PostgresPro

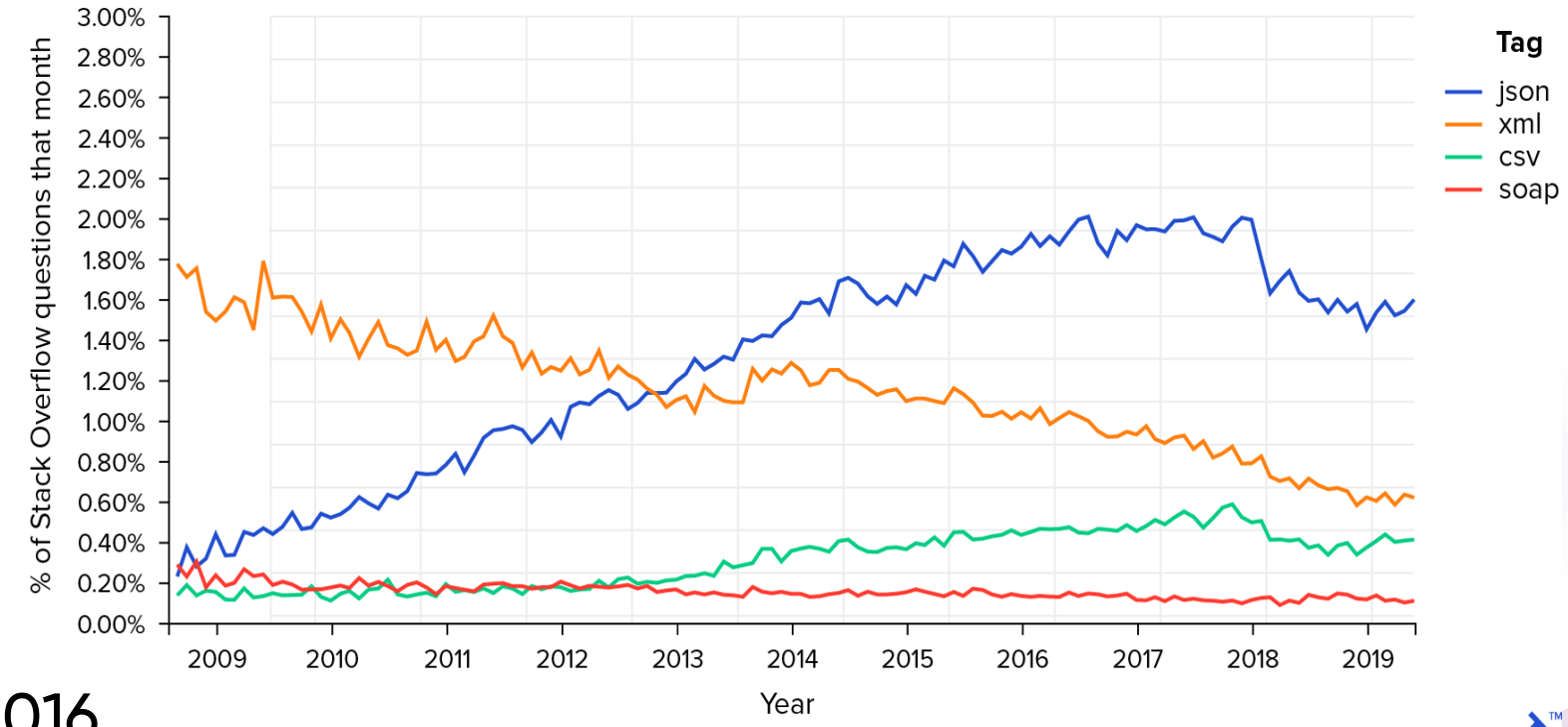
Roasted Toasted JSON

How to TOAST your JSON properly

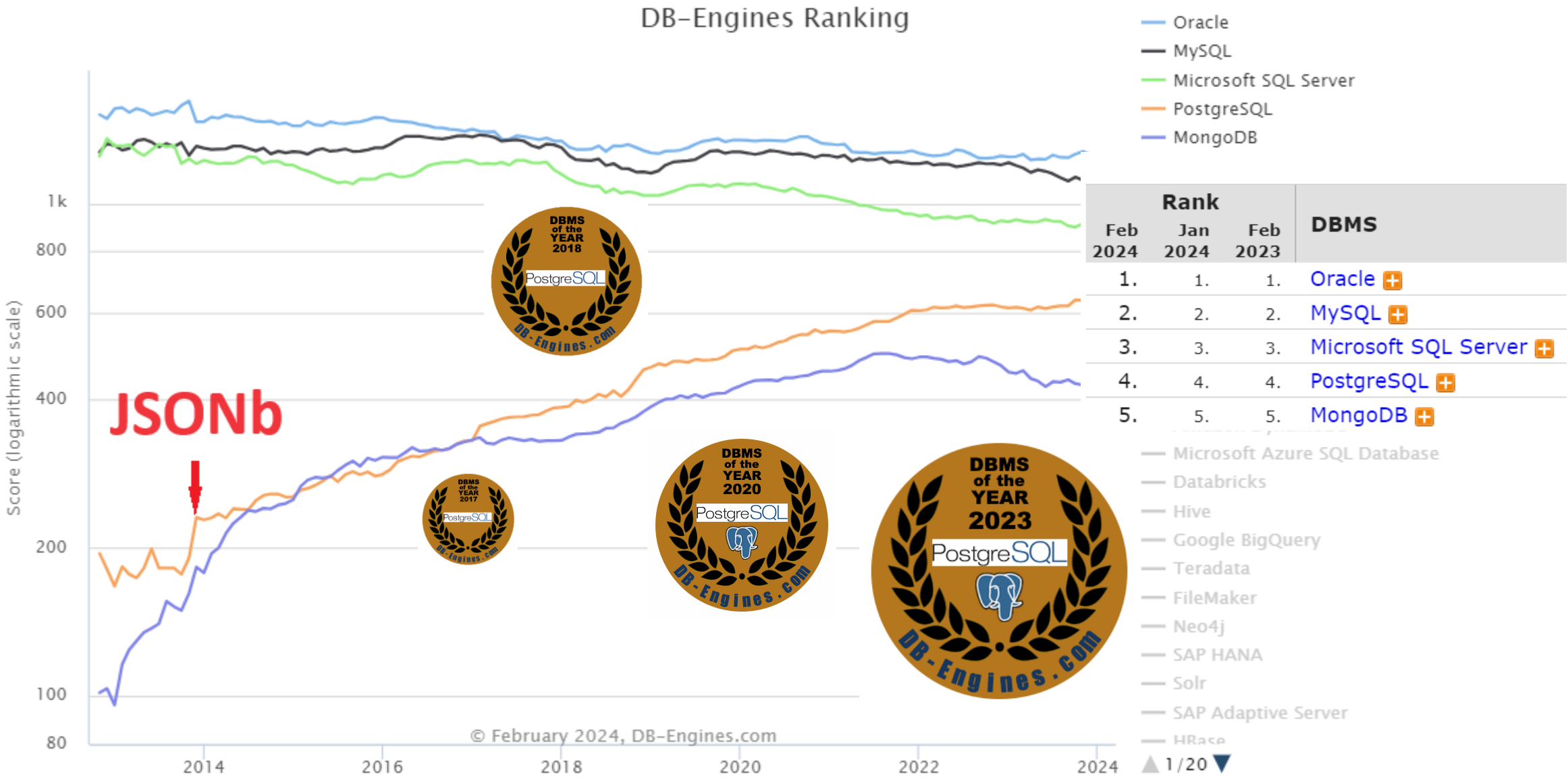
Nikita Malakhov, Senior Software Developer, Postgres Professional, Russia

JSON or Not JSON – It's Not a Question

- 2006 – Informational Specification
- 2014 – RFC 7159, JSON Reference
- 2017 – ISO/IEC JSON International Standard
- 2021 – SQL/JSON support functions in SQL:2016
- 2023 – JSON data type in SQL:2023



The History of PostgreSQL in Short



JSON Support

PostgresPro

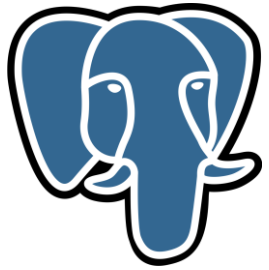
ORACLE

- JSON data type
- OSON binary type
- dot notation
- JSONPath
- JSON Schema
- IS JSON predicate
- SQL/ JSON Support

r23

- JSON binary type
- dot notation
- JSONPath
- Some predicates and schema support through functions
- SQL/ JSON Support

v8.0



- json data type
- jsonb binary type
- jsonpath data type
- JSONPath
- Schema validation support through extension
- IS JSON predicate
- SQL/ JSON Support

Keeping up with v17!

JSON in PostgreSQL

PostgresPro

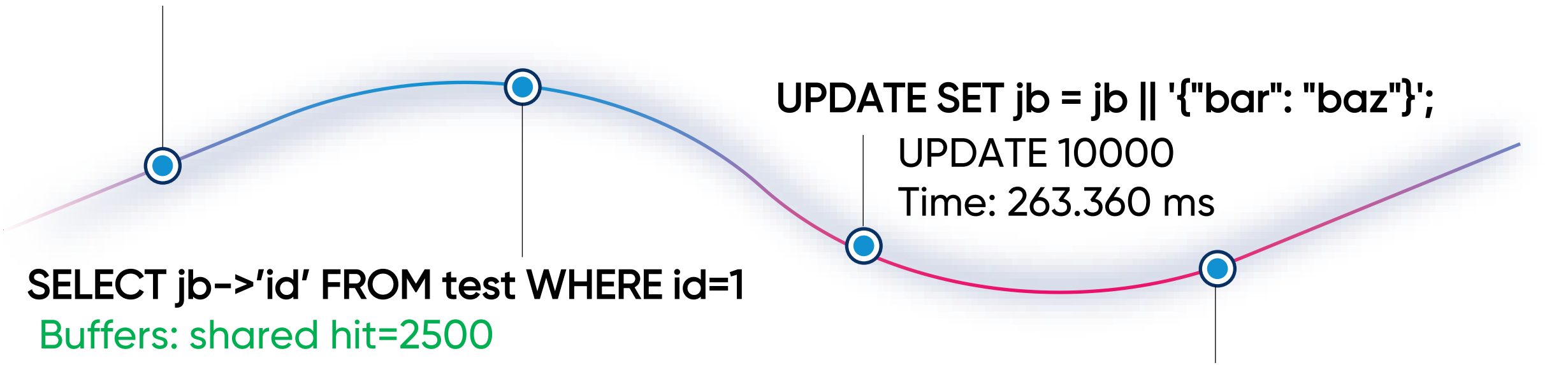
	json	jsonb
Storage format	Raw text	Binary, tree structure
Write	Slightly faster, minimal validation	Slower due to binary conversion
Read	Slower, reparsing needed	Faster
White spaces	Preserved	Removed
Duplicated keys	Preserved	Removed
Order of keys	Preserved	Could be altered
Index support	No	Yes
JSON item edit support	No	Yes

JSON, What's Wrong???

PostgresPro

Unpredictable performance degradation – simple update case:

INSERT values (1, '{"id":1,"foo":[0,0,...]}');



SELECT jb->'id' FROM test WHERE id=1

Buffers: shared hit=2500

Execution Time: 5.413 ms

UPDATE SET jb = jb || '{"bar": "baz"}';

UPDATE 10000

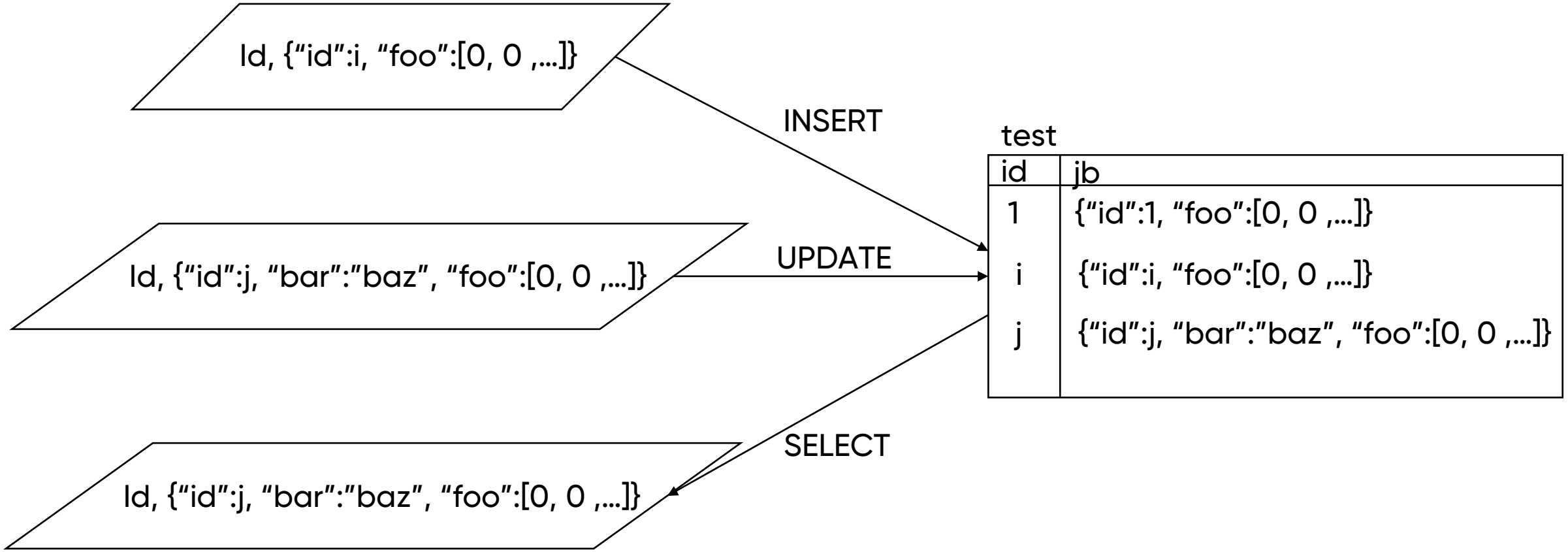
Time: 263.360 ms

SELECT jb->'id' FROM test WHERE id=1

Buffers: shared hit=30018

Execution Time: 34.089 ms

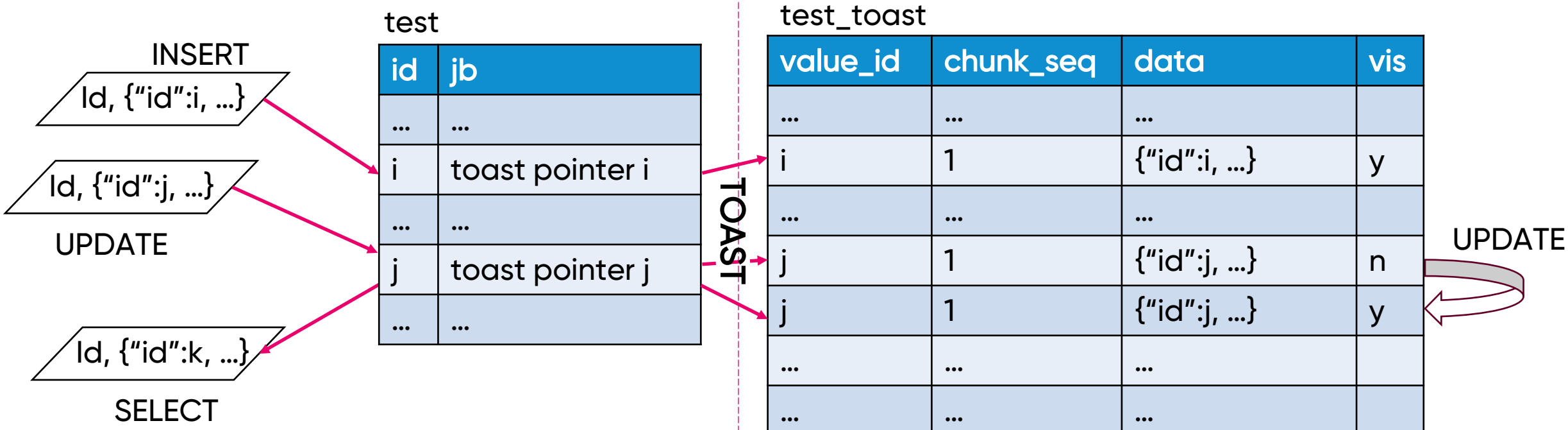
How It Looks - Data Flow Simplified



It's a Feature, Not a Bug

PostgresPro

(Dragon sleeps here!)



The Curse of TOAST

PostgresPro

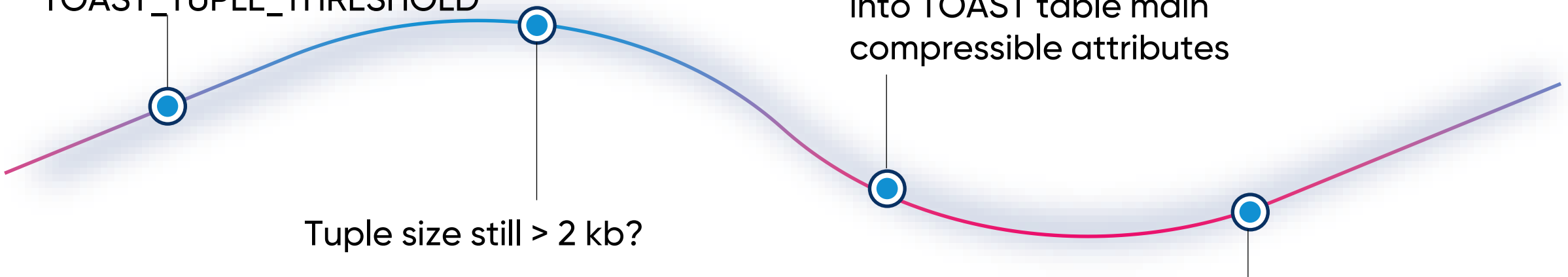
- Table and activity hidden from the User. User does not really know if data is stored in the original table or in the TOAST, and sometimes TOAST activities result in unexpected performance degradation;
- Bloating – TOAST does not implement real UPDATE, having TOAST table size growing very fast and lots of WAL traffic on large data and update-heavy tables as a side-effect;
- Implicit limitations which are not obvious to User, i.e. 4 billions TOAST value ids available, and sometimes strange behavior due to these limitations.

PostgresPro

How the TOAST is Cooked

Tuple size > 2 kb?

Compress and store into TOAST table biggest attributes > TOAST_TUPLE_THRESHOLD



Tuple size still > 2 kb?

Store into TOAST table all extended and external attributes

Tuple size still > 2 kb?

Compress and store into TOAST table main compressible attributes

Tuple still > 2 kb?

Store into TOAST table main attributes left

The TOAST Pointer

PostgresPro

1 byte	1 byte	4 bytes	4 bytes	4 bytes	4 bytes
va_header	va_tag	va_rawsize	va_extinfo	va_valueid	va_toastrelid

varlena header

toast pointer

TOAST Pointer stores
- original value size
- external values size and compression method
- external value ID for lookup
- TOAST table relation ID

toast relation

value_id	chunk_seq	data
...	...	...
i	j	...
...	...	...

Add Some Sugar and Spice

Hardcoded TOAST strategy

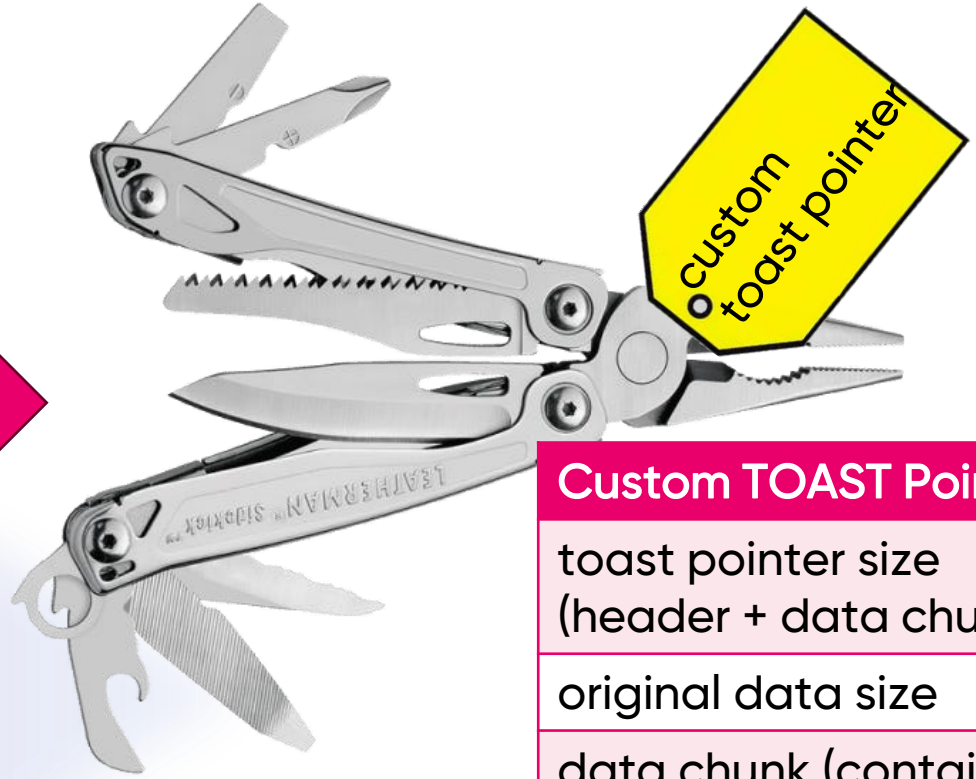
no UPDATE

rigid TOAST pointer format

It just works

Suitable for almost all data types

Flexible and adaptable

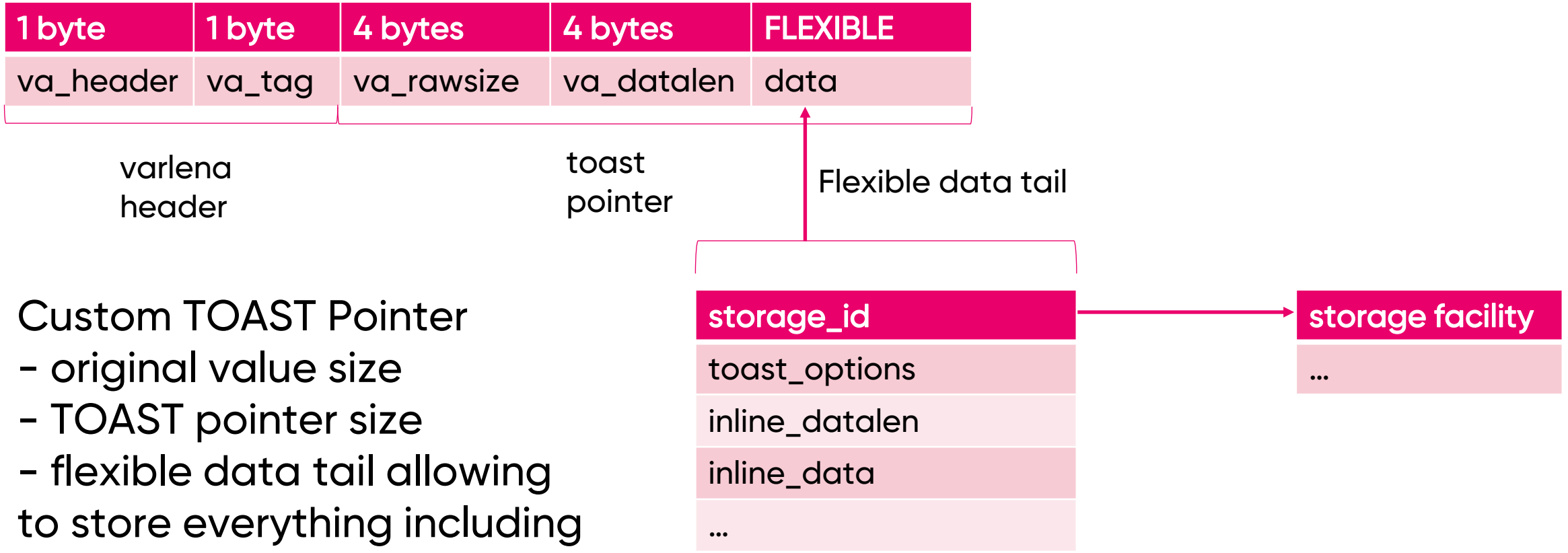


Custom TOAST Pointer
toast pointer size (header + data chunk size)
original data size
data chunk (container)

original data size
TOAST relation id

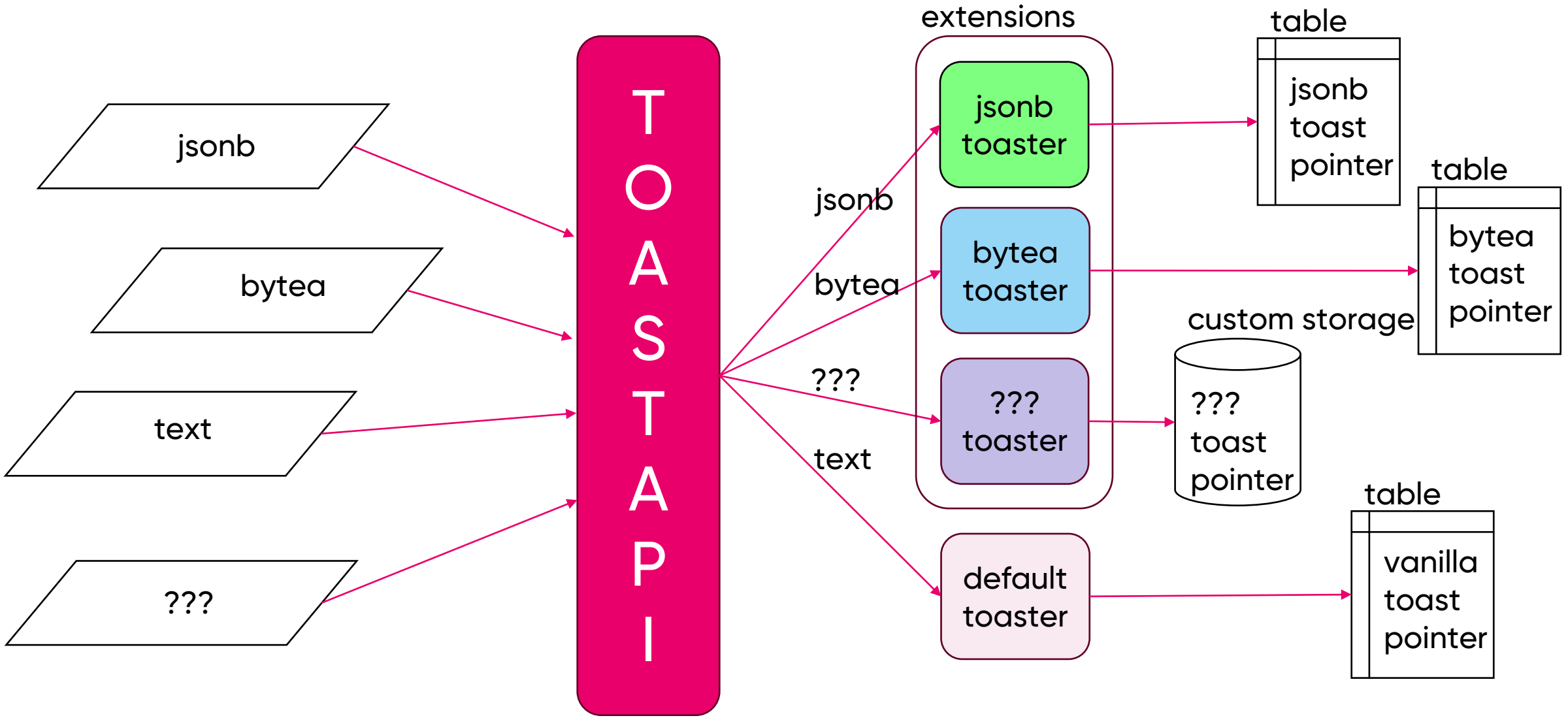
The Custom TOAST Pointer

PostgresPro



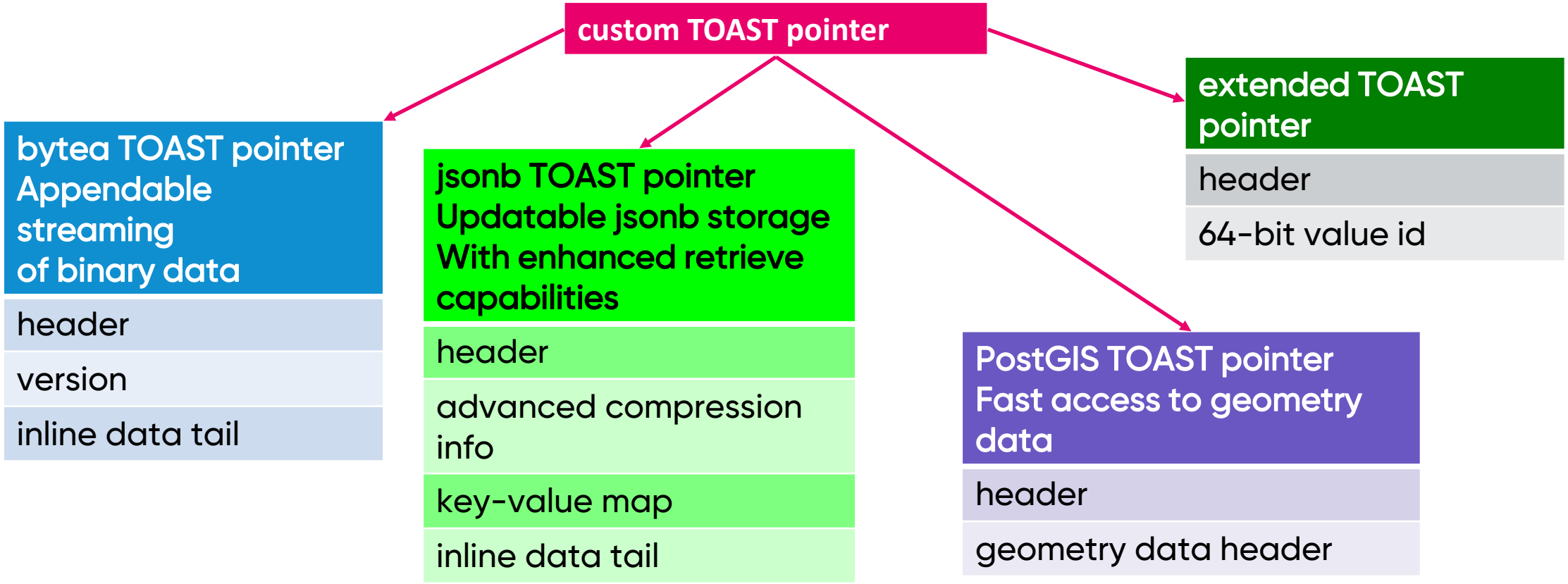
Custom TOAST Pointer
- original value size
- TOAST pointer size
- flexible data tail allowing to store everything including small chunks of inline data

The TOAST API



The Pluggable TOAST

- Faster access. Small amounts of data could be stored inline
- Specific. Could use knowledge of internals of data types being TOASTed
- Flexible. Data chunk could be used to store any kind of data we need



jsonb Toaster Highlights

PostgresPro

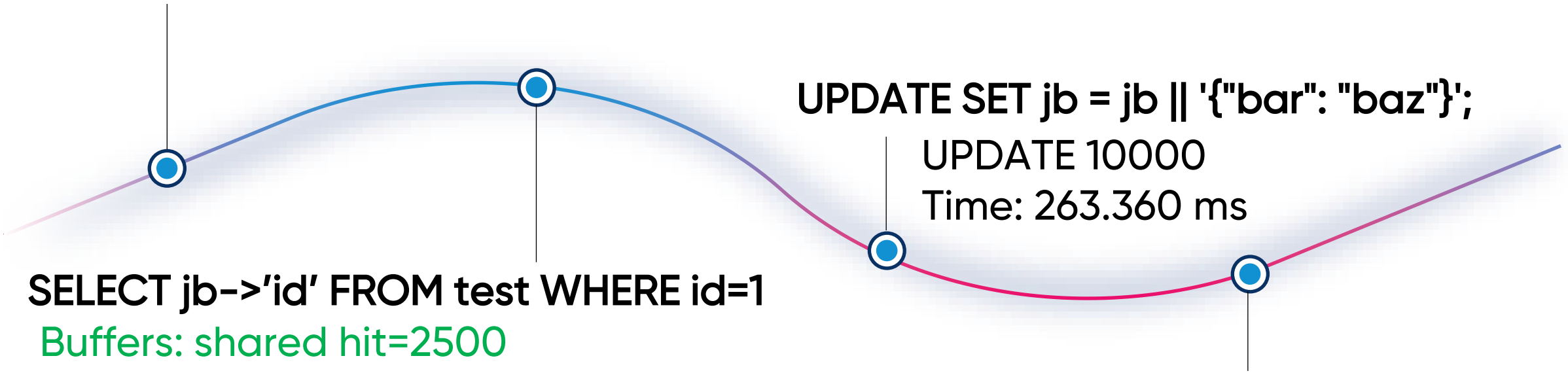
- Uses knowledge of internal structure of JSON objects
- Provides additional set of functions used by Postgres JSONb API
- Implements UPDATE for JSON objects
- Greatly improves performance of all operations on Jsonb object

JSONb, What's Wrong???

PostgresPro

Unpredictable performance degradation – simple update case:

INSERT values (1, '{"id":1,"foo":[0,0,...]}');



UPDATE SET jb = jb || '{"bar": "baz"}';

UPDATE 10000

Time: 263.360 ms

SELECT jb->'id' FROM test WHERE id=1

Buffers: shared hit=2500

Execution Time: 5.413 ms

7x!

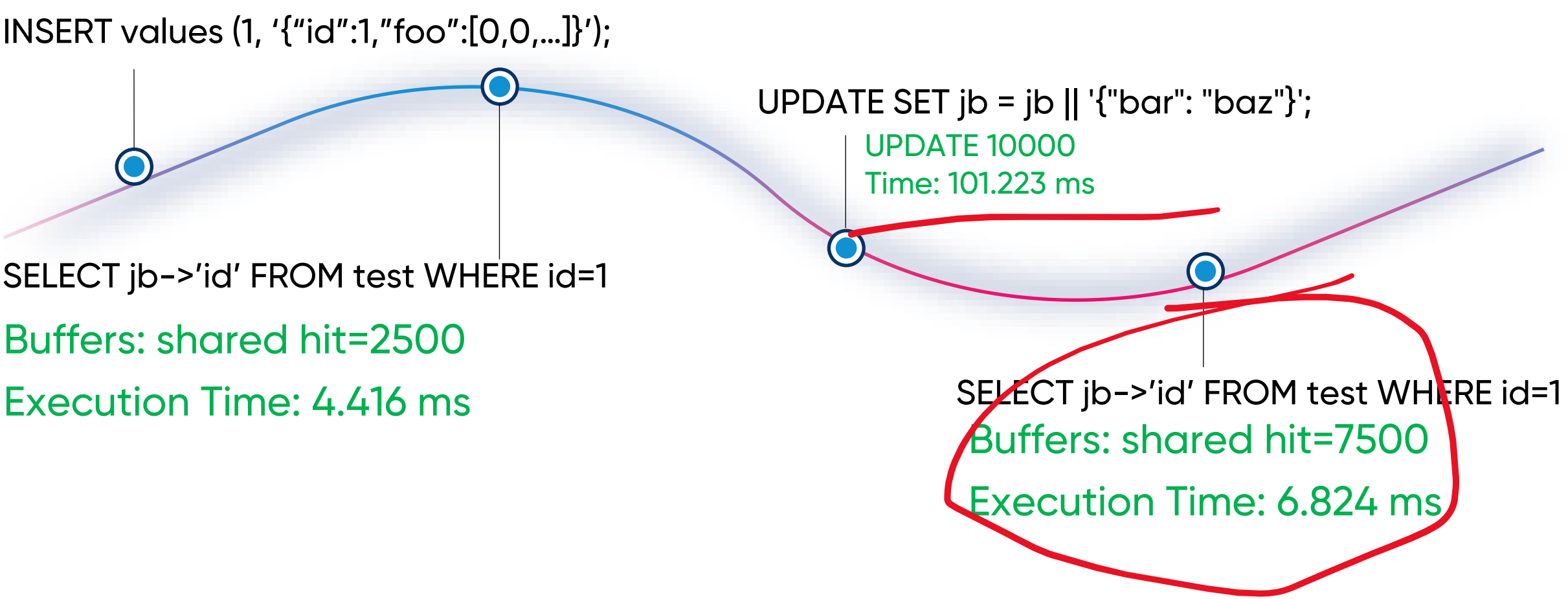
SELECT jb->'id' FROM test WHERE id=1

Buffers: shared hit=30018

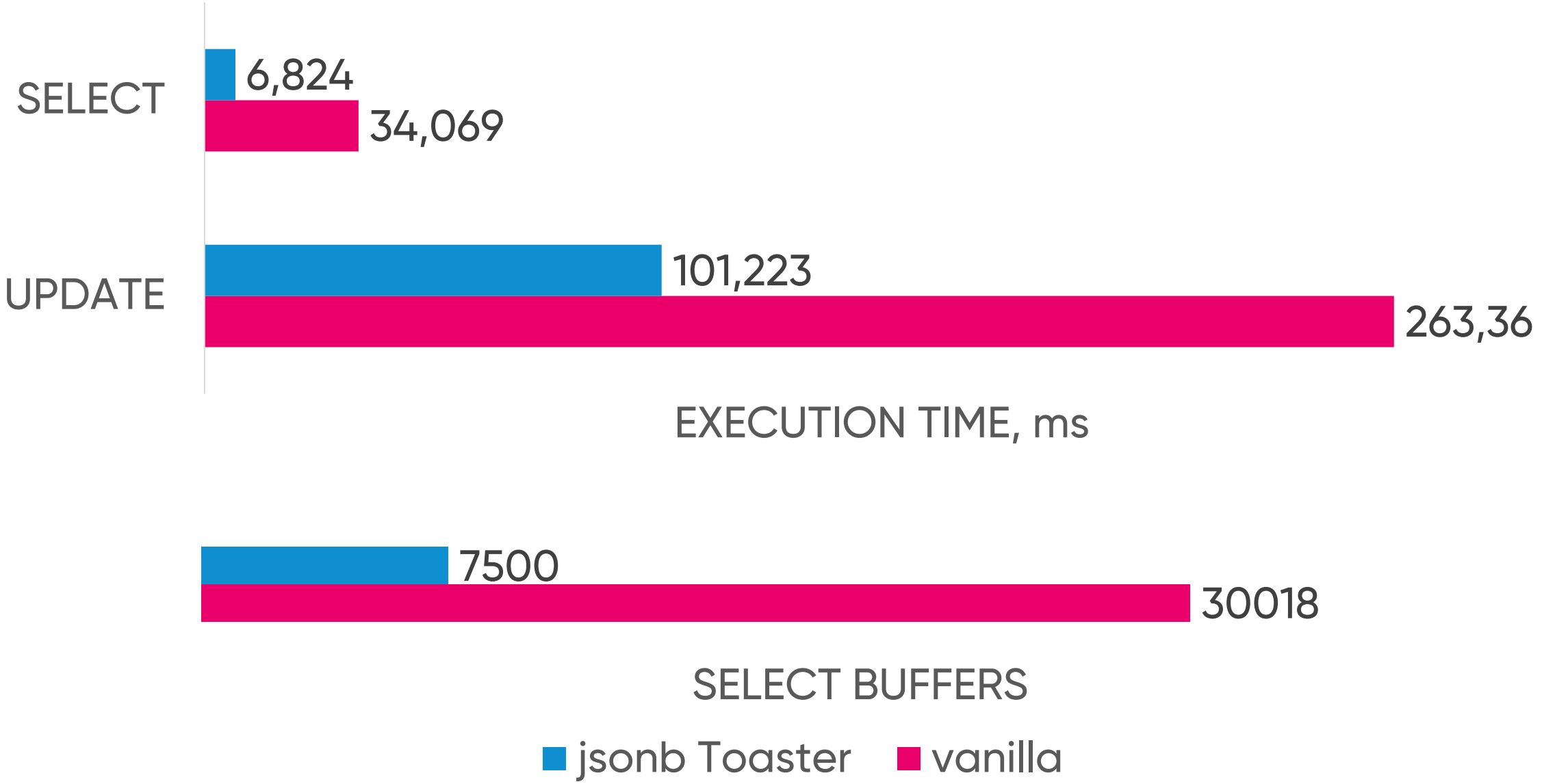
Execution Time: 34.089 ms

Let's Cook JSONb TOAST Anew

The same case as before – long jsonb object with array, simple update:



SELECT and UPDATE

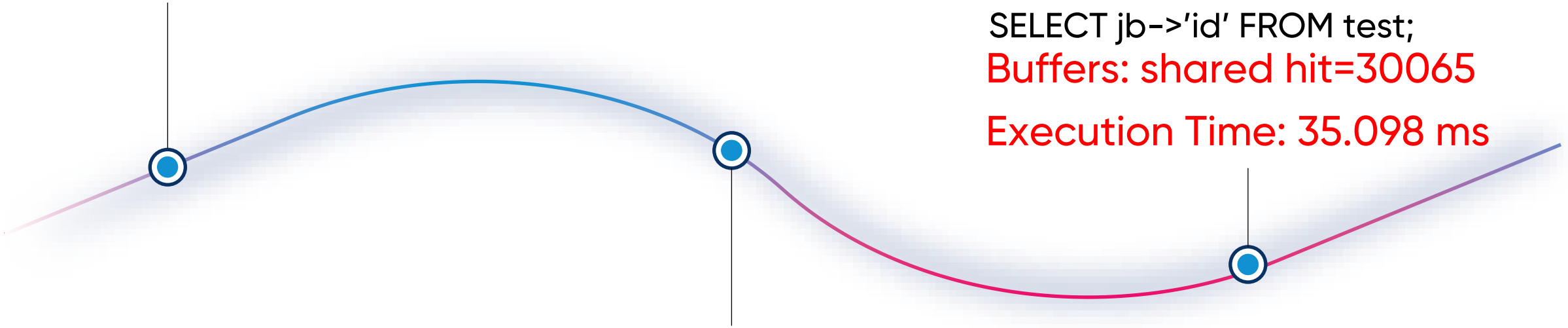


New Rules for the Game

PostgresPro

~300kb jsonb, short value after long key-value pair, **vanilla**:

INSERT values (1, '{"id":1,"bar":[0,0,...]}, "foo":"baz"');



SELECT jb->'id' FROM test;

Buffers: shared hit=30065

Execution Time: 35.098 ms

SELECT jb->'foo' FROM test;

Buffers: shared hit=30065

Execution Time: 37.631 ms

With Iterative Detoast and Sorted Keys

PostgresPro

Same value, same query, **jsonb Toaster**:

INSERT values (1, '{"id":1,"bar":[0,0,...]}, "foo":"baz"');

SELECT jb->'id' FROM test;

Buffers: shared hit=2500

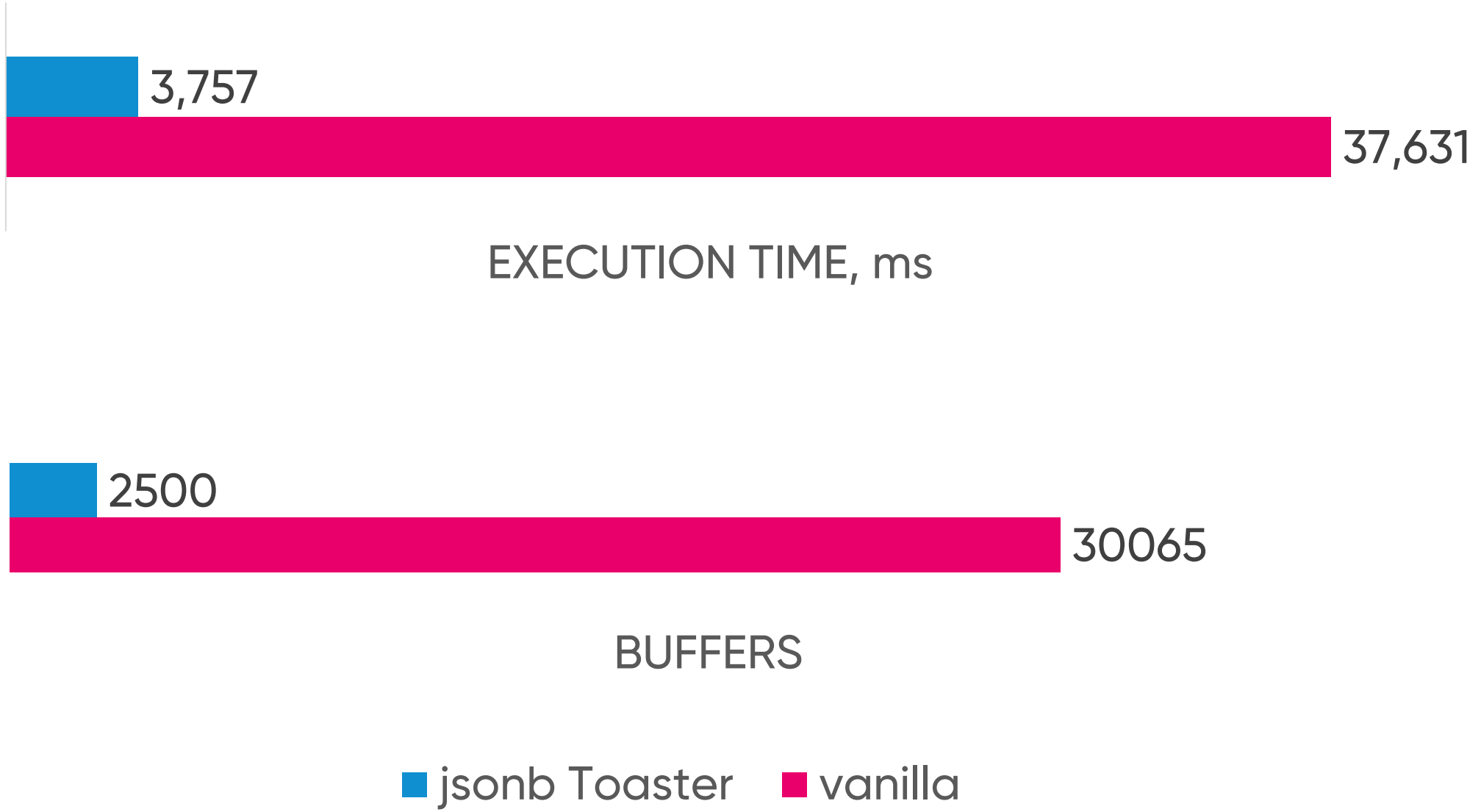
Execution Time: 3.757 ms

SELECT jb->'foo' FROM test;

Buffers: shared hit=2500

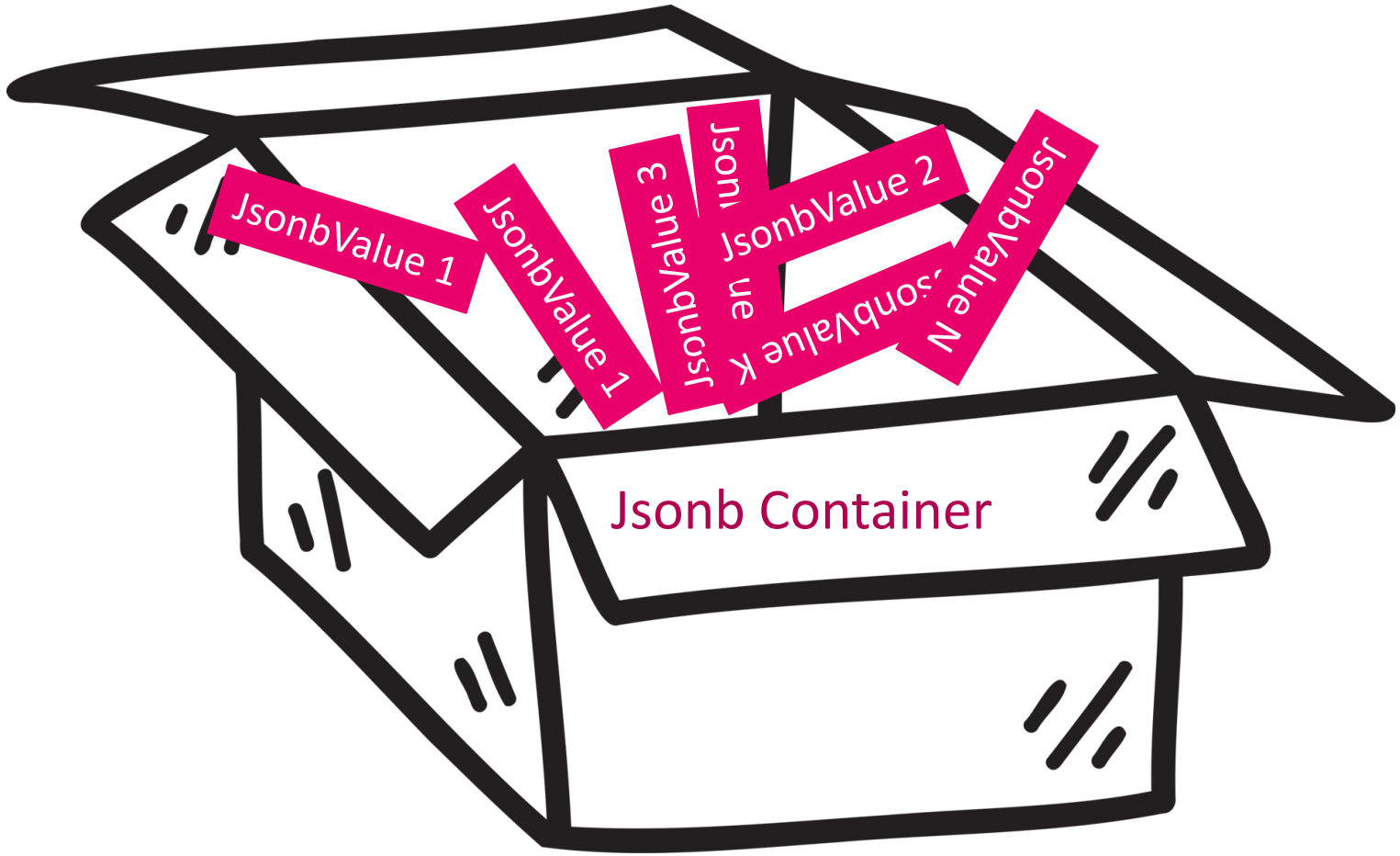
Execution Time: 4.901 ms

Full Detoast vs Partial



Vanilla Jsonb Container

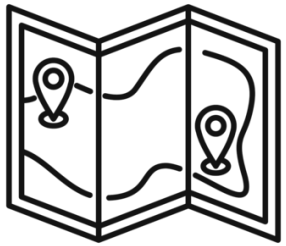
Keys sorted by length, name.
Values appear, in an order of keys.
Fully detoasted to access any value

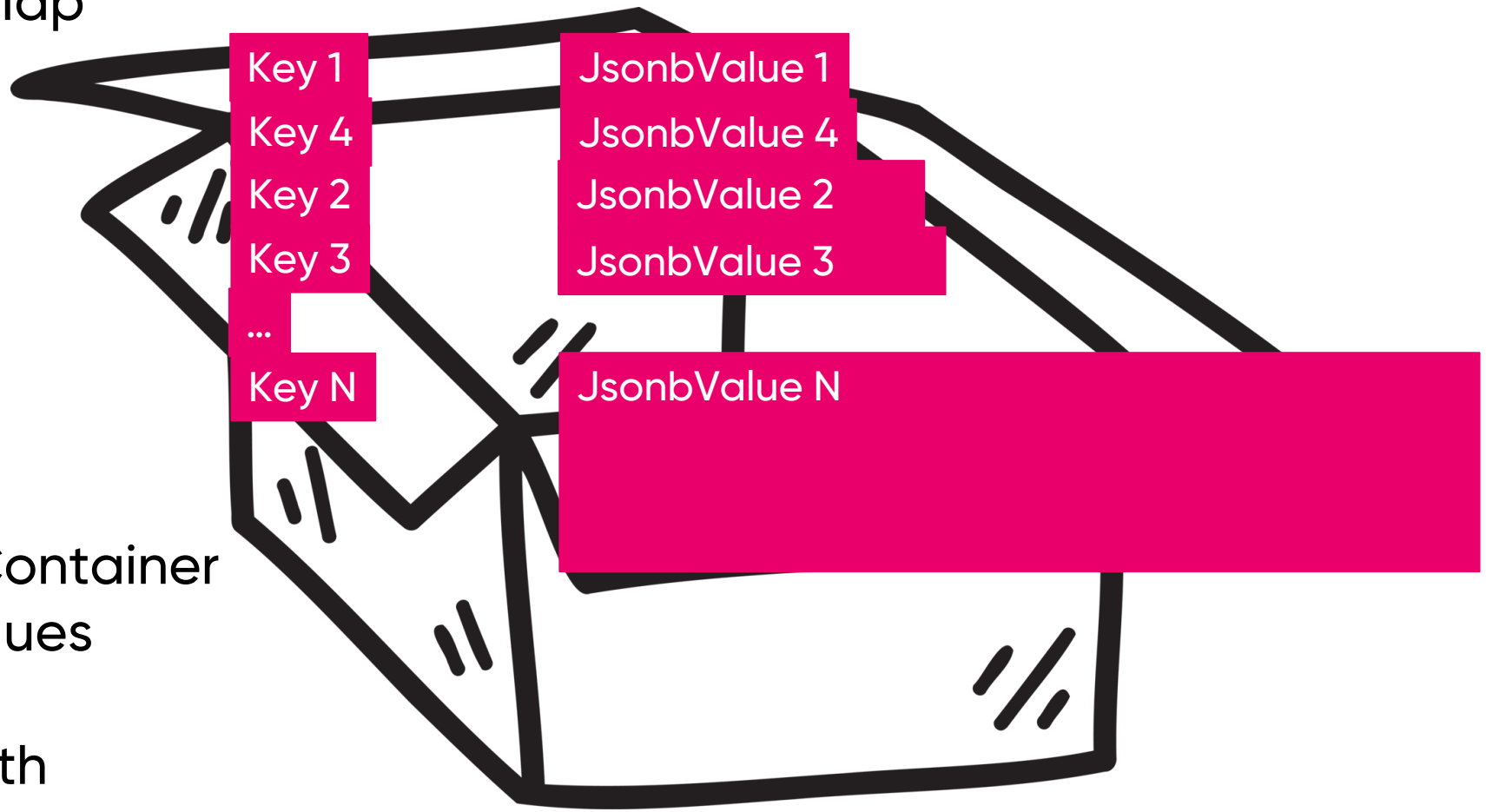


Sorted Keys

PostgresPro

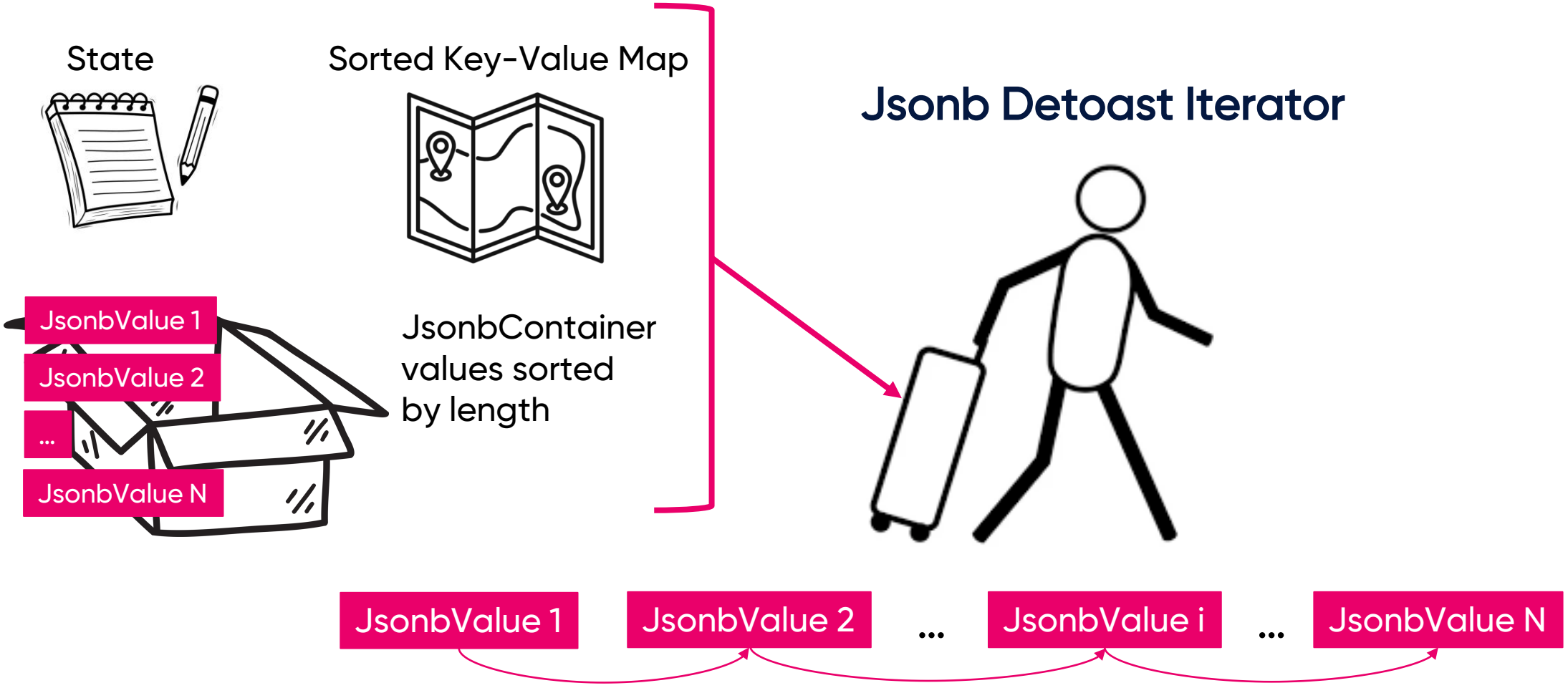
Sorted Key-Value Map





JsonbContainer
with values
sorted
by length

Partial Iterative Detoast



Detoast values one by one on demand

Out-of-line Long Values

PostgresPro

Jsonb Container

Nested Jsonb Container

Store long json values out-of-line, replacing with TOAST pointers, detoasting on demand

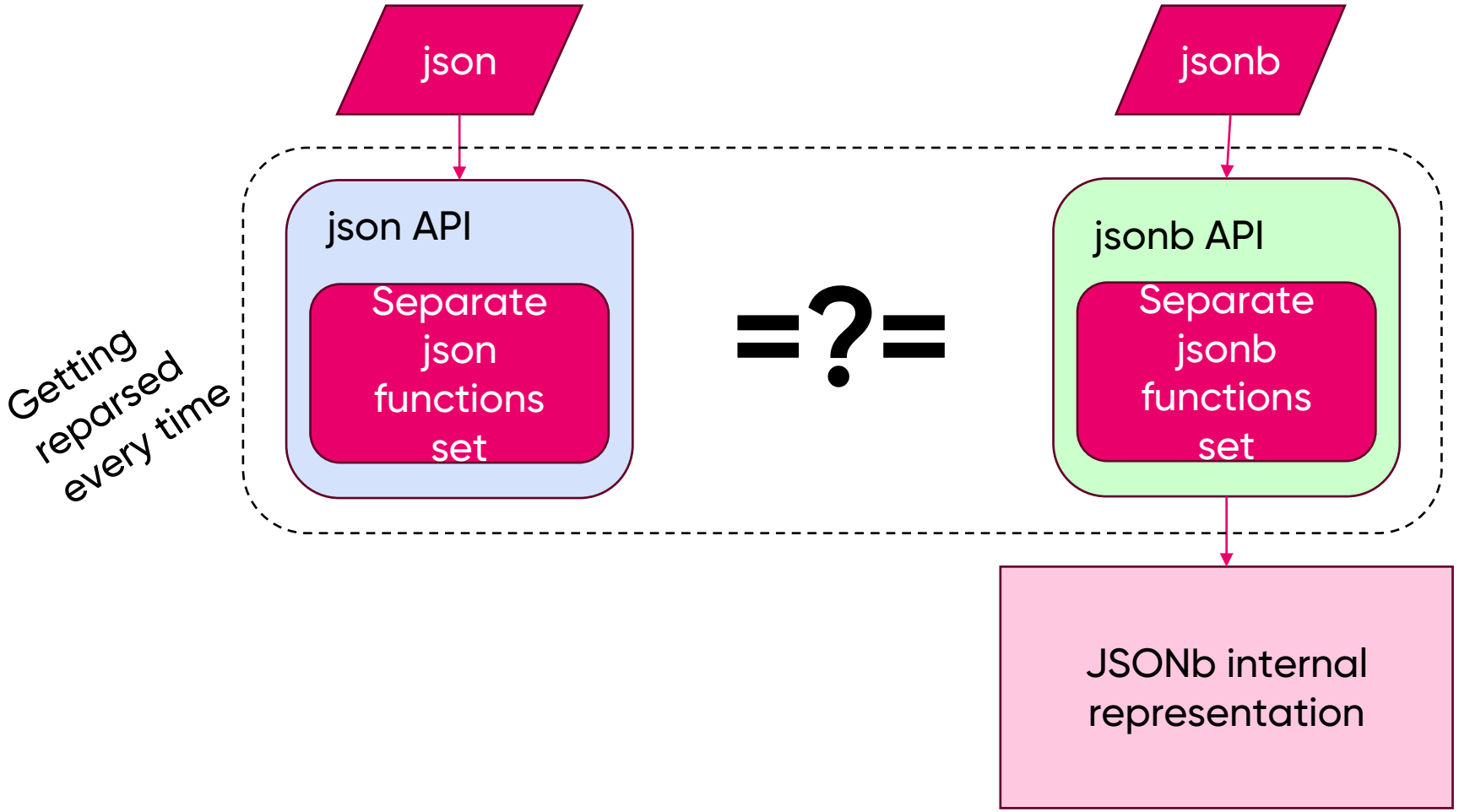
Further Improvements

PostgresPro

- Shared deTOASTed values. Do not discard value immediately after usage, to avoid sequential detoast of the same value – the PostgreSQL community is already working on it;
- Integrated support of JSON schema, which would benefit a lot from key-value maps and partial detoast;
- Common interface from JSON and JSONb facilities;
- Many more possible.

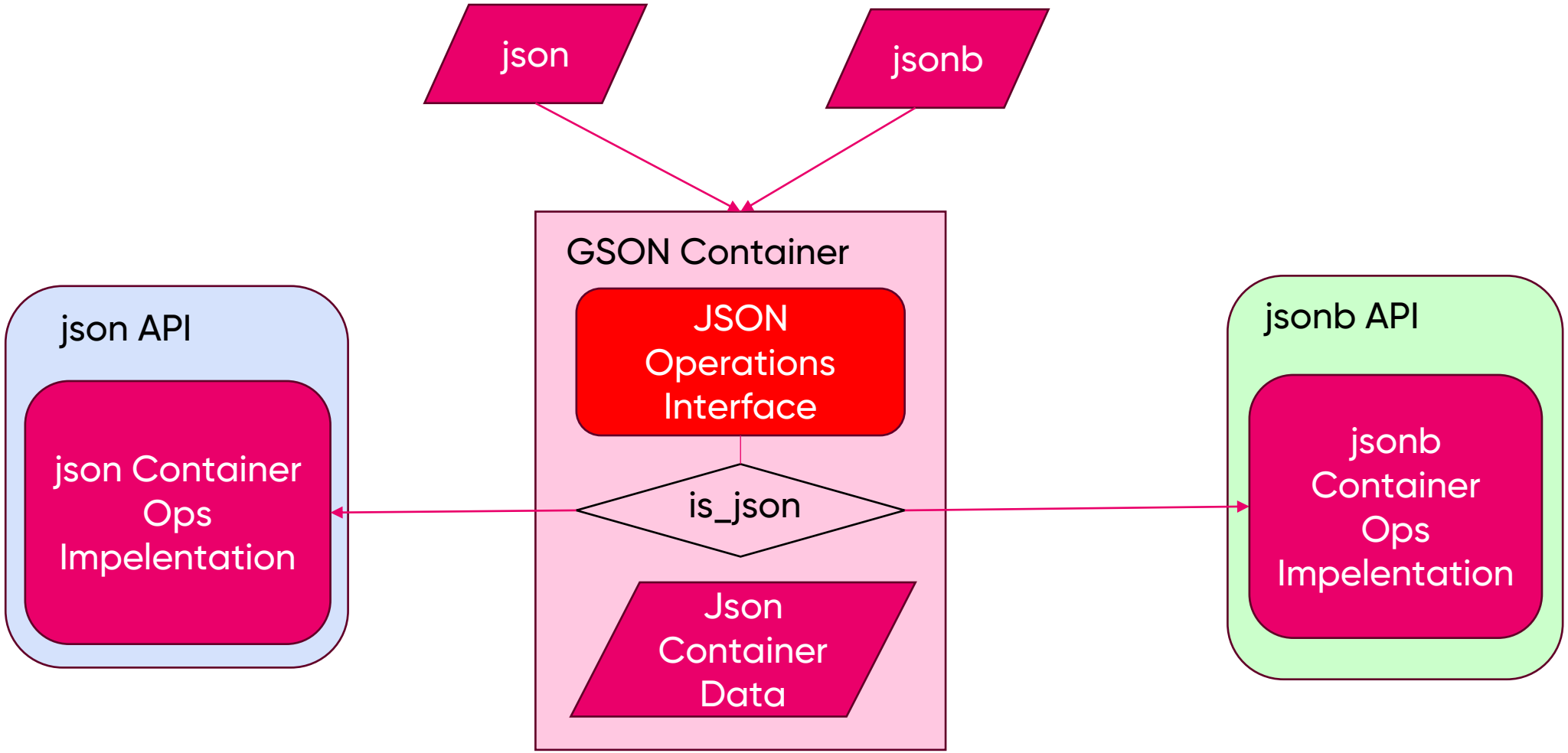
Json + Jsonb = ?

In Postgres, json and jsonb datatypes look somewhat different though describe the same JSON objects



Json + Jsonb = GSON!

Generic JSON - a common container for internal representation for both json and jsonb data with common functions interface, the json and jsonb types just use their own implementation



TODO

PostgresPro

- Commit custom TOAST pointer to vanilla;
- Port jsonb Toaster to current vanilla master. Current master has lots of changes compared to v15 where this feature was developed, and porting is complex and painful due to very invasive nature of the patch;
- Integrate jsonb Toaster with SQL/JSON support functions being commited into current master to further improve their capabilities;
- Benchmark jsonb Toaster with live data;
- Propose GSON to PostgreSQL community and collect a feedback on this feature.

Open Issues

PostgresPro

- Uncertainty in dropping custom TOAST implementation flow – to drop Toaster whole set of data processed with in needs to be re-TOASTed or data will be lost;
- Logical replication of updated TOASTed tuples it is a subject for further discussion;
- Controversial points of view on the TOAST API - should this feature be an API or advanced TOAST mechanics should not be extensions but parts of a certain datatypes

Conclusion

PostgresPro

TOAST API, Pluggable TOAST and custom Toasters are promising complex solution for some of the long-lived PostgreSQL issues, though a lot of work still have to be done, and this approach is not perfect and has both strengths and weaknesses.

- Strength:

Performance. The legal way to introduce powerful optimizations;

Flexibility. Use any knowledge of stored data structure and any storage.

- Weakness:

Dependency on extension. Losing extension could result in data loss;

Complexity. Developing custom Toaster requires deep knowledge of Postgres internals and data being processed;

Acknowledgements

- Oleg Bartunov – the lead evangelists of PostgreSQL in Russia, and the visioner of it's future
- Nikita Glukhov – the major contributor of JSONb and SQL/JSON, inventor of GSON, without whom this work won't be possible

References

PostgresPro

- New TOAST in Town http://www.sai.msu.su/~megera/postgres/talks/toast-nizhny-2022.pdf
- One TOAST Fits All http://www.sai.msu.su/~megera/postgres/talks/toast-highload-2022.pdf
- Json or not Json http://www.sai.msu.su/~megera/postgres/talks/jsonb-spb-2021.pdf
- Scaling JSONB - http://www.sai.msu.su/~megera/postgres/talks/jsonb-pgvision-2021.pdf
- Jsonb on Steroids - http://www.sai.msu.su/~megera/postgres/talks/jsonb-highload-2021.pdf
- Jsonb Internals - http://www.sai.msu.su/~megera/postgres/talks/jsonb-pgconfonline-2021.pdf
- Understanding Jsonb performance
http://www.sai.msu.su/~megera/postgres/talks/jsonb-pgconfnyc-2021.pdf
- JSON and JSONB Unification (GSON)
http://www.sai.msu.su/~megera/postgres/talks/json-unification-database-meetup-2020.pdf

- Meet us at GitHub https://github.com/postgrespro/postgres/tree/toasterapi_clean

PostgresPro

Thank you for your attention!