

# BBc-1 design document

revision 1

for v1.0

21 May 2018

# 本資料について

- BBc-1 の設計思想についてまとめる
  - 対象のBBc-1はgithubに公開されたv1.0 (2018/5/1バージョン) である
    - <https://github.com/beyond-blockchain/bbc1>
  - BBc-1のWhitePaper、YellowPaper (Analysys)はgithubリポジトリのdocs/、および下記URLに公開されている
    - <https://beyond-blockchain.org/public/bbc1-design-paper.pdf>
    - <https://beyond-blockchain.org/public/bbc1-analysis.pdf>
  - 本資料の内容に起因するあらゆるトラブルには責任を負わない
- 作成日：2018/5/21
- 作成者：takeshi@quvox.net (t-kubo@zettant.com)

# Collaborators' github account

- kichinosukey
- oopth
- kzkwatanabe
- junkurihara
- imony
- ks91

# Change log

- 2018/5/21: 初版

# 目次

タイトル	ページ
設計思想とアーキテクチャ	6
用語の定義	12
システム構成	16
トランザクション	21

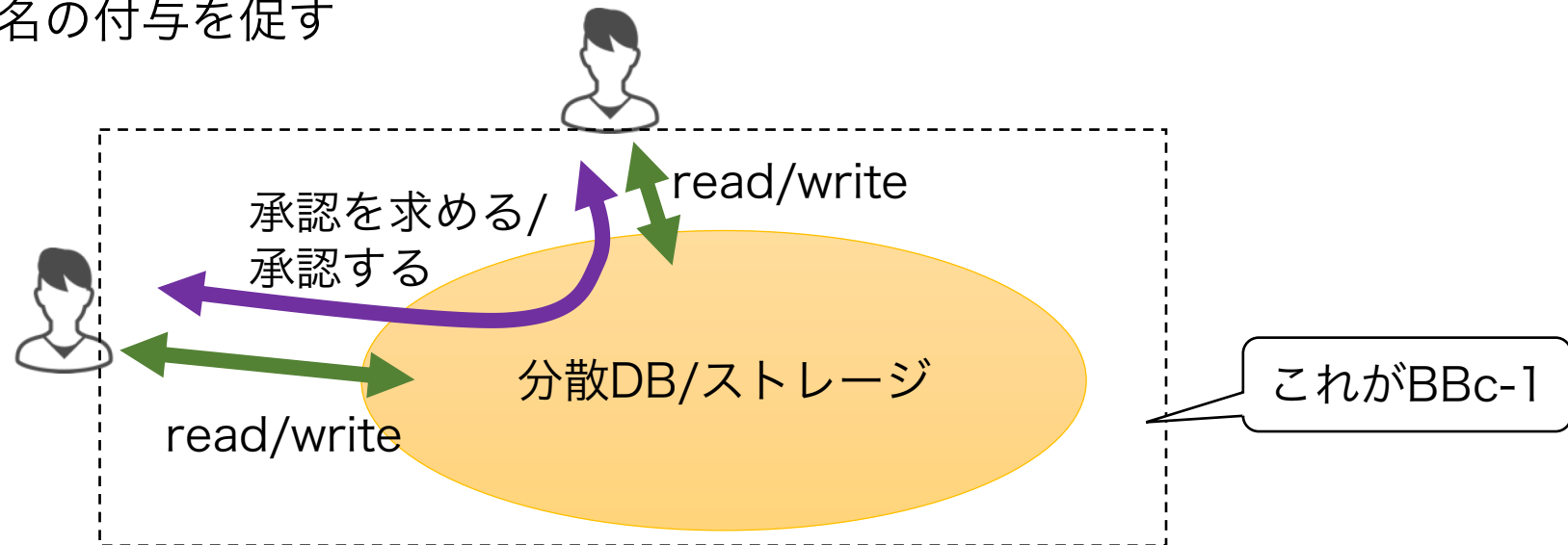
# 設計思想とアーキテクチャ

# BBc-1 とは

- BBc-1は既存のブロックチェーン技術およびそれを利用したプラットフォームが抱える下記のような課題を解決することを目的に開発された技術およびプラットフォームの総称である
  - ネイティブ通貨（暗号通貨）の市場価値暴落によるシステム機能不全リスク
  - ブロックのマイニングによるファイナリティの問題
  - 小規模な運営主体で全データを改ざんされた場合に改ざんの有無すら認識できない問題
  - コンセンサスアルゴリズムと実ビジネスにおける合意の概念の不整合
- <https://github.com/beyond-blockchain/bbc1> に公開されている実装は、pythonによるリファレンス実装である
  - 特に重要なのは、P.21のトランザクションのセクションに示したトランザクションデータの構成の仕方であり、この要件を満たせば、様々なバリエーションや言語による実装が可能である

# BBc-1は何ができるのか？

- 改ざんが困難な分散DB/ストレージ
  - アセット間の関連付けを可能にするデータ構造を用いた情報保存
  - デジタル署名に基づく検証
  - 改ざん検出・回復
- 書き込みの前に「他のentityに承認を得る/承認する」ためのメッセージング機能
  - デジタル署名の付与を促す





# BBc-1 の設計思想

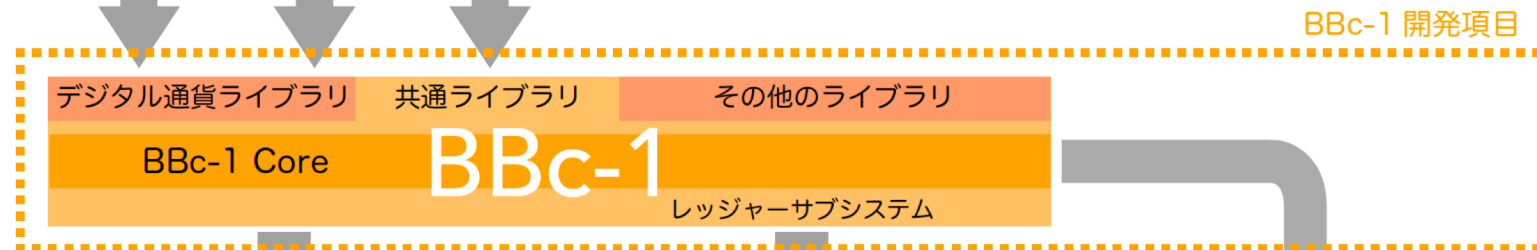
- 本当に大事なことにだけフォーカスする

情報が、たしかにそのentityによって登録され、改ざんされていないということを他entityが確認できるようにする

- シンプルで拡張性が高くインターネットにおけるTCP/IPのようなイメージ
- これさえできれば、真贋証明、価値移転、スマートコントラクトなど現在ブロックチェーン関連技術で注目されている全てのサービスが実現できる

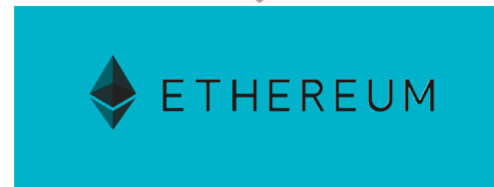
# BBc-1 アーキテクチャ

アプリケーション



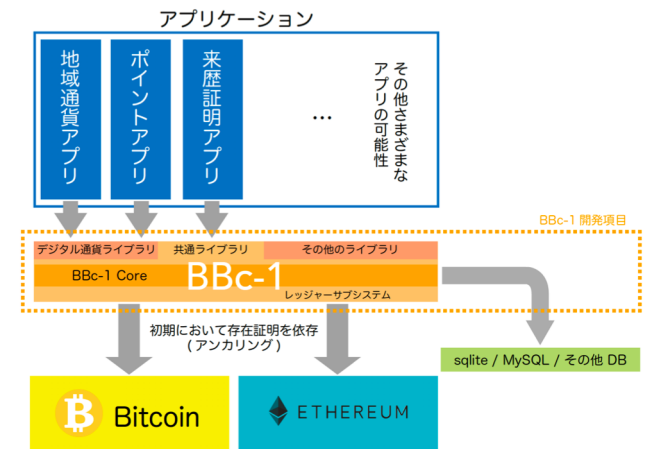
初期において存在証明を依存  
(アンカリング)

sqlite / MySQL / その他 DB



# BBc-1 アーキテクチャ

- 既存プラットフォームの利用
  - BBc-1 の認知度が低い段階
    - BBc-1 単体でも存在証明可能だが、その証拠を外部システム（EthereumやBitcoin）に求め、追認してもらえるようにする
    - 利用する外部システムは自由に選択、切り替え可能
  - BBc-1 が普及した時
    - 外部システムを切り離し、BBc-1 単体で動作させる
      - タイミングはシステム開発者が決めればよい（初期から切り離しても良い）
- 分散DB（レジャー）/ストレージ
  - ファイルの格納場所には、BBc-1 のコアシステム内だけでなく、外部ストレージも指定できる



# 用語の定義

# BBc-1における基本概念

- ユーザ (user)
  - BBc-1を利用するentity
  - userは、公開鍵のペアを自分自身で生成しておく必要がある
- アセット (asset)
  - userが保有するデジタル資産で、BBc-1にその存在が登録されたもの
  - assetに関するメタ情報も含まれる (所有者など)
- トランザクション (transaction)
  - asset本体および複数のasset間の関係性を記述した情報
- アセット種別 (asset\_group)
  - 複数種類のassetを区別するための識別子
    - 例：社内機密ファイル、地域通貨、土地、センサーデータ、etc,,,
- ドメイン
  - トランザクションやアセットの情報を共有する範囲
- 履歴交差
  - あるドメインで登録されたトランザクションの識別子を、別のドメインのトランザクションに埋め込むことで、外部ドメイン (利害関係がないことが望ましい) にトランザクションの存在を証明してもらう考え方

# BBc-1 システムの構成要素

- コアノード (core nodeまたは単にnode)
  - BBc-1のサービスを提供するコンピュータ
  - node同士がP2P overlay networkを構成し、連携してサービスを提供する
- ユーザ (user)
  - アセットの所有者、アプリケーションの利用者
- クライアント (client)
  - コアノードに接続するアプリケーション
- ドメイン (domain)
  - コアノードが構成する論理的なP2P overlay network
  - 同一domainの中でのみtransactionの情報やメッセージが共有、伝送される
- domain\_global\_0
  - domain\_id (次頁) が0 (256bitすべて0) のドメイン
  - 履歴交差情報を異なるドメイン間で交換するための特別なP2P overlay network

# BBc-1の識別子

名前	説明	ユニークネス
node_id	ノード識別子 (ドメインごとに作成される)	domainごとにID空間が定義され、その中でユニークでなければならない。 なお、transaction_idはトランザクションのダイジェストである（計算方法は別紙(How_BBc1_works_v1.0_ja)参照
user_id	ユーザentityの識別子 (entityごとに鍵ペアを持つ)	
asset_id	アセット識別子 (アセットのファイル名にもなる)	
transaction_id	トランザクション識別子	
asset_group_id	アセット種別の識別子	
domain_id	ドメイン識別子	グローバルにユニークでなければならない（ただし完全にプライベートだけで使うならその中でユニークであればよい）

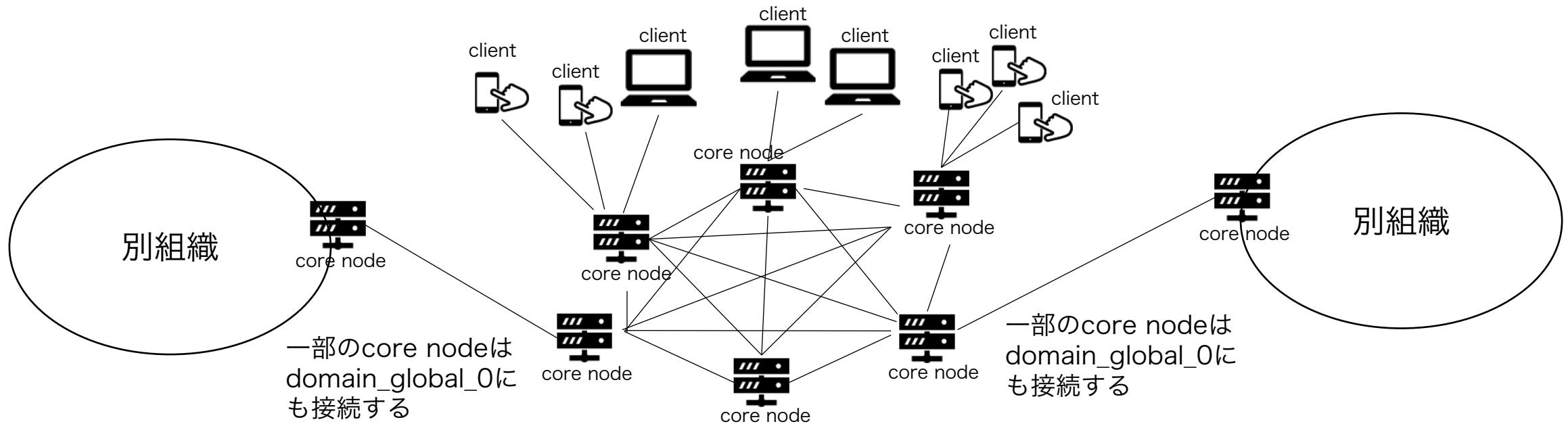
※ 識別子は全て256bitである (SHA256で発生させる)

# システム構成

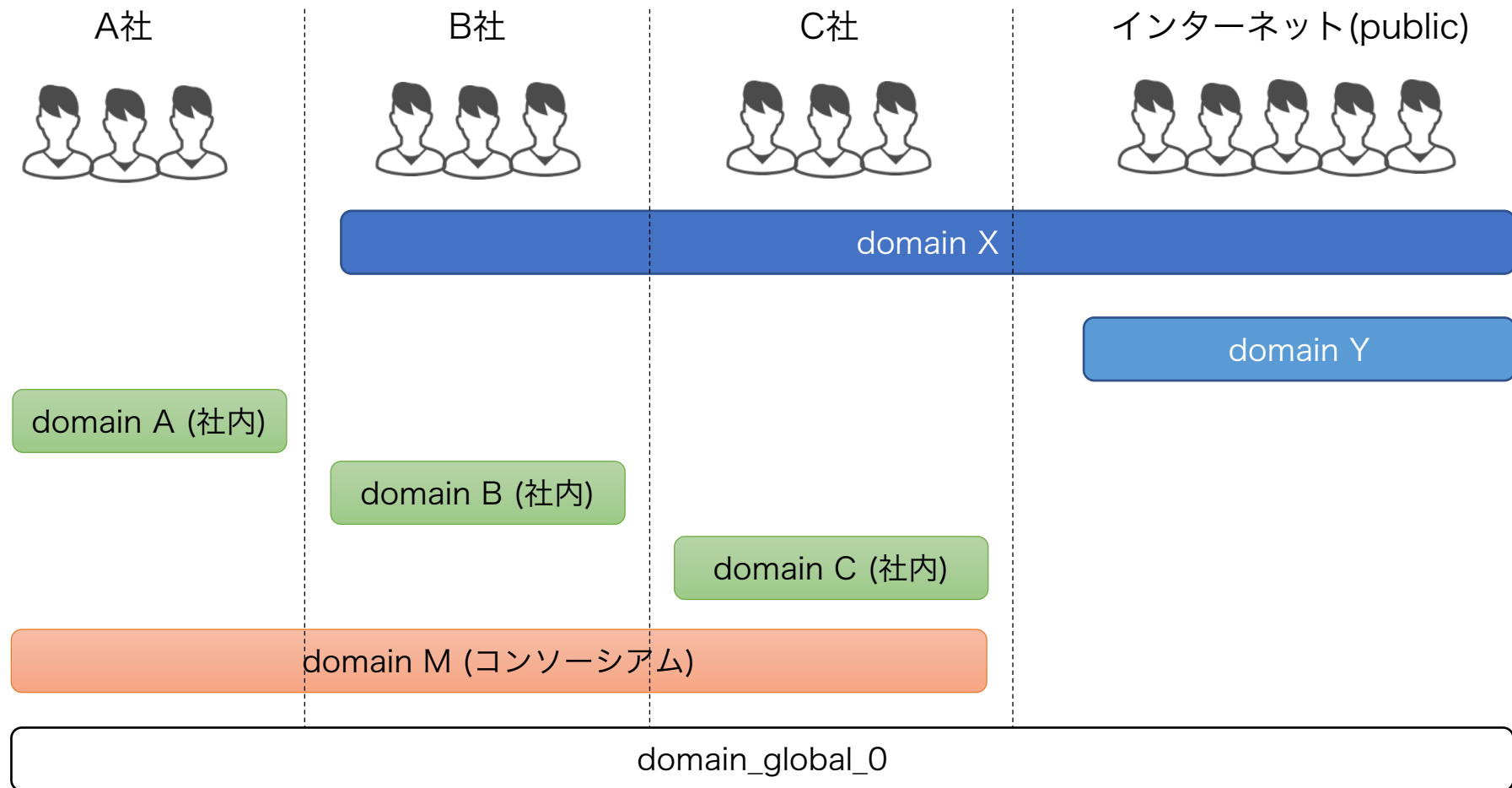


# 典型的なシステム構成

- core nodeは社内、コンソーシアム内、インターネット上に配置される
- clientは社内ネットワークやインターネットを通してcore nodeに接続する
- サービス開発・提供者がcore node上にdomainを作りアプリケーション提供する



# domainのイメージ

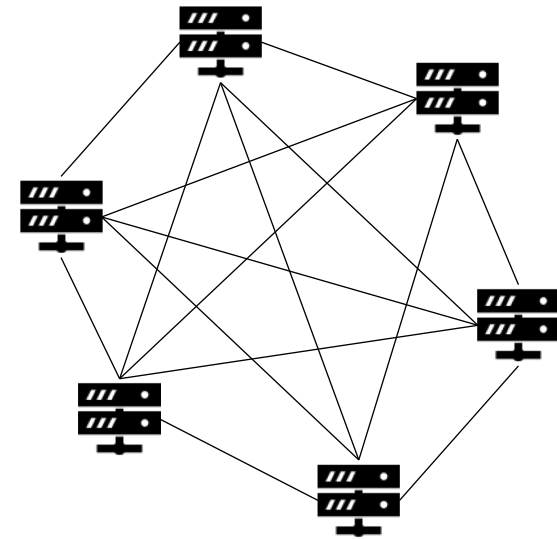


# domainを構成するP2Pネットワーク

- domain毎にフルメッシュのトポロジを持つP2Pネットワークを形成する
  - つまり、1つのcore nodeが複数のdomainに属していればその数だけP2Pネットワークが作られる

- フルメッシュ

- 全てのcore nodeがお互いを隣接nodeとして認識し、接続し合う



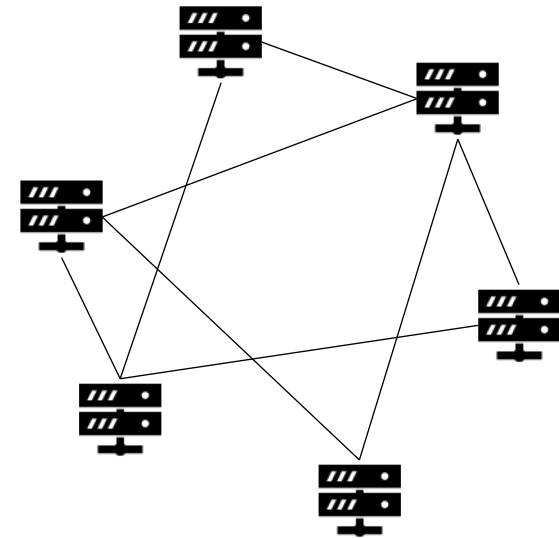
- メリット：耐障害性が高く、制御が容易
- デメリット：スケーラビリティが低い

# domain\_global\_0のP2Pネットワーク (予定)

- domain\_global\_0は将来的には全世界に広がるので、より効率的なP2Pネットワークを作る必要がある
  - v1.0時点ではフルメッシュを利用
  - Kademliaなどのアルゴリズムが候補

## • 効率的トポロジ

- 適当なnode同士だけが接続し合う
- 様々な形成アルゴリズムが存在する



- メリット：スケーラビリティが高い
- デメリット：耐障害性は多少低く、制御が難しくなる

# トランザクション

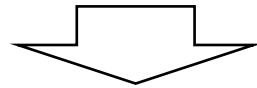
# トランザクションとは

- 定義
  - asset本体および複数のasset間の関係性を記述し、そこに関係者の署名を付加したデータ
- 特徴
  - 1つのトランザクションに、複数のassetを含めることができる
  - 1つのトランザクションに、複数のassetやトランザクションとの関係性を記述できる
  - 1つのトランザクションに、複数の署名を付加できる
- 関係性
  - 関係性とは、具体的には「別のトランザクションへのポインタ」である
  - そこにどのような意味付けをするかはアプリケーション次第である
- 電子署名
  - トランザクションに電子署名を付加することによって、署名者がそのトランザクションの内容を承認したことを表す

# トランザクションが表現できること

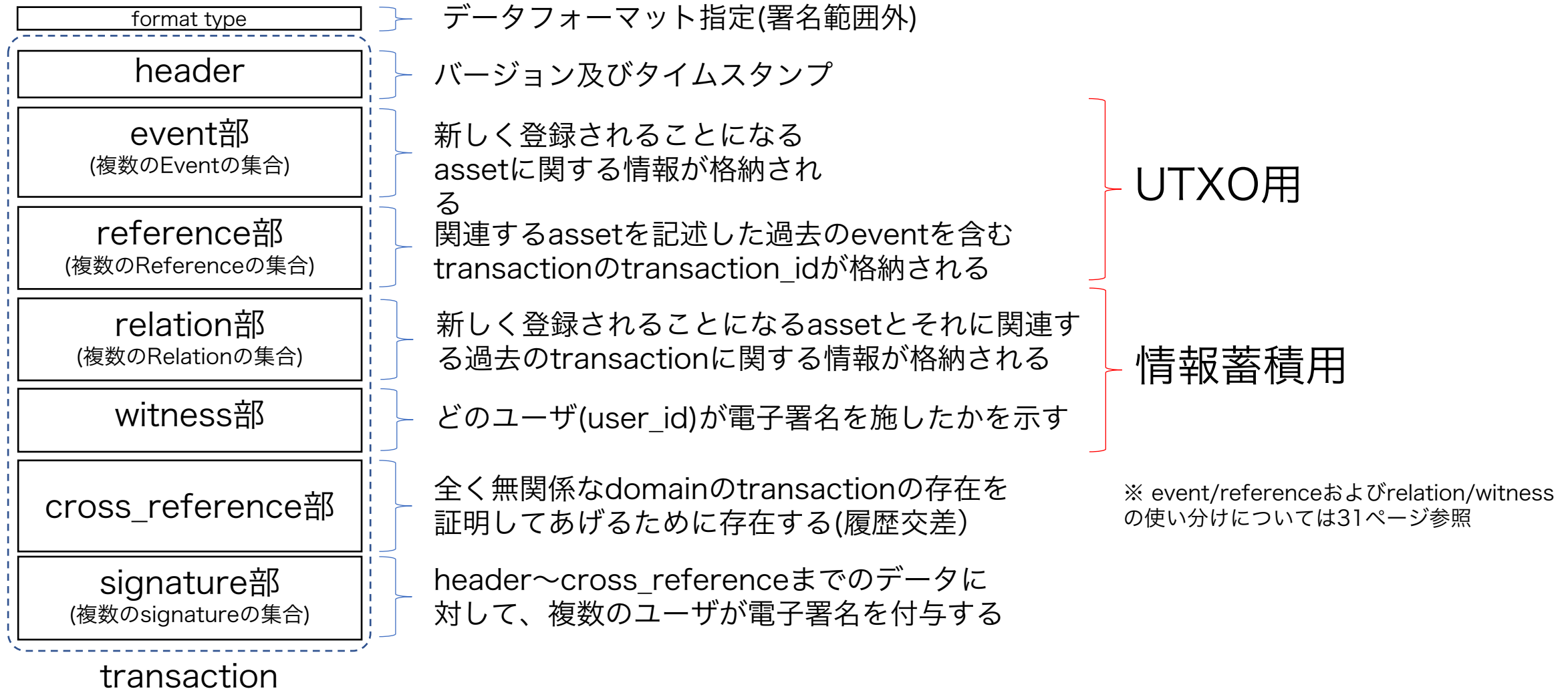
- 情報蓄積型（存在証明、来歴証明など向け）
  - 様々な情報(=asset) をトランザクションに格納する
  - 記録した情報そのものを承認するために、そのトランザクションに電子署名を付加する
- UTXO型（通貨やトークンの支払履歴向け）
  - 通貨やトークン (=asset) の支払いを台帳形式でトランザクションに記述する
  - 受け取ったassetを利用する（さらに支払いする）際に、そのassetの所有者が承認する（つまり電子署名を付加する）

1つのトランザクションで、これらを複数同時に記述できる



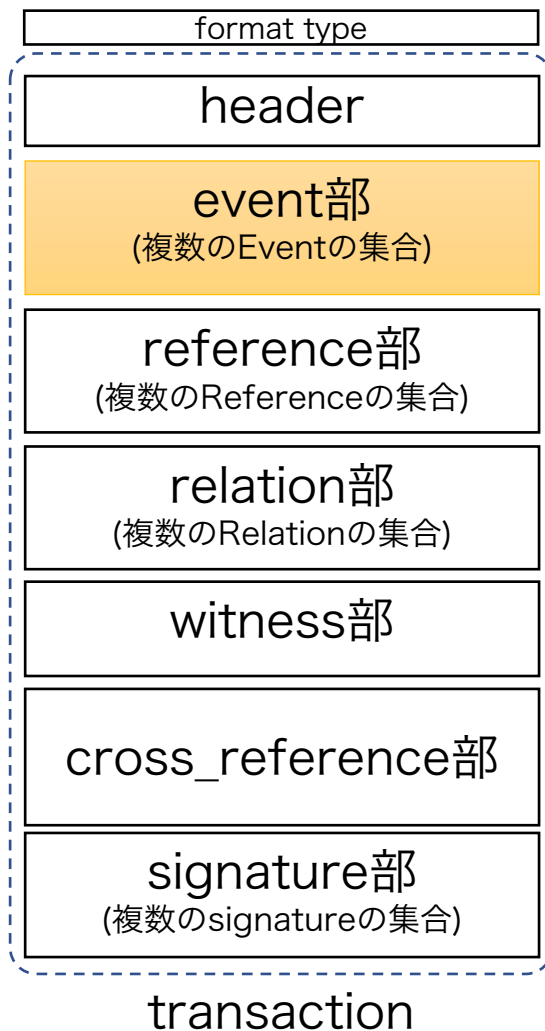
複数のassetに対する処理結果にatomic性  
(不可分性) を与える事ができる

# トランザクションのデータ構造 (概要)





# Event部



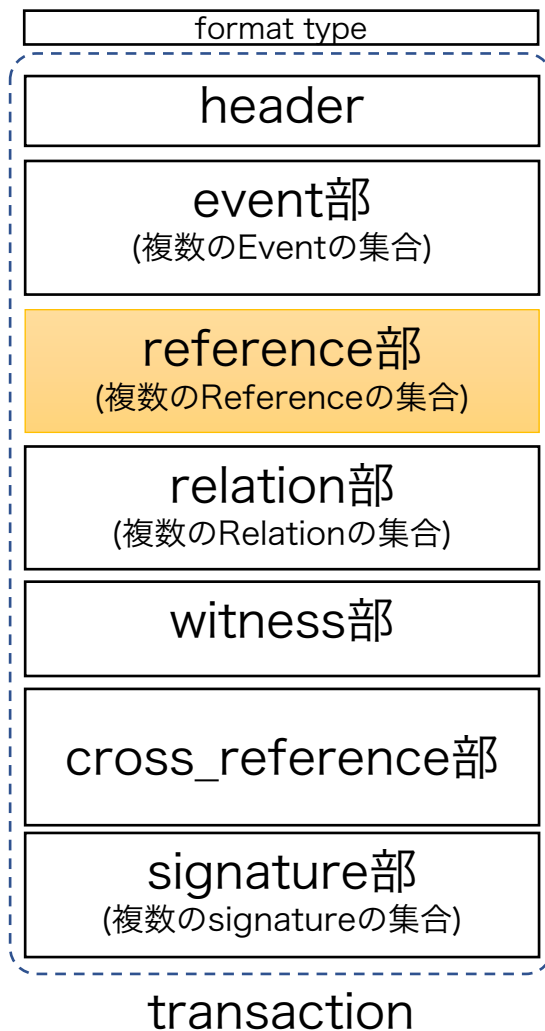
- 含まれる情報

- このEventに書かれたasset（トークンなど）を変更する（支払う）場合に、誰の承認を得なければならないか
  - mandatory approvers' user\_id
  - optional approvers' user\_id
    - optionalによって、M-out-of-N のマルチシグが可能になる

- assetの情報

- asset\_id
- asset\_group\_id
- 作成者 user\_id
- asset本体のサイズ
- asset本体
  - 外部ファイルとして分離保存することも可能
- asset本体のダイジェスト
  - 外部ファイルとして分離した場合に、改ざんがないことを確認するため

# Reference部



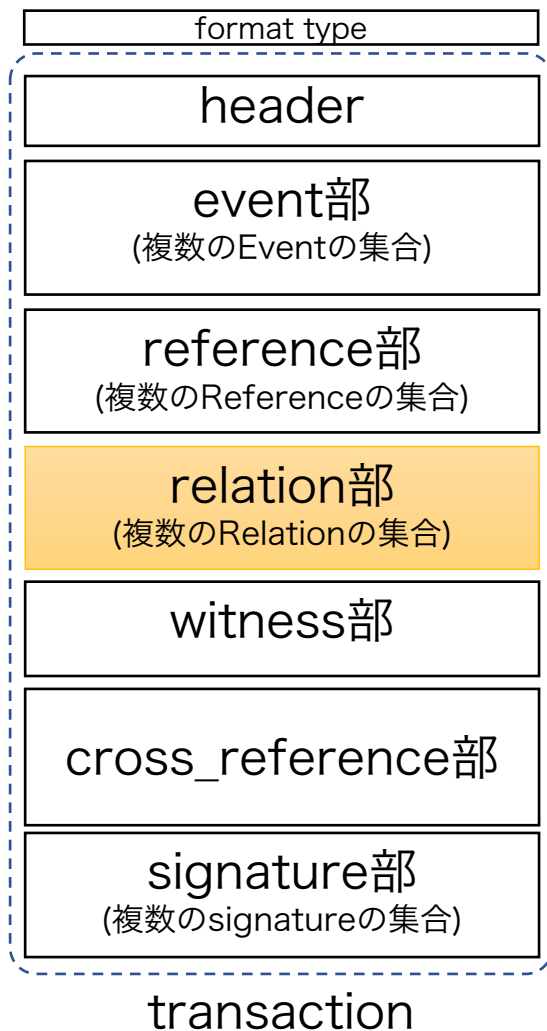
- 含まれる情報

- 参照すべきtransactionとその中のevent
  - asset\_group\_id
  - transaction\_id
  - そのtransactionの何番目のeventか？

- 電子署名による承認

- 過去のtransactionのevent部で登録されたassetを、このtransactionで変更する（例えばトークンを別の人に支払う）ことが承認されたことを表現する
  - 誰に承認を得るべきかは、指定されたevent部のmandatory approversおよびoptional approversに記載されている
- ここで指定したtransactionのevent部で指定されているapproverによる署名がsignature部に付加されなければならない

# Relation部



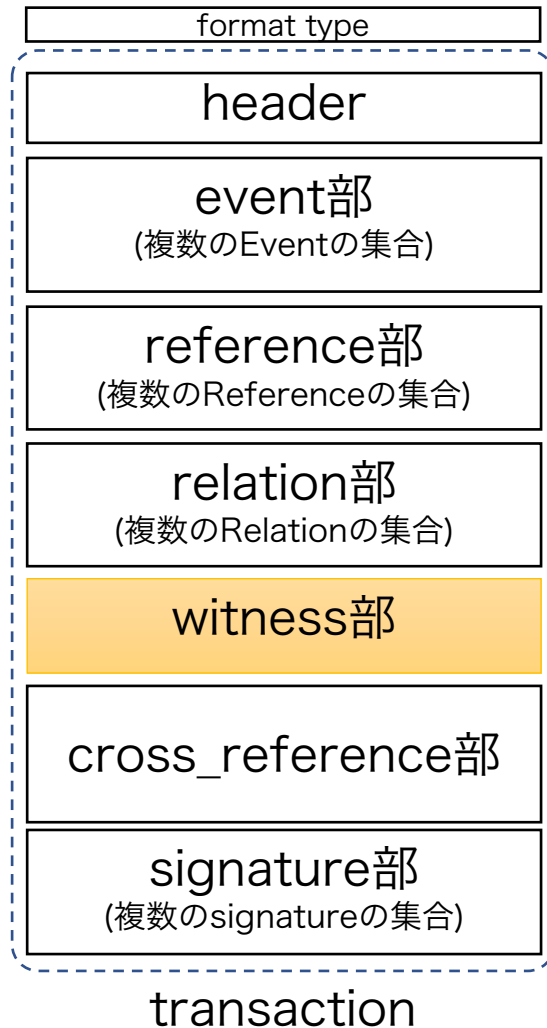
## • 含まれる情報

- asset\_group\_id
- 過去の関連するtransaction/asset情報(=pointer情報)
  - transaction\_id
  - asset\_id
- assetに関する情報 (Event部の中のassetと全く同じ)

## • Event部との違い

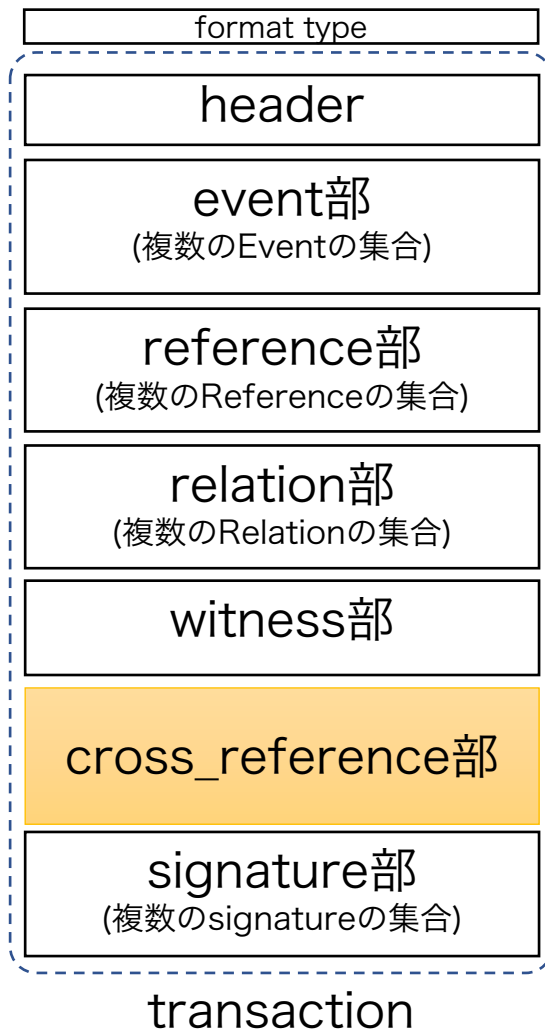
- 過去の関連するtransactionやassetを参照する事ができるが、それが何を意味するかは、アプリケーション次第である
  - EventのようなUTXOなど特定の形式を意図しておらず、approverという概念を持たない
- 単純にassetの情報を記述する

# Witness部



- 含まれる情報
  - user\_idのリスト
  - signatureの配列要素のリスト
- 電子署名を付与していることの明示
  - どのユーザ(user\_id)がどの署名を行ったかを示す
  - そのトランザクション全体を承認したことを表す
    - ただし、何をもって承認するかはアプリケーション次第である

# Cross\_Reference部



- 含まれる情報
  - domain\_id
  - transaction\_id
- 効用 (→履歴交差)
  - 無関係なdomainのtransaction\_idを記載する
  - ここに書かれたtransaction\_idが「そのdomain\_idに確かに存在した」ということを証明することになる
    - 可能なら無関係なdomain上のトランザクションのtransaction\_idをここに載せるべき
    - トランザクション削除などの改竄によってtransaction自体の存在を否認するためには、無関係なdomainのtransactionまで改ざんしなければならないため、強い改ざん耐性をもたせることができる

# Signature部



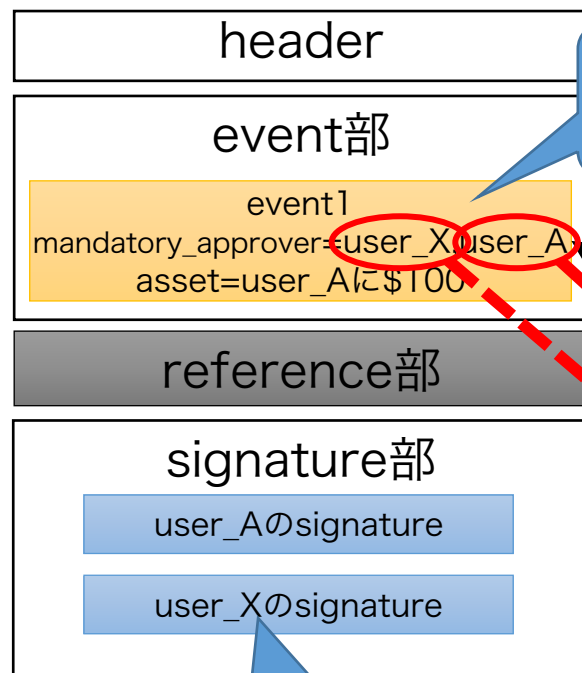
- 含まれる情報
  - 鍵タイプ
  - 鍵長
  - 公開鍵
    - v1.0では、非圧縮のECDSA (SECP256k1) に対応
  - 電子署名

# Event/ReferenceとRelation/Witnessの 使い分け

- EventとReferenceはUTXO型の表現に用い、「assetを操作する権利が移動したこと」を表すための構造を持っている
  - あるtransactionのEventに含まれているassetは、そのEventでapproverに指定されている人の承認がないとassetを操作できない
  - 操作するときは、次の新しいtransactionの中にReferenceを置き、その中で「approverを指定しているEvent/Transaction」を指定し、署名を付与する
  - つまり、UTXOの構造を意識している
- Relationは、UTXO型に拘らずに、単に情報を蓄積し、また他のassetやtransactionと関連付けを行いたいときに利用する
  - 何に関する関連性を表すかはアプリケーションが自由に定義すれば良い
- Witnessは、transactionに何らかの理由で署名を付与する場合に利用する
  - どのような理由で署名を付与するかはアプリケーションが自由に定義すれば良い

# 事例1 (トークン支払い)

## トークン発行トランザクション



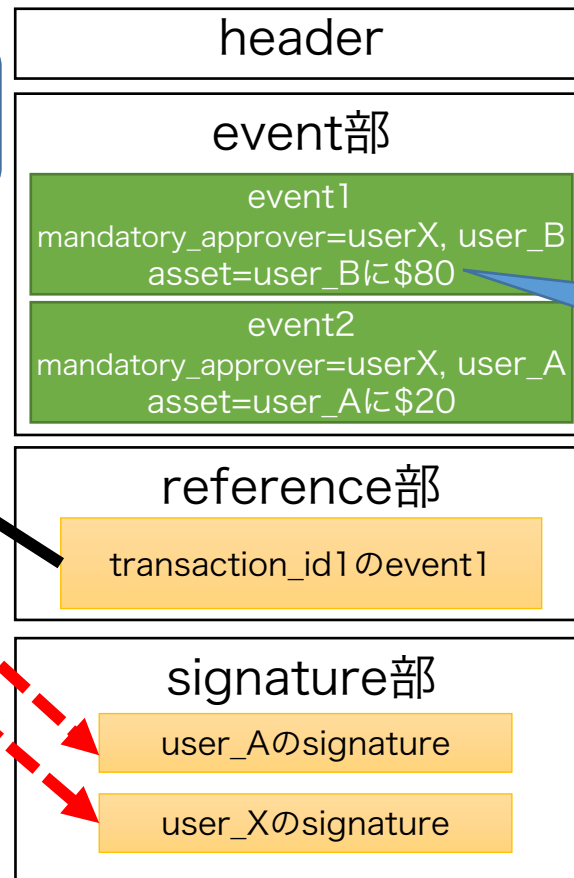
user\_Aが\$100  
を入手した

user\_Xはトークン  
発行者で、健全に運  
営する責務がある

参照先にuser\_Xと  
user\_Aの署名をもら  
うよう記載されている  
のでuser\_Xとuser\_Aの  
署名を付加する

参照

## トークン支払トランザクション



## シナリオ

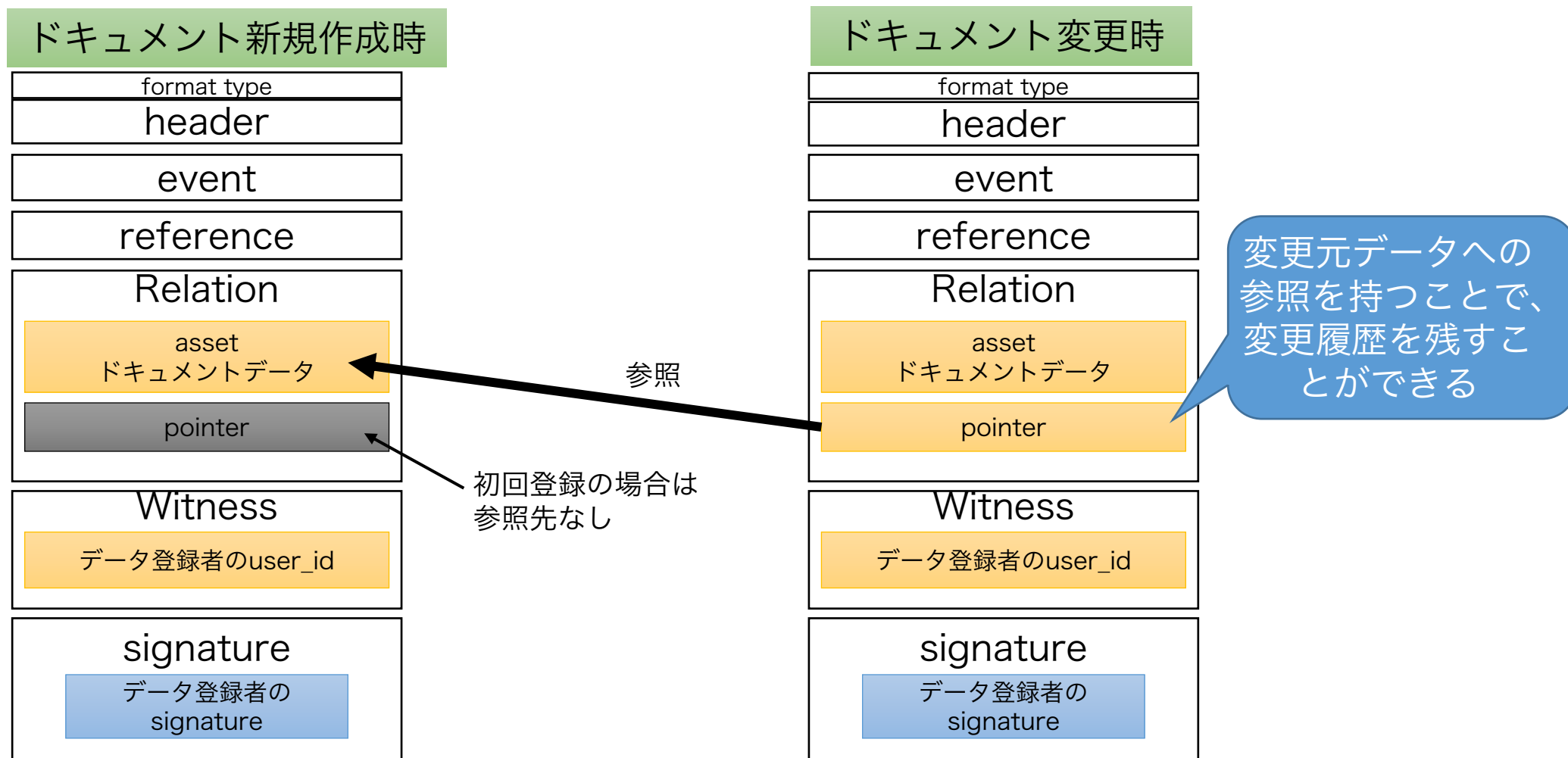
user\_Aがuser\_Bに通貨  
を\$80支払い、お釣りを  
\$20受け取る

user\_Bが\$80  
を入手した

ただし、user\_Xは過去のtransaction  
群をチェックして、**トークンの2重消  
費などの不正が行われていないことを  
確認**しなければならない。確認して問  
題なければ、user\_Xは署名を付加する



# 事例2 (ドキュメント変更履歴)

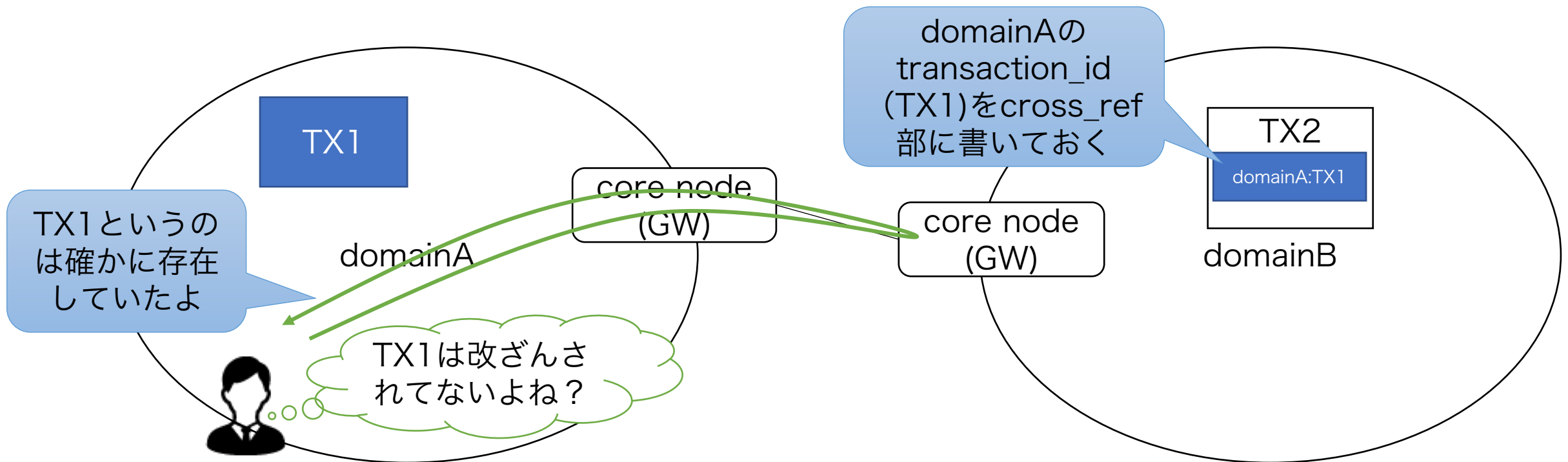


# トランザクションの登録、検索、復旧

- 登録処理
  - clientからcore nodeに投入されたトランザクションは、core nodeによって署名を検証し、正しければドメイン内の全てのDBに複製を登録する
  - 同じくトランザクションに付随するassetの外部ファイルもドメイン内の全てのストレージに複製が格納される
- 検索処理
  - core nodeはclientから検索要求を受け、ドメイン内のDBからトランザクションを検索する
  - core nodeが改ざんを検知した場合（署名照合失敗）、core nodeはその旨をclientに返答する
- 復旧処理
  - 検索の結果、改ざんが検出された場合、clientからcore nodeに対して当該トランザクションの復旧要求を出す
  - core nodeはドメイン内のDBから正しい（改ざんされていない）トランザクションを探し出し、不正があったDBを正しいもので上書きする
  - つまり、ドメイン内全てのDBの当該トランザクションが改ざんされてしまうと復旧不能となる

# 履歴交差 (Cross\_ref)

- あるドメインで作成、保存されたトランザクションの存在を、外部のドメインのトランザクションに証明してもらう仕組み



以上