

분산 시스템의 어려움부터 Kafka까지 투아보기

@AUSG 7기 윤종원

분산 시스템의 어려움부터 Kafka까지 투아보기

@AUSG 7기 윤종원

발표자 소개

윤종원

- ❖ 소프트웨어학부 복수 전공
- ❖ 2021 ~ 2023 '디어코퍼레이션'에서 산업기능요원으로 근무
- ❖ AUSG 7기
- ❖ Contact : eatingcookieman@gmail.com





COMMUNITY DAY



윤종원 (AUSG)

카오스 엔지니어링과 AWS Fault Injection Simulator

분산 시스템과 마이크로서비스 아키텍처가 늘어나면서 시스템은 복잡해지고, 시스템을 테스트하는 것은 더욱 어려워졌습니다. 이를 대응하기 위한 카오스 엔지니어링은 무엇인지 살펴보고, AWS FIS를 이용해 쉽게 카오스 엔지니어링을 수행하는 방법을 알아봅니다.

AUSG 7기로 활동 중이며 '디어'에서 2년 동안 백엔드 개발자로 일했습니다. AWS DNA 4기에서 카오스엔지니어링 세션을 듣고 이를 업무에 활용해 보면서 해당 주제에 관심을 가지게 되었습니다.

목차

- 분산 시스템과 분산 알고리즘
 - 부분 실패(Partial Failure)
 - 비결정성(Nondeterminism)
- 장애 감지
 - 동기, 비동기, 부분 동기 모델
- 합의
- ZooKeeper, KRaft

분산 시스템(Distributed System)

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another

[Wikipedia](#)

분산 시스템(Distributed System)

분산 시스템(Distributed System)은 네트워크로 연결된 여러 컴퓨터에서 실행되는 여러 개의 독립적인 시스템의 집합입니다.

각 시스템은 서로 정보를 교환하고, 조정하기 위해 링크를 통해 메시지를 주고 받습니다.

분산 시스템(Distributed System)

- 분산 시스템은 여러 개의 **독립적인** 시스템의 집합입니다.
- 여러 개의 독립적인 시스템은 하나의 컴퓨터가 아니라 **네트워크**로 연결된 **여러 개의 컴퓨터**에서 실행됩니다.
- 시스템은 서로 정보를 주고받고, 조정을 위해 **링크**를 통해 메시지를 주고 받습니다.

분산 시스템(Distributed System)

- 분산 시스템은 여러 개의 **독립적인** 시스템의 집합입니다.
- 독립적인 시스템은 자신만의 **상태**를 지니고 있습니다.
- 시스템은 서로 **링크(네트워크)**를 통해 메시지를 주고 받습니다.

분산 알고리즘 (Distributed Algorithm)

- Distributed algorithms are algorithms designed to run on multiple processors, without tight centralized control.
- In general, they are harder to design and harder to understand than single-processor sequential algorithms.

Wikipedia

분산 알고리즘 (Distributed Algorithm)

- 분산 시스템을 위한 알고리즘
- 단일 머신에서 실행되는 알고리즘보다 만들기도 어렵고 이해하기도 어렵습니다

분산 시스템의 특성

- 부분 실패(Partial Failure)
- 비결정성(Nondeterminism)
- 이는 단일 컴퓨터에서 실행할 때는 나타나지 않는 특성들

부분 실패(Partial Failure)

부분 실패(Partial Failure)

In a distributed system, there will always be systems that are broken while others function normally. It is known as a **partial failure**

geeksforgeeks

부분 실패(Partial Failure)

분산 시스템에선 일부 시스템에 장애가 생기더라도 나머지 시스템이 정상 작동한다면
분산 시스템은 정상적으로 작동합니다.

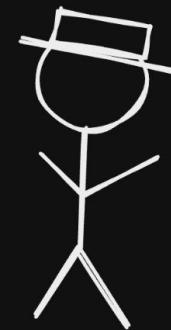
이를 **부분 실패(Partial Failure)**라고 부릅니다.

부분 실패(Partial Failure)

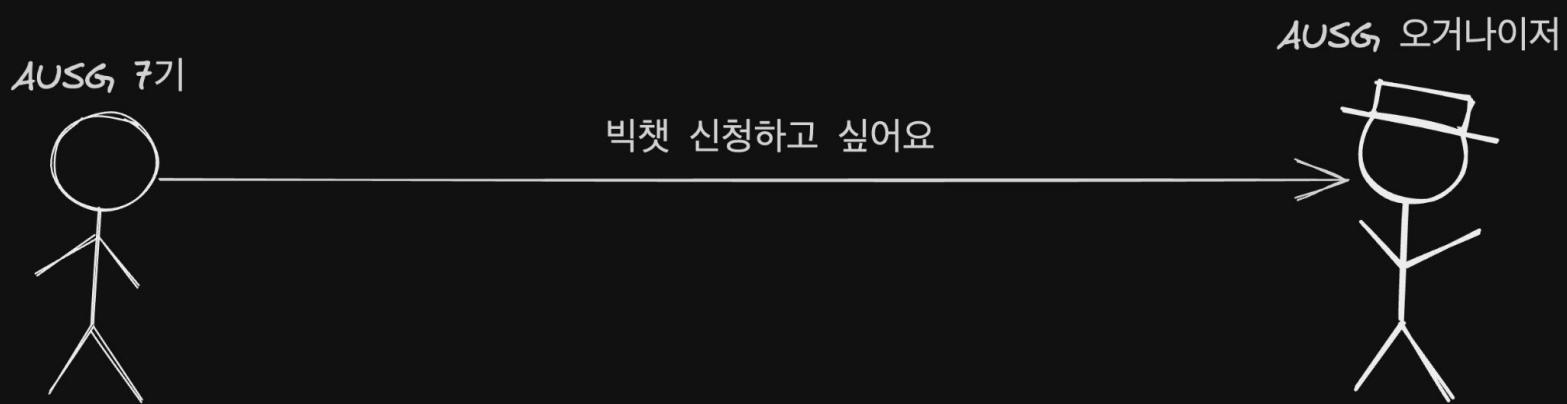
AUSG 7기



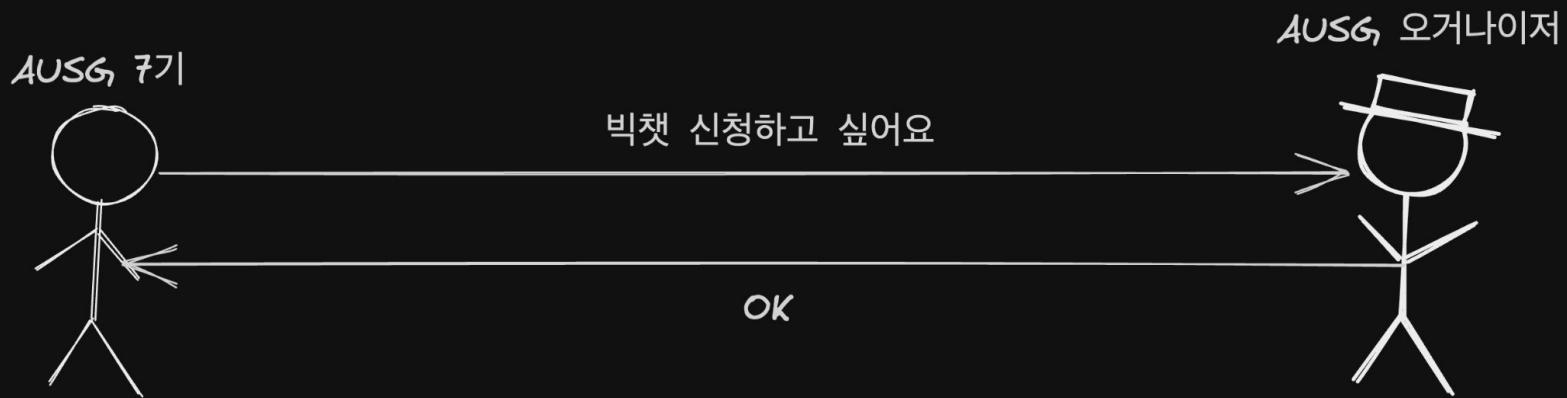
AUSG 오거나이저



부분 실패(Partial Failure)

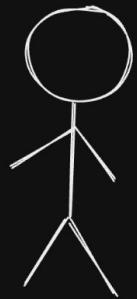


부분 실패(Partial Failure)

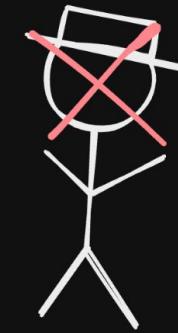


부분 실패(Partial Failure)

AUSG 7기

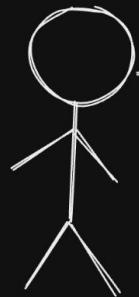


AUSG 오거나이저



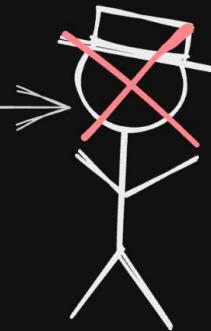
부분 실패(Partial Failure)

AUSG 7기

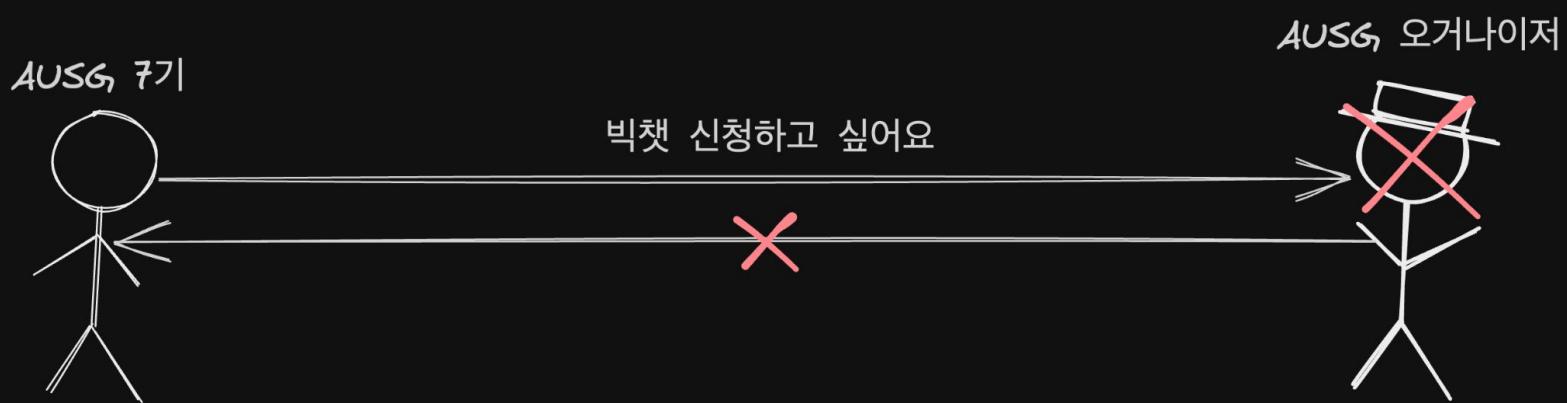


빅챗 신청하고 싶어요

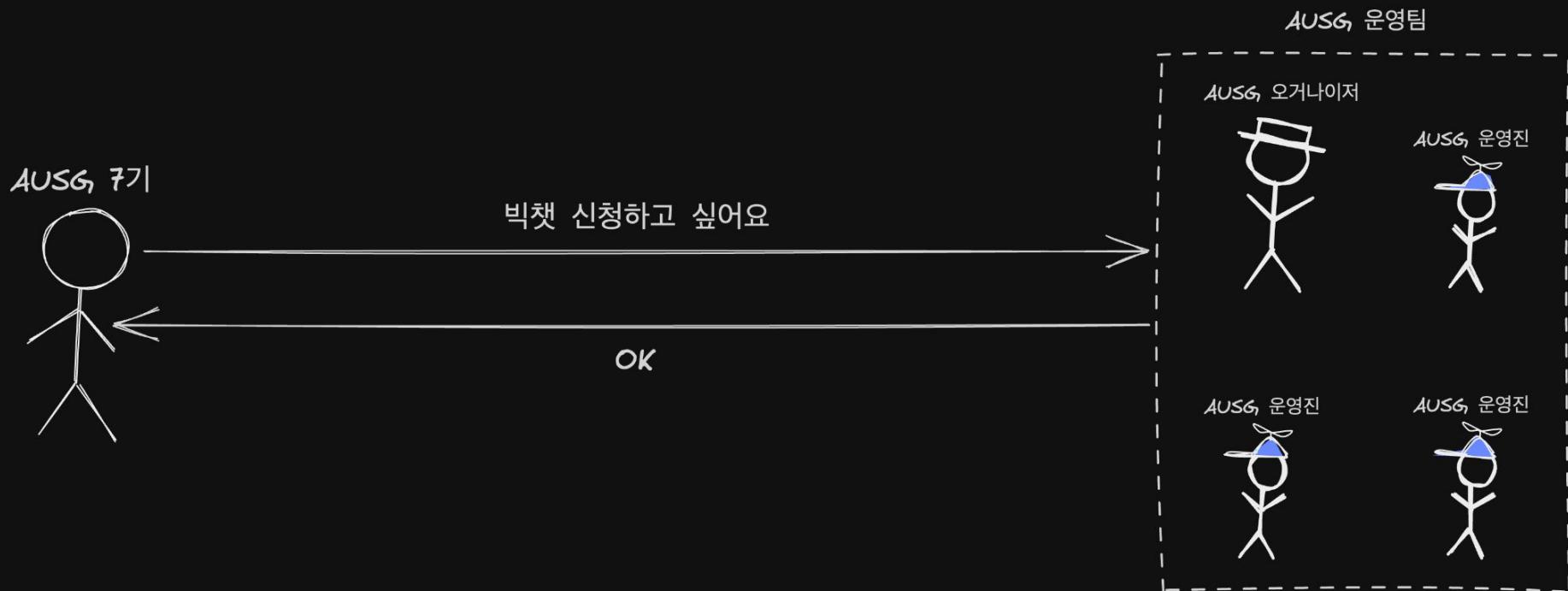
AUSG 오거나이저



부분 실패(Partial Failure)

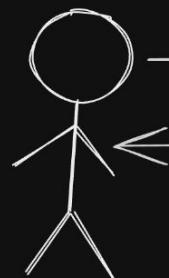


부분 실패(Partial Failure)



부분 실패(Partial Failure)

AUSG 7기



빅챗 신청하고 싶어요

OK

AUSG 운영팀

AUSG 오거나이저



AUSG 운영진



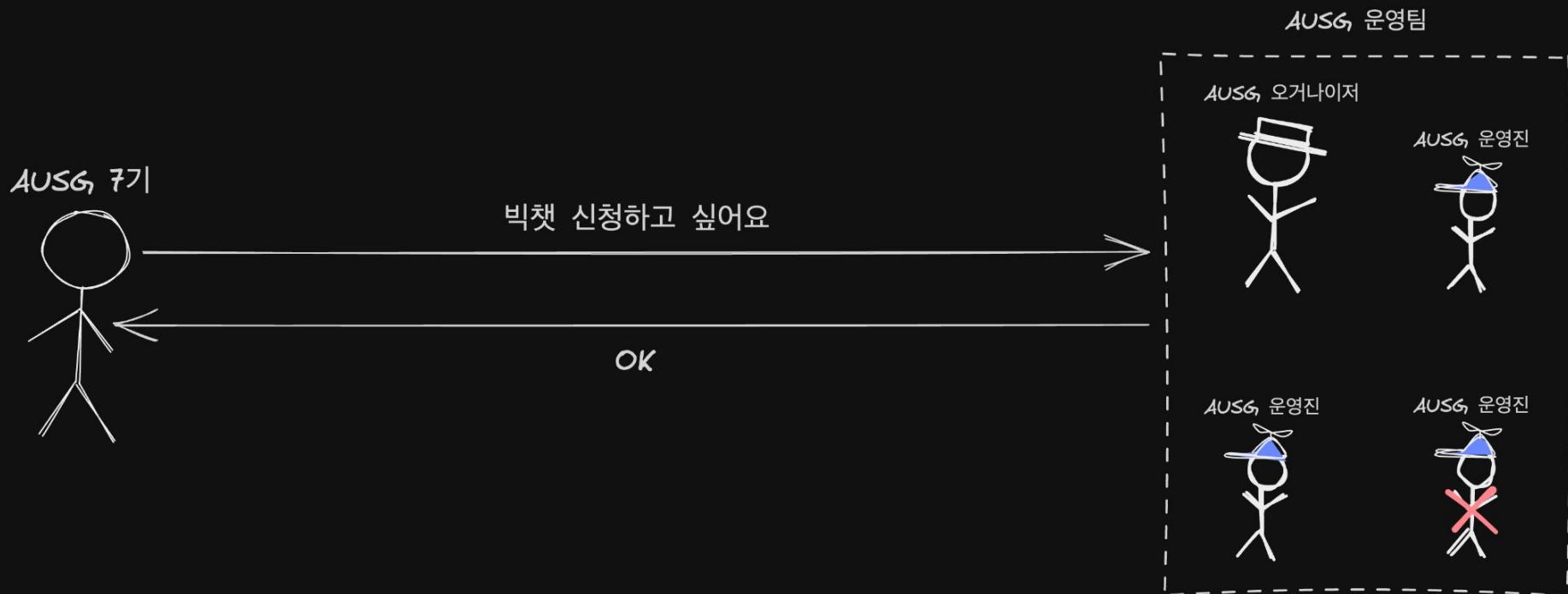
AUSG 운영진



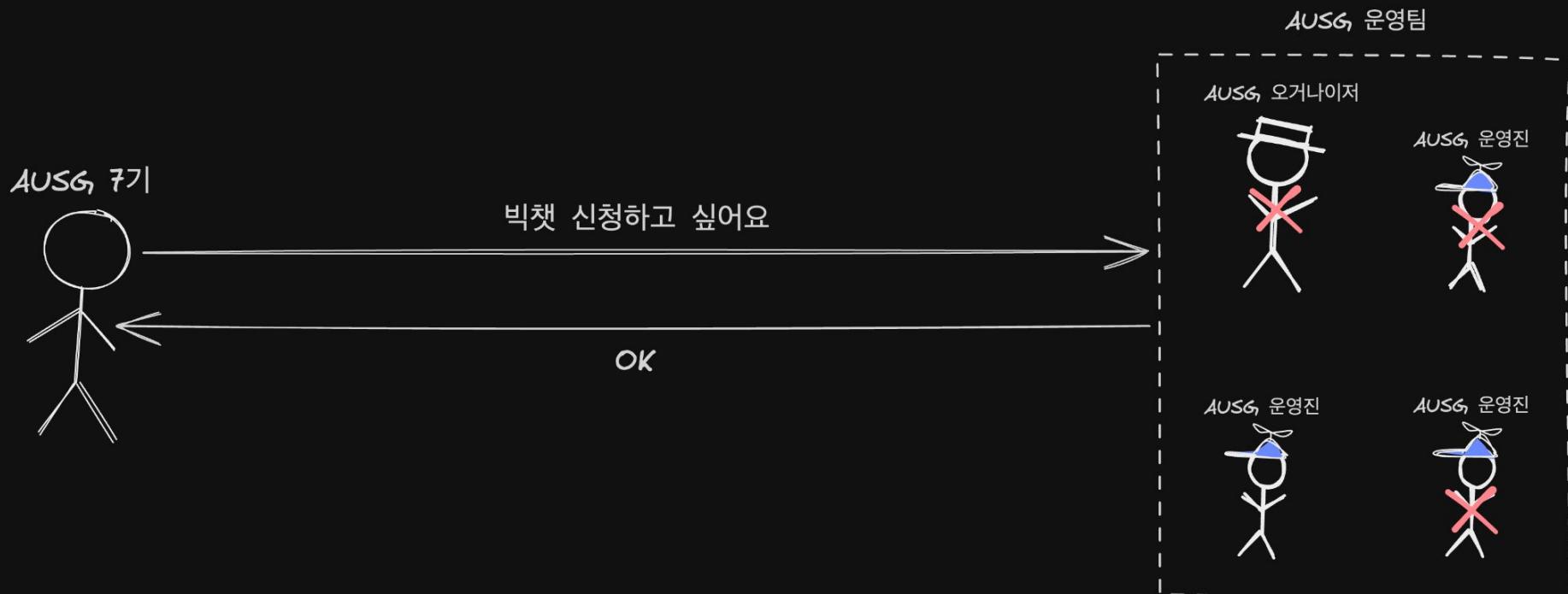
AUSG 운영진



부분 실패(Partial Failure)

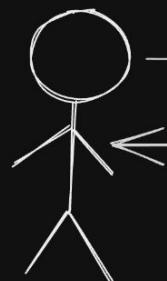


부분 실패(Partial Failure)



부분 실패(Partial Failure)

AUSG 7기



빅챗 신청하고 싶어요



AUSG 운영팀

AUSG 오거나이저



AUSG 운영진



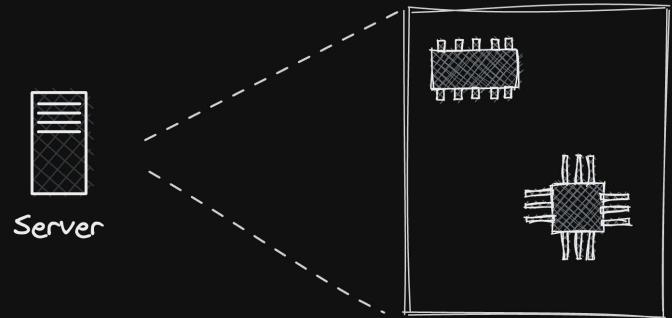
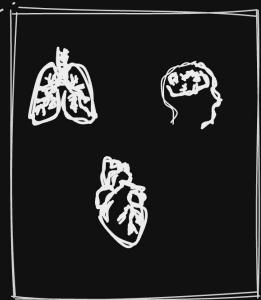
부분 실패(Partial Failure)

분산 시스템에선 일부 시스템에 장애가 생기더라도 나머지 시스템이 정상 작동한다면
분산 시스템은 정상적으로 작동합니다.

이를 **부분 실패(Partial Failure)**라고 부릅니다.

부분 실패(Partial Failure)

AUSG, 오거나이저

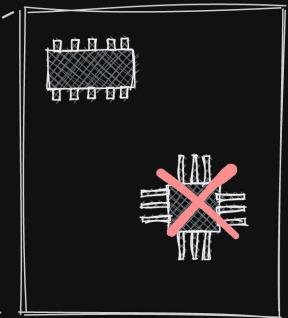


부분 실패(Partial Failure)

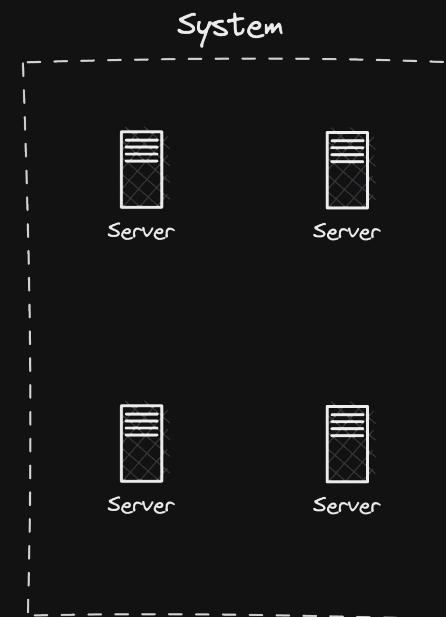
AUSG, 오거나이저



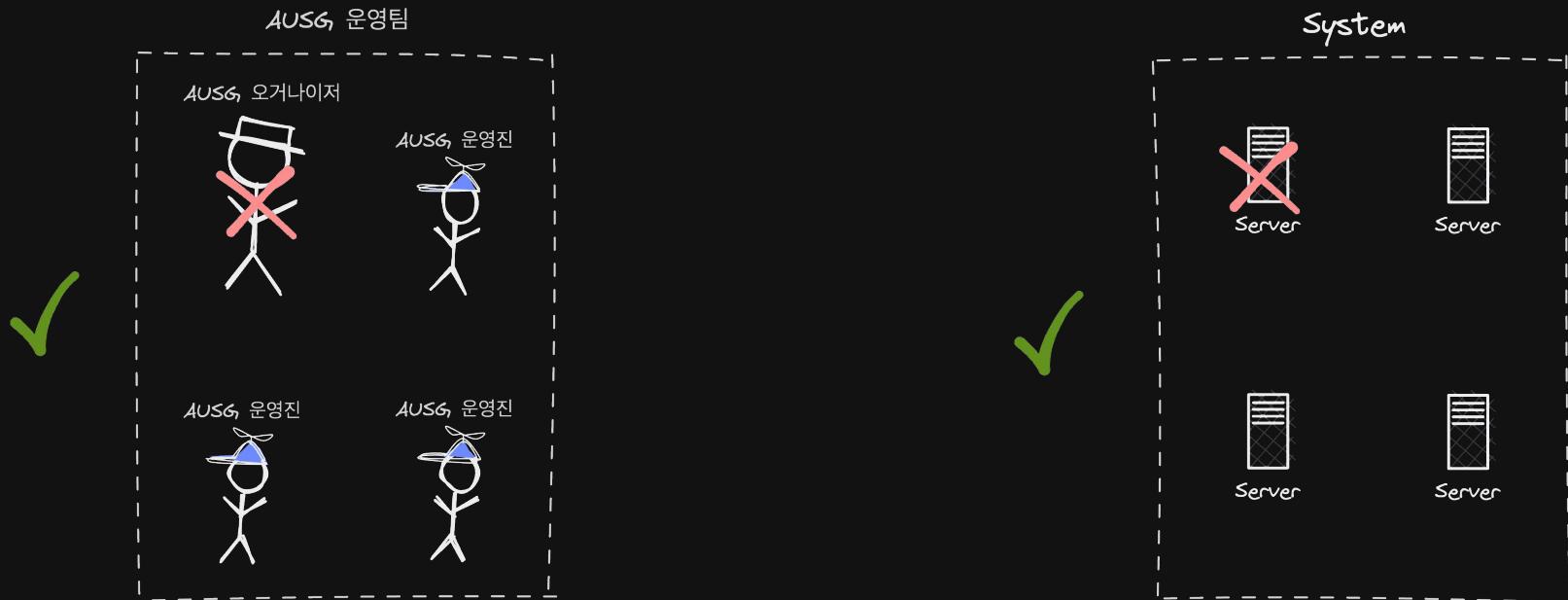
Server



부분 실패(Partial Failure)



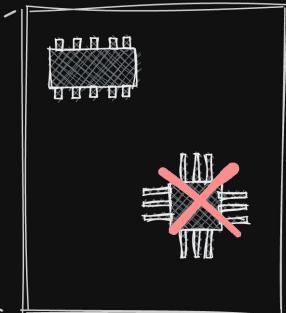
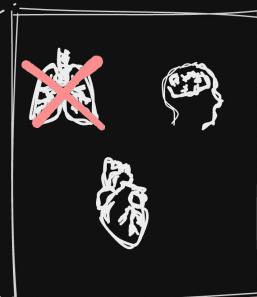
부분 실패(Partial Failure)



부분 실패(Partial Failure)

단일 시스템의 구성 요소 중 하나라도 문제가 생기면 시스템 전체가 작동을 멈춥니다.
즉, 단일 시스템에서는 **부분 실패**가 발생하지 않습니다.

AUSG 오거나이저

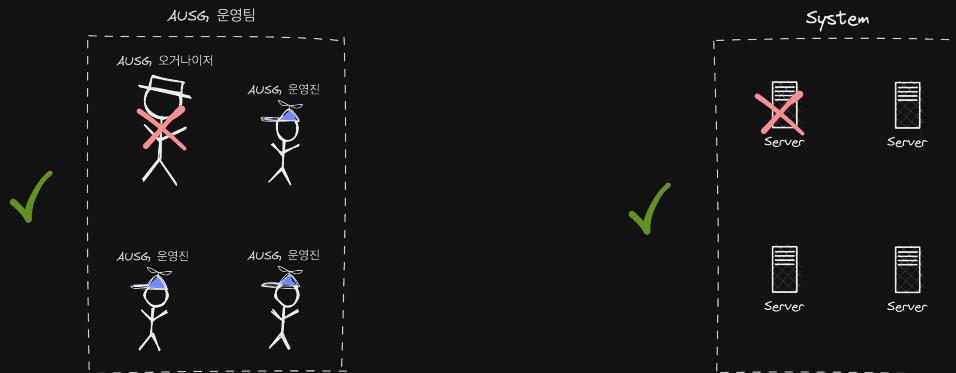


부분 실패(Partial Failure)

부분 실패는 분산 시스템을 이루고 있는 구성 요소 중 일부에 문제가 생겼을 때 발생합니다.

이를 다르게 말하면 **분산 시스템**은 단일 시스템과 다르게 구성 요소 중 **일부**에 문제가 생겨도 **정상적으로 작동**해야 합니다.

즉, 분산 시스템은 구성 요소 중 일부의 **장애**를 **감지**하고, 이에 대한 적절한 **조치**를 취할 수 있어야 합니다.
그런데 과연 분산 시스템은 **문제(장애)**를 **정확**하게 감지할 수 있을까요?

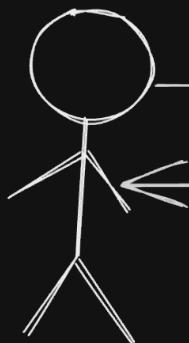


비결정성(Nondeterminism)

비결정성(Nondeterminism)



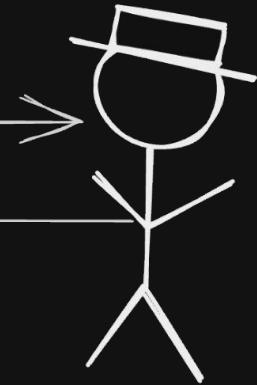
AUSG, 7기



빅챗 신청하고 싶어요



AUSG, 오거나이저



비결정성(Nondeterminism)



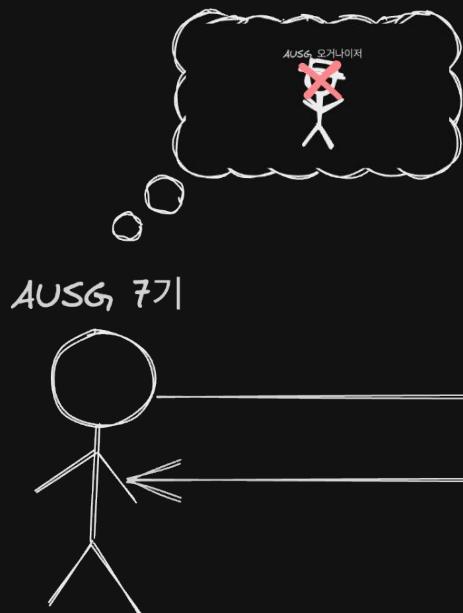
비결정성(Nondeterminism)



비결정성(Nondeterminism)

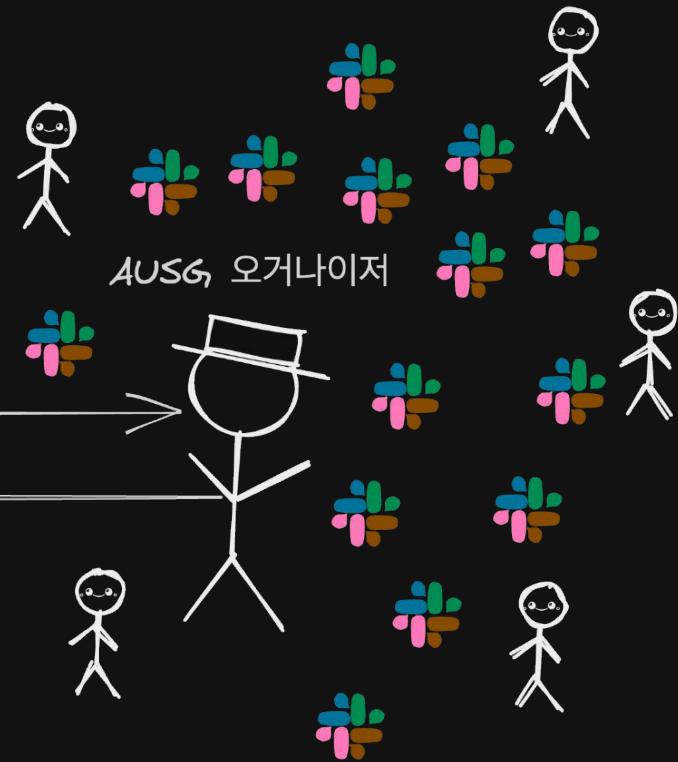


비결정성(Nondeterminism)



빅챗 신청하고 싶어요

©AUSG2023



비결정성(Nondeterminism)



비결정성(Nondeterminism)



비결정성(Nondeterminism)



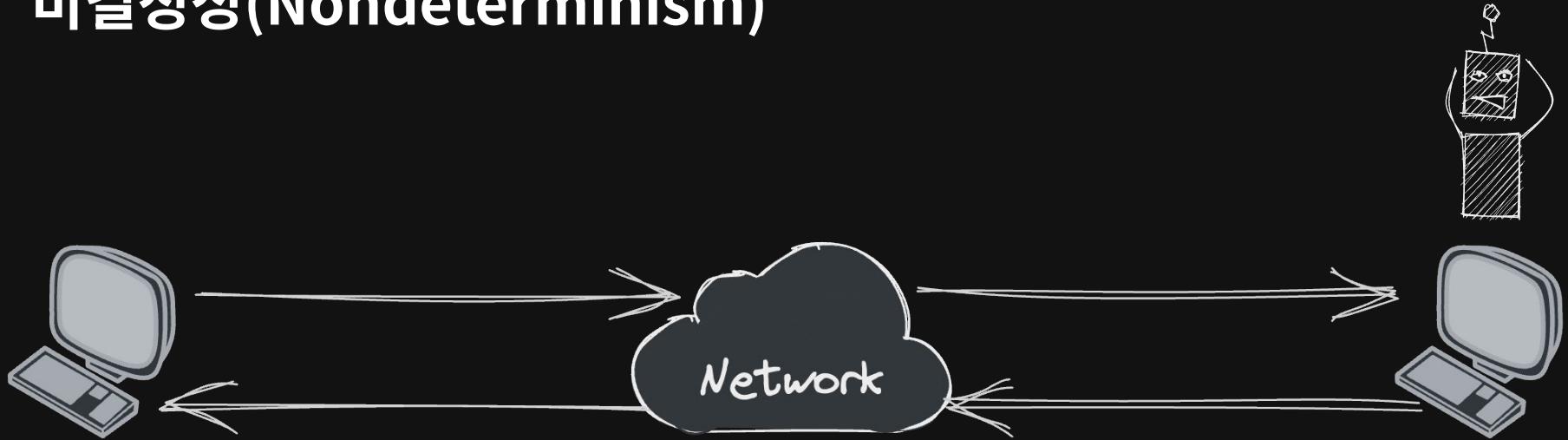
비결정성(Nondeterminism)



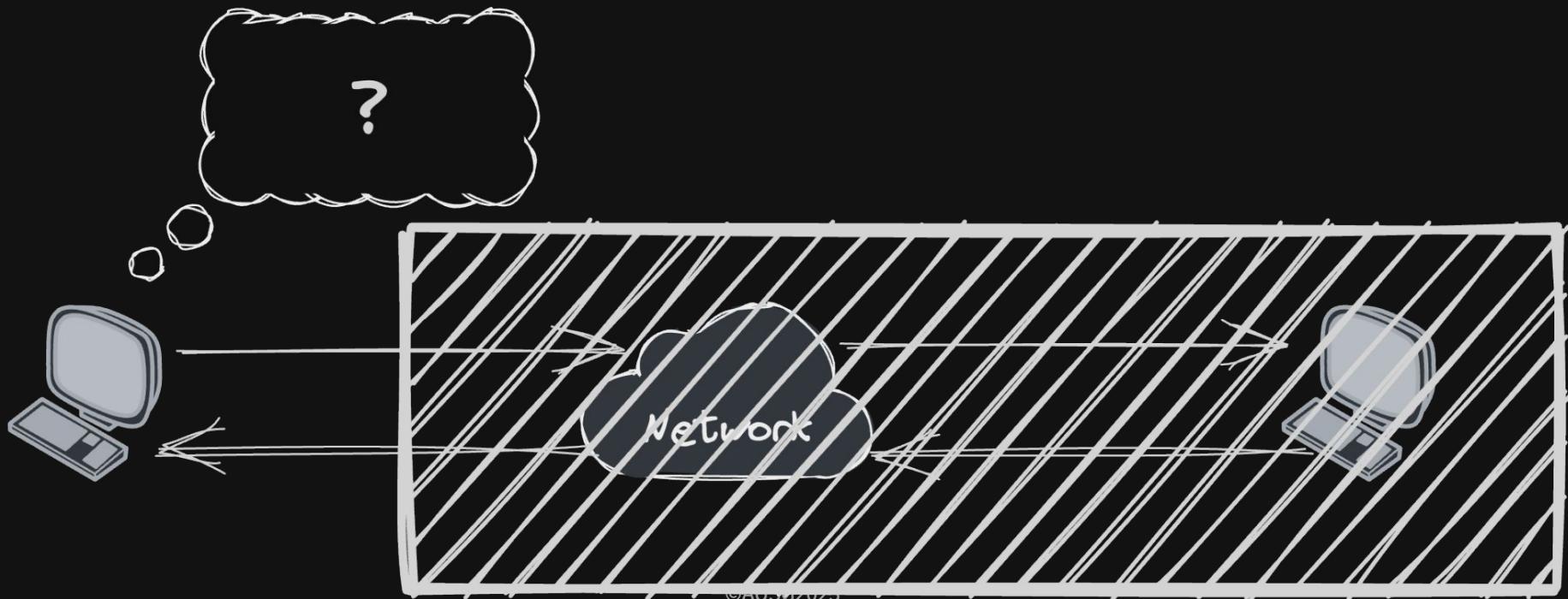
비결정성(Nondeterminism)



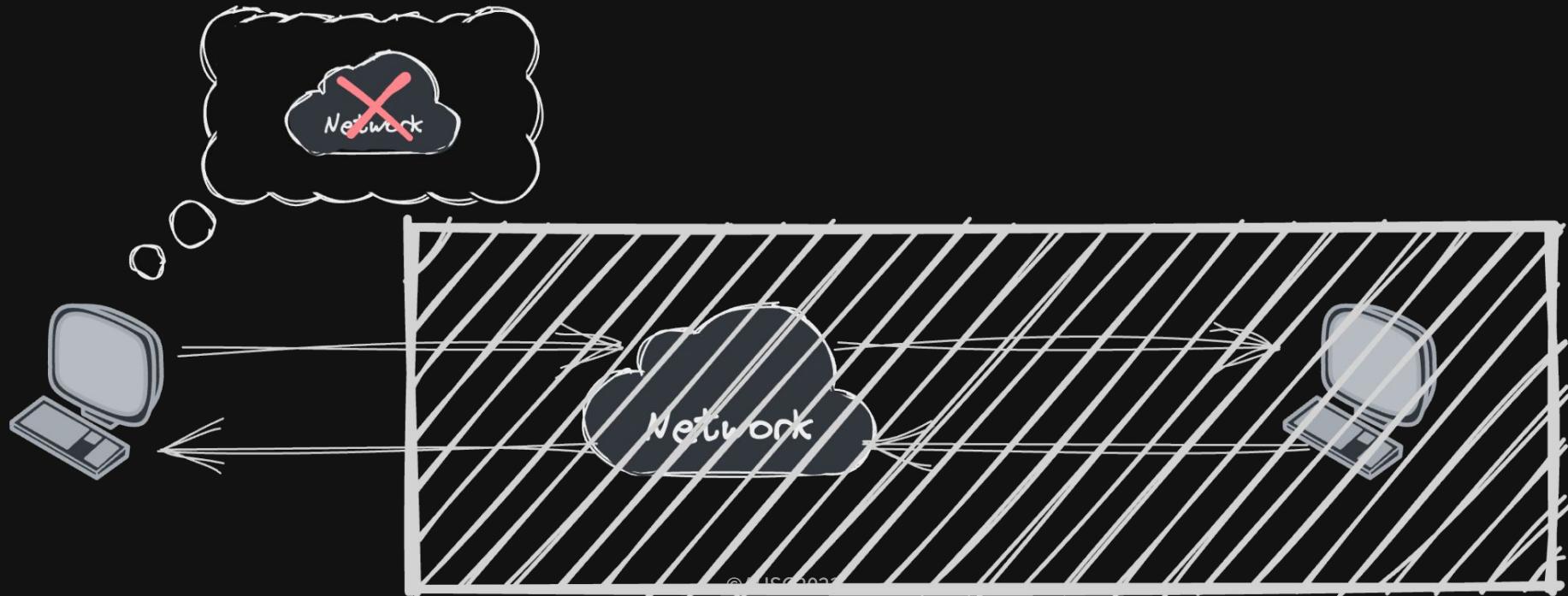
비결정성(Nondeterminism)



비결정성(Nondeterminism)



비결정성(Nondeterminism)

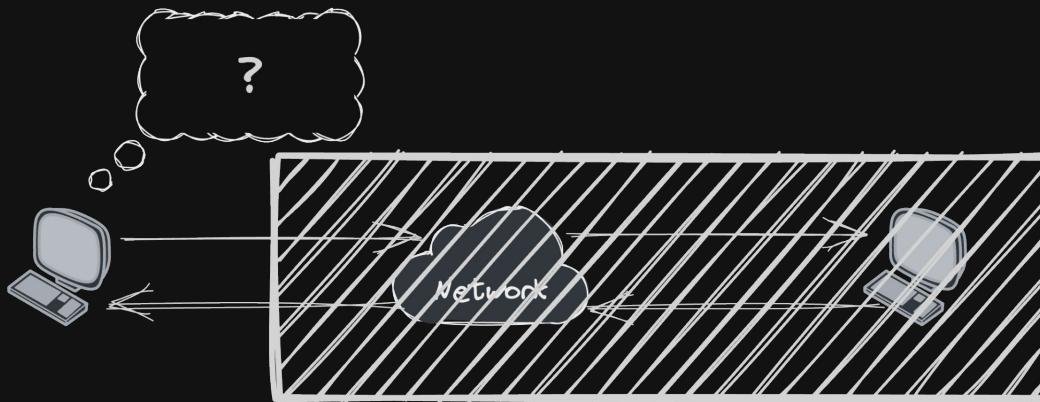


비결정성(Nondeterminism)

분산 시스템의 노드들은 네트워크를 통해 연결됩니다.

네트워크의 불확실성 때문에 전송 측은 패킷이 제대로 전달되었는지 여부조차 알 수 없습니다.

따라서 노드가 작동 중인지, 문제가 생겼는지조차 구별하기 어렵습니다.



부분 실패(Partial Failure)

분산 시스템에선 일부 시스템에 장애가 생기더라도 나머지 시스템이 정상 작동한다면
분산 시스템은 정상적으로 작동합니다.

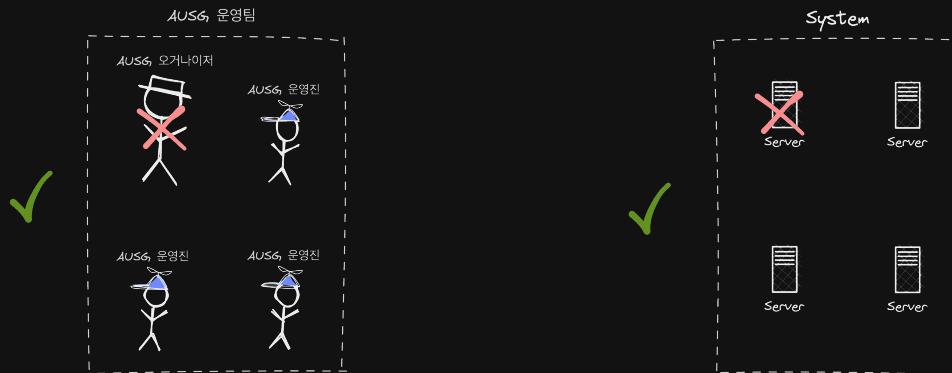
이를 **부분 실패(Partial Failure)**라고 부릅니다.

부분 실패(Partial Failure)

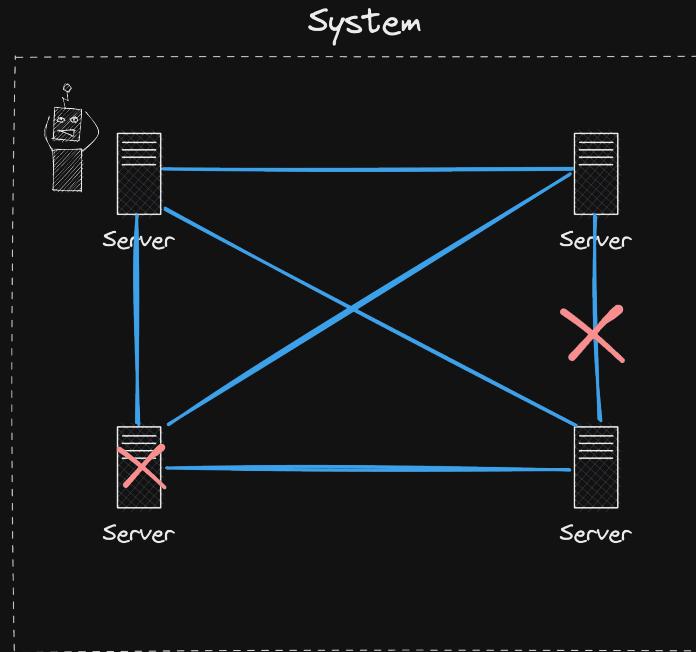
부분 실패는 분산 시스템을 이루고 있는 구성 요소 중 일부에 문제가 생겼을 때 발생합니다.

이를 다르게 말하면 **분산 시스템**은 단일 시스템과 다르게 구성 요소 중 **일부**에 문제가 생겨도 **정상적으로 작동**해야 합니다.

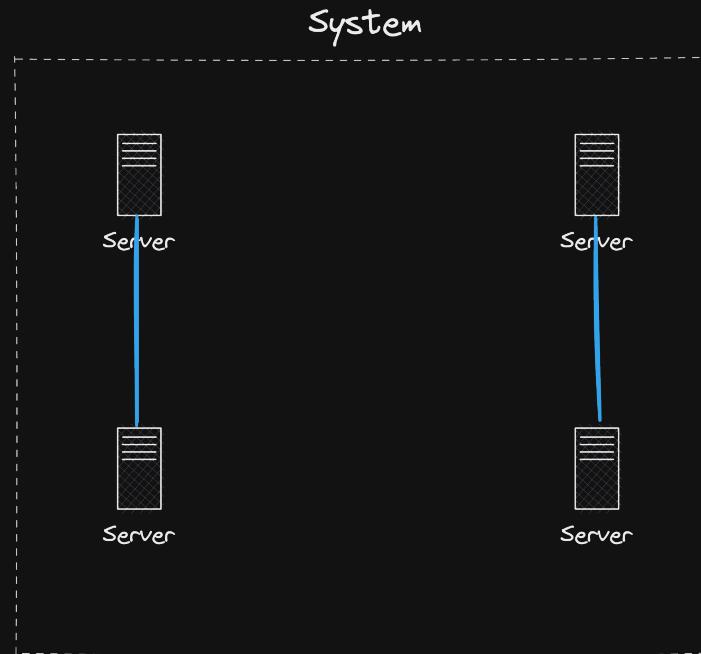
즉, 분산 시스템은 구성 요소 중 일부의 **장애를 감지**하고, 이에 대한 적절한 **조치**를 취할 수 있어야 합니다.
그런데 과연 분산 시스템은 **문제(장애)**를 **정확**하게 감지할 수 있을까요?



부분 실패와 비결정성



부분 실패와 비결정성



장애 감지

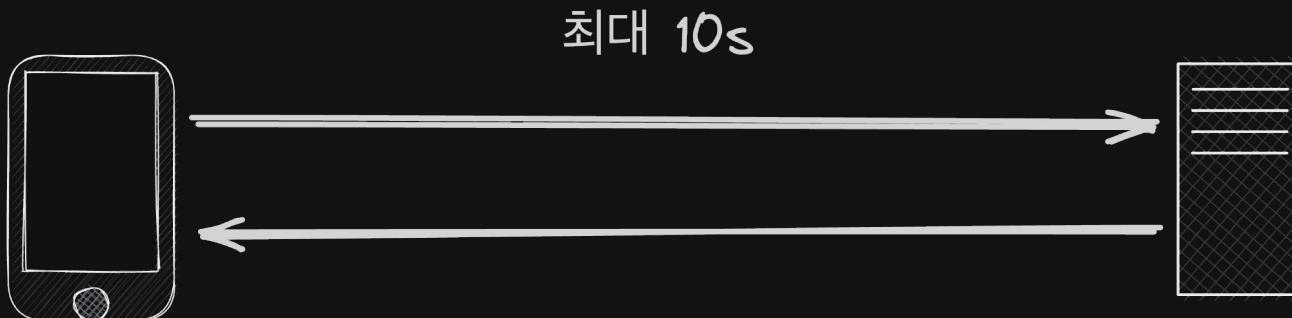
504 Gateway Time-out

nginx



네트워크의 동기 모델

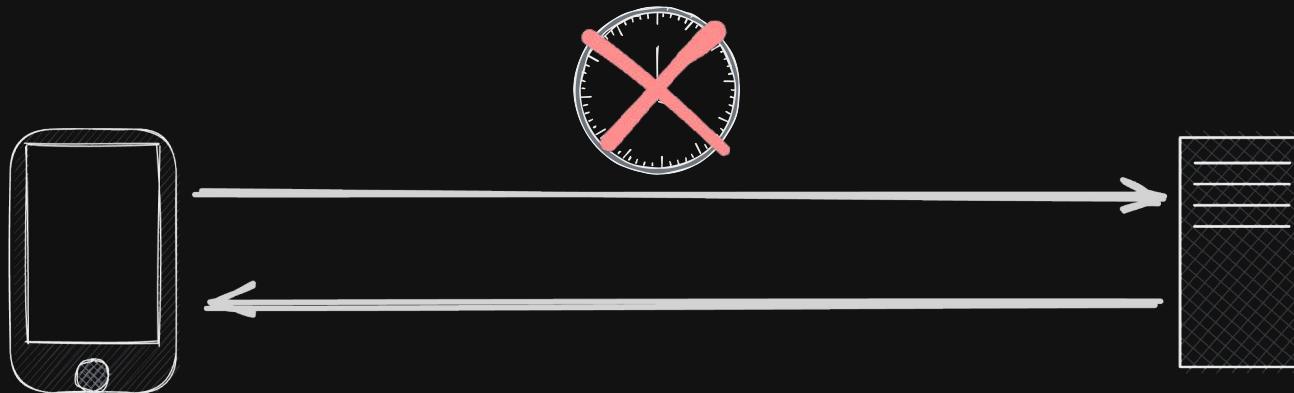
네트워크 지연, 프로세스 중단, 시계 오차에 모두 제한이 있다고 가정
즉, 모든 메시지가 **x 초 안**에 도착하는 것이 보장되는 모델



네트워크의 비동기 모델

타이밍에 대한 어떠한 가정도 할 수 없는 모델

즉, 메시지가 정해진 시간 안에 도착하는 것을 보장할 수 없는 모델

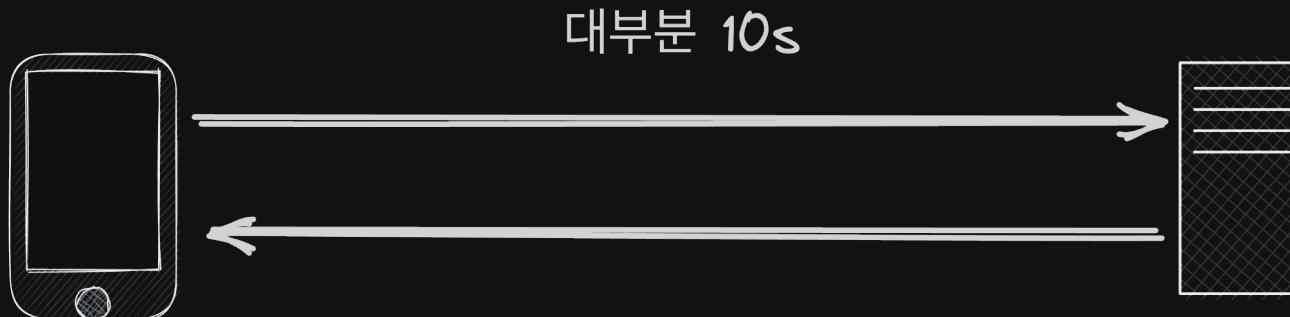


네트워크의 부분 동기 모델

대부분 동기 모델처럼 동작

그러나 가끔 네트워크 지연, 프로세스 중단, 시계 오차로 인해 정해진 시간 안에 메시지가 도착하지 않을 수 있는 모델

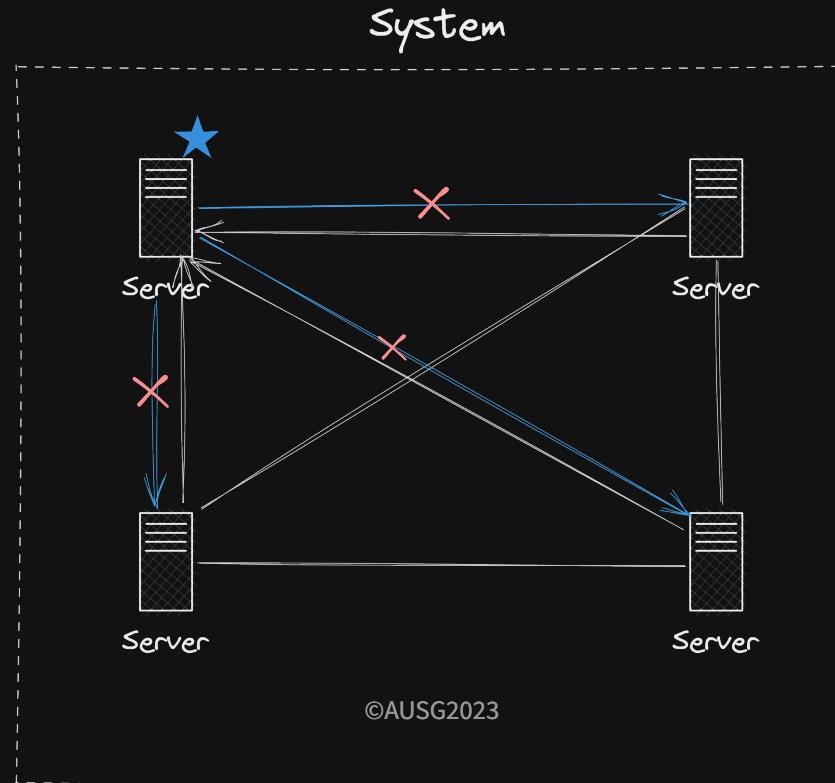
-> 현실적인 모델



장애 감지

- 요청 응답에 걸리는 시간을 기준으로 판단
- 다양한 기법들
 - Ping
 - Heartbeat
 - Heartbeat outsourcing
 - ϕ -accrual detection
 - Gossip
 - FUSE(Failure Notification Service)

정족수(Quorum)



정족수(Quorum)

분산 시스템에서 각 노드의 판단은 틀릴 수 있습니다.

따라서 분산 알고리즘에서 어떠한 결정을 내릴 때 종종 **투표**에 의존합니다.

정족수(Quorum)

“합의체가 의사를 진행하고 결정하는 데에 필요한 최소한의 출석 인원”

정족수(Quorum)

분산 시스템이 결정을 내리기 위해서 필요한 최소한의 동의 수
보통 과반수

분산 시스템의 특성

- 부분 실패(Partial Failure)
- 비결정성(Nondeterminism)

분산 시스템의 특성

- 부분 실패(Partial Failure)
- 비결정성(Nondeterminism)
- 신뢰성 없는 시계, NTP, Google True Time
- 비잔틴 장애 허용(BFT; Byzantine Fault Tolerance)
- 동기 모델, 부분 동기 모델, 비동기 모델
- 장애 감지
 - Heartbeat
 - ϕ Accrual Failure Detector
 - Gossip

합의(Consensus)

분산 시스템에서 합의는 왜 필요한가?

분산 시스템에서 합의는 왜 필요한가?

- 리더 선출(Leader Election)
- 분산 시스템 간의 일관된 데이터
- 멤버십 관리
- 등등..

분산 시스템에서 합의는 왜 필요한가?

합의 : 분산 시스템의 노드들이 무언가에 동의하는 것

분산 시스템에서 합의는 왜 필요한가?

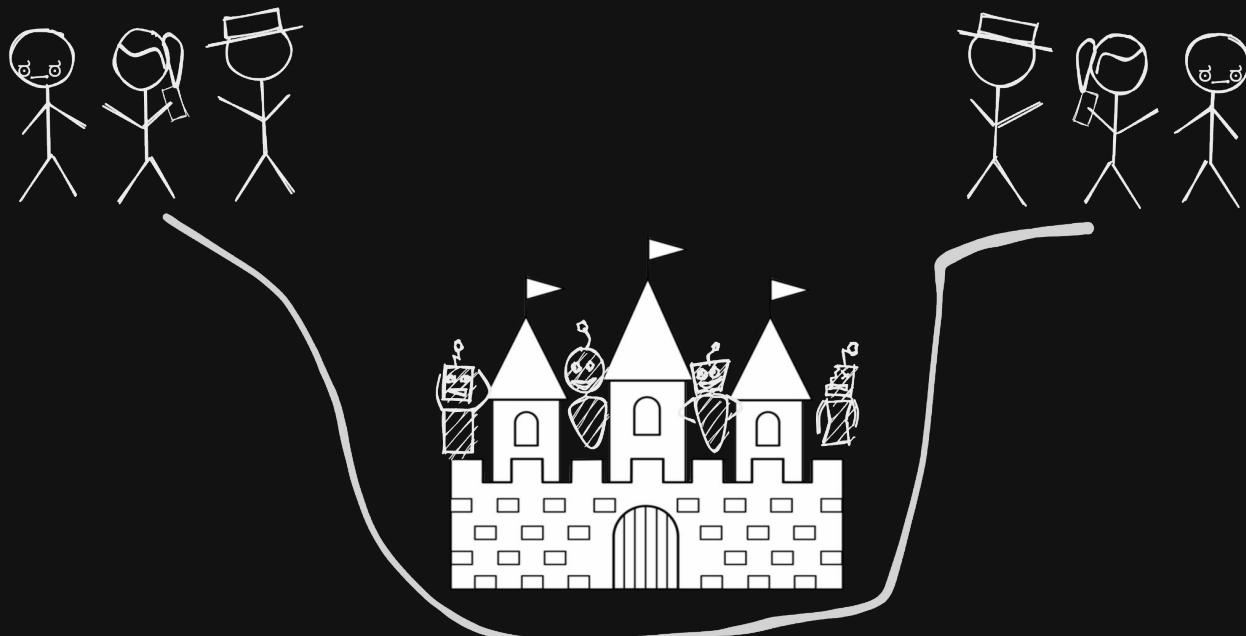
합의 : 분산 시스템의 노드들이 무언가에 동의해 값(상태)을 결정하는 것

분산 시스템에서 합의는 왜 필요한가?

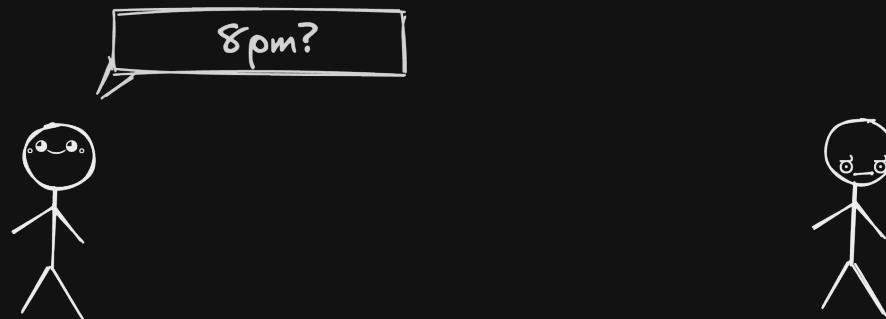
- 리더 선출(Leader Election)
 - Leader : Broker1
- 분산 시스템 간의 일관된 데이터
 - Cluster Name : My-Cluster
- 멤버십 관리
 - Members : [Broker1, Broker2, Broker3]

왜 분산 시스템에서 합의가 어려운가?

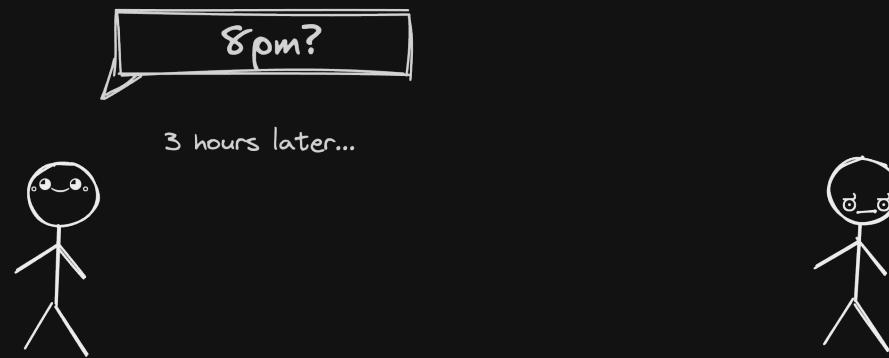
두 장군 문제(Two General Problem)



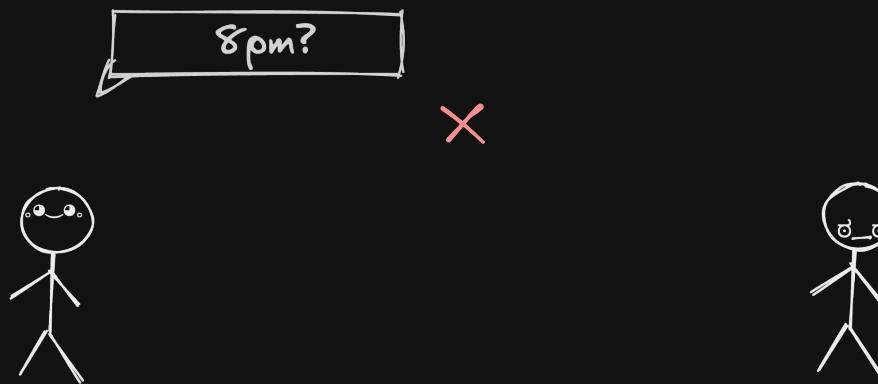
두 장군 문제(Two General Problem)



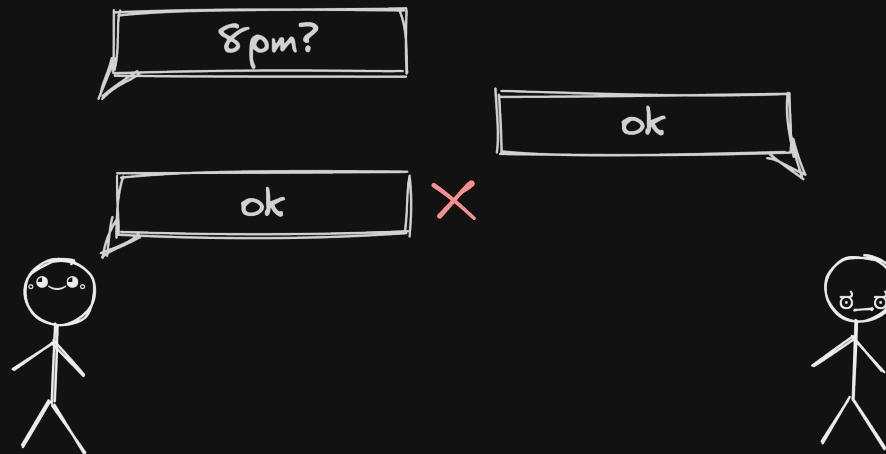
두 장군 문제(Two General Problem)



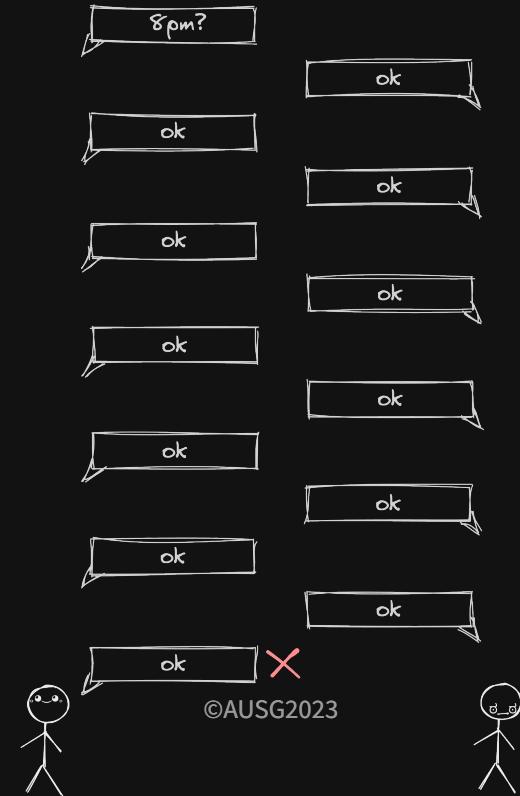
두 장군 문제(Two General Problem)



두 장군 문제(Two General Problem)



두 장군 문제(Two General Problem)



두 장군 문제(Two General Problem)

신뢰할 수 없는 두 네트워크 간의 합의는 불가능하다

두 장군 문제(Two General Problem)

<https://www.youtube.com/watch?v=s8Wbt0b8bwY>

<https://www.youtube.com/watch?v=IP-rGJKSZ3s>

비잔틴 장군 문제(Byzantine Generals Problem)

- 두 명 이상의 장군이 존재하고
- 그 사이에 배신자가 존재할 수 있을 때
- 충직한 장군끼리 합의를 이뤄내는 문제

FLP Impossibility

비동기 네트워크 모델에서 하나의 노드라도 fail-stop(단순 크래시)하면
합의가 불가능하다

[Impossibility of Distributed Consensus with One Faulty Process](#)
By MICHAEL J. FISCHER , NANCY A. LYNCH, MICHAEL S. PATERSON

FLP Impossibility

- 비동기 네트워크 모델에서
- 다음 3가지 속성을 모두 만족하는 합의 알고리즘은 불가능하다
- Liveness
- Safety
- Fault Tolerance

FLP Impossibility

- 비동기 네트워크 모델에서
- 다음 3가지 속성을 모두 만족하는 합의 알고리즘은 불가능하다
- **Liveness**
- Safety
- **Fault Tolerance**
- -> Liveness over Safety
 - PoW(작업 기반 증명)

FLP Impossibility

- 비동기 네트워크 모델에서
- 다음 3가지 속성을 모두 만족하는 합의 알고리즘은 불가능하다
- Liveness
- Safety
- Fault Tolerance
- -> Safety over Liveness
 - PBFT(Practical Byzantine Fault Tolerance) 계열

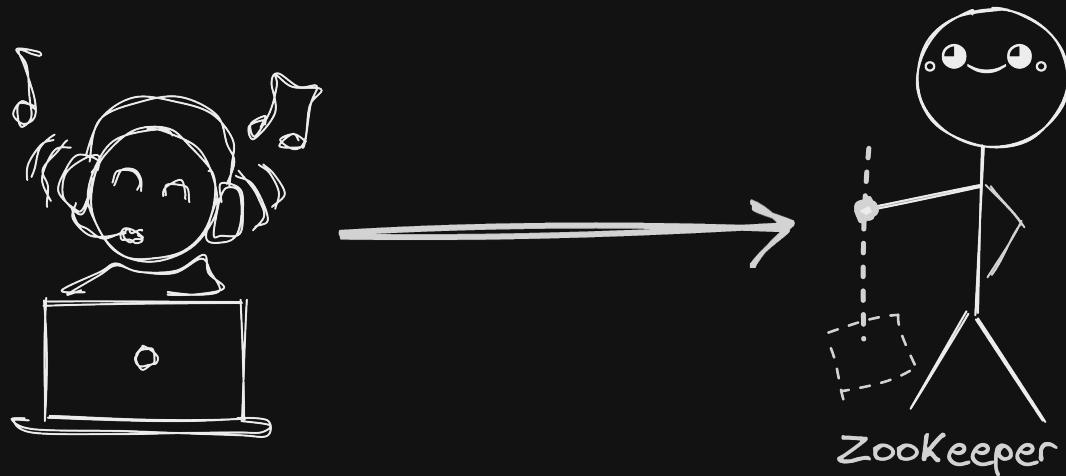
FLP Impossibility

- 비동기 네트워크 모델에서
- 다음 3가지 속성을 모두 만족하는 합의 알고리즘은 불가능하다
- **Liveness**
- **Safety**
- **Fault Tolerance**

분산 시스템 만들기 너무 어렵다...



분산 시스템 만들기 너무 어렵다...



Apache ZooKeeper

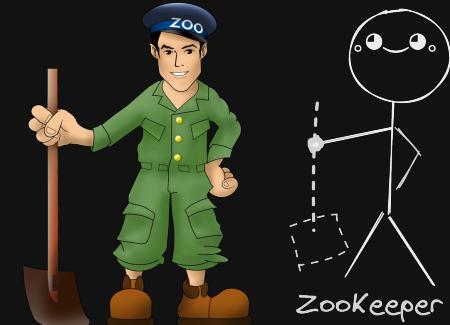
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

All of these kinds of services are used in some form or another by distributed applications.

Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.

Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage.

Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.



[Apache Zookeeper](#)

Apache ZooKeeper

- 다음 기능들은 분산 시스템에서 자주 쓰입니다.
 - 설정 정보 관리(maintaining configuration information)
 - 분산 동기화(Distributed Synchronization)
 - Naming service
 - Group Service
- 하지만 구현도 어렵고, 문제가 생기기도 쉽습니다.
- ZooKeeper는 centralized service로 이러한 기능들을 제공해줍니다.

Apache ZooKeeper

- 결국 ZooKeeper는 클러스터 간의 동기화된 **상태**를 제공합니다
 - 현재 클러스터의 리더는 누구지?
 - 현재 클러스터의 Node1이 맡은 작업이 뭐지?
 - 현재 클러스터 내에 살아있는 노드가 누구누구지?

ZooKeeper가 제공하는 API?

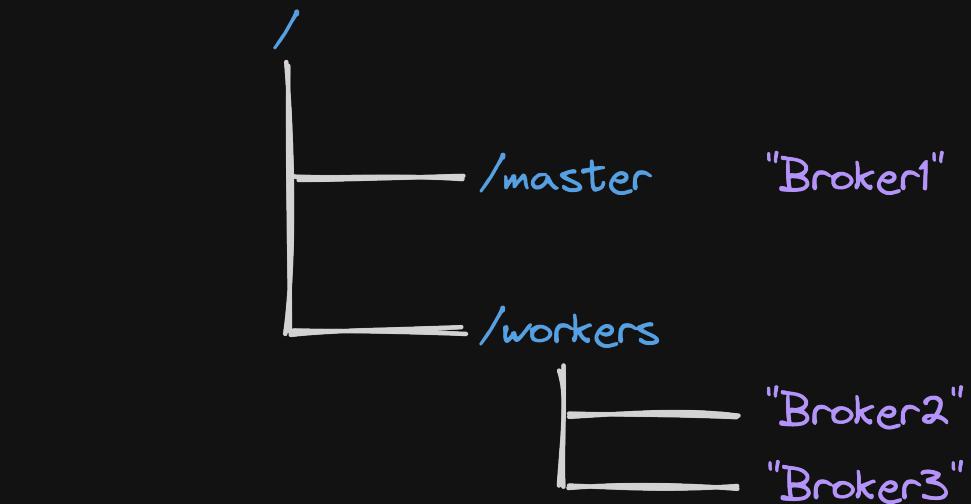
- 그렇다면 ZooKeeper는 어떠한 API를 제공하고 있을까요?
 - electNewLeader()?
 - getAliveNodes()?
 - leaveGroup()?
 - joinGroup()?

ZooKeeper가 제공하는 API?

- ZooKeeper는 일종의 작은 분산 파일시스템에 대한 API를 제공합니다.
 - 강한 일관성(Consistency)이 곁들여진
- 이때 각 파일을 “file”이 아니라 “znode”라고 부릅니다.

ZooKeeper가 제공하는 API?

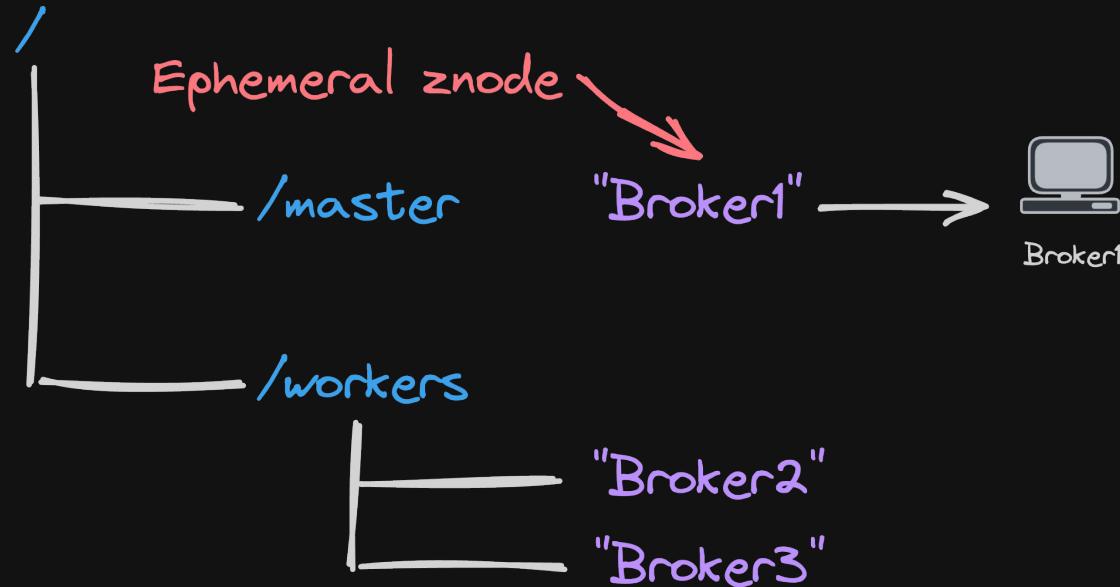
- ZooKeeper가 제공하는 API
 - Create, delete, exists, setData, getData, getChildren



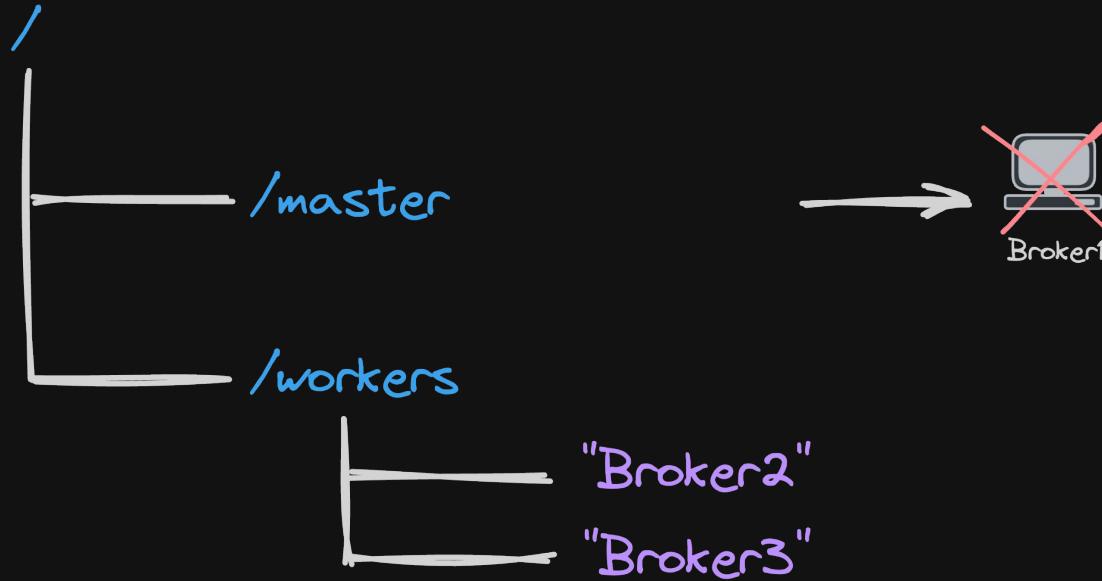
ZooKeeper가 제공하는 API?

- Persistent znode
 - 삭제하기 전까지 유지되는 영구적인 znode
- **Ephemeral znode**
 - znode를 생성한 클라이언트가 연결이 끊기면 사라지는 znode
 - 즉, znode의 존재로 특정 클라이언트가 연결되어 있는지 알 수 있습니다

ZooKeeper를 이용한 리더 선출



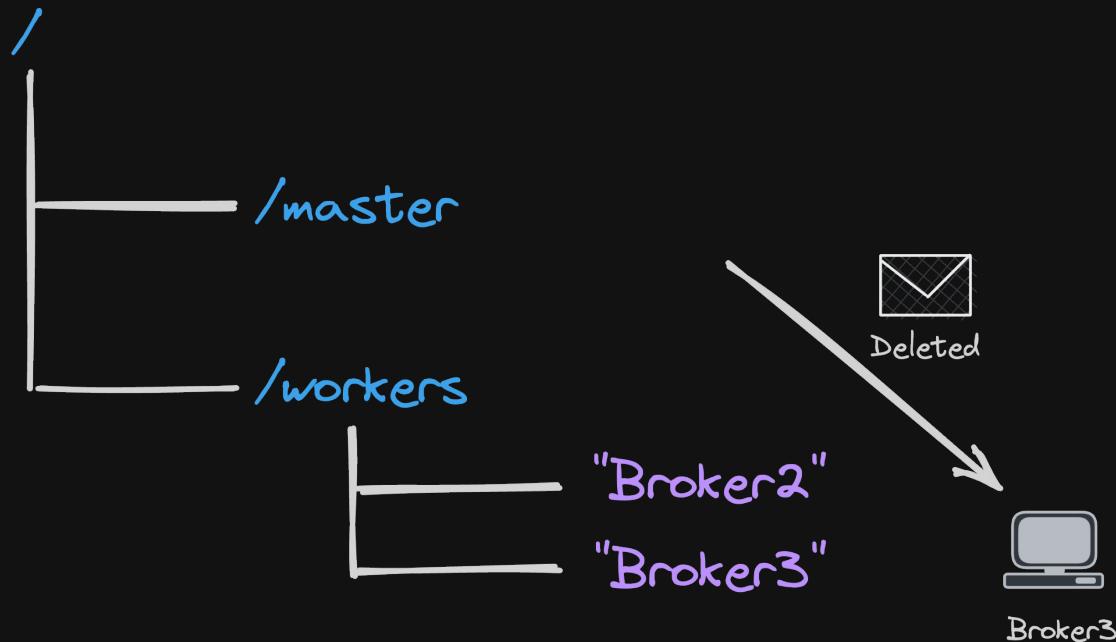
ZooKeeper를 이용한 리더 선출



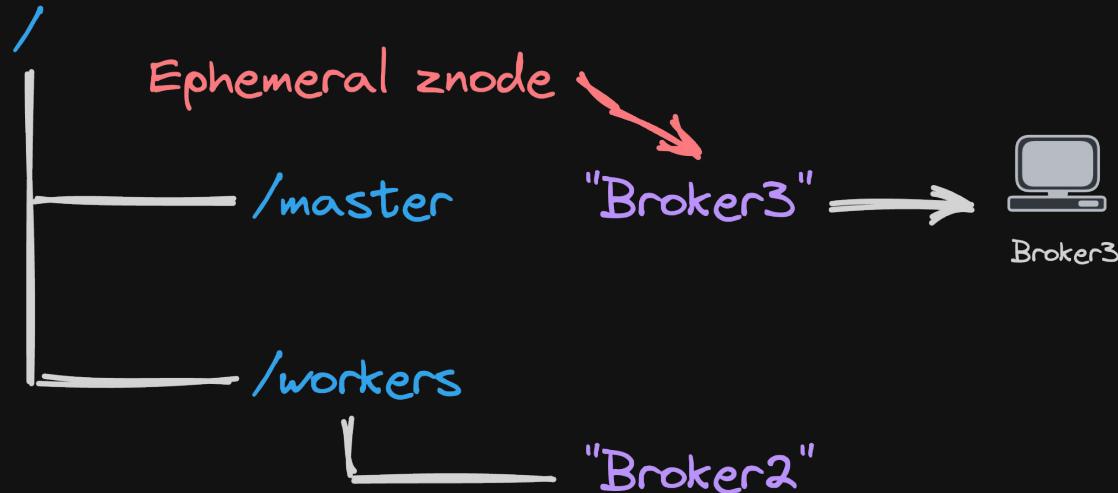
ZooKeeper의 Watch

- znode에 변화가 생기면 미리 Watch를 설정했던 Watcher에게 이벤트를 보내줍니다
 - NODE_CREATED: 노드가 생성 됨을 감지
 - NODE_DELETED: 노드가 삭제 됨을 감지
 - NODE_DATA_CHANGED: 노드의 데이터가 변경 됨을 감지
 - NODE_CHILDREN_CHANGED: 자식 노드가 변경 됨을 감지

ZooKeeper를 이용한 리더 선출



ZooKeeper를 이용한 리더 선출



Kafka의 리더 선출

```
● ● ●  
1 class ControllerChangeHandler(eventManager: ControllerEventManager) extends ZNodeChangeHandler {  
2   override val path: String = ControllerZNode.path  
3  
4   override def handleCreation(): Unit = eventManager.put(ControllerChange)  
5   override def handleDeletion(): Unit = eventManager.put(Reelect)  
6   override def handleDataChange(): Unit = eventManager.put(ControllerChange)  
7 }  
8
```

Kafka의 리더 선출

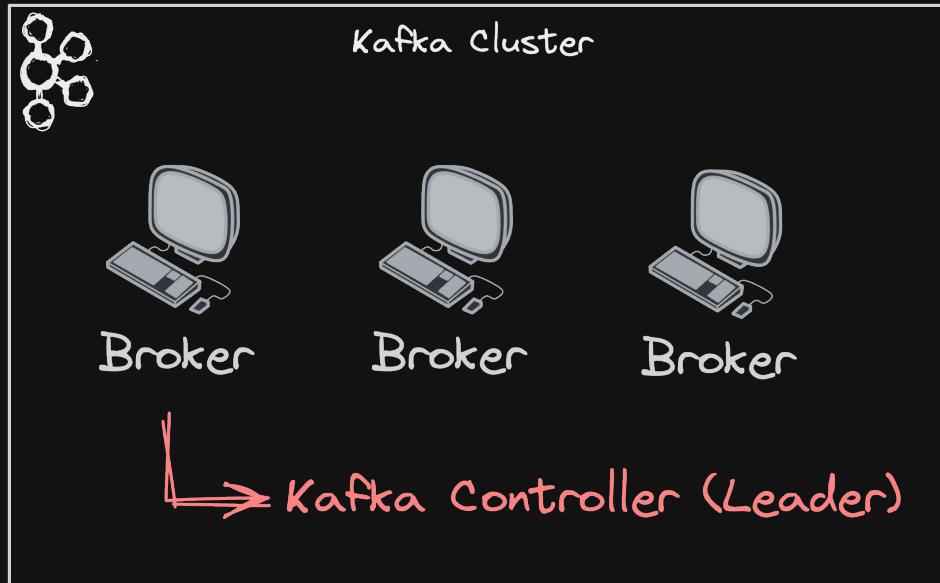
```
● ● ●  
1 private def elect(): Unit = {  
2     activeControllerId = zkClient.getControllerId.getOrElse(-1)  
3  
4     if (activeControllerId != -1) {  
5         return  
6     }  
7  
8     try {  
9         val (epoch, epochZkVersion) = zkClient.registerControllerAndIncrementControllerEpoch(config.brokerId)  
10        controllerContext.epoch = epoch  
11        controllerContext.epochZkVersion = epochZkVersion  
12        activeControllerId = config.brokerId  
13  
14        onControllerFailover()  
15    } catch {  
16        case e: ControllerMovedException =>  
17            maybeResign()  
18  
19        case t: Throwable =>  
20            triggerControllerMove()  
21    }  
22 }
```

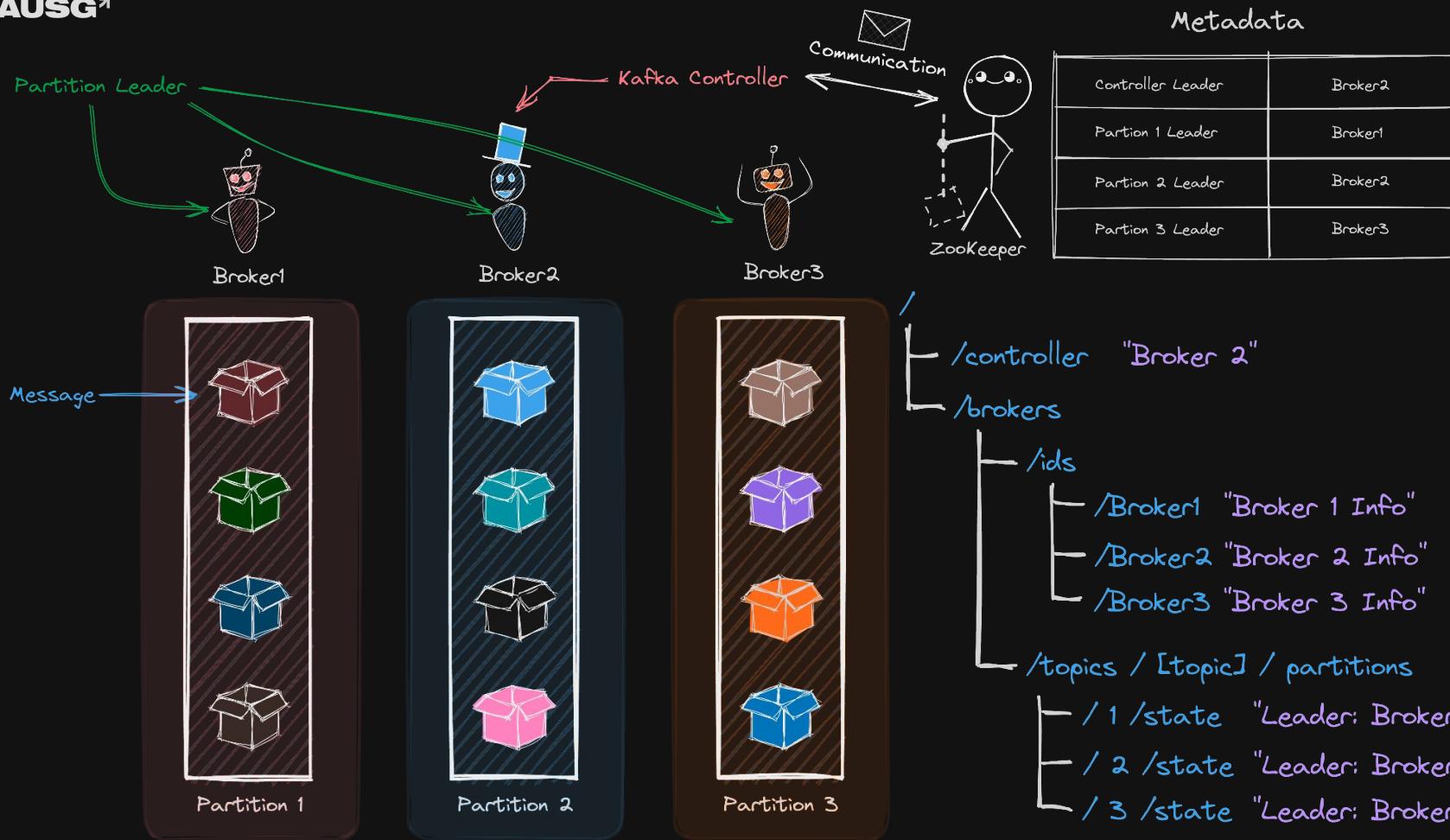
ZooKeeper

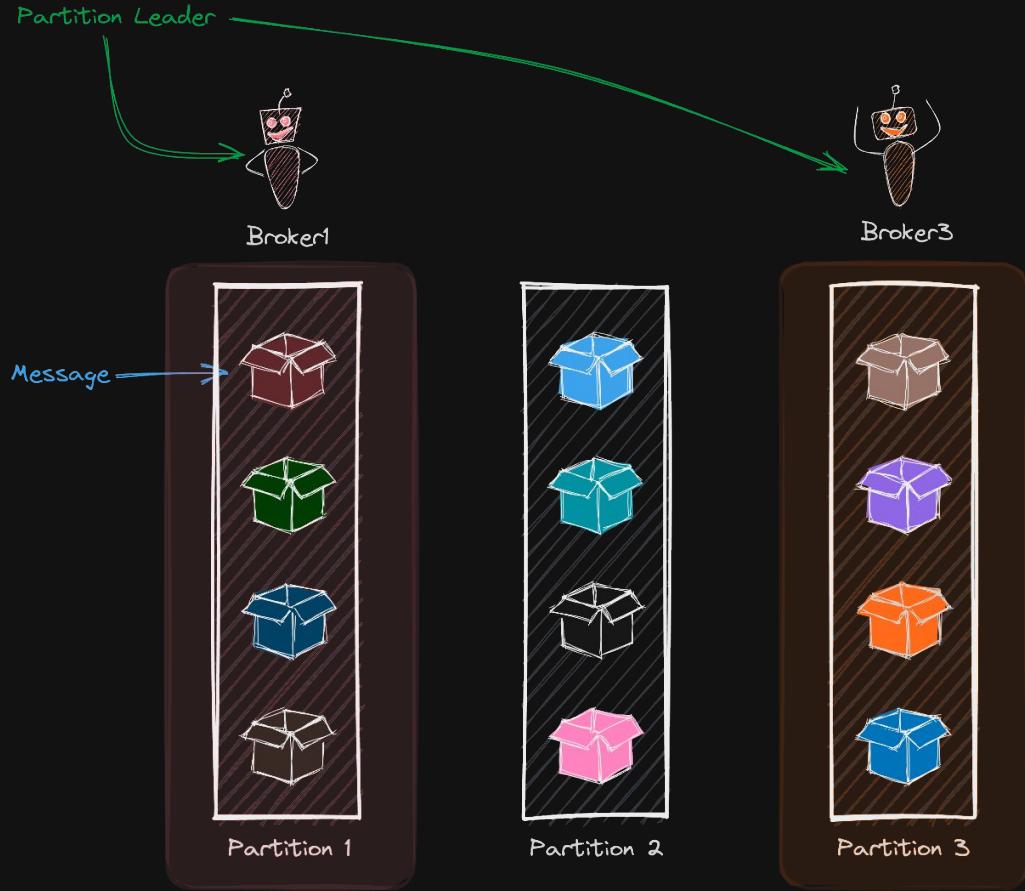
- ZAB Protocol(ZooKeeper Atomic Broadcast Protocol)
- 전체 순서 브로드캐스트(total order broadcast)
- CAP 이론
- 선형성
- 직렬성
- 순서화

Kafka와 ZooKeeper

Kafka와 ZooKeeper

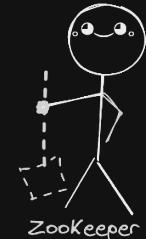






Metadata

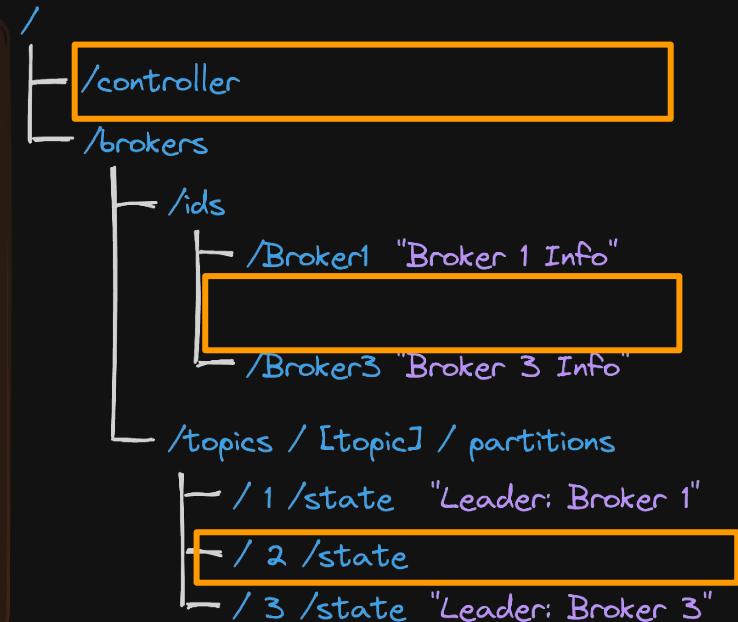
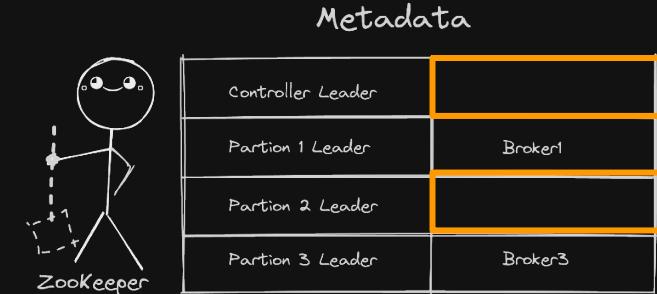
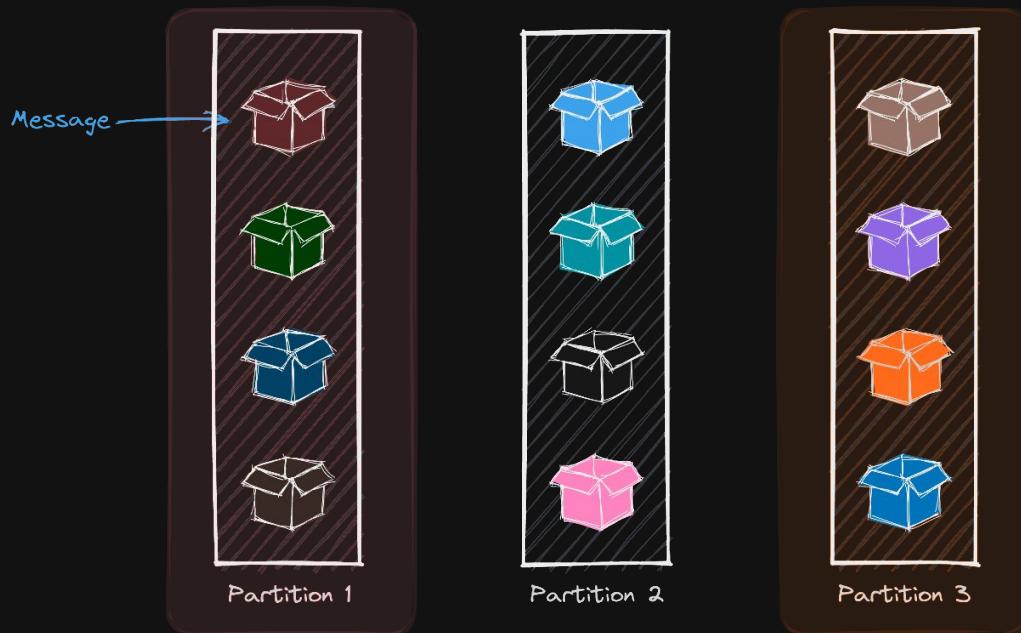
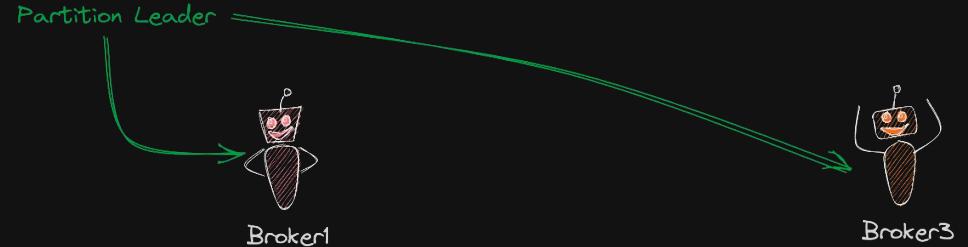
Controller Leader	Broker2
Partition 1 Leader	Broker1
Partition 2 Leader	Broker2
Partition 3 Leader	Broker3

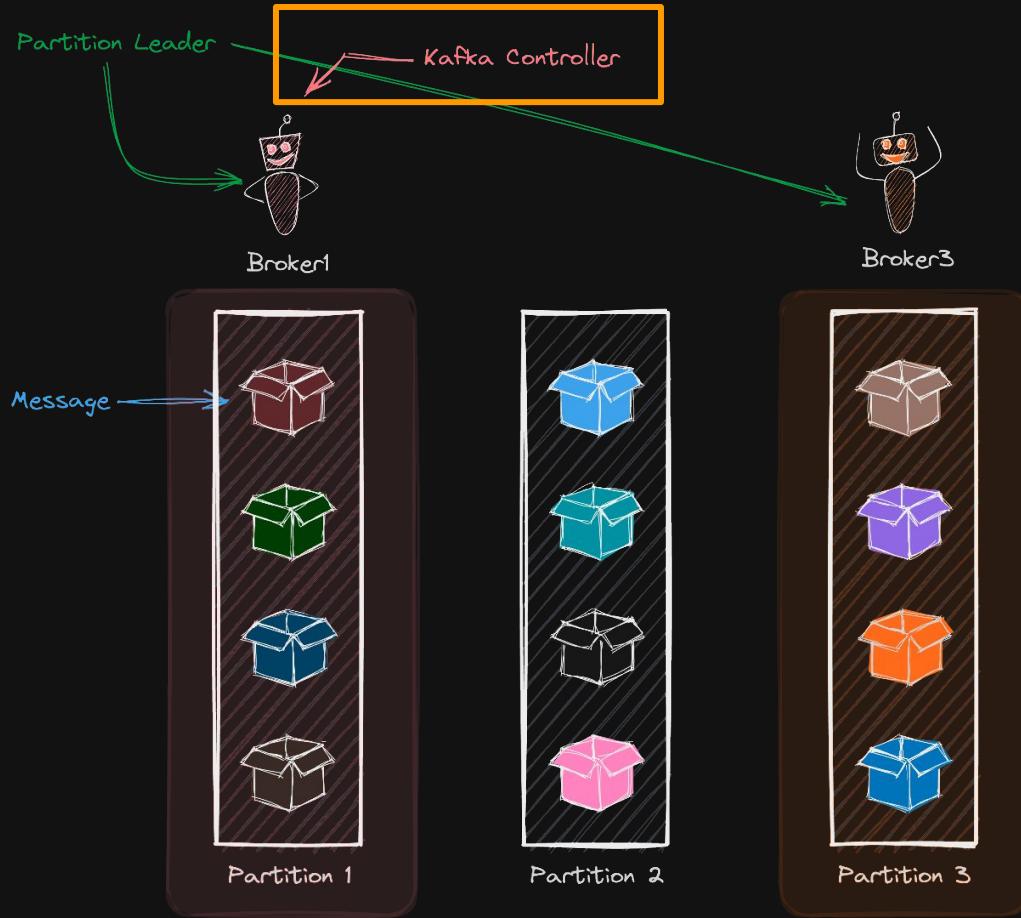


```

    /controller "Broker 2"
    /brokers
        /ids
            /Broker1 "Broker 1 Info"
            /Broker2 "Broker 2 Info"
            /Broker3 "Broker 3 Info"
        /topics / [Topic] / partitions
            / 1 /state "Leader: Broker 1"
            / 2 /state "Leader: Broker 2"
            / 3 /state "Leader: Broker 3"

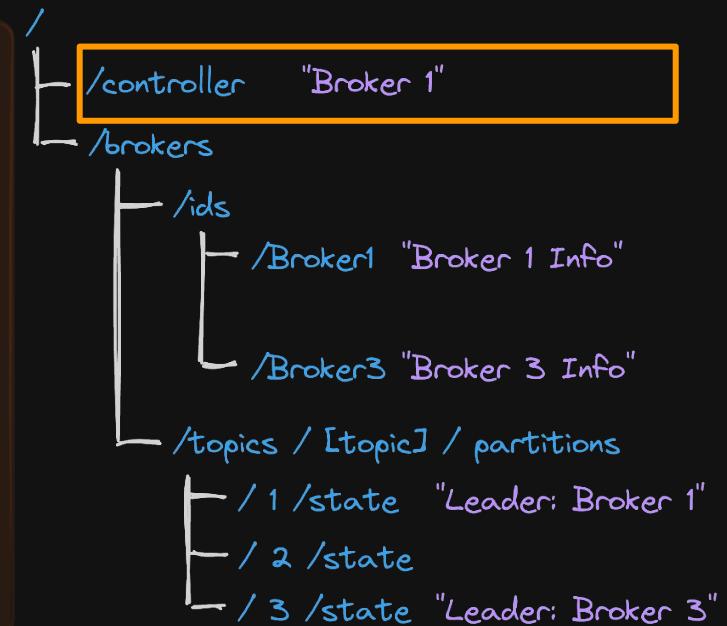
```

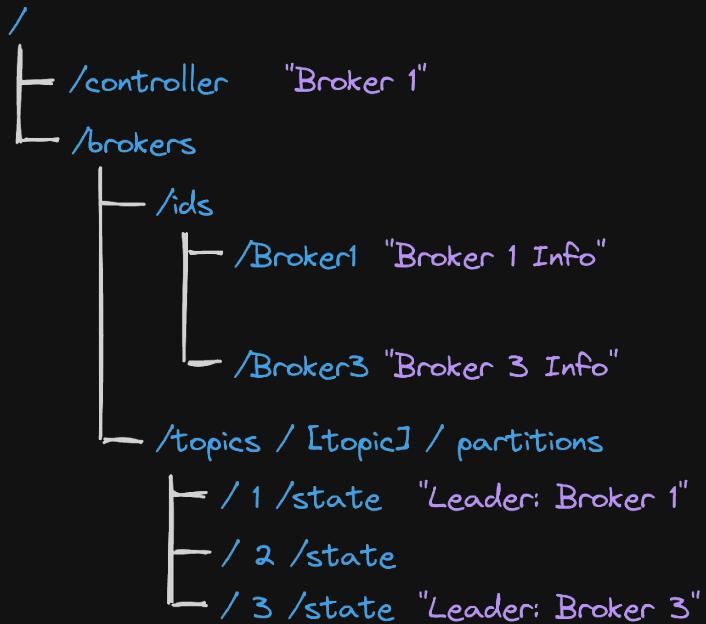
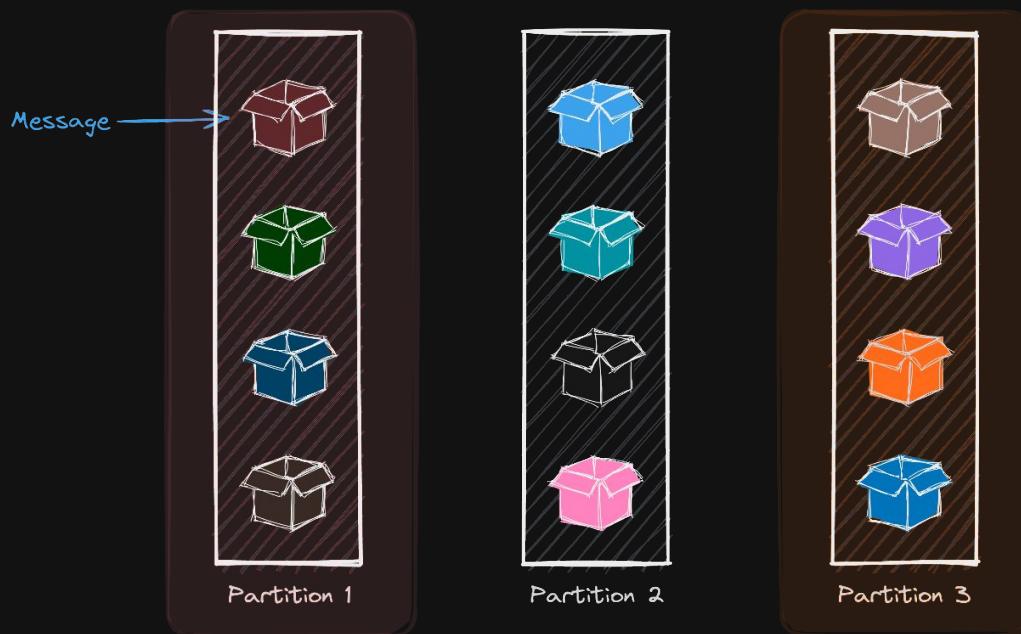
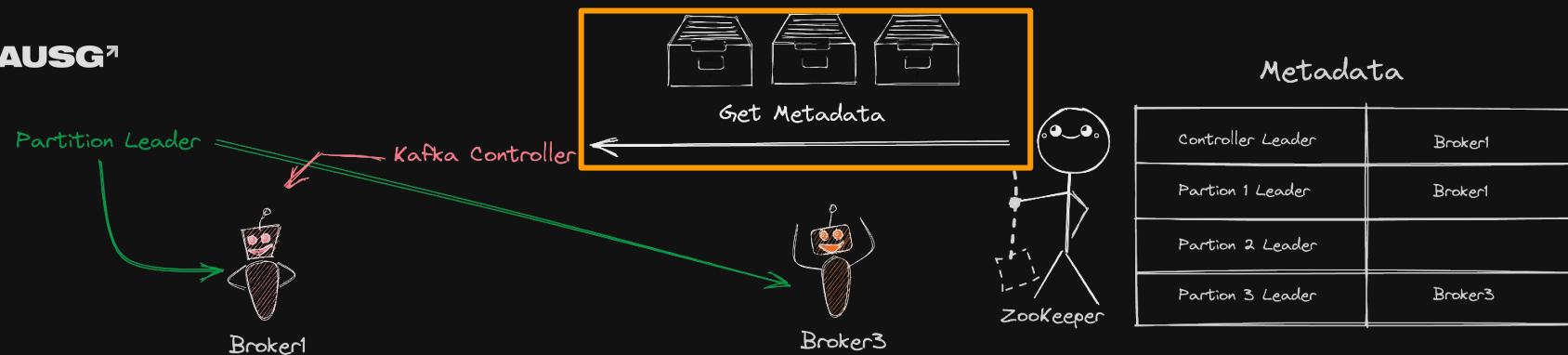


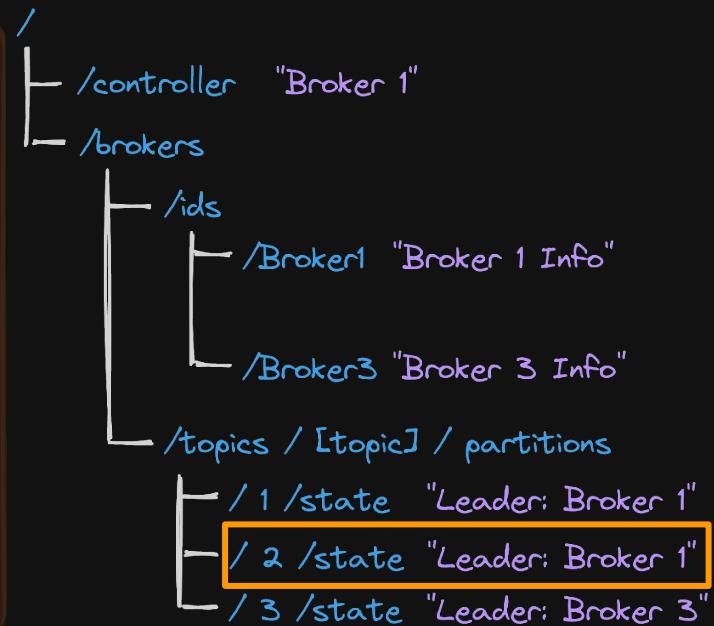
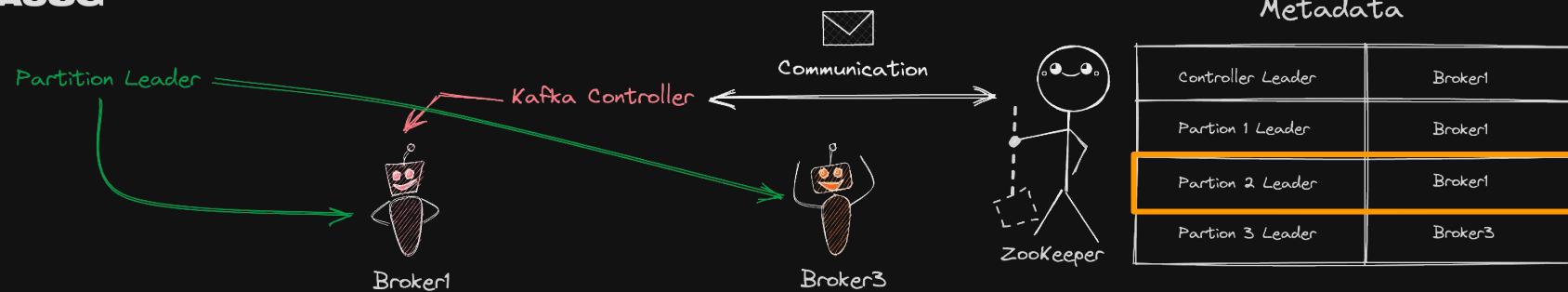


Metadata

Controller Leader	Broker1
Partition 1 Leader	Broker1
Partition 2 Leader	
Partition 3 Leader	Broker3







KIP-500

Replace ZooKeeper with a Metadata Quorum

KRaft Mode (Kafka Raft metadata mode)

ZooKeeper의 문제점

ZooKeeper의 문제점 : 추가적인 리소스 운영 및 관리



```
1 version: '3.8'
2 services:
3   zookeeper:
4     image: wurstmeister/zookeeper:latest
5     container_name: zookeeper
6     ports:
7       - "2181:2181"
8   kafka:
9     image: wurstmeister/kafka:latest
10    container_name: kafka
11    ports:
12      - "9092:9092"
13    environment:
14      KAFKA_ADVERTISED_HOST_NAME: 127.0.0.1
15      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
16    volumes:
17      - /var/run/docker.sock:/var/run/docker.sock
```

ZooKeeper의 문제점 : 외부 메타데이터 관리

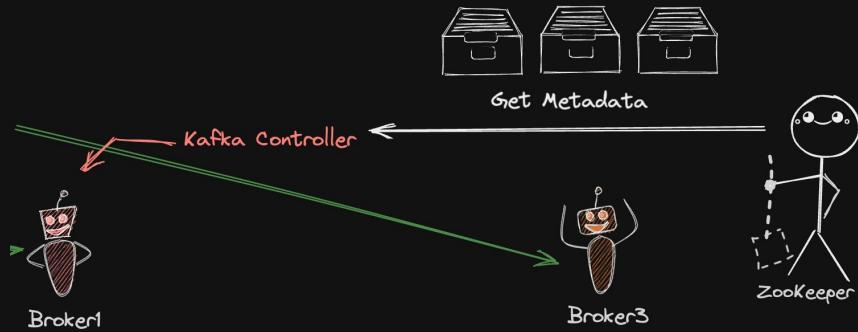
So what is the problem with ZooKeeper?

Actually, the problem is not with ZooKeeper itself
but with the concept of **external metadata management**

- [Confluent Blog](#)

ZooKeeper의 문제점 : 외부 메타데이터 관리

- 더 많은 리소스 자원 필요
- 이중 캐싱
- 메타데이터 로딩에 대한 부담
- Kafka의 확장성 저하
- Kafka Topic 생성 속도 저하
- 메타데이터 동기화 문제

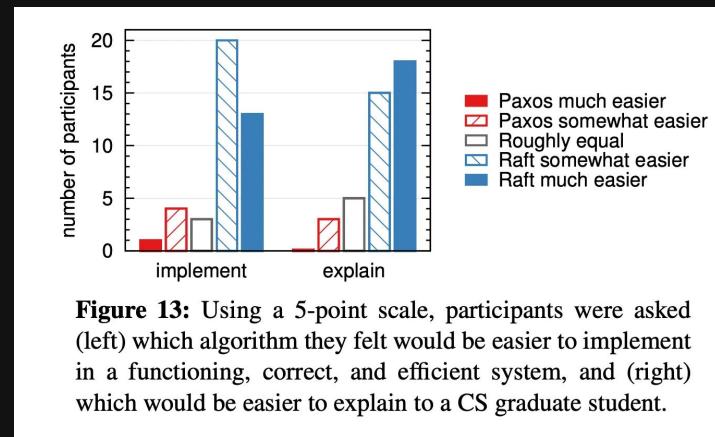


RAFT

RAFT



- 2014년 USENIX에서 발표된 “In Search of an Understandable **Consensus Algorithm**”라는 논문에서 소개
- 즉, 이해가능한 합의 알고리즘
 - **PAXOS**
- CockroachDB, MongoDB, Redis Cluster, Etcd, ClickHouse 등 다양한 곳에서 사용

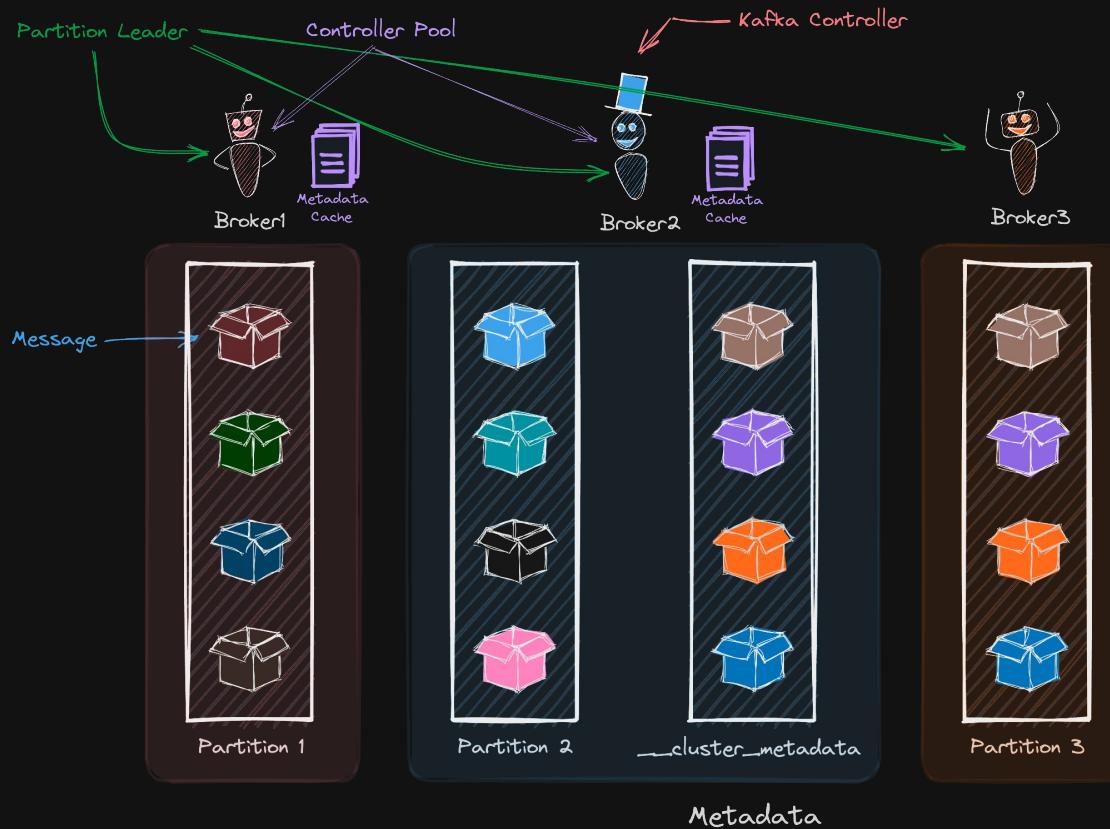


KRaft mode(Kafka Raft metadata mode)

ZooKeeper 대신 Raft 알고리즘을 이용해서 Kafka 만으로 리더 선출, 메타데이터 관리를 수행

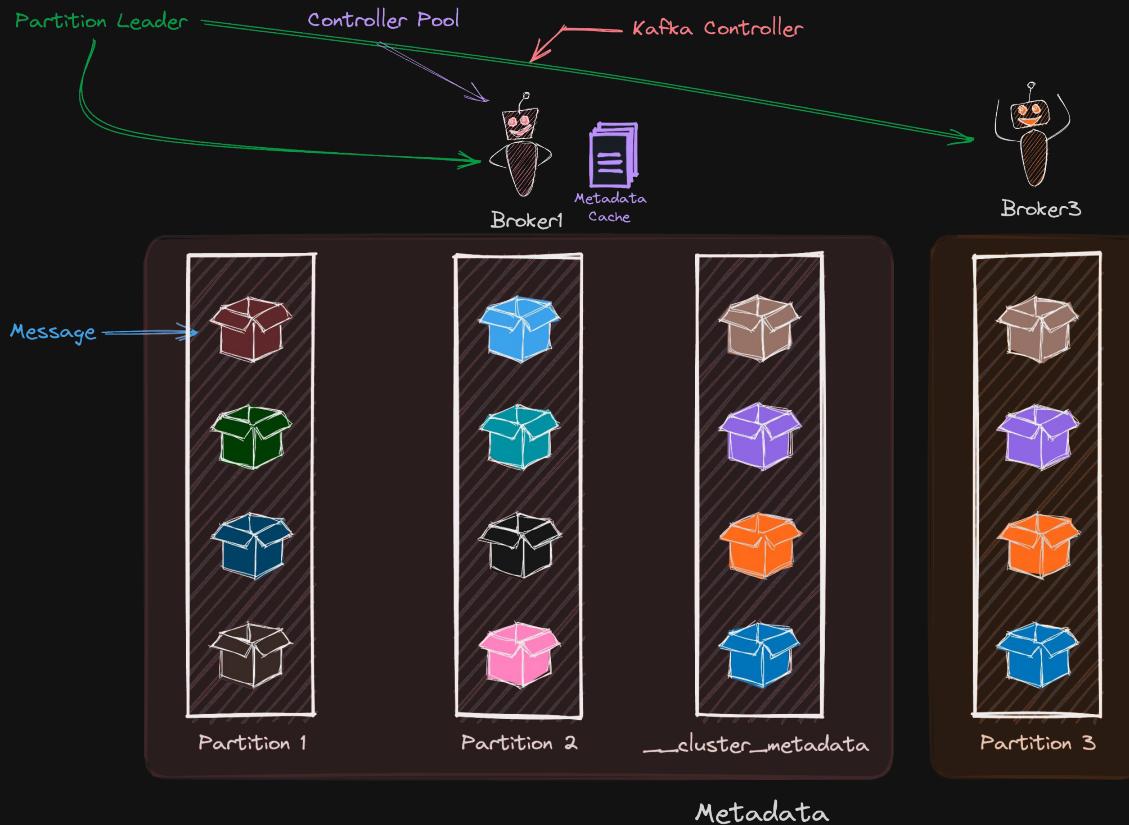
KRaft mode(Kafka Raft metadata mode)

- Controller Pool : 컨트롤러가 될 수 있는 후보 풀
- Metadata Cache : 메타데이터를 인-메모리에 캐시



Metadata

Controller Leader	Broker2
Partition 1 Leader	Broker1
Partition 2 Leader	Broker2
Partition 3 Leader	Broker3

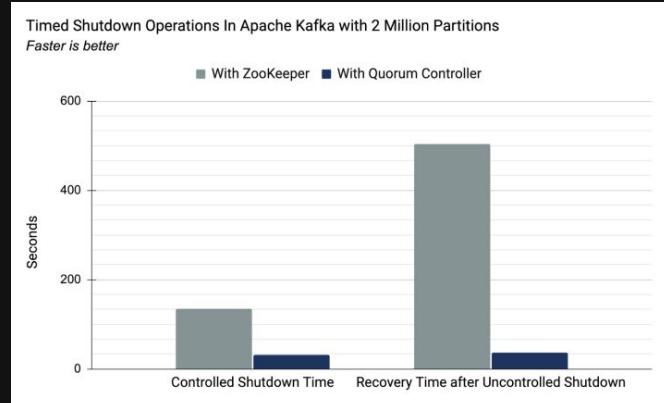
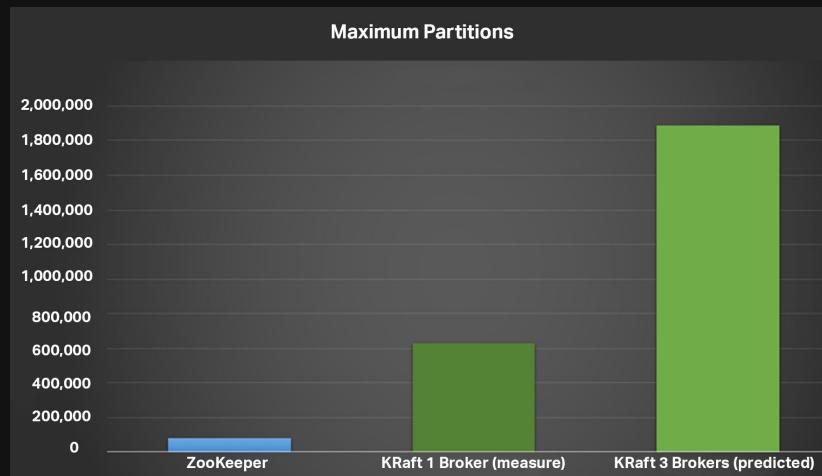


Metadata

Controller Leader	Broker1
Partition 1 Leader	Broker1
Partition 2 Leader	Broker1
Partition 3 Leader	Broker3

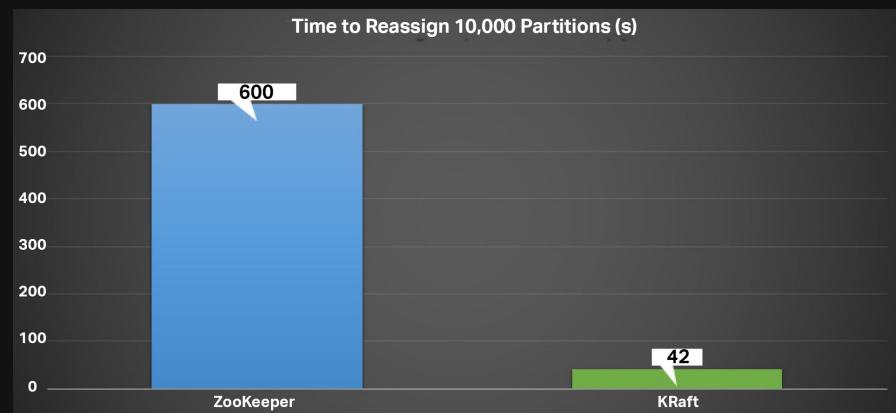
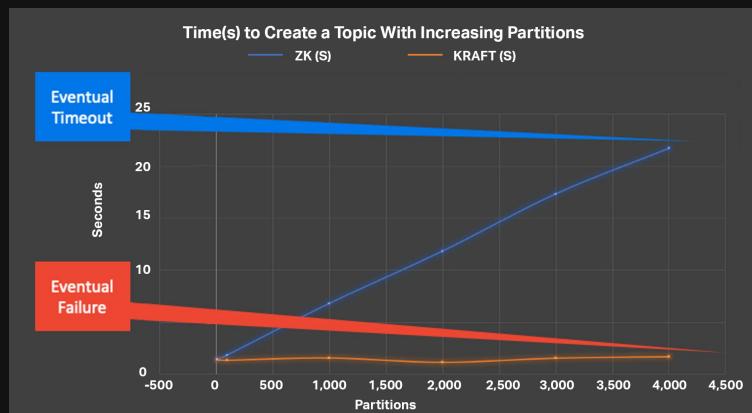
KRaft

ZooKeeper 대신 Raft 알고리즘을 이용해서 Kafka 만으로 리더 선출, 메타데이터 관리를 수행



KRaft

ZooKeeper 대신 Raft 알고리즘을 이용해서 Kafka 만으로 리더 선출, 메타데이터 관리를 수행



KRaft

- Kafka 2.8 버전부터 KRaft Mode 사용 가능 (2021년 4월)
- Kafka 3.3.1 버전에서 KRaft Mode가 Production Ready 되었음을 발표 (2022년 10월)
- Kafka 3.5 버전부터 ZooKeeper Mode Deprecated (2023년 6월)
- Kafka 3.7 버전이 ZooKeeper를 지원하는 마지막 버전 (미정)
- Kafka 4.0 버전부터 KRaft 모드만 지원
- Confluent Platform도 7.5 버전부터 ZooKeeper 모드를 Deprecated 시켜버림
 - 새 배포에는 KRaft 모드를 적용하는 것을 권장
- 다만 AWS MSK는 Kafka 3.5.1 버전에 대해서 여전히 ZooKeeper 모드만 지원

긴 발표를 들어주셔서 감사합니다