

# 하이퍼레저 패브릭 기반의 프라이빗 블록체인 개발 과정

2020-10-20

빅픽처랩(주)

안 휘

# 전체 목차

- **하이퍼레저 패브릭 개요 (1 day)**
  - 블록체인의 본질
  - 프라이빗 블록체인의 현재
  - 하이퍼레저 패브릭을 구성하는 기술들
  - 하이퍼레저 패브릭 구동 실습
- **하이퍼레저 패브릭 개발 (2 day)**
  - 스마트컨트랙트 개요
  - 하이퍼레저 패브릭 아키텍처
  - 하이퍼레저 패브릭 스마트컨트랙트 개발 실습

# 스마트 컨트랙트 개요

2020-10-20

빅픽처랩(주)

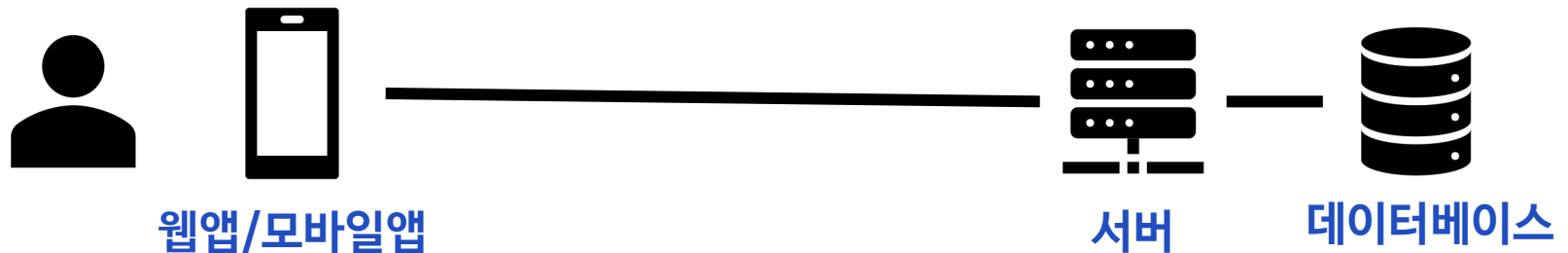
안휘

# 목차

1. 블록체인 기반 소프트웨어 시스템
2. Client-Server-Blockchain 시스템 개발
3. 블록체인과 일반 데이터베이스 개발의 차이
4. 스마트컨트랙트 설계 원칙

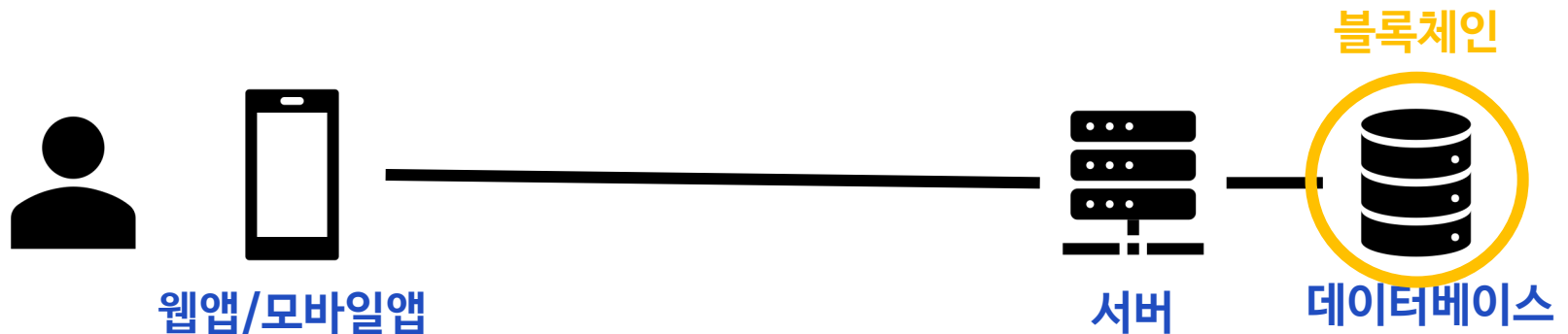
# 1. 블록체인 기반 소프트웨어 시스템

- 소프트웨어 시스템: 소프트웨어 서비스를 가능케 하는 체계



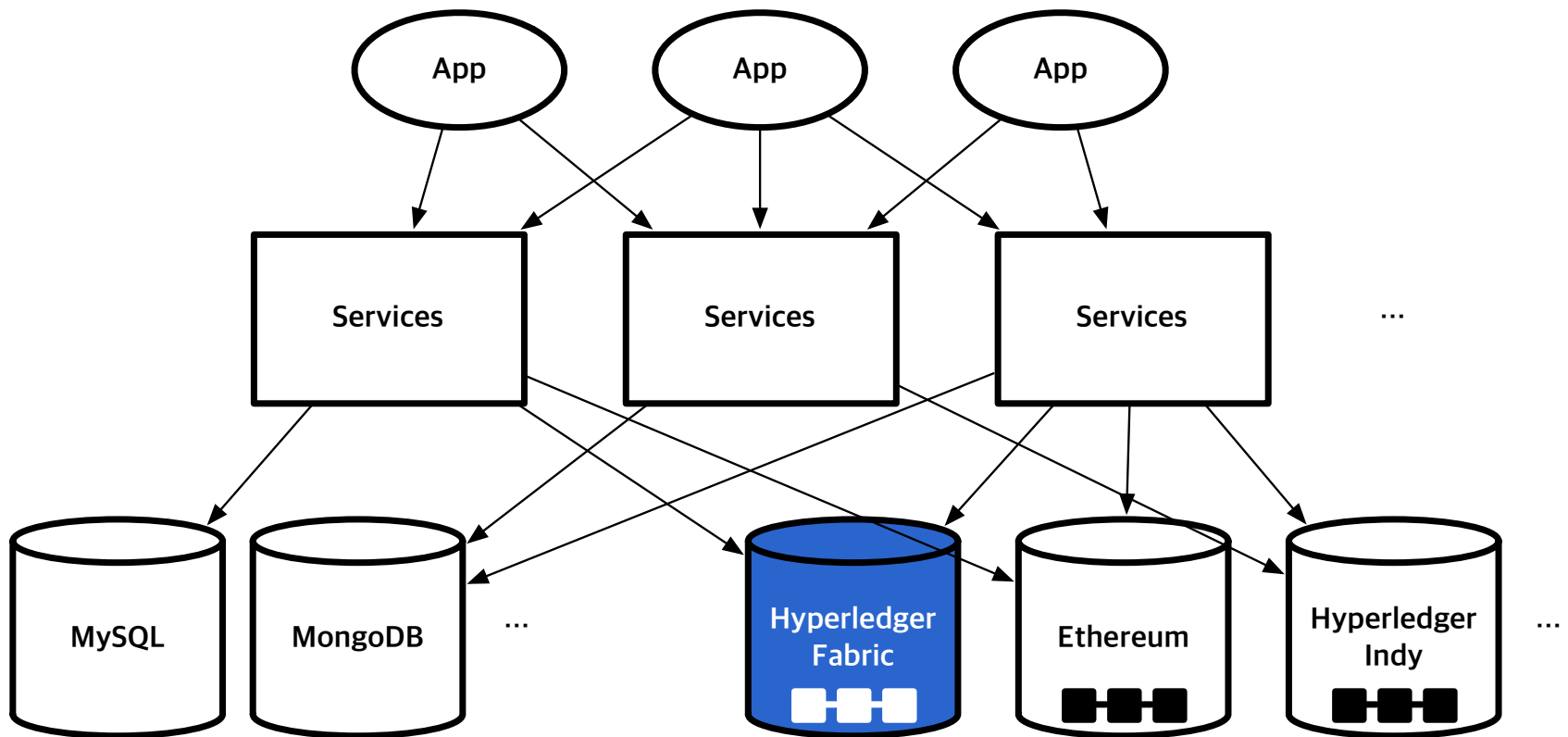
# 1. 블록체인 기반 소프트웨어 시스템

- 소프트웨어 시스템: 소프트웨어 서비스를 가능케 하는 체계



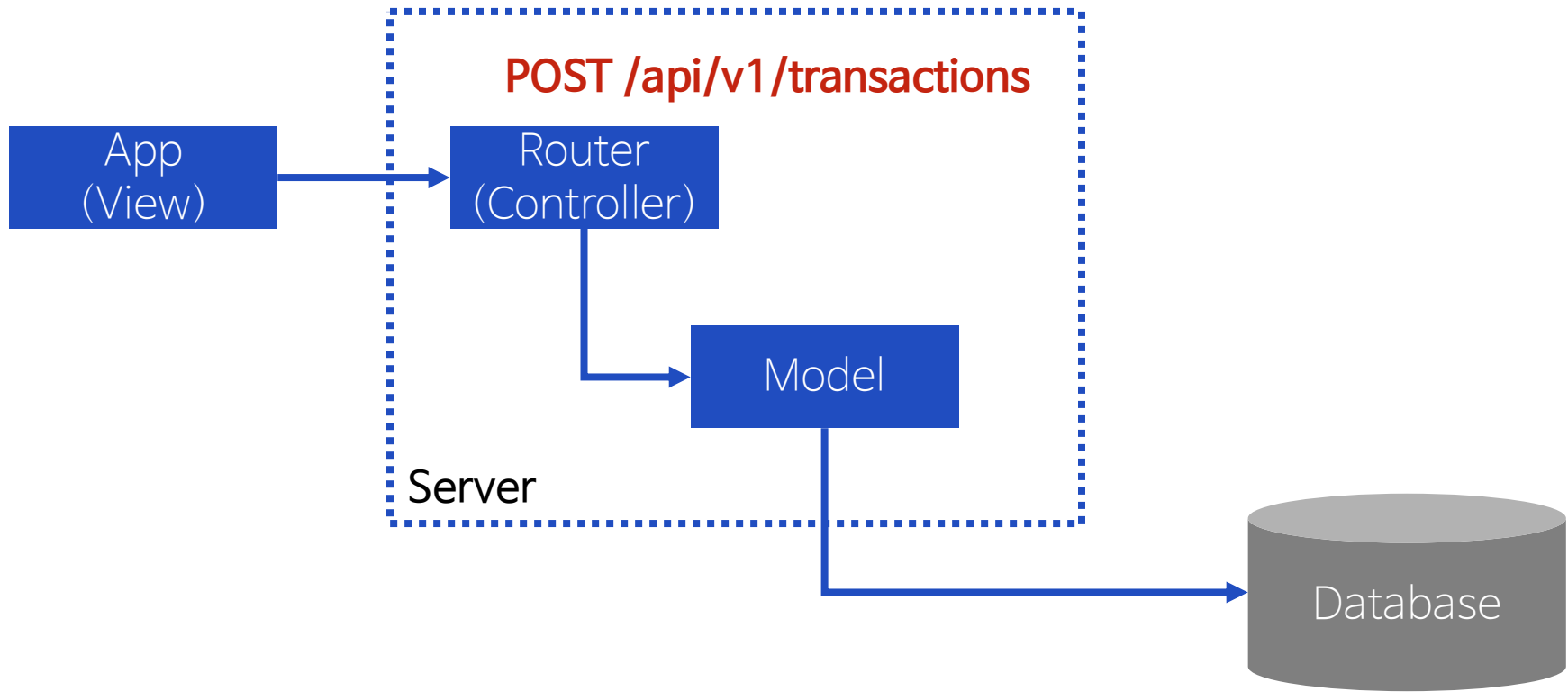
우리가 개발하는 전체 시스템에서  
데이터를 저장하는 부분 중 하나가  
블록체인인 시스템

# 1. 블록체인 기반 소프트웨어 시스템



## 2. Client-Server-Blockchain 시스템 개발

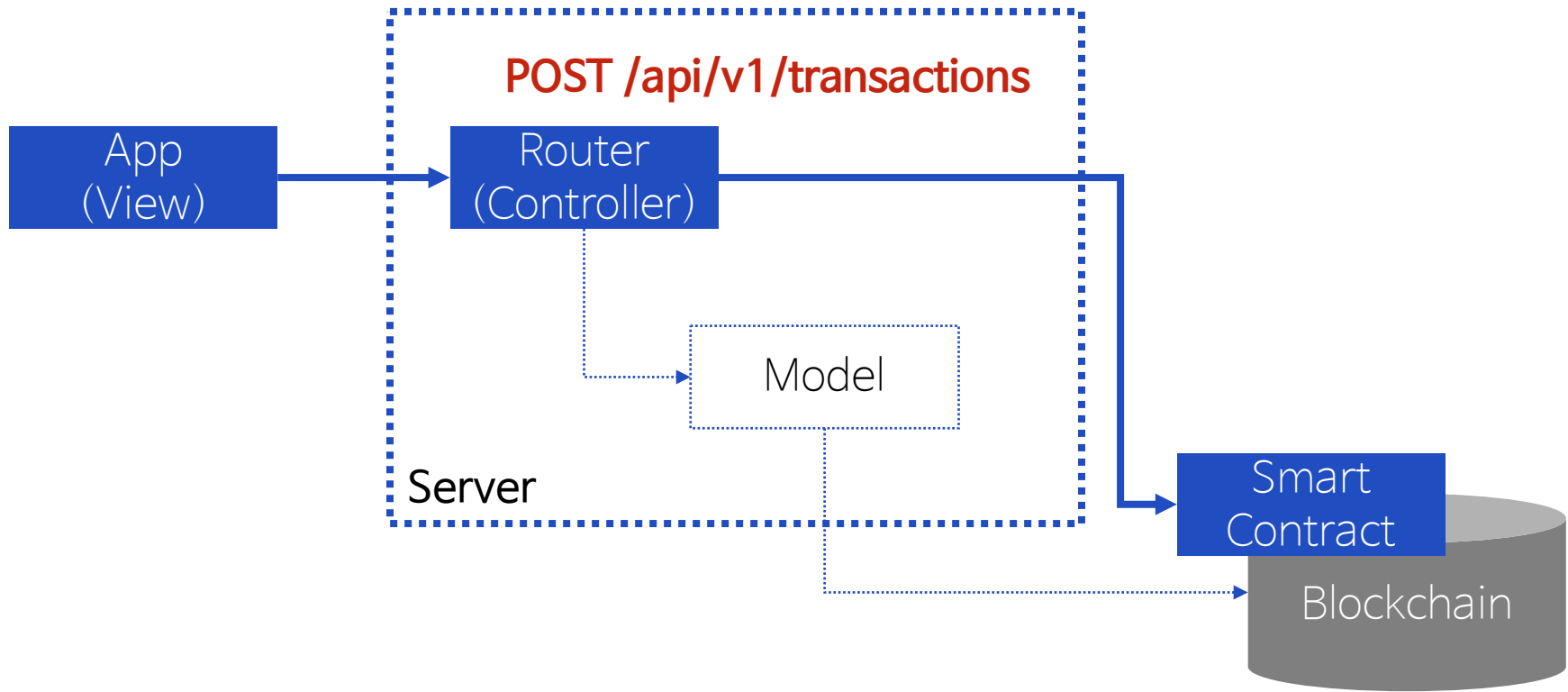
- 일반적인 Client-Server-Database 시스템의 서버 개발





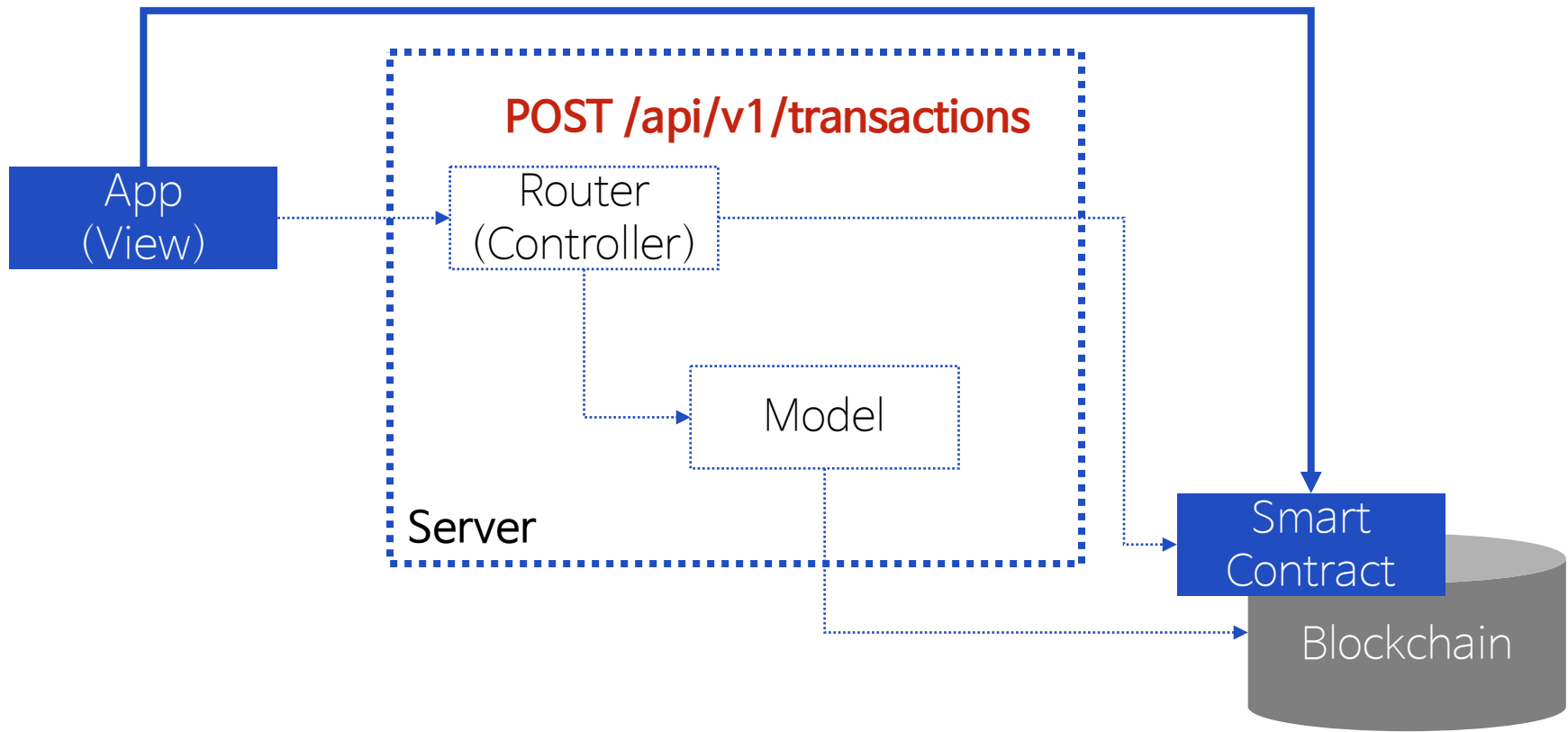
## 2. Client-Server-Blockchain 시스템 개발

- Client-Server-Blockchain 시스템의 서버 개발 (Hyperledger Fabric)



## 2. Client-Server-Blockchain 시스템 개발

- Client-**Server**-Blockchain 시스템의 서버 개발 (Ethereum)



## 2. Client-Server-Blockchain 시스템 개발

- Client-Server-Database 시스템의 서버 개발
  - REST API를 위한 Router 정의
  - DB 연결 코드 작성
    - DB 전용 SDK 사용
    - `db.connect(...)` 이런식으로 호출
    - 보통 서버가 실행될 때 1번 호출되어, DB와 연결을 수립
  - Model 코드 작성
    - DB에 저장되는 객체의 모습 정의 (Schema)
    - DB에 저장된 객체를 불러오는 함수들 작성 (a.k.a CRUD 함수)
    - “서버의 로직은 Model에 담긴다”

## 2. Client-Server-Blockchain 시스템 개발

- Client-Server-Blockchain 시스템의 서버 개발

- REST API를 위한 Router 정의
- DB 연결 코드 작성
  - DB 전용 SDK 사용 → fabric-sdk, web3.js
  - ~~• db.connect(...) 이런식으로 호출~~
  - ~~• 보통 서버가 실행될 때 1번 호출되어, DB와 연결을 수립~~
  - **연결, 트랜잭션 관리 등을 모두 직접 해주어야 할 가능성이 높음**
- ~~• Model 코드 작성 → 스마트컨트랙트 코드 작성~~
  - DB(블록체인)에 저장되는 객체의 모습 정의 (Schema)
  - DB(블록체인)에 저장된 객체를 불러오는 함수들 작성 (a.k.a CRUD 함수)
  - “서버의 로직은 ~~Model~~ **스마트컨트랙트**에 담긴다”

## 2. Client-Server-Blockchain 시스템 개발

- Client-Server-Blockchain 시스템의 서버 개발

- REST API를 위한 Router 정의
- DB 연결 코드 작성 → Client-side(e.g. Web App, Mobile App)로 이동
  - DB 전용 SDK 사용 → web3.js
  - ~~db.connect(...) 이런식으로 호출~~
  - ~~보통 서버가 실행될 때 1번 호출되어, DB와 연결을 수립~~
  - **연결, 트랜잭션 관리 등을 모두 직접 해주어야 할 가능성이 높음**
- ~~Model 코드 작성~~ → **스마트컨트랙트** 코드 작성
  - DB(블록체인)에 저장되는 객체의 모습 정의 (Schema)
  - DB(블록체인)에 저장된 객체를 불러오는 함수들 작성 (a.k.a CRUD 함수)
  - “서버의 로직은 ~~Model~~ **스마트컨트랙트**에 담긴다”

# 3. 블록체인과 일반 데이터베이스 개발의 차이

- **고수준의 DBMS 부재**

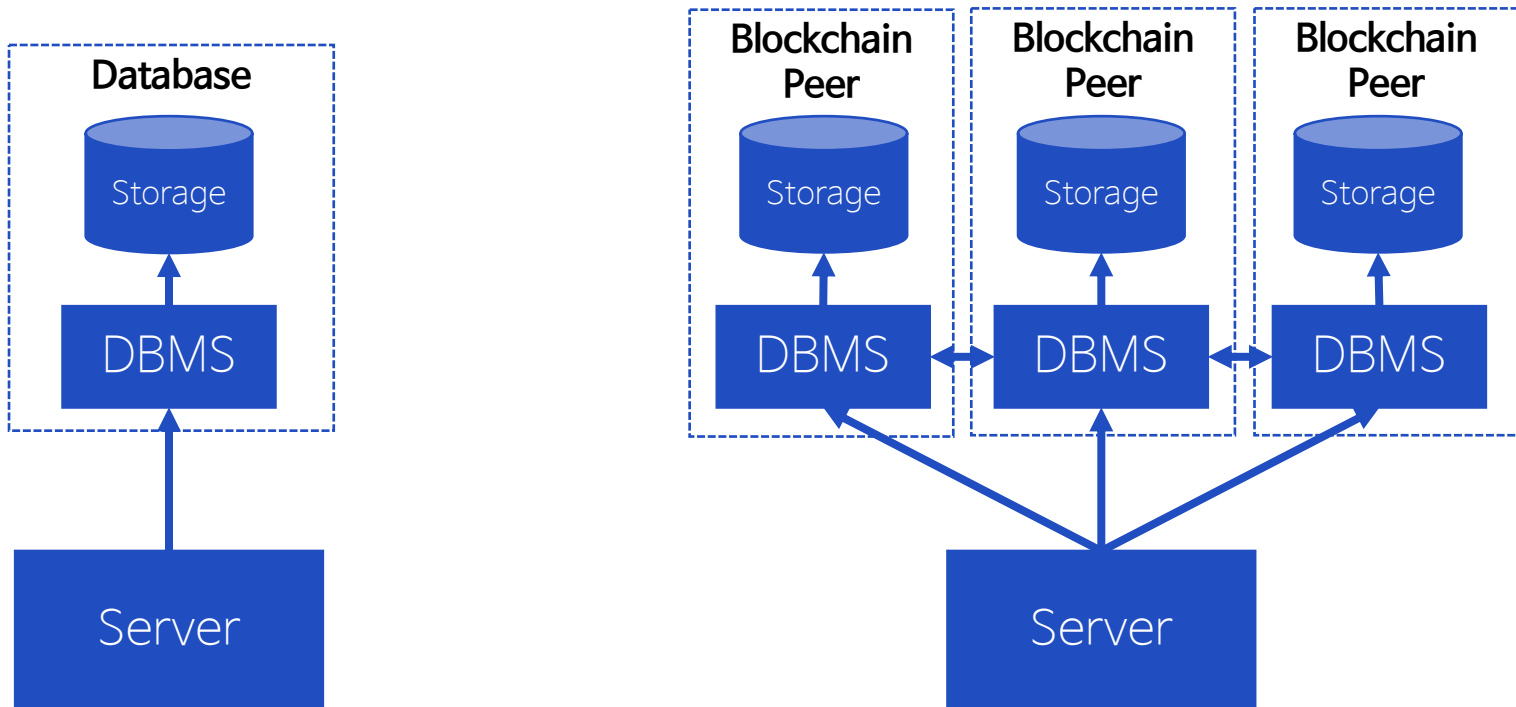
- DBMS: DataBase Management System
- DB 자체도 서버: 다중 접속 제어
- DB 내 데이터에 대한 무결성(데이터 무결성)을 보장함
  - 저장된 데이터들의 일관성을 보장
  - 쏟아지는 트랜잭션들을 잘 정렬&조절하여 데이터 변경의 일관성을 보장
  - 트랜잭션 실패 시 데이터의 원상복구를 보장
  - a.k.a **트랜잭션 관리**
- db.put(...) 과 같은 high-level 함수 속에 저런 기능들이 모두 보장되고 있음

**데이터베이스로 통하는 모든  
트랜잭션이 일단 통과해야 하는  
중앙화 된 서버 시스템**

# 3. 블록체인과 일반 데이터베이스 개발의 차이

- 고수준의 DBMS 부재

- 블록체인에는 **중앙화 된 DBMS가 없음**
- 트랜잭션 처리 시 **동시성 문제**가 발생할 수 있음
  - 읽는 도중 다른 곳에서는 데이터를 써서 같은 키에 대해 서로 다른 값을 본다던가...



### 3. 블록체인과 일반 데이터베이스 개발의 차이

- 트랜잭션을 그냥 보내기만 하면 DBMS 가 알아서 데이터 저장을 잘 해줄 것이라는 믿음을 버려야 함
- MVCC Read Conflict
  - 먼저 보낸 트랜잭션이 미처 반영되기 전에 같은 키에 대해 또다른 트랜잭션을 보냄
- Phantom Read Conflict
  - DB에서 Range 검색 시, 처리 도중 다른 트랜잭션의 삽입으로 처음 읽을 때 없던 값들이 추가되거나 있던 값들이 삭제됨





# 4. 스마트컨트랙트 설계 원칙

- 일반적인 “Client-Server-Database” 시스템의 DB모델 설계 원칙과 같음
  - ER 다이어그램으로 DB 구조 설계 잘하고,
  - 서버 보안 잘 챙기고,
  - 앱은 UI/UX 잘 만들고, ... 등등

그러나...

# 4. 스마트컨트랙트 설계 원칙

- 블록체인은 다른 데이터베이스에 비해 ...

- 1) 매우 **느리고**,
- 2) **용량을 많이** 차지하고,
- 3) **고수준의 DBMS가 없습니다.**

=> 그럼에도 여기에  
**저장해야하는 것은 무엇인가**

# 4. 스마트컨트랙트 설계 원칙

- 무엇을 저장할지 결정

- 저장할 만한 데이터 (AND 조건)

- 위변조를 반드시 막아야 하는 데이터
    - 참여자들이 모두 함께 공유해야 하는 데이터
    - 예: 투표 결과, 돈 거래 내역, 물건의 운송 이력, 이해관계자들 사이의 중요 문서(계약서 등)

- 저장하면 안되는 데이터 (OR 조건)

- 실시간성을 요구하는 데이터는 안됨
    - 복잡한 Query가 중요한 데이터 (End-User의 앱 서비스에 직접 연관되는 데이터)
      - JOIN 등을 요구하는 복잡한 구조의 데이터
    - 공유되어서는 안되는 개인 데이터
    - 예: 주민등록번호 같은 개인정보, 로그

# 4. 스마트컨트랙트 설계 원칙

- **NoSQL의 DB 설계 원칙과 동일**

- 하이퍼레저 패브릭과 이더리움 모두 기본적으로 NoSQL 데이터베이스
  - 둘다 LevelDB를 저장소로 사용함
    - 구글에서 만든 빠르고 가볍지만 불편한 DB
- Key-Value 데이터베이스인데, Key 구조를 잘 설계하는 것이 중요
  - Query 에 맞춰 Key를 설계
  - LevelDB는 FindByAttributes 이런 거 안됨
  - Index 도 없음 → 대신 Key 를 알파벳 순서로 정렬은 해줌

# 4. 스마트컨트랙트 설계 원칙

- 블록체인은 언제나 서브 DB

- 블록체인을 서비스를 위한 메인 DB로 쓰지 마라

- 사용자 앱의 첫 페이지를 띄우기 위한 데이터를 fetching 해오는 DB로 쓰면...



- 블록체인은 올라운드 플레이어가 아니라, 특정 영역에서 아무도 못하는 일을 해주는 **스페셜 플레이어**임
  - 사용자가 블록체인이 뒤에서 동작하는 것을 모르는 것이 최고의 블록체인 기반 소프트웨어 서비스

# 하이퍼레저 패브릭 아키텍처

2020-10-20

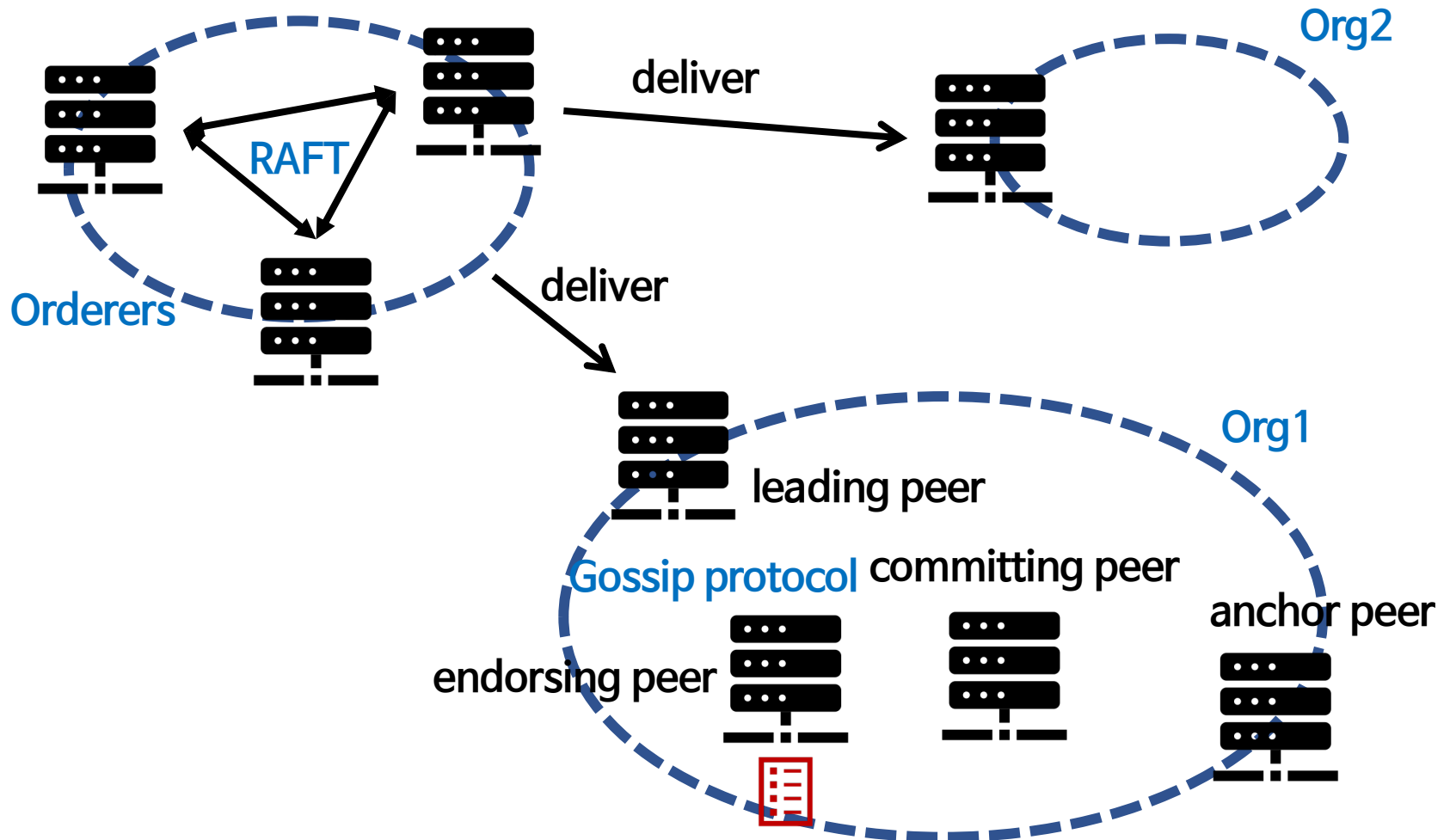
빅픽처랩(주)

안휘

# 목차

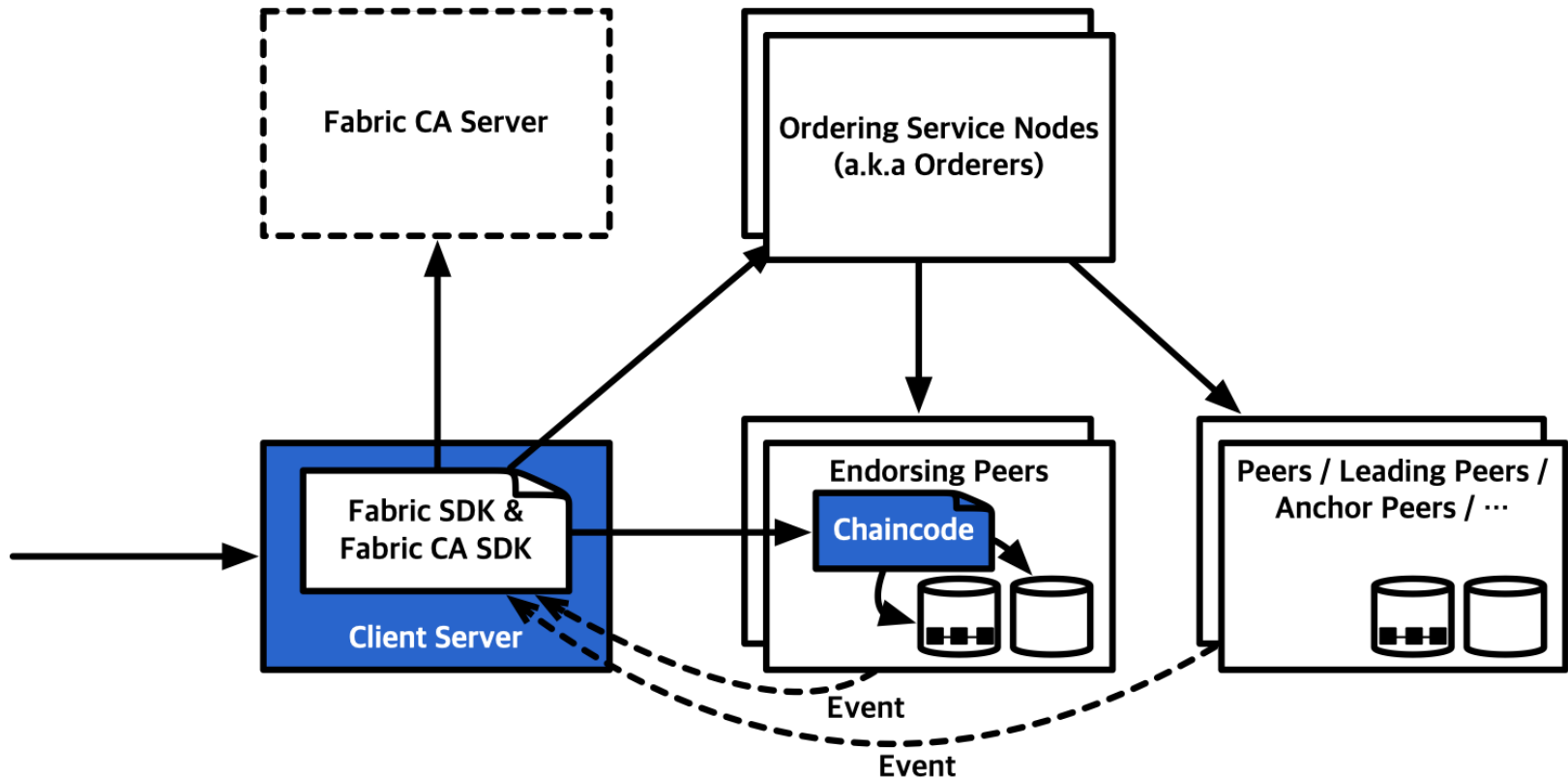
1. 하이퍼레저 패브릭 구조
2. 하이퍼레저 패브릭 네트워크 구동 시나리오
3. 트랜잭션 전송 시나리오
4. 체인코드 구조
5. 체인코드 개발 시 주의사항

# 1. 하이퍼레저 패브릭 구조





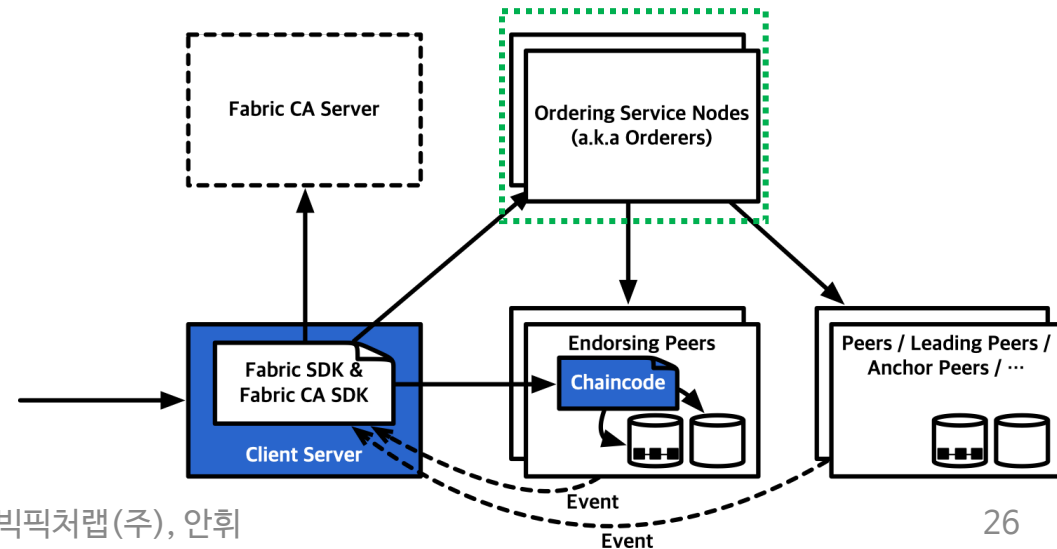
# 1. 하이퍼레저 패브릭 구조



# 1. 하이퍼레저 패브릭 구조

## • Orderers

- 트랜잭션을 순서대로 나열함
  - Apache Kafka
- 블록을 생성함
- 검증은 하지 않음 → Committing Peer가 담당
- RAFT: Orderer들 사이의 Fault Tolerance를 보장, Byzantine Fault Tolerance는 보장하지 못함



# 1. 하이퍼레저 패브릭 구조

- **Committing Peer**

- Orderer로부터 블록을 받아 저장하는 Peer (즉, 채널에 가입된 모든 Peer)

- **Endorsing Peer**

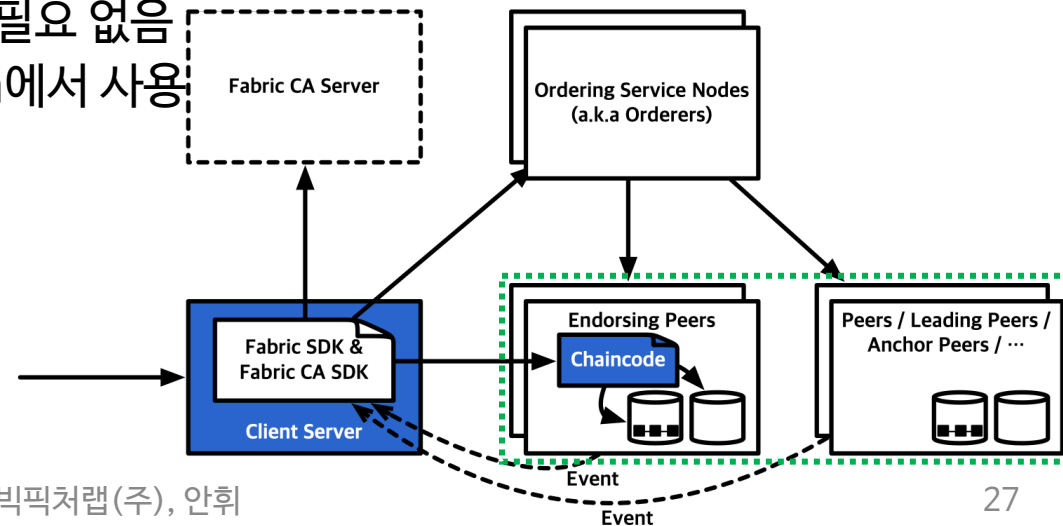
- Chaincode를 실행하는 Peer
- 트랜잭션을 미리 실행해보고, 에러가 있는지 없는지 확인

- **Anchor Peer**

- 조직 외부에서 조직 내 Peer들을 찾을 때 사용
- 외부에서 모든 Peer를 알고 있으면 필요 없음
- Service Discovery, Private Data에서 사용

- **Leading Peer**

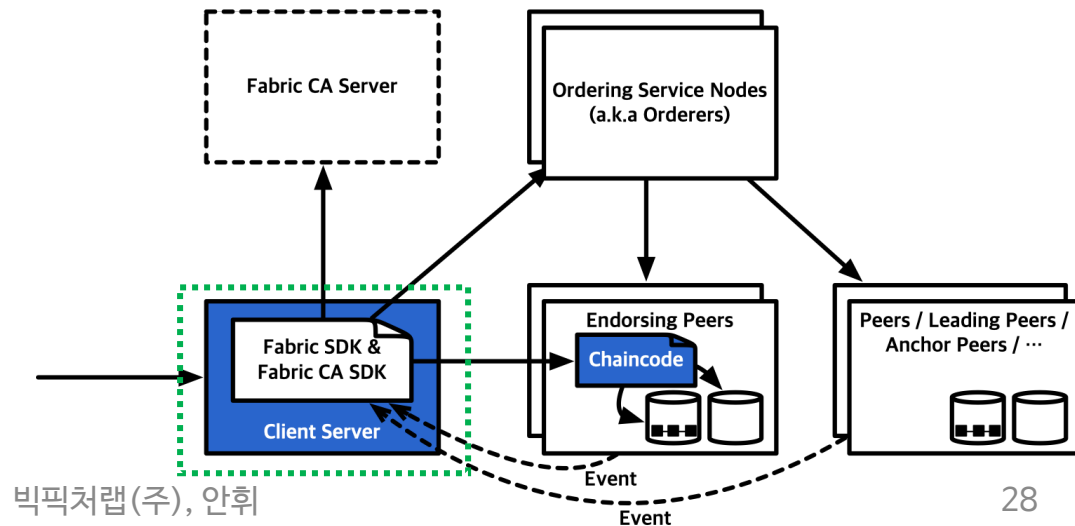
- 조직을 대표하여, Orderer로부터 블록을 deliver 받음
- 자동 선출



# 1. 하이퍼레저 패브릭 구조

## • Client Server

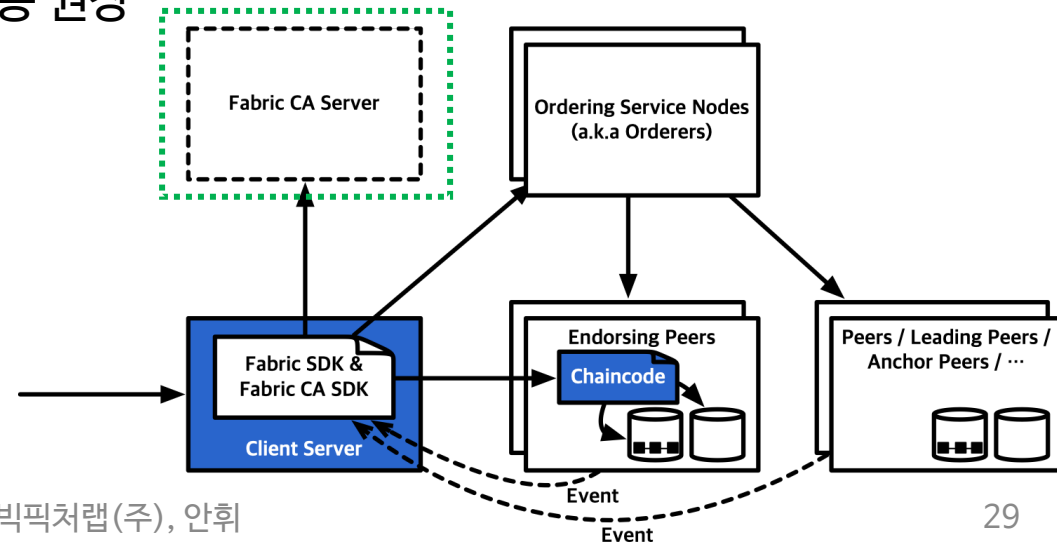
- 트랜잭션 생성
- 모든 네트워크 엔터티들과 접속 필요
  - Fabric CA: End user 키 Register 및 Enroll
  - Endorsing Peer 또는 Anchor Peer: Endorsement 생성
  - Orderer: 트랜잭션을 전송  
(Client Server가 수집한 Endorsement와 함께 직접 보내야 함)



# 1. 하이퍼레저 패브릭 구조

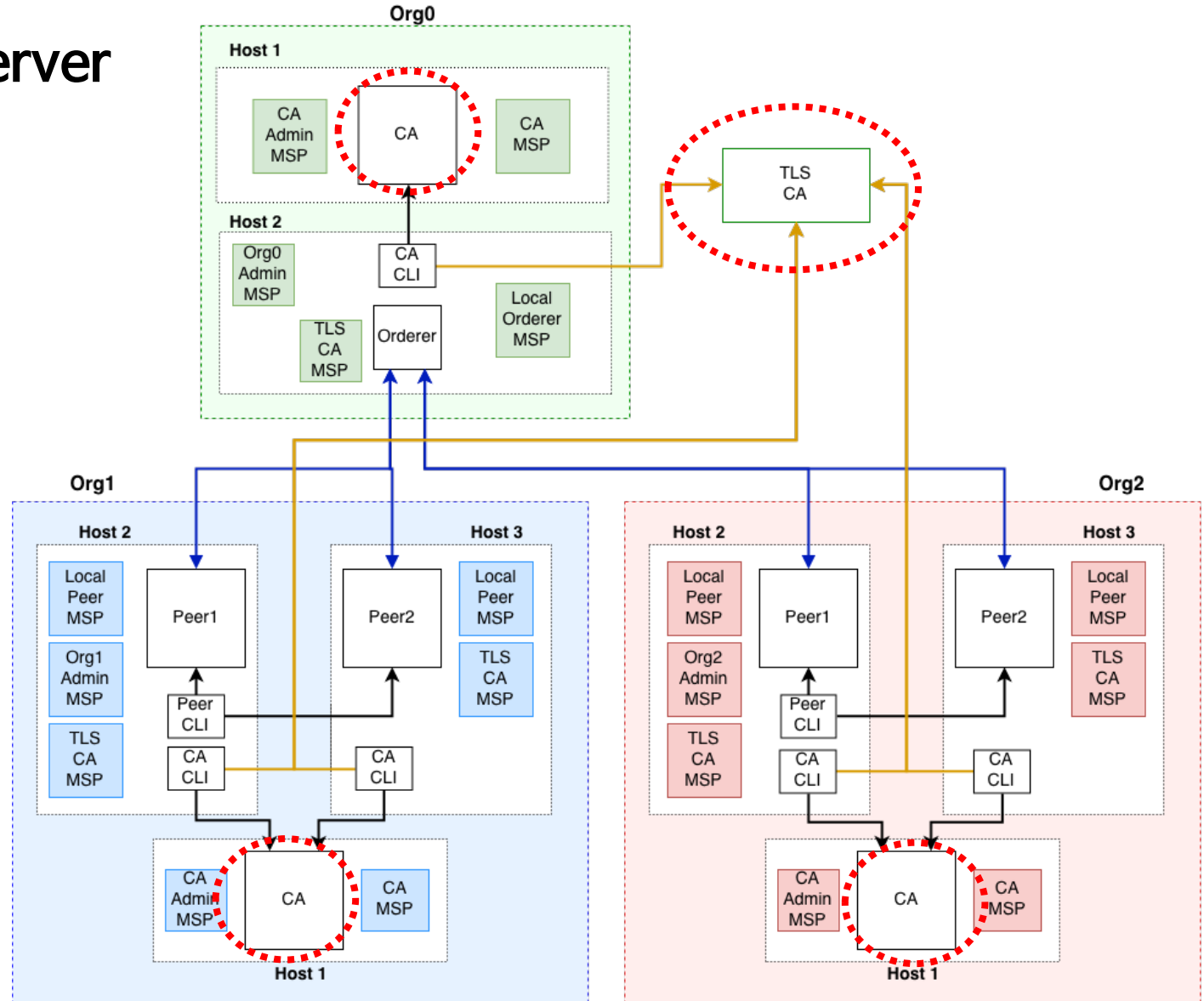
## • Fabric CA Server

- Optional (하지만 대부분 사용)
- 다음 키 들을 관리 (register, enroll, revoke)
  - 사용자 키: 트랜잭션 서명에 사용
  - Orderer/Peer 키: MSP로 사용
  - TLS 키: TLS 통신에 사용
- 키 생성이 완료되면, 오프라인 상태가 되어도 상관 없음
- 테스트에서는 MSP, TLS 키는 cryptogen 으로 생성, 사용자만 Fabric CA 사용
- 프로덕션에서는 모두 Fabric CA 사용 권장



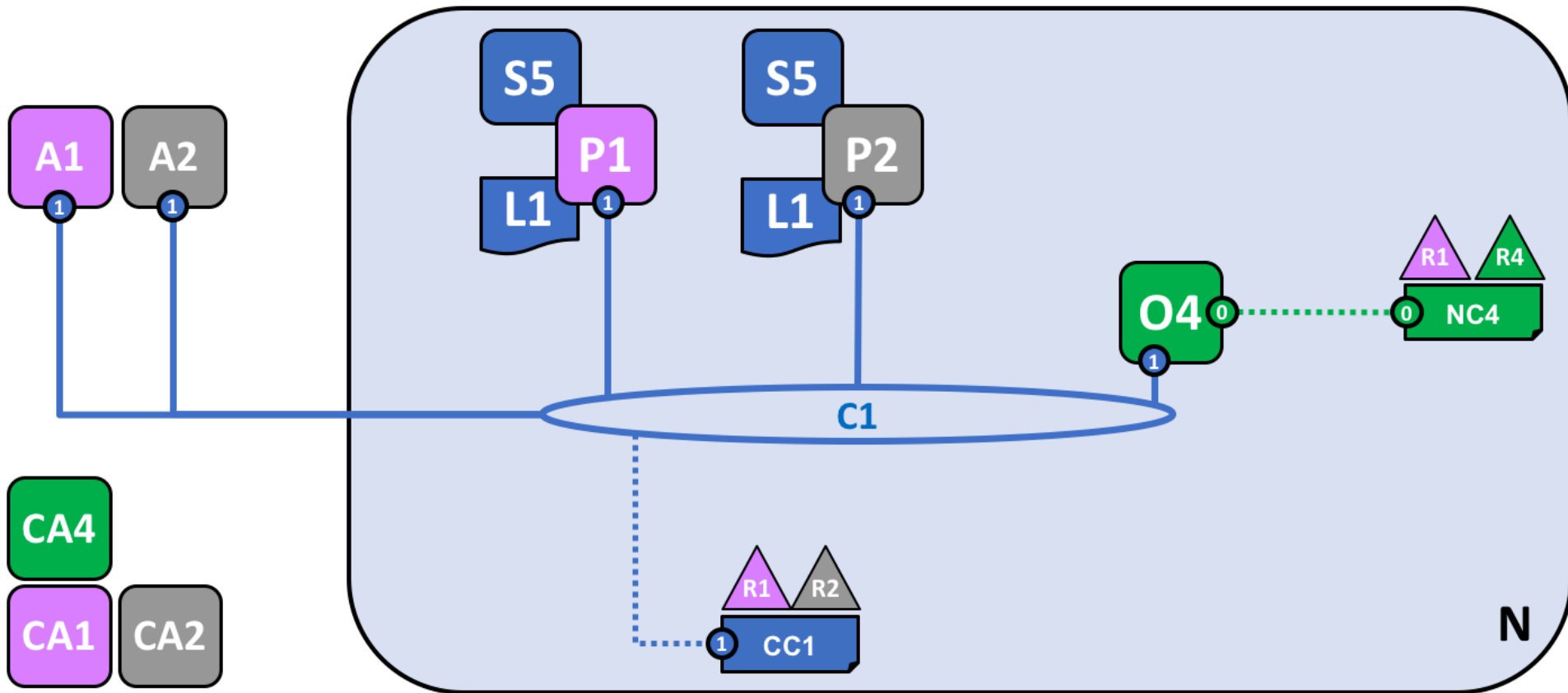
# 1. 하이퍼레저 패브릭 구조

- Fabric CA Server



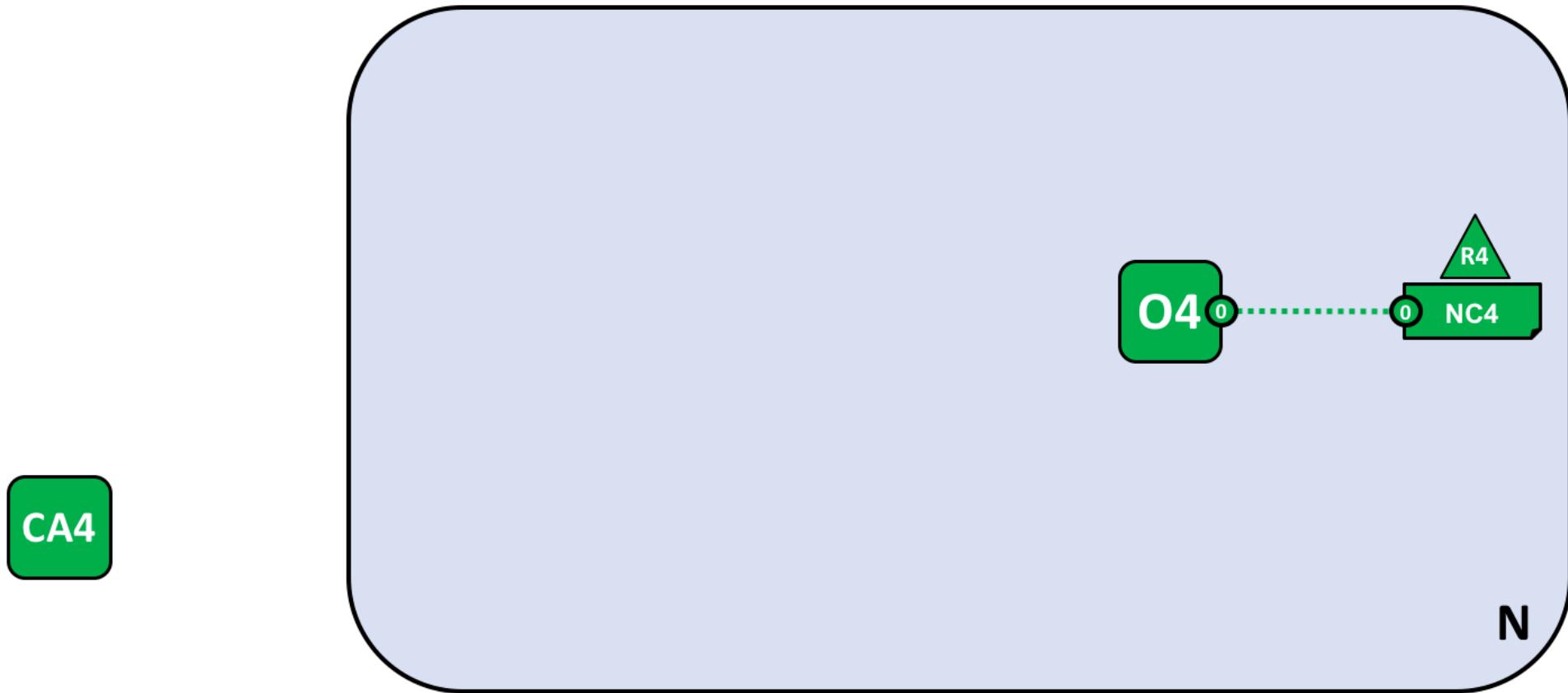
## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- 최종 네트워크



## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

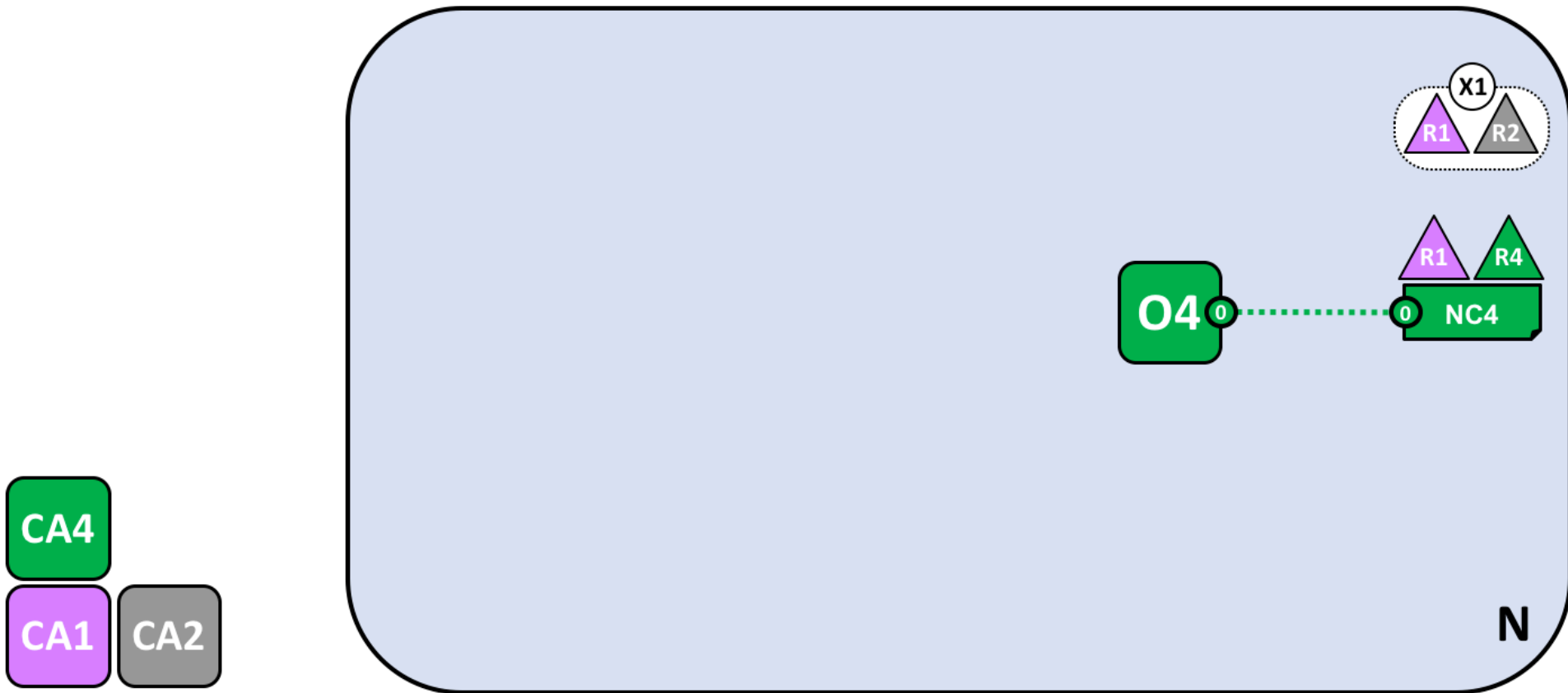
- Orderer 구동





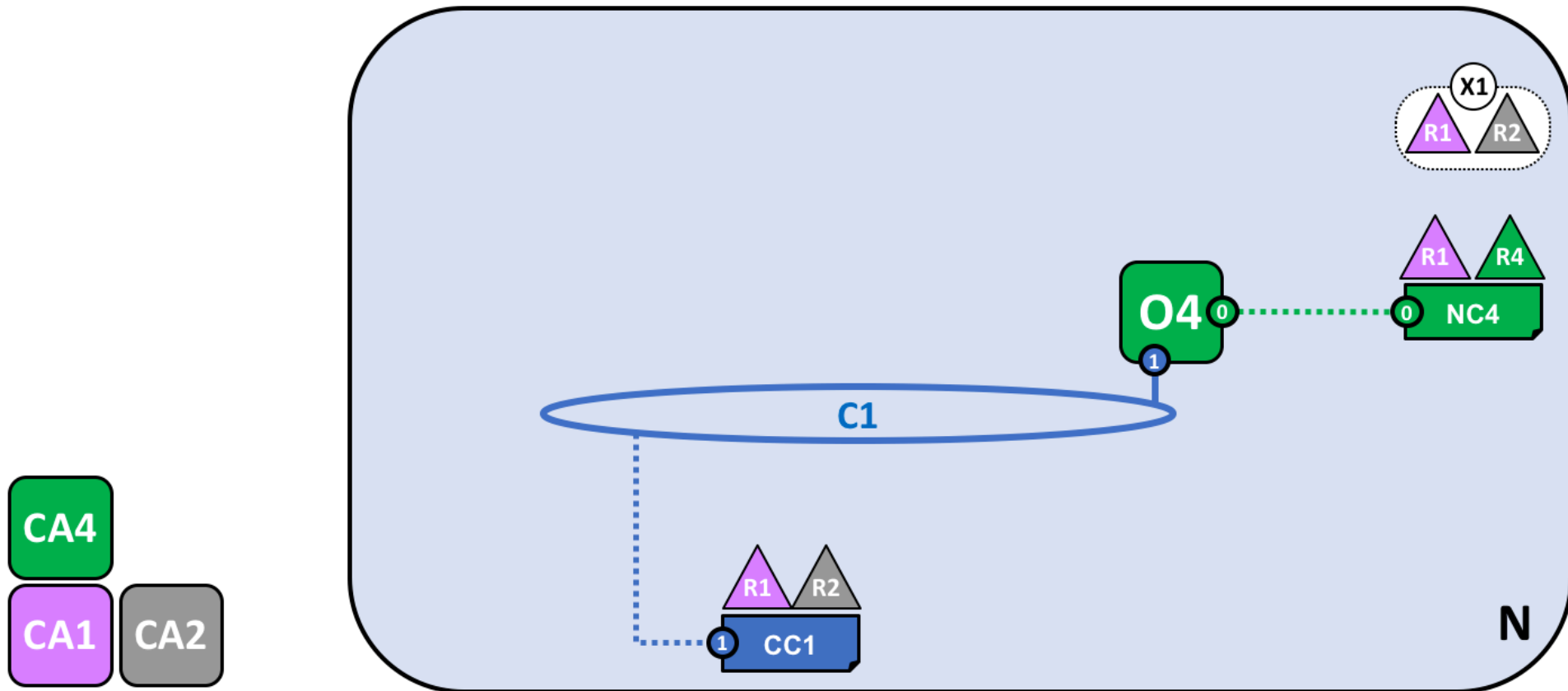
## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- 콘소시움 정의



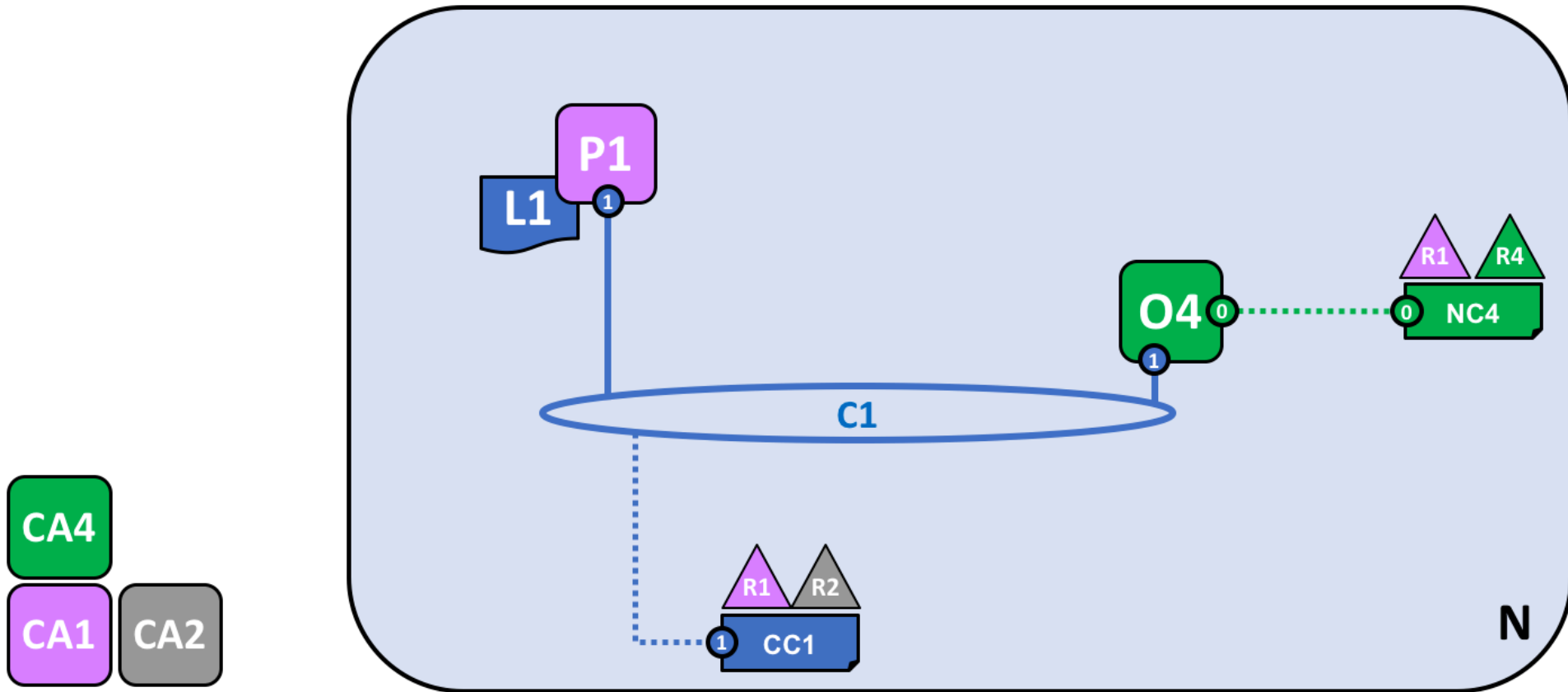
## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- 채널 생성



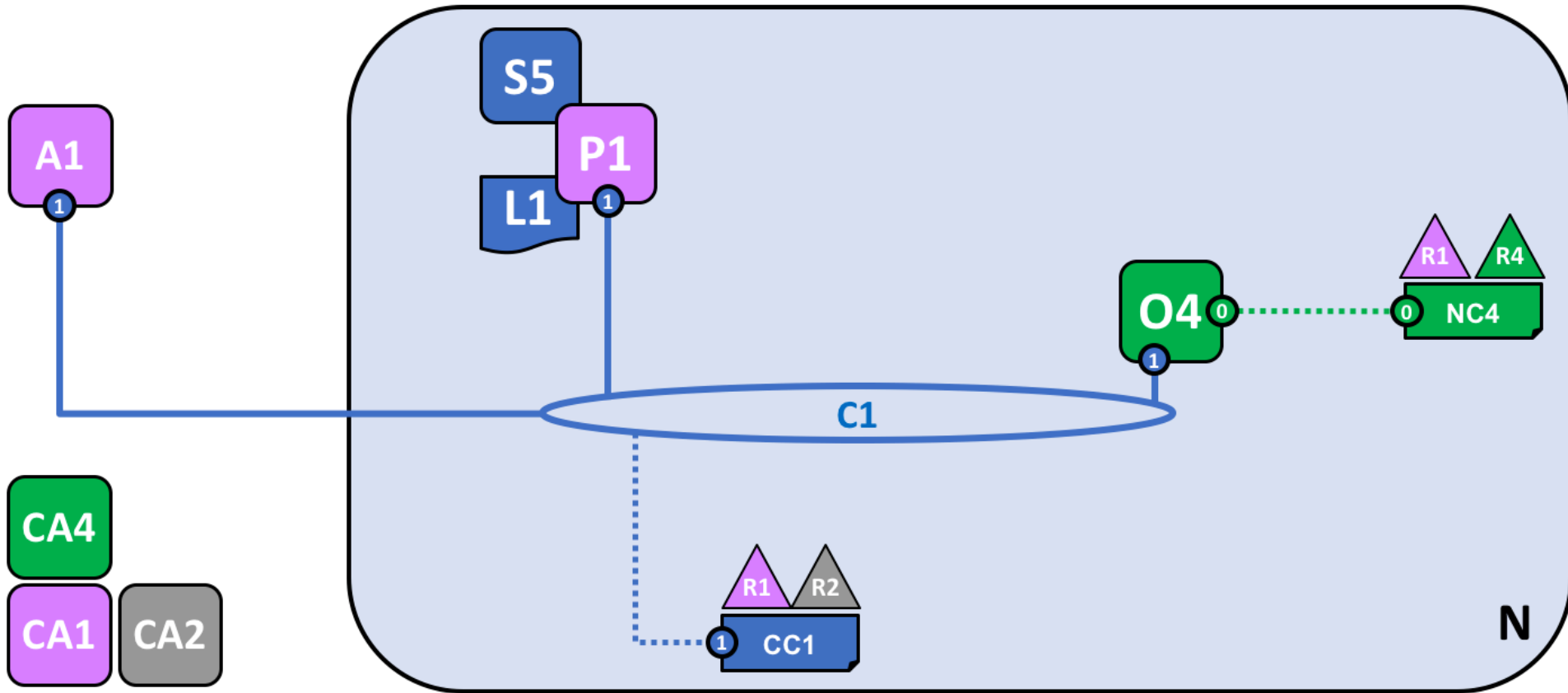
## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- Org1의 Peer1 채널 조인



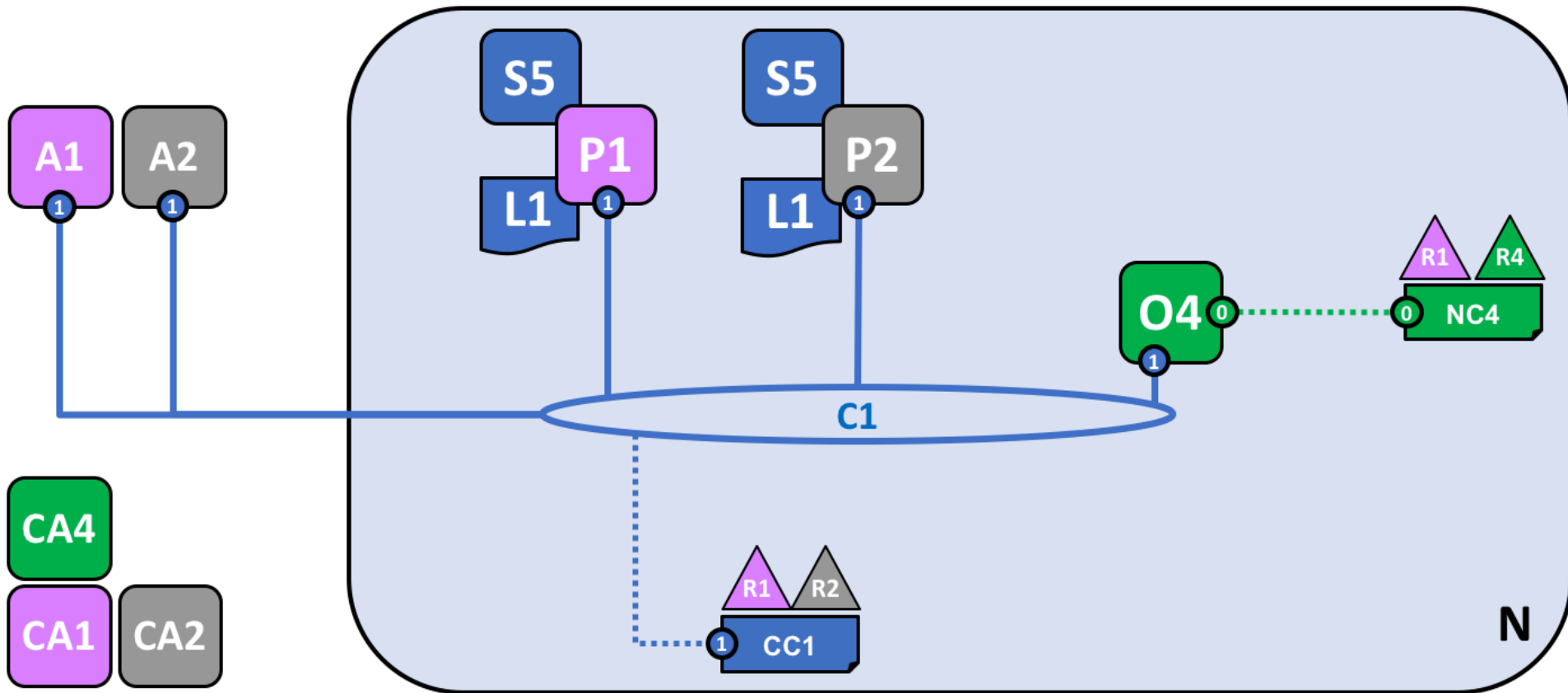
## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- Org1의 Peer1에 체인코드 배포하고, Client Server 구동

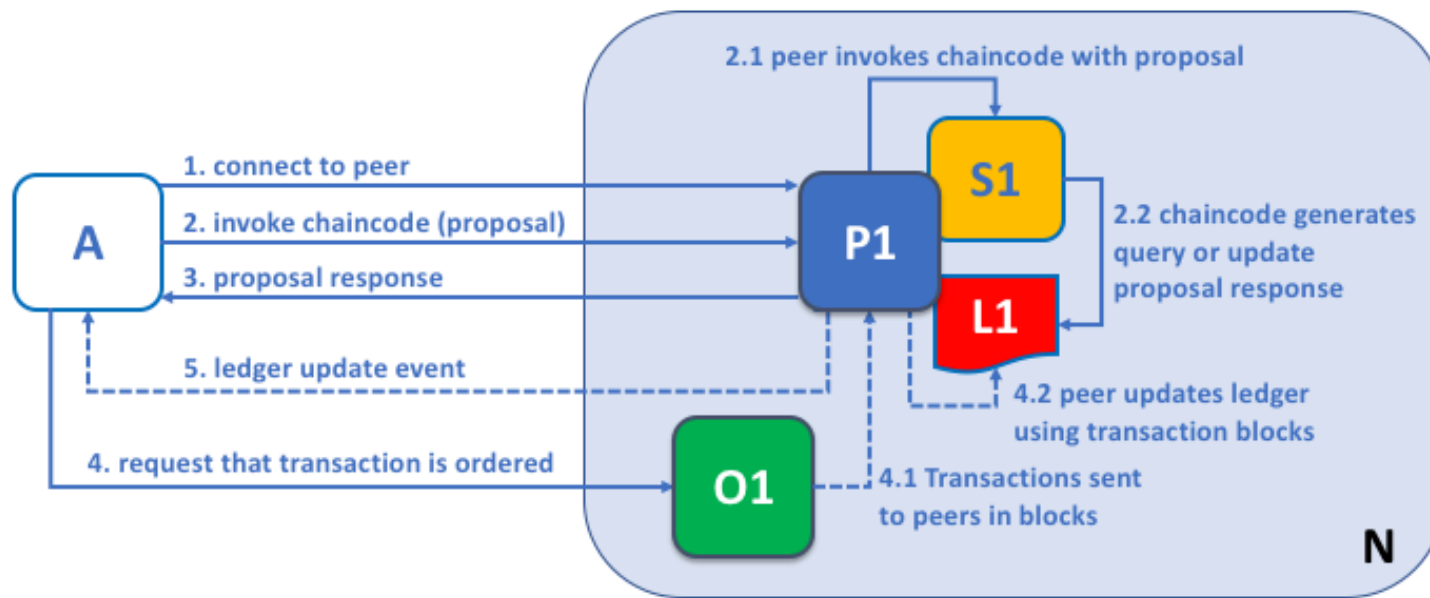


## 2. 하이퍼레저 패브릭 네트워크 구동 시나리오

- Org2에 대해 채널 조인, 체인코드 배포, Client Server 구동



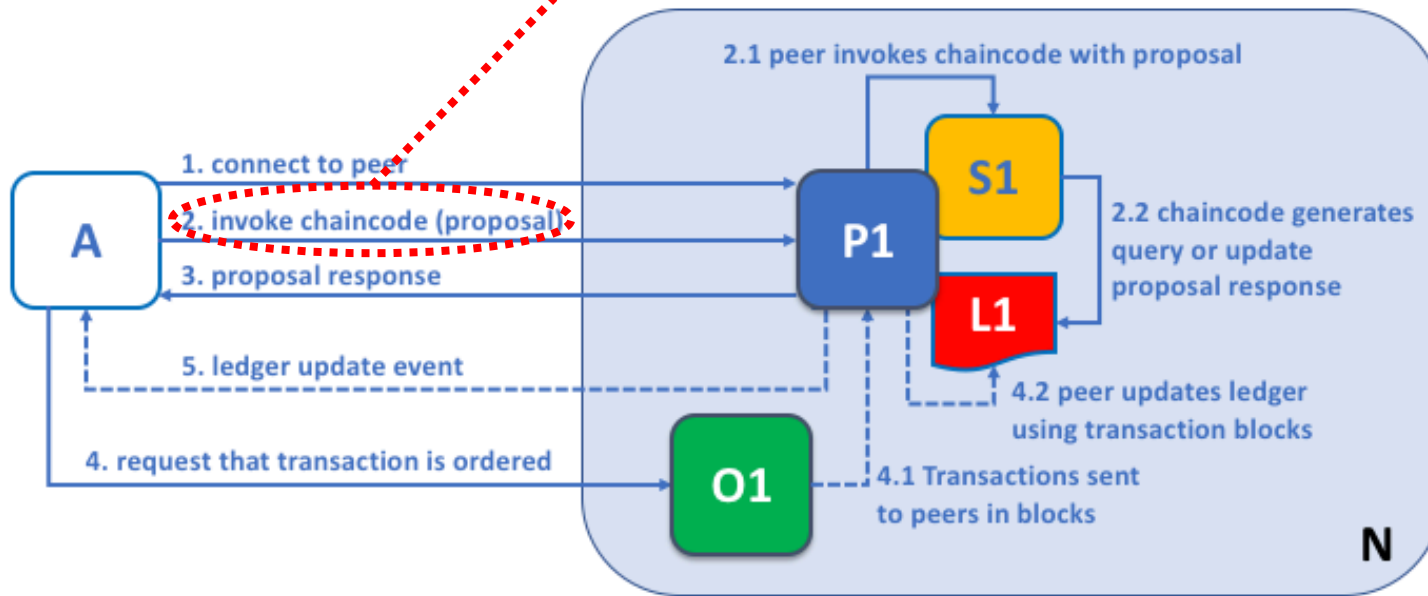
# 3. 트랜잭션 전송 시나리오



N	Blockchain Network
A	Application
P	Peer
S	Chaincode
L	Ledger
O	Orderer

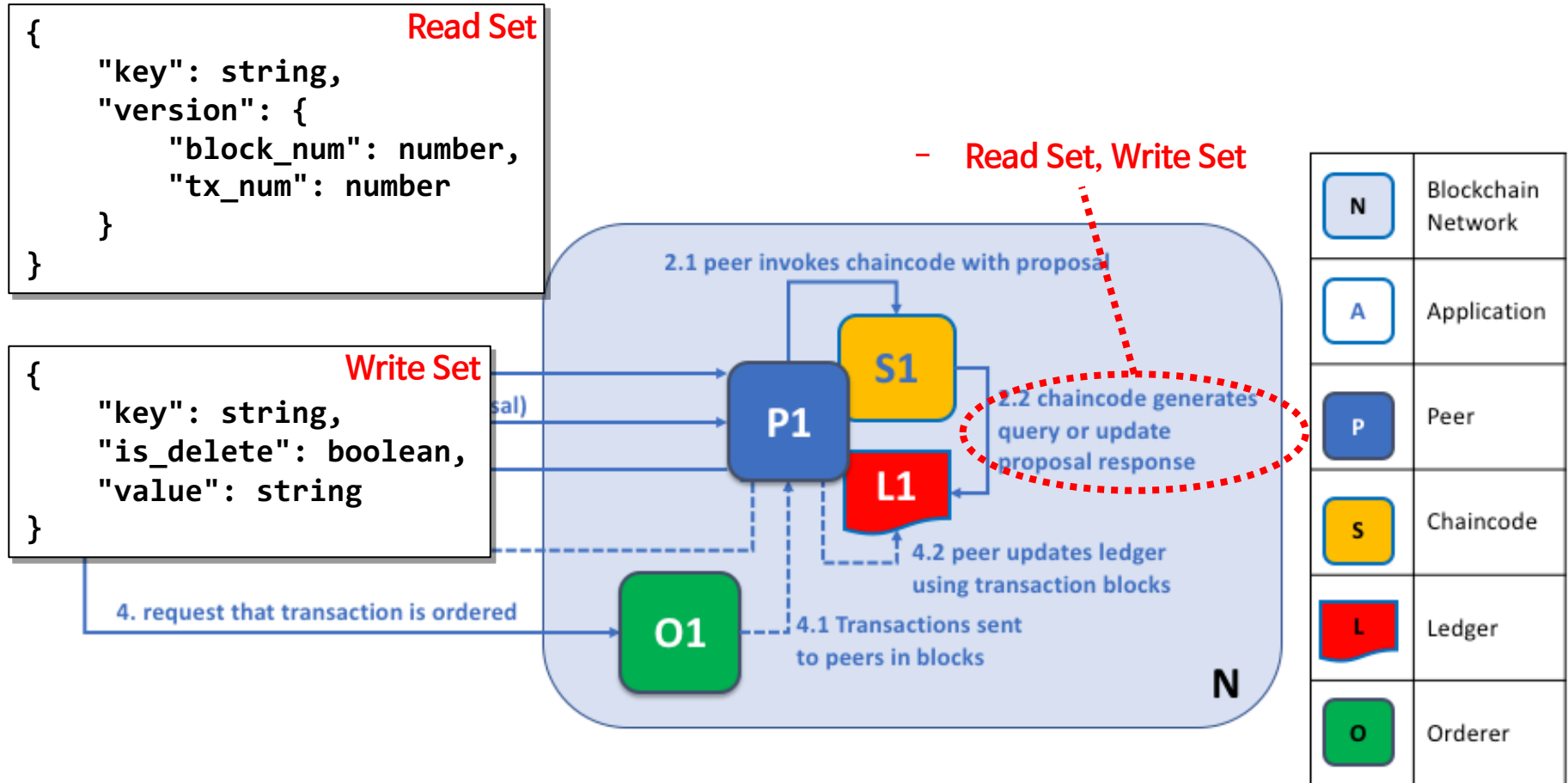
# 3. 트랜잭션 전송 시나리오

- 어떤 Peer에게 보낼지 명시적으로 결정
- Service Discovery 사용



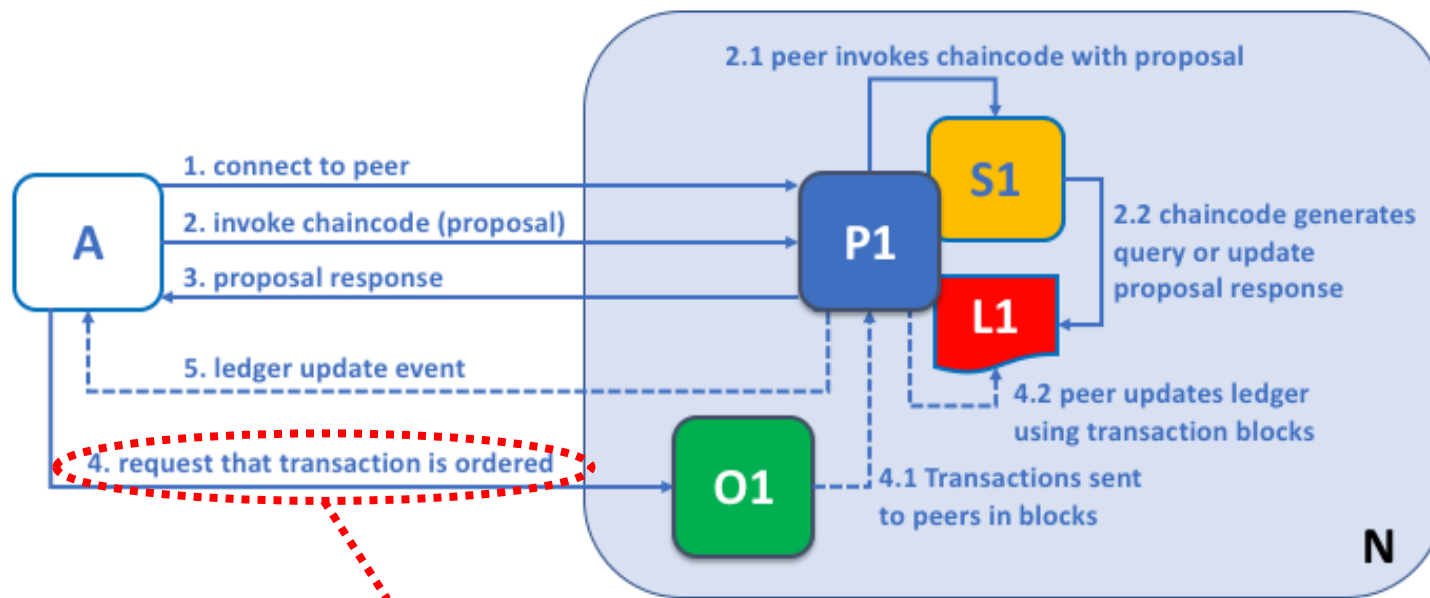
N	Blockchain Network
A	Application
P	Peer
S	Chaincode
L	Ledger
O	Orderer

# 3. 트랜잭션 전송 시나리오





# 3. 트랜잭션 전송 시나리오



N	Blockchain Network
A	Application
P	Peer
S	Chaincode
L	Ledger
O	Orderer

- 명시적으로 어떤 Orderer에게 전송할지 결정 해주어야 함

# 4. 체인코드 구조

- 체인코드 SDK

- Go, JavaScript (node.js), Java

- 체인코드 실행 플로우

- Init 또는 Invoke 실행
- Function & Arguments 파싱
- Function에 따라 분기
- Arguments 파싱
- 로직 실행 후 pb.response 객체 반환

…그냥 LevelDB에 대한 CRUD 함수이다

# 4. 체인코드 구조

- 체인코드의 새로운 SDK (2.0 ~)

- github.com/hyperledger/fabric-contract-api-go/contractapi (Go)
  - <https://pkg.go.dev/github.com/hyperledger/fabric-contract-api-go/contractapi>
- fabric-contract-api (JavaScript)
  - <https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/>
- org.hyperledger.fabric.contract (Java)
  - <https://hyperledger.github.io/fabric-chaincode-java/release-2.2/api/org/hyperledger/fabric/contract/package-summary.html>

## 4. 체인코드 구조

```
func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (*Asset, error) {  
    ...  
}
```

```
await contract.evaluateTransaction(ReadAsset, 'asset13');
```



The diagram illustrates a function call. A red dotted arrow originates from the `ReadAsset` parameter in the `await contract.evaluateTransaction` call and points to the `ReadAsset` function definition in the `SmartContract` struct. Another red dotted arrow originates from the `'asset13'` argument and points to the `id string` parameter in the `ReadAsset` function definition. Both the function name and the argument are circled with red dotted lines.

# 5. 체인코드 개발 시 주의사항

- Query가 좀 더럽다

- Query 할 일이 많지 않도록 -> 상시 벌어지는 Query들은 Off-chain DB 활용
- Key-Value DB (NoSQL DB)
  - Key는 알파벳 순으로 정렬해줌
    - Key를 잘 설계하는 것이 매우 중요
    - stub.CreateCompositeKey 적극 활용

```
stub.CreateCompositeKey(primaryKey, []string{attr1, attr2, ...})
```

```
primaryKey_abc_123_..  
primaryKey_abc_124_..  
primaryKey_abcd_234_..  
primaryKey_abcd_235_..  
              attr1  attr2
```

```
stub.GetStateByPartialCompositeKey(primaryKey, []string{attr1, ...})
```

# 5. 체인코드 개발 시 주의사항

- Random 또는 Endorser마다 달라질 수 있는 값을 생성하는 코드는 절대 삽입 금지
  - ex) `time.Now().Unix()`
  - 커밋 단계에서 PHANTOM CONFLICT 에러 발생



...저만 이런 멍청한 실수를  
하는 건 아니겠죠...?

# 5. 체인코드 개발 시 주의사항

- 사실 체인코드는 별개 다 됩니다.
  - 외부로 HTTP 콜도 보낼 수 있음
  - 다른 체인코드도 호출할 수 있음 (근데 Query만 가능)
- ...사실 Go로 가능한건 대부분 됩니다

하지만 하지 맙시다. 커밋단계에서 깨질 가능성이 너무 랜덤하게 높아집니다.

# 하이퍼레저 패브릭 스마트컨트랙트 개발 실습

2020-10-20

빅픽처랩(주)

안 휘



# 실습 목차

- 개발 환경 구성 (1시간)
  - Go 설치
  - asset-transfer-basic 테스트 구동
- 체인코드 살펴보기 (1시간)
- balance-transfer 개발 (1시간)
  - 사용자 생성
  - 사용자 잔액 확인
  - 송금
- Client Server에서 사용자 생성, 잔액 확인, 송금 호출 (1시간)

# Go 설치

- Go 바이너리 다운로드 및 설치

- `cd ~/`
- `wget https://golang.org/dl/go1.14.10.linux-amd64.tar.gz`
- `sudo tar -xvf go1.14.10.linux-amd64.tar.gz`
- `sudo mv go /usr/local`

- 환경변수 설정

- `vim ~/.bashrc`
- 마지막에 다음 세줄 입력
  - `export GOROOT=/usr/local/go`
  - `export GOPATH=$HOME/go`
  - `export PATH=$GOPATH/bin:$GOROOT/bin:$PATH`

- 설치 확인

- `go version`

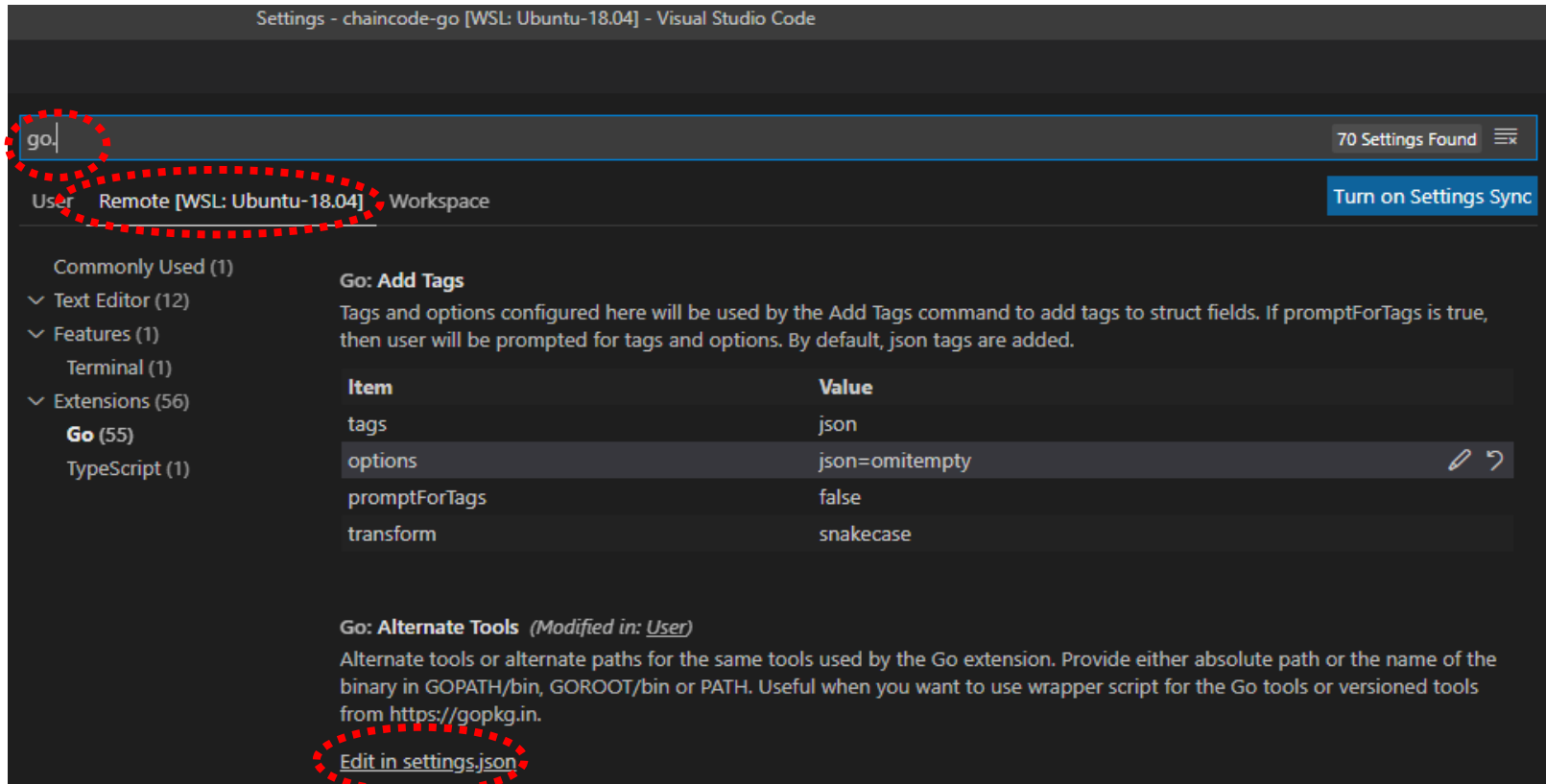
# Go 설치

- **build-essential 설치**
  - `sudo apt-get install build-essential`

# Go 설치

- VSCode 에서 Go 플러그인 설정

- ~/fabric-samples/asset-transfer-basic/chaincode-go 에서 VSCode 오픈
- extension에서 Go 검색
- Install on WSL: Ubuntu-18.04 선택
- VSCode 설정 열기 (Ctrl + ,)



# Go 설치

- VSCode 에서 Go 플러그인 설정

- extension에서 Go 검색
- Install on WSL: Ubuntu-18.04 선택
- VSCode 설정 열기 (Ctrl + ,)
- 설정 JSON 파일에 아래와 같이 입력
  - gopath는 터미널에서 echo \$GOPATH 를 통해 획득

```
{  
  "go.gopath": "/home/byron1st/go",  
  "go.goroot": "/usr/local/go",  
  "go.useLanguageServer": true,  
}
```

- VSCode가 필요한 go 툴을 모두 설치하도록 놔둠
- VSCode 재시작

# asset-transfer-basic 테스트 구동

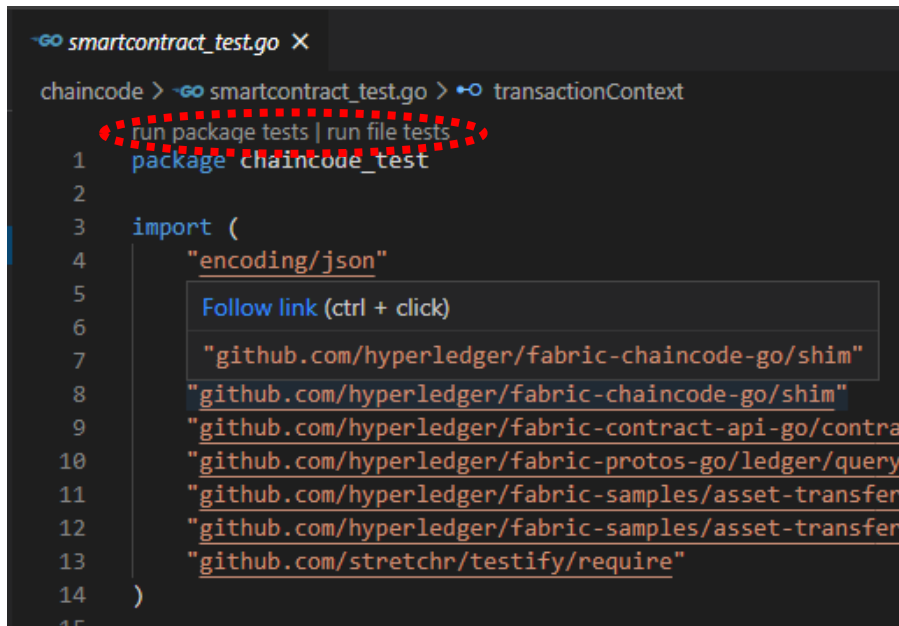
- VSCode 에서 Go 플러그인 설정
  - ~/fabric-samples/asset-transfer-basic/chaincode-go 이동
  - 다음 커맨드 입력
    - go test github.com/hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/chaincode -v

```
byron1st@DESKTOP-D240NBD:~/fabric-samples/asset-transfer-basic/chaincode-go$ go test github.com/hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/chaincode -v
=== RUN   TestInitLedger
--- PASS: TestInitLedger (0.00s)
=== RUN   TestCreateAsset
--- PASS: TestCreateAsset (0.00s)
=== RUN   TestReadAsset
--- PASS: TestReadAsset (0.00s)
=== RUN   TestUpdateAsset
--- PASS: TestUpdateAsset (0.00s)
=== RUN   TestDeleteAsset
--- PASS: TestDeleteAsset (0.00s)
=== RUN   TestTransferAsset
--- PASS: TestTransferAsset (0.00s)
=== RUN   TestGetAllAssets
--- PASS: TestGetAllAssets (0.00s)
PASS
ok      github.com/hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/chaincode 0.026s
```

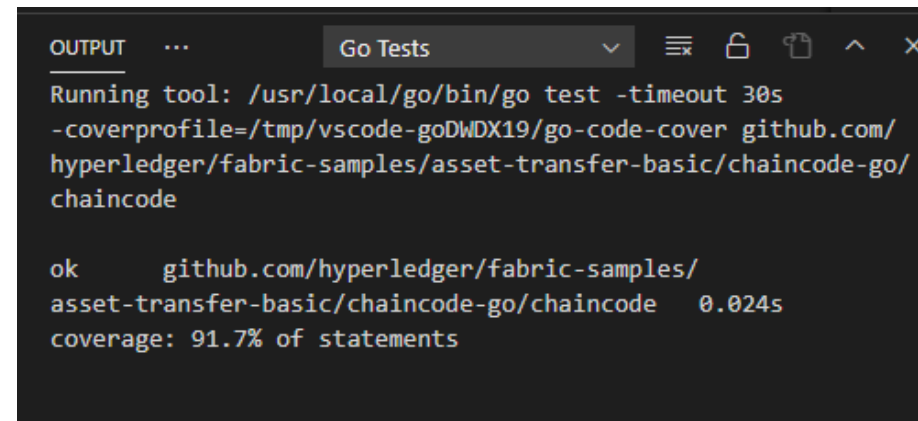
# asset-transfer-basic 테스트 구동

- VSCode 에서 Go 플러그인 설정

- ~/fabric-samples/asset-transfer-basic/chaincode-go를 VSCode에서 오픈
- chaincode/smartcontract\_test.go 파일 선택



```
Go smartcontract_test.go X
chaincode > Go smartcontract_test.go > transactionContext
run package tests | run file tests
package chaincode_test
1
2
3 import (
4     "encoding/json"
5     "github.com/hyperledger/fabric-chaincode-go/shim"
6     "github.com/hyperledger/fabric-chaincode-go/shim"
7     "github.com/hyperledger/fabric-chaincode-go/shim"
8     "github.com/hyperledger/fabric-contract-api-go/contractapi"
9     "github.com/hyperledger/fabric-protos-go/ledger/querypattern"
10    "github.com/hyperledger/fabric-protos-go/ledger/querypattern"
11    "github.com/hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/chaincode"
12    "github.com/hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/chaincode"
13    "github.com/stretchr/testify/require"
14 )
15
```



```
OUTPUT ... Go Tests
Running tool: /usr/local/go/bin/go test -timeout 30s
-coverprofile=/tmp/vscode-goDWDX19/go-code-cover github.com/
hyperledger/fabric-samples/asset-transfer-basic/chaincode-go/
chaincode

ok      github.com/hyperledger/fabric-samples/
asset-transfer-basic/chaincode-go/chaincode  0.024s
coverage: 91.7% of statements
```

# 체인코드 살펴보기

- Go 언어에서의 method

- Struct 와 함수 연결
- 입력, 반환 파라미터 표시 방법

```
// SmartContract provides functions for managing an Asset
type SmartContract struct {
    contractapi.Contract
}

...
func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
    ...
}

func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (*
Asset, error)

...
}
```



# 체인코드 살펴보기

- **ctx.getStub()**
  - shim.ChaincodeStubInterface 객체를 반환
    - GetState
    - PutState
      - []byte 타입은 객체를 json.marshal 하여 얻음
    - DelState
    - ...
- **json.marshal / unmarshal**
- **golang tag**

# balance-transfer 개발

- 저장할 모델 정의

- 사용자 ID, 잔액 정보 정도면 충분할 듯

- 함수 2개 만들기

- createUser
  - user ID의 중복 여부 확인
  - 만들면 자동으로 기본 돈 100원 지급
- balance
  - userID 가 없으면 에러
  - 있으면 잔액 반환
- transferMoney
  - senderID, recipientID, value를 받아서 처리
  - 돈이 부족하면 에러 반환

# Client Server에서 사용자 생성, 잔액 확인, 송금 호출

- **asset-transfer-basic/application-javascript**

- test-network 는 - ca 모드로 실행되어야 함
  - ./network.sh up -ca
  - ./network.sh createChannel
  - ./network.sh deployCC
- wallet 폴더의 키들은 실행 전에 삭제되어야 함
  - 해당 키 들은 예전에 생성했던 test-network의 키들임