Устойчивость к произвольными отказами

Сухорослов Олег Викторович

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

19.12.2020

Произвольные (византийские) отказы

- Любое отклонение поведения узлов PC от заданного протоколом
  - Наиболее общая модель отказов, включающая ранее рассмотренные
  - Процесс может быть атктивен, но при этом работать некорректно
- Намеренные атаки
  - Пропуск, вставка, изменение, повтор собщений, 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)

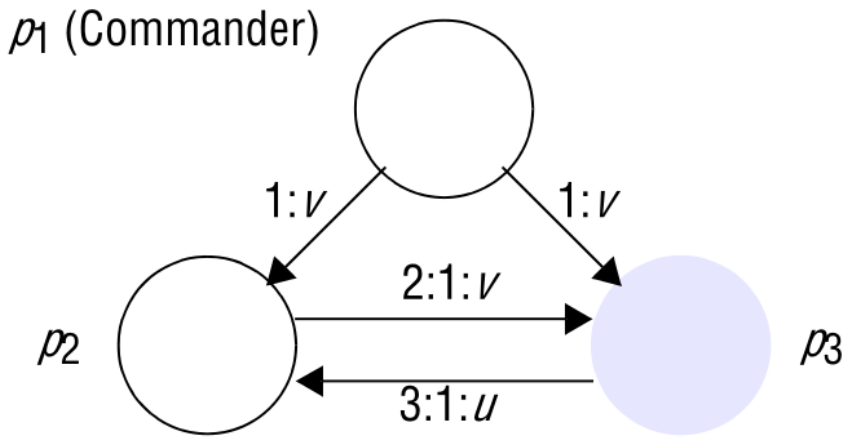
Византийские генералы

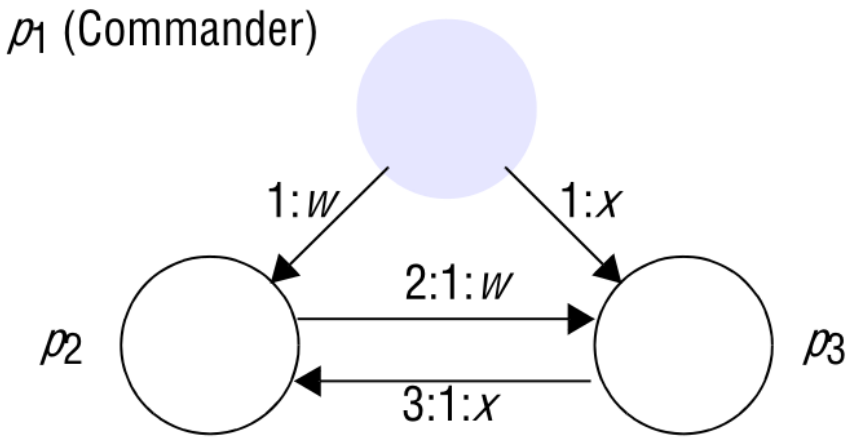
Lamport L., Shostak R., Pease M. The Byzantine Generals Problem (1982)

Задача с командиром

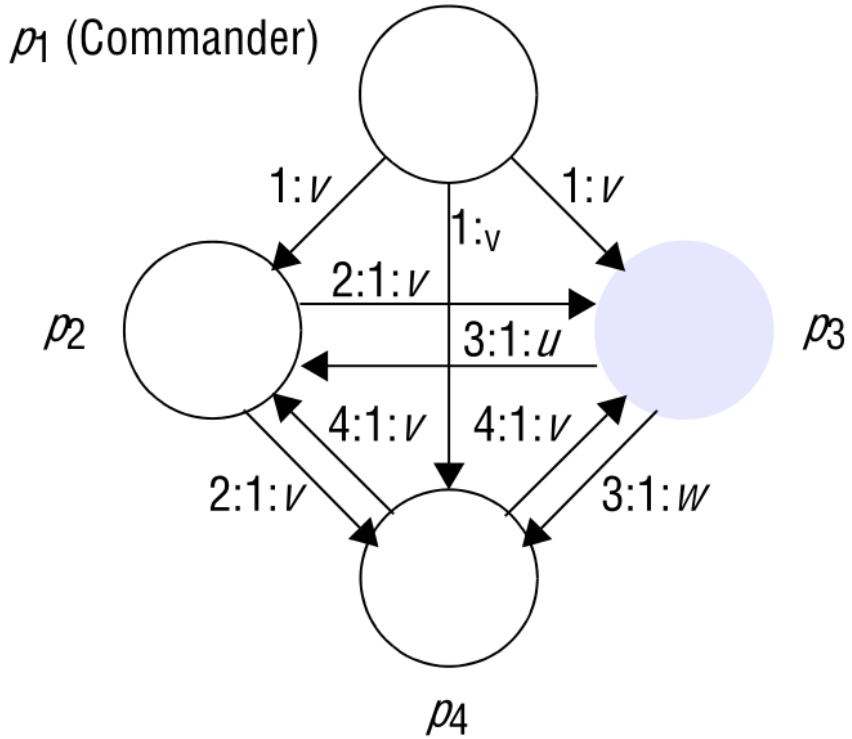
- Главный генерал (командир) рассылает свой приказ (наступать, отступать) остальным генералам (лейтенантам)
- Требования
  - Все верные лейтенанты должны выполнить одинаковый приказ
  - Если командир не был предателем, то каждый верный лейтенант выполнит его приказ
- Предположения
  - Сообщения доставляются 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?

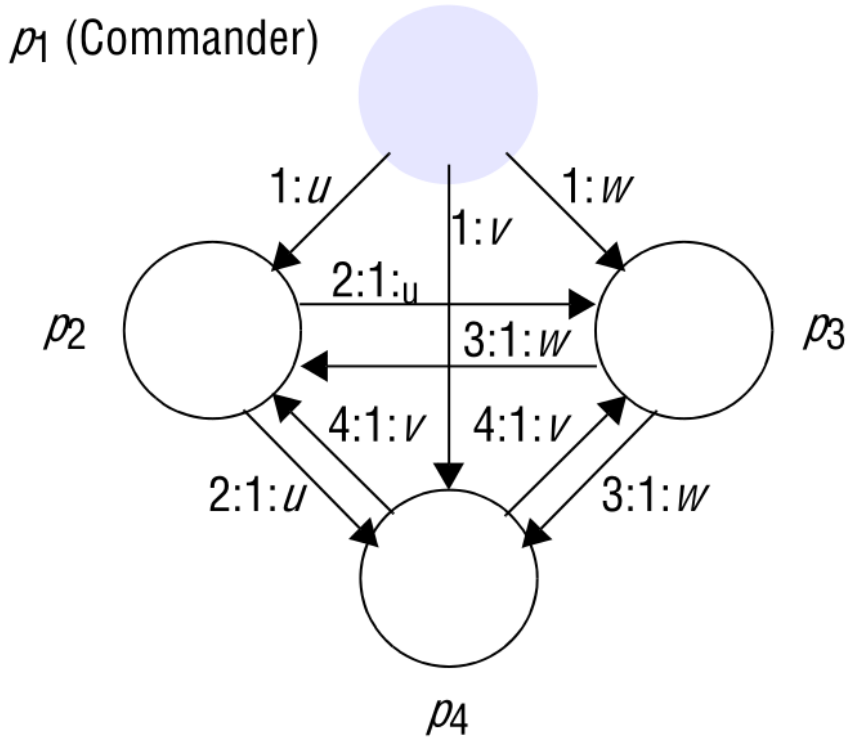
Три генерала (N=3, f=1)





Четыре генерала (N=4, f=1)



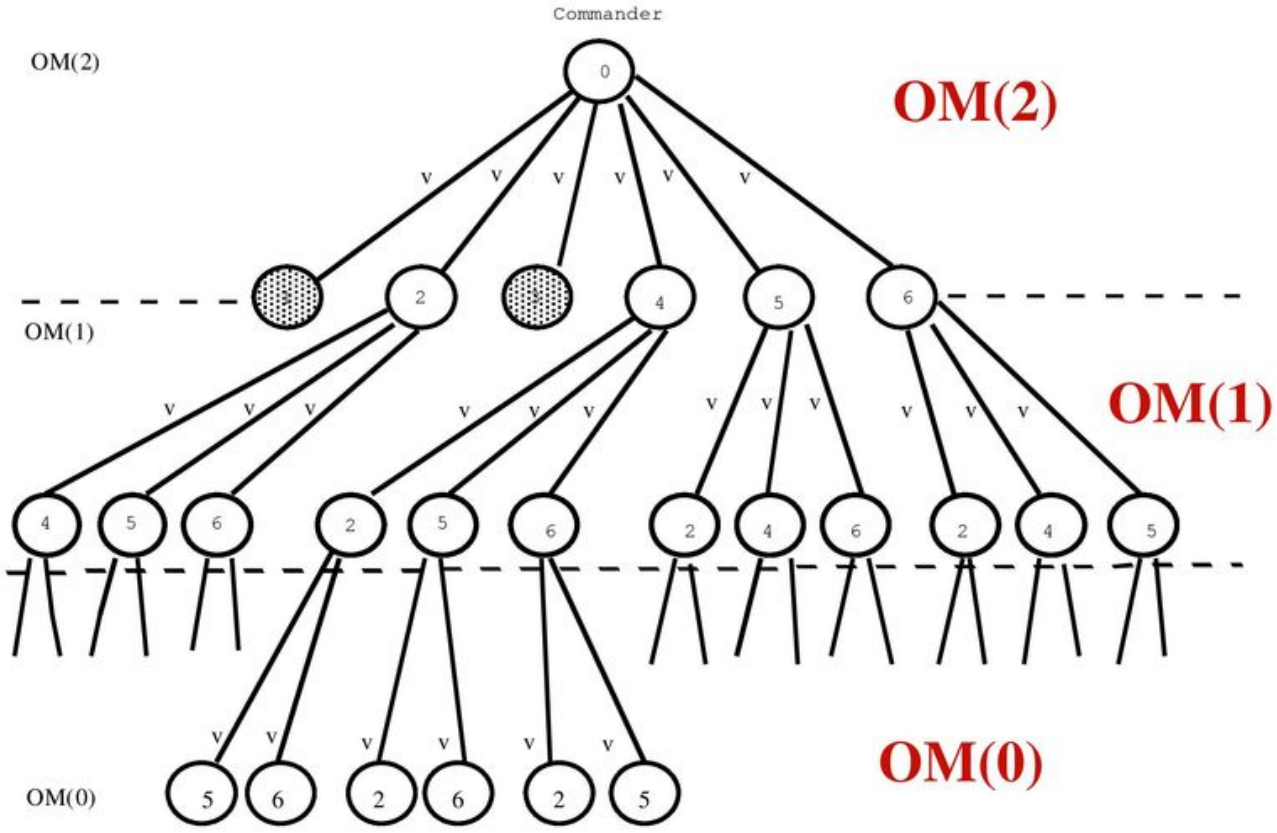


Oral Messages Algorithm

- Algorithm $OM(0)$
  1. The commander sends its value to every lieutenant
  2. Each lieutenant uses the value he received from commander, or uses RETREAT if he received no value
- Algorithm $OM(m)$
  1. The commander sends its value to every lieutenant
  2. $\forall i$, let v_i be the value lieutenant i receives from the commander, or RETREAT if it has received no value. Lieutenant i acts as the commander of algorithm $OM(m-1)$ to send the value v_i to each of the other $n-2$ other lieutenants
  3. $\forall j \neq i$, let v_j be the value received by i from j in Step 2 of algorithm $OM(m-1)$ or RETREAT if no value. Lieutenant i uses the value $\mathsf{majority}(v_1, ..., v_{n-1})$ (deterministic function)

m - число предателей

Семь генералов (N=7, f=2)

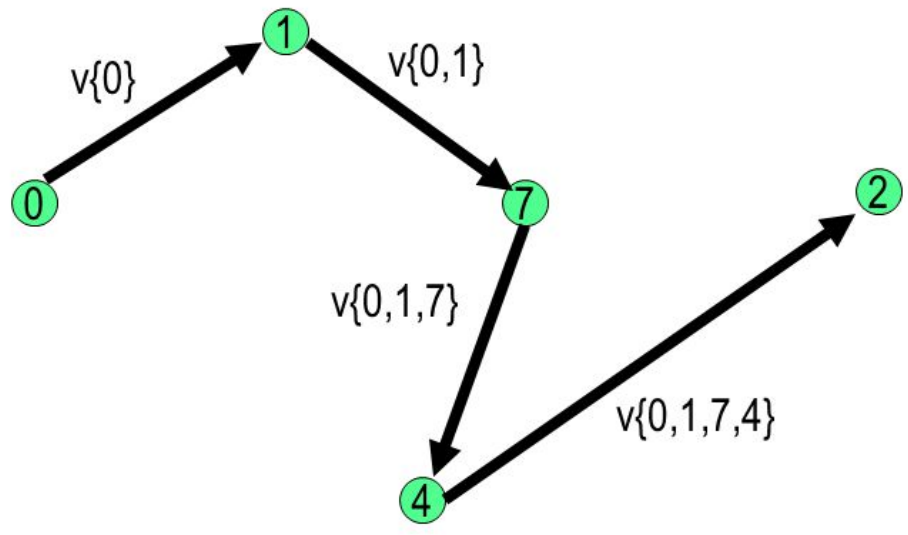


Свойства алгоритма

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

Подписанные сообщения

- Сообщения подписаны отправителем
- Подпись верного генерала не может быть подделана
- Каждый генерал может проверить подпись и аутентичность 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 $v\{0 : j_1 : \ldots : j_k\}$ и если v нет в V_i
    - добавляет v в V_i
    - если $k < f$, то отправляет сообщение $v\{0 : j_1 : \ldots : j_k : i\}$ лейтенантам, которые еще не подписали сообщение
  - когда сообщений больше не ожидается, вычисляет финальный приказ с помощью функции $choice(V_i)$
- Сообщения, не прошедшие проверку с помощью 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 узлов предлагают значения
- Требуемые свойства
  - Никакие два корректных узла не должны принять разные значения
  - Никакой узел не принимает значение дважды
  - Если узел принял значение, то оно было предложено некоторым корректным узлом
  - Каждый корректный узел в конце концов принимает значение
- Традиционные алгоритмы консенсуса (Paxos, Raft)
  - не поддерживают произвольные отказы
  - требуют не менее $2f + 1$ узлов

Practical Byzantine Fault Tolerance (PBFT)

Castro M., Liskov B. Practical Byzantine Fault Tolerance (1999)

- Протокол для state machine replication
  - Реплицированный сервис с состоянием и детерминированными операциями над ним
  - Клиенты отправляют запросы с операциями и ожидают ответов
- Предположения
  - Частично синхронная модель, ненадежные каналы (см. традиционные алгоритмы)
  - Сообщения могут быть проверены на подлинность с помощью цифровых подписей
  - f узлов могут быть подвержены независимым произвольным отказам
- Требует $3f + 1$ узлов

PBFT: Оптимальность

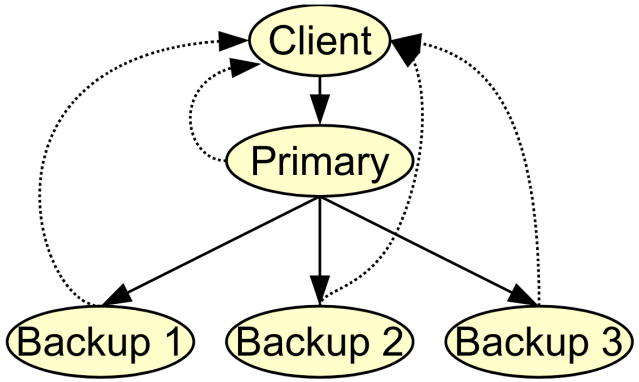
- Для ответа на запрос клиента должно быть достаточно получить ответы от $N - f$ реплик, так как f реплик могут не ответить
- Но f реплик, от которых не пришли ответы, могут просто работать медленно, и среди ответивших могут быть f отказавших реплик
- Число ответов от корректных реплик $N - 2f$ должно превышать f, откуда получаем $N > 3f$

PBFT: Конфигурация системы

- Каждый узел (реплика) имеет идентификатор *i* от 0 до *N-1*
- Конфигурация системы называется *view*
  - аналог *term* в Raft
- Внутри *view* один из узлов является *главным (primary)*
  - принимает запросы клиентов
  - присваивает им номера (sequence number) и рассылает по остальным узлам
  - определяется однозначным образом, например $i = v \bmod N$
- Остальные узлы играют роль *резервных (backup)*
  - принимают запросы от главного узла
  - участвуют в процессе принятия решения о коммите операции
- При отказе главного узла происходит смена *view*

PBFT: Базовая схема работы

- Клиент отправляет *подписанный* запрос главному узлу
- Главный узел рассылает запрос резервным узлам
- С помощью кворумов достигается консенсус относительно порядка операций и их коммита
- Как только консенсус достигнут, узлы выполняют запрос и отправляют ответ напрямую клиенту
- Клиент ожидает получения $f + 1$ идентичных ответов
- Если ответы не приходят, то клиент рассылает запрос повторно *всем* узлам

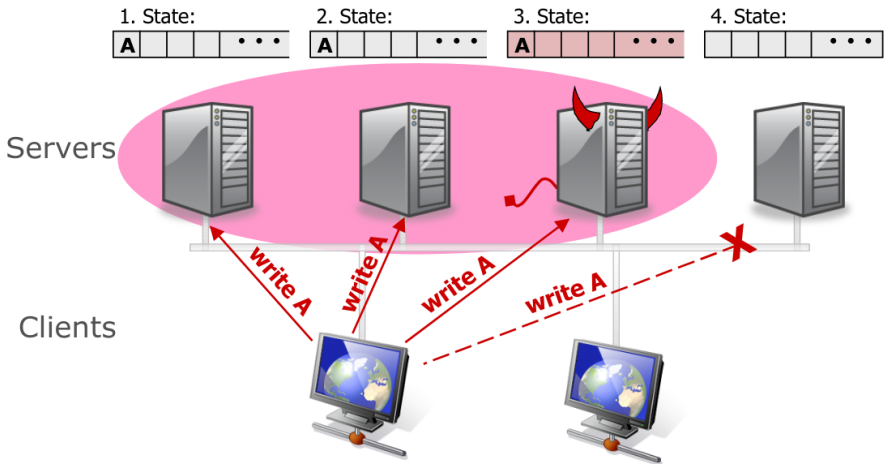


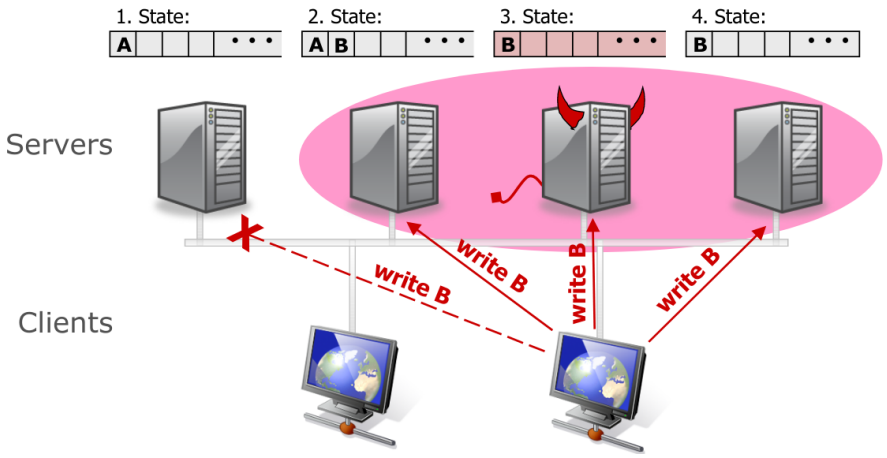
PBFT: Проблемы и меры

- Может отказать главный узел
  - игнорирует запросы, назначает один номер разным запросам, пропускает номера...
  - резервные узлы отслеживают поведение главного и могут инициировать его смену
- Может отказать резервный узел
  - некорректно сохраняет и применяет операции, переданные корректным главным узлом
  - для защиты от таких отказов используются кворумы
- Узел может отправить некорректный ответ клиенту
  - клиент ожидает получения $f + 1$ идентичных ответов

PBFT: Основная идея

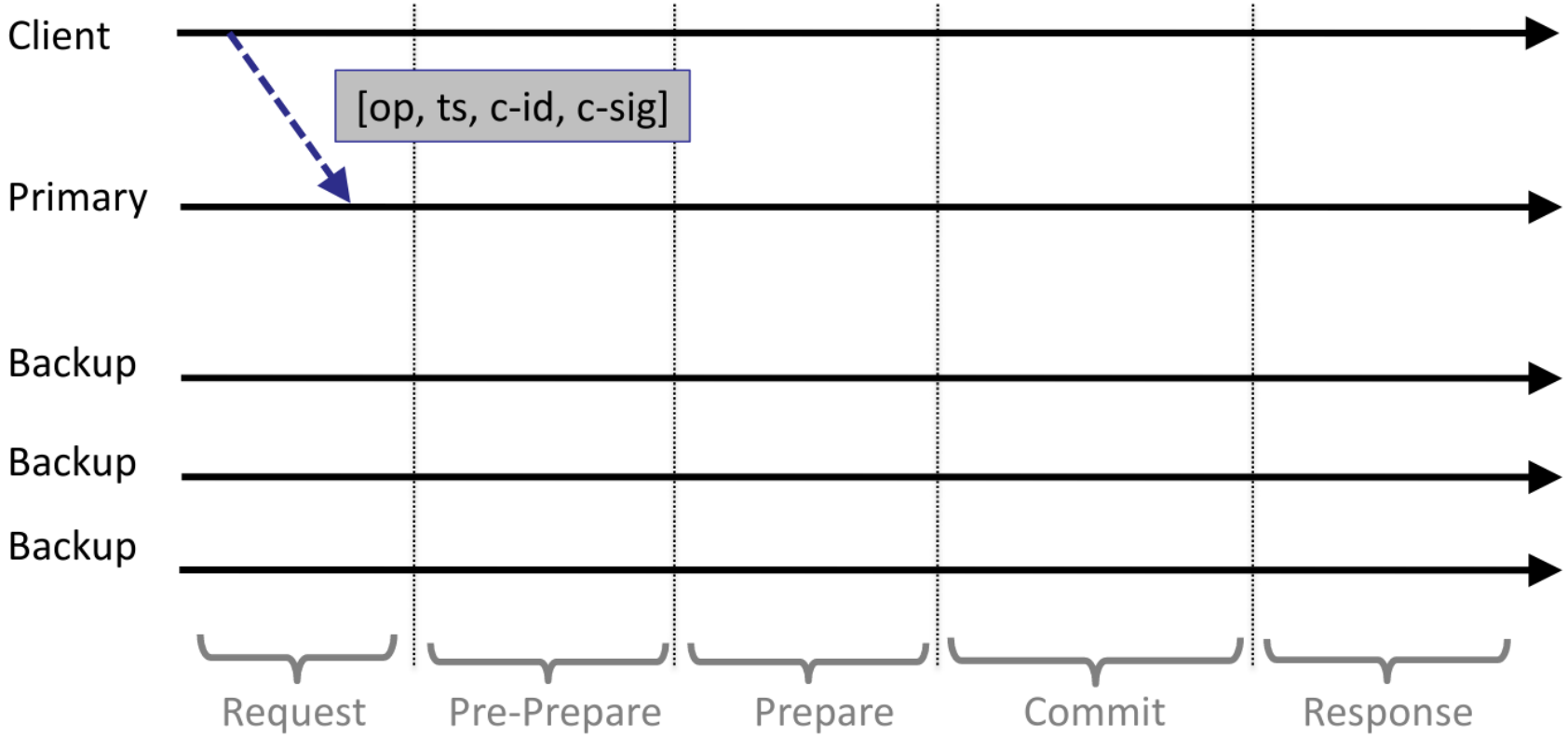
- Шаги протокола координируются с помощью т.н. *сертификатов*
  - Набор подписанных сообщений от кворума узлов, подтверждающих выполнение нек. свойства
- Кворумы имеют размер $N - f = 2f + 1$
  - пересечение любых двух кворумов содержит по крайней мере один корректный узел



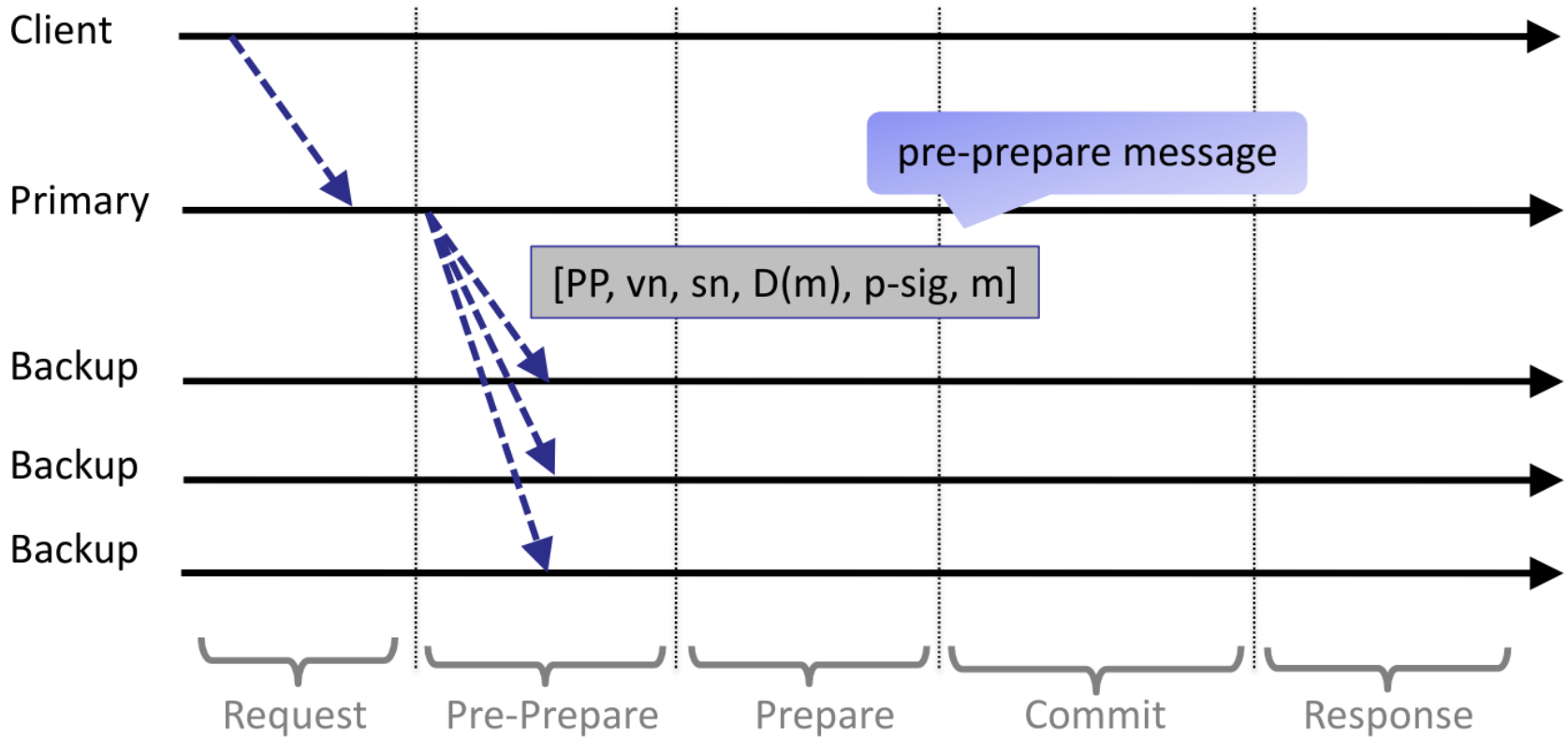


PBFT: Протокол

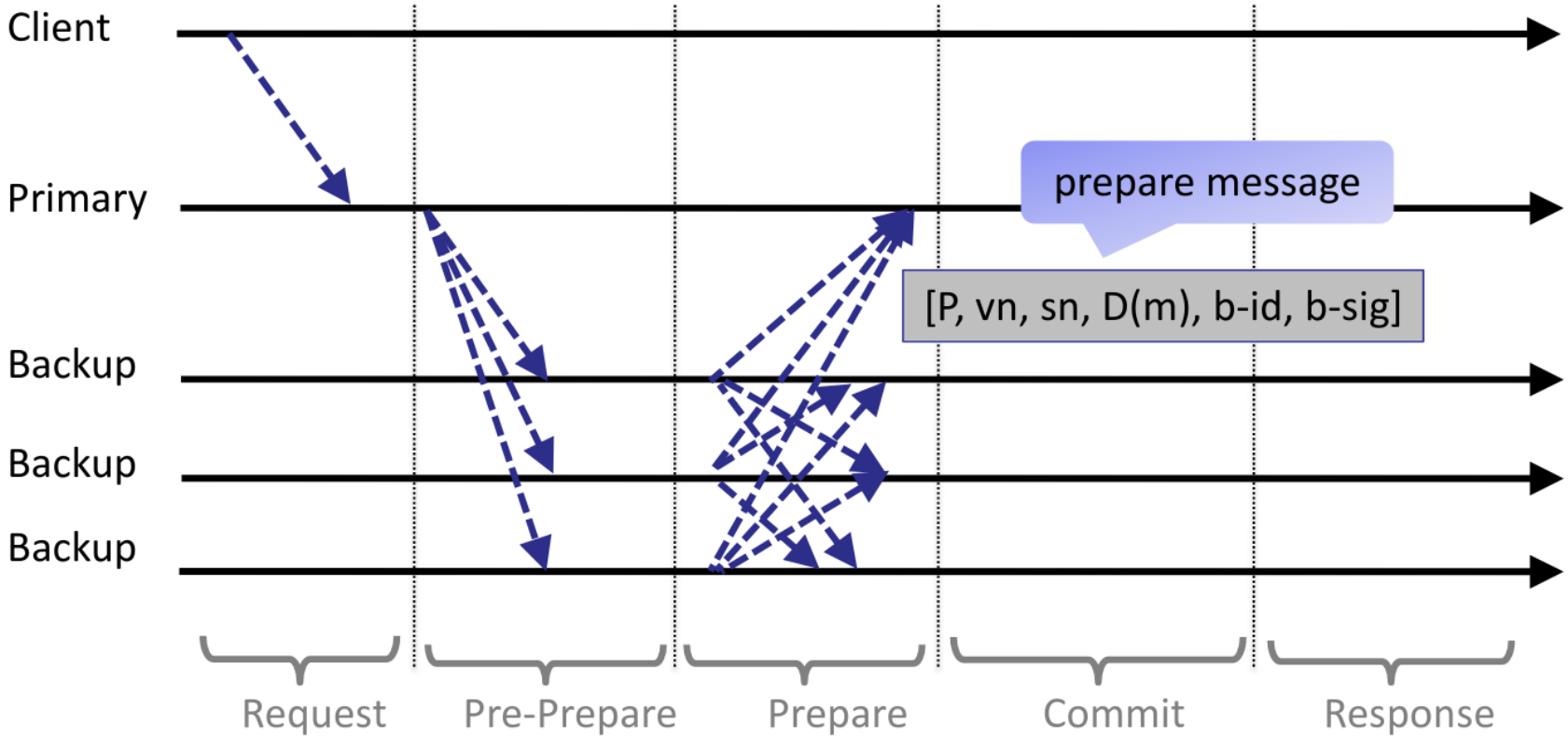
PBFT: Фаза Request



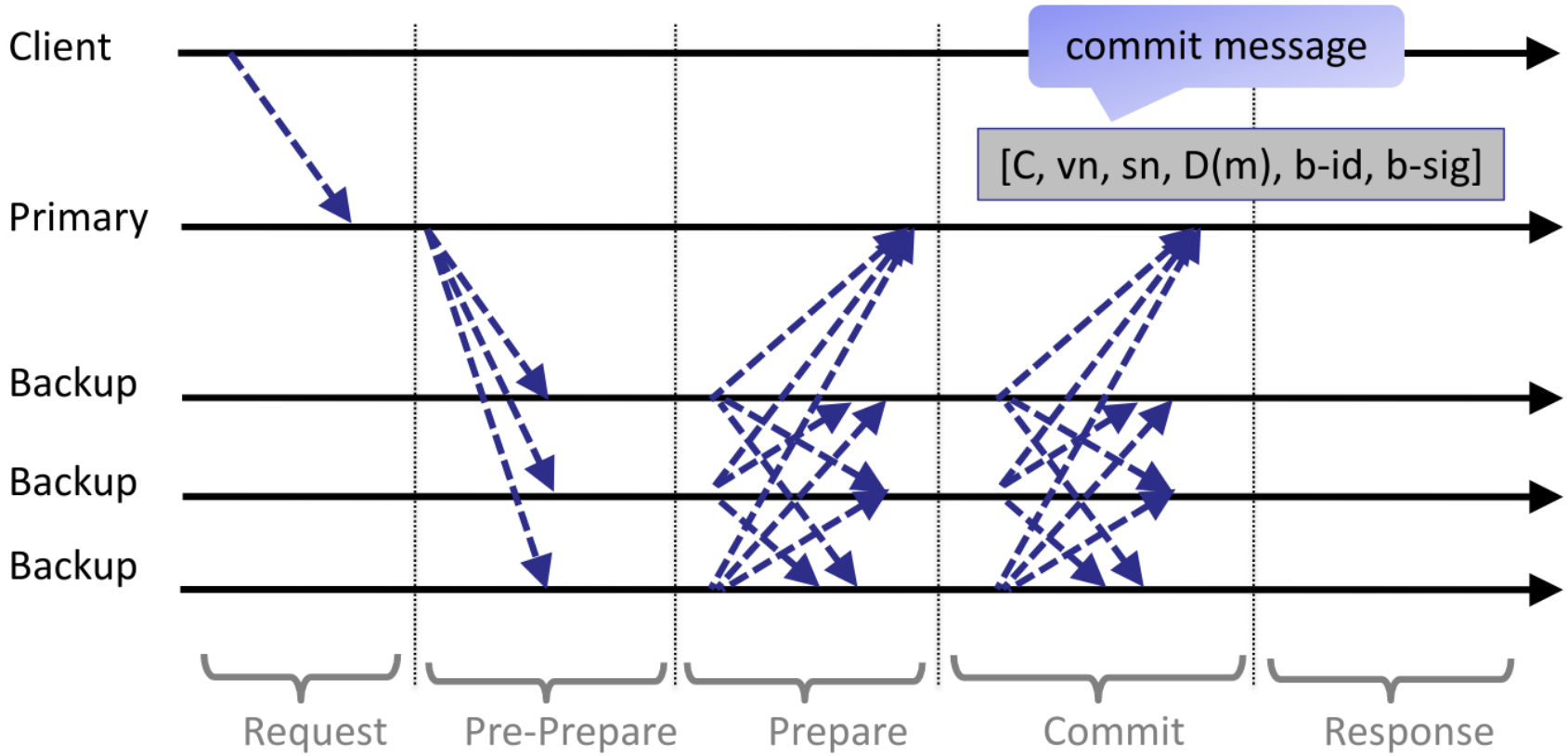
PBFT: Фаза Pre-Prepare



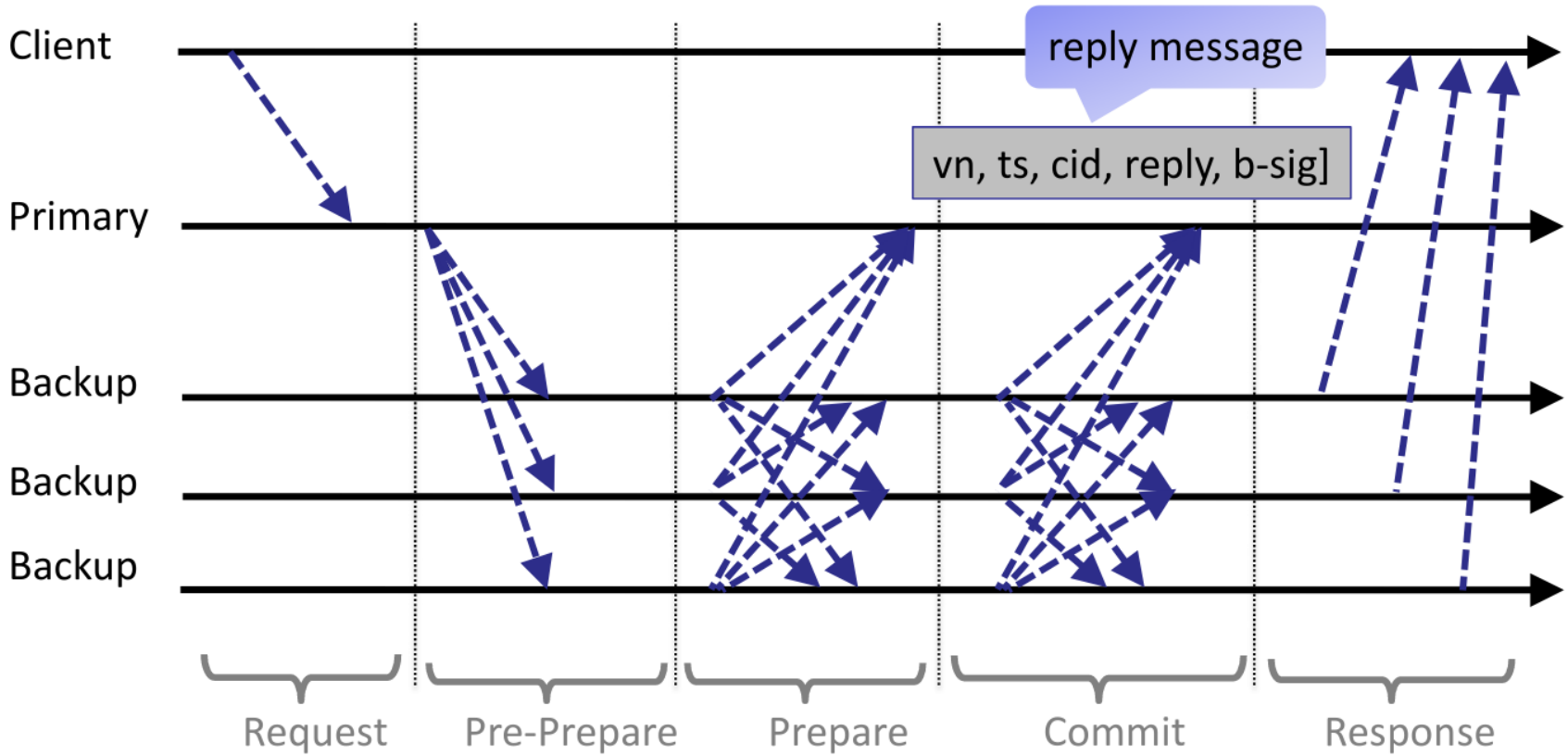
PBFT: Фаза Prepare



PBFT: Фаза Commit



PBFT: Фаза Response



Обнаружение отказа главного узла

- Клиент не дожидается ответа
  - Или запрос клиента потерялся или главный узел отказал
- После таймаута клиент рассылает запрос *всем* узлам
  - Узел, который уже закоммитил результат, отправляет его
  - Узел, который не получал *pre-prepare*, направляет запрос главному и ждёт *pre-prepare*
  - Если узел не получит *pre-prepare*, то он признает главного отказавшим
- В случае обнаружения отказа главного, узел инициирует изменение *view*
  - Номер *view* увеличивается на 1, определяется новый главный узел ($v \bmod N$)
  - Узел отправляет всем сообщение *view change*
  - Новый главный узел вступает в права после получения *2f* сообщений от других узлов

PBFT: Детали

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

PBFT: Свойства

- Операции упорядочены с помощью номеров *view* и *sequence number*
- Если корректный узел закоммитил *[m, vn, sn]*, то никакой другой корректный узел не может закоммитить *[m', vn, sn]*, где *m != m'* и *D(m) != D(m')*
- Если клиент получил результат, то никакой корректный узел не закоммитит другой результат
- Протокол в конце концов завершается (клиент получает результат)

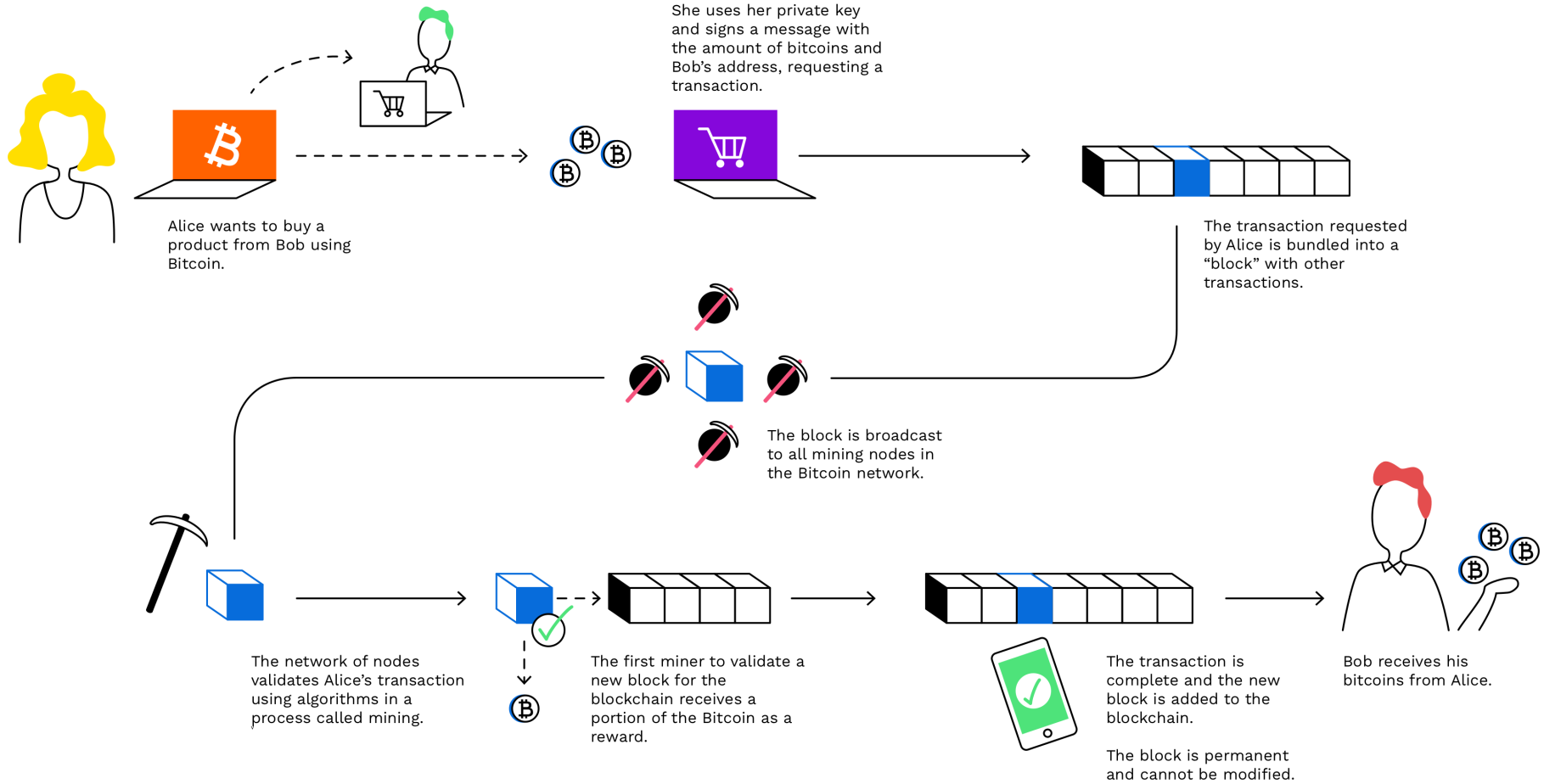
PBFT: Недостатки

- Отказавший главный узел может замедлить работу протокола
  - Игнорирует изначальный запрос клиента
  - Отправляет *pre-prepare* в ответ на повторный приход запроса от других узлов
- Низкая масштабируемость
  - Требуется $O(N^2)$ сообщений и full peering
- Не подходит для систем с открытым участием без изначального доверия
  - Самостоятельное подключение новых узлов
  - Защита от атаки Сивиллы (Sybil attack)

PBFT: Применение и развитие

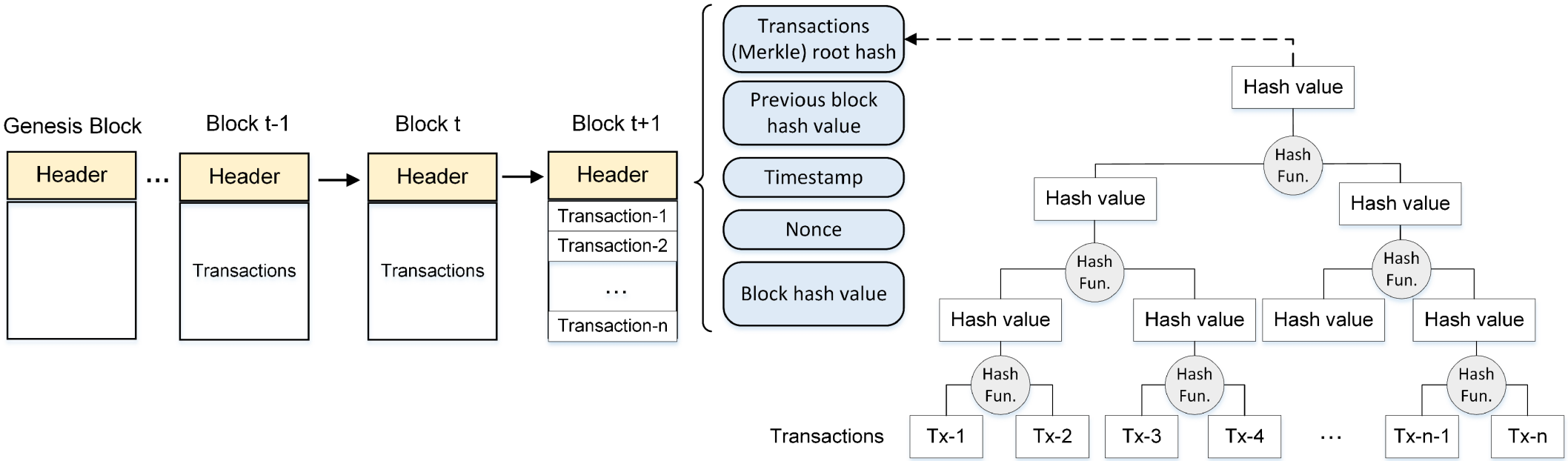
- Permissioned (private) blockchains
  - Hyperledger Sawtooth
- Другие протоколы
  - Zyzzyva, Tendermint, HotStuff, LibraBFT

Bitcoin



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

Blockchain


Genesis Block
Block t-1
Block t
Block t+1
Header
Transactions
Transaction-1
Transaction-2
Transaction-n
Transactions (Merkle) root hash
Previous block hash value
Timestamp
Nonce
Block hash value
Hash value
Hash Fun.
Transactions
Tx-1
Tx-2
Tx-3
Tx-4
Tx-n-1
Tx-n

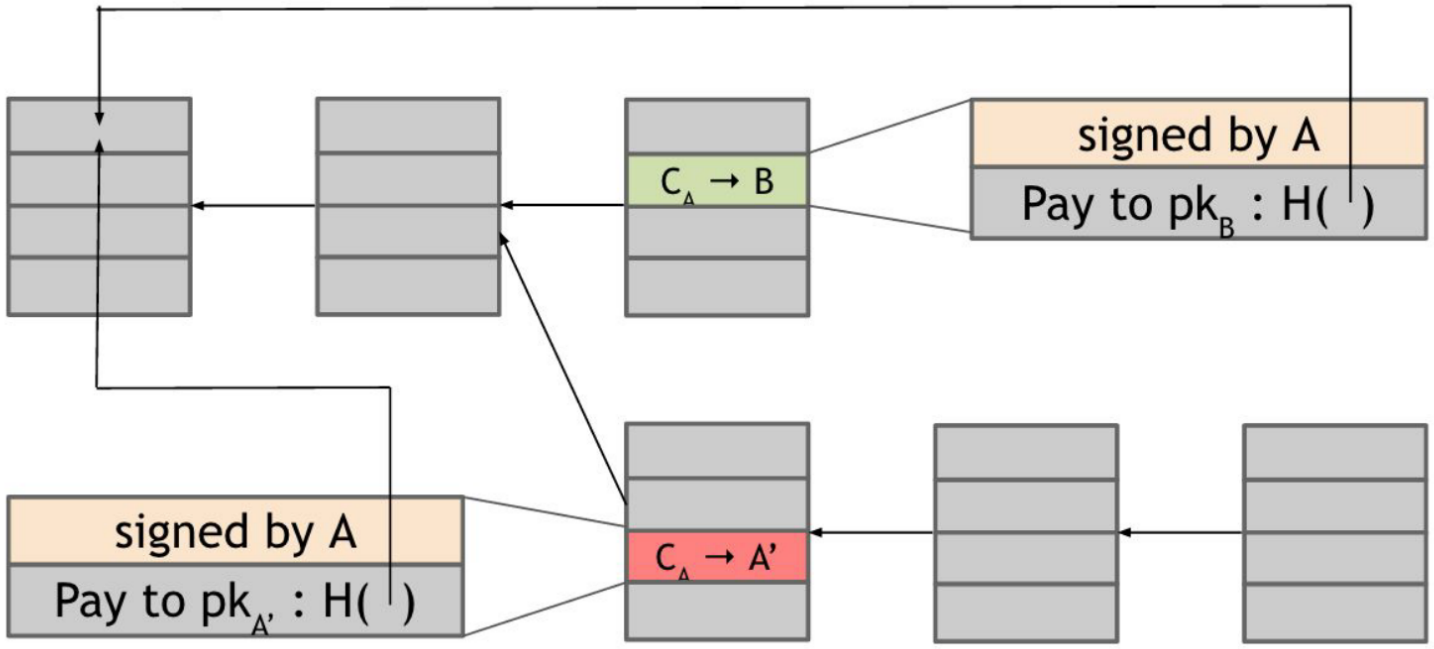
Связь с SMR и консенсусом

- 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 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 блока 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 решения - нет

Литература

- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms. Pearson, 2017.
  - Consensus in faulty systems with arbitrary failures (глава 8)
- Coulouris G.F. et al. Distributed Systems: Concepts and Design. Pearson, 2011 (раздел 15.5)
- Narayanan A. et al. Bitcoin and Cryptocurrency Technologies (главы 1-2)

Дополнительно

- Упомянутые статьи
- Driscoll K. et al. Byzantine Fault Tolerance, from Theory to Reality (2003)
- Narayanan A., Clark J. Bitcoin's Academic Pedigree
- Mickens J. The Saddest Moment