# PGroongaの実装

ぴーじーるんがのじっそう

須藤功平　株式会社クリアコード

PostgreSQLカンファレンス2015
2015-11-27

# （建前の）目的

PostgreSQLの
amindexを説明

# 方法

# PGroongaの実装を紹介

PGroongaはamindexとして実装されているため
ただし、ヒントだけなので詳細はPGroongaのソースを参照
https://github.com/pgroonga/pgroonga

# 本当の目的

PGroongaの自慢

# PGroongaとは

amindexの一種

# amindex

# 索引の実装をPostgreSQLに追加する仕組み

# amindexの使い方

```
CREATE INDEX name ON table
  USING pgroonga (column);
```

組み込みの索引と同じ
(USINGを指定するだけ)

# PGroongaの提供機能

索引を使った
**超高速な**
全文検索機能

# PostgreSQLと全文検索

課題あり

# PostgreSQLと全文検索1

```
CREATE INDEX name ON table
  USING gin (to_tsvector('english', column));
SELECT * FROM table
  WHERE to_tsvector('english', column) @@ '...';
```

## 組み込みの全文検索機能
## 日本語非対応

http://www.postgresql.org/docs/current/static/textsearch.html

# PostgreSQLと全文検索2

```
CREATE INDEX name ON table
  USING gin (column gin_trgm_ops);
SELECT * FROM table
  WHERE column LIKE '%...%';
```

contrib/pg_trgm
日本語非対応

http://www.postgresql.org/docs/current/static/pgtrgm.html

# PostgreSQLと全文検索

日本語非対応

# PGroongaとPostgreSQL

日本語対応！

# PGroongaを使う

```
CREATE INDEX name ON table
  USING pgroonga (column);
SELECT * FROM table
  WHERE column @@ '全文検索';
```

日本語対応！

しかも
速い！

# ヒット数と検索時間

| ヒット数 | 検索時間 |
|---:|---:|
| 368 | 0.030s |
| 17,172 | 0.121s |
| 22,885 | 0.179s |
| 625,792 | 0.646s(*) |

(*) work_memを10MBに増やしている

## 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

# 検索時間：比較

| 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** |

(*) 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！

# なぜ速いのか

バックエンドが
外部の本格的な
全文検索
ライブラリー

# amindexでのポイント

- _PG_init()
  - ライブラリーを初期化
- on_proc_exit()
  - 後始末

# 全文検索ライブラリー

## Groonga
## （ぐるんが）

# Groonga

## 本格的な全文検索エンジン

サーバーとしてもライブラリーとしても使える

# 本格的な例1

長い文書でも検索性能が落ちない

この特徴が有用なサービス例：

- Wiki
  - イメージ：Wikipedia
- ドキュメント検索
  - イメージ：ファイルサーバー検索

# 長い文書でも速い理由

完全
転置索引

# 転置索引

- 完全：位置情報あり
  - Groonga
- 無印：位置情報なし
  - GIN
  - Groonga（必要なければ入れないことができる）

# 転置索引の違い

文書

| ID | 内容 |
|---|---|
| 10 | 全文検索 |
| 20 | 全文を検索 |

分解

全文|検索
1　2

全文|を|検索
1　2　3

完全転置索引（文書ID+位置）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10:1, 20:1 |
| 検索 | 10:2, 20:3 |
| を | 20:2 |

転置索引　（文書IDのみ）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10, 20 |
| 検索 | 10, 20 |
| を | 20 |

# 索引での検索の違い

クエリー

全文検索

↓ 分解

全文 | 検索

1　2

文書

| ID | 内容 |
|---|---|
| 10 | 全文検索 |
| 20 | 全文を検索 |

完全転置索引（文書ID+位置）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10:1, 20:1 |
| 検索 | 10:2, 20:3 |
| を | 20:2 |

隣接

AND → 隣接チェック → 10 結果

転置索引　（文書IDのみ）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10, 20 |
| 検索 | 10, 20 |
| を | 20 |

AND → 10, 20 結果候補

# 検索速度の違い

- 完全転置索引：安定して速い
  - 索引だけで検索完了
- 転置索引：速さが安定しない
  - 索引での検索+全件スキャン
  - 候補文書が多い・長い→遅くなる

# amindexでのポイント

- 全件スキャンを無効にする
  - scan->xs_recheck = false
  - tbm_add_tuples(..., false)

# 本格的な例2

常時更新・常時検索に強い

この特徴が有用なサービス例：

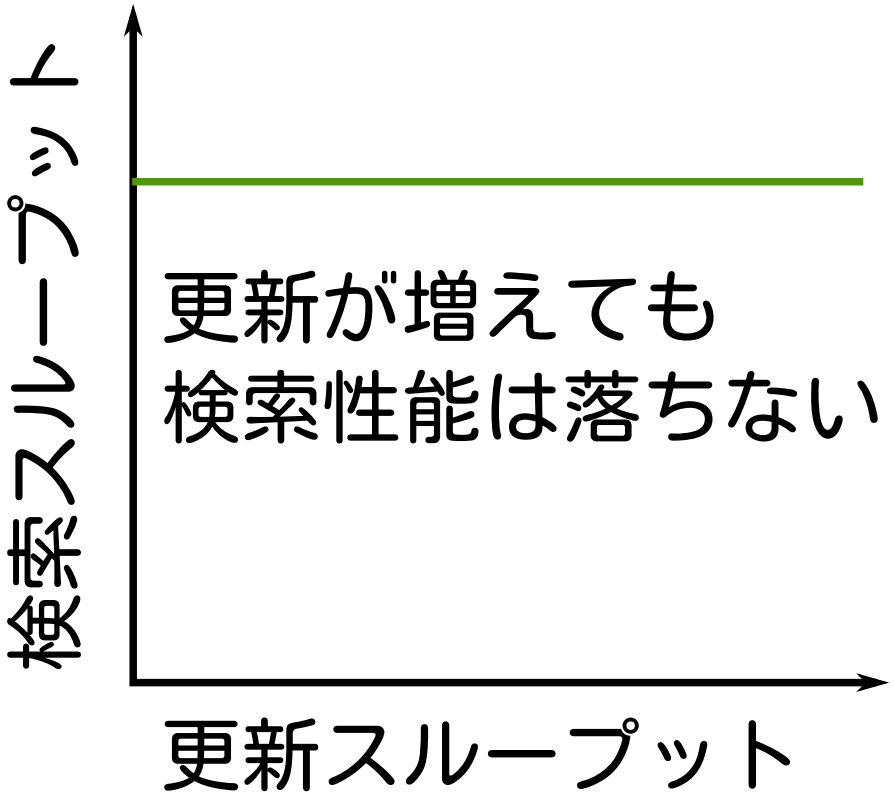
- SNS
  - イメージ：Twitter
- ナレッジ共有サービス
  - イメージ：Qiita・teratail

# 常時更新・検索に強い？

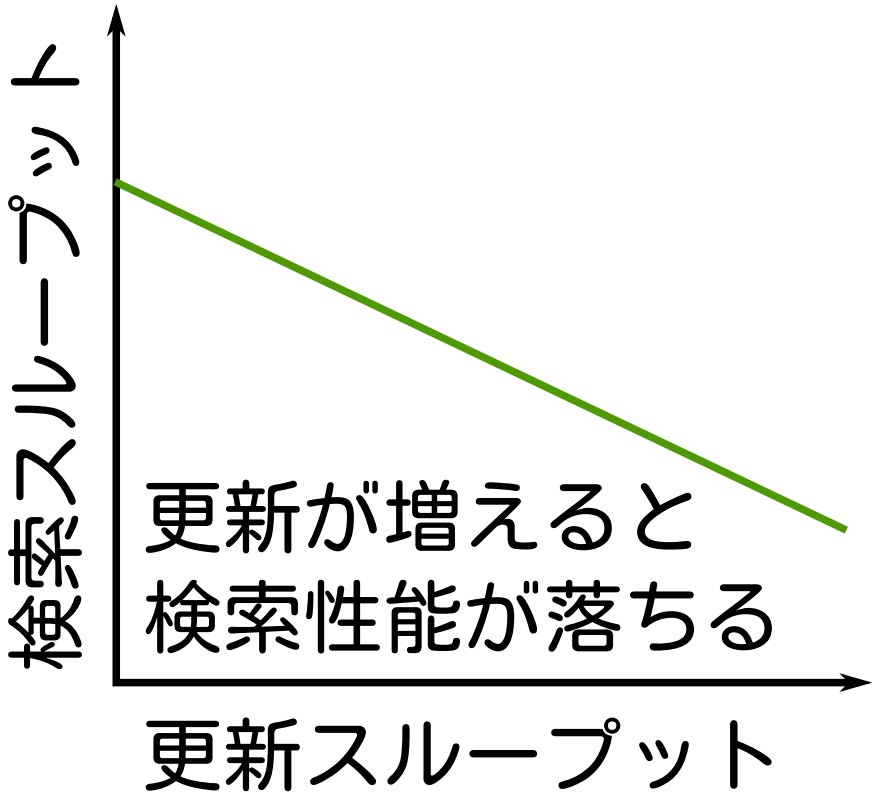
更新中も
参照性能が
落ちない

# 性能傾向

Groonga




GIN



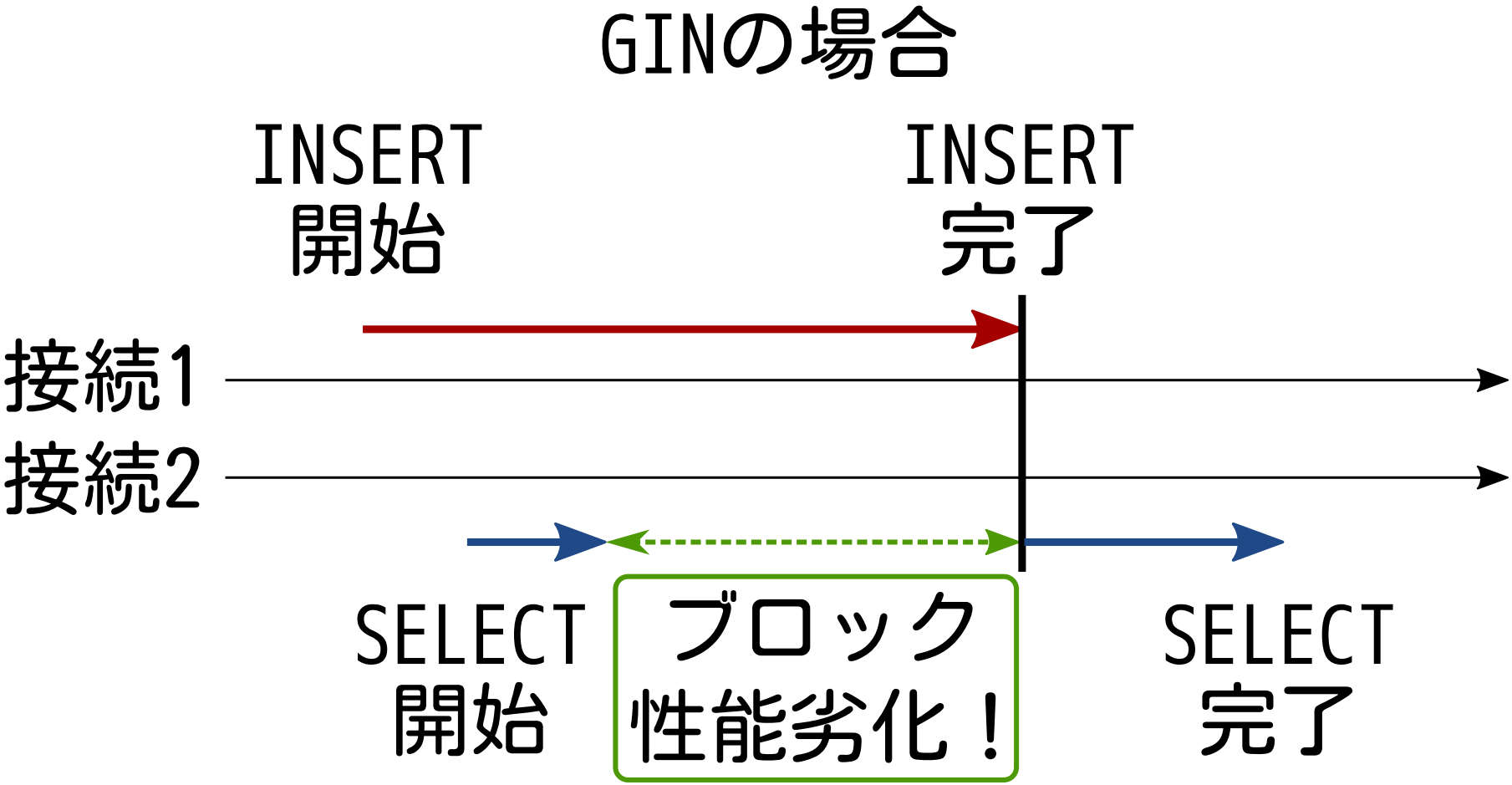
# 落ちない理由

更新時に
参照ロックなし

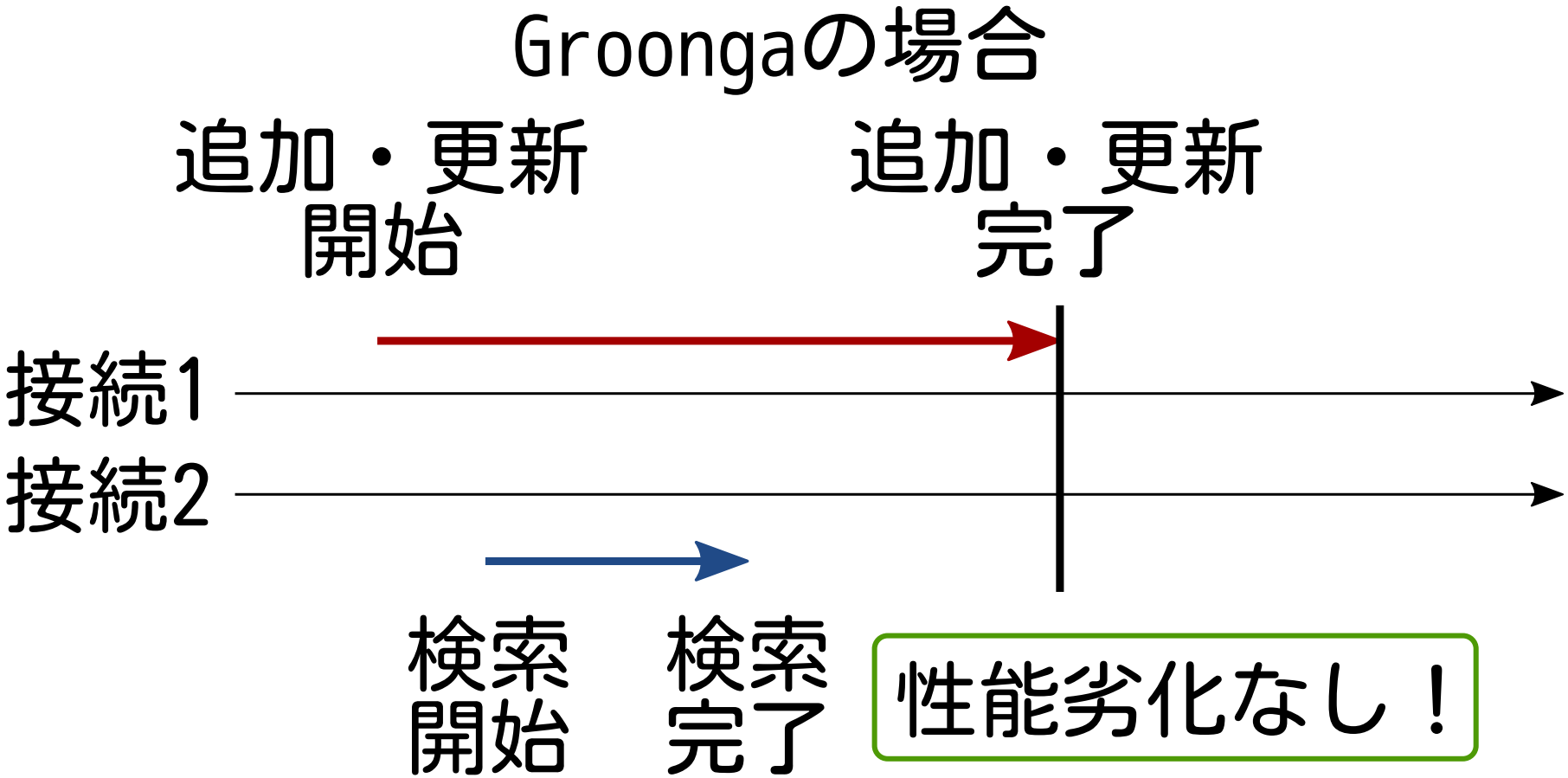
# 参照ロック

獲得したら
他の処理は
参照不可になる
ロック

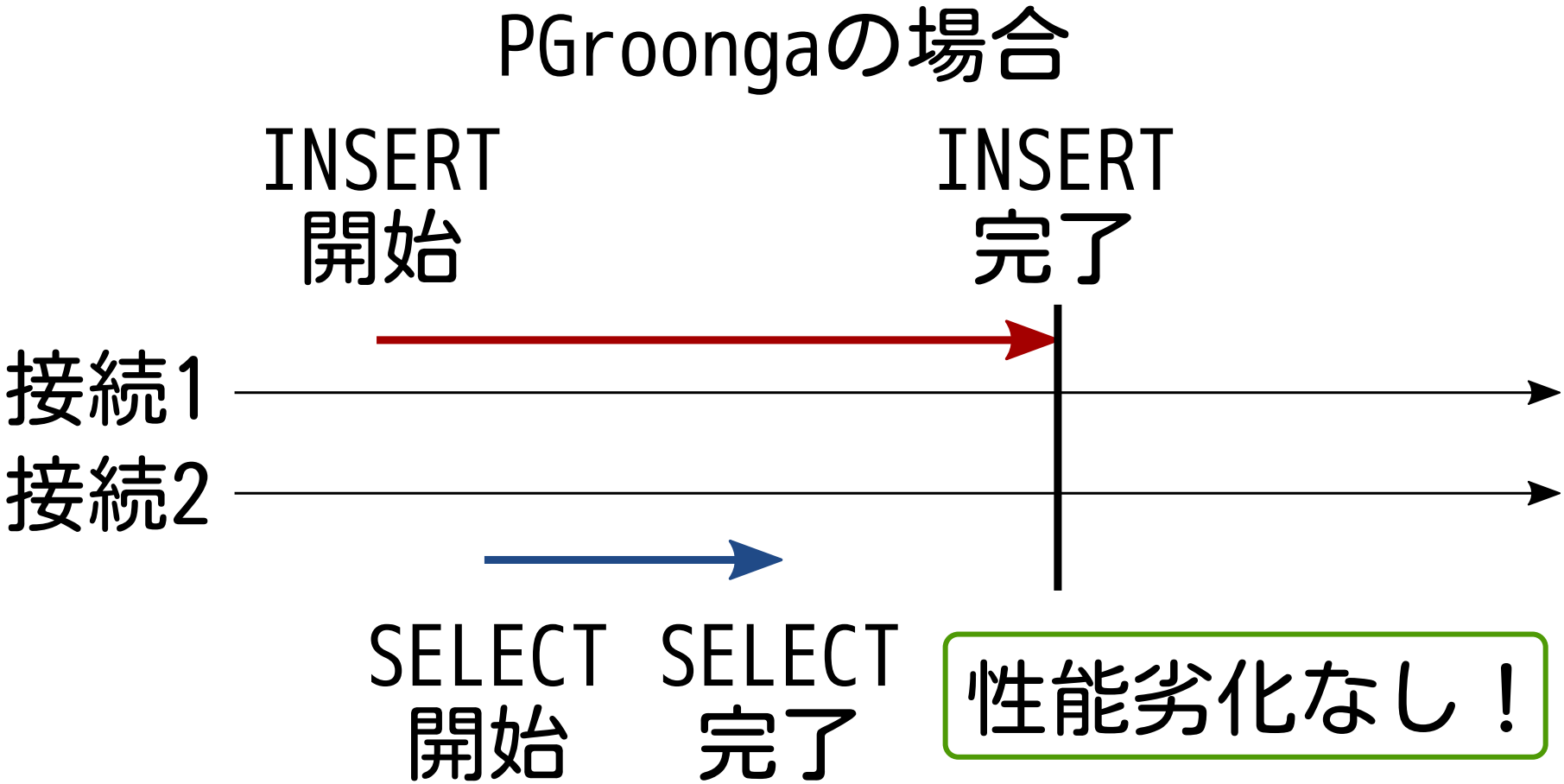
# GINと更新と参照


GINの場合
INSERT
開始
INSERT
完了
接続1
接続2
SELECT
開始
ブロック
性能劣化！
SELECT
完了

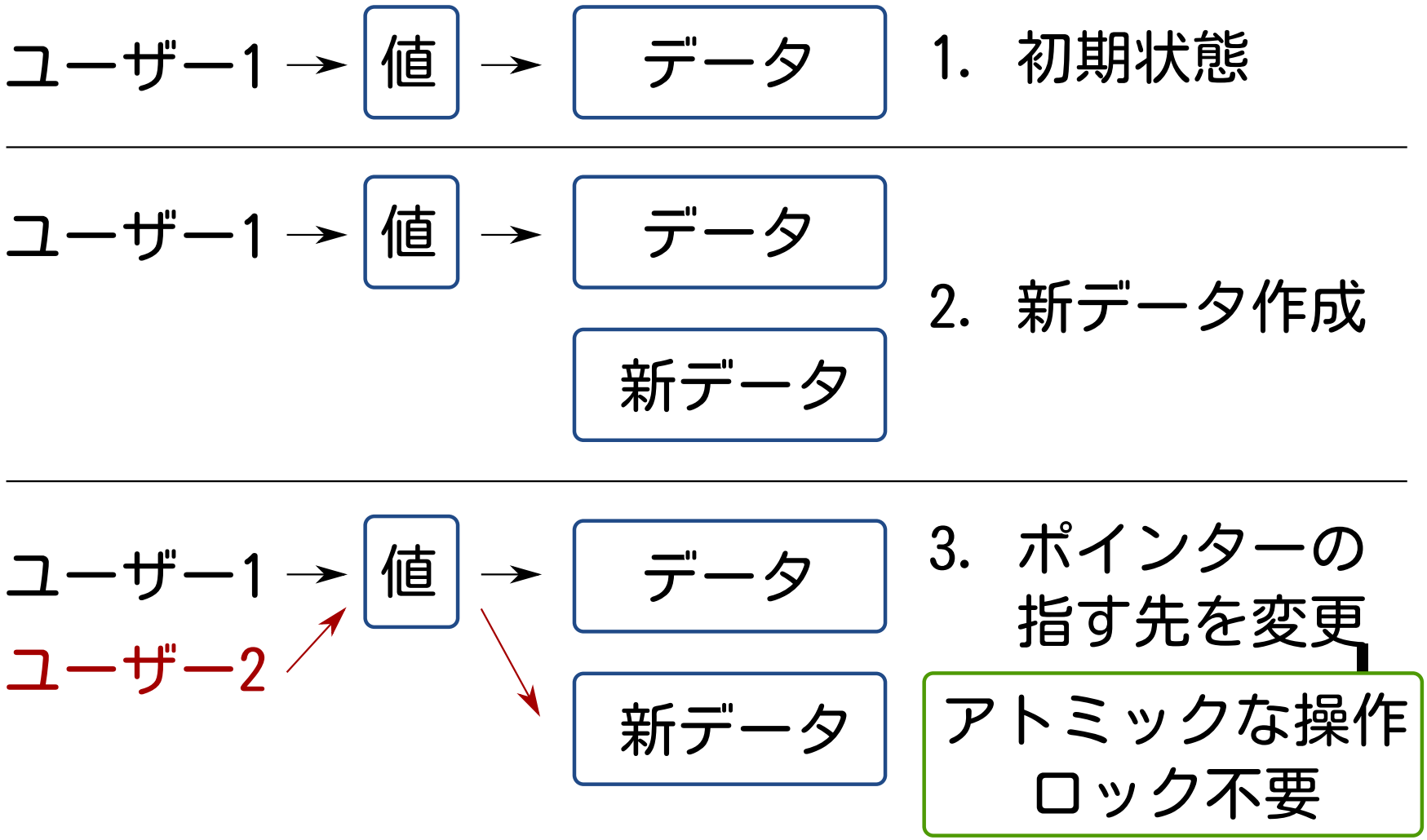
# Groongaと更新と参照



# PGroongaと更新と参照



# 参照ロックフリーの実現



# amindexでのポイント

- Lock*()(*)を呼ばない
  (*) PostgreSQL提供のロックAPI
  - 呼ぶとGroongaの参照ロックフリーの有意性を殺してしまう

# 本格的な例3

## 継続的な更新に強い

この特徴が有用なサービス例：

- SNS
  - イメージ：Twitter・Facebook
- チャット
  - イメージ：Slack

# 継続的な更新に強い？

- 間欠的な性能劣化がない
  - 更新も検索も
- GINは両方ある
  - FASTUPDATEを無効にしていない場合

# 間欠的性能劣化：Groonga

Groongaは間欠的性能劣化なし

- 常に最新ポスティングリストを維持しているから
  - 更新負荷が高くならない対策入り
    https://github.com/groonga/groonga/wiki/Memo

# Groongaの索引更新

文書

| ID | 内容 |
|---|---|
| 10 | 全文検索 |
| 20 | 全文を検索 |

分解

全文|検索
1　2

全文|を|検索
1　2　3

完全転置索引（文書ID+位置）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10:1, 20:1 |
| 検索 | 10:2, 20:3 |
| を | 20:2 |

更新を続けても
最新状態を維持

# 間欠的性能劣化：GIN

GINは間欠的性能劣化あり

- 最新ポスティングリスト維持をサボって高速化しているから
  - サボったつけを払うときに性能劣化
  - 例：検索時・更新が溜まりすぎた時

# GINの索引更新

文書

| ID | 内容 |
|---|---|
| 10 | 全文検索 |
| 20 | 全文を検索 |

分解

全文|検索
1　2

全文|を|検索
1　2　3

転置索引　（文書IDのみ）

| キーワード | ポスティングリスト |
|---|---|
| 全文 | 10 |
| 検索 | 10 |

とりあえず放り込む

最新状態維持をがんばらない

ペンディングリスト

溢れた!

| 全文 20 | を 20 | 検索 20 | ・・・ |
|---|---|---|---|

# GINの索引検索

クエリー

全文検索

↓ 分解

全文 | 検索

1　2

転置索引　（文書IDのみ）

| キーワード | ポスティングリスト |
| --- | --- |
| 全文 | 10 |
| 検索 | 10 |

マージ → 10, 20

結果候補

文書

| ID | 内容 |
| --- | --- |
| 10 | 全文検索 |
| 20 | 全文を検索 |

ペンディングリスト

| 全文 20 | を 20 | 検索 20 | |
| --- | --- | --- | --- |

# 本格的な例4

索引の作り直しが速い

この特徴が有用なケース：

- ダンプのリストア
- サービス復旧

# 索引作成

| 元データの<br>ロード時間 | 索引<br>作成時間 |
|---|---|
| 16分31秒 | 25分37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

# 索引作成：比較

| PGroonga | pg_bigm |
| --- | --- |
| 25分37秒 | 5時間56分15秒 |

pg_bigmより約14倍速い！

# 速い理由

## 静的
## 索引構築を
## サポート

# 索引構築方法

- **動的**索引構築
  - 構築中も完了した分は検索可能
  - 即時反映可・一括登録不向き
- **静的**索引構築
  - 構築完了まで使えないが速い
    （処理時間は入力に比例。指数関数的ではない。）
  - 即時反映不可・一括登録向き

# SQLの違い

```
-- 動的索引構築
CREATE INDEX ...;
INSERT ...;
-- 静的索引構築
INSERT ...;
CREATE INDEX ...;
```

# amindexでのポイント

- aminsert()
  - 動的索引構築を実装
- ambuild()
  - 静的索引構築を実装

# 速い理由のまとめ

- 索引だけで検索可能
- 更新中も検索可能
- 間欠的な性能劣化なし
- 静的索引構築をサポート

# 速さ以外の利点1

## 便利な独自機能

# 独自機能1

## 見慣れた
## クエリー言語

# 例

```
body @@ 'PostgreSQL OR MySQL -Oracle'
```

- Web検索エンジン互換
  - →ユーザーの入力をそのまま使える
- デフォルトAND
- OR・-（除外）あり

# 独自機能2

## 配列+全文検索

# 例

```
CREATE TABLE logs (hosts text[]);
INSERT INTO logs
  VALUES (Array['web', 'db']);
CREATE INDEX index ON logs
  USING pgroonga (hosts);
SELECT * FROM logs
  WHERE hosts @@ '各ホスト名を全文検索';
```

# 独自機能3

# JSON+全文検索

# 例

```
INSERT INTO logs (record)
  VALUES ('{"host": "ダウンホスト"}'),
         ('{"message": "シャットダウン"}');
SELECT * FROM logs
  WHERE record @@ 'string @@ "ダウン"'
--              record
-- ----------------------------
-- {"host":    "ダウンホスト"}
-- {"message": "シャットダウン"}
```

JSON内の全テキストから全文検索

# 独自機能4

ノーマライザー

# ノーマライザー

- 文字を正規化するモジュール
  - 表記の違いを吸収できる
- アルファベット：全部小文字
- ひらがな・カタカナ：全部全角
- ㍉→ミリ
- UnicodeのNFKCベース

# 独自機能5

トークナイザー

# トークナイザー

- キーワード切り出しモジュール
  - クエリーに指定できる
    キーワードを調整
- 例：すもも|も|もも|も

# デフォルト：可変長Ngram

- 英語：字種区切り
  - 例：Hello|World|!!!
  - Bigramだとノイズが多い
- 日本語：Bigram
  - 例：ポスグレ→ポス|スグ|グレ|レ
  - 漏れがない

# 形態素解析器ベース

- MeCab：OSS
  - 新語対応には辞書メンテが必要
  - 参考：mecab-ipadic-neologd
- JMAT：商用製品
  - ジャストシステム社製
  - ATOKでも使っている辞書を提供
  - 新語にも強い
    参考：「JMAT Groonga Tokenizer Talks」で検索

# amindexでのポイント

- amoptions()
  - CREATE INDEXでのオプションを定義

```
CREATE INDEX index ON memos
  USING pgroonga (content)
   WITH (normalizer = 'NormalizerMySQLUnicodeCI',
         tokenizer  = 'TokenMecab');
```

# 速さ以外の利点2

## 見慣れた機能

B-tree・GINの代わりに使える

# タグ検索

```
CREATE TABLE memos (
  tags varchar(1023)[]
);
CREATE INDEX index ON memos
  USING pgroonga (tags);
SELECT * FROM memos
  WHERE tags %% 'タグ';
```

# 範囲検索

```
CREATE TABLE users (age int);
CREATE INDEX index ON users
  USING pgroonga (age);
SELECT * FROM users
  WHERE age < 20;
```

# Index Only Scan

- 索引がデータも返す
  - テーブルにアクセスしないので高速
- PGroonga・B-tree：サポート
  PostgreSQL 9.5からはGiSTもサポート
- GIN：未サポート

# amindexでのポイント

- amcanreturn()
  - trueを返す
- amgettuple()
  - scan->xs_want_itupなら
    scan->xs_itupにデータを設定

# LIKE

```
CREATE INDEX index ON memos
  USING pgroonga (content);
SELECT * FROM memos
  WHERE content LIKE '%...%';
```

索引を使って高速検索
アプリケーションの変更不要

# マルチカラムインデックス

```
CREATE INDEX index ON memos
  USING pgroonga (title, content);
SELECT * FROM memos
  WHERE title @@ '...' AND
        content @@ '...';
```

titleでもcontentでもマッチ！
と書けないのでそんなにうれしくない

設計ミス

# text @@ text

## 組み込みの定義と競合

ts_vector @@ ts_queryにキャストされる

## 回避方法：

```
ALTER DATABASE name
  SET search_path = '$user',public,pgroonga,pg_catalog;
```

# jsonb @@ text

## 全文検索にすればよかった

```
jsonb @@ 'string @ "キーワード"'
-- ↓
jsonb @@ 'キーワード'
```

今の細かい検索条件を指定できる機能は別演算子にする

# 今後

# 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

# 同義語展開サポート

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

Groongaでは使える

# ステミングサポート

found/finds

↓

find

Groongaでは使える

# text @@ pgroonga.query

```
body @@ 'ポスグレ'::pgroonga.query
```

組み込みのtext @@ textとの競合回避

# 重みサポート

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

Groongaでは使える

# 複合主キーサポート

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

# まとめ

時間が余ったらチュートリアルを自慢する

# まとめ

- PGroongaは速い
- PGroongaは便利
- PGroongaには設計ミスがある
- PGroongaはもっと便利になる

PGroongaを使おう！

# おしらせ

- Groonga Meatup 2015
  https://groonga.doorkeeper.jp/events/31482
  - PGroongaの話題もアリ
  - 多少空きアリ（定員を多少増やせる）
- 11月29日（日）13:30開始
- 来年2月9日は「MySQLとPostgreSQLと日本語全文検索」
  https://groonga.doorkeeper.jp/events/35295