3. How Ethereum Blockchain works

Contents

1. 이더리움 소개

이더리움이란?

- 비트코인과의 차이점 : 스마트 컨트랙트와 탈중앙화 애플리케이션(DApps)
- 이더리움의 목표 : “세계 컴퓨터"를 구축

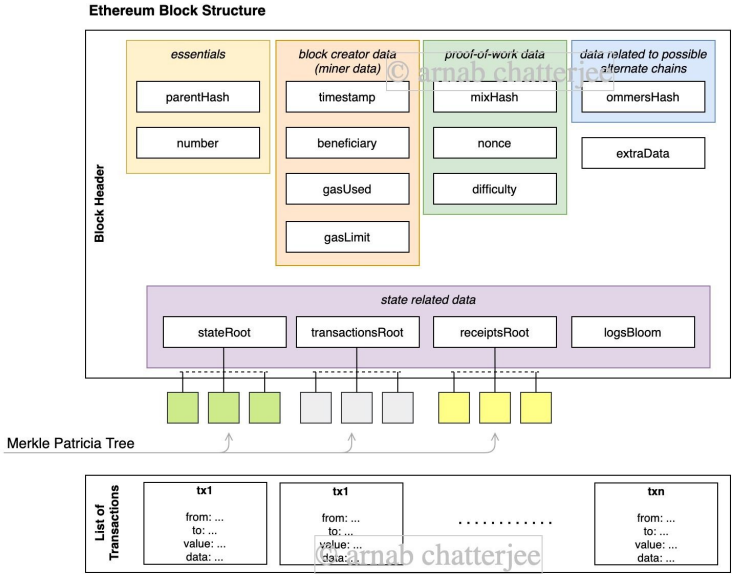
핵심 구성 요소

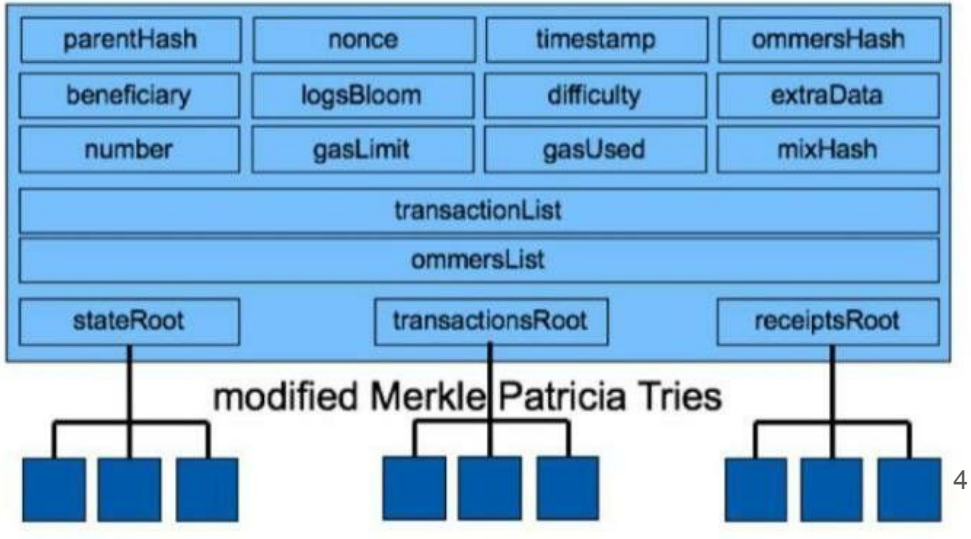
- 스마트 컨트랙트
- 이더(Ether, ETH) : 네트워크에서 사용되는 기본 토큰

2. 이더리움 블록체인 구조

블록 구성 요소

- **Block Header :** Parent Hash, Nonce, Difficulty 등
- **Block Body :** 트랜잭션 데이터 및 스마트 컨트랙트 호출





2. 이더리움 블록체인 구조

Account 모델

- 비트코인 UTXO 모델과 달리, 이더리움은 **계정(Account)**을 기반으로 잔액과 상태를 관리
- 트랜잭션은 계정 간의 잔액변동 및 상태 업데이트로 이루어짐

Account 모델의 두 종류

- **External Account**
  - 개인 사용자가 소유
  - 개인 키로 제어되며, ETH 전송 및 스마트 컨트랙트 호출에 사용
  - **구성요소 :** 잔액(Balance), 개인 키(Private Key) 와 공개 키 (Public Key)
- **Contract Account**
  - 스마트 컨트랙트를 기반으로 동작
  - 특정 조건이 충족되면 자동으로 실행
  - **구성요소 :** 스마트 컨트랙트 코드, 상태(state : 스마트 컨트랙트가 저장하는 데이터)

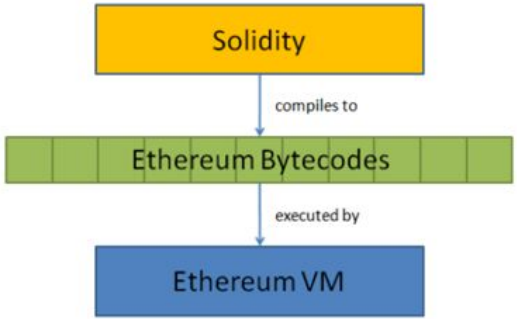
2. 이더리움 블록체인 구조

특징	UTXO 모델 (비트코인)	Account 모델 (이더리움)
기반	UTXO (미사용 트랜잭션 출력)	계정 (Account) 및 상태 (State)
잔액 관리 방식	여러 UTXO를 소모하고 새로운 UTXO를 생성	계정 간 잔액을 변경
스마트 컨트랙트 지원	불가능	가능
트랜잭션 추적	각 UTXO를 추적해 전체 경로를 확인	계정 잔액 변화를 통해 단순히 확인
복잡도	트랜잭션 검증이 상대적으로 복잡함	상태 업데이트가 간단
주요 용도	디지털 화폐 송금 (비트코인)	스마트 컨트랙트 및 DApp 실행 (이더리움)

3. 이더리움 가상 머신(EVM)

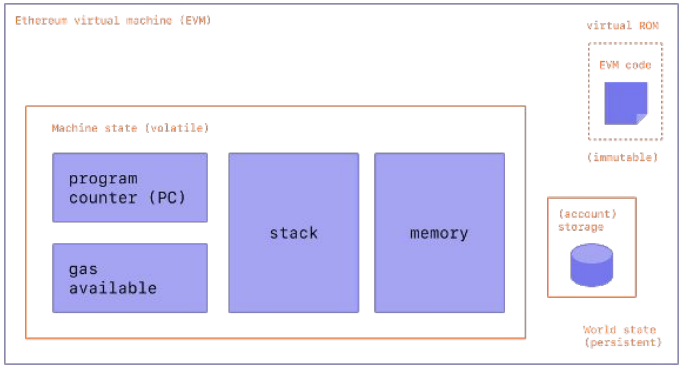
EVM의 역할

- 스마트 컨트랙트 실행 환경
- 바이트코드(Bytecode)를 해석하고 실행



작동 방식

- 상태(State) 트랜지션과 코드 실행
  * EVM 무료 아니며, GAS비 지불해야함



스마트 컨트랙트 언어

- Solidity, Vyper 등의 언어로 작성된 코드를 EVM에서 실행

4. 트랜잭션 작동 원리

이더리움 트랜잭션 구조

- 이더리움 트랜잭션은 송금, 스마트 컨트랙트 호출 등 블록체인 상의 모든 작업 요청을 의미
- 트랜잭션은 아래와 같은 주요 필드로 구성

필드	설명
송신자 (Address)	트랜잭션을 보낸 계정의 주소. 해당 주소의 개인 키로 서명됨
수신자 (Address)	트랜잭션을 받는 계정 (외부 계정 또는 컨트랙트 계정)
값 (Value)	수신자에게 보낼 Ether (ETH)의 양
데이터 (Data)	스마트 컨트랙트 호출 시 전송할 데이터 (함수명, 매개변수 등) 스마트 컨트랙트 호출이 아닐 경우 빈값
Gas Limit	트랜잭션 실행에 사용 가능한 최대 Gas 양
Gas Price	Gas 당 지불할 가격
Nonce	송신자의 트랜잭션 번호로, 중복 방지를 위해 사용됨
서명 (Signature)	송신자의 개인 키로 생성된 서명. 트랜잭션의 무결성과 정당성을 증명

4. 트랜잭션 작동 원리

트랜잭션 실행 과정

1. **트랜잭션 생성 및 서명**
   - 1.1. 송신자가 트랜잭션 데이터 입력하고 자신의 **개인 키(Private Key)**로 서명
   - 1.2. 서명된 트랜잭션은 네트워크로 브로드캐스트

2. **노드 간 전파**
   - 2.1. P2P 네트워크를 통해 트랜잭션이 주변 노드에 전파
   - 2.2. 모든 노드는 트랜잭션을 **Mempool(트랜잭션 대기열)**에 저장

3. **트랜잭션 검증**

   각 노드가 트랜잭션의 유효성을 확인
   - 3.1. **서명 검증 :** 공개 키로 서명 무결성 확인
   - 3.2. **Nonce 검증 :** 중복 트랜잭션 방지를 위해 송신자의 마지막 Nonce와 비교

     * 이더리움에서는 트랜잭션 순서 보장을 위해 nonce 를 사용 (각 계정마다 고유)
   - 3.3. **잔액 확인 :** 송신 계정의 잔액이 충분한지 확인 (전송 금액 + Gas Fee)

예를 들어:

1. **계정 A**가 트랜잭션을 보낼 때:
   - 처음 트랜잭션: Nonce = `0`
   - 두 번째 트랜잭션: Nonce = `1`
   - 세 번째 트랜잭션: Nonce = `2`
2. **계정 B**도 트랜잭션을 보낸다면:
   - 처음 트랜잭션: Nonce = `0`
   - 두 번째 트랜잭션: Nonce = `1`

4. 트랜잭션 작동 원리

트랜잭션 실행 과정

4. EVM에서 실행

4.1. 유효한 트랜잭션은 **Ethereum Virtual Machine(EVM)**에서 실행

4.2. 스마트 컨트랙트 호출 시, 데이터(Data) 필드에 포함된 코드가 실행되어 상태(State)가 변경

5. 상태(State) 업데이트

5.1. 트랜잭션이 실행된 후, 결과가 글로벌 상태(Global State Trie)에 반영

- 송신자의 잔액 감소
- 수신자의 잔액 증가 또는 스마트 컨트랙트 상태 변경

5.2. 실행 결과는 블록에 기록되고 블록체인에 영구 저장

4. 트랜잭션 작동 원리

Gas Fee (가스 요금)

- **Gas란?**
  - 이더리움에서 스마트 컨트랙트와 트랜잭션 실행에 필요한 계산 작업의 단위
  - 트랜잭션 실행 시, Gas 사용량과 Gas 가격을 곱한 값이 수수료로 부과

- **Gas Fee 계산 공식**

  Total Fee = Gas Used X Gas Price

  - Gas Used : 트랜잭션 실행에 사용된 Gas 양
  - Gas Price : 사용자가 설정한 Gas 가격 (일반적으로 Gwei 단위)

- **Gas Limit 과 Gas Used**
  - **Gas Limit :** 사용자가 설정한 Gas 사용 한도 (트랜잭션 실행에 필요한 양보다 낮으면 실행 실패)
  - **Gas Used :** 실제로 트랜잭션 실행에 소모된 Gas 양 (사용하지 않은 Gas는 돌려 받음)

4. 트랜잭션 작동 원리

Gas Fee (가스 요금)

- **비용 최적화의 중요성**
  - 사용자는 네트워크 혼잡도와 Gas Price를 고려해 적절한 Gas Price를 설정해야 함
  - **네트워크 혼잡 시 :**
    높은 Gas Price 를 설정하면 트랜잭션이 더 빠르게 처리
  - **스마트 컨트랙트 최적화 :**
    복잡한 코드 실행은 더 많은 Gas를 소모하므로 효율적으로 설계해줘야 함

- **예제**
  - **Gas Limit :** 21,000 (기본 전송 트랜잭션의 Gas 소모량)
  - **Gas Price :** 50 Gwei
  - **Total Fee :** 21,000 X 50 Gwei = 0.00105 ETH
    (cf. 1ETH = 1,000,000,000 Gwei, 1ETH의 10억 분의 1)

5. PoS 합의 알고리즘

이더리움 2.0의 Proof of Stake(PoS) 전환은 에너지 효율성을 높이고 확장성을 개선하기 위한 주요 변화

항목	Proof of Work (PoW)	Proof of Stake (PoS)
에너지 소비	매우 높음	낮음
보안	강력 (작업 증명 기반)	지분 기반 보안(지분 집중 가능성)
블록 생성 방식	SHA-256 해시 계산	지분량 기반 확률적 선택
블록체인 적용 사례	비트코인, 라이트코인 등	이더리움 2.0, 카르다노 등

[PoW와 PosW의 차이점]

5. PoS 합의 알고리즘

PoS에서 검증자(Validator)의 역할

- **Validator란?**
  - 검증자는 블록 생성 및 네트워크의 무결성을 유지하는 역할을 수행
  - PoS에서 노드는 검증자가 되기 위해 일정량의 ETH를 네트워크에 스테이킹(Staking) 해야함
    * 이더리움의 경우 최소 32 ETH를 예치해야 검증자가 될 수 있음

- **Validator의 주요 작업**
  - **블록 생성**
    - 네트워크가 검증자를 선택하면 해당 검증자가 새로운 블록을 생성
    - 생성된 블록은 네트워크에 브로드캐스트 됨
  - **블록 검증**
    - 다른 검증자가 블록의 유효성을 확인 (블록 내 트랙잭션 무결성 확인)
    - 블록 헤더의 Previous Hash 와 체인 연결 확인
  - **합의 참여**
    - 블록의 유효성에 대해 다른 검증자와 합의에 도달
    - 합의 메커니즘을 통해 새로운 블록이 체인에 추가

5. PoS 합의 알고리즘

블록 생성 과정 (PoS에서의 합의 메커니즘)

- **블록 생성 단계**
  - **검증자 선정**
    - 네트워크는 **랜덤 알고리즘**을 사용해 검증자를 선택
    - 스테이킹된 **ETH의 양에 따라 선택 확률**이 결정됨
  - **블록 생성**
    - 선택된 검증자가 블록을 생성하고 네트워크에 전파
  - **블록 검증 및 합의**
    - 다른 검증자들이 새로 생성된 블록의 트랜잭션과 상태(State)가 올바른지 검증
    - 검증에 성공한 블록은 체인에 추가되고, 검증자는 보상을 받음

- **합의의 주요 요소**
  - PoS는 Finality(최종성)라는 개념을 사용
    - 블록이 네트워크에 완전히 확장되면 더 이상 변경 불가능
    - 검증자들이 **다수의 서명**을 통해 블록의 최종성을 결정

6. 스마트 컨트랙트와 DApps

스마트 컨트랙트란?

- **정의 :**
  - 스마트 컨트랙트는 자동화 된 계약을 실행하는 프로그램
  - 이더리움 블록체인에 배포된 코드로, 특정 조건이 충족되면 미리 정의된 동작을 자동으로 실행

- **스마트 컨트랙트의 특징**
  - **자동 실행**
    - 조건이 만족되면 코드가 자동으로 실행
      (ex. 구매자가 결제 완료하면 판매자에게 디지털 상품이 자동으로 전달)
  - **신뢰성**
    - 블록체인에 저장되어 누구도 변경할 수 없음
    - 계약 조건은 모든 사용자가 검증 가능
  - **탈중앙화**
    - 중앙 기관 없이 네트워크의 모든 노드가 스마트 컨트랙트를 실행
  - **투명성**
    - 코드와 실행 결과가 블록체인네 기록되어 모두가 확인 가능

6. 스마트 컨트랙트와 DApps

스마트 컨트랙트 주요 구성 요소

- **함수(Function) :** 특정 동작을 정의
- **상태(State) :** 계약 상태를 저장
- **이벤트(Event) :** 상태 변경 시 발생하는 로그

6. 스마트 컨트랙트와 DApps

스마트 컨트랙트의 실행 과정

1. **트랜잭션을 통해 호출**
   - 1.1. 사용자가 스마트 컨트랙트를 호출하려면 트랜잭션을 생성하여 네트워크에 전송
   - 1.2. 트랜잭션은 스마트 컨트랙트 주소(Address)와 함께 실행할 함수 및 데이터를 포함

2. **코드 실행 (EVM)**
   - 2.1. 트랜잭션이 이더리움 가상 머신(EVM)에서 실행 됨
   - 2.2. 사용 된 Gas에 따라 트랜잭션 비용이 계산됨

3. **상태(State) 변경**
   - 3.1. 스마트 컨트랙트 실행 결과로 상태가 변경됨
     ex. 사용자가 토큰을 전송하면 스마트 컨트랙트의 잔액 데이터가 업데이트

6. 스마트 컨트랙트와 DApps

스마트 컨트랙트의 코드 예제 (Solidity 기반)

```
// ERC-20 간단 구현 예제
pragma solidity ^0.8.0;

contract Token {
    mapping(address => uint256) public balances;

    // 토큰 초기 공급
    constructor() {
        balances[msg.sender] = 1000; // 컨트랙트 소유자에게 1000 토큰 제공
    }

    // 토큰 전송 함수
    function transfer(address recipient, uint256 amount) public {
        require(balances[msg.sender] >= amount, "잔액 부족");
        balances[msg.sender] -= amount;
        balances[recipient] += amount;
    }
}
```

6. 스마트 컨트랙트와 DApps

탈중앙화 애플리케이션 (DApps)

- **DApp이란?**
  - 탈중앙화 애플리케이션(DApp)은 스마트 컨트랙트를 백엔드로 사용하는 애플리케이션
  - 사용자 인터페이스는 일반 웹 애플리케이션처럼 보이지만, 데이터와 로직은 탈중앙화된 블록체인에 저장됨

- **DApp의 구조**
  - **프론트엔드**
    - 사용자와 상호작용하는 웹 또는 모바일 애플리케이션
    - 일반적으로 HTML, CSS, JavaScript 를 사용
    - MetaMask와 같은 지갑을 통해 블록체인과 상호작용

  - **스마트 컨트랙트 백엔드**
    - DApp의 핵심 로직을 처리하는 스마트 컨트랙트 (ex. 토큰 전송, 데이터 기록, 투표 결과 집계)

  - **DApp의 작동 원리**
    - 사용자가 DApp의 프론트엔드를 통해 작업을 요청
    - 요청이 트랜잭션 형태로 블록체인에 전송되어 스마트 컨트랙트를 실행
    - 실행 결과가 블록체인에 저장되고, 사용자에게 반환

6. 스마트 컨트랙트와 DApps

DApp의 예시

- **DeFi (탈중앙화 금융)**
  - **DeFi**는 탈중앙화된 금융 서비스를 제공하는 DApp
    * 예시 : 대출, 예금, 유동성 공급
  - 대표적인 DeFi 플랫폼
    - Avae : 암호화폐 대출 및 차입 서비스
    - Uniswap : 탈중앙화 거래소(DEX)

- **NFT 마켓플레이스**
  - **NFT(Non-Fungible Token)** 마켓플레이스는 디지털 자산의 소유권을 거래하는 플랫폼
    * 예시 : 디지털 아트, 음악, 게임 아이템
  - 대표적인 NFT 마켓플레이스
    - OpenSea : NFT생성, 구매 및 판매
    - Rarible : 사용자 주도형 NFT 플랫폼

6. 스마트 컨트랙트와 DApps

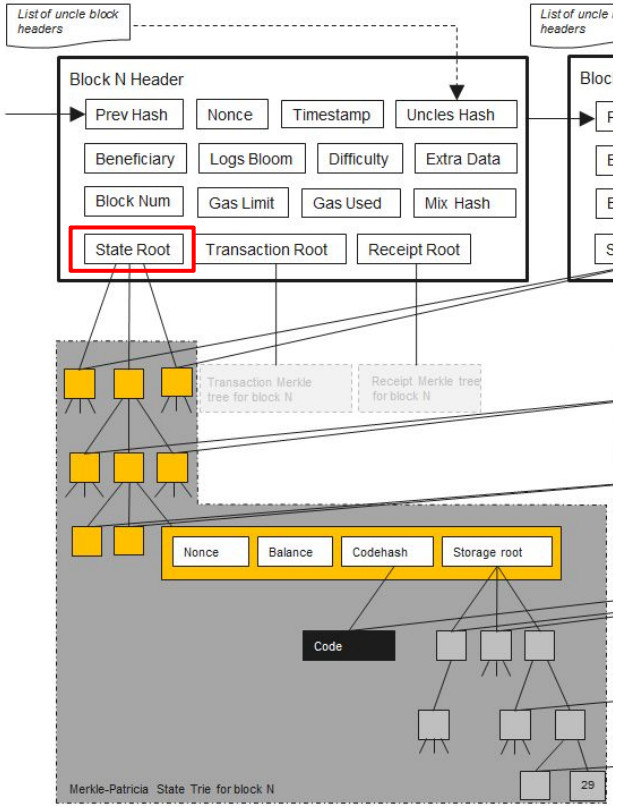
스마트 컨트랙트와 DApp의 차이

항목	스마트 컨트랙트	DApp
정의	블록체인 상의 자동화된 계약 코드	스마트 컨트랙트를 기반으로 작동하는 앱
역할	특정 조건이 충족되면 실행	사용자와 블록체인 간 상호작용 지원
구성 요소	함수, 상태, 이벤트	프론트엔드 + 스마트 컨트랙트 백엔드
예시	간단한 토큰 전송, 데이터 저장	DeFi, NFT 마켓플레이스

7. 상태(state)와 스토리지 관리

상태(State)란?

- 상태는 이더리움 블록체인에서 모든 계정과 스마트 컨트랙트의 현재 데이터를 나타냄
- 상태는 블록체인에서 글로벌 상태(Global State)로 관리되며, 블록이 추가될 때마다 업데이트됨

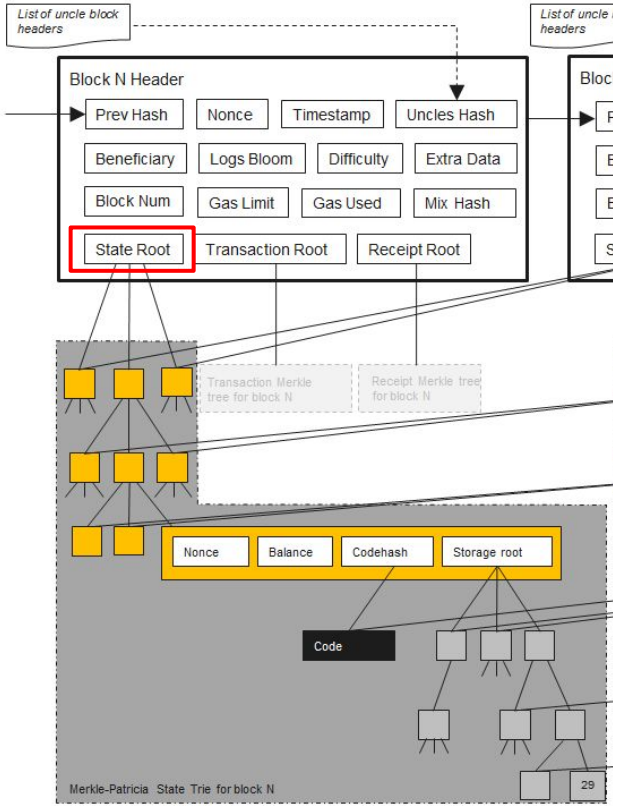


7. 상태(state)와 스토리지 관리

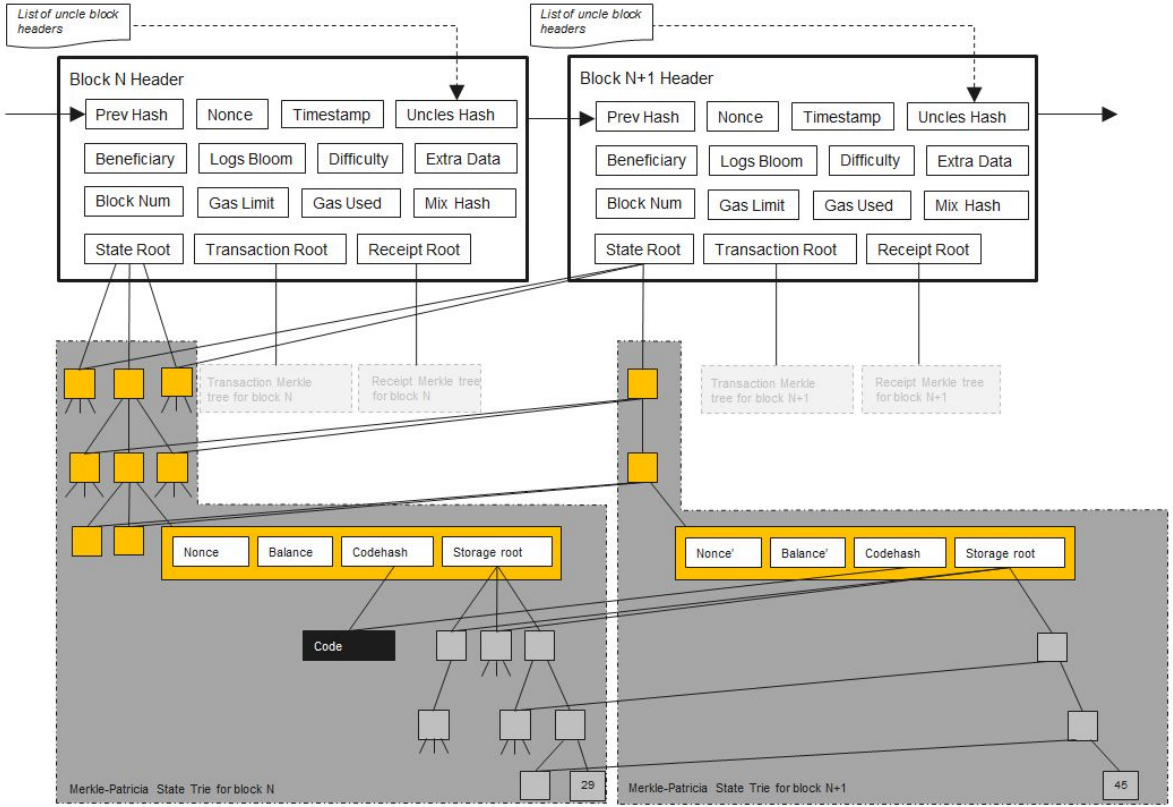
글로벌 상태 트리(Global State Trie)

- 이더리움의 상태는 **Merkle Patricia Trie**라는 트리 데이터 구조를 사용해 저장 됨
  - Merkle Patricia Trie는 **트리 구조**와 **해시 알고리즘을 결합**한 데이터 구조

- **구조**
  - 각 계정(Account)은 트리의 노드로 표현됨
  - 트랜잭션이 발생하면 해당 계정의 상태가 변경되고, 트리 전체가 업데이트 됨
  - 최상단 노드의 해시값은 블록 헤더의 **State Root**에 저장됨



7. 상태(state)와 스토리지 관리



7. 상태(state)와 스토리지 관리

계정(Account)에 대한 상태 저장

이더리움 계정은 두 가지 종류로 구분되며, 각각 상태 데이터를 다르게 저장함

1. **외부 소유 계정 (Externally Owned Account, EOA)**
   - 상태 데이터 :
     i. 잔액(Balance, ETH 보유량)
     ii. Nonce(중복 트랜잭션 방지를 위한 번호)
   - 일반 사용자가 소유하는 계정

2. **컨트랙트 계정 (Contract Account)**
   - 상태 데이터 :
     i. 스마트 컨트랙트 코드
     ii. 컨트랙트 실행에 필요한 변수 및 상태 정보
   - 스마트 컨트랙트를 기반으로 동작하는 계정

7. 상태(state)와 스토리지 관리

스토리지 (Storage)

스토리지란?

- 스토리지(Storage)는 스마트 컨트랙트가 데이터를 영구적으로 저장하는 공간
- 컨트랙트가 실행되면서 생성되는 변수나 데이터는 스토리지에 기록되며, 이는 트랜잭션과 상태 업데이트 과정에서 중요한 역할을 함

스토리지의 작동 방식

- 스마트 컨트랙트가 실행되면 데이터를 **키-값(Key - Value)** 쌍 형태로 스토리지에 저장
  ex. balances[msg.sender] = 100 : 특정 사용자의 잔액 데이터를 저장
- 스토리지 위치
  - 각 스마트 컨트랙트는 고유한 스토리지 공간을 가짐
  - 스토리지 데이터는 트리 구조로 관리되면, 블록체인에 영구적으로 저장

7. 상태(state)와 스토리지 관리

스토리지 (Storage)

스토리지 비용

- Gas 비용
  - 스마트 컨트랙트가 데이터를 저장할 때마다 Gas 가 소모됨
  - 스토리지 작업은 가장 많은 Gas 비용을 요구 :
    - 새로운 데이터를 저장: 20,000 Gas
    - 데이터를 읽기/업데이트 : 5,000 Gas
- 비용이 높은 이유
  - 블록체인에 영구적으로 데이터를 기록하는 작업이므로 많은 리소스 필요

7. 상태(state)와 스토리지 관리

스토리지 (Storage)

비용 효율적인 스토리지 설계

- **필요한 데이터만 저장**
  - 중요한 데이터만 블록체인에 저장하고, 임시 데이터는 로컬이나 오픈체인에 저장

- **데이터 압축**
  - 저장할 데이터를 압축하거나, 최소한의 공간을 사용하도록 설계

- **Off-Chain 저장**
  - 큰 데이터(이미지, 파일 등)는 IPFS 같은 분산 저장 네트워크를 활용
    * IPFS : 데이터를 분산 네트워크에 저장하는 시스템, 중앙 서버에 저장하지 않고 여러 컴퓨터에 분산해서 저장. (비유 : 기존방식 - 모든 책이 도서관에 있어 책 읽으려면 도서관 가야함 / IPFS 방식 - 여러 사람이 책의 사본을 나눠 가지고 있어, 주변 사람에게 바로 빌릴 수 있는)

7. 상태(state)와 스토리지 관리

항목	상태 (State)	스토리지 (Storage)
정의	모든 계정과 스마트 컨트랙트의 현재 데이터	스마트 컨트랙트가 영구적으로 데이터를 저장하는 공간
저장 위치	글로벌 상태 트리(Global State Trie)	각 컨트랙트의 개별 스토리지
업데이트	트랜잭션 실행 시 상태가 변경	컨트랙트 실행 중 데이터가 추가, 삭제, 수정됨
비용	상태 업데이트 비용은 상대적으로 저렴	스토리지 작업은 높은 Gas 비용 요구

8. 이더리움의 도전과제

확장성 문제

- 트랜잭션 처리 속도 제한
- 가스 비용 상승

샤딩(Sharding)과 레이어2 솔루션

- 이더리움의 확정성 문제 해결 방안
- 라이트닝 네트워크와 롤업(Rollups)

https://github.com/habit80/myBlockPanda

End