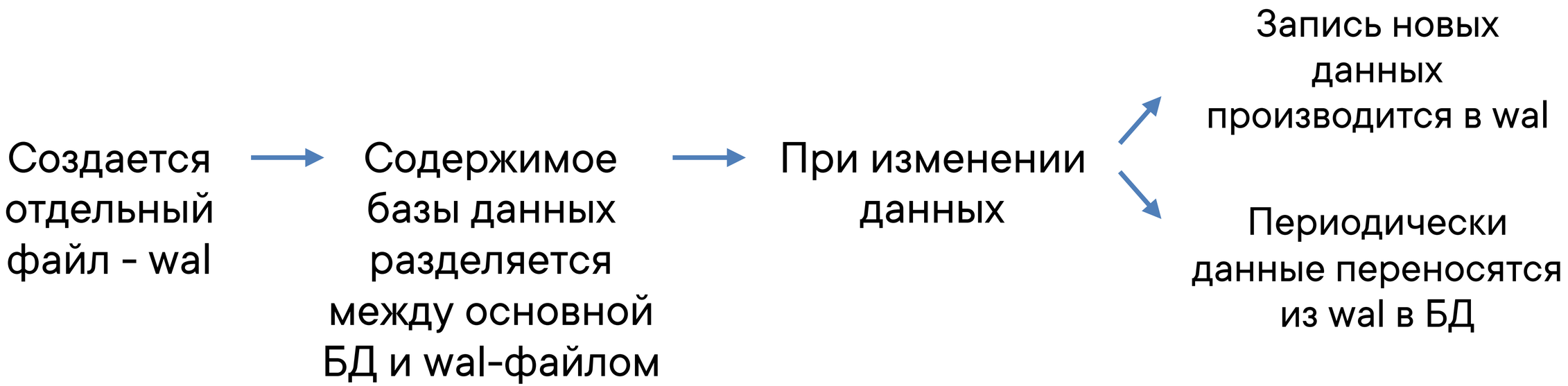
НЕДОСТАТКИ:

Невозможно одновременное чтение и запись в базу данных разными потоками

В Android для режима journal используется только один SQLiteConnection, т.е. все запросы исполняются последовательно

WAL

Принцип работы:



WAL

ПРЕИМУЩЕСТВА:

Параллельное исполнение read/write запросов - write не блокирует read

В Android для режима wal используются 4 SQLiteConnection

В среднем, быстрое завершение транзакции, т.к. данные переносятся в основную БД не сразу

WAL: КАК ВКЛЮЧИТЬ

```
class OpenHelper(context: Context, name: String)
    : SQLiteOpenHelper(context, name, null, 1) {

    override fun onConfigure(db: SQLiteDatabase) {
        db.enableWriteAheadLogging()
        db.execSQL("PRAGMA synchronous=NORMAL")
    }

}
```

JOURNAL VS WAL

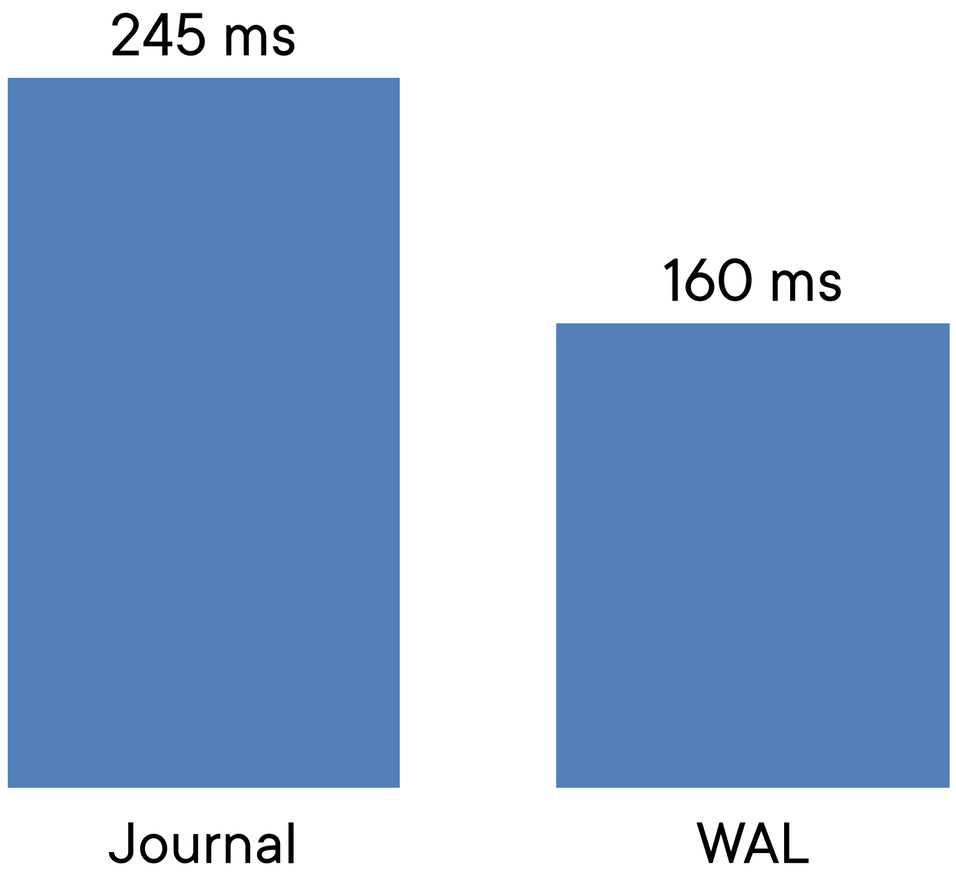
messages
id
text
time

Тест производительности

- Таблица messages
- Оценка длительности вставки 10.000 записей одной транзакцией

https://vk.cc/9p7JFW

JOURNAL VS WAL



~1,5X
быстрее

JOURNAL VS WAL

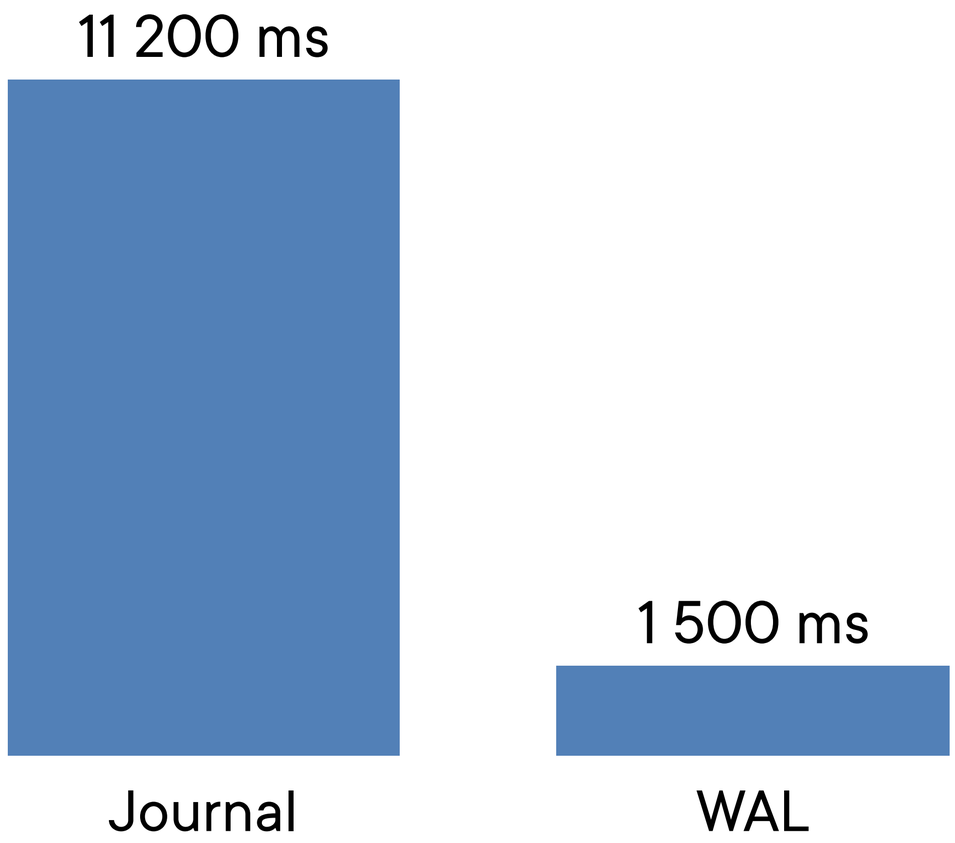
messages
id
text
time

Тест производительности

- Таблица messages
- Вставка 10.000 записей, каждая запись - отдельная транзакция
- Оценка длительности commit транзакции

https://vk.cc/9p7JFW

JOURNAL VS WAL



~7,5X
быстрее

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов
- Включить WAL (Write Ahead Log)
- Уменьшение количества “только читающих” транзакций

УМЕНЬШЕНИЕ КОЛИЧЕСТВА “ТОЛЬКО ЧИТАЮЩИХ” ТРАНЗАКЦИЙ

КОГДА ИСПОЛЬЗОВАТЬ:

Возможно одновременное выполнение тяжелых транзакций, где происходят только SELECT-запросы

Частая запись в базу данных параллельно с чтением

ТРАНЗАКЦИИ

В SQLite / Android:

- EXCLUSIVE = beginTransaction()
- IMMEDIATE = beginTransactionNonExclusive()
- DEFERRED = не поддерживается

ТРАНЗАКЦИИ: EXCLUSIVE

Блокирует любой доступ к БД других потоков

ТРАНЗАКЦИИ: IMMEDIATE

- Считается write-транзакцией
- Только одна такая транзакция может исполняться за раз
- T.e. SELECT-only транзакция может долго выполняться из-за объемного insert на стороннем потоке

ТРАНЗАКЦИИ: DEFERRED

- Позволяет выполнять параллельно “только читающие” транзакции
- Не поддерживается android-оберткой
- db.execSQL(“BEGIN DEFERRED”) - не поможет :(

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов
- Включить WAL (Write Ahead Log)
- Уменьшение количества “только читающих” транзакций
- Собрать свой SQLite :)

CUSTOM SQLITE

КОГДА ИСПОЛЬЗОВАТЬ:

Когда вы сделали все прочие оптимизации и хотите выжать “ещё чуть-чуть”

Потребность в специфичных или новых функциях

CUSTOM SQLITE

МИНУСЫ:

Лишний *.so файл:
~700кб (для armeabi-v7a)

Небольшая трата
времени на загрузку и
инициализацию *.so файла

CUSTOM SQLITE

ВЗЯТЬ ГОТОВЫЙ:

https://sqlite.org/android/doc/trunk/www/install.wiki

CUSTOM SQLITE

СОБРАТЬ СО СВОИМИ НАСТРОЙКАМИ:

https://www.sqlite.org/compile.html

НАШИ НАСТРОЙКИ:

SQLITE_THREADSAFE=2
SQLITE_DEFAULT_MEMSTATUS=0
SQLITE_DEFAULT_PAGE_SIZE=4096
SQLITE_MAX_EXPR_DEPTH=0
SQLITE_DEFAULT_WAL_SYNCHRONOUS=1
SQLITE_LIKE_DOESNT_MATCH_BLOBS
SQLITE_USE_ALLOCA
SQLITE_ENABLE_FTS3
SQLITE_ENABLE_BATCH_ATOMIC_WRITE

SQLITE_DEFAULT_FOREIGN_KEYS=0
SQLITE_DEFAULT_LOCKING_MODE=0
SQLITE_ALLOW_COVERING_INDEX_SCAN=1
SQLITE_OMIT_DECLTYPE
SQLITE_OMIT_SHARED_CACHE
SQLITE_OMIT_AUTHORIZATION
SQLITE_OMIT_BUILTIN_TEST
SQLITE_OMIT_COMPILEOPTION_DIAGS
NDEBUG

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов
- Включить WAL (Write Ahead Log)
- Уменьшение количества “только читающих” транзакций
- Собрать свой SQLite :)

КАК УСКОРИТЬ SQLITE-ЗАПРОСЫ. САМОЕ ВАЖНОЕ

- Упрощение структуры и минимизация количества sql-запросов
- Индексы
- Использование placeholder вместо значений в sql-запросах
- Настройка размера кэша для скомпилированных sql-запросов
- Включить WAL (Write Ahead Log)
- Уменьшение количества “только читающих” транзакций
- Собрать свой SQLite :)

РЕАЛИЗАЦИЯ КЭША СООБЩЕНИЙ ВКОНТАКТЕ. ВЫВОДЫ

СПОСОБ ЗАГРУЗКИ И ДОЛГОСРОЧНОГО ХРАНЕНИЯ ИСТОРИИ

- Сообщения хранят информацию о том, есть ли перед/после них разрывы в истории
- Такой подход позволяет хранить историю сообщений максимально долго и не скачивать все события после запуска приложения
- Долгосрочный локальный кэш улучшает offline-возможности приложения

СПОСОБ ХРАНЕНИЯ СТРУКТУРЫ “СООБЩЕНИЕ”

- Хранится в сериализованном виде в одной SQLite-таблице
- Увеличивается скорость вставки / выборки (за счет меньшего количества запросов)
- Легкая схема поддержки и развития

ОПТИМИЗАЦИЯ РАБОТЫ С SQLITE

- Настройка SQLite в Android “из коробки” не самая оптимальная
- Правильное использование индексов, WAL и транзакций увеличивает скорость запросов в десятки раз

Q&A

СОРОКИН АЛЕКСАНДР, ВКОНТАКТЕ

http://vk.com/alexoro

uas.sorokin@gmail.com