Private Blockchain & PBFT

Daegeun Yoon

KAIST, Electrical engineering department

2019/03/13

Overview

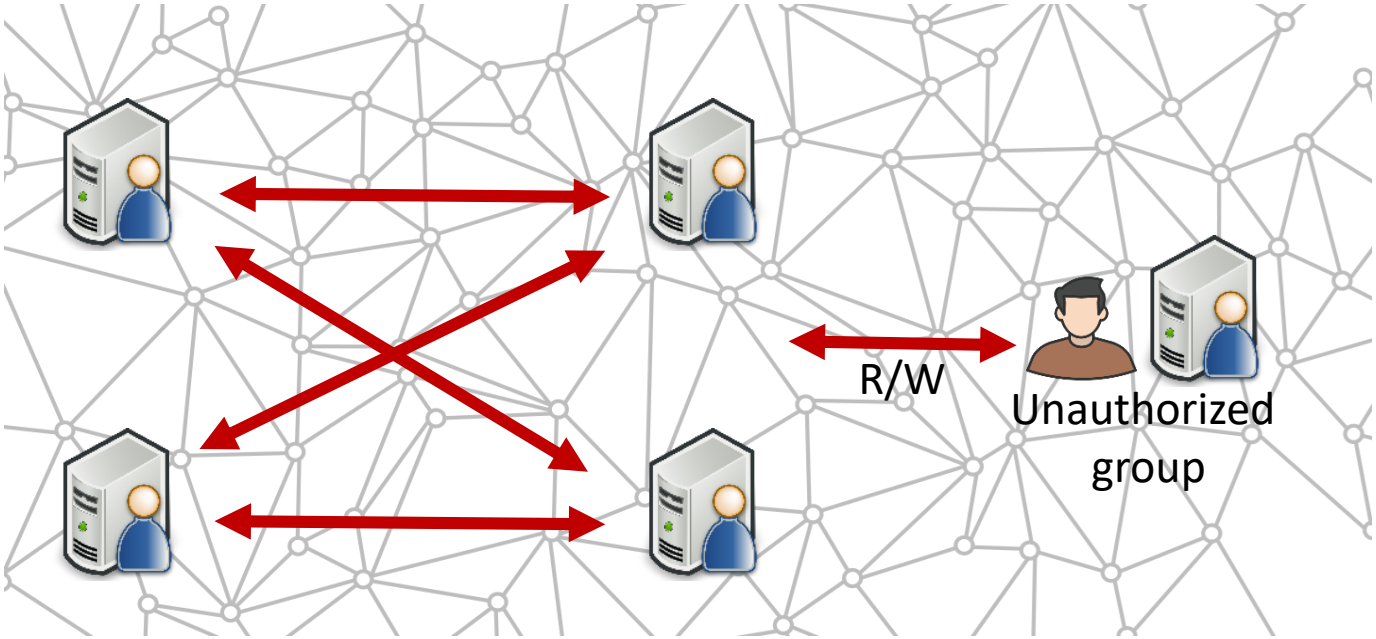
- Private Blockchain
  - Understanding Private Blockchain
  - Hyperledger Project
- PBFT
  - FLP Impossibility
  - Two Generals Problem & Byzantine Generals Problem
  - Byzantine Fault Tolerance
  - Practical Byzantine Fault Tolerance
- Hyperledger Fabric
  - Architecture
  - Transaction Flow
  - Consensus

Private Blockchain Overview

Blockchain in Enterprises

- Public Blockchain
  - ~~Low performance~~
  - ~~Visible to all~~
  - ~~Writable by all~~

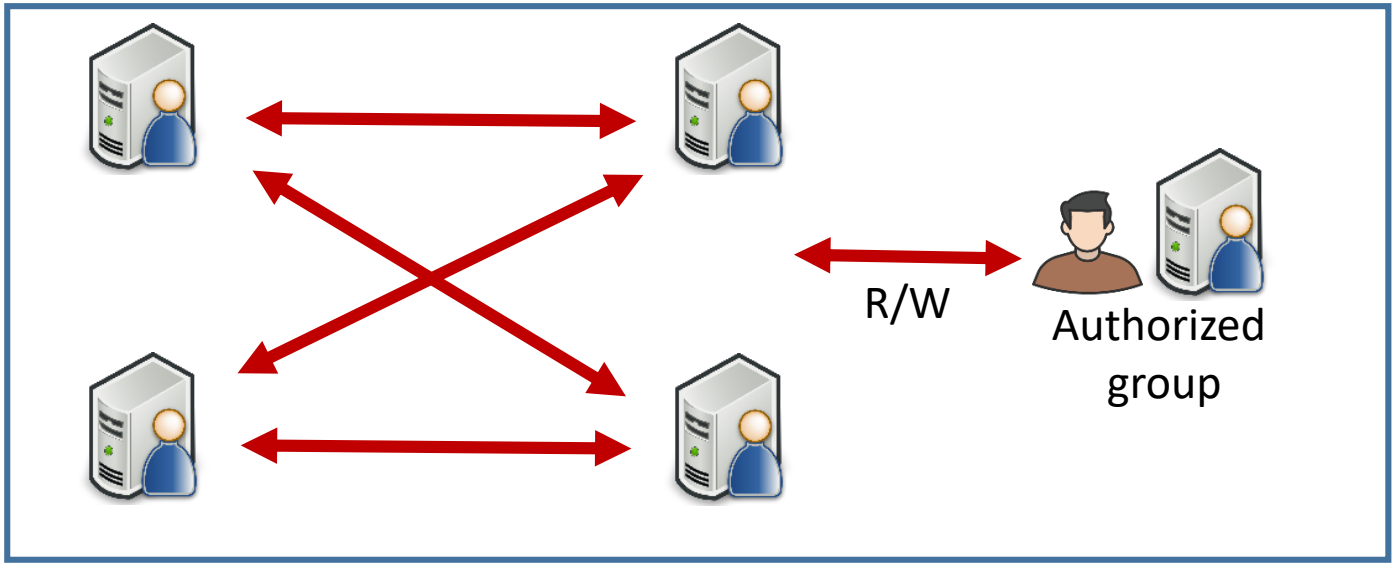
Not appropriate for enterprises purpose!



Public Blockchain Network

Blockchain In enterprises

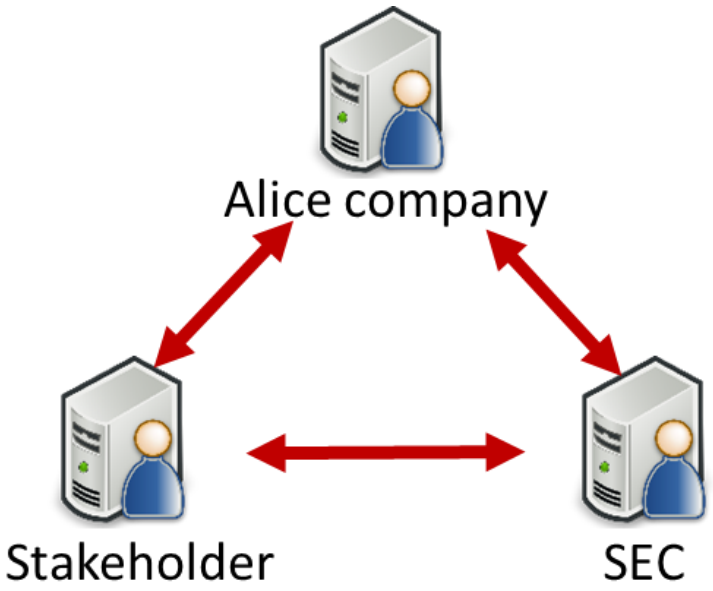
- Private Blockchain
  - Better performance
  - Centralized governance
  - Visible to authorized users
  - Writable by authorized users



Private Blockchain Network

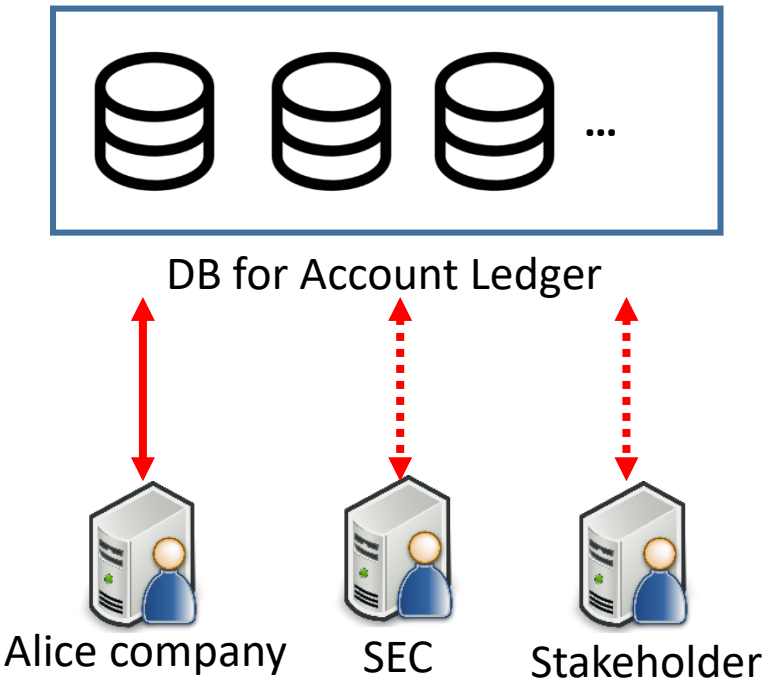
Case

- What if Alice company, the stakeholders, and SEC establish the consortium to watch over Alice company's account ledger?



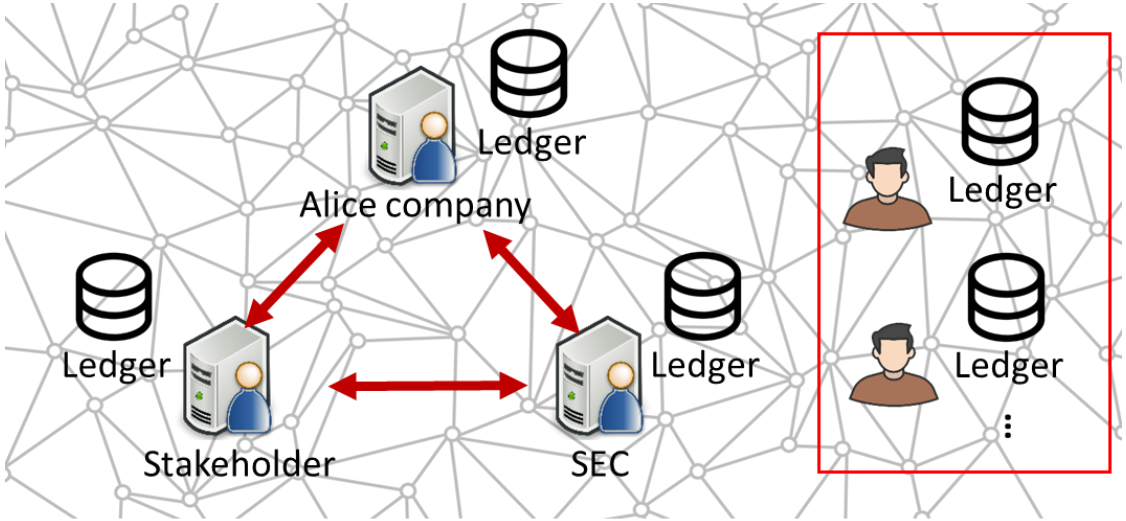
In the case of legacy database

- Hard to find any evidence of cheating
  - Alice may operate the account ledger DB
  - It will be easy to modify the account ledger



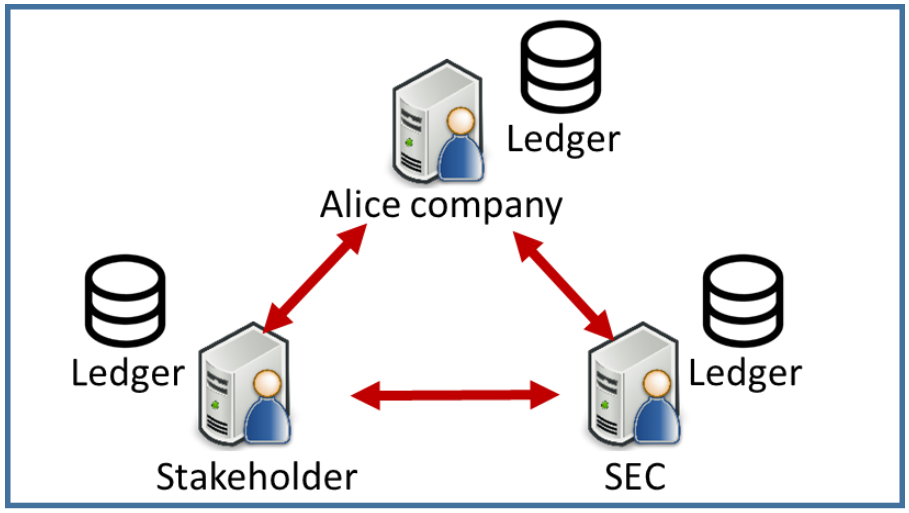
In the case of public blockchain

- Clear evidence of any changes or modifications since all data are recorded and modified under the consortium's agreement, but…
- Poor performance
- Anyone can have access to sensitive data



In the case of private blockchain

- Clear evidence of any changes or modifications since all data are recorded and modified under the consortium's agreement, and
- Better performance
- Only authorized organizations have access to the data



Private Blockchain Network

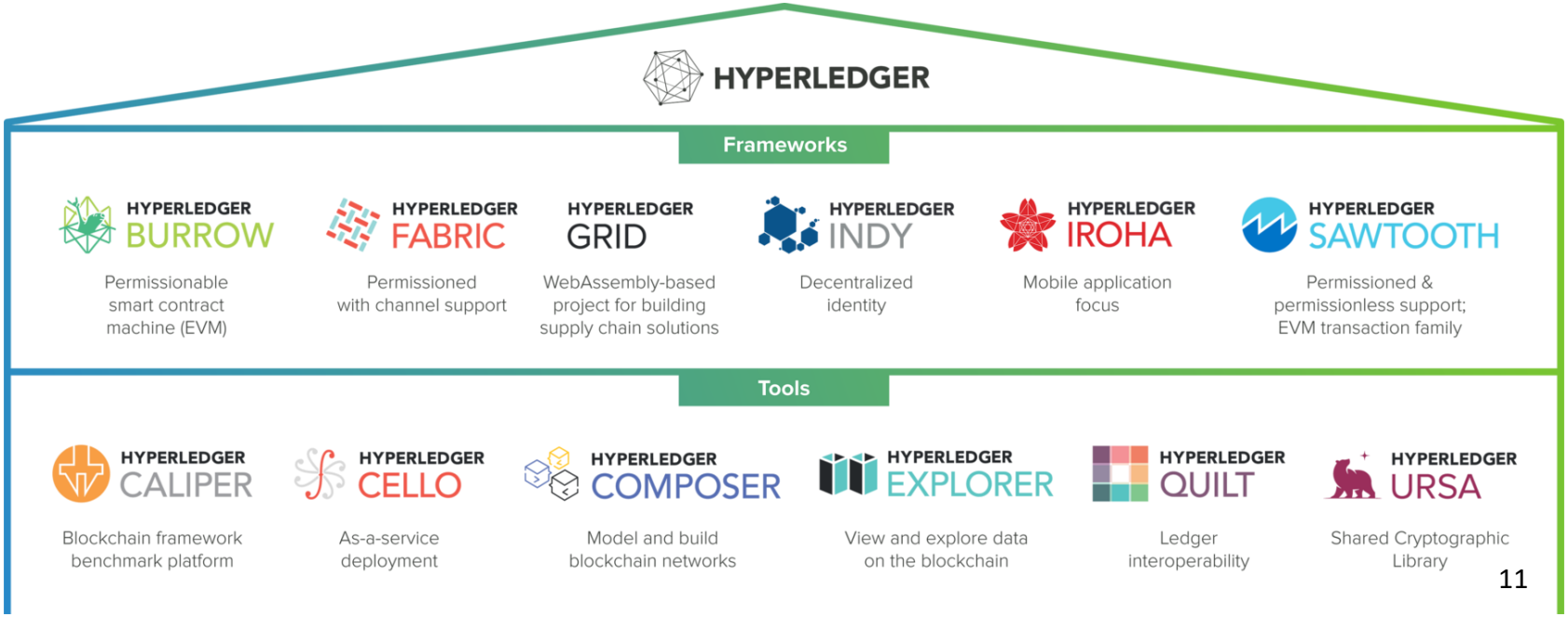
Hyperledger Project

Hyperledger Frameworks

Hyperledger Tools

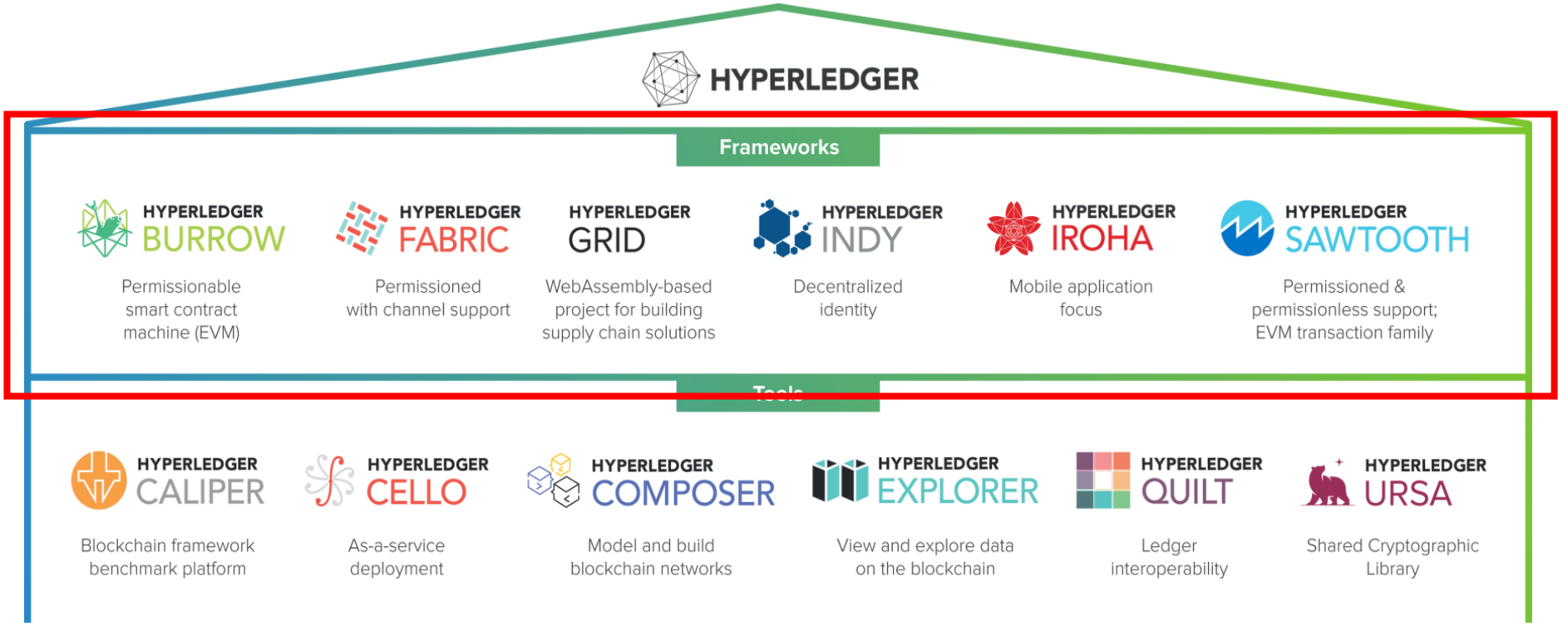
Hyperledger Project

- Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies.
- Many Hyperledger projects are actively being developed by IBM



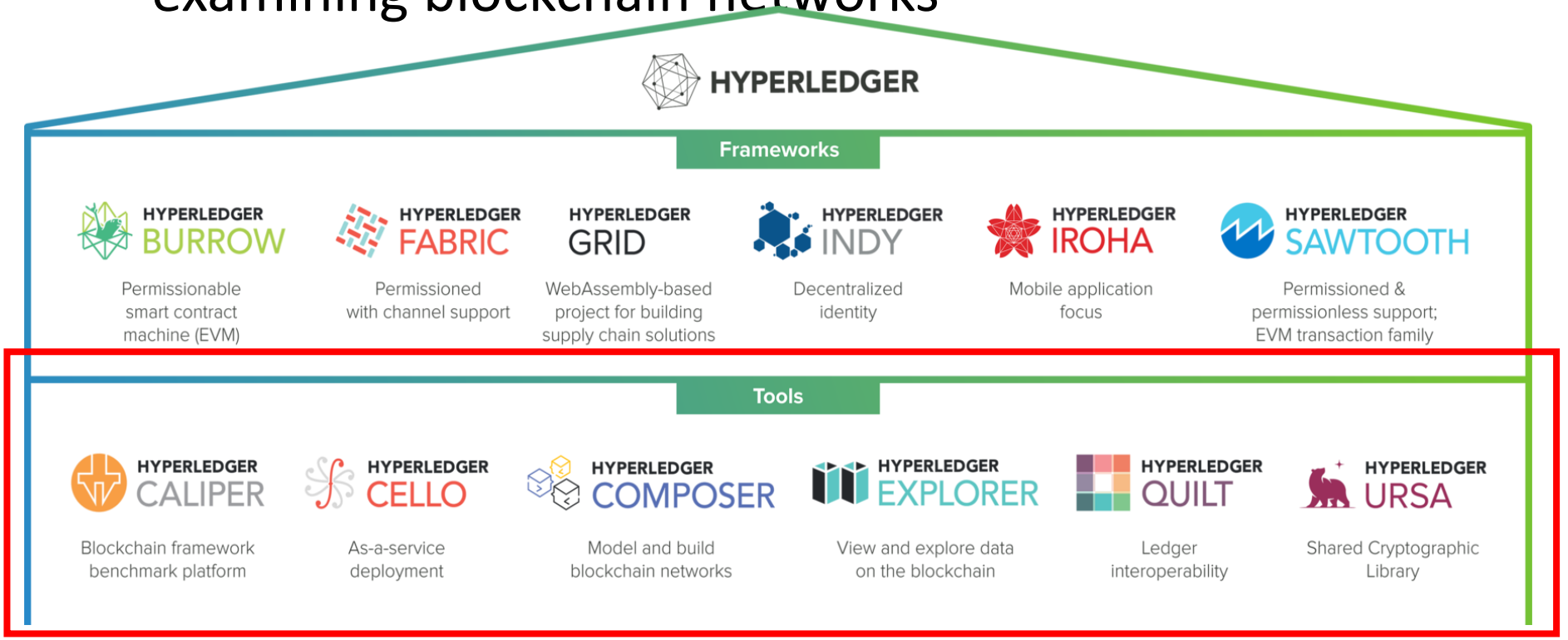
Hyperledger Frameworks

- Hyperledger Business blockchain frameworks



Hyperledger Tools

- Software used for deploying, maintaining, and examining blockchain networks



- Most projects have not reached v1.0 yet

PBFT

FLP Impossibility

Two Generals' Problem

Byzantine Generals' Problem

Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance

FLP impossibility

FLP Impossibility

- Paper
  - Impossibility of Distributed Consensus with One Faulty Process, by Fischer, Lynch, and Paterson (1985)

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 **[Computer-Communication Networks]**: Network Protocols–*protocol architecture*; C.2.4 **[Computer-Communication Networks]**: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 **[Performance of Systems]**: Reliability, Availability, and Serviceability; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation–*parallelism*; H.2.4 **[Database Management]**: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

FLP Impossibility

- Validity (Safety)
  - The value agreed upon must have been proposed by some
- Agreement (Safety)
  - All deciding processes agree on the same value
- Termination (Liveness)
  - At least one non-faulty process eventually decides

- Distribute consensus is impossible when at least one process might fail
  - Choose at most two!

FLP Impossibility

- Liveness over Safety
  - ex) Proof of Work (e.g. Bitcoin, Ethereum, etc.)
  - Lottery-based algorithm
  - The longest chain rule (Liveness ↑)
  - The longest chain can be wrong (Safety ↓)

- Safety over Liveness
  - PBFT (e.g. Tendermint, Hyperledger Indy, etc.)
  - Voting-based algorithm
  - Block after consensus is hard to be modified (Safety ↑)
  - Transactions may not be delivered (Liveness ↓)

Two Generals' Problem & Byzantine Generals' Problem

Two Generals' Problem

- "Some Constraints and Tradeoffs in The Design of Network Communications", E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber (1975)

SOME CONSTRAINTS AND TRADEOFFS
IN THE DESIGN OF
NETWORK COMMUNICATIONS*

E. A. Akkoyunlu
K. Ekanadham
R. V. Huber†
Department of Computer Science
State University of New York at Stony Brook

A number of properties and features of interprocess communication systems are presented, with emphasis on those necessary or desirable in a network environment. The interactions between these features are examined, and the consequences of their inclusion in a system are explored. Of special interest are the time-out feature which forces all system table entries to "die of old age" after they have remained unused for some period of time, and the insertion property which states that it is always possible to design a process which may be invisibly inserted into the communication path between any two processes. Though not tied to any particular system, the discussion concentrates on distributed systems of sequential processes (no interrupts) with no system buffering.

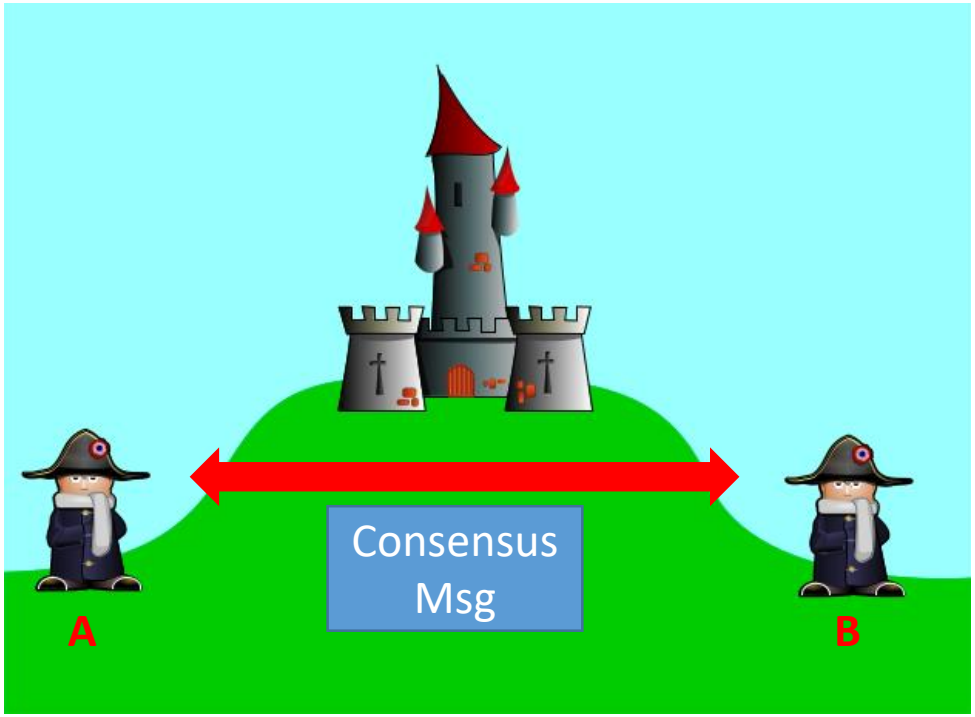
Key Words and Phrases: interprocess communication, computer networks, ports.

CR Categories: 3.81, 4.32, 4.39

Two Generals' Problem

- Two generals need to attack the enemy at the same time
  - Consensus message is sent across the enemy's territory

- A sends B the consensus msg
  - A has no way of knowing if the message was received by B
  - A has no way of knowing if the message was forged by the enemy
- B sends A the response message
  - B also has no way of knowing if the message was received by A
  - B also has no way of knowing if the message was forged by the Enemy
- No way to reach consensus between A and B



BGP (Byzantine Generals Problem)

- "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.

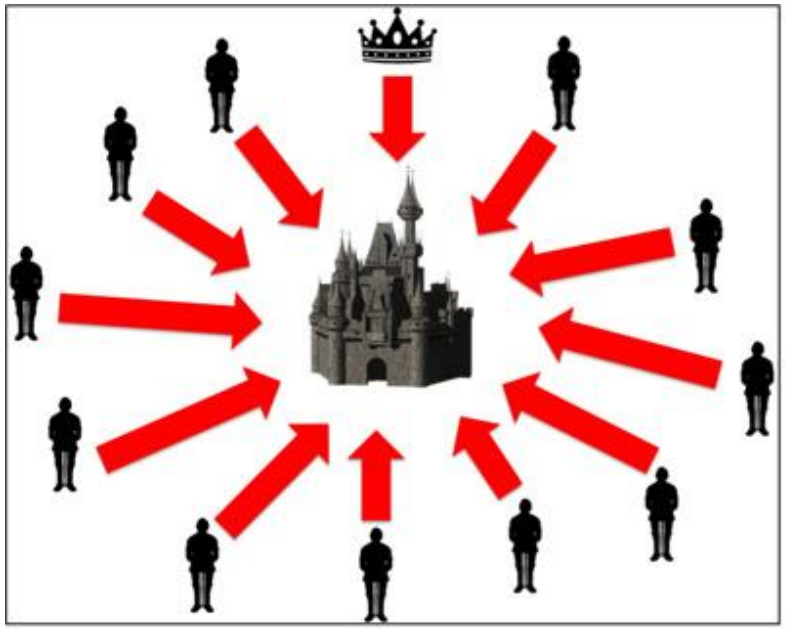
Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

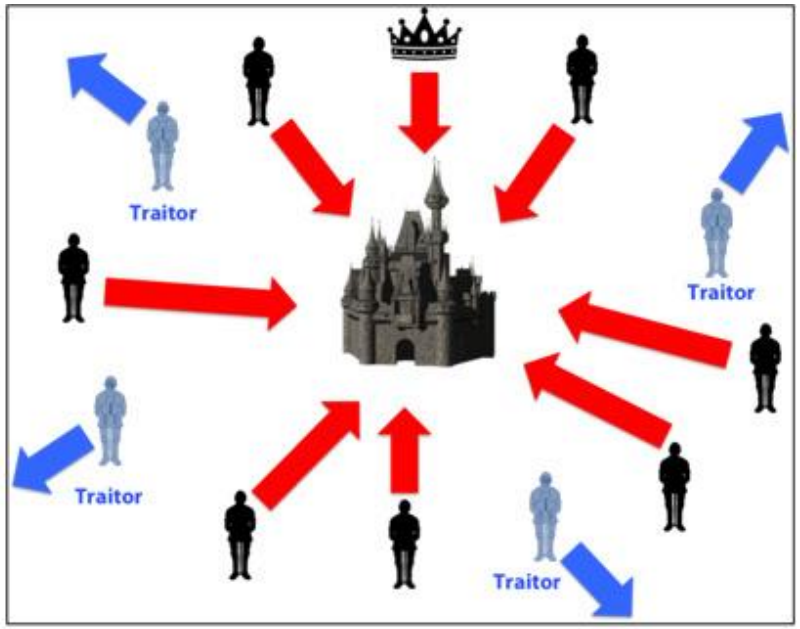
Additional Key Words and Phrases: Interactive consistency

BGP (Byzantine Generals Problem)

- More than two generals need to attack the enemy at the same time
  - Same problems as the Two Generals' Problems



Coordinated Attack Leading to Victory



Uncoordinated Attack Leading to Defeat

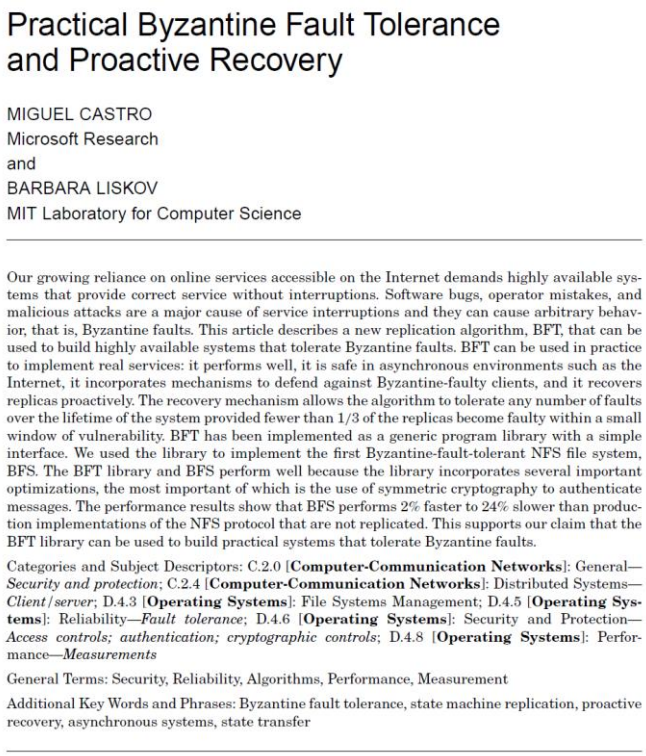
Practical Byzantine Fault Tolerance

BFT (Byzantine Fault Tolerance)

- Byzantine Fault
  - May represent general attack on the system or system error
- Byzantine Failure
  - The loss of a system service due to Byzantine Fault
- Byzantine Fault Tolerance
  - A system that is resilient/tolerant of a Byzantine Fault

PBFT (Practical Byzantine Fault Tolerance)

- Paper
  - "Practical Byzantine Fault Tolerance and Proactive Recovery“, Castro, M.; Liskov, B. (2002). ACM Transactions on Computer Systems.



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

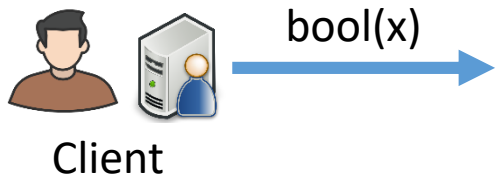
Additional Key Words and Phrases: Byzantine fault tolerance, state machine replication, proactive recovery, asynchronous systems, state transfer

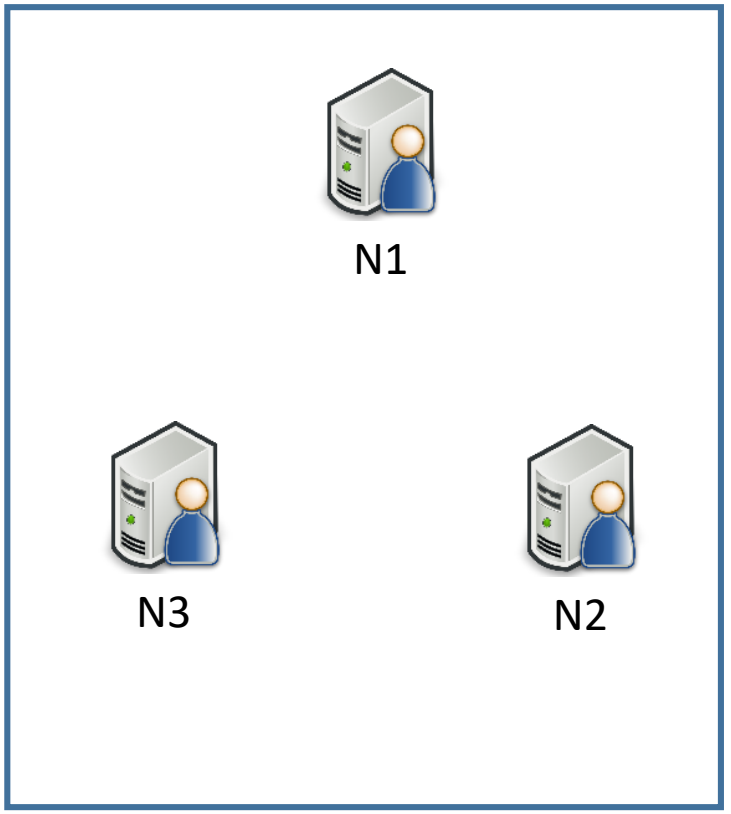
PBFT (Practical Byzantine Fault Tolerance)

- System Model
  - Allows asynchronous network
  - Possible faults
    - failure to deliver messages
    - delayed messages
    - delivery out of order
    - Byzantine faults
  - Node failure is independent
  - Assume eventual time bounds for liveness
  - N = 3f+1, N = # of nodes in the network, f = # of Faulty nodes

Why N = 3f+1?

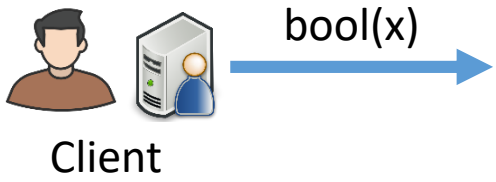
- Assume N = 2f+1, f = 1
- Client is waiting for f+1 responses

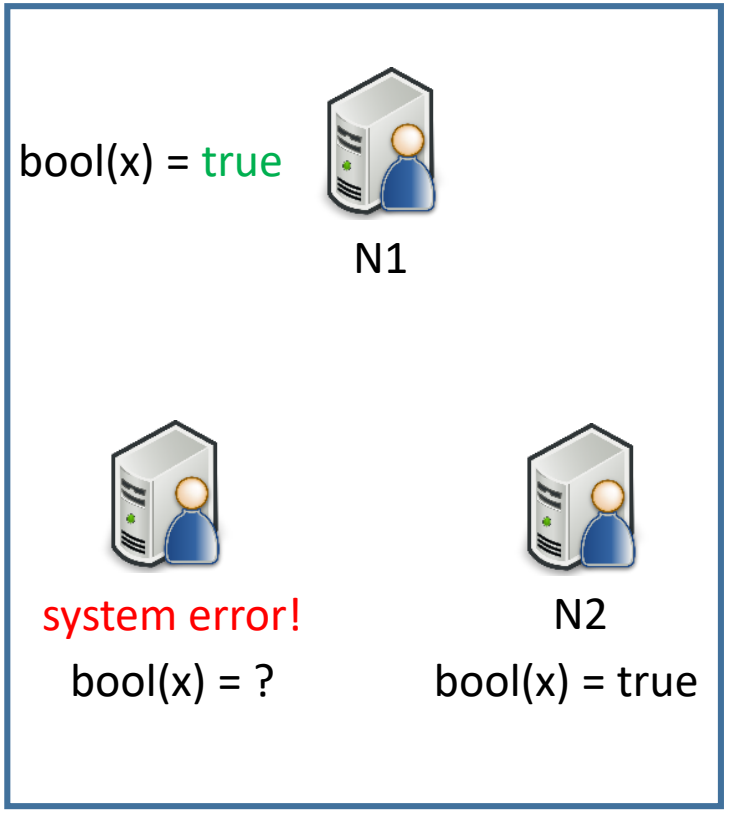




Why N = 3f+1?

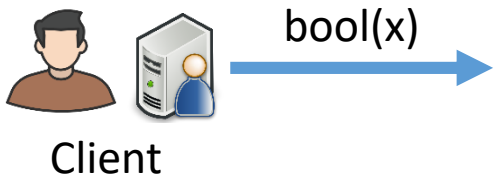
- Assume N = 2f+1, f = 1
- Client is waiting for f+1 responses

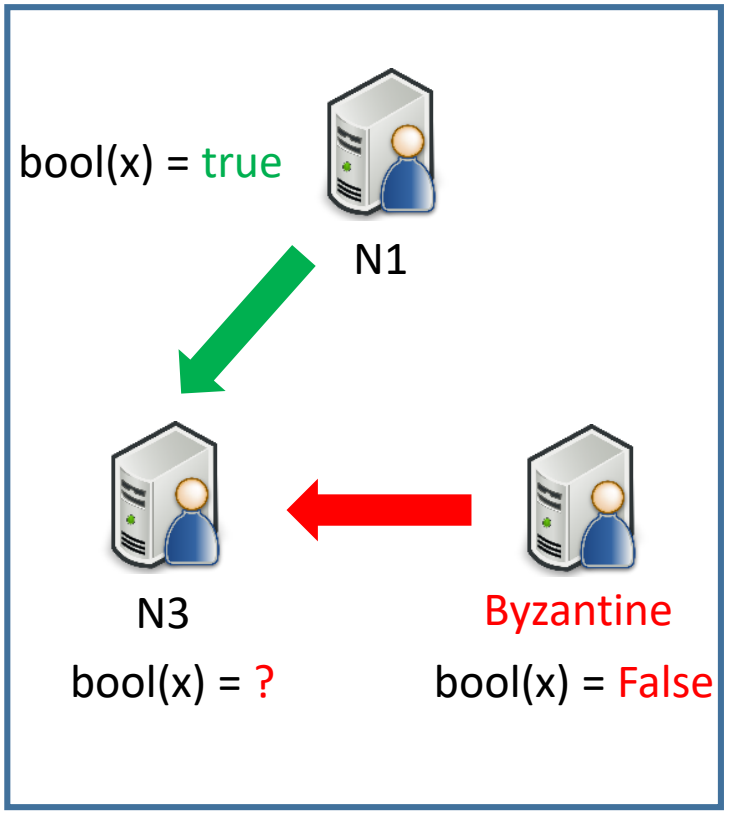




Why N = 3f+1?

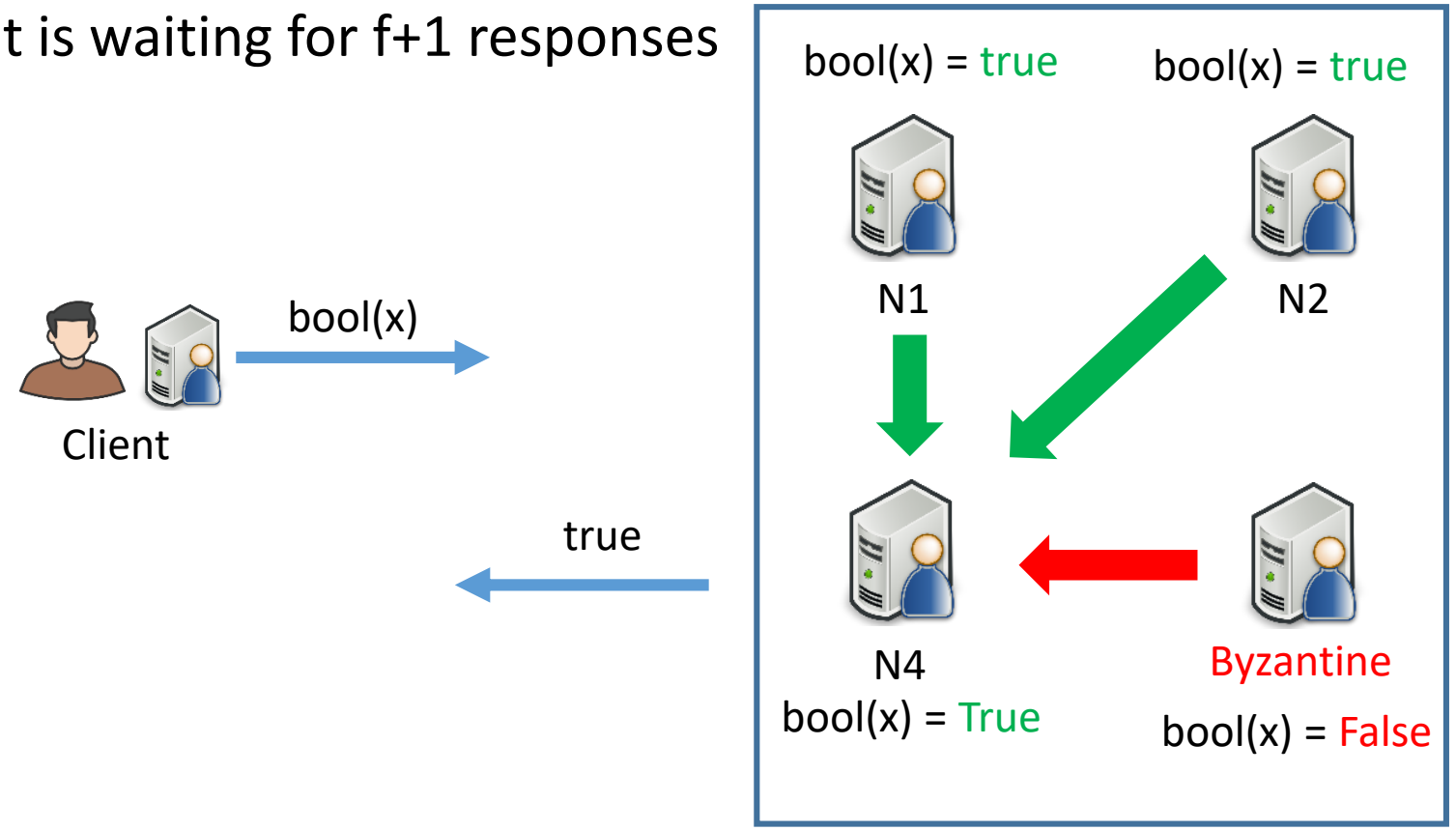
- Assume N = 2f+1, f = 1
- Client is waiting for f+1 responses





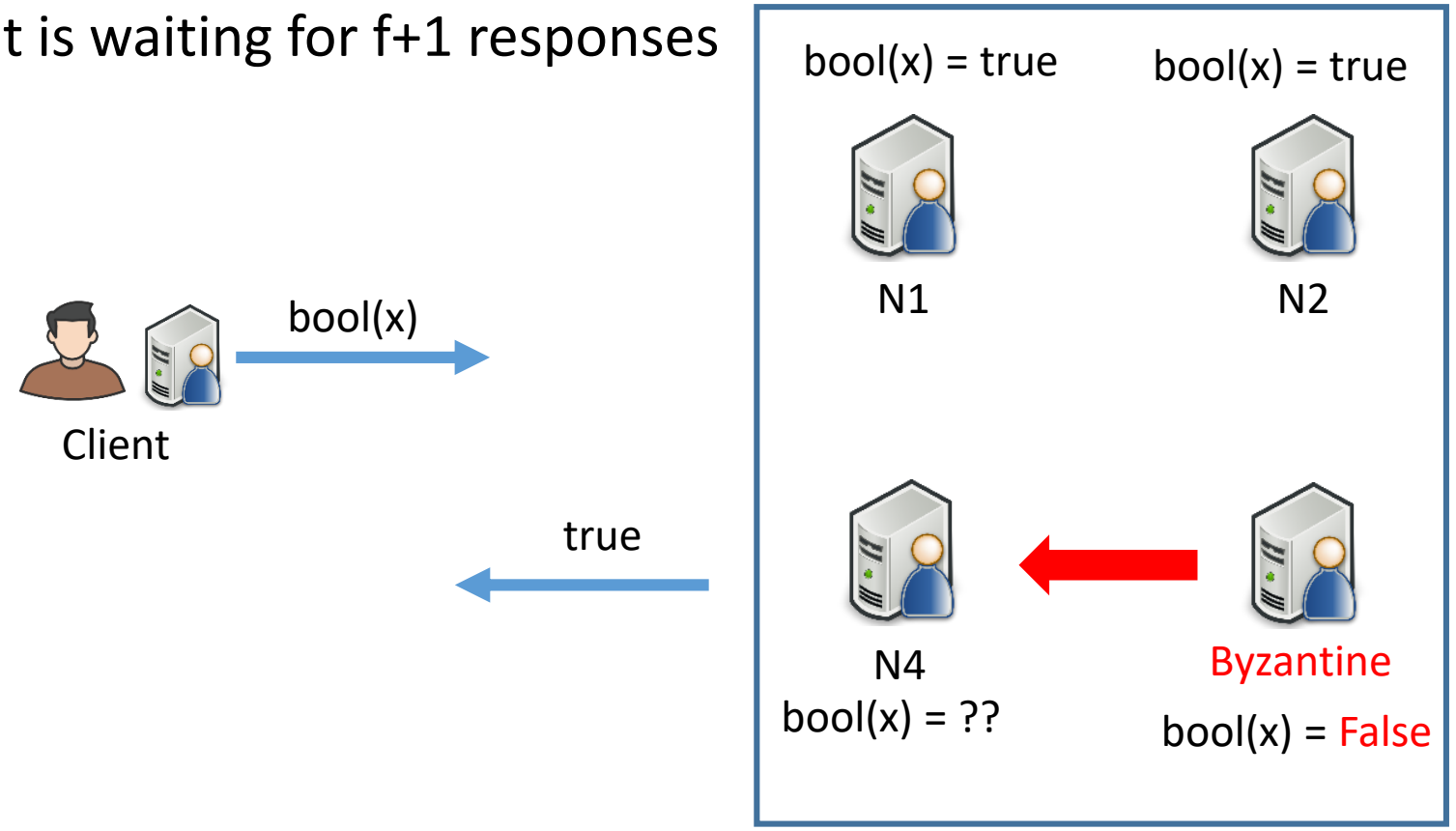
Why N = 3f+1?

- Assume N = 3f+1, f = 1
- Client is waiting for f+1 responses



Why N = 3f+1?

- Assume N = 3f+1, f = 1
- Client is waiting for f+1 responses

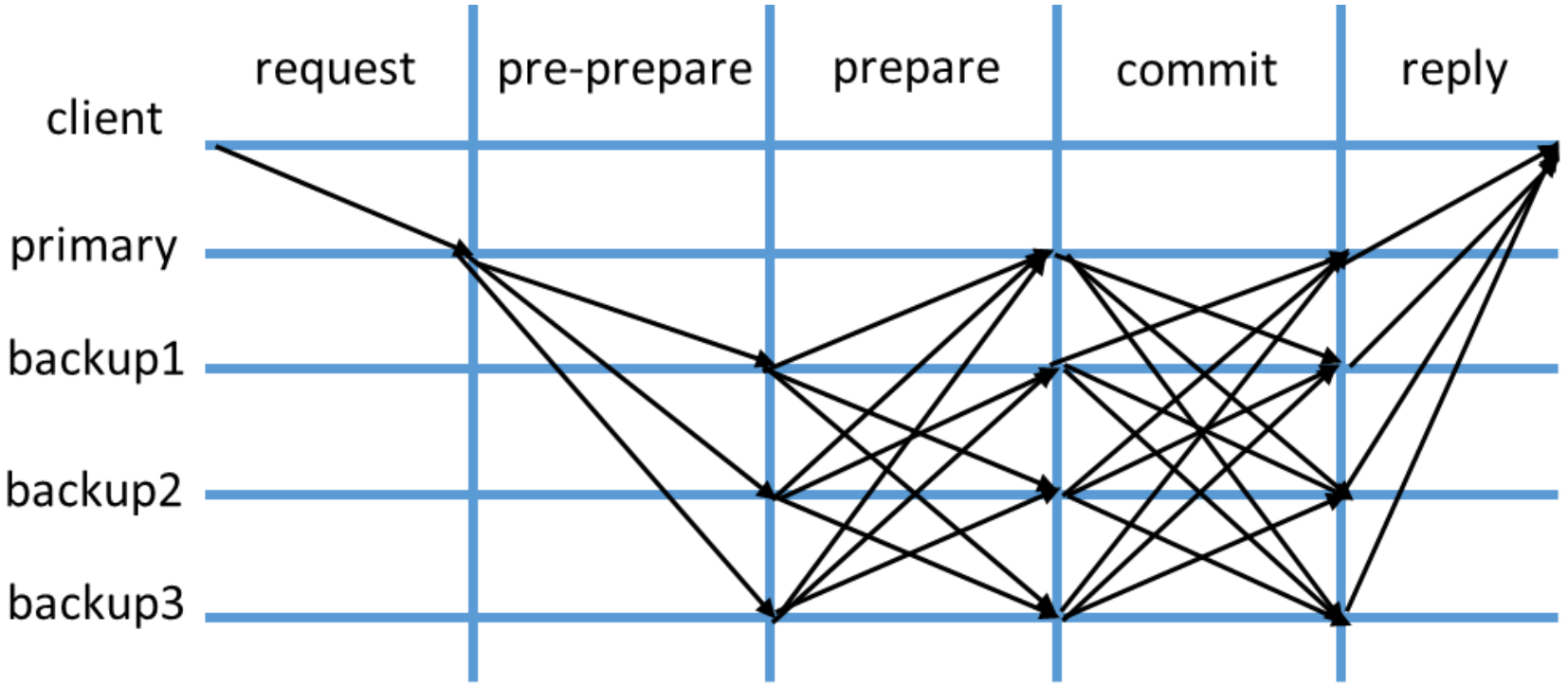


Why N= 3f + 1?

- When the message is not sent
  - Consensus should be reached among (N-f)
    - f = node whose message is not sent

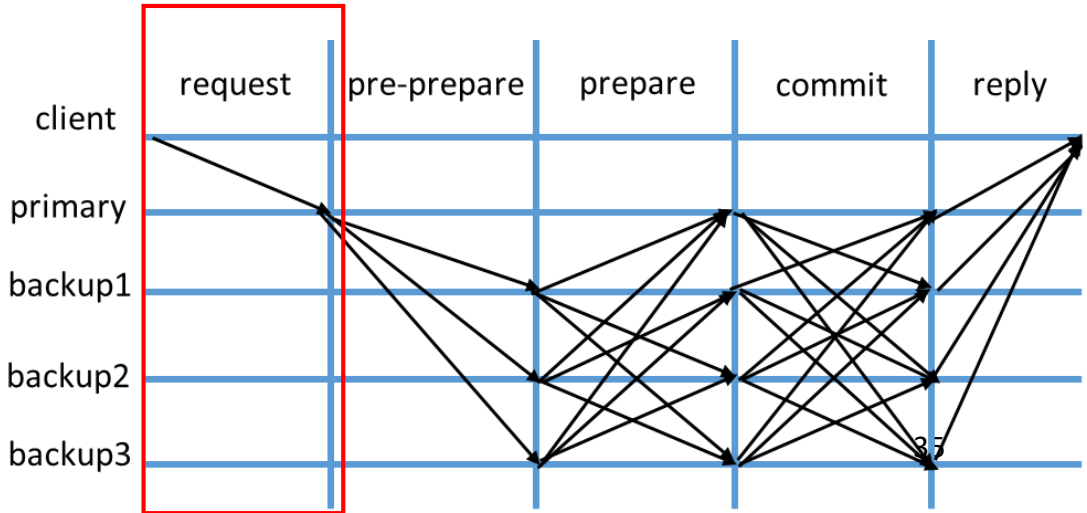
- When the message is sent with wrong information
  - Consensus can be reached only when N-f-f > f
    - f = node whose message has wrong information
  - N>3f
  - Minimum requirement of N=3f+1

PBFT in rough



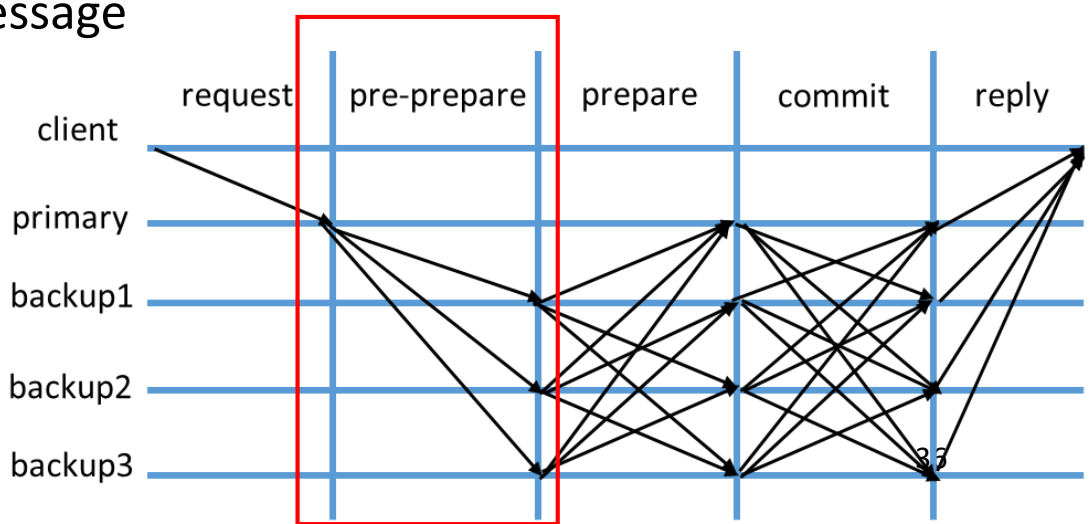
Request

- A client sends a service request to the primary
- (Request, o, t, c)s_c
  - o: requested operation
  - t: timestamp
  - c: client identity
  - (message)s_c: message signed by c



Phase 1: Pre-prepare

- The primary assigns a unique sequence number and multicasts this message to all backups
- (PRE-PREPARE, v, n, d)s_p, m)
  - v: view number
  - n: sequence number
  - d: m's digest
  - m: client's requested message

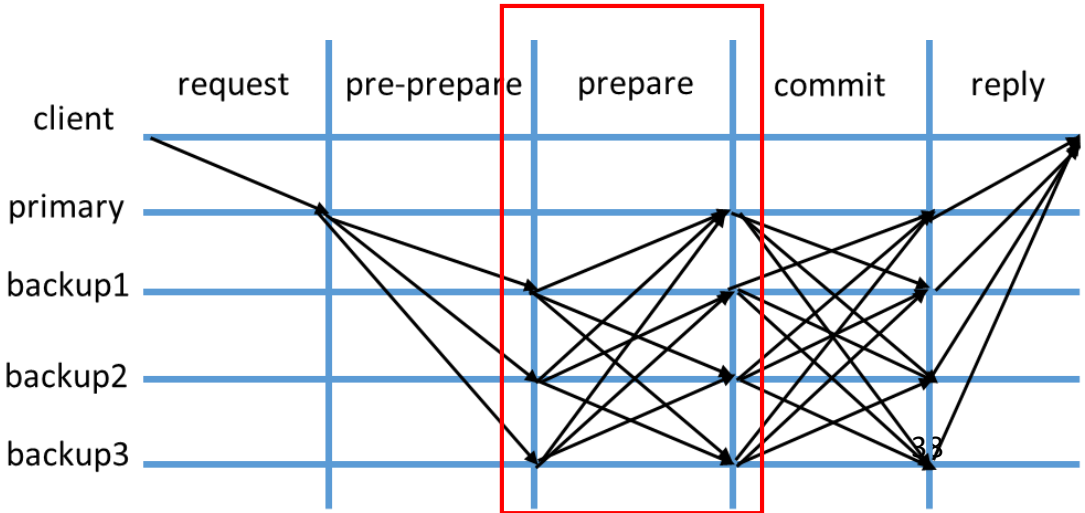


Phase 1: Pre-prepare

- A backup will accept the message iff
  - v, n, d are valid
  - n is between h and H, h = lower bound, H = upper bound
  - digest(m) is different from digest of other messages

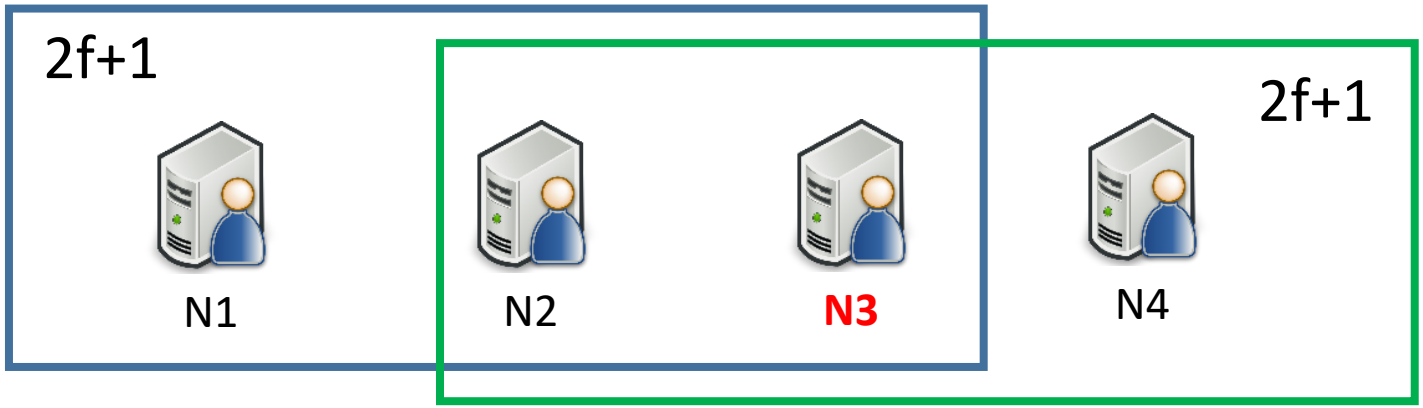
Phase 2: Prepare

- If backup i accepts PRE-PREPARE message, it enters the prepare phase by multicasting PREPARE message to all other backups
- (PREPARE, v, n, d, i)s_i
  - i: backup number



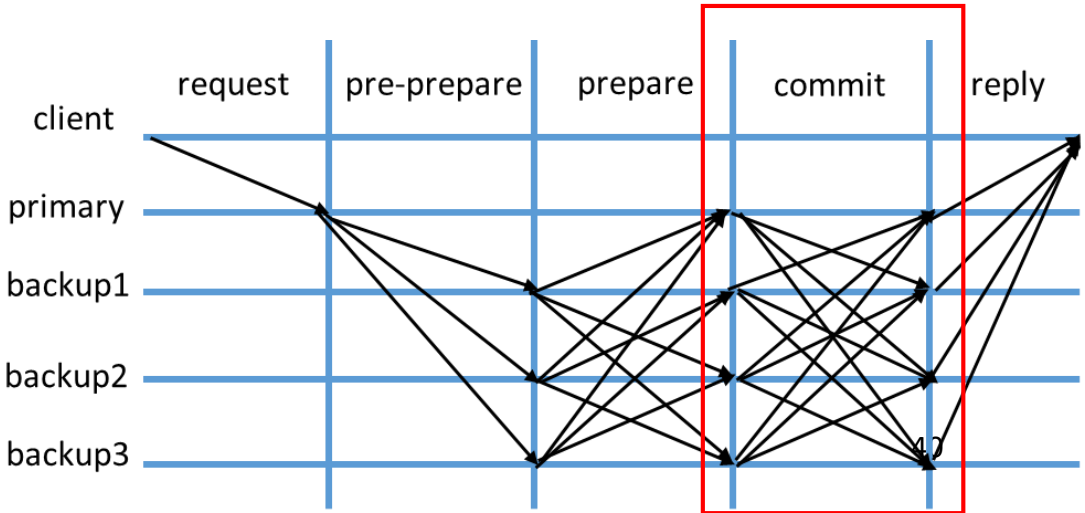
Phase 2: Prepare

- Prepared(m, v, n, i) is true iff backup i
  - PRE-PREPARE for message m has been received
  - 2f + 1 (including itself) distinct and valid PREPARE messages received



Phase 3: Commit

- Backup i multicasts a COMMIT message to the other backups when prepared(m, v, n, i) becomes true
- (COMMIT, v, n, d, i)s_i



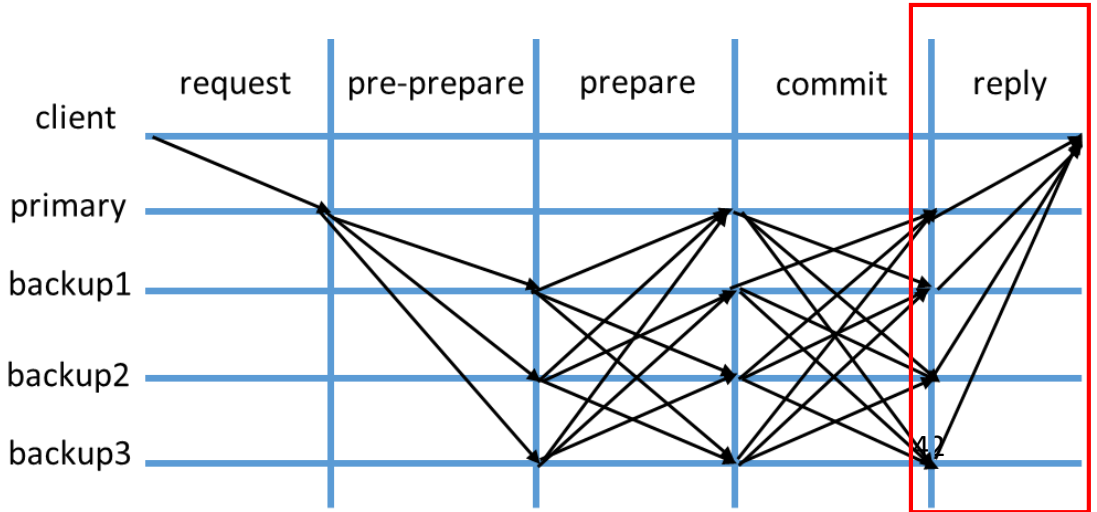
Phase 3: Commit

- committed(m, v, n) is true iff
  - prepared(m, v, n, i) is true for all i in some set of f + 1 non-faulty backups

- committed-local(m, v, n, i) is true iff
  - prepared(m, v, n, i) is true
  - i has accepted 2f + 1 commits (including itself) from different backups

Reply

- Each backup i executes the operation requested by client
- After executing the operation, backups send a reply to the client
  - (REPLY, v, t, c, i, r)s_i
    - r: execution result
- Client waits for f+1 replies

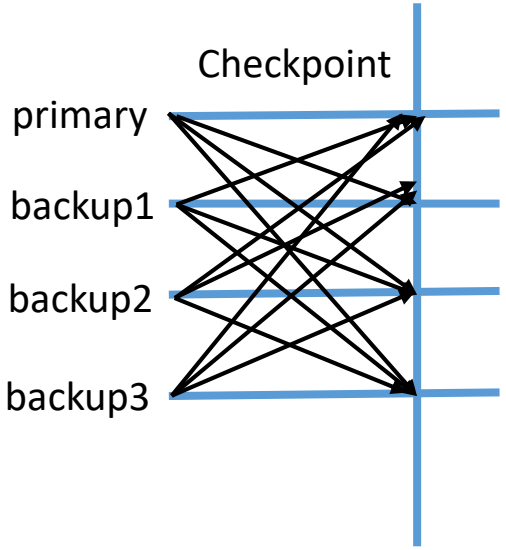


Checkpoint

- Every node saves a log of Pre-prepare, Prepare, Commit in their storage
  - **Reason**: nodes may miss some messages
  - **Problem**: limited storage size
  - **Solution**: make a checkpoint and discard the message below the checkpoint

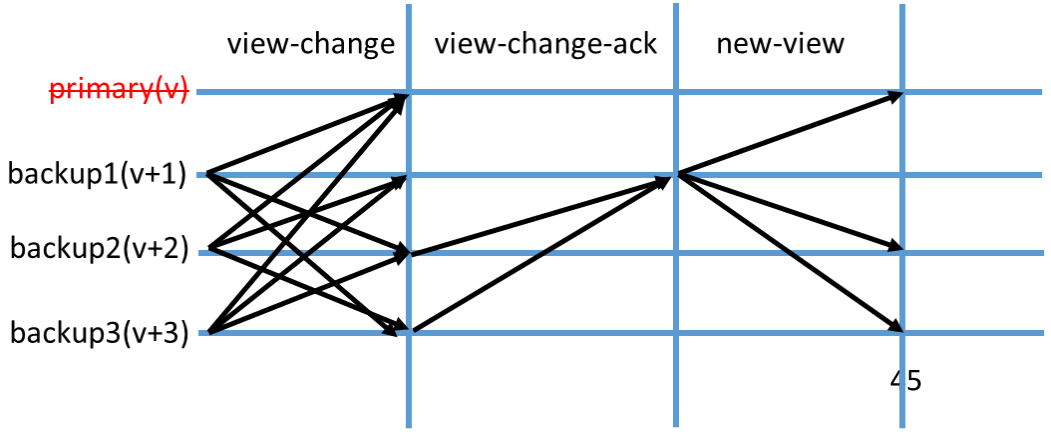
Checkpoint

- Step
  - Multicast (CHECKPOINT, n, d, i)s_i
  - Collect 2f + 1 CHECKPOINT messages
  - After completing each checkpoint, discard the messages below n
  - update the bound of sequence number
    - lower bound h = n
    - upper bound H = n + k, k = some constant k



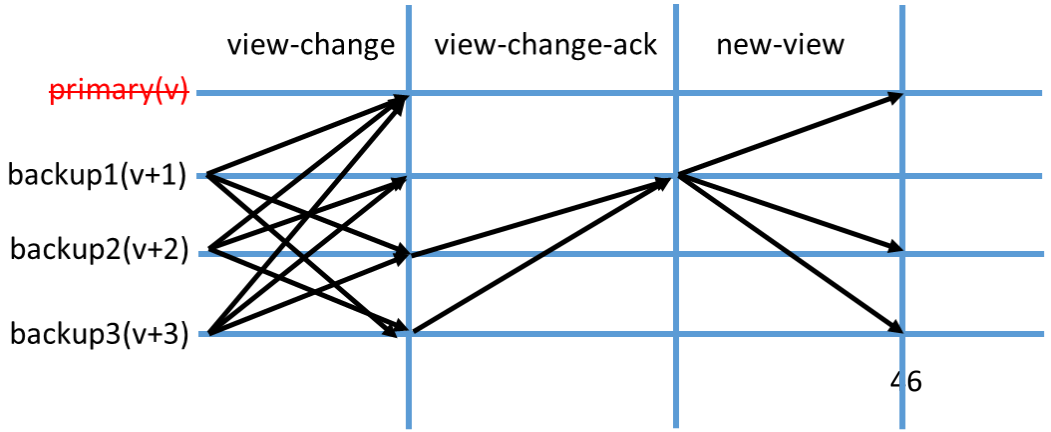
View Changes

- Backups monitor the primary to prevent faulty primaries
- Backups propose a view change when a timer expires
  - View change protocol is started if 2f+1 backups do not have a valid message from the primary v within the timer



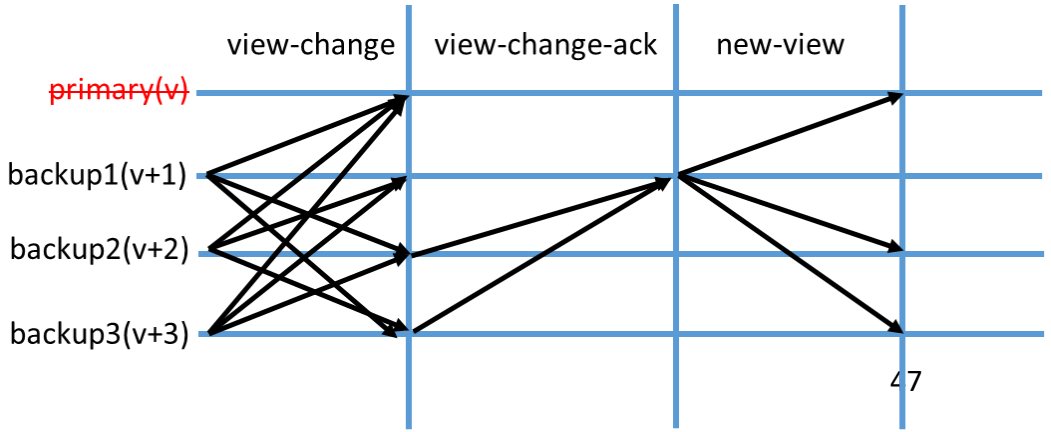
View Changes

- Multicast (VIEW-CHANGE, v+1, n, C, P, Q, i)S_i
  - n: sequence number of current checkpoint
  - C: 2f checkpoint messages
  - P: set of Pre-prepared(m, v, `n, i), `n>n, n = `n sequence number, n = current checkpoint
  - Q: set of Prepared(m, v, `n, i), `n>n, n = `n sequence number, n = current checkpoint



View Changes

- If v+1 node collects 2f+1 view-change messages, the node proposes (NEW-VIEW, v+1, V, O)s_v+1
  - V: 2f+1 VIEW-CHANGE messages
  - O: set of PRE-PREPARE messages
    - If v+1 node misses some messages, the node is able to update the messages from O



View Changes

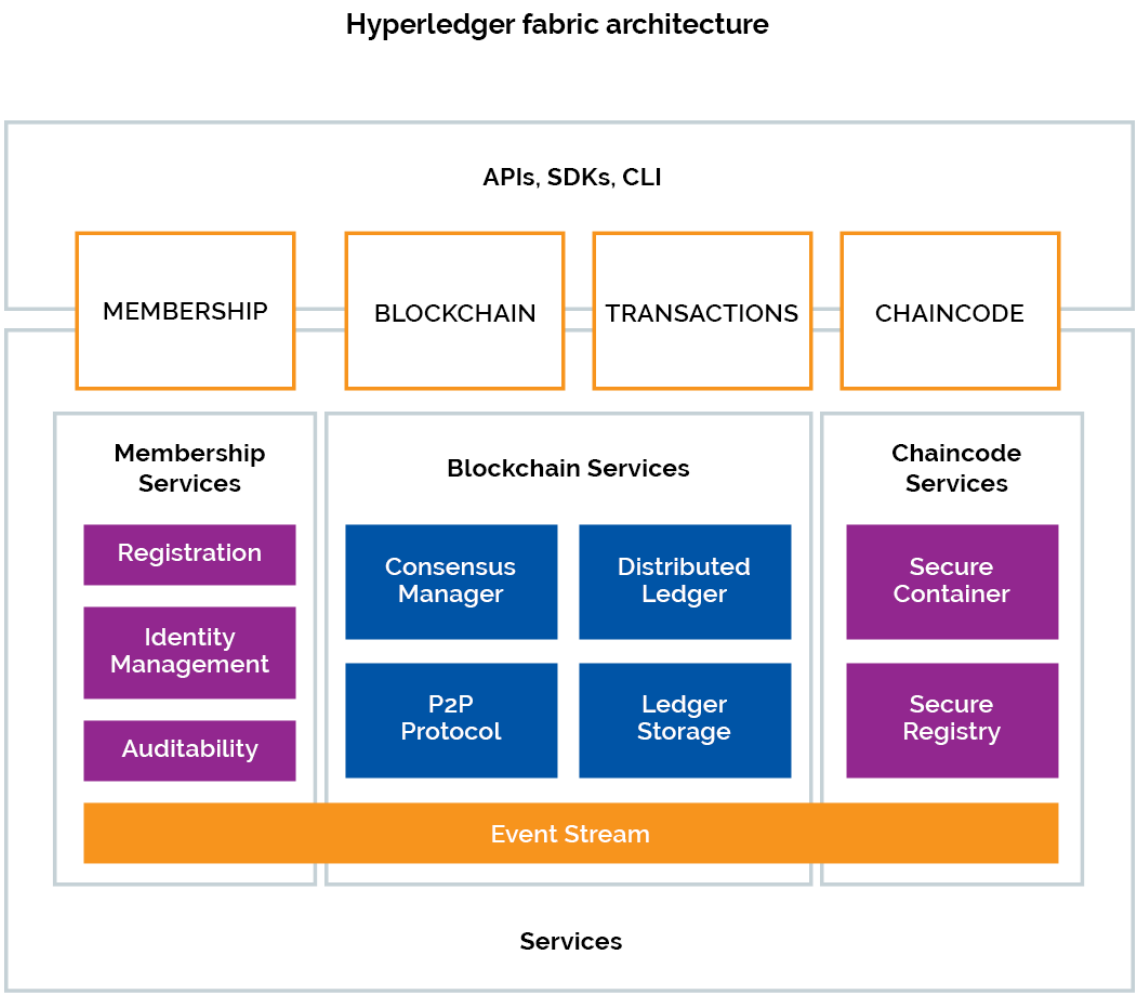
Node #	Last stable checkpoint	Sequence # in P
1(new primary)	100	190
2	200	200
3	200	200
4	200	200

Is PBFT a perfect solution?

- Of course not
  - Delay↑ since every node has to reach consensus at every phase
  - Traffic↑(O(n^2) for total traffic, O(N) for each node)
- Will it be okay, since it will be used in a private blockchain?
  - Hyperledger fabric has changed PBFT to Kafka when they updated their version from v0.6 to v1.0

Hyperledger Fabric

Architecture

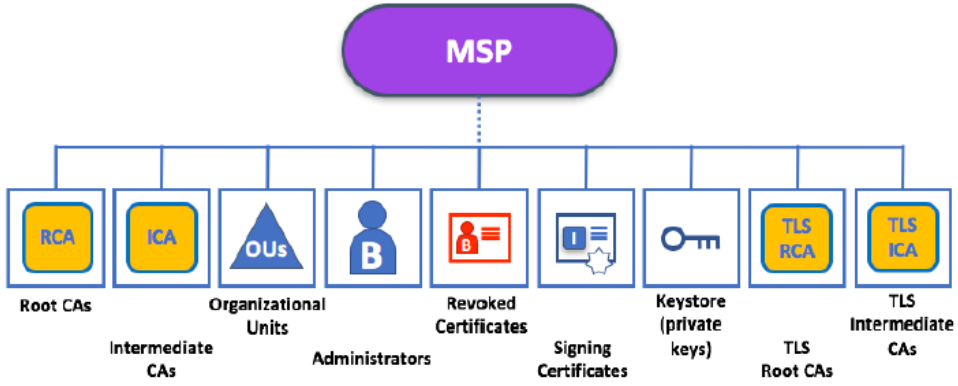


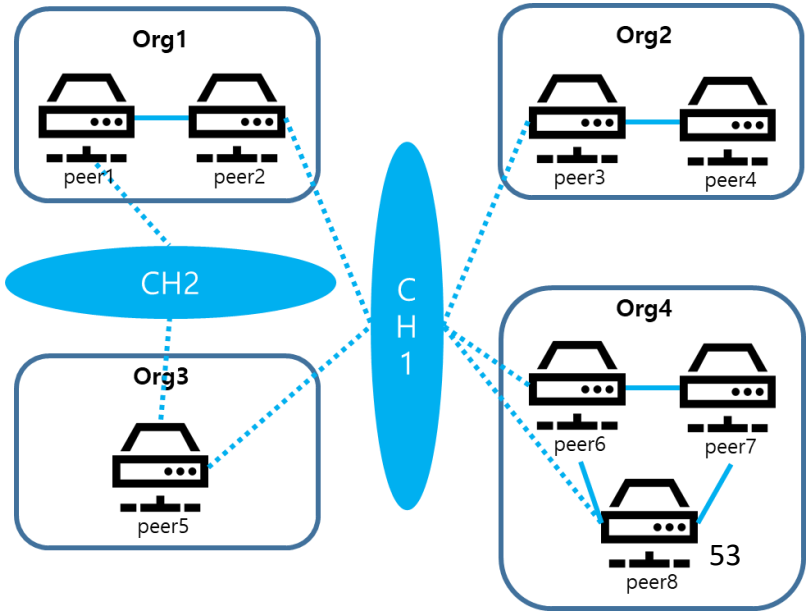
Nodes

- Committing Peer
  - Validation and commit of the block
- Endorsing Peer
  - Give transaction endorsement to the client
- Orderer
  - Get the transaction from clients and deliver to the kafka cluster
- Kafka
  - Order delivered transactions, create a new block with the transactions, and deliver the block to the orderer

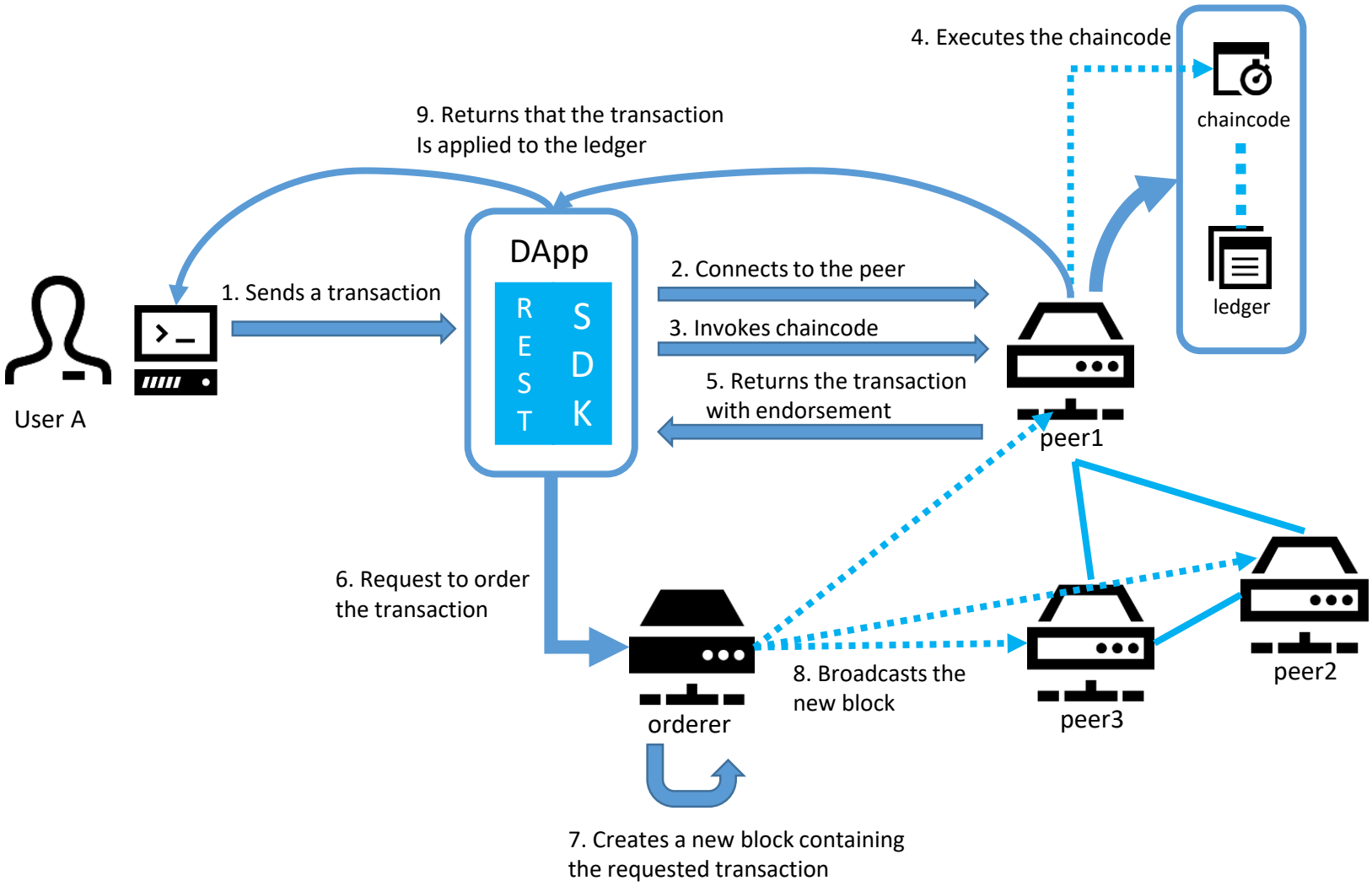
Channel & MSP

- Channel
  - Ensure privacy and confidentiality between organizations
  - No data can be shared between different channels
- MSP
  - PKI based certification management system



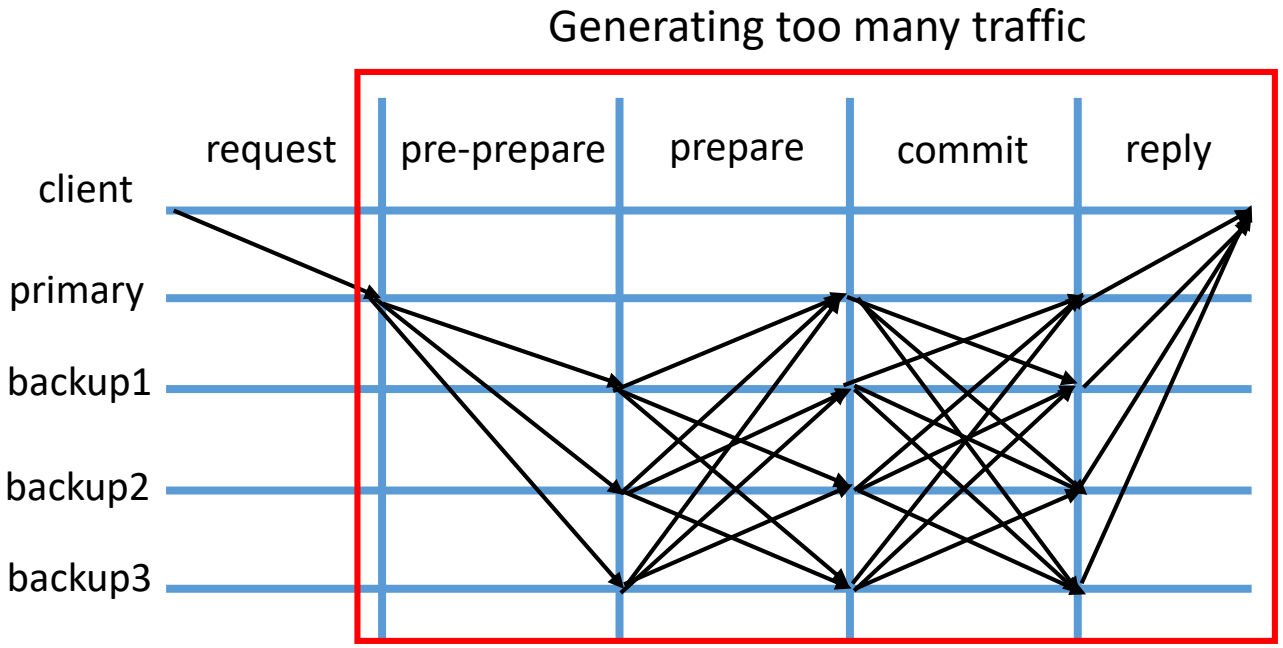


Transaction Flow



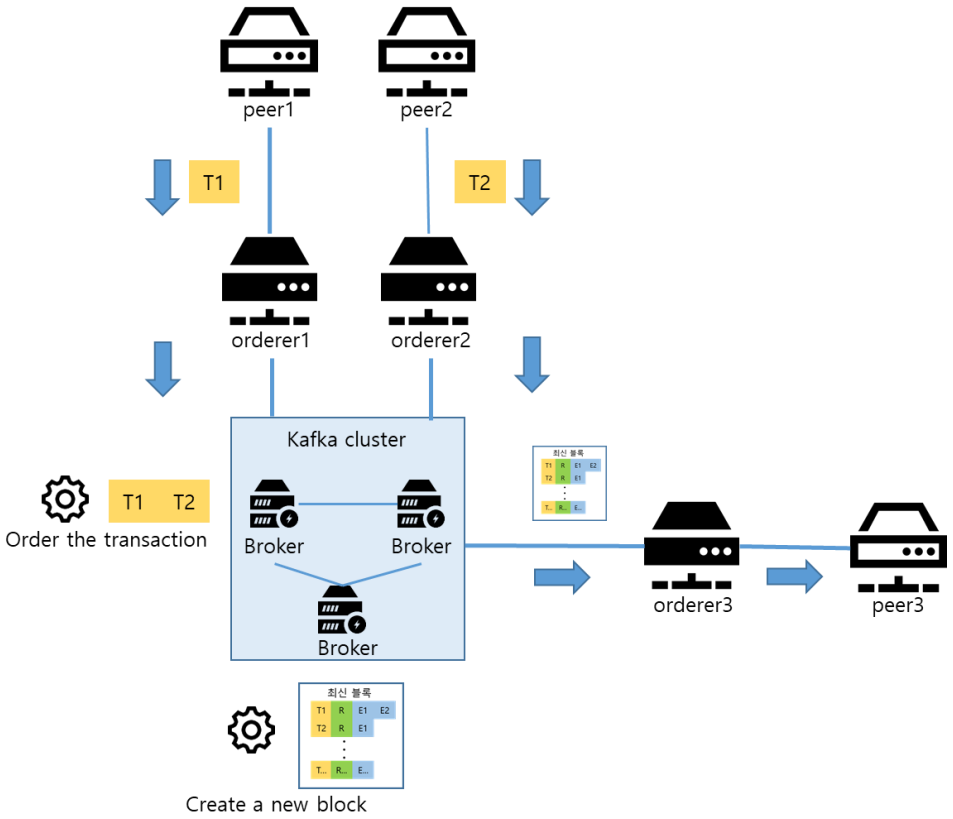
Does PBFT exist in fabric?

- No, because serious scalability problems that were encountered when fabric used PBFT in v0.6



Ordering Service

- No consensus, but ordering service
  - Kafka: provides Crash fault tolerant
  - No BFT exists in consensus algorithm



Conclusion

- The most popular private blockchain project is Hyperledger fabric
  - Other hyperledger projects have not reached v1.0 yet
- PBFT is the most popular consensus algorithm in private blockchains
  - It is useful for small scale blockchain platforms
- Hyperledger fabric has no BFT-based consensus yet
  - Serious scalability problems were encountered when fabric used PBFT in v0.6
  - There are plans to implement BFT based consensus to the upcoming version of fabric

Thank you