BOOL NETWORK

以太坊桥接

杭州加密矩阵科技有限公司

HANGZHOU ABMATRIX TECHNOLOGY CO., LTD.

初心

- BOOL谐音为不二网络，不二在佛教中，意思为无彼此之别，万物生而平等，出自《佛学大辞典》“一实之理，如如平等，而无彼此之别，谓之不二”。

- 同时BOOL在计算机中表示要么是真要么是假的布尔变量；即表示布尔网络还原了中本聪的区块链初衷。做到真正的公平，公开，公正，完全透明去中心化。

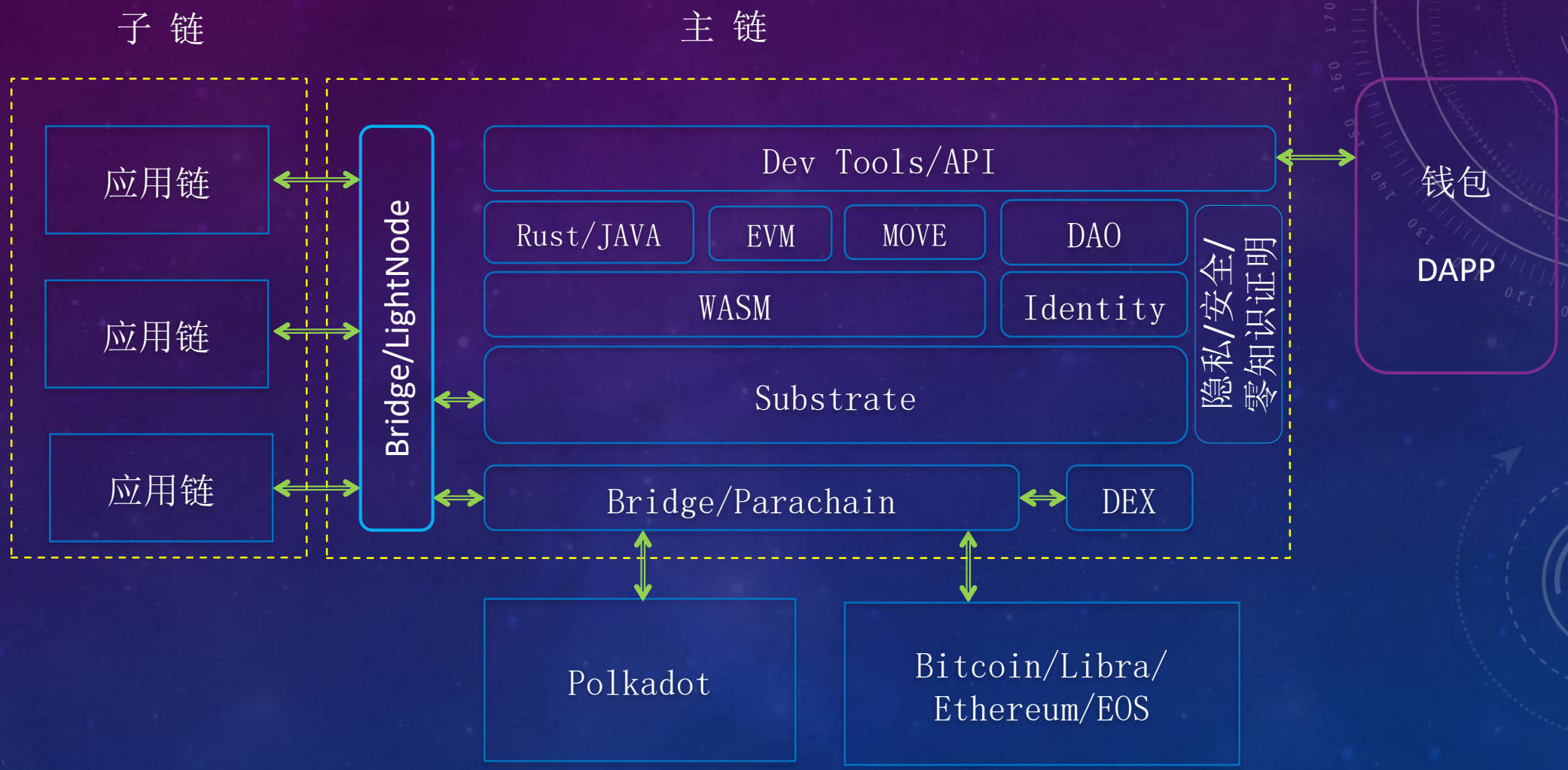
目标

用技术构建和连接信任

一个人人为我，我为人人的可信网络联盟

- 布尔网络是基于Substrate技术打造的区块链中台服务，为客户提供一站式多链及跨链技术支撑，适合多个应用层协议（如DEX、稳定币、节点自动部署等）衔接的商业场景，使得区块链前台的一线业务会更敏捷，更快速适应瞬息万变的市场。

技术架构


子 链
主 链
应用链
应用链
应用链
Bridge/LightNode
Dev Tools/API
Rust/JAVA
EVM
MOVE
DAO
WASM
Identity
Substrate
隐私/安全/
零知识证明
Bridge/Parachain
DEX
钱包
DAPP
Polkadot
Bitcoin/Libra/
Ethereum/EOS

功能架构


电商
金融
医疗
物联网
……
服务层
接口服务
SDK
API
ABI
钱包
通证经济
Bancor协议
资产协议
稳定币协议
交易协议
数据上链
数据存证
产品溯源
供应链
数据共享
可信分析
报 表
OLAP
数据挖掘
AI
中间件
跨链服务
监控服务
应用开发平台
智能合约
基础平台
公有链
联盟链
私有链
布尔网络

以太坊跨链桥接

转接桥介绍

链接两条异构或者同构链，支持链之间数据的可信互换。

桥实现难点

- **安全性**
- **去中心化程度**
- **响应性能**
- **数据上链费用**
- **是否对用户操作友好**
- **链价值兑换率**

转接桥介绍

现有的桥

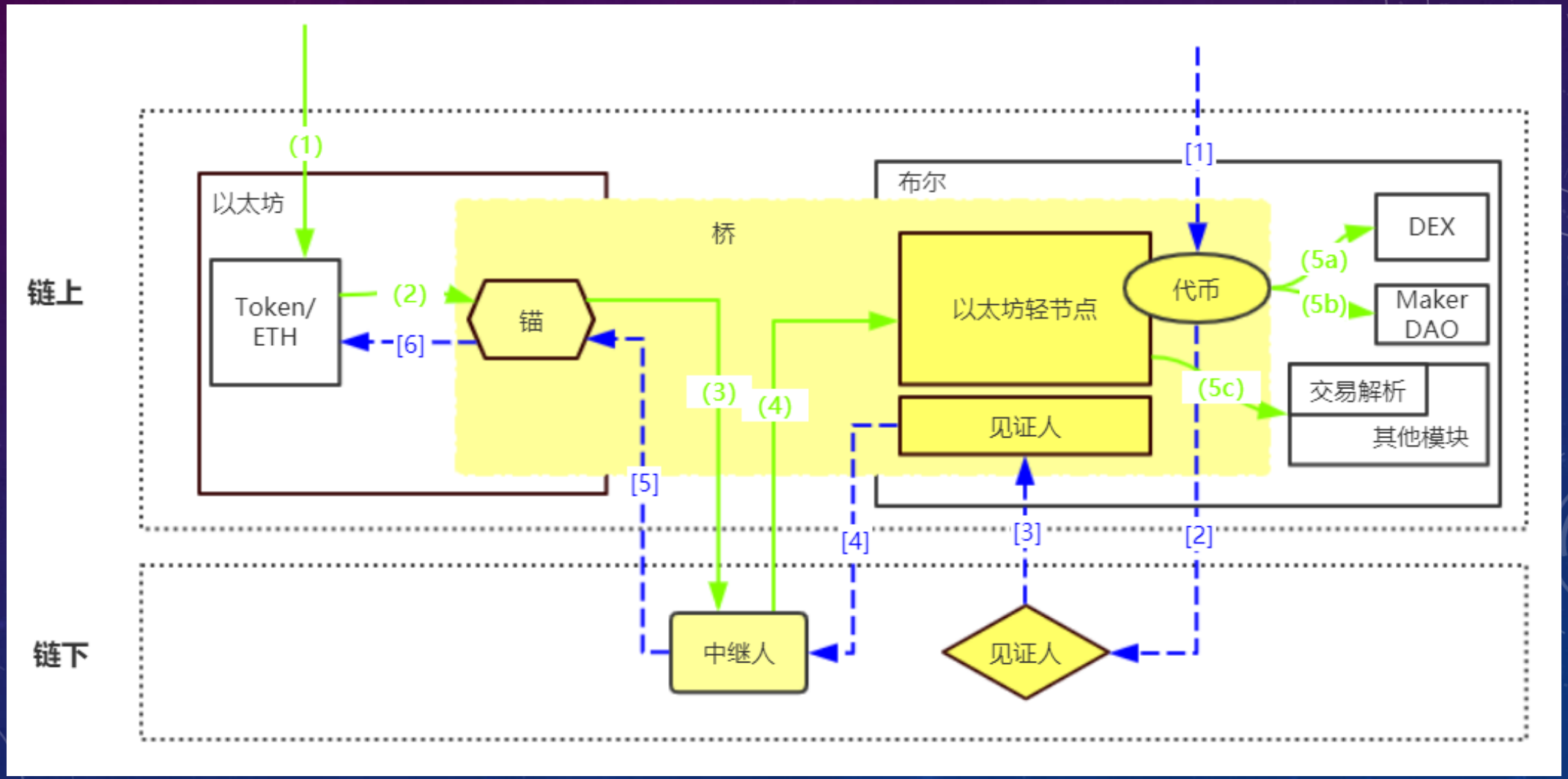
01 BTC-Relay： 单向桥，BTC-ETH，以太坊智能合约实现比特币节点。

02 Waterloo （kyber Network）：双向桥，ETH-EOS/EOS-ETH，互为轻节点。

03 Chainx： 双向桥，BTC-PCX/PCX-BTC，实现比特币轻节点，多签返回比特币。 去中心化交易所

04 BOOL： 双向桥，ETH-BOOL/BOOL-ETH，实现以太坊轻节点，多签返回以太坊。 去中心化交易所

转接桥架构


链上
链下
以太坊
Token/
ETH
桥
锚
布尔
以太坊轻节点
代币
见证人
DEX
Maker
DAO
交易解析
其他模块
中继人
见证人
(1)
(2)
[6]
(3)
(4)
[5]
[1]
(5a)
(5b)
(5c)
[4]
[3]
[2]

转接桥实现

介绍Substrate

- 从较高的层面来看，Substrate 是一个可以创建数字货币和其他去中心化系统的框架。

- Substrate提供了共识算法，P2P，存储等区块链基础组件。开发者只要在运行时层编写业务逻辑，就能完成一条链。

- Substrate使用大量宏来更容易编写运行时模块：
  - construct_runtime!
  - decl_module!
  - decl_storage!
  - decl_event!
  - 等等...

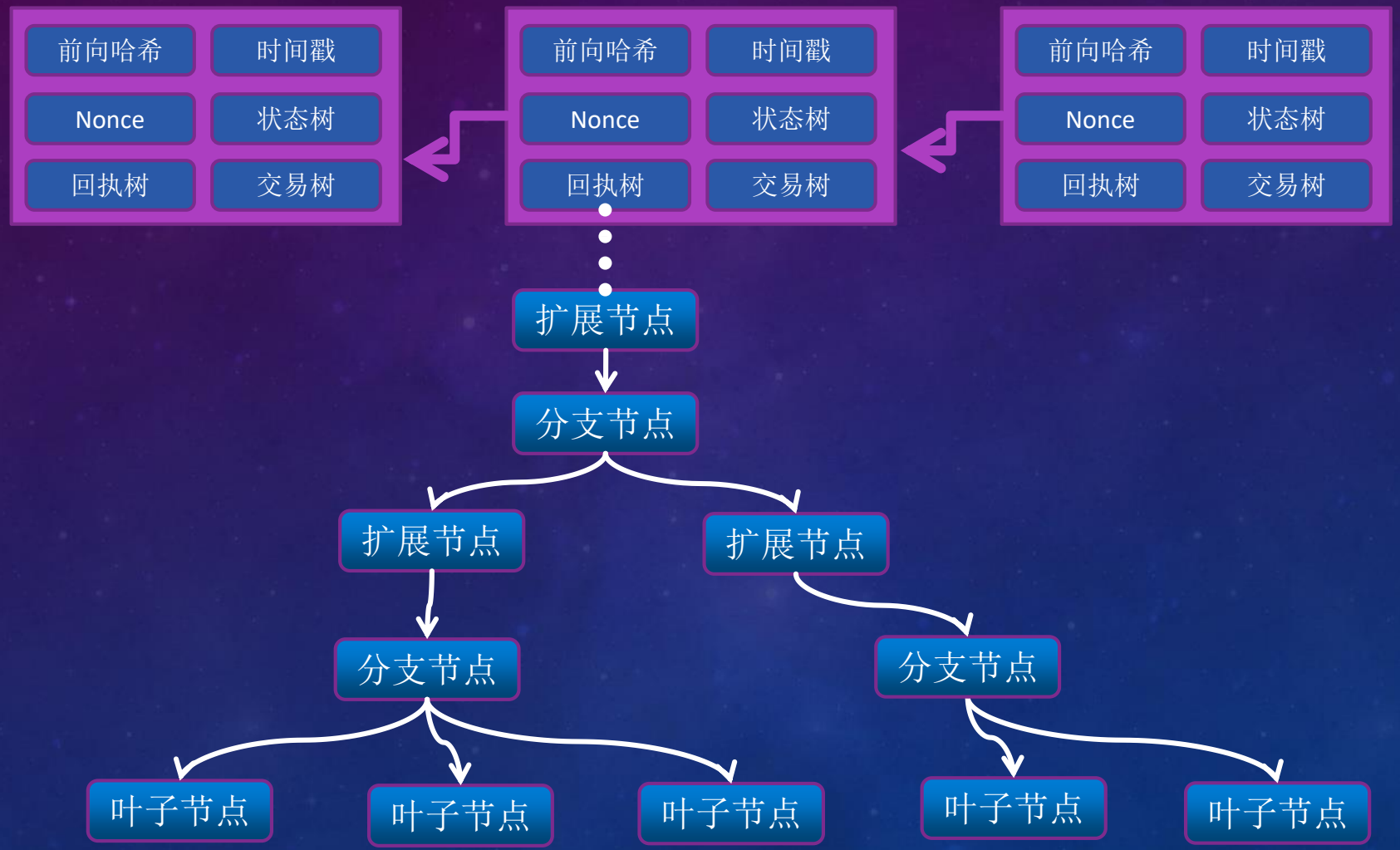
转接桥实现

ETH -> BOOL

- 在runtime层实现以太坊MPT树验证、以太坊区块头、交易以及回执解析基础上，完成验证逻辑。具体验证步骤如下：
  - 在BOOL链的创世块中初始化以太坊N高度区块头。
  - 中继人监听以太坊高度变化，提交以太坊区块头到BOOL链，所有节点在链上验证区块头合法性，例如前向哈希、时间戳、工作量证明。
  - 用户/中继人从以太坊上获取交易证明，该证明包括（i）该交易的在块中的索引。（ii）该交易所在的区块头哈希。（iii）该交易回执的证明路径。递交到BOOL链。

转接桥实现

回顾以太坊存储



转接桥实现

以太坊轻节点

转接桥实现 CODING：结构定义

```rust
/// The header of ethereum
#[derive(Clone, PartialEq, Eq, Encode, Decode)]
#[cfg_attr(feature = "std", derive(Debug))]
pub struct Header {
    /// Parent hash.
    pub parent_hash: H256,
    /// Block timestamp.
    pub timestamp: u64,
    /// Block number.
    pub number: BlockNumber,
    /// Block author.
    pub author: Address,
    /// Transactions root.
    pub transactions_root: H256,
    /// Block uncles hash.
    pub uncles_hash: H256,
    /// Block extra data.
    pub extra_data: Bytes,
    /// State root.
    pub state_root: H256,
    /// Block receipts root.
    pub receipts_root: H256,
    /// Block bloom.
    pub log_bloom: Bloom,
    /// Gas used for contracts execution.
    pub gas_used: U256,
    /// Block gas limit.
    pub gas_limit: U256,
    /// Block difficulty.
    pub difficulty: U256,
    /// Vector of post-RLP-encoded fields.
    pub seal: Vec<Bytes>,
    /// Memoized hash of that header and the seal.
    pub hash: Option<H256>,
}
```

```rust
/// Information describing execution of a transaction.
#[derive(Clone, PartialEq, Eq, Encode, Decode)]
#[cfg_attr(feature = "std", derive(Debug))]
pub struct Receipt {
    /// The total gas used in the block following execution of the transaction.
    pub gas_used: U256,
    /// The OR-wide combination of all logs' blooms for this transaction.
    pub log_bloom: Bloom,
    /// The logs stemming from this transaction.
    pub logs: Vec<LogEntry>,
    /// Transaction outcome.
    pub outcome: TransactionOutcome,
}
```

RLP

```rust
#[derive(Clone)]
#[cfg_attr(feature = "std", derive(Debug, PartialEq))]
pub struct Proof {
    pub nodes: Vec<Vec<u8>>
}
```

Keccak

RLP

```rust
#[derive(PartialEq, Eq, Clone, Encode, Decode, Default)]
#[cfg_attr(feature = "std", derive(Debug))]
pub struct ActionRecord {
    /// The index of action in proof trie.
    pub index: u64,
    /// Proof of action
    pub proof: Vec<u8>,
    /// Hash of block that include the action
    pub header_hash: H256,
}
```

转接桥实现 CODING： 模块定义

以太坊桥存储定义:

```
decl_storage! {
    trait Store for Module<T: Trait> as Bridge {
        /// Ethereum information
        pub BeginHeader get(begin_header): Option<EthHeader>;
        /// The best header in main chain
        pub BestHeader get(best_header): BestHeaderT;
        /// The header
        pub HeaderOf get(header_of) : map H256 => Option<EthHeader>;
        /// Storage the best chain header by number.
        pub BestHashOf get(best_hash_of): map u64 => Option<H256>;
        /// Storage hash at number.
        pub HashsOf get(hashs_of): map u64 => Vec<H256>;
        /// The number of block of delay for verify transaction.
        pub LazyNumber get(lazy_number): Option<u64>;
        pub ActionOf get(action_of): map T::Hash => Option<ActionRecord>;
        pub HeaderForIndex get(header_for_index): map H256 => Vec<(u64, T::Hash)>;
    }

    add_extra_genesis {
            config(header): Option<Vec<u8>>;
            config(number): u64;
            build(|config| {
                if let Some(h) = &config.header {
                    let header: EthHeader = rlp::decode(&h).expect("can't deserialize the header");
                    BeginNumber::put(header.number);

                    Module::<T>::genesis_header(header);
                } else {
                    BeginNumber::put(config.number);
                }
            });
    }
}
```

转接桥实现

CODING： 模块定义

以太坊桥操作接口定义:

```
decl_module! {
    pub struct Module<T: Trait> for enum Call where origin: T::Origin {
        fn deposit_event() = default;

        /// Submit a ethereum header of rlp coding
        fn submit_header(origin, header_rlp: Vec<u8>) -> Result {
            Ok(())
        }

        /// Submit a ethereum action of rlp coding
        fn submit_action(origin, rlp_proof: Vec<u8>) -> Result {
            Ok(())
        }
    }
}
```

以太坊桥事件定义:

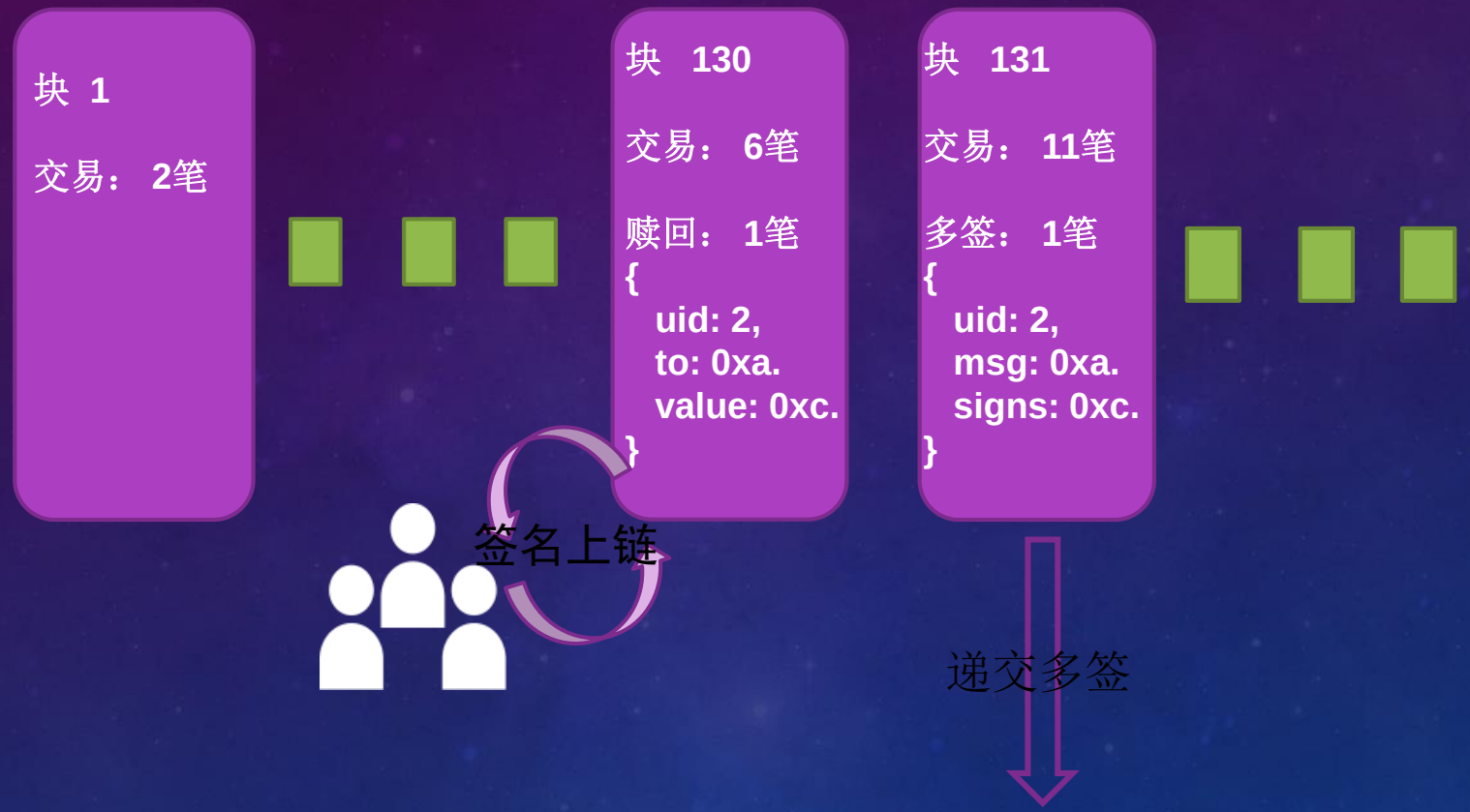
```
decl_event!(
    pub enum Event<T> where
      <T as system::Trait>::AccountId
     ,<T as system::Trait>::Hash
    {
        HeaderPass(AccountId, Vec<H256>),
        ActionPass(AccountId, Vec<Hash>)
    }
);
```

转接桥实现 BOOL -> ETH

- 在Runtime层实现多签见证人机制，作用于EVM智能合约资金释放。具体步骤如下：
  - 在BOOL链的创世块中初始化见证人集合。
  - 见证人节点监听BOOL链赎回事件，并对其签名后递交到BOOL链。
  - 用户/中继人从BOOL链上获取赎回多签，提交到以太坊。 BOOL链的见证人集合与以太坊中设置的多签集合是等价的。

转接桥实现

锚合约与见证人



转接桥实现 CODING： 模块定义

见证人存储定义:

```
decl_storage! {
    trait Store for Module<T: Trait> as Witness {
        /// Witness collection
        pub Authors get(authors) config(): Vec<T::AccountId>;
        /// Collector for signatures
        pub SignatureRecordOf  get(signature_record_of) : map T::Hash => Vec<(T::AccountId, Vec<u8>)>;
        /// Collector for message
        pub MessageRecordOf get(message_record_of) : map T::Hash => (Vec<u8>, u32);
        /// Minimum number of signatures supported
        pub MinimumAuthor get(minimum_author) config(): u32;
    }
}
```

见证人操作接口定义:

```
decl_module! {
    pub struct Module<T: Trait> for enum Call where origin: T::Origin {
        fn deposit_event() = default;

        fn collect_signature(origin, message: Vec<u8>, signature: Vec<u8>) -> Result {
            Ok(())
        }

        fn set_authors(public_keys: Vec<u8>, num: u64) -> Result {
            Ok(())
        }
    }
}
```

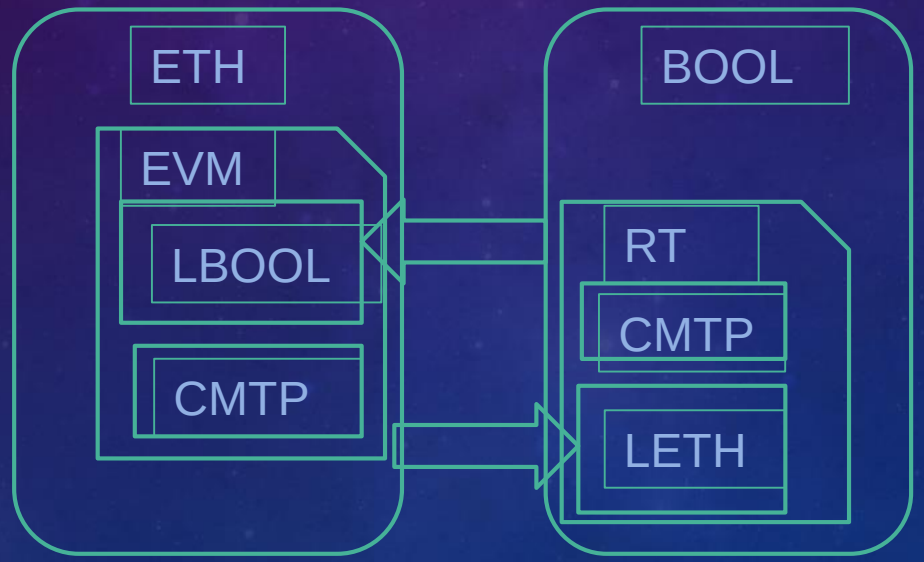
见证人事件定义:

```
decl_event!(
    pub enum Event<T> where <T as system::Trait>::Hash
    {
        FinalSignature(Hash, Vec<u8>, Vec<u8>),  //message hash, message, signatures
    }
);
```

转接桥实现

未来工作

- 跨链消息传输协议（CMTP）支持任意消息跨链
- 以太坊智能合约实现BOOL轻节点



谢谢