Distributed systems

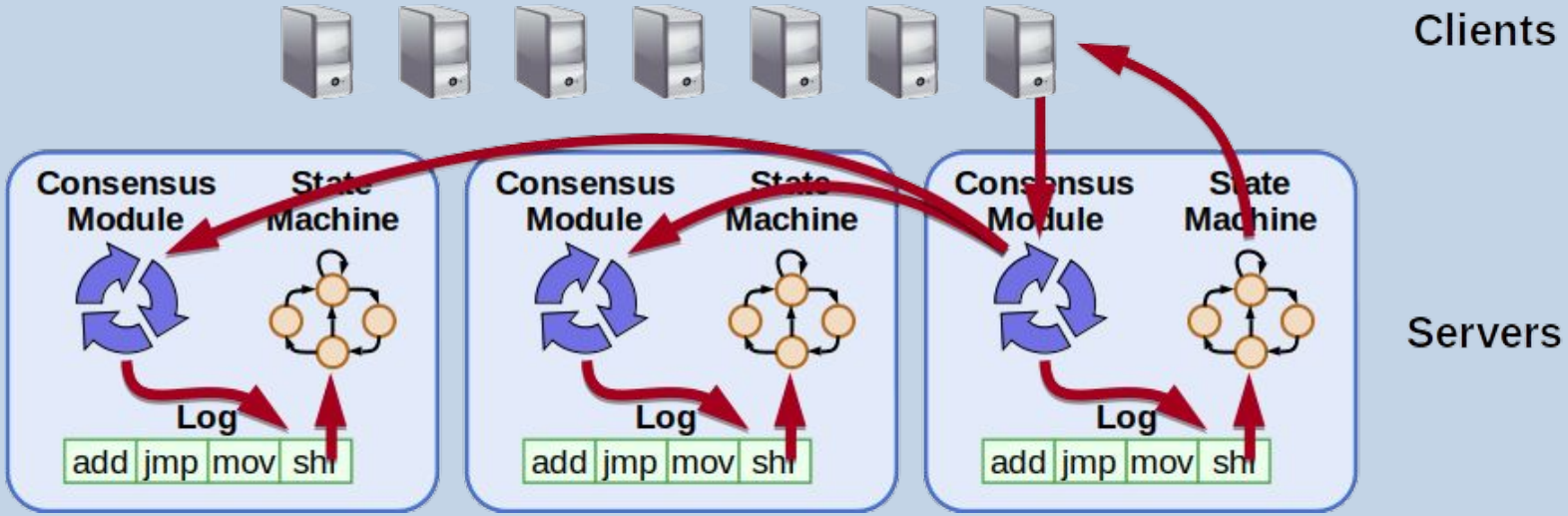
Consensus algorithm

Неформально, консенсус - способ обеспечить согласование какого-то действия между всеми участниками

Свойства, которым должен удовлетворять алгоритм консенсуса:

- *Uniform agreement* - никакие две ноды не принимают разные решения
- *Integrity* - никакая нода не принимает решение дважды
- *Validity* - если нода приняла какое-то решение, то оно было предложено кем-то из участников
- *Termination* - работающая исправно нода в конечном счете принимает какое-то решение

Raft consensus algorithm



- *Replicated log => replicated state machine*
  (Все ноды исполняют одни и те же команды в одном и том же порядке)
- Модуль консенсуса обеспечивает правильную репликацию лога
- Система работает пока работает большинство ее узлов

Способы достижения консенсуса

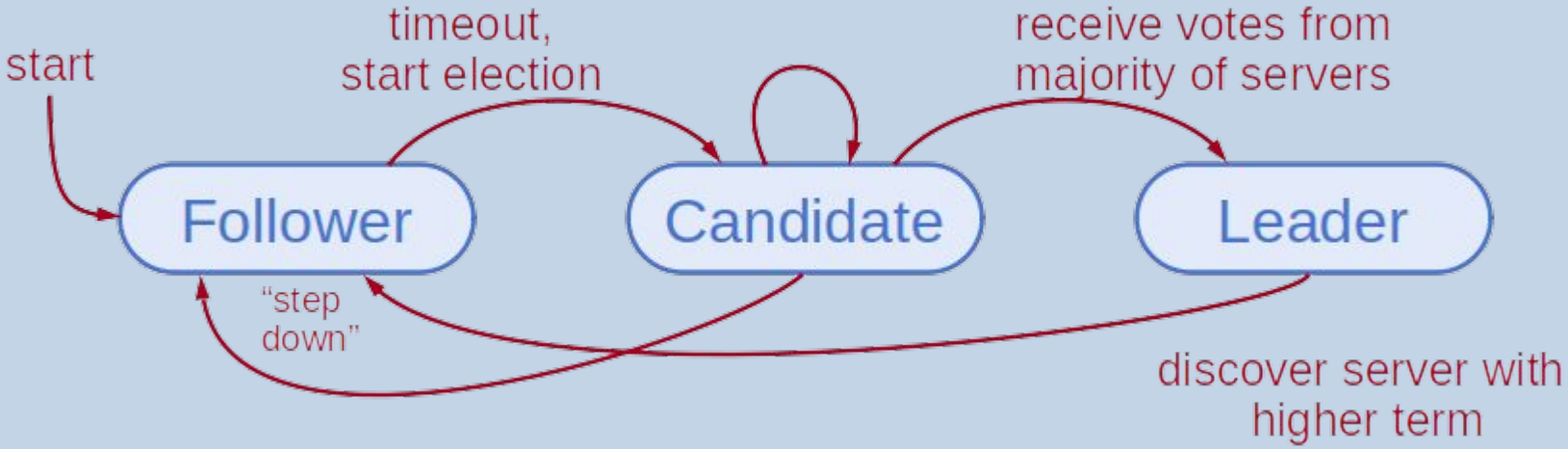
- Симметричный, *leaderless*
  - Нет мастера, все ноды равноценны
  - Клиенты могут отсылать запросы на любую ноду
- Асимметричный, *leader based*
  - Все команды исходят от мастер-ноды
  - Запросы клиента идут на мастер
- *Raft* использует *leader based* подход

Основные стадии Raft

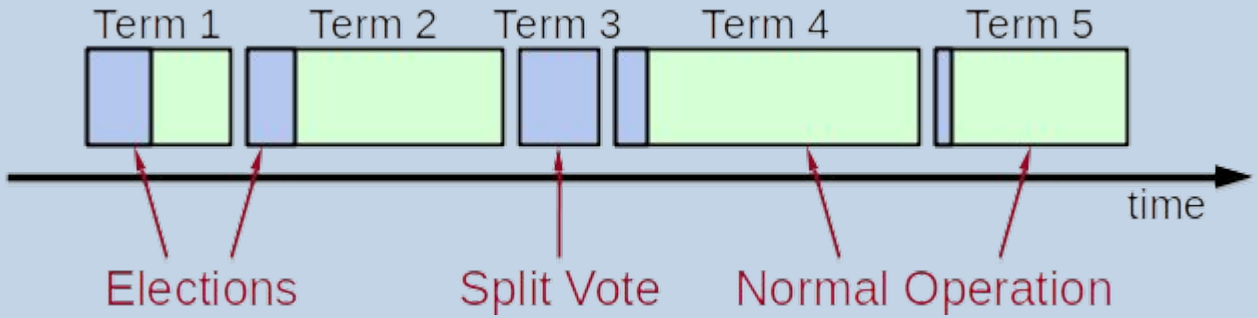
1. Выборы лидера:
   a. Выбор одной ноды в качестве лидера
   b. Обнаружение сбоя лидера и перевыборы
2. Выполнение команд (репликация лога)
3. Поддержание сохранности и консистентности лога после перевыбора мастера
4. Нейтрализация старых лидеров
5. Взаимодействие клиентов: *linearizable* consistency
6. Изменение конфигурации: добавление или удаление нод

Состояния нод

- В любой момент времени каждая нода находится в одном из трех состояний
  - *Leader* - обрабатывает все запросы клиентов, реплицирует лог
    - Не более одного лидера
  - *Follower* - пассивный, только принимает запросы от leader
  - *Candidate* - состояние используется для выбора лидера
- В нормальном состоянии: 1 leader n-1 followers



Семестры (terms)



- Время разделено на интервалы
  - Выборы
  - Состояние выполнения операций
- Не большего одного лидера в интервале
- Некоторые интервалы не имеют лидера (неуспешные выборы)
- Каждая нода поддерджывает значние текущего интервала
- Роль интервалов: точно идентифицировать информацию

Heartbits and timeouts

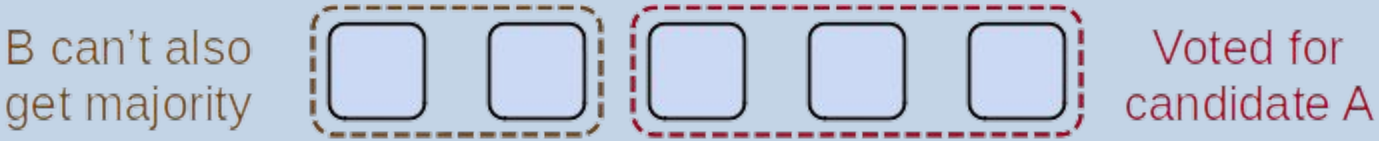
- Изначально все ноды находятся в состоянии *follower*
- *Followers* ожидают команд от *leader* или *candidates*
- *Leader* должен отправлять сигналы *heartbits*, чтобы поддерживать свою роль
- Если у *follower* наступил *timeout* без *heartbit* от лидера
  - нода считает, что лидер потерпел сбой
  - начинает новые выборы
  - таймауты обычно 100-500ms

Алгоритм проведения выборов

- Увеличиваем счетчик семестра
- Переход в состояние Candidate
- Отдаем голос за себя
- Отправляет запросы с предложением голосовать за себя на все ноды, повторяет пока не...
  - Нода получит большинство ответов от других
    - Становится лидером
    - Отправляет *heartbeat* на все ноды
  - Получит *heartbeat* от лидера
    - Возвращается в состояние *follower*
  - Никто не собрал большинство, произошел *election* таймаут
    - Увеличивает счетчик семестра, начинает новые выборы

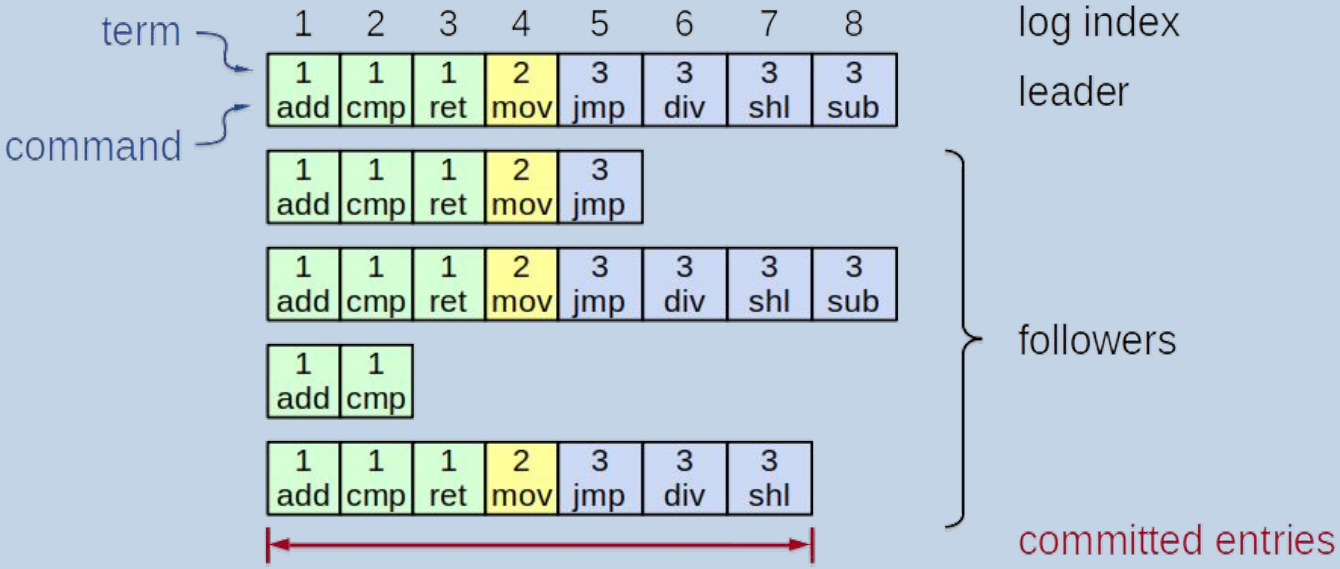
Инварианты выборов

- **Safety**: не больше одного лидера в семестре
  - Каждая нода может отдать только один голос в семестр (решение записывается на диск)
  - Две ноды не могут набрать большинство в одном семестре:



- **Liveness**: какая нода в состоянии candidate должна выиграть в конечном счете
  - Ноды выбирают разный таймаут в промежутке [T, 2T]
  - Одна из нод просыпается и выигрывает выборы пока другие еще спят
  - Алгоритм хорошо работает, если T >> broadcast time

Log structure



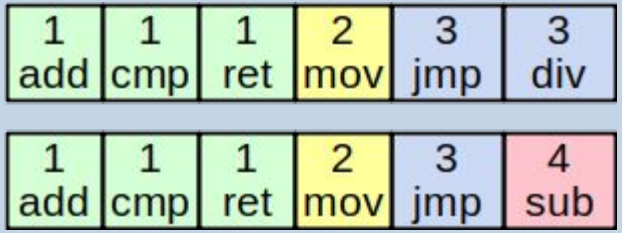
- Каждая запись в логе - индекс, номер семестра и команда
- Лог хранится на диске, который устойчив к отказам
- Закомиченная запись в логе будет хранится на большинстве серверов

Работа в режиме выполнения операций

- Клиент посылает команды лидеру
- Лидер записывает команду себе в лог
- Лидер отсылает команду *AppendEntries* всем фоловерам
- Если значение коммитится
    - Лидер посылает команду на выполнение фоловерам, отдает результат клиенту
    - Лидер информирует фоловеров о коммите последующими вызовами
    - Фолловеры записывают закоммиченную команду к себе в лог
- Лидер повторяет команды пока большинство нод не примут

Log consistency

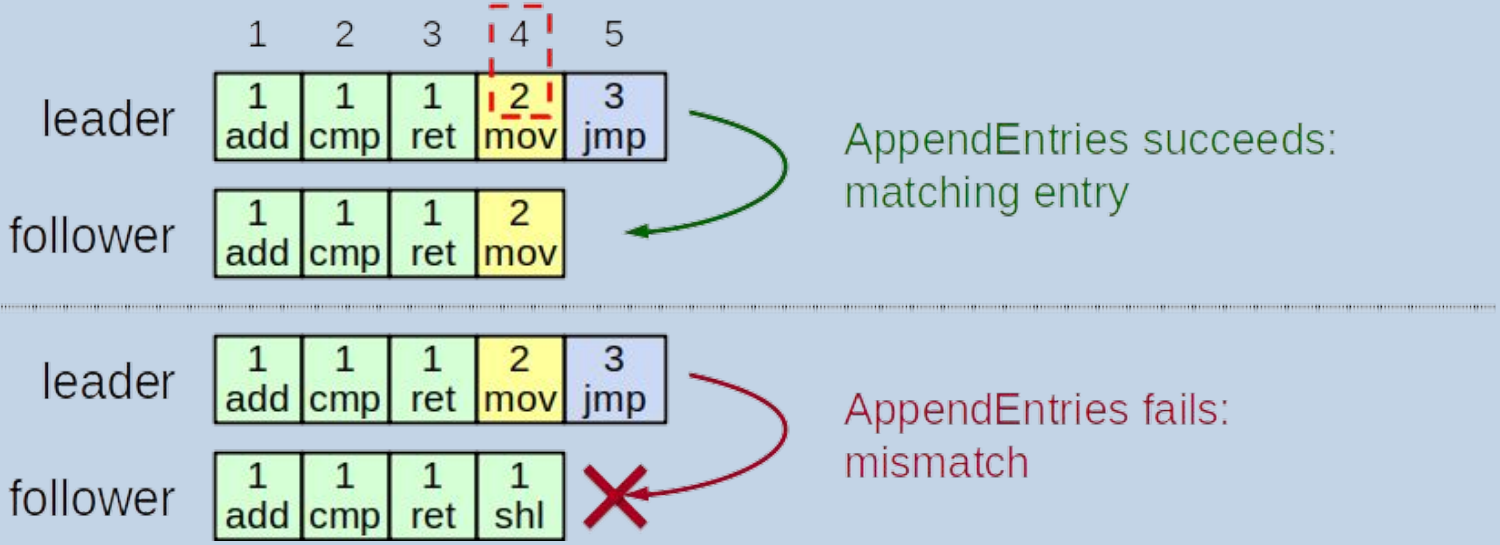
- Если записи в логе на разных нодах имеют один и тот же индекс и номер семестра
  - Они хранят ту же команду
  - Логи совпадает для всех предыдущих значений



- Если данная запись в логе закоммичена, то и все предыдущие записи также закомичены

AppendEntries Consistency Check

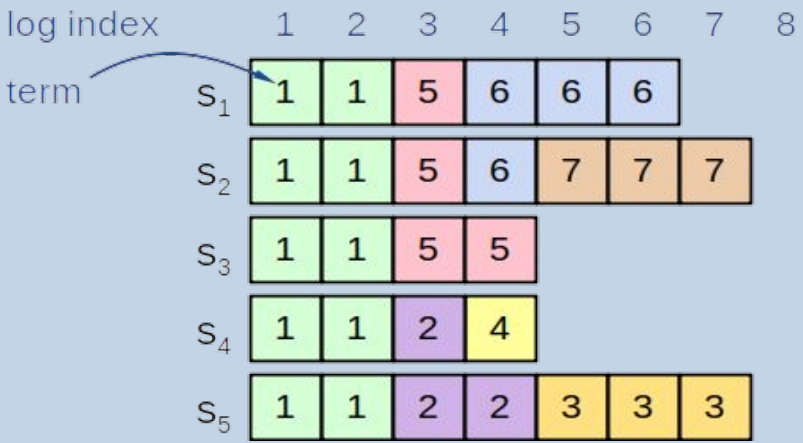
- Каждый вызов *AppendEntries* содержит индекс, номер семестра предыдущей записи
- У фолловера в логе должна содержаться та же запись, иначе он не не примет запрос лидера
- По индукции можно доказать консистентность лога



Смена лидера

В начале семестра после выборов:

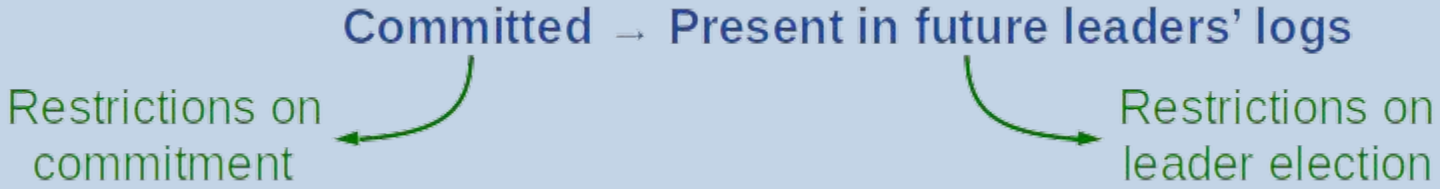
- У старого лидера могут остаться частично реплицированные записи
- Нет специальных действий, которые нужно выполнить новому лидеру
- Лог лидера содержит актуальные записи
- Логи фолловеров в конечном счете сравняются с мастером
- Много отказов нод могут привести к серьезному расхождение логов



Safety гарантии

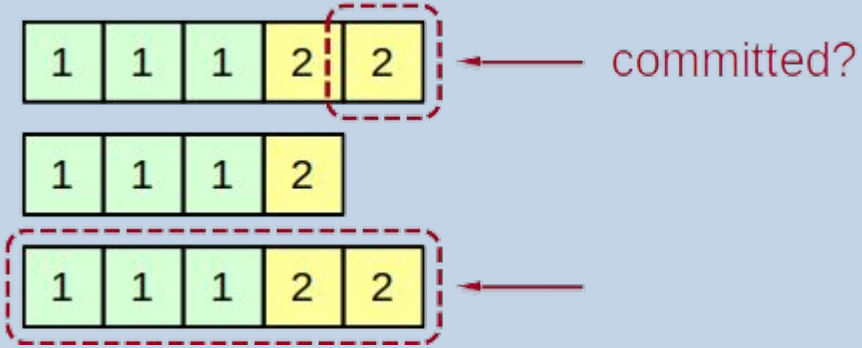
Если команда была записана в лог...

- Raft safety property:
  - Если лидер принял решение о коммите записи, то эта запись будет содержаться в логе у всех последующих лидеров
- Лидер никогда не перезаписывает записи в своем логе
- Только записи из лога лидера могут быть закоммичены
- Запись должна быть закоммичена прежде, чем примениться к state машине



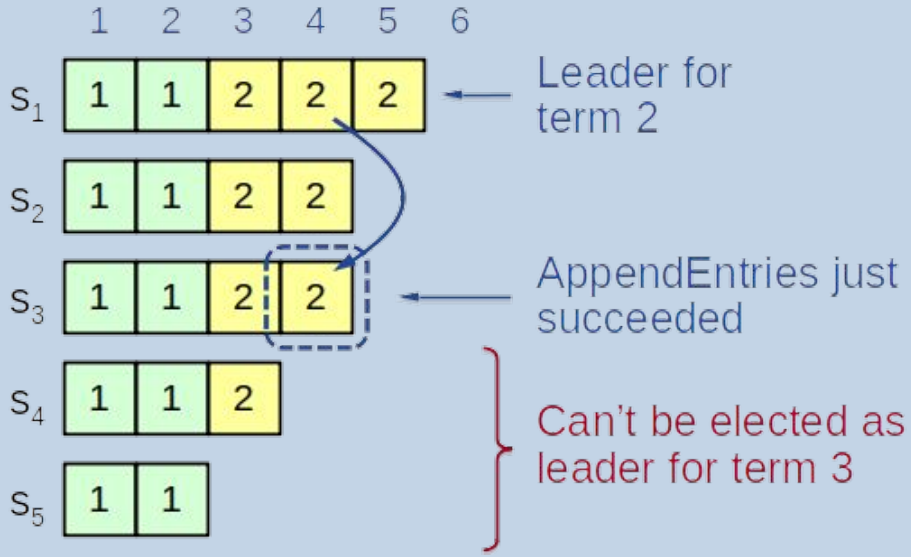
Выбор наиболее актуального лидера

- Нельзя понять какая запись закомиченна



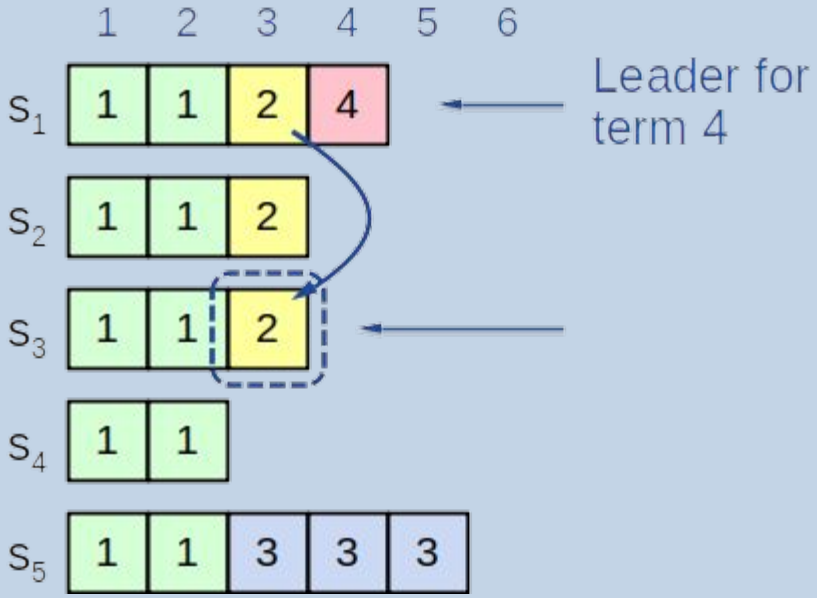
- Во время выборов выбираем кандидата с наиболее полным логом
  - Кандидаты отсылают индекс и семестр последней записи в логе
  - Принимающая нода не принимает запрос, если ее лог более актуальный:
    (lastTermV > lastTermC) ||
    (lastTermV == lastTermC) && (lastIndexV > lastIndexC)
  - В результате новый лидер должен содержать наиболее полный лог среди голосовавших нод

Лидер коммитит запись в своем семестре



- Safe property: Лидер в 3 семестре будет содержать запись 4

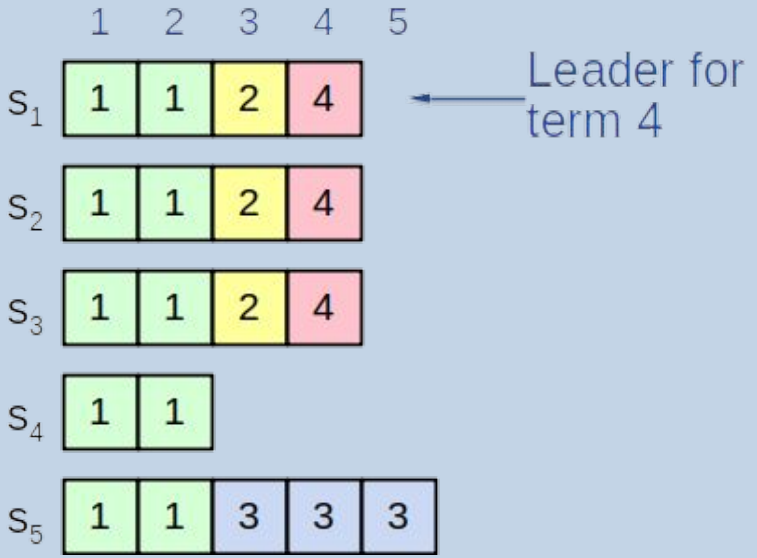
Лидер коммитит запись из предыдущего семестра



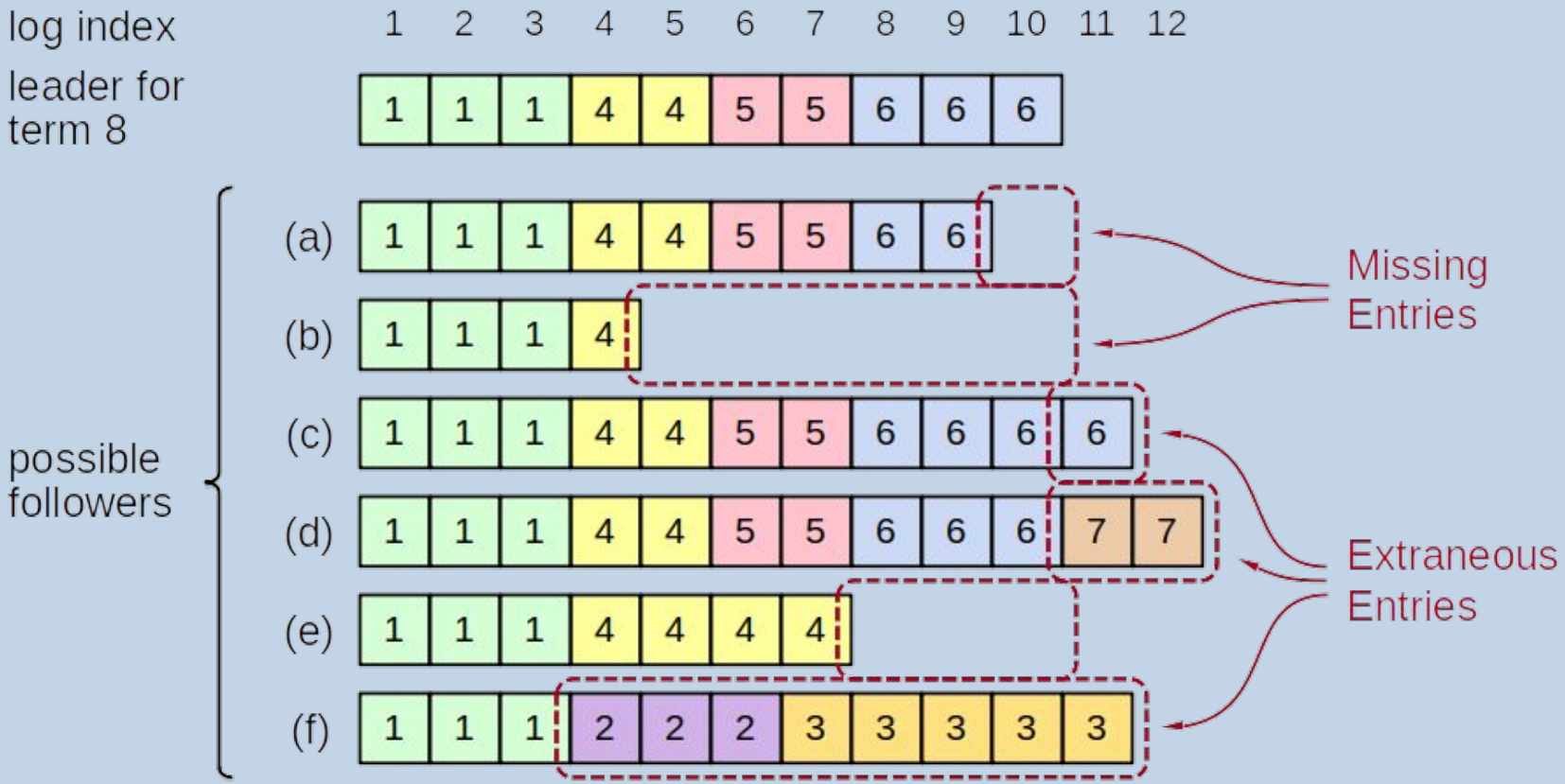
- Запись 3 не закомиченна надежно
  - S5 может быть выбрана лидером в 5 семестре
  - Если ее выберут она перезапишет запись 3 на остальных нодах

Корректировка правил коммита

- Чтобы лидер счел запись закомиченной:
  - Она должна храниться на большинстве нод
  - Должна быть закомиченна хотя бы одна запись из семестра текущего лидера
- Когда запись 4 закомиченна:
  - S5 не может стать лидером для 5го семестра
  - Записи 3 и 4 закомиченны надежно
- Combination of election rules and commitment rules makes Raft safe

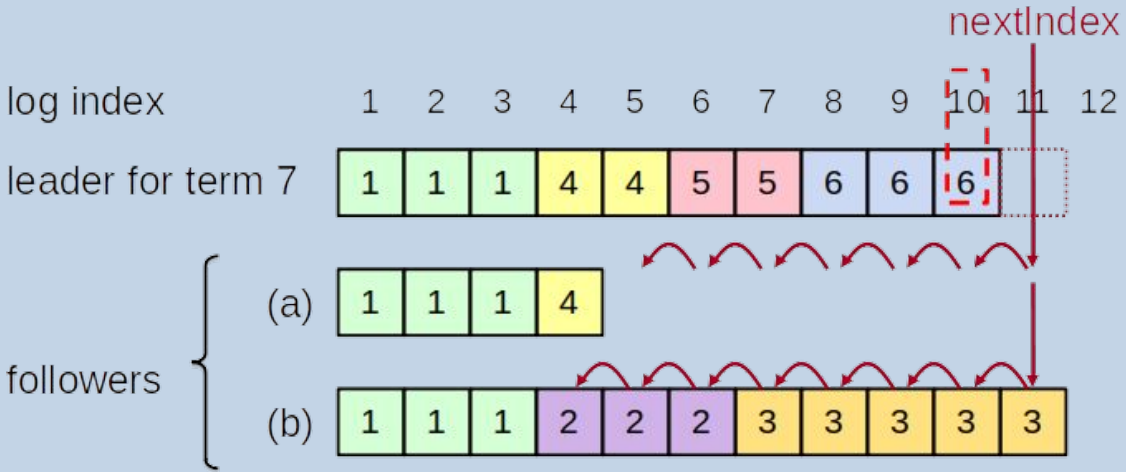


Возможная неконсистентность лога



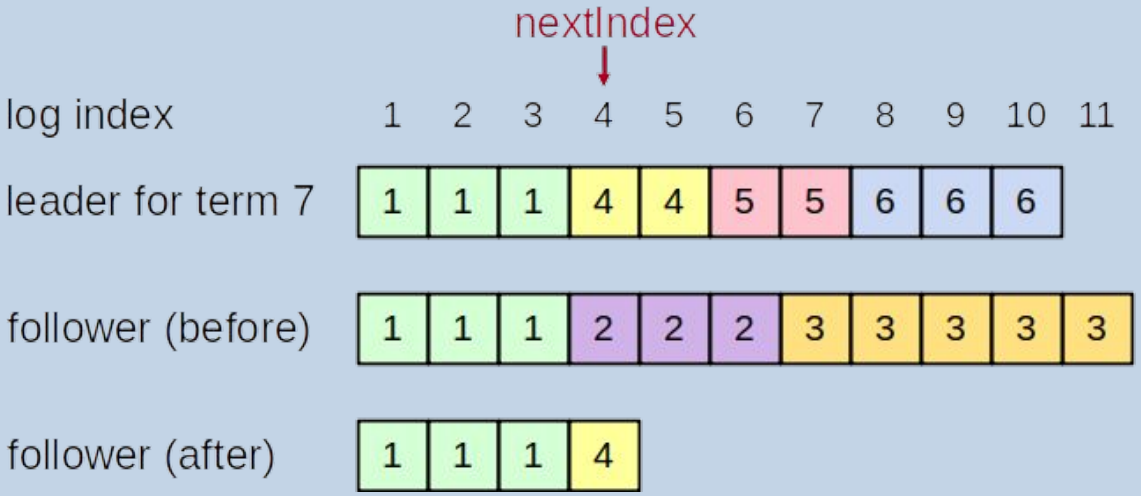
Восстановление логов фоловеров

- Новый лидер восстанавливает логи фолловеров в соответствии со своим
  - Удаляет лишние записи
  - Добавляет отсутствующие
- Лидер хранит индекс следующей записи для каждого фолловера
  - Индекс след. записи для отправки
  - Начинается с 1 + leader’s last index
- Если AppendEntries consistency check fails, декременет и повтор алгоритма



Восстановление логов фоловеров

- Если фолловер перезаписывает какую-то ячейку, он удаляет все последующие



Нейтрализация старых лидеров

- Свергнутый лидер может быть жив:
  - Временно недоступна сеть
  - Другие ноды выбирают нового лидера
  - Старый лидер восстанавливает сеть и пытается писать
- Правила обнаружения старых лидеров
  - Каждый запрос содержит номер семестра
  - Если семестр отправителя старый, его запрос отклоняется,
    нода переходит в состоянии фолловера и инкрементит семестр
  - Если семестр получателя старый,
    он переходит в состоянии фолловера и инкрементит семестр, начинает
    обрабатывать запросы в нормальном режиме
- Свергнутый лидер не может добавлять новые записи в лог

Клиентское взаимодействие

- Отправляем запросы лидеру
  - Если лидер неизвестен, отправляет любой ноде
  - Если данная нода не лидер она переадресует клиента
- Лидер не отвечает клиенту, пока запрос не будет записан в лог, закоммичен и исполнен на state machine
- Если произошел таймаут, или лидер сбойнул
  - Клиент перезапрашивает другой сервер
  - В конечном счете переадресуется на нового лидера
  - Пробуй повторить запрос

Клиентское взаимодействие

- Что если лидер упал после выполнения команды, но перед ответом
  - Мы не должны позволить выполнения запроса дважды
- Решение - добавлять в каждый запрос уникальный id
  - Сервер добавляет id запроса в лог
  - Перед добавлением проверяет записан ли уже этот id
  - Если записан команда игнорируется, возвращается результат прошлой команды
- Exactly-once семантика