Merklux: 이더리움 플라즈마 어플리케이션 개발 라이브러리

임완섭(email@wanseob.com)

이더리움 플라즈마란?

한정된 처리용량

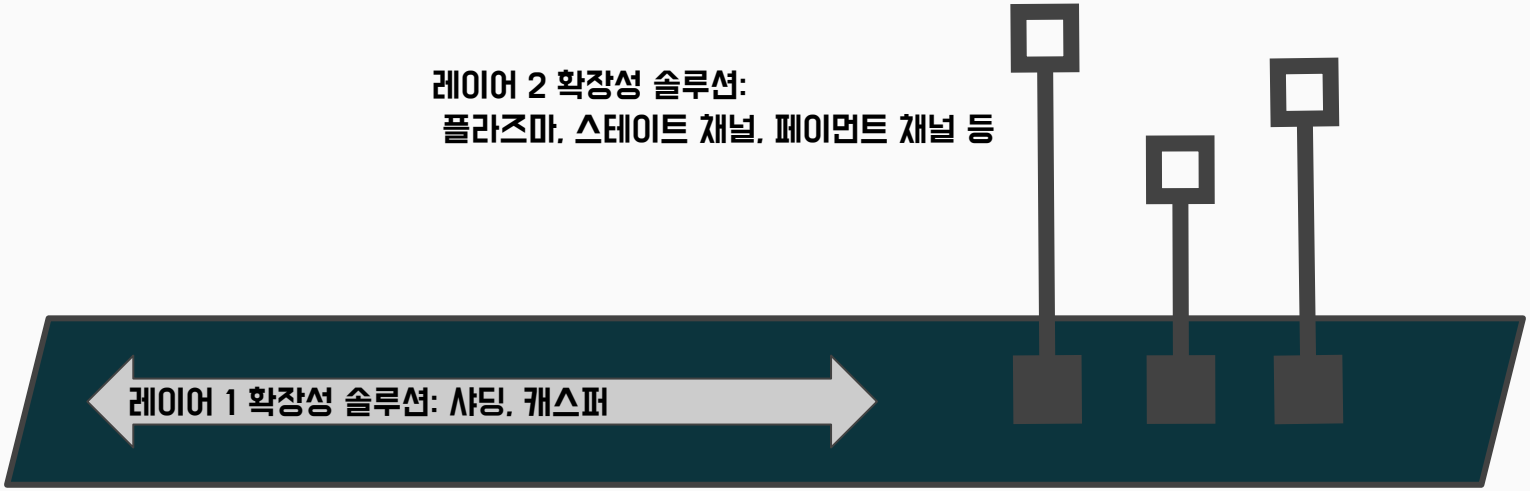
이더리움이 현재 보여주는 처리용량: 10 Transactions Per Second

비자카드가 1초에 처리하는 거래: 24000

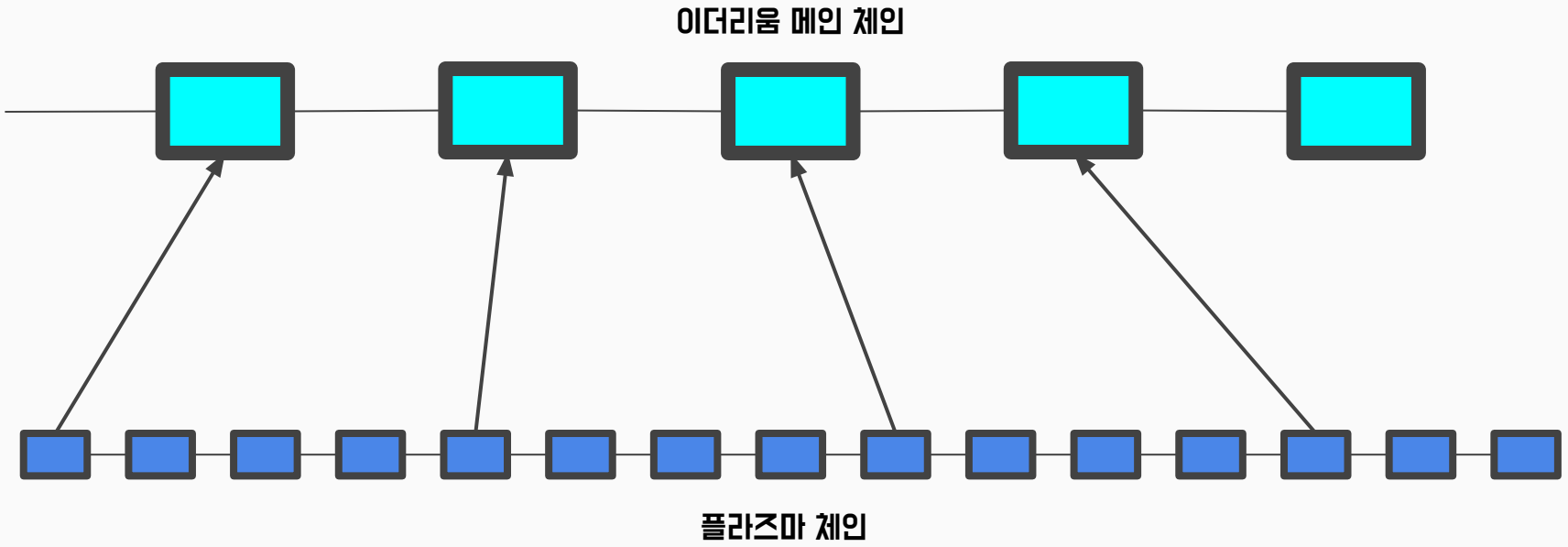
페이스북이 1초에 처리하는 요청: ~million

즉, 탈중앙된 블록체인의 특성상 dApp을 구동하기에 부족한 상태

이더리움의 레이어 2 솔루션



이더리움 플라즈마란?



플라즈마 구현의 현재

플라즈마 MVP:
비트코인 방식의 UTXO 아이디어를 차용하여 검증가능한 거래를 오프체인상에서 수행한 뒤 문제가 발생할 경우 챌린지 시스템을 통해 탈출게임을 진행하는 방식

거래에만 초점이 맞추어져 있으며
일반적인 상태변화를 사용하고자 하는
어플리케이션에 응용하기는 어려움

플라즈마 캐쉬:
플라즈마 MVP에 Non-fungibl [illegible] 없이도 안전하게 거래할 수 있는 시스템

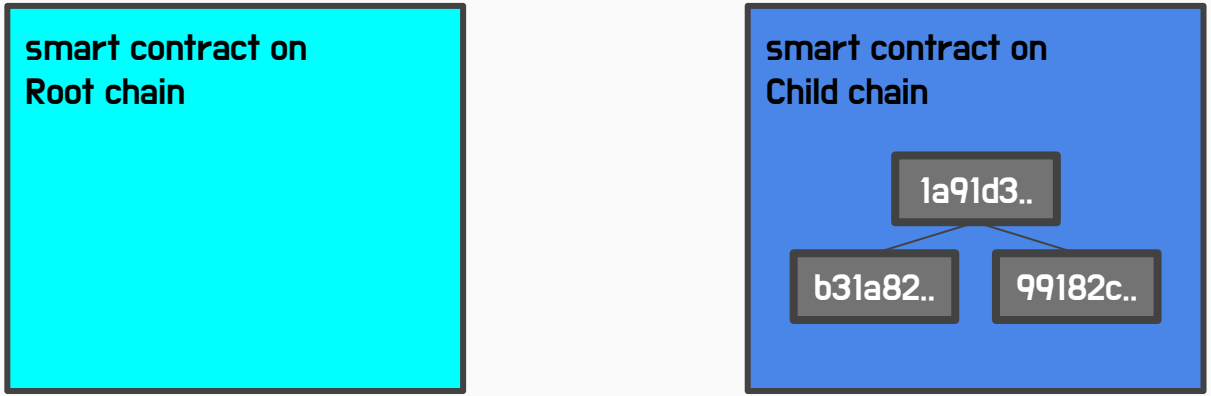
we need a
general state plasma
implementation!

General State 플라즈마 앱을 작성하기 위한 라이브러리

Merklux

Q1. 상태검증을 하는 방법?

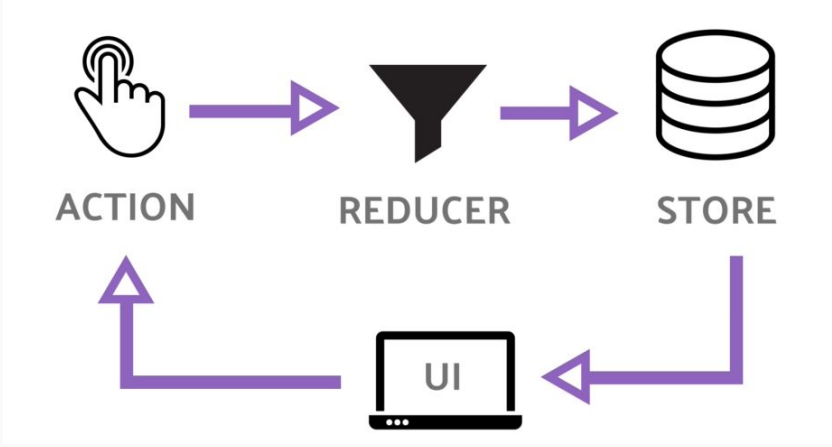
A: 머클 패트리시아 트리를 스마트 컨트랙에 구현할 경우,
스마트 컨트랙트를 사용하여 체인 간 상태변이를 검증할 수 있다.



Q2. 상태검증을 쉽게 하는 방법?

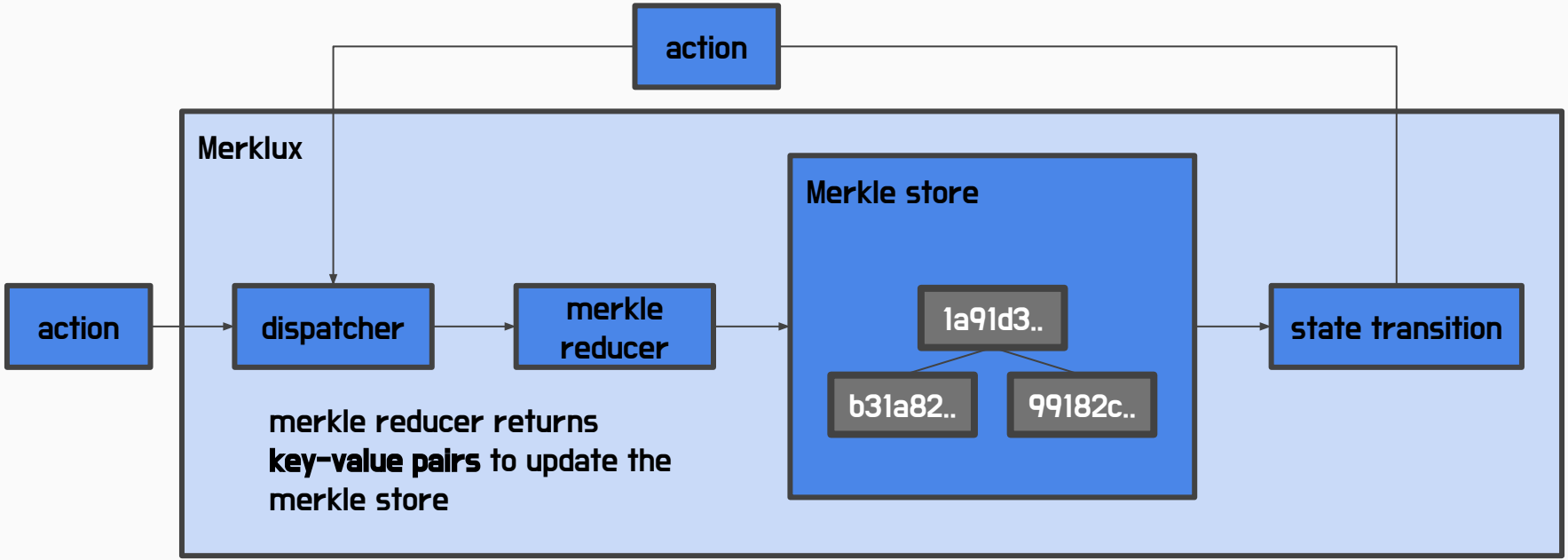
A: We can use Flux pattern like Redux

to manage the state transition of the plasma



Redux

Merklux


action
Merklux
action
dispatcher
merkle
reducer
Merkle store
1a91d3..
b31a82..
99182c..
state transition
merkle reducer returns
key-value pairs to update the
merkle store

```
contract Merklux is Secondary {

    using Block for Block.Object;
    using Store for Store.Object;
    using Transition for Transition.Object;
    using SafeMath for uint256;
    using ECDSA for bytes32;

    // Every action dispatches increment the height
    uint256 public height;
    bytes32[] storeList;
    bytes32[] reducerList;

    mapping(bytes32 => bytes32[]) references;

    Transition.Object[] transitionsOfCurrentBlock;

    mapping(bytes32 => Reducer) private reducers;
    mapping(bytes32 => Store.Object) private stores;

    bytes32[] blocks;
```

플라즈마 체인상의 데이터 중 머클럭스 스마트 컨트랙트만 루트 체인에서 믿을 수 있는 데이터이다.

*머클럭스만 루트 체인에서 검증 가능하기 때문

```
library Store {
    struct Object {
        string name;
        string[] actions;
        mapping(string => bytes32) allowedReducers;
        MerkluxTree tree;
    }

    function initialized(Object memory store) internal pure returns (bool) {
        return (store.tree != address(0));
    }

    function getStoreHash(Object storage store) internal view returns (bytes
        bytes32[] memory reducers = new bytes32[](store.actions.length);
        for (uint i = 0; i < store.actions.length; i ++) {
            reducers[i] = store.allowedReducers[store.actions[i]];
        }
//          keccak256(abi.encodePacked(store.actions));
        return keccak256(abi.encodePacked(
                store.name,
//                store.actions, //FIXME  nested dynamic array error
                reducers,
                store.tree.getRootHash()
            )
        );
    }
```

머클럭스는 키-값을 머클트리에 저장하는 **머클 스토어**를 사용하여 플라즈마의 모든 상태를 관리한다.

그리고 머클럭스 스마트 컨트랙트만이 해당 머클 스토어를 업데이트하는 권한을 지닌다

```
* @param _store The name of the store to update
* @param _action The name of the action
* @param _data RLP encoded data set
*/
function dispatch(string _store, string _action, bytes _data) external returns
    bytes32 _storeKey = keccak256(abi.encodePacked(_store));
    Store.Object storage store = stores[_storeKey];
    // stores should be initialized
    require(store.initialized());

    // It should have a reducer to handle the _action
    bytes32 reducerKey = store.allowedReducers[_action];
    require(reducerKey != bytes32(0));

    // The reducer also should exist
    require(reducers[reducerKey] != address(0));
    bytes memory key;
    bytes memory value;
    bytes32[] memory referredKeys;
    (key, value, referredKeys) = reducers[reducerKey].reduce(store.tree, msg.se
    store.tree.insert(key, value);

    // set references
    _addReferences(_storeKey, referredKeys);

    // record transition
    _dispatchTransition(
        _store,
```

머클스토어는 플럭스 패턴에 따라 액션을 **디스패치**해야만 업데이트 가능하며.

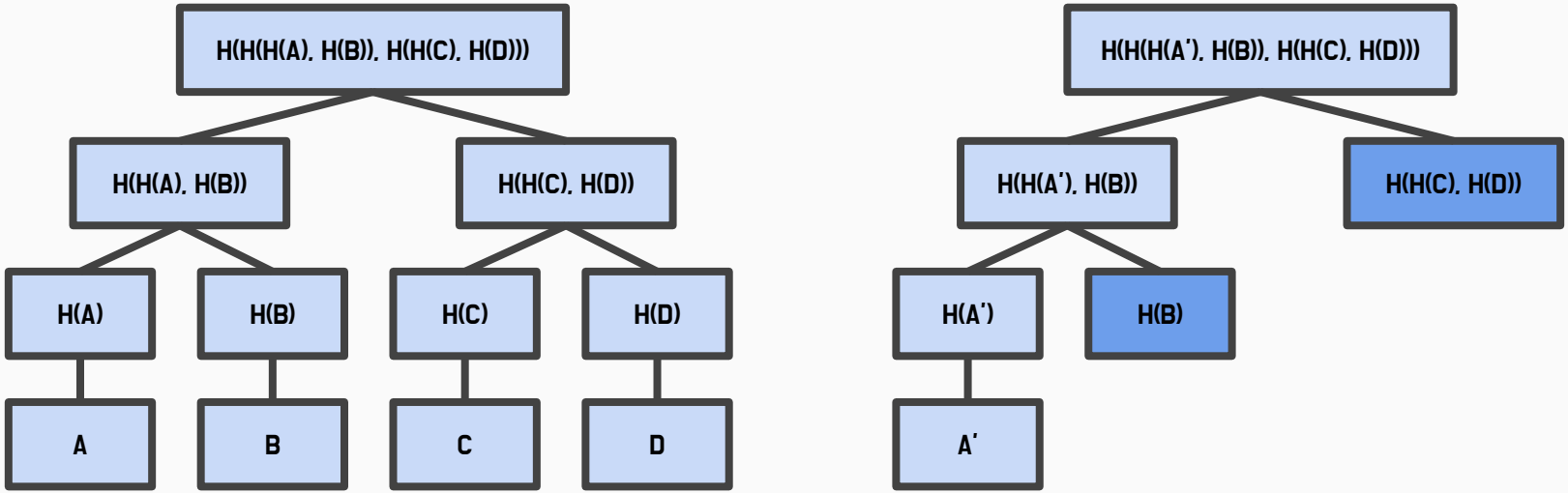
액션을 디스패치했을 경우 머클럭스 스마트 컨트랙트가 **리듀서**로부터 어떤 키-값을 업데이트해야할지 값을 받아온 뒤.
머클스토어를 업데이트 하고. 플라즈마 블록 높이를 1 증가시킨다.

플라즈마 앱 만들기:

머클럭스를 세팅하고
리듀서를 작성하면 끝

Partial Merkle Tree
경량 상태검증을 위한 부분적인 머클 트리

Partial Merkle Tree



Data Availability Problem이란?

Casper FFG

Plasma Plant:
머클럭스 앱에 PoS 플라즈마를 통해 Security 추가하기

플라즈마 블록의 생성

1. 모두가 **지분을 예탁**하는 것을 통해 플라즈마에 노드로 참가할 수 있음. 지분 예탁량은 공격비용 및 해당 플라즈마 체인의 TPS와 연관되어 있음.
2. 모든 에폭(epoch)마다 **블록 제안자들이 노드 풀에서 슈도랜덤**하게 선출
3. 각각의 블록 제안자들은 새로 생성되는 **블록마다 각기 다른 블록 생성 우선순위**를 가진다
4. 블록 제안자들은 각 블록에 대해 가지고 있는 고유의 **우선순위에 지수적으로 비례하는 난이도를 지닌 작업증명**을 할 경우, 블록을 제안할 수 있다.
   (PEPoW, Priority Exponential Proof of Work)

플라즈마 블록의 검증 및 1차 완결

1. 에폭(epoch)마다 블록 검**증인들이 노드풀에서 슈도랜덤(pseudo-random)하게 선택**된다.
2. PEPoW를 위한 캐스퍼 FFG는 블록검증인들이 서명을 진행할 때, **제 1 우선순위 블록 제안자가 생성한 블록에 서명했을 경우 최대의 보상**을 받는다.
   (PEPoW를 위한 캐스퍼FFG로 인해, 과도한 해쉬파워를 사용하는 것에 대한 보상이 사라진다. 왜냐면 우선순위가 낮은 블록 제안자가 새치기를 해도 검증인에게 선택받지 못할 가능성이 크기 때문에. 따라서 자연스러운 지연함수로 동작할 수 있게 된다)

고소 시스템

1. **에폭마다 제출되는 스냅샷에 문제**가 있을 경우, 참여 노드는 이 **스냅샷을 고소**할 수 있다
2. 스냅샷이 고소당했을 경우, 스냅샷 제출자는 이 사건을 **변호**해야 한다.
3. **사건을 변호하기** 위해, 스냅샷 제출자는 이전 스냅샷으로부터 제출된 스냅샷의 상태 변이가 정당했음을 증명하기 위해 중간에 참조된 머클 트리의 노드 값들과 리듀서 값들, 그리고 트랜잭션 히스토리를 루트체인에 제출한다.
4. 효율적인 데이터 제출을 위해 제출을 같이 할 **변호인단**을 추가로 선임할 수 있다.
5. 피고인이 사건 변호에 **실패**했을 경우, 관련된 **검증인 및 스냅샷 제출자는 예탁금을 몰수**당한다
6. 피고인이 방어에 성공했을 경우에는 **고소인이 변호 비용을 물어주고 예탁금을 몰수**당한다.

Q&A