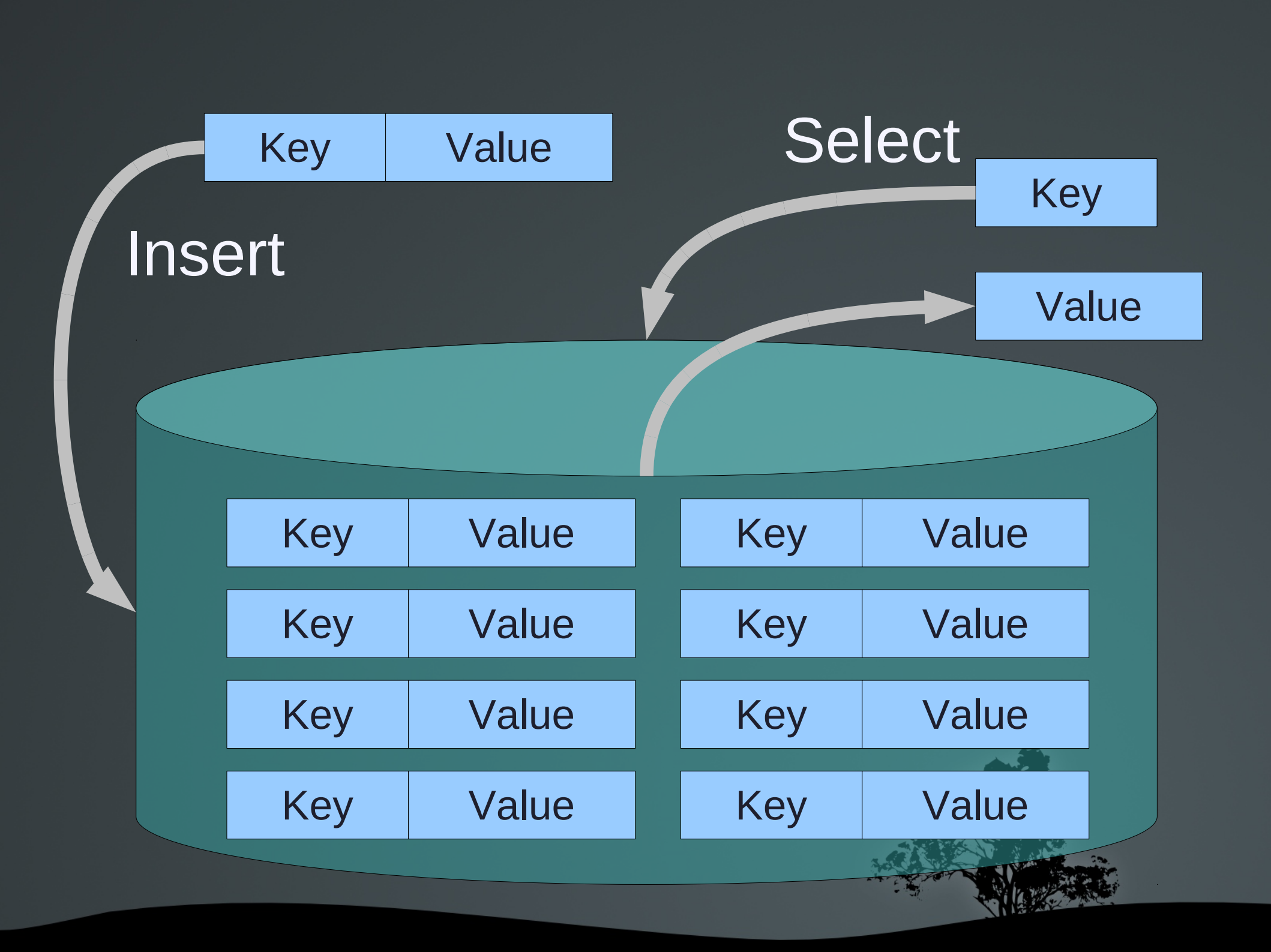
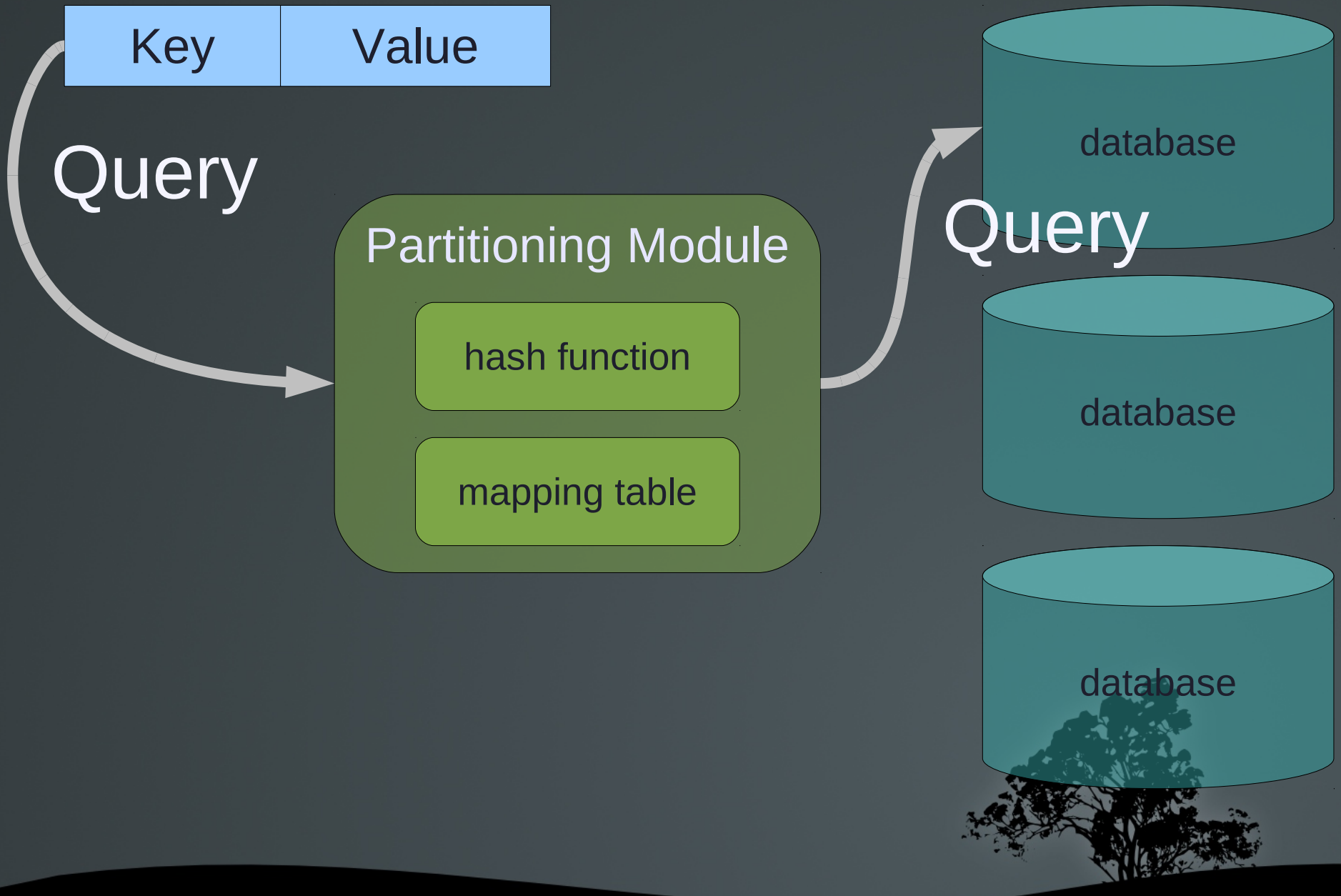
# KVS の基礎と Tokyo Cabinet

平林幹雄　mikio@fallabs.com

# Key-Value Store の「傾向」

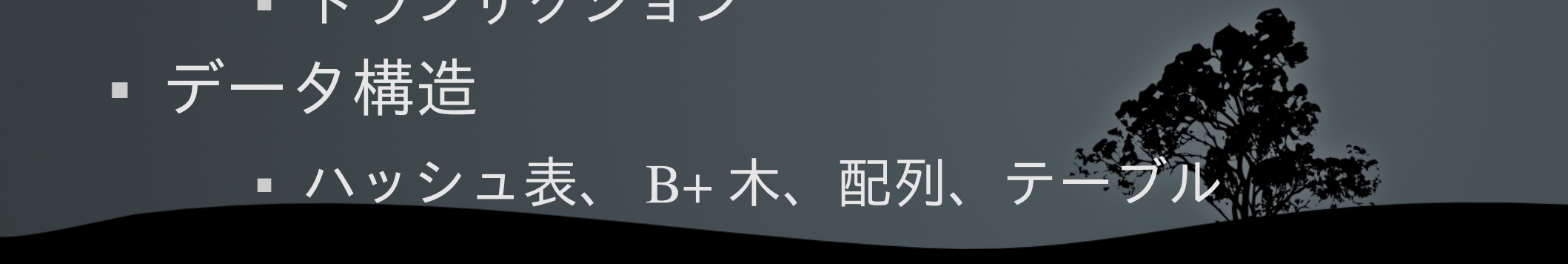
- 連想配列の永続化
  - Key も Value も非構造
  - Key による完全一致検索
  - プロセスの寿命 ＜ データの寿命
- 単純化による高速化と分散対応
  - クエリの解析が不必要
  - 集合演算や関係演算を省略
  - パーティションキーによって水平分散


Key
Value
Insert
Select
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value
Key
Value


Key
Value
Query
Partitioning Module
hash function
mapping table
Query
database
database
database

# Tokyo Cabinet

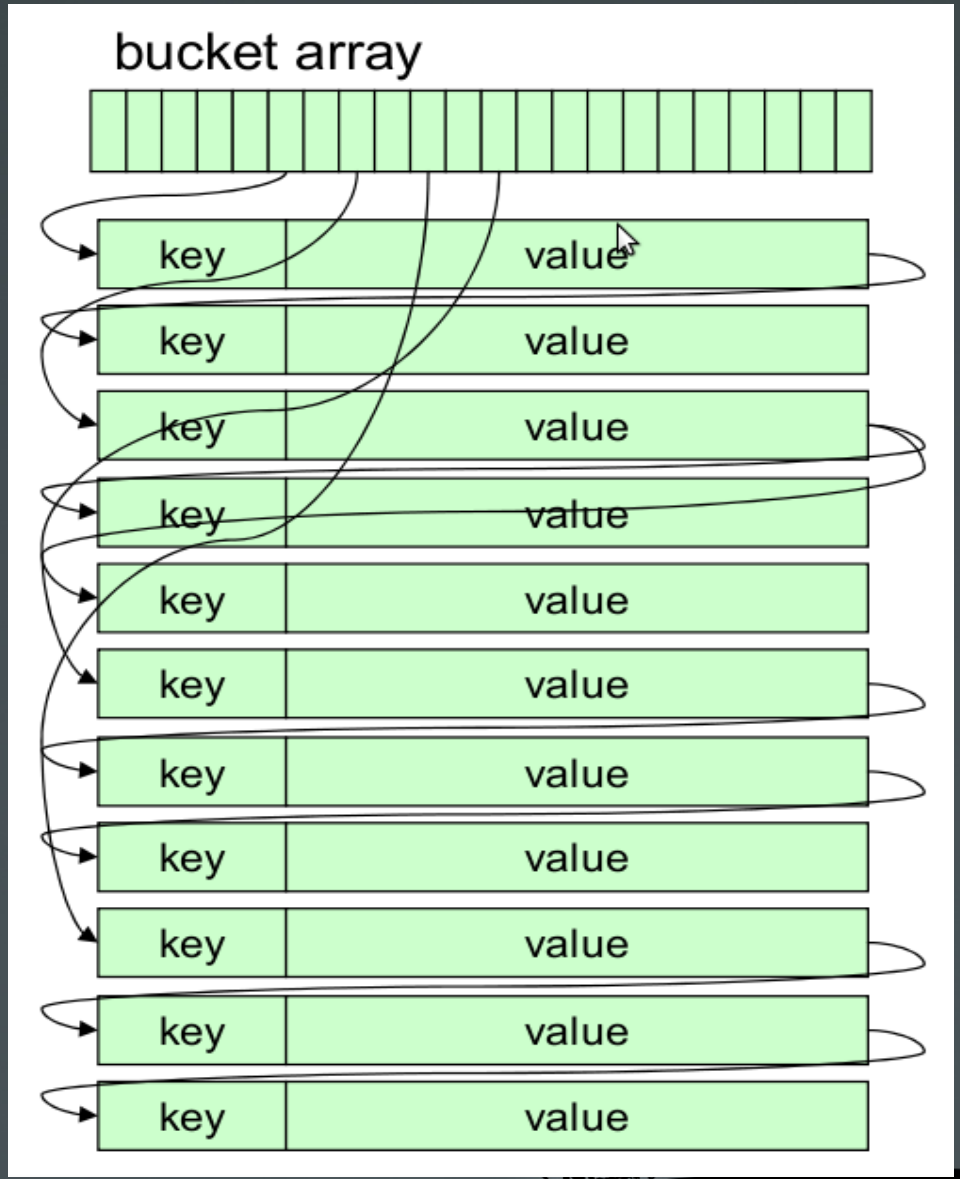
- UNIX DBM の末裔
  - DBM (Ken Thompson), Berkeley DB (Oracle)
  - プロセス組み込み
    - DB サーバではない
- 現代化
  - マルチスレッド対応
  - 細粒度ロック
  - トランザクション
- データ構造
  - ハッシュ表、 B+ 木、配列、テーブル



```
require 'TokyoCabinet'

db = TokyoCabinet::HDB::new
db.open("mydatabase.hdb");

db["key1"] = "value1"
db["key2"] = "value2"

printf("%s\n", db["key1"])
printf("%s\n", db["key2"])

db.each do |key, value|
  printf("%s: %s\n", key, value)
end

db.close
```

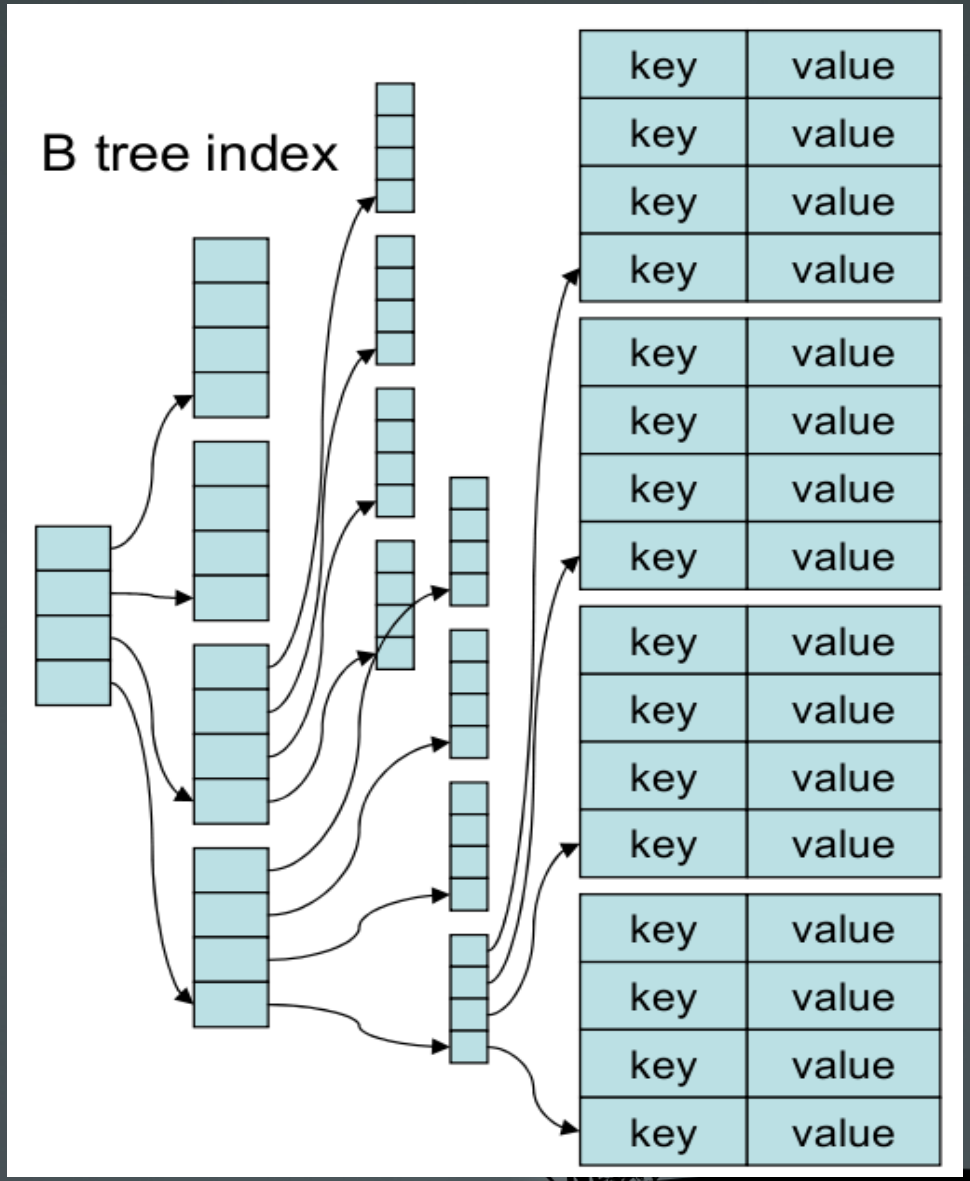
# ハッシュ表

- ハッシュ関数で探索領域を限定
  - O(1)
  - 完全一致のみ
- 衝突は二分探索木で管理
  - O(log N)
- フリーブロックプール
- 自動デフラグ



# B+ 木

- 比較関数で探索領域を限定
  - O(log N)
  - 範囲検索、順制御
- ページキャッシュ
  - 実質　O(1)
- ハッシュ DB に保存
  - 空間効率を継承



# 利点と欠点

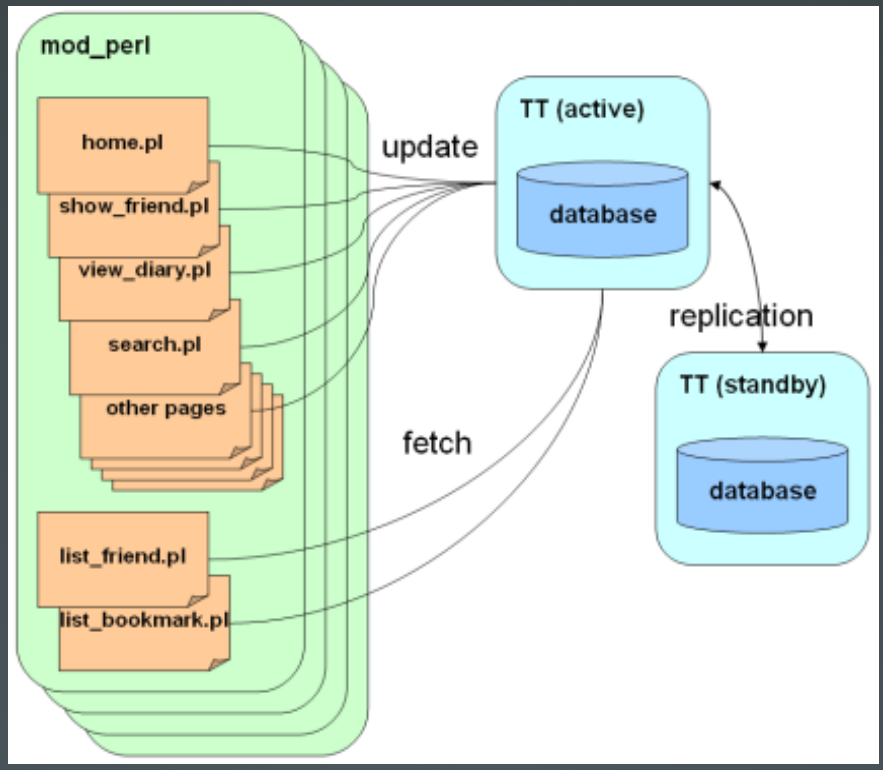
- 利点
  - やたら高速
    - 2,000,000 query/sec
  - 管理が楽
    - サーバ不要
    - 各言語の組み込みハッシュと互換
- 欠点
  - ローカルのみ
    - 　分散機能なし
  - 集合演算や関係演算がない
    - アプリケーション側で実装

# Tokyo Tyrant

- Tokyo Cabinet のネットワーク対応
  - 独自バイナリプロトコル
  - memcached 互換、 HTTP 互換
- 分散はクライアント任せ
  - consistent hashing, mapping table 等
- 非同期レプリケーション
  - マスタ・スレーブ、マスタ・マスタ
- Lua 言語拡張
  - ユーザ定義関数

# mixi での事例

- ログインタイムスタンプ
- 1 台に集約
  - 以前は memcached で分散
  - 最新ログイン順ソート
- 全ページで参照
  - read = N0,000 qps?
  - write= 10,000 qps?
- マスタ・マスタレプリケーション



# 他システムでの利用

- KVS
  - flare(GREE), ROMA( 楽天 ), kumofs(ficia)
  - LightCloud(Plurk), CloudKit, etc.
- その他
  - Tokyo Dystopia, Tokyo Promenade
  - bogofilter, mutt, gearman, etc.

# 「クラウド」との関係

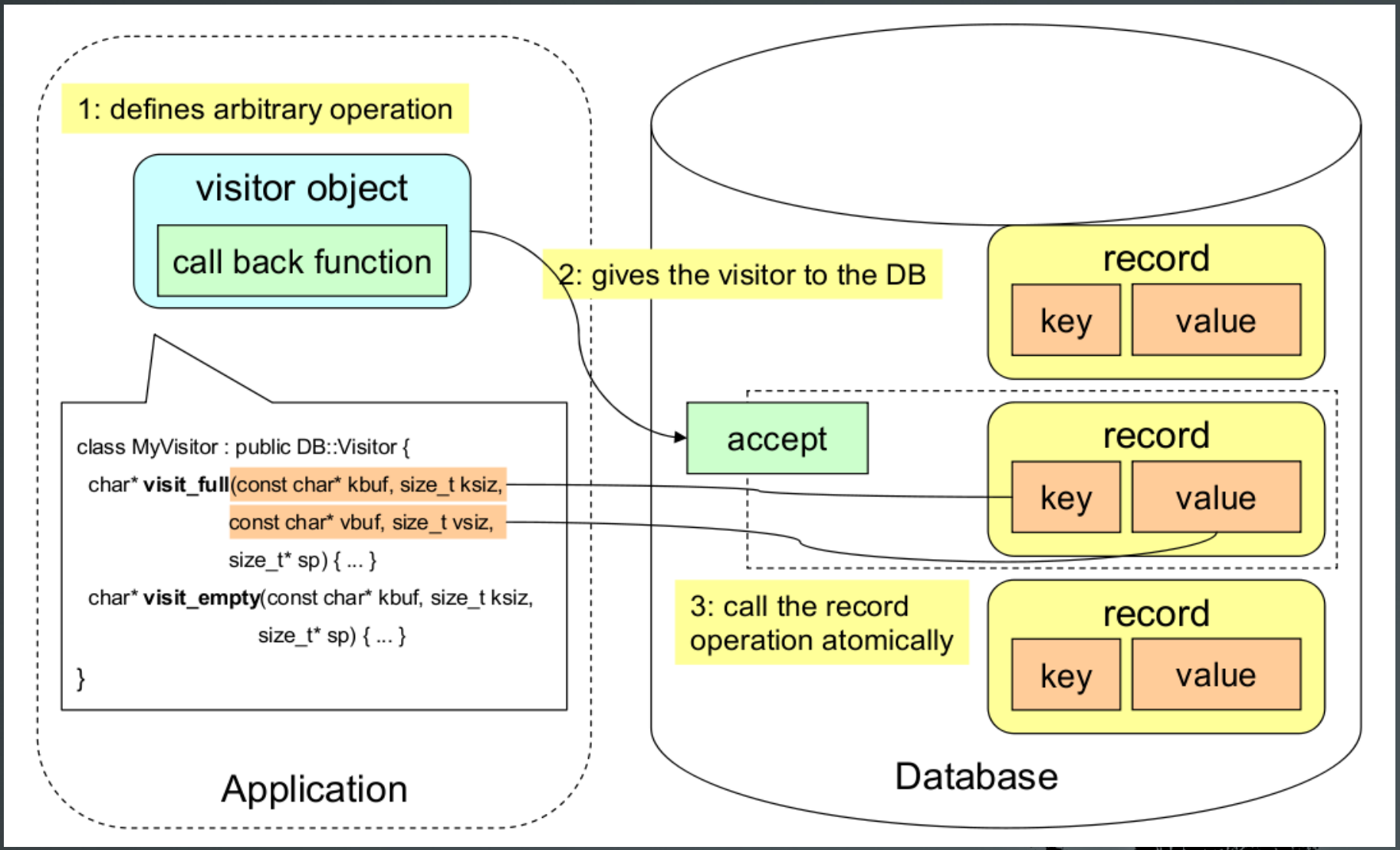
- クラウド時代の非クラウドソリューション
  - クラウド基盤の中で使われる可能性

- DB ではなく永続連想配列ライブラリとして
  - データマイニング、検索用インデックス
- 単純だが高負荷な処理
  - 足あと、タイムスタンプ、採番、 abuse 監視
- キャッシュ
  - セッション情報

# Kyoto シリーズ

- Kyoto Cabinet
  - Tokyo Cabinet の再実装（ C++ ）
  - マルチコア最適化
  - Windows 対応
  - Visitor インターフェイス
- Kyoto Tycoon
  - Tokyo Tyrant の再実装（ C++ ）
  - キャッシュ最適化（ auto expiration ）
- デュアルライセンス
  - GPLv3 と商用ライセンスの両立

# Visitor インターフェイス



# 今後

- Tokyo シリーズの保守は継続
  - 不具合修正のみ。機能追加はしない
- Kyoto シリーズの普及活動
  - 大規模 Web サービスでの導入促進
  - 運用支援ツール開発
    - レプリケーション
    - 分散管理
    - バックアップ

ご清聴ありがとうございました