TOAST, for breakfast and compression

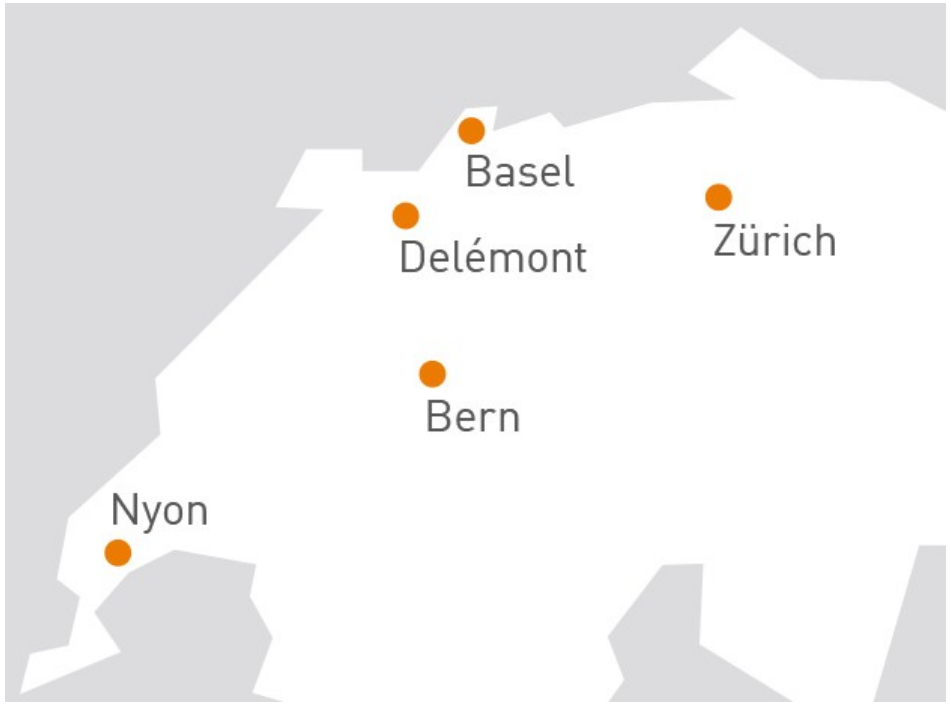
Who we are

The Company

- Founded in 2010
- More than 100 specialists
- Specialized in the Middleware Infrastructure
  - The invisible part of IT
- Customers in Switzerland and all over Europe

Our Offer

- Consulting
- Service Level Agreements (SLA)
- Trainings
- License Management



swiss safety center ISO 20000 certified system

swiss safety center ISO 27001 certified system

About me

Daniel Westermann

Technology Leader

Principal Consultant

+41 79 927 24 46

daniel.westermann@dbi-services.com

@westermanndanie

Daniel Westermann

This is TOAST for breakfast

… but what is TOAST for compression about?

... before we start to talk about TOAST ...


... we need to talk about blocks or pages!

Blocks / Pages

PostgreSQL (and other databases as well) stores all data in blocks / pages

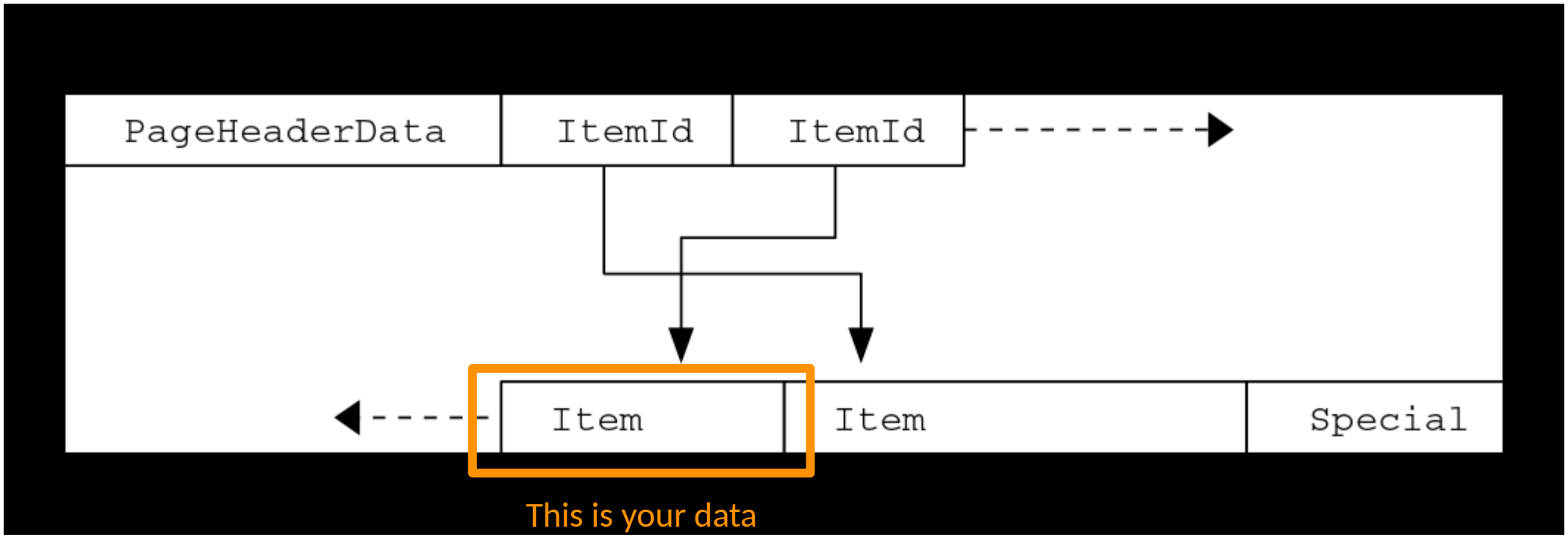
- Usually, a block is 8kB
  - This can be changed when PostgreSQL is compiled from source code

```
postgres@debian11pg:/home/postgres/postgresql/ [pg16] ./configure --help | grep block
  --with-blocksize=BLOCKSIZE
                          set table block size in kB [8]
  --with-wal-blocksize=BLOCKSIZE
                          set WAL block size in kB [8]
```

Blocks / Pages

How does a block/page look like?

> https://www.postgresql.org/docs/current/storage-page-layout.html



Items / Rows / Tuples

This leads us to the next question

- What exactly an item / a row / a tuple is made of?

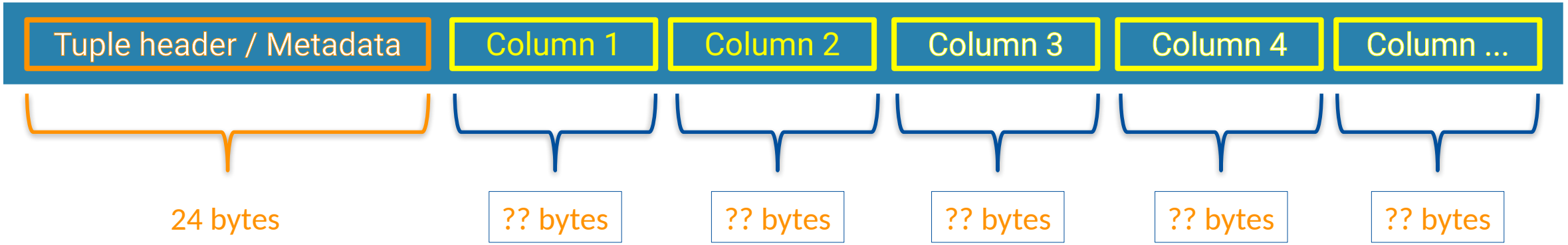
- What is in header data?
  - e.g. the transaction IDs which created / deleted the tuple and some more
  - https://www.postgresql.org/docs/current/storage-page-layout.html#PAGEHEADERDATA-TABLE

- The more important question here is: What defines the content of a column?
  - The **data types**, which are used for the columns

Sizing

This leads us to the next question

- How large can a column become?



- You need to know the data type to answer that!

Sizing

How do you know what a data type will consume on disk?

- The easy answer is: Check the documentation
  - https://www.postgresql.org/docs/current/datatype.html

Chapter 8. Data Types

Table of Contents

Table 8.2. Numeric Types

Name	Storage Size
smallint	2 bytes
integer	4 bytes
bigint	8 bytes
decimal	variable
numeric	variable
real	4 bytes
double precision	8 bytes
smallserial	2 bytes
serial	4 bytes
bigserial	8 bytes

Sizing

How do you know what a data type will consume on disk?

- You can also ask PostgreSQL directly about the size of a data type, or the complete tuple
  - This is an empty tuple or, in other words: the header data

```
postgres=# select pg_column_size ( row() );
 pg_column_size
----------------
             24
(1 row)
```

  - This is the size of a smallint

```
postgres=# select pg_column_size ( row(0::smallint) ) - 24;
 ?column?
----------
        2
(1 row)
```

Table 8.2. Numeric Types

Name	Storage Size
smallint	2 bytes
integer	4 bytes
bigint	8 bytes
decimal	variable
numeric	variable
real	4 bytes
double precision	8 bytes
smallserial	2 bytes
serial	4 bytes
bigserial	8 bytes

Sizing

When it becomes more tricky

- Not all data types have a fixed size
  - How does this look like?

```
postgres=# select pg_column_size ( row(1.1::numeric) ) - 24;
 ?column?
----------
        7
(1 row)
```

  - This can increase

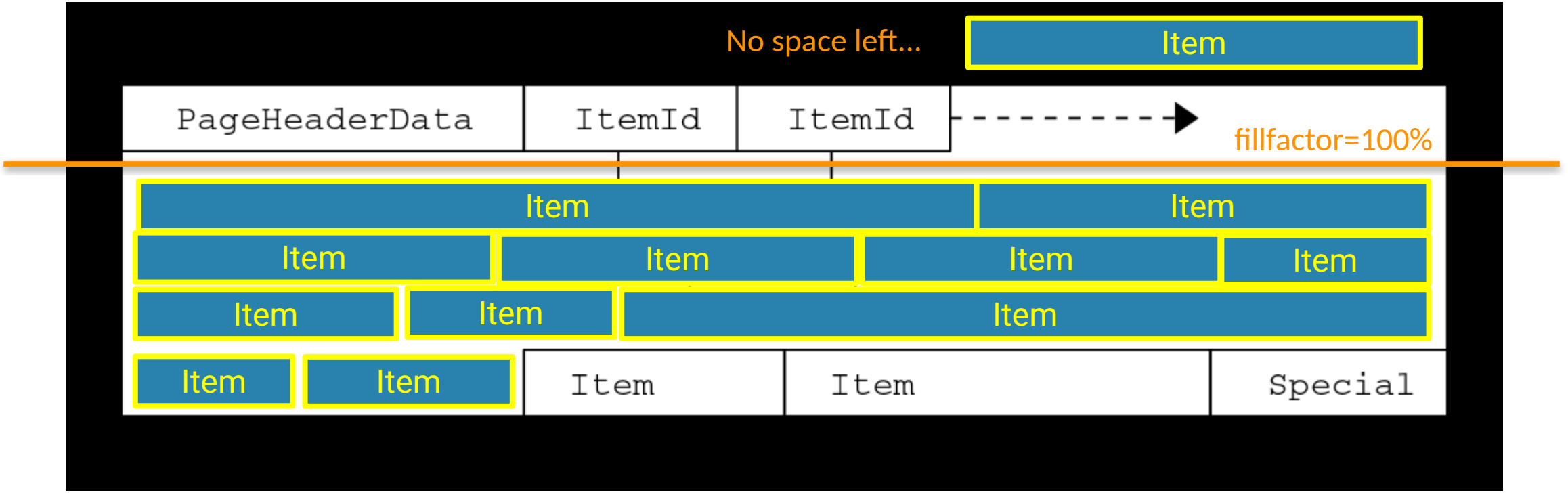
```
postgres=# select pg_column_size ( row(1111111.1111111111::numeric) )
                 - 24;
 ?column?
----------
       13
(1 row)
```

Table 8.2. Numeric Types

Name	Storage Size
smallint	2 bytes
integer	4 bytes
bigint	8 bytes
decimal	variable
numeric	variable
real	4 bytes
double precision	8 bytes
smallserial	2 bytes
serial	4 bytes
bigserial	8 bytes

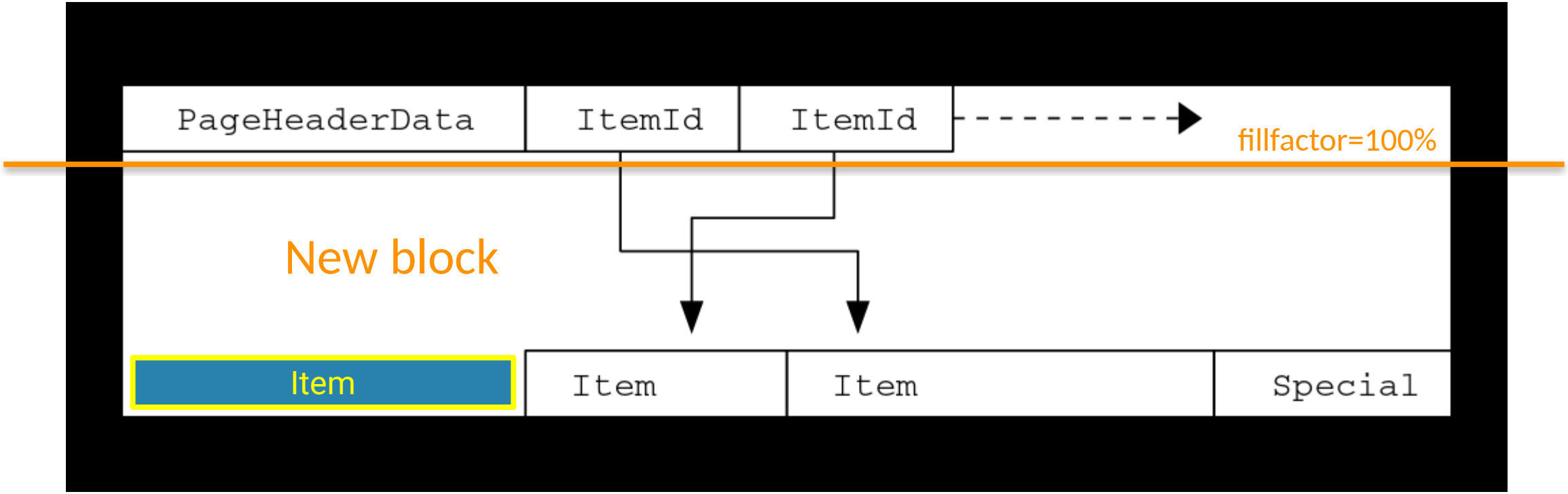
Tuples in blocks

Multiple items/rows/tuples go into one block



Tuples in blocks

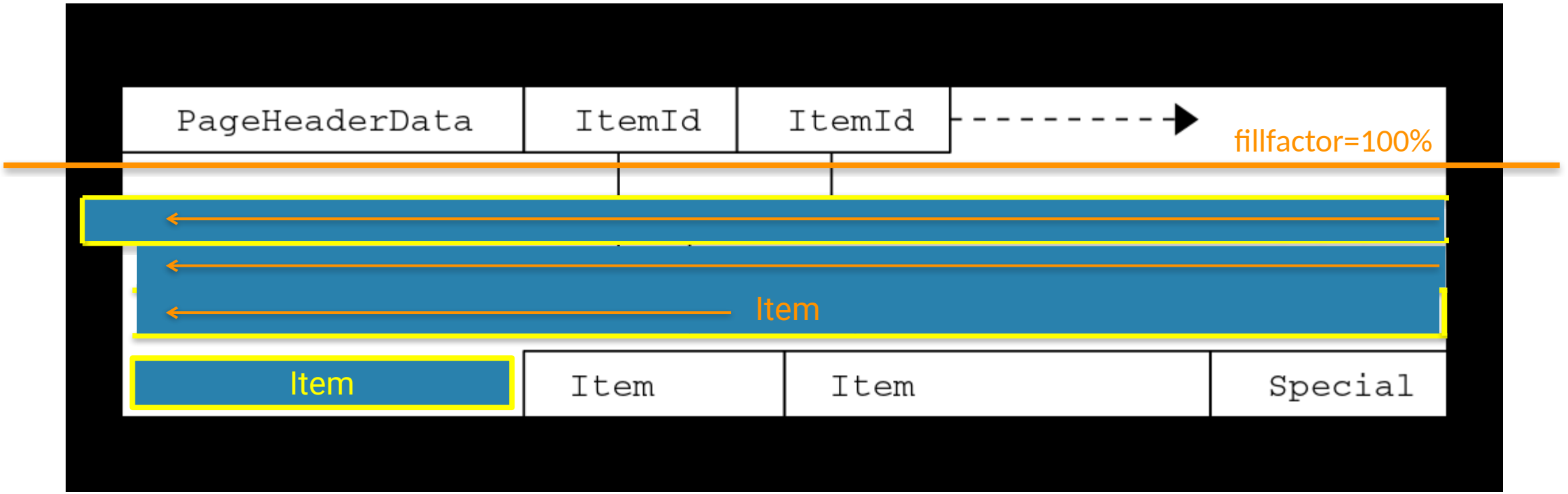
Multiple items/rows/tuples go into one block



Tuples in blocks

The more complicated case

> An item arrives which does not fit into the block at all



Tuples in blocks

The more complicated case

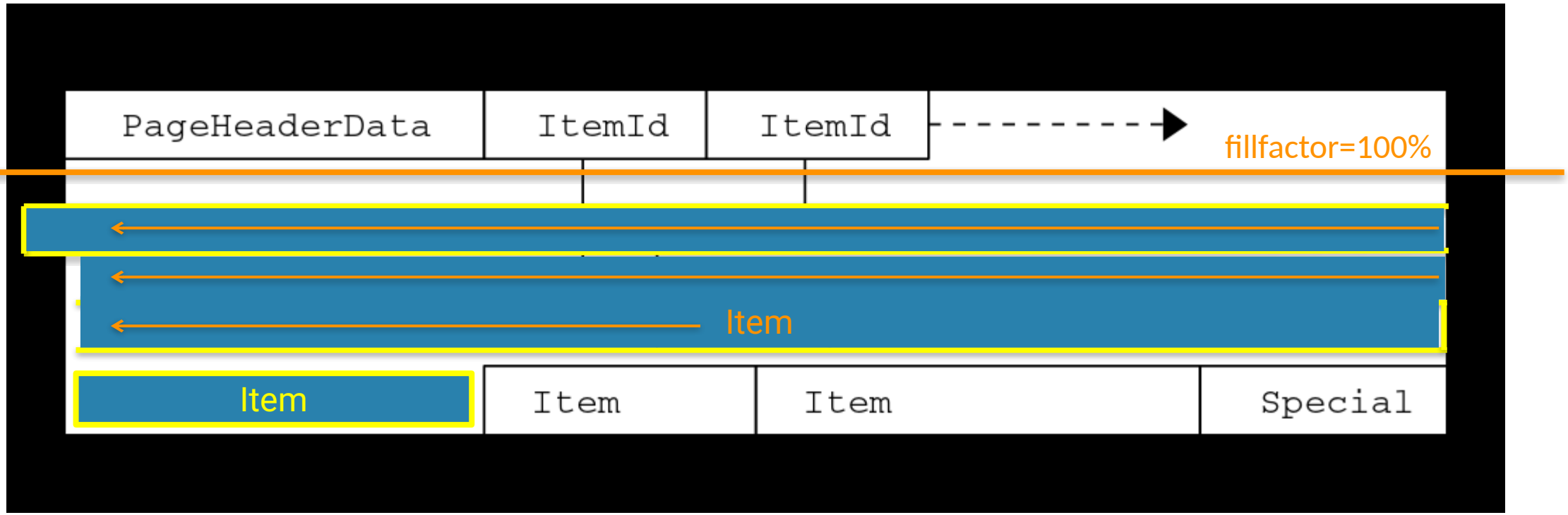
- PostgreSQL does not allow a row/tuple to span multiple blocks
  - There is no concept such as row chaining in Oracle
  - This comes with advantages
    - Large data only needs to be pulled out when required, the main table stays small
- What happens is
  - Depending on the data type (more on that later)
    - Compress the row
    - Maybe store it outside the block
    - Put a pointer in the row/tuple to locate the out of line storage
  - TOAST
    - **T**he **O**versized-**A**ttribute **S**torage **T**echnique
      - or
    - **the best thing since sliced bread**

Say goodbye to breakfast

Tuples in blocks

In fact, this information so far is not accurate

- An item arrives which does not fit into the block at all



Triggering the toaster

In fact, this information is not accurate

- PostgreSQL will try to store at least 4 tuples per block
  - src/include/access/heaptoast.h

```
#define TOAST_TUPLES_PER_PAGE   4
```

- This means, with a standard block size of 8kB
  - 2000 bytes will trigger the toaster (excluding the header)

- This can be controlled per table

```
postgres=# alter table t set ( toast_tuple_target = N );
```

Triggering the toaster

In prosa, the algorithm is

- PostgreSQL will try to store at least 4 tuples per block
  - If the new row exceeds one fourth of the block (excluding the header)
  - TOAST will kick in
    - Loop through all the attributes/columns, which have either **external or extended strategies?**

External and extended?

Storage strategies

There are four different strategies for TOASTing

Strategy	Effect
PLAIN	Prevents either compression or out-of-line storage; the only option for non-TOAST-able data types.
EXTENDED	Allows both compression and out-of-line storage; first compression, then out-of-line.
EXTERNAL	Allows out-of-line storage but not compression.
MAIN	Allows compression but not out-of-line storage.

> Remember: The strategy is on the data type, not the tuple/row!

Storage strategies

How do I know the strategy of a data type when it comes to toasting?

```
postgres=# create table t ( a int, images bytea );
CREATE TABLE
```

- Metadata of columns is stored in pg_attribute

```
postgres=# select attname, atttypid::regtype,
                  case attstorage when 'p' then 'plain'
                                  when 'e' then 'external'
                                  when 'm' then 'main'
                                  when 'x' then 'extended'
                  end AS strategy
             from pg_attribute
            where attrelid = 't'::regclass and attnum > 0;
 attname | atttypid | strategy
---------+----------+----------
 a       | integer  | plain
 images  | bytea    | extended
```

Storage strategies

If you know in advance, that data is not compressible

- e.g. images, pdfs, whatever
  - Skip compression by default, by changing the strategy to external

```
postgres=# alter table t alter column images set storage external;
ALTER TABLE
postgres=# select attname, atttypid::regtype,
                  case attstorage when 'p' then 'plain'
                       when 'e' then 'external'
                       when 'm' then 'main'
                       when 'x' then 'extended'
                  end AS strategy
             from pg_attribute
            where attrelid = 't'::regclass and attnum > 0;
 attname | atttypid | strategy
---------+----------+----------
 a       | integer  | plain
 images  | bytea    | external
```

Triggering the toaster

In prosa, the algorithm is

- PostgreSQL will try to store at least 4 tuples per block
  - If the new row exceeds one fourth of the block (excluding the header)
  - TOAST will kick in
    - Loop through all the attributes/columns, which have either external or extended strategies?
      - Start with the longest attribute
    - **Compress** extended attributes


Compression?
How?

TOAST compression

The default TOAST compression

```
postgres=# \dconfig default_toast_compression
 List of configuration parameters
         Parameter          | Value
----------------------------+-------
 default_toast_compression | pglz
(1 row)
```

- \dconfig is a new PostgreSQL 15 feature to list parameters
  - You may also use wild cards and "+" to show more information

```
postgres=# \dconfig+ *toast_compression*
                    List of configuration parameters
         Parameter          | Value | Type | Context | Access privileges
----------------------------+-------+------+---------+-------------------
 default_toast_compression | pglz  | enum | user    |
(1 row)
```

TOAST compression

How \dconfig came in and how the discussion went

- https://www.postgresql.org/message-id/flat/3118455.1649267333%40sss.pgh.pa.us

How about a psql backslash command to show GUCs?

Lists:pgsql-hackers

From: Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>
To: pgsql-hackers(at)lists(dot)postgresql(dot)org
Cc: Mark Dilger <mark(dot)dilger(at)enterprisedb(dot)com>
Subject: How about a psql backslash command to show GUCs?
Date: 2022-04-06 17:48:53
Message-ID:3118455.1649267333@sss.pgh.pa.us
Views: Raw Message | Whole Thread | Download mbox | Resend email
Lists: pgsql-hackers

It's not difficult to get psql to show you the current value
of a single GUC --- "SHOW" does that fine, and it has tab
completion support for the GUC name. However, I very often
find myself resorting to the much more tedious

```
select * from pg_settings where name like '%foo%';
```

TOAST compression

Before PostgreSQL 14 there is only pglz

- This is a build-in compression algorithm

Starting with PostgreSQL 14 there is more choice

```
postgres=# alter system set default_toast_compression = [TAB][TAB]
DEFAULT  lz4      pglz
```

- https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)
  - Focus on compression and decompression speed
  - Good tradeoff between compression speed and compression ratio
- lz4 needs to be enabled

```
postgres@debian11pg:/home/postgres/postgresql/ [pgdev] ./configure --help | grep -i lz4
  --with-lz4              build with LZ4 support
```

TOAST compression

The compression method can be a per column setting

```
postgres=# alter table t alter column a set compression [TAB][TAB]
DEFAULT  LZ4      PGLZ
```

Triggering the toaster

In prosa, the algorithm is

- PostgreSQL will try to store at least 4 tuples per block
  - If the new row exceeds one fourth of the block (excluding the header)
  - TOAST will kick in
    - Loop through all the attributes/columns, which have either external or extended strategies?
      - Start with the longest attribute
    - Compress extended attributes
      - If the result is still larger than ¼ of the block/page, **move it to the TOAST table**


TOAST tables?

TOAST tables

As soon as there is a data type which potentially can be moved out-of-line

- PostgreSQL automatically will create a **TOAST table**

```
postgres=# create table t ( a int );
CREATE TABLE
postgres=# create table tt ( b text );
CREATE TABLE
postgres=# select relname,reltoastrelid::regclass
             from pg_class
             where relname in ('t','tt');

 relname |      reltoastrelid
---------+-------------------------
 t       | -
 tt      | pg_toast.pg_toast_16401
(2 rows)
```

TOAST tables

TOAST tables are in a special schema, not visible by default

```
postgres=# \dn
      List of schemas
  Name  |       Owner
--------+--------------------
 public | pg_database_owner
(1 row)

postgres=# \dnS
            List of schemas
        Name         |       Owner
---------------------+--------------------
 information_schema | postgres
 pg_catalog         | postgres
 pg_toast           | postgres
 public             | pg_database_owner
(4 rows)
```

TOAST tables

How do TOAST tables look like?

```
postgres=# \d pg_toast.pg_toast_16401
TOAST table "pg_toast.pg_toast_16401"
   Column   |  Type
------------+---------
 chunk_id   | oid
 chunk_seq  | integer
 chunk_data | bytea
Owning table: "public.tt"
Indexes:
    "pg_toast_16401_index" PRIMARY KEY, btree (chunk_id, chunk_seq)
```

- Data is sliced into (compressed) chunks of data and indexed
- The index is always used to access the data

TOAST tables

In prosa, the algorithm is

- PostgreSQL will try to store at least 4 tuples per block
  - If the new row exceeds one fourth of the block (excluding the header)
  - TOAST will kick in
    - Loop through all the attributes/columns, which have either external or extended strategies?
      - Start with the longest attribute
    - Compress extended attributes
      - If the result is still larger than ¼ of the block/page, move it to the TOAST table
    - External attributes are handled the same, but no compression
    - If the row / tuple still does not fit, take next attribute and proceed the same way
    - If all attributes have been compressed but the row/tuple is still too large
      - Move all attributes to the TOAST table

TOAST demo

Demo setup

- A simple dummy table and the corresponding TOAST table

```
postgres=# create table toast_demo ( id int primary key, content text );
CREATE TABLE
postgreska
  relname   |      reltoastrelid
------------+-------------------------
 toast_demo | pg_toast.pg_toast_16411
(1 row)
```

TOAST demo

Demo setup

- First simple insert

```
postgres=# insert into toast_demo values (1, repeat('x',10000));
INSERT 0 1
```

- Will this trigger the TOASTER?

```
postgres=# select count(*) from :reltoastrelid;
 count
-------
     0
(1 row)
```

- Why?
  - Compression kicked in and all fits into the block without the need to move out of line

TOAST demo

Demo setup

- Generating a random string

```
postgres=# select string_agg (md5(random()::text),'') from generate_series(1,5);
                                  string_agg

--------------------------------------------------------------------------------

895d7132addc558026ff32c75875a03885634aac6255372a68ba0c7ad783b230e074778ad67488deddfa49719
428593862c079a54e29c09b4a03a8fa97bfa6dce793cce663cff2a1bb1ec3a56aff2b20
(1 row)

postgres=# with dummy_string as
             ( select string_agg (md5(random()::text),'') as dummy
                 from generate_series(1,5000) )
           insert into toast_demo
           select 2, dummy_string.dummy from dummy_string;
INSERT 0 1
```

TOAST demo

Demo setup

- We got 81 compressed chunks in the TOAST table

```
postgres=# select count(*) from :reltoastrelid;
 count
-------
    81
(1 row)

postgres=# select * from :reltoastrelid limit 3;
 chunk_id | chunk_seq |  chunk_data
 ----------------------------------------------------
    16418 |         0 | \x336439343438356...534613>
    16418 |         1 | \x366531363465363...863303>
    16418 |         2 | \x633362346636613...461363>
(3 rows)
```

TOAST demo

Demo setup

- Be carefull with "**select ***" when you have TOASTed data
- Let's do something weird and load the PostgreSQL documentation

```
postgres=# \! wget
https://www.postgresql.org/files/documentation/pdf/14/postgresql-14-A4.pdf
postgres=# \! ls -lha postgresql-14-A4.pdf
-rw-r--r-- 1 postgres postgres 14M Aug 11 15:26 postgresql-14-A4.pdf
postgres=# create table toast_demo2 ( id int, doc bytea );
CREATE TABLE
postgres=# select pg_read_binary_file('/home/postgres/postgresql-14-A4.pdf')
                  as file; \gset
postgres=# insert into toast_demo2 select i, :'file' from generate_series(1,50) i;
INSERT 0 50
postgres=# select pg_size_pretty(pg_relation_size('toast_demo2'));
 pg_size_pretty
----------------
 8192 bytes
(1 row)
```

TOAST demo

Demo setup

- Be carefull with "**select ***" when you have TOASTed data

```
postgres=# select pg_size_pretty(pg_total_relation_size('toast_demo2'));
 pg_size_pretty
----------------
 475 MB
(1 row)
postgres=# \timing on
Timing is on.
```

TOAST demo

Demo setup

- Be carefull with "**select ***" when you have TOASTed data

```
postgres=# select id from toast_demo2 where id < 5;
 id
----
  1
  2
  3
  4
(4 rows)

Time: 6.402 ms
```

- This is fast, as the TOASTed valued are not touched

TOAST demo

Demo setup

> Be carefull with "**select ***" when you have TOASTed data

```
postgres=# select * from toast_demo2 where id < 5;
 id | doc




----|--------------------------------
  1 | \x255...1746f722028446f63426f6>
  2 | \x255...1746f722028446f63426f6>
  3 | \x255...1746f722028446f63426f6>
  4 | \x255...1746f722028446f63426f6>
(4 rows)

Time: 587.352 ms
postgres=#
```

If you don't need the TOASTed data, don't touch it!

TOAST demo

Explain analyze is misleading

- Completely discards the output

```
postgres=# explain (analyze,buffers) select * from toast_demo2;
                                                 QUERY PLAN

-------------------------------------------------------------------------------------------------------
---------------
 Seq Scan on toast_demo2  (cost=0.00..1.50 rows=50 width=22) (actual time=0.016..0.024
rows=50 loops=1)
   Buffers: shared hit=1
 Planning Time: 0.053 ms
 Execution Time: 0.054 ms
(4 rows)

Time: 4.772 ms
```


What are the plans for the future?

TOAST, the future?

https://postgrespro.com/blog/pgsql/5969559

New TOAST in town: the “pluggable TOAST API” concept and what it means for the community

- Work is going on for a type aware TOASTER
- The goal is to provide a TOAST API so additional TOASTERS can be implemented as extensions

TOAST, the future?

https://commitfest.postgresql.org/39/3490/

Pluggable toaster

From: Teodor Sigaev <teodor(at)sigaev(dot)ru>
To: pgsql-hackers <pgsql-hackers(at)postgreSQL(dot)org>
Subject: Pluggable toaster
Date: 2021-12-30 16:40:09
Message-ID: 224711f9-83b7-a307-b17f-4457ab73aa0a@sigaev.ru
Views: Raw Message | Whole Thread | Download mbox | Resend email
Thread: 2021-12-30 16:40:09 from Teodor Sigaev <teodor(at)sigaev(dot)ru>
Lists: pgsql-hackers

```
Hi!

We are working on custom toaster for JSONB [1], because current TOAST is
universal for any data type and because of that it has some disadvantages:
    - "one toast fits all"  may be not the best solution for particular
      type or/and use cases
    - it doesn't know the internal structure of data type, so it  cannot
      choose an optimal toast strategy
    - it can't  share common parts between different rows and even
      versions of rows
```

What is the real-world issue these patches are trying to solve


JSONB

A few words about JSONB

JSONB

- Introduced in PostgreSQL 9.4, 2014
  - Successor of the JSON data type, which was based on text
  - Binary storage of JSON documents
    - Comes with indexing support (GIN)
  - Made PostgreSQL popular for use with unstructured data
  - One database for both, structured and unstructured data
  - But
    - What is the TOAST strategy for jsonb?
      - Extended
    - JSON documents easily can become quite large
      - You know what happens around 2kB of data
        - The TOASTER will kick in

A few words about JSONB

What happens if you need to update a JSON document?

- Fetch the whole value
- Update the document where required
- Rewrite the whole document
  - Once more, compress, then store out-of-line

Does that sound efficient?

- TOAST is great, but you need to be aware of the current limitations

TOAST, final words

https://de.wikipedia.org/wiki/Toastbrot

Im deutschsprachigen Raum ist die *Bähschnitte* (von *bähen*: leicht rösten)[6] bereits im Mittelalter bekannt.[7] In Deutschland gibt es das typische helle Toastbrot erst seit den 1950er-Jahren. Toaster gab es zwar bereits seit etwa 1910, „Brotröster“ genannt, sie wurden aber für das übliche Graubrot verwendet. „Mit Hilfe einer Vermarktungsorganisation amerikanischer Weizenproduzenten und deutscher Toastgerätehersteller wurde Toast als gehobene und ‚praktische‘ Brotmahlzeit nach dem Zweiten Weltkrieg in Deutschland mit einem hohen Werbeaufwand populär gemacht.“[8] 1963 wurde von Bäckereien eine *Arbeitsgemeinschaft zur Förderung des Toastbrotverzehrs* gegründet, die die Marke Golden Toast auf den Markt brachte. Seit den 1980er-Jahren gibt es in Deutschland auch dunkleren Vollkorntoast.[9] Zur Popularität des Toasts als Imbiss trug wesentlich die Erfindung des Toast Hawaii durch den Fernsehkoch Clemens Wilmenrod Mitte der 1950er-Jahre bei.

Any questions?

Please do ask!

We would love to boost your IT-Infrastructure

How about you?