データセンターの効率的な資源活用のためのデータ収集・照会システムの設計

株式会社ネットワーク応用通信研究所　前田　修吾

2014年11月20日

NaCl

NaCl

本日のテーマ

- **データセンターの効率的な資源活用のためのデータ収集・照会システムの設計**
  - 時系列データを効率的に扱うための設計

NaCl

システムの目的

- データセンター内の機器のセンサーなどからデータを取集し、その情報を元に機器の制御を行うことで、電力消費量を抑制する
- 収集したデータの中長期的なトレンドを分析し、上記の制御を効率的に行うために活用する

NaCl

アプリケーションの例

- 直近の値に応じて機器を制御するアプリケーション
- 値の推移をグラフで表示するアプリケーション
- 中長期的なトレンドを分析するためのバッチ処理

NaCl

収集する主なデータ

- **時刻**
  - データが発生した時刻
  - Epochからの経過マイクロ秒数（精度は変更の可能性あり）
  - 整数値
- **データソースID**
  - データの発生源を表すID
  - 整数値
- **値**
  - データソースの種別によって意味が異なる値
    - 温度、湿度、CPU使用率、メモリ使用量、消費電力など
  - 任意の整数値

NaCl

性能上の課題

- **データ量**
  - データソース数
    - 1コンテナあたり20,000点程度
  - データの取得間隔と保存期間
    - 取得間隔を5秒、保存期間を1年とすると
      1コンテナあたり約126G records
- **レイテンシ**
  - データの発生～データの登録にかかるレイテンシ
  - 問合せ～応答にかかるレイテンシ
    - 数百ミリ秒～数秒
- **同時アクセス数**

NaCl

Impala/BigQueryの採用

- **Impala**
  - Hadoop上で動作するSQLクエリエンジン
  - MapReduceの代りに独自の仕組みでクエリを分散実行する
  - MapReduceを使用したSQLクエリエンジンであるHiveに比べて、メモリ使用量が大きい代りに高速
- **BigQuery**
  - Googleが提供するビッグデータ分析サービス
  - カラム型データストアとツリー構造のサーバ構成によりクエリを高速処理

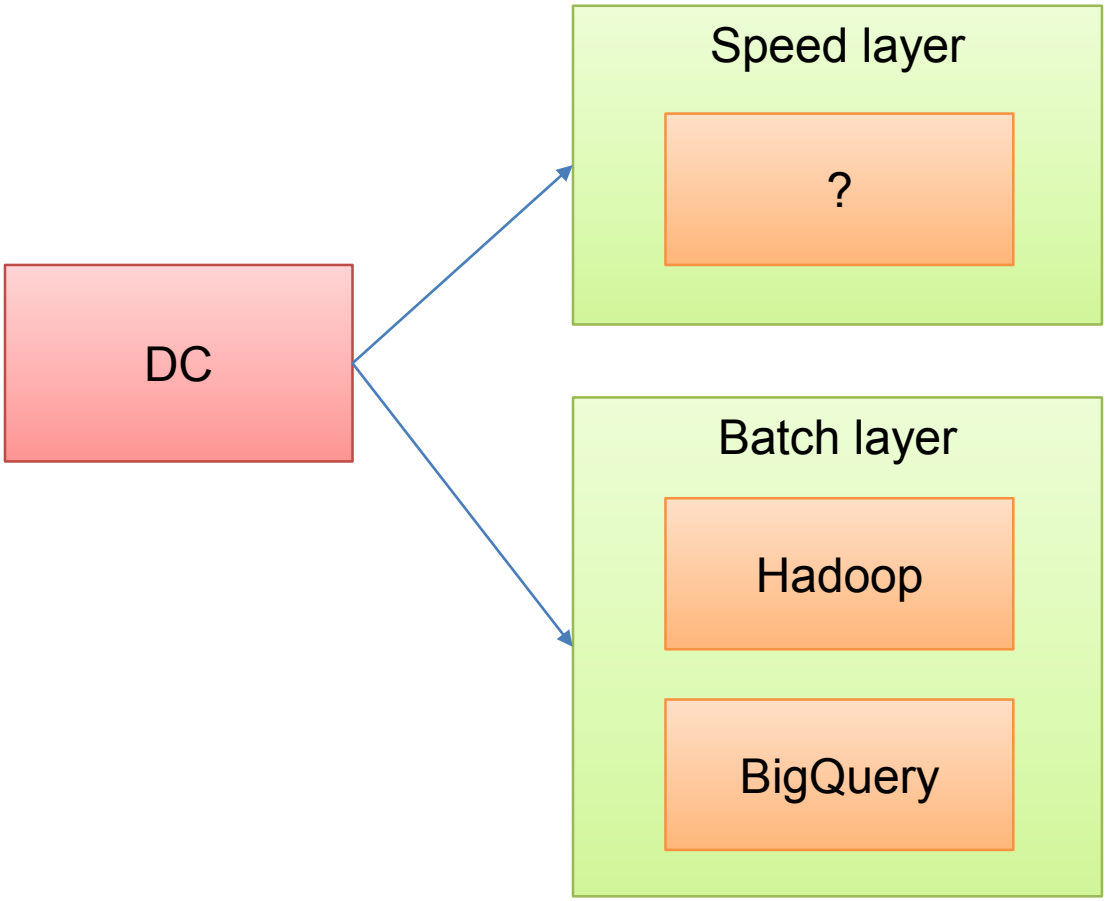
NaCl

データ登録のレイテンシ

- **Imapala**
  - CSVをHDFSに書き込んだ上でParquetフォーマットに変換
    - レイテンシが高い
- **BigQuery**
  - Google Cloud StorageにアップロードしたCSV/JSONファイルをBigQueryにロード
    - レイテンシが高い
  - Streaming Insertで逐次登録
    - レイテンシが低い
    - quota: 10,000 ～ 100,000 rows per sec
    - コストが高い: $0.01 per 100,000 rows
      - 100,000 rows per secで一日分 = $864 ≒ 10万円

NaCl

レイヤーの分離



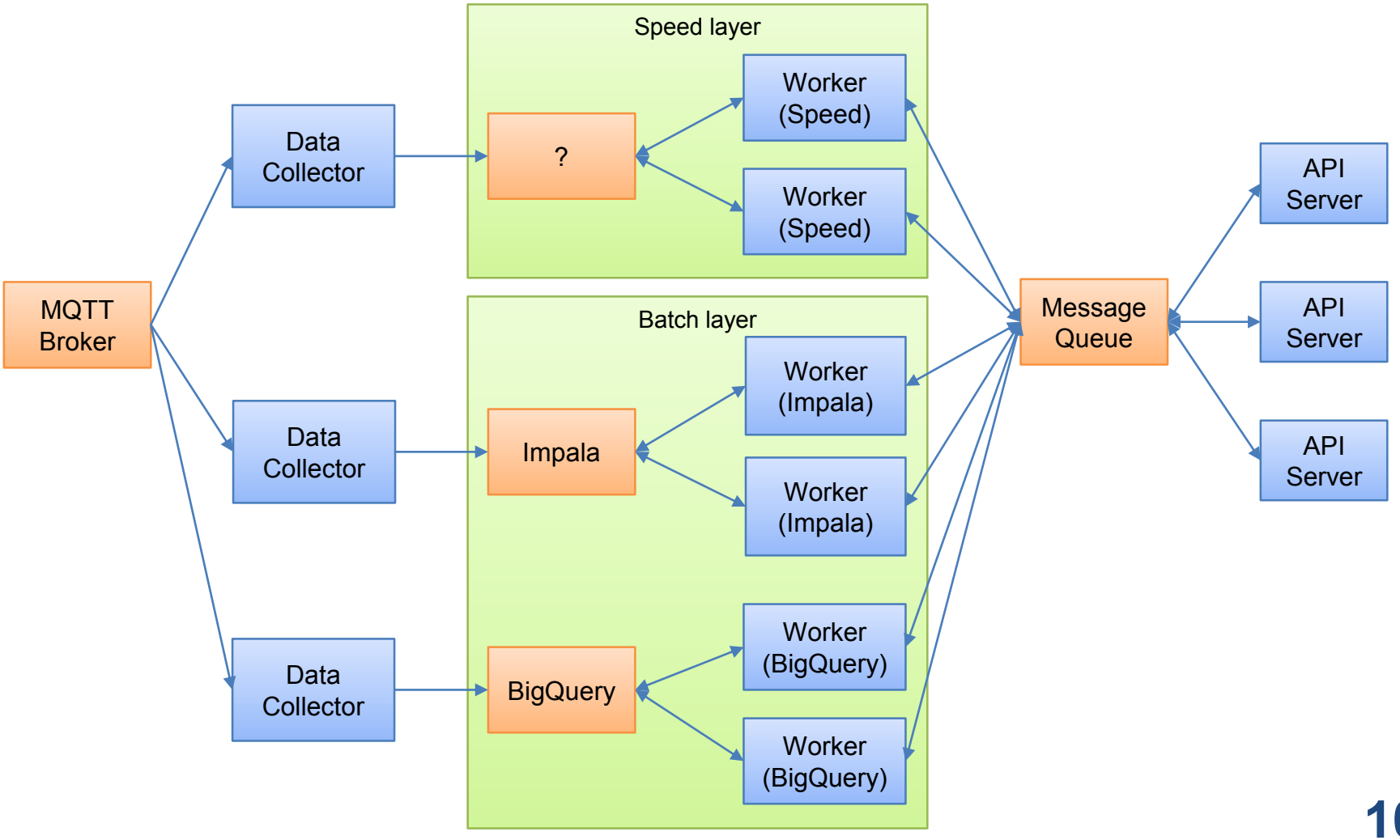
NaCl

各レイヤーの役割

- **Speed layer**
  - 機器の制御に利用するようなデータを扱う
  - データの保存期間は短い
  - 低レイテンシ・少データ
- **Batch layer**
  - 分析に利用するため全データを扱う
    - Speed layerのデータを含む
  - データの保存期間は長い
  - 高レイテンシ・多データ
- **Speed layerのデータは一時的なもの**
  - 時間が経てばBatch layerにすべてのデータが格納される

NaCl

システム構成


Speed layer
Worker (Speed)
?
Worker (Speed)
Data Collector
MQTT Broker
Batch layer
Worker (Impala)
Data Collector
Impala
Worker (Impala)
Worker (BigQuery)
Data Collector
BigQuery
Worker (BigQuery)
Message Queue
API Server
API Server
API Server

NaCl

Speed layerのデータストア

- RDBMS or KVS?

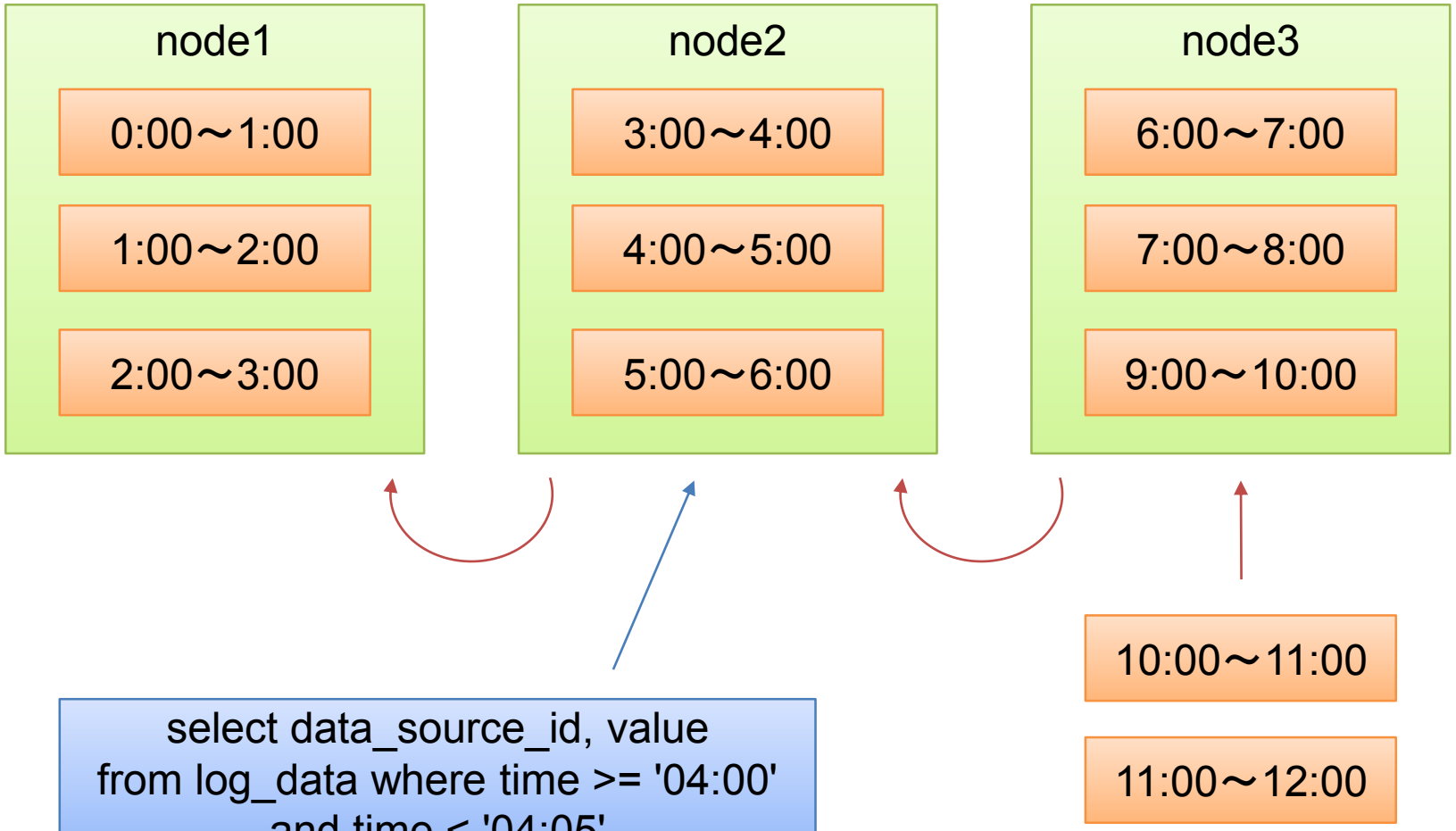
NaCl

シャーディング

- **行単位でデータを複数サーバに分散**
  - シャードキーと呼ばれる特定の列の値によってデータを格納するシャードを決定する方法が一般的
- **シャードキーの選択**
  - 時刻をキーにする場合
    - そのままシャードキーに使用すると、常に現在の時刻が含まれるシャードに登録が集中し、シャードの再配置が頻繁に起こる
    - ハッシュ値によるシャーディングではその問題はないが、参照の局所性が失われる
  - データソースIDをキーにする場合
    - 特定のデータソースのデータを検索する場合は、一つのシャードにアクセスするだけでよいため効率的
    - 同時刻のすべてのデータが欲しい場合には、すべてのシャードに問合せが必要

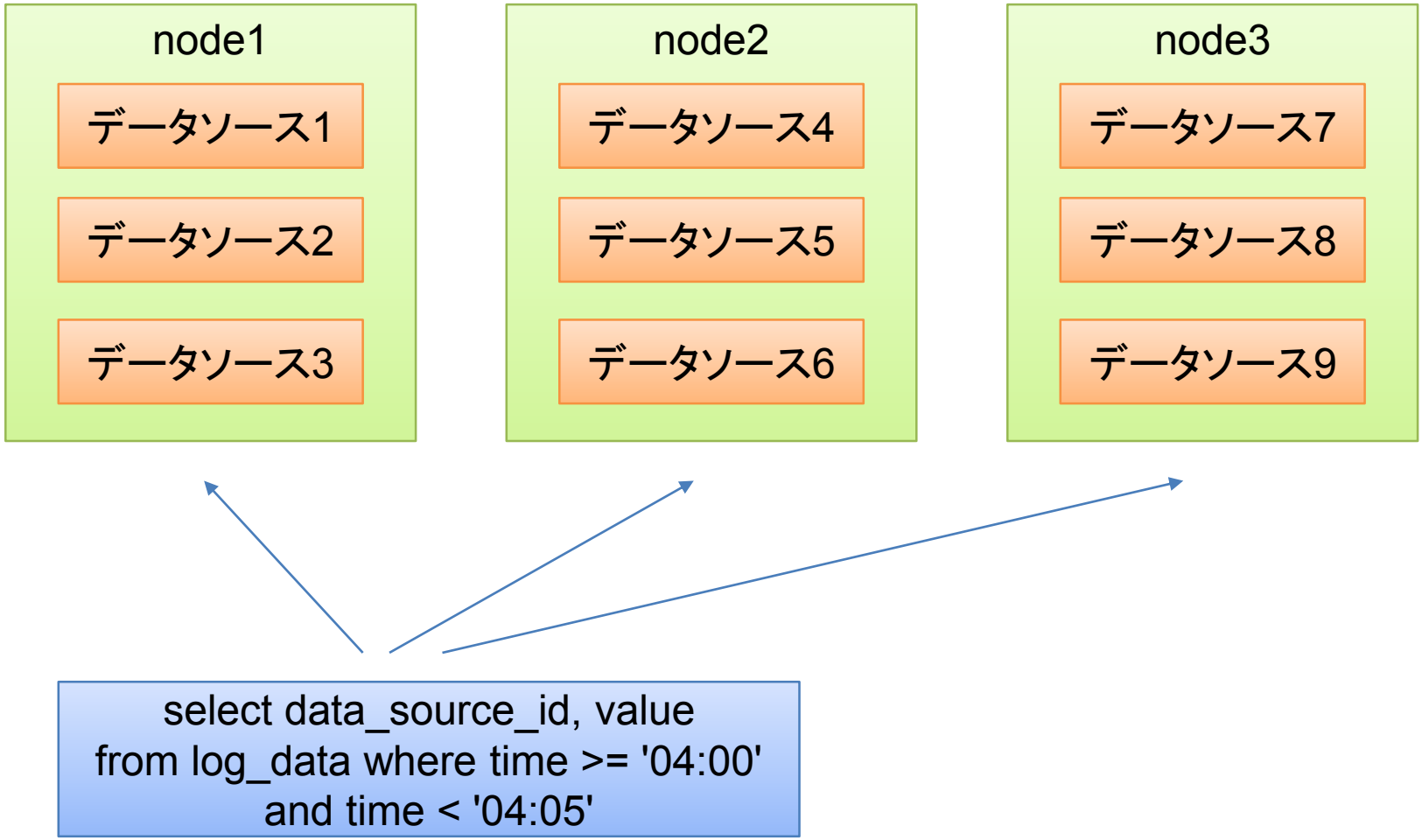
NaCl

時刻によるシャーディング



データソースIDによるシャーディング

NaCl



NaCl

InfluxDB

- Time Series Database (TSDB)
- SQLライクなクエリをサポート
- 時刻の範囲によってシャードを分割
- 各シャード毎にLevelDBにデータを格納

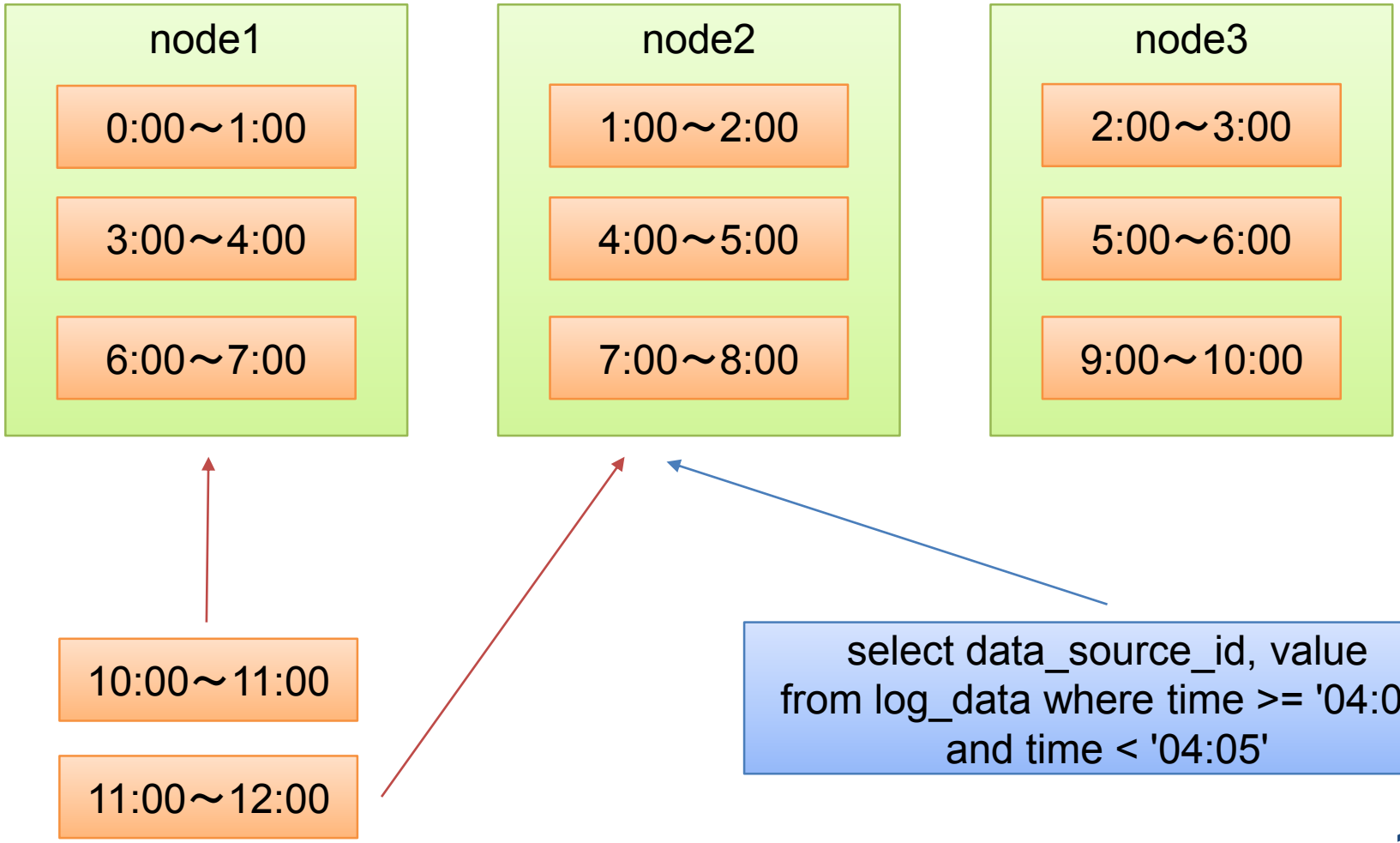
NaCl

LevelDB

- **ネットワークAPIを持たないシンプルなKVS**
- **キーによってデータがソートされている**
  - Sequential Read / Writeが高速
- **複数のレベルに分けてデータを保存**
  - 新しいデータはLevel-0に入り、古くなるにつれてより容量の大きいレベルに移動
- **Bloom filterによって探索するレベルの枝刈り**

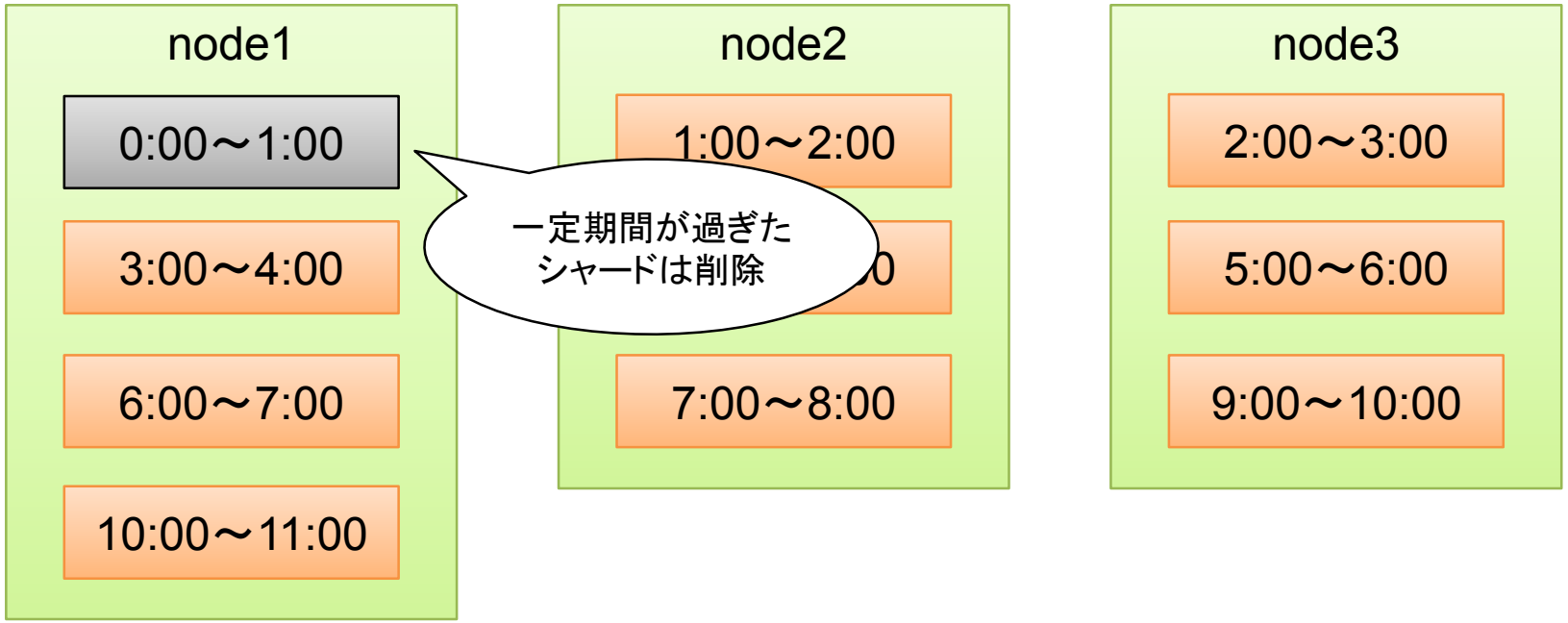
NaCl

InfluxDBのシャーディング



NaCl

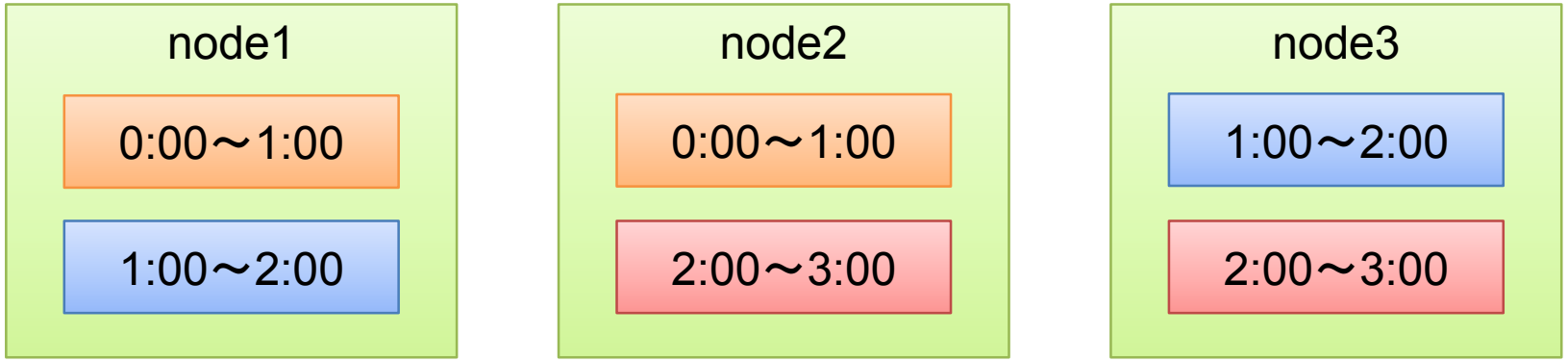
シャードのexpire



シャーディング・レプリケーションと負荷分散

NaCl

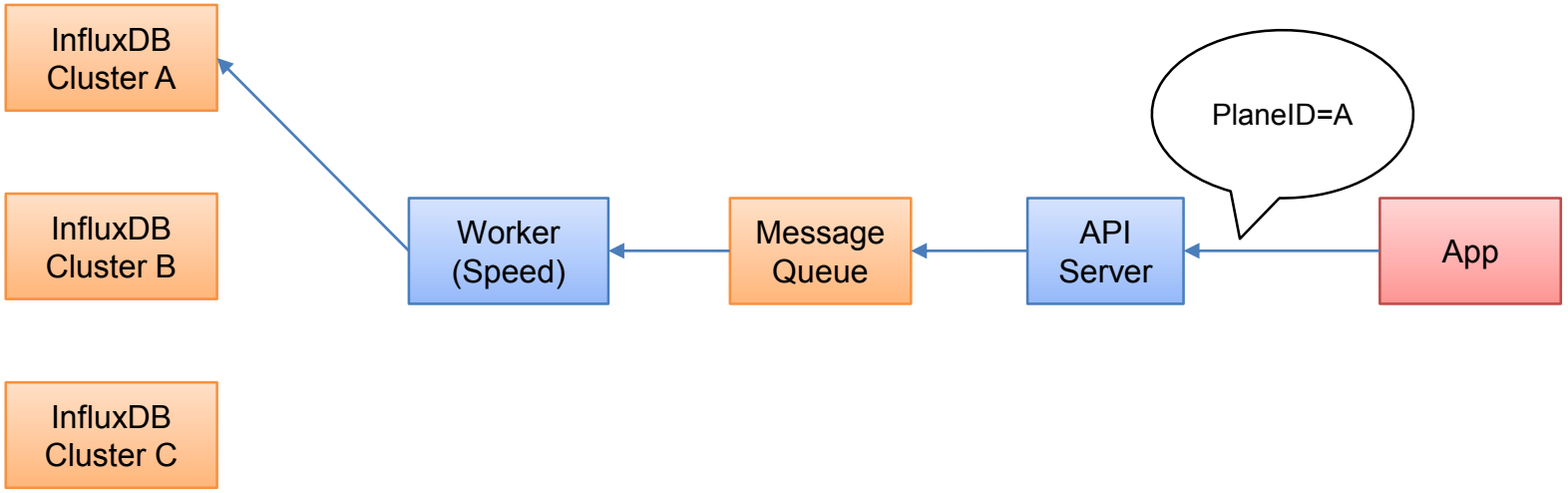
- 参照は負荷分散できるが登録は負荷分散できない
- データソースでシャードを分ければ登録負荷を分散できるが、参照時に複数シャードへのアクセスが必要



NaCl

プレーンの分割

- データソース8000点 / 8コンテナを一つの単位（＝プレーン）として InfluxDBクラスタを分割
- クラスタ分割の前段階としてテーブル分割する？



NaCl

まとめ

- スケーラビリティと低レイテンシを両立させるためレイヤーを分割
- Speed layerには時系列データに適したInfluxDBを採用
- InfluxDBの性能限界を考慮したプレーン分割

NaCl

補足

- **InfluxDBでは時刻以外の検索や集計処理に時間がかかる**
- **Continuous Query**
  - 入力データに対して検索・集計を継続的に実行
  - 結果は別テーブルに保存

NaCl

データソース毎の分割

- データソースID毎に別のテーブルにデータを保存する

```
select * from log_data into log_data.[data_source_id]
```

- データソースIDで検索する代りに分割されたテーブルを参照する

```
select * from log_data.213 where time > now() - 1h
```

NaCl

ダウンサンプリング

- **1時間毎の平均値を別のテーブルに保存する**

```
select data_source_id, mean(value) from log_data
  group by time(1h)
  into log_data.mean.1h
```