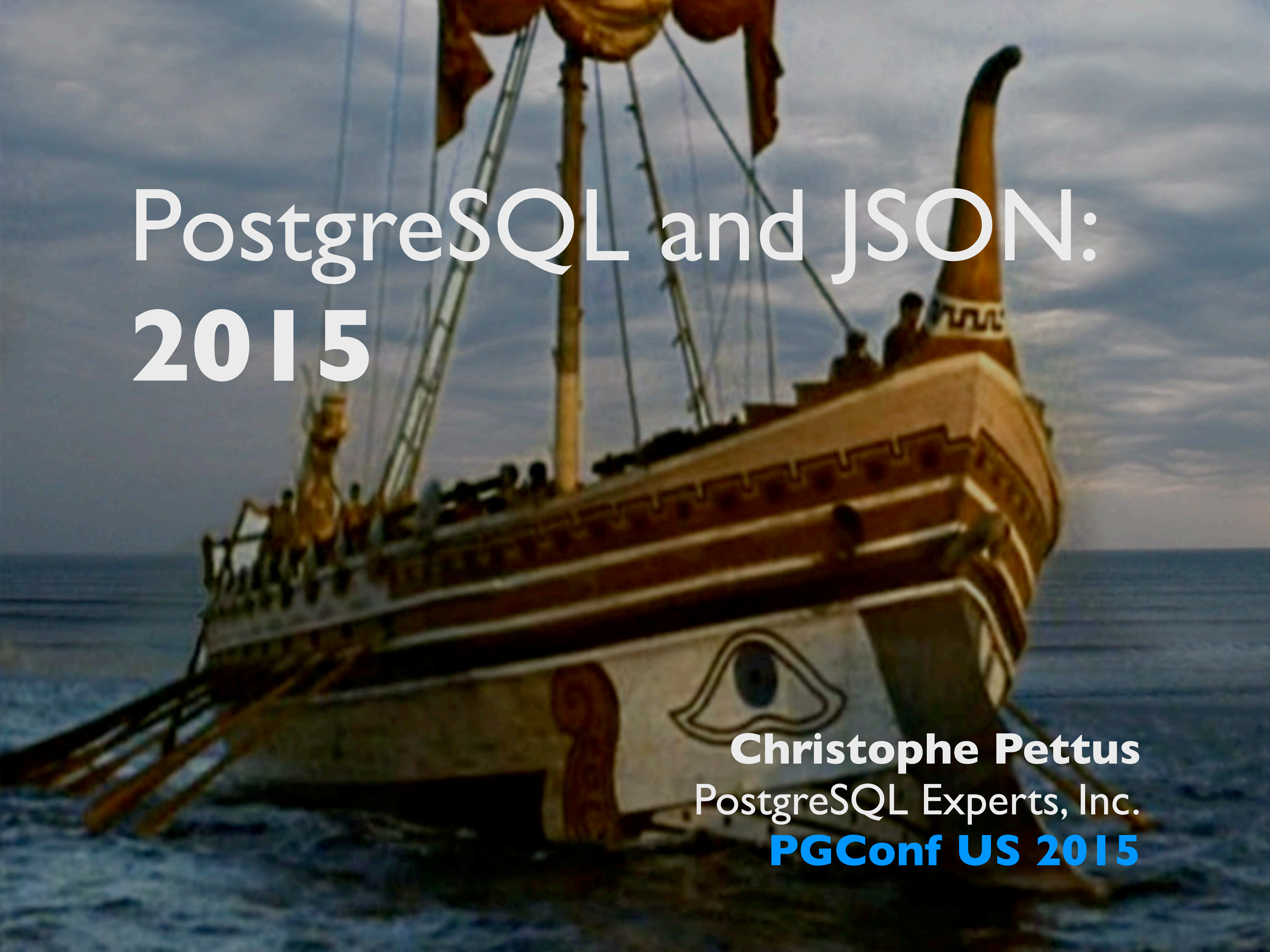


PostgreSQL and JSON: **2015**

Christophe Pettus
PostgreSQL Experts, Inc.

PGConf US 2015

Greetings!

- Christophe Pettus
- Consultant with PostgreSQL Experts, Inc.
- thebuild.com — personal blog.
- pgexperts.com — company website.
- Twitter @Xof
- christophe.pettus@pgexperts.com

PGX POSTGRESQL EXPERTS, INC.

JSON, what is?

- JavaScript Object Notation.
- A text format for serializing nested data structures.
- Based on JavaScript's declaration syntax.
- Intended to be passed directly into JavaScript's eval() function (don't do this!)

PGX POSTGRESQL EXPERTS, INC.

JSON Primitive Types.

- Strings, always Unicode.
  - De facto, always UTF-8 in flight.
- Numbers, integer and float.
- Boolean: `true` and `false`.
- `null`

PGX POSTGRESQL EXPERTS, INC.

JSON Structured Types.

- Arrays, using [ ].
- Hash / dictionaries / whatever you want to call them (the JSON spec calls them Objects), using { }
  - { "string" : value }
  - Keys have to be strings; values can be anything.

More complex types.

- Everything else is built out of those.
- There’s no type declaration mechanism.
  - “Object” is unfortunate terminology.
- There’s no “schema” or similar validation method.
- Everything is delegated to the application.

The good…

- It's super-simple to generate and parse.
  - The operational part of the spec is five pages, with diagrams.
- It's the de facto standard for data interchange in web APIs.
  - POST format is still used, but apps that do that are wrong.

The bad…

- No higher-level standards.
  - How is a datetime represented? I dunno, you figure it out.
- Remember SQL injection attacks? Now we have JSON injection attacks.
  - Don't use eval(). Just. Don't.

PGX POSTGRESQL EXPERTS, INC.

And PostgreSQL has JSON!

- It's a core type.
  - Not a contrib/ or extension module.
- Introduced in 9.2.
- Enhanced in 9.3.
- And really enhanced in 9.4.

We liked JSON so much…

- … we created two types.
  - json
  - jsonb
- json is a pure text representation.
- jsonb is a parsed binary representation.
- Each can be cast to the other, of course.

json type.

- Stores the actual json text.
- Whitespace included.
- What you get out is what you put in.
- Checked for correctness, but not otherwise processed.

Why use json?

- You are storing the json and never processing it.
- You need to support two JSON "features":
  - Order-preserved fields in objects.
  - Duplicate keys in objects.
- For some reason, you need the *exact* JSON text back out.

PGX
POSTGRESQL
EXPERTS, INC.

Oh, and…

- jsonb wasn't introduced until 9.4.
- So, if you are on 9.2-9.3, json is what you've got.
- Otherwise, you want to use jsonb.

jsonb

- Parsed and encoded on the way in.
- Stored in a compact, parsed format.
- Considerably more operator and function support.
- Has indexing support.

They're just types.

- Fully transactional, can have multiple `json`/`jsonb` fields in a single table, etc.
- Uses the TOAST mechanism.
    - Can be up to 1GB.
- Can be a NULLable field if you like.

PGX POSTGRESQL EXPERTS, INC.

Basic Operators (both json and jsonb)

- -> gets a JSON array element or object field, as JSON.
- ->> gets the array element or object field cast to TEXT.
- #> gets the array element or object field at a path.
- #>> … cast to TEXT.

jsonb only!

- @> — Does the left-hand value contain the right-hand value?
- <@ — Does the right-hand value contain the left hand value?

Containment

- Containment work at the top level of the json object only, and on full JSON structures.
- It does not apply to individual keys.
- It does not apply to nested elements.

@>

```
postgres=# select '{"a": 1, "b": 2}'::jsonb @> '{"a": 1}'::jsonb;
 ?column?
----------
 t
(1 row)


postgres=# select '[1, 2, 3]'::jsonb @> '[1, 3]'::jsonb;
 ?column?
----------
 t
(1 row)

postgres=# select '{"a": {"b": 7, "c": 8}}'::jsonb @>
                  '{"a": {"c": 8}}'::jsonb;
 ?column?
----------
 t
(1 row)
```

but.

```
postgres=# select '{"a": {"b": 7}}'::jsonb @> '{"b": 7}'::jsonb;
 ?column?
----------
 f
(1 row)

postgres=# select '{"a": 1, "b": 2}'::jsonb @> '"a"'::jsonb;
 ?column?
----------
 f
(1 row)
```

?, ?|, ?&

- True if:
  - ? — The key on the right-hand side appears in the left-hand side.
  - ?| ?& — Any of the array of keys on the right-hand side appear on the left-hand side.
  - PostgreSQL array type, not JSON array.

?, ?|, ?&

```
postgres=# select '{"a": 7, "b": 4}'::jsonb ? 'a';
 ?column?
----------
 t
(1 row)

postgres=# select '{"a": 7, "b": 4}'::jsonb ?& ARRAY['a', 'b'];
 ?column?
----------
 t
(1 row)

postgres=# select '{"a": 7, "b": 4}'::jsonb ?| ARRAY['a', 'q'];
 ?column?
----------
 t
(1 row)
```

but.

```
postgres=# select '{"a": {"b": 7, "c": 8}}'::jsonb ? 'b';
 ?column?
----------
 f
(1 row)

postgres=# select '[1, 2, 3, 4]'::jsonb ?| ARRAY[1, 100];
ERROR:  operator does not exist: jsonb ?| integer[]
LINE 1: select '[1, 2, 3, 4]'::jsonb ?| ARRAY[1, 100];
                                     ^
HINT:  No operator matches the given name and argument type(s). You might
need to add explicit type casts.

postgres=# select '[1, 2, 3, 4]'::jsonb ?| '[1, 2]'::jsonb;
ERROR:  operator does not exist: jsonb ?| jsonb
LINE 1: select '[1, 2, 3, 4]'::jsonb ?| '[1, 2]'::jsonb;
                                     ^
HINT:  No operator matches the given name and argument type(s). You might
need to add explicit type casts.
```

PGX POSTGRESQL EXPERTS, INC.

JSON functions

- Lots and lots and lots.
- Create JSON from records, arrays, etc.
- Expand JSON into records, arrays, rowsets, etc.
- Many have both json and jsonb versions.

Example: row_to_json

- Accepts an arbitrary row.
- Returns a json (not jsonb) object.
- For non-string/int/NULL types, uses the output function to create a string.
- Properly handles composite/array types.

PGX POSTGRESQL EXPERTS, INC.

Behold!

```
xof=# select row_to_json(rel.*) from rel where array_length(tags, 1) > 2  order by id limit 3;
                                                         row_to_json
-----------------------------------------------------------------------------------------------------------
 {"id":636572,"first_name":"OLENE","last_name":"OGRAM","tags":["female","square","violet"]}
 {"id":636744,"first_name":"SHAYNE","last_name":"GALPIN","tags":["female","square","silver","aquamarine","green","octogon"]}
 {"id":636769,"first_name":"YASMIN","last_name":"AKEN","tags":["female","red","green"]}
(3 rows)
```

PGX POSTGRESQL EXPERTS, INC.

But seriously…

- … can be used in a trigger to append to an audit table regardless of the schema.
- Extremely useful for shared triggers.

Example: jsonb_each_text

- Takes a jsonb object, and returns a rowset of key/value pairs.
- Returns each as text object.
- Can be used to write the world's most expensive EAV query!

Behold!

```
xof=# WITH s AS (
xof(# SELECT row_to_json(rel.*)::jsonb AS j FROM rel ORDER BY id LIMIT 3
xof(# ) SELECT (s.j->>'id')::bigint AS entity, key as attribute, value FROM s,
LATERAL jsonb_each_text(s.j) WHERE key <> 'id';
 entity | attribute  |   value
--------+------------+------------
 636526 | tags       | ["female"]
 636526 | last_name  | EILTS
 636526 | first_name | REGENA
 636527 | tags       | ["male"]
 636527 | last_name  | POTO
 636527 | first_name | ANTONIO
 636528 | tags       | ["female"]
 636528 | last_name  | LUFSEY
 636528 | first_name | ROXY
(9 rows)
```

PGX POSTGRESQL EXPERTS, INC.

But seriously…

- … it can be used to expand jsonb into relational data for JOINs and the like.
- Often more efficient than using the extraction operators.

Indexing.

Indexing json

- The textual json type has no inherent indexing (that you'd ever use).
- Can do an expression index on extracted values…
- … but that requires knowing exactly which fields / elements you are going to query on.
- If you know that, make that data relational.

PGX POSTGRESQL EXPERTS, INC.

jsonb indexing.

- jsonb has GIN indexing.
- Default type supports queries with the @>, ?, ?& and ?| operators.
- The query must be against the top-level object for the index to be useful.
- Can query nested objects, but only in paths rooted at the top level.

jsonb_path_ops

- Optional GIN index type for jsonb.
- Only supports @>.
- Hashes paths for each item, rather than just storing the key itself.
- Faster for @> operations with nesting.

PGX POSTGRESQL EXPERTS, INC.

jdoc @> '{"tags": ["qui"]}'

- Both index types support this.
- jsonb_ops (the default) will seach for everything that has "tags", has "qui", AND them, and then do a recheck for the path structure.
- jsonb_path_ops will go directly to entries for that path.

Which to use?

- If you just need @>, jsonb_path_ops will probably be faster.
- If you need the other supported operators, you need jsonb_ops.
- But let's find out!


ADAC
ÖAMTC
FO4304OZ01
0.15

Test results.

The Usual Caveats

- The universe of possible workloads and schemas is infinite.
- Always build and test using data that simulates your real application.
- Don’t take these results as being applicable to every situation.
- Relative, not absolute results.

That said…

- Four column schema:
  - id — Primary key, bigint.
  - first_name, last_name — Text.
  - tags — Array of short text tags. Two extremely common ones (one per record), a diminishing number of rare ones.

PGX POSTGRESQL EXPERTS, INC.

The test setup.

- Amazon i2.2xlarge instance.
- Ubuntu.
- PostgreSQL 9.4.0.
- Basic tuning for instance size.

Test data.

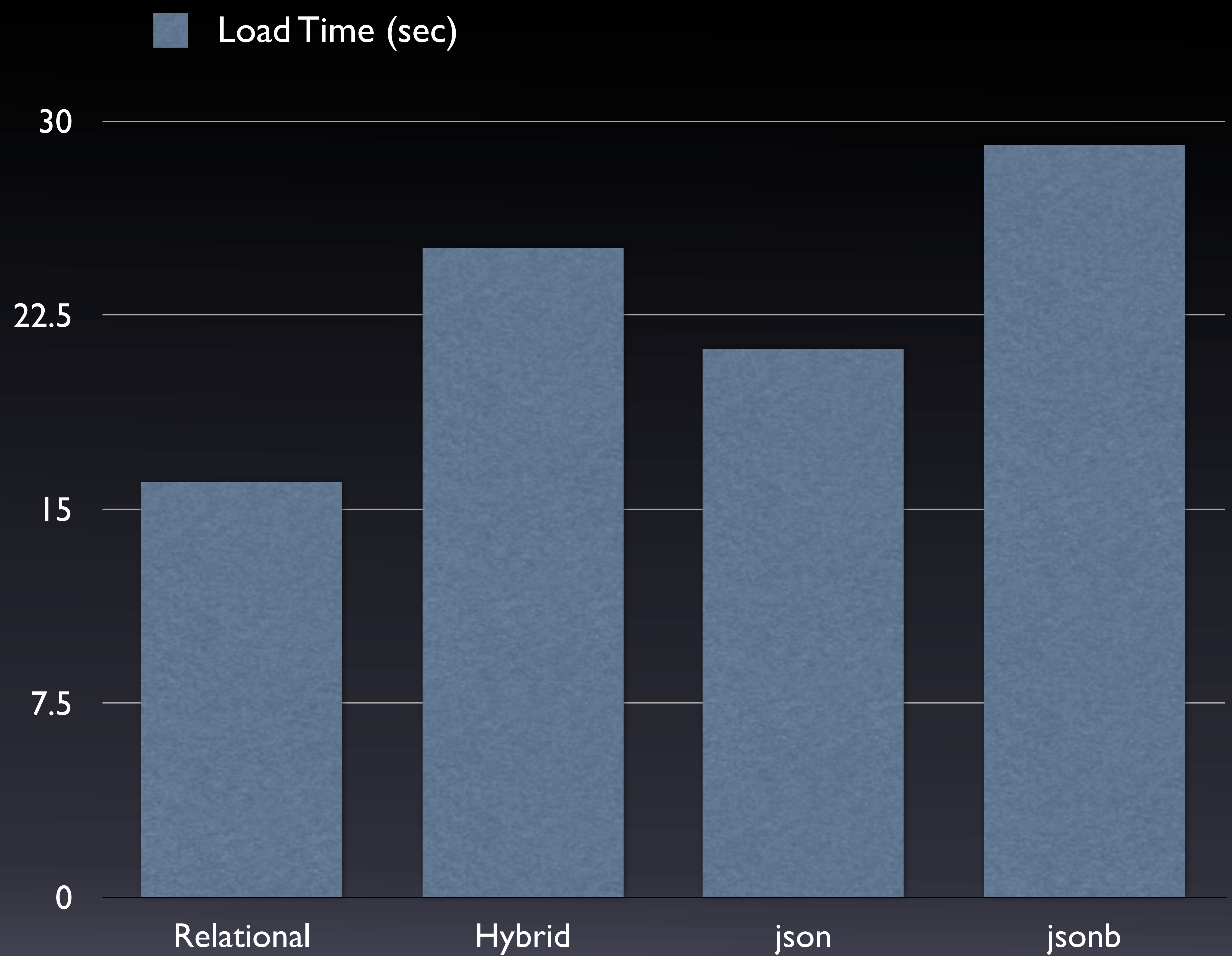
- 10,000,000 records generated at random.
- Schemas:
  - Pure relational data.
  - hybrid (names in relational, tags jsonb).
  - json and jsonb for non-ID.

Methodology

- 100 iterations per test, top and bottom 10 rejected.
- Query execution time only; does not include time to return results.
  - Python test harness can distort considerably if objects need to be created.

Test #1: Load

- Load 10,000,000 records using COPY.
- No index rebuilds.
- Relational, "hybrid," all json, all jsonb.



Load Time (sec)
30
22.5
15
7.5
0
Relational
Hybrid
json
jsonb

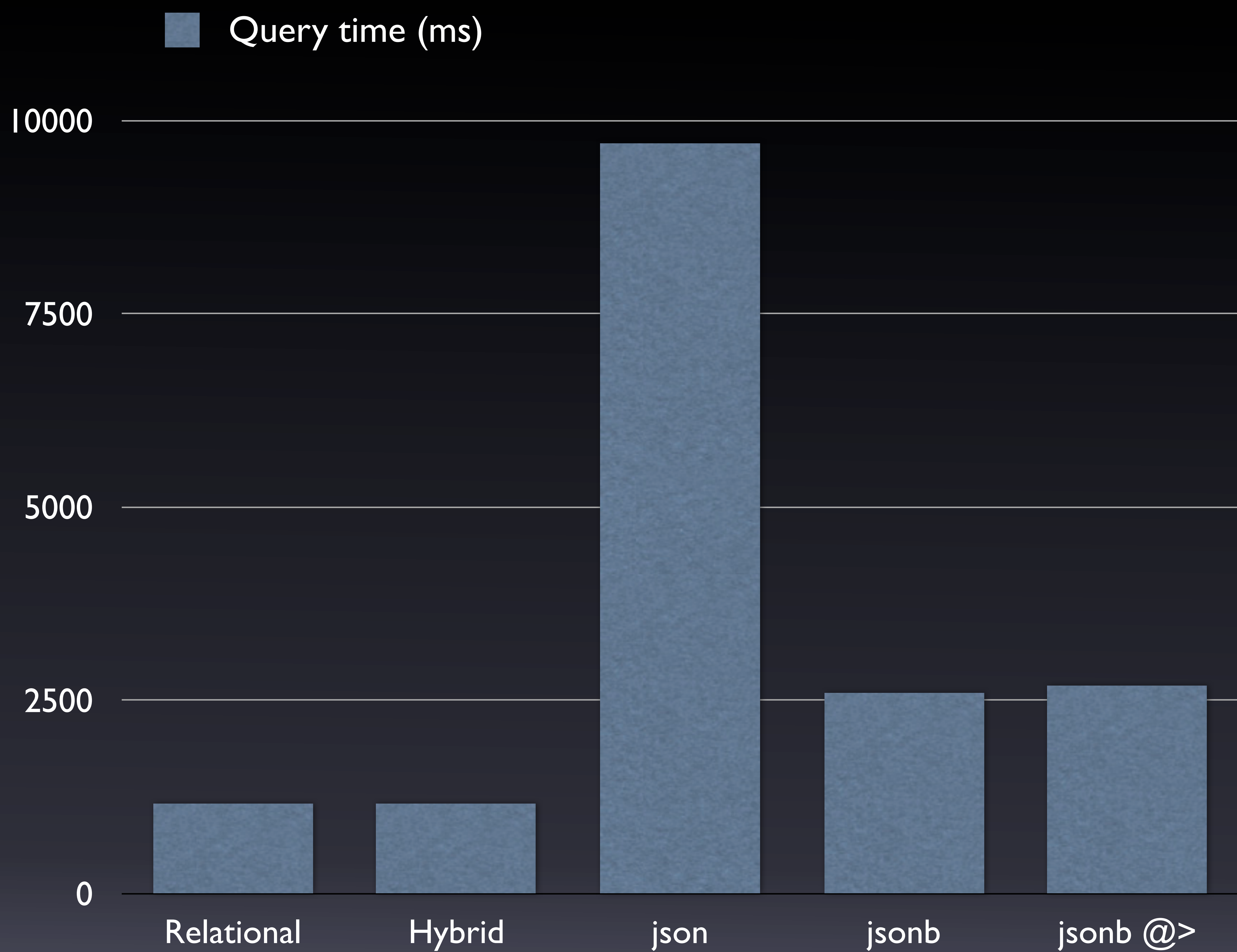
Test #1: Results

- Relational beats everything (no surprise).
- jsonb is slower to load than json.
  - Parsing and conversion time.
- The same order of magnitude.

Test #2: Sequential scan for a single last name.

- Scan table sequentially (no index) for a single last name.
- Uses a relational field for relational and hybrid.
- Uses ->> operator for json and jsonb.
- Also tried with @> operator for jsonb.

PGX POSTGRESQL EXPERTS, INC.


Query time (ms)
10000
7500
5000
2500
0
Relational
Hybrid
json
jsonb
jsonb @>

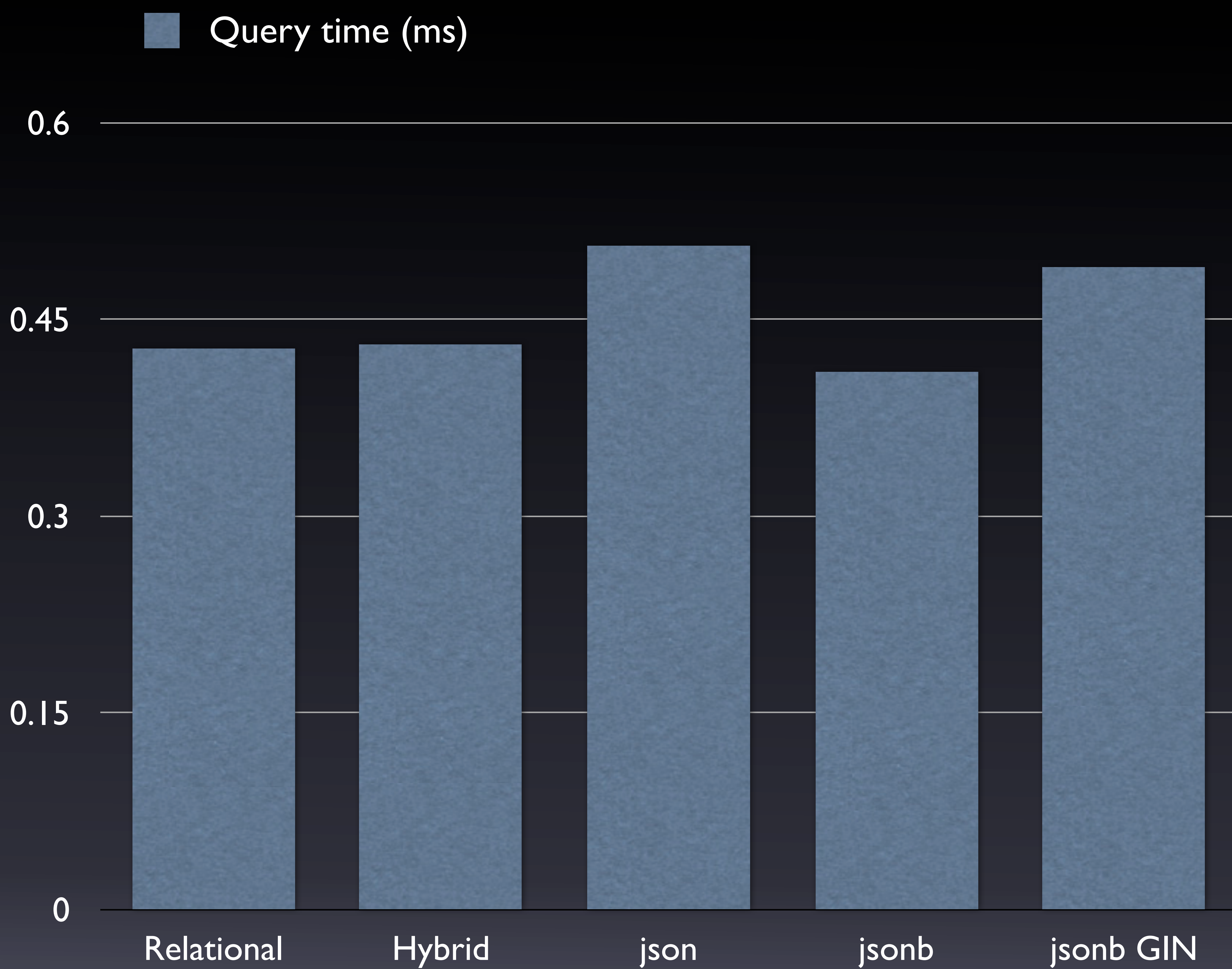
Test #2: Results.

- json dramatically slower than jsonb.
- Relational faster than jsonb by about 2x.
- ->> and @> operators roughly same speed in this application.

Test #3: b-tree index lookup by name.

- Create a traditional b-tree index.
- Directly on last_name for relational and hybrid.
- Expression index on (jdoc->>'last_name') for json and jsonb.
- Also tried GIN index on jsonb field, using @>.

PGX POSTGRESQL EXPERTS, INC.



PGX POSTGRESQL EXPERTS, INC.

Test #3: Results.

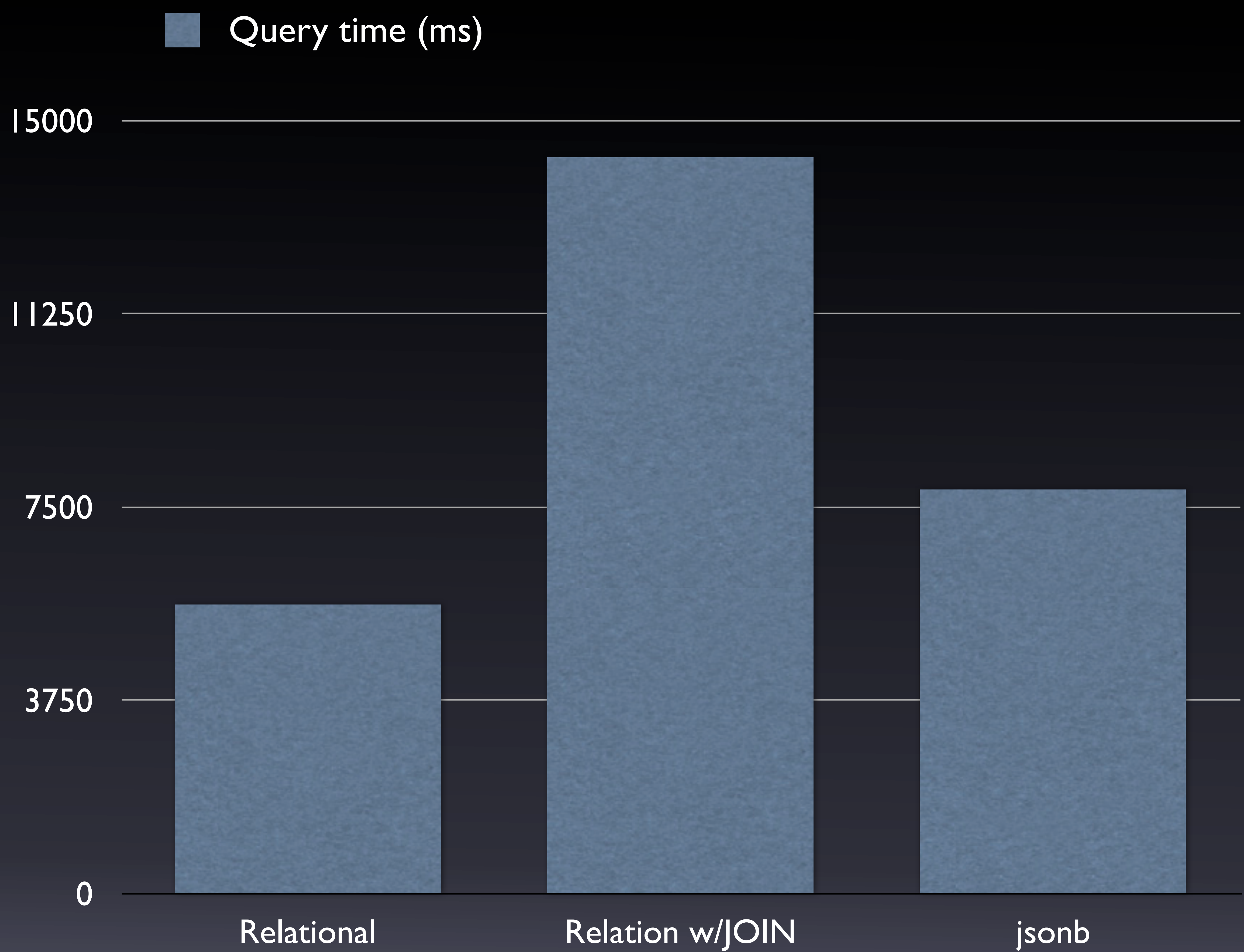
- All of comparable speed.
- jsonb actually faster than anything else!
- json somewhat slower due to extraction overhead.
- Always the fastest way to look up a highly selective field.

Test #3: Results, 2

- jsonb w/GIN very comparable to b-tree index.
- Didn’t have to specify a particular field in advance.
- Huge improvement over 9.3 days.

Test #4: Common tag lookup by seq scan.

- Every record has a ‘male’ or ‘female’ tag, 50%/50%.
- Scan looking for all of one.
- Uses @> operator for tag array.
- Uses @> operator for jsonb.
- Also tried with a secondary table of tags to which we join.



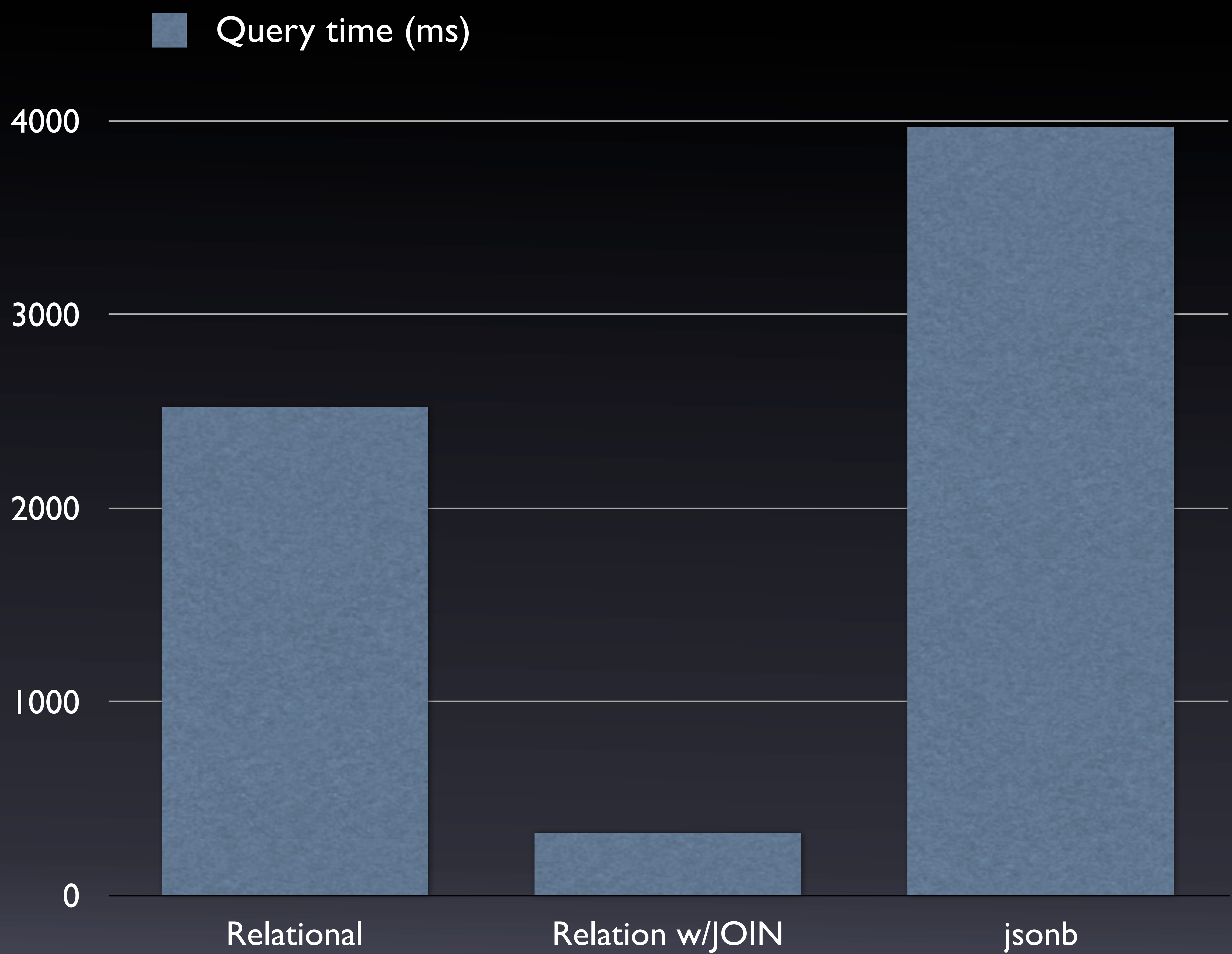
PGX
POSTGRESQL
EXPERTS, INC.

Test #4: Results.

- Secondary join table a huge loss in this scenario.
- jsonb slower than relational, but within the same general range.

Test #5: Rare tag lookup by seq scan.

- Scan for a rare tag (0.075% of records).
- Uses @> operator for relational.
- Uses @> operator for jsonb.
- Also tried with JOIN table.
  - In both cases, JOIN table indexed on tag, but didn't use in seq scan case.



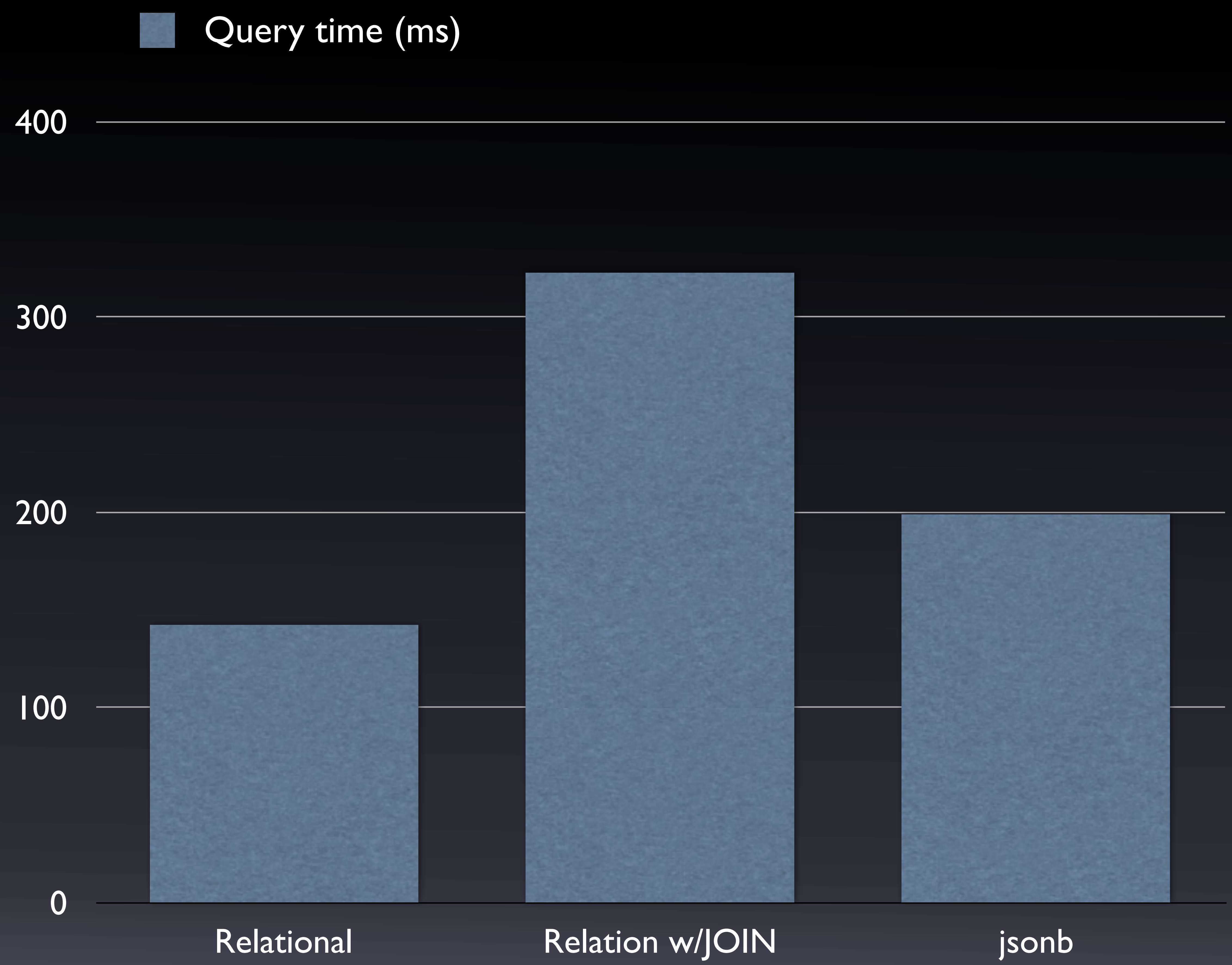
PGX POSTGRESQL EXPERTS, INC.

Test #5: Results.

- Secondary join table a huge win in this scenario.
  - Unsurprising, since it can isolate the rare tag faster.
- jsonb remains slower but comparable.

Test #6: Rare tag lookup by index.

- Create a GIN index on relational array field and jsonb document
- Use @> operator for tag array.
- Use @> operator for jsonb.
- Also tried with JOIN table.



PGX
POSTGRESQL
EXPERTS, INC.

Test #6: Results.

- Relational fastest in this situation…
- … but jsonb performs comparably.
- If you are storing rare tags and don't need full JSON, consider an array field.

Note: GIN indexes and selectivity.

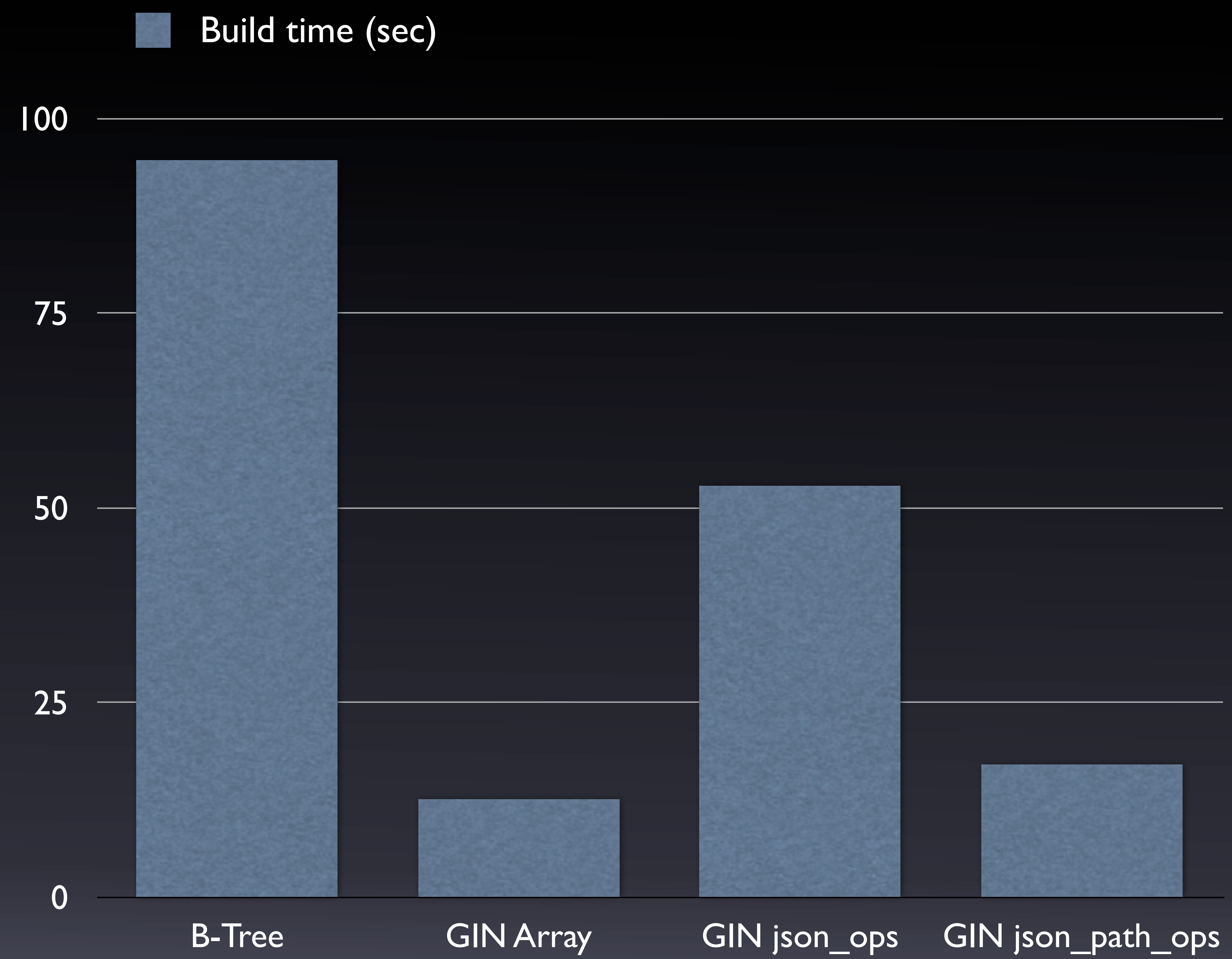
- GIN indexes on jsonb fields have hard-wired selectivity calculations (as of 9.4).
- Will almost always use the index even if selectivity is very low.
- This can result in bad performance in cases of low selectivity.
- An area that definitely needs attention.

PGX POSTGRESQL EXPERTS, INC.

Test #7: Index Creation.

- Timed index creation for the various index types.
- last_name b-tree on relational.
- GIN on relational array.
- GIN json_ops and json_path_ops on jsonb.

PGX POSTGRESQL EXPERTS, INC.



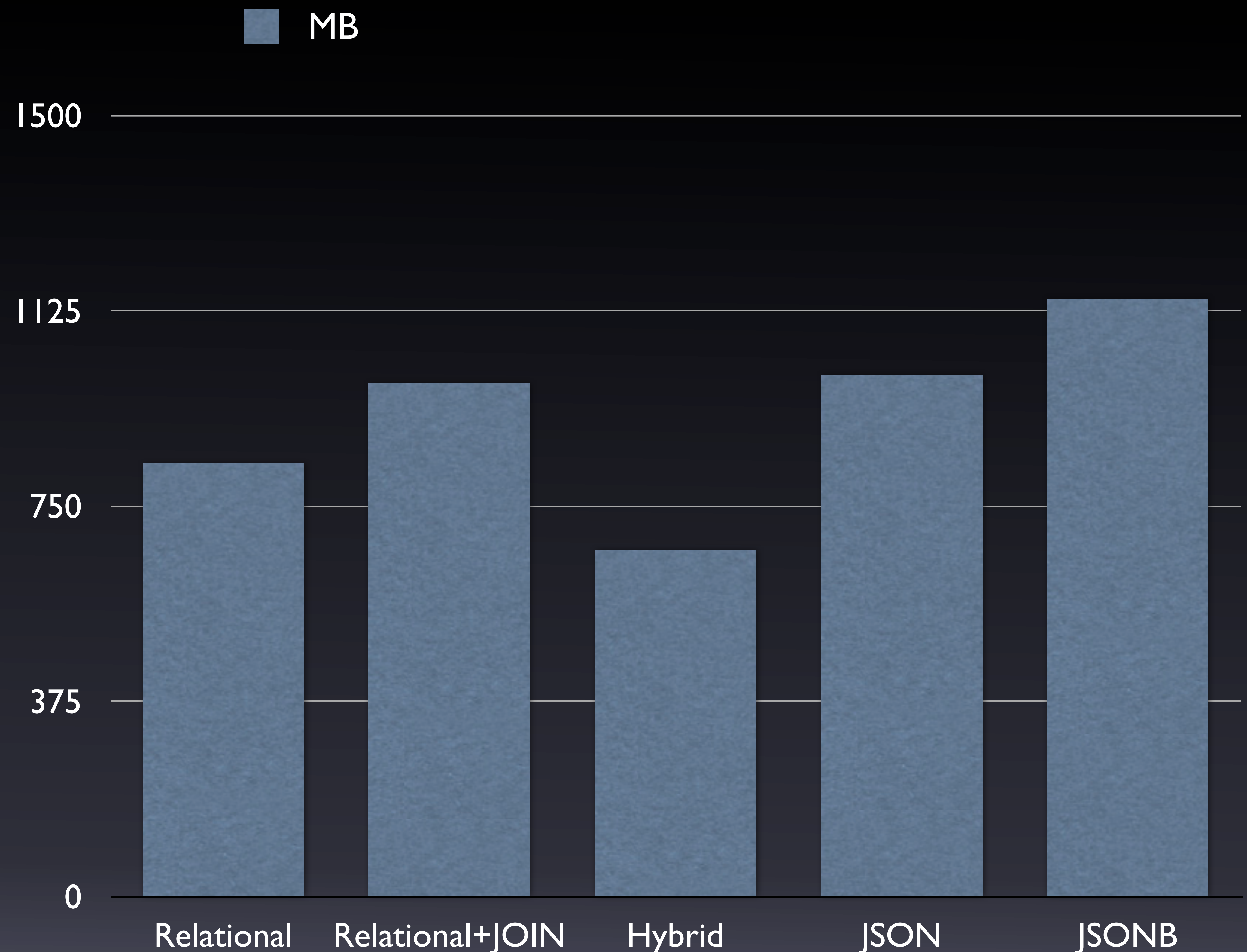
PGX
POSTGRESQL
EXPERTS, INC.

Test #7: Results.

- GIN build time is very fast.
- json_path_ops build time is very fast.
- GIN indexing on arrays, too.

Test #8: Relation size.

- Total size, excluding indexes.
- For relation + JOIN table, includes JOIN table as well.



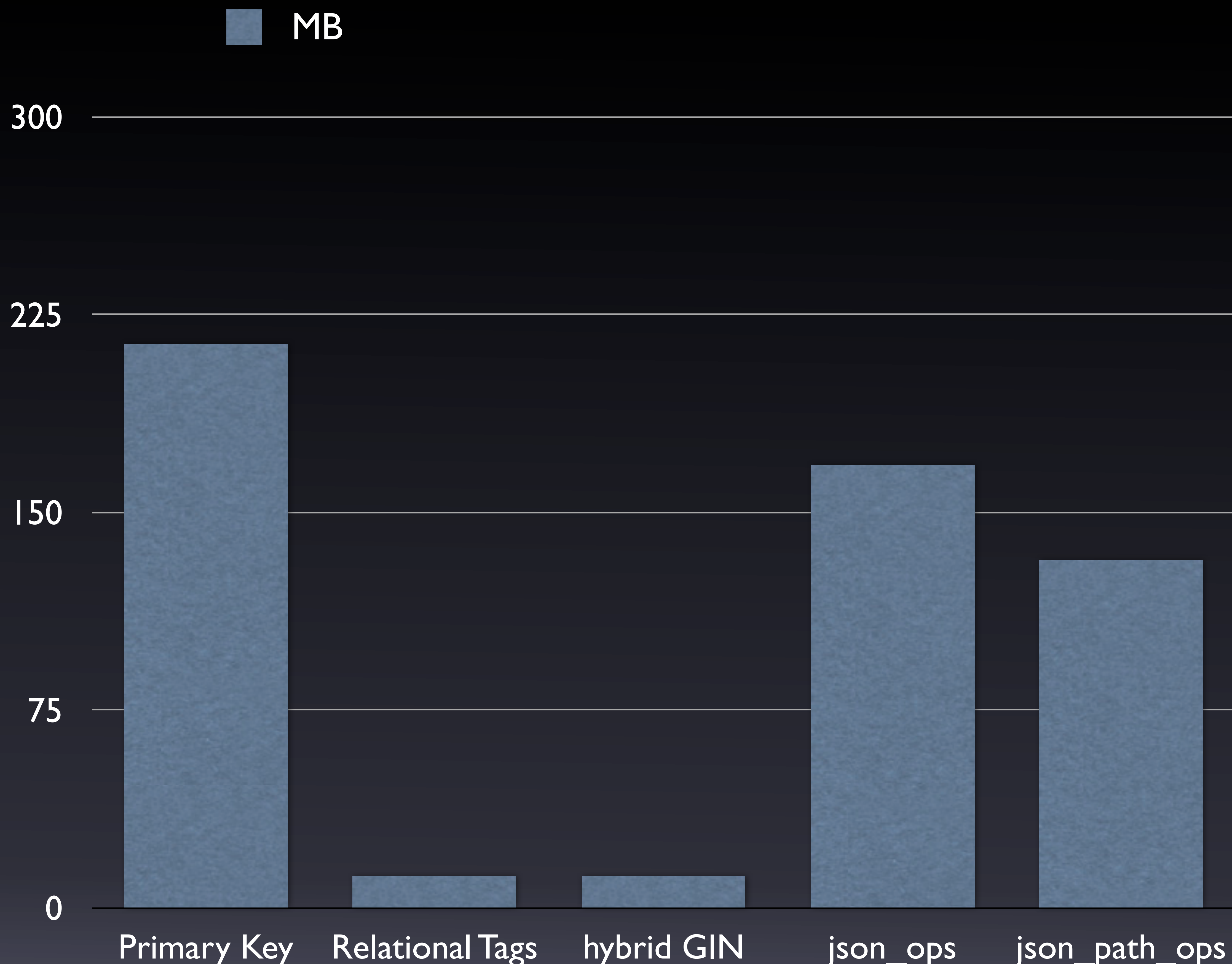
PGX
POSTGRESQL
EXPERTS, INC.

Test #8: Results.

- Generally comparable size.
- hybrid is the most compact by a significant margin.
- jsonb slightly larger than json due to internal structure overhead.

Test #9: Index size.

- Size of various indexes.
- Primary key index (same for all tables).
- GIN index on relational tags.
- json_ops
- json_path_ops


MB
300
225
150
75
0
Primary Key
Relational Tags
hybrid GIN
json_ops
json_path_ops

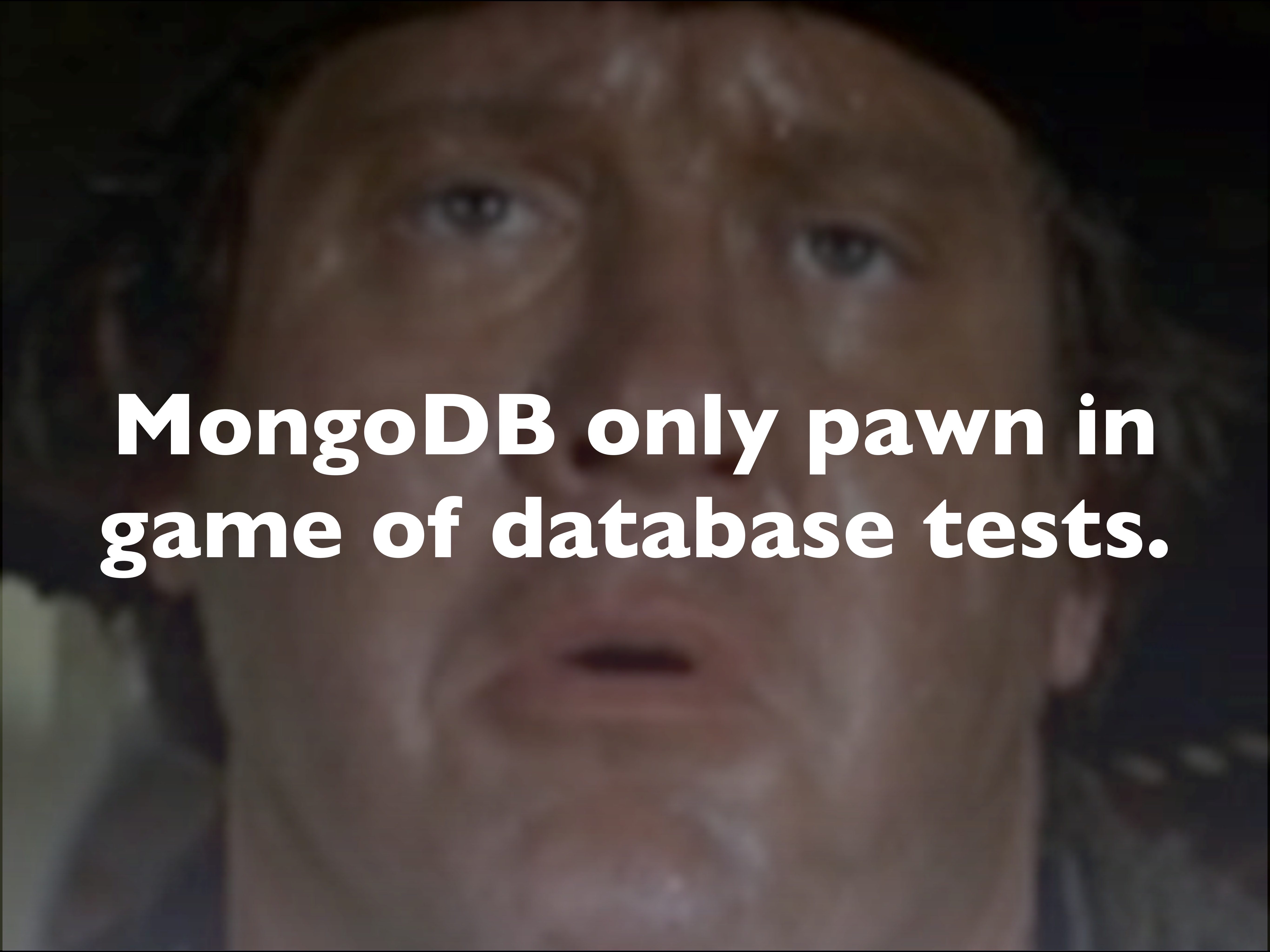
Test #9: Results.

- Indexes on just the tags are very compact.
- json_path_ops indexes are (as expected) somewhat smaller than json_ops indexes.

Now that we know this, what do we know?

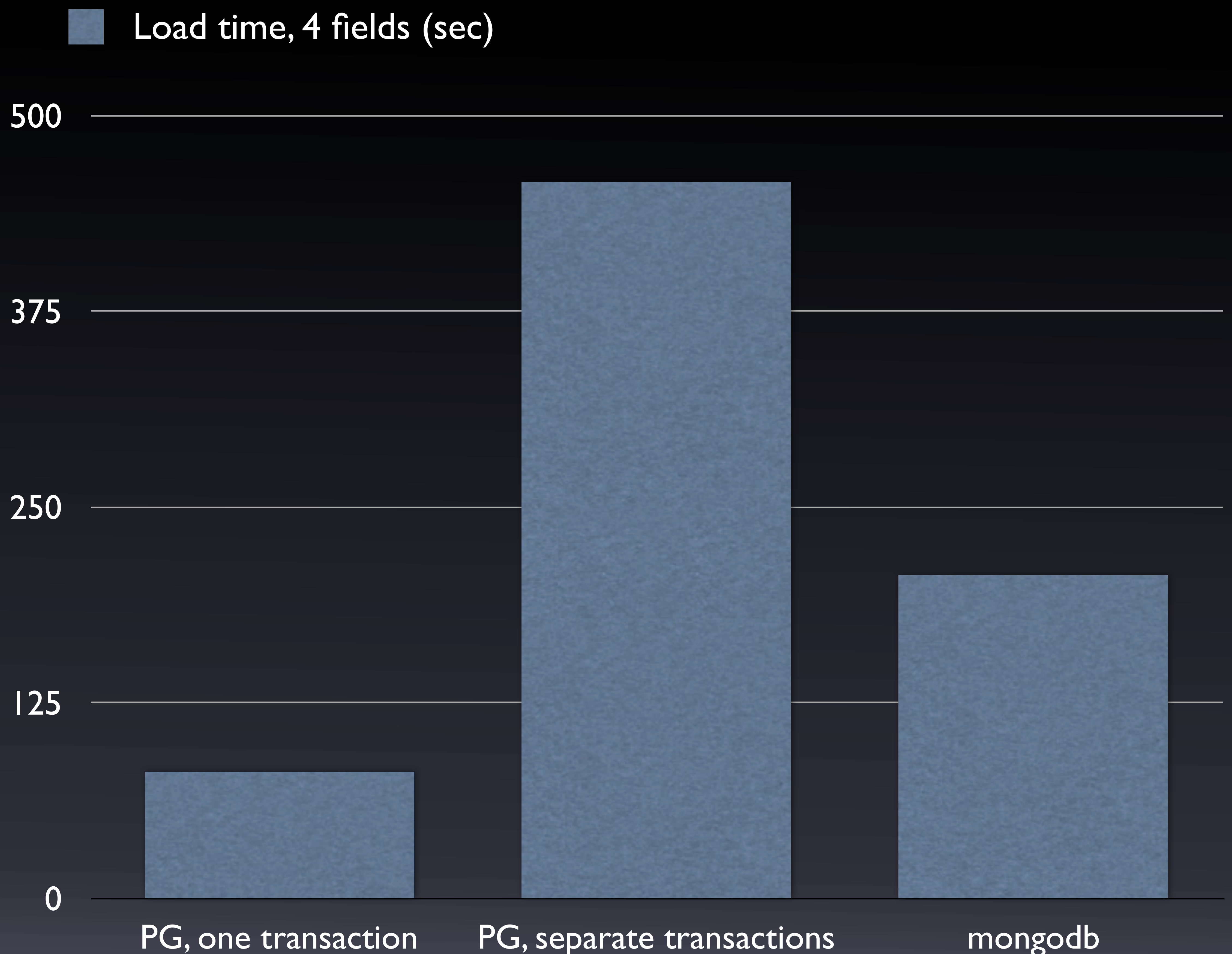
The One-Slide Oversimplification.

- Use relational data for the basic set of attributes.
- Use either array fields or jsonb for extended attributes.
- Use file-system storage for really big stuff.
- Always use jsonb. No reason to use json.


MongoDB only pawn in
game of database tests.

How do we compare to mongodb?

- mongodb 3.0.1
  - New, faster storage engine.
- Two data sets:
  - Each with 1 million records.
  - One with 4, one with 200, json fields.



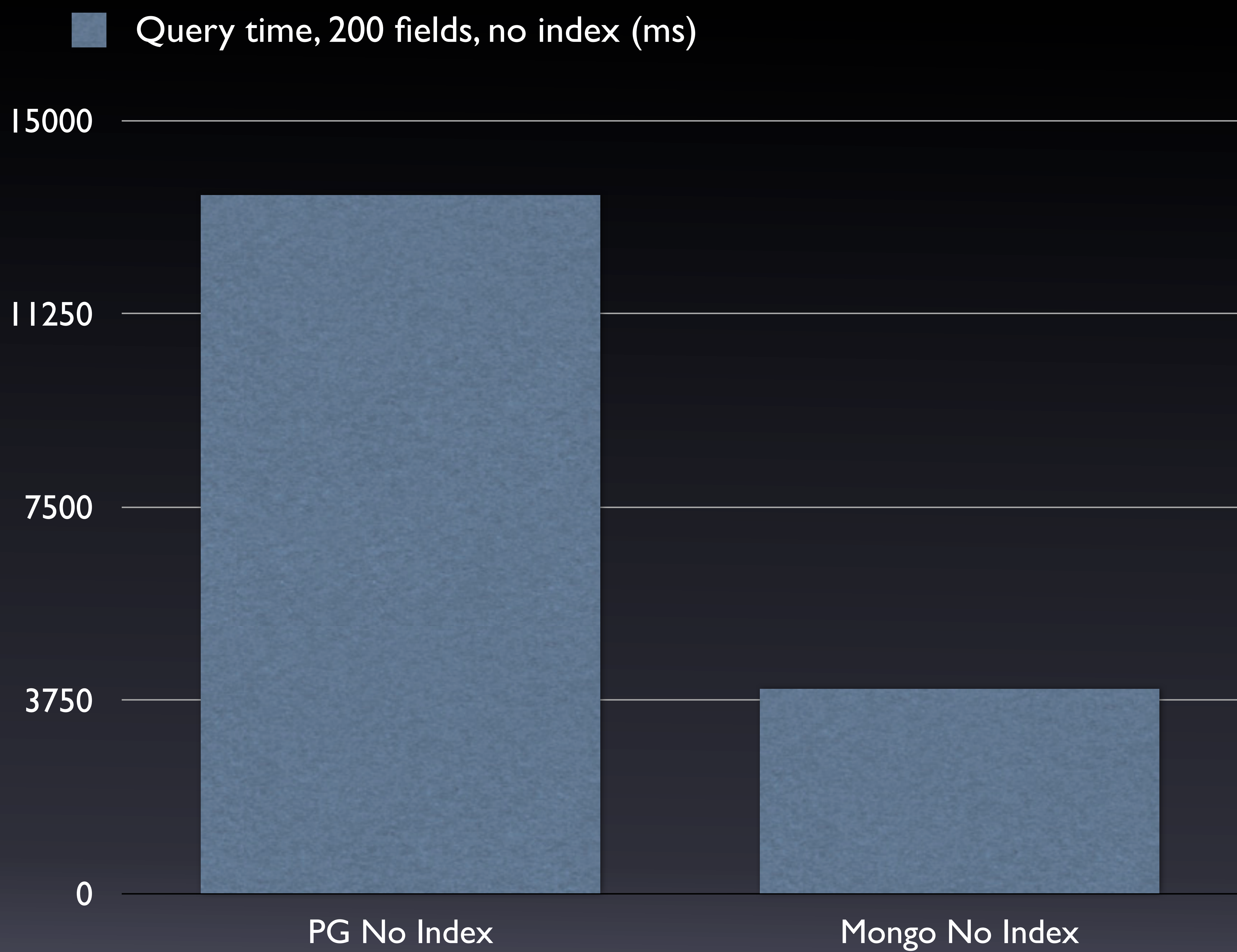
PGX
POSTGRESQL
EXPERTS, INC.


Load time, 200 fields (sec)
800
600
400
200
0
PG, one transaction
PG, separate transactions
mongodb

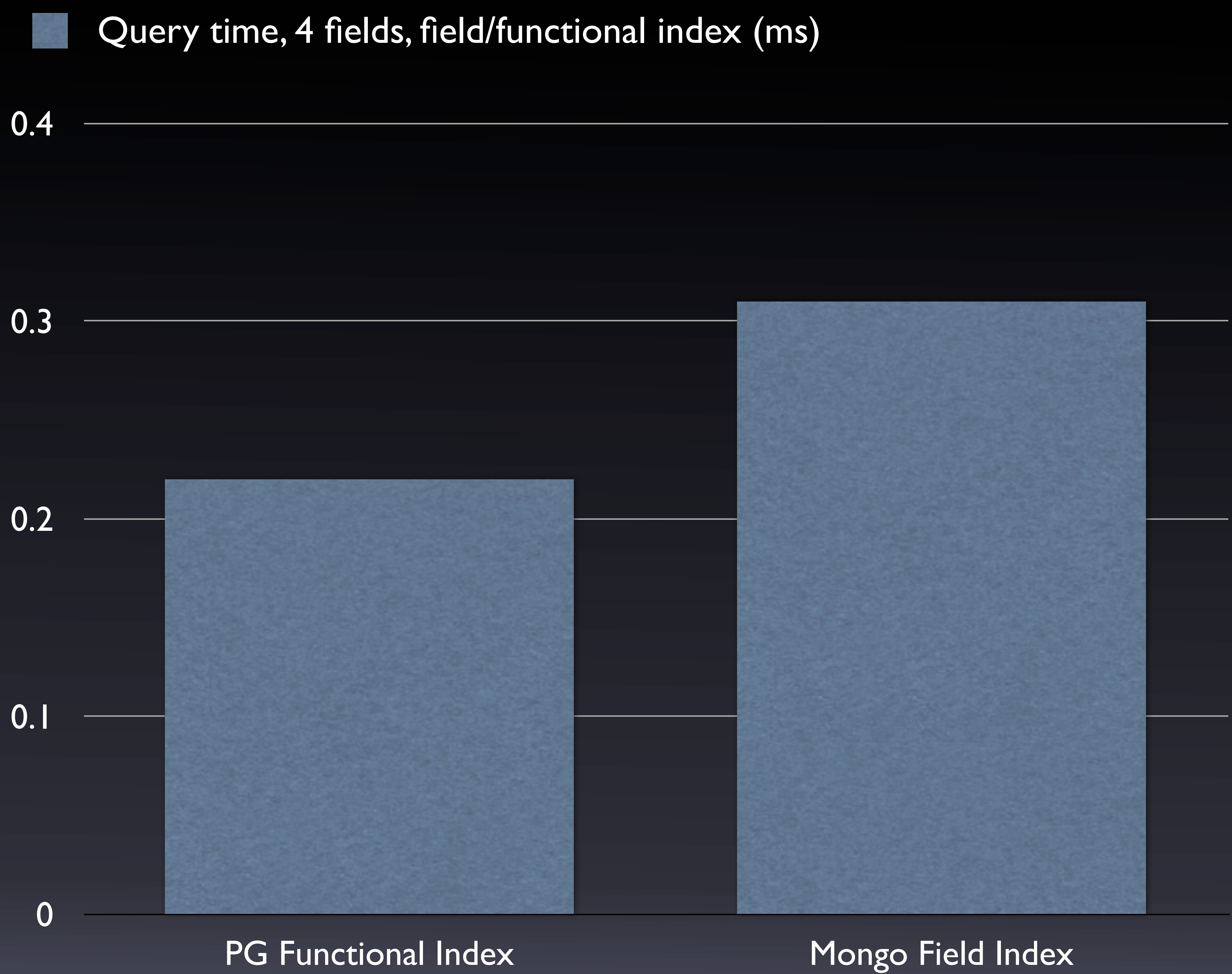
Query for a single field, single value

- PostgreSQL:
  - No index, functional index, GIN index, GIN index, with jsonb_path_ops,
- Mongo
  - No index, field index.

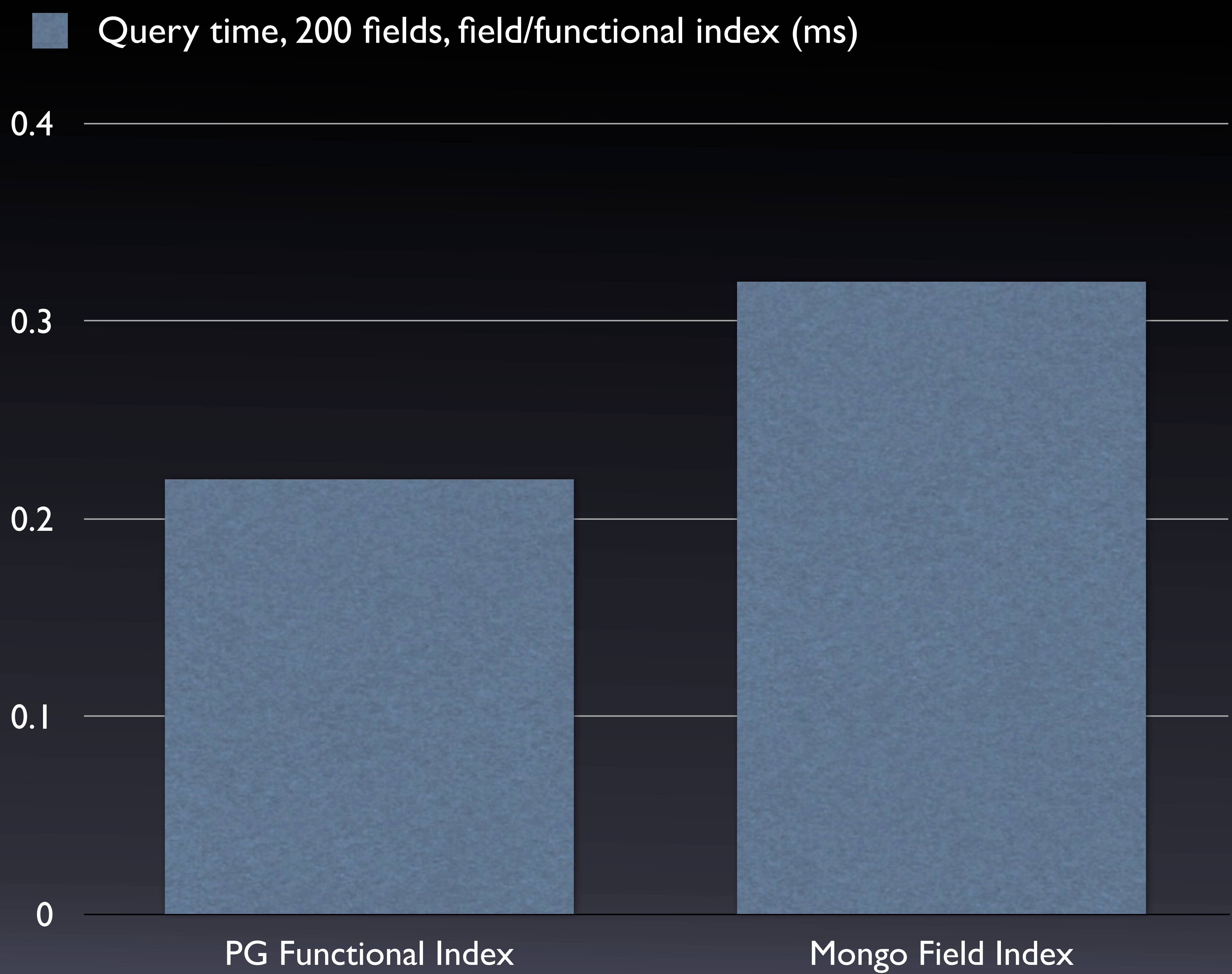
PGX
POSTGRESQL
EXPERTS, INC.



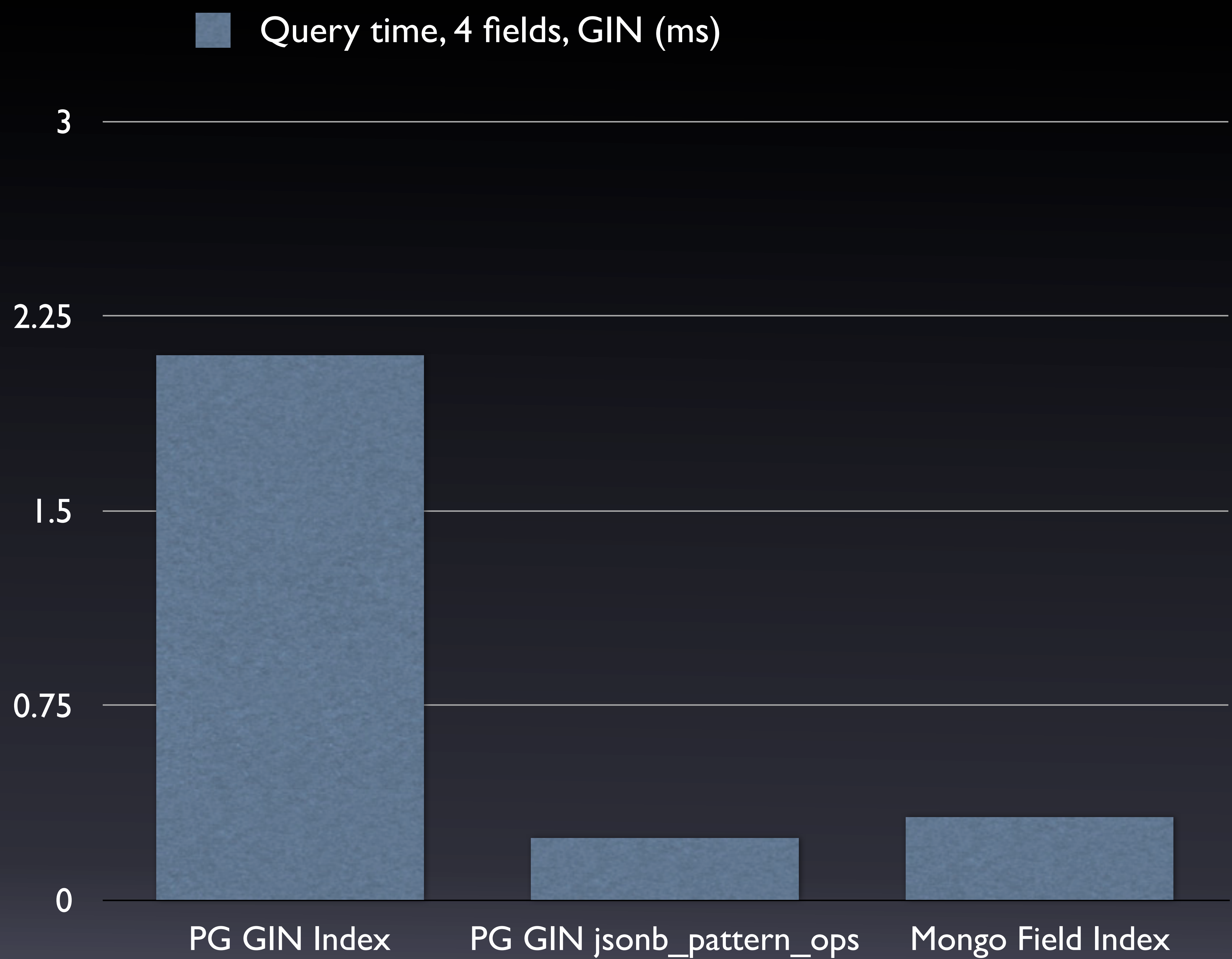
PGX
POSTGRESQL
EXPERTS, INC.



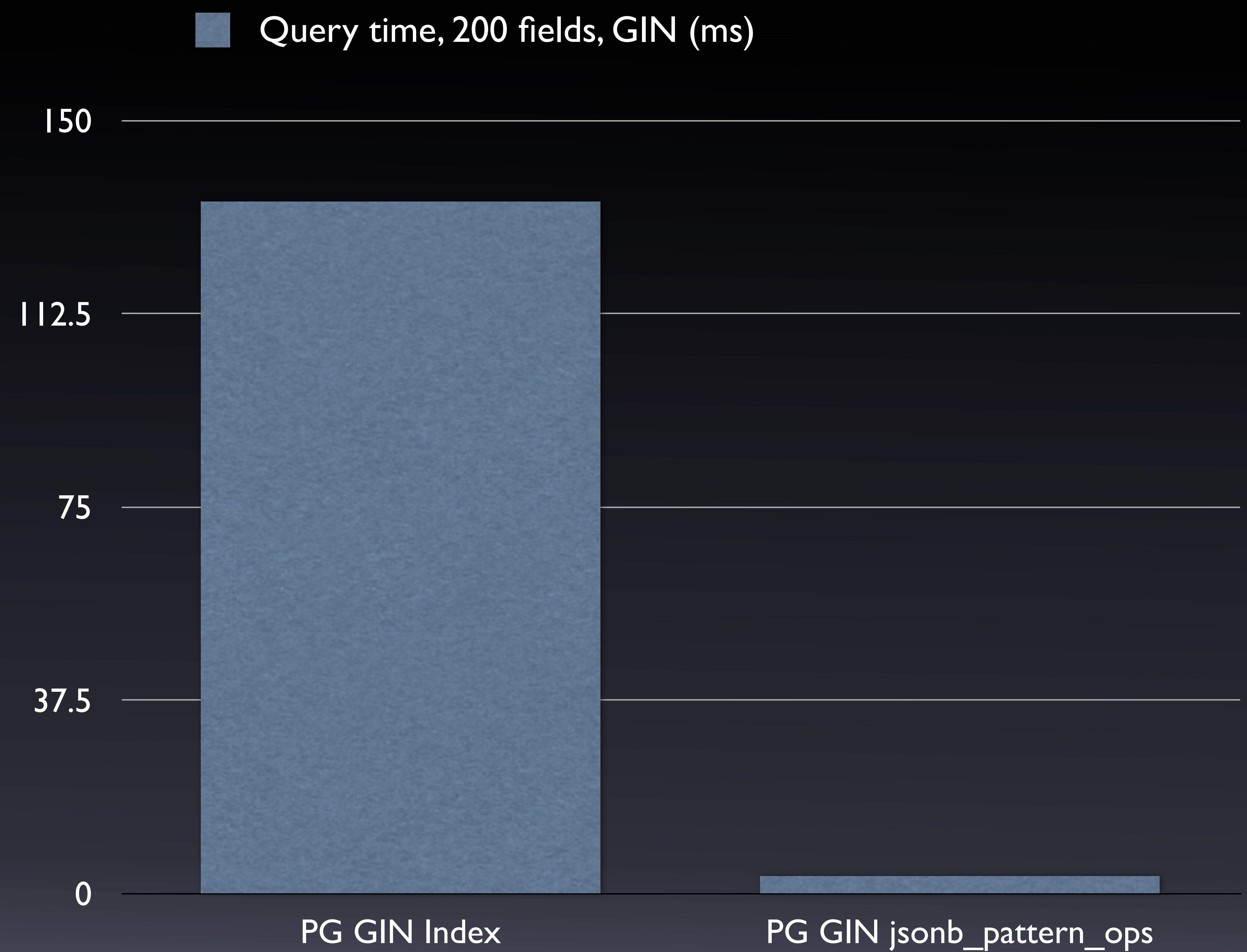
PGX
POSTGRESQL
EXPERTS, INC.



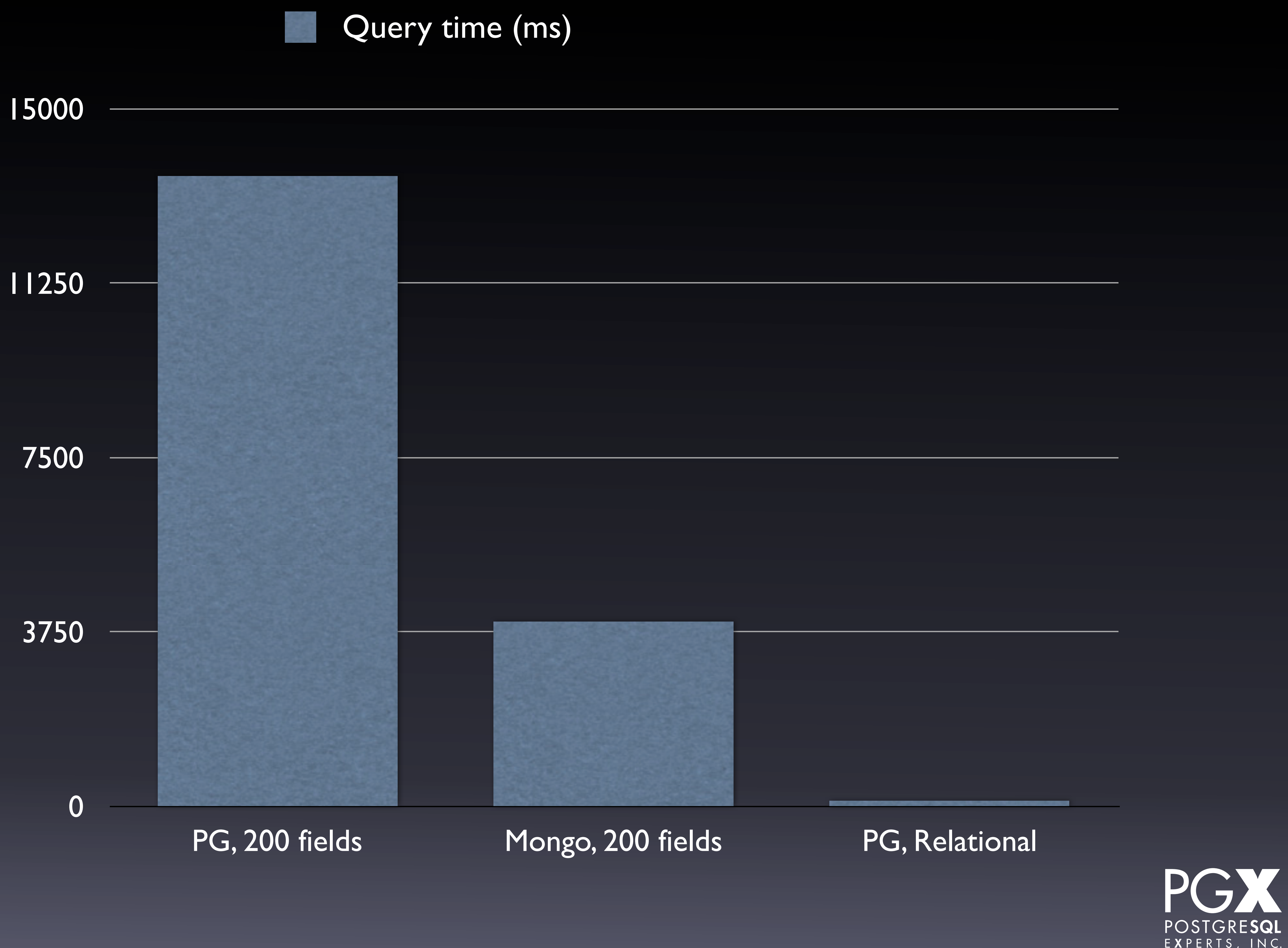
PGX
POSTGRESQL
EXPERTS, INC.



PGX
POSTGRESQL EXPERTS, INC.



PGX
POSTGRESQL
EXPERTS, INC.


Query time (ms)
15000
11250
7500
3750
0
PG, 200 fields
Mongo, 200 fields
PG, Relational
PGX
POSTGRESQL
EXPERTS, INC.

Mongo Notes.

- Mongo 3.0 is much improved.
- Performs very well in extracting a single field from a very large JSON body.
- Indexing a single field performs comparably to a PostgreSQL index.
- No real equivalent of PostgreSQL GIN.
- Game on!

Conclusions.

- Mongo does well for documents with a large number of JSON fields, sequentially scanned.
- PostgreSQL `jsonb` performs better in most other cases.
- PostgreSQL relational performance destroys JSON performance, of course.

And here we are!

- PostgreSQL 9.4 has world-class JSON support.
- Mix and match! Use JSON for what is good for, relational data for speed.
    - (Check out ToroDB.)

Thank you!

Questions?

- thebuild.com — personal blog.
- pgexperts.com — company website.
- Twitter @Xof
- christophe.pettus@pgexperts.com

PGX POSTGRESQL EXPERTS, INC.