PGroonga

Make PostgreSQL fast full text search platform for all languages!

Kouhei Sutou

ClearCode Inc.

PGConf.ASIA 2016
2016-12-03

PostgreSQL and me

PostgreSQLと私

Some my patches are merged

いくつかパッチがマージされている

Patches
パッチ

- #13840: pg_dump generates unloadable SQL
  pg_dumpがリストアできないSQLを出力する
- #14160: DROP ACCESS METHOD IF EXISTS isn't impl.
  DROP ACCESS METHOD IF EXISTSが実装されていない

They are found while developing PGroonga
どちらもPGroonga開発中に見つけた問題

PGroonga dev style
PGroongaの開発スタイル

- When there are problems in related projects including PostgreSQL
  PostgreSQLを含む関連プロジェクトに問題があった場合
  - We fix these problems in these projects instead of choosing workaround in PGroonga
    PGroonga側で回避するのではなく
    関連プロジェクトの方で問題を直す

PostgreSQL and FTS
PostgreSQLと全文検索

- PostgreSQL has built-in full text search feature
  PostgreSQLには組込の全文検索機能がある
  - It has some problems...
    ただ、いくつか問題がある
  - We fixed them by PGroonga
    PGroongaを開発することでそれらの問題を修正した
  - instead of fixing PostgreSQL 😥
    PostgreSQLを修正するのではなくて…

Because...
理由は…

1. Our approach is different from PostgreSQL's approach
   PGroongaのやり方はPostgreSQLのやり方と違う
2. PG provides plugin system
   PostgreSQLはプラグインの仕組みを提供している
   - Implementing as a plugin is PostgreSQL way!
     プラグインでの実装はPostgreSQLらしいやり方！

PG FTS problem
PostgreSQLの全文検索の問題

Many langs aren't supported
サポートしていない言語がたくさんある

- e.g.: Asian languages
  例：アジアの言語
  - Japanese, Chinese and more
    日本語や中国語など

FTS for Japanese1

日本語の全文検索1

```
SELECT
  to_tsvector('japanese',
              'こんにちは');
-- ERROR:  text search configuration
--         "japanese" does not exist
-- LINE 2:   to_tsvector('japanese',
--                       ^
```

FTS for Japanese2

日本語の全文検索2

```
CREATE EXTENSION pg_trgm;
SELECT show_trgm('こんにちは');
--   show_trgm
-- -----------
--   {} ← Must not empty!
-- (1 row)
```

Existing solution
既存の解決策

pg_bigm

pg_bigm

- An extension
  拡張機能
- Similar to pg_trgm
  pg_trgmと似ている
  - Operator class for GIN
    GIN用の演算子クラス

pg_bigm: Usage

pg_bigm：使い方

```
CREATE INDEX index ON table
  USING GIN (column gin_bigm_ops);
--      ↑Use GIN    ↑Specify op class
```

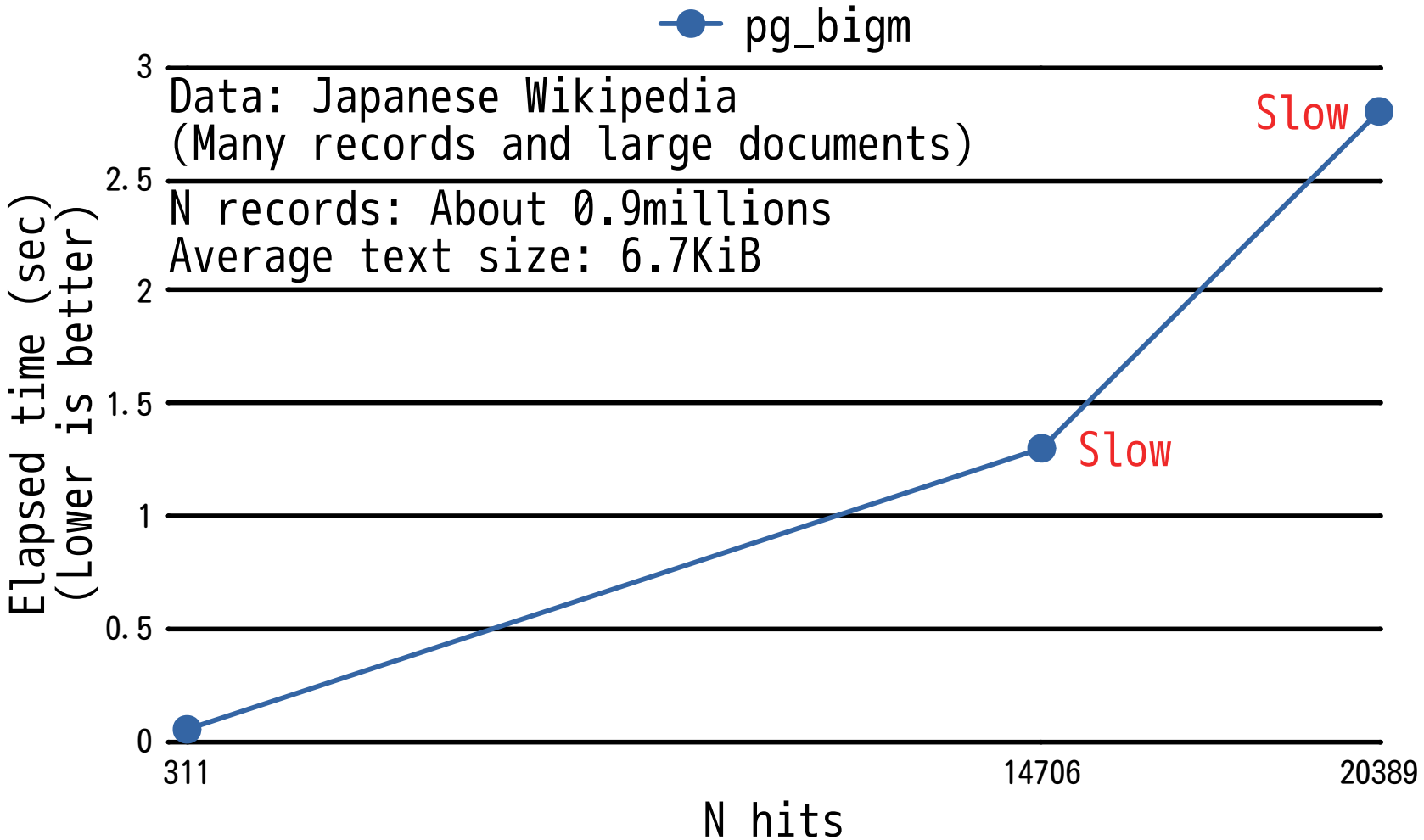
pg_bigm: Demerit
pg_bigm：デメリット

- Slow for large document
  文書が長いと遅い
  (Normally, we want to use FTS for large document)
  （普通は長い文書に対して全文検索したい）
  - Because it needs "recheck"
    「recheck」が必要だから

"recheck"

- "Exact" seq. search after "loose" index search
  「ゆるい」インデックス検索の後に実行する「正確な」シーケンシャルサーチ
  - The larger text, the slower
    対象テキストが大きければ大きいほど遅くなる
  - text = doc size * N docs
    対象テキスト = 文書サイズ * 文書数

Benchmark
ベンチマーク



New solution
新しい解決策

pgroonga

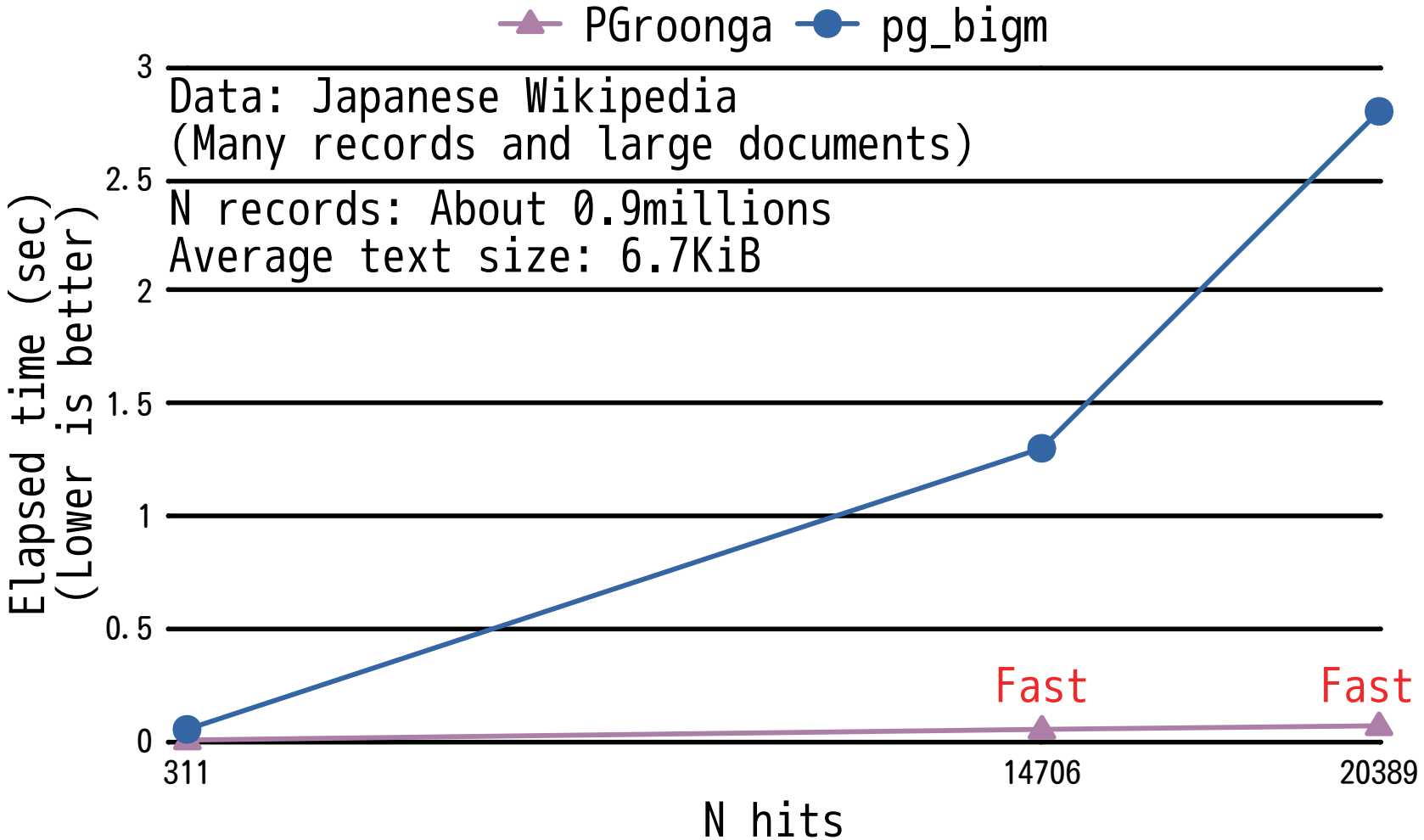
PGroonga

- Pronunciation: pí:zí:lúngά
  読み方：ぴーじーるんが
- An extension
  拡張機能
- Index and operator classes
  インデックスと演算子クラス
  - Not operator classes for GIN
    GINの演算子クラスではない

PGroonga layer

Operator class	textsearch pg_trgm pg_bigm	PGroonga
Index	GIN	PGroonga

Benchmark
ベンチマーク



Wrap up1
まとめ1

- PostgreSQL doesn't support Asian languages
  PostgreSQLはアジアの言語をサポートしていない
- pg_bigm and PGroonga support all languages
  pg_bigmとPGroongaはすべての言語をサポートしている

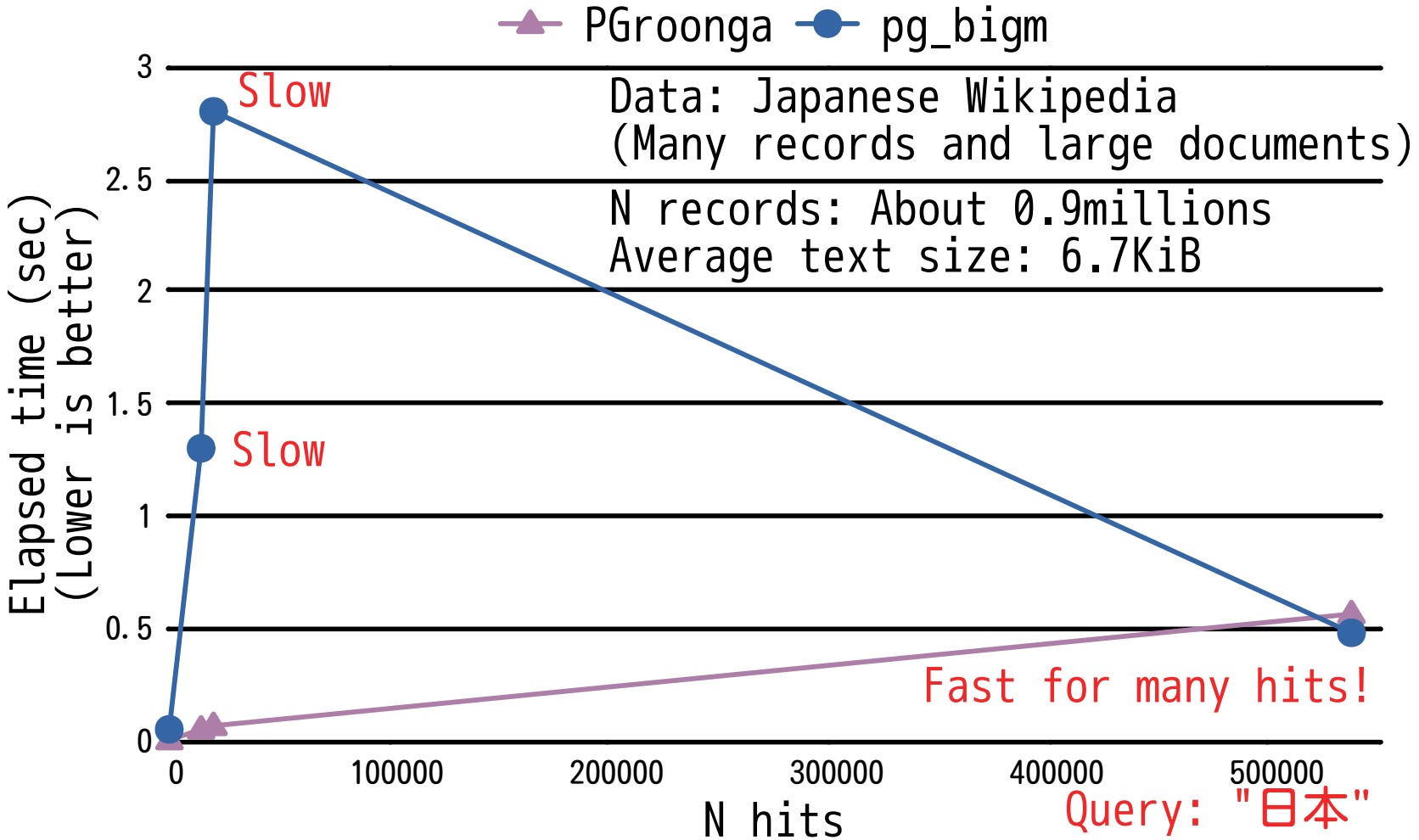
Wrap up2
まとめ2

- Many hits case:
  ヒット数が多い場合
  - pg_bigm is slow
    pg_bigmは遅い
  - PGroonga is fast
    PGroongaは速い

Why is PGroonga fast?
PGroongaはどうして速いのか

- Doesn't need "recheck"
  「recheck」が必要ないから
- Is "recheck" really slow?
  本当に「recheck」が遅いの？
  - See one more benchmark result
    もう一つベンチマーク結果を見てみましょう

Benchmark
ベンチマーク



Why is pg_bigm fast?
pg_bigmはどうして速いのか

- Query is "日本"
  クエリーは「日本」
  - Point: 2 characters
    ポイント：2文字
- pg_bigm doesn't need "recheck" for 2 chars query
  pg_bigmは2文字のクエリーに「recheck」の必要がない
  - It means that "recheck" is slow
    つまり「recheck」が遅いということ

N-gram and "recheck"

N-gramと「recheck」

- N-gram approach needs "phrase search" when query has N+1 or more characters
  N+1文字以上のクエリーには「フレーズ検索」が必要
  - N=2 for pg_bigm, N=3 for pg_trgm
    pg_bigmはN=2でpg_trgmはN=3
- GIN needs "recheck" for "phrase search"
  GINは「フレーズ検索」には「recheck」が必要

Phrase search
フレーズ検索

- Phrase search is "token search" and "position check"
  フレーズ検索は「トークン検索」と「位置チェック」
  - Tokens must exist and be ordered
    トークンは同じ順序で出現していないといけない
  - OK: "car at" for "car at" query
  - NG: "at car" for "car at" query

N-gram and phrase search

1. Split text to tokens
   テキストをトークンに分割
   - "cat"→"ca","at"
2. **Search all tokens**
   すべてのトークンを検索
   - "ca" & "at" exist: Candidate!
3. **Check appearance pos.**
   出現位置をチェック
   - "ca" then "at": Found!

N-gram and GIN: Create

N-gramとGIN：作成

Documents

ID	Text
10	cat
20	at car

Tokenize

"ca","at"

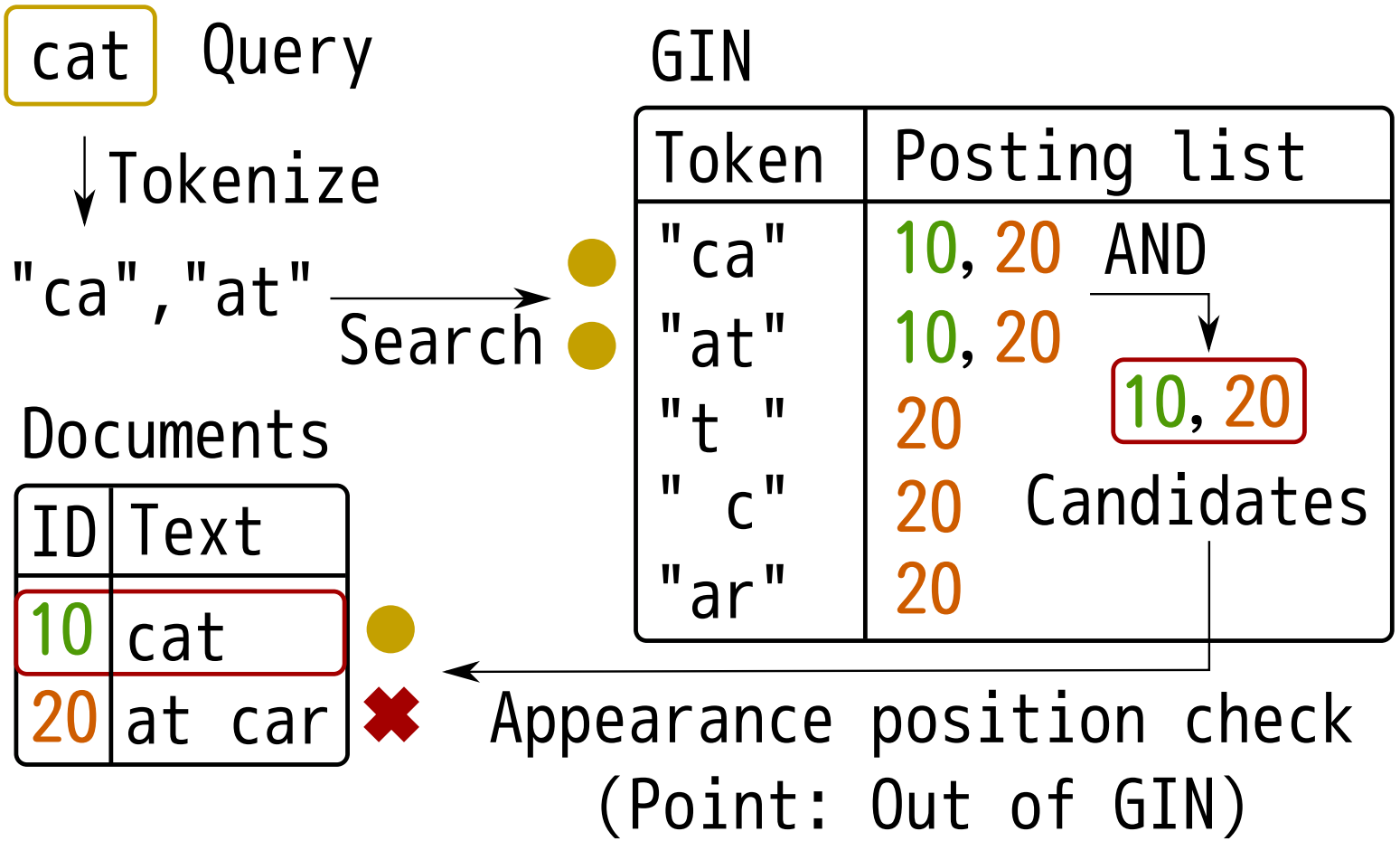
"at","t "," c","ca","ar"

GIN

Token	Posting list
"ca"	10, 20
"at"	10, 20
"t "	20
" c"	20
"ar"	20

N-gram and GIN: Search

N-gramとGIN：検索



GIN and phrase search
GINとフレーズ検索

- Phrase search needs position check
  フレーズ検索では出現位置チェックが必要
- GIN doesn't support position check
  GINは出現位置チェックをサポートしていない
  - →GIN needs "recheck"→Slow!
    GINでは「recheck」が必要だから遅い

Why is PGroonga fast?

PGroongaはどうして速いのか

- PGroonga uses N-gram by default
  PGroongaはデフォルトでN-gramを使っている
- But doesn't need "recheck"
  PGroongaは「recheck」の必要がない

Why no "recheck"?

どうして「recheck」が必要ないのか

PGroonga uses
full
inverted index

PGroongaは**完全**転置インデックスを使っているから

Full inverted index
完全転置インデックス

Including position
位置情報を含む

Inverted index diff

転置インデックスの違い

Documents

ID	Text
10	cat
20	at car

Tokenize →

"ca","at"
1 2

"at","t "," c","ca","ar"
1 2 3 4 5

Not full: Only doc ID

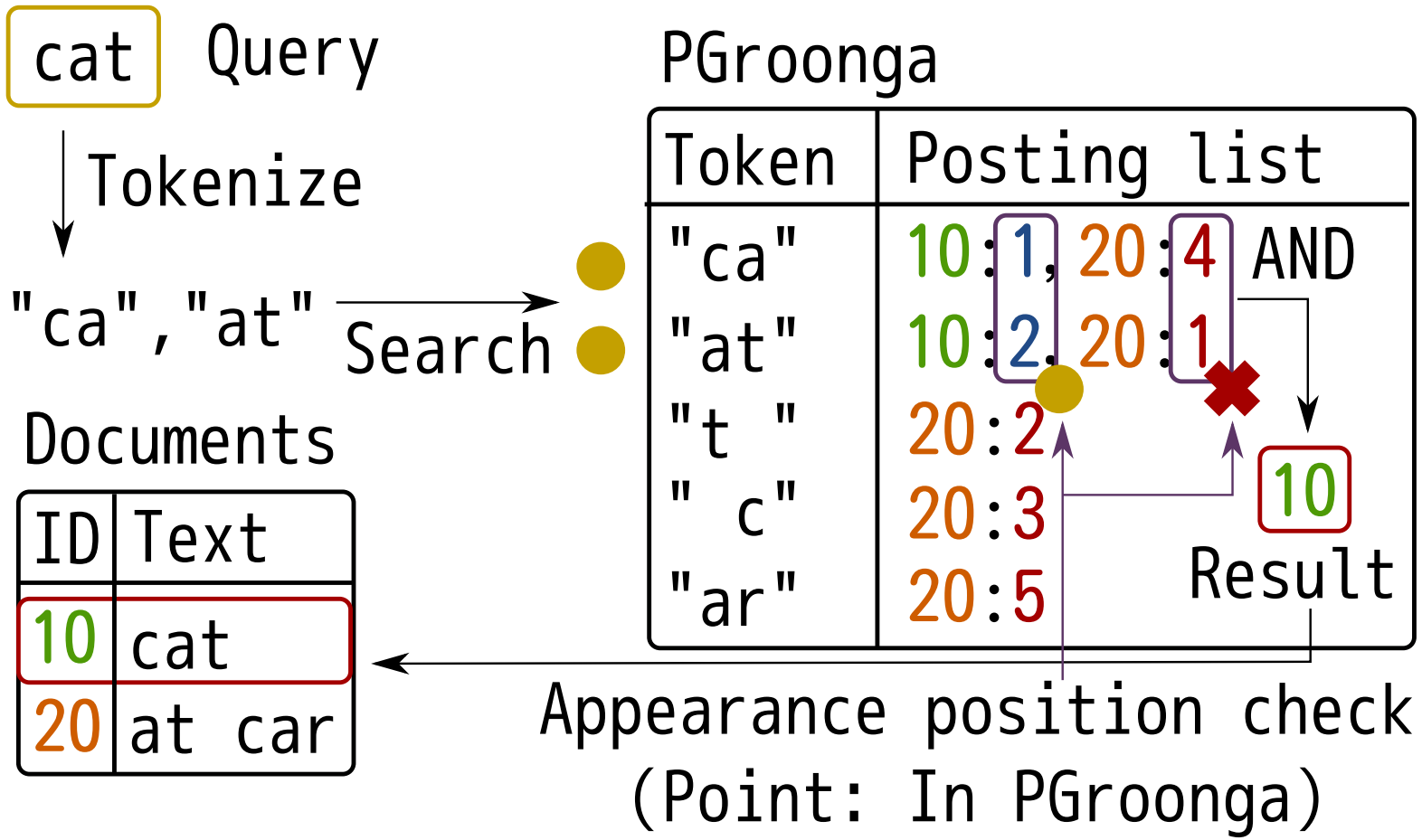
Token	Posting list
"ca"	10, 20
"at"	10, 20
"t "	20
" c"	20
"ar"	20

Full: Doc ID + pos

Token	Posting list
"ca"	10:1, 20:4
"at"	10:2, 20:1
"t "	20:2
" c"	20:3
"ar"	20:5

N-gram/PGroonga: Search

N-gramとPGroonga：検索



Wrap up
まとめ

- N-gram needs phrase search
  N-gramの場合はフレーズ検索が必要
- Full inverted index provides fast phrase search
  完全転置インデックスを使うと高速にフレーズ検索できる
  - GIN isn't full inverted index
    GINは完全転置インデックスではない
  - PGroonga uses full inverted index
    PGroongaは完全転置インデックスを使っている

FTS and English(*)
全文検索と英語

- Normally, N-gram isn't used for English FTS
  普通は英語の全文検索にN-gramを使わない
  - N-gram is slower than word based approach (textsearch approach)
    N-gramは単語ベースのやり方（textsearchのやり方）より遅め
  - Stemming/stop word can't be used
    N-gramではステミングとストップワードを使えない

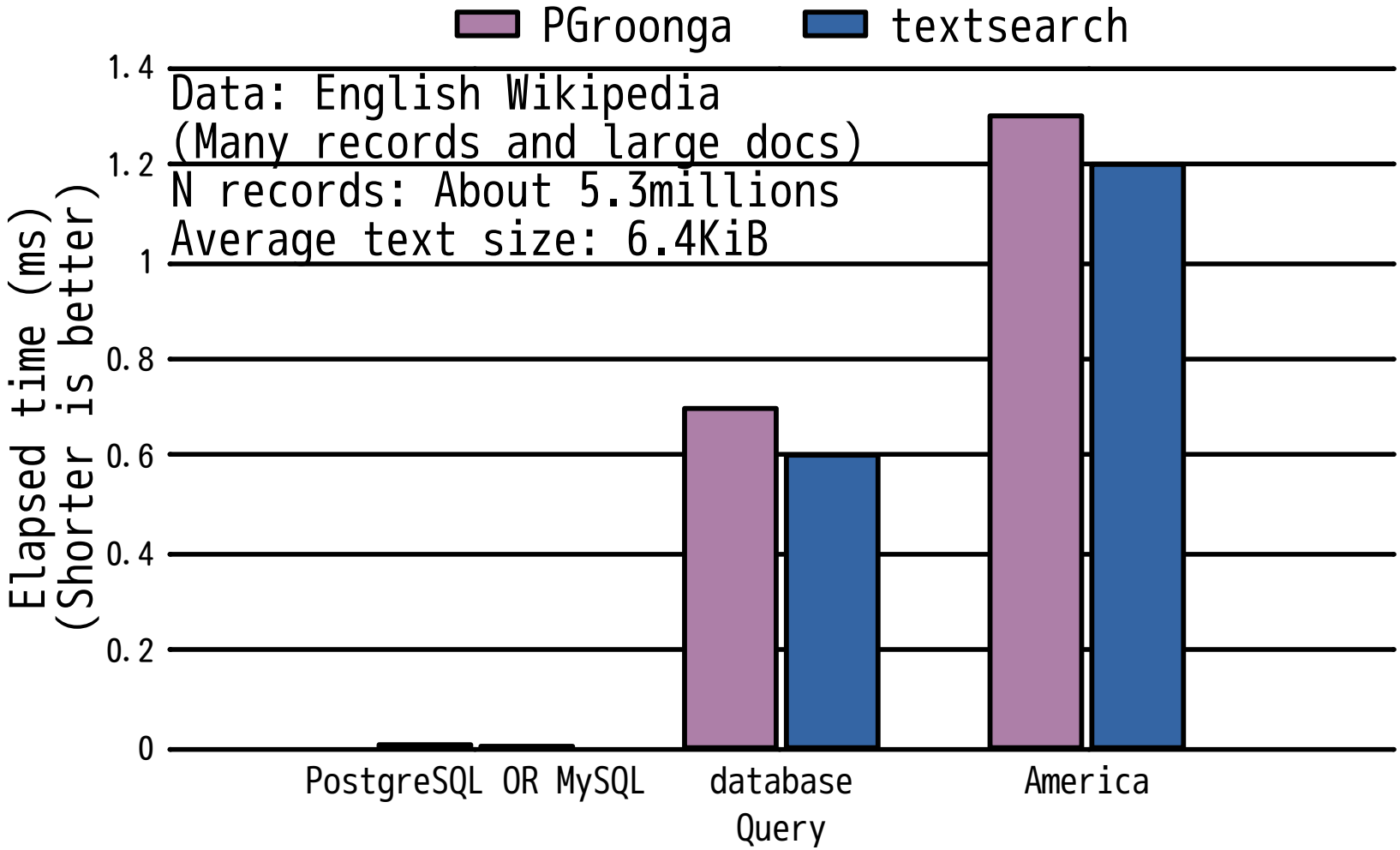
(*) English≒Alphabet based languages

PGroonga and English
PGroongaと英語

- PGroonga uses N-gram by default
  PGroongaはデフォルトではN-gramを使う
  - Is PGroonga slow for English?
    ではPGroongaは英語では遅いのか？
  - No. Similar to textsearch
    遅くない。textsearchと同じくらい

PGroonga: Search

PGroonga：検索



PGroonga's N-gram

- Variable size N-gram
  可変長サイズのN-gram
  - Continuous alphabets are 1 token (= word based approach)
    連続したアルファベットは1トークン（=単語ベース）
    Hello→"Hello" not "He","el",…
  - No alphabet is 2-gram
    非アルファベットは2-gram
    こんにちは→"こん","んに",…

Wrap up1
まとめ1

- PGroonga's search is fast for all languages
  PGroongaの検索はすべての言語で速い
  - Including alphabet based languages and Asian languages mixed case
    アルファベットベースの言語とアジアの言語が混ざった場合でも速い
    (textsearch doesn't support mixed case)
    (textsearchは言語を混ぜることはできない)

Wrap up2
まとめ2

PGroonga makes PostgreSQL
fast full text search platform
for all languages!

PGroongaでPostgreSQLが
全言語対応高速全文検索プラットフォームになる！

More about PGroonga

PGroongaについてもっと

- Performance
  性能
- Japanese specific feature
  日本語向けの機能
- JSON support
  JSONサポート
- Replication
  レプリケーション

Performance
性能

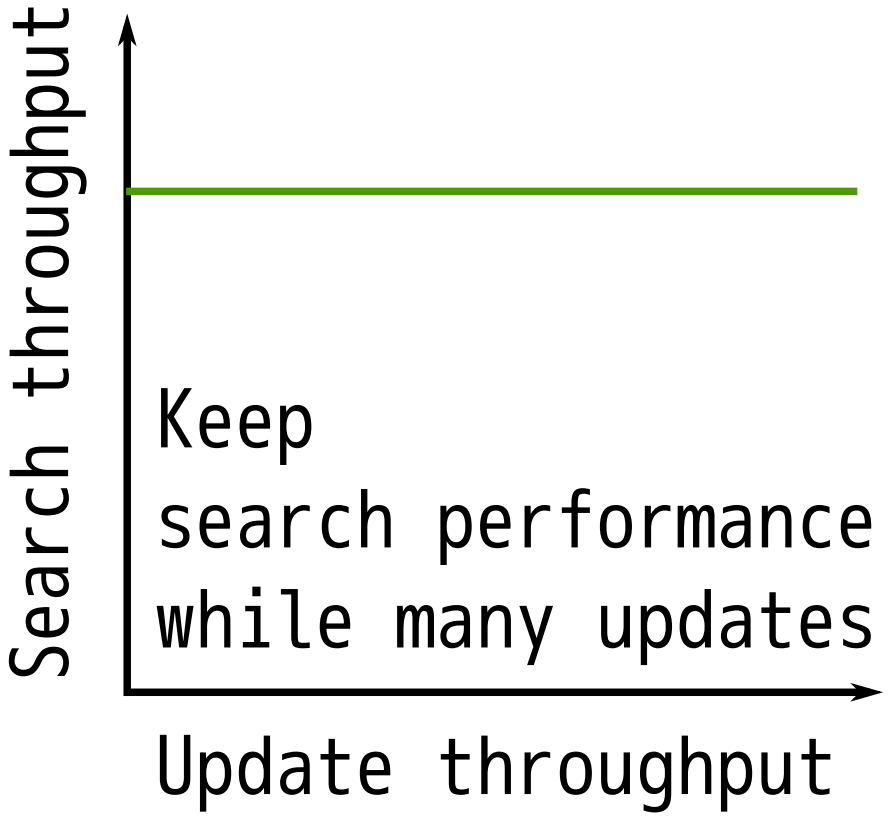
- Search and update
  検索と更新
- Index only scan
  インデックスオンリースキャン
- Direct Groonga search
  直接Groongaで検索
- Index creation
  インデックス作成

Search and update
検索と更新

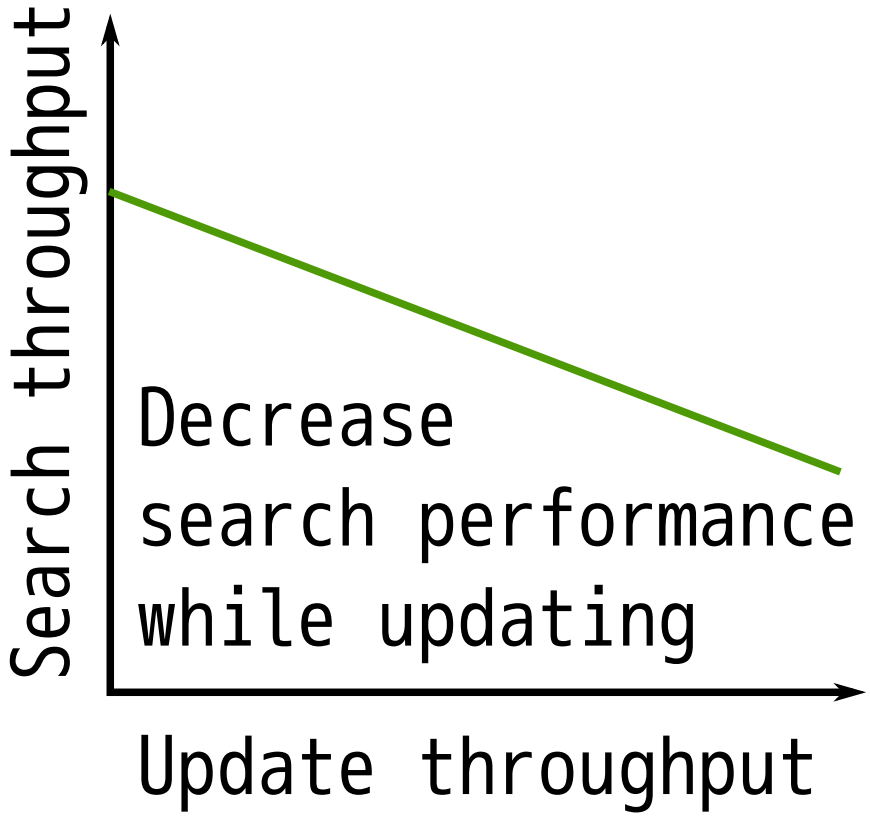
- Doesn't decrease search performance while updating
  更新中も検索性能が落ちない
  - It's good characteristics for chat application
    チャットアプリでうれしい傾向
  - Zulip supports PGroonga
    Zulip: OSS chat app by Dropbox
    ZulipはPGroongaをサポートしている
    ZulipはDropboxが開発しているOSSのチャットアプリ

Characteristics
傾向

PGroonga



GIN

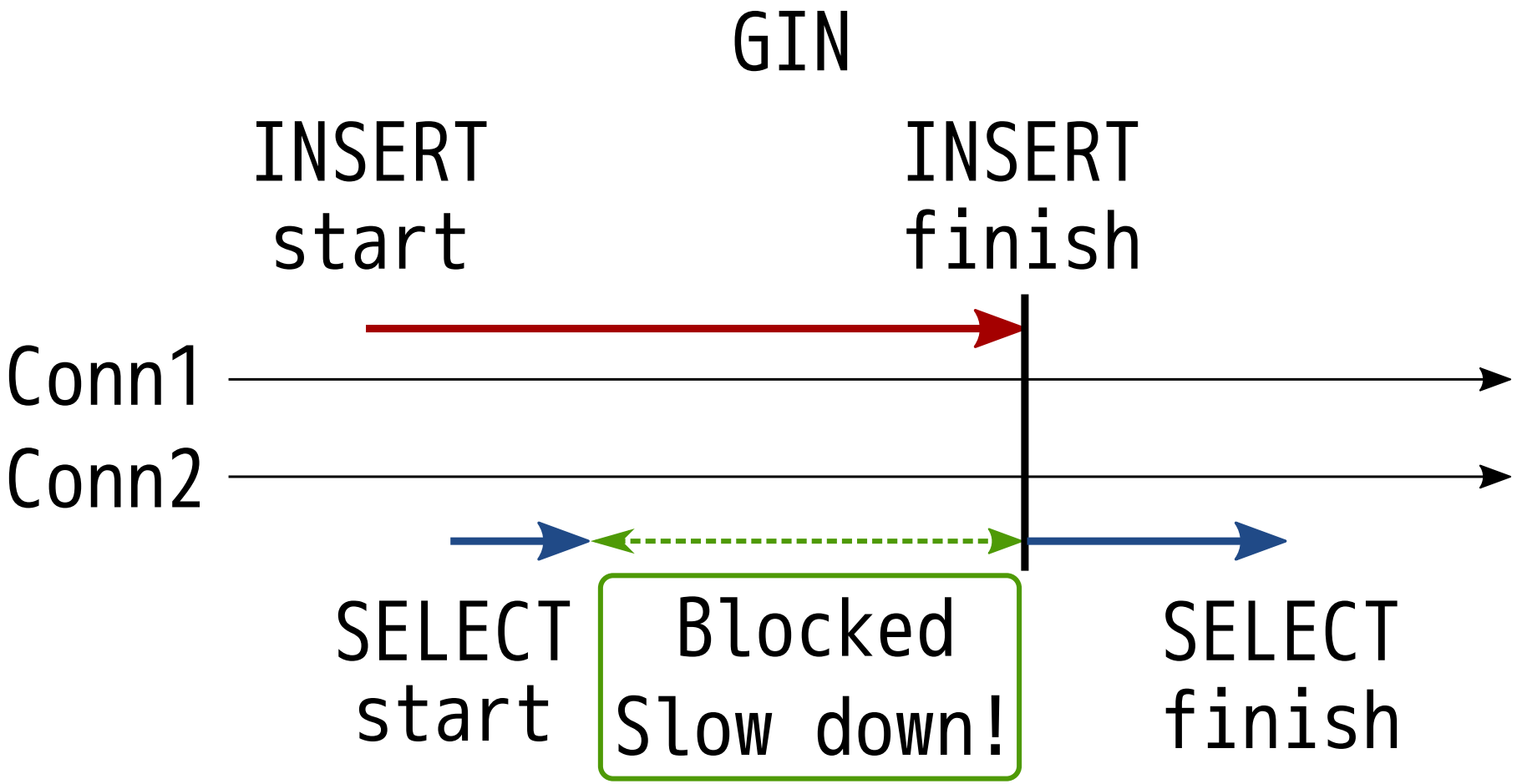


Update and lock
更新とロック

- Update without **read** locks
  参照ロックなしで更新
  - **Write** locks are required
    書き込みロックは必要

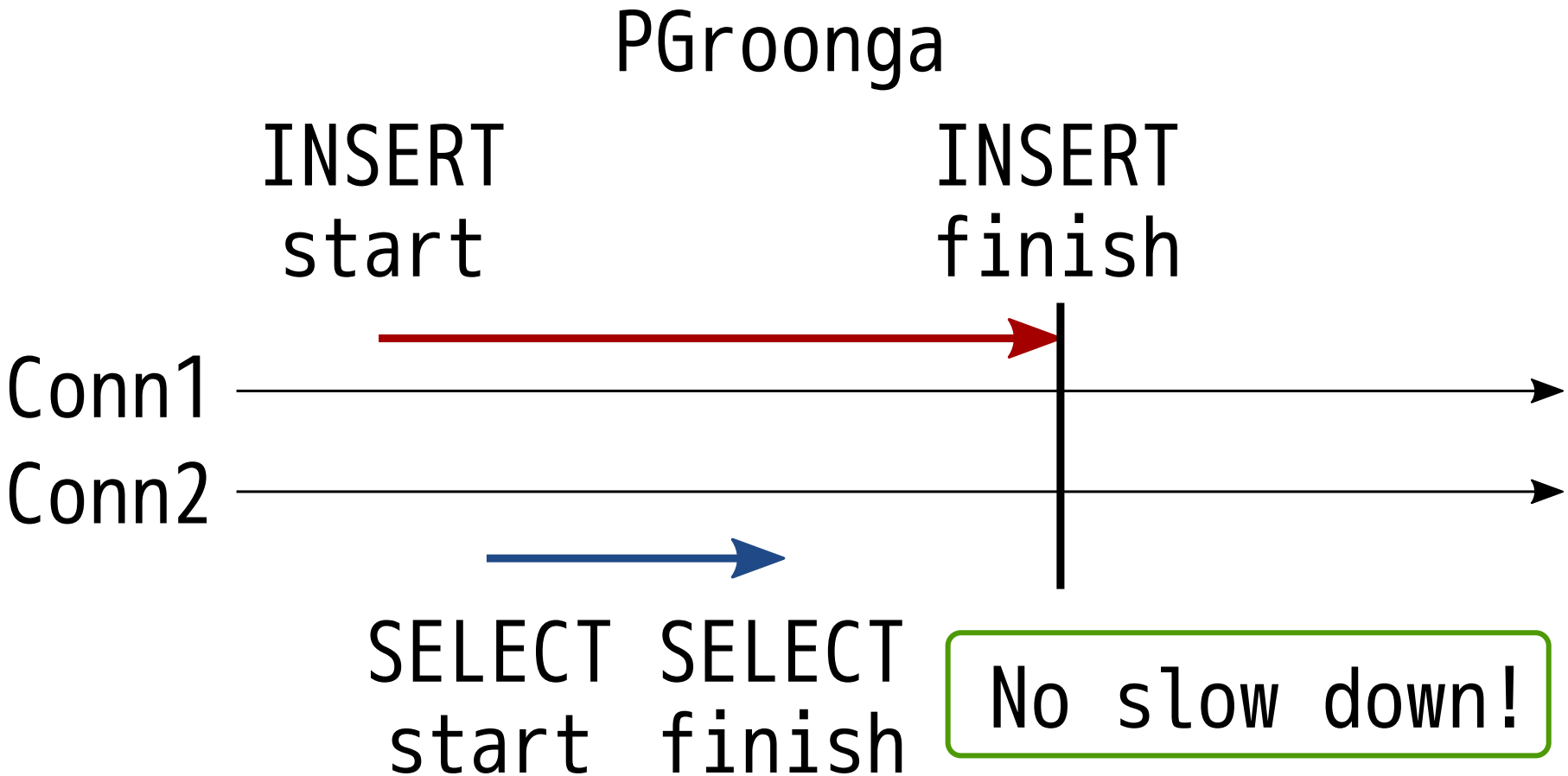
GIN: Read/Write

GIN：読み書き



PGroonga: Read/Write

PGroonga：読み書き



Fast stably
安定して速い

- GIN has intermittent performance decrements
  GINは間欠的な性能劣化がある
  - For details:🔎"GIN pending list"
    詳細は「GIN pending list」で検索
- PGroonga keeps fast search
  PGroongaは高速な検索を維持
  - PGroonga keeps index updated
    PGroongaのインデックスは常に最新状態

Index only scan
インデックスオンリースキャン

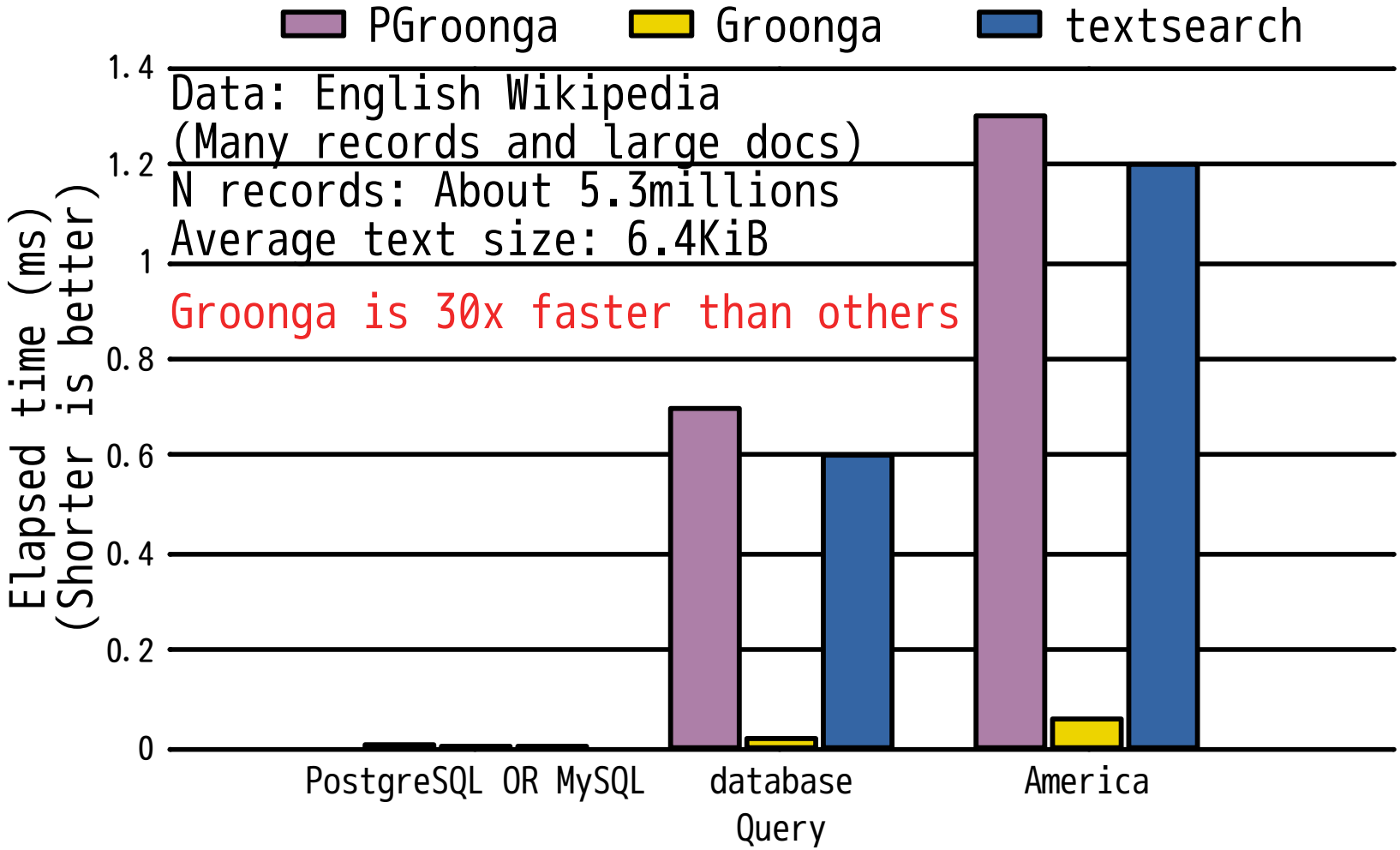
- GIN: Not supported
  GIN：未サポート
- PGroonga: Supported
  PGroonga：サポート

More faster search
より高速な検索

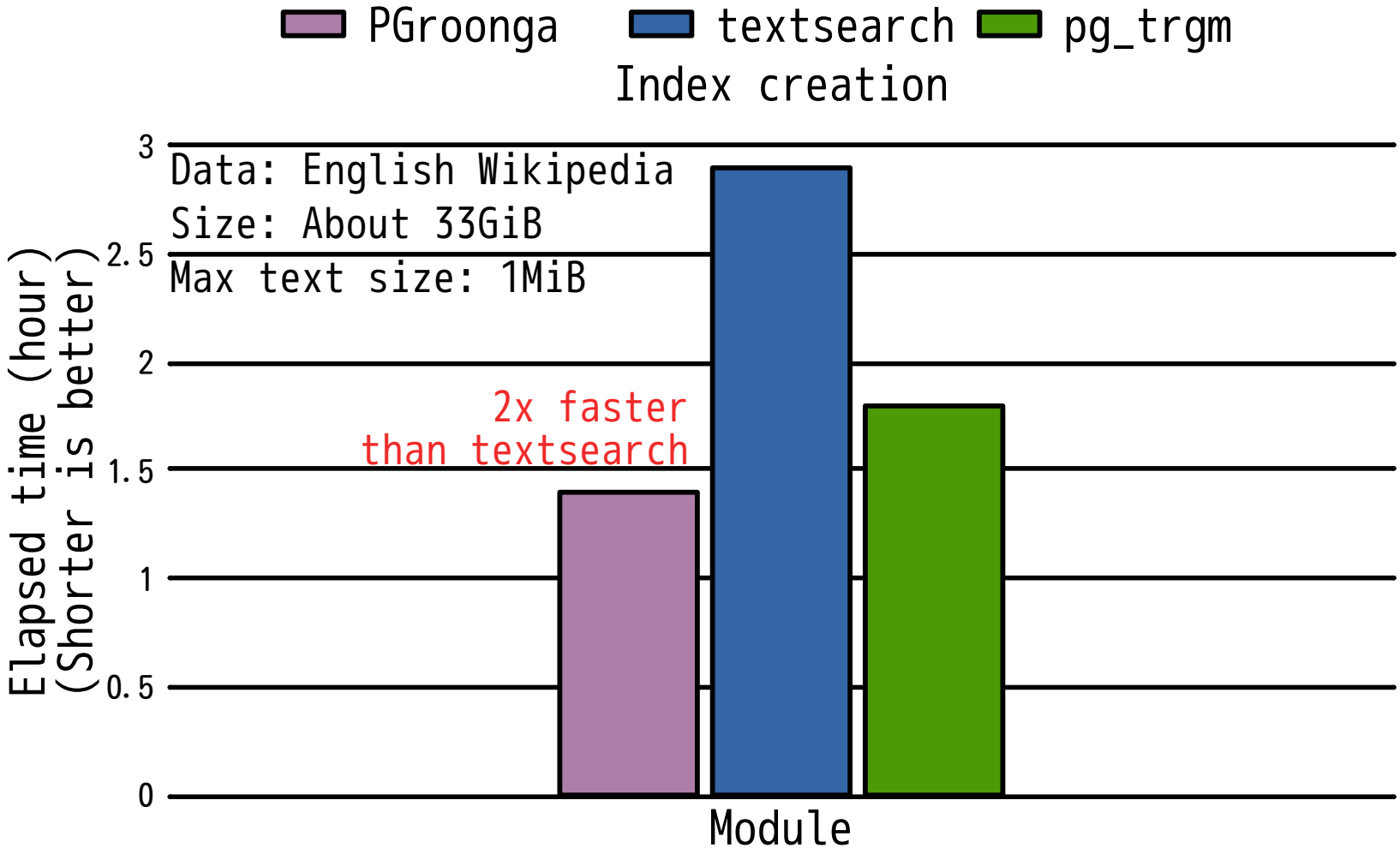
- Direct Groonga search is more faster
  直接Groongaで検索するとさらに高速
  - Groonga: Full text search engine PGroonga uses
    Groonga：PGroongaが使っている全文検索エンジン

Direct Groonga search

直接Groongaで検索



Index creation time
インデックス作成時間



Performance: Wrap up
性能：まとめ

- Keep fast search w/ update
  更新しながらでも高速検索を維持
- Support index only scan
  インデックスオンリースキャンをサポート
- Direct Groonga search is more faster
  Groonga直接検索はもっと速い
- Fast index creation
  インデックス作成も速い

Japanese specific feature
日本語向けの機能

- Completion by Romaji
  ローマ字による入力補完

Completion: Table
入力補完：テーブル

```
CREATE TABLE stations (
  name text,
  readings text[]
  -- ↑Support N readings
);
```

Completion: Data
入力補完：データ

```
INSERT INTO stations VALUES
('Tokyo',
 ARRAY['トウキョウ']),
-- ↑In Katakana
-- (...),
('Akihabara',
 ARRAY['アキハバラ', 'アキバ']);
```

Completion: Index
入力補完：インデックス

```
CREATE INDEX pgroonga_index
  ON stations
  USING pgroonga (
    -- ↓For prefix and prefix Romaji/Katakana search
    name     pgroonga.text_term_search_ops_v2,
    -- ↓For prefix and prefix Romaji/Katakana search
    --   against array
    readings pgroonga.text_array_term_search_ops_v2);
```

Completion: Search
入力補完：検索

```
SELECT name, readings
  FROM stations
 WHERE name &^ 'tou' OR
       -- ↑Prefix search
       readings &^~> 'tou'
       -- ↑Prefix Romaji/Katakana search
 ORDER BY name LIMIT 10;
```

Completion: Result
入力補完：結果

Hit by
prefix Romaji/Katakana search
"tou"(Romaji)→"トウ"(Katakana)

前方一致RK検索でヒット

```
 name   |    readings
--------+----------------
 Tokyo  | {トウキョウ}
(1 row)
```

For Japanese: Wrap up
日本語向け機能：まとめ

- Support prefix Romaji/Kana search
  前方一致RK検索をサポート
  - Useful for implementing auto complete feature in search box
    検索欄にオートコンプリート機能を実装する時に便利
  - Users don't need to convert Romaji to Kanji
    ユーザーはローマ字を漢字に変換する必要がない

JSON support
JSONサポート

- Support full text search
  全文検索対応
  - Target: All texts in JSON
    JSON内のすべてのテキスト
  - Not only a text in a path
    特定のパスのテキストだけではない
    (GIN supports only this style)
    (GINはこのやり方だけサポート)

JSON: FTS: Data

JSON：全文検索：Data

```
CREATE TABLE logs (
  record jsonb
);
INSERT INTO logs (record)
  VALUES ('{"host": "app1"}'),
         ('{"message": "app is down"}');
```

JSON: FTS: Index

JSON：全文検索：インデックス

```
CREATE INDEX message_index ON logs
  USING GIN
    ((record->>'message') gin_trgm_ops);
-- {"message": "HERE IS ONLY SEARCHABLE"}
CREATE INDEX record_index ON logs
  USING pgroonga (record);
-- All string values are searchable
```

JSON: FTS: GIN

JSON：全文検索：GIN

```
SELECT * FROM logs
  WHERE record->>'message' LIKE '%app%';
-- ↑ {"host": "app1"} isn't target
--              record
-- ----------------------------
--  {"message": "app is down"}
-- (1 row)
```

JSON: FTS: PGroonga

JSON：全文検索：PGroonga

```
SELECT * FROM logs
  WHERE record @@ 'string @ "app"';
-- ↑ All string values are target
--                record
-- ---------------------------
--  {"host": "app1"}
--  {"message": "app is down"}
-- (2 rows)
```

JSON: Wrap up
JSON：まとめ

- Support full text search against all texts in JSON
  JSON内の全テキスト対象の全文検索をサポート

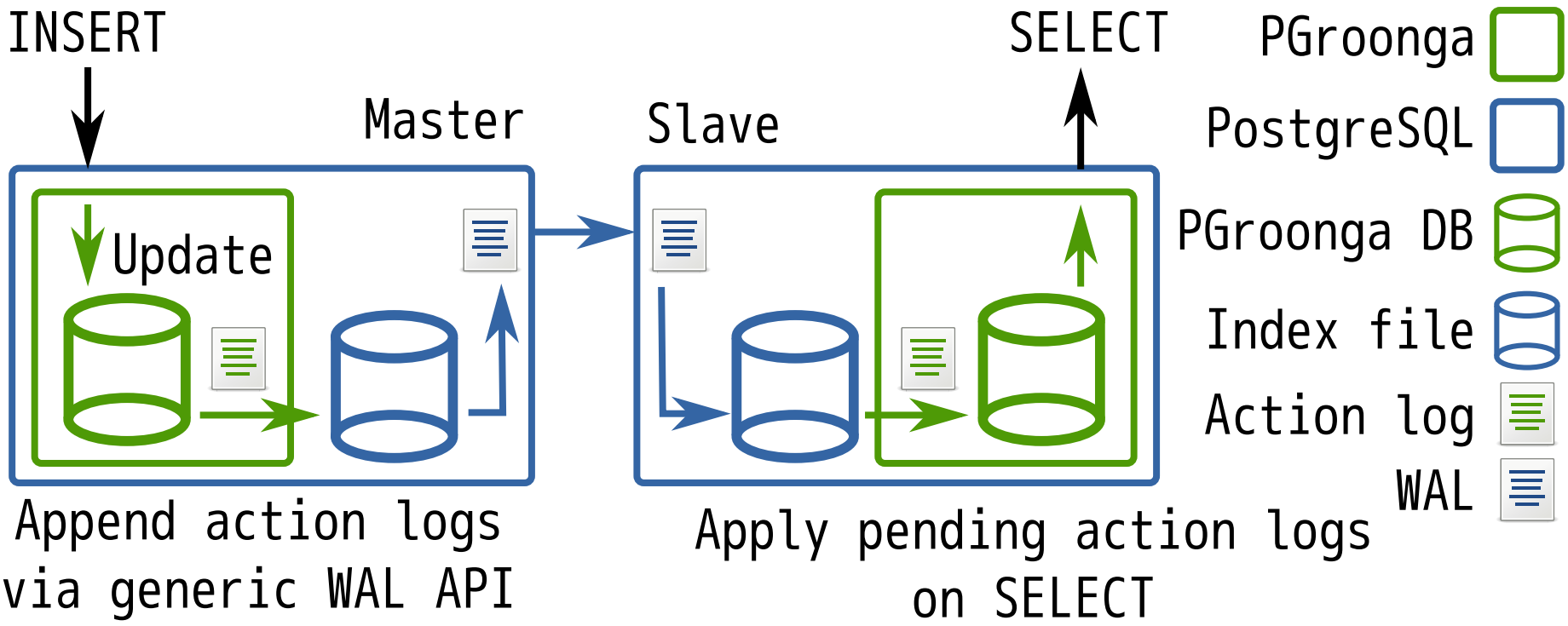
Replication
レプリケーション

- Support with PG 9.6!
  PostgreSQL 9.6で使う場合はサポート！
- PostgreSQL 9.6 ships "generic WAL"
  PostgreSQL 9.6で「generic WAL」機能が追加
  - Third party index can support WAL generation
    サードパーティーのインデックスもWALを生成できる

Implementation
実装

1. Master: Encode action logs as MessagePack
   マスター：アクションログをMessagePack形式に変換
2. Master: Write the action logs to WAL
   マスター：アクションログをWALに書き込み
3. Slaves: Read the action logs and apply them
   スレーブ：アクションログを読み込んで適用

Overview
概要


INSERT
Master
Update
Append action logs
via generic WAL API
SELECT
Slave
Apply pending action logs
on SELECT
PGroonga
PostgreSQL
PGroonga DB
Index file
Action log
WAL

Action log: "action"

アクションログ：「アクション」

```
{
  "_action": ACTION_ID
}
# ACTION_ID: 0: INSERT
# ACTION_ID: 1: CREATE_TABLE
# ACTION_ID: 2: CREATE_COLUMN
# ACTION_ID: 3: SET_SOURCES
```

Action log: INSERT
アクションログ：INSERT

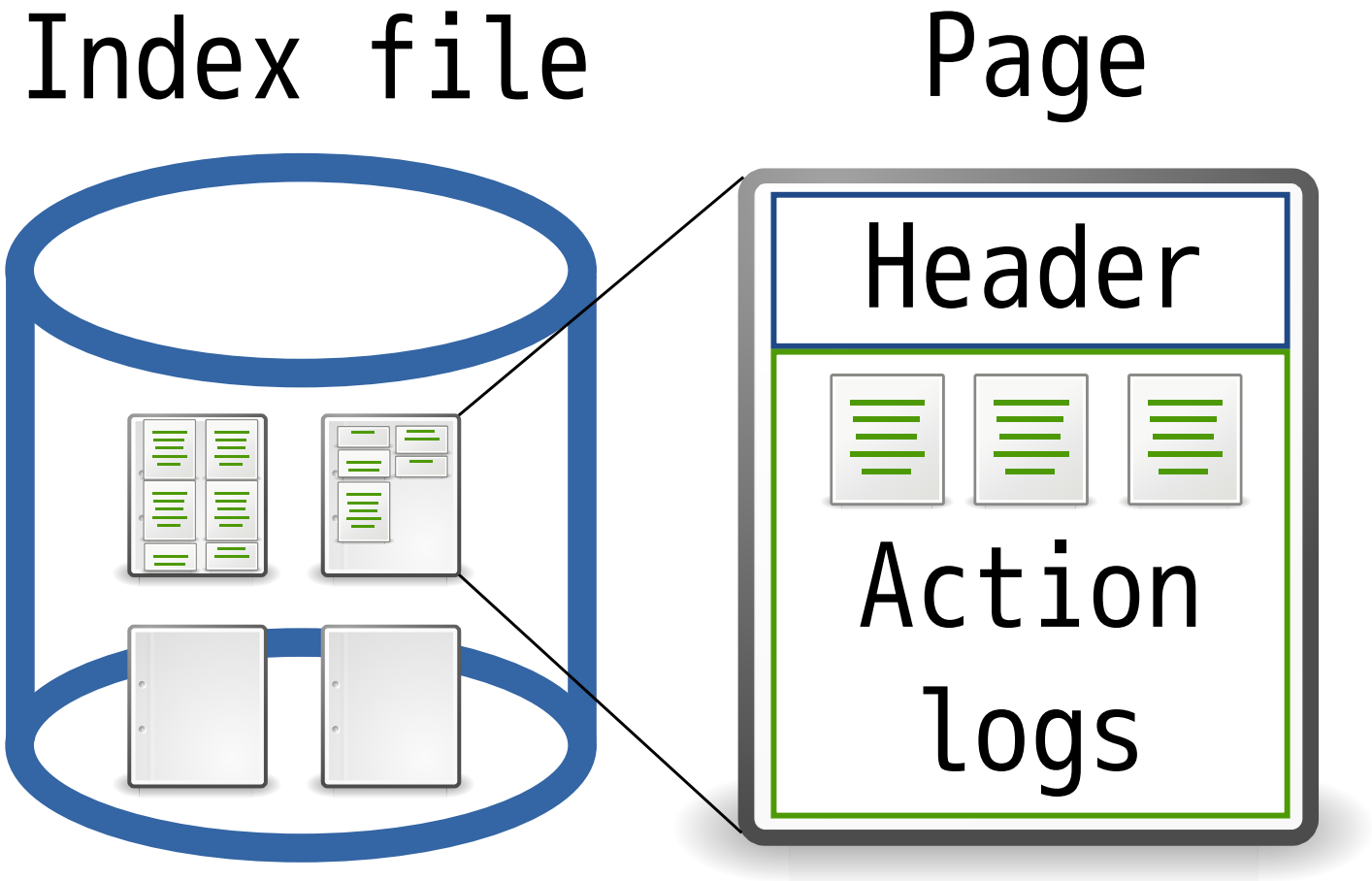
```
{
  "_action": 0,
  "_table":  "TABLE_NAME",
  "ctid":    PACKED_CTID_VALUE,
  "column1": COLUMN1_VALUE,
  ...
}
```

Action log: Logs

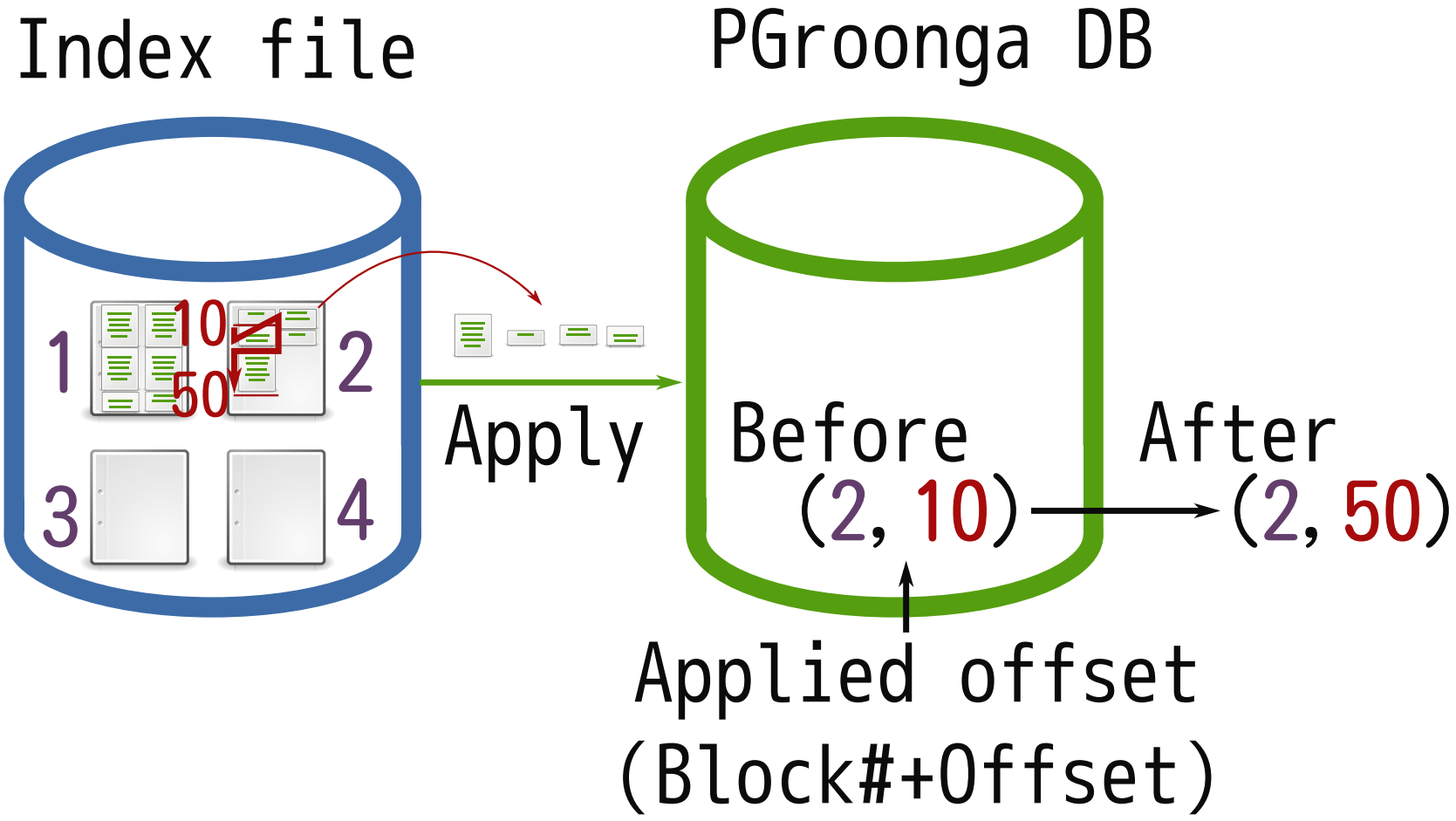
アクションログ：複数ログ

```
{"_action": ACTION_ID, ...}
{"_action": ACTION_ID, ...}
{"_action": ACTION_ID, ...}
...
```

Write action logs
アクションログの書き込み



Apply action logs
アクションログの適用



Action log: Why msgpack?
アクションログ：どうしてmsgpack？

- Because MessagePack supports streaming unpack
  MessagePackはストリーミングで展開できるから
  - It's useful to stop applying action logs when WAL is applied partially on slaves
    スレーブでWALが途中までしか書き込まれていないときにアクションログの適用を中断できるので便利

Replication: Wrap up
レプリケーション：まとめ

- Support with PG 9.6!
  PostgreSQL 9.6で使う場合はサポート！
- Concept: Action logs on WAL
  コンセプト：WAL上にアクションログを書く
  - It'll be an useful pattern for out of PostgreSQL storage index
    PostgreSQL管理外のストレージを使うインデックスではこのパターンが使えるはず

Wrap up1
まとめ1

- PostgreSQL doesn't support FTS for all languages
  PostgreSQLの全文検索は一部の言語のみ対応
- PGroonga supports FTS for all languages
  PGroongaの全文検索は全言語対応

Wrap up2
まとめ2

- PGroonga is fast stably
  PGroongaは安定して速い
- PGroonga supports FTS for all texts in JSON
  PGroongaはJSON中の全テキストに対する全文検索に対応

Wrap up3
まとめ3

- PGroonga supports replication
  PGroongaはレプリケーション対応
  - PostgreSQL 9.6 is required
    ただしPostgreSQL 9.6が必要

Wrap up4
まとめ4

PGroonga makes PostgreSQL
fast full text search platform
for all languages!

PGroongaでPostgreSQLが
全言語対応高速全文検索プラットフォームになる！

See also

https://pgroonga.github.io/

- Tutorial: /tutorial/
- Install: /install/
- Reference: /reference/
  - Includes replication doc and benchmark docs
- Community: /community/