- PL/pgSQL je **jednoduchý** interpret
- Každý výraz se převádí na SQL příkaz
- Každý SQL příkaz má svůj vlastní prováděcí plán
- Výsledek každého dotazu je transformován

ČÍM MÉNĚ KÓDU TÍM LÉPE
OVŠEM POZOR NA ČITELNOST

```
IF jmeno IS NOT NULL THEN
      s := jmeno;
END IF;
IF prijmeni IS NOT NULL THEN
      IF s IS NOT NULL THEN
            s := s || ' ';
      END IF;
      s := s || prijmeni;
END IF;
```

144 ms

74 ms

```
s := COALESCE(jmeno || ' ' || prijmeno, jmeno);
```

Přepis do SQL funkce: 20 ms

```
CREATE OR REPLACE FUNCTION public.f1(text, text)
 RETURNS text
 LANGUAGE plpgsql
AS $function$#option dump
declare s text;
begin
  if $1 is not null then
    s := $1;
  end if;
  if $2 is not null then
    if s is not null then
      s := s || ' ';
    end if;
    s := s || $2;
  end if;
  return s;
end;
$function$
```

```
Function's data area:
    entry 0: VAR $1              type text (typoid 25) atttypmod -1
    entry 1: VAR $2              type text (typoid 25) atttypmod -1
    entry 2: VAR found             type bool (typoid 16) atttypmod -1
    entry 3: VAR s               type text (typoid 25) atttypmod -1

Function's statements:
  3:BLOCK <<*unnamed*>>
  4:  IF 'SELECT $1 is not null' THEN
  5:    ASSIGN var 3 := 'SELECT $1'
      ENDIF
  7:  IF 'SELECT $2 is not null' THEN
  8:    IF 'SELECT s is not null' THEN
  9:      ASSIGN var 3 := 'SELECT s || ' ''
      ENDIF
 11:    ASSIGN var 3 := 'SELECT s || $2'
      ENDIF
 13:  RETURN 'SELECT s'
    END -- *unnamed*
```

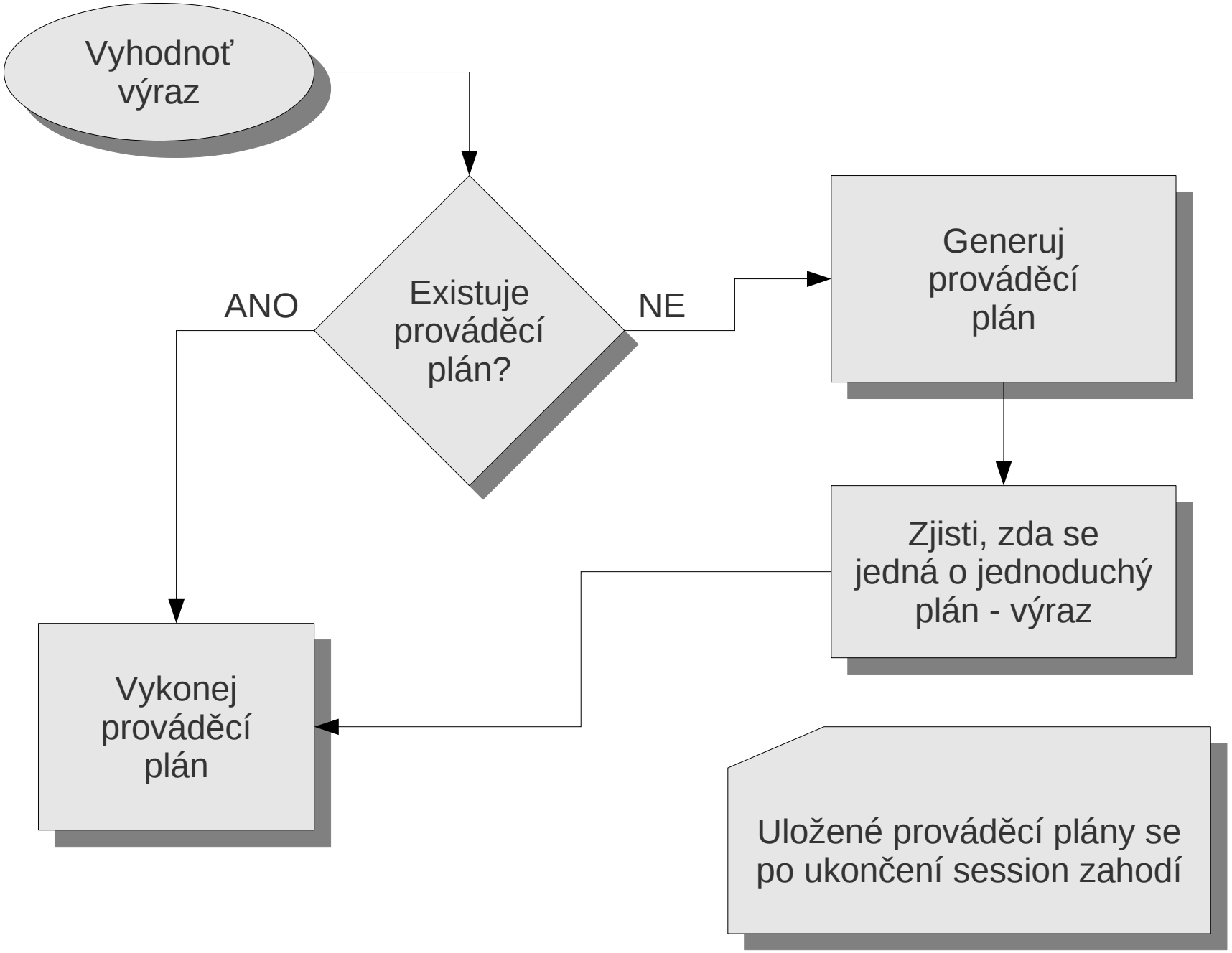
```
CREATE OR REPLACE FUNCTION public.f2(text, text)
 RETURNS text
 LANGUAGE plpgsql
AS $function$#option dump
begin
  return coalesce($1 || ' ' || $2, $1);
end;
$function$
```

```
Function's data area:
    entry 0: VAR $1             type text (typoid 25) atttypmod -1
    entry 1: VAR $2             type text (typoid 25) atttypmod -1
    entry 2: VAR found            type bool (typoid 16) atttypmod -1

Function's statements:
  2:BLOCK <<*unnamed*>>
  3:  RETURN 'SELECT coalesce($1 || ' ' || $2, $1)'
    END -- *unnamed*
```

- Uložené procedury snižují množství
  - komunikace po síti
  - interproces komunikace
- I špatně napsaná procedura může rychlejší nežli kód na straně klienta.
- Dobře napsaný kód v PL může být řádově rychlejší nežli kód na straně klienta.

```
      CREATE OR REPLACE FUNCTION public.f(a integer)
       RETURNS integer
       LANGUAGE plpgsql
1      AS $function$
2      #option dump
3      DECLARE s int := 0; r record;
4      BEGIN
5        FOR r IN SELECT * FROM foo WHERE foo.a = f.a
6        LOOP
7          s := s + r.a;
8        END LOOP;
9        RETURN s;
10     END;
11     $function$
```

```
Function's data area:
    entry 0: VAR $1               type int4 (typoid 23) atttypmod -1
    entry 1: VAR found              type bool (typoid 16) atttypmod -1
    entry 2: VAR s                type int4 (typoid 23) atttypmod -1
                                DEFAULT 'SELECT 0'
    entry 3: REC r
    entry 4: RECFIELD a                 of REC 3

Function's statements:
  4:BLOCK <<*unnamed*>>
  5:  FORS r 'SELECT * FROM foo WHERE foo.a = f.a'
  7:    ASSIGN var 2 := 'SELECT s + r.a'
      ENDFORS
  9:  RETURN 'SELECT s'
    END -- *unnamed*
```


Vyhodnoť výraz
Existuje prováděcí plán?
ANO
NE
Generuj prováděcí plán
Zjisti, zda se jedná o jednoduchý plán - výraz
Vykonej prováděcí plán
Uložené prováděcí plány se po ukončení session zahodí

- Nevhodný přístup
  - PL/pgSQL není Fortran – vše co nemusíme programovat, ***nesmíme*** programovat.
- Psaní jednořádkových funkcí v PL/pgSQL.
- Optimalizace naslepo nezafunguje – občas je nutné použít *dynamické SQL*.
- Paměťová náročnost – sčítání řetězců, úpravy polí
- Opakování deTOAST procesu.
- Opakované transformace hodnot při neshodě typů.

```
SELECT * FROM tab
      WHERE jmeno = 'Tonda';


DECLARE r RECORD;
BEGIN
      FOR r IN SELECT * FROM tab
                        WHERE jmeno = $1
      LOOP
            ...
      END LOOP;


DECLARE r RECORD;
BEGIN
      FOR r IN
            EXECUTE 'SELECT * FROM tab WHERE jmeno = $1'
                  USING $1
      LOOP
            ...
      END LOOP
```

```
DECLARE a int[] := '{}'
BEGIN
	FOR i IN 1..100
	LOOP
		a[i] := 0;
	END LOOP;
```

```
DECLARE a int[] = array_fill(0, ARRAY[100]);
BEGIN
	...
```

```
DECLARE
        _a int[];
        s bigint = 0;
BEGIN
        _a := (SELECT a FROM tab WHERE ... )
        FOR i IN array_lower(a,1) .. array_upper(a,1)
        LOOP
                s := s + _a[i];
        END LOOP;


-- workaround
DECLARE
        _a int[]; _b int[];
BEGIN
        _a := (...);
        _b := _a;       /* provede deTOAST _a */
```

```
DECLARE
        a float;
        b numeric;
BEGIN
        FOR ...
        LOOP
                b := sin(x);            /* neefektivní */
                b := sin(x)::numeric;    /* efektivní */
```

V případě neshody typu používá PLpgSQL relativně náročnou IO transformaci