ТЕХНОСФЕРА

Современные методы и средства построения систем информационного поиска

ЛЕКЦИЯ 7: Индексация и булев поиск

Москва, Mail.ru 2016

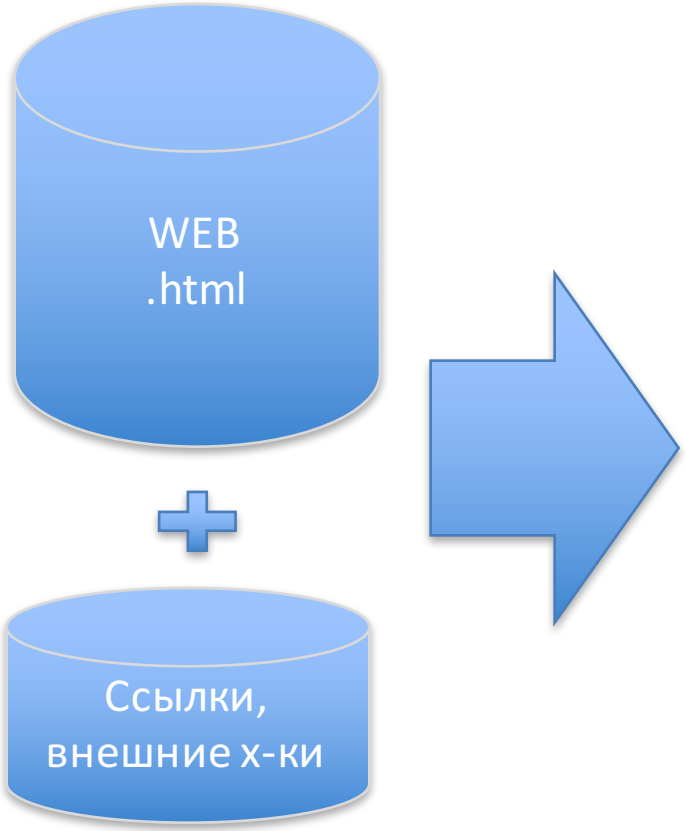
Прежде чем приступить…

План лекции

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

Состав и назначение индекса

Общая схема базы поиска



- Обратный индекс
- Копии документов для построения сниппетов
- Числовые характеристики документов
- Числовые характеристики сайтов

Общая схема базы поиска

Назначение индекса

- Список 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

- Много архитектурных решений основаны на ограничениях, накладываемых используемым оборудованием.
- Архитектура всегда соответствует особенностям окружения. Кол-во запросов, надежность оборудования, и даже … внешний вид выдачи.
- 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»

Статистика (2016 год)

Компонент	Данные go.mail.ru
CPU	Xeon: 2x8 core, HT; 2.4 Ghz
RAM	48 Gb, ECC
Диски	1 Tb+ SATA
… seek-time	10 ms (!)

Дисковая подсистема

- Можно поставить несколько дисков
  - JBOD (just bunch of disks)
  - RAID (Redundant Array Of Independent Disks):
    - RAID-0 (stripe)
    - RAID-1 (mirror)
    - RAID-5, …
  - Если RAID, то аппаратный или программный?

Диски бывают

- SCSI, SATA, SAS (Serial Attached SCSI)
- SSD
- 5400 RPM, 7200 RPM, 10000RPM, 15000RPM
- 2”, 3”

SATA/SAS/SSD – что выбрать

- SATA дешевый в пересчете $/Мб
  - 8ms seek latency
  - большинство – desktop
- SAS надежный, но относительно дорогой
  - 3.8ms seek time
  - Рассчитан на 100% workload
- SSD
  - 40000 IOPS – норм
  - Надежность при постоянной записи - низкая

Индекс в объемах

Оценим размеры WEB

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

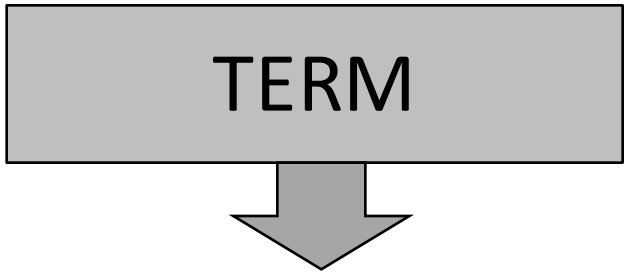
Варианты индекса

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

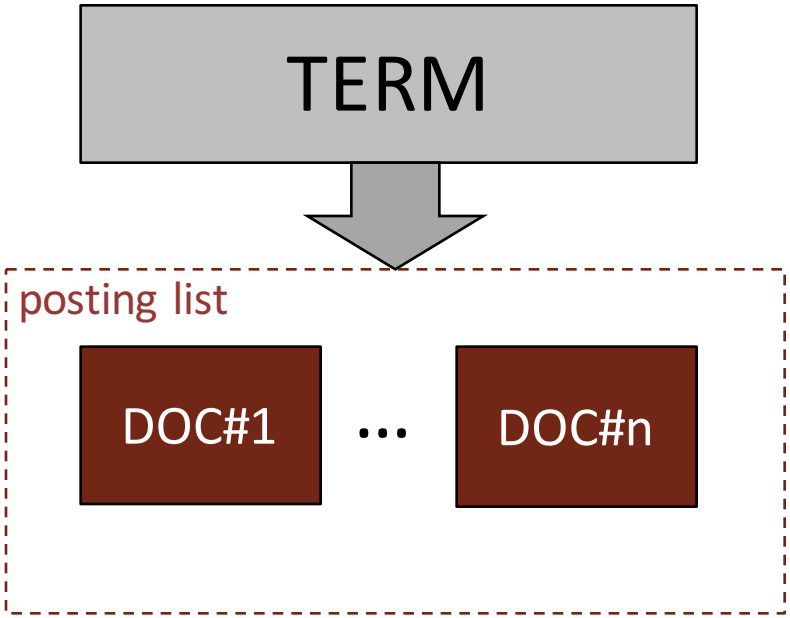
Варианты индекса (2)

- 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

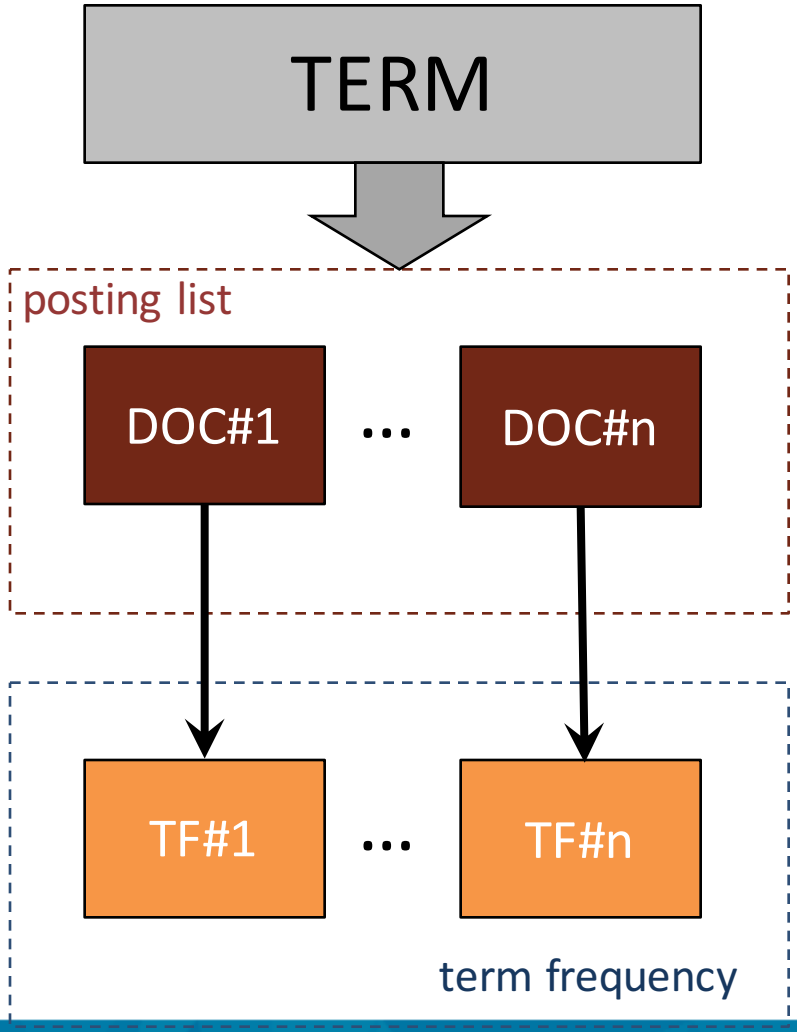
Состав обратного индекса



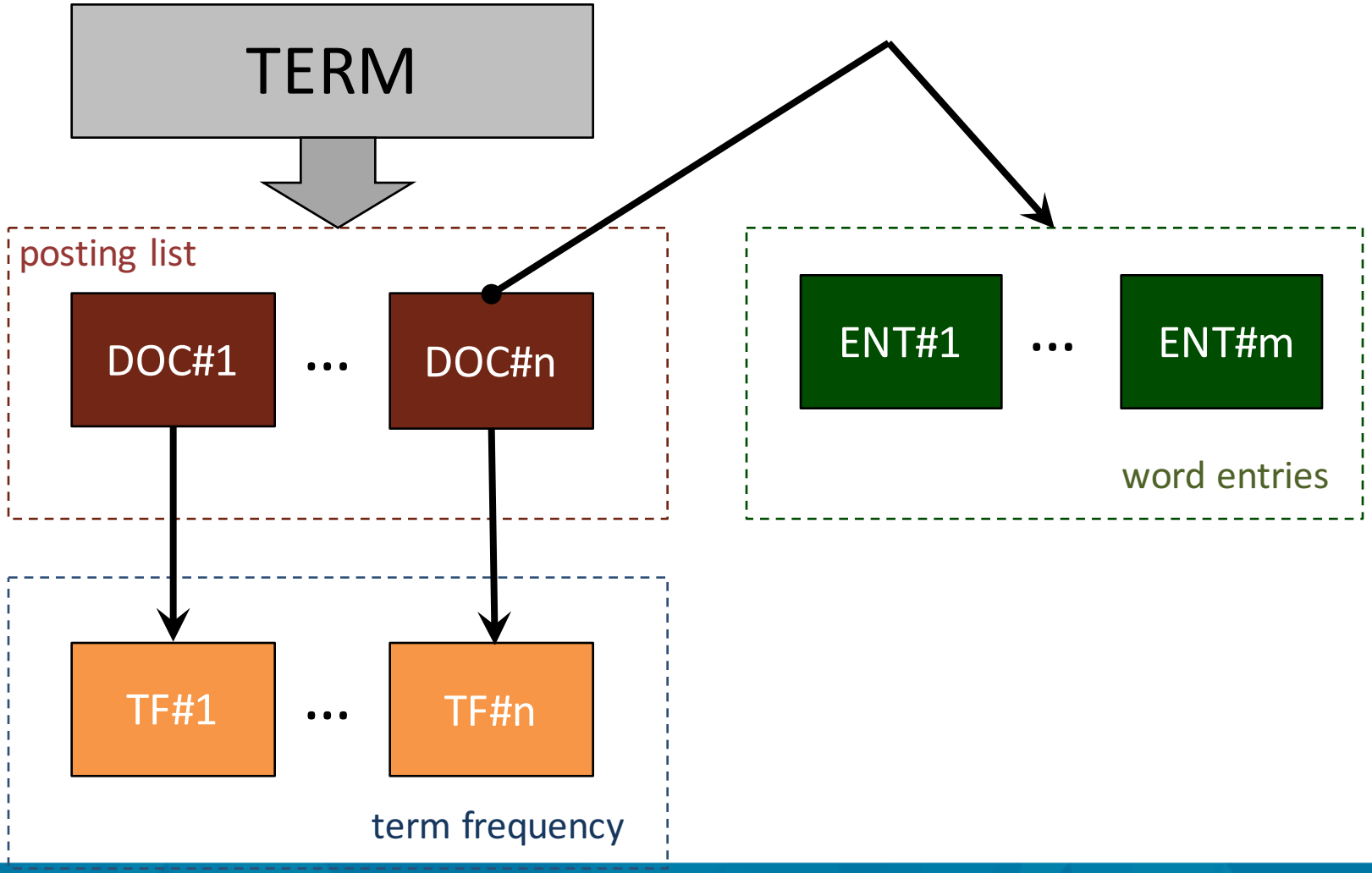
Состав обратного индекса



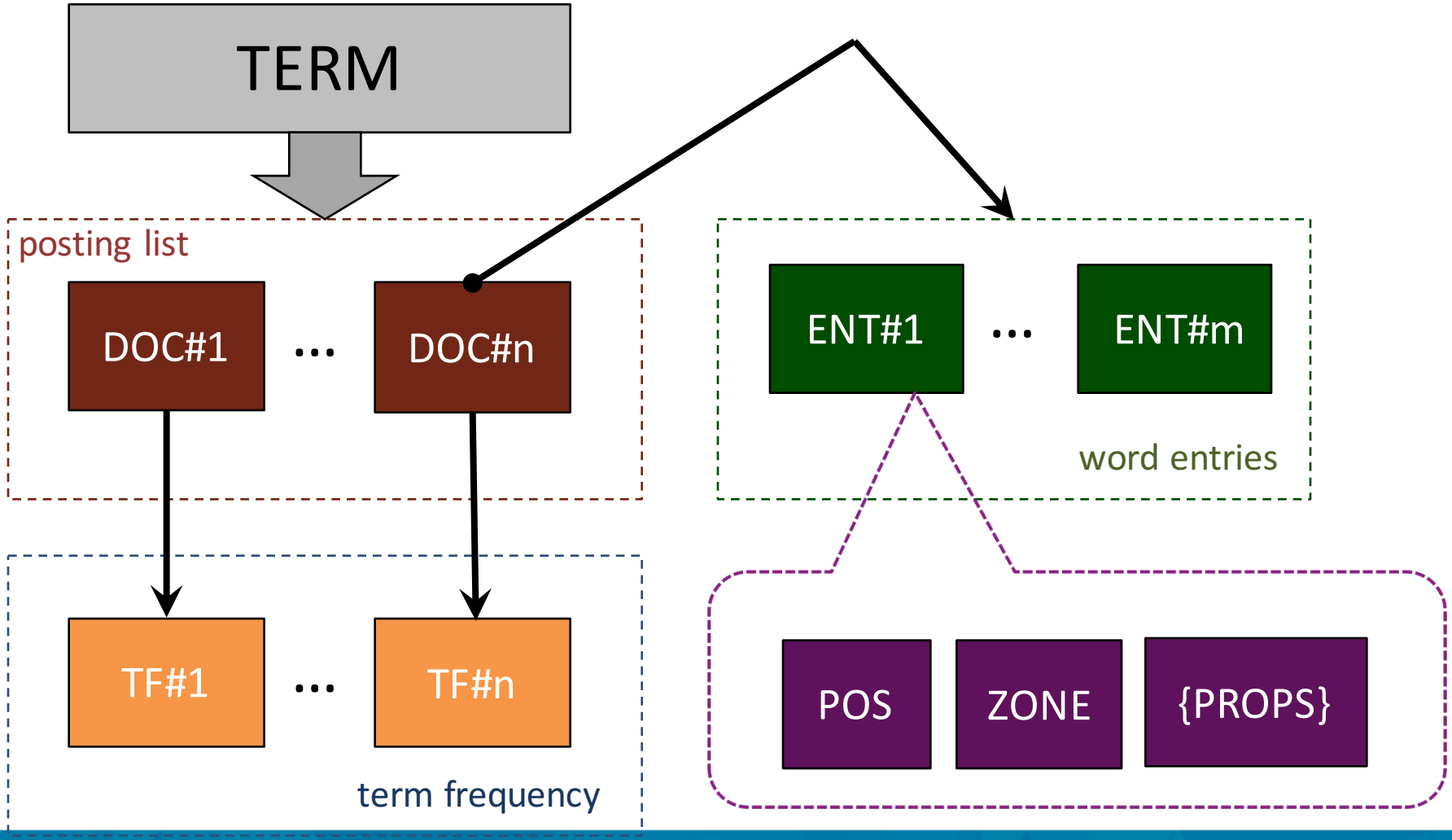
Состав обратного индекса



Состав обратного индекса



Состав обратного индекса



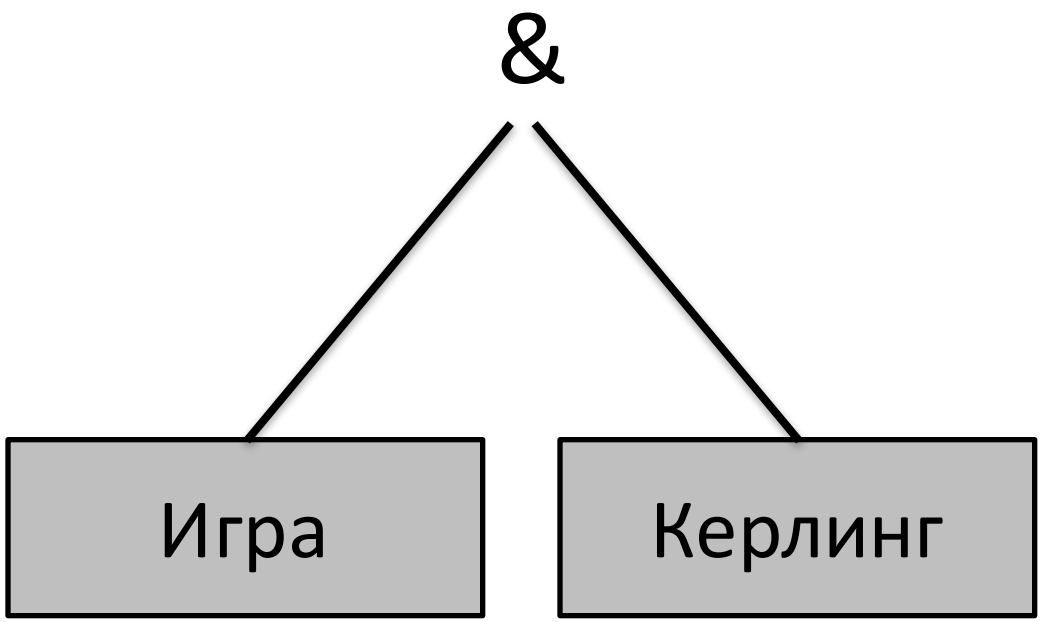
Что нужно в runtime

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

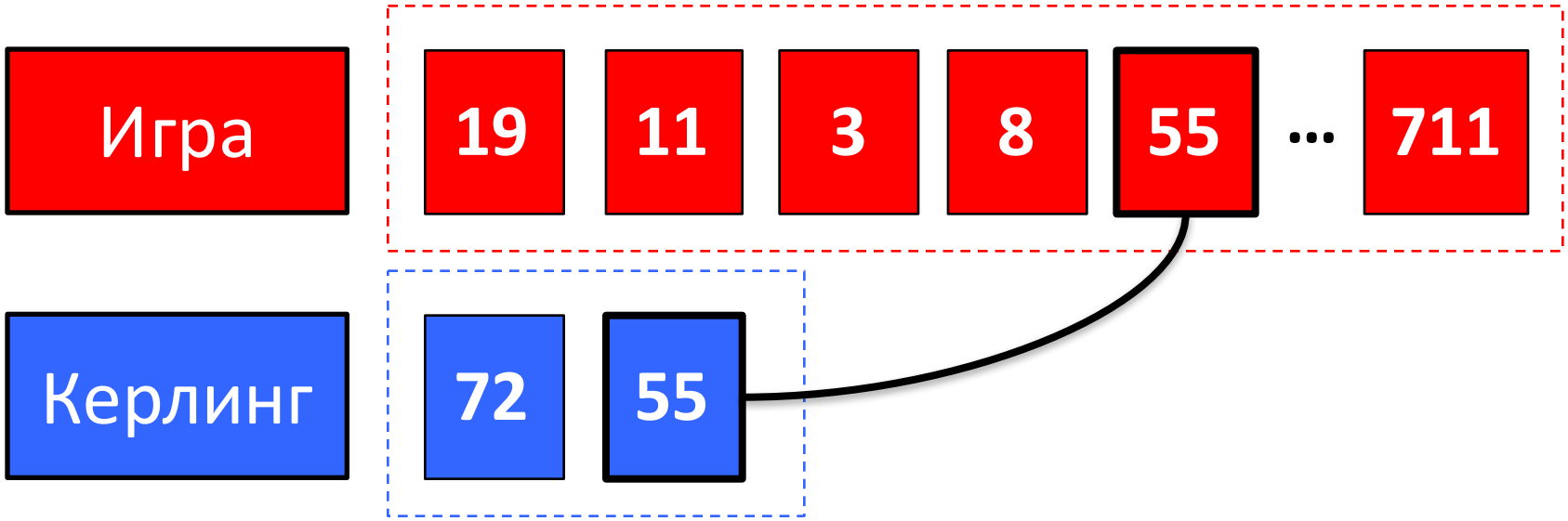
Быстрое пересечение блоков

Типичный запрос

В виде дерева



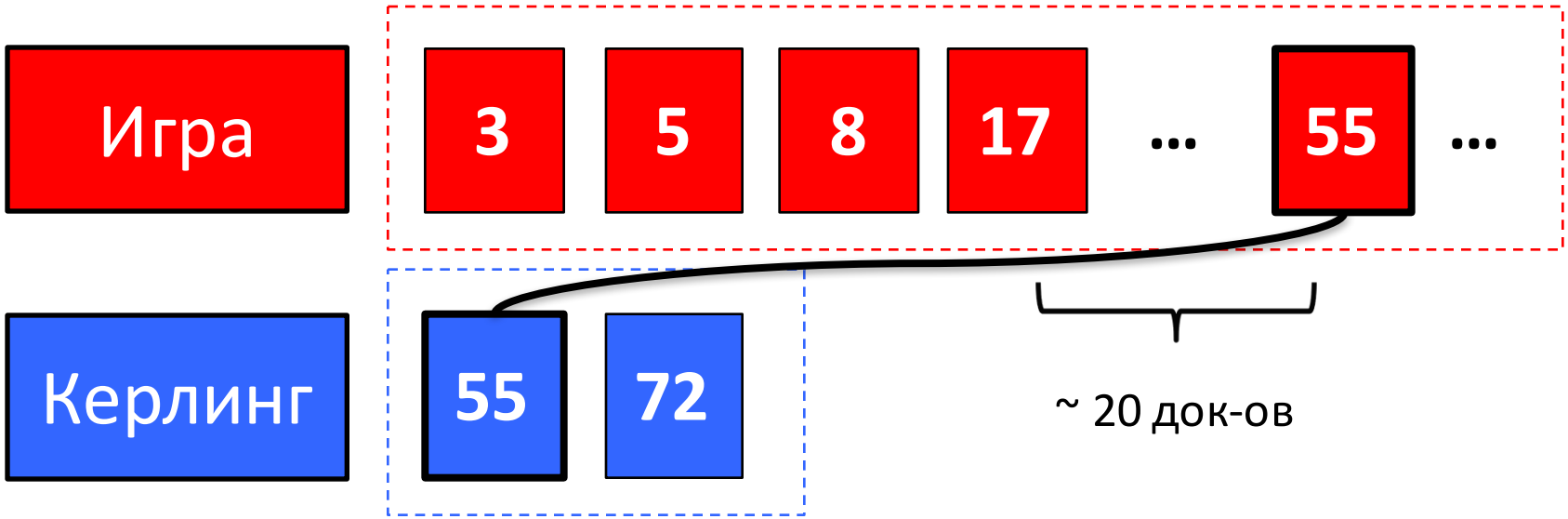
Posting lists



|P(Игра)| = **1000** x |P(Керлинг)|

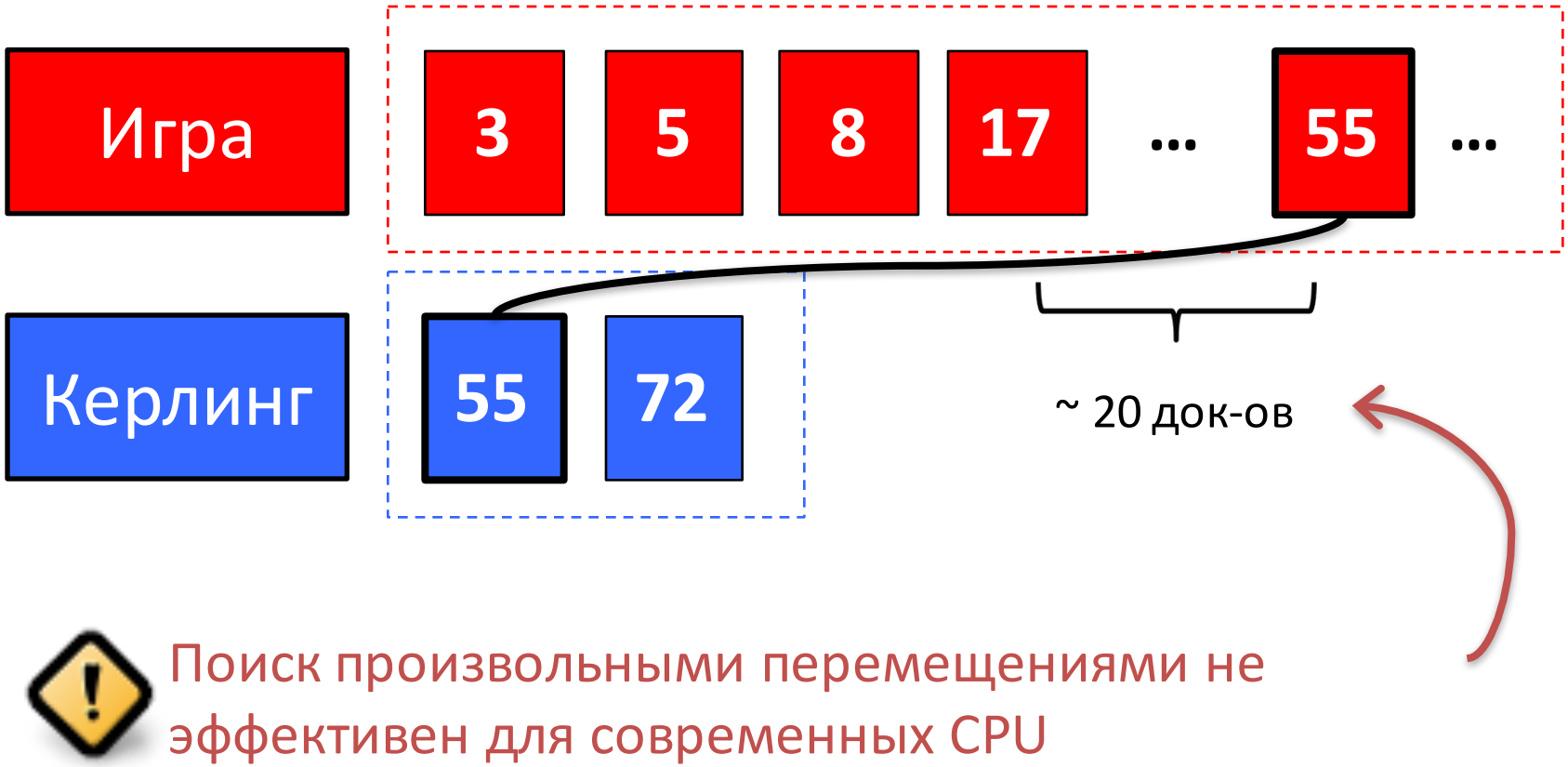
Оптимизация пересечения

1. Отсортируем списки



Оптимизация пересечения

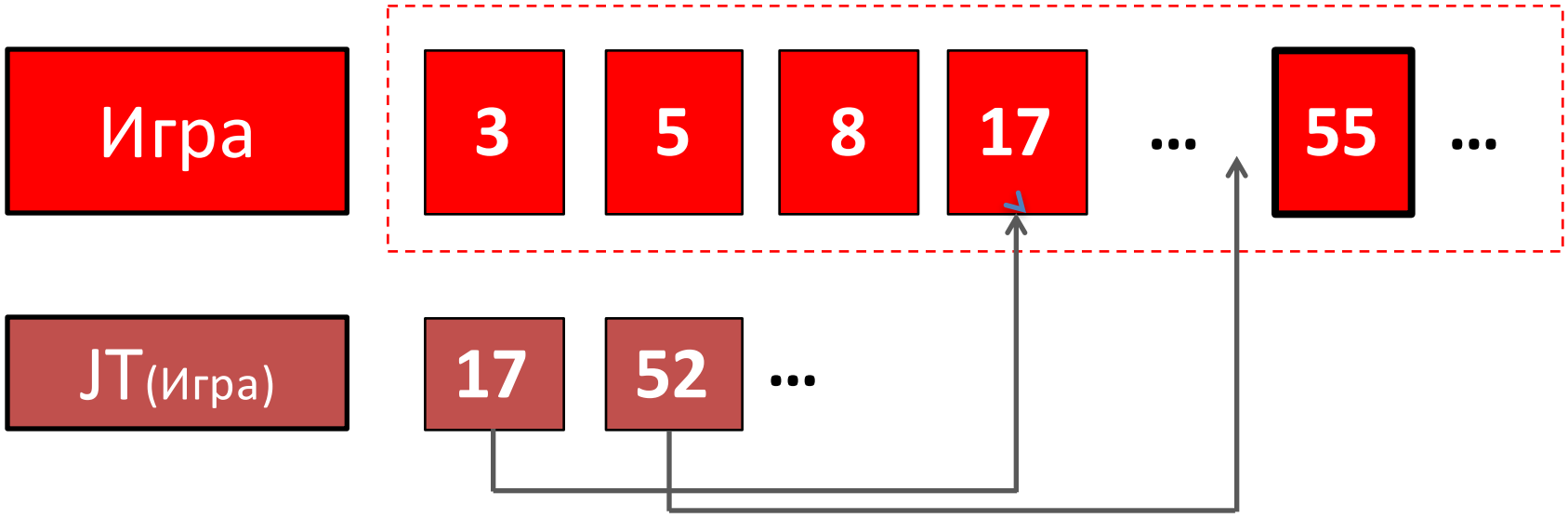
2. Будем искать двоичным поиском по оставшейся части?



Поиск произвольными перемещениями не эффективен для современных CPU

Jump Tables

3. Используем таблицы прыжков

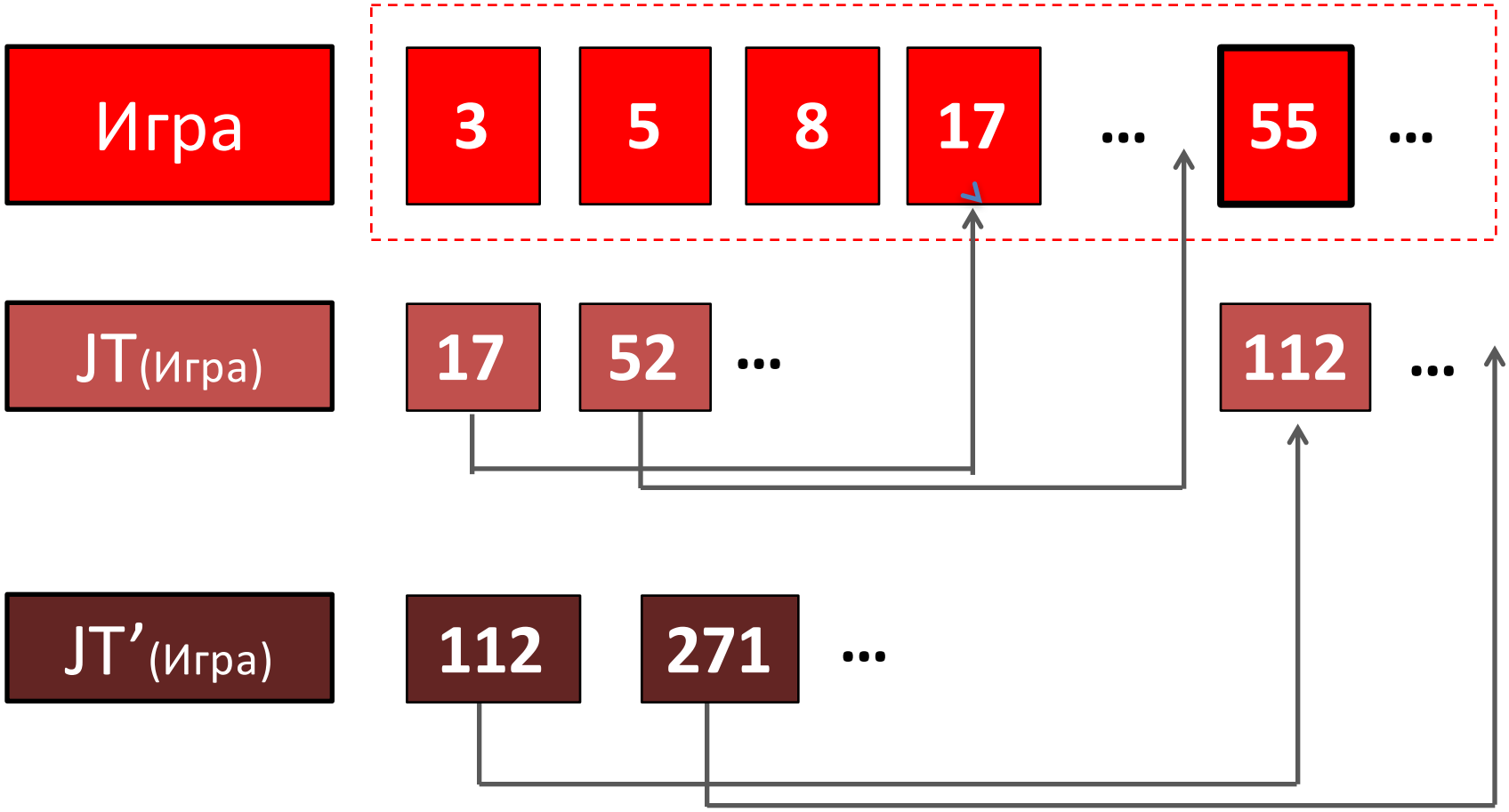


Размещаем смещение идентификатора документа

Что если и этого мало?

More Jump Tables

3. Используем таблицы-индексы дважды – почему нет?



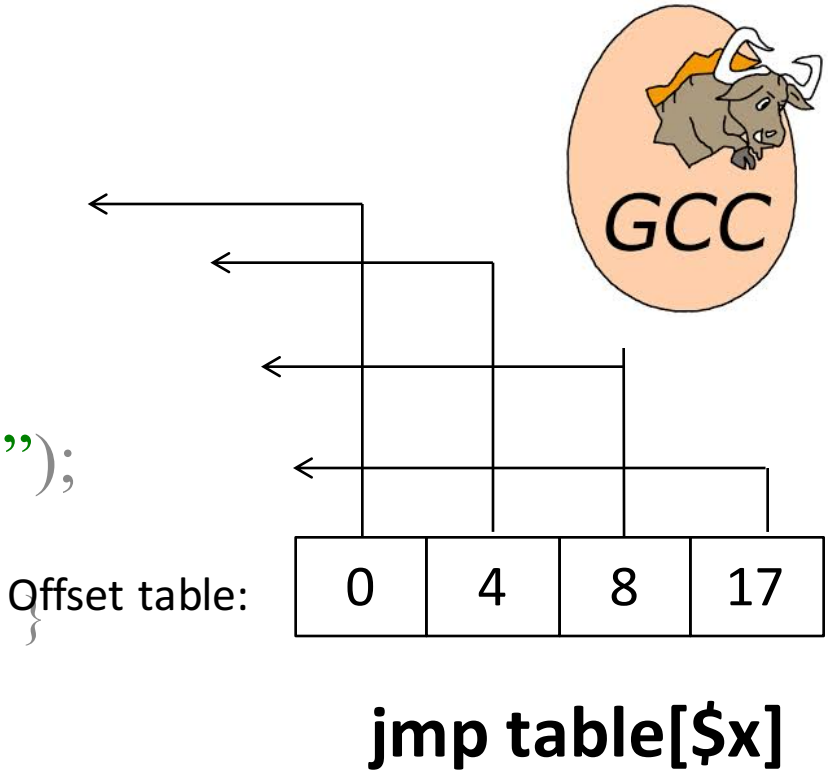
Jump Tables на практике

```
switch(x) {
  case 0: p++; break;
  case 1: p--; break;
  case 2: p = pbegin; break;
  case 3: err(1, “Abnormal code”);
}
```

Jump Tables на практике

```
if (x == 0) p++;
else if (x == 1) p--;
else if (x == 2) p = pbegin;
else if (x == 3) {
  err(1, “Abnormal code”);
}
```

```
switch(x) {
  case 0: p++; break;
  case 1: p--; break;
  case 2: p = pbegin; break;
  case 3: err(1, “Abnormal code”);
}
```



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

- Jpeg/архиватор Попова
- 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

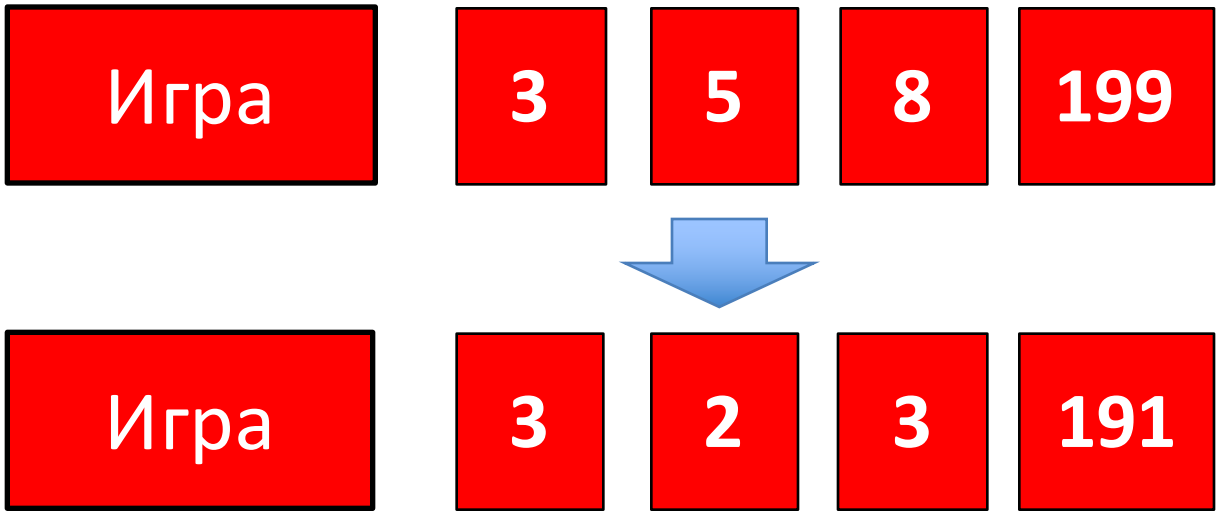
- Координатные блоки – существенная часть обратного индекса
- Будем сжимать каждый постинг
- В булевском индексе – это docID

Цель

- Предположим, мы индексируем 1 МЛН документов
  - можем использовать log2 100,000
  - 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

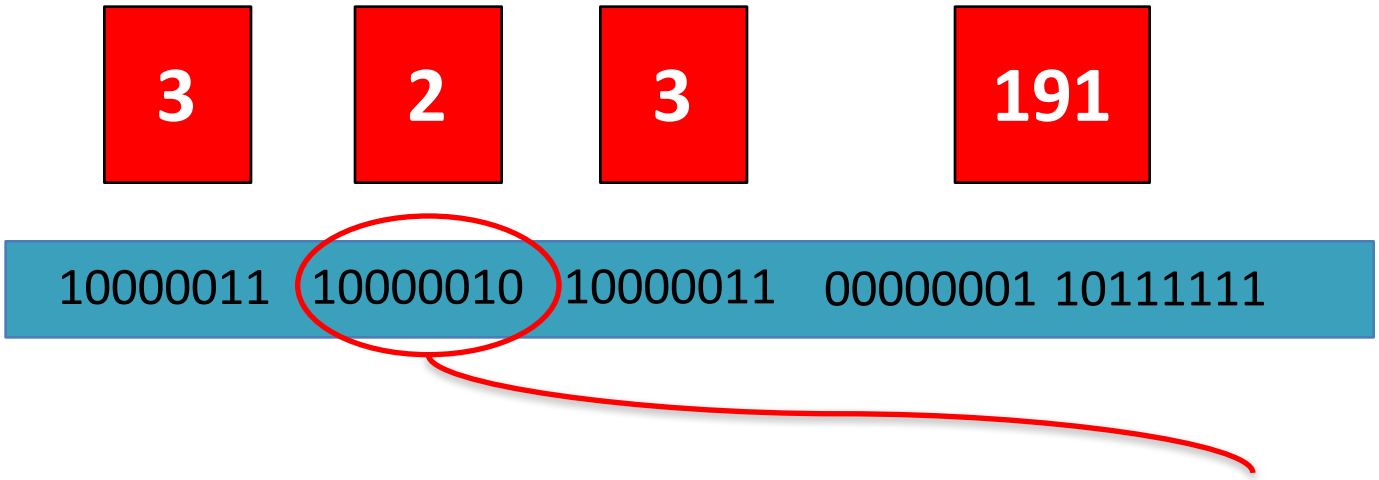
Пример кодирования varbyte

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

Похожая схема

UTF-8

```
$ echo привет! | hexdump –C
```

! = 0x21

\n = 0x0a (line feed)

Прочие символы – 2 байта

От байт к битам

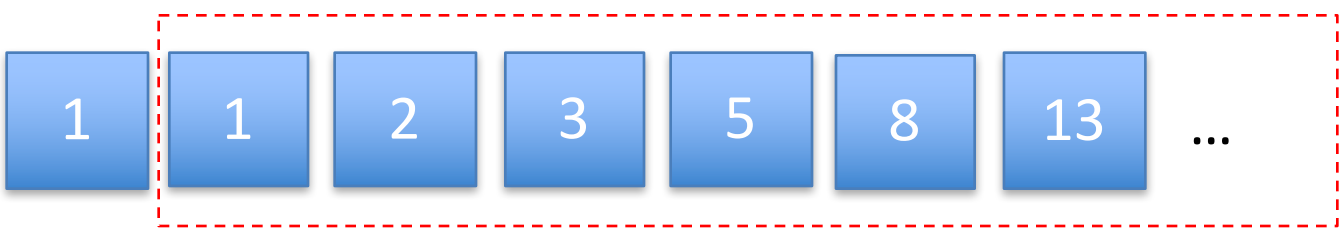
- Кодирование по байтам избыточно для малых промежутков
- Будем использовать битовое кодирование

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

Код Фибоначчи

1 1 2 3 5 8 13 …

Код Фибоначчи



Представим основанием СС:

…	13	8	5	3	2	1	FIB
…	32	16	8	4	2	1	BIN

Код Фибоначчи

- Self-terminating: 11 – недопустимая комбинация
- Алгоритм довольно простой
- К сожалению, кодирование, декодирование не эффективно для CPU

Гамма-код (Elias Gamma)

- Кодирование:
  1. Записываем число в 2ой 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 2ой 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
- Могут использоваться для любого распределения чисел.
- 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.

Промежуточные итоги

- Теперь мы можем создать небольшой индекс для булевского поиска
- 10-15% от размера текста корпуса
- Но мы не хранили координатную информацию
- Т.е. в реальности индексы больше размером
  - Но методы сжатия похожи на рассмотренные.
- Рассмотрим ещё несколько алгоритмов

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

От битов к … байтам

- Низкая скорость распаковки
- Не оптимизировано под CPU
- Границы машинных слов нарушаются
- …
- => Наверняка, не будет реализовано в SSE

- Необходимо нечто среднее: с преимуществом битового кодирования (размер) и байтового (скорость)

Другие коды переменной длинны

- Вместо байта можно использовать другие границы выравнивания: 32, 16, 4 бита.
- Меньшая граница выравнивания позволяет не терять биты на мелких промежутках – полезно, если много мелких промежутков.
- Коды переменной длины:
  - Часто используется
  - Хорошо ложатся на архитектуру ЭВМ.
- Можно так же упаковывать несколько промежутков в одно слово.

Simple9

- Нужен код, выровненный по словам: 32 бита
- Разобьём слово на две части: 4 бита – управление, 28 - данные
- Что можно сохранить в 28-и битах?
  - 1 28-и битное число
  - 2 14-и битных числа
  - 3 9-и битных числа (и теряем 1 бит)
  - 4 7-и битных числа
  - 5 5-и битных чисел (и теряем 3 бита)
  - 7 4-х битных чисел
  - 9 3-х битных чисел (и теряем 1 бит)
  - 14 2-х битных чисел
  - 28 1 битных чисел
- И запишем в 4 управляющих бита схему
  - И ещё могут быть исключения, число больше 28 бит

Свойства Simple9

- Просто упаковать
  - Следующие 28 чисел помещаются в 1 бит?
    - Если нет, то 14 в два бита?
      - И т.д.
- Быстро распаковать (один goto на слово)
- Отлично представим кодогенераторами
- Хорошо сжимает
- Есть варианты, например, Simple16
- … Ожидается аппаратная поддержка

Дополнительные приемы сжатия

Дополнительно к компресии

- Блочная компрессия работает лучше, если похожие документы располагаются рядом
- Удобно использовать вместе с системой поиска дубликатов
- Для веба есть простая эвристика: номера docid должны соответствовать отсортированному в лексикографическом порядке списке URL.

Разделение данных

- Выгодно сжимать разные данные разными методами
- Паттерны доступа разные
- => разбиваем части для локальности доступа

Разделение данных (TF)

- Зачастую одинаковые числа
- Можем использовать RLE:
  - 1 1 1 1 1 => 1x5
- TF нужны реже чем DocIDs =>
  - Можем вынести в отдельную часть индекса

Разделение данных (remap)

- Т.к. используем переменное кодирование, то малый код выгоден
- Можем переназначить номера исходя из частоты
- => Декодирование быстрое, т.к. табличное

До	После
Body (1) FREQ = 6 … Title (5); FREQ = 80	Title (5 -> 1) Body (1 -> 2) …

Когда какой метод использовать

- Posting Lists: Simple9
  - Вхождений много
  - Требуется большая скорость распаковки
- Сами вхождения: Rice | VarByte
  - Вхождений терма часто бывает 1,2
  - Наиболее вероятно что не в начале
- Мелкие числа: Elias
  - Зоны (предварительно упорядочив)

Подведем итог

- Можем быстро объединять списки
- Применяя сжатие значительно уменьшаем размер индекса (-200%+)
- Разделяем индекс на отдельные части для
  - Гибкости
  - Локальности данных

Что осталось за кадром

- Создание словаря индекса
- Индексация больших объемов данных
- Индексация в большом WEB
- Схема поиска большого WEB
- Работа с памятью

Спасибо!

Вопросы?

Домашнее задание

- Есть дамп lenta.ru
  - 1/100
  - Предобработан boilerpipe
- Необходимо
  1. Создать булев индекс (VB, Simple9)
  2. Реализовать jump tables
  3. Реализовать поиск (&, |, !, (,))