Введение. Key-Value.

«Использование баз данных»

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

ТЕХНОПОЛИС

26 апреля 2017 г.

О лекторе

- http://kspt.icc.spbstu.ru/people/tsesko/
- http://digiteklabs.ru
- https://yandex.ru
- https://compscicenter.ru/teachers/12/
- https://v.ok.ru

Опрос

- Какие реляционные СУБД приходилось использовать?
- Что вызывает сложности при использовании реляционных СУБД?
- Каковы ограничения реляционных СУБД?
- С какими NoSQL СУБД знакомы?
- Что хотели бы узнать про NoSQL?

Формат

- По средам с 18:30 до 21:45
- 3 лекции
- 3 семинара/контрольных занятий
- Групповые проекты

Проект

- Делимся на группы
- Выбираем тему
- Собираем требования
- Проектируем
- Прототипируем
- Представляем результаты
- Итеративно

Материалы

- https://polis.mail.ru/curriculum/program/discipline/35/
- QCon, Devoxx, Strange Loop, etc.
- Ссылки на книги, статьи и блоги
- https://github.com/papers-we-love/papers-we-love
- http://highscalability.com/start-here/

Цели

- Принципы построения систем хранения (и обработки) данных
- Проблемы и варианты их решения
- Промышленные примеры реализации принципов
- Навыки оценки решений

О чём мы будем говорить

- Классификации баз данных
- Web Scale: Scalability, Availability, CAP
- SQL vs NoSQL
- Транзакции
- Производительность
- Примеры баз данных
- Проектирование систем хранения и обработки

О чём мы НЕ будем говорить

- Реляционная модель данных
- Теория нормальных форм
- SQL

База данных

A database

is an organized collection of data[a].

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

- Телефонная книга
- Книжная библиотека
- Коллекция фотографий
- Товарный чек
- Социальная сеть
- Файловая система
- Географическая карта
- ...

Операции над данными

CRUD

Create, read, update, delete[a].

[a] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

Например

Operation	SQL	HTTP
Create	INSERT	POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE

Бумажная телефонная книга

- Create
  - Открыть страницу по первой букве фамилии
  - Вписать в конец страницы ФИО и телефон
- Read
  - Открыть страницу по первой букве фамилии
  - Просматривать записи по порядку до совпадения
- Update
  - Открыть страницу по первой букве фамилии
  - Просматривать записи по порядку до совпадения
  - Зачеркнуть ФИО и телефон
  - Вписать в конец страницы ФИО и телефон
- Delete
  - Открыть страницу по первой букве фамилии
  - Просматривать записи по порядку до совпадения
  - Зачеркнуть ФИО и телефон

Про «скорость» телефонной книги

- Create — «быстрая» вставка
- Read, Update, Delete — всё сводится к поиску записи

Инвариант

В телефонной книге есть 0 или 1 незачёркнутая запись для любой ФИО

Если хотим оценить точнее, то нужно кое-что вспомнить...

$O(n)$ и $o(n)$

Формальное определение[a]

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

$$f(n) \in O(g(n)) \Leftrightarrow \exists k > 0 \; \exists n_0 \; \forall n > n_0 \; f(n) \leq g(n) \cdot k$$
$$f(n) \in o(g(n)) \Leftrightarrow \forall k > 0 \; \exists n_0 \; \forall n > n_0 \; f(n) \leq g(n) \cdot k$$

$o(n)$: Сравнение

Компьютер мощностью 1 Gops:

$o()$	**1 Gops**	**1 Tops**
$o(n)$	1 секунда	16 минут
$o(log_2(n))$	30 микросекунд	40 микросекунд
$o(n^2)$	32 года	32 миллиона лет
$o(n \cdot log_2(n))$	30 секунд	11 часов
$o(k \cdot n)$	k с.	$1000k$ с.

$o(n)$: Примеры

- $o(n)$ — последовательный перебор
- $o(n^2)$ — сравнение всех со всеми
- $o(log_2(n))$ — бинарный поиск
- $o(n \cdot log_2(n))$ — сортировка Quick Sort
- $o(k \cdot n)$ — сортировка Radix Sort

Вернёмся к телефонной книге

Пусть в книге 100 записей.

- Если совсем не сортировать, то поиск за
  - $100/2 = 50$ операций в среднем
- Если всё полностью отсортировано, то поиск за
  - $log_2(100) \approx 6.64$ операции
- А в бумажной телефонной книге
  - $log_2(28) + 100/28/2 \approx 6.59$ операции

Ограничения физического мира

```
L1 cache reference ..................... 0.5 ns
Branch mispredict ....................... 5 ns
L2 cache reference ...................... 7 ns
Mutex lock/unlock ..................... 25 ns
Main memory reference ................. 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns = 3 us
Send 2K bytes over 1 Gbps network ....... 20,000 ns = 20 us
SSD random read ........................ 150,000 ns = 150 us
Read 1 MB sequentially from memory ..... 250,000 ns = 250 us
Round trip within same datacenter ...... 500,000 ns = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns = 1 ms
Disk seek ........................... 10,000,000 ns = 10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns = 150 ms
```

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Disclaimer

- Общепризнанных устоявшихся классификаций нет[1]
- Классификации неполны по определению
- Многие хранилища данных являются гибридными или настраиваемыми
- Любые противоречия случайны и на совести автора

[1] http://en.wikipedia.org/wiki/Database#Database_type_examples

Data Model

- Hierarchical
- Relational
- Graph
- Object
- Document
- Key-value
- Columnar
- Deductive
- Spatial
- ...

Deployment

- Embedded (in-process)
- Separate
- Distributed
- ...

Storage

- In-memory
- On-disk
- Data warehousing
- ...

Usage patterns

- Messaging
- Counters
- Logging + Batch processing
- OLTP
- Entity enrichment
- Versioning
- ...

И это ещё не всё

- Query Language
- Consistency
- Availability & Replication
- Partition tolerance
- Transactions
- Backups
- Security
- ...

Web Scale

In my limited experience I've dealt with long-lived network partitions in a single data center (DC), PDU [power distribution unit] failures, switch failures, accidental power cycles of whole racks, whole-DC backbone failures, whole-DC power failures, and a hypoglycemic driver smashing his Ford pickup truck into a DC's HVAC [heating, ventilation, and air conditioning] system. And I'm not even an ops guy.
Coda Hale

Examples

- Одноклассники
  - 48 M DAU
  - 1.2 Tb/s
- Instagram
  - 300 M DAU
  - 4.2 B likes per day
  - 95 M videos/photos per day
- Facebook
  - 1.23 B DAU
  - 350 M photos per day
  - 4 PB data per day
- The "Next-Gen"is Now[2]

[2] http://2016.secr.ru/lang/en/program/invited-speakers/crystal-valentine

Scalability

- Replication
  - Share nothing arhitectures
  - Machines/disks/NICs
- Vertical partitioning
  - By function, transaction, user type, ...
  - Loose coupling
  - Batch vs interactive
  - Independent scaling
- Horizontal partitioning
  - By year, geo
  - Using hash, segment, utilization, ...

Fault-tolerance

- Everything **constantly** breaks
  - HDD[3]
  - Memory
  - NIC
  - Chipset
  - CPUs
- Redundancy
  - RAID
  - Error correction: ECC, Reed-Solomon, ...
  - (Geo-)replication

[3] https://research.google.com/pubs/pub32774.html

Microservices

- Became hot in 2014[4]
- Embrace failures[5]
- Eventual consistency
- Polyglot persistence, CQRS, CRDT
- REST/RPC/(async) message passing (instead of SQL)
- Independent human/resources scaling
- Independent deployment
- Containers

[4] https://martinfowler.com/microservices/

[5] http://www.reactivemanifesto.org

Why NoSQL

- Scalability
  - Very large datasets
  - Write throughput
- Specialized queries
- Restrictive relational schema
- Planet-scale services
  - Geo distribution
  - Fault-tolerance

6

[6] T. Limoncelli et al. The Practice of Cloud System Administration: DevOps and SRE Practices for Web Services, Volume 2.

Drawbacks of NoSQL

- "Do one thing, and do it well."
- Not mature enough[7]
- No common standards/classification
- Requires deep understanding of internals
- Too much hype/marketing bullshit

[7] http://jepsen.io

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?

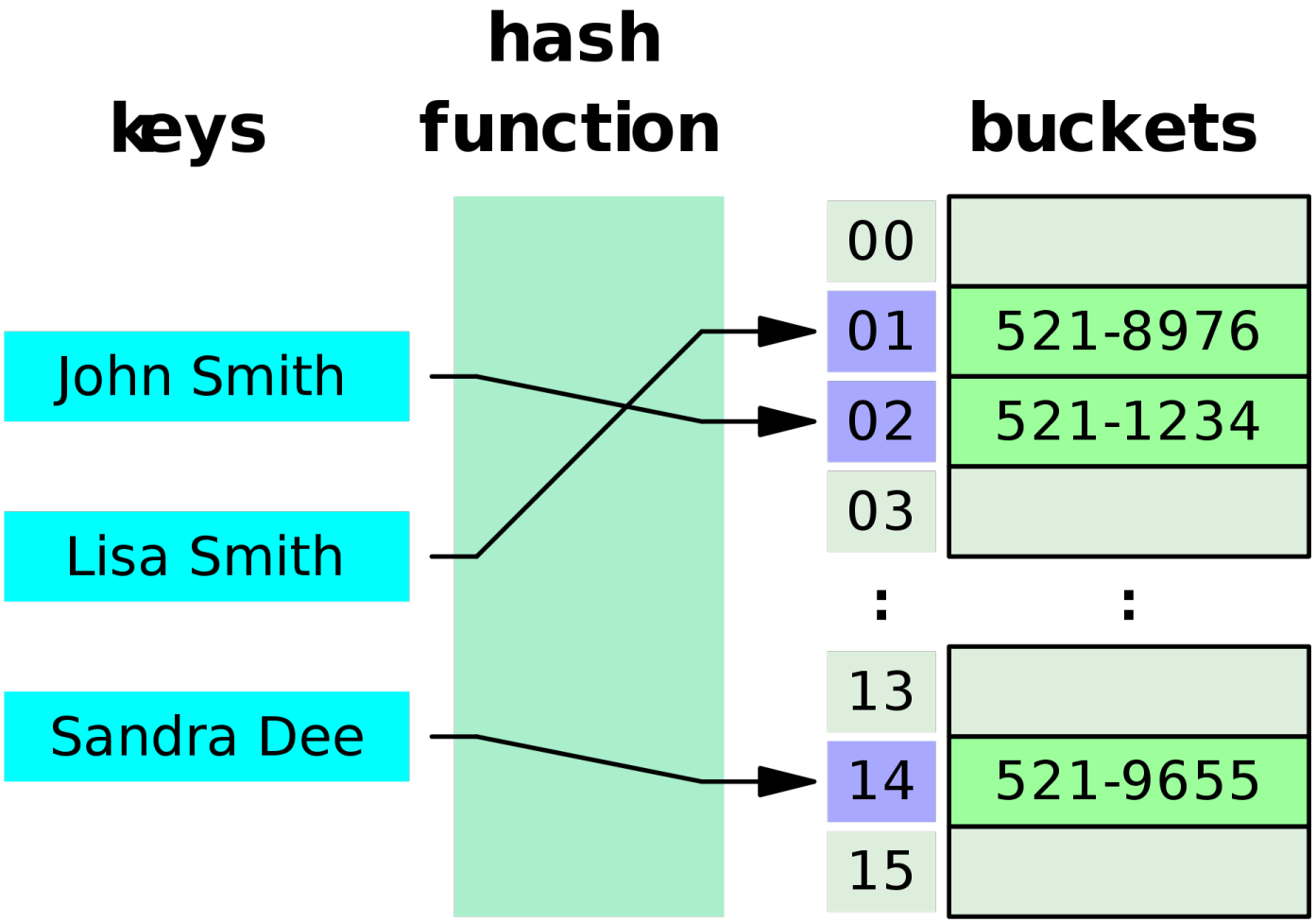
Hash table

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

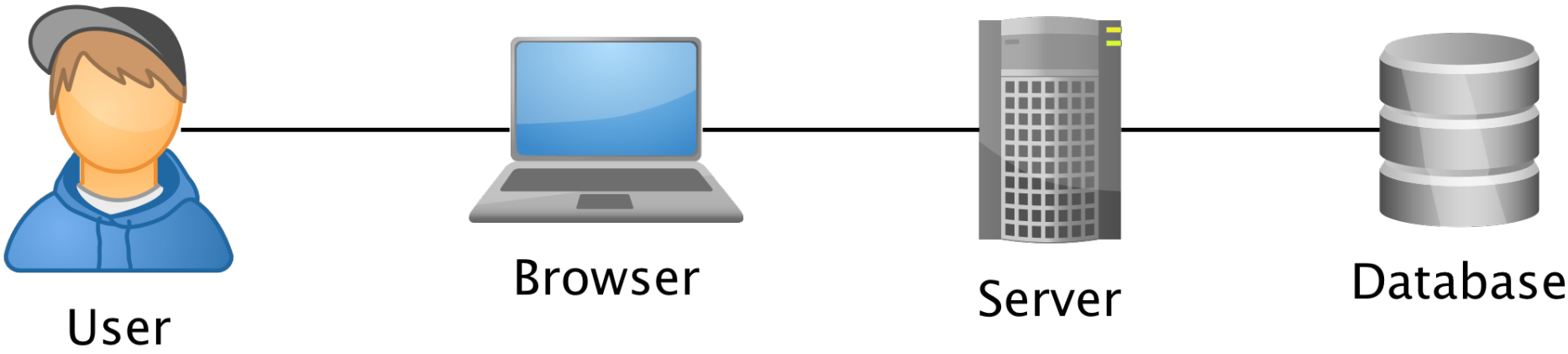
- `java.util.HashMap`
- `java.util.concurrent.ConcurrentHashMap`
- `http://en.wikipedia.org/wiki/Hash_table`
- Chris Okasaki. *Purely Functional Data Structures*[8]

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

Классические архитектуры Web-приложений

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

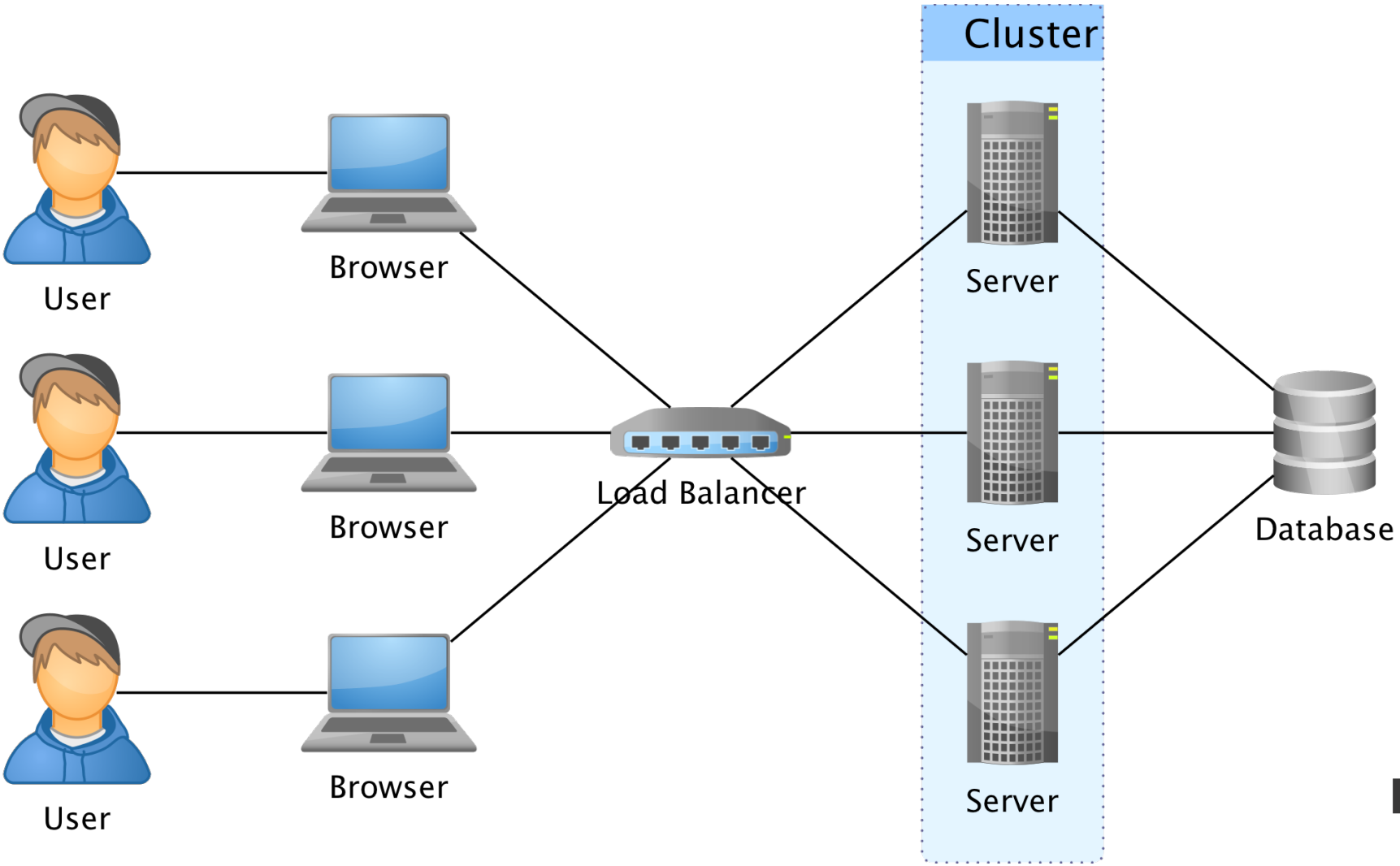
Трёхзвенный клиент-сервер



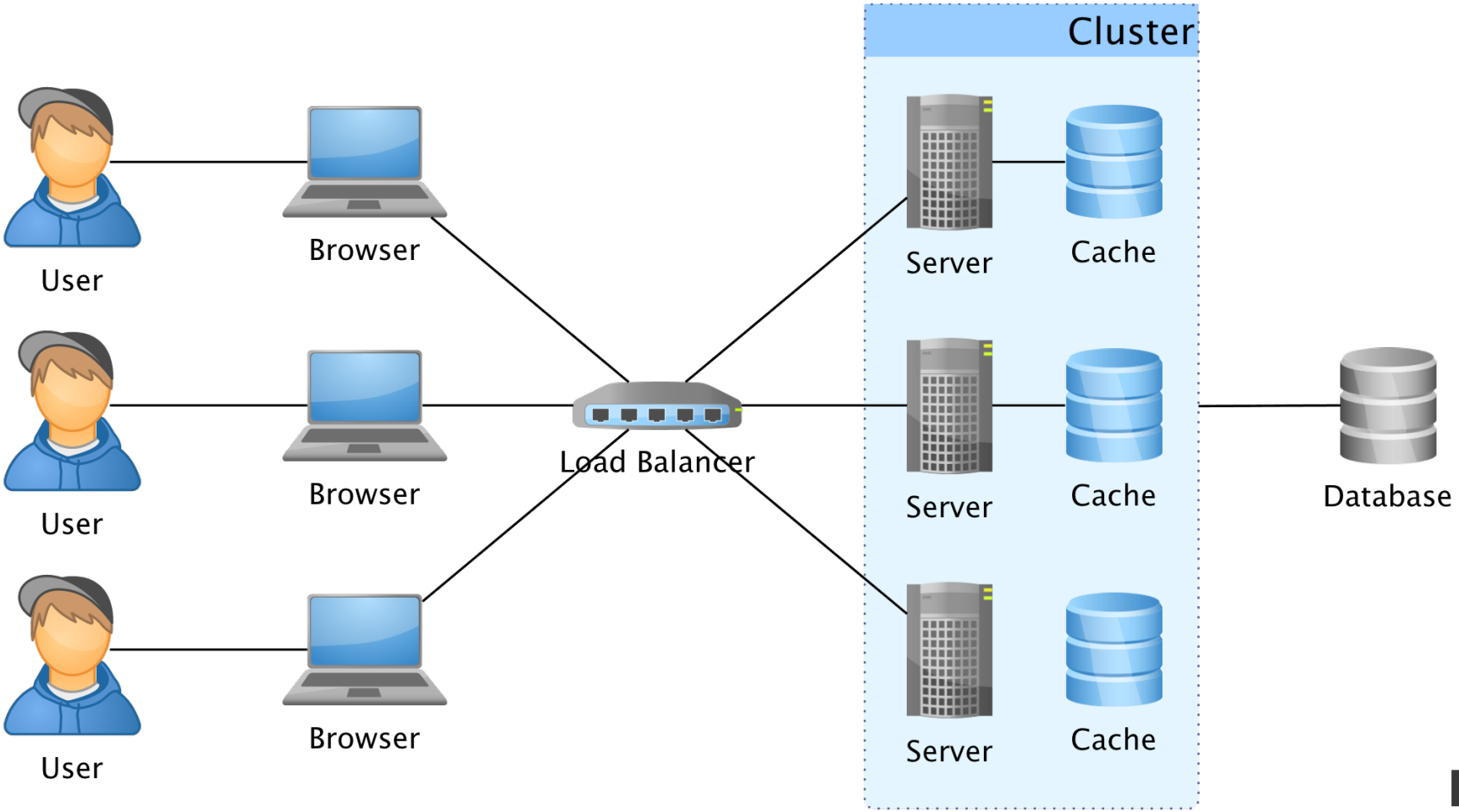
Зачем нужен Server:

- 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) getOrElse {
5            val response = buildResponse(request)
6            cache.put(request, response)
7            response
8        }
9    }
```

Анализ cache per server

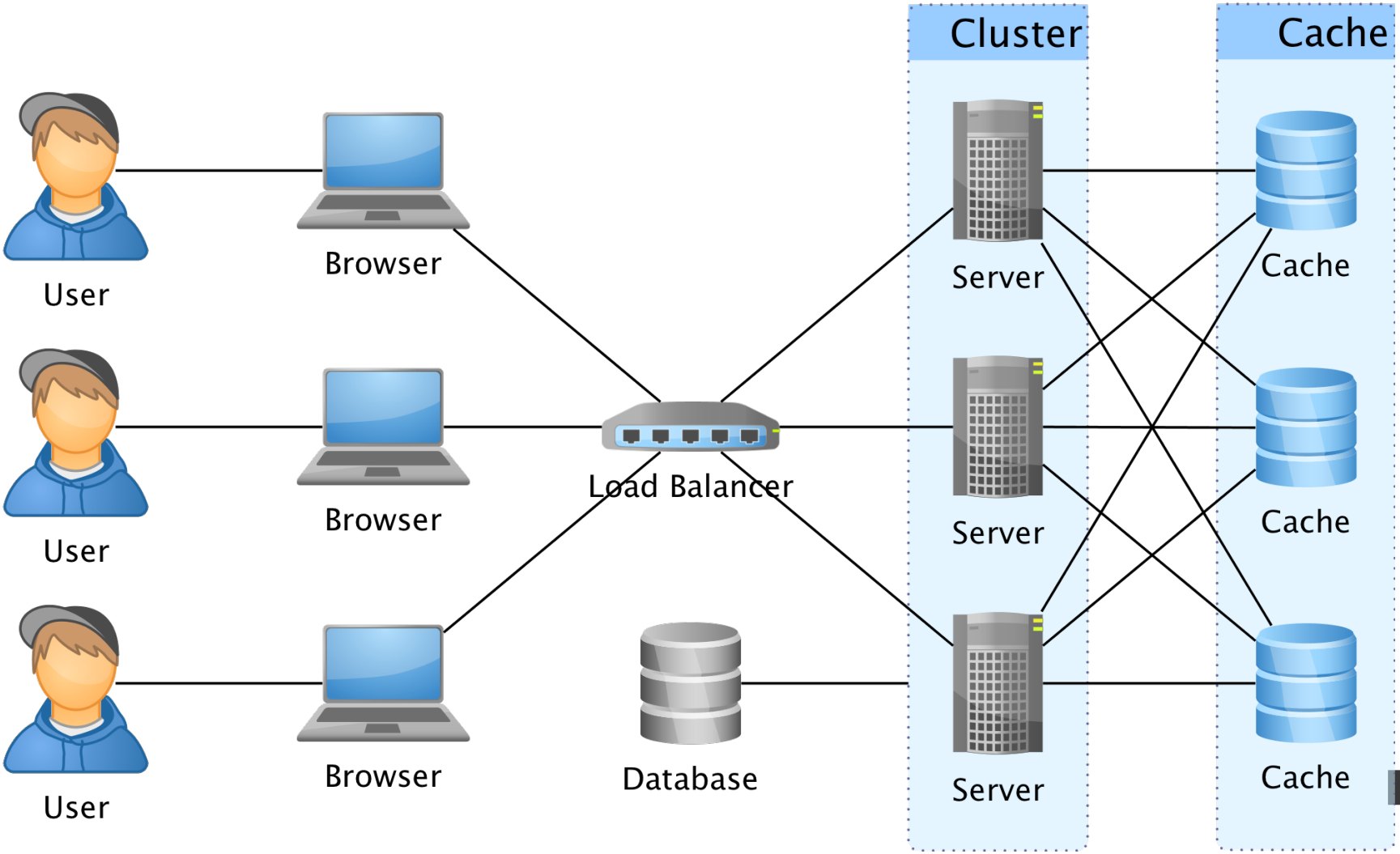
Достоинства

- 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.mail.ru", "cache02.mail.ru")
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

- 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

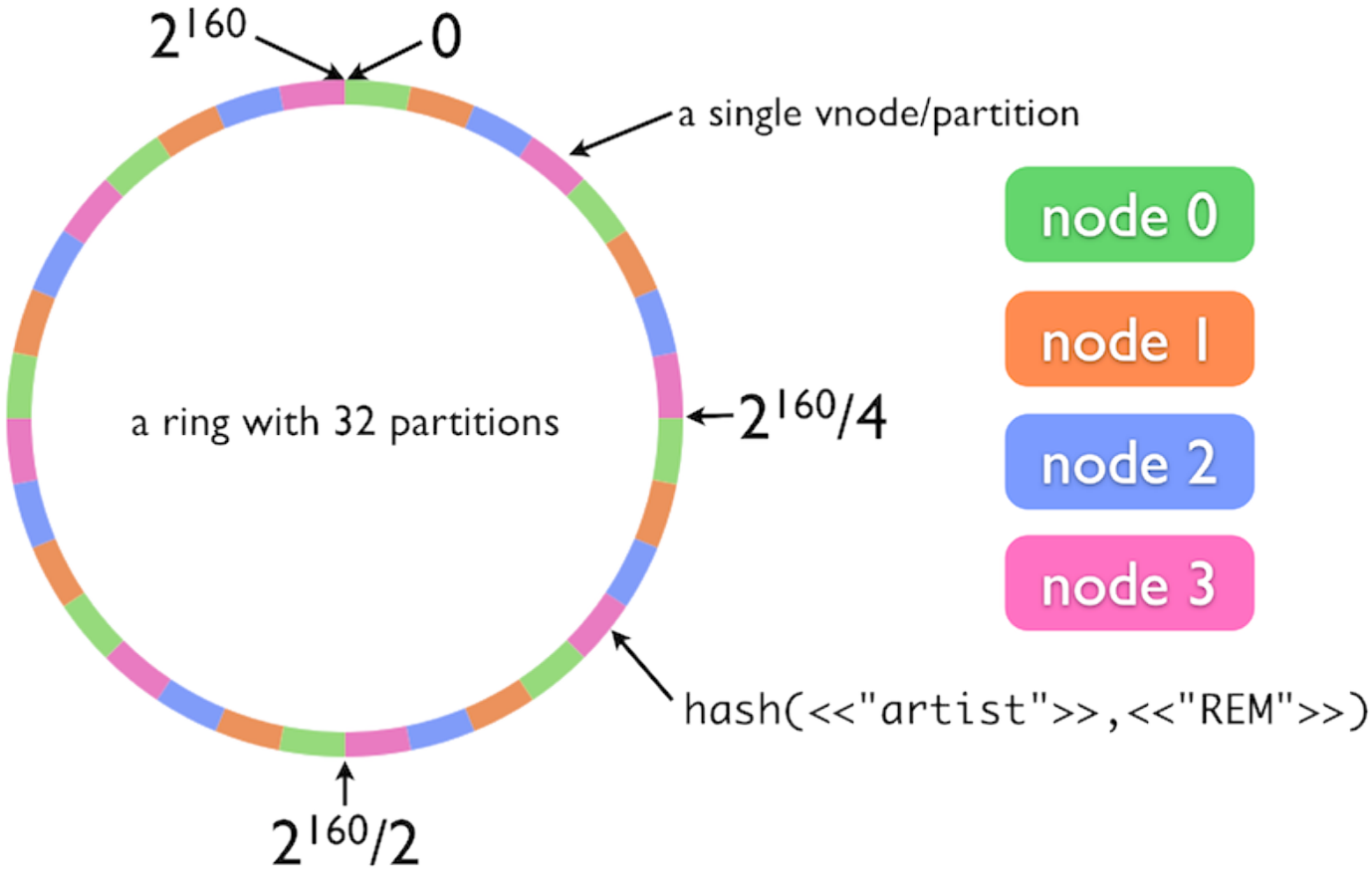
Будем отображать ключи в слоты, а затем слоты на машины:

```
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:

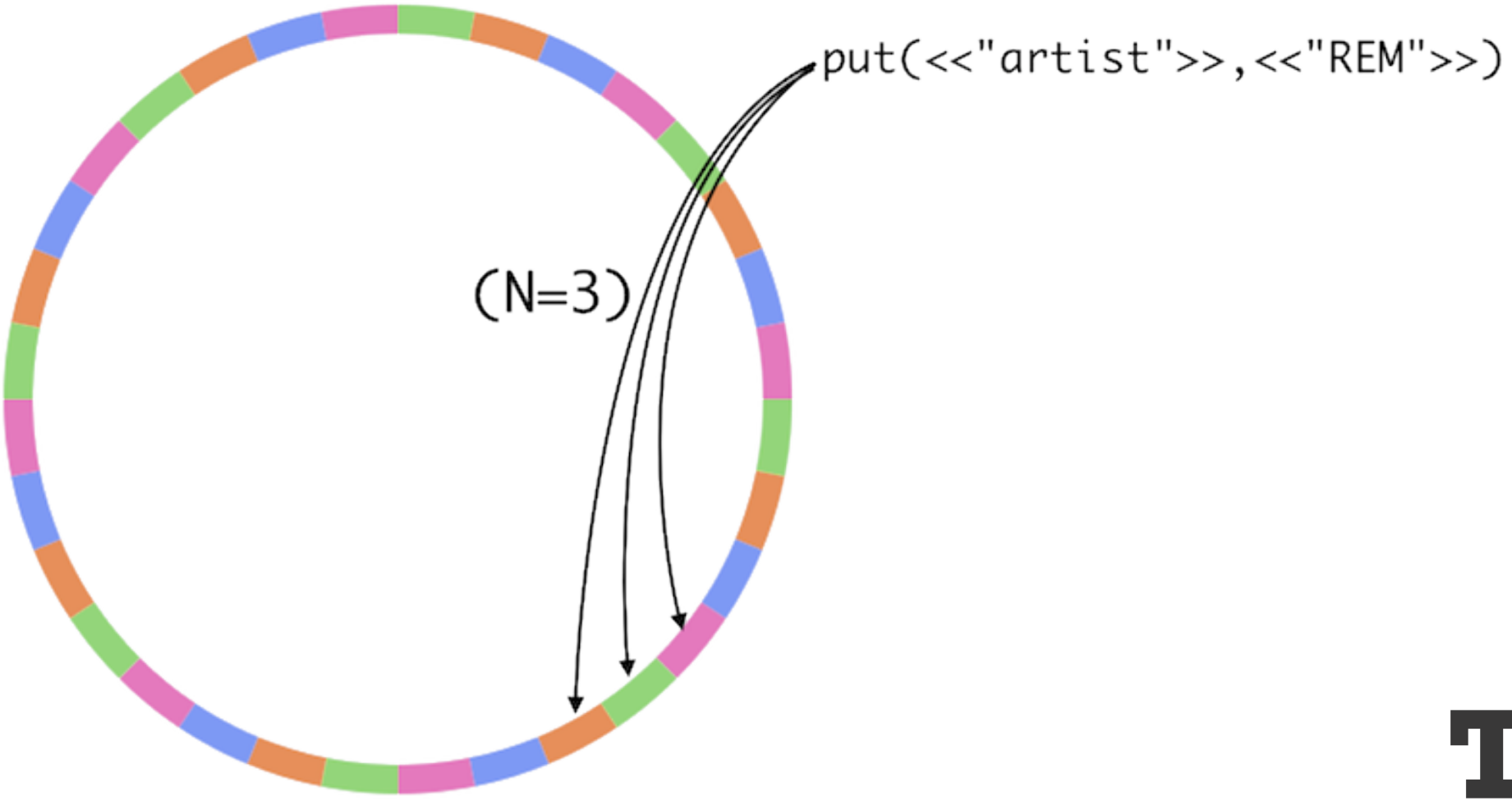
```
1    val slots = Vector(100, 200, 275, 325, 400)
```

Consistent hashing: по-взрослому



http://docs.basho.com/riak/1.1.4/references/

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[9]
- 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[10]

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

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

Сейчас

- Делимся на команды
- Выбираем тему
- Уточняем постановку
  - Минимальная функциональность
  - Объёмы
  - Ограничения
  - Нагрузка
  - Масштабируемость
  - Отказоустойчивость
- Начинаем проектировать

Темы

- Сервис авторизации
- Профили пользователей
- Социальный граф
- Рекомендации друзей
- Фотки
- Сообщения
- Группы
- Счётчики
- Рекомендации
- Реклама
- ...

К следующему занятию

- Презентация: схемы и диаграммы[11]
- Схема хранения данных
- API
- Архитектура
- Этапы
- Вопросы
- Ответы
- Уточнение решения
- Начинаем прототипировать

[11] `http://plantuml.com`

Как

- Реп на Github
- Артефакты + wiki
- Ссылку на mail@incubos.org

Non functional requirements

- Масштабируемость
- Отказоустойчивость
- Безопасность
- Бекапы
- Аудит
- Апгрейд
- Деградация
- А что, если...
- ...[12]

[12] https://en.wikipedia.org/wiki/Non-functional_requirement

Вопросы?

- https://polis.mail.ru/blog/view/111/
- https://incubos.org/contacts/
- Не забудьте про ДЗ