SQLite: Past, Present, and Future

Aalborg University

November 18, 2022

Kevin P. Gaffney
Martin Prammer
Jignesh M. Patel

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

D. Richard Hipp
Larry Brasfield
Dan Kennedy

SQLite

About me

Recent trip to Australia

Bachelor's degrees in Computer Science and Biochemistry at University of Oklahoma.

Currently pursuing PhD in Computer Sciences at the University of Wisconsin Madison.

Interested in all aspects of database system performance.

If you have any questions about database research, life as a grad student, etc., please reach out! My email is kpgaffney@wisc.edu.

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

The most widely deployed database engine in the world. Why?

Cross-platform

Database file can be copied between hardware architectures. Recommended storage format by US Library of Congress.

Compact and self-contained

Source code is contained in a single C file and compiles to < 1 MB library. No external dependencies. Runs in the host process.

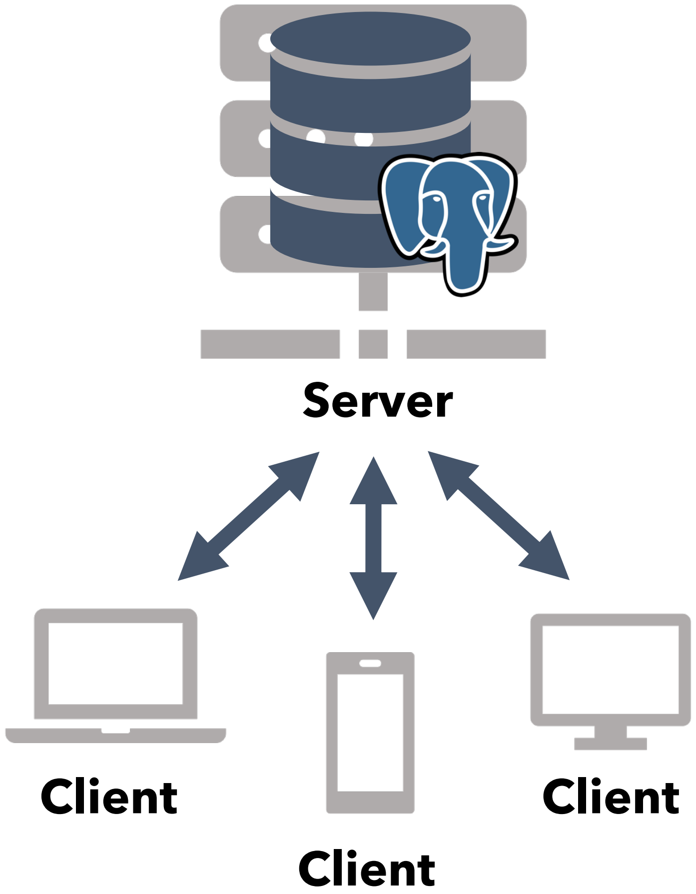
Reliable

100% machine branch test coverage. Over 600 lines of test code for every line of library code.

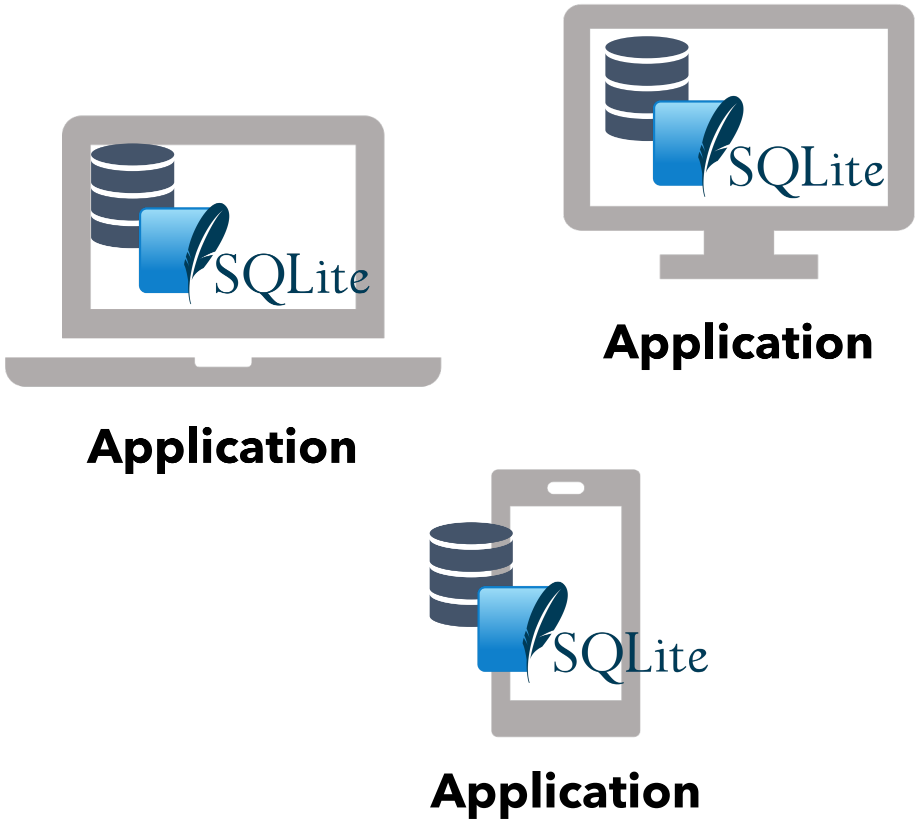
Fast

10s of thousands of transactions per second. Can be faster and more space-efficient than the filesystem.

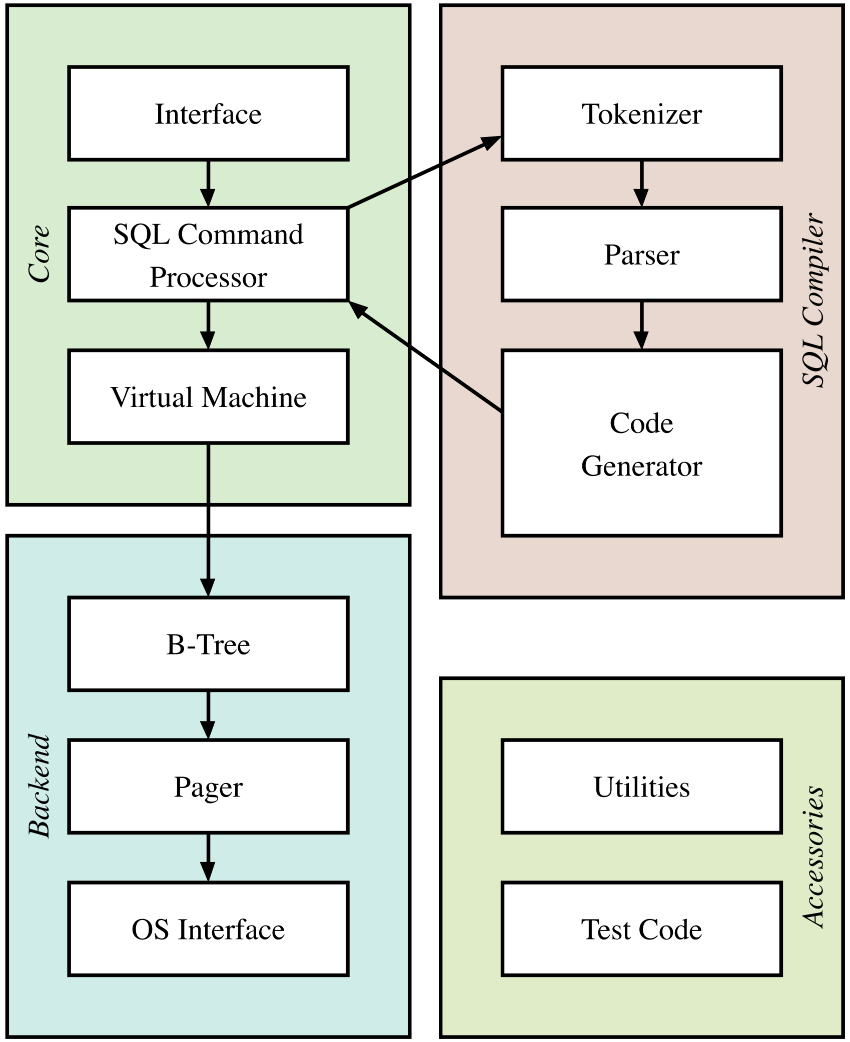
Client-server DBMS



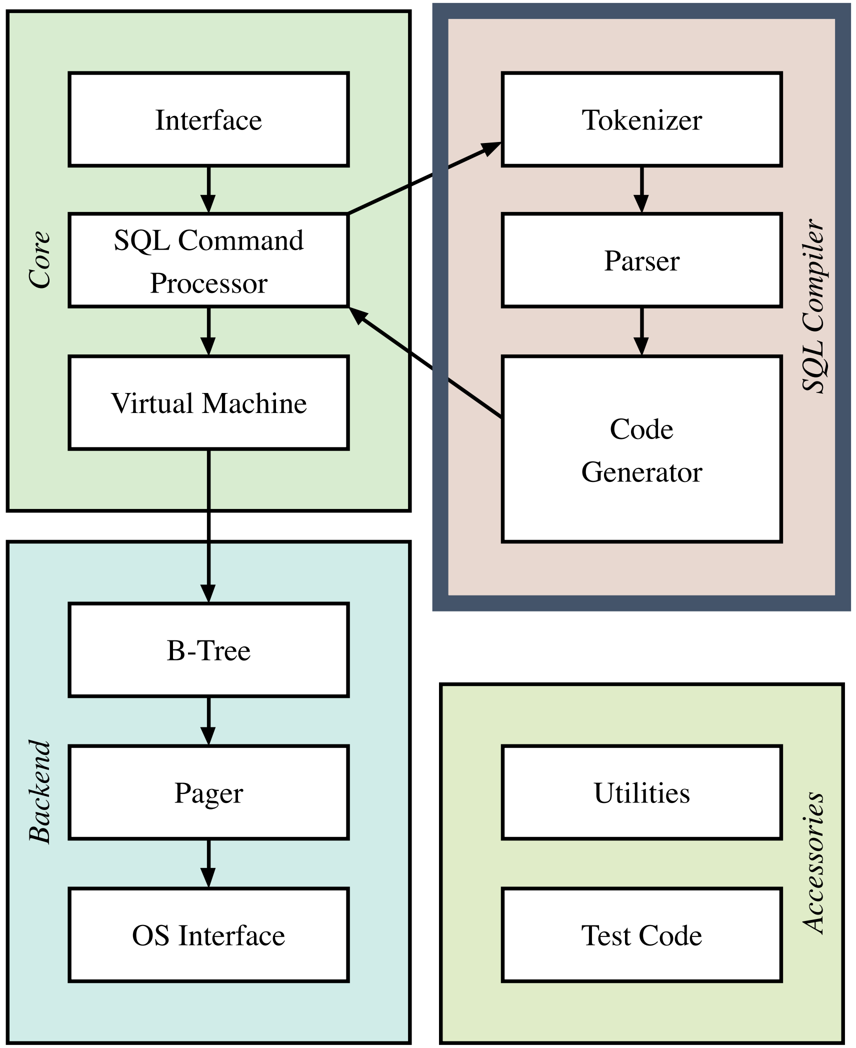
In-process DBMS



Architecture



SQL compiler modules



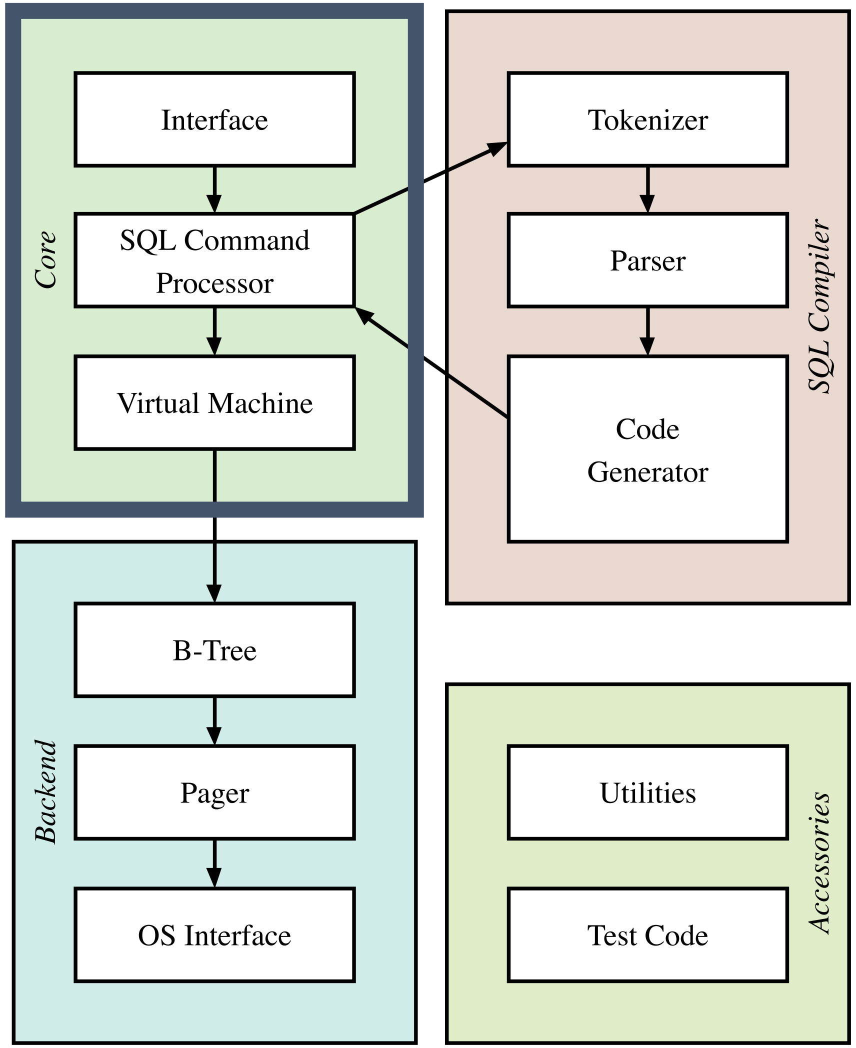
SQL compiler modules

```sql
SELECT SUM(lo_extendedprice * lo_discount)
FROM lineorder, dates
WHERE lo_orderdate = d_datekey
  AND d_year = 1993
  AND lo_discount BETWEEN 1 AND 3
  AND lo_quantity < 25;
```

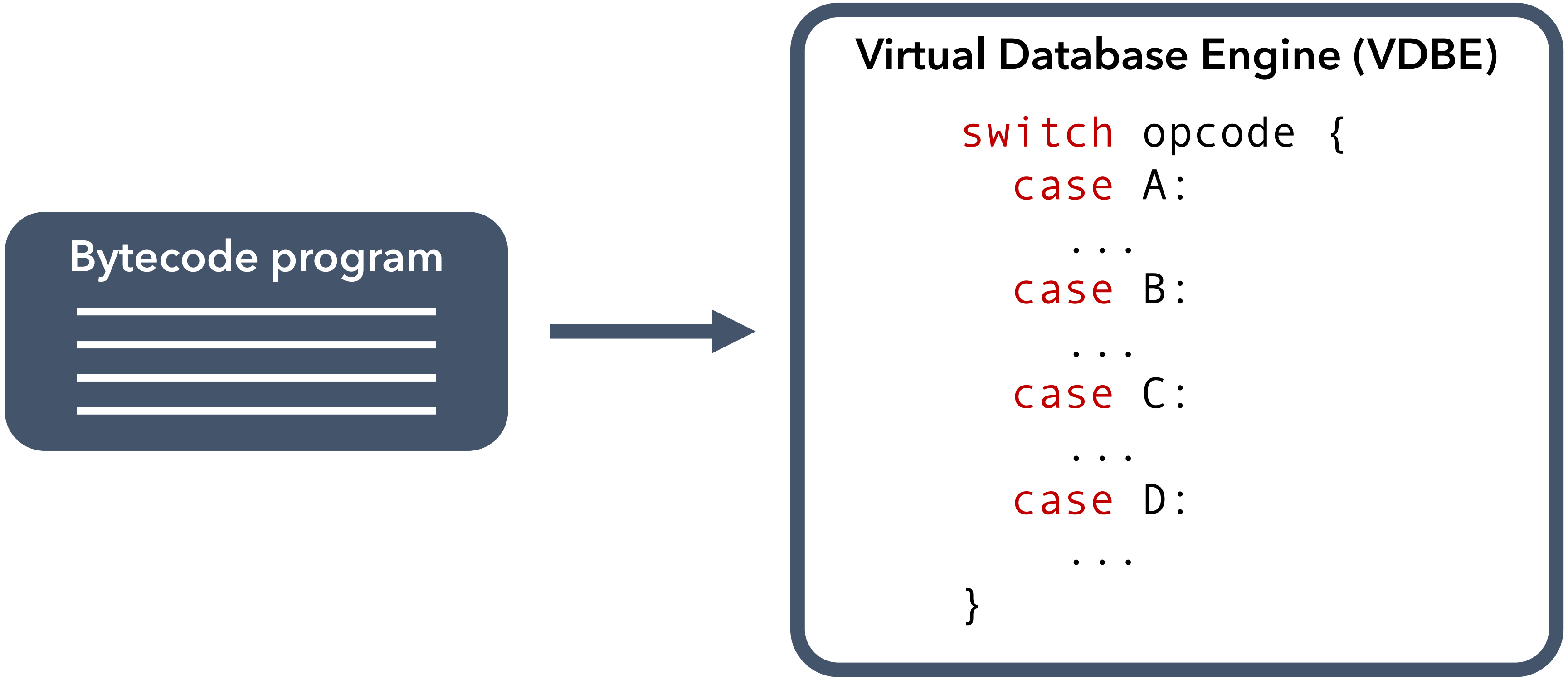
Address	Opcode	P1	P2	P3	P4	P5
0	Init	1	23	0		00
1	Null	0	1	3		00
2	OpenRead	0	7	0	12	00
3	OpenRead	1	6	0	5	00
4	Rewind	0	19	0		00
5	Column	0	11	4		00
6	Lt	6	18	4	BINARY-8	54
7	Gt	7	18	4	BINARY-8	54
8	Column	0	8	4		00
9	Ge	8	18	4	BINARY-8	54
10	Column	0	5	9		00
11	SeekRowid	1	18	9		00
12	Column	1	4	4		00
13	Ne	10	18	4	BINARY-8	54
14	Column	0	9	5		00
15	Column	0	11	11		00
16	Multiply	11	5	4		00
17	AggStep1	0	4	1	sum(1)	01
18	Next	0	5	0		01

⋮

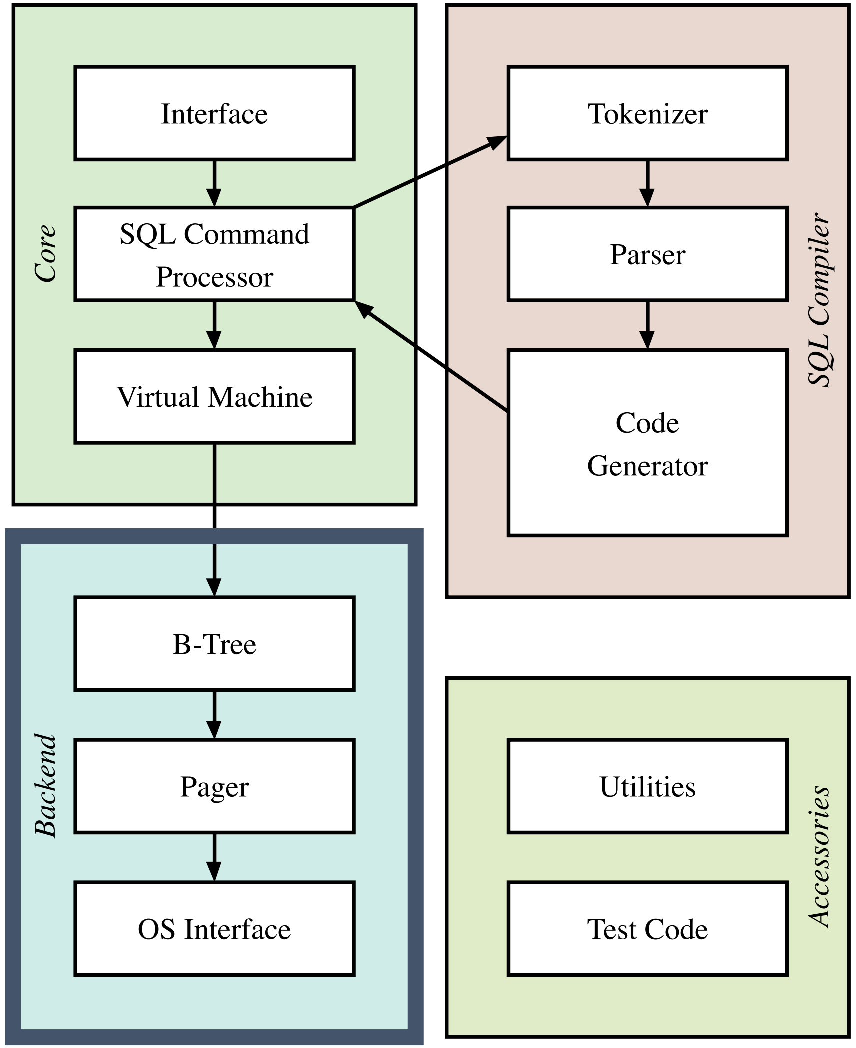
Core modules



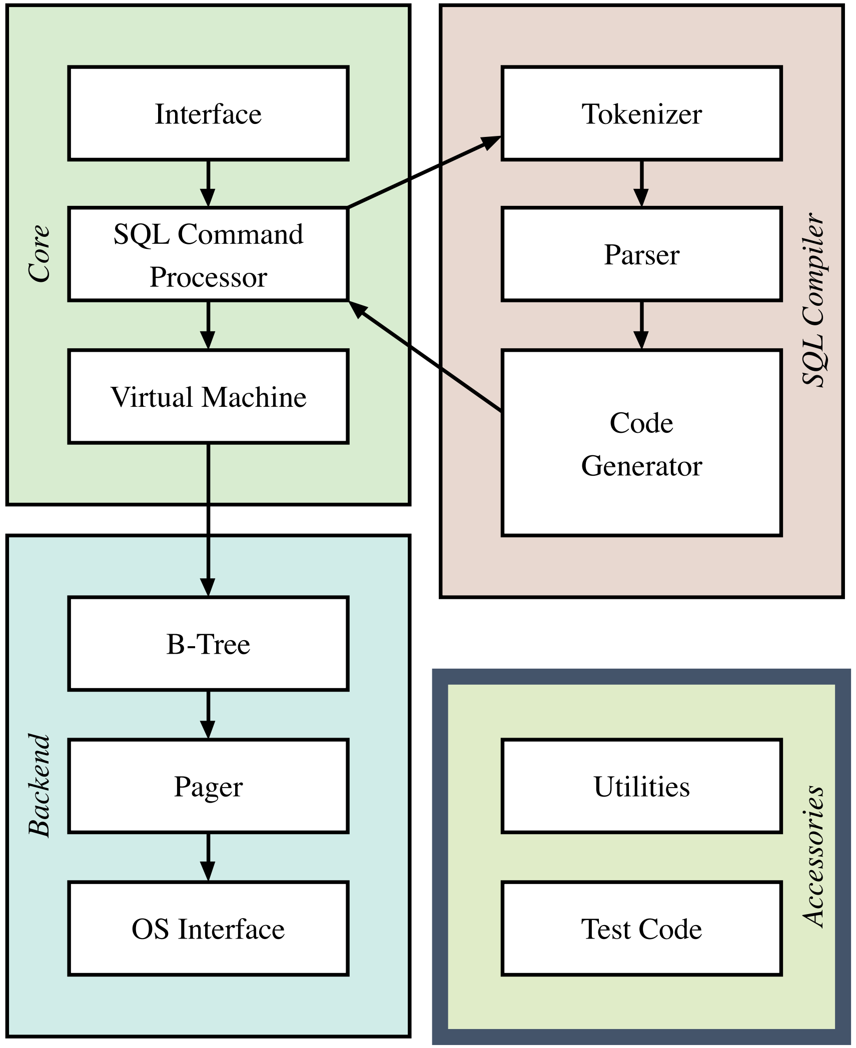
Core modules



Backend modules



Accessory modules



Transactions

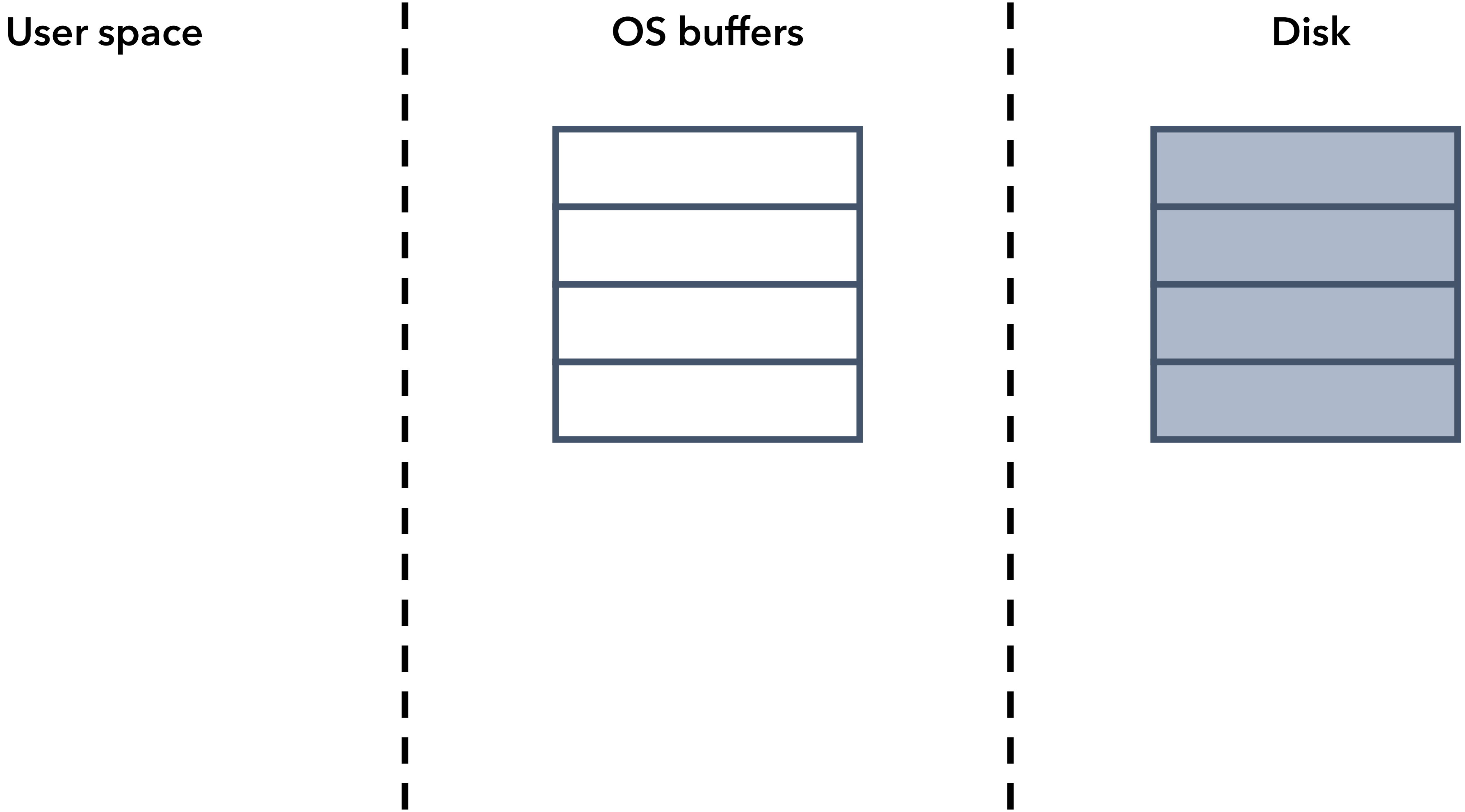
How does SQLite achieve ACID guarantees?

Two transaction modes:

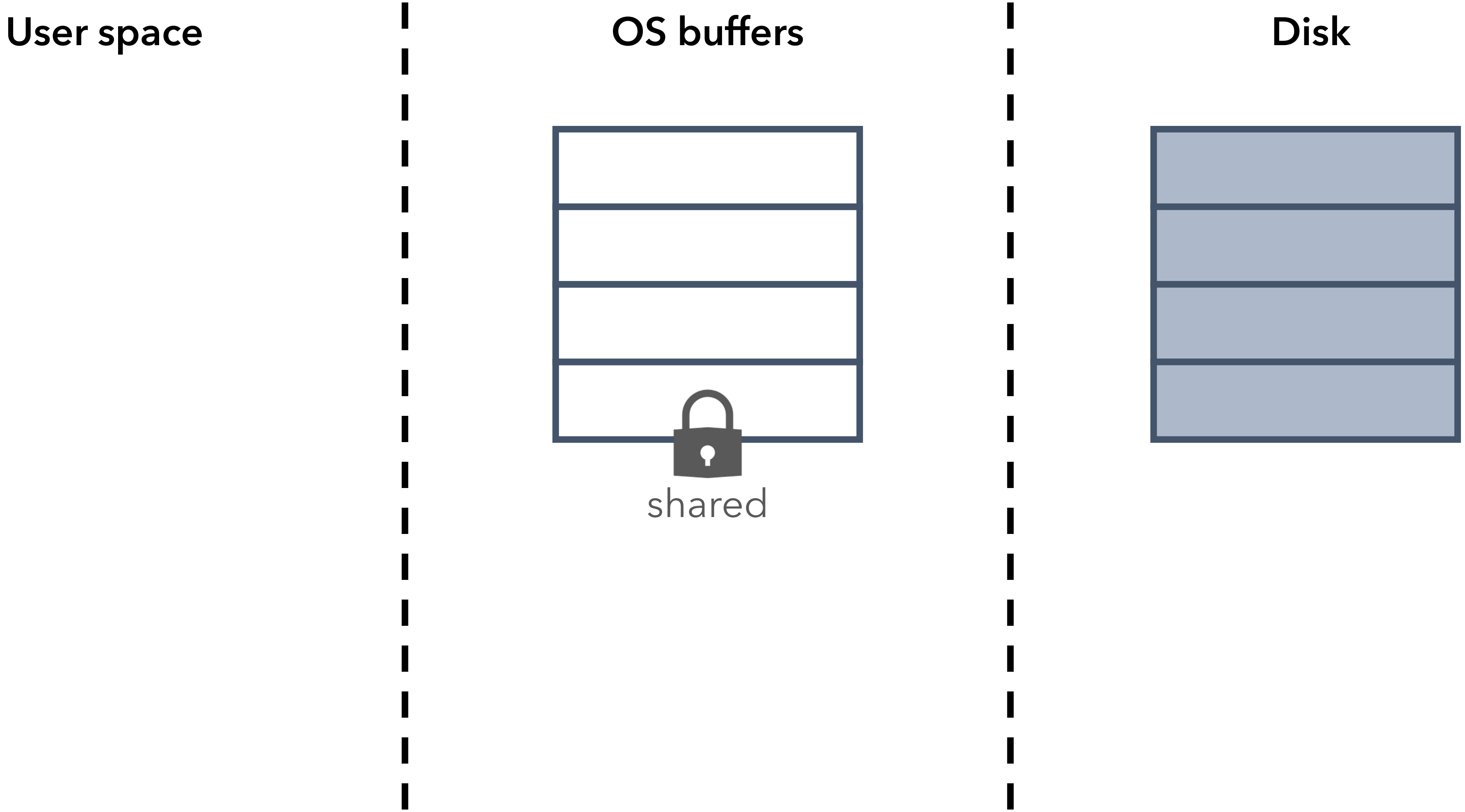
Rollback mode

WAL mode

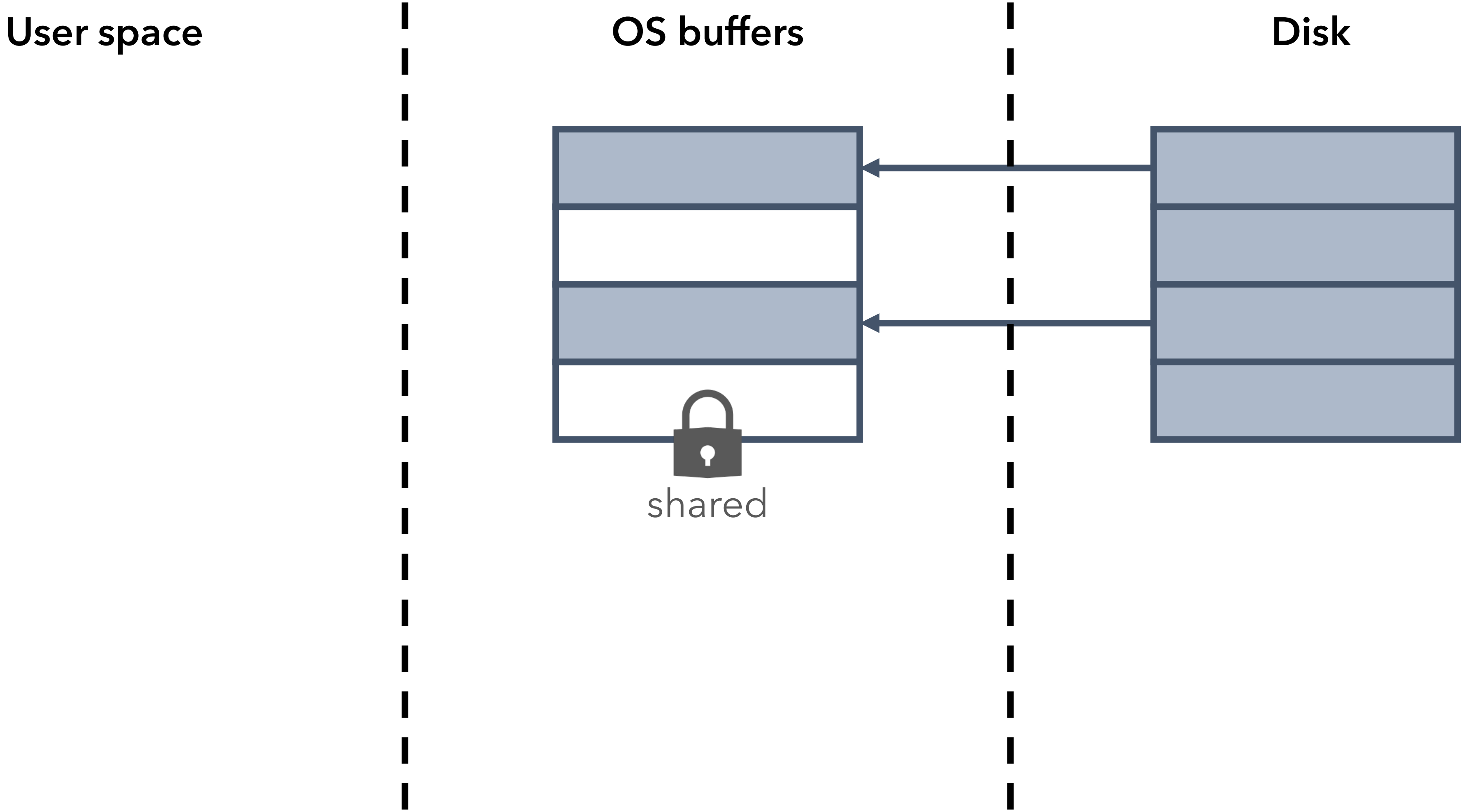
Rollback mode



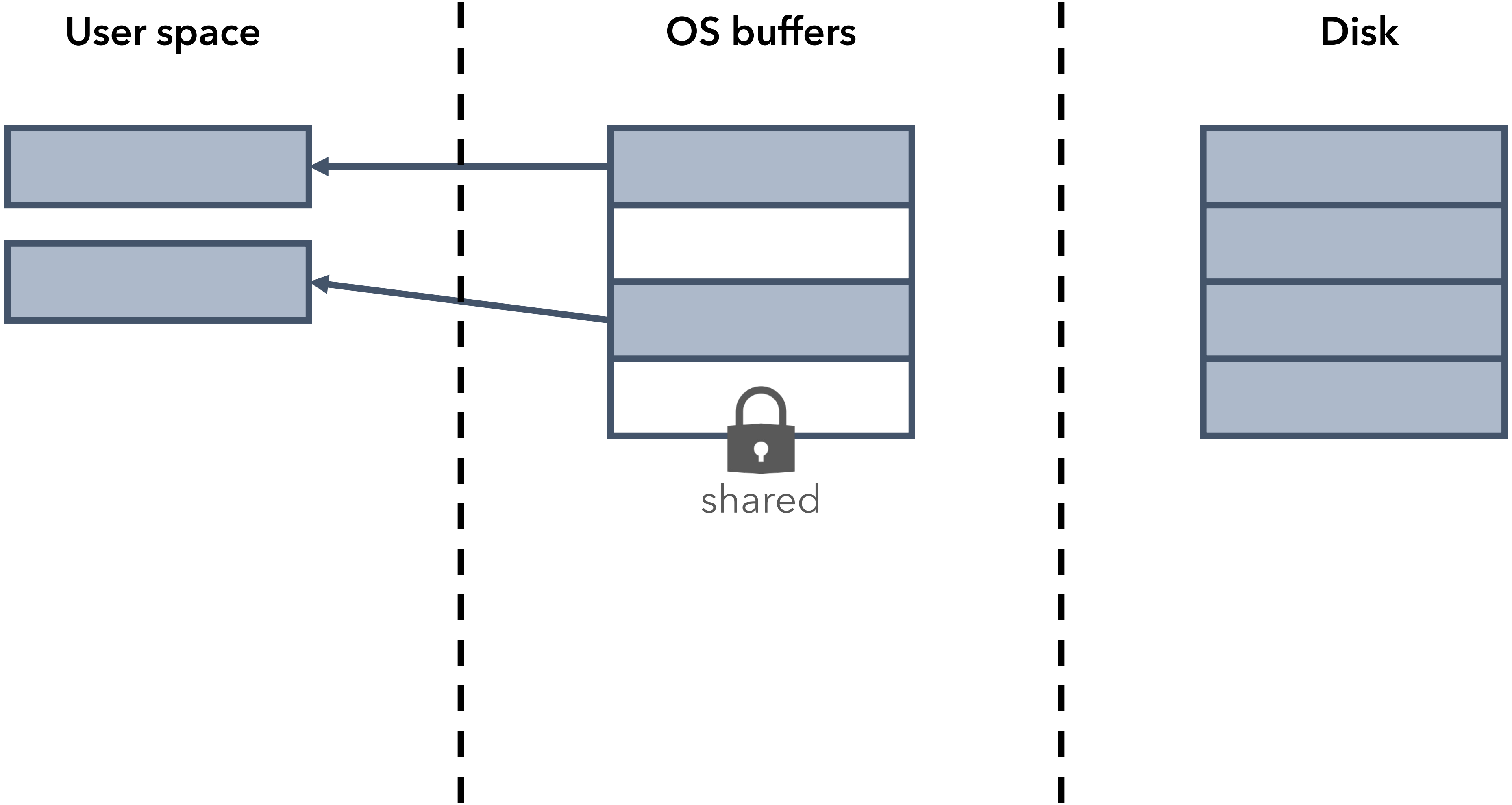
Rollback mode



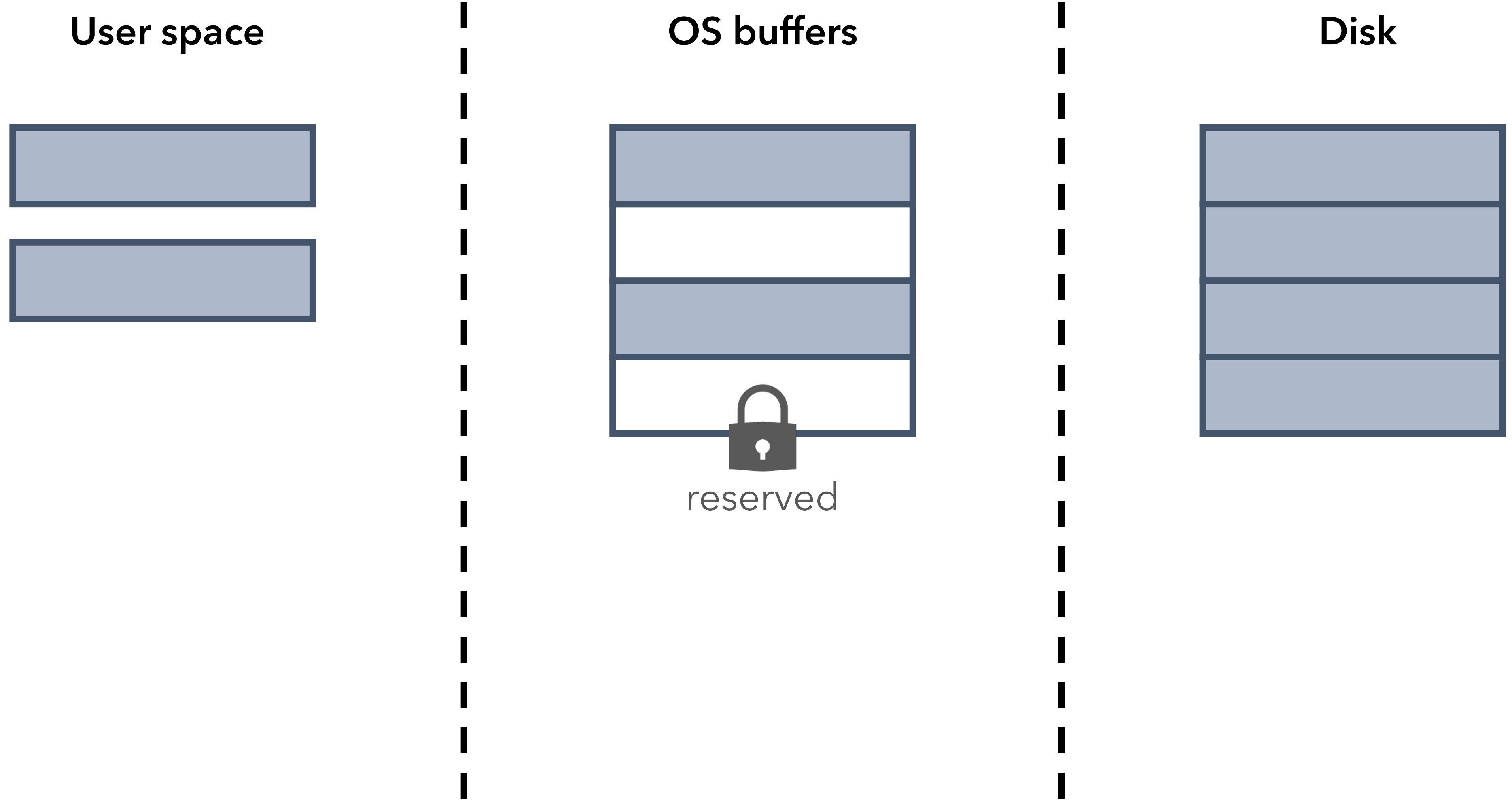
Rollback mode



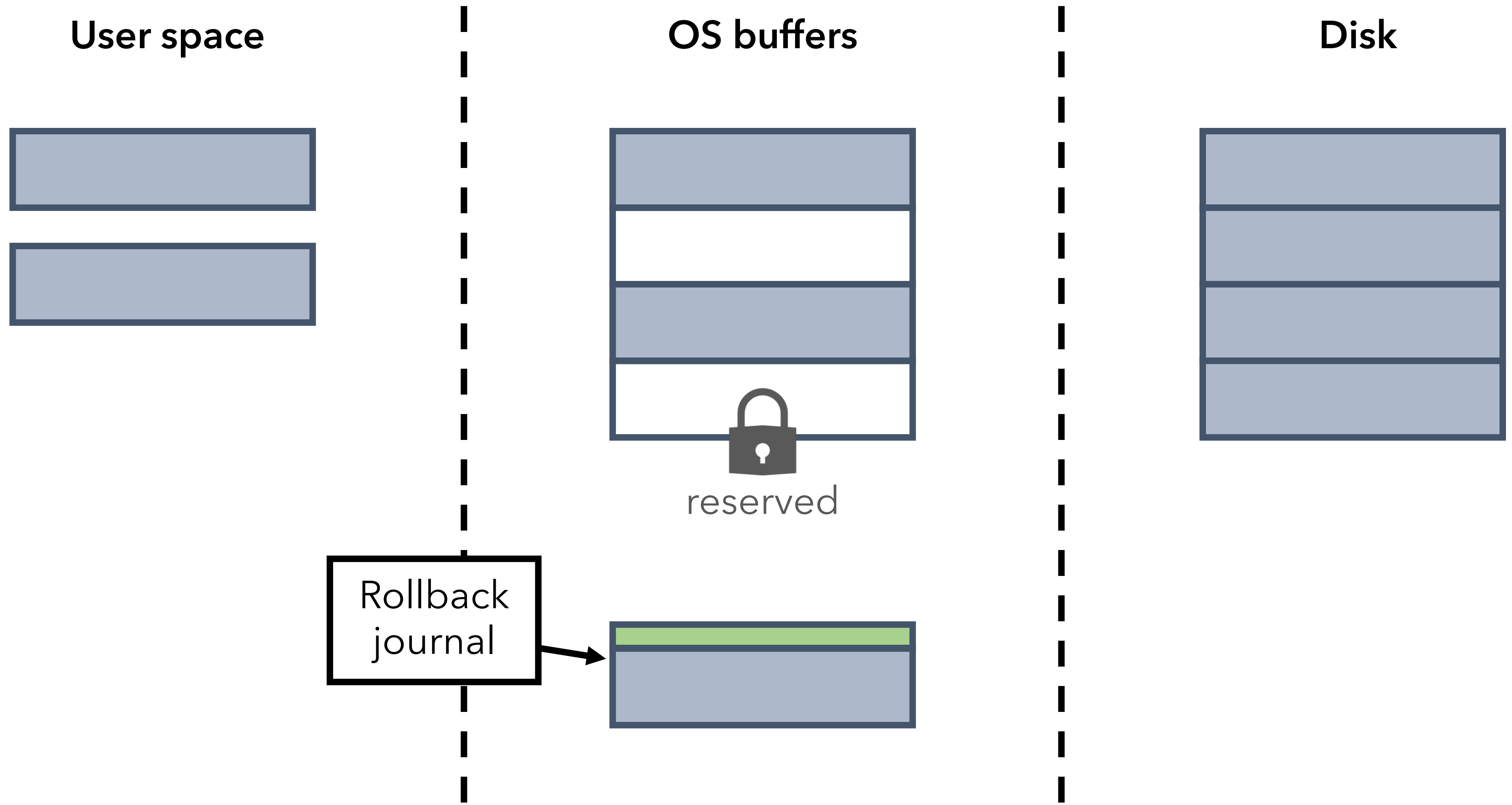
Rollback mode



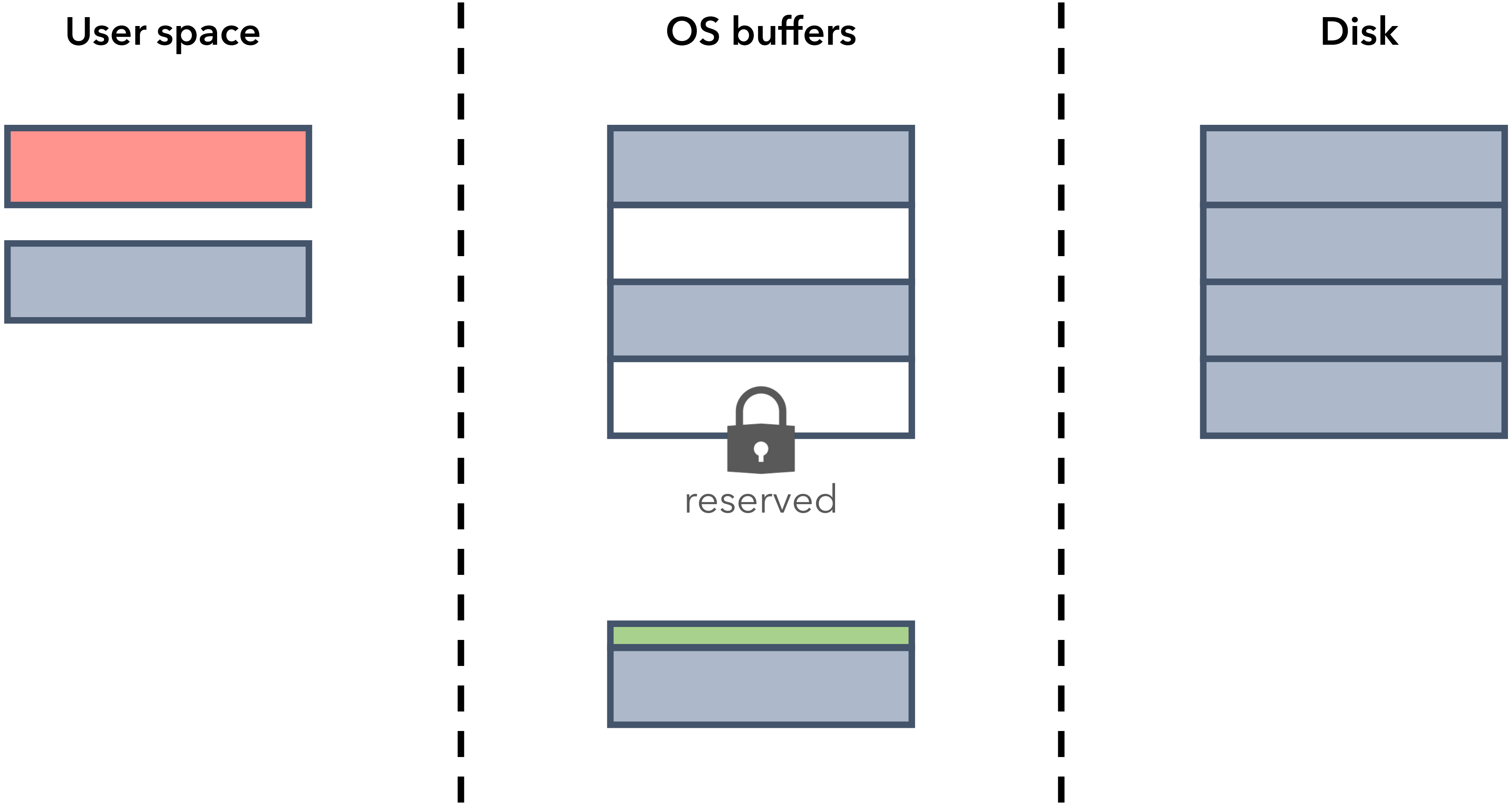
Rollback mode



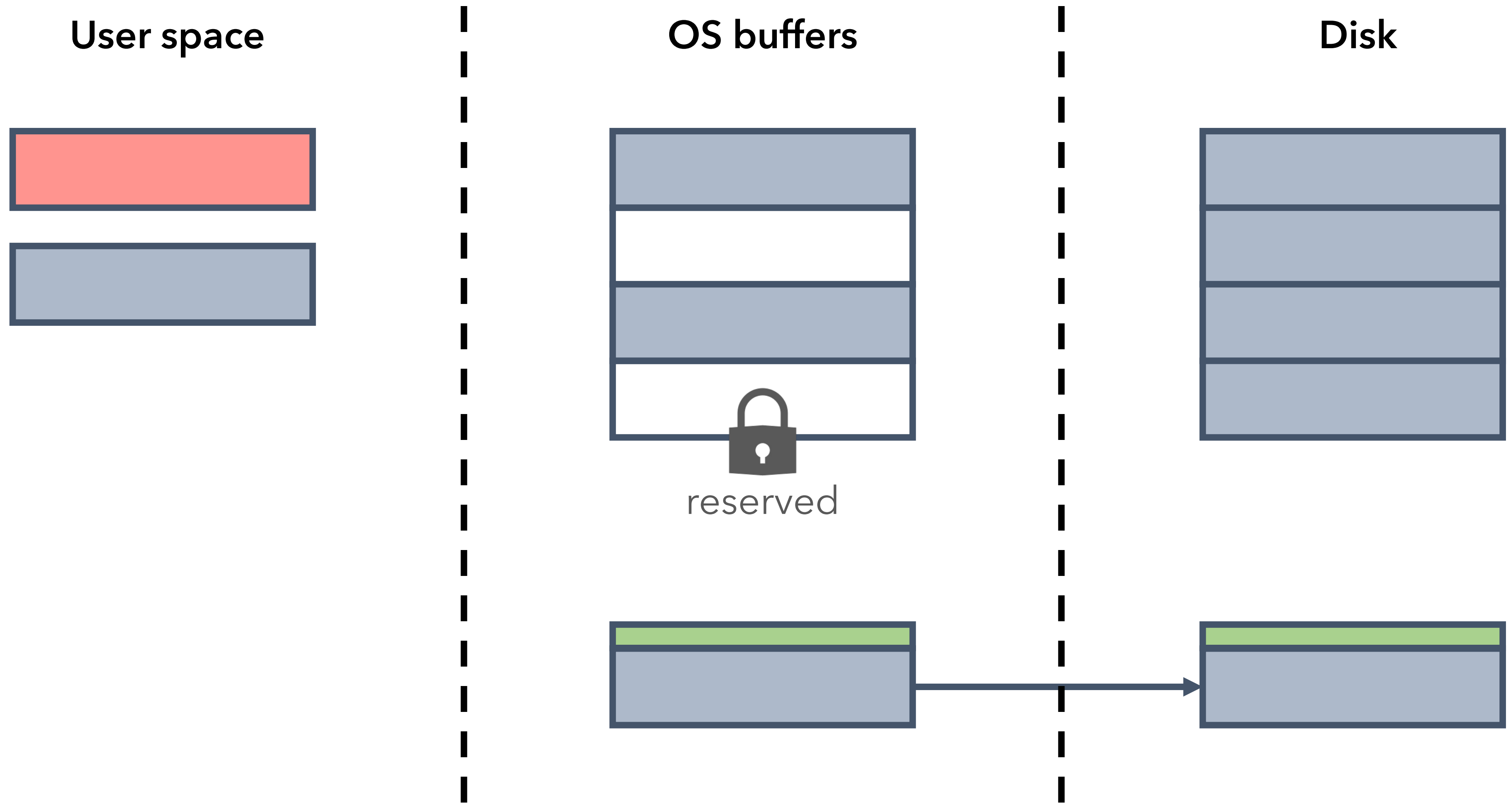
Rollback mode



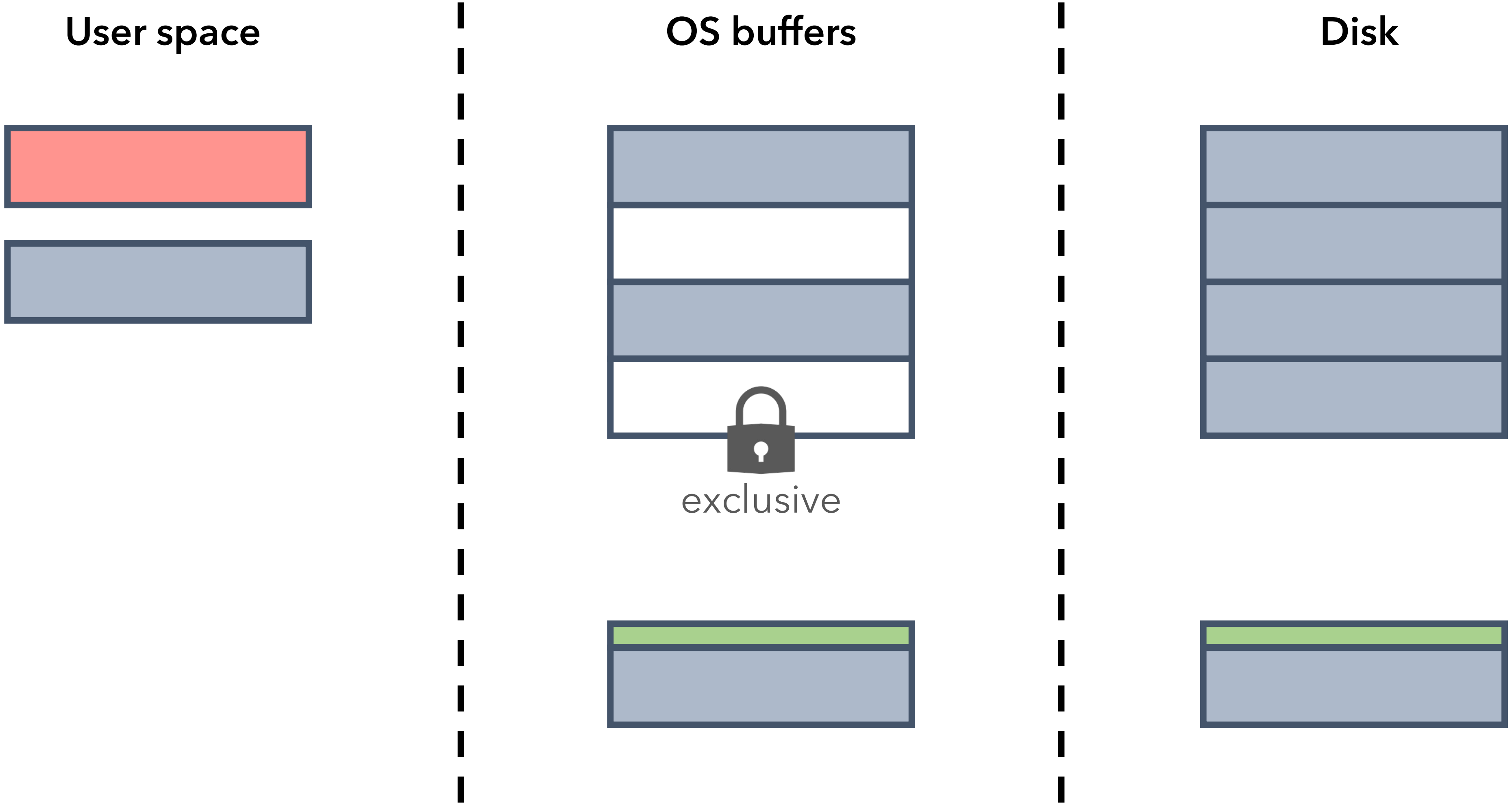
Rollback mode



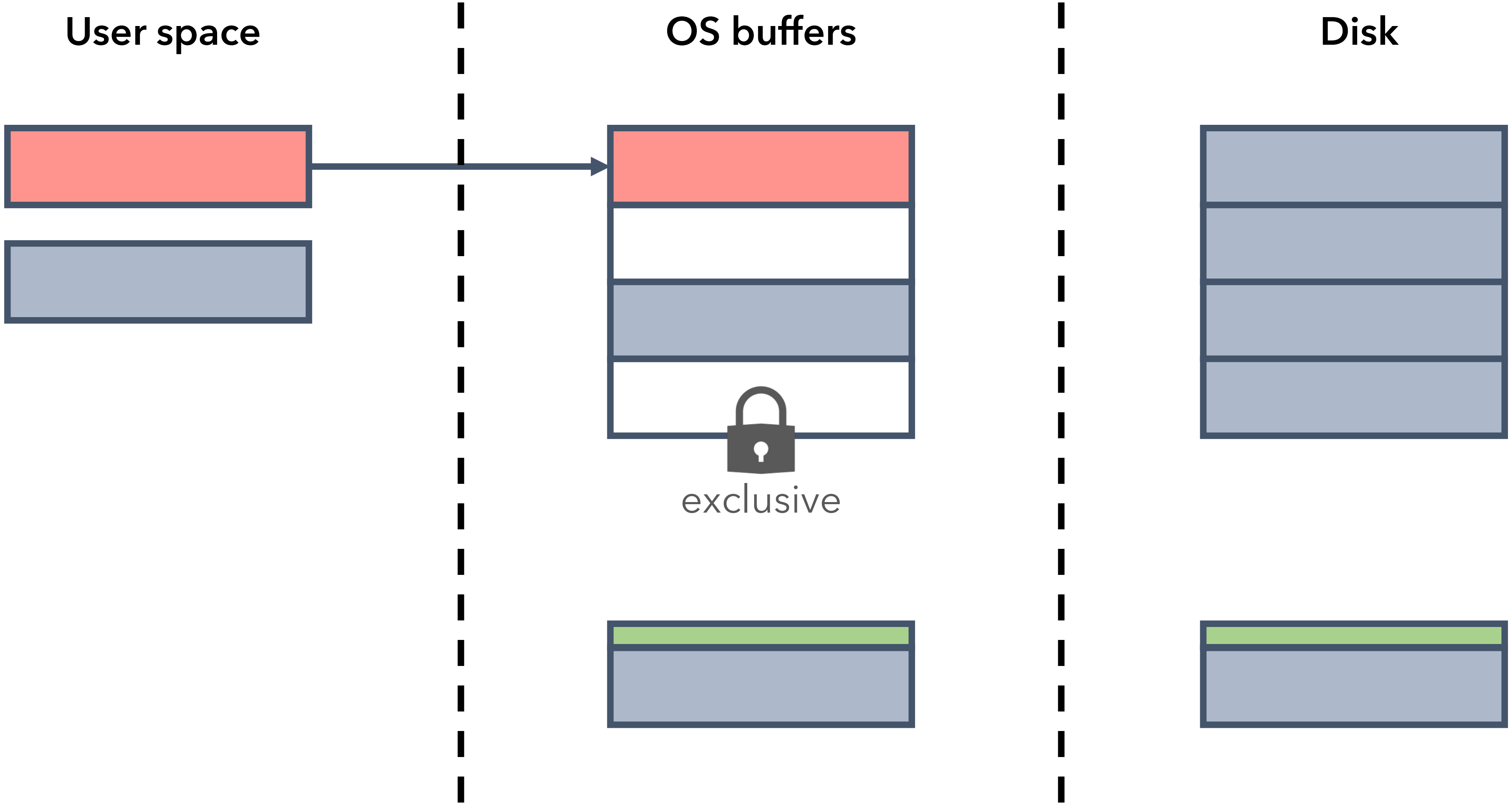
Rollback mode



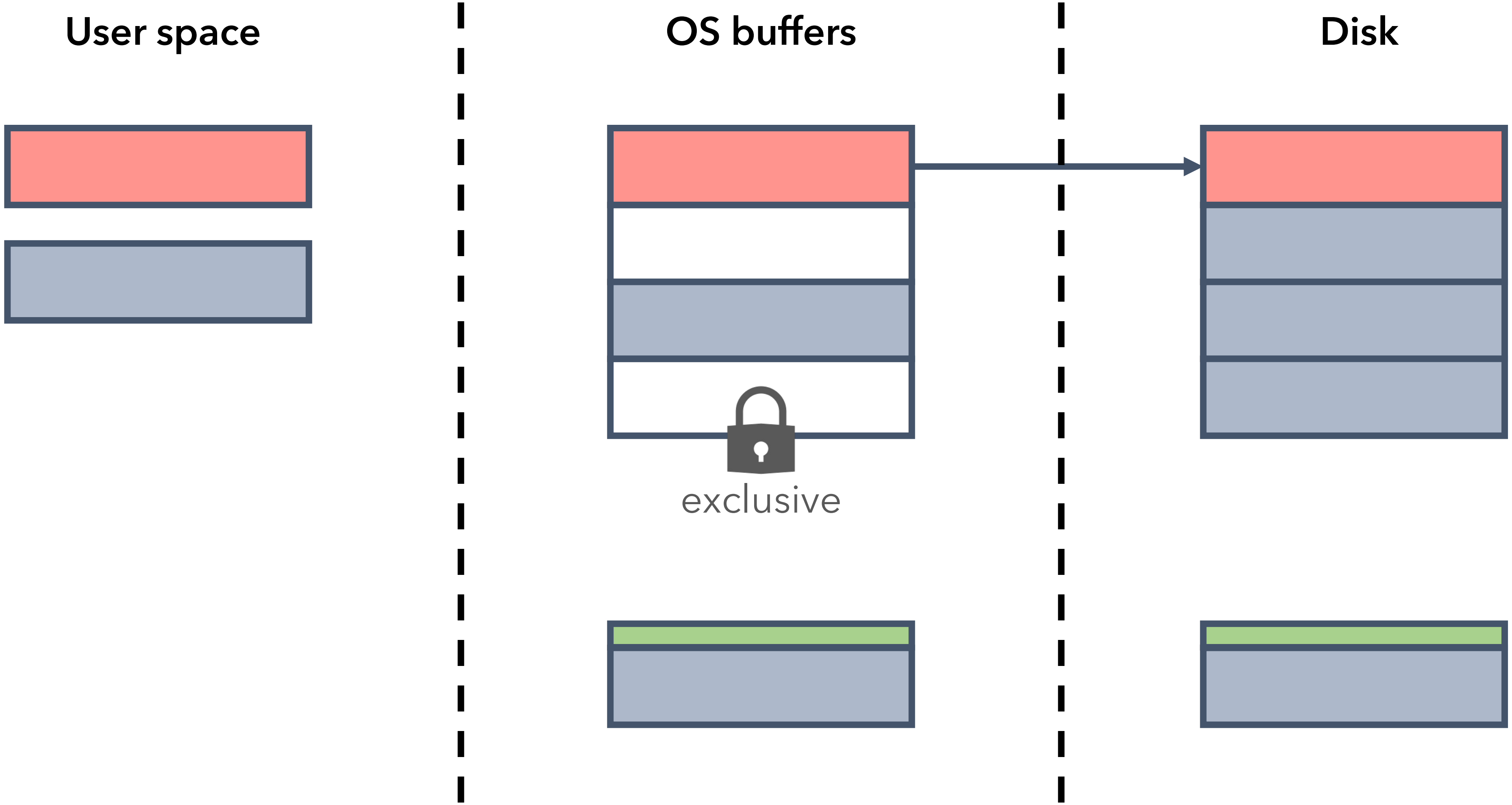
Rollback mode



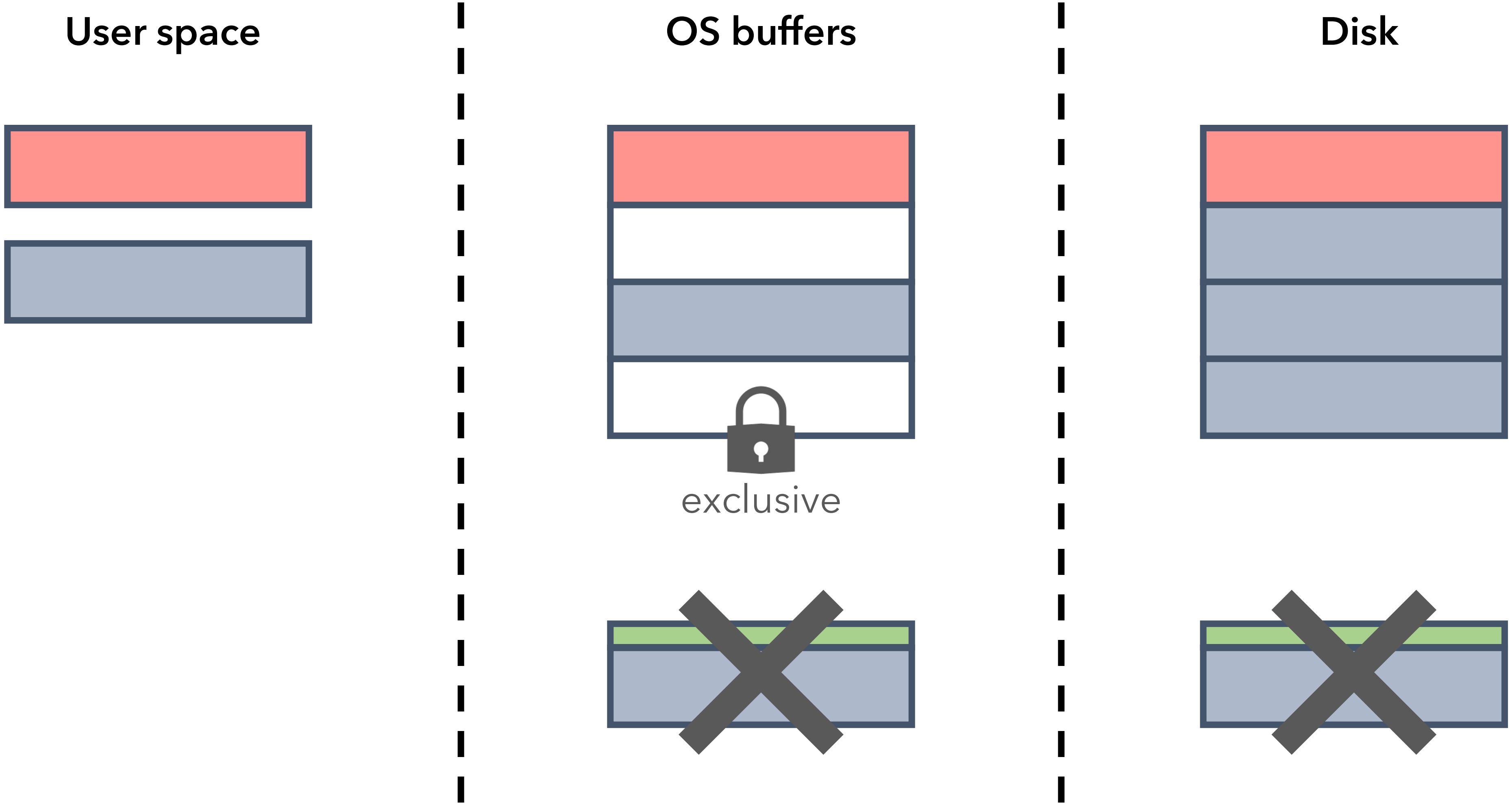
Rollback mode



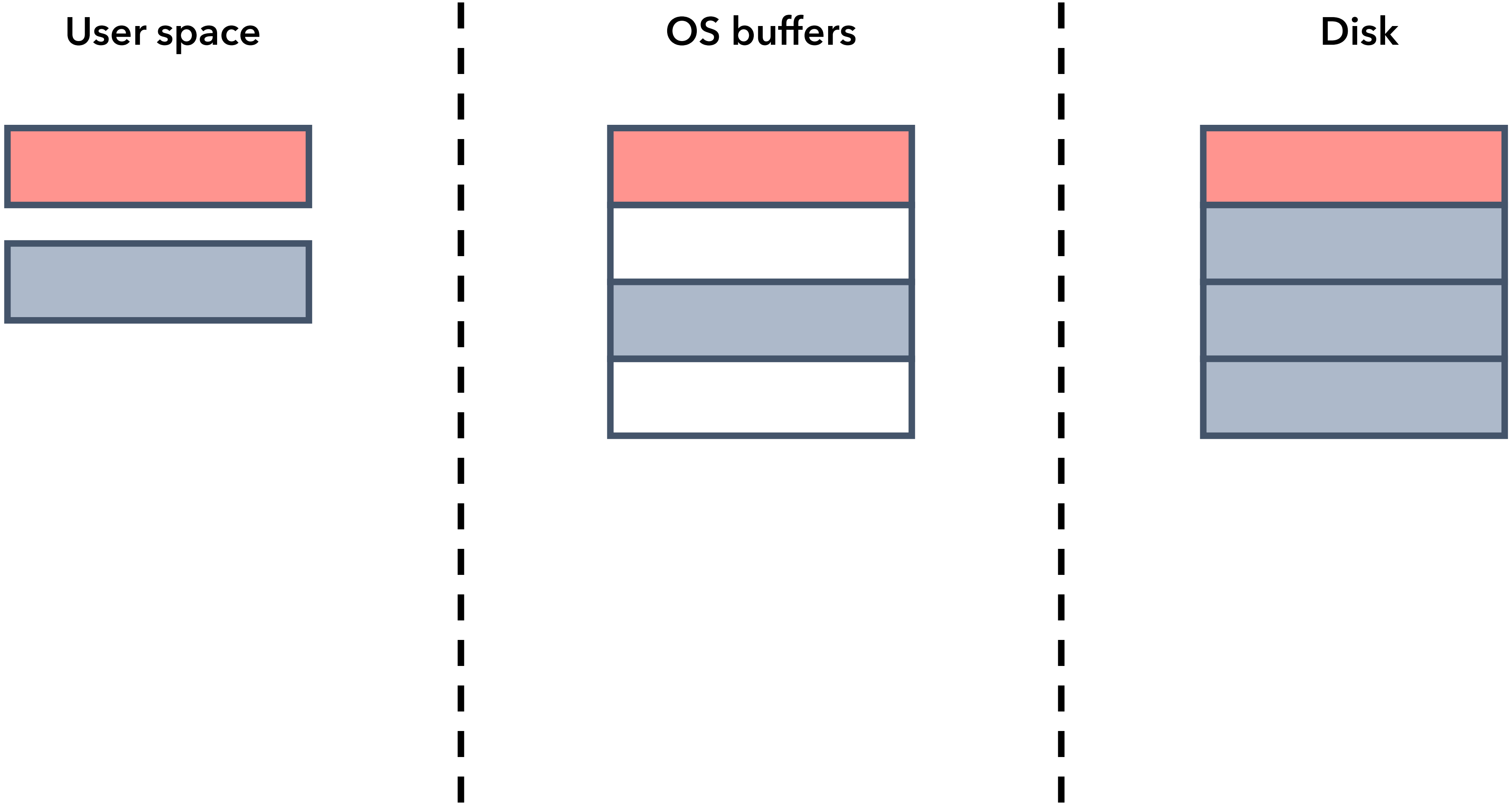
Rollback mode



Rollback mode



Rollback mode



Rollback mode

Invalidating the rollback journal

DELETE

The rollback journal is **deleted** at the end of the transaction.

TRUNCATE

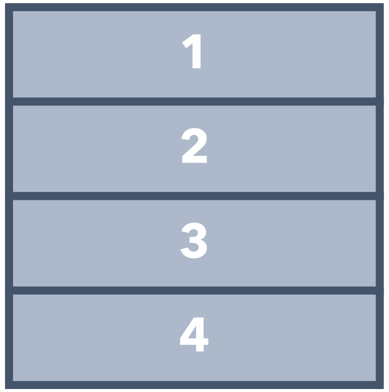
The rollback journal is **truncated** to zero length at the end of the transaction.

PERSIST

The rollback journal is **overwritten** with zeros at the end of the transaction.

Write-ahead log (WAL) mode

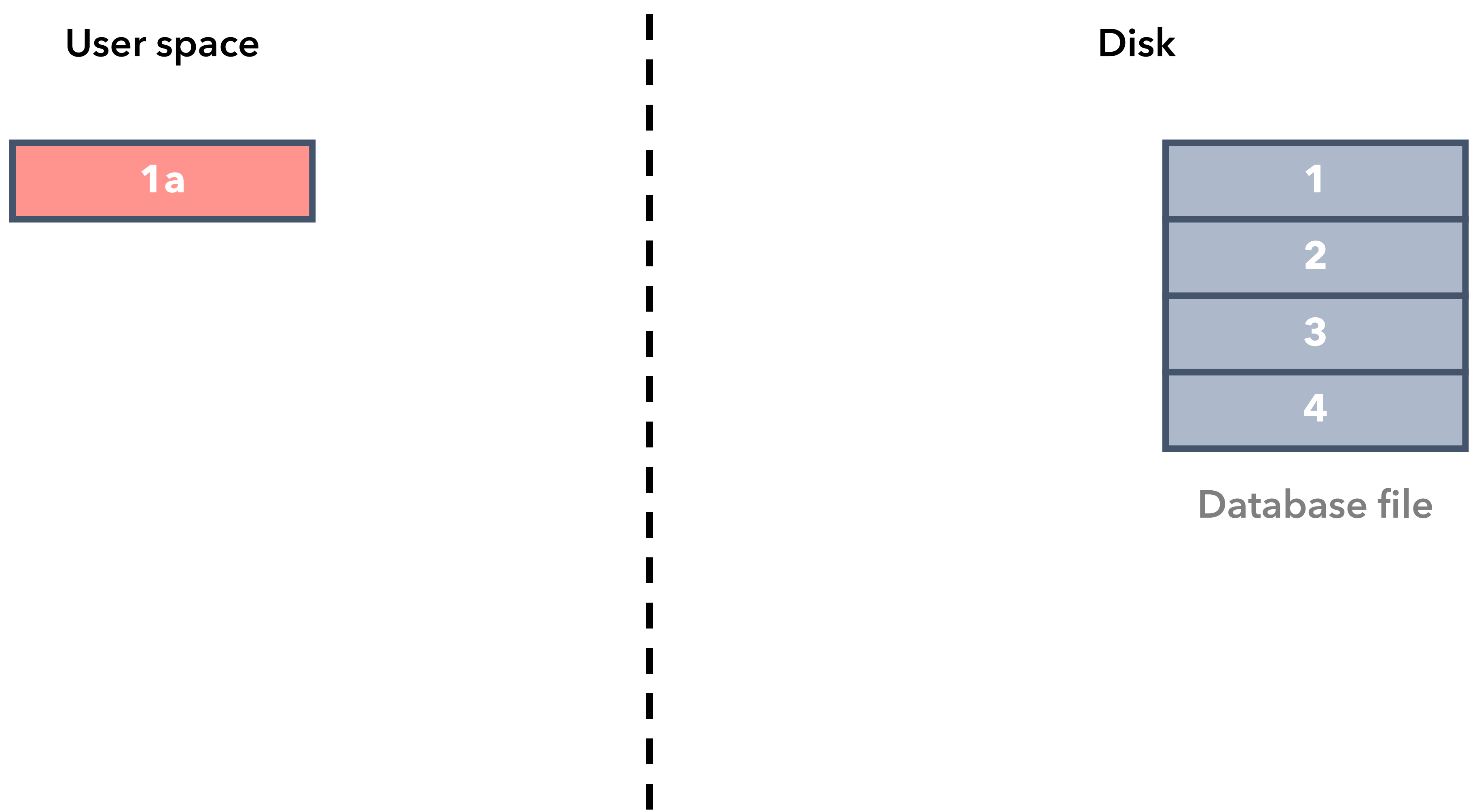
User space | **Disk**



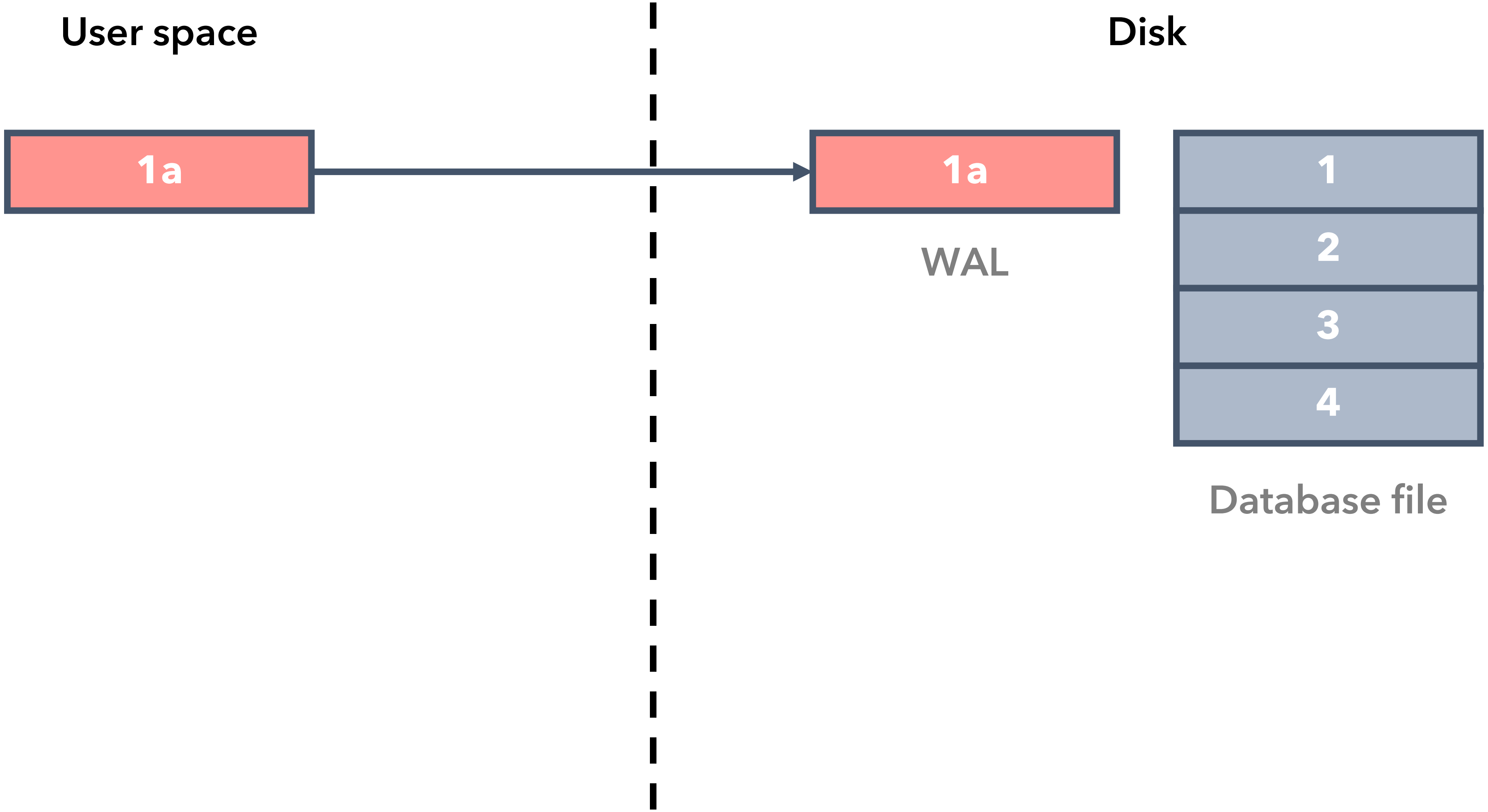
Database file

Write-ahead log (WAL) mode

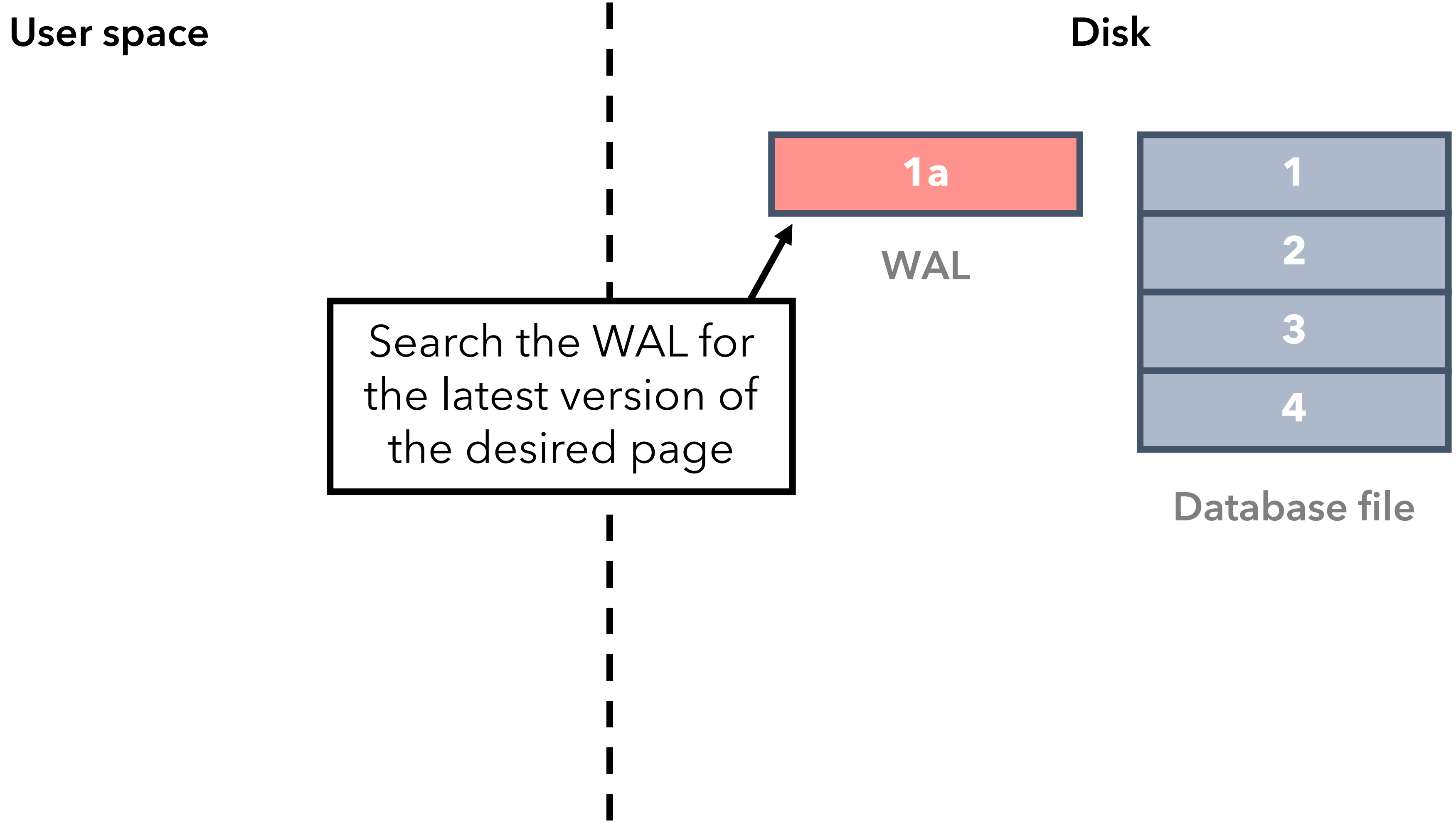
Write-ahead log (WAL) mode



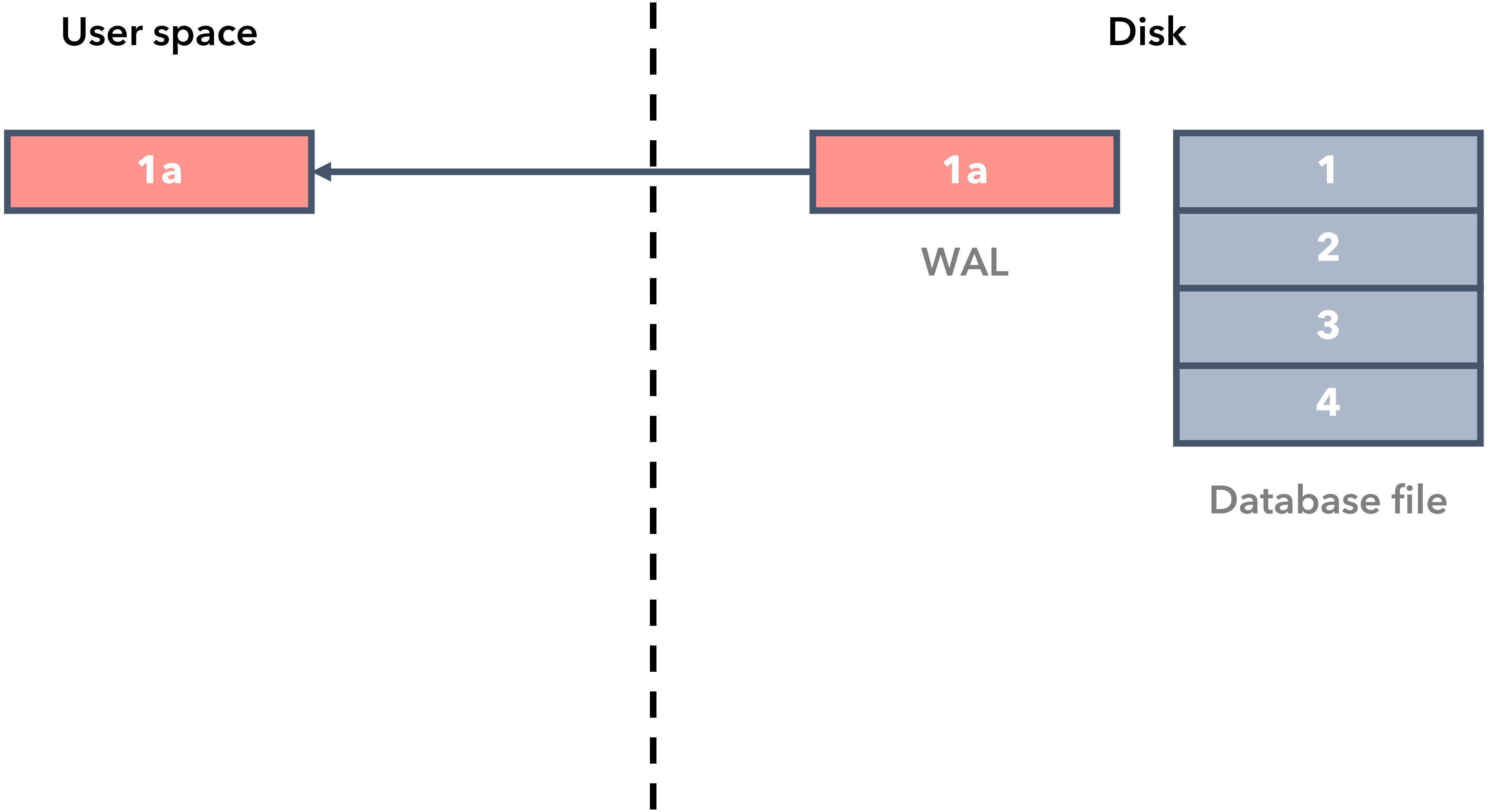
Write-ahead log (WAL) mode



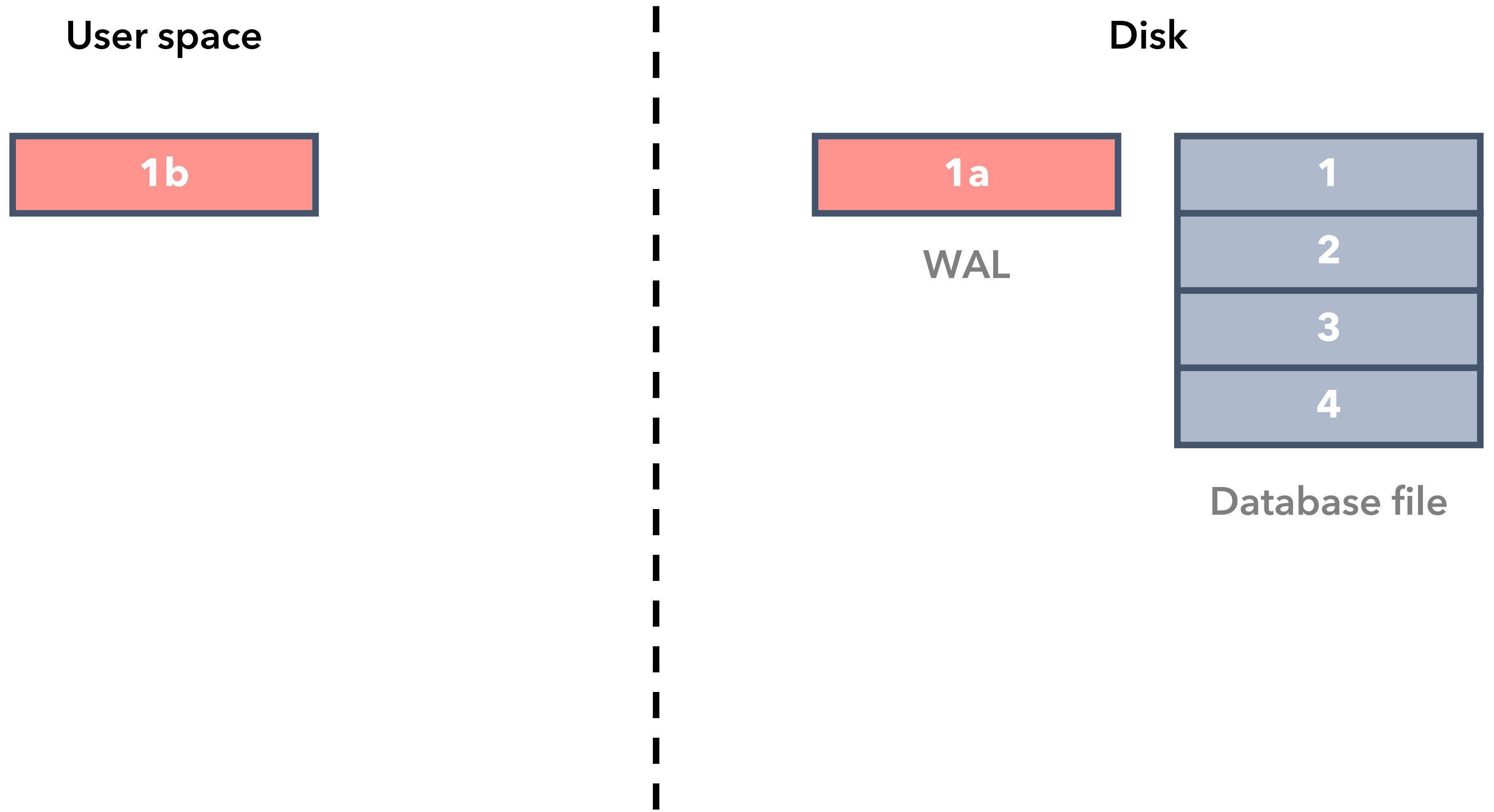
Write-ahead log (WAL) mode



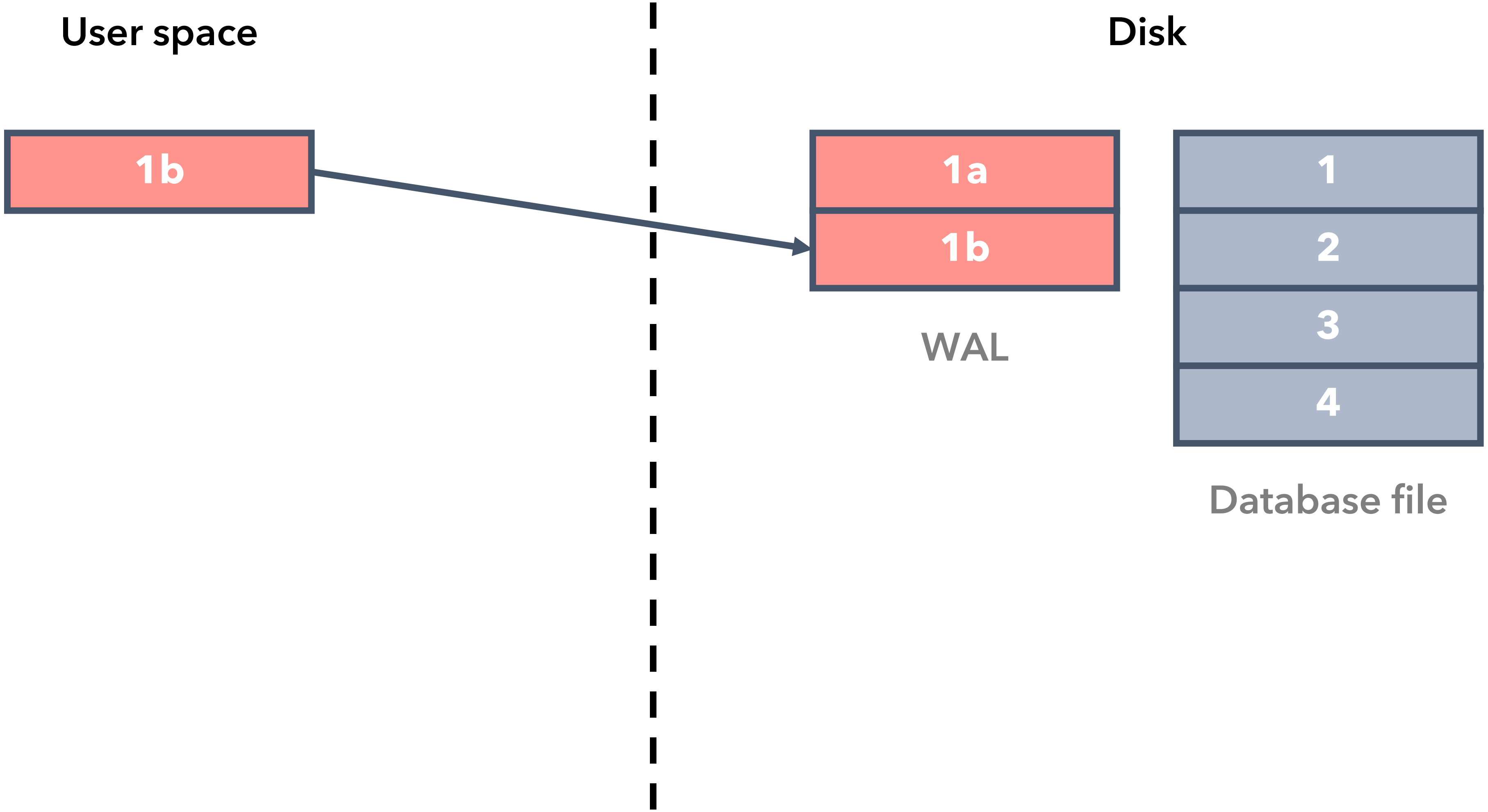
Write-ahead log (WAL) mode



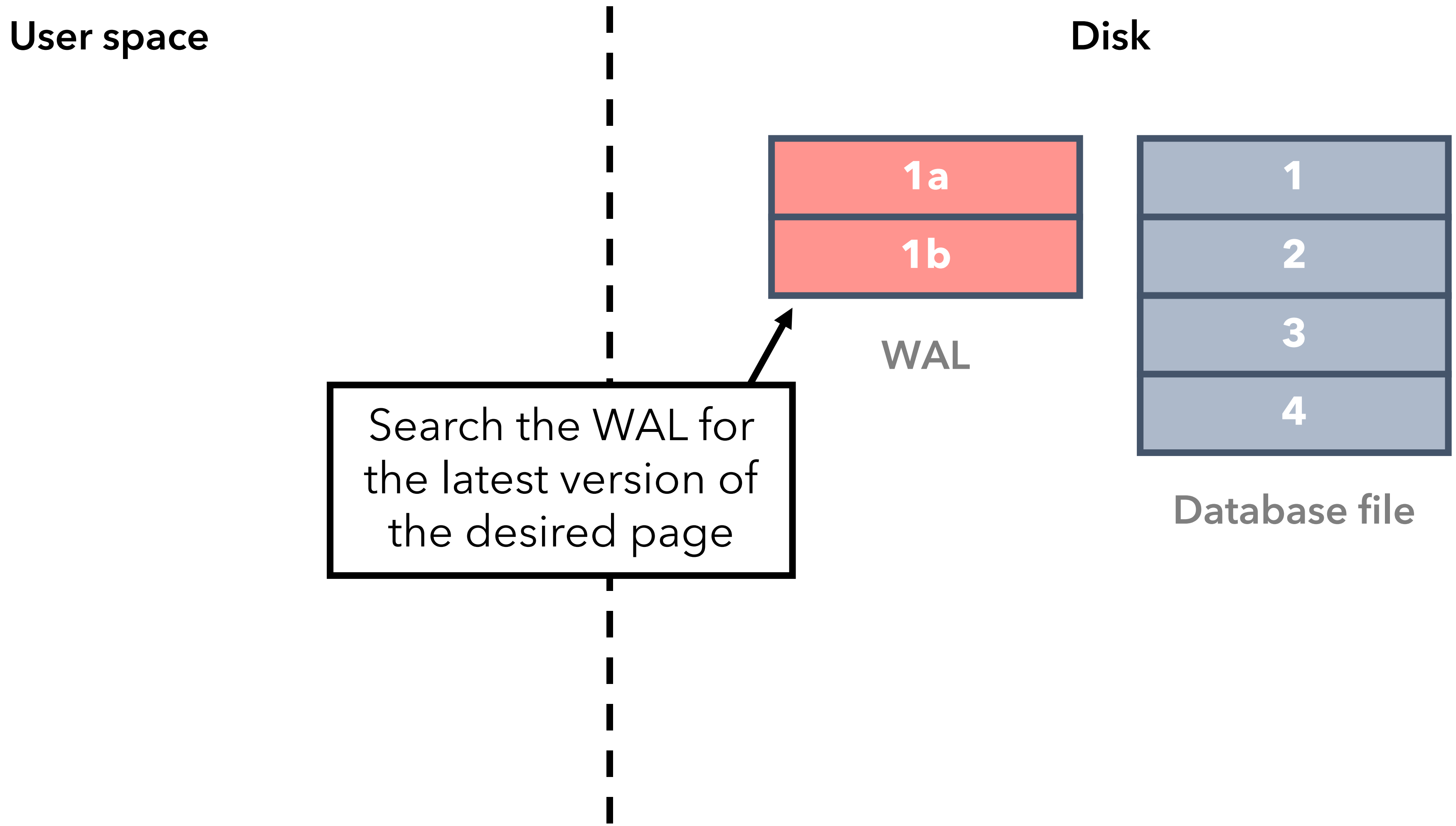
Write-ahead log (WAL) mode



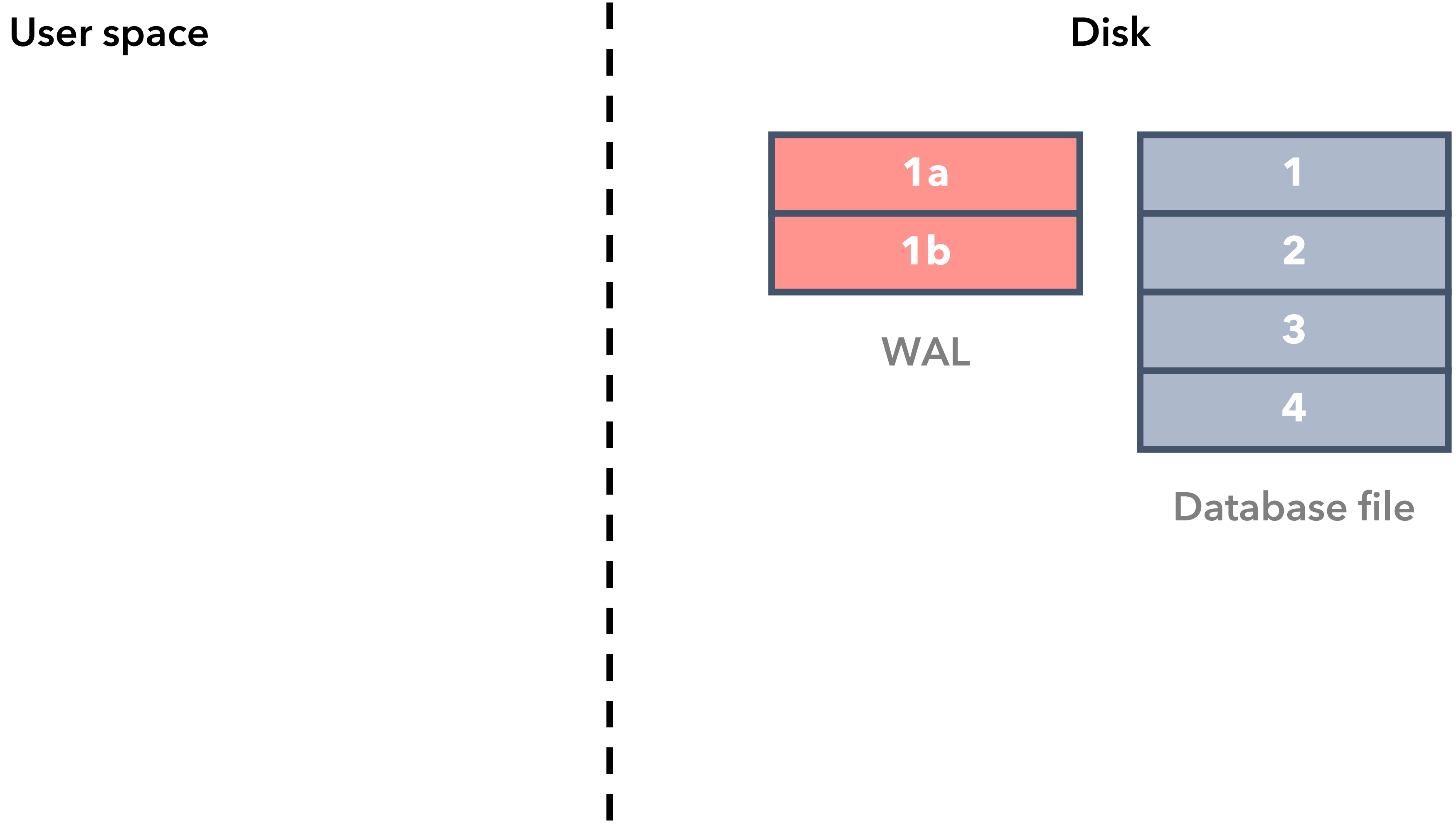
Write-ahead log (WAL) mode



Write-ahead log (WAL) mode



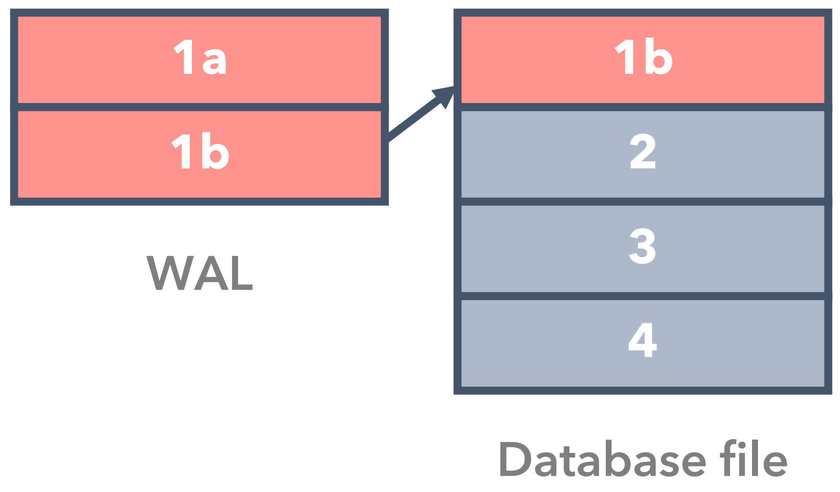
Write-ahead log (WAL) mode



Write-ahead log (WAL) mode

User space

Disk



Transaction modes

Performance considerations

Rollback mode	WAL mode
Allows unlimited readers **OR** one writer. Requires **two** writes for each modified page (one to the rollback journal and one to the database file).	Allows unlimited readers **AND** one writer. Requires **one** write for each modified page (and occasional checkpoints).

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

What is SQLite used for today?

Android mobile phone study

Large portion of workload consisted of key-value reads and writes.

Significant tail of complex OLAP queries involving nested SELECT statements and joins between several tables.

25% of all statements were writes.

DELETE statements were the most expensive and often involved nested SELECT statements.

Oliver Kennedy, Jerry Ajay, Geoffrey Challen, and Lukasz Ziarek. 2015. Pocket Data: The Need for TPC-MOBILE. In *Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*.

The rise of in-process OLAP

Interactive data analysis

Initial steps in data science workflows: selection, projection, join, aggregation.

ML datasets are often packaged as SQLite database files.

Python includes the sqlite3 module in the standard library.

Edge computing

Data analysis is often pushed to edge devices.

Reduces network traffic, server load, and transmission of privacy-sensitive data.

Requires lightweight but performant data analytics tools.

	SQLite	DuckDB
Storage format	Row-store	Column-store
Query execution	Row-by-row	Vectorized
Concurrency control	Lock-based	Batch-optimized MVCC
Parallelism	Inter-query only	Inter- and intra-query

How does SQLite perform on a variety of workloads, and how can we make it faster?

Goals

1. **Evaluate SQLite and DuckDB** together on a range of representative benchmarks, including online transaction processing (OLTP), OLAP, and BLOB processing.
2. **Identify key bottlenecks** slowing down SQLite's OLAP performance.
3. **Implement solutions** to improve SQLite's OLAP performance.

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

Performance evaluation

Transaction processing

Benchmark: **TATP**.

Measure: **throughput**.

Tests efficiency of index searches and small reads, updates, inserts, and deletes.

Analytical processing

Benchmark: **SSB**.

Measure: **latency**.

Tests efficiency of selections, projections, joins, and aggregates.

BLOB processing

Benchmark: **BLOB**.

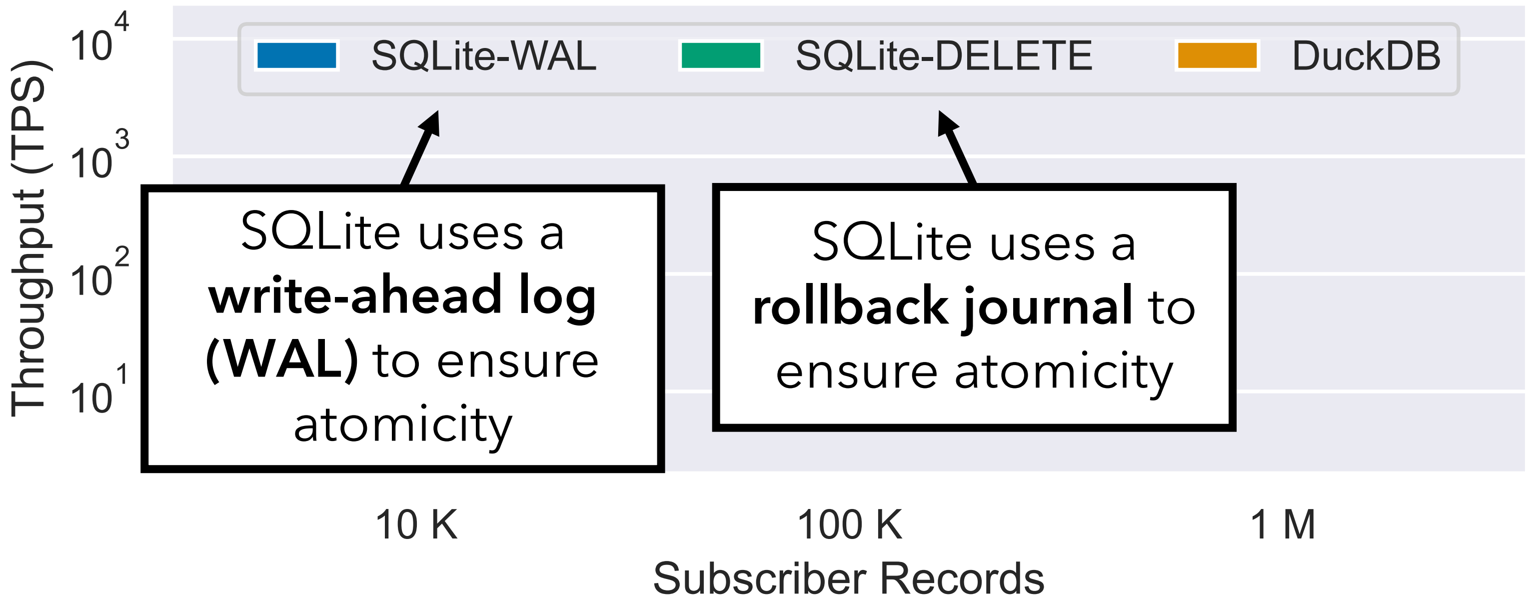
Measure: **throughput**.

Tests efficiency of transactional reads and writes of large binary data.

Hardware: Cloud server and Raspberry Pi. Results are shown for the Raspberry Pi.

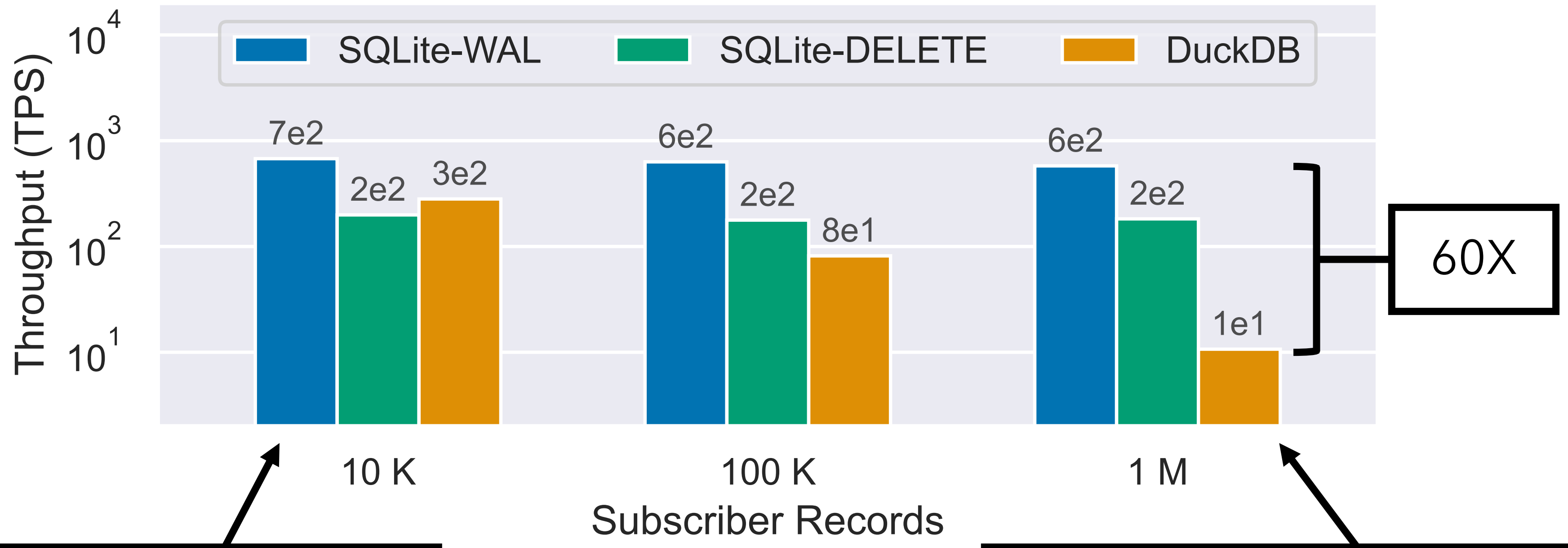
Transaction processing

Representative benchmark: **TATP**



Transaction processing

Representative benchmark: **TATP**

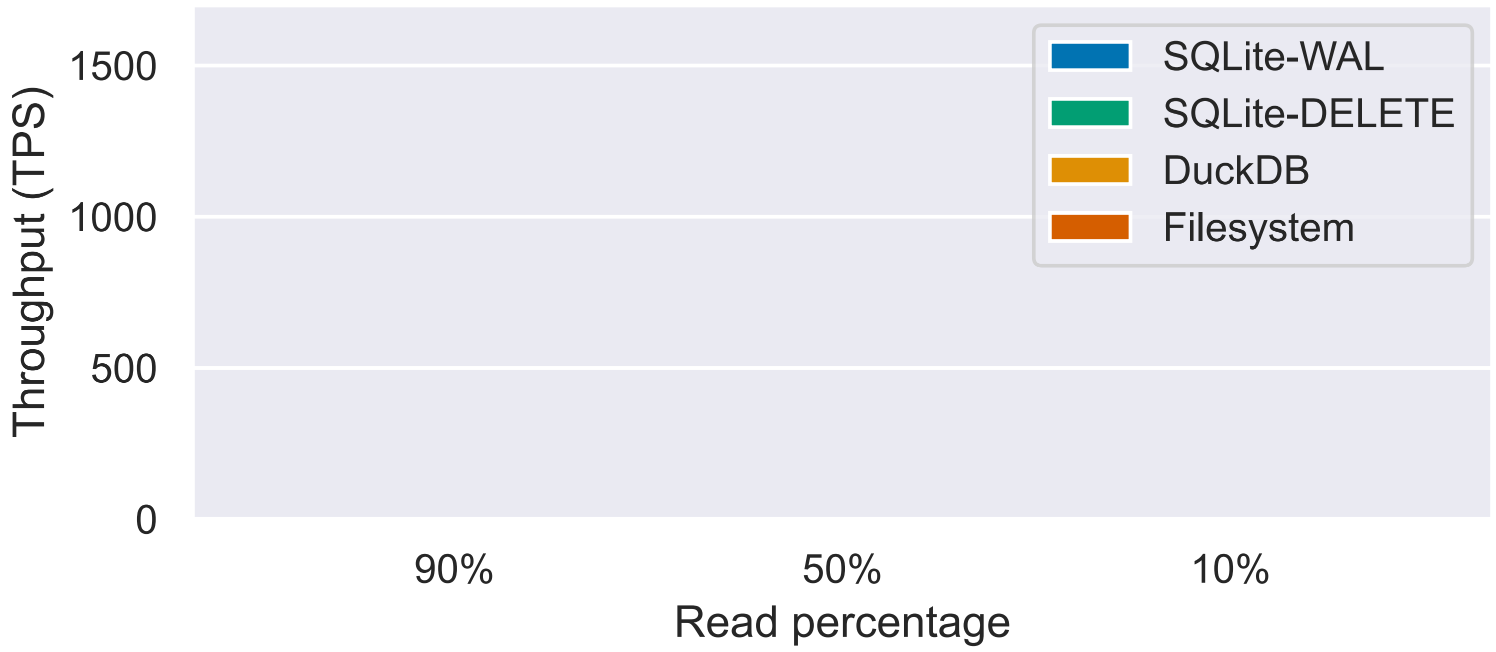


SQLite's WAL journal mode is ~3X faster than DELETE journal mode

DuckDB's throughput degrades as the benchmark scales; SQLite's performance is steady

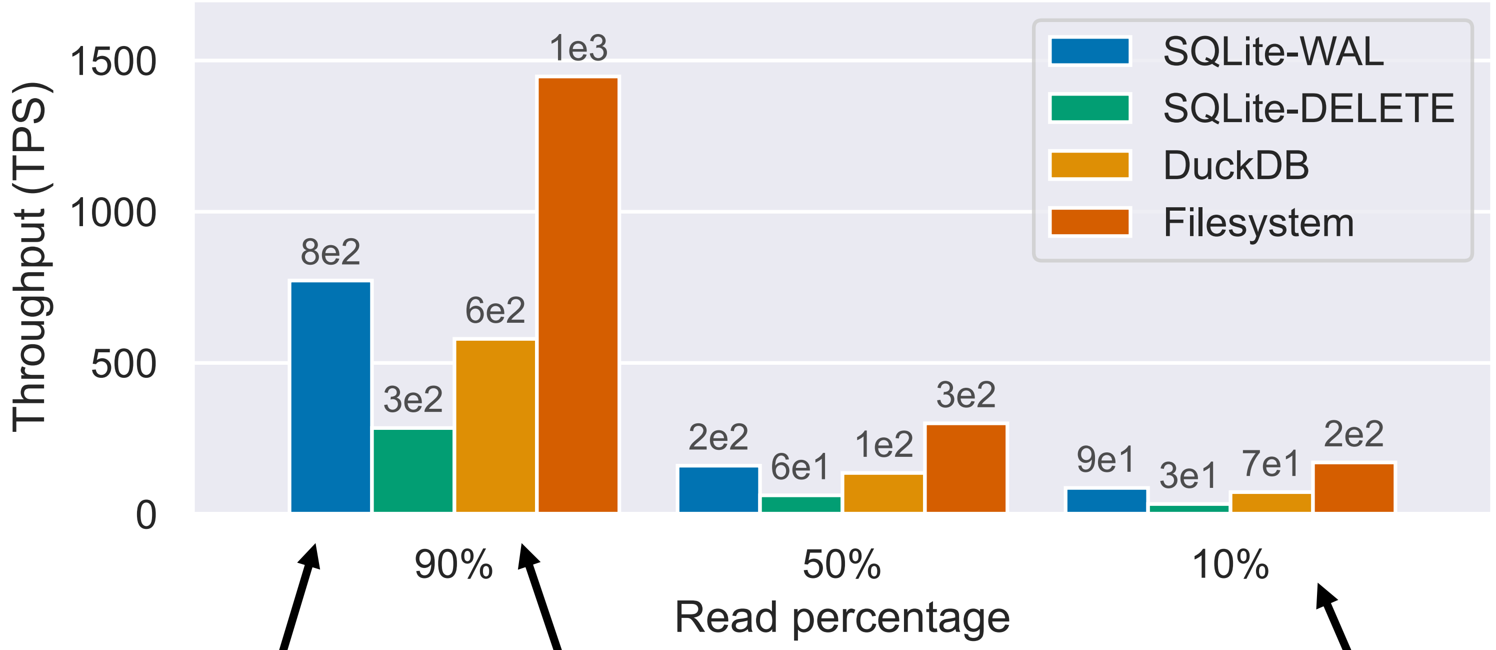
BLOB processing

Representative benchmark: **BLOB (100 KB)**



BLOB processing

Representative benchmark: **BLOB (100 KB)**

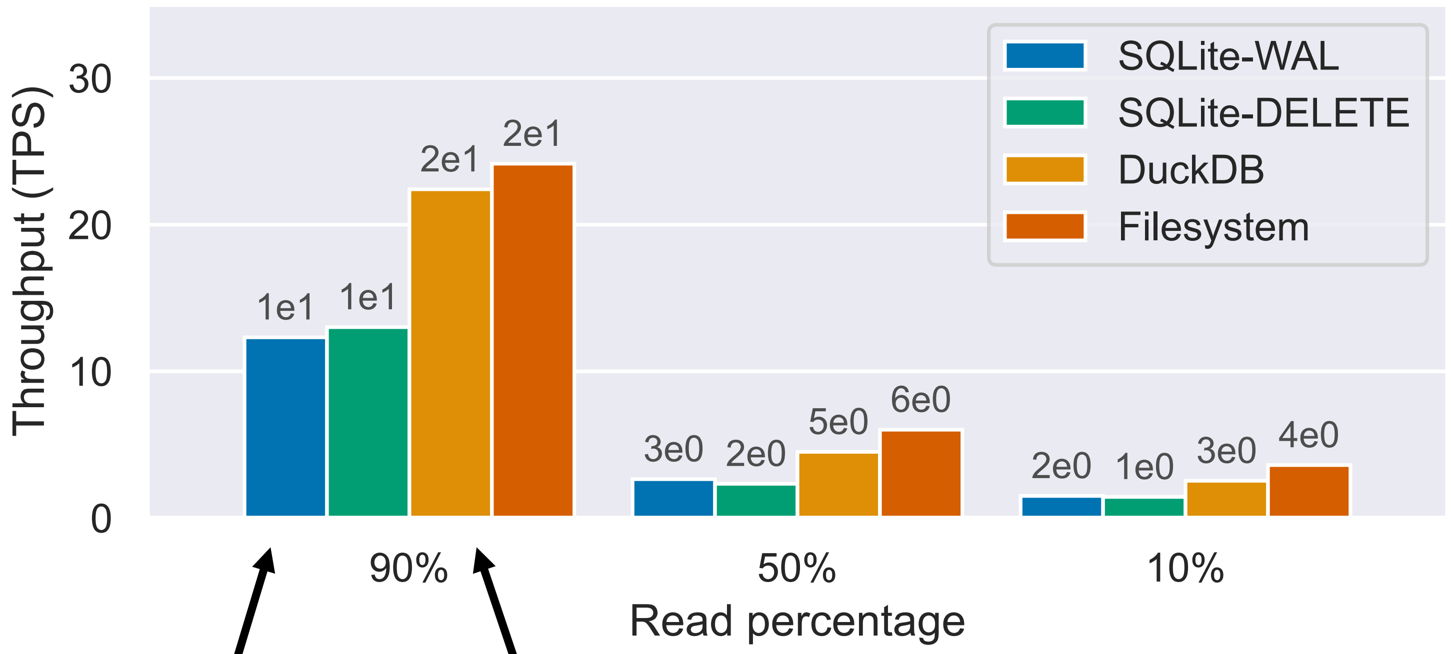


SQLite-WAL is over 30% faster than DuckDB

All systems decrease in throughput as read percentage decreases

BLOB processing

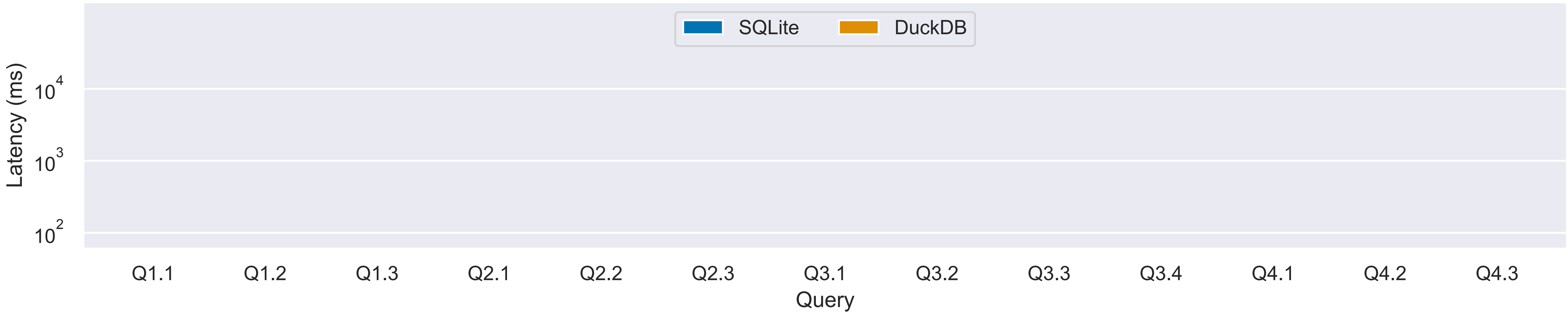
Representative benchmark: **BLOB (10 MB)**



For large BLOBs, DuckDB is up to 2X faster

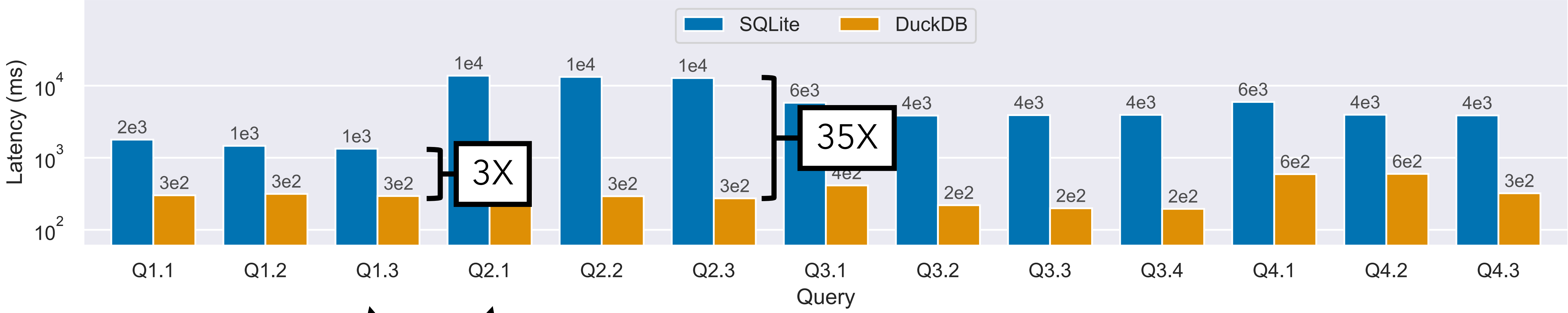
Analytical processing

Representative benchmark: **SSB**



Analytical processing

Representative benchmark: **SSB**



SQLite has more inter-query performance variation (up to 10X)

DuckDB's fastest queries are in flight 3; SQLite's fastest are in flight 1

Resource footprints

Library footprints

System	Library size	Compile time	Compile memory
SQLite (-Os)	900 KB	15 s	340 MB
SQLite (-O3)	1.5 MB	30 s	380 MB
DuckDB (-Os)	32 MB	5 m	7.7 GB
DuckDB (-O3)	37 MB	10 m	7.6 GB

Database footprints

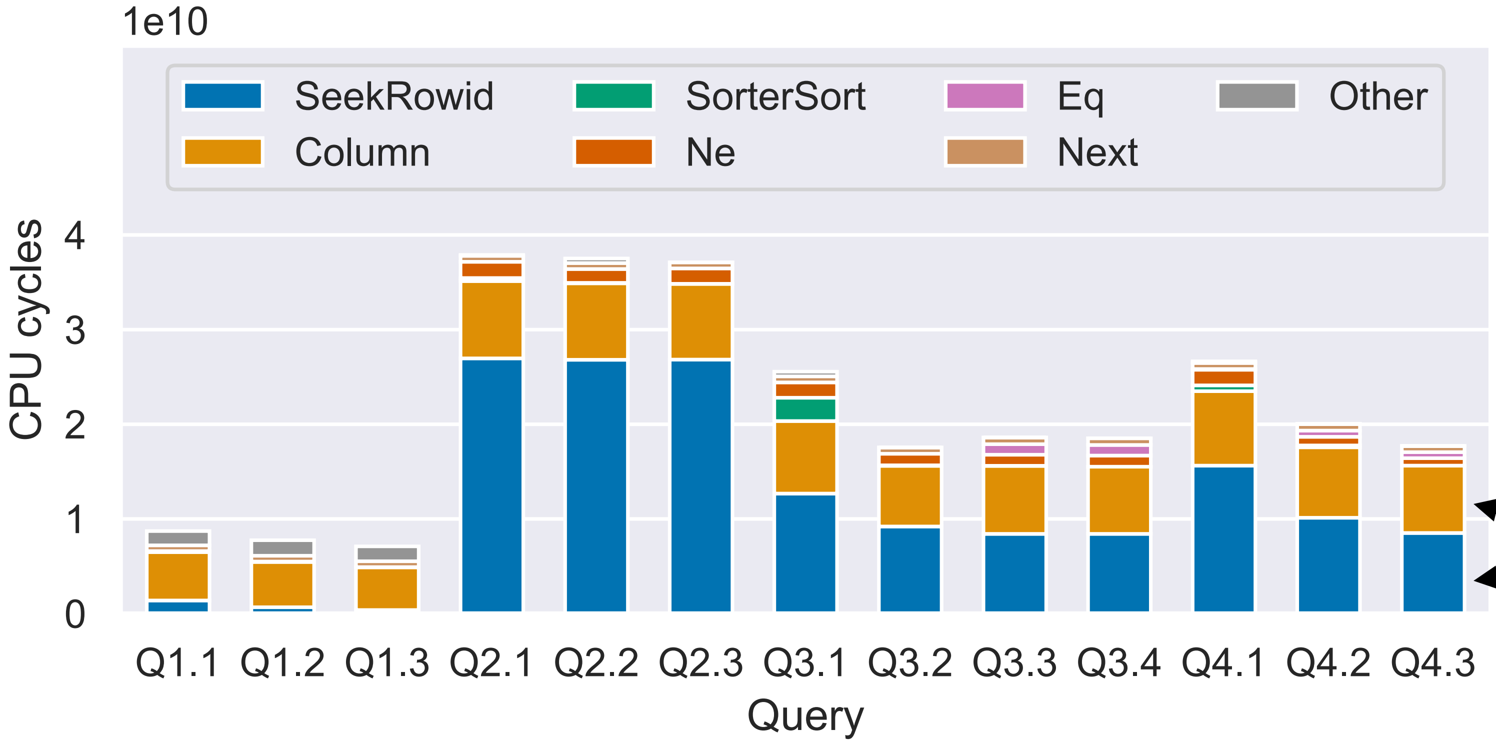
System	TATP size	SSB size	SSB load time
SQLite	520 MB	2.8 GB	82 s
DuckDB	270 MB	1.8 GB	100 s

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

SSB performance profiling

Which of SQLite's virtual instructions take the most cycles?



`SeekRowid` and `Column` are responsible for the vast majority of cycles

SeekRowid and Column instructions

`SeekRowid`

Given a row ID (*i.e.*, primary key), return a cursor to the corresponding row in a table.

`Column`

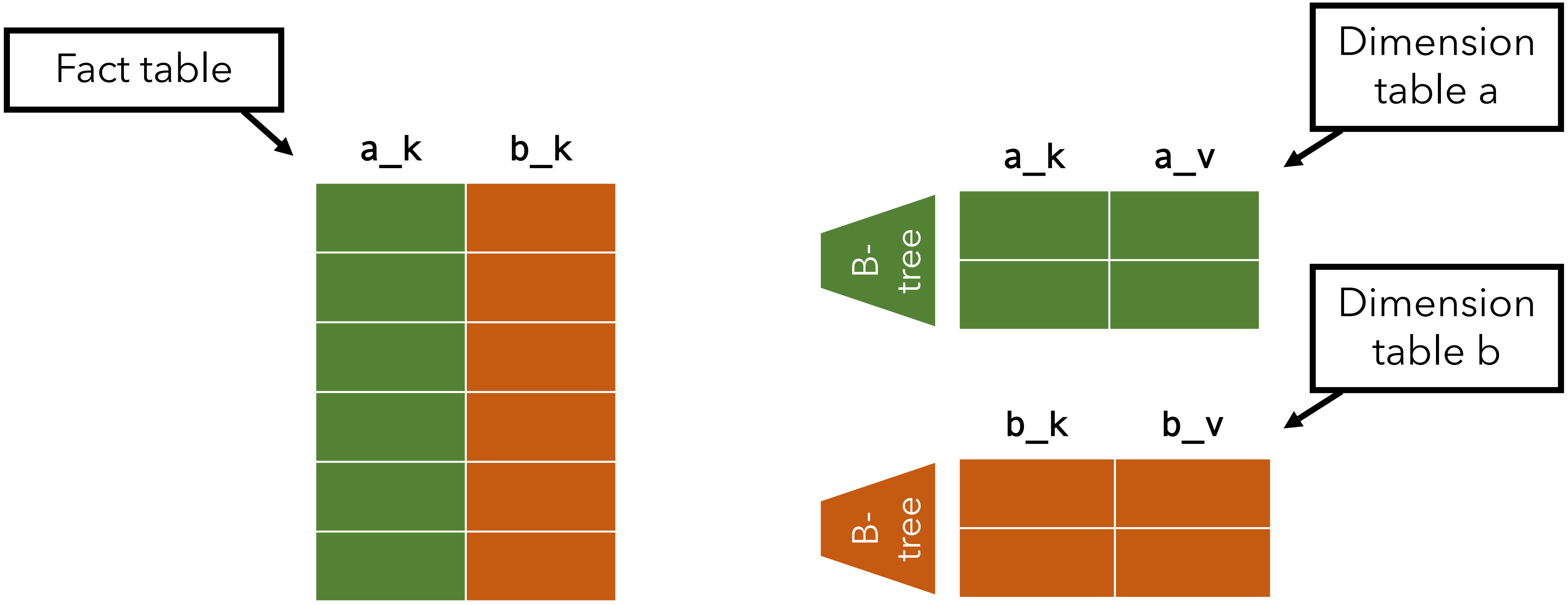
Given a cursor to a row in a table, return the value of a specified column in the row.

SeekRowid and Column instructions

`SeekRowid`	`Column`
Given a row ID (*i.e.*, primary key), return a cursor to the corresponding row in a table.	Given a cursor to a row in a table, return the value of a specified column in the row.

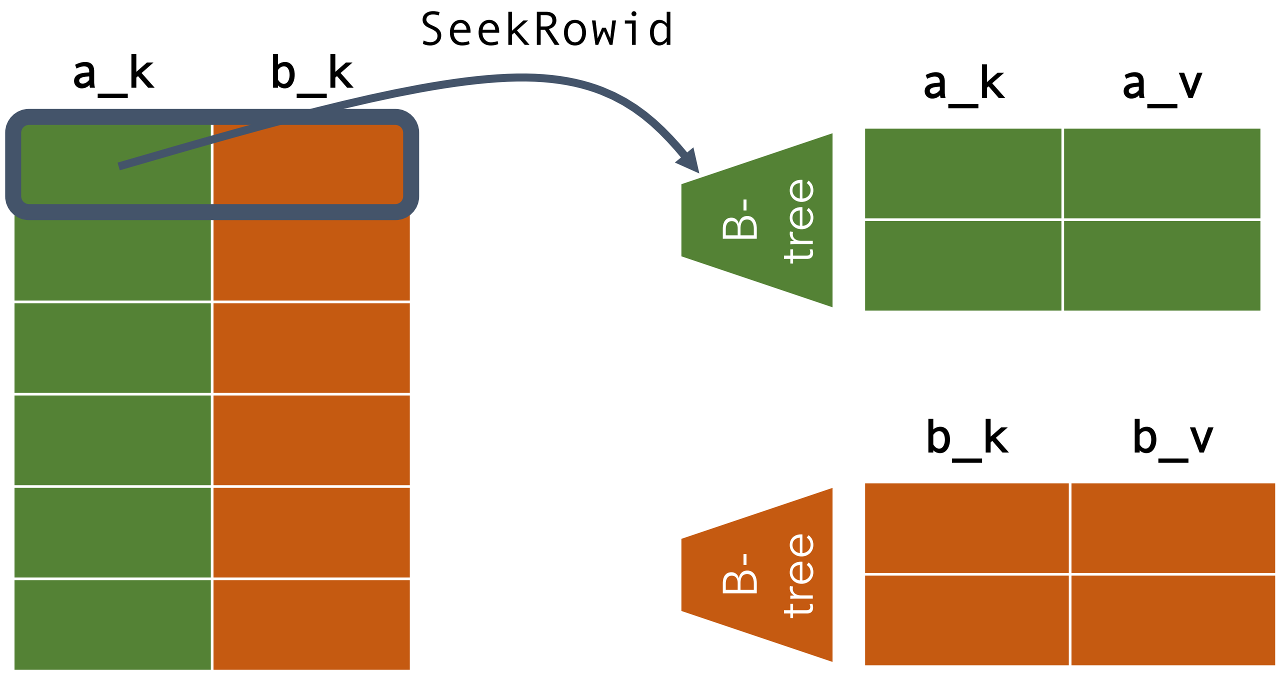
Joins in SQLite

(Index) nested loops



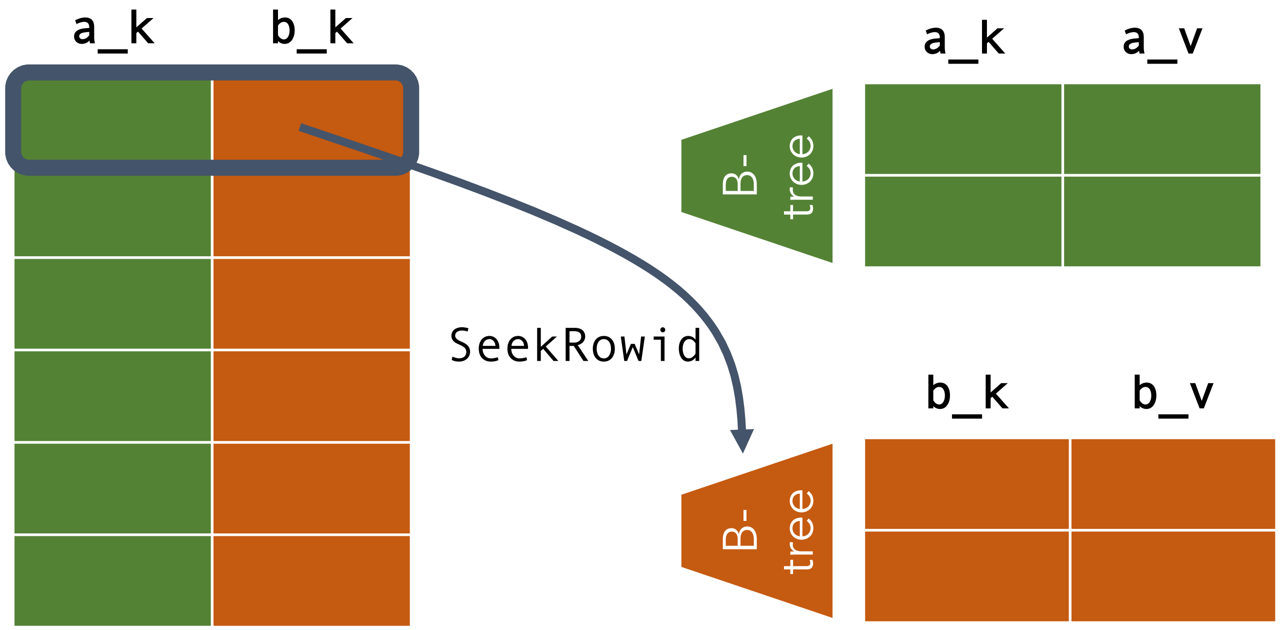
Joins in SQLite

(Index) nested loops



Joins in SQLite

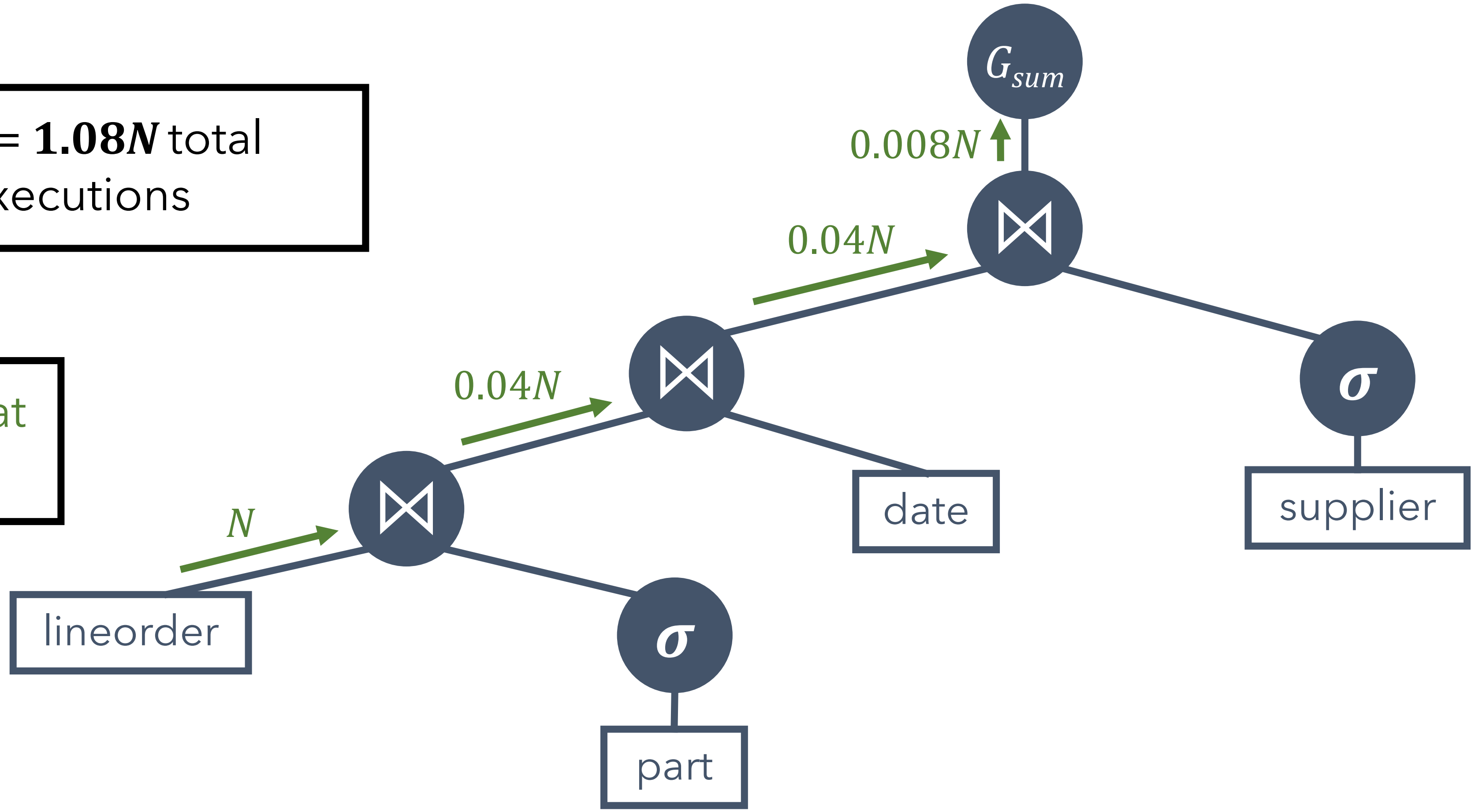
(Index) nested loops



Analyzing `SeekRowid` executions

$(1 + 0.04 + 0.04)N = \mathbf{1.08N}$ total `SeekRowid` executions

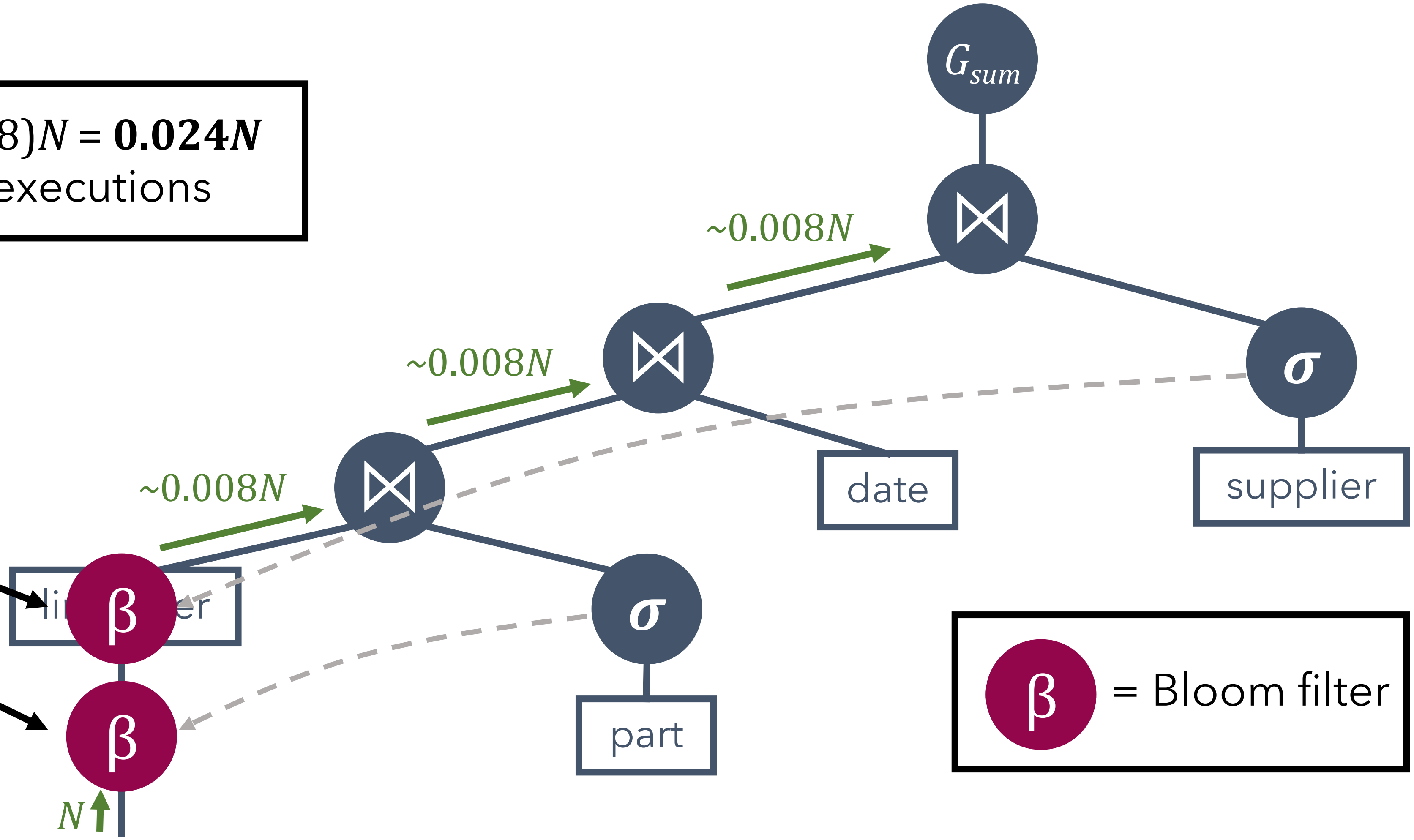
Number of tuples that survive each join



Reducing `SeekRowid` executions with LIP

$(0.008 + 0.008 + 0.008)N = \mathbf{0.024N}$
total `SeekRowid` executions

LIP: applying Bloom filters early in the pipeline substantially reduces the number of join probes (assuming low false positive rate)



Jianqiao Zhu, Navneet Potti, Saket Saurabh, Jignesh M. Patel. 2017. Looking Ahead Makes Query Plans Robust. *PVLDB*.

Implementing LIP in SQLite

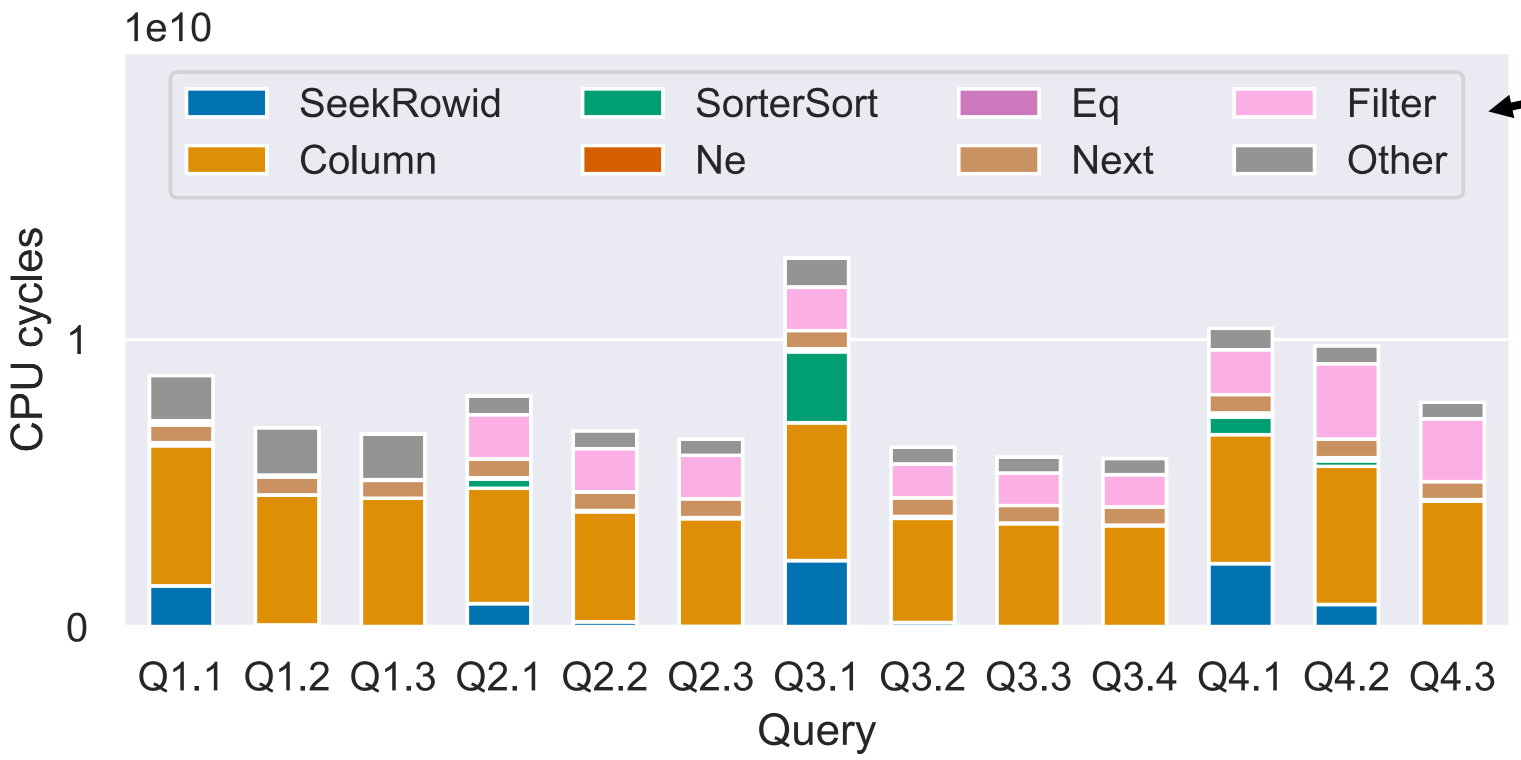
Two new virtual instructions: `FilterAdd` and `Filter`.

Build a Bloom filter if:

1. The number of rows in the table is known.
2. The number of expected searches exceeds the number of rows.
3. Some searches are expected to find zero rows.

Push all Bloom filter probes to the beginning of the outer table loop
(**before probes of the inner tables**).

Reducing `SeekRowid` executions with LIP

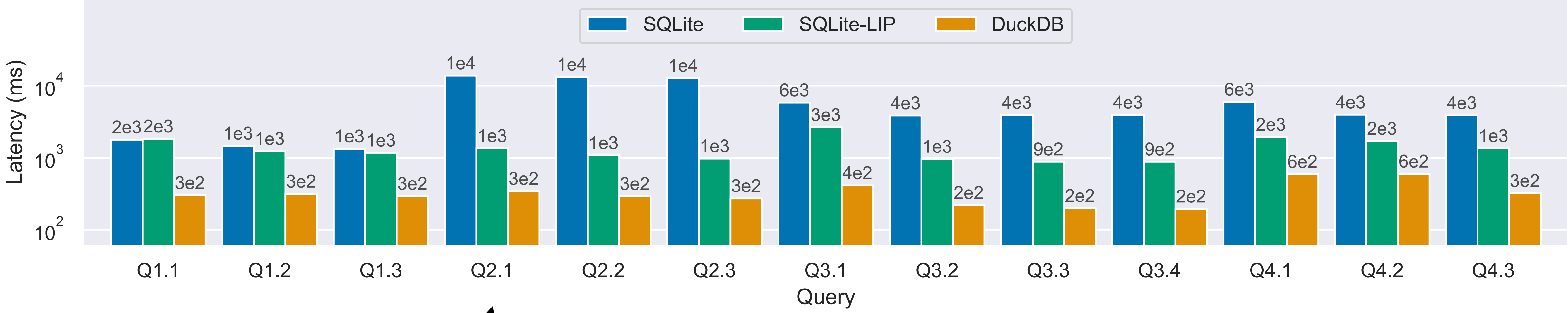


`Filter` instruction is a Bloom filter probe.

The number of cycles spent in `SeekRowid` is substantially reduced

Analytical processing with optimized SQLite

Representative benchmark: **SSB**



Some queries up to 10X faster

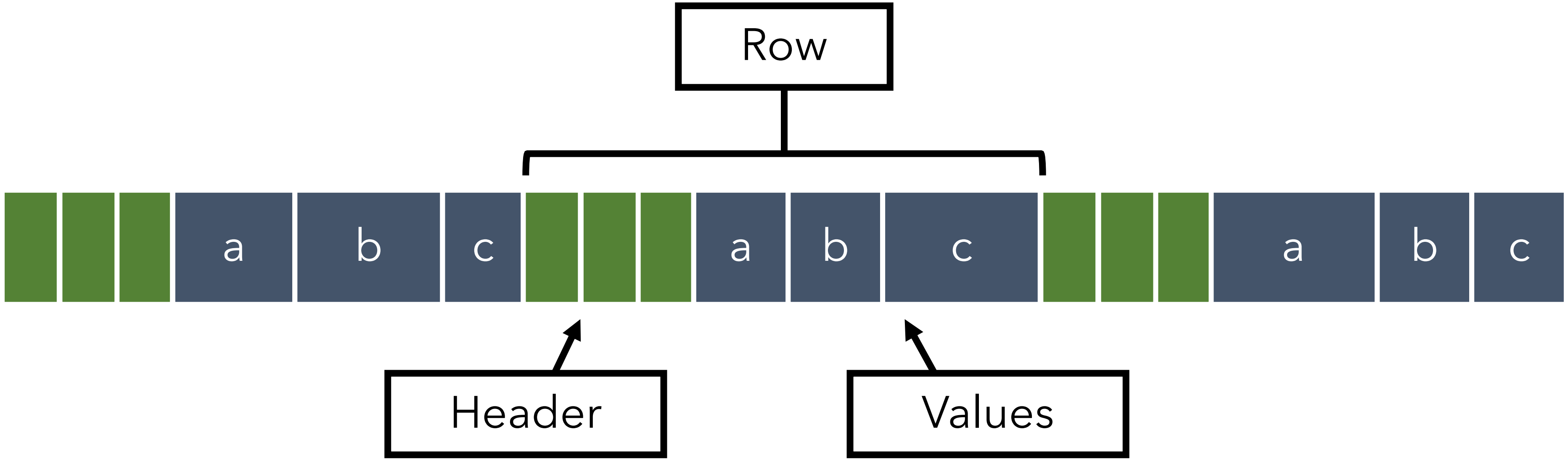
4.2X speedup overall

Roadmap

- Background
- Motivation
- Evaluation
- Optimization
- Discussion

Future optimizations

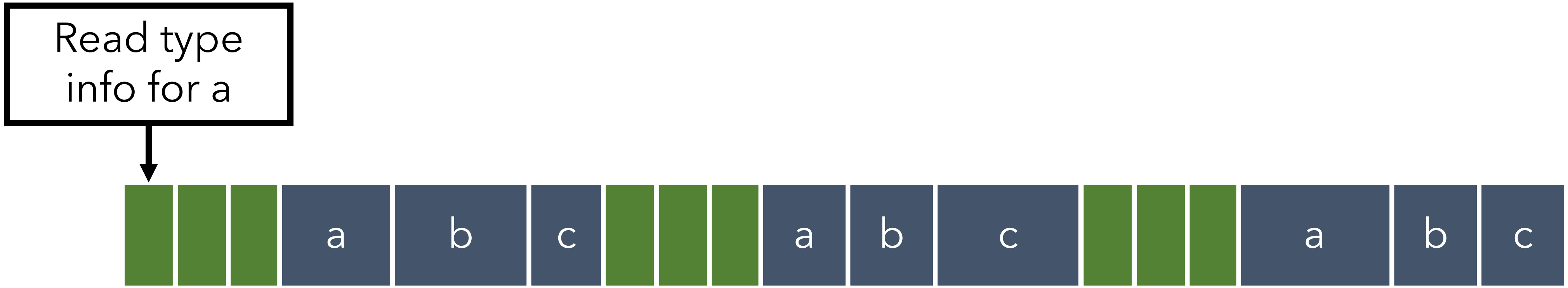
Streamlining value extraction (`Column` instruction)



Future optimizations

Streamlining value extraction (`Column` instruction)

```sql
SELECT c FROM table;
```



Future optimizations

Streamlining value extraction (`Column` instruction)

```sql
SELECT c FROM table;
```

Future optimizations

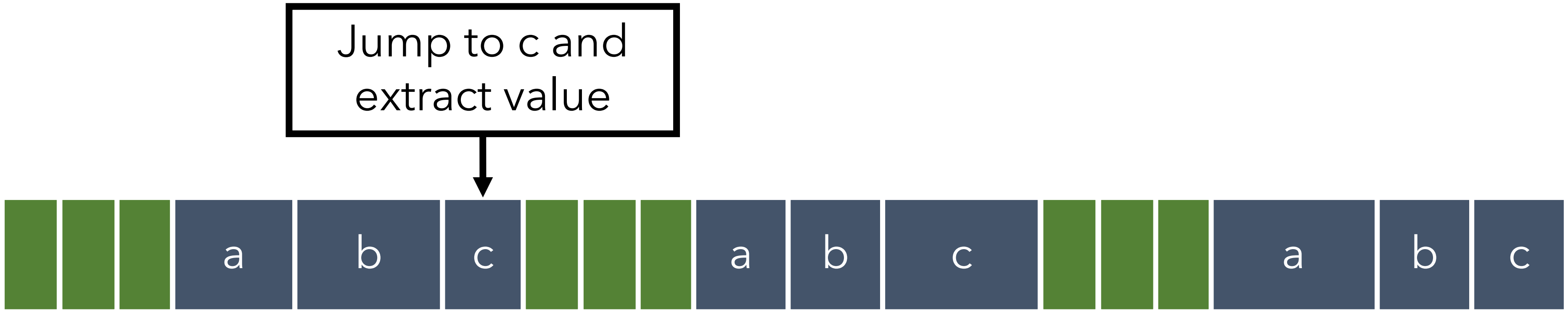
Streamlining value extraction (`Column` instruction)

```sql
SELECT c FROM table;
```

Future optimizations

Streamlining value extraction (`Column` instruction)

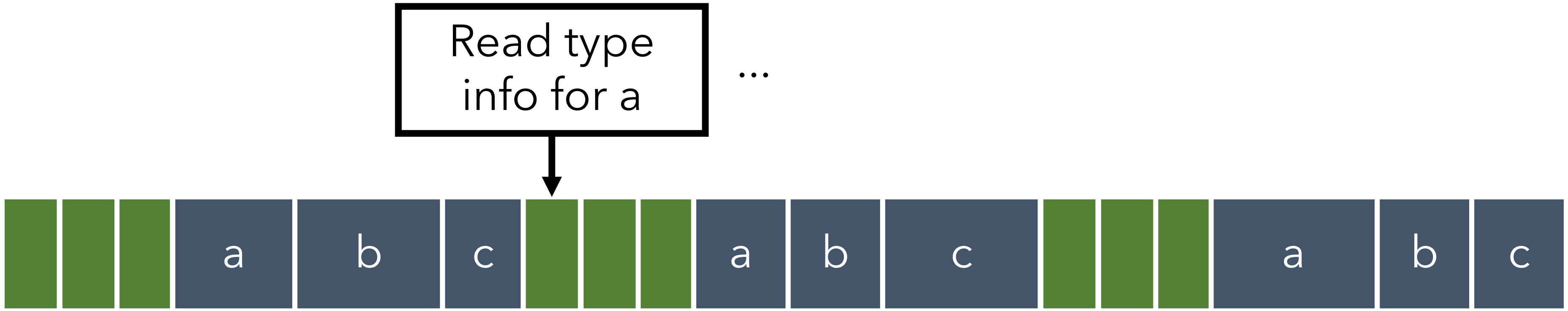
```sql
SELECT c FROM table;
```



Future optimizations

Streamlining value extraction (`Column` instruction)

```sql
SELECT c FROM table;
```



Future optimizations

Streamlining value extraction (`Column` instruction)

```sql
SELECT c FROM table;
```

Future optimizations

Intra-query parallelism

Modern mobile devices have a substantial amount of hardware parallelism.

A15 Bionic chip

New 6-core CPU with 2 performance and 4 efficiency cores

New 5-core GPU

New 16-core Neural Engine

Conclusion

SQLite offers high-performance embedded transaction processing in a compact, reliable, and portable library.

Although SQLite is over 20 years old, it is still rapidly developing.

Our optimizations (released in SQLite version 3.38.0) increase performance on SSB by 4.2X.

Future optimizations must balance performance gains with compactness and portability.

For more results and discussion, please see our VLDB 2022 paper.

Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy, and Jignesh M. Patel. SQLite: Past, Present, and Future. PVLDB, 15(12): 3535 - 3547, 2022.

Thank you!

This work was supported by Semiconductor Research Corporation (SRC) and the National Science Foundation (NSF).