

СИСТЕМЫ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Стек технологий



К.Т.Н.
Папулин Сергей Юрьевич
rapulin_bmstu@mail.ru

Лекция 6. Распределенная координация. Zookeeper

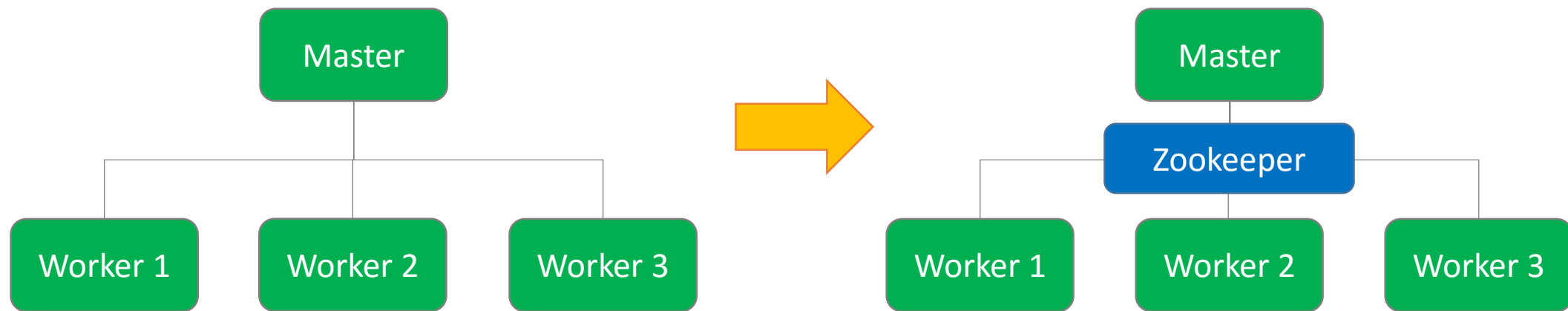


- Особенности Zookeeper
- Архитектура
- Чтение/запись
- Пространство имен Zookeeper
- Отслеживание изменений (watch)
- Производительность

Назначение и особенности Zookeeper

Zookeeper – координирующий сервис для распределенных приложений. Позволяет распределённым процессам координировать свои действия друг с другом

Предоставляет набор примитивов для реализации распределенного взаимодействия



- Высокая пропускная способность (распределенная архитектура)
- Малые задержки (данные в оперативной памяти)
- Высокая надежность (репликации, выборы лидера)
- Строго упорядоченный доступ (идентификаторы операций)

- znode
- watch
- ensemble
- leader/follower
- Packet/proposal/message

- Последовательная согласованность (Sequential Consistency):
изменения от клиента будут применены по порядку, в котором они были отправлены серверу
- Атомарность (Atomicity):
изменения либо применяются целиком, либо нет. Невозможно частичное изменение
- Надежность (Reliability):
Если изменения были применены, то они сохранятся до следующего изменения клиентом
- X (Timeless):
Клиент видит систему гарантированно актуальной в пределах определенного временного интервала

Назначение и особенности Zookeeper

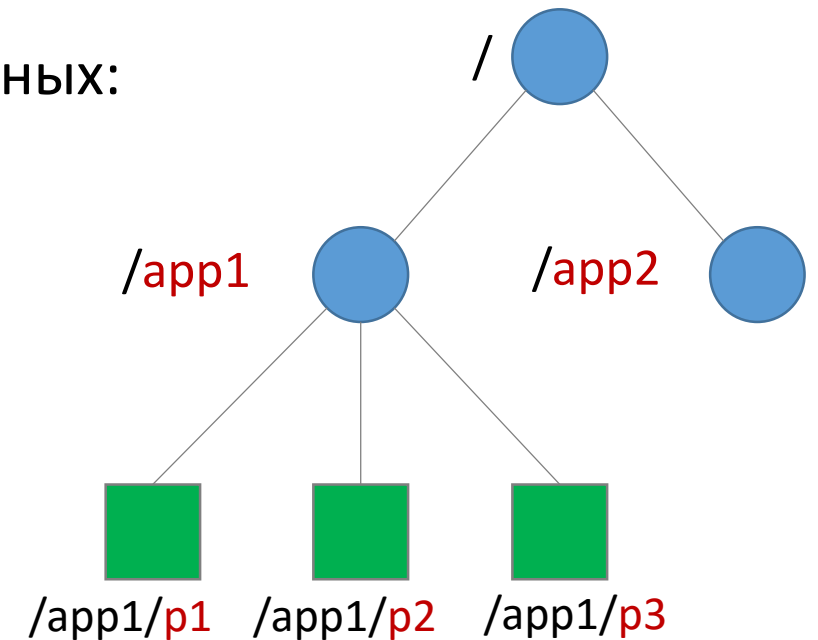
- **Zookeeper** поддерживает разделяемое иерархическое пространство имен
- пространство имен состоит из регистров данных – **znode**
- **znode** может по аналогии с файловой системой выполнять одновременно роль файла и папки
- данные хранятся в оперативной памяти
- данные копируются на множество узлов, называемых ансамблем (**ensemble**)
- Zookeeper больше ориентирован на чтение, чем на запись в пропорции 10:1
- Клиенты – тысячи узлов

znode

➤ Каждый узел в пространстве имен может одновременно иметь данные, ассоциируемые с ним, и дочерние znode'ы

➤ Используется для хранения координирующих данных:

- Информацию о статусах
- Конфигурации
- Информацию о размещении



➤ Данные каждого узла, как правило, небольшого размера (В - KB)

Данные **znode** читаются и записываются атомарно:

- при чтении можно получить только все данные **znode**
- при записи все данные **znode** заменяются новыми

Эфемерный (ephemeral) **узел** – znode, который существует на протяжении активной сессии, создавшей его. Когда сессия завершается, узел удаляется

Параметры **znode**

czxid – zxid при создании

mzxid – zxid последнего изменения

ctime – время в миллисекундах от эпохи создания **znode**

mtime – от последнего изменения

version – количество изменений данных **znode**

cversion – количество изменений дочерних узлов

aversion – количество изменений в acl

ephemeralOwner – id сессии владельца **znode**, если это эфемерный **znode**. Если нет, то 0

dataLength – размер данных **znode**

numChildren – количество дочерних **znode**'ов

Пример распределённого приложения

➤ Выбор мастера (Master election)

Назначение задач worker'ам

➤ Обнаружение отказа (Crash detection)

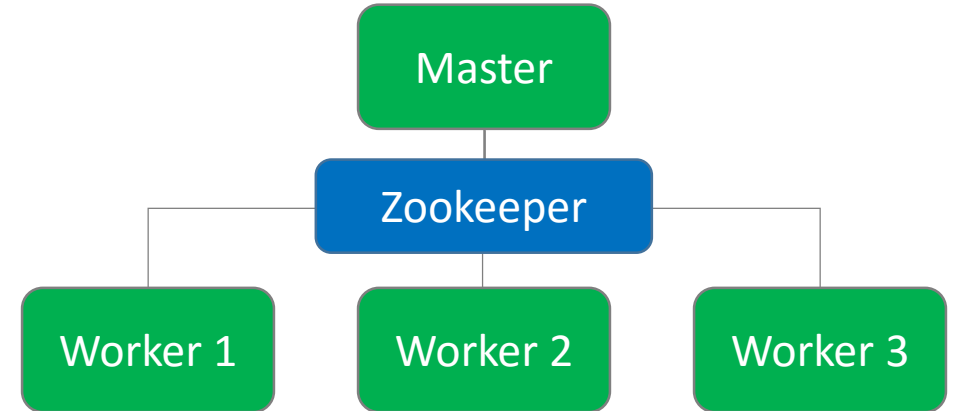
Мастер должен уметь определять, что worker вышел из строя или с ним потеряно соединение

➤ Управление группами (Group membership management)

Мастер должен знать, какие worker'ы доступны для выполнения задач

➤ Управление метаданными (Metadata management)

Мастер и worker'ы должны хранить задания и статусы выполнения надёжным способом



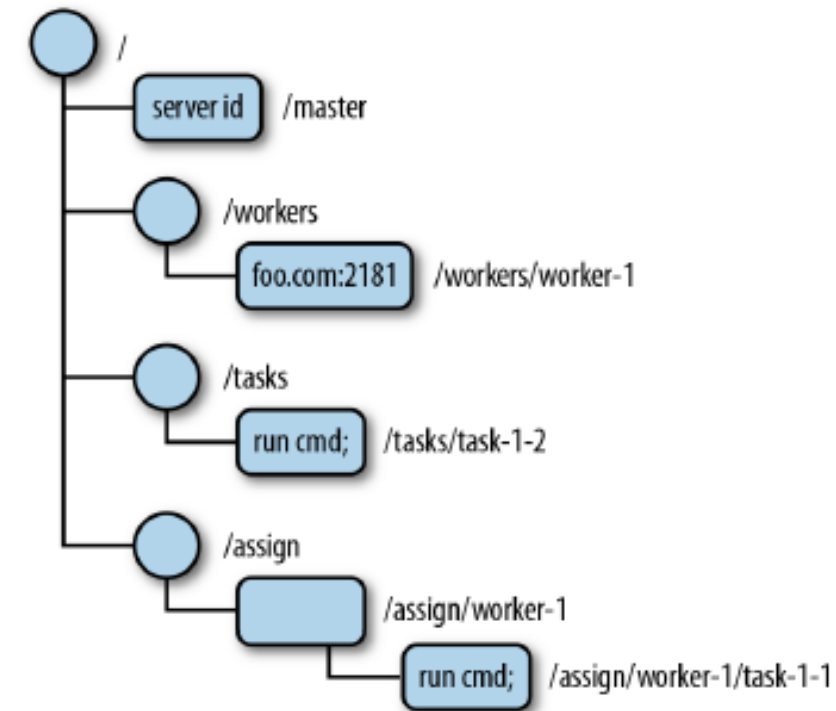
Пример распределённого приложения

/master znode содержит данные о мастере

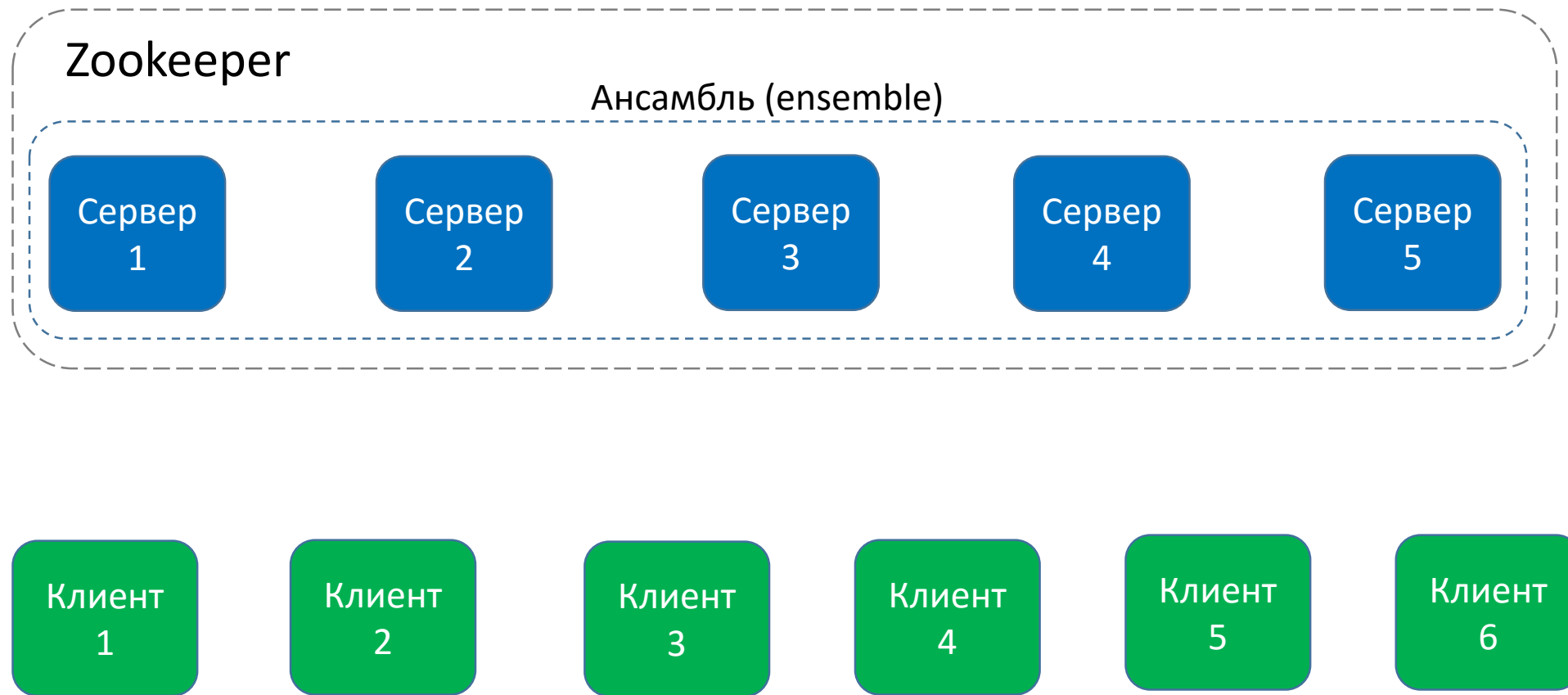
/workers znode – родительский znode для всех znode'ов, соответствующих доступным worker'ам в системе. Если worker становится недоступным, то znode должен быть удален из /workers

/tasks znode – родительский znode для всех созданных задач, ожидающих выполнения на worker'ах

/assign znode – родительский znode для распределённых по worker'ам задач

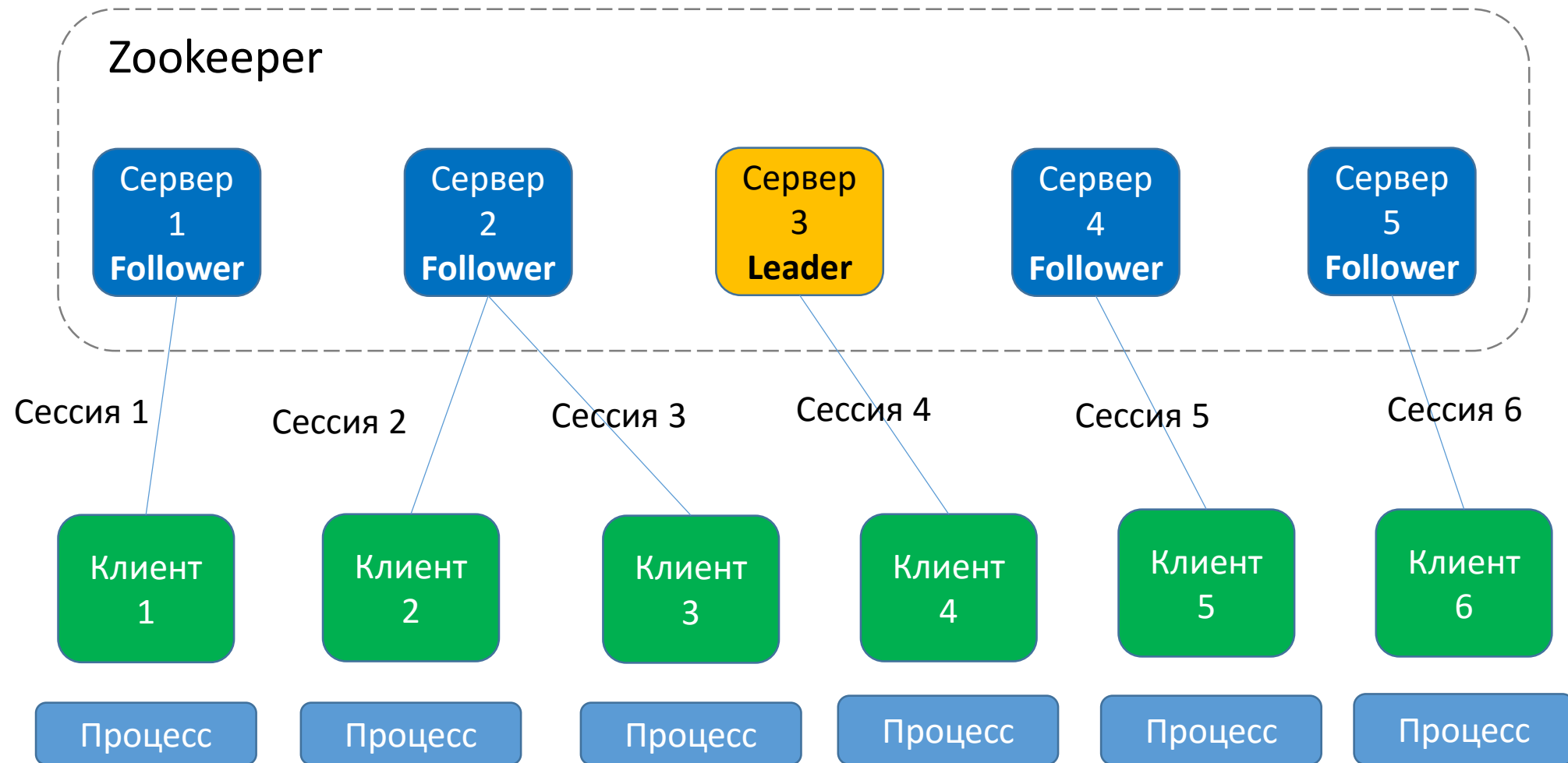


Архитектура Zookeeper



- Все серверы Zookeeper знают друг о друге
- Поддерживают:
 - Образ состояния (в оперативной памяти)
 - Логи транзакций (в постоянной памяти)
 - Снепшот (в постоянной памяти)

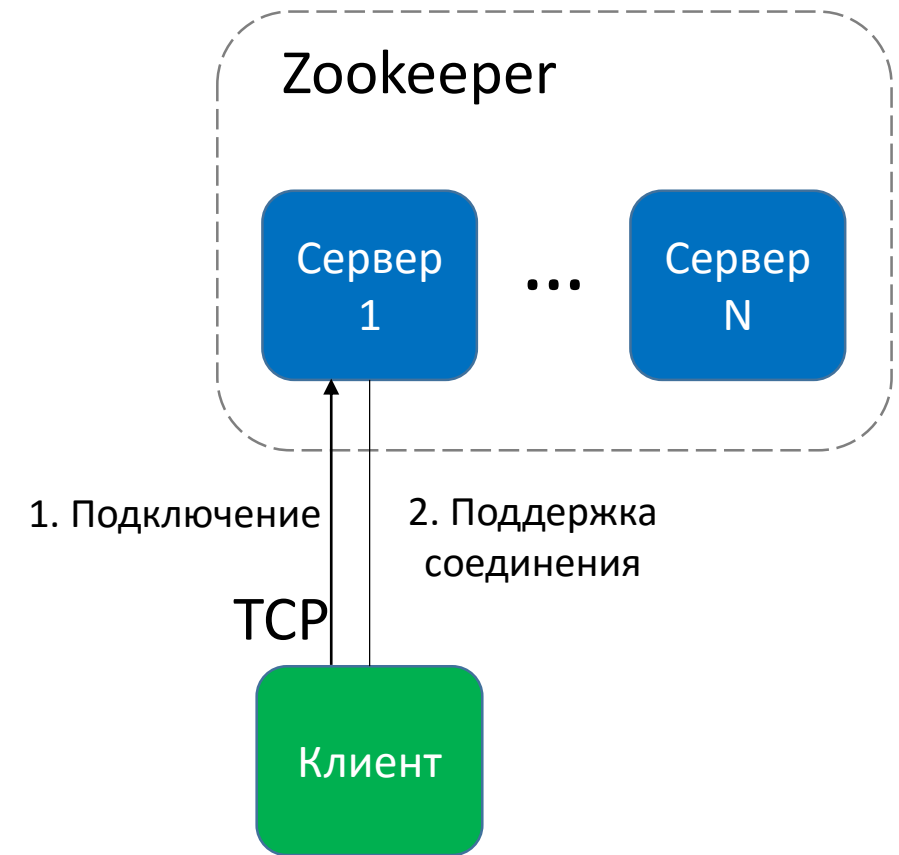
Архитектура Zookeeper. Типы серверов



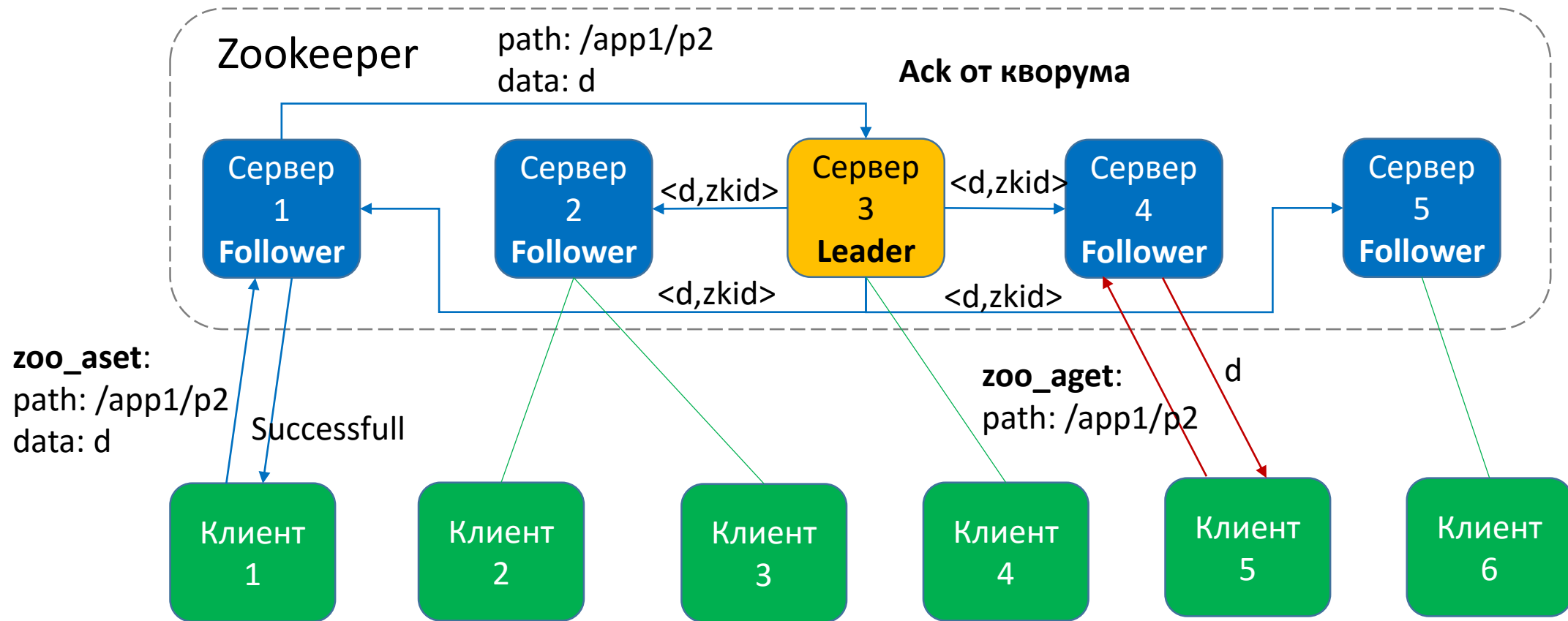
Распределенное приложение

Соединения между Клиентом и Сервером используется для:

- Отправки запросов
- Получения ответов
- Отслеживания событий
- Отправки heartbeat



Архитектура Zookeeper. Запись/Чтение



Свойства Атомарной системы сообщений (Atomic Broadcast)



Надежная доставка (Reliable delivery)

Если сообщение m доставлено одним сервером, оно будет доставлено в конечном счете всеми серверами

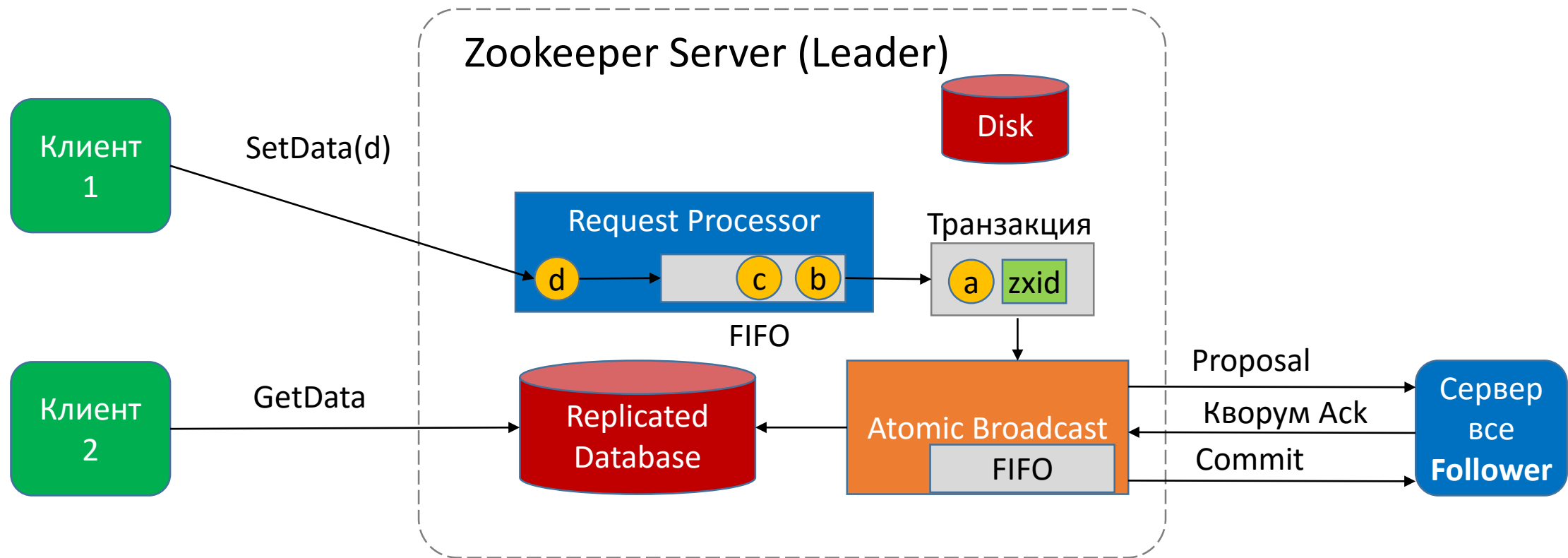


Total order



Casual order

Архитектура Zookeeper. Сервер



Replicated Database – база данных всего дерева пространства имен (в оперативной памяти)

Свойства Атомарной системы сообщений (Atomic Broadcast)



Packet

Последовательность байтов, отправленных через FIFO канал



Proposal

Единица согласования. Proposal обмениваются пакетами с кворумом серверов. Proposal, как правило, содержат message (кроме NEW_LEADER)



Message

Последовательность байтов, которая атомарно распространяется на все серверы. Message кладется в proposal и согласуется перед доставкой

Свойства Атомарной системы сообщений (Atomic Broadcast)

Обмен сообщениями состоит из двух фаз:

- Активация лидера (Leader activation)
Лидер устанавливает корректное состояние системы и готов начать создавать proposal
- Активный обмен сообщениями (Active messaging)
Лидер берет messages, чтобы предложить и координировать доставку сообщений

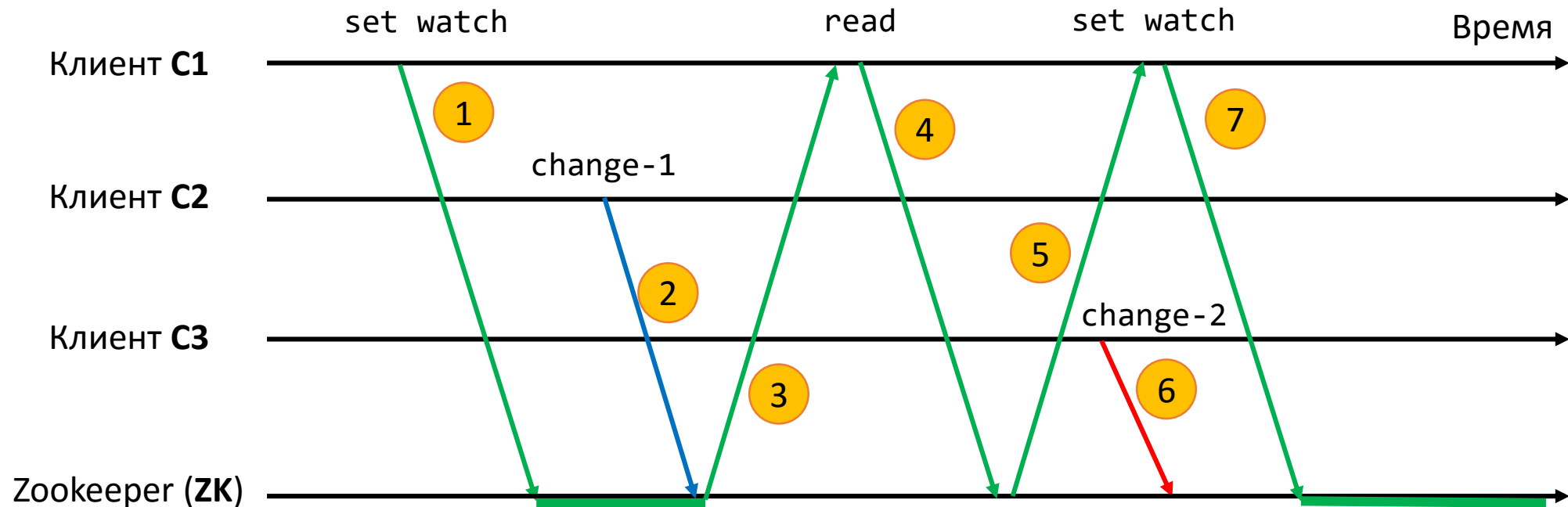
Zookeeper поддерживает следующие операции:

- **Create** создает узел (**znode**) в пространстве имен
- **Delete** удаляет узел
- **Exists** проверяет существует ли узел по пути
- **Get data** читает данные из узла
- **Set data** записывает данные в узел
- **Get children** возвращает список дочерних узлов
- **Sync** ожидает пока данные будут распределены между Zookeeper серверами

- CREATE: можно создавать дочерние узлы
- READ: можно получить данные от узла и список детей
- WRITE: можно записывать данные в узел
- DELETE: можно удалять дочерние узлы
- ADMIN: можно устанавливать разрешения

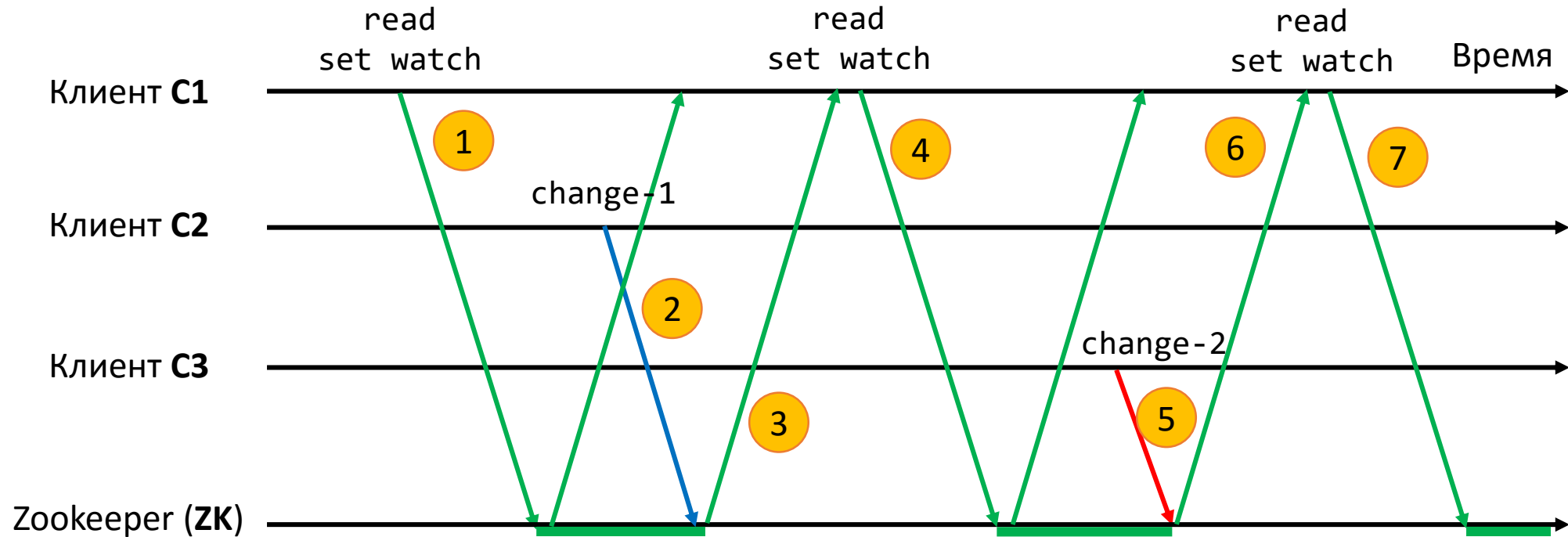
- Клиент может установить наблюдение (watch) на **znode**
- Если **znode** изменится, то клиент получит уведомление
- После уведомления **watch** сбрасывается
- Также **watch** уведомляет клиента, если потеряно соединение с сервером Zookeeper (на стороне клиента)

Watch на изменение данных. Проблема



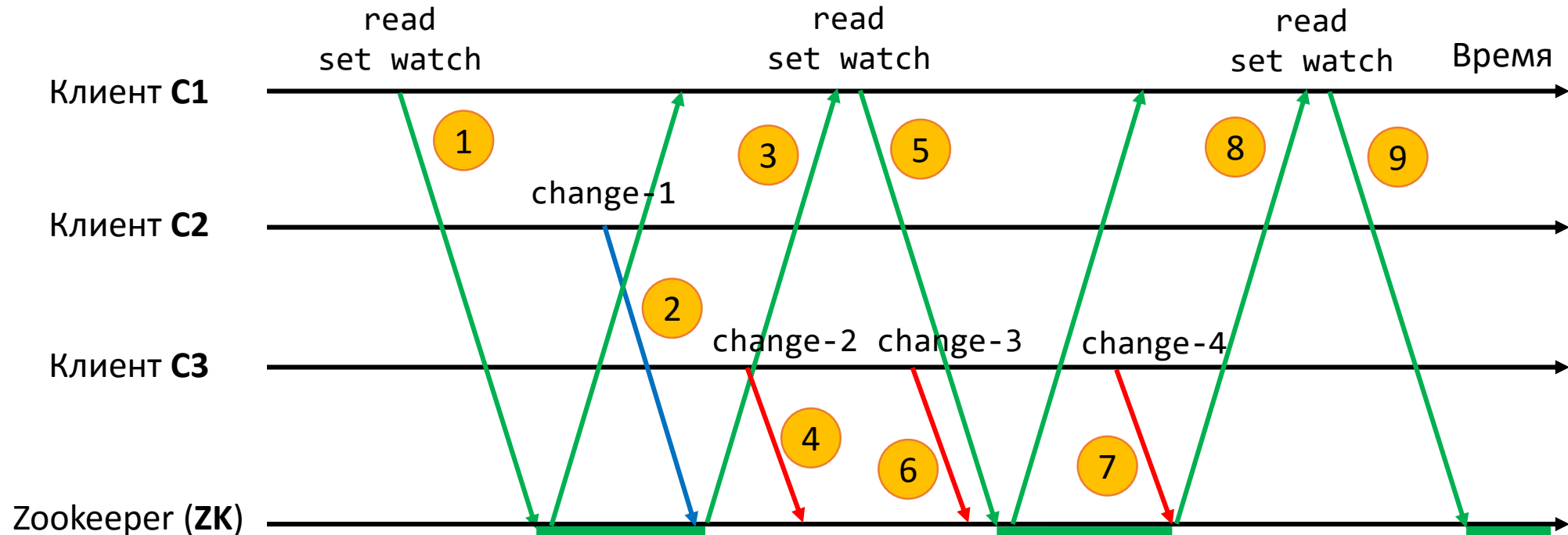
C1 не узнает об изменении **change-2**

Watch на изменение данных. Решение



C1 узнает об изменении **change-2**

Watch на изменение данных. Решение



- **C1** не получит уведомления об изменениях **change-2** и **change-3**
- однако **C1** получит состояние после внесения изменения **change-3** при чтении (шаг 5)

Watch устанавливаются вместе с операциями **getData()**, **getChildren()** и **exists()**

WatchedEvent

- KeeperState
- EventType
- path

EventType

- None (-1),
- NodeCreated (1) (exists),
- NodeDeleted (2) (exists, getData),
- NodeDataChanged (3) (exists, getData),
- NodeChildrenChanged (4) (getChildren);

KeeperState

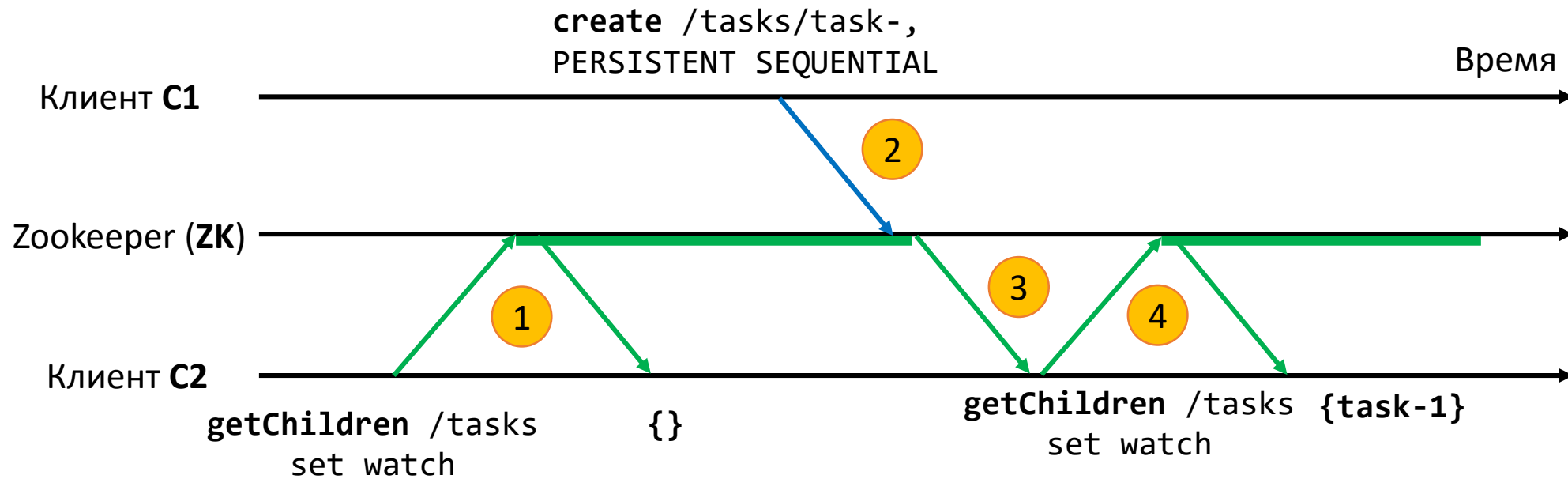
- Unknown (-1)
- Disconnected (0)
- NoSyncConnected (1)
- SyncConnected (3)
- AuthFailed (4)
- ConnectedReadOnly (5)
- SaslAuthenticated (6);
- Expired (-112);

WatcherType

- Children
- Data
- Any

<https://github.com/apache/zookeeper/tree/master/zookeeper-server/src/main/java/org/apache/zookeeper>

Watch на изменение данных. getChildren()

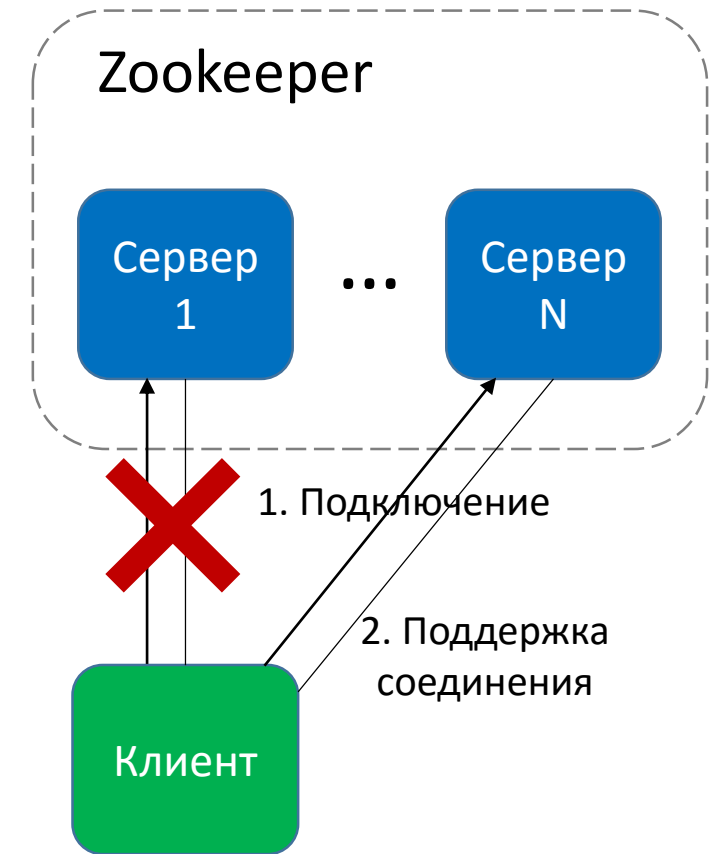


- 1 **C2** запрашивает список заданий + ставит **watch** на изменения. Получает список (пустой)
- 2 **C1** создает задание
- 3 Срабатывает установленный **watch** и **C1** получает уведомление об изменении
- 4 **C2** запрашивает список заданий + ставит **watch** на изменения. Видит новое задание

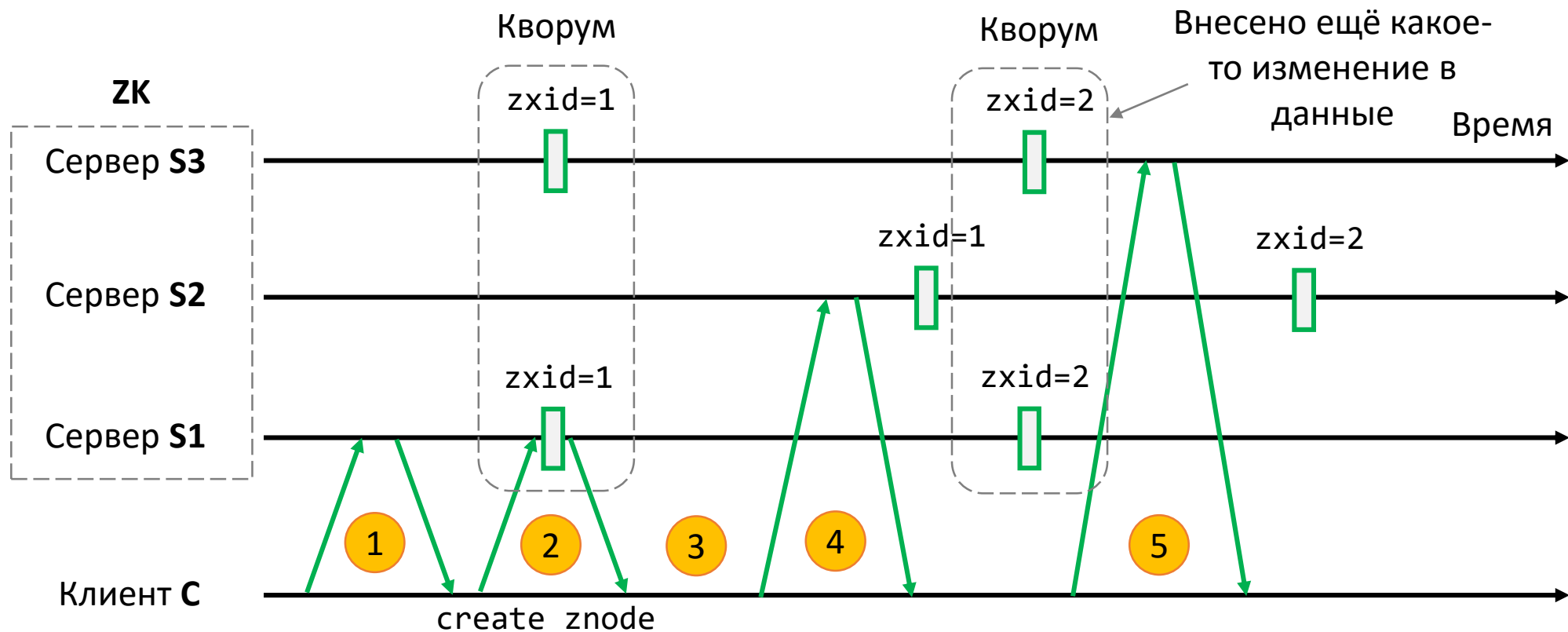
Повторное подключение. Session timeout

Session timeout (T) – период времени, в течение которого существует сессия и соединение может быть восстановлено без установки новой сессии. После этого периода сессия истекает (expired).

- Если в течение **T** сервер не получает сообщение от клиента, то сессия истекает
- Если клиент не может получить ответ от сервера в течение $1/3$ от **T**, то он посылает heartbeat сообщение серверу
- Начиная с $2/3$ от **T** клиент пытается найти другой сервер. У него на это есть $1/3$ от **T**
- Сервер удаляет эфемерные узлы, созданные клиентом, отправляет уведомления другим клиентам и пр.
- После повторной установки соединения клиент получает уведомление об истечении сессии от сервера



Повторное подключение. Пример

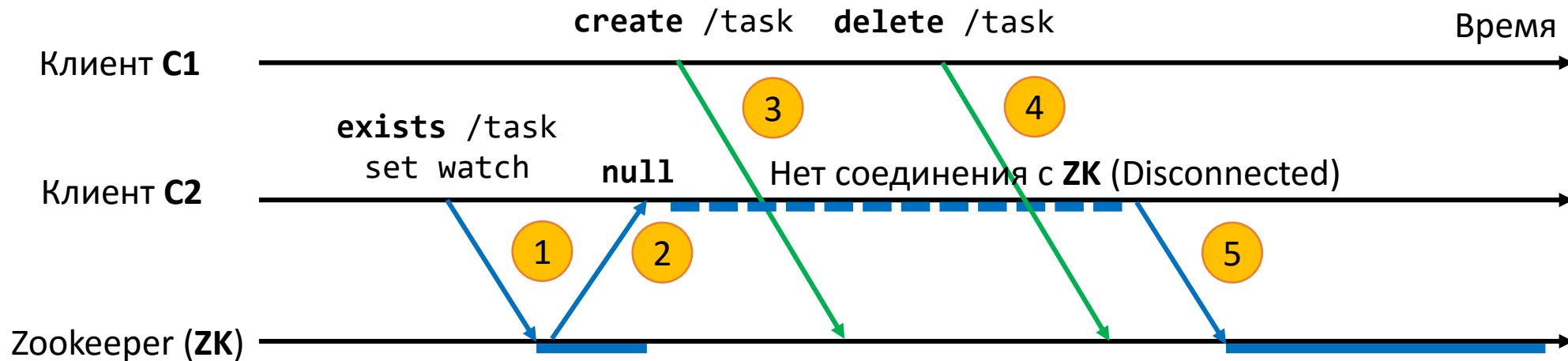


- 1 С подключается к **S1**
- 2 С выполняет **create** операцию, которой соответствует транзакция **zxid=1**
- 3 С теряет соединение с **S1**
- 4 С пытается подключиться к **S2**, но сервер имеет **zxid** меньше 1
- 5 С подключается к **S3**

Восстановление соединения

- При восстановлении соединения восстанавливаются (клиентом) также **watches** на новом сервере
- Когда клиент подключается к серверу ZK, он отправляет список watches и последний zxid, который он видел
- Сервер просматривает временные метки последних изменений соответствующих znode. Если есть новые изменения, то сервер отправит клиенту уведомление, т.е. сработает watch

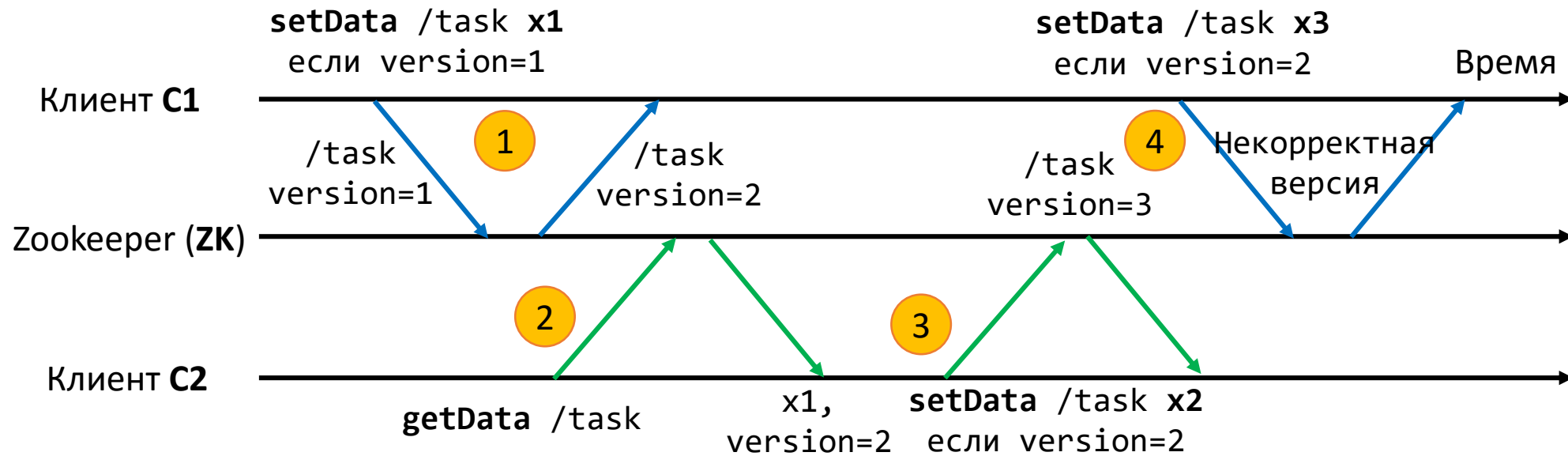
Восстановление соединения. Проблема exists()



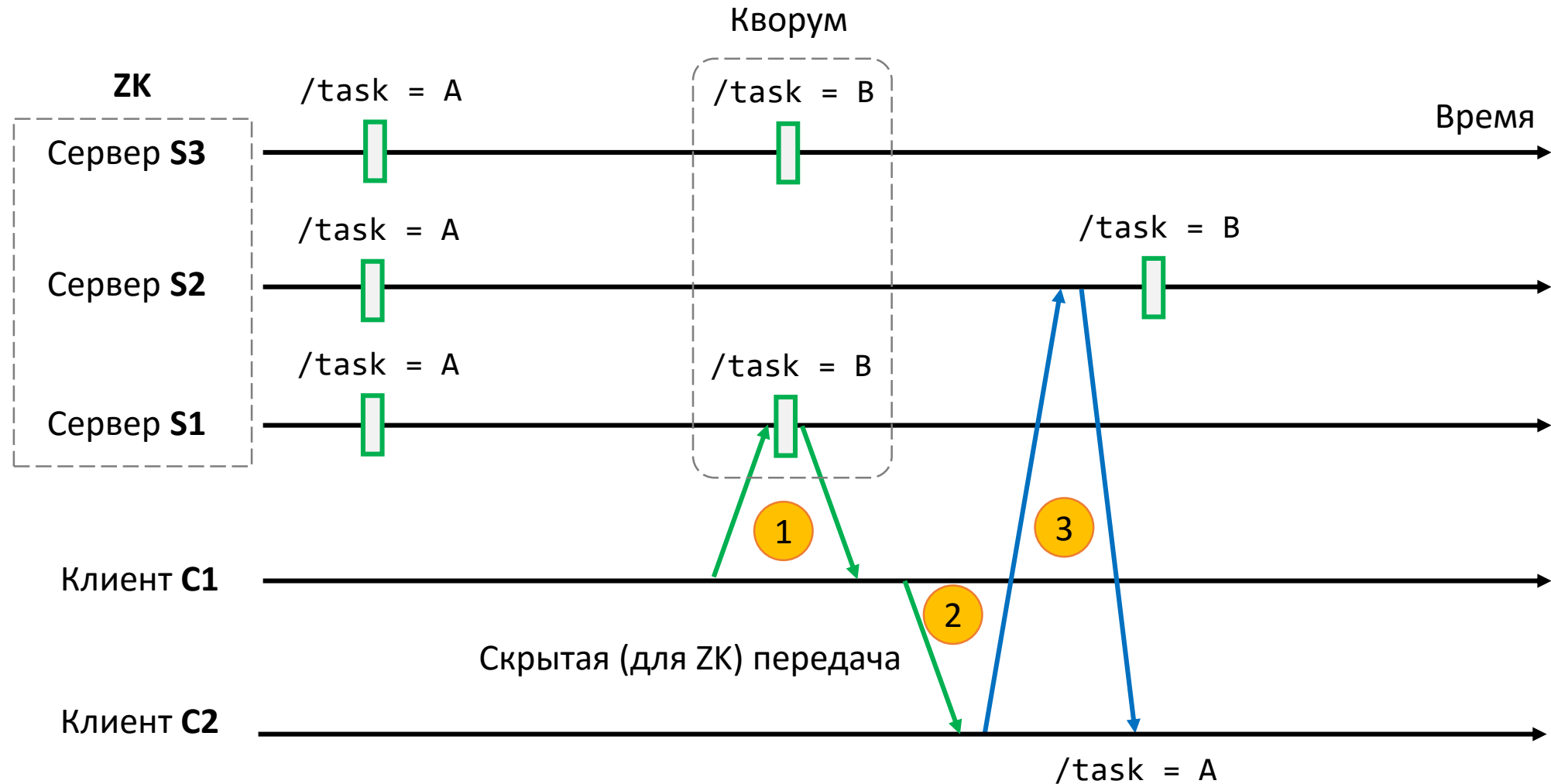
- 1 C2 запрашивает существование `/task` + ставит **watch** на изменения. Получает ответ, что `/task` не существует
- 2 C2 теряет соединение с ZK
- 3 C1 создает `/task` 4 C1 удаляет `/task`
- 5 C2 восстанавливает соединение и **watch** на `/task`. **Watch** не срабатывает и C2 не узнает о шагах 3 и 4

- Каждый **znode** имеет соответствующий ему номер версии
- Значение версии увеличивается на 1 при каждом изменении данных
- Операции с указанием версии: setData, delete
- Операция выполняется, только если указанная версия соответствует текущей версии **znode** на сервере

Версия znode. Запись с условием



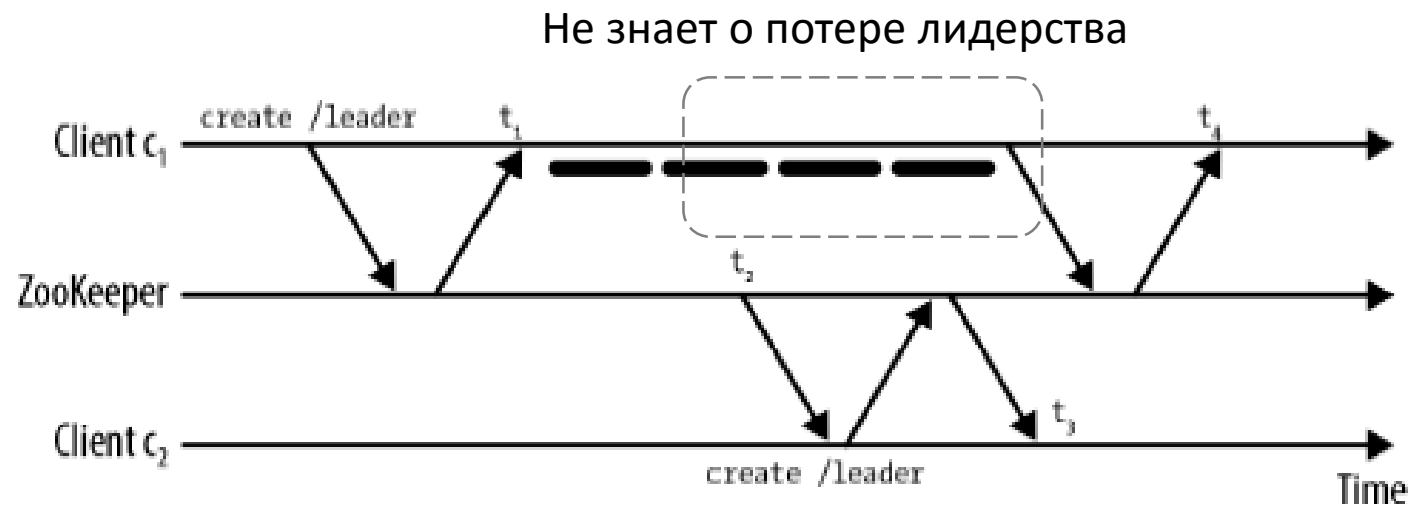
Нарушение порядка. Скрытая передача



- 1 C1 изменяет `/task` на `B`
- 2 C1 сообщает C2 в обход ZK, что изменилась задача

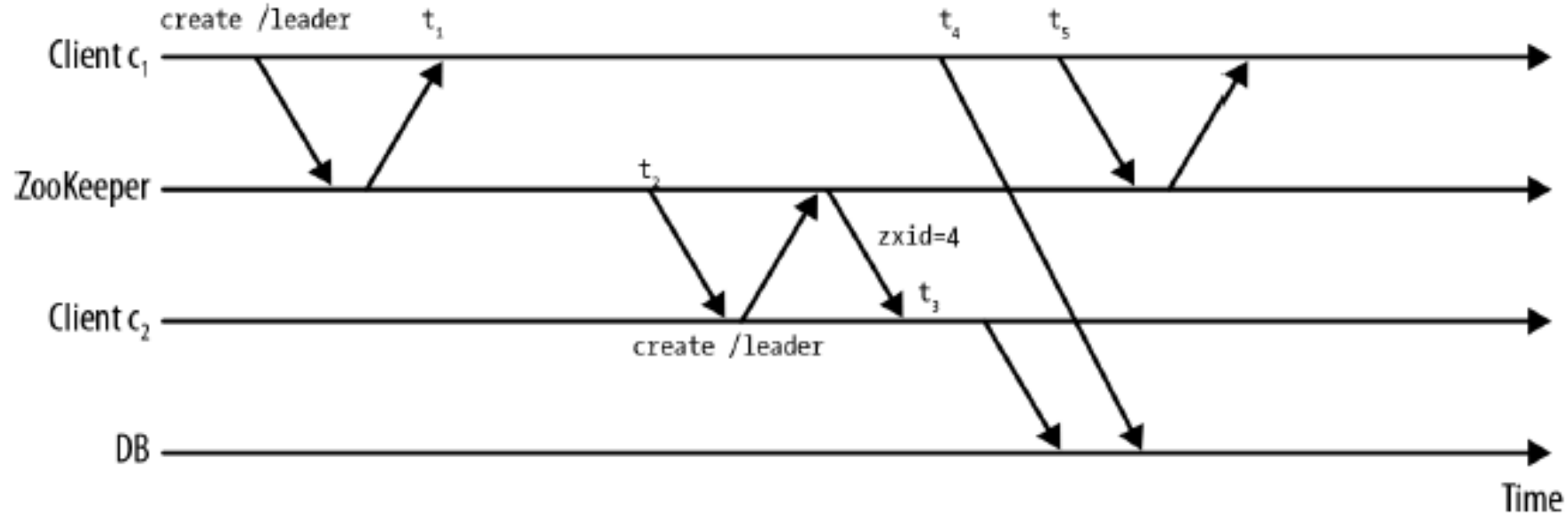
- 3 C2 запрашивает новую задачу, но получает и обрабатывает старый вариант

Нарушение порядка. Пример с двумя мастерами



- t_1 Client c_1 loses connection to ZooKeeper.
- t_2 Client c_2 is notified that c_1 is dead.
- t_3 Client c_2 gets leadership.
- t_4 Client c_1 reconnects to ZooKeeper and finds connection expired.

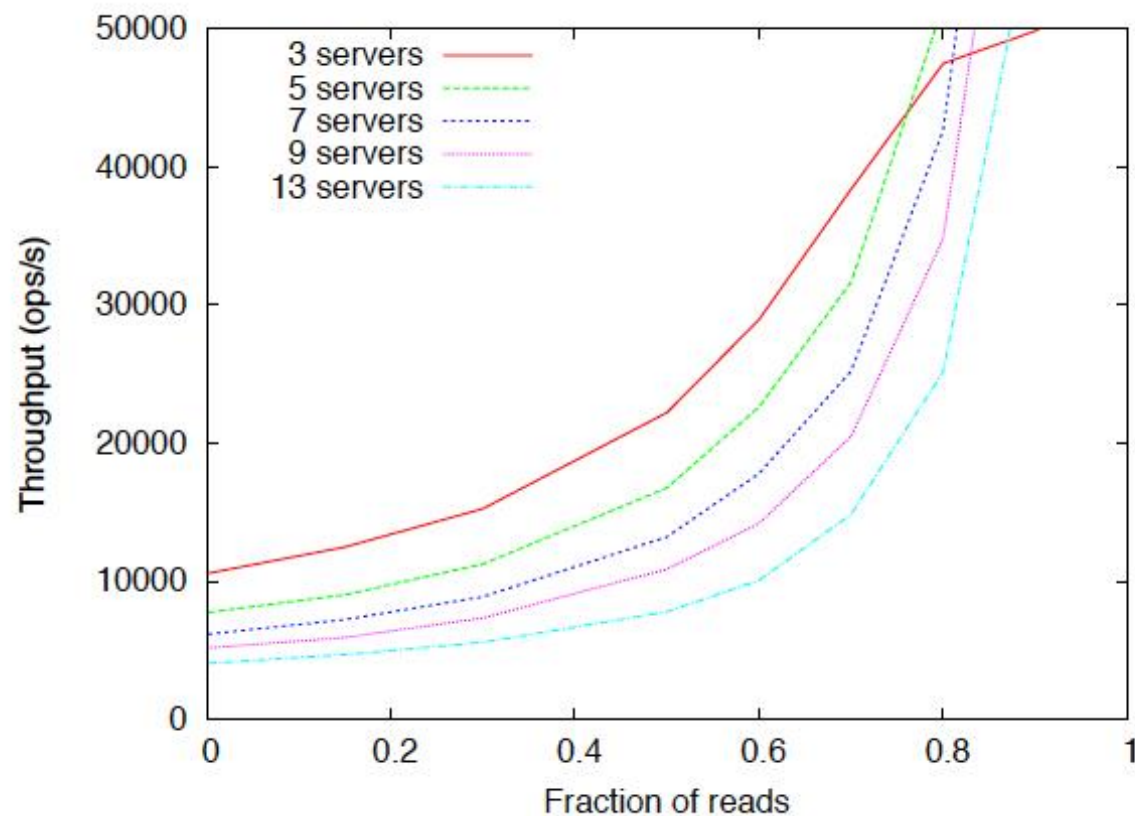
Нарушение порядка. Пример с двумя мастерами



- t_1 Client c_1 starts garbage collecting.
- t_2 ZooKeeper declares c_1 dead.
- t_3 Client c_2 becomes the leader, syncs with DB, and updates data.
- t_4 Previously queued and delayed update gets sent to database.
- t_5 Client c_1 reconnects to ZooKeeper and finds that its session has expired.

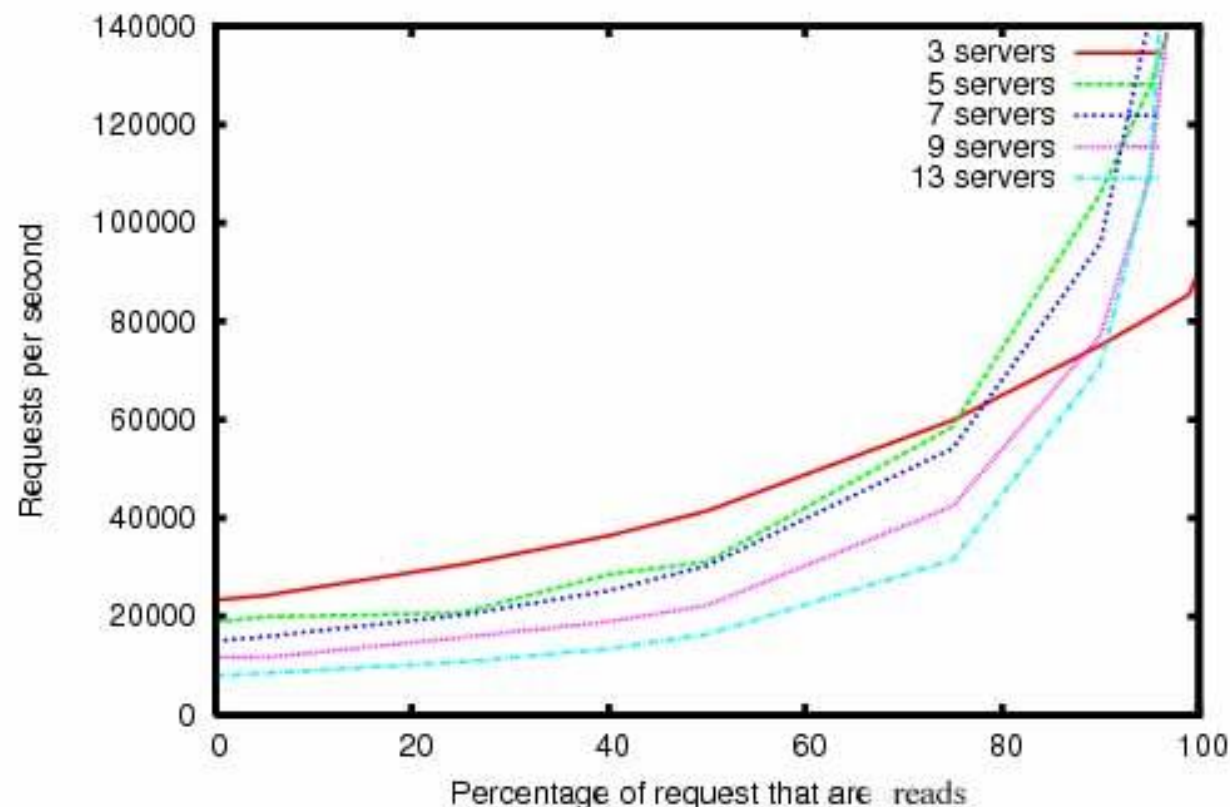
Производительность Zookeeper. Чтение/запись

v3.1.1



v3.3.5

910 clients



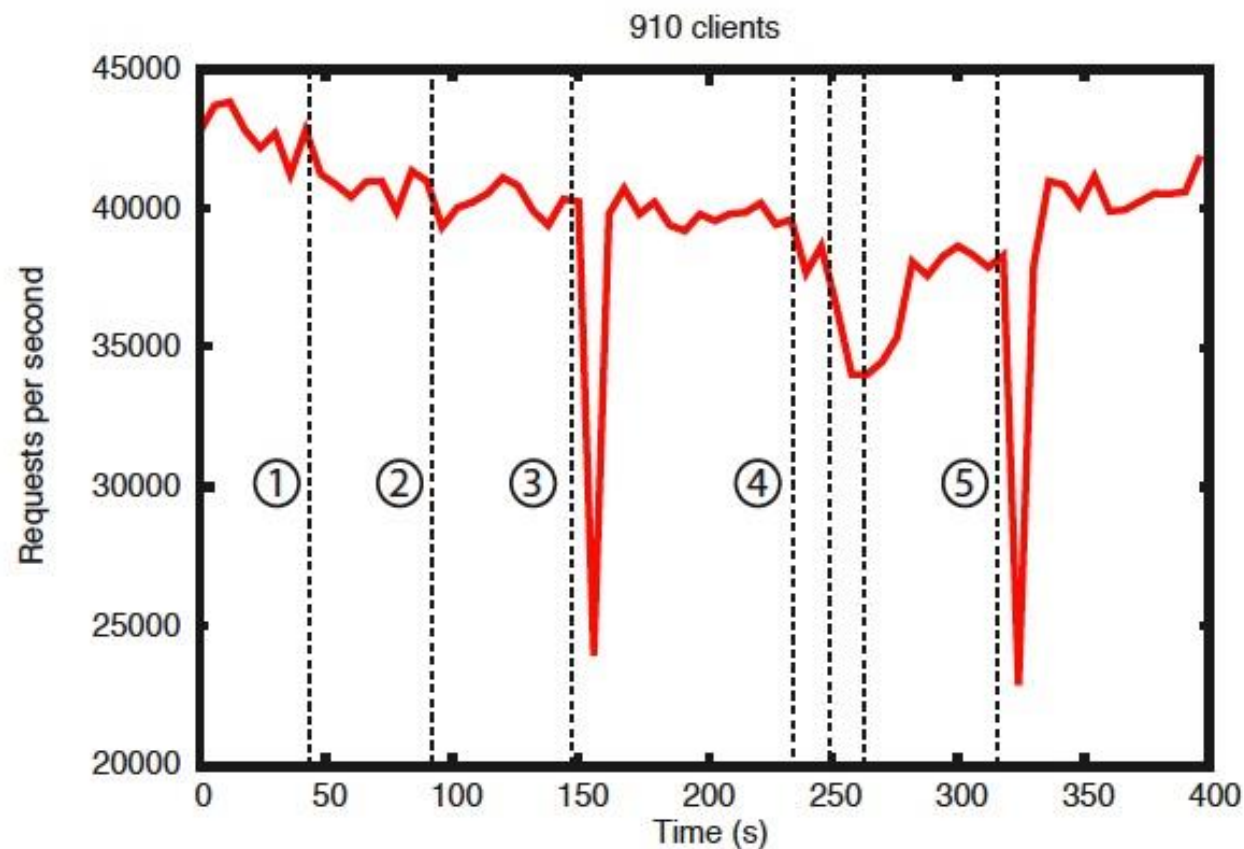
<http://zookeeper.apache.org/doc/r3.1.1/zookeeperOver.html>

<http://zookeeper.apache.org/doc/r3.3.5/zookeeperOver.html>

Производительность при сбоях

7 серверов
Запись 30%

1. Сбой и восстановление follower
2. Сбой и восстановление другого follower
3. Сбой leader
4. Сбой и восстановление двух follower
5. Сбой нового leader



<http://zookeeper.apache.org/doc/r3.3.5/zookeeperOver.html>

Curator Framework высокоуровневое API, которое упрощает работу с Zookeeper'ом

Некоторые особенности:

- Автоматическое восстановление соединения при возникновении ошибок
- Простое и понятное API для работы с методами и событиями ZK
- Готовые реализации выбора лидера, распределённой блокировки, распределённых очередей и др.



<http://curator.apache.org/curator-framework/>

ZooKeeper by Flavio Junqueira and Benjamin Reed (book)

[Zookeeper](#) (github source code)

[ZooKeeper: A Distributed Coordination Service for Distributed Applications](#) (doc)

[ZooKeeper Programmer's Guide](#) (doc)

[ZooKeeper Recipes and Solutions](#) (doc)

[В чем польза ZooKeeper для админов и разработчиков. Семинар в Яндексе](#) (blog)