TOAST LARGE OBJECTS

PGConf.ru, Moscow, June, 21, 2022

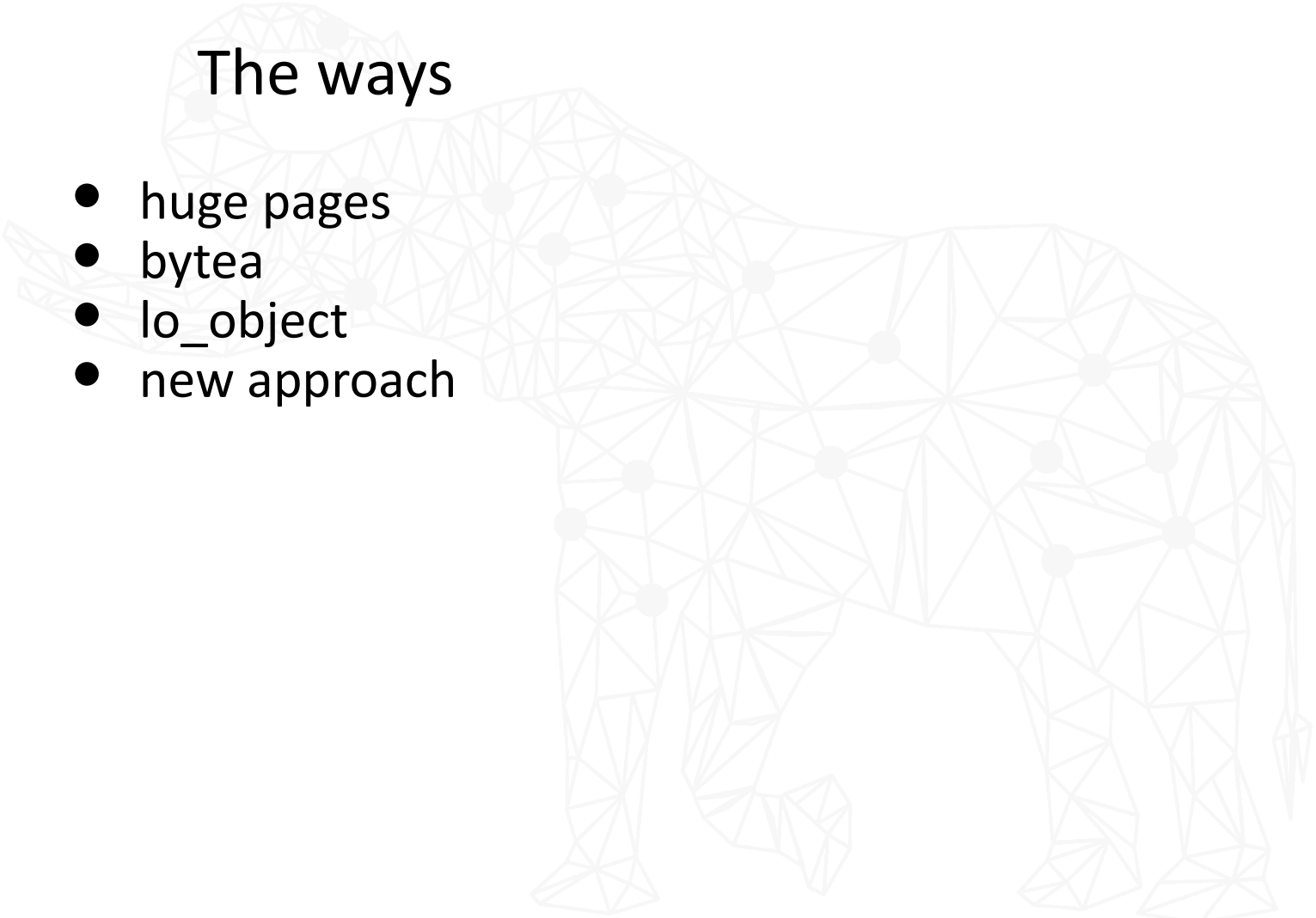
Customers

We would like to store file in relation database. How?
And do not ask us why.

PostgresPro

The ways

- huge pages
- bytea
- lo_object
- new approach



PostgresPro

1) huge pages

- increase page size up to 64kb
- first class citizen!
- no toast
- not so huge value …

2) bytea/text

- first class citizen
- toast issue (see TOASTER topic)
- hard limit 1Gb (varlena struct, protocol)
- soft limit < 1Gb
    - serialization in text - 1/2Gb
    - serialization in tuple - Σ <= 1Gb (1/2Gb in text)
    - dump issues

3) lo_object

- Oid lo_creat(PGconn *conn, int mode);
- file-like interface with oid as filedescriptor (very close)
- store oid in user table

Second class citizen

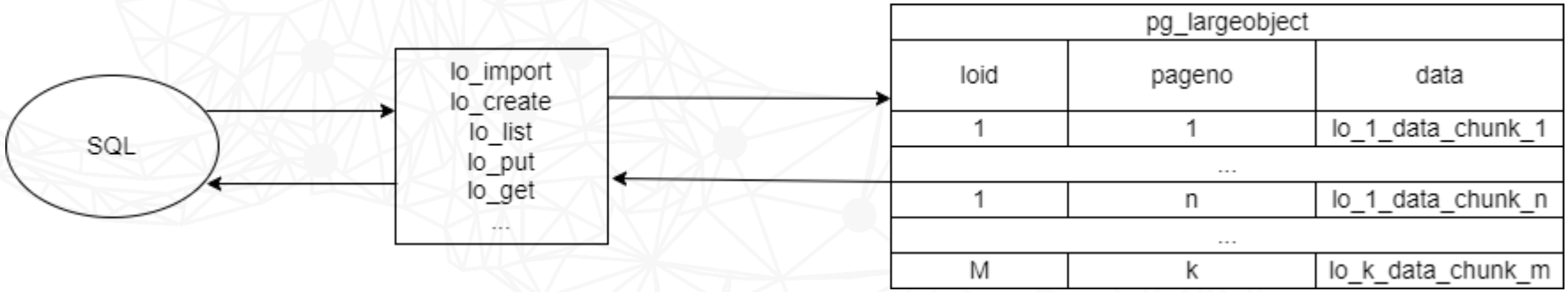
PostgresPro

3) lo_object

```
# \d pg_largeobject
Table "pg_catalog.pg_largeobject"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
   loid |  oid    |           | not null |
 pageno | integer |           | not null |
   data |  bytea  |           | not null |
Indexes:
"pg_largeobject_loid_pn_index" PRIMARY KEY, btree (loid, pageno)
```

PostgresPro

3) lo_object



- Limited access from SQL Level
- Creating, loading, and other operations are executed manually with lo_… functions set
- All large objects are stored in 1 (one) special system table
- Very tight system limits on objects storage

Surprise - pg_largeobject is, actually, Toast table too, with all limitations =)

PostgresPro

3) lo_object

- only 2^32 objects per database (infinite loop)
- only 32TB per database (max relation size, ERROR)
- 2TB per large object (2^31 * 2KB)
- only 2^15 1GB objects (~16000 films)

Restrictions do not correspond to modern realities

New Approach

TOAST means “The Oversized Attribute Storage Technique”.
What if we try to use our Pluggable TOAST to deal with such bulky entity like Large Object?

Let’s imagine we have Large Object datatype, that can be added as a column to regular tables, is fully accessible from SQL level - with INSERT, UPDATE, SELECT, functions to play with object’s internal data, and much less limits than set by PostgreSQL large objects.

Is it possible?

Let’s set the goal.

PostgresPro

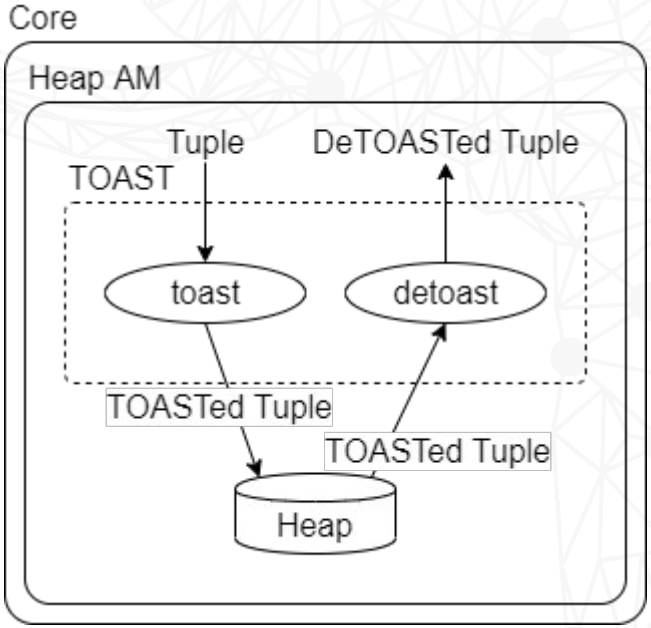
The Goal

1. Large Object - first class citizen in PostgreSQL;
2. Eliminate limitations of LO_OBJECT functionality;
3. Introduce human readable interface instead of ‘file descriptions’ being the OID numbers;
4. Keep the same functionality as provided by lo_* because of maximum in-memory object size of 1 Gb, including a lot of internal limits of 1 Gb, including protocol

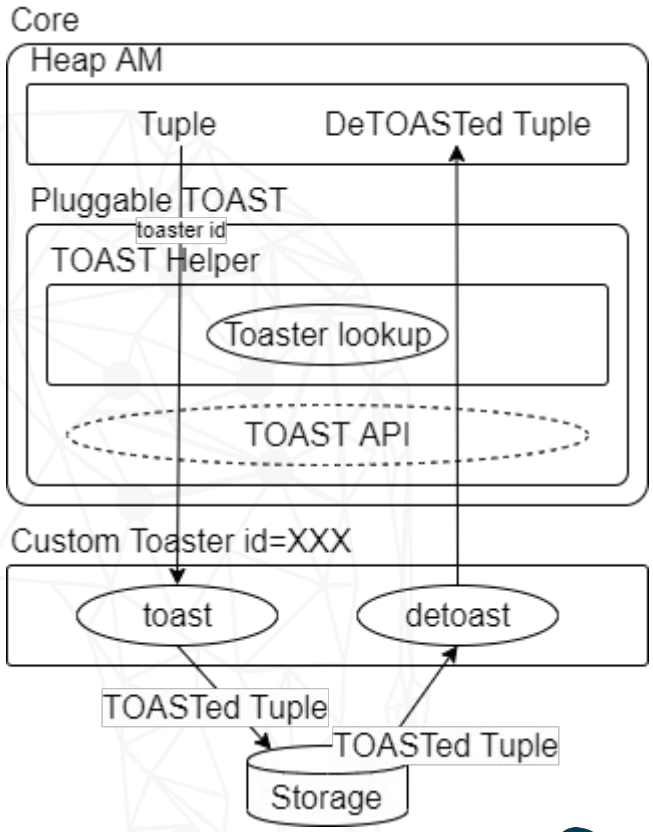
The Pluggable TOAST seems to give perfect opportunities to achieve these goals.

What is Pluggable TOAST?

Current TOAST



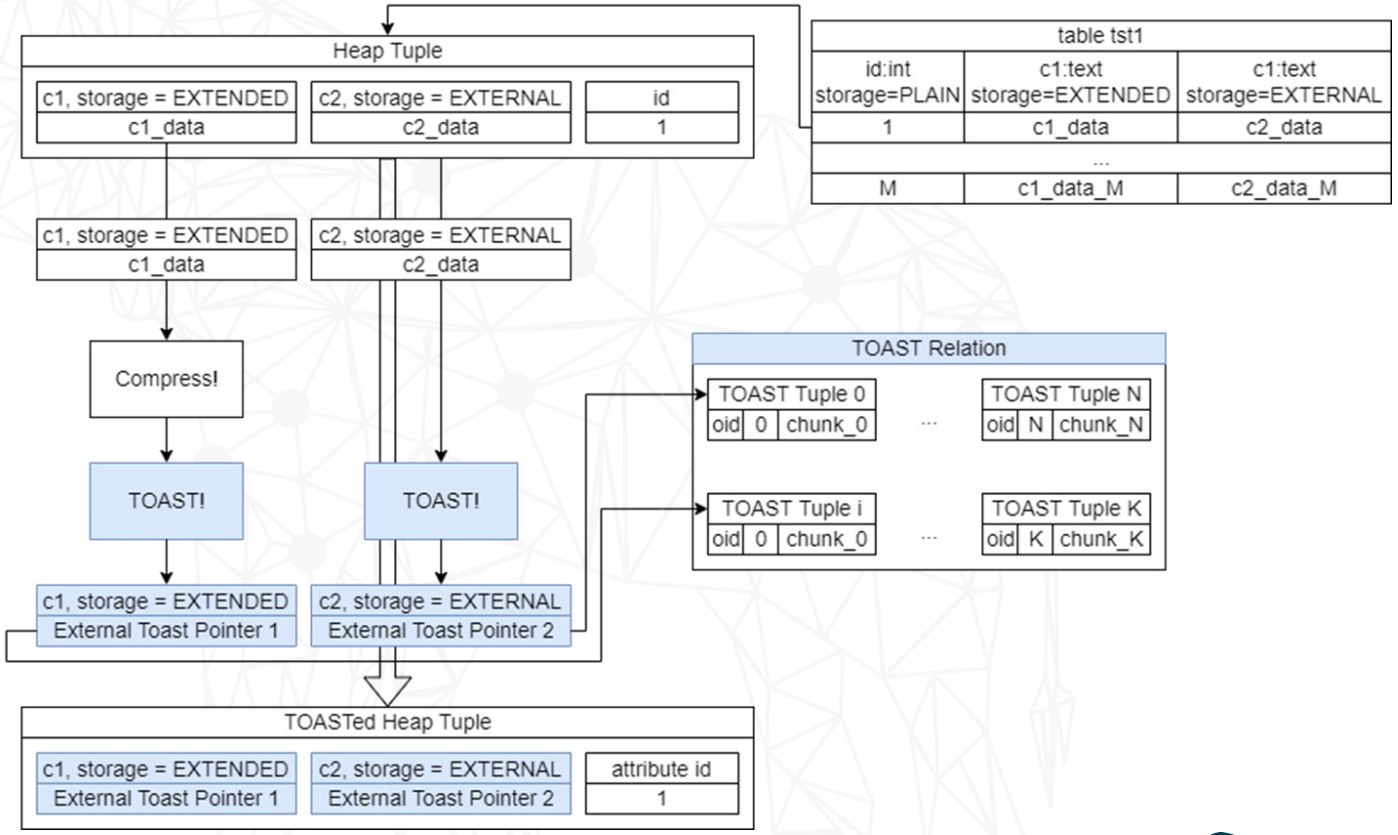
VS



PostgresPro

How TOAST Works

- When tuple size exceeds some internal limit Heap decides to TOAST large attributes
- Attribute stored in TOAST relation and replaced with Toast Pointer
- 4 strategies depending on storage type
- TOAST does not know anything about data being TOASTed



PostgresPro

Current TOAST Limitations

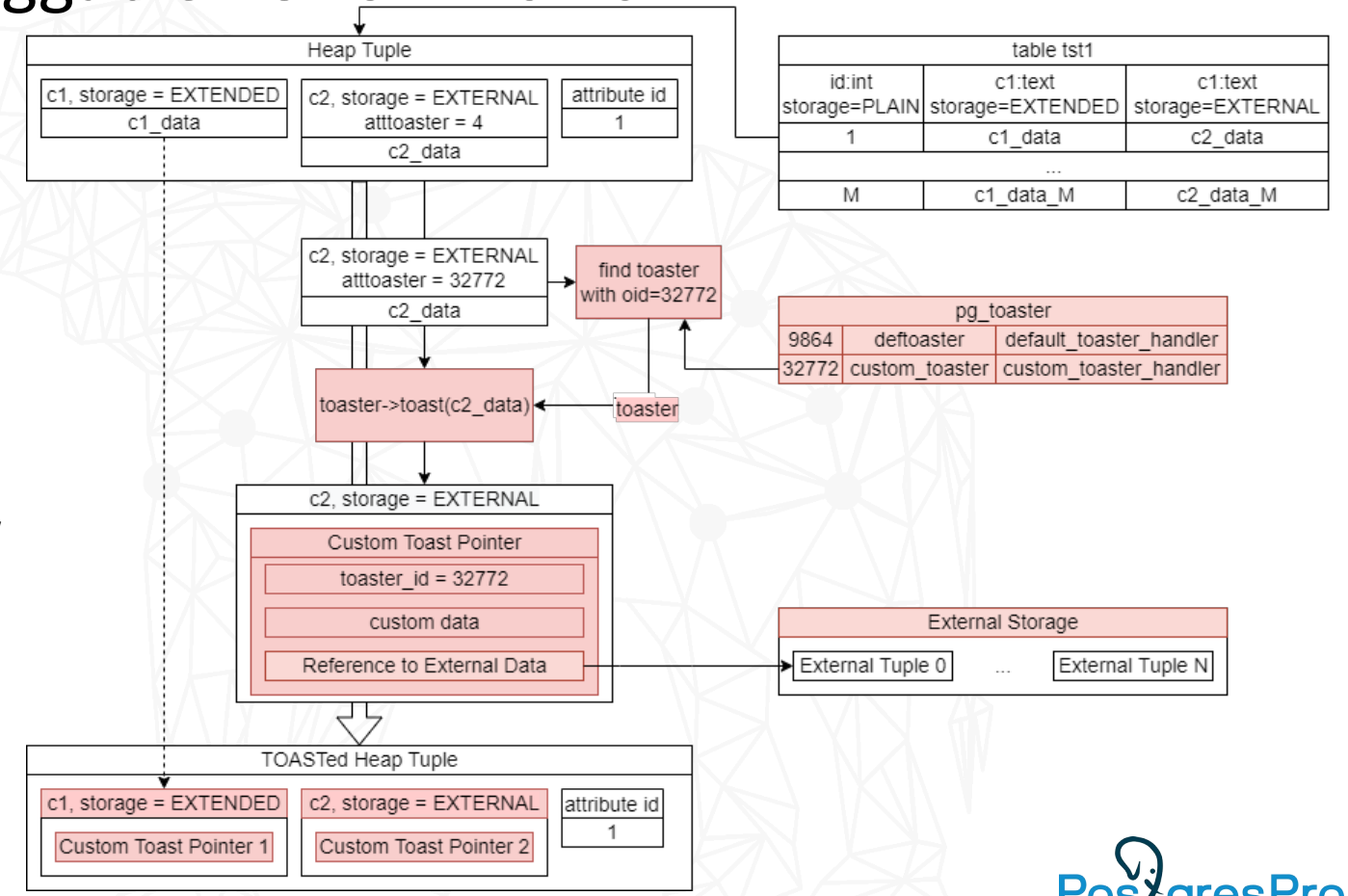
TOAST technology is trusted and reliable, and it just works, but it has limitations:

- One TOAST relation per table;
- TOAST relations is limited to 32 Tb in size as any other table
- TOASTed valued OIDs are taken from global system OID pool which is limited to 2^32 4294967295 values. When system OID pool is expended no object creations is possible and queries hang in endless waiting for free OID;
- Same TOAST strategy for all TOASTable datatypes;
- Unpredictable performance of queries related to TOAST tables.

Almost all these limitations could be avoided with use of custom Toaster fit to data being TOASTed

How Pluggable TOAST Works

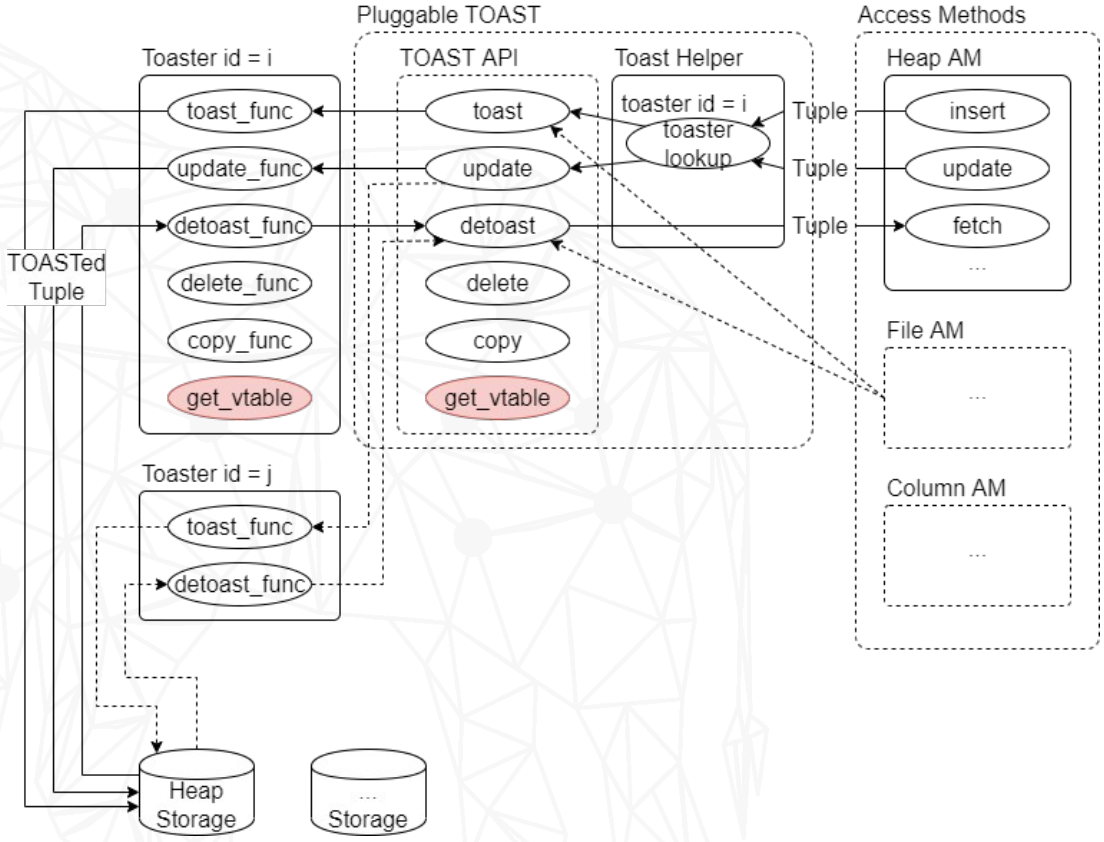
- Attribute is replaced with Custom Toast Pointer created by specific Toaster
- TOAST and storage details are hidden from AM behind the Toaster
- Toaster can use any knowledge of data structure and workflow
- Same data could be TOASTed with different Toasters



PostgresPro

TOAST API

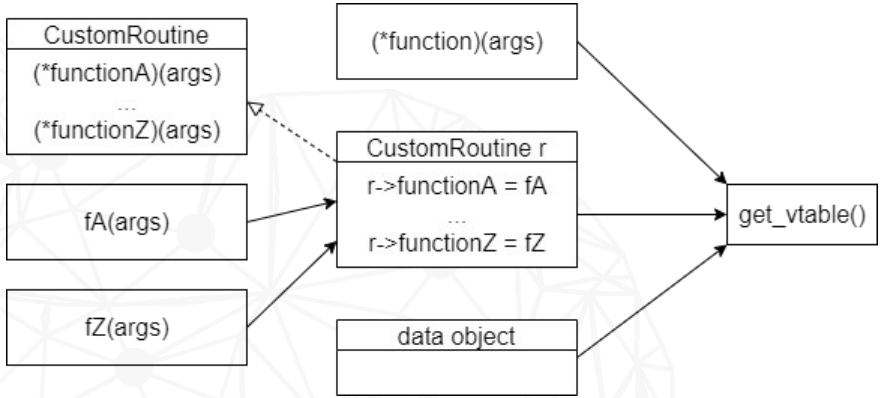
- Helper provides lookup functions and calls Toasters via API
- API defines entry points for main TOAST functions called by Helpers/AMs
- Toasters are responsible for storage-related functionality - store, fetch, compress etc.



PostgresPro

Extend The Extensible

- TOAST API already includes interface for user-defined custom functions



void * get_vtable()

A void pointer can be casted to any type – object, function or container

Yo dawg we heard you like APIs so we put an API in our API so you can extend Postgres while you extend Postgres

PostgresPro

TOAST API SQL Syntax

How superfile_toaster looks from SQL level:

```
create extension superfile_toaster;

create table t1 (id int, data superfile storage external toaster
superfile_toaster);

create table t2 (id int, data superfile);

alter table t3 alter column data set storage external;
alter table t3 alter column data set toaster superfile_toaster;
```

PostgresPro

Tools Ready To Use

What do we already have -

- TOAST API (TOAST Helper, SQL syntax support, core changes)
- Default (generic) Toaster implemented via TOAST API
- bytea appendable Toaster for bytea datatype with custom ‘append’ operation
- JSONB updatable Toaster for JSONB datatype with lots of optimizations

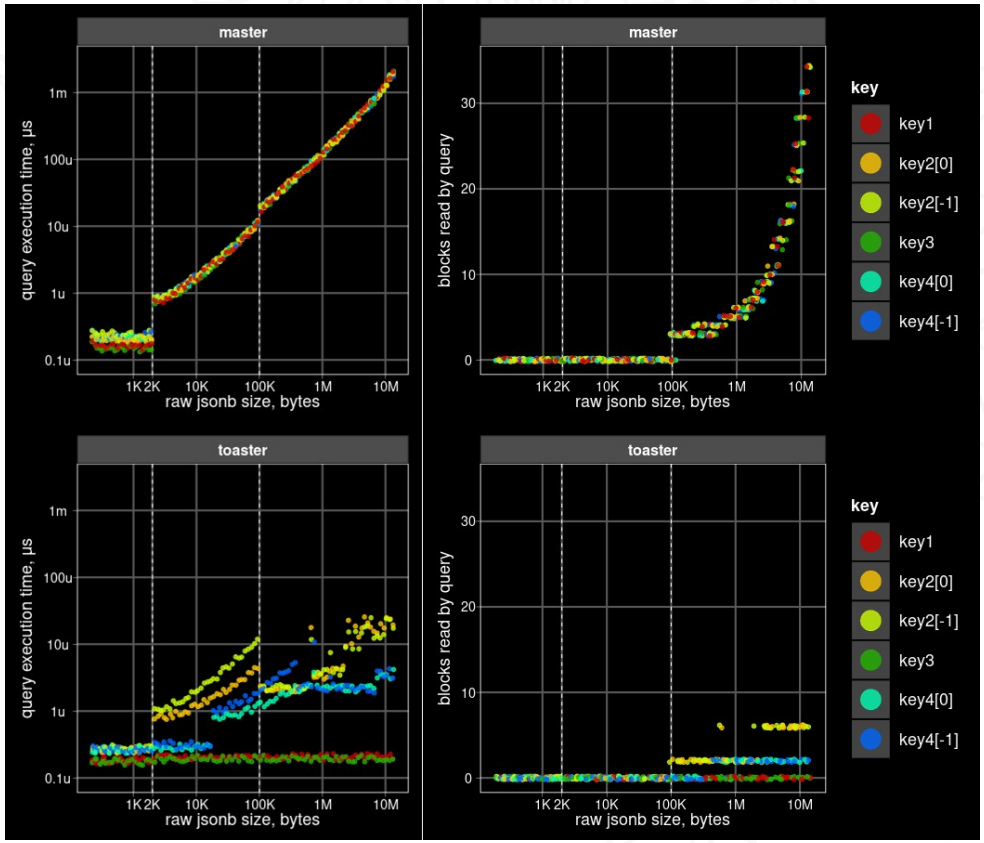
Large Objects look a lot like bytea data, but there are slight differences: they could be much, much bigger, have some special means of identification, i.e. by name.

PS: TOAST API does not have any noticeable overhead in comparison with original TOAST

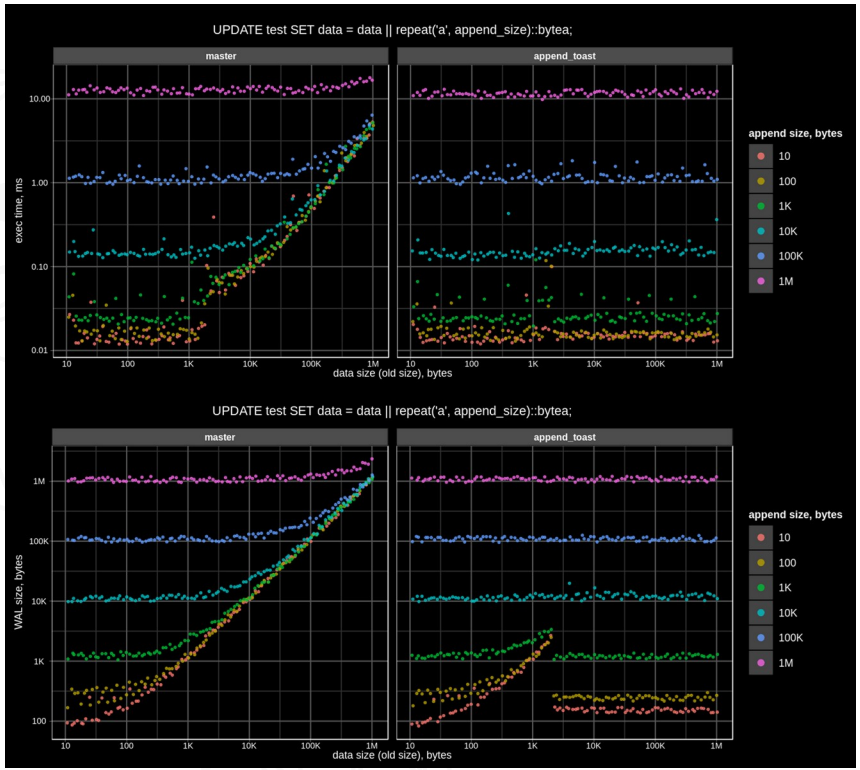
PostgresPro

Power of Custom Toasters

JSONB Toaster



bytea appendable Toaster



PostgresPro

What? Large Objects in Database?!

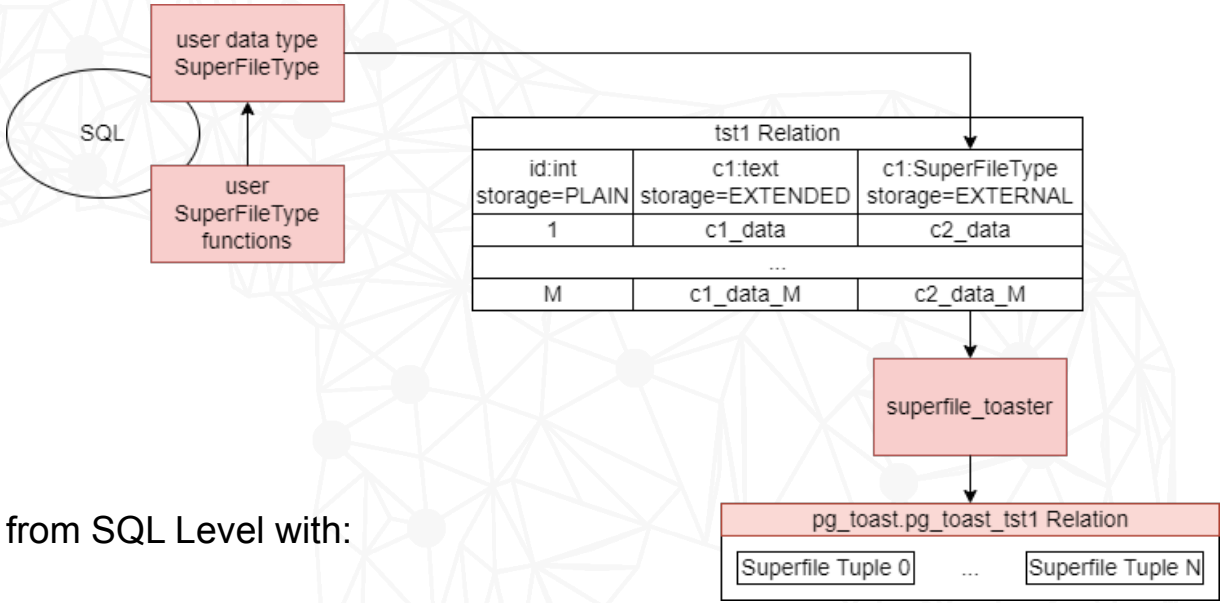
Yes, they are. With high-speed and large volume of modern storage systems - you can have very reliable data storage, because -

- Databases have large set of interfaces allowing safe access to data from almost any type of remote systems
- Stored large objects are replicated, dumped, restored by DBMS along with all your other data

From user level -

- Full-scale SQL access, not restricted with very limited set of no-sql functions
- New approach allows a lot of different customizations - extension of data type, new user functions and, new strategies and storage optimizations in Toaster
- Total objects size is limited only by Storage
- No impact on DBMS functionality and existing data, full backwards compatibility
- New datatype, Toaster and user functions are provided as an Extension

Large Objects With Benefits



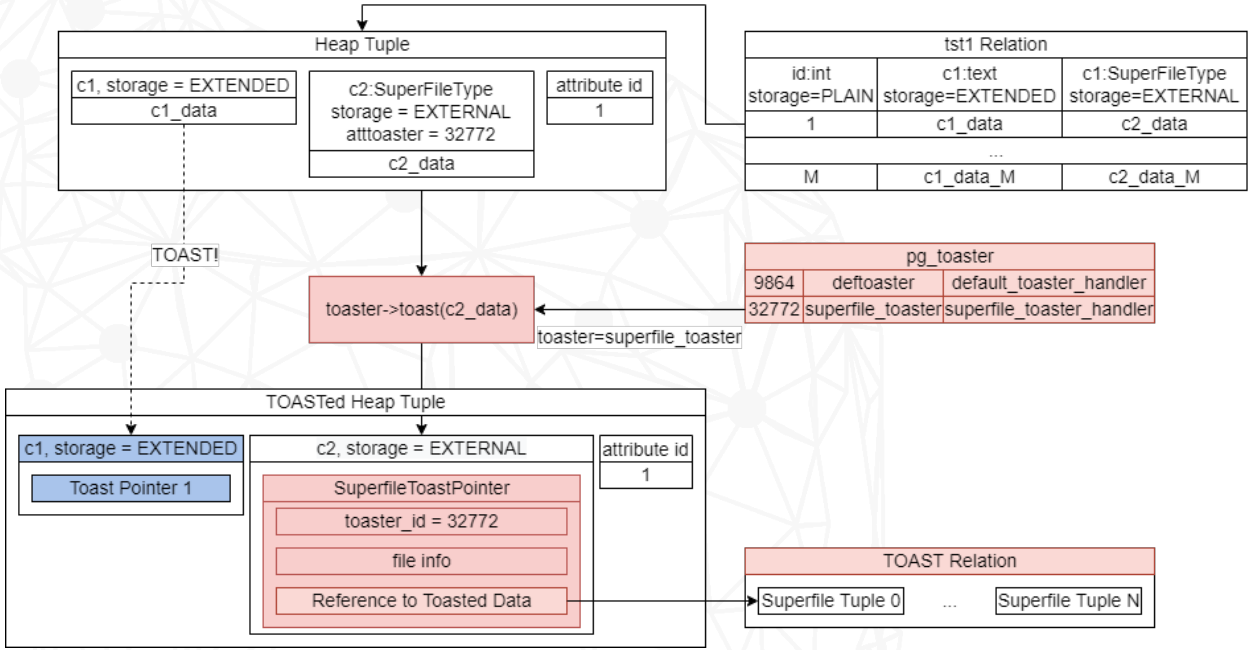
Full access from SQL Level with:

- user data type '**superfile**' to use as regular datatype
- user functions set to play with large data internals - store, fetch, search, etc.
- superfile_toaster to provide special storage support for **superfile** user type

PostgresPro

Let’s TOAST ‘Em!

- New user type stored in original table contains only information necessary to retrieve contents
- Large object contents stored in Toast tables with custom Toaster to hide all storage-related internal operations
- Custom large object Toaster could implement compression and other optimizations



SuperFile Prototype - User Type

```
CREATE FUNCTION sft_in(cstring)
RETURNS superfile
AS 'MODULE_PATHNAME'
LANGUAGE C IMMUTABLE RETURNS NULL ON NULL INPUT;

CREATE FUNCTION sft_out(superfile)
RETURNS cstring
AS 'MODULE_PATHNAME'
LANGUAGE C IMMUTABLE RETURNS NULL ON NULL INPUT;

CREATE TYPE superfile (
    INTERNALLENGTH = -1,
    INPUT = sft_in,
    OUTPUT = sft_out,
    STORAGE = external
);
```

PostgresPro

SuperFile Prototype - Toaster And Interface

```
CREATE FUNCTION superfile_toaster_handler(internal)
RETURNS toaster_handler
AS 'MODULE_PATHNAME'
LANGUAGE C;
CREATE TOASTER superfile_toaster  HANDLER superfile_toaster_handler;

CREATE FUNCTION sf_put(superfile, bytea, integer) RETURNS superfile
AS 'MODULE_PATHNAME'
LANGUAGE C;

CREATE FUNCTION sf_get(superfile, integer, integer) RETURNS cstring
AS 'MODULE_PATHNAME'
LANGUAGE C;
```

PostgresPro

SuperFile Prototype - Internals

C internal representation:

```
typedef struct {
    int32    varlena;
    int64    valid;
    int64    size;
    int32    last_chunk;
    Oid      relation;
    Oid      toastrelation;
    Oid      toasterid;
    char name[FLEXIBLE_ARRAY_MEMBER];
} SuperFileType;
```

Due to internal restriction of PostgreSQL values could not exceed size of 1 Gb.
The solution is set of user functions writing buffers less then 1 Gb (actually, less then 1 Gb) in size

PostgresPro

Superfile In Action

```
create table t1 (id int, data superfile storage external toaster
superfile_toaster);
CREATE TABLE
Time: 11.430 ms
insert into t1 values (1, '(0, 0, 0, 0, 0, 0, /etc/passwd )'::superfile)
returning data;
                  data
---------------------------------------
 (0, 0, 0, 0, 0, 0, /etc/passwd)
(1 row)

INSERT 0 1
Time: 4.617 ms
```

Data buffer insert used for testing:

```
select sf_put('(1, <value_size>, <last_chunk_number>, 0, 32867, 16386,
/etc/passwd)'::superfile, b, 0);
```

PostgresPro

Superfile Compared to Disk and LO

Raw (direct) disk write 20 Gb file - 542 Mb/s:

```
sudo dd if=/dev/zero of=output.dat bs=500M count=40
40+0 records in
40+0 records out
20971520000 bytes (21 GB, 20 GiB) copied, 38.6741 s, 542 MB/s
```

Loading 20 Gb file into lo_object - 136 Mb/s

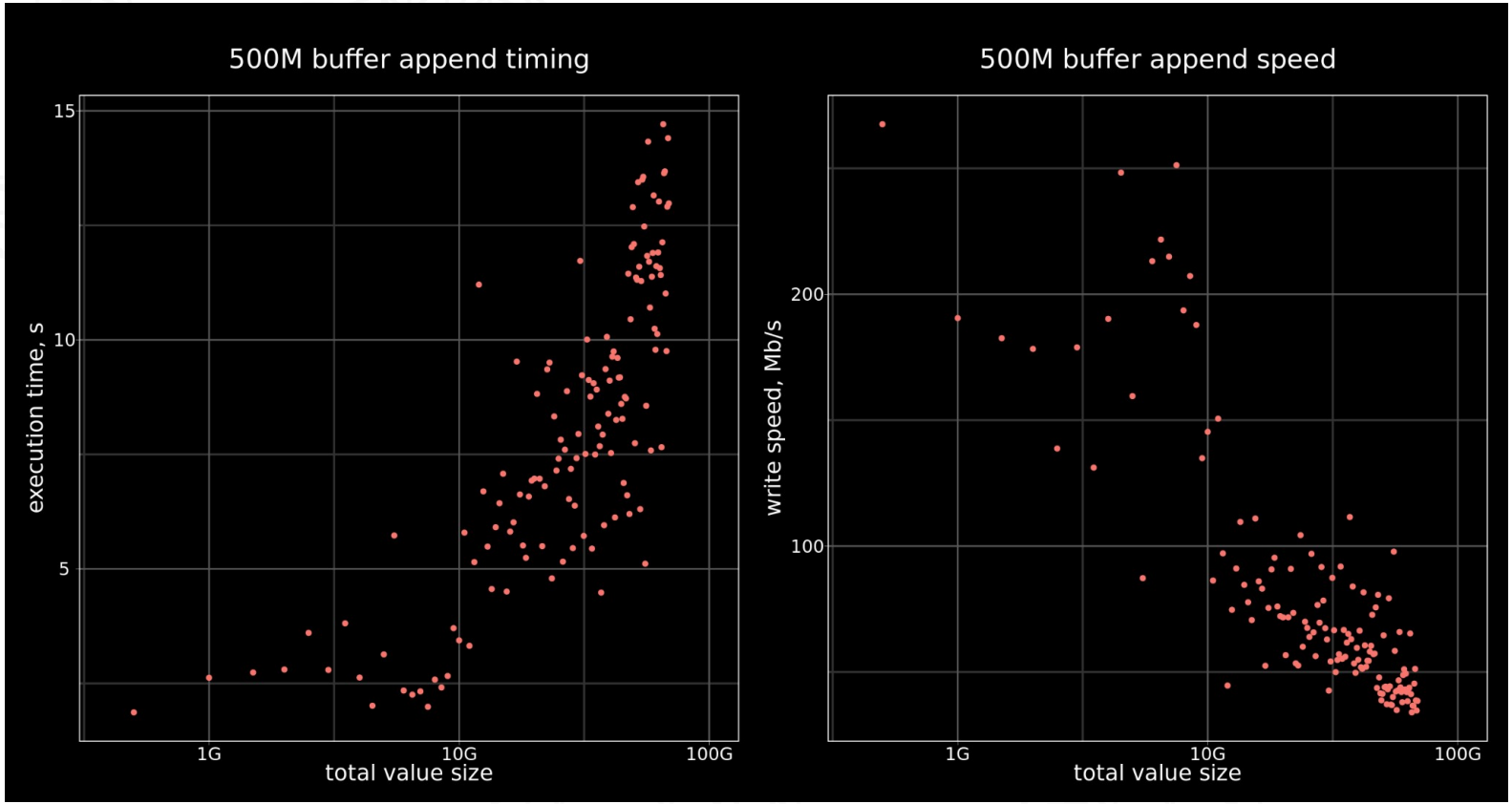
```
select lo_import('/usr/local/pgsql/bin/output.dat');
 lo_import
-----------
     49221
(1 row)
Time: 147553.673 ms (02:27.554)
```

Loading 20 Gb by Superfile interface - 133 Mb/s:

```
select avg(500000000/timing) from t_log_wr_res where v_size <= 20000000000;
        avg
--------------------
 135598.44814301998
(1 row)
Time: 1.247 ms
```

PostgresPro

Prototype Results - Insert Large Object



PostgresPro

Issues

Some issues encountered during prototype development:

- Writing large buffers. Insert time increases with value size because of heavy WAL traffic. Unlogged tables are not safe due to data loss because of unexpected server failures, but provide almost constant write speed;
- Reading large buffers - user must carefully control reading buffer size and buffers ranges. Could be implemented with iterators;
- Current TOAST functionality does not allow forced TOASTing - there is a tricky strategy to decide Toast or not data. Forced TOAST must be implemented in core;
- TOASTed valued OIDs are taken from global system OID pool which is limited to 4294967295 values. When system OID pool is depleted no object creations is possible and queries hang in endless waiting for free OID;

PostgresPro

TODO

- Research different storage scenarios;
- Research transactional behavior;
- Currently prototype does not allow update due to mostly used append operation for large files, it allows only insert and append, select (get) and delete functionality. Update operation for large objects could be very tricky;
- Modify core to enable forced TOAST (could be done by introducing toaster options);
- Implement iterators for more convenient access to stored data;
- Develop benchmarks
- Currently, Pluggable TOAST is not committed into Pg15, but we hope to commit it in Pg16

PostgresPro

Conclusions

- New approach is very promising, but there are many decisions to be made and a lot of work to do;
- Pluggable TOAST allows to introduce very powerful solutions to existing problems and bottlenecks of PostgreSQL;
- We'll be glad to introduce this extension in Pg16;
- We welcome developers to participate in development and testing of this cool new feature

PostgresPro

References

Our experiments:
New TOAST in TOWN. One Toast Fits All
http://www.sai.msu.su/~megera/postgres/talks/toast-pgcon-2022.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
Scaling JSONB - http://www.sai.msu.su/~megera/postgres/talks/jsonb-pgvision-2021.pdf
Slides of this talk http://www.sai.msu.su/~megera/postgres/talks/toast-pgcon-2022.pdf
Appendable Bytea TOAST
http://www.sai.msu.su/~megera/postgres/talks/bytea-pgconfonline-2021.pdf
Pluggable TOAST at Commitfest https://commitfest.postgresql.org/38/3490/
Jsonb_toaster @Github (check License.txt) :
https://github.com/postgrespro/postgres/tree/jsonb_toaster
Jsonb is ubiquitous and is continuously developing
JSON[B] Roadmap V2, Postgres Professional Webinar, Sep 17, 2020
JSON[B] Roadmap V3, Postgres Build 2020, Dec 8, 2020

PostgresPro

Thank you for your attention!

PostgresPro


ALL
YOU
NEED
POSTGRES
IS