Microsoft

Internal PostgreSQL Lecture Series 2024-01-12

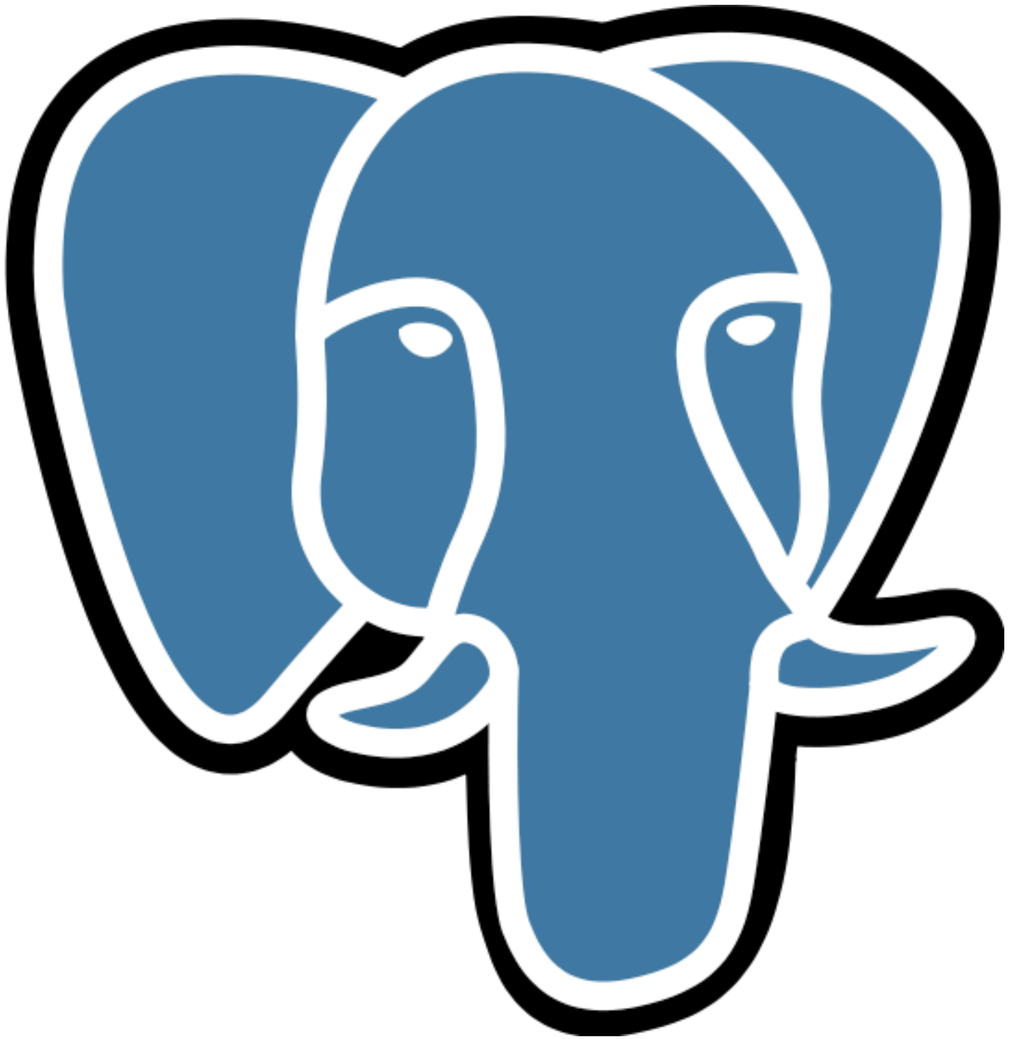
Daniel Gustafsson

Daniel Gustafsson

Hacking Postgres at Microsoft since April 2023

Based in Sweden

Timezone nomad..



Contributor since 2014

Committer (core, web)

Stockholm PUG (Sweden)

Nordic PGDay (Nordic region)

PGConf EU (Europe)

FOSDEM PGDay (Belgium)

PGConf.dev (Canada)

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system The main design goals of the new system are to

1) provide better support for complex objects,
2) provide user extendibility for data types, operators and access methods,
3) provide facilities for active databases (i e , alerters and triggers) and inferencing including forward- and backward-chaining,
4) simplify the DBMS code for crash recovery,
5) produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips, and
6) make as few changes as possible (preferably none) to the relational model

https://modelrailwaylayoutsplans.com/wp-content/uploads/2021/06/03.-5757.jpg

What you need
**Maybe* not all*
that you want

Hooks

Injection points via global function pointers

Invoked when set to non-NULL from shared libraries

Daisy-chained via good citizenry

```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    if (prev_hook)
       prev_hook();
}

void
_PG_init(void)
{
   prev_hook = imaginary_hook;
   imaginary_hook = my_hook_function;
}

void
_PG_fini(void)
{
   imaginary_hook = prev_hook;
}
```

```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    if (prev_hook)
       prev_hook();
}

void
_PG_init(void)
{
   prev_hook = imaginary_hook;
    imaginary_hook = my_hook_function;
}

void
_PG_fini(void)
{
   imaginary_hook = prev_hook;
}
```

Save the current hook to not break the chain

```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    if (prev_hook)
       prev_hook();
}

void
_PG_init(void)
{
   prev_hook = imaginary_hook;
   imaginary_hook = my_hook_function;
}

void
_PG_fini(void)
{
   imaginary_hook = prev_hook;
}
```

Set our own hook as the head of the chain

```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    if (prev_hook)
       prev_hook();
}

void
_PG_init(void)
{
   prev_hook = imaginary_hook;
   imaginary_hook = my_hook_function;
}

void
_PG_fini(void)
{
   imaginary_hook = prev_hook;
}
```

Execute hooks in first come first serve order

```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    if (prev_hook)
       prev_hook();
}

void
_PG_init(void)
{
   prev_hook = imaginary_hook;
   imaginary_hook = my_hook_function;
}

void
_PG_fini(void)
{
   imaginary_hook = prev_hook;
}
```

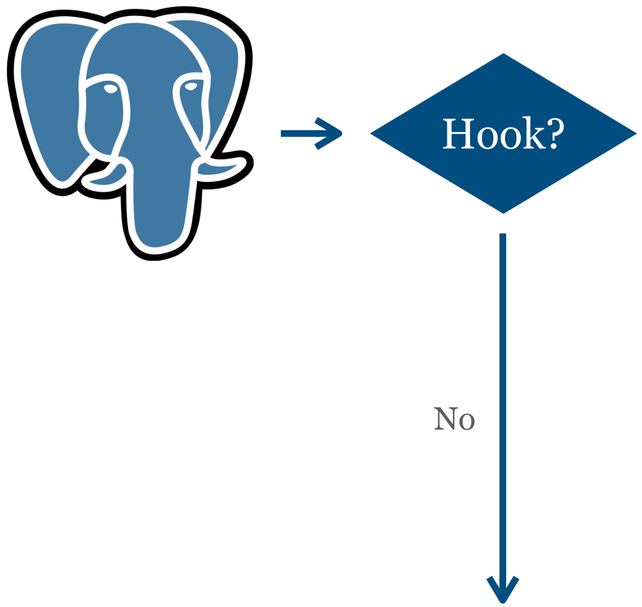
Restore the chain on module unload

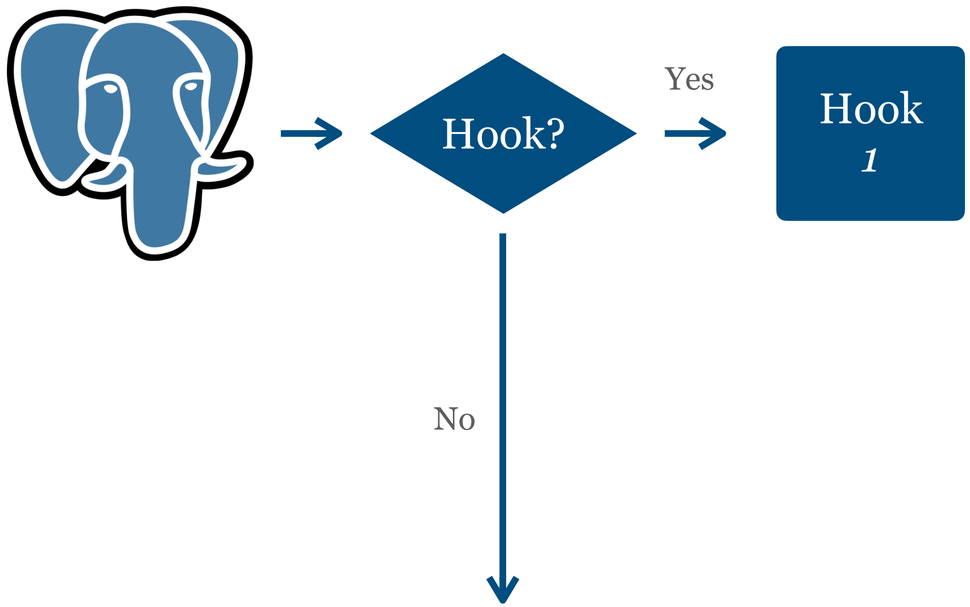
```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    /* do stuff */

    if (prev_hook)
       prev_hook();
    else
       execute_replaced_code();

    /* do more stuff perhaps */
}
```

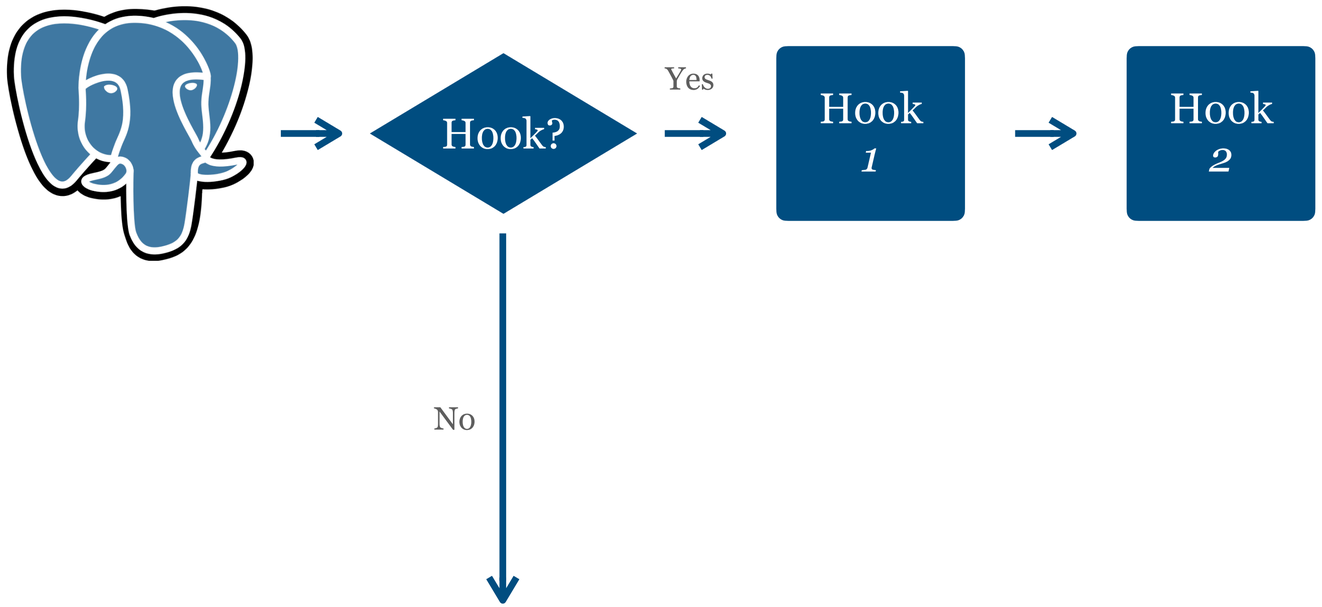
Only one in the chain can run the replaced code..

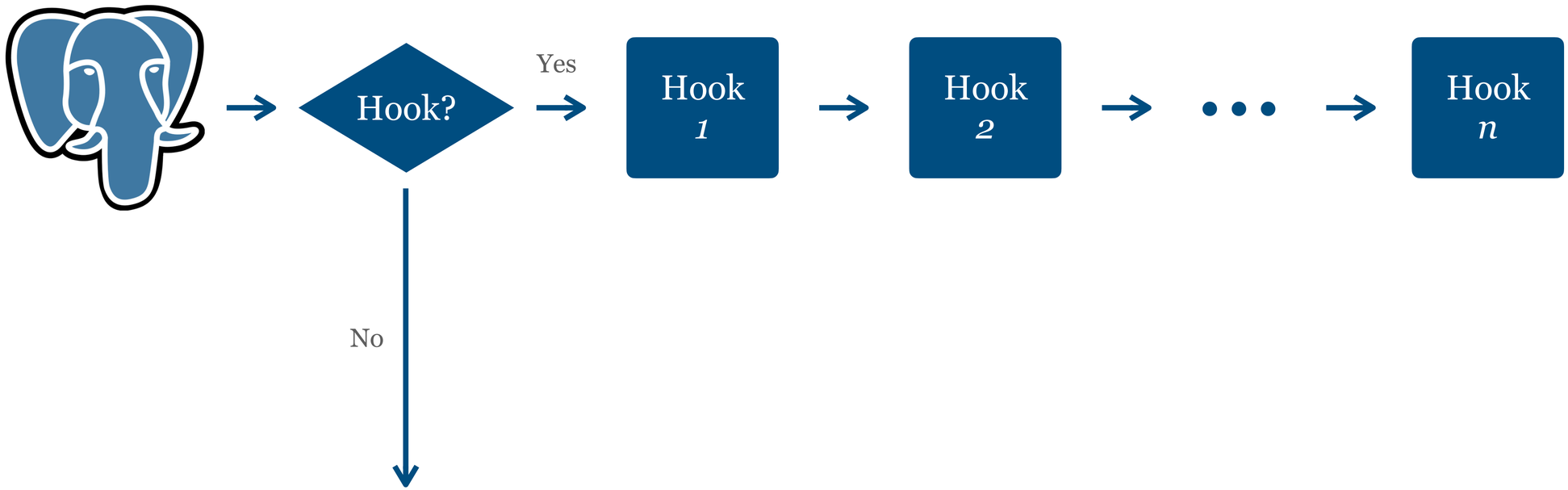
```
static imaginary_hook_type *prev_hook;

static void my_hook_function(void)
{
    /* do stuff */

    if (prev_hook)
       prev_hook();
    else
       execute_replaced_code();

    /* do more stuff perhaps */
}
```

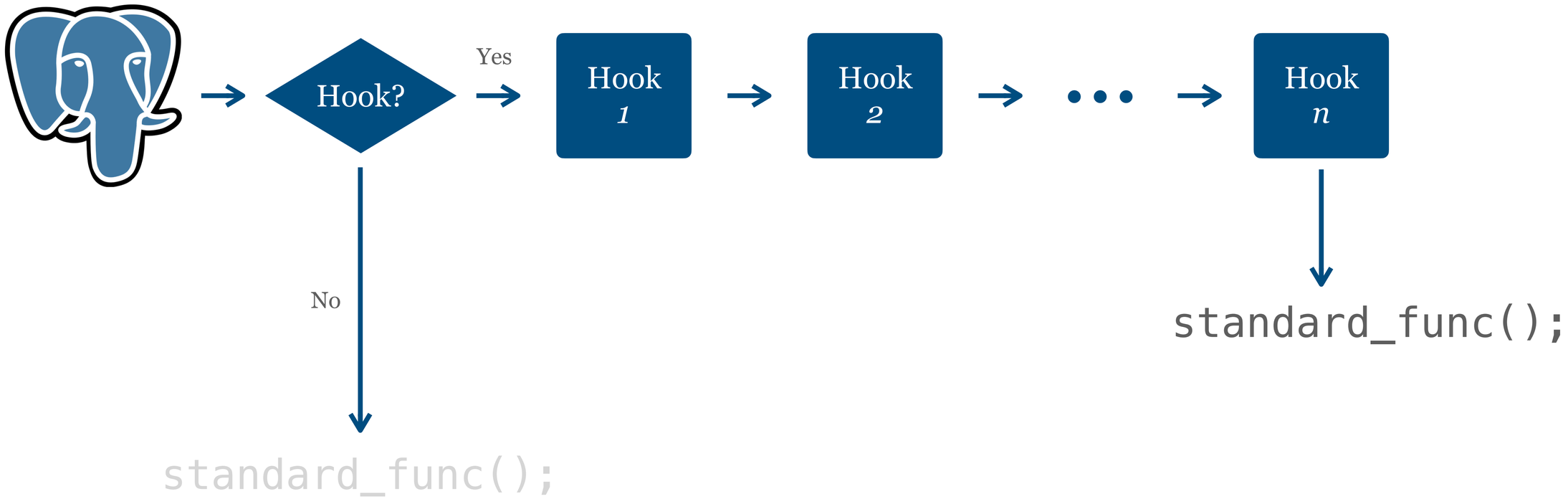
Only one in the chain can run the replaced code..

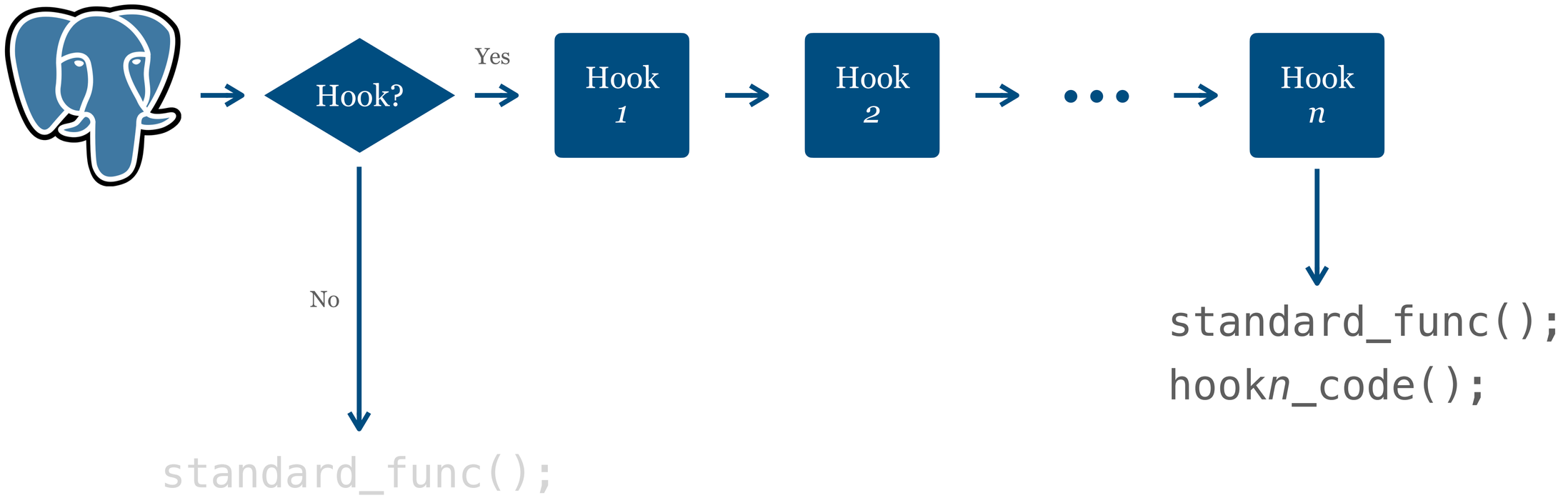

Hook?
No
standard_func();

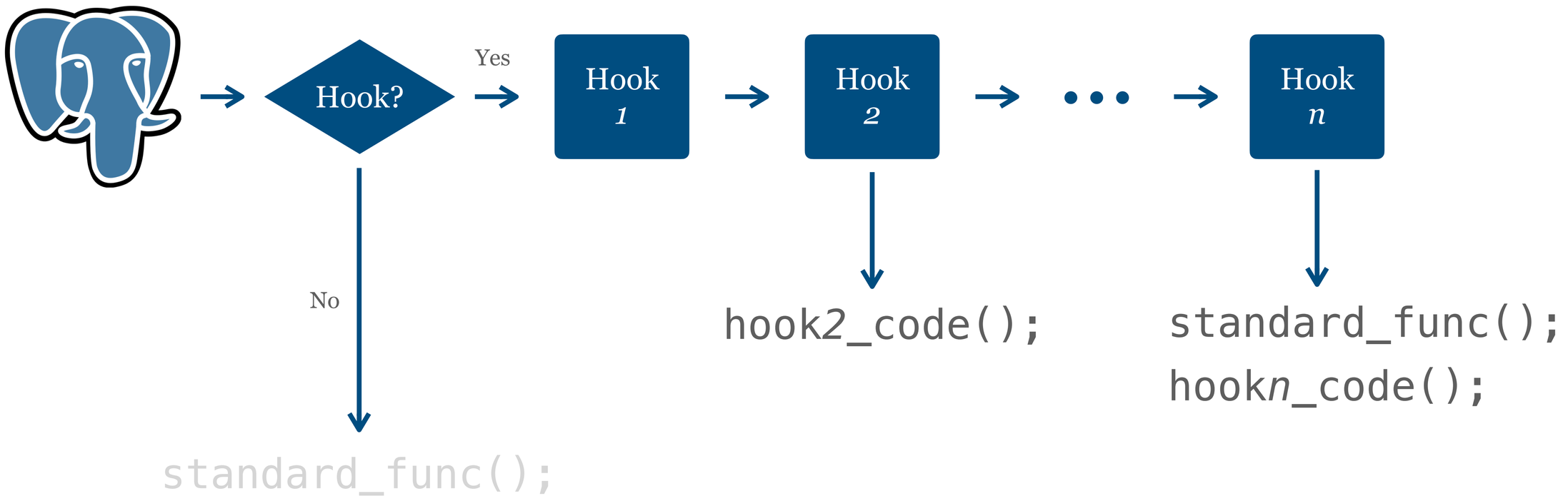

Hook?
Yes
Hook
1
No
standard_func();

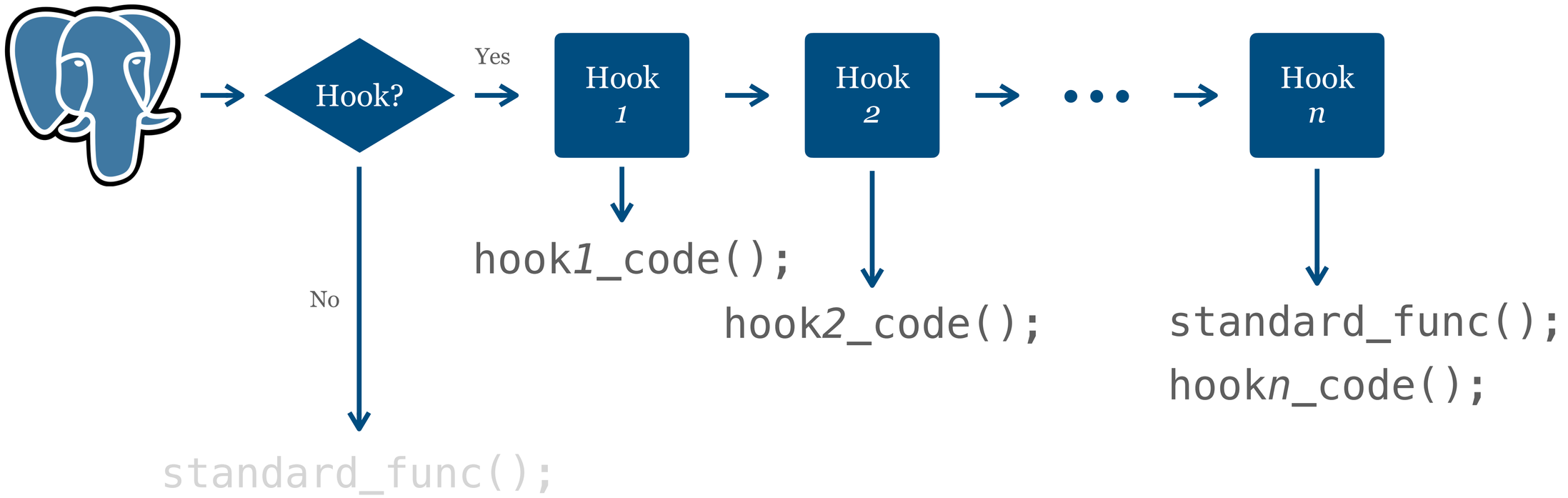

Hook?
Yes
Hook
1
Hook
2
No
standard_func();


Hook?
Yes
No
Hook
1
Hook
2
Hook
n
standard_func();


Hook?
Yes
No
Hook
1
Hook
2
Hook
n
standard_func();
standard_func();


Hook?
Yes
No
Hook
1
Hook
2
Hook
n
standard_func();
hookn_code();
standard_func();


Hook?
Yes
No
Hook
1
Hook
2
Hook
n
hook2_code();
standard_func();
hookn_code();
standard_func();


Hook?
Yes
No
Hook 1
Hook 2
Hook n
hook1_code();
hook2_code();
standard_func();
hookn_code();
standard_func();

Hooks

Unfortunately undocumented

Not guaranteed to be stable across minor revisions, best efforts apply

Tread carefully

Injecting at a hook means running code in the backend un-sandboxed

Does your code work if run in a parallel worker?

Conditional compilation features

Examples

`check_password_hook`: Enforcing password constraints. Additive.

`join_search_hook`: Replace join ordering algorithm and bypass GEQO

https://flickr.com/photos/bionicteaching/26149805112/

Join Order Selection

Join Order Enumeration

Given the set of relations in a query, find the optimal order in which to access the relations in order to satisfy the query

```
A.a = B.b AND
A.d = D.d AND
B.c = C.c
```

```
A.a = B.b AND
A.d = D.d AND
B.c = C.c
```

N! join orderings:

```
ABCD, ABDC, ADBC, DABC ...
```

```
A.a = B.b AND
A.d = D.d AND
B.c = C.c
```

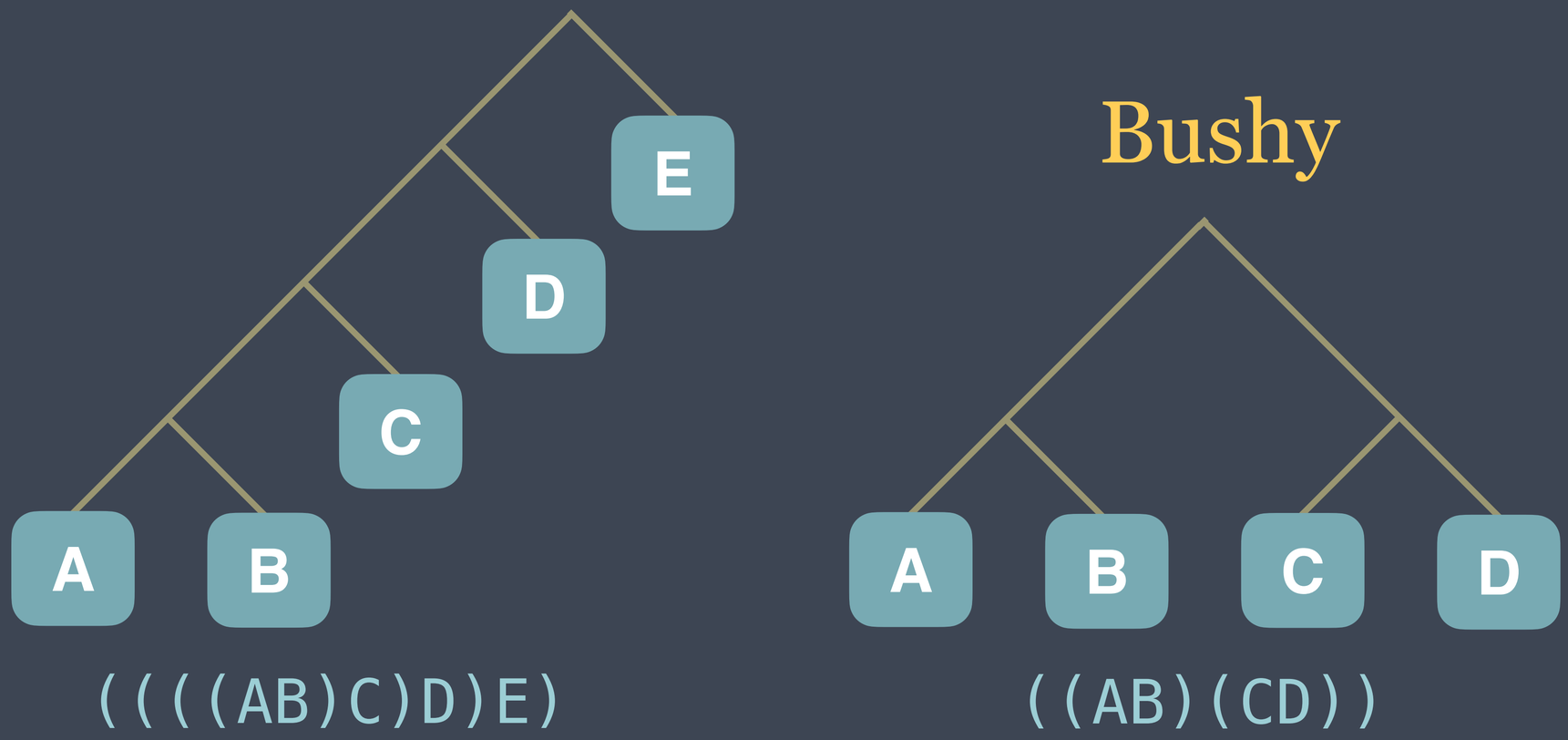
N! join orderings:

ABCD, ABDC, ADBC, DABC ...

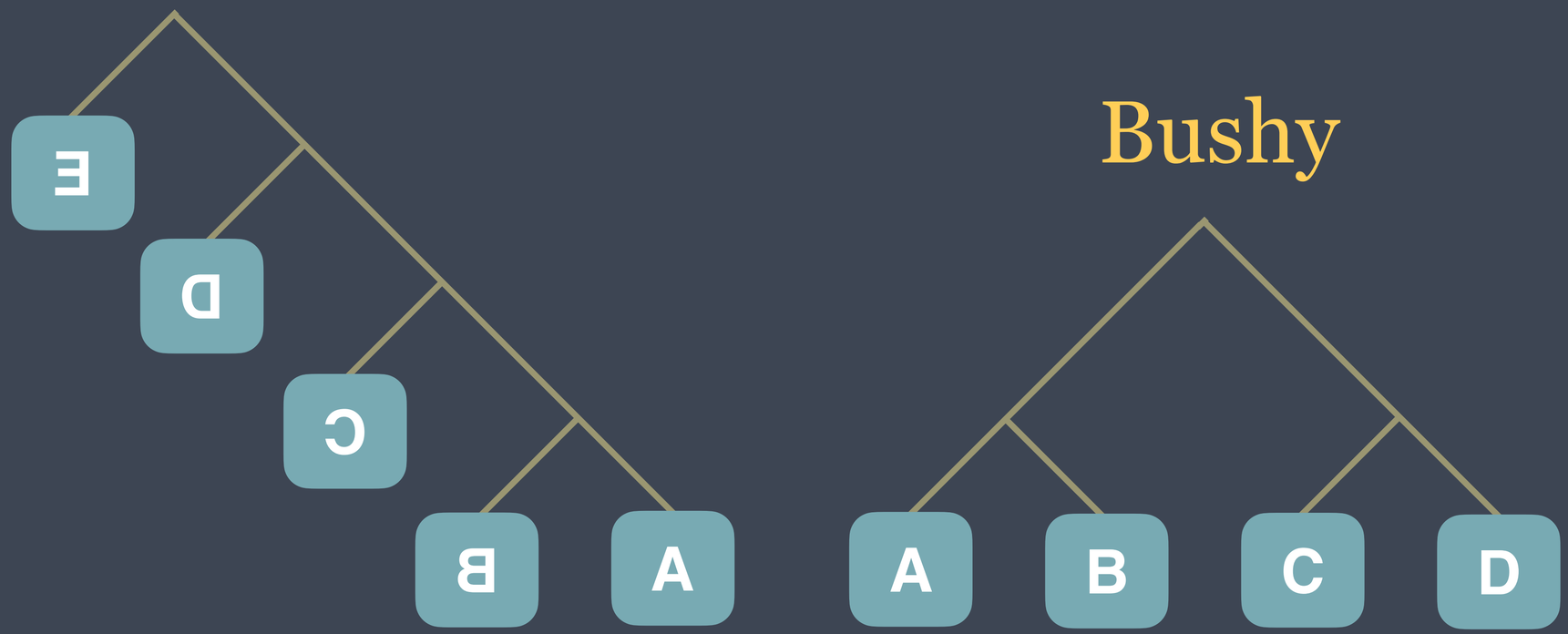
(N-1)! plans per join order:

(((AB)C)D), ((AB)(CD)) ...

Join Trees



Join Trees



N! × (N-1)! possible plans

4 way join → 144 plans

10 way join → 1,316,818,944,000 plans

20 way .. ☠️💣

Naive, and/or, exhaustive approaches doesn't scale

https://www.youtube.com/watch?v=bvnWpTJPPUg

ENDLICH ANGEKOMMEN

ATARI PERSONAL COMPUTER SYSTEM

ATARI 400 (16 K) und ATARI 800 (bis 48 K) sind das Herz des kompletten Personal Computer Systems. Color-PAL-Signal für jeden Fernseher. 6502 Mikroprozessor. Grafik, Sound und über 160 Farben. Programmiersprachen BASIC, PASCAL, PILOT und ASSEMBLER-EDITOR. ROM-Programm-Module. Disk-Drives, Drucker, Programm-Recorder, Interface, Light Pen, Joysticks usw. als geprüftes ATARI-Zubehör. Umfangreiche ATARI-Software-Bibliothek. Lieferung nur über den qualifizierten Fachhandel.

ATARI®

A Warner Communications Company

Computers for people

Fordern Sie jetzt ausführliche Informationen an!

Name: _____

Beruf: _____

Straße: _____

PLZ/Ort: _____

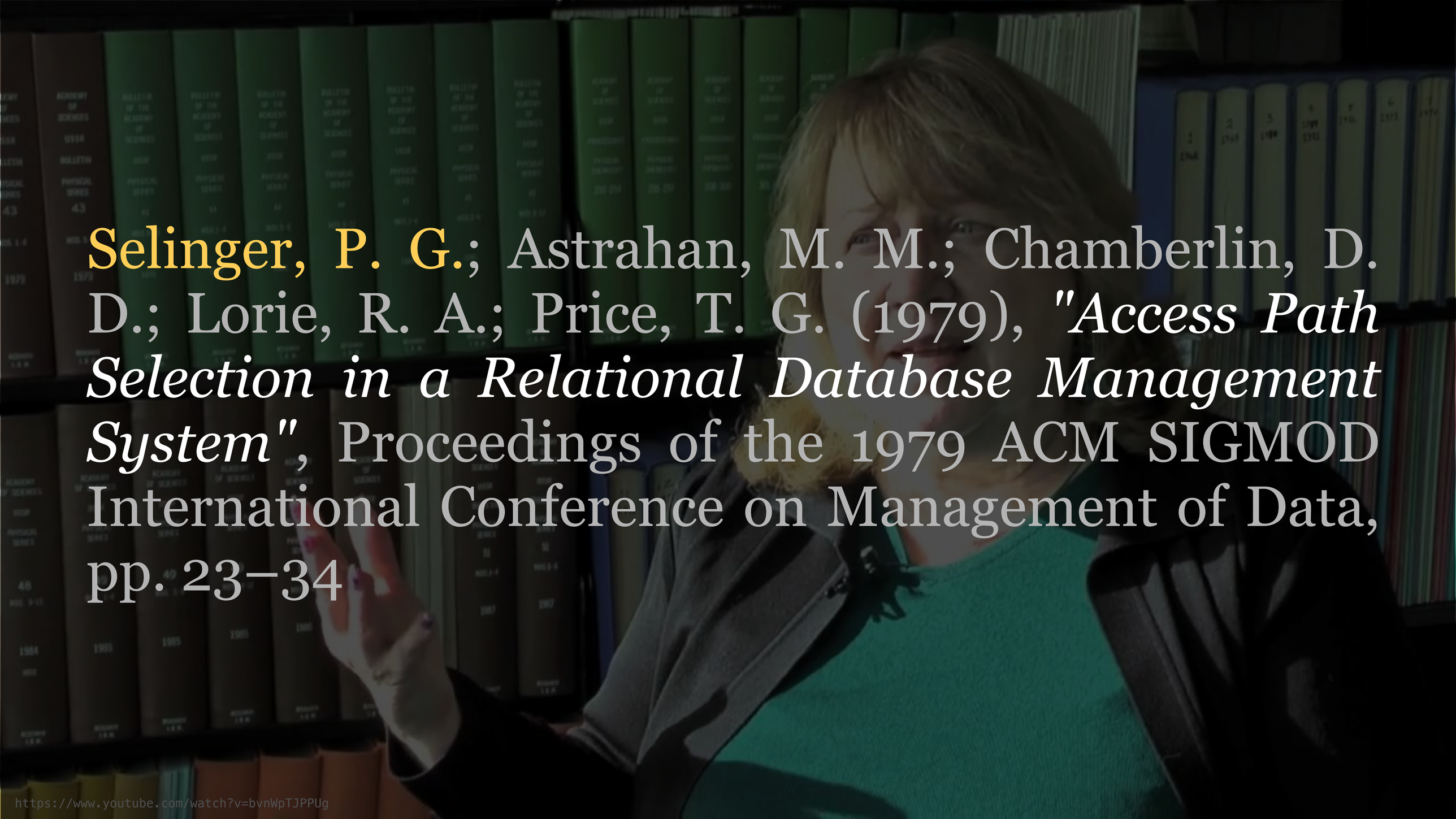
geplanter Einsatzbereich:

☐ Beruf ☐ Hobby ☐ Ausbildung ☐ Unterhaltung

Atari Elektronik Vertriebsgesellschaft mbH
Bebelallee 10 · 2000 Hamburg 60

Machen Sie Ihr Hobby zum Beruf.
Atari sucht „Computer-Freaks" als Mitarbeiter.
Für viele Bereiche.
Schreiben Sie an
Herrn Ollmann.


IBM
System/R



Selinger, P. G.; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A.; Price, T. G. (1979), *"Access Path Selection in a Relational Database Management System"*, Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, pp. 23–34

https://www.youtube.com/watch?v=bvnWpTJPPUg

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Step 2 • Consider all ways to join two relations, using best access path as computed in step one

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Step 2 • Consider all ways to join two relations, using best access path as computed in step one

Step 3 • Consider all ways to join 3 relations, reusing cached calculations from step 2

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Step 2 • Consider all ways to join two relations, using best access path as computed in step one

Step 3 • Consider all ways to join 3 relations, reusing cached calculations from step 2

...

Step n • Consider all ways to join n relations, reusing cached calculations from step n-1

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Cost based

Step 2 • Consider all ways to join two relations, using best access path as computed in step one

Step 3 • Consider all ways to join 3 relations, reusing cached calculations from step 2

...

Step n • Consider all ways to join n relations, reusing cached calculations from step n-1

Selinger Algoritm

Step 1 • Enumerate all access paths to individual relations, keep the cheapest around

Cost based

Step 2 • Consider all ways to join two relations, using best access path as computed in step one

Step 3 • Consider all ways to join 3 relations, reusing cached calculations from step 2

...

Step n • Consider all ways to join n relations, reusing cached calculations from step n-1

Dynamic Programming

```
A.a = B.b AND
A.d = D.d AND
B.c = C.c
```

Step 1 · Access Paths

```
A = OptimalAccess(A_relation);
B = OptimalAccess(B_relation);
...
```

Step 2 · 2-way Join

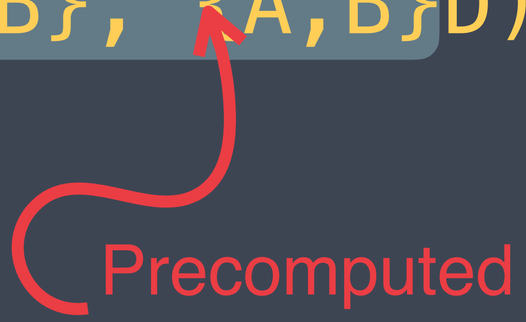
```
{A,B} = Cheapest(AB,BA);
{B,C} = Cheapest(BC,CB);
...
```

Step 3 · 3-way Join

```
{A,B,C} = Cheapest(A{B,C}, {B,C}A,
                   B{A,C}, {A,C}B,
                   C{A,B}, {A,B}C);

{A,B,D} = Cheapest(A{B,D}, {B,D}A,
                   B{A,D}, {A,D}B,
                   D{A,B}, {A,B}D);

...
```

Step 3 · 3-way Join

```
{A,B,C} = Cheapest(A{B,C}, {B,C}A,
                   B{A,C}, {A,C}B,
                   C{A,B}, {A,B}C);

{A,B,D} = Cheapest(A{B,D}, {B,D}A,
                   B{A,D}, {A,D}B,
                   D{A,B}, {A,B}D);

...
```



Step n · n-way Join

```
{A,B,C,D} = ...
```

Step n · n-way Join

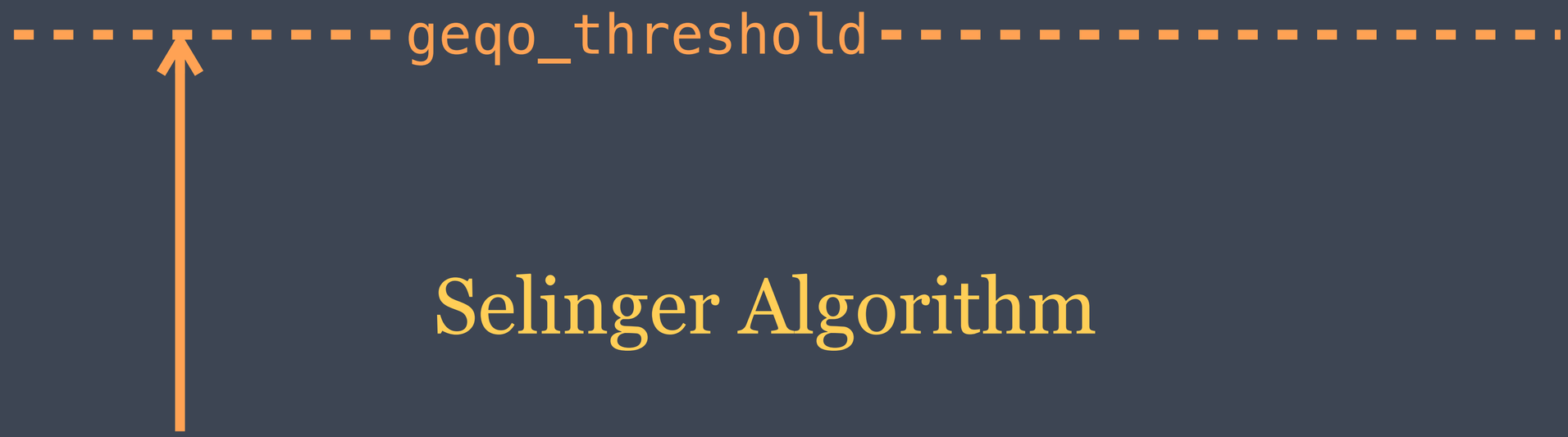
```
{A,B,C,D} = ...
```

Cheapest join order for query reached

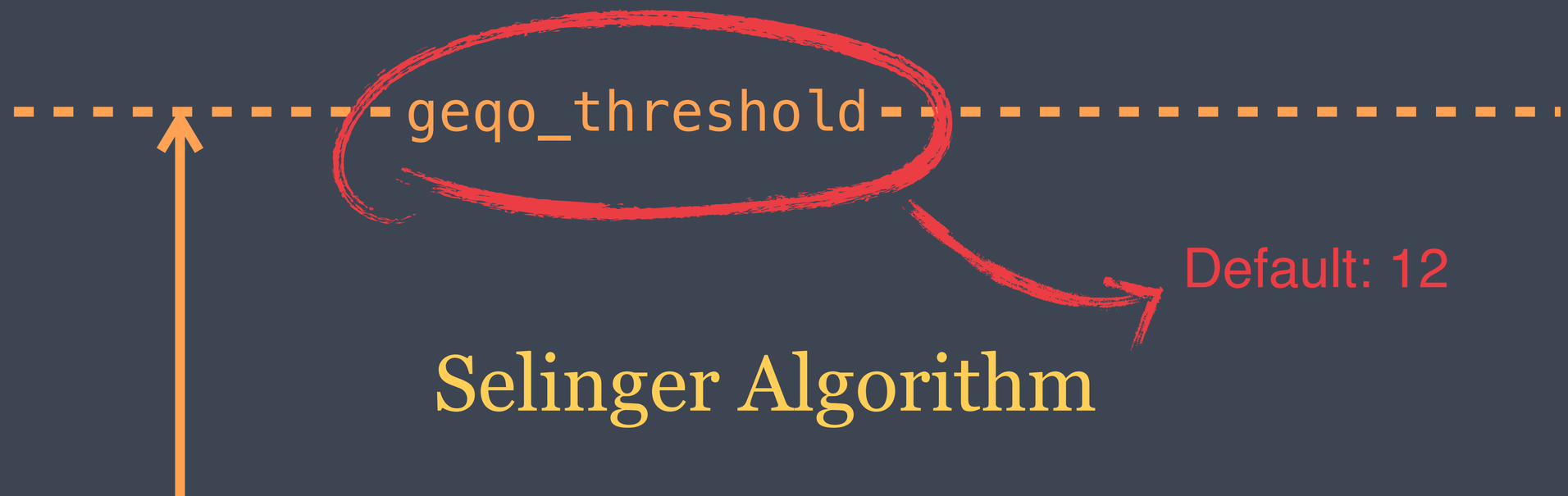
or

Cheapest join order with the correct ordering iff cheaper than cheapest overall + final sort-step

GEQO - Genetic Query Optimizer



GEQO - Genetic Query Optimizer



GEQO

Heuristics are required as search space grows

Travelling salesman type algorithm across the relations

Plans, but not very good plans

```
void
ExecutorStart(QueryDesc *queryDesc, int eflags)
{
    /*
     * In some cases (e.g. an EXECUTE statement) a query execution will skip
     * parse analysis, which means that the query_id won't be reported.  Note
     * that it's harmless to report the query_id multiple times, as the call
     * will be ignored if the top level query_id has already been reported.
     */
    pgstat_report_query_id(queryDesc->plannedstmt->queryId, false);

    if (ExecutorStart_hook)
        (*ExecutorStart_hook) (queryDesc, eflags);
    else
        standard_ExecutorStart(queryDesc, eflags);
}
```

```
void
ExecutorStart(QueryDesc *queryDesc, int eflags)
{
    /*
     * In some cases (e.g. an EXECUTE statement) a query execution will skip
     * parse analysis, which means that the query_id won't be reported.  Note
     * that it's harmless to report the query_id multiple times, as the call
     * will be ignored if the top level query_id has already been reported.
     */
    pgstat_report_query_id(queryDesc->plannedstmt->queryId, false);

    if (ExecutorStart_hook)
        (*ExecutorStart_hook) (queryDesc, eflags);
    else
        standard_ExecutorStart(queryDesc, eflags);
}
```

```
static void
pgss_ExecutorStart(QueryDesc *queryDesc, int eflags)
{
    if (prev_ExecutorStart)
        prev_ExecutorStart(queryDesc, eflags);
    else
        standard_ExecutorStart(queryDesc, eflags);

    /* If query has queryId zero, don't track it. */
    if (pgss_enabled(nesting_level) && queryDesc->plannedstmt->queryId != UINT64CONST(0))
    {
        /* Set up to track total elapsed time in ExecutorRun. */
        if (queryDesc->totaltime == NULL)
        {
            MemoryContext oldcxt;

            oldcxt = MemoryContextSwitchTo(queryDesc->estate->es_query_cxt);
            queryDesc->totaltime = InstrAlloc(1, INSTRUMENT_ALL, false);
            MemoryContextSwitchTo(oldcxt);
        }
    }
}
```

```
static void
pgss_ExecutorStart(QueryDesc *queryDesc, int eflags)
{
    if (prev_ExecutorStart)
        prev_ExecutorStart(queryDesc, eflags);
    else
        standard_ExecutorStart(queryDesc, eflags);

    /* If query has queryId zero, don't track it. */
    if (pgss_enabled(nesting_level) && queryDesc->plannedstmt->queryId != UINT64CONST(0))
    {
        /* Set up to track total elapsed time in ExecutorRun. */
        if (queryDesc->totaltime == NULL)
        {
            MemoryContext oldcxt;

            oldcxt = MemoryContextSwitchTo(queryDesc->estate->es_query_cxt);
            queryDesc->totaltime = InstrAlloc(1, INSTRUMENT_ALL, false);
            MemoryContextSwitchTo(oldcxt);
        }
    }
}
```

```
static void
pgss_ExecutorStart(QueryDesc *queryDesc, int eflags)
{
    if (prev_ExecutorStart)
        prev_ExecutorStart(queryDesc, eflags);
    else
        standard_ExecutorStart(queryDesc, eflags);

    /* If query has queryId zero, don't track it. */
    if (pgss_enabled(nesting_level) && queryDesc->plannedstmt->queryId != UINT64CONST(0))
    {
        /* Set up to track total elapsed time in ExecutorRun. */
        if (queryDesc->totaltime == NULL)
        {
            MemoryContext oldcxt;

            oldcxt = MemoryContextSwitchTo(queryDesc->estate->es_query_cxt);
            queryDesc->totaltime = InstrAlloc(1, INSTRUMENT_ALL, false);
            MemoryContextSwitchTo(oldcxt);
        }
    }
}
```

Undocumented

Hooks are not an API

No clear lines of separation from surrounding backend code

Things can change in minor revs

```
commit 604ffd280b955100e5fc24649ee4d42a6f3ebf35
Author: Tom Lane <tgl@sss.pgh.pa.us>
Date:   Fri May 25 17:54:25 2007 +0000

    Create hooks to let a loadable plugin monitor (or even replace) the planner
    and/or create plans for hypothetical situations; in particular, investigate
    plans that would be generated using hypothetical indexes.  This is a
    heavily-rewritten version of the hooks proposed by Gurjeet Singh for his
    Index Advisor project.  In this formulation, the index advisor can be
    entirely a loadable module instead of requiring a significant part to be
    in the core backend, and plans can be generated for hypothetical indexes
    without requiring the creation and rolling-back of system catalog entries.

    The index advisor patch as-submitted is not compatible with these hooks,
    but it needs significant work anyway due to other 8.2-to-8.3 planner
    changes.  With these hooks in the core backend, development of the advisor
    can proceed as a pgfoundry project.
```

Postgres Packaged Extensions

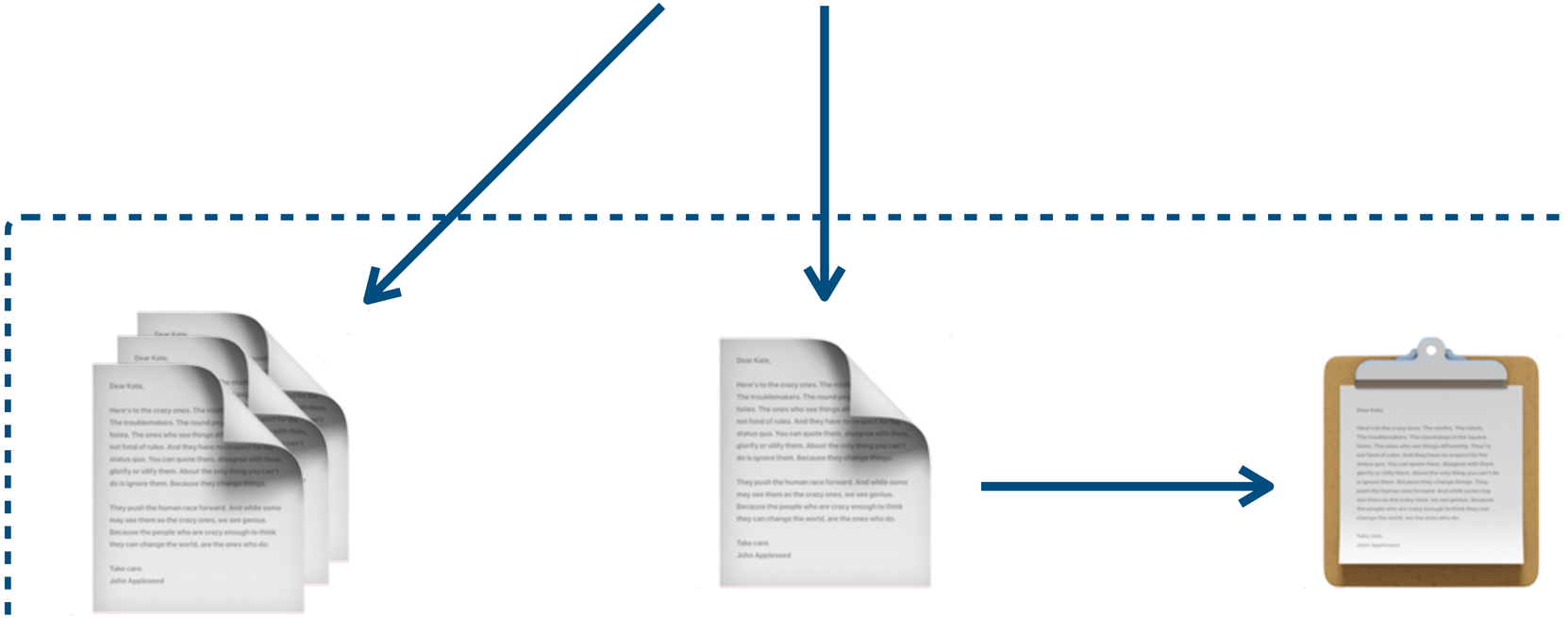
Introduced in 9.0

Contributed by Microsoft employee Dimitri Fontaine (*then 2ndQuadrant*)

Bundle and add SQL objects via SQL commands

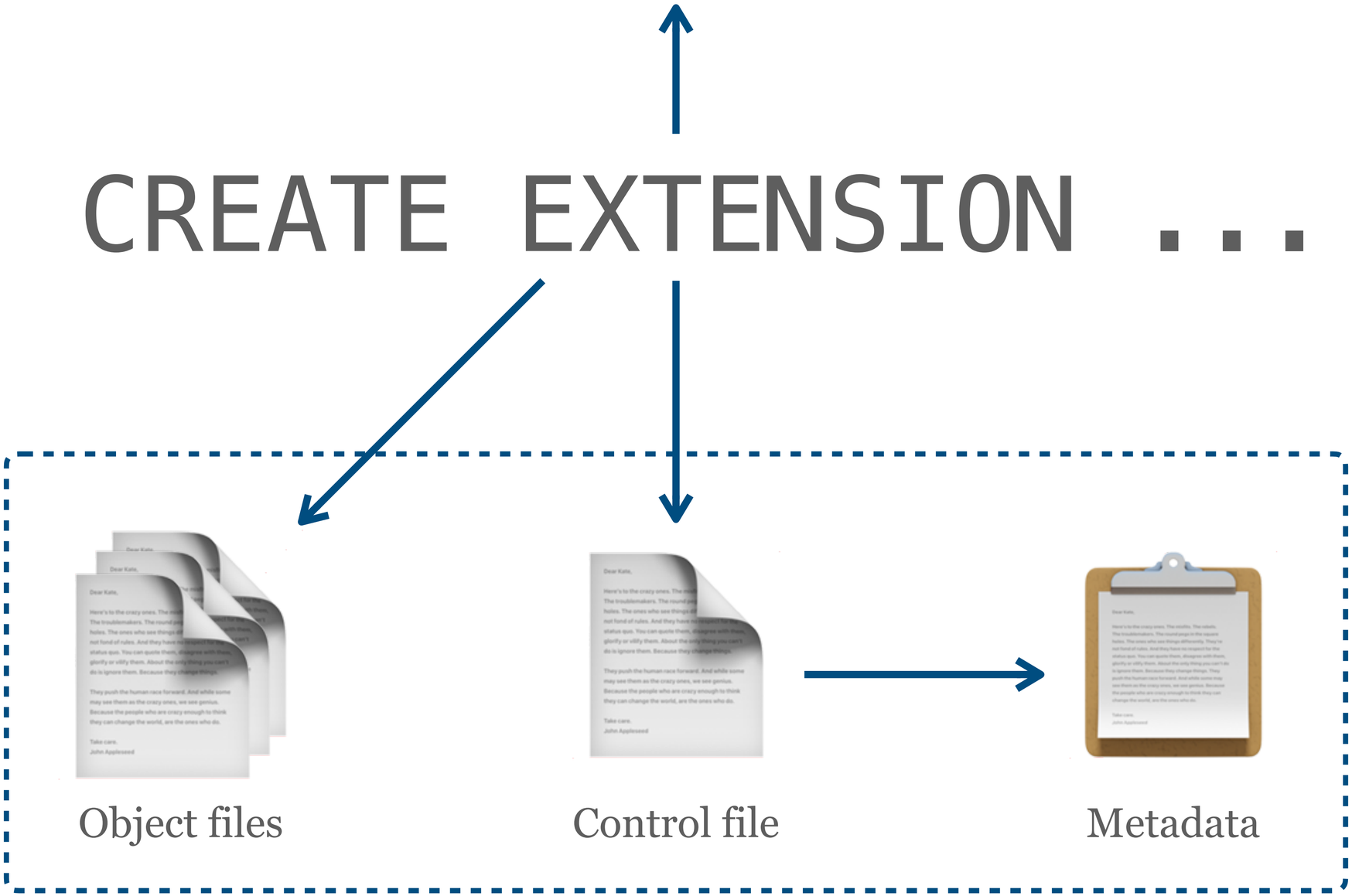
CREATE EXTENSION ...

CREATE EXTENSION ...



pg_catalog.

CREATE EXTENSION ...



Postgres Packaged Extensions

Loose coupling

No archive format, objects are grouped by naming convention

Provides packaging infrastructure, not package management

Control file

Metadata about the extension

Dependencies

Must be in `$SHAREDIR/extension`

Secondary controlfiles for version-specific metadata (*not all*)

Object files

Plain SQL files or compiled objects

At least one SQL file

Named `extension--version.sql`

Extension Upgrades

Update scripts

`ALTER EXTENSION ..`

Named `extension--from--to.sql`

Installed in sequence to reach `to`

PGXS

Buildsystem for compiling extensions

Not a general purpose buildsystem

Not a package manager

Provides infrastructure for creating and installing packages

No distribution or discovery

JIT inlining of extension code

Extensions built with PGXS against a JIT enabled server get bitcode compiled and installed

Can be inlined like core postgres code

```
$ ls lib/bitcode/pg_stat_statements*
lib/bitcode/pg_stat_statements.index.bc

lib/bitcode/pg_stat_statements:
pg_stat_statements.bc
```

```
$ ls lib/bitcode/pg_stat_statements*
lib/bitcode/pg_stat_statements.index.bc

lib/bitcode/pg_stat_statements:
pg_stat_statements.bc

$ llvm-dis lib/bitcode/pg_stat_statements/pg_stat_statements.bc

; ModuleID = 'lib/bitcode/pg_stat_statements/pg_stat_statements.bc'
source_filename = "pg_stat_statements.c"
target datalayout = "e-m:o-p270:32:32-p271:32:32-p272:64:64-i64:64-
target triple = "x86_64-apple-macosx12.0.0"
```

```
$ ls lib/bitcode/pg_stat_statements*
lib/bitcode/pg_stat_statements.index.bc

lib/bitcode/pg_stat_statements:
pg_stat_statements.bc

$ llvm-dis lib/bitcode/pg_stat_statements/pg_stat_statements.bc

; ModuleID = 'lib/bitcode/pg_stat_statements/pg_stat_statements.bc'
source_filename = "pg_stat_statements.c"
target datalayout = "e-m:o-p270:32:32-p271:32:32-p272:64:64-i64:64-
target triple = "x86_64-apple-macosx12.0.0"
```

User defined Aggregates

Support common or new aggregates for datatypes, like `sum(..)`

Initial value

State transition function

Final function

Operators

More than just syntactic sugar for calling function

Instructs the planner

COMMUTATOR, NEGATOR, RESTRICT, JOIN, HASHES, MERGES

Data types

A fairly small and simple C API

in/out function for textual input

recv/send functions for binary input

TOAST in case of variable-length

Data types

```
CREATE FUNCTION foo_in(cstring) ..
CREATE TYPE foo ( .. );
```

Array support is automatically added

Indexing support

Adding a new datatype is trivial, but it can't be indexed just yet

Need an Operator Class to interface with the datatype

Example

A sample datatype for Complex numbers

```
#define Mag(c)  ((c)->x*(c)->x + (c)->y*(c)->y)

static int
complex_abs_cmp_internal(Complex * a, Complex * b)
{
    double      amag = Mag(a),
                bmag = Mag(b);

    if (amag < bmag)
        return -1;
    if (amag > bmag)
        return 1;
    return 0;
}

PG_FUNCTION_INFO_V1(complex_abs_lt);

Datum
complex_abs_lt(PG_FUNCTION_ARGS)
{
    Complex    *a = (Complex *) PG_GETARG_POINTER(0);
    Complex    *b = (Complex *) PG_GETARG_POINTER(1);

    PG_RETURN_BOOL(complex_abs_cmp_internal(a, b) < 0);
}
```

```
CREATE FUNCTION complex_abs_lt(complex, complex) RETURNS bool
   AS '_OBJWD_/complex' LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR < (
   leftarg = complex, rightarg = complex, procedure = complex_abs_lt,
   commutator = > , negator = >= ,
   restrict = scalarltsel, join = scalarltjoinsel
);
```

```
CREATE FUNCTION complex_abs_lt(complex, complex) RETURNS bool
   AS '_OBJWD_/complex' LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR < (
   leftarg = complex, rightarg = complex, procedure = complex_abs_lt,
   commutator = > , negator = >= ,
   restrict = scalarltsel, join = scalarltjoinsel
);
```

Rinse and repeat for the other operators..

```
CREATE OPERATOR CLASS complex_abs_ops
    DEFAULT FOR TYPE complex USING btree AS
        OPERATOR        1       < ,
        OPERATOR        2       <= ,
        OPERATOR        3       = ,
        OPERATOR        4       >= ,
        OPERATOR        5       > ,
        FUNCTION        1       complex_abs_cmp(complex, complex);
```

```
CREATE OPERATOR CLASS complex_abs_ops
    DEFAULT FOR TYPE complex USING btree AS
        OPERATOR        1       < ,
        OPERATOR        2       <= ,
        OPERATOR        3       = ,
        OPERATOR        4       >= ,
        OPERATOR        5       > ,
        FUNCTION        1       complex_abs_cmp(complex, complex);
```

```
INSERT INTO test_complex VALUES ('(56.0,-22.5)', '(-43.2,-0.07)');
INSERT INTO test_complex VALUES ('(-91.9,33.6)', '(8.6,3.0)');

CREATE INDEX test_cplx_ind ON test_complex USING btree(a complex_abs_ops);
```

But.. what about the indexes itself?

Index Access Methods

Postgres has no hardcoded knowledge about indexes, it only knows how to interface with them using the *Index Access Method* API

Indexes are extensions

Index Access Methods

Handler specifying the strategies

Functions for building, inserting, deleting and scanning

Uniqueness

..tables?

Table Access Methods

Postgres doesn't really know much about tables either..

Table Access Method API

Need to support TIDs

Not original, bolted on afterwards

..tables somewhere else?

Foreign Data Wrappers

Formally defined in SQL/MED part of the SQL standard

Only FDW supported, not datalinks

Not the standardized API (*which hardly anyone use anyways*)

Foreign Data Wrappers

Functions for planning, scanning, modifying, locking

Can support parallel execution

Pick and choose in the API based on the underlying service

Discovery

Package Manager

There have been attempts at partial solutions

PGXN, modelled on CPAN, had some traction but is now in life-support

Postgres-as-a-service specific

Other Package Managers

`yum.postgresql.org`,
`apt.postgresql.org`,
`zypp.postgresql.org`

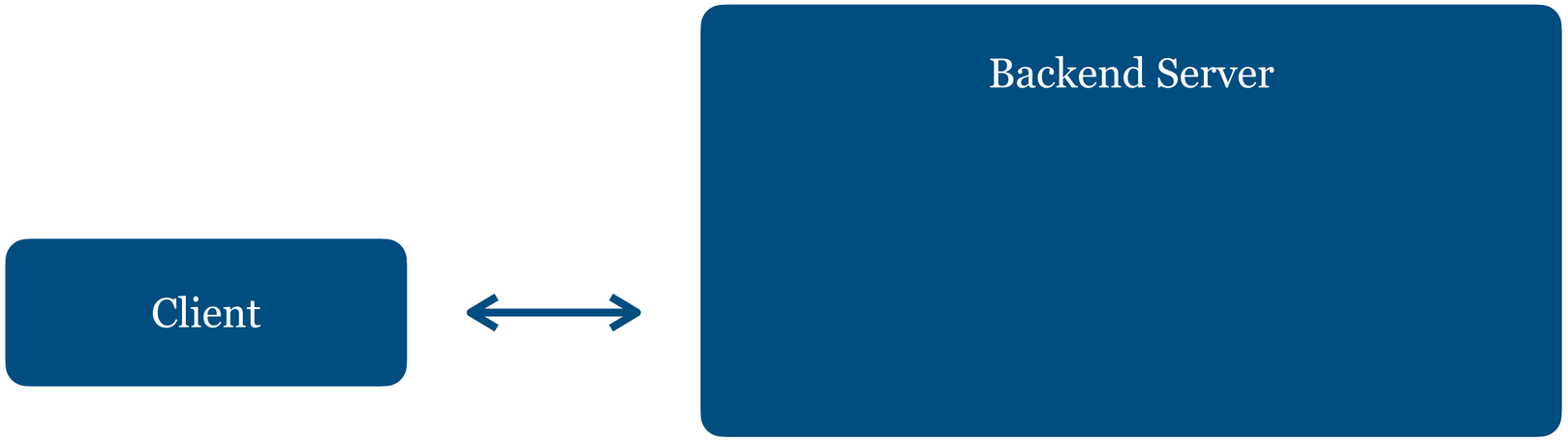
Homebrew, Chocolatey ..

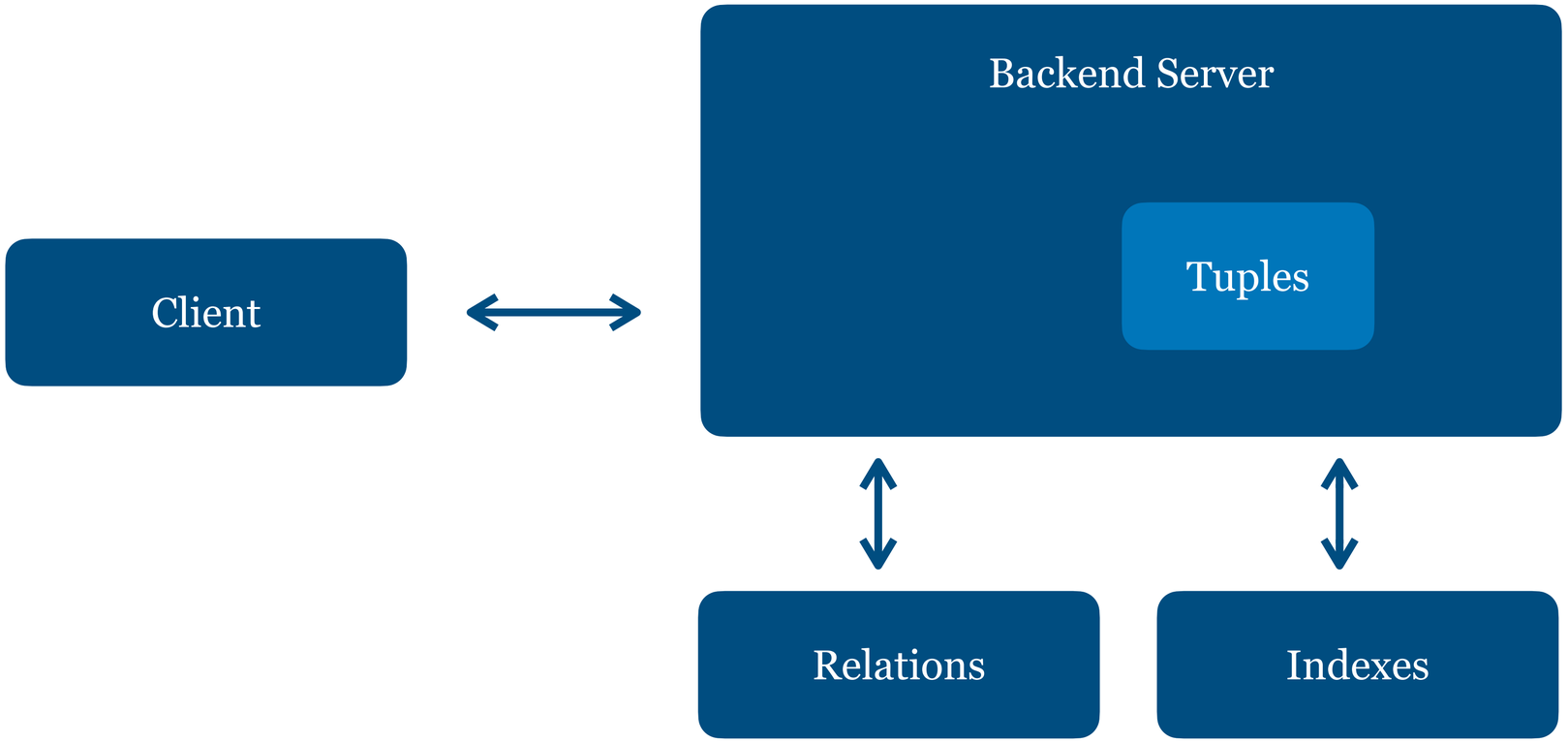
Contrib

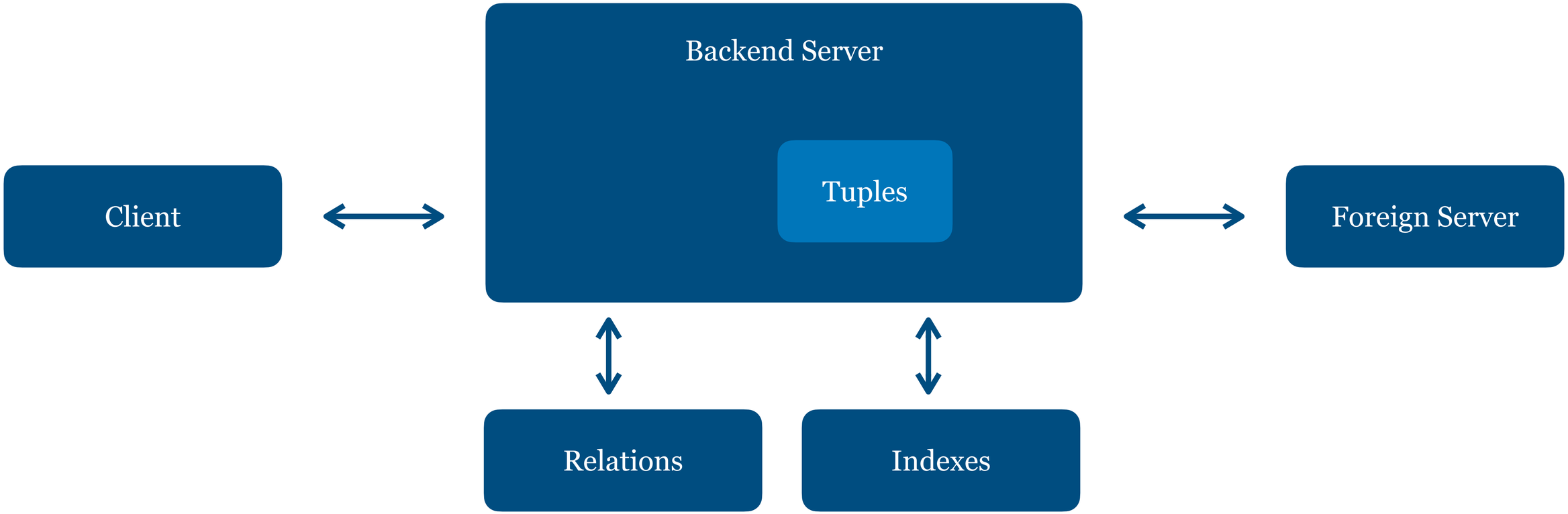
Origins traced to Postgres95, imported a month after the main repo started in August 18 1996

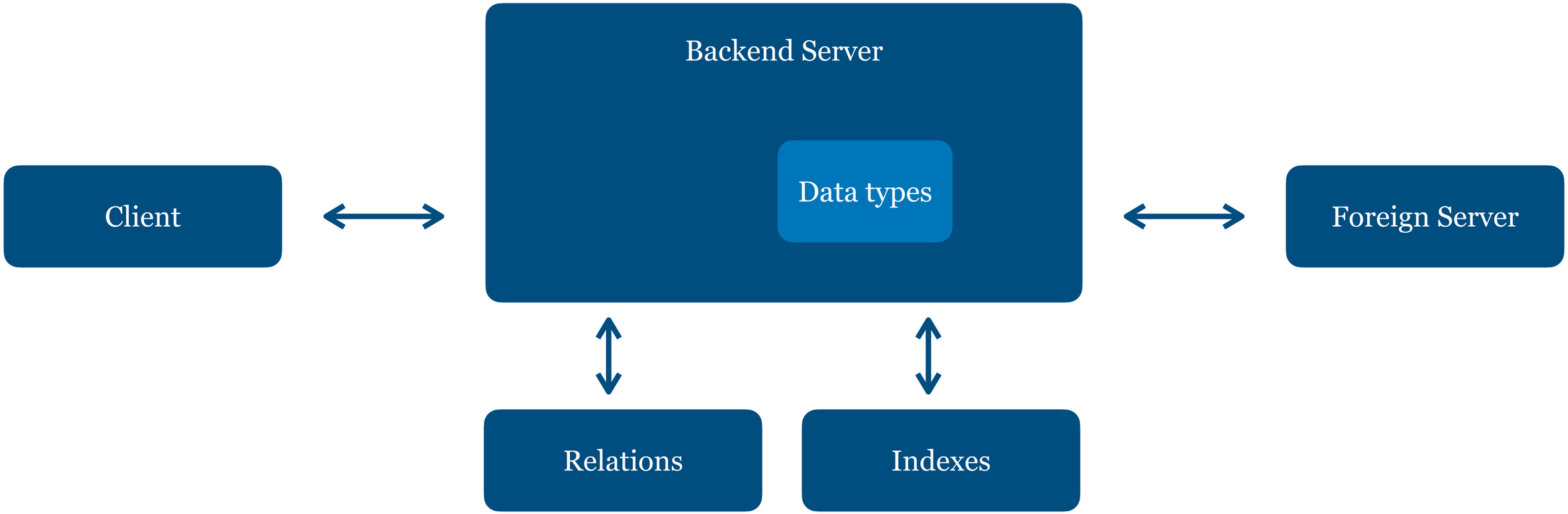
Originally for useful contributed tools, it is no longer used for that

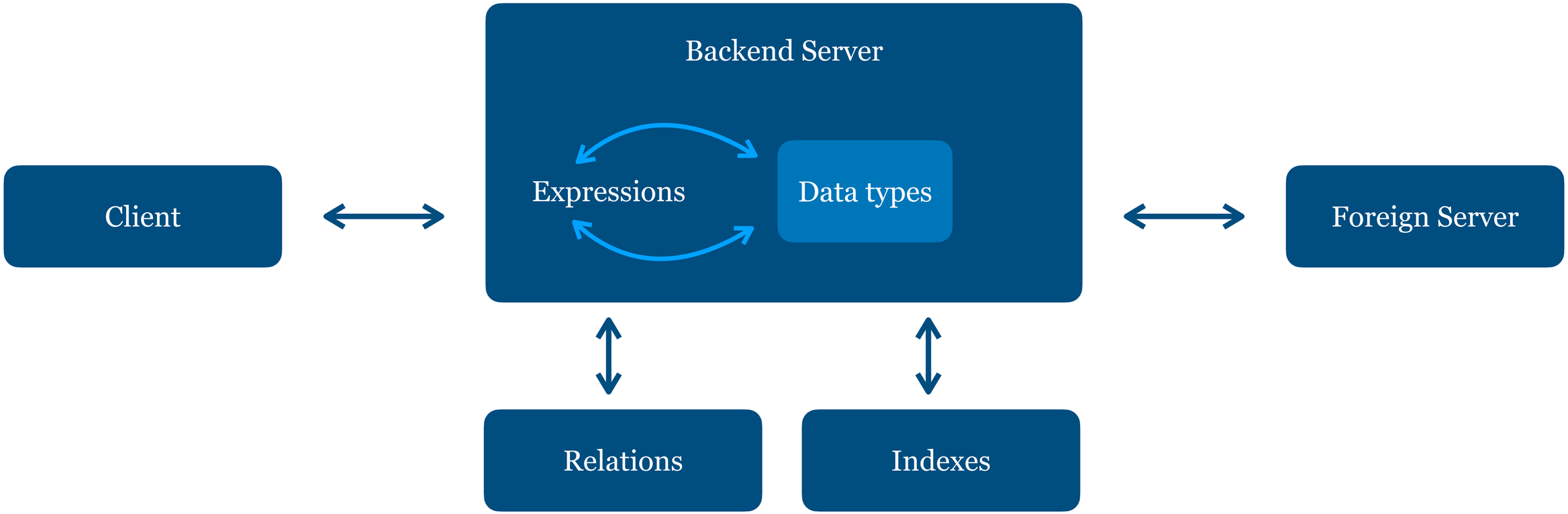
tl;dr

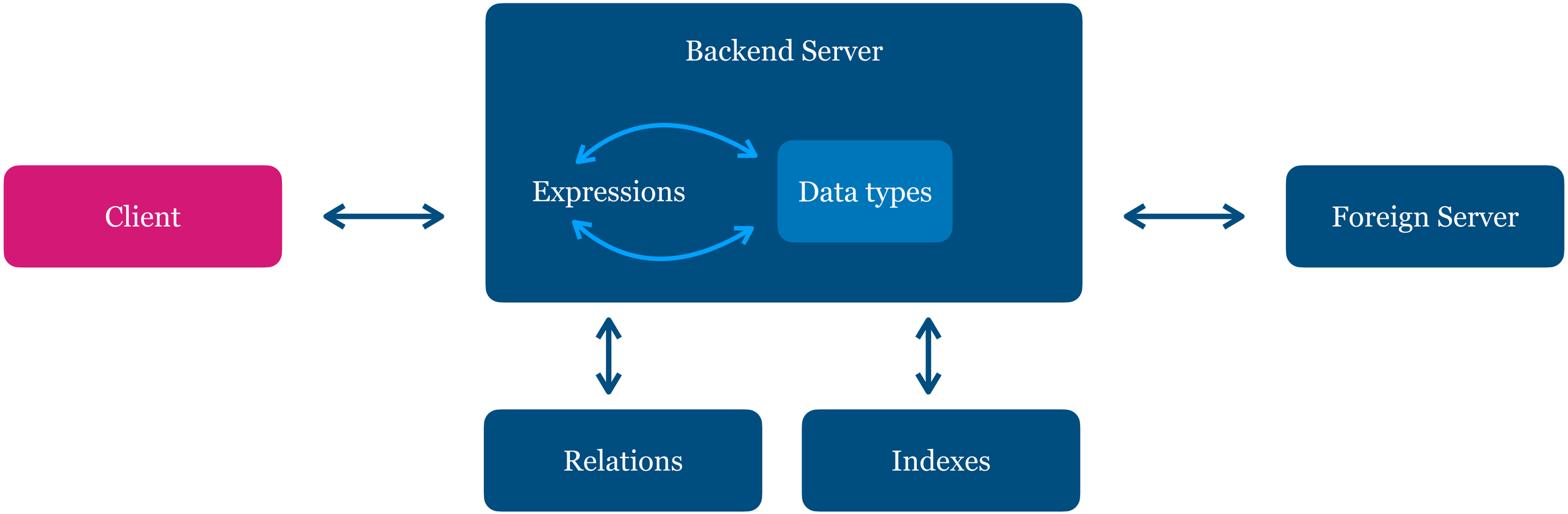

Backend Server
Client

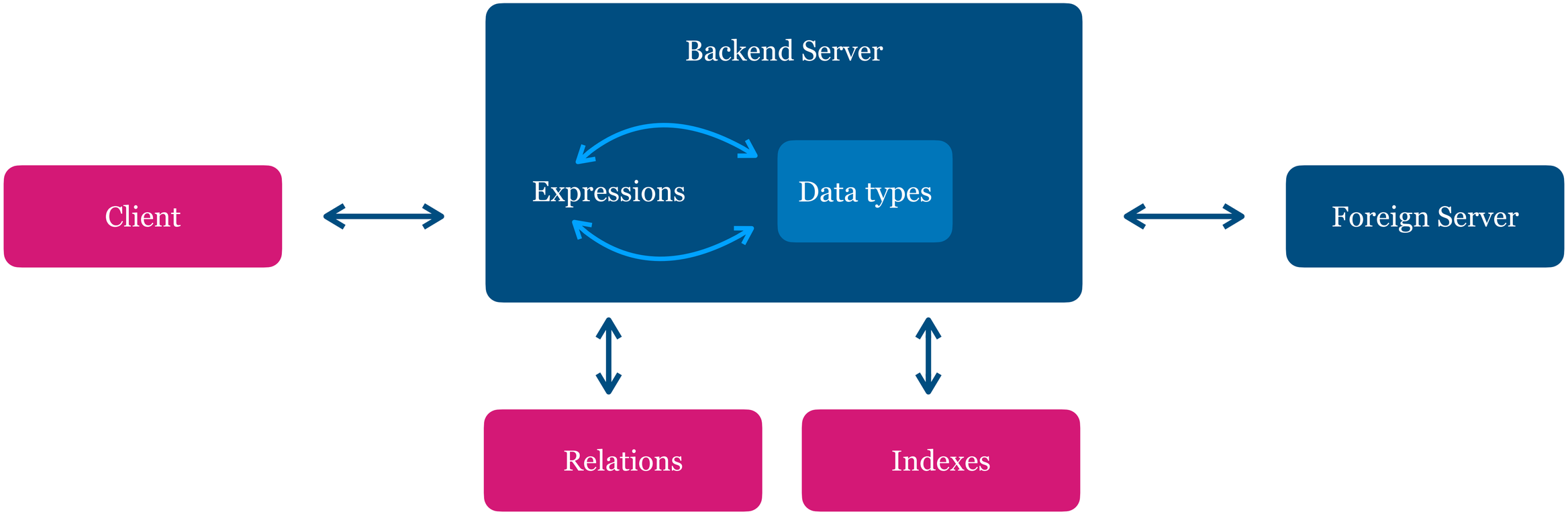

Backend Server
Tuples
Client
Relations
Indexes

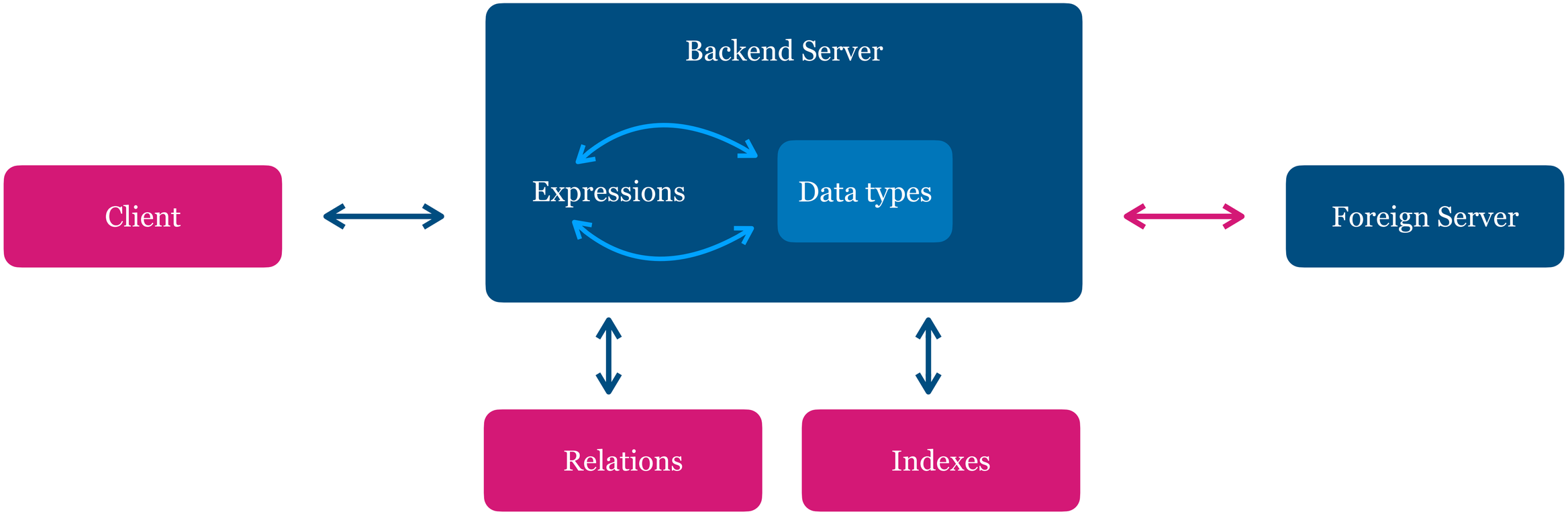

Backend Server
Tuples
Client
Foreign Server
Relations
Indexes

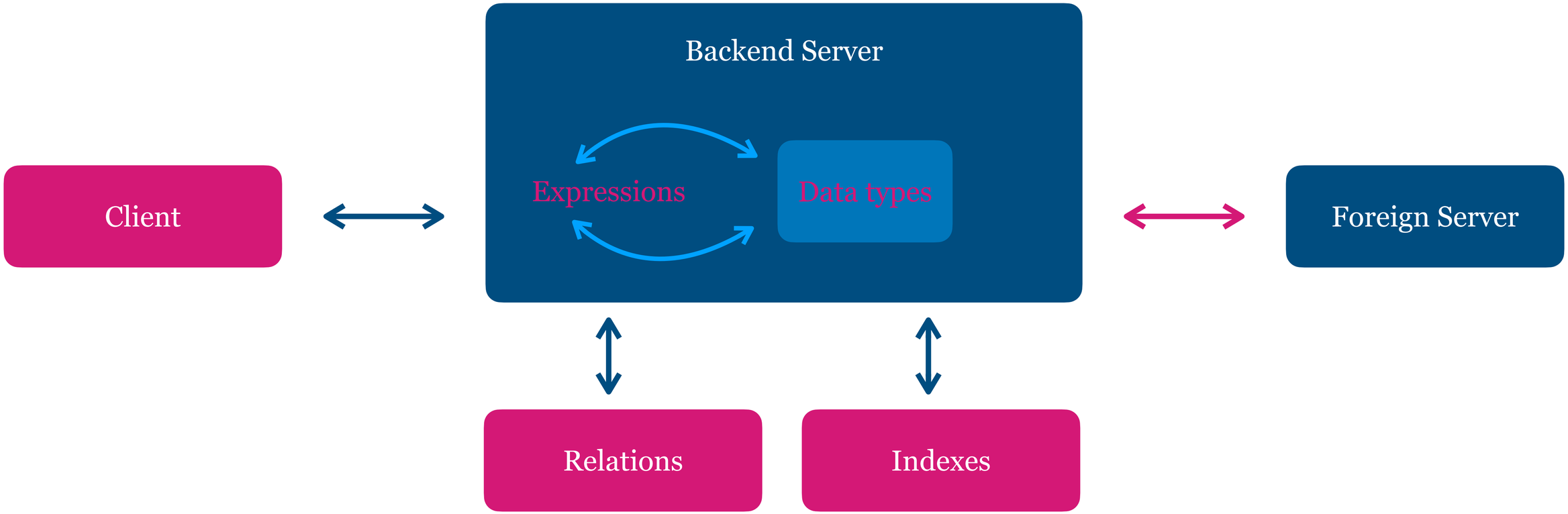

Backend Server
Data types
Client
Foreign Server
Relations
Indexes


Backend Server
Client
Expressions
Data types
Foreign Server
Relations
Indexes


Backend Server
Client
Expressions
Data types
Foreign Server
Relations
Indexes


Backend Server
Expressions
Data types
Client
Foreign Server
Relations
Indexes


Backend Server
Expressions
Data types
Client
Foreign Server
Relations
Indexes


Backend Server
Expressions
Data types
Client
Foreign Server
Relations
Indexes

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system The main design goals of the new system are to

1) provide better support for complex objects,
2) provide user extendibility for data types, operators and access methods,
3) provide facilities for active databases (i e , alerters and triggers) and inferencing including forward- and backward-chaining,
4) simplify the DBMS code for crash recovery,
5) produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips, and
6) make as few changes as possible (preferably none) to the relational model

Microsoft

Thank you!

pg-os-dev@service.microsoft.com

dgustafsson@microsoft.com