スマートコントラクトを利用した約定照合をやってみるテスト

２０１６年5月１１日

小岩井 航介

一応・・・

・このスライドは、完全に余暇で作ってます。
・所属する会社の資源は全く使ってません。
・所属する会社、その関連会社がブロックチェーンに関連して何かしているとしても、この資料の作成時点で、公開情報を除き、全く知りません。
・そもそも会社で開発のお仕事はしていません。

スマートコントラクトとは

専門家でも意見は分かれるらしい。 (*1)

- デジタルの形の約束事 (概念提唱者のNick Szaboの定義)
- 賢い契約 (直訳して、言葉をそのままに解釈)
- プログラム化された契約
- 自力執行権のある契約 (self-enforcing contract by Nicola Dorier)
- スマートコントラクト・プラットフォーム上で動くプログラムのこと
  - 狭義でEthereum上で動くプログラム、コントラクトのこと

(*1) http://qiita.com/hshimo/items/093f40b856ba2436fbba

スマートコントラクトとは

たぶん、こういうこと。（自分の勝手な理解）

・ブロックチェーン上で動くプログラムがある

・そのプログラムで、何らかの権利の「状態」（名義人、口座残高とか）が計算され、示される

・ 参加者全員が、そのプログラム上の「状態」と現実の権利が紐付くと合意している

できること

- たとえば、クラウドファンディング
  - 出資者が、あるプロジェクトに出資の申し込みをして、「コントラクト」にビットコインを送金する
  - 「コントラクト」は、一定期間内に十分な資金が集まったら、プロジェクトの実行者に全額を送金する
  - 「コントラクト」は、一定期間内に十分な資金が集まらなかったら、出資者に出資額を返金する
- こういうことが、いったん「コントラクト」を作ってしまえば、人の手を介在せずにできる。ブロックチェーン上で動くから、不正もないし、胴元（システムを運用する人）も必要ない。

証券業界で使えそうな
ユースケース

- 「ほふり」のやっていること（一例：超簡単に）
  - 「取引照合」→「債務引受」→「精算」
  - 取引照合：取引先同士が約定データを送信して、お互いの認識が合致することを確認）
  - 債務引受：取引照合が終わった約定について、ほふりがその取引を保証（この時点で取引相手は取引先からほふりに変わる：ここまで原則約定日当日中に完了）
  - 精算：同じ銘柄の売買を相殺（ネッティング）して、決済を実行

証券業界で使えそうな
ユースケース

- 「ほふり」のやっていること（一例：超簡単に）
  - 「取引照合」→「債務引受」→「精算」
  - 取引照合：取引先同士が約定データを送信して、お互いの認識が合致することを確認）
  - 債務引受：取引照合が終わった約定について、ほふりがその取引を保証（この時点で取引相手は取引先からほふりに変わる：ここまで原則約定日当日中に完了）
  - 精算：同じ銘柄の売買を相殺（ネッティング）して、決済を実行

ビットコイン建債券とかならできそう。

ということで今回は、
「取引照合」に絞って
やってみます。

取引照合

・ほふり（証券保管振替機構）のホームページを参照
（注：もちろん一般的に公開されている情報です）

・一番シンプルな、「二者間センタ・マッチング型」の照合を実装してみます。

参照：決済照合システム 概要説明 https://www.jasdec.com/download/finance/stp_riyousha1.pdf

取引照合

- 二者間センタ・マッチング型

  ① 双方からほふりに取引データを送信

  ② ほふりから双方にマッチしたことを通知

参照：決済照合システム 概要説明 https://www.jasdec.com/download/finance/stp_riyousha1.pdf

取引照合

- 照合条件（下線がキー）
  1. **約定照合キー**
  2. **機関投資家**
  3. **売り手**
  4. **買い手**
  5. **ファンドコード**
  6. **銘柄コード**
  7. **約定日**
  8. **決済日**
  9. **売買区分**
  10. **数量**
  11. **単価**
  12. **約定金額**
  13. **消費税**
  14. **国内手数料**
  15. **決済金額**

へぇ、「約定金額」って条件に入らないんだ。
→小数点の有効桁数等で、計算誤差があってもマッチするためと思われ。

参照：「決済照合システム」 システム概要 http://www.jasdec.com/download/topics/c.pdf

取引照合

- とりあえず今回は、下記の条件だけを対象とします。

1. **売り手**
2. **買い手**
3. **銘柄コード**
4. **約定日**
5. **決済日**
6. **数量**
7. **単価**
8. **約定金額**

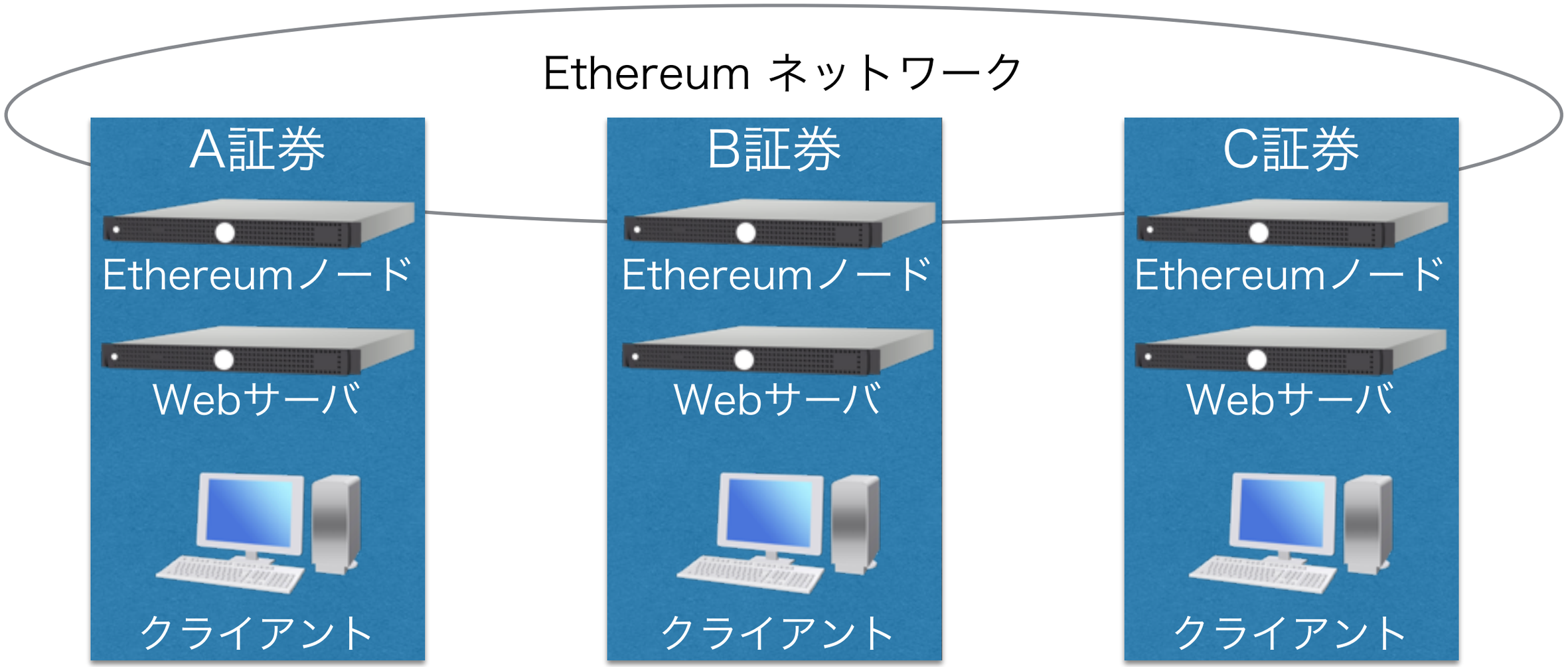
参照：「決済照合システム」 システム概要 http://www.jasdec.com/download/topics/c.pdf

実装

- 使ったプラットフォーム
  - Ethereum : ブロックチェーン
  - Solidity：Ethereum上のプログラム言語
  - Meteor：Ethereumと連携するWebフレームワーク
  - PHP : 実際はサーバサイドで連携する仕組みがmeteorに無かったので自前でEthereum (geth) とのインターフェースを実装

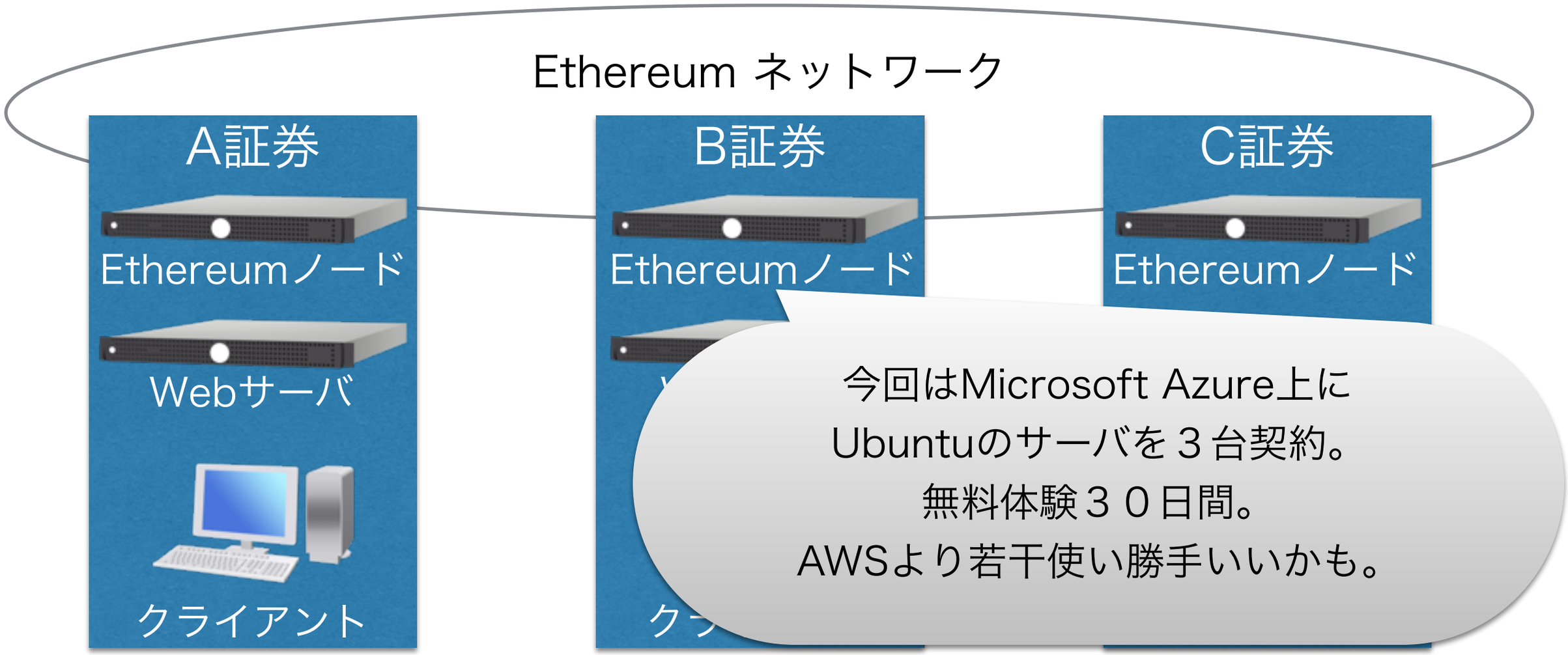
実装

- （２社じゃ意味ないので）３社の参加と仮定
- それぞれ、Ethereumのノードが１台ずつ参加



実装

- （２社じゃ意味ないので）３社の参加と仮定
- それぞれ、Ethereumのノードが１台ずつ参加



実装

- GitHubへのリンク
  - https://github.com/kkoiwai/SolidityTradeMatcher
- コントラクト自体のプログラムより、それを操作するGUIを作る方が大変
  - Solidityコントラクト：73行（コメント、空行含む）
  - PHP インターフェース：209行（同上）
  - meteorで作ったGUI（自分で書いた部分）：約200行

```
contract TradeMatching{


    // Trades to be matched
    struct Trade{
        address sender;
        bytes32 senderid;
        address seller;
        address buyer;
        bytes12 seccode;
        uint32 tradedate; //YYYYMMDD
        uint32 deliverydate; //YYYYMMDD
        uint quantity;
        uint32 price;
        int deliveryamount;
        uint matchedtrade;
    }

    // The storage to save all trades
    uint numTrades = 1; //start with 1 to make clear 0=null
    mapping(uint => Trade) trademap;

    // data structure to link trades by Date
    mapping(uint32 => mapping(uint => uint)) tradesByDate;
    mapping(uint32 => uint) tradesByDateCounter;

    // The function
    function addTrade(bytes32 senderid, address seller, address buyer,  bytes12 seccode, uint32 tradedate, uint32 deliverydate, uint q
        uint tradeID = numTrades++;
        trademap[tradeID] = Trade(msg.sender,senderid, seller,buyer,seccode,tradedate,deliverydate,quantity,price, deliveryamount,0);

        // store tradeID to tradesByDate
        uint count = tradesByDateCounter[tradedate];
        tradesByDate[tradedate][count] = tradeID;
        tradesByDateCounter[tradedate] = count + 1;

        /* here starts "matching" part! */

        // for the rest of the trade date's trades,
        uint matchingTradeID;

        while(count > 0){
            count--;
            matchingTradeID = tradesByDate[tradedate][count];
            Trade t = trademap[matchingTradeID];
            if(t.sender == msg.sender || t.matchedtrade != 0) continue;
            if(t.seller == seller && t.buyer == buyer && t.seccode == seccode && t.tradedate == tradedate && t.deliverydate == delivery
                trademap[matchingTradeID].matchedtrade = tradeID;
                trademap[tradeID].matchedtrade = matchingTradeID;
                break;
            }
        }
    }
```

```
26
27      // The function
28      function addTrade(bytes32 senderid, address seller, address buyer,  bytes12 seccode, uint32 tradedate, uint32 deliverydate, uint q
29          uint tradeID = numTrades++;
30          trademap[tradeID] = Trade(msg.sender,senderid, seller,buyer,seccode,tradedate,deliverydate,quantity,price, deliveryamount,0);
31
32          // store tradeID to tradesByDate
33          uint count = tradesByDateCounter[tradedate];
34          tradesByDate[tradedate][count] = tradeID;
35          tradesByDateCounter[tradedate] = count + 1;
36
37          /* here starts "matching" part! */
38
39          // for the rest of the trade date's trades,
40          uint matchingTradeID;
41
42          while(count > 0){
43              count--;
44              matchingTradeID = tradesByDate[tradedate][count];
45              Trade t = trademap[matchingTradeID];
46              if(t.sender == msg.sender || t.matchedtrade != 0) continue;
47              if(t.seller == seller && t.buyer == buyer && t.seccode == seccode && t.tradedate == tradedate && t.deliverydate == delivery
48                  trademap[matchingTradeID].matchedtrade = tradeID;
49                  trademap[tradeID].matchedtrade = matchingTradeID;
50                  break;
51              }
52          }
53      }
54
55      // getter
56      function getTrade(uint tradeID) constant returns (address sender, bytes32 senderid, address seller, address buyer,  bytes12 seccode
57          Trade t = trademap[tradeID];
58          sender = t.sender;
59          senderid = t.senderid;
60          seller = t.seller;
61          buyer = t.buyer;
62          seccode = t.seccode;
63          tradedate = t.tradedate;
64          deliverydate = t.deliverydate;
65          quantity = t.quantity;
66          price = t.price;
67          deliveryamount = t.deliveryamount;
68          matchedtrade = t.matchedtrade;
69      }
70
71
72      function getTradeIdByTradedate(uint32 tradedate,uint count) constant returns (uint tradeID){
73          tradeID = tradesByDate[tradedate][count];
74      }
75  }
76
77
```

デモ

実装

- 困ったこと（言語としての仕様・制約）
  - Date型がない！小数型（FloatとかDecimalとか）もない！
    → 構造体(struct)を定義できるけど、構造体を引数にも返値にもできない！
  - uint i = 0 のときに i --しちゃうと while(i>0)が無?限ループ
- まだ開発段階なので自分で手作りしないといけない部分が多い
  - でもEthereum内部はバイトコードなので単にコンパイラが進化すればもうすこし楽になりそう

実装

- 困ったこと(ブロックチェーンとしての仕様・制約)
  - トランザクションが発生する場合（ブロックチェーン上で状態が変化する場合）、返値(return value)が戻ってこない。つまり送信に成功したのか分からない、ブロックチェーン上の取引のユニークIDが分からない
  - 送信側で個別にユニークIDを振っておいて、getterで調べるしかない

考察

- 現実で使う前に考えないといけないこと
  - 内容チェック
    - もちろん「契約」なんだからどんな契約でも双方が合意すればいいんだけど、どこまでチェックすればよいか
      - あり得ない日付
        - 過去日とか、「１３月３２日」とか、債券の償還期限過ぎてるパターン
      - ありえない通貨
        - １００万ゼニーの債券とか
- 参加者全員に手の内見られちゃう。
  - 実装の手段はある。と思う。（秘密鍵公開鍵）
- ロジックを入れれば入れるほど複雑になって、実行速度も落ち、バグも混ざる
  - 「運用でカバー」：困ったら反対売買することにする？

考察

- 現実で使う前に考えないといけないこと
  - 処理スピード・計算量・計算コスト
    - 計算参加者が増えても計算量は増えない
    - パブリックネットワークですら、全体として古いスマホ１台分の計算力しかない？らしい？（要検証*
    - パブリックネットワークだと、この簡単なプログラムですら、今の相場で１取引あたり約２０００円かかる計算
    - 債券取引ならまだ現実味あるけど、株取引はどうだろう。。。

*) “Ethereum network, regardless of the number of nodes that forms it, is equal to a single smartphone from 1999.” https://dappsforbeginners.wordpress.com/tutorials/your-first-dapp/

考察

- 債務引受って可能なのかしら
  - 引き受けるための様々な条件をすべてプログラムとして記載するのは相当大変
  - 現実的には引受人（ほふり）の元で別で動くプログラムが取引内容を承認するような仕組みが必要そう
  - そもそも「権威ある第三者」が介在する時点で債務引受はブロックチェーン的ではない気がする

・・・などと、こういった感じで様々な知見を得ようとしているのが昨今の各社の取り組みと思われる（妄想）

以上、ご清聴

ありがとうございました。