

Распределённая координация

ZooKeeper

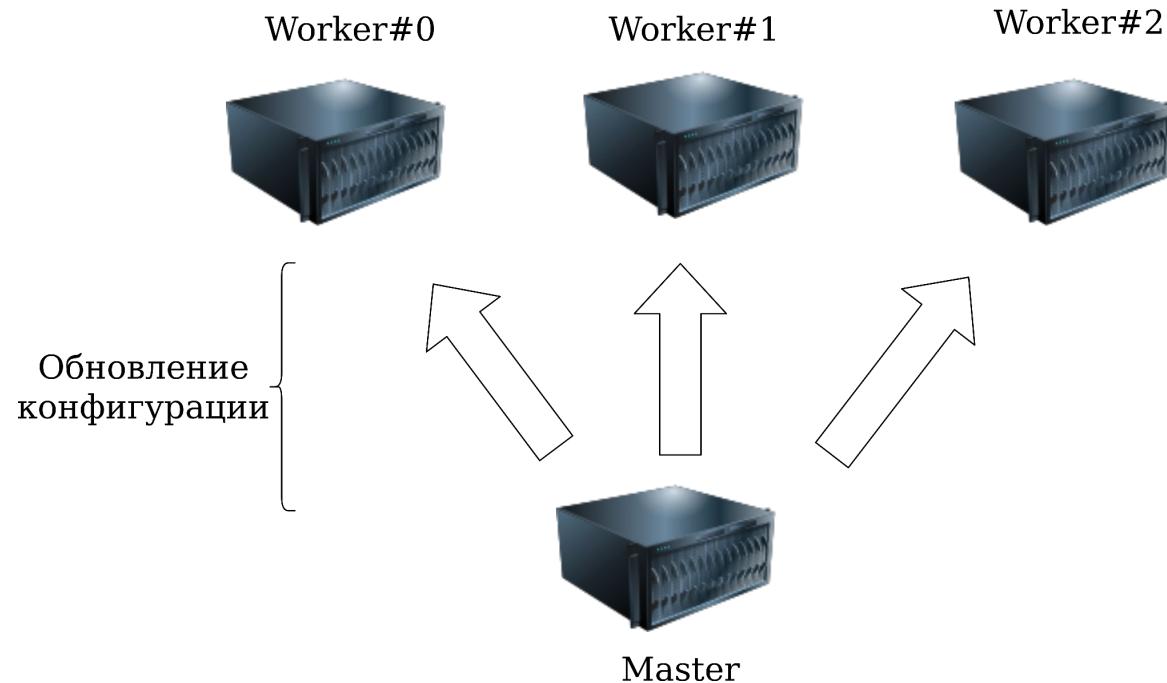


Илья Кокорин

kokorin.ilya.1998@gmail.com

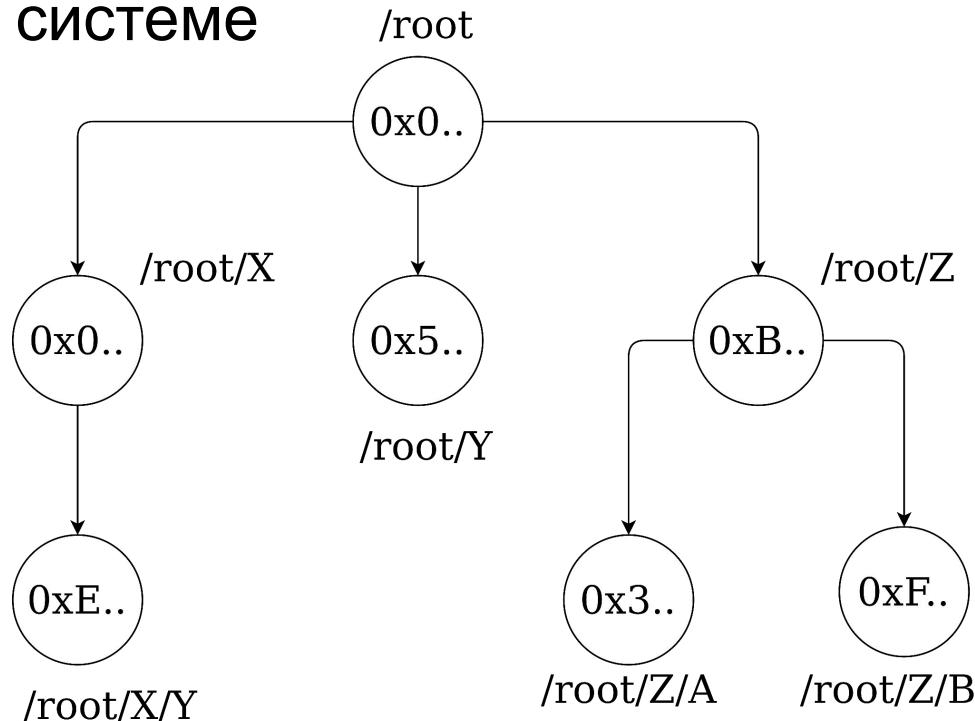
Разнообразие примитивов координации

- Существует очень много примитивов распределённой координации
 - Блокировки
 - Барьеры
 - Выбор лидера
 - Конфигурация
 - ...
- Хотим придумать универсальный примитив



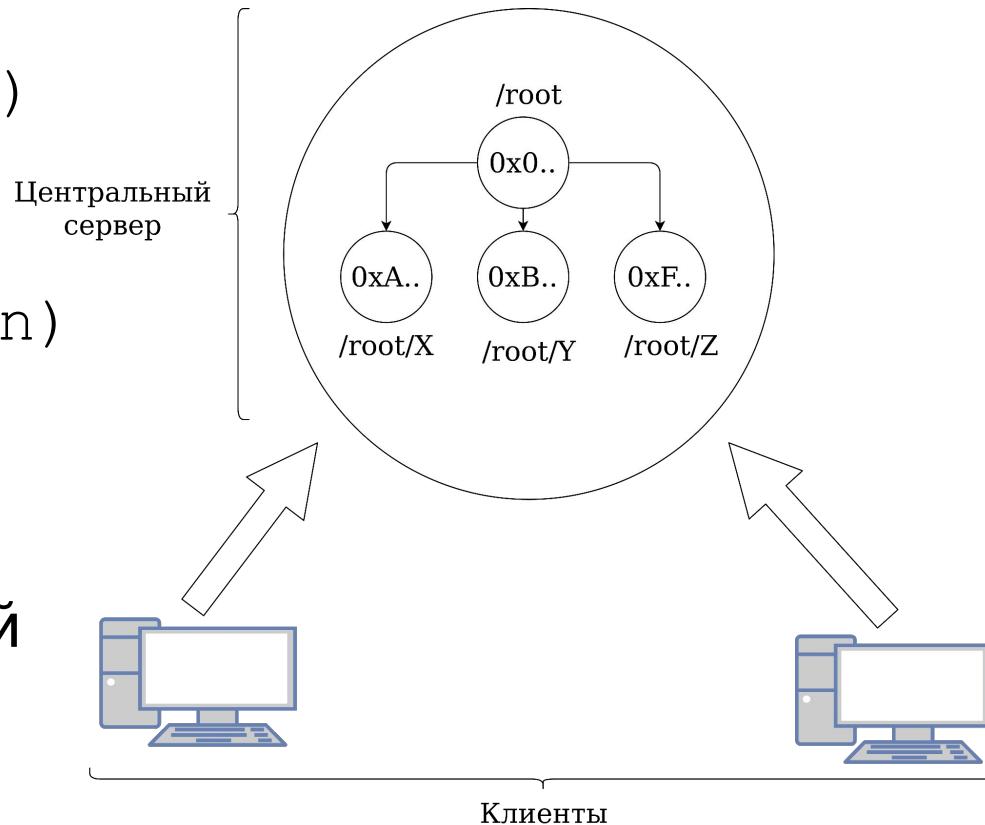
ZooKeeper: модель данных

- Иерархическое key-value хранилище
- Ключи образуют дерево
 - Прямо как в файловой системе
- Значения - произвольные байтовые строки
- Каждый узел хранит значение и может иметь детей
 - Комбинация файла и директории



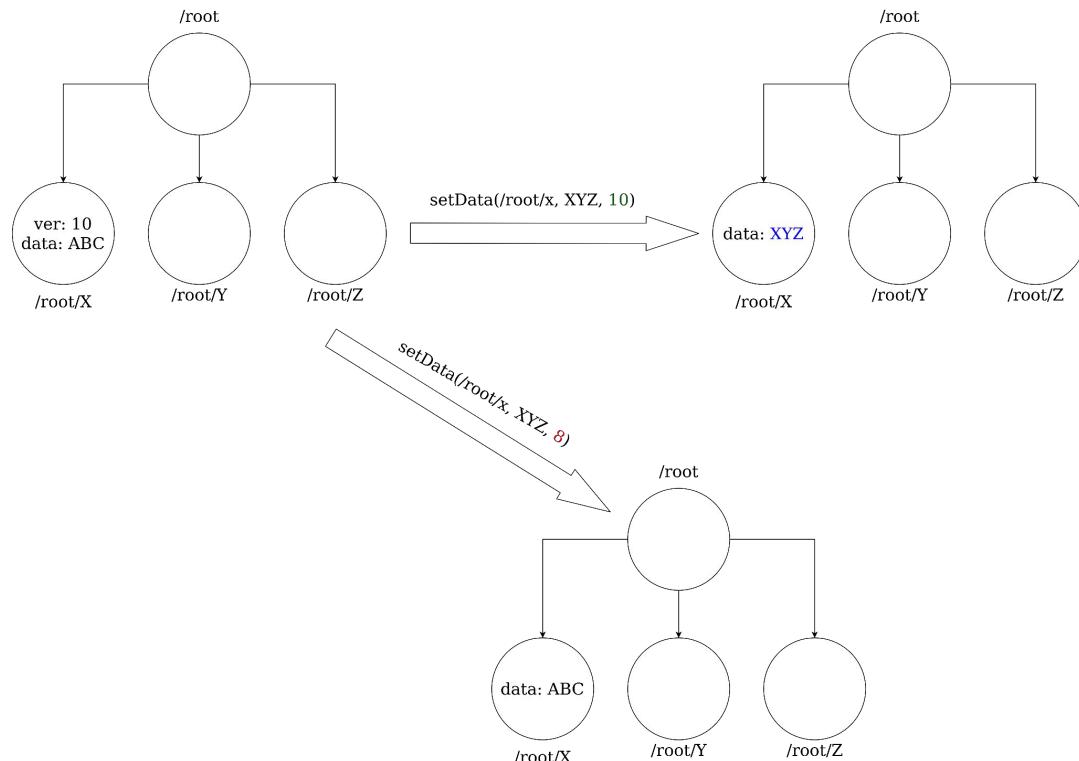
ZooKeeper на единственном сервере

- create(path, data)
- delete(path, version)
- exists(path)
- getData(path)
- setData(p, d, version)
- getChildren(path)
- sync()
 - Загадочная операция
 - Позже поговорим о ней



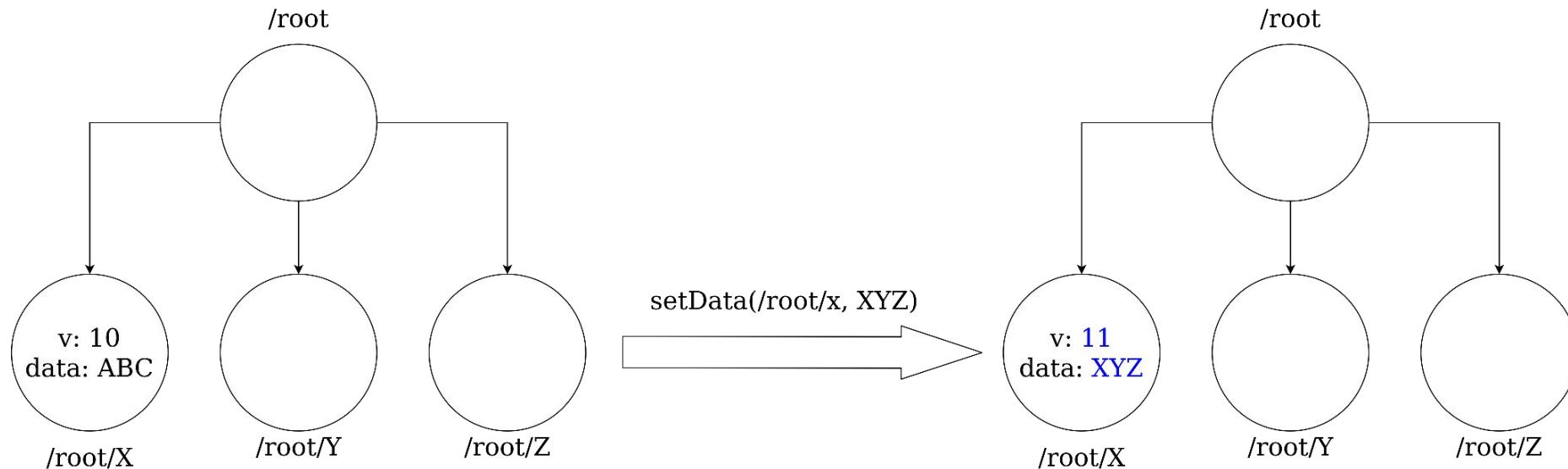
ZooKeeper: Версии узлов

- Версия узла передаётся в каждом модифицирующем запросе
- Если версия совпала с версией узла - операция будет совершена
 - Иначе будет проигнорирована
- `version = -1` делает операцию безусловной



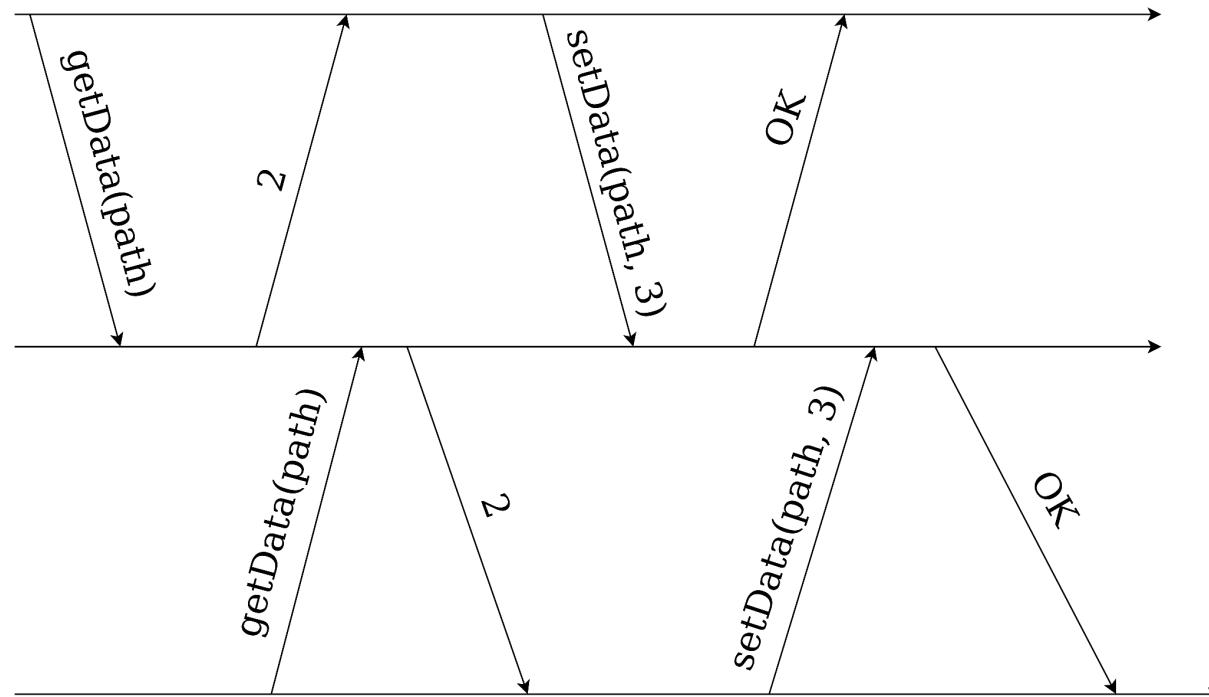
ZooKeeper: Версии узлов

- Версия узла увеличивается при каждом **успешном изменении**



Зачем нужны версии: инкремент

- Хотим увеличить значение в узле на единицу
 - val := zk.getData(path)
 - zk.setData(path, val + 1)
- Проблема потерянных обновлений



Зачем нужны версии: инкремент

- Используем условные обновления для реализации корректного инкремента
 - Похоже на lock-free универсальную конструкцию из конкурентного программирования

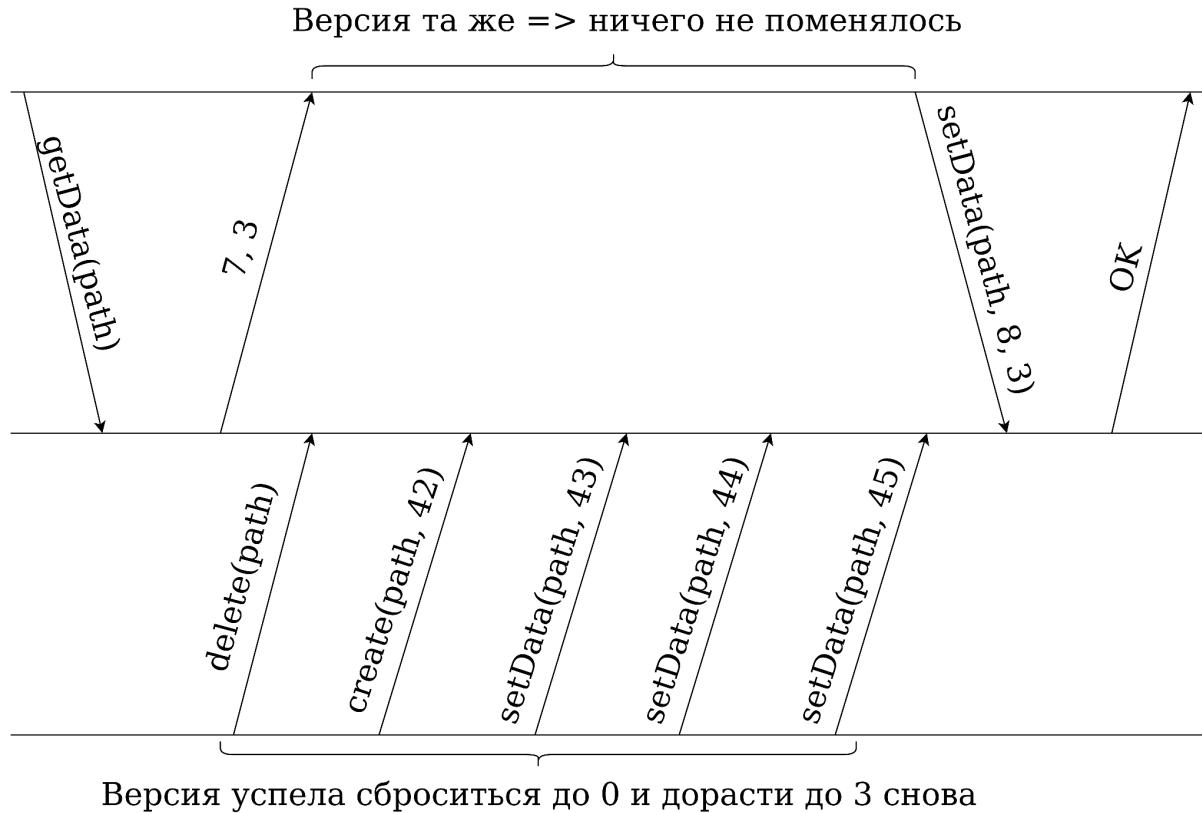
```
1 while true:  
2     x, version := zk.getData(path)  
3     success = zk.setData(path, x + 1, version)  
4     if success:  
5         break
```

Версионирование: подводные камни

- После удаления узел создаётся с нулевой версией
- `create(path, data)`
 - `version = 0`
- `setData(path, new_data)`
 - `version = 1`
- `setData(path, very_new_data)`
 - `version = 2`
- `delete(path)`
- `create(path, some_data)`
 - `version = 0`

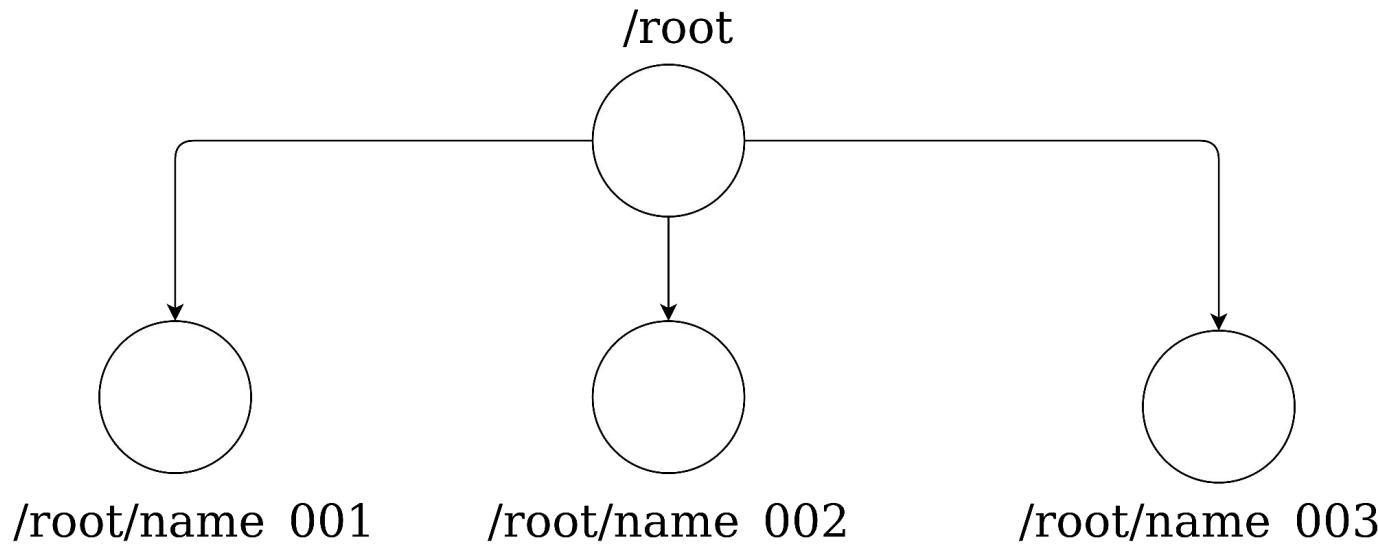
Версионирование: подводные камни

- После удаления узел создаётся с нулевой версией



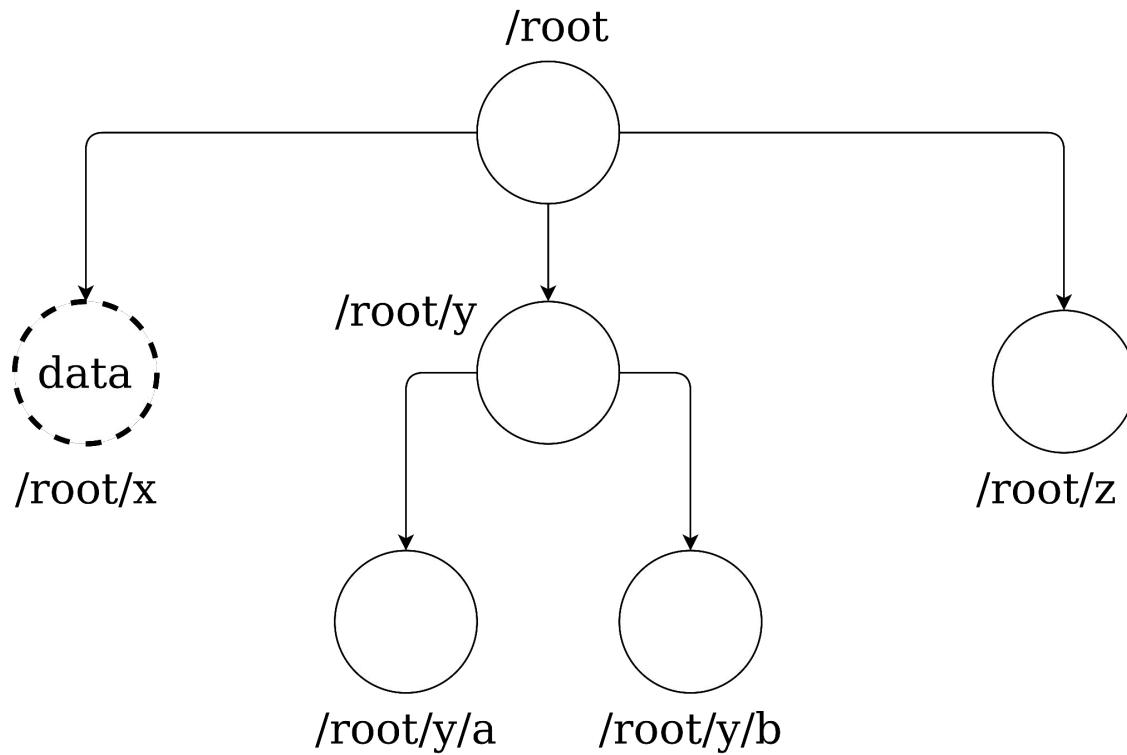
Последовательные узлы

- name := create(/path/to/node_, SEQUENTIAL)
- Узел будет создан всегда
 - Имя будет состоять из заданного префикса
 - И последовательно увеличивающегося числа



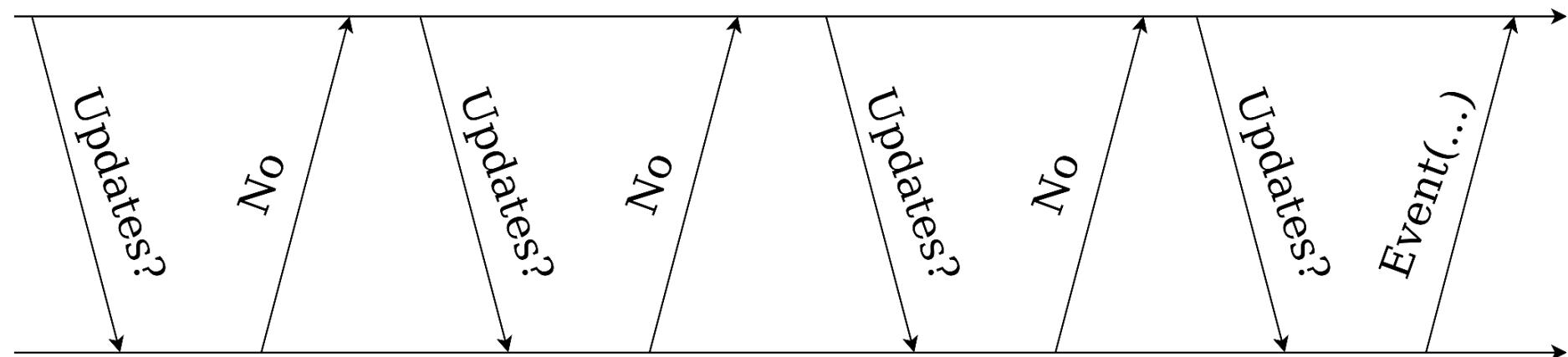
Эфемерные узлы

- `create(/path/to/node, EPHEMERAL)`
- Существуют только пока создавший их клиент подключён к ZooKeeper
 - Пока шлёт хартбиты
- Не могут иметь детей
 - Но могут хранить данные



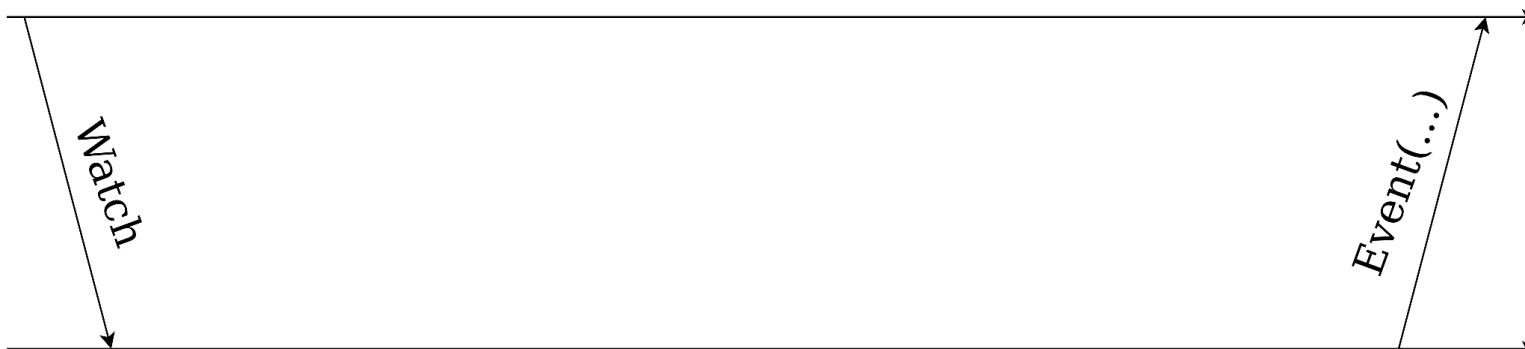
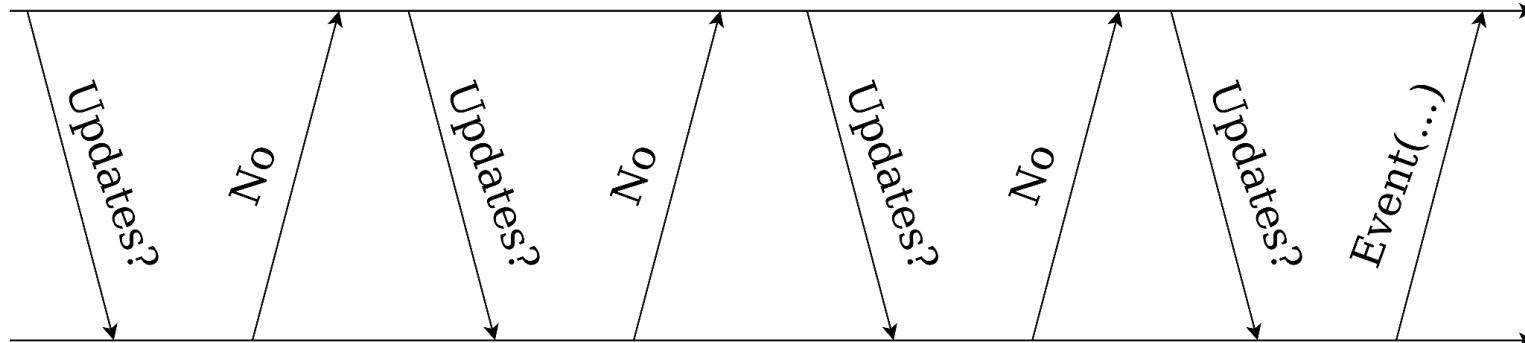
Уведомления / Watches

- Хотим дождаться какого-то события
 - Узел появился
 - Узел удалён
 - У узла изменился список детей
 - Изменились данные, которые хранит узел
- Можно использовать поллинг



Уведомления

- Но уведомления использовать эффективней



Уведомления

- Уведомление говорит, что узел изменился
 - Но не говорит, как именно изменился
 - Например какие данные в него записаны
- Когда придёт уведомление, нужно пойти и прочитать узел чтобы это узнать

```
watch_id :=  
    zk.Watch(/path/to/node, DATA_CHANGED)  
zk.Wait(watch_id)  
new_data := zk.GetData(/path/to/node)
```

Уведомления

- Отдельной операции zk.Wait(path, event) не существует
- Начинаем следить за событиями в результате читающих операций
 - zk.Exists(path, watcher) следит за созданием, удалением и изменением данных узла
 - zk.GetData(path, watcher) следит за удалением узла и изменением его данных
 - zk.getChildren(path, watcher) следит за удалением узла и изменением списка его детей

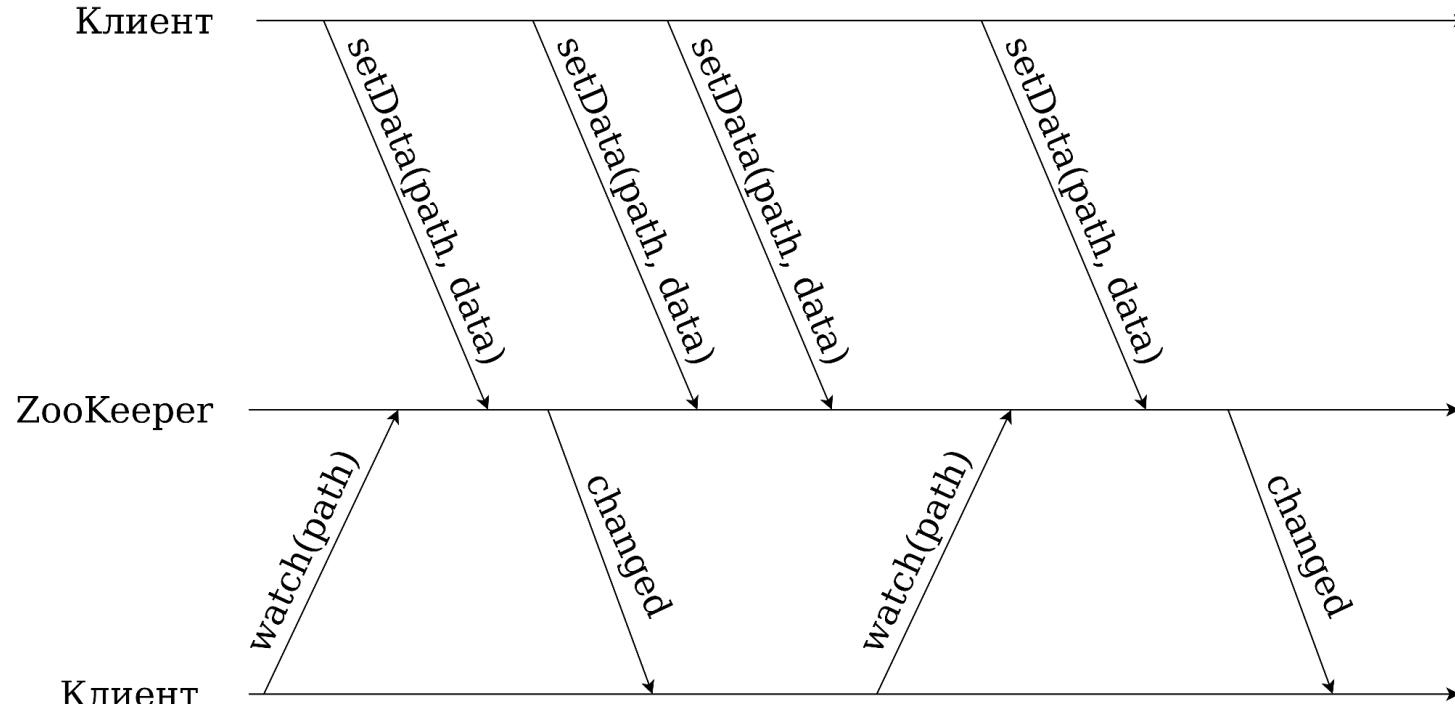
Уведомления

- Что такое watcher в zk.ReadOp(path, watcher)?
- Это коллбек
 - В него приходят события, он реагирует
 - Подробнее посмотрим на практике
- Дальше продолжим писать в псевдокоде

```
1 data := zk.GetData(path, event -> {
2     if event.Type = NODE_DELETED:
3         log.Printf("Node %v deleted", path)
4     elif event.Type = DATA_CHANGED:
5         log.Printf("Data at %v changed", path)
6 })
```

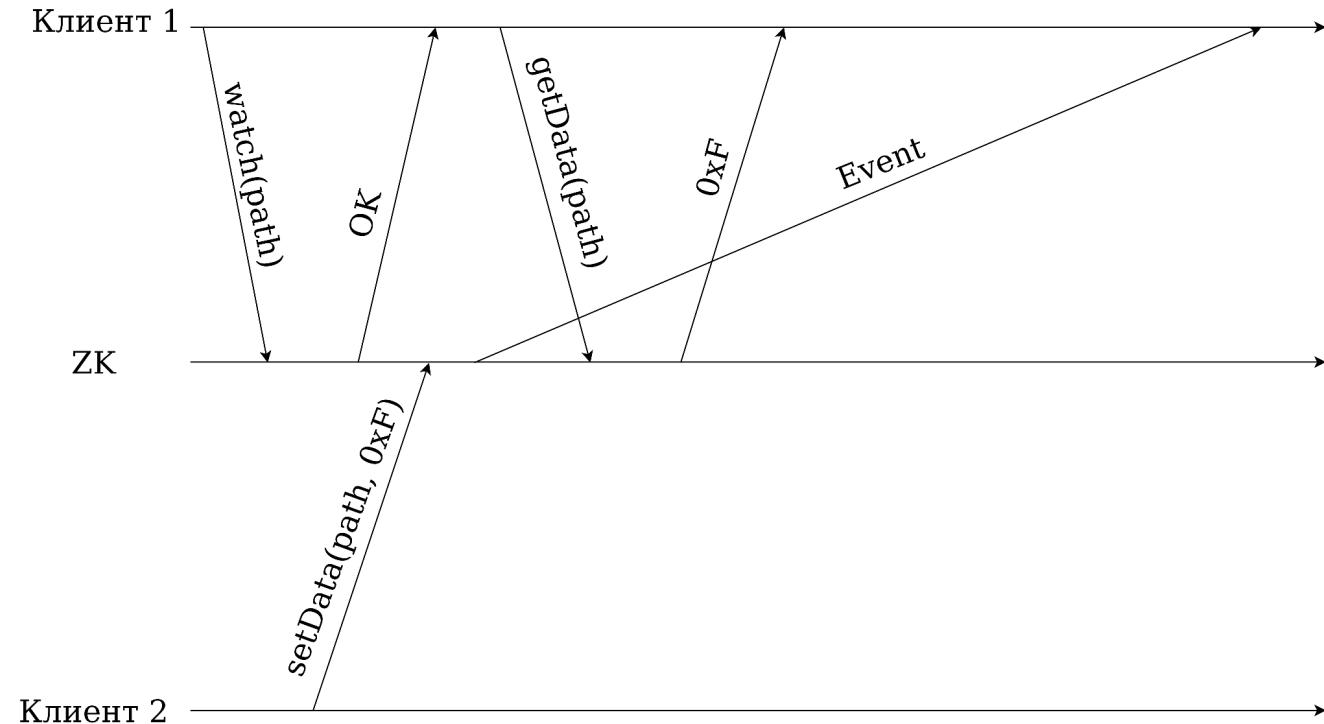
Сброс и переустановка уведомлений

- Уведомление сбрасывается после срабатывания
 - Нужно переустанавливать вручную



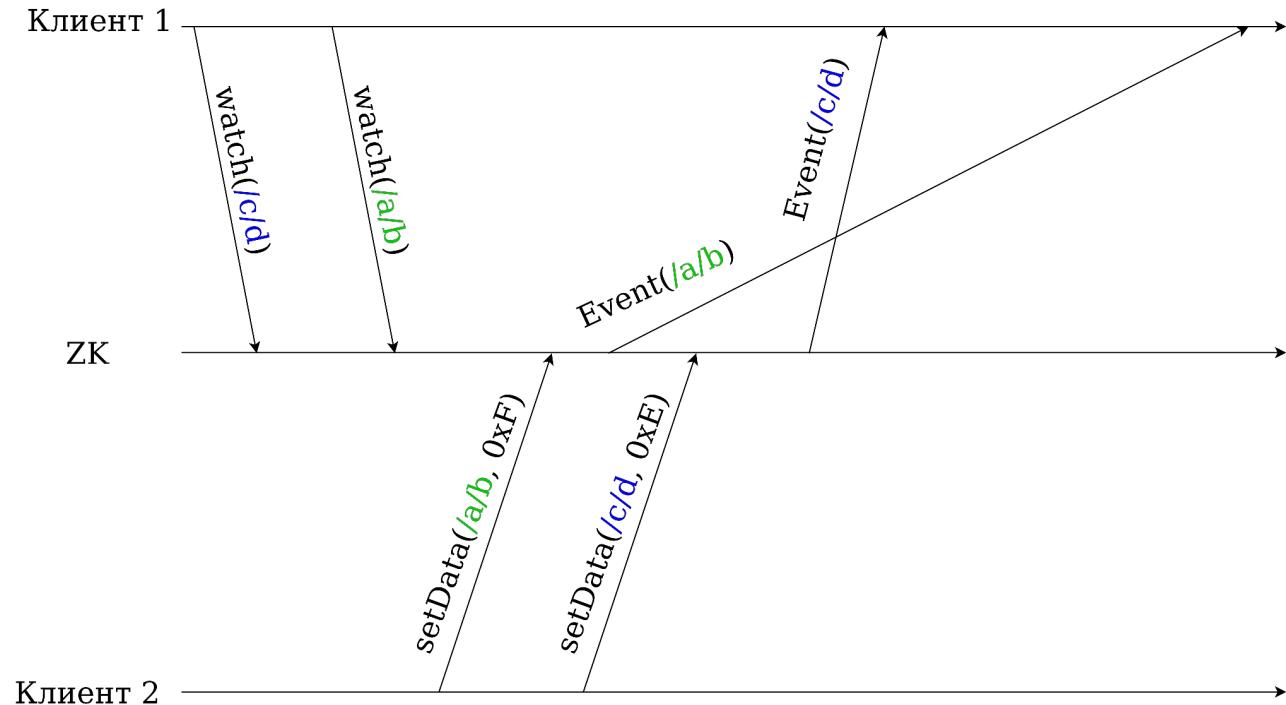
Порядок доставки уведомлений

- Сначала клиент увидит уведомление об изменении, и только потом сможет увидеть само изменение
- Уведомления и результаты используют один FIFO-канал
 - TCP на практике



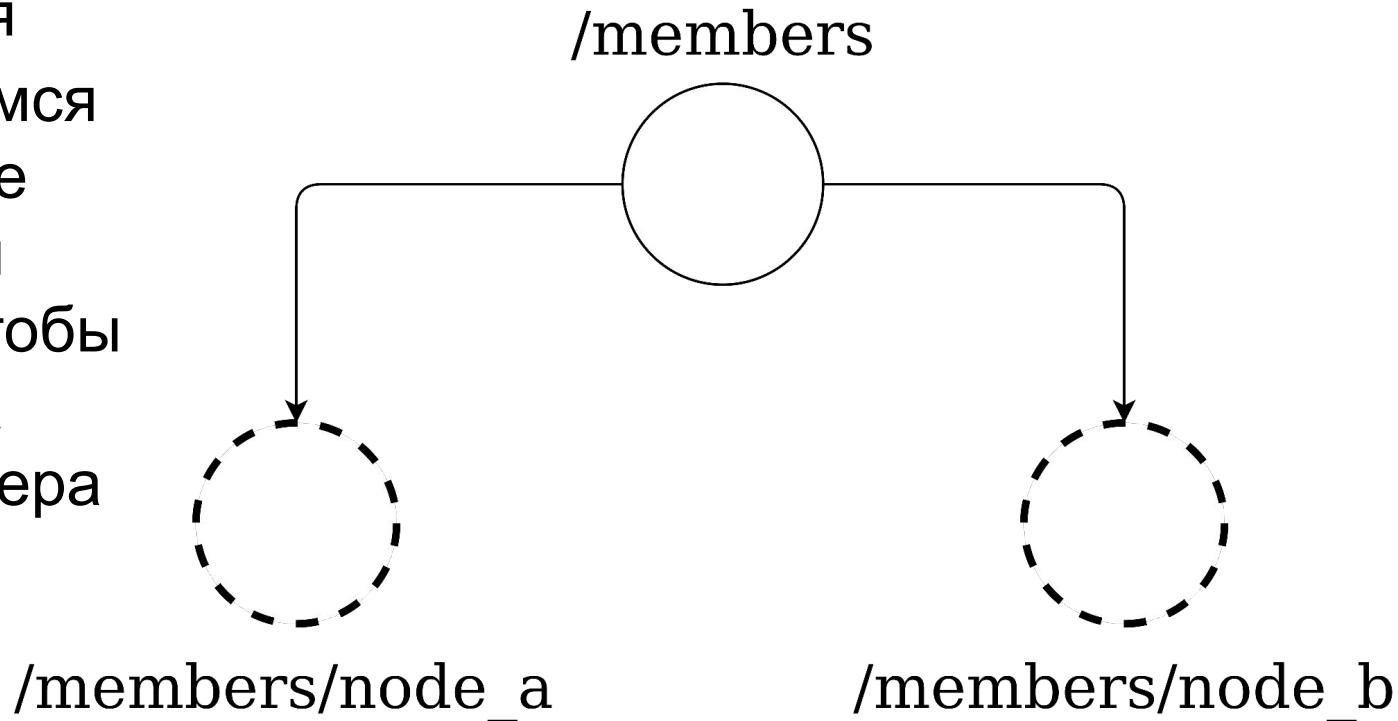
Порядок доставки уведомлений

- Клиент получает уведомления в том порядке, в котором менялись узлы
- Не в порядке установки уведомлений
- Посыпаем уведомления по по одному FIFO-каналу



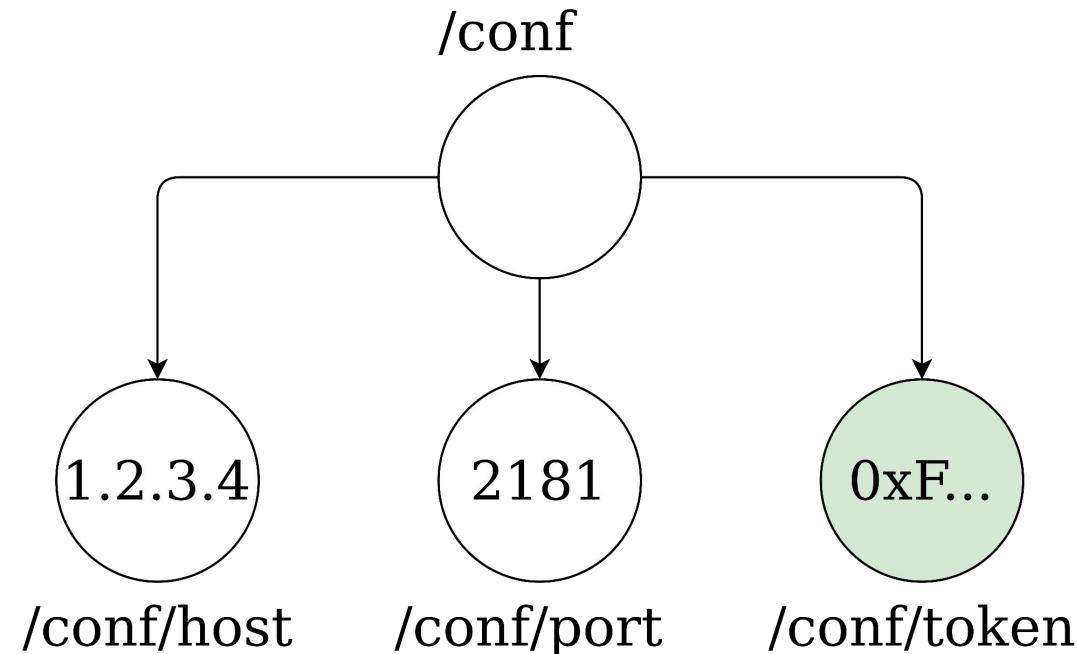
Примитивы координации: состав кластера

- Создаём эфемерный узел /members/node_id
 - При отключении узла от кластера узел автоматически удаляется
- Подписываемся на изменение списка детей /members чтобы узнать, когда состав кластера меняется



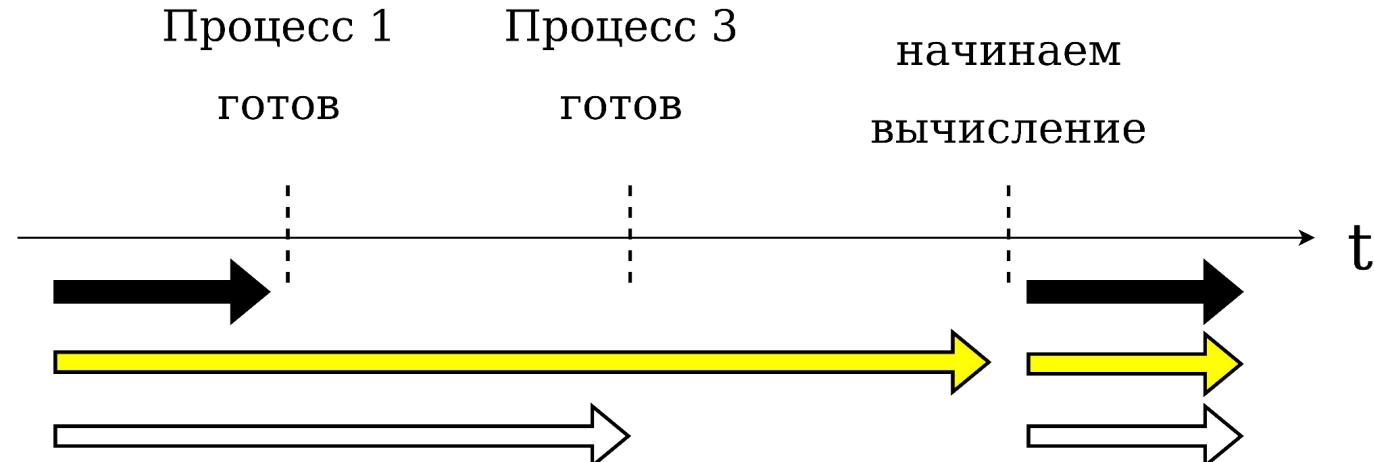
Примитивы координации: конфигурация

- Зависит от того, как именно мы храним конфигурацию
- Подписываемся на изменение значения /conf/port чтобы узнать, когда поменяется порт
- Подписываемся на изменение списка детей /conf чтобы узнать, когда появляются новые элементы конфигурации



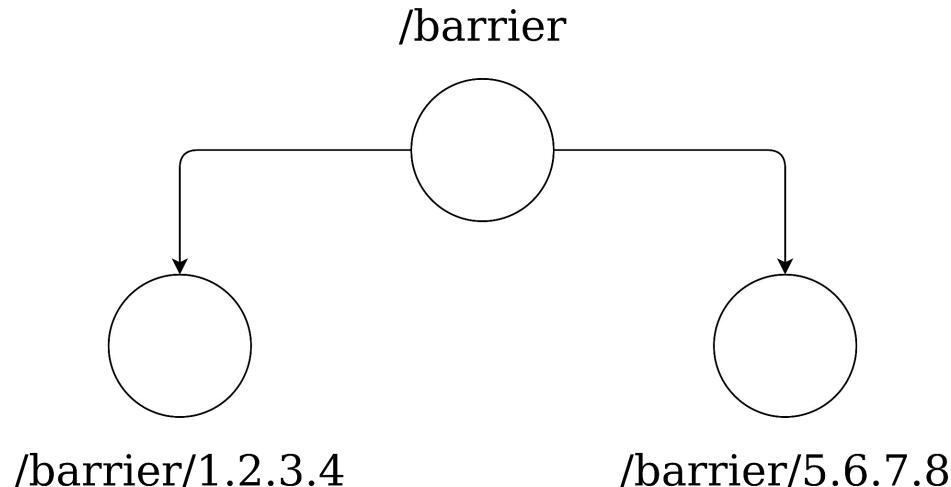
Примитивы координации: барьеры

- Есть K процессов
 - В произвольный момент времени может заявить о своей готовности начать работу
 - После этого процесс ждёт остальных
 - Начинаем работать как только все готовы
- Все процессы готовы, начинаем вычисление



Примитивы координации: барьеры

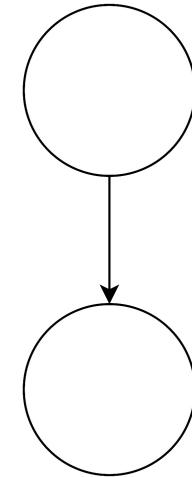
```
1 fun Barrier(node_id, k):  
2     zk.Create("/barrier/{node_id}")  
3     while true:  
4         children := zk.GetChildren()  
5         if children.Size() < k:  
6             watch_id := zk.Wait("/barrier", CHILDREN_LIST_CHANGED)  
7             zk.Wait(watch_id)  
8         else:  
9             return
```



Примитивы координации: блокировки

```
1 fun Lock(lock_id):
2     path := "/locks/" + lock_id
3     while true:
4         created := zk.Create(path)
5         if created:
6             return
7         watch_id := zk.Watch(path, NODE_DELETED)
8         zk.Wait(watch_id)
9
10 fun Unlock(lock_id):
11     path := "/locks/" + lock_id
12     zk.Delete(path)
```

/locks

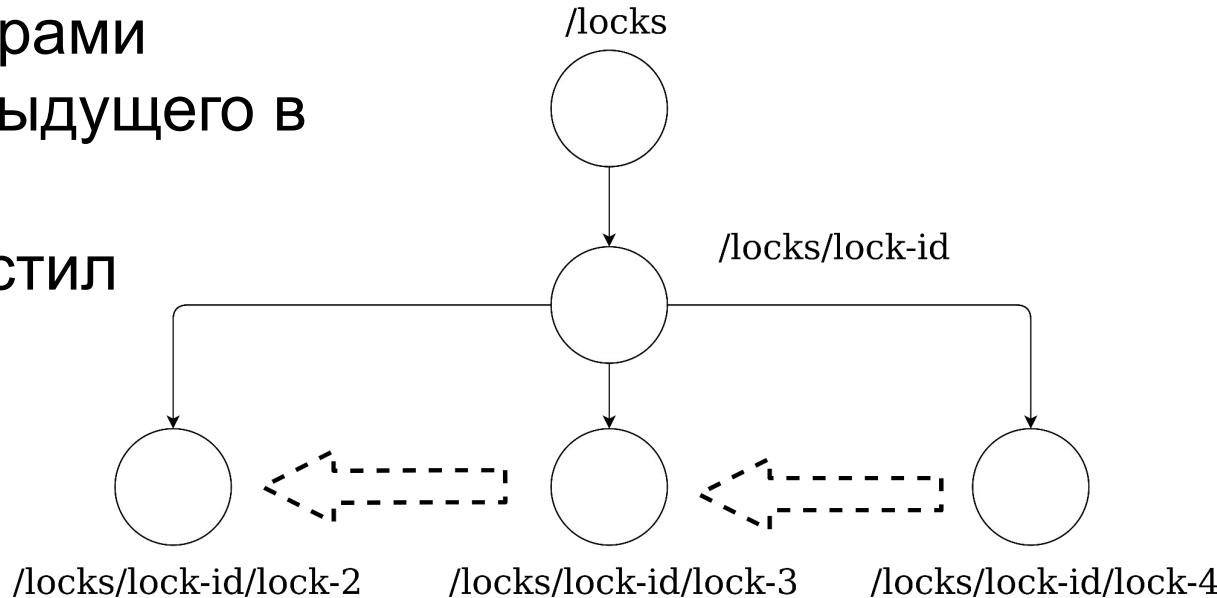


- Что случится при разблокировке?
 - Herd effect
 - Все клиенты проснутся и попытаются захватить блокировку

/locks/lock_id

Блокировки без Herd effect

- Выстраиваем желающих получить блокировку в очередь
 - Каждому выдаём номер билета - монотонно возрастающее число
 - Блокировку берём в порядке, заданном номерами
- Ждём только предыдущего в очереди
- Предыдущий отпустил блокировку - мы проснулись



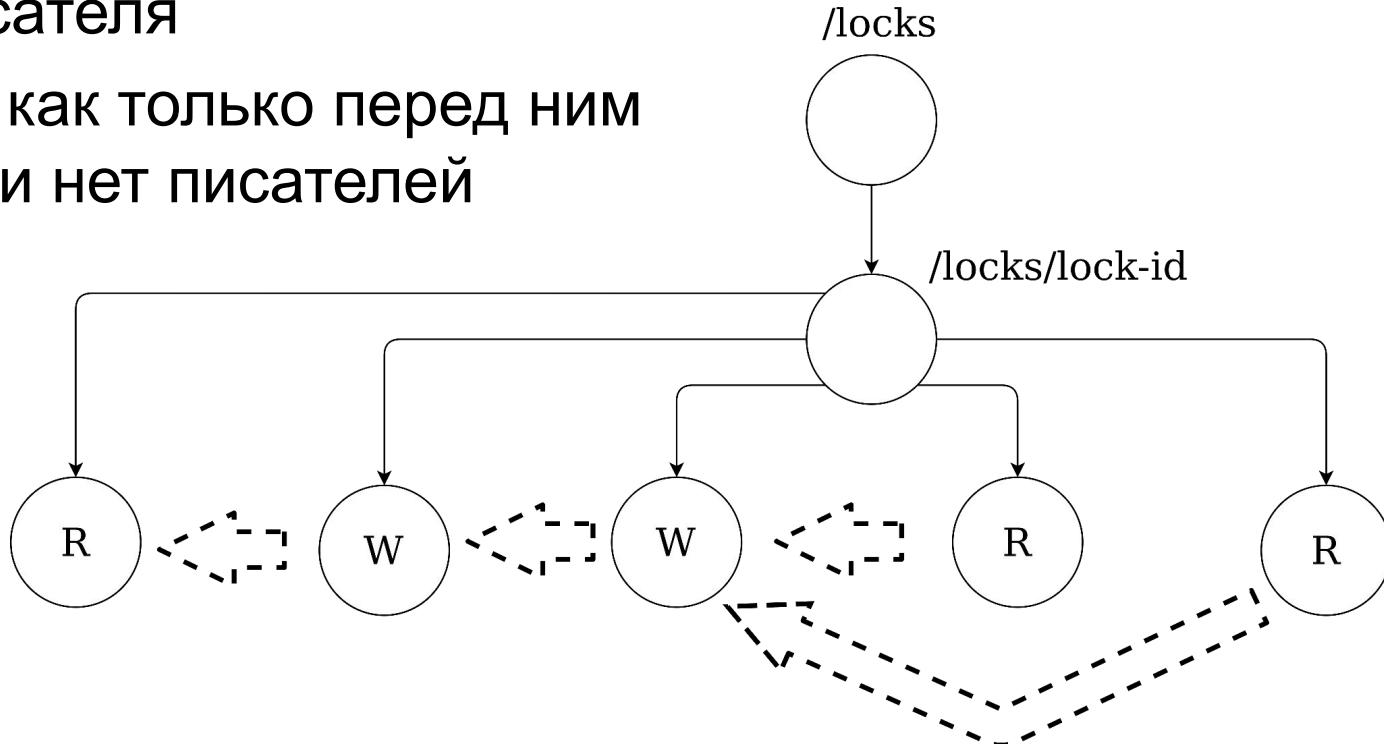
Блокировки без Herd effect

- Отпуская блокировку, удаляем свой узел из дерева
 - Даём возможность зайти следующему

```
1 fun Lock(lock_id):  
2     path := "/locks/{lock_id}"  
3     name := zk.Create("{path}/lock_", SEQUENTIAL)  
4     while true:  
5         children := zk.getChildren()  
6         if children[0] = name:  
7             return name  
8         prev_name := children[children.Find(name) - 1]  
9         watch_id := zk.Watch("{path}/{prev_name}", NODE_DELETED)  
10        zk.Wait(watch_id)  
11  
12    fun Unlock(lock_id, name):  
13        zk.Delete("/locks/{lock_id}/{name}")
```

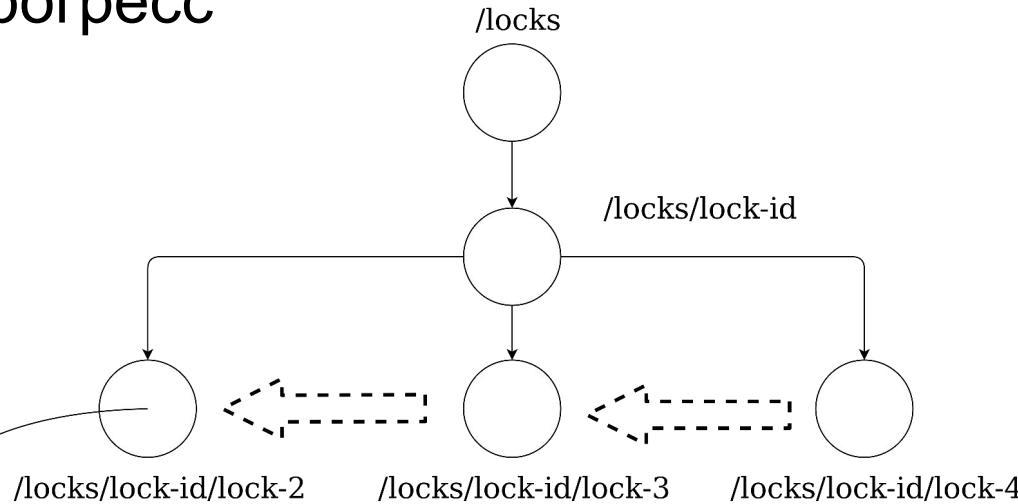
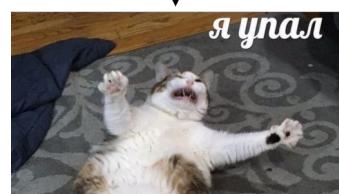
Read/Write блокировки

- Для каждого узла известен его тип: R или W
- Читатель следит за состоянием ближайшего к нему в очереди писателя
 - Заходит, как только перед ним в очереди нет писателей
- Писатель следит за стоящим перед ним в очереди



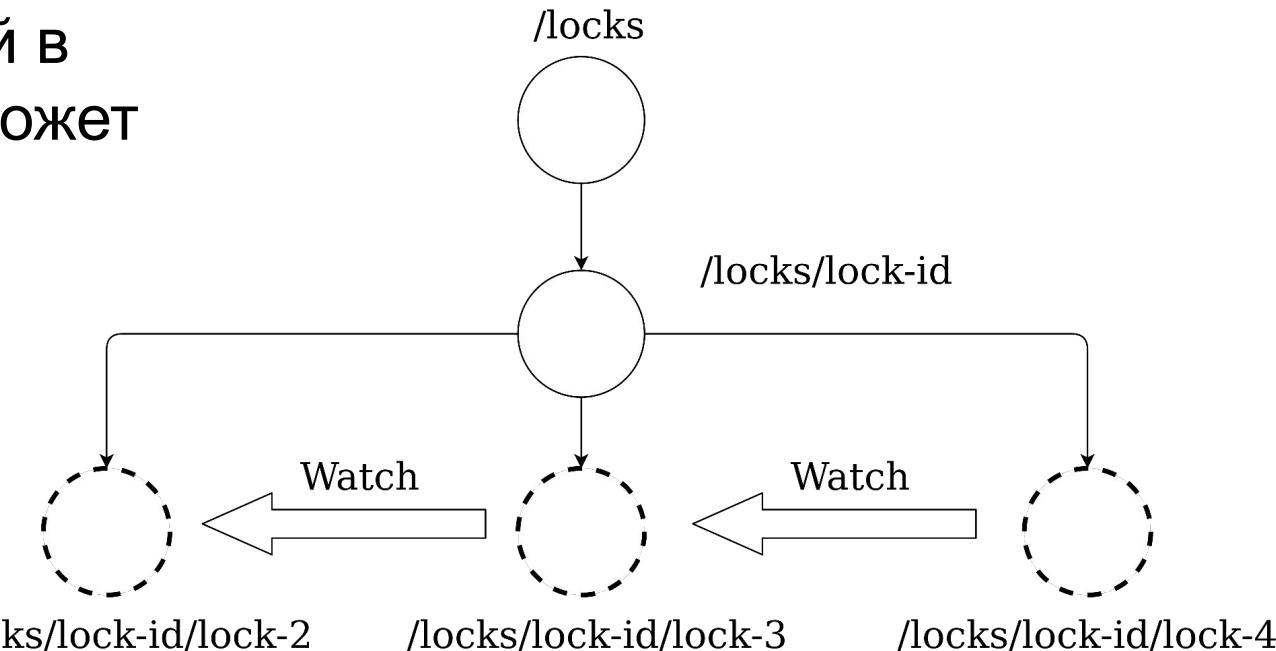
Блокировки и эфемерные узлы

- Процесс захватил блокировку и упал
- Система не совершает прогресс
 - Рабочие процессы не могут перехватить блокировку
- Ждём, пока хозяин блокировки восстановится



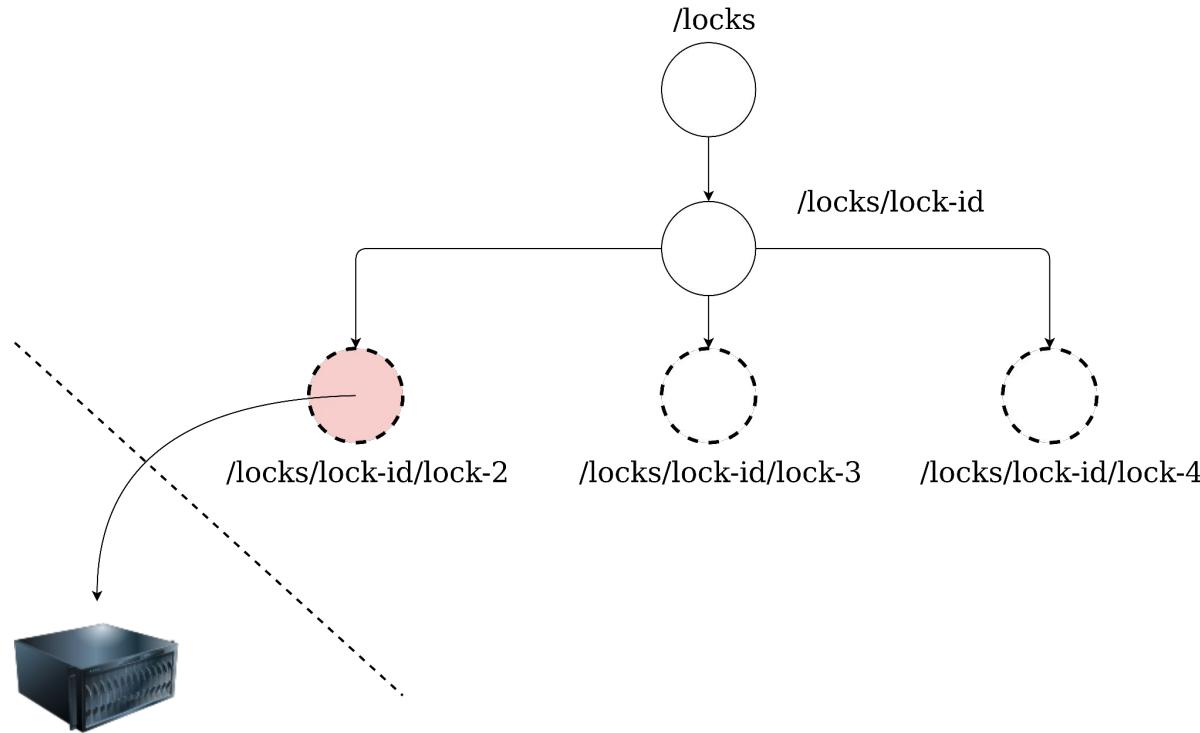
Блокировки и эфемерные узлы

- Используем в очереди эфемерные узлы
- При сбое клиент отключится от ZooKeeper
 - И его узел будет автоматически удалён
- Следующий в очереди сможет получить блокировку



Блокировки и эфемерные узлы

- Клиент может быть жив, но его сообщения могут не доходить до ZooKeeper
- Клиент считает, что у него всё ещё есть блокировка
- ZooKeeper уже удалил его узел
- Тем самым отдав блокировку другому узлу



Блокировки и эфемерные узлы: Leases

- Клиент считает: если уже T секунд он не получал от ZooKeeper ответы на хартбиты, то его отключило от сервера
 - Значит, он лишился блокировки
- ZooKeeper считает клиента отключенным если хартбиты от него не приходили уже $2 * T$ секунд
- Считаем, что часы синхронизированы достаточно точно
 - Не бывает такого, что клиент считает, что T секунд ещё не прошло
 - А сервер считает, что $2 * T$ секунд уже прошло

Блокировки и эфемерные узлы

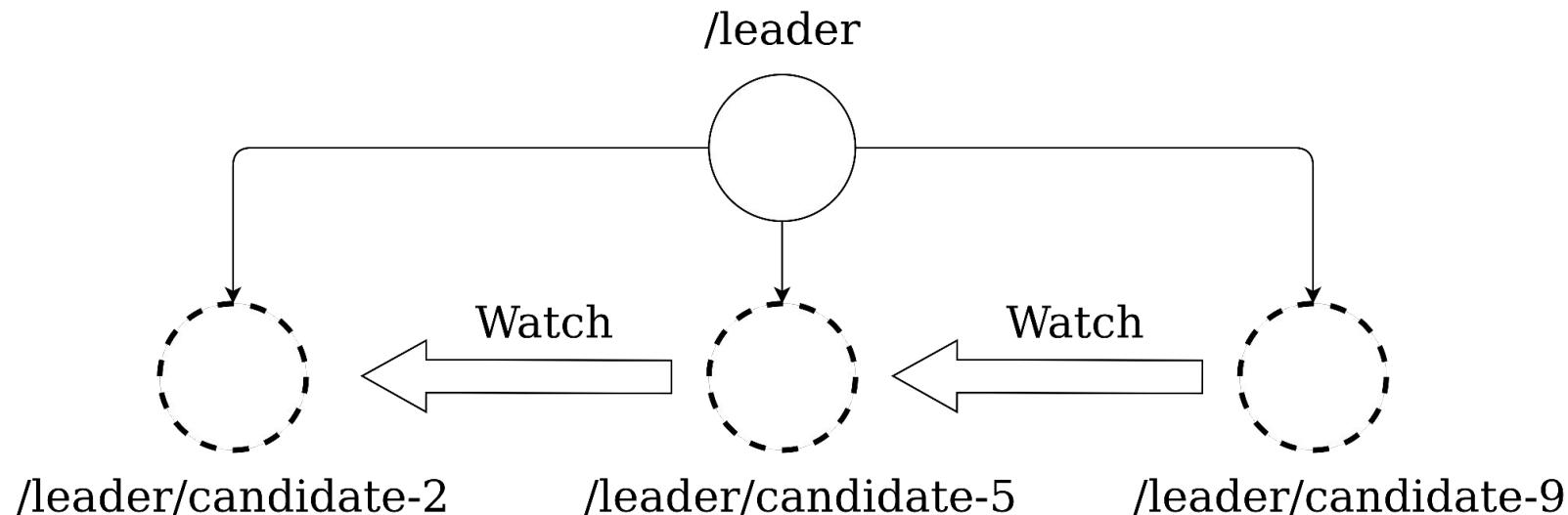
- Клиентский код:

```
zk_locker.Lock(file_lock)
file = fs.CreateFile("data.txt")
file.Write("Hello, ")
file.Write("world\n")
file.Close()
zk_locker.Unlock(file_lock)
```

- Клиент может лишиться блокировки, выполнив только часть действий
 - Система в неконсистентном состоянии
- Надо как-то с этим жить

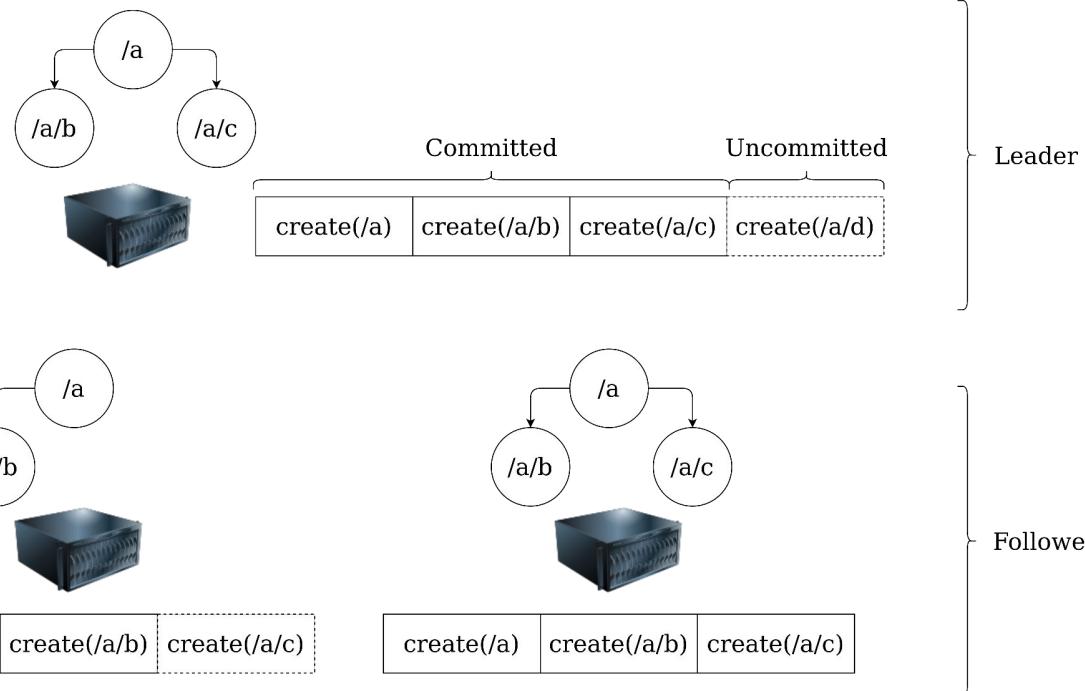
Выбор лидера

- Эквивалентен блокировке
 - У кого блокировка, тот и лидер
- Узлы должны быть эфемерные
 - Чтобы умерший лидер переставал быть лидером



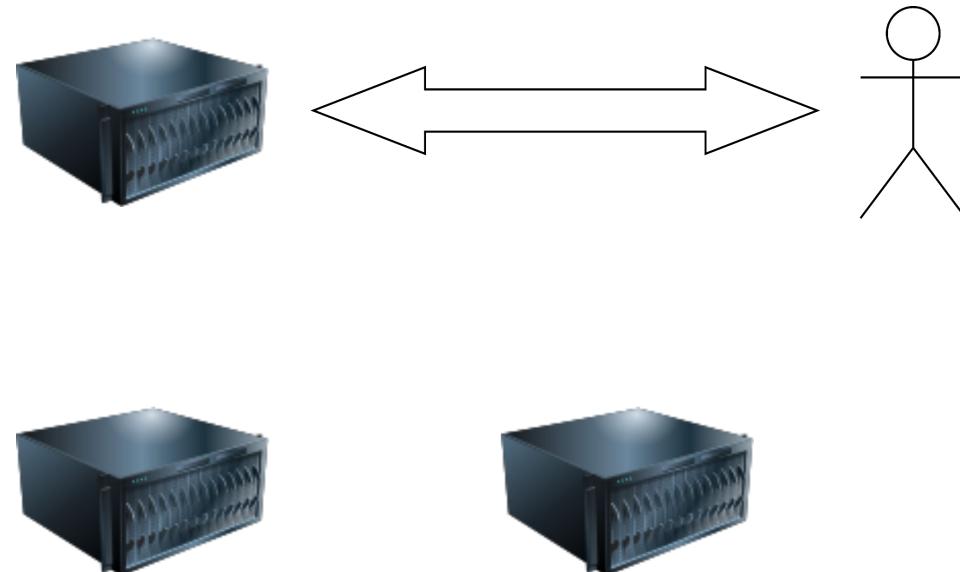
Распределённый ZooKeeper

- Кластер узлов
- Идейно очень похоже на Raft
 - Но не Raft
- Репликация журнала
- По журналу строим локальную структуру данных



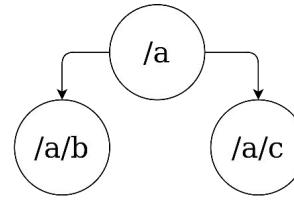
Взаимодействие с клиентом

- Клиент подключен к единственному узлу
 - Все команды направляет на этот узел
 - Только этот узел посылает клиенту уведомления
- Переподключается к другому узлу только при сбоях

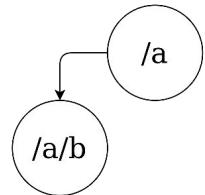


Распределённый ZooKeeper: чтение

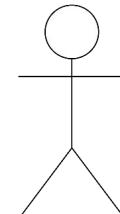
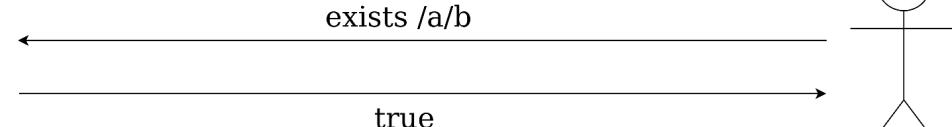
- Запросы на чтение работают локально
- Сервер смотрит в локальную структуру данных и отвечает



create(/a)	create(/a/b)	create(/a/c)	create(/a/d)
------------	--------------	--------------	--------------

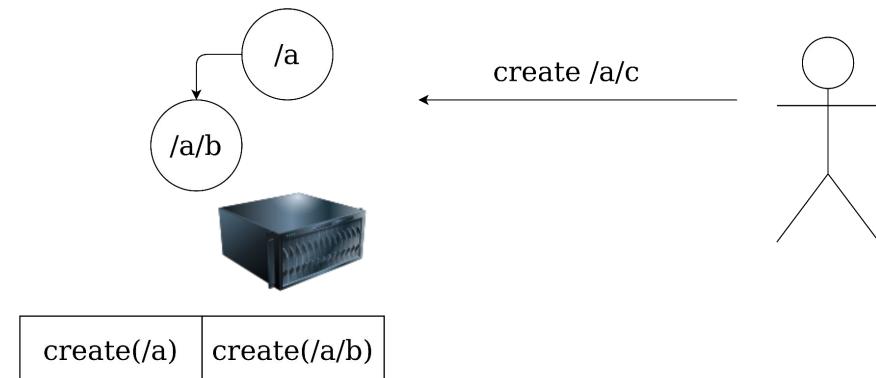
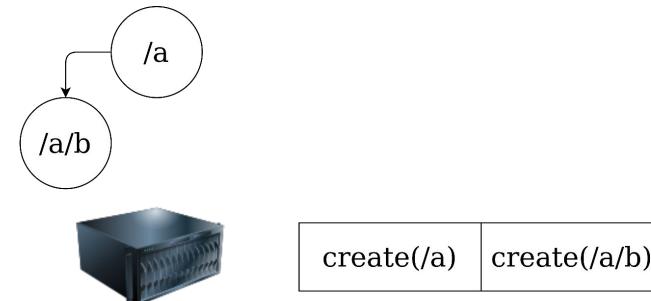


create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------



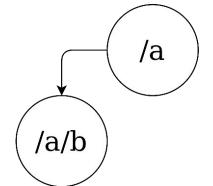
Распределённый ZooKeeper: записи

- Клиент делает запись на узел, к которому он подключён

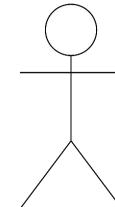
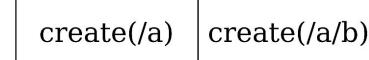
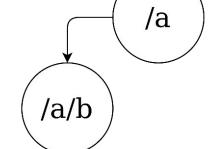


Распределённый ZooKeeper: записи

- Узел пересыпает запись лидеру
- Лидер добавляет её в лог
 - И помечает как незакоммиченную

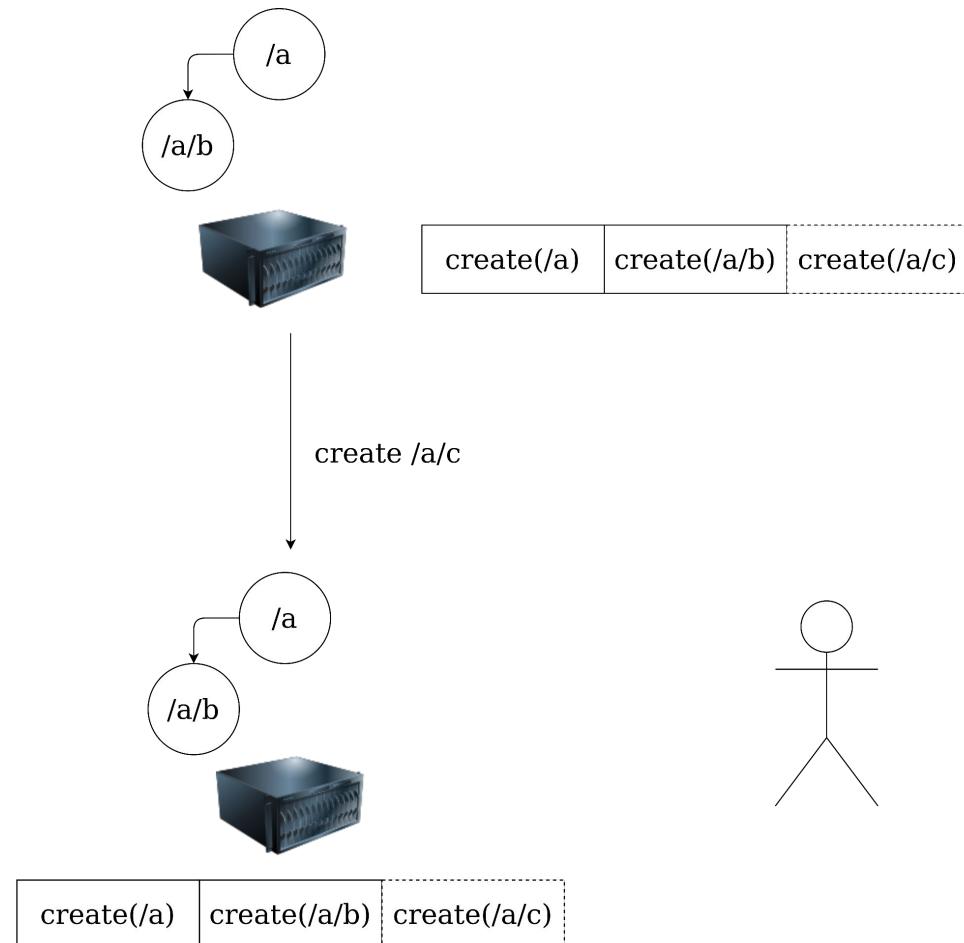


create /a/c



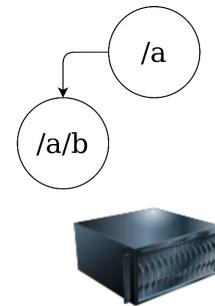
Распределённый ZooKeeper: записи

- Лидер реплицирует запись на кворум узлов
 - Пока что запись у всех не закоммичена

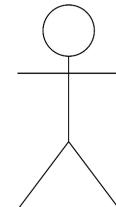
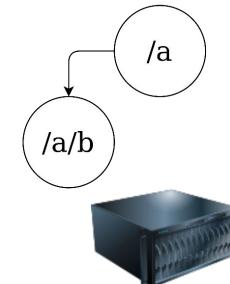


Распределённый ZooKeeper: записи

- Лидер помечает запись как закоммиченную
 - После того, как эта запись есть у кворума узлов



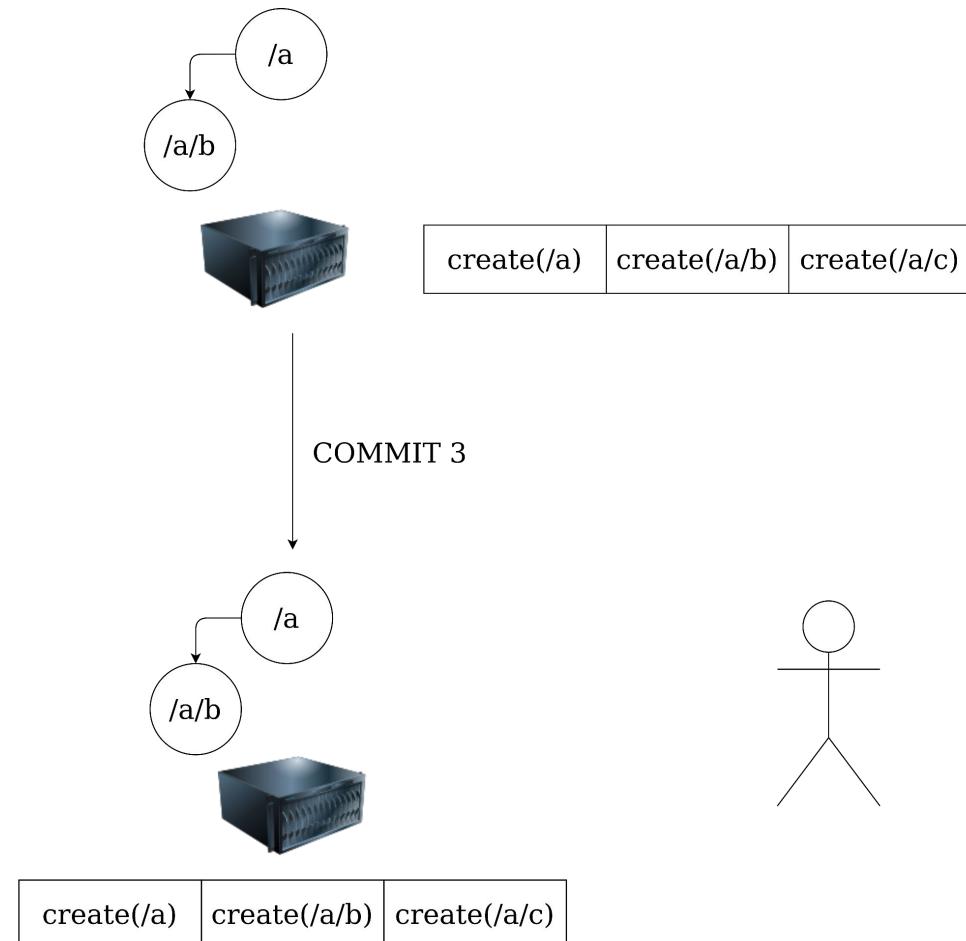
create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------



create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------

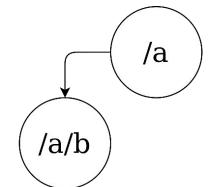
Распределённый ZooKeeper: записи

- Сообщает узлам о том, что эта запись закоммичена
 - В том числе и узлу, к которому подключён клиент

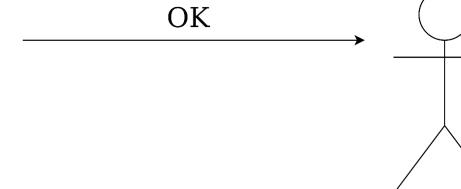
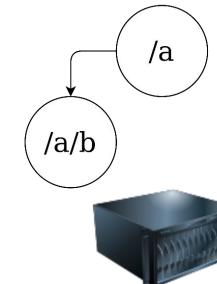


Распределённый ZooKeeper: записи

- Узел сообщает клиенту, что его запрос выполнен



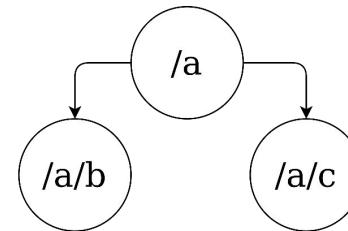
create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------



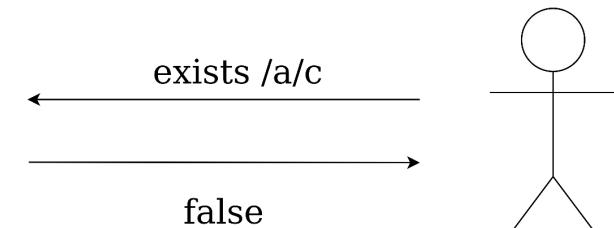
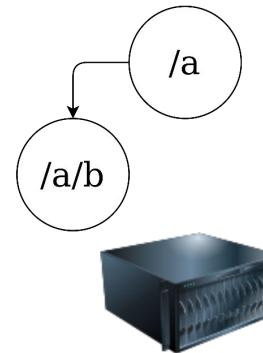
create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------

Чтение устаревших данных

- Чтения исполняются локально
 - Можно прочитать устаревшие данные



create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------



create(/a)	create(/a/b)	create(/a/c)
------------	--------------	--------------

Чтение устаревших данных: sync

- Настало время поговорить
- Каждый узел содержит префикс лога
 - “Дырок” быть не может
- Воспользуемся этим фактом



- sync()
 - Загадочная операция
 - Позже поговорим о ней

create(/a)	create(/a/b)	create(/a/c)	create(/a/d)
------------	--------------	--------------	--------------

create(/a)	create(/a/b)
------------	--------------

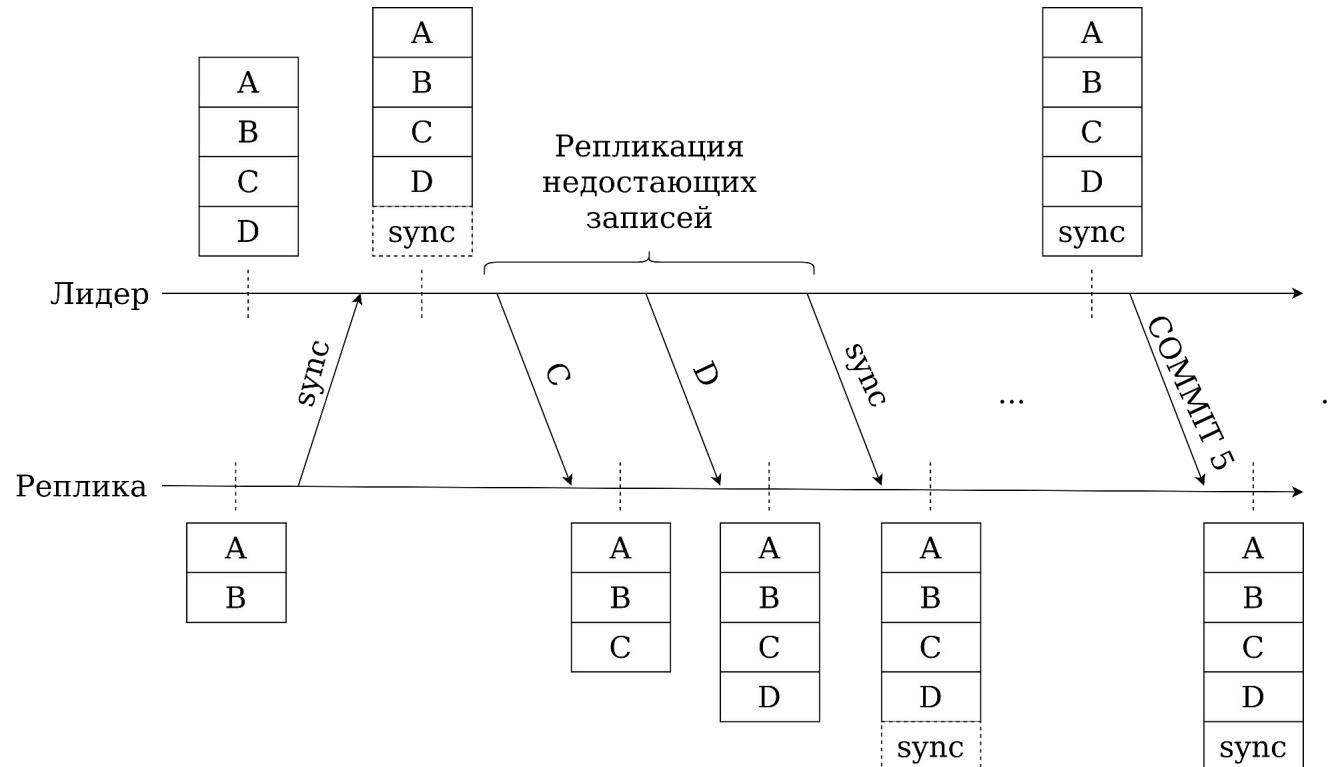


create(/a)	create(/a/b)	create(/a/c)	create(/a/d)
------------	--------------	--------------	--------------

create(/a)	???	create(/a/c)
------------	-----	--------------

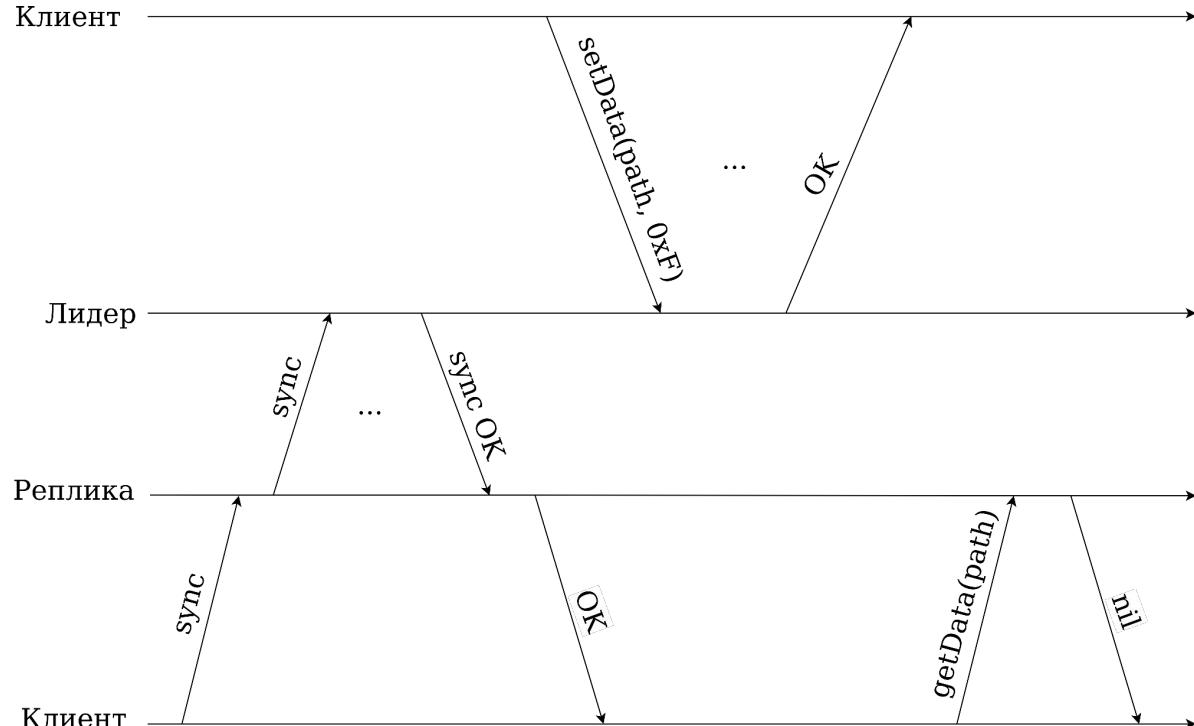
Чтение устаревших данных: sync

- sync не меняет состояние системы
- Но исполняется через кворумную репликацию
- На момент коммита sync у нас в логе есть все предыдущие записи



Чтение устаревших данных: sync

- `zk.Sync()`
`data := zk.GetData(path)`
- Прочитаем самые свежие данные?
 - Нет



Чтение устаревших данных

- Для чтения актуальных данных чтение нужно делать через кворум
 - Аналогично записи
 - Узел отправляет команду чтения лидеру
 - Ждёт, пока эта команда будет отреплицирована и закоммичена
 - Отправляет пользователю ответ
- `data := zk.GetDataQuorum(path)`
 - ZooKeeper такое не поддерживает

Чтение устаревших данных в блокировках

- Репликация осуществляется без “дырок”



create(/lock/lock-1)	create(/lock/lock-2)	create(/lock/lock-3)
----------------------	----------------------	----------------------



create(/lock/lock-1)	create(/lock/lock-2)
----------------------	----------------------



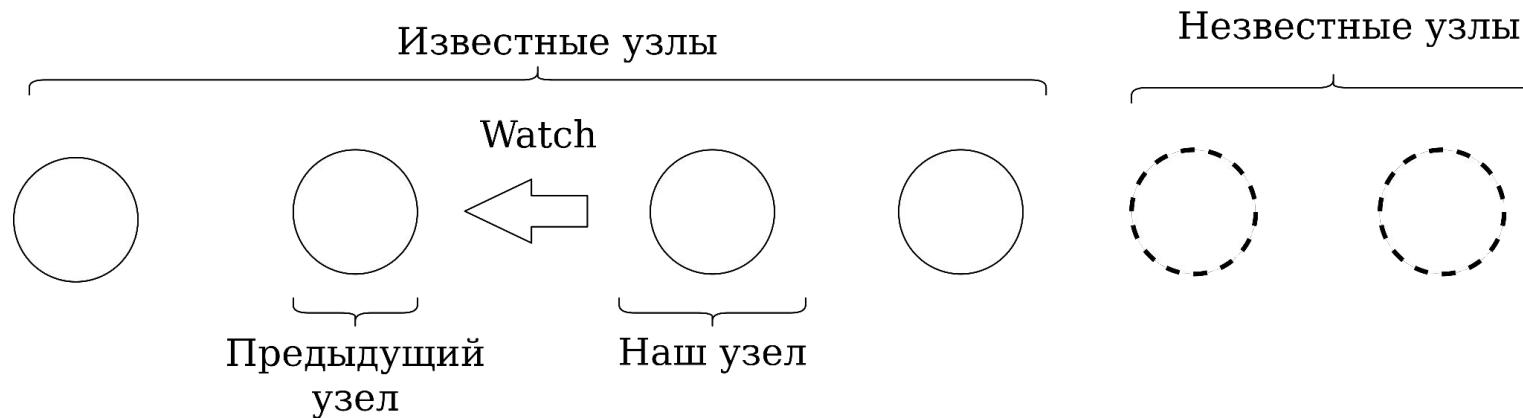
create(/lock/lock-1)	create(/lock/lock-2)	create(/lock/lock-3)
----------------------	----------------------	----------------------



create(/lock/lock-1)	???	create(/lock/lock-3)
----------------------	-----	----------------------

Чтение устаревших данных в блокировках

- Существует узел /lock/lock-N, что все узлы в очереди до него видимы, а все узлы после - нет
 - Нет чередования узлов
- Все узлы до нашего видимы нам
 - Так работает репликация
 - Только эти узлы могут взять блокировку до нас



Сбои узлов

- При сбое узлов клиент подключается к другому узлу
- У нового узла лог должен быть не короче, чем у предыдущего
 - Чтобы чтение было монотонным



entry ₁	entry ₂	entry ₃
--------------------	--------------------	--------------------



entry ₁	entry ₂
--------------------	--------------------



entry ₁	entry ₂	entry ₃	entry ₄
--------------------	--------------------	--------------------	--------------------

Сбои узлов: перенос уведомлений

- Переносим уведомления, которых мы ожидаем, на новый узел
- Часть из них могла сработать из-за операций на суффиксе
- Сообщаем об этом клиенту перед началом нормальной работы
- Остальные уведомления сохраняем

Журнал на
предыдущем узле

Ищем сработавшие
уведомления

entry ₁	entry ₂	entry ₃	entry ₄	entry ₅	entry ₆	entry ₇
--------------------	--------------------	--------------------	--------------------	--------------------	--------------------	--------------------

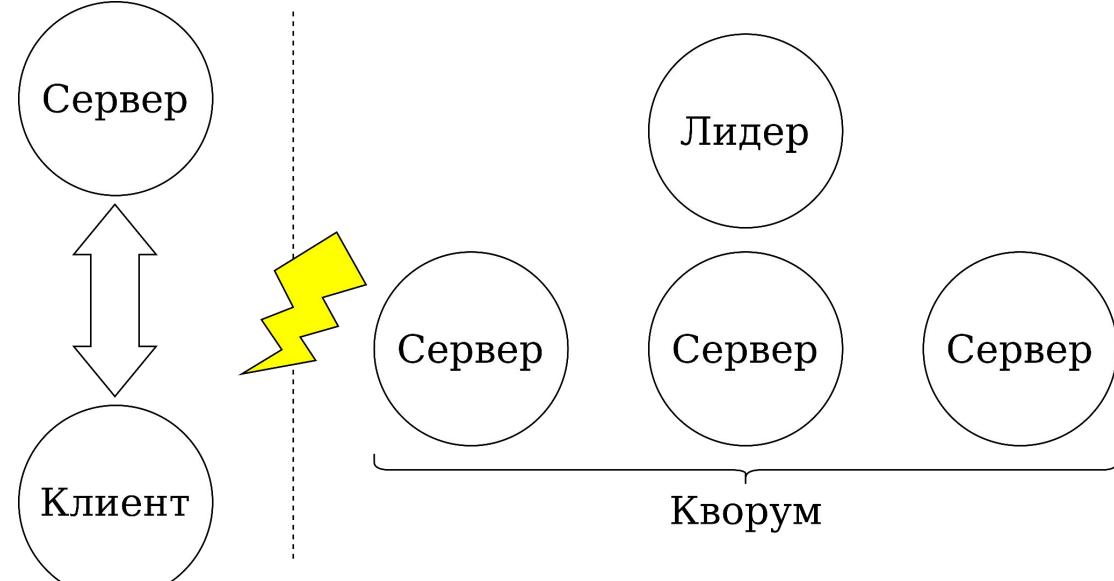
Журнал на новом узле

Эфемерные узлы

- Лидер следит за состоянием клиентов, которые создали эфемерные узлы
 - Удаляет эфемерные узлы, если долго не было хартбитов

Эфемерные узлы и разделение сети

- Клиент подключён к живому серверу, но этот сервер изолирован от лидера
- Возникает угроза удаления эфемерных узлов
- Сервер сообщает об этом клиенту
- Клиент подключается к другому серверу
- Или отказывается от блокировки/лидерства/etc



Программирование ZooKeeper

- При создании клиента передаём ему коллбек
 - Будет следить за состоянием соединения
- Watch - это тоже коллбек

- Логика работы в трёх местах

```
1 zk := ZooKeeper(host, port, event -> {
2     if event.Type = DISCONNECTED:
3         // Disconnect processing logic here
4     elif event.Type = CONNECTED:
5         // Connect processing logic here
6 }
7 // Coordination logic here
8 data := zk.GetData(path, event -> {
9     // Watch logic here
10 })
```

Программирование ZooKeeper

- Любой коллбек просто записывает данные в канал
- Мы забираем из канала события по одному
 - И реагируем на них
- Вся логика работы в теле цикла

```
1 channel := new Channel<Event>()
2 zk := ZooKeeper(host, port, event -> channel.send(event))
3 while true:
4     cur_event := channel.receive()
5     if cur_event.Type = CONNECTED:
6         data := zk.GetData("/root", event -> channel.send(event))
7     elif cur_event.Type = DISCONNECTED:
8         ...
9     elif cur_event.Type = WATCH_TRIGGERED:
10        ...
```

Team ZooKeeper



Apache
MESOS™

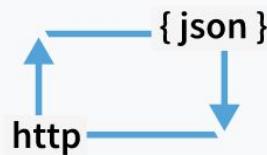


Аналоги ZooKeeper: etcd

Features

Simple interface

Read and write values using standard HTTP tools, such as curl



Key-value storage

Store data in hierarchically organized directories, as in a standard filesystem

```
/config  
└── database  
└── feature-flags  
    └── verbose-logging  
└── redesign
```

Watch for changes

Watch specific keys or directories for changes and react to changes in values



Optional SSL client certificate authentication



Benchmarked at 1000s of writes/s per instance



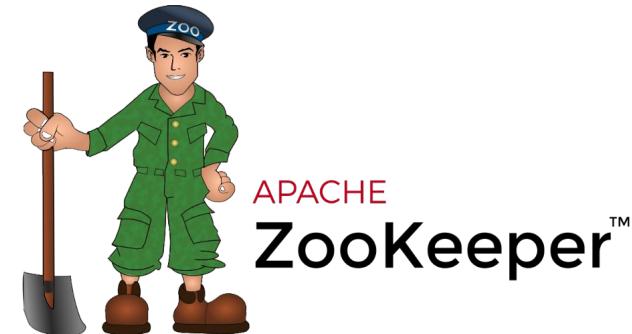
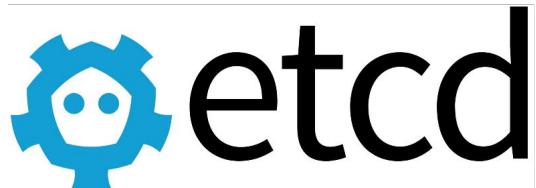
Optional TTLs for keys expiration



Properly distributed via Raft protocol

Аналоги ZooKeeper: etcd

- Зачем на слайде
две одинаковые
пары картинок?



Что почитать: ZooKeeper и ZAB

- [ZooKeeper Programmer's guide](#)
- [Jepsen: Zookeeper](#)
- *Hunt P. et al.* ZooKeeper: Wait-free Coordination for Internet-scale Systems
- *Junqueira F., Reed B.* ZooKeeper: distributed process coordination
- *Junqueira F. P., Reed B. C., Serafini M.* Zab: High-performance broadcast for primary-backup systems
- *Medeiros A.* ZooKeeper's atomic broadcast protocol: Theory and practice
- *Reed B., Junqueira F. P.* A simple totally ordered broadcast protocol

Что почитать: Другие системы координации

- *Burrows M.* The Chubby lock service for loosely-coupled distributed systems
- *Chandra T. D., Griesemer R., Redstone J.* Paxos made live: an engineering perspective
- etcd.io

Thanks for your attention

