Making an Embedded DBMS JIT-friendly

Carl Friedrich Bolz Darya Kurilova (CMU) Laurence Tratt

Software Development Team
ECOOP
2016-07-20

Motivation

```
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
```

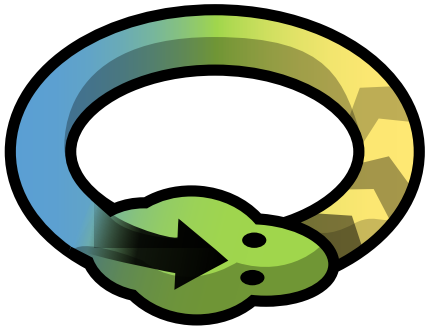
Motivation

- SQLite is an embedded database
- Commonly combined with a (dynamic) language, e.g. Python
- Getting the data across the boundary is slow
- Can we improve it by adding a JIT to SQLite?

Motivation

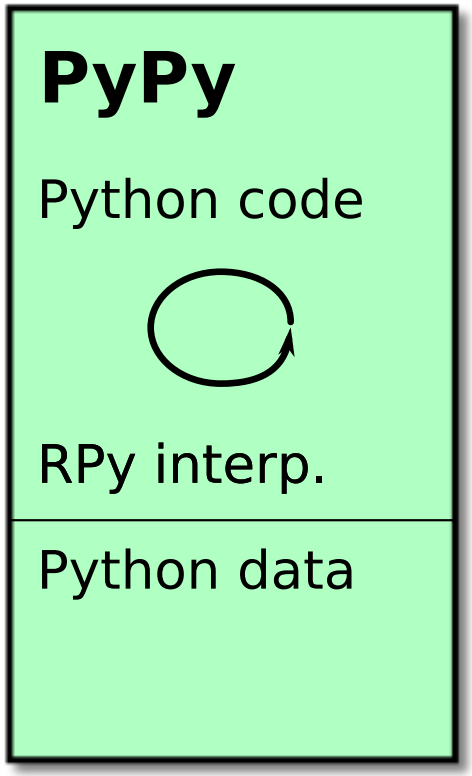
- SQLite is an embedded database
- Commonly combined with a (dynamic) language, e.g. Python
- Getting the data across the boundary is slow
- Can we improve it by adding a JIT to SQLite?
- We call this combined Python/SQlite JIT “SQPyte”

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



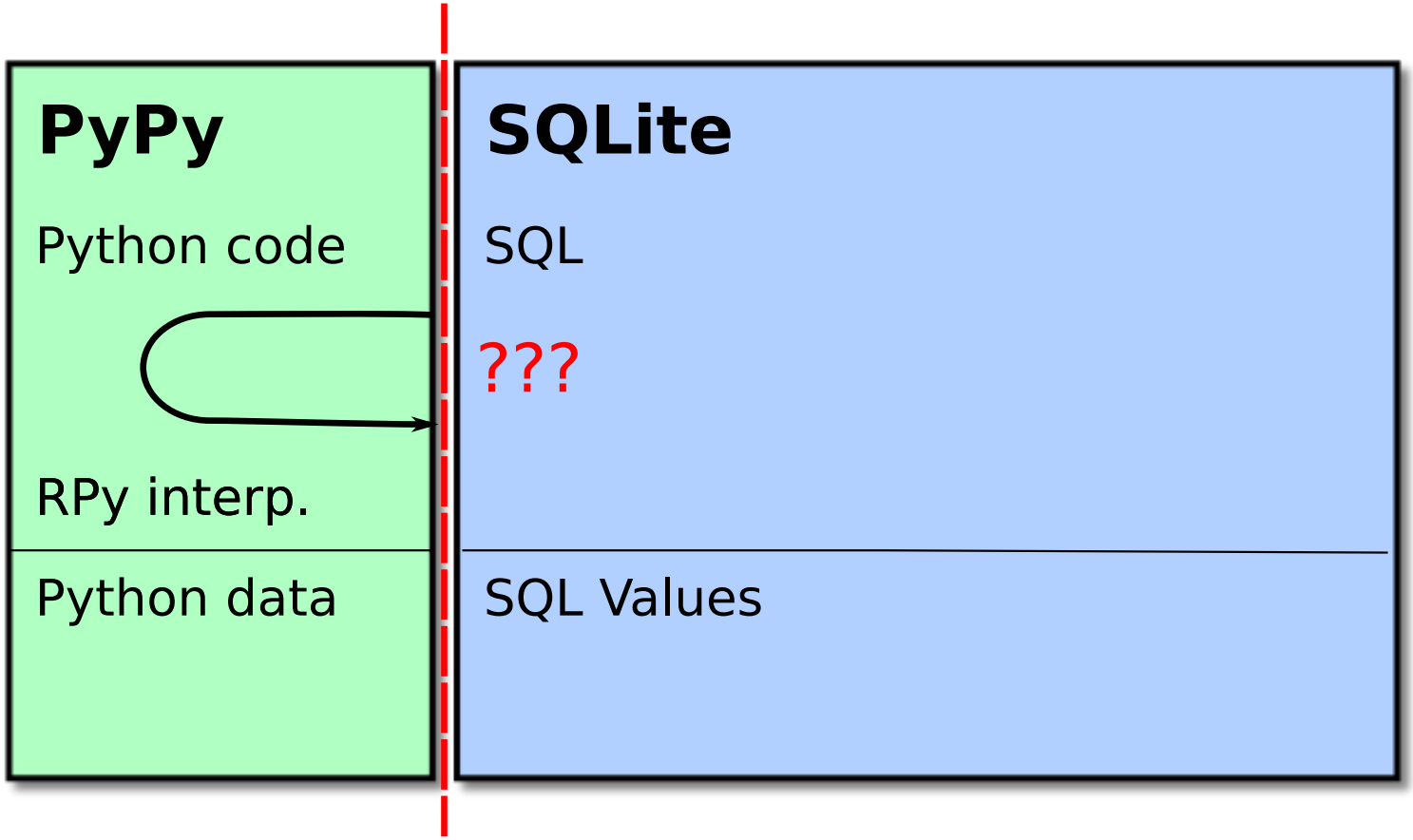
PyPy

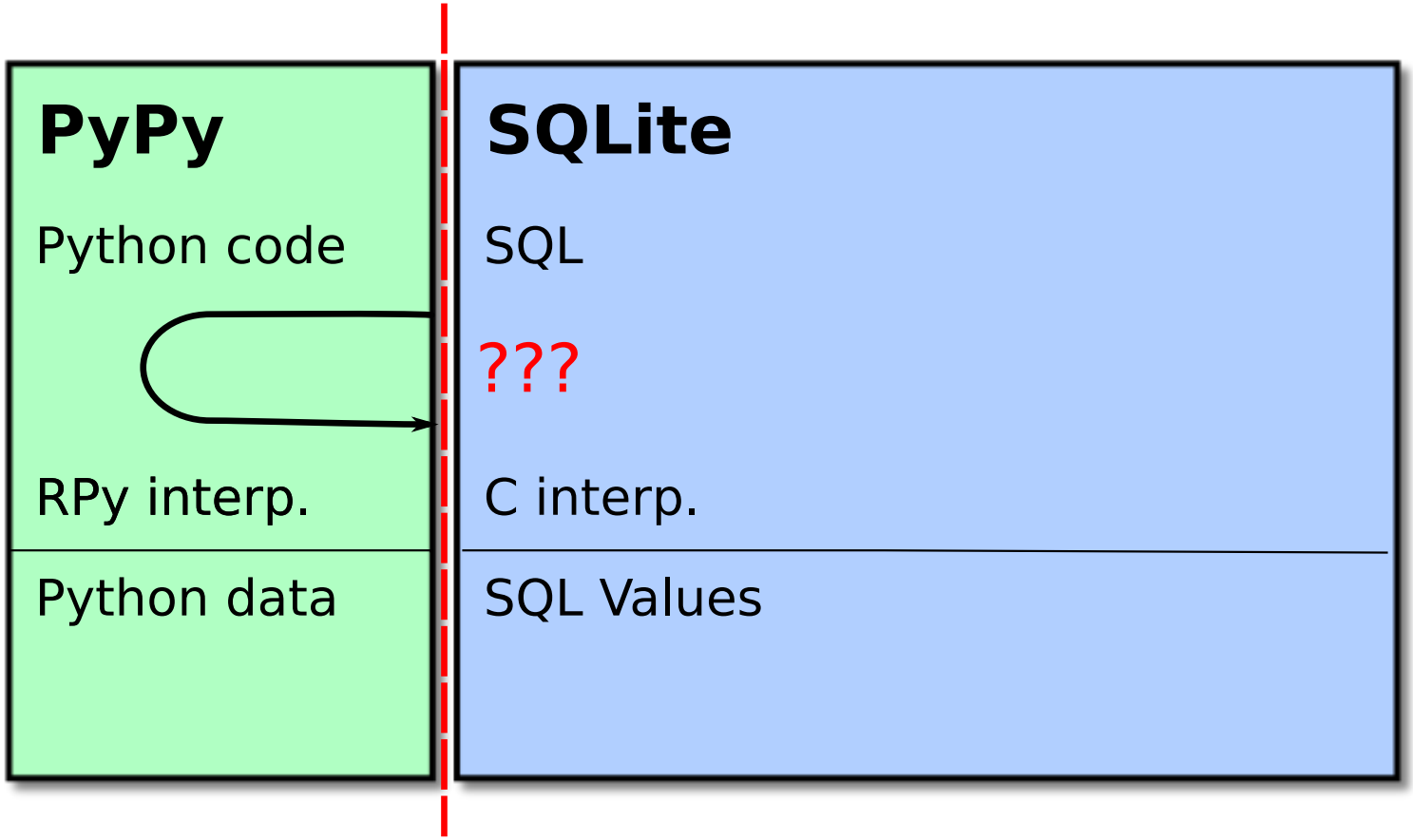
- reimplementation of Python in RPython
- good JIT via the RPython JIT framework
- which adds a tracing JIT to PyPy semi-automatically

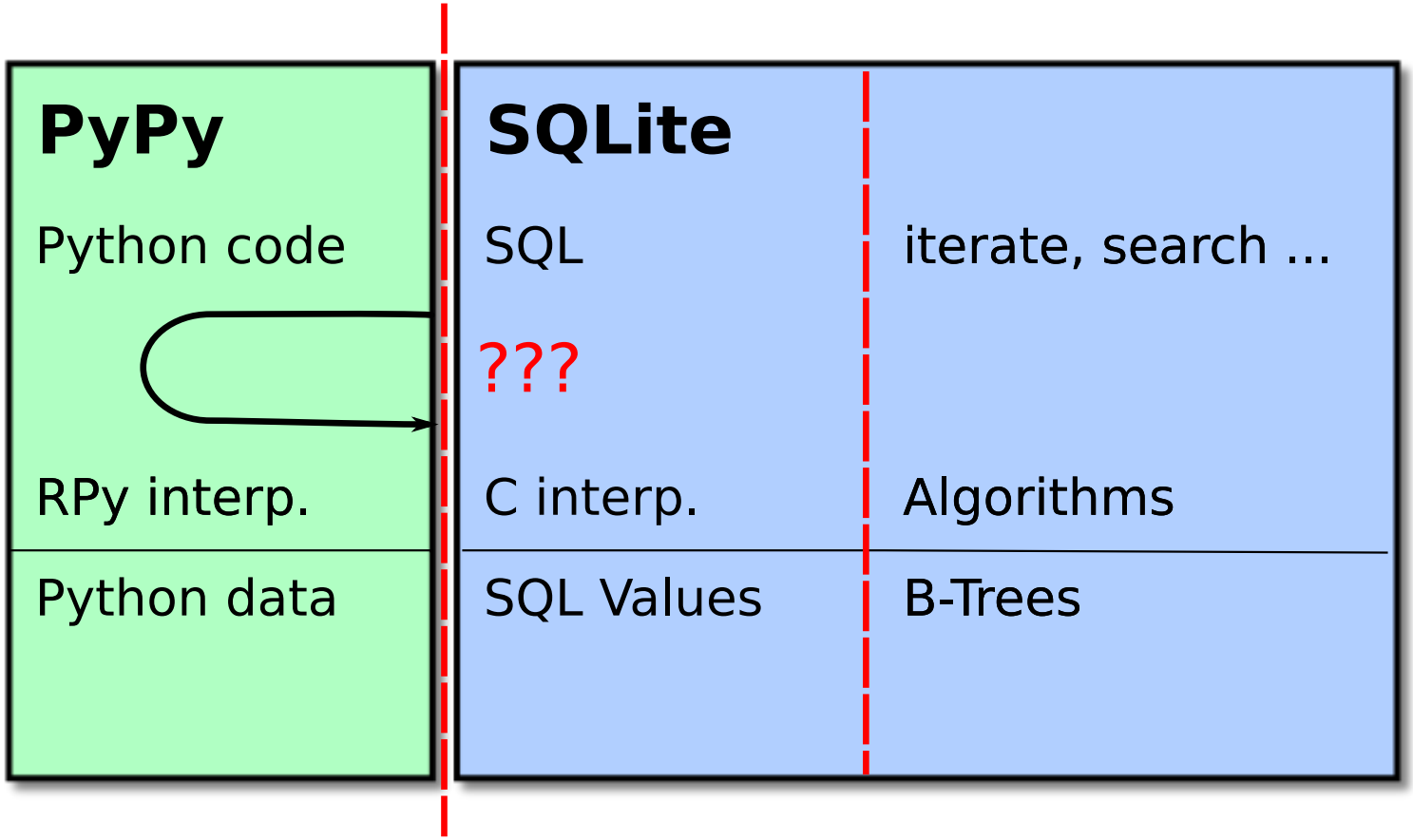

PyPy
Python code
RPy interp.
Python data

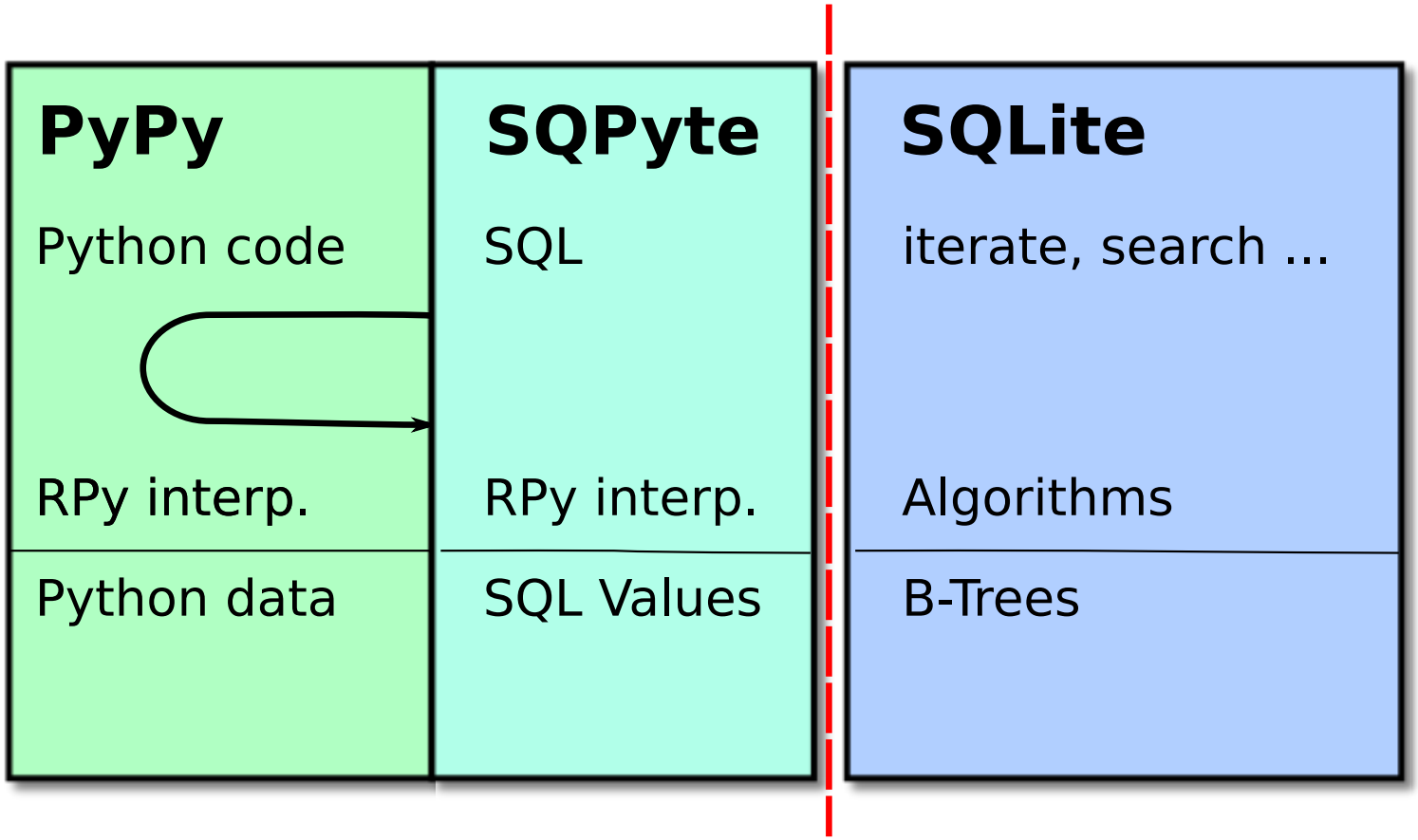
Motivation

```
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

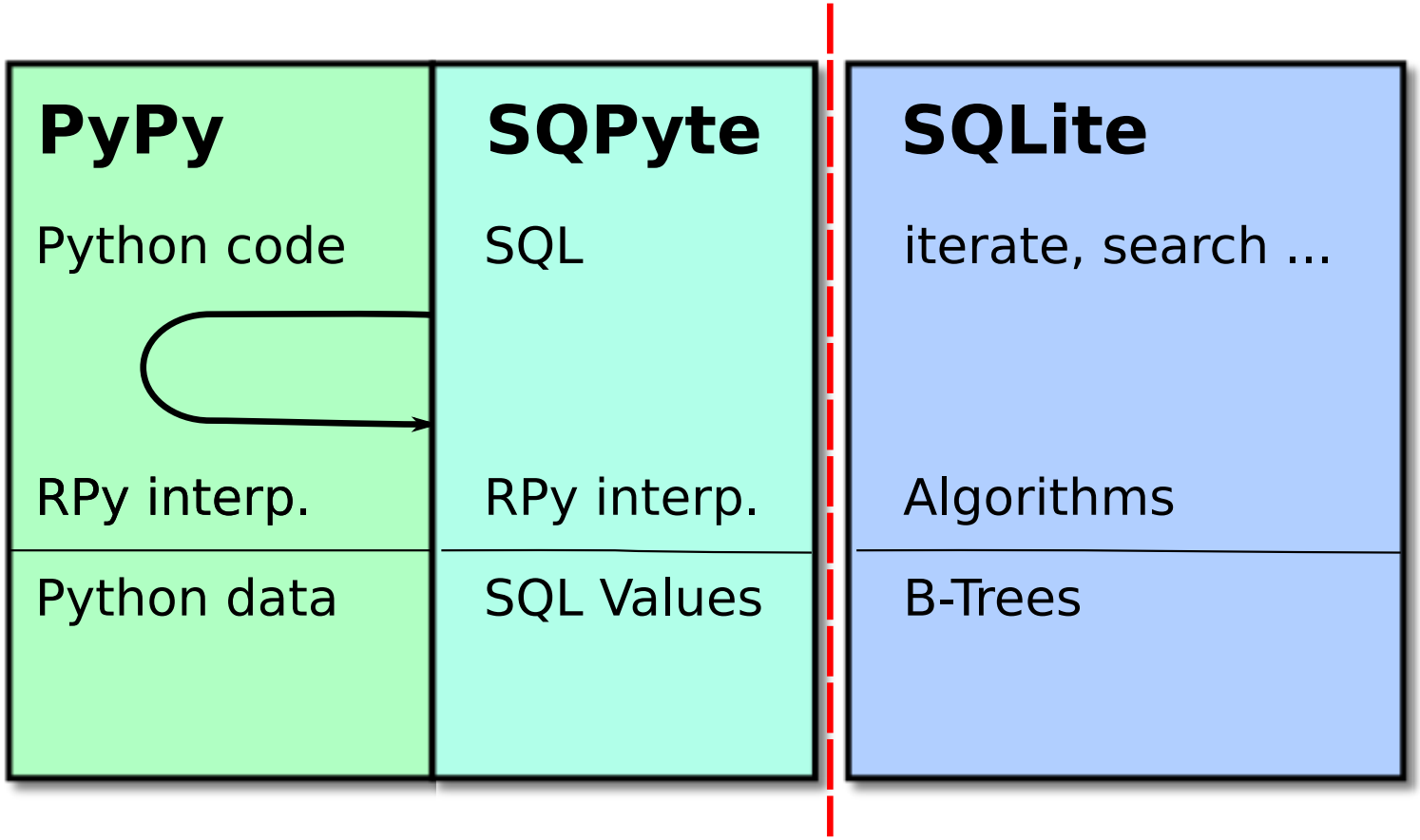

PyPy
Python code
RPy interp.
Python data
SQLite
SQL
???
C interp.
SQL Values
iterate, search ...
Algorithms
B-Trees

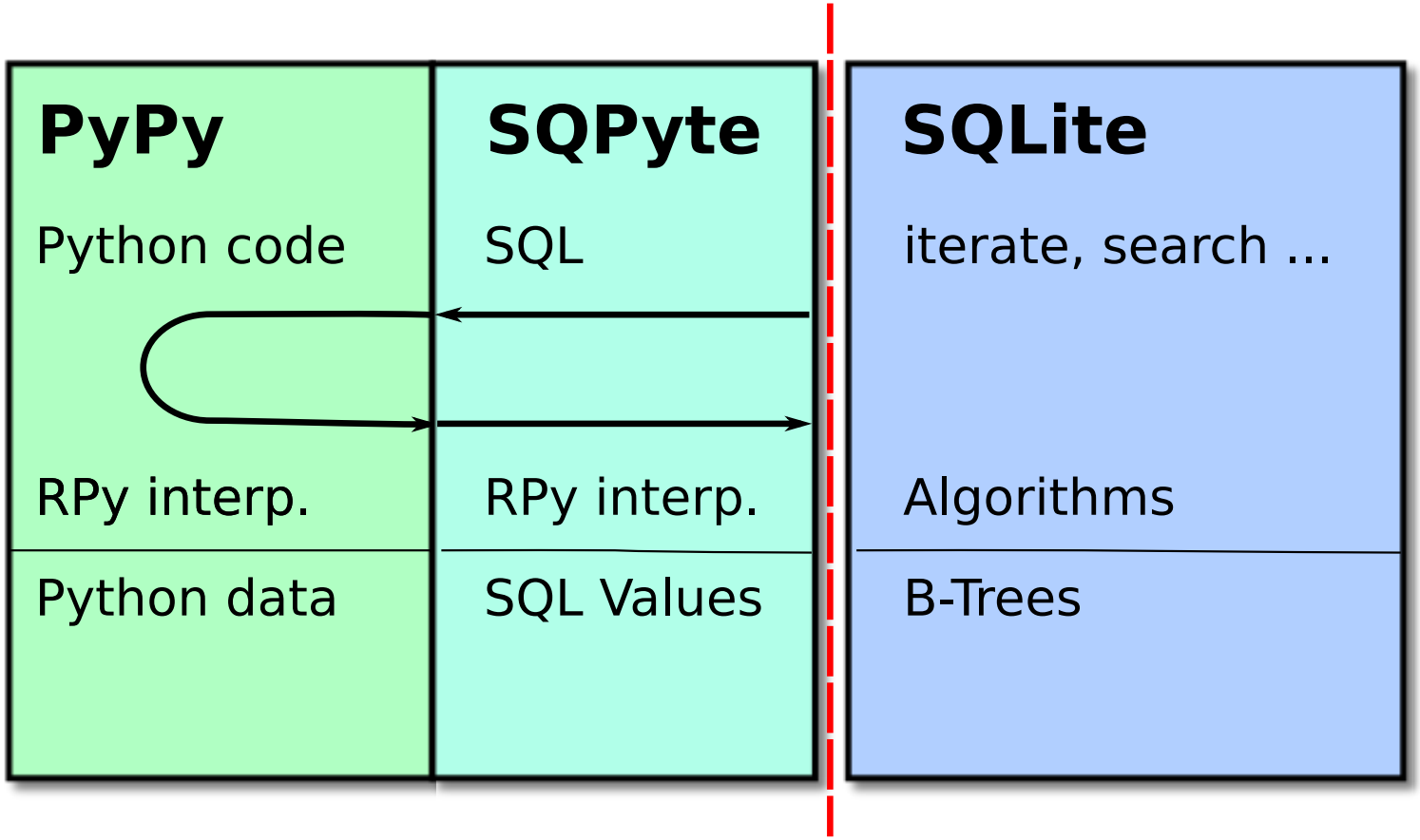

PyPy
Python code
RPy interp.
Python data
SQPyte
SQL
RPy interp.
SQL Values
SQLite
iterate, search ...
Algorithms
B-Trees

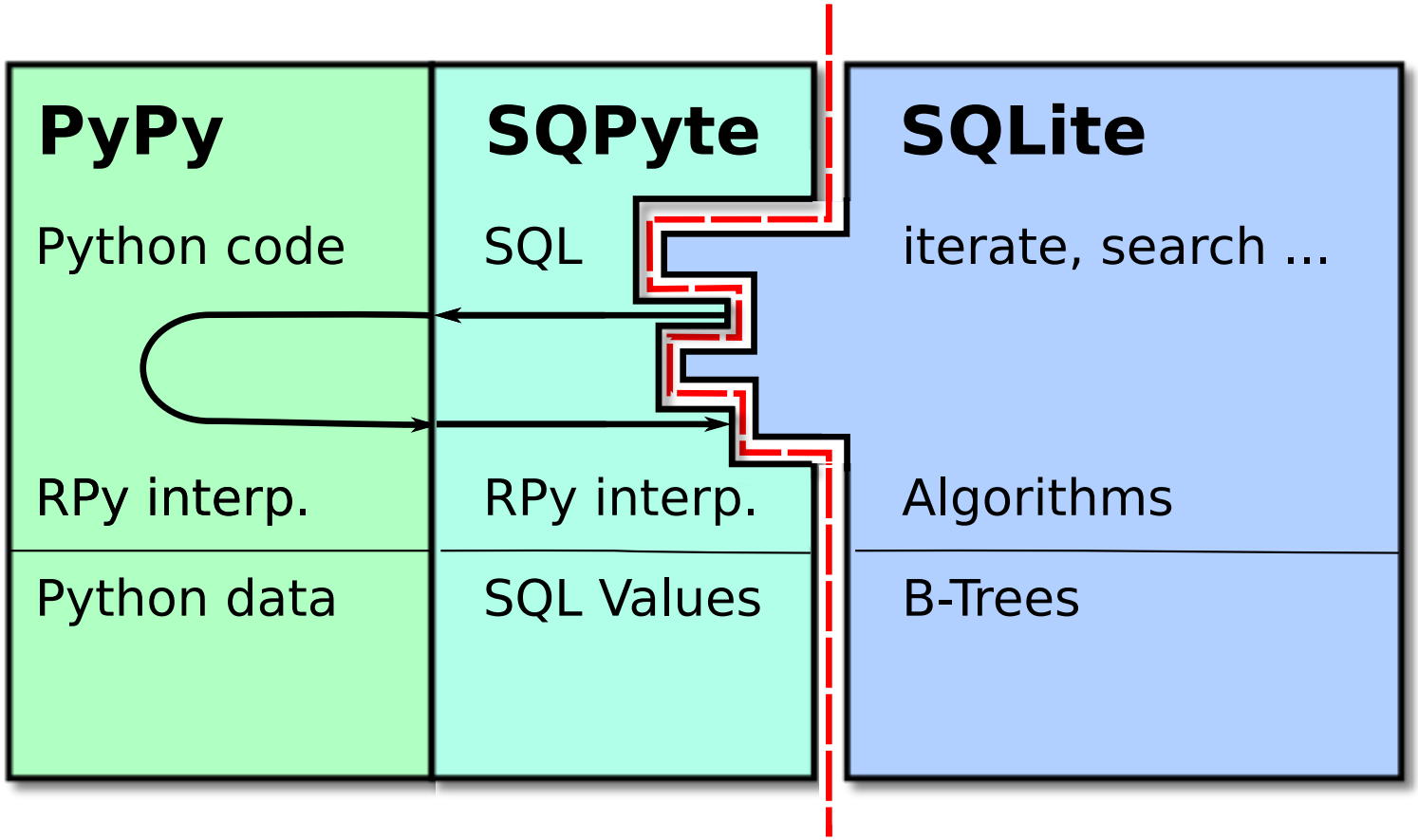
```
case OP_Return: {
  pIn1 = &aMem[pOp->p1];
  assert(pIn1->flags == MEM_Int);
  pc = (int)pIn1->u.i;
  pIn1->flags = MEM_Undefined;
  break;
}
```

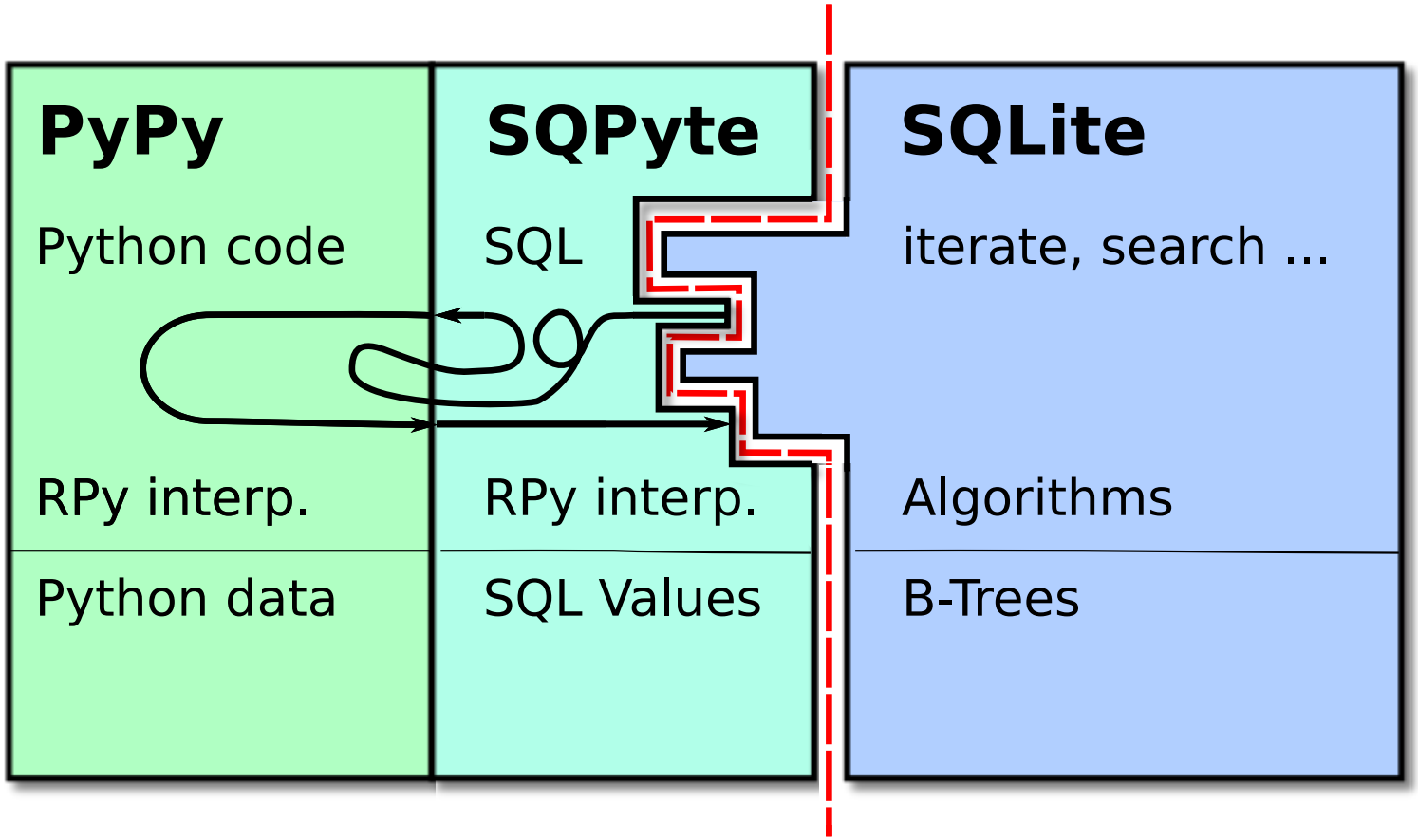
```
case OP_Return: {
  pIn1 = &aMem[pOp->p1];
  assert(pIn1->flags == MEM_Int);
  pc = (int)pIn1->u.i;
  pIn1->flags = MEM_Undefined;
  break;
}
```

```
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

- inline across database/language boundary
- type conversion optimization
- dynamic typing in SQLite

Evaluation: Hypotheses

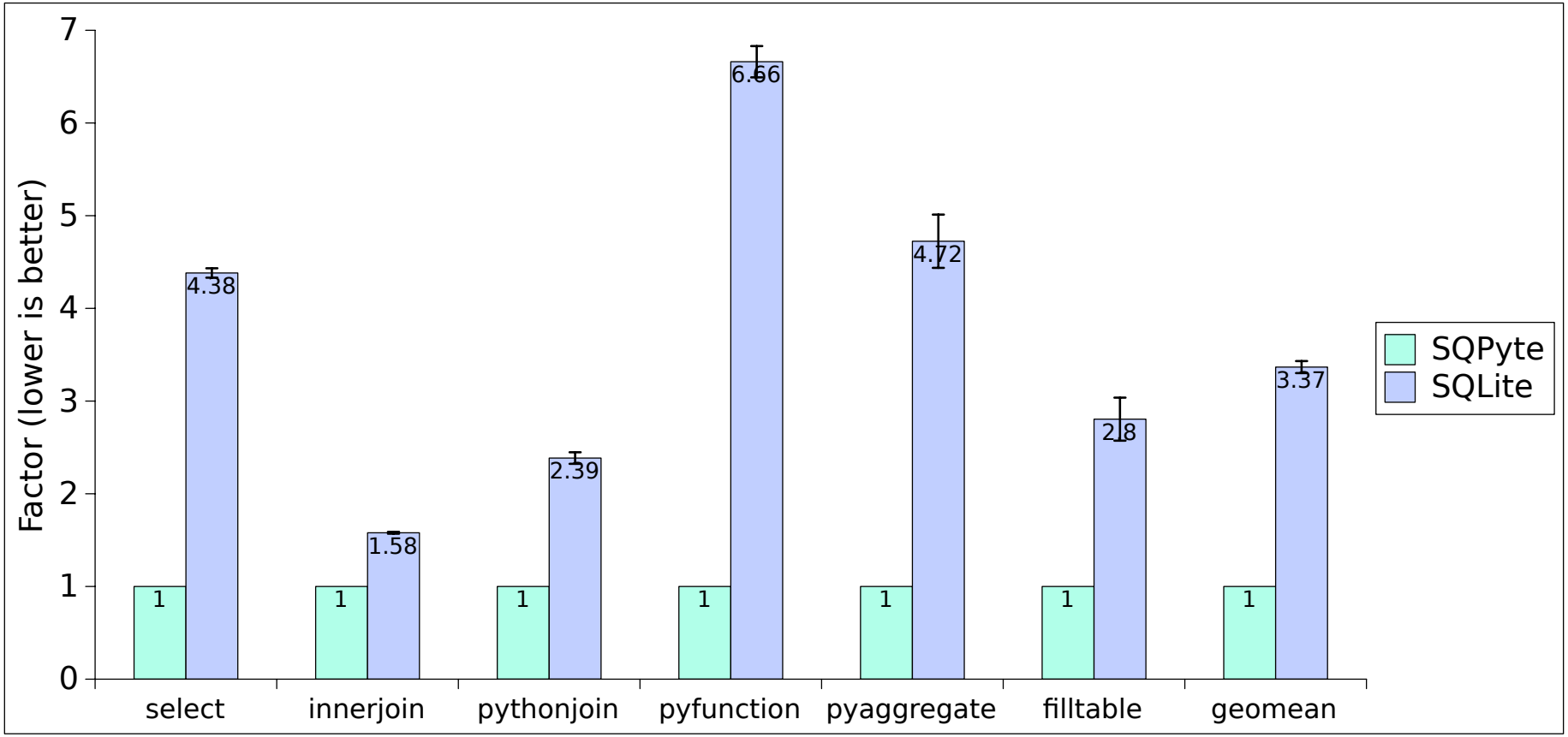
H1 Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.

Evaluation: Hypotheses

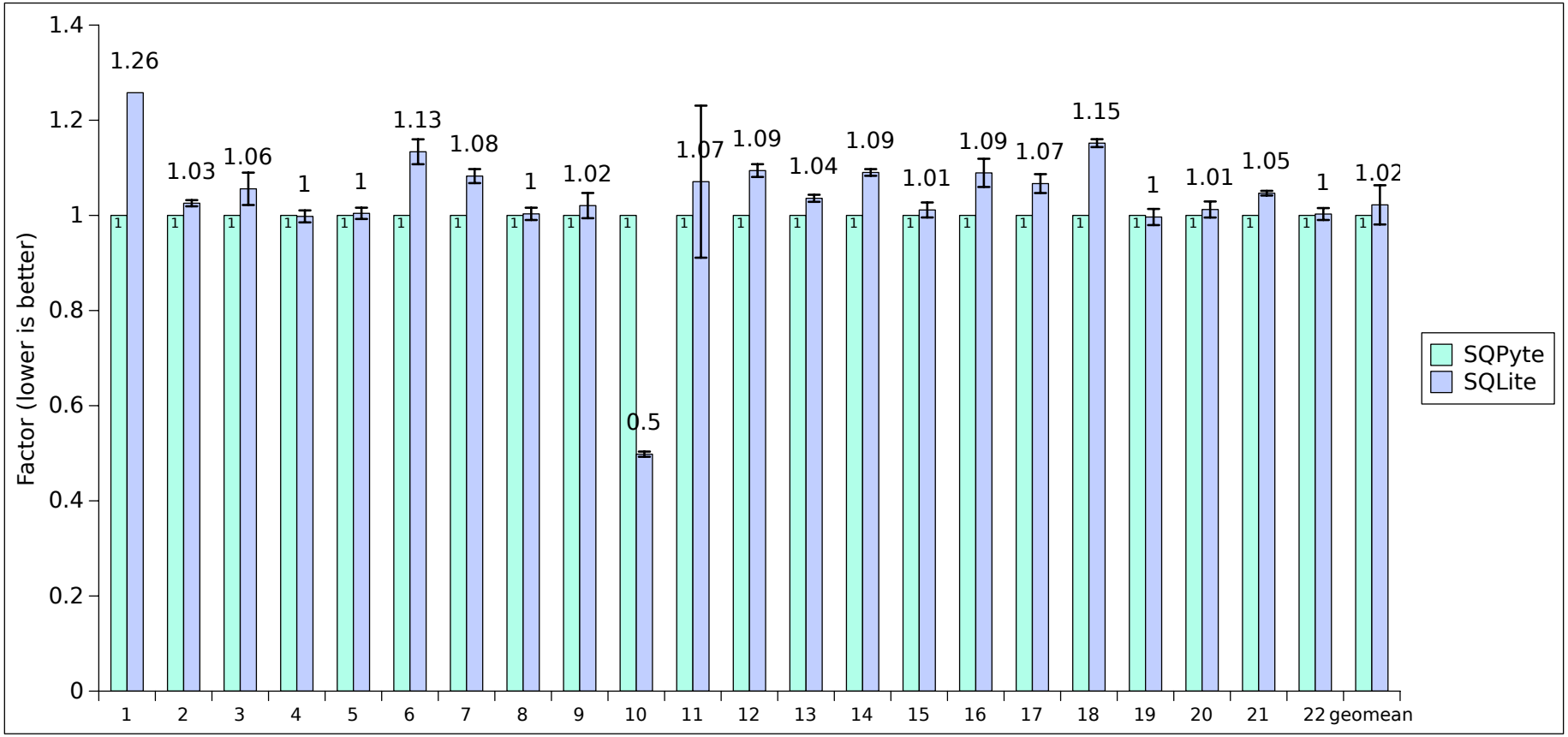
H1 Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.

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

Microbenchmarks



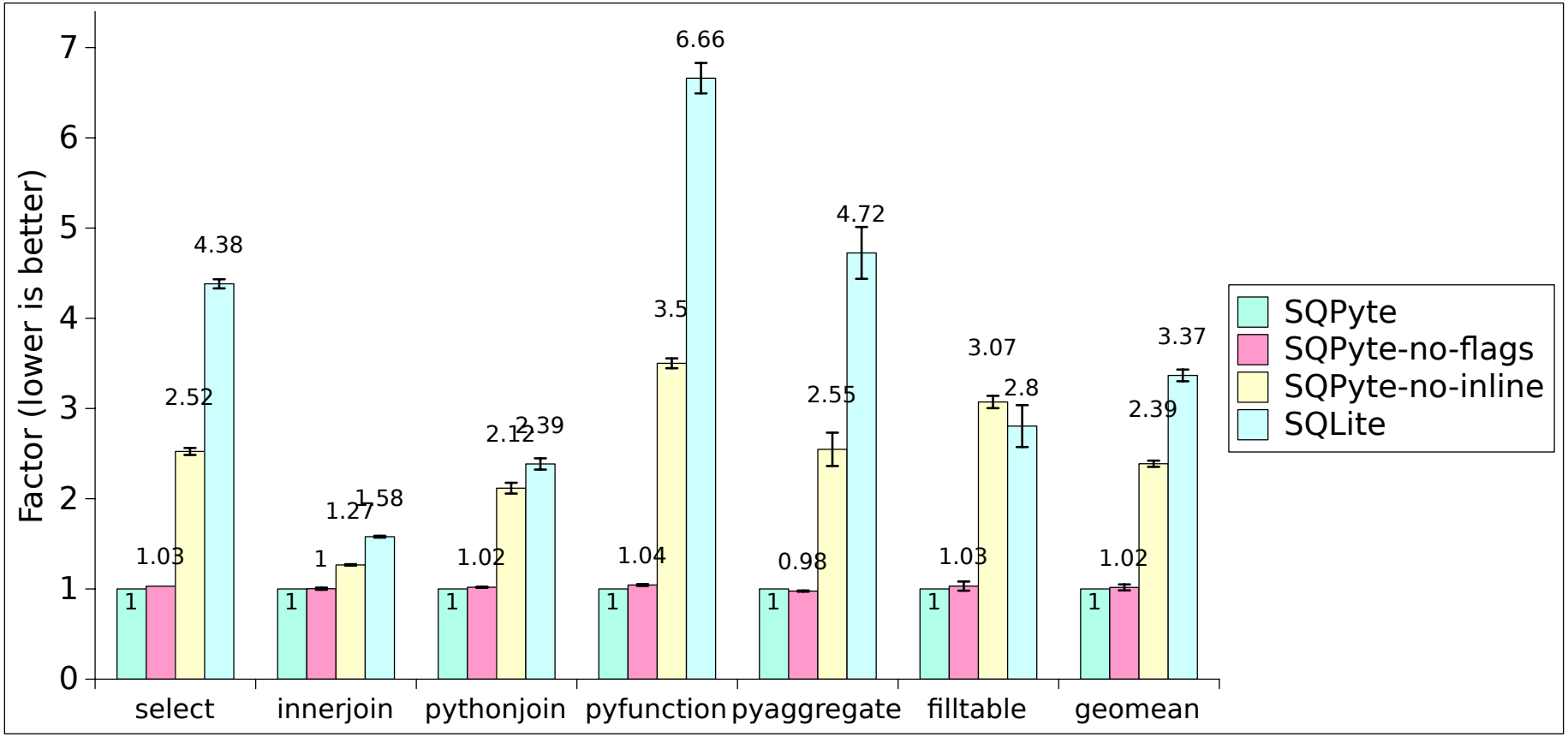
TPC-H


Factor (lower is better)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.26
1.03
1.06
1
1
1.13
1.08
1
1.02
0.5
1.07
1.09
1.04
1.09
1.01
1.09
1.07
1.15
1
1.01
1.05
1
1.02
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

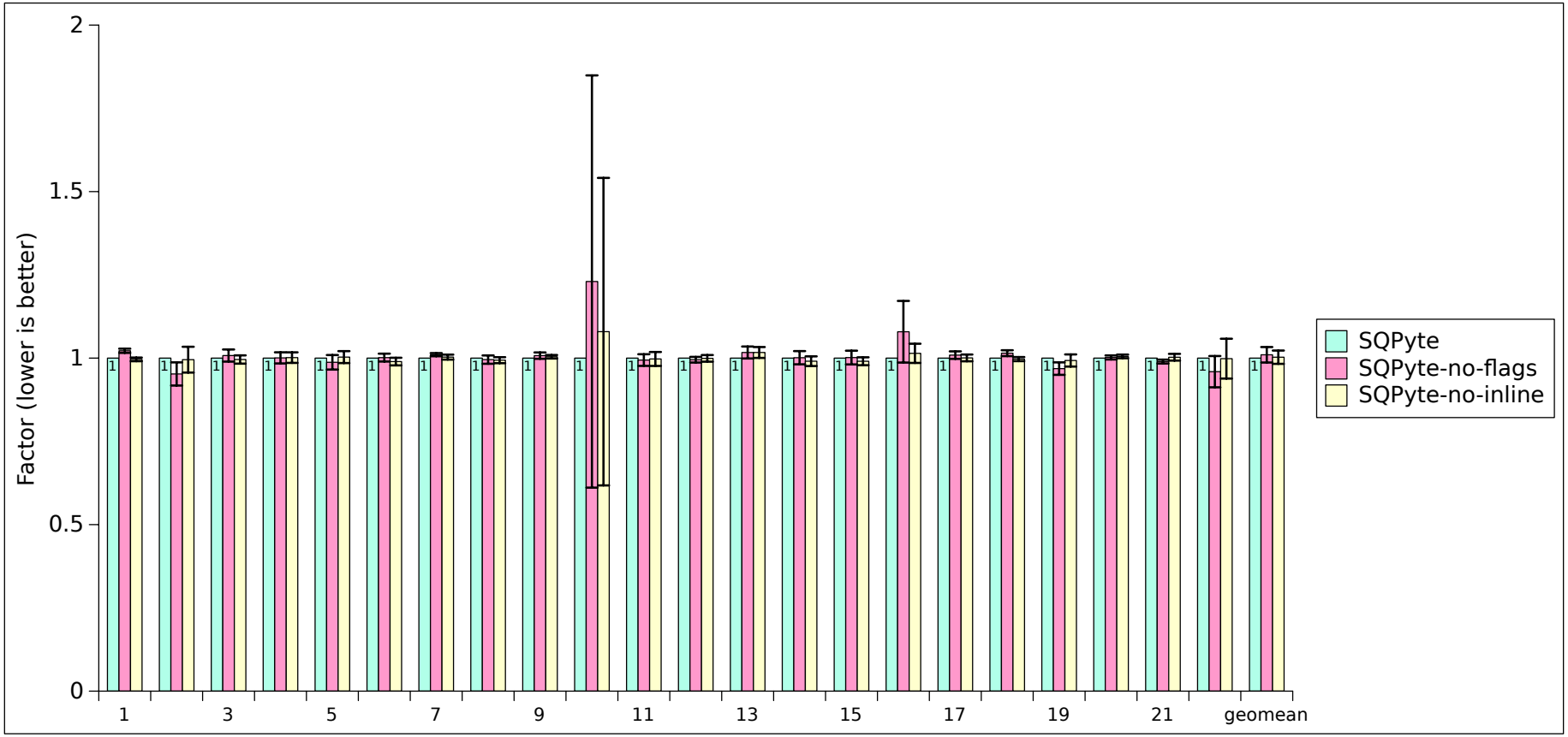
Where do the speedups come from?

- Optimizing the type checks of the dynamically typed SQLite DB?
- Inlining across the languages?

Microbenchmarks Analysis



TPC-H Analysis



Summary

- Optimizing across the language/database boundary with a JIT can give good performance improvements
- Could reuse significant parts of the SQLite codebase

Summary

- Optimizing across the language/database boundary with a JIT can give good performance improvements
- Could reuse significant parts of the SQLite codebase

Future Work

- Are there less intrusive ways to get some of the performance improvements?
- How much further can the approach be pushed?
- Interaction with an ORM
- Try with “real” DB?