За гранью NoSQL: NewSQL на Cassandra

Олег Анастасьев

Ведущий разработчик

Одноклассники, ok.ru

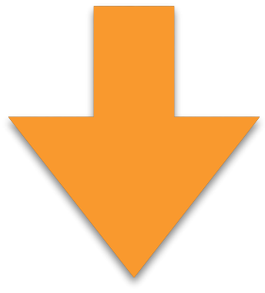
Cassandra @ одноклассники

* Используем с 2010 года
  - 0.6, 1.2, 2.0
* Сейчас
  - 27 кластеров
  - 500 нод
  - 300 TB данных

* Самый быстрый: 1M оп/сек (48 нод)
* Самый большой: 130TB (96 нод)

SQL Server 2005

* Используем с начала времен
* 200 серверов, 50 TB данных
* Шардинг
  - F(Entity_Id) -> Token -> Node
  - F(Master_Id) === F(Detail_Id)
  - Группировка таблиц по функциональной близости



Группа Локальной Транзакции

Fast SQL Server 2005

* ~~DB JOIN~~
* ~~Foreign key constraints~~
* ~~Stored Procs, Triggers~~
* Read uncommitted (noTx)
* Короткие транзакции <100ms
* Нет массовых UPDATE, DELETE
* Запросы только по индексам

Проблемы с SQL

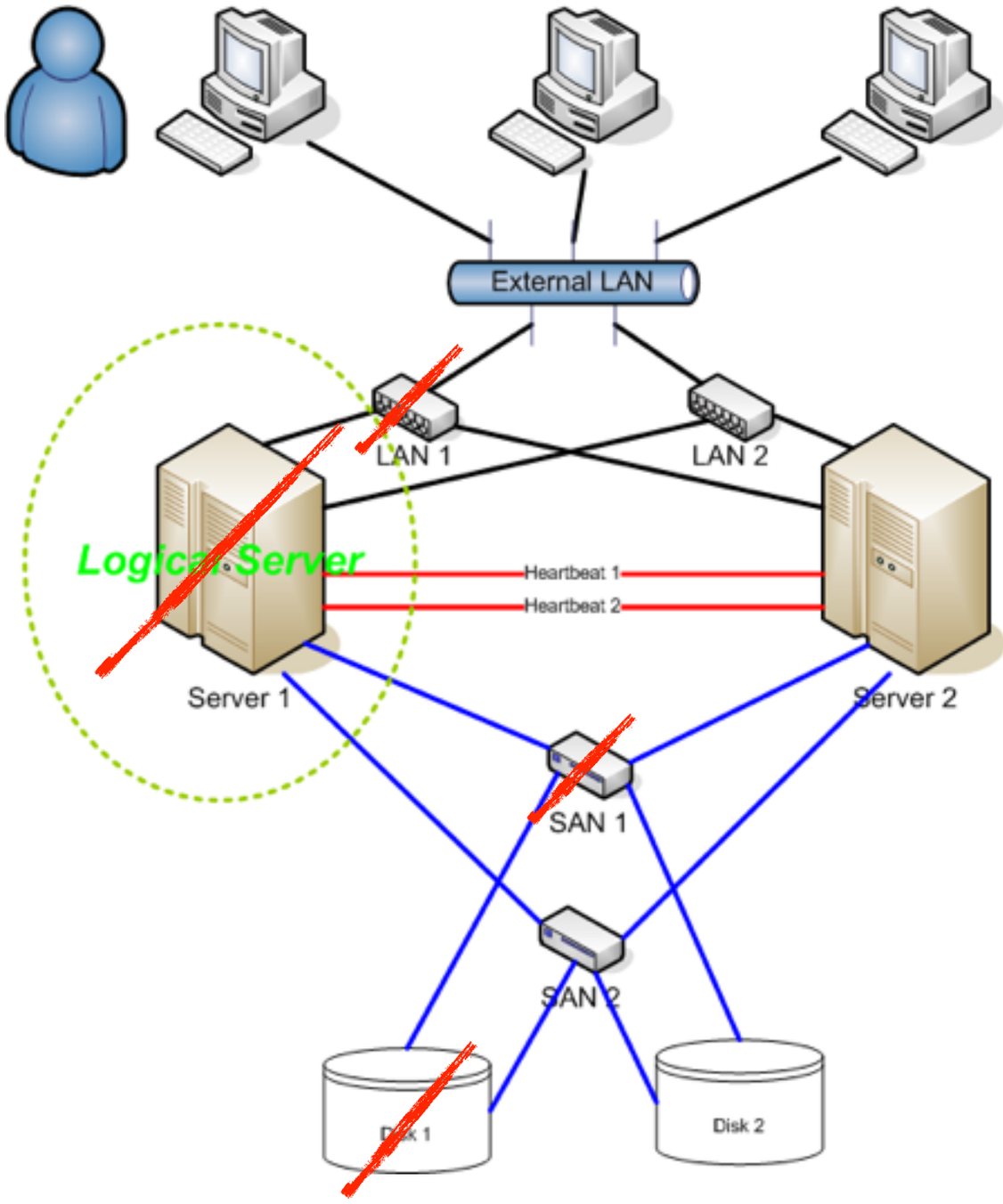
* BSoD
* Ручное масштабирование
* Downtime при работах с БД, ДЦ
* Скорость на запись
* Плохая отказоустойчивость

1 БД сервер = раз в 3 года

64 сервера = раз в 3 недели

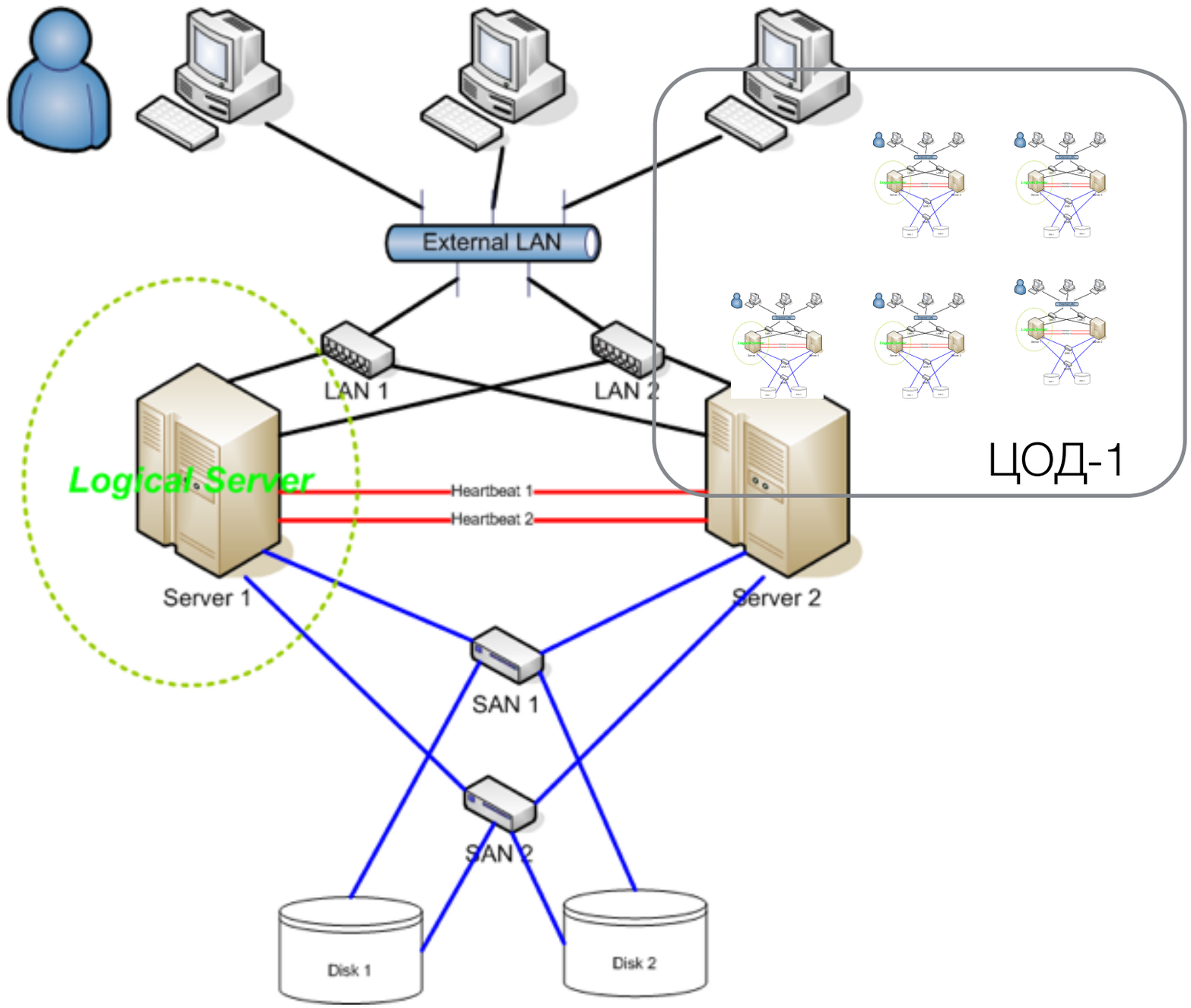
200 серверов > раза в неделю

Отказоустойчивость

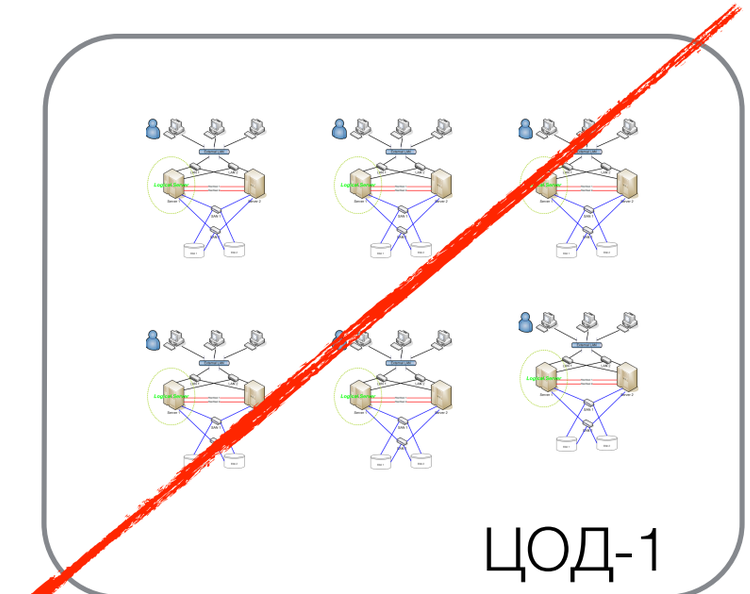


http://en.wikipedia.org/wiki/High-availability_cluster

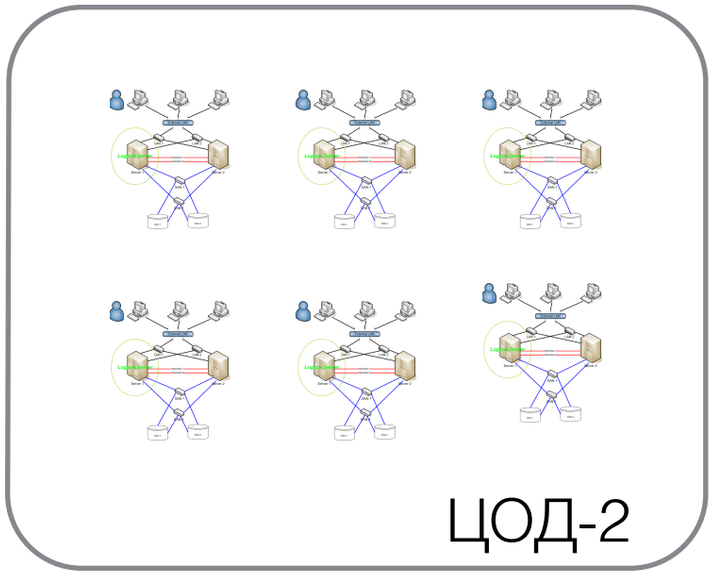
Отказоустойчивость

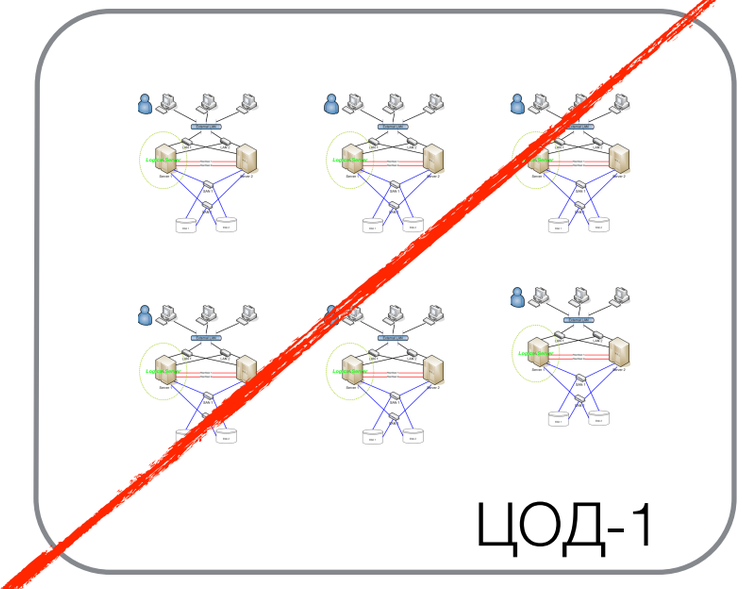


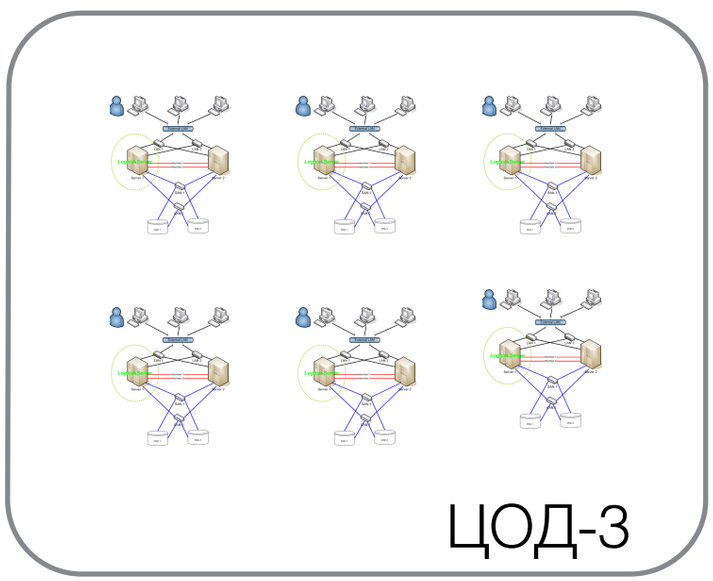
Отказоустойчивость

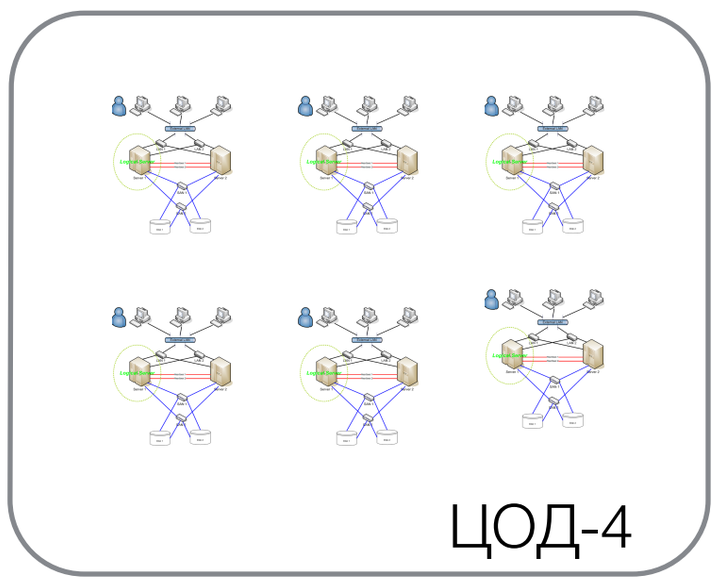


Отказоустойчивость





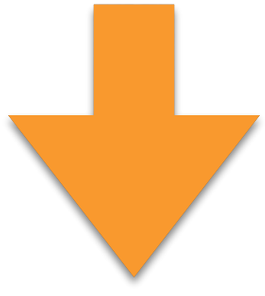




одноклассники

Отказоустойчивость SQL

* Дорогое оборудование
  - x10 стоимости 1U сервера
* Глючно и сложно
  - Штормит при failover
  - 10 % переключений неудаются
* Её нет
  - при отказе ЦОД



Нужно новое хранилище

Простая транзакция

```
TX.start(“Albums”, id);
Album album = albums.lock(id);
Photo photo = photos.create(…);

if (photo.status == PUBLIC ) {
   album.incPublicPhotosCount();
}

TX.commit();
```

* Чтение – модификация – изменение
* Разные сущности, несколько таблиц
* Возможна конкурентная модификация

Новое хранилище 

* Программистам:
  - ACID
  - SQL
* Админам:
  - Чтение и запись при отказе ЦОД
  - Масштабируемость на ходу
  - Дешевые сервера

* Фиксабельный код (OpenSource,Java)

SQL ?

* Масштабируемость
* Отказоустойчивость
  - Кластер
  - Конфликты
  - SQL

* ~~DB JOIN~~
* ~~FK~~
* ~~SP, Triggers~~
* ~~2х фазный коммит~~
* Read uncommitted
* Нет массовым UPDATE, DELETE
* Запросы только по индексам

NoSQL ?

* ACID
* ~~SQL~~

* Cassandra 2 CQL

Cassandra 2.0

```
CREATE TABLE photos
   (id bigint KEY, owner bigint,…)

SELECT * FROM photos WHERE id=?

UPDATE photos SET … WHERE id=?
```

* Что еще есть:
  - Масштабируемость
  - Скорость https://github.com/jbellis/YCSB
  - Кворумы, speculative retry
  - Batchlog
  - “Lightweight” transactions ?

Cassandra 2.0

* Что дописать:
  - ACID транзакции
  - OLTP индексы
  - Генератор монотонных ид

```
CREATE TABLE photos
   (id bigint KEY, owner bigint,…)

// OLAP index
SELECT * FROM photos WHERE geomark=‘SPB’;

// OLTP index
SELECT * FROM photos WHERE owner=? AND status=?;
```

C*One

Cassandra
Gossip & Messaging

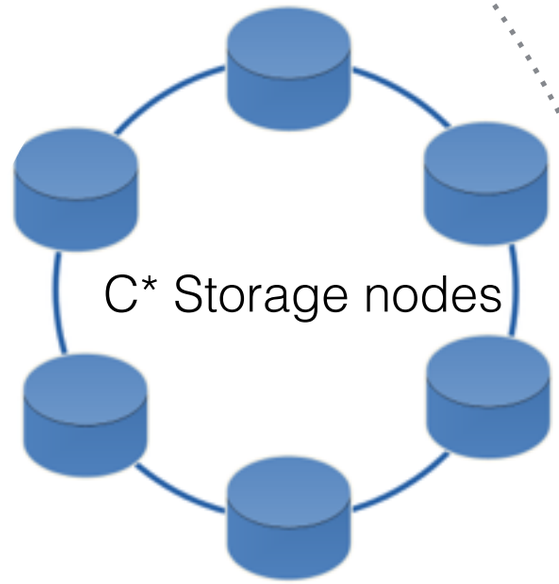
clients

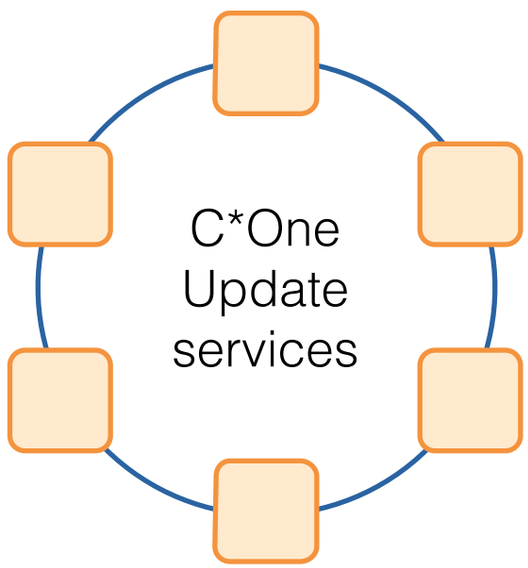
"Heartbeat"

Схема данных

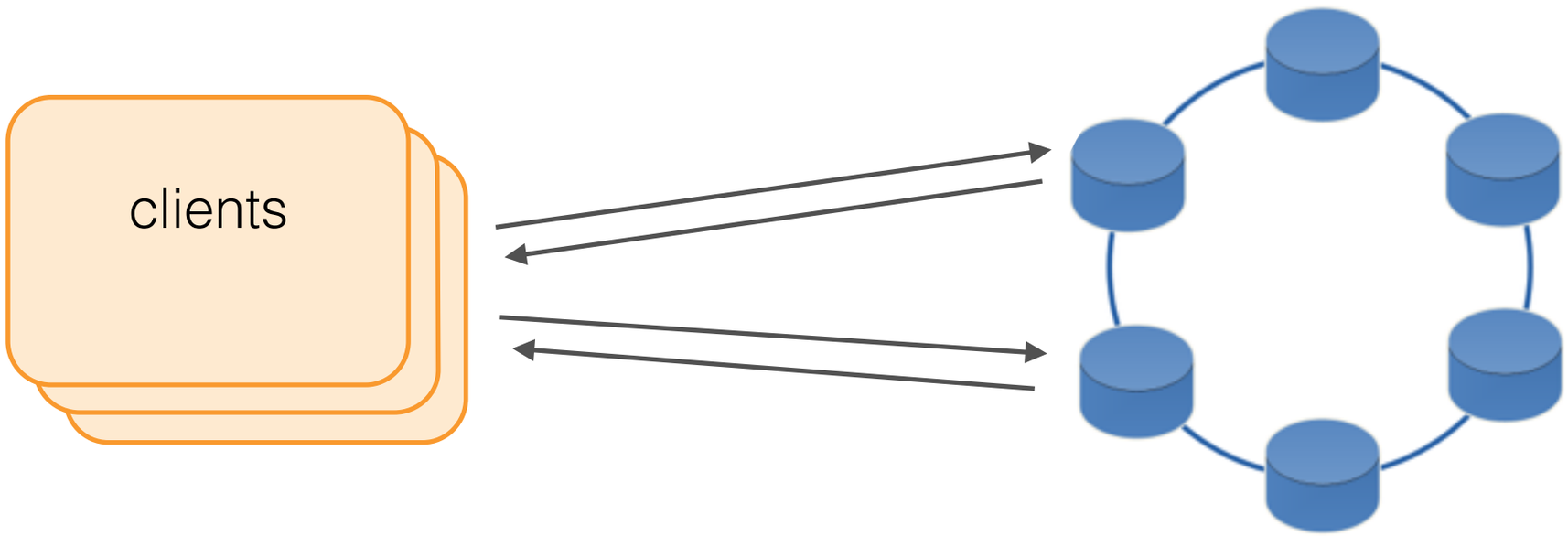
Схема партиционирования

Топология кластера



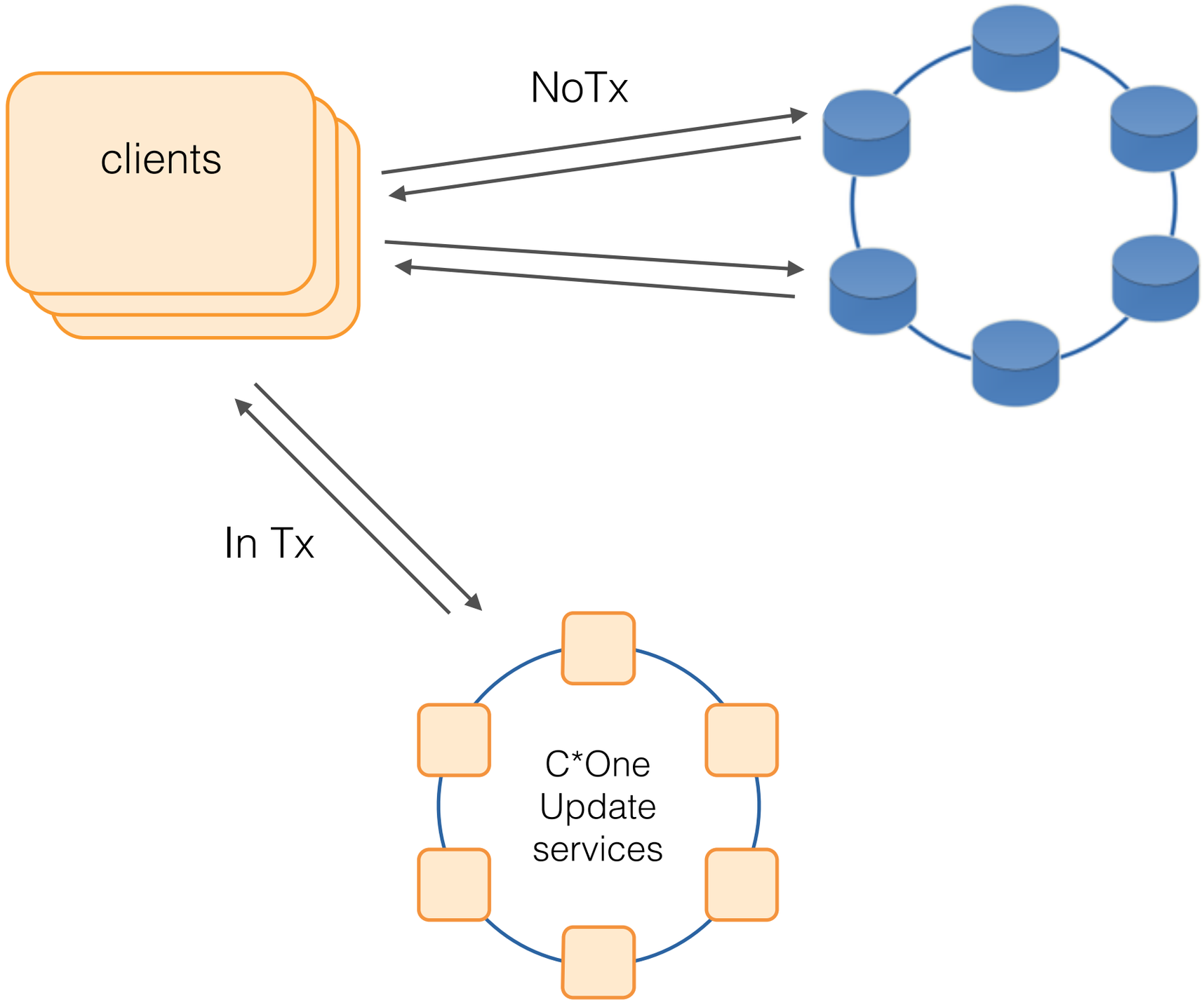


Клиенты



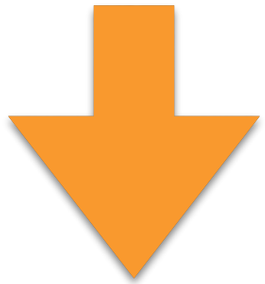
* Fat client протокол
* Координатор = клиент
* Надежнее, быстрее

Клиенты



C*One Update Srvs

* Управляет блокировками
* Выдает монотонный timestamp



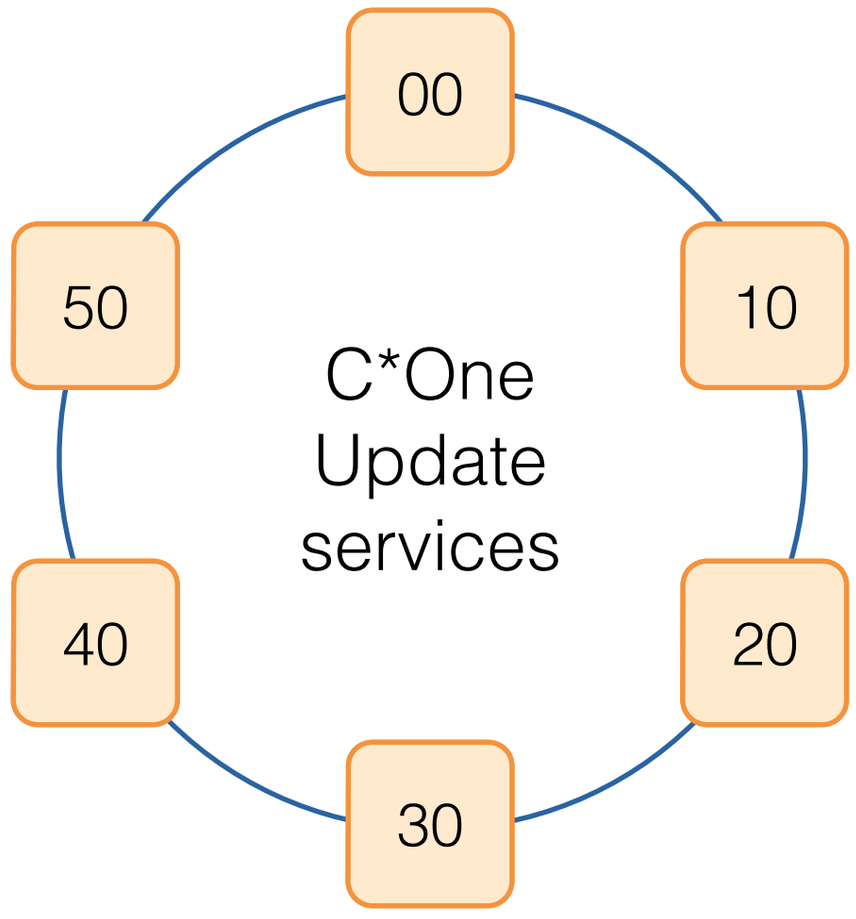
Lamport Timestamp

https://ru.wikipedia.org/wiki/Часы_Лэмпорта

* Управляет транзакциями
* … и изменением данных в них

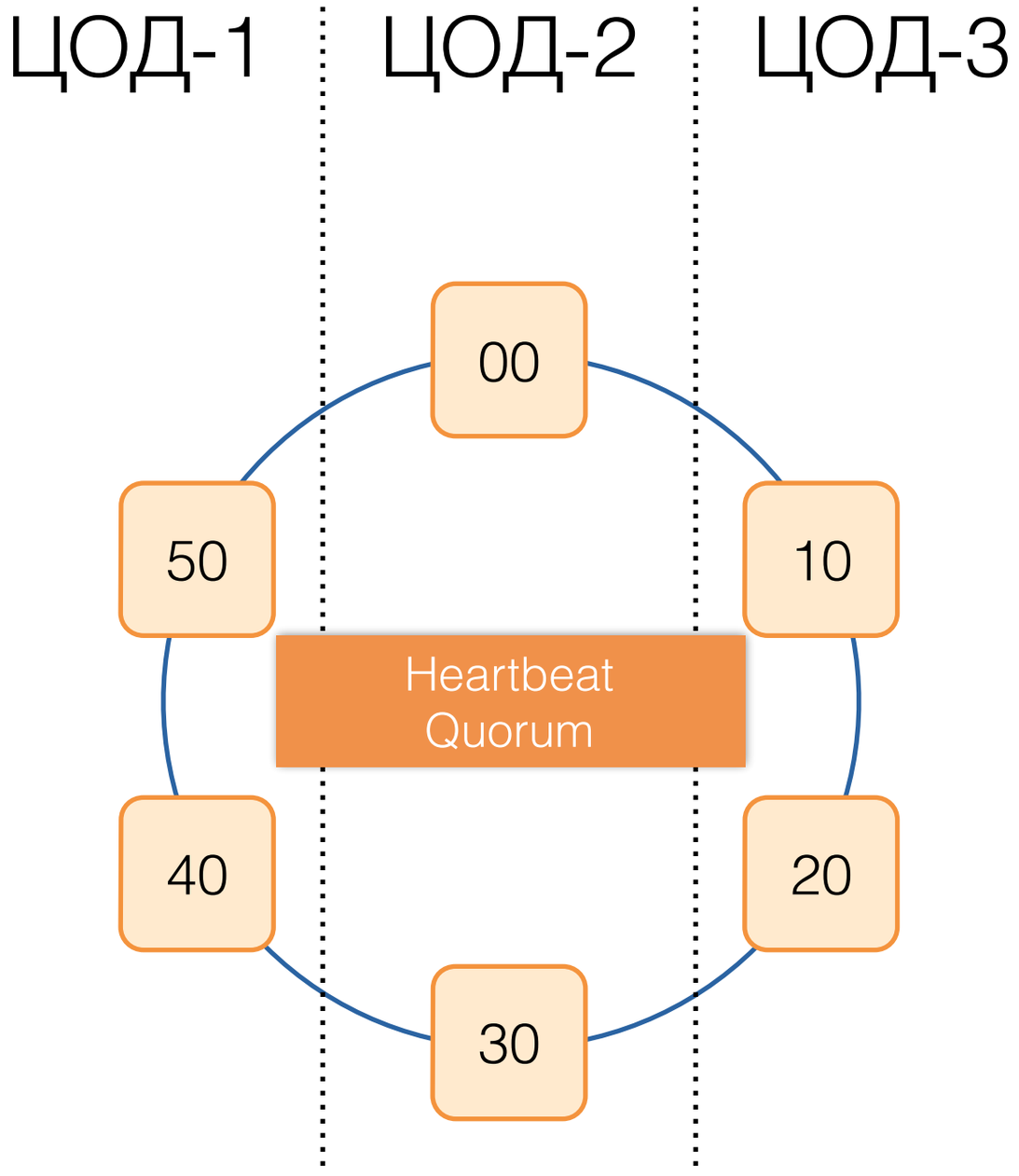
Блокировки

* Шардинг
  - F(Entity_Id) -> Token -> Node
  - F(Master_Id) === F(Detail_Id)



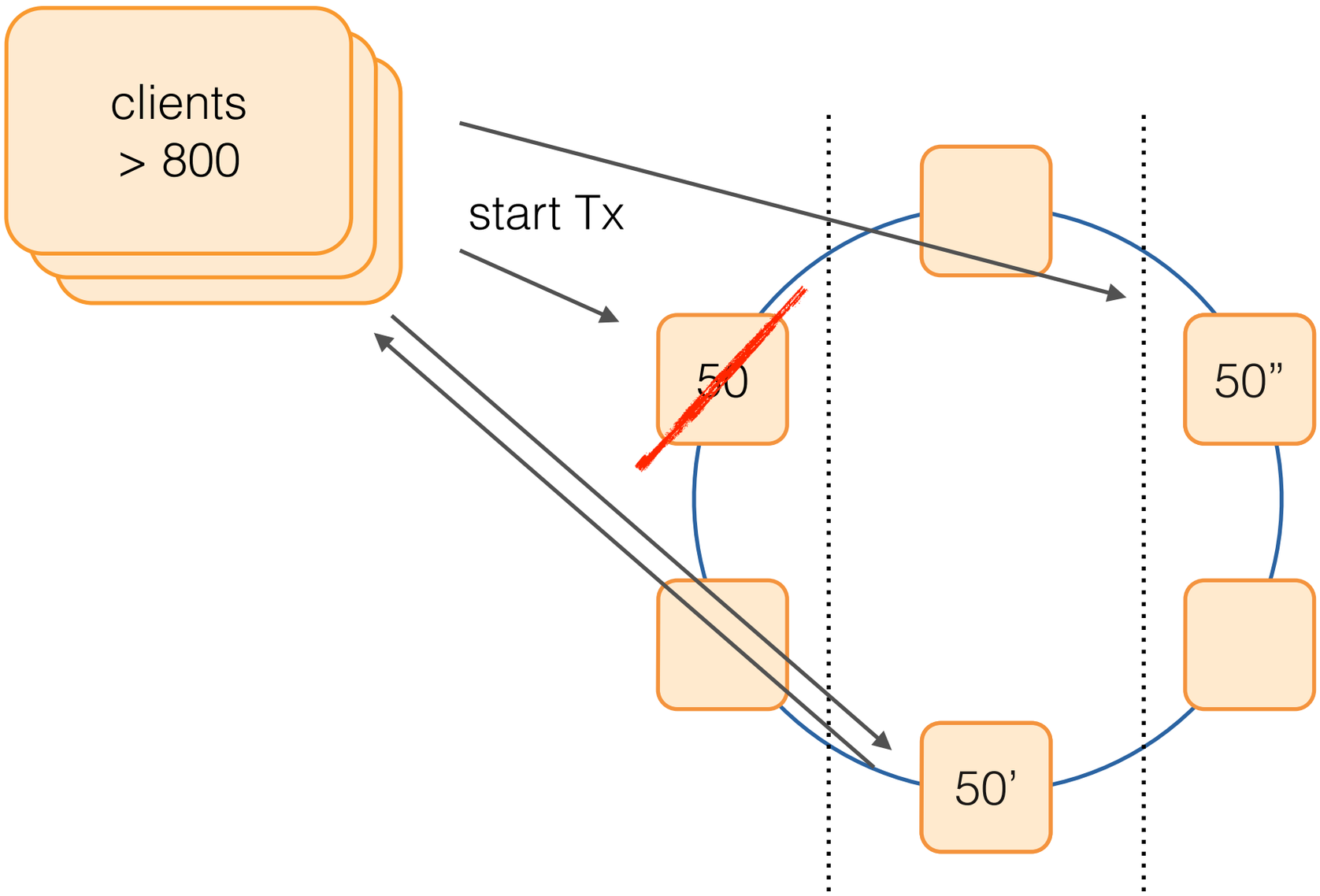
* Мастера групп транзакций
* Простые блокировки в памяти

Отказы



* 50ms тик
* G1 GC
* 200ms до обнаружения отказа

Резервирования

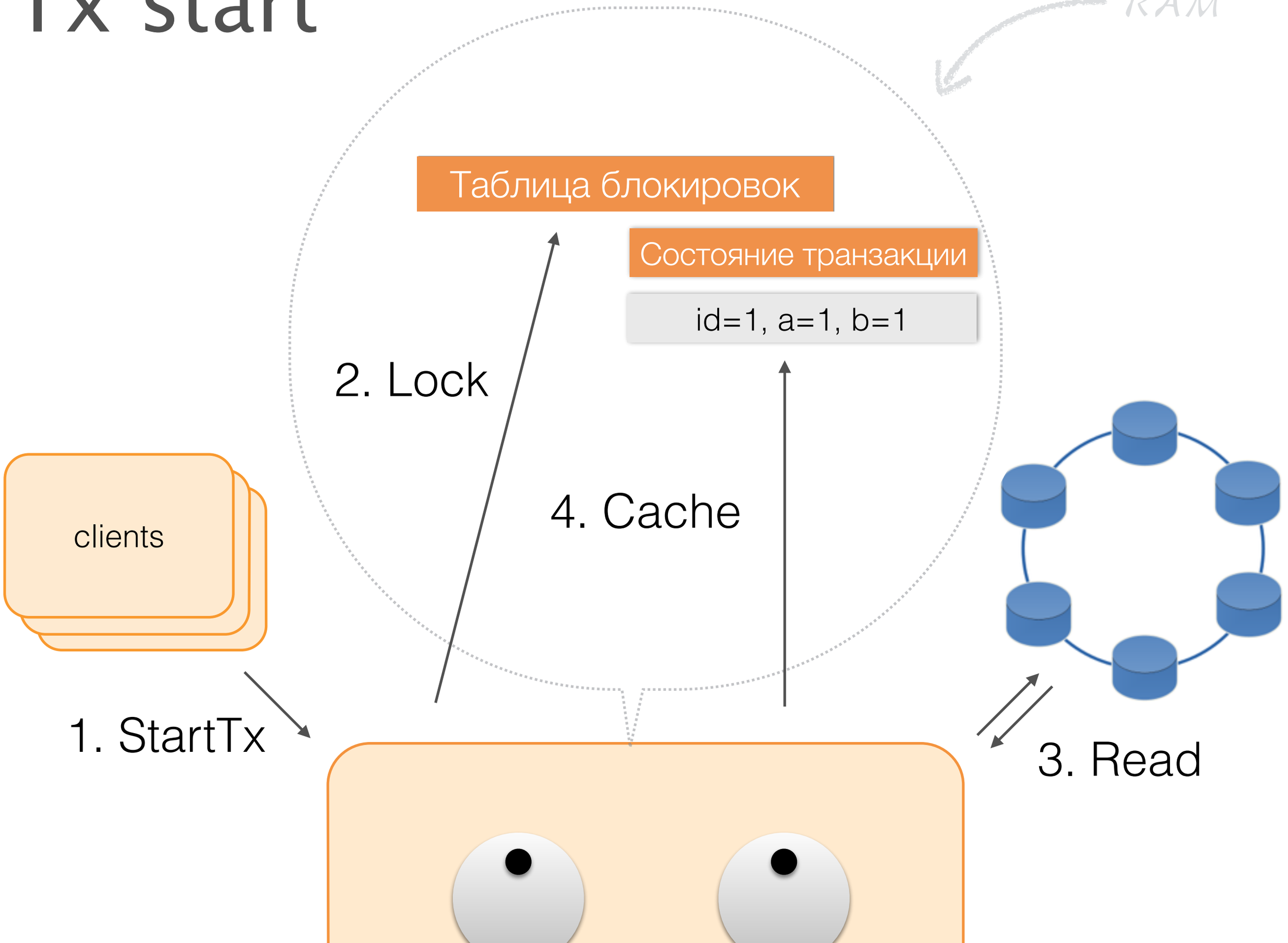


* ~~Протокол выбора мастера~~
* Спекулятивный старт транзакции

Нерожденные транзакции

* Запросы на открытие в очередь
  - (в памяти резервных)
  - Удаляются через таймаут
* Если происходит отказ мастера
  - отрабатываем их из очереди
  - посылаем ответ клиенту
    (отказывается, если уже открыл)

Tx start


RAM
Таблица блокировок
Состояние транзакции
id=1, a=1, b=1
2. Lock
4. Cache
clients
1. StartTx
3. Read

Tx write

RAM

Таблица блокировок

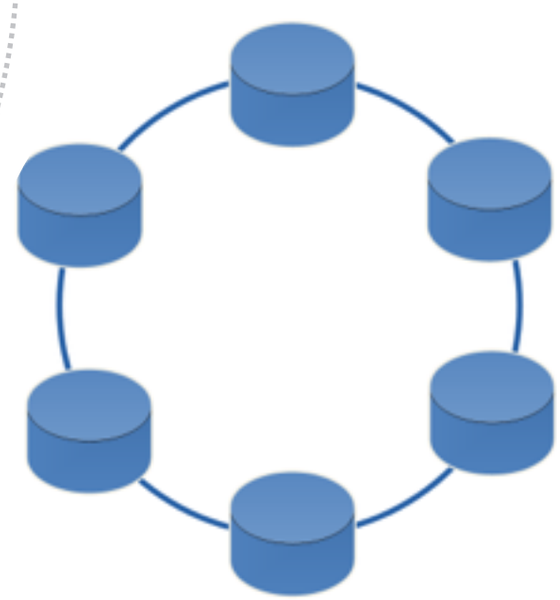
Состояние транзакции

id=1, a=2, b=2

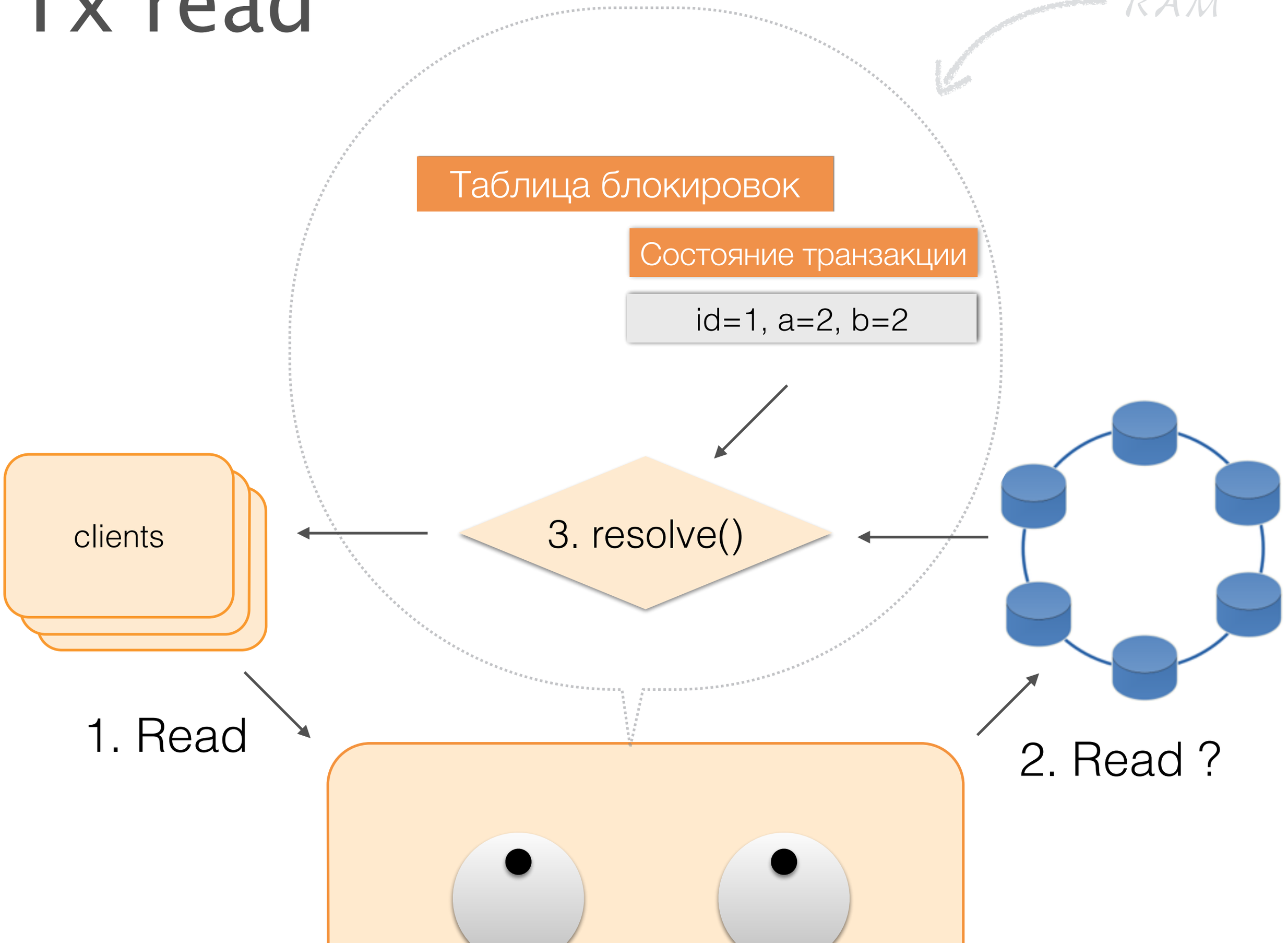
2. File

clients

1. UPDATE



Tx read


RAM
Таблица блокировок
Состояние транзакции
id=1, a=2, b=2
clients
3. resolve()
1. Read
2. Read ?

Tx commit

RAM

Таблица блокировок

Состояние транзакции

id=1, a=2, b=2

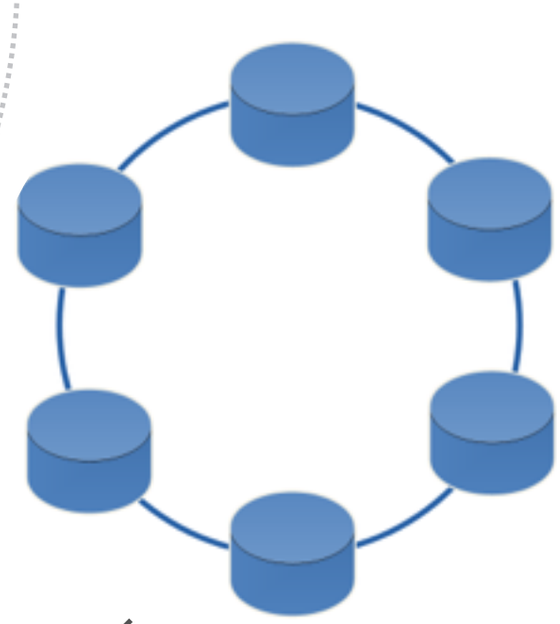
2

LOGGED BATCH

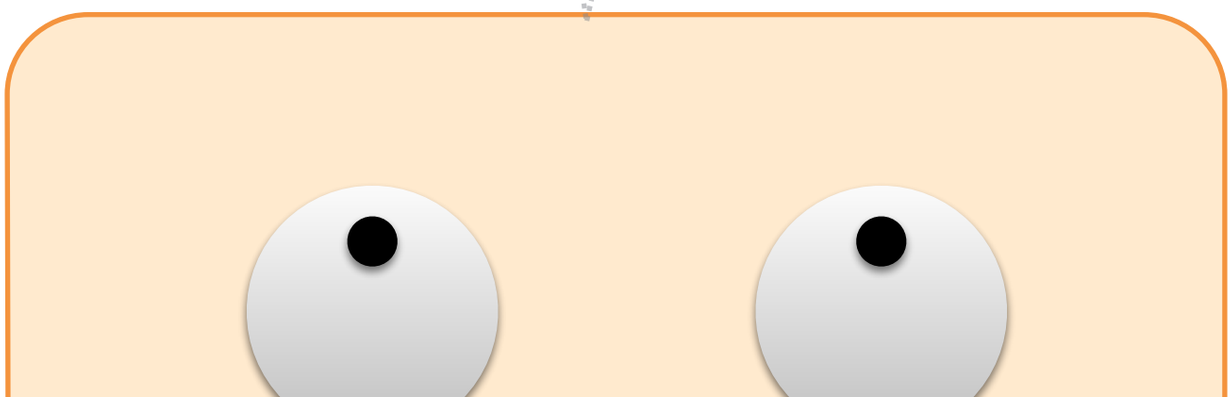
clients

1. Commit

3



4. Ack



Tx rollback

RAM

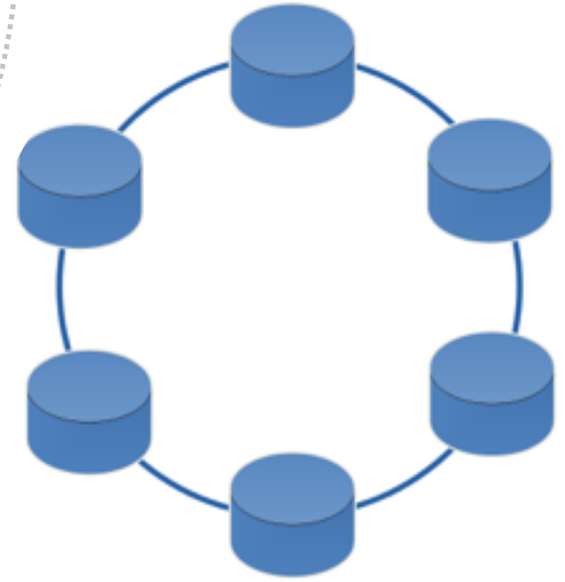
Таблица блокировок

Состояние транзакции

id=1, a=2, b=2

clients

1. Rollback



ACID

* Atomicity
  - logged batch или ничего
* Consistency
  - приложение
* Isolation
  - Блокировки
  - Read Committed
* Durability
  - кворумные чтения и запись в Cassandra

Индексы в Cassandra 2

```
CREATE TABLE photos (
   id bigint primary key,
   owner bigint,
   modified timestamp
```

```
SELECT *
   WHERE owner=?
     AND modified>?
```

* ~~CREATE INDEX (owner, modified) ?~~
  - Нет составных индексов
* ~~CREATE INDEX (owner), filter in RAM ?~~
  - high cardinality
  - 100K удалений макс
  - Не скалируется, очень медленно читать

Индексы в C*One

Primary Key

id	owner	modified	caption	access	…
1	111	9.10.2014	“kitty miau”	PUB	…

```
INDEX i1 ON photos (owner, modified)
   VALUES (caption,access,…);
```

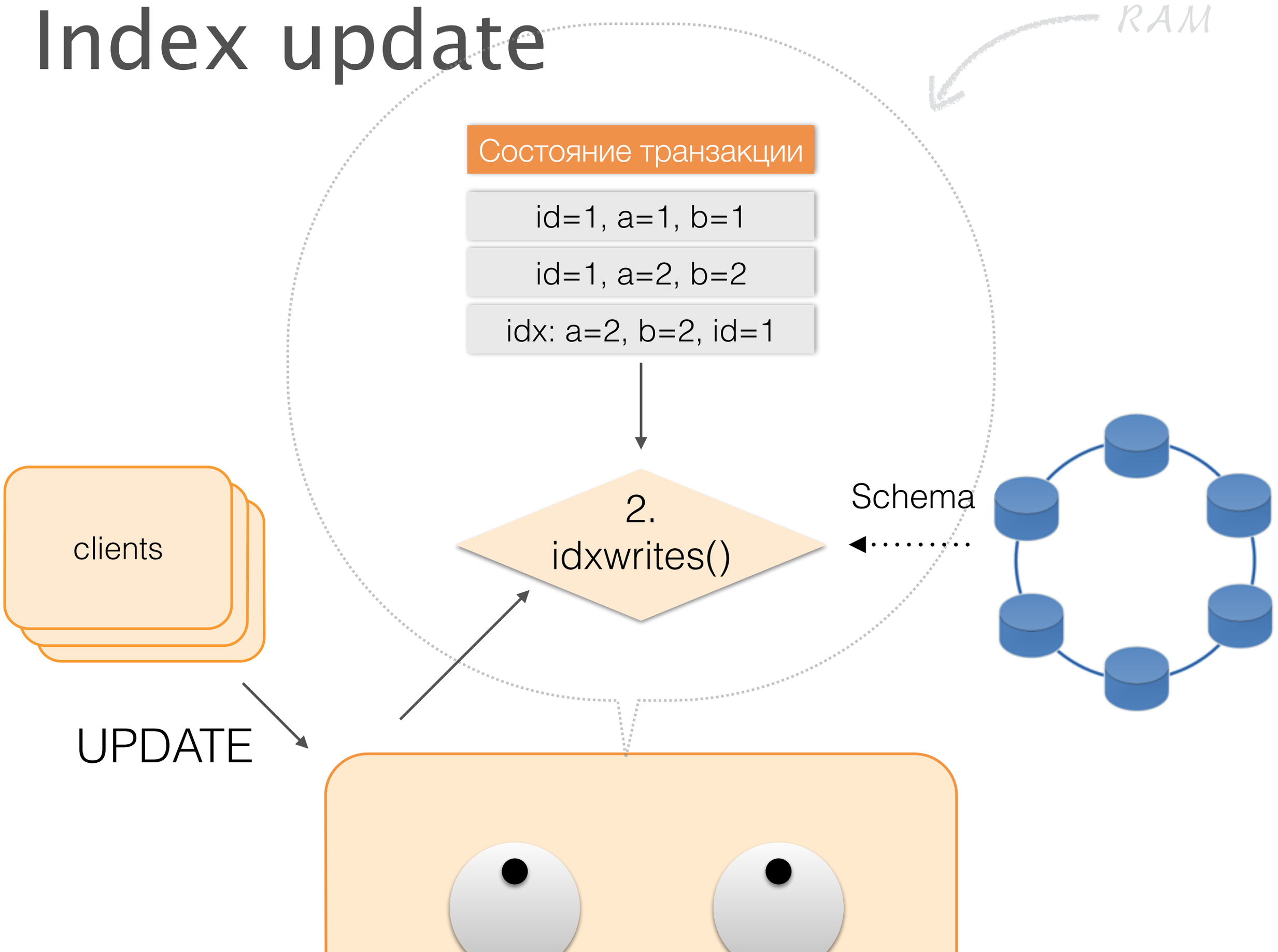
Primary Key

owner	modified	id	caption	access	…
111	9.10.2014	1	“kitty miau”	PUB	…
Partition Key	Clustering Key				

```
SELECT *
   WHERE owner=?
     AND modified>?
```

```
SELECT * FROM i1_test
   WHERE owner=?
     AND modified>?
```

Index update



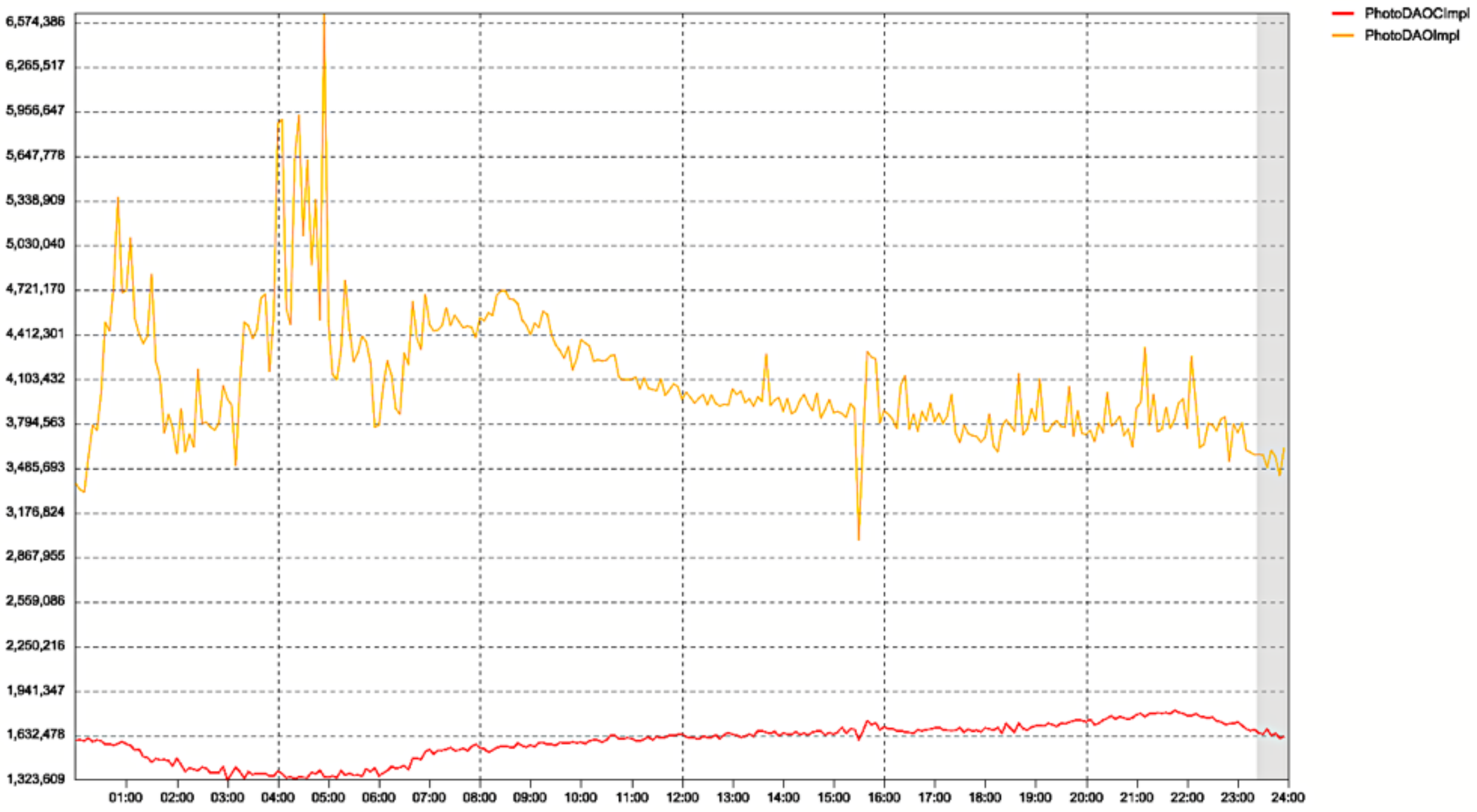
ACID

* Индексы “как в SQL”
  - Консистентные
  - По нескольким полям
  - Скалируемые и быстродействующие
  - Встроенные в язык
  - Без написания прикладного кода
  - Не влияющие на скорость записи 
* Генератор монотонных ид
  - Без дубликатов
  - SELECT nextval FROM seqName;

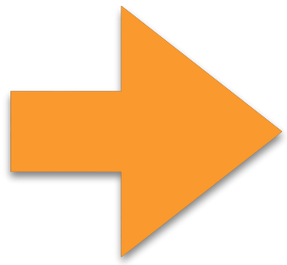
Фотки

* 11 миллиардов записей
* 80 K reads/sec, 2k-8k tx/sec
* SQL
  - RF=1 (RAID 10)
  - 32 MS SQL + 16 standby + 10 backup = 58
  - загружена на 100%
* После
  - RF=3 (в разных ЦОД)
  - 63 C* + 6 upd = 69, 1/3 по цене
  - загружена на 30%

Скорость



Фотки: результаты

* Ошибки: 8500 в день  1/100
* Длительность транзакции: <40ms
* Commit latency avg: <2ms
* Read, write, avg <2ms, 99% ~ 3ms

Что сделано в C*

* 21 патч в issues.apache.org
  - mainly range thombstone and queries fixes
* более быстрый batchlog manager (CASSANDRA-6134)
* более надежный always retry policy (CASSANDRA-6866)
* Night of the Living Dead (CASSANDRA-7872)
* Read repair of range tombstones (CASSANDRA-6863,CASSANDRA-8013)

одноклассники

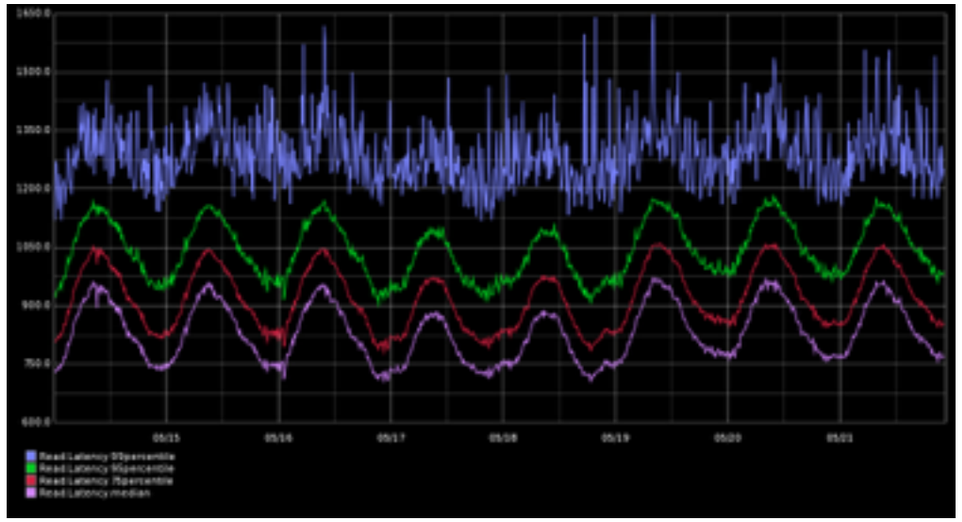
СПАСИБО !

Олег Анастасьев

oa@ok.ru
ok.ru/oa
@m0nstermind

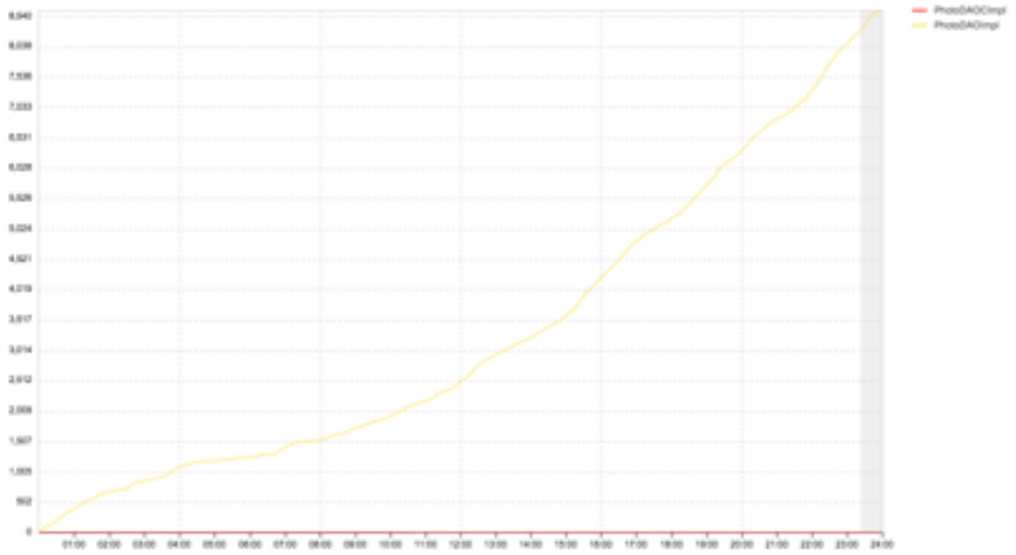
slideshare.net/m0nstermind

http://v.ok.ru

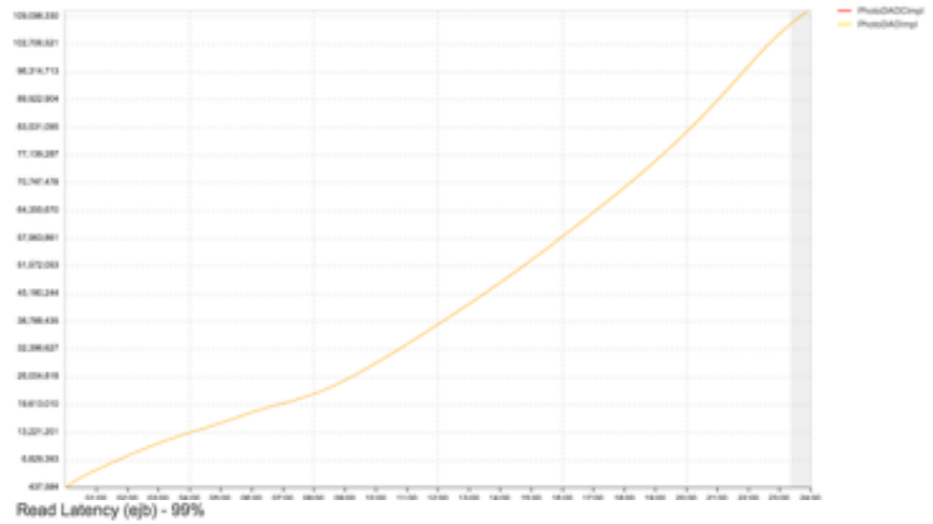


SQL vs C*One avg latency, writes, ns

SQL vs C*One failures, accumulative

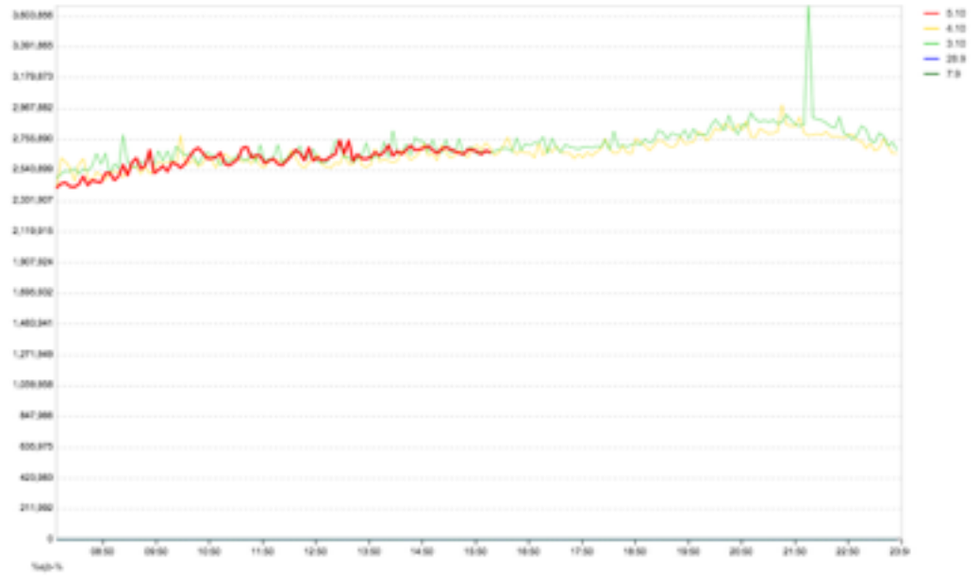


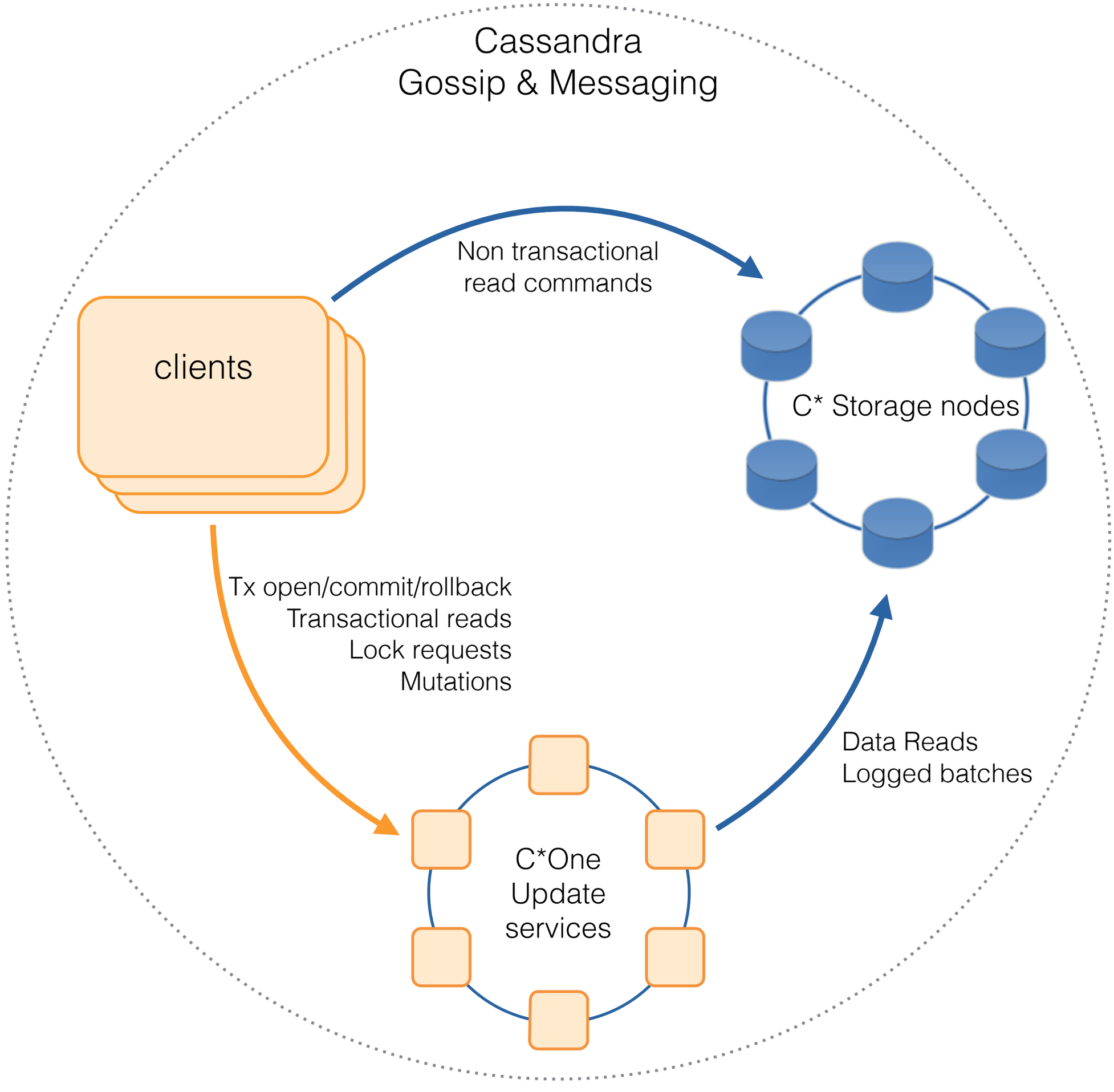
SQL vs C*One failures, accumulative



Write Latency (Proxy) - 99%

Read Latency (ejb) - 99%




Cassandra
Gossip & Messaging
Non transactional
read commands
clients
C* Storage nodes
Tx open/commit/rollback
Transactional reads
Lock requests
Mutations
Data Reads
Logged batches
C*One
Update
services

Почему не перкона

http://www.percona.com/doc/percona-xtradb-cluster/5.6/limitation.html

- Due to cluster level **optimistic concurrency control**, transaction issuing **COMMIT may still be aborted at that stage.** There can be two transactions writing to same rows and committing in separate Percona XtraDB Cluster nodes, and only one of the them can successfully commit. The failing one will be aborted. For cluster level aborts, Percona XtraDB Cluster gives back deadlock error code:

- **The write throughput of the whole cluster is limited by weakest node. If one node becomes slow, whole cluster is slow**. If you have requirements for stable high performance, then it should be supported by corresponding hardware.

Почему не MySQL Cluster

https://mariadb.com/sites/default/files/High_Availability_Solutions_for_the_MariaDB_and_MySQL_Database_-_MariaDB_White_Paper.pdf Page 13

- 2pc protocol
- A data node therefore always needs to communicate with other data nodes and SQL nodes over the network, which requires low network latency. **Spreading MySQL Cluster in two data centers connected in a WAN is therefore not recommended.** A MySQLdatabase can run with several data nodes, which allows distribution of the data for scalability and performance, while data is replicated for redundancy on up to four data nodes.

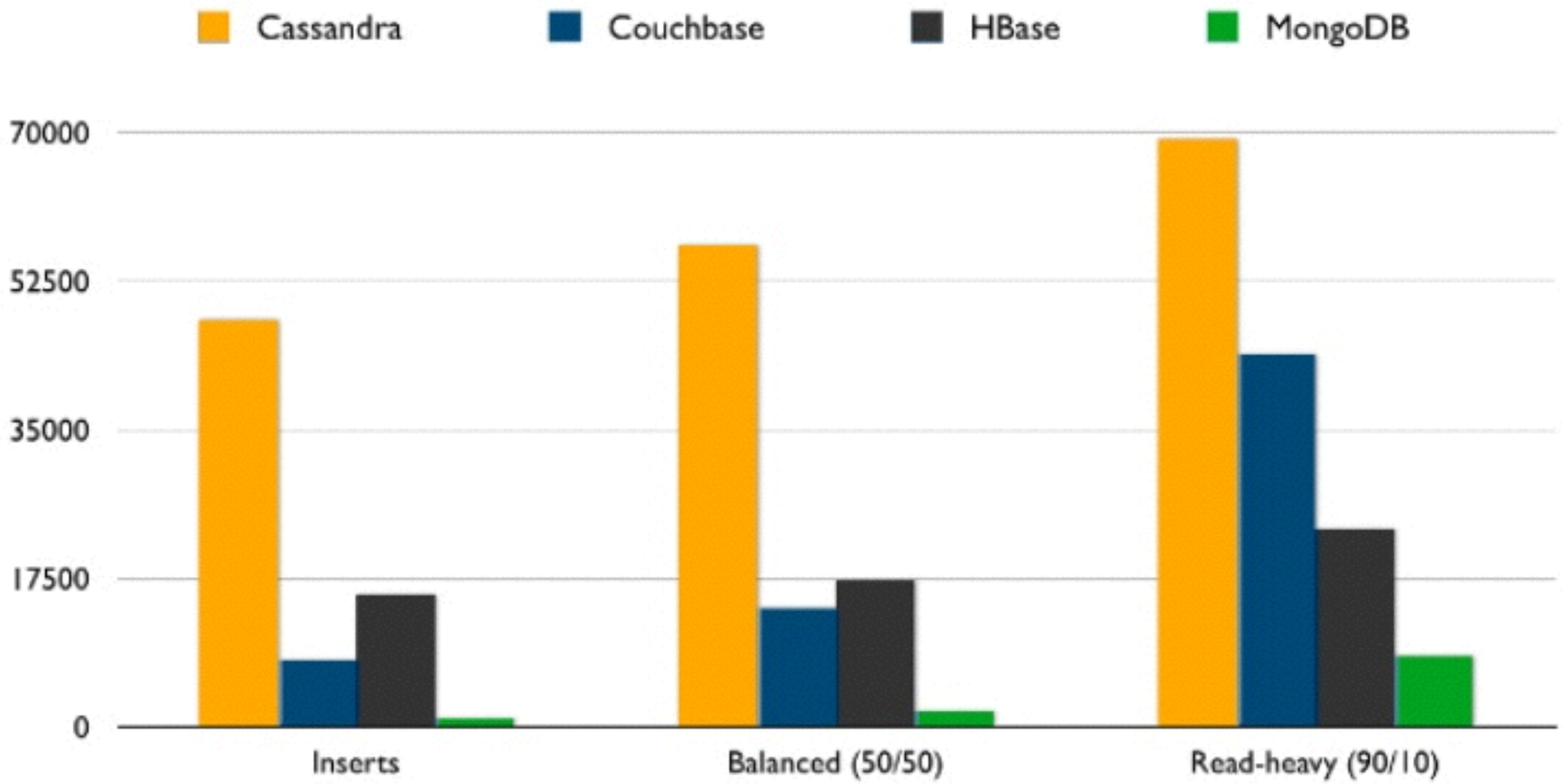
MariaDB Galera

https://mariadb.com/kb/en/mariadb/documentation/replication-cluster-multi-master/galera/mariadb-galera-cluster-known-limitations/

- Performance: by design **performance** of the cluster **cannot be higher than performance of the slowest node**; however, even if you have only one node, its performance can be considerably lower comparing to running the same server in a standalone mode (without wsrep provider). It is particularly true for big enough transactions (even those which are well within current limitations on transaction size quoted above).

- After a temporary split, if the 'good' part of the cluster was still reachable and its state was modified, **resynchronization occurs**. As a part of it, nodes of the 'bad' part of the cluster drop all client connections. **It might be quite unexpected**, especially if the client was idle and did not even know anything wrong was happening. Please also note that after the connection to the isolated node is restored, **if there is a flow on the node, it takes a long time for it to synchronize**, during which the "good" node says that the cluster is already of the normal size and synced, while the rejoining node says it's only joined (but not synced). The connections keep getting 'unknown command'. It should pass eventually.

Competitive performance



https://github.com/jbellis/YCSB