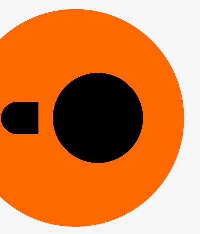
DuckDB Testing Present & Future

Mark Raasveldt

DuckDB Labs

Who Am I?



Mark Raasveldt

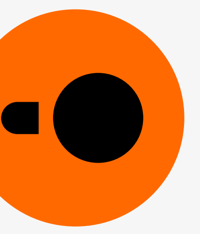
CTO of DuckDB Labs

Postdoc at CWI

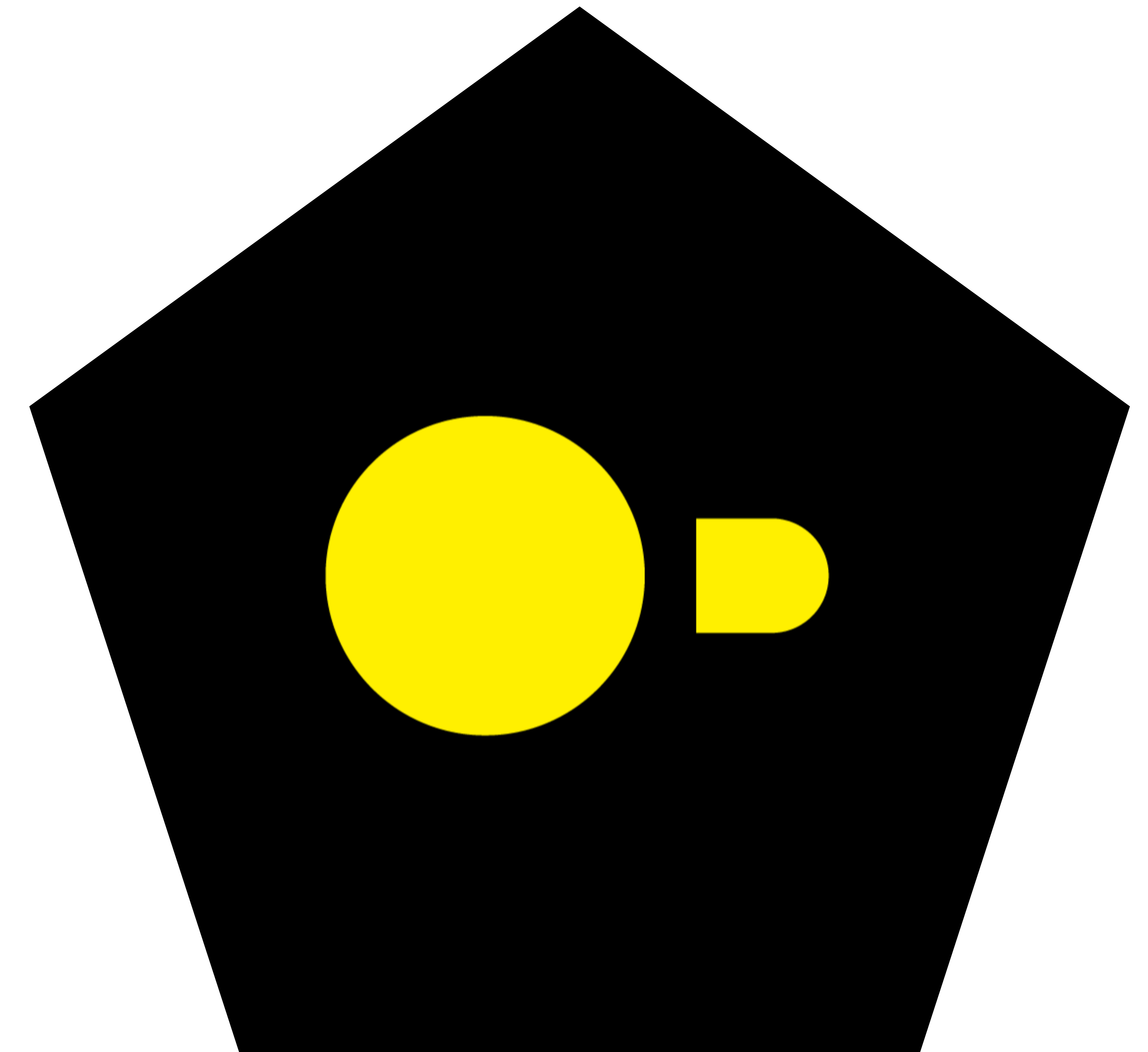
  Database Architectures Group

PhD at CWI

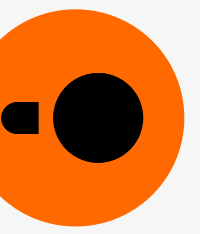
What is DuckDB?



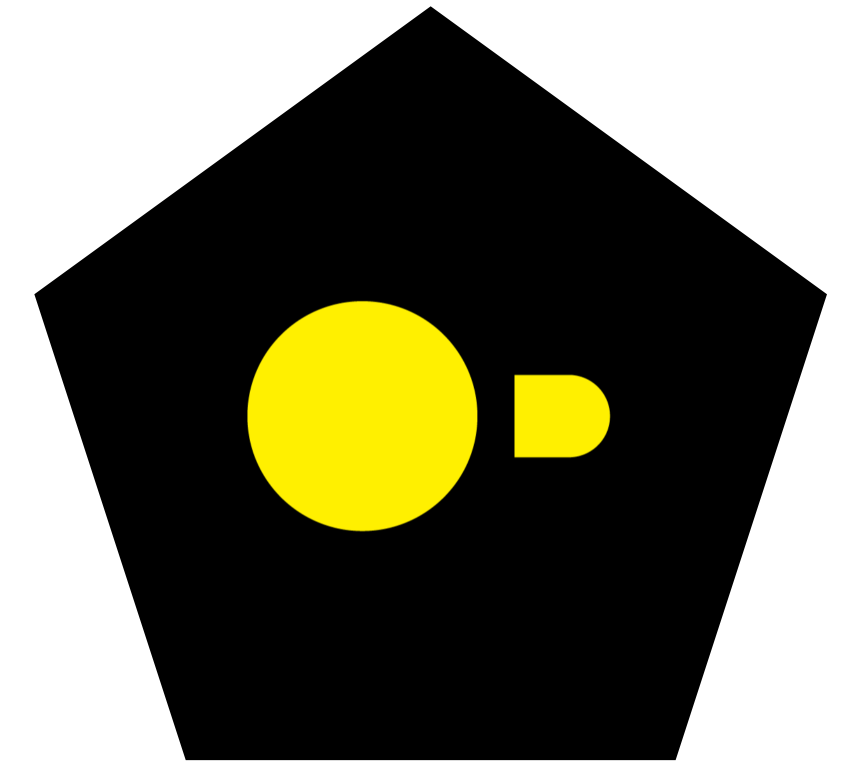
- DuckDB
- In-Process OLAP DBMS
  - “The SQLite for Analytics”
- Free and Open Source (MIT)
- duckdb.org



What is DuckDB?

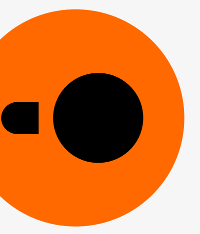


- SQLite inspired us in many ways:
  - Robustness
  - Easy installation
  - Ease of use



- We strive for DuckDB to achieve the same properties

What is DuckDB?



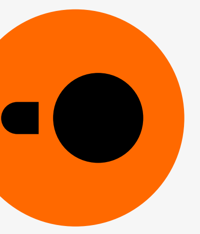
- One big difference

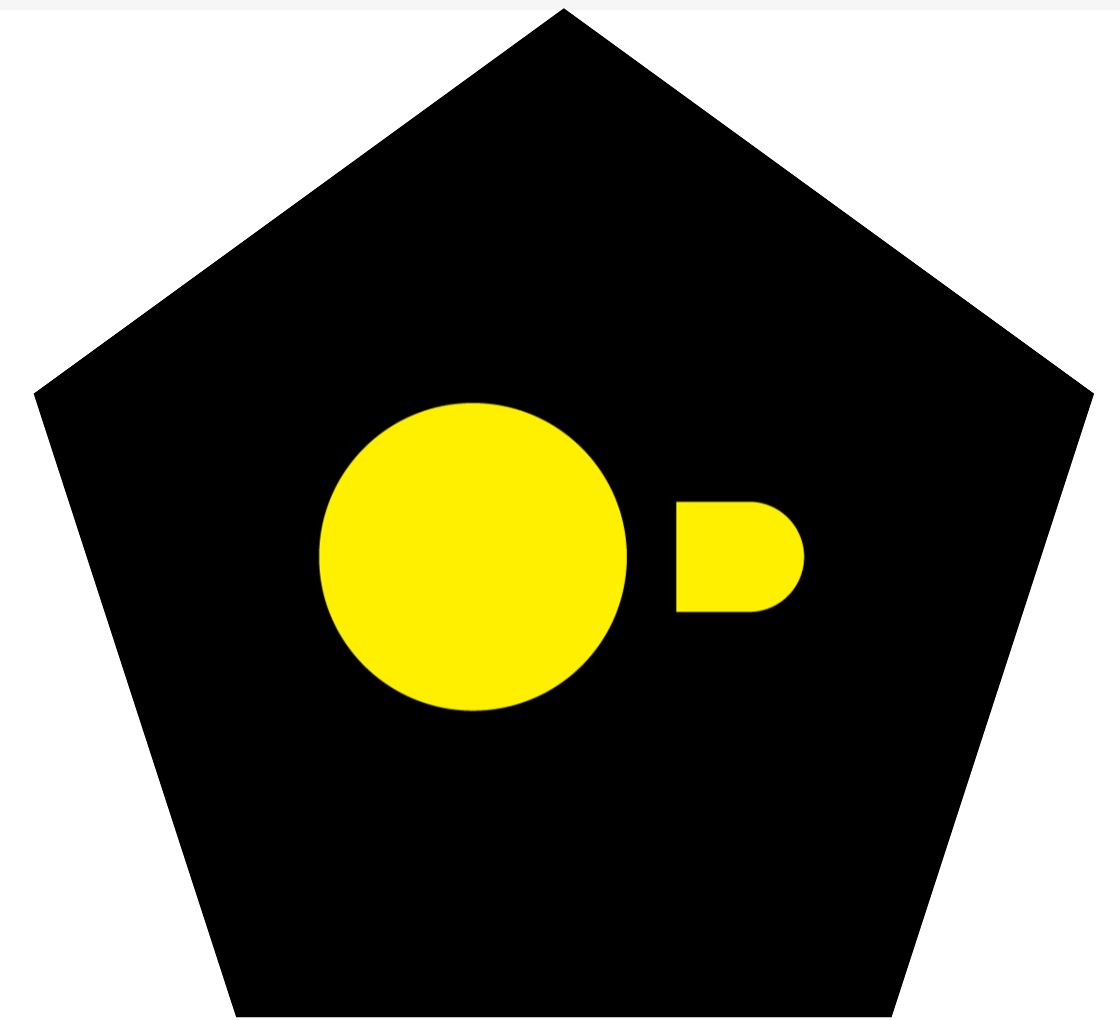
OLTP → many simple queries

OLAP → few complex queries

- OLAP queries require extensive functionality
- DuckDB aims to be “batteries included” to accommodate this

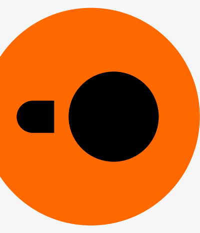
What is DuckDB?



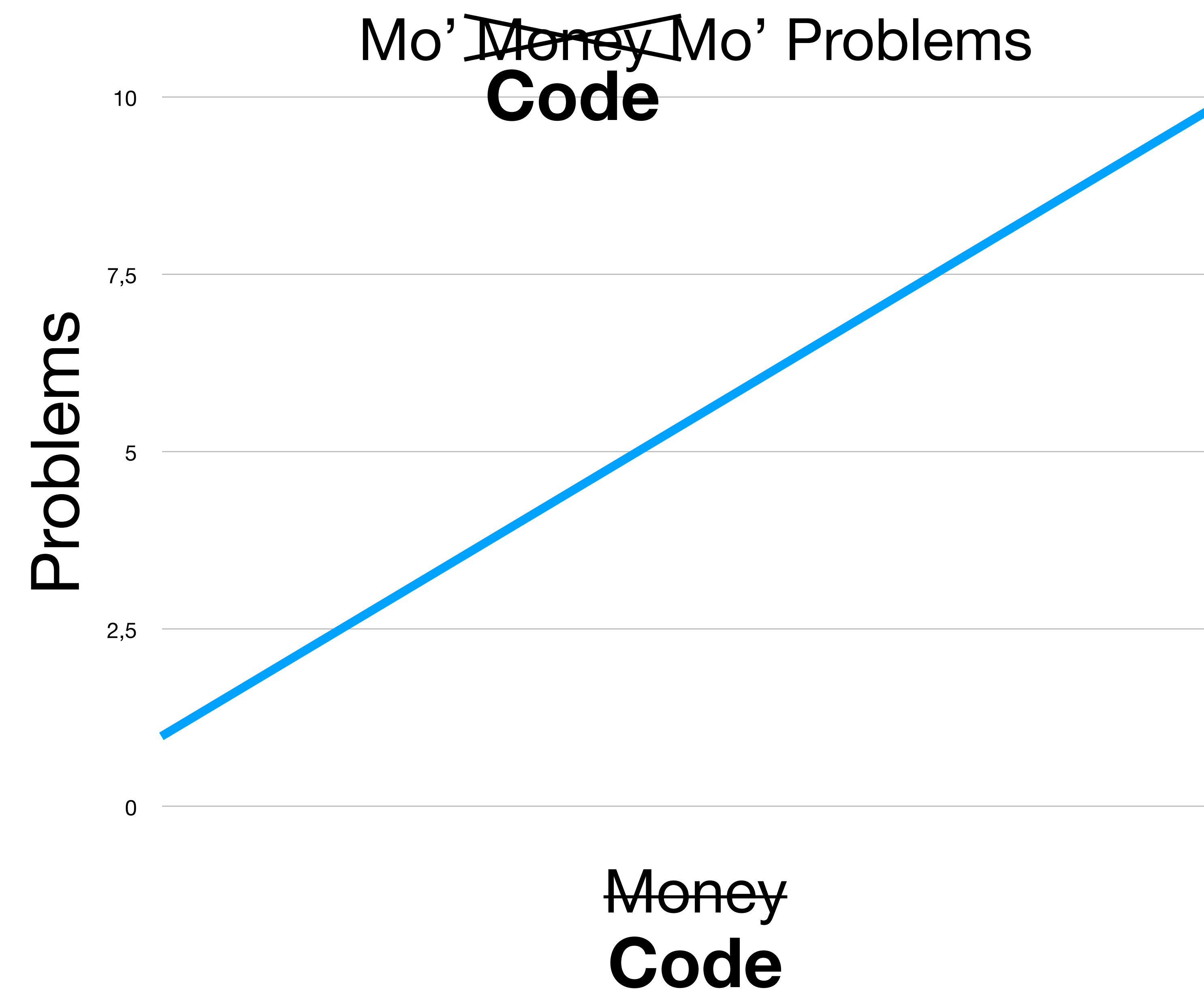


- What are those “batteries”?
  - Complex SQL support
  - Types! (decimals, timestamps, nested types, …)
  - Many functions, aggregates, window functions, …
  - Readers for many formats (Parquet, CSV, DataFrames, …)
  - Parallel processing
  - Time zones, collations, …

What is DuckDB?



- That all sounds nice
  - Buut…



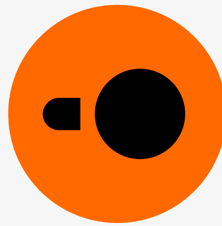
- More features → More code → More problems

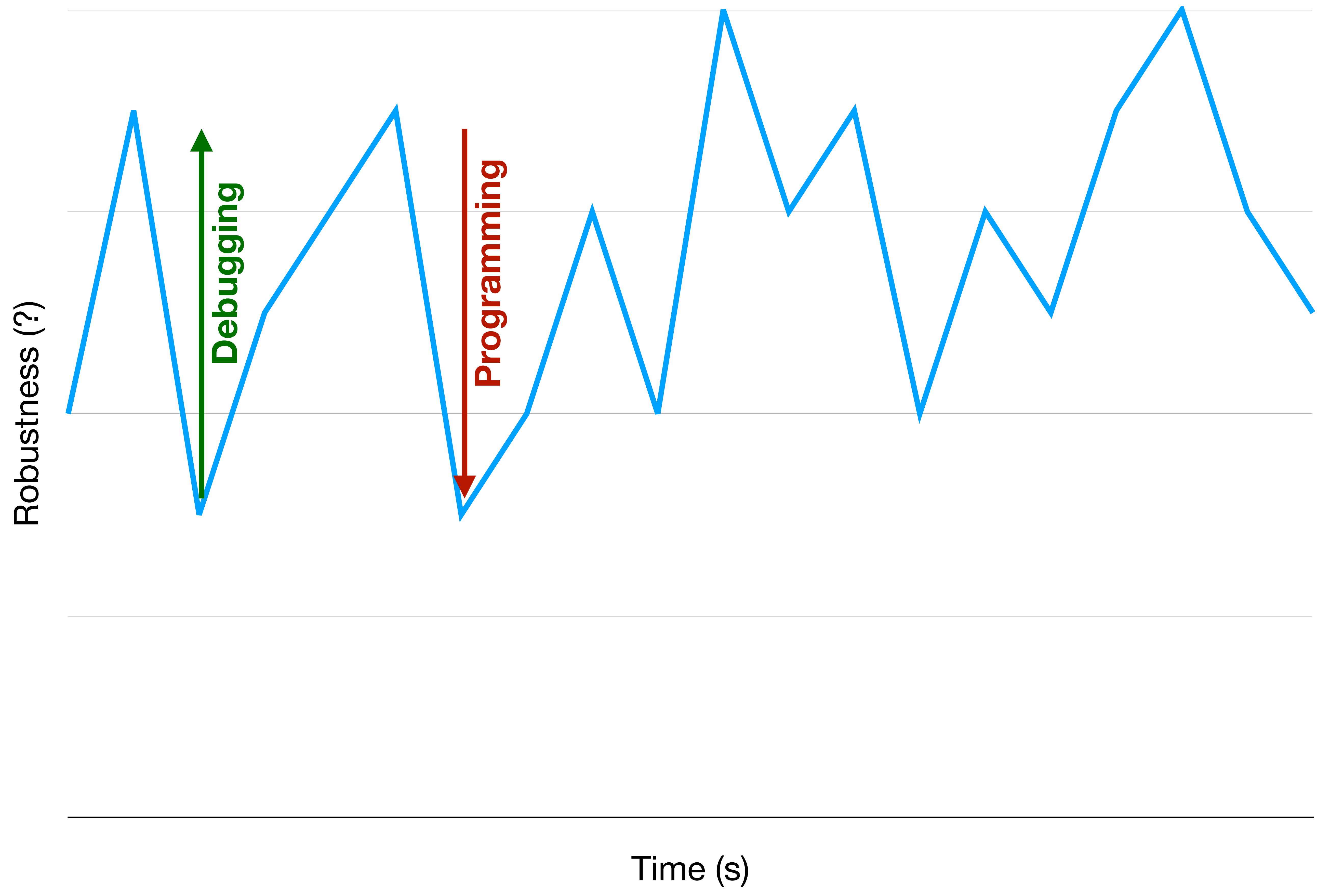
Features vs Robustness

"If debugging is the process of removing software bugs, then programming must be the process of putting them in"

Edsger W. Dijkstra

Features vs Robustness





Features vs Robustness

- **Robustness** and **feature development** are at odds

- Yet we want both!
- What can we do?

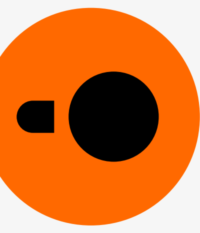
Database Testing

Database Testing

- Why is testing so important?
  - Verify that code is correct
  - Prevent bugs from re-appearing!
  - Catch bugs before they become a problem!

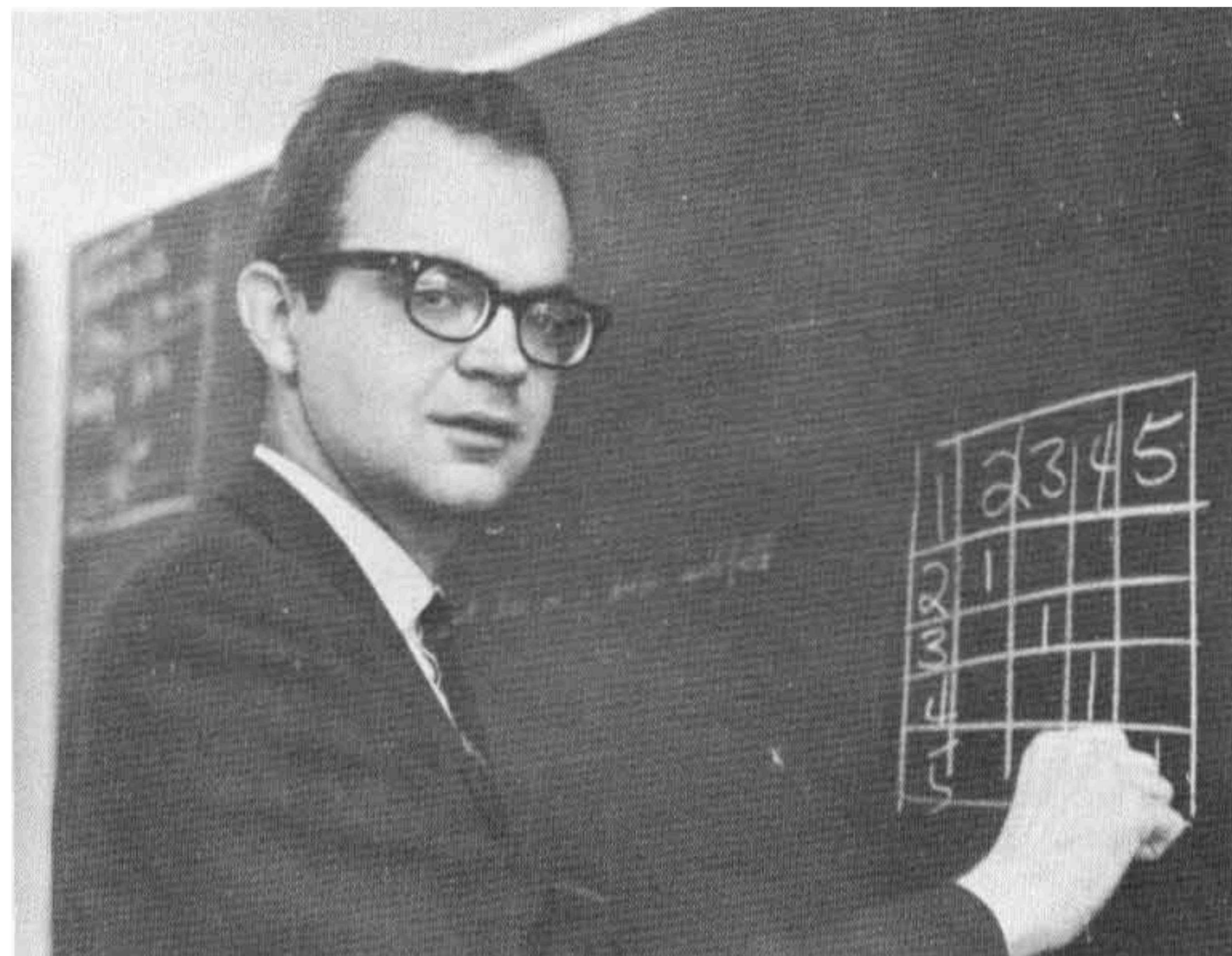
Preaching to the choir

Database Testing

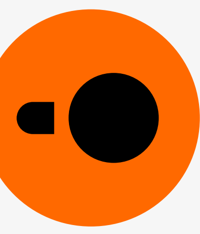


"Beware of bugs in the above code; I have only proved it correct, not tried it."

Donald Knuth



Database Testing



- Testing is crucial for feature development!
- It allows you to…
  - **Add new features**
  - **Optimize code**
  - **Refactor**
- **..** without breaking existing functionality*

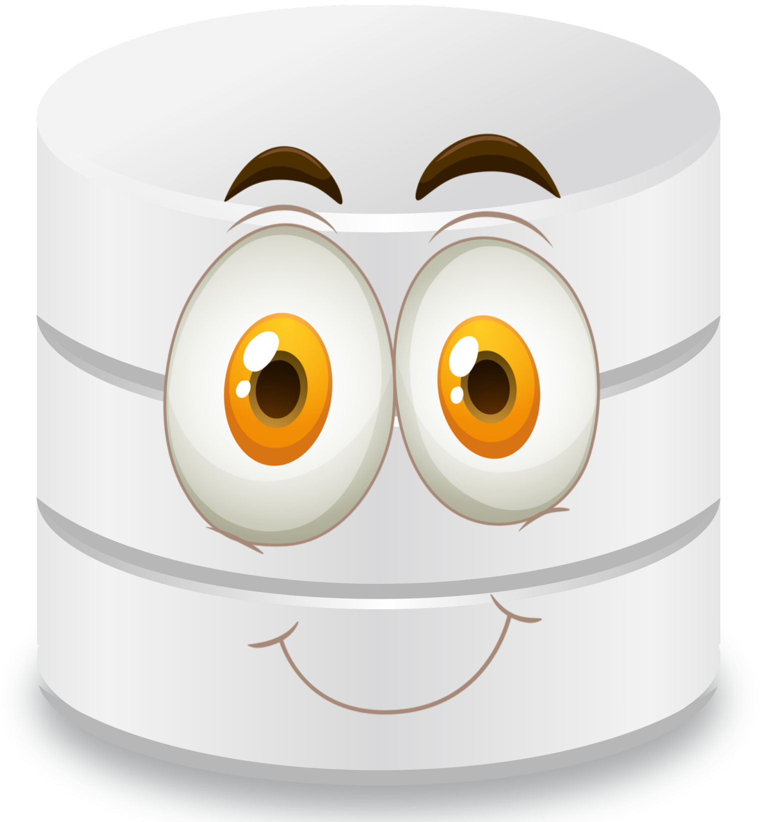
At least functionality that is tested…

Database Testing

- Testing is important for **software development in general**
  - What makes databases special?

- Database system development:
  - **Double-edged sword**

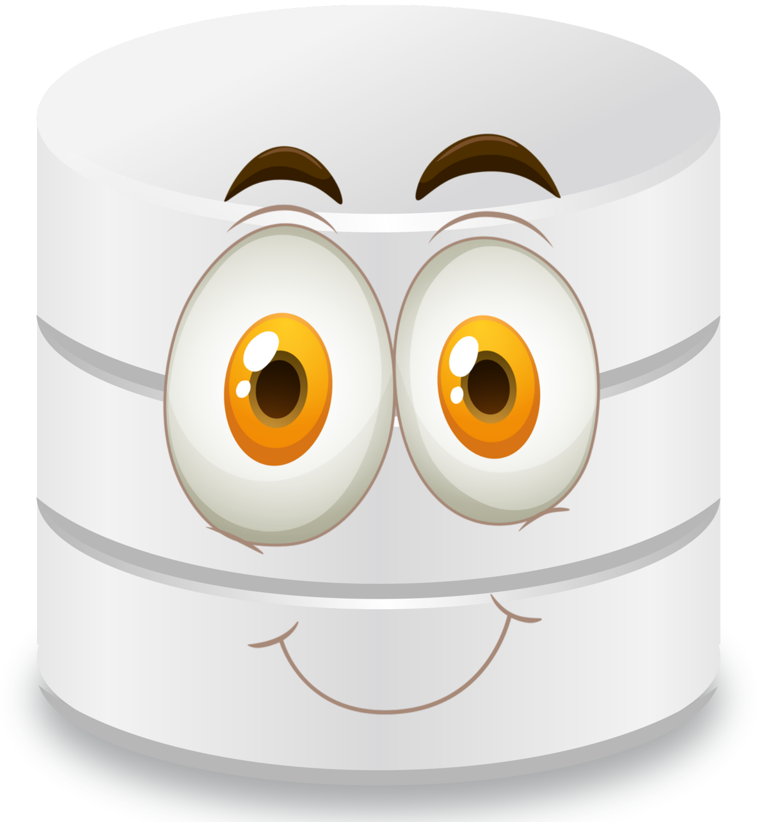
Database Testing



- Functionality is (~) **well-defined**
  - SQL is an ANSI Standard

- Changes in requirements are **unlikely**
  - At least from a functionality perspective…

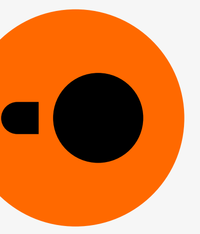
Database Testing

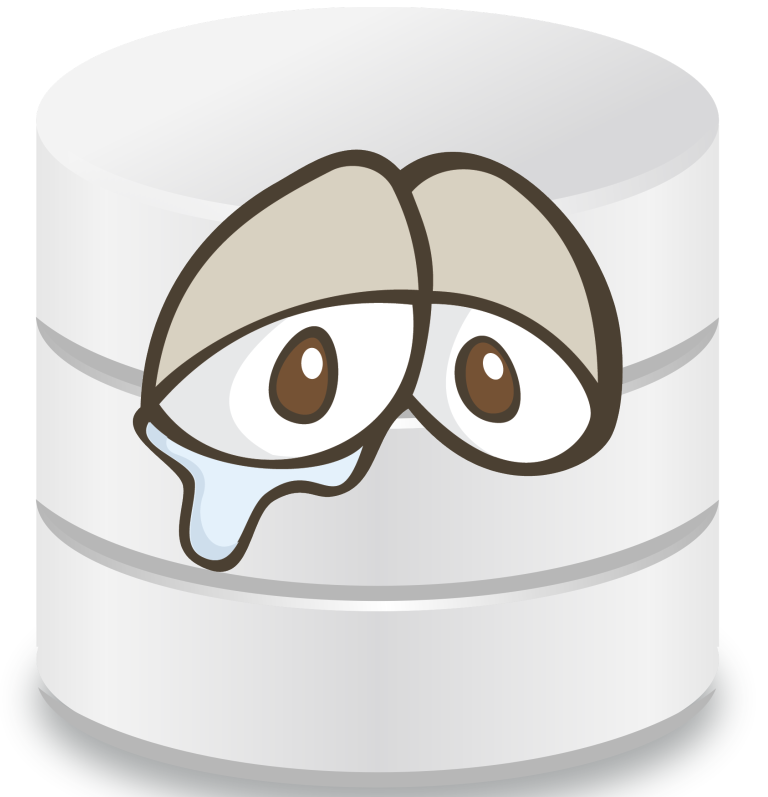


- Write SQL tests once
  - They will be valid forever!*
- **Adopt tests** from other systems
- There is a **source of truth***
  - Verify new code against other systems

*Except for custom extensions

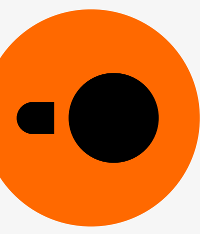
Database Testing

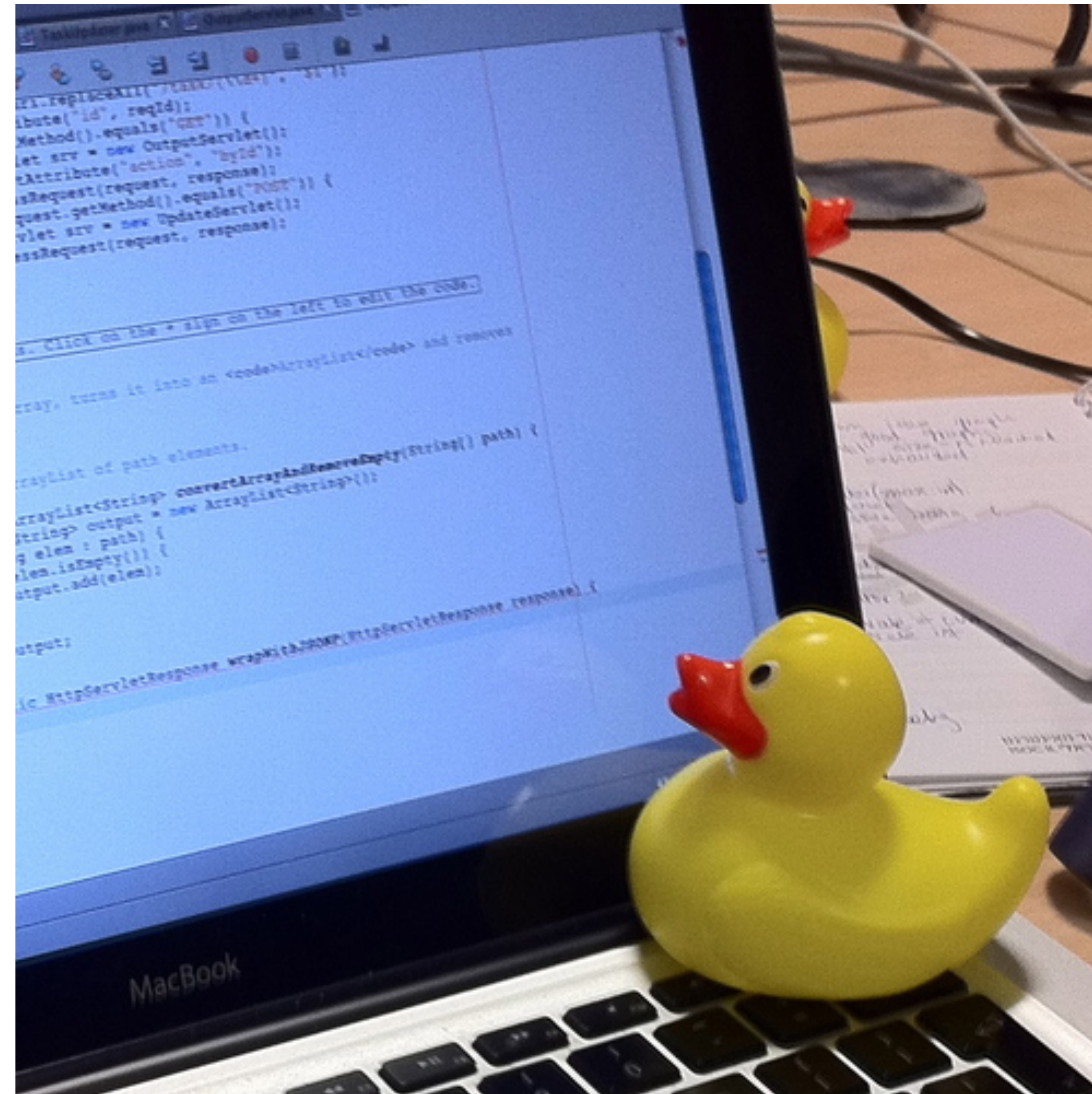




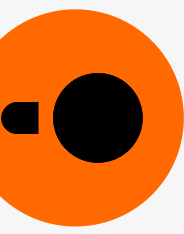
- People have **high expectations** of robustness
- DBMS need to be robust in **extreme scenarios**
  - Low memory, power outages, …
- DBMS need to be robust against **many inputs**
  - Many different data distributions/query workloads

Testing in DuckDB





Testing in DuckDB

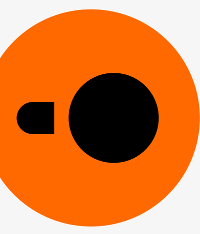


- The first test in DuckDB (22 July 2018)

```
duckdb_database database;
duckdb_connection connection;
duckdb_result result;

if (duckdb_open(NULL, &database) != DuckDBSuccess) {
    fprintf(stderr, "Database startup failed!\n");
    return 1;
}

if (duckdb_connect(database, &connection) != DuckDBSuccess) {
    fprintf(stderr, "Database connection failed!\n");
    return 1;
}

if (duckdb_query(connection, "SELECT 42;", &result) != DuckDBSuccess) {
    return 1;
}
```

- Problems
- **C API** - Failures result in memory leaks
  - Clutters failed tests with sanitizer leak errors!
- Repeated boilerplate

Testing in DuckDB



- Moved to use Catch framework
- C++ Tests
- Mix of SQL tests and internal component tests

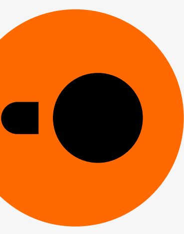
```
TEST_CASE("Date parsing works", "[date]") {
    REQUIRE(Date::ToString(Date::From
    REQUIRE(Date::ToString(Date::From
            Date::Format(1992, 1, 1))
    REQUIRE(Date::ToString(Date::From
            Date::Format(1992, 10, 10
    REQUIRE(Date::ToString(Date::From
            Date::Format(1992, 9, 20)
    REQUIRE(Date::ToString(Date::From
            Date::Format(1992, 12, 31
```

```
TEST_CASE("Test TPC-H SF0.01", "[tpch]") {
    float sf = 0.01;

    // generate the TPC-H data for SF 0.01
    DuckDB db(nullptr);
    REQUIRE_NOTHROW(tpch::dbgen(sf, db.catalog));

    // test the queries
    DuckDBConnection con(db);

    // check if all the counts are correct
    unique_ptr<DuckDBResult> result;
    REQUIRE_NOTHROW(result = con.Query("SELECT COUNT(*) FROM orders"));
```

Testing in DuckDB

- Problems
- Every test needs to be compiled and linked
    - Change in common header → Recompile everything!

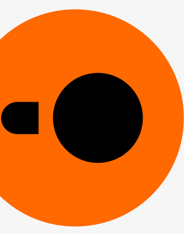
- Problem if you have many tests
    - Which you **really** want to have!



- Problems
- Component tests were **especially** problematic
  - Change to internals → need to change tests!

- **Locks in** implementation details!
  - **Internal tests** are not (really) required
  - We can just use **SQL**!

Testing in DuckDB



- **Current**: Interpreted SQLLogicTests
- Avoid internal tests **as much as possible**

```
# create table
statement ok
CREATE TABLE a (i integer, j integer);

# insertion: 1 affected row
query I
INSERT INTO a VALUES (42, 84);
----
1

query II
SELECT * FROM a;
----
42	84
```

Testing in DuckDB

- SQL Tests are great
- … but do have limits

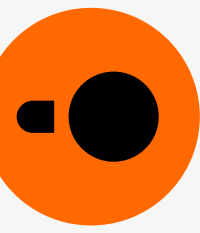
- Difficult to test:
  - Internal properties
  - Optimizers
  - Performance

Testing in DuckDB: Drive-by Testing

- **Testing internal properties**
- "Drive-by testing"

- Idea: we are running millions of queries
- Can the system **verify internals while running?**

Testing in DuckDB: Drive-by Testing



- Verification flag is enabled in most tests

```
# name: test/sql/projection/t
# description: Test simple pr
# group: [projection]

statement ok
PRAGMA enable_verification
```

Testing in DuckDB: Drive-by Testing

- **Verification**: Optimized vs Unoptimized
  - Run query **with optimization**
  - Run query **without optimization**
- Verify that the same result is produced

- **Powerful!**

Testing in DuckDB: Drive-by Testing

- Verify functions that round-trip
  - `(De)serialize`, `Copy`, `ToString` → `Parser`

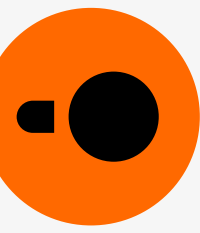
- Re-running after round-trip should provide **same result**
  - Finds problems in these methods **immediately**
  - Otherwise **hard to detect**

Testing in DuckDB: Drive-by Testing

- Verify **internal state** of intermediates
  - UTF8-validity
  - No null-bytes in strings
  - List offsets and lengths are in bounds

- Verify **statistics**
  - NULL-ness, min/max, unicode, …

Testing in DuckDB: Drive-by Testing



- More **bang** for your buck!
- Extract maximum value from your test suite

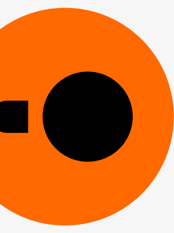
- Downside: tests are slower
  - Esp. if the optimizer turns a cross-product into a join!

Testing in DuckDB: Optimizer Tests

- **Optimizers** are hard to verify
  - Query **should be** correct with or without optimizer!

- How do we know **the optimizer had an effect?**

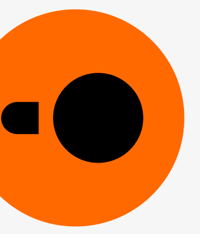
Testing in DuckDB: Optimizer Tests



- **Idea**: check `EXPLAIN` output
- Compare to result of optimization
  - if SQL rewrite is possible

```
statement ok
PRAGMA explain_output = OPTIMIZED_ONLY;

# no special symbols optimization: aaa -> S=a
query I nosort nosymbols
EXPLAIN SELECT S LIKE 'aaa' FROM test
----

query I nosort nosymbols
EXPLAIN SELECT S='aaa' FROM test
----
```

Testing in DuckDB: Optimizer Tests



- If SQL rewrite is not possible: **inspect plan directly**
  - **Downside**: breaks if `EXPLAIN` output changes significantly

```
# between where lhs is bigger than rhs: we can prune this entirely
query II
EXPLAIN SELECT * FROM integers WHERE i BETWEEN 3 AND 2
----
logical_opt→<REGEX>:.*EMPTY_RESULT.*
```

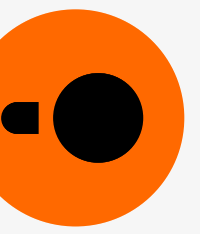
Development in DuckDB

Development in DuckDB

- Rely heavily on **test-driven development**
  - Tests are easy to write & run
  - Encourage writing **many tests**

- Avoid exclusively **happy-path tests**
  - Think of as many corner cases as possible

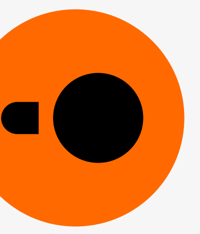
Development in DuckDB



- Development happens in **forks**
  - **main branch** should always be stable

- **Target**: able to create a release from main **at any point**
  - We continuously publish pre-releases!

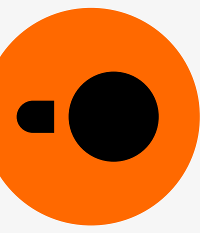
Development in DuckDB



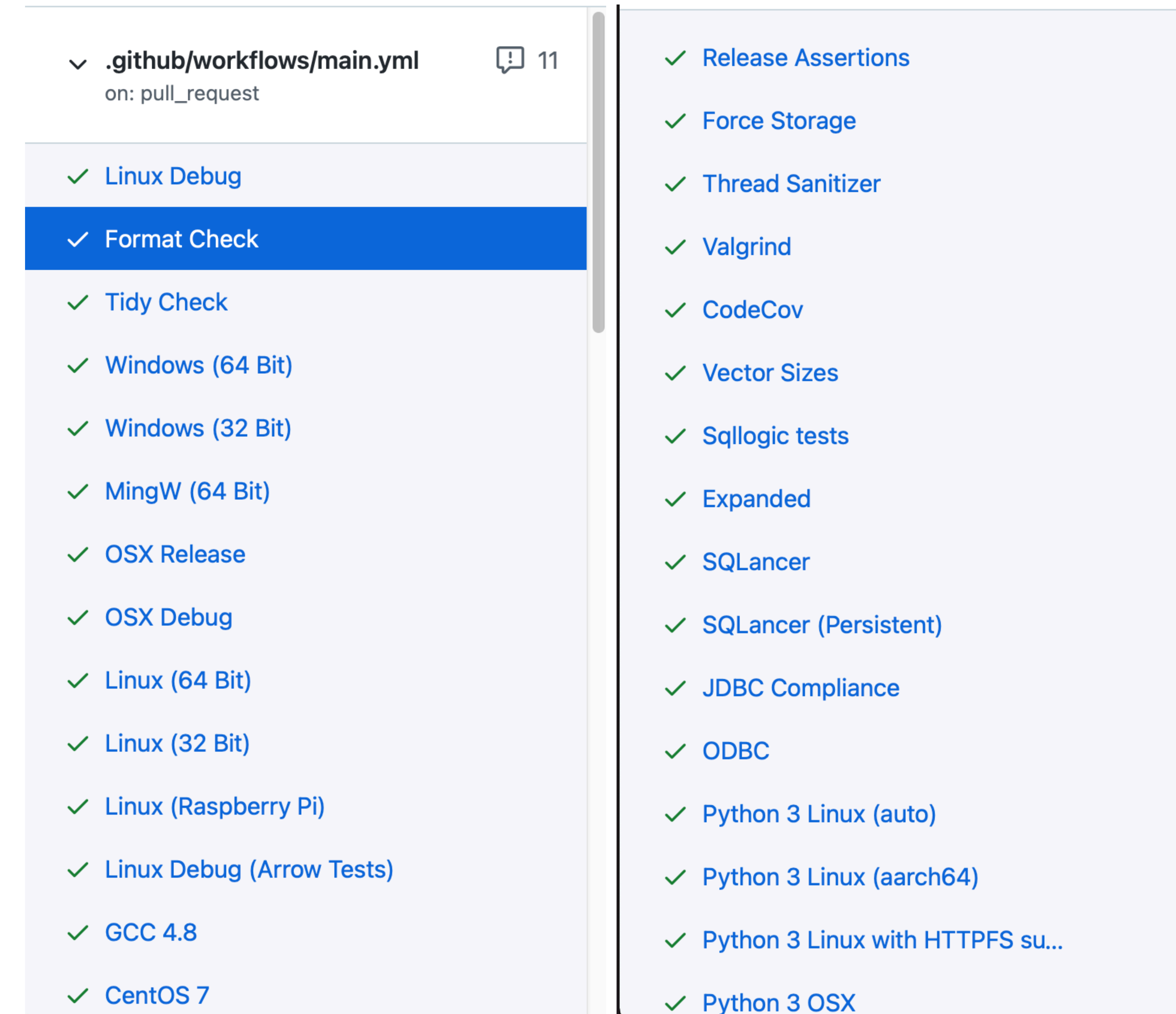
- Continuous Integration (CI) ensures **stability**
  - Code that breaks CI is **not allowed to be merged** into main

- Our experience:
- Anything that is not **automatically verified by CI** will (eventually) be broken!

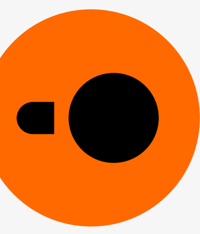
Development in DuckDB



- Extensive CI suite
  - Currently **79** workflows
- Verify that tests pass
- Compilation on many setups
- Client interfaces



Development in DuckDB

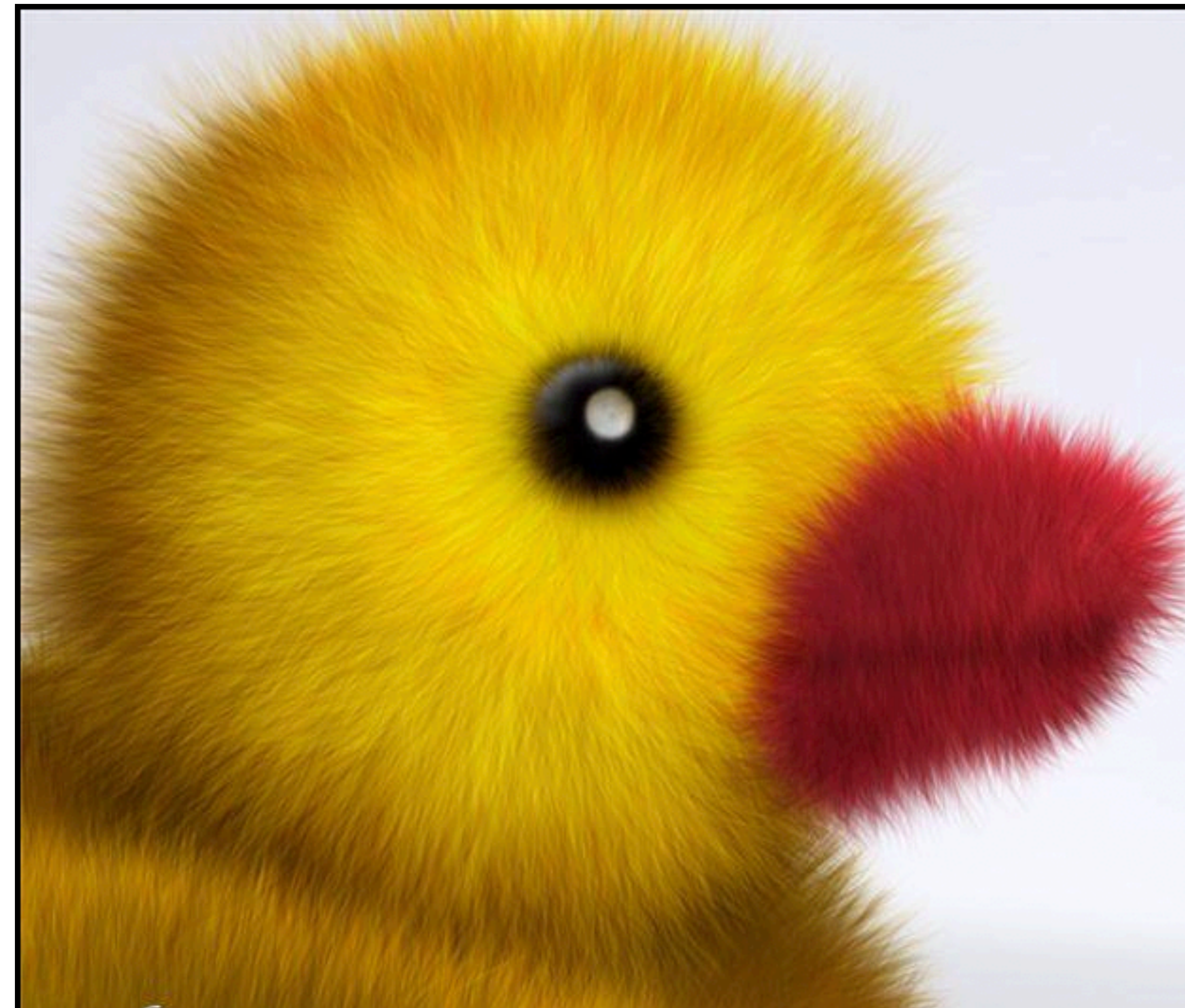


- Verify **code style** and **formatting**
  - No need for comments in PRs about code style!
- Many different configurations:
  - Different vector sizes
  - In-memory/persistent back-end
  - Many sanitizers (TSan, ASan, LeakSan, etc)
- **Maximize** number of bugs detected

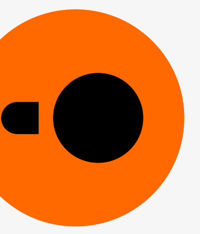
.github/workflows/
on: pull_request

- Linux Debug
- Format Check
- Tidy Check
- Windows (64 Bit)
- Windows (32 Bit)
- MingW (64 Bit)
- OSX Release
- OSX Debug
- Linux (64 Bit)
- Linux (32 Bit)
- Linux (Raspberry Pi
- Linux Debug (Arrow
- GCC 4.8
- CentOS 7

Fuzzing in DuckDB

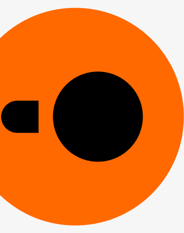


Fuzzing in DuckDB



- **2020**: we thought we had a pretty robust system…
- Thousands of tests from various systems
  - SQLite, Postgres, …
- Hundreds of our own tests

Fuzzing in DuckDB



- … and then Dr. Rigger came along!

mrigger commented on 25 Apr 2020

Contributor

Consider the following statements:

```
CREATE TABLE t0(c0 INTEGER);
INSERT INTO t0(c0) VALUES (-2);
SELECT t0.c0 FROM t0 WHERE -1 BETWEEN t0.c0::VARCHAR AND 1; -- expected: {-2}, actual: {}
```

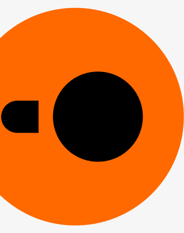
Unexpectedly, the `SELECT` does not fetch the row. I found this based on commit `1d2e40e`.

Fuzzing in DuckDB

- Using SQLancer, Manuel found **80~** bugs in DuckDB
  - Not found using test suites of other systems!

- DBMS are **complex**
  - Each system has **their own bugs**
- Tests from other systems are **extremely helpful**
  - … but not a silver bullet!

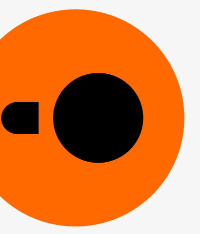
Fuzzing in DuckDB



- Fuzzers find **many bugs initially**
- Diminishing returns

- Fuzzers find bugs in a **specific domain**
  - After all bugs are fixed, few (if any) new bugs are found

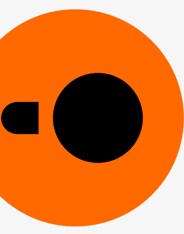
- Important to run many different types of fuzzers!



- Running a fuzzer once is not enough
  - At least in an evolving system

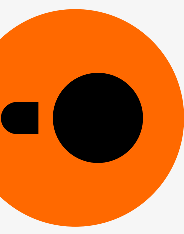
- Fuzzers must be **run continuously!**
  - New code has to be fuzzed extensively as well!
  - Might have bugs in the domain of the fuzzer

Fuzzing in DuckDB



- We can continuously fuzz on a separate machine
- We tried this…
    - Machine needs attention (e.g. when it goes down)
    - If fuzzer is no longer rapidly uncovering issues…
- Fizzled out

Fuzzing in DuckDB



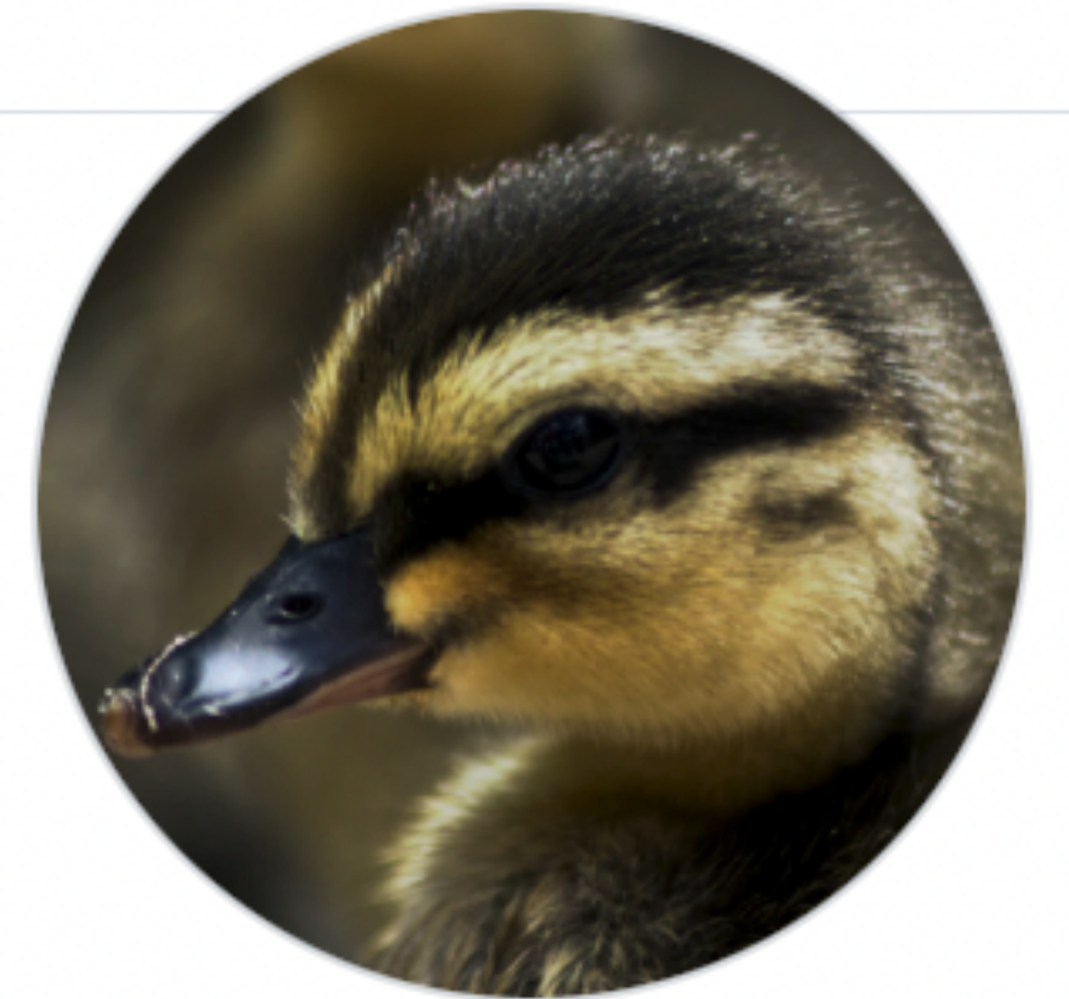
- **Idea**: run fuzzers in CI!
- **Problem**: fuzzers are unpredictable
  - CI will fail in unrelated commits/PRs!

- Not an option if failing CI is used to block merges
  - Which it should be!

Fuzzing in DuckDB

- **New idea:** fuzz, but **do not fail the CI** on problems
  - Instead, automatically file github issues

- The **Fuzzer of Ducks** was born

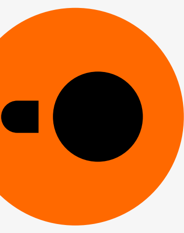


Fuzzing in DuckDB

- The **Fuzzer of Ducks** runs fuzzers in CI
  - **Now**: SQLancer and SQLSmith
- If a bug is found:
  - Creates a **reproducible test case**
  - Performs **test-case reduction**
  - Files an issue*

*Also auto-closes fixed issues!

Fuzzing in DuckDB

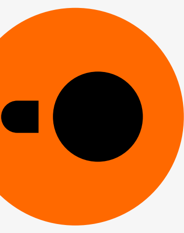


- Test case reduction: done using DuckDB!
  - Eat your own duck food

- `reduce_sql_statement` shows reduction candidates
  - Reduce while problem persists, or until timeout

```
D SELECT * FROM reduce_sql_statement('SELECT * FROM tbl ORDER BY 1, 2');
┌──────────────────────────────────────────┐
│                   sql                    │
├──────────────────────────────────────────┤
│ SELECT * ORDER BY 1, 2                   │
│ SELECT * FROM tbl ORDER BY 2             │
│ SELECT * FROM tbl ORDER BY 1             │
│ SELECT * FROM tbl                        │
│ SELECT NULL FROM tbl ORDER BY 1, 2       │
└──────────────────────────────────────────┘
```

Fuzzing in DuckDB



fuzzerofducks commented 1 hour ago

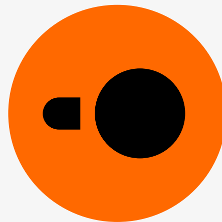
To Reproduce

```
CREATE TABLE t0(c0 BOOLEAN DEFAULT(true), c1 VARCHAR);
INSERT INTO t0(c0, c1) VALUES (534898561, 1156365055), (524523641, '0.46680525959210206');
SELECT NULL FROM t0 ORDER BY strptime(NOT(t0.c0 BETWEEN t0.rowid AND t0.rowid), 1407974306)
```

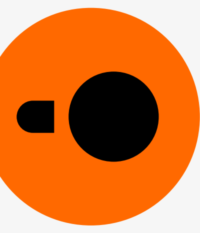
Error Message

```
/usr/include/c++/9/bits/stl_vector.h:1043:34: runtime error: reference binding to null pointer of type 'stru
```

Performance Regression Testing



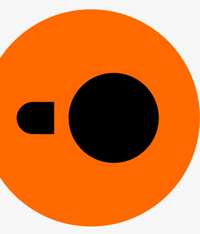
Performance Regression Testing



- **SQLLogicTests** detect regressions in **functionality**
  - But not regressions in **performance!**

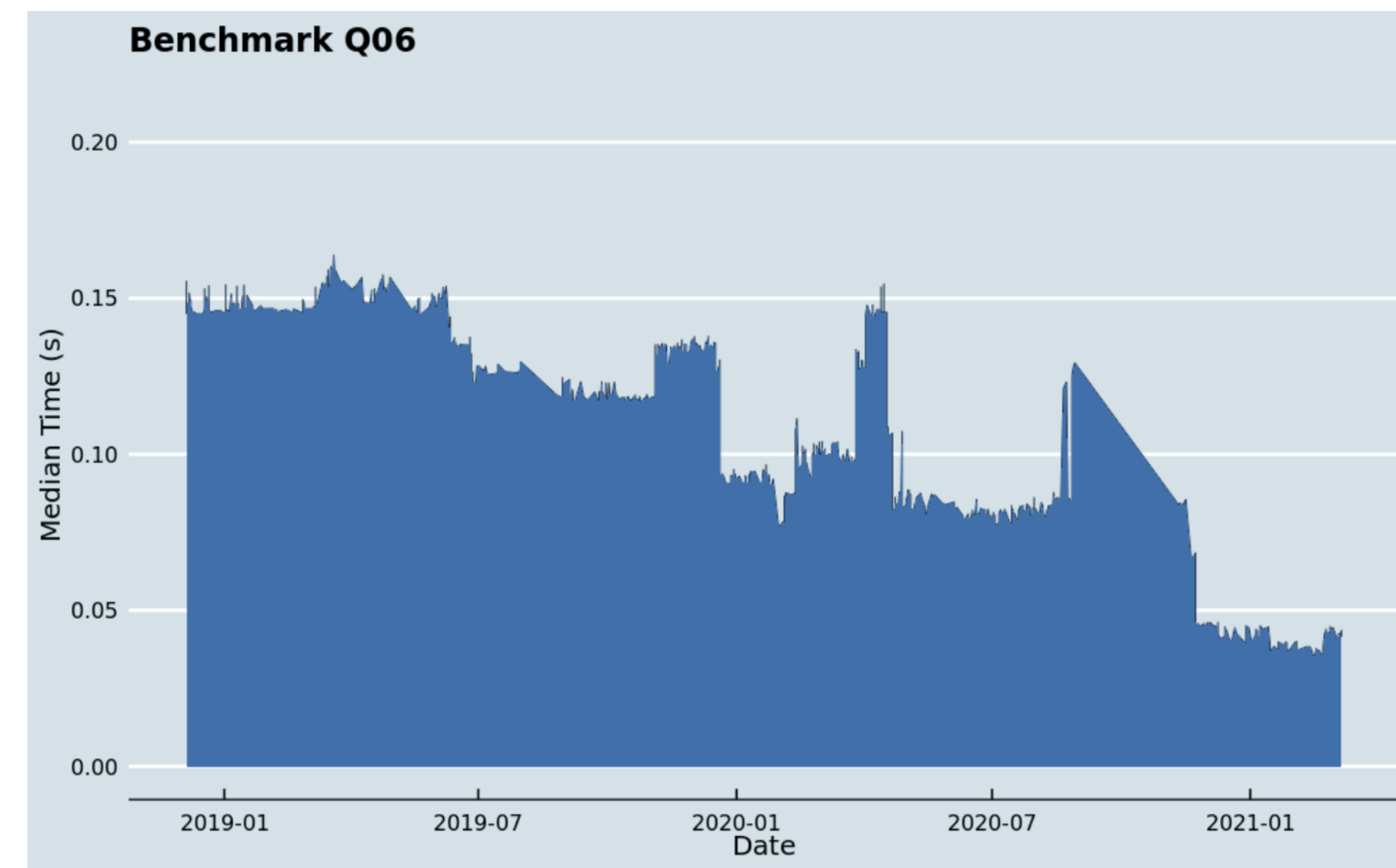
- Performance is **important** for DBMS!
  - How do we **maintain good performance?**

Performance Regression Testing



- Initial version:
- Benchmark **each commit** on separate machine
- Generate **history of performance timings**

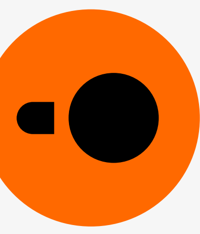
	d9bc	740c	3555	3b53	7299
Q01	2.95 [Q/L/E]	2.92 [Q/L/E]	1.68 [Q/L/E]	1.69 [Q/L/E]	0.83 [Q/L/E]
Q02	0.11 [Q/L/E]	0.11 [Q/L/E]	0.11 [Q/L/E]	0.11 [Q/L/E]	0.11 [Q/L/E]
Q03	0.25 [Q/L/E]	0.25 [Q/L/E]	0.26 [Q/L/E]	0.24 [Q/L/E]	0.25 [Q/L/E]
Q04	0.24 [Q/L/E]	0.24 [Q/L/E]	0.23 [Q/L/E]	0.23 [Q/L/E]	0.24 [Q/L/E]



Performance Regression Testing

- **Useful**
  - Detects regressions per commit
  - Visualizes trend lines
- **Problem:** The machine broke…
  - You can guess the rest

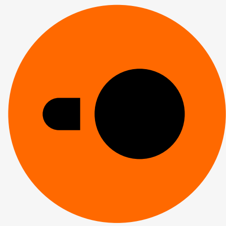
Performance Regression Testing



- **Idea:** Run regression tests in the CI!
  - Run a **representative subset** of benchmarks
  - Fail CI if **significant** regression is found
- **False positive** is annoying!
  - Run many times
  - Only if **all** runs regress, fail

```
332
333  ==========================================================
334  ============== NO REGRESSIONS DETECTED  ======
335  ==========================================================
336
337  benchmark/tpch/sf1/q01.benchmark
338  Old timing: 0.379769
339  New timing: 0.348475
340
341  benchmark/tpch/sf1/q02.benchmark
342  Old timing: 0.052537
343  New timing: 0.054486
344
```

Future Testing Plans



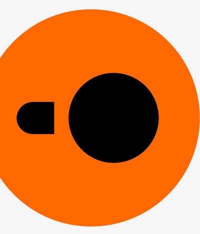
Future Testing Plans

- **Verify result-equivalent operators**
  - e.g. nested loop join and hash join
  - Try both in a query → verify same result

Future Testing Plans

- **Integrate additional fuzzers**
  - AFLplusplus trained on our test suite
  - KLEE for testing edge-cases in input (Parquet, CSV, …)

Future Testing Plans

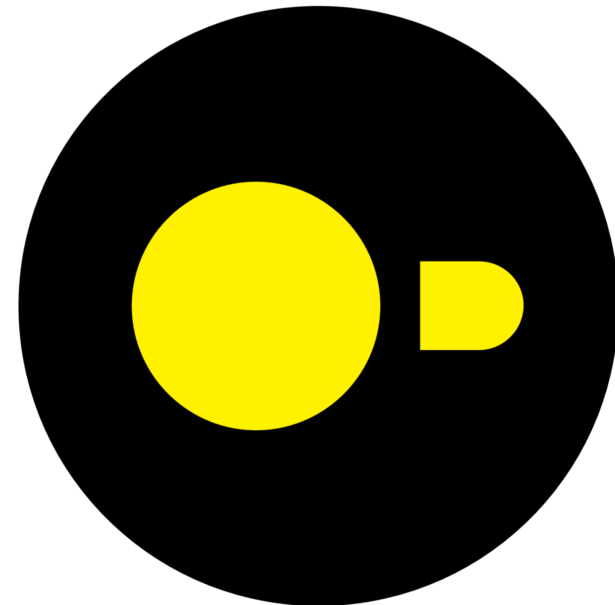


- **Backwards compatibility testing**
  - **Goal**: v1.0 onwards should be backwards compatible
  - Use existing test suite to test this
    - Create database using old version
    - Run tests using new version

Future Testing Plans

- **Destructive testing**
  - Introduce memory allocation errors
  - Crash/power loss tests
  - Introducing disk/ram errors

Thanks for having me!



Any questions?