CTC Engineer's Insight # 5

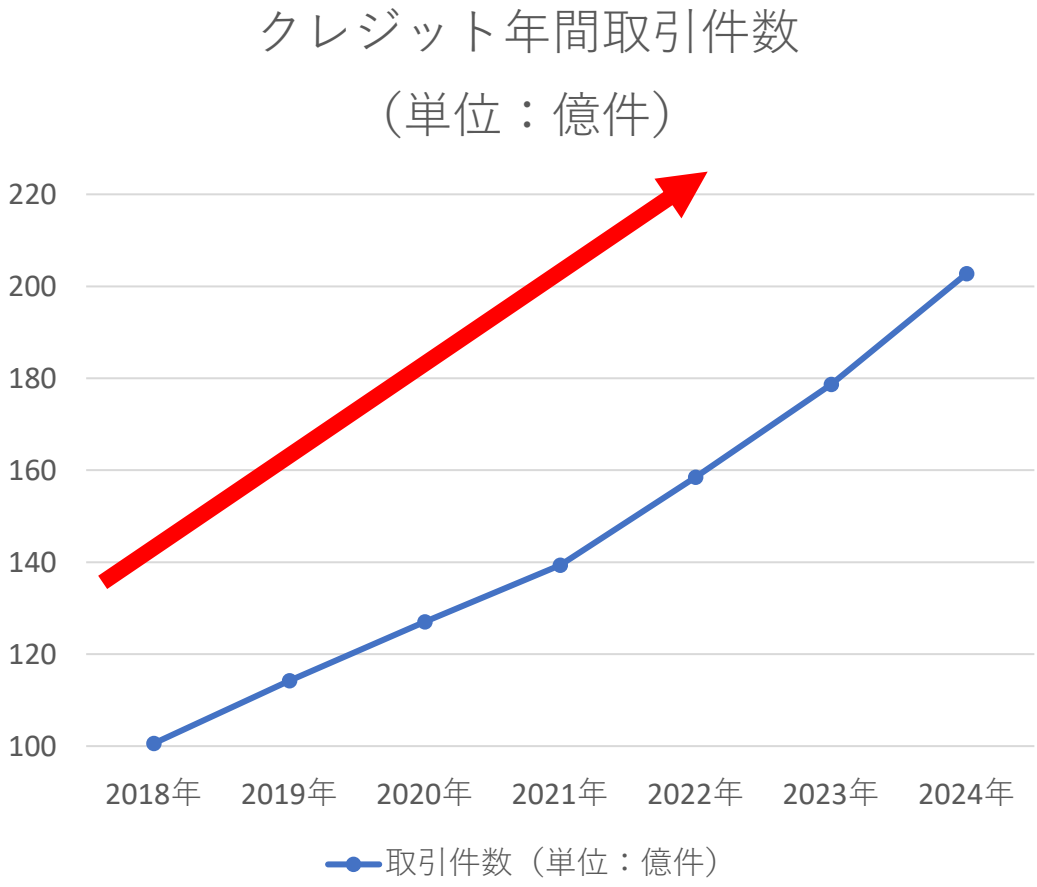
Ｋａｆｋａを用いた構成改善による性能・スケーラビリティ改善

伊藤忠テクノソリューションズ株式会社

１．ＰＪ背景と課題

- ネットショッピングの利用増加や
  非接触決済の普及により、
  クレジットカードの利用は
  年々増加傾向にあり、
  取引件数（売上件数）は
  ６年間で２倍以上の件数増となっている。

- 取引量増加により各カード会社は
  １日で数百万、数千万件の
  取引データを処理する必要があり、
  処理性能が追い付かなくなってきている。



出典：日本クレジット協会　日本のクレジット統計　より
https://www.j-credit.or.jp/information/statistics/

１．ＰＪ背景と課題

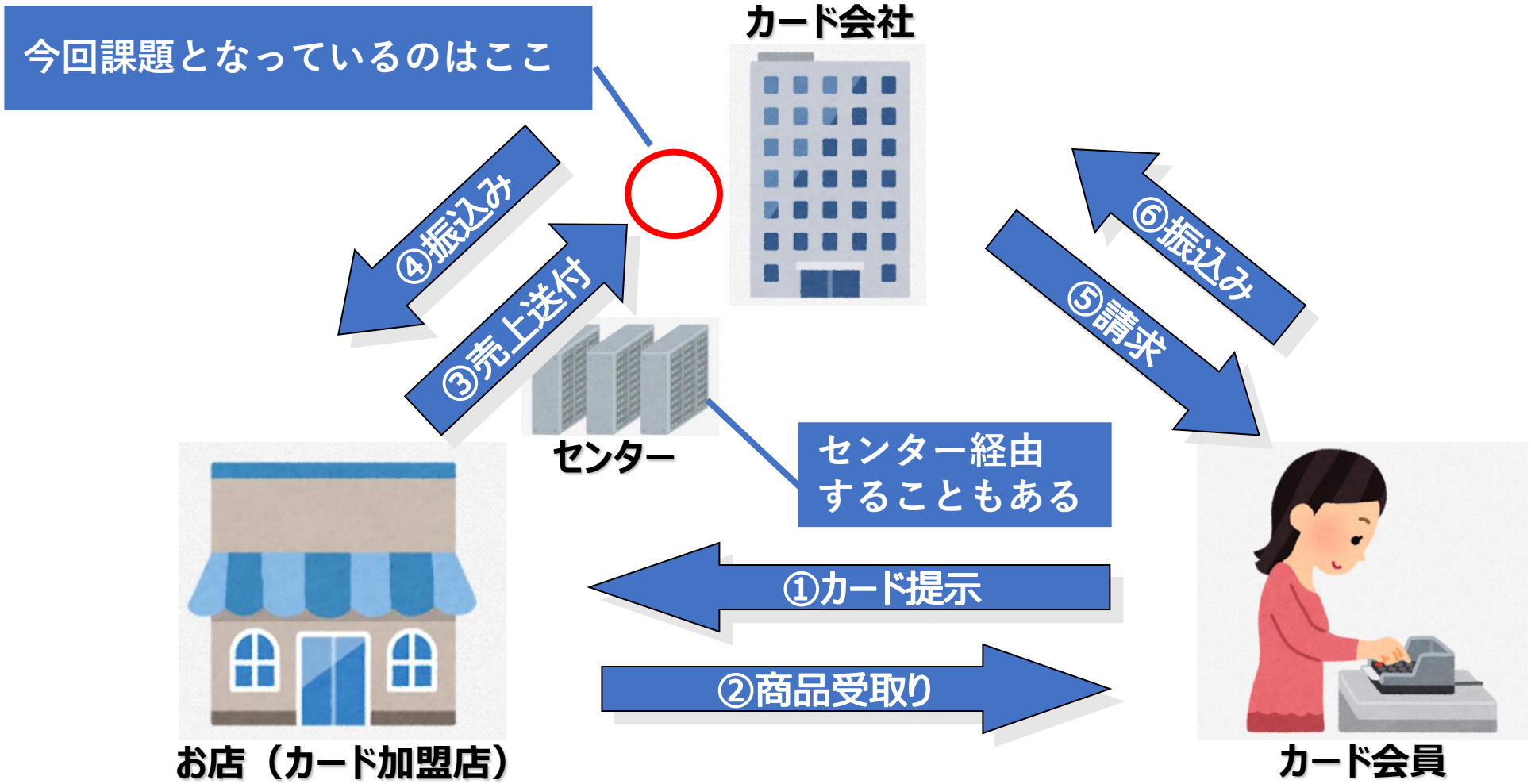
- 某カード会社様でも取引件数増加により、各加盟店からの売上データに対する基幹システムへの登録処理の時間が年々増加してきており、このままいくと数年後には時限を超過する見込みであった。

- アプリ・SQLチューニングなどの小規模なパフォーマンス改善を実施しているが、この対応では限界があるため大幅なシステム改修を行い性能改善したいとのご要望があった。

- 本件について、データアクセスの最適化、不要ロジックの削除、処理構成変更等の対応を実施したが、本日は処理構成変更（Kafka導入）について、ご紹介する。

１．ＰＪ背景と課題

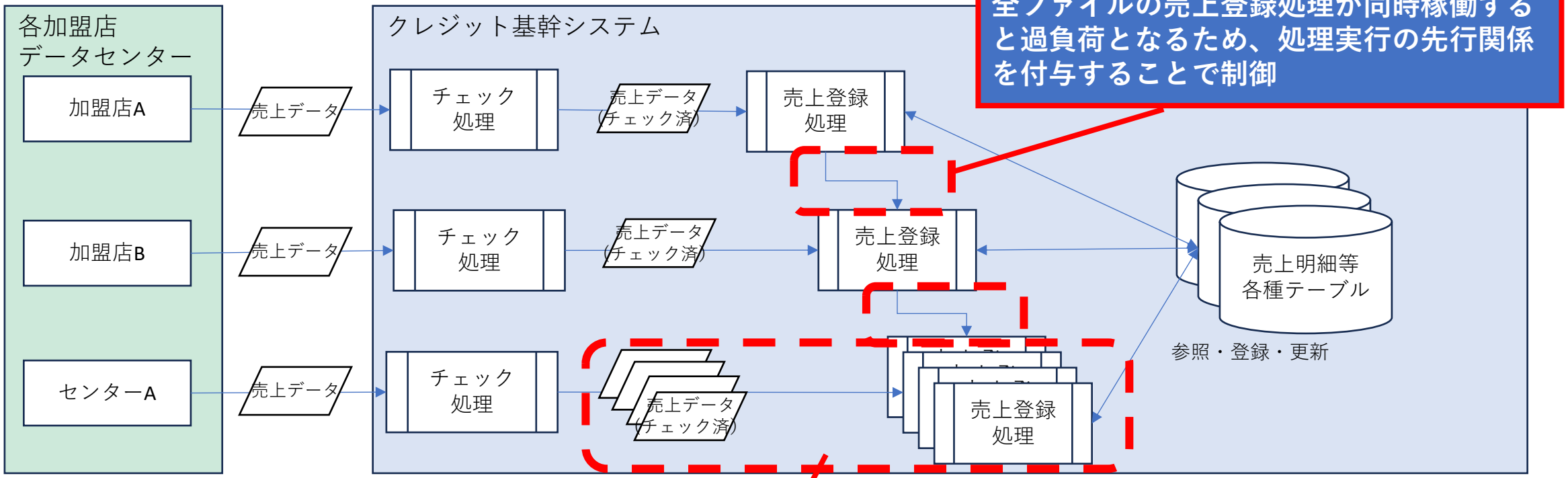
前提知識：カード利用の流れ



１．ＰＪ背景と課題

性能改善対象の処理概要は下図の通り、各加盟店からファイルで受領した
売上データをファイルごとにチェックし、チェックが全件完了したら、売上登録を実施する。

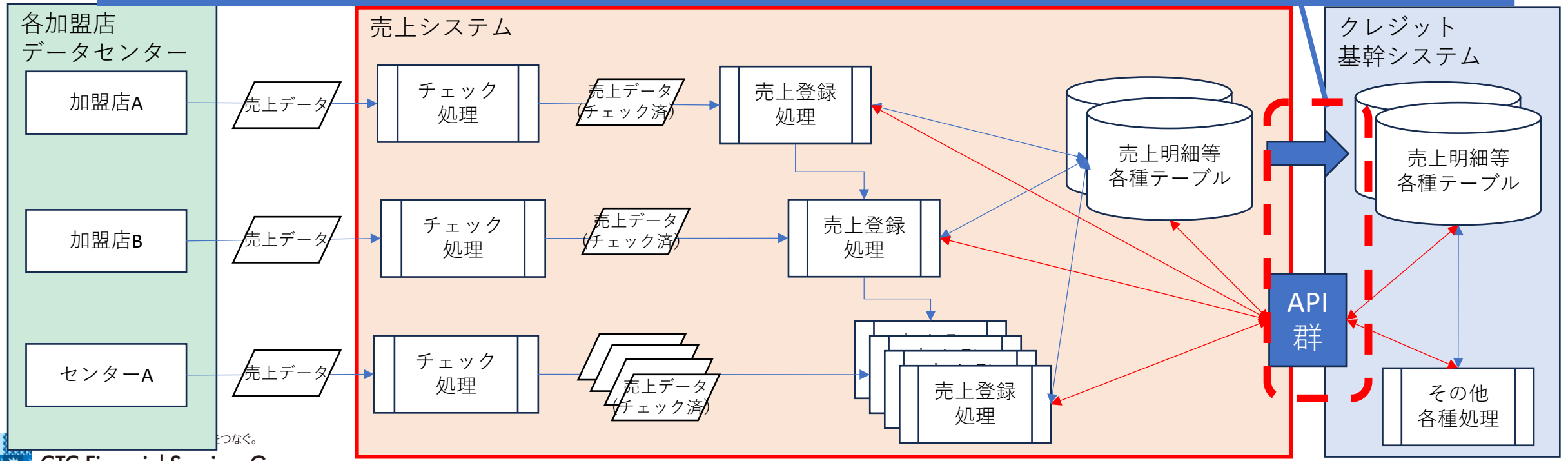
※下図は3ファイル分のみ記載しているが、実際はもっと多くの加盟店･データセンター等から売上データを受領し、同様の仕組みで後続処理として存在



2．対策検討

システムのあるべき姿も検討し、対策として、売上機能を別システムとして切り出し、
基幹システムとは別のリソースを使用することで、性能改善を図ることを検討。
（基幹システムの全体負荷軽減や、マイクロサービス化にも繋がる）

＜ポイント＞
各種DBテーブルについては整理の上、
①新規システム上で登録した内容を基幹システム側へ
レプリケーションや一括ロード等の高速な方法によるデータ同期を実施
②単純同期が難しいものはAPIを作成し、API経由で別システムのDBテーブルを照会・更新



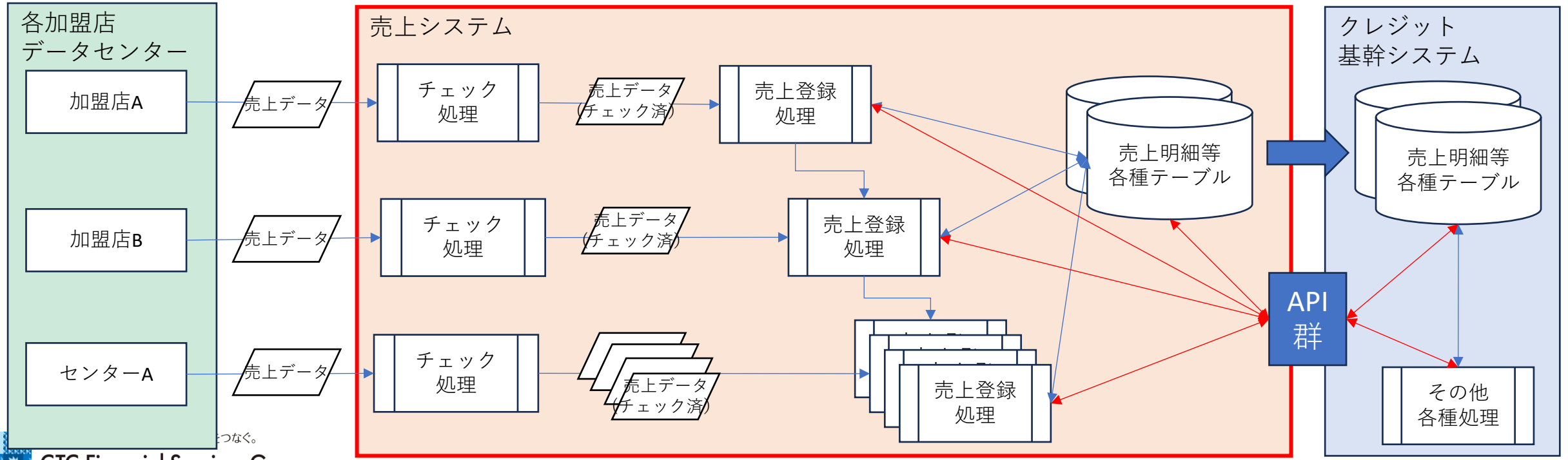
CTC Financial Services Group

2．対策検討

前ページの方針で検討したが、DBテーブルへのアクセスに課題が多く、断念。

- 他処理からの参照・更新等も膨大で、レプリケーションなどの単純同期で対応するにはテーブル構成の見直しから必要。
- APIでの照会・更新とするにしても影響範囲が広く、対応が必要な処理が膨大となり、レイテンシー等による処理時間増の懸念や、APIの大量開発が必要。

⇒コスト高、リスク高と想定される。



CTC Financial Services Group

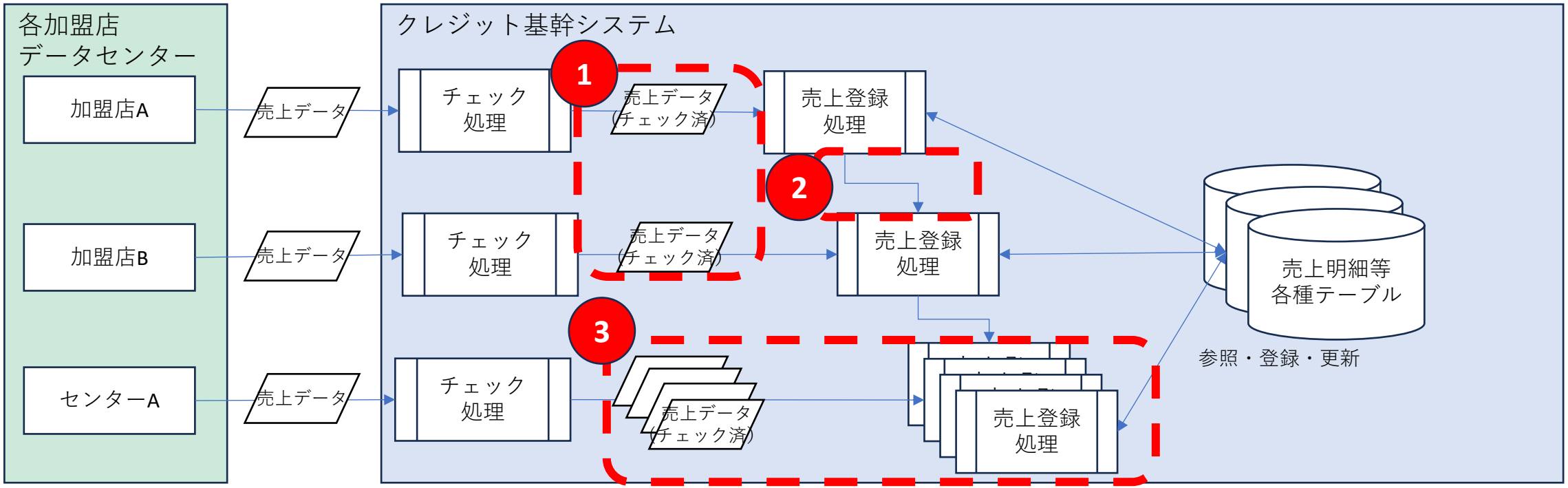
2．対策検討

現行処理の課題を分析。以下、改善できそうなポイントあり。

①チェック処理が終わりきってから売上登録処理が稼働するため、チェックが完了するまで待ち時間が発生する。

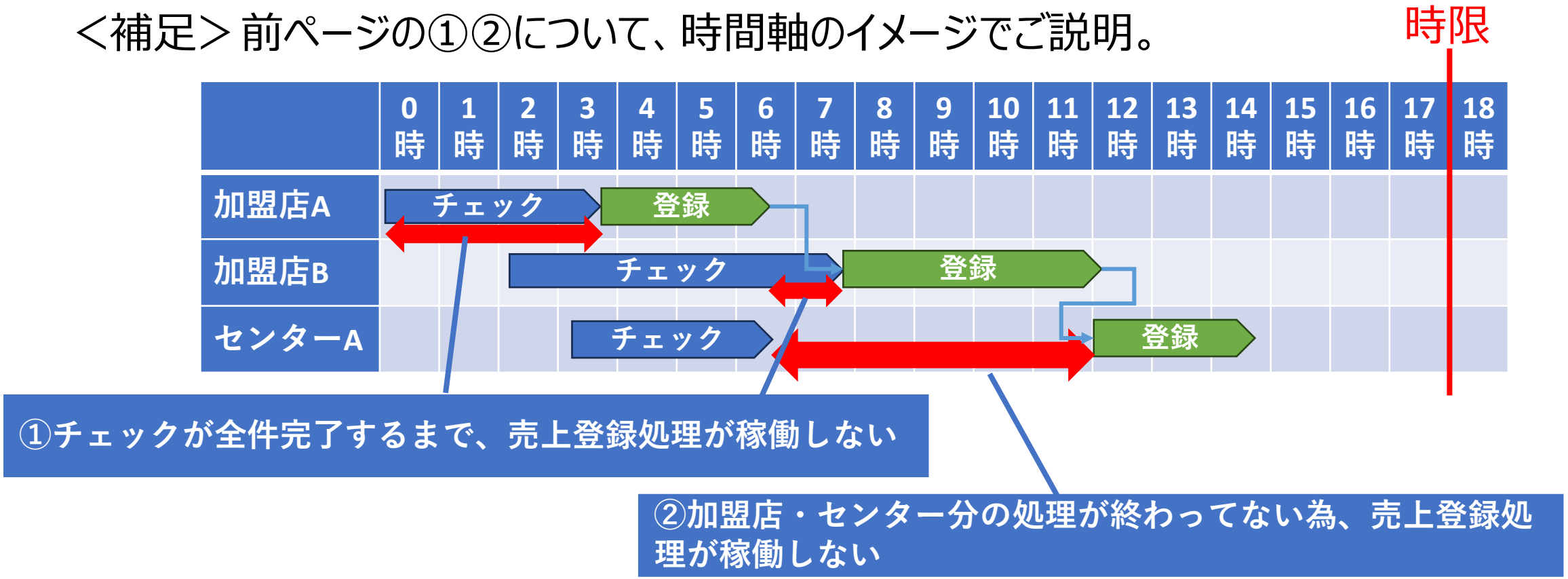
②前の加盟店・センター分の処理が終わってからでないと、次の加盟店・センター分が処理開始できない。

③売上登録処理を高速化するため、実際には多重化しているが、加盟店・センター毎に多重化対応を行う必要があり、また件数に変動があった場合は追加対応が必要となる。



2．対策検討

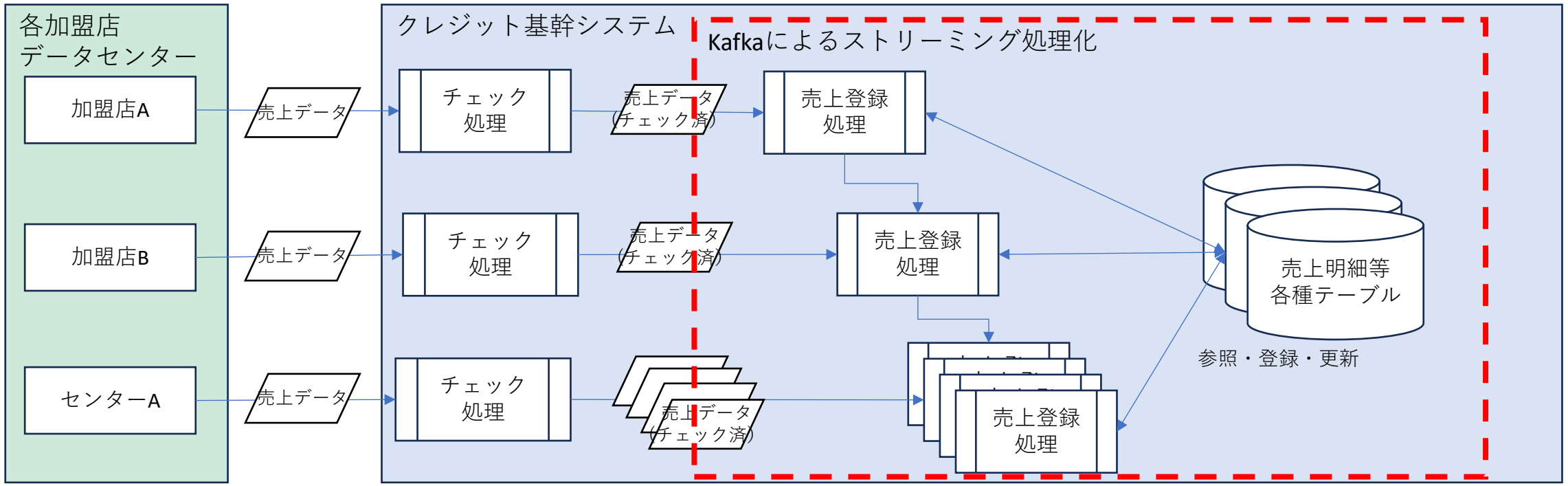
＜補足＞前ページの①②について、時間軸のイメージでご説明。



なお、データ件数は日によって変動し、それに伴い処理時間も毎日変動するため、
現状の仕組みで、待ち時間が発生しないような処理構成とするのは困難。

2．対策検討

前述課題を分析した結果、売上登録処理にKafkaを導入して処理を
脱バッチし、ストリーミング化することで課題解決する方針とした。

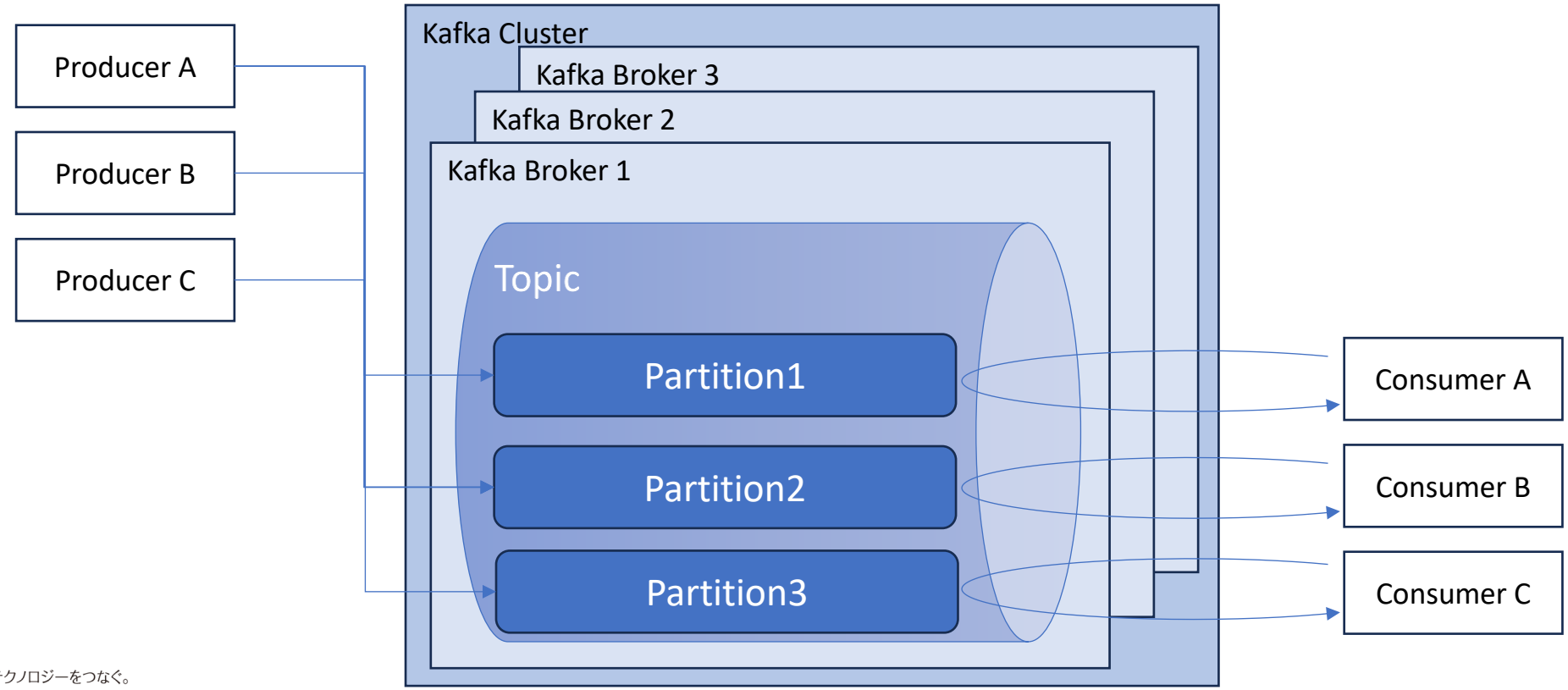


3．Apatch Kafkaとは

分散メッセージキューで、システム間のデータの受け渡しを仲介し、データを一時的に保持（キューイング）するミドルウェア。システム間の通信（処理）を非同期化することで、データ流量の急激な増加によるシステムの負荷上昇を抑制する。

（アーキテクトではないのでKafka詳細仕様に関するご質問はご容赦ください）

＜Kafka概略図＞



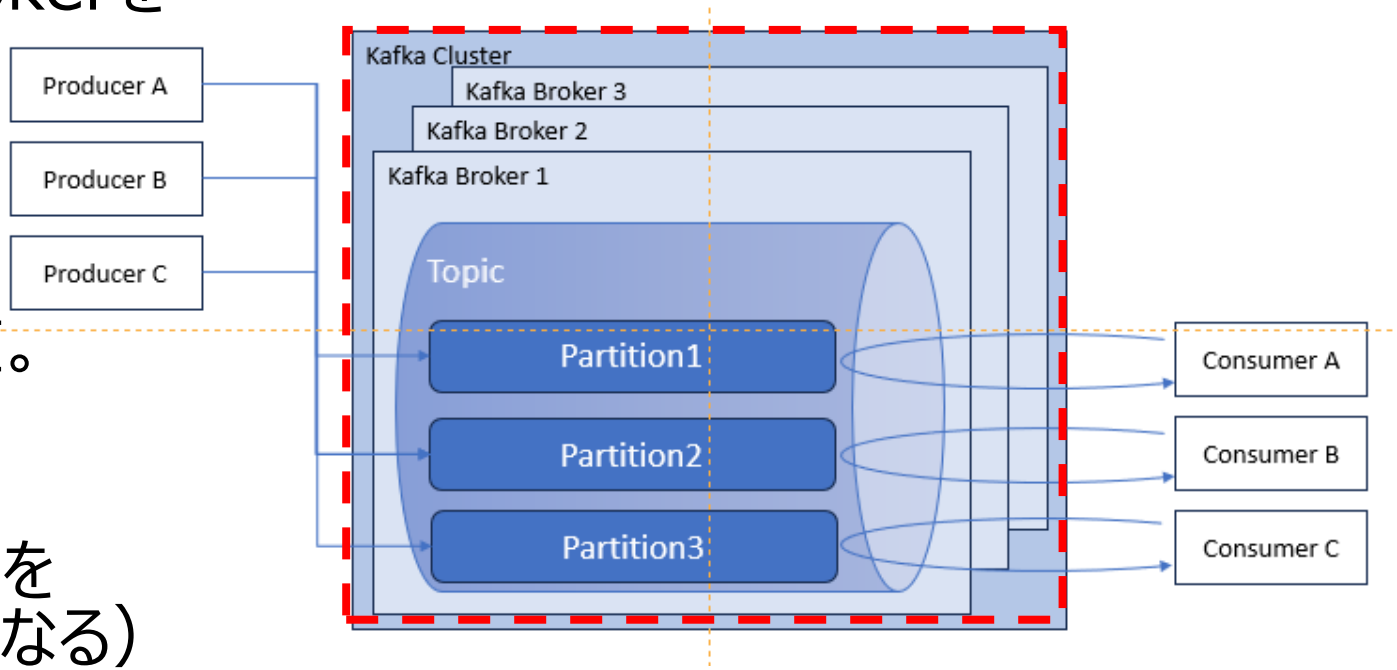
3．Apatch Kafkaとは

- Kafka Broker
  Kafkaが稼働するサーバ

- Kakfa Cluster
  可用性/信頼性向上のためにBrokerを
  複数台で構成したもの

- Topic
  データ（メッセージ）の保存先。
  データベースのテーブルに近い存在。

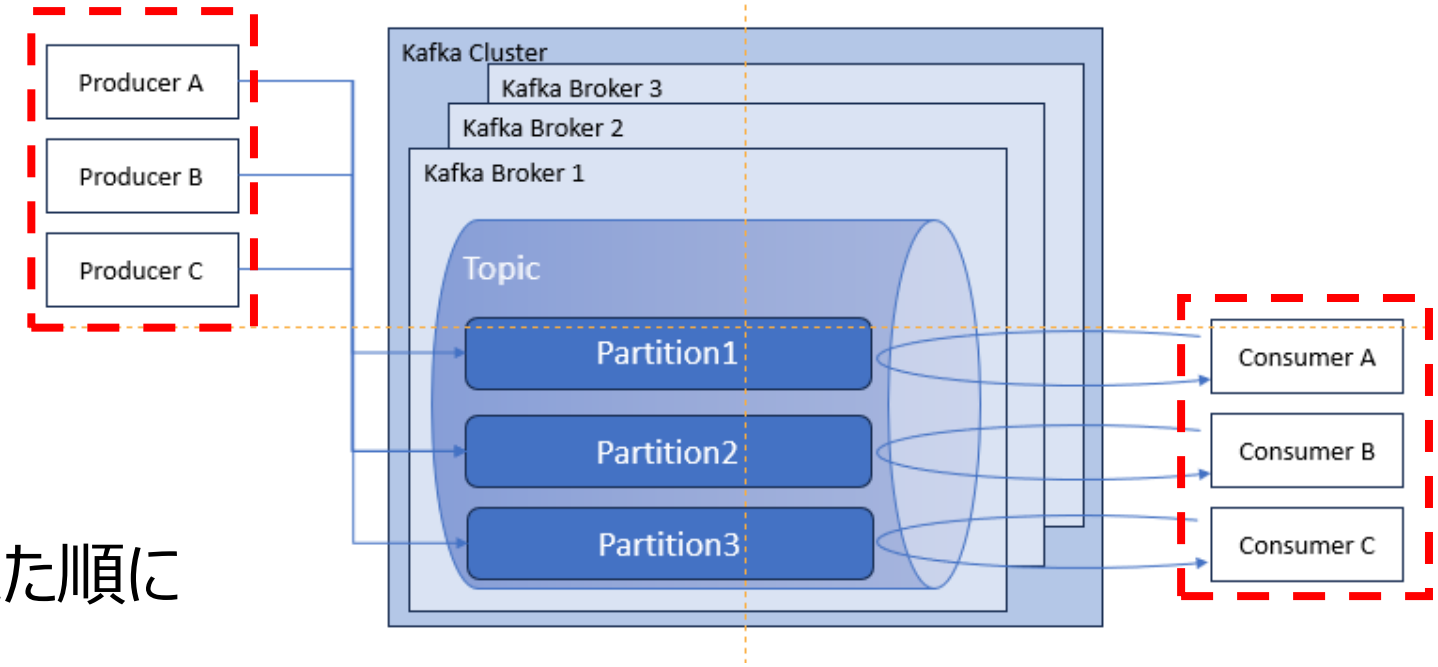
- Partition
  Topicを複数に分割してメッセージを
  分散させる（並列度のファクターとなる）



3．Apatch Kafkaとは

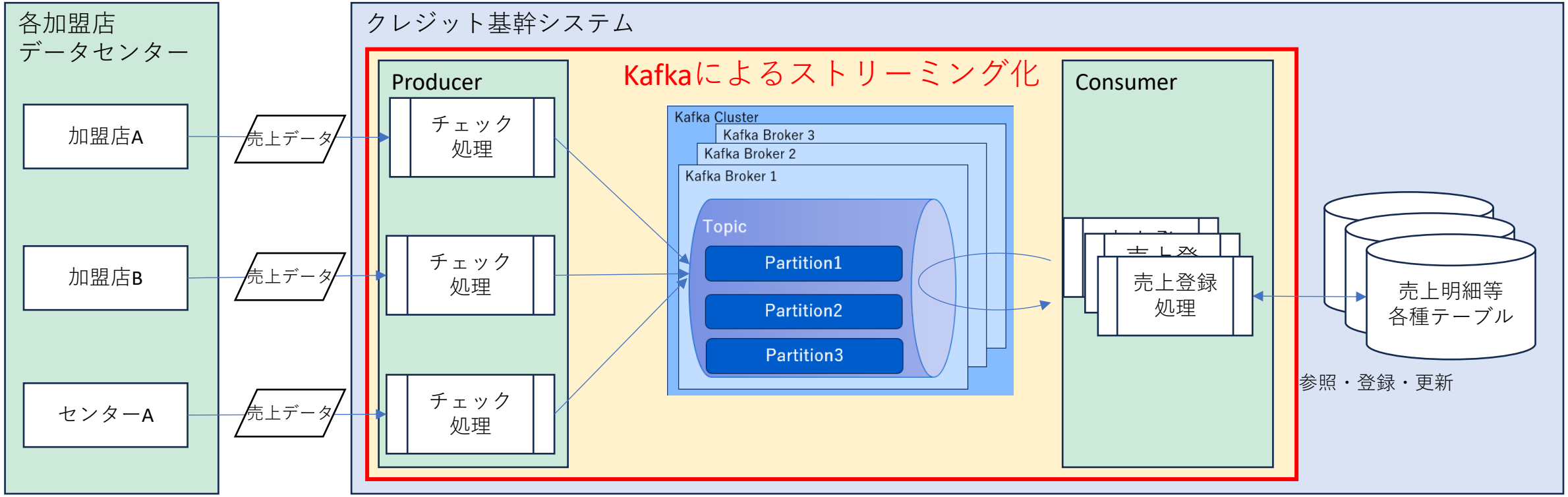
- Producer
  メッセージを作成し、Brokerの特定のTopicを指定してメッセージを送信するアプリケーション。（Producer数とPartition数は関連性なし）

- Consumer
  Brokerの特定のTopicを
  指定してメッセージを取得し、
  処理を行うアプリケーション。
  基本的にはメッセージを
  一定間隔で取得しつづける
  常駐型処理となる。
  PartitionとConsumerは
  1:1またはN:1の関係となる。
  これによりPartitionに登録された順に
  メッセージが取得される。



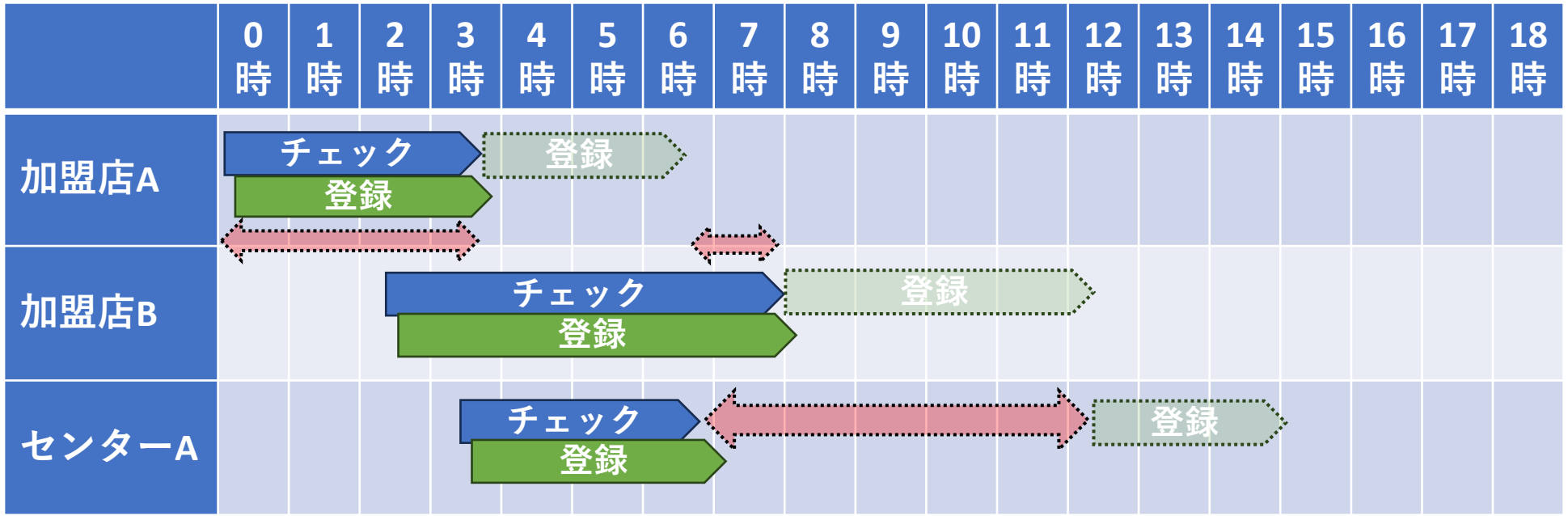
４．Kafkaを利用した新処理概要

Kafka導入により以下の構成にて構築。
・consumerとパーティションを増やせば多重度を変更でき、スケールアウトしやすい。
・加盟店・センター毎に多重化対応を行う必要がなく、今後の保守開発工数削減。
・加盟店・センターの日単位での件数変動も、リソースの無駄なく処理可能。

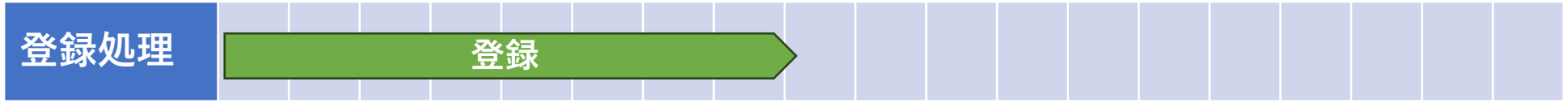


４．Kafkaを利用した新処理概要

待ち時間が発生していた⇔部分の無駄な待ち時間が解消。
全体としての処理時間が大幅削減。



なお、実際は売上登録処理は、以下のように加盟店・センターにかかわらず、共通の常駐処理として稼働。



５．まとめ

- あるべき姿を意識して、解決策を考える
- とはいえ、現実的ではない案に固執せず、
  コストが高い、リスクが高い等の場合は、
  早い段階で軌道修正する。
- 課題がどこにあるのかきちんと分析することが大切。
- アプリケーションチューニングだけでは限界がある。
  新アーキテクチャの導入による解決策も検討する。

無限の未来と、
幾千のテクノロジーをつなぐ。

CTC Financial Services Group

CTC
Challenging Tomorrow's Changes