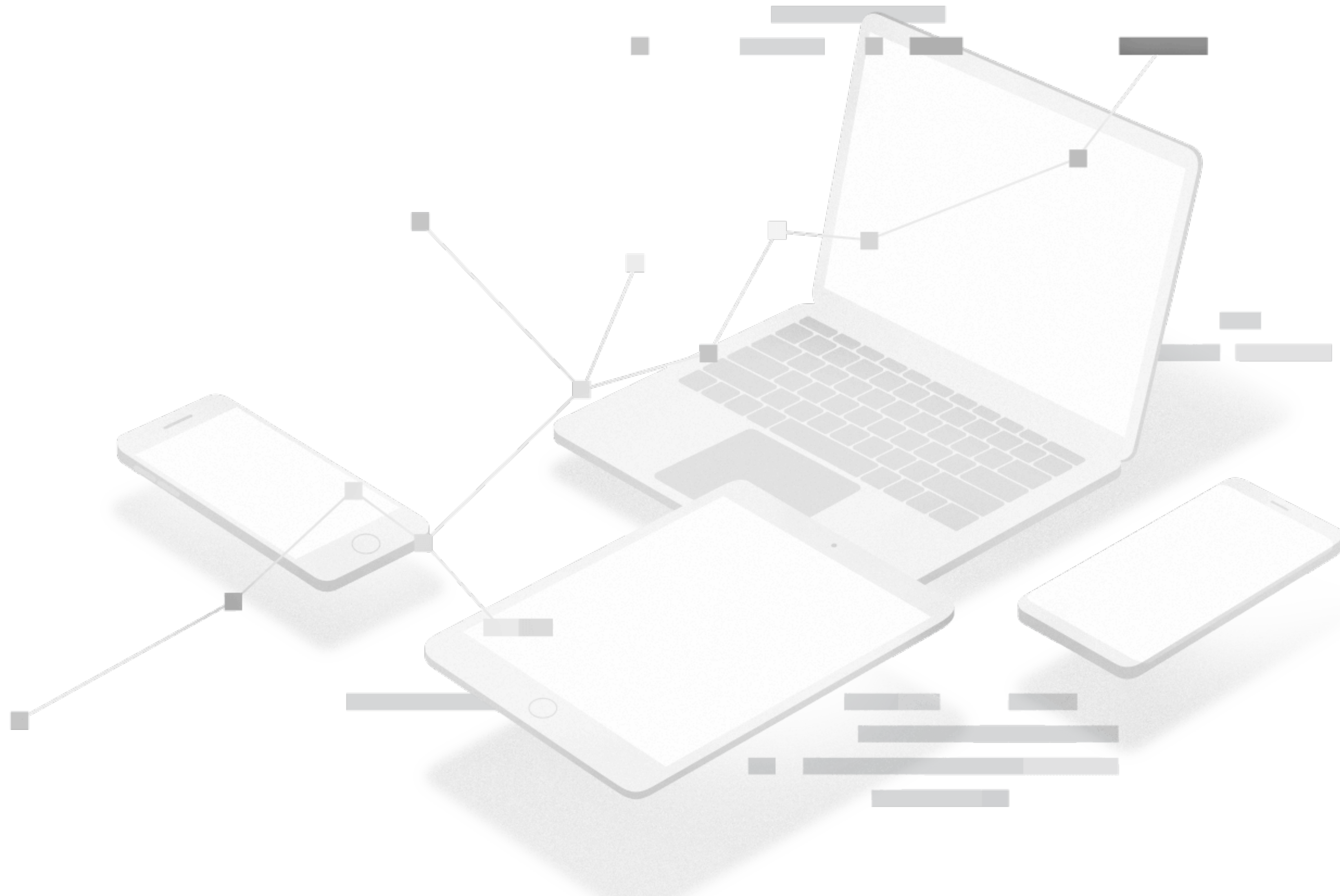


# Golang on Airbridge AIRBLOC

ab180



## 발표자 소개

---



김효준

Airbridge **Android SDK Developer**

Airbridge **Backend Developer**

Airbloc **Blockchain Engineer** (2018 ~)



**이 발표로  
얻을 수 있는 것**  
—

애널리틱스에서는 Go를 어떻게 활용할까?

기존 서비스 스택에 Go를 어떨 때 도입하면 좋을까?

왜 블록체인 개발을 Go로 하는걸까?

에어블록은 어떻게 Go를 활용할까?

**이 발표로  
얻을 수 있는 것**

—

애널리틱스에서는 Go를 어떻게 활용할까?

기존 서비스 스택에 Go를 어떨 때 도입하면 좋을까?

왜 블록체인 개발을 Go로 하는걸까?

에어블록은 어떻게 Go를 활용할까?

**얻을 수 없는 것**

—

재미, 연륜

# 목차

—

## Airbridge

1. Postback Request Worker에 Go 도입기
2. Bypass Worker에 Go 도입기

## AIRBLOC

1. 왜 블록체인 프로젝트가 Go를 쓸까?
2. 에어블록은 어떻게 Go를 쓸까?

처음 도입한 계기

# Airbridge

# Airbridge


사용자의 유입 경로와  
광고의 기여도를 분석해,  
마케팅 비용을 최적화해주는  
모바일 마케팅 애널리틱스

< ☰

Airbridge

English

dev@ab180.co



ab180블로그 ⚙️

First Owner

New Statistics Report

Insights

Reinstalls

Retention Rate

Fraud Insights

CTIT Report

Management

Tracking Links

Integrated Channel

CTIT Report ?

CSV

2018. 08. 06 → 2018. 08. 12

CTIT Summary from selected period ?

Search..

Total 5 Now 1-5

Second

Channel

Channel	All Installs	CTIT Summary				CTIT Average(Second)
		Under 10s	10s and over ~ Under 20s	20s and over ~ Under 10m	10m and over	
jojojo PAID	18	0 0%	3 16.67%	10 55.56%	5 27.78%	14,978.82
민티그럴 PAID	3	0 0%	1 33.33%	2 66.67%	0 0%	101.67
(not set) VIRAL	2	0 0%	0 0%	0 0%	2 100%	3,605

# Airbridge

## Service Delivery Pipeline

### Collection

Mobile  
SDK

Web  
SDK

Events

WAS

### Workers

Attribution  
Engine

Event  
Workers

### Provides

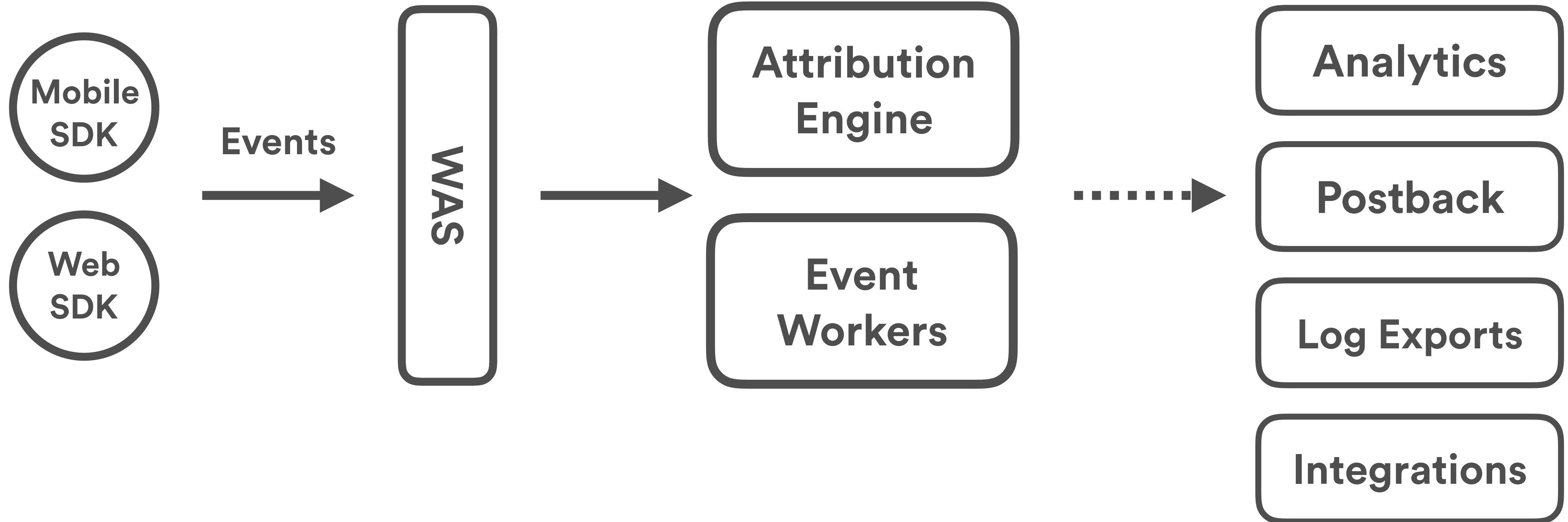
Dashboard

Analytics

Postback

Log Exports

Integrations



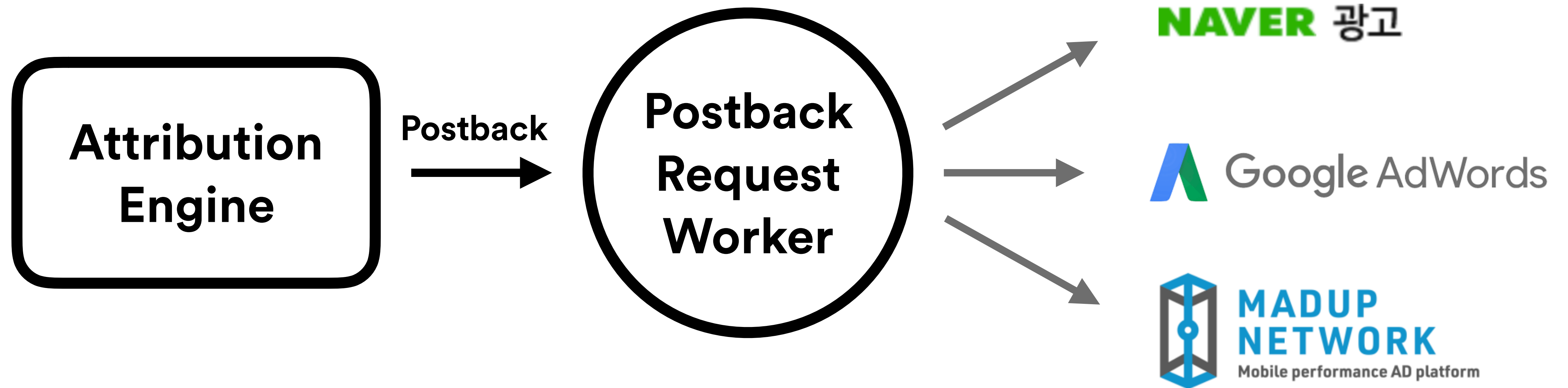


처음 도입한 컴포넌트

# Postback Request Worker

처음 도입한 컴포넌트

# Postback Request Worker



Airbridge Postback Request Worker

*Python*  
*Monolithic*  
*= 200 TPS*

---

**느 리 다!**

## Today

ab180

**ab180** 7:37 PM

빠르게 만들어주세요 ㅎㅎ

+

ㅋㅋㅋㅋ아니 잠깐ㅁ

**\*bold\*** *\_italics\_* ~strike~

# 요구사항을 정리해보자

---

- 빨라야 한다.
- 안정적이어야 한다. (광고비 정산!!)
- 비동기 리퀘스트다.
- 여러곳에 로그를 남겨야 한다.
  - New Relic
  - MySQL
  - Kafka → HDFS



**Airbridge** Postback Request Worker

# 요구사항을 정리해보자

---

- 빨라야 한다. **빠름!**
- 안정적이어야 한다. **에러처리 직관적!**
- 비동기 리퀘스트다. **고루틴 짱**
- 여러곳에 로그를 남겨야 한다.
  - New Relic
  - MySQL
  - Kafka → HDFS

**빠르게 하기 위해서...**



Package easyjson provides a fast and easy way to marshal/unmarshal Go structs to/from JSON without the use of reflection. In performance tests, easyjson outperforms the standard `encoding/json` package by a factor of 4-5x, and other JSON encoding packages by a factor of 2-3x.

easyjson aims to keep generated Go code simple enough so that it can be easily optimized or fixed. Another goal is to provide users with the ability to customize the generated code by providing options not available with the standard `encoding/json` package, such as generating "snake\_case" names or enabling `omitempty` behavior by default.

## Usage

```
# install
go get -u github.com/mailru/easyjson/...

# run
easyjson -all <file>.go
```

The above will generate `<file>_easyjson.go` containing the appropriate marshaler and unmarshaler funcs for all structs contained in `<file>.go`.

Please note that easyjson requires a full Go build environment and the `GOPATH` environment variable to be set. This is because easyjson code generation invokes `go run` on a temporary file (an approach to code generation borrowed from [ffjson](#)).

# fasthttp

---

Fast HTTP implementation for Go.

Currently fasthttp is successfully used by [VertaMedia](#) in a production serving up to 200K rps from more than 1.5M concurrent keep-alive connections per physical server.

[TechEmpower Benchmark round 12 results](#)

[Server Benchmarks](#)

[Client Benchmarks](#)

[Install](#)

[Documentation](#)

[Examples from docs](#)

[Code examples](#)

[Switching from net/http to fasthttp](#)

**Airbridge** Postback Request Worker

도입을 통해 해결되었던 것 :  
**성능, 안정성**  
**마이크로서비스화**

# 성능

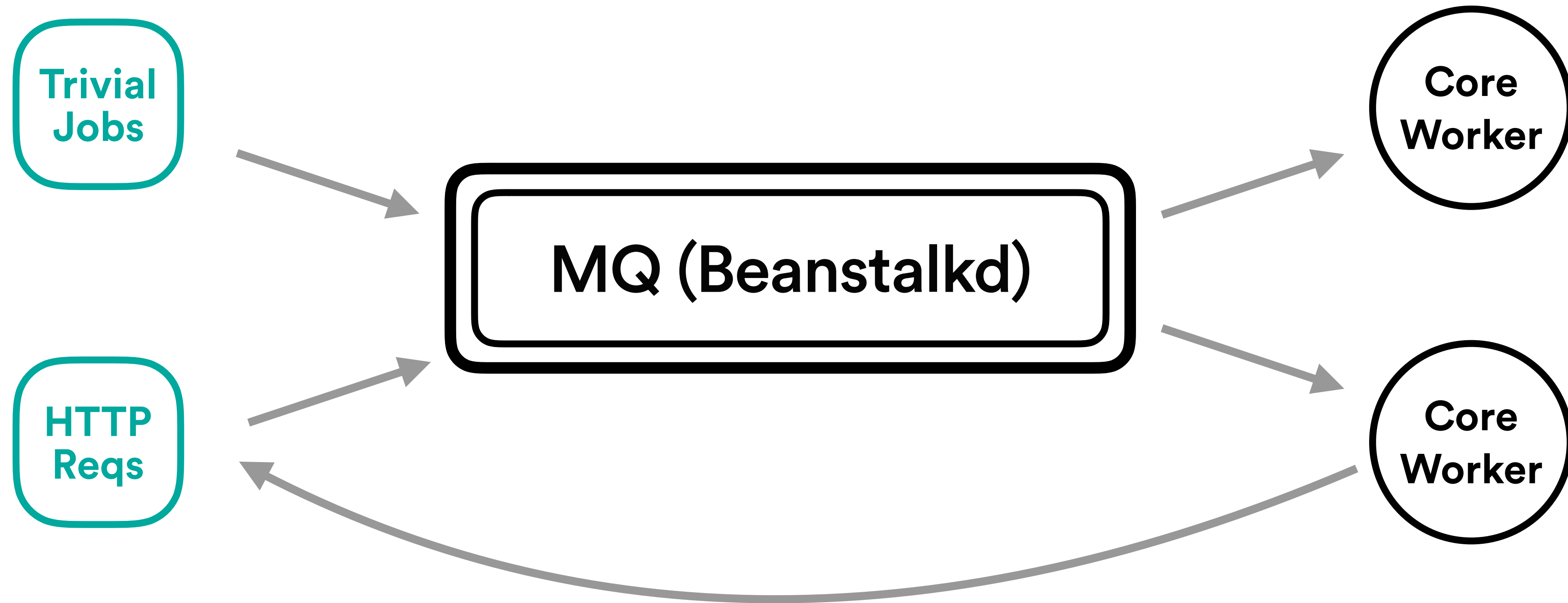
—

Before 200 RPS

**After 4000 RPS**

# 마이크로서비스화

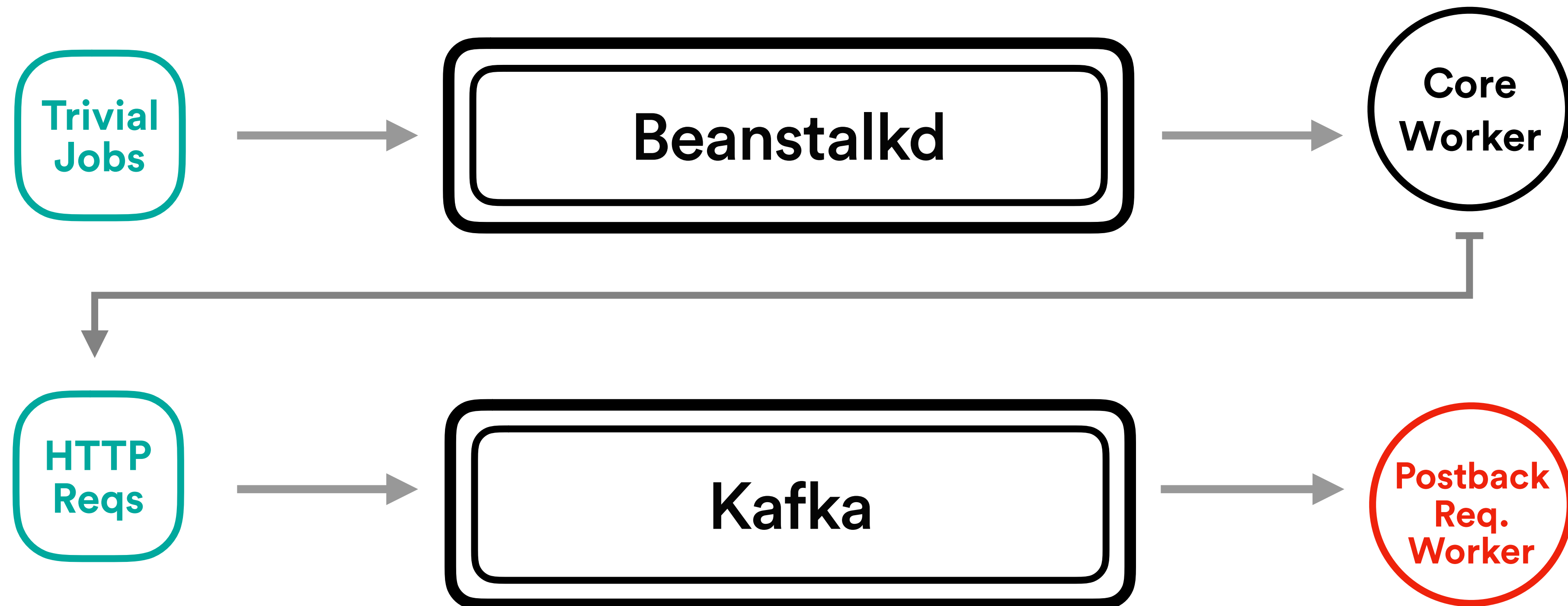
Before



**Airbridge** Postback Request Worker

# 마이크로서비스화

After!



Airbridge Postback Request Worker

# 도입을 통해 해결된 것 —

- 성능의 비약적인 향상 (200 -> 4000 TPS)
  - 당시 요구사항인 1000 TPS 를 훨씬 웃돌음
  - 따라서 Scale Down
  - 비용절감!
- 매우 안정적임
- 마이크로서비스화가 크다고 생각함
  - **Easy to deploy!**

# A Struggle: **Graceful Exit**



# 알 수 없는 버그?

## Graceful Exit

로그가 종종 누락돼요...  
재배포할 때인것 같은데

# 발견한 문제점

## Graceful Exit

```
// setup loggers. you can add more Loggers in here.  
loggers = NewLoggers(  
    NewStdoutLogger(),  
    NewKafkaLogger(producer, config.LogTopic),  
    NewMySQLLogger(config.MySQLConnection),  
)
```

# 발견한 문제점

## Graceful Exit

```
// save a log to multiple loggers.  
go loggers.Save(log, request, agent, trx)
```

# 발견한 문제점

—

# Graceful Exit

WaitGroup coverage

Kafka  
Message

Send Postback Request

Log  
Async

Unsafe!

Airbridge Postback Request Worker

# 발견한 문제점

## Graceful Exit

- Sarama를 사용해 간단한 Kafka Consumer Interface 를 구현
- 기존에 고려가 없어서 되는 웹 프레임워크를 쓰다 보니 몰랐다...
- Graceful Exit 처리에 대한 고려가 조금 부족했음

# 첫 도입에서의 총평

—



**Successful Debut!**

**다른 곳에도 사용해보자!**

# 다음에 도입한 곳

## **Bypass WAS**

**Airbridge** Bypass WAS

# Before : Monolithic

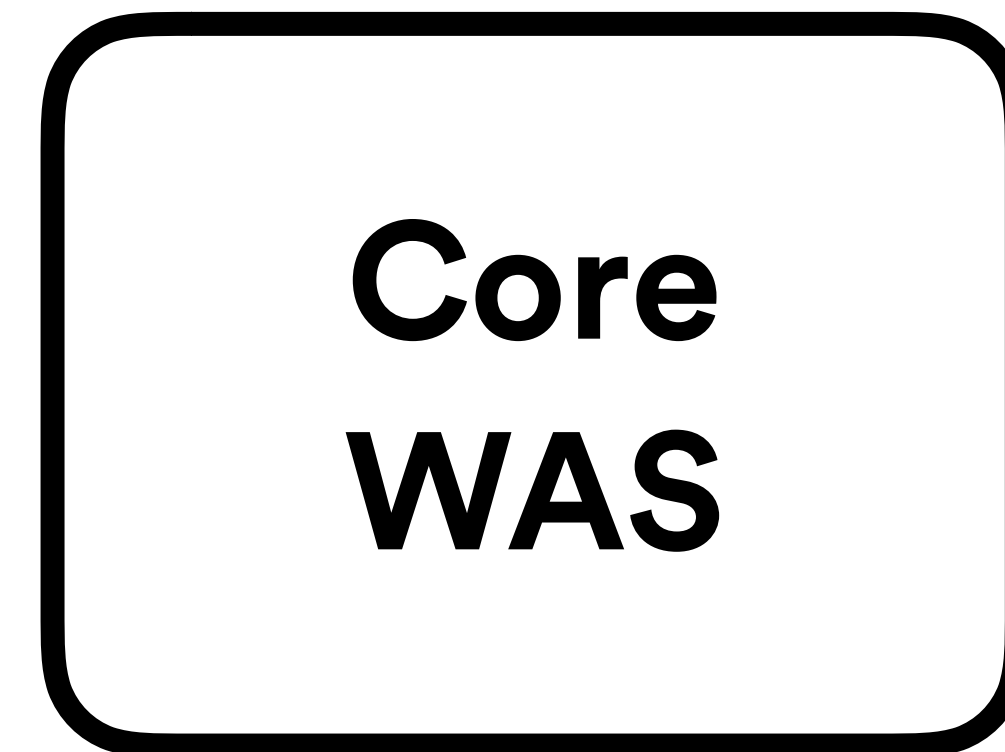
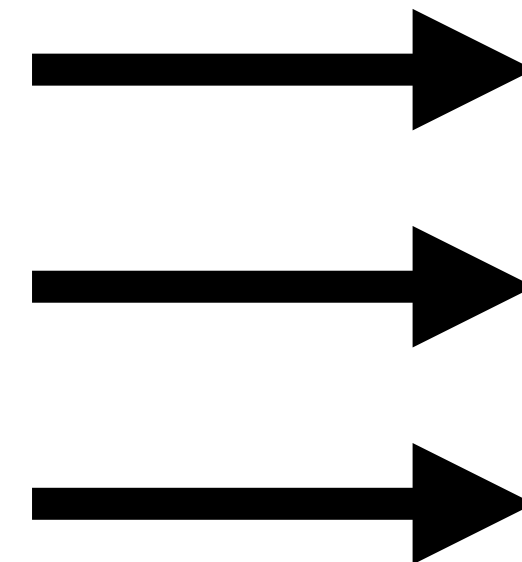
---

Touchpoints

Conversions

Ad Integration Requests

Fire & Forget Events



Airbridge Bypass WAS



Fire & Forget 이벤트를  
**별도로 빼면 어떨까?**

**Core  
WAS**

Touchpoints  
Conversions

**Bypass  
WAS**

Fire & Forgets  
using Go

**Airbridge** Bypass WAS

Fire & Forget  
Events



**Bypass  
WAS**



**kafka**

모듈화를 통한 개발비용 최소화  
그에 비해 큰 비용절감

# 작업중인, 많은 프로젝트들

—

- Go Consumer (Deduplicator)
- Summary Worker
- Identity Engine
- Postback Target Worker

# 프로덕션 도입 총평

---

- I/O Heavy한 병목 포인트나 Micromodule에 적용하기 좋았다.
- 서비스 관리가 편하고, CD가 편하다.
  - 안정적이므로 관리 코스트가 없음.
  - 바이너리 단위의 배포도 가능.
- 하지만...

# 프로덕션 도입 총평

—

비즈니스 로직 작성엔 아직까지 적합하지 않았다.  
Airbridge는 데이터 파이프라인 뒷단으로 갈수록  
각종 포스트백 및 연동으로 인해 비정형 데이터를 많이 다룸.

그럼에도, Identity Engine 등 Tradeoff를 이룰 수 있음

Airbridge

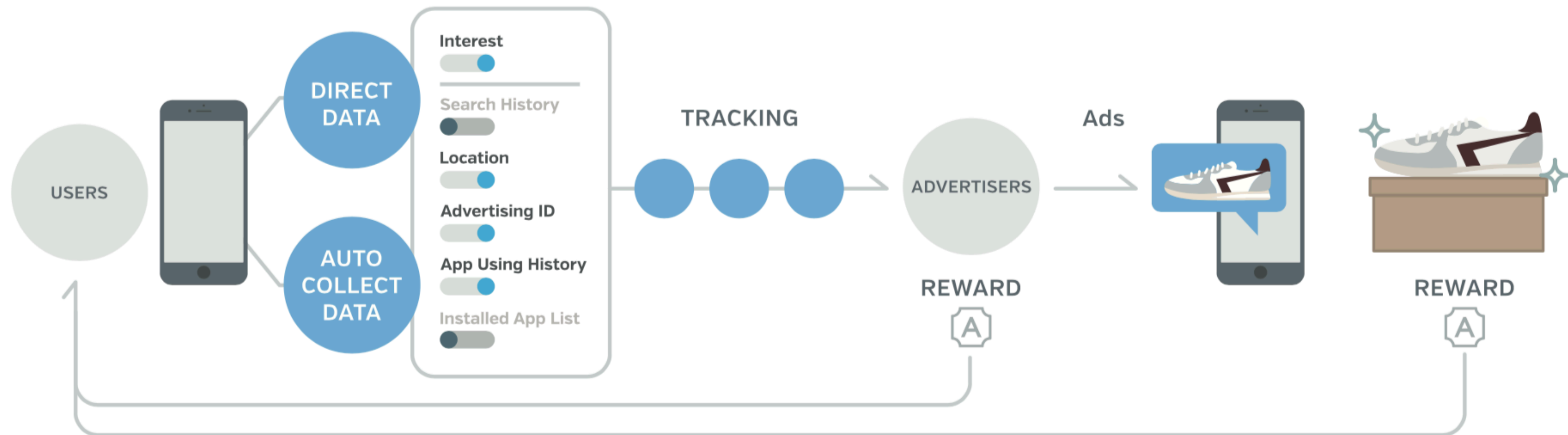
그 다음 제품

AIRBLOC



# AIRBLOC

개인 데이터를 투명하게 수집해  
정당한 가치를 받고  
거래할 수 있게 해주는  
탈중앙화된 데이터 교환 플랫폼



# 시작하기 전에 배경 설명

1. 블록체인이 뭐길래
2. Go 를 쓰기 적합할까요?

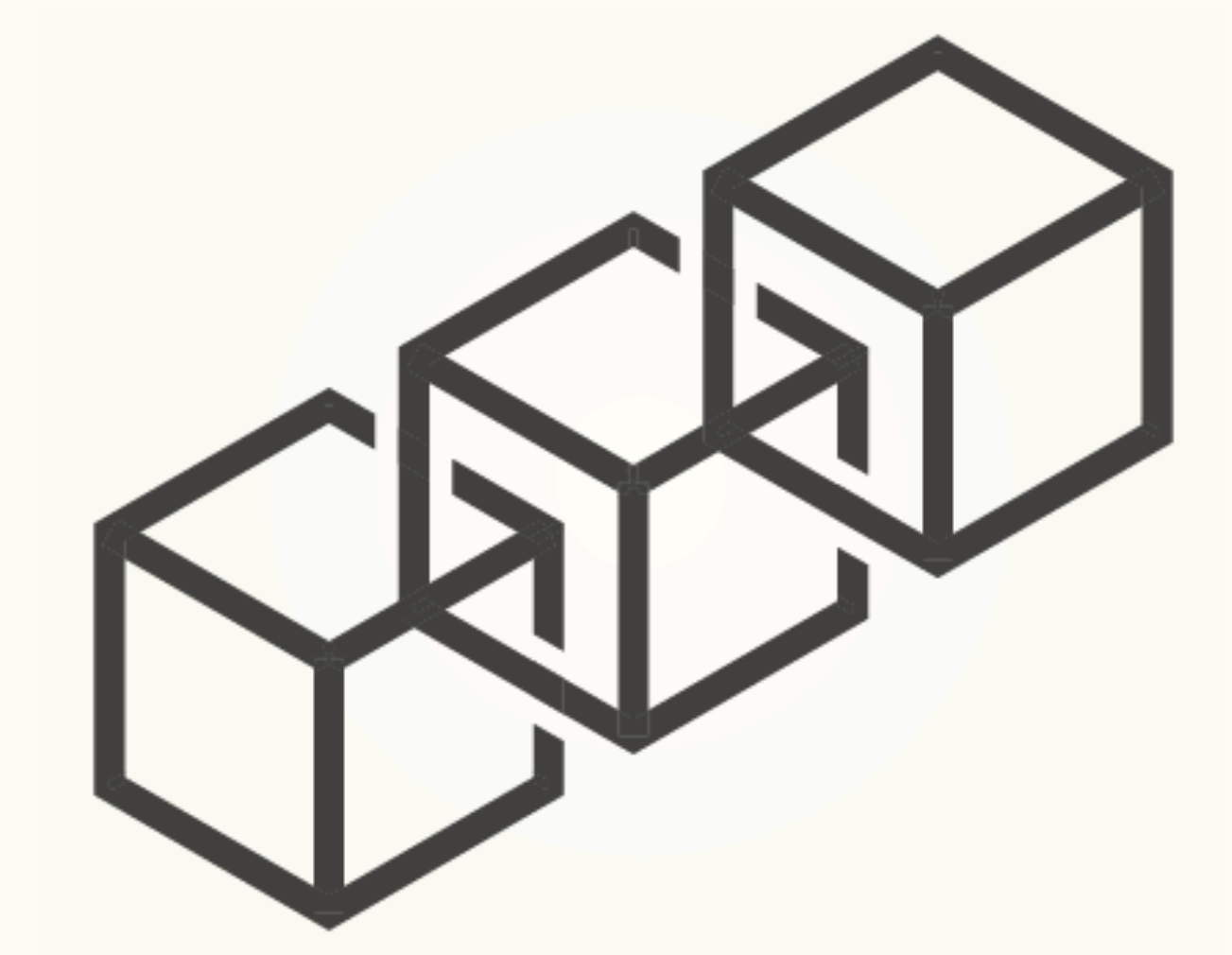
1.

# 블록체인이 뭐길래

Definition of Blockchain

# **A Distributed Trustless State Machine**

- Byzantine Fault Tolerance
- Sybil Attack Tolerance
- DoS Attack Tolerance



Defi

Wow

A T

Sta

- B

- S

- D

easy understandable

luv bloquechaine

so intuitive

easy crypto

I wanna study

AIRBLOC

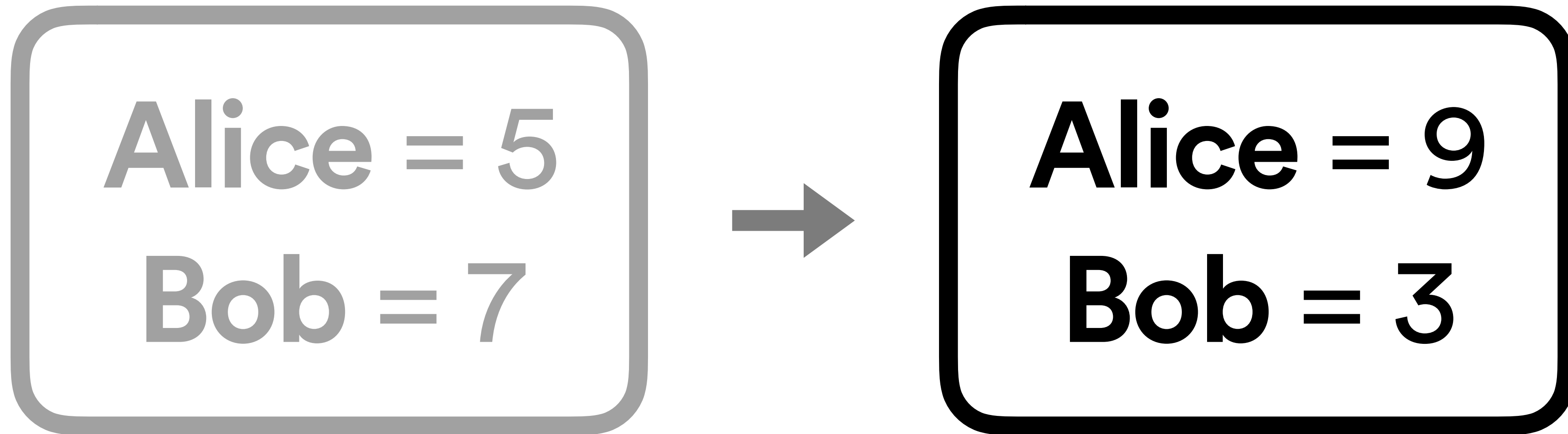
여기 **State Machine**<sup>0</sup>이 있습니다!

**Alice = 5**

**Bob = 7**

# State Machine

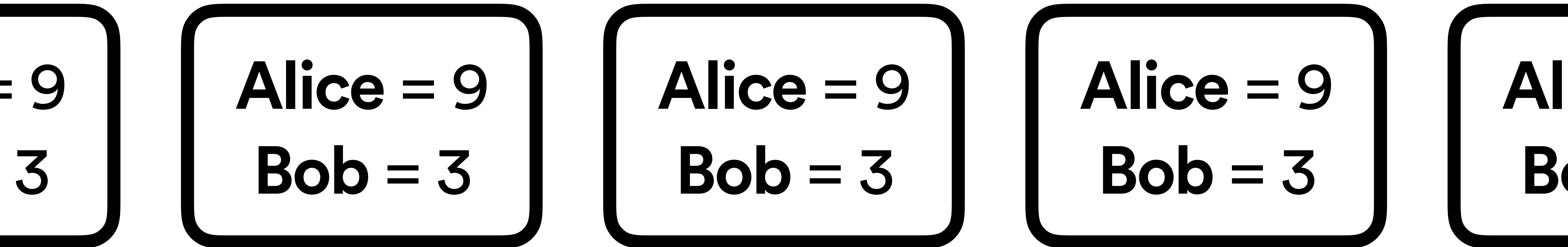
값을 변화시키는걸 State Transition이라고 함





# Distributed State Machine

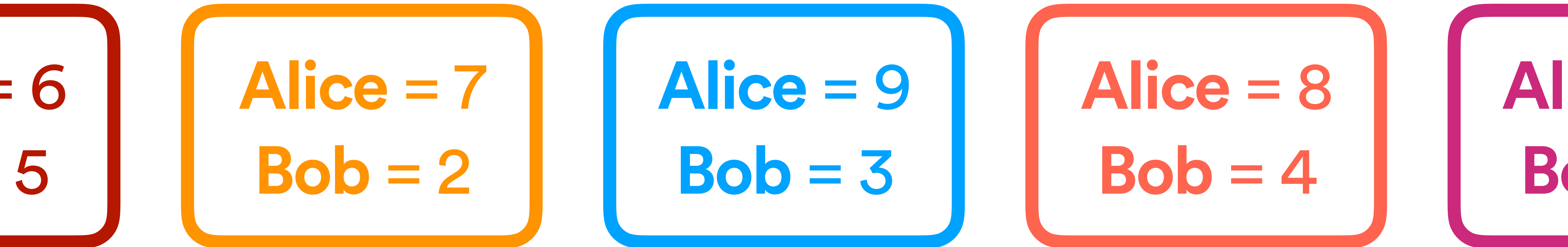
Replication을 통해  
Distributed State Machine을 만든다!





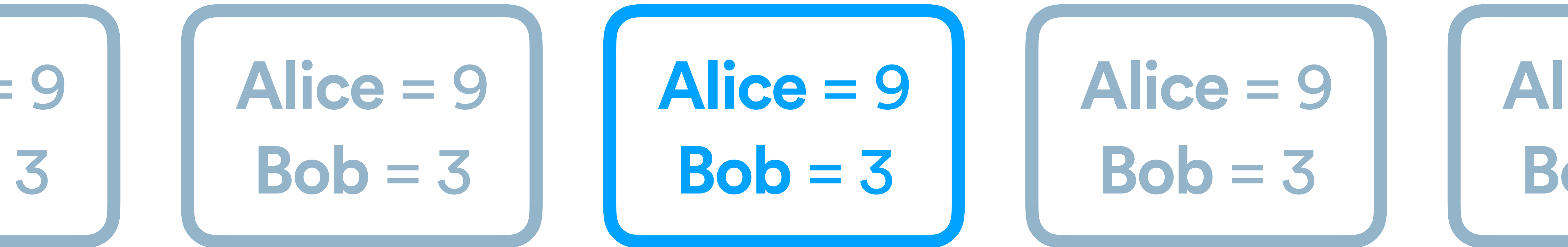
# Distributed State Machine

만약 서로 다 다르게  
값이 바뀌면 어떡할까?



# Distributed State Machine

일반적으로 **Leader**를 정한다.



**Leader**

AIRBLOC

# Distributed State Machine

만약 **Leader**가 죽는다면?

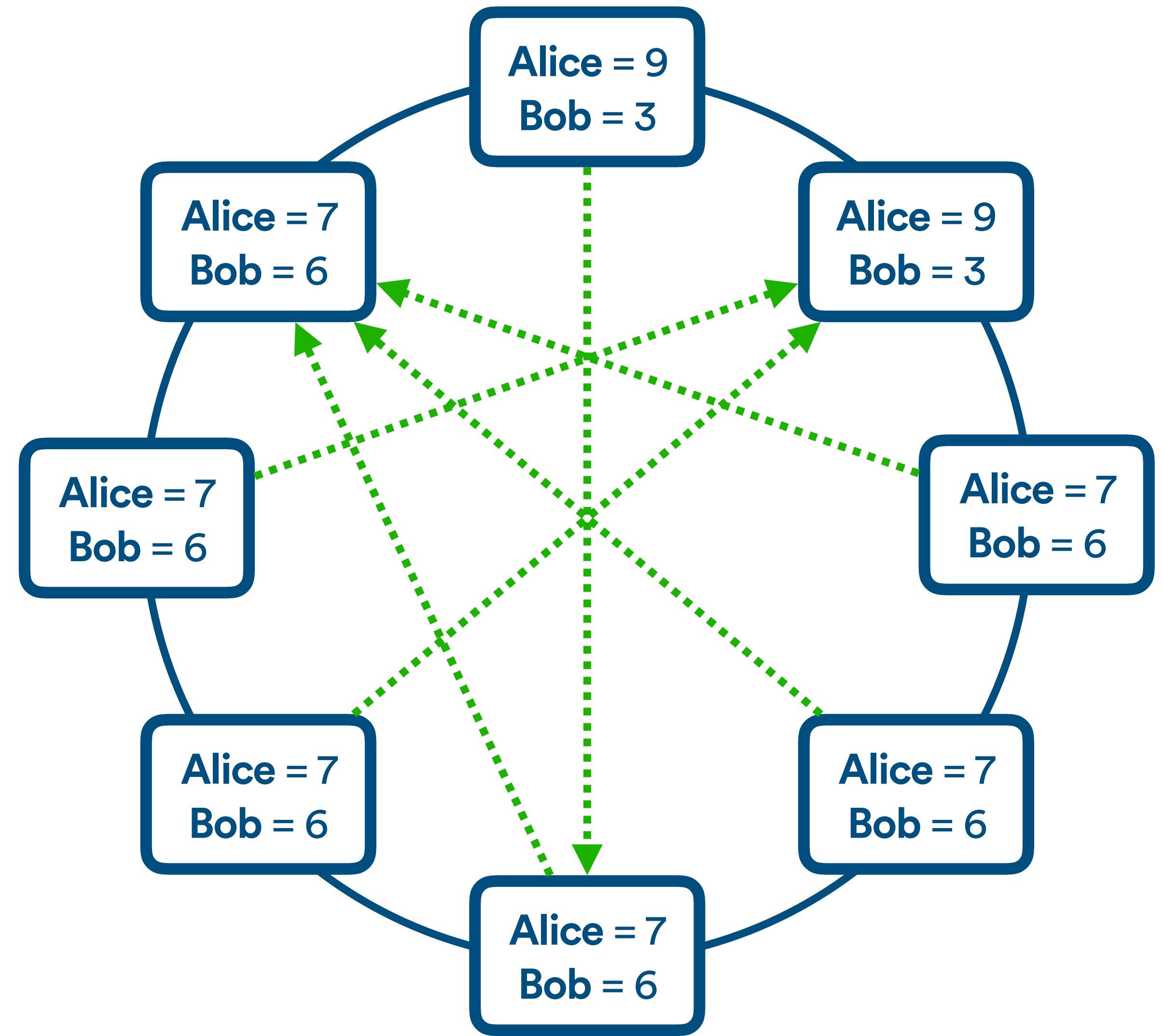


Leader

AIRBLOC

# Distributed State Machine

Leader를 새로 선출해  
뭐가 올바른 상태인지 합의  
**Consensus**

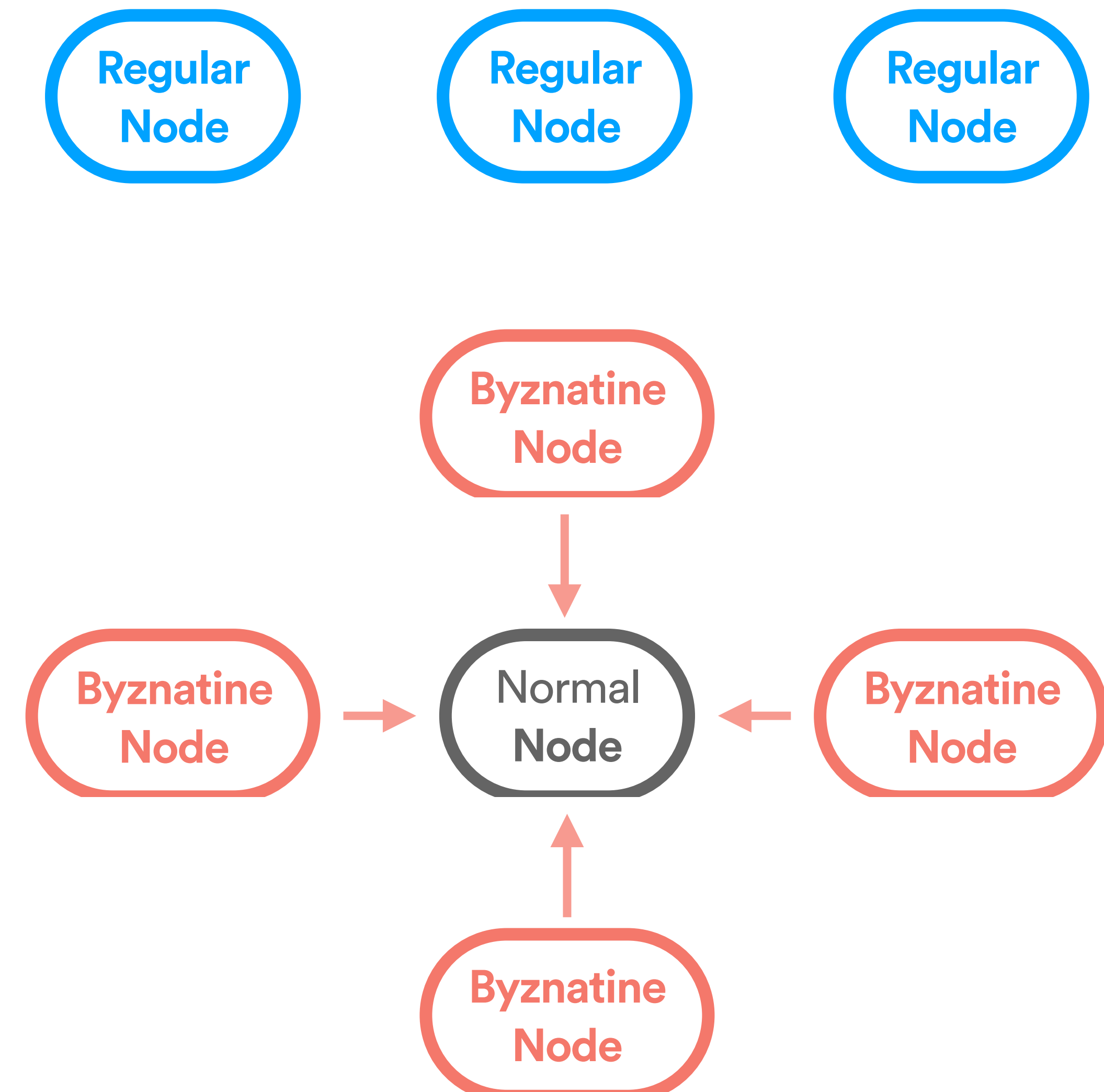


# Distributed State Machine

**Problem:**

## Byzantine General Problem

내 주변 노드들이 거짓말하면 어떡해?



# Distributed State Machine

**Answer**

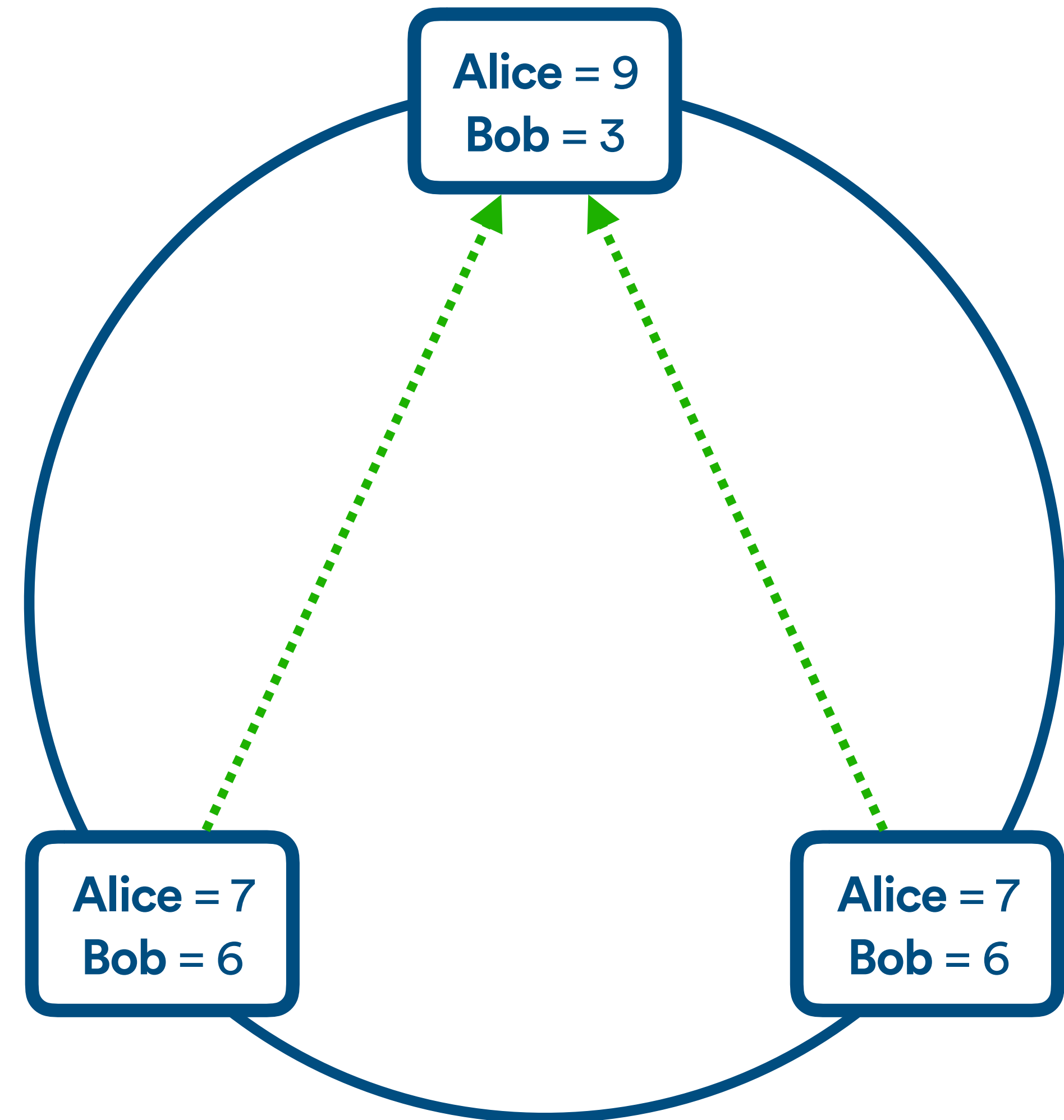
**BFT (Byzantine Fault Tolerance). But...**

AIRBLOC

# Distributed State Machine

## Problem

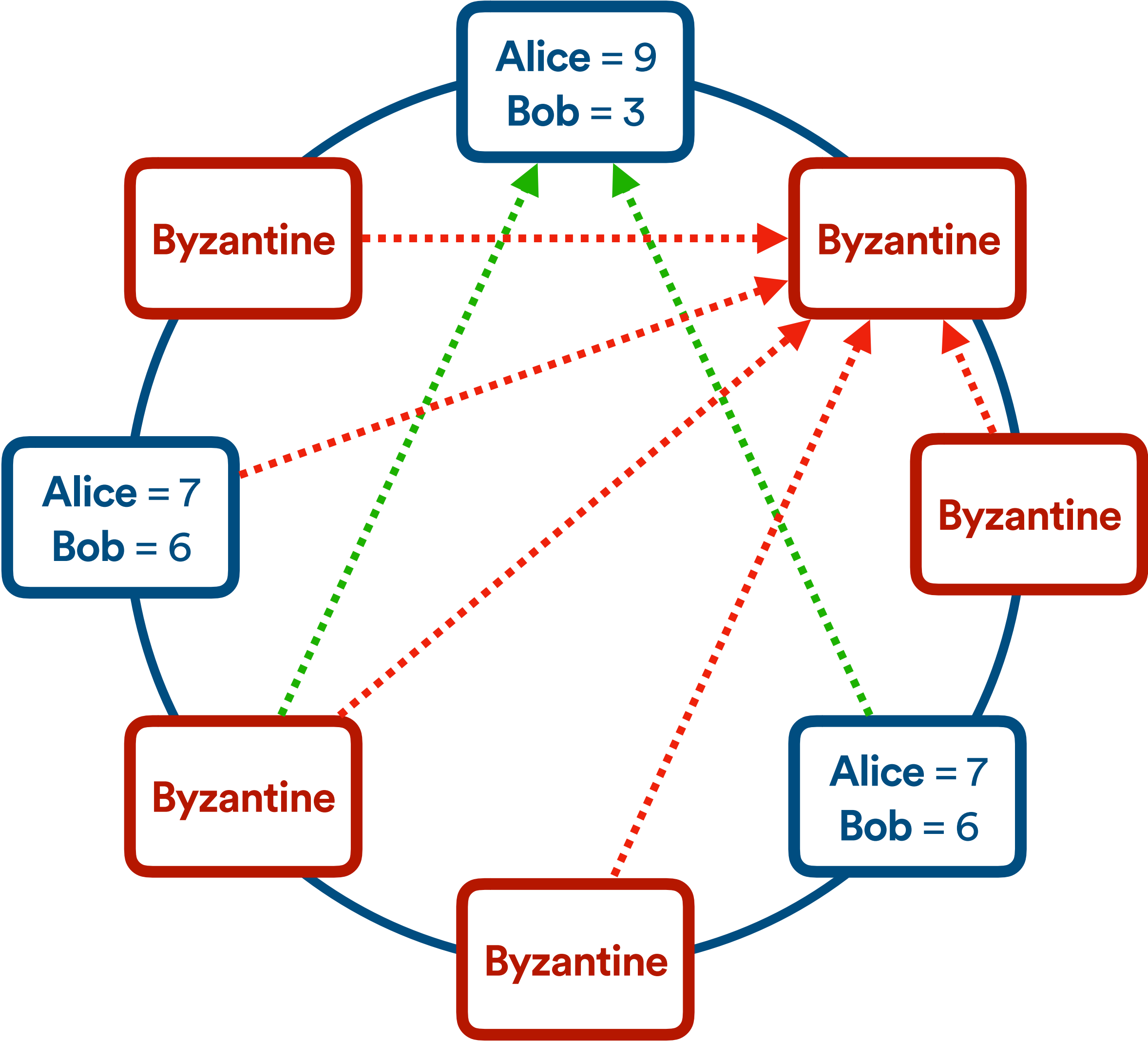
누구나 참여할 수 있는 네트워크



# Distributed State Machine

## Problem

누구나 참여할 수 있는 네트워크

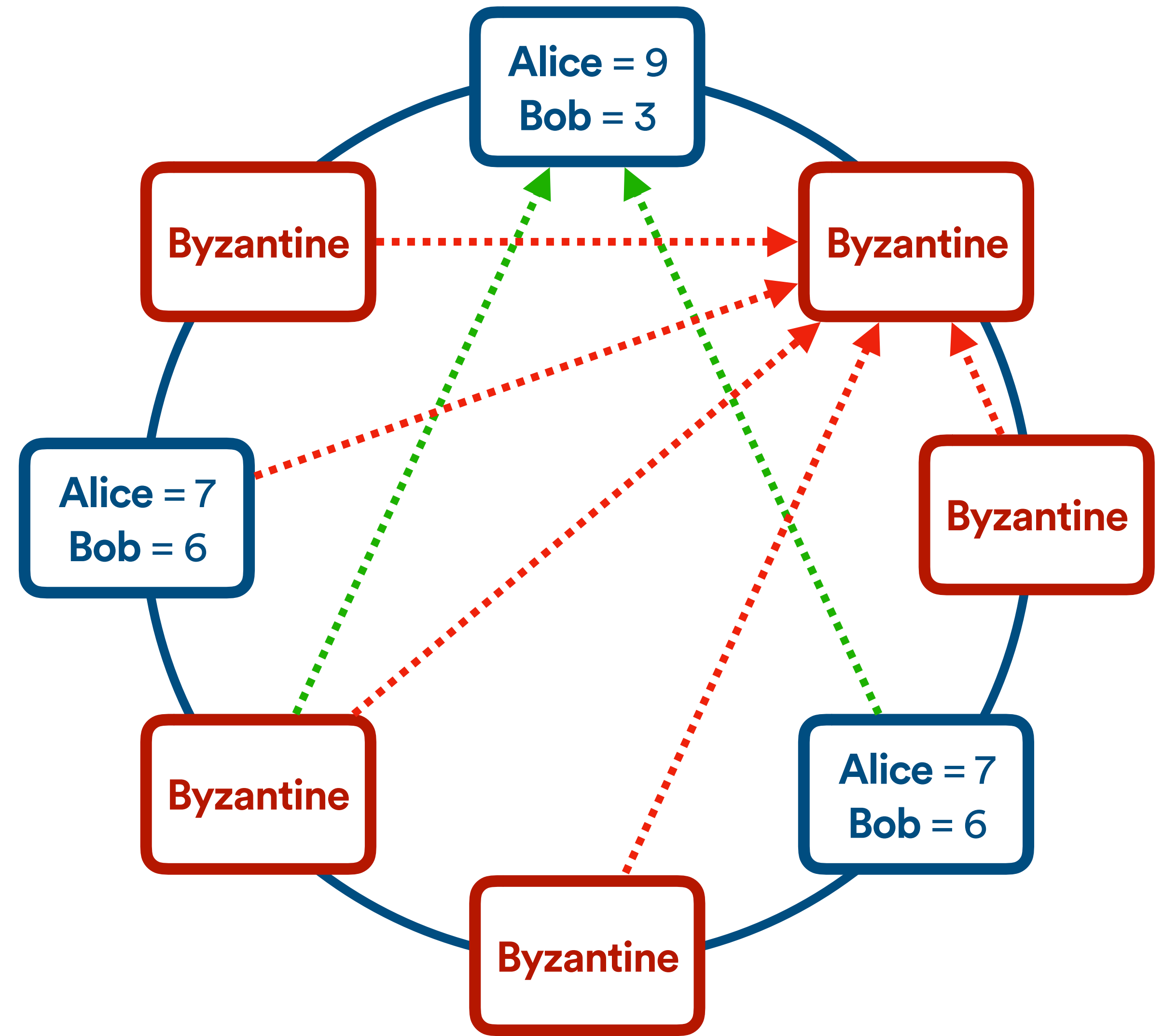




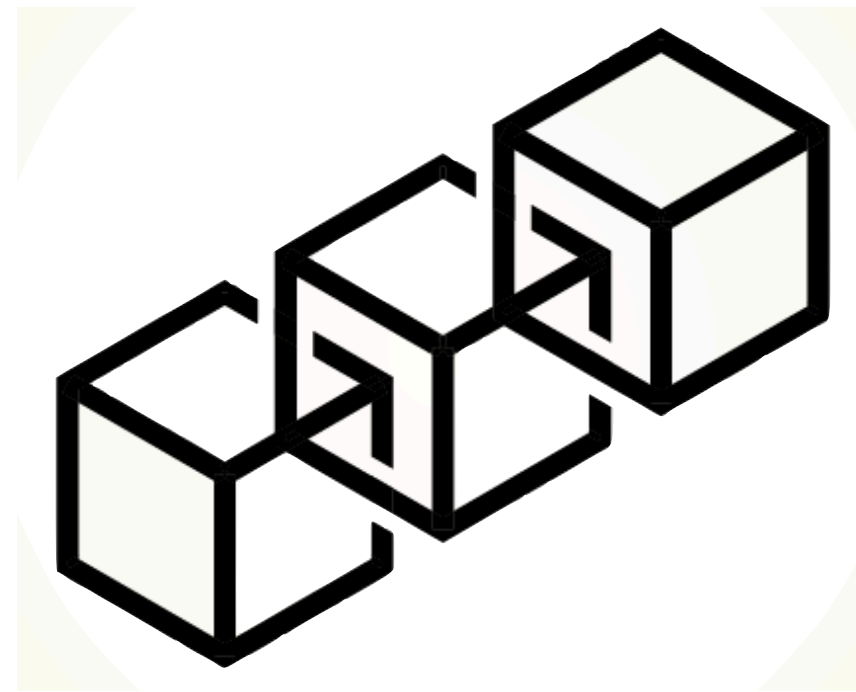
# Distributed State Machine

악성 참여자가 가짜 신원 생성해  
합의 결과를 조작한다면?

## Sybil Attack



# Distributed State Machine



Let's make a blockchain!

AIRBLOC

# Distributed State Machine

Transaction들의 리스트로 상태를 정의하자.

**Alice** → 9  
**Bob** → 3

Transaction #1

**Alice** → 4

Transaction #2

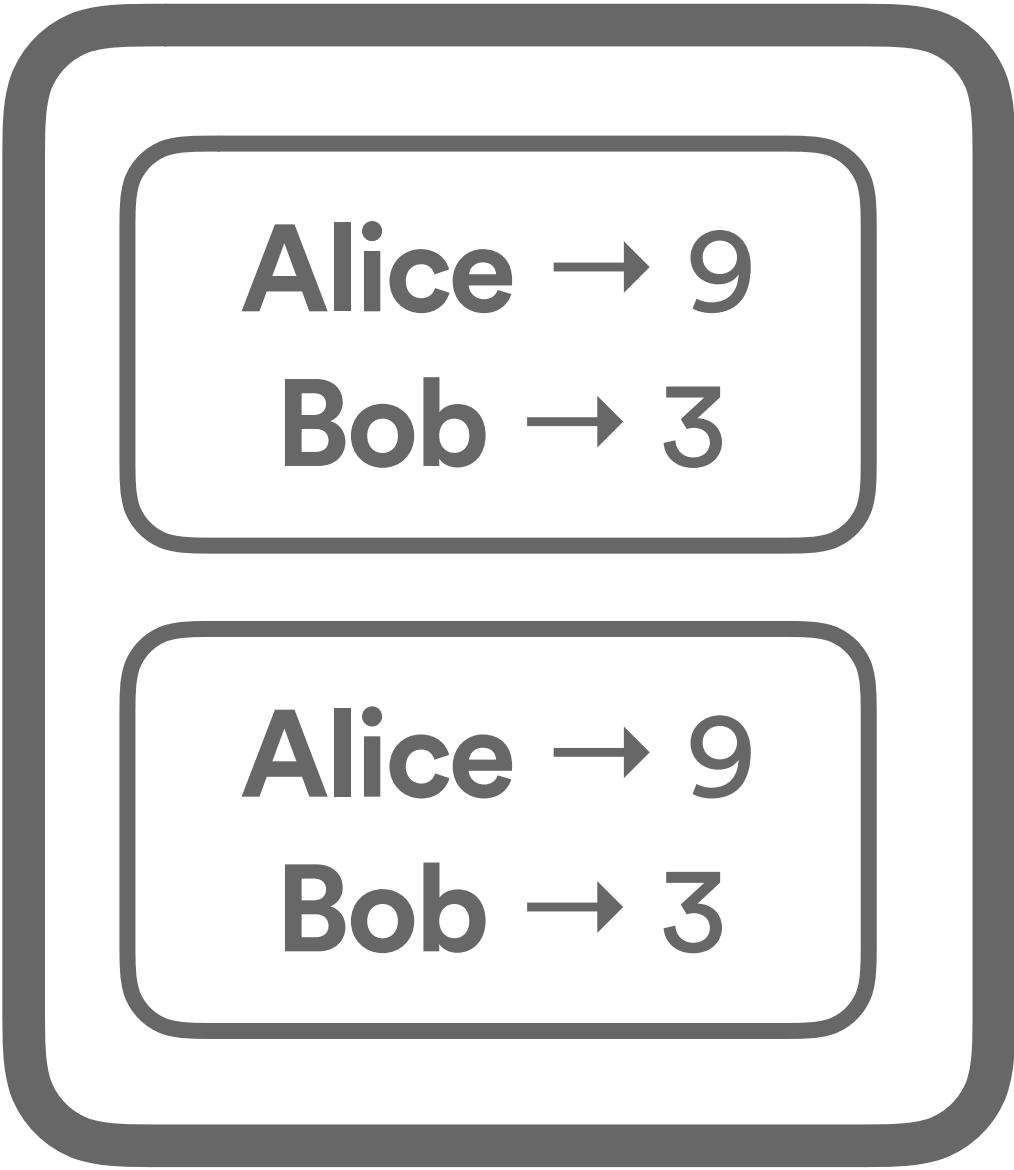
**Bob** → 2

Transaction #3

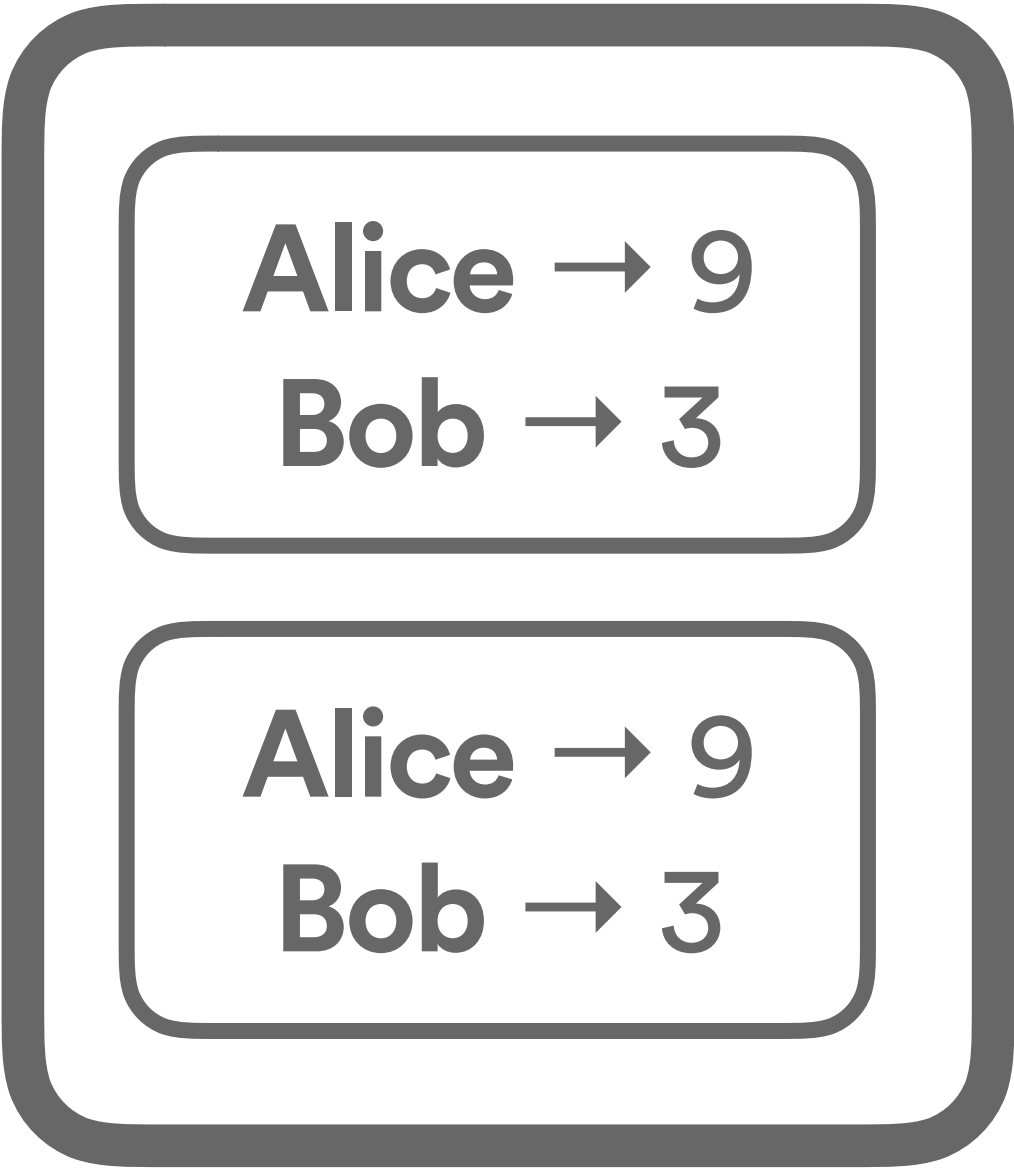
AIRBLOC

# Distributed State Machine

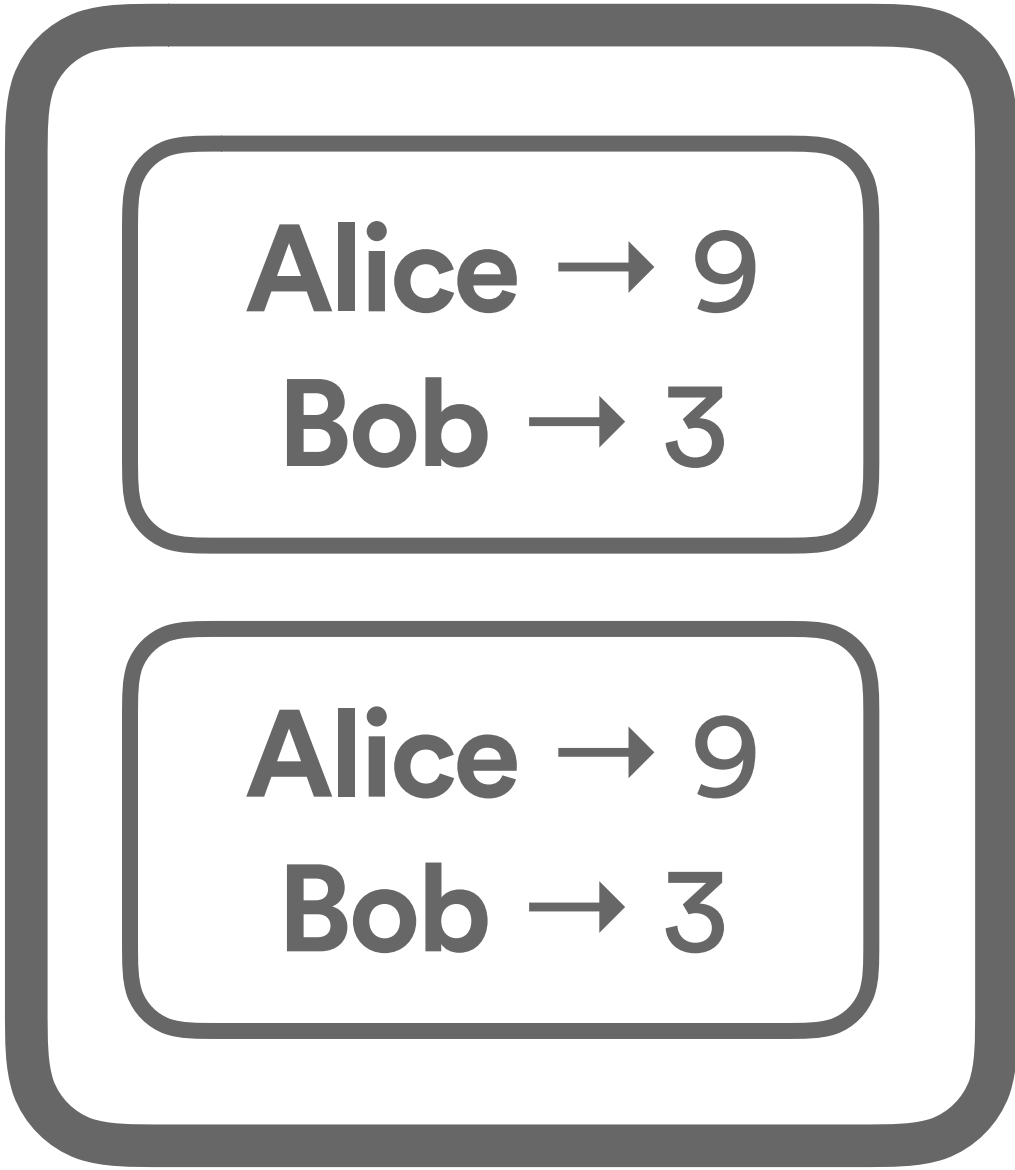
Transaction들을 모아서 Block을 만들자.



Block #1



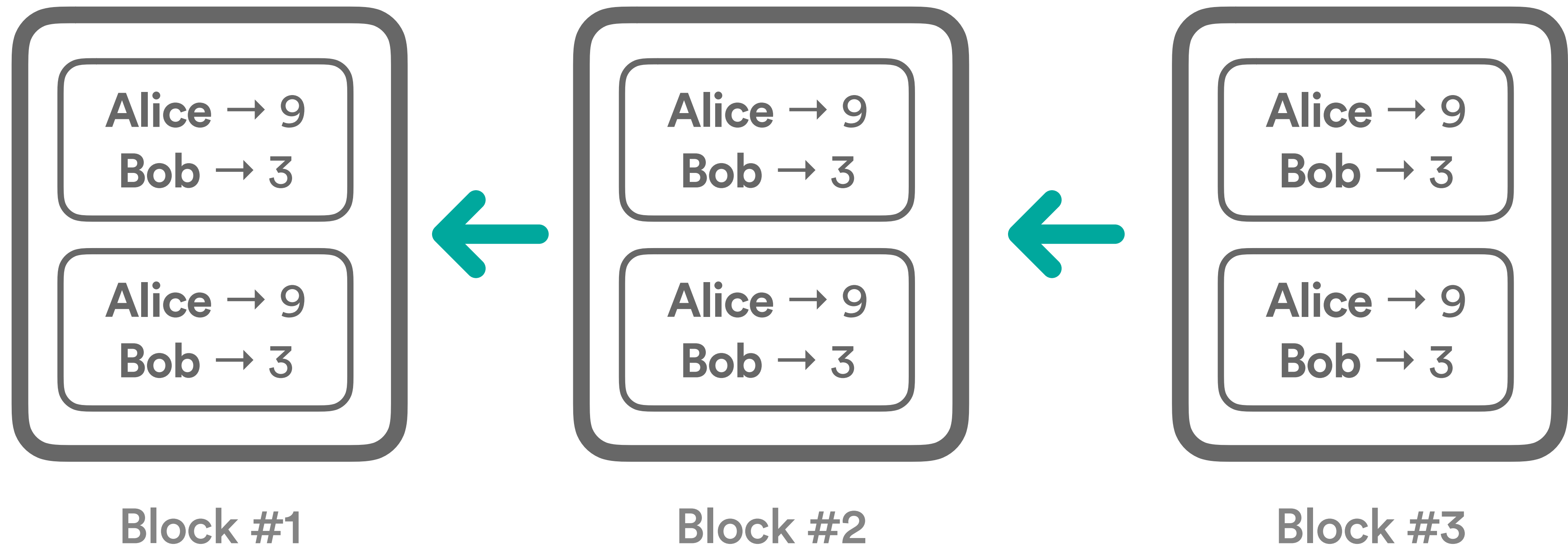
Block #2



Block #3

# Distributed State Machine

Block들을 믿을 수 있어야 하니까 **Hashed List**를 사용하자.



AIRBLOC

# Distributed State Machine

아직 남아있는 문제

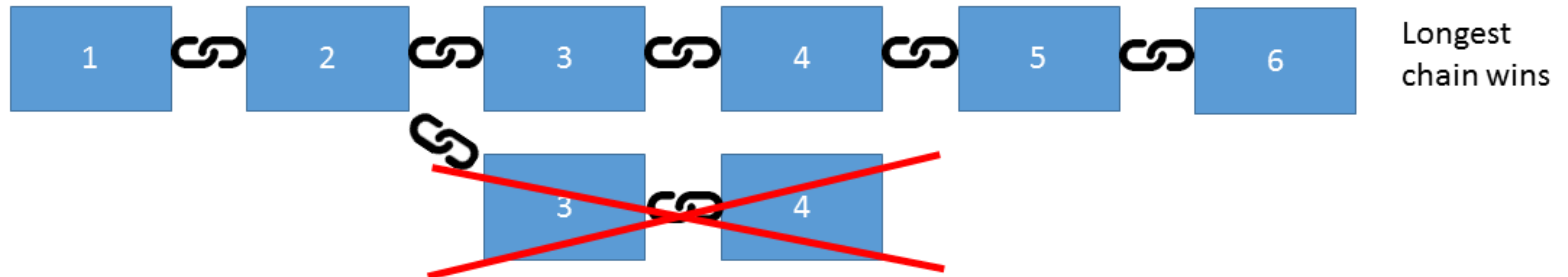
Byzantine Fault

Sybil Attack

DoS Attack

AIRBLOC

랜덤으로 리더를 선출하자!  
가장 긴 블록체인이 정답이다!

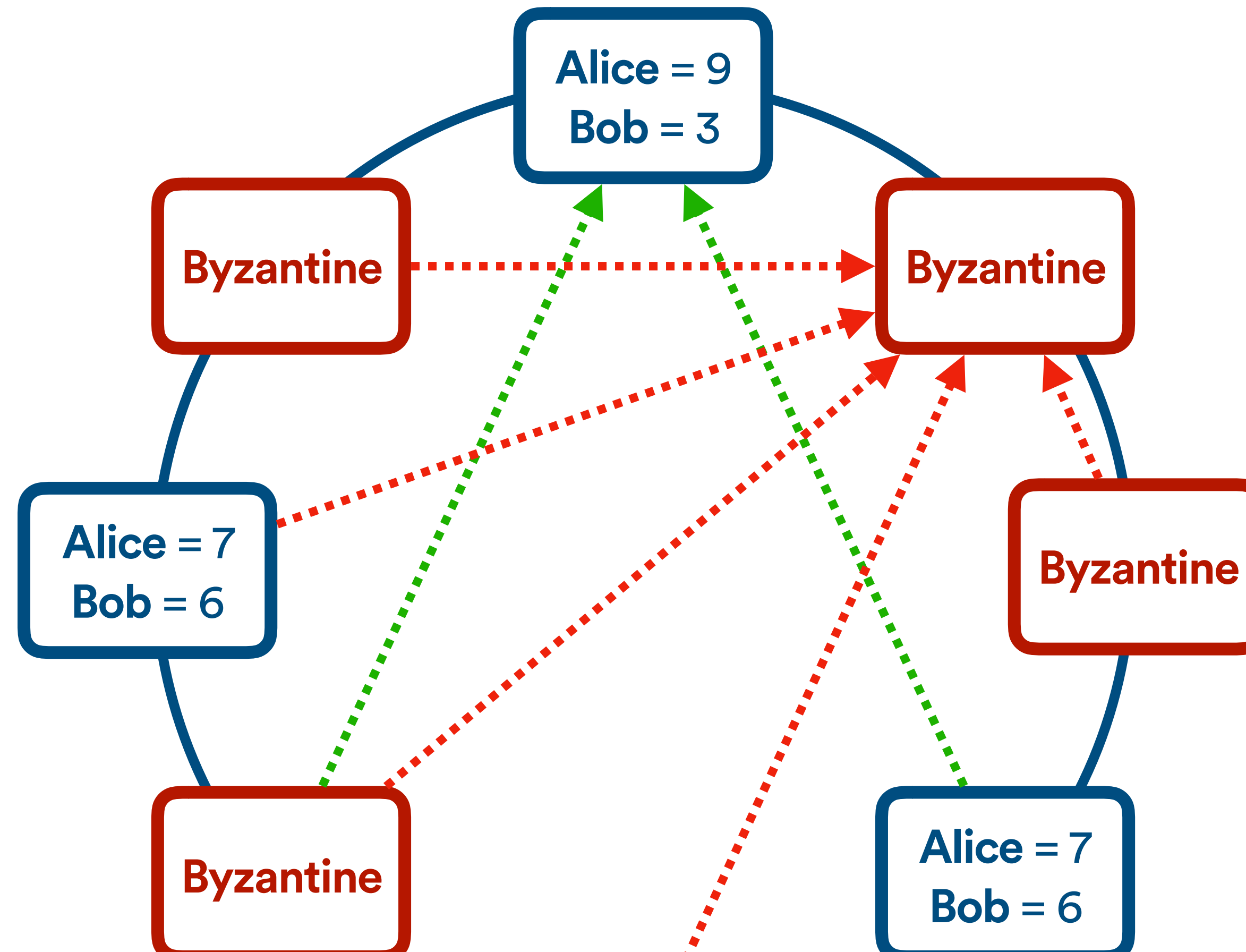


## 문제

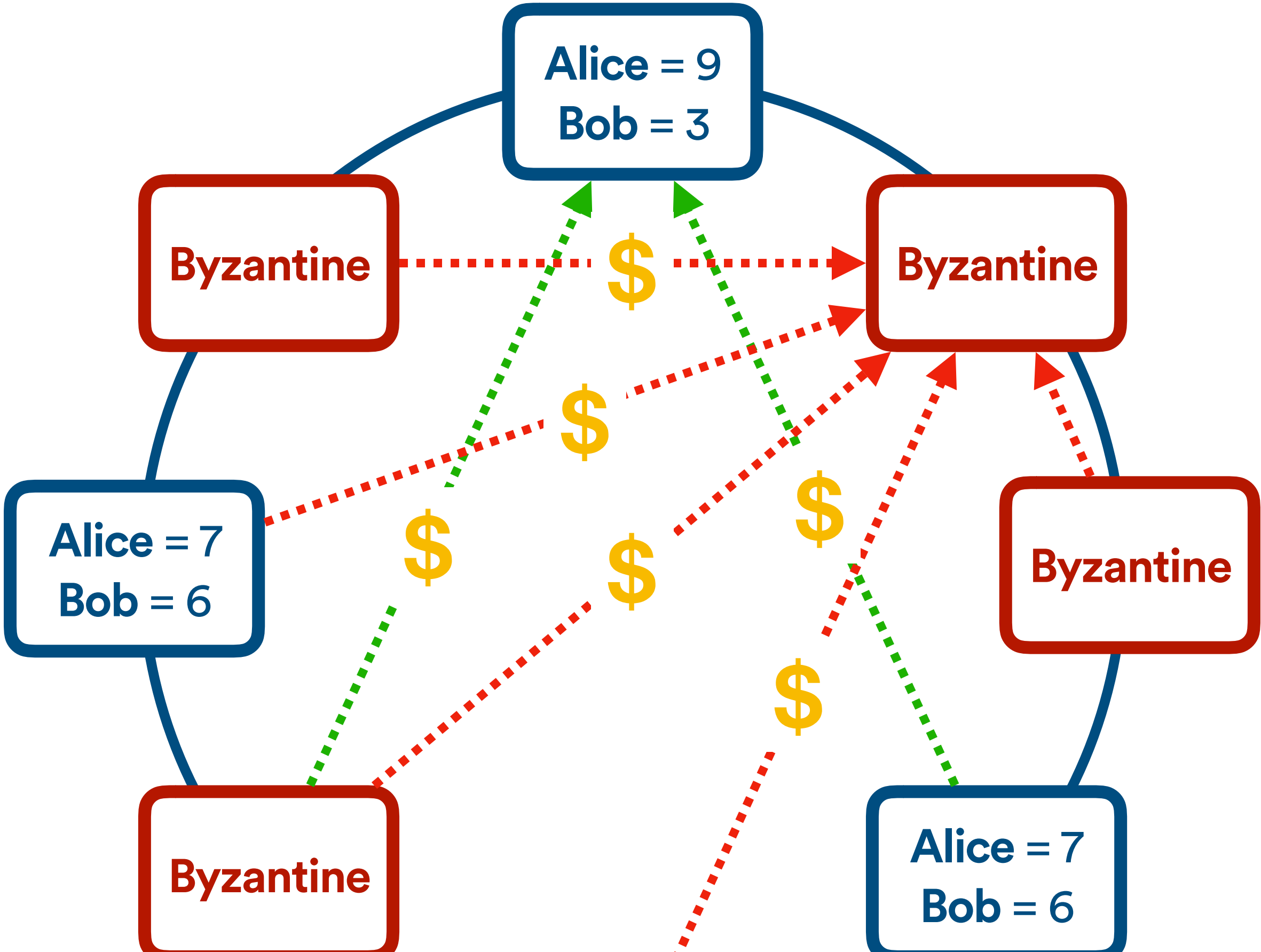
1. 중앙 주체가 없는데 어떻게 랜덤으로 선출?
2. 투표는 조작 가능하잖아! **Sybil Attack**



컨센서스 문제로 다시 돌아가보자.



한 표에 비용을 부과한다면?



**한 표에 컴퓨팅 비용을 부과한다면?**  
일정 시간이 걸리는 연산 문제를 주고  
맞춘 사람에게 블록 생성 기회 (리더)를 주자!

한 표에 **컴퓨팅** 비용을 부과한다면?  
일정 시간이 걸리는 연산 문제를 주고  
맞춘 사람에게 블록 생성 기회 (리더)를 주자!  
**랜덤, Sybil Attack 해결 가능**

Distributed State Machine

Byzantine Fault

Sybil Attack

DoS Attack

# Proof-of-Work

—

어떤 **어려운 문제**를 풀어서, 맞춘 놈이 블록을 생성한다!

비용이 드니까, 대신 문제 맞춘 놈에겐 **보상**을 주자

AIRBLOC

Distributed State Machine

Byzantine Fault

Sybil Attack

DoS Attack

# Proof-of-Work

---

어떤 **어려운 문제** 를 풀어서 **(채굴)**, 맞춘 놈이 블록을 생성한다!  
비용이 드니까, 대신 문제 맞춘 놈에겐 **보상**을 주자

AIRBLOC

Distributed State Machine

Byzantine Fault

Sybil Attack

DoS Attack

# Proof-of-Work

---

어떤 **어려운 문제** 를 풀어서 **(채굴)**, 맞춘 놈이 블록을 생성한다!  
비용이 드니까, 대신 문제 맞춘 놈에겐 **보상 (암호화폐)**을 주자

AIRBLOC

Distributed State Machine

Byzantine Fault

Sybil Attack

DoS Attack

# DoS Proof — Fee Model

AIRBLOC



Distributed State Machine

Byzantine Fault  
Sybil Attack  
DoS Attack

# DoS Proof — Fee Model

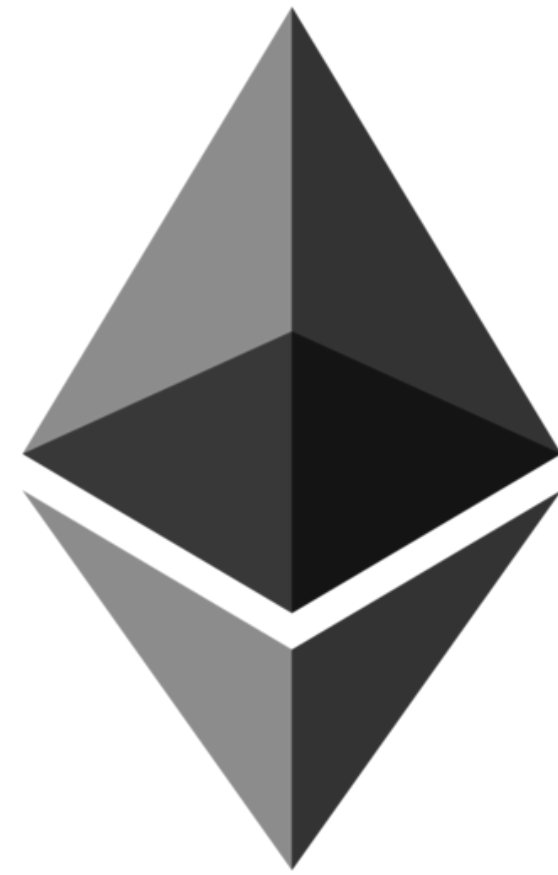


Alternative : Resource Allocation Market

AIRBLOC

# 블록체인 프로젝트에서 **왜 Go를 쓸까요?**

# 블록체인 프로젝트에서 왜 Go를 쓸까요?



AIRBLOC



# 이더리움

## Blockchain App Platform

### 스마트 계약을 최초로 도입한 플랫폼

AIRBLOC



이더리움

Blockchain App Platform

스마트 계약을 최초로 도입한 플랫폼

**이더리움, 너는 왜 썼어?**

AIRBLOC

# 이더리움은 왜 Go 일까?

## 요구사항

- 컴퓨팅 자원이 많이 소모됨
- 병렬화되어야 하는 블록 검증 작업

AIRBLOC

ethereum / go-ethereum

<> Code

Issues 736

Pull requests

Official Go implementation of the Ethereum

go

blockchain

ethereum

p2p

geth

9,971 commits

23 branches

Branch: master

New pull request



manxiaqu and karalabe trie: fix typo (#17498)

.github

swarm: network

accounts

accounts: fixed

build

build: do not re

cmd

cmd, eth: clear

common

les: implement

consensus

miner: fix state

console

console: fixed

containers/docker

containers: dro

contracts

backends: con

core

miner: fix state

# 1. 컴퓨팅 자원이 많이 소모됨

채굴 연산을 수행해야 함. (Ethash)

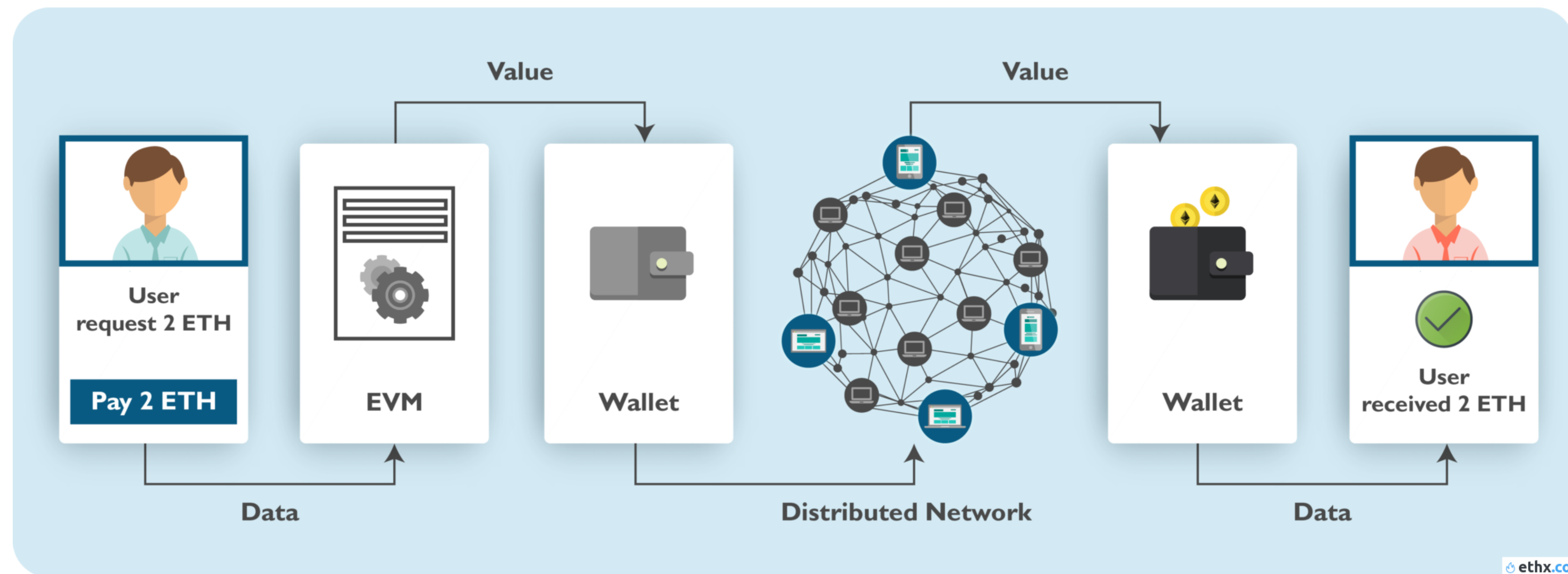
**EVM** (Ethereum VM) 구동

ECIES, KECCAK256 등 각종 암호화

## 2. 병렬화되어야 하는 블록 검증 작업

블록당 수백 개의 트랜잭션이 들어감.

트랜잭션마다 EVM을 돌려서, 해당 트랜잭션을 전부 실행해봐야 함.



AIRBLOC



# 이더리움은 왜 Go 일까?

## 요구사항

- 컴퓨팅 자원이 많이 소모됨
- 병렬화되어야 하는 블록 검증 작업
- **Good to go!**

AIRBLOC

ethereum / go-ethereum

<> Code

Issues 736

Pull requests

Official Go implementation of the Ethereum

go

blockchain

ethereum

p2p

geth

9,971 commits

23 branches

Branch: master ▾

New pull request



manxiaqu and karalabe trie: fix typo (#17498)

.github

swarm: network

accounts

accounts: fixed

build

build: do not re

cmd

cmd, eth: clear

common

les: implement

consensus

miner: fix state

console

console: fixed

containers/docker

containers: dro

contracts

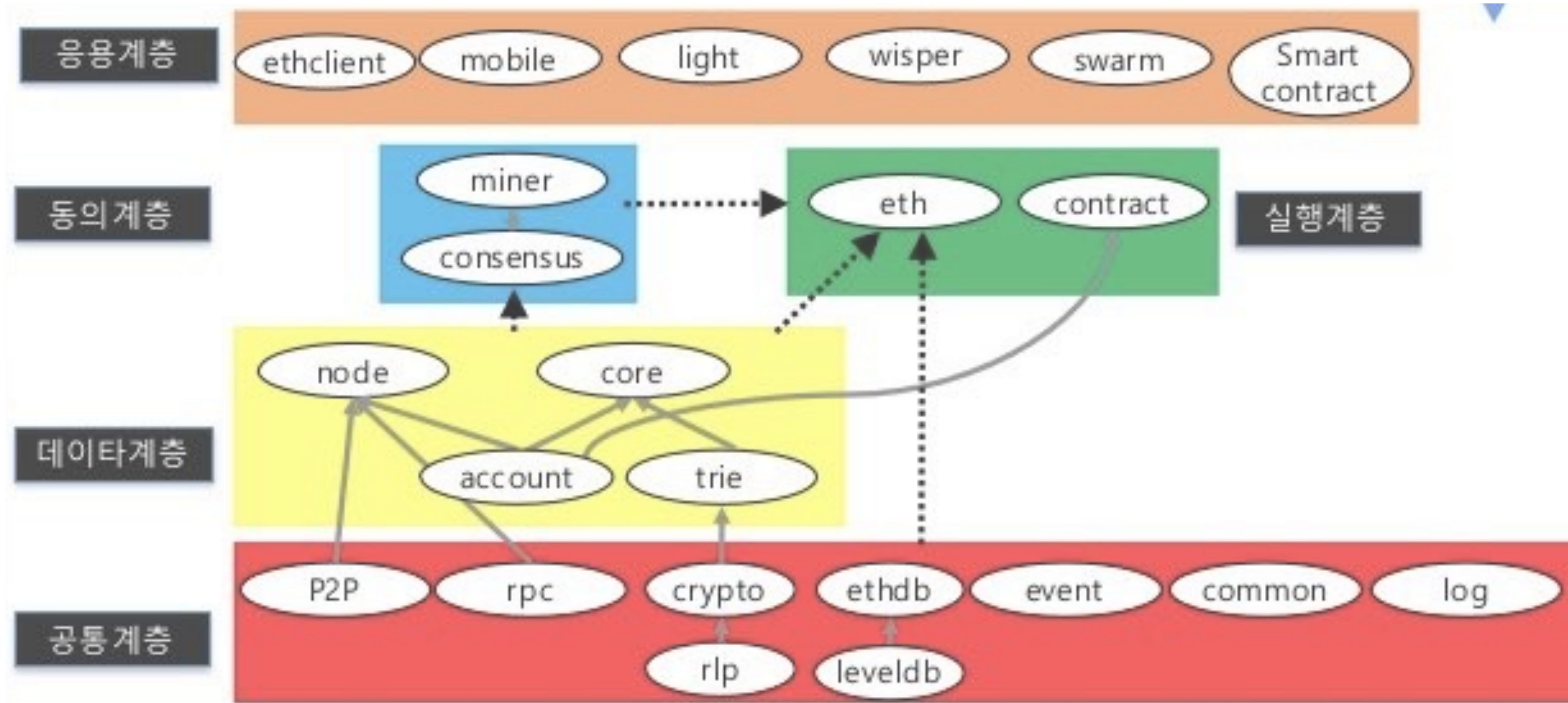
backends: con

core

miner: fix state

그럼에도,  
C++ 구현체보다 **Go** 구현체가  
널리 사용되는 이유는 무엇일까?

# Modular Design in Ethereum





# Modular Design In Ethereum

모든 내부 모듈들이 Pluggable하게 디자인됨.

별도의 SDK를 만들 필요도, 쓸 필요도 없음.

ex) Gorange —

Contract Deployment Framework

AIRBLOC

```
1 package gorange
2
3 import (
4     "fmt"
5     "log"
6     "math/big"
7
8     "github.com/ethereum/go-ethereum/accounts/keystore"
9     ethutils "github.com/ethereum/go-ethereum/common"
10    "github.com/ethereum/go-ethereum/common/fdlimit"
11    "github.com/ethereum/go-ethereum/core"
12    "github.com/ethereum/go-ethereum/eth"
13    "github.com/ethereum/go-ethereum/node"
14    "github.com/ethereum/go-ethereum/params"
15    whisper "github.com/ethereum/go-ethereum/whisper"
16 )
17
18
19 type Config struct {
20     NetworkId uint64
21     HTTPHost  string
22     HTTPPort  int
23     WSHost    string
24     WSPort    int
25     Accounts  []common.Address // count
26     Balances  int64             // ETH
27     Period    uint64
```

# Modular Design In Ethereum

모든 내부 모듈들이 Pluggable하게

별도의 SDK를 만들 필요도, 쓸 필요도

ex) Gorange —

Contract Deployment Framework

```
1 package gorange
2
3 import (
4     "fmt"
5     "log"
6     "math/big"
7
8     "github.com/ethereum/go-ethereum/accounts/keystore"
9     ethutils "github.com/ethereum/go-ethereum/cmd/utls"
10    "github.com/ethereum/go-ethereum/common"
11    "github.com/ethereum/go-ethereum/common/fdlimit"
12    "github.com/ethereum/go-ethereum/core"
13    "github.com/ethereum/go-ethereum/eth"
14    "github.com/ethereum/go-ethereum/node"
15    "github.com/ethereum/go-ethereum/params"
16    whisper "github.com/ethereum/go-ethereum/whisper/whisperv6"
17 )
18
19 type Config struct {
20     NetworkId uint64
21     HTTPHost  string
22     HTTPPort  int
23     WSHost    string
24     WSPort    int
25     Accounts  []common.Address // count
26     Balances  int64             // ETH
27     Period    uint64
```

# Back to *Airbloc*...

AIRBLOC

에어블록은 어떻게  
**Go를 쓸까요?**

AIRBLOC

# Airbloc is P2P Network

다양한 종류의 노드들이  
네트워크를 이룸.

Provider Node

Consumer Node

Identity Keeper Node

## APPLICATION

DMP

Tracker

Wallet

Apps

...

## API

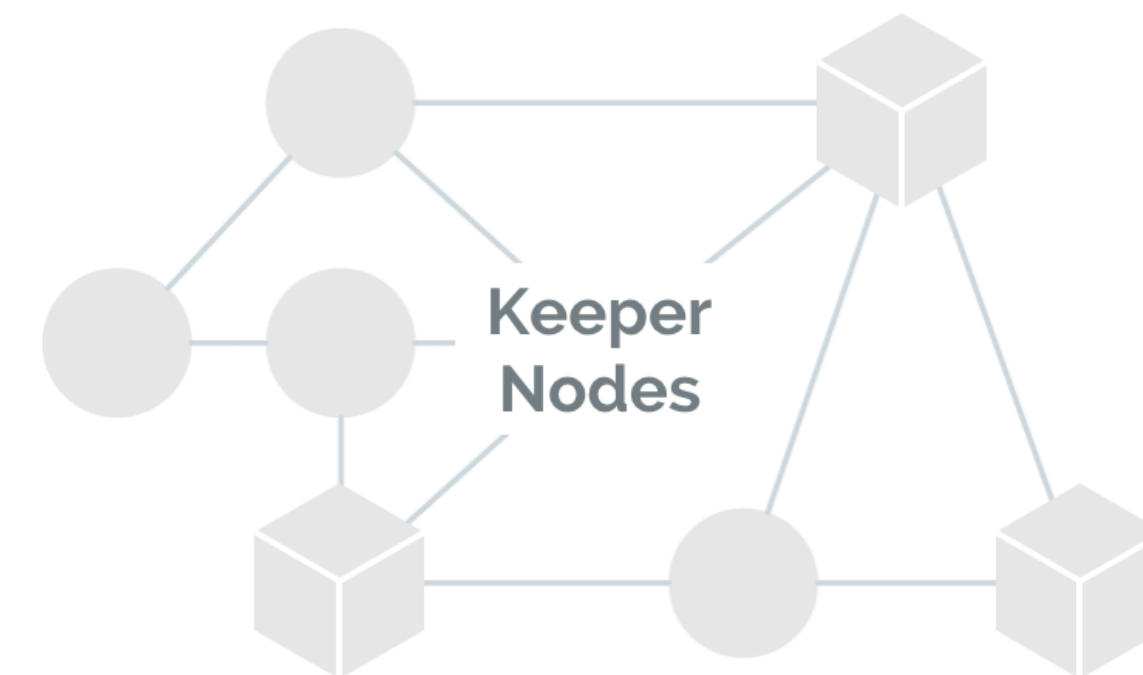
Gateway

Provider SDK

Consumer SDK

## SERVICE

### AIRBLOC



Privacy Shield

DAuth Registry

Reputation Graph

## CORE

PLASMA CHAIN

Child blockchain with main application



NUCYPHER

Re-Encryption Proxy for ACL



ETHEREUM

Root blockchain for security



BIGCHAINDB

Queryable Metadata Store



# Airbloc is P2P Network

다양한 종류의 노드들이  
네트워크를 이룸.

Provider Node

Consumer Node

Identity Keeper Node

이 노드들을 Go로 구현

## APPLICATION

DMP

Tracker

Wallet

Apps

...

## API

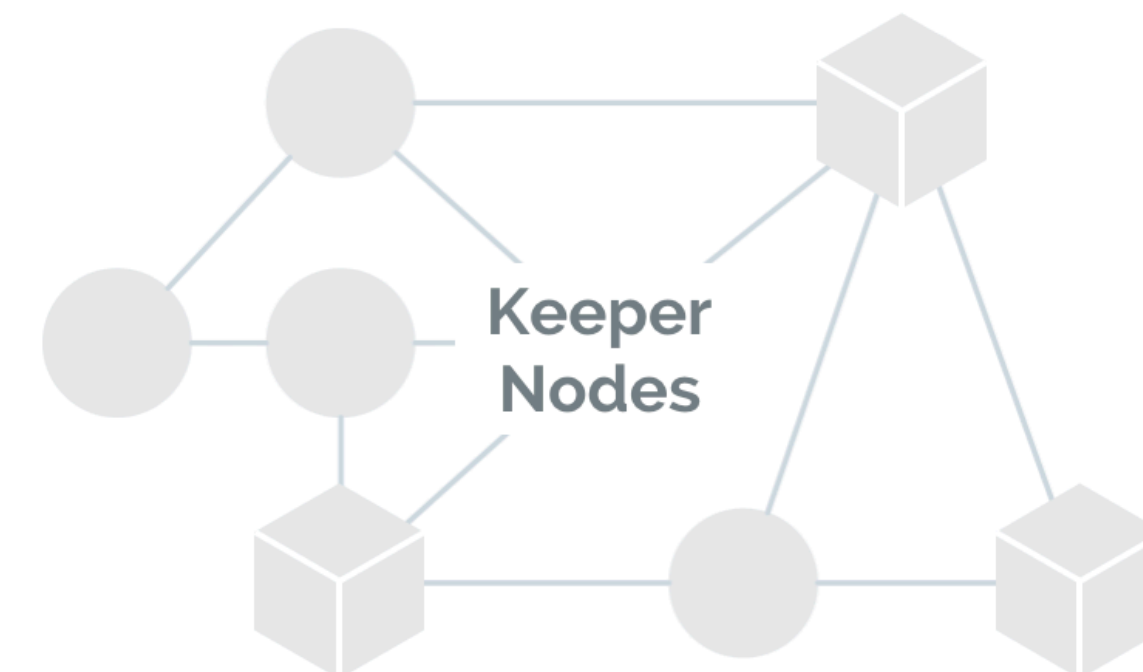
Gateway

Provider SDK

Consumer SDK

## SERVICE

### AIRBLOC



## CORE

**PLASMA CHAIN**  
Child blockchain with main application

**NUCYPHER**  
Re-Encryption Proxy for ACL

**ETHEREUM**  
Root blockchain for security

**BIGCHAINDB**  
Queryable Metadata Store

# AS-IS

—

Python으로 초기 PoC 버전을 개발함  
성능이 느리다. 암호화에 따른 오버헤드가 꽤 있었음.

# 요구사항

—

- 우린 블록체인 플랫폼을 만드는게 아니다. Ethereum을 쓴다.
- 기존 네트워크 스택을 최대한 활용한다.
- 수많은 양의 데이터를 올려야 한다. 대규모의 I/O 발생함.
- 암호화 및 영지식 증명 기술을 활용한다.

# 요구사항

—

- **우린 블록체인 플랫폼을 만드는게 아니다. Ethereum을 쓴다.**
- 기존 네트워크 스택을 최대한 활용한다.
- 수많은 양의 데이터를 올려야 한다. 대규모의 I/O 발생함.
- 암호화 및 영지식 증명 기술을 활용한다.

# 요구사항

—

- 우린 블록체
- 기존 네트워
- 수많은 양의
- 암호화 및 영

```
import (  
    "fmt"  
    "log"  
    "math/big"  
  
    "github.com/ethereum/go-ethereum/accounts/keystore"  
    ethutils "github.com/ethereum/go-ethereum/cmd/utils"  
    "github.com/ethereum/go-ethereum/common"  
    "github.com/ethereum/go-ethereum/common/fdlimit"  
    "github.com/ethereum/go-ethereum/core"  
    "github.com/ethereum/go-ethereum/eth"  
    "github.com/ethereum/go-ethereum/node"  
    "github.com/ethereum/go-ethereum/params"  
    whisper "github.com/ethereum/go-ethereum/whisper/whisperv6"  
)
```

쓴다.

# 요구사항

—

- 우린 블록체인 플랫폼을 만드는게 아니다. Ethereum을 쓴다.
- **기존 네트워크 스택을 최대한 활용한다.**
- 수많은 양의 데이터를 올려야 한다. 대규모의 I/O 발생한다.
- 암호화 및 영지식 증명 기술을 활용한다.



AIRBLOC

# 요구사항

—

- 우린 블록체인 플랫폼을 만드는게 아니다. Ethereum을 쓴다.
- 기존 네트워크 스택을 최대한 활용한다.
- **수많은 양의 데이터를 올려야 한다. 대규모의 I/O 발생함.**
- 암호화 및 영지식 증명 기술을 활용한다.

**얻을 수 있던 점**

AIRBLOC



**감사합니다!**