A JIT for SQLite

Carl Friedrich Bolz Darya Kurilova (CMU) Laurence Tratt

Software Development Team
2015-05-04

Motivation

- SQLite is an embedded database
- The most used database
- Commonly combined with a (dynamic) language
- Getting the data across the boundary is slow
- Can we improve it?

Meta-Motivation

- Language composition with actual use cases
- Understand a little bit about DBs

Hypotheses

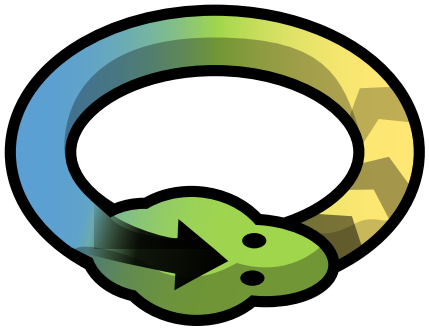
H1 Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.

Hypotheses

H1 Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.

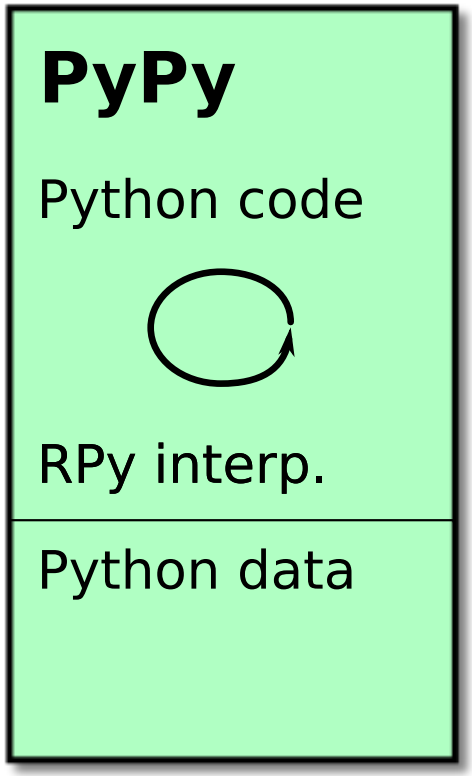
H2 Replacing the query execution engine of a DBMS with a JIT reduces execution time of standalone SQL queries.

- Small embedded SQL database
- dynamically typed
- used a bit everywhere (Mac OS, Android, ...)

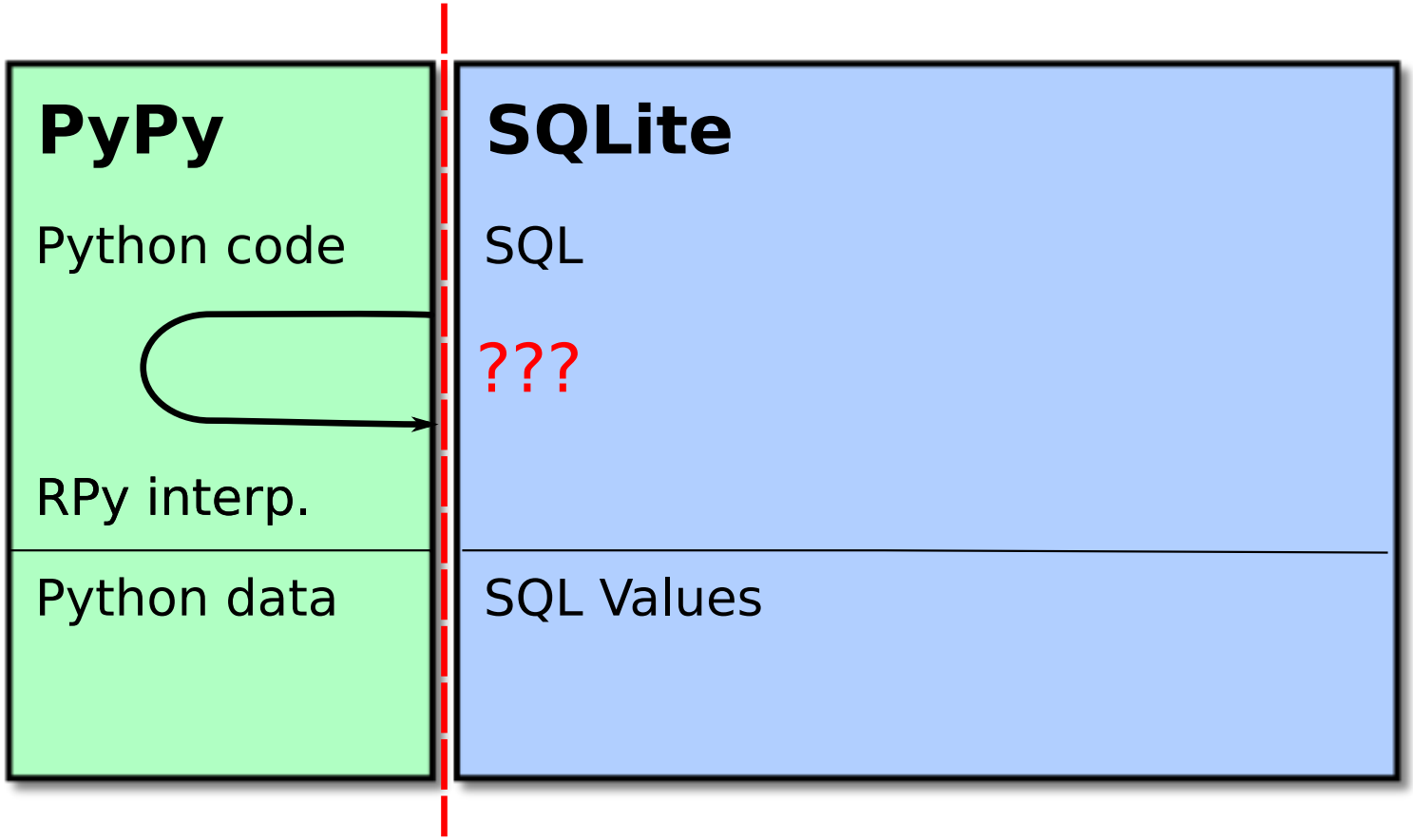


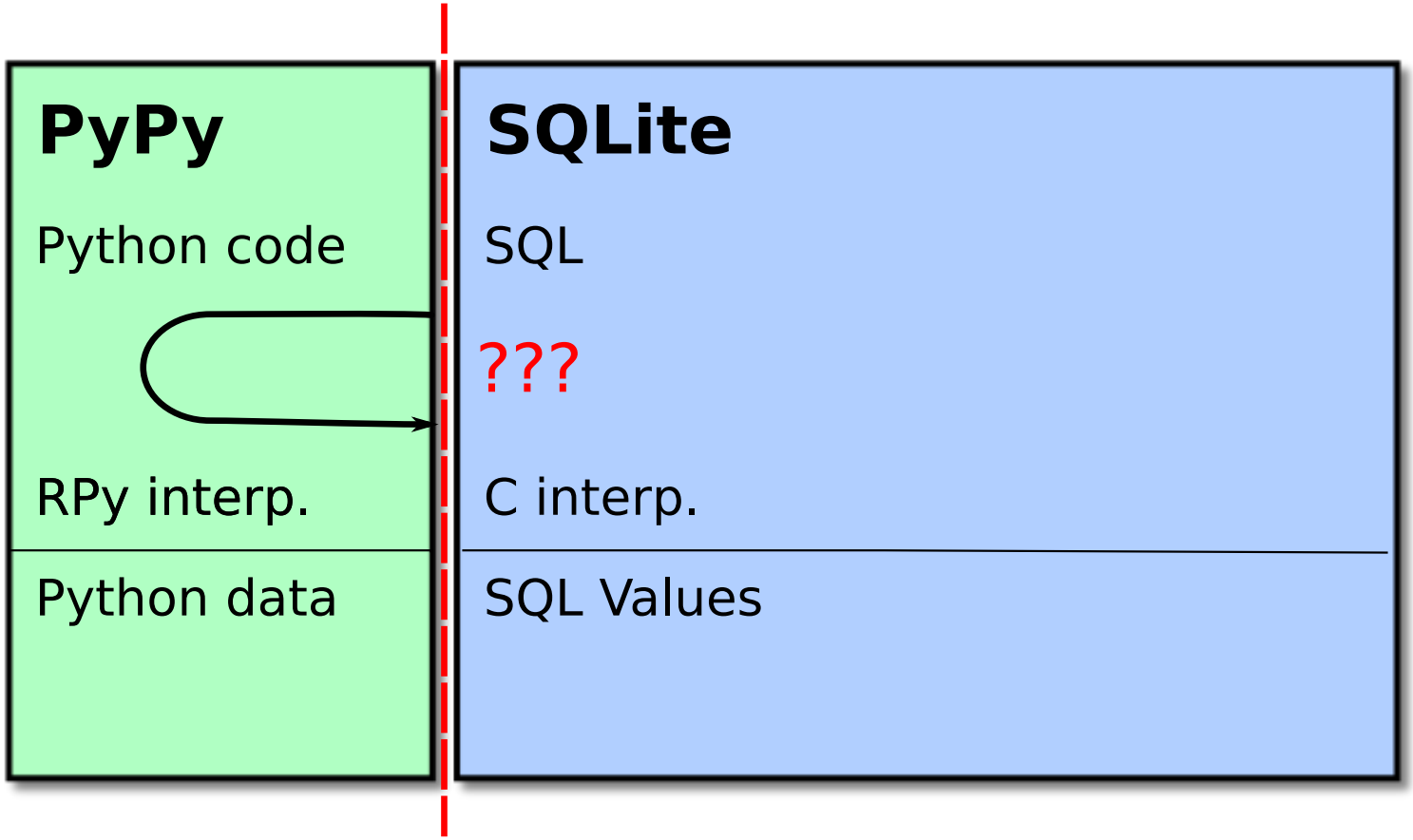
PyPy

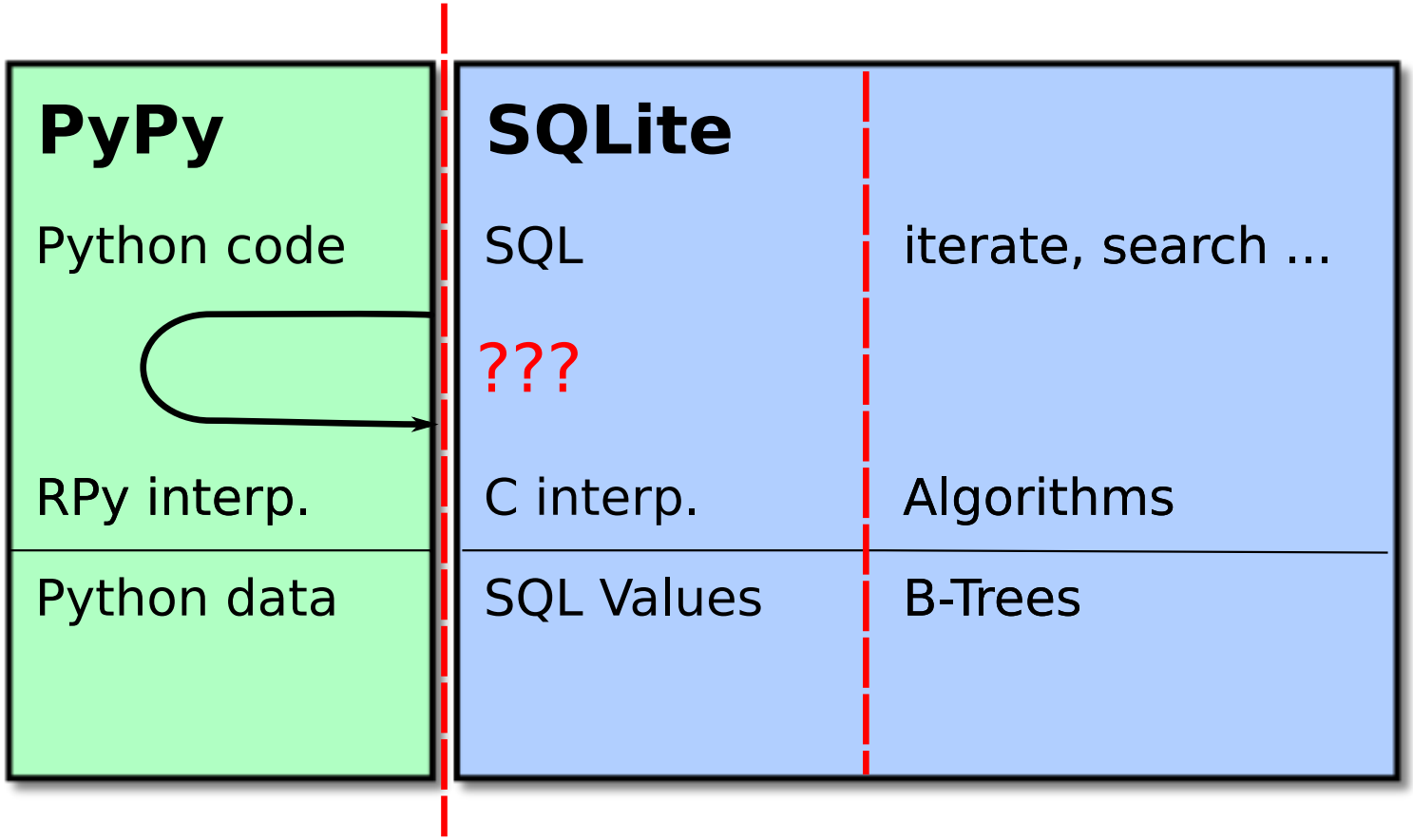
- reimplementation of Python in RPython
- good JIT via the RPython JIT framework

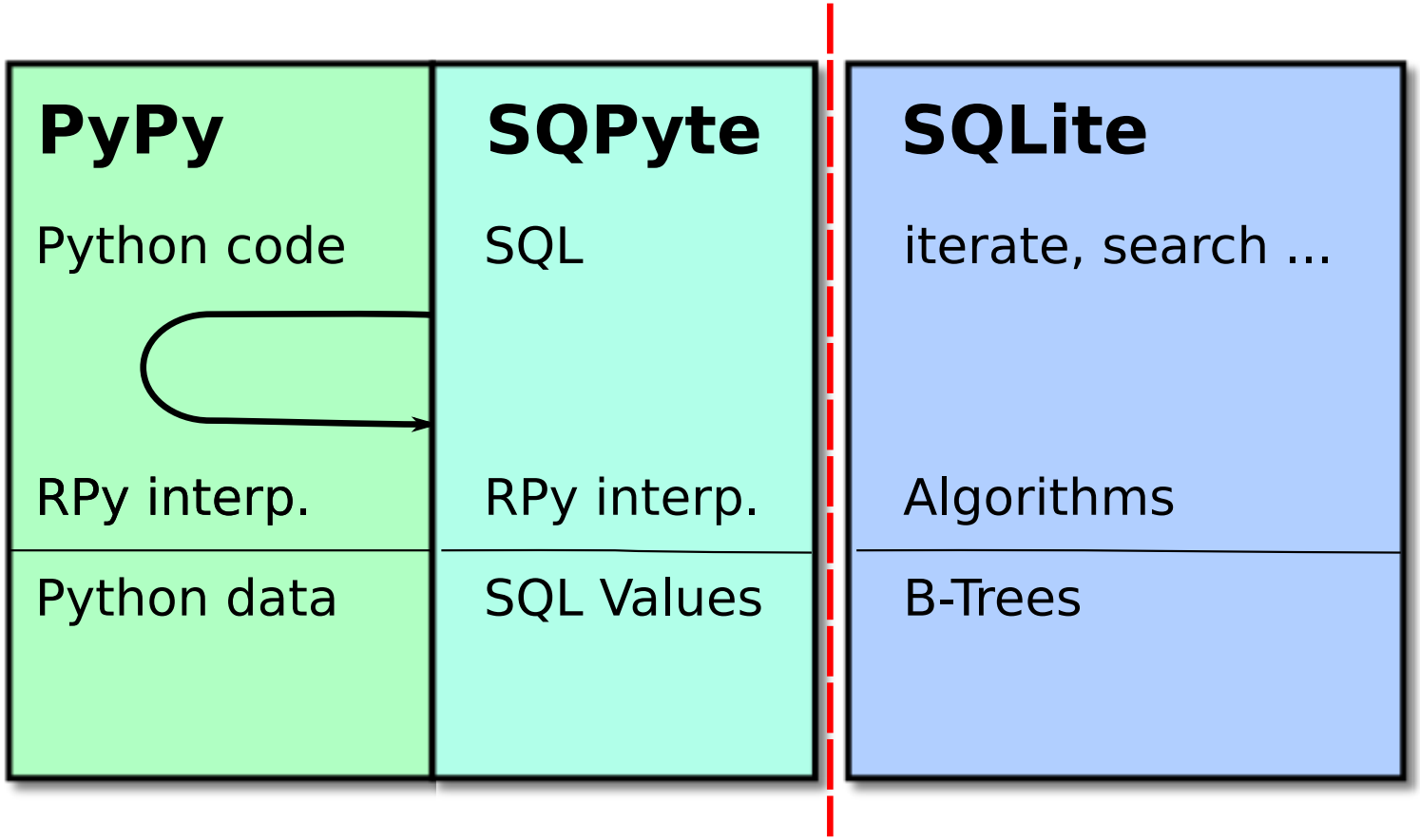

PyPy
Python code
RPy interp.
Python data

```
def select():
    iterator = conn.execute(
        """select quantity, extendedprice, discount
           from lineitem""")
    sum_qty = 0
    sum_base_price = 0
    sum_disc_price = 0
    for quantity, extendedprice, discount in iterator:
        sum_qty += quantity
        sum_base_price += extendedprice
        sum_disc_price += extendedprice * (1 - discount)
    return sum_qty, sum_base_price, sum_disc_price
```


PyPy
Python code
RPy interp.
Python data
SQLite
SQL
???
SQL Values


PyPy
Python code
RPy interp.
Python data
SQLite
SQL
???
C interp.
SQL Values

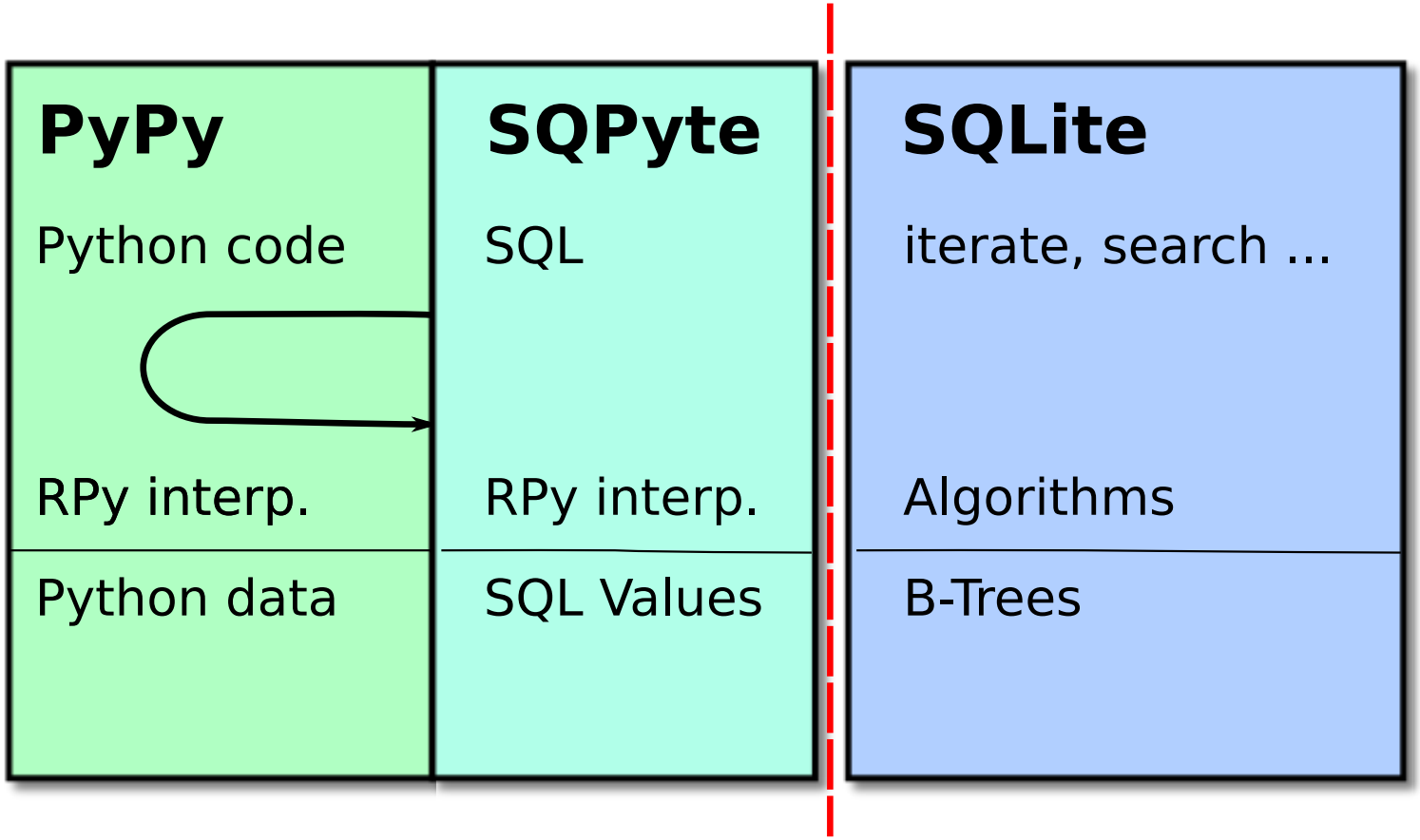

PyPy
Python code
RPy interp.
Python data
SQLite
SQL
???
C interp.
SQL Values
iterate, search ...
Algorithms
B-Trees

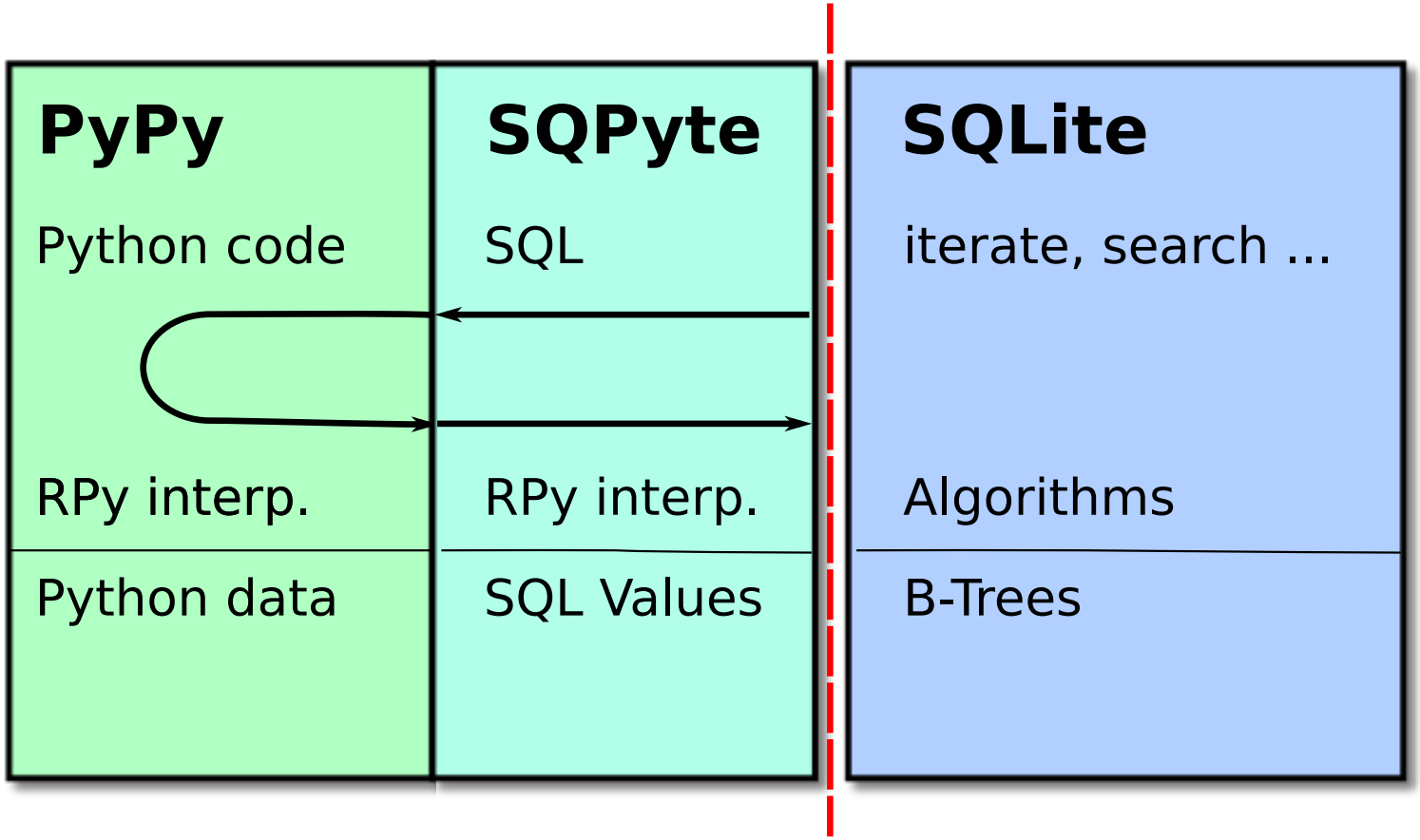

PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees

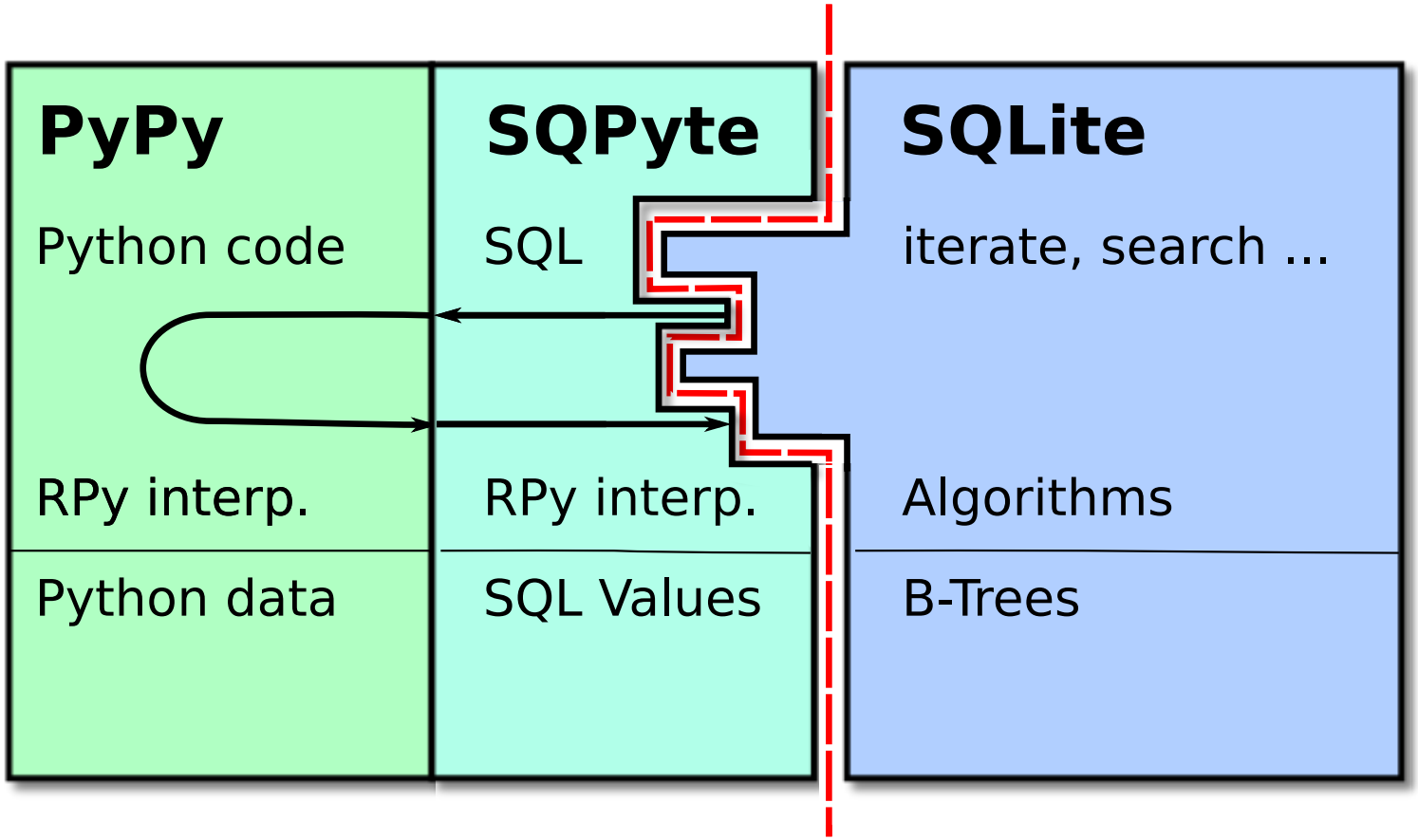
```
case OP_Return: {
  pIn1 = &aMem[pOp->p1];
  assert(pIn1->flags == MEM_Int);
  pc = (int)pIn1->u.i;
  pIn1->flags = MEM_Undefined;
  break;
}
```

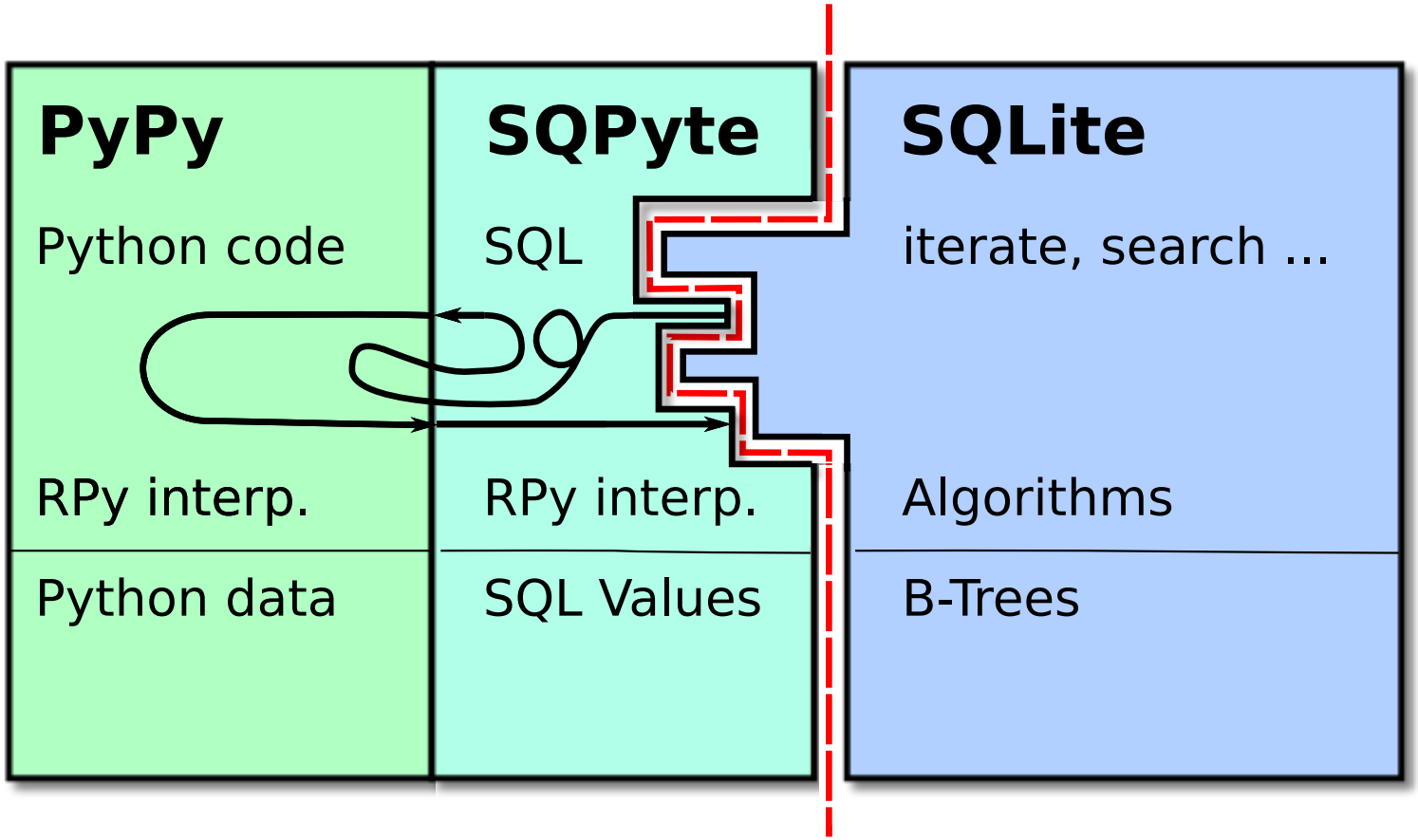
```
case OP_Return: {
  pIn1 = &aMem[pOp->p1];
  assert(pIn1->flags == MEM_Int);
  pc = (int)pIn1->u.i;
  pIn1->flags = MEM_Undefined;
  break;
}
```

```
def python_OP_Return(hlquery, op):
    pIn1 = op.mem_of_p(1)
    assert pIn1.get_flags() == CConfig.MEM_Int
    pc = pIn1.get_u_i()
    pIn1.set_flags(CConfig.MEM_Undefined)
    return pc
```


PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees


PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees

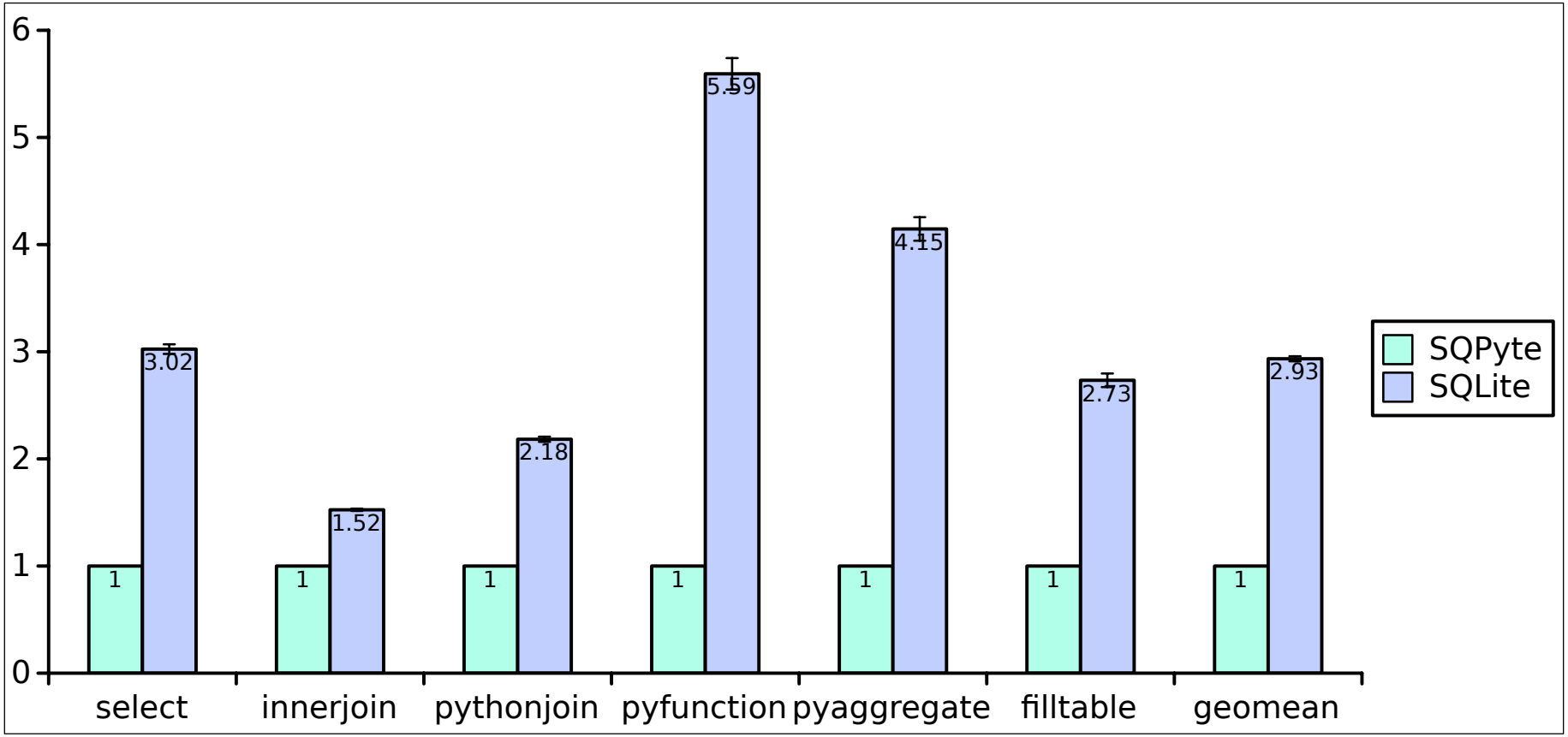

PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees


PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees

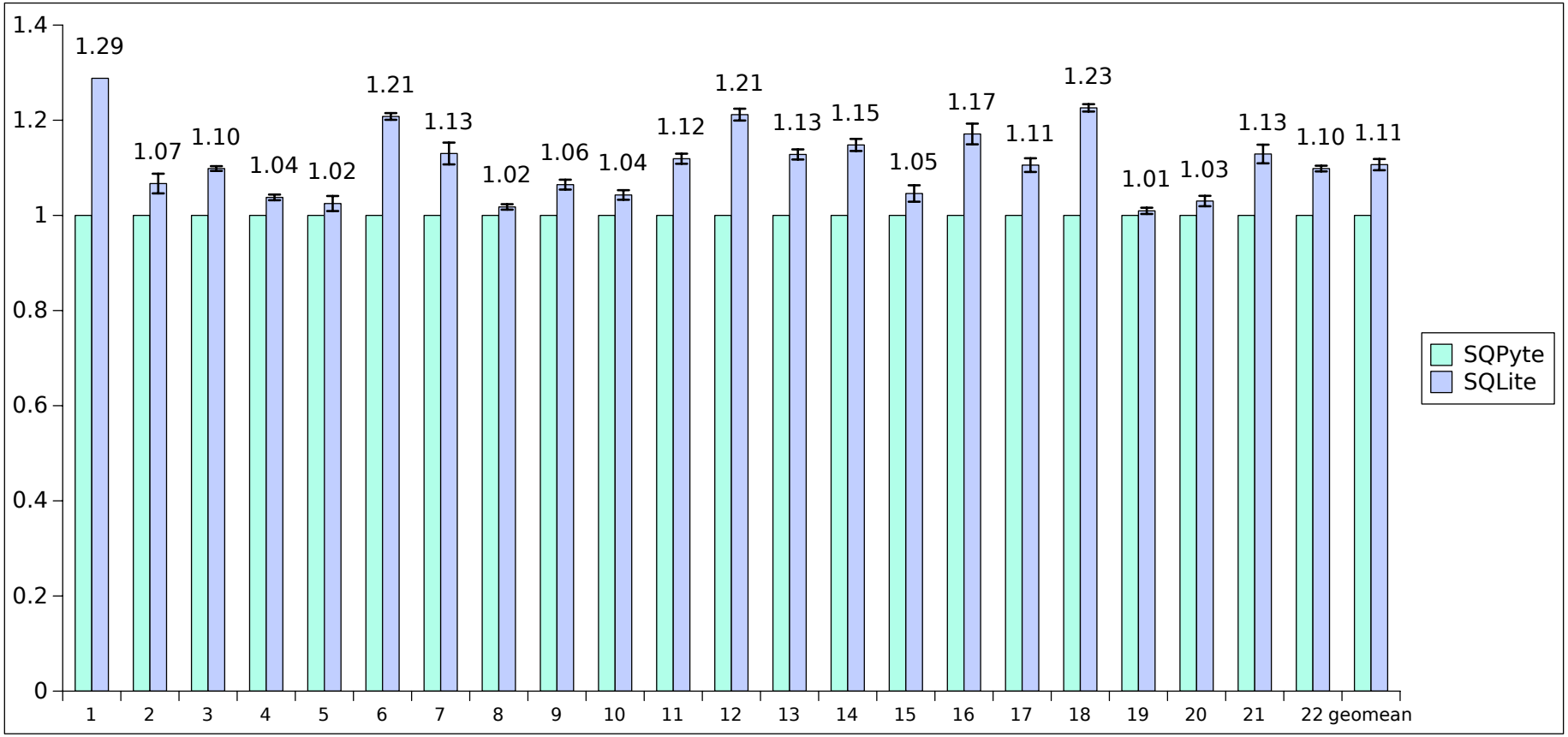
Optimizations

- language crossing, type conversion
- dynamic typing in SQLite
- inlined user-defined functions and aggregates

Microbenchmarks



TPC-H


1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.29
1.07
1.10
1.04
1.02
1.21
1.13
1.02
1.06
1.04
1.12
1.21
1.13
1.15
1.05
1.17
1.11
1.23
1.01
1.03
1.13
1.10
1.11
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 geomean
SQPyte
SQLite

Summary

- H1 confirmed
- H2 maybe
- Python-SQLite can be improved a lot

Summary

- H1 confirmed
- H2 maybe
- Python-SQLite can be improved a lot

Future Work

- Where exactly is the speedup coming from?
- Interaction with an ORM
- Try with "real" DB?