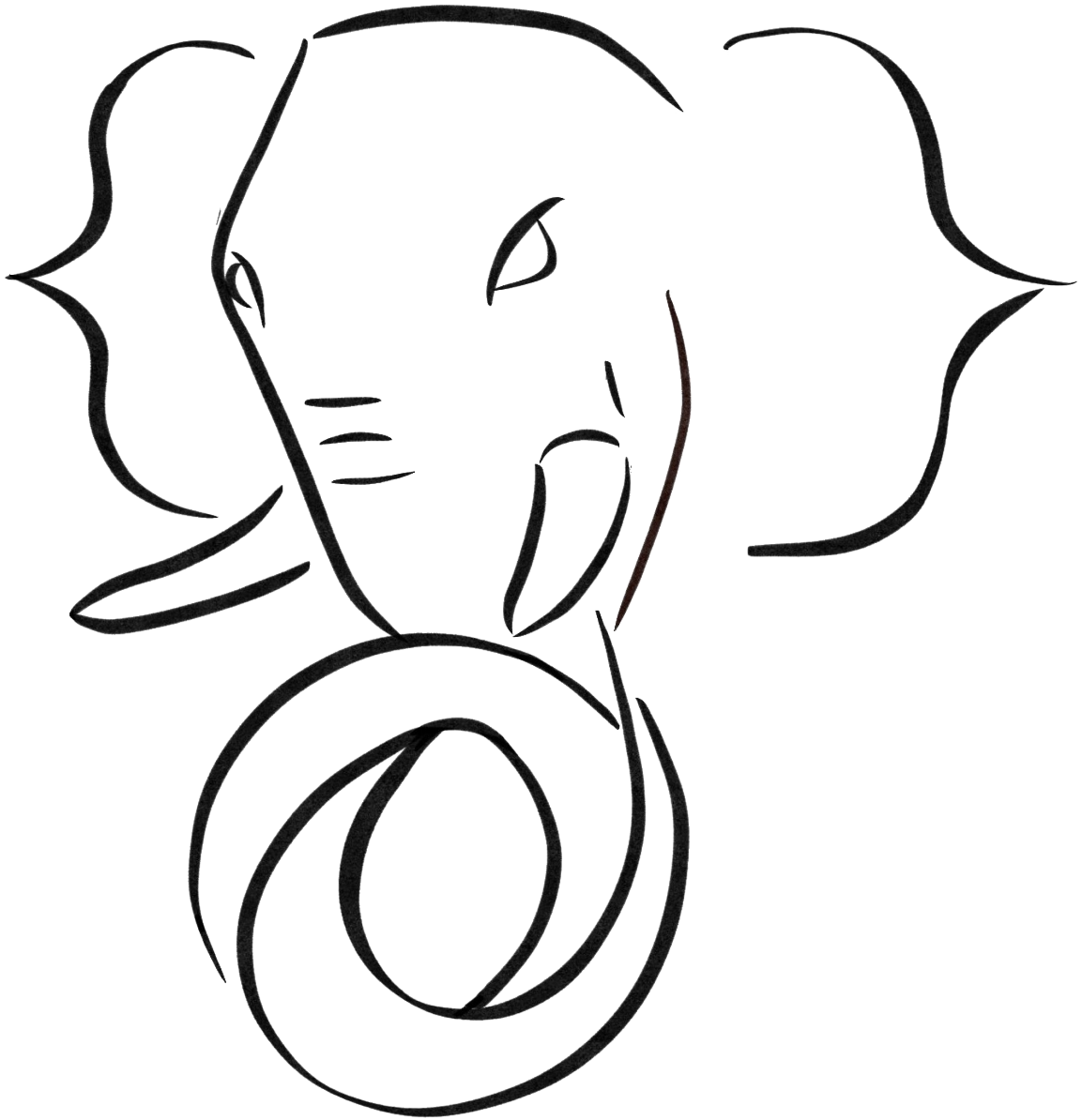
FOSDEM'21



Speed up the JSONB.

What we can do to improve performance.

Oleg Bartunov
Nikita Glukhov

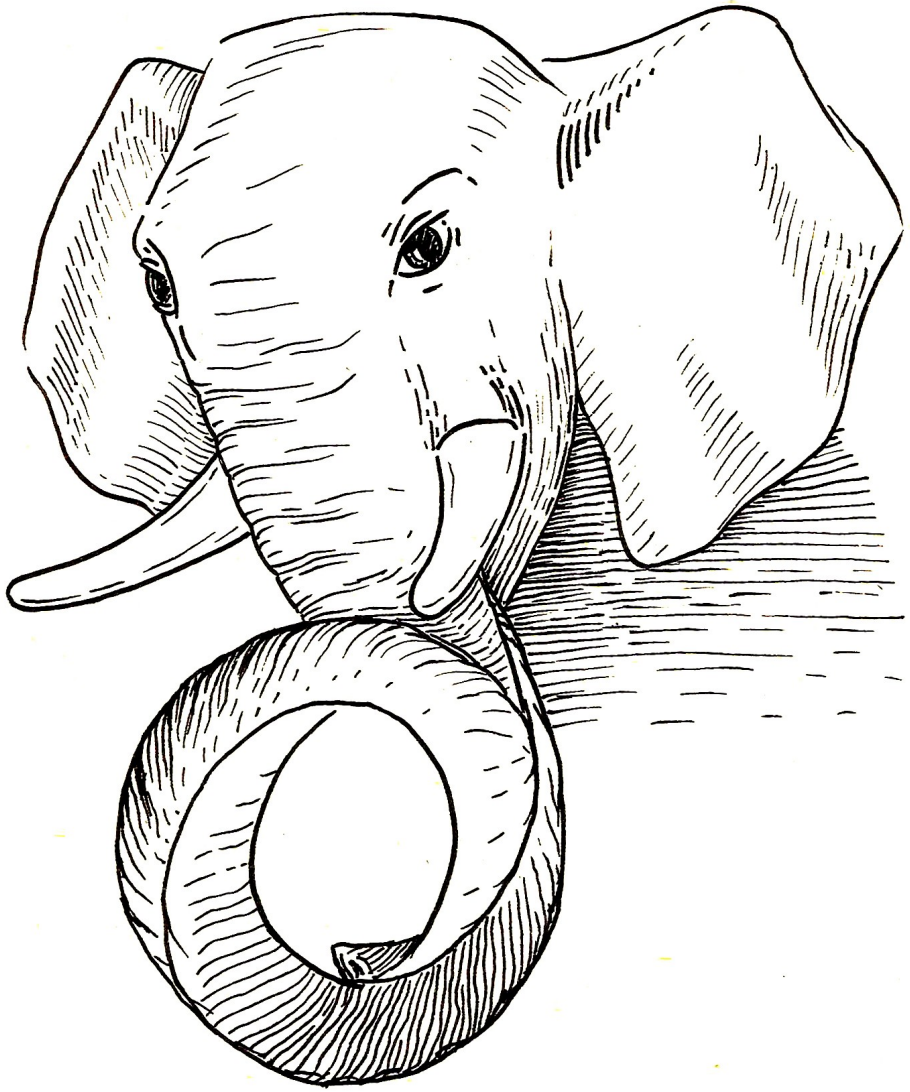
PROFESSIONAL Postgres

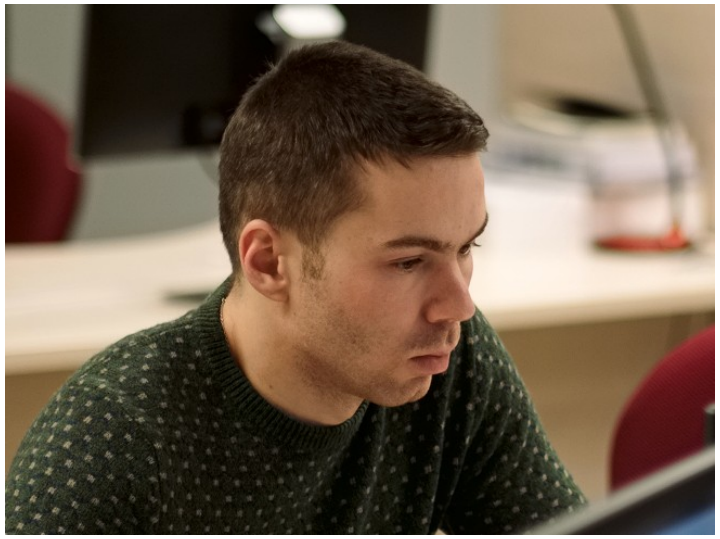
Since Postgres95

Research scientist @
Moscow University
CEO Postgres Professional
Major PostgreSQL contributor

PostgresPro

Nikita Glukhov





Senior developer @Postgres Professional
PostgreSQL contributor

Major CORE contributions:
Jsonb improvements
SQL/JSON (Jsonpath)
KNN SP-GiST
Opclass parameters

Current development:
SQL/JSON functions
Jsonb performance

PostgresPro

Quick Summary

- Jsonb is ubiquitous and is constantly developing
  - JSON[B] Roadmap V2, Postgres Professional Webinar, Sep 17, 2020
  - JSON[B] Roadmap V3, Postgres Build 2020, Dec 8, 2020
- There is a need to improve its performance:
  - Investigate and optimise access to keys (metadata) for nontoasted and toasted jsonb
- We demonstrate step-by-step performance improvements, which lead to significant speedup (orders of magnitude)
  - Repositories: Jsonb partial decompression, Jsonb partial detoast
  - Slides of this talk (PDF, Video)
- Contact obartunov@postgrespro.ru, n.gluhov@postgrespro.ru for collaboration.

PostgresPro

Motivational example (synthetic test)

- A table with 100 jsonbs of different sizes (100B - 10MB, compressed to 100B - 20KB):

```
CREATE TABLE test_toast AS SELECT
  i id,
  jsonb_build_object(
    'key1', i, 'key2', i::text,
    'key3', long_value,
    'key4', i, 'key5', i::text
  ) jb
FROM
  generate_series(1, 100) i,
  repeat('a', pow(10, 1 + 6.0 * i / 100.0)::int) long_value;
```

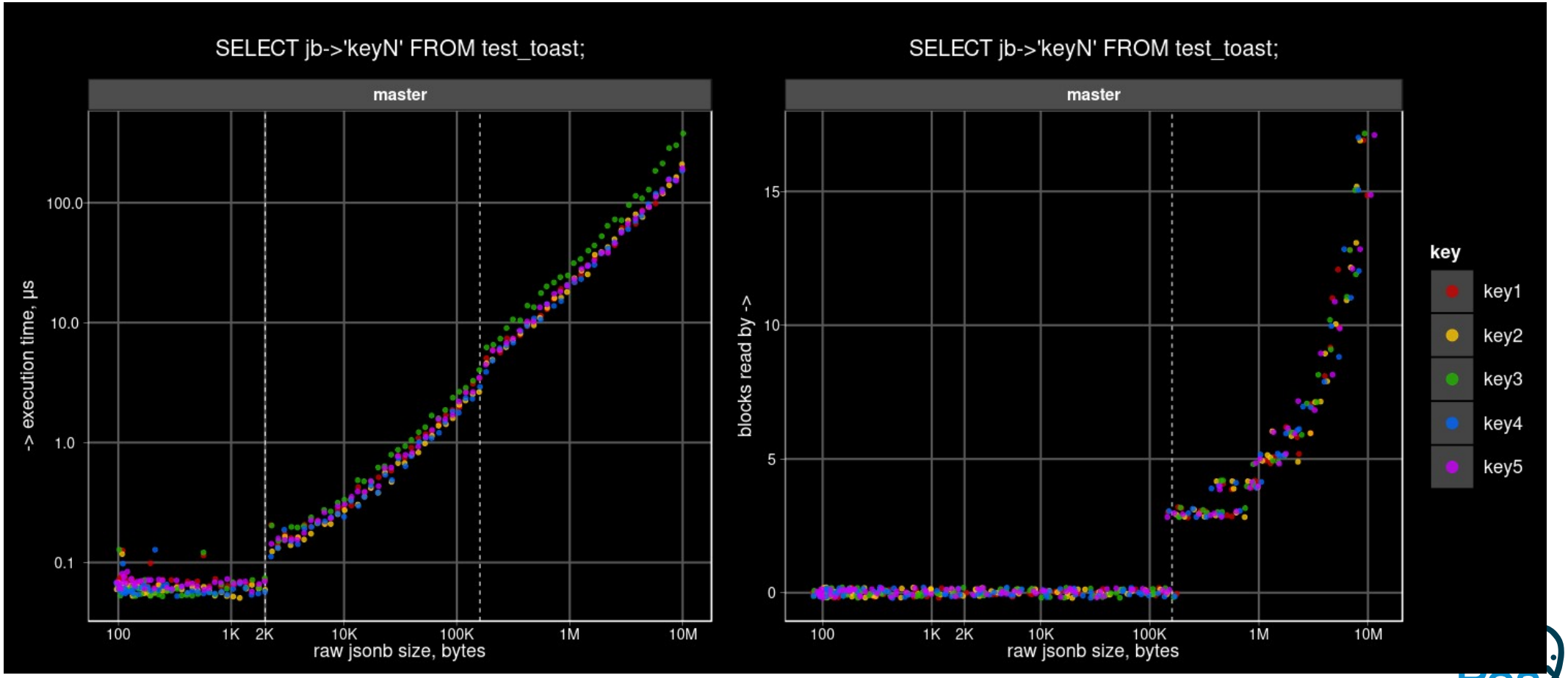
- Each jsonb looks like: `key1,key2, loooong key3, key4,key5.`
- We measure execution time of operator `->(jsonb, text)` for each row by repeating it 1000 times in the query:

```
SELECT jb -> 'keyN', jb -> 'keyN', … jb -> 'keyN' FROM test_toast WHERE id = ?;
```

PostgresPro

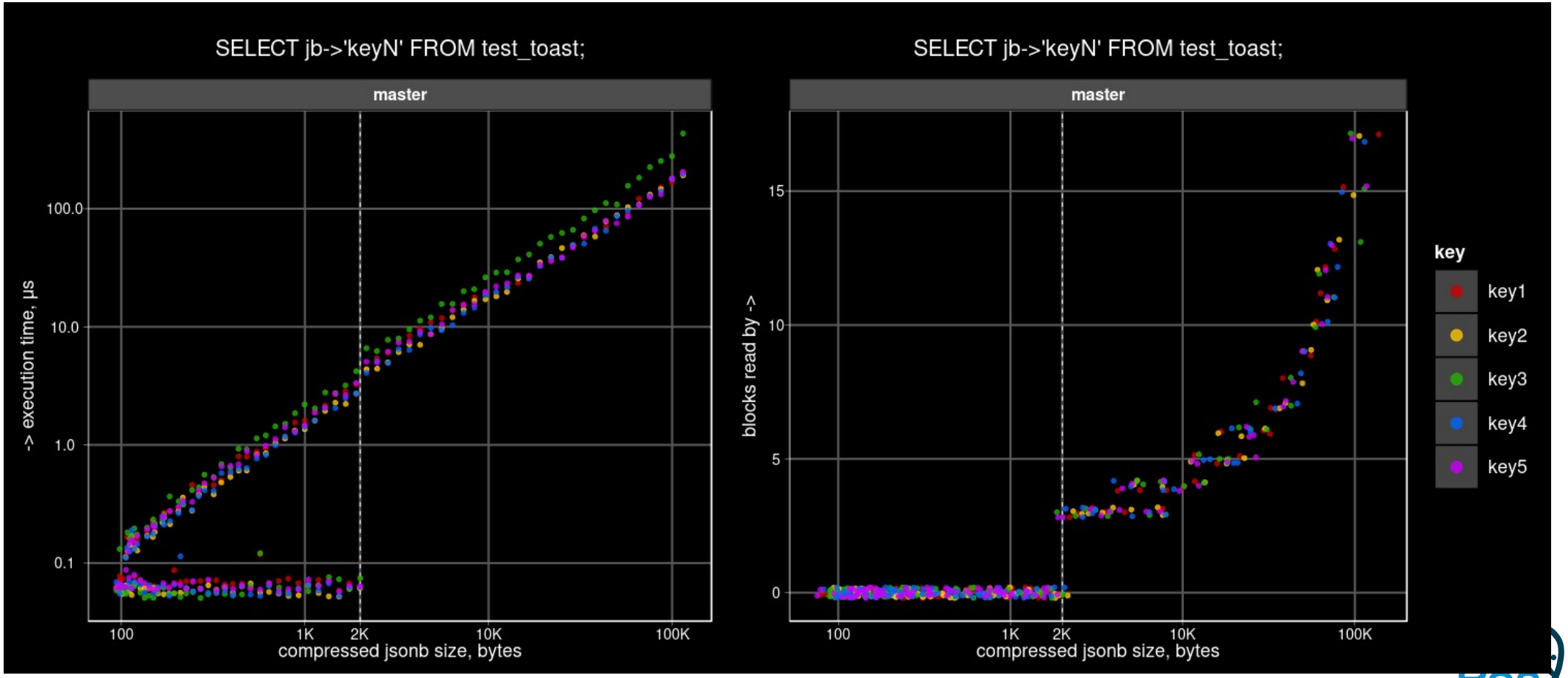
Motivational example (synthetic test)

Key access time for TOASTed jsonbs linearly increase with jsonb size, regardless of key size and position.



TOAST performance problems (synthetic test)

The test shows that key access time for TOASTed jsonbs is linearly increased with jsonb size, regardless of key size.



PostgresPro

Motivational example (IMDB test)

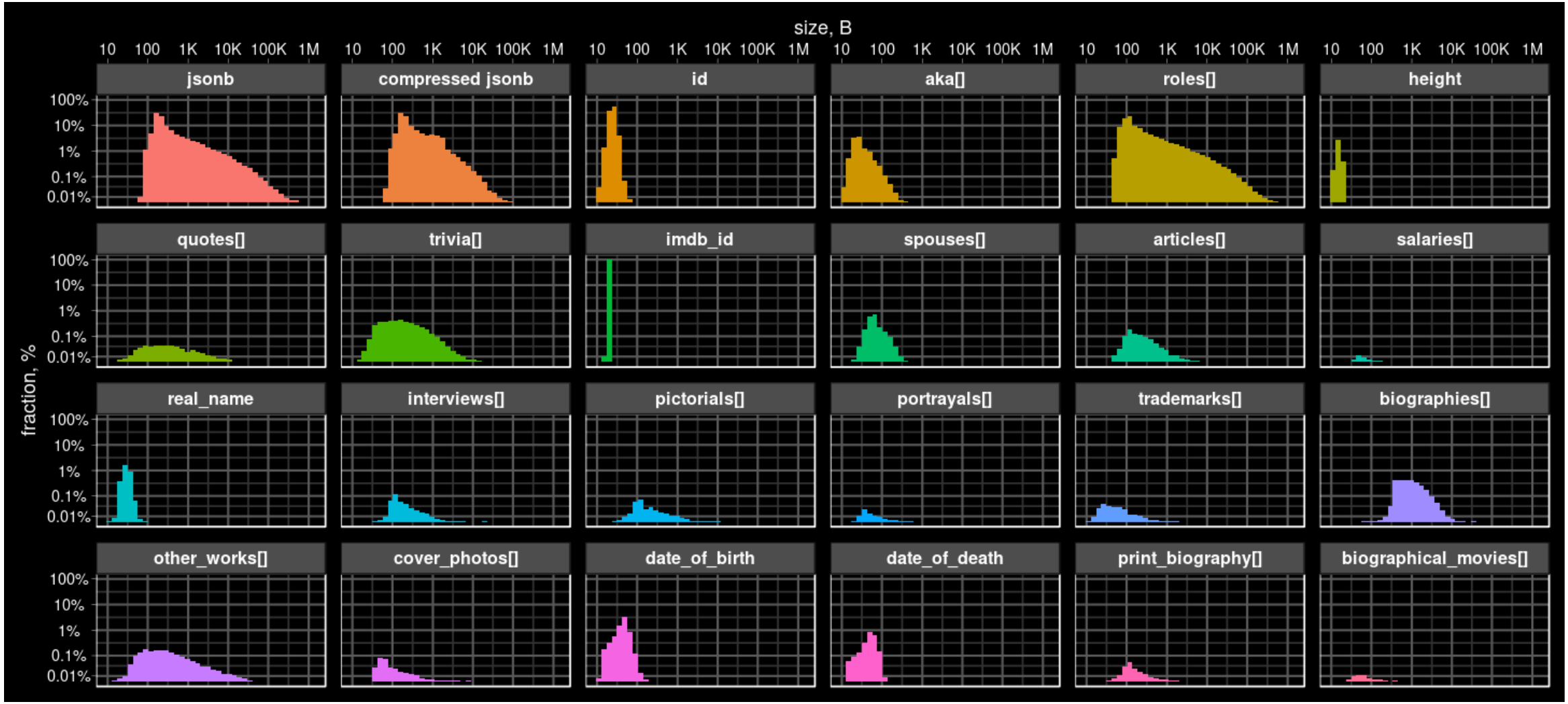
- Real-world JSON data extracted from IMDB database (imdb-22-04-2018-json.dump.gz)
- Typical IMDB «name» document looks like:

```
{
    "id": "Connors, Steve (V)",
    "roles": [
        {
            "role": "actor",
            "title": "Copperhead Creek (????)"
        },
        {
            "role": "actor",
            "title": "Ride the Wanted Trail (????)"+
        }
     ],
     "imdb_id": 1234567
 }
```

- There are many other rare fields, but only `id`, `imdb_id` are mandatory, and `roles` array is the biggest and most frequent (see next slide).

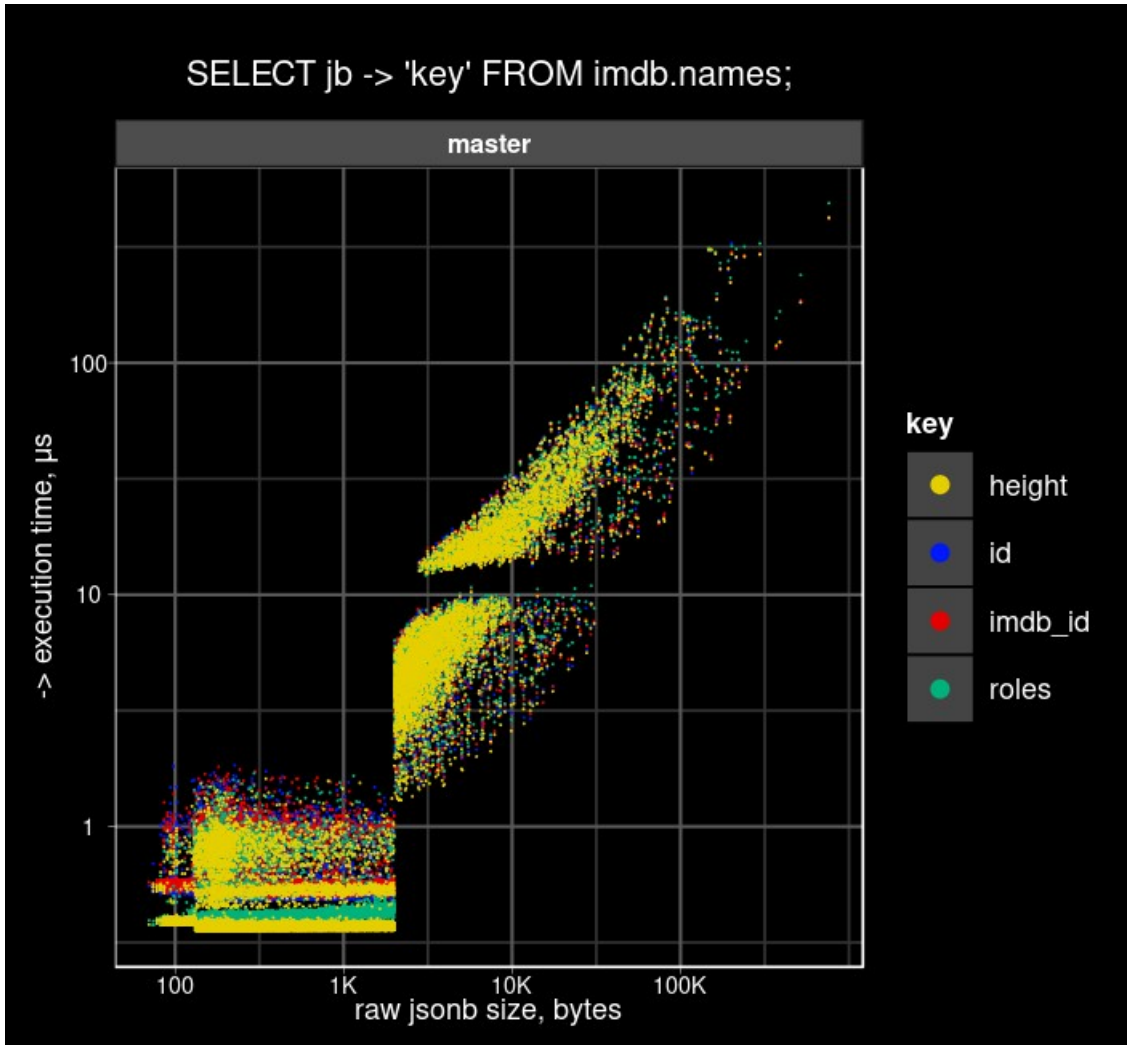
PostgresPro

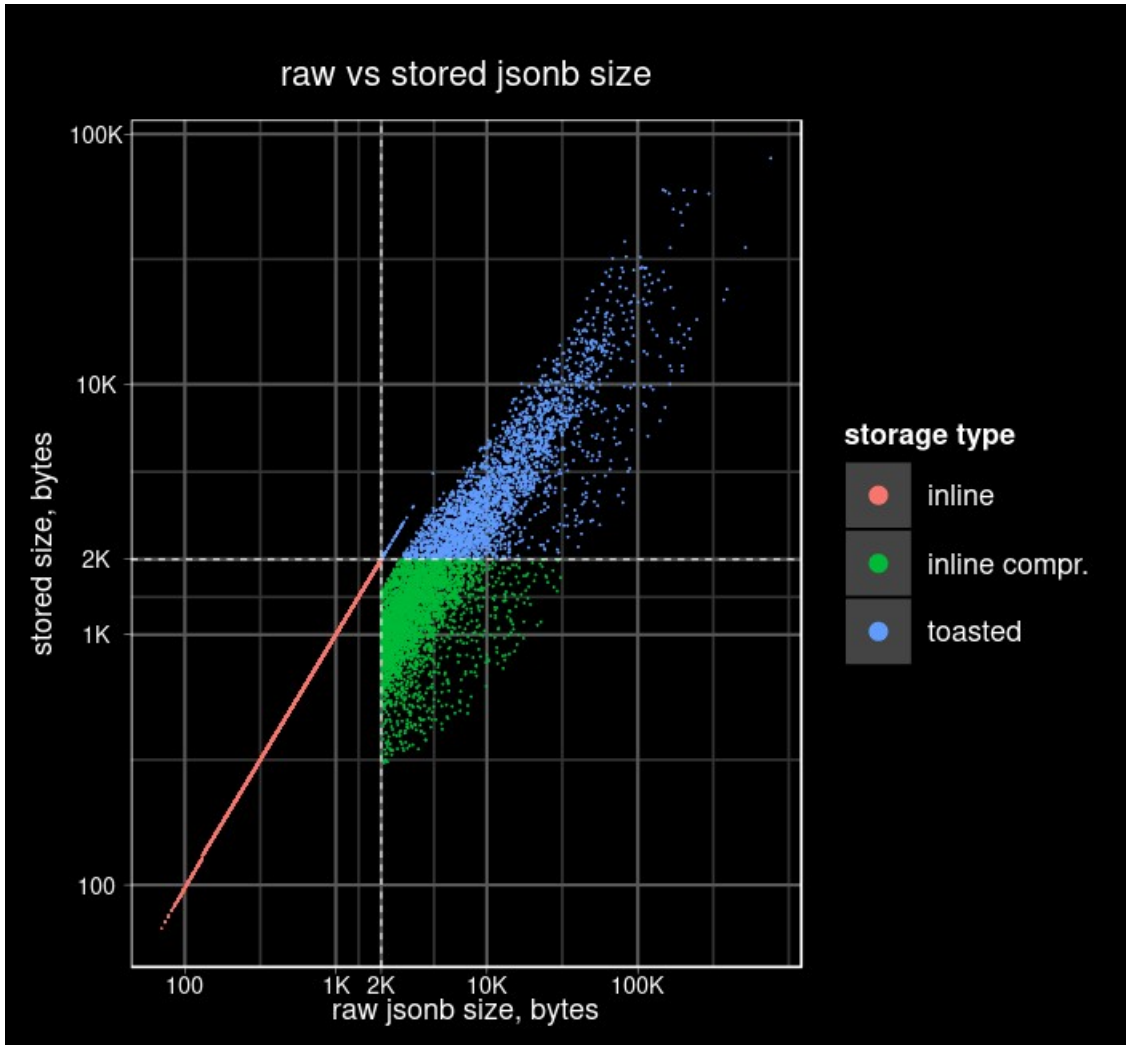
IMDB data set field statistics



PostgresPro

Motivational example (IMDB test)





PostgresPro

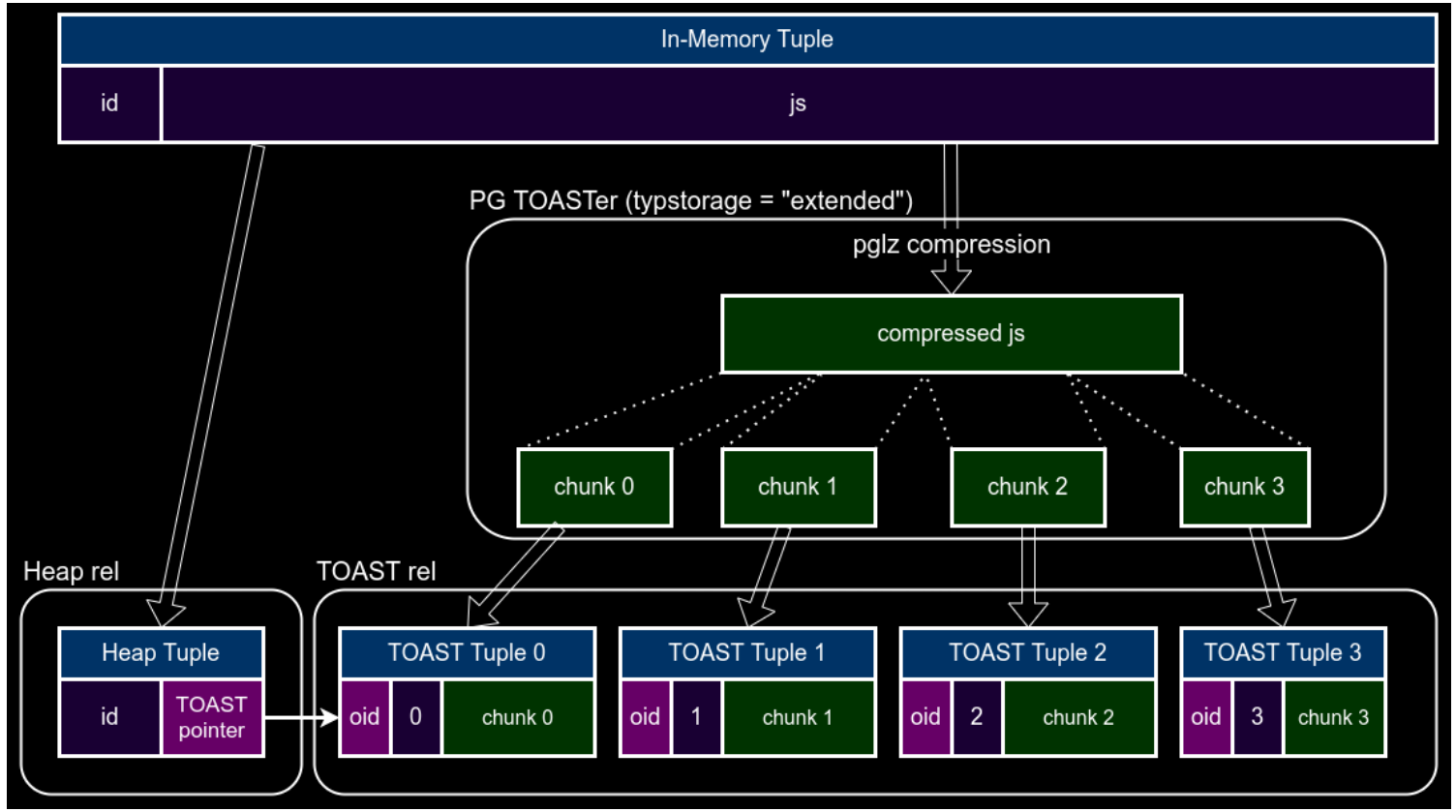
Motivation

- Decompression is the biggest problem. Big overhead of decompression of the whole jsonb limits the applicability of jsonb as document storage with partial access.
  - Need partial decompression
- Toast introduces additional overhead - read too many block
  - Read only needed blocks — partial detoast

PostgresPro

TOAST process and its internal structure

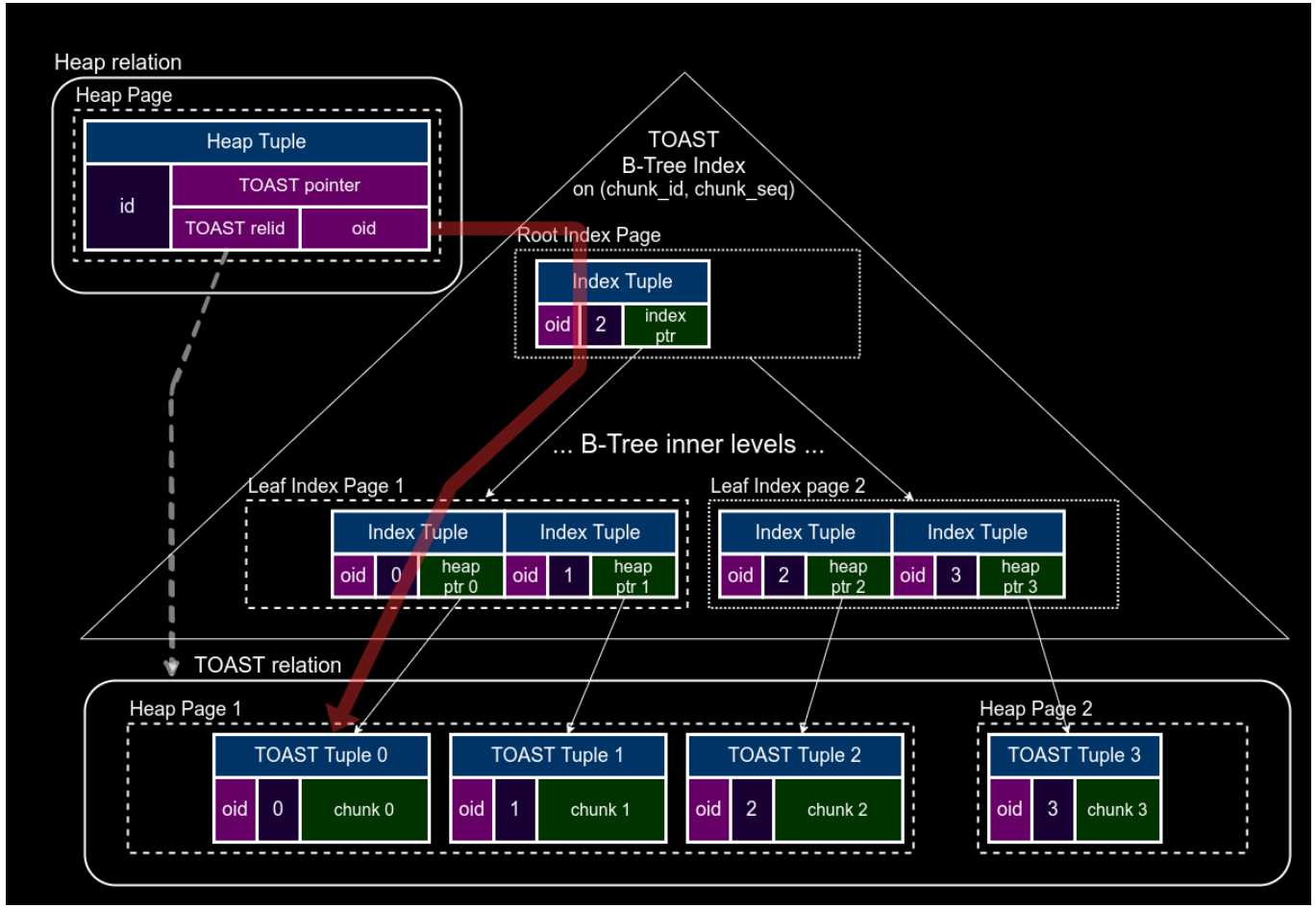
- TOASTed value is pglz compressed
- Compressed value is split into fixed-size TOAST chunks (1996B for 8KB page)
- TOAST chunks are augment with generated Oid `chunk_id`, sequnce number `chunk_seq` and written as tuples into special TOAST relation `pg_toast.pg_toast_XXX`, created for each table containing TOASTable attributes
- Attribute in the original heap tuple is replaced with TOAST pointer (18 bytes) containing `chunk_id, toast_relid, raw_size, compressed_size`



PostgresPro

TOAST access

TOAST pointers does not refer to heap tuples with chunks directly. Instead they contains Oid chunk_id and we need to descent by index (chunk_oid, chunk_seq).



So, to read only a few bytes from the first chunk we need to read 3, 4 or even 5 additional blocks.

PostgresPro

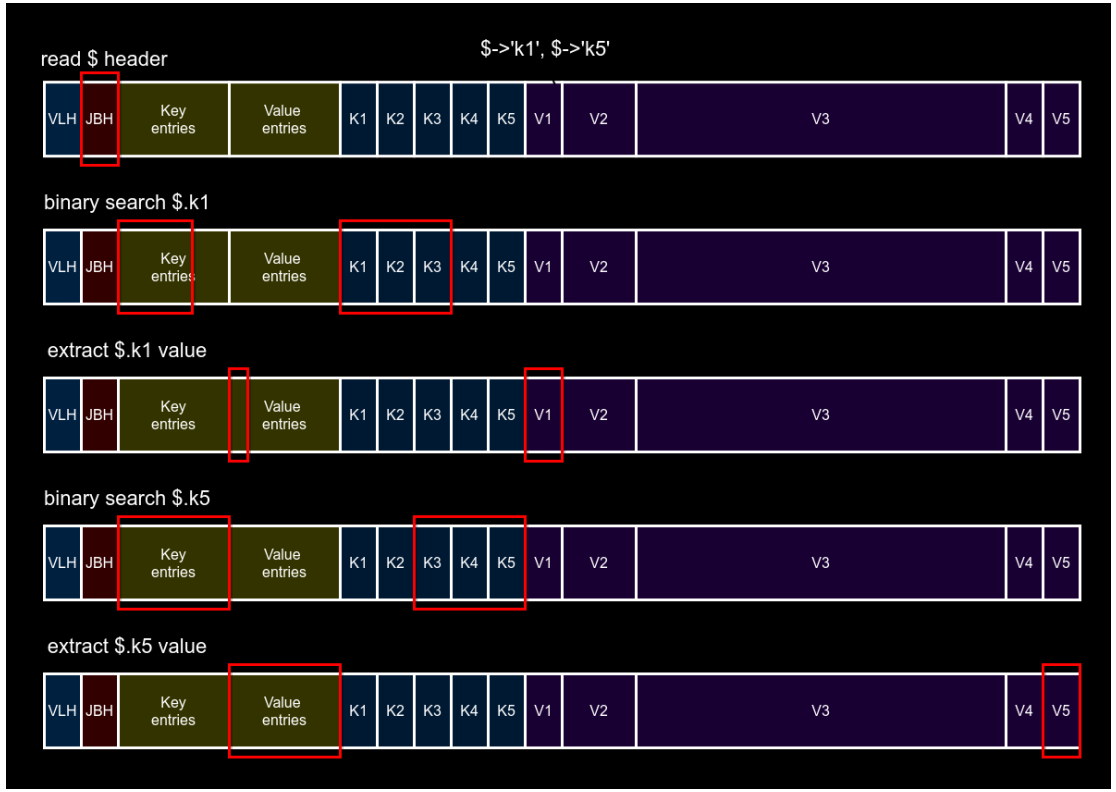
Jsonb deTOAST improvements

- Partial pglz decompression
- Sort jsonb object key by their length
- Partial deTOASTing using TOAST iterators
- Inline TOAST
- Random access TOAST
- Partial compression (???)

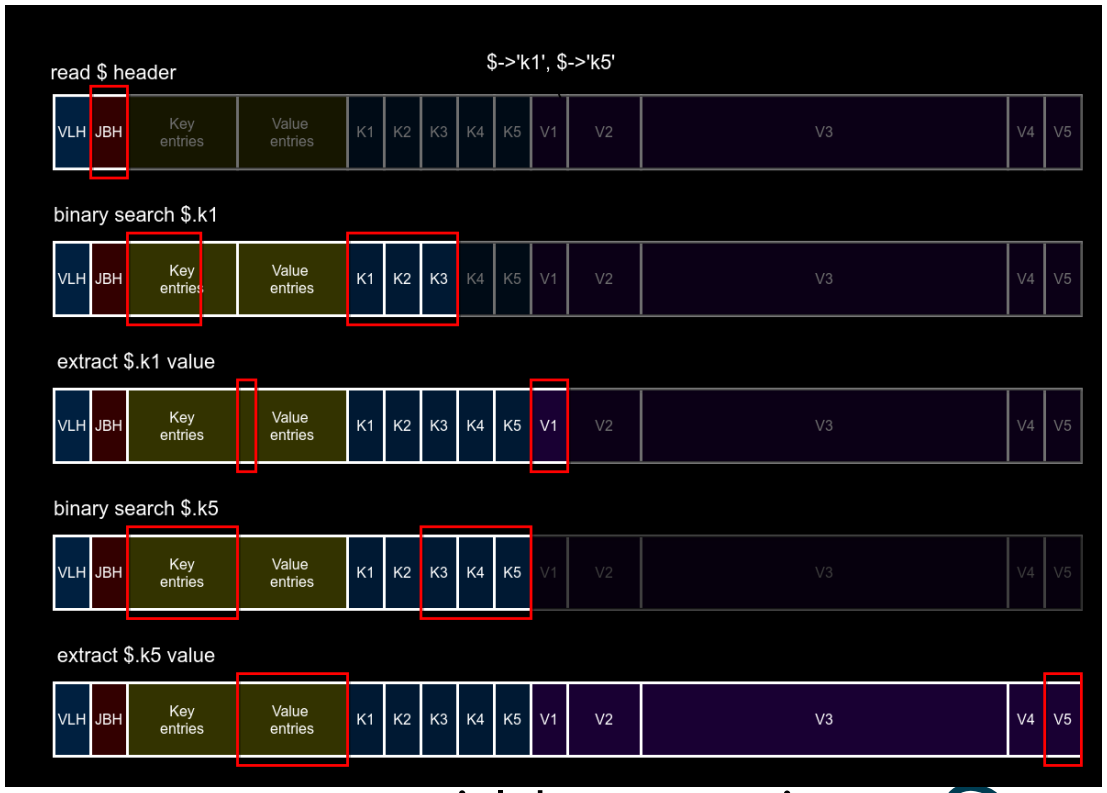
PostgresPro

Jsonb partial decompression

- Partial decompression eliminates overhead of pglz decompression of the whole jsonb.
- Jsonb is decompressed step by step: header, KV entries array, key name and key value.
  Only prefix of jsonb has to be decompressed to acccess a given key !



full decompression



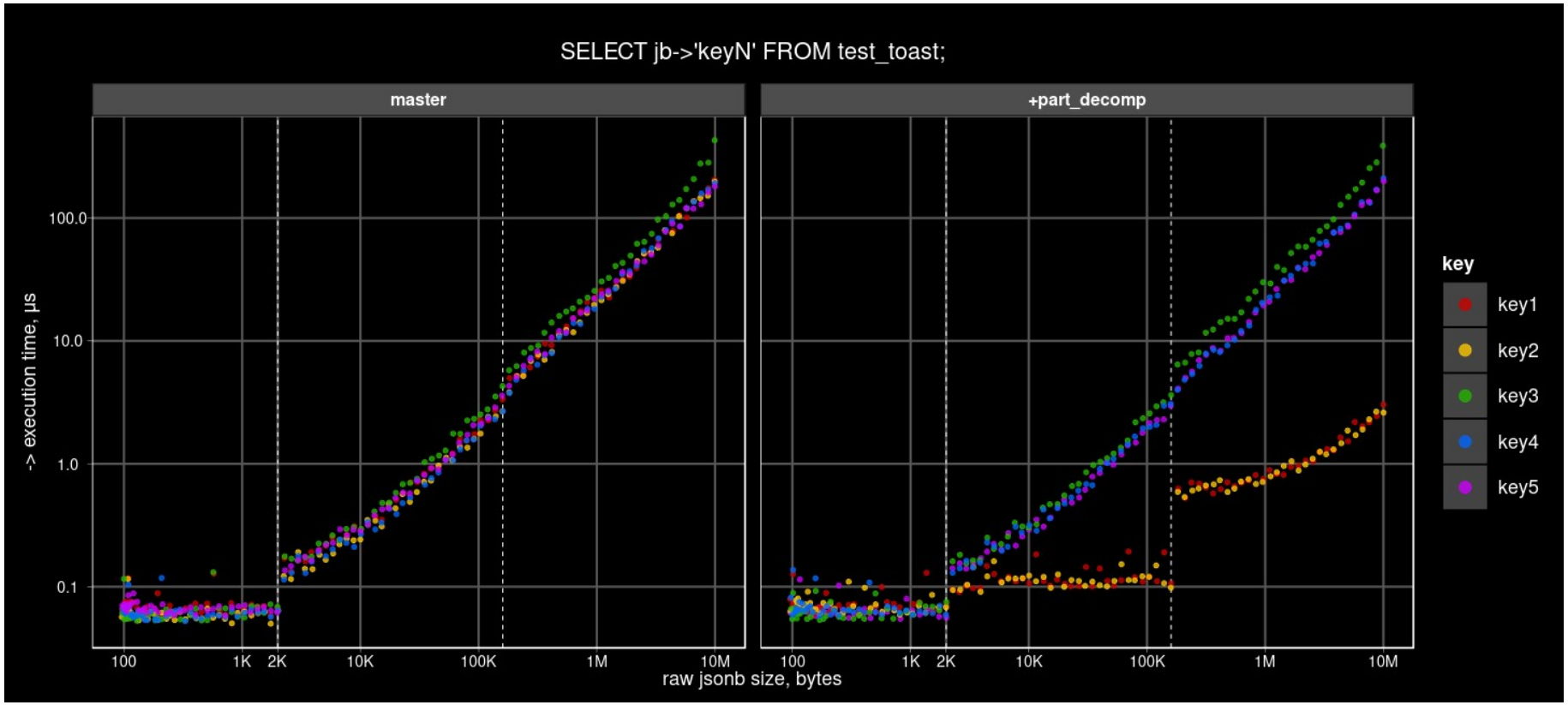
partial decompression

PostgresPro

Jsonb partial decompression results (synthetic)

Access to key1 and key2 (at the beginning of jsonb) was significantly speed up:

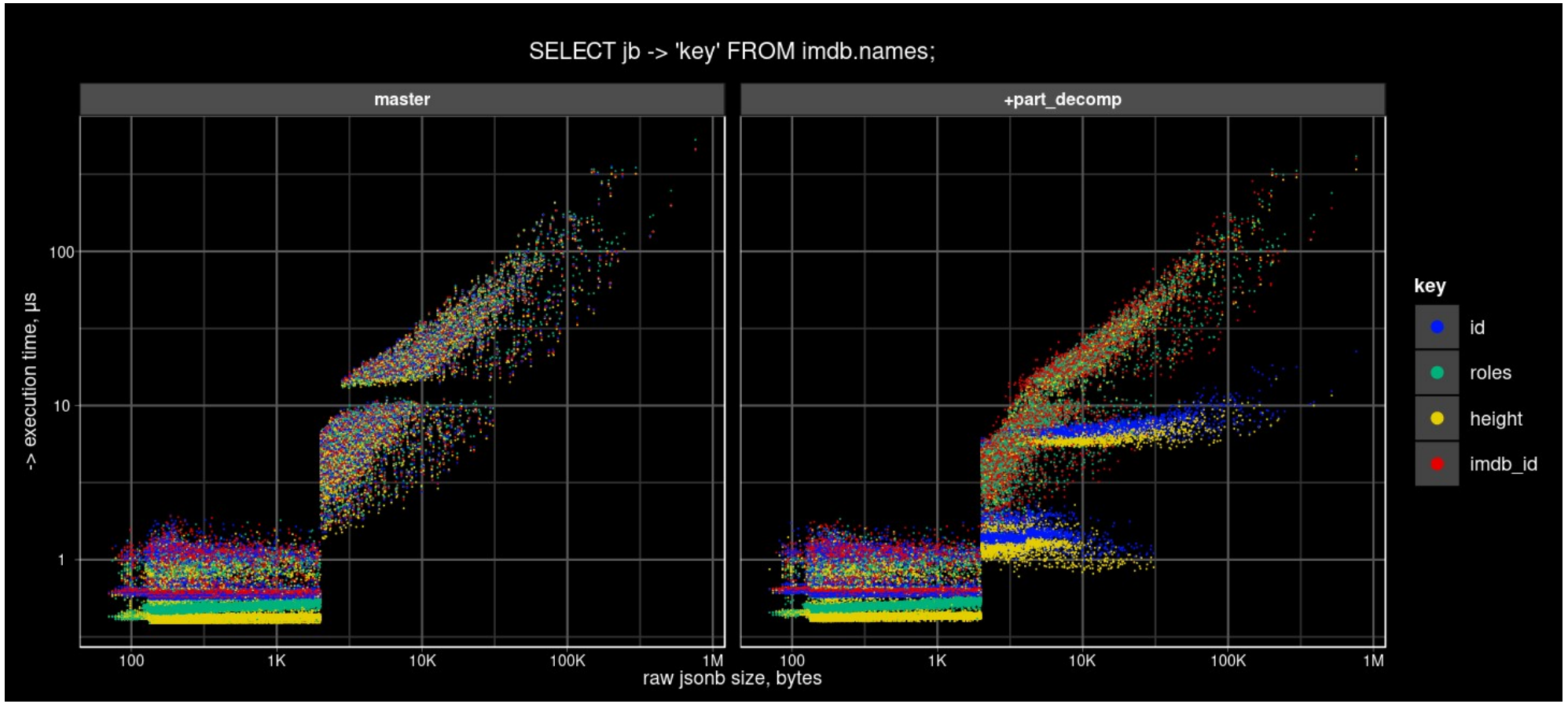
- For inline compressed jsonb access time becomes constant
- For jsonb > 1MB acceleration is of order(s) of magnitude.



PostgresPro

Jsonb partial decompression results (IMDB)

- Access to the first key «id» and rare key «height» was speed up.
- Access time to big key «roles» and short «imdb_id» placed at the end after «imdb_id» is mostly unchanged



PostgresPro

Sorting jsonb keys by length

In the original jsonb format object keys are sorted by (length,name), so the short keys with longer or alphabetically greater names are placed at the end and cannot benefit from the partial decompression. Sorting by length allows fast decompressions of the shortest keys (metadata).

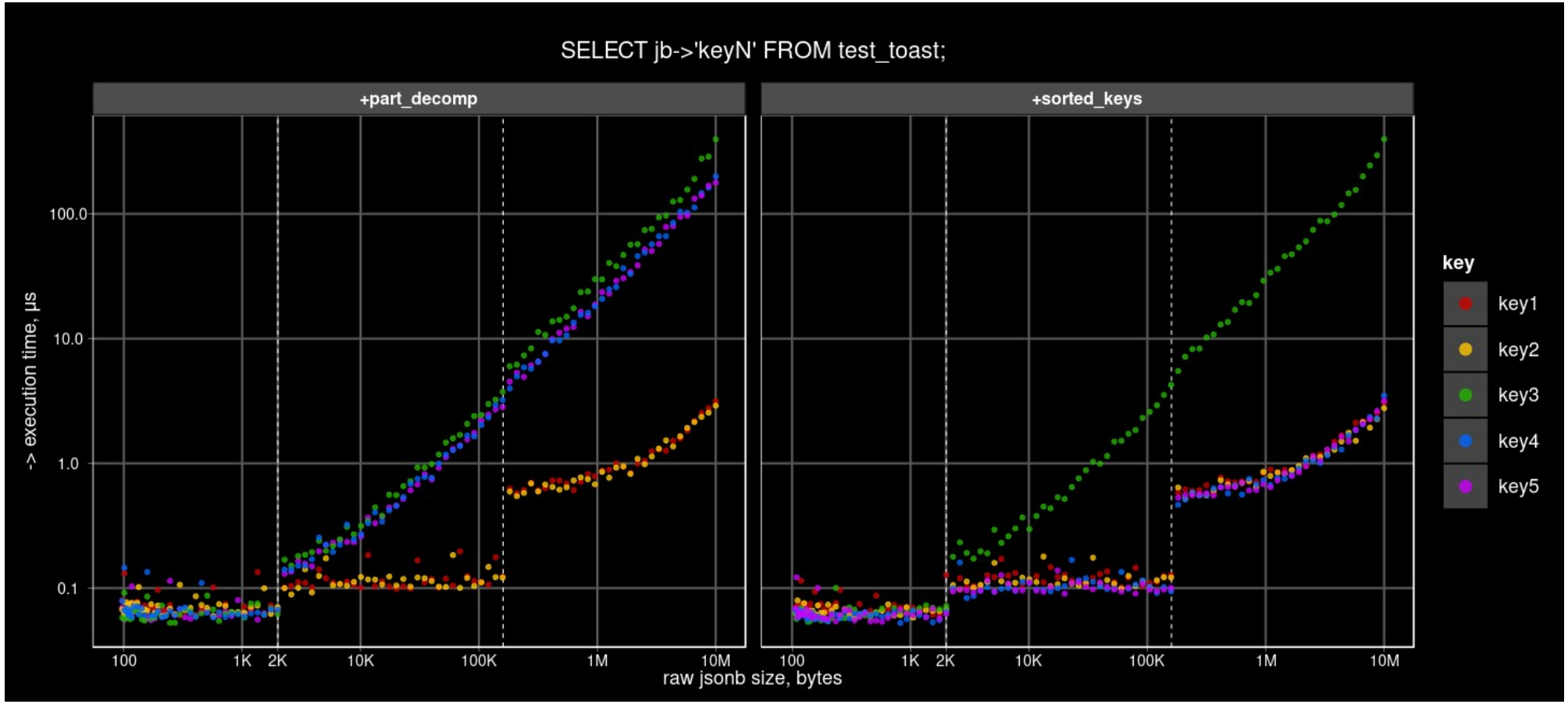
original: keys names and values sorted by key names

new: keys values sorted by their length

PostgresPro

Sorting jsonb keys by length results (synthetic)

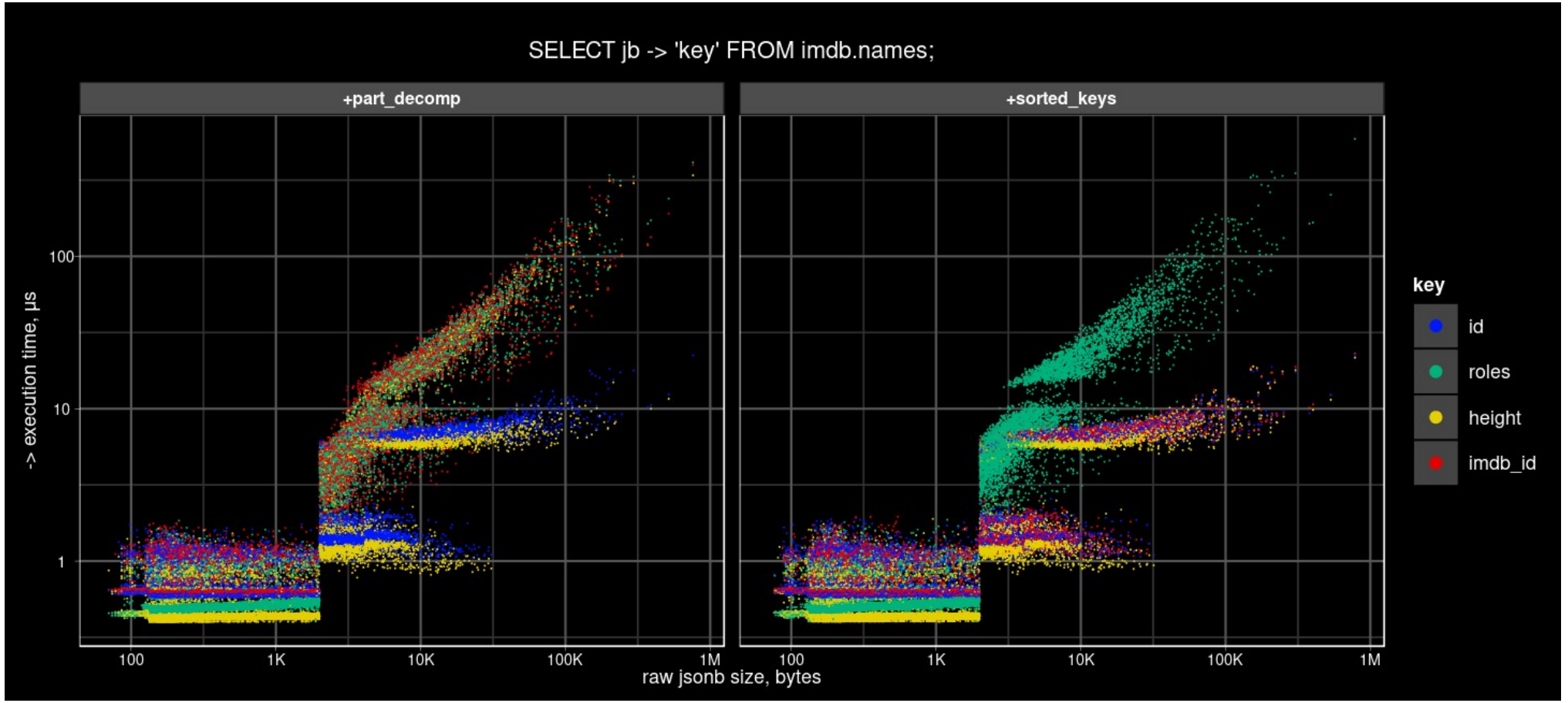
Access to the all short keys (excluding long key3, placed now at the end of jsonb) was significantly speed up:



PostgresPro

Sorting jsonb keys by length results (IMDB)

- Access to the last short key «imdb_id» now also was speed up.
- There is big difference in access time (~5x) between inline and TOASTed values.



PostgresPro

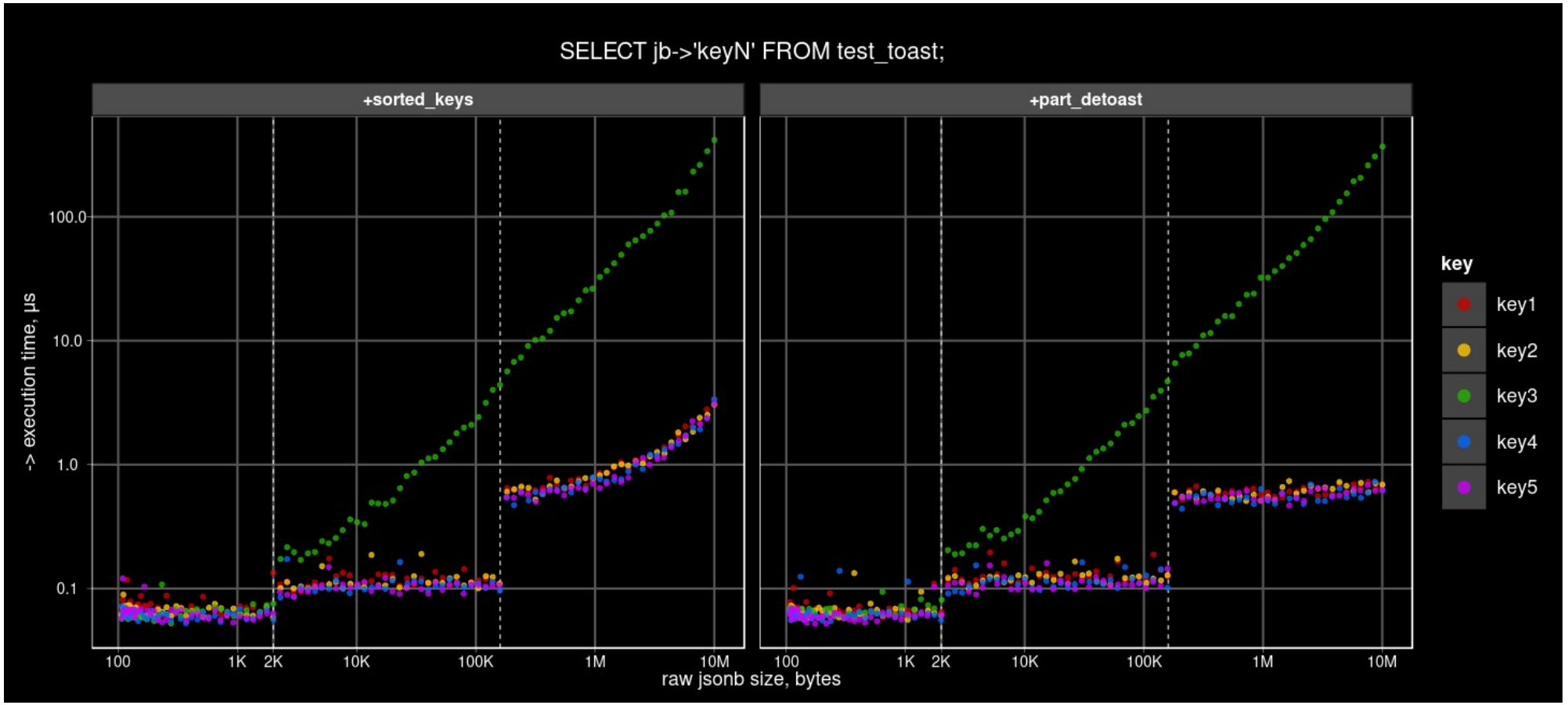
Partial deTOASTing

- We used patch «de-TOAST'ing using a iterator» from the CommitFest. It was originally developed by Binguo Bao at GSOC 2019.
- This patch gives ability to deTOAST and decompress chunk by chunk. So if we need only the jsonb header and first keys from the first chunk, only that first chunk will be read (really, some index blocks also will be read).
- We modified patch adding ability do decompress only the needed prefix of TOAST chunks.

PostgresPro

Partial deTOASTing results (synthetic)

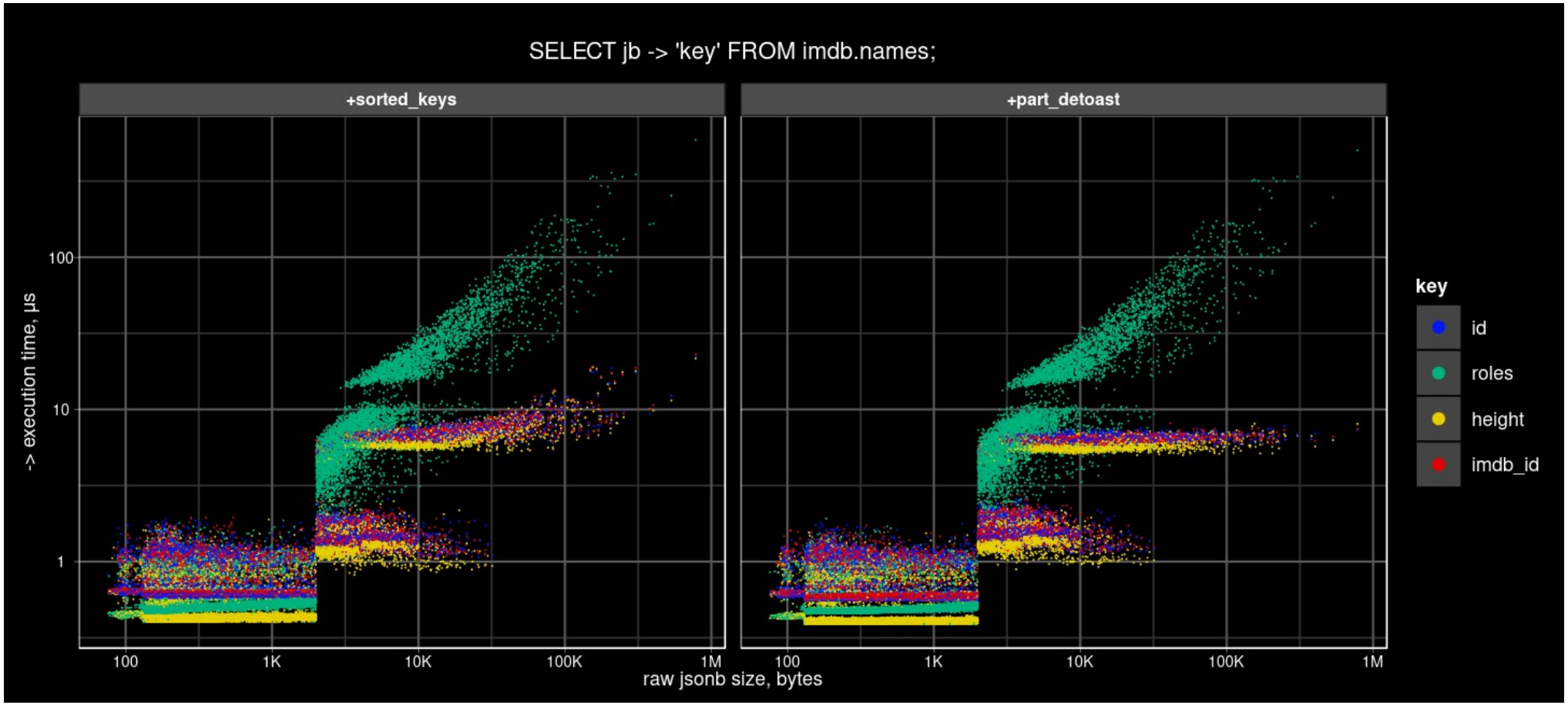
Partial deTOASTing speeds up only access to short keys of long jsonbs, making access time almost independent of jsonb size.



PostgresPro

Partial deTOASTing results (IMDB)

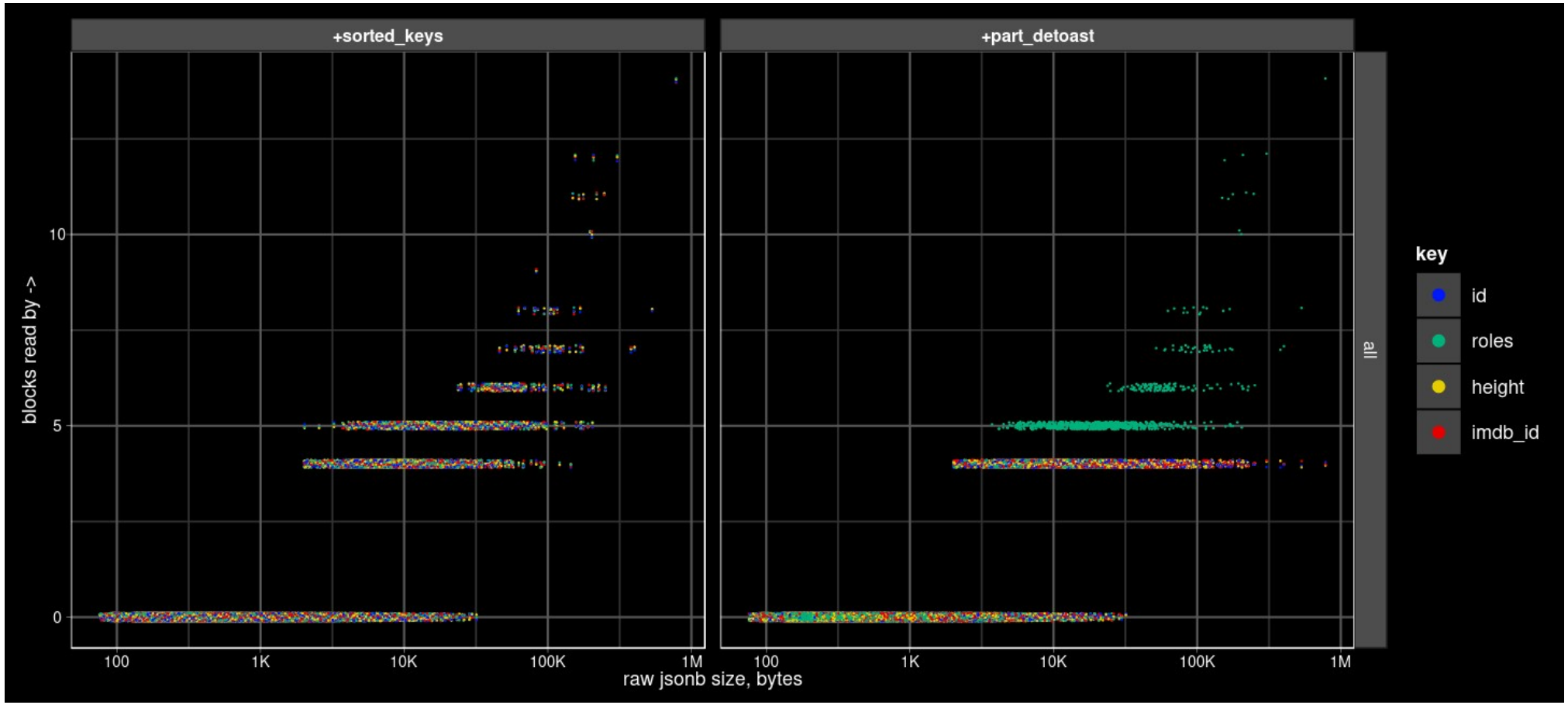
- Results are the same, but not so noticeable because the are not many big (> 100KB) jsonbs.
- A big gap in access time (~5x) between inline and TOASTed values is still here.



PostgresPro

Partial deTOASTing results (IMDB)

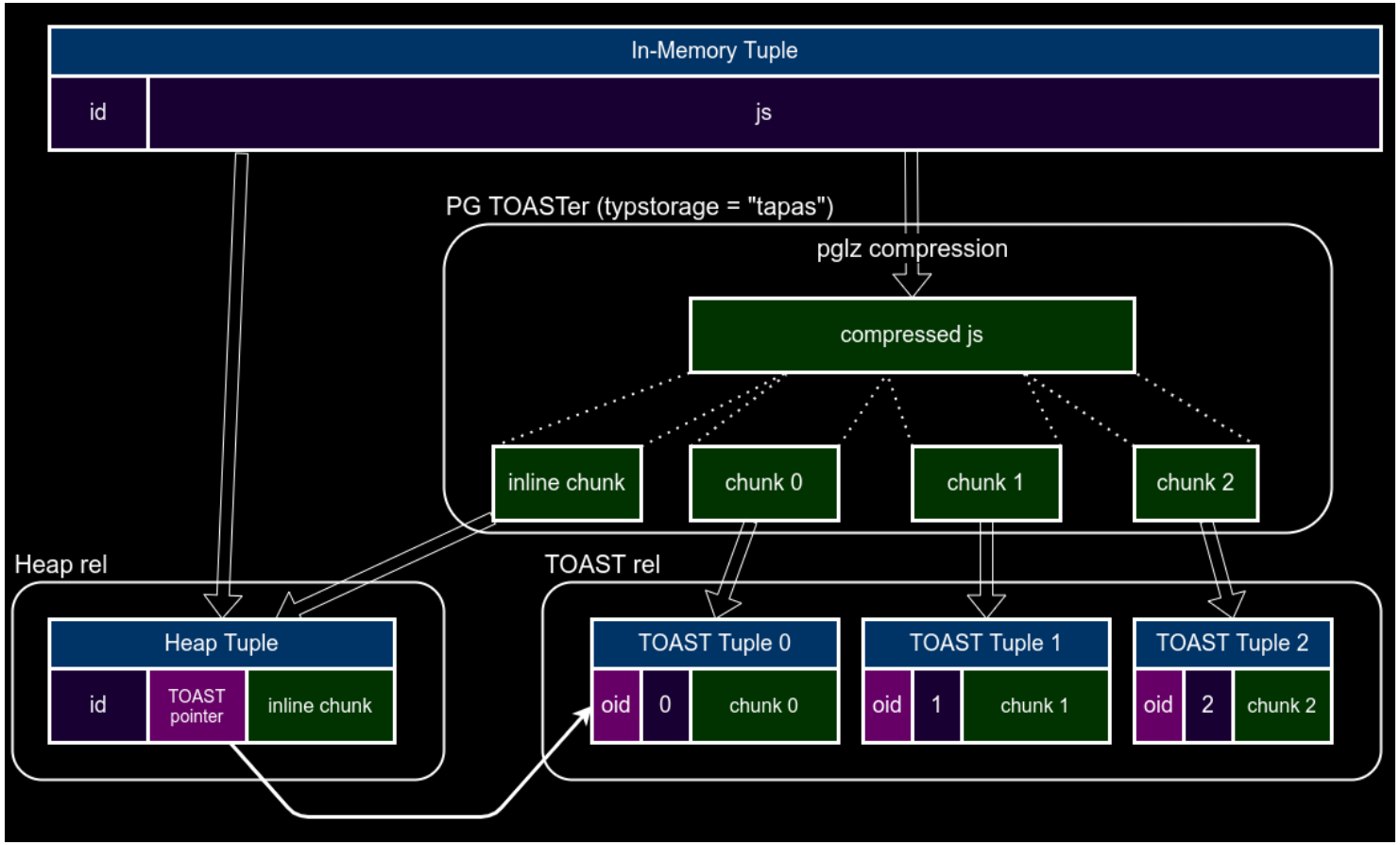
- This graph for blocks read by operator -> shows that after enabling partial deTOASTing during access to short keys always read only 4 blocks (3 index and 1 heap block).



PostgresPro

Iniline TOAST

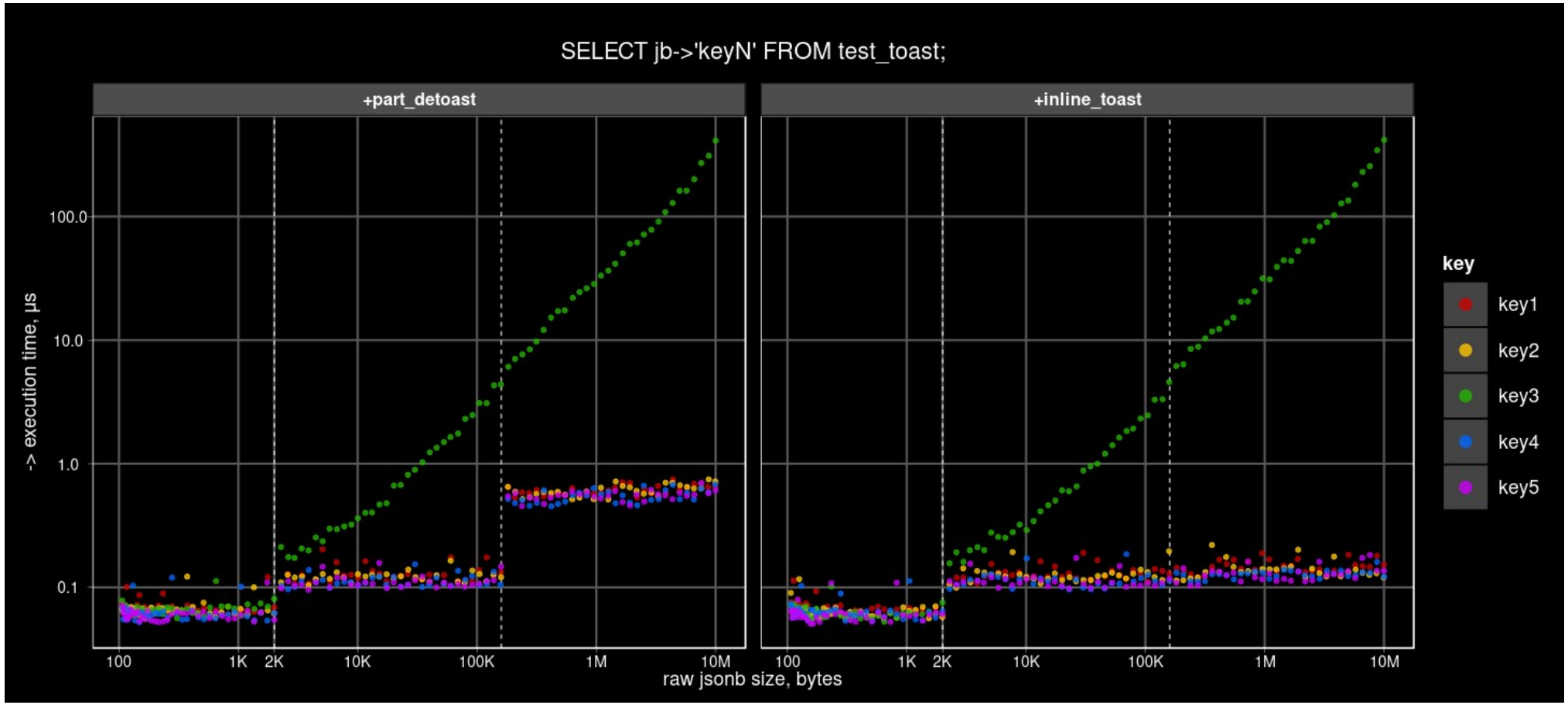
- The idea is to store first TOAST chunk containing jsonb header and possibly some short keys inline in the heap tuple.
- We added new typstorage «tapas» that works similarly to «extended», except that it tries to fill the tuple to 2KB (if other attrubutes occupy less thabn 2KB) with the chunk cut from the beggining of compressed data.



PostgresPro

Inline TOAST results (synthetic)

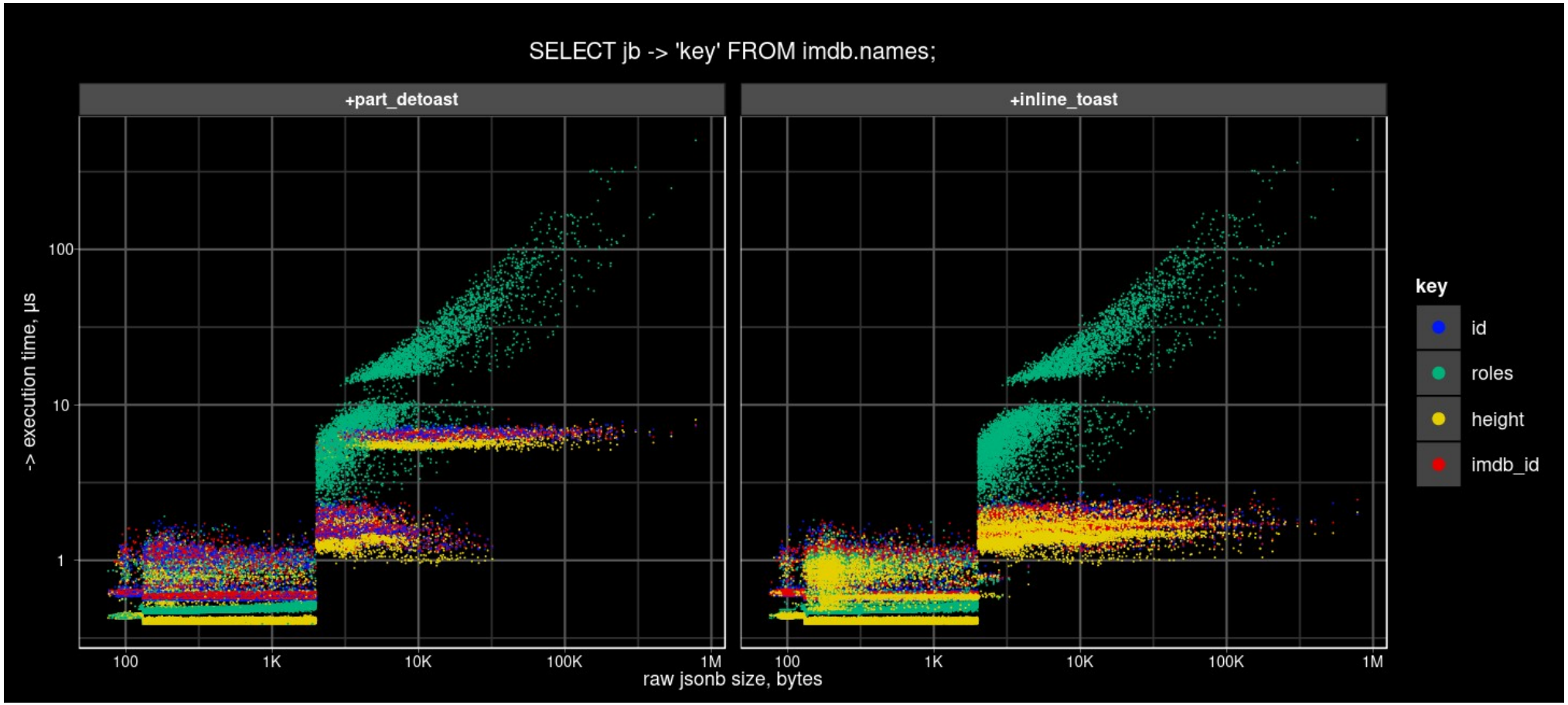
Partial inline TOAST completely removes gap in access time to short keys between long and mid-size jsonbs.



PostgresPro

Inline TOAST results (IMDB)

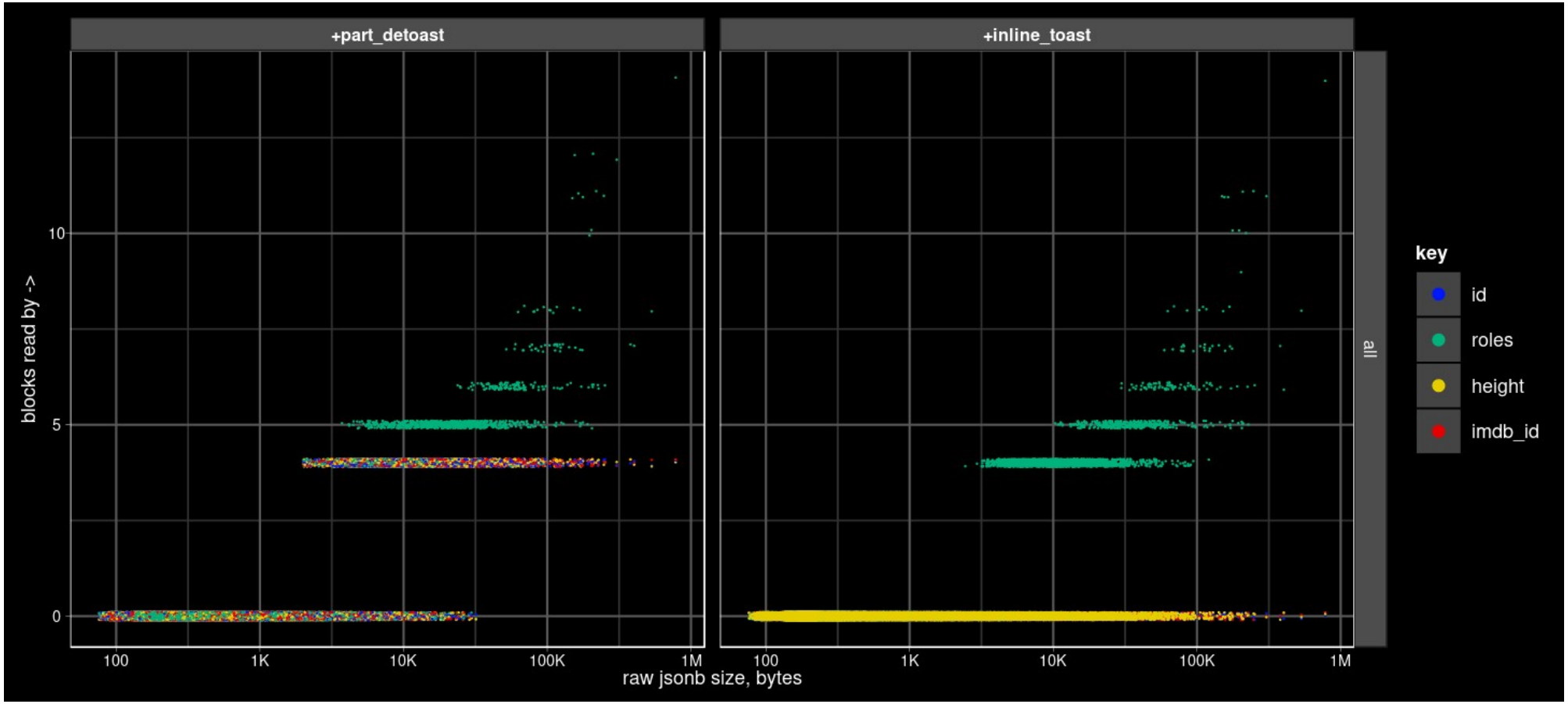
- Results are the same as in synthetic test.
- There is some access time gap between compressed and non-compressed jsonbs.



PostgresPro

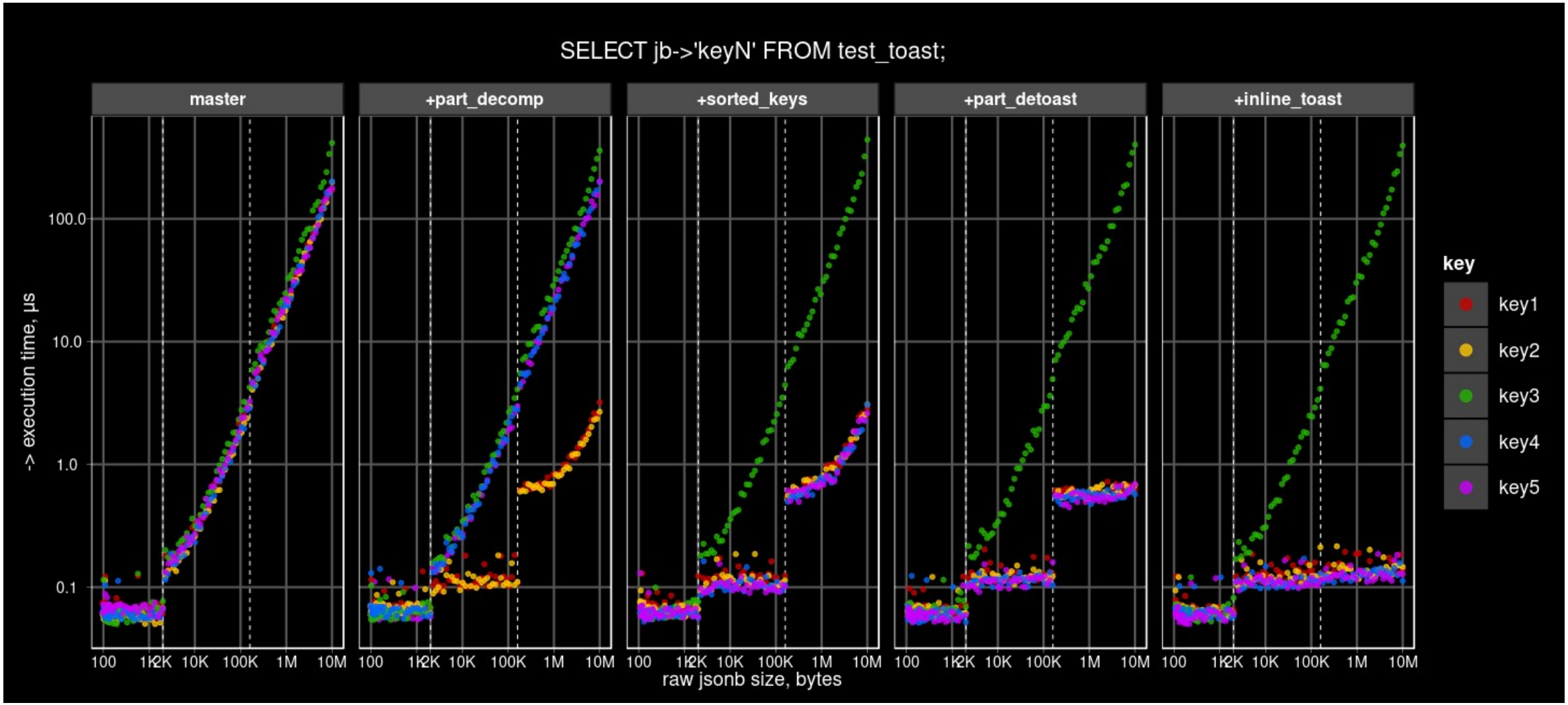
Inline TOAST results (IMDB)

This graph for blocks read by operator -> shows that after enabling inline TOAST during access to short keys always read no additional blocks.



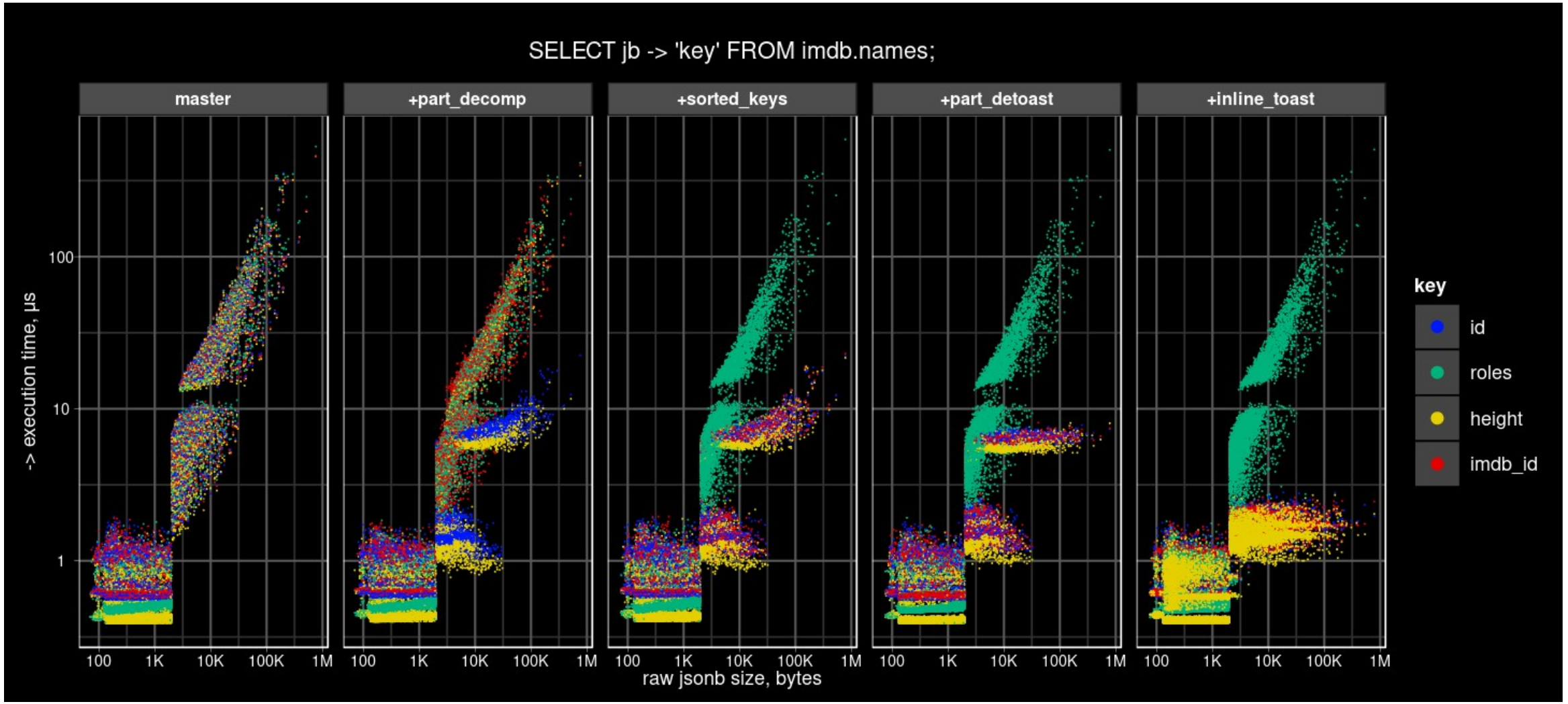
PostgresPro

Step-by-step results (synthetic)



PostgresPro

Step-by-step results (IMDB)



PostgresPro

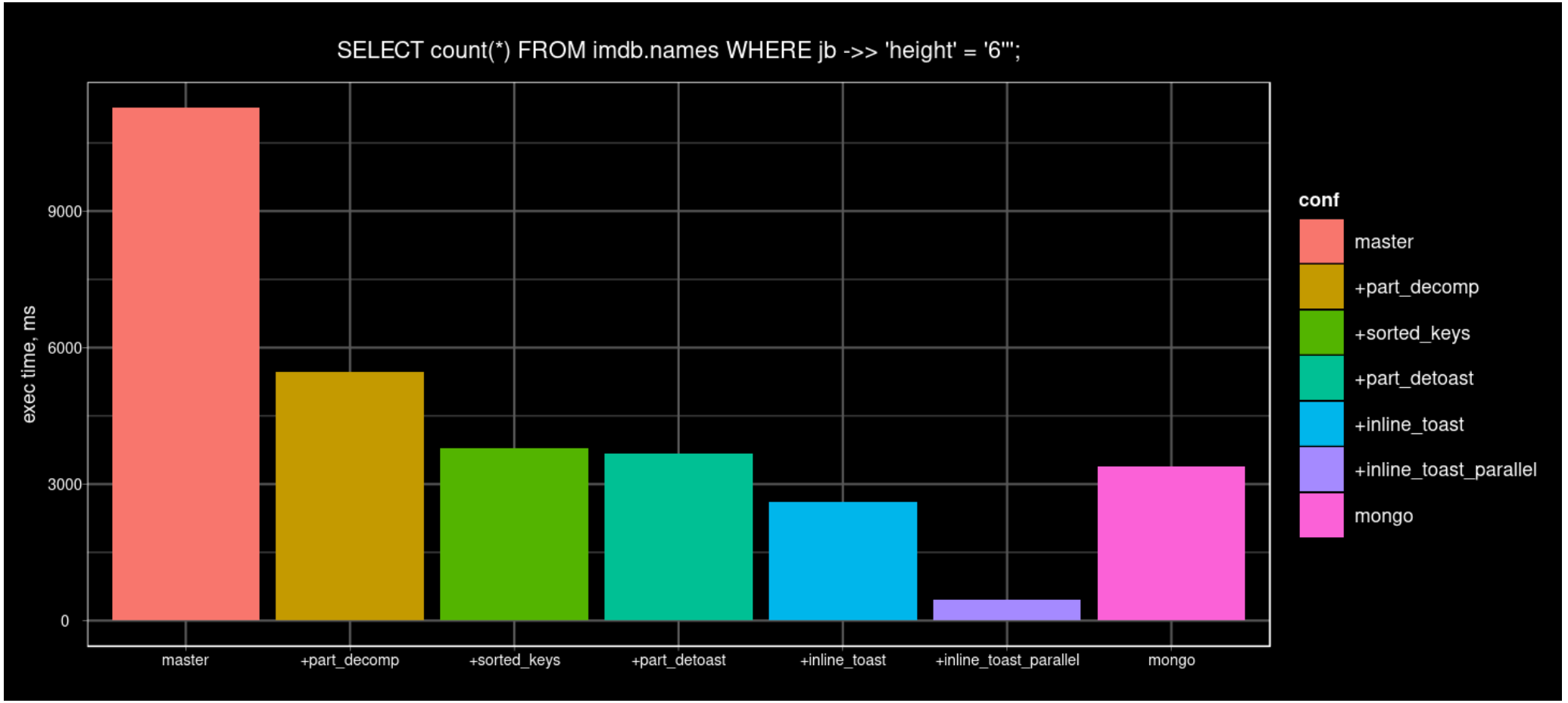
Conclusions

- A series of rather simple and straight-forward algorithms and storage optimizations can greatly speed up access to short keys of jsonb.
  The same technique can be applied to any data types with random access to parts of data (arrays, hstore, movie, pdf …).
- A lot of further work is expected:
  - Random access TOAST - to read only the required TOAST chunks to speed up access to mid-size keys (e.g. if jsonb contains 100 fields of 1KB size)
  - TOAST cache - to avoid duplication of deTOASTing, if the query contains two or more jsonb operators and function on the the same jsonb attribute.
  - DeTOAST deferring in the chain of accessors (js→'roles'→5), not needed for jsonpath.

PostgresPro

Non-scientific comparison PG vs Mongo (4.09)

- Seqscan, everything in memory (shared buffers 16 GB, Mongo — 22 GB)

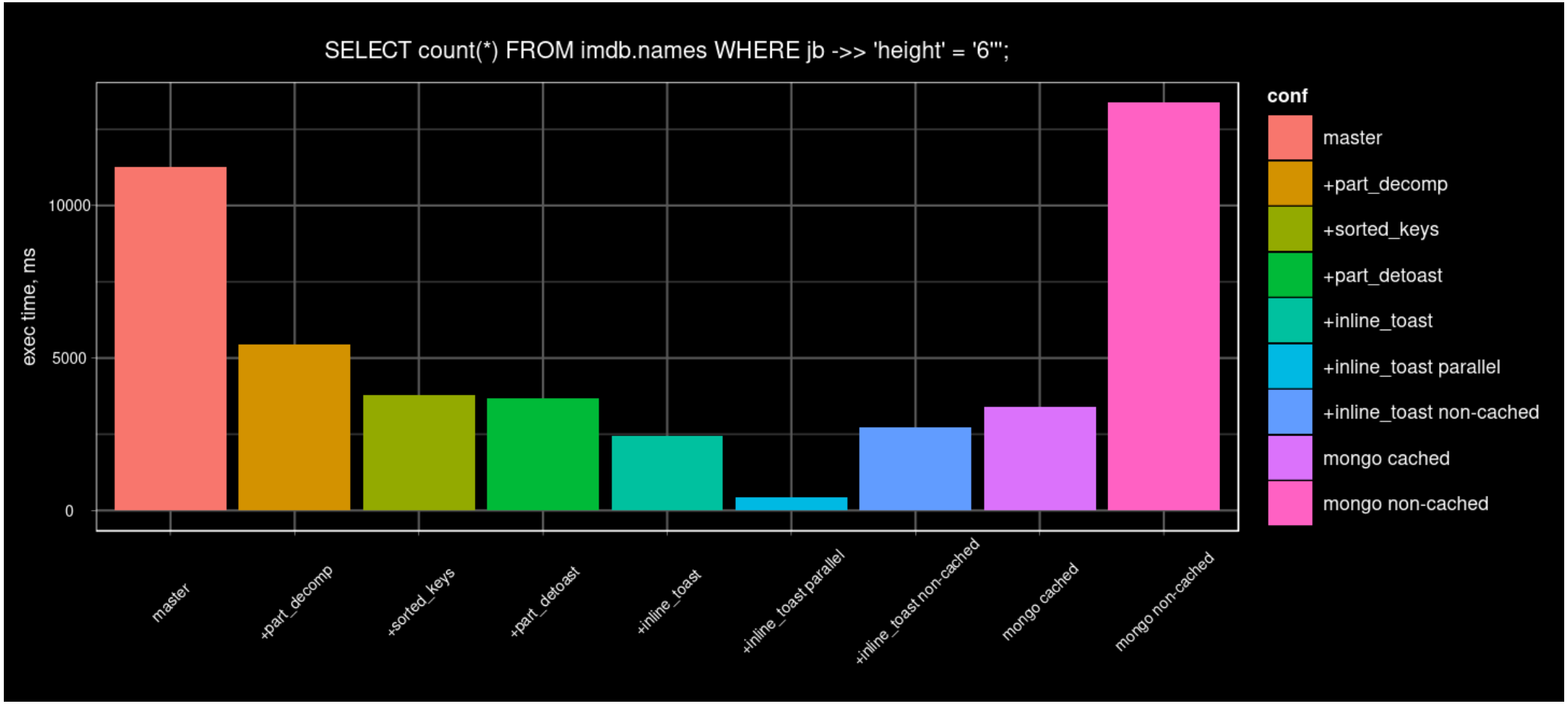


- How many 6-inch tall people are in IMDB database ? (11980)

PostgresPro

Non-scientific comparison PG vs Mongo (4.09)

- Seqscan, non-cached (shared buffers 100 MB, Mongo - 100 MB)



- How many 6-inch tall people are in IMDB database ? (11980)

PostgresPro

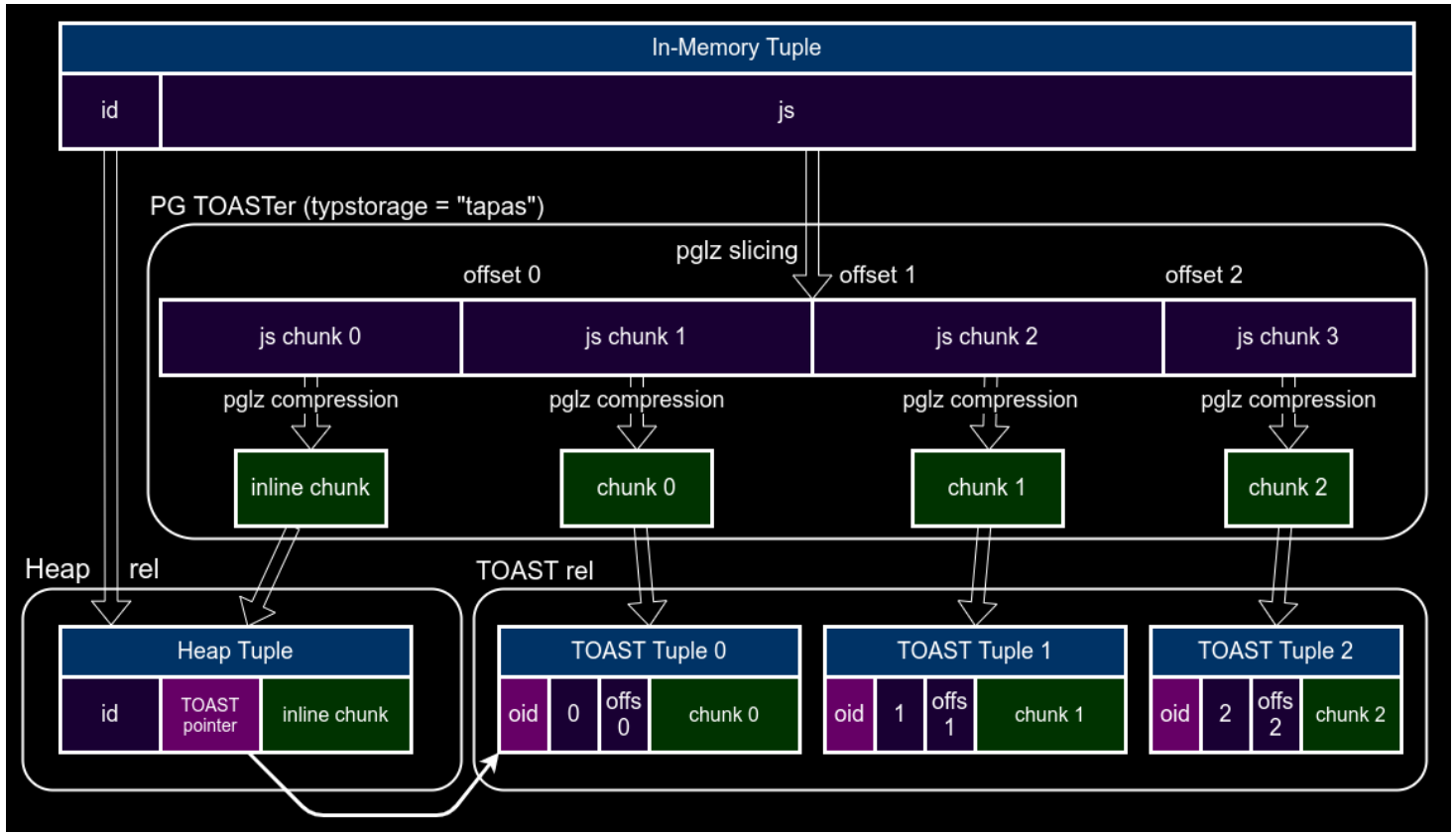
More details and results will be available at PGConf.Online 2021 (March 1-3)
https://www.postgresql.org/about/event/pgconfonline-2021-2401/

You are welcome with your questions !

PostgresPro

Random access TOAST (future)

- Random access TOAST allows to skip reading and decompressing of unneeded TOAST chunks.
- Each chunk should be compressed separately.
- The mapping of jsonb offsets to chunk numbers can be implemented with the additional field chunk_offset
  and the index on (chunk_id, chunk_offset).



PostgresPro


ALL
YOU
NEED
POSTGRES
IS