14. Устойчивость к произвольным отказам

Сухорослов Олег Викторович

11.12.2023

План лекции

- Произвольные (византийские) отказы
- Задача о византийских генералах
- Протокол Practical Byzantine Fault Tolerance (PBFT)
- Консенсус в сети Bitcoin

Произвольные (византийские) отказы

- Любое отклонение поведения частей РС от заданного протоколом
  - Наиболее общая модель отказов, включающая ранее рассмотренные
  - Процесс может быть активен, но при этом работать некорректно
- Намеренные атаки
  - Пропуск, вставка, изменение, повтор сообщений, сговор...
- Ненамеренные сбои
  - Сбои аппаратуры, баги в ПО...
  - Shuttle Mission STS-124 (2008)
  - Amazon S3 Availability Event (2008)
  - How the Boeing 737 Max Disaster Looks to a Software Developer (2019)

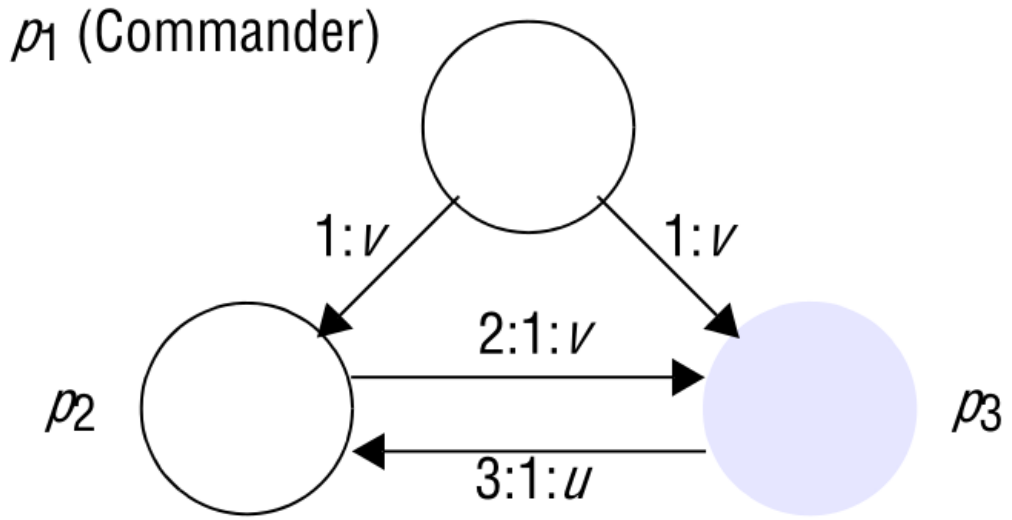
Византийские генералы

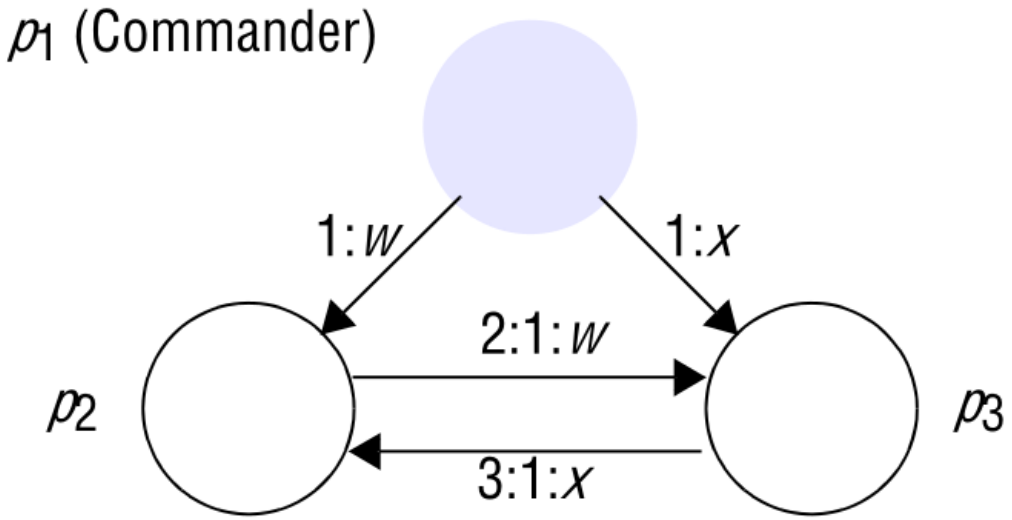
Lamport L., Shostak R., Pease M. The Byzantine Generals Problem (1982)

Задача с командиром

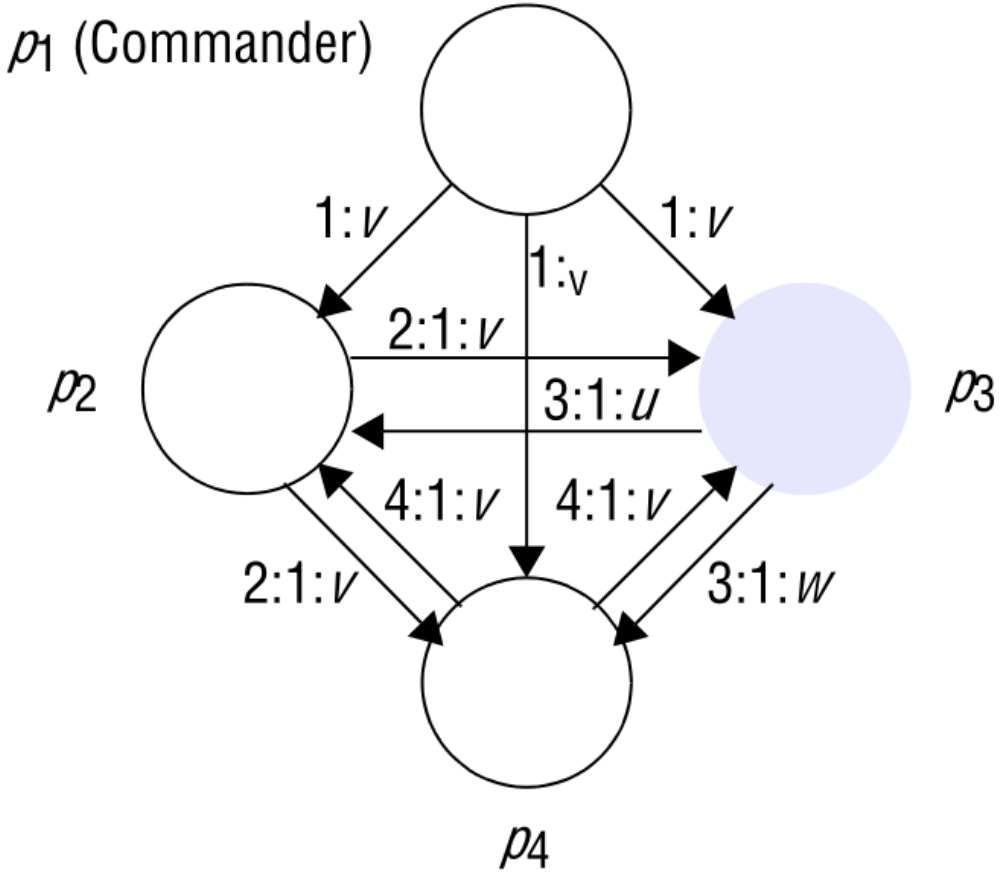
- Главный генерал (командир) рассылает свой приказ (наступать, отступать) остальным генералам (лейтенантам)
- Требования
  - Все верные лейтенанты должны выполнить одинаковый приказ
  - Если командир не был предателем, то верные лейтенанты выполнят его приказ
- Предположения
  - Сообщения доставляются без изменений (контроль целостности)
  - Получатель может определить, кто отправил сообщение (аутентификация)
  - Отсутствие сообщения может быть определено (синхронная модель), в этом случае используется некоторое значение по умолчанию
- Сколько всего генералов N требуется в случае f предателей?

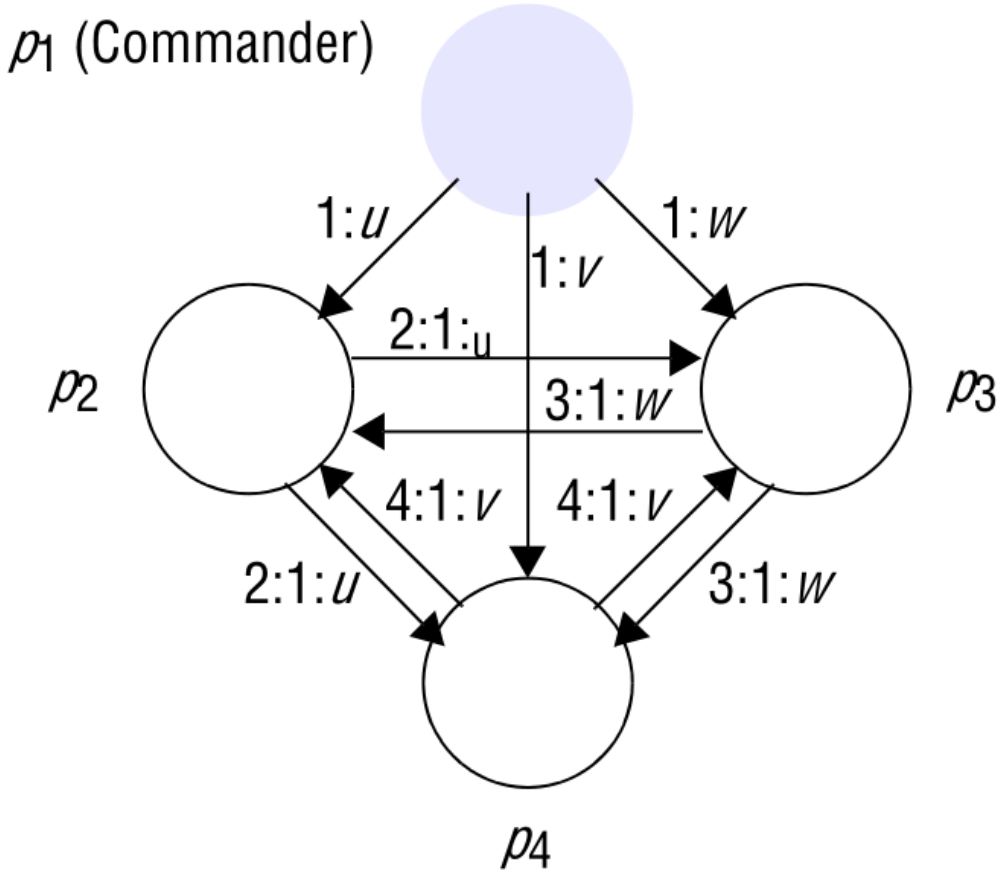
Три генерала (N=3, f=1)





Четыре генерала (N=4, f=1)

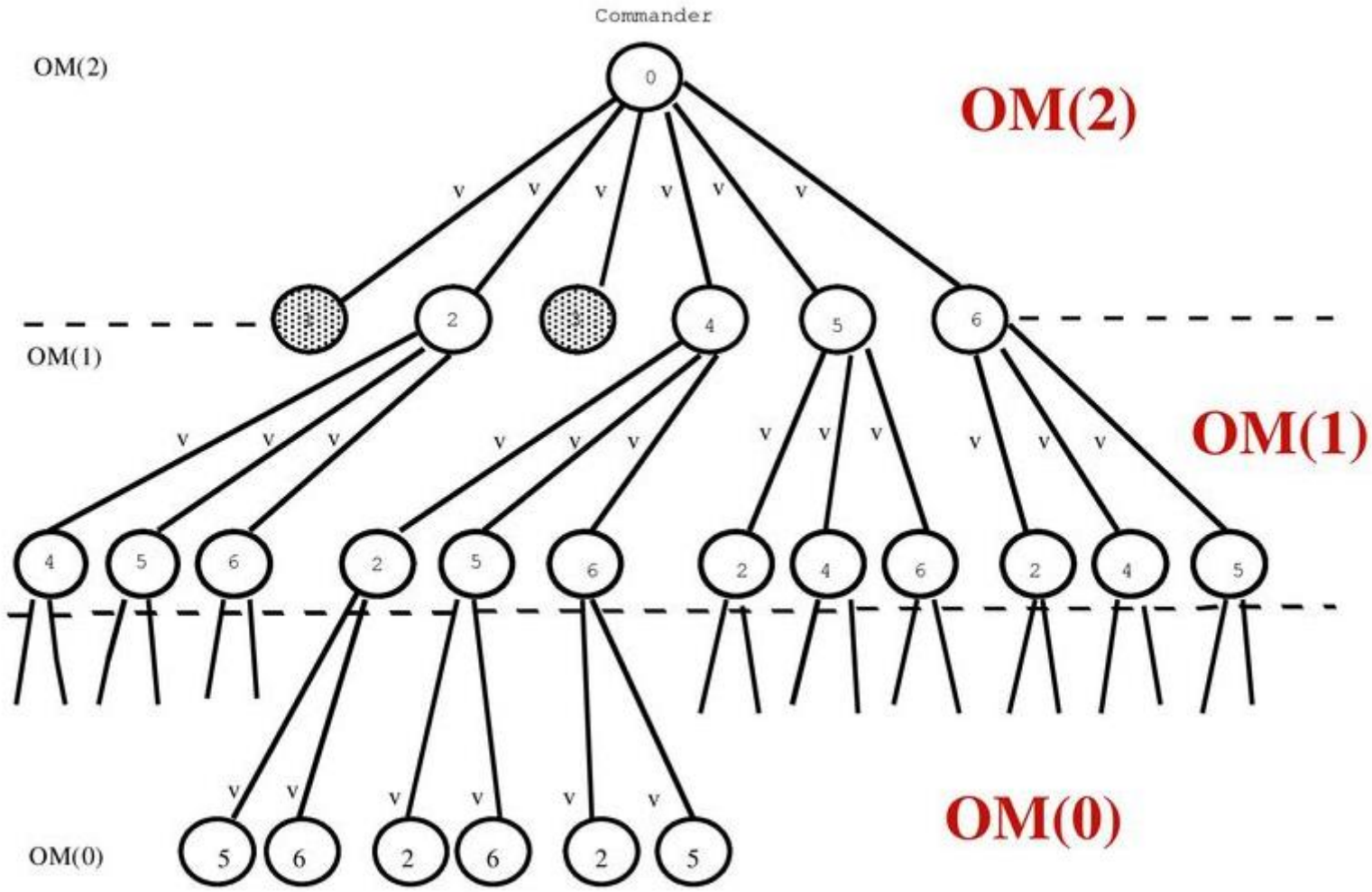




Алгоритм с устными сообщениями

- Алгоритм $OM(0)$
  1. Командир рассылает свое значение всем лейтенантам
  2. Каждый лейтенант использует полученное значение (или RETREAT, если ничего не получил)
- Алгоритм $OM(f)$
  1. Командир рассылает свое значение всем лейтенантам
  2. Пусть v_i – значение, которое лейтенант i получил от командира (или RETREAT, если ничего не получил). Лейтенант i запускает алгоритм $OM(f-1)$, играя в нем роль командира и рассылая v_i остальным $N-2$ лейтенантам.
  3. Пусть v_j – значение, которое лейтенант i получил от лейтенанта j ($j \neq i$) в ходе алгоритма $OM(f-1)$, или RETREAT, если ничего не получил. Лейтенант i использует значение детерминированной функции $majority(v_1, \dots, v_{N-1})$.

Семь генералов (N=7, f=2)

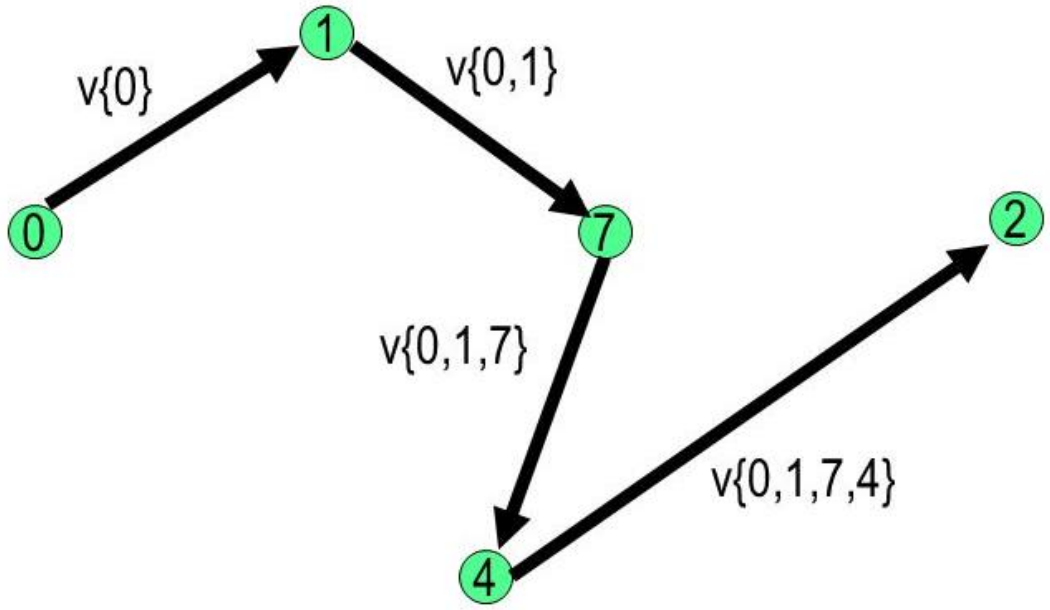


Алгоритм с устными сообщениями

- В случае f предателей требуется
  - не менее $3f + 1$ генералов
  - $f + 1$ раундов взаимодействий
  - $O(N^{f+1})$ сообщений
- Зависимость от таймаутов
  - уязвимость к атакам на корректные процессы (например, DoS)

Подписанные сообщения

- Сообщения подписаны отправителем
- Подпись верного генерала не может быть подделана
- Каждый генерал может проверить подпись и подлинность сообщения



Алгоритм с подписанными сообщениями

- Генерал подписывает и отправляет свой приказ лейтенантам
- Лейтенант i
  - при получении сообщения $v\{0\}$ от командира запоминает v в V_i, подписывает его и отправляет $v\{0, i\}$ остальным лейтенантам
  - при получении сообщения $v\{0, j_1, \dots, j_k\}$ и если v нет в V_i
    - добавляет v в V_i
    - если $k < f$, отправляет сообщение $v\{0, j_1, \dots, j_k, i\}$ лейтенантам, которые еще не подписали сообщение
  - когда сообщений больше не ожидается, вычисляет финальный приказ с помощью функции $choice(V_i)$
- Сообщения, не прошедшие проверку подписей, отбрасываются

Алгоритм с подписанными сообщениями

- Требуется
  - не менее $f + 2$ генералов
  - $f + 1$ раундов взаимодействий
  - $O(N^{f+1})$ сообщений
    - можно уменьшить до $O(N^2)$

Консенсус

- 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 корректным процессом
  - Каждый корректный процесс в конце концов принимает значение
- Традиционные алгоритмы консенсуса (Paxos, Raft)
  - Не поддерживают произвольные отказы
  - Требуют не менее $2f + 1$ процессов

Practical Byzantine Fault Tolerance (PBFT)

- Castro M., Liskov B. Practical Byzantine Fault Tolerance (1999)
- Протокол для state machine replication на системе из N узлов-реплик
  - Репликация сервиса с состоянием и детерминированными операциями над ним
  - Клиенты отправляют запросы с операциями и ожидают ответов
- Предположения
  - Частично синхронная модель, ненадежные каналы… (см. предположения традиционных алгоритмов консенсуса)
  - Подлинность сообщений может быть проверена с помощью цифровых подписей
  - f реплик могут быть подвержены независимым произвольным отказам
- Требует $3f + 1$ реплик для обеспечения требуемых свойств
  - Реплицированный сервис удовлетворяет линеаризуемости (safety)
  - Клиенты в конце концов получают ответы на свои запросы (liveness)

PBFT: Оптимальность числа реплик

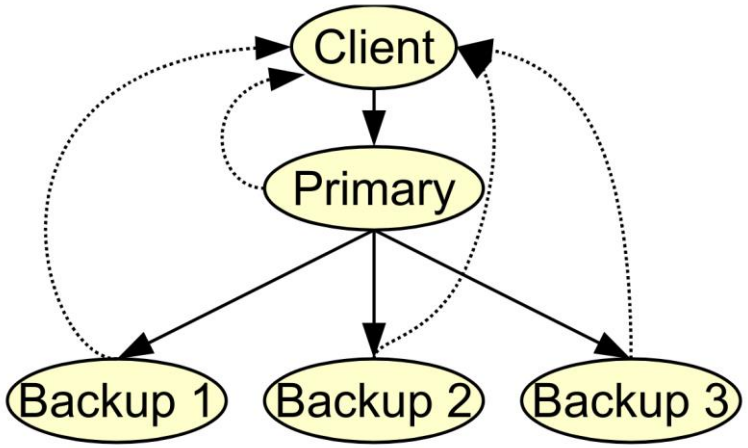
- Для ответа на запрос клиента должно быть достаточно получить ответы от $N - f$ реплик, так как f византийских реплик могут не ответить
- Но f реплик, от которых не пришли ответы, могут просто работать медленно, и среди ответивших опять же могут быть f византийских реплик
- Число ответов от корректных реплик в худшем случае $N - 2f$, при этом оно должно быть больше f, откуда получаем $N > 3f$

PBFT: Конфигурация системы

- Каждая реплика имеет идентификатор i от 0 до $N - 1$
- Конфигурация системы называется **view** (аналог эпохи в Raft)
- Внутри *view* одна из реплик является **главной (primary)**
  - принимает запросы от клиентов
  - присваивает им номера (sequence number) и рассылает по остальным репликам
  - определяется однозначным образом, например $i = view \ mod \ N$
- Остальные реплики играют роль **резервных (backup)**
  - принимают запросы от главной реплики
  - участвуют в процессе принятия решения о коммите операции
- При отказе главного реплики происходит смена *view*

PBFT: Схема обработки запросов

- Клиент отправляет *подписанный* запрос главной реплике
- Главная реплика рассылает запрос резервным
- С помощью кворумов достигается консенсус относительно порядка выполнения запросов и их коммита



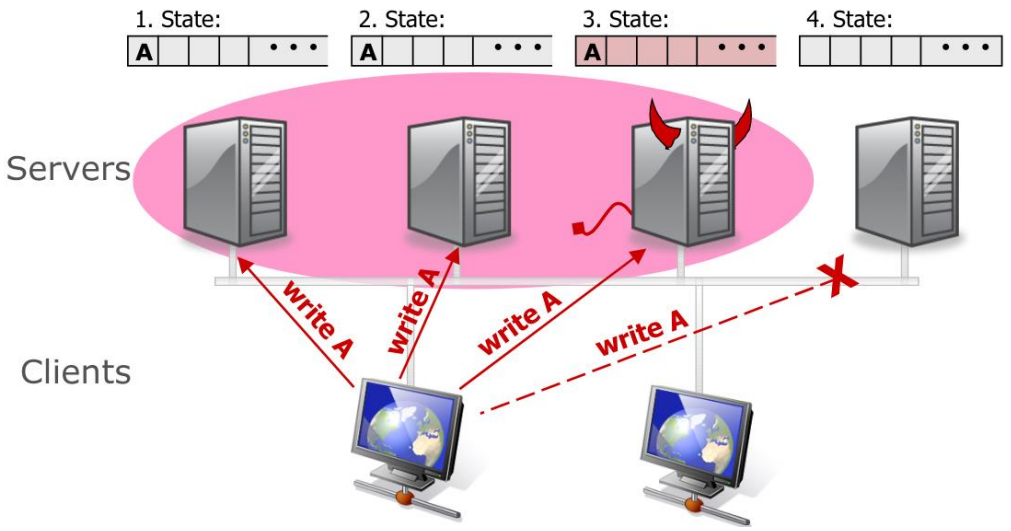
- Как только консенсус достигнут, реплики выполняют запрос и отправляют ответ напрямую клиенту
- Клиент ожидает получения $f + 1$ идентичных ответов от разных реплик
- Если клиент не дождался ответов, то он рассылает запрос всем репликам

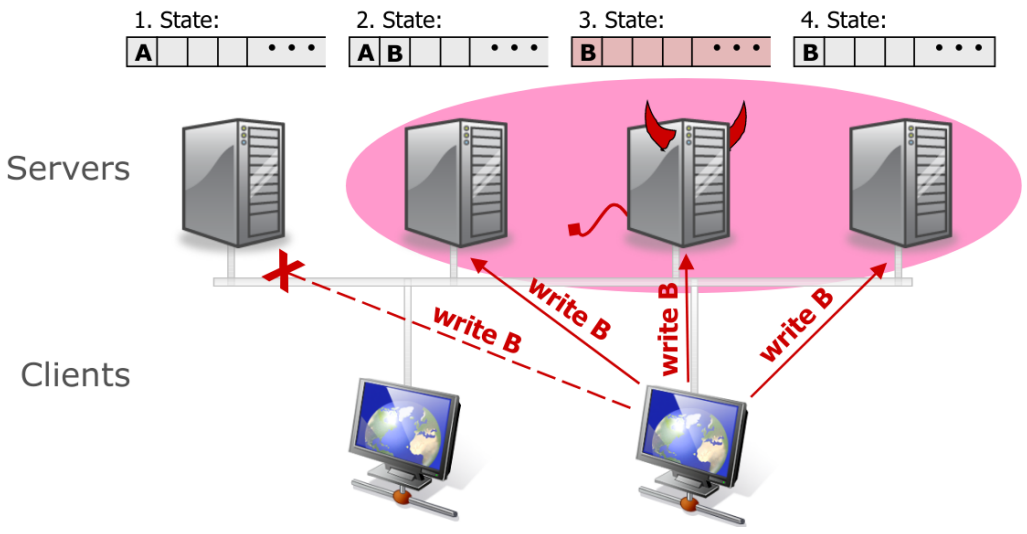
PBFT: Возможные проблемы

- Может отказать главная реплика
  - игнорирует запросы, назначает один номер разным запросам, пропускает номера...
  - резервные реплики отслеживают поведение главной и могут инициировать её смену
- Может отказать резервная реплика
  - некорректно сохраняет и применяет запросы, переданные корректной главной репликой
  - для защиты от таких отказов используются кворумы
- Реплика может отправить некорректный ответ клиенту
  - клиент ожидает получения $f + 1$ идентичных ответов от разных реплик

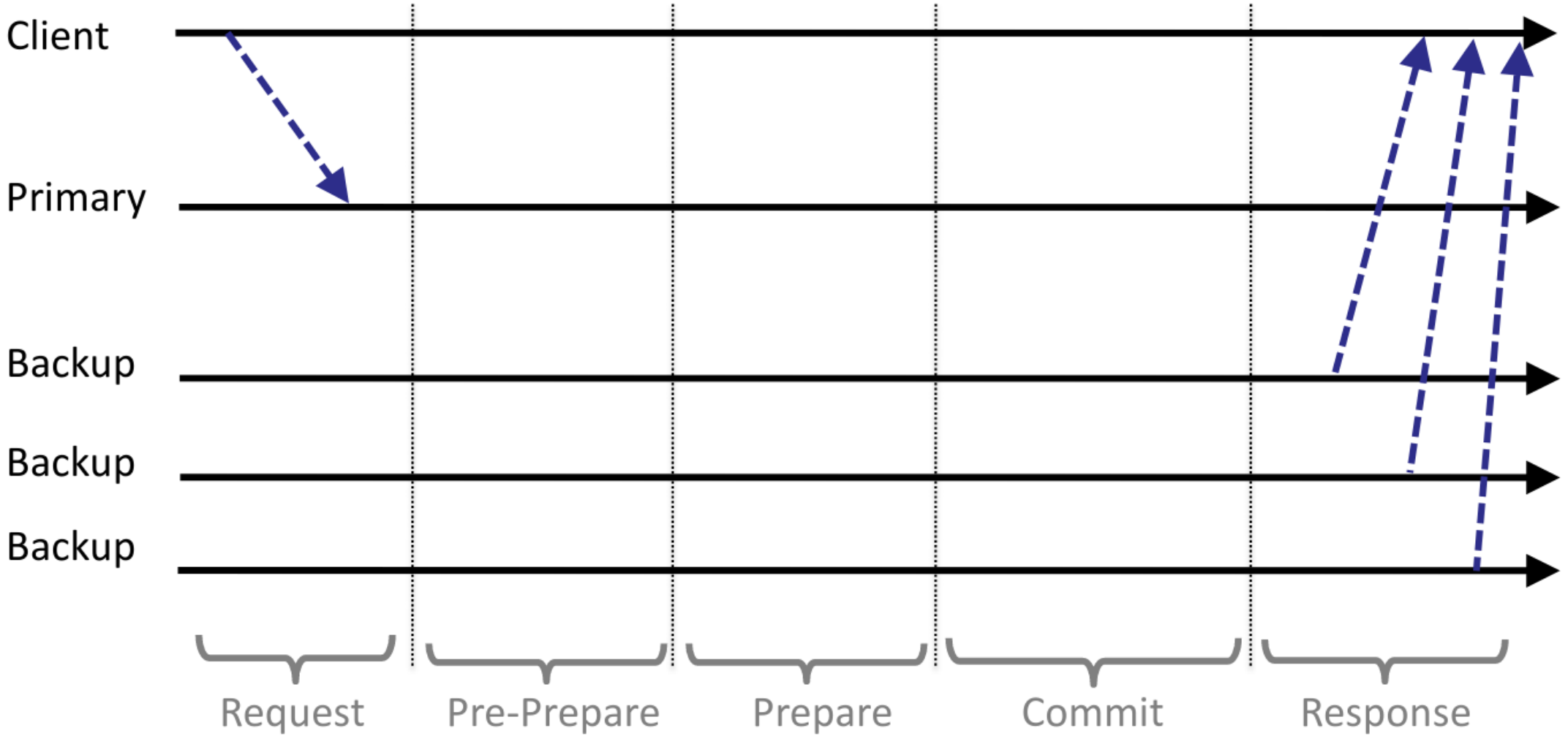
PBFT: Основная идея

- Шаги протокола координируются с помощью т.н. **сертификатов**
  - Набор подписанных сообщений от кворума реплик, подтверждающих выполнение некоторого свойства
- Кворумы имеют размер $N - f = 2f + 1$
  - пересечение любых двух кворумов содержит хотя бы одну корректную реплику

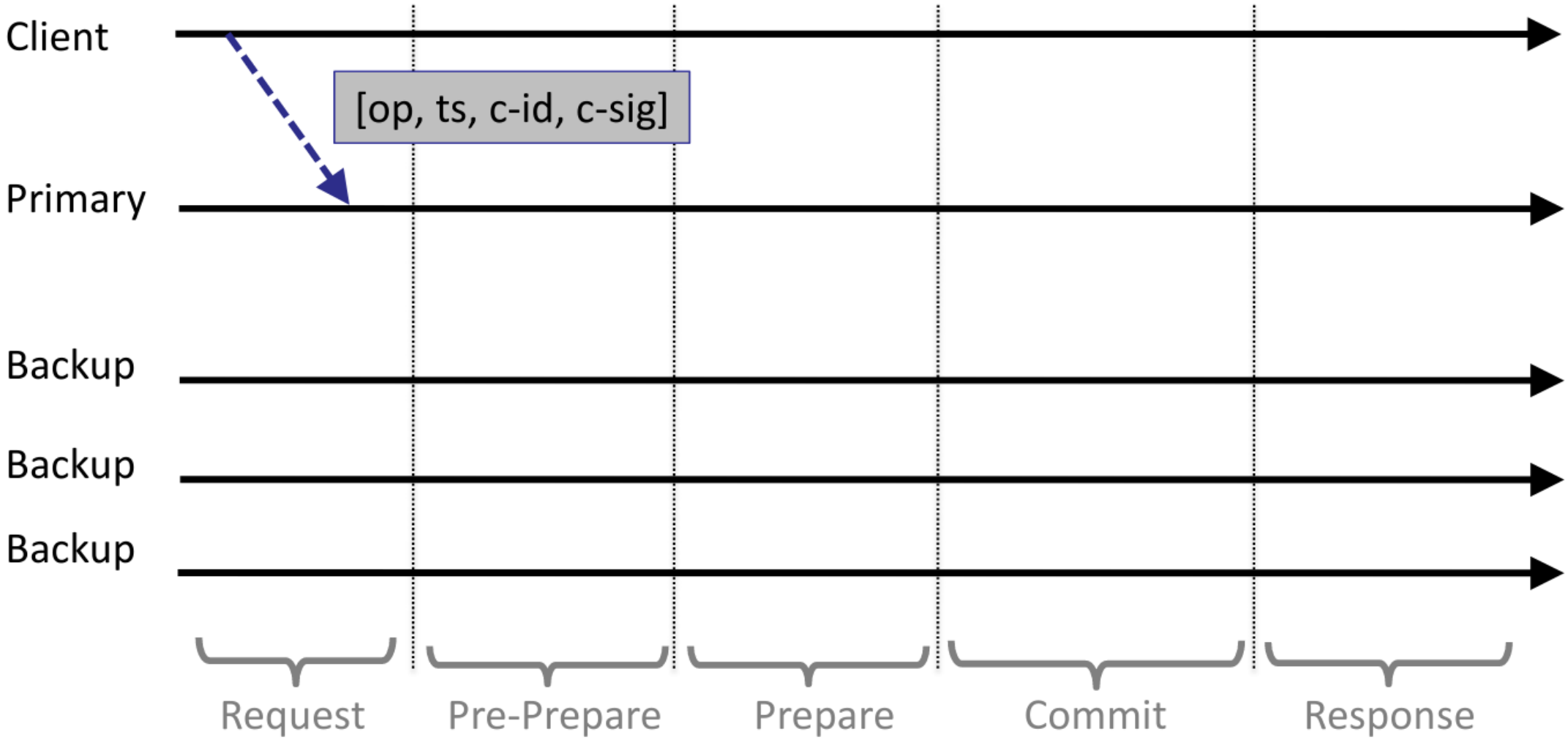




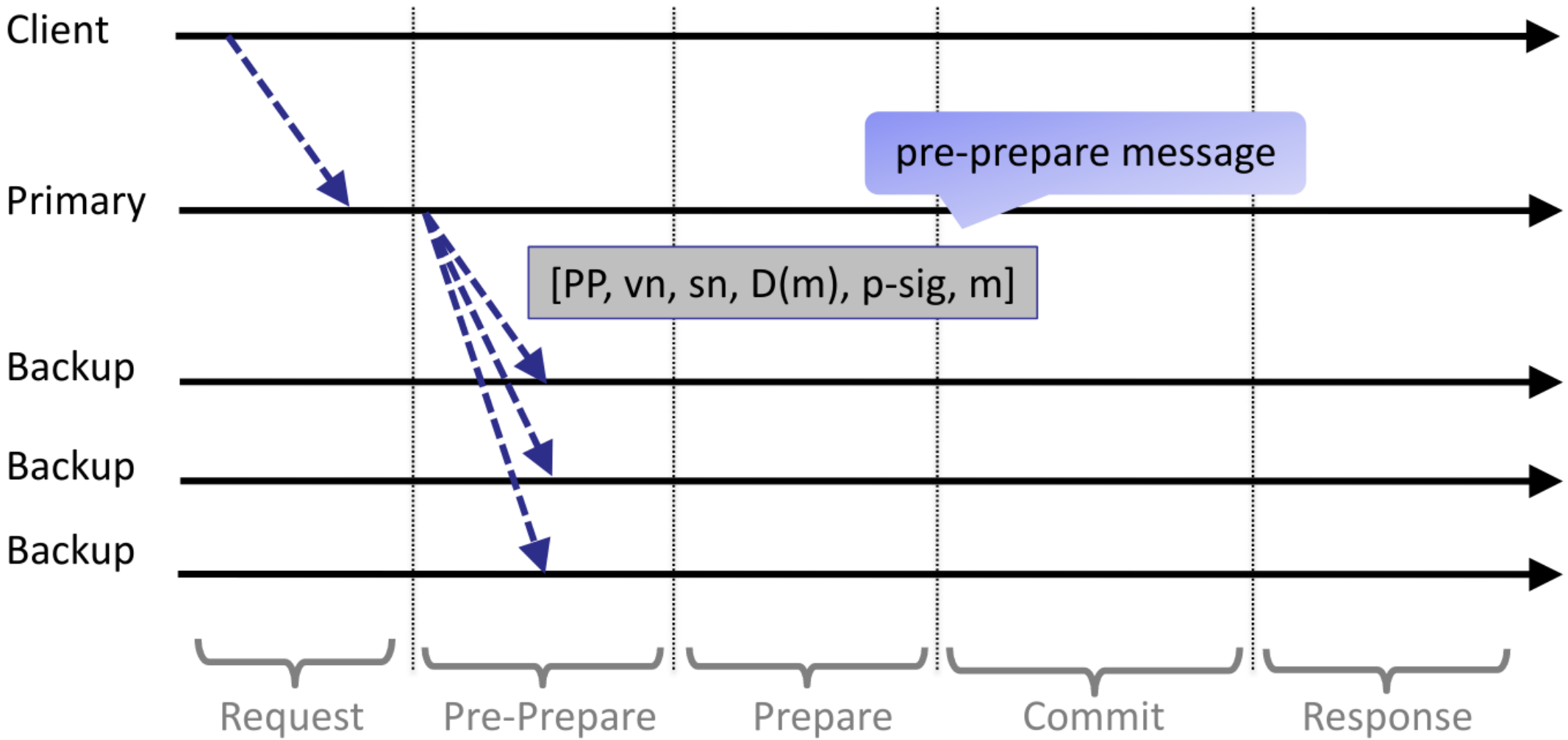
PBFT: Протокол



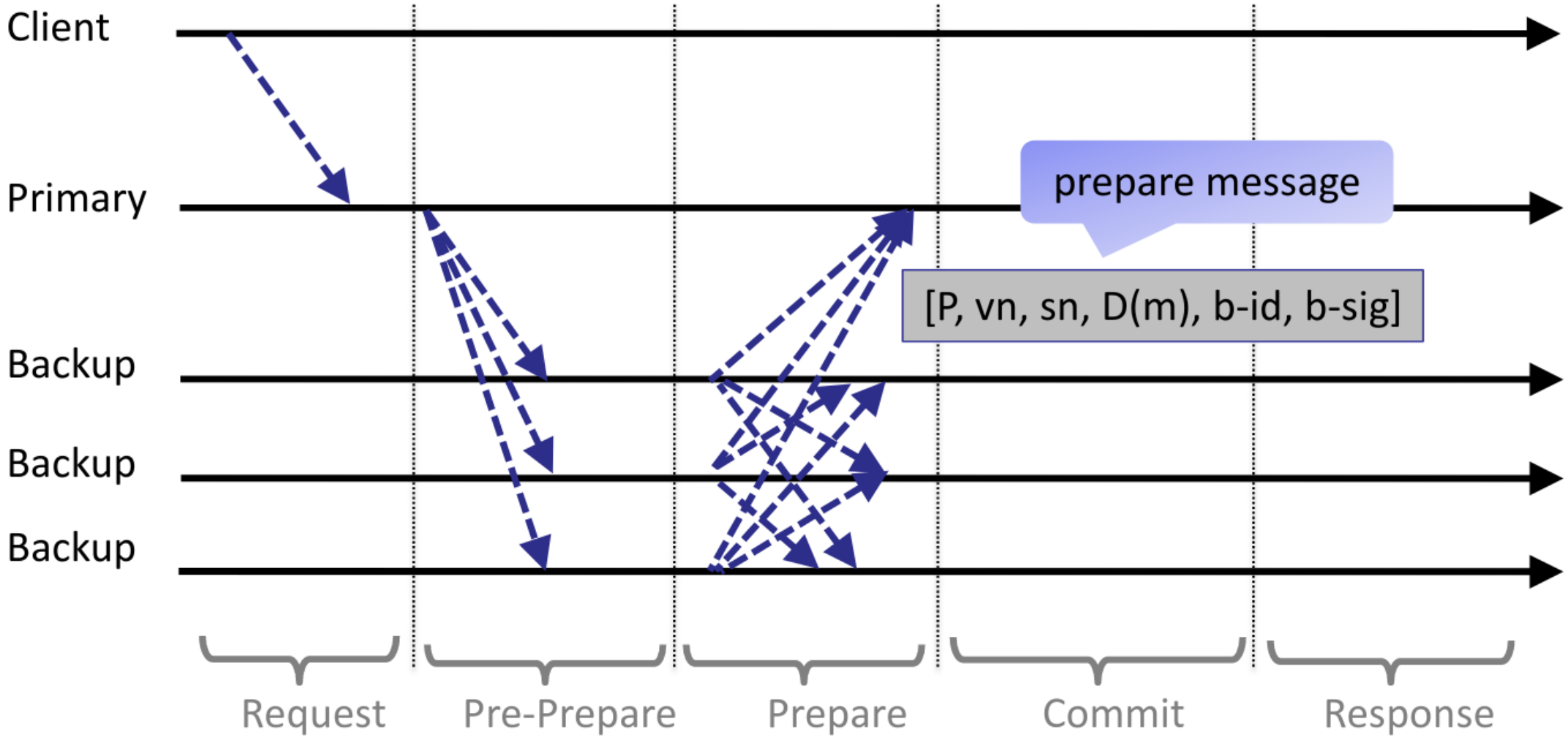
PBFT: Шаг Request



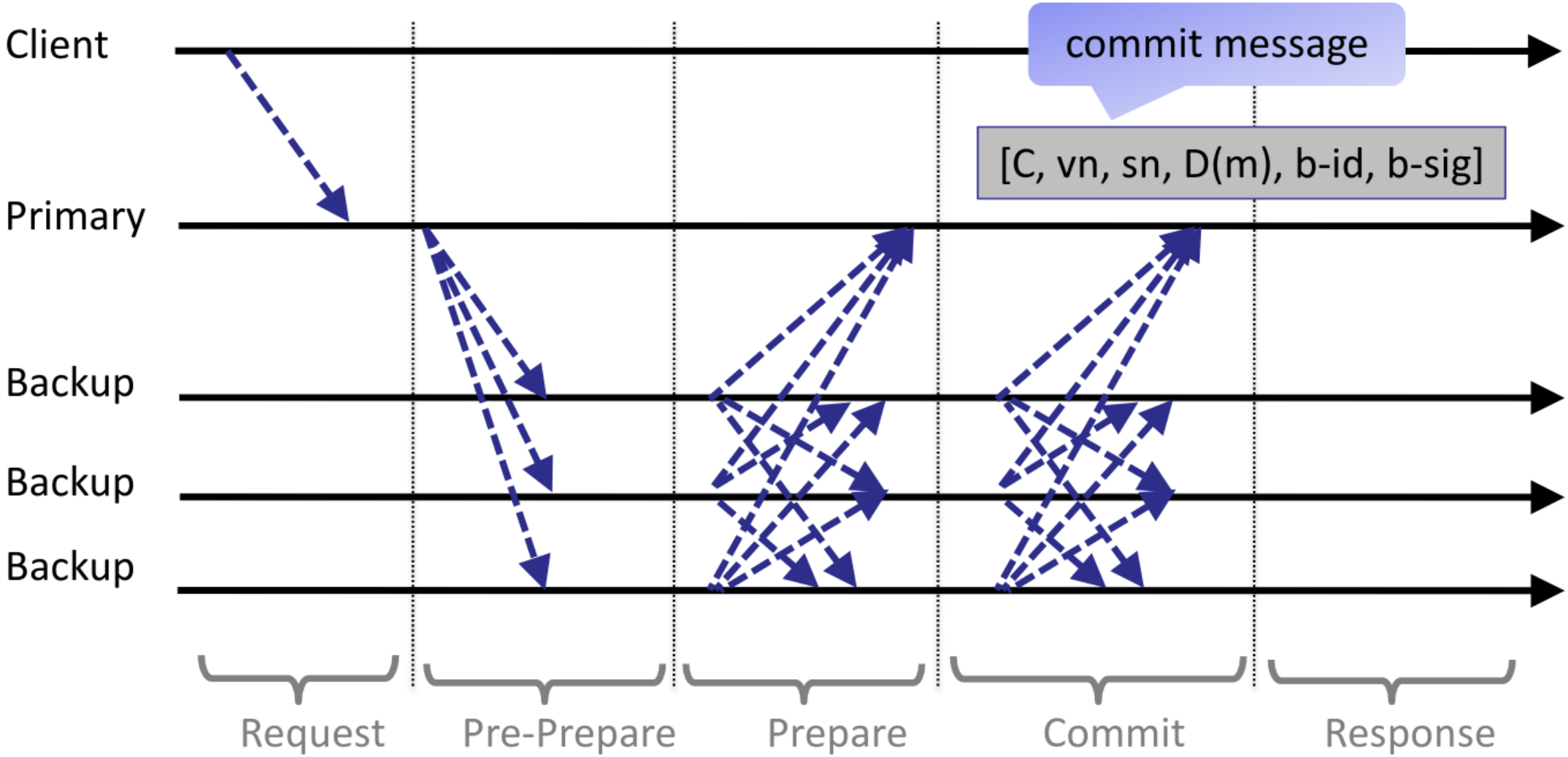
PBFT: Шаг Pre-Prepare



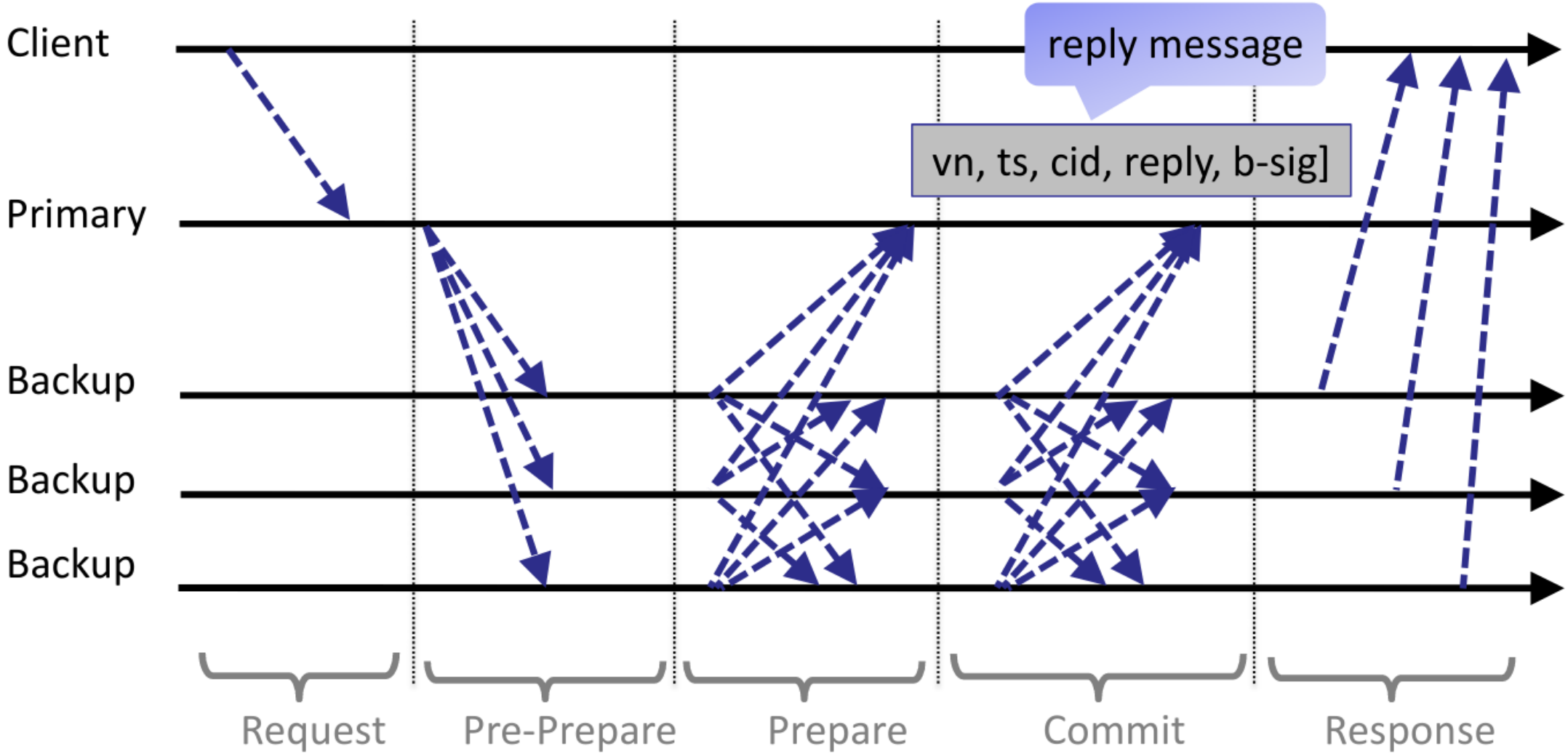
PBFT: Шаг Prepare



PBFT: Шаг Commit



PBFT: Шаг Response



Обнаружение отказа главной реплики

- Клиент не дожидается ответа на свой запрос
  - Или запрос клиента потерялся или главная реплика отказала
- После таймаута клиент рассылает запрос **всем** репликам
  - Реплика, которая уже закоммитила результат, отправляет его
  - Реплика, которая не получала *pre-prepare* от главной реплики, направляет запрос ей и ждёт прихода *pre-prepare*
  - Если реплика не получит *pre-prepare*, то главная реплика считается отказавшей
- В случае обнаружения отказа главной реплики, резервная инициирует изменение *view*
  - Номер *view* увеличивается на 1, определяется новая главная реплика
  - Реплика отправляет всем сообщение *view change*
  - Новая главная реплика вступает в права после получения $2f$ сообщений от других реплик

PBFT: Детали

- Смена view
- Сборка мусора
- Восстановление отказавшей реплики
- Оптимизации

См. статью

PBFT: Свойства

- Операции упорядочены с помощью номеров *view* и *sequence number*
- Если корректная реплика закоммитила $[m, vn, sn]$, то никакая другая корректная реплика не может закоммитить $[m', vn, sn]$, где $m \neq m'$ и $D(m) \neq D(m')$
- Если клиент получил результат, то никакая корректная реплика не закоммитит другой результат
- Протокол в конце концов завершается (клиент получает результат)

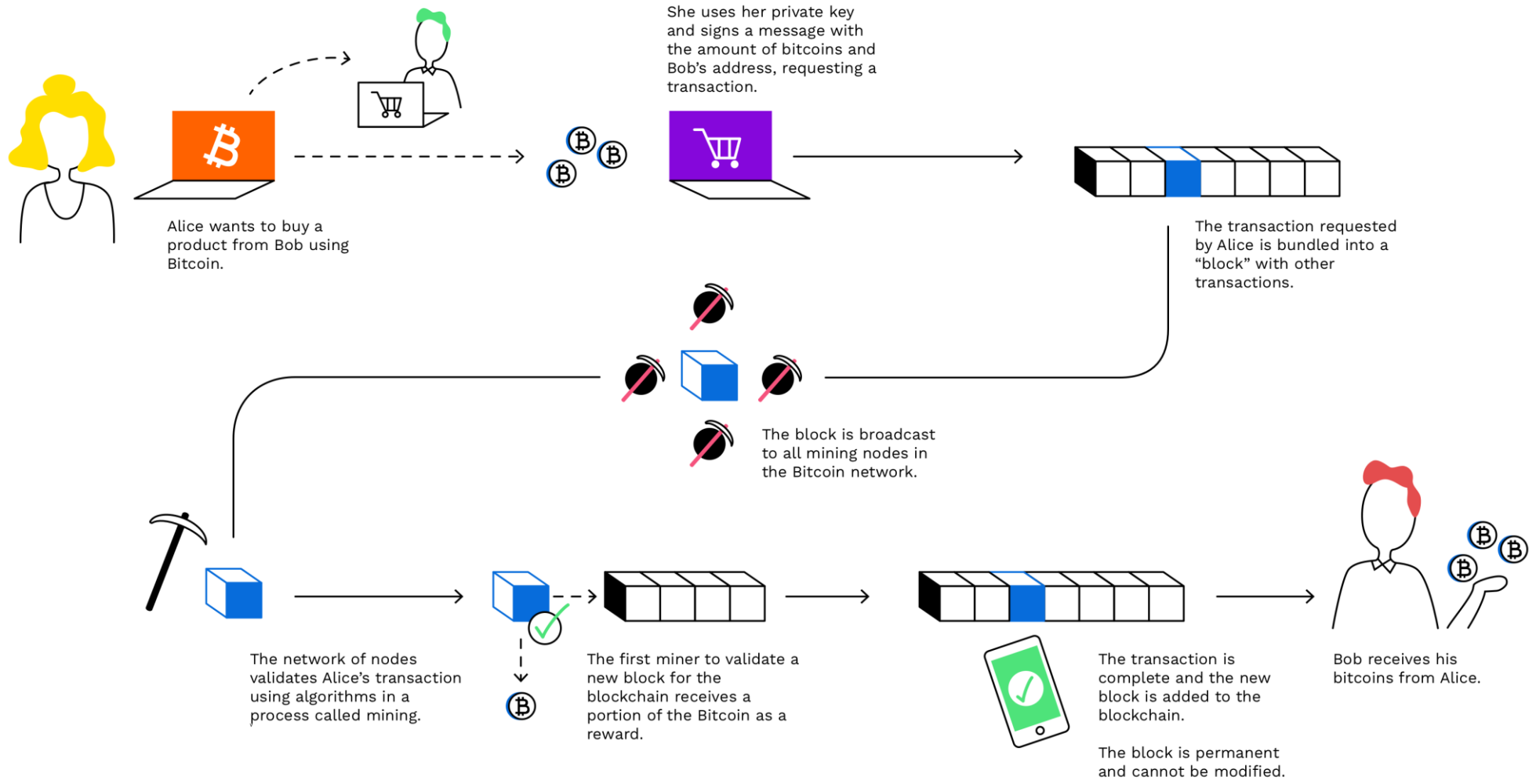
PBFT: Недостатки

- Отказавшая главная реплика может замедлить работу протокола
  - Игнорирует изначальный запрос клиента
  - Отправляет *pre-prepare* только в ответ на повторный приход запроса от других узлов
- Ограниченная масштабируемость
  - Требуется $O(N^2)$ сообщений и взаимодействие каждый с каждым
- Не подходит для систем с открытым участием без изначального доверия
  - Самостоятельное подключение новых узлов
  - Защита от атаки Сивиллы (Sybil attack)

PBFT: Применение и развитие

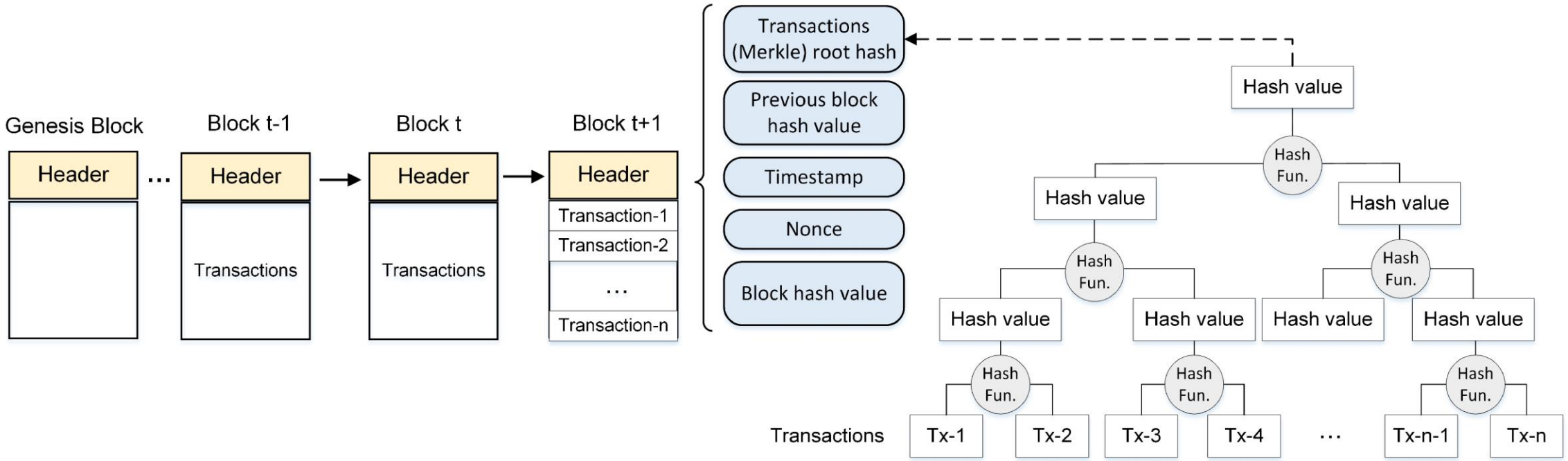
- Permissioned (private) blockchains
  - Hyperledger Sawtooth
- Другие протоколы
  - Zyzzyva, Tendermint, HotStuff, LibraBFT...

Bitcoin



Nakamoto S. Bitcoin: A peer-to-peer electronic cash system (2008)

Цепочка блоков (blockchain)



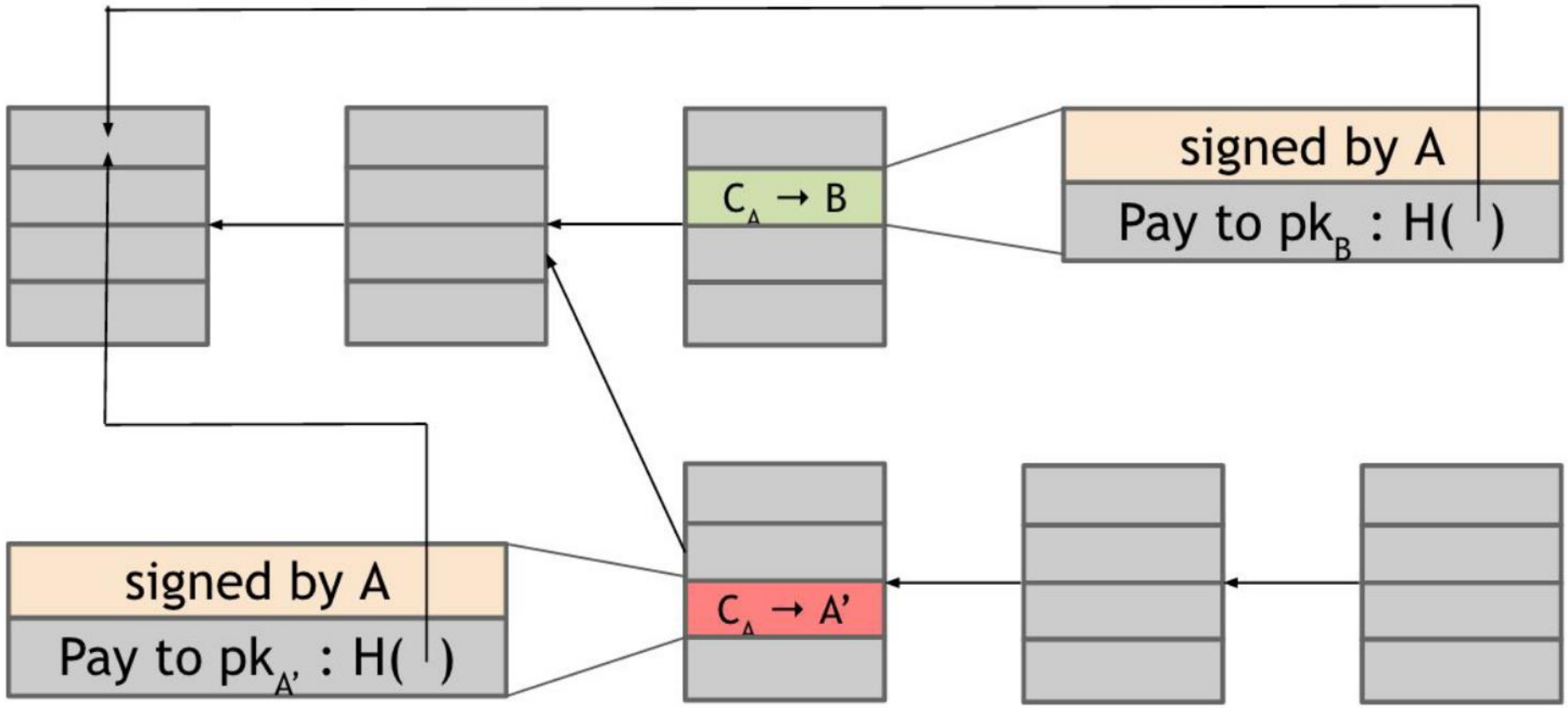
Связь с SMR и консенсусом

- Репликация автомата (State Machine Replication)
  - Состояние – набор балансов счётов, блокчейн
  - Операции – транзакции между счётами, добавление блока
- Консенсус
  - Какие транзакции валидные и в каком порядке они выполнены
  - Какие блоки и в каком порядке включены в блокчейн
- Особенности
  - Децентрализованная система, нет регулирующего органа (authority)
  - Публичная система, каждый может подключиться, причем анонимно
  - Не все участники подключены друг к другу
  - Отсутствие взаимного доверия между участниками

Консенсус в Bitcoin

- Особенности
  - Рандомизация и вероятностные гарантии
  - Мотивация участников действовать честно путем вознаграждений в криптовалюте
- Базовый алгоритм
  - Новые транзакции рассылаются по всем узлам
  - Каждый узел собирает транзакции в новый блок
  - В каждом раунде выбирается **случайный** узел, который рассылает всем свой новый блок
  - Остальные узлы принимают блок, если все транзакции в нём валидны
  - Узлы подтверждают принятие блока путем включения его хеша в следующий создаваемый ими блок

Двойное расходование



Мотивация участников

- Наказать узел за то, что он создал "плохой" блок?
  - Нет, участники анонимны
- Поощрить узел за то, что он создал блок, попавший в стабильную ветвь блокчейна?
  - Да, у нас уже есть криптовалюта и её можно распределять анонимно
- Block reward
  - Создатель блока включает в него транзакцию с вознаграждением
  - Вознаграждение будет получено только, если блок попадет в стабильную ветвь
  - Сумма вознаграждения фиксированная и периодически уменьшается
- Transaction fee
  - Создатель транзакции может включить в неё "чаевые" за обработку

Как выбрать тот случайный узел?

- Требования
  - Создание блока не должно быть легким
  - Надо защититься от атаки Сивиллы
- Proof-of-work
  - Выбор узла пропорционально доступному его *владельцу* ресурсу (вычислительной мощности)
  - Для создания блока узел должен решить **hash puzzle** – найти значение $nonce$, такое что

$$H(nonce||prev\_hash||tx_1||tx_2||\ ...\ ||tx_i) < target$$

  - Значение $target$ динамически подбирается так, чтобы среднее время между созданиями блоков было 10 минут
  - Поиск решения hash puzzle вычислительно сложен, а проверка – нет

Материалы

- Distributed Systems: Principles and Paradigms (разделы 8.2.5, 2.5.3, 5.4.5, 8.2.6, 9.4.2, 9.4.3)
- Distributed Systems: Concepts and Design (раздел 15.5)
- Bitcoin and Cryptocurrency Technologies (главы 1-2)
- Byzantine Fault Tolerance, from Theory to Reality
- Bitcoin's Academic Pedigree
- The Saddest Moment