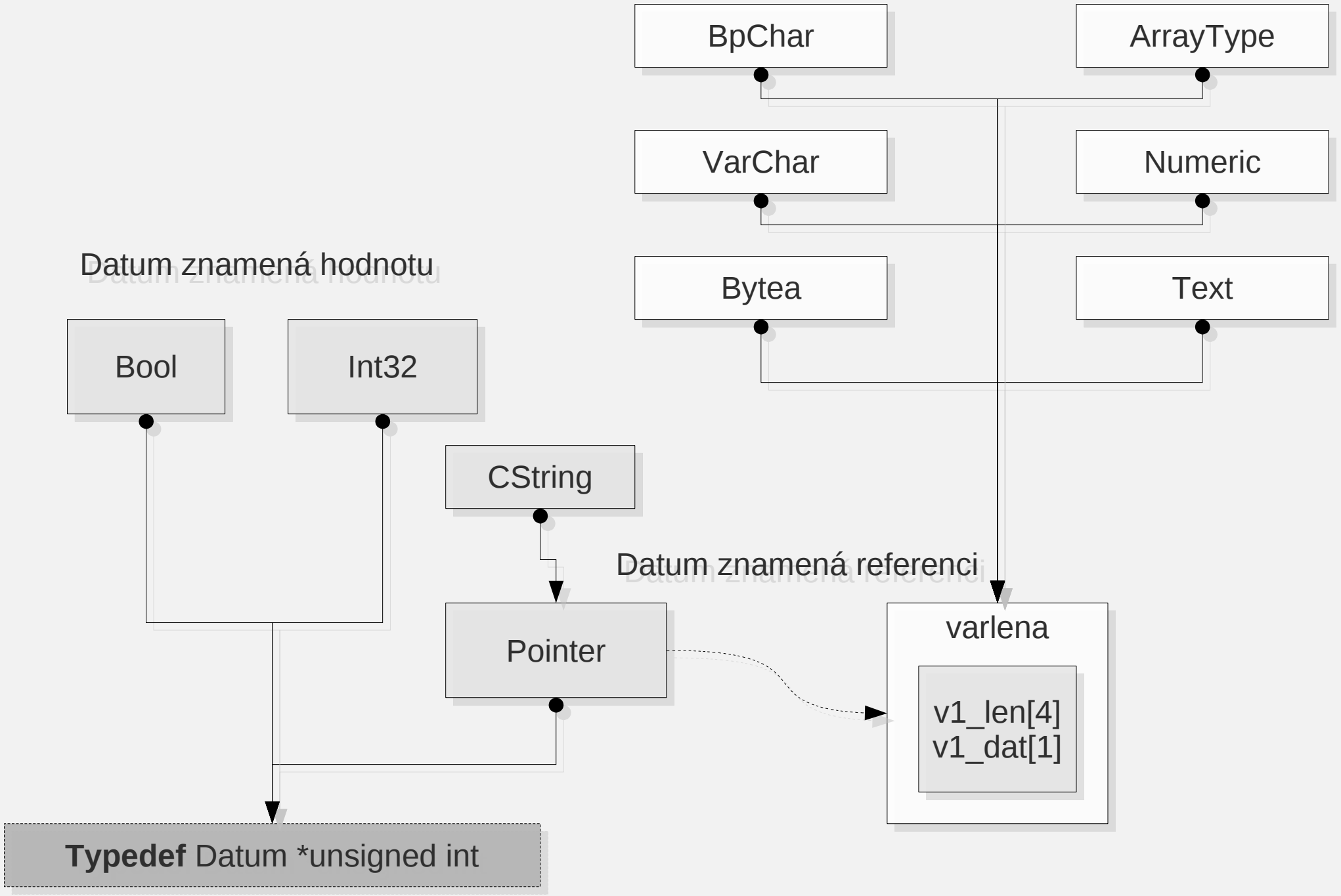
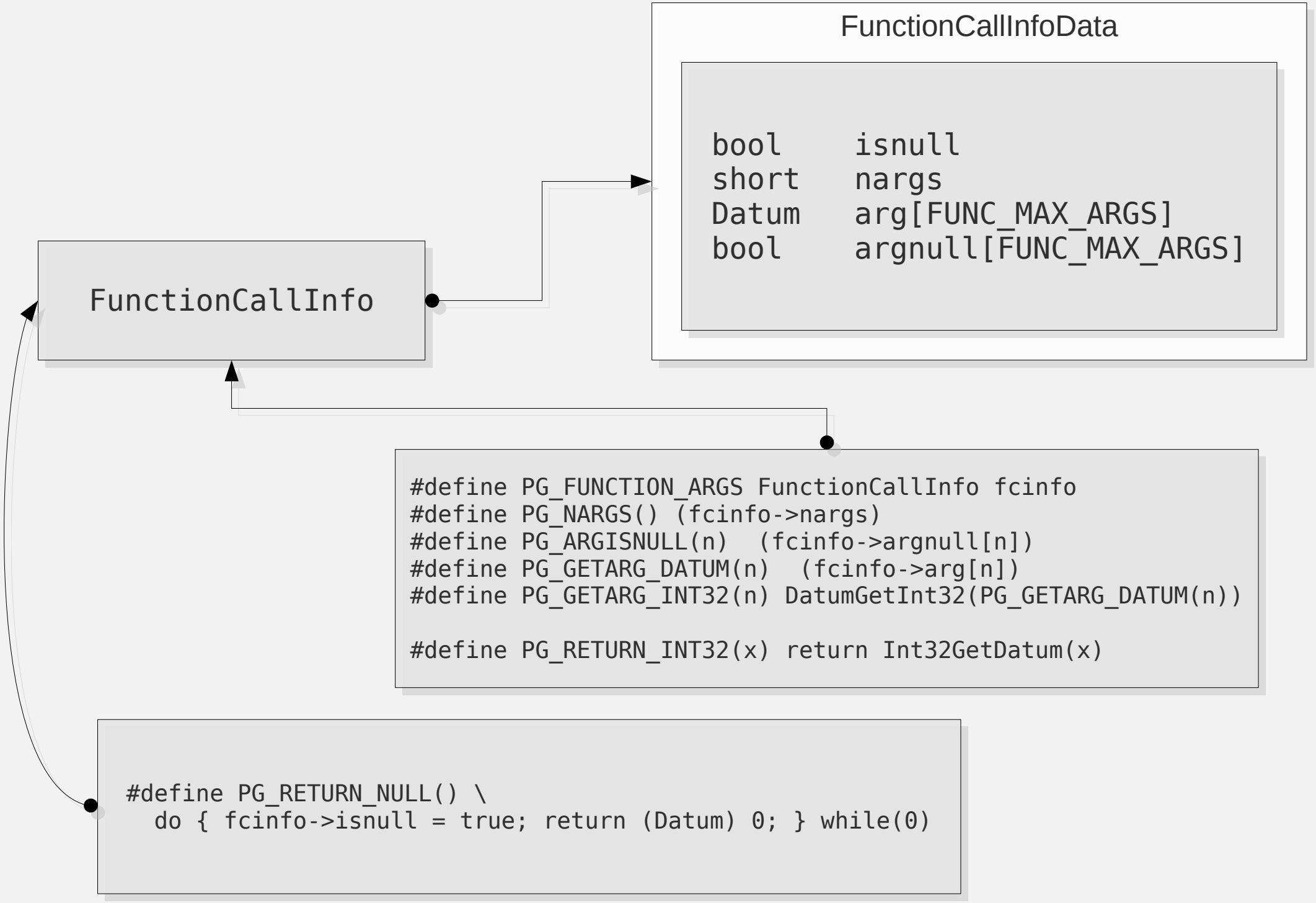
Účel modulů v jazyku C:

- Rozšiřování a upravování stávající funkcionality
  - PLToolbox, Orafce
- Vlastní datové typy
  - PostGIS
  - Hstore
  - Citext
- Zpřístupnění funkcionality z knihoven
  - PL/R
  - PL/Perl, PL/Python
  - xml2 - (contrib)

Modul v C lze použít k obfuskaci kódu


BpChar
ArrayType
VarChar
Numeric
Bytea
Text
Datum znamená hodnotu
Bool
Int32
CString
Datum znamená referenci
Pointer
varlena
v1_len[4]
v1_dat[1]
Typedef Datum *unsigned int


FunctionCallInfoData
bool isnull
short nargs
Datum arg[FUNC_MAX_ARGS]
bool argnull[FUNC_MAX_ARGS]
FunctionCallInfo
#define PG_FUNCTION_ARGS FunctionCallInfo fcinfo
#define PG_NARGS() (fcinfo->nargs)
#define PG_ARGISNULL(n) (fcinfo->argnull[n])
#define PG_GETARG_DATUM(n) (fcinfo->arg[n])
#define PG_GETARG_INT32(n) DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_RETURN_INT32(x) return Int32GetDatum(x)
#define PG_RETURN_NULL() \
do { fcinfo->isnull = true; return (Datum) 0; } while(0)

```
Datum
Int32_sum(PG_FUNCTION_ARGS)
{
    int a = PG_GETARG_INT32(0);
    int b = PG_GETARG_INT32(1);

    PG_RETURN_INT32(a + b)
}
```

FUNKCE MUSÍ BÝT OZNAČENA ATRIBUTEM **STRICT**

```
Datum
Int32_sum(FunctionCallInfo fcinfo)
{
    int a = DatumGetInt32(fcinfo->arg[0]);
    int b = DatumGetInt32(fcinfo->arg[1]);

    Return Int32GetDatum(a + b);
}
```

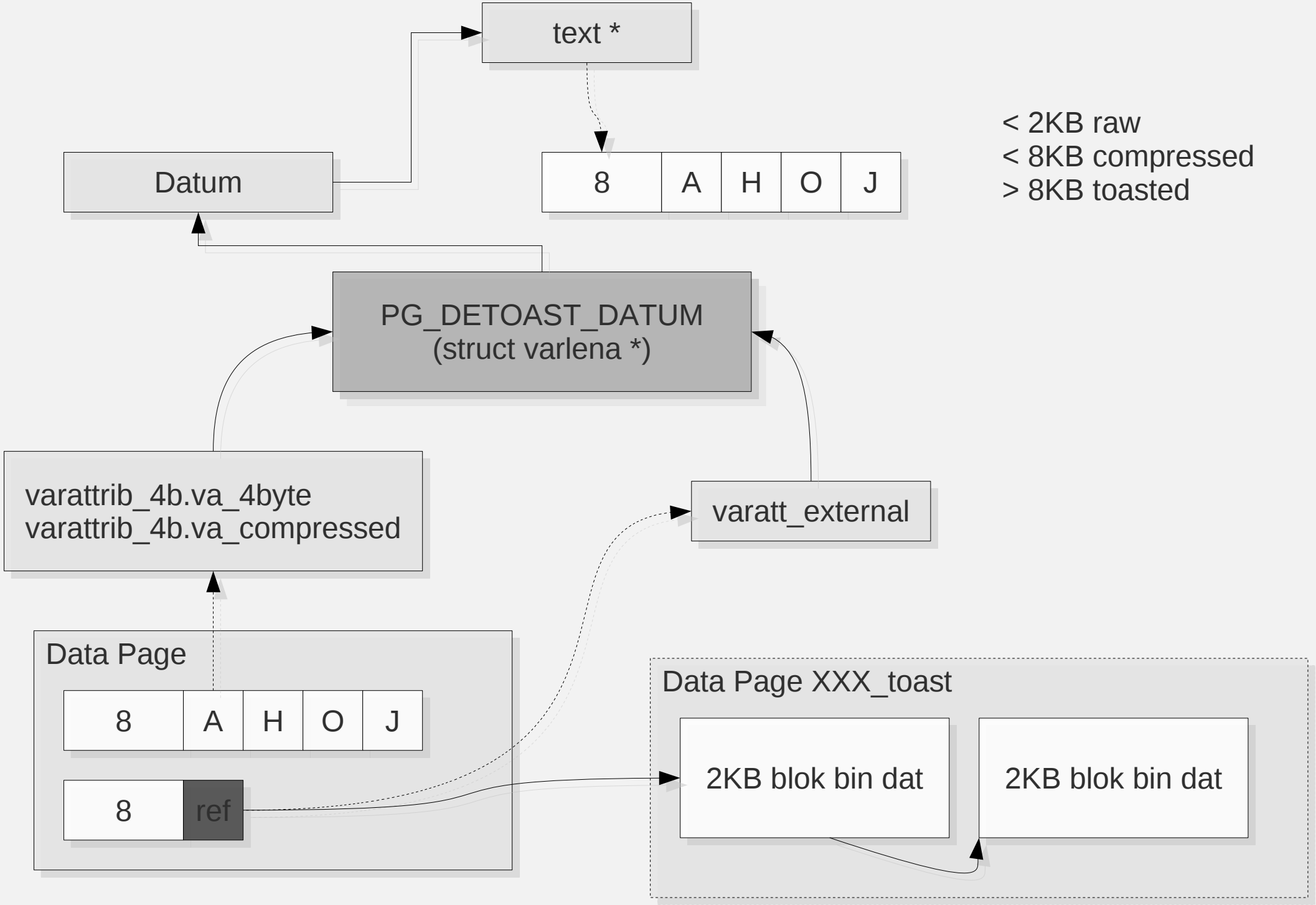
```
#define PointerGetDatum(X) ((Datum) X)
#define DatumGetPointer(X) ((Pointer) X)
#define Int32GetDatum(X) ((Datum) SET_4_BYTES(X))
#define DatumGetInt32(X)  ((int32) GET_4_BYTES(X))
#define CStringGetDatum(X) (PointerGetDatum(X))
#define DatumGetCString(X) ((char *) DatumGetPointer(X))

#define PG_DETOAST_DATUM(X) \
  pg_detoast_datum((struct varlena *) DatumGetPointer(X))

#define DatumGetTextP(X) ((text *) PG_DETOAST_DATUM(X))
#define DatumGetByteaP(X) ((bytea *) PG_DETOAST_DATUM(X))

#define PG_GETARG_TEXT_P(X) DatumGetTextP(PG_GETARG_DATUM(X))
#define PG_RETURN_TEXT_P(X) PG_RETURN_POINTER(X)
```

FUNKCE MUSÍ BÝT OZNAČENA ATRIBUTEM **STRICT**

```
Datum
concat(PG_FUNCTION_ARGS)
{
    text *a = PG_GETARG_TEXT_P(0);
    text *b = PG_GETARG_TEXT_P(1);
    text *result;
    int len1, len2, len;
    char *ptr;

    len1 = VARSIZE(a) – VARHDRSZ;
    len2 = VARSIZE(b) – VARHDRSZ;
    Len = len1 + len2 + VARHDRSZ;

     result = (text *) palloc(len);
    ptr = VARDATA(result);
    memcpy(ptr, VARDATA(a), len1);
    memcpy(ptr + len1, VARDATA(b), len2);
    SET_VARSIZE(result, len);

    PG_RETURN_TEXT_P(result);
}
```

```
Text *a = PG_GETARG_TEXT_P(0);
```

```
text *a = ((text *) pg_detoast_datum(
       (struct *varlena) DatumGetPointer((fcinfo->arg[0]))))
```


text *
Datum
8
A
H
O
J
< 2KB raw
< 8KB compressed
> 8KB toasted
PG_DETOAST_DATUM
(struct varlena *)
varattrib_4b.va_4byte
varattrib_4b.va_compressed
varatt_external
Data Page
8
A
H
O
J
8
ref
Data Page XXX_toast
2KB blok bin dat
2KB blok bin dat

```
#include “lib/stringinfo”

Datum
HelloFunc(PG_FUNCTION_ARGS)
{
    StringInfoData ds;
    char *str;

    initStringInfo(&ds);
    str = text_to_cstring(PG_GETARG_TEXT_P(0));
    appendStringInfo(&ds, “Hello %s“, str);

    PG_RETURN_TEXT_P(cstring_to_text_with_len(ds.data,
                                              ds.len));
}
```

StringInfoData

```
char    *data;
Int      len;
Int      maxlen;
Int      cursor;
```

```
CREATE OR REPLACE FUNCTION hello_func(text)
RETURNS text AS '$libdir/hello', 'pst_hello_func'
LANGUAGE C IMMUTABLE STRICT;
```

```
        ...

     appendStringInfo(&ds, “Hello %s“, str);

    result = cstring_to_text_with_len(ds.data, ds.len);
    result = DatumGetTextP(
               DirectFunctionCall1(upper,
                  TextPGetDatum(result)));

    PG_RETURN_TEXT_P(result);
}
```

```
#define InitFunctionCallInfoData(Fcinfo, Flinfo, Nargs, ..) \
  do { \
    (Fcinfo).isnull = false; \
    (Fcinfo).nargs = Nargs; \
  } while (0)

Datum
DirectFunctionCall1(PGFunction func, Datum arg1)
{
    FunctionCallInfoData fcinfo;
    Datum   result;

    InitFunctionCallInfoData(&fcinfo, NULL, 1, ..);

    fcinfo.arg[0] = arg1;
    fcinfo.argisnull[0] = false;

    result = (*func) (fcinfo)
    if (fcinfo.isnull)
        elog(ERROR, “function %p returns NULL”, func);
    return result;
}
```

POZOR – funkce musí být **STRICT**

POZOR – používá se UTF8

```
Datum
pst_left(PG_FUNCTION_ARGS)
{
    text *txt = PG_GETARG_TEXT_P(0);
    text *result;
    char *str = VARDATA(txt);
    int len = VARSIZE(txt) – VARHDRSZ;
    char *ptr = *str;
    int processed = 0;
    int n = PG_GETARG_INT32(1);
    int mblen;

    while (processed < len)
    {
        If (n-- == 0)
            break;
        mblen = pg_mblen(ptr);
        ptr += mblen;
        processed += mblen;
    }

    result = cstring_to_text_with_len(str, processed);
    PG_RETURN_TEXT_P(result);
}
```

```
/*
 * Bez signatury je funkce volána
 * s klasickou C (0) konvencí předávání
 * parametrů! Tudíž pokud dojde k odkazu
 * na fcinfo, dojde k pádu aplikace.
 */
PG_FUNCTION_INFO_V1(pst_left);

Datum
pst_left(PG_FUNCTION_ARGS)
{
     ...
   PG_RETURN_TEXT_P(result);
}
```

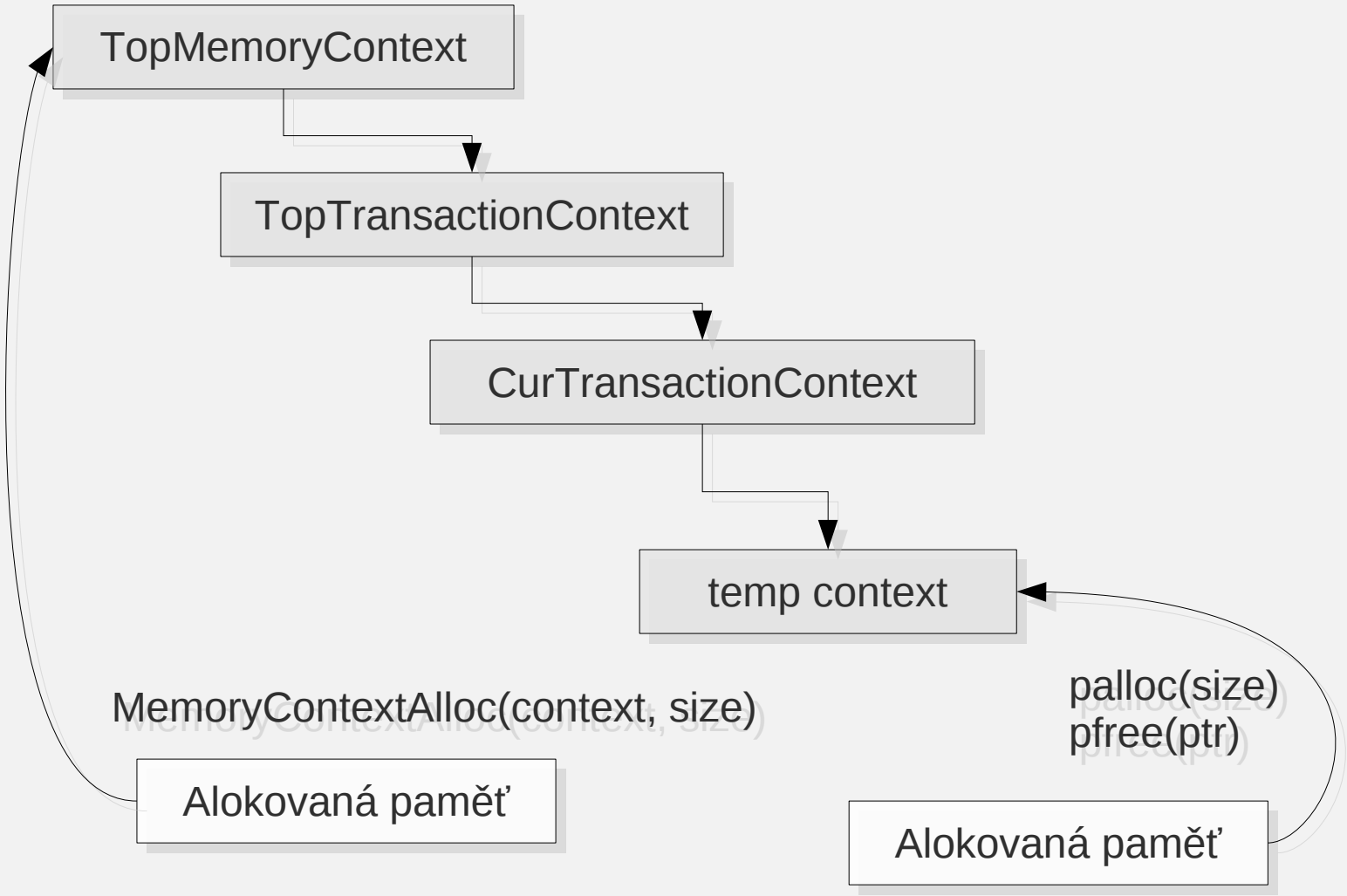
```
const Pg_finfo_record *
pg_finfo_pst_left (void)
{
    static const pg_finfo_record myfinfo = {1};
    return &myfinfo;
}
```

```
const Pg_finfo_record *
fetch_finfo_record(void *filehandle, char *funcname)
{
    infofuncname = (char *) palloc(strlen(funcname) + 10);
    strcpy(infofuncname, "pg_finfo_");
    strcat(infofuncname, funcname);

    infofunc = lookup_external_function(filehandle,
                                        Infofuncname);
    if (infofunc == 0)
    {
        /* version 0; */
    }
    inforec = (*infofunc)();

    switch (inforec->api_version)
        ...
}
```

```
PG_FUNCTION_INFO_V1(pst_left);

Datum
pst_left(PG_FUNCTION_ARGS)
{
    text *txt = PG_GETARG_TEXT_P(0);
    int n = PG_GETARG_INT32(1);

    ...

    PG_RETURN_TEXT_P(result);
}
```

```
CREATE OR REPLACE FUNCTION left(text, int)
RETURNS text AS '$libdir/hello', 'pst_left'
LANGUAGE C IMMUTABLE STRICT;
```

- Nepodceňovat varování překladače
- Nezapomínat na signaturu V1 volající konvence
- Překlad PostgreSQL s parametrem –**enable-cassert**
- Echo – elog(NOTICE, “...”)
- Je třeba zvládnout základy gdb – **bt** backtrace z core dumpu


TopMemoryContext
TopTransactionContext
CurTransactionContext
temp context
MemoryContextAlloc(context, size)
Alokovaná paměť
palloc(size)
pfree(ptr)
Alokovaná paměť

```
MemoryContext oldctx;
MemoryContext tmpctx;

tmpctx = AllocSetContextCreate(CurrentMemoryContext,
                               "plpsm execution context",
                               0, 8 * 1024, 8 * 1024 * 1024);
oldctx = MemoryContextSwitchTo(tmpctx);
  ...

ptr = palloc(sizeof(..));
  ...
MemoryContextReset(tmpctx);
MemoryContextSwitchTo(oldctxt);
```

```
/*
 * Pokud pracujeme s rozumě dlouhými řetězci mimo cyklu nemusíme explicitně
 * uvolňovat alokovanou paměť. Po vykonání funkce executor provede reset
 * aktuálního paměťového kontextu. Pozor – k tomu nedochází při přímém volání
 * funkce skrze DirectFunctionCall.
 */
#include “lib/stringinfo”

Datum
HelloFunc(PG_FUNCTION_ARGS)
{
    StringInfoData ds;
    char *str;

    /* paměť je alokována string builderem */
    initStringInfo(&ds);

    /*
     * text_to_cstring alokuje paměť, případně
     * DETOAST alokuje další paměť a pracujeme
     * s kopií dat.
     */
    str = text_to_cstring(PG_GETARG_TEXT_P(0));
    appendStringInfo(&ds, “Hello %s“, str);

    /* cstring_to_text_with_len alokuje paměť */
    PG_RETURN_TEXT_P(cstring_to_text_with_len(ds.data,
                                              ds.len));
}
```

```
#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif
```

```
#define PG_MODULE_MAGIC \
const Pg_magic_struct * \
Pg_magic_func(void) \
{ \
        static const Pg_magic_struct Pg_magic_data = { \
                                           sizeof(Pg_magic_struct), \
                                           PG_VERSION_NUM / 100, \
                                           FUNC_MAX_ARGS, \
                                           INDEX_MAX_KEYS, \
                                           NAMEDATALEN, \
                                           FLOAT4PASSBYVAL, \
                                           FLOAT8PASSBYVAL \
                                       }; \
        return &Pg_magic_data; \
} \
```

```
/* Check the magic function to determine compatibility */
magic_func = (PGModuleMagicFunction)
pg_dlsym(file_scanner->handle, PG_MAGIC_FUNCTION_NAME_STRING);
if (magic_func)
{
       const Pg_magic_struct *magic_data_ptr = (*magic_func) ();

       if (magic_data_ptr->len != magic_data.len ||
          memcmp(magic_data_ptr, &magic_data, magic_data.len) != 0)
       {
              /* copy data block before unlinking library */
              Pg_magic_struct module_magic_data = *magic_data_ptr;
```

- Nepodceňovat varování překladače
- **Nezapomínat na signaturu modulu**
- Nezapomínat na signaturu V1 volající konvence
- Překlad PostgreSQL s parametrem –**enable-cassert**
- Echo – elog(NOTICE, “...”)
- Je třeba zvládnout základy gdb – **bt** backtrace z core dumpu

Pozor – nefunguje!

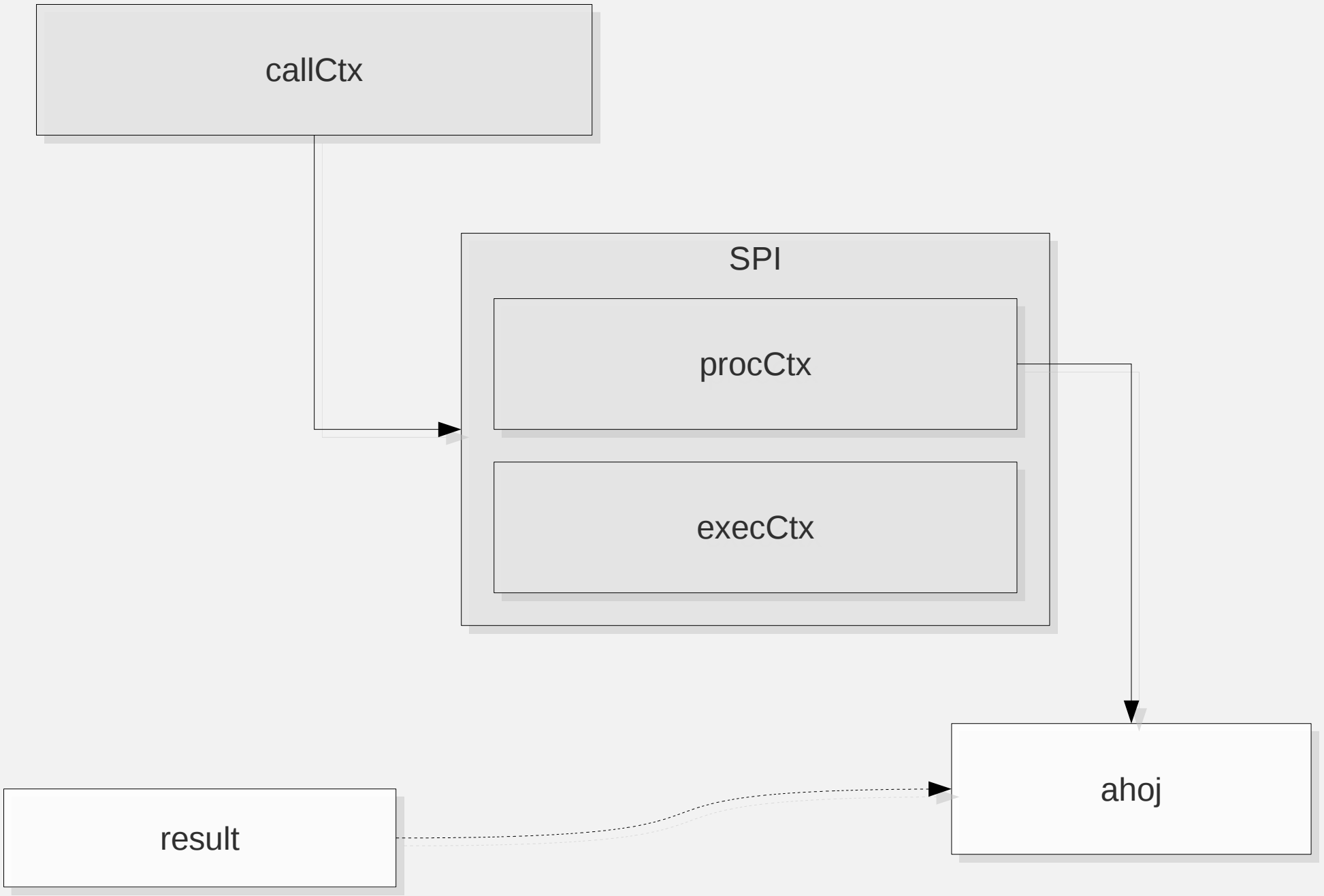
```
char *result = NULL;
char *query = text_to_cstring(PG_GETARG_TEXT_P(0));

SPI_connect();

ret = SPI_exec(query, 0);
if (ret > 0 && SPI_tuptable != NULL)
{
        TupleDesc tupdesc = SPI_tuptable->tupdesc;
        SPITupleTable *tuptable = SPI_tuptable;
        HeapTuple tuple = tuptable->vals[0];

        if (tupdesc->natts > 1)
                elog(ERROR, “Query returned more columns”);

        if (SPI_processed > 1)
                elog(ERROR, “Query returned more rows”);
        else if (SPI_processed == 1)
                result = SPI_getvalue(tuple, tupdesc, 1);
}
SPI_finish();
if (result != NULL)
        PG_RETURN_TEXT_P(cstring_to_text(result))
else
        PG_RETURN_NULL();
```

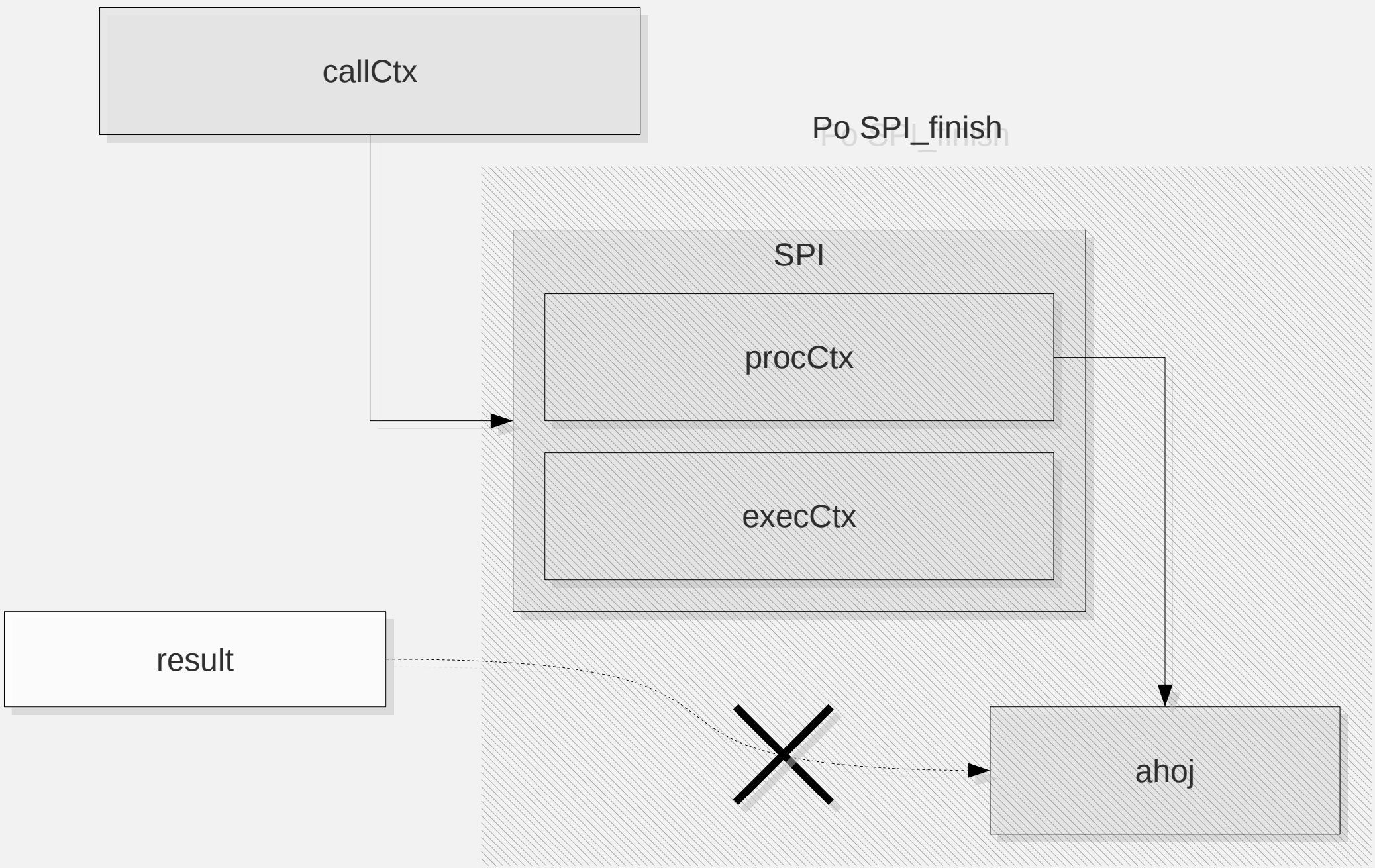
```
/* nefunkční */
SPI_finish();
if (result != NULL)
       /* je zkopírován neplatný blok paměti */
       PG_RETURN_TEXT_P(cstring_to_text(result))
else
       PG_RETURN_NULL();
```

špatně

```
if (result != NULL)
{
   spiCtx = MemoryContextSwitchTo(callCtx);
   result = cstring_to_text(result);
   MemoryContextSwitchTo(spiCtx);
}
SPI_finish();
if (result)
       PG_RETURN_TEXT_P(result);
else
       PG_RETURN_NULL();
```

OK


callCtx
SPI
procCtx
execCtx
ahoj
result


callCtx
Po SPI_finish
SPI
procCtx
execCtx
result
ahoj

```
postgres=# select eval('select $1 + $2', 10, 3.14);
 eval
-------
 13.14
(1 row)

postgres=# select eval('select $1 || '' '' || $2', 'Hello', 'World');
    eval
-------------
 Hello World
(1 row)

postgres=# select eval('select upper($1 || '' '' || $2)', 'Hello', 'World');
    eval
-------------
 HELLO WORLD
(1 row)
```

- Nepodceňovat varování překladače
- Nezapomínat na signaturu modulu
- Nezapomínat na signaturu V1 volající konvence
- Překlad PostgreSQL s parametrem –**enable-cassert**
- Echo – elog(NOTICE, “...”)
- Je třeba zvládnout základy gdb – **bt** backtrace z core dumpu
- **Když něco nefunguje, zamyslet se nad správou paměti**