Аудиоотпечатки для индексации всей музыки ВКонтакте

Алексей Акулович

https://ater.me/conf/devconf2016.pdf

DevConf

Про что вообще речь?

400кк аудио файлов
это порядка 4ПБ

+ 150к новых в день
1.5ТБ

Что со всем этим можно делать?

Найти в поиске по названию, взятому из id3 тегов

Что со всем этим можно делать?

Найти в поиске по названию, взятому из id3 тегов

- Дубли в поиске
- Не соответствия названий и содержимого
- И больше никакого функционала...

А что бы хотелось уметь?

- Фильтровать дубли в поиске
- Предлагать варианты лучшего качества
- Обложки, тексты, мета (исполнители, альбомы, …)
- Легализация
- Рекомендации?

А что бы хотелось уметь?

Объединять **похожие на слух**

Требования к устойчивости

- Перекодирование (смена битрейта и т.п.)
- Добавление тишины/шума в начало/конец
- Убирание небольшой части из начала/конца

Требования к устойчивости

- Перекодирование (смена битрейта и т.п.)
- Добавление тишины/шума в начало/конец
- Убирание небольшой части из начала/конца

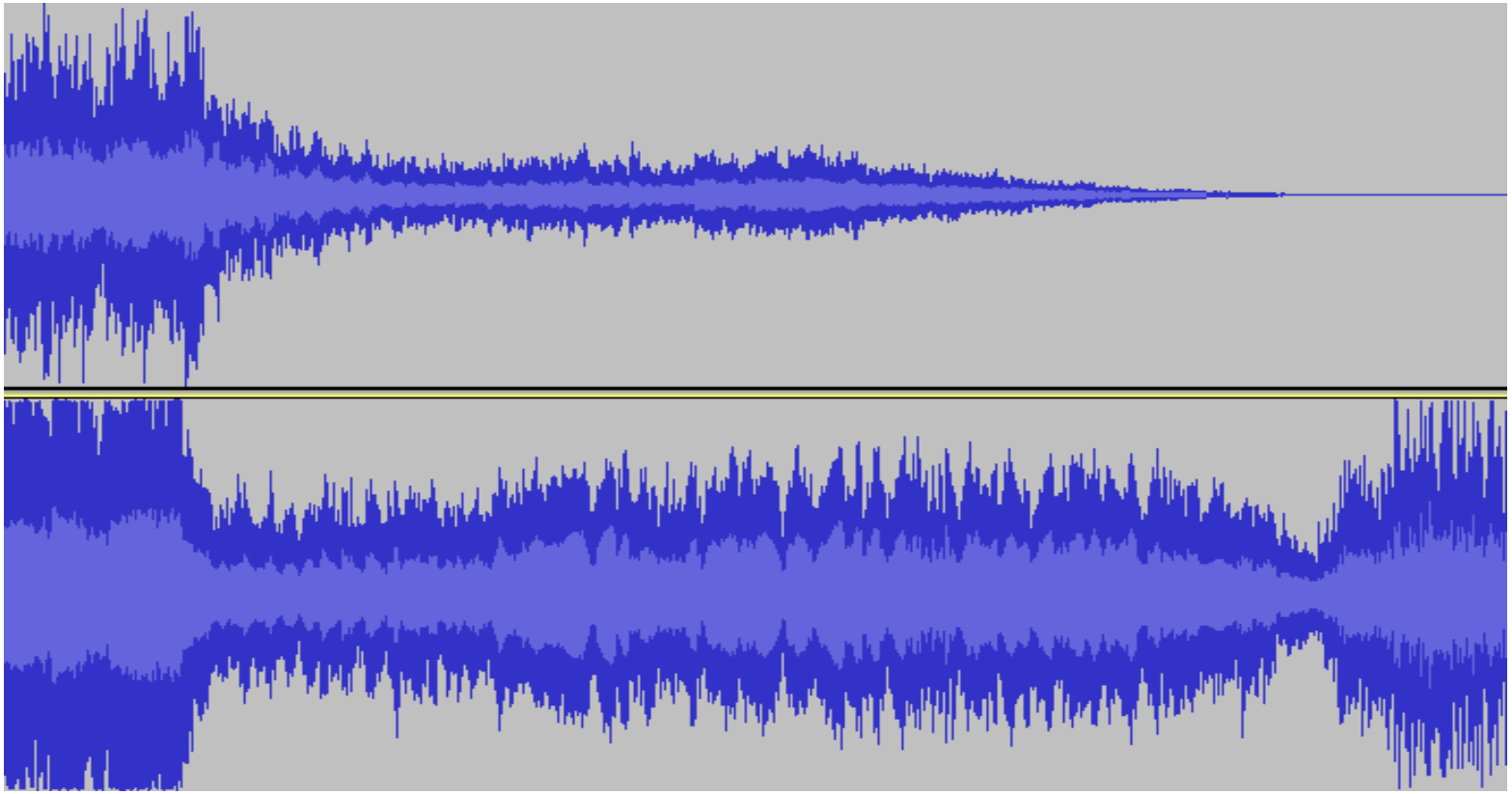
Возьмем готовое решение!

Что вышло?

- Решение лишь создавало отпечатки (cli приложение на C++)
- Поиск по ним мы сочинили свой
- В целом работало

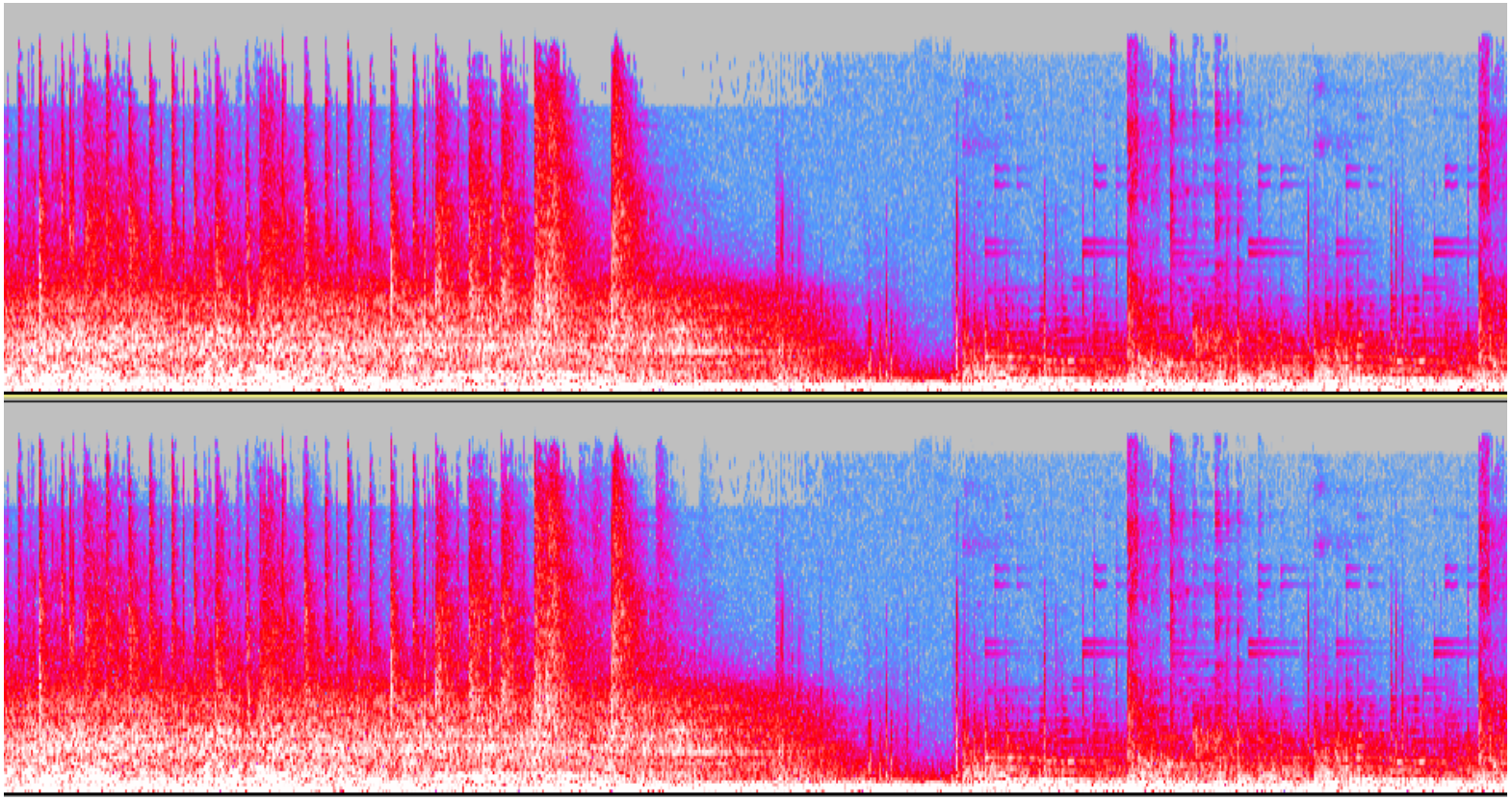
Дополнительные требования

Музыка на фоне, вставки на радио, live



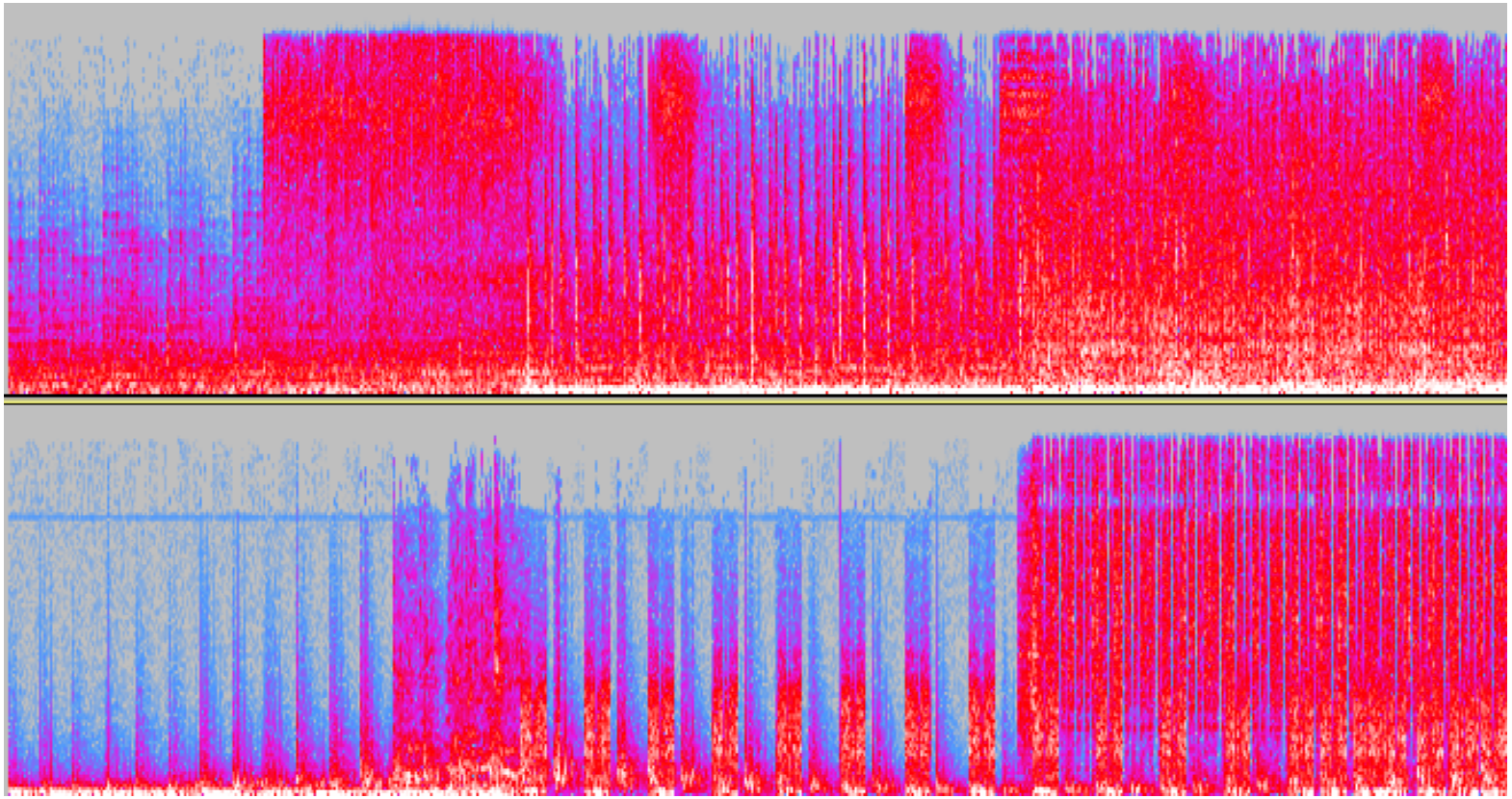
Дополнительные требования

Эквалайзер



Дополнительные требования

Ремиксы, склейки целиком или фрагментов



Дополнительные требования

В идеале и поиск по зашумленному фрагменту

Дополнительные требования

Используемое решение не подошло

Что делать?

Как мы себе это представляли

- Из исходного MP3 достать «сырые» аудио фреймы
- Совершить некую магию
- Получить некие данные (аудио отпечатки)
- Сравнивать треки по отпечаткам

Декодирование MP3

MP3 на вход → аудио данные (звуковая волна) на выходе

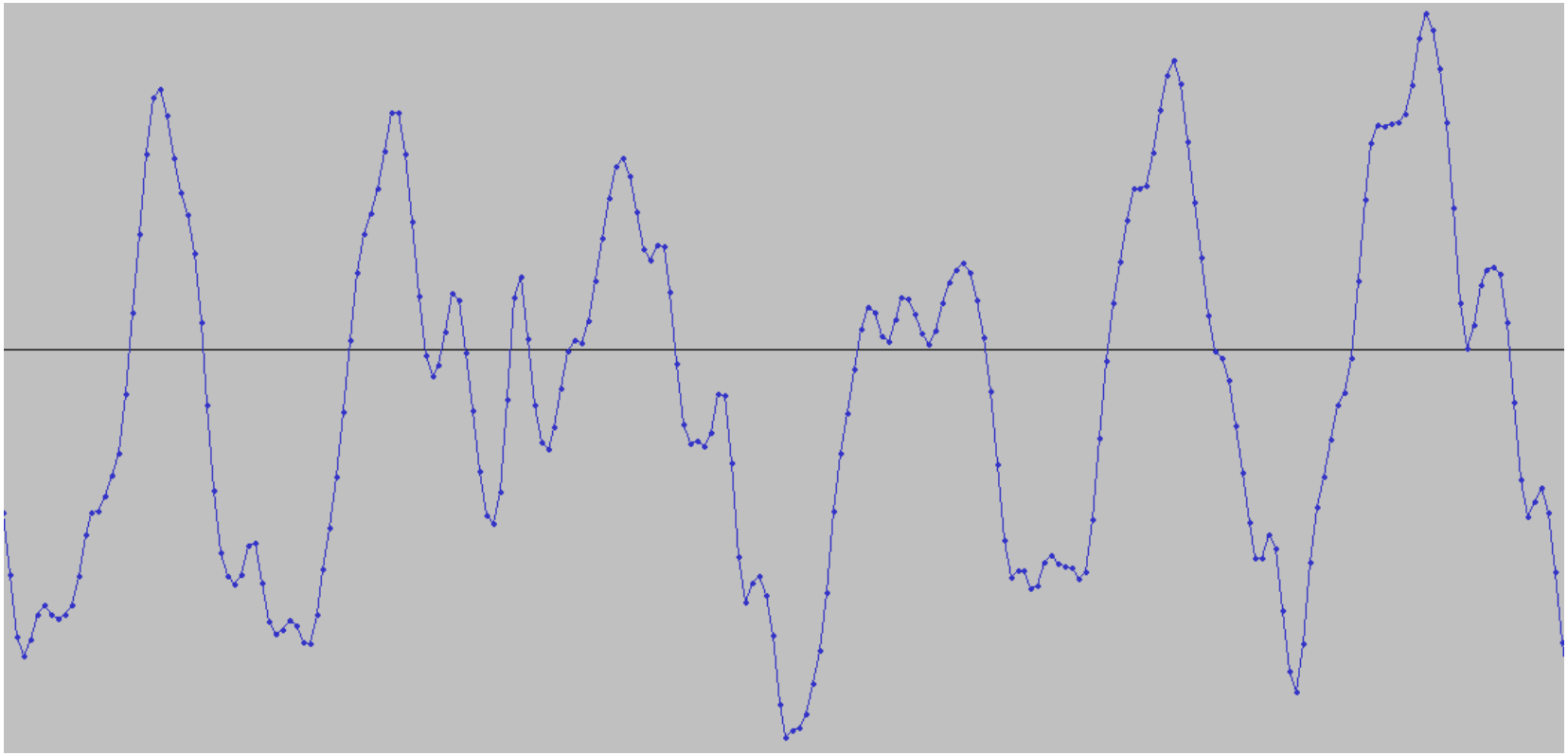
Нативная библиотека **libmad** [1]

\+ свой враппер на Go [2]

1 — http://www.underbit.com/products/mad/

2 — https://github.com/AterCattus/fennec-tiny (файл mad.go)

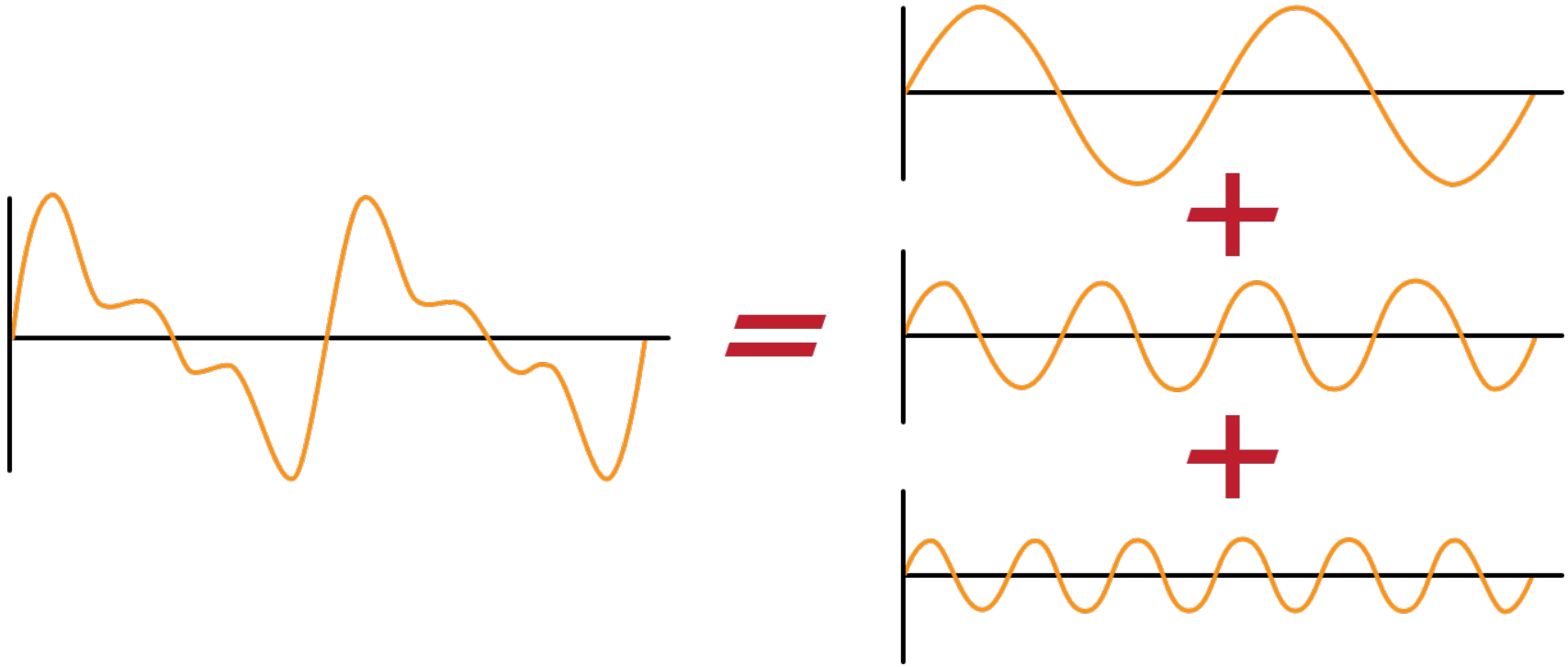
Декодированная амплитуда сигнала



Что делаем?

- ~~Из исходного MP3 достать «сырые» аудио фреймы~~
- Совершить некую магию
- Получить некие данные (аудио отпечатки)
- Сравнивать треки по отпечаткам

Звуковая волна



Преобразование Фурье

БПФ (FFT) на Go

Используем пакет GO-DSP [1]

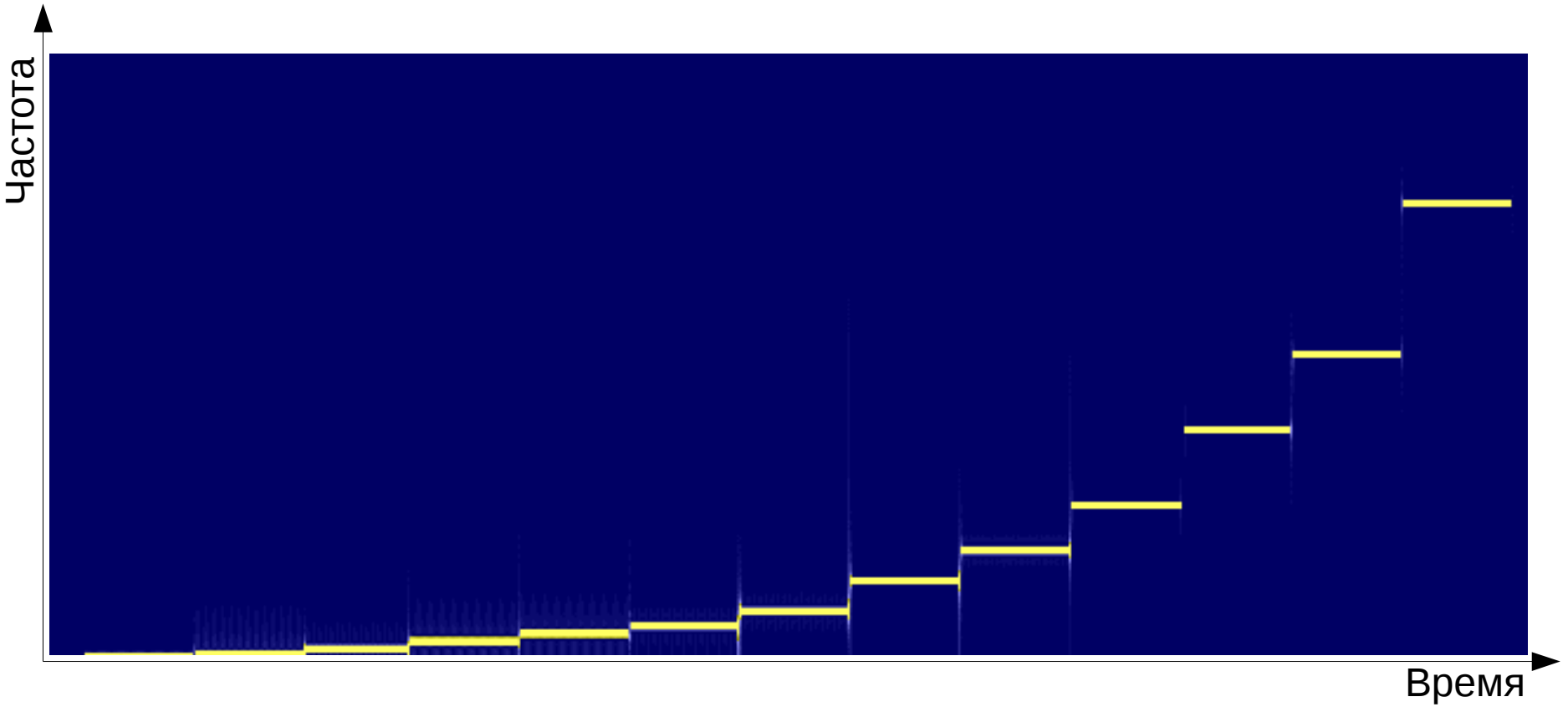
(digital signal processing package)

go-dsp/fft — непосредственно прямое БПФ

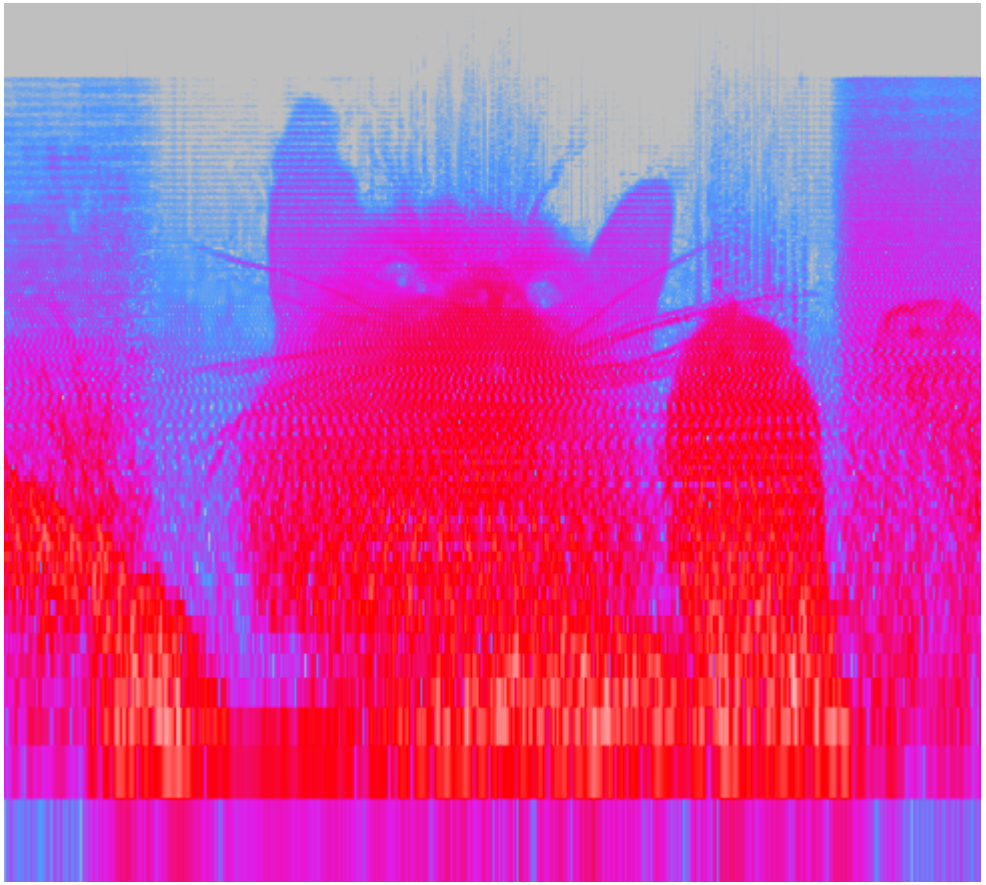
go-dsp/window — формирование оконной функции

1 — github.com/mjibson/go-dsp

Результат FFT - спектрограмма



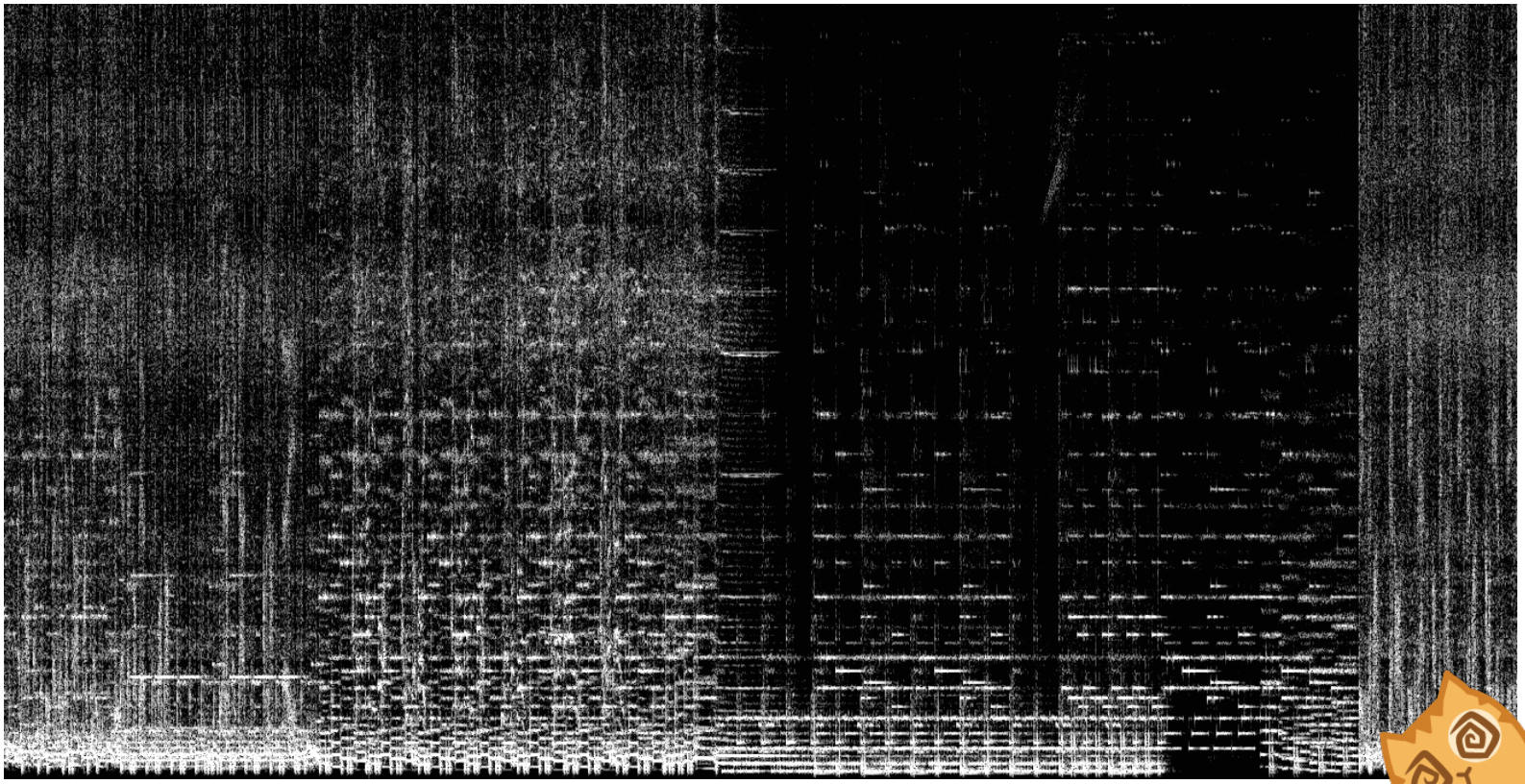
Спектрограмма



Audacity

Venetian Snares - Songs About My Cats

Спектрограмма

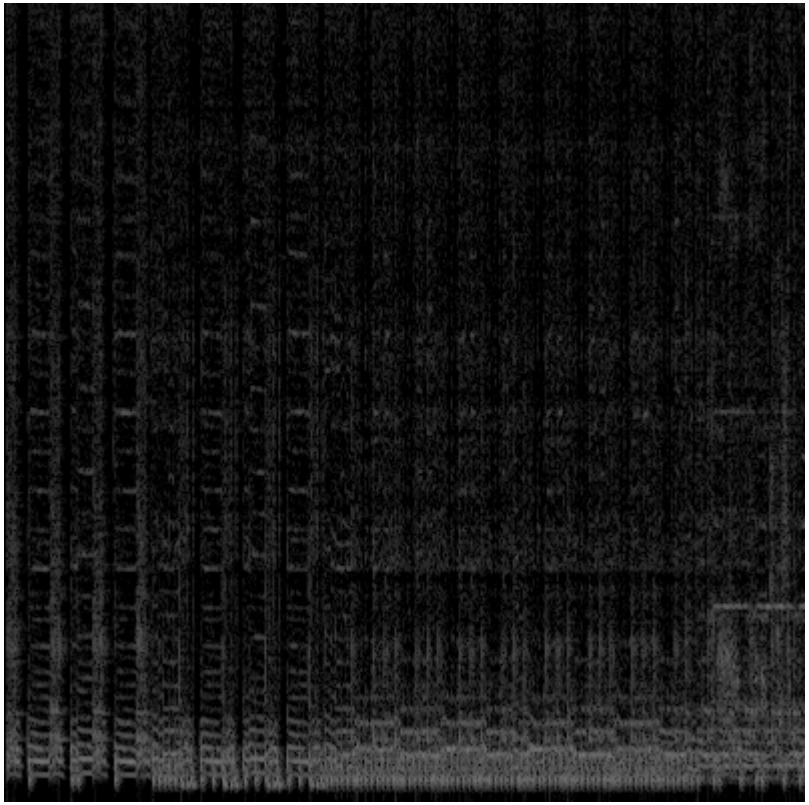


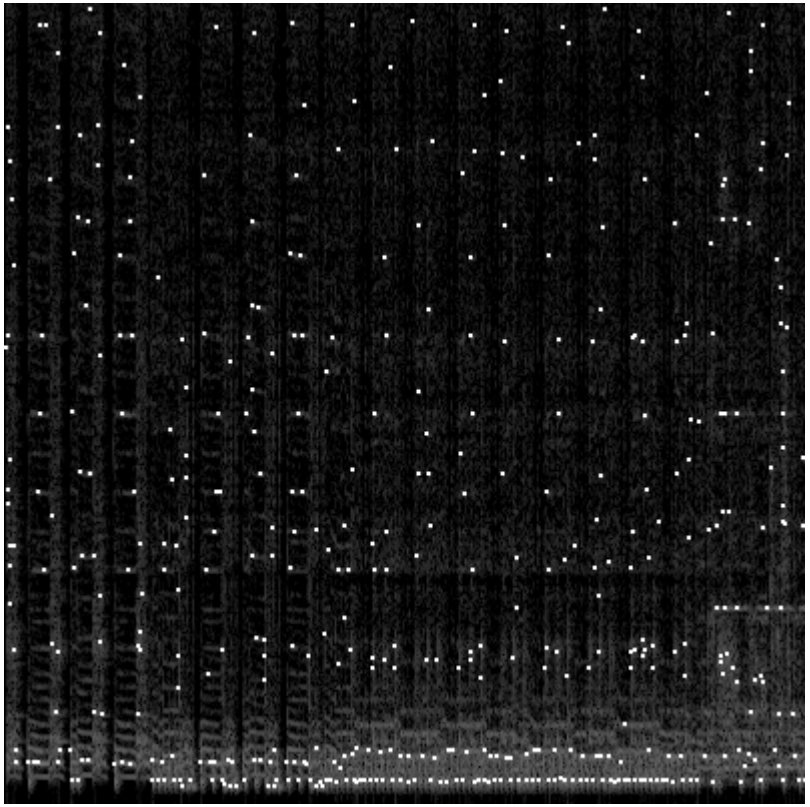
С нашими параметрами это займет 10 ПБ

Что делаем?

- ~~Из исходного MP3 достать «сырые» аудио фреймы~~
- ~~Совершить некую магию~~
- Получить некие данные (аудио отпечатки)
- Сравнивать треки по отпечаткам

Выбор ключевых точек в спектре

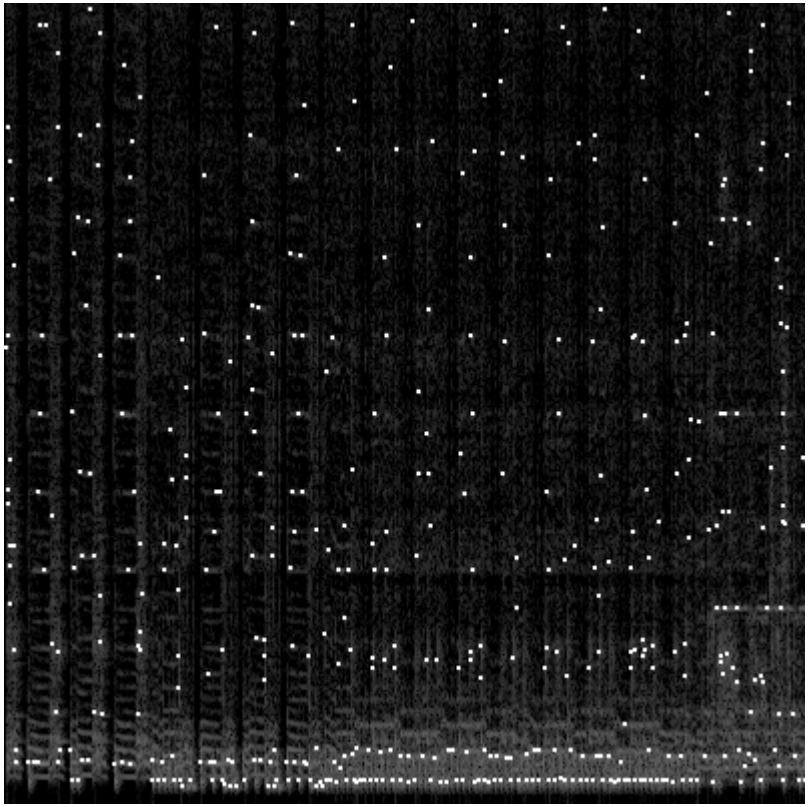




Уменьшаем объем данных в ~200 раз

Получение отпечатка

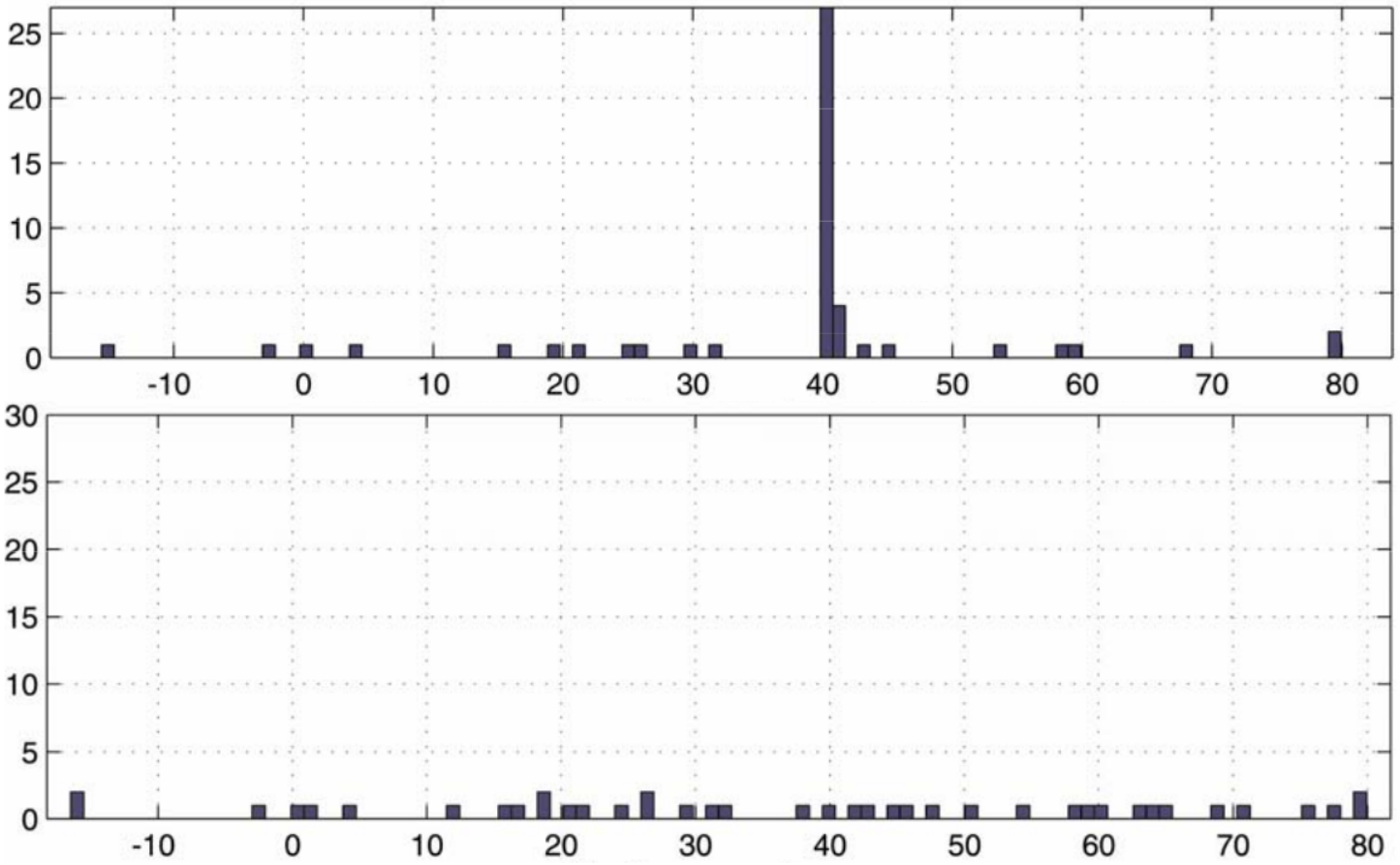
- Пики это (время, частота)
- Время → uint32
- Частота → uint32
- (время, частота) → uint64
- Отпечаток: массив uint64



Что делаем?

- ~~Из исходного MP3 достать «сырые» аудио фреймы~~
- ~~Совершить некую магию~~
- ~~Получить некие данные (аудио отпечатки)~~
- Сравнивать треки по отпечаткам

Сравнение двух отпечатков



Avery Li-Chun Wang, 2003

Что делаем?

- ~~Из исходного MP3 достать «сырые» аудио фреймы~~
- ~~Совершить некую магию~~
- ~~Получить некие данные (аудио отпечатки)~~
- ~~Сравнивать треки по отпечаткам~~

Можно попробовать самим

Реализация ранее описанного публично доступна

https://github.com/AterCattus/fennec-tiny

Fennec SK042 Vulpes zerda Art Print by S-Schukina

Движок генерации отпечатков

- Работает на каждой аудио upload машине
- Его задача: libmad, FFT, фильтрация пиков
- RPC сервер, поддерживающий 1 команду
- Параллелизация на уровне отдельных MP3 файлов
- На Go пишется быстро и получается компактно

Движок генерации отпечатков

- Большую часть времени простаивает
- Но когда работает, то ест много памяти и по ~ядру на файл
- Отработал → runtime.GC() + debug.FreeOSMemory()
- debug.SetGCPercent()
- sync.Pool бесмысленнен
- Для типичной песни (5-7 минут) отрабатывает за ~2-4 сек.

Для аудио книг — линейный рост времени.

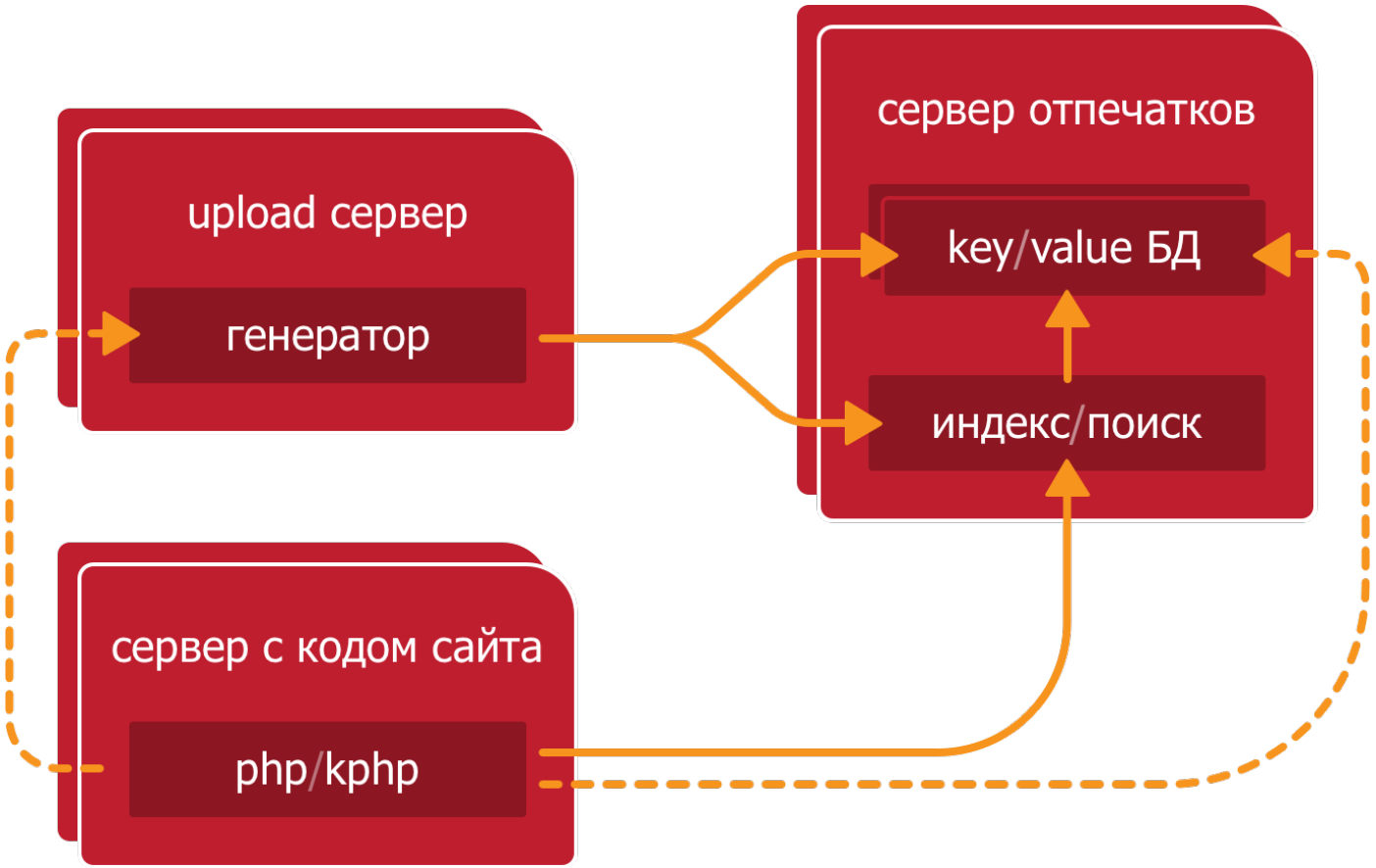
Поиск по базе отпечатков

Получаем 20ТБ отпечатков

- Как и где их все хранить?
- Как по ним искать?

- Кто подставил кролика Роджера?

Архитектура



Движок индексирования и поиска

- Хранит в RAM обратные индексы: какой пик в каких треках есть

- На запрос поиска:
  - Скачивает из хранилища отпечаток
  - Выбирает по нему нужные линии индекса
  - Достает часто встречающиеся треки
  - Для этих треков скачивает из хранилища отпечатки
  - Сравнивает скачанные отпечатки с искомым
  - Отдает N наиболее похожих

Движок индексирования и поиска

- Ориентируемся на 10-25кк треков на процесс движка.

  Это примерно 12-32ГБ RAM на процесс

- Сейчас на 3-4кк треков задержки GC до 1мс.

  0.1% wall cpu

- Работает на ~~16~~ 32 машинах (рядом с key/value БД)

- Без cgo (т. к. не используется libmad), pure Go

Движок индексирования и поиска

Вовсю использует горутины. Параллелится почти все:

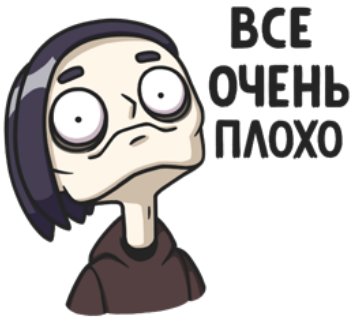
- Работа с сетью (мультиплексирование)
- Глобальный индекс (шардирование sync.RWLock)
- Пулы внутренних воркеров

Движок индексирования и поиска

Насколько движок быстро работает?

Движок индексирования и поиска

Прогноз: 12 месяцев



Движок индексирования и поиска

Был прогноз: 12 месяцев

Повсеместное использование sync.Pool

Получили: 10 месяцев

Движок индексирования и поиска

Был прогноз: 10 месяцев

Использование container/heap

Получили: 5 месяцев

Движок индексирования и поиска

Был прогноз: 5 месяцев

Правки stdlib (свой container/heap)

Получили: 3 месяца

Движок индексирования и поиска

Был прогноз: 3 месяца

Обновления Go 1.5 → 1.6.2

Получили: 2.5 месяца

Go vs. C++ perf

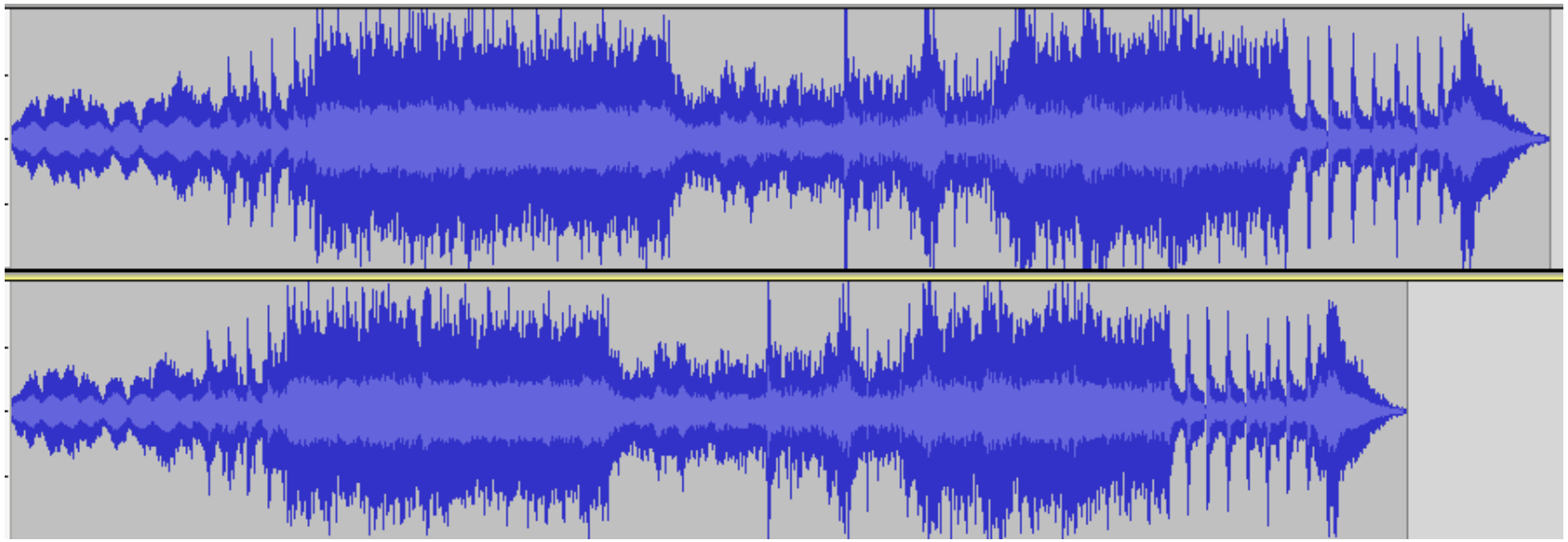
Go	g++	g++ -O3	g++ -O3 via cgo
T	2T	T/3	~T

* Речь про чистые алгоритмические однопоточные вещи

Ooops! на production тестировании

Ooops! на production тестировании

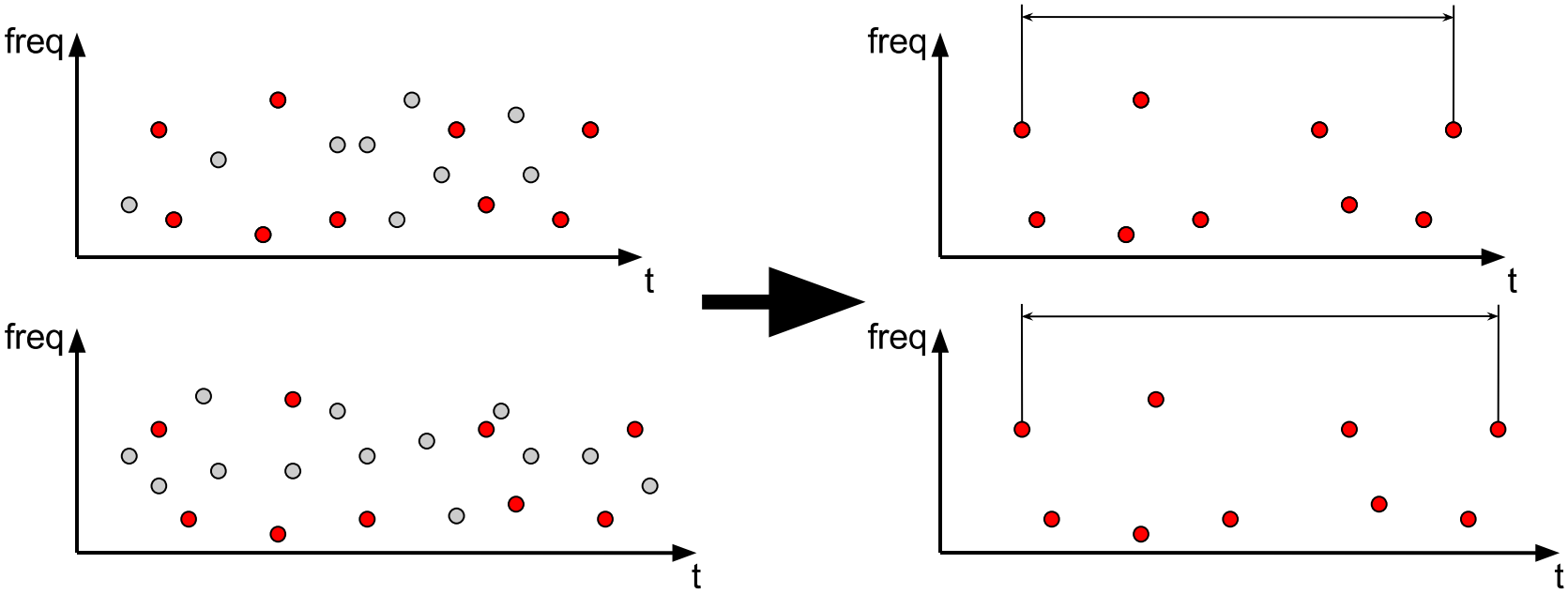
Измененная скорость



Измененная скорость

Наибольшая общая подпоследовательность

LCS (longest common subsequence)

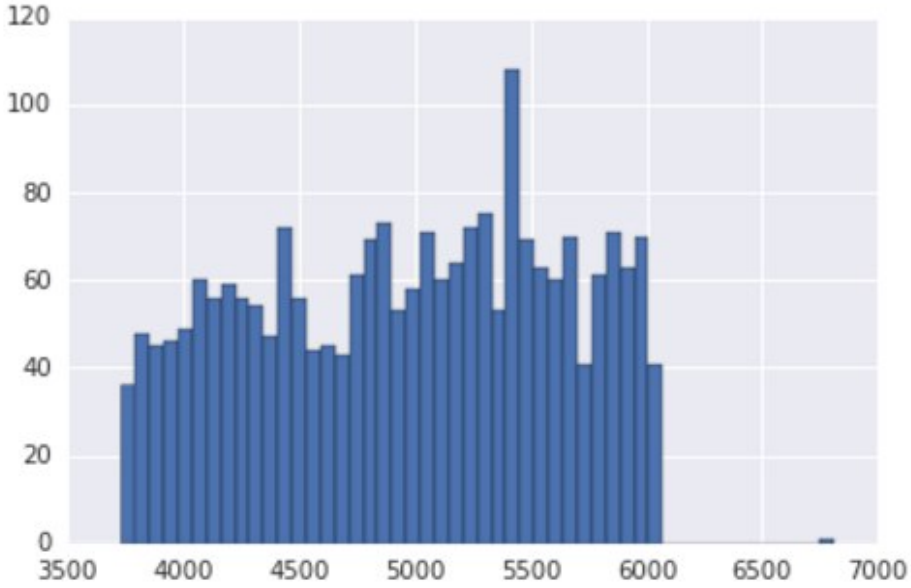


Ooops! на production тестировании

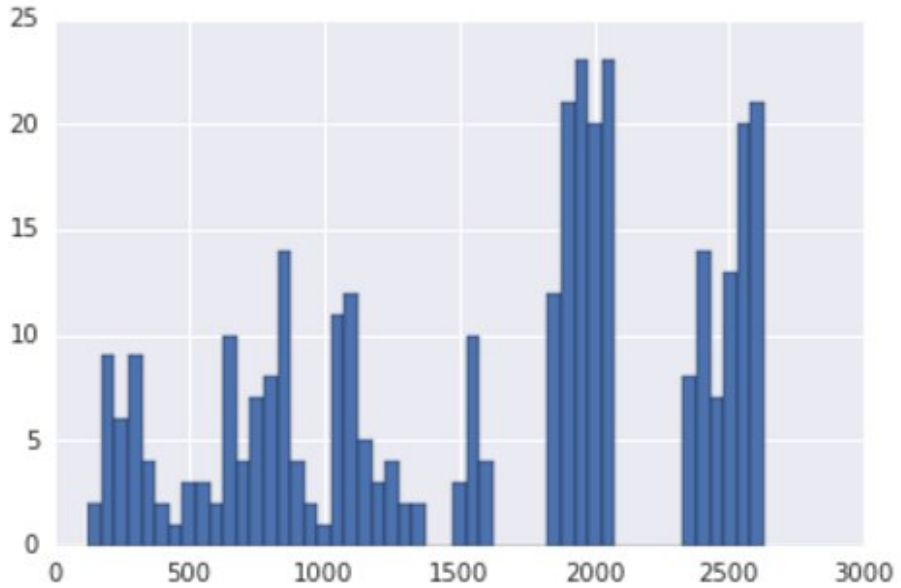
Изменения фрагментов:

- Миксы и склейки
- Версии на разных языках
- «happy birthday»
- реп «школьников»

Изменения фрагментов



Одно совпадение



Несколько совпадений

Выводы

- Что хотели — получили
- Еще есть куда расти
- Можно развивать (рекомендации и т. п.)
- Полный контроль над реализацией
- Использование Go упростило и ускорило разработку

Вот и всё

Презентация:
https://ater.me/conf/devconf2016.pdf

Пример реализации:
https://github.com/AterCattus/fennec-tiny

Для вопросов после:
https://vk.com/ac

И да пребудет с вами Персик!