О какой производительности речь?

Критерии производительной системы:

- минимальное время отклика
- минимум железа
- разумный минимум кода

О чем мы поговорим

- Как уместить максимум нужного в одну JVM
- Как можно просто разгрузить GC
- Как надо паралелить запрос
- Как не надо паралелить запрос

Эффективность vs Оптимизация

Просто поиск

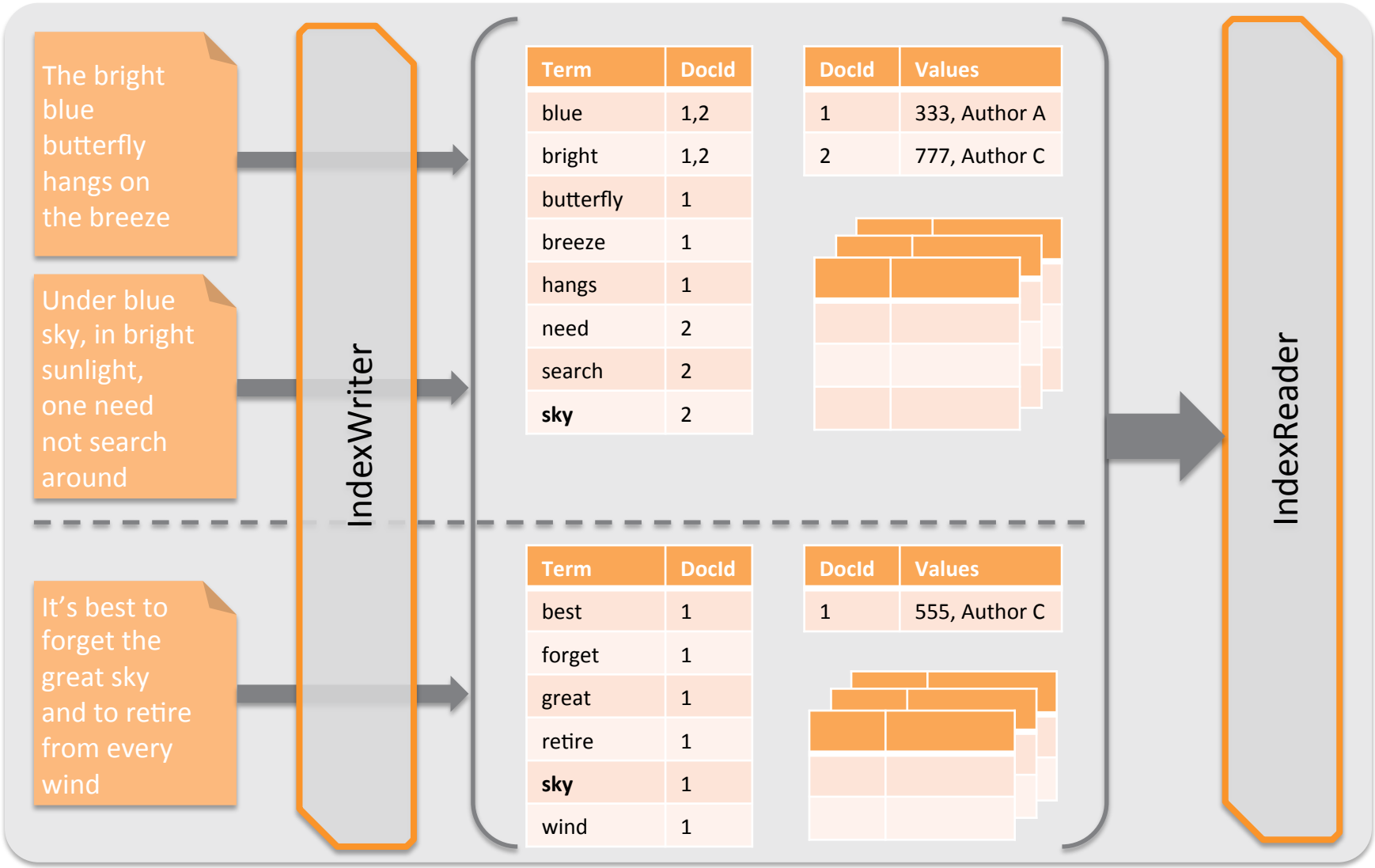
- SELECT * FROM Users WHERE name like ?
- List<User>

Просто поиск пользователей

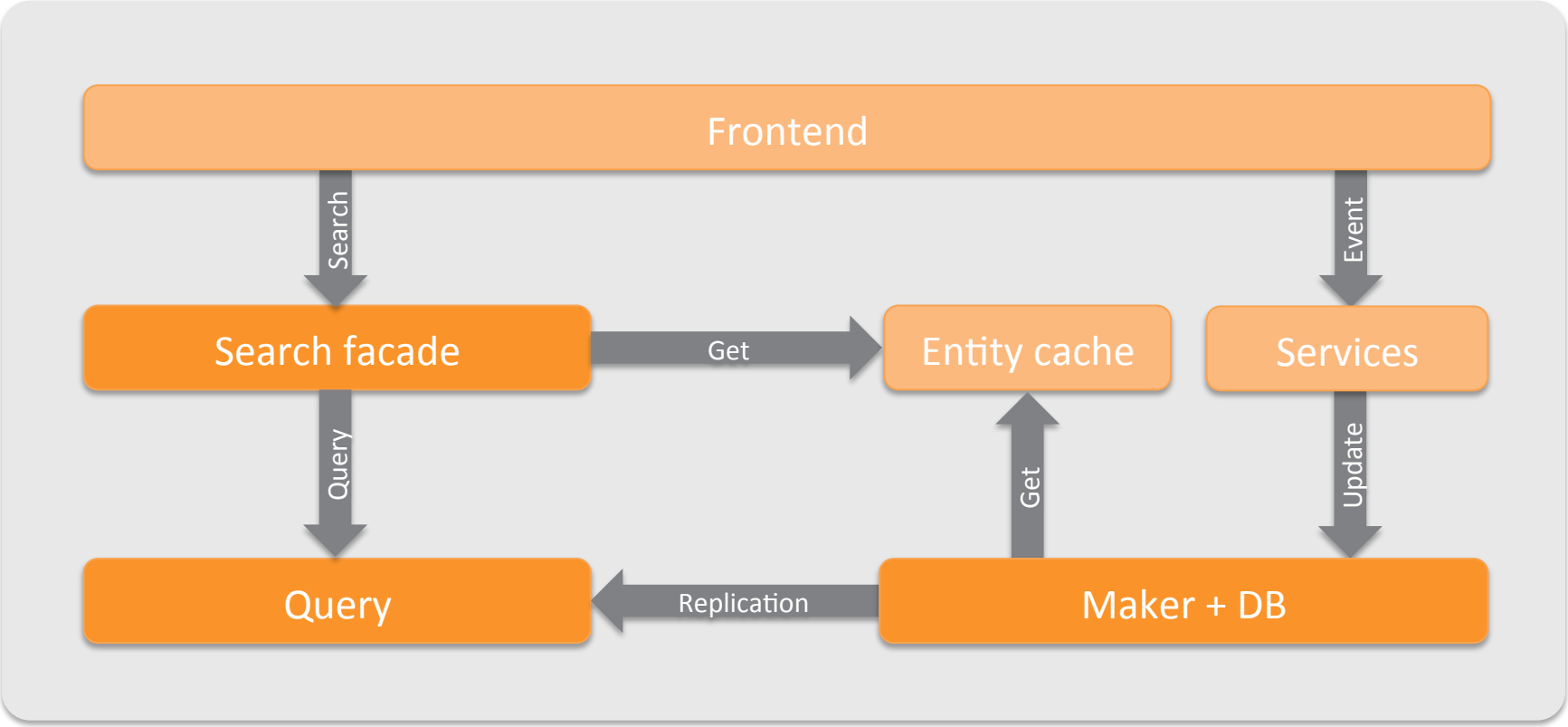
User это десяток полей

- 5 лет назад было 37 000 000
  - 370 000 000 объектов
  - 37M * 512B = 17 GB
- Сегодня 250 000 000
  - 2 500 000 000 объектов
  - 250M * 512 = 128 GB

Как это решает индустрия?



Архитектура



1 500 000 000 объектов

1 500 000 000 объектов в JVM

В одной JVM у нас могут уживаться:

- Индекс на 1 000 000 000 объектов
- Кэш запросов на 500 000 000 объектов
- Словарь исправлений на 10 000 000 объектов

JVM:

- Xmx=16G
- stop the world < 0.02 sec раз в 3 секунды

Сколько объектов видит GC?

Папка 1: своя коллекция

```
class Match {
  int doc;
  int rank;
  long id;
  int data;
}

PriorityQueue<Match>(10000)
```

```
class MatchedDocs {
  int size;
  int doc[10000];
  int rank[10000];
  int data[10000];
  long id[10000];

  int addDoc(...)
  int getDoc(int pos);
  .....
}
```

Папка 2: упакуй их полностью

```
class IndexTerm { // 12 bytes
  String term; // 16 + (12 + 4 + ? * 2) + 4 + 4 + 4) bytes
  int freq; // 4 bytes
  long pointer; // 8 bytes
} 68 + term.len * 2
```

prefixLen:VInt	suffLen:VInt	chars:char	freq:VInt	ptr:long

```
win    = 0;3;win  ;1024;12345678 = 18
wine   = 3;1;e    ; 120;23456789 = 13
wonder = 1;5;onder;  42;34567890 = 21
                                 = 12 + 4 + 52 = 68 bytes
```

Папка 3: небезопасная

Unsafe:

- Быстрее массивов на 5-10%
- Риск SegFault
- Необходимость управлять ресурсами

Где он хорош:

- Кэши объектов
- Коллекции которые не помещаются в 2GB

Как сделать хип безлимитным

- Классы с минимумом объектов
- Кодируйте данные уменьшая их объем
- Архивируйте то что не нужно всем

Стены мусора

Когда GC не справляется

Ключи которые есть на любой нашей JVM:

- -verbose:gc
- -XX:+PrintGCDetails
- -XX:+PrintGCTimeStamps
- -XX:+PrintGCApplicationStoppedTime

Сколько мусора оставляет запрос?

Делая систему важно знать ответы на вопросы:

- Сколько объектов создает запрос?
- Сколько занимают созданные объекты?

Вы всегда можете сравнить вашу оценку с:

- Очищенная память / выполненые запросы

Из архива не вынимать!

prefixLen:VInt	suffLen:VInt	chars:char	freq:VInt

```
while (it.next()) {
  String itTerm = it.termString();
  if (itTerm.startsWith(qryTerm)) {
     queue.push(new Prefix(itTerm, it.termFreq()));
  }
}
```

```
while (it.next()) {
  if (it.startsWith(prefixTerm)) {
    queue.push(it);
  }
}
queue.toList();
```

Фантики

- Нужен long
- Получали минимум 5 объектов:

```
class Document { // 12
  List<Field> fields; // 12 + 4 + (16 + 40)
}

class Field { // 12
  String name; // String.intern()
  Object data; // 16 + ?
}
```

Лучше без фантиков

```
docs = new MatchedDocs();
parser = new StoredFieldParser() {
  void fieldRaw(byte[] buf, int off, int len) {
    docs.add(query.doc(), Utils.parseLong(buf, off, len);
  }
}

while(query.next()) {
  index.read(parser, query.doc());
}
```

Ручное управление

Мы вызываем System.gc() перед:

- входом в ротацию
- операциями которым нужно много памяти

Это лучше работает когда выставлен ключ:

- -XX:+ExplicitGCInvokesConcurrent

Поймать с поличным!

Поймать с поличным помогают ключики JVM:

- -XX:+HeapDumpBeforeFullGC
- -XX:+HeapDumpOnOutOfMemoryError

Дамп открываем в Eclipse Memory Analyzer

- важно отключить удаление мертвых объектов

Как жить чище

- Чем чаще что-то выполняется, тем меньше лишнего должно создаваться
- Используйте примитивы
- Переиспользуйте объекты

Перпендикулярности

Перпендикулярности

2ое работают, 6 наблюдают

- Чтение индекса меняет его состояние
- Индекс редко состоит из одного сегмента
- Захватывать можно по сегментам

Пул лопат и ям

- Открытие индекса дорогой процесс
- Даже для RamDrive IO не бесплатен
- Пулы не решают проблему дублирования

Яму видел? Копай такую же!

```
idx = provider.cloneIndex();
qry = prepare(idx, str);
docs = idx.search(qry);
ids = idx.loadIds(docs);
```

```
idx = provider.getIndex();
try {
  qry = prepare(idx, str);
  docs = idx.search(qry);
  ids = idx.loadIds(docs);
} finally {
  provider.release(idx);
}
```

Unsafe быстрей!

Сложности

- Управление ресурсами
- Сложнее код
- Шанс выиграть Segfault

Экономика

- byte[] серверов 120
- Unsafe серверов 110

Пицца параллельности

Пицца параллельности

1 пиццей зал не накормишь

Вы покушать или закусить?

Query server

- Статистика
  - 200 запросов в сек.
  - 40 мс на 1 запрос
  - 200 * 40 / 1000 = 8
- Нагрузка на CPU
  - 8 из 24 доступных

Façade server

- Статистика
  - 280 запросов в сек.
  - 55 мс на 1 запрос
  - 280 * 55 / 1000 = 15
- Нагрузка на CPU
  - 6 из 24 доступных

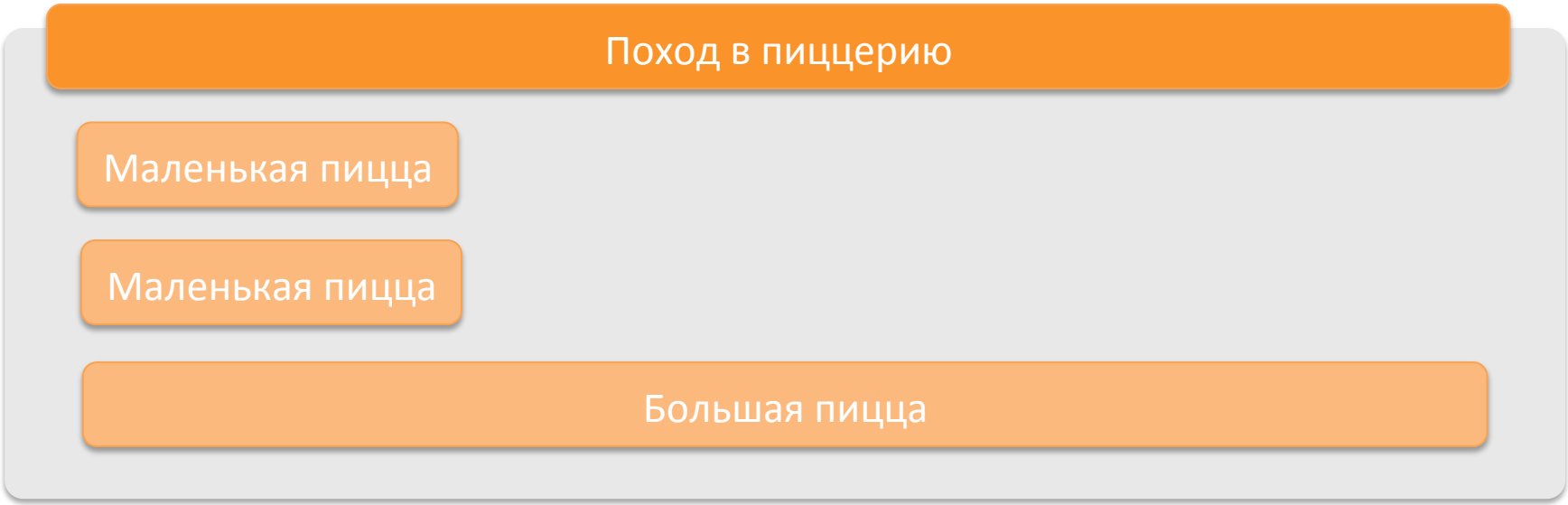
Проблемы с доставкой

Как делить пиццу?

Учитывайте:

- сколько доступно процессоров
- накладные расходы
- сколько работы реально уйдет другим
- стоимость большого числа потоков

Друзей не бросают



Итоги

Мерьте и контролируйте все!

- Сколько занимает запрос
- Сколько занимает поход в другой сервис/базу
- Сколько занимает ваш алгоритм
- Сколько времени вас держал GC
- …

Берите только нужное!

- Данные можно оптимизировать
- Пишите простой код *^%#!
- Знайте используемые библиотеки
- Захватывайте ресурсы там где они нужны

Можно узнать больше!

Odnoklassniki.ru

http://v.ok.ru

Интеграция с Odnoklassniki.ru

http://connect.ok.ru

one-nio

slideshare.net/m0nstermind/presentations
github.com/odnoklassniki/one-nio

Cassandra

github.com/odnoklassniki/apache-cassandra
cassandra.apache.org

Алексей Шевчук

aleksey.shevchuk@odnoklassniki.ru
ok.ru/mrSearch