Groonga族 2015

須藤功平　株式会社クリアコード

Groonga Meatup 2015
2015-11-29

内容

- Groonga族の概要
- Groonga族の最新情報
- Groonga族の今後の情報

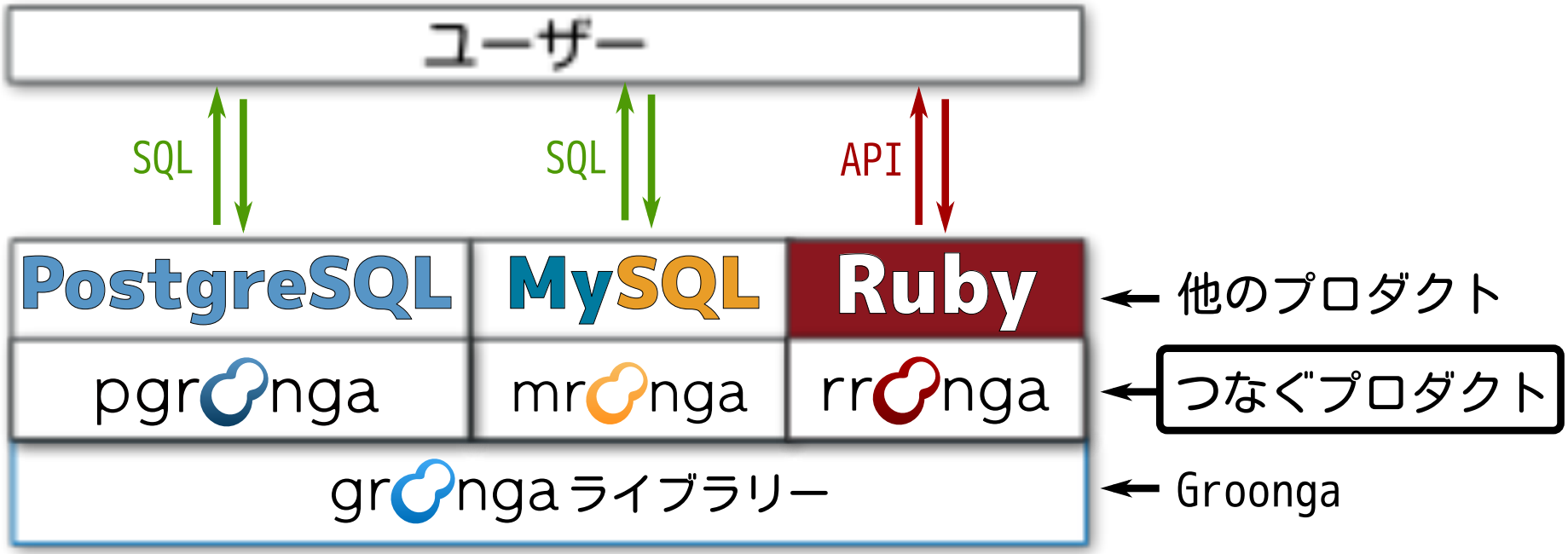
Groonga族

総称

Groonga族

- Groongaそのもの
- Groongaと他のプロダクトを
  つなぐプロダクト
- 名前がXroonga

つなぐプロダクト



ポイント
Groongaはライブラリーとして他のプロダクトに埋め込める！

Groonga族：Groonga

ぐるんが

Groongaのよいところ1

高速な全文検索機能

（全文検索エンジンなので当然）

Groongaのよいところ2

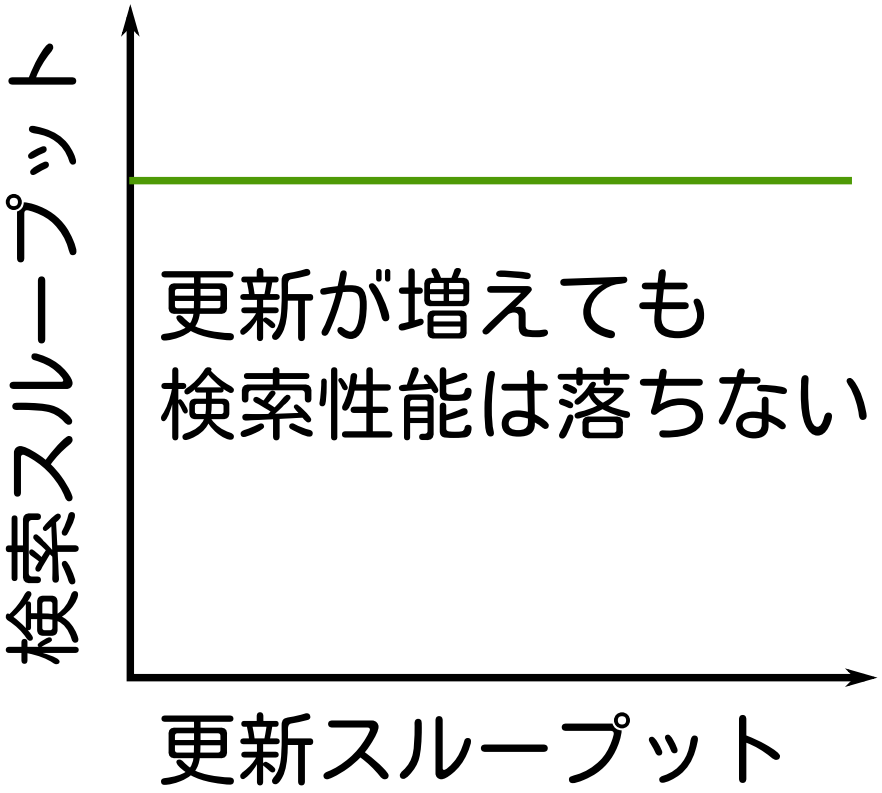
即時更新

即時更新

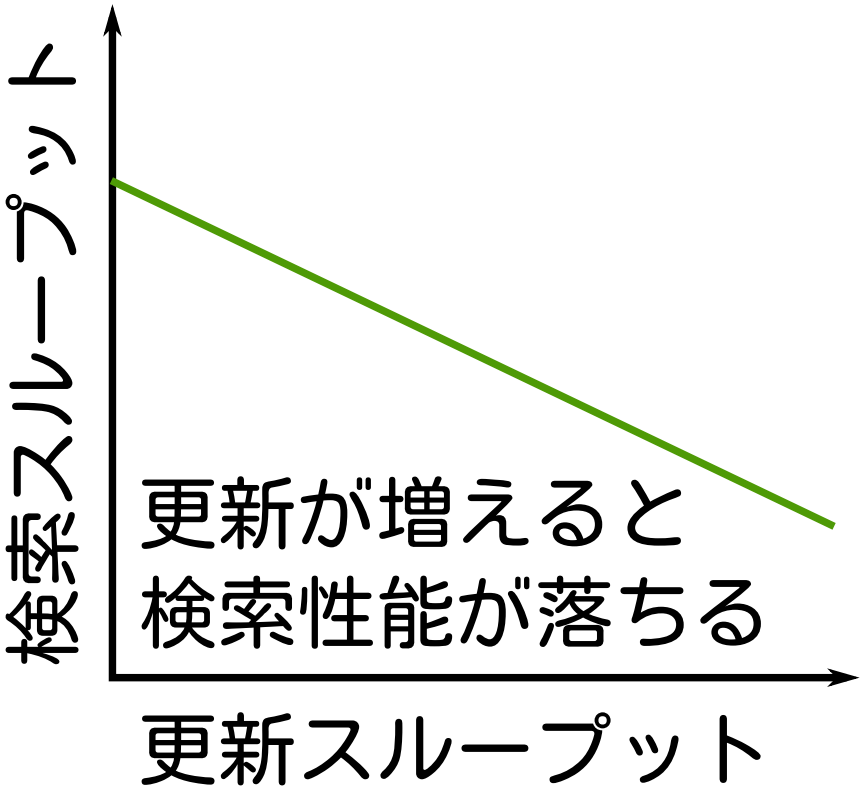
- 更新→すぐに検索可能
  - 思想：新鮮なデータは高価値
  - 新しいデータはすぐに入れて！
- 工夫：更新中も検索性能キープ
  - 即時データ投入を妨げる理由を減少
  - 実装：参照ロックフリー

性能の傾向

Groonga



更新中は参照不可の
エンジン



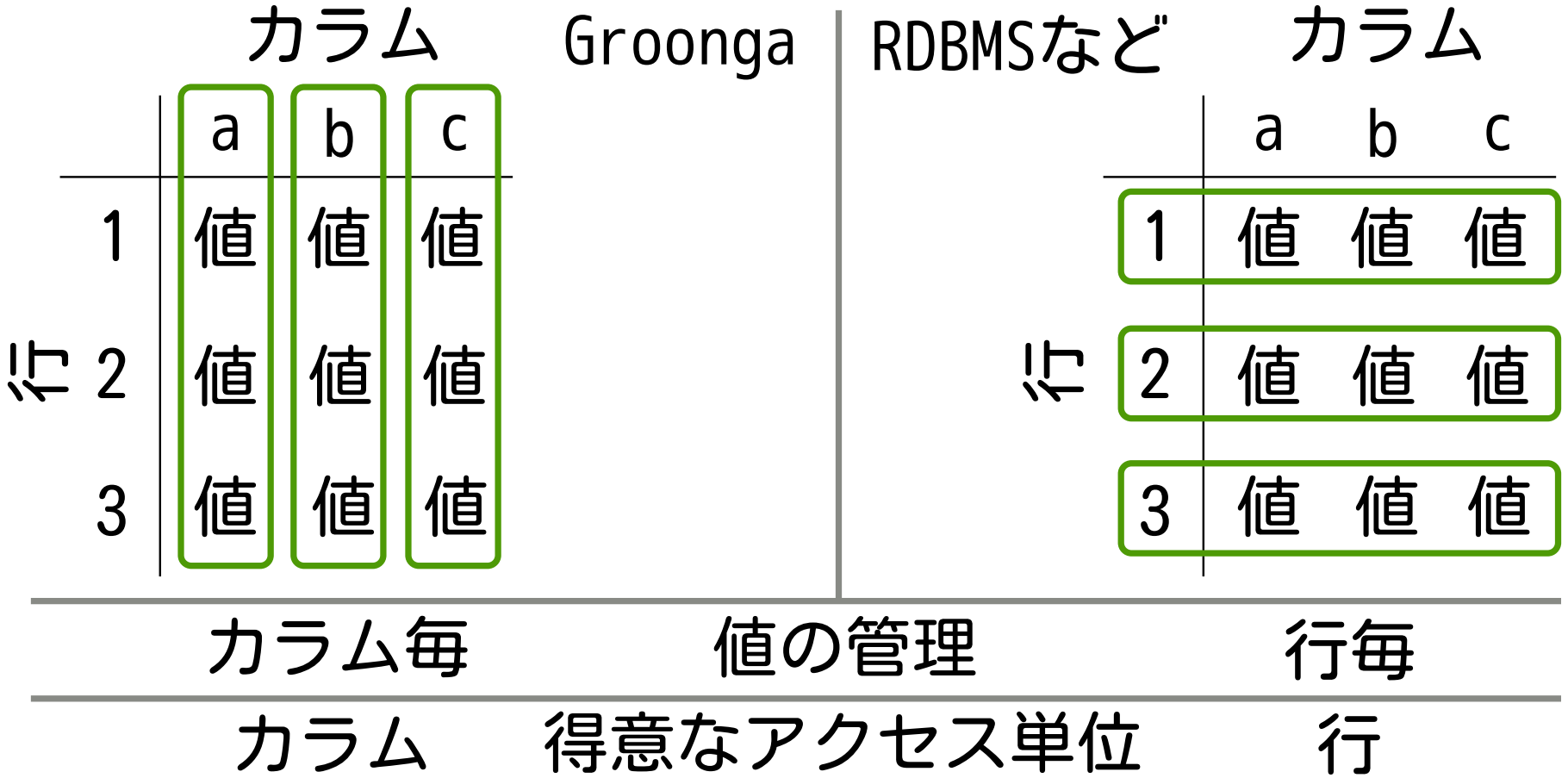
Groongaのよいところ3

高速な集計処理

高速な集計処理

- 集計処理
  - 同一カラムの値を一気にアクセス
- 工夫：↑向けデータ構造を採用
  - 高速な集計処理を実現
  - 実装：カラムストア

カラムストア


カラム
Groonga
RDBMSなど
カラム
a b c
1 値 値 値
2 値 値 値
3 値 値 値
a b c
1 値 値 値
2 値 値 値
3 値 値 値
カラム毎
値の管理
行毎
カラム
得意なアクセス単位
行

Groongaのよいところ3

手厚い
日本語サポート

サポート例：RKサーチ

ローマ字でカナを検索
Romaji to Kana prefix search

- 例：「ya」→「ヤキニク」
  - 「や」・「ヤ」→「ヤキニク」も可
- 用意するもの
  - カタカナでのヨミガナだけ
    （ひらがな・ローマ字はいらない）
    （ローマ字の訓令式・ヘボン式の差はGroongaが吸収）

RKサーチの利用例1

入力補助

- 例：タグの入力
  同じことに違うタグをつけると検索精度があがらない
  全体で一貫性があることは重要
  既存のタグを入力補助することで同じことに同じタグをつけやすくする
- 使い所：妥当な値の集合がある
  - タグ入力なら既存タグ

RKサーチの利用例2

検索ボックスでの補完

- Googleもやっているやつ
- ぐるなびはGroongaで実現

ぐるなびでの利用例

「yu」→「ユウラクチョウ」→「有楽町」

出典：Groongaのサジェストで四苦八苦 by Walker
https://speakerdeck.com/redfigure/groongafalsesaziesutodesi-ku-ba-ku

補完の使い方1

スキーマ作成

```
% groonga-suggest-create-dataset \
    DB_PATH stations
```

補完の使い方2

データ準備

```
[
  {
    "_key": "有楽町",
     "kana": ["ユウラクチョウ"],
     "boost": 100
  }
]
```

boostを100以上にすること

補完の使い方3

データロード

```
% curl \
    'localhost:10041/d/load?table=item_stations' \
    --header 'Content-Type: application/json' \
    --data-binary @stations.json
```

curlを使うときは--dataではなく--data-binaryを使うこと
--dataは改行を削除するため

補完の使い方4

補完

```
% curl \
    'localhost:10041/d/suggest' \
    --get \
    --data-urlencode 'types=complete' \
    --data-urlencode 'table=item_stations' \
    --data-urlencode 'column=kana' \
    --data-urlencode 'query=yu'
[
  [...],
  {"complete": [[1],[...],["有楽町",101]]}
]
```

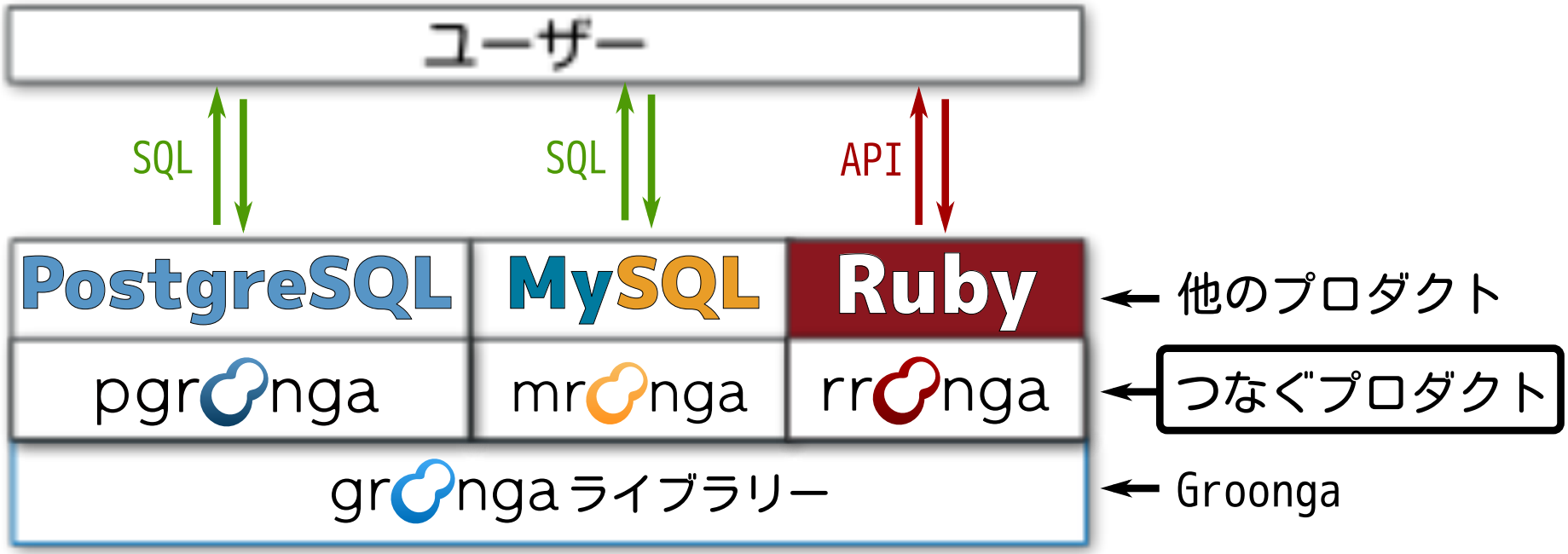
Groongaのよいところ4

組み込める

組み込める

- ライブラリーとして利用可能
  - ＝インターフェイスを変えられる
  - ＝使いやすい方法で使える

つなぐプロダクト



ポイント
Groongaはライブラリーとして他のプロダクトに埋め込める！

Groonga族：Mroonga

むるんが

MySQL + Groonga

Mroongaのよいところ1

簡単に使える

簡単

- SQLで使える
- 全文検索の知識なしで
  使い始められる

Mroongaを使う：スキーマ

```
CREATE TABLE texts (
  content TEXT,
  FULLTEXT INDEX (content)
  -- ↑と↓を追加
) ENGINE=Mroonga
  DEFAULT CHARSET=utf8mb4;
```

ポイント：全文検索の索引を作ると言っているだけ

Mroongaを使う：検索

```
SELECT * FROM texts
  WHERE
    MATCH(content)
    AGAINST('*D+ キーワード1 キーワード2'
            IN BOOLEAN MODE);
```

ポイント：Web検索エンジンと同様にクエリーを書ける

なぜ知識なしで使えるの？

適切な
デフォルト値

デフォルト値

- トークナイザー：可変長Bigram
  （トークナイザー≒検索キーワード抽出モジュール）
  - 日本語+英語でもいい感じに動く
- ノーマライザー：MySQL互換
  （ノーマライザー≒テキスト正規化モジュール）
  - 一部非互換：☕≠🍵

Mroongaのよいところ2

MySQLの機能を使える

MySQLの機能

- レプリケーション
- ユーザー管理
- クライアントライブラリー
- ...

Mroongaのよいところ3

速い！

例：MySQL 5.7と比較

ポイント：MySQL 5.7のInnoDBは日本語全文検索対応！

クエリー	文字数	InnoDB mecab	InnoDB bigram	Mroonga bigram
COUNT	1	2.62s	N/A	**0.74s**
COUNT	2	**0.07s**	1.74s	0.27s
ORDER BY	1	2.78s	N/A	**0.83s**
ORDER BY	2	**0.09s**	2.72s	0.31s

出典：MySQLの全文検索に関するあれやこれや by yoku0825
http://www.slideshare.net/yoku0825/mysql-47364986

- InnoDB mecabは遅いことがある
- Mroongaは安定して速い

MySQL 5.7で日本語全文検索

```
CREATE TABLE texts (
  content TEXT,
  FULLTEXT INDEX (content)
    WITH PARSER mecab -- ←がポイント
) ENGINE=InnoDB
  DEFAULT CHARSET=utf8mb4;
```

mecabの代わりにngramも指定できるが実用的ではなさそう
mecabを使い場合は↑の他にMeCabの設定が必要
https://dev.mysql.com/doc/refman/5.7/en/fulltext-search-mecab.html

Mroonga vs MySQL 5.7

- Mroonga
  - デフォルトでいい感じに動く
    （選びたければ指定できる）
  - 安定して速い
- MySQL 5.7 InnoDB
  - パーサーを選ばないといけない
    （mecab?ngram?違いは？）
  - 遅いこともある

Mroongaが速い理由：1

Groonga直結

↓

Groongaが速いと
Mroongaも速い

Powered by Rabbit 2.1.9

Mroongaが速い理由：2

ORDER BY LIMIT
を最適化

ORDER BY LIMIT最適化

```
SELECT * FROM texts
  WHERE MATCH(content1)
        AGAINST('...' IN BOOLEAN MODE)
  ORDER BY priority
  LIMIT 10;
```

MySQLに先頭10件しか返さない

ORDER BY LIMITをMySQLが処理するより
Groongaが処理する方が速いので速くなる

ORDER BY LIMIT最適化！

```
SELECT * FROM texts
  WHERE MATCH(content)
        AGAINST('...' IN BOOLEAN MODE)
    AND n_likes < 100 -- 追加
  ORDER BY priority
  LIMIT 10;
```

Groonga:MATCH AGAINST以外も処理

MySQLが処理するよりGroongaが処理する方が速いので速くなる

Mroongaが速い理由：3

必要なカラム
のみ
アクセス

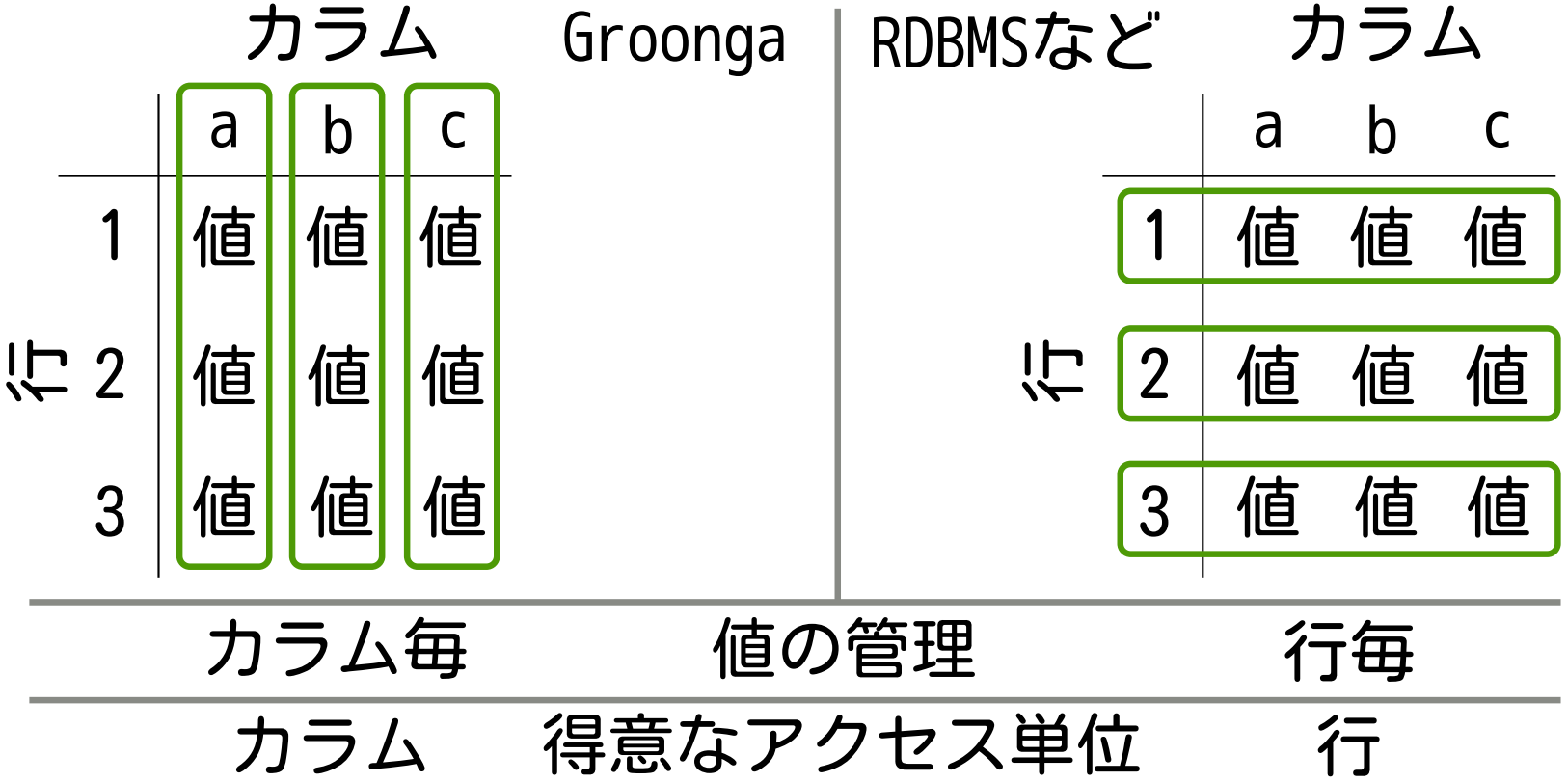
必要なカラムのみアクセス

```
SELECT c1, c3, c5 FROM t;
```

c2, c4にはアクセスしない
→I/Oが減って速い

Groongaはカラムストア

必要なカラムのみアクセスできる理由



Groonga族：PGroonga

ぴーじーるんが

PostgreSQL + Groonga

PGroongaのよいところ1

簡単に使える

簡単

- SQLで使える
- 全文検索の知識なしで
  使い始められる

PGroongaを使う：索引

```
CREATE INDEX name ON texts
  USING pgroonga (content);
```

ポイント：全文検索の索引を作ると言っているだけ

PGroongaを使う：検索

```
SELECT * FROM texts
  WHERE
    content @@ 'キーワード1 キーワード2';
```

ポイント：Web検索エンジンと同様にクエリーを書ける

なぜ知識なしで使えるの？

適切な
デフォルト値

デフォルト値

- トークナイザー：可変長Bigram
  （トークナイザー≒検索キーワード抽出モジュール）
  - 日本語+英語でもいい感じに動く
- ノーマライザー：NFKC
  （ノーマライザー≒テキスト正規化モジュール）
  - Unicode正規化形式の1つ

PGroongaのよいところ2

PostgreSQLの
機能を使える

PostgreSQLの機能

- トランザクション
- ユーザー管理
- クライアントライブラリー
- ...

PGroongaのよいところ3

速い！

例：pg_bigmと比較

ポイント：PostgreSQL標準では日本語全文検索非対応

ヒット数	PGroonga	pg_bigm
368	**0.030s**	0.107s
17,172	**0.121s**	1.224s
22,885	**0.179s**	2.472s
625,792(*)	0.646s	**0.556s**

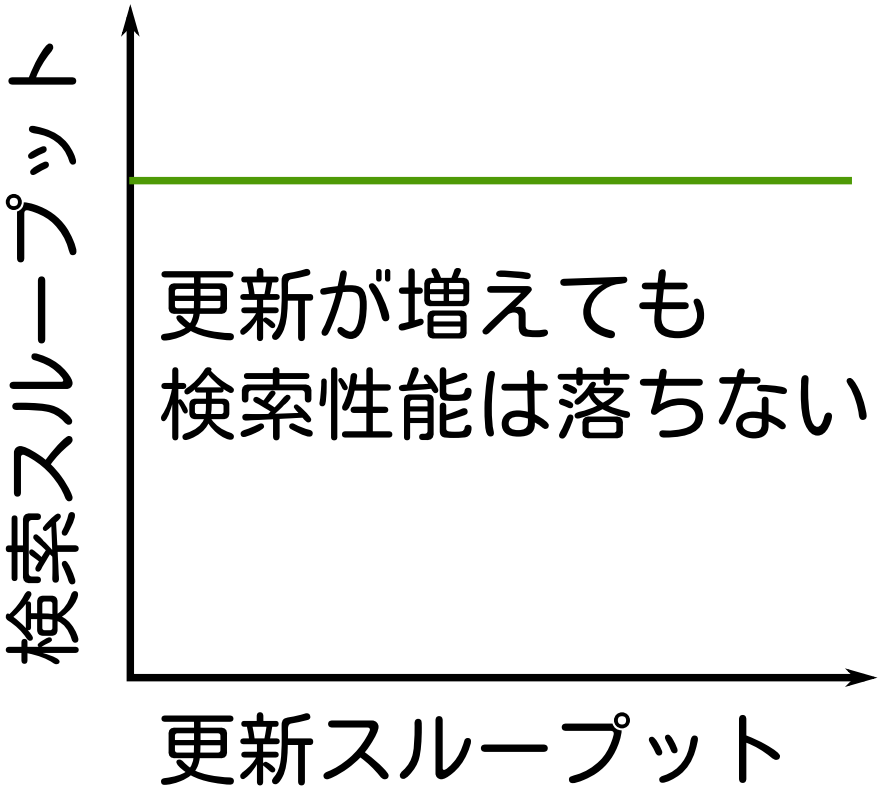
データ：Wikipedia日本語版
約184万レコード・平均サイズ約3.8KB
詳細：http://www.clear-code.com/blog/2015/5/25.html

(*) 他は検索語が3文字以上でこれだけ2文字

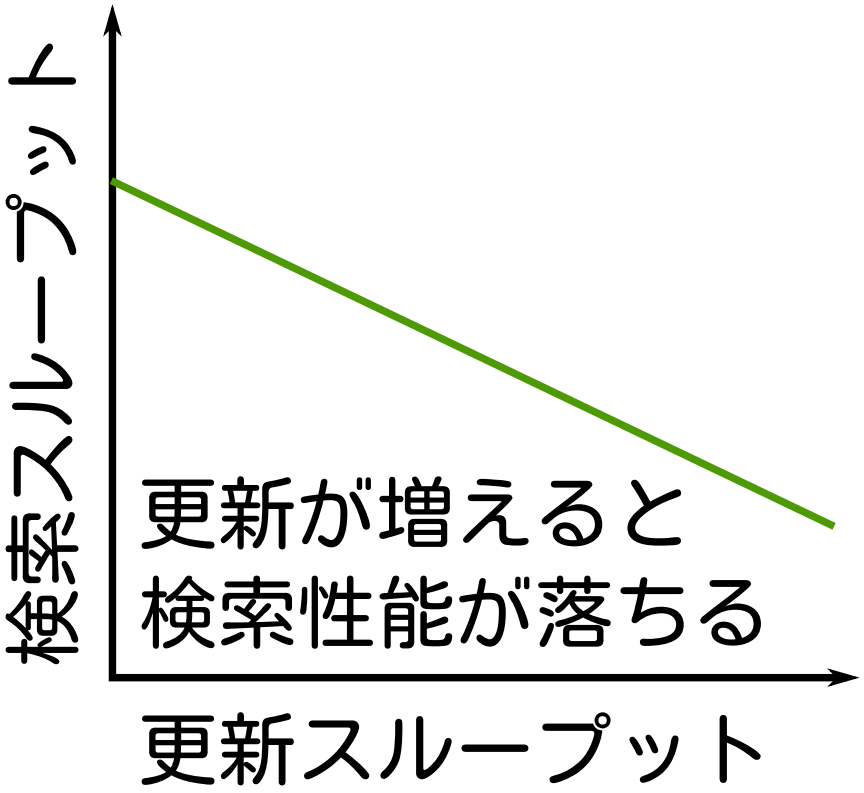
PGroongaは安定して速い！

更新中も性能維持

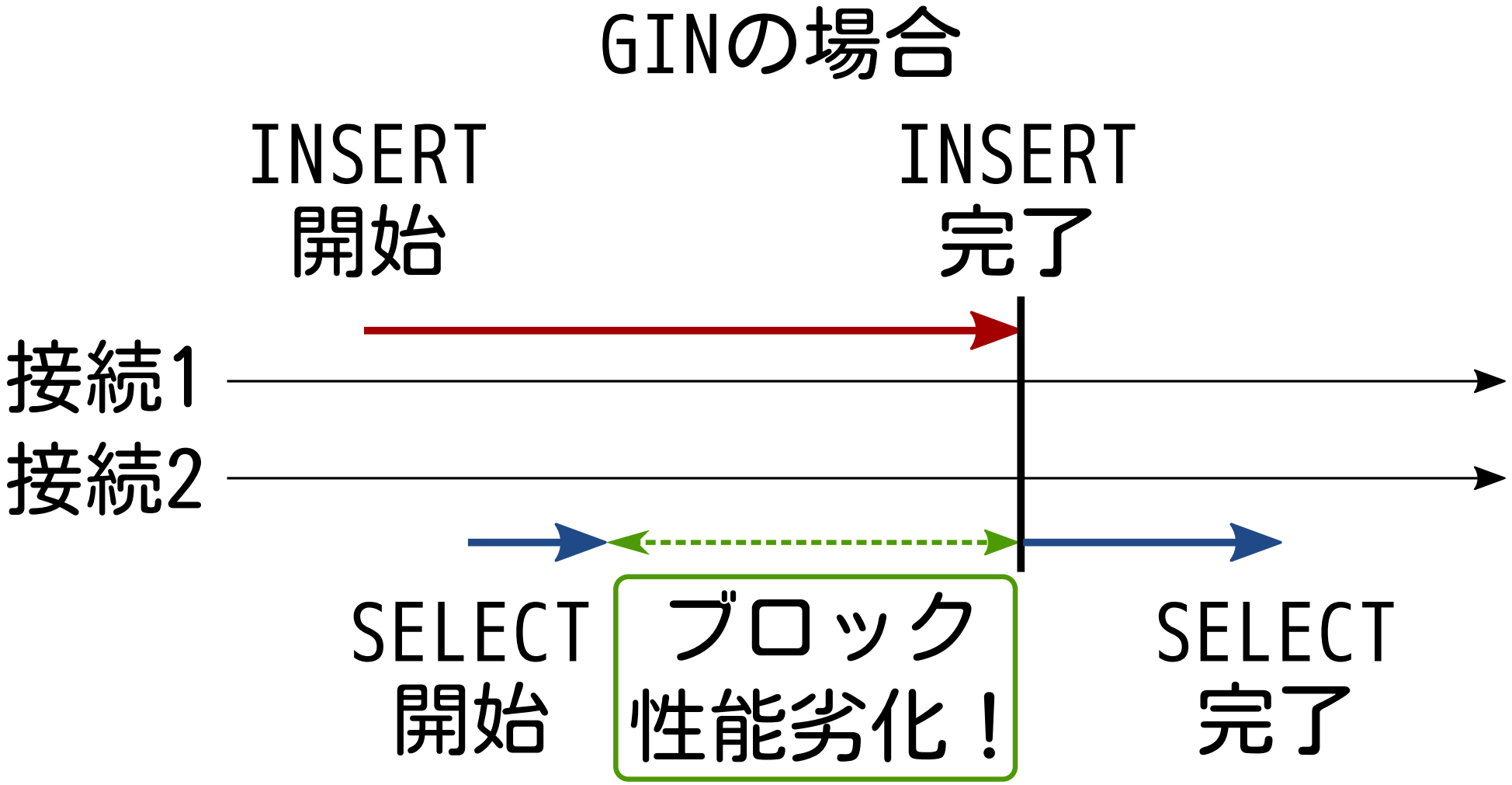
Groonga



更新中は参照不可の
エンジン

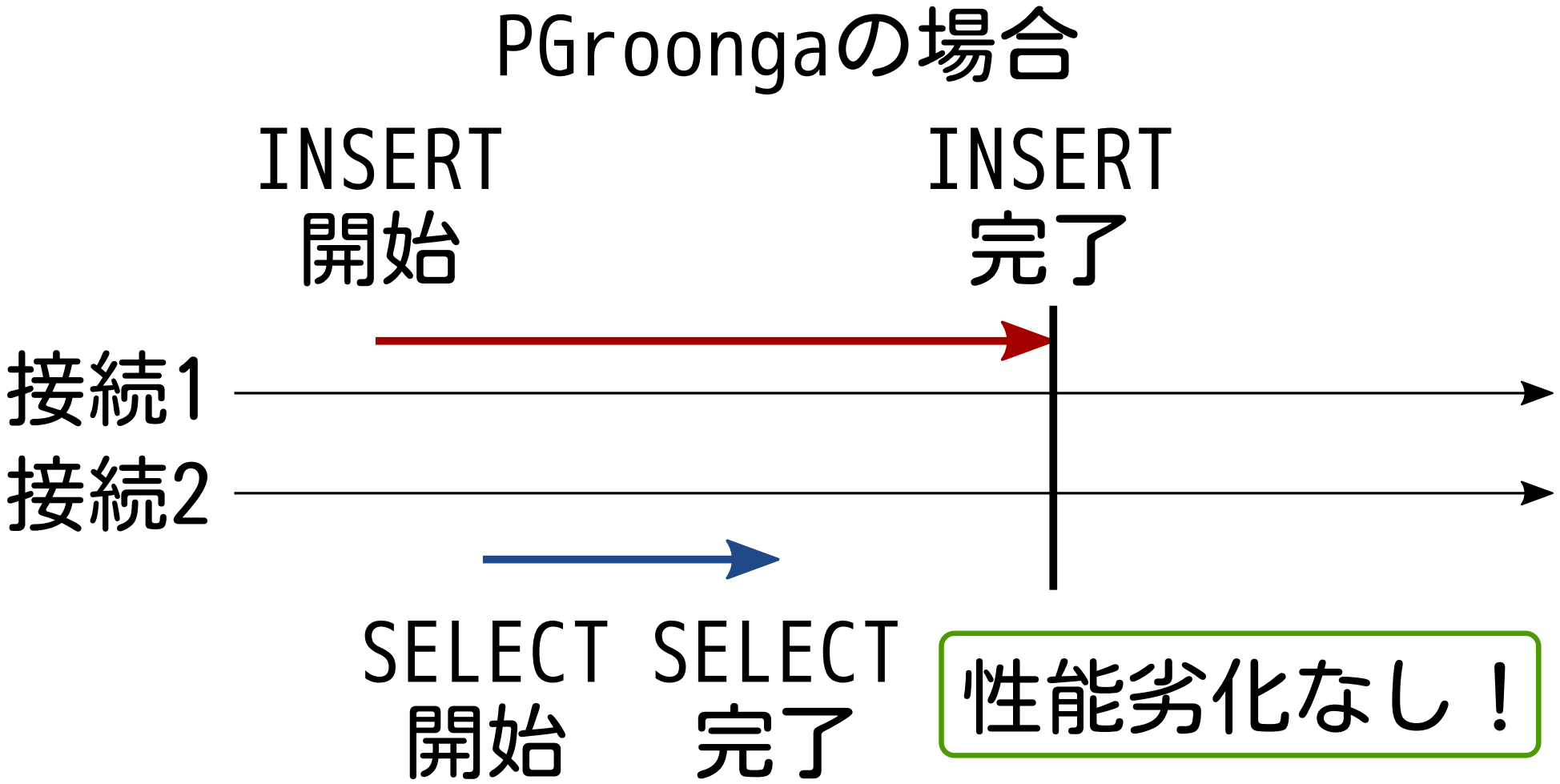


GINと更新と参照



GINはpg_bigmが使っているPostgreSQL標準の転置索引実装

PGroongaと更新と参照



PGroongaはGroongaを使っているので参照ロックフリー

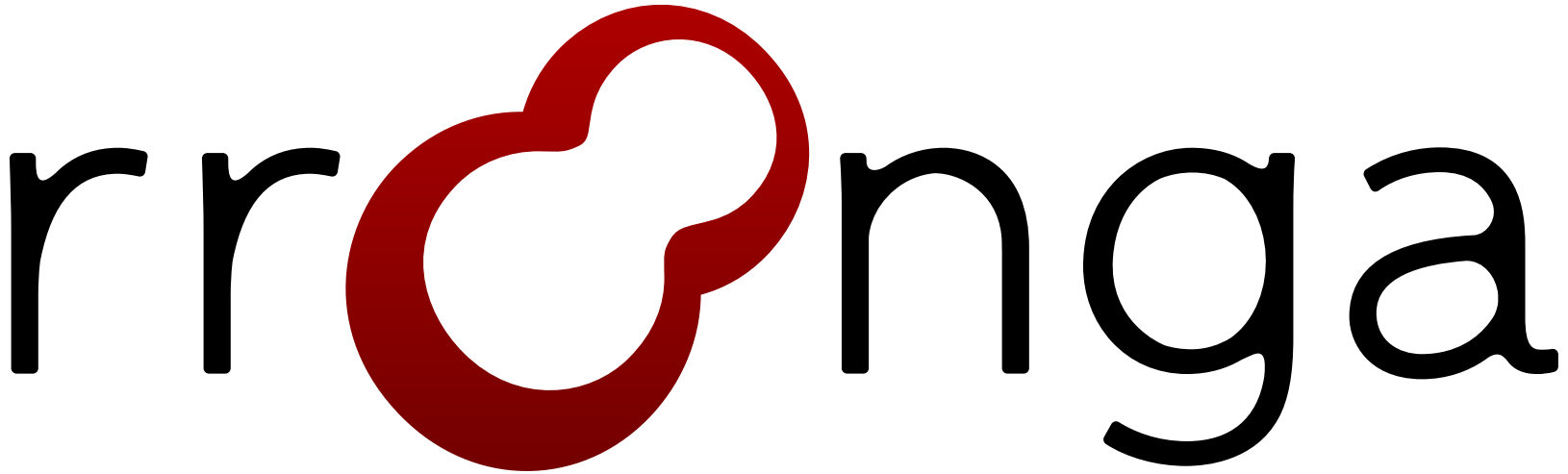
ロード時間も速い

元データの ロード時間	索引 作成時間 PGroonga	索引 作成時間 pg_bigm
16分31秒	25分37秒	5時間56分15秒

データ：Wikipedia日本語版
約184万レコード・平均サイズ約3.8KB
詳細：http://www.clear-code.com/blog/2015/5/25.html

pg_bigmより約14倍速い！

Groonga族：Rroonga



るるんが

Ruby + Groonga

Rroongaのよいところ1

Rubyで書ける

RubyでDB作成

```
Groonga::Database.create(:path => "/tmp/db")
```

Rubyでテーブル定義

```
Groonga::Schema.define do |schema|
  schema.create_table("Users",
                      :type => :hash) do |table|
    table.short_text("name")
    table.int8("age")
  end
end
```

Rubyでデータロード

```
users = Groonga["Users"]
users.add("alice",
          :name => "Alice",
          :age => 29)
```

Rubyでデータロード

```
users = Groonga["Users"]
users.add("alice",
          :name => "Alice",
          :age => 29)
```

Rubyで検索

```ruby
twentys = users.select do |record|
  (record.age >= 20) & (record.age < 30)
end
twentys.each do |twenty|
  p twenty.name # => "Alice"
end
```

Rroongaのよいところ2

サーバーいらず

サーバーいらず

- ローカルのDBを操作
  - SQLiteと同じ
- 向き不向き
  - 向き：1台で扱えるデータ量
  - 不向き：1台で扱えないデータ量

Rroongaの利用事例1

Milkode
ソースコード検索

http://milkode.ongaeshi.me/

Mlikode

- 行指向のソースコード検索
  grepの親戚
- 数万ファイルも余裕
  - grepだと遅いけど1台で扱える量
- 検索方法
  - Rroongaでファイル検索
  - →絞り込んだファイルをgrep
    ポイント：絞り込んだ後ならgrepでも速い！

Rroongaの利用事例2

ROMA
分散KVS

http://roma-kvs.org/

ROMA

- Rakuten/Ruby On-Memory Architecture
- Rubyで実装された分散KVS
- ストレージにGroongaを利用
  Tokyo Cabinetも使える

Rroongaの利用事例3

Droonga

Groonga族：Droonga

どぅるんが

Distributed Groonga

分散対応Groonga

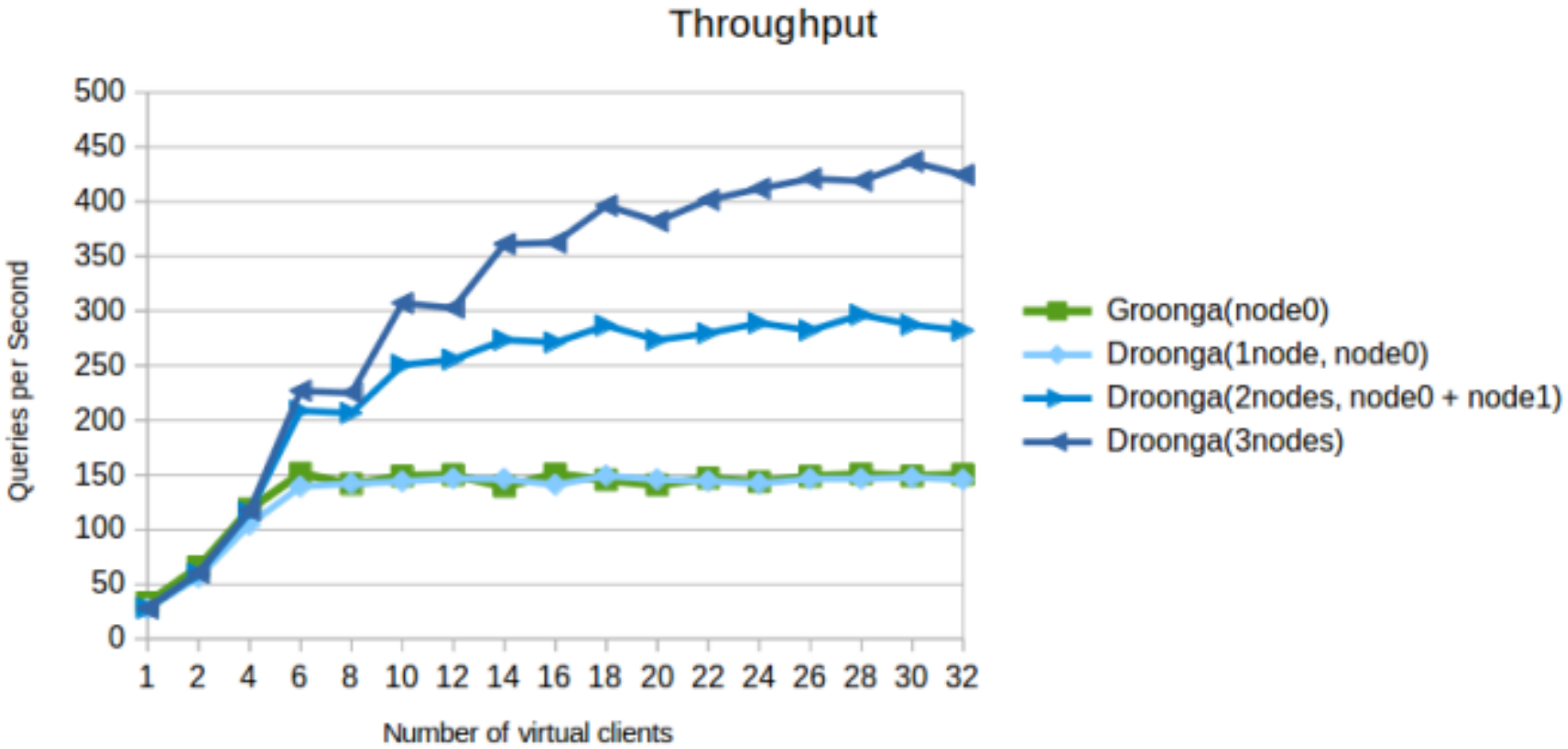
Droongaのよいところ1

分散機能

分散機能

- レプリケーション：実装済み
  ダウンタイムなしでノード追加可能
  - データ複製
  - 参照性能・可用性向上
- シャーディング：実装予定
  - データ分割
  - 更新性能・最大データ量向上

レプリケーション時の性能



Droongaのよいところ2

Groonga互換API

Groonga互換API

- Groongaと同じように使える
  - Groongaからの移行手順あり
    http://droonga.org/ja/tutorial/groonga/
  - DroongaからGroongaも可
- クライアント無変更で動く

概要のまとめ

- Groonga：高速・日本語得意
- Mroonga：高速・簡単
- PGroonga：高速・簡単
- Rroonga：高速・手軽
- Droonga：高速・スケーラブル

最新情報

最新情報：Groonga

ぐるんが

Groonga: 1

groonga-httpdを
正式機能に

Powered by Rabbit 2.1.9

groonga-httpd

- nginxにGroongaを組み込み
  - Groonga族
  - nginxのモジュールとして実装
- Groonga HTTPサーバー互換
- ↑より高機能 (HTTPサーバーとして)
  - 認証・TLS・HTTP/2

Groonga: 2

WindowsでのDBサイズ減少

（1/3以下になるケースもあり）

Groonga: 3

ドリルダウンの高機能化

ドリルダウンの高機能化1

集計処理を追加

- 既存：
  - カウントのみ
- 追加：
  - 合計・最小値・最大値・平均

ドリルダウン+平均

グループでドリルダウンして
グループ毎の平均年齢も計算

```
% curl \
    'localhost:10041/d/select' \
    --get \
    --data-urlencode 'table=users' \
    --data-urlencode 'drilldown=group' \
    --data-urlencode 'drilldown_calc_types=AVG' \
    --data-urlencode 'drilldown_calc_target=age' \
    --data-urlencode 'drilldown_output_columns=_key,_avg'
```

平均の出力例

```
[... [
  [100], # 100グループ
  [
    ["_key", "ShortText"],
    ["_avg", "Float"]
  ],
  [
    ["グループ1", 29.0], # 平均29.0歳
    ["グループ2", 22.9], # 平均22.9歳
    ...
  ]]]
```

ドリルダウンの高機能化2

複数キーでのドリルダウン

↓

クロス集計

例：市と年齢でのクロス集計

市＼年齢	20歳	21歳
札幌市	999人	100人
仙台市	500人	200人

複数キードリルダウン例

市と年齢でドリルダウン

```
% curl \
    'localhost:10041/d/select' \
    --get \
    --data-urlencode 'table=users' \
    --data-urlencode \
      'drilldown[city_age].keys=city,age' \
    --data-urlencode \
      'drilldown[city_age].output_columns=
        _value.city,_value.age,_nsubrecs'
```

出力例

```
[... [
  [100], # 100パターン
  [
    ["city", "ShortText"],
    ["age", "Int8"]
    ["_nsubrecs", "Int32"]
  ],
  [
    ["札幌市", 20, 999], # 札幌市の20歳は999人
    ["札幌市", 21, 100], # 札幌市の21歳は100人
    ["仙台市", 20, 500], # 仙台市の20歳は500人
    ...
  ]]]
```

Groonga: 4

リクエスト キャンセル 対応

リクエストキャンセル

```
% curl \
    'localhost:10041/d/select' \
    --get \
    --data-urlencode 'table=users' \
    --data-urlencode 'request_id=XXX' &
% curl \
    'localhost:10041/d/request_cancel' \
    --get \
    --data-urlencode 'id=XXX'
```

注：リクエストIDを重複無く付けることはクライアントの責任

Groonga: 5

dumpの
ストリーム対応

dumpのストリーム対応

- Before
  - メモリー上で全部作ってから出力
  - 大きなDBではメモリー使用量が多い
- After
  - ダンプ内容を順次出力
  - 大きなDBでもメモリー消費が少ない

Groonga: 6

シャーディング対応

シャーディング対応

- 同一DB内
  - 複数のホストにデータ分散ではない
- 目的
  - テーブルの最大レコード数制限突破
    1テーブルあたり最大約2億レコード
- サポート
  - 月・日でのシャーディング

シャーディング：定義

```
% curl 'localhost:10041/d/table_create' \
    --get \
    --data-urlencode 'name=Logs_20151129' \
    --data-urlencode 'flags=TABLE_NO_KEY'
% curl 'localhost:10041/d/column_create' \
    --get \
    --data-urlencode 'table=Logs_20151129' \
    --data-urlencode 'name=timestamp' \
    --data-urlencode 'flags=COLUMN_SCALAR' \
    --data-urlencode 'type=Time'
```

シャーディング：ロード

```
[
{"timestamp": "2015-11-29 00:00:29"},
{"timestamp": "2015-11-29 00:00:30"},
{"timestamp": "2015-11-29 00:00:31"}
]
```

シャーディング：準備

```
% curl 'localhost:10041/d/plugin_register' \
    --gets \
    --data-urlencode 'name=sharding'
```

シャーディング：検索

```
logical_select \
  --logical_table Logs \
  --shard_key timestamp \
  --min "2015-11-29 00:00:00" \
  --min_border "include" \
  --max "2015-11-30 00:00:00" \
  --max_border "exclude" \
  --filter "..."
```

Groonga: 7

正規表現対応

正規表現

- 構文はRubyと同じ
  - エンジンがRubyと同じだから
- 一部のパターンは索引を使える
  - 前方一致：\A#{リテラル}
  - 中間一致：#{リテラル}
  - 後方一致：#{リテラル}\z

正規表現：使い方

```
select \
  --filter 'content @~ "\\AHello"'
```

- 演算子は@~
- エスケープが煩雑なので注意

Groonga: 8

カスタム
スコア関数対応

スコア関数

- 文書がどのくらいクエリーに適合しているかの度合いを返す関数
- これまではTFのみ
  - TF=単語の出現数
  - =単語の出現数がスコア

新スコア関数

- score_tf_idf
  - TF-IDFベースのスコア関数
  - 単語の出現数を単語の重要度で補正
- score_tf_at_most
  - 上限値付きTFベースのスコア関数
  - スパマー対策ならこれで十分

スコア関数：使い方

```
select Logs \
  --match_columns \
    "scorer_tf_at_most(body, 3.0)" \
  --query "keyword1 keyword2"
```

match_columnsで指定

Groonga: 9

ログ
ローテーション
対応

ログローテーション

- 使い方
  - groongaコマンドの起動オプション
- オプション
  - --log-rotate-threshold-size
  - --query-log-rotate-threshold-size

Groonga: 10

キャッシュの最大キーサイズ増加

キャッシュのキーサイズ

- Before
  - 4KiB
- After
  - 64KiB - 1

Groonga: 11

io_flush追加

io_flush

- メモリー上のデータを
  ディスクに書き込むコマンド
  - 通常はOSに任せている
- 目的：クラッシュ対策
  - UNIX: プロセスが死んでもカーネルが生きていればいずれ書き込まれる
  - Windows: プロセスが死ぬと書き込まれない

Groonga: 12

使った索引を
ログ出力

使った索引をログ出力

- チューニングに便利
  - SQLのEXPLAINの不便バージョン
    サーバーのログを確認しないといけないから
- ログレベルに注意
  - デフォルトでは出力されない
  - debugにすること

Groonga: 13

Windows イベントログ対応

Windowsイベントログ

使い方

```
% groonga --use-windows-event-log
```

Groonga: 14

スレッド数の動的変更対応

スレッド数変更

最大8スレッドに変更する例

```
% curl \
    'localhost:10041/d/thread_limit' \
    --get \
    --data-urlencode 'max=8'
```

Groonga: 15

column_copy追加

column_copy

- カラムの値をコピー
  - サーバーサイドで実行
- 利用例
  - テーブル・カラム定義変更
  - 新カラム作成→コピー→
  - 旧カラム削除→リネーム

Groonga: 16

Pretty print 対応

従来

```
% curl 'localhost:10041/d/status'
[[0,...],{"alloc_count":248,...}]
```

Pretty print

```
% curl \
    'localhost:10041/d/status' \
    --get \
    --data-urlencode 'output_pretty=yes'
[
  [
    0,
    ...
  ],
  {
    "alloc_count": 248,
    ...
  }
]
```

Groonga: 17

新コマンド

新コマンド

- reindex
  - 既存インデックスを再作成
- schema
  - スキーマを返す
- thread_limit
  - 最大スレッド数を動的に変更

Groonga: 18

新関数

新関数

- prefix_rk_search()
  - ローマ字で検索
  - selectで補完できる

Groonga: 19

Groonga Admin 進化

Groonga Admin

- Groongaの新しい管理画面
  - 旧：Webからコマンドを実行できる
  - 新：詳細を知らずに使える
- 試せる
  - http://packages.groonga.org:10041/

最新情報：Mroonga

むるんが

MySQL + Groonga

Mroonga: 1

```
mroonga_boolean_mode_syntax_flags
```

- IN BOOLEAN MODEでの構文を変更
  - スクリプト構文→使える演算子が増加

スクリプト構文

```
SET mroonga_boolean_mode_syntax_flags =
  "SYNTAX_SCRIPT";

SELECT * FROM memos
  WHERE MATCH(title)
        AGAINST('title @~ ".+roonga"'
                IN BOOLEAN MODE);
```

正規表現を使う例
他にも関数を使えたり比較演算子を使えたりする
MATCHに指定していないカラムでも検索できる

Mroonga: 2

MariaDB カスタム パラメーター 対応

インデックスパラメーター

```
CREATE TABLE memos (
  body text,
  FULLTEXT INDEX body_index (body)
    TOKENIZER='TokenMecab'
    NORMALIZER='NormalizerAuto'
    TOKEN_FILTERS='TokenFilterStopWord'
    -- ↑COMMENTに書かなくてよい！
) ENGINE=Mroonga DEFAULT CHARSET=utf8mb4;
```

カラムパラメーター

```
CREATE TABLE tags (
  name VARCHAR(64) PRIMARY KEY
) ENGINE=Mroonga DEFAULT CHARSET=utf8mb4;

CREATE TABLE bugs (
  tag VARCHAR(64) GROONGA_TYPE='tags'
) ENGINE=Mroonga DEFAULT CHARSET=utf8mb4;
```

Mroonga: 3

MySQL 5.7対応

MySQL 5.7対応

- MySQL 5.7でビルドできる
- JSON型対応

JSON型

```
CREATE TABLE logs (
  record JSON
) ENGINE=Mroonga
  DEFAULT CHARSET=utf8mb4;
INSERT INTO logs
  VALUES ('{"message": "start"}');
```

Mroonga: 4

MariaDB 10.1 対応

最新情報：Droonga

どぅるんが

Distributed Groonga

分散対応Groonga

Droonga: 1

ダウンタイム
なしの
ノード追加対応

今年のまとめ

- Groonga：たくさん改良
- Mroonga
  - 使い勝手改良・最新対応
- PGroonga: 今年デビュー
- Rroonga：最新対応
- Droonga：可用性向上

今後

ヒント

- 実現は保証しない
  - 実現しようとはするよ！
- 実現可能性を高めるには…
  - 開発に参加
    コード・ドキュメントを書く・テストする・宣伝する
  - 開発者が開発する時間を増やす
    他のユーザーをサポート・情報公開・仕事を頼む

今後：Groonga

ぐるんが

Groonga: 1

カラムストアをもっと活かす

- シーケンシャルサーチを速く！
- ソートを速く！
- ドリルダウンを速く！

↓

grn_ts（ぐるんたす）

Groonga: 2

mrubyをデフォルト有効

- プラグインをRubyで書ける
- 式を書き換えられる
- オプティマイザーを書ける

Groonga: 3

プラグインエコシステム

- プラグイン管理コマンド提供
  - 例：grn plugin install XXX
- プラグインリポジトリー提供
  GoのようにGitから直接インストールでもいいかも
- 対応プラグイン
  - Rubyで書かれたプラグイン

Groonga: 4

位置指定マッチ対応

- N番目のトークンならマッチ
  - 例：1番目のトークンはhello？
  - マッチする：hello world
  - マッチしない：hey! hello world

位置指定マッチ応用例1

正規表現の前方一致検索

- Groongaでの構文：\A
  - 例：\Ahello
- 実装：
  - 1番目のトークンかチェック

位置指定マッチ応用例2

ベクターのN番目の要素にマッチ

```
{
  "rank": ["Alice", "Bob", "Chris"]
}
# rank[0] @ "Bob"→マッチしない
# rank[1] @ "Bob"→マッチする
```

Groonga: 5

selectで計算結果での ドリルダウン・ソートに対応

- 例
  - 元データ：日時 (2015-11-29 13:30)
  - 計算結果：年月日 (2015-11-29)
  - ドリルダウン：計算結果を使用
    (2015-11-29: 1件、2015-11-30: 2件)

select例

```
select Logs \
  --column[access_day].type Time \
  --column[access_day].source \
    'time_floor_year(access_time)' \
  --drilldown access_day
```

Groonga: 6

JSONでのパラメーター指定
（も使えるようにする）

- メリット
  - プログラムから指定しやすい
    ことがあるかもしれない
- デメリット
  - クエリーログに残らない

クエリーストリング

現行

```
% curl \
    'localhost:10041/d/select' \
    --get \
    --data-urlencode 'table=users' \
    --data-urlencode 'limit=100'
```

JSON

```
% cat select.json
{
  "table": "users",
  "limit": "100"
}
% curl \
    'localhost:10041/d/select' \
     --header 'Content-Type: application/json' \
     --data-binary @select.json
```

Groonga: 7

ドキュメント改善

- 未ドキュメントな項目を書く
- 古いドキュメントを更新

今後：Mroonga

むるんが

MySQL + Groonga

Mroonga: 1

ラッパーモードで ロールバック対応

- 現状：ロールバック非対応
  （変更が残ってしまい検索結果に不整合発生）
  - 今後：コミット時にまとめて更新
    （ロールバックしても変更が残らない）
- 現状：トランザクション中に 更新データを検索可能
  - 今後：検索不可能

Mroonga: 2

MariaDBで
テーブルパラメーター対応

```
CREATE TABLE xxx (
) ENGINE=Mroonga
  COMMENT='ENGINE "InnoDB"'
  -- ↑を↓と書ける
  WRAP ENGINE=InnoDB;
```

Mroonga: 3

generated column対応

MySQL 5.7で導入。MariaDBでいうvirtual column。

```
CREATE TABLE users (
  first_name text,
  last_name text,
  full_name text AS
    (CONCAT_WS(' ', first_name, last_name))
    STORED
) ENGINE=Mroonga;
```

Mroonga: 4

ネイティブパーテション対応

MySQL 5.7で導入。FULLTEXT INDEXを使えるようになる！

```
CREATE TABLE memos (
  created_time DATETIME,
  content text,
  FULLTEXT INDEX (content)
) ENGINE=Mroonga
  PARTITION BY RANGE (TO_DAYS(timestamp)) (
    -- ...
  );
```

Mroonga: 5

JSONの全文検索対応

```
CREATE TABLE logs (
  message JSON,
  FULLTEXT INDEX (message)
) ENGINE=Mroonga;
SELECT * FROM logs
  WHERE MATCH(message)
        AGAINST('*D+ error' IN BOOLEAN MODE);
```

Mroonga: 6

同義語展開対応

```
SELECT * FROM memos
  WHERE MATCH(content)
        AGAINST(mroonga_expand_query('*D+ 焼き肉')
                IN BOOLEAN MODE);
-- → AGAINST('*D+ 焼き肉 OR 焼肉' IN BOOLEAN MDOE)
-- 同義語はMroongaのテーブルにして
-- SQLで管理できるようにする
```

Mroonga: 7

ドキュメント改善

- Sphinx→GitHub Pages
  - Sphinx・reSTはオーバースペック
  - Markdownで十分
  - typoを見つけても正しく直せない

今後：PGroonga

ぴーじーるんが

PostgreSQL + Groonga

PGroonga: 1

text @@ pgroonga.queryを導入

```
SET enable_indexscan = off;
SET enable_bitmapscan = off;
SELECT * FROM WHERE content @@ '全文検索';
-- ↑PostgreSQL組み込みの@@が使われる
SELECT * FROM
  WHERE content @@ '全文検索'::pgroonga.query;
-- ↑PGroonga提供の@@が使われる
```

PGroonga: 2

もっとGroongaを活かす

PGroonga	pg_bigm	Groonga
0.646s	0.556s	**0.085s**

ヒット数635,792、検索語は2文字

生Groongaは1桁速い！

詳細：https://github.com/groonga/wikipedia-search/issues/3

PGroonga: 3

同義語展開対応

```
body @@ pgroonga.expand_query('ネジ')
-- ↓
body @@ 'ネジ OR ねじ OR ボルト'
```

PGroonga: 4

ステミング対応

found/finds→find

```
CREATE INDEX index ON entries
  USING pgroonga (title)
    WITH (token_filters = 'TokenStem');
```

PGroonga: 5

重み対応

```
-- タイトルのほうが本文より10倍重要
body @@ ('title * 10 || body', 'ポスグレ')
```

PGroonga: 6

複合主キー対応

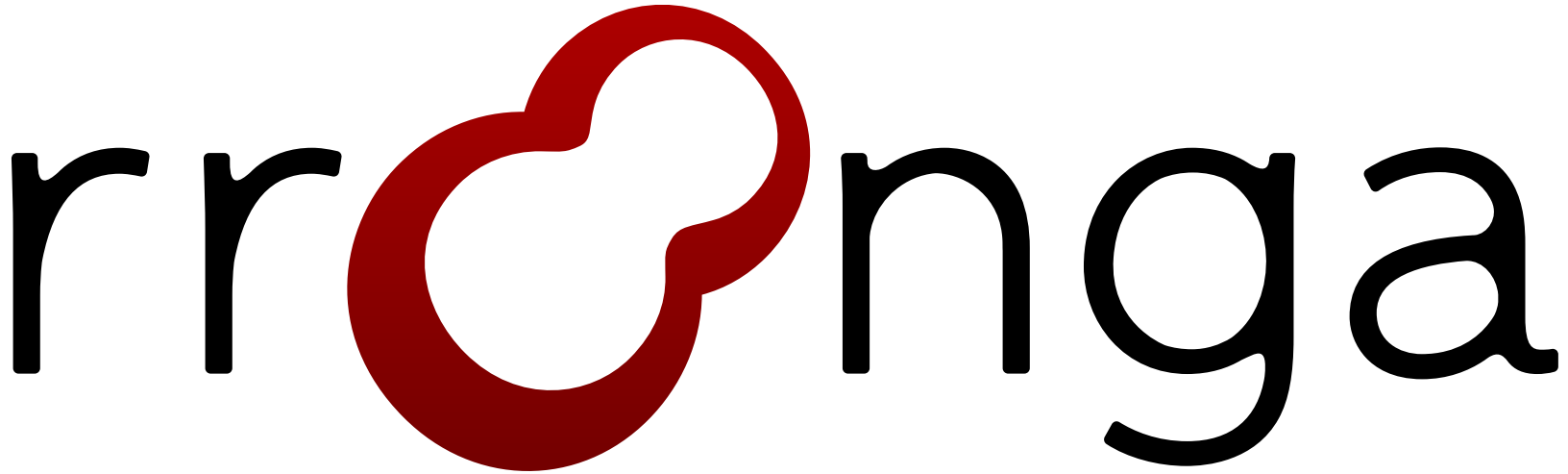
```
CREATE TABLE t (
  c1 INT,
  c2 INT,
  PRIMARY KEY (c1, c2)
);
CREATE INDEX index ON t
  USING pgroonga (c1, c2);
```

PGroonga: 7

pg_dumpのWITH問題を解消

```
CREATE INDEX index ON t
  USING pgroonga (c)
    WITH (tokenizer = 'TokenMecab');
-- ↓pg_dump: クォートがとれる→小文字に正規化される
CREATE INDEX index ON t
  USING pgroonga (c)
    WITH (tokenizer = TokenMecab);
```

今後：Rroonga



るるんが

Ruby + Groonga

Rroonga: 1

ドキュメント改善

- Textile→Markdown
- 英語化
- 説明追加（主にコード例）

今後：Droonga

どぅるんが

Distributed Groonga

分散対応Groonga

Droonga: 1

シャーディング対応

Groonga族以外

- キラーアプリが欲しい
- 世界規模で認知されたい
  - Mroonga・PGroongaをきっかけに
    MroongaはMariaDBにバンドルされている
- 本を出したい
  - 初心者向けのやつ

まとめ

- Groonga族は今年も進化した
- Groonga族は来年も進化する
- 来年はもっと世界規模で
  ユーザーを増やしたい

おしらせ1

- Groongaで学ぶ全文検索
  ↑で検索！
  - 隔週金曜の夜開催
  - 予習復習なしで時間内で
    効率よく全文検索を学ぶ会
  - 参加者のレベルは問わない
  - 内容は毎回参加者に合わせて決める

全文検索を学びたい方はどうぞ！

おしらせ2

MySQLとPostgreSQLと
日本語全文検索

来年2月9日（肉の日）
DMM.comラボにて