

Where It Came From And Where It Is Going

BerkeleyDB
LevelDB
RocksDB
etc ...



- Embedded
- Key/Value





- Embedded
- SQL





- Client/Server
- SQL



SQLite used in....

- Every Android device (~2.5 billion)
- Every Mac and iOS device (~1.5 billion)
- Every Win10 machine (~2 billion)
- Every Chrome and Firefox browser (~5 billion)
- Every Skype, iTunes, WhatsApp (~3 billion)
- Millions of other applications
- Many billions of running instances
- Trillions of individual database files

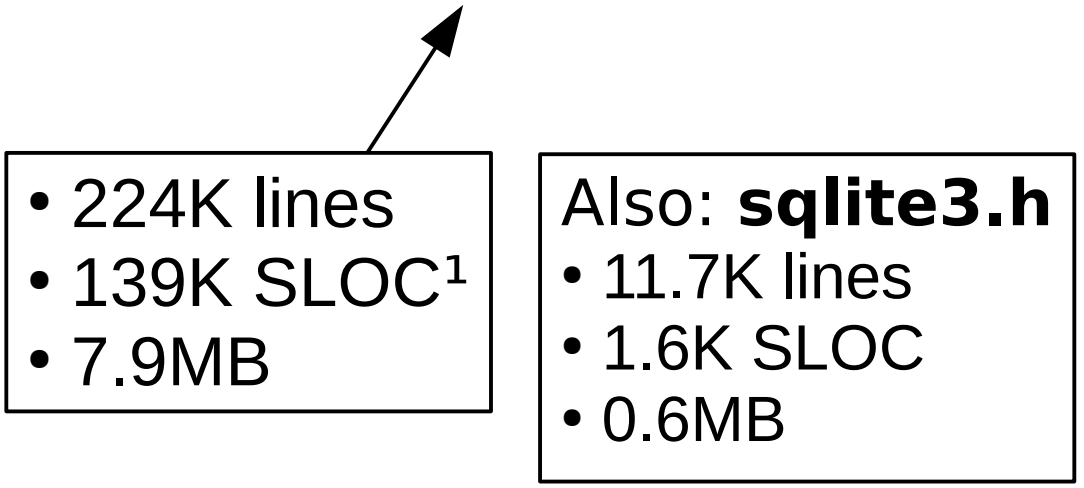
More Copies of SQLite Than...

- Linux
- Windows
- MacOS and iOS
- All other database engines combined
- Any application
- Any other library[1]

[1] *except maybe zLib*

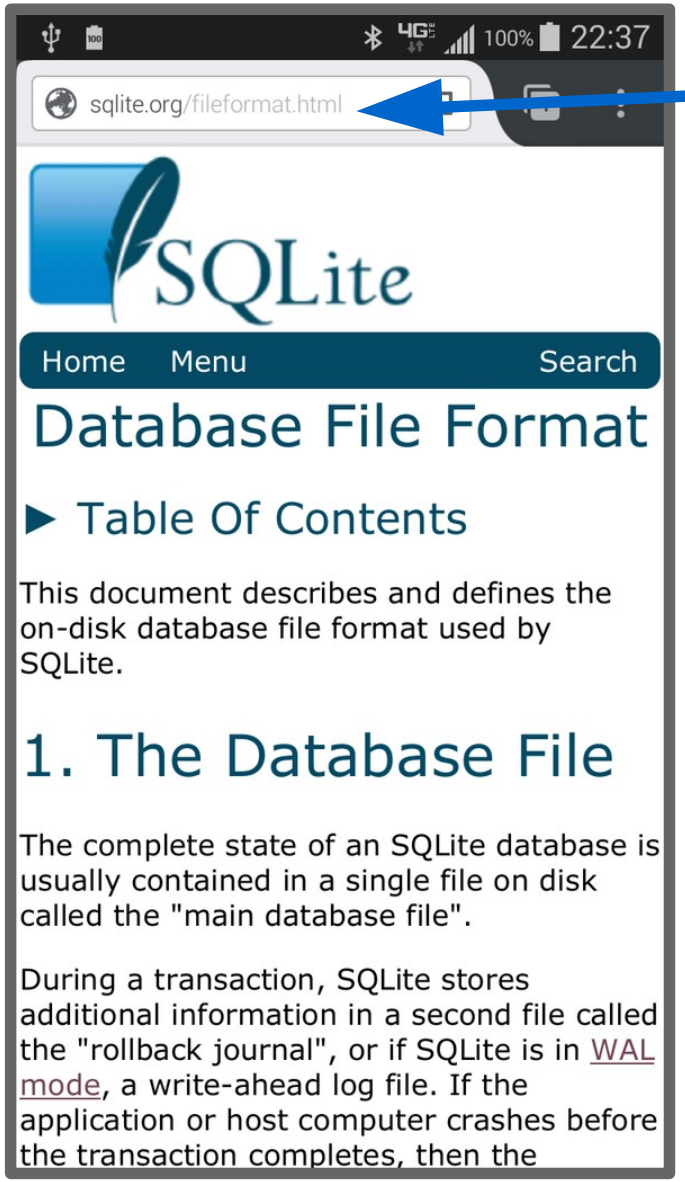
One File Of C-code

sqlite3.c



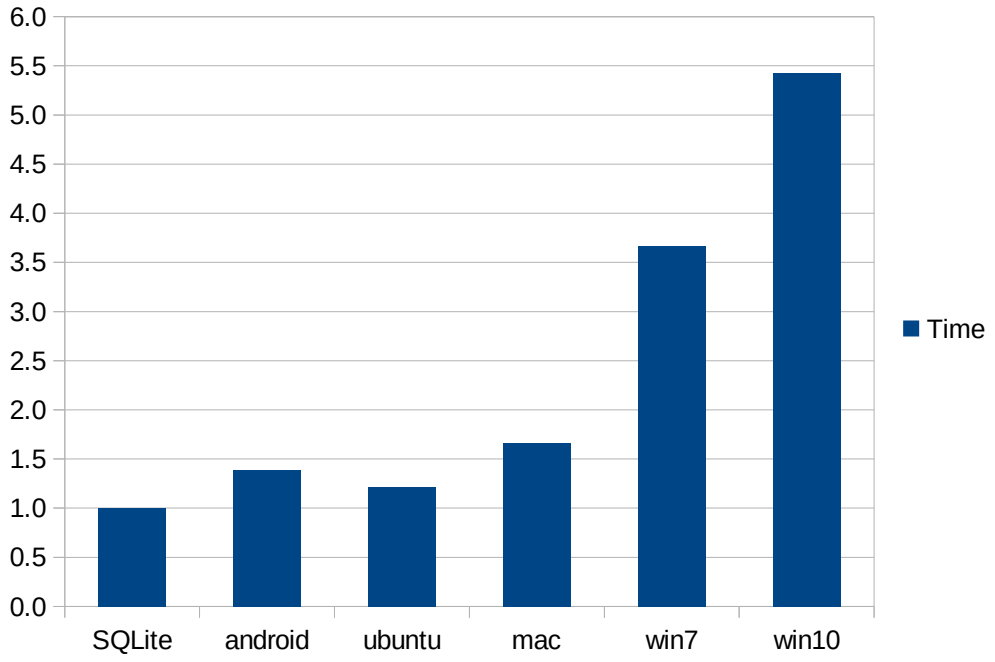
[1]SLOC: "Source Lines Of Code" - Lines of code not counting comments and blank lines.

Open File Format



- sqlite.org/fileformat.html
- Single-file database
- Cross-platform
  - 32-bit ↔ 64-bit
  - little-endian ↔ big-endian
- Backwards-compatible
- Space efficient encoding
- Readable by 3rd-party tools
- Supported through 2050

Faster Than The File System



Time to read 100,000 BLOBs with average size of 10,000 bytes from SQLite versus directly from a file on disk.

https://sqlite.org/fasterthanfs.html

Aviation-Grade Testing

- DO-178B development process
- 100% MC/DC, as-deployed, with independence



- Refactor and optimize without breaking things
- Minimize cruft
- Maintainable by a very small team

https://sqlite.org/testing.html


Copyright

Storage Decision Checklist

Remote Data?

Big Data?

Concurrent Writers?

Gazillion transactions/sec?



Otherwise



Storage Decision Checklist **FAIL!**

Remote Data?

Big Data?

Concurrent Writers?

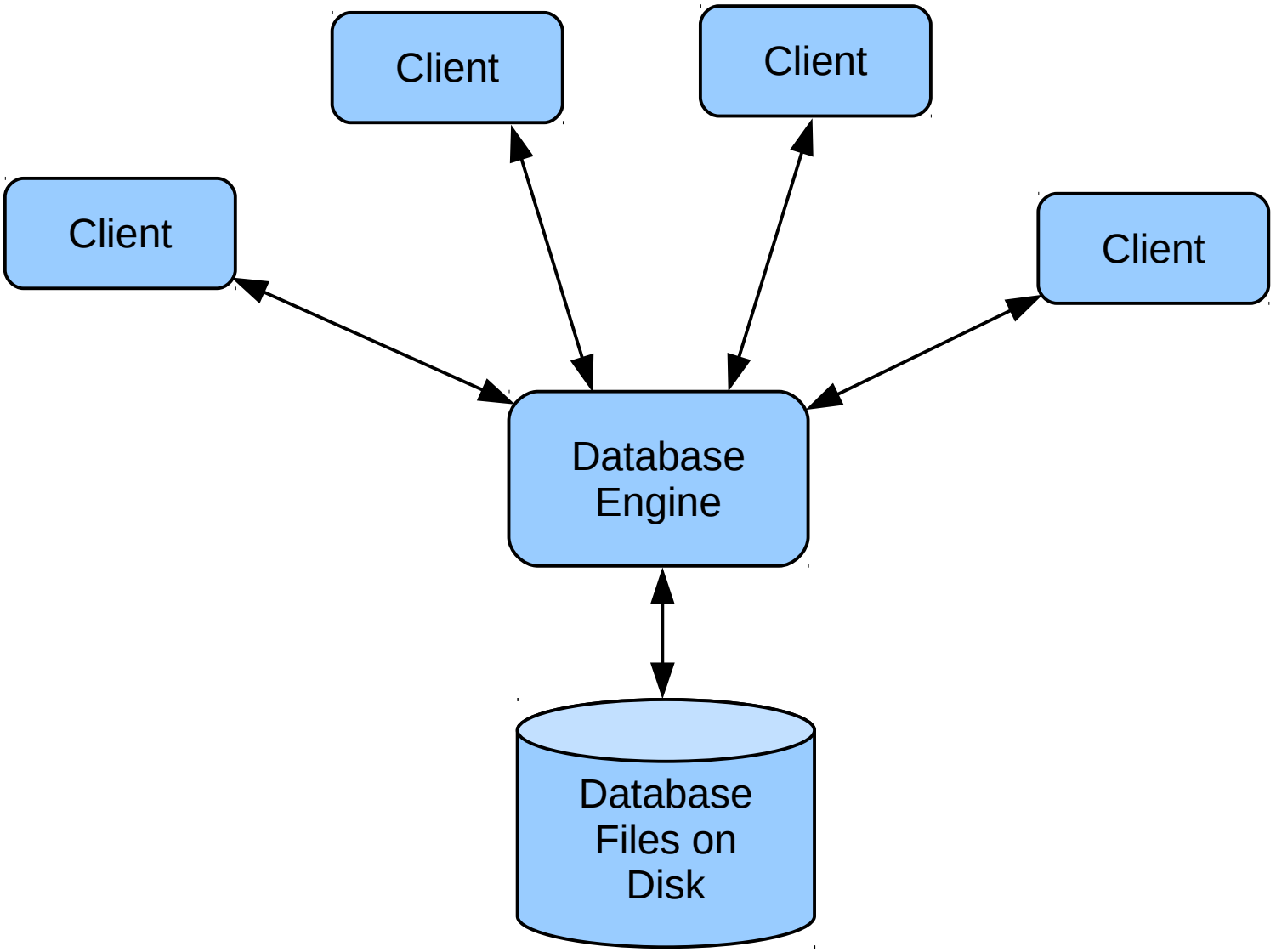
Gazillion transactions/sec?

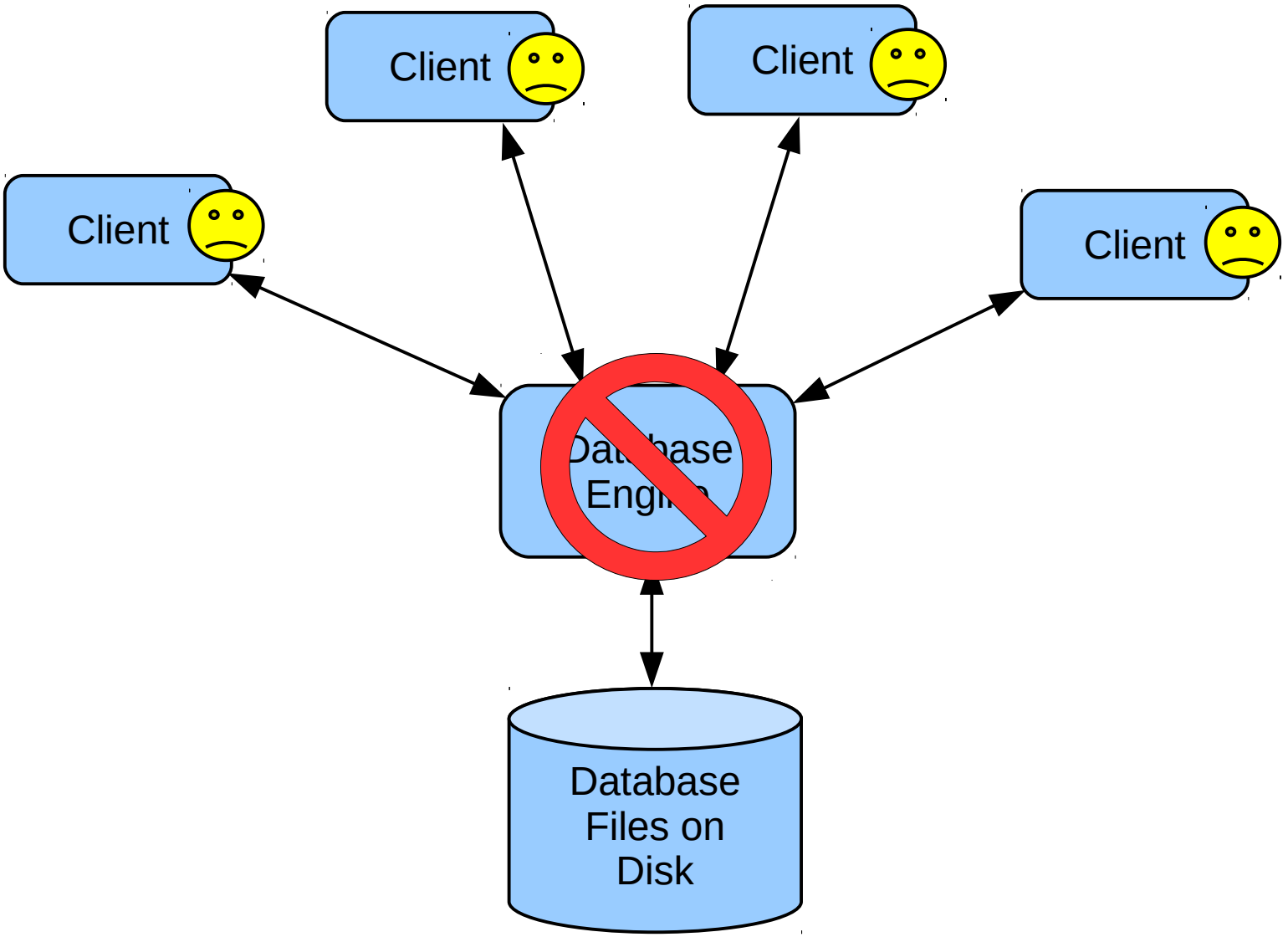


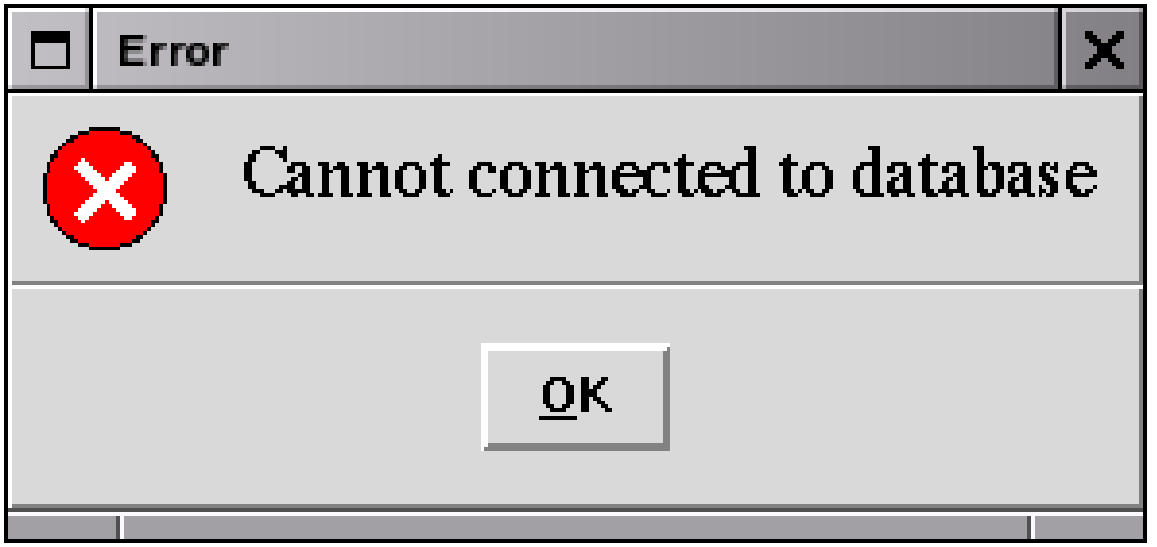
No!

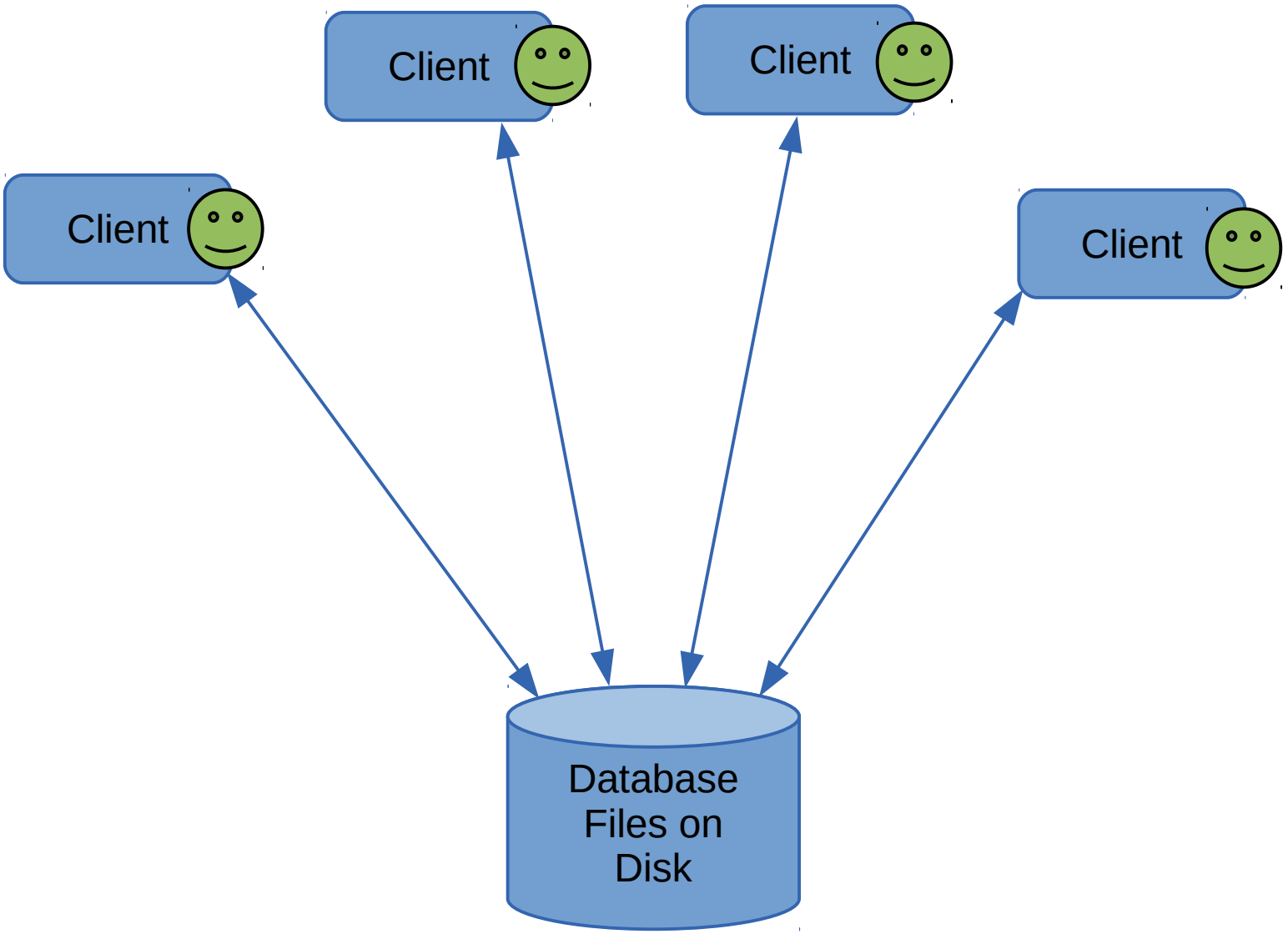
Otherwise → fopen()

Where Did SQLite Come From?


Client
Client
Client
Client
Database
Engine
Database
Files on
Disk


Client
Client
Client
Client
Database
Files on
Disk


Error
Cannot connected to database
OK



First code: 2000-05-29



- Tcl (Tool Control Language) invented by John Ousterhout in the 1980s
- Very popular in the 1990s
- Still widely used today, though less famous
- Extensible, by design
- Tk is a popular Tcl extension used for desktop GUIs

SQLite is a Tcl Extension that has escaped into the wild



Legacy Of Tcl In SQLite

- TCL bindings native to SQLite
  - Other language bindings are 3rd-party extensions
- SQLite uses flexible typing, like TCL
- The primary test cases are written in TCL
- A good chunk of SQLite source code is generated by TCL scripts
- SQLite website built by and uses TCL
- Tcl/Tk tools used in day-to-day production

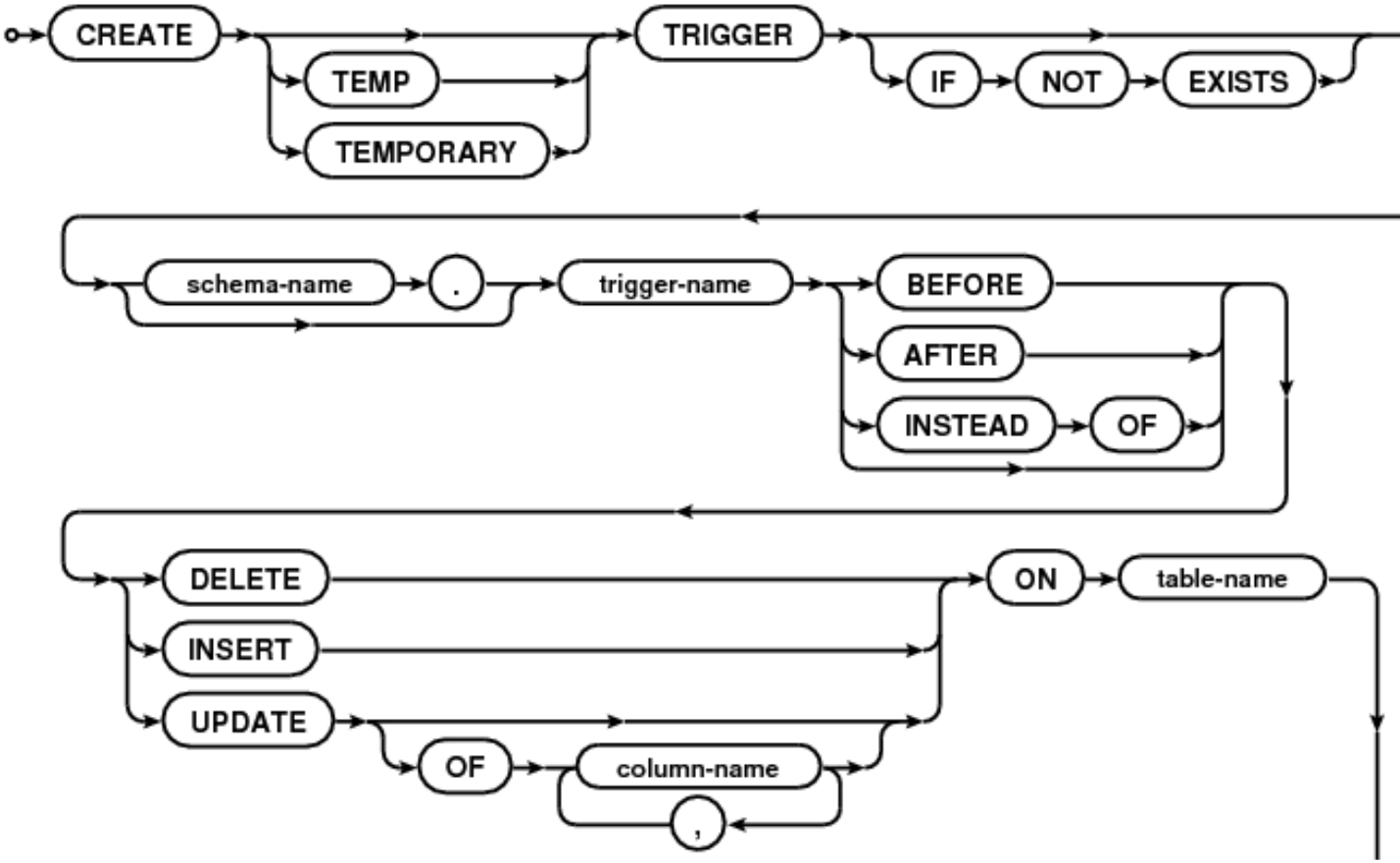
Flexible Typing

CREATE TABLE t1(a VARCHAR, b INT);

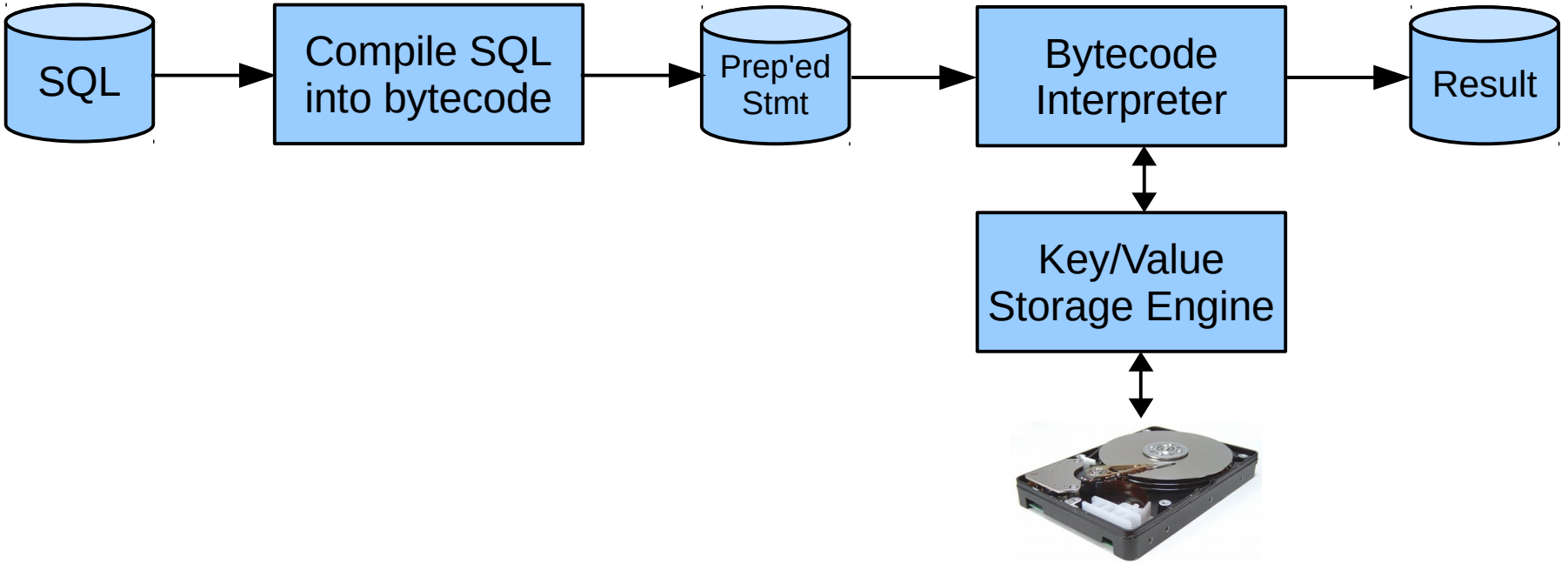
	PostgreSQL	SQLite
• INSERT INTO t1(a) VALUES(5)	√	√
• INSERT INTO t1(b) VALUES('8')	√	√
• INSERT INTO t1(b) VALUES('Hi!')	X	√
• CREATE TABLE t2(x,y,z)	X	√

Flexible Parsing

```
CREATE TRIGGER AFTER INSERT ON t1 WHEN new.a NOT NULL BEGIN
  SELECT true WHERE (SELECT a, b FROM (t1)) IN ();
END;
```



Ins & Outs of SQLite





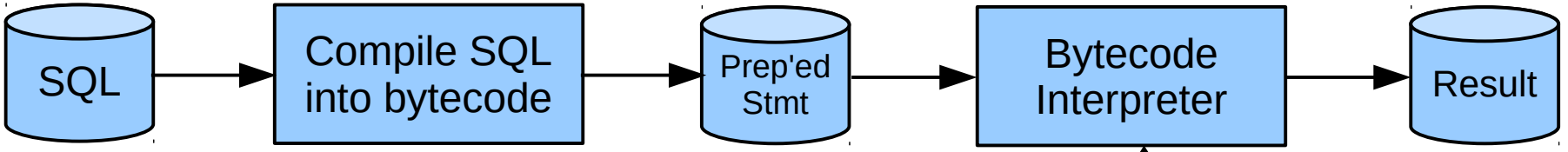
EXPLAIN SELECT price FROM tab WHERE fruit='Orange'

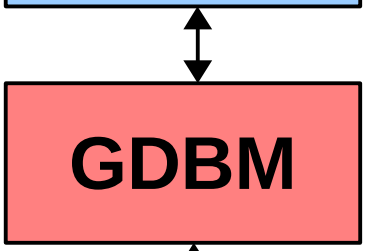
```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     12    0                    00  Start at 12
1     OpenRead       0     2     0     3              00  root=2 iDb=0; tab
2     Explain        0     0     0     SCAN TABLE tab 00
3     Rewind         0     10    0                    00
4       Column         0     0     1                    00  r[1]=tab.Fruit
5       Ne             2     9     1     (BINARY)       69  if r[2]!=r[1] goto 9
6       Column         0     2     3                    00  r[3]=tab.Price
7       RealAffinity   3     0     0                    00
8       ResultRow      3     1     0                    00  output=r[3]
9     Next           0     4     0                    01
10    Close          0     0     0                    00
11    Halt           0     0     0                    00
12    Transaction    0     0     1     0              01
13    TableLock      0     2     0     tab            00  iDb=0 root=2 write=0
14    String8        0     2     0     Orange         00  r[2]='Orange'
15    Goto           0     1     0                    00
```

Opcode documentation: **https://www.sqlite.org/opcode.html**

Documentation generated from comments in the **vdbe.c** source file.

Version 1

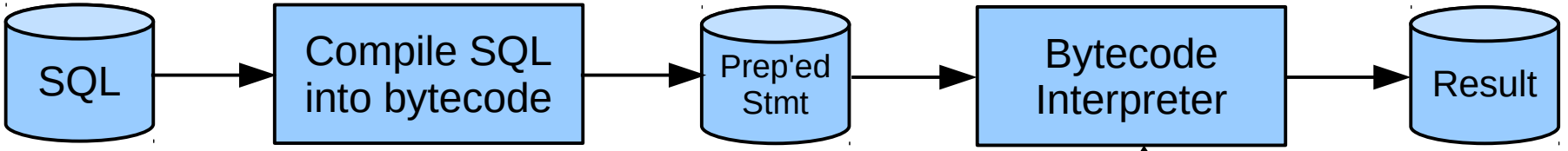


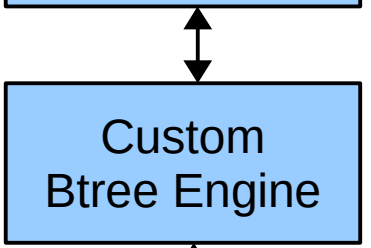




- Unordered keys
- No COMMIT or ROLLBACK
- Separate file for each table and index
- Infectious GPL license
- 2000-05-29 to 2001-07-23

Version 2

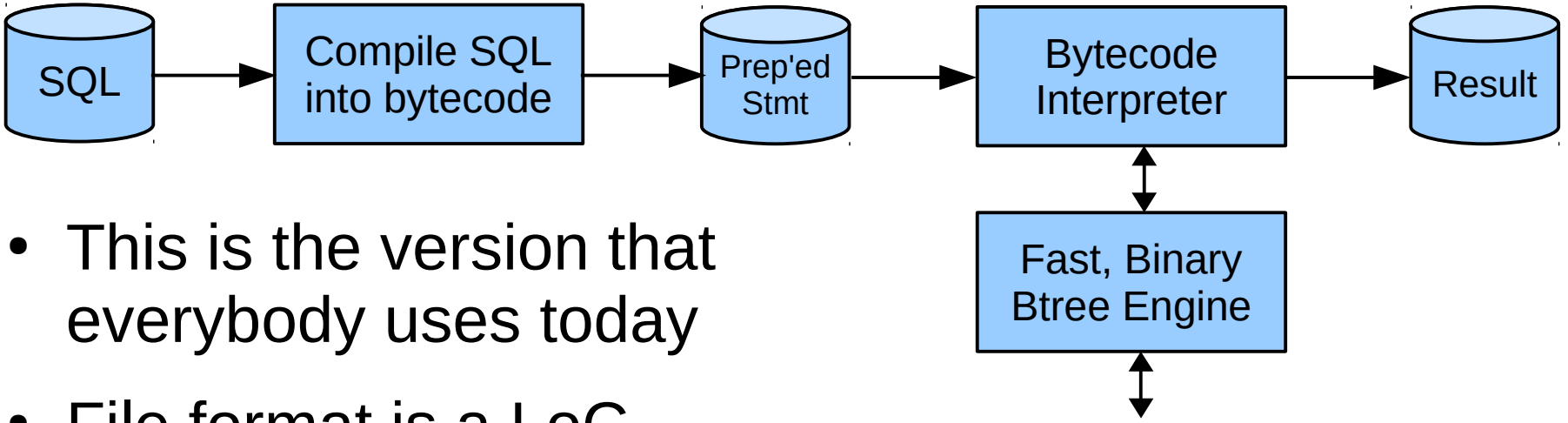






- Efficient range queries
- Single-file database
- String keys and data → hard to store binary data
- Public domain (2001-09-16)
- 2001-08-19 to 2004-04-23 with maintenance through 2007-01-08

Version 3



- This is the version that everybody uses today
- File format is a LoC recommendation for long-term archival storage
- 2004-04-23 to 2050-05-29

A Few Other Milestones

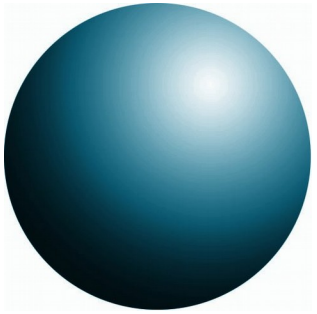
- 2006-08-12 → Virtual tables
- 2007-03-27 → Amalgamation
- 2008-01-09 → Register-based VM
- 2009-07-27 → 100% MC/DC testing
- 2010-05-03 → Write-ahead log
- 2013-06-26 → Next-Gen query planner
- 2014-02-03 → Common table expressions
- 2015-09-04 → Indexes on expressions
- 2016-09-07 → Row values
- 2018-09-15 → Window functions

SQLite Implementation Overview

- Row-store
- Variable-length records
- Forest of B-trees - all in a single disk file
  - One B-tree for each table and each index
  - Table key: PRIMARY KEY or ROWID
  - Index key: indexed columns + table key
- Transaction control using separate rollback-journal or write-ahead log files
  - Atomic writes without a journal when supported by the filesystem (ex: F2FS)

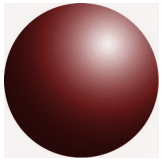
Two Key Objects

sqlite3 – *Database Connection*



- An open connection to a database
- Connects to one or more databases
- May contain multiple prepared statements

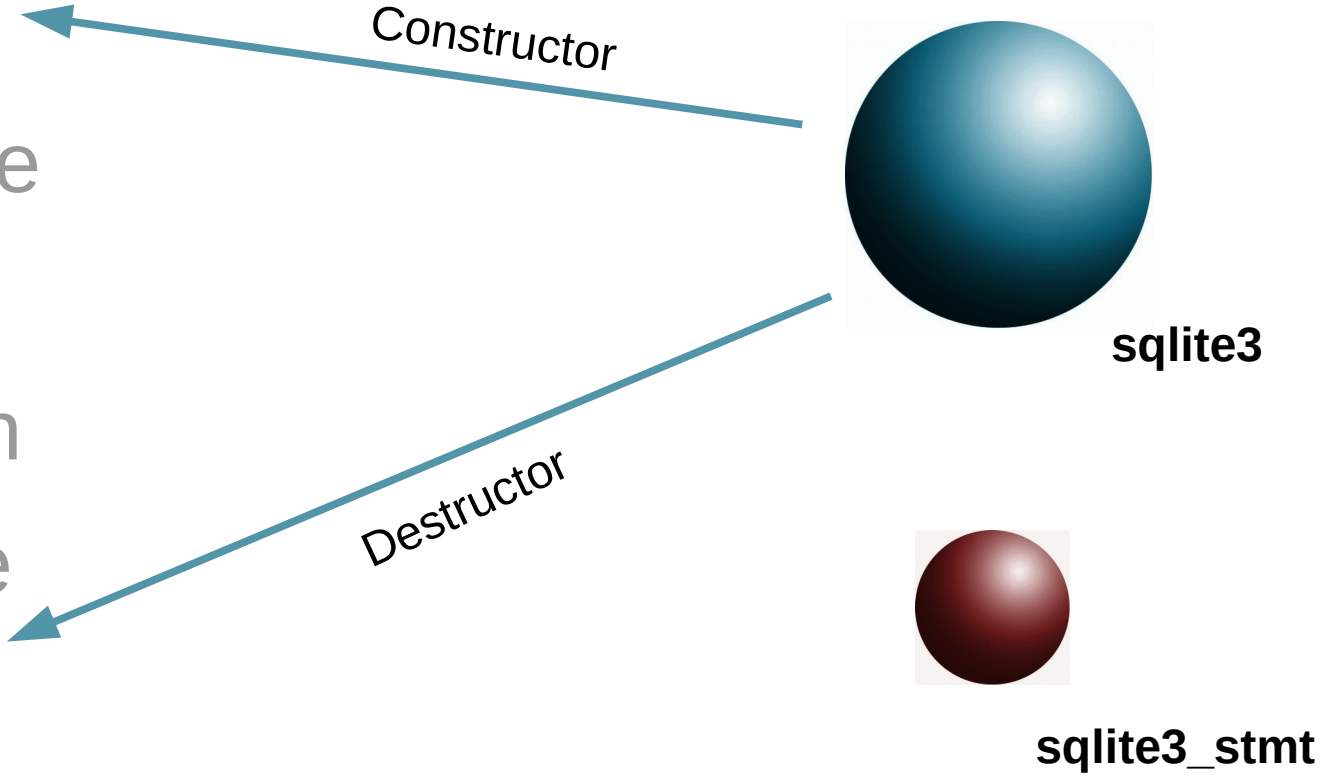
sqlite3_stmt – *Prepared Statement*



- A single SQL statement
- Associated with a single connection

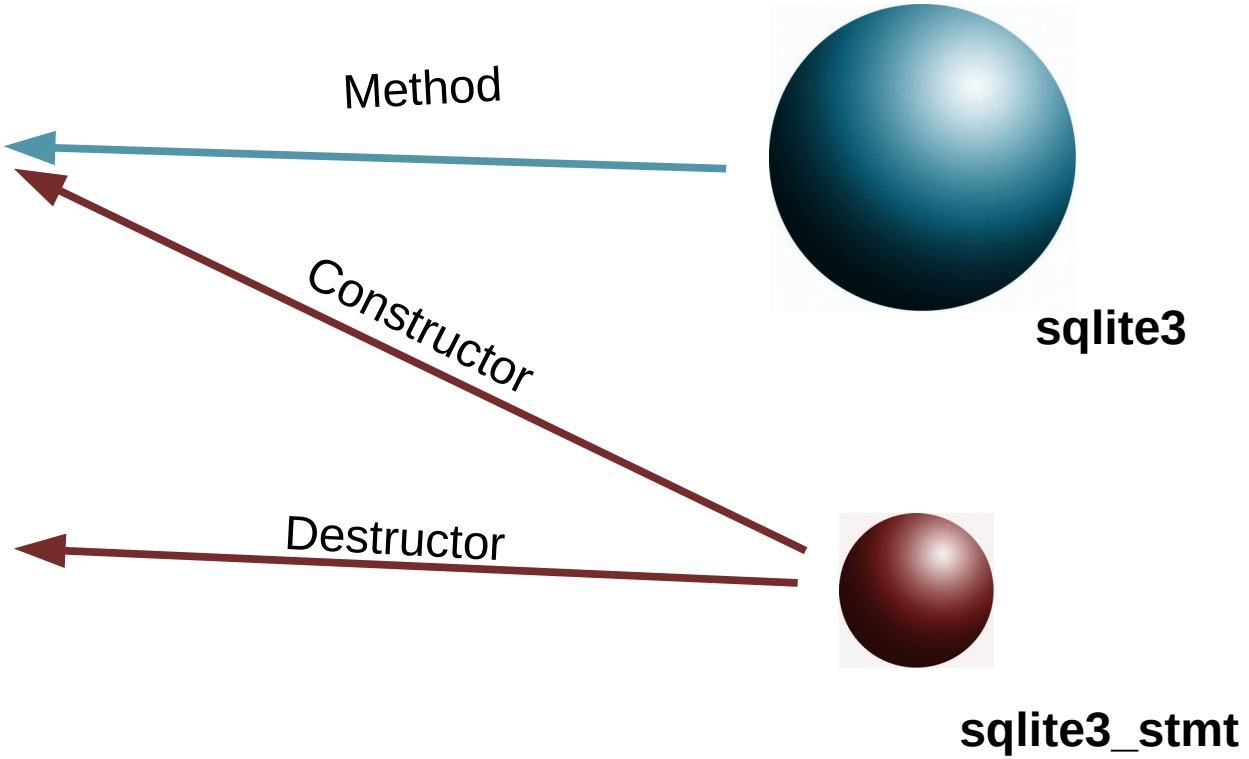
Key Methods

- sqlite3_open
- sqlite3_prepare
- sqlite3_step
- sqlite3_column
- sqlite3_finalize
- sqlite3_close



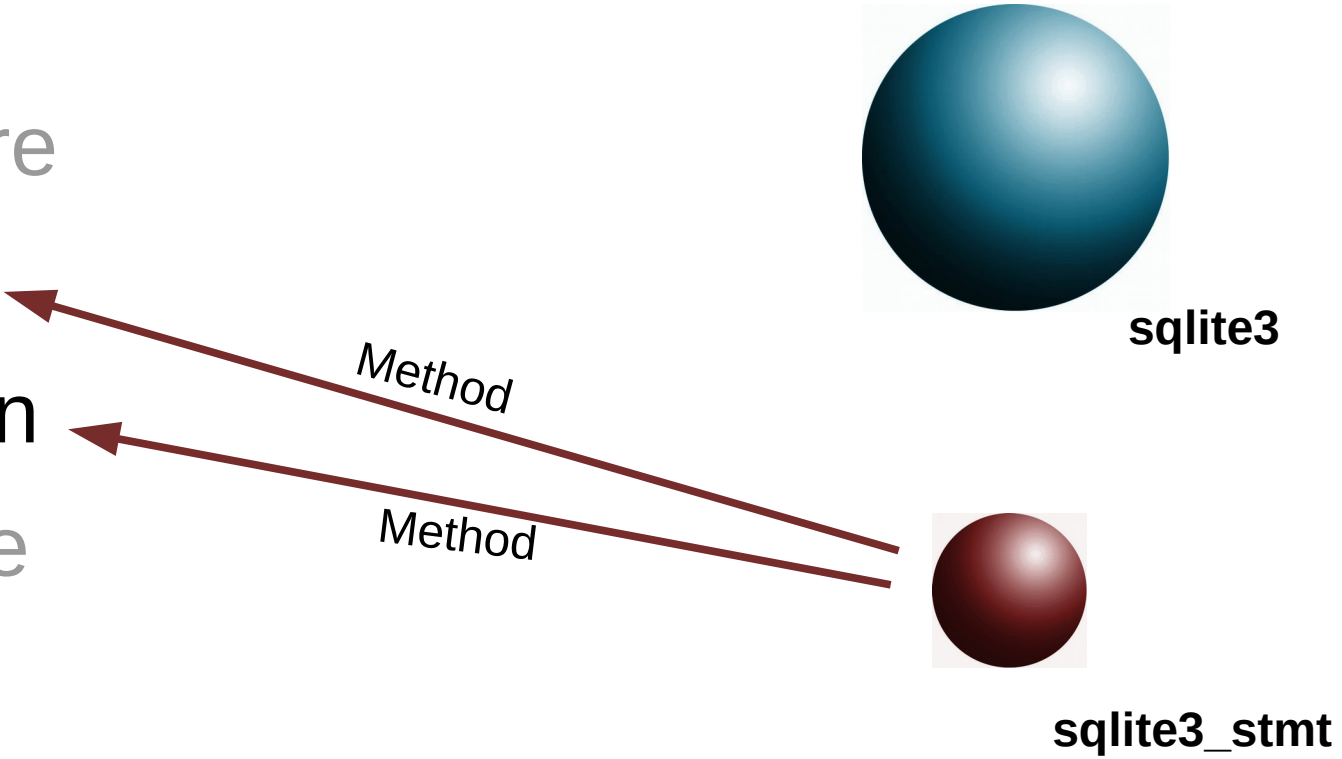
Key Methods

- sqlite3_open
- sqlite3_prepare
- sqlite3_step
- sqlite3_column
- sqlite3_finalize
- sqlite3_close



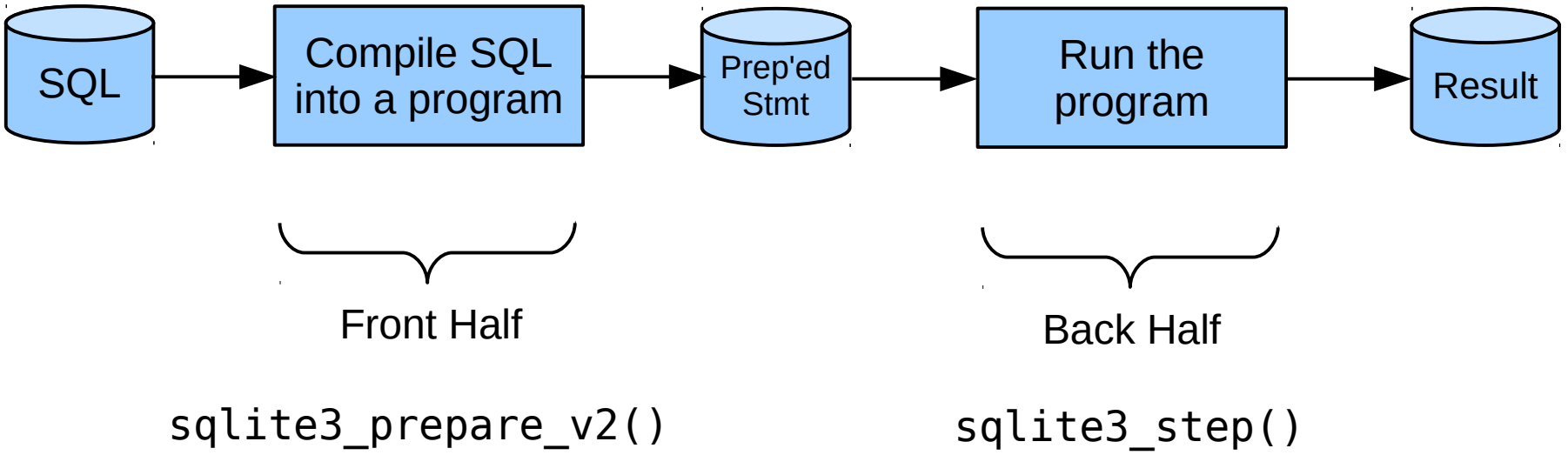
Key Methods

- sqlite3_open
- sqlite3_prepare
- sqlite3_step
- sqlite3_column
- sqlite3_finalize
- sqlite3_close



Ins & Outs

- SQLite consists of...
  - Compiler to translate SQL into byte code
  - Virtual Machine to evaluate the byte code



Hacking SQLite

./configure --enable-debug && make

Enable hacker features

- PRAGMA vdbe_trace;
- PRAGMA vdbe_debug;
- PRAGMA parser_trace;
- PRAGMA vdbe_addoptrace;

test_addop_breakpoint

- .selecttrace 0xffff
- .wheretrace 0xfff
- .eqp on|full|trace
- .breakpoint

test_addop_breakpoint

Hacking SQLite

- sqlite3TreeViewExpr(0, pExpr, 0)
- sqlite3TreeViewExprList(0, pExprList, 0, 0)
- sqlite3TreeViewSelect(0, pSelect, 0)



Invoke from a debugger to see the complete content of one of these parse-tree objects.

Silly SQLite Trick #1

```
$ sqlite3 database.db
sqlite> .excel
sqlite> SELECT * FROM sometable;
sqlite> .q
```

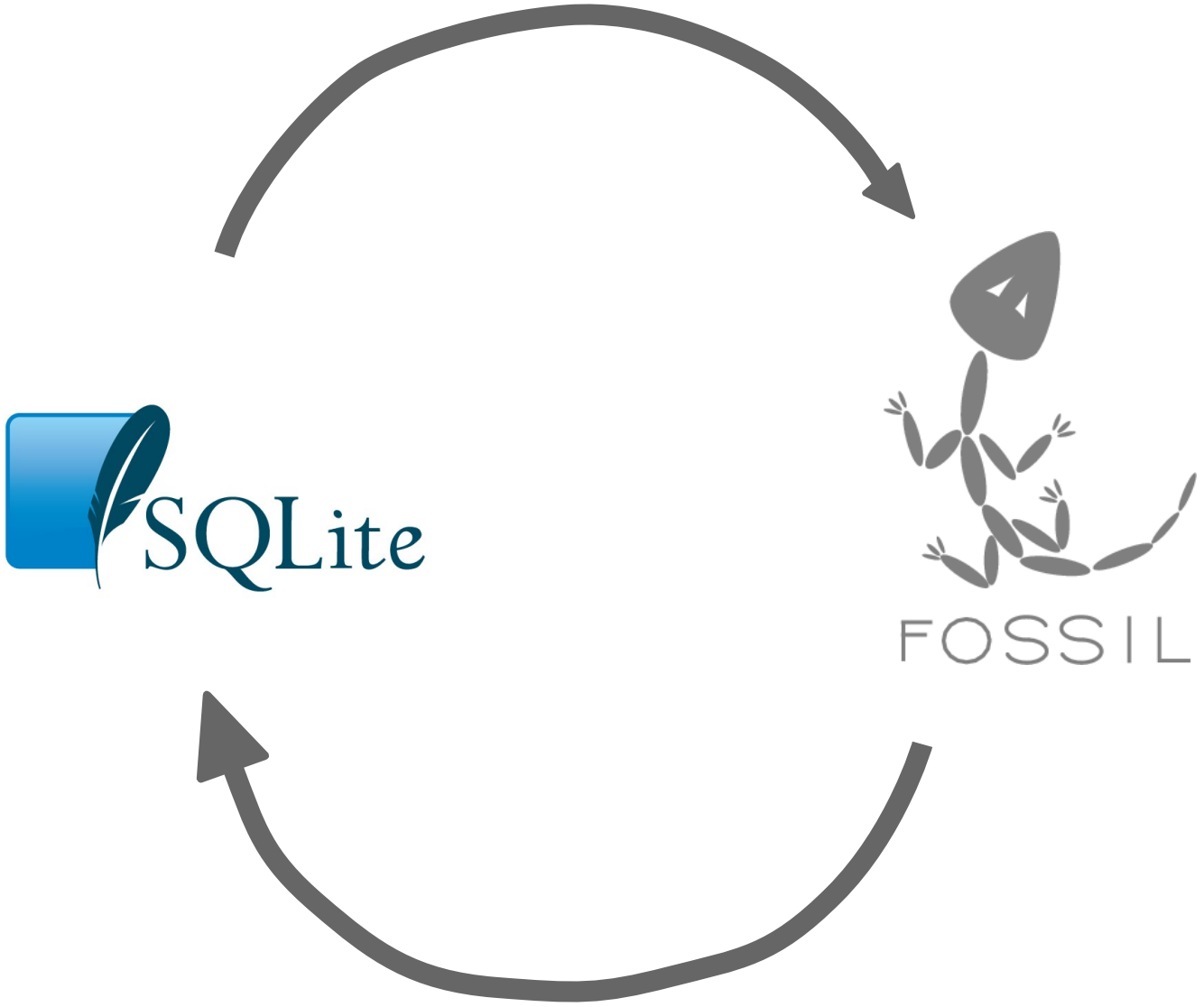
See also: ".once" and ".once -e"

Silly SQLite Trick #2

sqldiff *DB1 DB2 >diff.sql*

- Tries to output the minimum SQL needed to transform DB1 into DB2
- No guarantee that the SQL will be minimal
- “sqldiff --help” for further information

Custom Version Control



Unstoppable Ideas Behind SQL

- Transactions
  - *The system moves atomically from one consistent state to the next.*
- Data Abstraction
  - *“Representation is the essence of computer programming.”*
- Declarative Language
  - *Push the semantics of the query down into the storage engine and let it figure out what to do.*

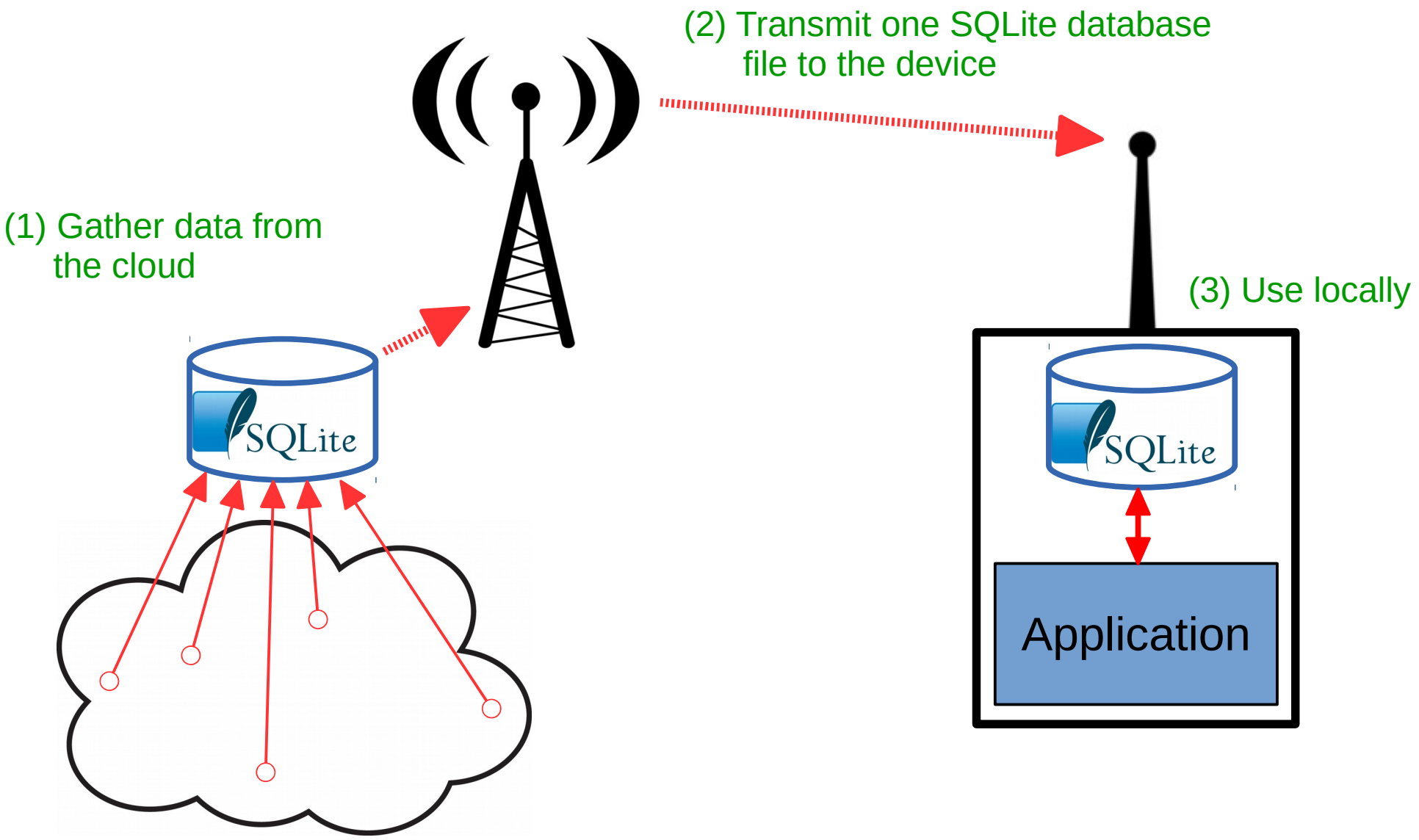
```
INSERT INTO users VALUES('alex','Alexander Fogg',29,3341);

DELETE FROM users WHERE uid='alex';

UPDATE users SET officeId=4217 WHERE uid='alex';
```

```
SELECT
  blob.rid,
  uuid,
  datetime(event.mtime,'localtime') AS timestamp,
  coalesce(ecomment, comment),
  coalesce(euser, user),
  (SELECT count(*) FROM plink WHERE pid=blob.rid AND isprim=1),
  (SELECT count(*) FROM plink WHERE cid=blob.rid),
  NOT EXISTS(SELECT 1 FROM plink
              WHERE pid=blob.rid
               AND coalesce((SELECT value FROM tagxref
                             WHERE tagid=8 AND rid=plink.pid), 'trunk')
                 = coalesce((SELECT value FROM tagxref
                             WHERE tagid=8 AND rid=plink.cid), 'trunk')),
  bgcolor,
  event.type,
  (SELECT group_concat(substr(tagname,5), ', ') FROM tag, tagxref
    WHERE tagname GLOB 'sym-*' AND tag.tagid=tagxref.tagid
      AND tagxref.rid=blob.rid AND tagxref.tagtype>0),
  tagid,
  brief
 FROM event JOIN blob
WHERE blob.rid=event.objid
 ORDER BY event.mtime DESC LIMIT 20;
```

SQLite = Data Container



SQLite Archiver

```
CREATE TABLE sqlar(
  name TEXT PRIMARY KEY,  -- name of the file
  mode INT,               -- access permissions
  mtime INT,              -- last modification time
  sz INT,                 -- original file size
  data BLOB               -- compressed content
);
```

- https://sqlite.org/sqlar
- Transactional
- Concurrent & random access
- File size similar to ZIP

SQLAR is smaller than ODP!

```
-rw-r--r--  1 drh  staff  10514994 Jun  8 14:32 self2014.odp
-rw-r--r--  1 drh  staff  10464256 Jun  8 14:37 self2014.sqlar
-rw-r--r--  1 drh  staff  10416644 Jun  8 14:40 zip.odp
```

SQLAR is only 0.46% larger than ZIP

SQLite versus ZIP

Yes	Container for files	Yes
Yes	A trillion instances in the wild	Yes
Yes	Compact	Yes
Yes	Well-defined open format	Yes
Yes	Container for small objects	No
Yes	Cross-platform small objects	No
Yes	Transactions	No
Yes	Query language	No
Yes	Schema	No

SQLite versus ZIP

Yes	Container for files	Yes
Yes	A trillion instances in the wild	Yes
Yes	Compact	Yes
Yes	Well-defined open format	Yes
Yes	Container for small objects	No
Yes	Cross-platform small objects	No
Yes	Transactions	No
Yes	Query language	No
Yes	Schema	No

The Three Unstoppable Ideas

The SQLite CLI understands both ZIP and SQL archives

Create:

```
sqlite3 one.sqlar -Acv *.txt
sqlite3 one.zip -Acv *.txt
```

List:

```
sqlite3 one.sqlar -Atv
sqlite3 one.zip -Atv
```

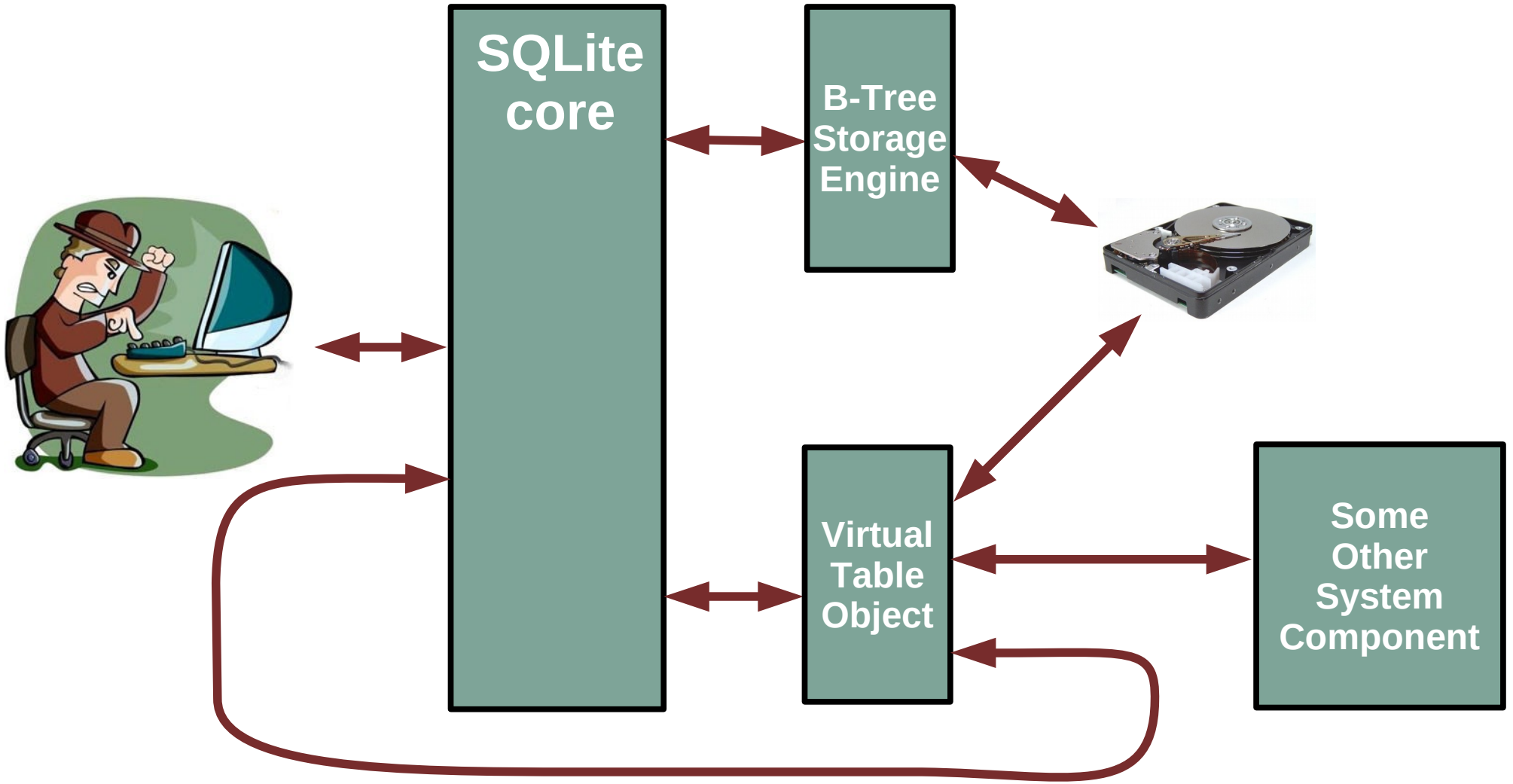
Update:

```
sqlite3 one.sqlar -Auv *.txt
sqlite3 one.zip -Auv *.txt
```

Extract:

```
sqlite3 one.sqlar -Axv
sqlite3 one.zip -Axv
```

https://sqlite.org/cli.html#sqlar


SQLite core
B-Tree Storage Engine
Virtual Table Object
Some Other System Component

```
SELECT
    pid, system_time, user_time, name
FROM
    processes, users
WHERE
    users.username = 'drh' AND
    users.uid = processes.uid
ORDER BY
    system_time + user_time DESC
LIMIT 10;
```



pid	system_time	user_time	name
1437	5019	54414	ibus-daemon
1472	4567	13469	ibus-x11
1831	4134	5422	pulseaudio
1825	1353	7198	gnome-terminal
1530	1142	6353	ibus-engine-sim
1469	600	4928	ibus-ui-gtk3
620	352	4596	soffice.bin
9585	333	3092	firefox
1538	580	2409	xfdesktop
1682	1434	922	xscreensaver

What If....

OpenDocument was an SQLite database
Instead of a ZIP archive of XML files...

- Fast and low-I/O save of small changes
- Fast startup
- Reduced memory usage (no need to hold the entire presentation in memory at once)
- No need for “recovery” after a crash
- No need to “file save”
- Undo across sessions
- Large searchable database of slides

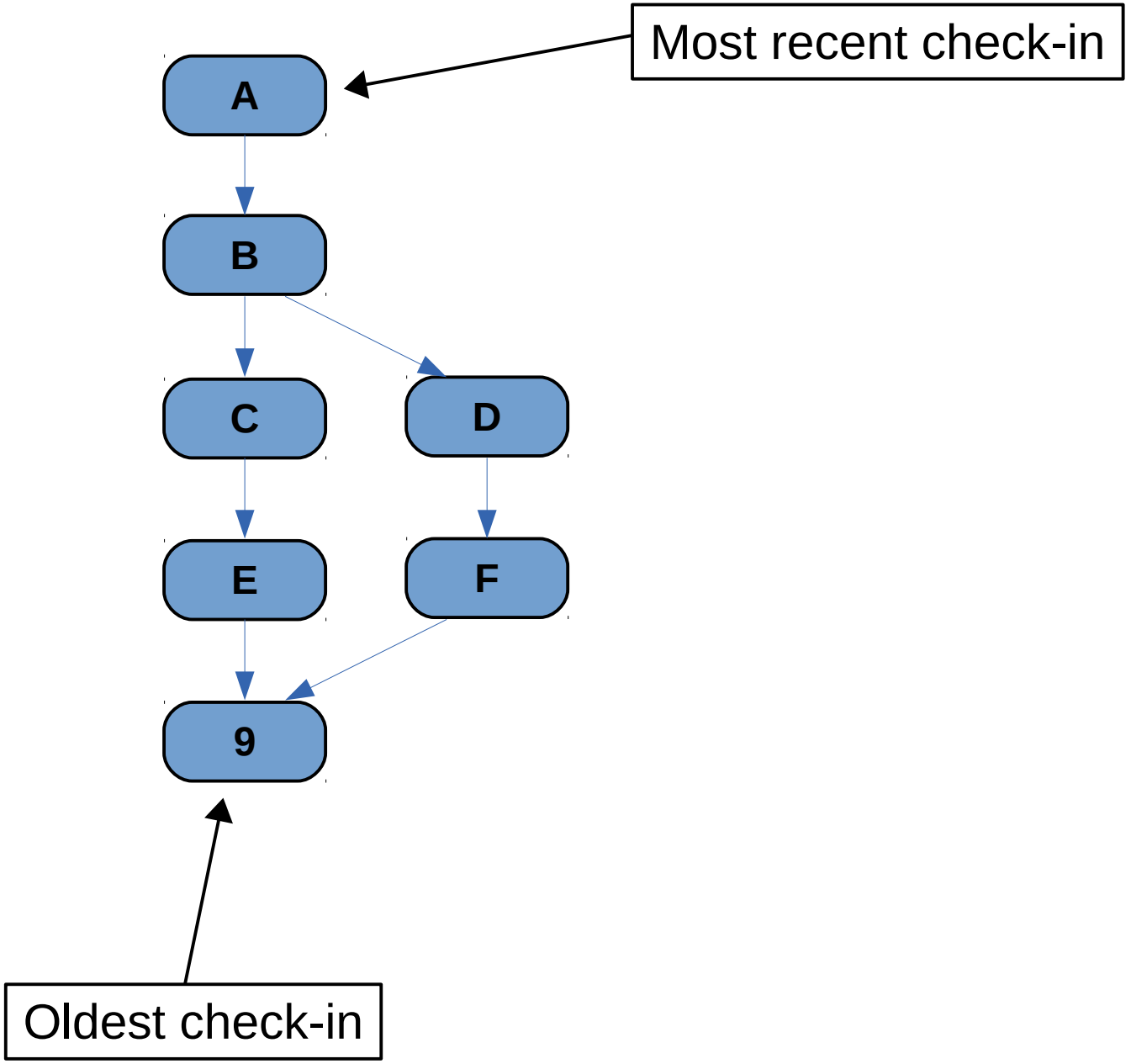
Silly SQLite Trick #3

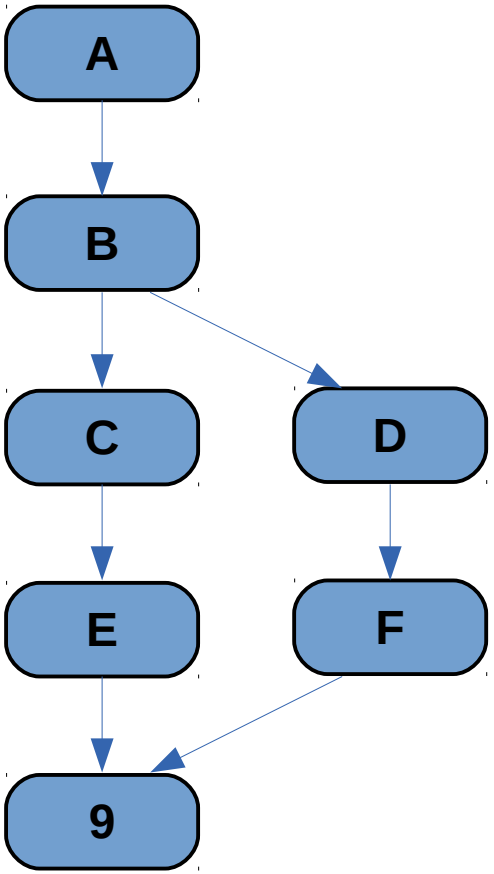
```
bash$ sqlite3 sqlite-vienna-20190625.odp
SQLite version 3.29.0 2019-05-27 11:21:43
Enter ".help" for usage hints.
sqlite> .schema
CREATE VIRTUAL TABLE zip USING zipfile('sqlite-vienna-20190625.odp')
/* zip(name,mode,mtime,sz,rawdata,data,method) */;
sqlite> SELECT name, sz FROM zip;
  ...
Pictures/10000000000027100000122D4A26DD782AACB4F.jpg|80726
Pictures/10000000000027100001221CDFE0703F589F84.jpg|80726
styles.xml|96239
settings.xml|9998
META-INF/manifest.xml|6134
content.xml|453133
sqlite> UPDATE zip SET data=replace(data,'vienna','anneiv')
   ...> WHERE name='content.xml';
sqlite> .quit
```

What if...

Git stored content in an SQLite database instead of a bespoke “packfile” key/value store?

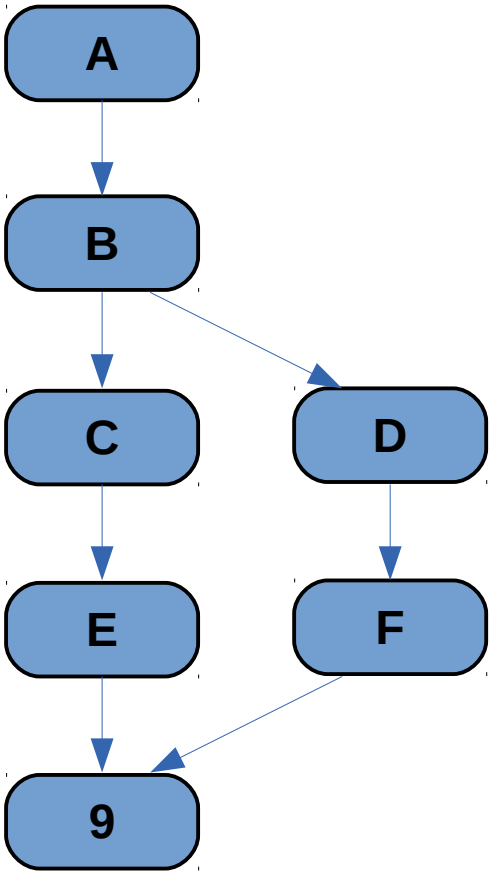
- Ability to find descendents of check-ins
- Advanced queries for a richer user interface
- Proof against crashes
- Wiki and Tickets
- Concurrent access
- Coding errors less likely to corrupt repository
- Single-file repository


Most recent check-in
A
B
C
D
E
F
9
Oldest check-in




```
CREATE TABLE lineage(
    parent HASH,
    child HASH,
    rank INT
);
```

(B,A,0)
(C,B,0)
(D,B,1)
(E,C,0)
(F,D,0)
(9,E,0)
(9,F,0)




```
CREATE TABLE lineage(
    parent HASH,
    child HASH,
    rank INT
);
```

(B,A,0)
(C,B,0)
(D,B,1)
(E,C,0)
(F,D,0)
(9,E,0)
(9,F,0)

```
SELECT parent FROM lineage WHERE child='E';
SELECT child FROM lineage WHERE parent='E';
```

Computing $n most recent decendents of check-in $root

```
CREATE TABLE lineage(
    parent HASH,        -- parent check-in
    child HASH,         -- child check-in
    rank INT,           -- 0 for primary parent
    mtime DATETIME      -- time of child check-in
);

WITH RECURSIVE
  dx(id,mtime) AS (
    SELECT $root, mtime FROM lineage WHERE child=$root
    UNION
    SELECT lineage.child, lineage.mtime FROM dx, lineage
      WHERE lineage.parent=dx.id"
      ORDER BY 2
  )
SELECT id FROM dx LIMIT $n
```

Some Things Git Does Not Compute Because Of Its Use of Key/Value

- Descendents of a check-in
- Check-ins during a particular time period
- Change history for a single file
- Locate forks or detached heads
- Check-ins containing a particular version of a file
- Check-ins that touch a certain file
- First N check-ins
- List of files sorted by number of changes
- All check-ins associated with a particular user
- Check-ins before/after/around a point in time
- ... and so forth

Key/Value Limitation Leaks Into Other Aspects Of The System

- Tags must be unique and have name restrictions
- No support for tickets, wiki, documentation
- Server infrastructure is a (large) external addition (ex: GitLab)
- Large codebase dedicated to processing the bespoke “packfile” database format
- Slow response to the SHAttered attack
- Many useful reports omitted because they are hard to implement

The SQLite Business Model

- “Open-Source” but not “Open-Contribution”
  - The code is completely free, but some of the test cases are proprietary
  - The name “SQLite” is a registered trademark
  - Is it really open-source, then?
- Income sources:
  - Proprietary extensions
  - Technical support
  - Warranty of Title
- Low overhead - no VCs - sustainable

The Future Of SQLite

- Support through 2050-05-29 (version 3.205.0)
- 5 or 6 releases per year
- Keep it robust, backwards-compatible, free, and organic
  - Low drama
  - “It just works”
- Constantly improving the query planner AI
- New extensions and virtual tables



Small, Fast, Reliable
Choose Any Three