Apache Arrow Flight SQLで PostgreSQLをもっと速く！

須藤功平

株式会社クリアコード

PostgreSQL Conference Japan 2023

2023-11-24

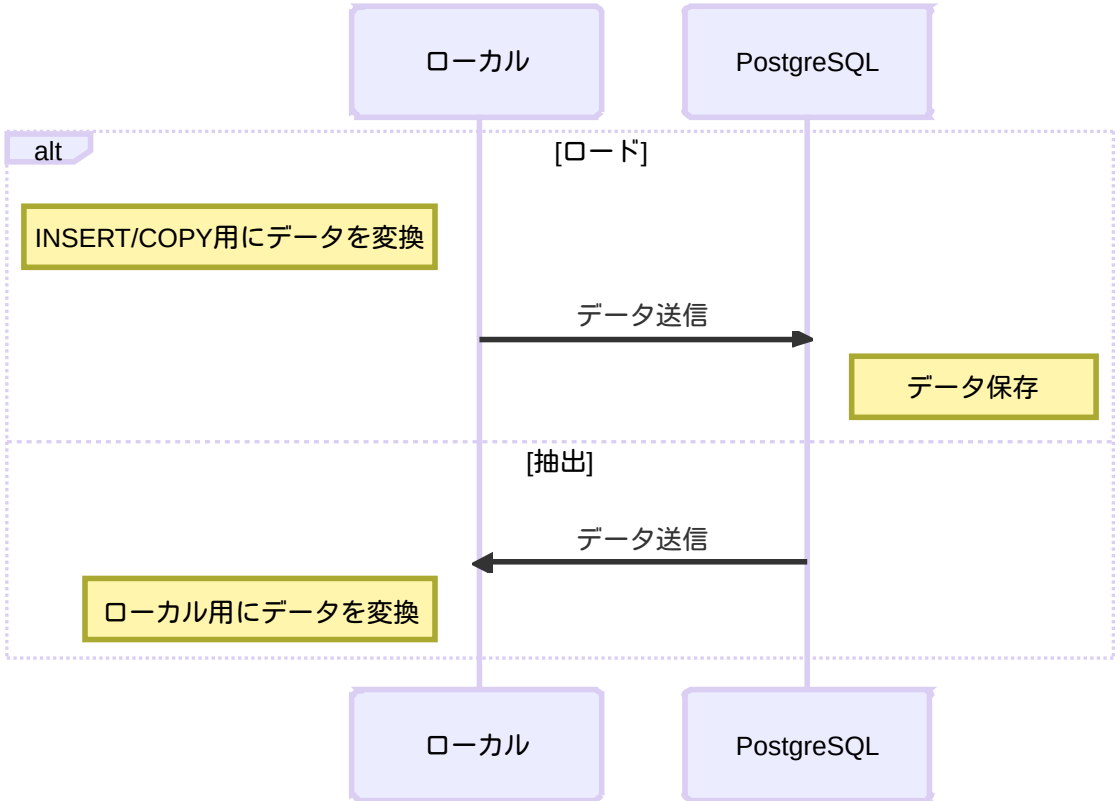
対象ユースケース

- ETL/ELT（Extract, Transform, Load）
  - PostgreSQL**から**の大量データの**読み**込み
  - PostgreSQL**へ**の大量データの**書き**込み
- 探索的データ分析
  - PostgreSQL内のデータを理解
  - サブセットをローカルにダウンロードし、インタラクティブに集計・加工・可視化など

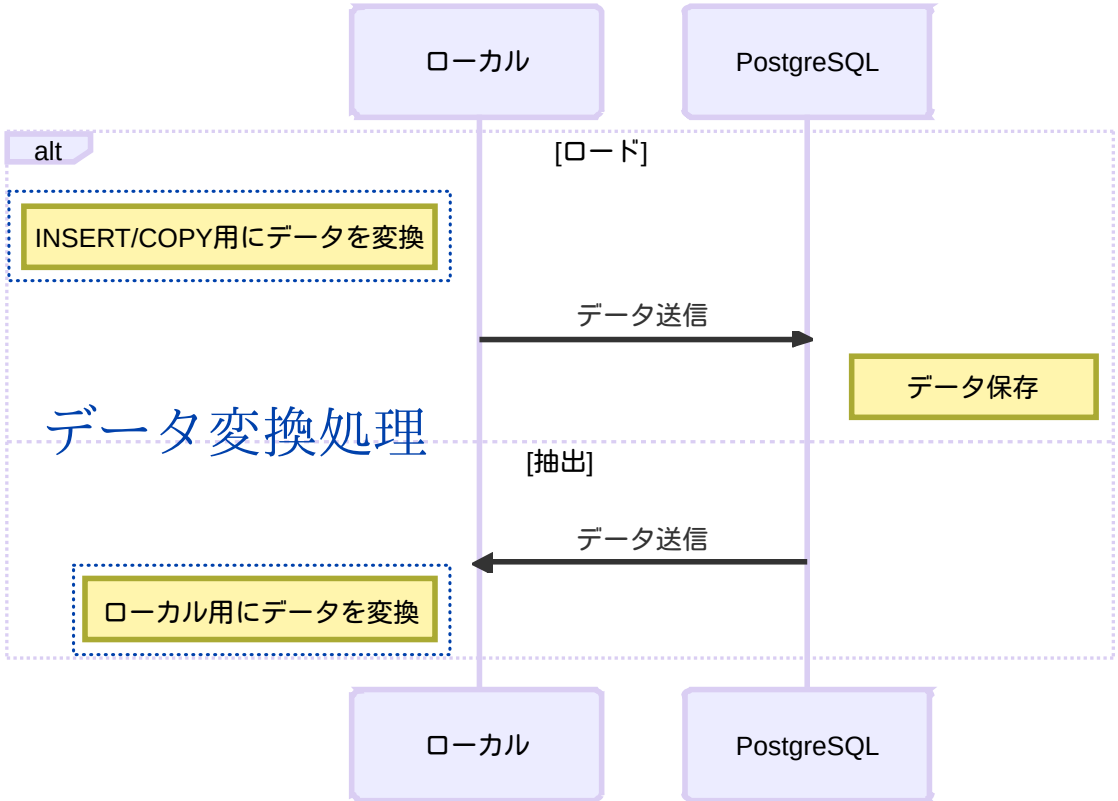
対象ユースケースの特徴

- ✓ 大量ローカルデータ→PostgreSQLデータ
- ✓ 大量PostgreSQLデータ→ローカルデータ

対象ユースケースの処理



注目ポイント


ローカル
PostgreSQL
alt
[ロード]
INSERT/COPY用にデータを変換
データ送信
データ保存
データ変換処理
[抽出]
データ送信
ローカル用にデータを変換
ローカル
PostgreSQL

なぜデータ変換処理に注目するのか

データ量に比例する処理だから

↓

大量データだと影響が大きい

高速化アプローチ

低データ変換コストの
フォーマットを使う

↓

Apache Arrow
フォーマット

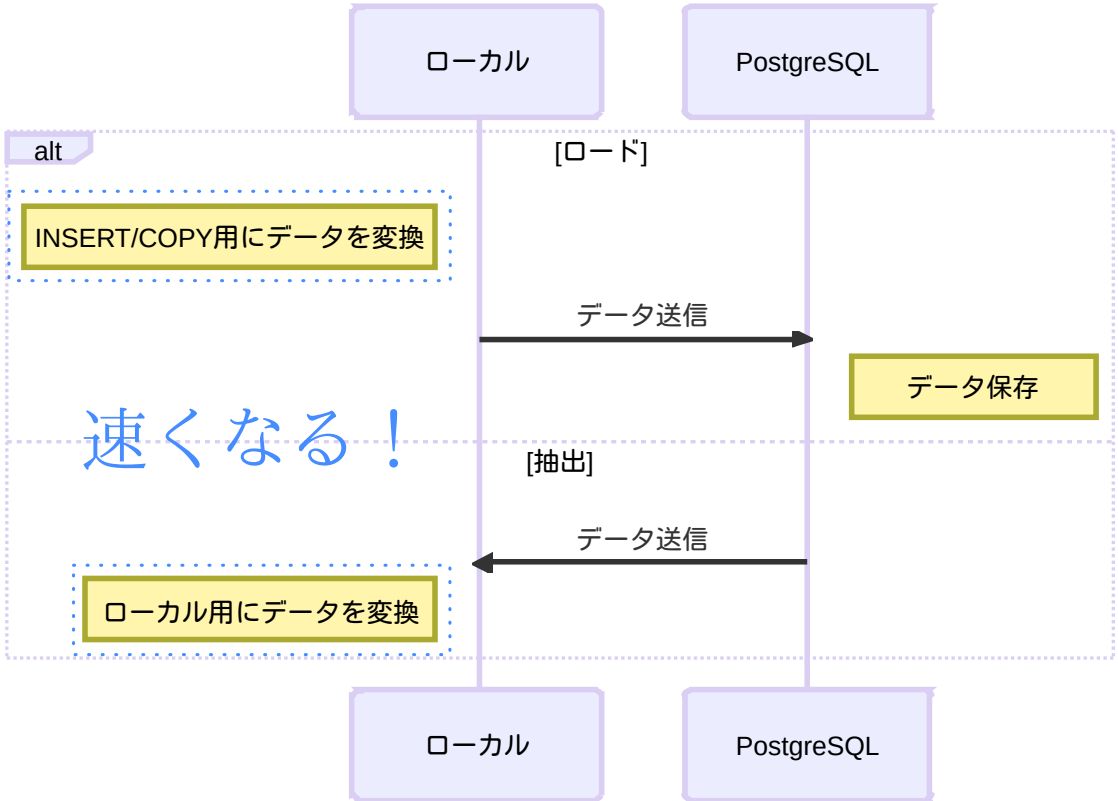
自己紹介

- ✓ 須藤功平/@kou/@ktou
- ✓ Apache Arrowのコミット数1位
- ✓ Apache Arrowの3代目PMC代表
  PMC：プロジェクト管理委員会

Apache Arrowフォーマット

- ✓ データ交換に最適化
  - ✓ パースコストほぼ0
  - ✓ データ転送コストのみで交換できる
- ✓ インメモリーのデータ分析に最適化
  - ✓ ローカルデータの表現としても優秀
  - ✓ 受け取ったデータをそのまま使える

高速化ポイント


ローカル
PostgreSQL
alt
[ロード]
INSERT/COPY用にデータを変換
データ送信
データ保存
速くなる！
[抽出]
データ送信
ローカル用にデータを変換
ローカル
PostgreSQL

実現方法

- ✓ PostgreSQLプロトコルで
  Apache Arrowを扱えるようにする
  - ✓ COPYのFORMATでApache Arrowサポート
- ✓ Apache Arrowを扱える
  別のプロトコルを使う

COPYでApache Arrow

- メリット：
  - 既存の仕組み・ツールを使える
- デメリット：
  - PostgreSQL開発者は受け入れてくれるのか？

Apache Arrow対応の別プロトコル

- メリット：
  - PostgreSQLの拡張機能で実現すれば PostgreSQLとは独立して開発できる
- デメリット：
  - 既存の仕組み・ツールを使えない

選んだ方法

Apache Arrow対応の
別プロトコル

↓

**Apache Arrow
Flight SQL**

Apache Arrow Flight SQL

- ✓ クエリー言語：SQL
  - ✓ PostgreSQLになじむ
- ✓ データフォーマット：Apache Arrow
  - ✓ 高速化できる
- ✓ ベースのプロトコル：gRPC
  - ✓ 既存の仕組みを再利用できる

注意：まだexperimentalなプロトコル

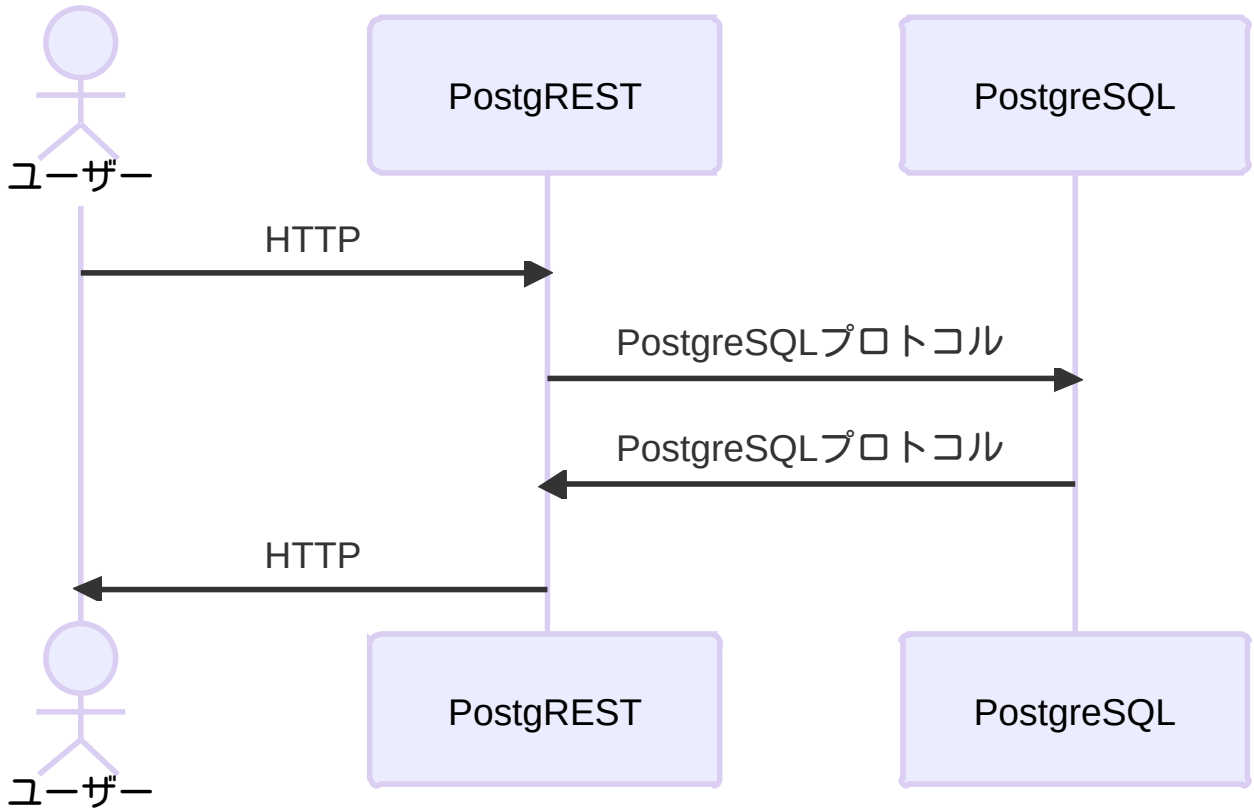
別プロトコルを選んだ理由

- ✓ Apache Arrowでこの問題を解決できるか広く検証したい
  - ✓ PostgreSQLの変更だと試してくれる人が少なそう
- ✓ PostgreSQL開発者と相談するにはApache Arrowでどのくらい改善するかのデータが必要
  - ✓ このアプローチでの実績をもってPostgreSQL開発者と相談したい

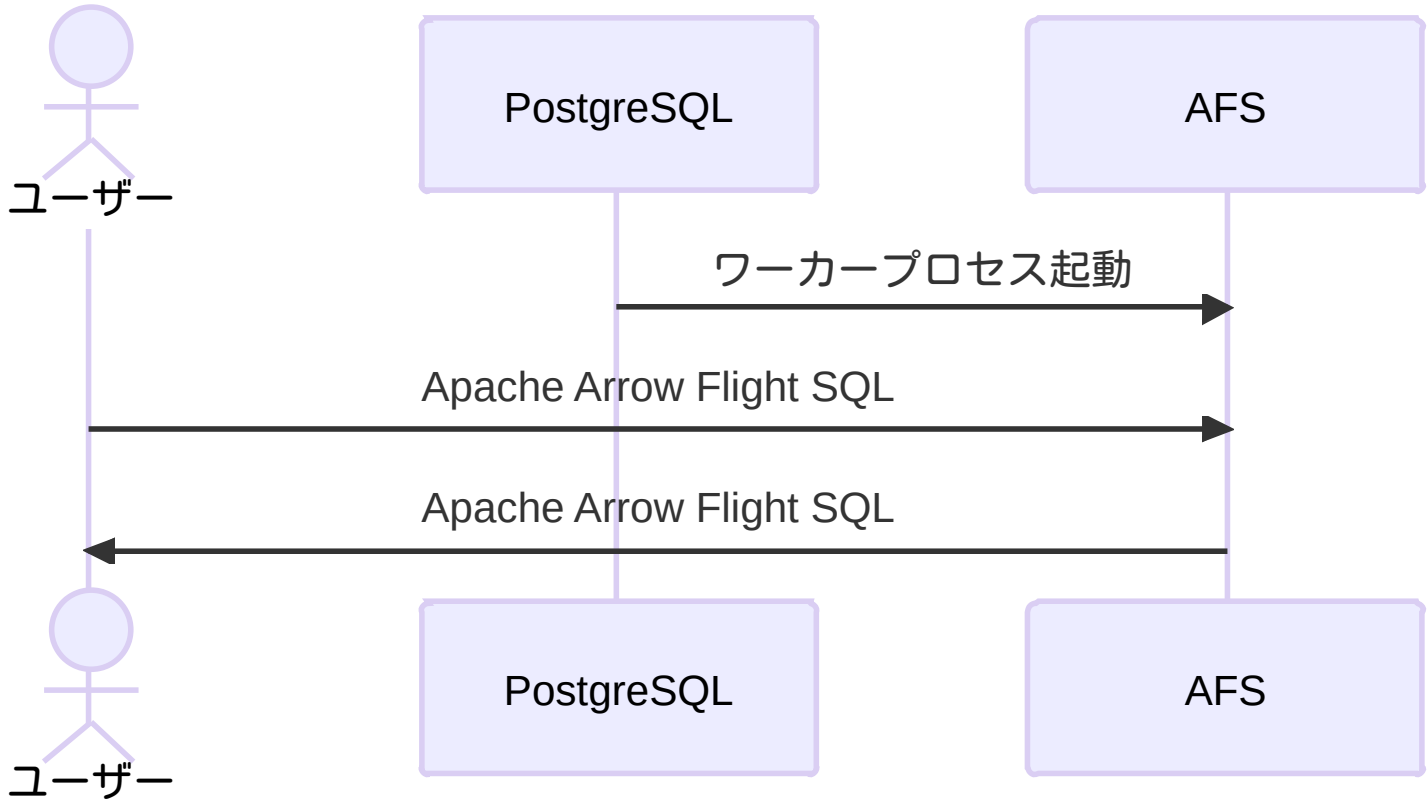
実現方法

- ✓ PostgreSQLの拡張機能として実装
  - ✓ Apache Arrow Flight SQL adapter for PostgreSQL
    https://arrow.apache.org/flight-sql-postgresql/
    （プロジェクト名が長い！以後AFSと省略）
- ✓ 別の実現方法
  - ✓ PostgreSQLのリバースプロキシとして実装
    例：PostgREST

新プロトコル対応方法：PostgREST



新プロトコル対応方法：AFS



Apache Arrowで速くなるのか？

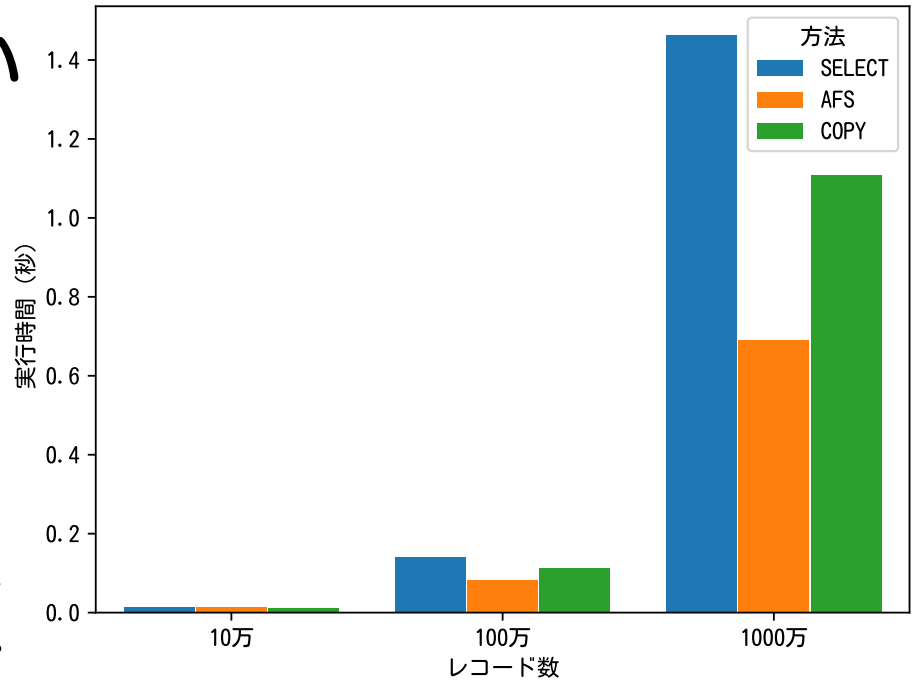
- ✓ 検証1：integerのSELECT
- ✓ 検証2：stringのSELECT
- ✓ 検証3：integerのINSERT
- ✓ 検証4：stringのINSERT

検証1：integerのSELECT

- 1カラムのみN件のランダムデータ
  - N: 10万、100万、1000万
- 比較対象
  - libpqでSELECTして結果をパース：ベースライン
  - AFS：提案方法
  - libpqでCOPYして結果をパース：既存の高速な方法
- PostgreSQL：17（未リリース）

検証1：integerのSELECT：結果

- ✓ バーが短いほど速い
- ✓ 10万レコードでは
  どれも変わらない
- ✓ ✓AFSが速い
- ✓ 参考：カラム数が
  増えるほど差は開く

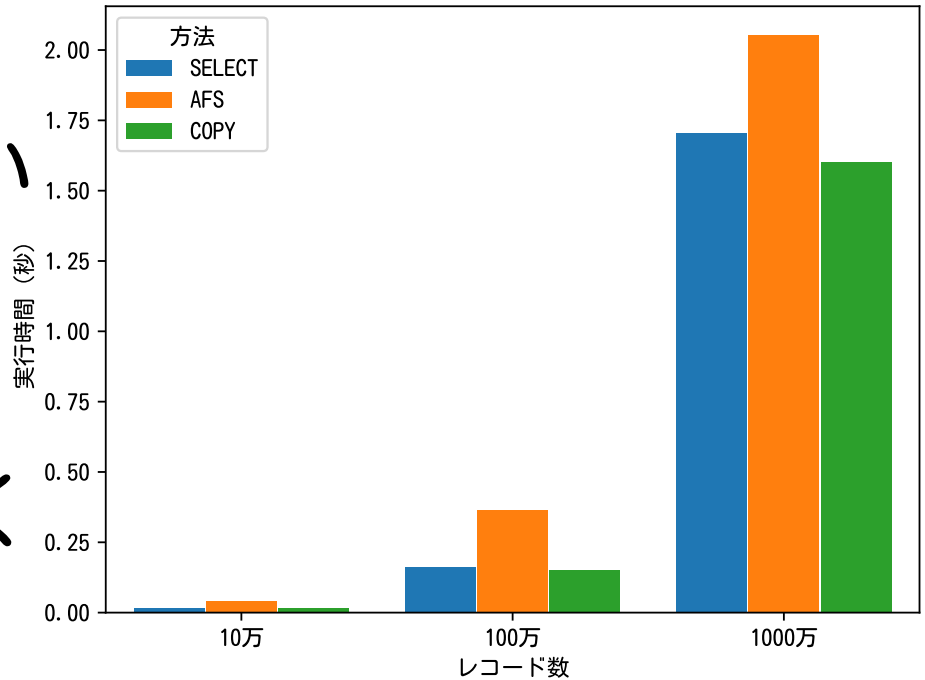


検証2：stringのSELECT

- 1カラムのみN件の64バイトランダムデータ
  - N: 10万、100万、1000万
- 比較対象
  - libpqでSELECTして結果をパース：ベースライン
  - AFS：提案方法
  - libpqでCOPYして結果をパース：既存の高速な方法
- PostgreSQL：17（未リリース）

検証2：stringのSELECT：結果

- ✓ バーが短いほど速い
- ✓ ×AFSが遅い
- ✓ 参考：カラム数が増えるほど差は開く



AFSがなぜ遅いか

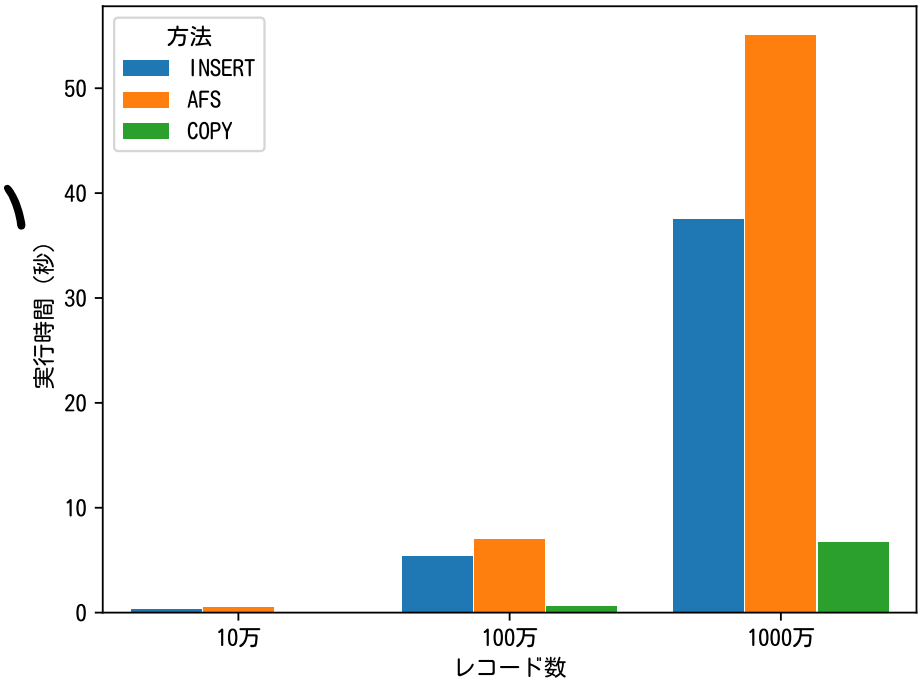
- ✓ そもそもSELECT/COPYが速い
  - ✓ データをそのまま送っているのでパースコスト0
  - ✓ 数値は文字列表現→数値の変換が必要
- ✓ 無駄なデータコピーがある？
- ✓ SPI_getbinval()が遅い？

検証3：integerのINSERT

- 1カラムのみN件のランダムデータ
  - N: 10万、100万、1000万
- 比較対象
  - INSERTを作ってlibpqで実行：ベースライン
  - AFS：提案方法
  - COPY用データを作って実行：既存の高速な方法
- PostgreSQL：17（未リリース）

検証3：integerのINSERT：結果

- ✓ バーが短いほど速い
- ✓ ×AFSが遅い
- ✓ COPYが速い



AFSがなぜ遅いか

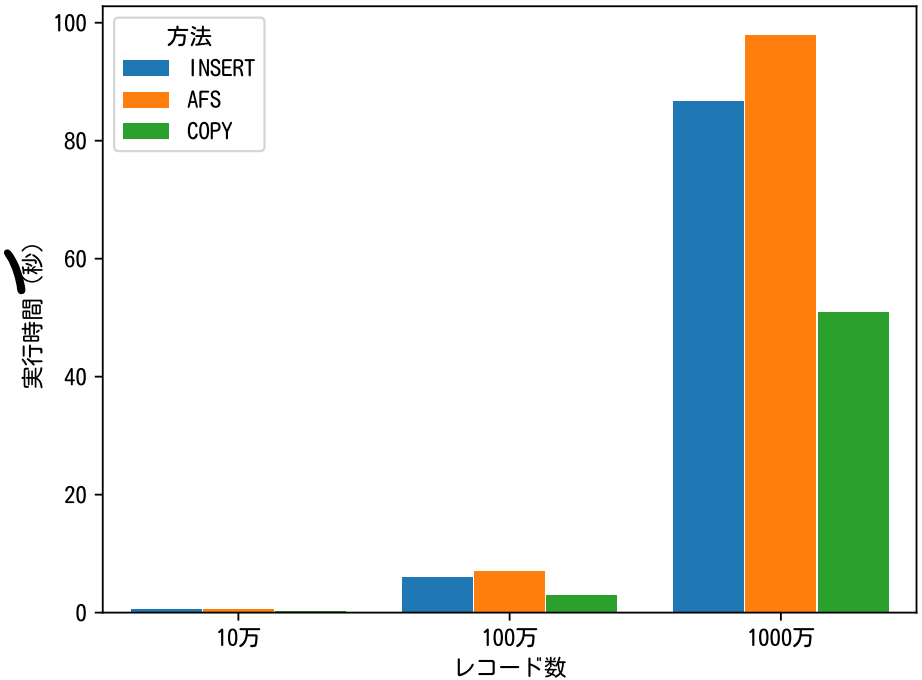
- ✓ N回SPI_execute("INSERT")しているから
  - ✓ 現在はFlight SQLにバルクインサート用APIがない
  - ✓ PREPARE + パラメーターバインドで実現
- ✓ 今後：
  - ✓ バルクインサート用APIを設計中
    https://github.com/apache/arrow/issues/38255
  - ✓ より効率的な内部実装にできる予定
    （でもSPIでCOPY FROM STDINを使えないんだよな…）
    （SPI_register_relation()は速いのかな？）

検証4：stringのINSERT

- 1カラムのみN件の64バイトランダムデータ
  - N: 10万、100万、1000万
- 比較対象
  - INSERTを作ってlibpqで実行：ベースライン
  - AFS：提案方法
  - COPY用データを作って実行：既存の高速な方法
- PostgreSQL：17（未リリース）

検証4：stringのINSERT：結果

- ✓ バーが短いほど速い
- ✓ integerと同じ傾向



ユーザー向けの話のまとめ

- ✓ 課題：大量データの読み書きが遅い
- ✓ 解決方法：
  - ✓ Apache Arrowフォーマット/Flight SQL
- ✓ 現状：
  - ✓ ✓数値の読み込みは速い→今すぐ体験できる！
  - ✓ ×文字列の読み込みは遅い→今後に期待！
  - ✓ ×書き込みは遅い→今後に期待！

開発者向けの話

- ✓ Apache Arrow Flight SQL
- ✓ 認証
- ✓ 追加プロトコル対応

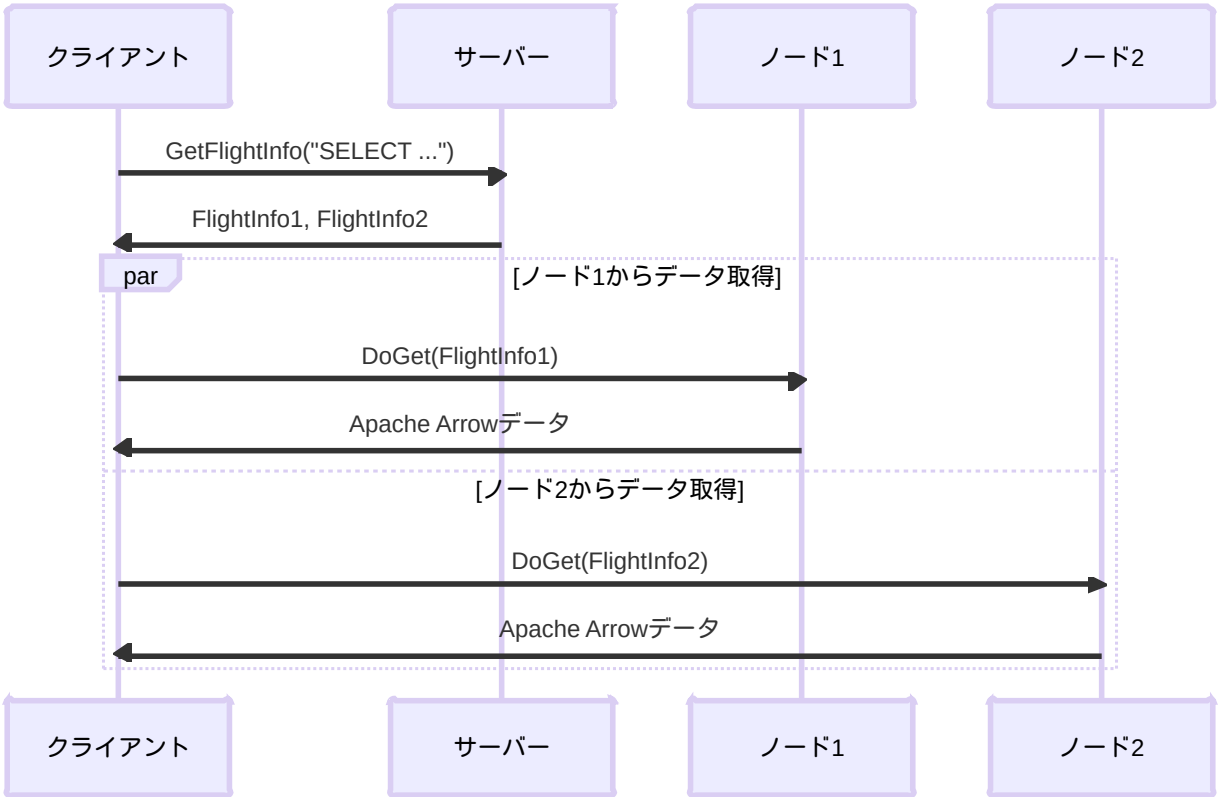
Apache Arrow Flight SQL

- ✓ プロトコル
- ✓ Apache Arrow（列指向）データに最適化
  - ✓ すでにArrowベースなデータ処理システムになじむ
  - ✓ 例：Polars, pandas, PG-Strom, ...
- ✓ 複数ノードから並列ダウンロード可
  - ✓ postgres_fdwを使ったPostgreSQLクラスターから並列ダウンロードとか？

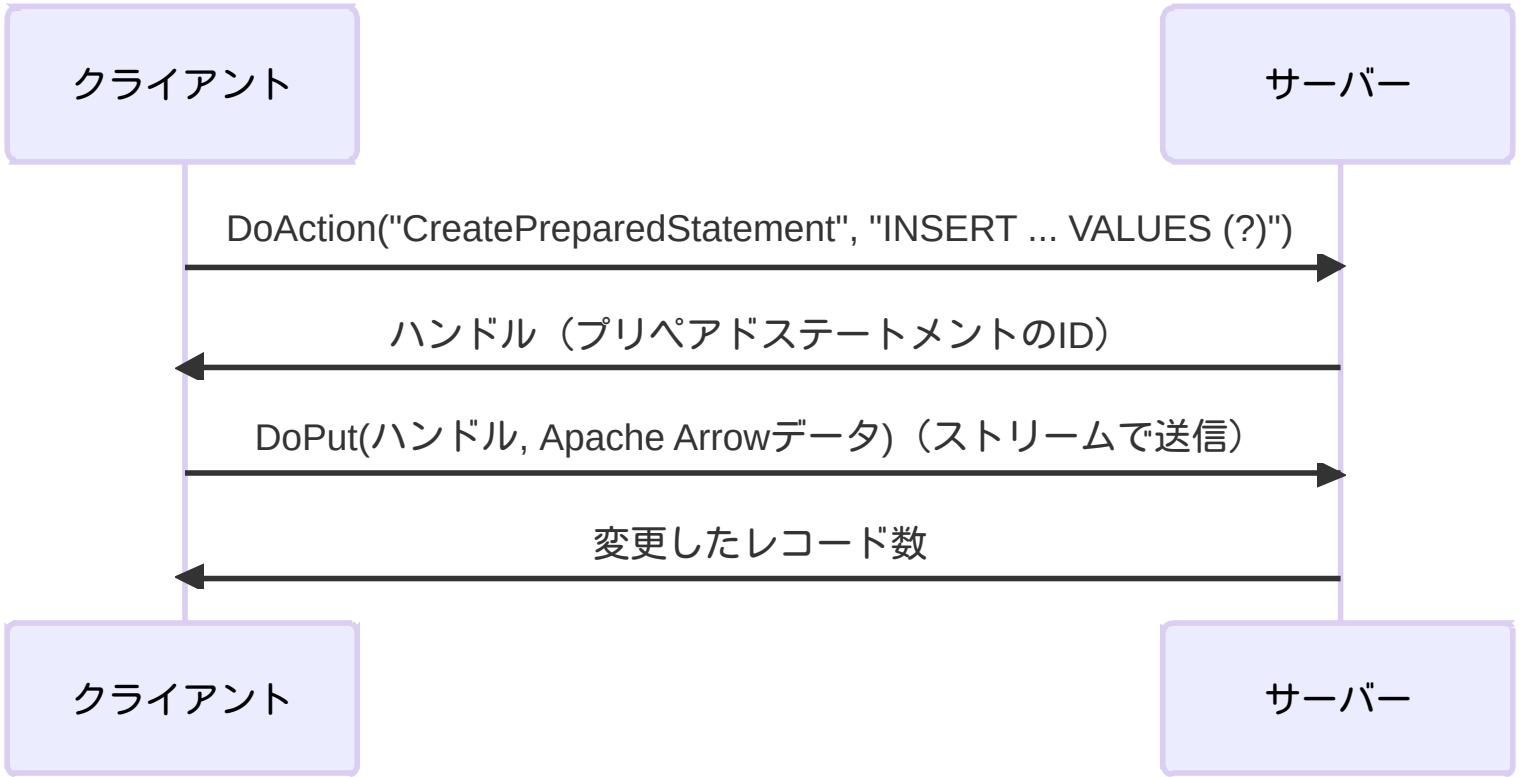
SELECT


クライアント
サーバー
GetFlightInfo("SELECT ...")
FlightInfo（データ取得のための情報）
DoGet(FlightInfo)
Apache Arrowデータ（ストリームで送信）
クライアント
サーバー

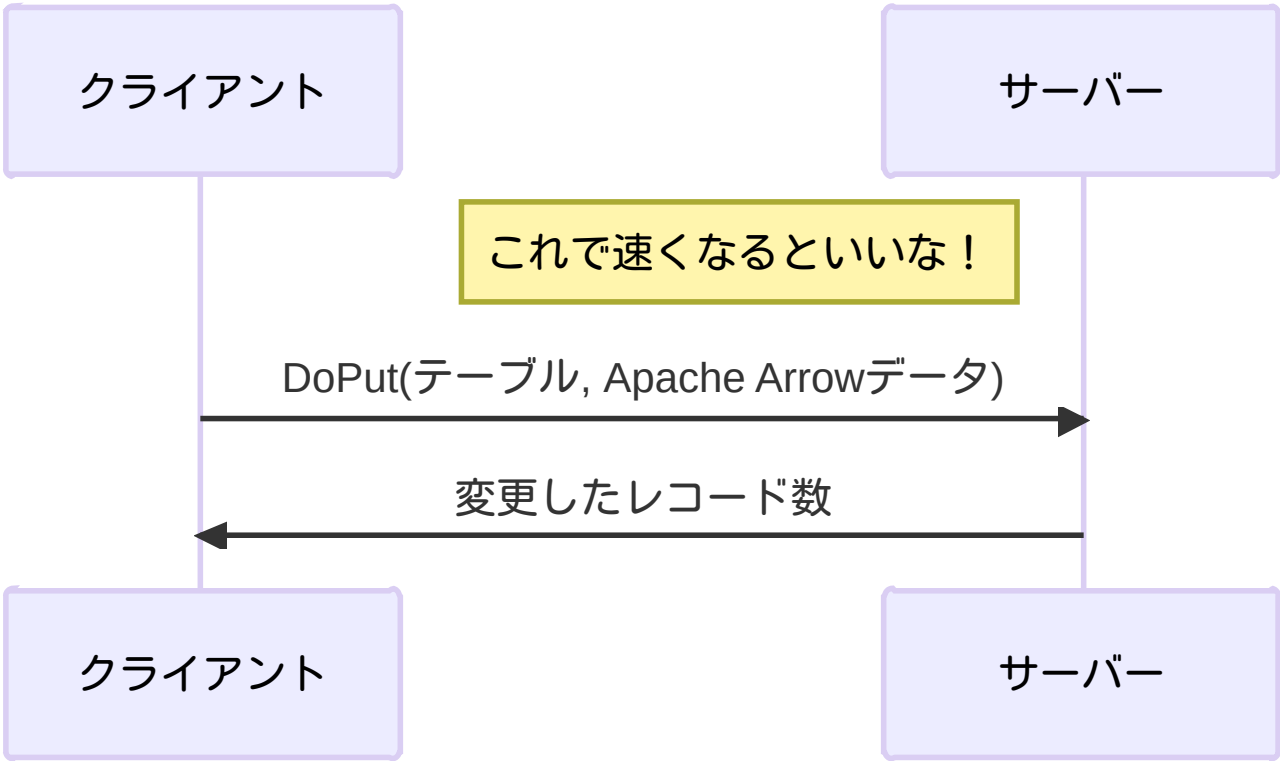
SELECT：並列ダウンロード



INSERT


クライアント
サーバー
DoAction("CreatePreparedStatement", "INSERT ... VALUES (?)")
ハンドル（プリペアドステートメントのID）
DoPut(ハンドル, Apache Arrowデータ)（ストリームで送信）
変更したレコード数
クライアント
サーバー

INSERT：今後


クライアント
サーバー
これで速くなるといいな！
DoPut(テーブル, Apache Arrowデータ)
変更したレコード数
クライアント
サーバー

開発者向けの話：認証

- ✓ Apache Arrow Flight SQL
- ✓ **認証**
- ✓ 追加プロトコル対応

認証

- ✓ PGプロトコルを使わない→自前実装が必要
  - ✓ PGの認証実装はPGプロトコルと蜜結合
- ✓ でも、認証情報の二重管理はしたくない！
  - ✓ →PostgreSQLの認証情報で認証

PostgreSQLの認証情報

- ✓ pg_hba.conf
  - ✓ ホストベースの認証情報
  - ✓ 接続可の接続元は？SSLは必須？とか
- ✓ CREATE ROLE
  - ✓ PostgreSQL内にユーザーを作成
  - ✓ パスワードはPostgreSQLで管理または別途管理

対応済み認証方法

- ✓ trust認証
  - ✓ 接続可のホストから接続されればOK
  - ✓ 本番環境では使わないこと！
- ✓ パスワード認証
  - ✓ PostgreSQL内のパスワードを使って認証
  - ✓ SSLを有効にすること！

その他の認証方法

- ✓ scram-sha-256
  - ✓ チャレンジレスポンスが必要なので未対応
  - ✓ 今はHTTP/2のauthorization: Basicを使っている
- ✓ 証明書認証（mTLS）
  （Mutual Transport Layer Security・相互TLS）
  - ✓ サーバーがクライアントの証明書を検証
  - ✓ 簡単に実装できるはずなので実装予定
    （実装にはSSL対応が必須）

SSL対応

- ✓ 実装済み
  - ✓ Apache Arrow Flight SQLはSSL対応
  - ✓ mTLSにも対応
- ✓ SSL対応に必要なもの
  - ✓ 証明書・証明書の秘密鍵・認証局
  - ✓ すべてPostgreSQLで設定可
  - ✓ →AFSは↑を使っている
    （二重管理を避ける！）

開発者向けの話：認証

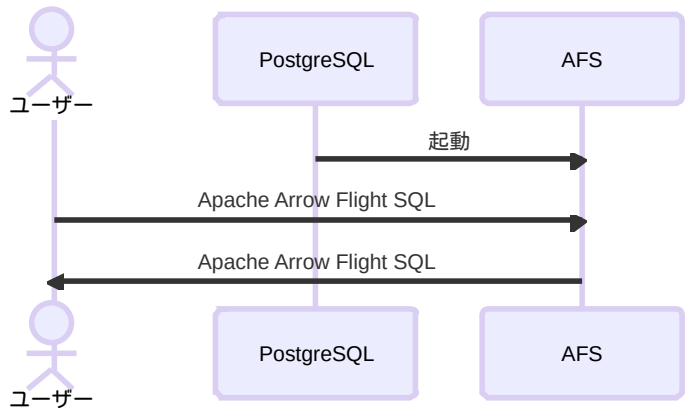
- ✓ Apache Arrow Flight SQL
- ✓ 認証
- ✓ **追加プロトコル対応**

追加プロトコル対応

- PostgreSQLは別プロトコルでの接続を想定していない
  - 拡張ポイントがたくさんあるPostgreSQLだがプロトコル追加用の拡張ポイントはない
- ソケットをlistenしないといけない
  - PostgreSQLはシングルスレッドなので既存プロセスでやるとブロックする

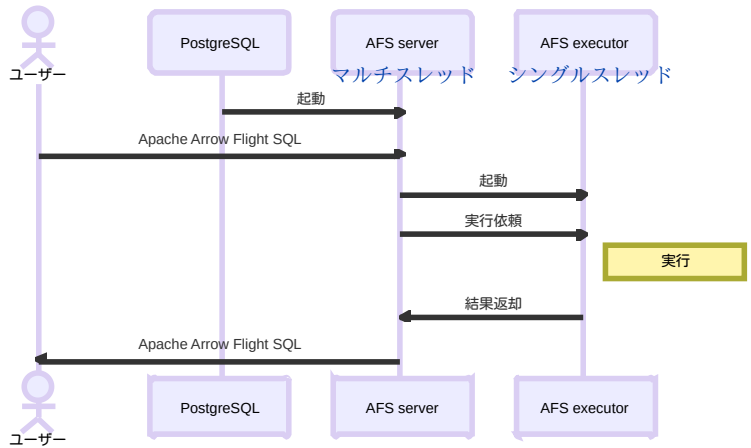
バックグランドワーカー

- PG管理の別プロセスを起動する仕組み
  - ここでlistenすれば既存処理をブロックしない



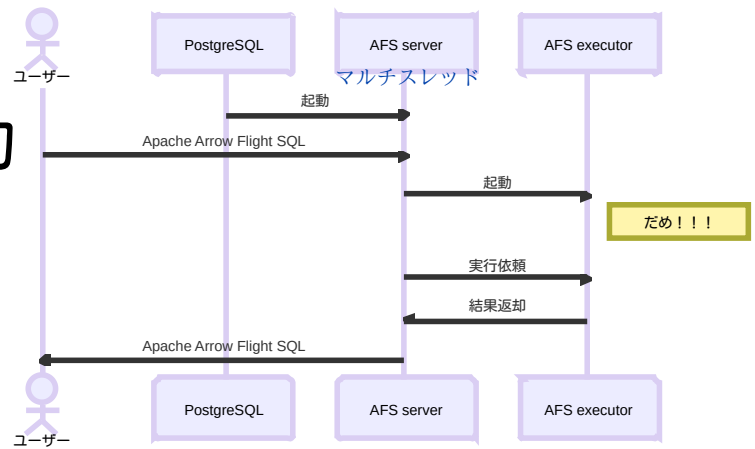
接続

- ✓ ×listenするプロセスでSQL実行
  - ✓ listenするプロセスはマルチスレッド
  - ✓ PGはマルチスレッド非対応→PGのAPIを呼べない
- ✓ プロセスを分離：
  - ✓ listenするプロセス：server
  - ✓ 各接続を処理するプロセス：executor（シングルスレッド）



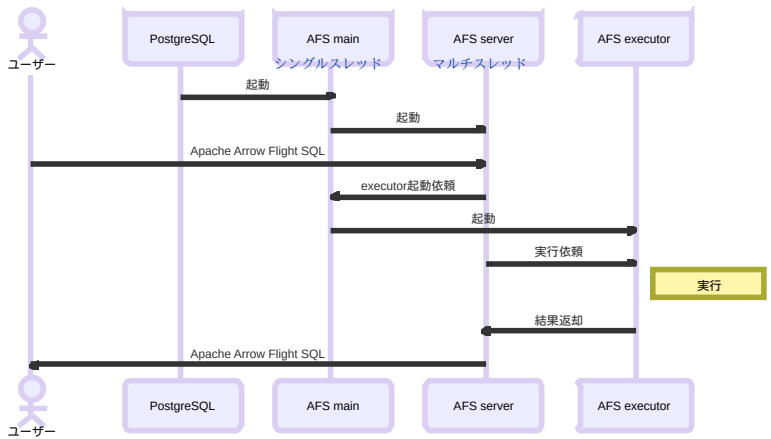
executorの起動方法

- ✓ ×マルチスレッド下でfork
  - ✓ serverはマルチスレッド
  - ✓ ×serverからexecutorを起動
- ✓ この構成は危険



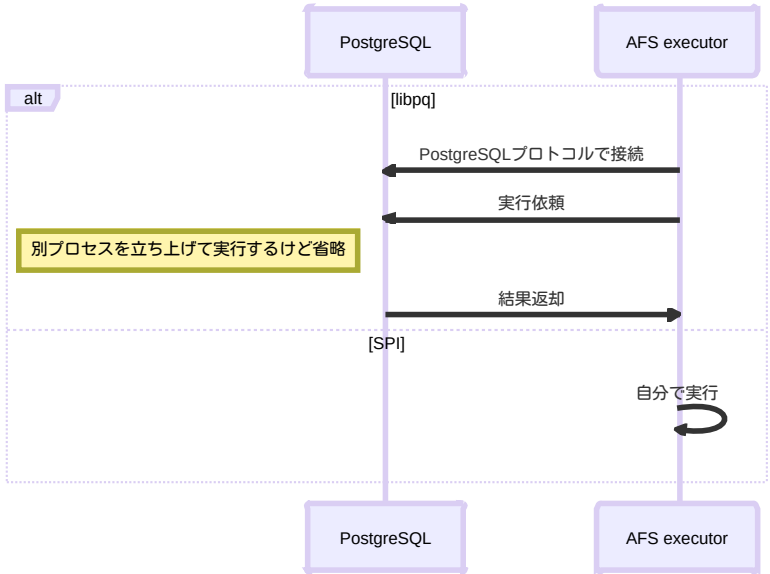
安全なexecutorの起動方法

- ✓ executor起動用プロセスを追加
  - ✓ main：シングルスレッド
- ✓ serverはmainに起動依頼
  - ✓ mainからexecutorを起動



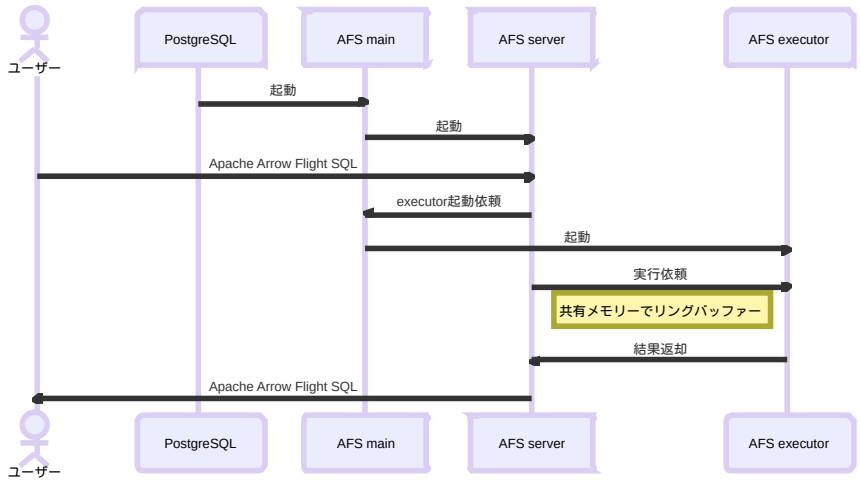
executorでの実行

- やり方は2つ
  - libpq：PostgreSQLに接続して実行
  - SPI：プロセス内で実行
    （Server Programming Interface）
- AFSはSPI
  - libpqだと意味がない
    （PostgreSQLプロトコルを使ってしまう）



プロセス間通信

- ✓ serverとexecutorで
  Apache Arrowデータのやりとりが必要
- ✓ 通信方法
  - ✓ 共有メモリーで
    リングバッファー
  - ✓ ナイーブな自前実装
    （高速化の余地あり）
  - ✓ shm_mqでいいかも…



開発者向けの話のまとめ

- ✓ 設計判断も含めて全体の構成を紹介
- ✓ 改良案も含めて課題も紹介

↓

一緒に開発しよう！

全体のまとめ

- ✓ 課題：大量データの読み書きが遅い
- ✓ 解決方法：
  - ✓ Apache Arrowフォーマット/Flight SQL
- ✓ 現状：✓数値の読み込みは速い
- ✓ 今後：
  - ✓ このアプローチの改良
  - ✓ COPYのApache Arrowフォーマット対応

サポート

- コミュニティー：
  - https://github.com/apache/arrow-flight-sql-postgresql/issues
  - https://arrow.apache.org/community/#mailing-lists
- クリアコードによる有償サポート：
  - https://www.clear-code.com/contact/
  - PGroongaのサポートも可
    PGroonga：PostgreSQLに高速日本語全文検索機能を追加