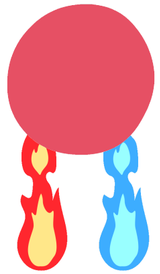
技術解説

HotStuff

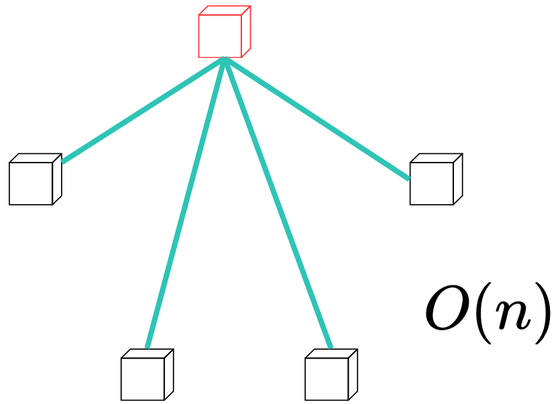


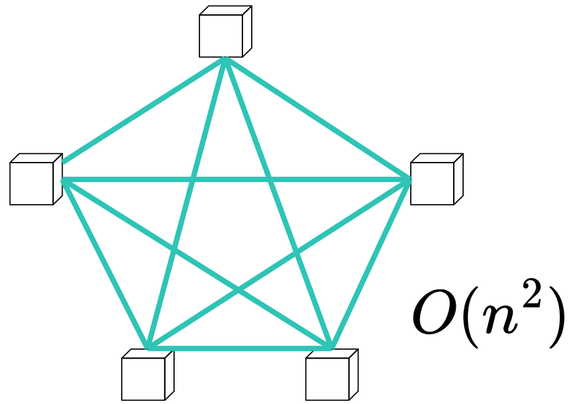


HotStuff



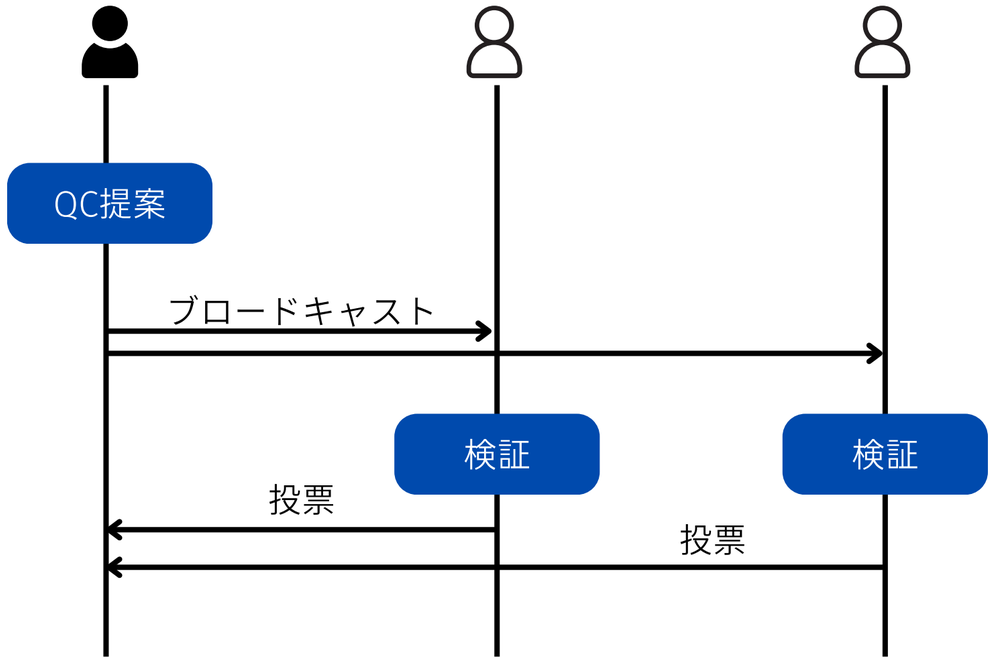
- HyperBFT などのベースになったコンセンサスアルゴリズム
- リーダー主導型BFT
- 部分同期通信モデル(通信が遅かったらリーダーを交代する)
- ビザンチン耐性を持ち、他と同じく正直ノードが3分の2よりも多ければ成立する
- ノード数 n に対して Order(n) の通信量で済む(PBFT などは $O(n^2)$)
- Basic HotStuff と Chained HotStuff がある



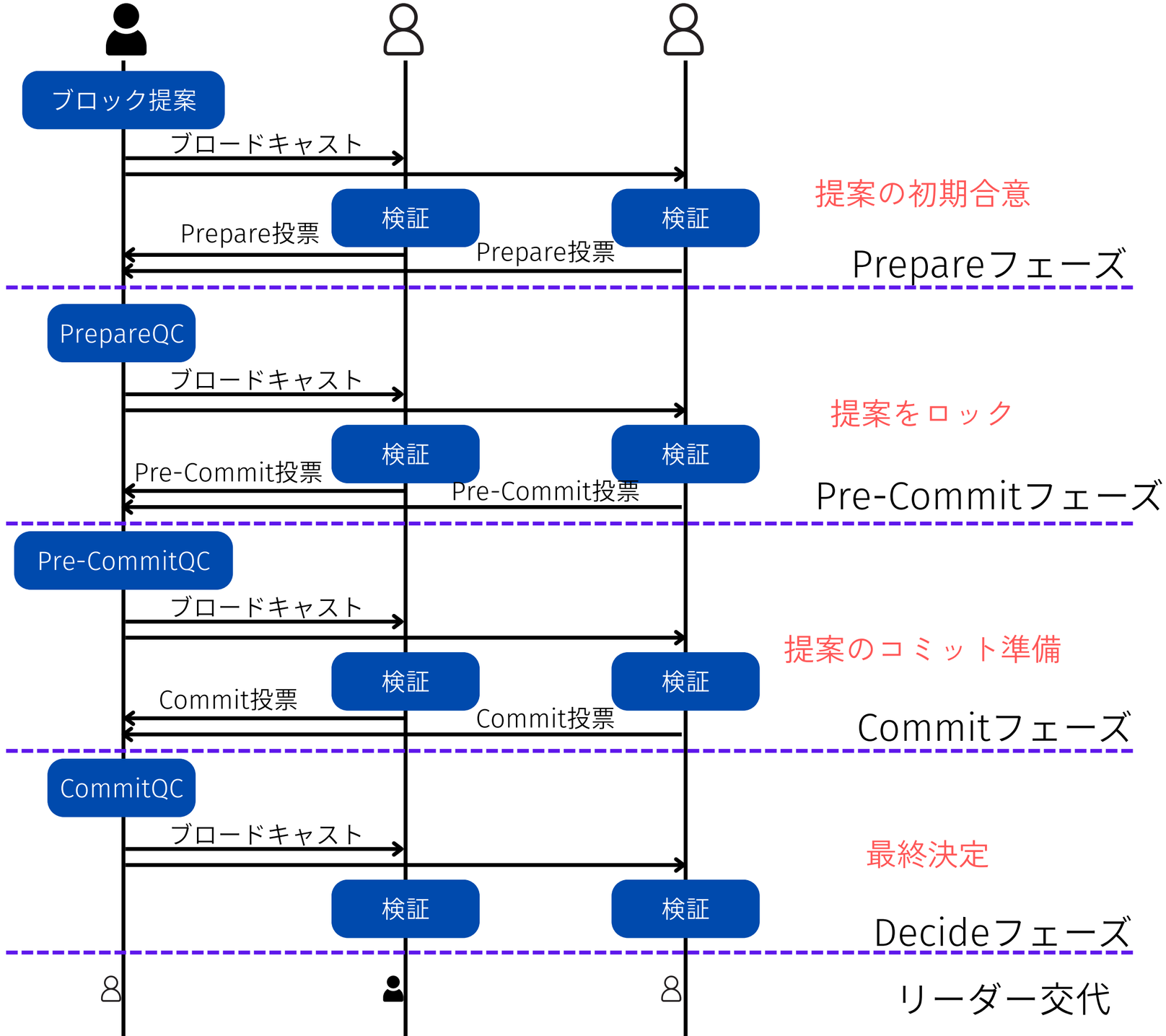


Basic HotStuff

- 4フェーズで段階的にコミットしていく
- 1回 4フェーズのことを「view」と呼ぶ
- リーダーは view ごとに交代する
- QC (Quorum Certificate) という閾値署名などで暗号計算したデータで合意を証明



Basic HotStuff



Chained HotStuff

- 4フェーズ進まないとファイナライズしないのは同じだが、パイプラインで並列に進めるようにした
- QC は前の QC を含むようにすることで並列にできるらしい
- Hyperliquid は “one-block finality” といっているが、HyperBFT がそうだとは言っていないのでユーザ的に 1ブロックに見えるのでは？という推測など。

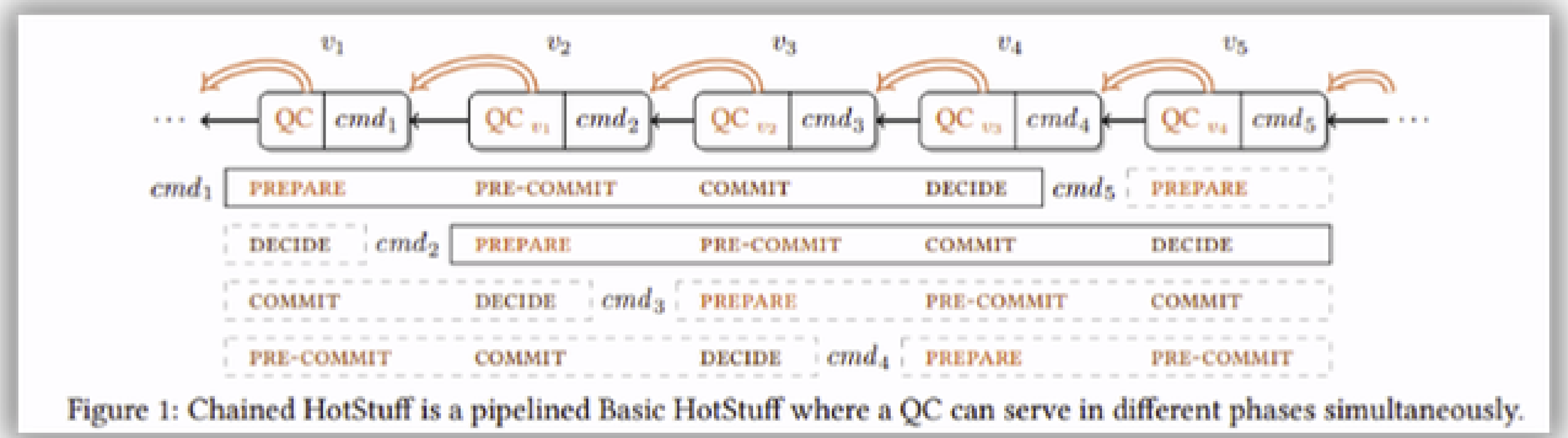


Figure 1: Chained HotStuff is a pipelined Basic HotStuff where a QC can serve in different phases simultaneously.

プロジェクト名	HotStuffとの関連	備考
Aptos (アプトス) 🚀	AptosBFTを採用。これはHotStuffをベースに、パイプライン処理や動的なバリデーターセットに対応するなど、大規模な分散型ネットワーク向けに改良されたものです。	旧Diemの技術基盤を引き継いでいます。
Sui (スイ)	Narwhal & Bullsharkコンセンサスエンジンを採用。データ伝播層としてNarwhal (DAG)、順序付け（コンセンサス）層としてBullsharkを利用しており、BullsharkはHotStuffの概念からインスピレーションを受けています。	DAG構造とHotStuffライクなコンセンサスを組み合わせ、高いスループットを実現しています。
Diem (ディエム) / Libra	LibraBFT（後にDiemBFT）を採用。HotStuffをベースに設計されたコンセンサスプロトコルです。	Meta（旧Facebook）が主導したプロジェクト。プロジェクト自体は終了しましたが、その技術はAptosなどに引き継がれています。
Celo (セロ)	HotStuffをベースにした独自のBFTプロトコルを採用しています。	モバイルファーストの金融プラットフォームとして設計されており、低いレイテンシ（遅延）が特徴です。

- PBFT (Practical BFT) : 1999年にMiguel CastroとBarbara Liskovによって考案
- iBFT (Istanbul BFT) → QBFT : QuorumやHyperledger BESU。JPMorgan開発でConsenSysに売却

Diem は2022年1月に売却して開発中止。

Diem のメンバーの一部がAptos Labsを立ち上げた。

Sui も Diemのメンバーが立ち上げたそうだ。

Diem – DiemBFT

- 改良点
  - Viewタイムアウト時に証明書を発行し、次ラウンドへの移行を同期
  - リーダーに障害があっても提案をコミットできるようにする

改良点	説明	HotStuffとの主な違い
ラウンド同期メカニズム（Timeout Certificates: TCs）	タイムアウト時に証明書を発行し、次のラウンドへの移行を同期。伝送遅延Δ以内のコミット遅延を保証。	HotStuffのPaceMakerを具体化・拡張し、部分同期条件下でのライブネスを強化。
Nil-Block Vote	リーダーの故障時でも、空の提案（nil-block）に対する投票を許可し、コミットを高速化。	HotStuffにない機能で、故障耐性を向上。将来的な実装予定。
TCB-Able Module（Trusted Compute Base対応）	安全モジュールをセキュアハードウェアエンクレーブ内で実行可能にし、攻撃面を低減。	HotStuffでは明示的にサポートされていないセキュリティ強化。
Epoch Reconfiguration	エポックごとにバリデーターセットを再構成し、ネットワークの進化に対応。	HotStuffの基本プロトコルに動的再構成を追加。
リーダーローテーションと投票ルール	ブロックごとにリーダーをローテーションし、故障時でもコミットを許可する投票ルールを強化。	HotStuffの公平性を高め、追加通信なしで実現。

Sui – Narwhal/Bullshark

- 改良点
  - Narwhalがデータの伝播を、Bullsharkがコンセンサスアルゴリズムを担当
    - 別になったのでスループットを上げやすい
  - リーダーは不要ではないが役割が軽いので交代コストを低くしている

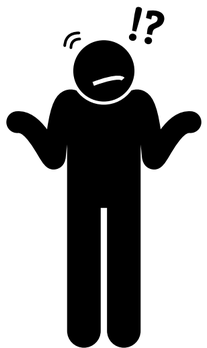
1. 役割の分離（データ伝播と順序付けの分離）

役割	Sui (Narwhal & Bullshark)
データ伝播 (Mempool)	**Narwhal**が担当。トランザクションデータを受信し、DAG構造で複製・保存する。
順序付け (Consensus)	**Bullshark**が担当。Narwhalが構築したDAG構造を**参照するだけで**、順序付けを行う。

特性	Sui (Bullshark)	HotStuff
コンセンサス構造	**DAGベース**。メッセージがグラフを構築し、リーダー（アンカー）の選出と順序付けがグラフの構造から決定される。	**チェーンベース**。ブロックを線形に接続し、QC（クォーラム証明）によって安全なブロックを確定していく。
ビューチェンジ	**ビューチェンジ（リーダー交替）メカニズムが不要**（または大幅に簡素化）。DAG構造が非同期的に成長し続けるため、遅延しているリーダーをスキップできる。	リーダーが故障または悪意の振る舞いをした場合、複雑なビューチェンジプロトコル（HotStuffの大きな貢献点ではある）とタイムアウトメカニズムが必要。
実行と並列処理	Suiの**オブジェクトセントリック**なデータモデルと組み合わせ、依存性のないトランザクションを**並列で実行**できる。	トランザクションは通常、ブロック内で**線形に（順番に）**処理される。

質問

- 他のBFTではいくつくらいフェーズがある？
  - PBFT は 2フェーズ(prepare, commit)
- リーダーの交代(View Change)はどうやる？
  - 決定論的に決まる。例えば View番号とノードIDがマッピングされているなど
- Pre-CommitやCommitってなに？
  - ブロックのファイナリティに向けた段階的な合意形成をしている
  - フェーズが多いのは安全性を高める目的ではなく、View Changeの効率化のため
    - PBFT ではメッセージ交換後に新リーダーが展開して $O(n^3)$ かかり、すべてのノードがログや証明を共有するので複雑で遅延が増大
    - HotStuff はロックとファイナリティを分離することで簡略化



FROST



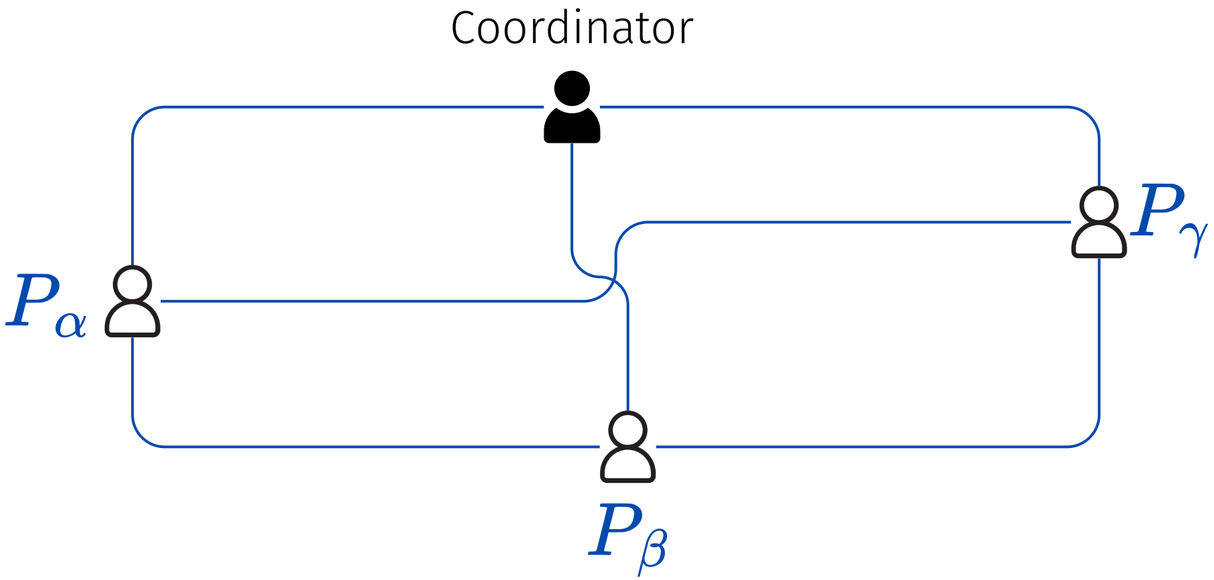


FROST

- Flexible Rounded-Optimized Schnorr Threshold signatures の略
- シュノア署名を基盤とした閾値署名で、RFC-9591で標準化されている
- Bitcoin では MultiSig を柔軟にするために使用する
  - 鍵数が多くてもオンチェーン上では 1鍵に集約される
  - 署名プロセスがMuSig や GG18 よりもラウンドが少ない(2ラウンド)
- 鍵を複数人から集めるので、誰かが止めてしまったり、鍵を集める人が集めなかったりする可能性がある。
  - ROAST署名プロトコルを使って回避することが多い
- 誰かの鍵が漏洩するだけでは影響しないという点ではマルチシグと同じような強度を持つ。ブルートフォース攻撃のようなやり方については単独鍵の強度になる。
  - 鍵を集約するタイプはそうなる

分散鍵生成 Distributed Key Generation : DKG

- 各参加者が鍵を作る
- VSS(Verifiable Secret Sharing)にはいろいろ方式ある
- リーダー的な「集約者」が存在するが、鍵を集約して秘密鍵を求めて署名するのではなく、シュノア署名できるデータを集めて秘密鍵を作らずに署名データだけ作成する。
- 「集約者」は固定でも持ち回りでもどちらでもよい。FROST的には決めていない。
  - 参加者であってもなくてもよい



DKG

P_j シェアを作る

$$f_j(x) = a_{j,0} + a_{j,1}x + a_{j,2}x^2 + \ldots + a_{j,t-1}x^{t-1} \pmod{q}$$

コミットメントを作る

$$C_{j,k} = g^{a_{j,k}}$$

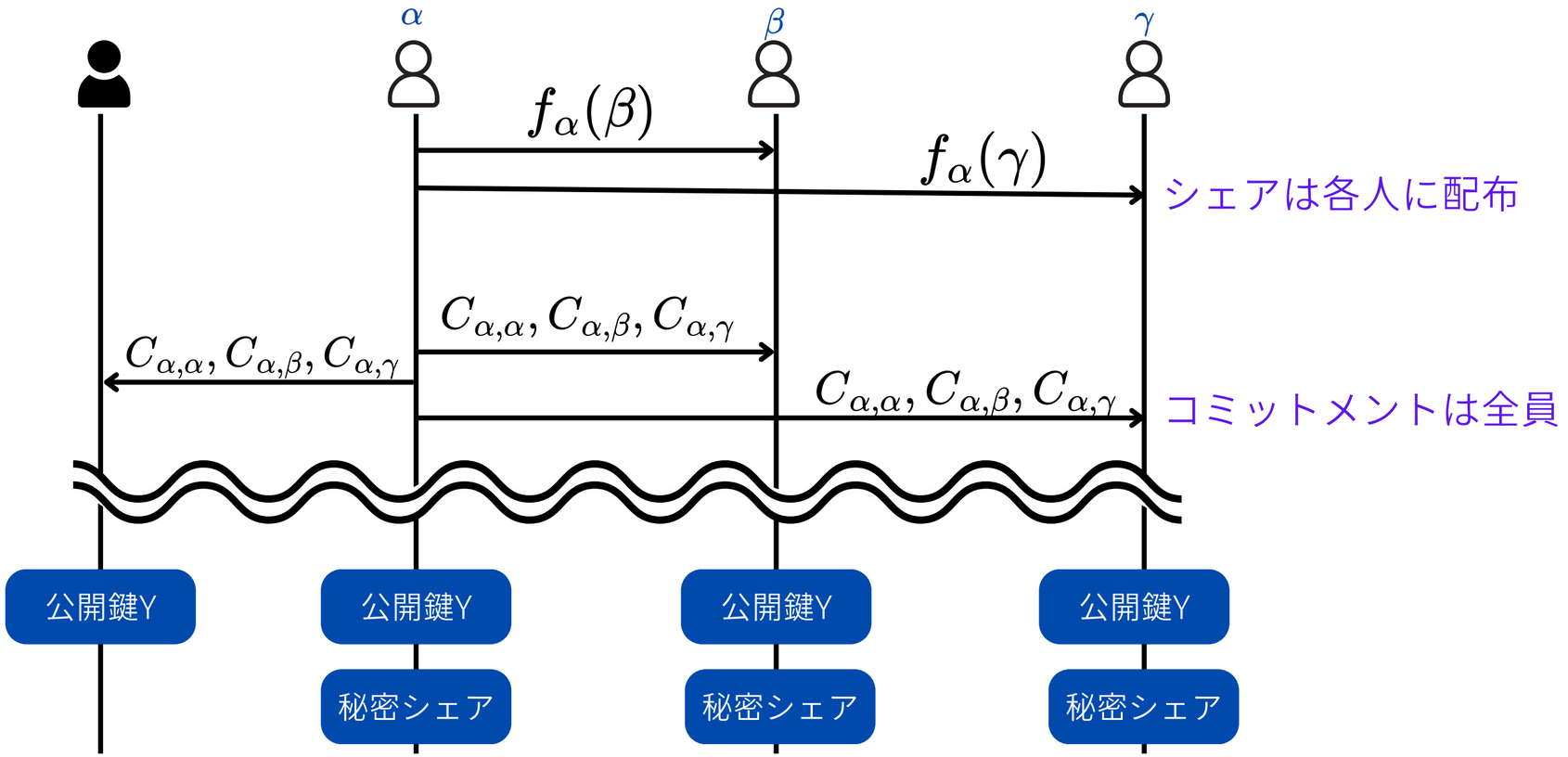
全体の公開鍵Y

この部分が秘密鍵

$$Y = g^{\sum_j a_{j,0}} = \prod_j C_{j,0}$$

秘密シェア

$$s_i = \sum_j f_j(i)$$

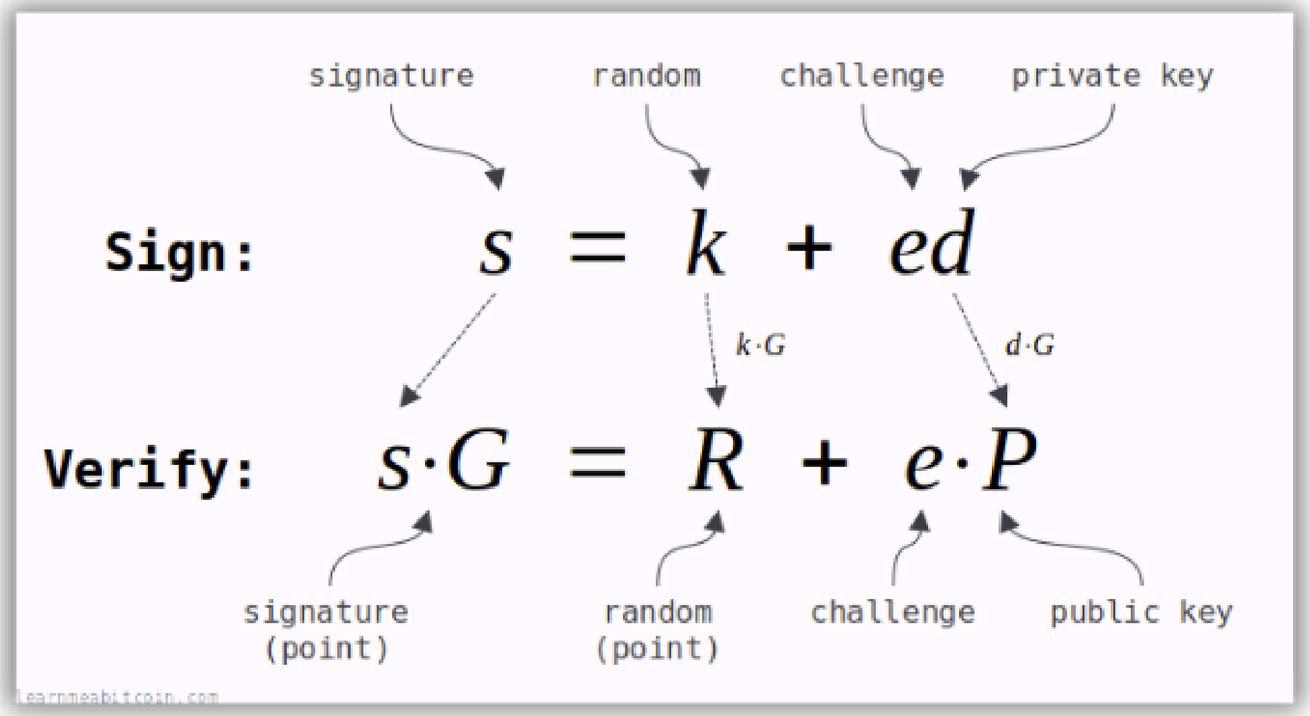


署名フェーズ

- 鍵生成フェーズは最初の1回だけで、それ以降は署名フェーズだけ行う
  - random (nonce) が変わる
  - challenge は BIP-340 のルールで計算される

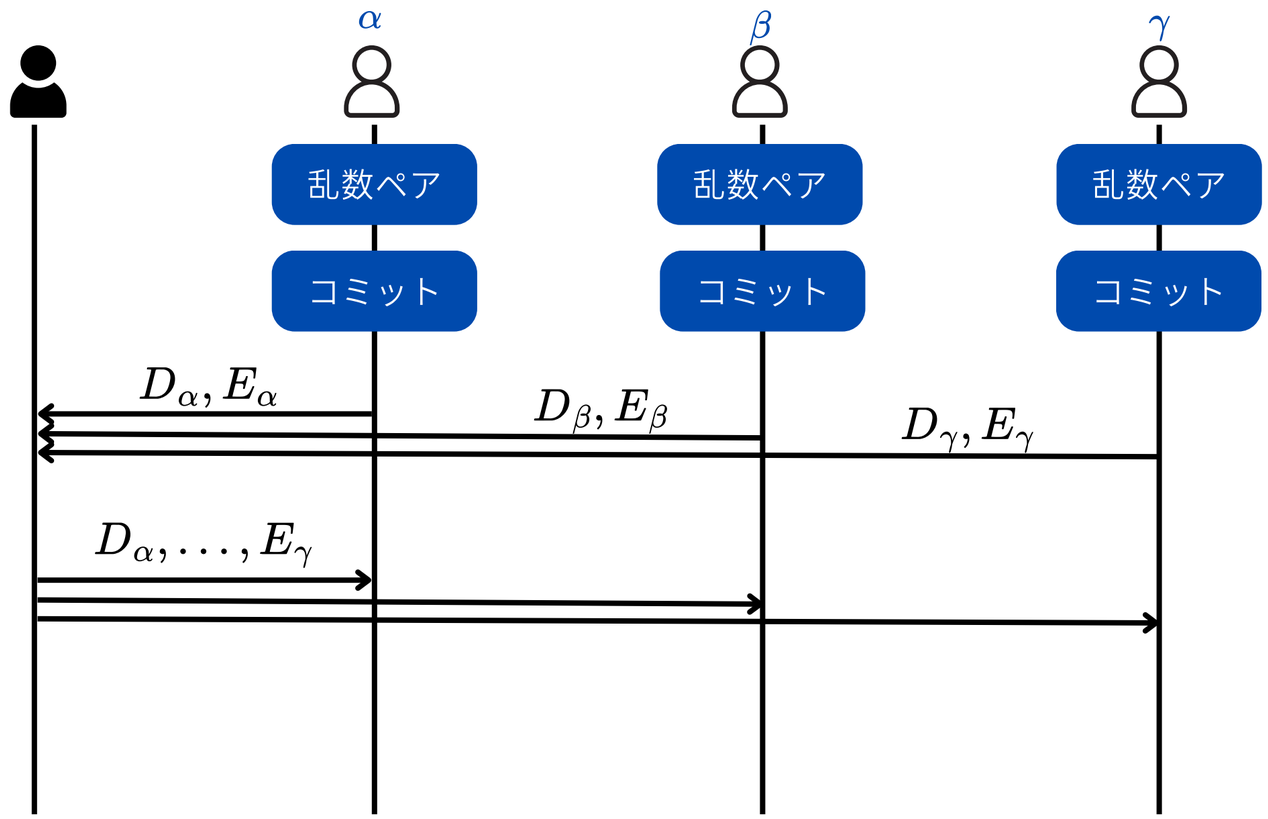
$$e = hash_{BIP0340/challenge}(R.x \parallel P.x \parallel m)$$

- 署名は 2ラウンドで計算される



Round 1: Nonceコミットメントの配布

- 各参加者 P_i が乱数のペア (d_i, e_i) を作る
- コミットメント $D_i = g^{d_i}, E_i = g^{e_i}$
- コミットメントは全員が持つようにする
  - ブロードキャストでもよいし、集約者に集めて配布でも良い



Nonce集約

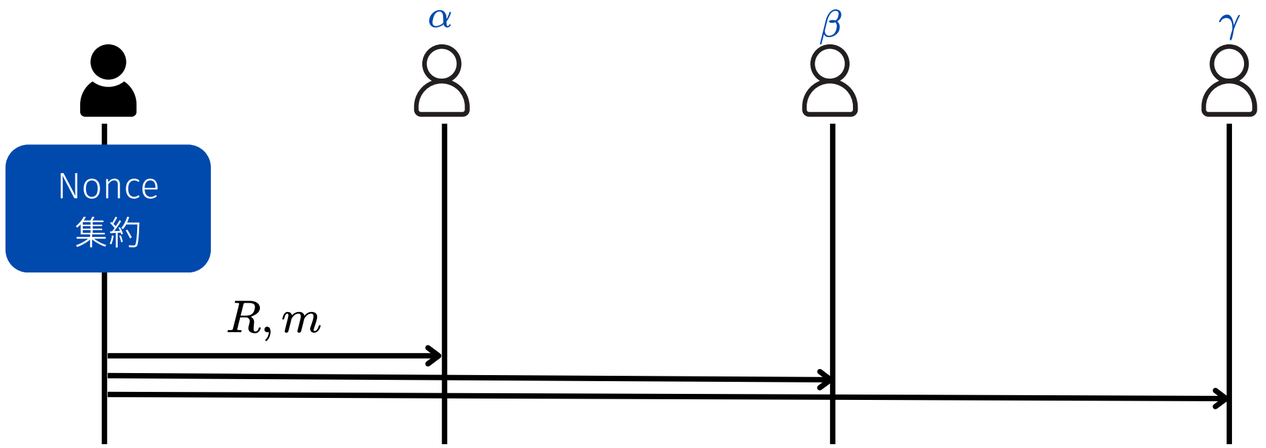
- 集めた Nonce を集約する工程
  - 結合係数
  - グループNonce: R
  - チャレンジ $c = H(R, Y, m)$
    - チャレンジはシュノア署名で使う値

全体の公開鍵Y

メッセージ



- R と m(署名対象のメッセージ)は集約者が全員に配布する
  - チャレンジの値だけだと集約者に悪意があってねつ造するかもしれない



Round 2: 署名シェアの生成

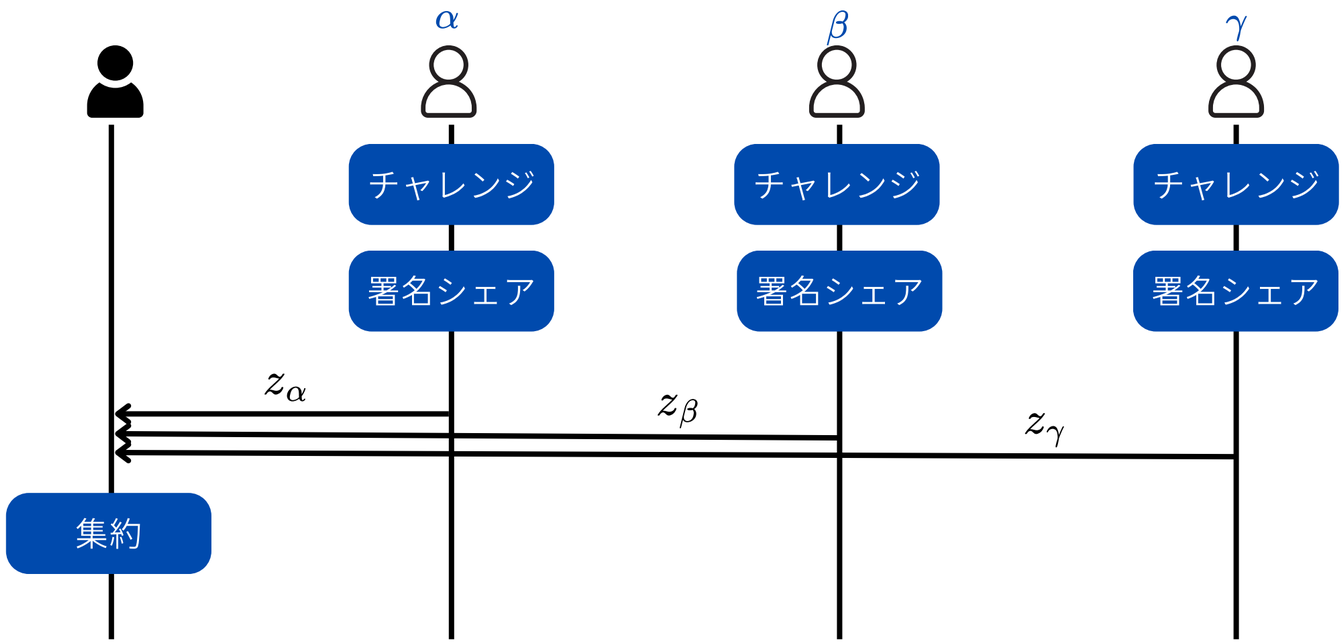
- 参加者 P_i は自分でチャレンジを計算し、署名シェアを計算する

$$z_i = \underbrace{d_i \cdot e_i}_{\text{乱数ペア}} + \underbrace{c}_{\text{チャレンジ}} \cdot \underbrace{\lambda}_{\text{ラグランジュ係数}} \cdot \underbrace{s_i}_{\text{秘密シェア}}$$

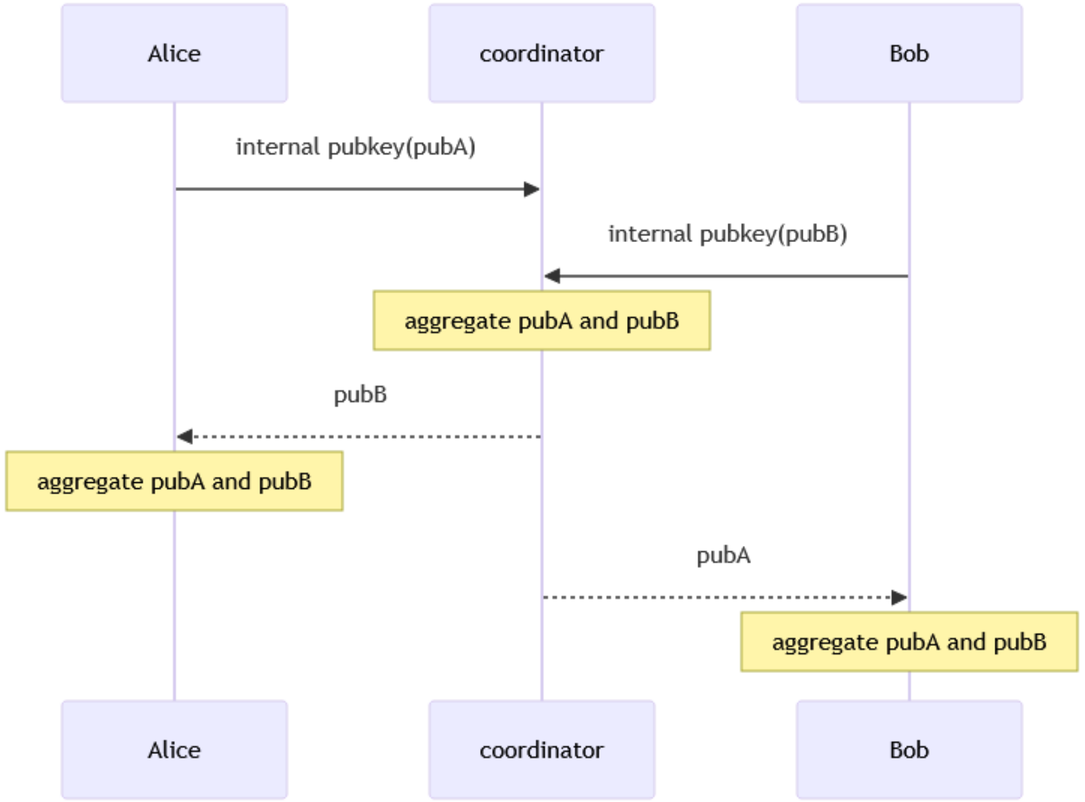
集約と完成

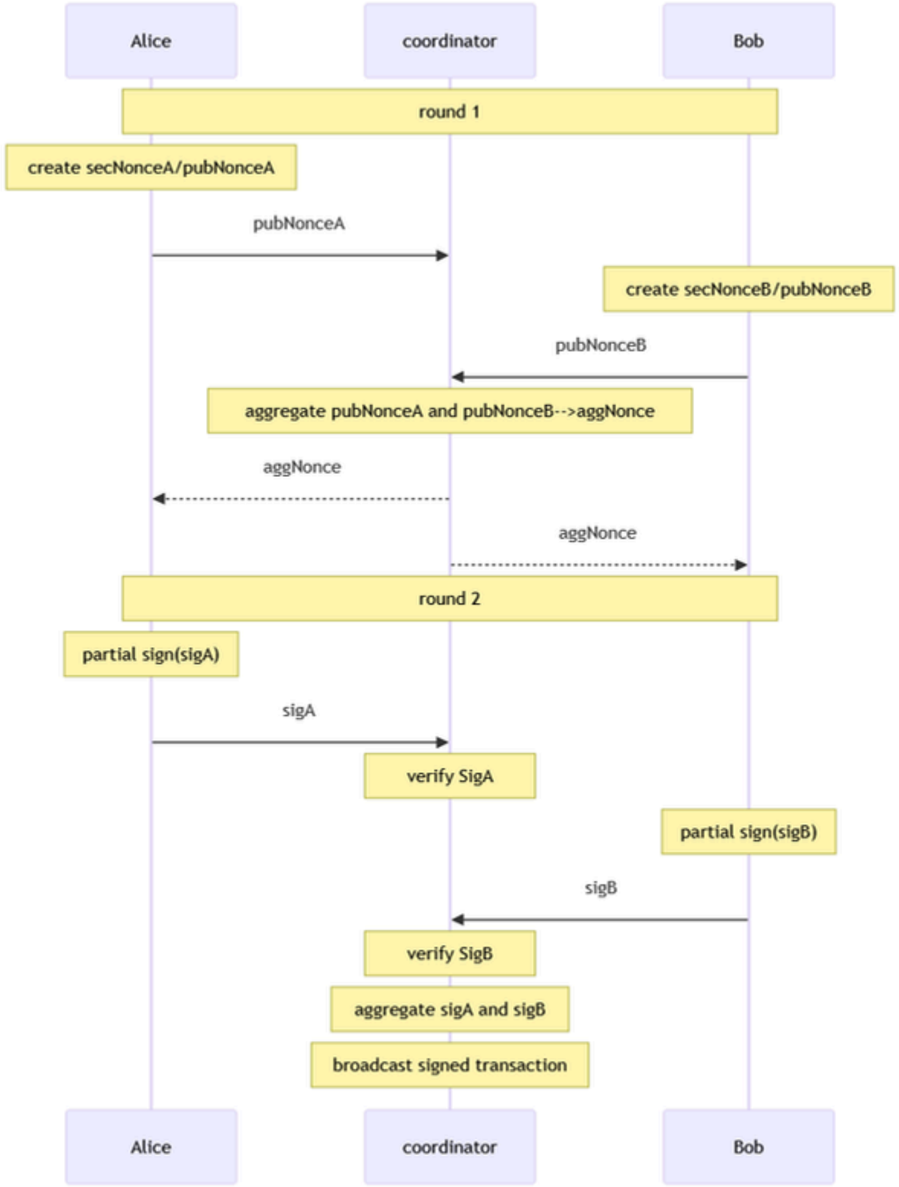
$$z = \sum_i z_i$$

$$sig = (R, z)$$



MuSig2


Alice
coordinator
Bob
internal pubkey(pubA)
internal pubkey(pubB)
aggregate pubA and pubB
pubB
aggregate pubA and pubB
pubA
aggregate pubA and pubB
Alice
coordinator
Bob


Alice
coordinator
Bob
round 1
create secNonceA/pubNonceA
pubNonceA
create secNonceB/pubNonceB
pubNonceB
aggregate pubNonceA and pubNonceB-->aggNonce
aggNonce
aggNonce
round 2
partial sign(sigA)
sigA
verify SigA
partial sign(sigB)
sigB
verify SigB
aggregate sigA and sigB
broadcast signed transaction
Alice
coordinator
Bob