Cloudbusting Machine（CBM）

統一的クエリーインターフェース

藤田昭人
株式会社IIJイノベーションインスティテュート

はじめに

- Project Gryfon
  - 会社設立を含む事業化のための研究開発プロジェクト

- 現在のミッション
  - クラウド・データセンタ向け分散キャッシュシステム（2009～）
    - 経産省のグリーンITの一環
    - クラウド・データセンタの省エネ化を目指す
    - 今年度末に理論的検証が完了予定
    - 事業化・商業化にはまだ時間を要する

  - クラウド・コンポーザビリティをサポートするPaaSシステム（2010～）
    - 経産省「次世代高信頼・省エネ型IT基盤技術開発事業」の委託に基づく
    - プラットホーム・レイアでのクラウド間連携を実現する
    - オープンソース・プロジェクト（2011年春から半年ごとに計5回のリリース）
    - リリースに並行して事業化活動にも着手

Cloudbusting Machine（CBM）

- Project Gryfon の開発成果に基づくコードリリース
  - 主に「クラウド・コンポーザビリティをサポートするPaaSシステム」の成果
  - ホームページ： http://www.gryfon.iij-ii.co.jp/

- 現時点でのリリース
  - Key-Value Store （KVS）向けのサポート機能が中心
  - リリース内容
    - 統一的クエリーインターフェース
      - C による GQL 互換エンジン
      - PHP-C バインディング
      - MySQL 5.0.77 サポート
      - Cassandra 0.7.2 サポート
      - MongoDB 1.8.2 サポート
    - 周辺ツール
      - 郵便番号データベースによるベンチマーク（zipbench）

本日のトピック

- 統一的クエリーインターフェースの話題を中心に・・・

  - 背景: Key-Value Store の概要
    - 特徴、メリット・デメリット、分類、利用
  - 統一的クエリーインターフェース
    - コンセプト、アーキテクチュア、内部構造、API
    - 現時点での評価
    - 開発の現状

Key-Value Store（KVS）とは？

- Key-Value Store とは？
  - キーと値のペアで格納するストア（ストレージ）
  - 広義には連想配列なども該当するが・・・
    - 本稿で取り扱うのはデータベースやデータストアを実現するシステム
    - 最近では NoSQL の典型的な実装として紹介されることが多い

- Key-Value Store の特徴
  - リレーショナルモデルに基づかない
    - スキーマレスなデータベース
  - スケーラビリティに優れる
    - 大規模データの格納が可能

- Key-Value Store の運用事例
  - 大規模ウェブサービスのバックエンドが典型的な事例
    - Google の BigTable
    - Amazon の Dynamo

Key-Value Store のメリット・デメリット

- リレーショナルデータベース(RDB)との対比で語られる事が多い
  - スキーマ: SQL vs NoSQL
    - RDB はスキーマベースで予めテーブルを定義する必要がある
    - KVS はスキーマレスで予めテーブルを定義する必要がない
  - 性能向上: スケールアップ vs スケールアウト
    - RDB はシステムのハードウェア性能を向上 → 垂直スケーラビリティ
    - KVS はシステムのハードウェア並列化 → 水平スケーラビリティ
  - データ一貫性: ACID vs Eventual consistency
    - RDB は ACID によりデータ一貫性を厳格に保証
      - ACID: 原始性(Atomicity)、一貫性(Consistency)、分離性(Isolation)、永続性(Durability)
    - KVS は Eventual consistency により緩やかに一貫性を維持

- CAP定理
  - http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf
  - ノード間のデータ複製において次の3つを同時に保証することはできない
    - 一貫性(Consistency)、可用性(Availability)、分断耐性(PartitionTolerance)
    - RDB は C+A
    - KVS には C+P と A+P の事例がある

Key-Value Store の分類

- KVS には様々な実装方式がある
  - 実装事例は今のところ百花繚乱の状態
    - どれがベストかわからない　→　目的に応じて使い分け？

- 分散構成による分類
  - マスター・スレーブ型（C+P）
    - BigTable、HBase、Hypertable、MongoDB などなど
  - P2P型（A+P）
    - Dynamo、Voldemort、Cassandra などなど

- データモデルによる分類
  - Yahoo! Cloud Serving Benchmark 論文による
    - http://www.brianfrankcooper.net/pubs/ycsb.pdf
    - ドキュメント型　CouchDB、MongoDB
    - カラムグループ型　BigTable、Hbase、Hypertable、Cassandra
    - ハッシュテーブル型　Voldemort

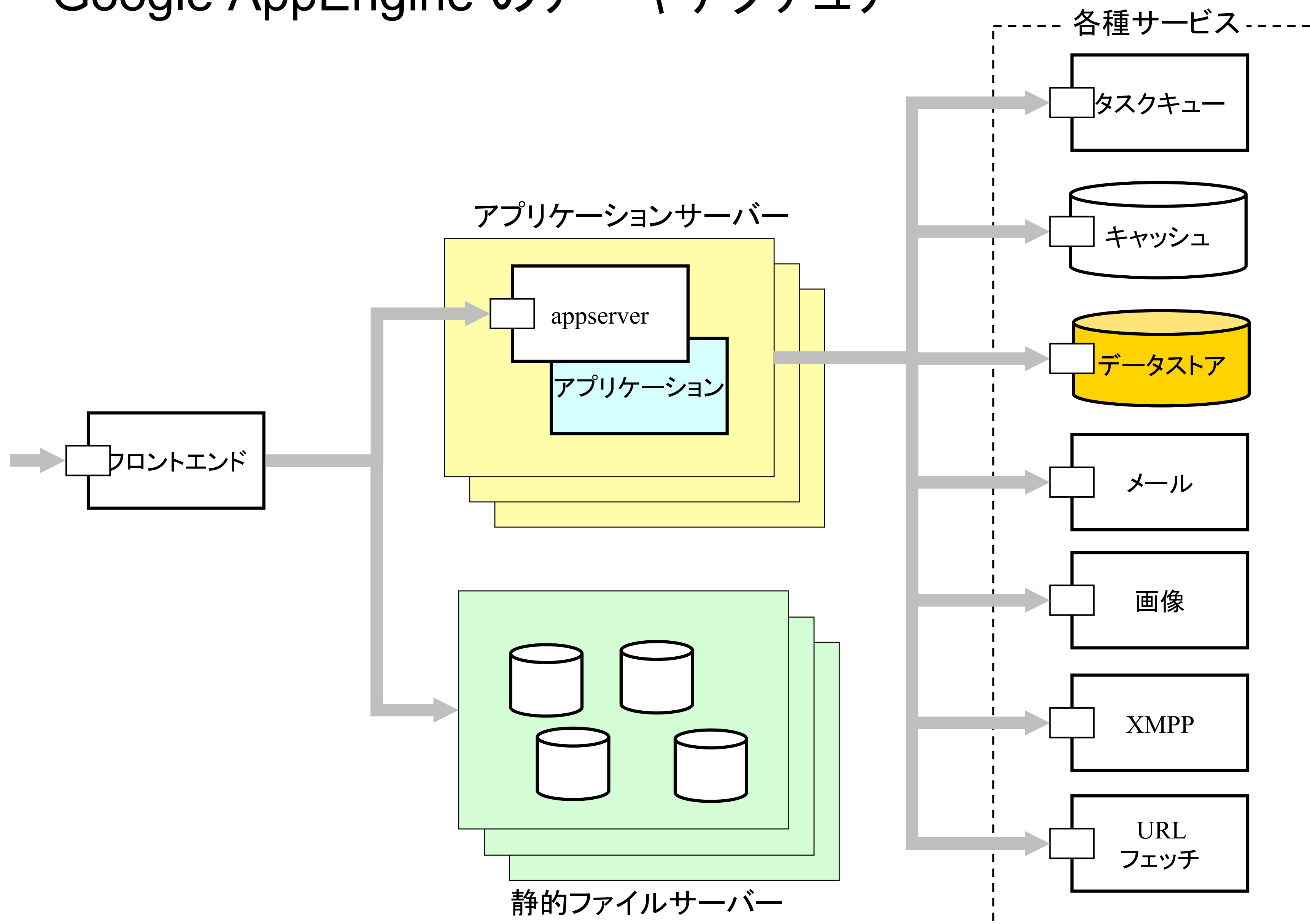
Key-Value Store の利用

- SQLデータベースの代替システムになり得るのか？
  - 主にデータモデルに起因して条件が変わってくる・・・
- ハッシュ型
  - いわゆるハッシュなのでクエリー・キャッシュなどの用途に限定される
- ドキュメント型
  - 機能的には概ねSQLデータベースのサブセットと考えて良いようだが、SQLが使えるわけではない　→　クエリーの書き換えが必要
    - MongoDB の場合
      - http://www.mongodb.org/display/DOCS/SQL+to+Mongo+Mapping+Chart
- カラムグループ型
  - SQLデータベースに比して機能的な欠落が多いので相当頑張らなければならない
    - RDB開発者におくるNoSQLの常識「RDBの機能をNoSQLで実現する」
      - http://www.atmarkit.co.jp/flinux/rensai/noSQL/noSQL_03/03_1.html
      - http://www.atmarkit.co.jp/flinux/rensai/noSQL/noSQL_04/04_1.html

統一的クエリーインターフェース（CQI） 開発目的

- 様々な KVS を対象に統一されたアクセス手段を提供する
  - CBMのクラウドアプリケーションから利用可能
  - 統一的なクエリー環境（スキーマ・クエリー言語）を定義
  - 同時に複数の KVS へのアクセスが可能
    - データベースの移行や複数データベースでの連携を想定している
- 既存のSQLデータベースと同じ感覚でKVSが利用できる
  - 主にウェブ・アプリケーション開発での利用を想定
    - 現時点では PHP にフォーカスしている
- データベース抽象化により KVS/RDB の特性に応じた利用を目指す
  - データモデルの異なるKVSやRDBを併用したシステム構成を可能にする
    例えば・・・
    - 水平スケーラビリティに優れたカラムグループ型KVSをマスターデータベースに
    - 厳格なデータ一貫性が期待されるデータはRDBに
    - データ投入にはスキーマレスのメリットが生かせるドキュメント型KVSに
    - ハッシュ型KVSを使ったクエリーキャッシュにより高速化
    - アプリケーションからは1つの仮想的なデータベースに見える

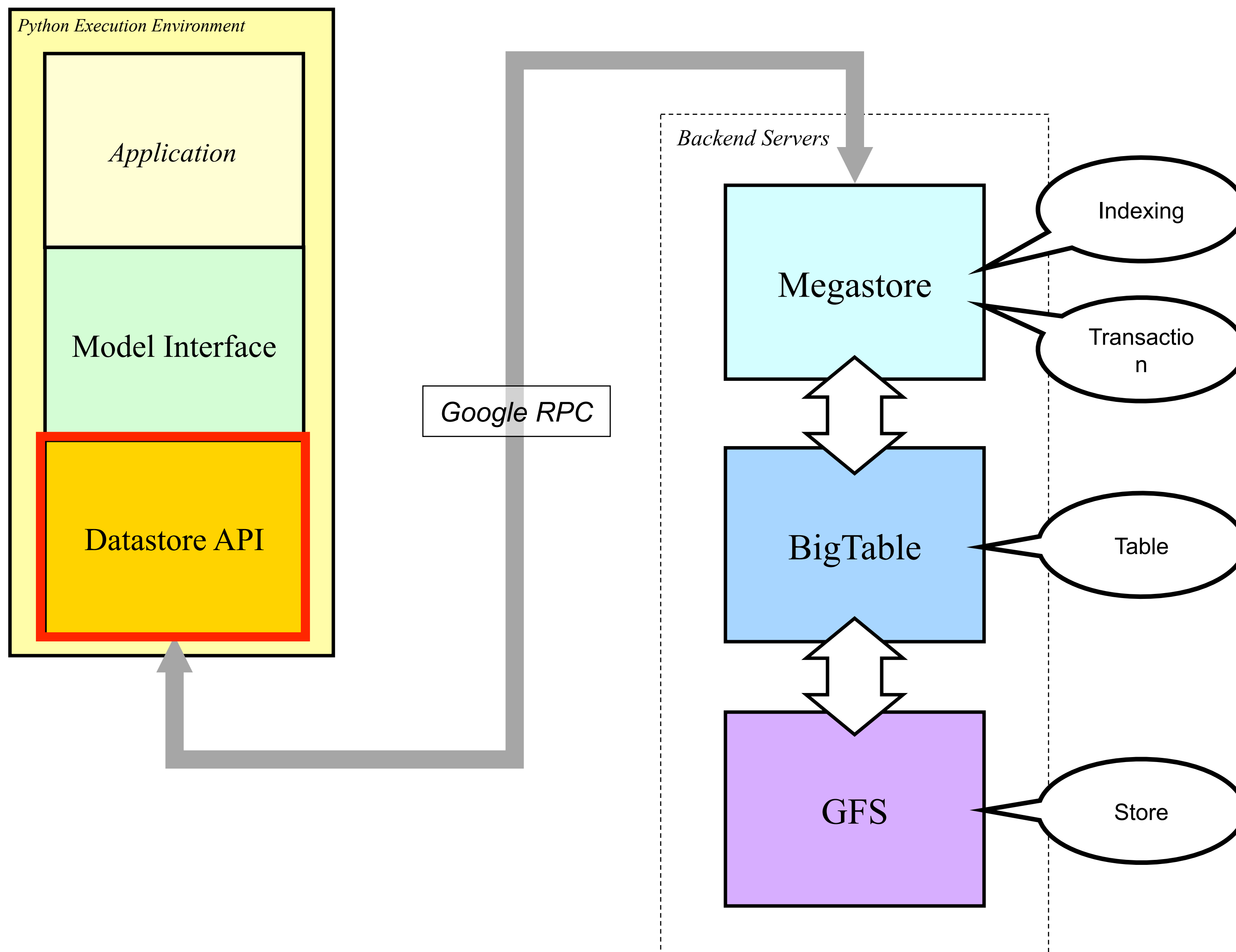
Google AppEngine のアーキテクチュア



Google AppEngine（GAE）のデータストア

- GAE のデータストア
  - データモデルは全てのサポート言語で共通
    - エンティティ:　GAE データストアの単位となるデータオブジェクト
    - キー:　エンティティの識別子
    - プロパティ:　エンティティに含まれる　SQLのカラムに相当（？）
  - API はサポート言語によって異なる
    - Python:　モデルインターフェースに基づく
    - Java:　JDO/JPA に基づく
    - Go:　コードがリリースされていないのでよくわからない
- CQI は C++/PHP ベースで GAE のデータストア機能を再現する試み（だった）
  - GAE/P であればソースコードが入手できる
  - Google AppEngine for Python SDK
    - http://code.google.com/intl/ja/appengine/downloads.html
    - GQE の各種サービスのアクセス・ライブラリとエミュレーション環境が付属
  - TyphoonAE
    - http://code.google.com/p/typhoonae/
    - GAE SDK を利用して GAE/P 環境をローカルで再現するプロジェクト

GAE/P のデータストアの内部構造



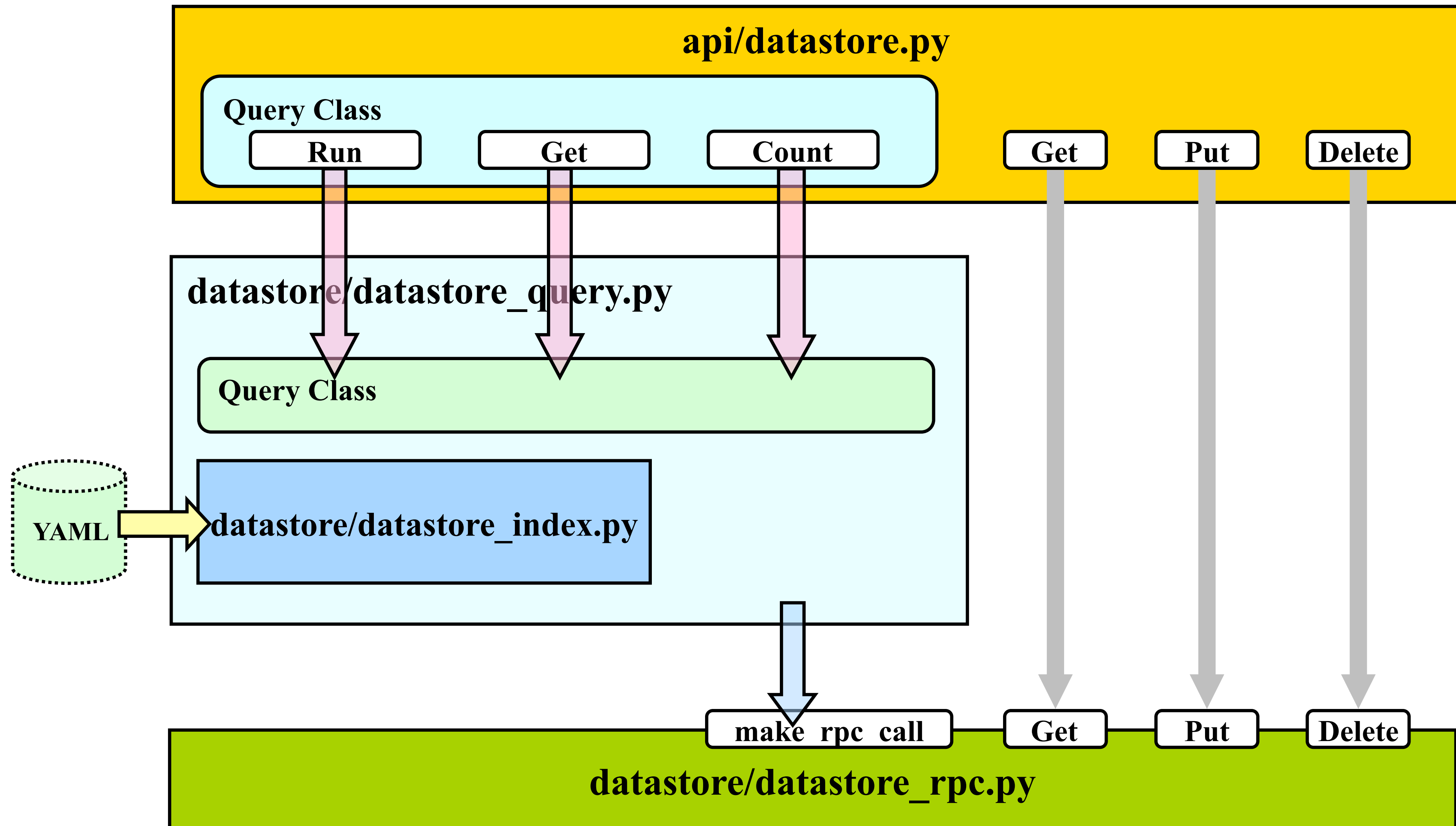
GAE/P Datastore API

- Datastore API: Google AppEngine のデータストア・サポートの要
  - モデルインターフェース（db パッケージ）はアプリケーション向けラッパー
    - アプリケーションからは Datastore API の実体は見えなくなっている
    - GQLのサポートは db パッケージで行っている

- GAE/P SDK で Python のソースを解析した結果・・・
  - インターフェース: google/appengine/api
  - 本体: google/appengine/datastore

  - 主要なコンポーネントは3つ
    - api/datastore.py API
    - datastore/datastore_query.py クエリーエンジン
    - datastore/datastore_rpc.py バックエンドとの通信
  - その他に重要なのは
    - api/datastore_types.py 主要なデータオブジェクトの定義
    - datastore/datastore_index.py カスタムインデックスの定義
    - datastore/entity_pb.py 各種プロパティの定義

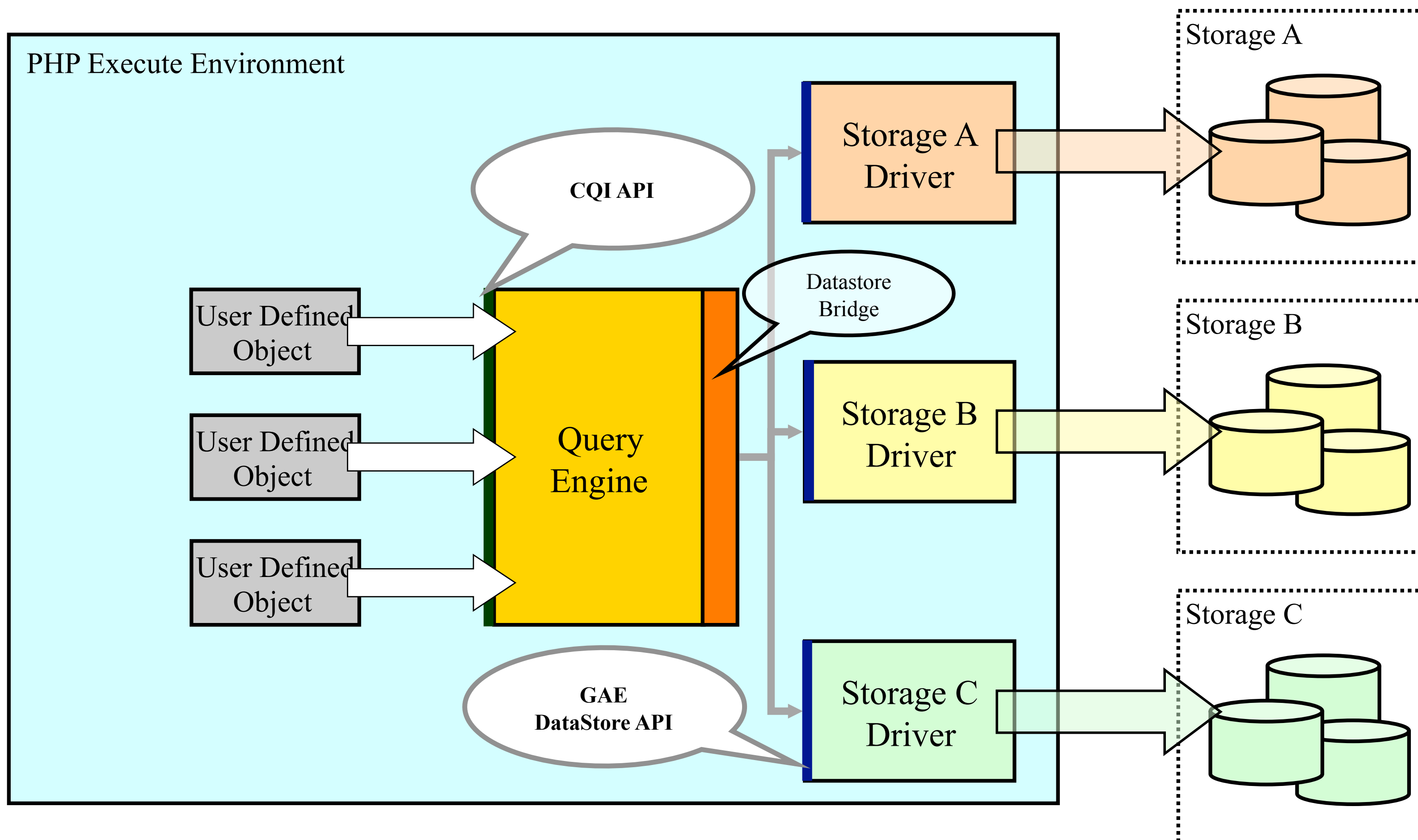
GAE/P Datastore API の内部構造



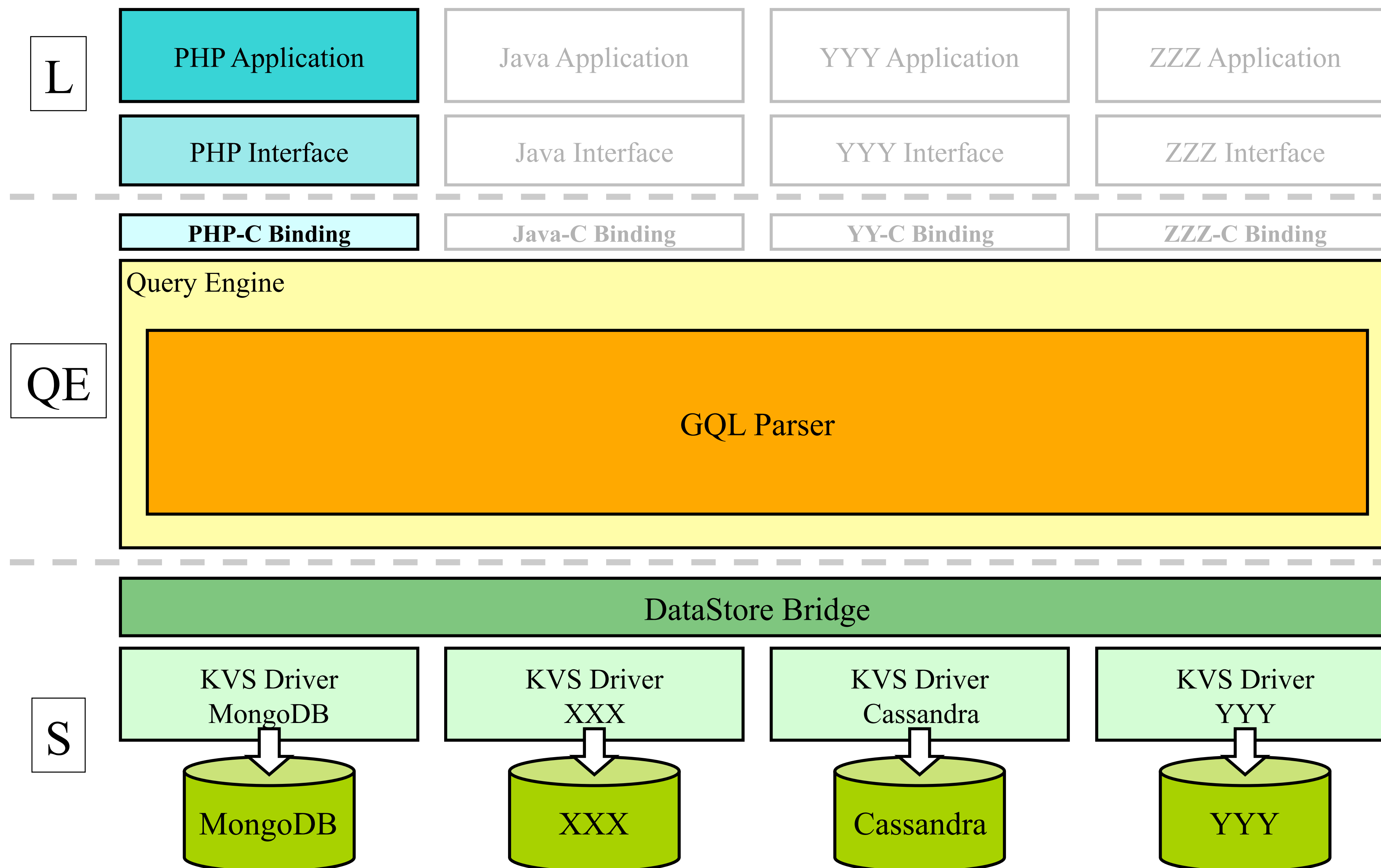
GAE/P でのクエリー

```
class MainPage(webapp.RequestHandler):
  def get(self):
    self.response.out.write('<html><body>')

    greetings = db.GqlQuery("SELECT * FROM Greeting ORDER BY date DESC LIMIT 10")

    for greeting in greetings:
      if greeting.author:
        self.response.out.write('<b>%s</b> wrote:' % greeting.author.nickname())
      else:
        self.response.out.write('An anonymous person wrote:')
      self.response.out.write('<blockquote>%s</blockquote>' %
                              cgi.escape(greeting.content))

    # Write the submission form and the footer of the page
    self.response.out.write("""
          <form action="/sign" method="post">
            <div><textarea name="content" rows="3" cols="60"></textarea></div>
            <div><input type="submit" value="Sign Guestbook"></div>
          </form>
        </body>
      </html>""")
```

統一的クエリーインターフェース　コンセプト



統一的クエリーインターフェース　アーキテクチュア



統一的クエリーインターフェース　内部構造

- ３種類のコンポーネントから構成される
  - Query Engine　GQL に準拠したクエリー言語を解釈
  - Datastore Bridge　API を定義し各処理ルーチンにディスパッチ
  - Storage Driver　GAE/P のDatastore API に相当する
各KVS毎に用意
- Query Engine
  - GQLはSELECTのみをサポートするクエリー言語
    - SELECT文を解釈してConditionとOrderingの情報のみを抽出
    - ANCESTORは取り合えず未対応
- Datastore Bridge
  - Storage Driver を選択するディスパッチャー
- Storage Driver
  - 各KVSのクライアントライブラリへのインターフェース変換を行う
    - 現在サポートしているのは Cassandra/MongoDBの２種類
    - 現時点でサポートを具体的に計画しているのは Hypertable など

- プログラミング・インターフェース
  - 現在は暫定的に C の関数スタイルのインターフェースをサポート

統一的クエリーインターフェース API

- Cによる API を規定している
  - ***${PREFIX}/include/cqi.h***

- CQI 構造体の生成・破壊

```
CQI_t      *cqi_new(char *driver, char *user, char *passwd, char *host, int port);
void        cqi_free(CQI_t *cqi);
```

- GQL によるクエリーの解釈

```
query_t    *cqi_analyze(CQI_t *cqi, char *gqlstr);
```

- ストレージへの接続・解除

```
int         cqi_attach(CQI_t *cqi, char *aux);
int         cqi_detach(CQI_t *cqi);
```

- ストレージへのアクセス

```
int         cqi_put(CQI_t *cqi, entity_t *ep);
int         cqi_delete(CQI_t *cqi, entity_t *ep);
entity_t   *cqi_fetch(CQI_t *cqi, query_t *qp, int *count);
int         cqi_count(CQI_t *cqi, query_t *qp);
```

Cによるサンプル

```
int
main(int argc, char *argv[])
{
        CQI_t *cqi;
        query_t *qp;
        entity_t *ep;
        char *gqlstr = "SELECT * FROM xzip";
        int i, n;

        cqi = cqi_new("Cassandra",           // driver
                      "root",            // user
                      "root",            // passwd
                      "127.0.0.1",       // host
                      9160);             // port
        cqi_attach(cqi, "xzipcode.xzip");    // connect datastore

        qp = cqi_analyze(cqi, gqlstr);
        cqi_query_show(qp);

        n = cqi_count(cqi, qp);
        printf("\n# %d entries\n", n);
        if (n > 0) {
                ep = cqi_fetch(cqi, qp, &n);
                if (ep != NULL) {
                        for (i = 0; i < n; i++) {
                                printf("\n# %d\n", i);
                                cqi_entity_show(&ep[i]);
                        }
                }
                cqi_entity_free(ep);
        }

        cqi_query_free(qp);
        cqi_detach(cqi);                 // disconnect datastore
        cqi_free(cqi);
        exit(0);
}
```

PHPによるサンプル

```
<?php
if(!extension_loaded('cqi_php')) {
        dl('cqi_php.' . PHP_SHLIB_SUFFIX);
}

$query_string = 'SELECT * FROM xzip';
$queryArray = array();
$gql = array('main_sentence' => $query_string,
             'param_count' => count($queryArray),
             'param' => $queryArray);

$result = cqi_analyze($gql);
$qarray = $result['query'];
$query = $qarray[0];

$driver = array('driver' => 'cassandra',
                'user'   => 'root',
                'passwd' => 'root',
                'host'   => '127.0.0.1',
                'port'   => 9160,
                'aux'    => 'xzipcode.xzip');

$result = cqi_count($driver, $query);
$result = cqi_fetch($driver, $query);
?>
```

統一的クエリーインターフェースに対する反応

- ８月時点での反応
  - ウェブ・アプリケーション系
    - クエリ言語そのものに関心がない：　ORM経由で利用しているから
    - KVSとえいばMongoDB：　RDBと同じ感覚で使えるから
  - 分散システムコミュニティ
    - 需要がイメージできない：　（KVSの併用は想定していない模様）
    - KVSといえばCassandra：　他のKVSは大規模運用が不可能
  - エンタープライズ系
    - オープンソース系KVSの運用実績のなさに二の足を踏んでいる模様
    - 抱えているデータの規模だとRDBでも対応可能・・・と考えている模様
- 漠然とした感想
  - KVS に対する実際的な需要はまだ本格化していない？
    - クラウド・ストレージはサービスとして利用するもの？
    - KVSを必要とするような大規模データを持っていない？

ストレージシステムに関わる技術動向

- サポート対象（候補を含む）のKVSの技術開発動向
  - 一部の KVS は SQL ライクなクエリー言語をサポートし始めている
    - CQL:　Cassandra
    - HQL:　Hypertable
  - SQL 的な構文によりクエリー実行が可能になっているが・・・
    - RDBの標準的な機能を全てサポートしているわけではない
    - 標準的な SQL でクエリーを実行するとエラーが多発する
      - 似て非なるものと思った方が良さそう

- オープンソースKVSを活用した運用事例は増えているが・・・
  - 大規模サービスを提供している事業者の試験的運用が主力？
    - ググるといろいろ出てきます
  - 中には「Oracle から Cassandra に移行した」みたいな勇ましい事例も
    - Netfilix: Replacing Datacenter Oracle with global apache cassandra on AWS
      - http://lanyrd.com/2011/cassandrasf/sgdwy/
      - http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra

現状に対する考察

- KVSの現状が集約方向に向かうのはまだ少し先の話？
  - 1つは無理にしても幾つかのシステムに集約されると予想していたが
  - 今もって「どれが本命なのか？」さえよくわからない
    - DataStax の説明を聞くと Cassandra が本命にも思えるが・・・
      - http://www.datastax.com/
    - Oracle まで NoSQL を出してきたし・・・
      - http://www.oracle.com/us/products/database/nosql/resources/index.html
- 現在のムーブメントを牽引しているのは供給サイド？
  - 受益者サイドはどちらかと言えば困惑ぎみ？
    - 元々大量・大規模データの処理の必要な既存ニーズはそれほど多くない？
    - 差し迫った需要にはクラウド・ストレージ・サービスを使えばよいし・・・
    - 自社で KVS を運用する需要を持つ企業は未だ少数？
- CQIとしては当初の開発目的に立ち戻る
  - 既存のSQLデータベースと同じ感覚でKVSが利用できる
    - PHP によるウェブ・アプリケーション開発での利便性に集中
    - PDO（PHP Data Object） のサポートに注力

PHP Data Object（PDO）拡張モジュール

- 概要
  - PHP標準のRDBデータベース抽象化ライブラリ
    - 標準でMySQLやSQLiteなどをサポート
    - 拡張モジュール（Cベース）として実装されているのでPEAR:DBよりも高速
    - PDO Core モジュールとPDO Driver モジュールが分離されている
      - CQI対応のドライバーモジュールさえ書けばPDOはサポート可能

- インターフェース
  - PHPアプリケーションは2つのPHPクラスでアクセス可
    - PDOクラス　データベースとの接続に関わるクラス
    - PDOStatement クラス　クエリー結果に関わるクラス

- 参考資料
  - http://www.php.net/manual/ja/class.pdo.php
  - http://www.php.net/manual/ja/class.pdostatement.php

PDO対応版CQIの開発課題

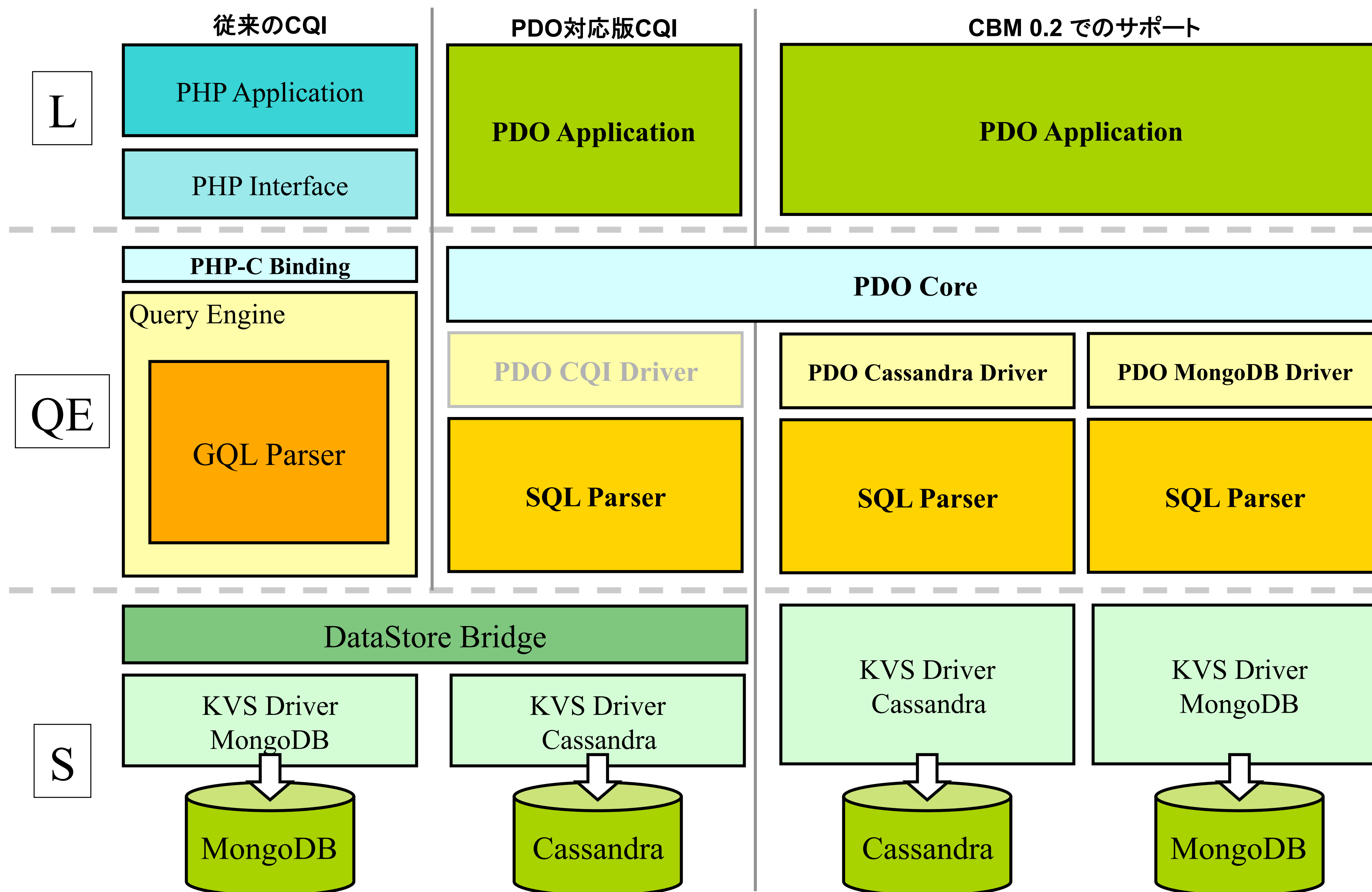
- PDOはSQLデータベースを抽象化するインターフェース
  - 既存の CQI 実装では対応できない機能が多々ある
  - 主要な問題点は下記の3つ

  - SQLのサポート
    - 一般的なSQL文を正しく解釈できる必要がある
    - サポートできる機能は各 KVS に依存
      - KVSドライバー毎に任意の機能をサポートできる構造でなければならない

  - スキーマのサポート
    - 一般的なSQL文はスキーマを仮定している
      - スキーマを持たないKVSではCQIで独自にスキーマを管理しなければならない
      - 既にスキーマレスで格納されているデータへのアクセス手段が必要

  - APIの非互換性
    - PDOのAPIは既存のCQIのAPIと互換性がない
      - ドライバーはほぼ全面的に再実装する必要がある

PDO対応版 CQI アーキテクチュア



PDO対応版 CQI 内部構造

- 現時点でサポートしているのは Cassadra と MongoDB
- 次の3つのコンポーネントから構成される
  - PDO Driver　　PDO Core に対するインターフェース
    - 全体の制御を司る
    - 現時点では Cassandra/MongoDB の各々に対して独立して存在
    - 最終的には PDO CQI Driver として統合する予定
  - SQL Parser　　SQL 構文の解釈
    - SQLite のSQL構文解釈モジュール（tokenize/parse）を流用して実装
      - http://www.sqlite.org/arch.html
    - 現時点でサポートしてるのは次の5つのコマンドのみ
      - CREATE TABLE/INSERT/SELECT/DELETE/DROP TABLE
    - それ以外のコマンドを解釈した場合は未実装のメッセージ
    - 各コマンドのオペラントのサポートは KVS ドライバーによる
  - KVS Driver　　KVS に対するインターフェース
    - 各KVSのクライアントライブラリに対するインターフェース
    - 従来の KVS Driver をベースに PDO 向けインターフェースに改変

PDO対応版 CQl　Cassandra Driver（1）

- 対応バージョン
  - Apache Cassandra 1.0
- 実装方法
  - 付属するThrift Driver をクライアントライブラリに使用
  - SQL を CQL（Cassandra Query Language）に変換して実行
    - Secondary Index は使用していない　→　WHERE句は未サポート
- 制約条件
  - CREATE TABLE
    - データ型はすべて「varchar」型に強制変換する
    - NOT NULL/PRIMARY KEY /UNIQUEは受け付けるが機能しない
  - INSERT
    - カラムに“KEY”という名前を設定できない
      - Cassandra のキーにためにリザーブされている
    - カラムのデータ型に関わらず全て文字列として格納する
      - PDOのインターフェースとの整合を取るため

PDO対応版 CQI　Cassandra Driver（2）

- 制約条件（続き）
  - DELETE
    - WHERE 句が使えない　→　事実上テーブルのクリアになる
  - DROP TABLE
    - （制限はない・・・はず）
  - SELECT
    - 基本的に全件検索のみ
      - WHERE句は使用できない　→　エラーを返す
      - ORDER BY句は使用できない　→　順序は不同
      - GROUP BY 句は使用できない　→　機能しない
      - HAVING句は使用できない　→　機能しない
    - JOINが伴う操作は全てできない
    - 指定されたカラム名のチェックは行っていない
      - 存在しないカラム名が指定された場合は空文字が返される
    - KEYを取得したい場合は、「KEY」と明示しなければならない
      - アスタリスク(*)ではKEYは取得されない

PDO対応版 CQI　MongoDB Driver（1）

- 対応バージョン
  - MongoDB 2.0
- 実装方法
  - mongo-c-driver 0.1 を使用
    - https://github.com/mongodb/mongo-c-driver
  - SQL を Mongo Query（BSONによるクエリー）に変換
- 制約条件
  - CREATE TABLE
    - データ型はすべて「varchar」型に強制変換する
    - NOT NULL/PRIMARY KEY/UNIQUEは受け付けるが機能しない
  - INSERT
    - MongoDBの主キー（_id）が取得できない
      - mongo-c-driver に対応するAPIがない
    - カラムのデータ型に関わらず全て文字列として格納する
      - PDOのインターフェースとの整合を取るため

PDO対応版 CQI　MongoDB Driver（2）

- 制約条件（続き）
  - DELETE/DROP TABLE
    - （制限はない・・・はず）
  - SELECT
    - WHERE句は機能するが・・・
      - ORDER BY句は使用できない　→　順序は格納順
      - GROUP BY 句は使用できない　→　機能しない
      - HAVING句は使用できない　→　機能しない
    - JOINが伴う操作は全てできない
    - SELECT 時の取得フィールドの順は定義時の順序に基づく

Cloudbusting Machine（CBM） 0.2

- PDO対応版CQIをサポートするリリース
  - 0.2　　　　　１１月１１日
    - Cassandra/ MongoDB をサポート
    - CQI ドライバーとしての統合は未対応
  - 0.2.1　　　　１２月２日
    - 統合化 CQI ドライバーのサポートを計画
  - 0.2.2　　　　１２月２３日
    - Hypertable のサポートを計画

- 開発課題:　　SQLサポートの品質向上？
  - 各種並べ替え処理（ORDER BY句や検索結果取得順序など）はローカルで
    - そんな機能はZend Engineがサポートしているだろう・・・
  - リレーショナルモデルでの結合演算（JOIN）はSQLデータベースと別の方法
    - KVSがスキーマレスである以上、結合演算を実行するのはそれ以外のどこか
    - PHP の拡張モジュール内で実装する処理としては重すぎる気がする
  - トランザクション処理は当面忘れる
    - KVSではPaxosアルゴリズムを用いた手法を用いるのが一般的だが・・・
      - http://en.wikipedia.org/wiki/Paxos_(computer_science)

おわりに

- ･･･に代えて　In Progress なので結論の代わりに問いを

- はたしてKVSとはデータベースなのか？
  - BigTable の論文では一貫して Store との表現が使われている
    - もちろん大量データの格納を想定しているのでクエリーは必須だが
  - SQL的クエリー機能の実装は正しいアプローチなのだろうか？
    - SQLは「建て増し続きで迷路のようになった邸宅」のような言語である
    - KVSがSQL的クエリーの実装を目指しても中途半端に終わる？
      - 我々はSQL→xQLマッピングに苦労していることがその証拠

- クラウドストレージを考える上では･･･
  - 「データベースの抽象化モデルとしてのSQLデータベース」
    - データモデルの異なるKVSやRDBを併用したシステム構成を可能にする
      - 水平スケーラビリティに優れたカラムグループ型KVSをマスターデータベースに
      - 厳格なデータ一貫性が期待されるデータはRDBに
      - データ投入にはスキーマレスのメリットが生かせるドキュメント型KVSに
      - ハッシュ型KVSを使ったクエリーキャッシュにより高速化
      - アプリケーションからは1つの仮想的なデータベースに見える