A Database for the Edge of the Network

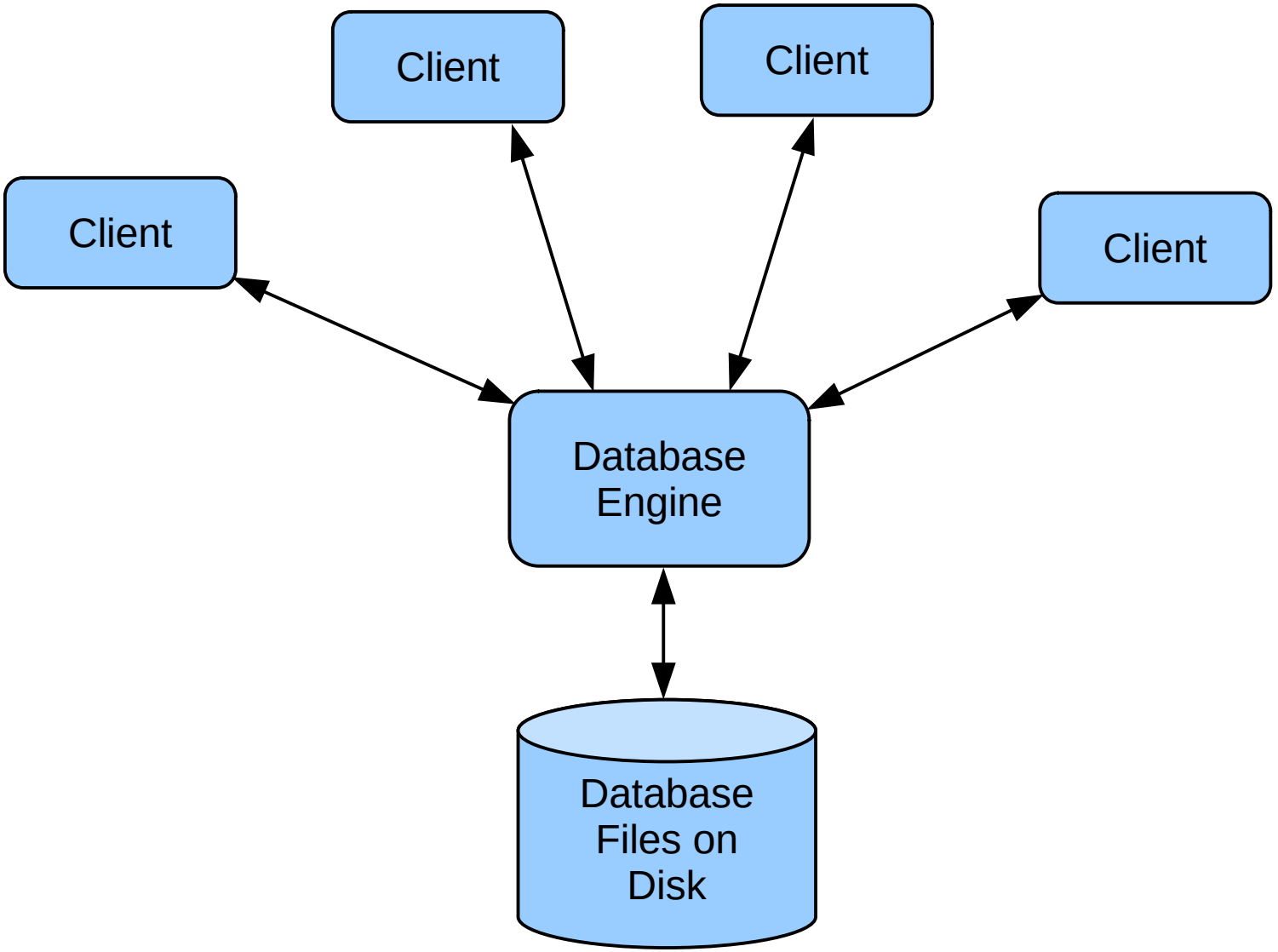
https://www.sqlite.org/talks/cmu-20150917.odp

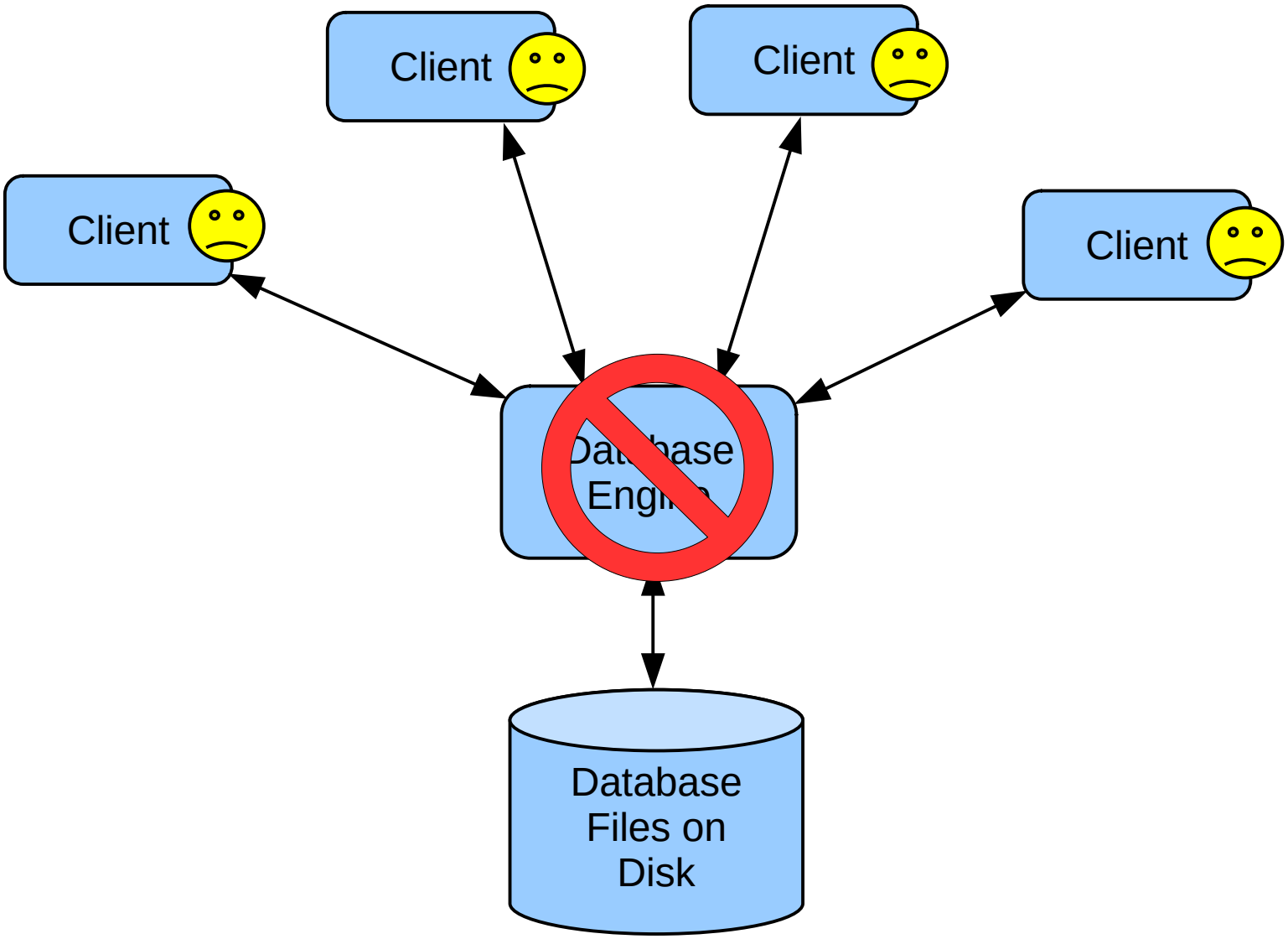
SQLite in a nutshell

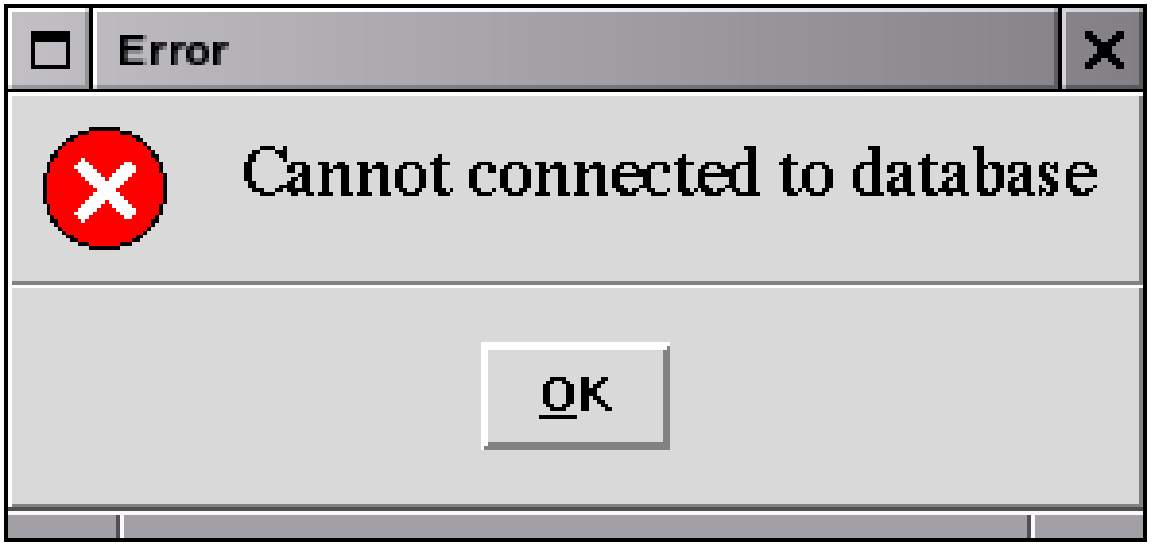
- An in-process library, not a system, not a server
- One file of ANSI-C code, approximately 500KB compiled
- Self-contained, low-dependency
  - memcmp, memcpy, memmove, memset, strcmp, strlen, strncmp
- A complete database stored as a single ordinary disk file
- Full-featured SQL
- Power-safe, serializable transactions
- Multi-thread and multi-process safe
- Simple API, zero-configuration, “just works”
- Source code is in the public domain

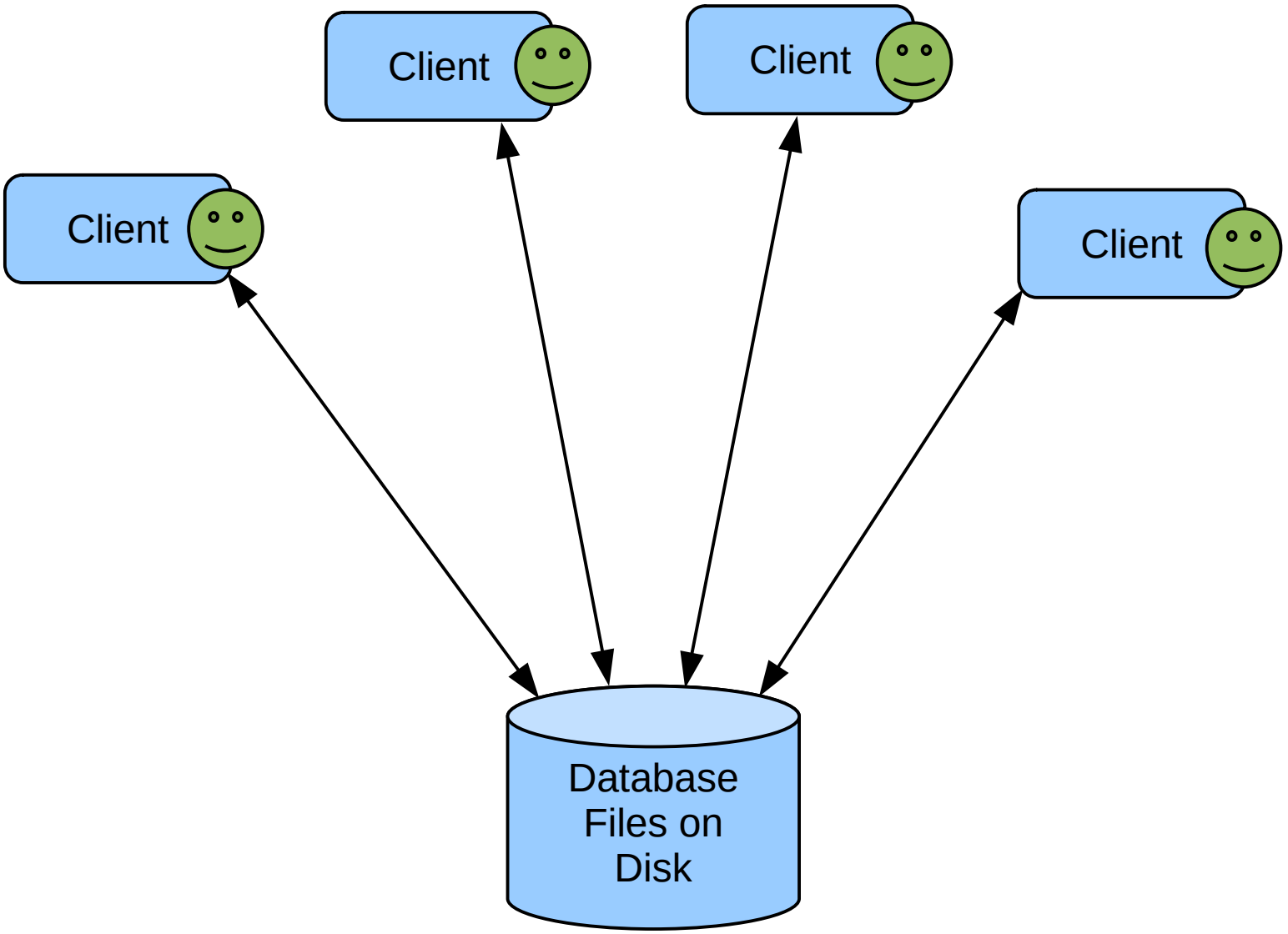
SQLite limits

- 1+ concurrent writer and N concurrent readers per database
- 1 gigabyte strings, BLOBs, content per row
- 140 terabytes per database
- 64-way joins
- 2000 columns per table or index
- No arbitrary limit on the number of tables or indexes or rows in a table


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

use cases

- Database for embedded devices and the internet of things
- Application file format - replacement for an *ad hoc* pile-of-files
- *Lingua Franca* for a federation of programs
- Local cache of enterprise data - disconnected operation
- Interchange format

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?

Otherwise

No!

fopen()

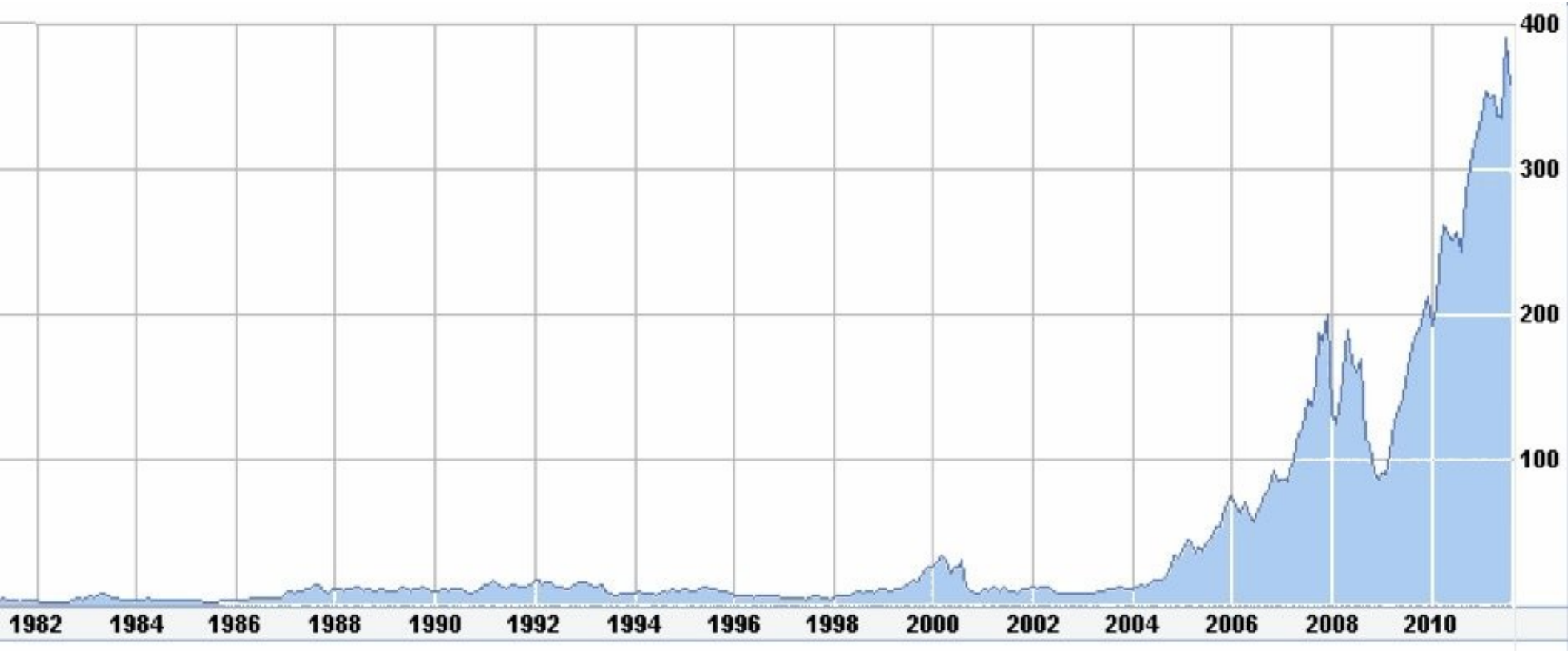
does not compete with

competes with **fopen()**

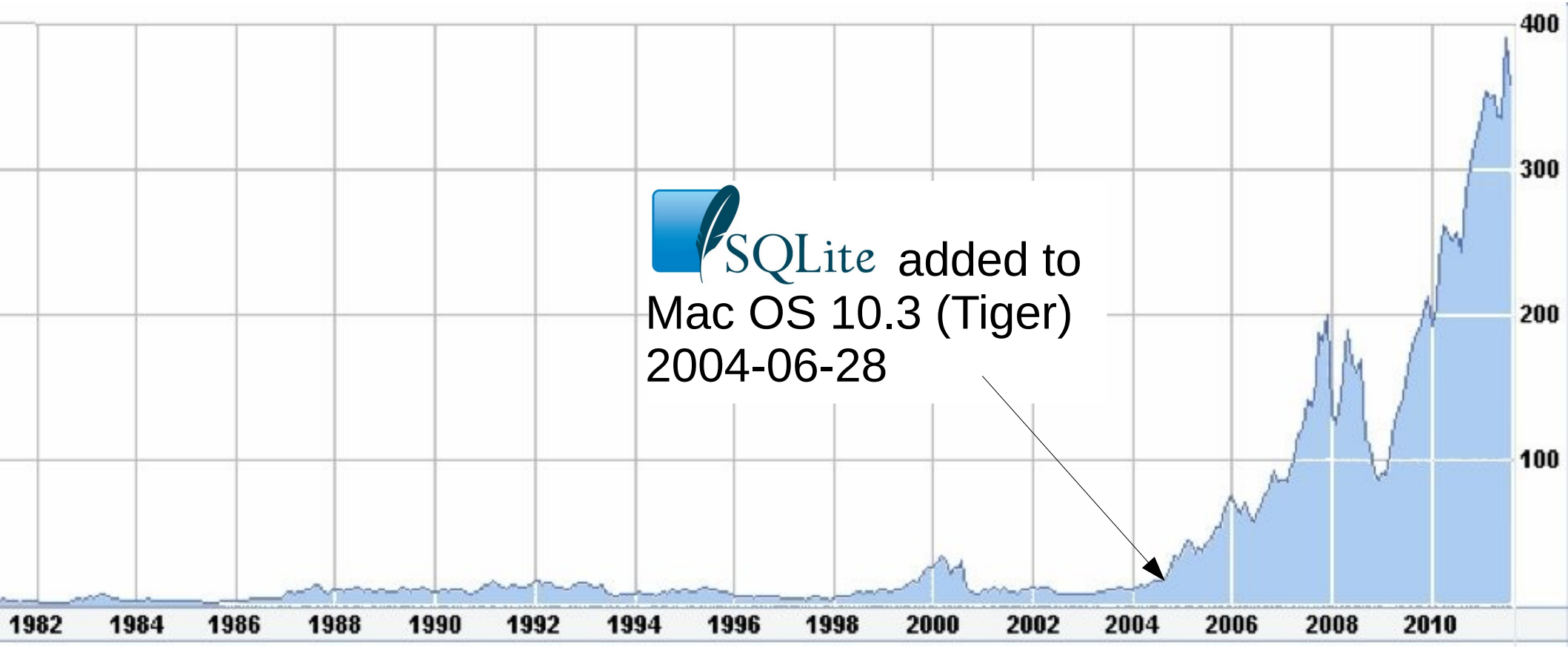
Found In...

- Every Android phone and device
- Every iPhone and iOS device
- Every Mac
- Every Windows10 machine
- Every Firefox, Chrome, and Safari browser
- Every installation of iTunes, Skype, Dropbox, QuickBooks, TurboTax
- Python and PHP
- Most TV sets and set-top cable boxes
- Automotive multimedia systems
- Countless millions of other applications....

Apple, Inc


400
300
200
100
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010

Apple, Inc


SQLite added to
Mac OS 10.3 (Tiger)
2004-06-28
400
300
200
100
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010

Implementation Overview

Where To Find Source Code

https://www.sqlite.org/download.html

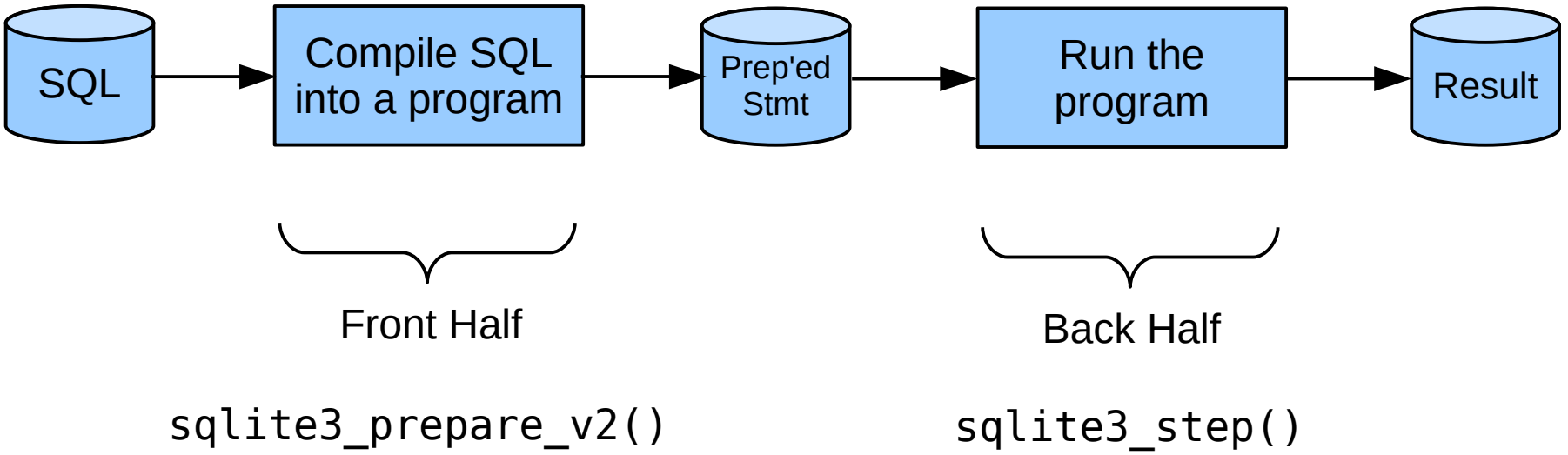
- To download a tarball

https://www.sqlite.org/src/tree?ci=trunk

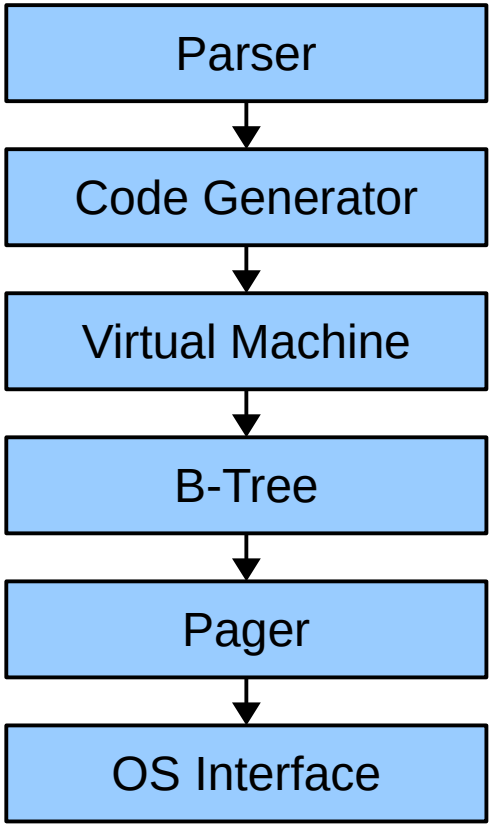
- To view sources on-line

Ins & Outs

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



The SQLite Stack



The SQLite Stack

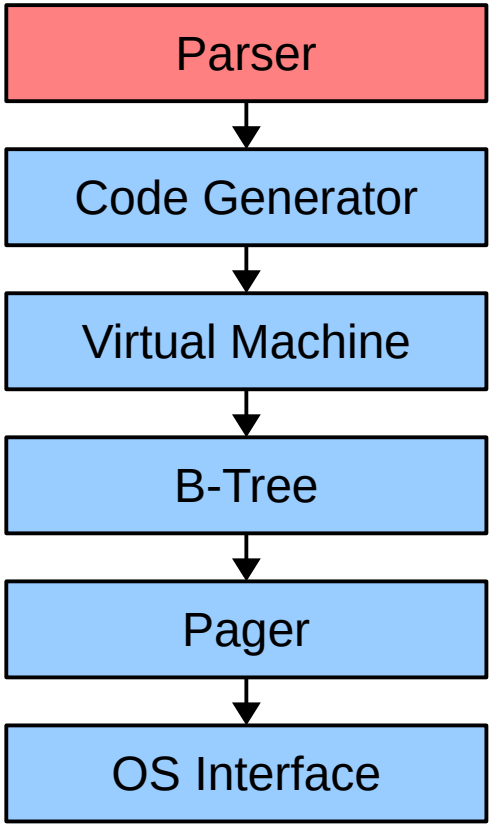
- Hand-written tokenizer
- Push-down automaton parser generated by “Lemon”
- Reentrant & threadsafe
- Output: Abstract Syntax Tree (AST)

tokenize.c

parse.y

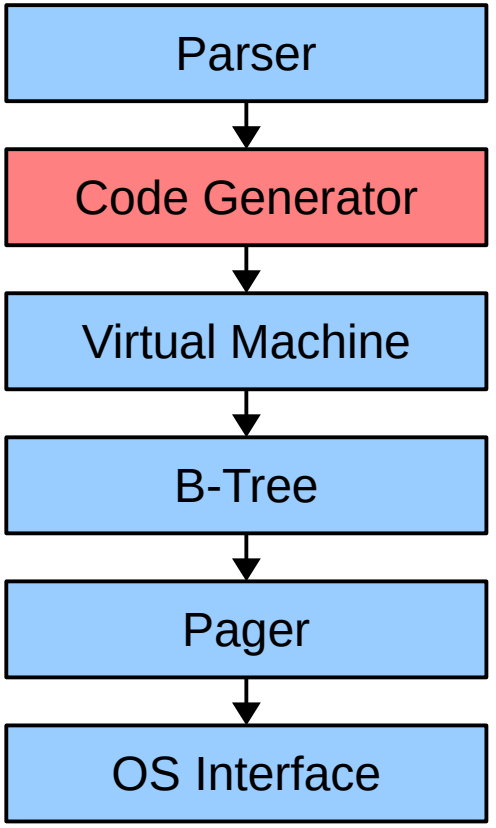
../tool/lemon.c
../doc/lemon.html

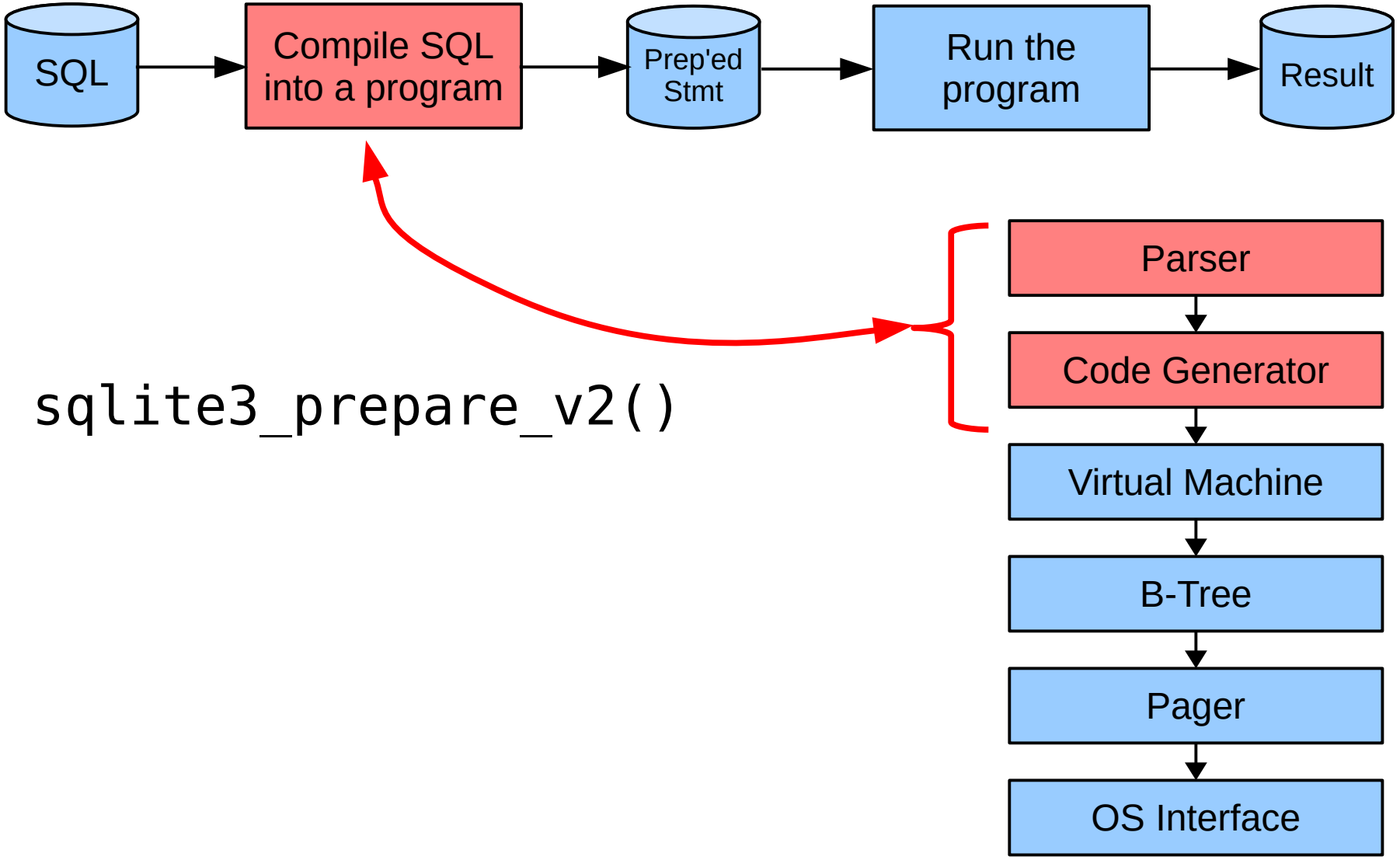
sqliteInt.h

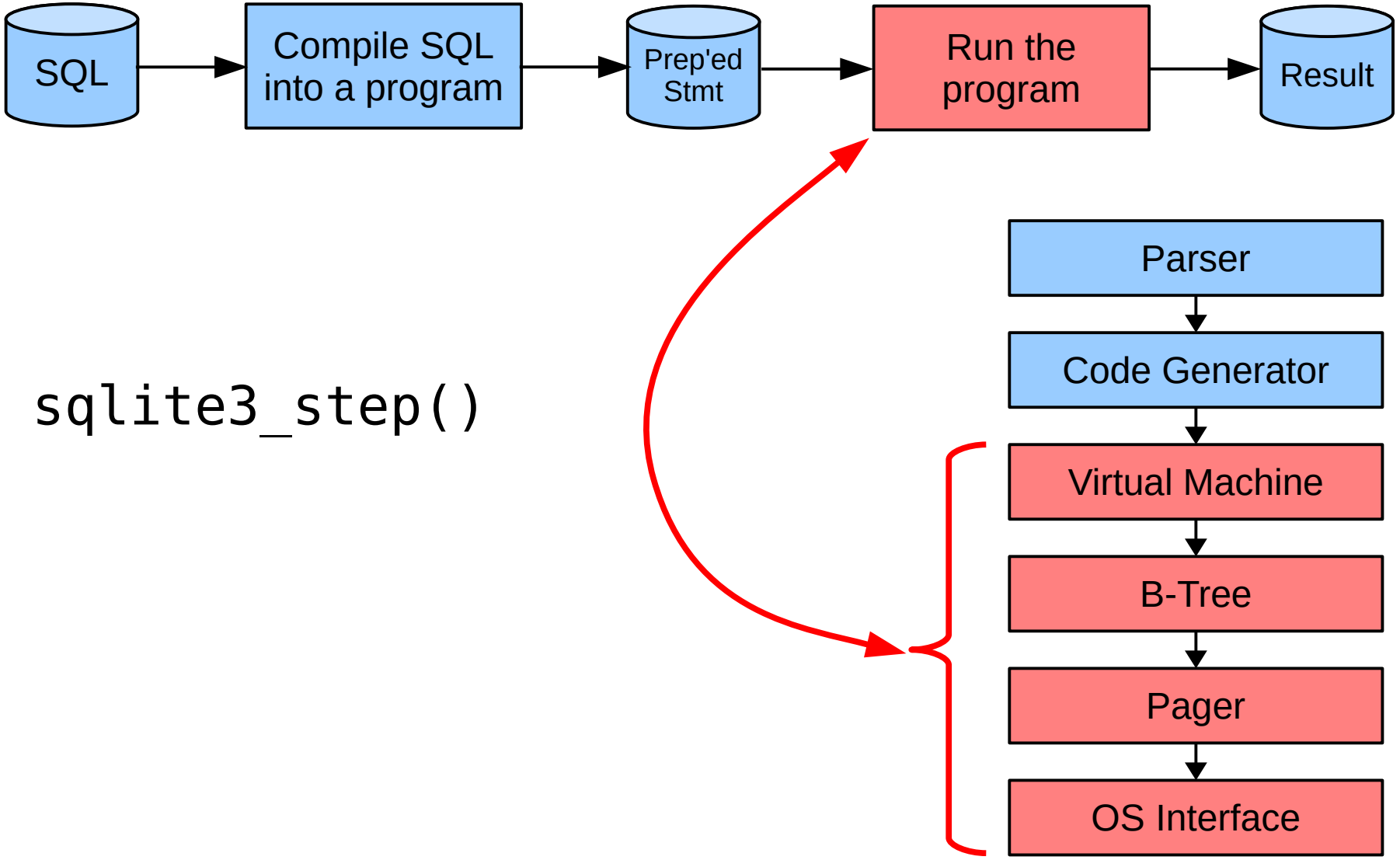


The SQLite Stack

- Semantic analysis
- AST transformations
- Query planning
- Byte-code generation
- Output: “prepared statement” (byte code)



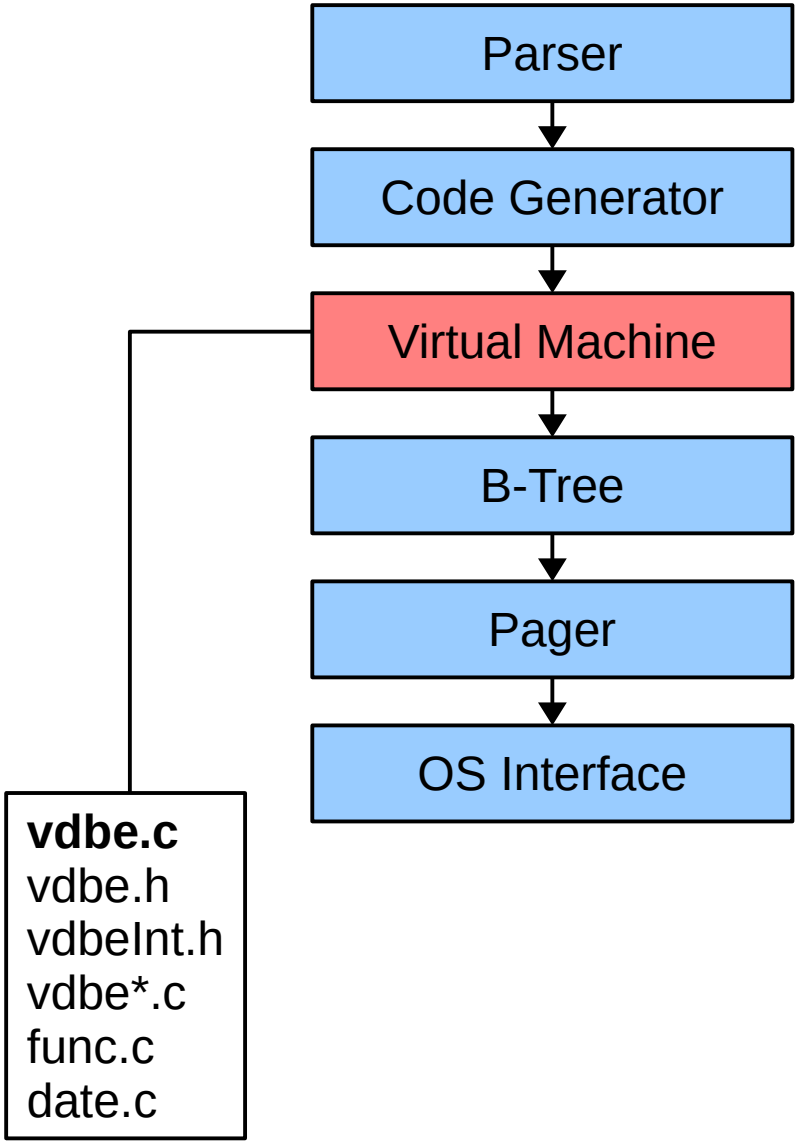

SQL
Compile SQL into a program
Prep'ed Stmt
Run the program
Result
sqlite3_prepare_v2()
Parser
Code Generator
Virtual Machine
B-Tree
Pager
OS Interface



`sqlite3_step()`

The SQLite Stack

- Byte code interpreter
- 3-address register machine
- Big `switch` statement inside a `for` loop.
- Built-in SQL functions



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.

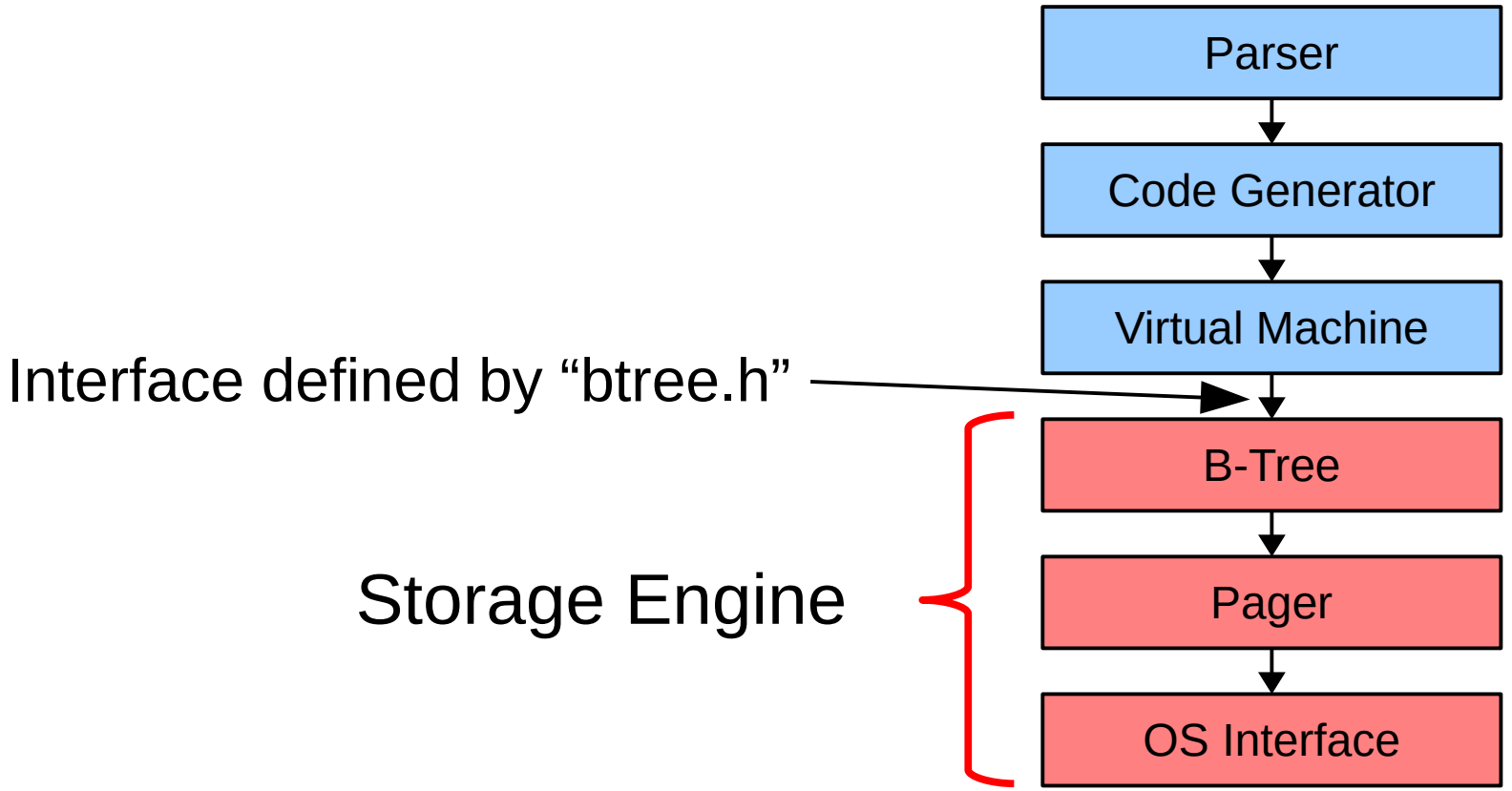
Viewing Bytecode

- Download & unpack source tarball
- ./configure --disable-shared
- OPTS=-DSQLITE_ENABLE_EXPLAIN_COMMENTS make
- ./sqlite3 *database-file*
  - .explain
  - explain select * from sqlite_master;

Optional. Avoids crazy autoconf shared-library complication

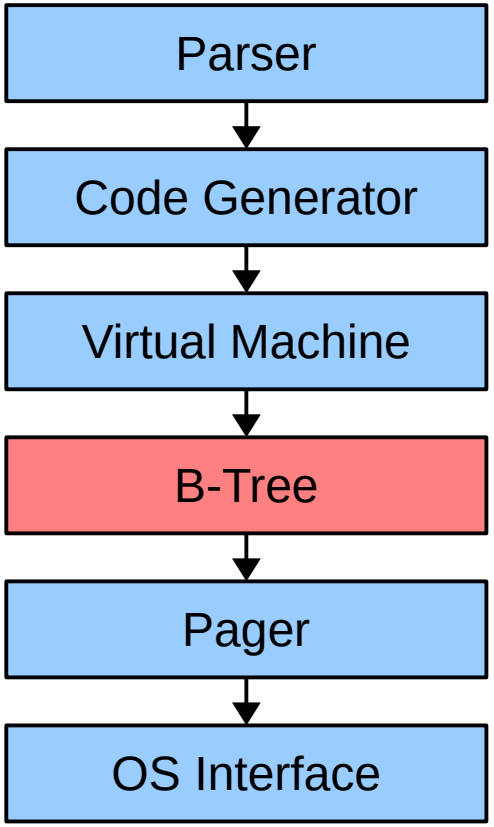
Sets of the command-line shell to format EXPLAIN output for easy reading.
See ***https://www.sqlite.org/cli.html#explain***

The SQLite Stack



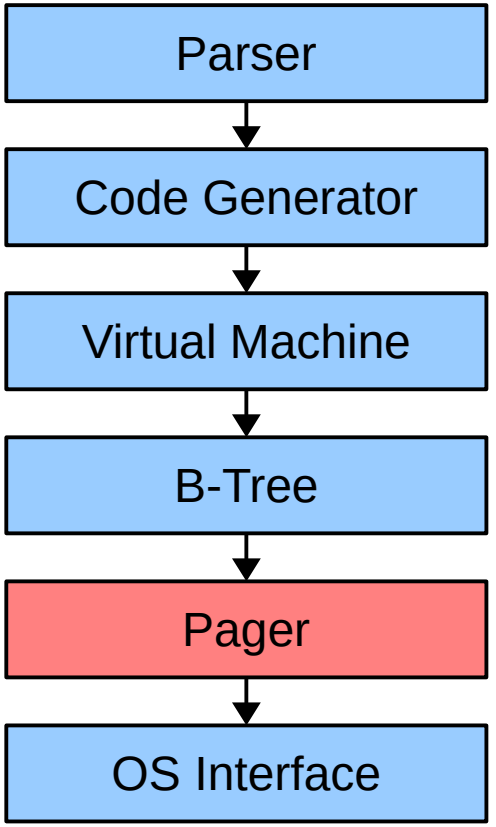
The SQLite Stack

- B-tree and B+trees
- Multiple B-trees per disk file
- Variable-length records
- Free page tracking & reuse
- Access via cursor
- Concurrent read/write of the same table using separate cursors



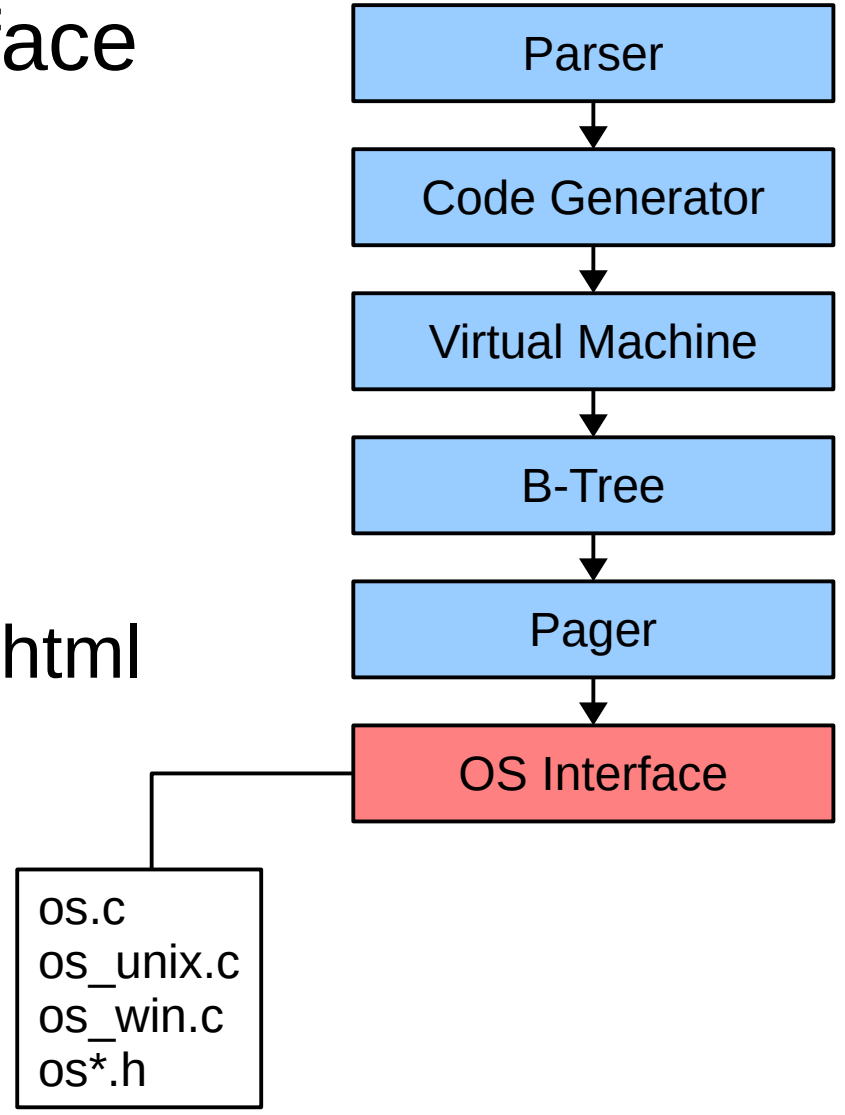
The SQLite Stack

- Atomic commit and rollback
- Uniform size pages numbered from 1
- 512 to 65536 bytes per page
- No interpretation of page content
- Cache
- Concurrency control



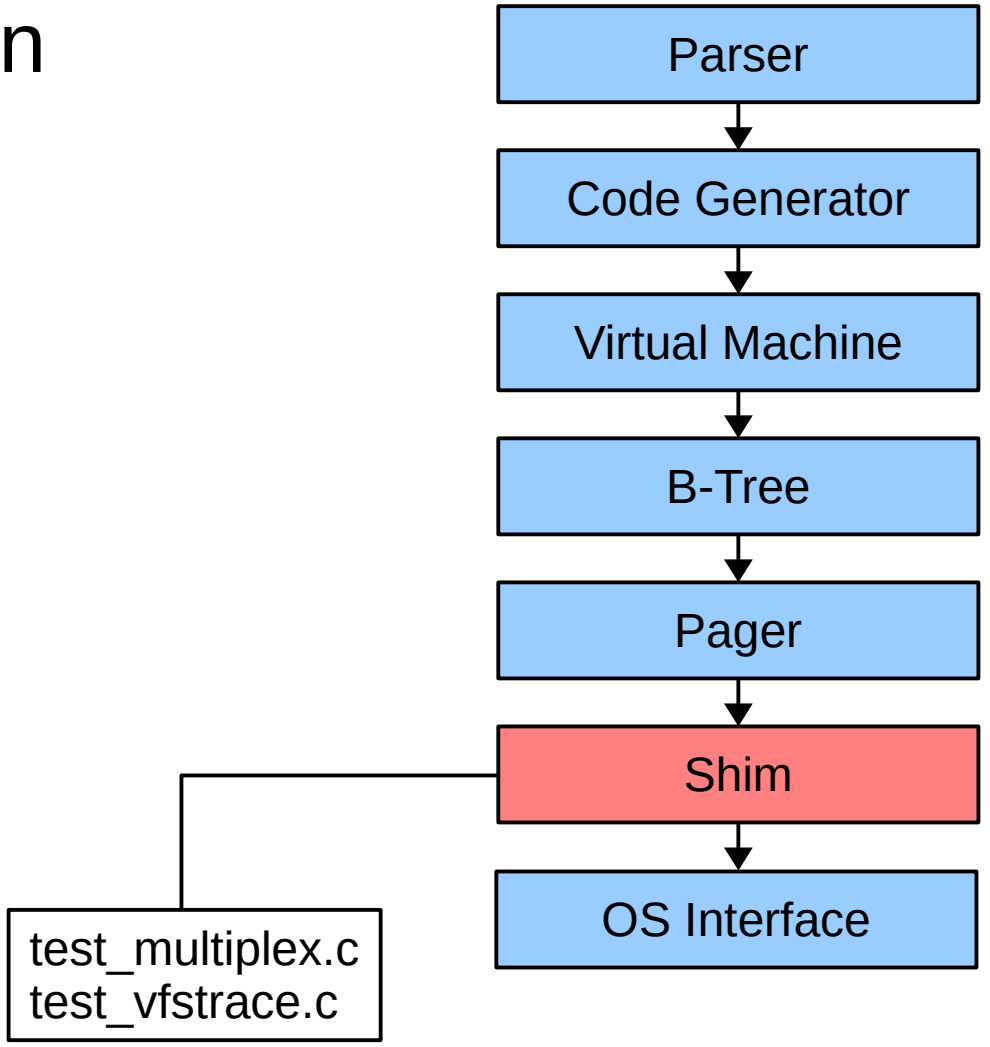
The SQLite Stack

- Platform-specific interface to the OS
- Run-time changeable
- Portability layer
- read()/write() or mmap()
- https://www.sqlite.org/vfs.html
- Direct I/O to hardware: test_onefile.c

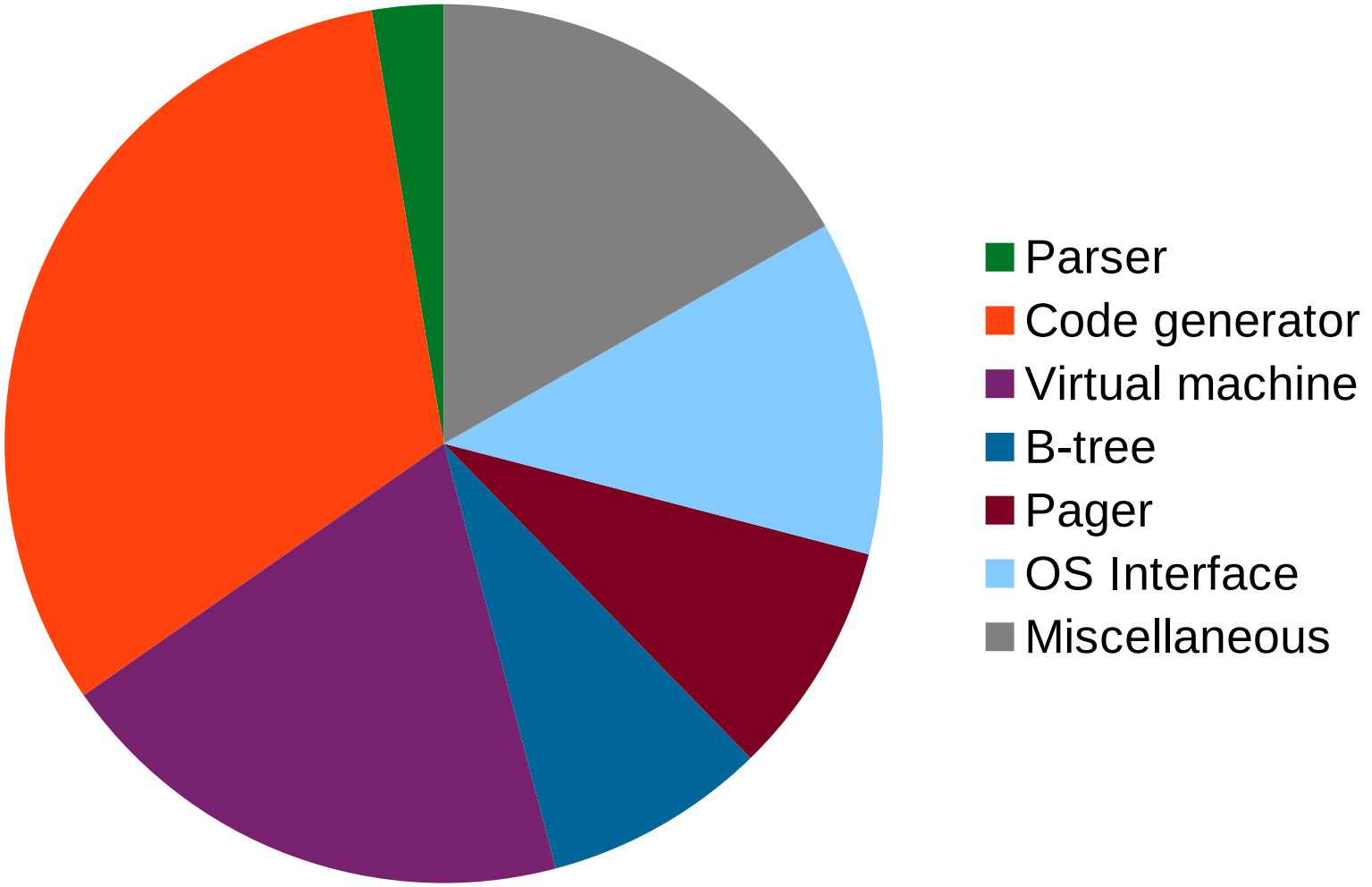


VFS Shims

- Inserted in between Pager and OS Interface
- Encryption
- Compression
- Logging
- Testing & fault injection
- And so forth...



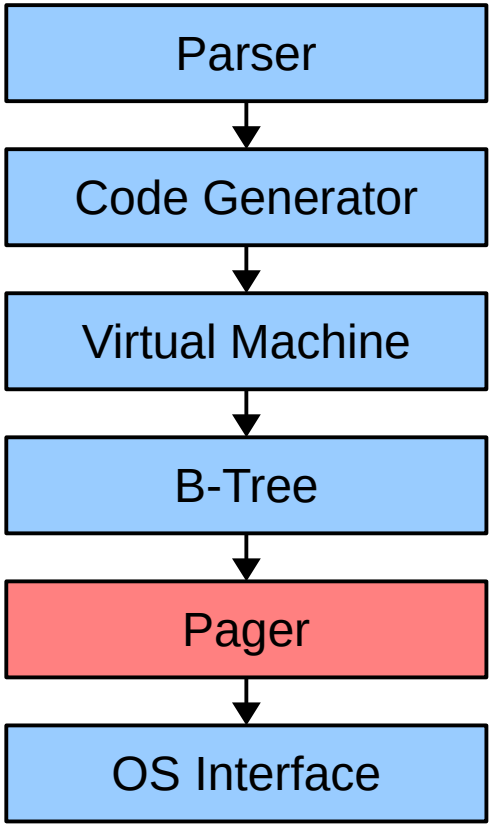
Lines of Source Code



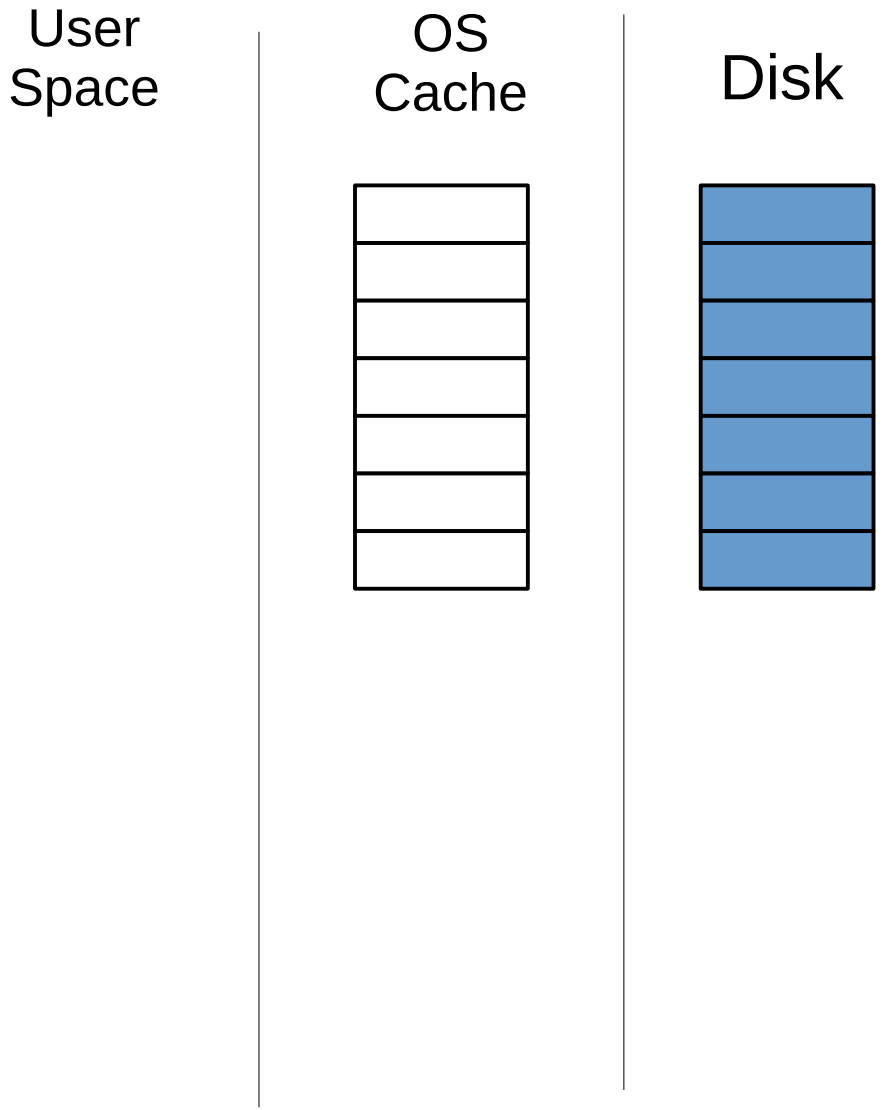
Further Details

The SQLite Stack

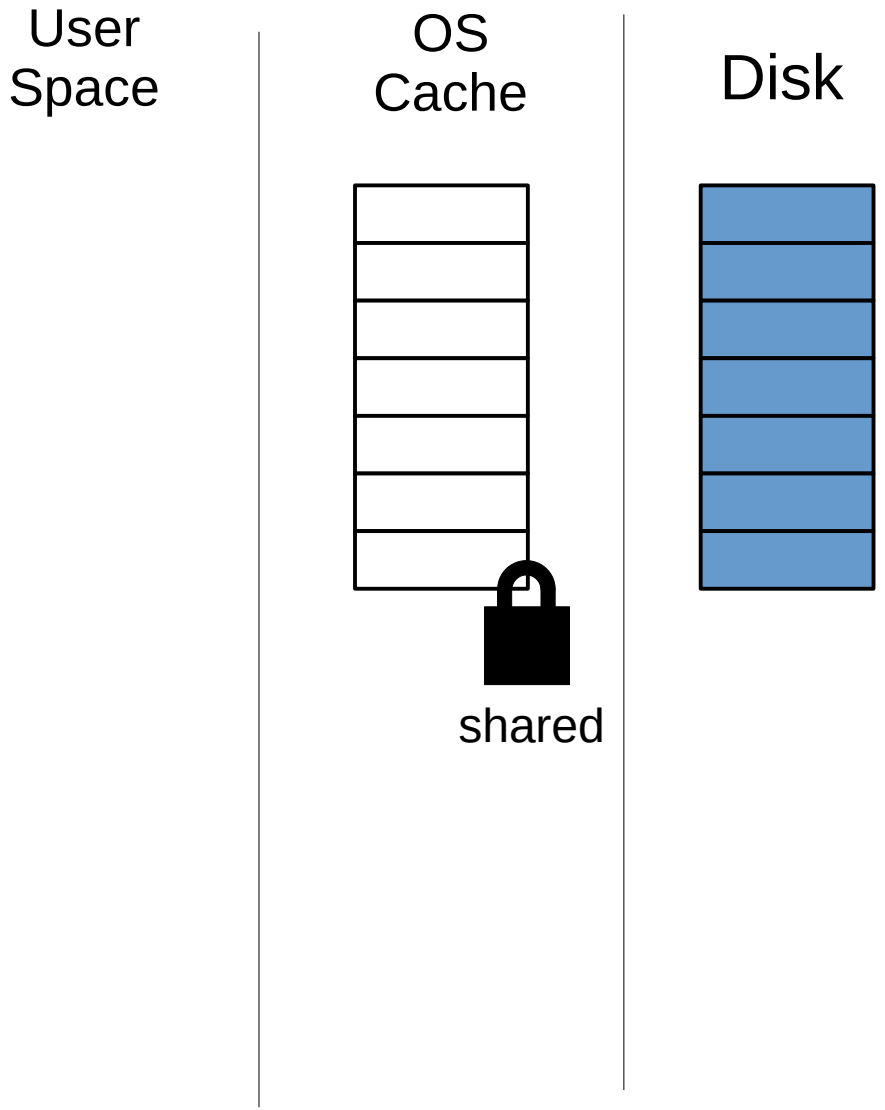
- Power-safe transactions
  - Rollback mode
  - Write-ahead log (WAL) mode
- Concurrency control
- In-memory cache of disk content



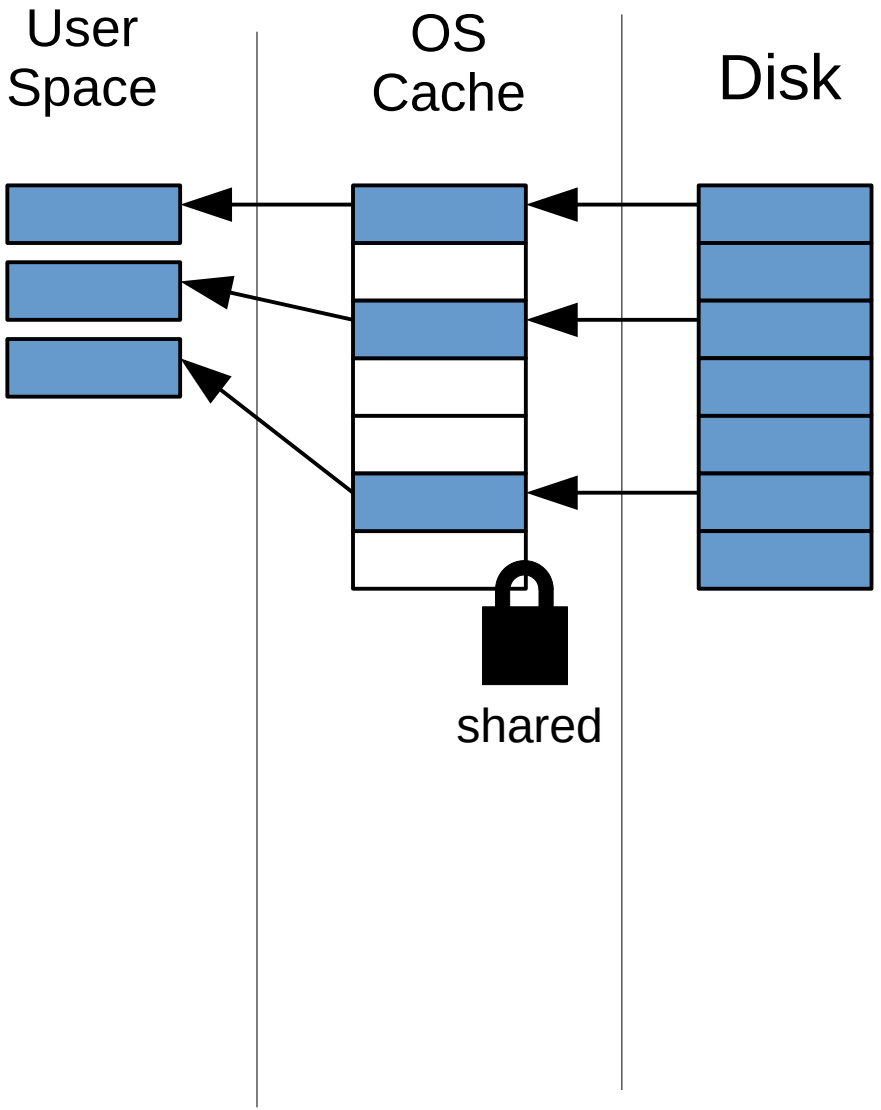
Rollback Journaling



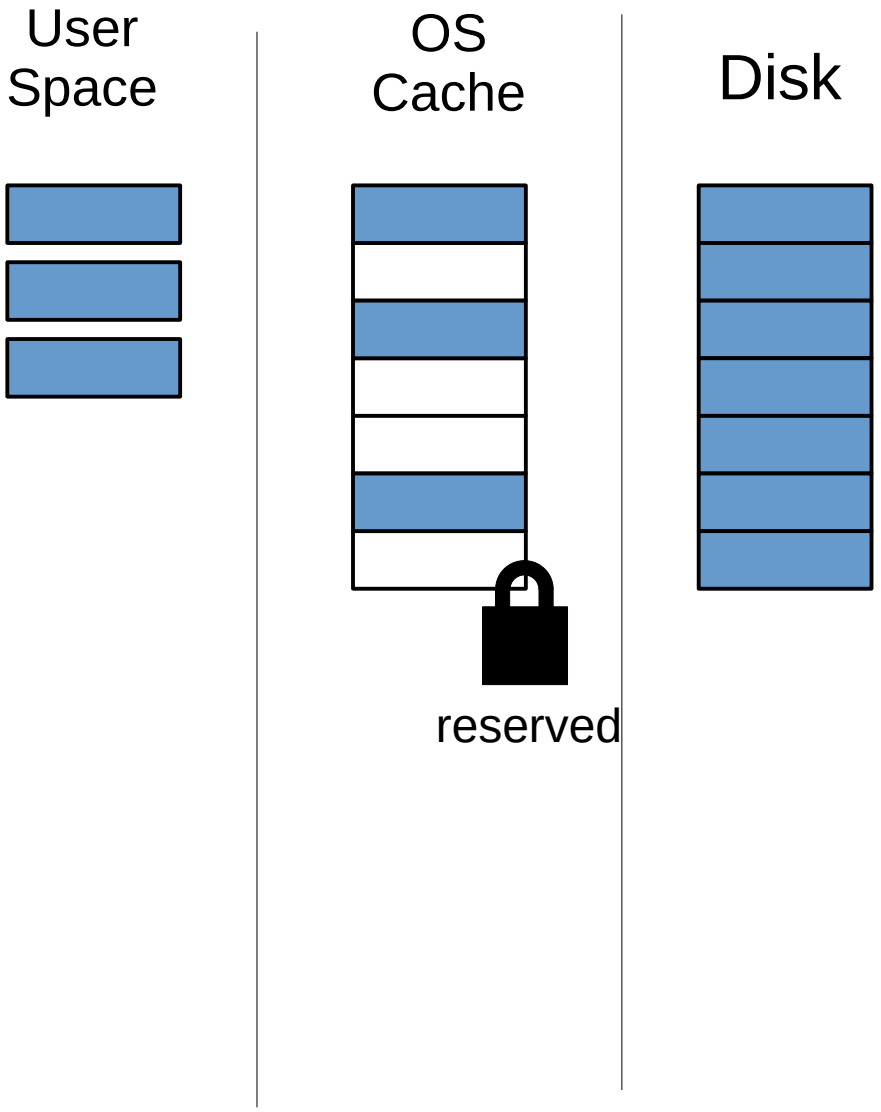
Rollback Journaling



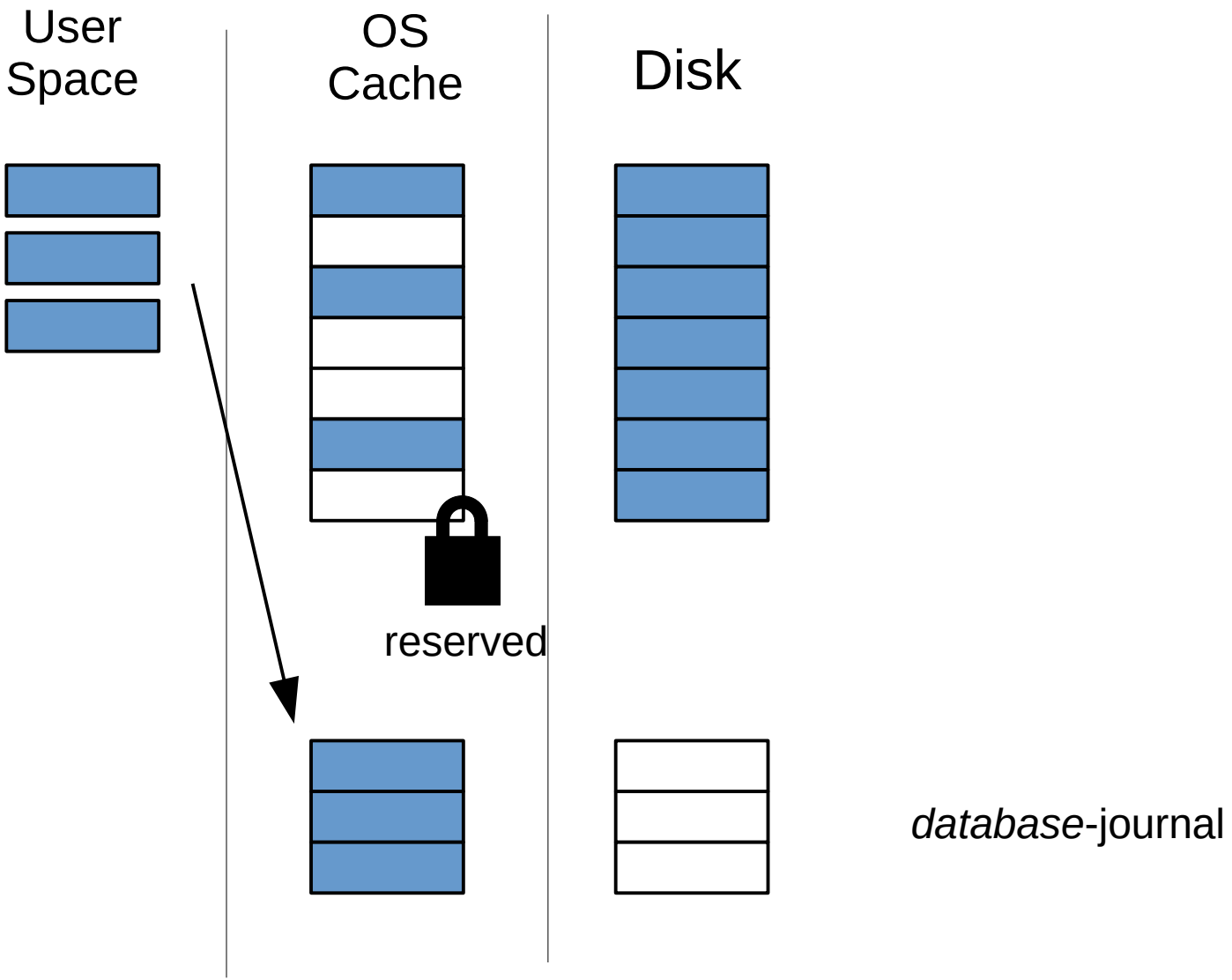
Rollback Journaling



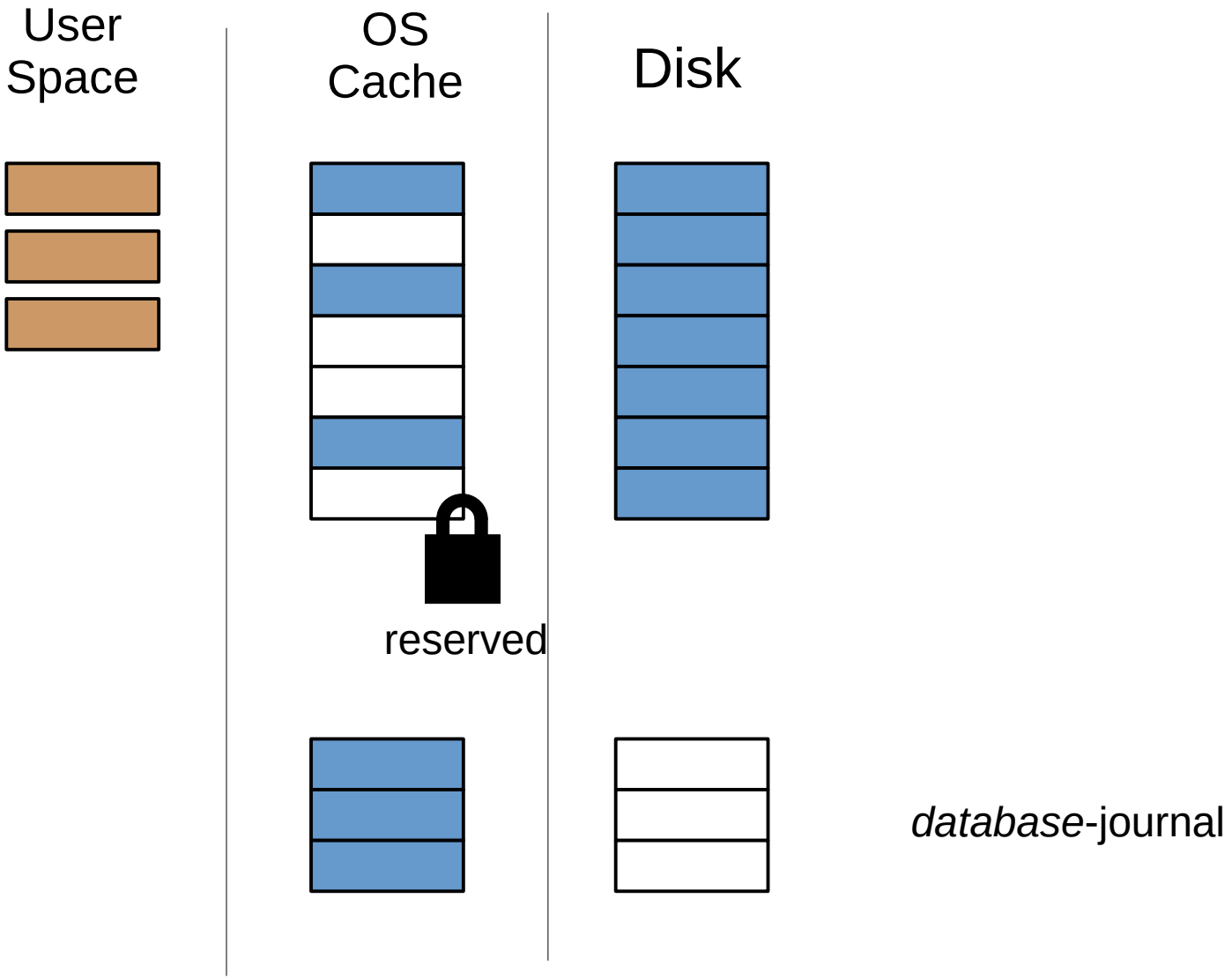
Rollback Journaling



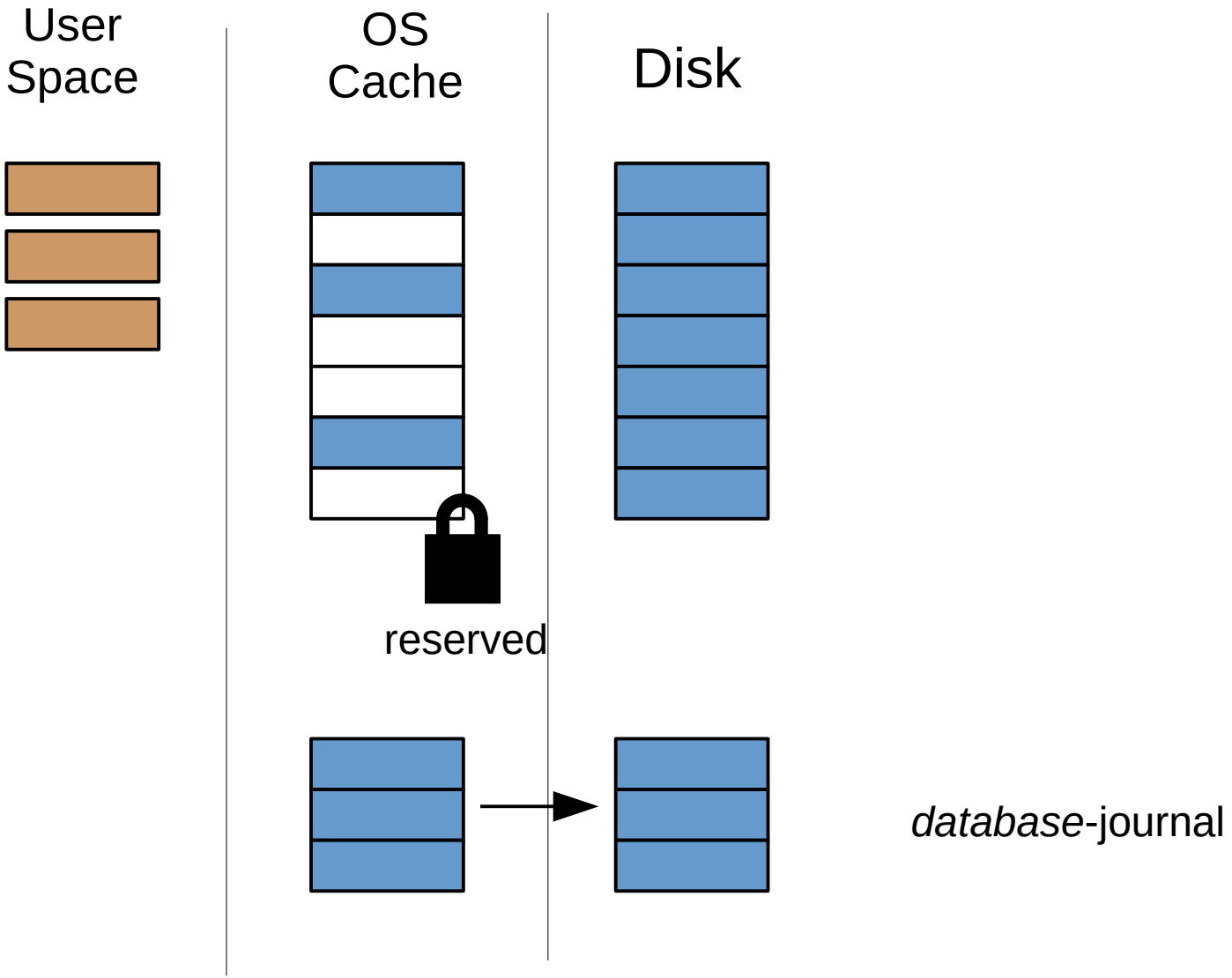
Rollback Journaling



Rollback Journaling

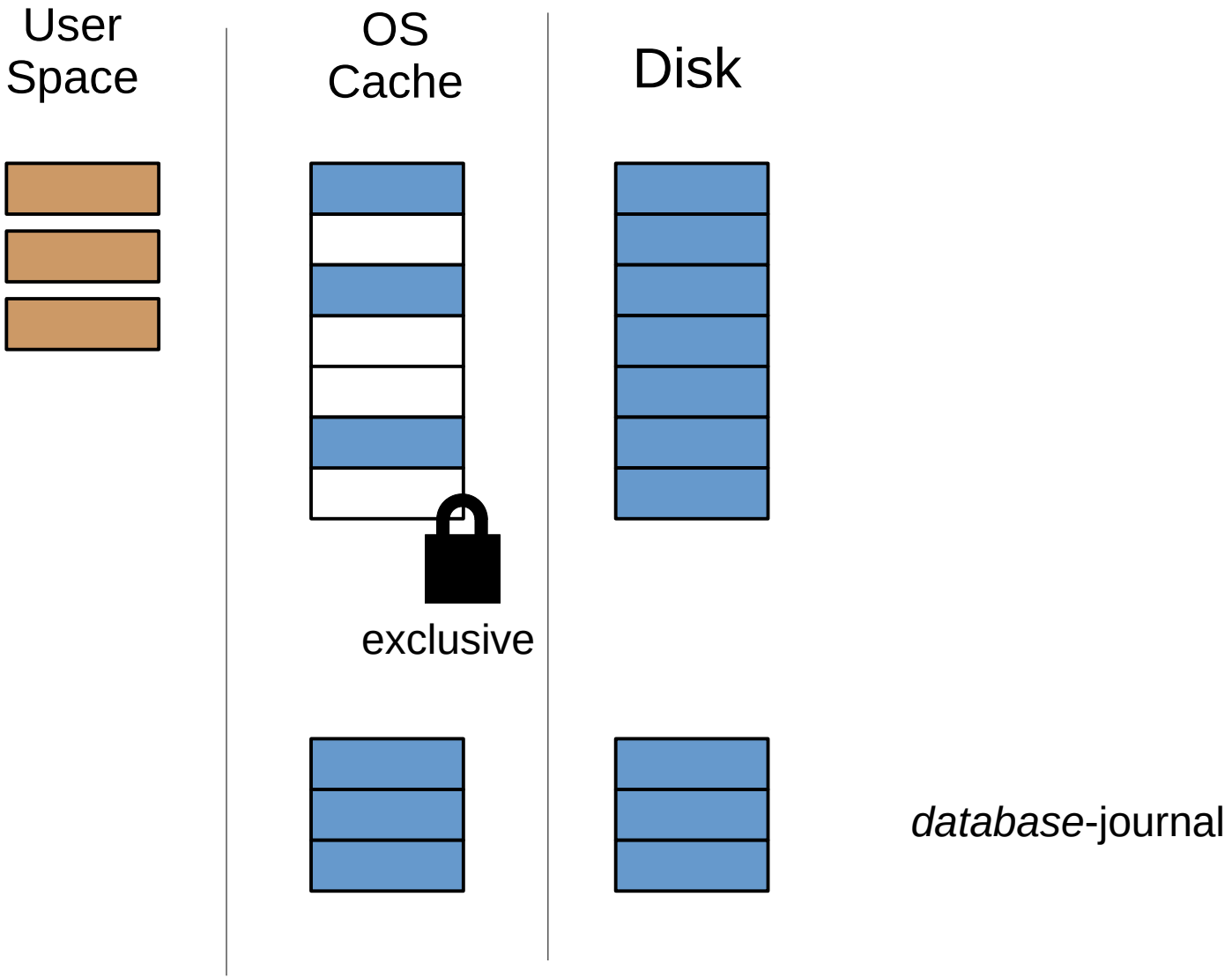


Rollback Journaling

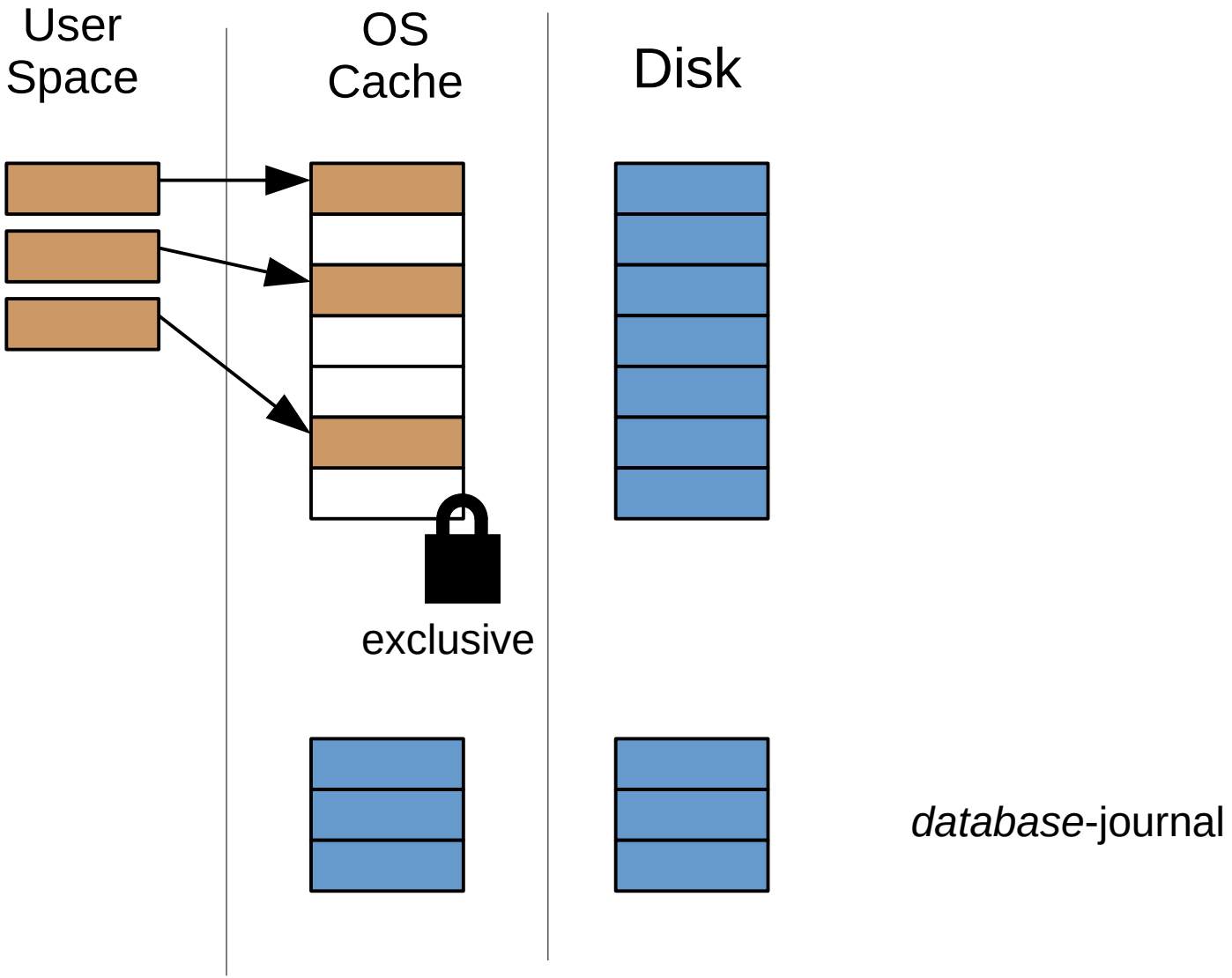


➔ *Disable this step using PRAGMA synchronous=OFF*

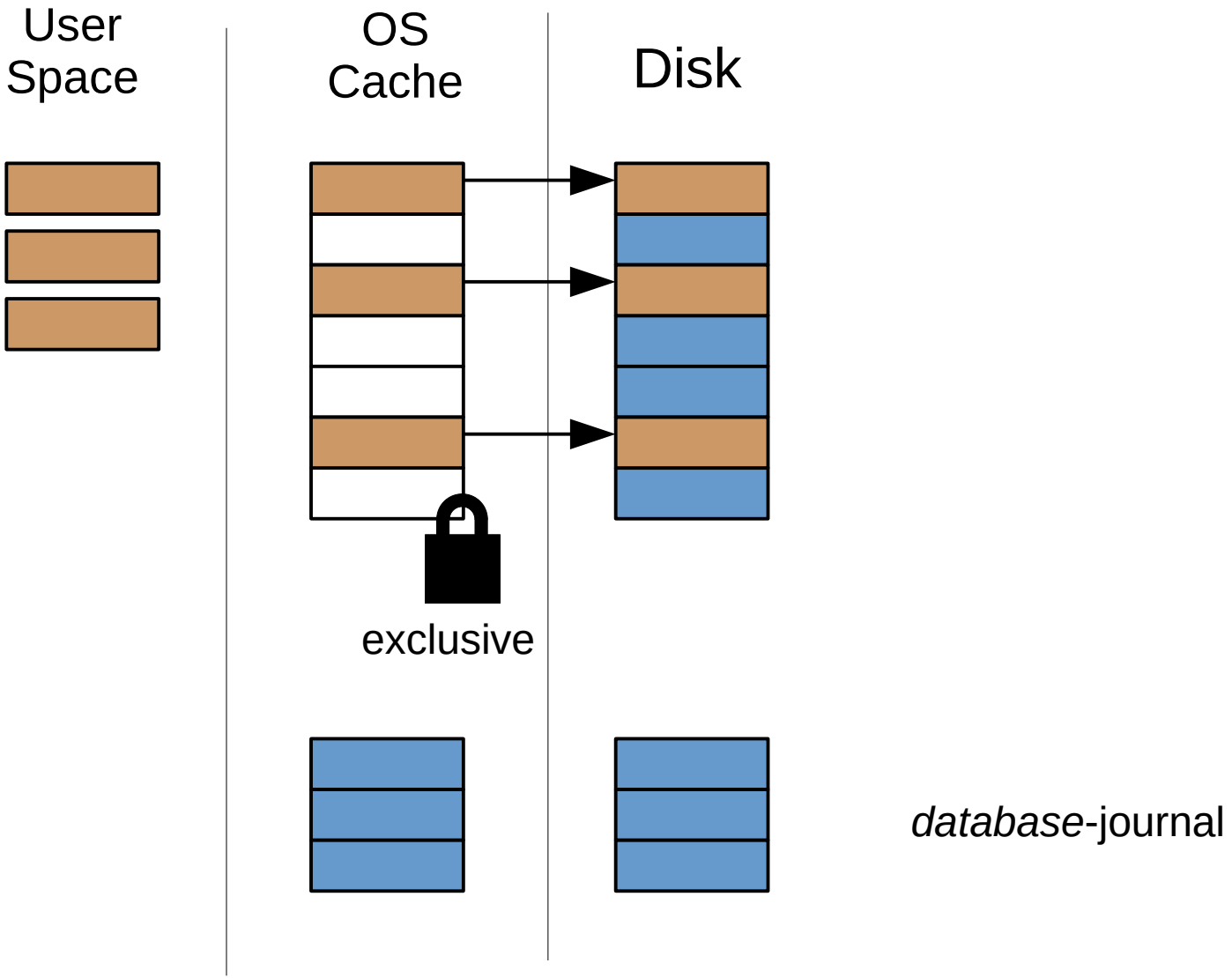
Rollback Journaling



Rollback Journaling

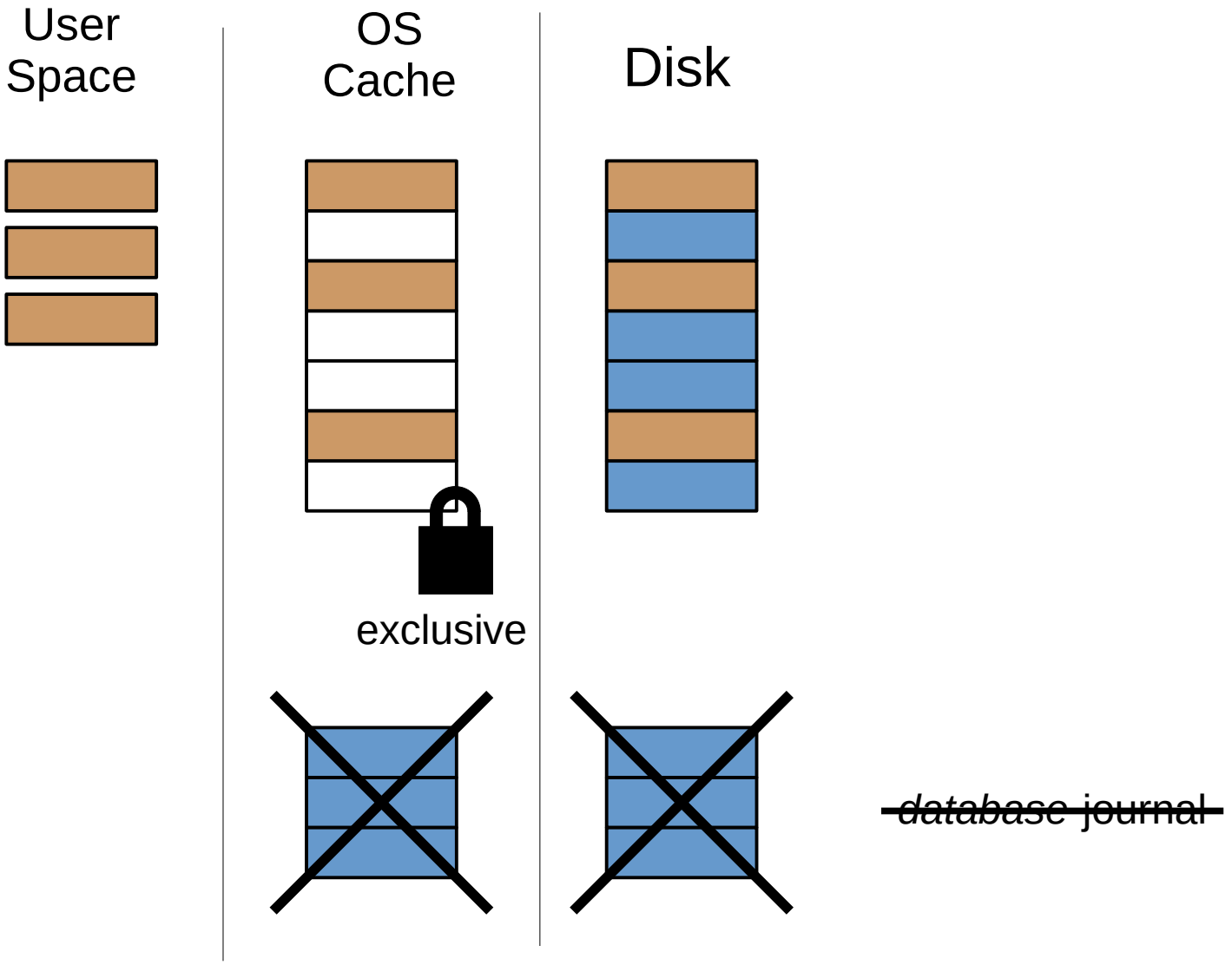


Rollback Journaling

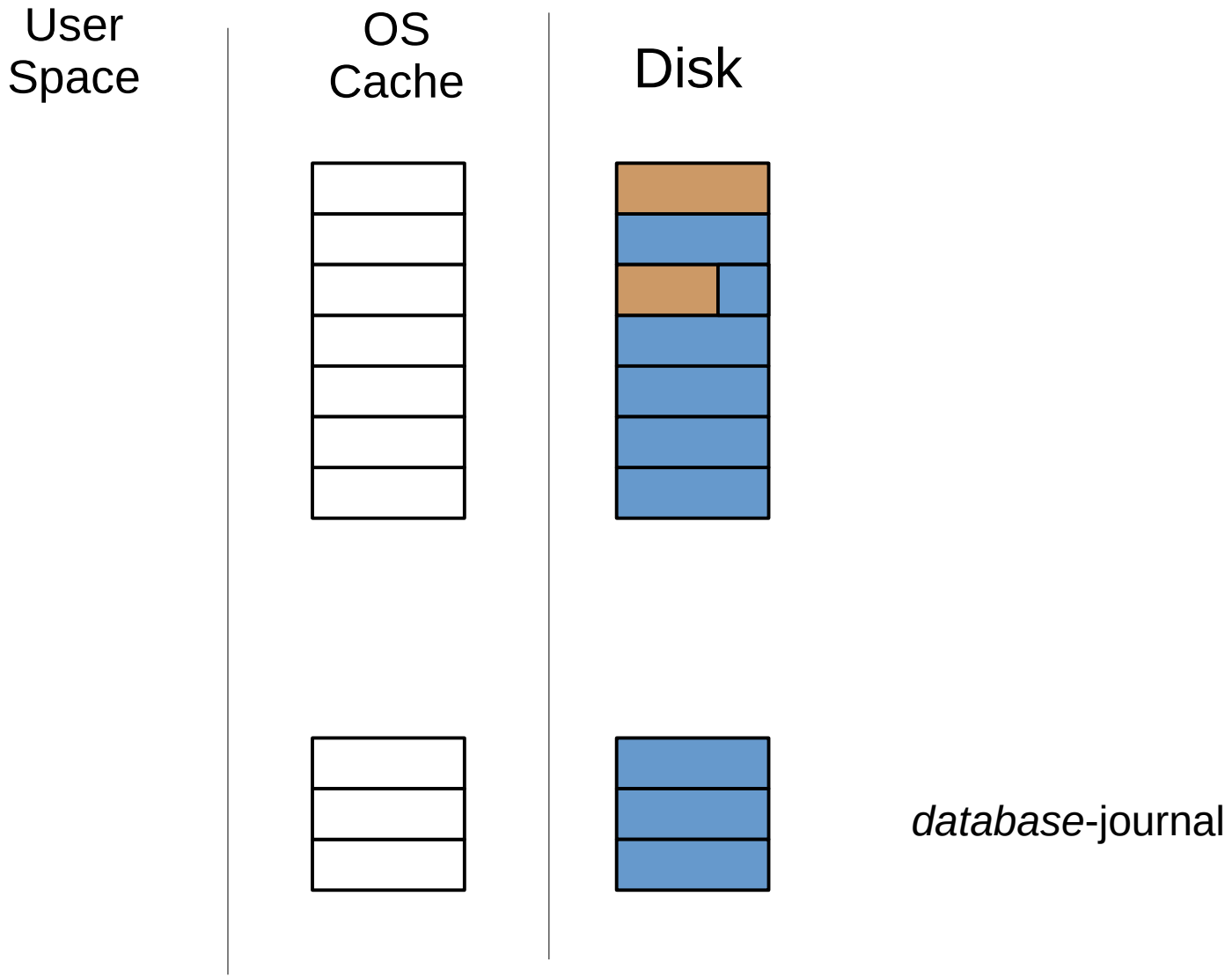


➔ *Disable this step using PRAGMA synchronous=OFF*

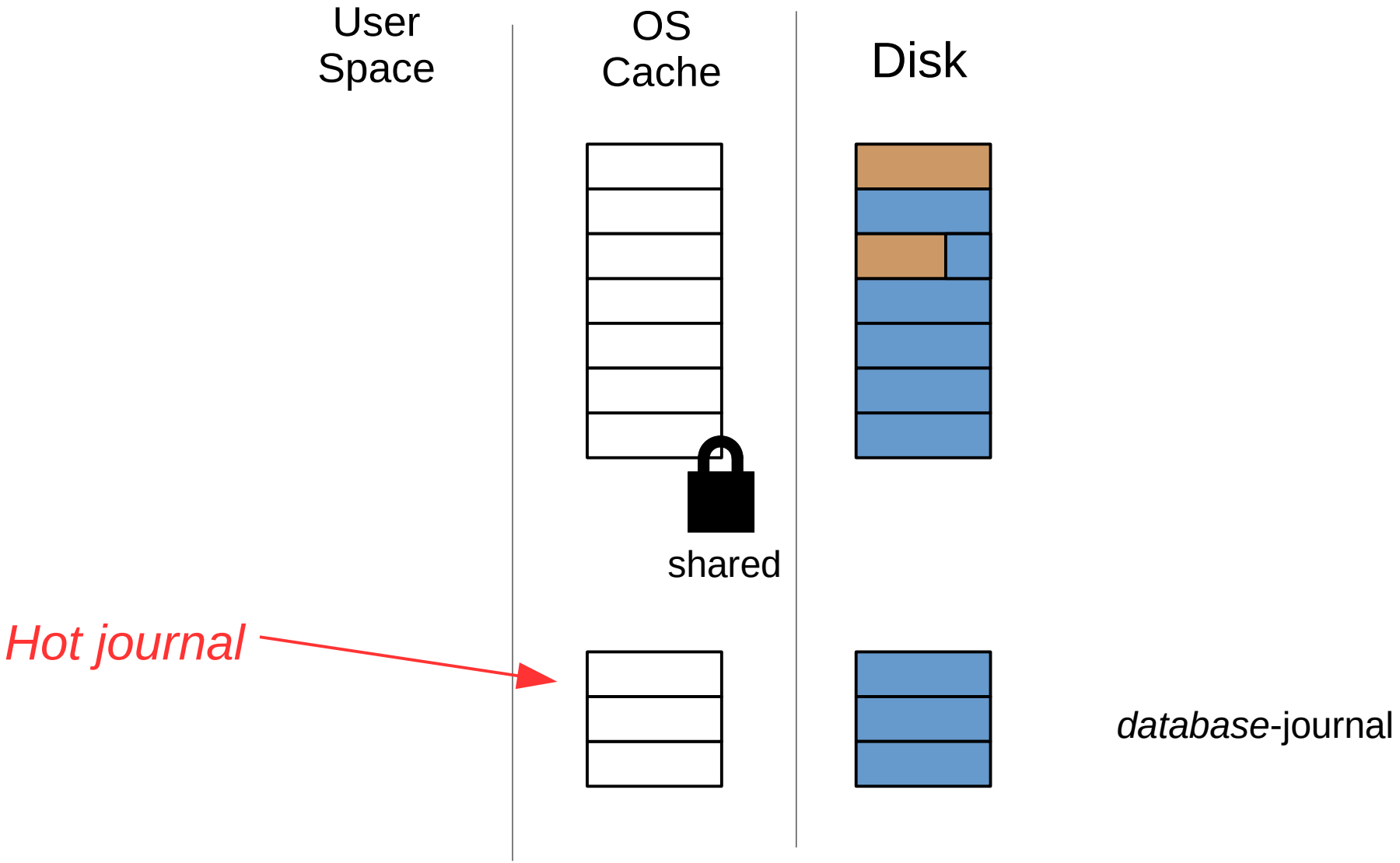
Rollback Journaling



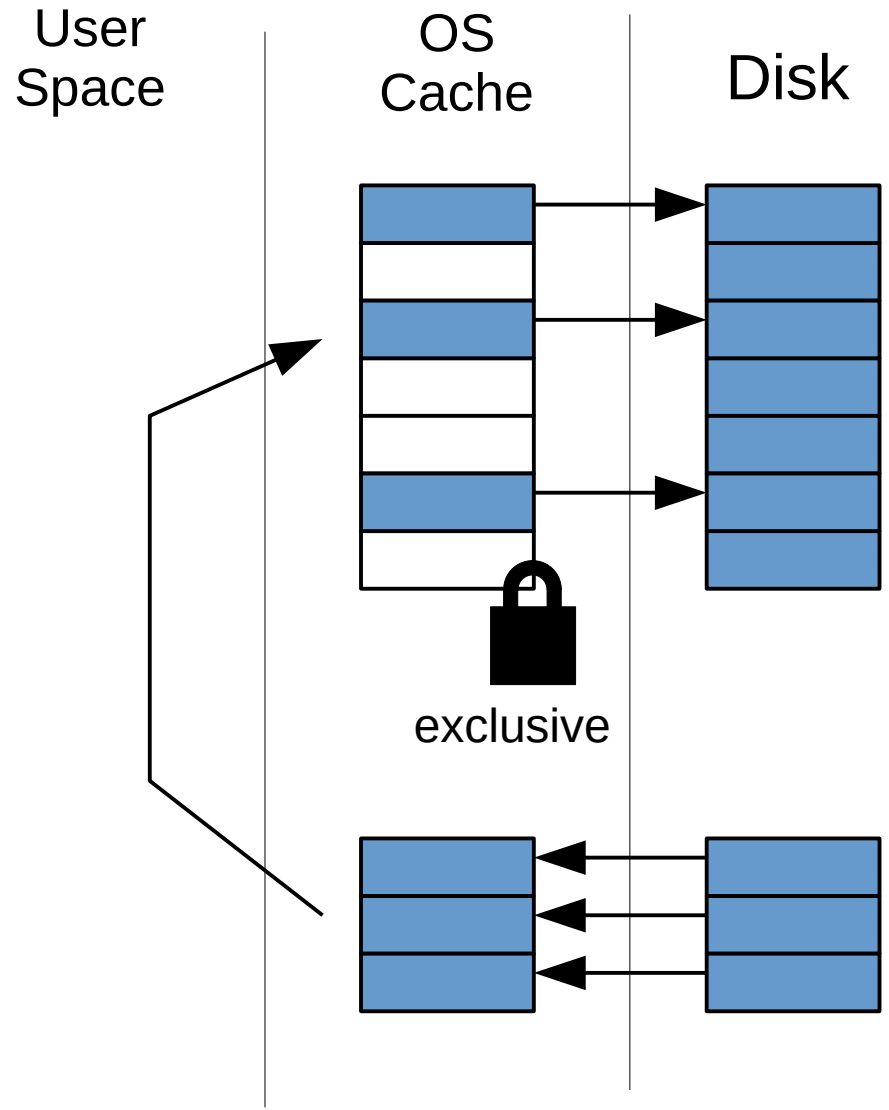
Rollback Mode Crash Recovery



Rollback Mode Crash Recovery

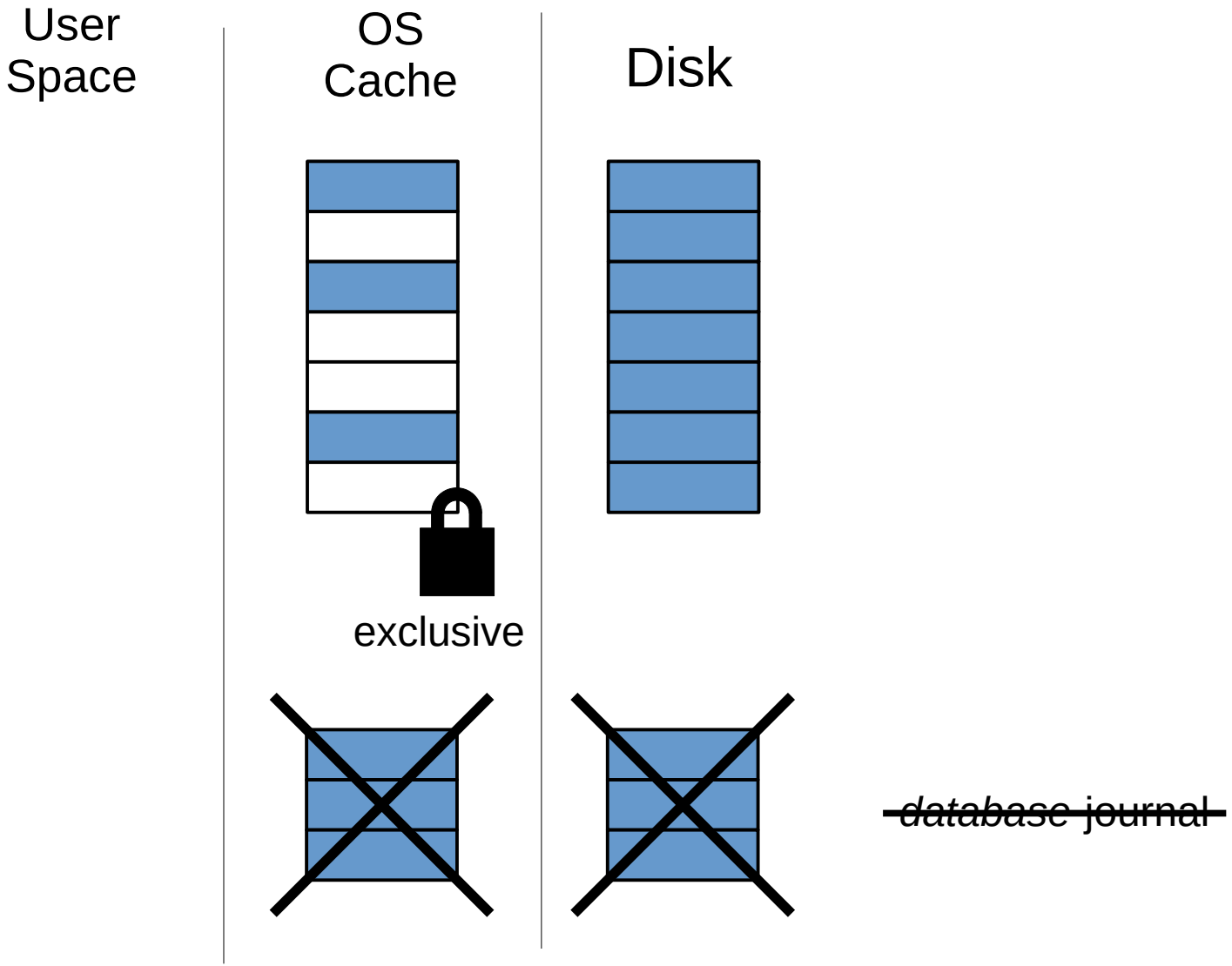


Rollback Mode Crash Recovery

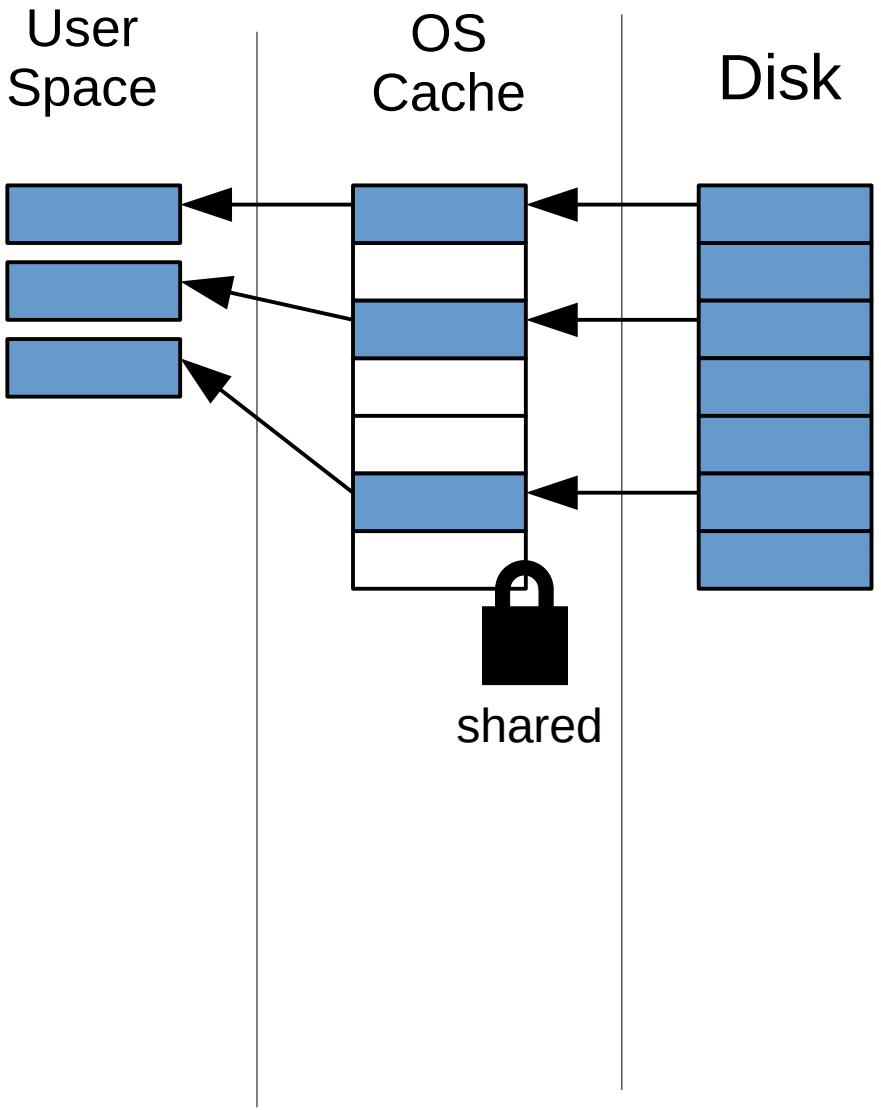


database-journal

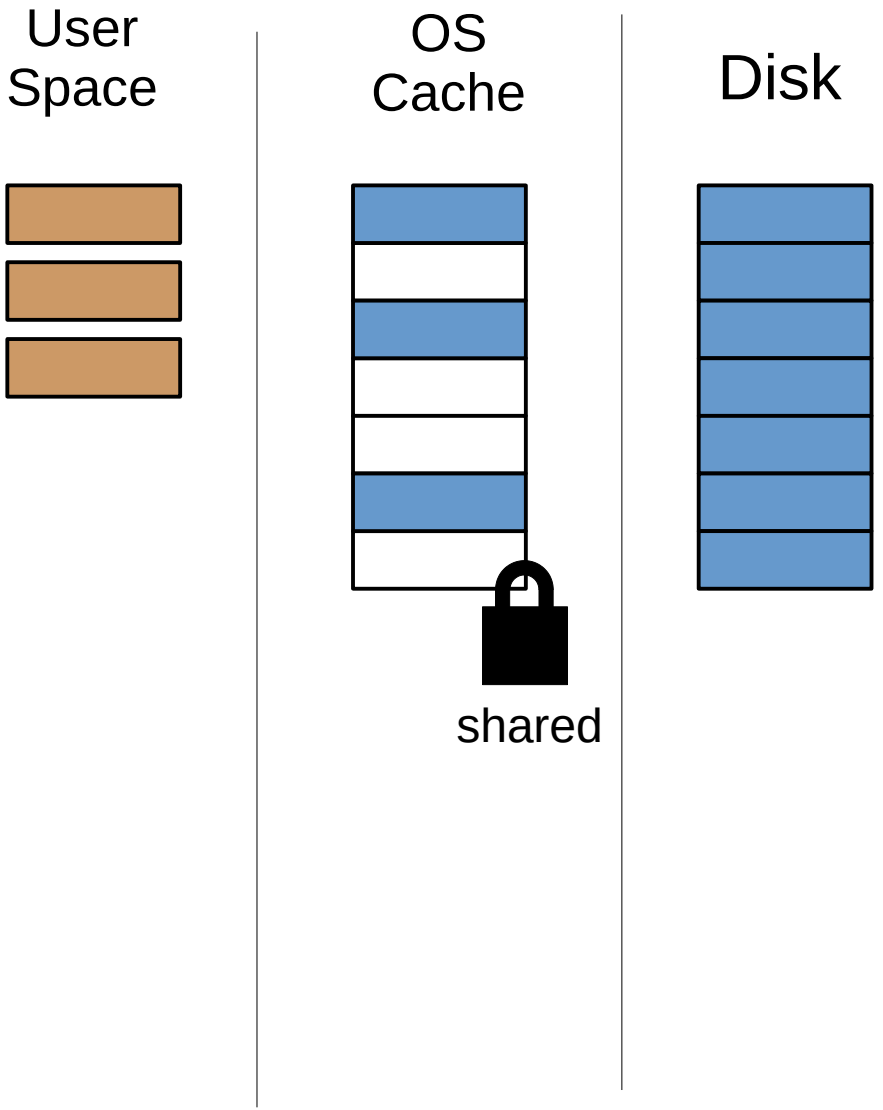
Rollback Mode Crash Recovery



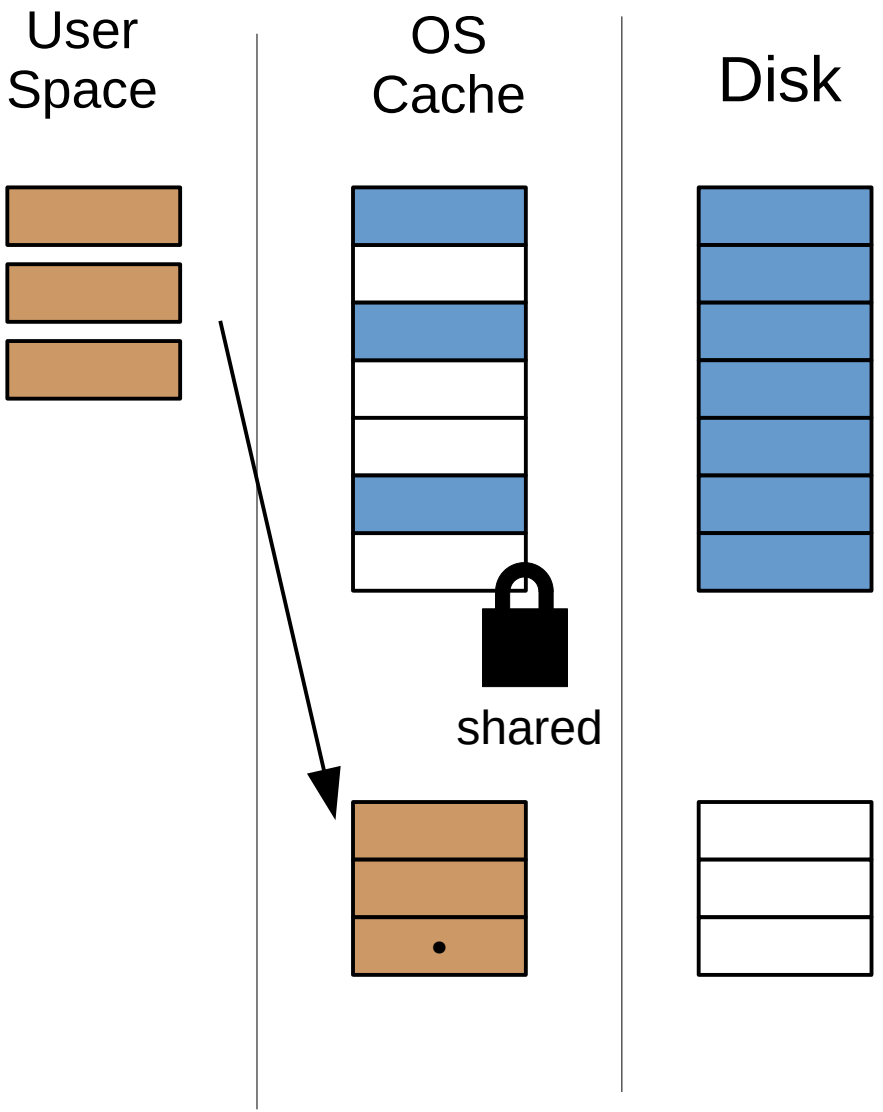
Write-Ahead Log



Write-Ahead Log

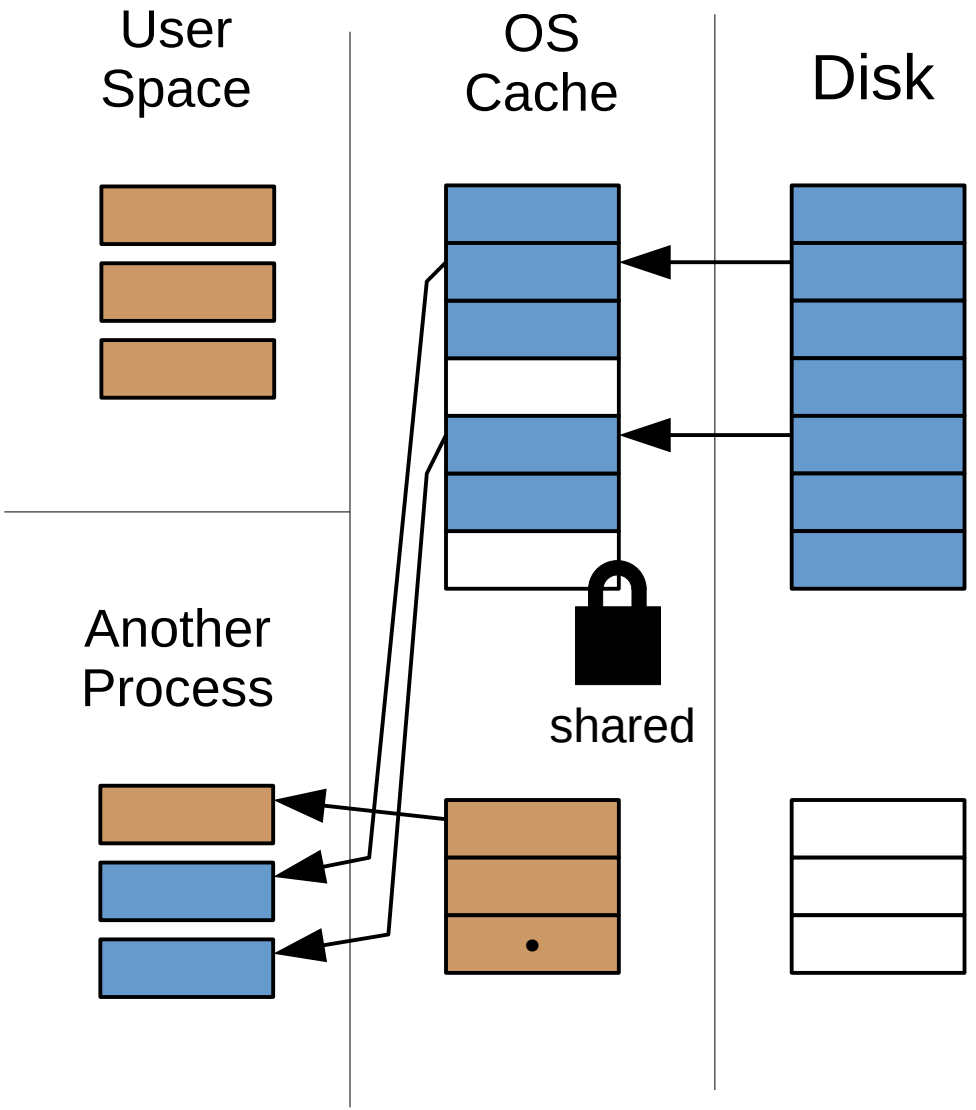


Write-Ahead Log



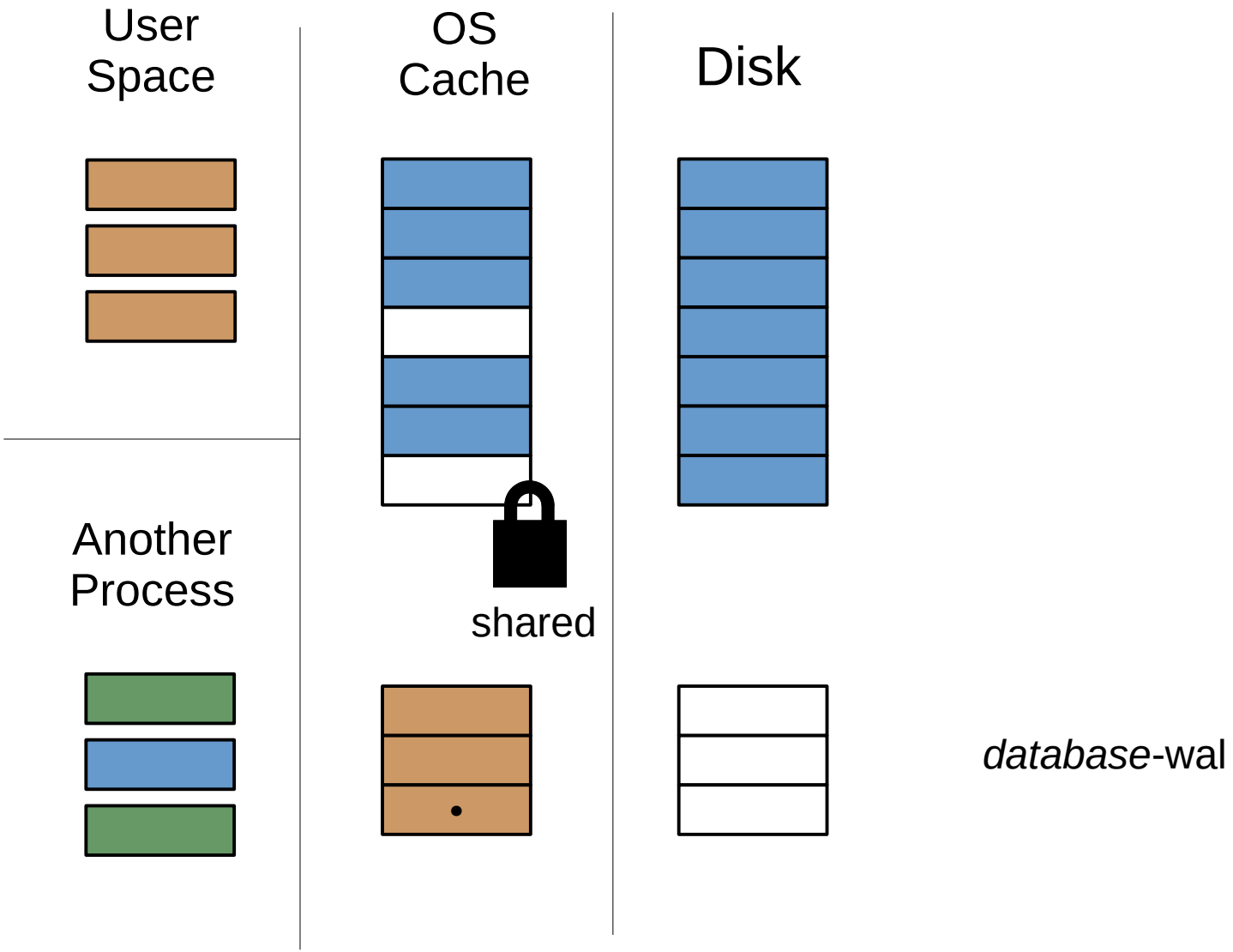
database-wal

Write-Ahead Log

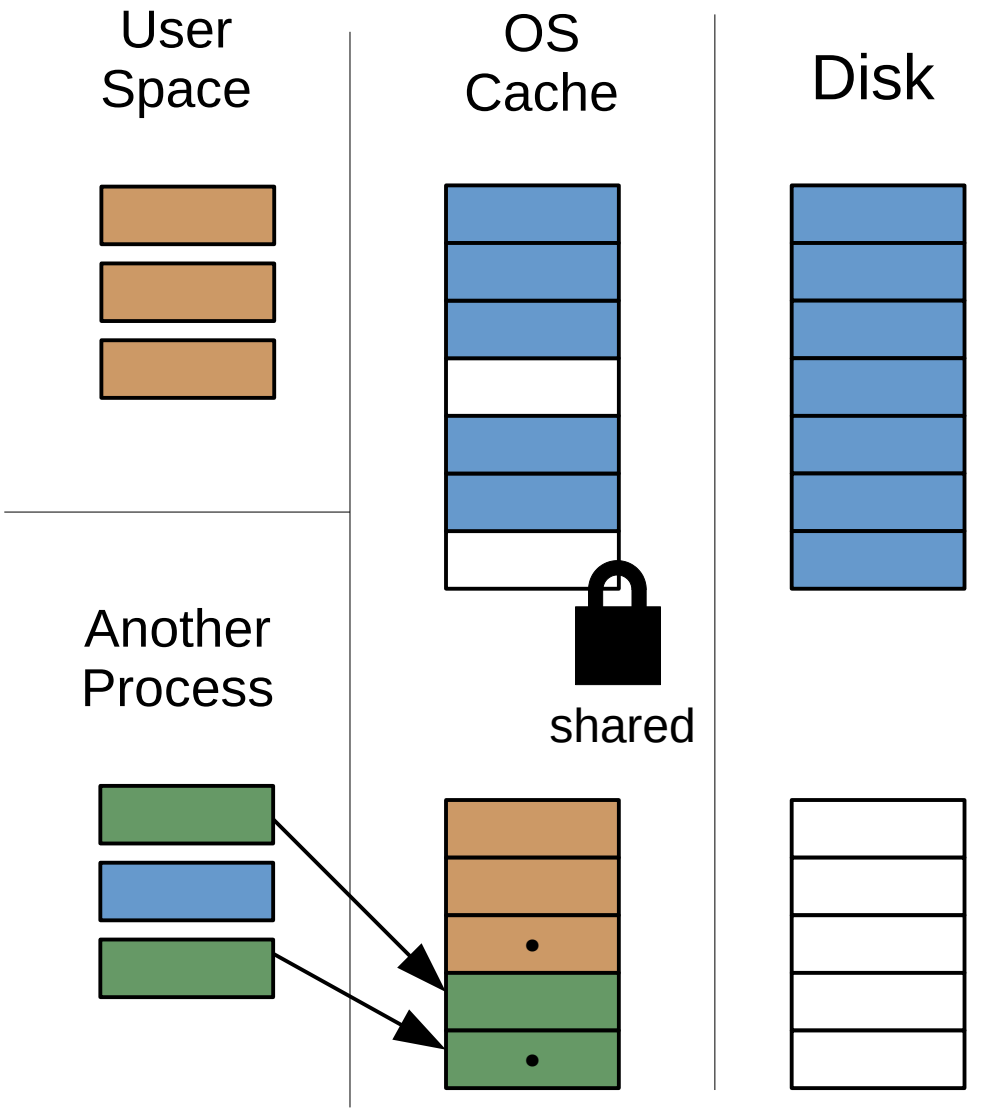


database-wal

Write-Ahead Log

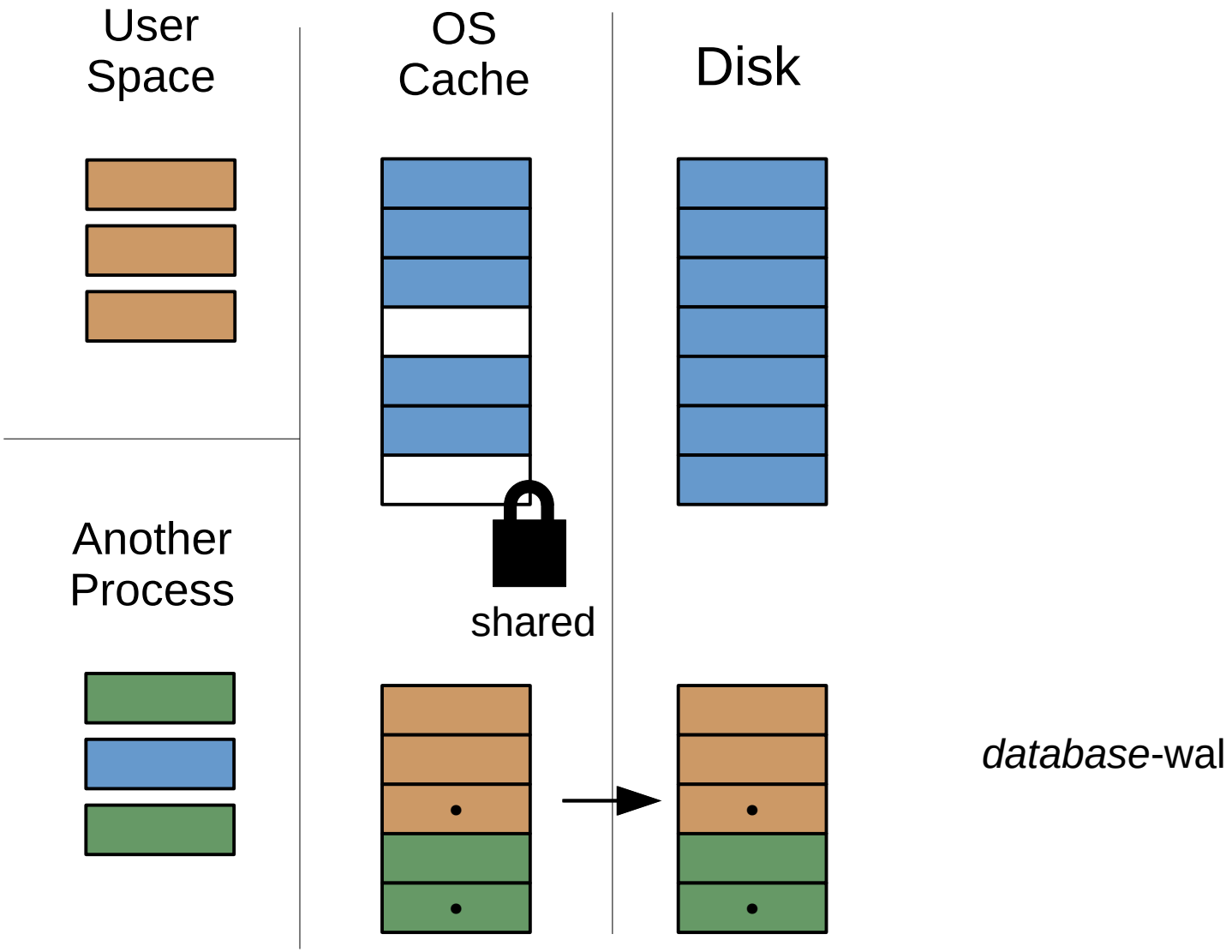


Write-Ahead Log



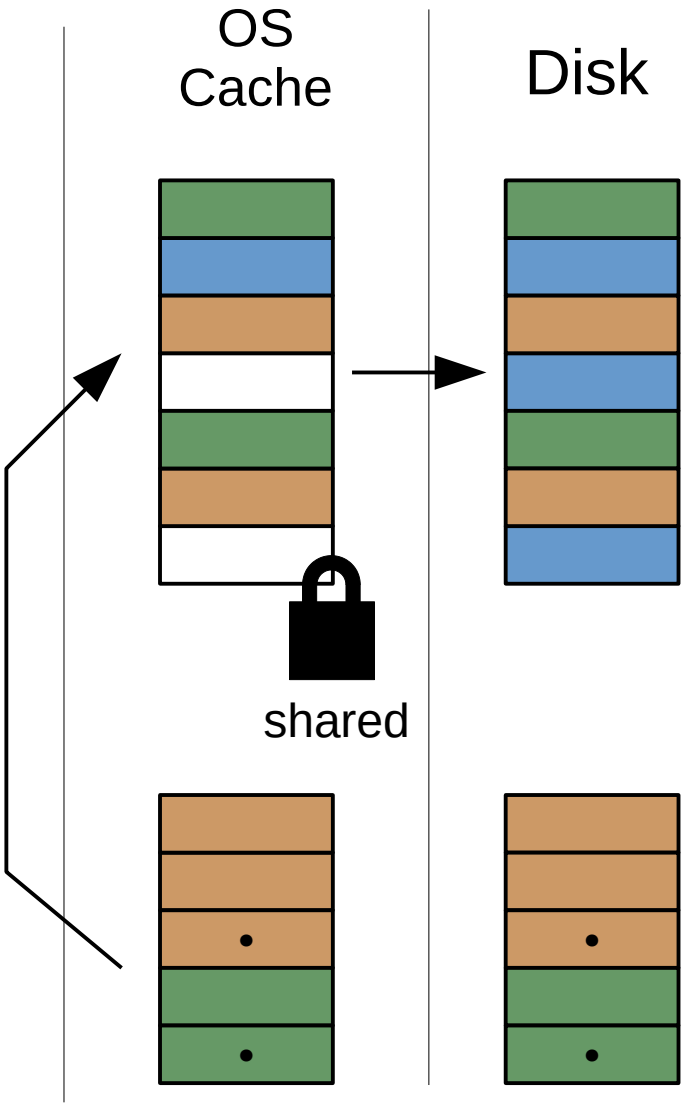
database-wal

Write-Ahead Log



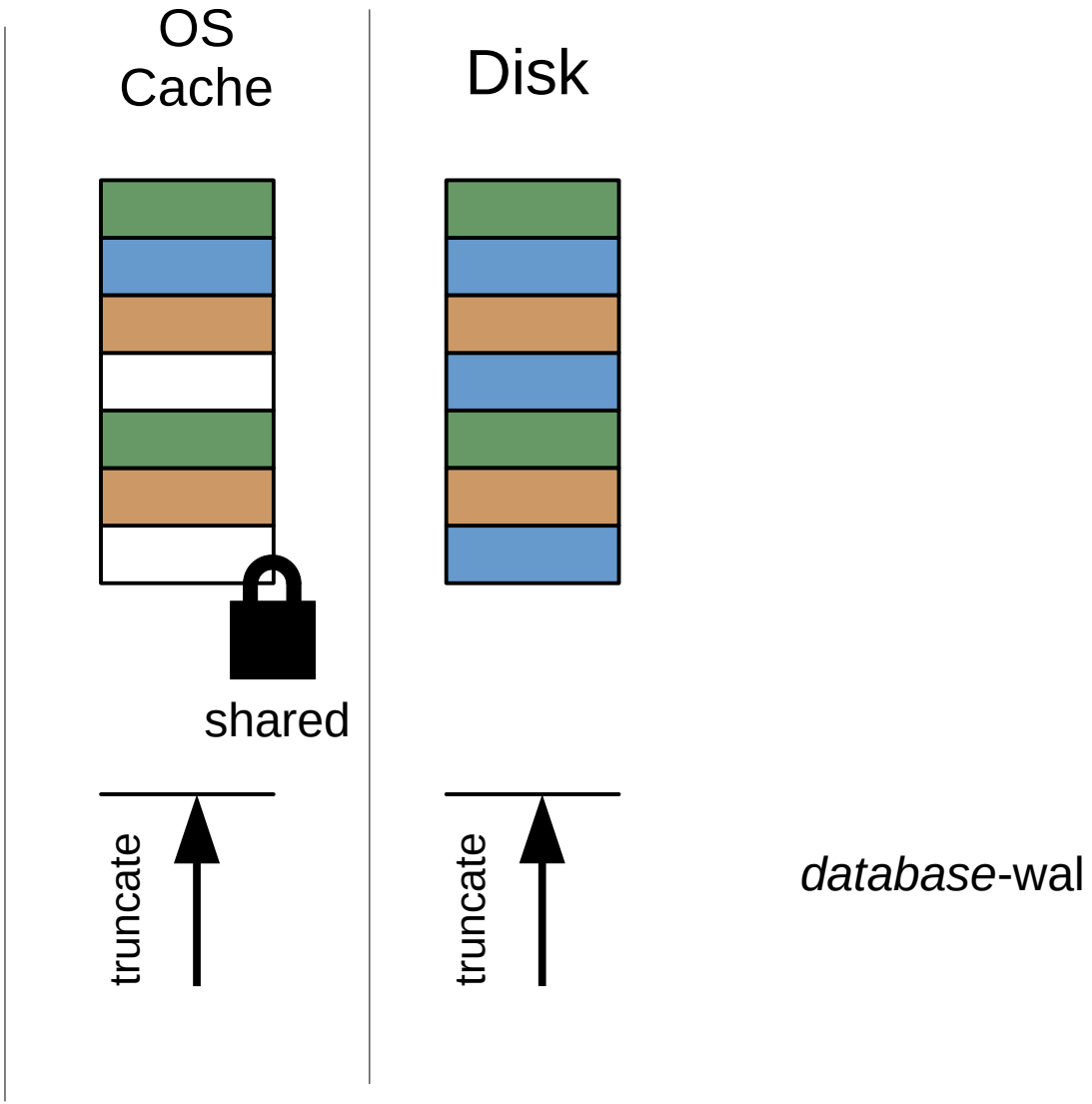
✔ *Done for every commit with PRAGMA synchronous=FULL*

Checkpoint



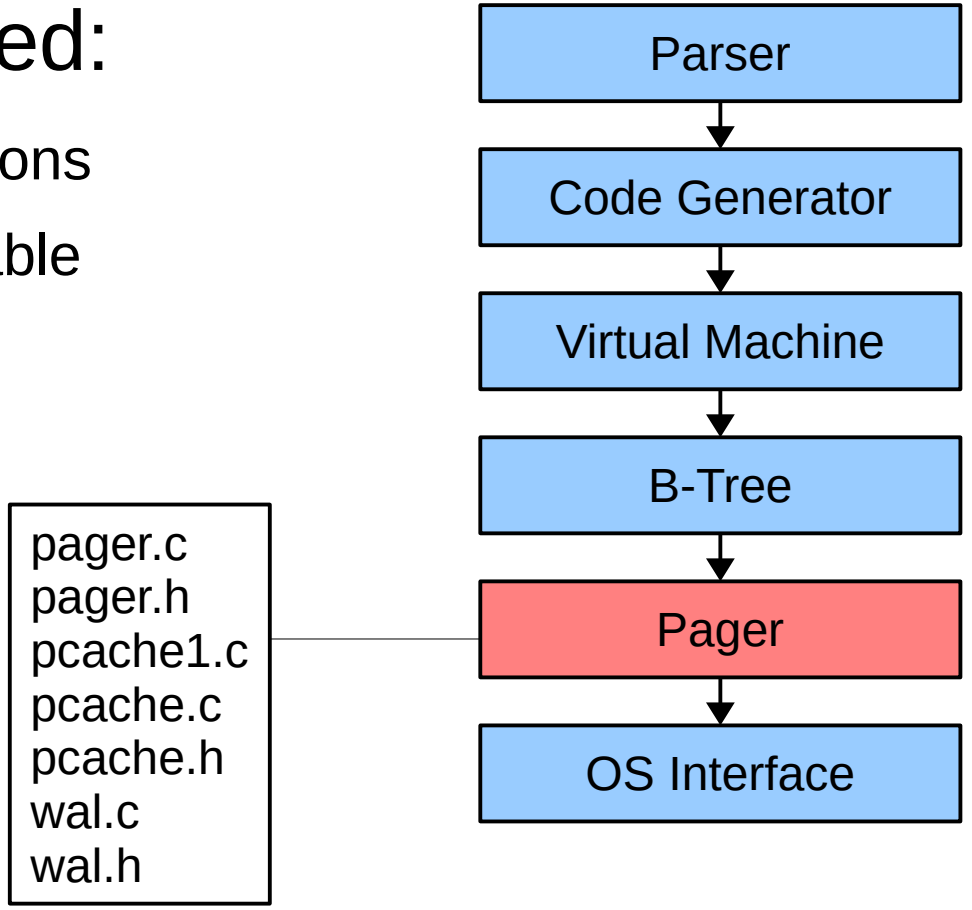
database-wal

Checkpoint



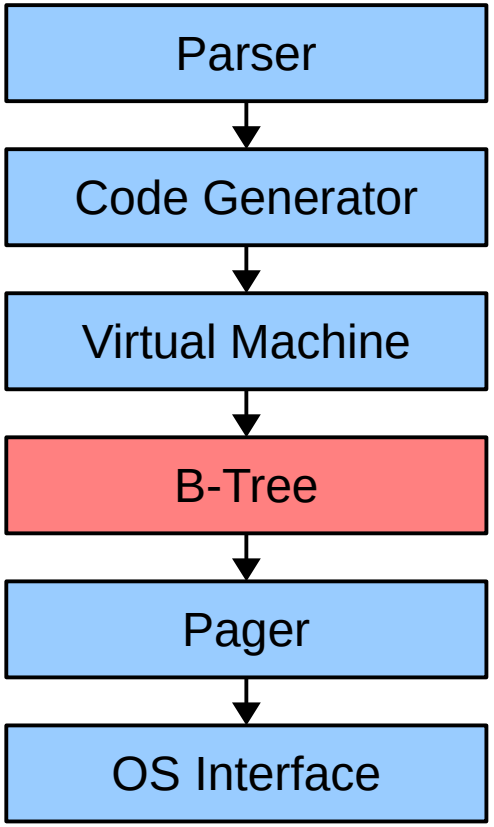
Pager Summary

- Topics not covered:
  - Nested Transactions
  - Start-time pluggable page cache
  - Crash testing

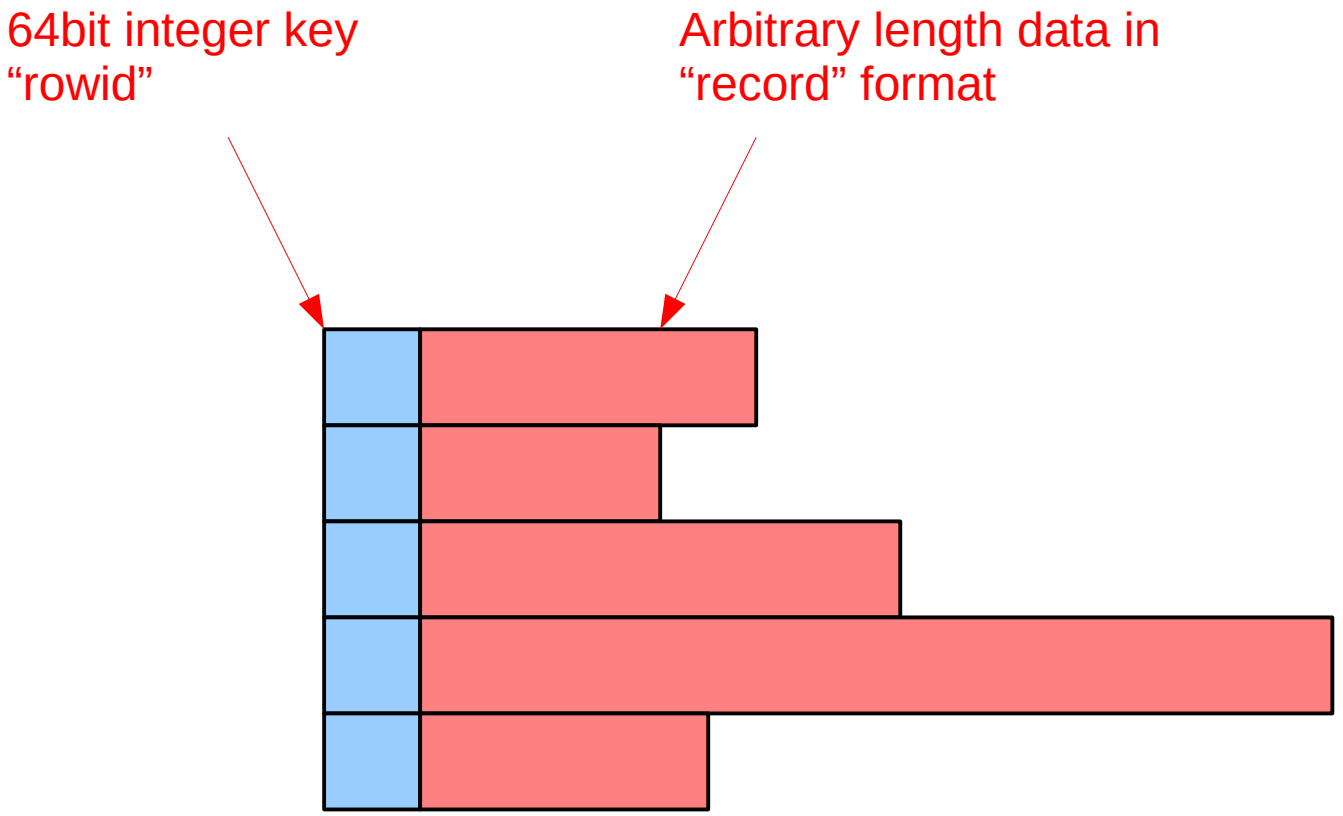


The B-tree Layer

- Multiple B-trees per file
- B+trees with 64-bit integer keys and arbitrary blob content
- B-trees with arbitrary blob keys and no content

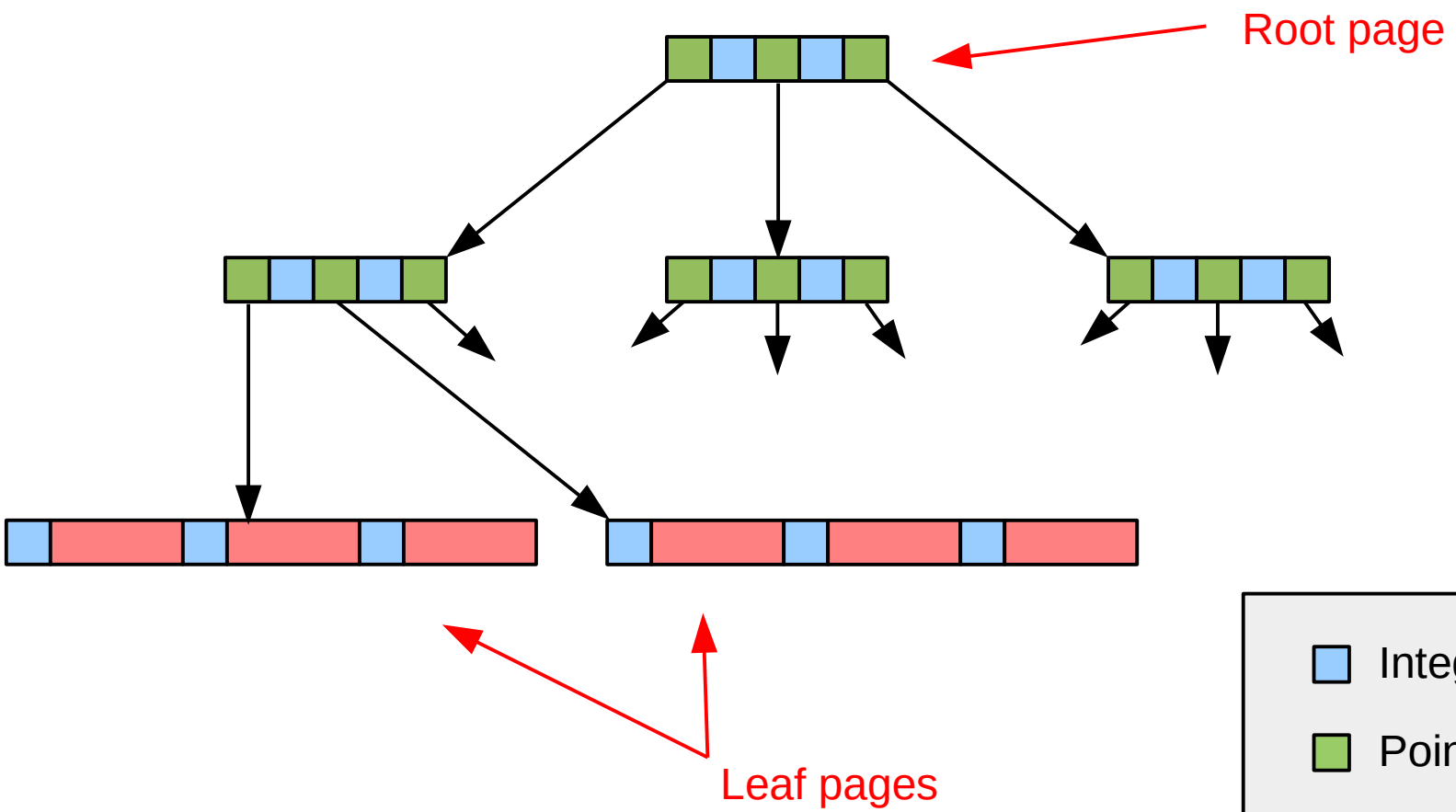


Logical View of SQL Table Storage



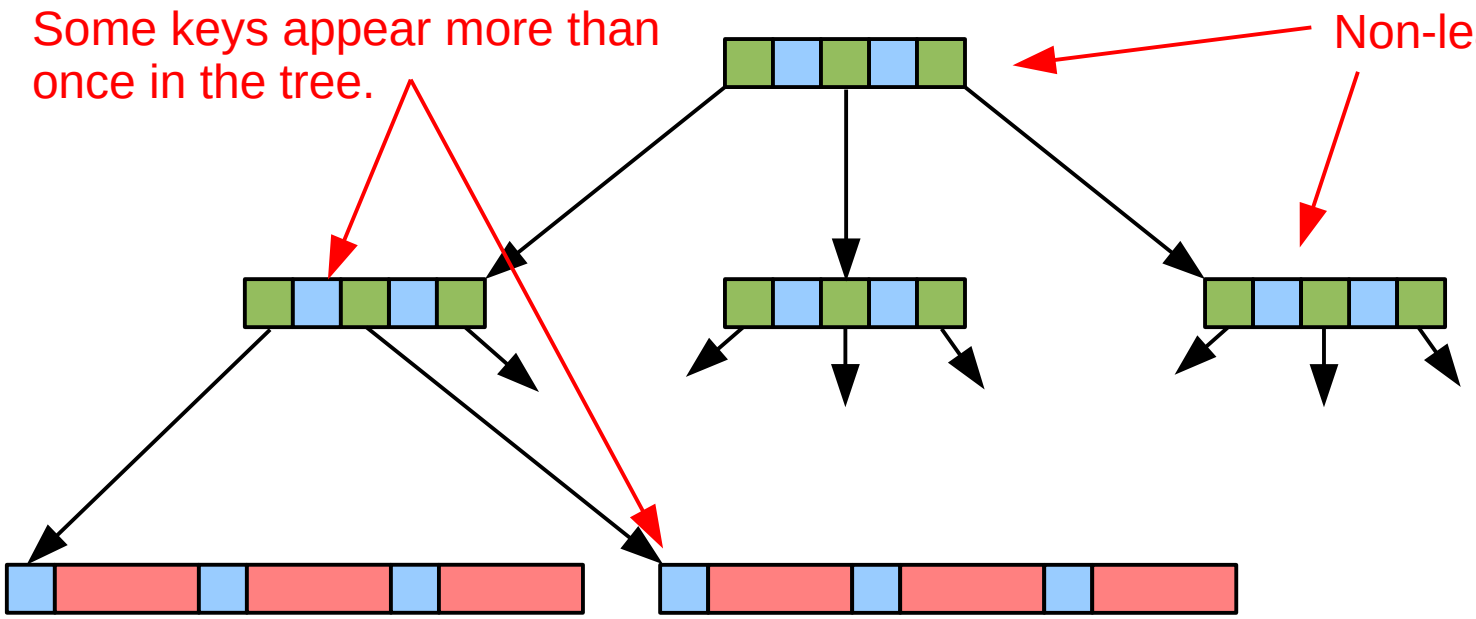
B+tree Structure

(used by most SQL tables)



B+tree Structure

(used by most SQL tables)



- Key + Data in leaves
- Combined key+data is a “cell”
- As few as one “cell” per page.

Note: B-tree balance operations are optimized for the appending.

Integer key

Pointer to lower page

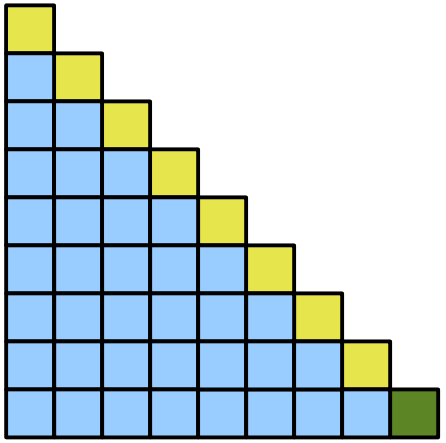
Binary content

Variable Length Integers

1xxxxxxx - high bit set. 7 bits of data

0xxxxxxx - high bit clear. 7 bits of data

xxxxxxxx - 8 bits of data



0 to 127
128 to 16383
16384 to 2097151
2097152 to 268435455
268435456 to 34359738367
34359738368 to 4398046511103
4398046511104 to 562949953421311
562949953421312 to 72057594037927935
Less than 0 or greater than 72057594037927935

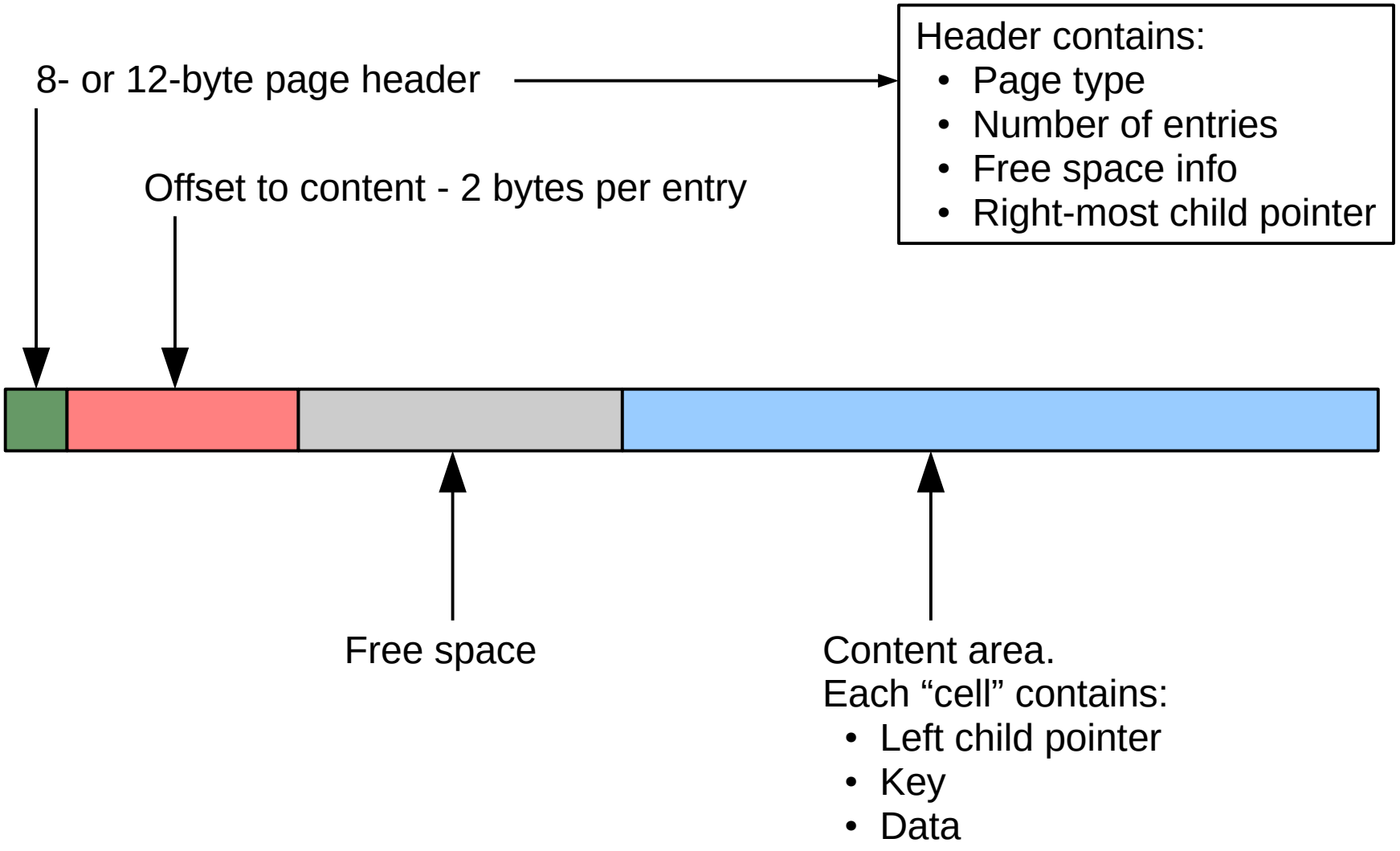
A Better Varint...

A0	Value
0-240	A0
241-248	240+256*(A0-240)+A1
249	2288+256*A1+A2
250	A1..A3 as a 3-byte big-endian integer
251	A1..A4 as a 4-byte big-endian integer
252	A1..A5 as a 5-byte big-endian integer
253	A1..A6 as a 6-byte big-endian integer
254	A1..A7 as a 7-byte big-endian integer
255	A1..A8 as a 8-byte big-endian integer

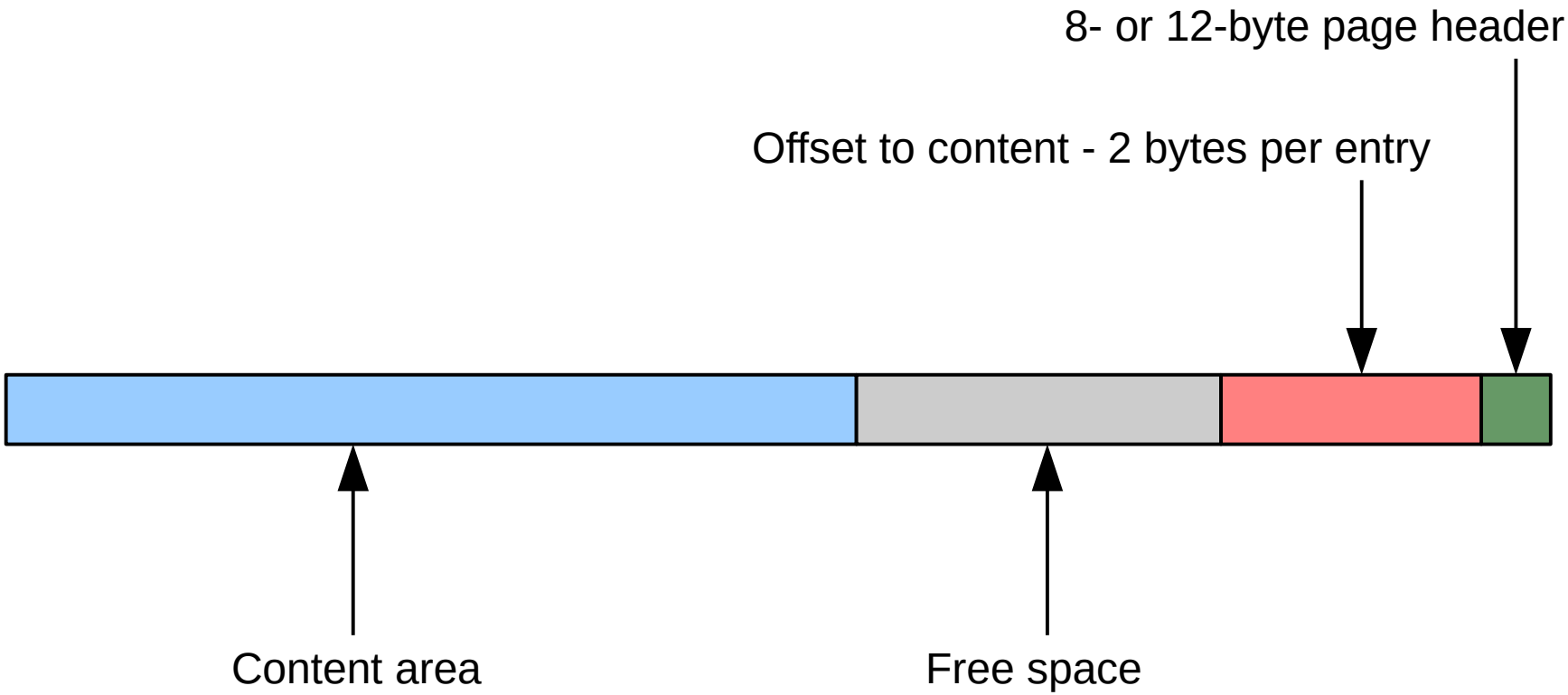
✔ *Size determined by the first byte*

✔ *memcmp() sorts in numeric order*

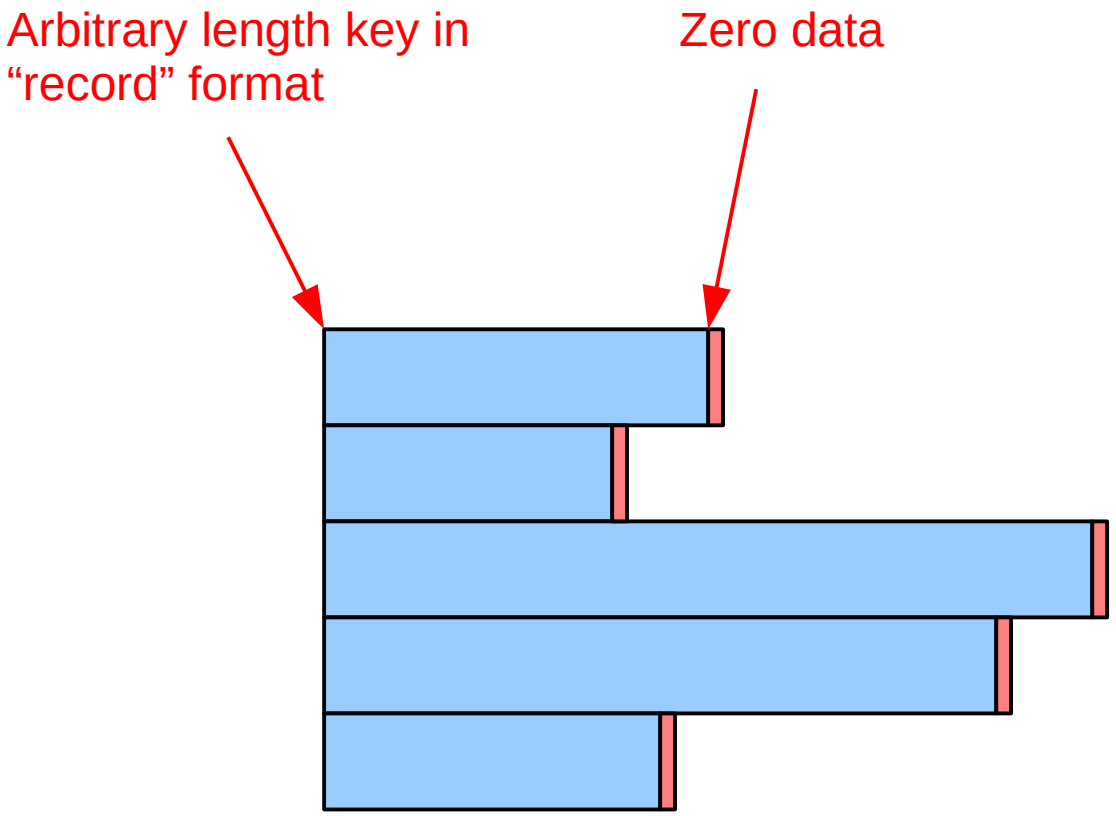
B-tree Page Layout



Better B-tree Page Layout

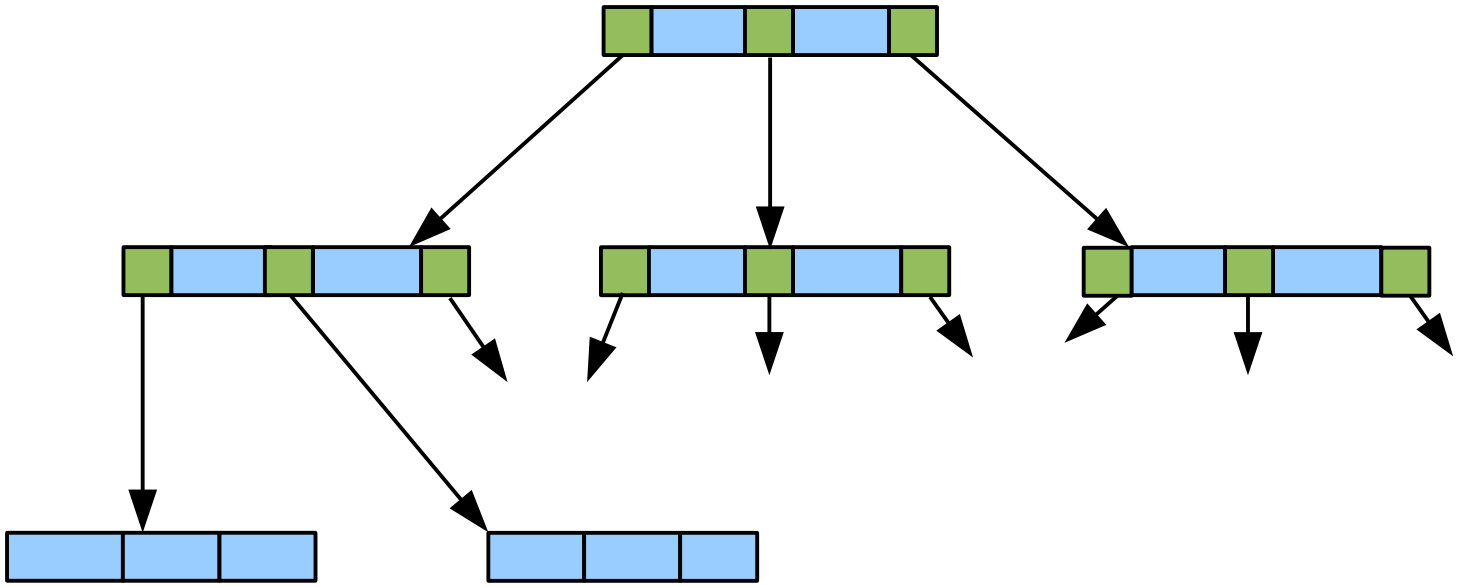


Logical View of SQL Index Storage

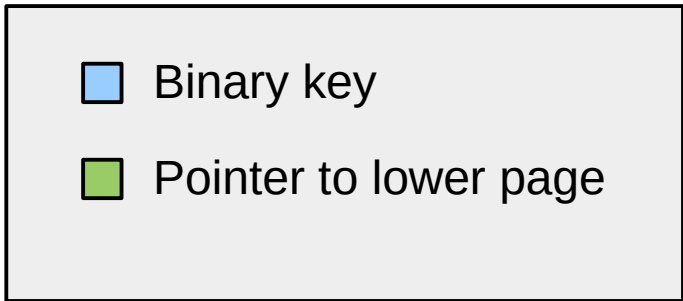


B-tree Structure

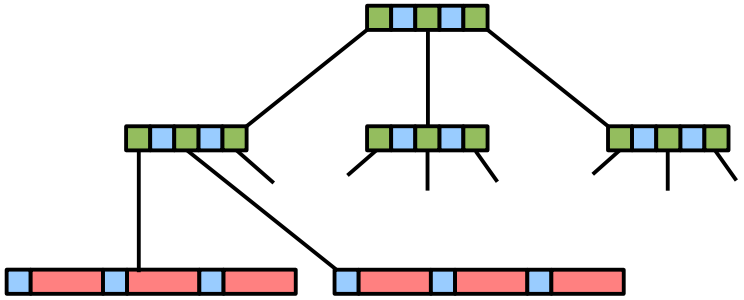
(used by indexes & WITHOUT ROWID tables)



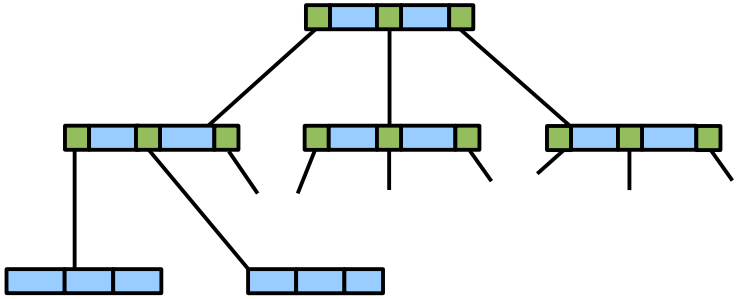
- Key only. No data. The key is the data.
- Larger binary keys, hence lower fan-out
- Each key appears in the table only once
- Minimum 4 keys per page



B-tree Types

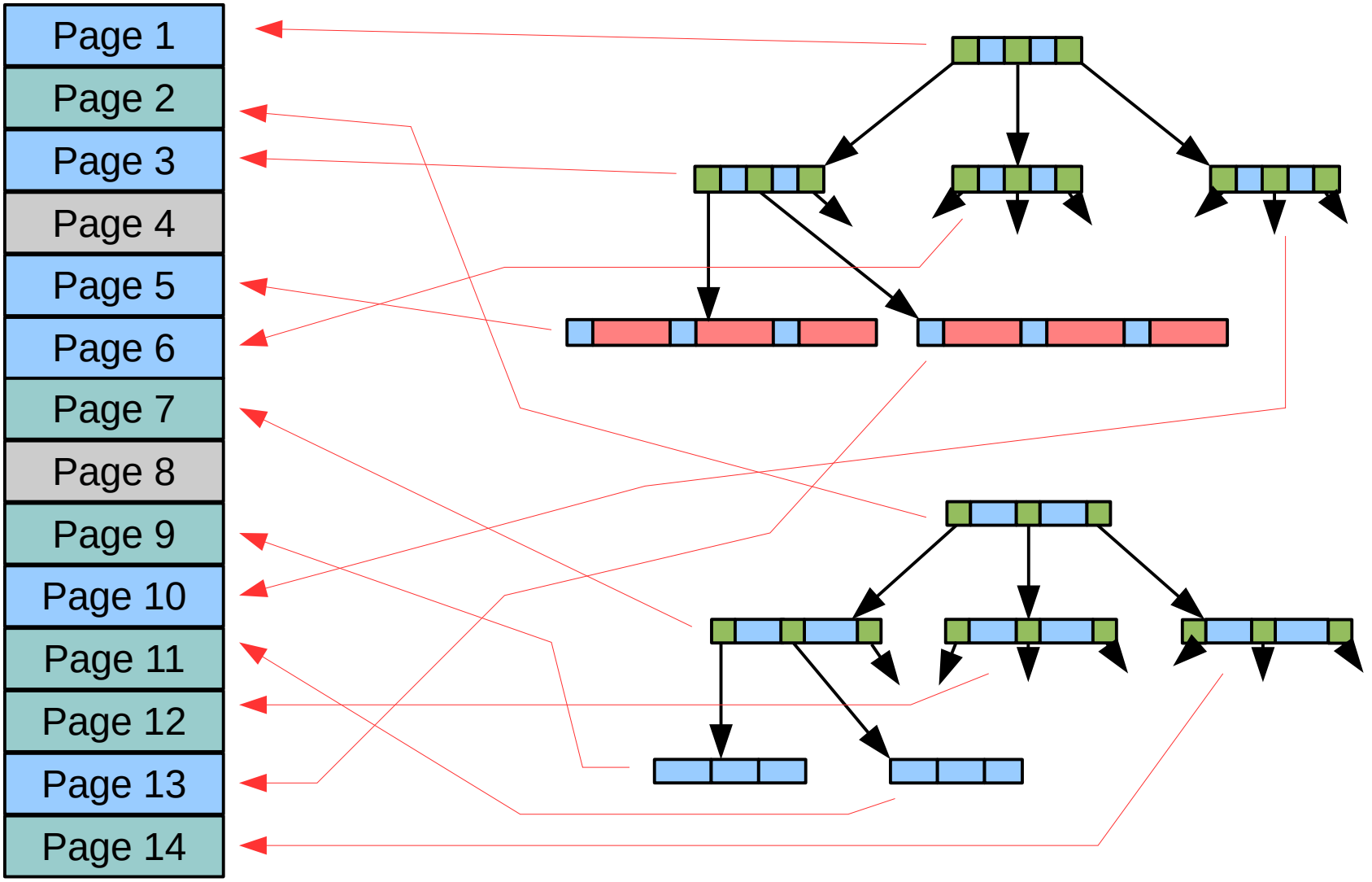


- SQL tables
- Integer keys
- Data in leaves
- Some keys on more than one page



- SQL indexes
- Arbitrary keys
- No data (key=data)
- Keys unique across all pages

Mapping B-trees Into Pages

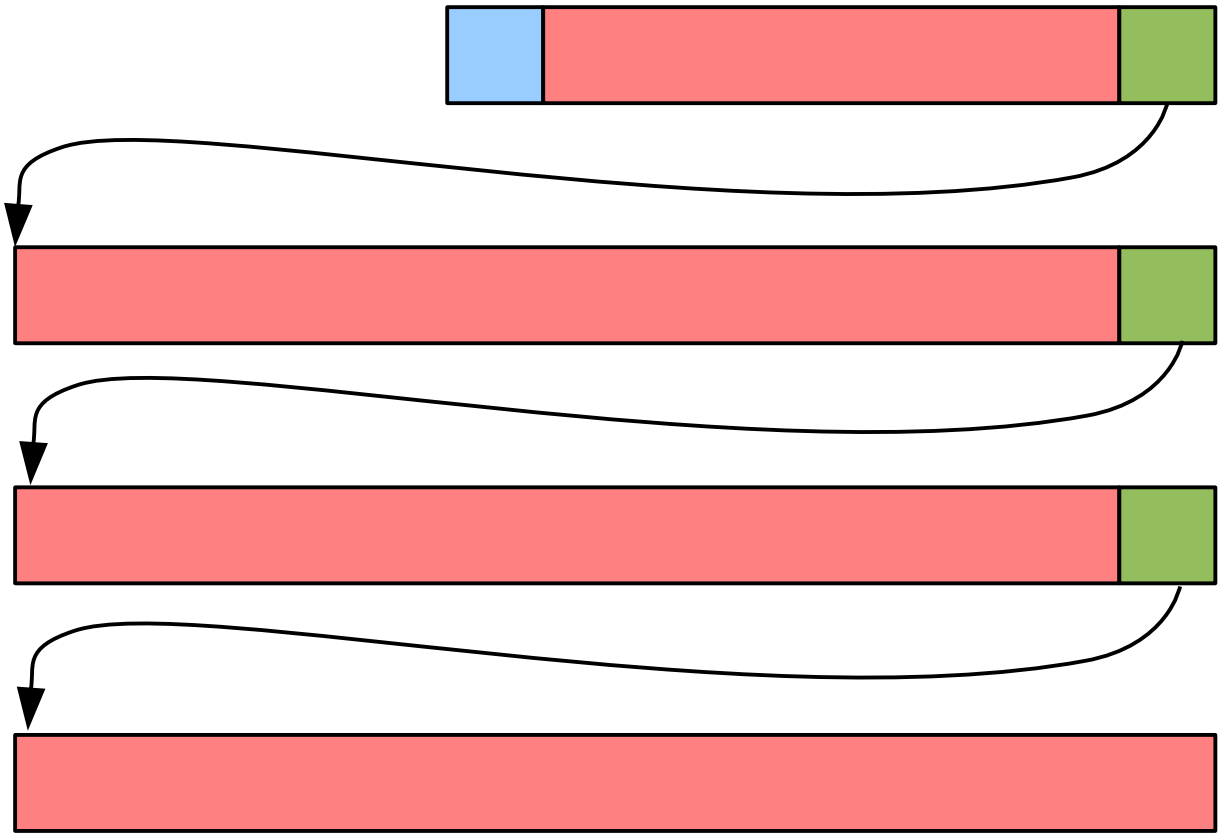


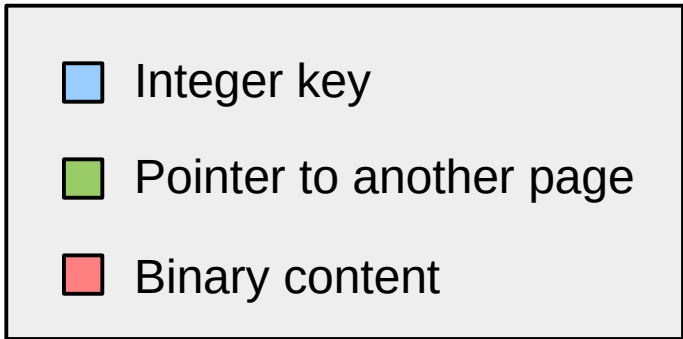
To see how pages are used:

- Download and unpack the source tarball
- ./configure --disable-shared
- make showdb
- ./showdb *database* pgidx

```
Available pages: 1..1146
    1: root leaf of table [sqlite_master]
    2: root interior node of table [blob]
    3: root interior node of index [sqlite_autoindex_blob_1]
    4: root interior node of table [delta]
    5: root interior node of table [rcvfrom]
    6: root leaf of index [sqlite_autoindex_rcvfrom_1]
    7: root leaf of table [config]
    8: root leaf of index [sqlite_autoindex_config_1]
    9: root leaf of table [shun]
   10: root leaf of index [sqlite_autoindex_shun_1]
   11: root leaf of table [private]
…
  264: leaf of table [blob], child 201 of page 2
  265: leaf of table [blob], child 202 of page 2
  266: overflow 1 from cell 0 of page 268
  267: overflow 2 from cell 0 of page 268
  268: leaf of table [blob], child 203 of page 2
…
```

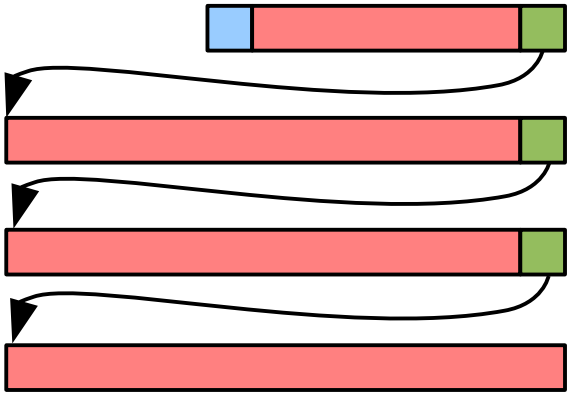
Overflow





Surprising Attributes of Overflow

- Multi-megabyte BLOBs and strings work well.
- Faster to store BLOBs in the database than directly on disk for sizes up to about 100K.



sqlite.org/intern-v-extern-blob.html

Database Page Size	BLOB size						
	10k	20k	50k	100k	200k	500k	1m
1024	1.535	1.020	0.608	0.456	0.330	0.247	0.233
2048	2.004	1.437	0.870	0.636	0.483	0.372	0.340
4096	2.261	1.886	1.173	0.890	0.701	0.526	0.487
8192	2.240	1.866	1.334	1.035	0.830	0.625	0.720
16384	2.439	1.757	1.292	1.023	0.829	0.820	0.598
32768	1.878	1.843	1.296	0.981	0.976	0.675	0.613
65536	1.256	1.255	1.339	0.983	0.769	0.687	0.609

B-tree Primitives

- Open cursor
- Seek
- Next
- Previous
- Key
- Data
- Delete
- Insert
- Close cursor

sqlite_master

```
CREATE TABLE sqlite_master(
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
```

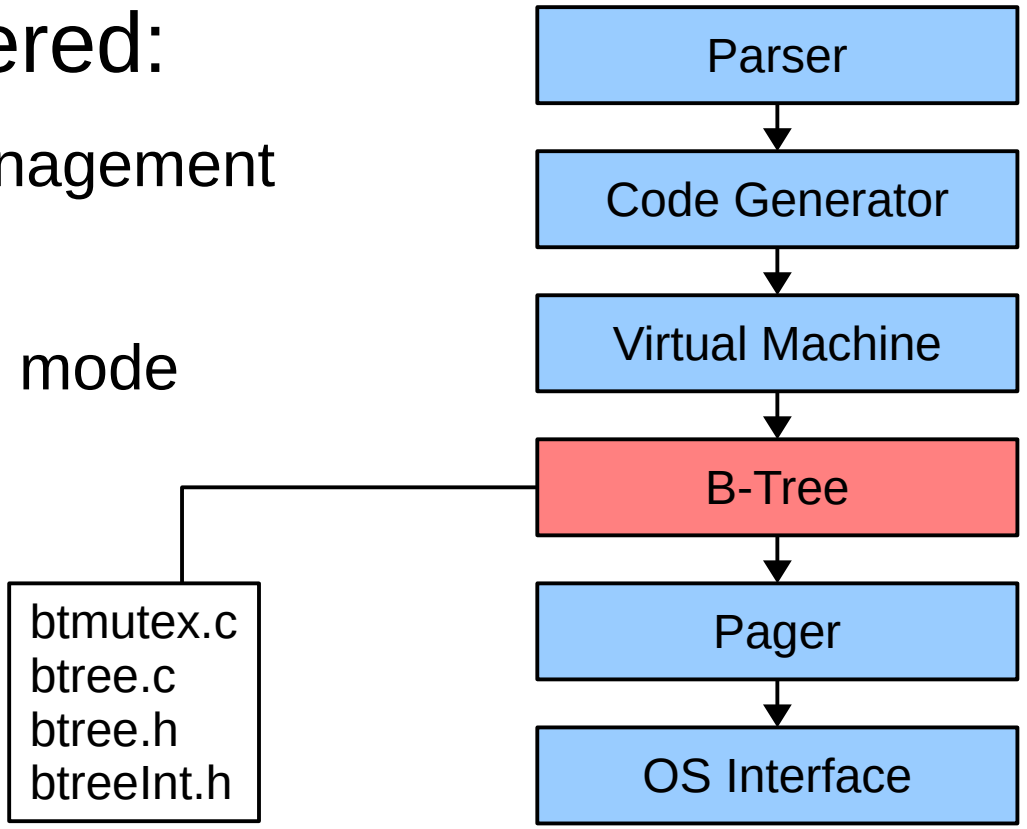
✔ *sqlite_master always rooted at page 1*

sqlite_master

```
sqlite> CREATE TABLE t1(x);
sqlite> .mode line
sqlite> SELECT * FROM sqlite_master;
    type = table
    name = t1
tbl_name = t1
rootpage = 2
     sql = CREATE TABLE t1(x)
```

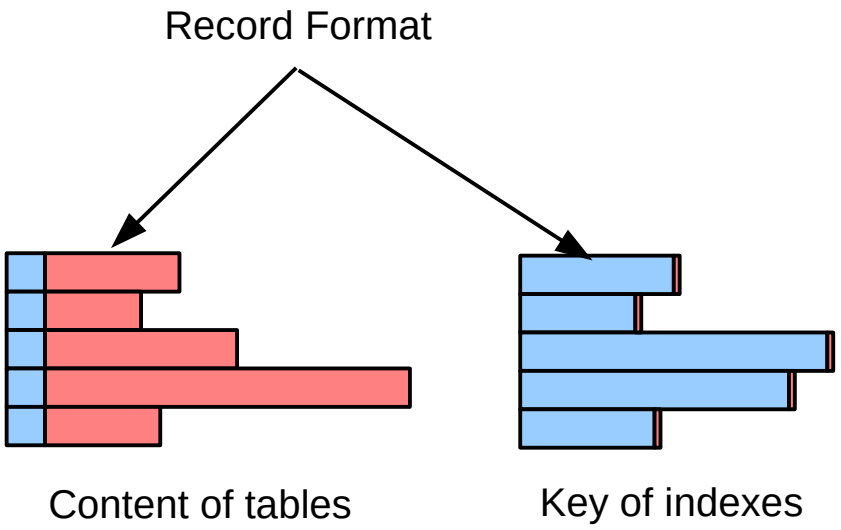
B-tree Summary

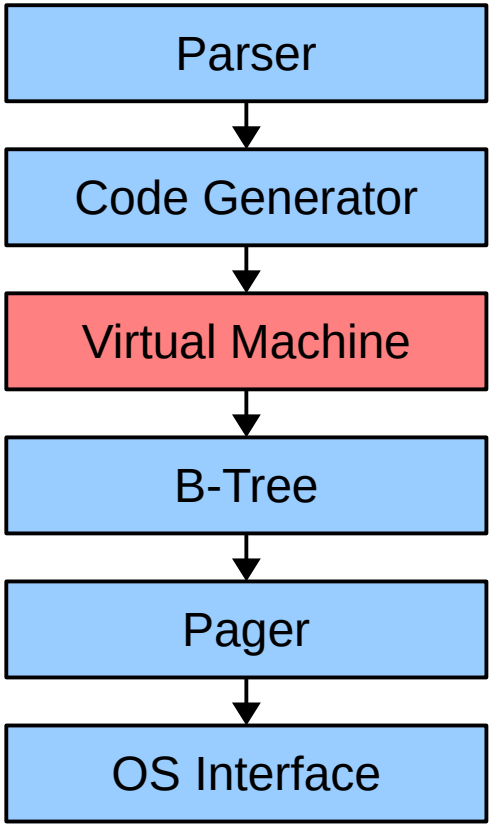
- Topics not covered:
  - free page management
  - auto-vacuum
  - shared-cache mode



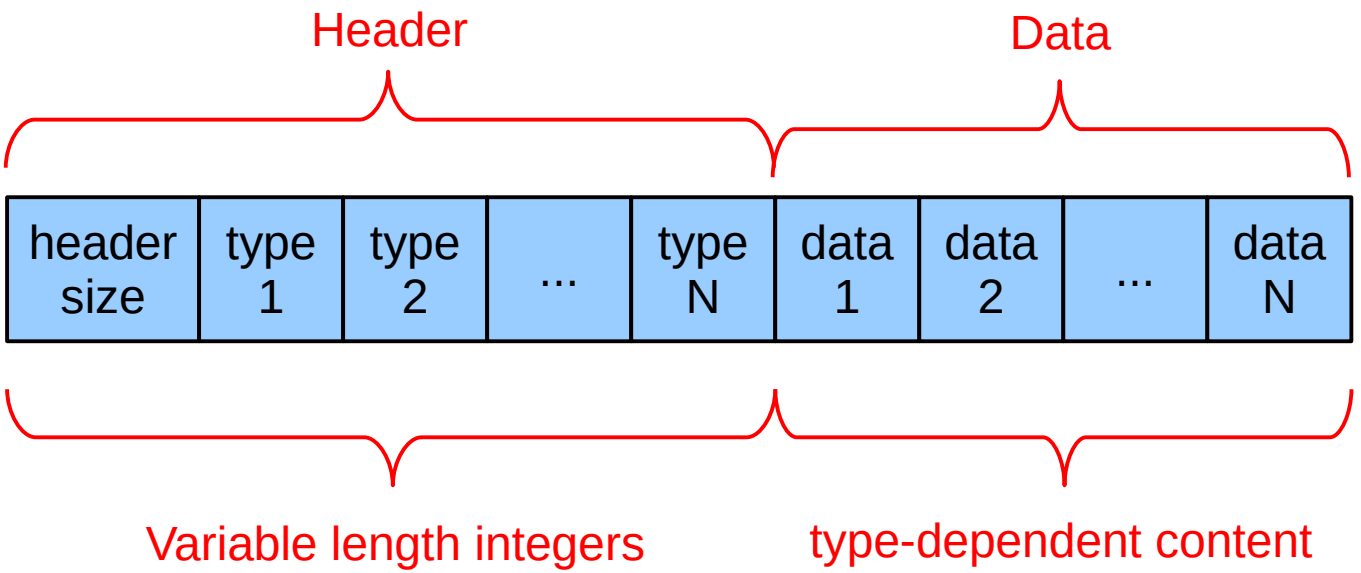
Virtual Machine

- Byte code interpreter
- Defines the “record format”





Record Format

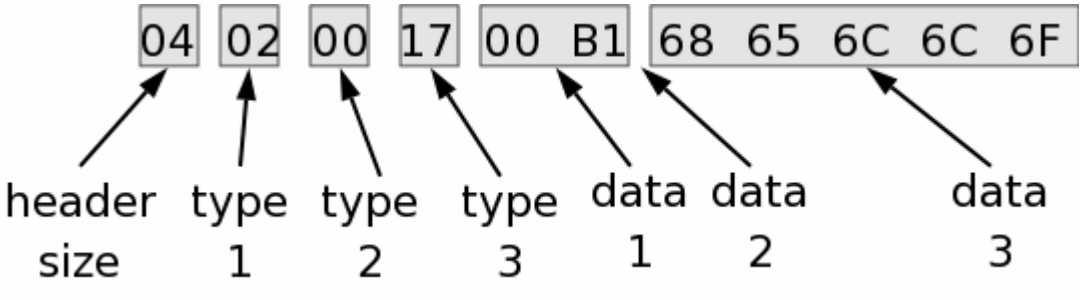


Integer Type Codes

Type	Meaning	Data Length
0	NULL	0
1	signed integer	1
2	signed integer	2
3	signed integer	3
4	signed integer	4
5	signed integer	6
6	signed integer	8
7	IEEE float	8
8	integer zero	0
9	integer one	0
10,11	not used	
N>=12 and even	BLOB	(N-12)/2
N>=13 and odd	string	(N-13)/2

Record Format Example

```
CREATE TABLE t1(a,b,c);
INSERT INTO t1 VALUES(177, NULL, 'hello');
```



✔ *Column datatypes are optional*
✔ *Datatypes are suggestions, not requirements.*

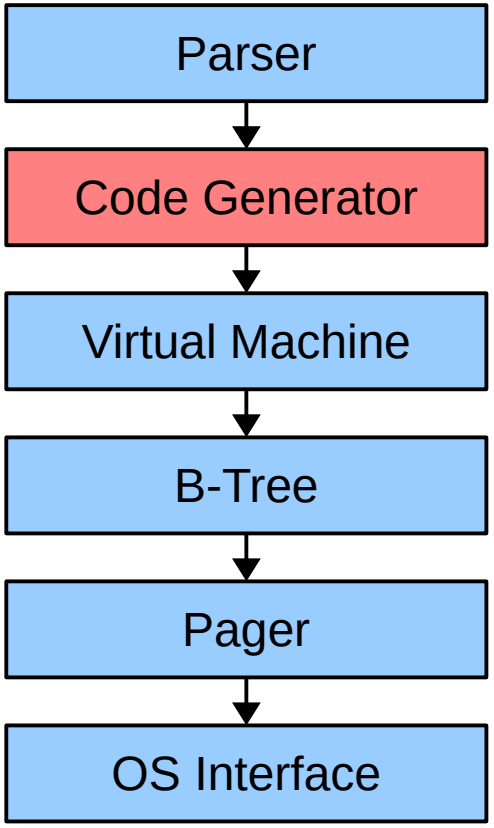
Code Generator

- AST transformations
- Join order determination
- Index selection
- Byte-code generation

select.c

where*.c
whereInt.h

build.c
delete.c
expr.c
insert.c
update.c



Debug-Enhanced Shell

- Download & unpack source tarball; ./configure
- Edit Makefile (or Makefile.msc) to add:
    (a) -DSQLITE_ENABLE_EXPLAIN_COMMENTS
    (b) -DSQLITE_ENABLE_SELECTTRACE
    (c) -DSQLITE_ENABLE_WHERETRACE
    (d) -DSQLITE_DEBUG
- “make” or “nmake /f Makefile.msc”
- Command-line shell now supports:
    (a) .selecttrace 0xfff
    (b) .wheretrace 0xfff

SELECT name, rootpage+10 FROM sqlite_master WHERE substr(type,1,1)='t'

```
#1.26D0380: after name resolution:
'-- SELECT (0x26D0380) selFlags=0x64
    |-- result-set
    |   |-- {0:1}  flags=0x20004
    |   '-- ADD  flags=0x4
    |       |-- {0:3}  flags=0x20004
    |       '-- 10
    |-- FROM
    |   '-- {0,*} sqlite_master tabname='sqlite_master'
    '-- WHERE
        '-- EQ  flags=0x4
            |-- FUNCTION 'substr'
            |   '-- LIST
            |       |-- {0:0}  flags=0x20004
            |       |-- 1
            |       '-- 1
            '-- 't'
```

Viewing The AST In A Debugger

Three main objects in the abstract syntax tree:

1. **Expr** - an expression
2. **ExprList** - a list of expressions
3. **Select** - a single SELECT statement

sqliteInt.h

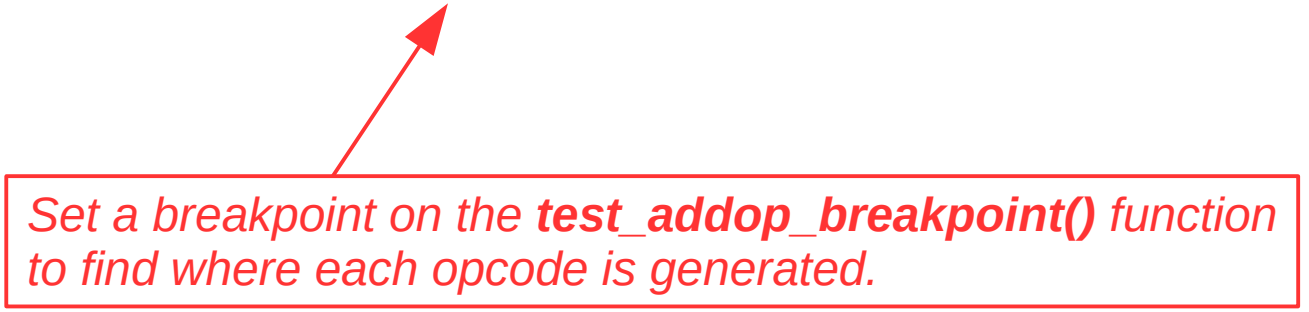
To view a syntax tree as ASCII-art:

```
(gdb) print sqlite3TreeViewExpr(0, pExpr, 0)
(gdb) print sqlite3TreeViewExprList(0, pList, 0, 0)
(gdb) print sqlite3TreeViewSelect(0, pSelect, 0)
```

✔ *Only available with -DSQLITE_DEBUG*

Debug-Enhanced Shell

- Debugging PRAGMAs:
  - (a) PRAGMA vdbe_trace=ON;
  - (b) PRAGMA vdbe_debug=ON;
  - (c) PRAGMA vdbe_addoptrace=ON;

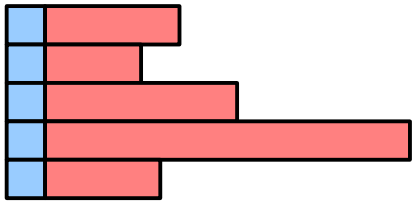


✔ *Only available with -DSQLITE_DEBUG*

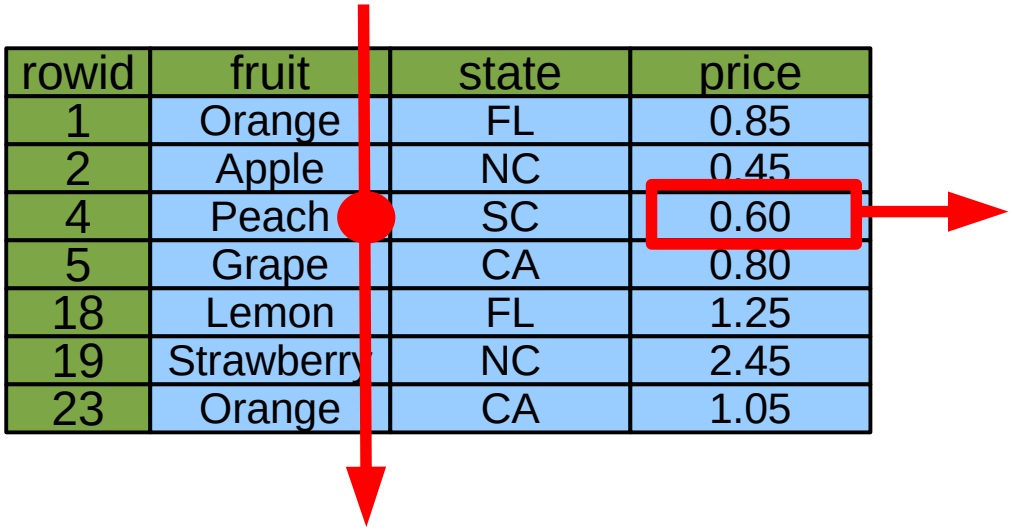
```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL
);
```

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Key **Data**

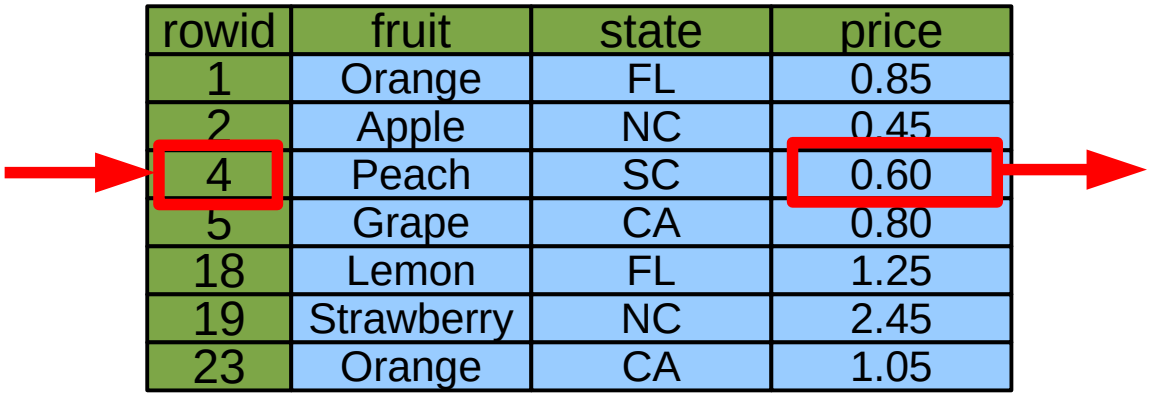


SELECT price FROM tab WHERE fruit='Peach'



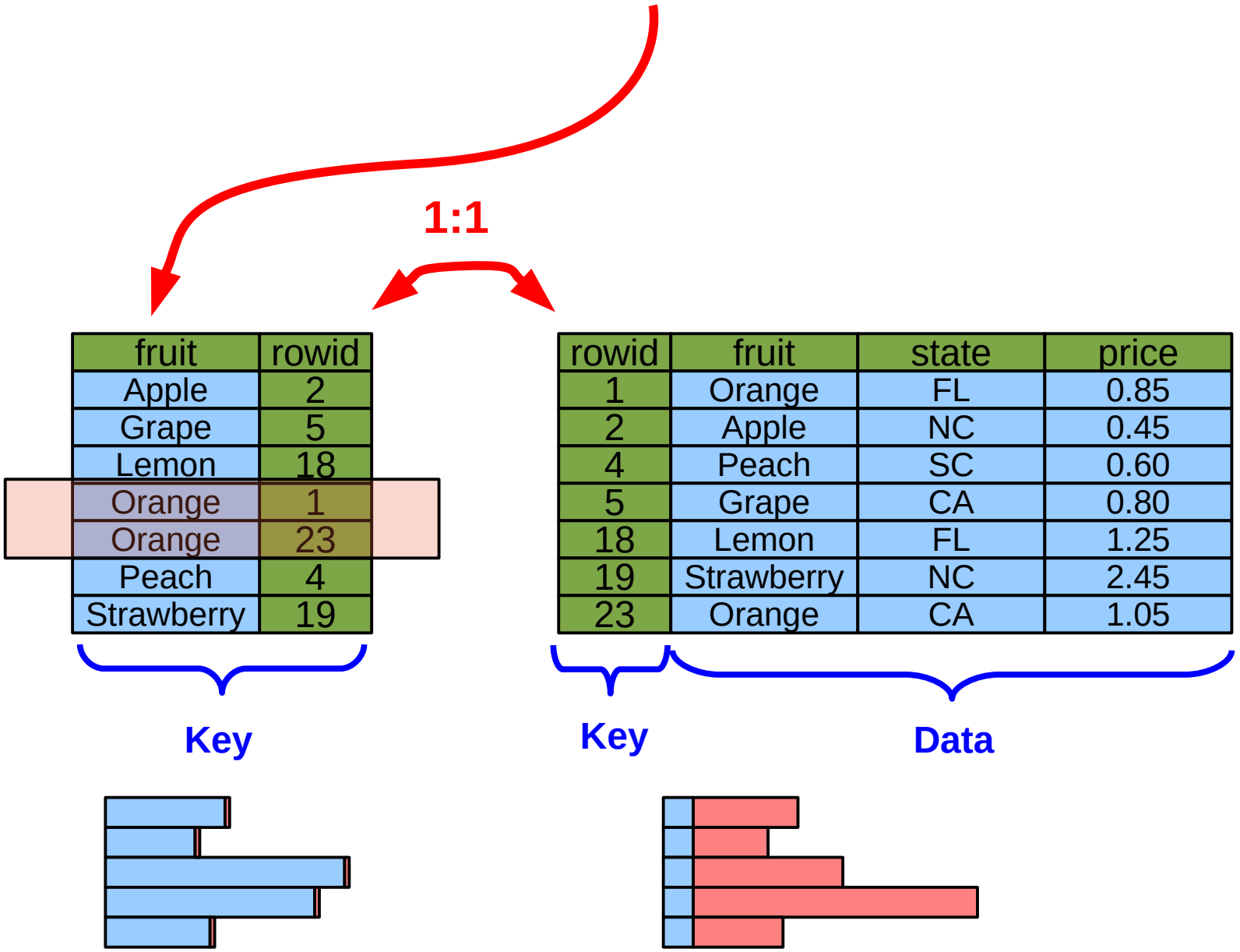
rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

SELECT price FROM tab WHERE rowid=4

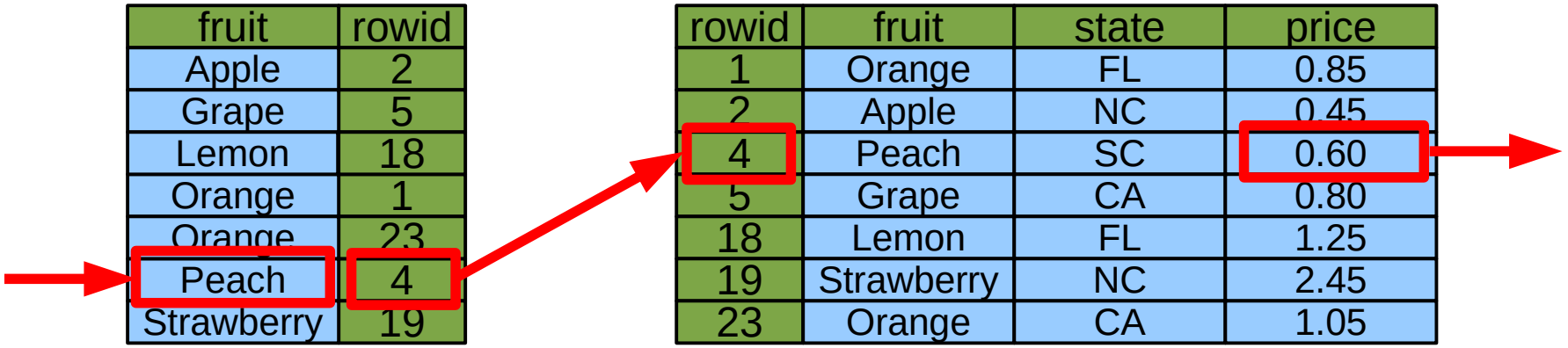


rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

CREATE INDEX idx1 ON tab(fruit)



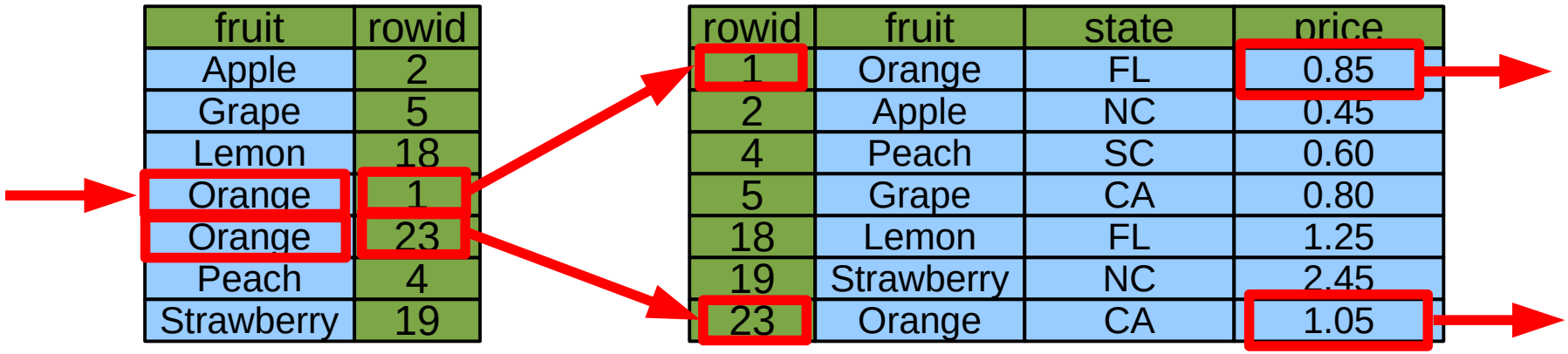
SELECT price FROM tab WHERE fruit='Peach'



fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

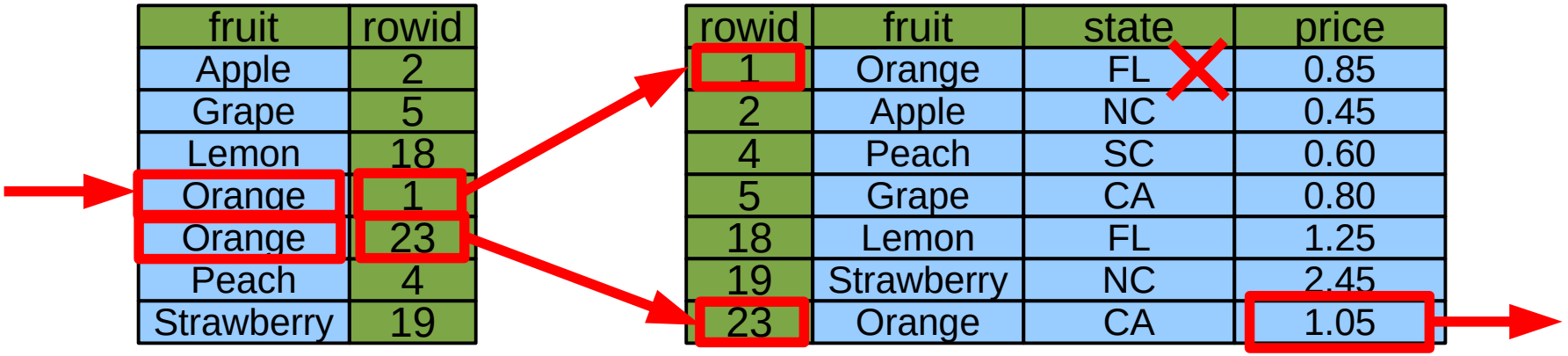
SELECT price FROM tab
WHERE fruit='Orange'



fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

SELECT price FROM tab
WHERE fruit='Orange'
AND state='CA'

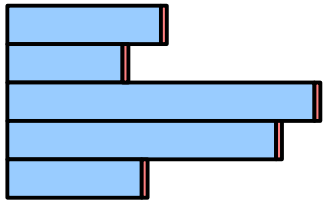


CREATE INDEX idx2 ON tab(state)

idx2

state	rowid
CA	5
CA	23
FL	1
FL	18
NC	2
NC	19
SC	4

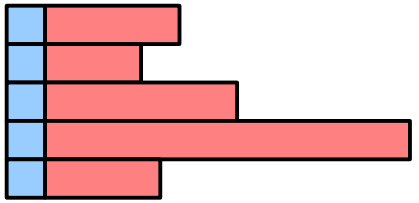
Key



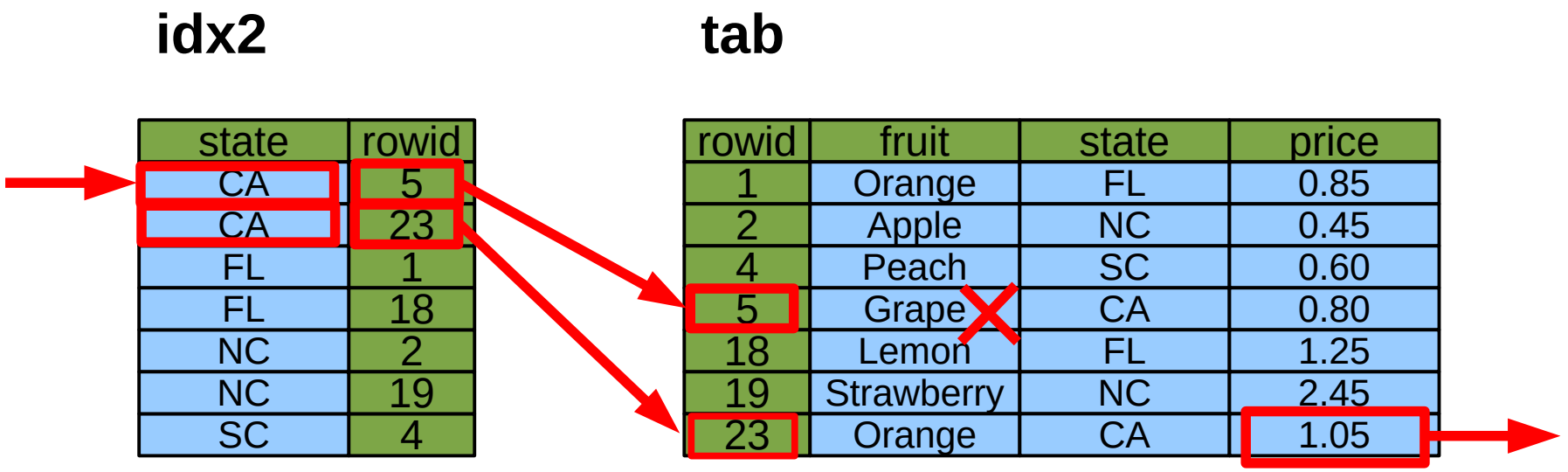
tab

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

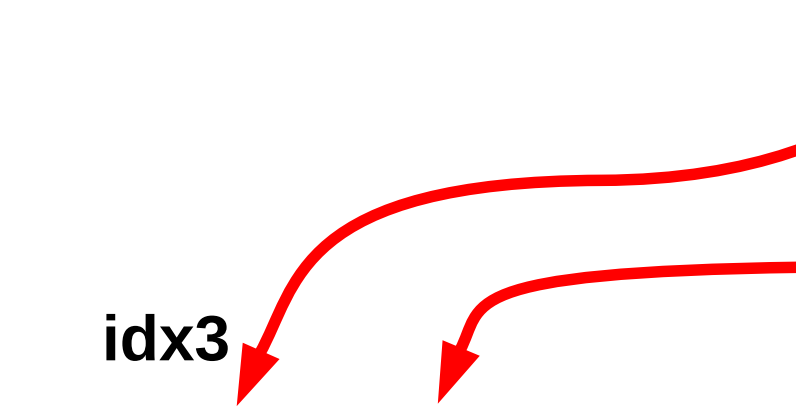
Key Data



```
SELECT price FROM tab
 WHERE fruit='Orange'
   AND state='CA'
```



CREATE INDEX idx3 ON tab(fruit, state)



idx3

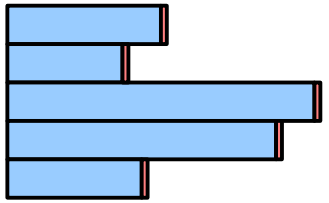
fruit	state	rowid
Apple	NC	2
Grape	CA	5
Lemon	FL	18
Orange	CA	23
Orange	FL	1
Peach	SC	4
Strawberry	NC	19

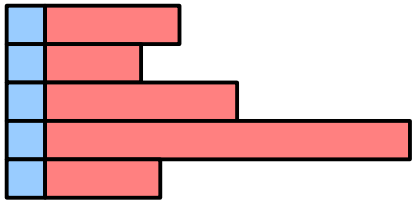
Key

tab

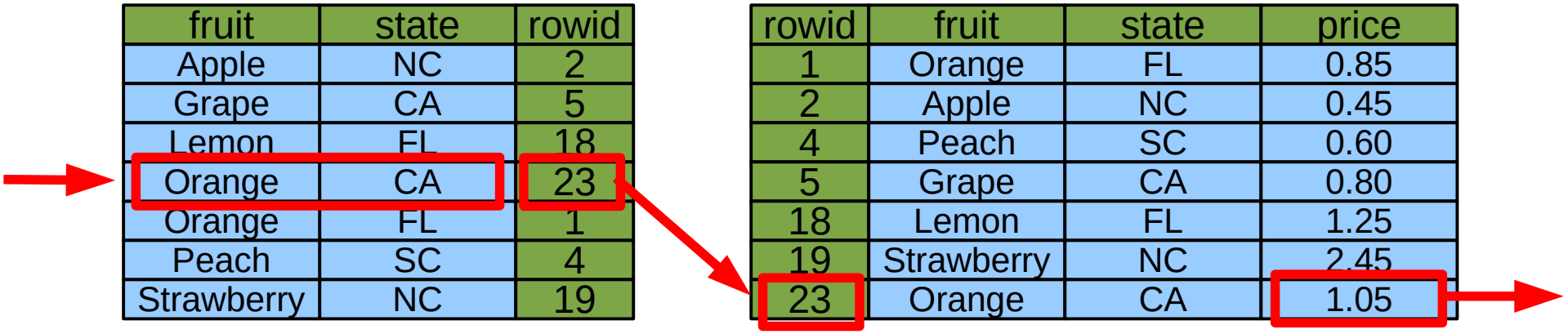
rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Key **Data**





```
SELECT price FROM tab
 WHERE fruit='Orange'
   AND state='CA'
```



fruit	state	rowid
Apple	NC	2
Grape	CA	5
Lemon	FL	18
Orange	CA	23
Orange	FL	1
Peach	SC	4
Strawberry	NC	19

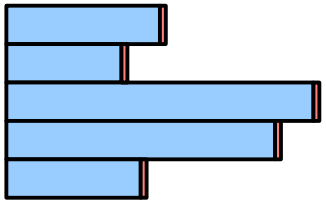
rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

CREATE INDEX idx5 ON tab(fruit, state, price)

idx5

fruit	state	price	rowid
Apple	NC	0.45	2
Grape	CA	0.80	5
Lemon	FL	1.25	18
Orange	CA	1.05	23
Orange	FL	0.85	1
Peach	SC	0.60	4
Strawberry	NC	2.45	19

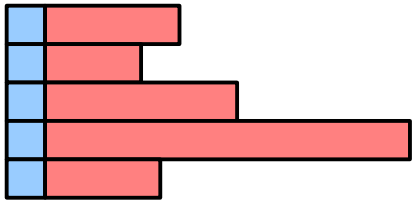
Key



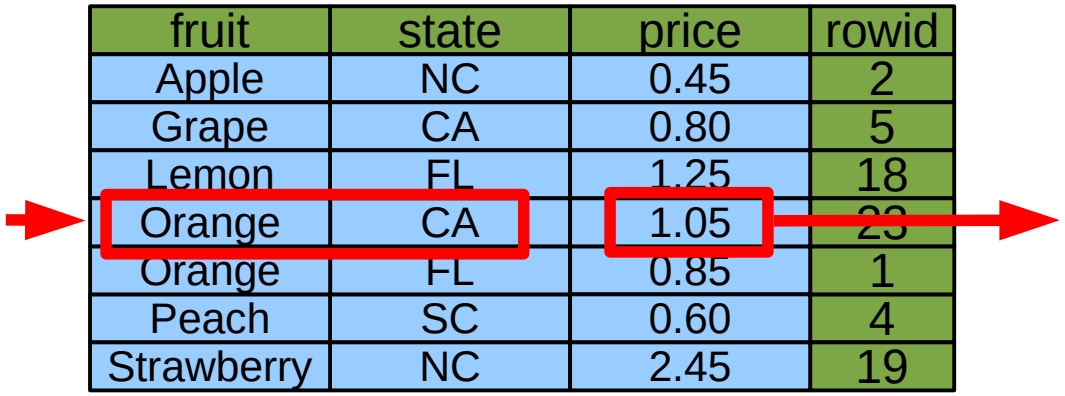
tab

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Key Data

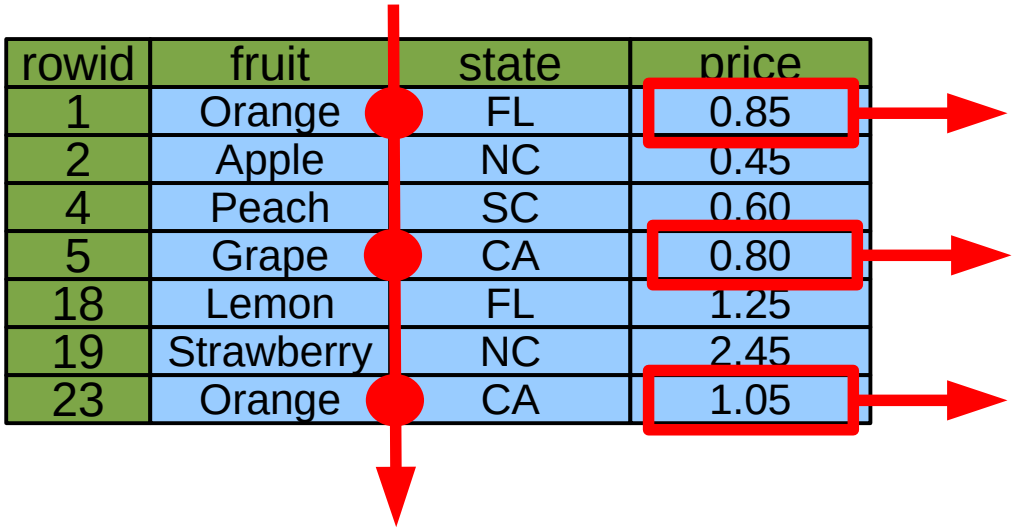


SELECT price FROM tab
WHERE fruit='Orange'
AND state='CA'



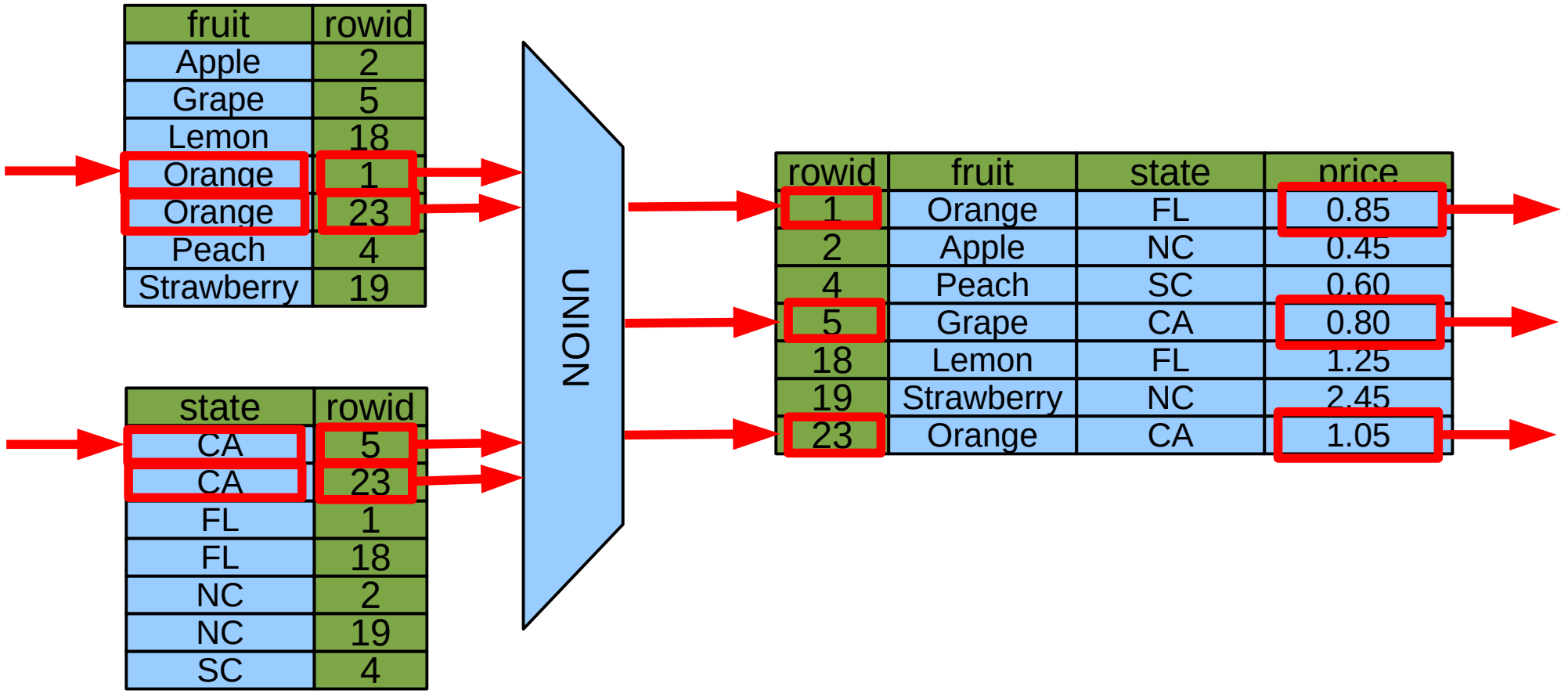
fruit	state	price	rowid
Apple	NC	0.45	2
Grape	CA	0.80	5
Lemon	FL	1.25	18
Orange	CA	1.05	23
Orange	FL	0.85	1
Peach	SC	0.60	4
Strawberry	NC	2.45	19

```
SELECT price FROM tab
WHERE fruit='Orange'
   OR state='CA'
```



rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

SELECT price FROM tab
WHERE fruit='Orange'
OR state='CA'

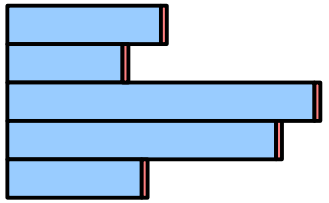


CREATE INDEX idx4 ON tab(state,fruit)

idx4

state	fruit	rowid
CA	Grape	5
CA	Orange	23
FL	Orange	1
FL	Lemon	18
NC	Apple	2
NC	Strawberry	19
SC	Peach	4

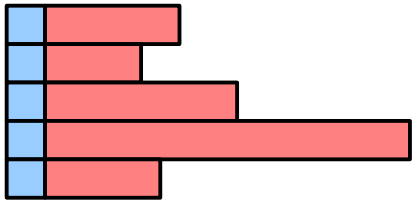
Key



tab

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Key Data



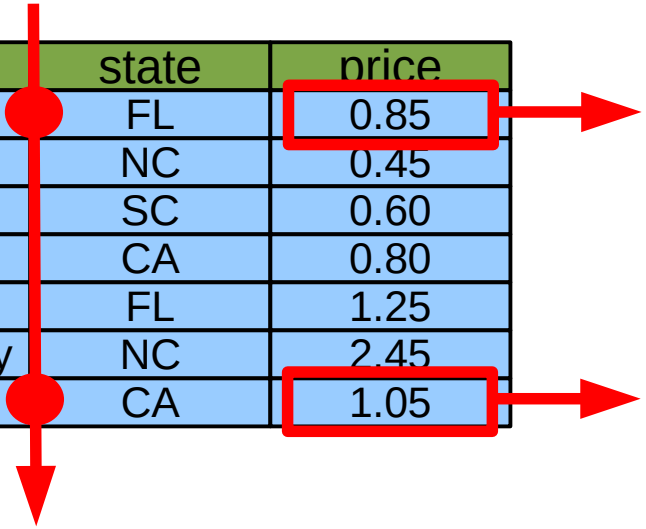
SELECT price FROM tab
WHERE fruit='Orange'

idx4

state	fruit	rowid
CA	Grape	5
CA	Orange	23
FL	Orange	1
FL	Lemon	18
NC	Apple	2
NC	Strawberry	19
SC	Peach	4

tab

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05



SELECT price FROM tab
WHERE fruit='Orange'

Added by the planner

AND state IN (SELECT state FROM tab)

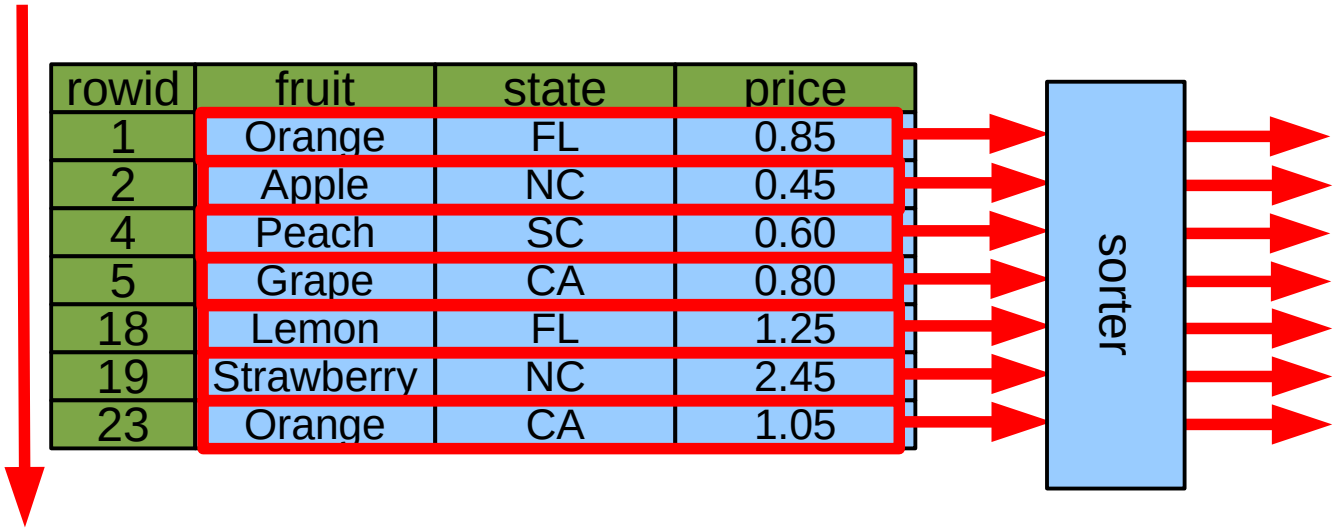
idx4

state	fruit	rowid
CA	Grape	5
CA	Orange	23
FL	Orange	1
FL	Lemon	18
NC	Apple	2
NC	Strawberry	19
SC	Peach	4

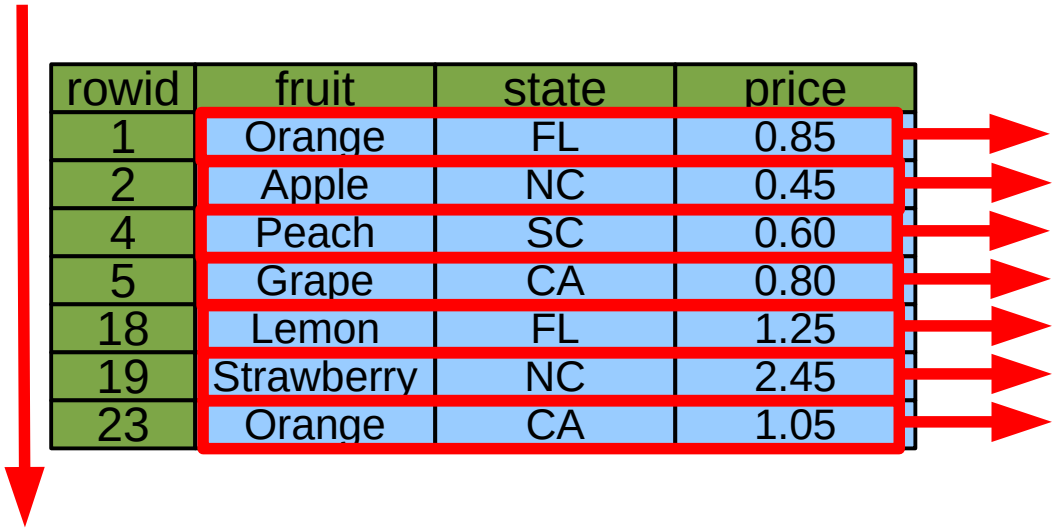
tab

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

SELECT * FROM tab ORDER BY fruit

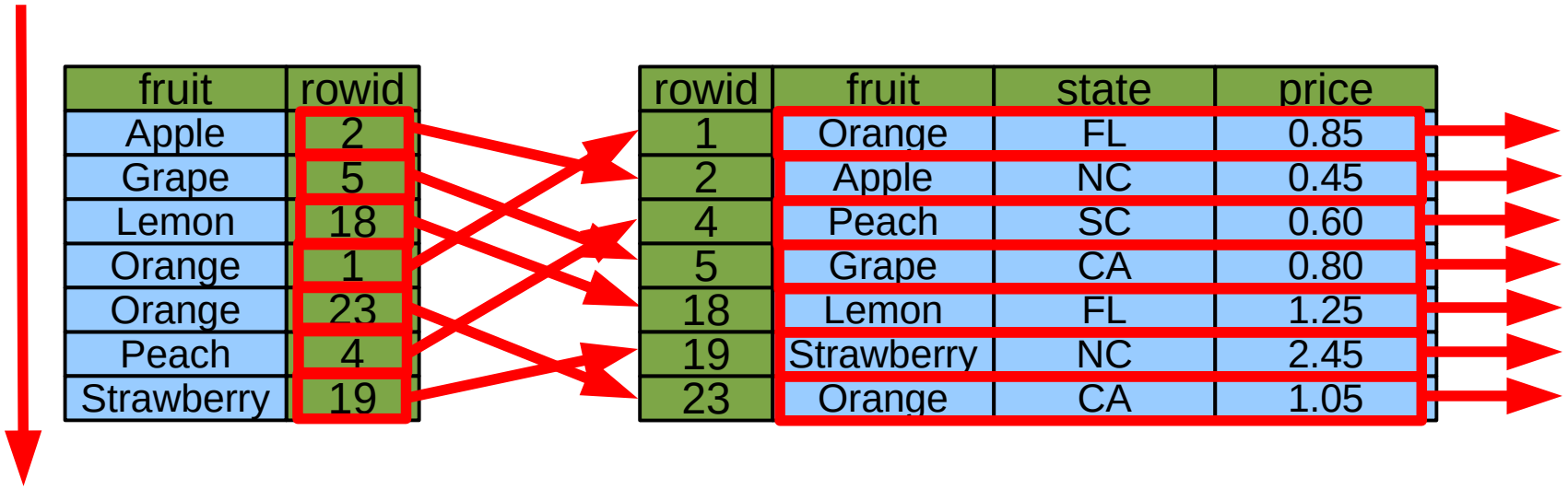


SELECT * FROM tab ORDER BY rowid

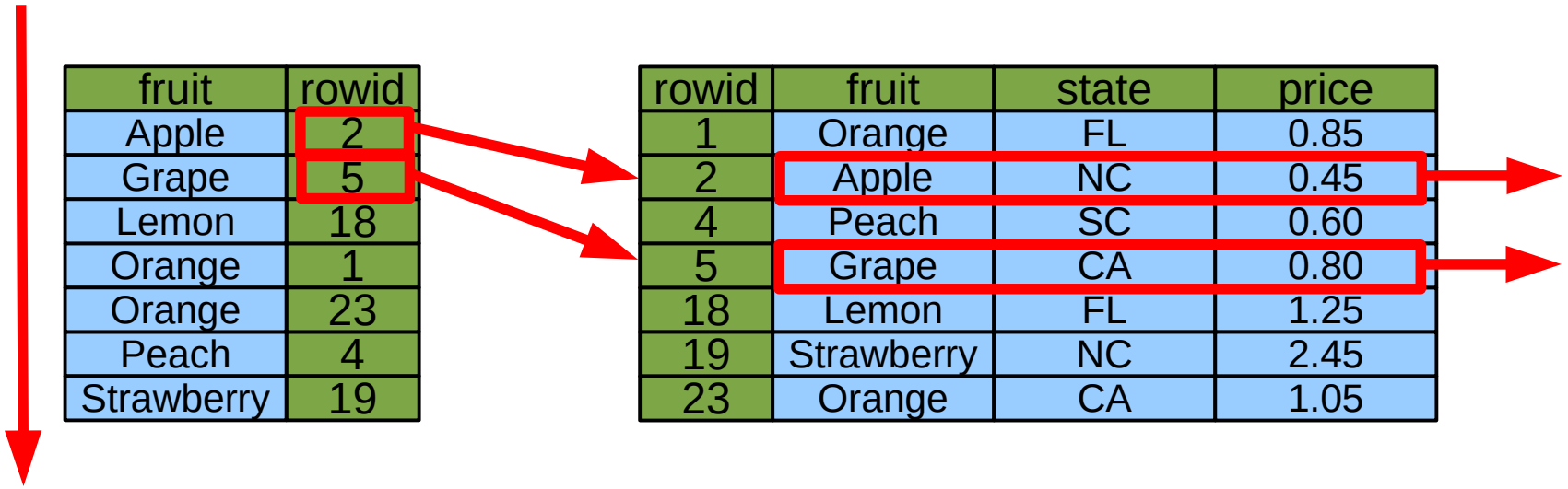


rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

SELECT * FROM tab ORDER BY fruit



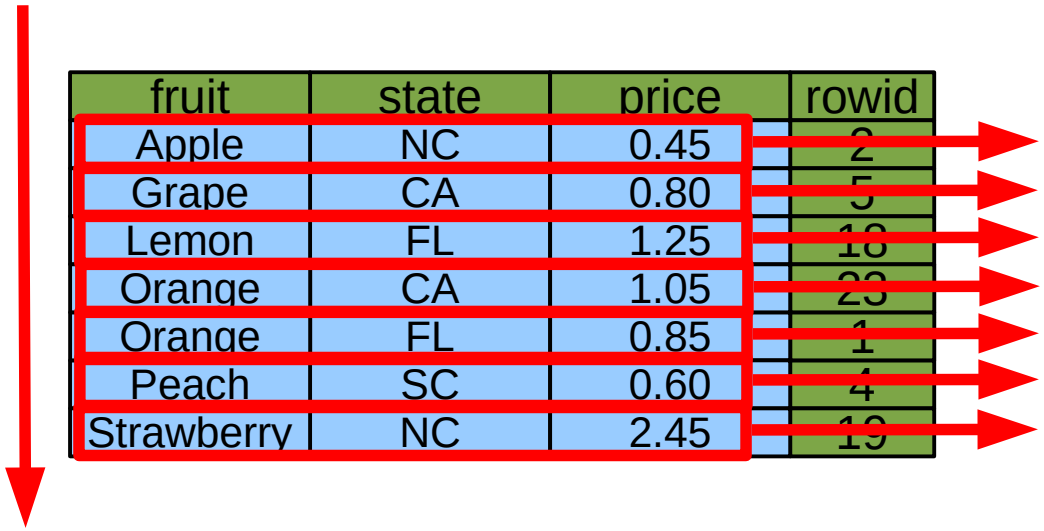
SELECT * FROM tab ORDER BY fruit LIMIT 2



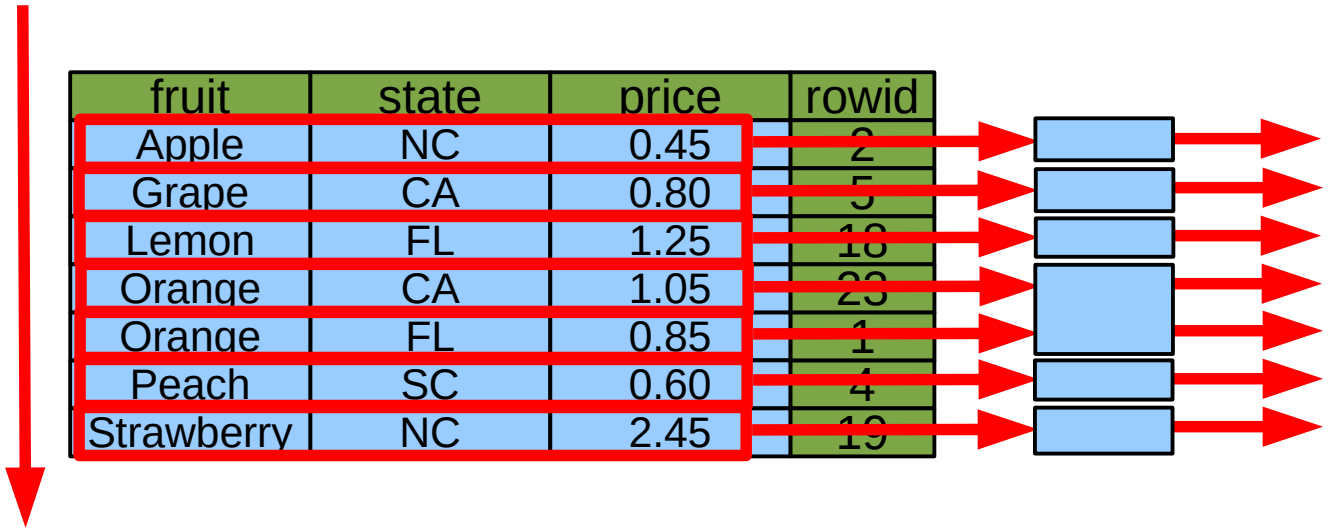
fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

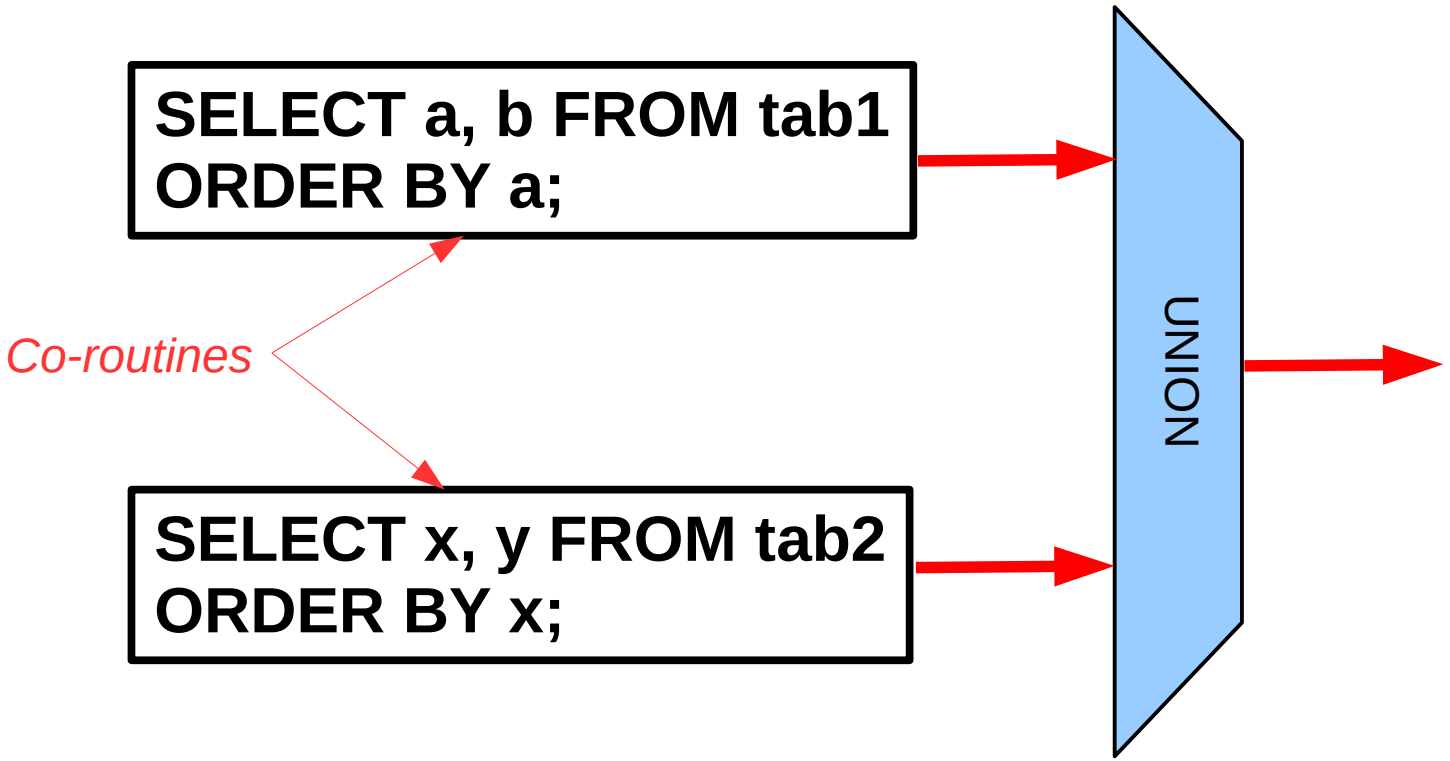
SELECT * FROM tab ORDER BY fruit



SELECT * FROM tab ORDER BY fruit, price

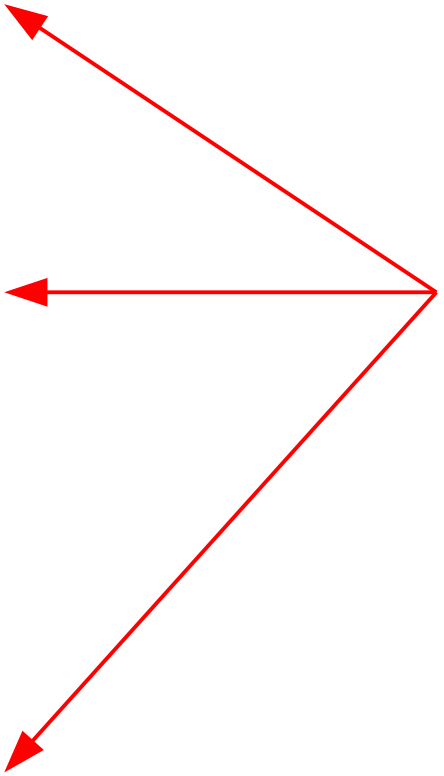


```
SELECT a, b FROM tab1
UNION
SELECT x, y FROM tab2
ORDER BY a;
```



```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL,
   PRIMARY KEY(Fruit, State)
);
```

```
 CREATE TABLE tab(
    Fruit TEXT,
    State TEXT,
    Price REAL,
    UNIQUE(Fruit, State)
 );
```



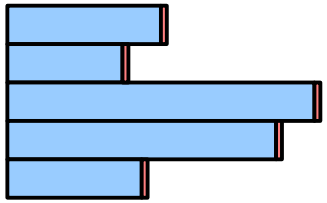
All mean the same thing (to SQLite)

```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL
 );
CREATE UNIQUE INDEX idx ON tab(Fruit,State);
```

```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL,
   PRIMARY KEY(Fruit, State)
) WITHOUT ROWID;
```

fruit	state	price
Apple	NC	0.45
Grape	CA	0.80
Lemon	FL	1.25
Orange	CA	1.05
Orange	FL	0.85
Peach	SC	0.60
Strawberry	NC	2.45

Key **Data**



```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL,
   PRIMARY KEY(Fruit, State)
);
```

With Rowid:

fruit	state	rowid
Apple	NC	2
Grape	CA	5
Lemon	FL	18
Orange	CA	23
Orange	FL	1
Peach	SC	4
Strawberry	NC	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

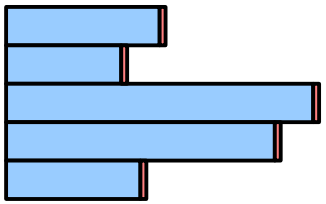
Without Rowid:

fruit	state	price
Apple	NC	0.45
Grape	CA	0.80
Lemon	FL	1.25
Orange	CA	1.05
Orange	FL	0.85
Peach	SC	0.60
Strawberry	NC	2.45

```
CREATE TABLE tab(
   Fruit TEXT,
   State TEXT,
   Price REAL,
   Amt INT,
   PRIMARY KEY(Fruit, State)
) WITHOUT ROWID;
CREATE INDEX tab_amt ON tab(amt);
```

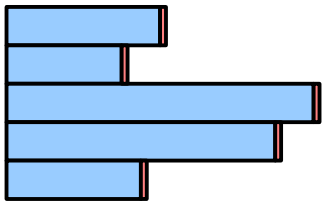
amt	fruit	state
138	Peach	SC
200	Grape	CA
375	Orange	FL
750	Lemon	FL
825	Orange	CA
980	Strawberry	NC
1000	Apple	NC

Key



fruit	state	price	amt
Apple	NC	0.45	1000
Grape	CA	0.80	200
Lemon	FL	1.25	750
Orange	CA	1.05	825
Orange	FL	0.85	375
Peach	SC	0.60	138
Strawberry	NC	2.45	980

Key Data



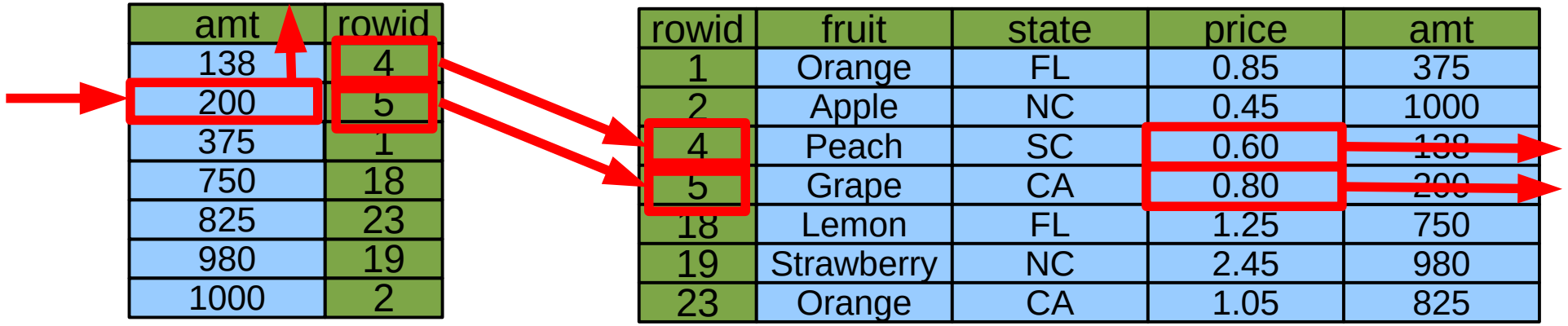
SELECT price FROM tab WHERE fruit='Peach';

fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price	amt
1	Orange	FL	0.85	375
2	Apple	NC	0.45	1000
4	Peach	SC	0.60	138
5	Grape	CA	0.80	200
18	Lemon	FL	1.25	750
19	Strawberry	NC	2.45	980
23	Orange	CA	1.05	825

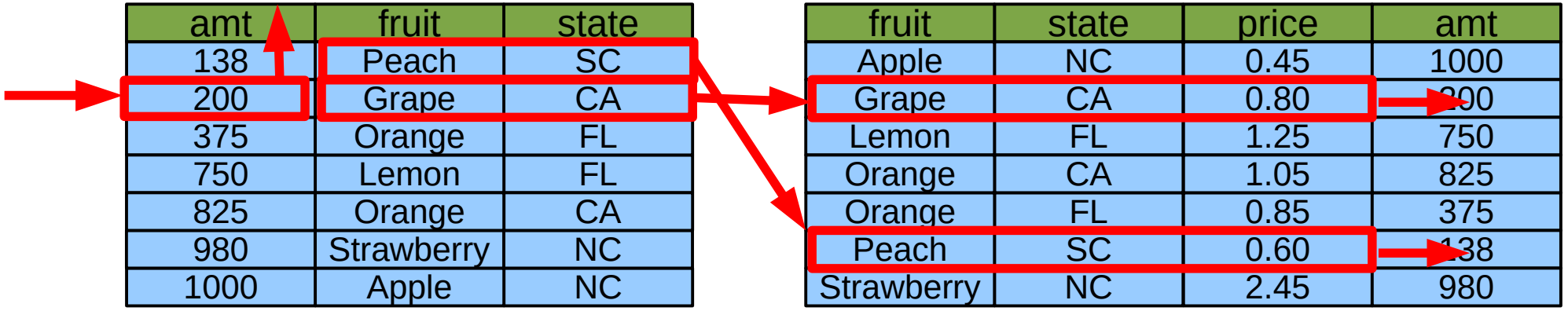
fruit	state	price	amt
Apple	NC	0.45	1000
Grape	CA	0.80	200
Lemon	FL	1.25	750
Orange	CA	1.05	825
Orange	FL	0.85	375
Peach	SC	0.60	138
Strawberry	NC	2.45	980

SELECT fruit, state, price FROM tab WHERE amt<250;



amt	rowid
138	4
200	5
375	1
750	18
825	23
980	19
1000	2

rowid	fruit	state	price	amt
1	Orange	FL	0.85	375
2	Apple	NC	0.45	1000
4	Peach	SC	0.60	138
5	Grape	CA	0.80	200
18	Lemon	FL	1.25	750
19	Strawberry	NC	2.45	980
23	Orange	CA	1.05	825

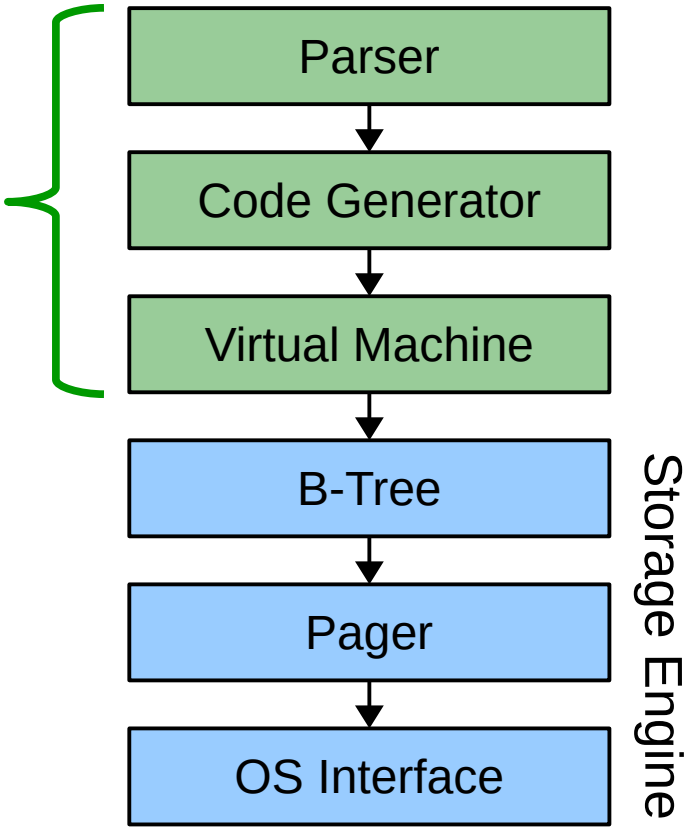


amt	fruit	state
138	Peach	SC
200	Grape	CA
375	Orange	FL
750	Lemon	FL
825	Orange	CA
980	Strawberry	NC
1000	Apple	NC

fruit	state	price	amt
Apple	NC	0.45	1000
Grape	CA	0.80	200
Lemon	FL	1.25	750
Orange	CA	1.05	825
Orange	FL	0.85	375
Peach	SC	0.60	138
Strawberry	NC	2.45	980

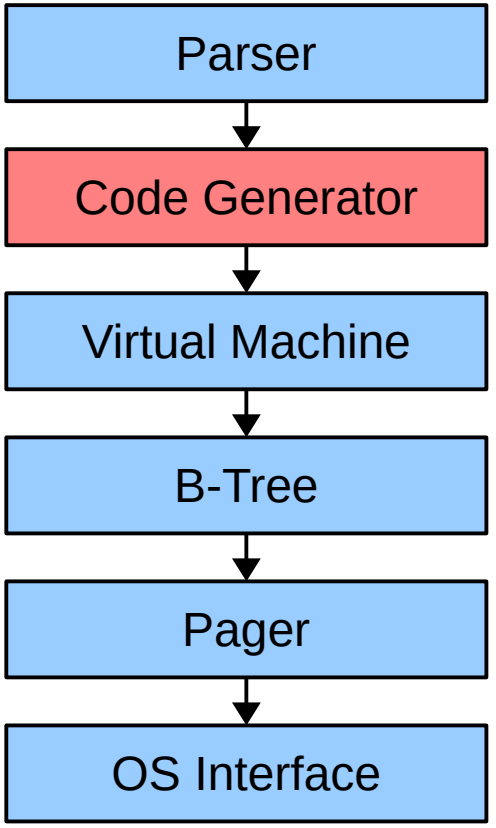
The Importance of a Query Language

- Well-defined schema
- Automatic index maintenance
- Automatic algorithm selection
- Fix performance problems using CREATE INDEX or WITHOUT ROWID - no recoding
- “Push query semantics down into the database engine and let it figure out what to do”



Code Generator Summary

- Topics not covered include:
  - Join order selection
  - Convert subqueries and IN operators into joins
  - LEFT JOIN
  - LIKE operator optimizations
  - Automatic indexes
  - ANALYZE
  - The “Query Planner Stability Guarantee”



Other Topics Not Covered

- Virtual tables
- Full-Text Search, and the implementation of LSM trees on top of B+Trees.
- R-Tree indexes
- Memory management
- Testing and validation
- ... and so forth

These slides: https://www.sqlite.org/talks/cmu-20150917.odp

Website: https://www.sqlite.org/
http://www2.sqlite.org/
http://www3.sqlite.org/

Sources: https://www.sqlite.org/src/timeline
http://www2.sqlite.org/src/timeline
http://www3.sqlite.org/cgi/src/timeline

Mailing list: sqlite-users@mailinglists.sqlite.org