Формальная верификация алгоритмов репликации в распределенных системах хранения данных

Вадим Плахтинский

Распределенные системы

– фундамент масштабируемых веб-сервисов:

- Dynamo – shopping cart в Amazon
- BigTable – Gmail в Google
- Kafka – транспорт логов в LinkedIn

Как обеспечить корректность?

Стандартные подходы

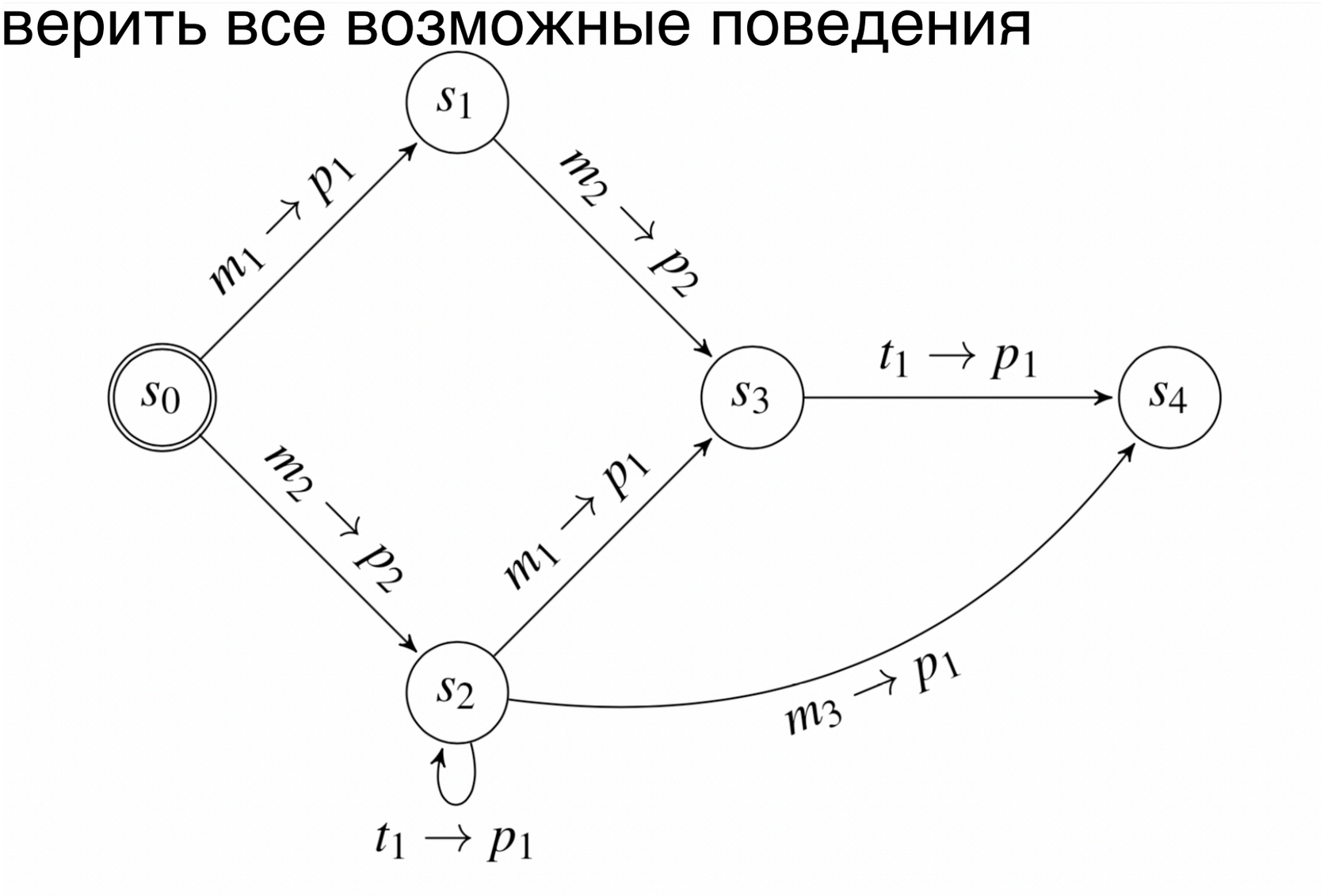
“We use deep design reviews, code reviews, static code analysis, stress testing, fault-injection testing, and many other techniques, but we still find that subtle bugs can hide in complex concurrent fault-tolerant systems. ...

We have found that testing the code is inadequate as a method to find subtle errors in design, as the number of reachable states of the code is astronomical. So we looked for a better approach”

– Инженеры AWS, 2014

Формальные методы

1. Построить модель системы (граф состояний) по спецификации (декларативному описанию)
2. Проверить все возможные поведения



Формальные методы

Спецификация системы – большая логическая формула

$$Spec \triangleq Init \wedge \Box[Next]_{vars}$$

```
Next ==
    \/ ControllerElectLeader
    \/ ControllerShrinkIsr
    \/ BecomeLeader
    \/ LeaderExpandIsr
    \/ LeaderShrinkIsr
    \/ LeaderWrite
    \/ LeaderIncHighWatermark
    \/ BecomeFollowerTruncateToHighWatermark
    \/ FollowerReplicate
```

```
ControllerElectLeader == \E newLeader \in quorumState.isr :
    /\ quorumState.leader # newLeader
    /\ ControllerUpdateIsr(newLeader, quorumState.isr)
    /\ UNCHANGED <<nextRecordId, replicaLog, replicaState>>
```

Применения в AWS (2014)

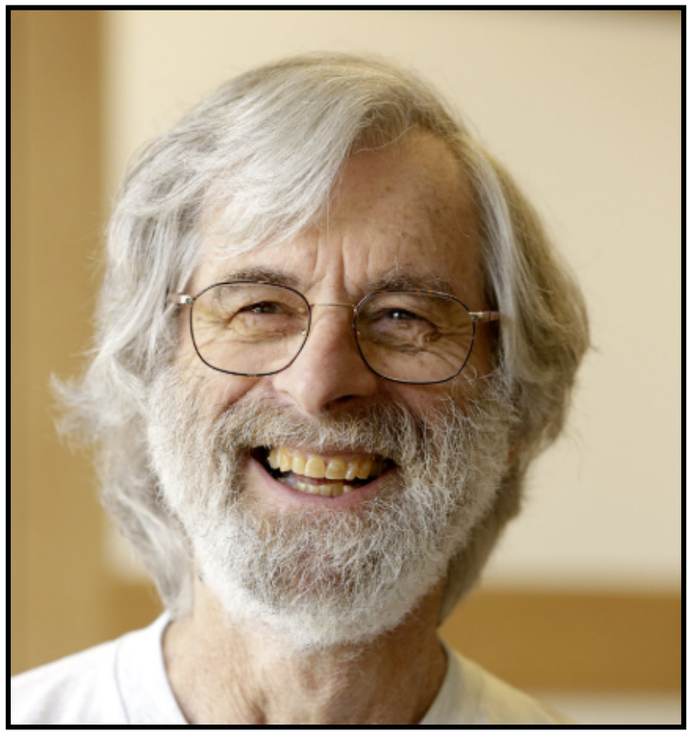
System	Components	Line count (excl. comments)	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 bugs. Found further bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 bug, and found a bug in the first proposed fix.
DynamoDB	Replication & group-membership system	939 TLA+	Found 3 bugs, some requiring traces of 35 steps
EBS	Volume management	102 PlusCal	Found 3 bugs.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA+	Found 1 bug. Verified an aggressive optimization.

Верификация всех слоев инфраструктуры!

Применения в индустрии

- 2017 – Яндекс, лок-фри аллокатор памяти
- 2018 – Apache Kafka, протокол репликации партиции
- 2018 – Microsoft Azure, модели согласованности для георасспределенной базы данных CosmosDB

Приемы моделирования систем на TLA+



Реактивность

Система – набор обработчиков событий

```
\* Defines how the variables may transition.
Next == /\ \/ \E i \in Server : Restart(i)
        \/ \E i \in Server : Timeout(i)
        \/ \E i,j \in Server : RequestVote(i, j)
        \/ \E i \in Server : BecomeLeader(i)
        \/ \E i \in Server, v \in Value : ClientRequest(i, v)
        \/ \E i \in Server : AdvanceCommitIndex(i)
        \/ \E i,j \in Server : AppendEntries(i, j)
        \/ \E m \in DOMAIN messages : Receive(m)
        \/ \E m \in DOMAIN messages : DuplicateMessage(m)
        \/ \E m \in DOMAIN messages : DropMessage(m)
        \* History variable that tracks every log ever:
     /\ allLogs' = allLogs \cup {log[i] : i \in Server}
```

Асинхронность

- Нет границы на время доставки сообщений
- Дрейф часов
- Разная скорость реакции реплик

Подходит для тестирования safety-свойств (SafeValue в Paxos, ElectionSafety в Raft)

Не получится проверять liveness-свойства (теорема FLP).

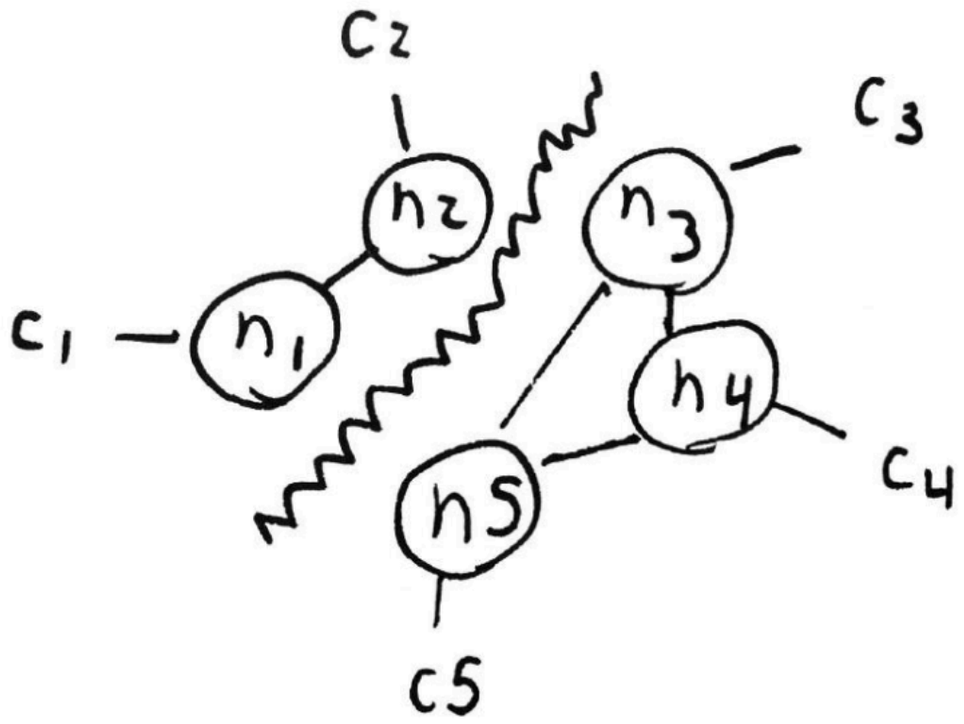
Моделирование сети

```
VARIABLE msgs  \* The set of all messages that have been sent.

Send(m) == msgs' = msgs \cup {m}


Phase1a(b) == /\ Send([type |-> "1a", bal |-> b])
              /\ UNCHANGED <<maxBal, maxVBal, maxVal>>

Phase1b(a) == /\ \E m \in msgs :
                  /\ m.type = "1a"
                  /\ m.bal > maxBal[a]
                  /\ maxBal' = [maxBal EXCEPT ![a] = m.bal]
                  /\ Send([type |-> "1b", acc |-> a, bal |-> m.bal,
                           mbal |-> maxVBal[a], mval |-> maxVal[a]])
              /\ UNCHANGED <<maxVBal, maxVal>>
```

Сбои сети

- Произвольный порядок доставки сообщений, задержки
- Потери и дублирование сообщений
- Партишены в сети

Jepsen vs msgs



Сбои узлов

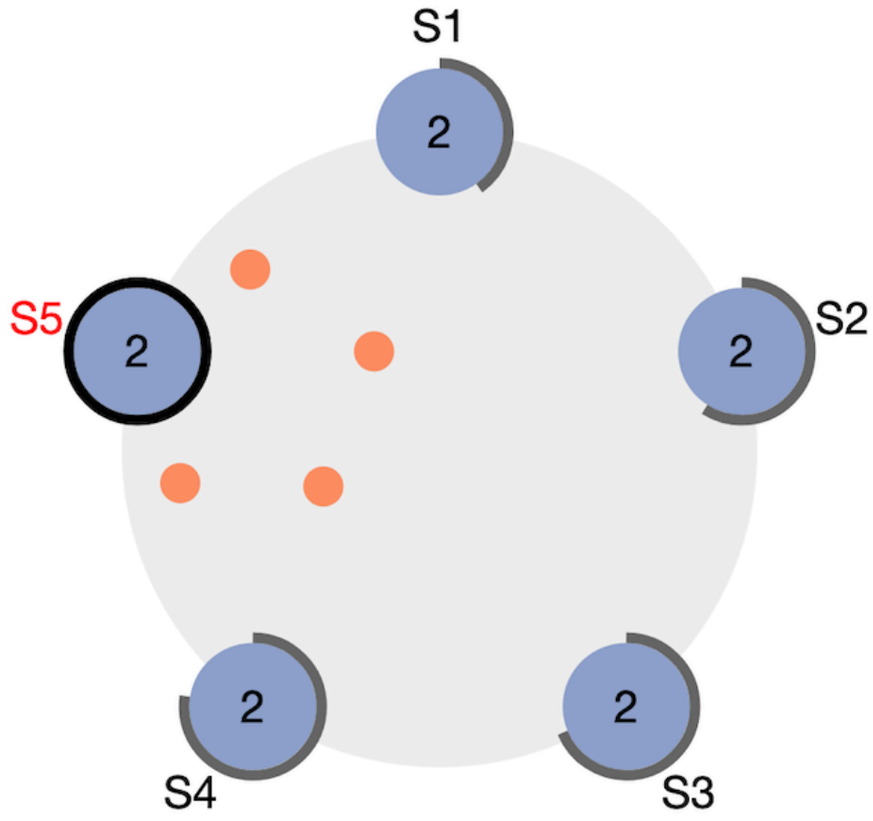
Отказы – неотличимы от асинхронности

Рестарты:

```
Restart(i) ==
    /\ state'          = [state EXCEPT ![i] = Follower]
    /\ votesResponded' = [votesResponded EXCEPT ![i] = {}]
    /\ votesGranted'   = [votesGranted EXCEPT ![i] = {}]
    /\ voterLog'       = [voterLog EXCEPT ![i] = [j \in {} |-> <<>>]]
    /\ nextIndex'      = [nextIndex EXCEPT ![i] = [j \in Server |-> 1]]
    /\ matchIndex'     = [matchIndex EXCEPT ![i] = [j \in Server |-> 0]]
    /\ commitIndex'    = [commitIndex EXCEPT ![i] = 0]
    /\ UNCHANGED <<messages, currentTerm, votedFor, log, elections>>
```

currentTerm, votedFor, log – в персистентном хранилище

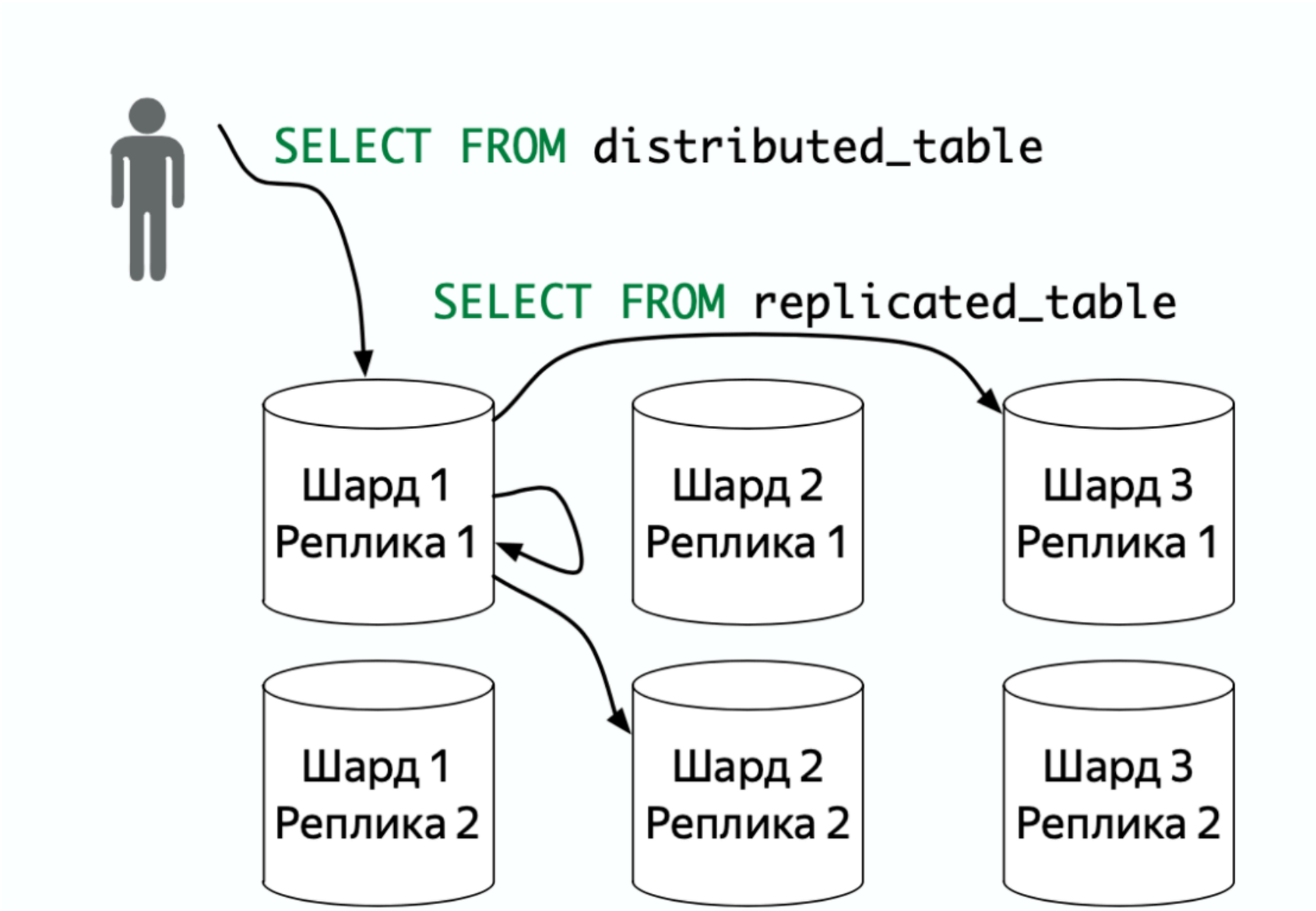
Таймеры и таймауты




```
Timeout(i) == /\ state[i] \in {Follower, Candidate}
              /\ state' = [state EXCEPT ![i] = Candidate]
              /\ currentTerm' = [currentTerm EXCEPT ![i] = currentTerm[i] + 1]
              \* Most implementations would probably just set the local vote
              \* atomically, but messaging localhost for it is weaker.
              /\ votedFor' = [votedFor EXCEPT ![i] = Nil]
              /\ votesResponded' = [votesResponded EXCEPT ![i] = {}]
              /\ votesGranted'   = [votesGranted EXCEPT ![i] = {}]
              /\ voterLog'       = [voterLog EXCEPT ![i] = [j \in {} |-> <<>>]]
              /\ UNCHANGED <<messages, leaderVars, logVars>>
```

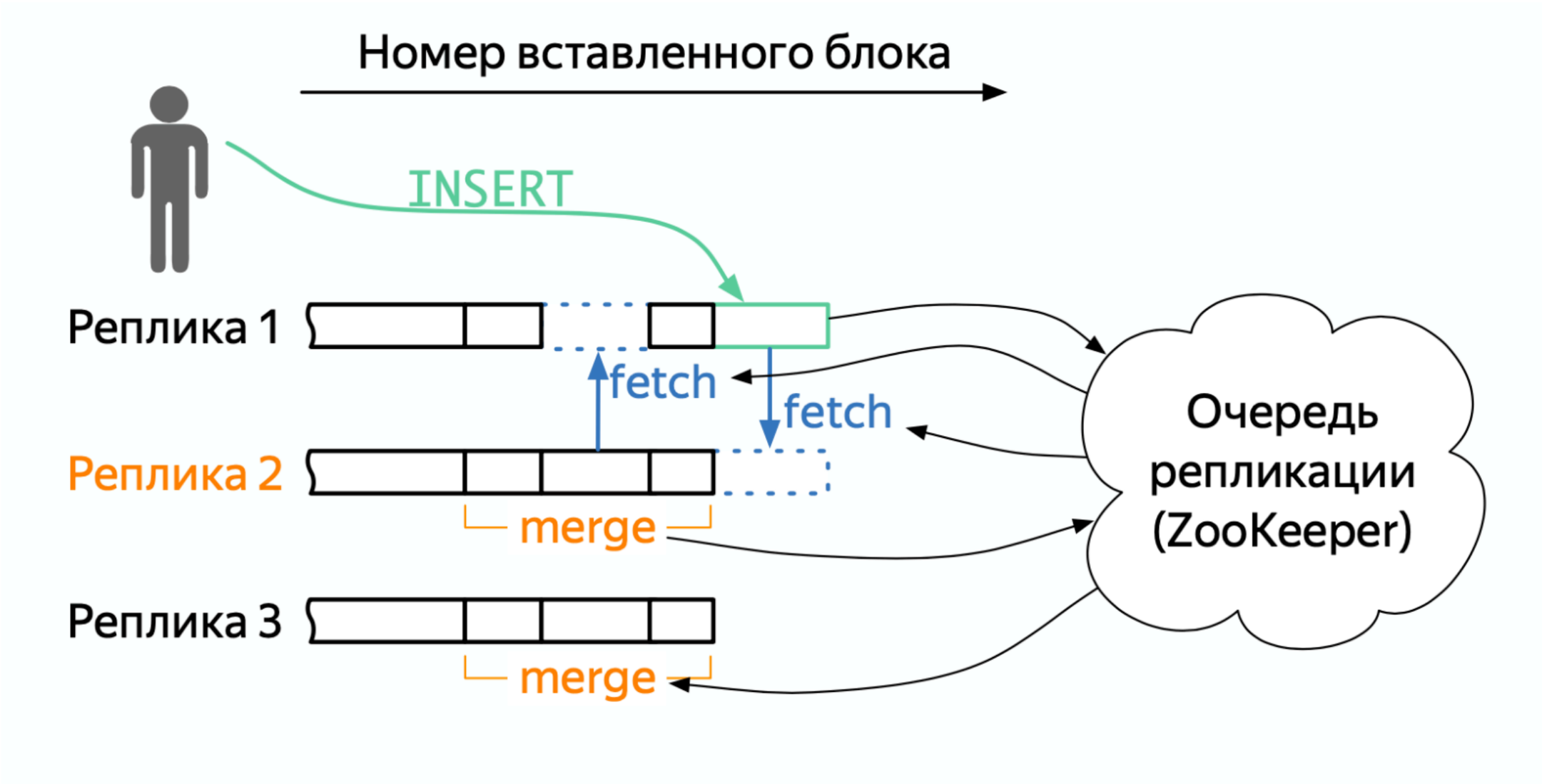
Верификация БД Кликхаус

Распределенные таблицы в KX



Движок Distributed

Репликация шарда в KX

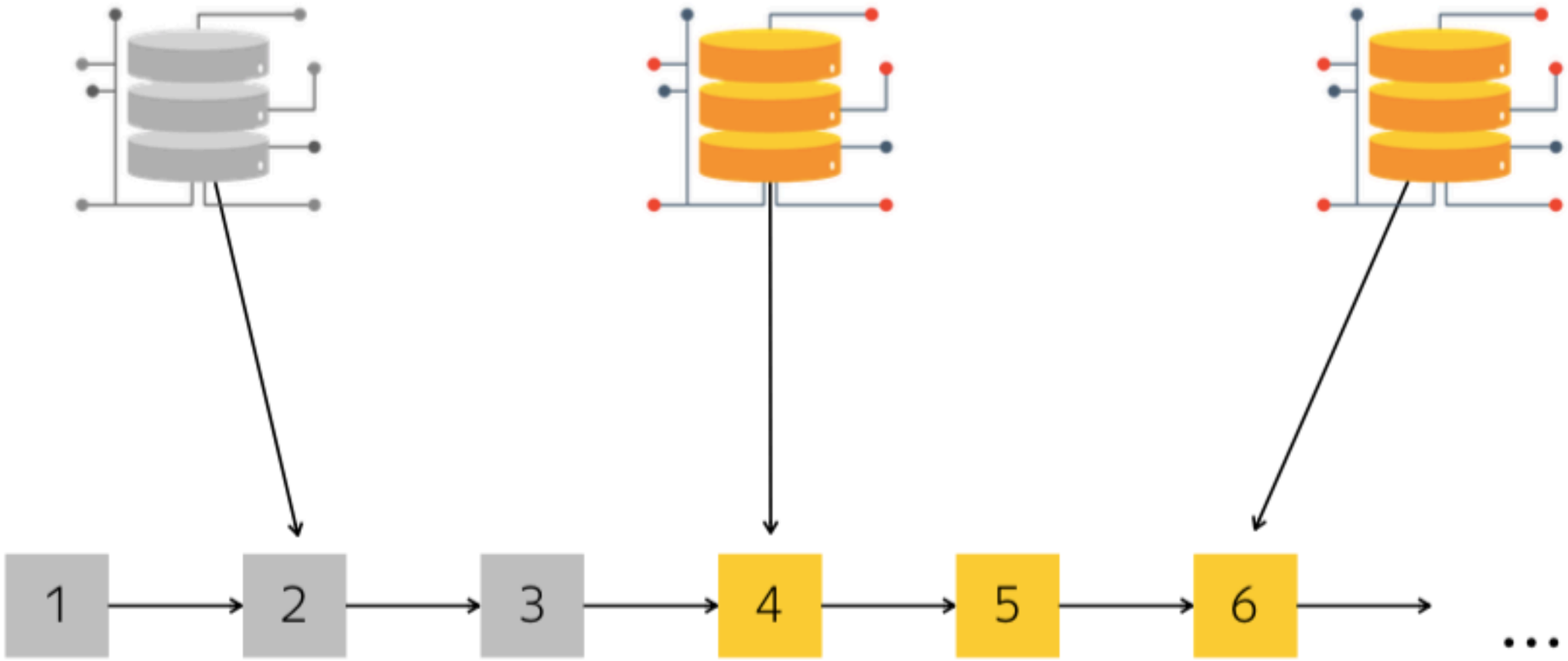


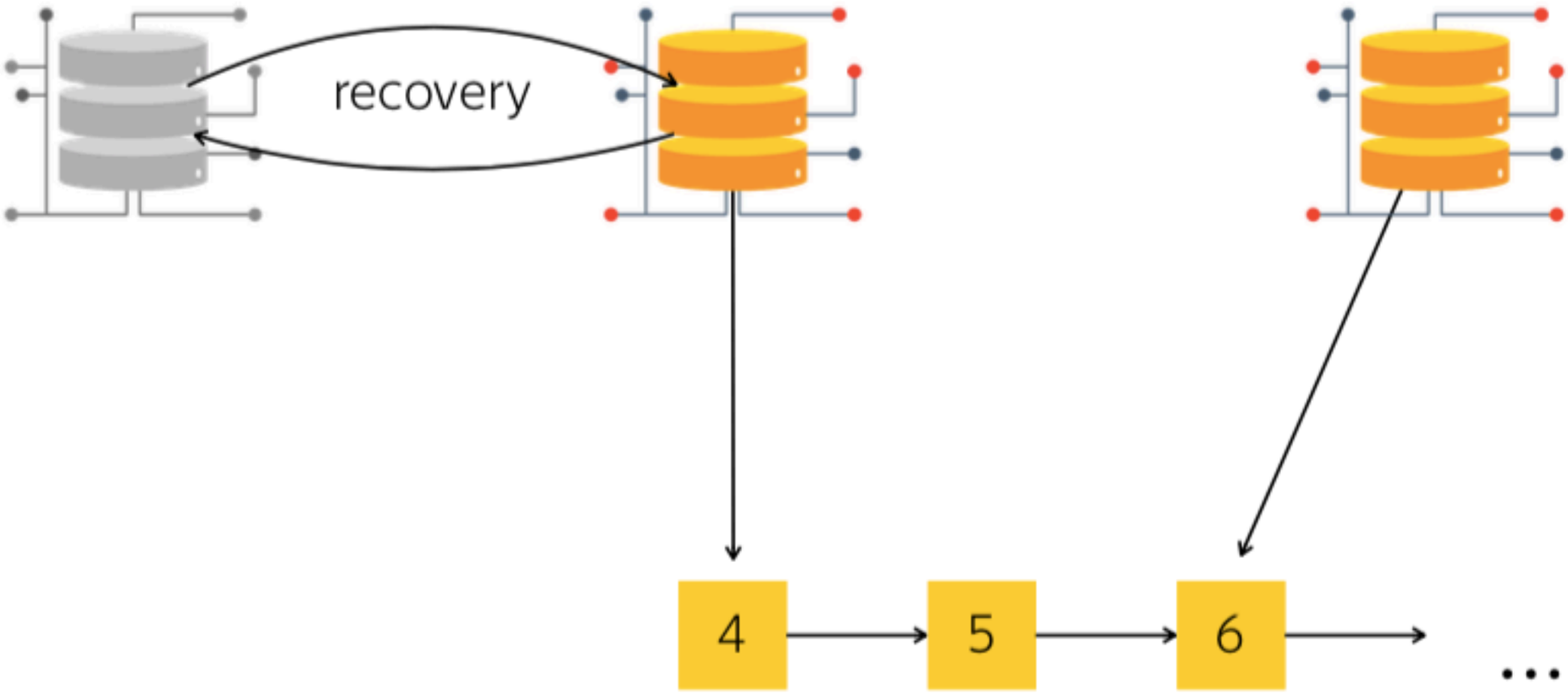
Движок ReplicatedMergeTree

Что верифицировал

- Обрезка лога апдейтов в ZooKeeper
- Кворумные вставки и sequentially consistent селекты

Обрезка лога




recovery
4
5
6
...

Кворумные вставки

insert_quorum ¶

Включает кворумную запись.

Кворумная запись

`INSERT` завершается успешно только в том случае, когда ClickHouse смог без ошибки записать данные в `insert_quorum` реплик за время `insert_quorum_timeout`. Если по любой причине количество реплик с успешной записью не достигнет `insert_quorum`, то запись считается не состоявшейся и ClickHouse удалит вставленный блок из всех реплик, куда уже успел записать данные.

Все реплики в кворуме консистентны, т.е. содержат данные всех более ранних запросов `INSERT`. Последовательность `INSERT` линеаризуется.

При чтении данных, записанных с `insert_quorum` можно использовать настройку select_sequential_consistency.

Seq Cst чтения

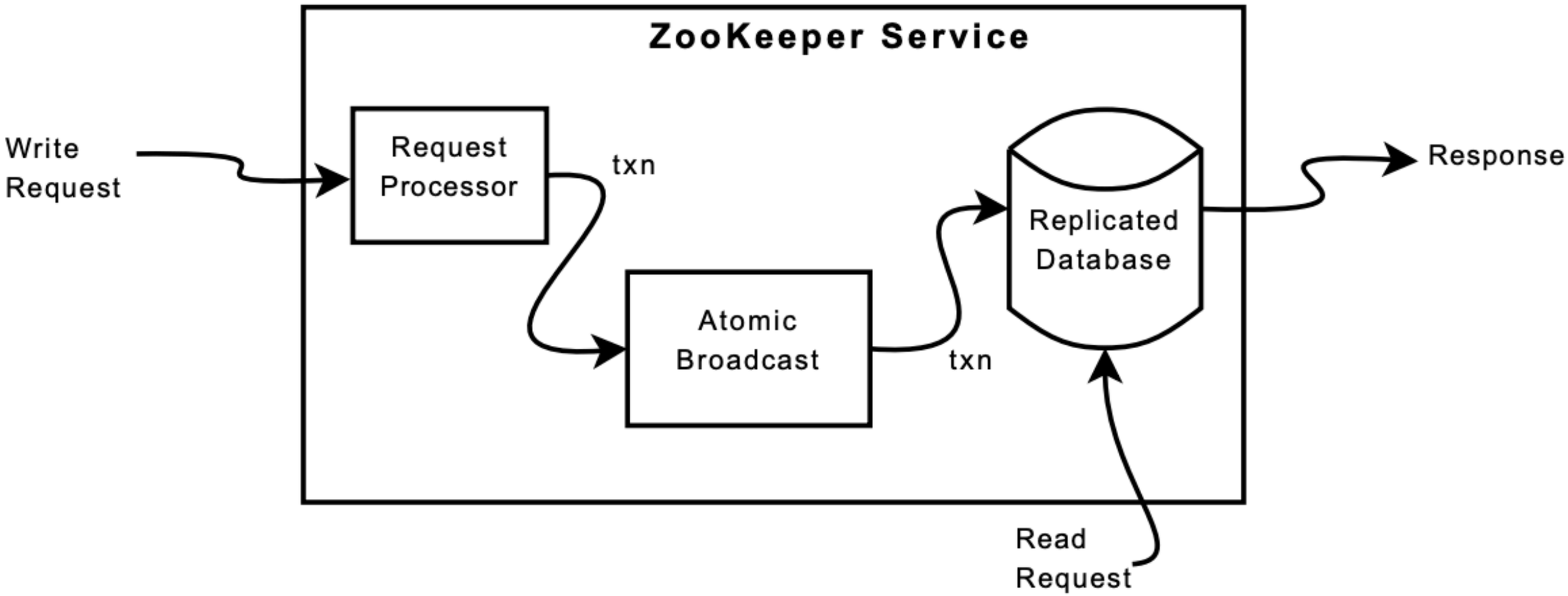
select_sequential_consistency ¶

Включает или выключает последовательную консистентность для запросов `SELECT`.

Использование

Когда последовательная консистентность включена, то ClickHouse позволит клиенту выполнить запрос `SELECT` только к тем репликам, которые содержат данные всех предыдущих запросов `INSERT`, выполненных с `insert_quorum`. Если клиент обратится к неполной реплике, то ClickHouse сгенерирует исключение. В запросе SELECT не будут участвовать данные, которые ещё не были записаны на кворум реплик.

Нарушение гарантий для селектов



history	<<[type |-> "StartInsert", record_id |-> A_1], [type |-> "EndInsert", record...
▶	[type |-> "StartInsert", record_id |-> A_1]
▶	[type |-> "EndInsert", record_id |-> A_1]
▶	[type |-> "StartInsert", record_id |-> A_2]
▶	[type |-> "EndInsert", record_id |-> A_2]
▶	[type |-> "Read", record_id |-> A_2]
▶	[type |-> "Read", record_id |-> A_1]

Спасибо за внимание!

https://github.com/VadimPlh/Arrival

@vadimplh