

Raft

Физические часы

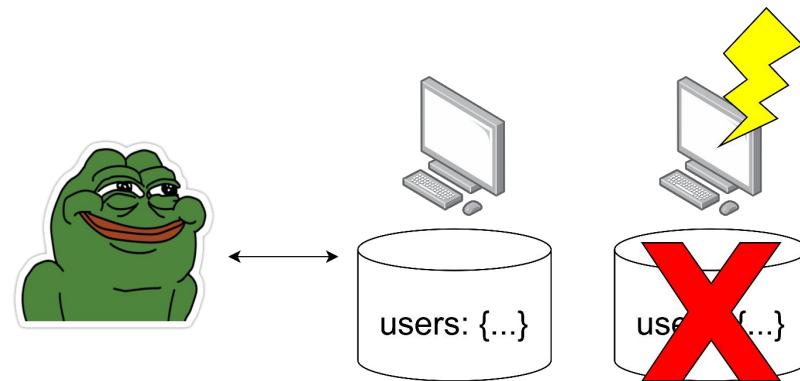
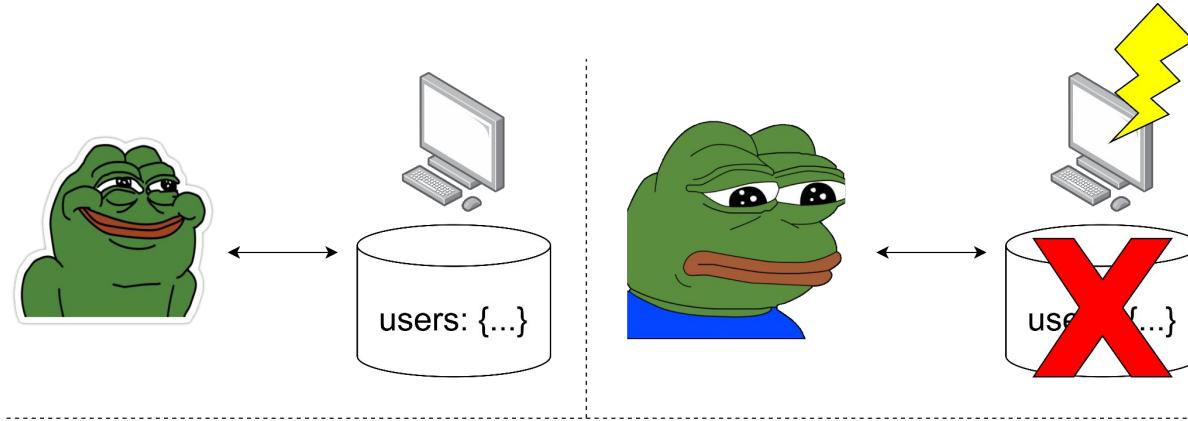


Илья Кокорин

kokorin.ilya.1998@gmail.com

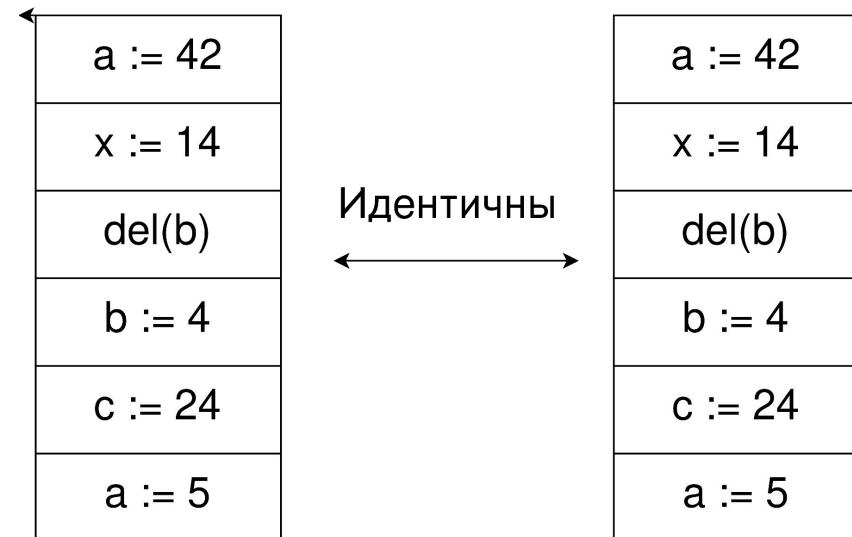
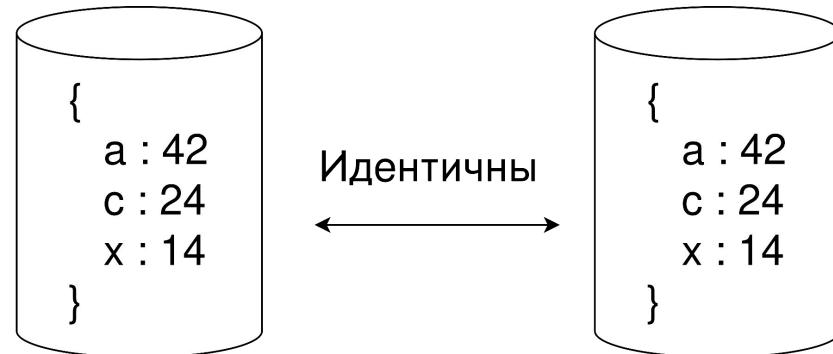
Репликация

- Хотим увеличить надёжность системы



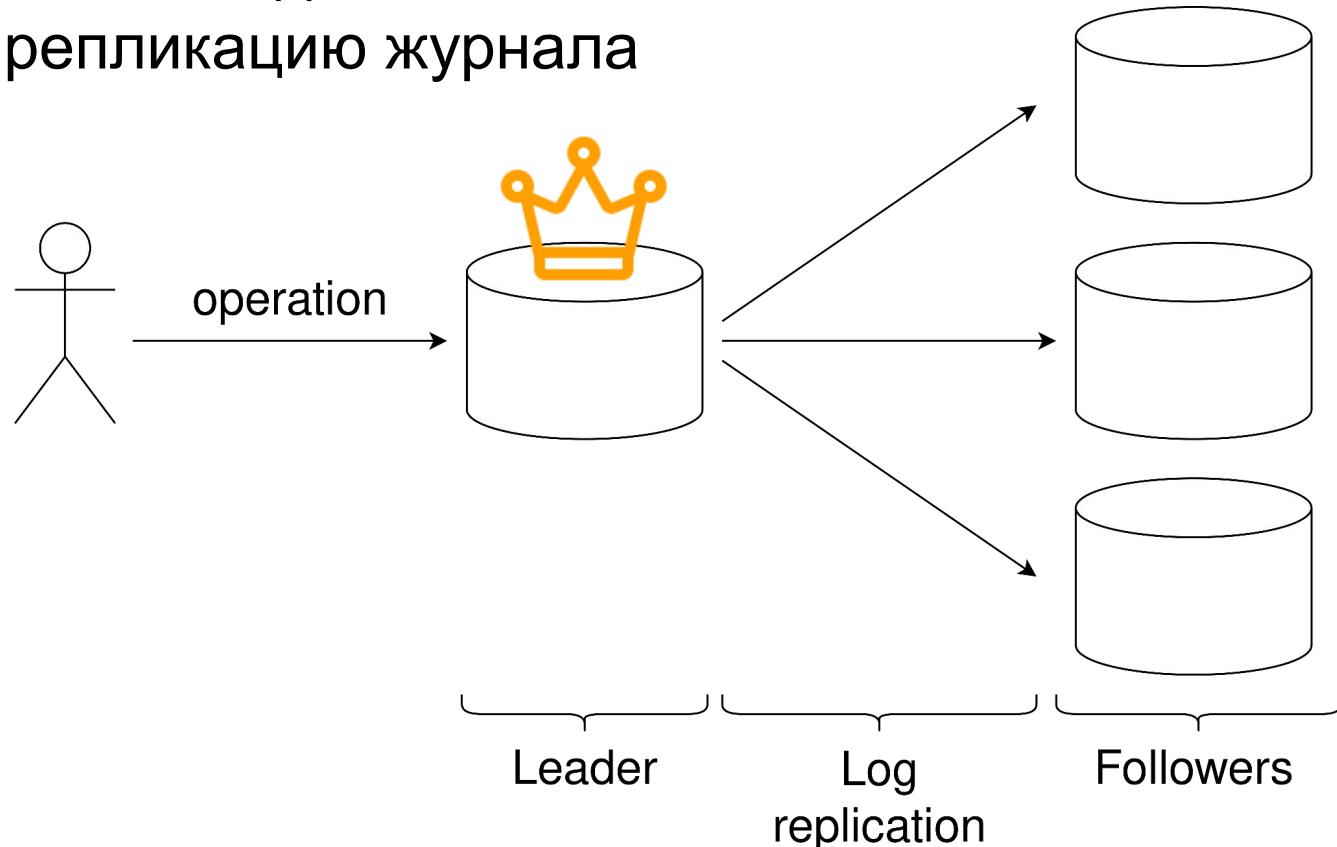
Replicated State Machines

- Структура данных детерминировано восстанавливается из журнала операций
- Добиваемся идентичности журналов на разных узлах
- Структуры данных тоже будут идентичны
- Решаем задачу репликации журнала



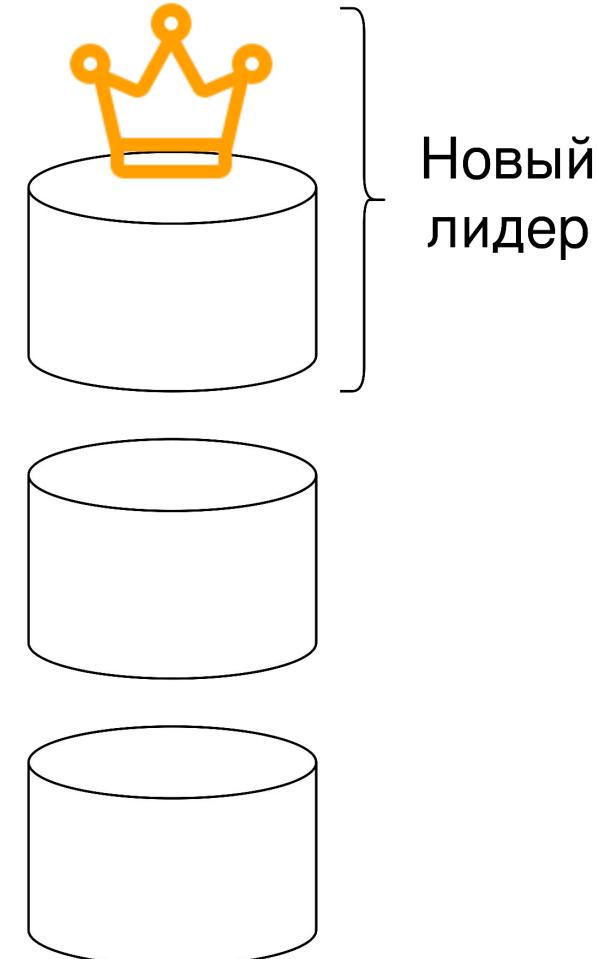
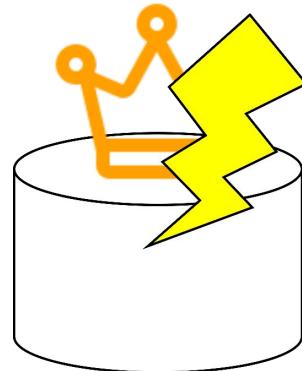
Raft: структура алгоритма

- Лидер принимает команды от пользователей
- Осуществляет репликацию журнала
- Остальные сервера принимают записи от лидера



Raft: перевыборы

- В случае сбоя лидера узлы выбирают нового



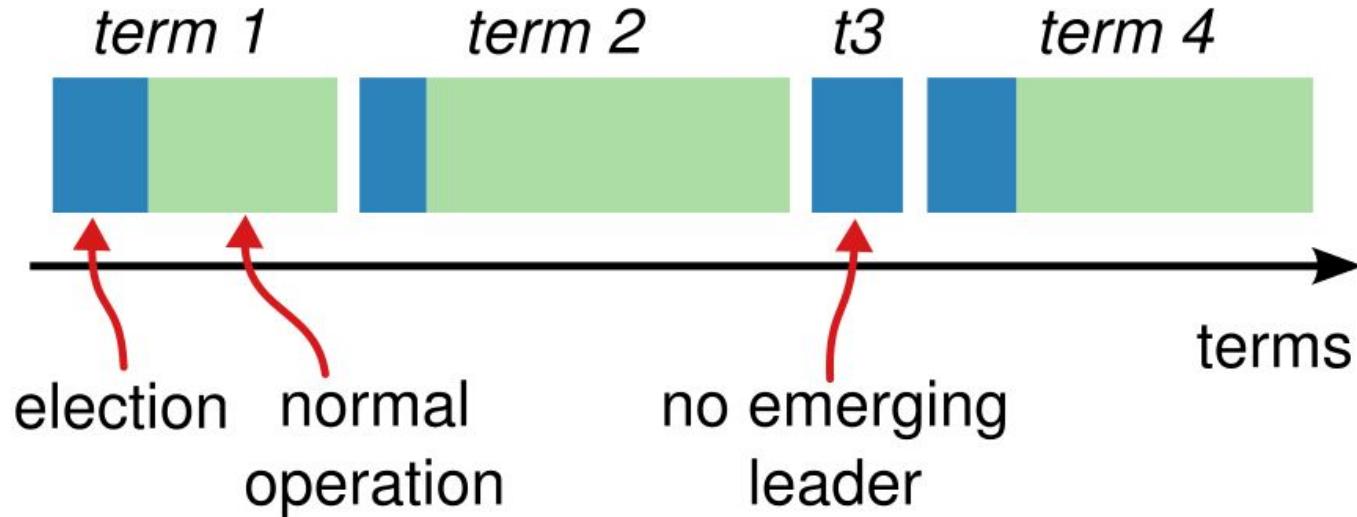
LOONEY TUNES



“That’s all Folks!”

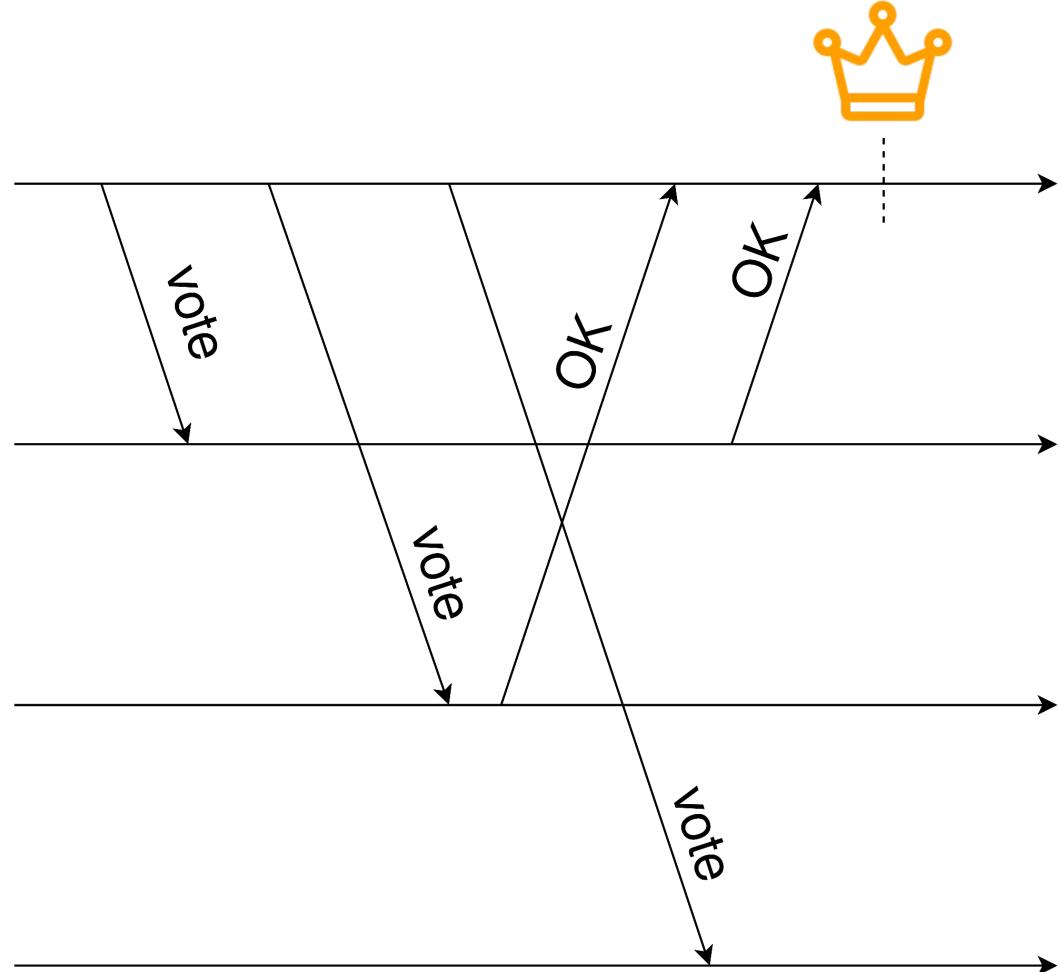
Raft: сроки и выборы

- Исполнение алгоритма делится на *сроки*
- Нумеруются возрастающими целыми числами
- Сначала выбираем одного лидера, потом реплицируем журнал
- Иногда выбрать лидера не удаётся



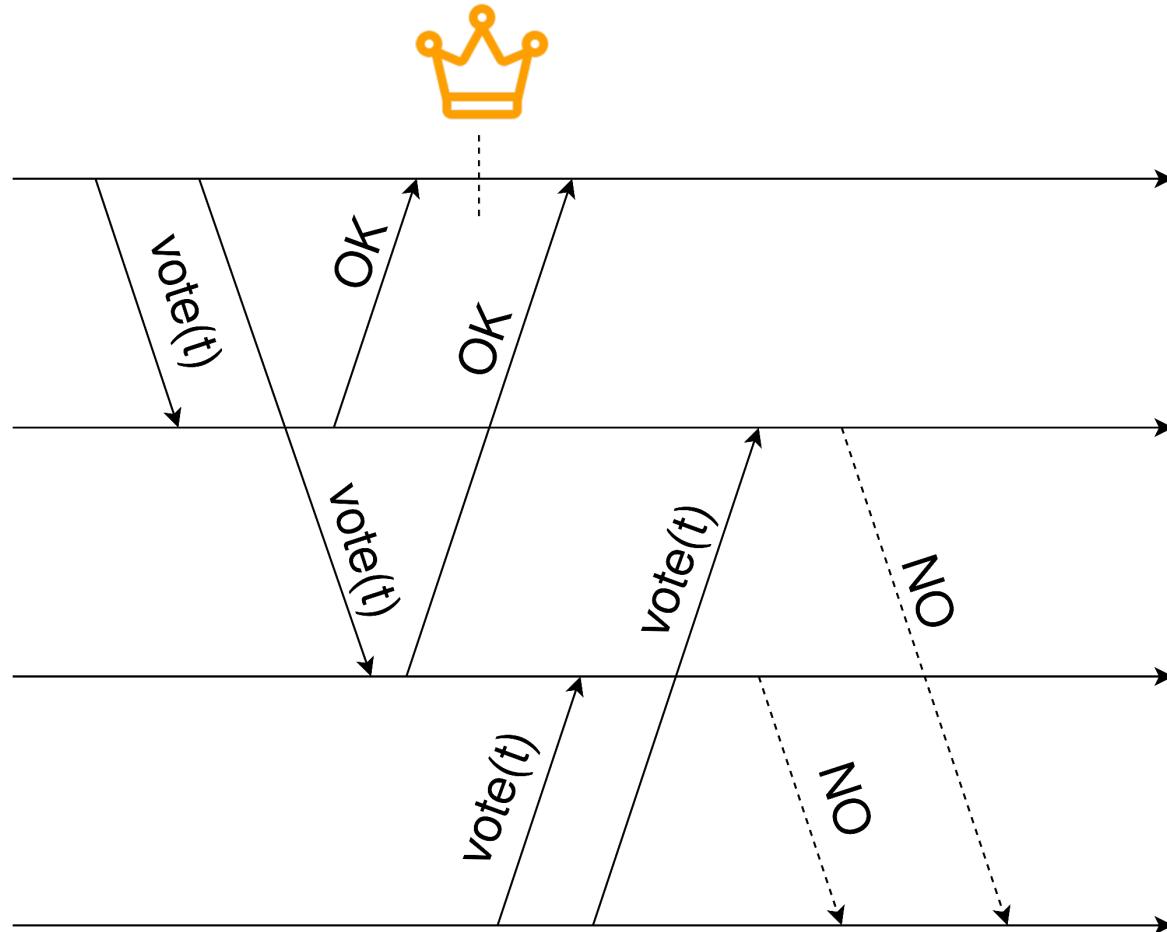
Raft: выборы

- Запрашиваем голоса у всех серверов
- Получив голоса у кворума, становимся лидером
- Голоса всех серверов не нужны, так как возможны сбои



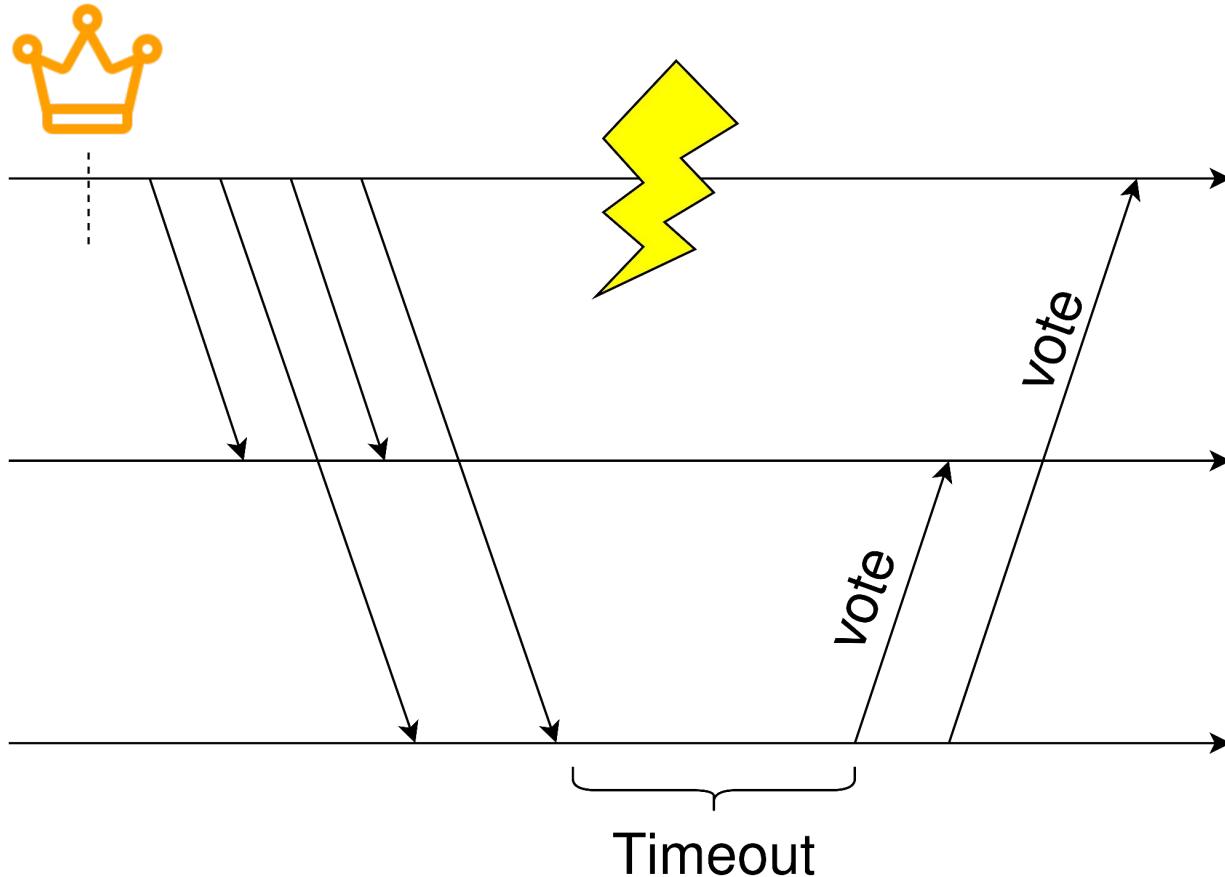
Raft: выборы

- В терме t процесс может отдать голос только первому процессу
- Включая себя
- На последующие запросы не отвечает
- Либо отвечает “нет”



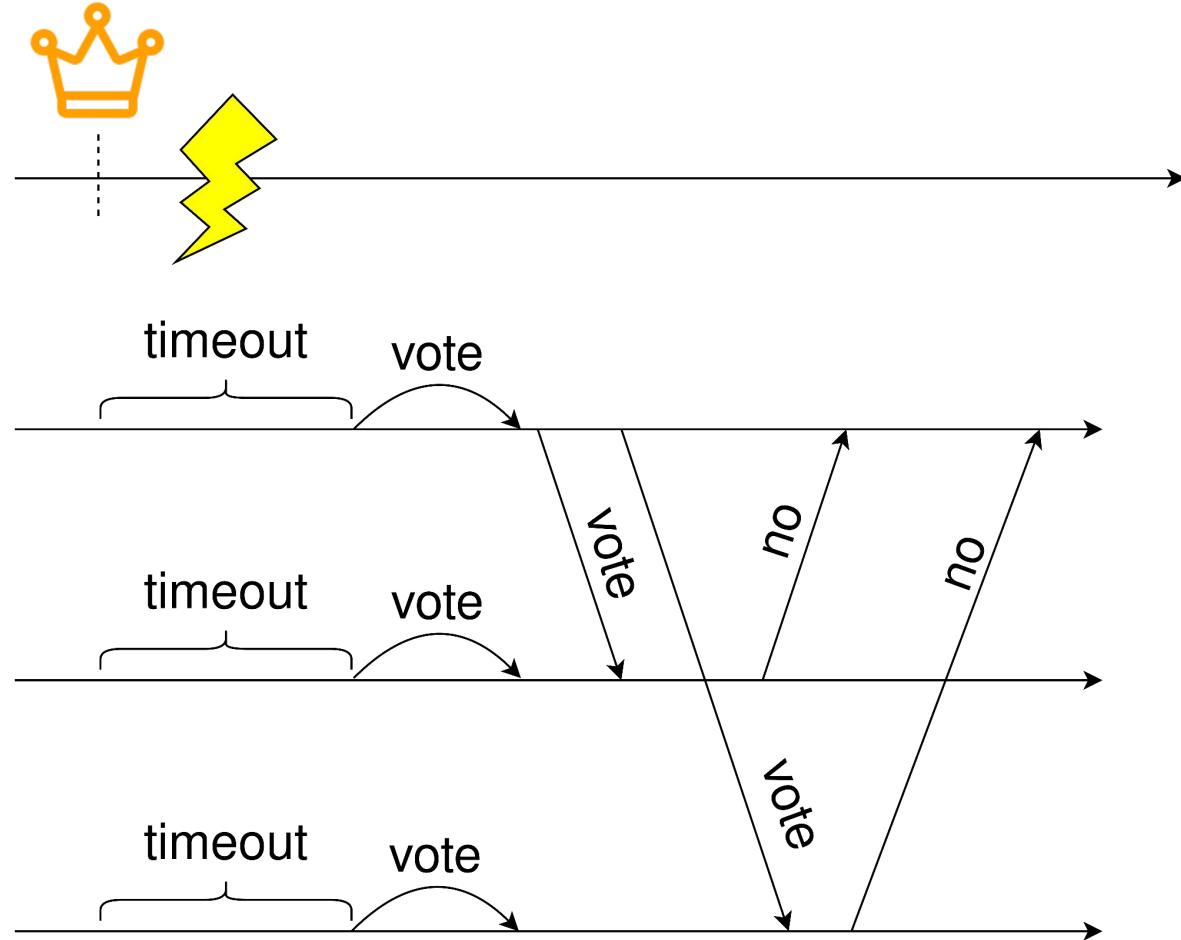
Raft: election timeout

- Если процесс не получал сообщений от лидера уже какое-то время, он считает лидера отказавшим
- Становится кандидатом



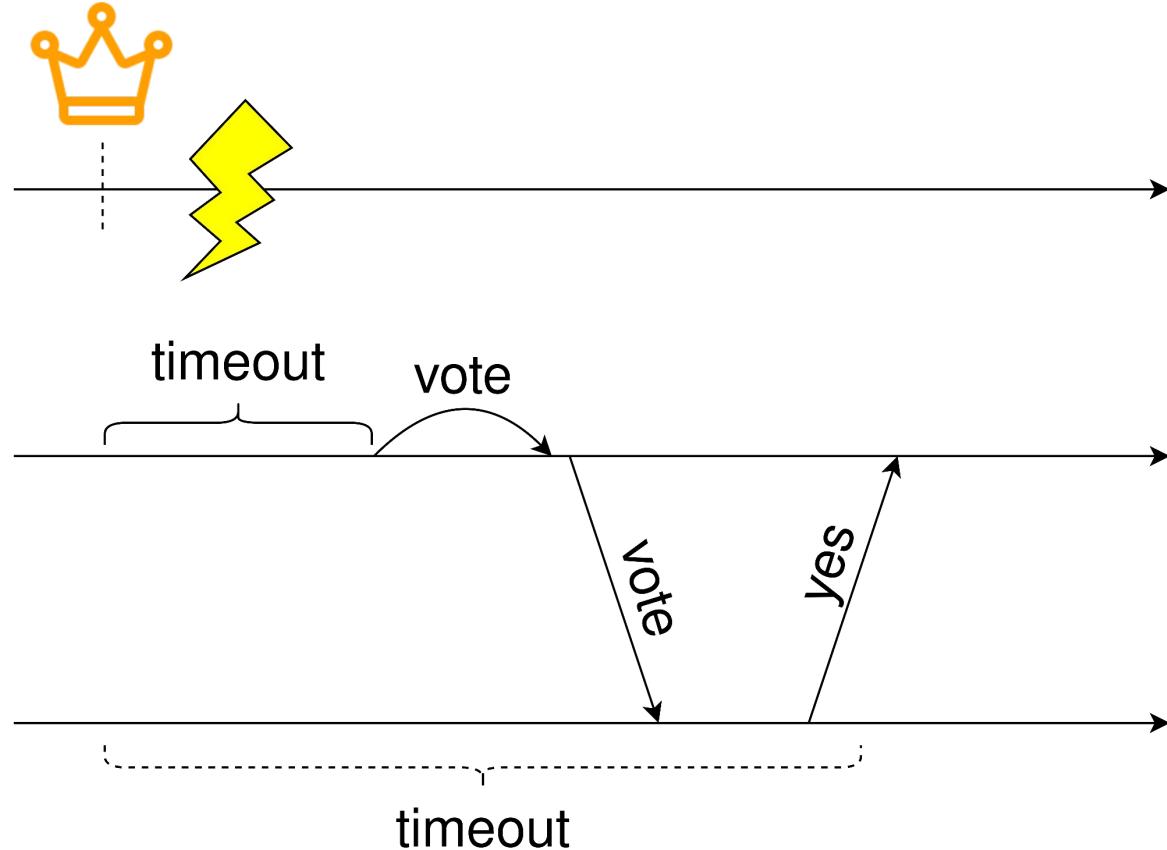
Raft: election timeout

- Все процессы становятся кандидатами в одно время
- Никто не может собрать половину голосов



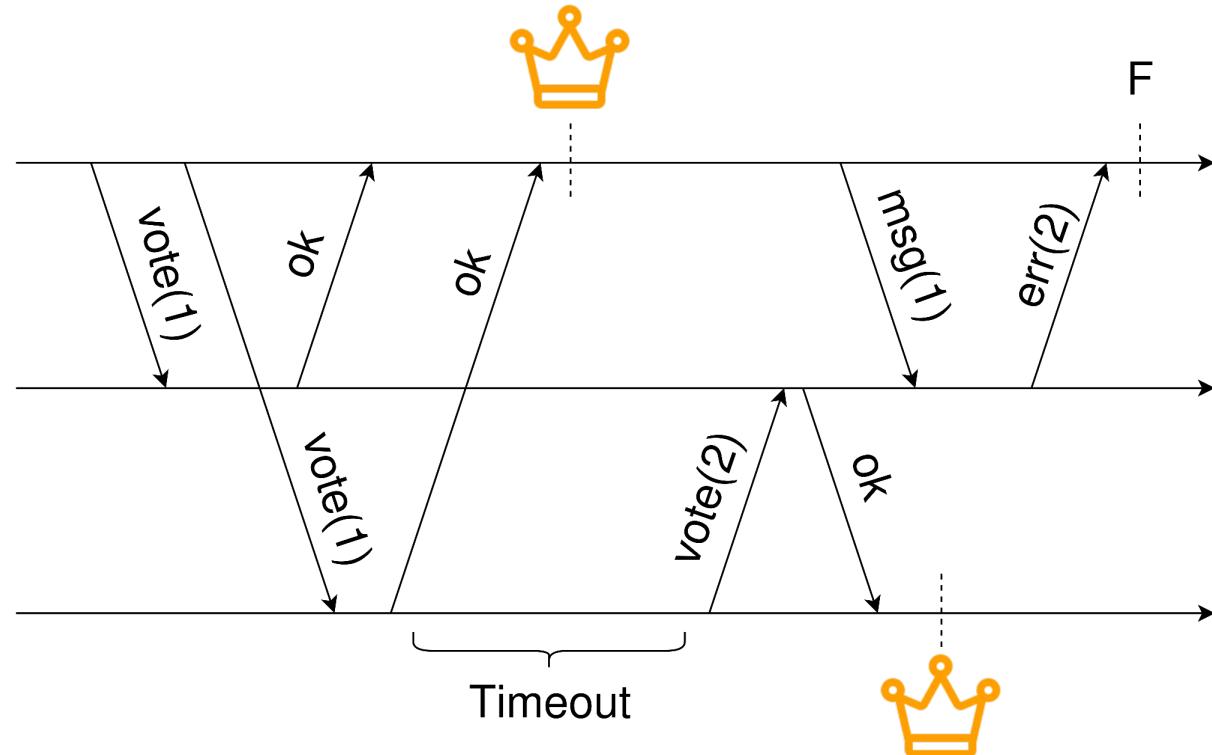
Raft: randomized election timeout

- Процесс становится кандидатом через случайное время
- С большой вероятностью, первый ставший успеет собрать голоса



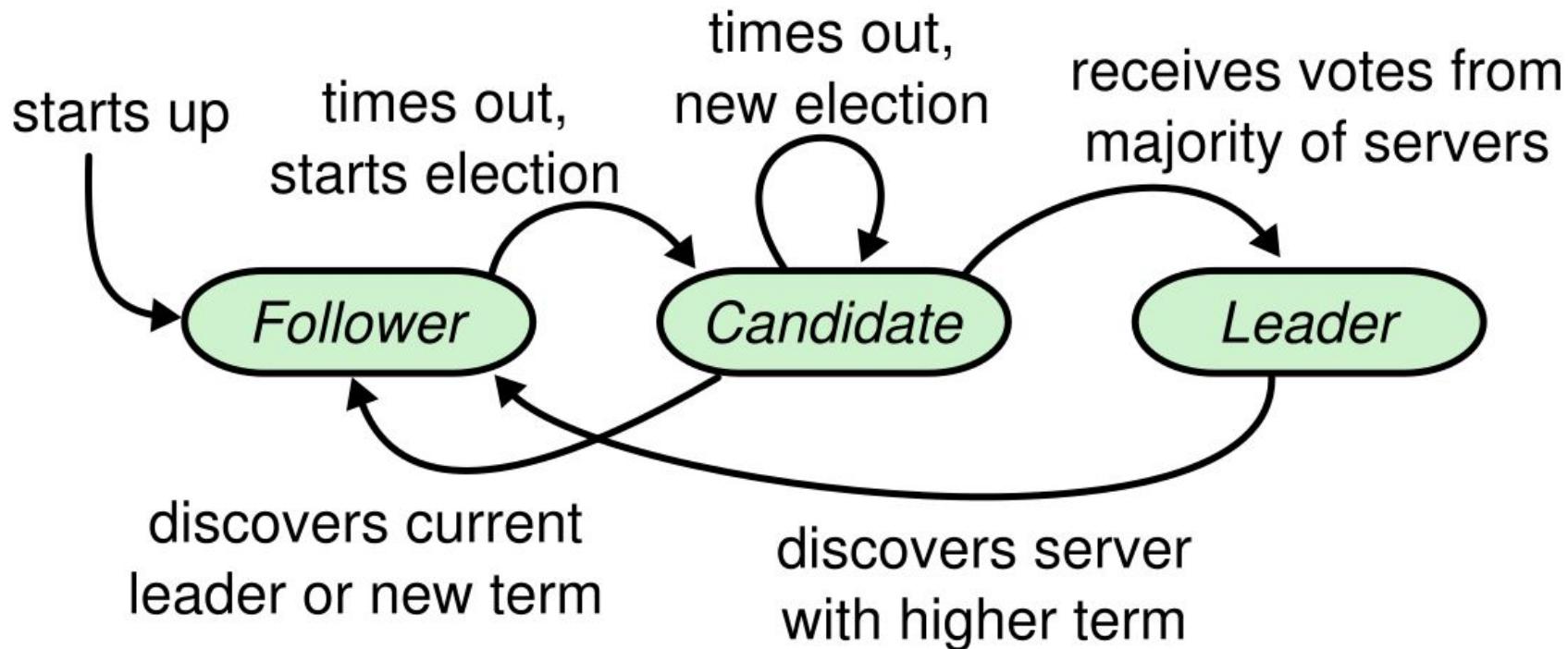
Raft: устаревшие термы

- Каждый процесс знает текущий терм
- Отвечает ошибкой, получив сообщение из старого терма
- Лидер старого терма отказывается от лидерства



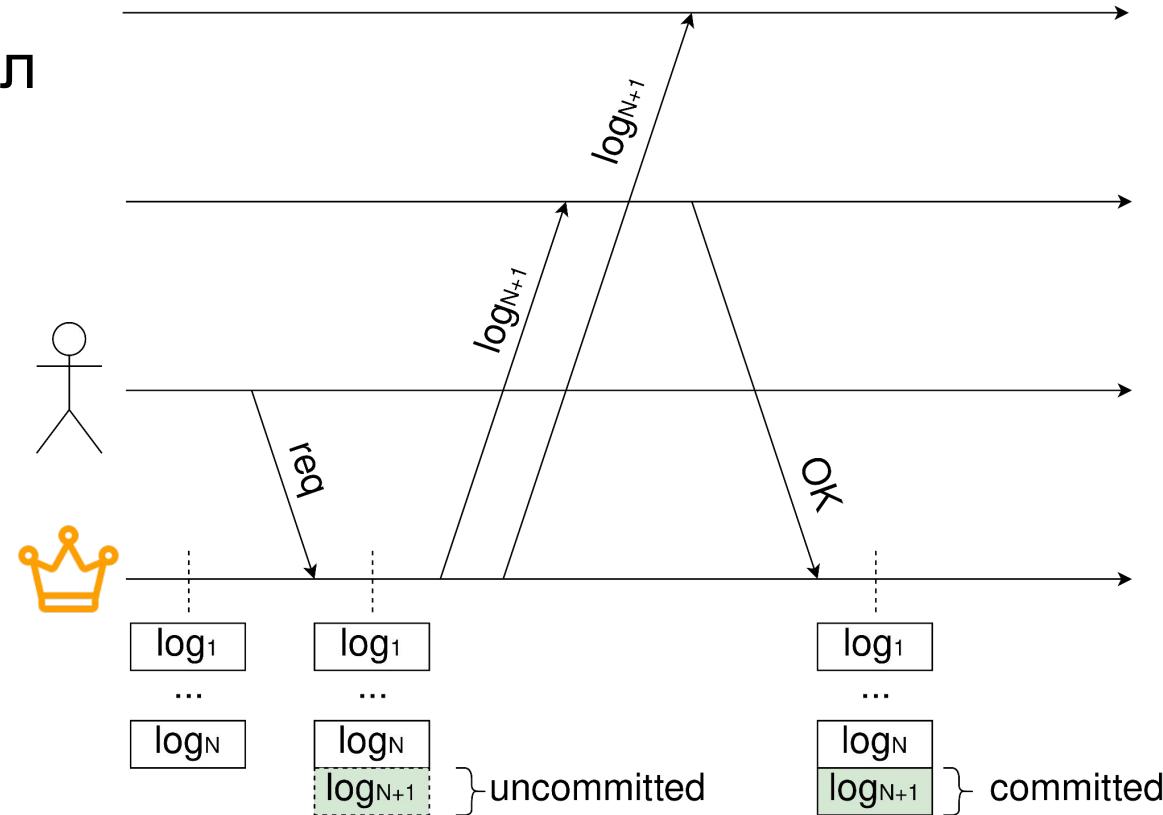
Raft: выбор лидера

- Простая диаграмма из трёх состояний



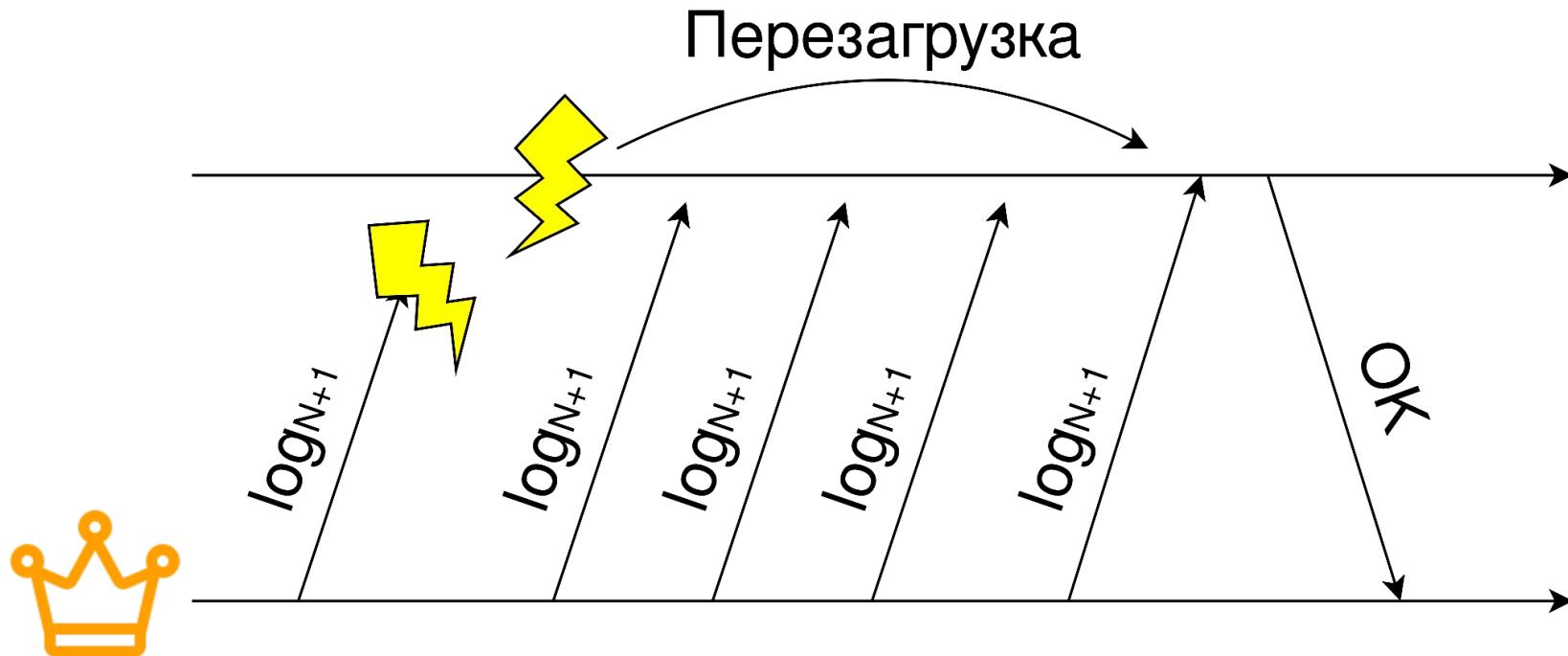
Raft: репликация журналов

- Клиент присыпает операцию лидеру
- Лидер добавляет запись в свой журнал
- Помечает как незакоммиченную
- Пересыпает запись всем участникам
- Реплицировав на кворум, помечает запись как закоммиченную



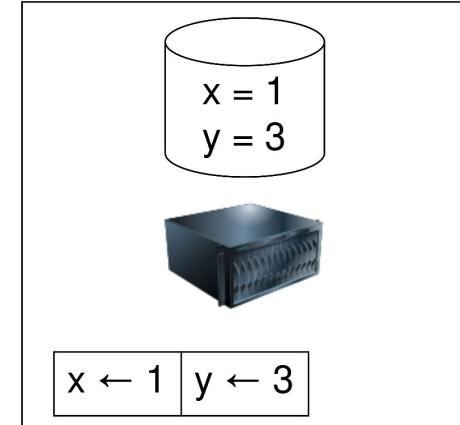
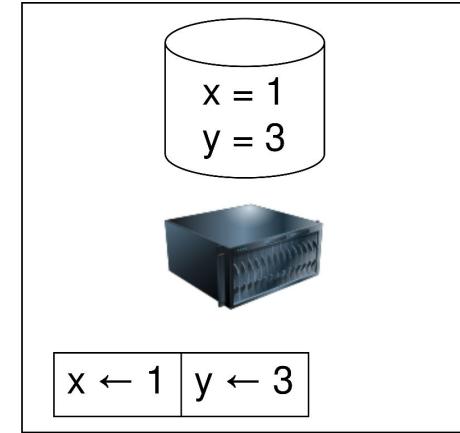
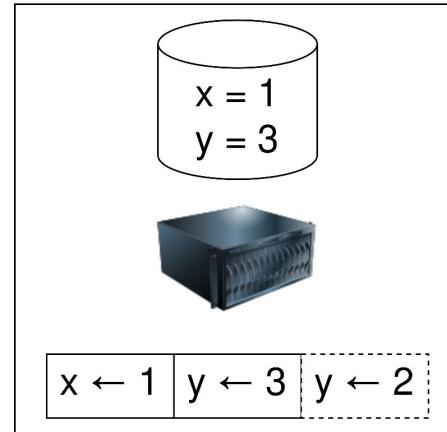
Raft: репликация журналов

- Реплицируем журнал на участника пока он не подтвердит запись



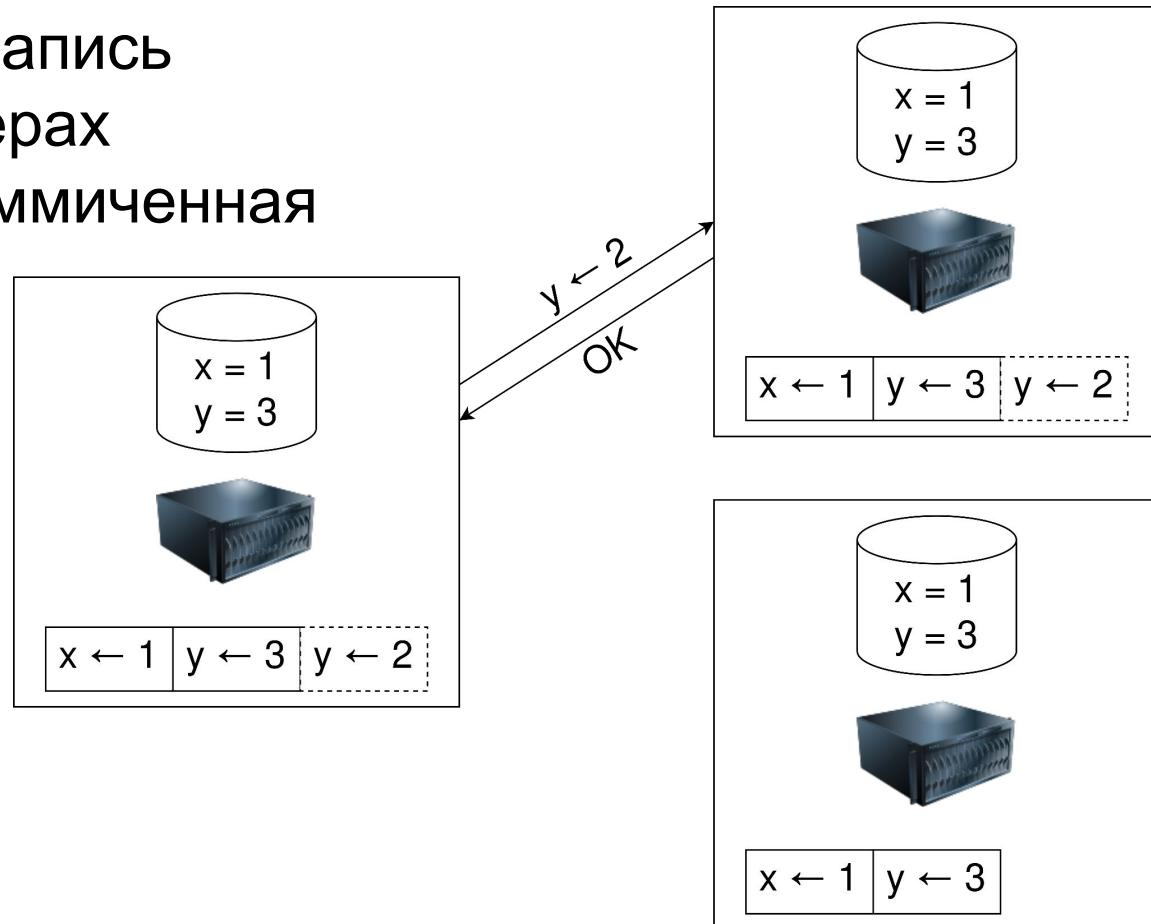
Raft: коммит записей

- Изначально запись есть только у лидера
- Помечена как незакоммиченная
- Лидер только добавляет записи в свой журнал
- Никогда не перезаписывает
- Остальные участники могут перезаписывать



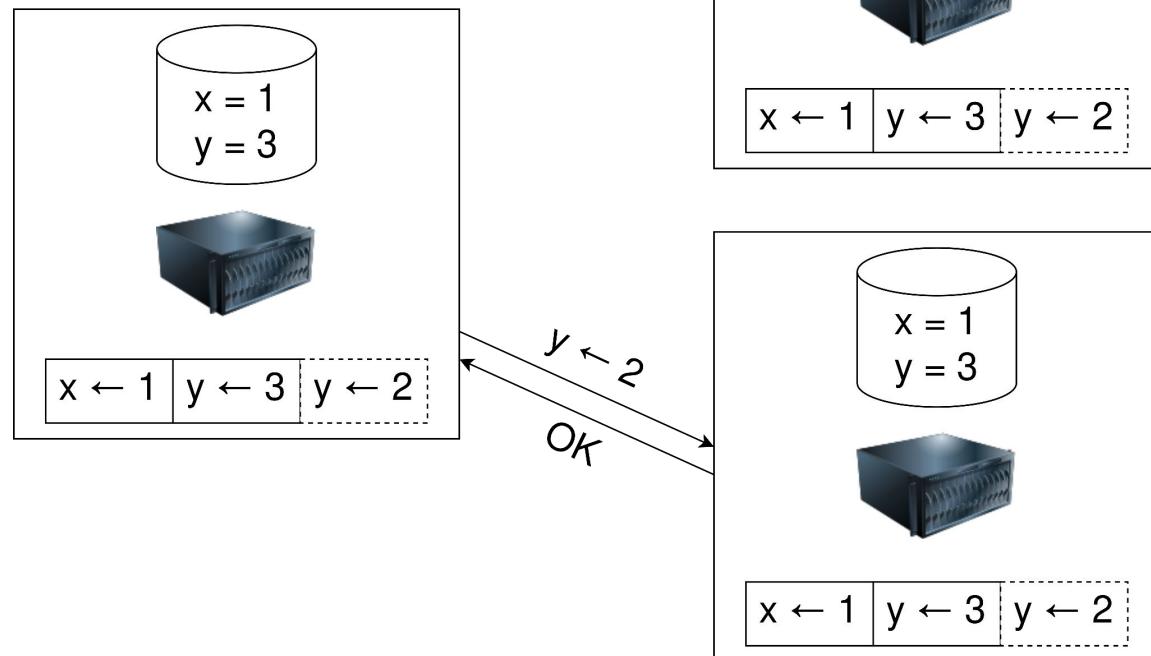
Raft: коммит записей

- Лидер реплицирует запись
- Запись на всех серверах помечена как незакоммиченная



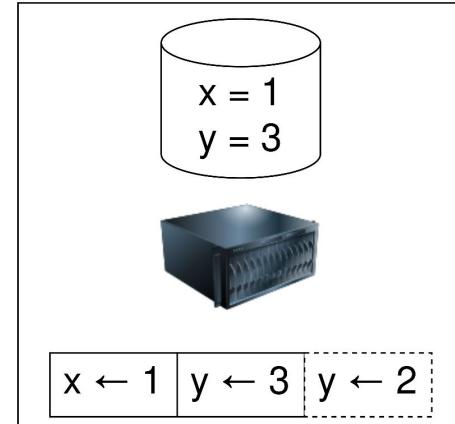
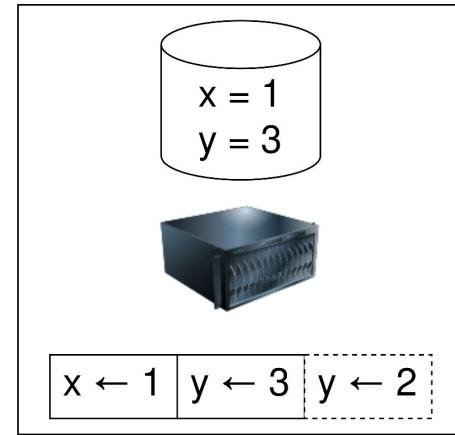
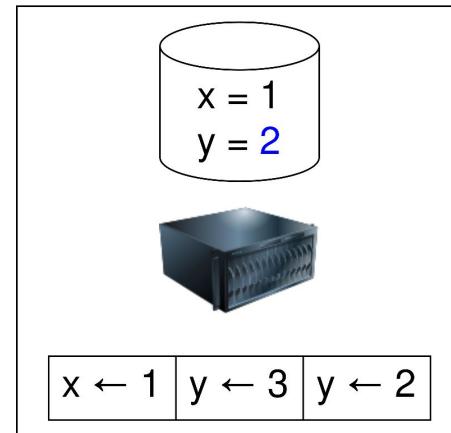
Raft: коммит записей

- Лидер заканчивает репликацию на кворум



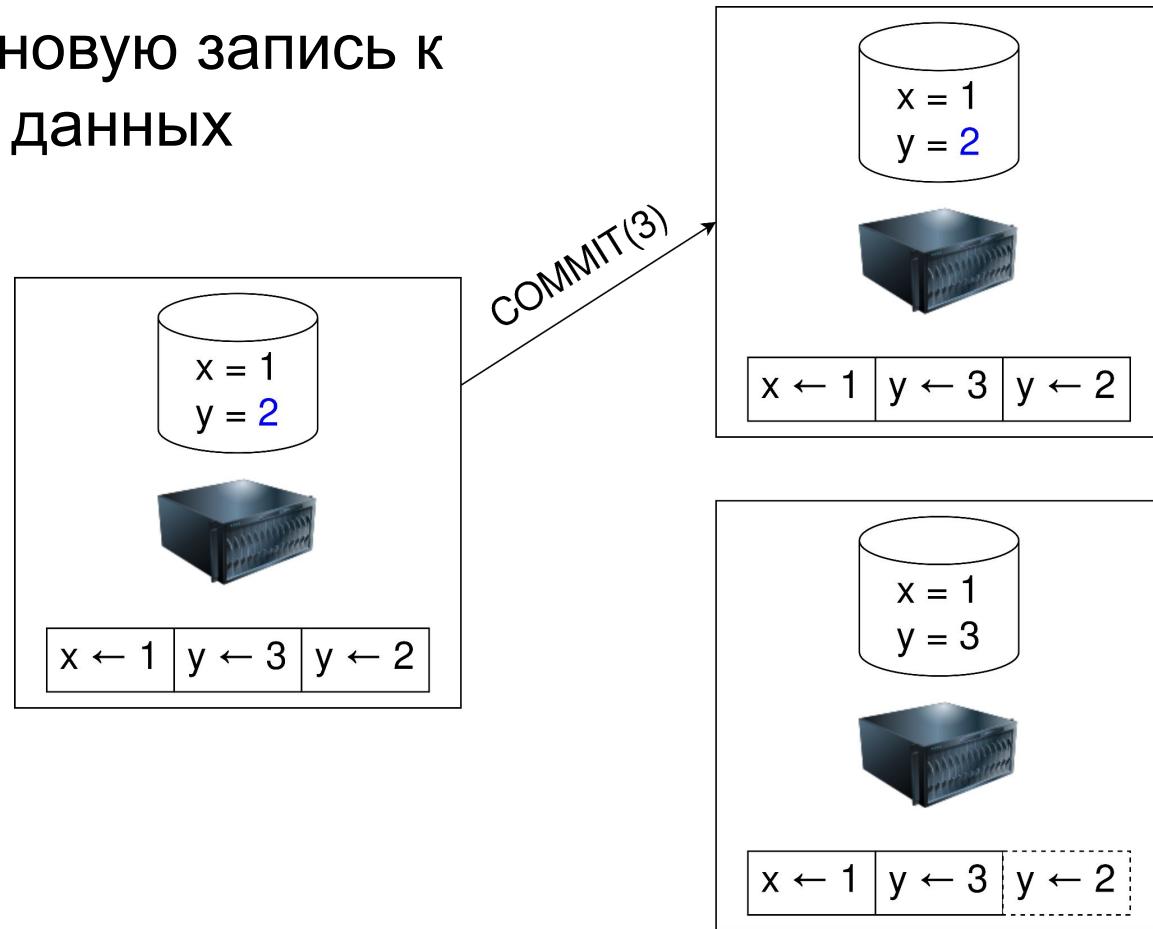
Raft: коммит записей

- Лидер применяет новую запись к локальной структуре данных
- Запись есть у кворума серверов — значит, навсегда останется на своей позиции
- Не будет перезаписана



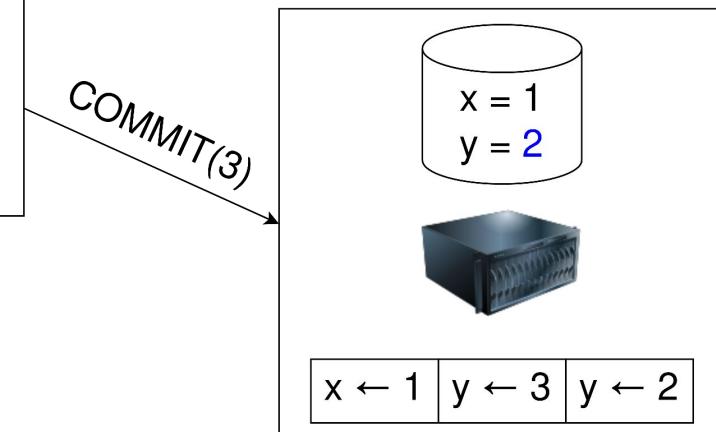
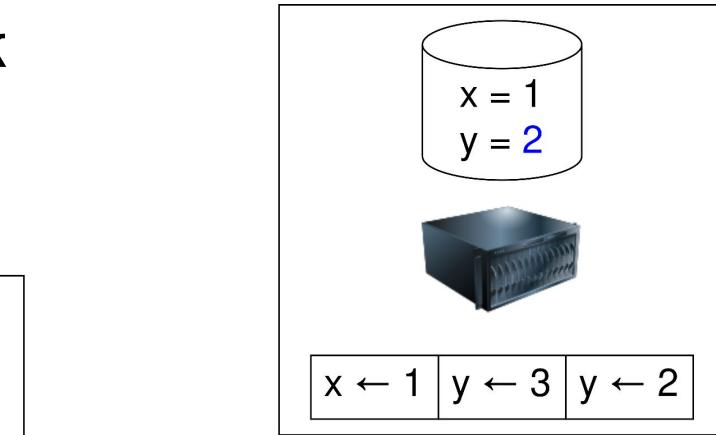
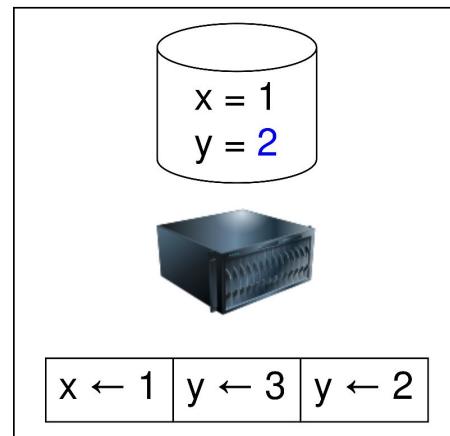
Raft: коммит записей

- Сервера применяют новую запись к локальной структуре данных



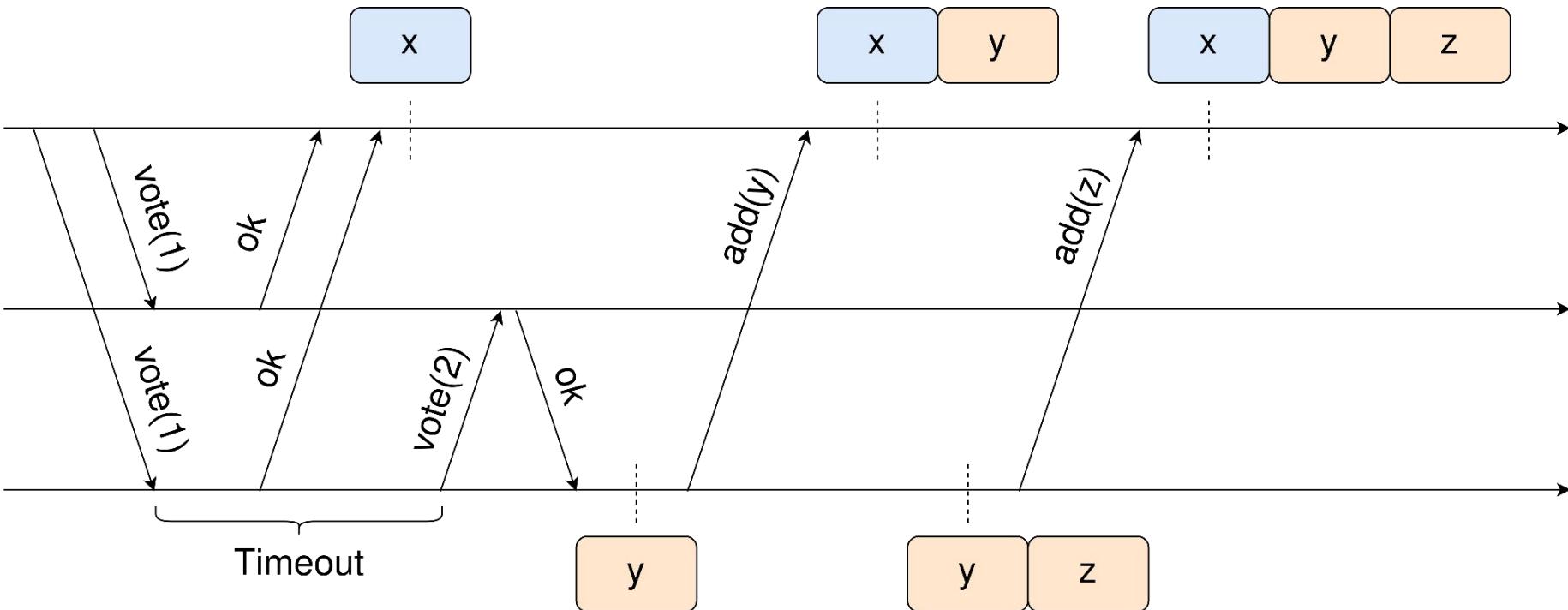
Raft: коммит записей

- Сервера применяют новую запись к локальной структуре данных



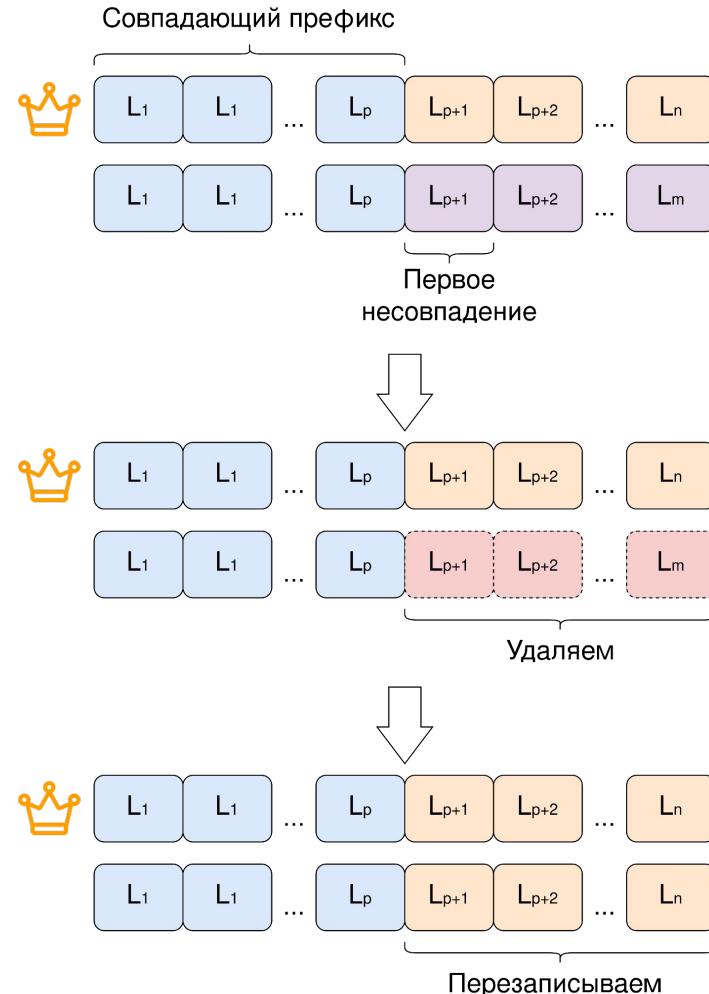
Raft: несовпадение журналов

- Старый лидер не успел реплицировать запись
- Поверх этой записи новый лидер прислал ему ещё пачку



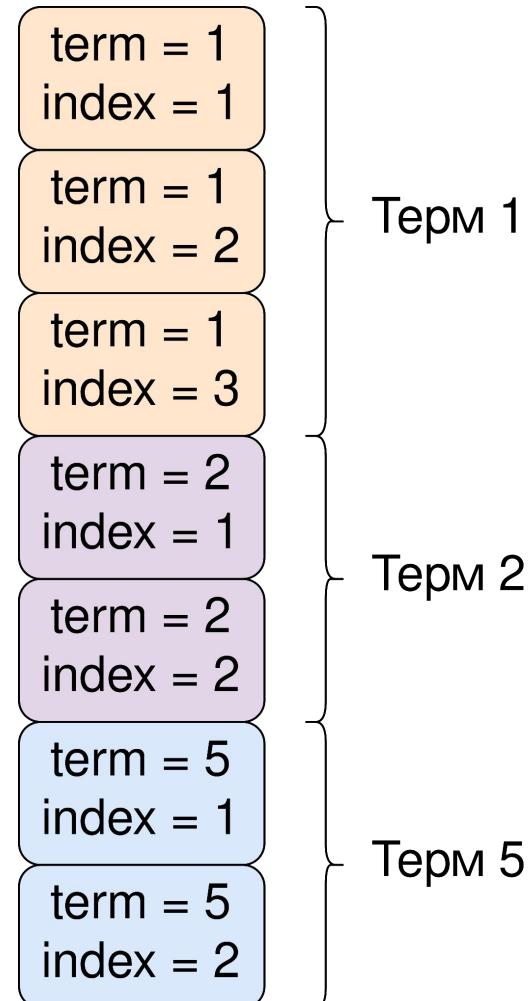
Raft: перезапись журнала

- *Лидер только добавляет записи в свой журнал*
- *Никогда не перезаписывает*
- *Остальные участники могут перезаписывать*
- *Лидер находит первую несовпадающую запись*
- *Перезаписывает суффикс журнала на свой*



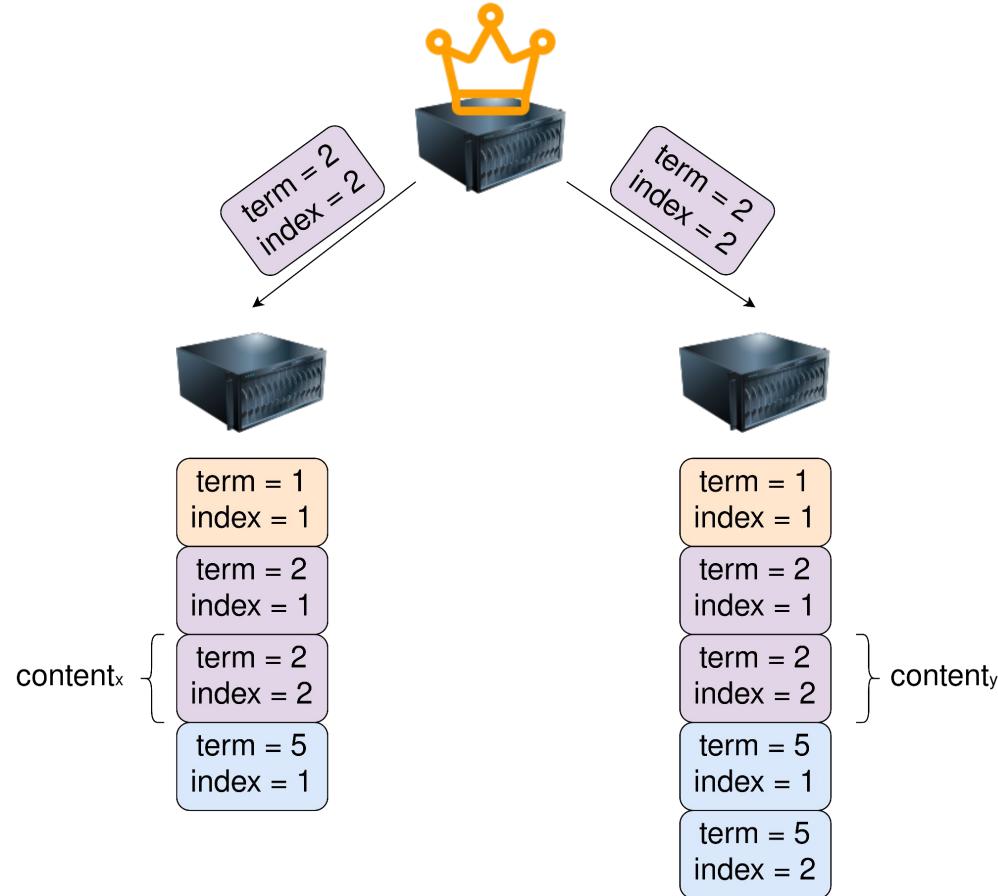
Raft: индексы в журнале

- Каждая запись в журнале идентифицируется парой чисел (`term_num`, `in_term_idx`)
- “Дырок” внутри терма быть не может
 - Лидер реплицирует логи последовательно
- Но каких-то термов может не быть
 - Не выбрали лидера



Raft: индексы в журнале

- Если две записи имеют одинаковый term_num и in_term_idx, то их содержимое одинаковое
- Потому что в терме всего один лидер
- Он и отреплицировал эту запись на оба сервера



Raft: условное добавление



- Лидер посылает серверу сообщение
`ADD(log_index, op, termprev, idxprev)`
- Добавляем запись в журнал только если терм и индекс $\log_index - 1$ 'ой записи совпадут с $(term_{prev}, idx_{prev})$
- Иначе отвечаем ошибкой

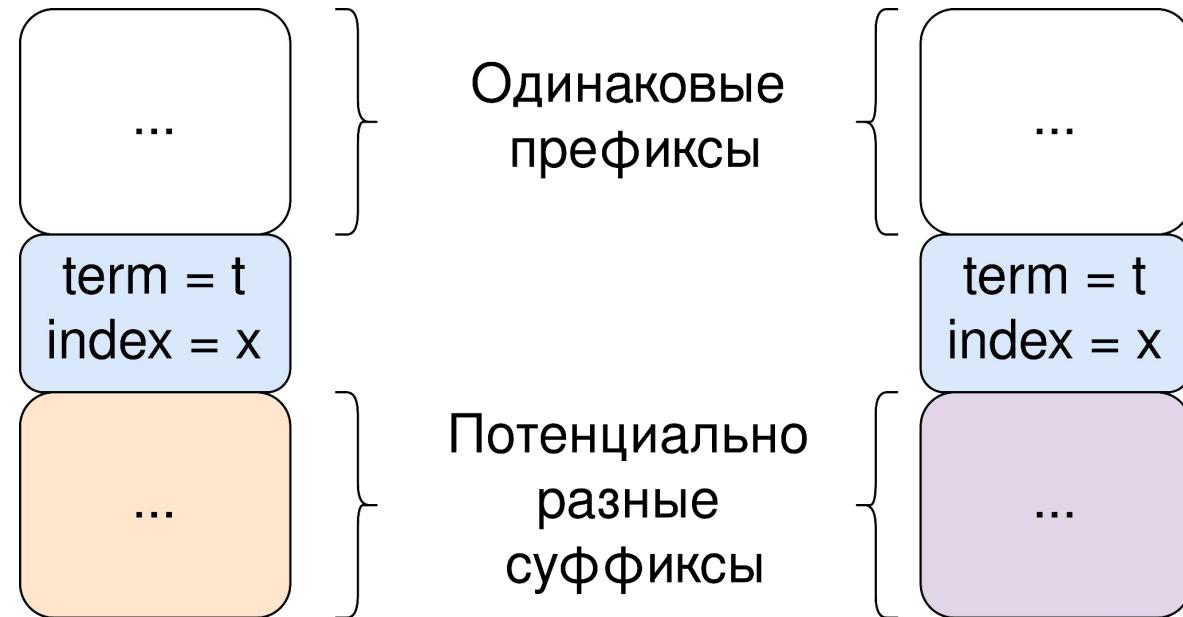
Raft: поиск точки расхождения

- Уменьшая индекс в журнале, находим первое расхождение
- Сервер переписывает расходящуюся запись
- Стирая все дальнейшие записи
- Репликация продолжается без проблем



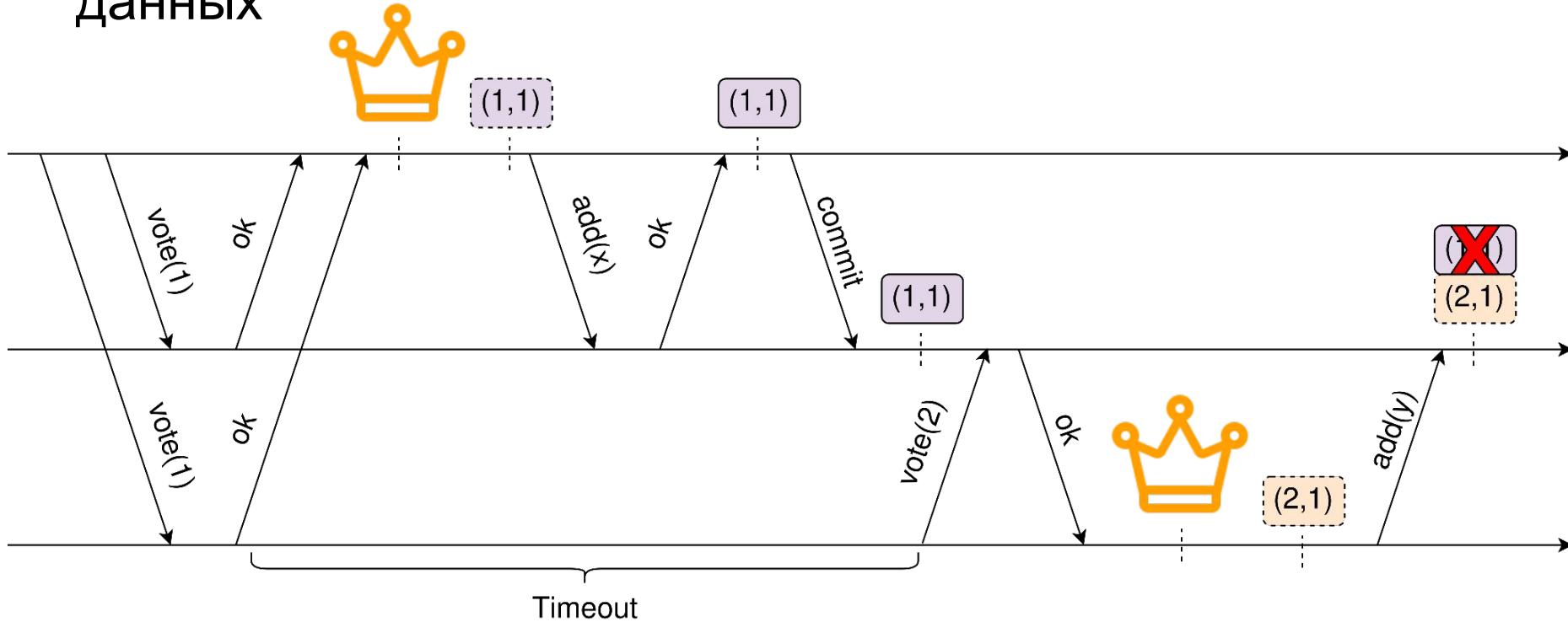
Raft: поиск точки расхождения

- Если в двух журналах есть записи с одинаковыми (term, index), то:
 - Они расположены на одной позиции в журналах
 - Журналы совпадают вплоть до этой позиции
- Можно использовать бинпоиск



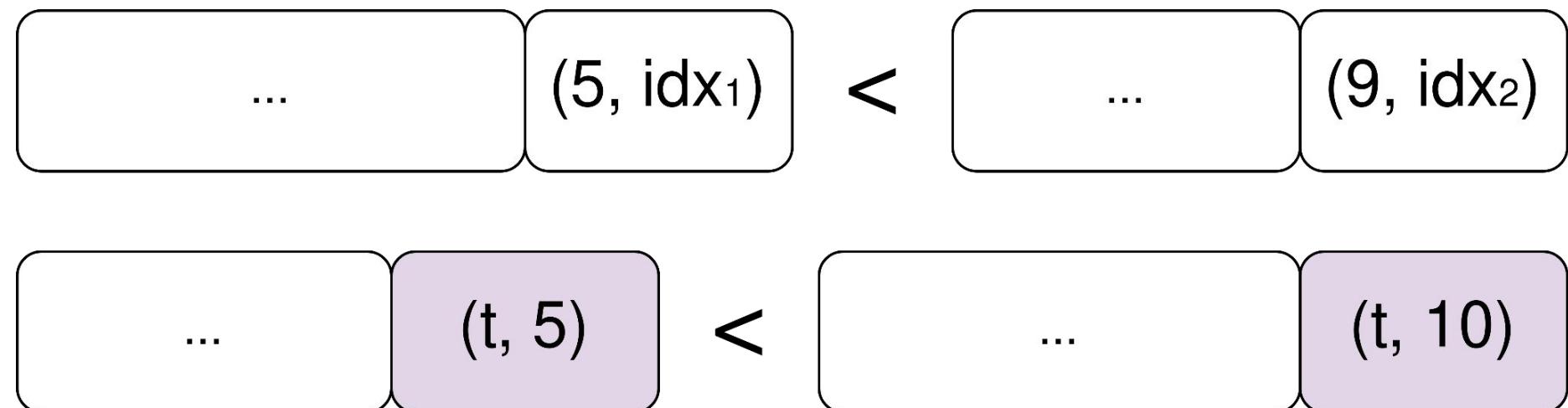
Raft: перезаписывание закоммиченных записей

- Не можем этого допустить
- Закоммиченные записи были применены к структуре данных



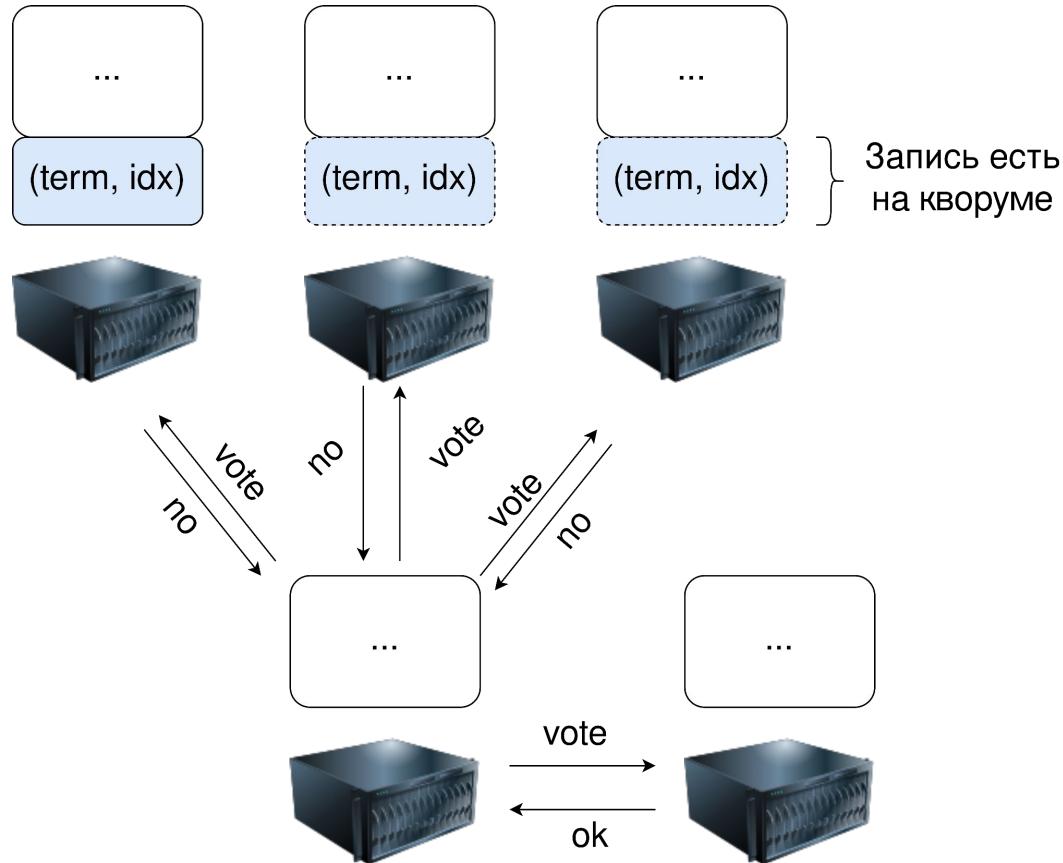
Raft: сравнение журналов

- Сначала сравниваем по терму последней записи
- При равенстве — по индексу внутри терма
- **Не отдаем голос за сервер с логом меньше, чем у нас**



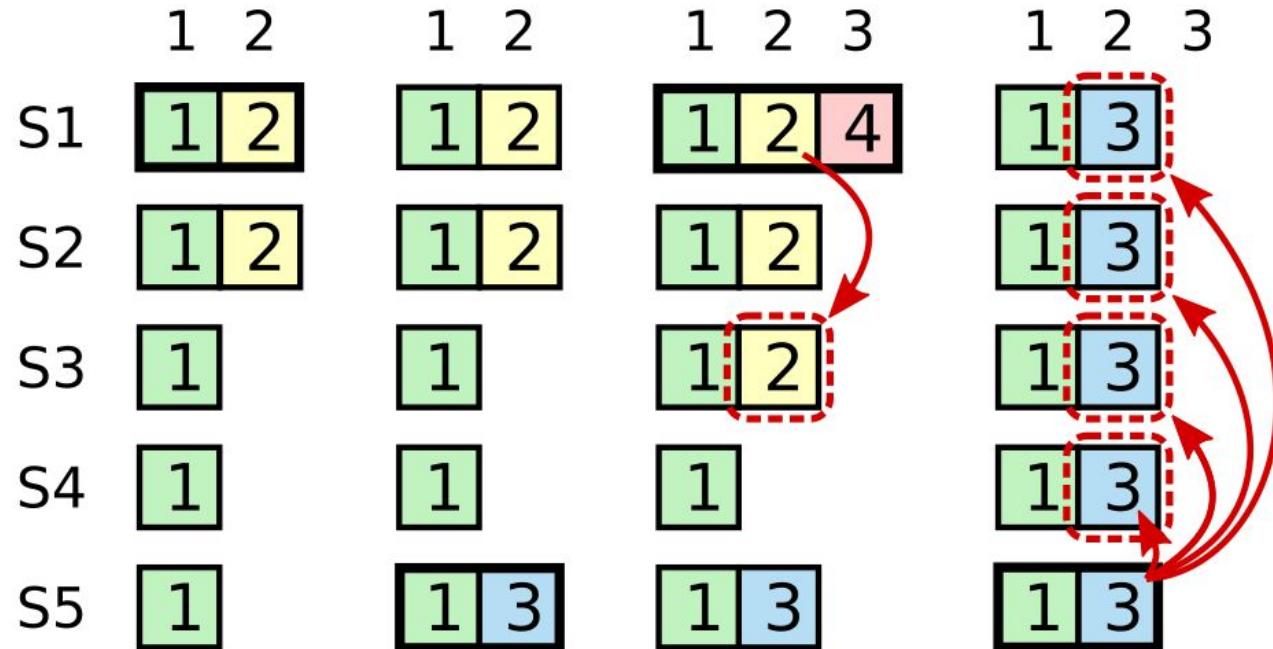
Raft: надёжность репликации

- Запись есть у большинства
- Это большинство не голосует за процесс, у которого записи нет
- Лидером никогда не станет процесс без этой записи
- Закоммиченная запись не будет стёрта



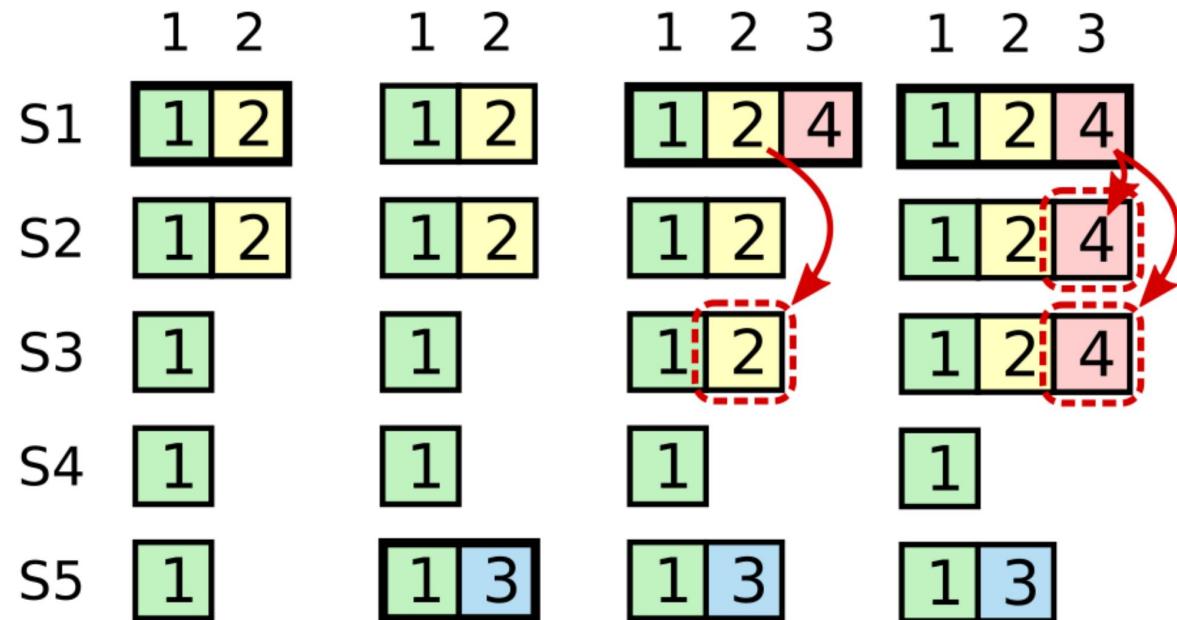
Raft: репликация записей из прошлых термов

- Репликация записи из прошлого терма на большинство серверов **не гарантирует**, что эта запись будет закоммичена



Raft: репликация записей из прошлых термов

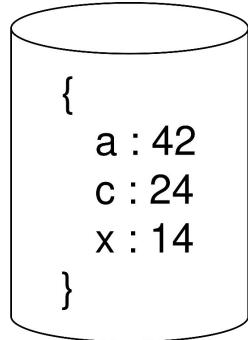
- Запись из прошлого терма считается закоммиченной только после того, как **хотя бы одна запись из нашего терма** была закоммичена



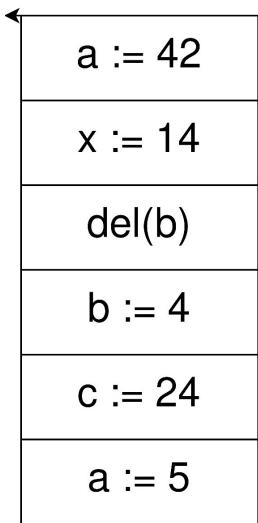
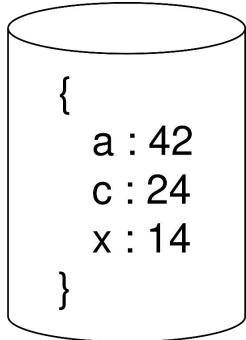
Raft: оптимизации

- Батчинг
 - Передаём много записей за раз
- Piggybacking
 - Передаём служебную информацию вместе с записями
- message AppendEntries {
 [Entries...],
 CurTerm, /* Do not append when obsolete */
 LogIndex,
 PrevEntryTerm, PrevEntryIndex,
 CommitIndex
}

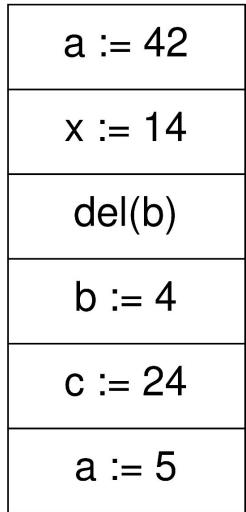
Raft: минутка занимательной филологии



↔ Идентичны ↔



↔ Идентичны ↔



Raft: минутка занимательной филологии



- Репликация это старый морской обычай



Team Paxos vs Team Raft

Heidi Howard @ Hydra



Amazon ECS



cassandra

PaxosStore:
High-availability Storage Made Practical in WeChat

Jianjun Zheng[†] Qian Lin[†]^{*} Jitao Xu[†] Cheng Wei[†]
Chuwei Zeng[†] Pingan Yang[†] Yunfan Zhang[†]

[†]Tencent Inc. ^{*}National University of Singapore
[>{rockzheng, sunnyxu, dengoswei, eddyzeng, ypaipyang, itanzhang}@tencent.com](mailto:{rockzheng, sunnyxu, dengoswei, eddyzeng, ypaipyang, itanzhang}@tencent.com)
linqian@comp.nus.edu.sg

Windows Azure Storage: A Highly Available
Cloud Storage Service with Strong Consistency

Brad Calder, Ju Wang, Aron Ogus, Nirajen Nalakantan, Ardit Skjelkvold, Sam McKeown, Yikang Xu,
Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jader Hanida, Chakravarthy Uddaraju,
Hemal Khetri, Andrew Edwards, Varman Bedekar, Shane Mainali, Rafeey Abbasi, Apti Agarwal,
Mian Fahim ul Haq, Muhammad Iram ul Haq, Deepali Bhardwaj, Soumya Dayanand,
Aritha Akusumit, Marvin Mellett, Srinam Sankaran, Kevaltha Manivannan, Leonidas Rigas

Microsoft



Megastore: Providing Scalable, Highly Available
Storage for Interactive Services

Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson,
Jean-Michel Leon, Tawfiq Lutfi, Alexander Lloyd, Vadim Yusupov
Google, Inc.

jasonbaker, christobond, jcorbett, jjfurman, abkerlin, jlarson, jay.yusupov@googlegroups.com

Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[†] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.



The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, Google Inc.



PaxosStore:
High-availability Storage Made Practical in WeChat

Jianjun Zheng[†] Qian Lin[†]^{*} Jitao Xu[†] Cheng Wei[†]
Chuwei Zeng[†] Pingan Yang[†] Yunfan Zhang[†]

[†]Tencent Inc. ^{*}National University of Singapore
[>{rockzheng, sunnyxu, dengoswei, eddyzeng, ypaipyang, itanzhang}@tencent.com](mailto:{rockzheng, sunnyxu, dengoswei, eddyzeng, ypaipyang, itanzhang}@tencent.com)
linqian@comp.nus.edu.sg

Clustrix



Doozer



github.com/tigrisdb



HashiCorp
Consul



hazelcast

RethinkDB

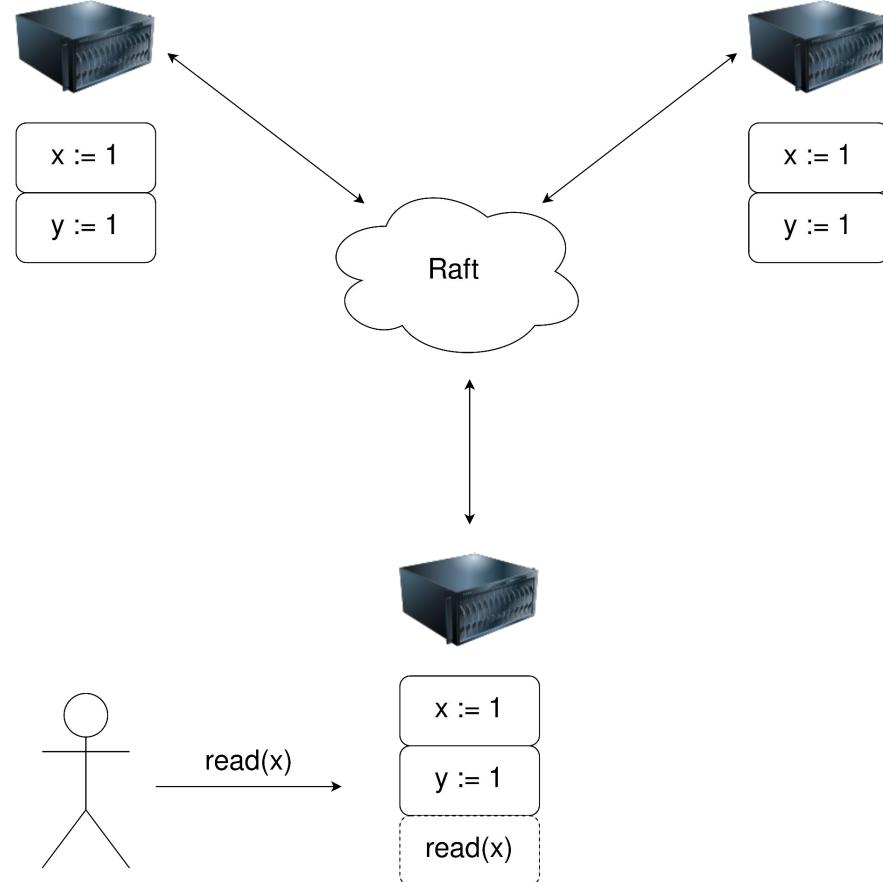


etcd

Cockroach DB

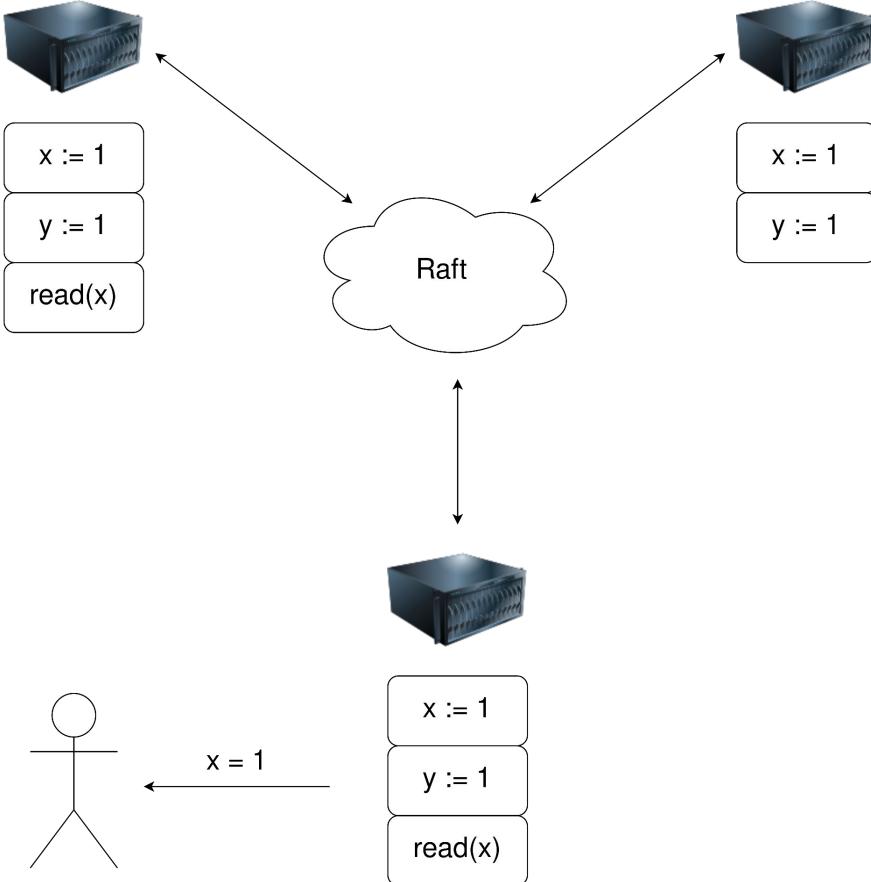
Raft: чтение

- Посыпаем команду чтения серверу
- Если надо, сервер пересыпает лидеру
- Лидер начинает репликацию



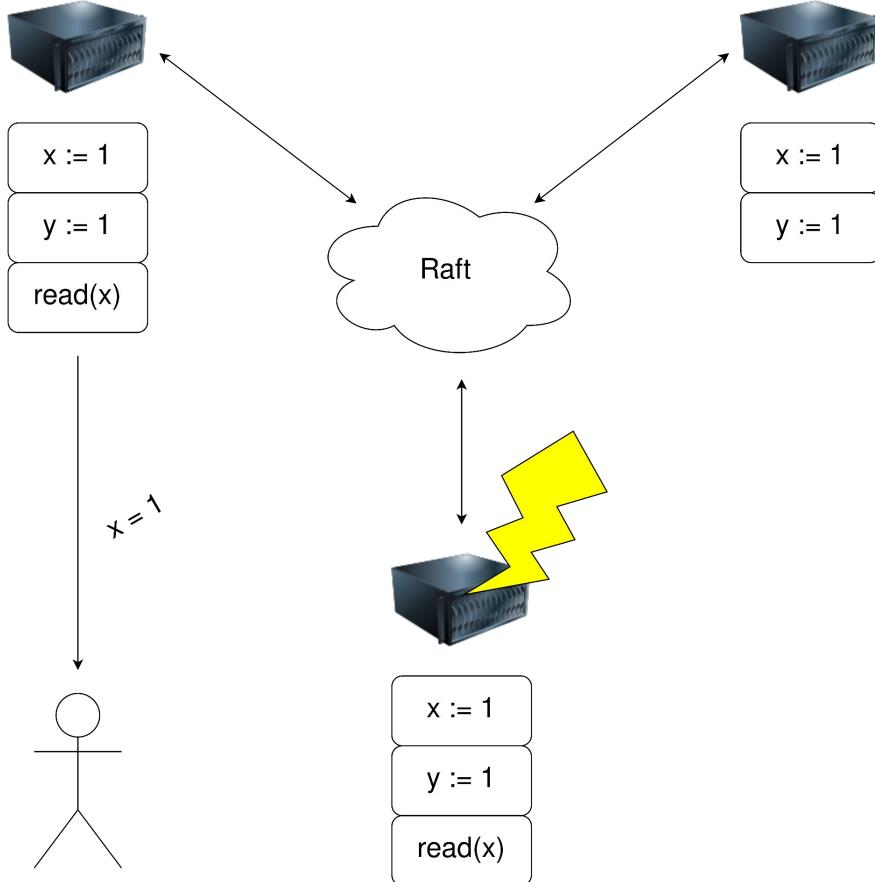
Raft: чтение

- Лидер добавляет запись в журнал
- Исходный сервер дожидается коммита
- Отвечает клиенту



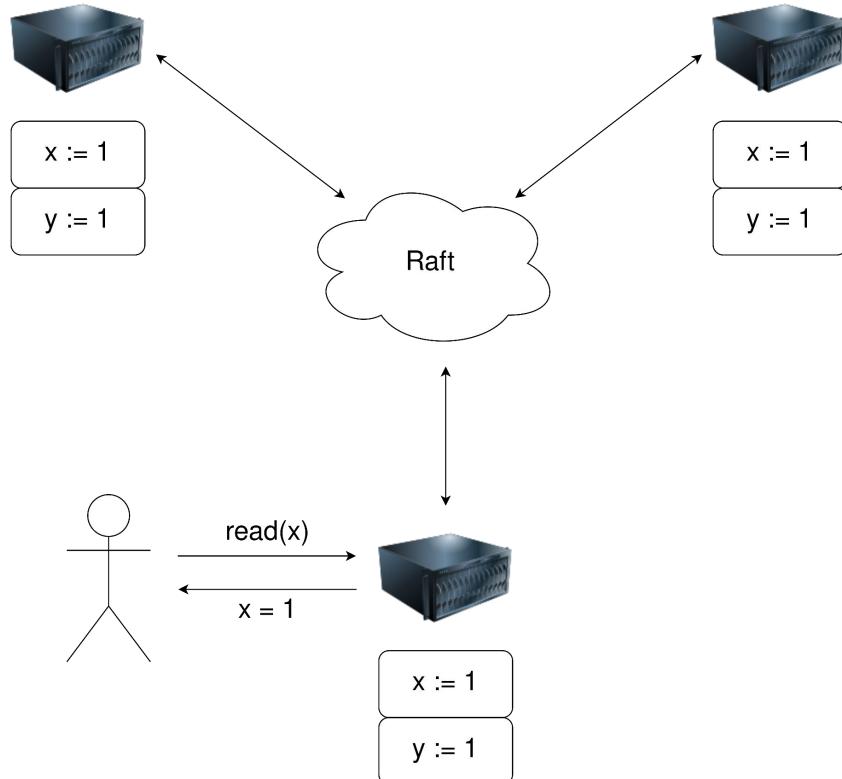
Raft: чтение

- В случае сбоя ответить клиенту может любой из серверов
- Хоть все сразу
- Ответ будет одним и тем же
- Так может работать чтение в любом алгоритме RSM



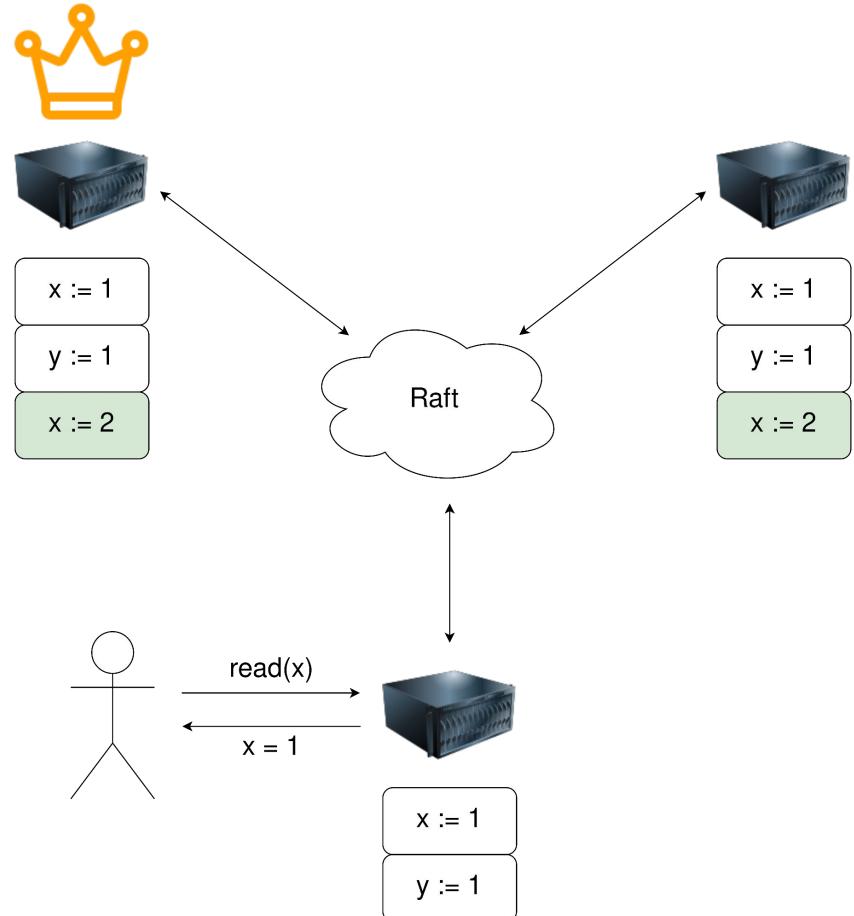
Raft: чтение без кворума

- Направляем команду на сервер
- Сервер отвечает, смотря только на локальную структуру данных
- Не используя механизм консенсуса



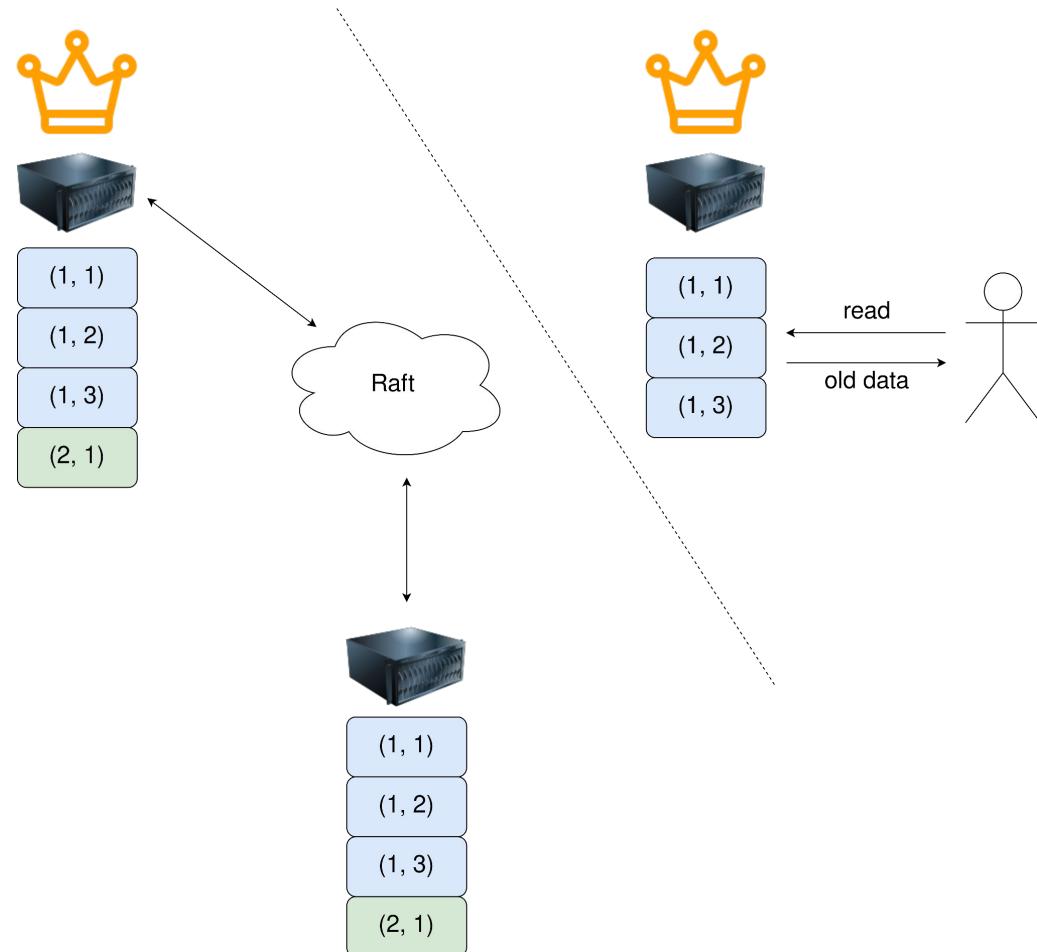
Raft: чтение без кворума

- Можем получить устаревшие данные
- Но только те данные, которые когда-то были закоммичены



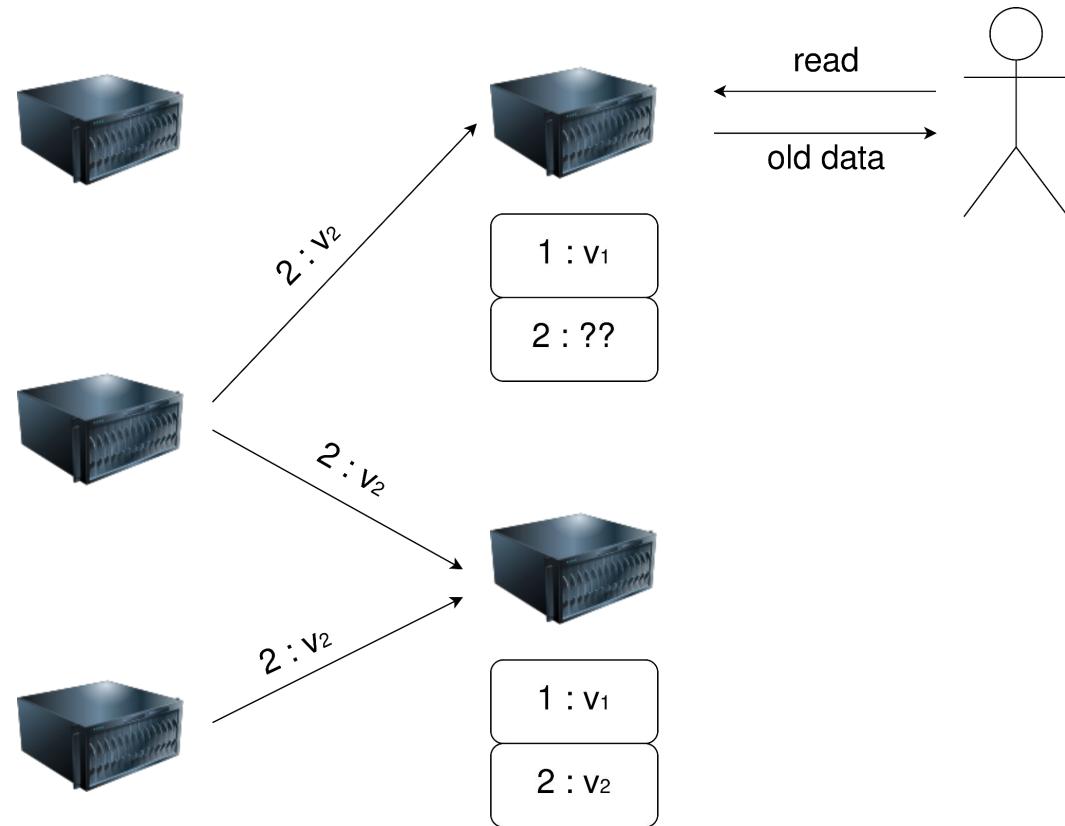
Raft: чтение с лидера

- Тоже может вернуть устаревшие данные
- Лидер может быть не связан с кворумом
- Кворум уже избрал нового лидера
- Старый лидер всё ещё считает себя лидером



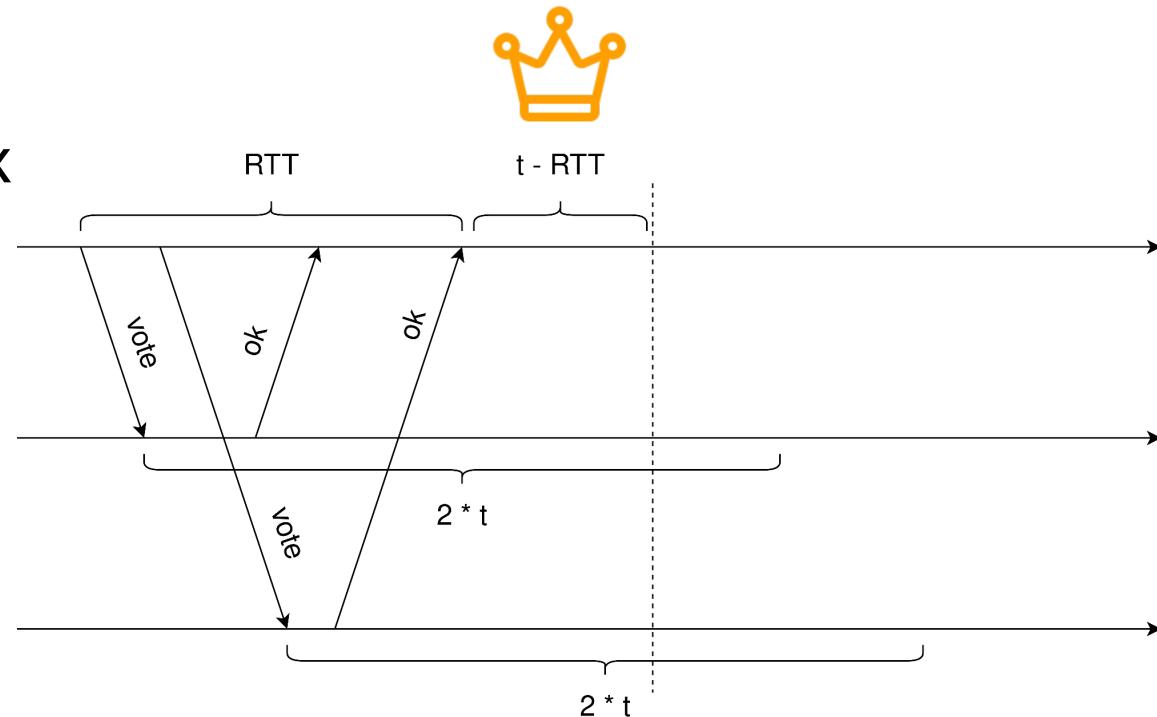
RSM Paxos: чтение без кворума

- У других RSM алгоритмов есть такая же проблема
- Например, в Paxos можем захотеть читать без кворума с узнающих
- Узнающий мог не получить результаты последнего голосования



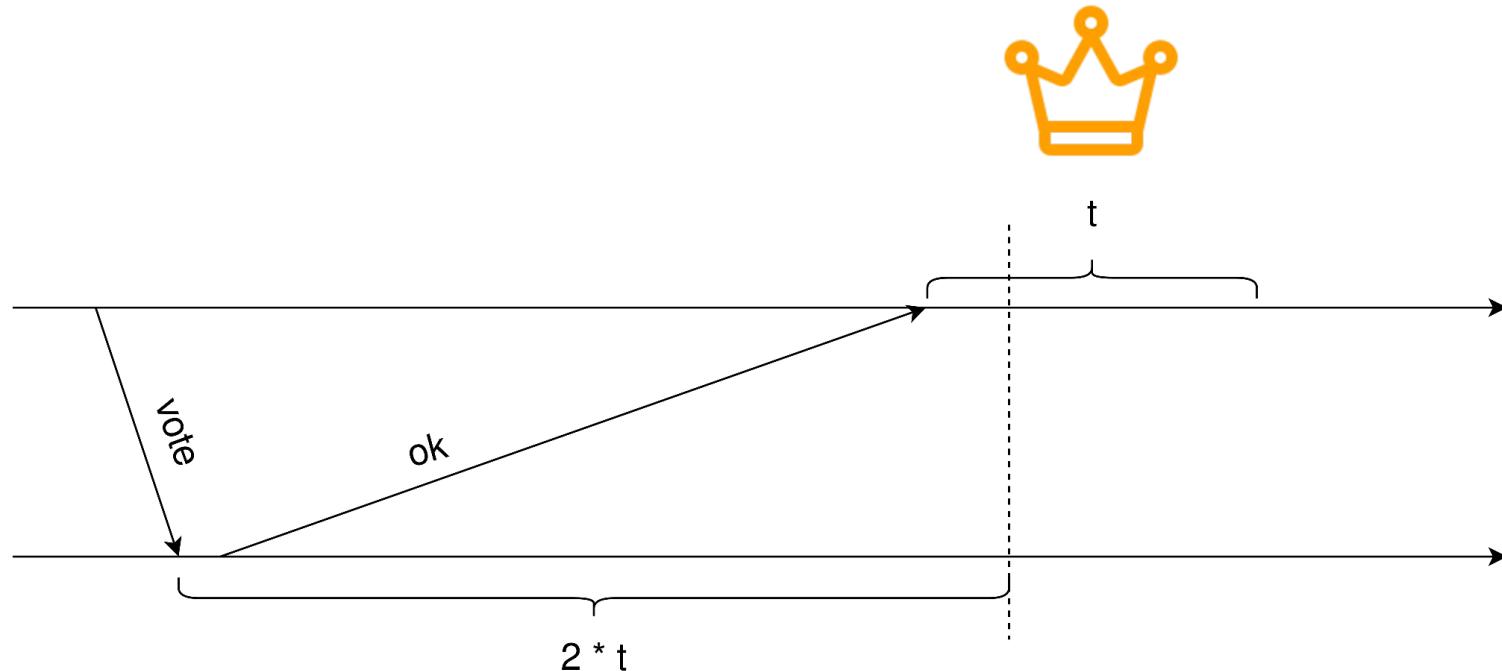
Raft: leases

- Процесс считает себя единственным лидером в течение $t - \text{RTT}$ секунд с момент получения **кворума голосов**
- Выбравшие его не могут участвовать в других голосованиях в течении $2 * t$ секунд с момента отдачи голоса
- Требуется синхронизация физических часов



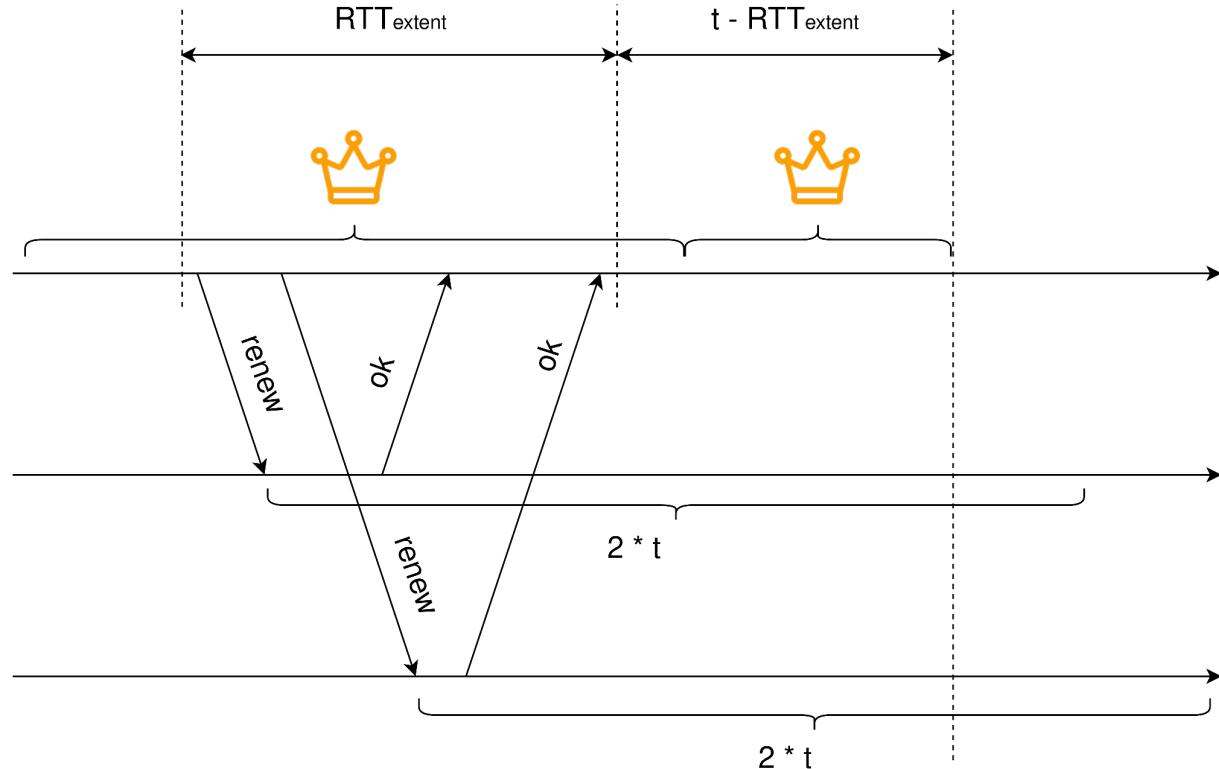
Raft: leases

- Зачем вычитать RTT?
- Иначе всё сломается, если голос шёл достаточно долго



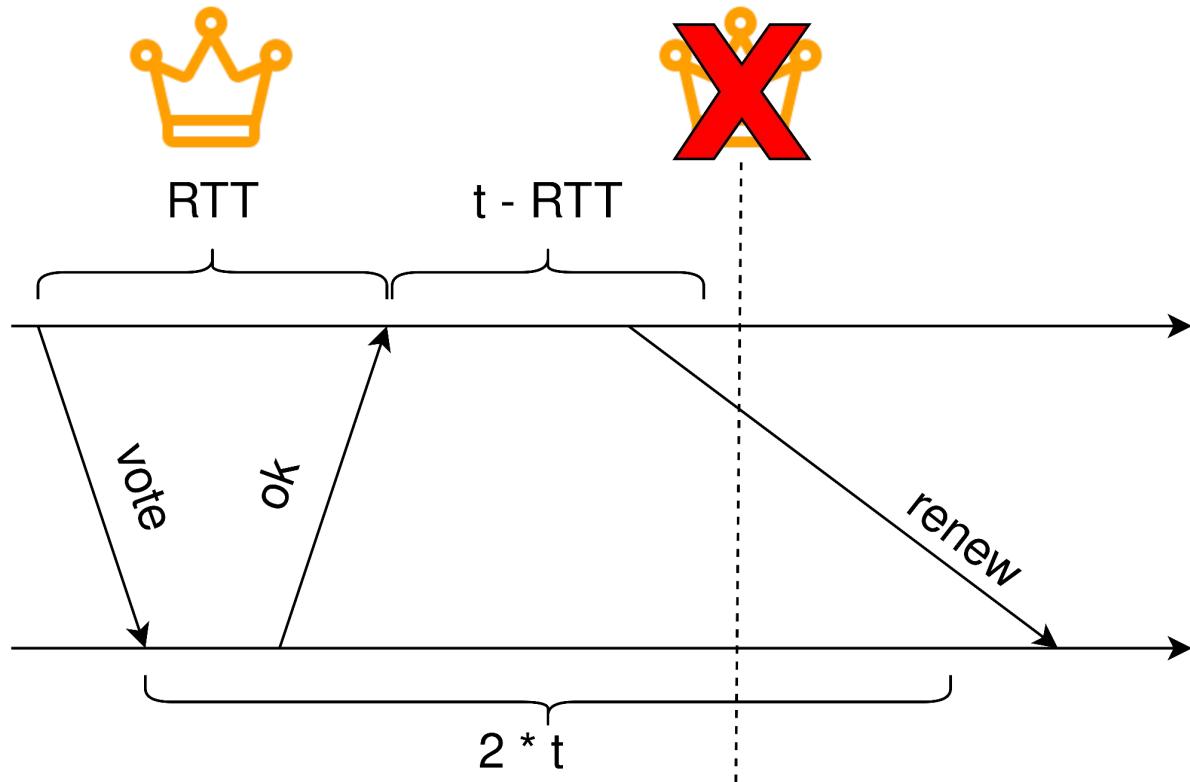
Leases: продление

- Работает по той же схеме, что и получение



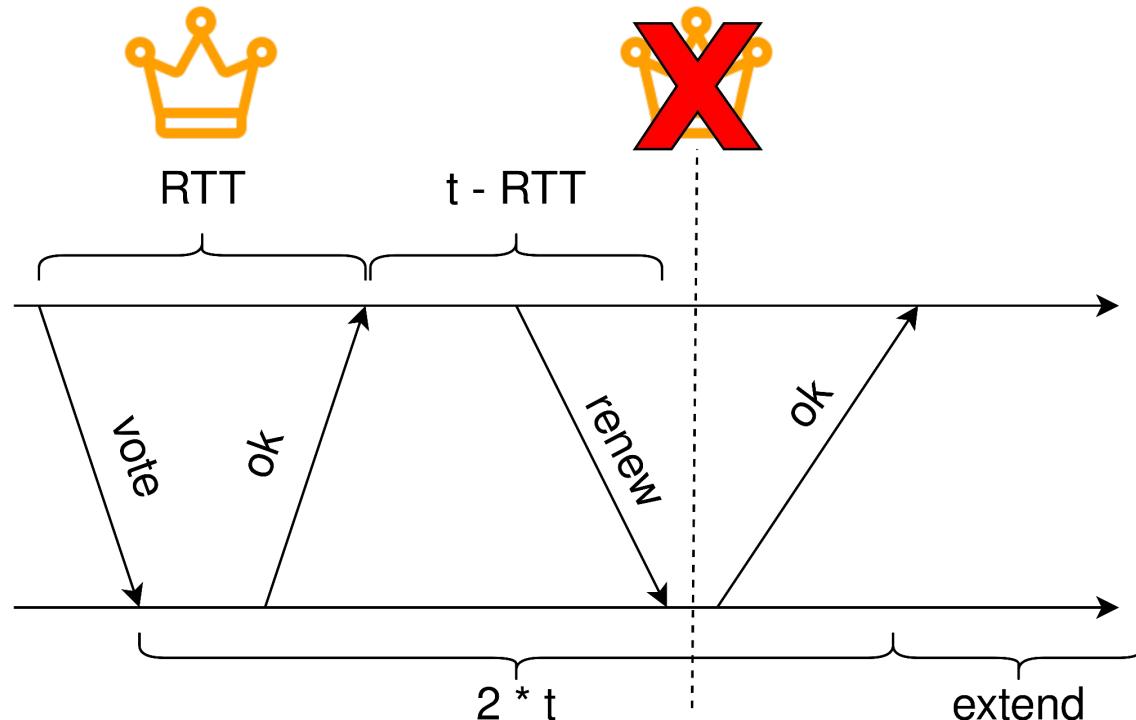
Leases: отречение от престола

- Если продлить не получилось, процесс перестаёт быть лидером
- Это произойдёт раньше, чем остальные процессы выберут нового лидера



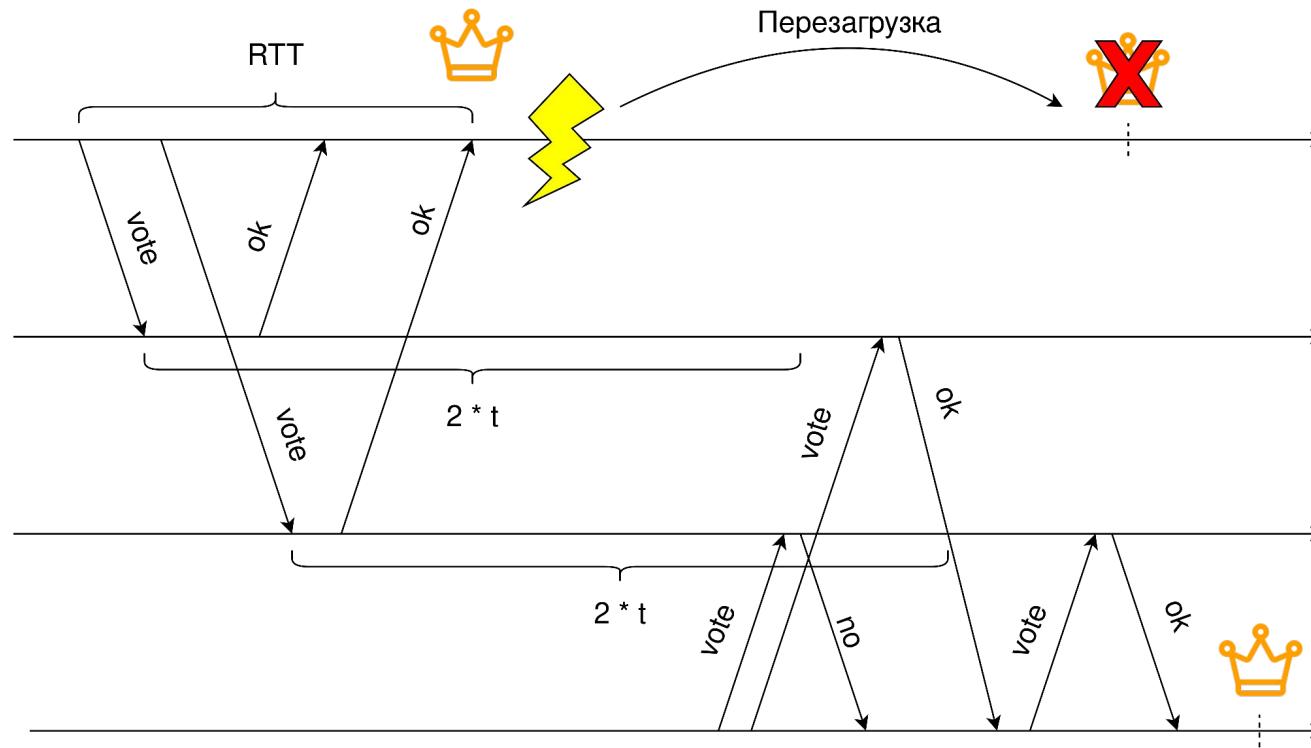
Leases: отречение от престола

- Если согласие продлить пришло уже позже нашего отречения — игнорируем его
- Возможно, оно долго шло
- Период продления у отправившего уже истёк
- И он успел проголосовать



Leases: перевыборы лидера

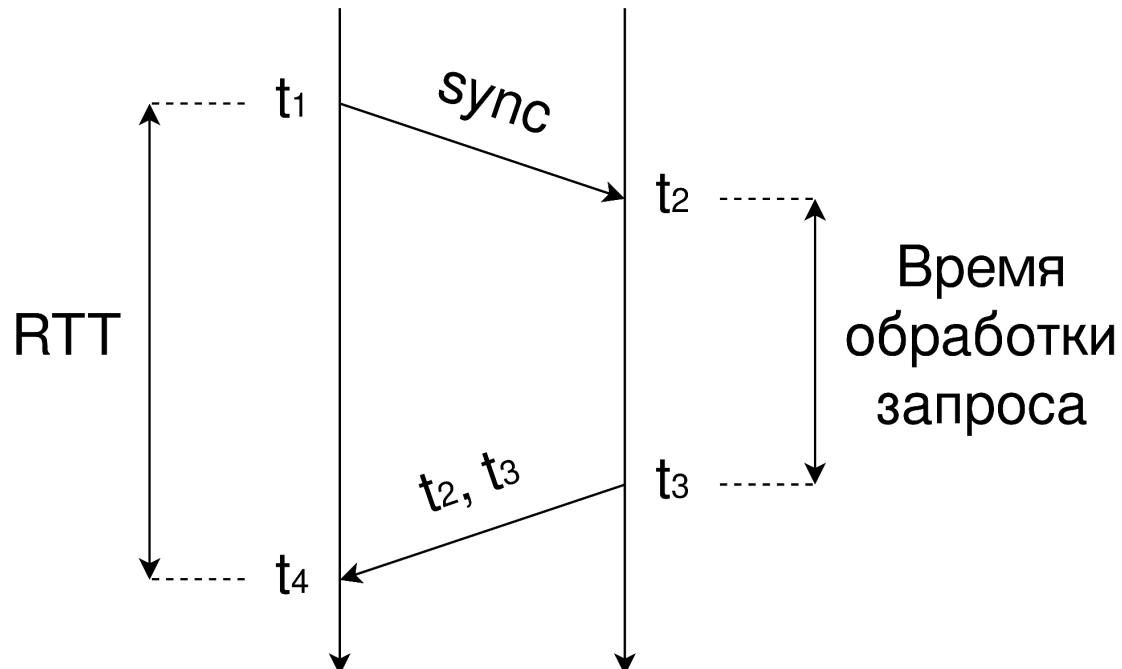
- При сбое можем выбрать нового лидера только после истечения периода в $2 * t$ секунд



NTP: синхронизация часов

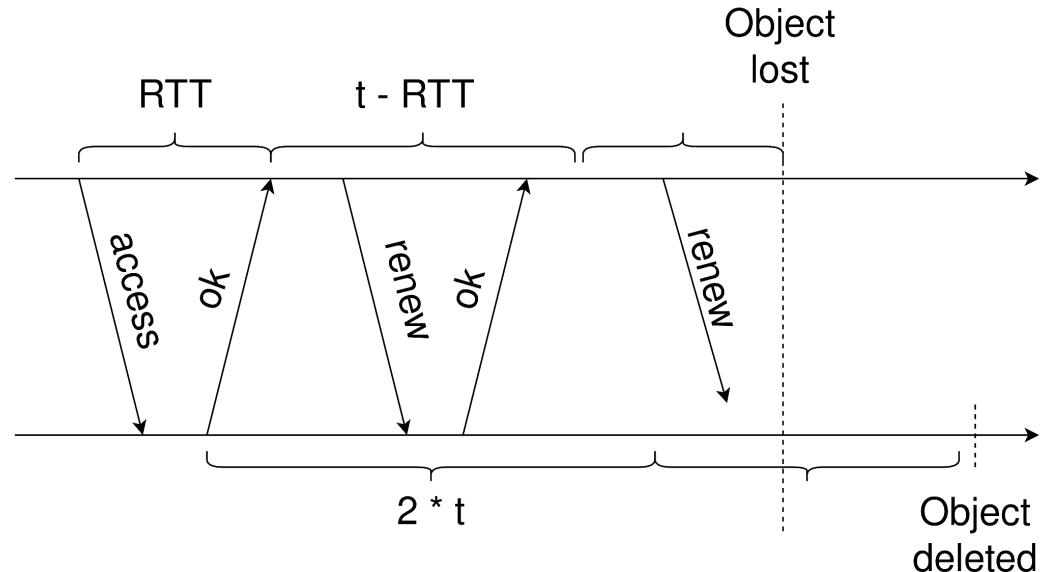
- В пути запрос провёл $\sigma = (t_4 - t_1) - (t_3 - t_2)$
- В одну сторону $\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$

$$T \leftarrow t_3 + \delta$$



Lease: распределённая сборка мусора

- Техника в распределённых системах используется повсеместно
- Распределённый GC реализован в Java RMI



Что почитать: Raft

- *Ongaro D., Ousterhout J.* In search of an understandable consensus algorithm (raft.github.io)
- *Howard H., Mortier R.* Paxos vs Raft: Have we reached consensus on distributed consensus
- *Huang D. et al.* TiDB: a Raft-based HTAP database
- *Sergey Petrenko, Boris Stepanenko. Solving Raft's practical problems in Tarantool*
- *A Byzantine failure in the real world*
- *Eli Bendersky. Implementing Raft*
- *Сергей Останевич. Повышаем живучесть Raft в реальных условиях*

Что почитать & посмотреть: лизы и часы

- *Chandra T. D., Griesemer R., Redstone J.* Paxos made live: an engineering perspective
- *Burrows M.* The Chubby lock service for loosely-coupled distributed systems
- *Li Y. et al.* Sundial: Fault-tolerant clock synchronization for datacenters
- [Martin Kleppmann. Clock synchronization](#)
- [Kevin Sookocheff. How Does NTP Work?](#)

Thanks for your attention

