ТЕХНОСФЕРА

Современные методы и средства построения систем информационного поиска

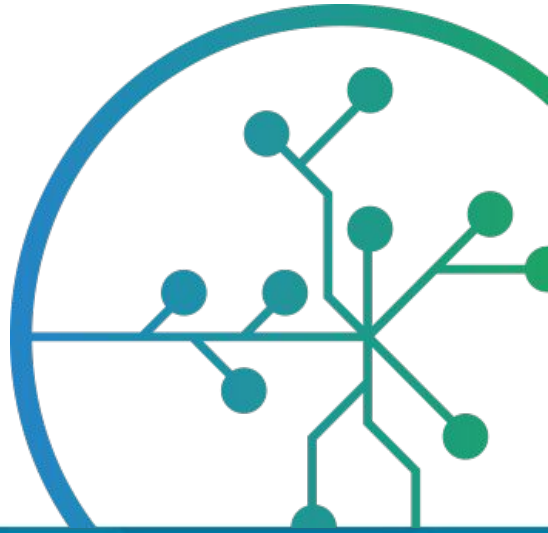
ЛЕКЦИЯ 4: Индексация и булев поиск

Москва, Mail.ru 2018

План лекции

- Состав и назначение индекса
- Несколько слов про аппаратуру
- Быстрое пересечение блоков
- Сжатие индекса
- Приемы увеличения сжатия
- Семинар
- ДЗ

Состав и назначение индекса



Общая схема базы поиска

Общая схема базы поиска

Назначение индекса

- Множество удовлетворяющих запросу документов (булевский поиск)
- Основная предпосылка для текстового ранжирования
- Координаты слов используются при построении сниппетов

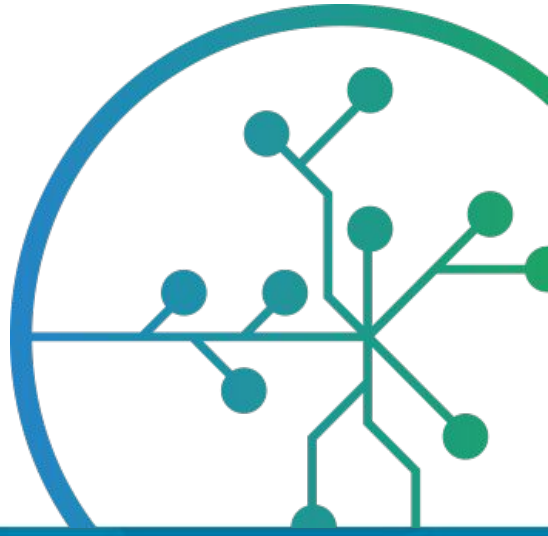
Должен быть

- Быстрым, т.к. основная нагрузка при поиске
- Компактным
  - При зачитывании с дисков
  - Вдвойне, при размещении в RAM
- Гибким как структура данных
  - Хранение атрибутов (Title/H1/Body), ссылок
  - Масштабируемым - разделяемым

Предпосылки для проектирования

- Ограничены оборудованием
- Архитектура всегда соответствует особенностям окружения.
- Начнем с обзора общих ограничений

Аппаратура



Лимиты

- Доступ к данным быстрее если находятся в RAM, не на диске (в 10+ раз)
- Современные жесткие диски – не более 120 random IOPS т. к. позиционирование головки
- Основной принцип оптимизации: чтение большого куска данных выгоднее чтения разрозненных кусков
- IO всегда блочное: считывается как правило 64-256 Кб
- Нет устойчивых к сбоям машин => используем множество машин вместо одной «навороченной»

Лимиты: задача

Имеется диск со скоростью чтения 100Mb/s, временем позиционирования 10ms
Наша задача: посчитать контрольную сумму по файлу 1Gb

- Линейным чтением
- Разрозненным чтением блоков по 1Mb
- Разрозненным чтением блоков по 1Kb

Статистика (2018 год)

Компонент	Данные go.mail.ru
CPU	Xeon: 2x12 core, HT; 2.4 Ghz
RAM	48 Gb, ECC
Диски	1 Tb+ SATA
… seek-time	10 ms (!)

Индекс в объемах

Оценим размеры WEB

– 10+ Млрд документов
– Каждый документ порядка 70 Кб
– Байтов на термин: ~ 8
– Большое число не словарных терминов

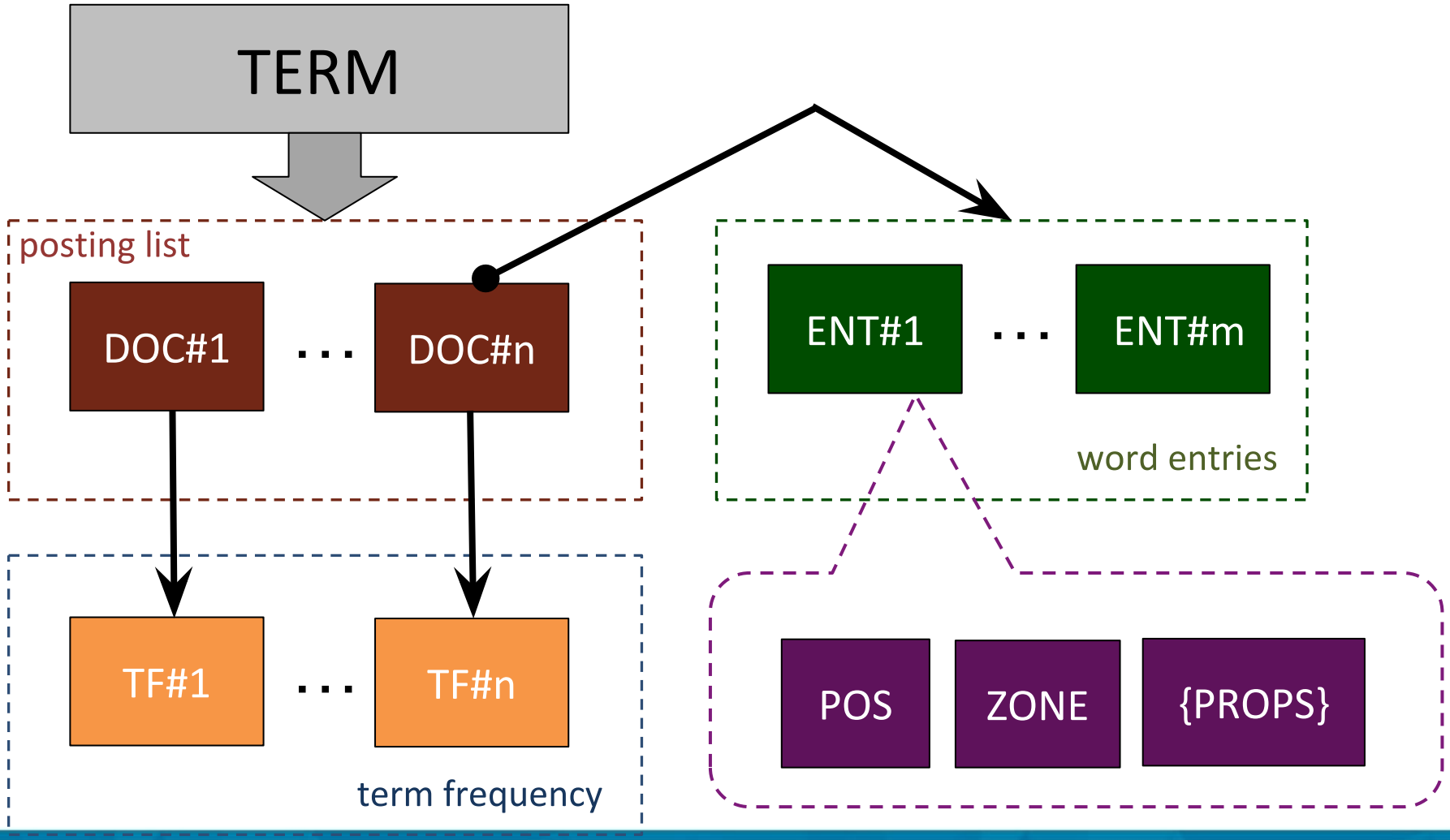
Варианты индекса

- Индекс по словам:
  - Отображение term -> все {docid}
  - Отдельные термы – сервера
  - Как будем хранить подокументные данные?
  - Невыгодно при дальнейшем поиске

Варианты индекса (2)

- Индекс по документам:
  - term -> часть {docid}
  - Отдельные группы документов – сервера
  - Удобно работать с целым документом
  - Удобно отлаживать
  - Ясно как применить к сниппетам
  - Подокументный поиск выгоднее технически

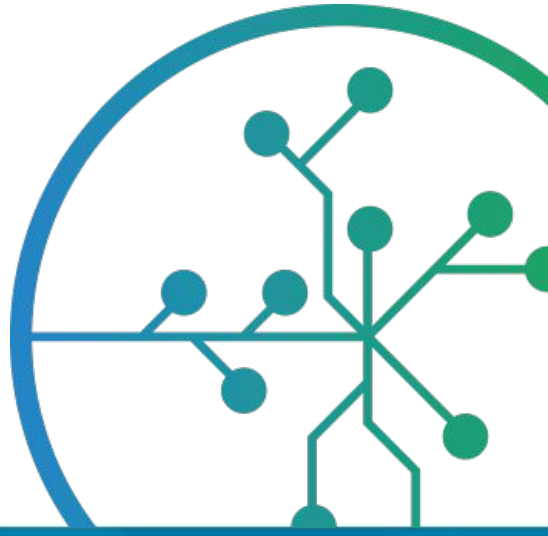
Состав обратного индекса



Что нужно в runtime

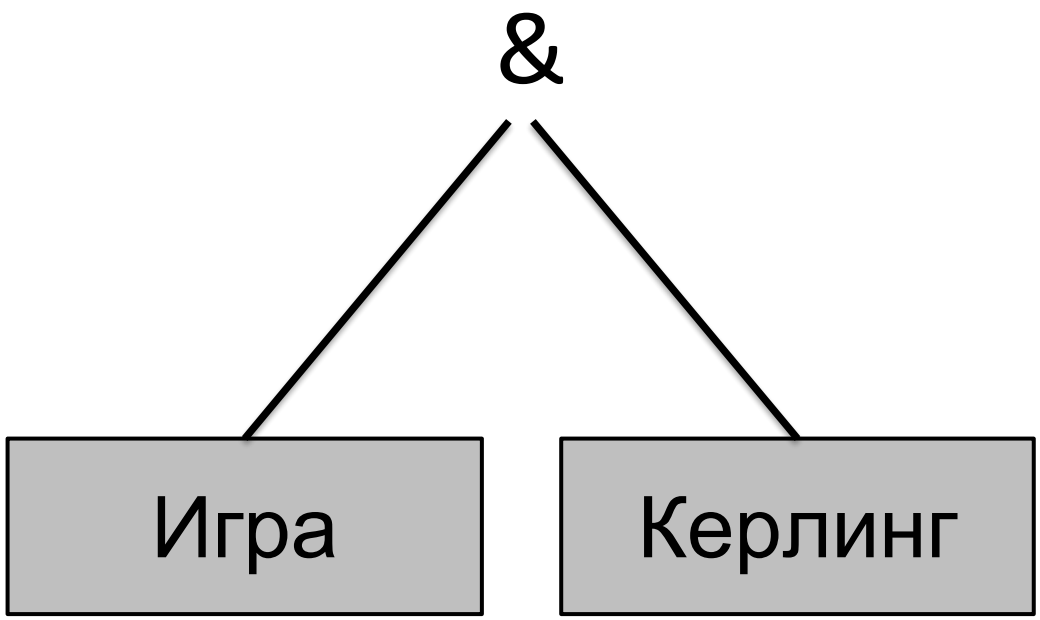
- Можем рассмотреть на примере пересечения массивов целых чисел
- создадим компактные массивы
- … с быстрым объединением

Быстрое пересечение блоков

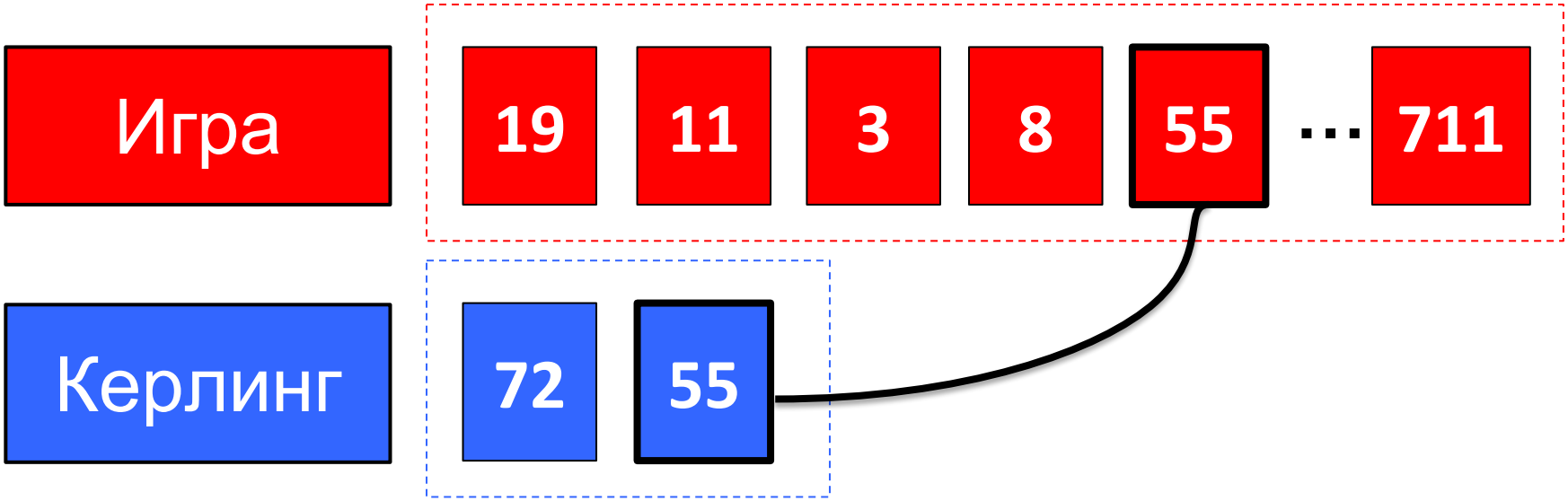


Типичный запрос

В виде дерева



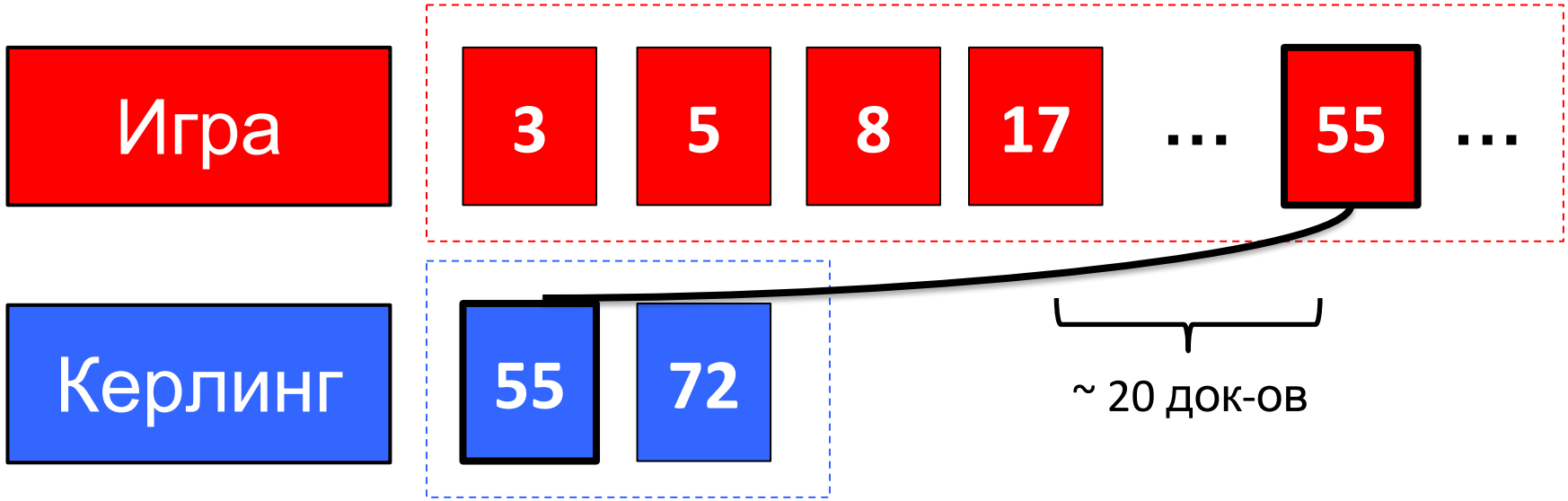
Posting lists



|P(Игра)| = **1000** x |P(Керлинг)|

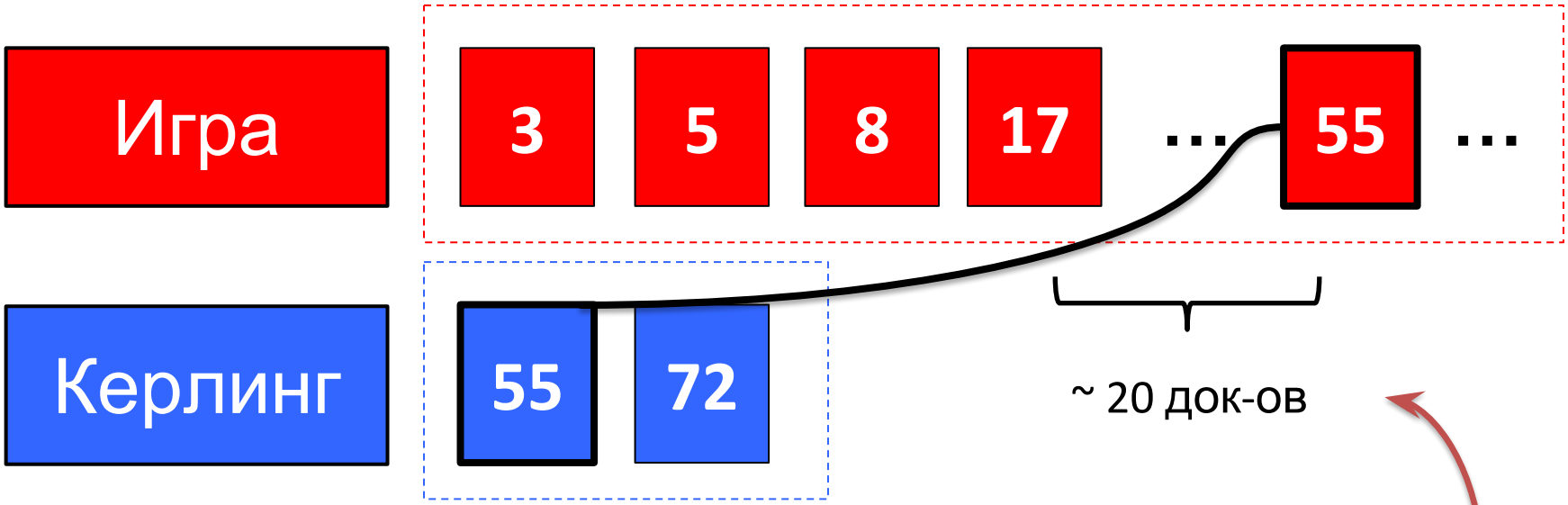
Оптимизация пересечения

1. Отсортируем списки



Оптимизация пересечения

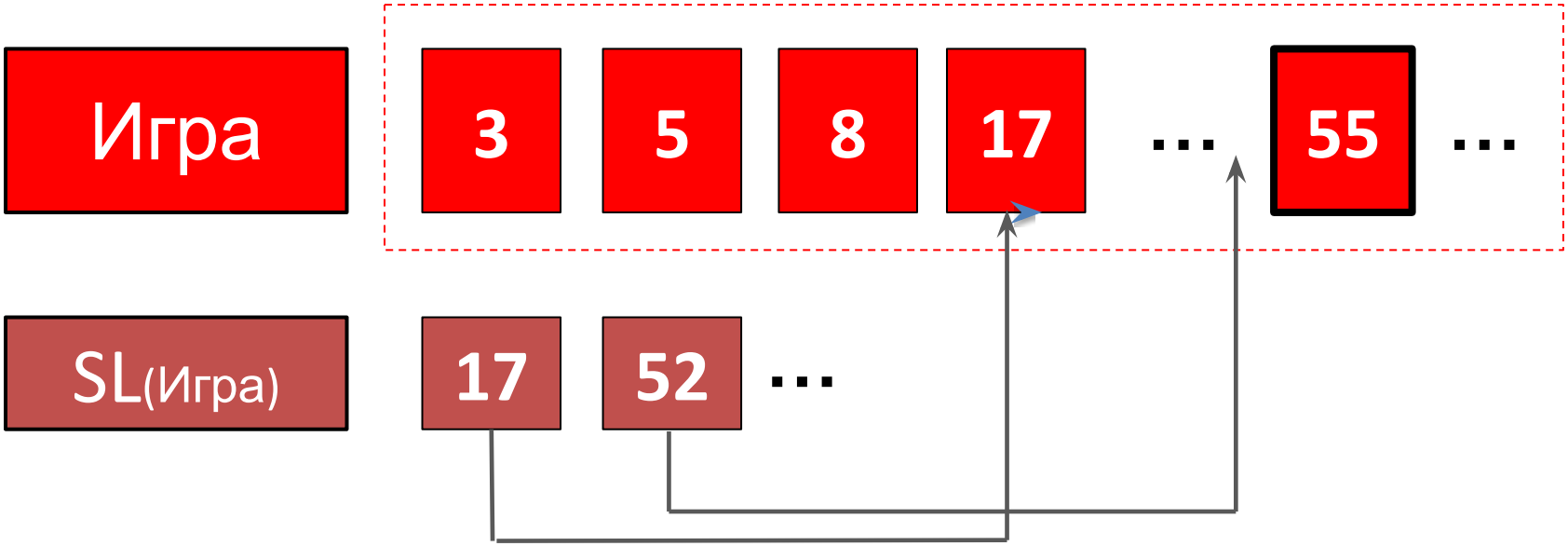
2. Будем искать двоичным поиском по оставшейся части?



Поиск произвольными перемещениями не эффективен для современных CPU

Skip List

3. Используем списки пропусков

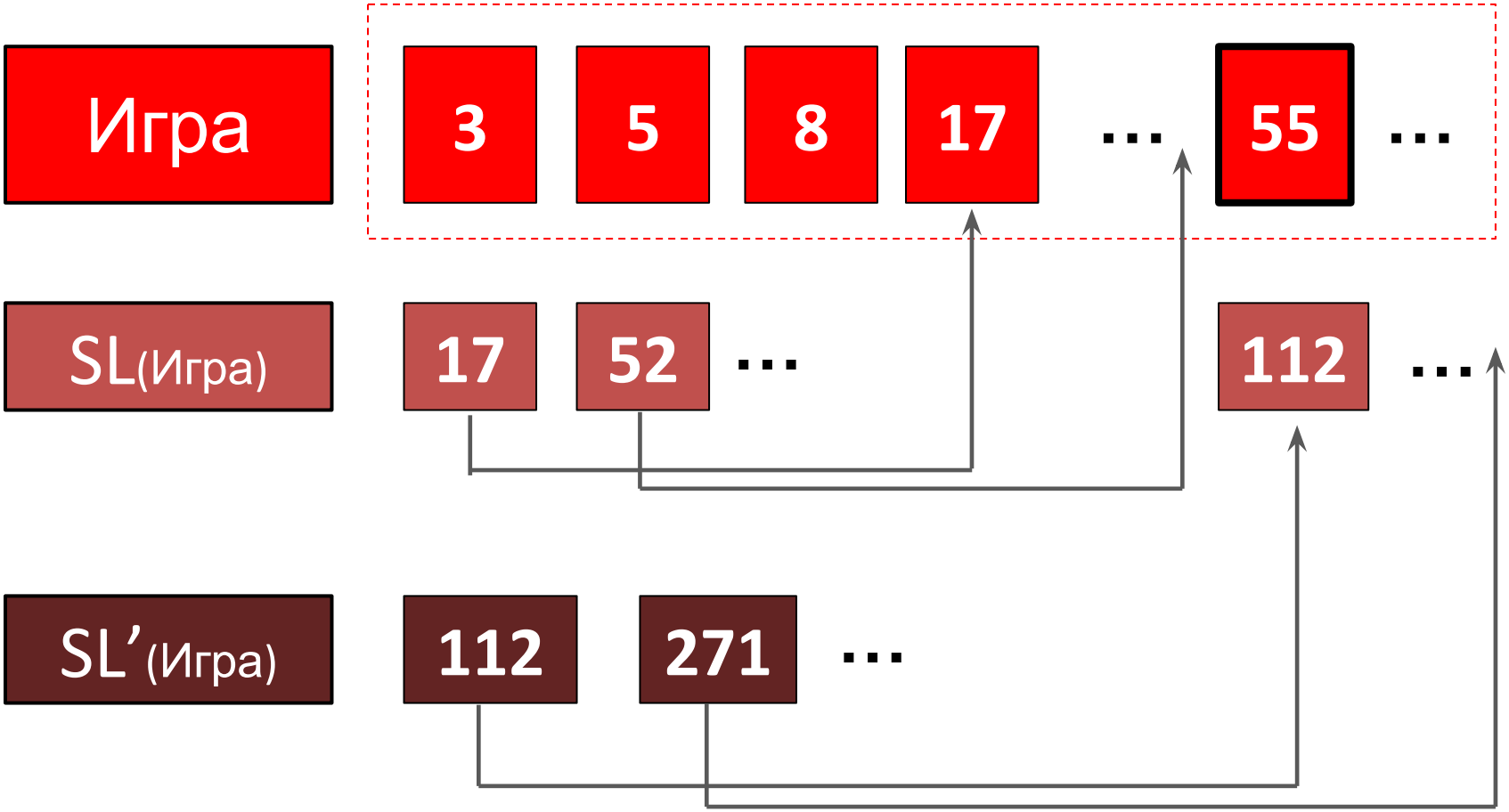


Размещаем смещение идентификатора документа

Что если и этого мало?

More Skip List

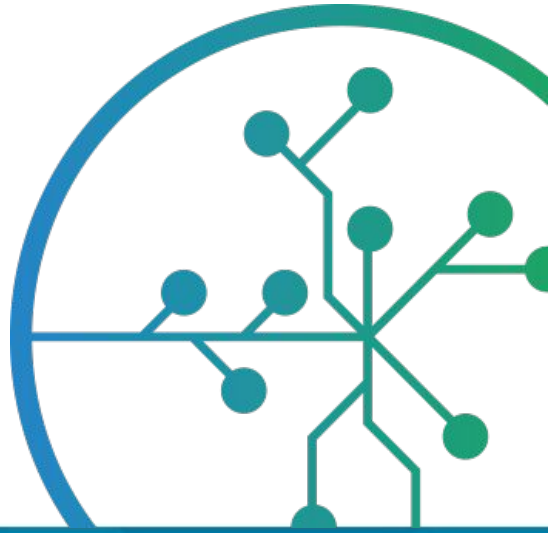
3. Используем таблицы-индексы дважды – почему нет?



Двигаемся дальше

- ОК, знаем как быстро пересекать списки
- А что с размером?

Сжатие индекса



Зачем сжимать

- Экономим место
  - Особенно если RAM
- Больше помещается в память
  - Быстрее передача данных
  - {Прочитать сжатое, распаковать} может быть быстрее чем {прочитать несжатое}
  - Больше можно закешировать

Виды сжатия

- Сжатие без потерь: вся информация остаётся как есть
  - gzip/rar/…
  - png
  - Обычно используем её в ИП.

Сжатие с потерями

Сжатие с потерями

- Что-то считаем возможным убрать
- Понижение капитализации, стоп-слова, морф. нормализация – может рассматриваться как сжатие с потерями.
- Ещё – удаление координат для позиций, которые вряд ли будут вверху на ранжировании.

Сжатие координатных блоков

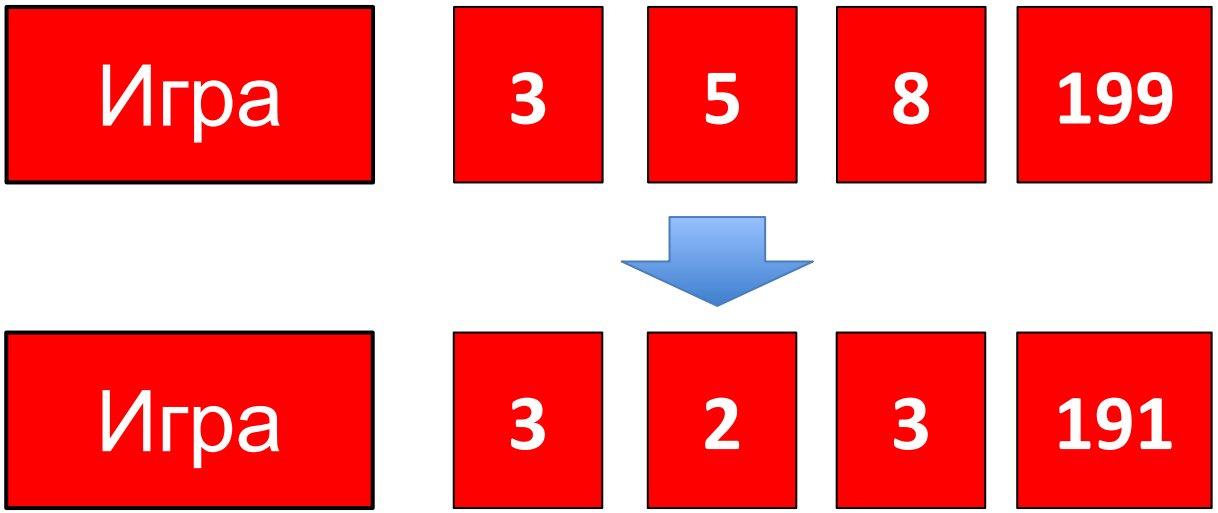
- Координатные блоки – существенная часть обратного индекса
- Будем сжимать каждый постинг
- В булевском индексе – это docID

Цель

- Предположим, мы индексируем 1 МЛН документов
  - можем использовать $\log_2 1{,}000{,}000$
  - 20 битов на DocID
- Наша задача: значительно меньше, чем 20 битов

Подготовка к компрессии

- Список документов выгодно хранить по возрастанию DocID
- Следовательно, можем кодировать промежутки



Цель кодирования

- Если средний промежуток размера G, мы хотим использовать ~$\log_2 G$ битов на промежуток.
- Главное: кодировать каждое целое число минимальным количеством битов.
- Требуется код с переменной длиной.
- Будем достигать желаемого тем, что будем назначать короткие коды небольшим промежуткам

Код Variable Byte (VB)

- Храним признак окончания числа
- Число G < 128 кодируется одним байтом
- Иначе берем остаток, и кодируем его тем же алгоритмом
- Для последнего байта c=1, для остальных c=0

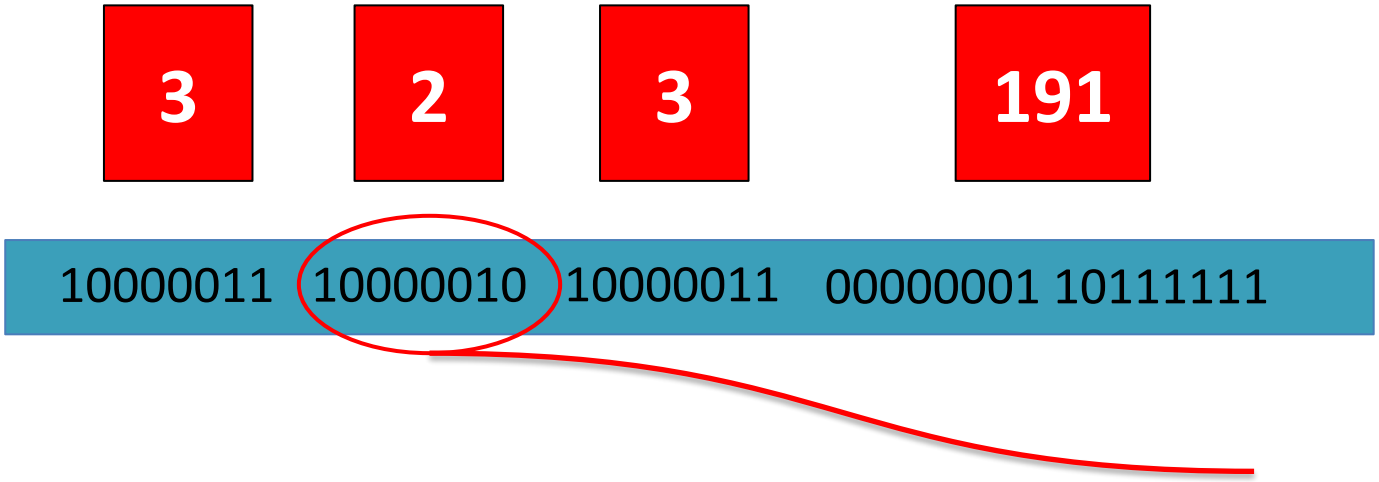
Пример кодирования varbyte

Записываем в виде непрерывной строки бит

10000011 10000010 10000011 00000001 10111111

+: Простота реализации
+: Хорошая скорость
+: Эффективно для CPU

Но есть и минусы



+: Простота реализации
+: Хорошая скорость
+: Эффективно для CPU

-: Гранулярность = 1 байт

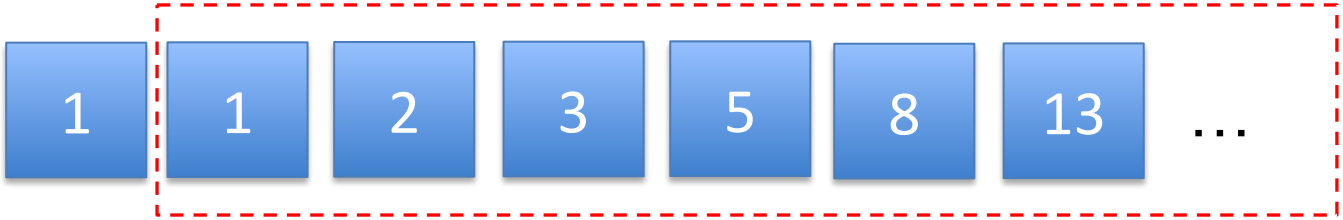
От байт к битам

- Кодирование по байтам избыточно для малых промежутков
- Будем использовать битовое кодирование

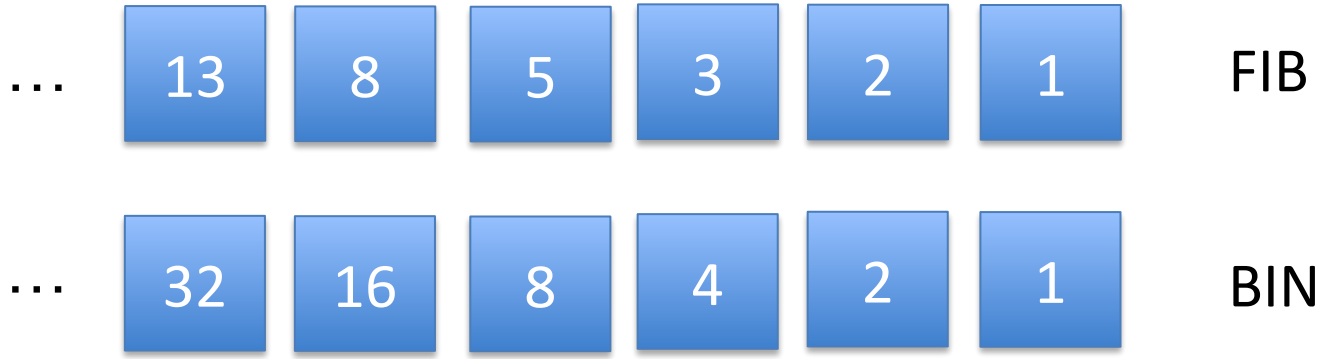
- Важное требование побитового сжатия:
  - Кодирование длины
  - Или недопустимая последовательность

Код Фибоначчи

Код Фибоначчи



Представим основанием СС:



Код Фибоначчи

- 11 – недопустимая комбинация
- Алгоритм довольно простой
- Сжимает эффективней varbyte

Код Фибоначчи: пример

- Закодируем число 11
- ”11” => 8+3 => 1,0,1,0,0
- Как определить конец числа?

Гамма-код (Elias Gamma)

- Кодирование:
  1. Записываем число в 2-ой форме
  2. Перед двоичным представлением числа дописать нули. Кол-во нулей на единицу меньше двоичного представления числа

Гамма-код (Elias Gamma)

- Кодирование:
  1. Записываем число в 2-ой форме
  2. Перед двоичным представлением числа дописать нули. Кол-во нулей на единицу меньше двоичного представления числа

Примеры:

Число	2е предст.	Кодирование
1	$2^0 + 0$	1
2	$2^1 + 0$	010
3	$2^1 + 1$	011
4	$2^2 + 0$	00100
5	$2^2 + 1$	00101

Гамма-код (Elias Gamma)

- Декодирование:
  1. Считываем нули, пусть = N
  2. Первая единица это 2^N. Считываем оставшиеся разряды числа.

Гамма-код (Elias Gamma)

- Все гамма коды имеют нечётное количество битов
- В два раза хуже лучшего результата, $\log_2 G$
- Гамма коды – префиксные коды, как и VB
- Не требует параметров.

Гамма-код (Elias Gamma)

- Нужно учитывать границы машинных слов – 8, 16, 32, 64 бит
  - Операции, затрагивающие границы машинных слов, значительно медленнее
- Работа с битами может быть медленной.
- VB кодировка выровнена по границам машинных слов и потенциально более быстрая.
- VB значительно проще в реализации.

Rice Encoding

- Рассмотрим среднее кодируемых чисел, =*g*
- Округлим *g* до ближайшей степени 2, =b
- Каждое число x будем представлять как
  - (*x*-1)/*b* в унарном коде
  - (*x*-1) mod *b* в бинарном коде

RiceEncoding пример

DocID: 34, 178, 291, 453

Промежутки: 34, 144, 113, 162

Среднее: $g = (34+144+113+162)/4 = 113{,}33$

Округляем: $b = 64$ (6 бит)

Число	Разложение	Кодирование
34	64*0 + (34-1)	0 100001
144	64*2 + (144-1) & 63	110 001111
113	64*1 + (113-1) & 63	10 110000
162	64*2 + (162-1) & 63	110 100001

Свойства RiceEncoding

- Можно подобрать g как для всего индекса, так и для отдельного терма
- Более того – можно подобрать для отдельных промежутков

- Лучше сжимает, но медленнее VarByte

Также см. Golomb Encoding

Семинар

- Скачайте https://cloud.mail.ru/public/8zrx/XLdMqLE8G
- Запустите jupyter-notebook
- (Выберите Python2 если у вас 3-й)

Подведем итог

- Можем быстро объединять списки
- Применяя сжатие значительно уменьшаем размер индекса (-400%)

Домашнее задание

- https://github.com/oleg-safonov/ts-idx-2018

Спасибо!

Вопросы?