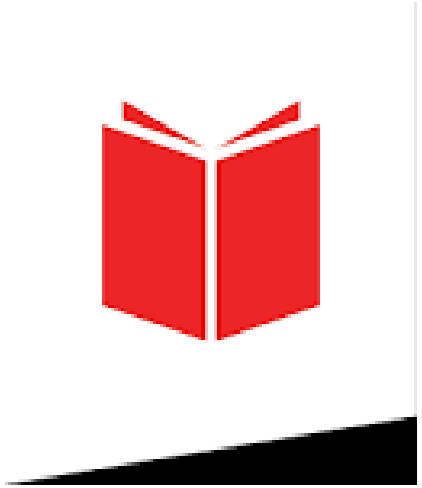
블록체인 메커니즘 및 플랫폼 (1 of 2)

목차



블록체인 메커니즘

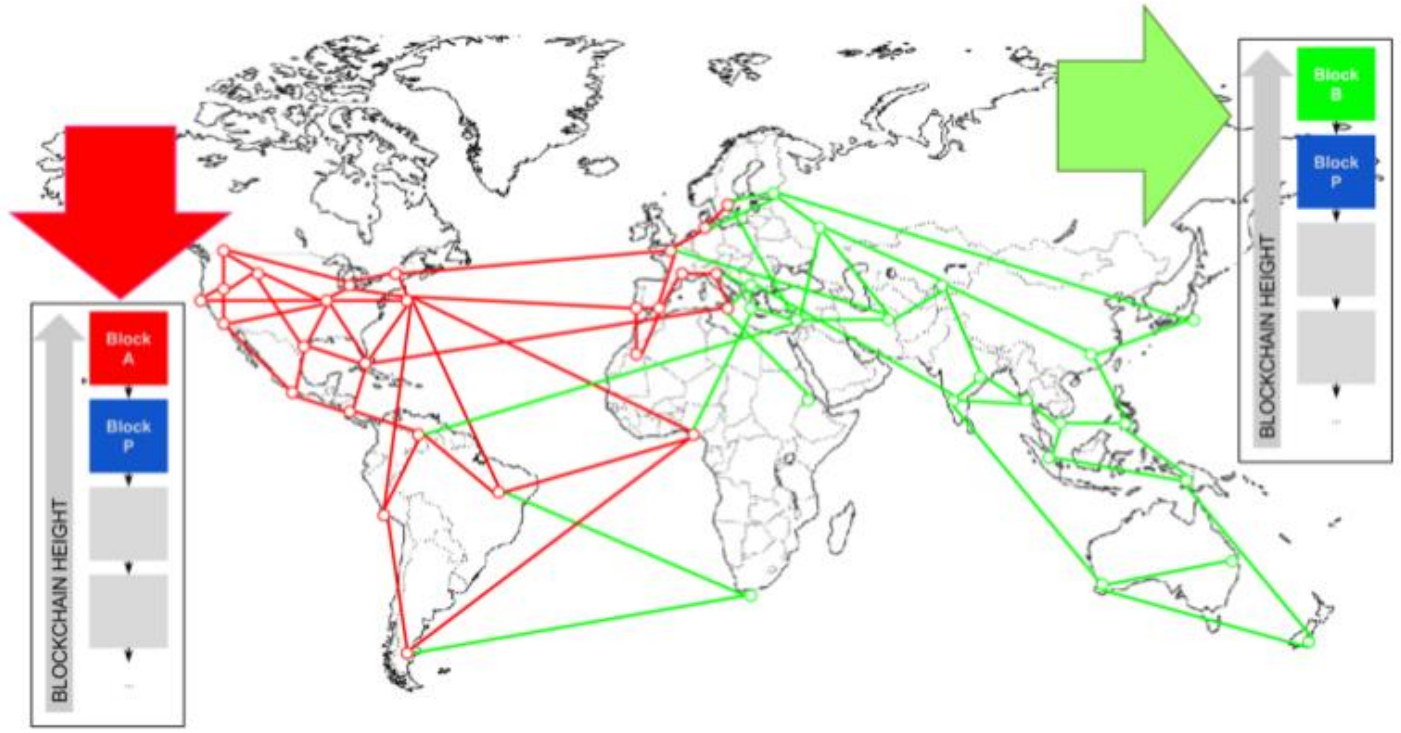
1. 블록체인 메커니즘: 거래정보 전파

- 멀리있는 노드들간 거래 내용과 순서는 다를 수 있음

출처 : https://homoefficio.github.io

1. 블록체인 메커니즘: 블록체인 충돌 해소

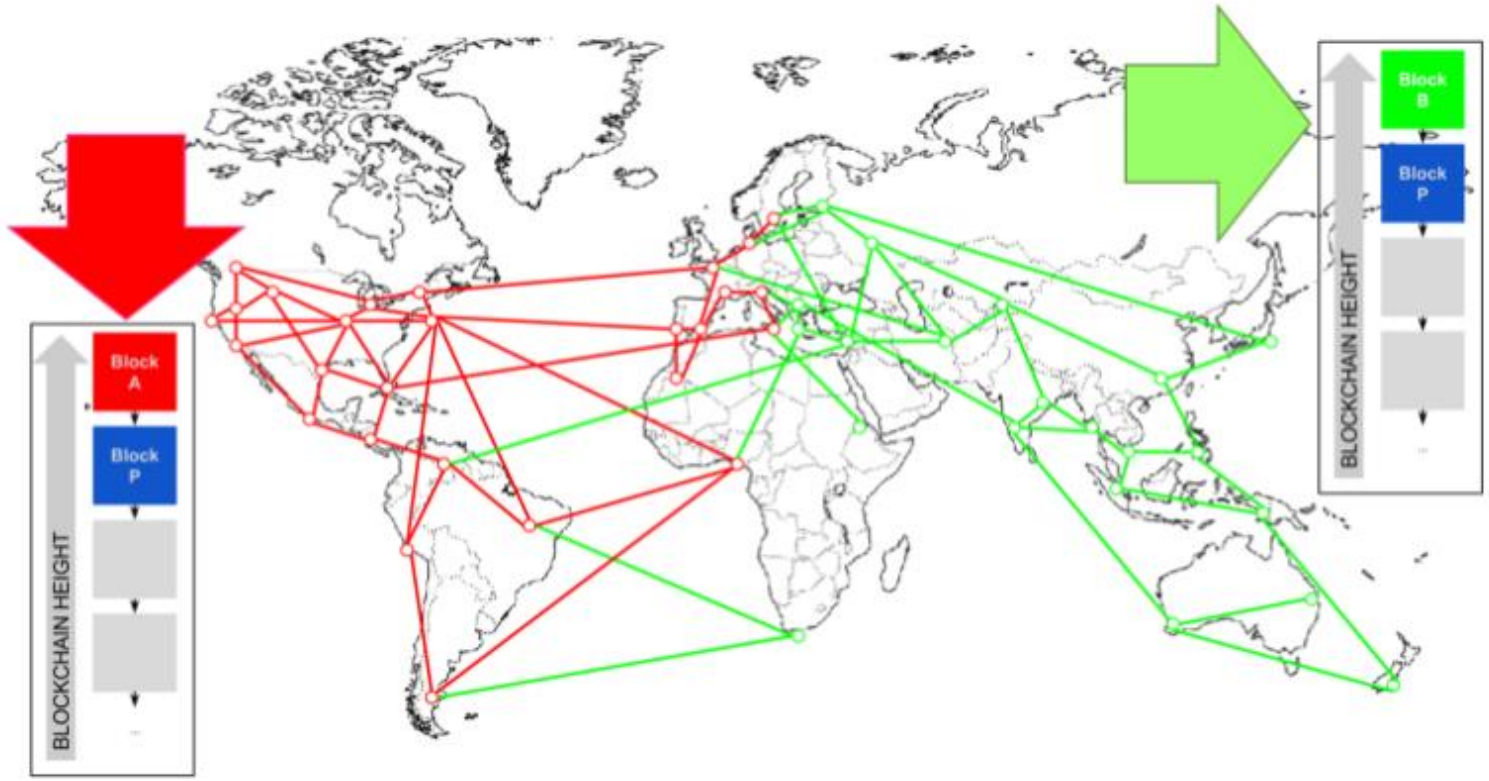
- 멀리있는 노드들이 거의 동시에 nonce값을 찾아 마지막 블록인 P에 각 새로운 블록을 추가하며 인접 노드들에게 전파



출처 : https://homoefficio.github.io

1. 블록체인 메커니즘: 블록체인 충돌 해소

- 초록블록을 먼저 받은 노드들이 빨간 블록을 전파받으면 무시하고, 반대의 경우도 마찬가지



출처 : https://homoefficio.github.io

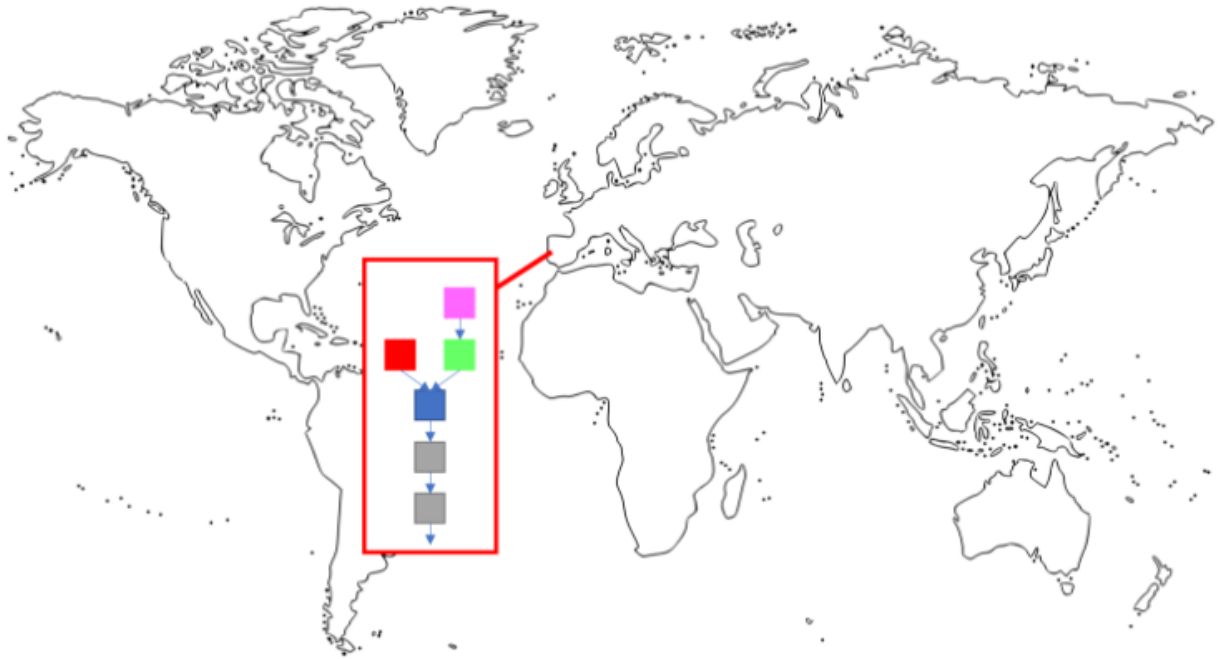
1. 블록체인 메커니즘: 블록체인 충돌 해소

- 중간지점의 초록색 블록을 이어받은 노드에서 nonce값을 구해 보라색 블록을 전파

출처 : https://homoefficio.github.io

1. 블록체인 메커니즘: 블록체인 충돌 해소

- 빨간색 블록을 형성했었던 노드에서 분기가 발생하면 더 긴 초록-보라 블록체인으로 교체



출처 : https://homoefficio.github.io

1. 블록체인 메커니즘 : 합의알고리즘

- 작업증명(PoW, Proof of Work)
- 지분증명(PoS, Proof of Stake)
- 위임지분증명(DPoS, Delegated Proof of Stake)
- 합의 알고리즘 정리 및 의의

1. 블록체인 메커니즘: 블록체인 기술의 특징

- **Proof-of-Work (PoW)** : Bitcoin, Ethereum, Litecoin
  - **Pros:** Very secure
  - **Cons:** Slow throughput, expensive computations
- **Proof-of-Stake (PoS)** : Dash, Stratis, NAV Coin, Peercoin, Decred, Nxt, Nova Coin
  - **Pros:** Attacks more expensive, energy efficient
  - **Cons:** Prone to centralisation
- **Delegated Proof-of-Stake (DPoS)** : Steemit, BitShares, EOS, Lisk, Ark, BitShares, Ethereum Casper, Tendermint, Slasher
  - **Pros:** Cheap transactions, scalable, energy efficient
  - **Cons:** Partially centralized
- **Proof-of-Authority (PoA)** : POA Network, Ethereum Kovan/Rinkeby testnet
  - **Pros:** Simple, Cost efficient, High throughput, scalable
  - **Cons:** Centralized
- **Byzantine Fault Tolerance (BFT)** : Hyperledger, NEO, Stellar, Ripple, Dispatch
  - **Pros:** High throughput, Transaction finality, Cost efficient, scalable
  - **Cons:** Centralized, Semi-trusted
  - Variantes: Practical Byzantine Fault Tolerance (PBFT) : Hyperledger, Federated Byzantine Agreement (FBA) : Stellar, Ripple, Delegated Byzantine Fault Tolerance (dBFT)

https://hackernoon.com/a-hitchhikers-guide-to-consensus-algorithms-d81aae3eb0e3

1. 블록체인 메커니즘 : 합의알고리즘

- P2P 네트워크에서는 정보의 지연과 미도달 사태를 피할수 없고 데이터를 변조할 의도가 없다 해도 이중 송신에 따른 처리 중복이나 잘못된 정보에 의한 오작동 등의 위험이 있어 정확한 정보를 공유 하기 어려움
- 블록체인과 같은 P2P 네트워크 시스템에서 각 노드간 정보 도달의 시간차이가 있을 때, 생성된 블록의 정당성을 검토하고 해당 블록을 블록체인에 연결하기 위해 네트워크 참가자들의 합의를 얻기 위한 알고리즘

블록체인 메커니즘 : 합의알고리즘 종류

- 비트코인의 경우 작업증명 알고리즘을 사용중이며, 이더리움의 경우 2017년 8월부터 작업증명과 지분증명 알고리즘을 Hybrid 형태로 테스트하고 있음. 향후, 지분증명으로의 전환을 목표로 하고 있음
- 작업증명 (Proof of Work, PoW)
  - 블록체인에서 가장 보편적으로 사용중인 합의 알고리즘으로 컴퓨팅 파워를 이용하여 특정 난이도의 해시값을 역함수 해시화 하여 Nonce값을 계산해내고 이를 검증하는 것으로 합의를 도출함
- 지분증명 (Proof of Stake, PoS)
  - PoW의 컴퓨팅 파워 낭비 문제를 해결하고자 개발된 합의 알고리즘으로 노드가 보유한 자산을 기준으로 권한을 분배하여 합의를 도출하고 보상을 분배하는 알고리즘
- Proof of Elapsed Time (PoET) 등의 다양한 알고리즘 존재

1. 블록체인 메커니즘 : 합의알고리즘 종류

- Private Blockchain의 경우 보편적으로 PBFT와 PAXOS 알고리즘을 사용하고 있으며, Enterprise Ethereum의 대표 프로젝트인 Quorum의 경우 Raft 알고리즘을 채택하여 사용하고 있음
- PAXOS
  - 가장 일반적인 합의 알고리즘으로 Leader를 선정하고 과반수의 동의에 의해 합의를 이룸
- Practical Byzantine Fault Tolerance (PBFT)
  - 비잔틴 장군 문제를 해결하고자 고안된 합의 알고리즘으로 투표 메카니즘을 도입한 3단계 프로토콜을 이용한 합의 도출로 프라이빗 블록체인에서 널리 이용됨
- Raft
  - PAXOS를 보완한 형태로, 투표와 랜덤 타임아웃을 통한 리더 선출로 절차를 단순화 하는 것이 특징
  - SBFT, Tendermint 등의 다양한 알고리즘 존재

1. 블록체인 메커니즘 : POW (Proof of Work)

- 비트코인: 최초의 블록체인 기술이 적용된 시스템
  - 합의 알고리즘으로 작업 증명(Proof of Work: PoW)과 가장 긴 체인(Longest Chain)을 선택하는 방법 사용
  - 최대 7 TPS밖에 처리할 수 없는 성능의 한계
  - 작업증명으로 인해 많은 에너지낭비
- 작업증명으로 부르기도 하며 풀기 어려운 문제를 빨리 해결한 사람에게 블록을 생성할 수 있는 권한을 주고 그 보상으로 코인을 제공
- 비트코인의 경우 해시 함수의 결과값이 특정 값보다 작아지도록 하는 입력 값(Nonce)을 찾는 문제
- 비트코인의 경우 약 10분 정도 걸려 풀릴 수 있도록 난이도 조절
- 비트코인은 Nonce값을 만드는데 SHA-256이라는 알고리즘 사용
  - SHA(Security Hash Algorithm)은 미국 표준 기술 연구소(NIST)에 의해 공표된 해시 알고리즘
  - SHA-256은 256비트로 구성되어 64자리 문자열을 반환
- PoW채용 코인의 종류: 비트코인, 라이트코인, 제트캐시, 모네로 등의 PoW방식 코인들이 있음

1. 블록체인 메커니즘 : POW의 작업정의

- 블록(B)의 해쉬값 Hash(B) <= M/D로 정의
  - D: 난이도 (Difficulty)
  - M은 난이도 D의 최대값 (2^256-1)
- Miner들은 반복적으로 위의 조건을 만족하는 블록 B의 해쉬값을 탐색하기위해 작업

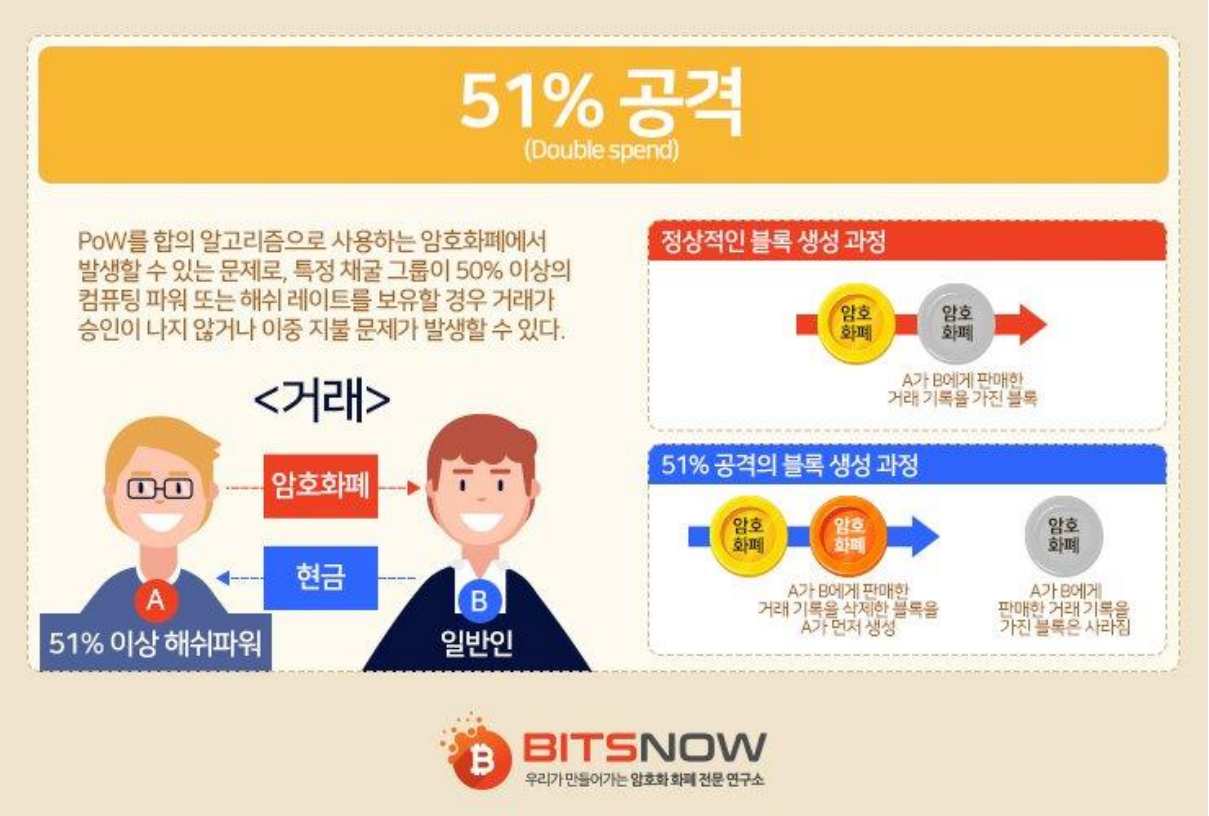
1. 블록체인 메커니즘 : POW 보상

- Nonce값을 구하기 위해선 많은 작업 비용이 들며 이러한 행위의 보상이 없다면 아무도 채굴하지 않을 것
- 비트코인에서 보상은 새로 발행되는 비트코인과 해당 블록에 포함된 거래의 거래 수수료의 합
- 블록의 첫 거래는 채굴자에게 보상을 주는 거래로 시작

출처 : : https://homoefficio.github.io/2017/11/19

1. 블록체인 메커니즘: PoW 51% 공격

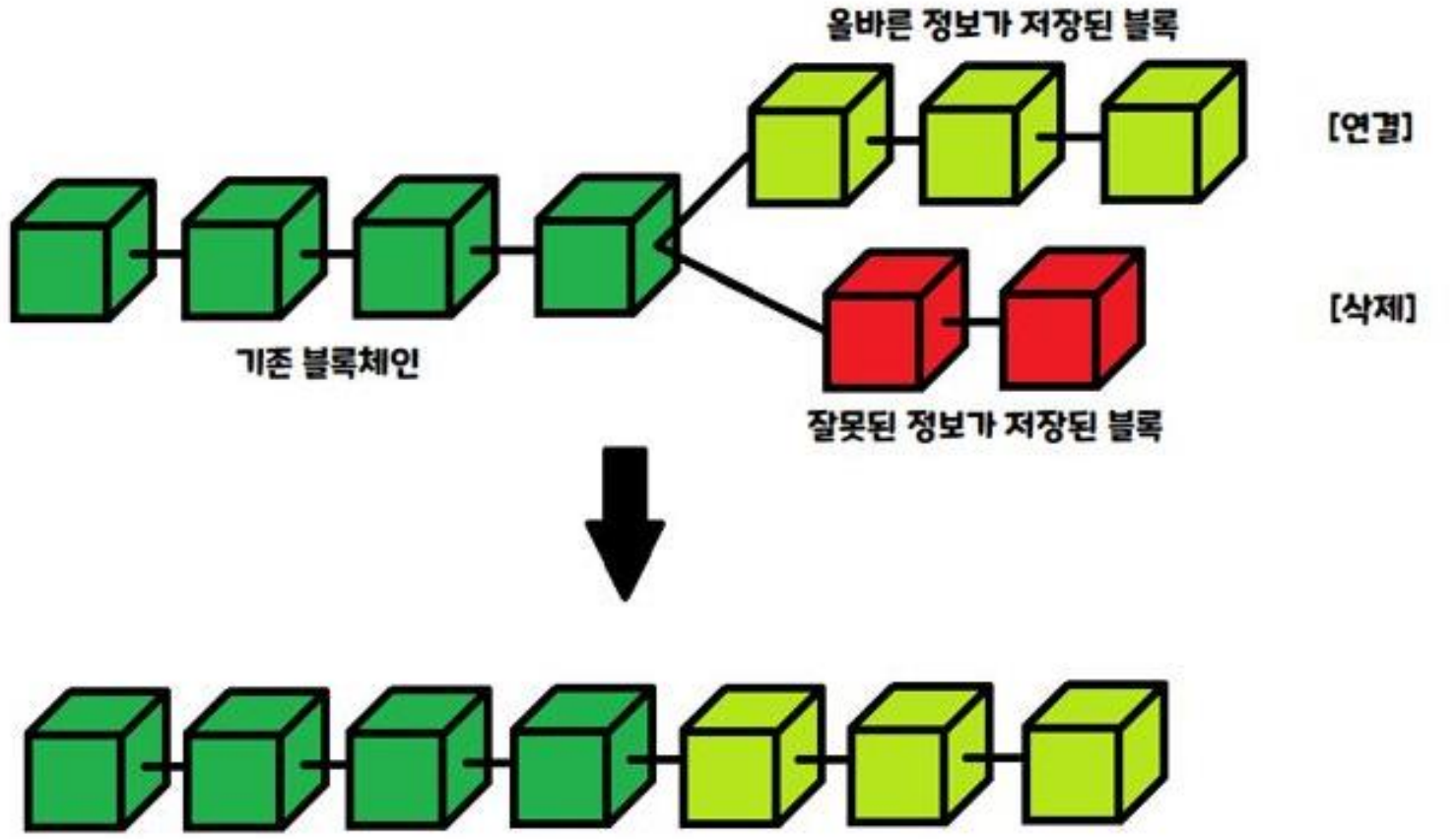
- PoW는 다수결로 결정을 내리는 알고리즘
- 악의적인 참여자가 절반이상의 해시 파워를 가지게 된다면, 블록 내용의 조작이 가능



https://blockinpress.com/archives/5740

1. 블럭체인 메커니즘: PoW 51% 공격

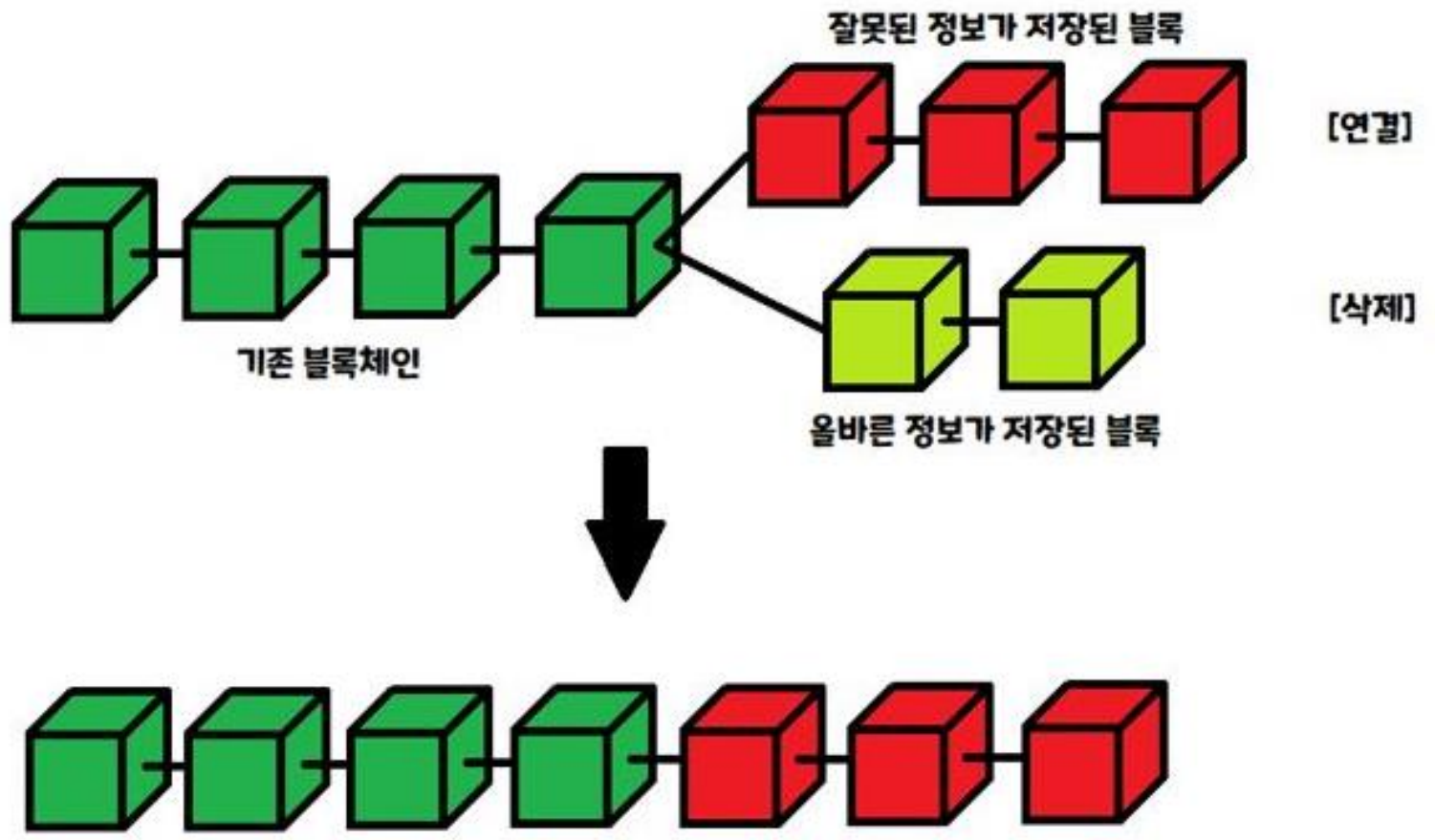
- 건강한 블록체인 연결방식



출처 : http://cryptochain.tistory.com/53

1. 블럭체인 메커니즘: PoW 51% 공격

- 51%공격을 받은 블록체인 연결방식



출처 : http://cryptochain.tistory.com/53

1. 블럭체인 메커니즘: PoW 파이널리티 불확실성

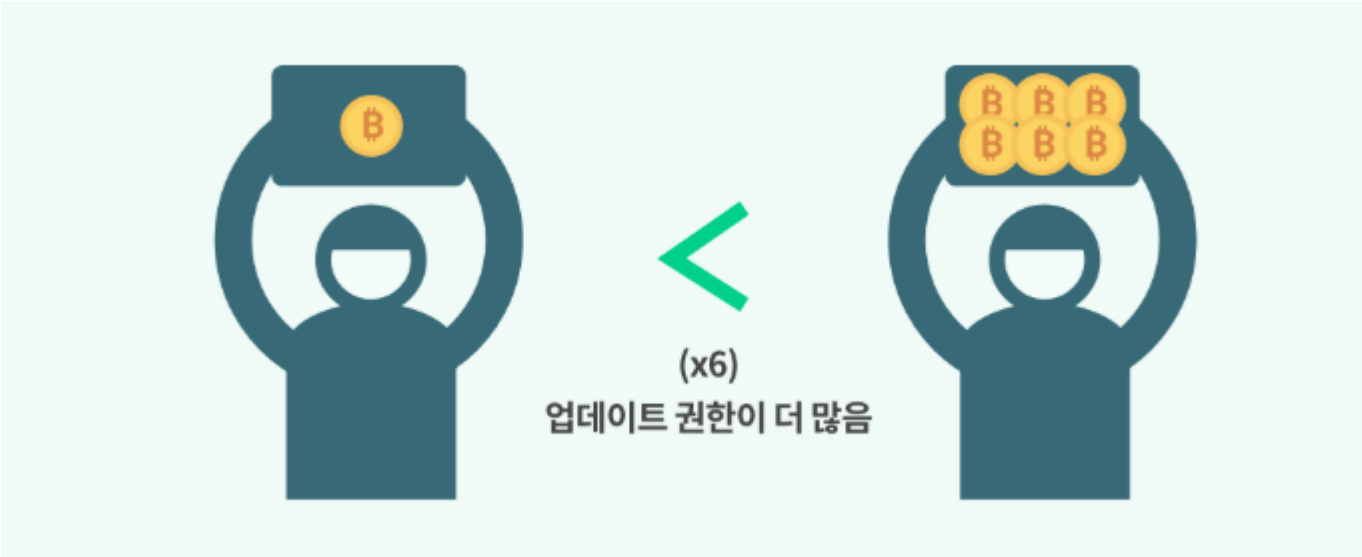
- PoW는 블록체인이 분기하는 경우 긴 체인을 올바른 것으로 판단
- 짧은 체인이 버려지는 경우 트랜잭션이 없었던 일로 되는 경우가 발생할 수 있음
- 비트코인같은 경우 이런 현상을 방지하기 위해 트랜잭션이 확정되더라도 6블럭 가량 기다리게 하는 등 제한을 함

1. 블럭체인 메커니즘: PoW 성능 한계

- P2P 네트워크에서 단일 정보를 공유하는 구조상 네트워크에 확산되는 시간을 없애기는 불가능
- 여러 노드간 합의를 통해 정보의 신뢰성을 담보하기 때문에 합의에 걸리는 시간이 필요
- 따라서, 성능(응답시간과 처리량)을 올리는 것이 어려움
- 블록생성에 약 10분 걸리므로 실시간성이 보장되지 않음

1. 블럭체인 메커니즘: PoS (Proof of Stake)개요

- 지분증명으로 부르기도 하며 앞의 PoW와 다르게 컴퓨터의 해시파워가 아닌, 참여자의 코인 지분을 기준으로 블록을 생성
- PoS는 참여자의 코인 지분이 많을 수록 유리해지는 방식



출처 : https://brunch.co.kr/@banksalad/313

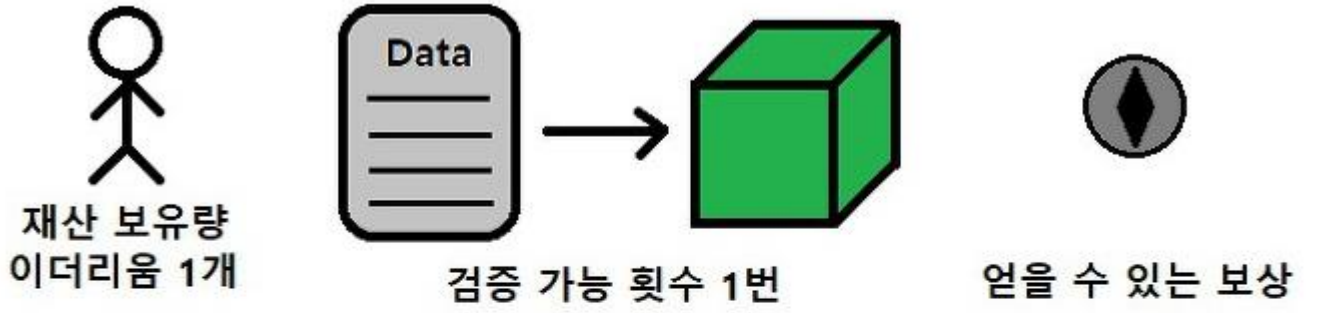
1. 블록체인 메커니즘: PoS의 Hash함수 정의

- Hash (Hash (Bprev), A, t) <= bal(A) M/D
  - Bprev는 이전블록, A는 계정 (Address), t는 타임스탬프, bal(A)는 address A가 현재 소유한 balance, D는 난이도, M은 D의 최대값을 의미
- 블록 B의 해쉬값은 A가 소유한 밸런스와 난이도의 영향을 받는다. 많은 지분 소유자가 쉬운 난이도의 문제를 풀게된다.

https://medium.com/@poolofstake/a-proof-of-stake-overview-445c52558d03

1. 블록체인 메커니즘: PoS 개요

- PoS는 PoW와 다르게 블록 생성시 보상이 없음
- 대신 PoS는 해당 지분만큼의 거래 수수료 를 받음



출처 : http://cryptochain.tistory.com/49

1. 블록체인 메커니즘: PoS 개요

- 자산을 보유하는 노드들은 자신이 합의하는 블록에 자산을 증명함으로써 데이터를 업데이트

출처 : https://brunch.co.kr/@banksalad/313

1. 블록체인 메커니즘: PoS개요

- PoW와 비슷하게 체인이 분기되면 과반수의 자산이 동의한 블록으로 합쳐지게 됨

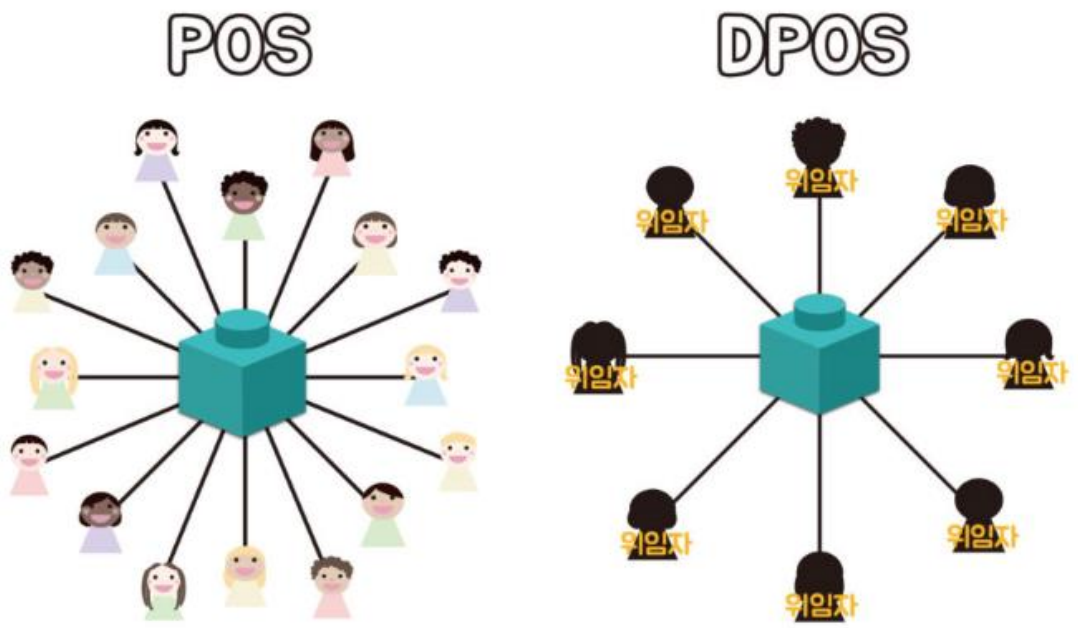
출처 : https://brunch.co.kr/@banksalad/313

1. 블록체인 메커니즘: Delegated Proof of Stake(DPoS)

- 위임지분증명이라 부르기도 하며 특정 인원에게만 PoS를 할 수 있도록 권한을 위임하는 것
- 네트워크상 노드들의 투표 결과로 선출된 상위 노드에게 권한을 위임하여 일종의 대표자가 되는 것
- 위임한 대표자와 수익을 배분함

1. 블록체인 메커니즘: DPoS

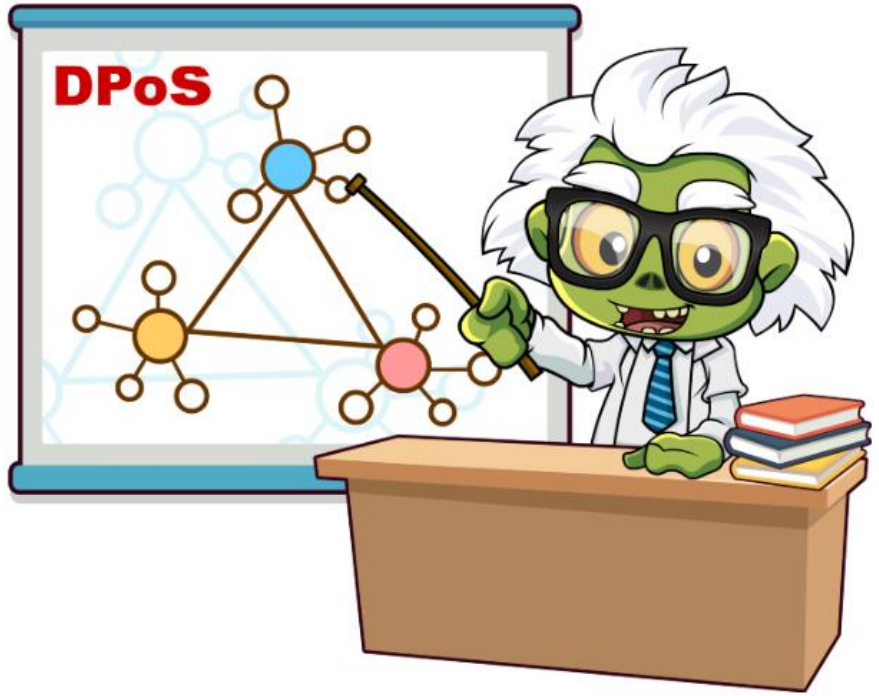
- PoS의 경우 일정 지분을 소유한 모든 노드에게 블록 생성 권한이 주어지기에 시간이 오래걸림
- DPoS의 경우 투표 결과로 정한 상위 노드 라는 비교적 적은 숫자로 인해 합의 시간과 비용이 줄어듬



이미지 출처 : https://brunch.co.kr/@mobiinside/1163

1. 블록체인 메커니즘: DPoS 블록 대표자(생산자) 수 Case

- EOS : 21
- BitShares : 101
- Steemit : 21
- Lisk : 101
- Ark : 51



https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef

1. 블록체인 메커니즘: 합의알고리즘의 중요성

- 앞에서 언급한 것과 같이 P2P 네트워크에서 다수의 참여자들이 존재하고 있어 이 참여자들의 하나의 블록체인을 유지하기 위한 수단
- 합의 알고리즘은 각 노드에서 블록체인을 공유하기 위해 사용되는 중요한 기능이며 블록체인 기술의 핵심이라 할 수 있음

1. 블록체인 플랫폼: 합의알고리즘 과제

- 최초의 블록체인 소프트웨어 비트코인은 PoW가 사용되고 있지만 자원소모, 파이널리티 불확성등의 문제로 컨소시엄형이나 프라이빗 블록체인에 이용하기엔 적절하지 않음
- 컨소시엄형에서의 이용을 전제로 한 블록체인이 등장하고 있으며 PoW가 아닌 합의 알고리즘을 채택하는 경우가 많아짐
- 각각의 합의 알고리즘의 특징을 정확히 이해하고 특성에 맞게 활용해야 함

PBFT
PBFT(Practical Byzantine Fault Tolerance)

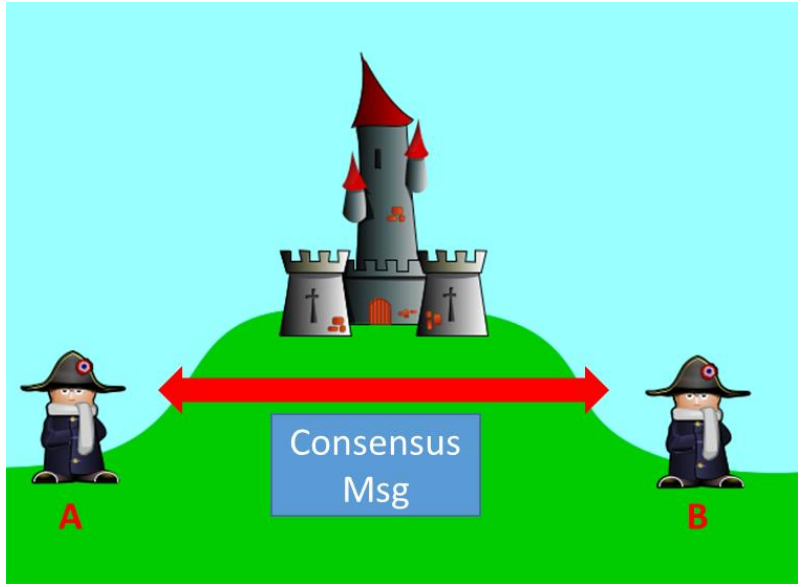
Two Generals' Problem

- 두 장군 A, B는 동시에 적을 공격해야 성을 함락시킬 수 있음
  - 두 장군은 적진을 사이에 두고 있으며, 합의 메시지는 적을 통과해야만 도착할 수 있음

- A가 B에게 공격 시간 합의 메시지를 보내는 경우
  - A는 B가 메시지를 전달 받았는지 확인할 수 없음
  - 또한, A의 메시지가 적에 의해 변조되었는지 확인할 수 없음

- B가 A에게 응답하는 경우
  - B는 A가 응답 메시지를 전달 받았는지 확인할 수 없음
  - 또한, B의 응답 메시지가 적에 의해 변조되었는지 확인할 수 없음

- A와 B 사이에 합의 하는 것은 불가능함



Byzantine Empire

Src: https://www.slideshare.net/YongRaeJo/pbft-86070872

BGP (Byzantine Generals Problem)

- **Paper**
  - "The Byzantine Generals Problem “, Lamport, L.; Shostak, R.; Pease, M. (1982). ACM Transactions on Programming Languages and Systems

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

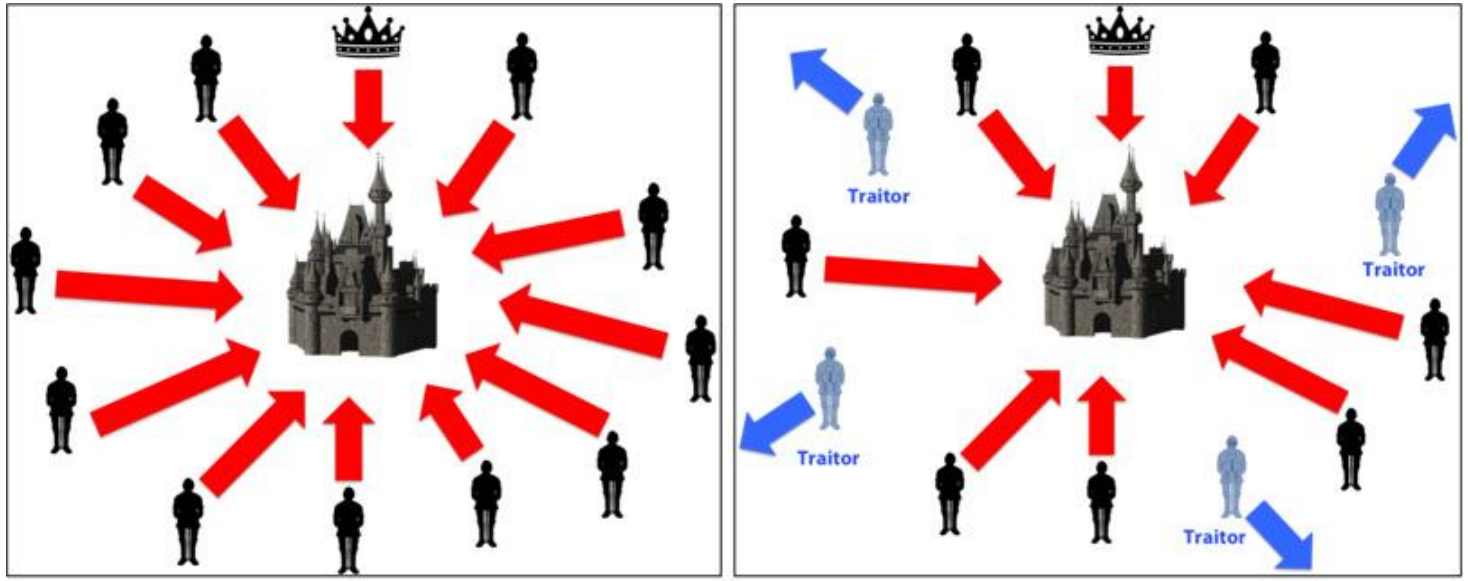
Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

BGP (Byzantine Generals Problem)

- 여러 장군들 중 악의적인 장군(비잔틴 장군)이 존재할 시 합의하는 방법을 연구



Coordinated Attack Leading to Victory

Uncoordinated Attack Leading to Defeat

PBFT (Practical Byzantine Fault Tolerance)

- **Paper**
  - **"Practical Byzantine Fault Tolerance and Proactive Recovery “, Castro, M.; Liskov, B. (2002). ACM Transactions on Computer Systems.**
- **N = 3f + 1**
  - **N = 전체 네트워크 노드 수**
  - **f = 비잔틴 노드 수**

Practical Byzantine Fault Tolerance and Proactive Recovery

MIGUEL CASTRO
Microsoft Research
and
BARBARA LISKOV
MIT Laboratory for Computer Science

Our growing reliance on online services accessible on the Internet demands highly available systems that provide correct service without interruptions. Software bugs, operator mistakes, and malicious attacks are a major cause of service interruptions and they can cause arbitrary behavior, that is, Byzantine faults. This article describes a new replication algorithm, BFT, that can be used to build highly available systems that tolerate Byzantine faults. BFT can be used in practice to implement real services: it performs well, it is safe in asynchronous environments such as the Internet, it incorporates mechanisms to defend against Byzantine-faulty clients, and it recovers replicas proactively. The recovery mechanism allows the algorithm to tolerate any number of faults over the lifetime of the system provided fewer than 1/3 of the replicas become faulty within a small window of vulnerability. BFT has been implemented as a generic program library with a simple interface. We used the library to implement the first Byzantine-fault-tolerant NFS file system, BFS. The BFT library and BFS perform well because the library incorporates several important optimizations, the most important of which is the use of symmetric cryptography to authenticate messages. The performance results show that BFS performs 2% faster to 24% slower than production implementations of the NFS protocol that are not replicated. This supports our claim that the BFT library can be used to build practical systems that tolerate Byzantine faults.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server*; D.4.3 [**Operating Systems**]: File Systems Management; D.4.5 [**Operating Systems**]: Reliability—*Fault tolerance*; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls; authentication; cryptographic controls*; D.4.8 [**Operating Systems**]: Performance—*Measurements*

General Terms: Security, Reliability, Algorithms, Performance, Measurement

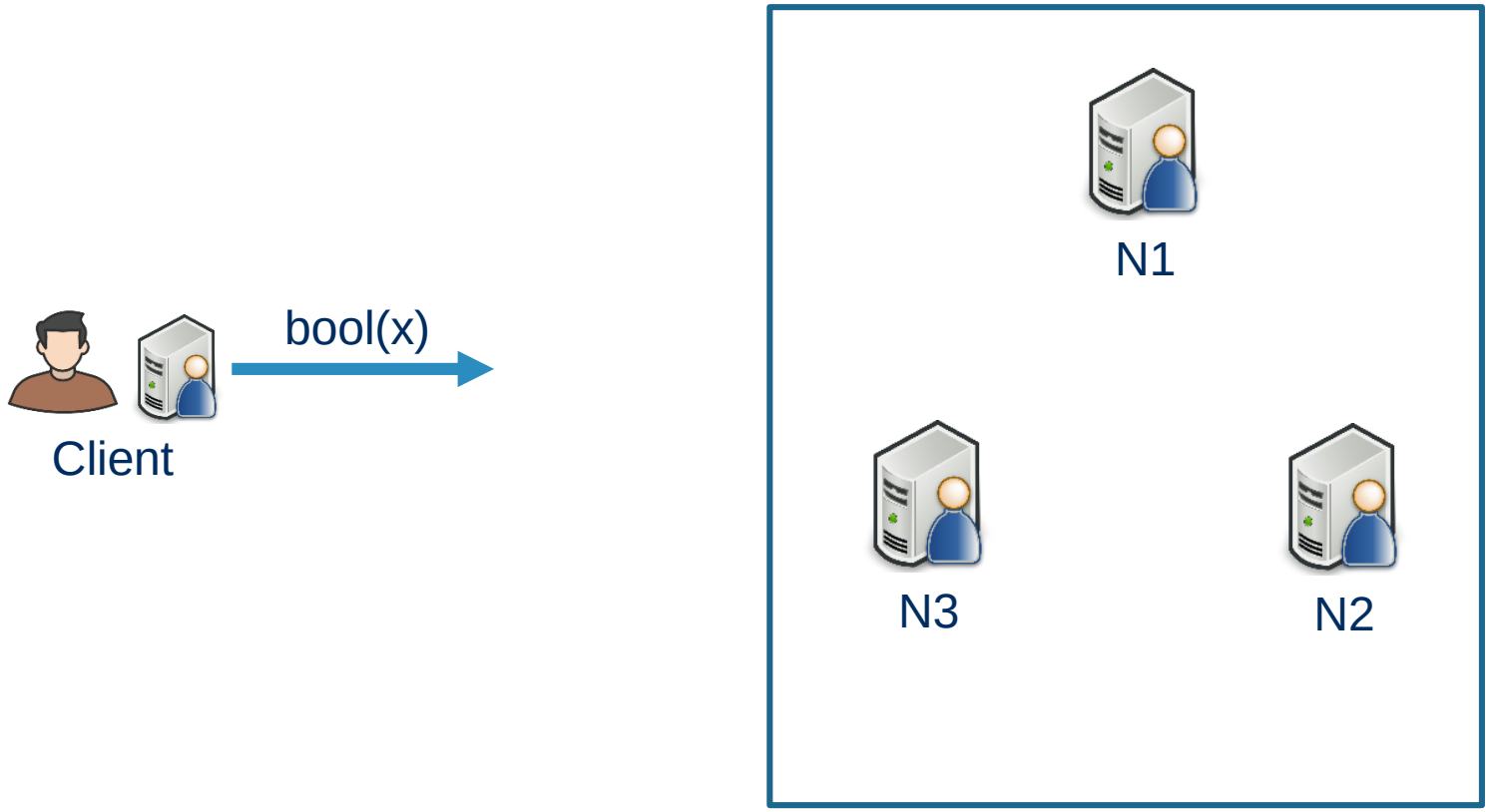
Additional Key Words and Phrases: Byzantine fault tolerance, state machine replication, proactive recovery, asynchronous systems, state transfer

BFT (Byzantine Fault Tolerance)

- Terminology
  - Byzantine Fault: 시스템 장애, 혹은 악의적인 공격에 의해 발생하는 장애
  - Byzantine Failure: Byzantine Fault에 의해 네트워크(서비스)가 중단된 경우
  - BFT: 분산 네트워크가 정상적으로 동작할 수 있는 비잔틴 노드의 수

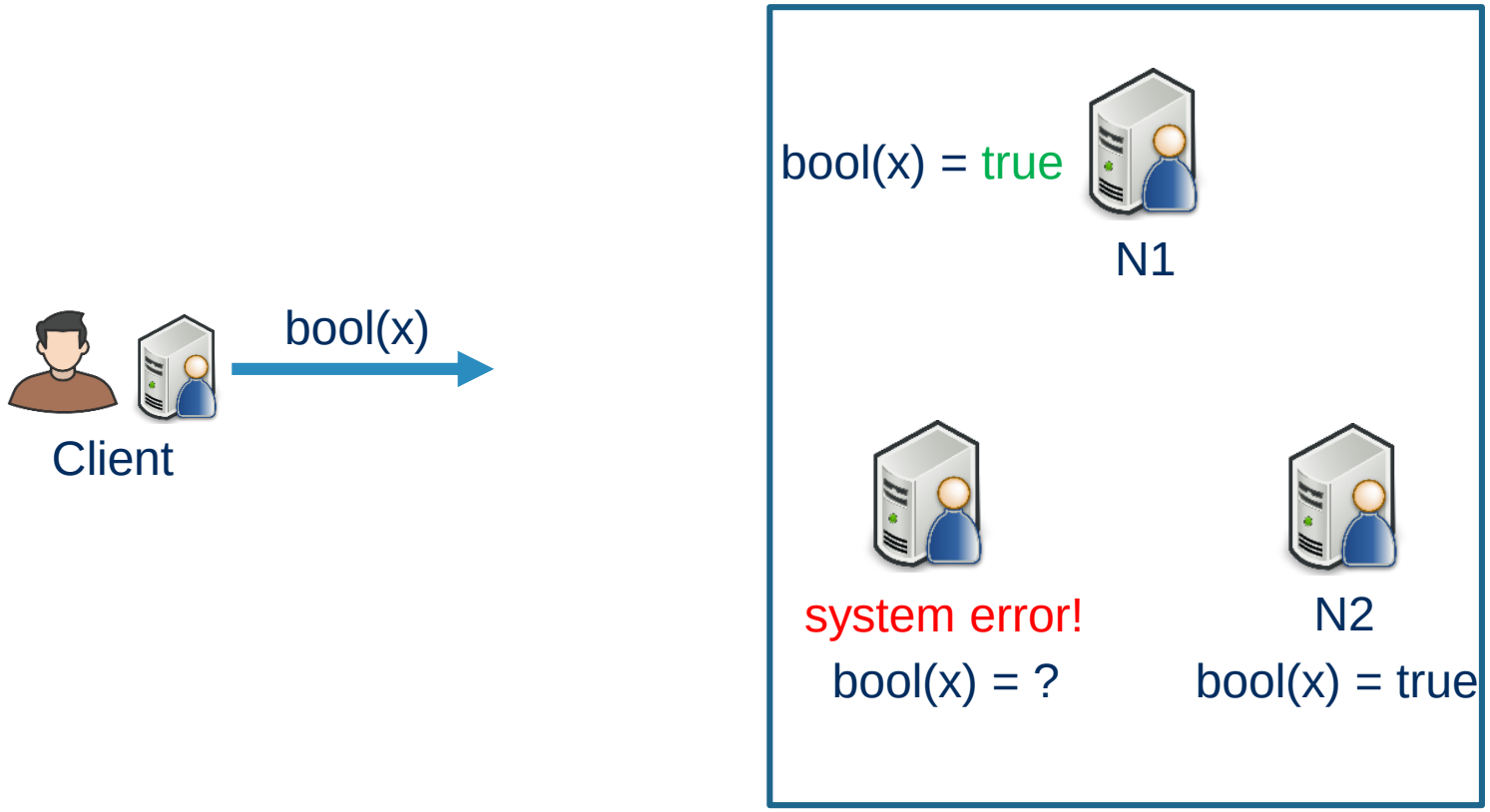
Why N = 3f + 1?

- **N = 2f + 1, f = 1 ?**



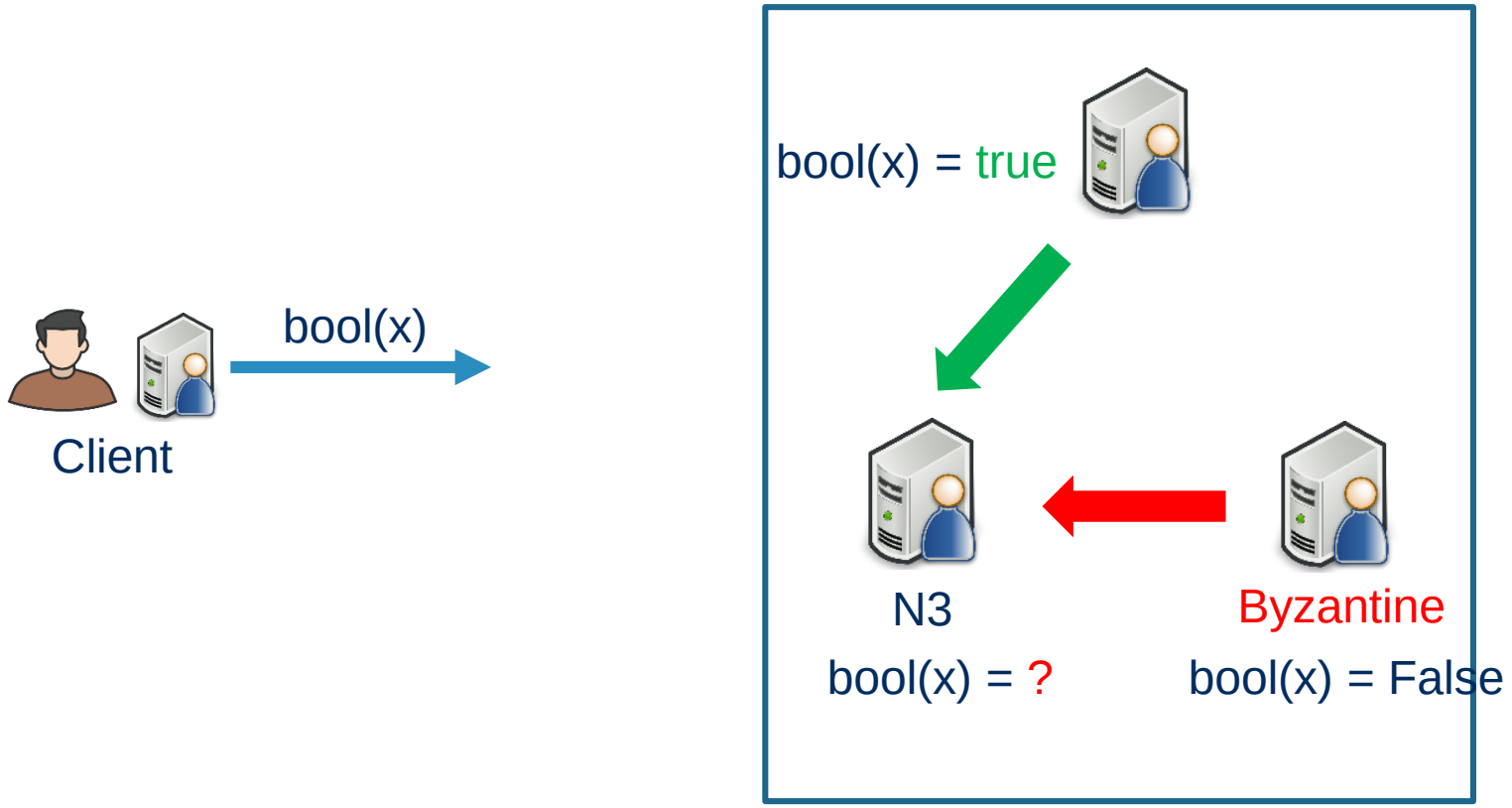
Why N = 3f + 1?

- **N = 2f + 1, f = 1 ?**



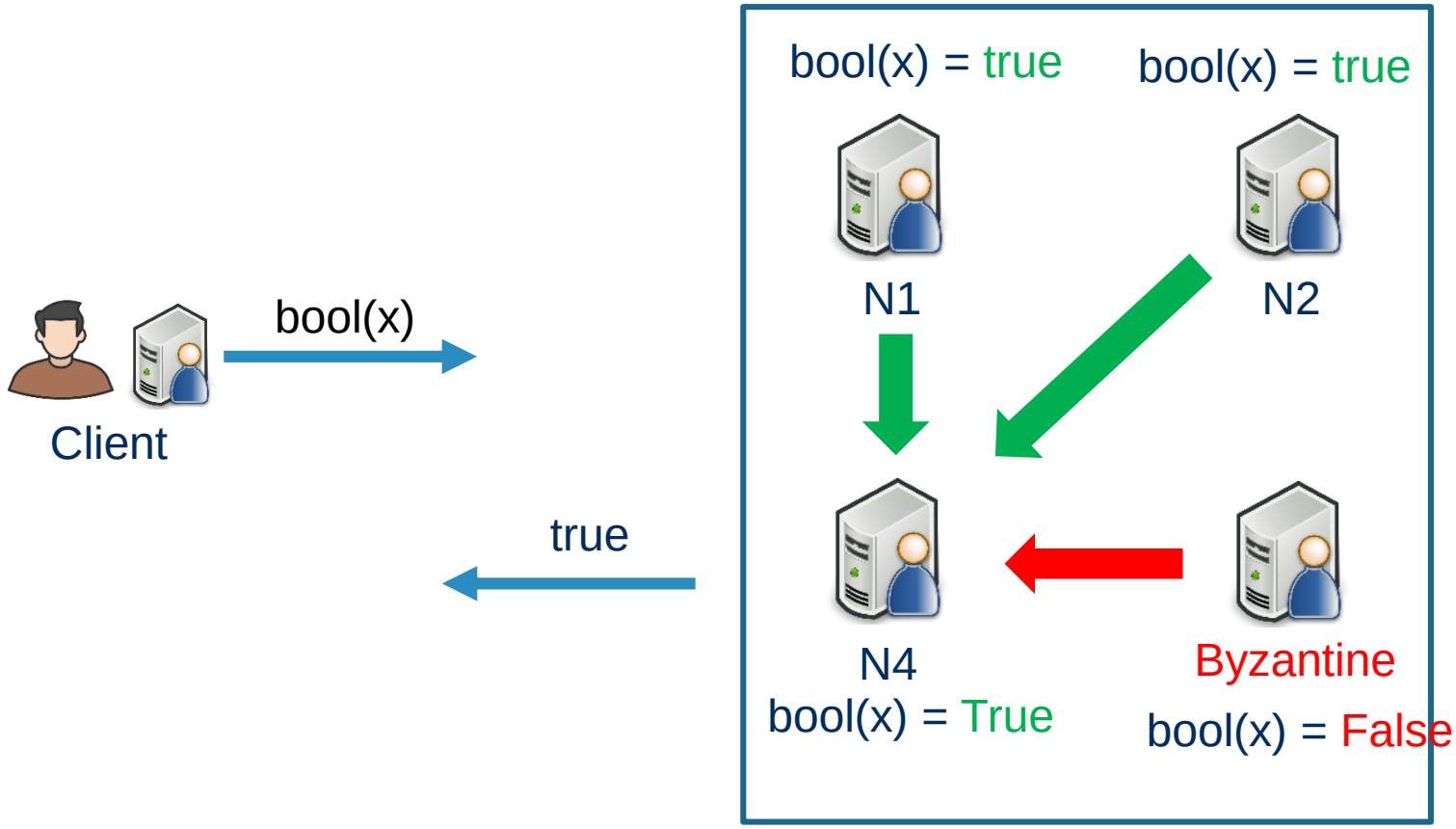
Why N = 3f + 1?

- **N = 2f + 1, f = 1 ?**



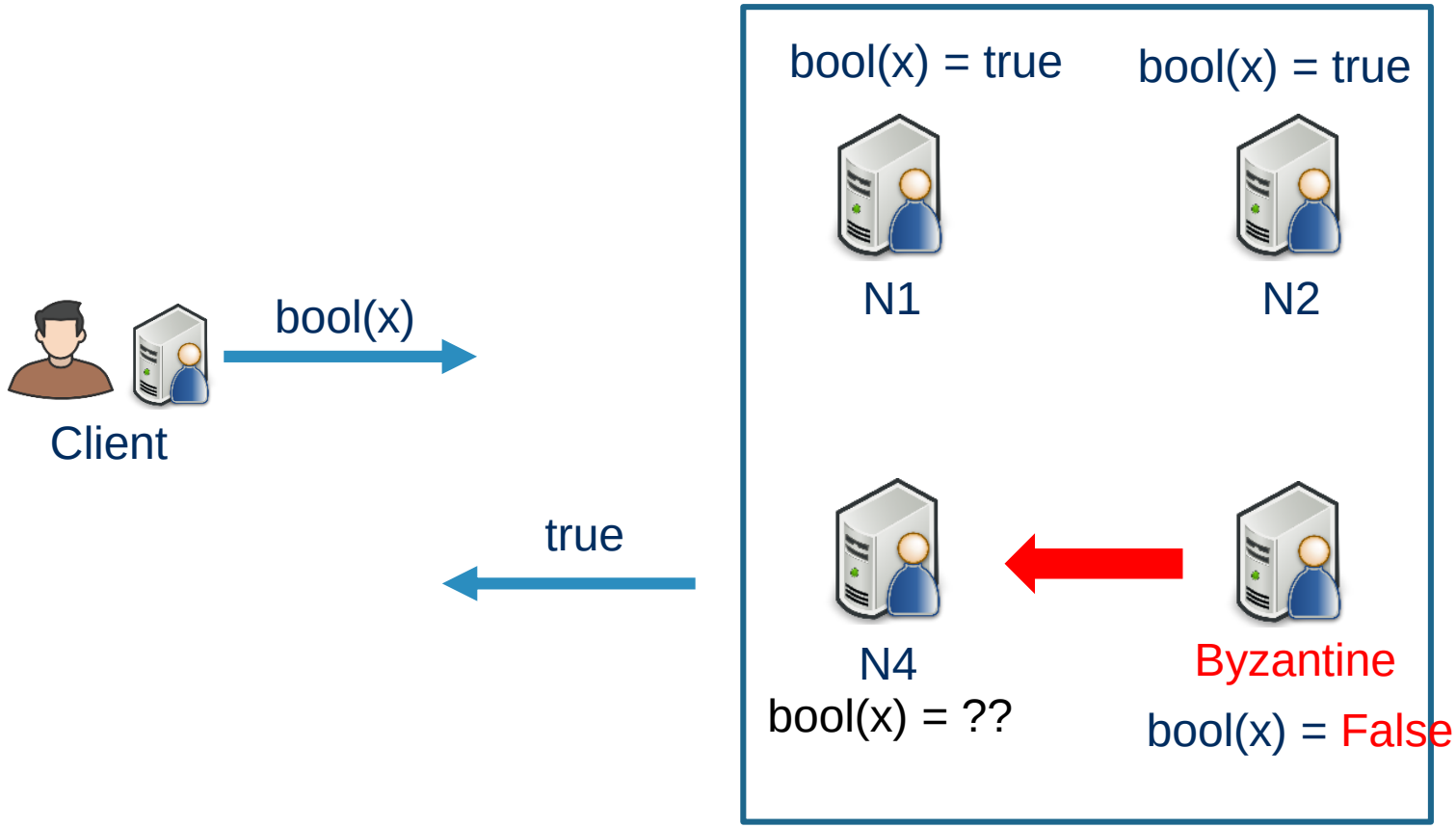
Why N = 3f + 1?

- **N = 3f + 1, f = 1 ?**



Why N = 3f + 1?

- **N = 3f + 1, f = 1 ?**



Why N=3f+1?

- 비동기 네트워크에서 발생할 수 있는 장애는 Case1과 Case2가 존재
  - 네트워크가 작동하기 위한 최소한의 조건이기 때문에 정상적인 합의를 위해서는 Case1, Case2 모두 만족해야 함

Case1

- 메시지 전송을 하지 않는 경우
  - N-f 노드 간 합의를 수행해야함(N = 전체 노드 수, f = 메시지를 보내지 않는 노드)

Case2

- 잘못된 메시지를 보내는 경우
  - Case 1에서 남은 (N-f)노드 중 악의적인 메시지를 보내는 노드 f개가 존재할 경우
  - (N-f)-f>f 조건이 만족되어야 합의가 이루어 짐
  - N>3f
  - 최소 N=3f+1개의 노드가 필요

P2P 시스템 장애 모델

- BYZANTINE FAULT 모델
  - 비잔틴 장군문제
    - 각 지휘관들은 지리적으로 떨어져 있는 상황에서 동시에 공격할 수 있도록 전령을 통해 합의를 하는중
    - 내부 배신자가 나타나 공격 시간을 교란하면 각 지휘관들은 어떻게 정확한 공격시간을 판단할 것인가?

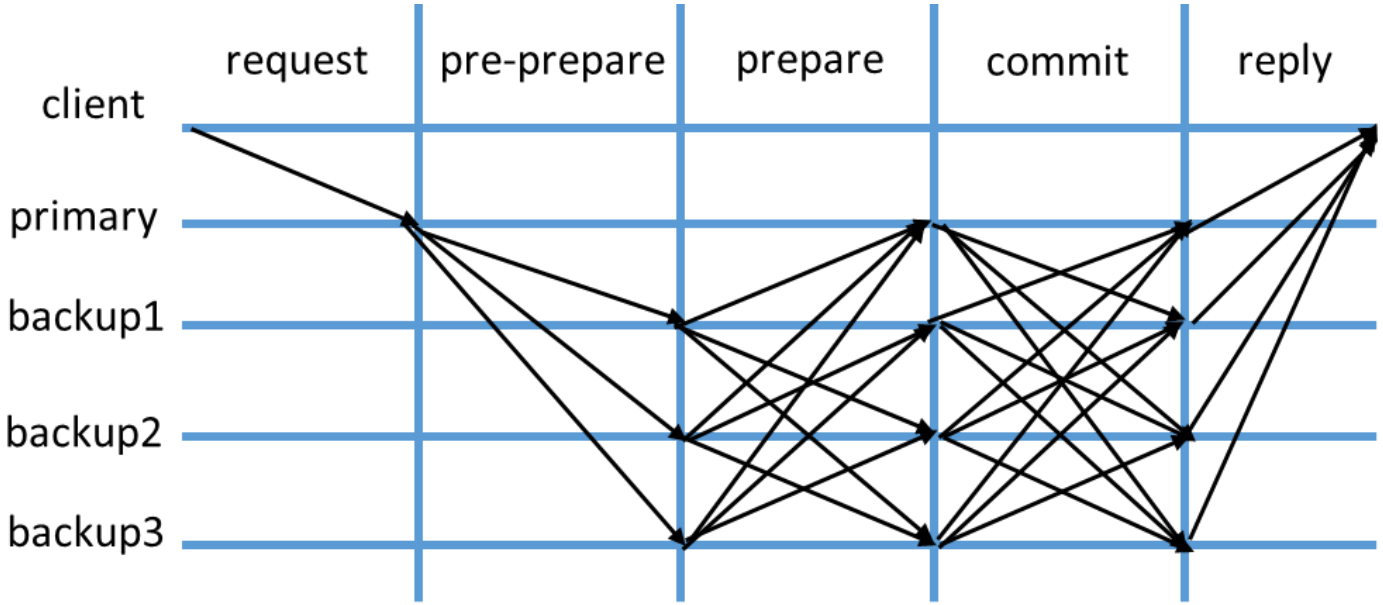
PBFT(Practical Byzantine Fault Tolerance)

- 비잔틴 장군 문제를 해결하기 위한 실질적인 프로토콜
  - 비잔틴 장군 문제 : 장군이 악의적인 행동을 할 수 있다는 점을 가정
  - 단순 고장난 노드 뿐만 아니라 악의적인 노드가 있음에도 불구하고 전체 시스템이 안정적으로 동작하도록 하는 프로토콜
- BFT 계열 프로토콜 중 실용적으로 쓰일 수 있는 가장 대표적인 프로토콜
- 리플리카 중 의사결정의 리더 역할을 하는 primary 노드가 있으며, primary 노드의 주도하에 순차적으로 명령이 수행됨
- primary 노드가 고장이 나거나 악의적인 행동을 하게 될 경우 'view change'라는 절차를 통해 primary 노드를 바꿈

PBFT(Practical Byzantine Fault Tolerance)

PBFT in rough

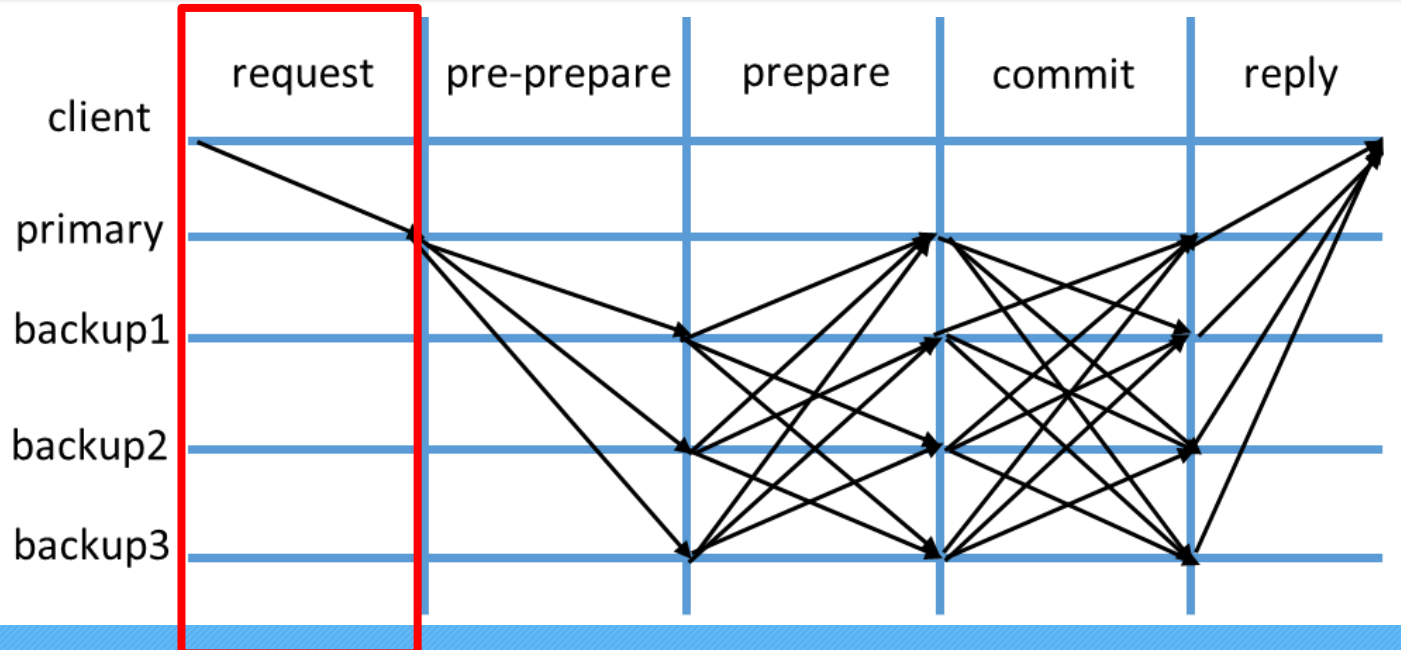
- 클라이언트는 Primary 노드에게 요청을 전송
- Primary 노드는 backup 노드에게 요청을 전파하고 합의 과정을 수행
- 합의 과정이 완료되면 Primary 노드와 Backup 노드는 클라이언트에게 완료 메시지를 전송
- 클라이언트는 f+1개 이상의 똑같은 답변을 backup노드로 부터 받으면 요청이 제대로 반영되었다는 것을 확신



PBFT(Practical Byzantine Fault Tolerance)

Request

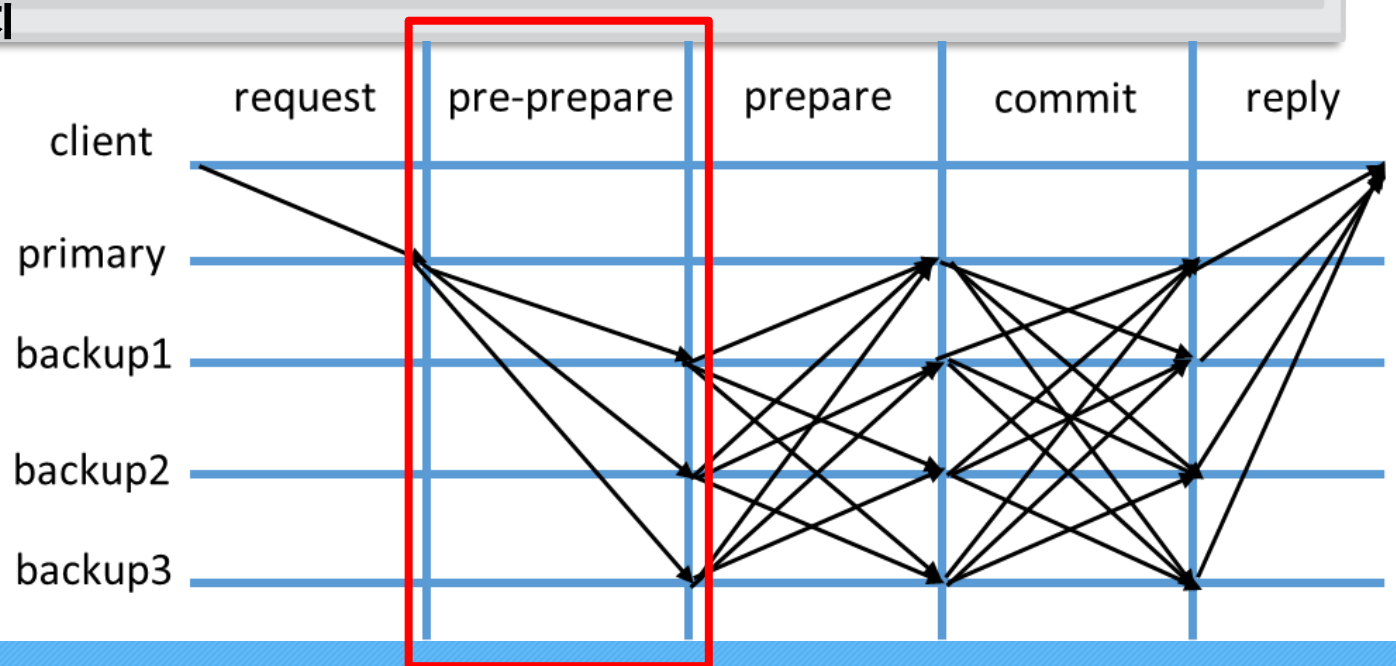
- 클라이언트는 Primary 노드에게 작업 요청
- (REQUEST, o, t, c)s_c
  - REQUEST: 합의 단계
  - o: 요청하고 싶은 작업
  - t: Timestamp
  - c: 클라이언트 식별자
  - s_c: 클라이언트 서명



PBFT(Practical Byzantine Fault Tolerance)

Pre-prepare

- Primary가 유효한 request를 받으면 아래 메시지를 생성 후 backup 노드에게 전송
  - Backup 노드: Primary와 Client를 제외한 모든 노드
  - ((PRE-PREPARE,v,n,d)s_p, m)
    - ✓ PRE-PREPARE: 합의 단계
    - ✓ v: view number(현재 Primary 노드가 누구인지 알 수 있음)
    - ✓ n: sequence number(몇 번째 request인지 알 수 있음)
    - ✓ d: m의 message digest. m이 조작되지 않았음을 확인
    - ✓ s_p: Primary 노드의 서명
    - ✓ m: 클라이언트가 보낸 메시지



PBFT(Practical Byzantine Fault Tolerance)

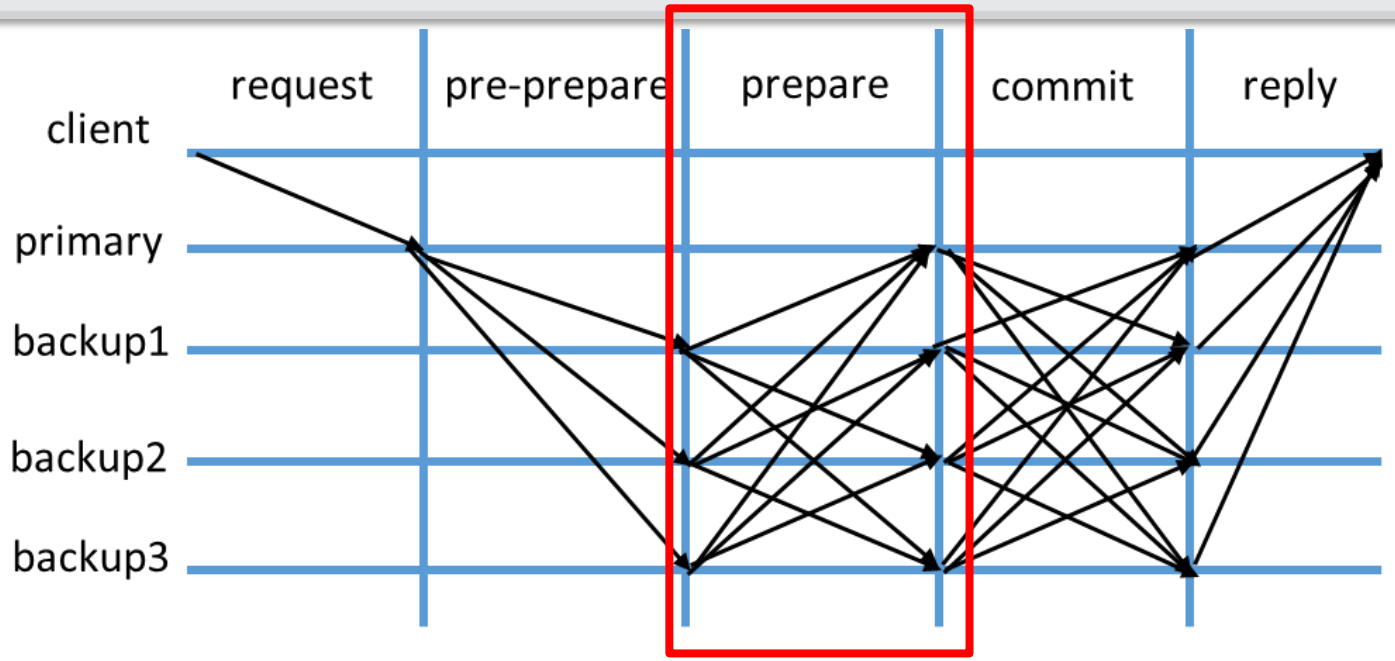
Pre-prepare

- backup 노드는 아래 조건을 만족하면 다음 단계로 넘어감
  - view number (Primary node), sequence number (트랜잭션 순서), d (메시지 무결성)
  - 체크포인트가 유효한지
  - 다른 다이제스트가 포함된 view number와 sequence number를 수신하지 않았는지

PBFT(Practical Byzantine Fault Tolerance)

Prepare

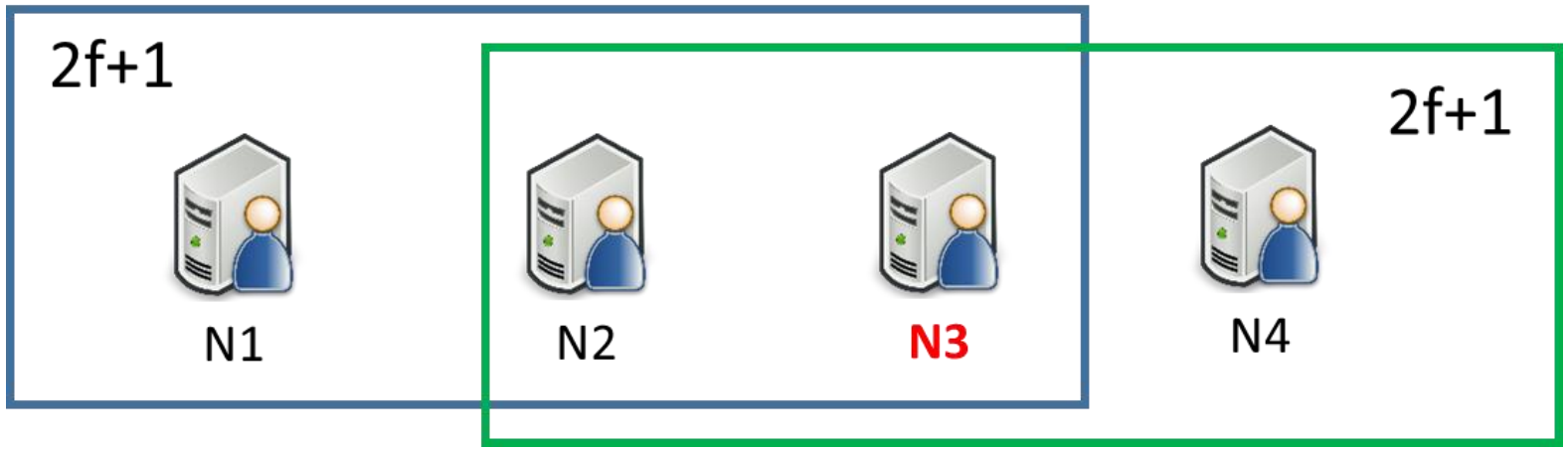
- Backup 노드는 아래 메시지를 생성하여 모든 노드에게 전송
  - ((PREPARE,v,n,d, i)s_i)
    - ✓ PREPARE: 합의 단계
    - ✓ i: 메시지를 전송하는 노드 i
    - ✓ s_i: 노드i의 서명
  - pre-prepare와 prepare를 메시지를 2f+1(including itself) 개 이상 수집한 노드는 prepared(m,v,n,i) 상태가 됨



PBFT(Practical Byzantine Fault Tolerance)

Prepare

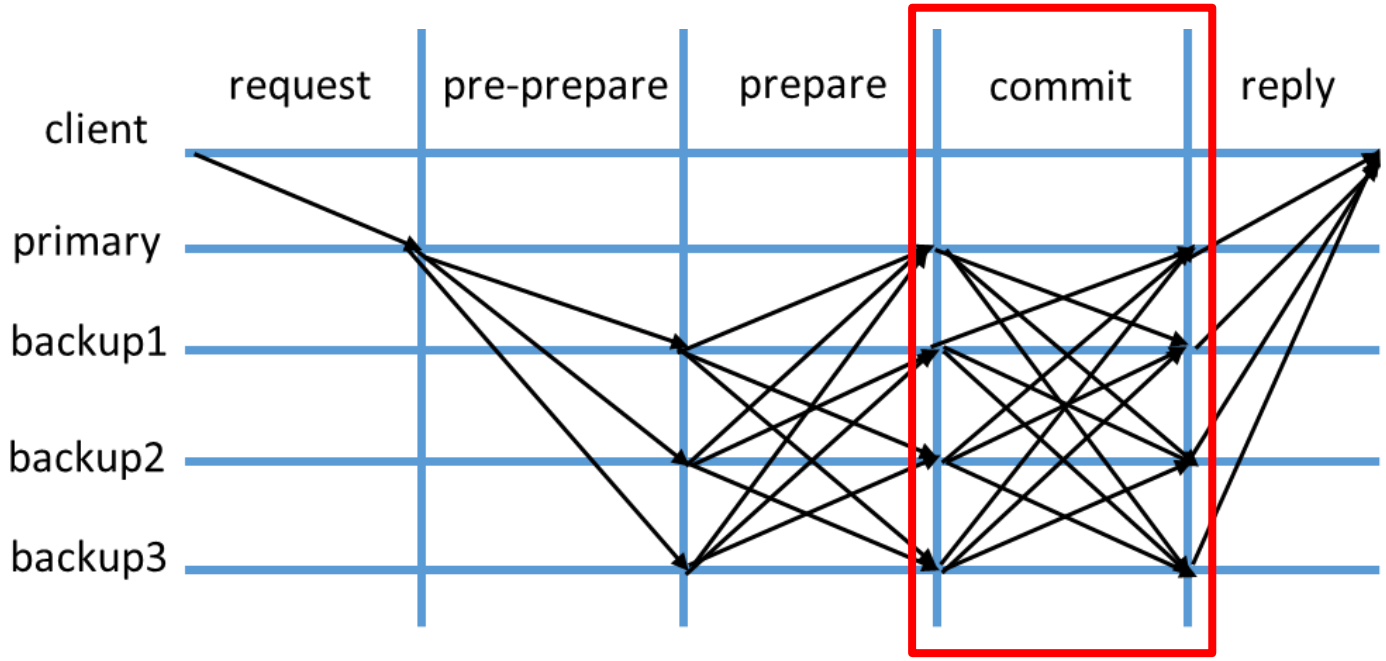
- Backup 노드는 아래 메시지를 생성하여 모든 노드에게 전송
  - ((PREPARE,v,n,d, i)s_i)
    - ✓ PREPARE: 합의 단계
    - ✓ i: 메시지를 전송하는 노드 i
    - ✓ s_i: 노드i의 서명
  - pre-prepare와 prepare를 메시지를 2f+1 개 이상 수집한 노드는 prepared(m,v,n,i) 상태가 됨



PBFT(Practical Byzantine Fault Tolerance)

Commit

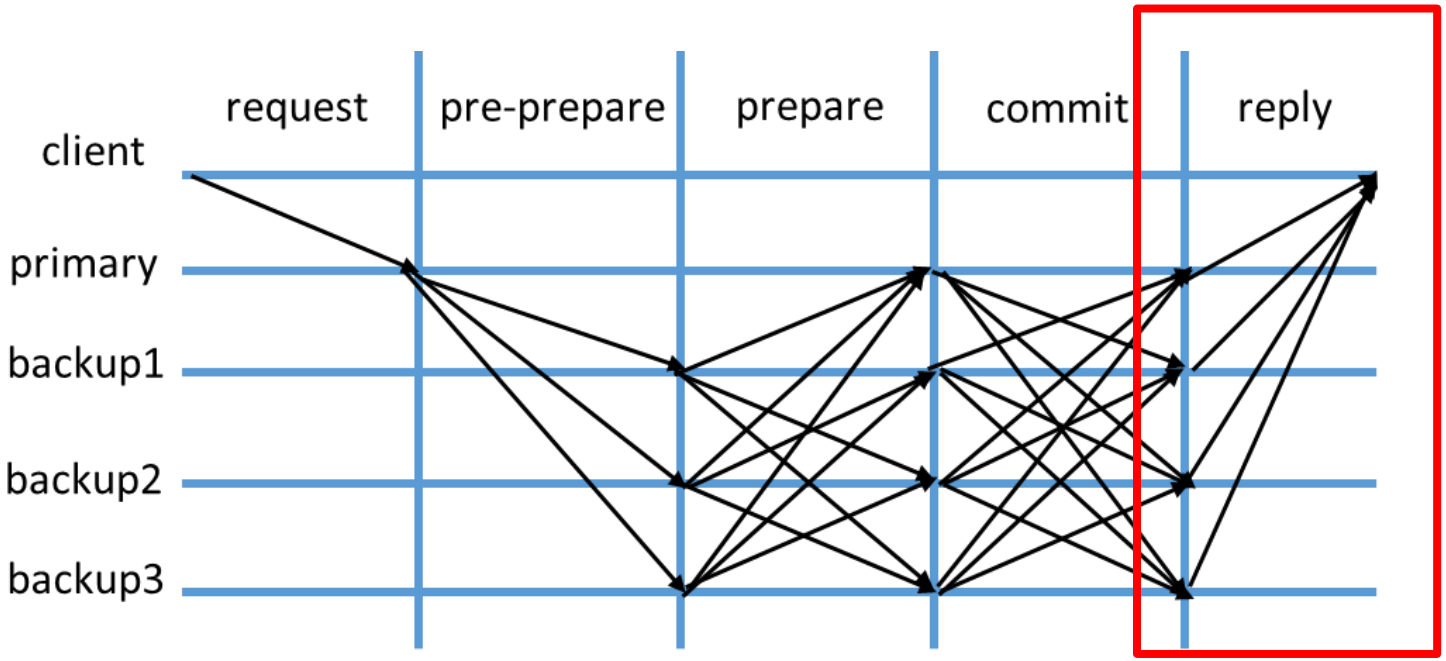
- Backup 노드는 prepared(m,v,n,i) 단계에 있는 노드가 f + 1개(non-faulty nodes) 인것을 확인하면 Committed(m, v, n) 상태가 되어 아래 메시지를 전송
  - ((COMMIT,v,n,d, i)s_i)



- ✓ v: view number(현재 Primary 노드가 누구인지 알 수 있음)
- ✓ n: sequence number
- ✓ m: 클라이언트가 보낸 메시지
- ✓ d: m의 message digest. m이 조작되지 않았음을 확인
- ✓ i: 메시지를 전송하는 노드 I
- ✓ s_i: 노드i의 서명

PBFT: Reply

- 클라이언트는 backup으로 부터 f+1개 이상의 동일한 응답을 받으면 요청이 완료됨을 확인함



PBFT(Practical Byzantine Fault Tolerance)

Checkpoint

- 모든 노드들은 Pre-prepare, Prepare, Commit 메시지를 log 저장소에 저장
- 지속적으로 저장되는 메시지 log 용량 문제를 해결하기 위해 Checkpoint 사용
  - 일정한 간격으로 Checkpoint를 생성하여 Checkpoint 이전의 메시지는 모두 폐기
  - H=h+k, sequence number가 H 가 되면 Checkpoint 작업 수행

PBFT : 결론

- 노드간 투표 방식의 합의 알고리즘 제공
- 각 Phase를 거치면서 높은 지연시간 발생
- 상당한 트래픽 발생
- 결론: 소규모 노드로 구성된 네트워크에 적합한 알고리즘

Q & A