

# Консенсус и связанные задачи

Олег Сухорослов

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

21.11.2022

The best way of building fault-tolerant systems is to find some general-purpose abstractions with useful guarantees, implement them once, and then let applications rely on those guarantees.

Martin Kleppmann. Designing Data-Intensive Applications.

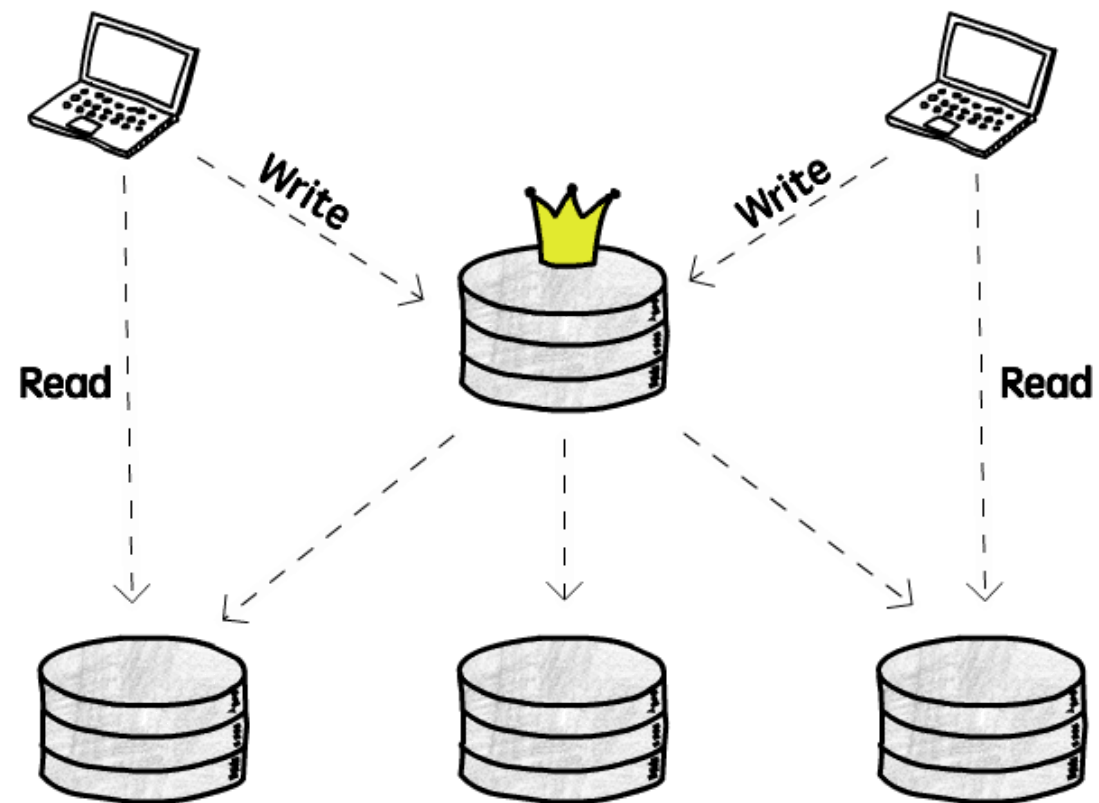
# Встречающиеся на практике требования

- Только один клиент должен вести запись в файл
- Только один узел в системе должен играть эту роль
- У каждого пользователя должно быть уникальное имя
- Баланс счёта не должен быть отрицательным
- Не должно быть продано больше товаров, чем есть на складе
- Реплицируемое хранилище должно обеспечивать линейризуемость
- Транзакция не должна быть зафиксирована в системе частично

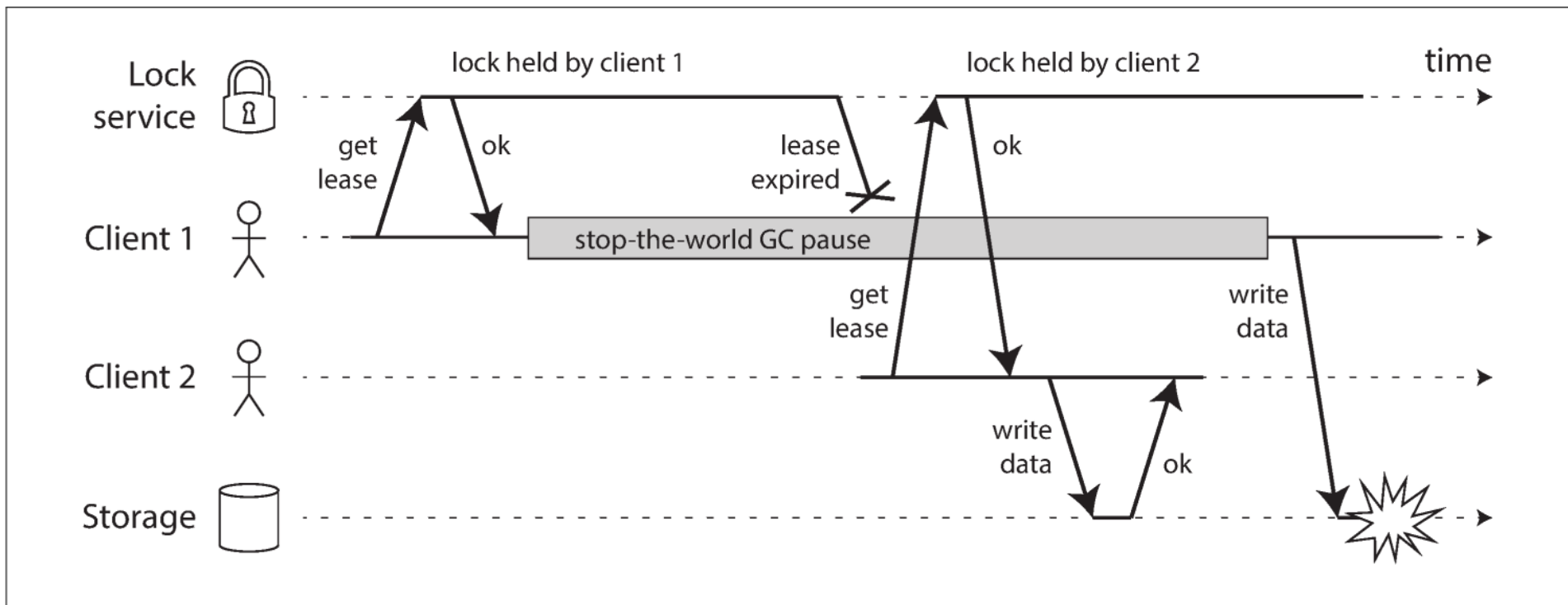
# Возможные решения

- Один сервер
  - Упорядочивание операций
  - Блокировки (locks)
  - Атомарные RMW-операции типа compare-and-swap (CAS)
- Распределенная система
  - Лидер
  - Сервис блокировок
  - Координатор транзакций
  - Атомарная рассылка
  - Консенсус

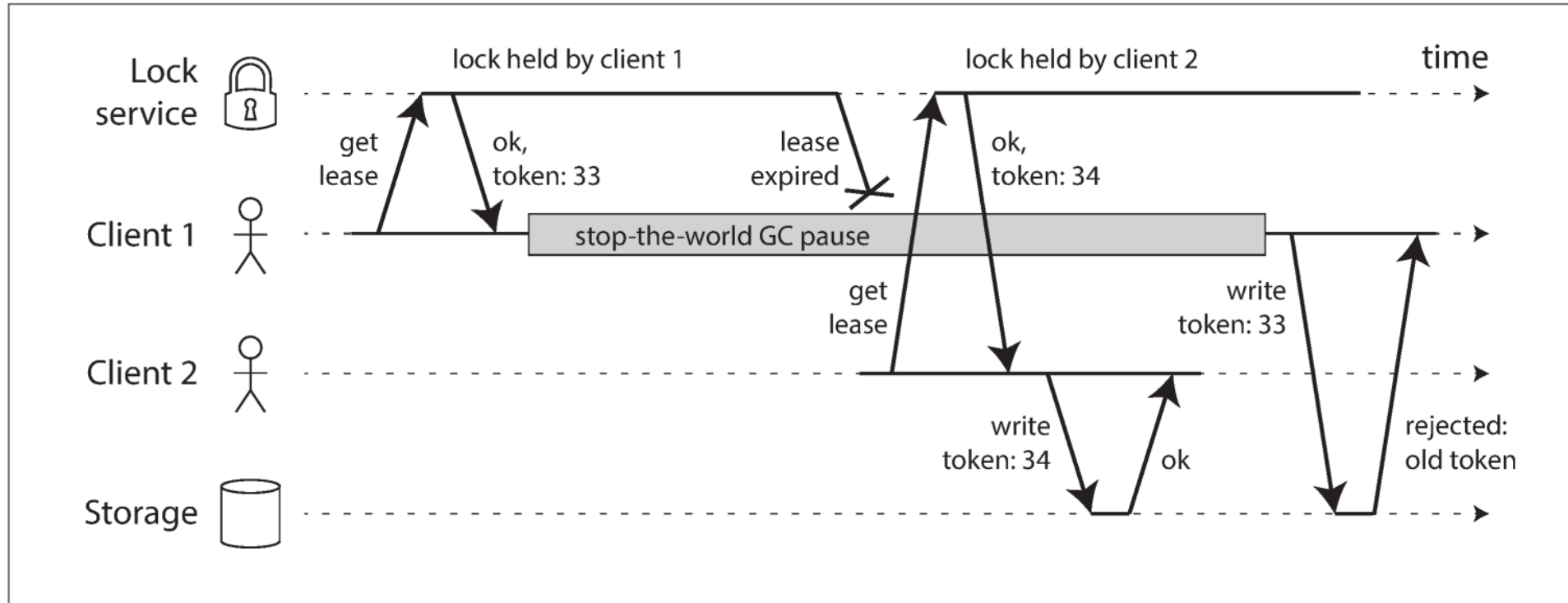
# Лидер



# Сервис блокировок



# Fencing Token

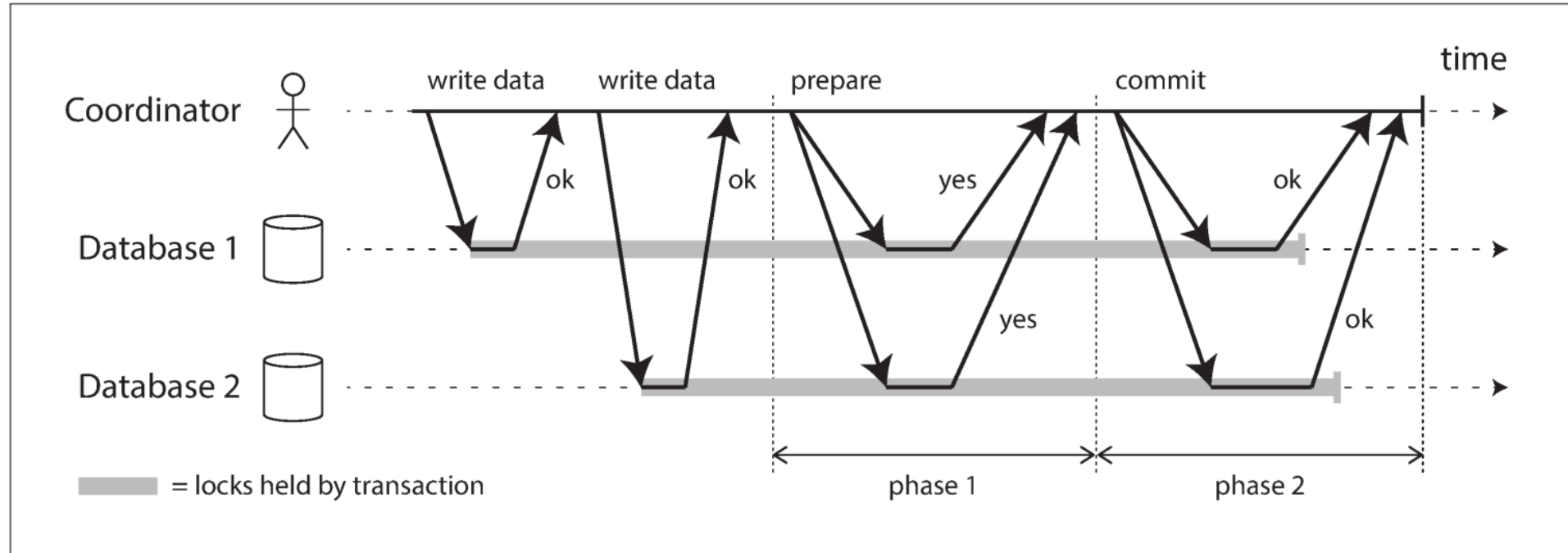


# Атомарная фиксация транзакции

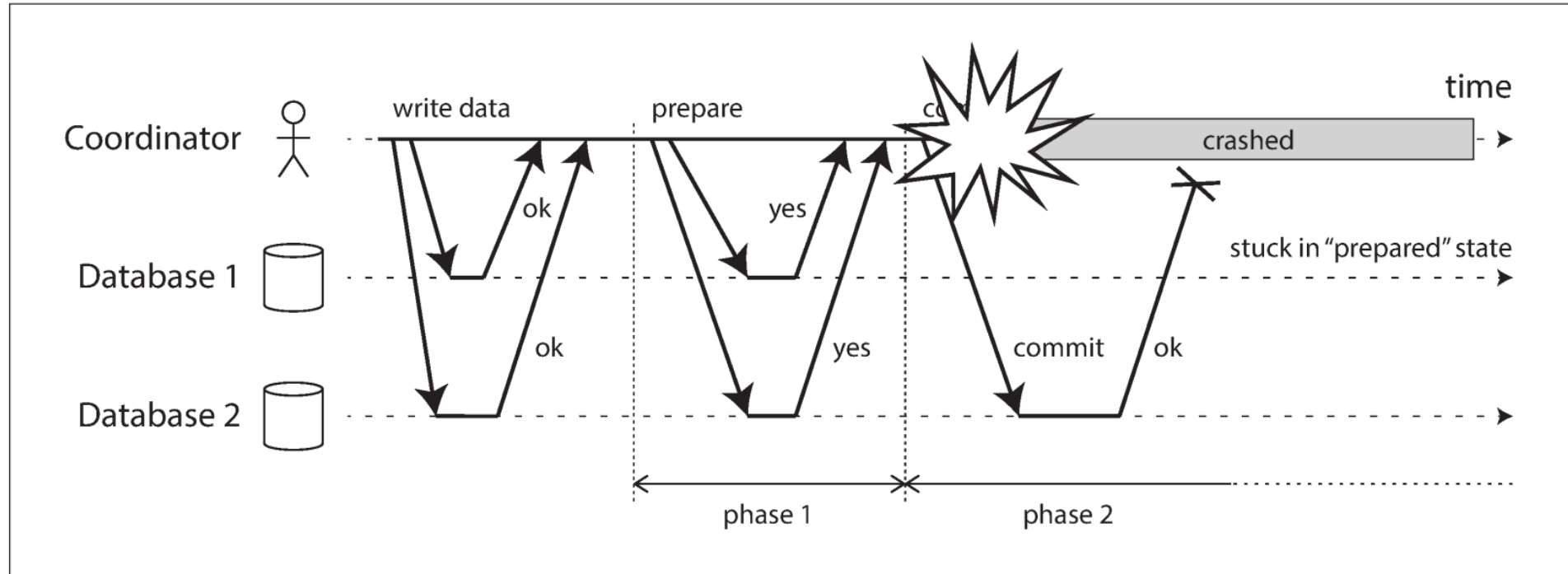
- Транзакция
  - Запрос, состоящий из нескольких операций над несколькими объектами в базе данных
  - Транзакция может быть выполнена либо целиком, либо не выполнена вообще
- Один сервер
  - Запись изменений в журнал на диске (write-ahead log)
  - Добавление записи о фиксации изменений (commit record)
  - Одно устройство (контроллер диска, на котором хранится журнал)
- Несколько серверов
  - Достаточно ли просто отправить commit на каждый сервер?
  - Изменения должны быть применены только если все серверы "согласны"
  - Примененные изменения нельзя отменить, они уже видны клиентам



# Two-Phase Commit (2PC)



# Отказ координатора

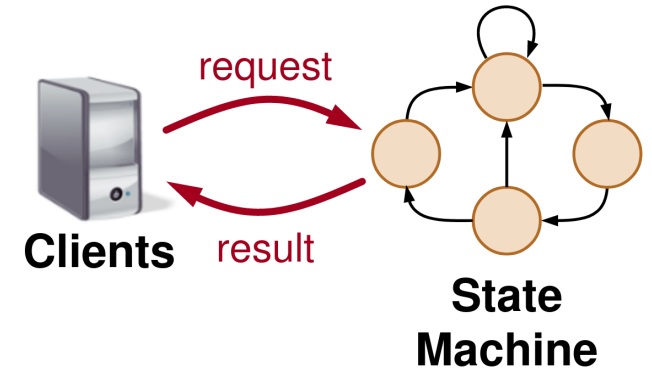


# Атомарная рассылка

- Надежная и полностью упорядоченная рассылка
  - Reliable + Total Ordered = **Atomic Broadcast** (или Total Order Broadcast)
  - если процесс доставил сообщение, то оно доставлено всеми корректными процессами
  - сообщение доставляется ровно один раз
  - порядок доставки сообщений одинаков для всех процессов
  - иногда также предполагается сохранение порядка отправки сообщений (FIFO-total)
- Гарантий на скорость доставки нет
  - некоторые получатели могут отставать от других

# Применение атомарной рассылки

- Репликация произвольных объектов (состояния, данных) с поддержкой линейризуемости
  - рассылка операций по репликам
  - каждая реплика применяет операцию локально
  - т.н. репликация автомата (state machine replication)
- Реплицированный журнал (append-only log)
  - полученное сообщение добавляется в конец журнала
  - на всех узлах префиксы журналов совпадают
- Упорядочивание блокировок
  - каждый запрос на блокировку добавляется в журнал
  - номер записи в журнале используется для fencing



# Применение атомарной рассылки

- Фиксация транзакции?
- Уникальность имени пользователя?

# Применение атомарной рассылки

- Фиксация транзакции
  - каждый участник рассылает свой голос (commit или abort)
  - узел В может проголосовать за А, если считает А отказавшим
  - каждый участник локально определяет окончательный исход, читая журнал
  - для каждого узла учитывается только голос из первого полученного сообщения
- Уникальность имени пользователя
  - добавить сообщение в журнал "планирую взять это имя"
  - читать журнал до тех пор, пока не будет получено это сообщение
  - проверить нет ли аналогичного сообщения от другого участника
  - (по сути получили linearizable register with atomic compare-and-set)

# Реализация атомарной рассылки

- Лидер, рассылающий сообщения с помощью надежной FIFO-рассылки
  - отказ лидера блокирует доставку сообщений
- Рассылка без лидера на основе часов Лэмпорта
  - см. прошлую лекцию
  - отказ любого узла блокирует доставку сообщений
- Как сделать рассылку отказоустойчивой?

# Эквивалентные проблемы и примитивы

- Атомарная рассылка
- Линеаризуемый регистр с compare-and-set
- Консенсус
- Решив одну из этих проблем, мы получим готовый примитив, с помощью которого можно:
  - решать рассмотренные ранее практические задачи
  - реализовать остальные (эквивалентные) примитивы



# Консенсус

- Согласие, единодушное принятие чего-либо группой участников
  - обозначает как процесс принятия решения, так и само решение
- Прийти к согласию относительно чего-то между узлами РС
  - кто будет лидером
  - порядок применения операций над репликами
  - применить или откатить транзакцию
  - какой пользователь получит данный username



# Консенсус (более формально)

- Один или несколько процессов предлагают значения
- Алгоритм консенсуса определяет, какое из значений принять
- Требуемые свойства
  - Никакие два корректных процесса не должны принять разные значения
  - Никакой процесс не принимает значение дважды
  - Если процесс принял значение, то оно было предложено некоторым процессом
  - Каждый корректный процесс в конце концов принимает значение

# FLP Impossibility

Fischer M.J., Lynch N.A., Paterson M.S. Impossibility of Distributed Consensus with One Faulty Process (1985)

- Не существует **детерминированного** алгоритма, гарантирующего достижение консенсуса в **асинхронной** системе, если хотя бы **один** процесс подвержен отказу типа crash-stop
  - Нарушается последнее свойство консенсуса (termination)
- Не мешает достигать консенсуса на практике
  - Условия, приводящие к постоянному отсутствию прогресса, маловероятны
  - Реальные системы являются частично синхронными
  - Алгоритмы консенсуса используют таймауты для обеспечения termination
  - Другие способы обхода FLP: рандомизация, надежные детекторы отказов

# Модель системы (предположения)

- Система является **частично синхронной**
  - система может вести себя как асинхронная только в течение конечных (но неизвестных) периодов времени
  - в остальное время система является синхронной
- Сеть моделируется с помощью **fair-loss links**
  - сообщения могут теряться, дублироваться и переупорядочиваться
  - сообщения в конце концов доходят, если не сдаваться
  - сообщения не искажаются, нет византийских отказов
- Узлы подвержены отказам типа **crash-recovery**
  - узел может внезапно упасть и потерять состояние памяти (можно использовать диск)
  - позже узел может включиться и продолжить выполнение
  - нет византийских отказов

# Отказоустойчивость

- Выполнение последнего свойства (termination) требует по меньшей мере **большинства** корректных (не отказавших) узлов
  - в случае *византийских отказов* требуется более  $2/3$  корректных узлов
- Первые три свойства (safety) могут быть гарантированы и при большем числе отказавших узлов

# Алгоритмы консенсуса

- **Viewstamped Replication** (Oki, Liskov, 1988)
- **Paxos** (Lamport, 1989)
- **Zab** (Junqueira, Reed, Serafini, 2011)
- **Raft** (Ongaro, Ousterhout, 2013)

Многие алгоритмы изначально спроектированы под консенсус не для одного значения, а для последовательности значений

— по сути реализуют репликацию журнала или атомарную рассылку

# Paxos

- 1989: Leslie Lamport разрабатывает новый алгоритм консенсуса
- 1998: Lamport L. The Part-Time Parliament // ACM TOCS
- 2001: Lamport L. Paxos Made Simple // ACM SIGACT News
- 2007: Chandra T. et al. Paxos Made Live: An Engineering Perspective // PODC

The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-)

NSDI reviewer

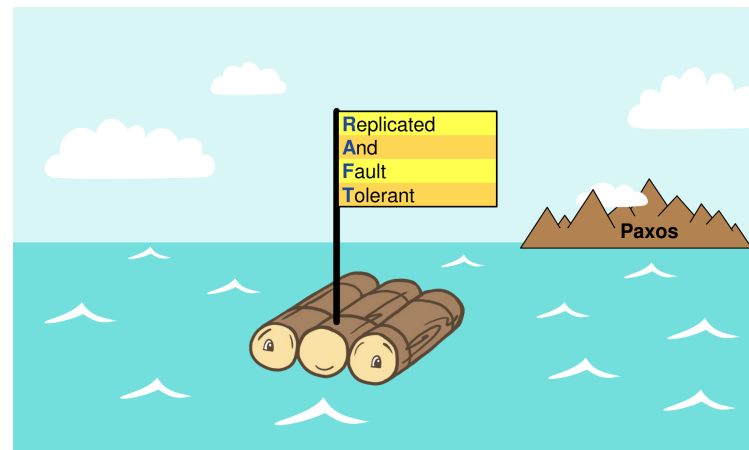
There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system... the final system will be based on an unproven protocol.

Chubby authors

# Raft

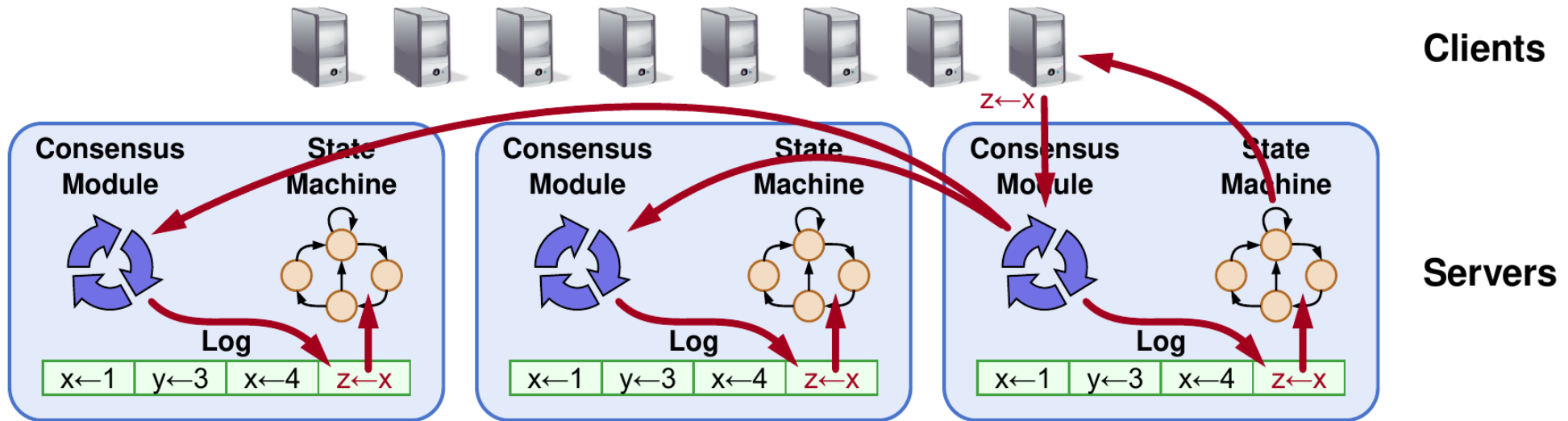
■ Ongaro D., Ousterhout J. In Search of an Understandable Consensus Algorithm (2014)

- Основная цель: простой для понимания и реализации алгоритм консенсуса
- Статья была отклонена 3 раза, в итоге получила Best Paper Award на Usenix ATC
- Десятки реализаций (LogCabin, etcd, hashicorp/raft)
- <https://raft.github.io/>





# Репликация автомата



- Все серверы (копии автомата) выполняют команды из журнала
- Консенсус относительно значений, записываемых в каждую позицию журнала

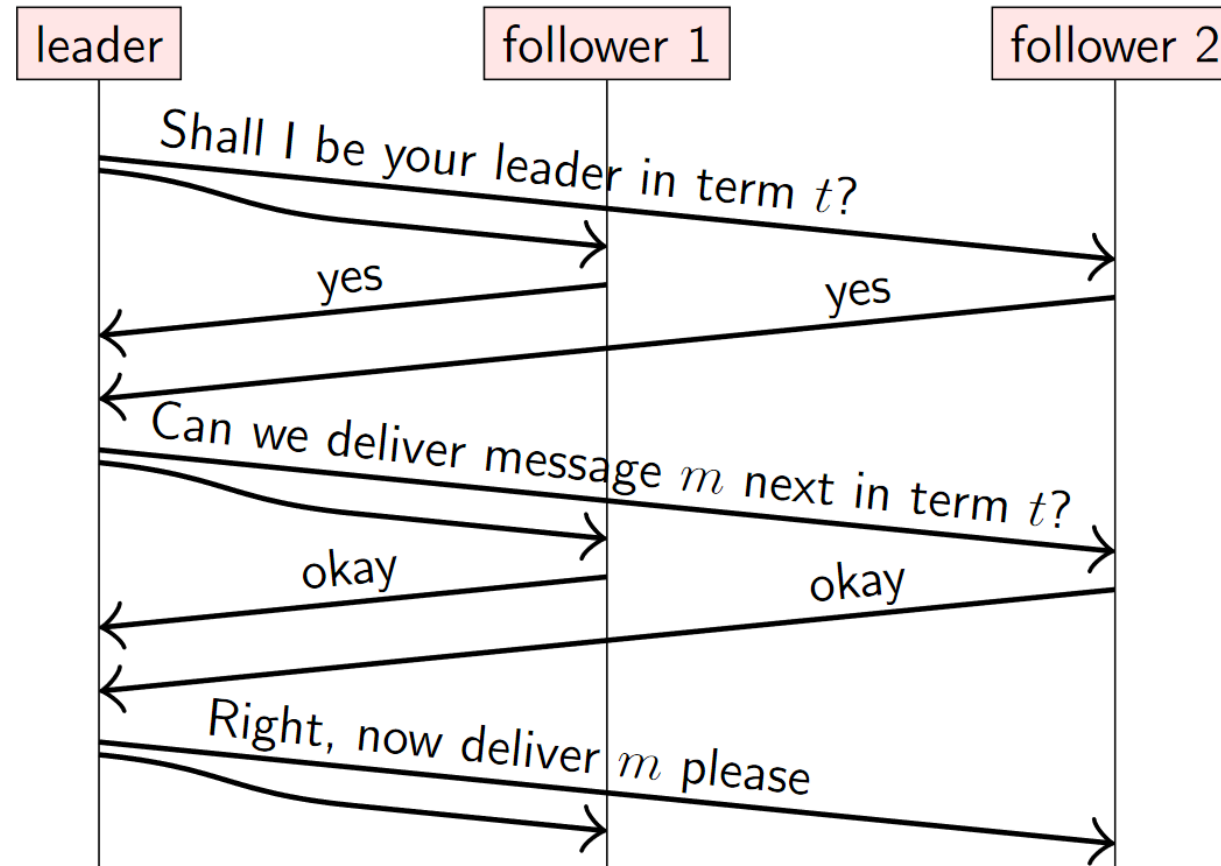
# Элементы Raft

- Выборы лидера
- Репликация журнала (нормальный режим)
- Обеспечение согласованности журнала
- Безопасная смена лидера

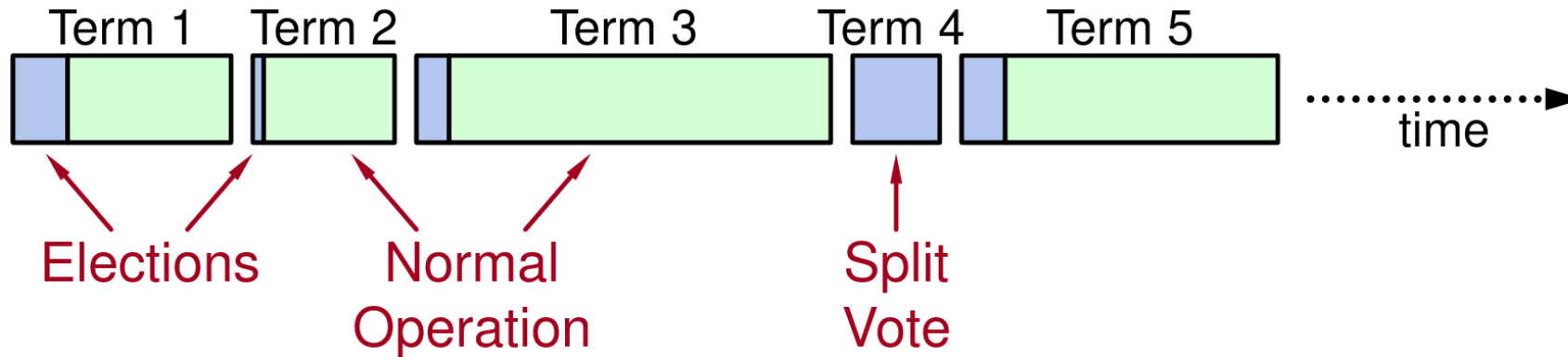
# Лидер в алгоритме консенсуса

- Лидер уникален только в пределах некоторой эпохи
  - С эпохой связан уникальный номер, который монотонно растет
- При отказе текущего лидера выбирается лидер новой эпохи
  - Номер эпохи увеличивается, лидер выбирается кворумом голосов
- Несколько узлов в системе могут считать себя лидерами
  - В случае конфликта преимущество имеет лидер с большим номером эпохи
- Перед принятием решения лидер должен проверить свое лидерство
  - Отправить предлагаемое значение всем узлам
  - Получить положительный ответ от кворума узлов
  - Узел отклоняет запрос, если знает лидера с большим номером эпохи
- **Кворумы голосований за лидера и за значение должны пересекаться**

# Лидер в алгоритме консенсуса

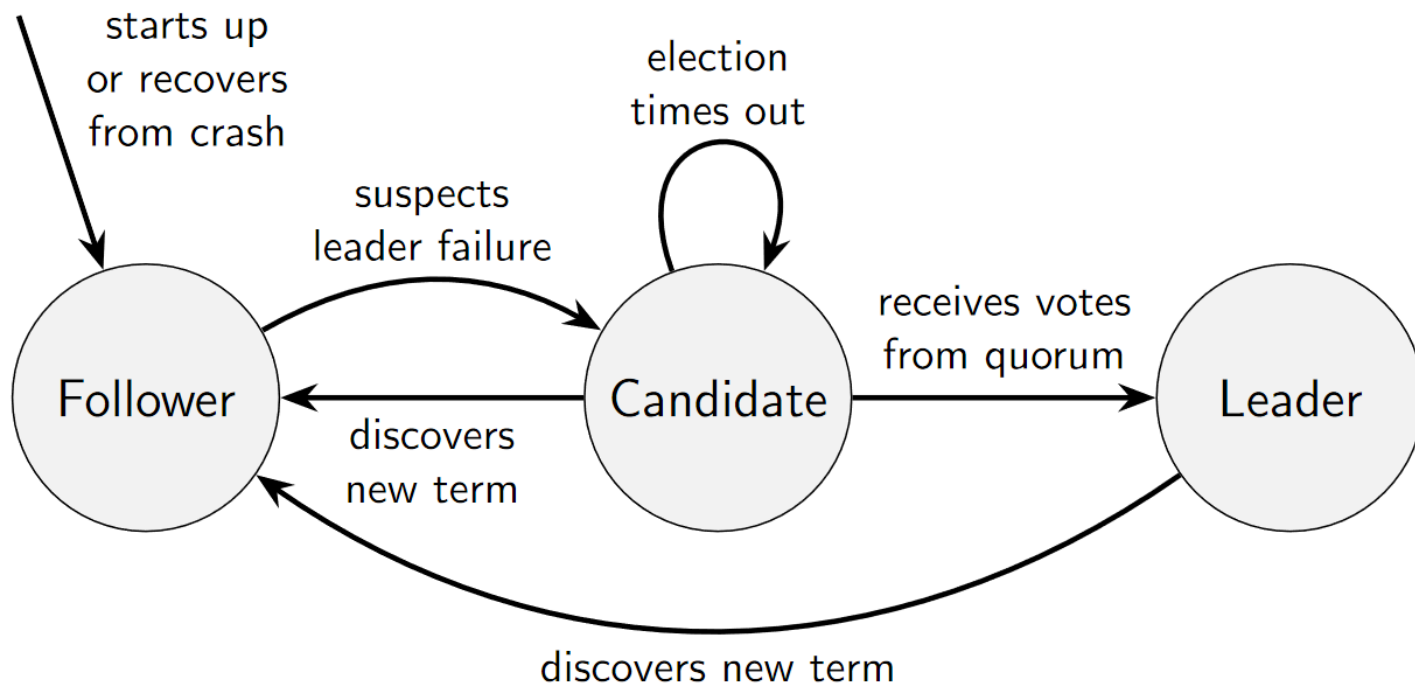


# Эпохи

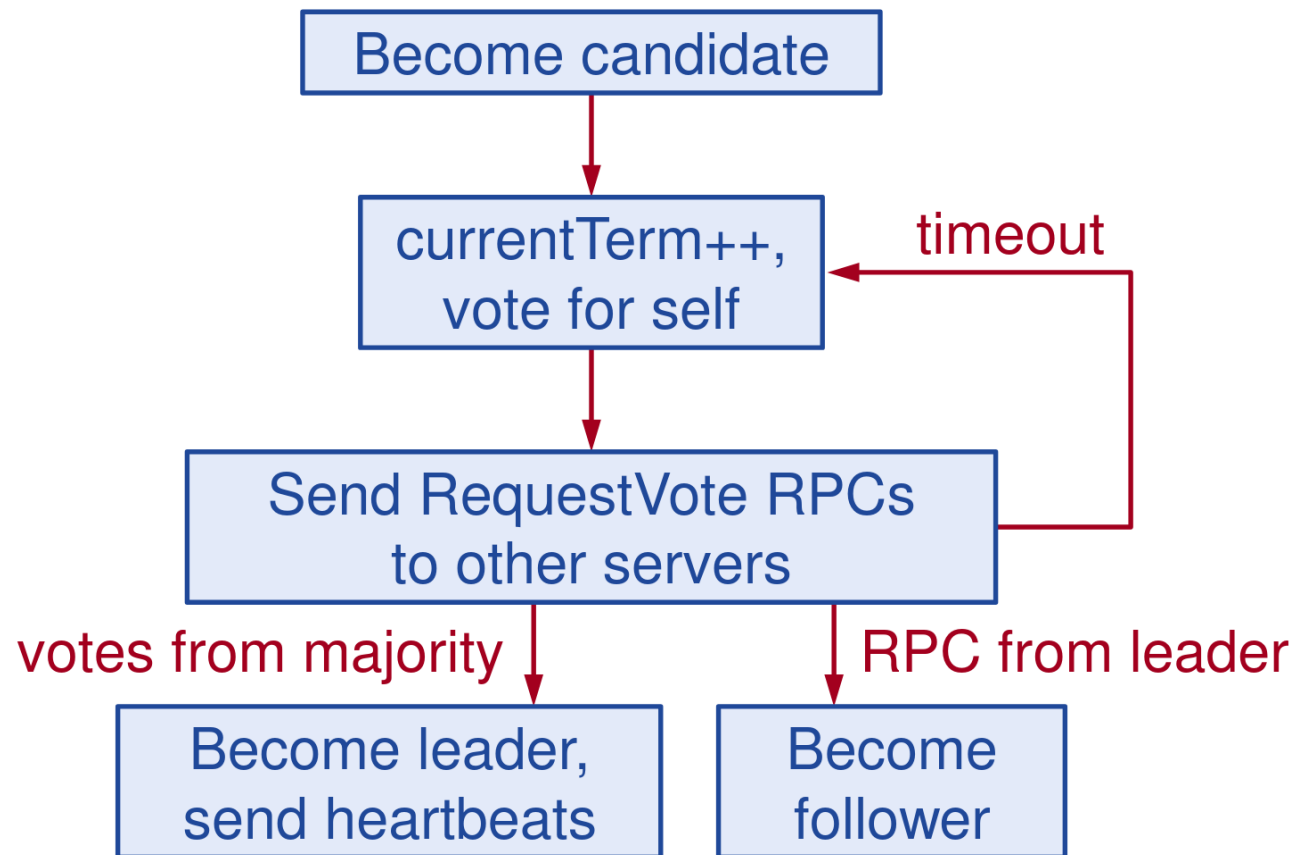


- В каждой эпохе (term) есть не более одного лидера
- Каждый узел хранит номер текущей (на его взгляд) эпохи
- Узлы обмениваются значениями эпох в сообщениях
- Эпохи позволяют избавиться от устаревшей информации

# Состояния узла в Raft



# Выборы лидера



# Корректность выборов

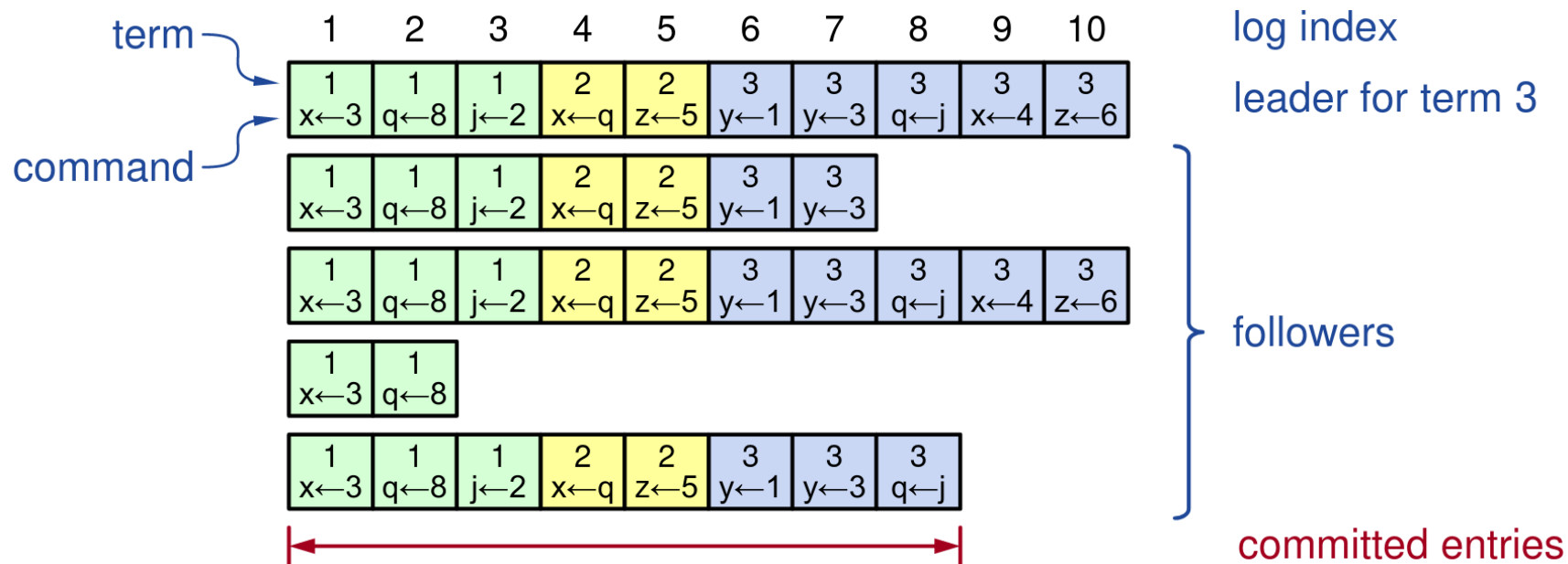
- Не более одного победителя в рамках эпохи (safety)
  - Узел голосует не более 1 раза в эпохе, голос сохраняется на диск
  - Два кандидата не могут одновременно набрать большинство голосов
- Какой-то кандидат в конечном счете побеждает (liveness)
  - Таймаут для запуска выборов выбирается случайно из интервала  $[T, 2T]$
  - Один узел обычно выигрывает выборы до того, как истечет таймаут у остальных
  - Значение  $T$  должно быть  $\gg$  времени рассылки сообщения
  - Подход с рандомизацией гораздо проще, чем ранжирование узлов



# Репликация журнала (норм. режим)

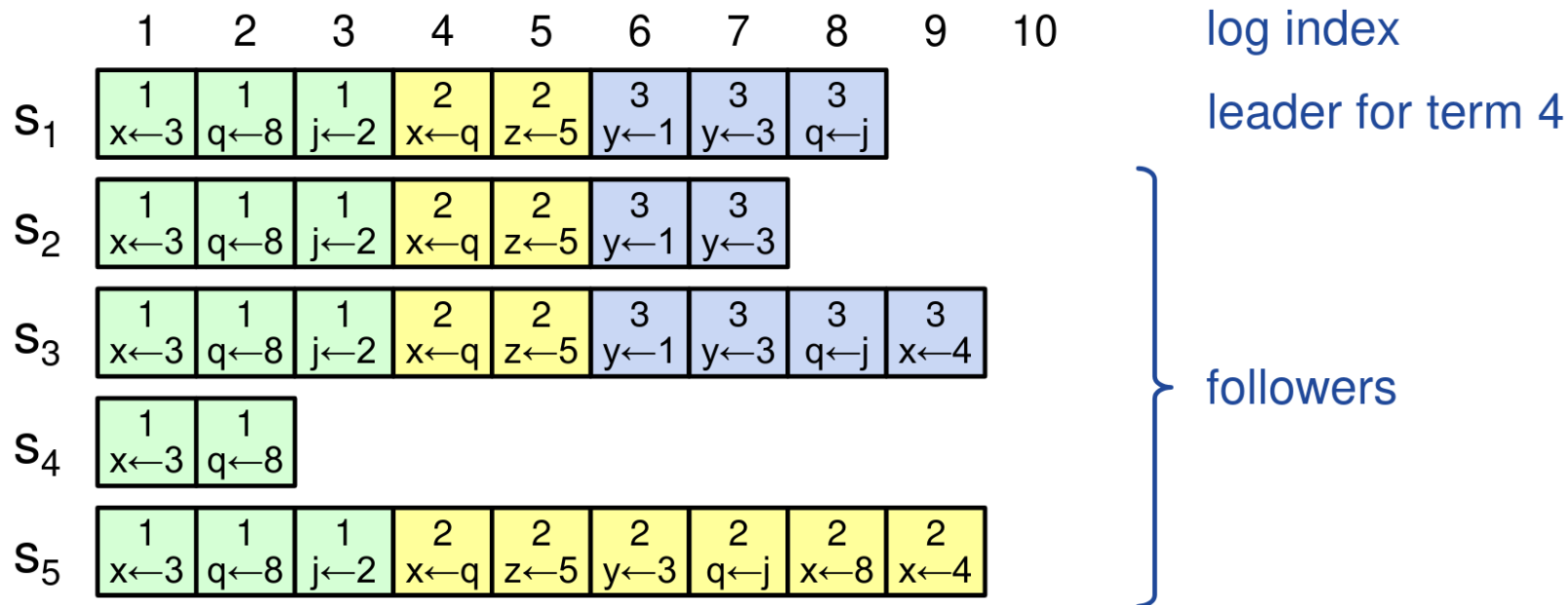
- Клиент отправляет команду лидеру
- Лидер записывает команду в конец своего журнала
- Лидер отправляет подчиненным сообщения ***AppendEntries***
- Как только большинство подчиненных ответило
  - Лидер выполняет команду на своей копии автомата (commit) и отвечает клиенту
  - Лидер уведомляет о **committed** записи подчиненных
  - Подчиненные выполняют команду на своих копиях автомата
- Сбой или отставание подчиненного?
  - Лидер повторно отправляет сообщения до достижения успеха
- При отсутствии отказов алгоритм оптимален в плане числа сообщений

# Структура журнала



- Запись включает индекс, эпоху и команду
- Журнал хранится на диске и должен переживать отказы
- Запись считается **committed**, если она записана в журналы большинства узлов

# Согласованность журнала




- Отказы могут приводит к нарушению согласованности копий журнала
- Raft восстанавливает согласованность журнала в ходе своей работы

# Гарантируемые свойства

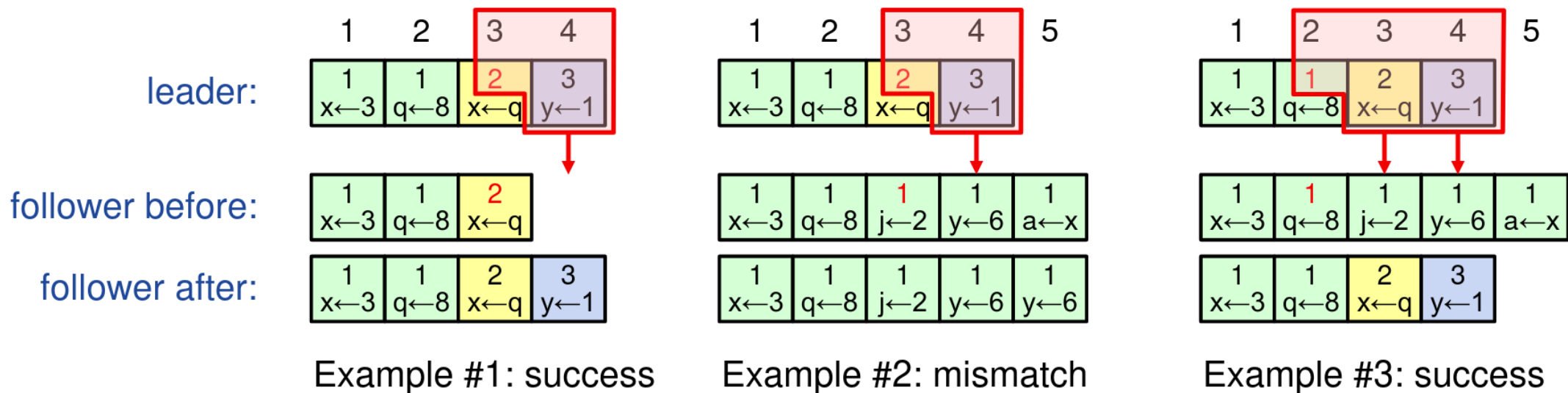
- Если записи на разных узлах имеют одинаковые индекс и эпоху, то
  - они содержат одну и ту же команду
  - все предыдущие записи в журналах идентичны

1	2	3	4	5	6	7	8	9	10
1 $x \leftarrow 3$	1 $q \leftarrow 8$	1 $j \leftarrow 2$	2 $x \leftarrow q$	2 $z \leftarrow 5$	3 $y \leftarrow 1$	3 $y \leftarrow 3$	3 $q \leftarrow j$	3 $x \leftarrow 4$	3 $z \leftarrow 6$
1 $x \leftarrow 3$	1 $q \leftarrow 8$	1 $j \leftarrow 2$	2 $x \leftarrow q$	2 $z \leftarrow 5$	3 $y \leftarrow 1$	4 $x \leftarrow z$	4 $y \leftarrow 7$		



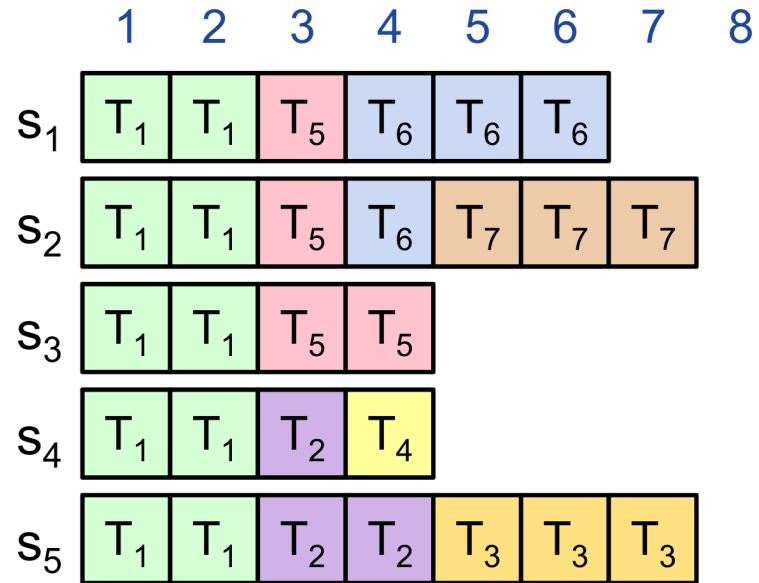
- Если запись *committed*, то все предыдущие записи тоже *committed*

# Проверки при передаче изменений



- Сообщение ***AppendEntries*** содержит индекс и эпоху предыдущей записи
- Подчиненный отвергает запрос, если предыдущая запись не совпадает
- Механизм обеспечивает требуемые гарантии согласованности

# Смена лидера



- Старый лидер мог оставить частично реплицированный журнал
- Новый лидер просто начинает выполнять нормальный режим
- Журнал лидера - "истина в последней инстанции"
- Журналы подчиненных в конце концов станут идентичны журналу лидера

# Безопасность (safety)

Как только команда была выполнена автоматом, никакой другой автомат не может выполнить другую команду для данного индекса в журнале

- Лидер никогда не перезаписывает записи в своем журнале
- Только записи из журнала лидера могут быть *committed*
- Запись должна быть *committed* до выполнения этой команды автоматом
- Как только запись стала *committed*, все последующие лидеры должны хранить её

# Выбор наилучшего лидера

- Узлы с неполными журналами не должны быть выбраны
- Кандидаты сообщают всем индекс и эпоху последней записи в их журнале
- Узел не отдает голос за кандидата, чей журнал "менее полный":

$(lastLogTerm_C < lastLogTerm_V)$  **or**

$(lastLogTerm_C = lastLogTerm_V \text{ and } lastLogIndex_C < lastLogIndex_V)$

## Leader election for term 4:

	1	2	3	4	5	6	7	8	9
s <sub>1</sub>	1	1	1	2	2	3	3	3	
s <sub>2</sub>	1	1	1	2	2	3	3		
s <sub>3</sub>	1	1	1	2	2	3	3	3	3
s <sub>4</sub>	1	1	1	2	2	3	3	3	
s <sub>5</sub>	1	1	1	2	2	2	2	2	2

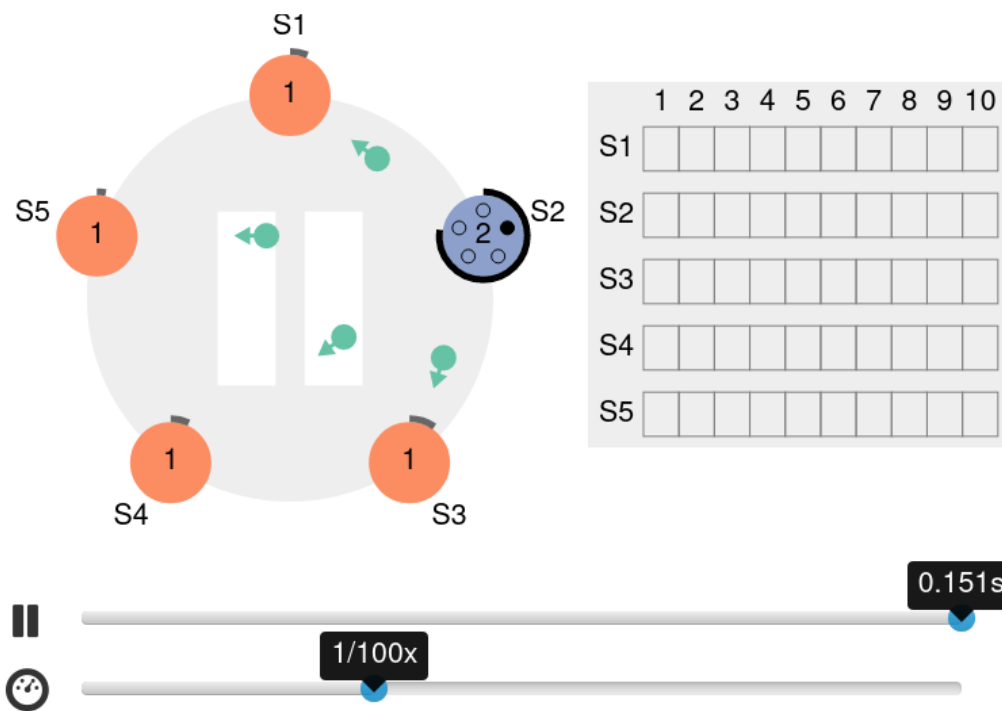


# Другие элементы Raft

- Изменение конфигурации кластера
- Сжатие журнала (log compaction)
- Протокол работы клиента

См. [статью про Raft](#)

# Визуализация работы Raft



- <http://thesecretlivesofdata.com/raft/>
- <https://raft.github.io/raftscope/>

# Недостатки алгоритмов консенсуса

- Не масштабируются горизонтально
  - производительность ограничена одним сервером
  - можно масштабировать только путем шардинга (несколько консенсус-кластеров)
- Накладные расходы на синхронизацию перед принятием значения
  - полусинхронная репликация, связанная с этим задержка
- Требуется большинство узлов
  - конфигурация из не менее 3 серверов (3 или 5 на практике)
  - недоступность одной части в случае разделения сети
- Требуется достаточно стабильное окружение
  - в основном рассчитаны на статическую конфигурацию кластера
  - нестабильность сети может приводить к отсутствию прогресса

# Сервисы для координации

- Реализация примитивов для координации распределенных процессов в виде отказоустойчивого сервиса
  - Обычно в виде реплицированного key-value хранилища (3-5 узлов)
  - Данных немного, они помещаются в памяти и не изменяются очень часто
  - Согласованность данных и отказоустойчивость достигаются с помощью консенсуса
- Варианты применения
  - Выборы лидера, блокировки, фиксация транзакций, очереди...
  - Сервис именования, хранение конфигурации, обнаружение отказов и group membership
- Примеры
  - Chubby (Paxos), ZooKeeper (Zab), Consul (Raft), etcd (Raft)

# Литература

- Kleppmann M. Distributed Systems (части 6 и 7.1)
- Ongaro D., Ousterhout J. In Search of an Understandable Consensus Algorithm (2014)

# Литература (дополнительно)

- Kleppmann M. Designing Data-Intensive Applications (глава 9)
- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms (8.2, 8.5)
- Howard H., Mortier R. Paxos vs Raft: Have we reached consensus on distributed consensus? (2020)
- Raft does not Guarantee Liveness in the face of Network Faults (2020)