Hash & Cache

Курс «Базы данных»

Цесько Вадим Александрович
http://incubos.org
@incubos

Computer Science Center

16 сентября 2013 г.

Содержание

Коллекции: сложность операций

`ArrayList`:

- `get()`: $O(1)$
- `insert()`: $O(n)$
- `delete()`: $O(n)$
- `append()`: $O(1)$ / $O(n)$

`LinkedList`:

- `get()`: $O(n)$
- `insert()`: $O(n)$
- `delete()`: $O(n)$
- `append()`: $O(1)$

Как ещё быстрее?

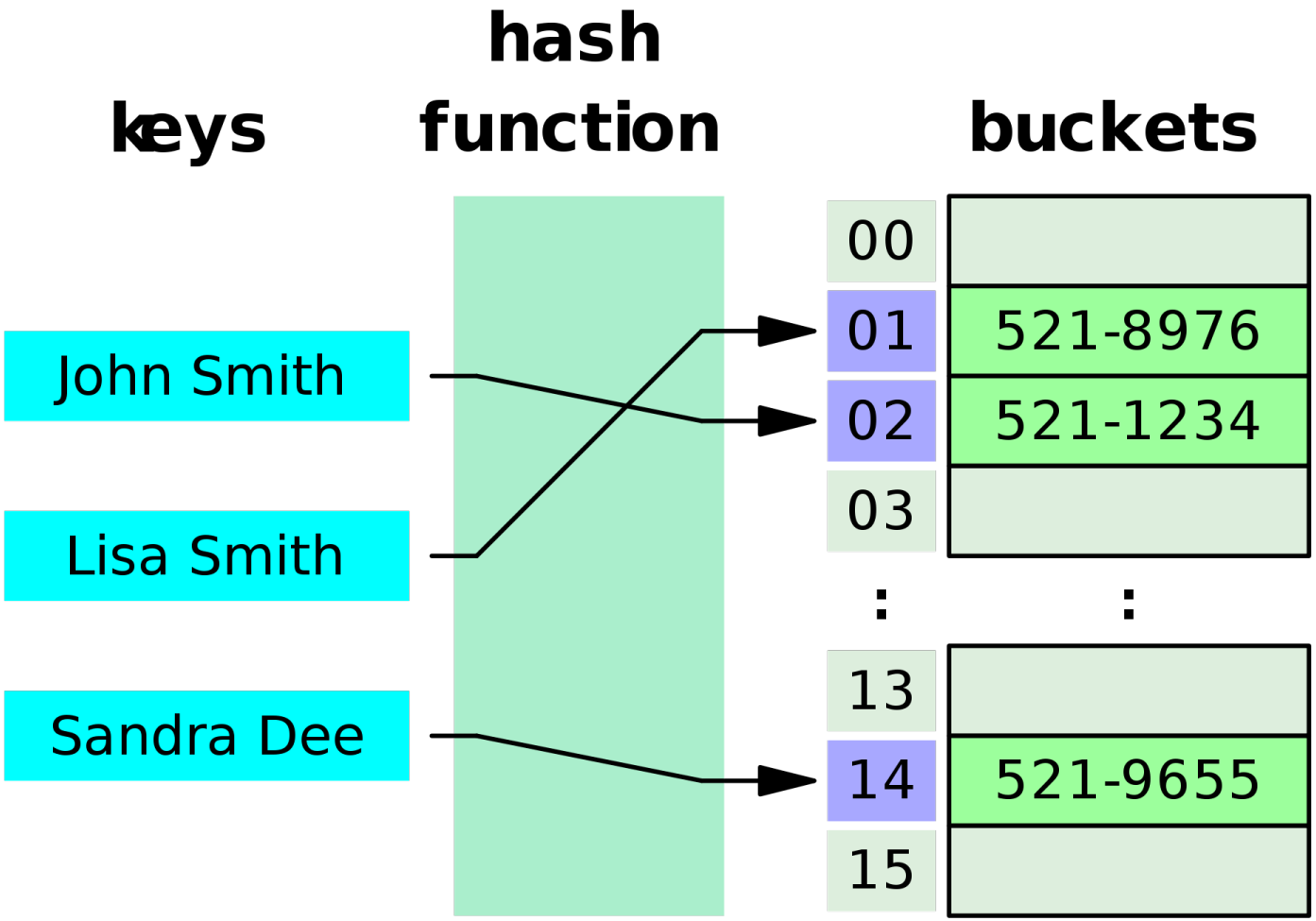
Hash table

Ассоциативный массив, отображающий ключи в значения[a]:

1. С помощью **хэш-функции** от ключа находим индекс в массиве bucket'ов
2. Перебираем ключи в найденном bucket'е до совпадения
3. Извлекаем значение

[a] http://en.wikipedia.org/wiki/Hash_table

Пример


keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655

Hash table: сложность операций

- `get()`: $O(1)$
- `insert()`: $O(1)$
- `delete()`: $O(1)$
- `append()`: $O(1)$

Допущения

- При **равномерной** функции хэширования
- Без учёта **ребалансировки**

Подводные камни

- Выбор функции хэширования
- Разрешение коллизий:
  - Separate chaining — лишняя память, CPU cache misses
  - Open addressing — элементов не больше 70% размера массива, CPU cache pollution
- Resize & rehash
- Concurrency

Материалы

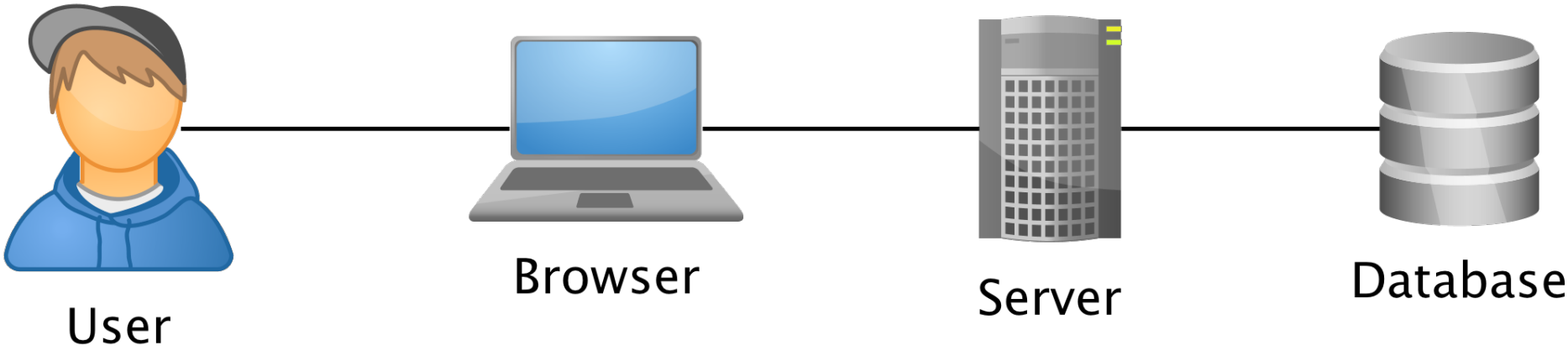
- `java.util.HashMap`
- `java.util.concurrent.ConcurrentHashMap`
- `http://en.wikipedia.org/wiki/Hash_table`
- Chris Okasaki. *Purely Functional Data Structures*[1]

[1] `http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf`

Классические архитектуры Web-приложений

- Как взаимодействуют с хранилищами
- Какие проблемы пытаются решить
- Каким образом

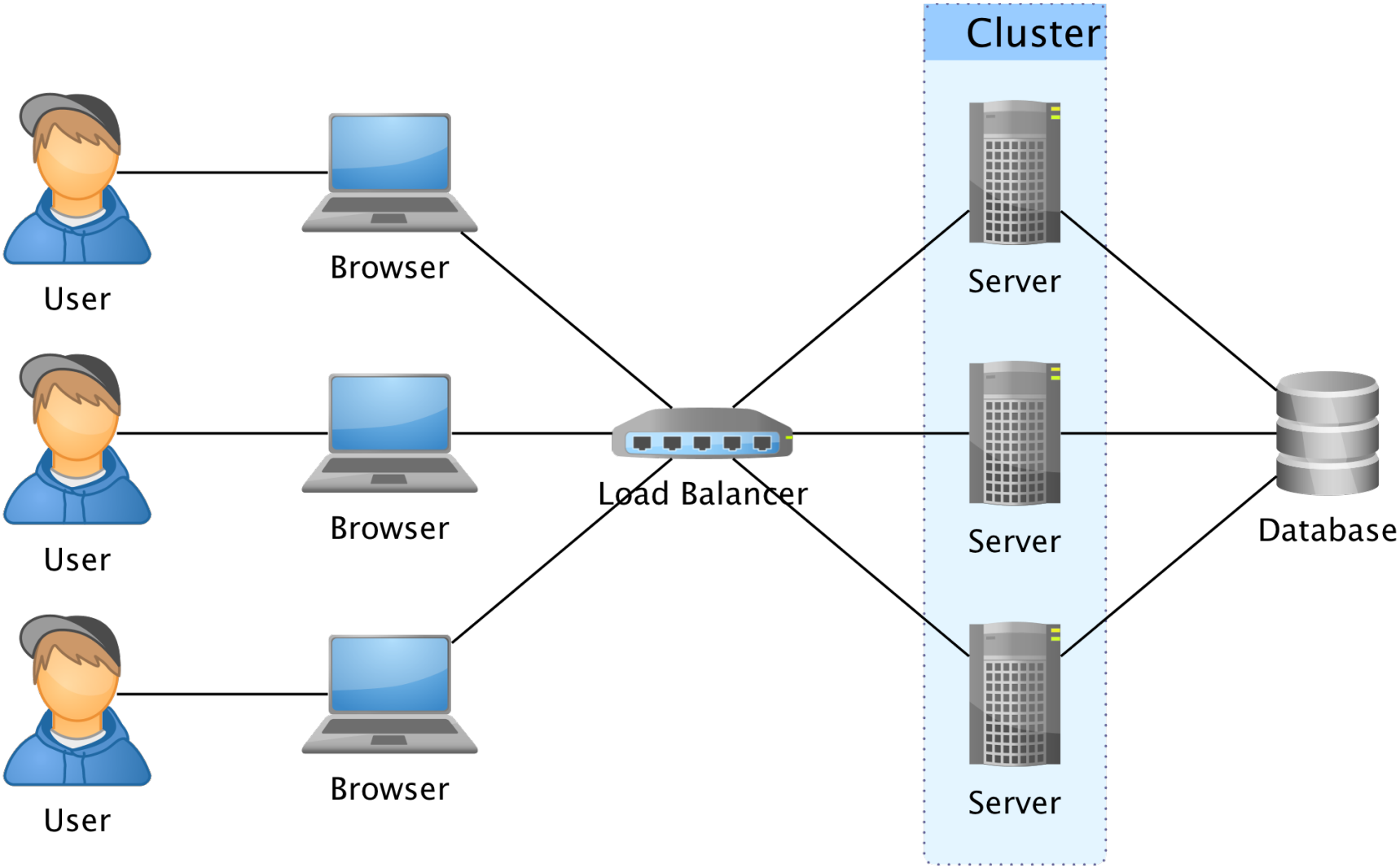
Трёхзвенный клиент-сервер



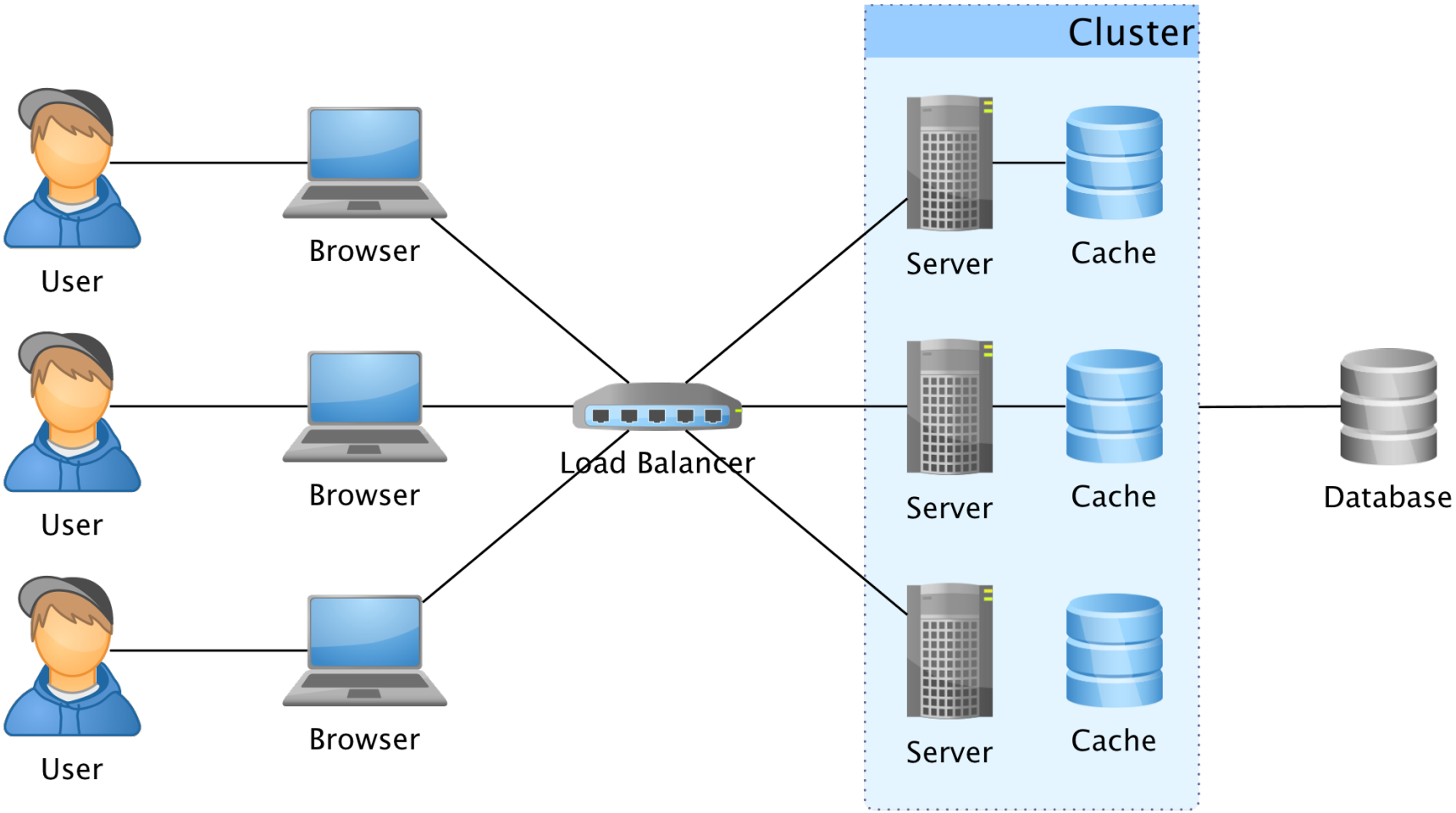
Зачем нужен Server:

- Преобразование форматов
- Логика: проверить, посчитать, ...
- Гибкость

Нагрузка растёт



Нагрузка ещё больше



Что такое cache

Cache

Реализация in-memory concurrent hash table с некоторыми особенностями.

```
1   def process(request: Request): Response = {
2       implicit val timeout = Timeout(50 milliseconds)
3
4       cache.get(request)
5           .getOrElse({
6           val response = buildResponse(request)
7           cache.put(request, response)
8           response
9       })
10  }
```

Анализ cache per server

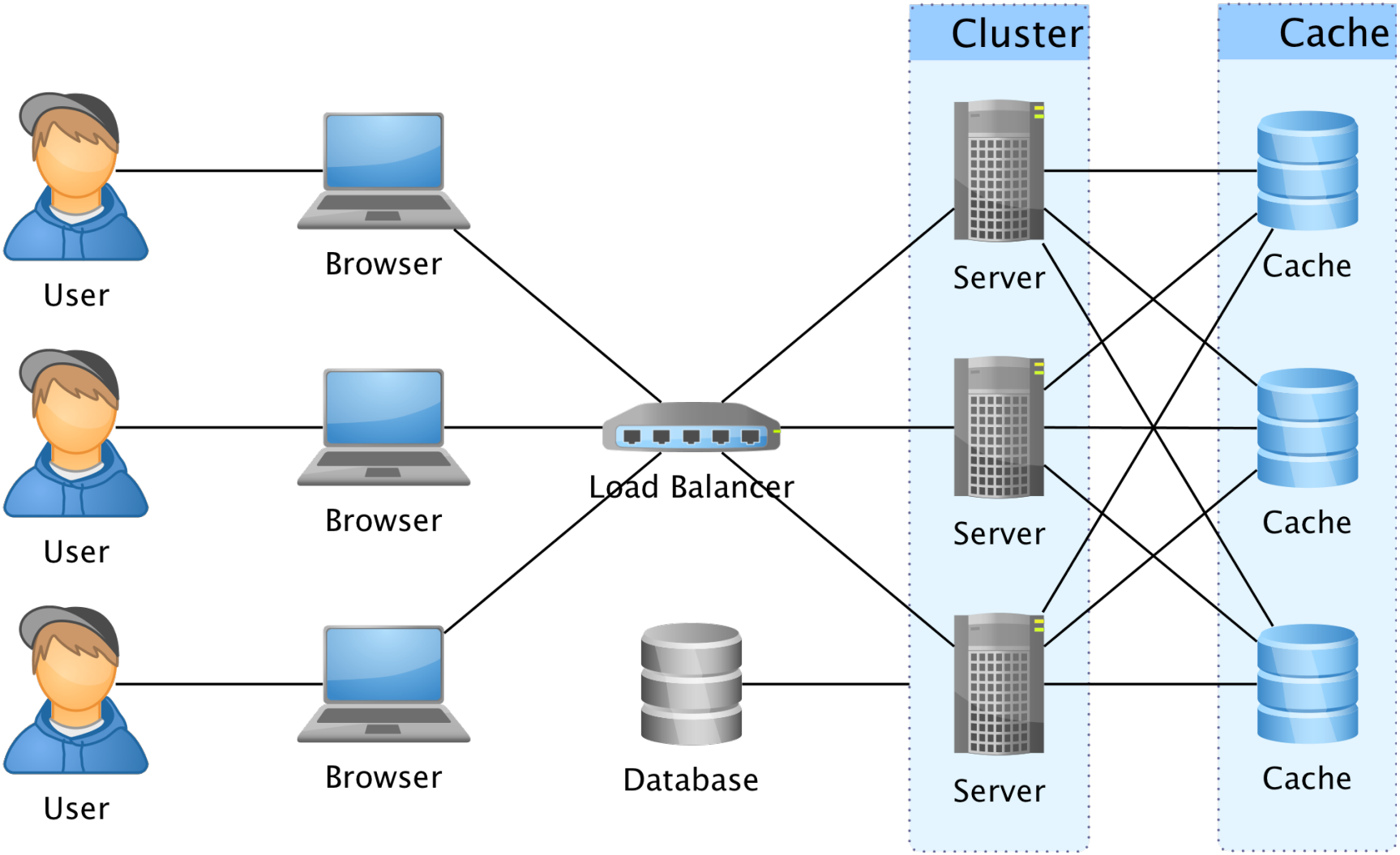
Достоинства

- Снизили нагрузку на CPU и БД (если нормальный cache hit ratio)

Недостатки

- В каждом cache одинаковые значения
- «Лишние» вычисления (нагрузка на CPU)
- Ограниченный размер каждого cache
- Получили спектр проблем с апгрейдом системы
- Нужны политики «протухания» и вытеснения
- Нужно «прогревать» кэш или перезапускать сервера постепенно

Distributed cache



Первый подход к снаряду

```
1   class Cache {
2       val servers =
3           Vector("cache01.yandex.net", "cache02.yandex.net")
4
5       protected def lookup(key: AnyVal): String =
6           servers(key.hashCode() % servers.length)
7
8       def get(key: AnyVal) =
9           remoteGet(lookup(key))
10
11      def put(key: AnyVal, value: Array[Byte]) =
12          remotePut(lookup(key), value)
13  }
```

Анализ distributed cache

Достоинства

- Считаем всё (*примерно*) один раз
- Больше суммарный объём
- Больше влезет значений

Недостатки

- Остался спектр проблем с апгрейдом системы
- Нужны политики «протухания» (TTL) и вытеснения (LRU)
- Нужно «прогревать» кэш

Memcached

- `http://www.memcached.org/`
- Разработан для LiveJournal в 2002
- Используется в YouTube, Flickr, Reddit, Facebook, Orange, Twitter, Tumblr, Wikipedia, Yandex
- Входит в Google App Engine, Windows Azure, Amazon Web Services
- In-memory
- Ключ — строка (до 250 байт)
- Многопоточный + `libevent`
- LRU + TTL (up to 30 days)
- Простой текстовый и бинарный протокол: `set`, `add`, `replace`, `append`, `prepend`, `get`, `delete`, ...

Проблема: Изменение кластера memcached

Причины

- Список серверов в клиенте
- Количество серверов в hash-функции

Решения

- Сервис-proxy перед memcached
- Consistent hashing

Сервис-proxy

Функции:

- Слушает клиентский порт
- Хэширует ключ и проксирует запрос
- Держит соединения ко всем экземплярам cache

Недостатки

- +1 сетевой hop
- Весь трафик через одну точку
- SPOF

Consistent hashing

Определение

When a hash table is resized and consistent hashing is used, only K/n keys need to be remapped on average, where K is the number of keys, and n is the number of slots[a].

[a] http://en.wikipedia.org/wiki/Consistent_hashing

Consistent hashing: своими руками

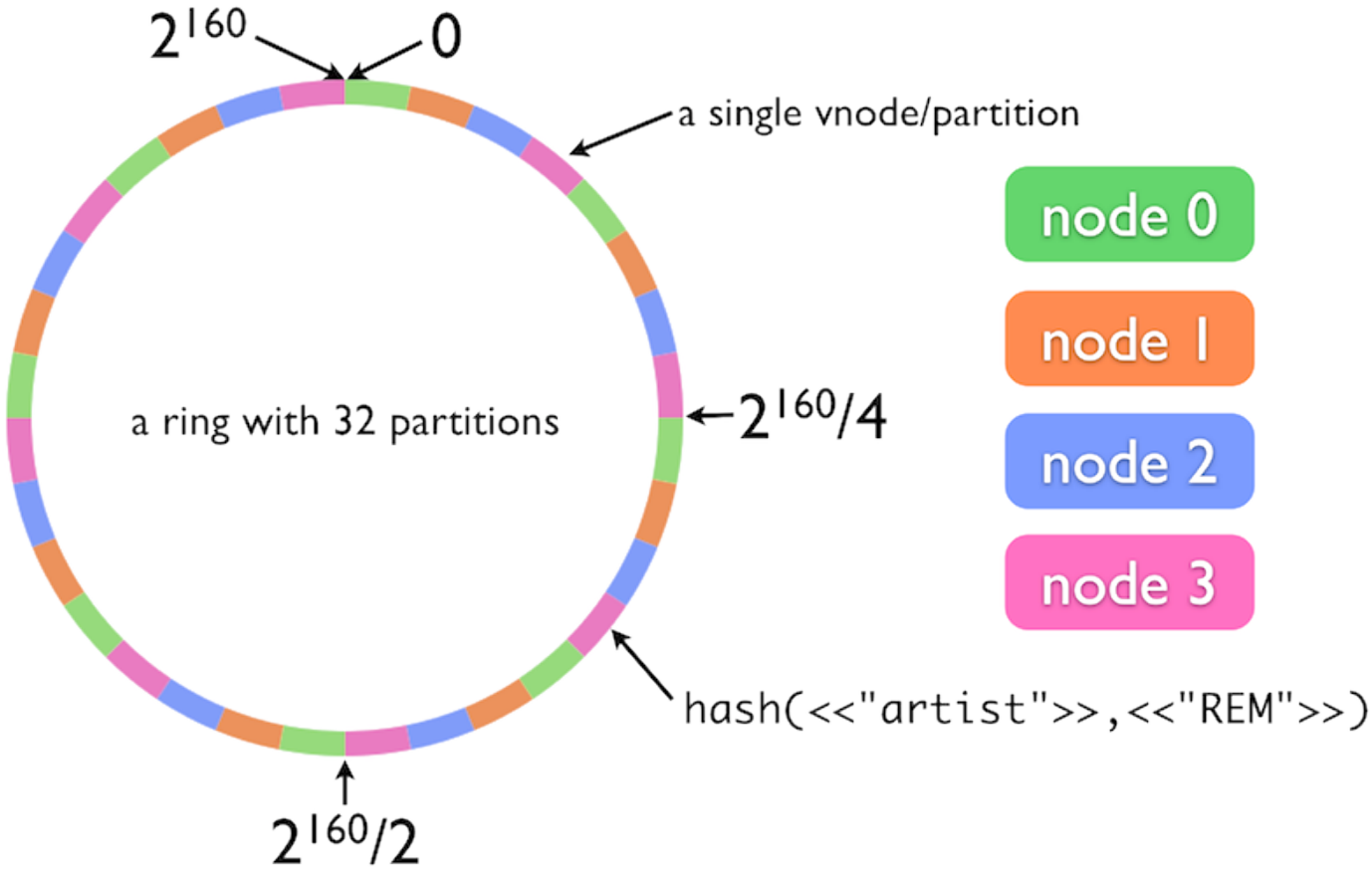
Будем отображать ключи в слоты, а затем слоты на машины:

```
1    val slots = Vector(100, 200, 300, 400)
2
3    def slot(key: AnyVal): Int = {
4        val hash = key % slots.last
5        slots.indexWhere(hash < _)
6    }
```

Если переполняется слот 200-299, то подвигаем:

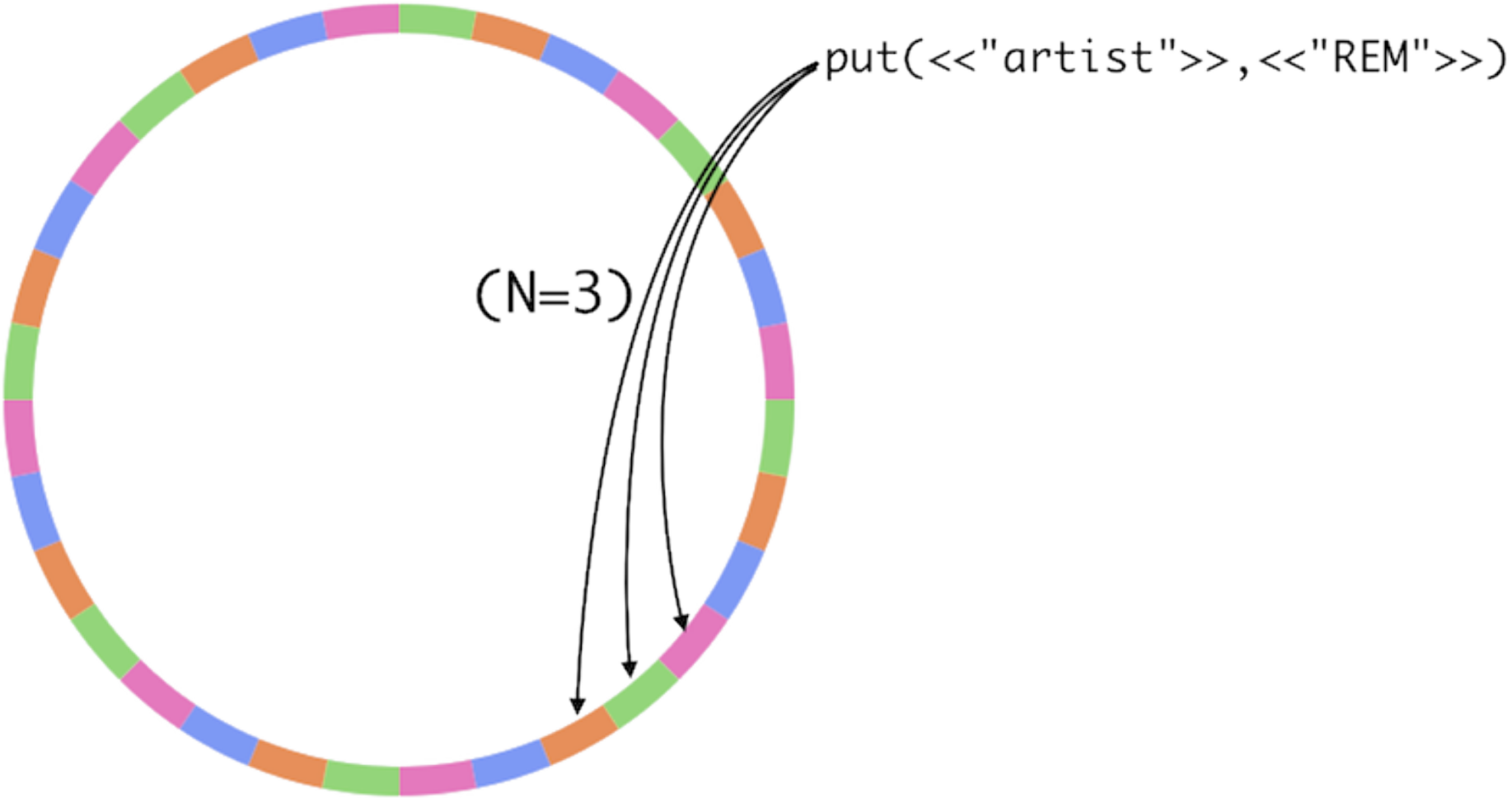
```
1    val slots = Vector(100, 200, 275, 325, 400)
```

Consistent hashing: по-взрослому



http://docs.basho.com/riak/1.1.4/references/appendices/concepts/#Clustering

Consistent hashing: репликация



Проблема: Перезапуск кластера

Причины

In-memory $\Rightarrow$ пустой кэш $\Rightarrow$ БД «ложится»

Решения

- «Разогрев»
- Persistent storage

Redis: Persistent Key-Value Storage

- `http://www.redis.io/`
- Наиболее популярный Key-Value Storage[2]
- Используется в GitHub, Disqus, Pinterest, Stackoverlow, Flickr, Blizzard, Instagram, Twitter
- Key-Value, but Value: `String`, `List`, `Set`, `SortedSet`, `Hash`
- In-memory and/or Snapshots and/or Commit Logs
- Single-threaded by design + Master-slave
- Опционально LRU и/или TTL
- Очень богатый API[3]

[2] `http://db-engines.com/en/ranking/key-value+store`

[3] `http://redis.io/commands`

Вопросы?

- http://incubos.org/contacts/
- Общие вопросы — в Twitter: @incubos
- Вопросы по лекциям — в комментариях: http://incubos.org/blog/
- Частные вопросы — в почту vadim.tsesko@gmail.com