Let's (D)Trace Postgres

tracing the madness

FUDO SECURITY

Adam Wołk

a.wolk@fudosecurity.com

awolk@openbsd.org

https://blog.tintagel.pl

@mulander

FUDO SECURITY

Mariusz Zaborski

FreeBSD

m.zaborski@fudosecurity.com

oshogbo@FreeBSD.org

https://oshogbo.vexillium.org

@oshogbovx

DTrace

Did you use DTrace?

What is DTrace?

- A dynamic tracing framework for software
  - D scripts
  - A subset of C
  - A subset that is hard to get hurt with
- There is no performance penalty when you don’t use it
- There is no performance penalty for the whole system

What can we trace?

- When and which function is being called
- Functions arguments
- The frequency of function calls
- The return code of functions
- The function call stack
- A whole lot more….

DTrace naming

Probe	something to trace
Provider	DTrace module that provides information about system
Module	software module (**kernel, libc, postgres**)
Function	A function (**open, openat**)
Predicate	Filtering DTrace probes
Action	A D script

DTrace naming

- **Probe**: syscall::write:entry
  - **Provider:** syscall
  - **Module:** none
  - **Function:** write
  - **Name**: entry
- **Probe**: syscall::write:return / arg1 > 10 /
  - **Provider:** syscall
  - **Module:** none
  - **Function:** write
  - **Predicate:** return value greater then 10
  - **Name**: return

Finding Probes

More than 50000 in FreeBSD.

dtrace -l

```
   ID   PROVIDER            MODULE                      FUNCTION NAME
74282        vfs               vop              vop_islocked entry
74283        vfs               vop              vop_islocked return
74284        vfs               vop                vop_lookup entry
74285        vfs               vop                vop_lookup return
74070        mac            kernel                       policy modevent
74071        mac            kernel                       policy register
74072        mac            kernel                       policy unregister
```

Finding Probes

dtrace -l -P syscall

```
   ID   PROVIDER            MODULE                          FUNCTION NAME
75237     syscall          freebsd32                        syscall entry
75238     syscall          freebsd32                        syscall return
75239     syscall          freebsd32                           exit entry
75240     syscall          freebsd32                           exit return
```

Finding Probes

dtrace -lv -f syscall:freebsd:read

```
   ID    PROVIDER            MODULE                          FUNCTION NAME
76353    syscall             freebsd                             read entry

  Argument Types
    args[0]: int
    args[1]: userland void *
    args[2]: size_t
```

Example

```
void
foo() { /* very long functiiton */ }

void
bar() { /* very long functiiton */ }

int
main(void)
{

        for (;;) {
                if (rand() % 2 == 1) {
                        foo();
                } else {
                        bar();
                }
        }
        return (0);
}
```

Example

```
void
foo() { /* very long functiiton */ }

void
bar() { /* very long functiiton */ }

int
main(void)
{

        for (;;) {
                if (rand() % 2 == 1) {
                        foo();
                } else {
                        bar();
                }
        }
        return (0);
}
```

```
# dtrace -n 'pid$target:::entry' -c ./a.out

[some information about _start and _main]
 11  80334                    rand:entry
 11  78659                    foo:entry
 11  80334                    rand:entry
 11  78659                    foo:entry
 11  80334                    rand:entry
 11  78658                    bar:entry
 11  80334                    rand:entry
 11  78659                    foo:entry
 11  80334                    rand:entry
 11  78658                    bar:entry
 11  80334                    rand:entry
```

Example

```
void
foo() { /* very long functiiton */ }

void
bar() { /* very long functiiton */ }

int
main(void)
{

        for (;;) {
                if (rand() % 2 == 1) {
                        foo();
                } else {
                        bar();
                }
        }
        return (0);
}
```

```
pid$target:::entry {
  @[probefunc] = count();
}
```

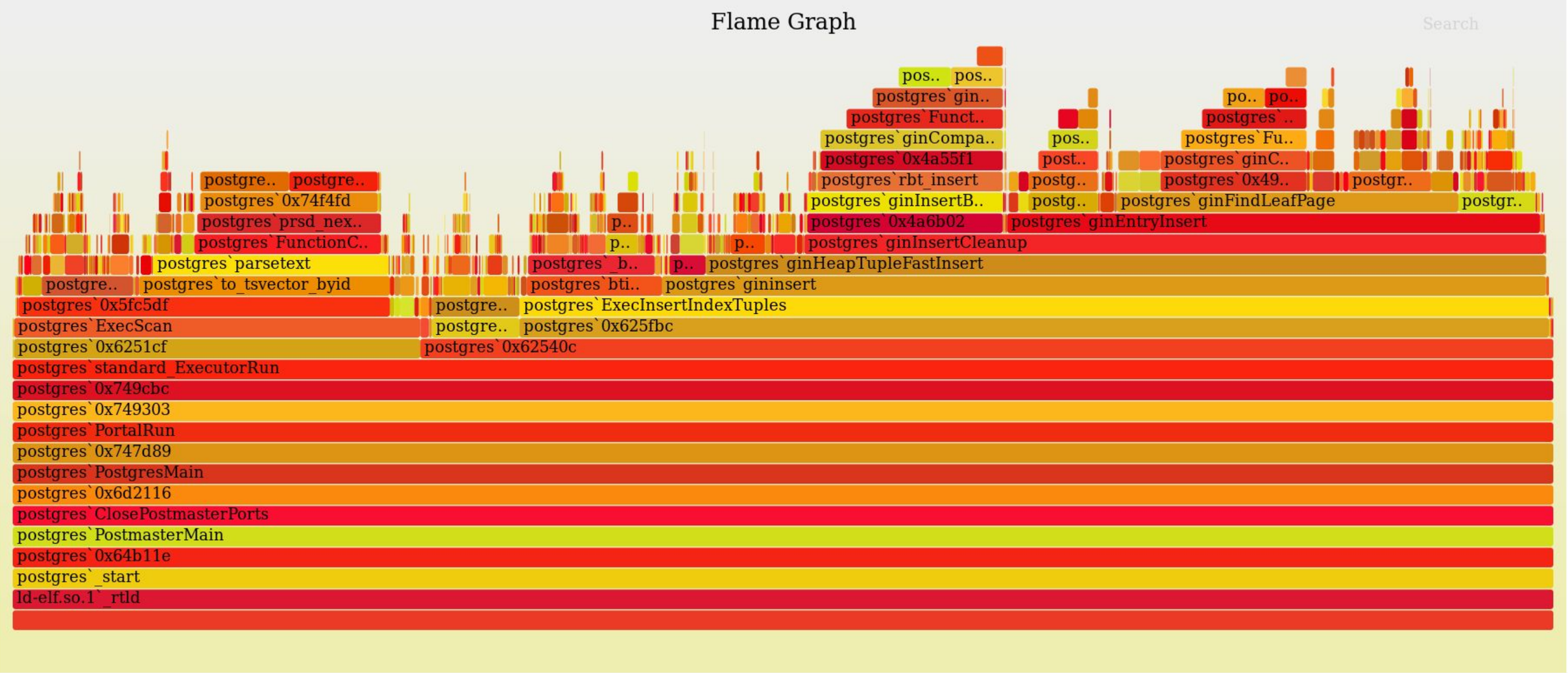
```
foo                                          370468
bar                                          370894
rand                                         741362
```

Example

```
void
foo() { /* very long functiiton */ }

void
bar() { /* very long functiiton */ }

int
main(void)
{

        for (;;) {
                if (rand() % 2 == 1) {
                        foo();
                } else {
                        bar();
                }
        }
        return (0);
}
```

```
pid$target:::entry {
  @[ustack()] = count();
}

a.out`foo+0x1
0x800000000001
                386613

a.out`bar+0x1
0x800000000001
                387132

libc.so.7`rand+0x1
0x800000000001
                773745
```

Example - flamegraph



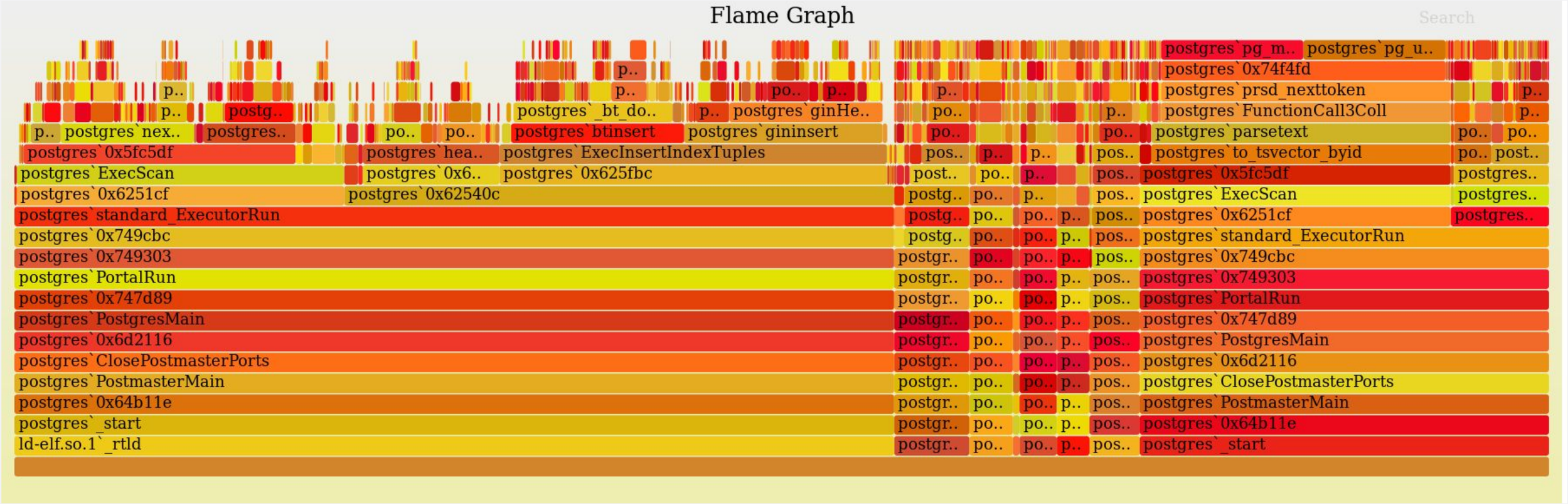
Example - flamegraph

```
$ git clone https://github.com/brendangregg/FlameGraph.git
$ perl stackcollapse.pl dtrace.out > stack.out
$ perl flamegraph.pl stack.out > stack.svg
```

Some hints which we learned the hard way:

- stackframes
- ustackframes/jstackframes

Example - flamegraph



Example

```c
struct test {
        int a;
        int b;
};

#include "test.h"

void
foo(struct test *t) { /* long function */ }

int
main(void)
{
        struct test t;

        t.a = 123;
        t.b = 999;
        foo(&t);

        return (0);
}
```

```
#include "test.h"

pid$target::foo:entry
{
  x = (struct test *)copyin(arg0,
        sizeof(struct test));
  printf("%d %d", x->a, x->b);
}
```

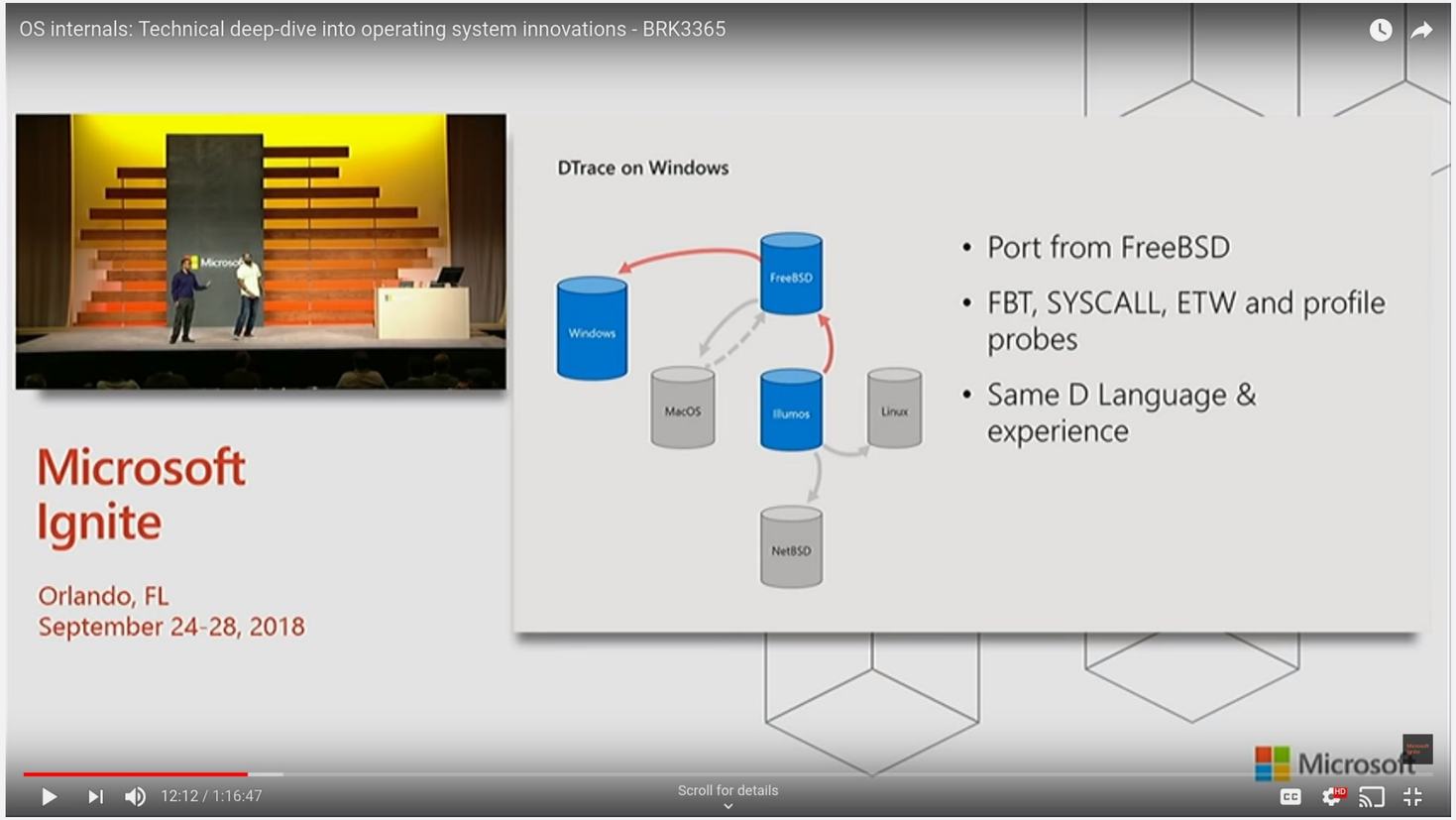
```
foo:entry 123 999
```

Where I can use it?

- FreeBSD
- MacOS
- NetBSD (kinda)
- Solaris
- Dtrace for Oracle Linux
- DTrace4linux

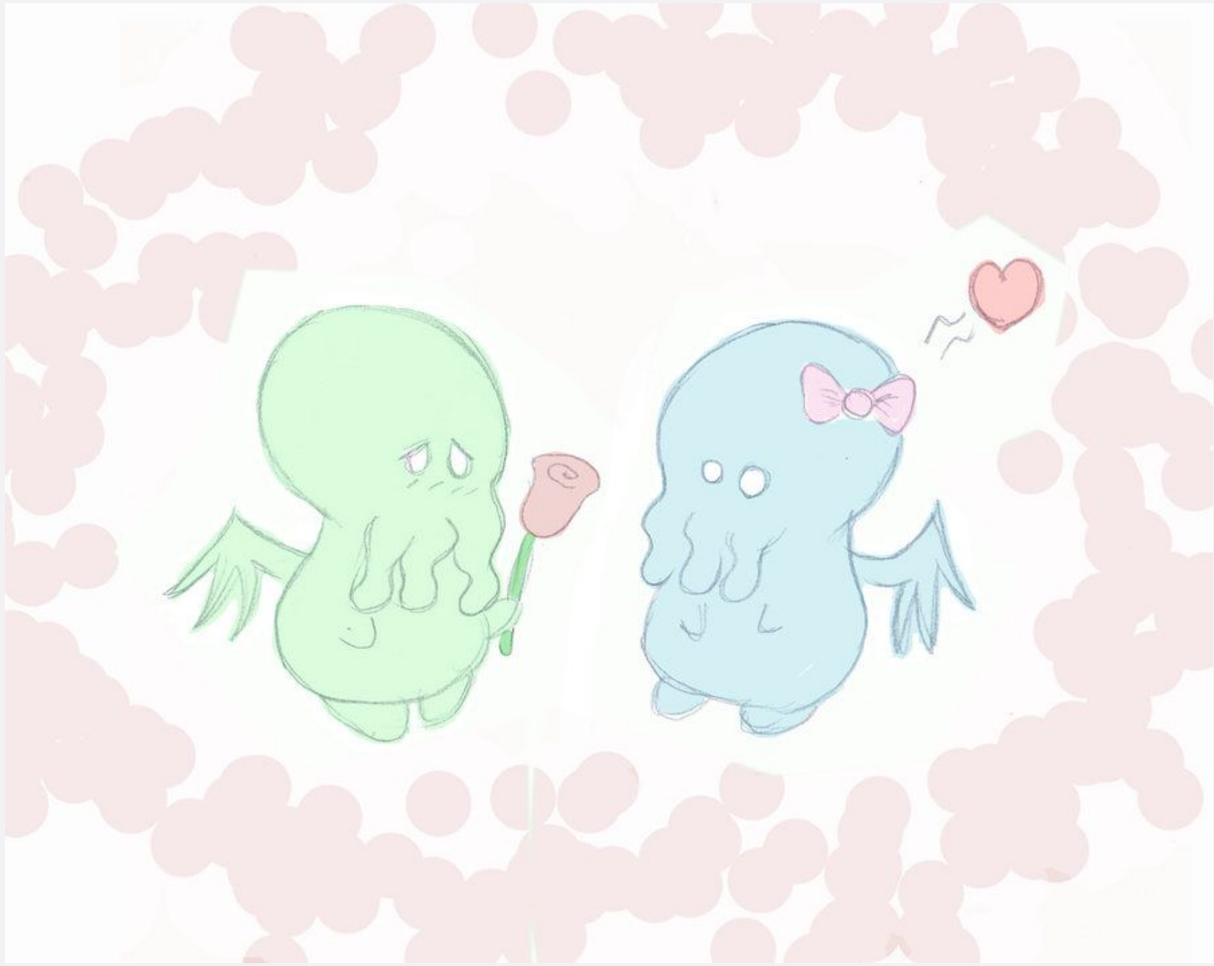
Where I can use it?

- FreeBSD
- MacOS
- NetBSD (kinda)
- Solaris
- Dtrace for Oracle Linux
- DTrace4linux
- Windows ?



DTrace&PostgreSQL

DTrace&PostgreSQL



DTrace&PostgreSQL

- https://wiki.postgresql.org/wiki/DTrace
- You need to rebuild it! (-enable-dtrace)
- Additional provider: postgresql
  - transaction-start
  - transaction-commit
  - transaction-abort
  - query-parse-done
  - and more (51)....

DTrace&PostgreSQL

```
postgresql$1:::transaction-abort
{
        @abort["Abort"] = count();
}
```


CALL of CTHULHU
The Official Video Game
Depths of Madness

GIN Indexes

What are indexes of type GIN?

- **G**eneralized **I**nverted I**n**dex
- used mostly for FTS (full-text search) but also for **json/jsonb** indexes
- Indexed items are composite values that contain zero or more keys
  - integer array **<-** integers
  - text **<-** lexemes (tsvector)
- optimized for cases where items contain many keys and the same key values appear in many different items

GIN Structure

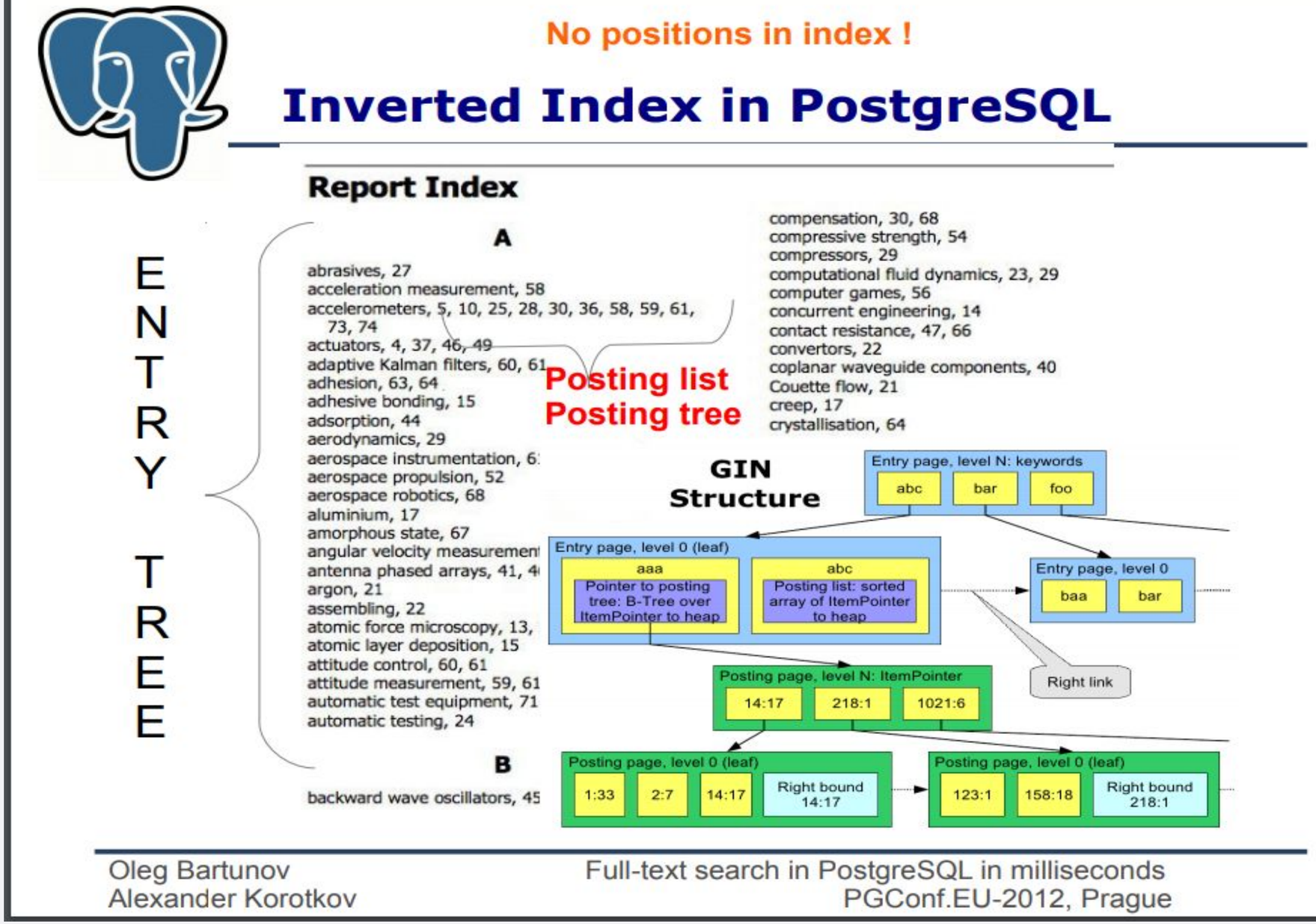
PGConf.EU-2012

Prague,

Oleg Bartunov

Alexander Korotkov

PostgreSQL GIN implementation authors

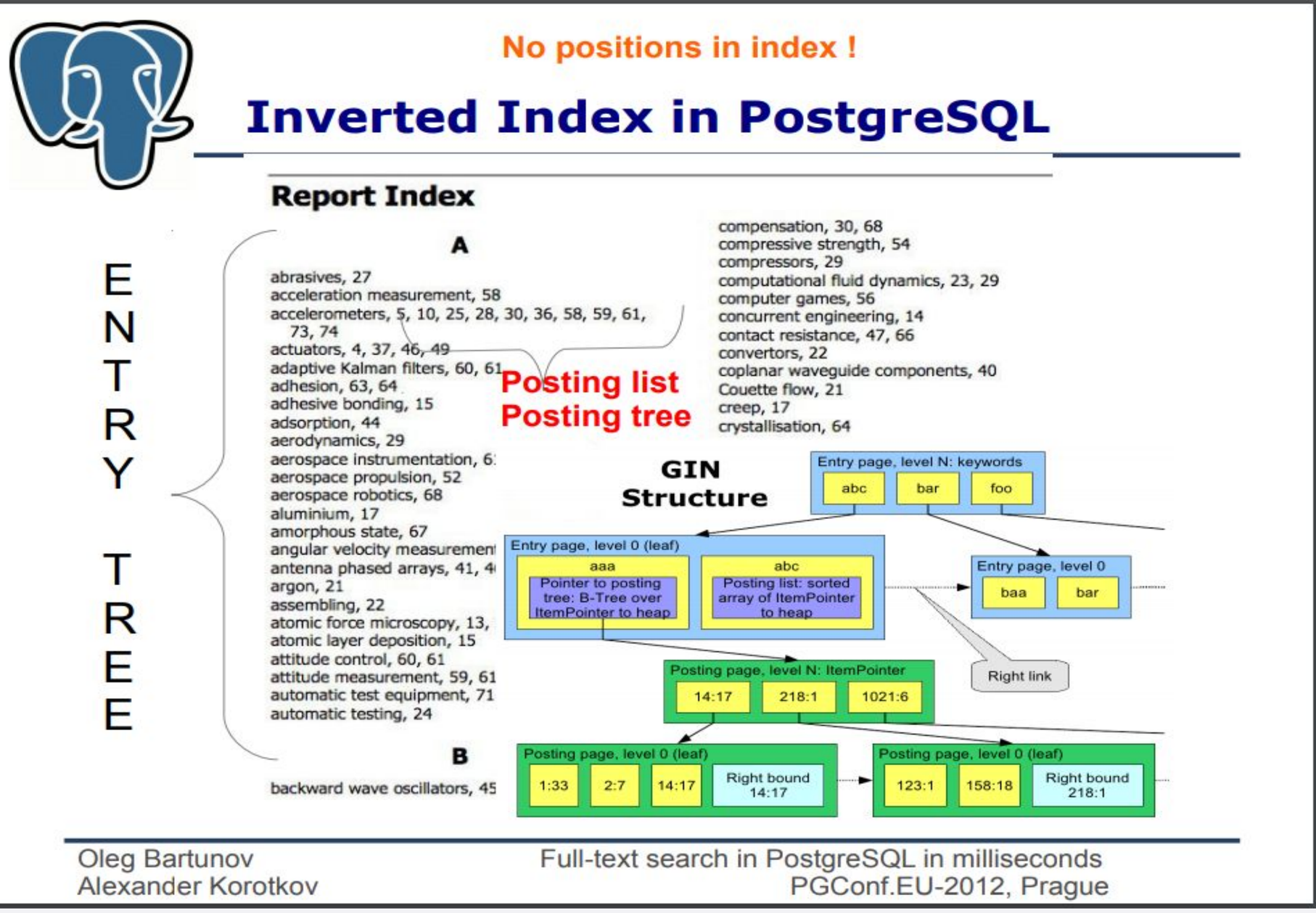


GIN Structure

Metapage

- control information
- index version
- statistics

Points to the **entry tree**



GIN Structure

entry tree

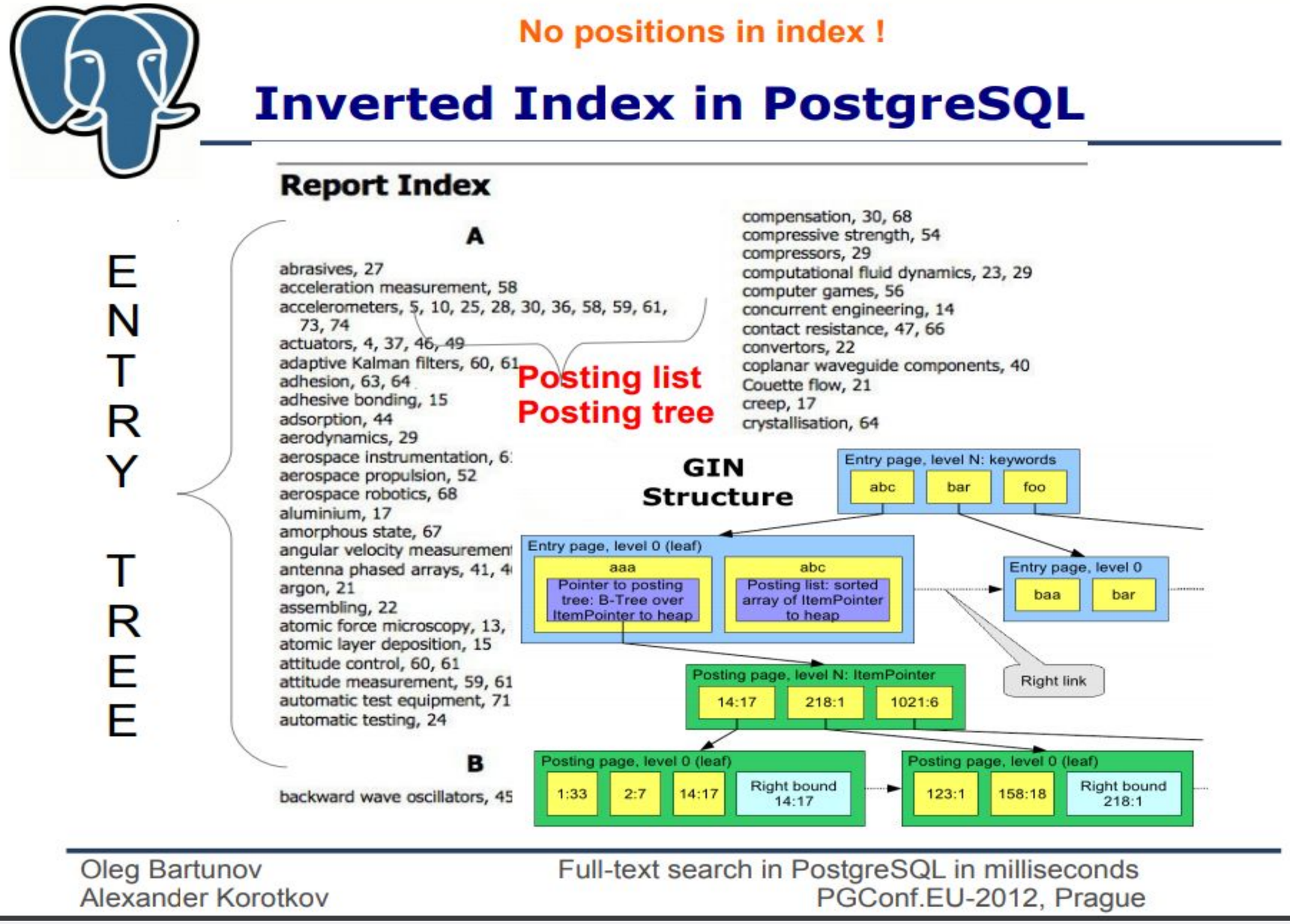
B-tree of key entries

possibly containing a

posting list

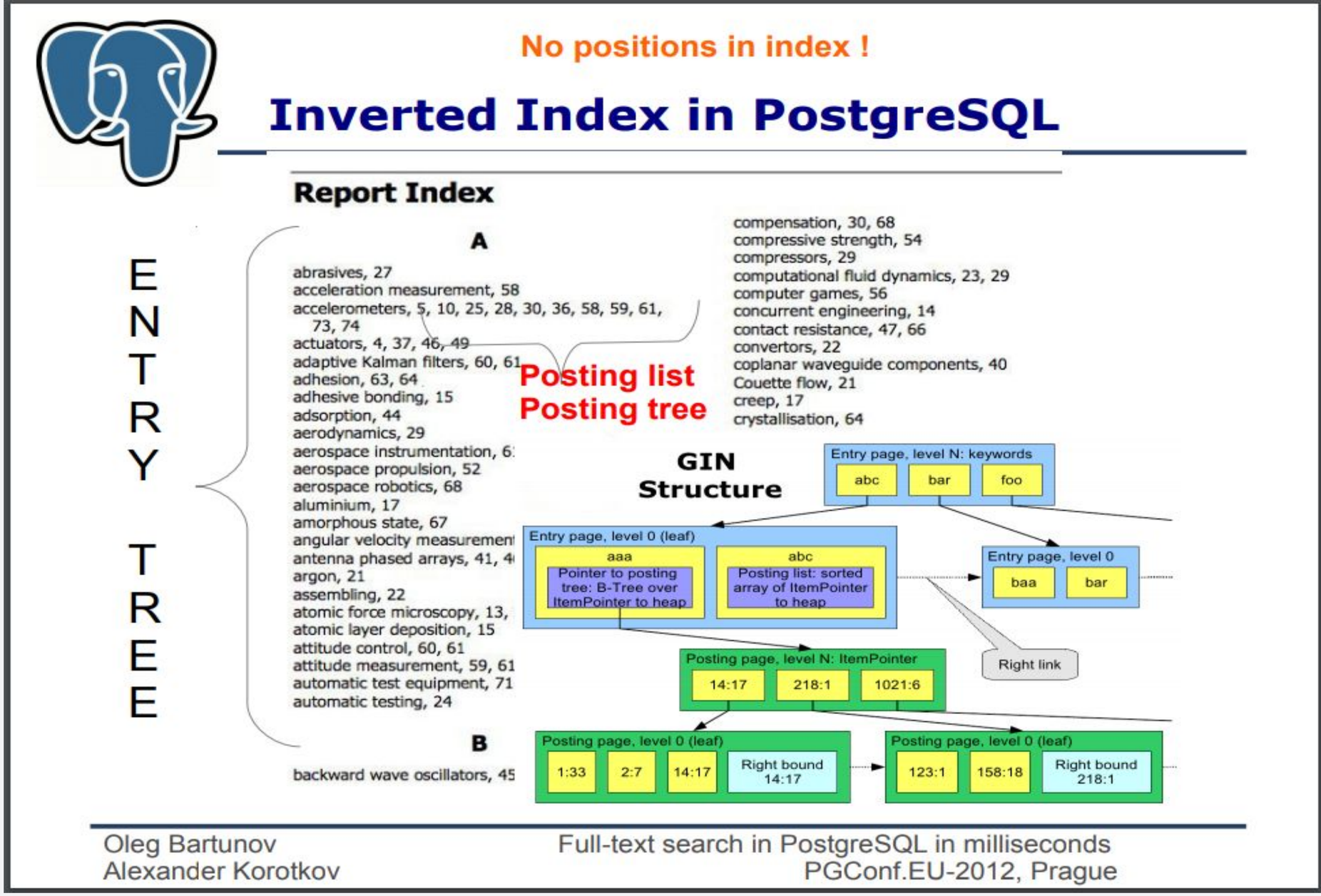
(optionally

compressed >9.3)



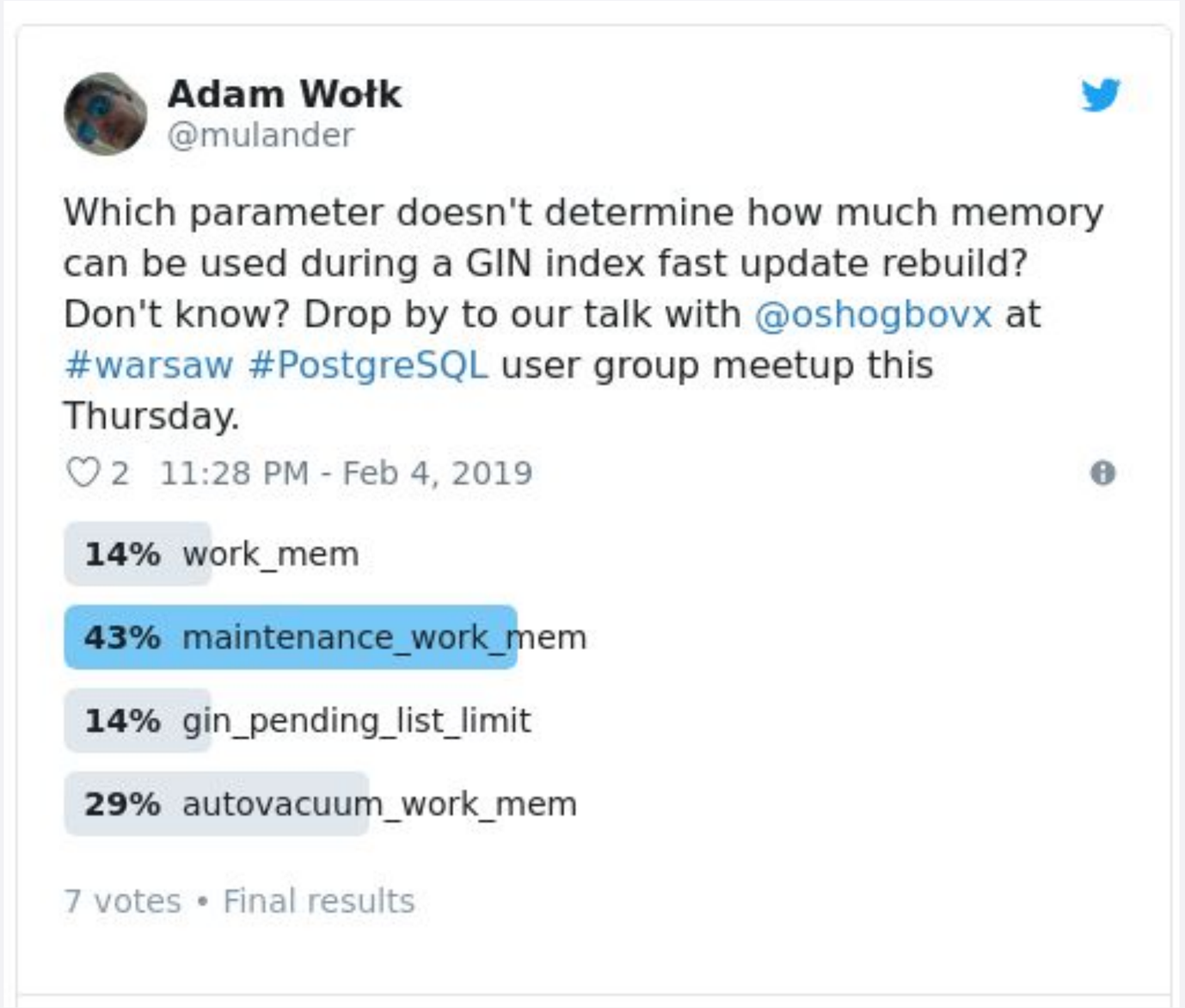
GIN Structure

posting tree (B-tree) created when the **posting list** is to big to fit along the key.



GIN Structure

- **pending list**
  - linked list of pending keys entries, that were not yet merged with the main btree
  - only when **fast update** is enabled (the **default**)
  - attached to the metapage


Adam Wołk
@mulander
Which parameter doesn't determine how much memory can be used during a GIN index fast update rebuild? Don't know? Drop by to our talk with @oshogbovx at #warsaw #PostgreSQL user group meetup this Thursday.
2 11:28 PM - Feb 4, 2019
14% work_mem
43% maintenance_work_mem
14% gin_pending_list_limit
29% autovacuum_work_mem
7 votes • Final results

How does fast update work?

- Index rebuilds are costly with bulk inserts
  - multiple searches/insertions when the same key appears in multiple new heap tuples
- Pending list
  - maintains a linked list along the index metapage, performs a linear search of it’s elements before searching the tree
- When inserting new records, instead of merging with the tree, append records to the pending list

How does fast update work?

- **Predicate Locking**
  - with **fastupdate=on** all index scans grab a lock on the metapage, which essentially is a lock on the whole index.

    This reflects the facts that an entry to the pending list might land anywhere in the btree so we can’t limit locking to a subset of it.

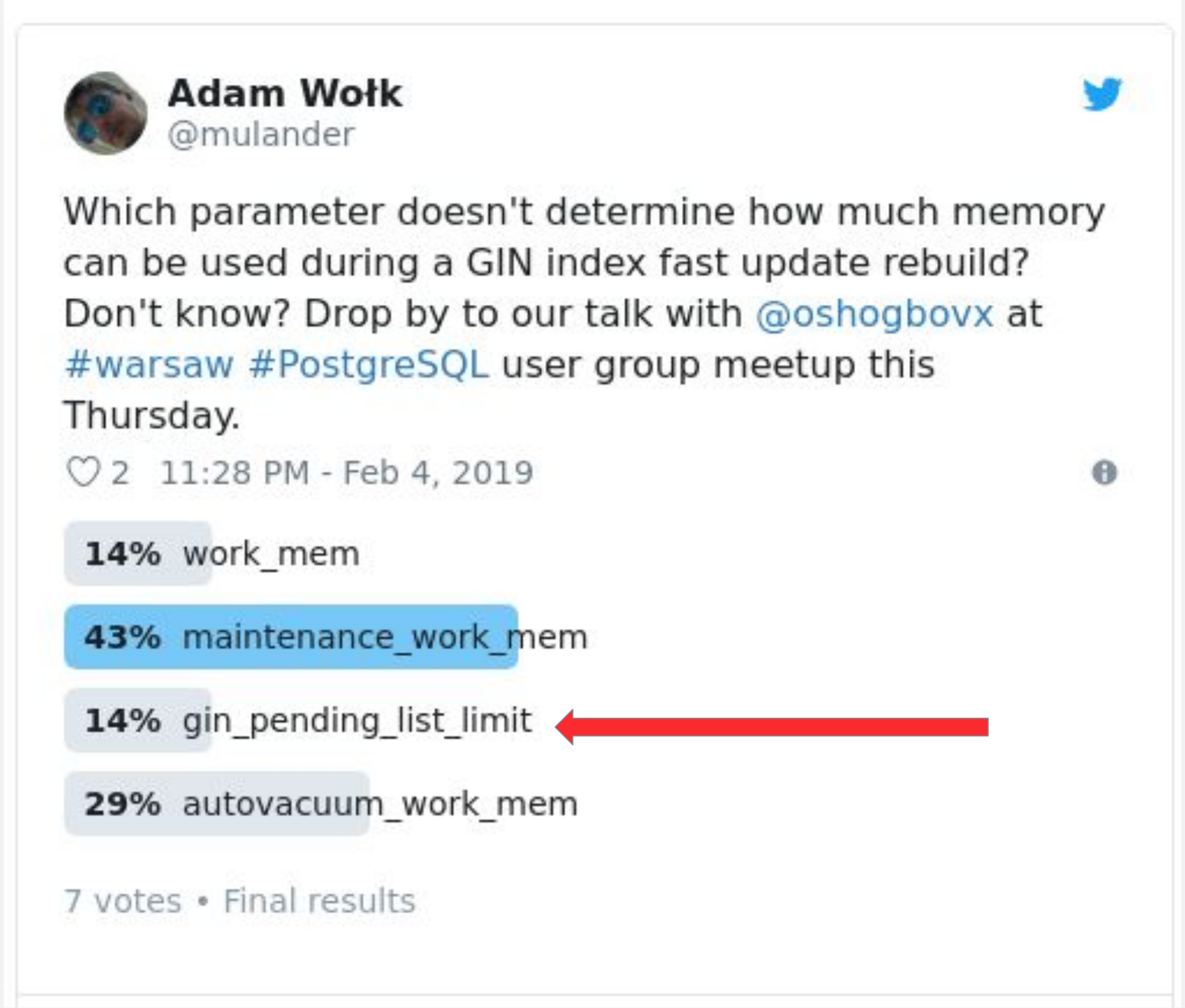
How does fast update work?

- Scanning the pending list linearly increases the cost of each query against the index, hence the pending list should be merged with the main btree before it gets too big.
- vacuum / autovacuum (also in **autovacuum analyze** but not on direct ANALYZE)
- post insert based on length check or triggered by a function call
  - work_mem (**< 9.5**)
  - gin_pending_list_limit (**>= 9.5**)
  - gin_clean_pending_list() (**>=9.6**)

How does fast update work?

- Merging the pending list will use at most:
  - **work_mem** - if **ginInsertCleanup** happened post INSERT
  - **autovacuum_work_mem** - if triggered during AUTOVACUUM and the parameter was set
  - **maintainance_work_mem** - if triggered by a call to **gin_clean_pending_list()** or triggered by AUTOVACUUM without autovacuum_work_mem being set.

Choosing a size for **gin_pending_list_limit** one should account for the above scenarios.


Adam Wołk
@mulander
Which parameter doesn't determine how much memory can be used during a GIN index fast update rebuild? Don't know? Drop by to our talk with @oshogbovx at #warsaw #PostgreSQL user group meetup this Thursday.
2 11:28 PM - Feb 4, 2019
14% work_mem
43% maintenance_work_mem
14% gin_pending_list_limit
29% autovacuum_work_mem
7 votes • Final results

Test

Test data

```
create table test_table(
        id bigserial primary key,
        text tsvector
) WITH (autovacuum_enabled = off);
CREATE INDEX test_idx ON test_table
USING gin(text);
```

Test data

```
INSERT INTO test_table(text)
    SELECT to_tsvector('english',
md5('dummy' || id::text))
    FROM generate_series(1, 2000000)
AS id;
```

Test configuration

work_mem = 4MB-64MB
gin_pending_list_limit = 4MB-64MB
max_worker_processes = 1

First two in order to observe their impact on our fast update index rebuilds.

The last one for testing ease, it's not a requirement for using DTrace.

Let’s look inside

What can we do without DTrace?

- pageinspect

```
# SELECT * FROM
   gin_metapage_info(get_raw_page('test_idx', 0));
-[ RECORD 1 ]----+------
pending_head     | 2
pending_tail     | 275
tail_free_size   | 1512
n_pending_pages  | 274
n_pending_tuples | 42182
n_total_pages    | 2
n_entry_pages    | 1
```

- pgstattuple

```
# SELECT * FROM
    pgstatginindex('test_idx');
-[ RECORD 1 ]--+------
version        | 2
pending_pages  | 274
pending_tuples | 42182
```

What we can do without DTrace?

- A lot of code
- We need to load it into progres
- We don’t know when it’s triggered
- Are those extensions compatible with your PostgreSQL version?
- The feature might be too new to have an extension

monitor.d

```
#pragma D option quiet
BEGIN {printf("%20s | %s\n", "WHAT", "TIME");}

pid$target::ginInsertCleanup:entry {
    flush = timestamp;
}

pid$target::ginInsertCleanup:return {
    printf("%20s | %d\n",
        "Flushing pending list", timestamp - flush);
}
```

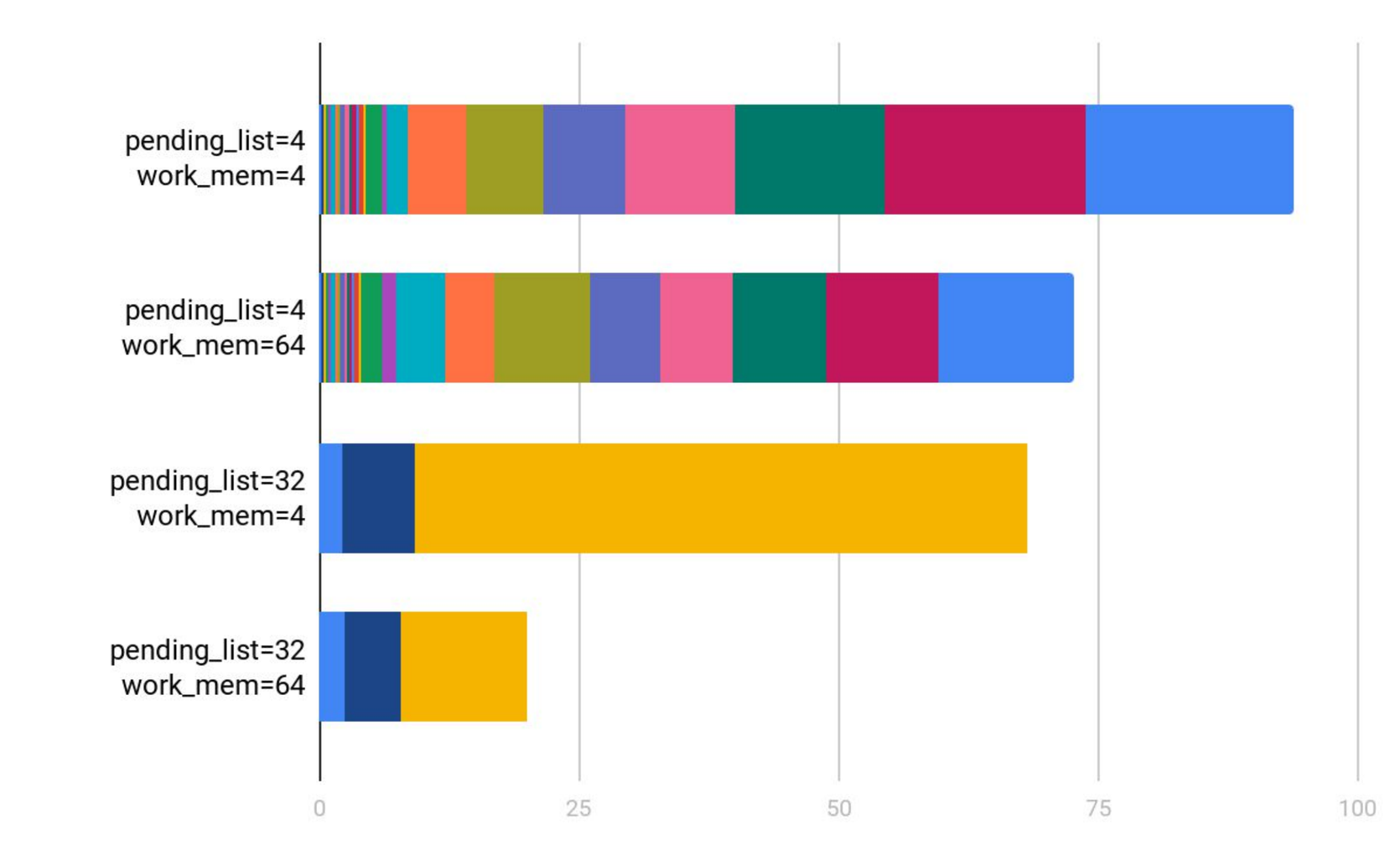
```
pid$target::standard_ExecutorRun:entry {
    insert = timestamp;
}

pid$target::standard_ExecutorRun:return {
    printf("%20s | %d\n", "exec",
        timestamp - insert);
}
```

monitor.d

WHAT	TIME [ns]
Flushing pending list	210369214
Flushing pending list	226589355
Flushing pending list	269698917
Flushing pending list	241673579
Flushing pending list	257309400
Flushing pending list	250452060
Flushing pending list	258158565
Flushing pending list	359692281
[...]	

WHAT	TIME [ns]
[...]	
Flushing pending list	14566890307
Flushing pending list	19177771442
Flushing pending list	20073140082
exec	124839310290

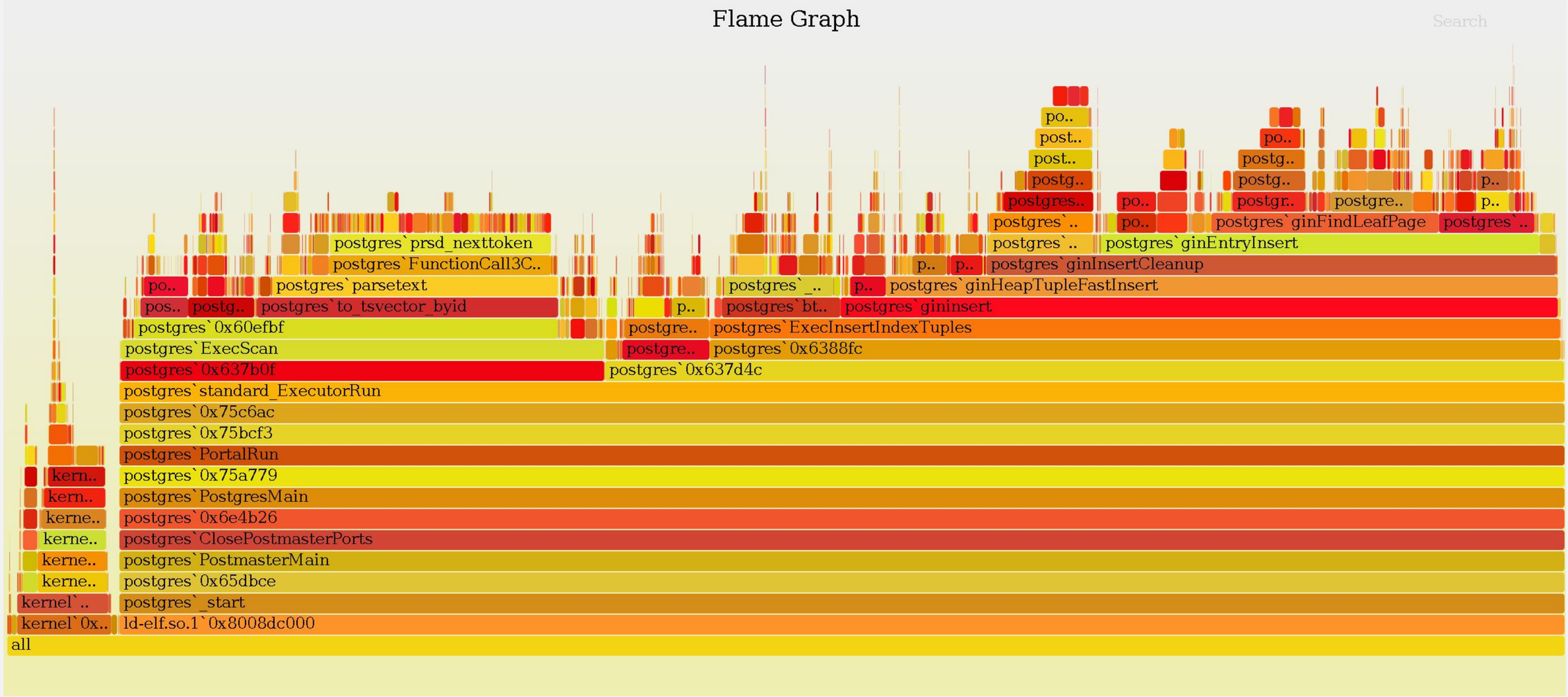
Results - monitor.d



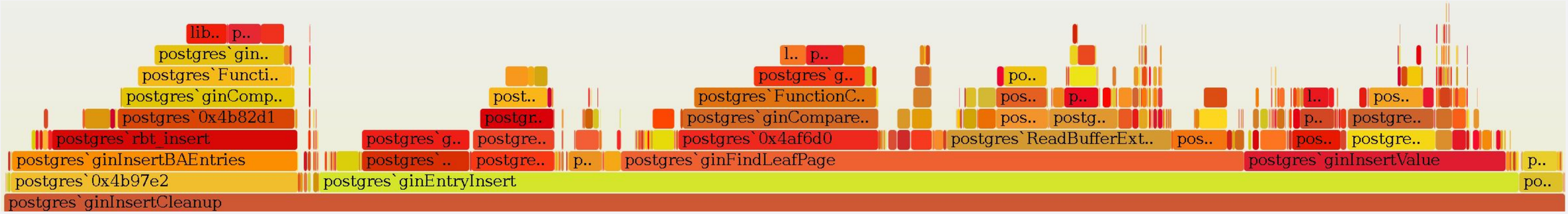
Dtrace FlameGraph

```
profile-5000 /arg1 && pid == $target/ {
     @[ustack()] = count();
}

profile-5000 /arg0 && pid == $target/ {
     @[stack()] = count();
}
```

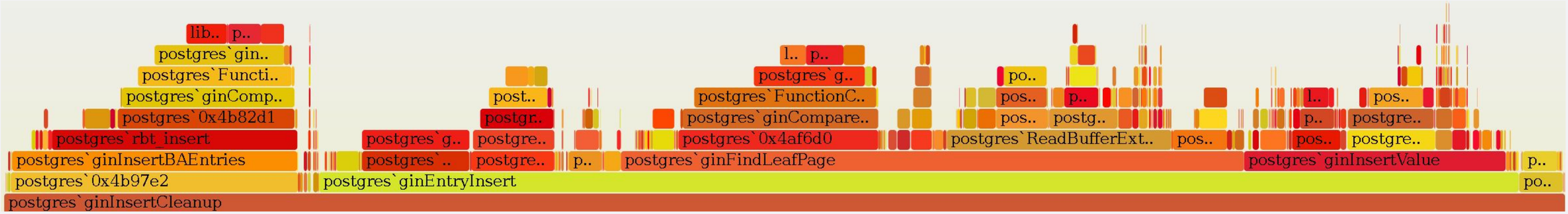
pending_list=32, work_mem=4

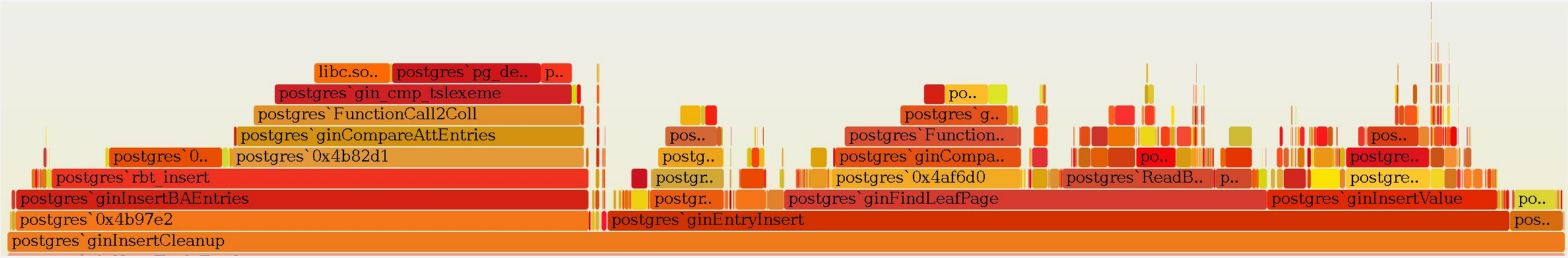


pending_list=32, work_mem=4



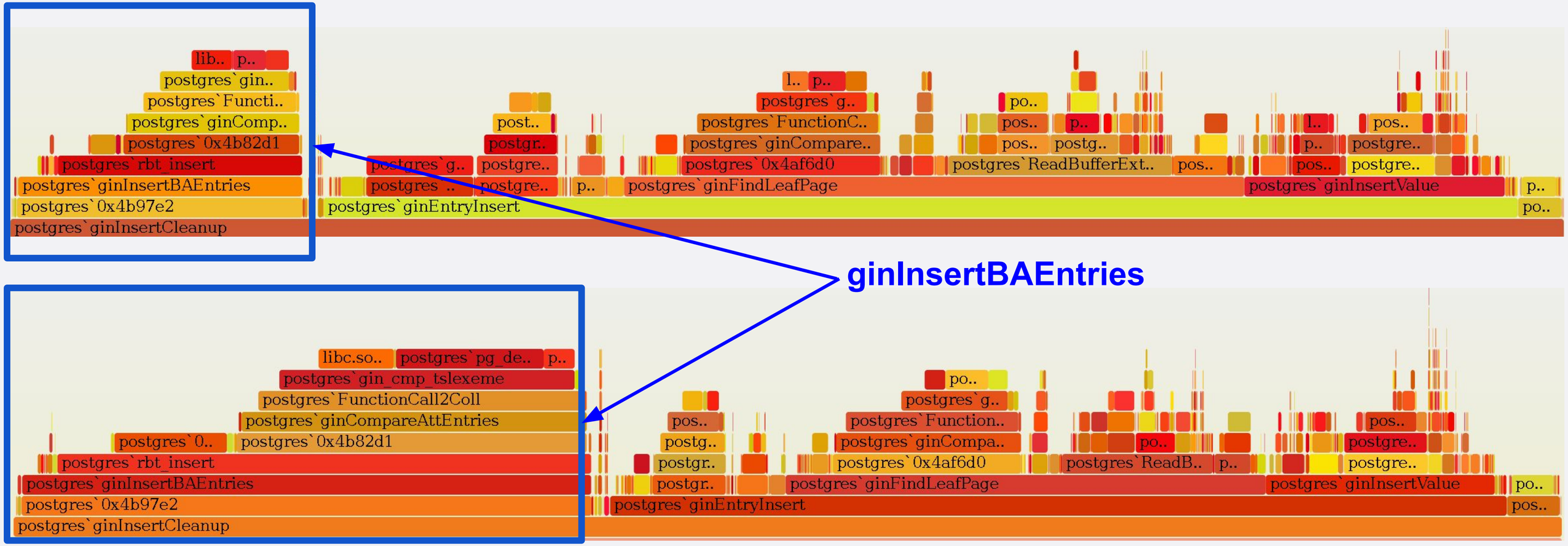
pending_list=32, work_mem=4





pending_list=32, work_mem=32

pending_list=32, work_mem=4



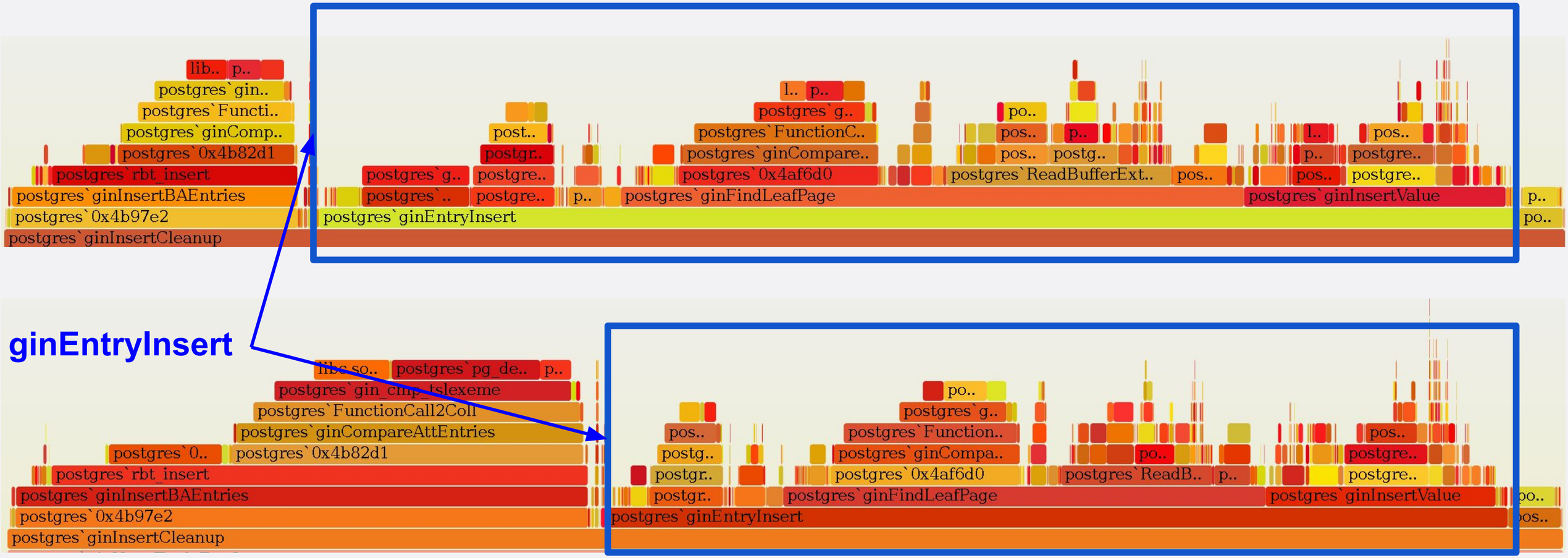
pending_list=32, work_mem=32

processPendingPage (inline of ginInsertBAEntries)

Collect data from a pending-list page in preparation for insertion into the main index.

Test	Samples
pending_list=32, work_mem=4	6000
pending_list=32, work_mem=64	12000

pending_list=32, work_mem=4



pending_list=32, work_mem=32

ginEntryInsert

Insert one or more heap TIDs associated with the given key value.

This will either add a single key entry, or enlarge a pre-existing entry.

Moving collected data into regular structure can take significant amount of time.

Insert hangs, what next?

Insert hangs, what next?

```
insert into test_table(text) values(to_tsvector('Ph''nglui
mglw''nafh Cthulhu R''lyeh wgah''nagl fhtagn'));
```

Insert hangs, what next?

Find a backend PID for this query

```
select * from pg_stat_activity where ...
-[ RECORD 1 ]----+---------------------
datid            | 16388
datname          | test
pid              | 35206
usesysid         | 16384
usename          | postgres
application_name |
client_addr      | 127.0.0.1
client_hostname  |
client_port      | 34627
backend_start    | 2018-12-11 13:11:58.832672+01
xact_start       | 2018-12-11 13:52:24.083262+01
query_start      | 2018-12-11 13:52:24.123961+01
state_change     | 2018-12-11 13:52:24.140457+01
waiting          | f
state            | active
query            | insert into test_table(text) values(to_tsvector('Ph''nglui mglw''nafh Cthulhu R''lyeh wgah''nagl fhtagn'));
```

Insert hangs, what next?

DTrace one liner:

dtrace -n ‘`profile-99 /arg1 && pid == $target/ {ustack(); exit(1)}’ -p PID`

Insert hangs, what next?

DTrace one liner:

dtrace -n ‘profile-99 /arg1 && pid == $target/ {ustack(); exit(1)}’ -p PID

```
postgres`PathNameOpenFilePerm+0x1f
postgres`0x752c3e
postgres`mdexists+0x92
postgres`0x732ed1
postgres`RecordPageWithFreeSpace+0x5b
postgres`ginInsertCleanup+0x76f
postgres`ginHeapTupleFastInsert+0x5e9
postgres`gininsert+0x118
```

Insert hangs, what next?

DTrace one liner:

dtrace -n ‘profile-99 /arg1 && pid == $target/ {ustack(); exit(1)}’ -p PID

```
postgres`PathNameOpenFilePerm+0x1f
postgres`0x752c3e
postgres`mdexists+0x92
postgres`0x732ed1
postgres`RecordPageWithFreeSpace+0x5b
postgres`ginInsertCleanup+0x76f
postgres`ginHeapTupleFastInsert+0x5e9
postgres`gininsert+0x118
```

What about the length of the pending list...

What about the length of the pending list...

```
struct GinMetaPageData
{
        uint32_t head;
        uint32_t tail;
        uint32_t tailFreeSize;
        uint32_t nPendingPages;
        int64_t nPendingHeapTuples;
        uint32_t nTotalPages;
        uint32_t nEntryPages;
        uint32_t nDataPages;
        int64_t nEntries;
       int32_t ginVersion;
};
```

What about the length of the pending list...

```
void
ginInsertCleanup(GinState *ginstate, bool full_clean,
                                 bool fill_fsm, bool forceCleanup,
                                 IndexBulkDeleteResult *stats)
{
        GinMetaPageData *metadata;

        metabuffer = ReadBuffer(index, GIN_METAPAGE_BLKNO);
        LockBuffer(metabuffer, GIN_SHARE);
        metapage = BufferGetPage(metabuffer);
        metadata = GinPageGetMeta(metapage);

        if (metadata->head == InvalidBlockNumber)
        {
```

What about the length of the pending list...

```
void
ginInsertCleanup(GinState *ginstate, bool full_clean,
                                bool fill_fsm, bool forceCleanup,
                                IndexBulkDeleteResult *stats)
{
        GinMetaPageData *metadata;

        metabuffer = ReadBuffer(index, GIN_METAPAGE_BLKNO);
        LockBuffer(metabuffer, GIN_SHARE);
        metapage = BufferGetPage(metabuffer);
        metadata = GinPageGetMeta(metapage);

        if (metadata->head == InvalidBlockNumber)
        {
```

```
0000000004b8d40 <ginInsertCleanup>:
4b8de1: add  rax,QWORD PTR [rip+0x40caa8]
4b8de8: jmp  4b8dfd <ginInsertCleanup+0xbd>
4b8dea: mov  rax,QWORD PTR [rip+0x432e07]
4b8df1: mov  ecx,r13d
4b8df4: not  ecx
4b8df6: movsxd rcx,ecx
4b8df9: mov  rax,QWORD PTR [rax+rcx*8]
4b8dfd: mov  r14d,DWORD PTR [rax+0x18]
4b8e01: cmp  r14d,0xffffffffffffffff
4b8e05: je   4b8e49 <ginInsertCleanup+0x109>
4b8e07: movsxd rcx,r15d
```

What about the length of the pending list...

```
void
ginInsertCleanup(GinState *ginstate, bool full_clean,
                                bool fill_fsm, bool forceCleanup,
                                IndexBulkDeleteResult *stats)
{
        GinMetaPageData *metadata;

        metabuffer = ReadBuffer(index, GIN_METAPAGE_BLKNO);
        LockBuffer(metabuffer, GIN_SHARE);
        metapage = BufferGetPage(metabuffer);
        metadata = GinPageGetMeta(metapage);

        if (metadata->head == InvalidBlockNumber)
        {
```

```
0000000004b8d40 <ginInsertCleanup>:
4b8de1: add  rax,QWORD PTR [rip+0x40caa8]
4b8de8: jmp  4b8dfd <ginInsertCleanup+0xbd>
4b8dea: mov  rax,QWORD PTR [rip+0x432e07]
4b8df1: mov  ecx,r13d
4b8df4: not  ecx
4b8df6: movsxd rcx,ecx
4b8df9: mov  rax,QWORD PTR [rax+rcx*8]
4b8dfd: mov  r14d,DWORD PTR [rax+0x18]
4b8e01: cmp  r14d,0xffffffffffffffff
4b8e05: je   4b8e49 <ginInsertCleanup+0x109>
4b8e07: movsxd rcx,r15d
```

What about the length of the pending list...

```
pid$target::ginInsertCleanup:c1 {
    ret = (struct GinMetaPageData *)
            copyin(uregs[R_EAX] + 0x18, sizeof(struct GinMetaPageData));
    printf("nPendingPages %d\n", ret->nPendingPages);
    printf("nPendingHeapTuples %d\n", ret->nPendingHeapTuples);
}
```

Useful resources and related materials

- WHAT POSTGRESQL FULL-TEXT-SEARCH HAS TO DO WITH VACUUM
- GIN implementation source code
- GIN tips in PostgreSQL documentation
- GIN implementation details in PostgreSQL documentation
- pageinspect GIN functions

Thank you!

Adam Wołk

a.wolk@fudosecurity.com

awolk@openbsd.org

https://blog.tintagel.pl

@mulander

Mariusz Zaborski

m.zaborski@fudosecurity.com

oshogbo@FreeBSD.org

https://oshogbo.vexillium.org

@oshogbovx