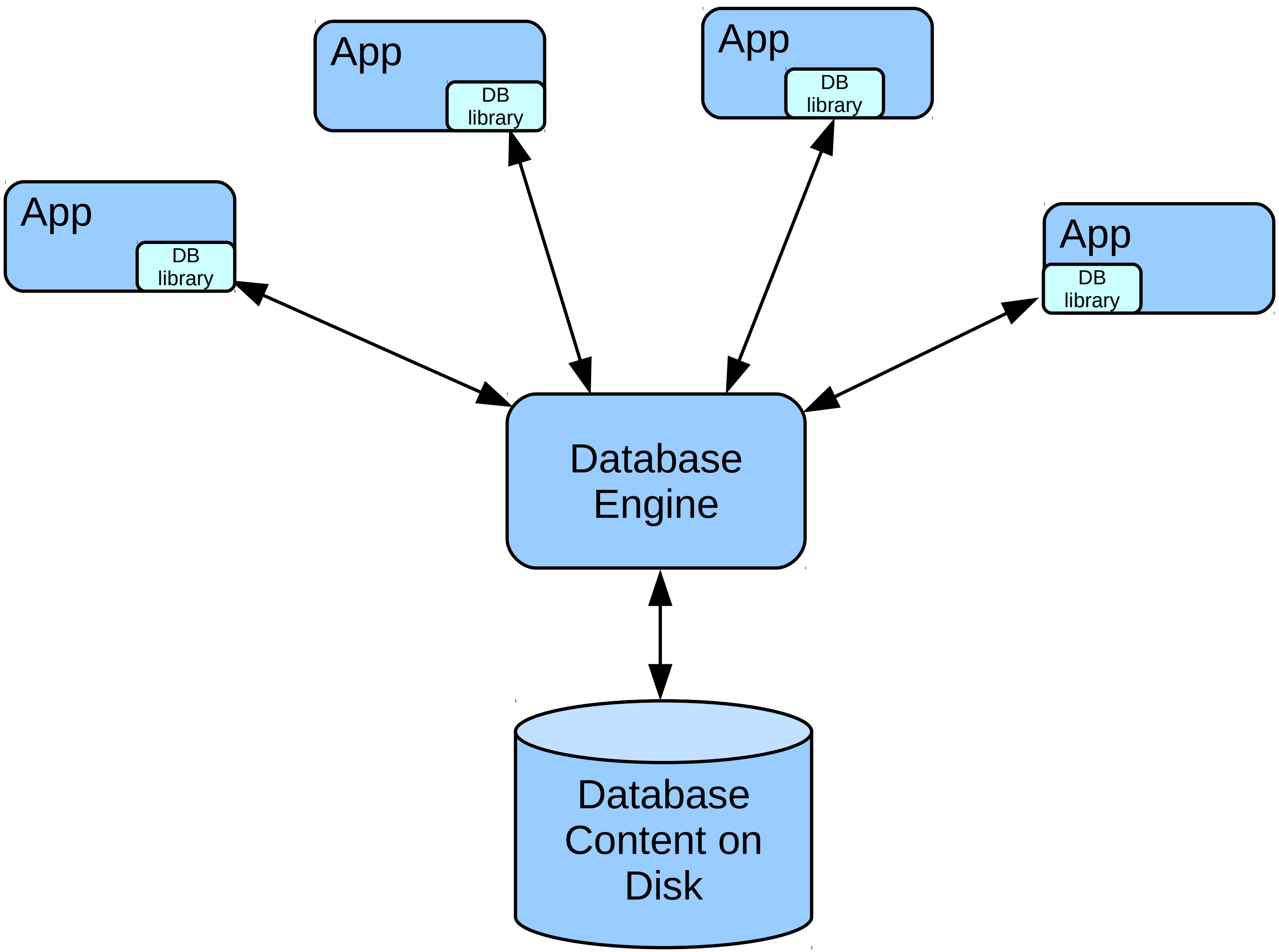
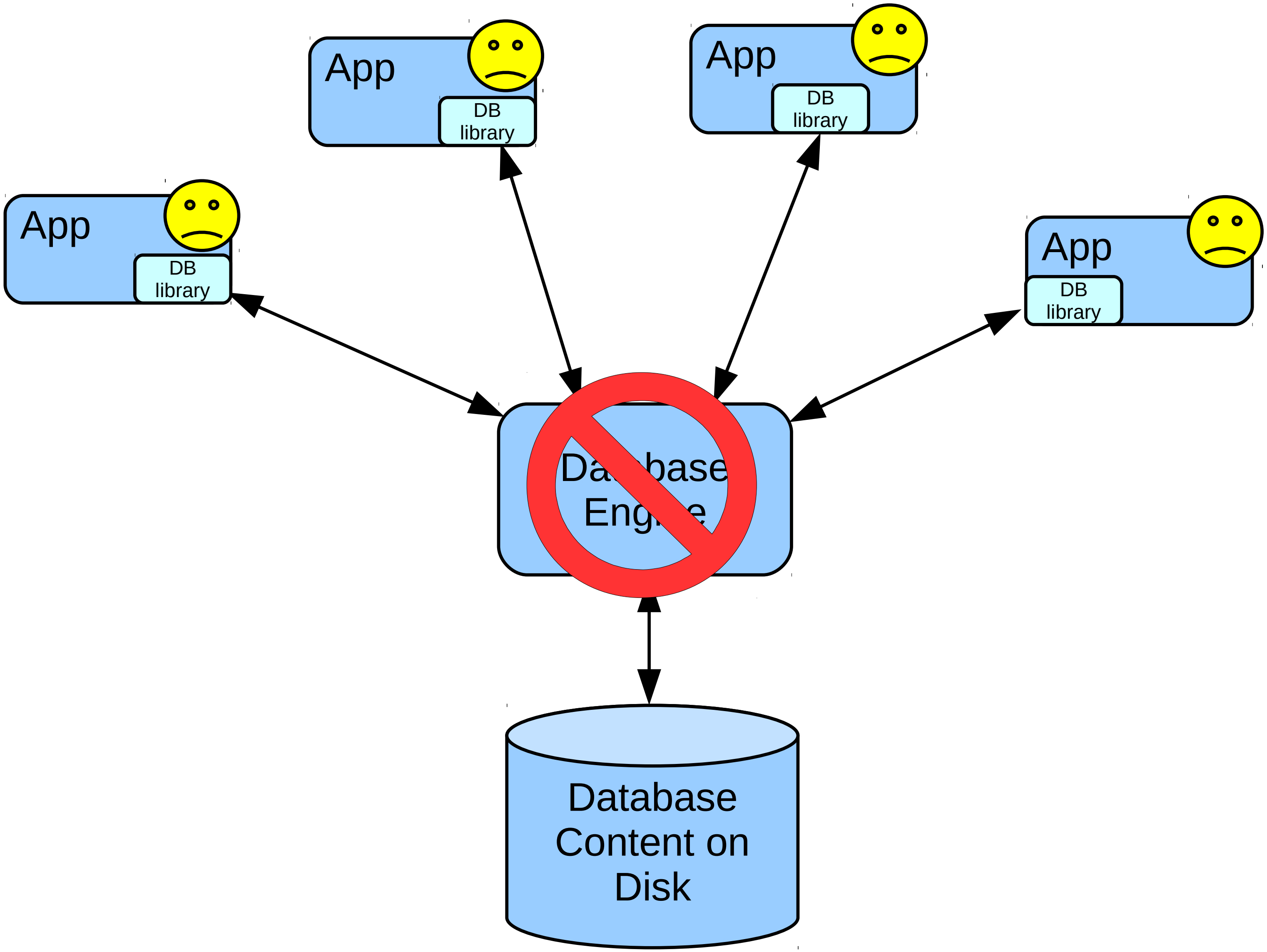
How It Works

https://sqlite.org/talks/howitworks-20240624.pdf

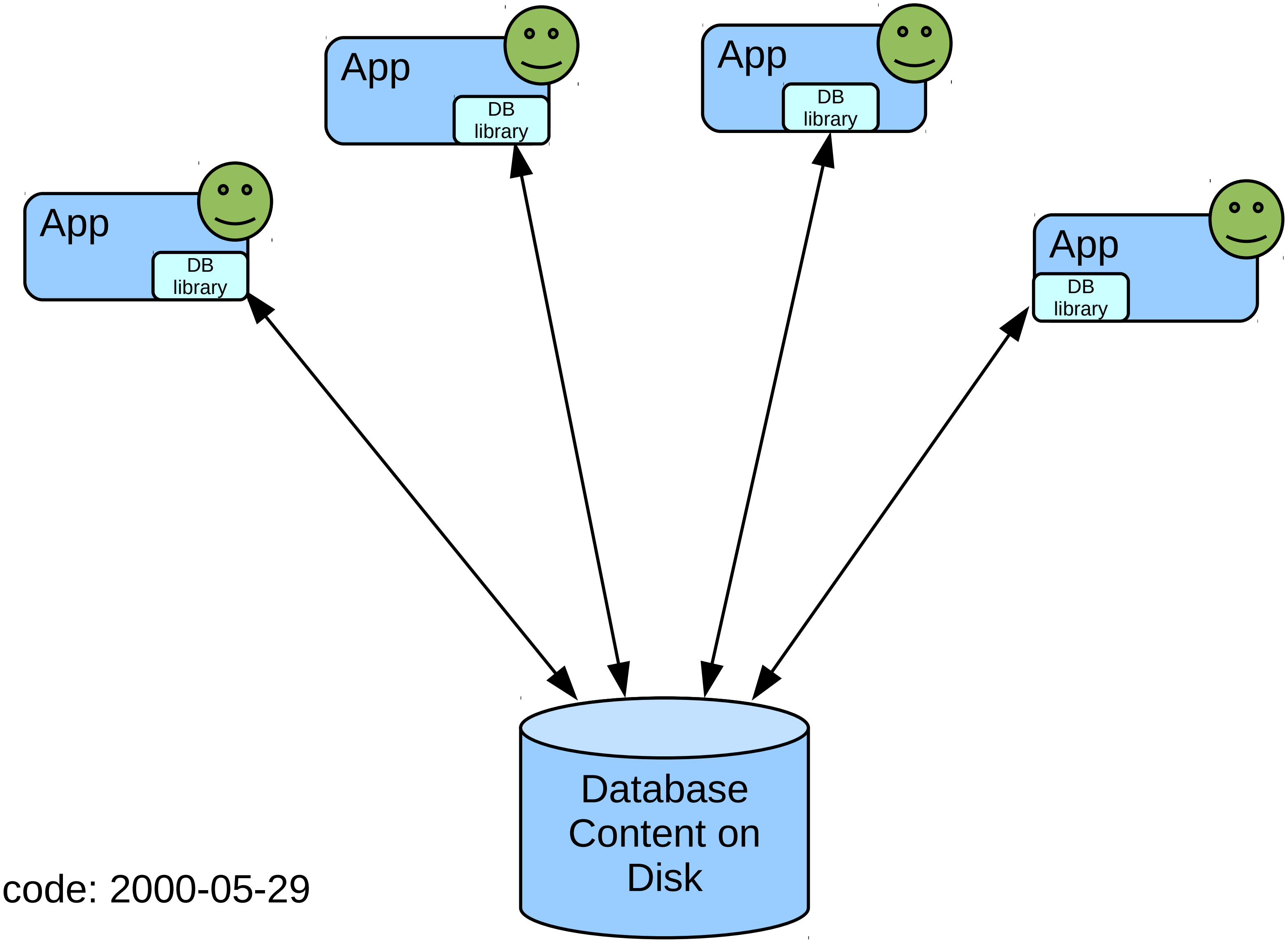
SQLite in a nutshell

- An in-process library → not a server
- One file of ANSI-C code
- Database is a single file
- Full-featured SQL
- Power-safe, serializable transactions
- Fast
- Simple API
- Public domain


App
DB
library
App
DB
library
App
DB
library
App
DB
library
Database
Engine
Database
Content on
Disk



App
DB library
App
DB library
App
DB library
App
DB library
Database Engine
Database Content on Disk



First code: 2000-05-29

	Embedded	Client/Server
Non-SQL	• GDBM • BerkeleyDB • LevelDB • RocksDB • ... and so forth	MongoDB® redis DynamoDB
SQL	SQLite	MySQL™ PostgreSQL Microsoft® SQL Server™ ORACLE®

One File Of C-code

sqlite3.c

- 257K lines
- 163K SLOC[1]
- 9.08MB

Also: **sqlite3.h**

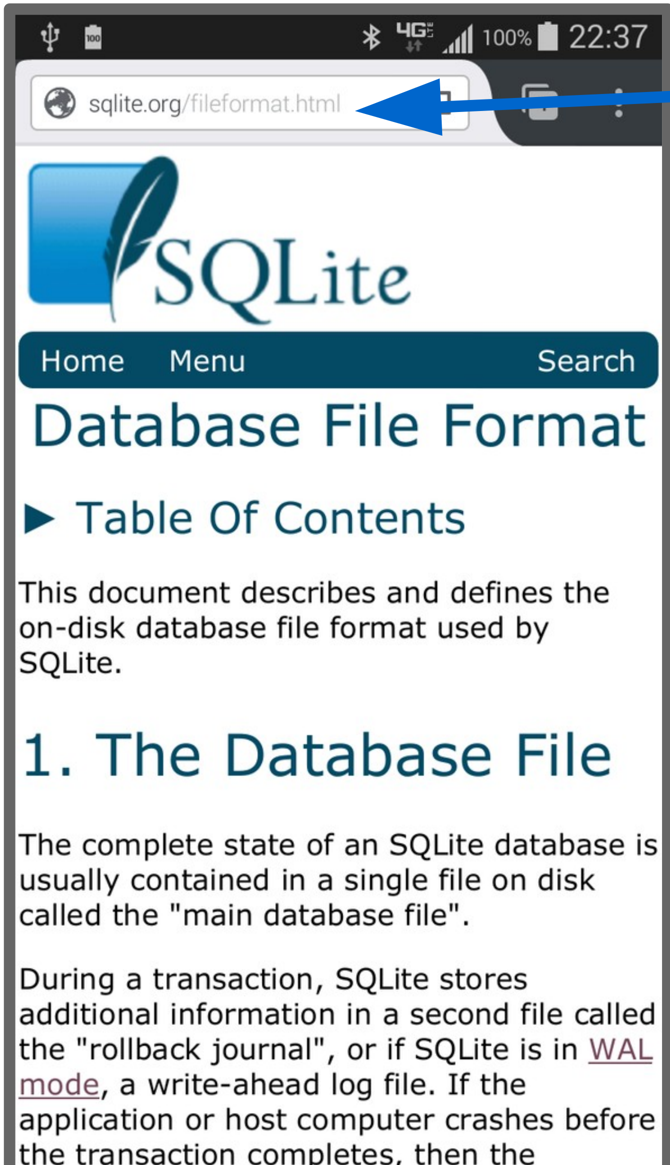
- 13.4K lines
- 1.8K SLOC
- 0.64MB

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

Single File Database

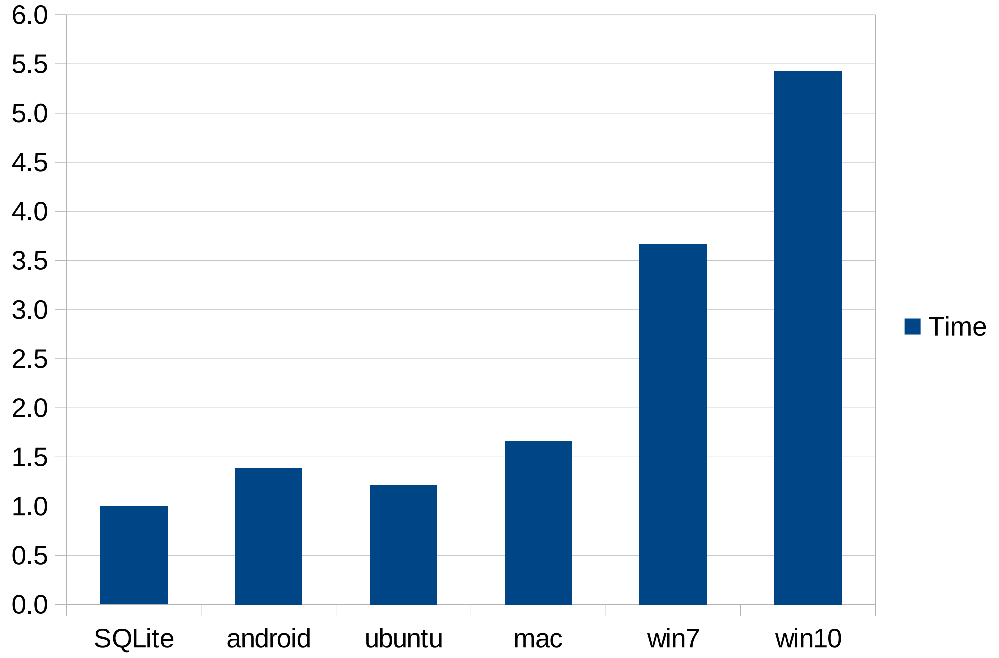
- 1000s of tables, indexes, and views
- Space efficient
- Send a complete database as an email attachment
- Name it whatever you like.
- Application file format

Open File Format



- sqlite.org/fileformat.html
  - ~20 pages
- Cross-platform
  - 32-bit ↔ 64-bit
  - little- ↔ big-endian
- Backwards-compatible
- Space efficient
- Readable by 3rd-party tools
- Supported through 2050
- Recommended by LoC

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

Copyright

“Lite” means “Low Overhead”

not “low capability”

- 1 writer + N concurrent readers
- 1 gigabyte strings and BLOBs
- 281 terabyte databases
- 64-way joins
- 2000 columns per table or index
- No arbitrary limit on the number of tables or indexes or rows in a table

Storage Decision Checklist

Remote Data?

Big Data?

Concurrent Writers?

Otherwise

Storage Decision Checklist **FAIL!**

Remote Data?

Big Data?

Concurrent Writers?

No!

Otherwise

fopen()

SQLite Found In...

- Every Android and iOS phone and device
- Every Mac and Windows10/11 computer
- Every Firefox, Chrome, and Safari browser
- Most desktop and phone software applications
- TVs, smart appliances, automotive “infotainment” systems
- Countless millions of other applications....
- More and more of websites

Implementation Overview

Interface

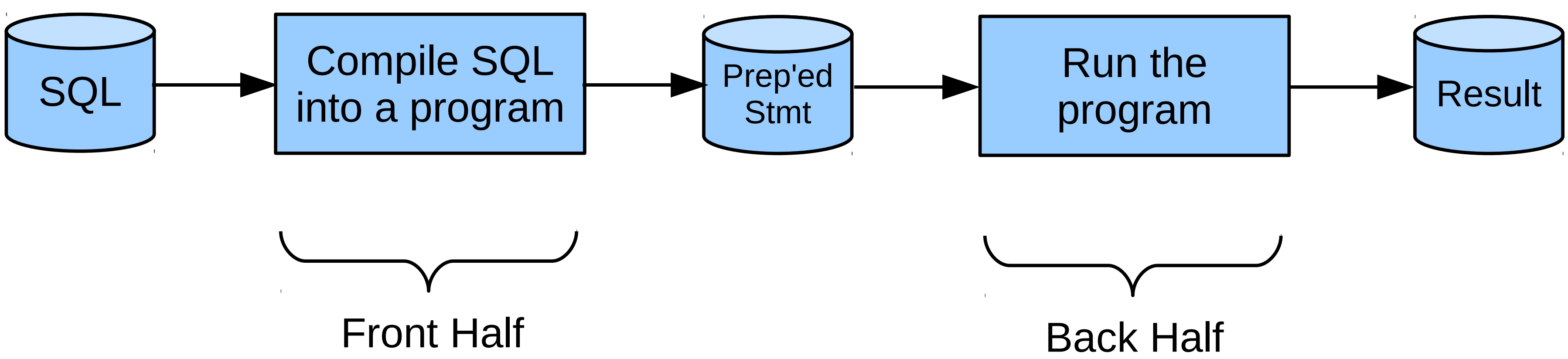
```
int main(int argc, char **argv){
  sqlite3 *db;            /* Database connection */
  sqlite3_stmt *pStmt;    /* One SQL statement */
  int nCol;               /* Number of columns in the result set */

  db = sqlite3_open(argv[1]);
  pStmt = sqlite3_prepare(db, argv[2]);
  nCol = sqlite3_column_count(pStmt);
  while( sqlite3_step(pStmt)==SQLITE_ROW ){
    int i;
    printf("Row:\n");
    for(i=0; i<nCol; i++){
      printf("  %s = %s\n",
        sqlite3_column_name(pStmt, i),
        sqlite3_column_text(pStmt, i)
      );
    }
  }
  sqlite3_finalize(pStmt);
  sqlite3_close(db);
  return 0;
}
```

***** Pseudo-code *****

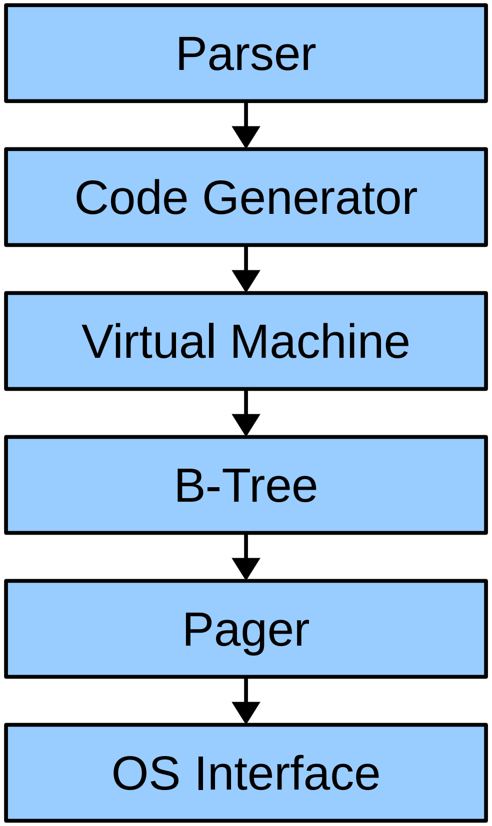
Ins & Outs

- SQLite consists of...
  - Compiler → translates SQL into bytecode
  - Virtual Machine → runs the bytecode



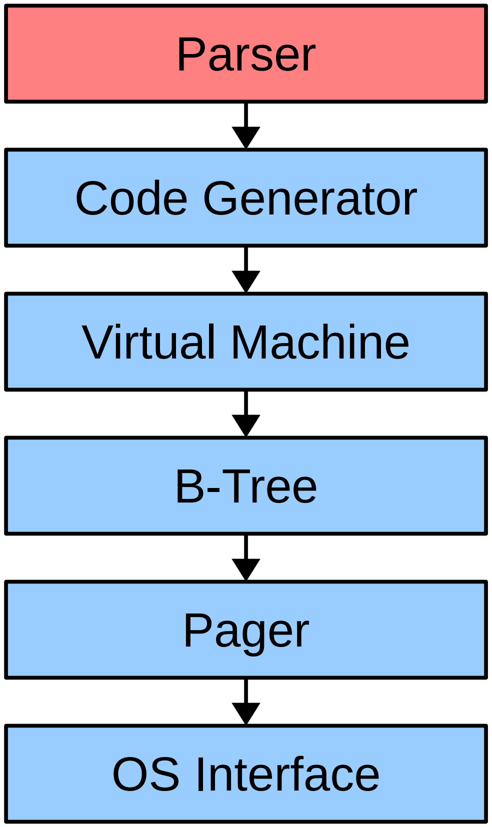
`sqlite3_prepare()` `sqlite3_step()`

The SQLite Stack



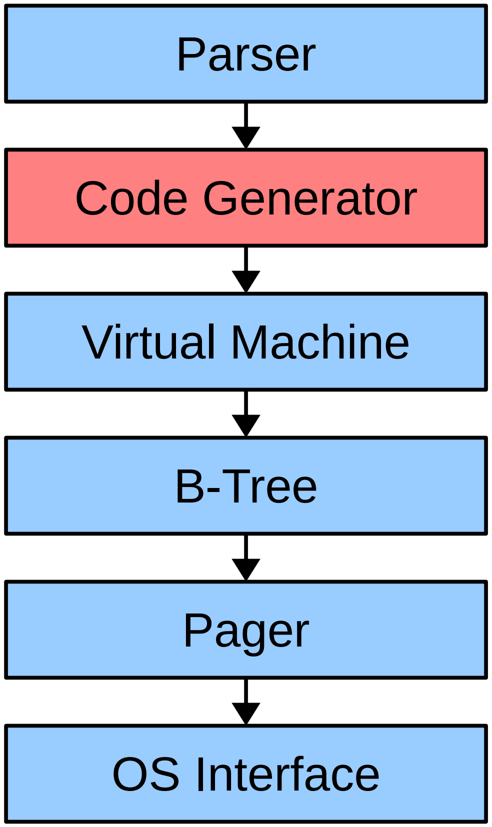
The SQLite Stack

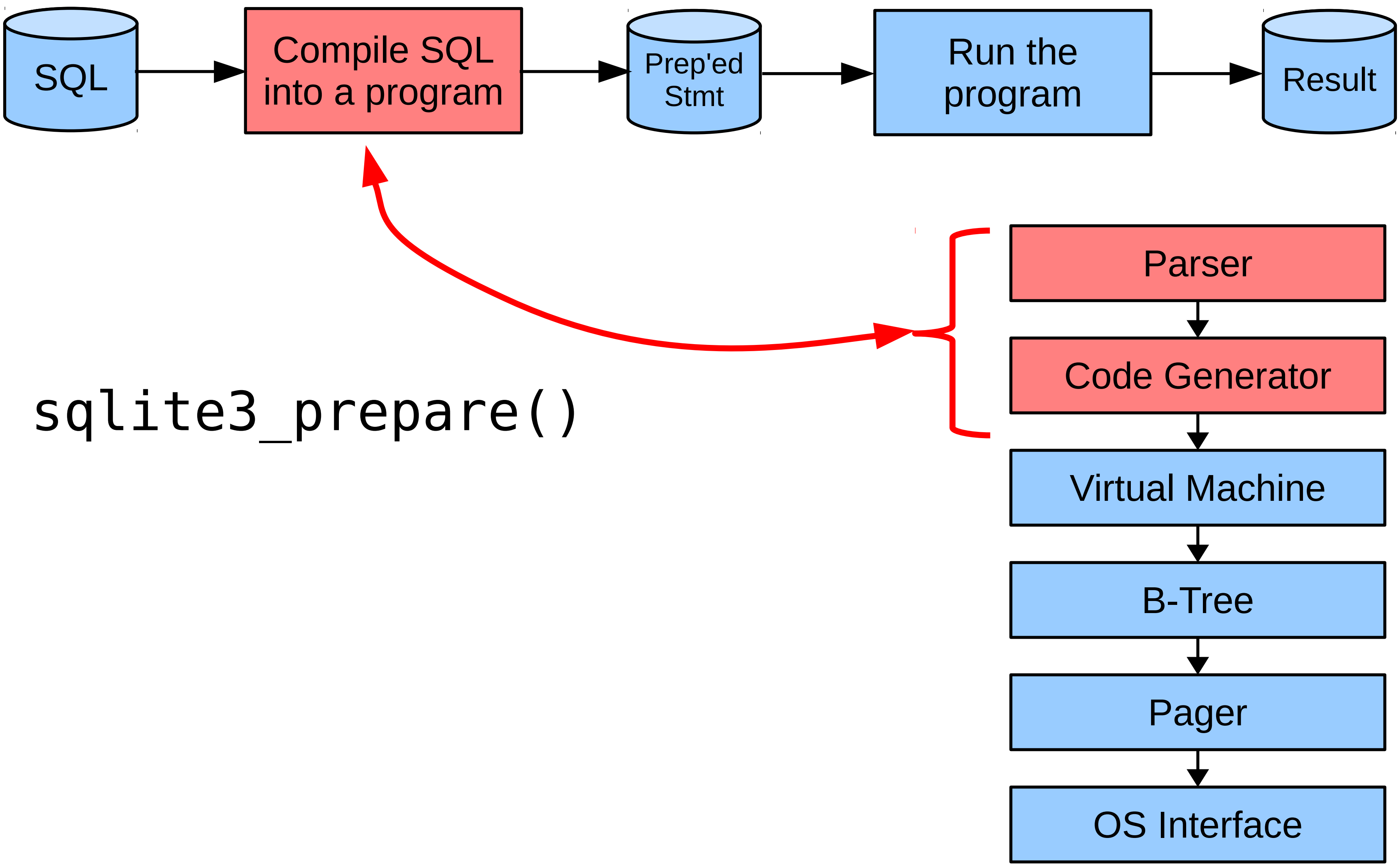
- LALR(1) parser generated by “Lemon”
- Reentrant & threadsafe
- Output: Abstract Syntax Tree (AST)
- Hand-written tokenizer

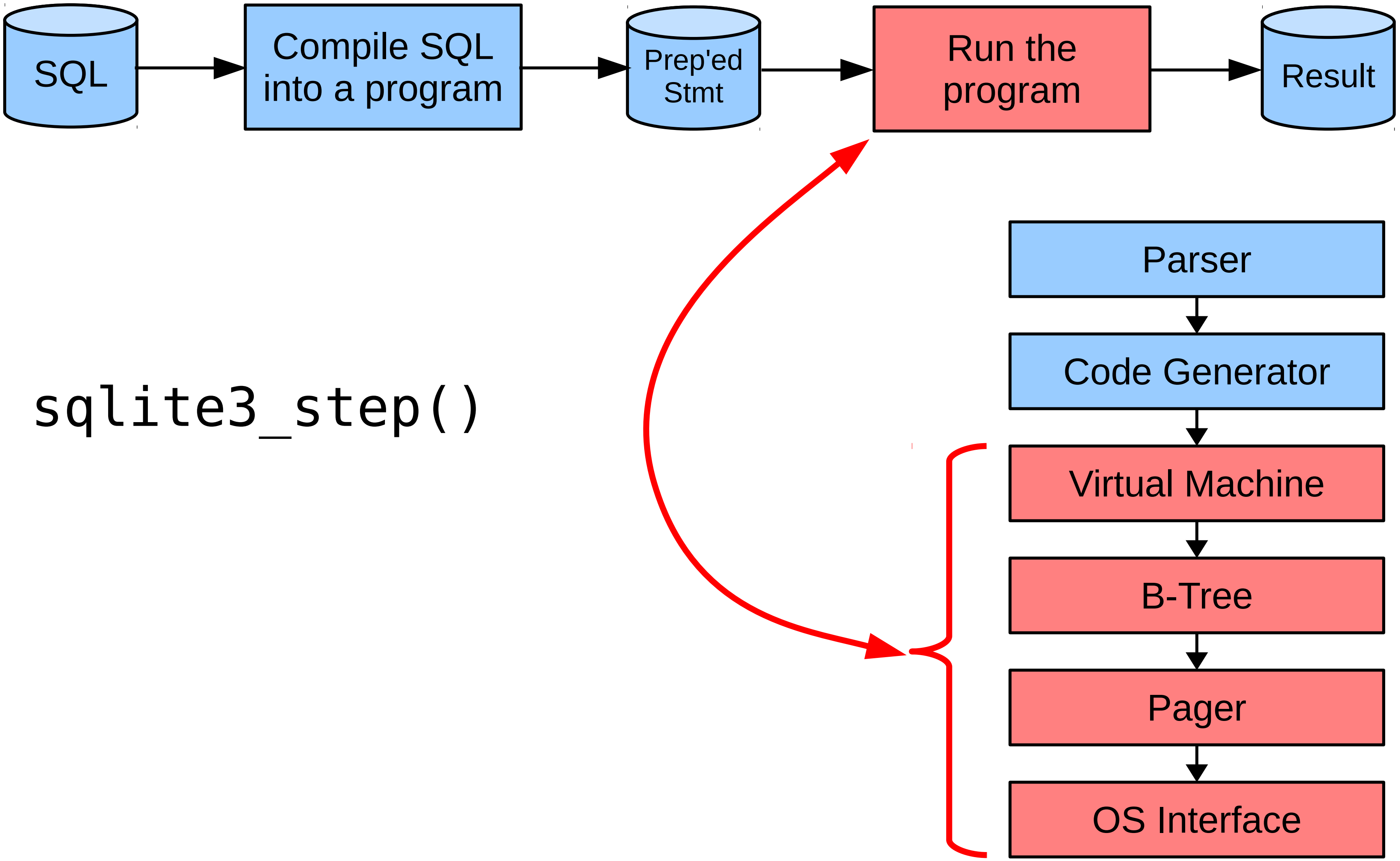


The SQLite Stack

- Semantic analysis
- AST transformations
- Query planning
- Output: “prepared statement” (bytecode)

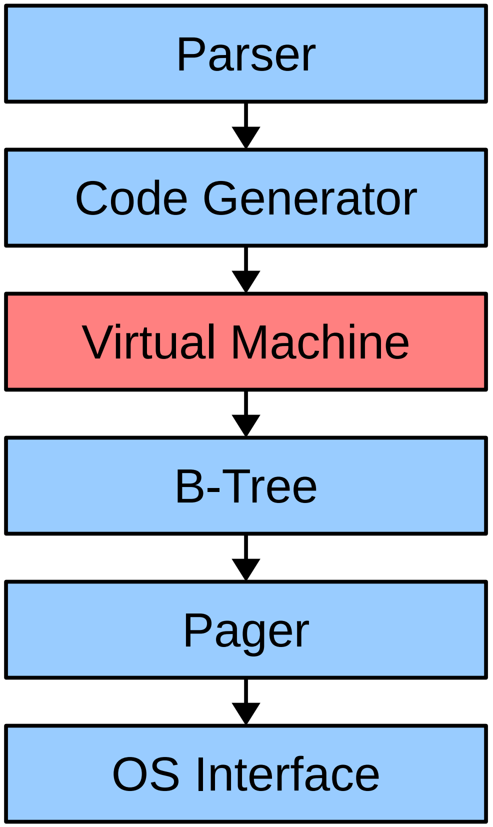



SQL
Compile SQL into a program
Prep'ed Stmt
Run the program
Result
sqlite3_prepare()
Parser
Code Generator
Virtual Machine
B-Tree
Pager
OS Interface


SQL
Compile SQL into a program
Prep'ed Stmt
Run the program
Result
sqlite3_step()
Parser
Code Generator
Virtual Machine
B-Tree
Pager
OS Interface

The SQLite Stack

- Bytecode interpreter
- 3-address, 5-operand register machine
- Big `switch` statement on the opcode and inside a loop.
- Special opcodes designed to help implement SQL statements



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://sqlite.org/opcode.html**

Grayed content appears only when using special compile-time options intended for debugging and analysis.

EXPLAIN SELECT price FROM tab WHERE fru

```
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
```

Indentation showing loop structure is inserted by the display logic in the command-line shell and is not part of the actual bytecode.

The “opcode” is really a small integer.

“Debug” version of the command-line tool for SQLite

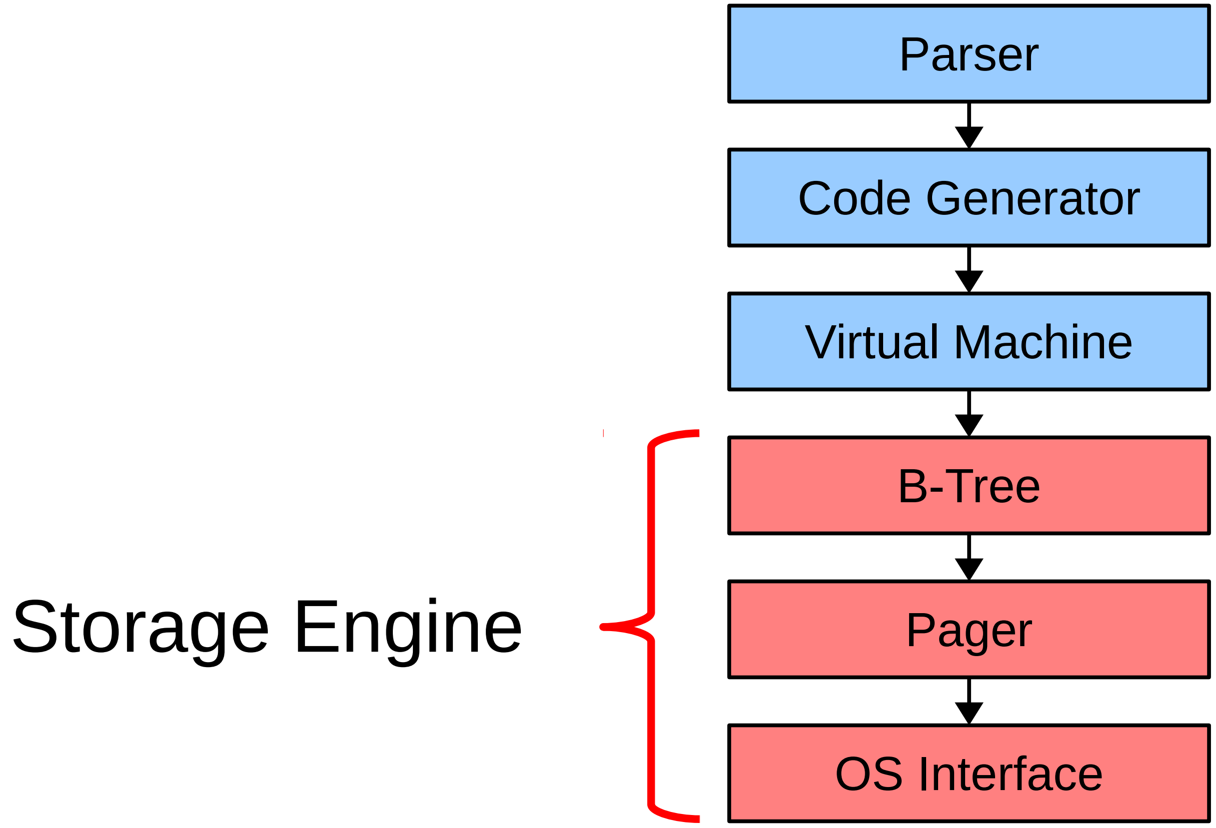
- Browser: https://sqlite.org/fiddle-debug
- Linux:
  - apt install build-essentials tcl-dev
  - ./configure --enable-debug && make sqlite3
- Mac:
  - Install XCode and TCL
  - ./configure --enable-debug && make sqlite3
- Windows: https://sqlite.org/src/doc/trunk/doc/compile-for-windows.md

Extra debugging commands

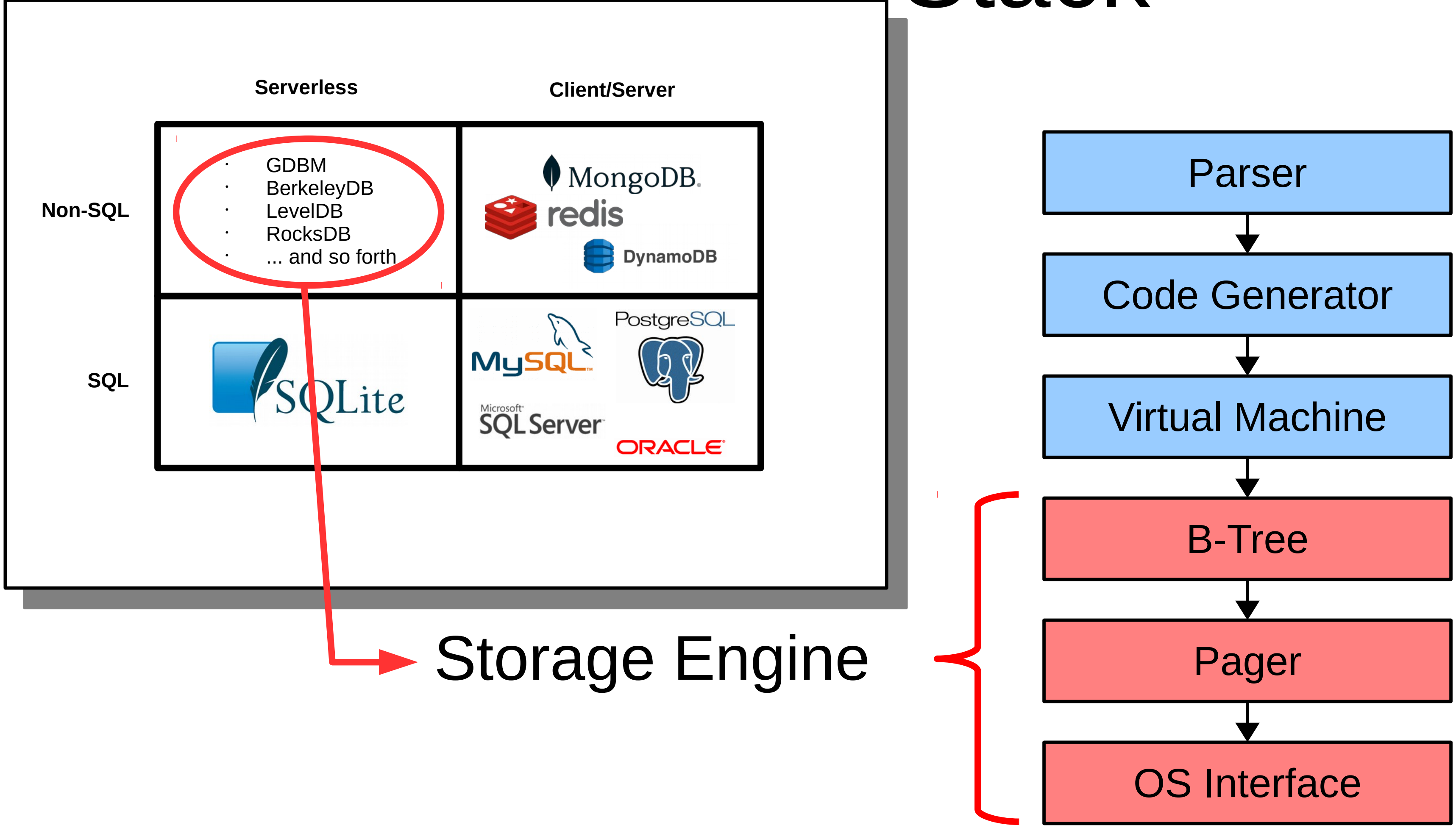
`.treetrace -1` ← *Show the AST as ASCII-art*

`.eqp trace` ← *Trace bytecode execution*

The SQLite Stack

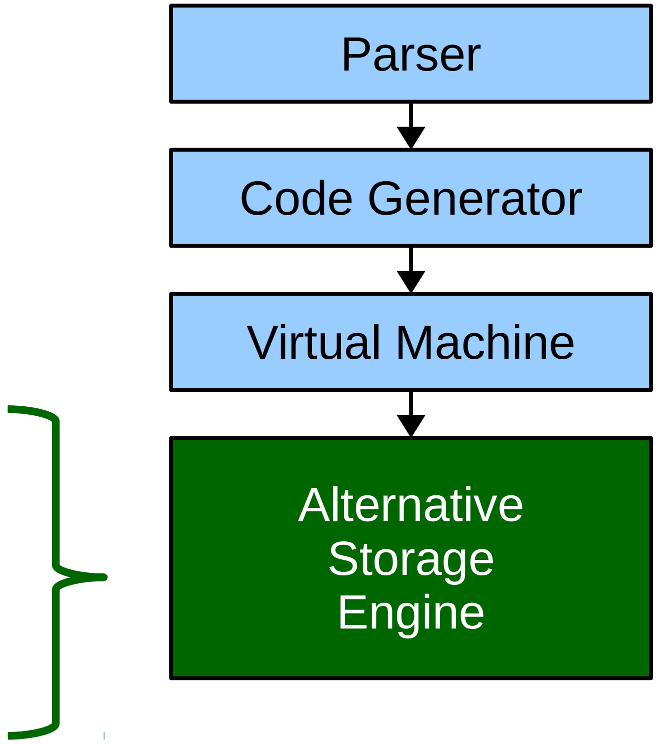


The SQLite Stack



The SQLite Stack

Notable example:
Comdb2 by Bloomberg
https://bloomberg.github.io/comdb2

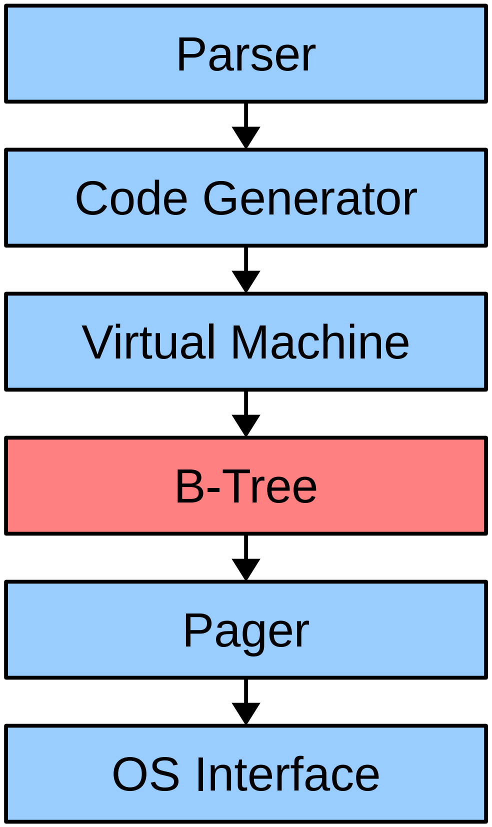


Storage Engine Summary

- Row-store
- Variable-length entries
- Forest of B-trees
  - One B-tree for each table and each index
  - Table key: PRIMARY KEY or ROWID
  - Index key: indexed columns + table key
- Transaction control using rollback-journal or write-ahead log.

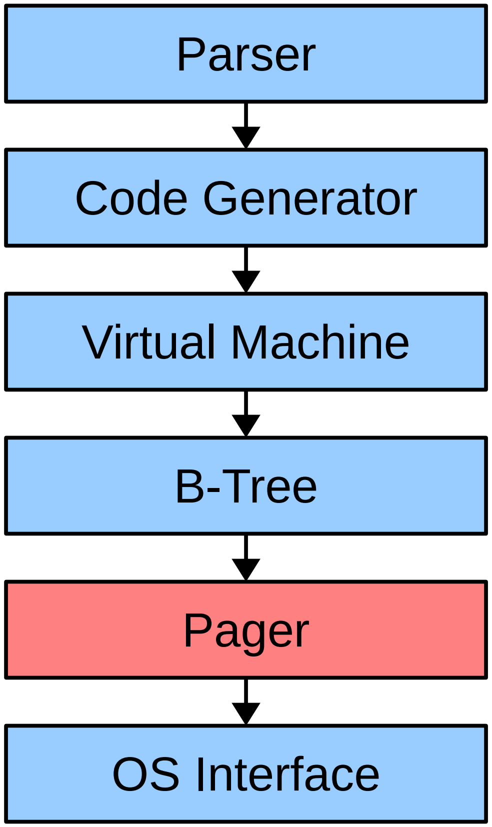
The SQLite Stack

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



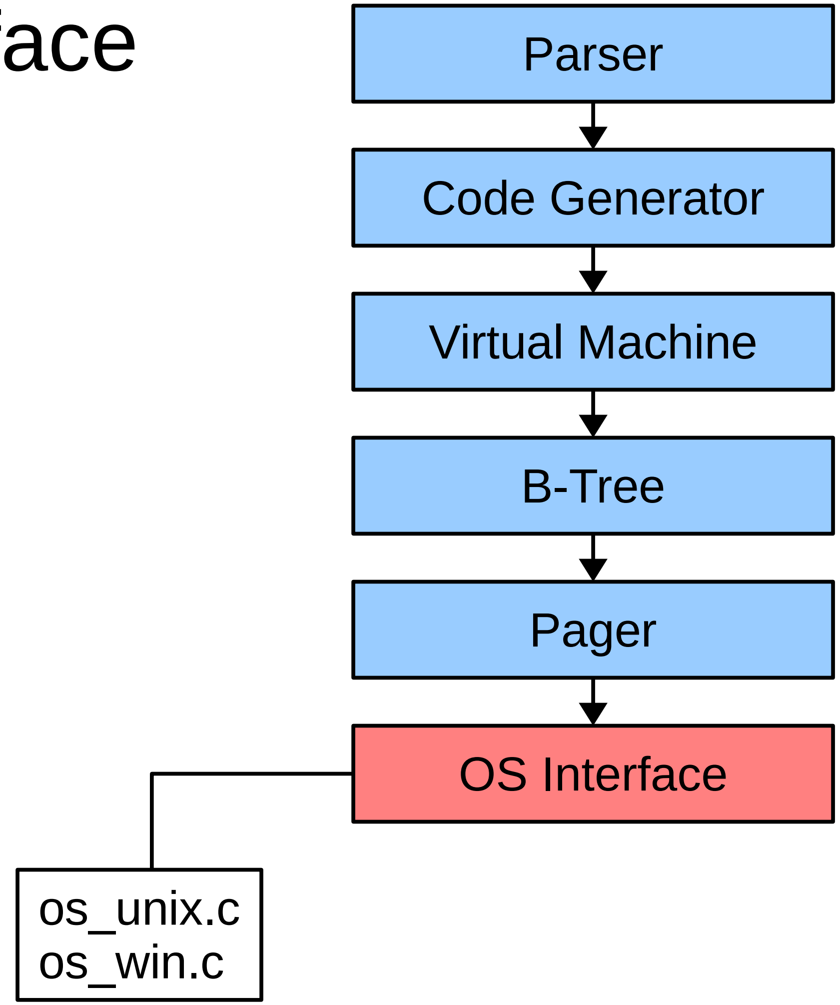
The SQLite Stack

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



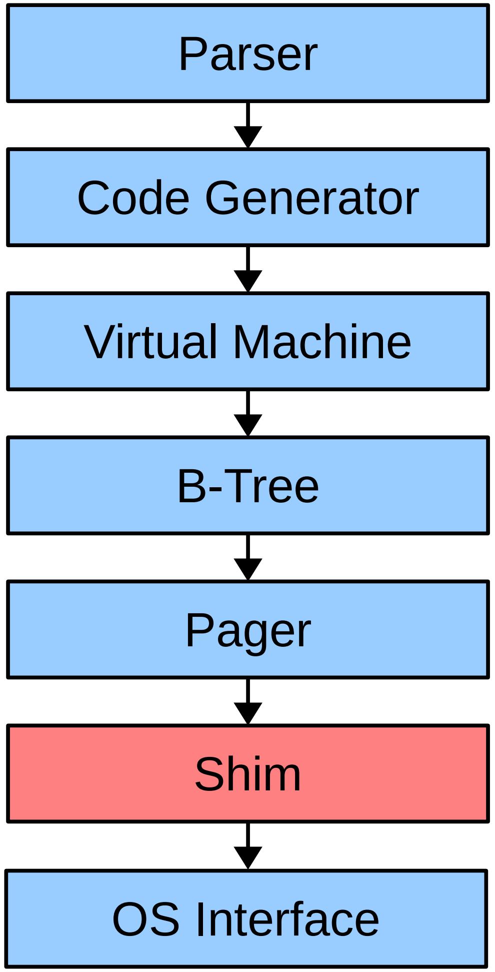
The SQLite Stack

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



VFS Shims

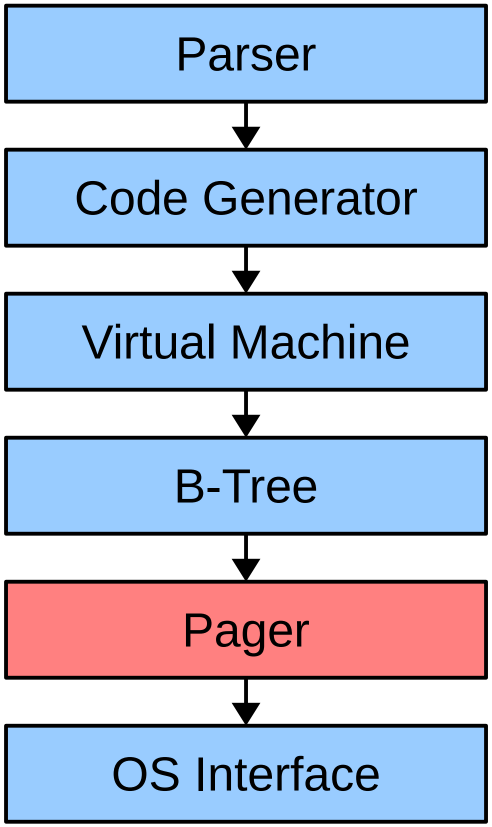
- Inserted in between Pager and OS Interface
- Encryption
- Compression
- Logging
- Testing & fault injection
- And so forth...



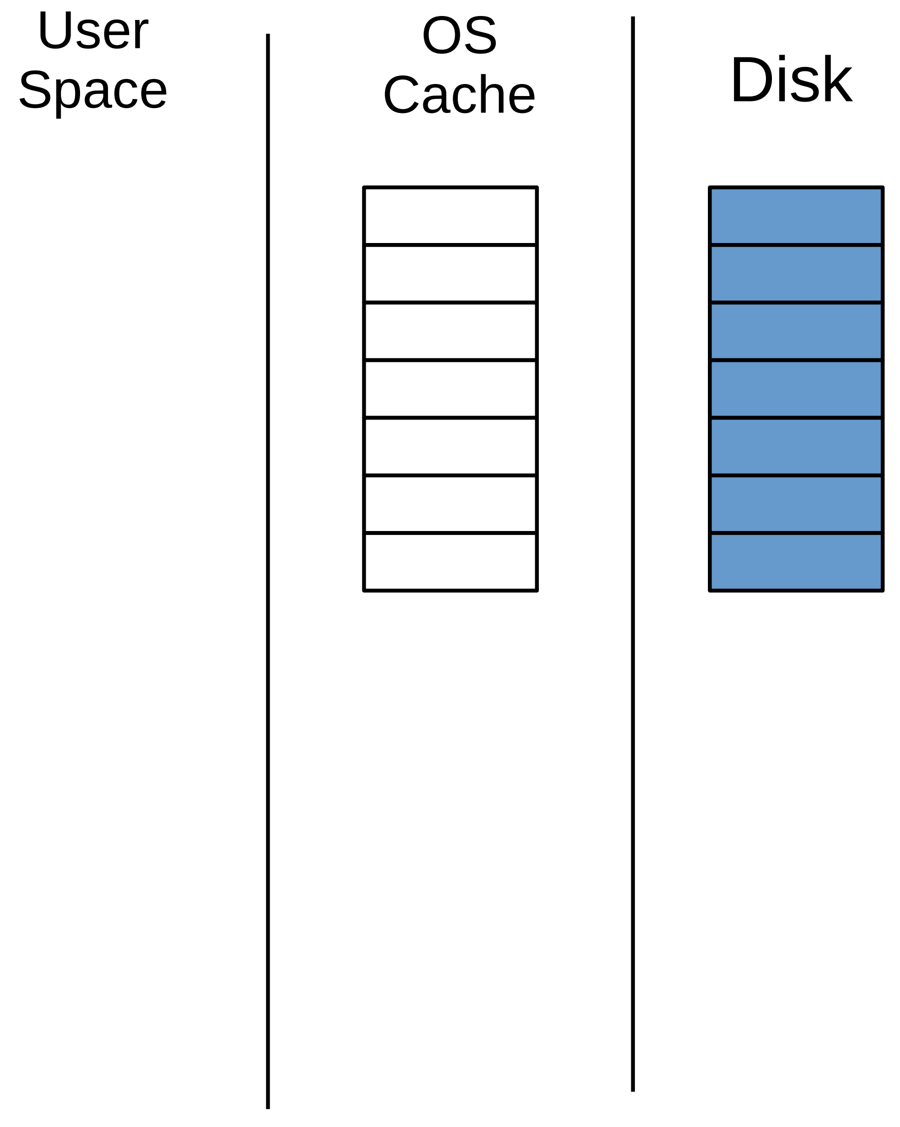
More Detail

The SQLite Stack

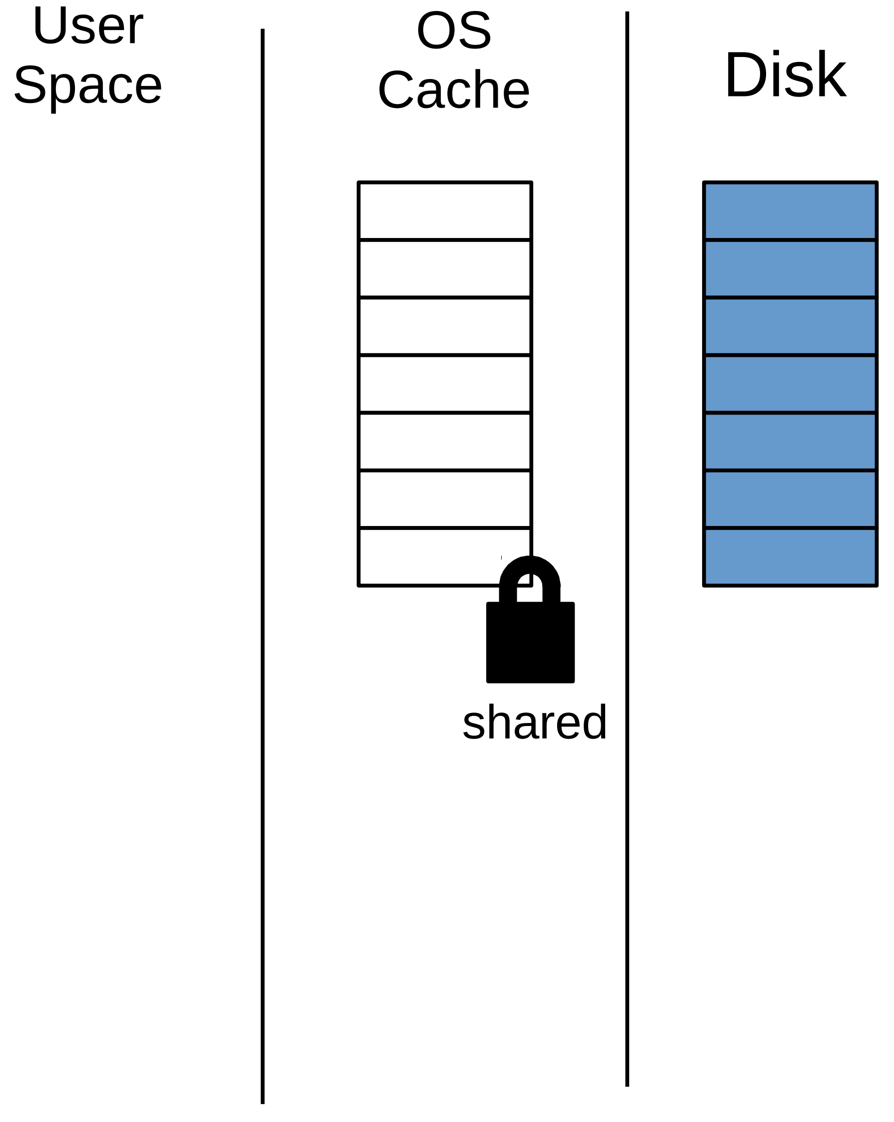
- Power-safe transactions
  - Rollback mode
  - Write-ahead log (WAL) mode
- Concurrency control
- In-memory cache of disk content



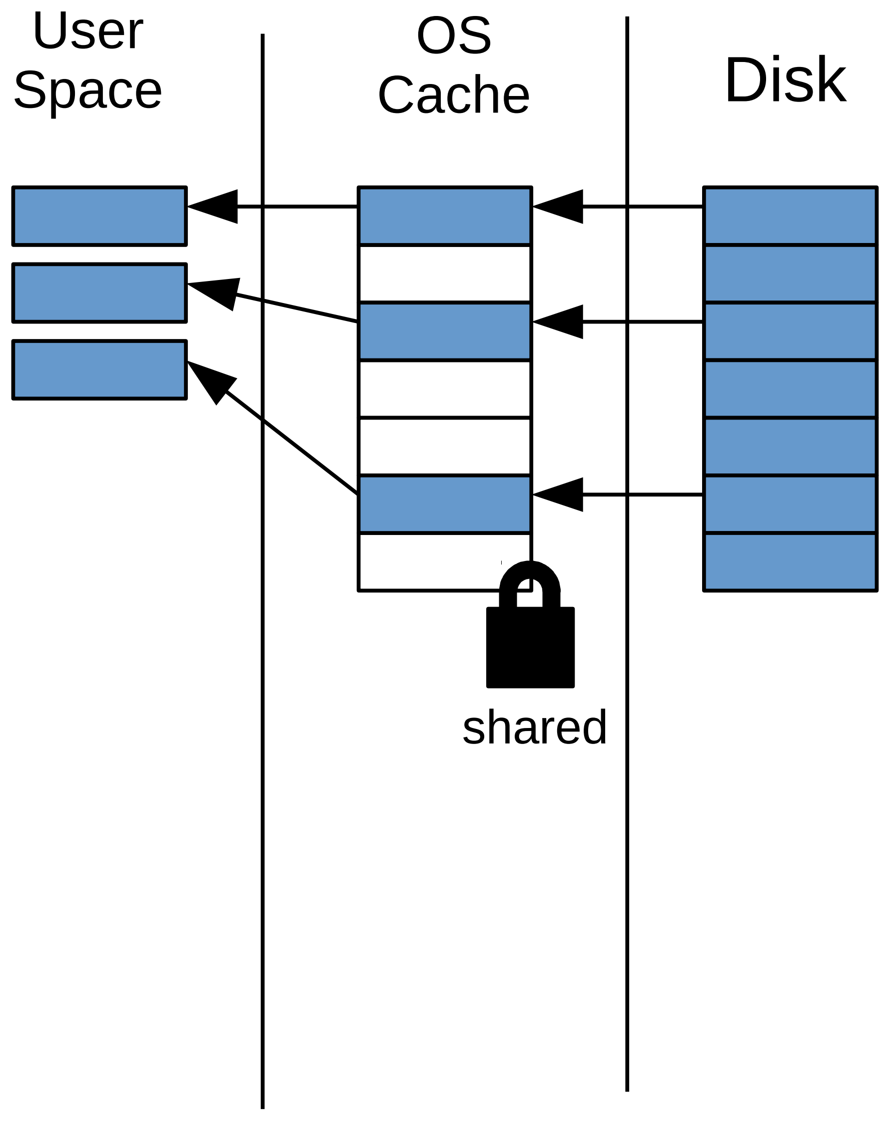
Rollback Journaling



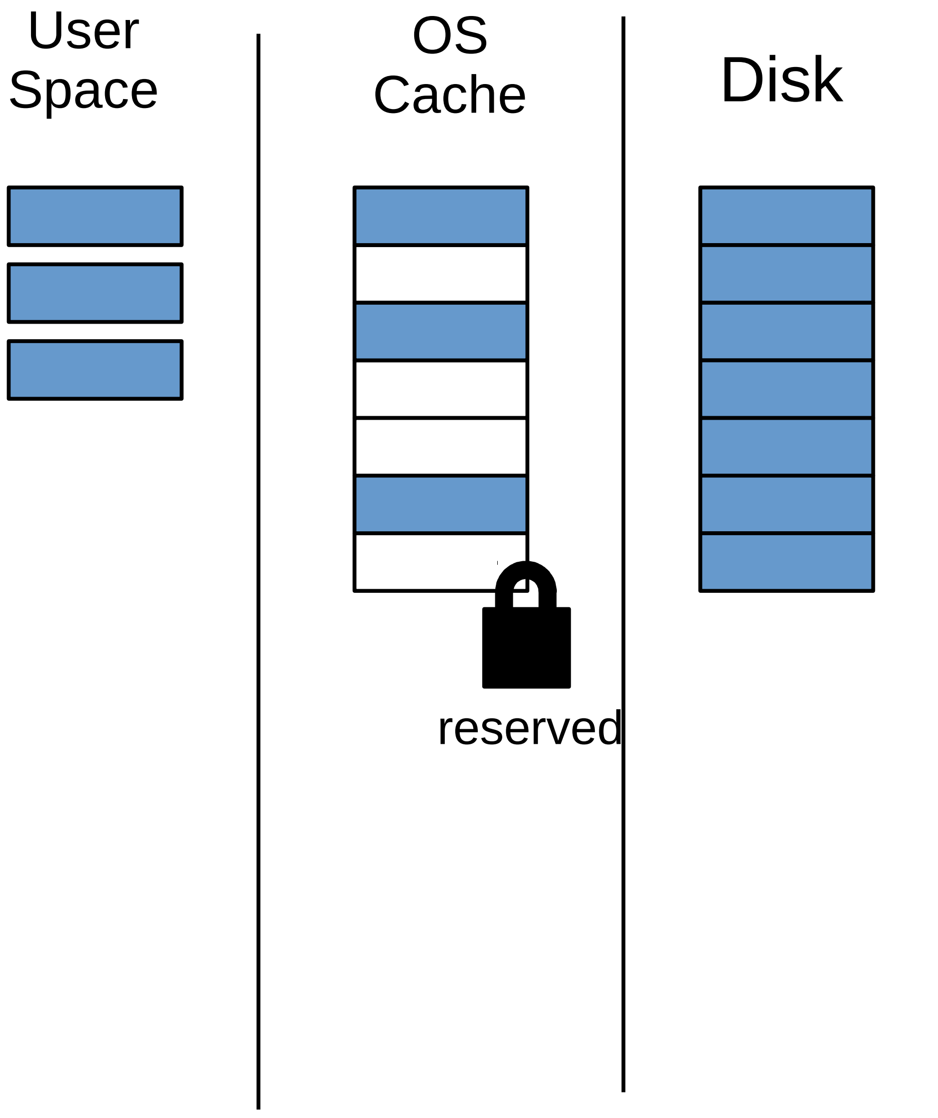
Rollback Journaling



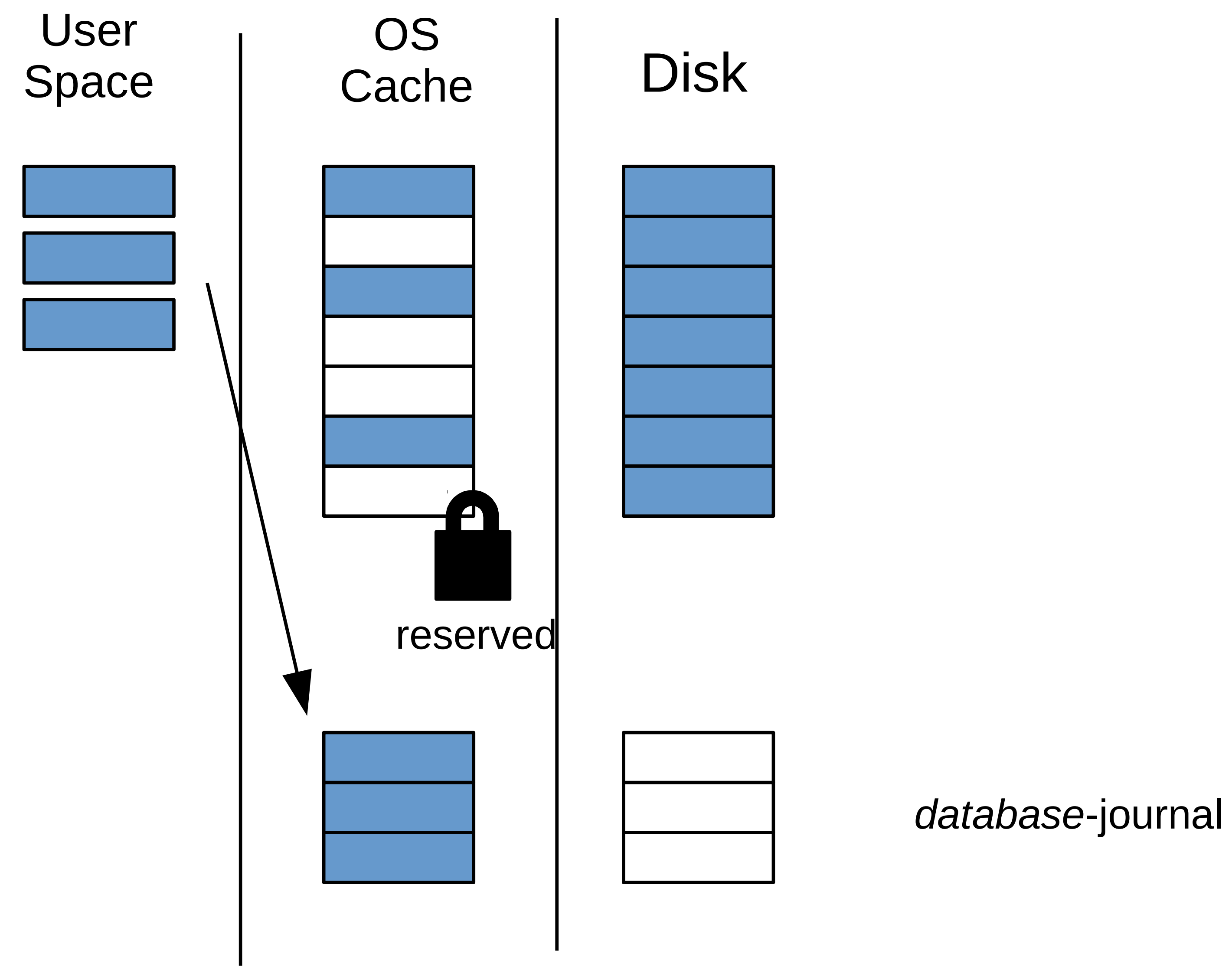
Rollback Journaling



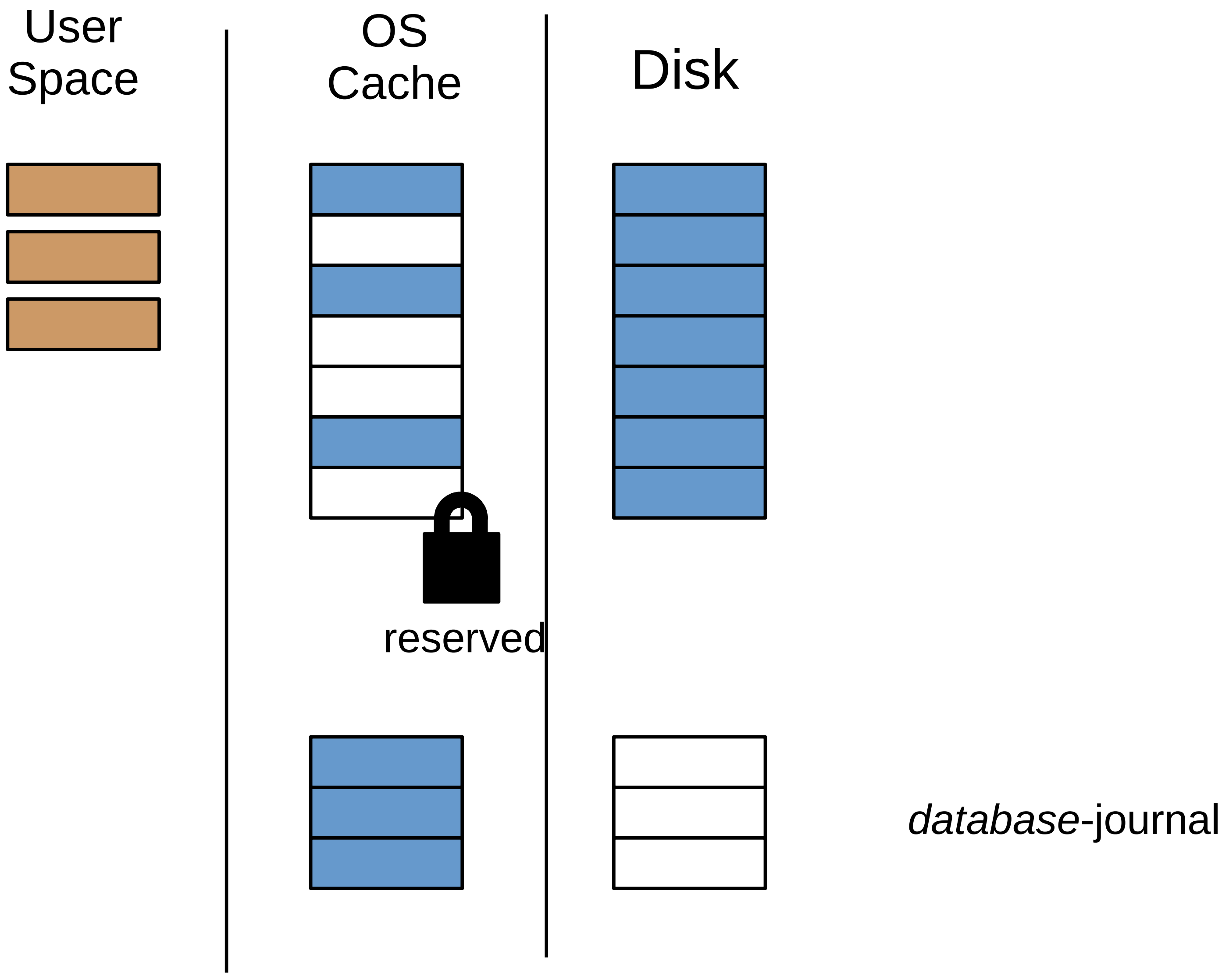
Rollback Journaling



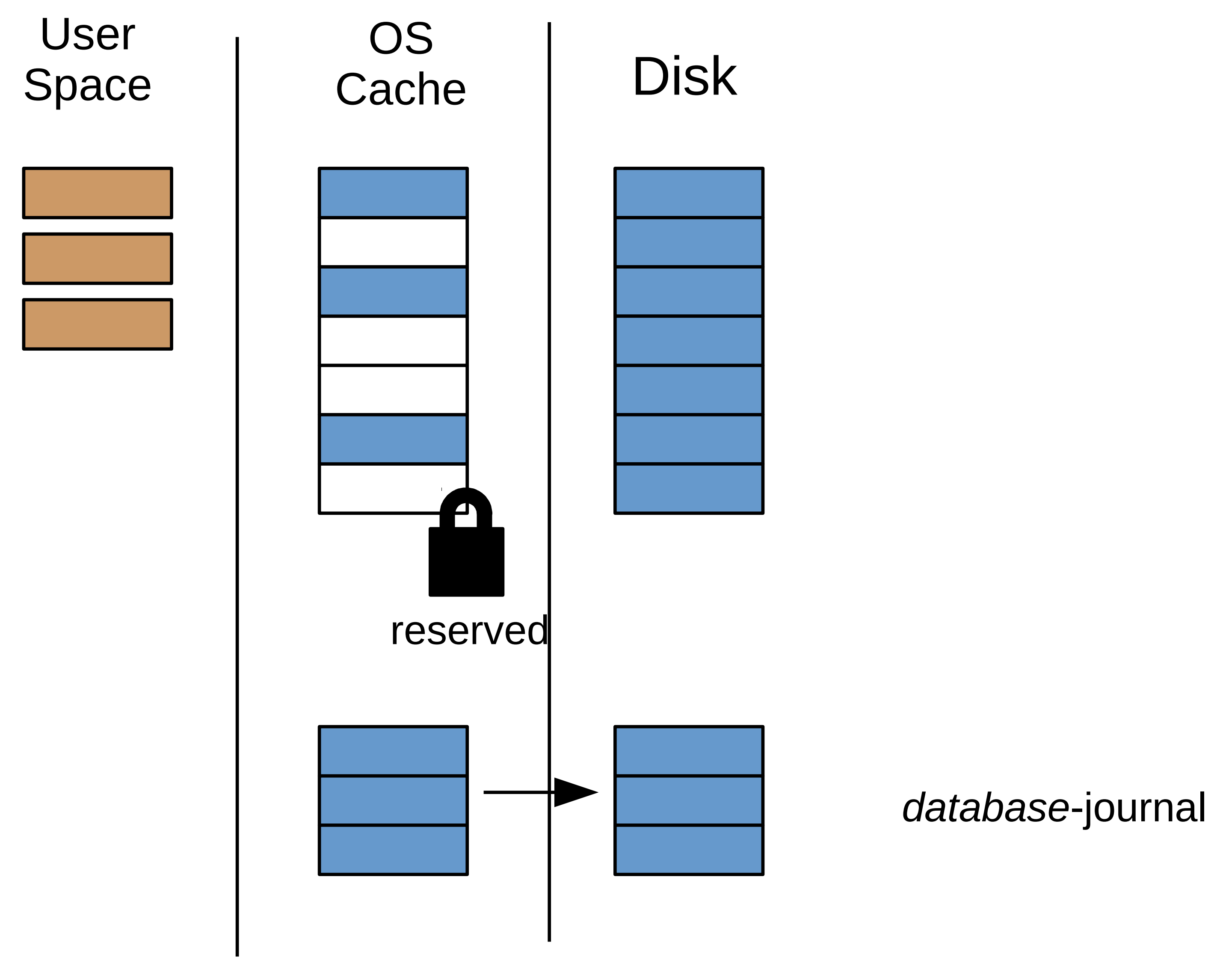
Rollback Journaling



Rollback Journaling

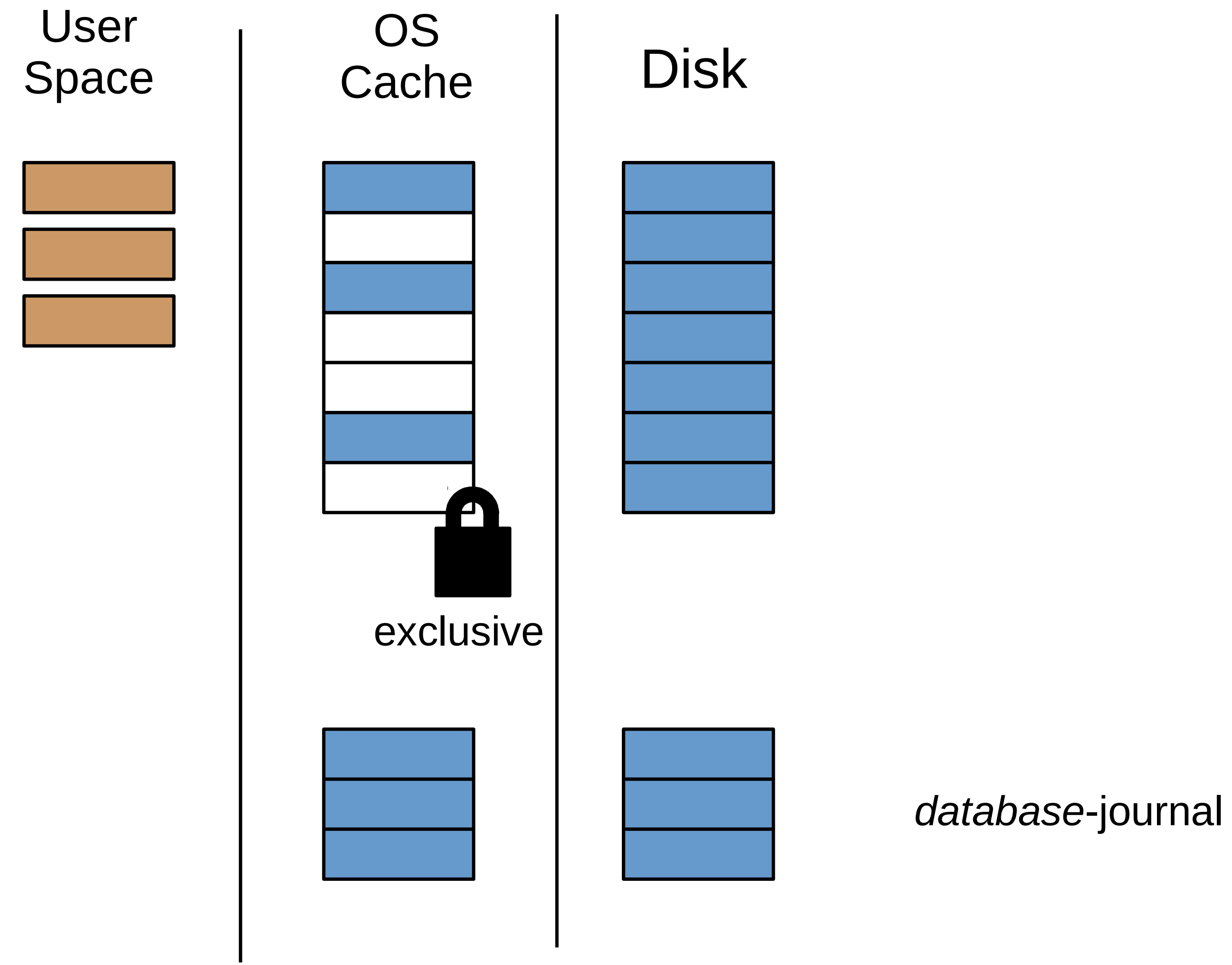


Rollback Journaling

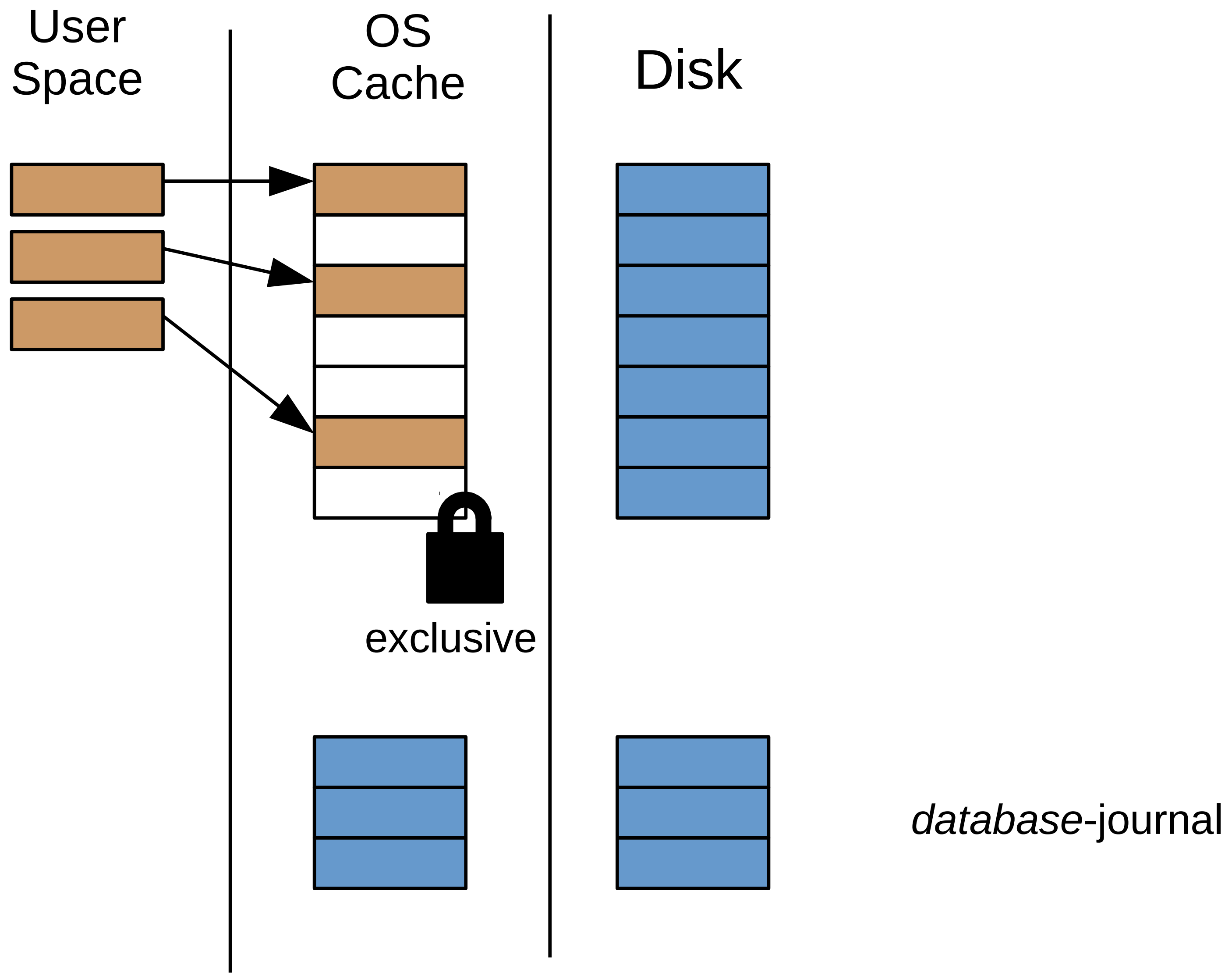


➔ *Disable this step using PRAGMA synchronous=OFF*

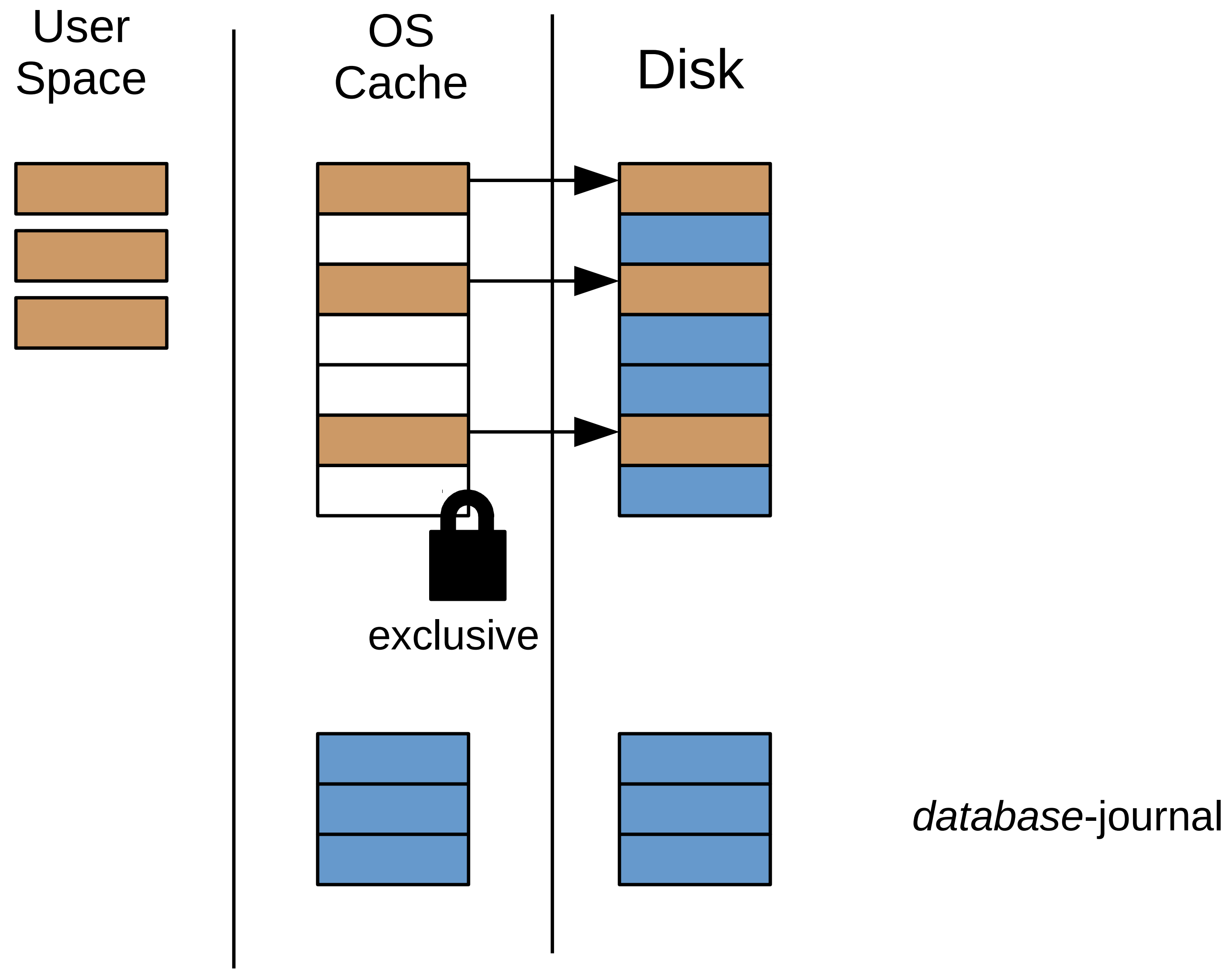
Rollback Journaling



Rollback Journaling

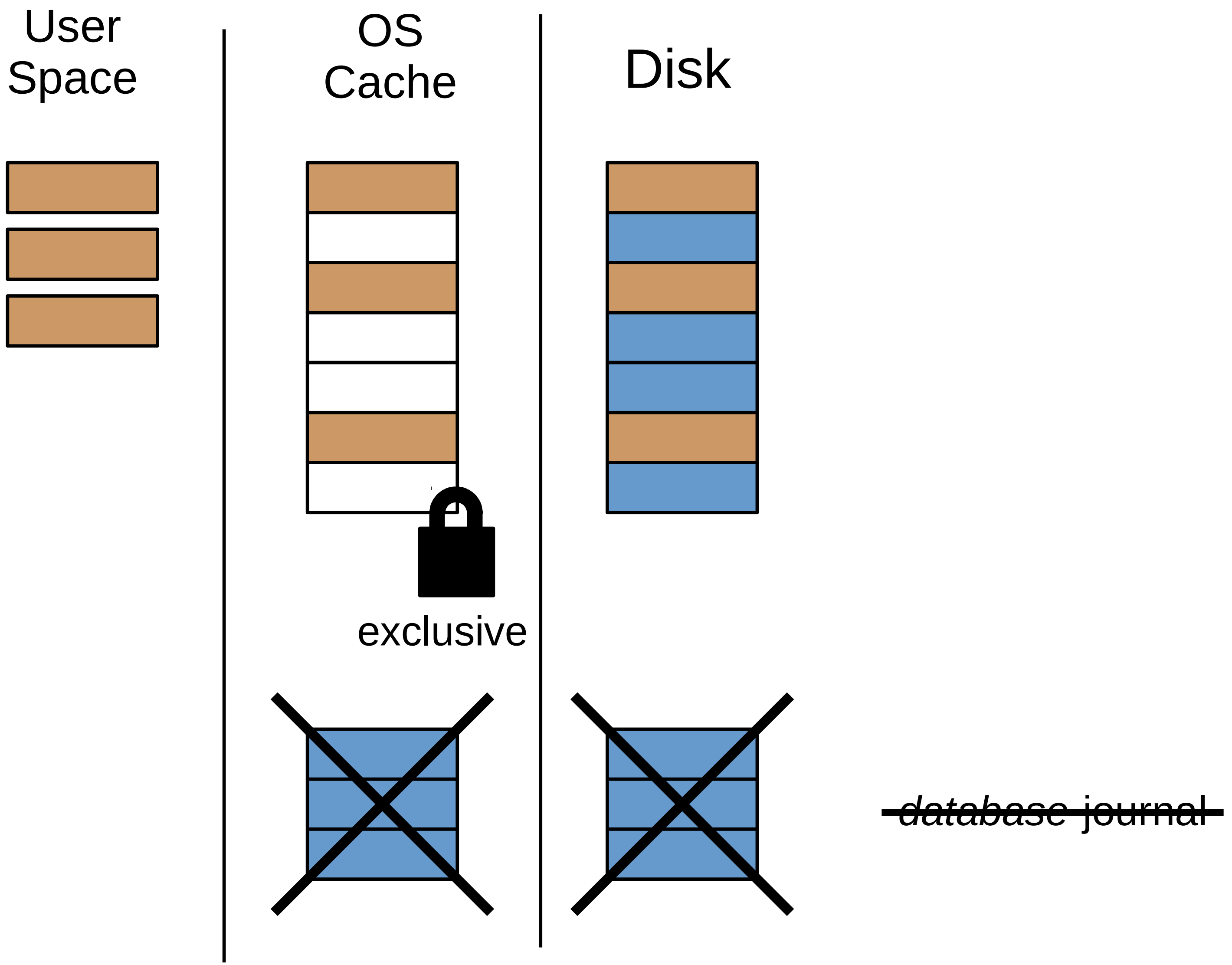


Rollback Journaling

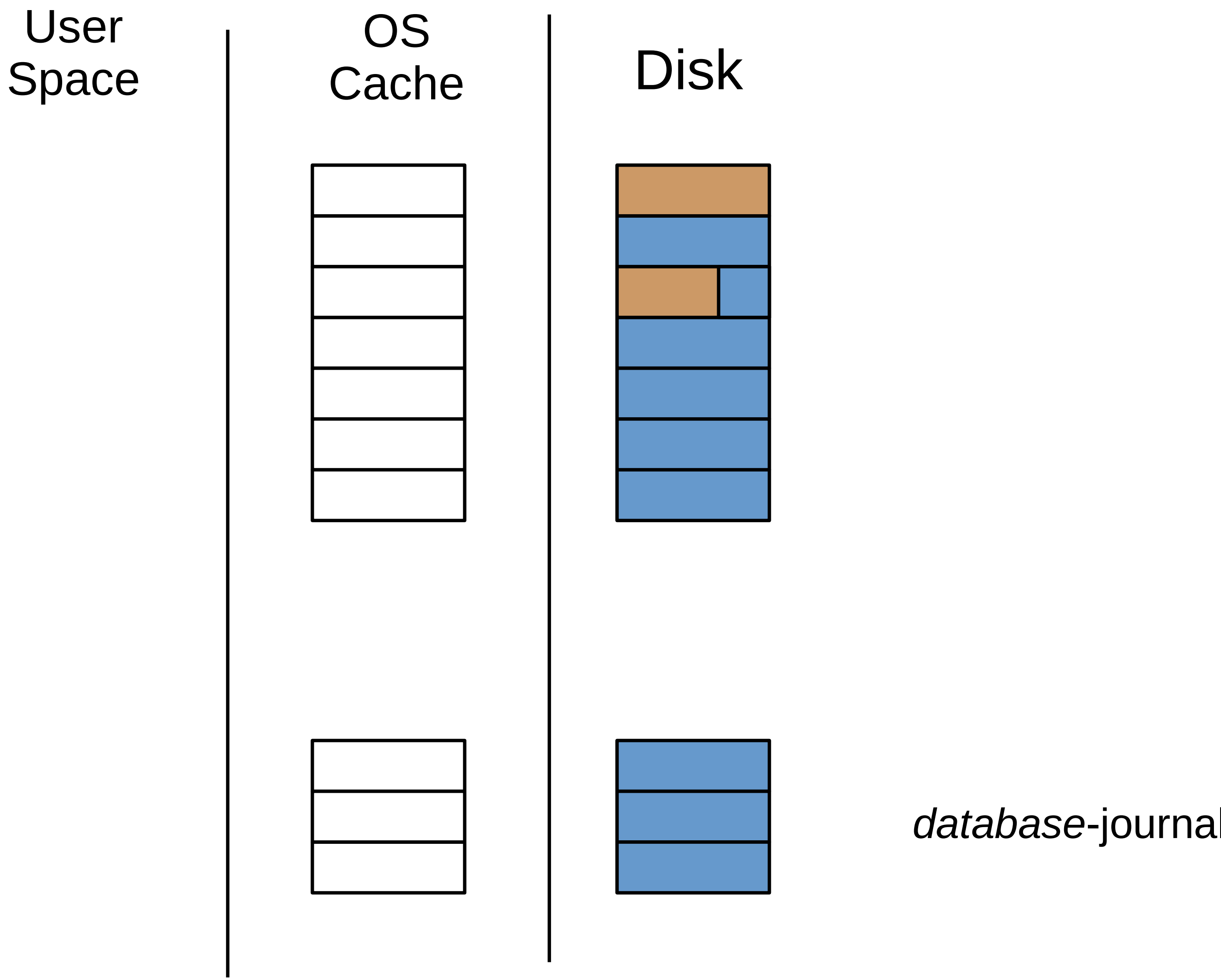


→ *Disable this step using PRAGMA synchronous=OFF*

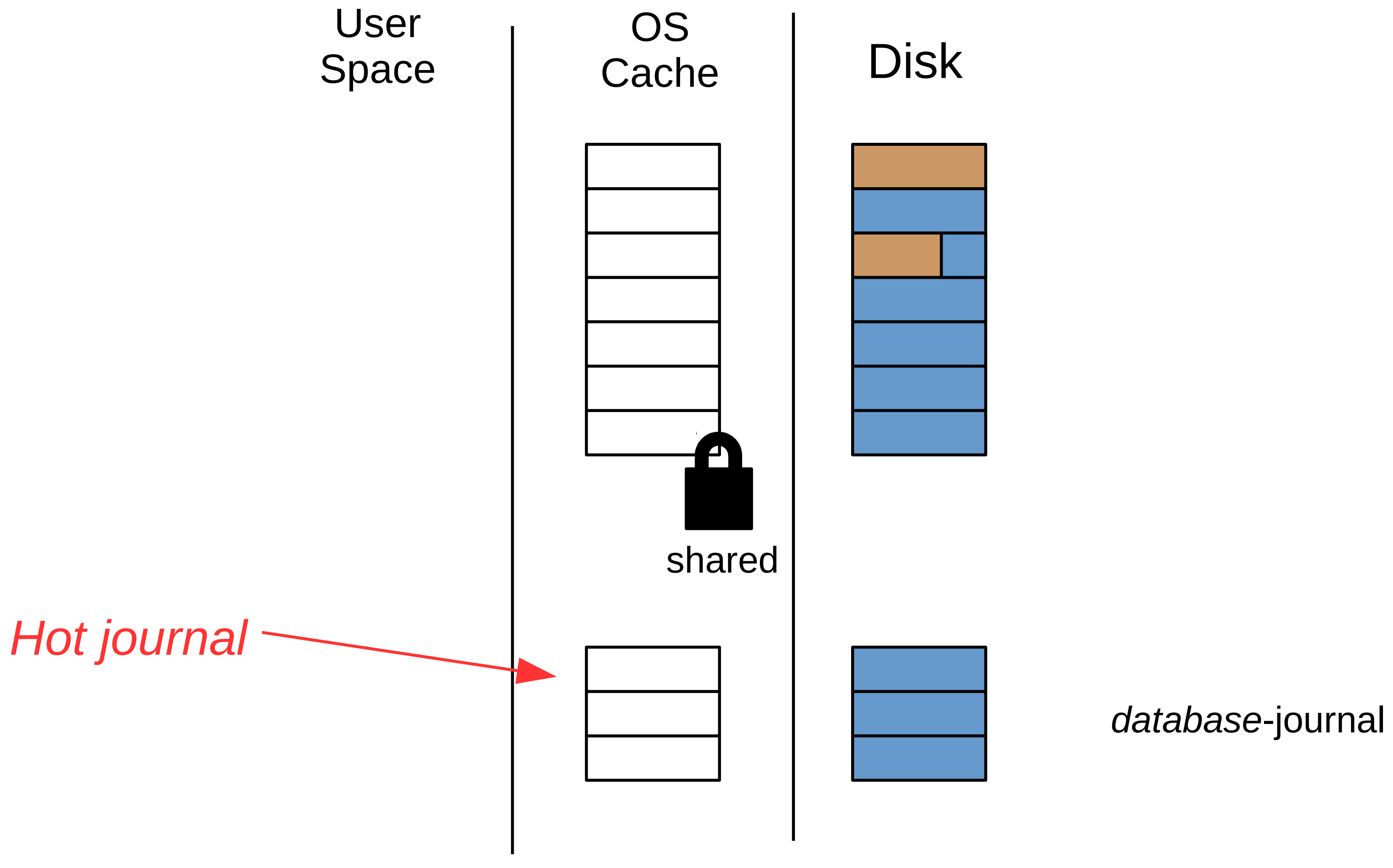
Rollback Journaling



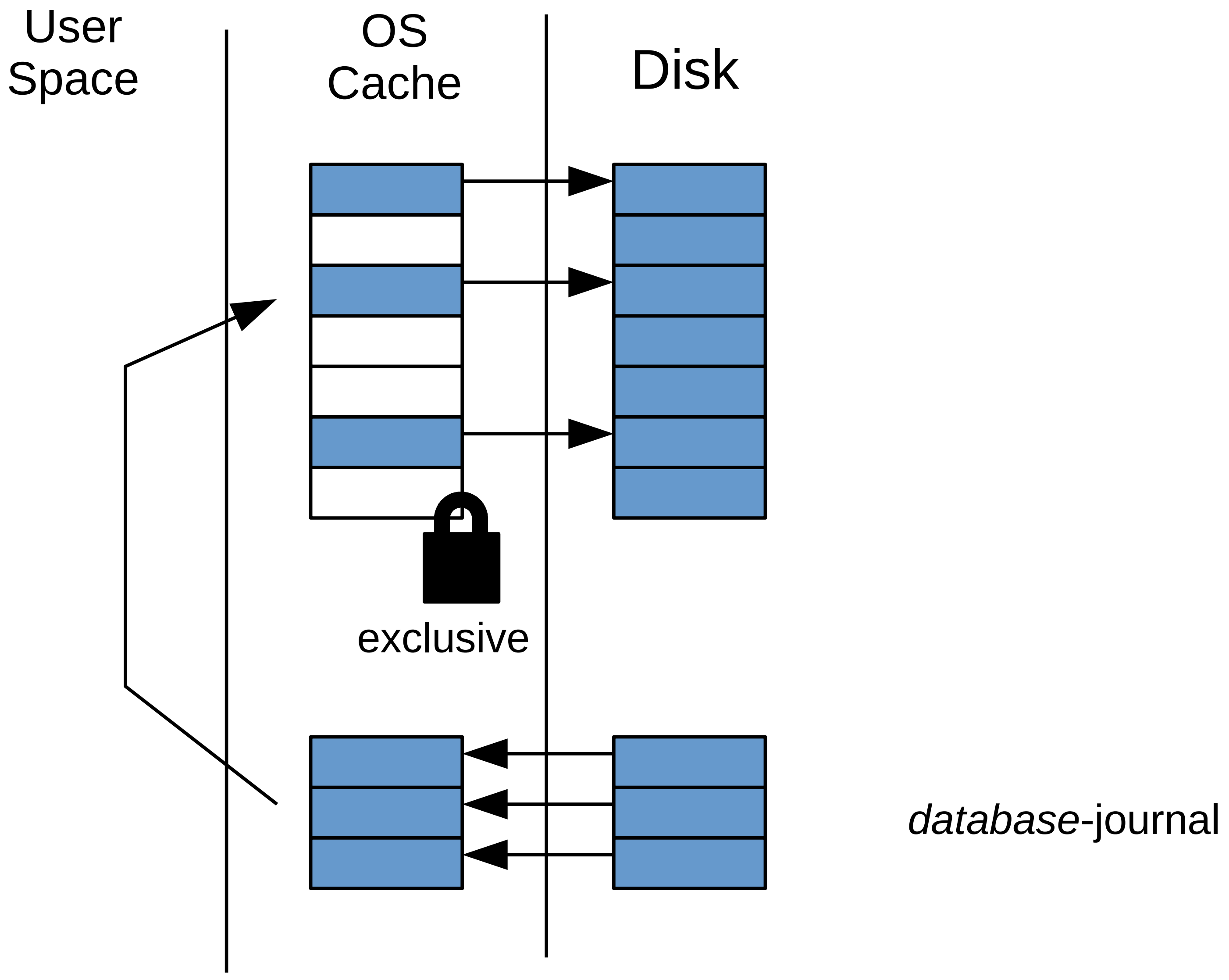
Rollback Mode Crash Recovery



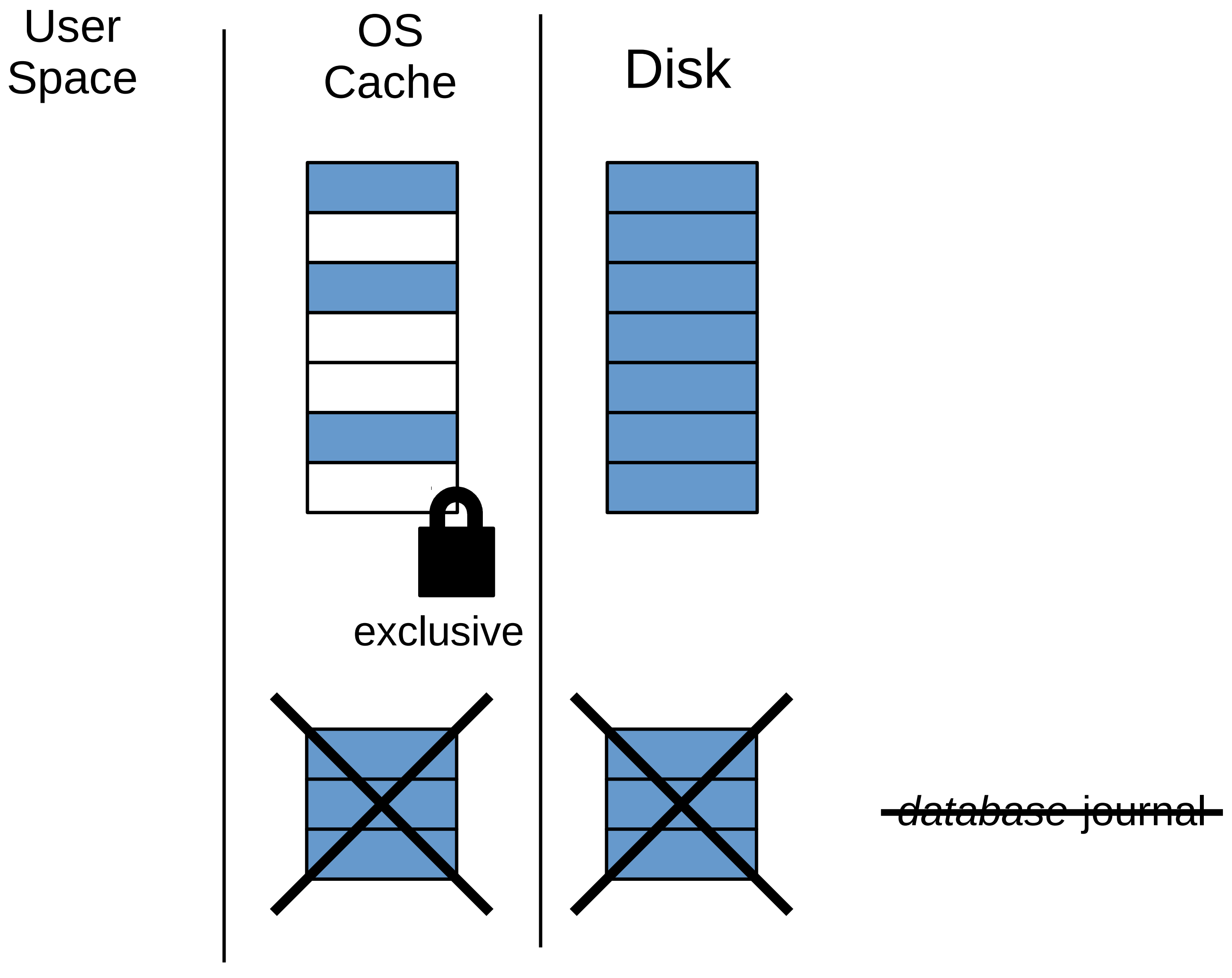
Rollback Mode Crash Recovery



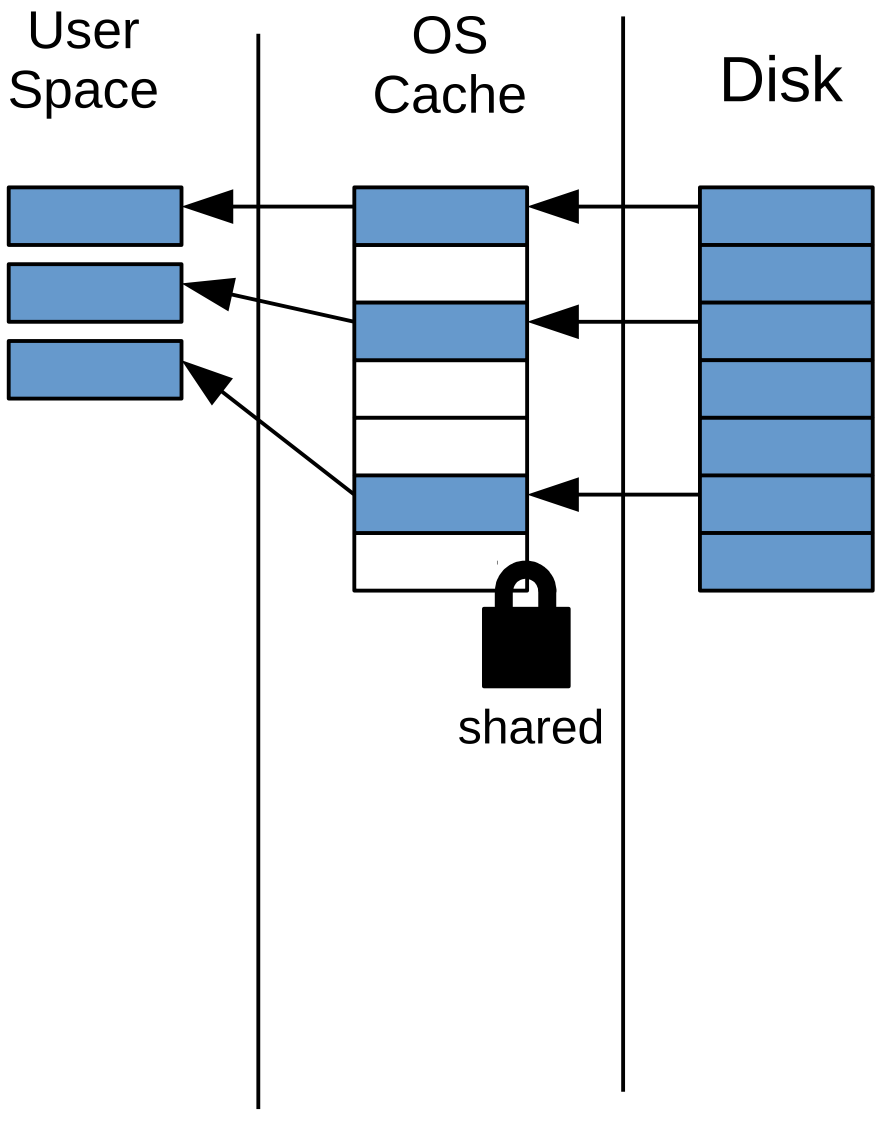
Rollback Mode Crash Recovery



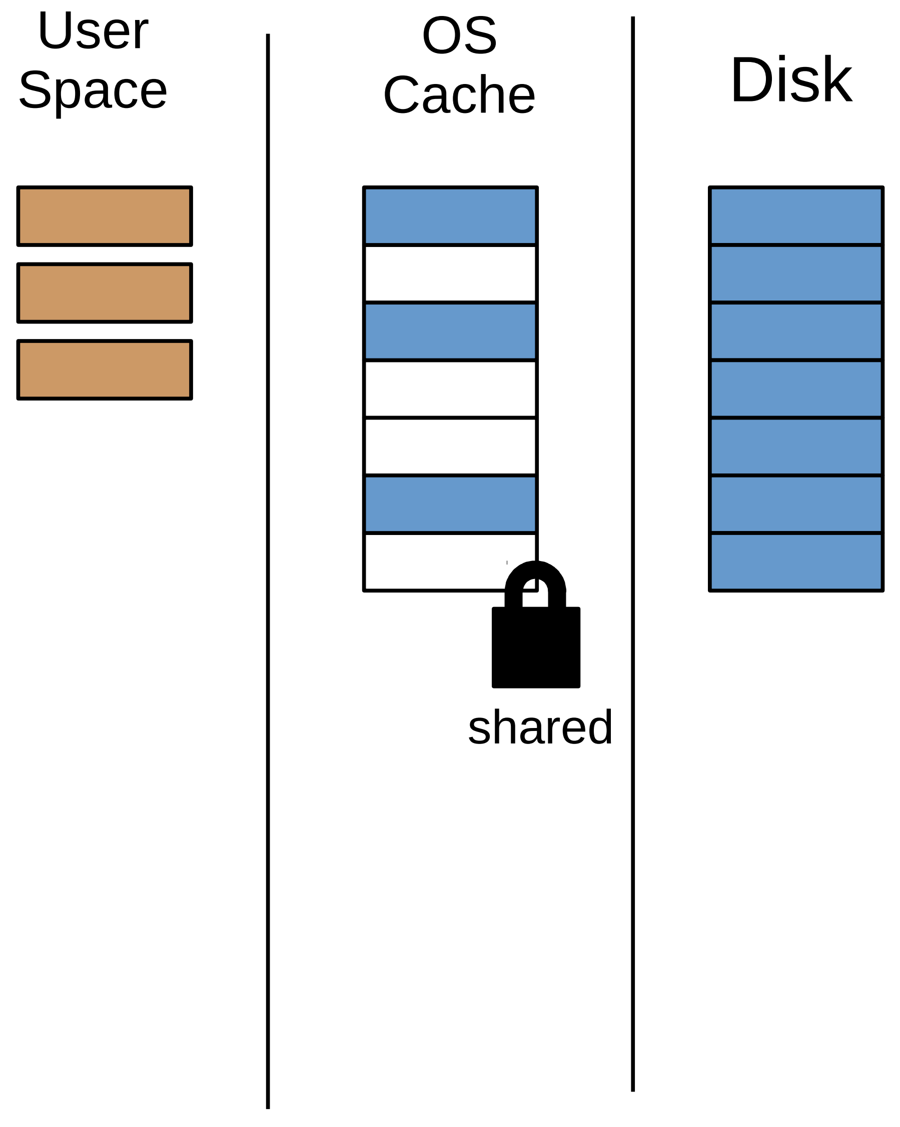
Rollback Mode Crash Recovery



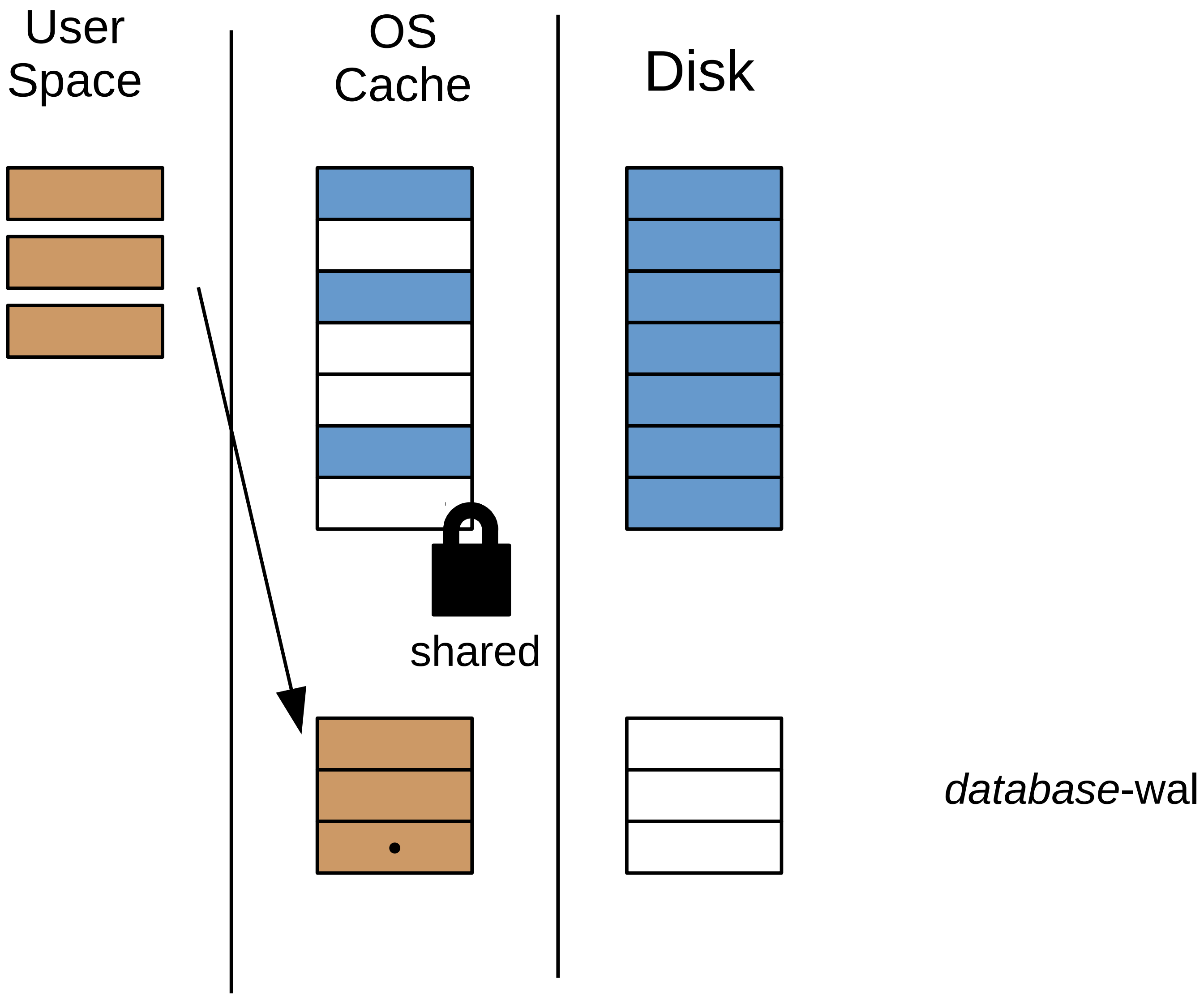
Write-Ahead Log



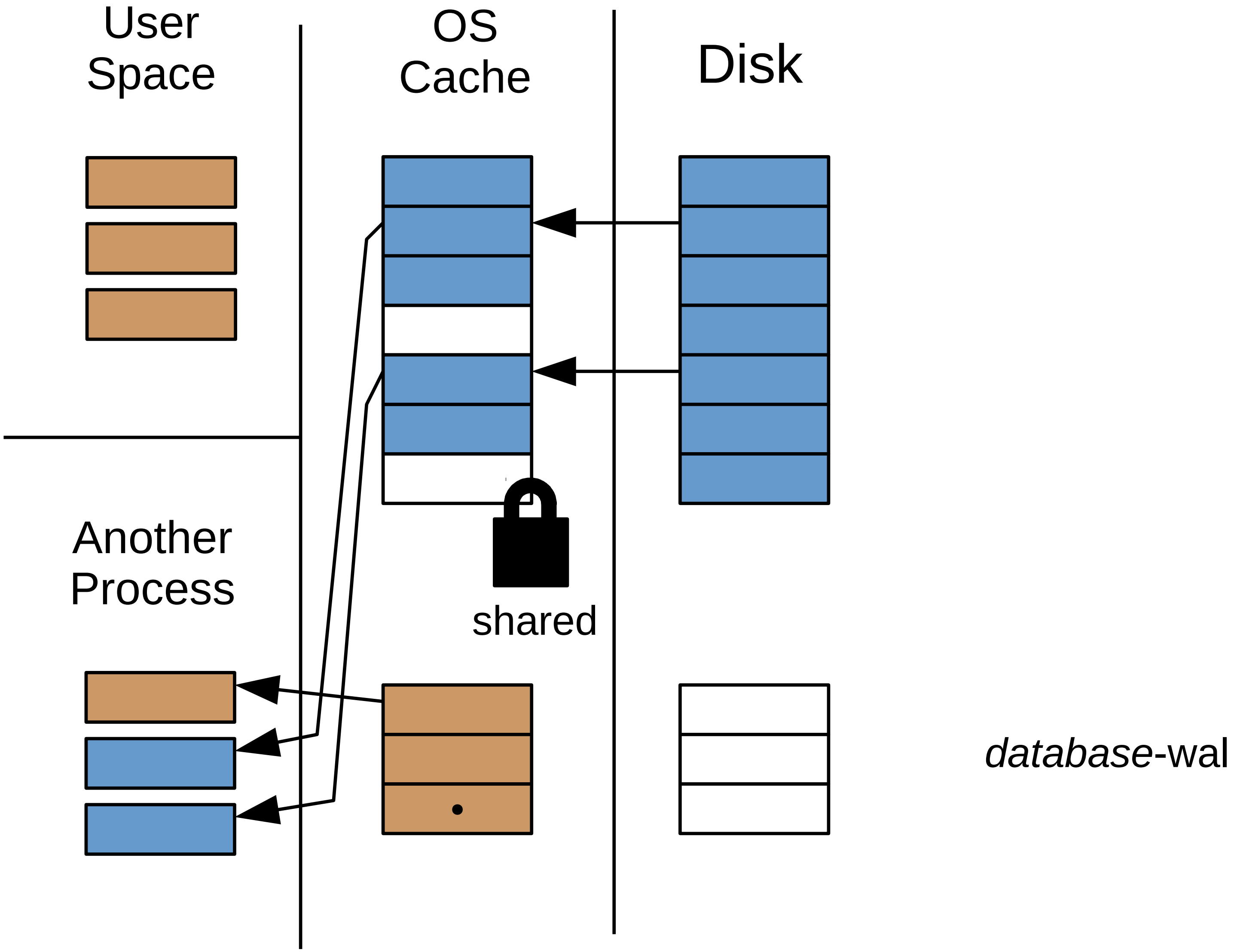
Write-Ahead Log



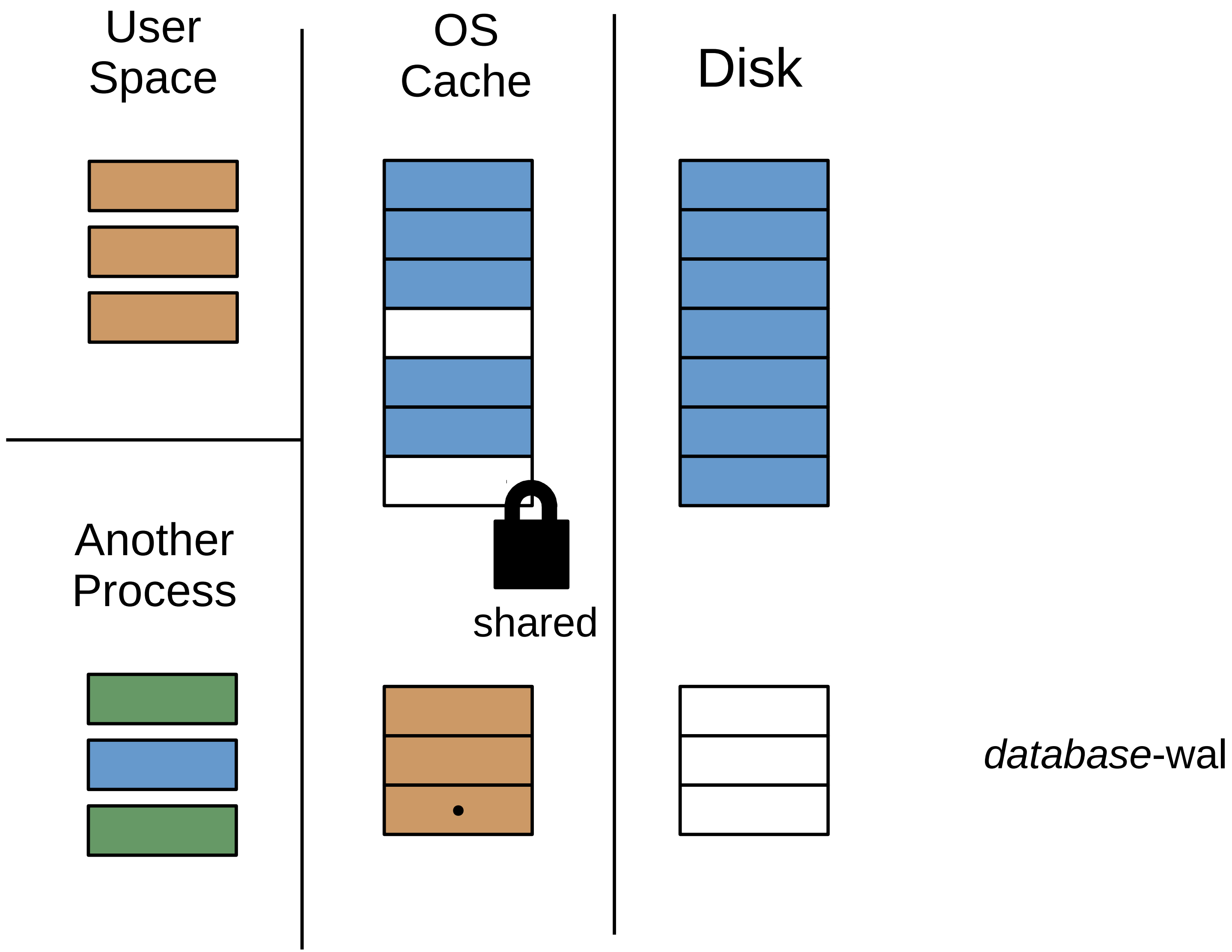
Write-Ahead Log



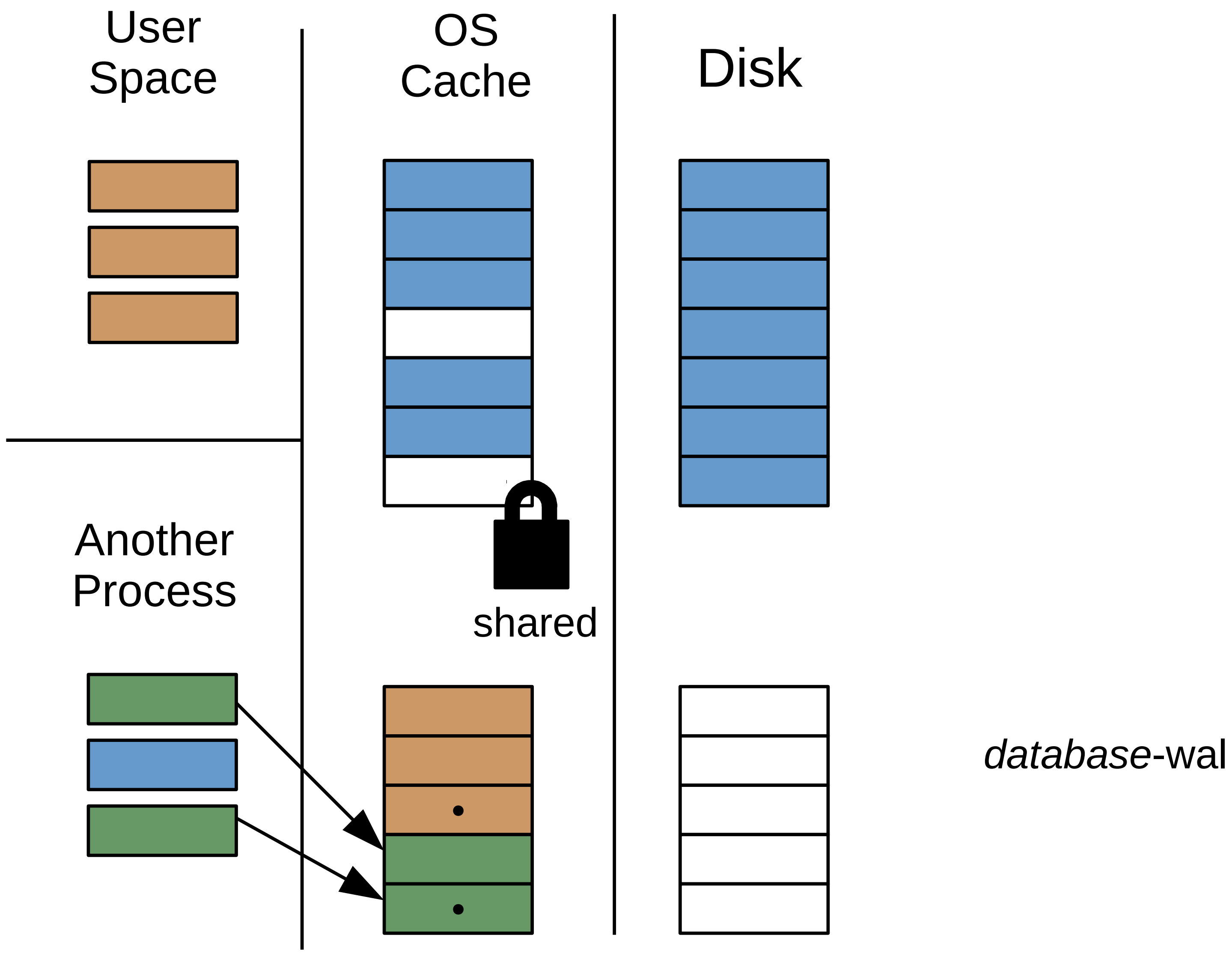
Write-Ahead Log



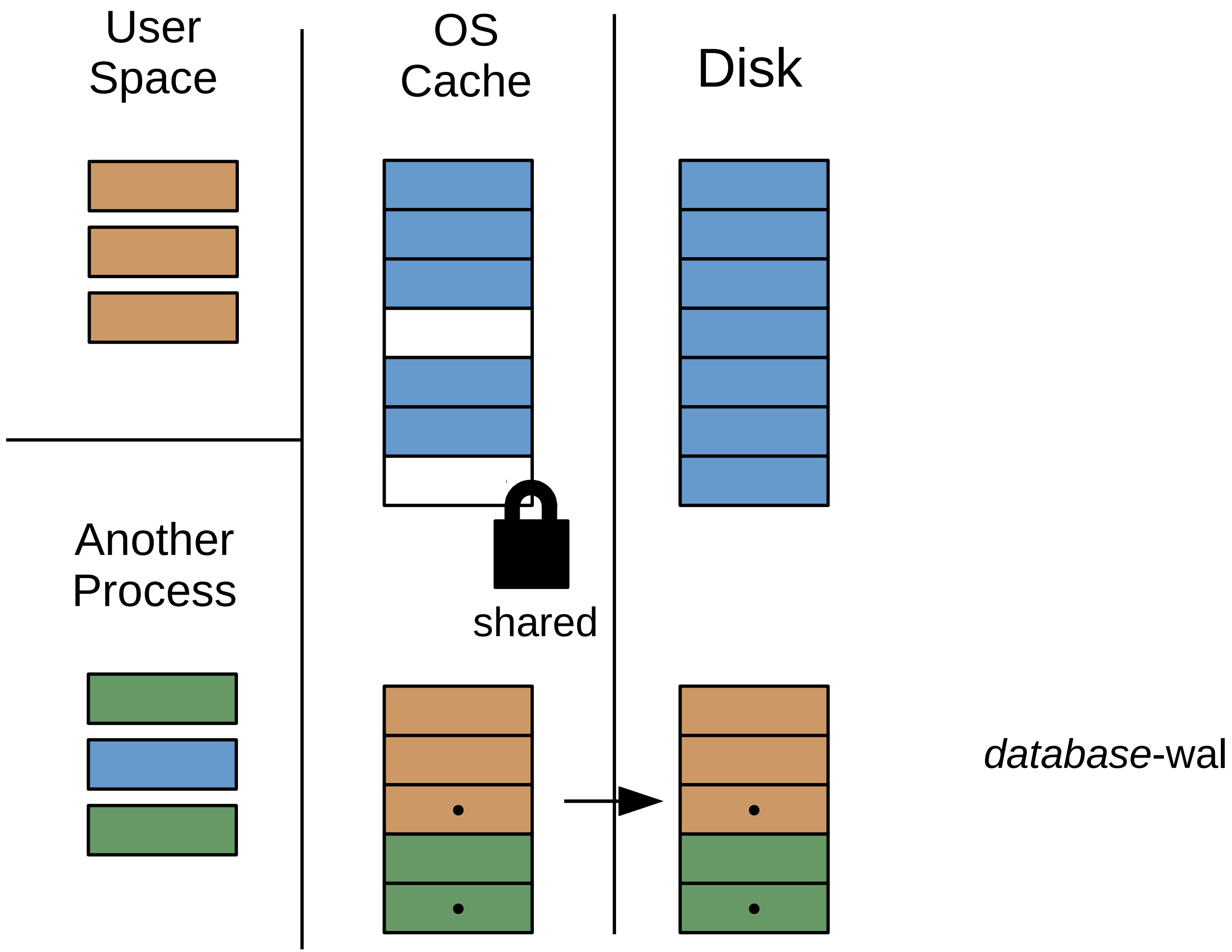
Write-Ahead Log



Write-Ahead Log

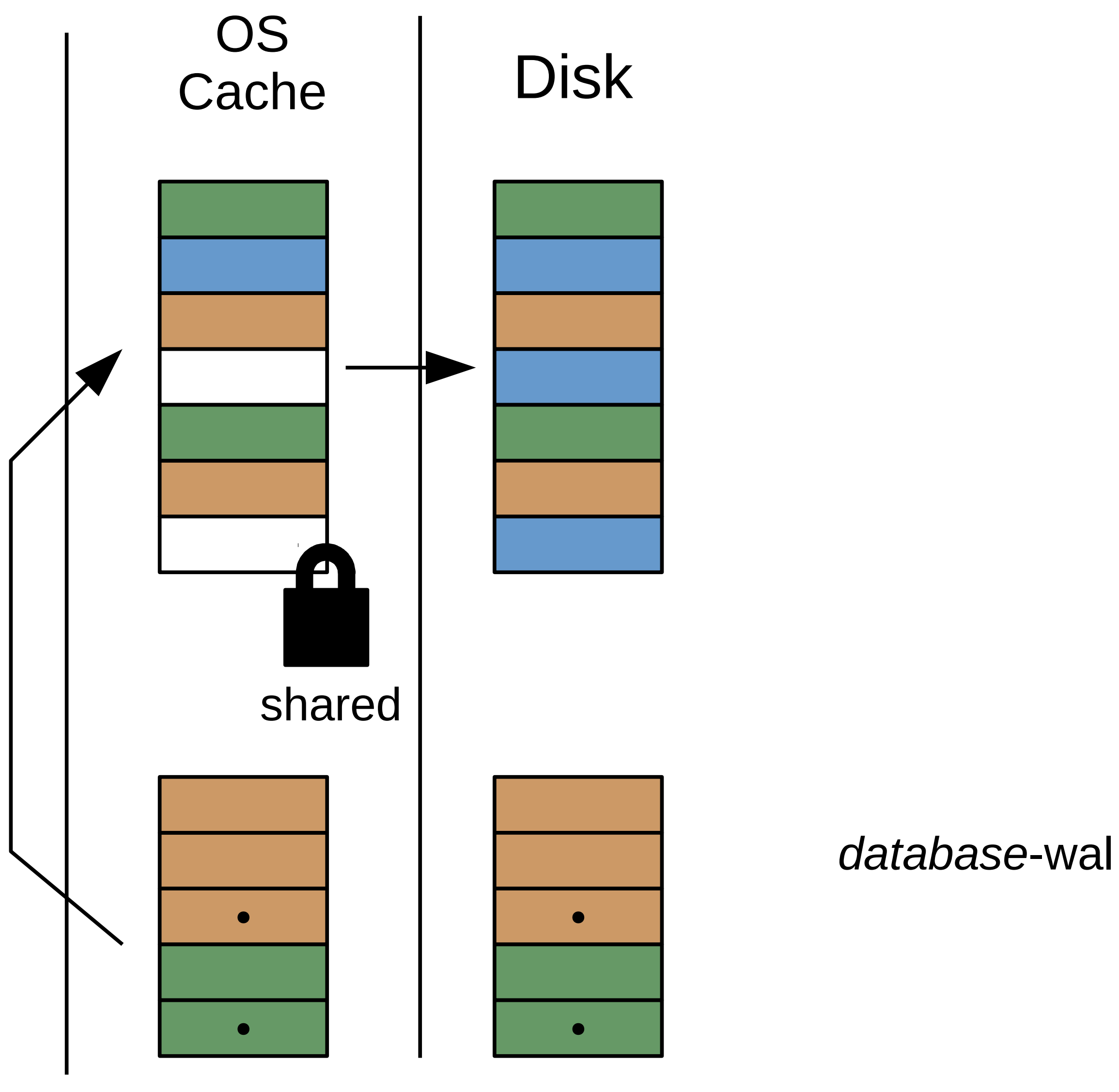


Write-Ahead Log

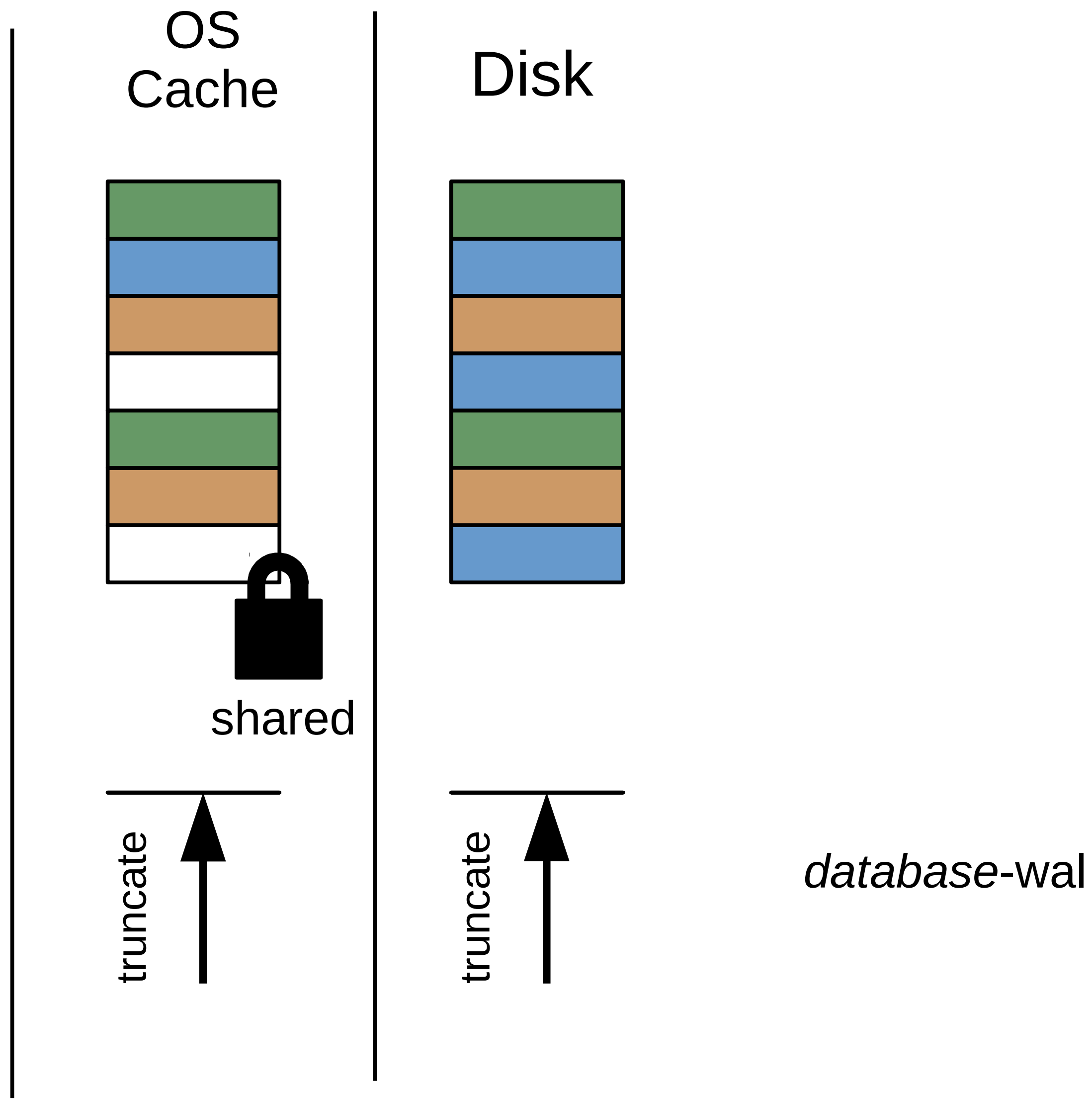


✔ Done for every commit with PRAGMA synchronous=FULL

Checkpoint

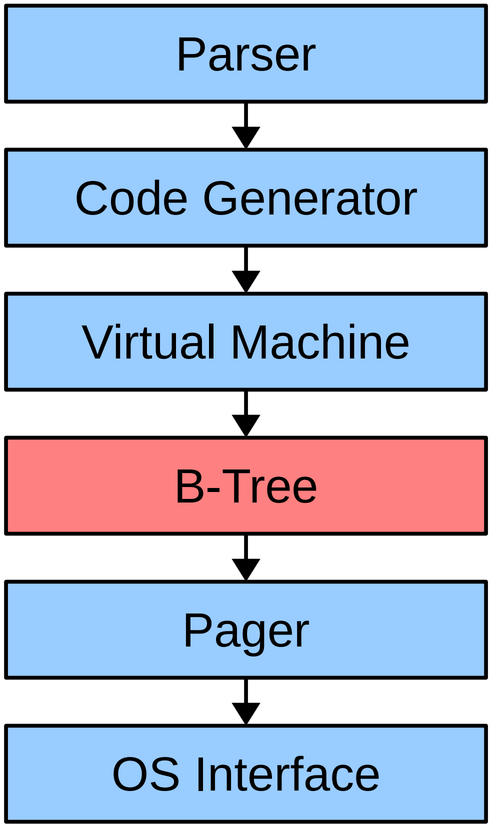


Checkpoint

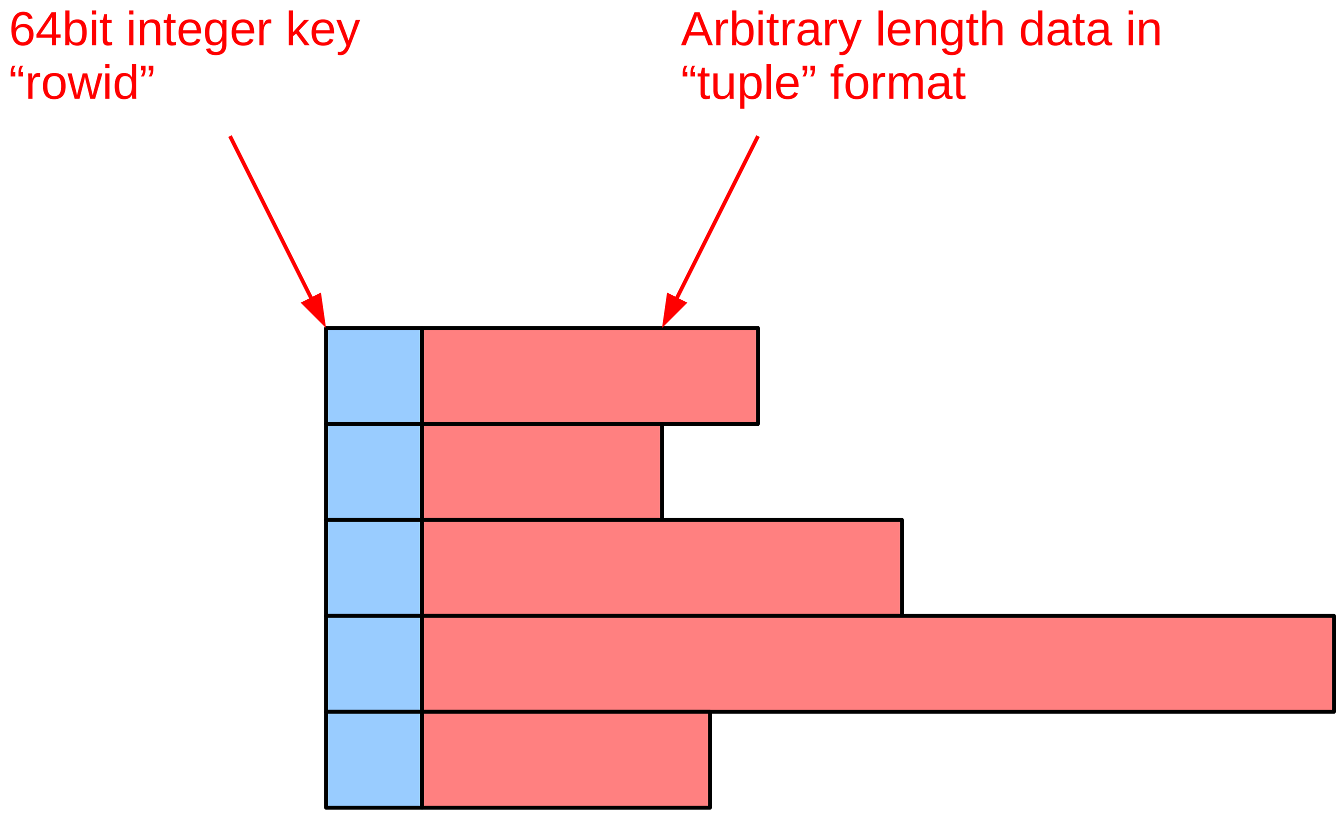


The B-tree Layer

- 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

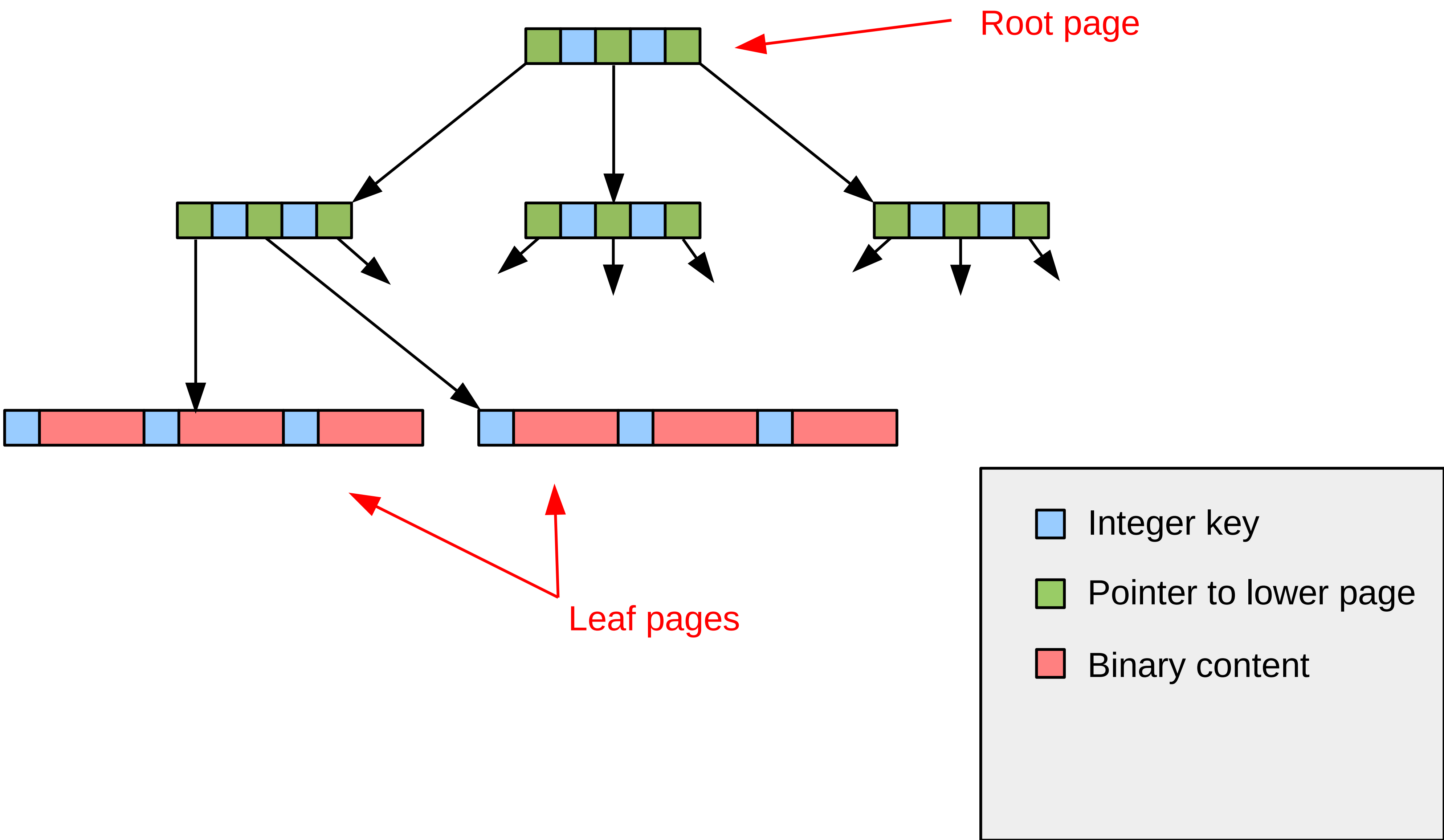


Logical View of SQL Table Storage

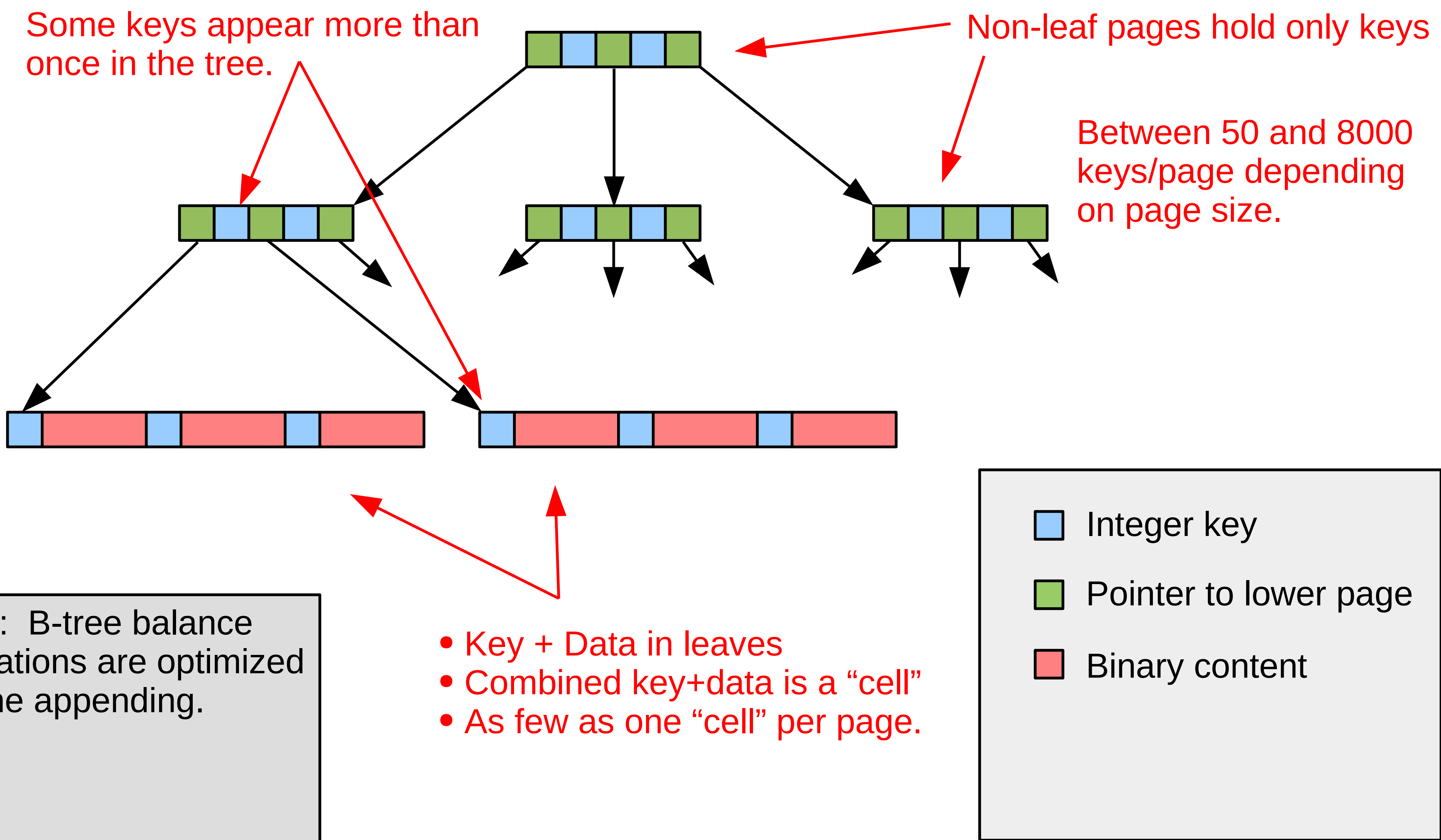


B+tree Structure

(used by most SQL tables)

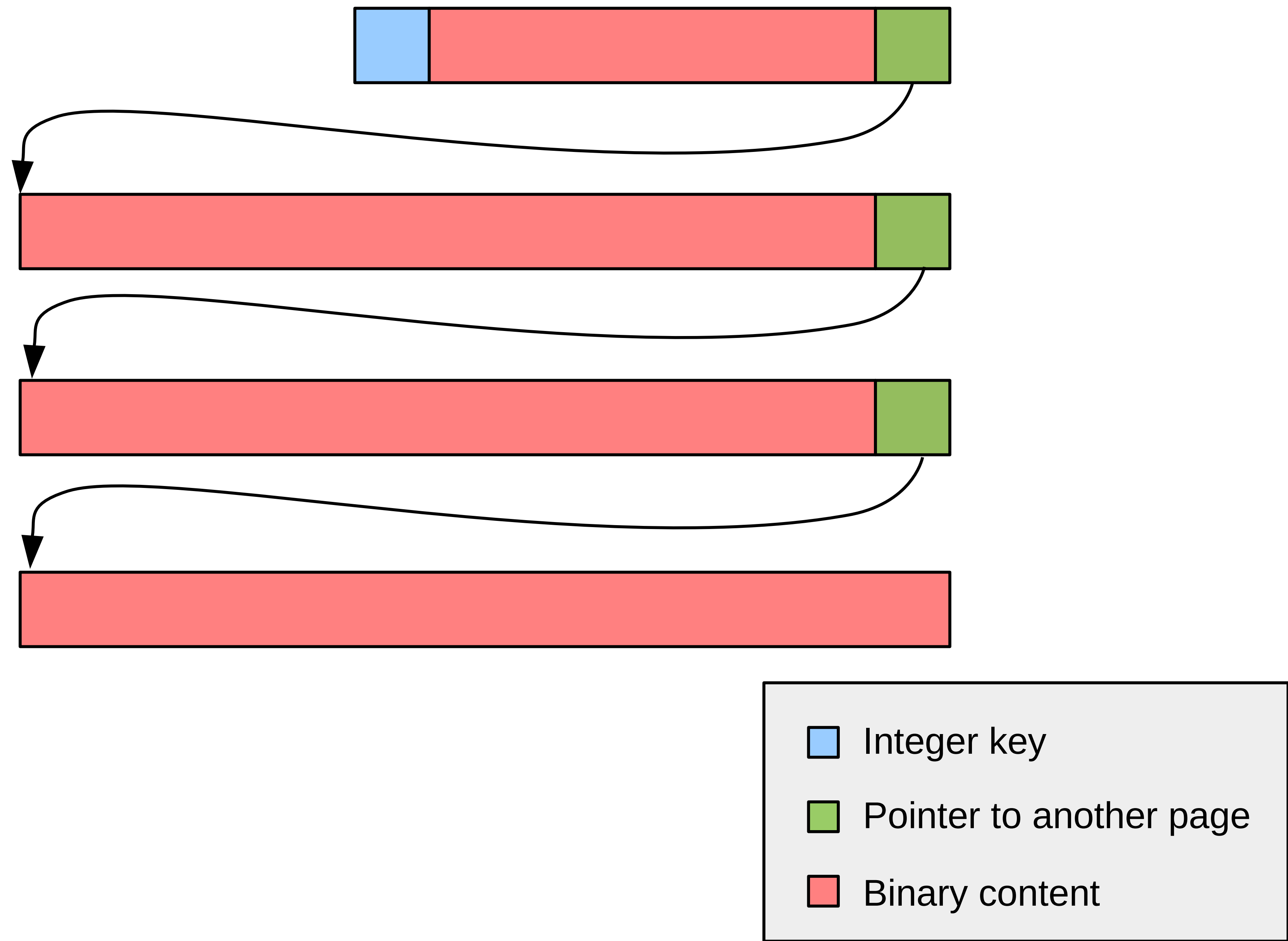


B+tree Structure
(used by most SQL tables)



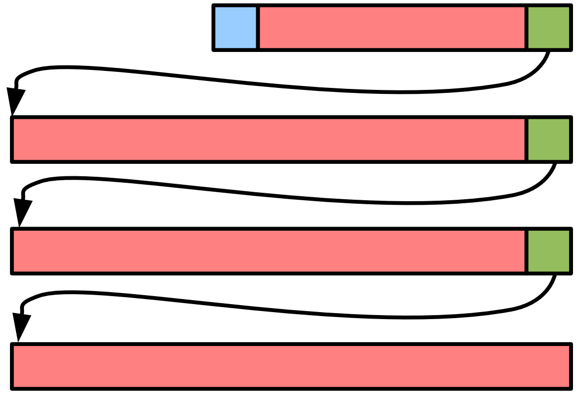
Note: B-tree balance operations are optimized for the appending.

Overflow

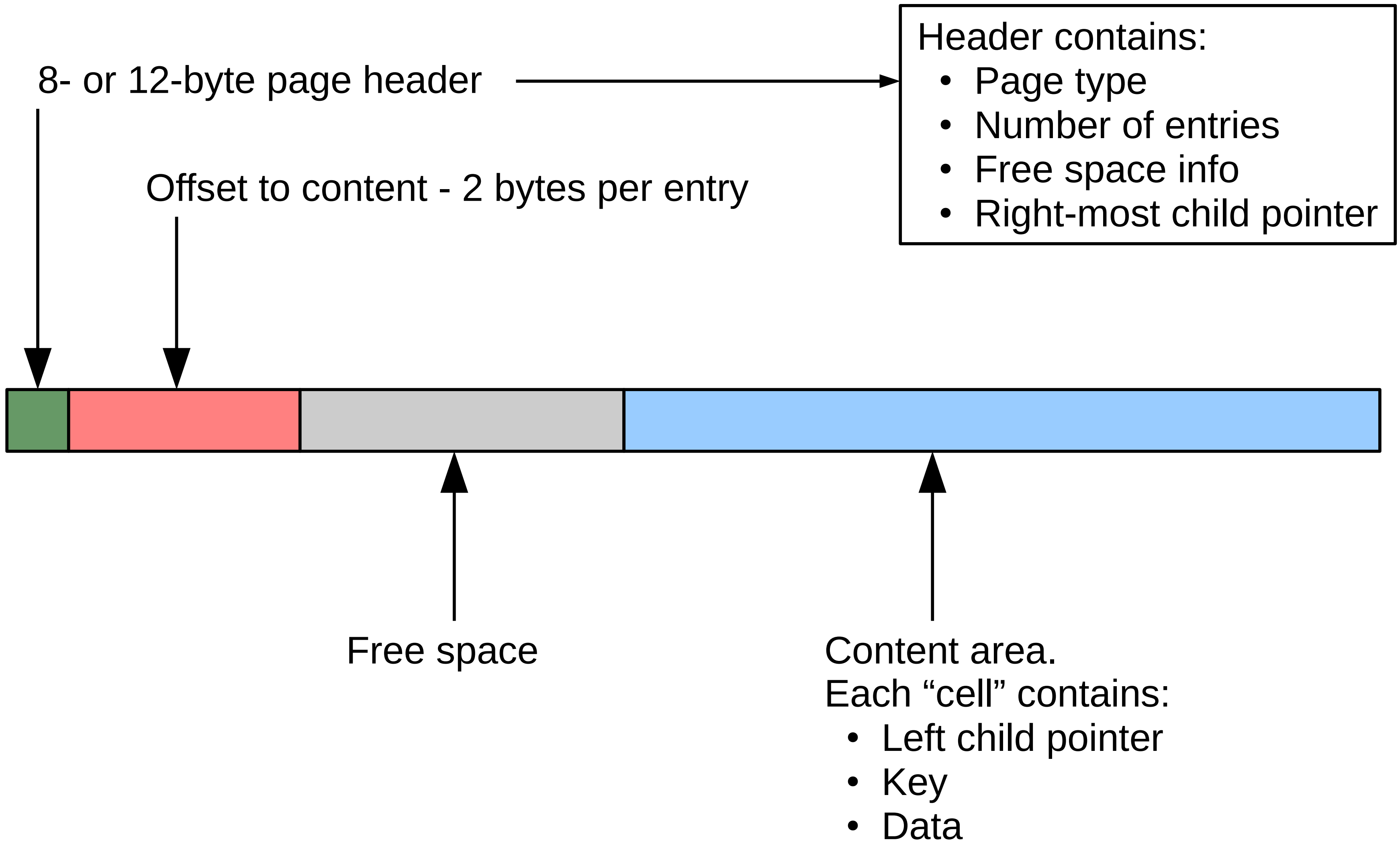


Surprising Attributes of Overflow

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



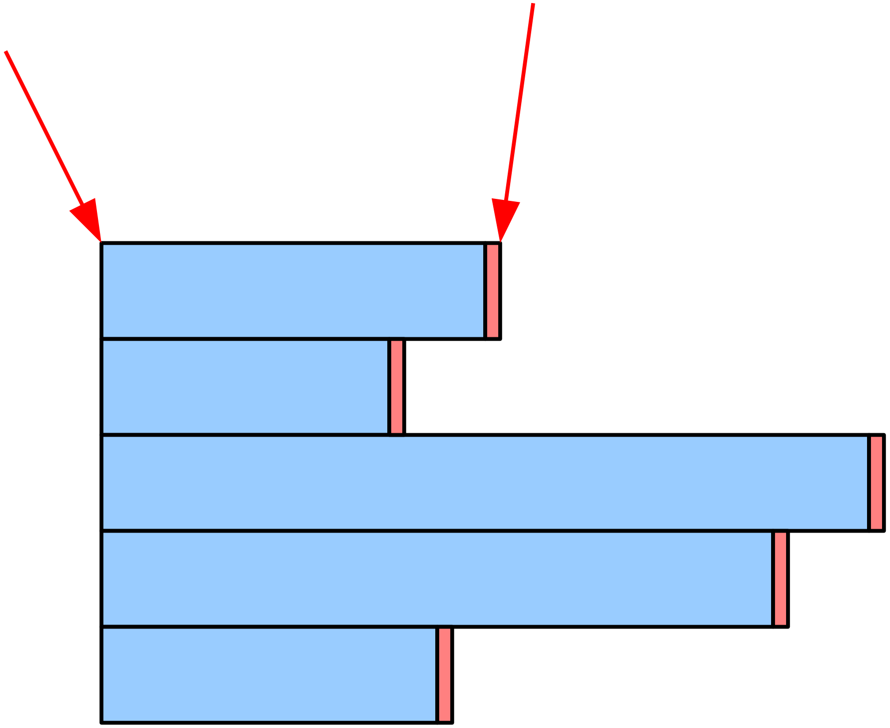
B-tree Page Layout



Logical View of SQL Index Storage

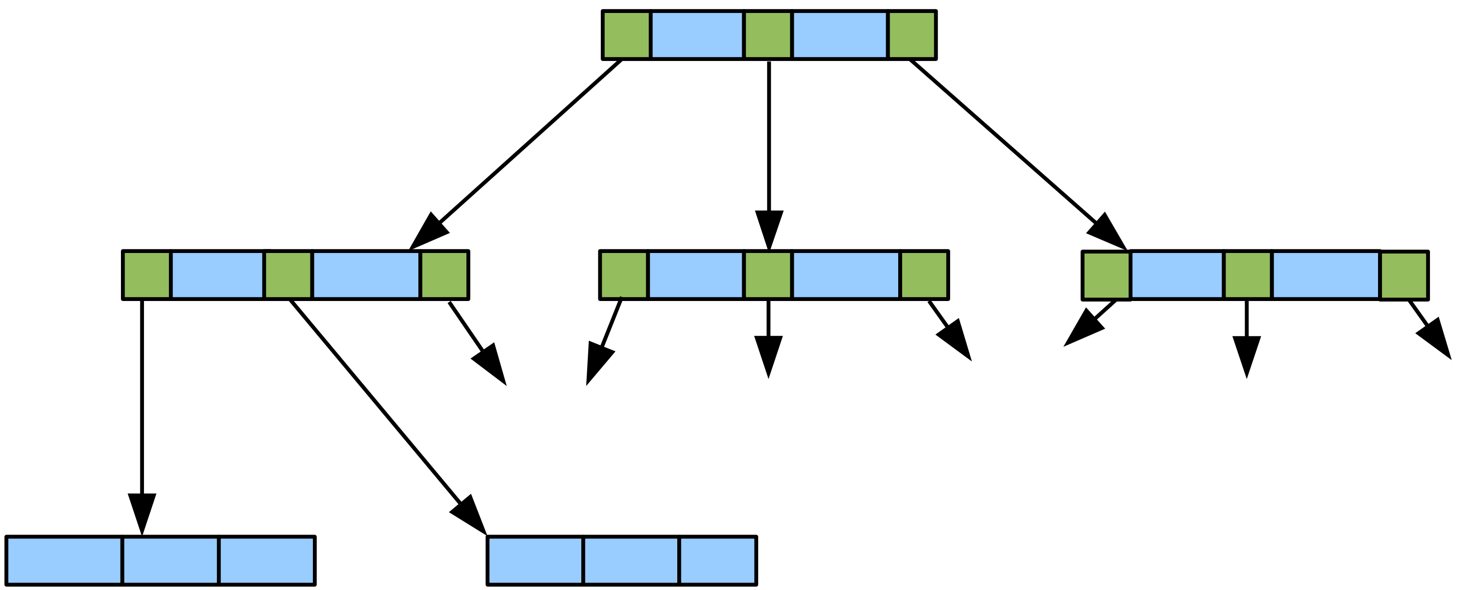
- Arbitrary length key.
- External comparison function
- “Tuple” format

- No data
- The key is the data

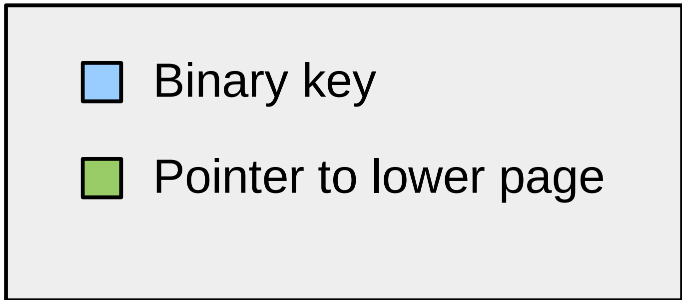


B-tree Structure

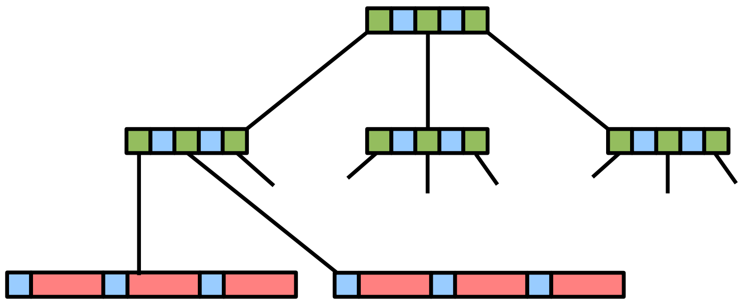
(used by indexes & WITHOUT ROWID tables)



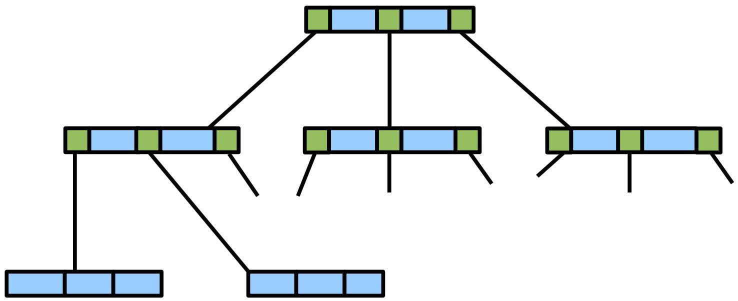
- Key only. No data. The key is the data.
- Larger binary keys, hence lower fan-out
- Each key appears in the table only once
- Minimum 4 keys per page



B-tree Types

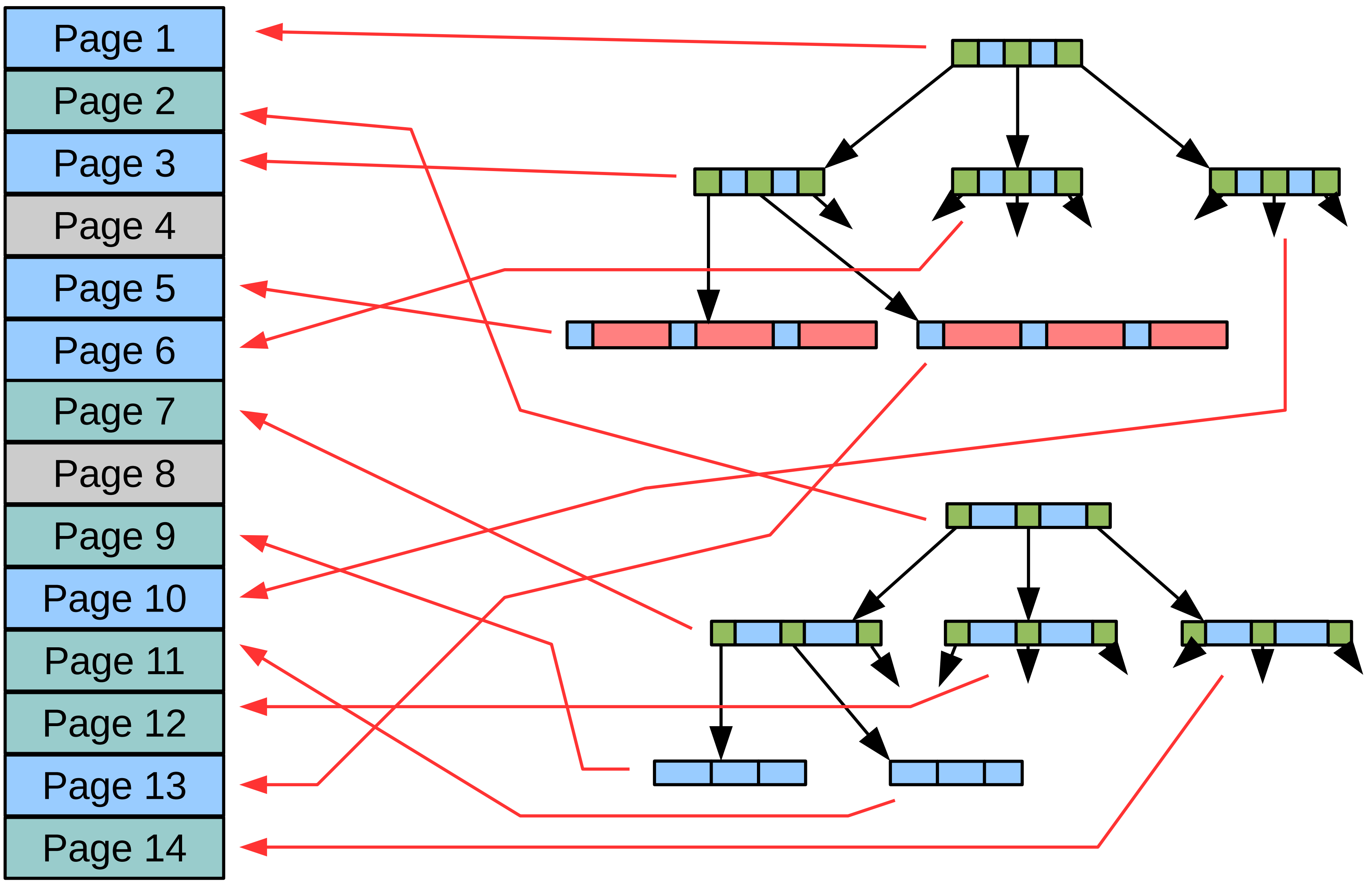


- SQL tables
- Integer keys
- Data in leaves
- Some keys on more than one page



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

Mapping B-trees Into Pages



sqlite_schema

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

✔ *sqlite_schema always rooted at page 1*

sqlite_schema

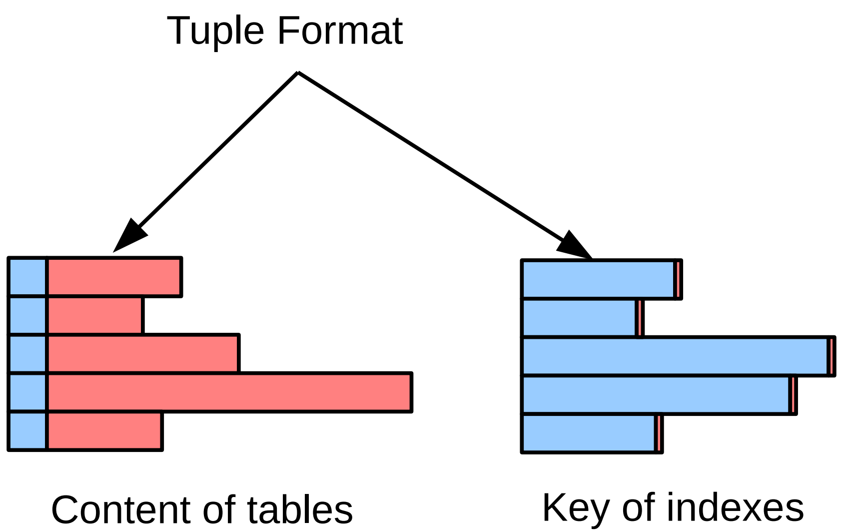
```
sqlite> CREATE TABLE t1(x);
sqlite> .mode qbox
sqlite> SELECT * FROM sqlite_schema;
```

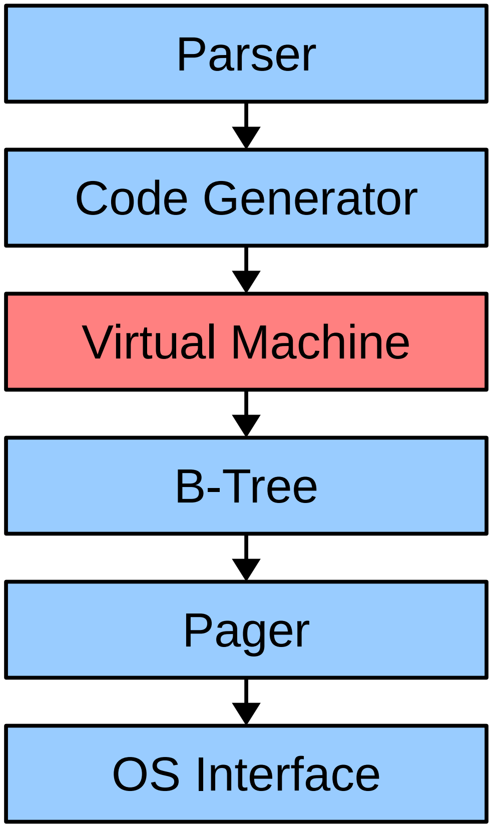
type	name	tbl_name	rootpage	sql
'table'	't1'	't1'	2	'CREATE TABLE t1(x)'

Try this yourself at https://sqlite.org/fiddle

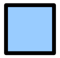
Virtual Machine

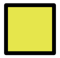
- Bytecode interpreter
- Defines the "tuple format"

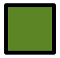


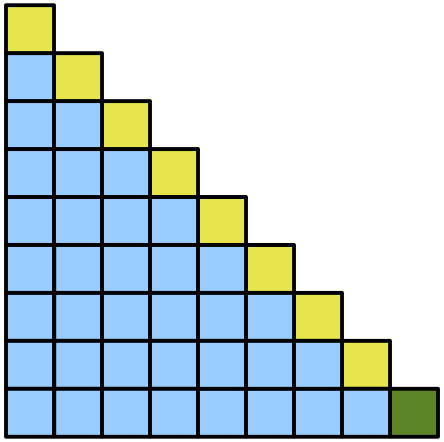


Variable Length Integers

 1xxxxxxx - high bit set. 7 bits of data

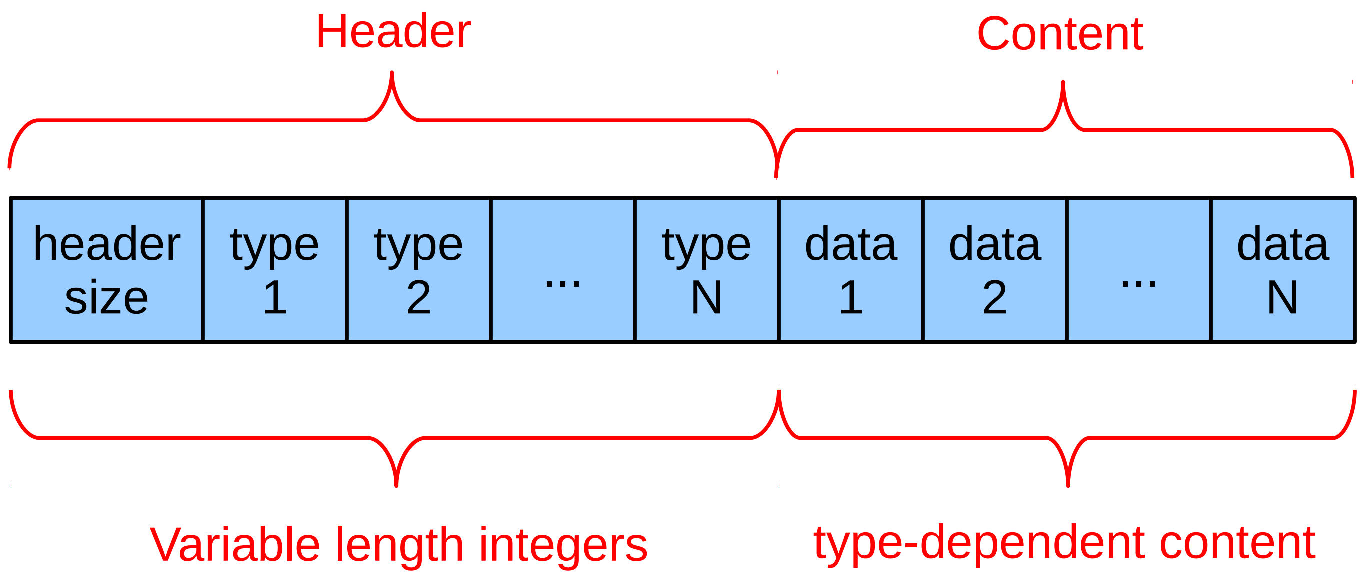
 0xxxxxxx - high bit clear. 7 bits of data

 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

Tuple Format

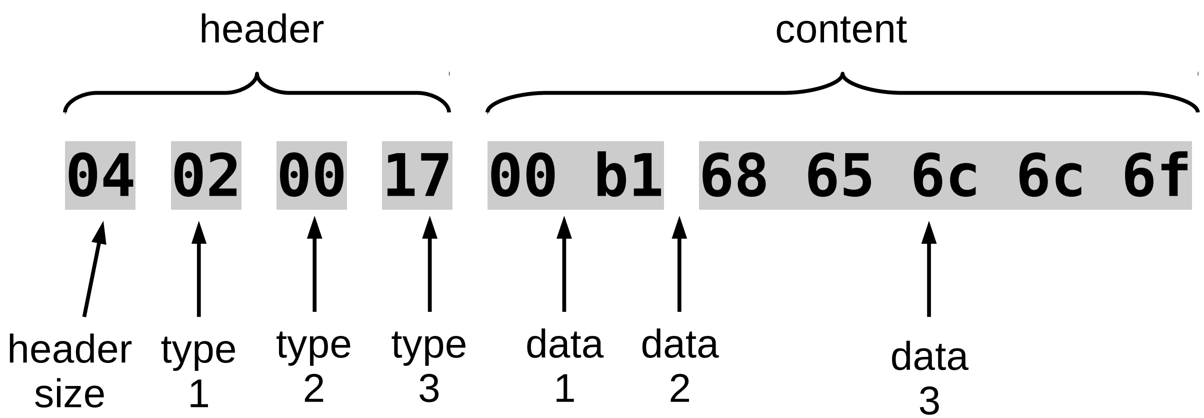


Integer Type Codes

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

Tuple Format Example

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



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

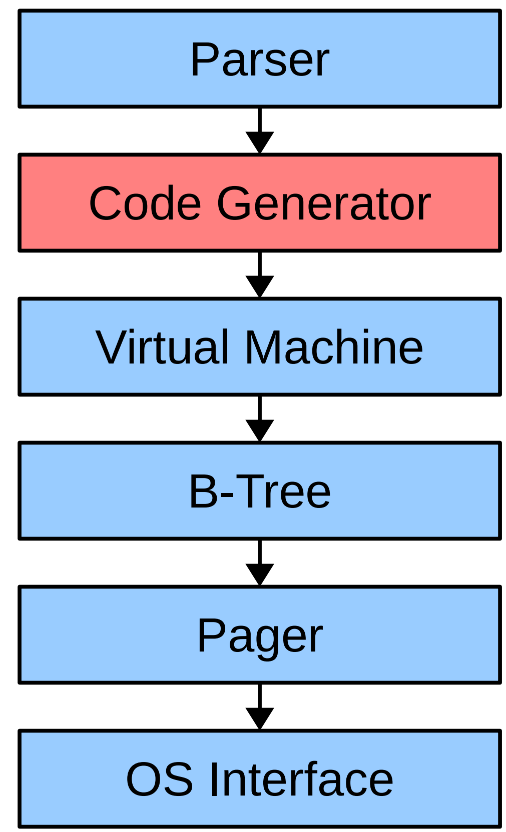
Code Generator

- AST transformations
- Join order determination
- Index selection
- Bytecode generation

select.c

where*.c
whereInt.h

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



AST Transformations

- Resolve table and column names and expand VIEWs
- Expand “*” in “SELECT * FROM ...”
- Move all constraints into the WHERE clause
- “Flatten” subqueries into outer queries
- Push WHERE clause terms in outer queries down into subqueries
- Outer join strength reduction
- And so forth....

Move Constraints Into WHERE

- NATURAL JOIN → JOIN USING
- JOIN USING → JOIN ON
- ON → WHERE

Terms that originate in the ON clause of OUTER joins have special flags set in the AST so that they get used appropriately.

Subquery Flattening

```
SELECT t1.a, t2.b
   FROM t2, (SELECT x+y AS a FROM t1 WHERE z<100)
WHERE a>5;
```

```
SELECT t1.x+t1.y AS a, t2.b
   FROM t2, t1
WHERE z<100 AND t1.x+t1.y>5;
```

WHERE clause push-down

```
CREATE VIEW v1(a,b) AS SELECT distinct a, b FROM t1;

SELECT x, y, b FROM t2, v1
 WHERE x=a AND b>7;
```

Expand view V1

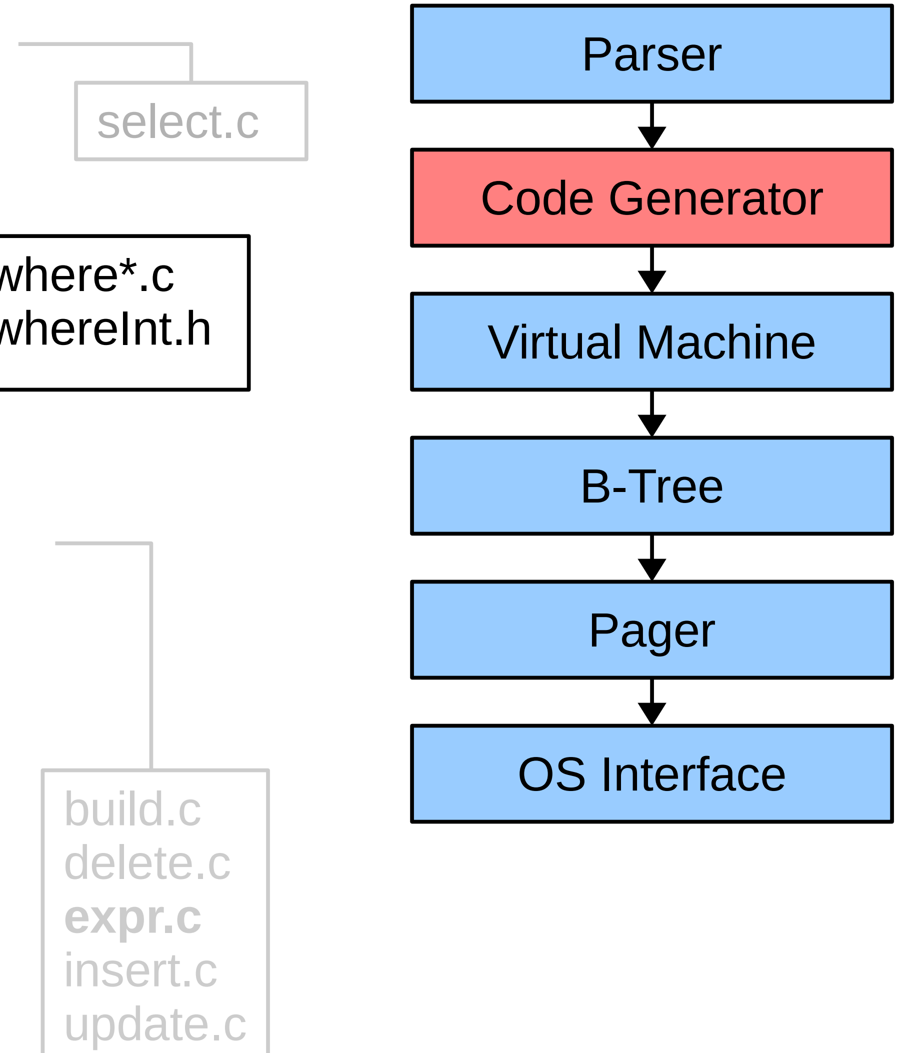
```
SELECT x, y, b FROM t2, (SELECT distinct a,b FROM t1)
 WHERE x=a AND b>7;
```

Push down “b>7”

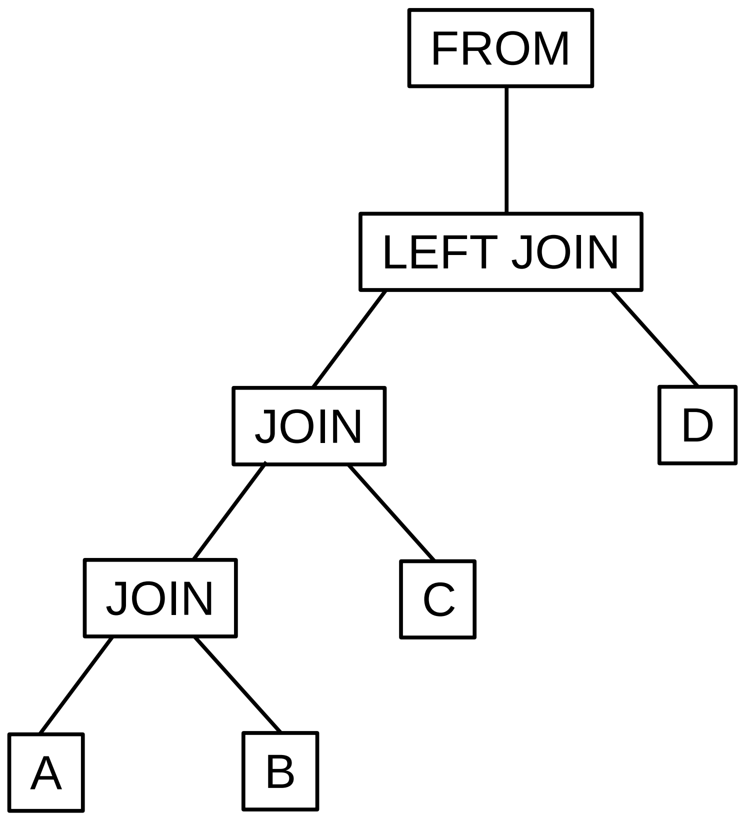
```
SELECT x, y, b FROM t2, (SELECT distinct a,b FROM t1 WHERE b>7)
 WHERE x=a AND b>7;
```

Query Planning

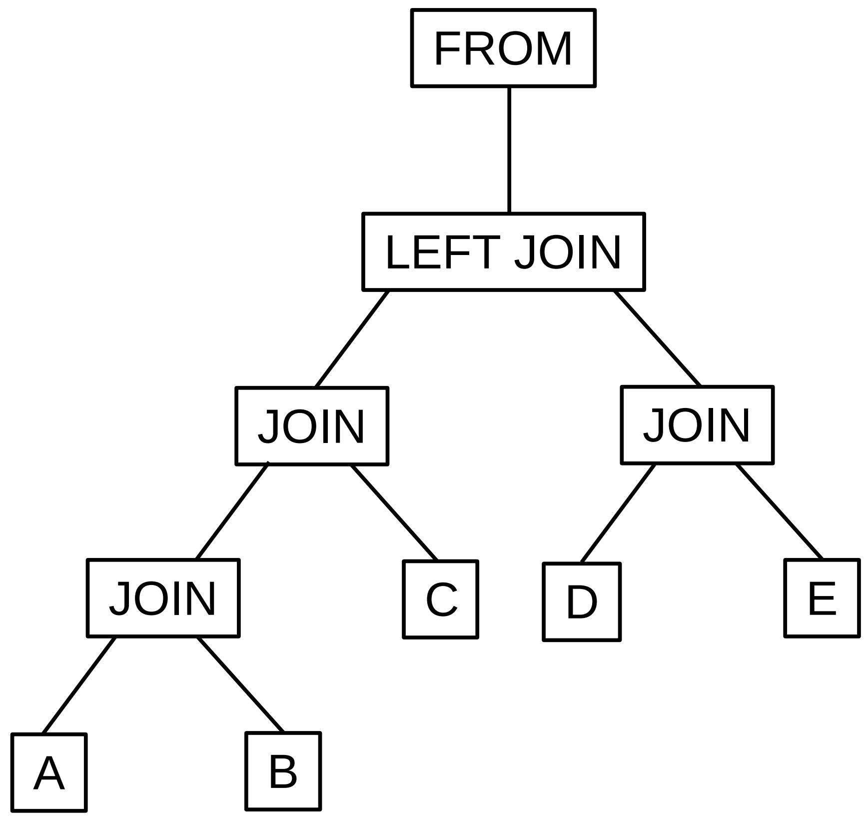
- AST transformations
- Join order determination
- Index selection
- Bytecode generation



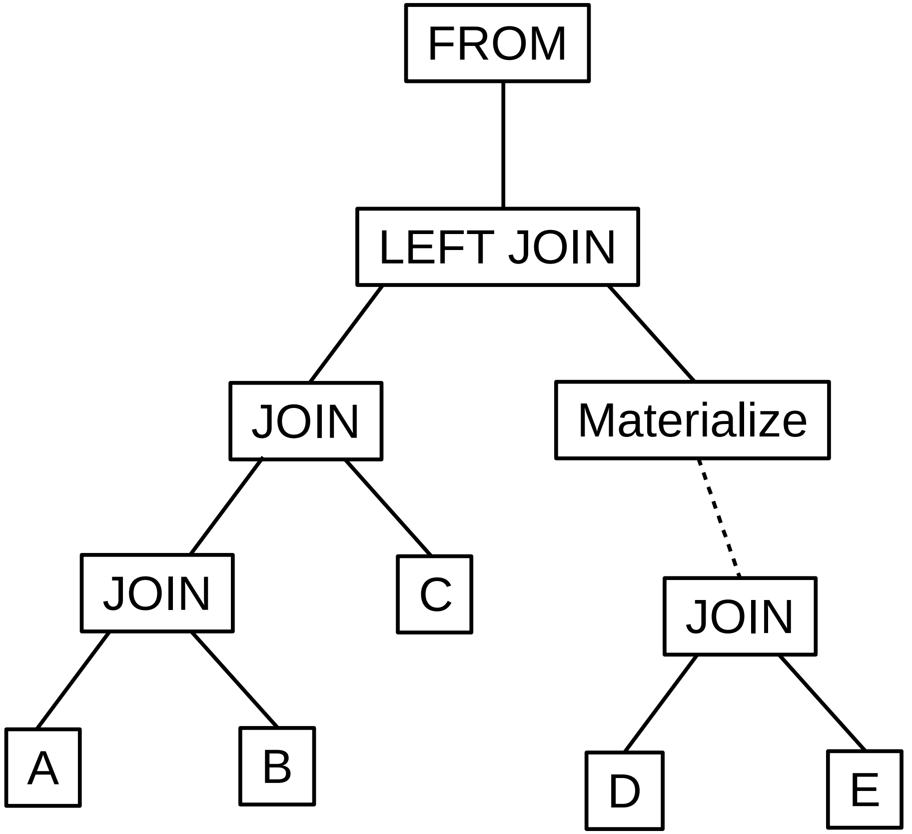
Left-to-Right Grouping Only



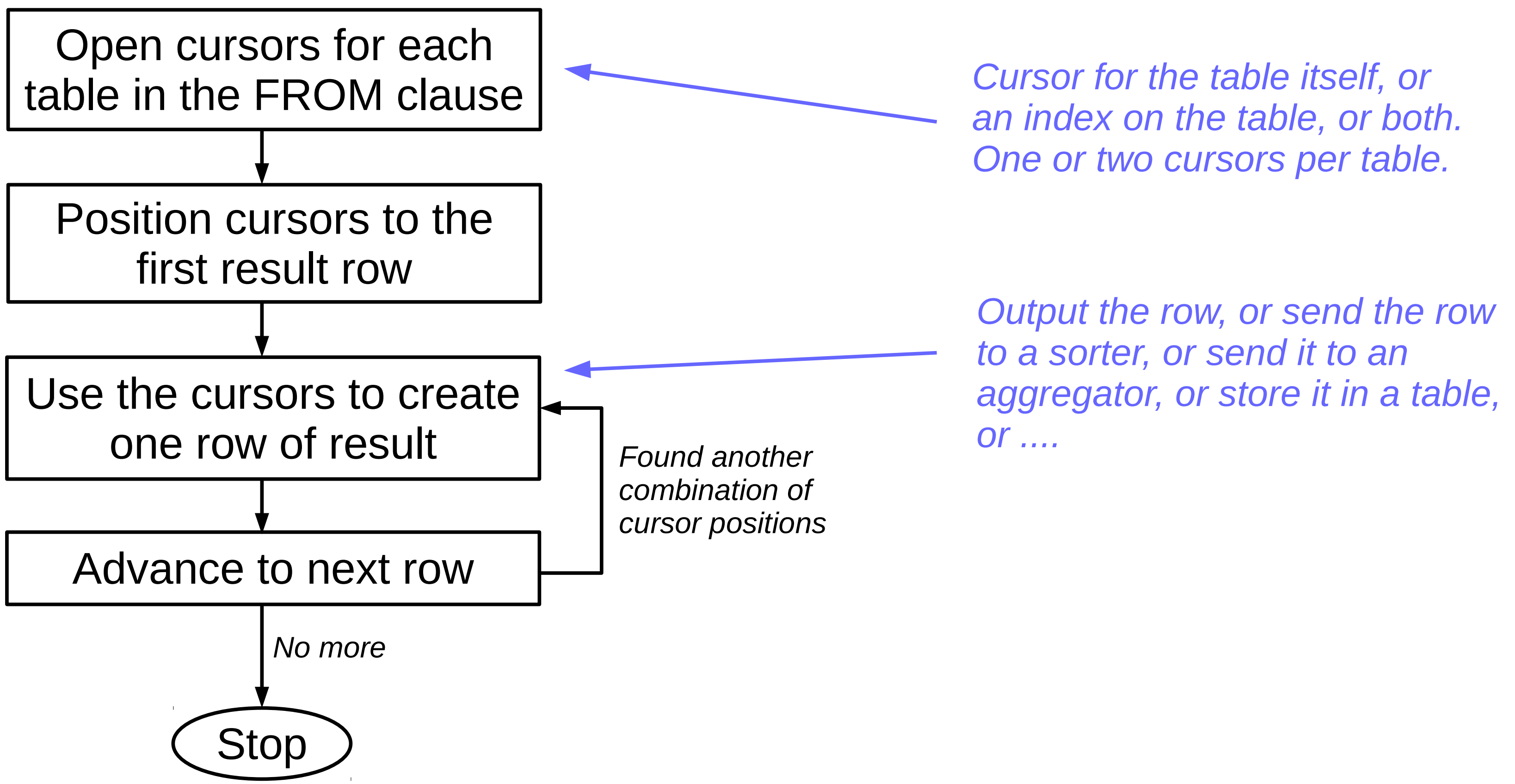
select * from A join B join C left join (D join E)



Transform Into This



Schematic Of The Bytecode We Are Trying To Generate:



To See The Bytecode For The Next Slide:

- Go to https://sqlite.org/fiddle
- Enter:

```
CREATE TABLE t1(a,b);
CREATE TABLE t2(c,d);
CREATE TABLE t3(e,f);
PRAGMA automatic_index=off;
EXPLAIN
SELECT a,c,e
   FROM t1 JOIN t2 ON b=c
                    JOIN t3 ON e=d;
```

Disables query-time indexes

SELECT t1.a, t2.c, t3.e
FROM t1 JOIN t2 ON t1.b=t2.c
JOIN t3 ON t3.e=t2.d

```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     24    0                    0   Start at 24
1     OpenRead       0     2     0     2              0   root=2 iDb=0; t1
2     OpenRead       1     3     0     2              0   root=3 iDb=0; t2
3     OpenRead       2     4     0     1              0   root=4 iDb=0; t3
4     Explain        4     0     0     SCAN t1        0
5     Rewind         0     23    0                    0
6       Explain        6     0     0     SCAN t2        0
7       Rewind         1     23    0                    0
8         Column         0     1     1                    0   r[1]= cursor 0 column 1
9         Column         1     0     2                    0   r[2]= cursor 1 column 0
10        Ne             2     21    1     BINARY-8       81  if r[1]!=r[2] goto 21
11        Explain        11    0     0     SCAN t3        0
12        Rewind         2     23    0                    0
13          Column         2     0     2                    0   r[2]= cursor 2 column 0
14          Column         1     1     1                    0   r[1]= cursor 1 column 1
15          Ne             1     20    2     BINARY-8       81  if r[2]!=r[1] goto 20
16          Column         0     0     3                    0   r[3]= cursor 0 column 0
17          Column         1     0     4                    0   r[4]= cursor 1 column 0
18          Column         2     0     5                    0   r[5]= cursor 2 column 0
19          ResultRow      3     3     0                    0   output=r[3..5]
20        Next           2     13    0                    1
21      Next           1     8     0                    1
22    Next           0     6     0                    1
23    Halt           0     0     0                    0
24    Transaction    0     0     3     0              1   usesStmtJournal=0
25    Goto           0     1     0                    0
```

Grayed content appears only when using special compile-time options intended for debugging and analysis.

Indentation showing loop structure is inserted by the display logic in the command-line shell and is not part of the actual bytecode

SELECT t1.a, t2.c, t3.e
FROM t1 JOIN t2 ON t1.b=t2.c
JOIN t3 ON t3.e=t2.d

```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     24    0                    0   Start at 24
1     OpenRead       0     2     0     2              0   root=2 iDb=0; t1
2     OpenRead       1     3     0     2              0   root=3 iDb=0; t2
3     OpenRead       2     4     0     1              0   root=4 iDb=0; t3
4     Explain        4     0     0     SCAN t1        0
5     Rewind         0     23    0                    0
6       Explain        6     0     0     SCAN t2        0
7       Rewind         1     23    0                    0
8         Column         0     1     1                    0   r[1]= cursor 0 column 1
9         Column         1     0     2                    0   r[2]= cursor 1 column 0
10        Ne             2     21    1     BINARY-8       81  if r[1]!=r[2] goto 21
11        Explain        11    0     0     SCAN t3        0
12        Rewind         2     23    0                    0
13          Column         2     0     2                    0   r[2]= cursor 2 column 0
14          Column         1     1     1                    0   r[1]= cursor 1 column 1
15          Ne             1     20    2     BINARY-8       81  if r[2]!=r[1] goto 20
16          Column         0     0     3                    0   r[3]= cursor 0 column 0
17          Column         1     0     4                    0   r[4]= cursor 1 column 0
18          Column         2     0     5                    0   r[5]= cursor 2 column 0
19          ResultRow      3     3     0                    0   output=r[3..5]
20        Next           2     13    0                    1
21      Next           1     8     0                    1
22    Next           0     6     0                    1
23    Halt           0     0     0                    0
24    Transaction    0     0     3     0              1   usesStmtJournal=0
25    Goto           0     1     0                    0
```

Open cursors

t1.b=t2.c

t3.e=t2.d

Generate one result row

SELECT t1.a, t2.c, t3.e
FROM t1 JOIN t2 ON t1.b=t2.c
LEFT JOIN t3 ON t3.e=t2.d

```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     29    0                    0   Start at 29
1     OpenRead       0     2     0     2              0   root=2 iDb=0; t1
2     OpenRead       1     3     0     2              0   root=3 iDb=0; t2
3     OpenRead       2     4     0     1              0   root=4 iDb=0; t3
4     Explain        4     0     0     SCAN t1        0
5     Rewind         0     28    0                    0
6       Explain        6     0     0     SCAN t2        0
7       Rewind         1     28    0                    0
8         Column         0     1     1                    0   r[1]= cursor 0 column 1
9         Column         1     0     2                    0   r[2]= cursor 1 column 0
10        Ne             2     26    1     BINARY-8       81  if r[1]!=r[2] goto 26
11        Explain        11    0     0     SCAN t3 LEFT-JOIN 0
12        Integer        0     3     0                    0   r[3]=0; init LEFT JOIN match flag
13        Rewind         2     23    0                    0
14          Column         2     0     2                    0   r[2]= cursor 2 column 0
15          Column         1     1     1                    0   r[1]= cursor 1 column 1
16          Ne             1     22    2     BINARY-8       81  if r[2]!=r[1] goto 22
17          Integer        1     3     0                    0   r[3]=1; record LEFT JOIN hit
18          Column         0     0     4                    0   r[4]= cursor 0 column 0
19          Column         1     0     5                    0   r[5]= cursor 1 column 0
20          Column         2     0     6                    0   r[6]= cursor 2 column 0
21          ResultRow      4     3     0                    0   output=r[4..6]
22        Next           2     14    0                    1
23        IfPos          3     26    0                    0   if r[3]>0 then r[3]-=0, goto 26
24        NullRow        2     0     0                    0
25        Goto           0     17    0                    0
26      Next           1     8     0                    1
27    Next           0     6     0                    1
28    Halt           0     0     0                    0
29    Transaction    0     0     3     0              1   usesStmtJournal=0
30    Goto           0     1     0                    0
```

Extra code to implement the LEFT JOIN

CREATE INDEX t2c ON t2(c);
SELECT * FROM t1 JOIN t2 ON t1.b=t2.c;

```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     20    0                    0   Start at 20
1     OpenRead       0     2     0     2              0   root=2 iDb=0; t1
2     OpenRead       1     3     0     2              0   root=3 iDb=0; t2
3     OpenRead       2     5     0     k(2,,)         2   root=5 iDb=0; t2c
4     Explain        4     0     0     SCAN t1        0
5     Rewind         0     19    0                    0
6       Explain        6     0     0     SEARCH t2 USING INDEX t2c (c=?) 0
7       Column         0     1     1                    0   r[1]= cursor 0 column 1
8       IsNull         1     18    0                    0   if r[1]==NULL goto 18
9       SeekGE         2     18    1     1              0   key=r[1]
10        IdxGT          2     18    1     1              0   key=r[1]
11        DeferredSeek   2     0     1                    0   Move 1 to 2.rowid if needed
12        Column         0     0     2                    0   r[2]= cursor 0 column 0
13        Column         0     1     3                    0   r[3]= cursor 0 column 1
14        Column         2     0     4                    0   r[4]= cursor 2 column 0
15        Column         1     1     5                    0   r[5]= cursor 1 column 1
16        ResultRow      2     4     0                    0   output=r[2..5]
17      Next           2     10    1                    0
18    Next           0     6     0                    1
19    Halt           0     0     0                    0
20    Transaction    0     0     4     0              1   usesStmtJournal=0
21    Goto           0     1     0                    0
```

Open cursors

Search index t2c for entries with t2.c=t1.b

Generate one result row

Try these on Fiddle:

- Range scans
- Constraints involving the IN operator
- OR-connected constraints with multiple indexes
- FULL OUTER JOINs

A query plan is just ...

- Nested loops over cursors
- One loop for each term in the FROM clause

The interesting questions are ...

- What is the best nesting order for the loops?
- Which indexes (if any) should be used for each loop?

WhereLoop object

- Which FROM term this loop implements
- **Prerequisites**: Which other FROM terms must occur in outer loops
- **Type of loop**: full-table scan, primary key, secondary index, query-time index, ...
- **Cost**
  - Setup cost (for query-time index)
  - Cost per iteration
  - Output rows per iteration

LogEst(x) = 10*log2(x)

Actual Value	LogEst Equivalent
1	0
2	10
5	23
100	66
10,000	132
0.9375	-1
0.1	-33

Query Planning Example

SELECT * FROM A JOIN B ON A.a2=B.b1; /* Index on B.b1 */

Id	Term	Prereq	Type	Cost
0	A	-	full-table scan	0, 216, 200
1	A	B	query-time index on A.a2	271, 53, 43
2	B	-	full-table scan	0, 216, 200
3	B	A	index on B.b1	0, 62, 33

Best query plan is an ordered list of entries such that:

- Exactly one entry for each FROM term
- All prerequisites satisfied
- Lowest cost

In this case: A(0), B(3)

Query Planning Example

SELECT * FROM A JOIN B ON A.a2=B.b1; /* Index on B.b1 */

Id	Term	Prereq	Type	Cost
0	A	-	full-table scan	0, 216, 200
1	A	B	query-time index on A.a2	271, 53, 43
2	B	-	full-table scan	0, 216, 200
3	B	A	index on B.b1	0, 62, 33

```
Computing the table

  for each FROM term:
     Add full-table scan entry
     Add query-time index entries if suitable terms exist in WHERE clause
     Add primary-key entry if suitable terms exist in WHERE clause
     for each index on the table:
        Add secondary-index entry if suitable terms exist in WHERE clause
  Purge unusable entries from the table
```

Query Planning Example #2

```
CREATE TABLE A(a1 INTEGER PRIMARY KEY, a2 INT);
CREATE TABLE B(b1 INT, b2 INT, b3 INT);          CREATE INDEX Bx1 ON B(b1);
CREATE TABLE C(c1 INT, c2 INT);                  CREATE INDEX Cx1 ON C(c1);

SELECT * FROM A JOIN B ON b1=a2 JOIN C ON c1=min(a2,b2) WHERE a1=c2;
```

Id	Term	Prereq	Type	Cost
0	A	B	query-time index	271, 53, 43
1	A	-	full-table scan	0, 216, 200
2	A	C	primary-key	0, 45, 0
3	B	A	index Bx1	0, 62, 33
4	B	-	full-table scan	0, 216, 200
5	C	A	query-time index	271, 53, 200
6	C	-	full-table scan	0, 216, 200
7	C	A, B	index Cx1	0, 62, 32

In this case, the best plan is: C(6), A(2), B(3)

Query Planning Example #3

```
SELECT A.a1, B.a2, B.a3, D.a4
  FROM A JOIN B ON A.id=B.a_id
         JOIN D ON D.b_id=B.id
         JOIN C ON C.id=D.c_id
WHERE  A.a1=42
       AND B.a3=12
       AND D.a2=25;
```

```
CREATE TABLE A(id INTEGER PRIMARY KEY, a1 INT);
CREATE TABLE B(id INTEGER PRIMARY KEY, a2 INT, a3 INT, a_id INT);
CREATE TABLE C(id INTEGER PRIMARY KEY);
CREATE TABLE D(id INTEGER PRIMARY KEY, a2 INT, a4 INT, b_id INT, c_id INT);
CREATE INDEX A1 ON A(a1);
CREATE INDEX Ba ON B(a_id); CREATE INDEX B3 ON B(a3);
CREATE INDEX Dc ON D(c_id); CREATE INDEX Db ON D(b_id);
CREATE INDEX D2 ON D(a2);
```

	Id	Term	Prereq	Type	Cost
A.a1=42 →	0	A	-	index on A.a1	0, 56, 33
A.id=B.a_id →	1	A	B	primary-key	0, 45, -1
B.a3=12 →	2	B	-	index on B.a3	0, 62, 33
B.id=D.b_id →	3	B	D	primary-key	0, 45, -1
B.a_id=A.id →	4	B	A	index on B.a_id	0, 62, 32
D.a2=25 →	5	D	-	index on D.a2	0, 62, 33
D.b_id=B.id →	6	D	B	index on D.b_id	0, 62, 32
D.c_id=C.id →	7	D	C	index on D.c_id	0, 62, 32
	8	C	-	full-table scan	0, 216, 200
C.id=D.c_id →	9	C	D	primary-key	0, 45, 0

In this case, the best plan is: D(5), B(3), A(1), C(9)

Query Planning Example #3

```
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     31    0                    0   Start at 31
1     OpenRead       2     5     0     5              0   root=5 iDb=0; D
2     OpenRead       4     11    0     k(2,,)         2   root=11 iDb=0; D2
3     OpenRead       1     3     0     4              0   root=3 iDb=0; B
4     OpenRead       0     2     0     2              0   root=2 iDb=0; A
5     OpenRead       3     4     0     0              0   root=4 iDb=0; C
6     Explain        6     0     0     SEARCH D USING INDEX D2 (a2=?) 0
7     Integer        25    1     0                    0   r[1]=25
8     SeekGE         4     30    1     1              0   key=r[1]
9       IdxGT          4     30    1     1              0   key=r[1]
10      DeferredSeek   4     0     2                    0   Move 2 to 4.rowid if needed
11      Explain        11    0     0     SEARCH B USING INTEGER PRIMARY KEY (rowid=?) 0
12      Column         2     3     2                    0   r[2]= cursor 2 column 3
13      SeekRowid      1     29    2                    0   intkey=r[2]
14      Column         1     2     3                    0   r[3]= cursor 1 column 2
15      Ne             4     29    3     BINARY-8       81  if r[3]!=r[4] goto 29
16      Explain        16    0     0     SEARCH A USING INTEGER PRIMARY KEY (rowid=?) 0
17      Column         1     3     5                    0   r[5]= cursor 1 column 3
18      SeekRowid      0     29    5                    0   intkey=r[5]
19      Column         0     1     3                    0   r[3]= cursor 0 column 1
20      Ne             6     29    3     BINARY-8       81  if r[3]!=r[6] goto 29
21      Explain        21    0     0     SEARCH C USING INTEGER PRIMARY KEY (rowid=?) 0
22      Column         2     4     7                    0   r[7]= cursor 2 column 4
23      SeekRowid      3     29    7                    0   intkey=r[7]
24      Column         0     1     8                    0   r[8]= cursor 0 column 1
25      Column         1     1     9                    0   r[9]= cursor 1 column 1
26      Column         1     2     10                   0   r[10]= cursor 1 column 2
27      Column         2     2     11                   0   r[11]= cursor 2 column 2
28      ResultRow      8     4     0                    0   output=r[8..11]
29    Next           4     9     1                    0
30    Halt           0     0     0                    0
31    Transaction    0     0     10    0              1   usesStmtJournal=0
32    Integer        12    4     0                    0   r[4]=12
33    Integer        42    6     0                    0   r[6]=42
34    Goto           0     1     0                    0
```

Open cursors

Loop over all D.a2=25

Find B where B.id=D.b_id

Skip if B.a3<>12

Find A where A.id=B.a_id

Skip if A.a1<>42

Find C where C.id=D.c_id

Generate one result row

Query Planning Example #3

```
SELECT A.a1, B.a2, B.a3, D.a4
  FROM A JOIN B ON A.id=B.a_id
         JOIN D ON D.b_id=B.id
         JOIN C ON C.id=D.c_id
WHERE  A.a1=42
       AND B.a3=12
       AND D.a2=25;
```

Id	Term	Prereq	Type	Cost
X	A	-	query-time index on A.a1	271,53,43
X	A	-	full-table scan	0,216,180
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
X	B	-	query-time index on B.a3	271,53,43
X	B	A	query-time index on B.a_id	271,53,43
X	B	-	full-table scan	0,216,180
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
X	D	-	query-time index on D.a2	271,53,43
X	D	B	query-time index on D.b_id	271,53,43
X	D	C	query-time index on D.c_id	271,53,43
X	D	-	full-table scan	0, 216, 200
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Query Planning Example #3

```
SELECT A.a1, B.a2, B.a3, D.a4
  FROM A JOIN B ON A.id=B.a_id
              JOIN D ON D.b_id=B.id
              JOIN C ON C.id=D.c_id
WHERE  A.a1=42
            AND B.a3=12
            AND D.a2=25;
```

Id	Term	Prereq	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

- *All possible outer loops*
- *WhereLoops without prerequisites*

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 2:

Path	Cost
A(0), B(2)	96, 66
A(0), B(4)	96, 65
A(0), D(5)	96, 66
A(0), C(8)	249, 233
...	...

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), B(2)	96, 66
A(0), B(4)	96, 65
A(0), D(5)	96, 66
A(0), C(8)	249, 233
...	...

X

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), B(4)	96, 65
A(0), D(5)	96, 66
A(0), C(8)	249, 233
...	...

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), B(4)	96, 65
A(0), D(5)	96, 66
A(0), C(8)	249, 233
B(2), A(0)	91, 66
B(2), A(1)	82, 32
B(2), D(5)	96, 66
B(2), D(6)	96, 65
B(2), C(8)	249, 233
...	...

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost	
A(0), B(4)	96, 65	X
A(0), D(5)	96, 66	
A(0), C(8)	249, 233	
B(2), A(0)	91, 66	X
B(2), A(1)	82, 32	
B(2), D(5)	96, 66	X
B(2), D(6)	96, 65	
B(2), C(8)	249, 233	
...	...	

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), D(5)	96, 66
A(0), C(8)	249, 233
B(2), A(1)	82, 32
B(2), D(6)	96, 65
B(2), C(8)	249, 233
...	...

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), D(5)	96, 66
A(0), C(8)	249, 233
B(2), A(1)	82, 32
B(2), D(6)	96, 65
B(2), C(8)	249, 233
D(5), A(0)	91, 66
D(5), B(2)	96, 66
D(5), B(3)	82, 32
D(5), C(8)	249, 233
D(5), C(9)	144, 66
...	...

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost	
A(0), D(5)	96, 66	X
A(0), C(8)	249, 233	
B(2), A(1)	82, 32	
B(2), D(6)	96, 65	X
B(2), C(8)	249, 233	
D(5), A(0)	91, 66	
D(5), B(2)	96, 66	X
D(5), B(3)	82, 32	
D(5), C(8)	249, 233	X
D(5), C(9)	144, 66	
...	...	

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), C(8)	249, 233
B(2), A(1)	82, 32
B(2), C(8)	249, 233
D(5), A(0)	91, 66
D(5), B(3)	82, 32
D(5), C(9)	144, 66
...	...

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost	
A(0), C(8)	249, 233	
B(2), A(1)	82, 32	
B(2), C(8)	249, 233	
D(5), A(0)	91, 66	
D(5), B(3)	82, 32	
D(5), C(9)	144, 66	
C(8), A(0)	257, 233	X
C(8), B(2)	263, 233	X
C(8), D(5)	263, 233	X
C(8), D(7)	263, 232	X

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), C(8)	249, 233
B(2), A(1)	82, 32
B(2), C(8)	249, 233
D(5), A(0)	91, 66
D(5), B(3)	82, 32
D(5), C(9)	144, 66

Now we know all paths of length 2!

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), C(8)	249, 233
B(2), A(1)	82, 32
B(2), C(8)	249, 233
D(5), A(0)	91, 66
D(5), B(3)	82, 32
D(5), C(9)	144, 66

Paths of length 3: *Same procedure for length 3...*

Path	Cost
B(2), A(1), C(8)	248, 232
D(5), B(3), A(1)	89, 31
D(5), B(3), C(9)	89, 32
D(5), C(9), A(0)	95, 66

Query Planning Example #3

Id	T	Pre	Type	Cost
0	A	-	index on A.a1	0, 56, 33
1	A	B	primary-key	0, 45, -1
2	B	-	index on B.a3	0, 62, 33
3	B	D	primary-key	0, 45, -1
4	B	A	index on B.a_id	0, 62, 32
5	D	-	index on D.a2	0, 62, 33
6	D	B	index on D.b_id	0, 62, 33
7	D	C	index on D.c_id	0, 62, 33
8	C	-	full-table scan	0, 216, 200
9	C	D	primary-key	0, 45, 0

Paths of length 1:

Path	Cost
A(0)	56,33
B(2)	62,33
D(5)	62,33
C(8)	216,200

Paths of length 2:

Path	Cost
A(0), C(8)	249, 233

Path length 4:

Path	Cost
D(5), B(3), A(1), C(9)	93, 31

And length 4...

Paths of length 3:

Path	Cost
B(2), A(1), C(8)	248, 232
D(5), B(3), A(1)	89, 31
D(5), B(3), C(9)	89, 32
D(5), C(9), A(0)	95, 66

Max 10 Paths/Step Because: TPC-H

Id	Term	Prereq	Type	Cost
0	P	-	index on type	0, 94, 70
1	P	L	primary-key	0, 42, -1
2	S	L	primary-key	0, 38, 0
3	S	N2	index on nationkey	0, 76, 53
4	S	-	full-table scan	0, 115, 99
5	L	O	index on orderkey, line	0, 54, 23
6	L	S	index on suppkey	0, 114, 92
7	L	P	index on partkey	0, 73, 49
8	L	-	full-table scan	0, 208, 192
9	O	C	index on custkey	0, 64, 38
10	O	L	primary-key	0, 43, -1

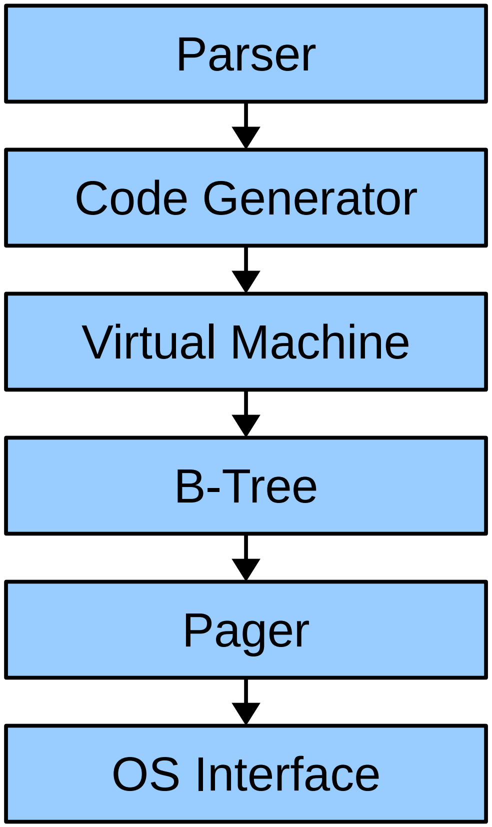
Id	Term	Prereq	Type	Cost
11	O	-	index on orderdate	0, 134, 112
12	C	N1	index on nationkey	0, 114, 92
13	C	O	primary-key	0, 42, 0
14	C	-	full-table scan	0, 154, 138
15	N1	R	index on regionkey	0, 48, 23
16	N1	C	primary-key	0, 32, 0
17	N1	-	full-table scan	0, 62, 46
18	N2	S	primary-key	0, 32, 0
19	N2	-	full-table scan	0, 62, 46
20	R	-	full-table scan	0, 39, 3
21	R	N1	primary-key	0, 28, -1

Best query plan: P(0), L(7), S(2), O(10), C(13), N1(17), R(21), N2(18) Cost: 176

max=9 plan: R(20), P(0), L(7), O(10), C(13), N1(17), S(2), N2(18) Cost: 179

Summary: How SQLite Works

- In-process library. No server.
- One-file database
- Power-safe transactions
- Row-store
- Everything's a b-tree
- Schema stored as a table of CREATE statements
- Bytecode
- Queries planned by solving TSP



Website: https://sqlite.org/

Sources: https://sqlite.org/src/timeline

Forum: https://sqlite.org/forum